NASA

Technical Memorandum 86111

DESIGN AND IMPLEMENTATION OF A COMPLIANT ROBOT WITH FORCE FEEDBACK AND STRATEGY PLANNING SOFTWARE

Timothy Premack, Franklin M. Strempek,
Leonard A. Solis, Steven S. Brodd,
Edwin P. Cutler, and Lloyd R. Purves

(NASA-TM-86111) DESIGN AND IMPLEMENTATION OF A COMPLIANT ROBOT WITH FORCE FEEDBACK AND STRATEGY PLANNING SOFTWARE (NASA) 99 p
HC A05/MF A01 CSCL 09B

N86-13947

Unclas
G3/63 04762

MAY 1984

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

TM 86111

DESIGN AND IMPLEMENTATION OF A COMPLIANT ROBOT WITH FORCE FEEDBACK AND STRATEGY PLANNING SOFTWARE

Timothy Premack, Franklin M. Strempek, Leonard A. Solis,
Steven S. Brodd, Edwin P. Cutler, and Lloyd R. Purves

MAY 1984

National Aeronautics and Space Administration

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland 20771

ABSTRACT

Force-feedback robotics techniques are being developed for automated precision assembly and servicing of NASA space flight equipment. Design and implementation of a prototype robot which provides compliance and monitors forces is in progress. Computer software to specify assembly steps and make force-feedback adjustments during assembly are coded and tested for three generically different precision mating problems. A model program demonstrates that a suitably autonomous robot can plan its own strategy.

ACKNOWLEDGEMENTS

The development of the Intelligent End Effector system has been a fairly major undertaking made possible by the contributions of a number of people. Mr. Lloyd Purves of the Goddard Space Flight Center developed the original concept and approach for this project, and he managed the overall development of the work. Mr. Timothy Premack, also of Goddard, made important contributions to the use of compliance in the system. Mr. Premack also designed the actuators and present compliant, force-feedback sensors. Mr. Frank Strempek of Goddard did all of the other mechanical design, including the original compliant force sensors and the compliant gripper. Mr. John Lallande and Mr. Woodrow Poland of the Goddard machine shop were very helpful in getting the equipment fabricated.

Mr. Premack did all of the control, system, and user software development work and used it to perform the first successful force-feedback insertion of pegs. Mr. Leonard Solis of Science Applications Research (SAR) then extended this software to carry out the insertion of bolts and electrical connectors. Mr. Steven Brodd of SAR developed the artificial intelligence algorithms to generate the robot command sequence for assembly from a definition of the assembled equipment. Mr. Ramesh Bhimarao of the University of Maryland wrote the software to generate the IGES database from a CAD database. Mr. Edwin Cutler of SAR managed the SAR software development effort and contributed significantly to the production of this report.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The research and development work described in this document has been undertaken as part of an effort to advance robotic techniques so as to be able to automatically and efficiently assemble or service NASA hardware either on ground or in orbit.

The basic impetus for this work is to achieve the efficiencies that robotic manipulators can offer over manual approaches. Given the added expenses, operational constraints, and safety requirements that affect the work man can do in space, the greatest potential for efficiently utilizing robots is in space. However, there is also significant labor required to prepare and refurbish launch system and payloads, and therefore there are also significant benefits to be gained from using robots on the ground to support such activities.

The particular activities pursued in the work reported on here are due to the special characteristics of NASA hardware, with respect to the goal of providing automatic robotic assembly and servicing. Some of the pertinent considerations of much NASA hardware are:

1. There are precision components with small clearances.

2. The hardware is highly complex.

3. The hardware is either a unique custom item, or has been produced in very small quantities.

The significance of point 1 is that small clearances essentially require the robot to have an advanced form of force feedback. This is because it is essentially impossible to use dead reckoning to position components having clearances on the order of .001 inches. Even if the robot can be guaranteed to have the required accuracy, the assembly into which components are being placed can be expected to be out of position by .001 inches due to thermal and load induced structural deformations or to the buildup of manufacturing tolerances. As is the case when a person performs assembly or servicing of precision hardware, the sensing and interpretation of interference forces is often more critical than vision feedback for compensating for the fine positioning errors. It seems that vision feedback is generally useful for positioning items to somewhere within about 0.1 to 0.5 inches and force feedback is used to correct for the remaining positional errors.

The other two considerations, namely the complexity and limited production volume of most NASA hardware means that there needs to be an efficient means of automatically generating the very large set of robot motions needed to assemble or service a piece of NASA hardware, such as a satellite. Based on past data of components per pound, a

satellite which represents an entire Shuttle payload, would have on the order of 1 million components. Recent changes in the ways hardware is designed, namely with the use of Computer Aided Design (CAD) equipment, offers a solution to this problem. By using CAD in the design of new space hardware, it is possible for a by product of the CAD design process to be a data base describing the geometric and other properties of the final product. With appropriate algorithms, many of which are being developed in artificial intelligence research programs, it will be possible for a computer program to analyze this data base and automatically generate from it the robot sequences needed to assemble or service the hardware represented by that CAD data base.

Therefore, the two activities pursued in this work have been the development of advanced robotic force feedback techniques and the automated generation of robot motions from geometric data bases. It should be noted that in certain instances these two activities merge, for the software which interprets the force feedback data can often require a very detailed knowledge of the interfering geometries and use artificial intelligence techniques to deduce what kind of positioning error, or other possible error source is causing the detected interference forces.

2.0 RESEARCH IN AUTOMATED ASSEMBLY

2.1 BACKGROUND

Research in automated assembly includes work in robot mechanisms and the software to control them. Work is now in progress on mechanisms from articulated hands (Salisbury and Craig 1982) to multi-legged robots (Klein and Briggs 1980; Orin 1976) and systems in which several robots work together (Ishida 1977). In support of robotics there is active research in end effectors (any of various devices located at the end of a robot arm or movable platform) (Frohlich 1979), vision systems (Brooks 1981), and tactile sensors (Harmon 1982; Hillis 1982).

In addition to work on robot mechanisms, research is in progress on the software to control them. Robot control software exists in a hierarchy of functional levels that ranges from the mechanism control level (Whitney 1976) to the level in which artificial intelligence can be used. At the lowest level (excluding the operating system that supports the robotics software) is the software that directly controls the mechanism itself: procedures that issue commands to the stepping motors (motors that translate rotational motion into very precisely controlled linear motion by "stepping" through many positions per rotation) and monitor analog-to-digital (A/D) converters. At the next level are procedures that compute forces, moments, and motor

speeds. Next is the software that effects primitive robot motions such as rotations and point-to-point motions, with numerous variations. Then comes the software that is responsible for executing robot tasks and, finally, the planning and strategy software that delineates the robot tasks (Fahlman 1973; Brooks 1983; Taylor 1976).

Research in automated assembly is important because of the potentially enormous benefits of its practical applications. For example, automated assembly could provide improvements in quality control, productivity, product cost, and employee health and safety. In addition to their applications in well-known areas such as the automobile industry and manufacturing facilities, the techniques developed in automated assembly research will find application in many other areas (Schratt 1980), especially those in environments in which it is either dangerous or economically infeasible for humans to work. Examples of such applications include nuclear power plant operation, toxic waste disposal, and space engineering.

There has also been research into the design of completely automated assembly systems (Ambler 1973; Lozano-Perez 1976) and software for describing and implementing assembly procedures (Popplestone, Ambler, and Bello 1980; Taylor 1976).

2.2 TOLERATING THE REAL WORLD

Assembly and repair in the real world have two motion domains: gross motion and precision motion. Gross motion moves a tool or part from a bin to the proximity of its final position. Gross motion, in this context, permits tolerances that are well within the accuracies of structured environments, that is, numerically specified engineering environments. The main problem to be solved during gross motion is to find an unobstructed path. Vision systems can assist in the determination of such paths, but in a structured environment, with its detailed knowledge of position and geometry, they are not necessary.

Precision motion to mate or match parts (such as cover plates and electrical connectors) or fasteners (such as bolts or screws) to a partially assembled mechanism requires the solution of a different problem. In practice there are always machining tolerances and tool and gripper sag due to gravity (or centrifugal forces in space). These effects combine to produce minor misalignment: bolts do not go into holes, electrical connectors resist mating, and cover plates do not seat properly.

Vision systems with limited resolution cannot reveal precision misalignments. In fact, the part, tool, or robot arm usually obscures the view. Despite this, work is being done to visually locate and identify partially hidden

objects (Bolles and Cain 1982; Tsuji and Nakamura 1975).

The solution to the problem of achieving precision motion is suggested by the machinist or mechanic who pushes, wiggles and loosens his grip until the part appears to mate itself; the mechanic uses force feedback and compliance as an adjunct to precision motion.

2.2.1 Force Feedback -

To assist in the development of technology useful to practical automated assembly systems, one focus of the research has been to develop a system that allows uncertainties in part placement and compensates for those uncertainties through the use of force feedback. To investigate the use of interpreted force feedback in assembly procedures, the system has been used in several insertion tasks which are described in detail in Section 4.0. These tasks provide an excellent vehicle for studying force feedback and its use in compensating for positional uncertainties since the tolerances involved in an insertion can be very small (.0005 inches is typical) and the misalignment that can be tolerated is correspondingly small. As an example of an insertion task, the problem of inserting a peg in a hole has been addressed by several investigators (Goto, Takeyasu, Inoyama 1980; Inoue 1974; Nevins, et al 1977).

2.2.2 Compliance -

Compliance is the capacity of a device to yield to forces, or be displaced, without suffering structural damage. For example, the sheet metal of the fender of a car has a relatively low compliance compared to a five mile-per-hour bumper that is designed to be displaced without being damaged.

When performing assembly tasks, humans often make use of varying levels of compliance. For example, when inserting a peg into a hole, a person can use gravity to help insert the peg. By relaxing the grip pressure, and thereby increasing compliance, the gravitational force exerted on the peg will center it in the hole. At first it may seem that to simulate such behavior it would be necessary only to have a device with high compliance since it is the ability to yield to the force of gravity that permits the centering motion to take place.

A compliant device may itself, however, be displaced by the same gravitational forces; the greater the compliance the greater the displacement. When the position of a compliant device has been changed, adjustments for gravitationally induced displacements can be made from a knowledge of the mass and stiffness of the device. When operating near zero gravity, such displacements are not a problem.

Compliance is not a common feature of present robot systems, but research has been done in this area (Paul and Shimano 1976; Nevins, et al 1977; Drake 1977, Klein and Briggs 1980)

3.0 THE INTELLIGENT END EFFECTOR (IEE)

3.1 INTRODUCTION

The Intelligent End Effector (IEE) consists of the robot hardware and software to control the tools necessary to perform assembly and servicing of NASA hardware.

Precision motion is an important element of these tasks and arises when the robot attempts to mate two parts, such as screwing in a bolt, inserting a peg, or fastening connectors, with clearances on the order of 0.0005 inches. Critical to the performance of precision motion is a recognition that the robot must compensate for positional uncertainties of the parts and of the robot itself. These uncertainties are on the order of half an inch in position and ten degrees in orientation. They are an accumulation of manufacturing tolerances, thermal expansion, part distortion, servo error, and general misalignment of fixtures.

To develop and test a system that would accomplish the goals chosen, a robot, a precision positioning system, a computer system, and several sets of associated software were deemed necessary.

3.2 HARDWARE DESCRIPTION

The hardware of the IEE consists of a robot, a VAX 11/780 computer, and a Computer Automated Measurement and Control (CAMAC) crate.

The robot itself is a three-force, three-moment compliant, force-feedback platform mechanism attached to a six-degree-of-freedom movable platform. Compliance in the platform mechanism relaxes the servo loops in the system, and prevents damage to the robot.

The mechanisms were designed and built at Goddard Space Flight Center, with the result that they easily interface to the VAX computer and have complete access to the control systems used.

The same VAX that controls the IEE is also being used to develop the CAD system that is an integral part of the project. It was chosen for both economic advantage and for the fact that its operating system is well-suited for software development. See Figure 1 for a block diagram of the system and Figure 2 for a photograph of the IEE.

3.2.1 IEE Support And Positioning Design -

The support and positioning device for the Intelligent End Effector is based on the design of an aircraft simulator mechanism (see Figure 3). A movable platform is supported above a stationary base by six axially extensible rods. Recirculating ballscrews provide the extensibility.

One of the goals in choosing components for this mechanism was to eliminate all possible backlash, for this reason, a solid preload Saginaw ballnut SSP-5700391 mounted on a Saginaw ball screw 1000-0200 was used. Each ballnut has 14 inches of travel along the screw. The ballscrew has a five-threads-per-inch pitch.

Stepper motors were chosen to drive the ballscrews. Using stepper motors eliminated the complexities introduced by servo loops. A new type of stepper and controller made by Compumotor, Incorporated was selected. Each of the six units required is comprised of an M83-135 stepper motor coupled with a 2100 series indexer. The motor is capable of 400 ounce-inches of static torque, 25,000 steps-per-revolution, 20 revolutions-per-second in angular velocity and 1000 revolutions-per-second squared in angular acceleration. Together with the ballscrew each actuator can exert 785 pounds of thrust. One step of the motor moves the ballnut eight micro-inches. These motors can execute various preset commands; the distance, velocity and acceleration are set in the controller before a move is

executed.

This positioning device is capable of moving the upper platform, and hence the IEE, within a one-cubic-foot envelope. Since the maximum translation is dependent on orientation and vice-versa, only nominal values from the equilibrium position can be given. The device can move at about 3.5 inches-per-second and has a load carrying capacity of approximately 2000 pounds.

The interface of the positioning platform to the computer was simplified by the design of the stepper motor controllers. Each controller has an RS-232C compatible I/O port. To control more than one motor, the controllers are serially daisy-chained, where each controller has a unique identification number. As an example, the ASCII string to set motor three to have an acceleration of 8 rpss, a velocity of 3.45 rps, and to move a distance of 3000 steps is: " 3A8 3V3.45 3D3000 ". As a result of this design, only one terminal port is required to control all six motors.

3.2.2 Compliant, Force-Feedback Design -

Compliance in the IEE is achieved through the use of a platform suspended from the active platform. The suspension mechanism consists of six spring-loaded pistons arranged in a geometry similar to the positioning actuators of the

movable platform. When resistive or gravitational forces are exerted on the IEE, the pistons are compressed or extended, providing compliance (Figure 4). This compliance is obtained by permitting strain on two opposing springs acting in the piston. A linear voltage differential transformer (LVDT) is used to measure the deflection of each spring. The force along each piston is obtained from the spring constant and the deflection measured by the LVDT. The forces and moments acting on the compliant platform are computed from the geometry and the forces along each piston, and force feedback is achieved. Forces measured when pressure displaces the spring-loaded compliant platform are relieved by adjusting the position of the movable platform to which the compliant platform is attached.

Each piston was designed to have about one inch of compliance and to be able to accept springs of various stiffness values. The current mechanism can support about 40 pounds dead weight and 25 foot-pounds of torque. In this configuration an accuracy of about 0.5 pounds is achieved.

A set of TRANS-TEK DC-DC gaging LVDTs was used. Since they work with a variable supply voltage, interfacing to the analog-to-digital converters was simplified. The supply voltage is provided by a KEPCO ATE 15-3M power supply, a very stable variable voltage power supply which is especially suited for this type of application.

The interface to the computer consists of an analog-to-digital converter driven by a CAMAC crate. The LVDT voltages are read by the A/D converter and this information is, in turn, read by the computer via the CAMAC crate. A Kinetic Systems 3514-A1A 16 channel A/D converter, capable of various input ranges, provides 12 bits of data.

3.3 SOFTWARE DESCRIPTION

The robotic software consists of a group of layered facilities for controlling the robot and accessing data about the robot and the forces it is sensing. The software has been developed as a set of self-contained modules, each one controlling some specific hardware task. There also exists a group of facilities which contain general library functions; some of these are robot independent, others are robot dependent.

The software is naturally partitioned into the main control software, which the user's program calls, and the force monitoring program which measures robot performance. The force monitoring program is run as a subprocess of the main program. This enables asynchronous monitoring of the force-feedback mechanism in real time.

Two typical functions of the software are to move the robot and to access the force-feedback data. These two functions are described to give an example of the flow through the software.

To move the robot, six data items which satisfy six degrees of freedom must be specified to target the new location and orientation of the movable platform in global space. In practice, motion is prescribed by providing the offset from the origin of the Cartesian coordinate system of the movable platform to the base platform and the three Eulerian angles which define the coordinate transformation tensor between the two systems. The six data items are passed to routine MTN_POSITION_TO. This routine performs an absolute translation and rotation of the movable platform to the given position. Figure 5 details the flow through the software to produce the motion.

Force-feedback data is accessed by FRC_GET_CONTACT_FORCES. This routine returns the three forces and three moments acting on the force-feedback mechanism. The units are in pounds and inch-pounds respectively. This routine must access the data passed to it by the subprocess. The flow is detailed in Figure 6.

The following is a list of the various user facilities. The routines are mainly written in FORTRAN, with a few hardware specific routines written in Macro. The code and inline documentation amounts to approximately 17000 lines of FORTRAN and 2000 lines of Macro.

1.	AD	Analog-to-Digital Facility
2.	AP	Active Platform Facility
3.	CMC	CAMAC Facility
4.	CP	Compliant Platform Facility
5.	FRC	Force Facility
6.	IEE	Intelligent End Effector Facility
7.	LIB	General Library Facility
8.	MTH	Mathematics Library Facility
9.	MTN	Motion Control Facility
10.	MTR	Compumotor Motor Control Facility
11.	OBJ	Object Facility
12.	SCRW	Screwing Control Facility
13.	SDF	Six-Degree-of-Freedom Facility
14.	SPC	Spatial Transformation Facility
15.	WRST	Wrist Facility

3.3.1 Analog-to-Digital Facility -

The analog-to-digital library contains routines which interface the analog-to-digital converter on the CAMAC crate with routines which need the data. The A/D converter is strobed by a subprocess running at a real-time priority which averages the data and passes it back to the main process via an installed section file. The subprocess is used to ensure that the readings are within a given band. Values outside this band indicate that the compliant, force-feedback mechanism has been displaced beyond a preset

limit. When this occurs the subprocess issues a halt to the stepper motors and thus prevents damage to the mechanism.

The facility consists of two basic modules, one which is called by the main process and the other which is called by the subprocess. The main process routine is an initialization routine which creates the subprocess. The subprocess is created with a termination mailbox. To ensure that the robot can't be run if the process is abnormally terminated a write attention asynchronous system trap (AST) is queued to this mailbox. The AST service routine executes when the subprocess terminates, writing the termination message to the screen and then stopping the main process.

The subprocess is in charge of scanning the A/D converter, averaging the data and if necessary stopping both the motors and itself if the readings are out of range. It runs at a real-time priority. Since the A/D converters can't update the readings as fast as the VAX can scan them, a timer is set after each scan. This also prevents the process from becoming compute-bound and degrading the system. The current scan time is 10 milliseconds, at which rate the subprocess uses only 3 to 5 percent of the CPU.

The mechanism used to pass the data from the subprocess to the main process is called a section file or shareable data file. It consists of a FORTRAN routine which is compiled, linked as shared and installed into the system as

writable using the VAX INSTALL utility. Both the main process and the subprocess are linked with this file. It maps the pages of this data area to the same physical pages in memory, which allows the data to be passed in a common memory area. This is the fastest way to pass data between two or more processes. There is no synchronization between the two processes, that is, no mutex to control the wait for read during a write. Even though the VAX provides this service with the lock manager, the service is not required since the A/D voltages will never change too much before the next scan. Furthermore, when the subprocess is writing out the data it is doing so at a priority of sixteen. Hence, there is little chance that it will be interrupted during its update.

ANALOG-TO-DIGITAL FACILITY

Routine	Function
ADMSG.MSG	Message file
AD___ASCEFC	Associates common event flag cluster.
AD_MAIN_INITIALIZE	Initialize the A/D facility.
AD_MAIN_RUNDOWN_AST	Executes when subprocess terminates.
AD_MAIN_STOP_SCANNER	Forces an exit of the subprocess (used in a termination handler.)
AD_READ_VOLTAGE	Routine to place data in common memory area.
AD_SHARE	Global section file executable (Passes data from sub to main

	process).
AD_SUB_INITIALIZE	Initializes the subprocess.
AD_SUB_SCAN_READINGS	Scans the A/D readings.
Error codes	
AD__OUTOFRANGE	This fatal error is signaled when one of the A/D values is out of range.
AD__SCANTERM	This fatal error is signaled when the scanning process is terminated by the main process.

3.3.2 Active Platform Facility -

The active platform facility controls the active platform at its lowest conceptual level. Routines are provided to start, stop, and position it in absolute coordinates. The active platform facility consists of the six ball screw stepper motor actuators and the two triangular aluminum weldments.

This software facility relies primarily on the SDF (Six-Degree-of-Freedom) and MTR (Motor) facilities to do the work. It keeps track of the position of the active platform and of the commands sent to the motors.

To execute a movement, a target position (x,y,z,roll,pitch,yaw) for the active platform is sent to the routine AP_SET_POSITION. This routine computes the length that each actuator will be when the new position is achieved. It then computes the necessary changes in the

lengths of the actuators and sends these changes of length to the associated motor controllers. The velocities of the motors are such that they terminate their moves at the same time.

The routine AP___FIND_VELOCITY is used to compute the velocity of each actuator. It uses the total travel time computed by AP___TRAVEL_TIME. The total travel time is computed by taking the distance the actuators are to be extended and the peak velocity of the actuators and then integrating over the velocity profile. The profile can have two shapes: an inverted "V" shape or a trapezoidal shape. The trapezoidal shape occurs when the distance is long enough for the stepper motor to reach its peak velocity. Given the travel time, AP___FIND_VELOCITY computes the actual velocity and sets each actuator so that it will travel the distance assigned to it in the time computed by AP___TRAVEL_TIME. (For equations, see Dieudonne, 1972).

ACTIVE PLATFORM FACILITY

Routine	Function
AP___MSG	Message file.
AP___FIND_VELOCITY	Computes the velocity of each actuator.
AP_GET_POSITION	Returns the position of the active platform (and places the data in the common memory area).
AP_INITIALIZE	Initializes the active platform.
AP___RESET	Resets the active platform data

	base when the motors are stopped before completing their preset commands.
AP___SET_ACTUATORS	Sets the motors for a move.
AP_SET_PEAK_VELOCITY	Defines the peak velocity at which an actuator can move.
AP_SET_POSITION	Set the movement of the platform.
AP_START_MOTION	Execute the set motion.
AP_STOP_MOTION	Stops the motion of the active platform.
AP___TRAVEL_TIME	Computes the time for a movement.
Error codes	
AP__OUTOFRANGE	Movement requested out of range of the active platform. Warning.
AP__ZEROMOVEMENT	No movement requested. Warning.

3.3.3 Computer Automated Measurement And Control Facility -

The Computer Automated Measur ment and Control Facility (CAMAC) provides the basic routines for accessing a CAMAC crate connected to a VAX. To access a foreign device on a VAX one can either write a full device driver or, if the device does not perform direct memory access, map the device into a virtual address space to reference the device registers. Since the crate controller used here does not perform direct memory access, the latter method was used.

The CAMAC crate controller is plugged into the UNIBUS. The physical address of the device is mapped into virtual address space using a system service call to $CRMPSC (Create and Map Section). It is called by CMC_MAP_CONTROLLER. This routine stores the virtual address of the crate for use by other routines which access the device registers.

CMC_INITIALIZE maps the device and then verifies that it is on line. To speed up the scanning of the A/D converter a special routine was written, CMC_READ_3514. This routine uses the auto index capability of the crate, thereby removing the need to set up special codes to access each channel.

Computer Automated Measurement and Control Facility

Routine	Function
CMC_READ_3514	Reads the channels of a Kinetic Systems 3514 analog-to-digital converter.
CMC_INITIALIZE	Initialize the CAMAC crate.
CMC_MAP_CONTROLLER	Map the CAMAC controller into our virtual address space.
CMC_WRITE_DATA	Write data to the CAMAC crate.

3.3.4 Compliant Platform Facility -

The compliant platform is the name used for the three-force, three-moment compliant, force-feedback mechanism. It consists of six passive pistons mounted between two plates. From the known spring constants and the measured tension or compression displacements associated with the pistons, the force on each piston and thus the forces and moments on the moving platform are computed.

CP___GET_LNGFRC is responsible for converting the output voltage of each linear voltage differential transformer (LVDT) to the actual length for each piston and the resistive force generated by it. The values are averaged if the robot is stationary; otherwise the readings are taken instantaneously. This routine accesses the voltage of each LVDT with the routine AD_READ_VOLTAGE. CP___UPDATE_POSITION uses the lengths of each piston to compute the position (x,y,z,roll,pitch,yaw) of the compliant platform relative to the compliant base. It uses the SDF library routines to do this. CP_GET_FORCES is used to compute the forces on the compliant base. This routine performs an equilibrium analysis on each piston. After finding the reaction forces at the pins where the pistons are attached to the compliant plate, it sums the forces and moments about the origin of the plate.

CP_INITIALIZE is used to initialize the constant data concerning the compliant platform.

COMPLIANT PLATFORM FACILITY

Routine	Function
CP_GET_FORCES	Computes forces and moments acting on the platform.
CP___GET_LNGFRC	Computes force and length of each piston.
CP_INITIALIZE	Initializes the compliant platform.
CP___UPDATE_POSITION	Updates the position of the platform.

3.3.5 Force Facility -

The force facility contains the basic routines for monitoring the forces on the compliant platform. They perform the tasks of obtaining the contact forces on the compliant platform and monitoring these forces while the robot is moving. The robot is stopped if any force has exceeded a prescribed limit. This is the logic for the move-until routines.

FRC_GET_CONTACT_FORCES computes the contact forces on the compliant platform. The reaction force supporting the platform is computed by CP_GET_FORCES. The gravity forces acting on the platform are then subtracted, yielding the contact forces. The gravity forces consist of the weight of the platform along with the weight of the object that is

attached to it.

FRC_MONITORING_WHILE_MOVING is the routine used to check the forces while the robot is moving. When the robot starts a move a timer is set. When the timer runs out an event flag is set to stop the move. This routine monitors the forces while the move is in progress; if the force is out of range it calls two routines: AP_STOP_MOTION and MTN___CLEANUP_POSITION. While the forces are within range it continues this loop until the event flag CEF_MTN_TIMER has been set by the timer. Once the event flag has been set, the robot has stopped moving and the routine exits.

FRC_MONITORING_WHILE_MOVING_1 is basically the same routine except that the routine which monitors the forces (FRC_CHECK_FORCES) is passed to the routine as an argument. LIB$CALLG is used to call FRC_CHECK_FORCES with its argument list. This routine provides the basic facility for monitoring while moving. It eliminates proliferation of monitoring routines.

FORCE FACILITY

Routine	Function
FRC_CHECK_FORCES	Checks if forces are within a band.
FRC_GET_CONTACT_FORCES	Computes the contact forces on the compliant platform (subtracts forces due to gravity).
FRC_GET_FORCES_AB	Contact forces on the compliant platform in the active base space.

FRC_MONITOR_WHILE_MOVING	Move until forces exceeded.
FRC_MONITOR_WHILE_MOVING_1	Move until user-supplied routine returns false.

Error codes

MAXEXCEEDED	Force exceeded the maximum range.
MINEXCEEDED	Force exceeded the minimum range.
INBAND	Force is within range.
USERTRUE	User routine returned a true.
USERFALSE	User routine returned a false.

3.3.6 Intelligent End Effector (IEE) Facility -

The IEE facility performs all initialization necessary to bring the robot on-line. IEE_INITIALIZE must be called before any of the robotic software can be used. It, in turn, calls all other initialization routines. If any facility needs to be initialized before use, it is called here.

Since many of the routines read data files, IEE_INITIALIZE_DATA provides a FORTRAN logical unit number through which to perform the I/O. To speed the initialization process and since many files need to be read, one common file is created by all the initialization routines. It is an unformatted file which if present is referenced by the logical name IEE_DATA. If not present, it is created with data from the original ASCII data files.

All exit handlers are established through IEE_HANDLER. Since their execution is last-in, first-out the sequence of establishment is critical.

IEE_INITIALIZE_MOTORS initializes the stepper motors. It assigns a channel to the motor controllers through the terminal port device IEE_PORT0.

IEE FACILITY

Routine	Function
IEE_INITIALIZE_DATA	Calls the initialization routines of all facilities.
IEE_HANDLER	Declares all exit handlers.
IEE_INITIALIZE	Initializes the Intelligent End Effector.
IEE_INITIALIZE_MOTORS	Initializes the motor I/O channel.

3.3.7 General Library Facility -

This library facility is a collection of general utility routines which are robot-independent. They can be used without the robot software. A brief description of each routine follows.

LIB_ARGNUM, when called, returns the number of arguments with which the subroutine which called it was called. This is useful if the subroutine function depends on the number of arguments, and is useful in some FORTRAN routines since FORTRAN cannot access the call stack.

LIB_FILL_VECTOR fills a vector (REAL*4) of length N with a scalar.

LIB_MAKE_ARGLIST creates an argument data structure. This is used to establish the argument list for the routines which require it, such as FRC_MONITORING_WHILE_MOVING_1.

LIB_PACK_VECTOR packs a vector (REAL*4) of length N with scalars S1,S2,...SN.

LIB_TRACE enables or disables the function of LIB_SIGNAL.

LIB_SIGNAL signals a condition if enabled. It is the same as LIB$SIGNAL except that the condition is passed by reference and LIB_TRACE turns off the signalling mechanism. This is useful in debugging programs.

LIB_WAIT waits N (where N is a real number) seconds and then returns. This routine reduces the proliferation of event flags throughout the program.

GENERAL LIBRARY FACILITY

Routine	Function
LIB_ARGNUM	Number of arguments with which routine was called.
LIB_FILL_VECTOR	Vi = scalar; i = 1,2,...,N.
LIB_MAKE_ARGLIST	Creates an argument list.
LIB_PACK_VECTOR	[V] = [scalar1 , scalar2 ,..., scalarN].

LIB_TRACE	Enables or disables the function of LIB_SIGNAL.
LIB_SIGNAL	Signals the condition if enabled.
LIB_WAIT	Waits N seconds.

3.3.8 Mathematics Library Facility -

The mathematics facility contains utility routines for performing mundane mathematical functions. A brief statement is given which describes each routine.

MATHEMATICS FACILITY

Routine	Function
MTH_ADD_VECTOR	Add two vectors.
MTH_CROSS	Calculate cross product.
MTH_DOT	Calculate dot product.
MTH_LNG	Magnitude of a vector.
MTH_MOVE_VECTOR	Move a vector.
MTH_MUL_MATRIX	Multiply two matrices.
MTH_MUL_VECTOR	Vector times a scalar.
MTH_NEG_VECTOR	Negate a vector.
MTH_PLANE_NRM	Calculate the normal to a plane.
MTH_ROTATE_VECTOR	Rotate a vector.
MTH_SUB_VECTOR	Subtract a vector.
MTH_TRANSPOSE_MATRIX	Transpose a matrix.
MTH_TRANS_MATRIX	Calculate a rotation matrix from three angles.
MTH_TRANS_TO_EULERS	Inverse of MTH_TRANSPOSE_MATRIX.

MTH_UNIT_VECTOR	Normalize a vector.
MTH_ZERO_VECTOR	Zero a vector.
Error codes	
ZEROLNGVECTOR	Vector has no magnitude.
COLLINEAR	Three points are collinear.
SINGULARMAT	Matrix is singular.

3.3.9 Motion Control Facility -

The motion library consists of the top-level movement routines. These are the routines which a user calls from his program. The basic motions are translation, rotation, and curvilinear motion. Rotations can be performed about any point. Motions can be specified in absolute or relative coordinates. A position has three coordinates (x,y,z) and three orientation angles (roll,pitch,yaw). A motion is specified by three position displacements, in inches, and a rotation of three angles, in radians.

The UNTIL routines provide for motion while monitoring forces. A force in a given direction is monitored while the robot is moving. If the force is increased beyond a prescribed envelope the robot is stopped. To provide a general move-until logic the routine MTN_MOVE_REL_UNTIL_1 is used. Here the user specifies his own check routine.

Since other routines may need to determine if the robot is moving or not, a state routine is provided. MTN_STATE returns the state of the motion.

The SET_STOP and SET_WAIT routines toggle flags which control the flow of the lower-level positioning routines. During a motion, after the motors have been sent the go command, the program waits for the motion to be completed by setting a timer. The timer in turn sets the event flag CEF_MTN_TIMER. If the wait mode is clear the routine MTN___SET_POSITION doesn't wait for this event flag but returns.

The SET_STOP routine controls the scanning process. If the flag is set and any of the A/D converter readings are out of a specified range, the robot will be brought to a controlled stop. The flag AD_STATE_STOPPED is then set to acknowledge this fact. This function was included because it is the fastest way to perform a move-until-touching. It provides a recoverable method for stopping the robot, as opposed to a failure caused by the scanning process with an AD_OUTOFRANGE error.

MOTION FACILITY

Routine	Function
MTN_MOVE_REL	Move the active platform (AP) a relative distance.
MTN_MOVE_REL_POLAR_AND_SCREW	Move the AP a relative

	distance in a direction and turn the bolt spinner.
MTN_MOVE_REL_POLAR_UNTIL	Similar to MTN_MOVE_REL_UNTIL except with polar move.
MTN_MOVE_REL_UNTIL	Move the AP a relative distance until a force is out of range or motion is completed.
MTN_MOVE_REL_UNTIL_1	Like MOVE_REL_UNIT except with user check routine.
MTN_MOVE_TO	Move the AP an absolute distance.
MTN_POSITION_REL	Position the AP a relative distance and rotate.
MTN_POSITION_TO	Position the AP an absolute distance and rotate.
MTN_ROTATE_REL	Rotate the AP a relative amount.
MTN_ROTATE_REL_ABOUT_CP_INAB	Rotates the AP a relative amount about a point in the compliant platform.
MTN_ROTATE_REL_ABOUT_INAB	Rotates the AP a relative amount about a point in active base space.
MTN_ROTATE_TO	Rotate the AP an absolute amount.
MTN_SET_STOP	Enable or disable the stopping logic.
MTN_SET_WAIT	Enable or disable the wait logic.
MTN_STATE	Returns the state of the machine (move or nomove).

3.3.10 Compumotor Motor Control Facility -

The motor control facility is used to control the Compumotor Series 2100 stepper motor controllers and motors. The motor controllers are microprocessors which control the

stepper motor power supplies. The processors accept ASCII command strings from a host computer over RS-232C terminal ports. There are numerous command strings; the ones used here control the distance to travel, the peak velocity during the motion, and the acceleration. To facilitate multiple controllers, they may be serially daisy-chained together: the echo of the command from one controller is fed into the input of the next controller. A device number precedes the command string if it is to be applied to only one controller. For example, to set the distance, velocity and acceleration of all of the motors in the same string the command would look like : " D25000 V3.23 A8.12 " where D25000 is a distance of 25000 steps, V3.23 is a velocity of 3.23 revolutions-per-second, and A8.12 is an acceleration of 8.12 revolutions-per-second squared. If one wanted only controller number three to have these characteristics the string would be: " 3D25000 3V3.23 3A8.12 ".

This library contains the necessary routines to send and receive the command strings. Also included is a routine which computes the time of motion given the distance, velocity and acceleration.

COMPUMOTOR MOTOR CONTROL FACILITY

Routine	Function
MTR_INITIALIZE	Initialize the motors.
MTR_MOTION_TIME	Computes the time and velocity for

	a move.
MTR_READ_COMMAND	Send and read command.
MTR_READ_POSITION	Read position of a motor.
MTR_REVS_TO_STEPS	Convert revolutions to steps.
MTR_SEND_COMMAND	Send commands.
MTR_SET_ACCELERATION	Set acceleration.
MTR_SET_DISTANCE	Set distance (steps).
MTR_SET_VELOCITY	Set peak velocity.
MTR_SEND_GO	Start motor(s).
MTR_STEPS_TO_REVS	Convert steps to revolutions.
Error codes	
DATACHECK	String echoed from controller not the same as the one sent.
INTERNALWRITE	Internal write error.
IDOUTOFRANGE	Controller ID number out of range.
ACCOUTOFRANGE	Acceleration out of range.
VELOUTOFRANGE	Velocity out of range.
DISOUTOFRANGE	Distance out of range.

3.3.11 Object Facility -

This facility is used to define data for objects carried by the IEE. The object, such as a gripper, a bolt spinner, or a peg, is attached to the compliant platform. A data file for the object is pointed to by the logical name IEE_OBJECT, and specifies the weight, center of gravity, and position of the object relative to the compliant platform.

This facility will be removed when the planned gripper is integrated into the system. At that time there will be a routine to reinitialize the data depending on the gripper's task.

OBJECT FACILITY

Routine	Function
OBJ_INITIALIZE	Initializes the object data.

3.3.12 Screwing Control Facility -

This facility controls a separate stepper motor which is used as a bolt spinner. SCRW_ROTATE rotates the motor a given number of revolutions. The SET_MOVE command is used to set up a movement; the motor is activated by the general motion commands.

Tnis library is under development. Since the bolt spinner motor torque is inadequate to perform the desired tasks, other methods are being studied.

SCREWING CONTROL FACILITY

Routine	Function
SCRW_ROTATE	Turns the bolt spinner.
SCRW_SET_MOVE	Sets up a movement for the spinner.

3.3.13 Six-Degree-of-Freedom Facility -

The Intelligent End Effector contains two six-degree-of-freedom mechanisms: the positioning platform and the force-feedback mechanism.

The routines for the mechanisms compute the position of the movable platform from the lengths of the six rods, and the inverse. They are called SDF_GET_LENGTHS and SDF_GET_POSITION. The algorithm uses six vector loop equations and Newton's method of solving simultaneous equations to solve for the location and orientation of the specific platform. (Dieudonne, et al, 1972).

SIX-DEGREE-OF-FREEDOM FACILITY

Routine	Function
SDF_INITIALIZE	Initializes the fixed data for the mechanism, and fills the position and transformation matrices.
SDF_GET_LENGTHS	Gets length vectors given position of movable platform.
SDF_GET_POSITION	Gets position of movable platform given lengths of extensible members.

3.3.14 Spatial Transformation Facility -

The robot contains five reference frames: active base (AB), active platform (AP) , compliant base (CB) , compliant platform (CP), and object (OBJ). The world or global space

is equivalent to AB space. To convert vectors from one space to another the set of routines called SPC_RF_nn_mm is used, where "nn" and "mm" are each one of the above reference frames. These routines multiply a given vector with a second order tensor to produce the desired conversion. SPC_TRN_nn_mm is used to compute the coordinates, in "nn" space, of a point in "mm" space.

SPATIAL TRANSFORMATION FACILITY

Routine	Function
SPC_RF_nn_mm	Converts a vector from reference frame "nn" to "mm".
SPC_TRN_nn_mm	Converts a point in space from Euclidean space "nn" to "mm".
	nn, mm = AB , AP , CB , CP , OBJ

3.3.15 Wrist Facility -

At present there is no real wrist mechanism on the robot. If there were, it would be located at the active platform and compliant base interface. This facility was created to provide the structure if such a mechanism is installed. Presently there is a tensor which converts the space from AP to CB space with constant data initialized at startup time. If a mechanism is put at this interface in the future then it will dynamically alter this data.

WRIST FACILITY

Routine	Function
WRST_INITIALIZE	Initializes the wrist data.

4.0 THE INTELLIGENT END EFFECTOR IN USE

Precision assembly was demonstrated for three different assembly configurations. For each test problem the compliant, force-feedback characteristics of the IEE were used as an adjunct to precision motion. A round peg was inserted into a round hole, a 25 pin D-type connector was mated, and a bolt was screwed into a threaded hole.

The three cases are described in the following sections. The use of compliance and force feedback and the achieved precision are explained in detail.

4.1 PEG INSERTION

The insertion of a round peg into a round hole was chosen as the first test case. The problem required that the robot insert a standard 0.375 inch dowel pin into an 0.3755 inch hole drilled normal to the surface of a plane.

The engineering data, which provided a structured environment, consisted of the diameter of the peg, the location of the peg in the robot's space, the depth of insertion, the location of the hole in global space, and the orientation of the hole in global space. The location and orientation of the hole were accurate to 0.5 inches and five degrees respectively.

The algorithm to insert the peg into the hole consists of six parts:

1. Find the plane of the hole by tilting the peg and touching three points around the hole with the edge of the peg.
2. Find the hole by dragging the edge of the peg along the plane until it protrudes slightly into the hole.
3. Center the peg in the hole by moving it back and forth perpendicular to the previous direction of motion until the sides of the hole are encountered.
4. Continue in the original direction of motion, but now along the centerline of the hole, until the far edge is contacted.
5. Reorient the peg until it is normal to the plane, keeping the end of the peg in the hole during the reorientation process.
6. Insert the peg into the hole while nulling out forces and moments.

A detailed description of how the software performs each of these six actions follows:

1. The plane-finding algorithm requires three points on the plane to determine its equation. The robot uses the peg to probe the surface of the plane, tilted at an angle such that point contact is made between the surface of

the plane and the peg. The coordinates of the touch point can be determined because the position and orientation of the circle which is the end of the peg are known.

The algorithm begins by tilting the peg to obtain the best touch geometry. The given data for the plane is used in this case. This is done by orienting the peg so that the axial vector of the peg is parallel to the gradient of the plane. This ensures (within a known error) that the lowest point on the end of peg will be the touch point.

The algorithm then determines three eligible touch points on the plane. For this case the three points chosen lie on a circle concentric with the hole with a radius of the given hole radius plus 1.5 times the assumed positional error of the hole. This ensures that the peg will not fall into the hole prematurely. The points are equally spaced around the circle. The robot then moves the touch point on the peg above each chosen point on the plane and then moves down toward the plane, monitoring the forces as it moves. When it encounters a change in the force it stops and computes the position of the peg, and hence the position of the plane.

2. In the next step, the robot must find the hole. Keeping the peg tilted as in step 1, the robot touches the peg to the plane below the hole and then drags the peg up the plane. The idea is that since the motion is parallel to the plane the force on the peg normal to the plane will remain relatively constant until the end of the peg protrudes into the hole. When this happens the robot has found the hole.

3. To center the peg in the hole the robot moves perpendicular to the previous direction, still parallel to the plane, until it encounters one edge of the hole. It then moves in the opposite direction to find the other edge. This line segment is a chord of the circle defined by the top of the hole. The perpendicular bisector of this chord is a line along the diameter of the top of the hole.

4. The robot moves the peg along this line in the same direction as the drag move in step 2 until the far side of the hole is encountered. When this happens the robot stops and checks the side forces to center the peg in the hole again.

5. Next, the robot orients the peg until its axis is parallel to the axis of the hole. The orientation of the hole is known from step 1. The robot performs this step iteratively, one degree at a time. After each step

it checks to determine that the end of the peg is still below the surface of the plane. It also checks the contact force of the peg against the hole. The robot always maintains pressure between them. After each iteration, the robot compares the angle between the peg and the hole. When this is within a satisfactory tolerance the robot moves to step 6.

6. Finally, the robot begins to insert the peg into the hole. As the peg is inserted two forces and one moment are monitored. The force normal to the axis of the hole is checked. This force is generated if the peg is pushing against one side of the hole. The force parallel to the axis of the hole is also checked. This force indicates that the peg is jammed. If the peg is cocked in the hole a moment will be generated. Since the peg is round, the robot does not monitor moments about the normal of the hole. When any of the forces or moments are out of range the robot stops and takes corrective action. For the force in the plane the robot moves in the direction of the force until it disappears. If the peg is cocked, the robot rotates until it eliminates the moment. If the peg is jammed, the robot withdraws the peg until the force disappears and performs a wiggle motion and then tries again. This force-feedback insertion continues until the peg has been inserted the required distance.

The peg routine takes approximately three minutes to run. Although the process is not fast enough for industrial applications, the basic steps have been worked out. This solution to the peg-in-a-hole problem demonstrates that a force-feedback robot is capable of mating two precision parts.

4.2 MATING A 25 PIN D-TYPE CONNECTOR

The D-type connector is an excellent example of a multisided component that has few symmetry properties. A 25 pin connector was used in this experiment. It consists of a male half that contains 25 pins and a female half with 25 corresponding sockets. To mate the connector components, the two halves must be properly aligned with the male half partially inserted into the female half. Such a partial insertion is possible since there is a gap of approximately 0.05 inches from the ends of the pins in the male half to the edge of the surrounding lips in the female half. The pins in the male can therefore be inserted into the sockets on the female slightly more than this distance before a resistant force must be overcome. When the halves are properly aligned and the direction of insertion is correct, the force that is required to fully mate the male and female halves is approximately 4.5 pounds.

4.2.1 Assumptions -

It is assumed that the male half is held in a fixture and that the female half is held by the IFE. In addition, the following were the maximum allowable errors in assumed position and orientation of the female:

- o eight degrees about any axis of rotation
- o one-half inch in X, Y, or Z axes

It is assumed that the free space volume in which the IEE is able to move the female half of the connector is a hemisphere of radius three inches, centered at the fixed male half of the connector.

4.2.2 Database -

Implicit within the program is an understanding of the geometry of the connector components. Therefore, the supporting database contains only the values of those parameters which quantify the geometry (length, width, height, short-side length, distance between pins, etc.). In addition, the program is given the assumed position of a single point on the object that is considered to be the origin, and the orientation of the object about that point in terms of the three Euler angles, specifying a rotational displacement about the X, Y, and Z axes.

4.2.3 Algorithm To Perform Mating -

The connector mating program uses the following strategy. First, it assumes that the given location of the connector is correct and attempts to perform the insertion immediately. It does this in two steps. In the first step, the IEE attempts to position the female partway into the male by moving down until it touches the object. If the male is located precisely at its assumed position, this will place the female just inside the lip of the male, with the pins not yet inserted. To determine whether or not the male is actually where it is supposed to be and whether or not the female is inside the lip of the male the IEE moves from side to side and determines the displacement of the female that occurs at the extreme ranges of this motion. If the resulting displacement of the female half is significantly less than the displacement of the movable platform to which the compliant platform is attached, it is assumed that the movement of the female half was constrained because it was partially seated inside the male half. If, indeed, the female half is partially seated, the insertion proceeds to completion. If not, the end effector proceeds with the following different mating strategy.

If the immediate insertion attempt fails, the end effector attempts to determine the orientation of the fixed male half of the connector by touching it at various points.

After it does so, the IEE then aligns the female with the male and touches the object again to accurately locate the side. (This must be done because, due to the rounded edges of the connector, the exact point that is being touched cannot be determined. This is not a problem in determining orientation, since only relative positions are of concern. The position of the connector must, however, be accurately known to perform the next step.) The IEE then positions the female directly above the center of the male and tilts the female so that one end can be used to probe into the male connector. It then moves down into the connector, and slides in the direction of the tilt until it finds the end of the male connector. In doing so, the IEE makes allowances for the possibility that it may get stuck on a pin in the male, mistakenly believing it has reached the end of the connector. Once it believes that it has found the end of the connector, the IEE then removes the tilt by pushing against the touched end of the connector and rotating about that point. This allows the female half to remain inside the connector at all times and improves the reliabilty of the operation. Once the halves are aligned the insertion operation is continued. As the insertion is being performed, the IEE monitors forces and torques and attempts to keep all forces except for the insertion force as low as possible. This helps to eliminate any remaining error in the alignment of the two connector halves.

4.2.4 Results -

The robot has demonstrated that it can mate the connector from initial starting positions that vary from backward to upside down. The solution to this problem demonstrates that force feedback with compliance can be used to mate nonsymmetrical connectors which require precision motion to avoid damaging functional parts.

4.3 SCREWING A BOLT INTO A THREADED HOLE

The operation of screwing a bolt into a threaded hole introduces several new problems for the IEE. Although it is similar to the task of putting a peg in a hole, there are important differences. Some of these are:

1. the bolt screwing task makes use of a tool to spin the bolt (the bolt spinner);
2. both the bolt and the hole have threads;
3. the bolt is not rigidly held by the bolt spinner; and
4. the bolt cannot simply be inserted into the hole but must be screwed in, meaning that the operation consists of two concurrent parts: turning the bolt in the correct direction and inserting the bolt into the hole at the correct speed.

4.3.1 Problem Description -

This experiment used a 0.375 UNC x 1.500 inches long socket head cap screw and a corresponding hole in an aluminum block with a steel 3/8" helicoil insert. The device used to spin the screw was a stepping motor (of the same type used to drive the IEE extensible rods) attached to the compliant platform of the IEE. Attached to the shaft of the motor was a device used to hold the screw. This device is shown in Figure 7. The cap bolt was held in place by three ball-detents. The bolt was seated in the holder by pressing lightly against the bolt head and rotating the holder until the hex-head driver in the holder was aligned with the hex head of the bolt. When that condition was met, the IEE then pressed the holder against the bolt head until the head was firmly seated into the holder.

4.3.2 Assumptions -

It was assumed that the block containing the hole was held rigidly in place and that the following were the maximum allowable errors in the assumed position and orientation of the hole:

- o 3/4 of one bolt radius in any direction

o 8 degrees about any axis of rotation

It was also assumed that there were no obstacles (other than the block itself) to impede the motion of the end effector.

4.3.3 Database -

As with the 25 pin connector mating program, implicit within the bolt screwing program is an understanding of the geometries of the components involved in the assembly task. Information in the database which is available to the program includes measurements that completely describe the bolt (pitch, size, length, drive type and size, and bolt type), and the assumed position and orientation of the hole.

4.3.4 Program To Perform Bolt Insertion -

As with the peg-in-hole program, it is assumed that the IEE must verify or refine the position and orientation information given about the location of the hole. Therefore, the IEE first attempts to determine the orientation of the block that contains the bolt hole. It does so by touching the surface of the block at three points and determining the equation that describes the plane that contains those three points. With this information, a point is found on the block that is two bolt-hole radii from the center of the assumed hole position and that provides a path

of steepest ascent to the hole. The bolt is then tilted in the direction of travel (towards the hole). This is done for two reasons. First, it provides a smaller surface area with which to touch the block. Second, it allows gravity and the compliance of the IEE to help center the bolt in the hole once it has been found. The IEE then gently presses the end of the bolt against the block and slides the bolt in the direction of the hole until the force exerted against the block diminishes, indicating that the IEE has found the hole. Once it has found the hole, a series of wiggling maneuvers are performed to center the bolt in the hole.

When the end of the bolt is in the top of the hole, it must be aligned with the hole's axis so that it can be screwed in place. The procedure to do this is very similar to the method used to orient the peg and the 25 pin connector in previous tasks. The bolt is pushed down and against one side of the hole and is reoriented toward the assumed alignment position while keeping the end of the bolt in the hole. This reorientation is performed in small angular increments to allow for adjustments, ensuring that the bolt remains in the hole while the alignment takes place. At the end of each incremental orientation motion the bolt is wiggled to help it seat itself and then is again pushed down into the hole and against one side.

Once the bolt's axis is aligned with the hole's axis, the screwing procedure is begun. First the bolt is pressed into the hole. Then while moving downward the bolt is slowly turned to allow it to thread itself partway into the hole. As screwing proceeds, the IEE nulls out any forces acting on it due to any remaining misalignment between the bolt and the hole. That is, if the block were removed the bolt would remain in the same position. The IEE continues screwing until the torque required to spin the bolt reaches the desired value, indicating that the bolt has been fully inserted and tightened to the desired torque.

4.3.5 Compliance -

The compliance of the IEE was used to advantage in this task by allowing gravity and forces generated due to misalignment to center the bolt in its hole. Without compliance, these forces would not affect the position of the bolt unless they caused some deformation in the mechanism or they exceeded the forces produced by the stepping motors that keep the IEE in a given position.

5.0 A MODEL FOR ASSEMBLY AND REPAIR STRATEGY

5.1 INTRODUCTION

The section most closely associated with the techniques of knowledge engineering has been given the acronym ASP for Automated Sequence Planner. This part of the project is responsible for determining a sequence of robot moves to effect construction from information available in a CAD database.

The basic strategy used is that of the reverse heuristic search (a search through a tree of possibilities that treats what would ordinarily be the goal as the starting node and what would ordinarily be the start as the goal, and that uses heuristics to limit the search). The ASP first synthesizes a disassembly of the indicated object, and then reverses that sequence to derive an assembly sequence. For those applications involving repair, both disassembly and assembly sequences, partial or complete, would be utilized.

The ASP is general in nature. Specific information about the format of the CAD database and particular robot or other assembly devices used is imparted in the form of databases.

To demonstrate the most important features of the ASP, and to determine the most difficult aspects of its implementation, a model program has been written in the LISP language. It has been successfully used to provide a construction sequence for a number of simple objects consisting of blocks held together with bolts.

5.2 DESCRIPTION

The following sections describe particular details of the model, as well as describing the generalized goals, where determined, for each part of the problem in the ASP program.

5.2.1 Input And Output -

Input to the ASP will occur via program generated calls to FORTRAN subroutines that will access and manipulate a CAD IGES (Initial Graphics Exchange Specification) format database, (a specification for geometric databases that has achieved considerable attention as a uniform, transportable system) and in some cases do considerable computation on acquired data. The goal will be input at a fairly abstract and database-independent level. In the model, the database queries are simulated by English requests for database information from an interactive user.

Output from the ASP will consist of a file of robot commands, giving all information necessary for construction, or in the case of repair, disassembly, part replacement, and reassembly. This file will in essence be a robot command language, to be interpreted by software associated with the robot. The model program writes such a file, with the production of certain information, such as tool placement, not yet implemented. This model file is interpretable by human or robot.

5.2.2 Demonstration Limitations -

The ASP will be made as general as it is feasible to do, so that it will be able to operate on complex parts and assemblies, including multi-path part trajectories, curved parts, different screw pitches, and so on. Particular limitations imposed by the CAD database or by the robot will be realized from their database descriptions. For example, in the case of the IEE, only those objects that can be assembled with one hand are viable candidates.

The model program has been necessarily limited in the scope of objects on which it can operate. Specifically, any object under consideration by the program is assumed to consist of a base to which other parts are attached. The base is held in a vise table that can rotate about three axes. Each part is attached either to the base or to

another part with one or more threaded bolts. It is assumed that each part may be eventually removed from the assembly in a trajectory that is a straight line along one of the six directions defined by the three coordinate axes: that is, no curved or multi-path trajectories are permitted. The fastening bolts are removed in the same way. Finally, each part must be specified by straight edges, flat planes, and right angles.

A number of assumptions are also made about the capabilities of the robot in the model program. Only assemblies that can be constructed with one hand plus the moving vise table are allowed, and only one face of the object can be approached at a time. The movement envelope of the IEE is respected. No calculations are made as to tool placement or the complications that tool positioning makes to the trajectory determination. In addition, all parts in the model are removable by a simple gripper or by a bolt spinner with one size of bolt head.

5.2.3 CAD Database -

The CAD database (in this case, in the IGES format), stores a description of the object in its assembled form, as well as information about each of the constituent parts in the object. This information, as well as information about the particular robot and tools that are available, is used

by the ASP in determination of the construction sequence. The model program assumes a particular database format, as follows: the assembly as a whole and each separate part are located in three-space coordinate systems with associated dimension and bolt attachment information; two triplets of numbers describe a part's location in the assembly; the first triplet gives the location of the origin in the part's reference frame in assembly coordinates; the second gives the angular rotation around the three coordinate axes (the Eulerian angles) to transform the part's original orientation to its orientation in the assembly. Thus when the model asks for a part's location and orientation, it expects the information in this form. When fully developed, the ASP will make these queries directly to the database interpretation subroutines.

The model program also demonstrates some necessary coordinate system transformations. The object as described in the CAD database exists in one system, but for efficiency during construction, the assembly is turned on the vise table and thus assumes a robot-oriented coordinate system that changes with each move of the table. For simplicity in its database queries, the program translates between these two systems, changing dimensions and orientations as appropriate through a filter that keeps track of prior moves of the vise table.

5.2.4 The Strategy Planner -

As has already been noted, the basic strategy of the ASP is to first determine a disassembly sequence for the given object, and then to reverse it to find an assembly sequence. For those applications involving the repair of an object, both sequences would be involved, either wholly or partially.

This strategy is implemented as a reverse heuristic search. The assembled object, which is really the goal, is taken to be the starting point, and the goal is any state in which the object is completely disassembled. Traversing the search path amounts to removing parts from the object one at a time, and the reversal of the search path is one of possibly many solutions to the opposite search, that is, from parts to assembly. It is assumed that the domain of objects is restricted to those which can be disassembled.

The CAD database contains an implicit tree that represents the totality of all possible search paths, successful or unsuccessful. The root node of the tree is the assembled object; at subsequent levels are lists of all the parts, possible removal trajectories, tool selections, tool placements, and so forth. The ASP is designed to intelligently make enough of this tree explicit so as to elicit a successful disassembly path. "Intelligently" in this context means that at each decision point in the tree,

as much heuristic information as can be queried or inferred from existing information as possible is used to make the best choice. For example, the best removal trajectory for a cylindrical part will probably be along the principal axis of the cylinder.

The major branches of the tree are those that represent the removal of individual parts. Amongst these branches, a heuristic ranking is given, if possible, to the various choices. The first ranking occurs among the parts. Once a part has been chosen, the remainder of the search is depth-first in the tree: all possibilities for removing the part will be exhaustively tried. Each choice, however, is still guided by heuristic information.

Once the successful removal sequence for a part has been determined, the removed part is taken from active consideration and another is chosen. It may be noted that once a point in the tree has been reached at which a part has been removed, the preceding tree structure is assumed correct. Because of the restrictions placed on the domain of objects, backtracking above this point need not occur to determine a correct disassembly sequence. Thus, this is a recursive problem, since the ASP is always presented with the situation of an object and a part to be found and removed, and since the same tree structure, in successively smaller manifestations, is always apparent. It differs from

a purely recursive problem in that information obtained during the removal of previous parts is accumulated and available to guide the removal of subsequent parts.

The model program incorporates most of the above features in the design of the ASP, including the use of heuristics and the tree search. As the first step, the program requests a list of the parts in the assembly and the dimension of the smallest enclosing cube. The latter is used to determine removal points for the parts. (When a part has been moved to a position at which it is entirely outside of the enclosing cube, it is considered to have been removed).

Beginning from the top of the assembled object, the program asks which of the parts are visible, and thus potentially accessible to the IEE gripper tool. (Recall that the model program restricts itself to IEE limitations, including operating on one face of the object at a time). It is then determined along which of the six trajectories these parts may be removed, and thus which are candidates for immediate removal. This is done by requesting the orientation of the principal axis of the attaching bolts in each part. These parts are removed by first removing the bolts and then the part to a point outside of the enclosing cube. (The problem of setting the parts down in a parts rack is not addressed; removal to a point outside the

enclosing cube or envelope is considered sufficient). Another query is made to see if parts formerly hidden from view are now visible. If so, disassembly from the top continues.

Once all possible parts have been removed from the top, previously obtained or inferred information is used to choose another face of the assembly on which to work. If no such information is available, a face is chosen randomly. The procedure continues until all parts have been removed.

As each part is removed, the information necessary to reproduce its removal is concatenated and placed in a disassembly list.

The final task of the ASP is to take the disassembly list that it has generated and either reverse it for assembly or reverse a part of it to accomplish repair. Unfortunately, the process of reversal is not as straightforward as might be hoped. A number of processes are by themselves irreversible, as for example the expansion of a spring; spring-loaded devices require either more tools or more moves to assemble than to disassemble. Gravity is also a factor, at least in earth-based applications: a part held by a bolt to another part may fall off if the bolt is removed, and thus must be held in place during assembly. Additional intelligence must be incorporated about such factor at this stage in the ASP

development, and limitations must be imposed on the types of assembly that can be done.

The model circumvents these problems through its restricted domain of constructible objects. The major problem solved in the model is the reversal of orientation changes as the vise table rotates, which are cumulative but not directly reversible. (This is a result of the fact that orientation transformations are not in general commutative).

5.2.5 Internal Databases -

In addition to the CAD database and the database storing information about the robot capabilities, the ASP will use and maintain internal databases. Here will be stored the information obtained from queries to these supplied databases. In addition, and very importantly, the heuristics and other rules about the process that can be coded in the ASP will be used by an inference engine to perform deductive reasoning on the database information, and to infer and store new information. In this way, all information will be used as fully as possible. Because an internal database is used, expensive outside queries, especially those involving extensive calculations in the CAD database, need only be done once. The internal database also makes it easy to remove a part from consideration. All references to it at the top level of the database are

removed. This renders the part invisible, but makes other information accessible to the remaining removal processes.

The ASP learns in the sense that all of this information is accumulated, so that as the disassembly proceeds, decisions can be made faster and more intelligently.

The model program uses the LISP property list feature to store and manipulate information in its internal database. Global variables and part names have attached values which store both the information directly requested from the CAD database and information deduced from those queries. As an example, each part has associated with it a bolt trajectory orientation, bolt hole positions, a position within the assembly, and an orientation change from its original coordinate system to the assembly coordinate system. As parts are removed, information about them in the database is either removed or made invisible to function calls.

5.3 RESULTS

The model program has been successfully used on sample objects in its restricted domain, producing correct and efficient assembly sequences. The program has also demonstrated the feasibility and significance of many of the knowledge engineering techniques that will be incorporated

into the ASP. First, the utilization of the flexible LISP property list database allows the program to augment its store of knowledge about the problem -- no piece of information is ever requested twice; second, through various rules of inference written into the LISP code, the information content from queries to the CAD database is maximized, in an attempt to minimize the number of queries necessary; third, the use of heuristics has been shown significant in increasing the speed of the search; fourth, the recursive nature of the problem has been naturally modeled in LISP; finally, the translation from implicit to explicit disassembly tree has been made, so that as the program translates from CAD database to robot command language, logical information inherent in the database is made explicit in the commands, and dynamic quantities like trajectories and changing orientations are added to the static geometric description.

5.4 IMPLEMENTATION

The model LISP program, descriptions of its functions, and a sample program run on the blocks model of Figure 8 are here provided.

5.4.1 LISP Code -

The LISP code which comprises the model program is listed.

```
(defun assemble ()
      (prog (port)
            (print '(Welcome to the Automatic Assembler))
            (terpri)
            (get-parts)
            (get-boundary)
            (putprop 'direction 'top 'why)
            (putprop 'orientation '(0 0 0) 'why)
            (putprop 'bolt-number 0 'why)
            (disassemble)
            (terpri)
            (print '(The assembly list may be found in file
                                                  assemlist))
            (terpri)
            (setq port (outfile '"assemlist.dat"))
            (print '(base gripper (0 0 0) (0 0 0) (0 0 0)
                                          (0 0 0)) port)
            (terpri port)
            (printout (chain-orientations (reverse-dis-list
                  (get 'dis-list 'why))) port]
(defun disassemble ()
      (cond ((null (get 'parts 'why)))
            (t (remove-a-part) (disassemble]
(defun reverse-dis-list (dis-list)
      (cond ((null dis-list) nil)
            (t (cons (rdll (car dis-list)) (reverse-dis-list
                                           (cdr dis-list]
(defun get-parts ()
      (print '(What are the parts?))
      (terpri)
      (putprop 'parts (remove 'base (read)) 'why]
(defun get-boundary ()
      (print
       '(What is the dimension of the smallest enclosing
cube?))
      (terpri)
      (putprop 'boundary (read) 'why]
(defun printout (lst port)
      (cond ((null lst) nil)
            (t (print (car lst) port) (terpri port)
(printout (cdr lst) port]
(defun chain-orientations (lst)
      (col lst '(0 0 0]
(defun col (lst orient)
      (cond ((null lst) nil)
```

```
            ((eq (cadar lst) 'wrench)
                  (cons (car lst) (col (cdr lst) orient)))
            (t (cons (co2 (reverse-orient orient) (car lst))
                  (col (cdr lst) (caddar lst]
(defun remove-a-part ()
      (prog (p-list i-list)
            (setq p-list (get 'possible-parts 'why))
            (setq i-list (get 'impossible-parts 'why))
            (cond ((null p-list)
                  (setq p-list (putprop 'possible-parts
                        (get-visible-parts i-list) 'why))))
            (cond ((null p-list) (turn-part i-list)
                                          (return)))
            (putprop 'possible-parts (cdr p-list) 'why)
            (rapl (car p-list]
(defun rdll (dis-elt)
      (list (car dis-elt) (cadr dis-elt) (caddr dis-elt)
            (cadddr dis-elt) (caddddr dis-elt)
                        (caddddr dis-elt]
(defun remove (atm lst)
      (cond ((null lst) nil)
            ((equal atm (car lst)) (cdr lst))
            (t (cons (car lst) (remove atm (cdr lst]
(defun rapl (part)
      (prog (b-list)
            (setq b-list (get part 'bolts))
            (cond ((null b-list) (putprop part (setq b-list
                  (get-bolt-orientation part)) 'bolts)))
            (cond ((correspond b-list (get 'direction 'why))
                        (rap2 part))
                  (t (putprop 'impossible-parts (cons part
                        (get 'impossible-parts 'why)) 'why]
(defun turn-part (i-list)
      (prog (b-list dir)
            (cond ((null i-list) (putprop
                                    'direction (setq dir
                  (next-direction (get 'direction 'why)))
                        'why))
                  (t (setq b-list (get (car i-list) 'bolts))
                        (putprop 'direction (setq dir
                              (get-direction b-list)) 'why
                        (remprop 'impossible-parts 'why)
                     (putprop 'possible-parts (list
                                    (car i-list)) 'why)))
            (putprop 'orientation (compute-orientation dir)
                                                'why]
(defun get-visible-parts (i-list)
      (gvpl (remove-list i-list (get 'parts 'why]
(defun rap2 (part)
      (prog (point dir removal-point answer)
            (setq point (get part 'position))
            (cond ((null point) (putprop part (setq point
```

```
                        (get-position part)) 'position)))
            (setq dir (get 'direction 'why))
            (setq removal-point (compute-removal-point
                  point dir (get 'boundary 'why)))
            (print (list 'Ignoring 'bolts 'can part
                  'be 'moved 'towards 'the dir))
            (terpri)
            (print (list 'from point 'to removal-point '?))
            (terpri)
            (setq answer (read))
            (cond ((eq answer 'yes) (rap3 part point
                  removal-point))
                  (t (putprop 'possible-parts (snoc part
                        (get 'possible-parts 'why)) 'why]
(defun correspond (b-list dir)
      (cond ((eq (get-direction b-list) dir]
(defun get-bolt-orientation (part)
      (prog ()
            (print (list 'What 'is 'the 'orientation 'of
'the
                  'bolts 'in part '?)) (terpri)
            (return (read]
(defun remove-list (a-list b-list)
      (cond ((null a-list) b-list)
            (t (remove-list (cdr a-list) (remove (car
a-list) b-list]
(defun gvpl (parts-list)
      (prog (answer)
            (cond ((null parts-list) (return nil))
                  (t (print (list 'Of 'the 'following
'parts: parts-list))
                        (terpri) (print (list 'which 'are
'visible 'from 'the
                        (get 'direction 'why) '?)) (terpri)
                        (setq answer (read))))
            (cond ((eq answer 'all) (return parts-list))
                  ((eq answer 'none) (return nil))
                  (t (return answer]
(defun get-direction (b-list)
      (cond ((equal (caddr b-list) -90) 'top)
            ((equal (cadr b-list) 90) 'front)
            ((equal (cadr b-list) -90) 'back)
            ((equal (cadr b-list) 0) 'left)
            ((equal (cadr b-list) 180) 'right]
(defun next-direction (dir)
      (cond ((eq dir 'top) 'front)
            ((eq dir 'front) 'back)
            ((eq dir 'back) 'left)
            ((eq dir 'left) 'right)
            ((eq dir 'right) 'top]
(defun compute-orientation (dir)
      (cond ((eq dir 'top) '(0 0 0))
```

```
            ((eq dir 'front) '(-90 0 0))
            ((eq dir 'back) '(90 0 0))
            ((eq dir 'left) '(0 0 -90))
            ((eq dir 'right) '(0 0 90]
(defun rap3 (part point rem-point)
      (remove-bolts (get-bolt-info part) part)
      (rap4 part point rem-point]
(defun compute-removal-point (point dir bound)
      (cond ((eq dir 'top)
                  (list (car point) (add (cadr point) bound)
                                          (caddr point)))
            ((eq dir 'front)
                  (list (car point) (cadr point)
                              (add (caddr point) bound)))
            ((eq dir 'back)
                  (list (car point) (cadr point)
                     (diff (caddr point) bound)))
            ((eq dir 'left)
                  (list (diff (car point) bound)
                              (cadr point) (caddr
point)))
            ((eq dir 'right)
                  (list (add (car point) bound)
                        (cadr point) (caddr point]
(defun get-position (part)
      (prog ()
            (print (list
               'What 'is 'the 'position 'of part '?))
(terpri)
            (return (read]
(defun snoc (atm lst)
      (cond ((null lst) (list atm))
            (t (cons (car lst) (snoc atm (cdr lst]
(defun rap4 (part point rem-point)
      (prog (dir orient)
            (setq dir (get 'direction 'why))
            (setq orient (get 'orientation 'why))
            (putprop 'dis-list (cons (list part 'gripper
orient
                  (add-orientation (get-orientation part)
                  (compute-orientation dir))
                  (transform-point point dir)
                  (transform-point rem-point dir))
                  (get 'dis-list 'why)) 'why)
            (putprop 'parts (remove part (get 'parts 'why))
                                                      'why)
            (putprop 'orientation '(0 0 0) 'why)
            (terpri) (terpri)
            (print (append '(The part named)
                  (cons part '(has been removed))))
                                    (terpri)(terpri]
(defun remove-bolts (bolist part)
```

```
      (cond ((null bolist) nil)
            (t (rbl (car bolist) part) (remove-bolts
                  (cdr bolist) part]
(defun get-bolt-info (part)
      (prog (bonum)
            (print (list 'How 'many 'bolts 'hold part
                  'to 'the 'assembly '?)) (terpri)
            (setq bonum (read))
            (return (gbil bonum 0 part]
(defun add-orientation (orient1 orient2)
      (list (aol (car orient1) (car orient2))
            (aol (cadr orient1) (cadr orient2))
            (aol (caddr orient1) (caddr orient2]
(defun transform-point (point dir)
      (cond ((eq dir 'top) point)
            ((eq dir 'front)
                  (list (car point) (caddr point)
                                    (minus (cadr point))))
            ((eq dir 'back)
                  (list (car point)
                        (minus (caddr point)) (cadr
point)))
            ((eq dir 'left)
                  (list (cadr point) (minus (car point))
                                          (caddr point)))
            ((eq dir 'right)
                  (list (minus (cadr point)) (car point)
                                          (caddr point]
(defun get-orientation (part)
      (prog ()
            (print (list 'What 'is 'the
                        'orientation 'of part '?))
(terpri)
            (return (read]
(defun rbl (bolt part)
      (prog (dir point)
            (setq dir (get 'direction 'why))
            (setq point (get bolt 'position))
            (putprop 'dis-list (cons (list bolt 'wrench
                                          '(0 0 0)
                  (add-orientation (get part 'bolts)
                  (compute-orientation dir))
                                    (transform-point
                  point dir) (transform-point
                        (compute-removal-point point dir
                  (get bolt 'length)) dir))
                                    (get 'dis-list 'why))
'why]
(defun gbil (bonum num part)
      (cond ((zerop bonum) nil)
            (t (cons (gbi2 (add1 num) part)
                  (gbil (sub1 bonum) (add1 num) part]
```

```
(defun aol (num1 num2)
      (prog (newnum)
            (setq newnum (add num1 num2))
            (cond ((greaterp newnum 180)
                        (return (diff newnum 360)))
                  ((lessp newnum -170)
                        (return (add newnum 360)))
                  (t (return newnum]
(defun reverse-orient (orient)
      (list (minus (car orient)) (minus (cadr orient))
                                    (minus (caddr orient]
(defun gbi2 (bonum part)
      (prog (boname pos len glonum)
     (setq glonum (add1 (get 'bolt-number 'why)))
     (putprop 'bolt-number glonum 'why)
            (print (list 'What 'is 'the 'position 'of 'bolt
                                                      'number
                  bonum 'in part '?)) (terpri)
            (setq pos (read))
            (setq len (bolt_length glonum))
            (setq boname (concat 'bolt glonum))
            (putprop boname pos 'position)
            (putprop boname len 'length)
            (return boname]
(defun co2 (orient dis-elt)
      (cons (car dis-elt) (cons (cadr dis-elt) (cons
            (add-orientation
            (caddr dis-elt) orient) (cdddr dis-elt]
```

5.4.2 Function Descriptions -

This section gives a brief description of each function in the preceding model LISP implementation. Included for each function is a list of the support functions that it calls.

assemble -- sets up the database, gets part names and boundary, starts disassembly process, prints messages, calls disassembly list reversing functions

calls -- disassemble, reverse-dis-list, get-parts, printout, chain-orientations, get-boundary

disassemble -- calls the part removal function until the parts list is empty

calls -- remove-a-part

reverse-dis-list -- recursively applies rdll to each element of the disassembly list

calls -- rdll

get-parts -- queries the user for the list of parts and reads it in, removing the part "base" if necessary, and stores it in the database

calls -- remove

get-boundary -- asks for the dimension of the smallest enclosing cube and stores it in the database

printout -- writes the assembly list to the file of robot commands

chain-orientations -- calls col with the initial orientation (0 0 0)

calls -- col

col -- performs the process of reversing the orientation information as the disassembly list is made into the assembly list

calls -- co2, reverse-orient

remove-a-part -- attempts to remove the first part in a list of possible parts; if it cannot, it turns the assembly to a new face

calls -- rapl, turn-part, get-visible-parts

rdll -- resequences a disassembly list element so that it can go on to the assembly list

remove -- removes an atom from a list at its first occurence

rapl -- gets the removal trajectory direction for a part from the orientation of its bolts, and either continues to attempt to remove the part or puts it on the impossible (from the current direction) list

calls -- rap2, correspond, get-bolt-orientation

turn-part -- if there is any impossible list, turn the assembly so that the first part on that list can be removed; otherwise, turn the assembly to a new direction

calls -- get-direction, next-direction, compute-orientation

get-visible-parts -- removes the impossible parts from the current parts list and calls gvpl on the remainder

calls -- remove-list, gvpl

rap2 -- gets the position of a part and computes its removal point; then asks whether the required removal can be made - if so, the removal process continues, if not, the part is put on the back of the possible list

calls -- rap3, compute-removal-point, get-position, snoc

correspond -- gets the direction from a bolt list and compares it to a given direction

calls -- get-direction

get-bolt-orientation -- queries the user for the orientation of the bolts in a particular part and reads the answer

remove-list -- removes all the elements of one list from a second list

calls -- remove

gvpl -- asks which of a list of parts are visible from a particular direction; accepts "all" and "none"

get-direction -- determines a direction from the orientation of a bolt

next-direction -- given a direction, this returns the next in the sequence

compute-orientation -- given a direction, returns the orientation triplet which, if applied to the object, would realign the top to that direction

rap3 -- finds out about and removes the bolts from a part; then removes the part

calls -- rap4, remove-bolts, get-bolt-info

compute-removal-point -- given a point, direction, and boundary, finds a new point at which a part will have been removed

get-position -- queries for the position of a part

snoc -- puts an element on the end of a list

rap4 -- does the actual removal of a part by making up a disassembly list element, removing the part from the database, and informing the user

calls -- add-orientation, transform-point, get-orientation, remove, compute-orientation

remove-bolts -- recursively calls rbl on a list of bolts

calls -- rbl

get-bolt-info -- asks for the number of bolts holding a part

calls -- gbil

add-orientation -- given two orientation triplets, calls aol on each member of the triplet

calls -- aol

transform-point -- translates the numbers in a location triplet to reflect a change in orientation of the object

get-orientation -- queries for the orientation triplet of a part

rbl -- removes a bolt by adding an element to the disassembly list

calls -- add-orientation, transform-point, compute-removal-point, compute-orientation

gbil -- calls gbi2 for the number of bolts in a part

calls -- gbi2

aol -- adds two orientation angles together; reduces if greater than 180 or less than -170

reverse-orient -- reverses an orientation by simply negating each of the three angles in the triplet

gbi2 -- asks for the position of a bolt in a part and calls the FORTRAN program "bolt_length" to get the length of the bolt from a database

co2 -- makes a new disassembly list element out of an old element and an orientation

5.4.3 Sample Program Run -

This section gives a sample output from the model LISP implementation. An object composed of five blocks and held together with seven bolts of varying lengths is verbally described to the program. The object is depicted in Figure 8. The following is output:

```
(base gripper (0 0 0) (0 0 0) (0 0 0) (0 0 0))
(inblock gripper (0 0 90) (90 90 90) (-3 13 3) (-3 4 3))
(bolt7 wrench (0 0 0) (0 180 90) (-2 6 2) (-2 5 2))
(sideblock gripper (0 0 180) (90 0 -90) (3 9 4) (3 0 4))
(bolt6 wrench (0 0 0) (90 0 -90) (2 -2 5) (2 -4 5))
(bolt5 wrench (0 0 0) (90 0 -90) (2 1 5) (2 -1 5))
(topblock gripper (0 0 90) (0 90 0) (3 14 4) (3 5 4))
(bolt4 wrench (0 0 0) (0 0 -90) (4 8 3) (4 6 3))
(bolt3 wrench (0 0 0) (0 0 -90) (4 8 1) (4 6 1))
(overhang gripper (0 0 0) (90 90 0) (0 17 4) (0 8 4))
(bolt2 wrench (0 0 0) (0 0 -90) (1 12 3) (1 8 3))
(bolt1 wrench (0 0 0) (0 0 -90) (1 12 1) (1 8 1))
```

Each line in this output contains the information necessary for a hypothetical robot to add the described part to the assembly. Words describe the parts and tools employed, and numerical triplets describe orientations and positions. The numbers in the orientation triplets define angular rotations in degrees about the X, Y, and Z axes respectively. Numbers in the position triplets define points in three-space. A three-axis coordinate system is defined for the assembly as a whole and for each part as it is located in the parts rack. One point in the assembly and one in each part is set to be the origin of the associated coordinate system. There are six elements in each output

line, which have the following significance:

1. Name of the part.
2. Name of the tool to be used.
3. Orientation change of the assembly for addition of the part.
4. Orientation change of the part.
5. Starting position of the part for addition.
6. Ending (assembled) position of the part.

As an example, consider the following output line:

(block gripper (0 0 90) (90 0 90) (3 4 6) (3 4 1))

This would cause the robot to perform the following functions: rotate the entire assembly ninety degrees about the Z axis; take the part called 'block' with the tool called 'gripper' from the parts rack; rotate the part ninety degrees about both the X and Z axes; move the origin point of the part to the point (3 4 6) in the assembly coordinate system; move the origin point of the part to the point (3 4 1) in the assembly coordinate system, thus adding the part to the assembly.

6.0 CONCLUSIONS AND FUTURE WORK

The development of robotic hardware and sequence planning software is an effort at Goddard Space Flight Center to provide robotic assistance in the design, assembly and servicing of NASA hardware for both space and ground-based applications. To this end the Intelligent End Effector (IEE), a robot equipped with compliance and force feedback for precision assembly, and knowledge engineering and robot control software techniques have been combined with an existing Computer-Aided Design (CAD) facility in a synergism of expertise, with promising results.

The IEE, with its compliance, force feedback, and six-degree-of-freedom capabilities, has been built and proved capable of functioning in the engineering environment for which it was designed. A significant body of software exists for controlling the IEE, for positioning the movable platforms, and for reading and interpreting its force-feedback sensors. In addition, software has been provided for future enhancements to the robot, including controlling programs for a gripper and a wrist mechanism.

Three problems were chosen for the demonstration of the IEE and for the development of higher-level robot control software: inserting a peg in a hole, mating a 25 pin D-type connector, and screwing a bolt into a threaded hole. Success was achieved in each case, even though considerable

uncertainty in position and orientation of parts was allowed, requiring the robot's acquisition of knowledge about the operating environment. The importance of compliance and force feedback in precision assembly was proved, and the design and use of a bolt spinner as one of several proposed tools was accomplished.

An important part of this project is the use of knowledge engineering techniques to address the problem of translating the implicit construction sequence inbedded in the information available about an object, the robot, and the process into an explicit sequence of robot commands. A system is under development to do this, using the technique of reverse heuristic search and the CAD geometric database description of the hardware under consideration. A model program was written to demonstrate and test significant features of the system, including knowledge acquisition, use of heuristics like part visibility and bolt hole position, dynamic databases, and recursive search.

Future work on automated assembly system will focus on two principal areas: the completion of the Automated Sequence Planner (ASP) program, and an expansion of the robot control software, with the inclusion of artificial intelligence techniques.

The ASP program will be implemented in the language Prolog, which allows a natural modeling of many of the aspects of the problem. The IGES format CAD database of a piece of hardware will be input to the program, which will automatically generate a sequence of robot moves to construct the object. Additional information about the characteristics of the robot and tools available will also be input to the program. Completion of the program will be the result of significant work in three-dimensional space planning and logical inference on the available data.

Once the robot move sequence has been generated, it will be provided to a second Prolog program, which will represent an expansion of current robot control software such as the connector mating program. This new program will be able to receive and intelligently interpret the force-feedback information produced as assembly occurs. Eventually, this program will be combined with the ASP in a system that will incorporate cooperative error analysis and contingency handling, with the possibility of designing dynamic robot move sequences based on operational information.

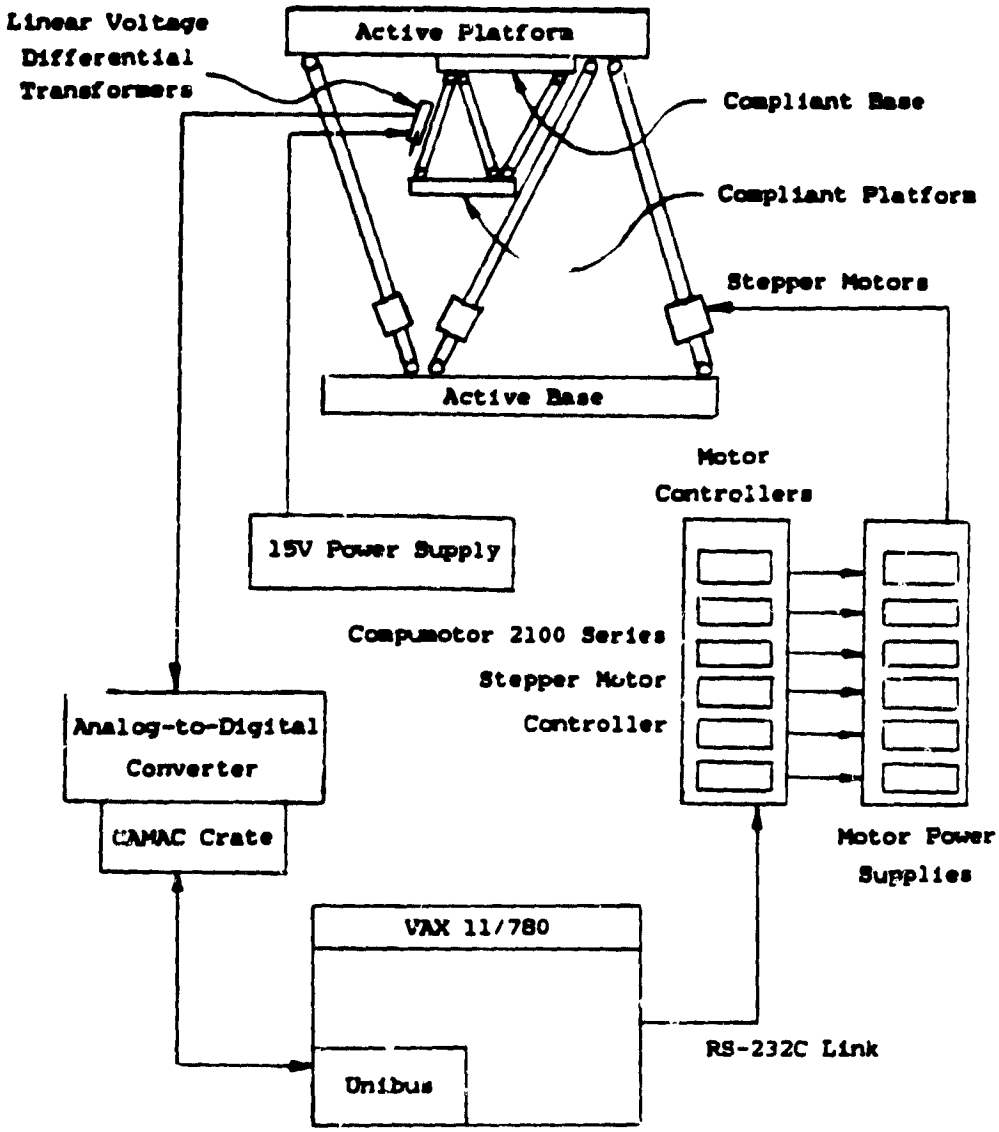


Figure 1

BLOCK DIAGRAM OF IEE SYSTEM

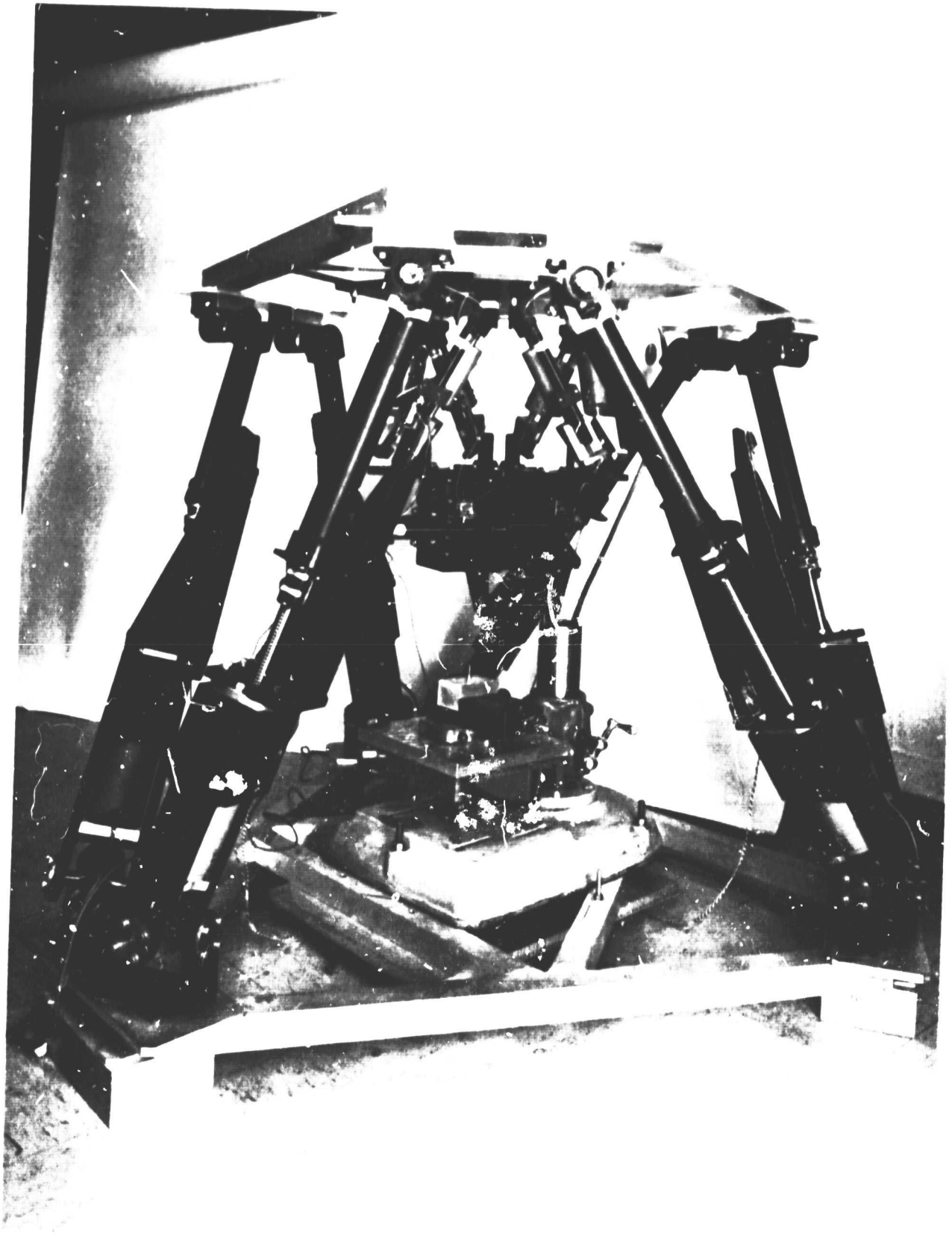


Fig. 2A - PHOTOGRAPH OF JEE

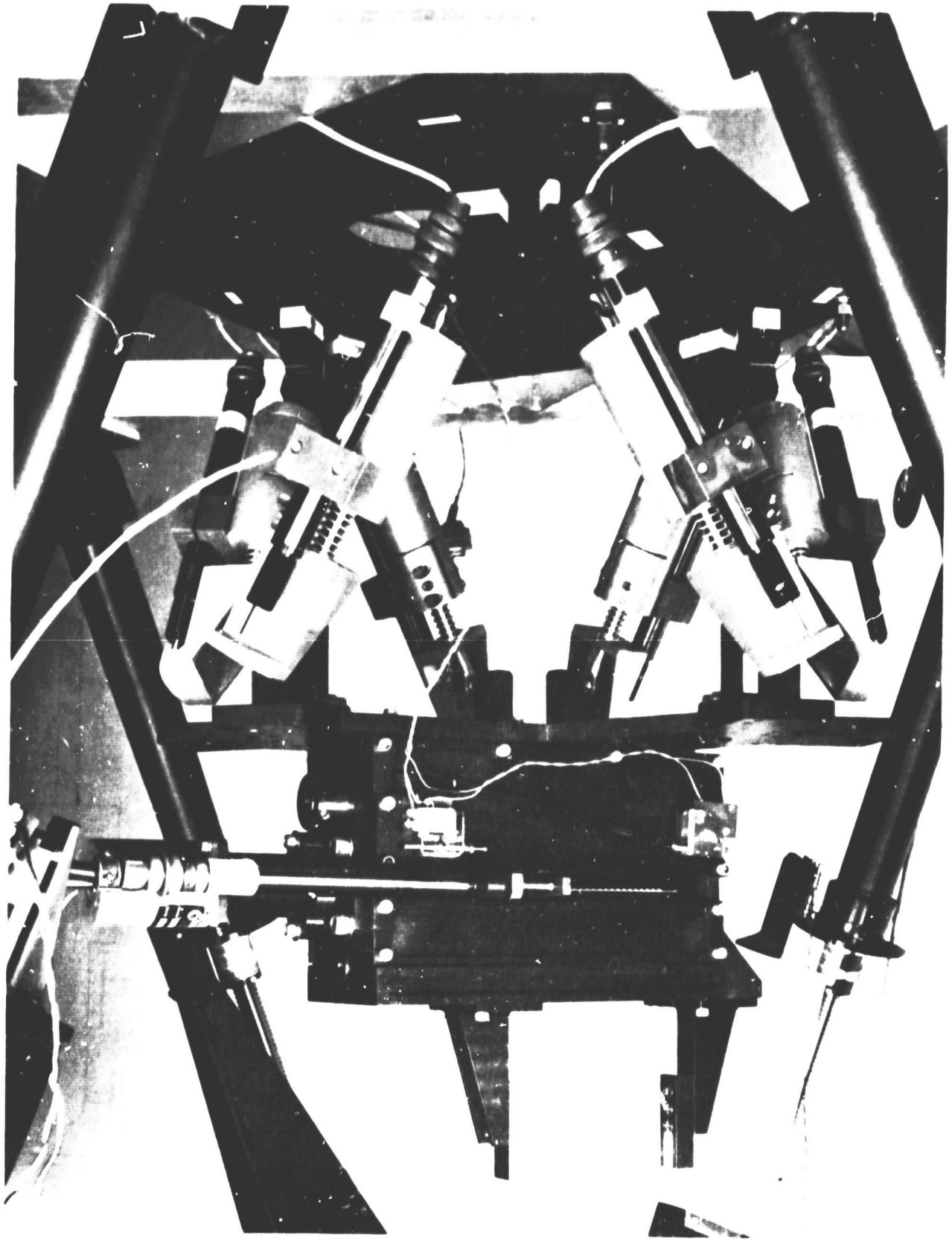


Fig. 2B - PHOTOGRAPH OF IEE

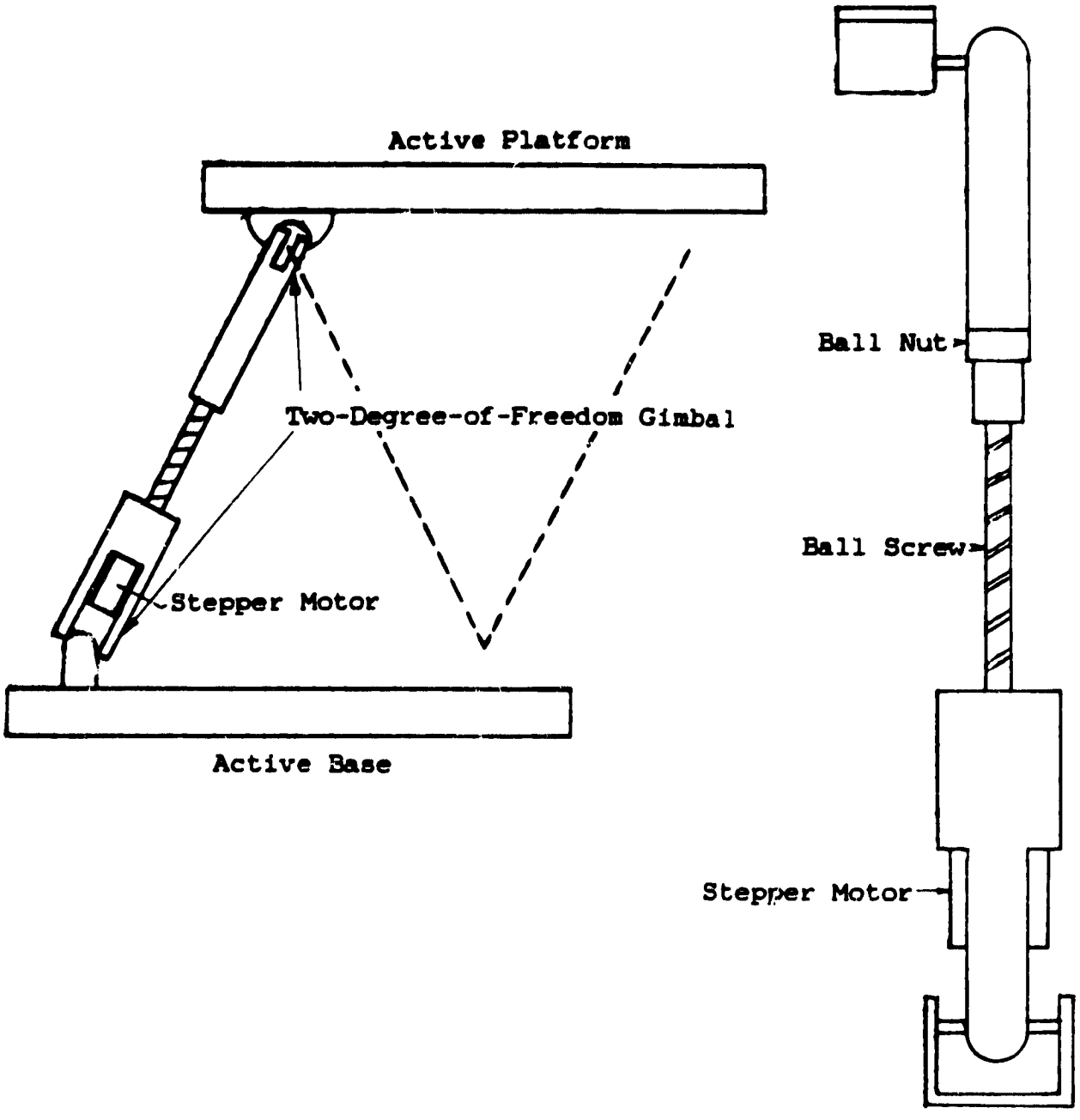


Figure 3

SIX-DEGREE-OF-FREEDOM POSITIONING MECHANISM

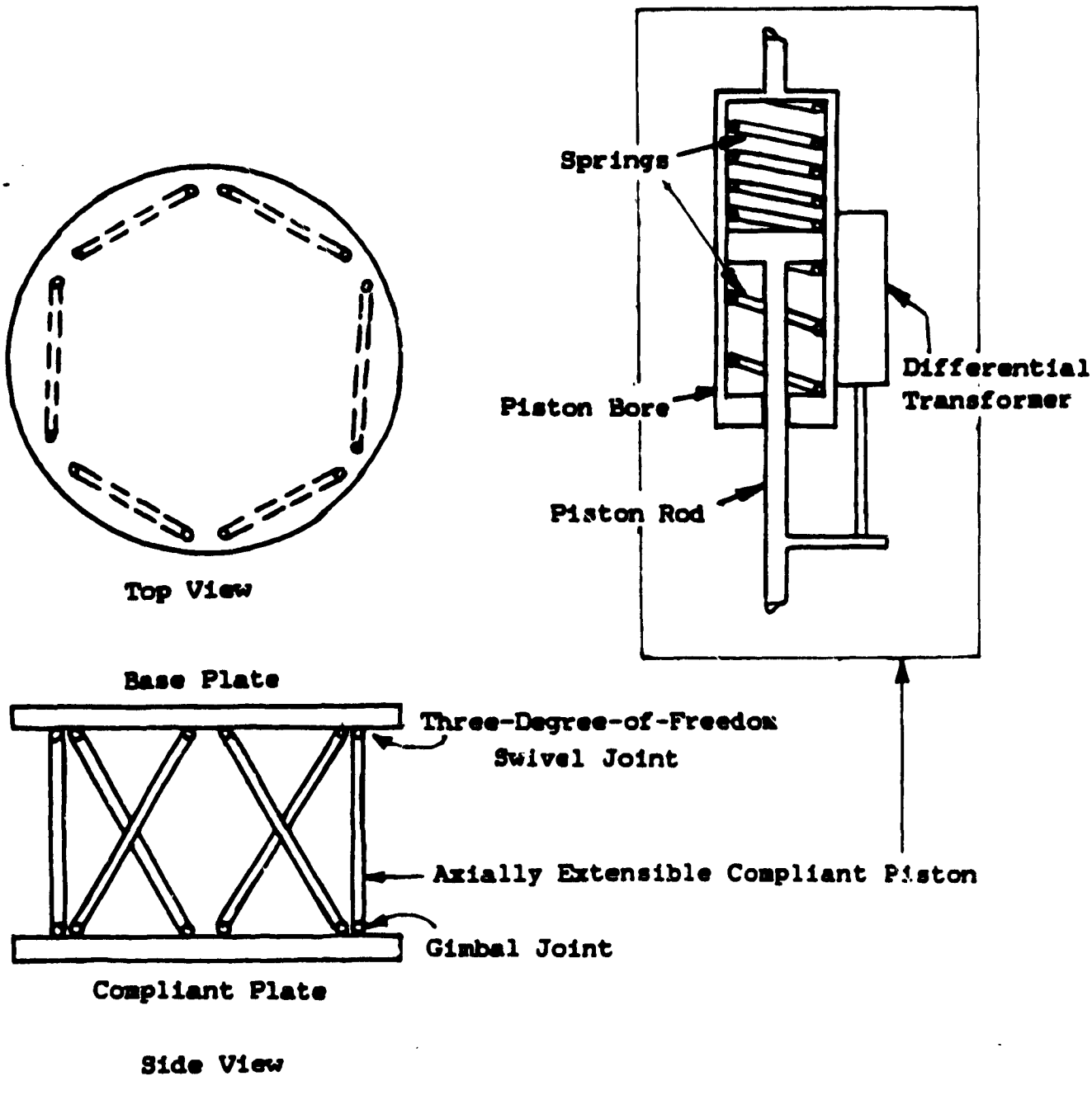


Figure 4

COMPLIANT, FORCE-FEEDBACK MECHANISM

```
MTN_POSITION_TO
  (moves the IEE to a specific point and orientation in space)

        MTN___SET_POSITION
          (main positioning routine)

                AP_SET_POSITION
                  (computes the new position of the active platform (AP))

                        SDF_GET_LENGTHS
                          (compute the new lengths of each actuator)

                        (compute the steps to send to each motor)

                        AP___SET_ACTUATORS
                          (sets the distance for each motor and calculates the
                           time for the move)

                                MTR_SET_DISTANCE
                                  (sends the distance command to each motor)

                                        (convert the distance in steps to an
                                         ASCII command string)

                                        MTR_SEND_COMMAND
                                          (sends the command to the motors)

                                AP___TRAVEL_TIME
                                  (computes the time it will take for the
                                   AP to complete the move)

                                AP___FIND_VELOCITY
                                  (computes the velocity with which each
                                   motor will move)

                                MTR_SET_VELOCITY
                                  (sends the velocity command to each motor)

                                        (convert the distance in steps to an
                                         ASCII command string)

                                        MTR_SEND_COMMAND
                                          (sends the command to the motors)

        MTN___START_MOTION
          (starts the AP moving)

                AP_START_MOTION
                  (starts the motors moving)

                        MTR_SEND_GO
                          (sends the go command to the motors)

                                        MTR_SEND_COMMAND
                                          (sends the command to the motors)

        MTN___CLEANUP_POSITION
          (cleans up after position the AP)

                CP_GET_POSITION
                  (resets the positonal data concerning the compliant
                   platform)
```

Figure 5

MOTION FLOW CONTROL

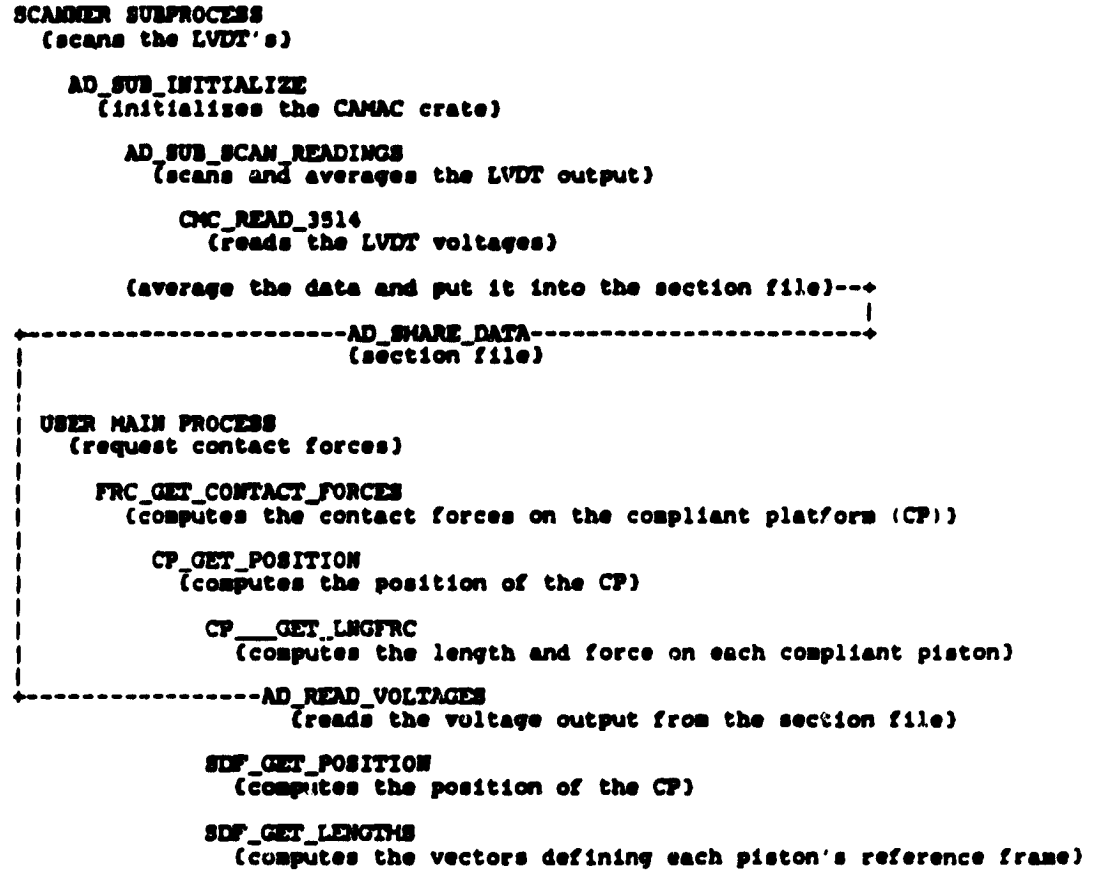
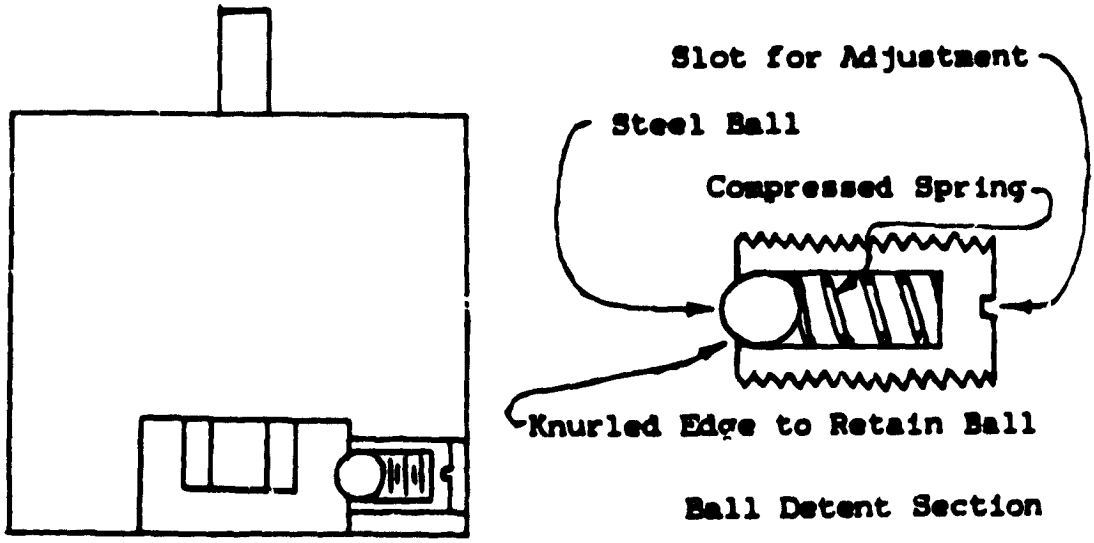


Figure 6

CONTROL FLOW OF FORCE ACQUISITION



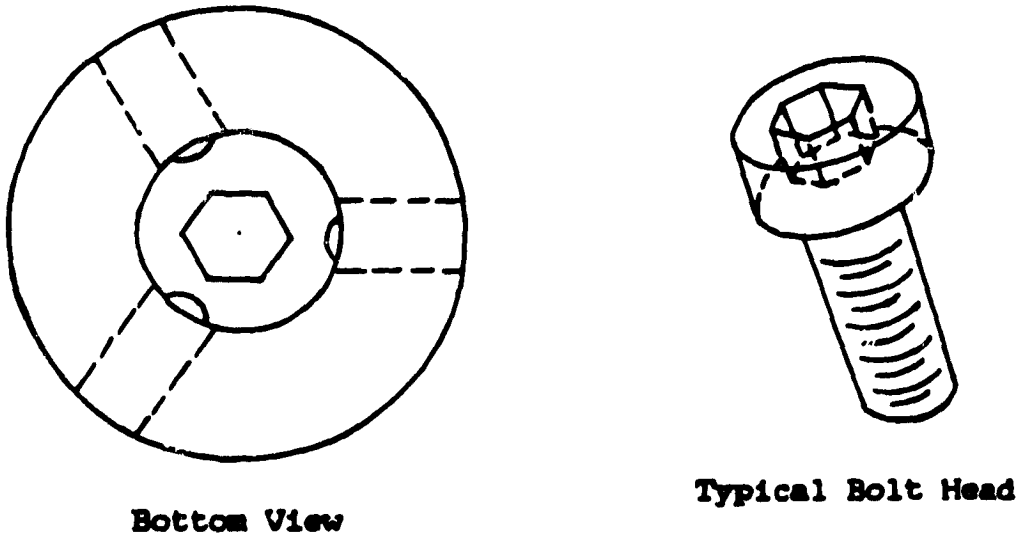


Figure 7

BOLT HOLDING DEVICE

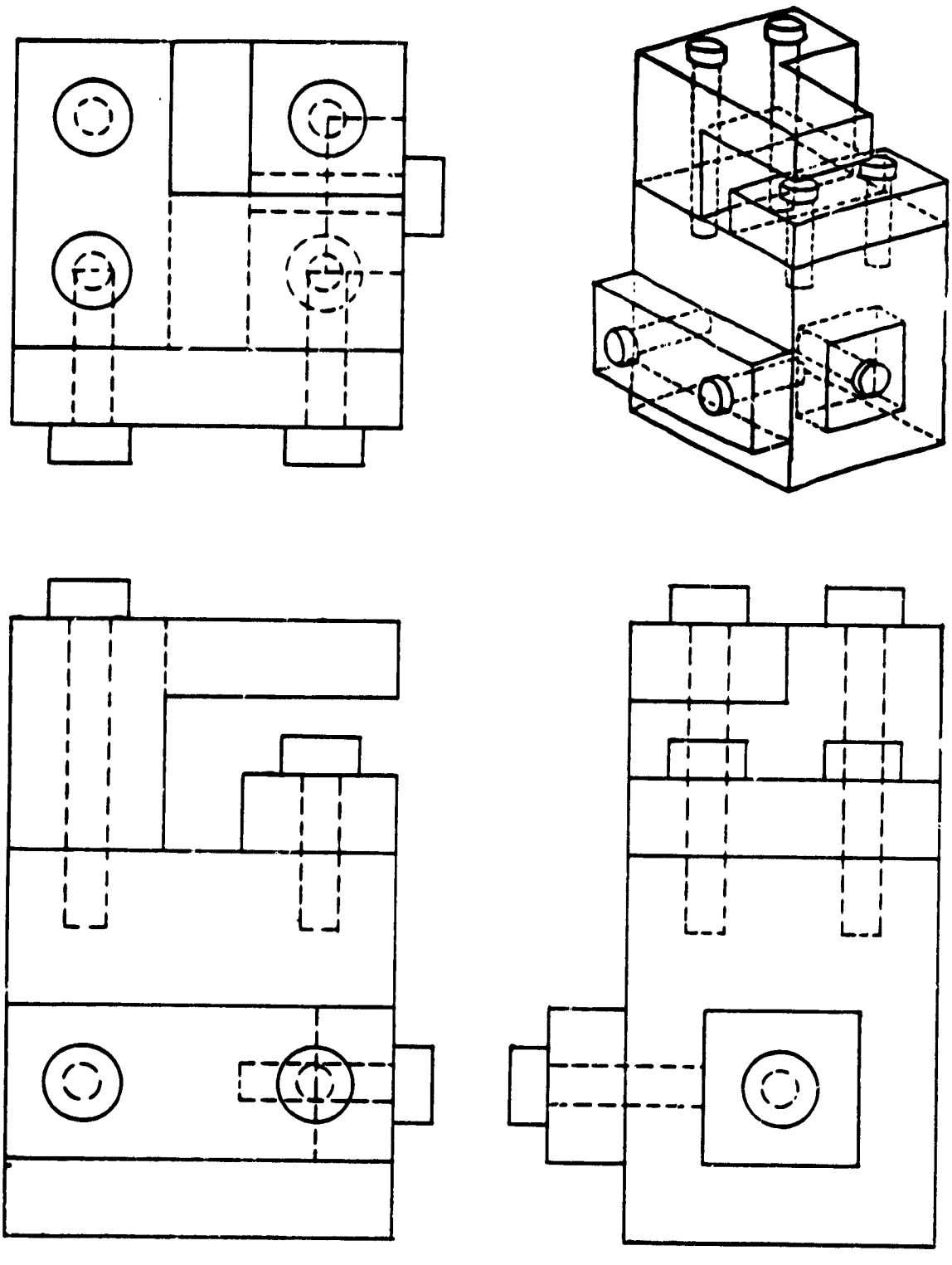


Figure 8

MECHANICAL DRAWING OF BLOCKS MODEL

REFERENCES

Ambler, A.P. et al. 1973 (August). "A Versatile Computer Controlled Assembly System". Proceedings of the Third International Joint Conference on Artificial Intelligence. Menlo Park, California: Stanford Research Institute. pp. 298-307.

Bolles, R.C. and Cain, R.A. 1982. "Recognizing and Locating Partially Visible Objects: The Local-feature-focus Method". International Journal of Robotics Research. Vol. 1, No. 3. pp. 57-82.

Brooks, R.A. 1983. "Symbolic Error Analysis and Robot Planning". Robotics Research. Vol. 1, No. 2. pp. 29-68.

Dieudonne, James E., Parrish, R.V., and Bardusch, Richard E. 1972 (November). "An Actuator Extension Transformation for a Motion Simulator and an Inverse Transformation Applying Newton-Raphson's Method". Technical Note D-7067. Washington, D.C.: National Aeronautics and Space Administration.

Drake, S. 1977 (September). "Using Compliance in Lieu of Sensory Feedback for Automatic Assembly". Report T-657. Cambridge, Massachusetts: Charles Stark Draper Laboratory.

Fahlman, S.E. 1973 (May). "A Planning System for Robot Construction Tasks". Technical Report 283. Cambridge, Massachusetts: Massachusetts Institute of Technology Artificial Intelligence Laboratory.

Goto, Takeyasu, and Inoyama. 1980. "Control Algorithm for Precision Insert Operation Robots". IEEE Transactions on Systems, Man, and Cybernetics. Vol. 10. pp. 19-24.

Harmon, L.D. 1982. "Automated Tactile Sensing". International Journal of Robotics Research. Vol. 1, No. 2. pp. 3-32.

Hillis, W.D. 1982. "A High Resolution Imaging Touch Sensor". International Journal of Robotics Research. Vol. 1, No. 2. pp. 33-44.

Inoue, H. 1974 (August). "Force Feedback in Precise Assembly Tasks". Massachusetts Institute of Technology Artificial Intelligence Laboratory Memo 308.

Ishida, T. 1977. "Force Control in Coordination of Two Arms". Proceedings of the Fifth International Joint Conference on Artificial Intelligence. Cambridge, Massachusetts: Massachusetts Institute of Technology Press. pp. 717-722.

Klein, C.A. and Briggs, R.L. 1980. "Use of Active Compliance in the Control of Legged Vehicles". IEEE Transactions on Systems, Man, and Cybernetics. Vol. 10, No. 7. pp. 393-400.

Lozano-Perez, T. 1976 (December). "The Design of a Mechanical Assembly System". Technical Report 937. Cambridge, Massachusetts: Massachusetts Institute of Technology Artificial Intelligence Laboratory.

Nevins, J., et al. 1977. "Exploratory Research in Industrial Modular Assembly". Report R-1111. Cambridge, Massachusetts: Charles Stark Draper Laboratory.

Orin, D.E. 1976. "Supervisory Control of a Multi-legged Robot. International Journal of Robotics Research. Vol. 1, No. 1. pp. 79-91.

Paul, R. and Shimano, B. 1976. "Compliance and Control". Proceedings of the 1976 Joint Automatic Control Conference. New York, New York: ASME. pp. 687-693.

Popplestone, R.J., Ambler, A.P., and Bellos, I.M. 1980. "An Interpreter for a Language for Describing Assemblies". Artificial Intelligence. Vol. 14. pp. 79-107.

Salisbury, J.K. and Craig, J.J. 1982. "Articulated Hands: Force Control and Kinematic Issues". International Journal of Robotics Research.

Schratt, R.D., et al. 1980. "Possibilities and Limits for the Application of Industrial Robots in New Fields". Proceedings of the Tenth International Symposium on Industrial Robots and the Fifth International Conference on Industrial Robot Technology. pp. 421-431.

Taylor, R.H. 1976 (July). "A Synthesis of Manipulator Control Programs from Task-level Descriptions". AIM 282. Stanford, California: Stanford University Artificial Intelligence Laboratory.

Tsuji, S. and Nakamura, A. 1975 (August). "Recognition of an Object in a Stack of Industrial Parts". Proceedings

C-2

of the Fourth International Joint Conference on Artificial Intelligence. Cambridge, Massachusetts: Massachusetts Institute of Technology Artificial Intelligence Laboratory. pp. 811-818.

Whitney, D.E. 1976. "Force-Feedback Control of Manipulator Fine Motions". Transactions of the ASME Journal of Dynamic Systems Measurement and Control. Vol. 99, No. 2. pp. 91-97.

BIBLIOGRAPHIC DATA SHEET

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
TM 86111		

4. Title and Subtitle	5. Report Date
Design and Implementation of a Compliant Robot with Feedback and Strategy Planning Software	May 29, 1984
	6. Performing Organization Code
7. Author(s) T. Premack, F. M. Strempek, L. A. Solis, S. S. Brodd, E. P. Cutler, L. R. Purves	8. Performing Organization Report No.
9. Performing Organization Name and Address Code 753 Engineering Design Branch	10. Work Unit No.
	11. Contract or Grant No. NAS5-28200
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address Code 753 NASA/GSFC Greenbelt, MD 20771	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Force-feedback robotics techniques are being developed for automated precision assembly and servicing of NASA space flight equipment. Design and implementation of a prototype robot which provides compliance and monitors forces is in progress. Computer software to specify assembly steps and makes force-feedback adjustments during assembly are coded and tested for three generically different precision mating problems. A model program demonstrates that a suitably autonomous robot can plan its own strategy.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Robotics, Force Feedback, Compliance, Knowledge Engineering, Artificial Intelligence, Strategy Planning, Automated Autonomous Activity	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
		97	

*For sale by the National Technical Information Service, Springfield, Virginia

GSFC 25-44 (10/77)