

**Supervisory Control Algorithms
For Telerobotic Space Structure Assembly**

by

DAVID ELLIOTT ANDERSON

**B.S., Massachusetts Institute of Technology
(1986)**

**Submitted in partial fulfillment of
the requirements for the degree of**

**MASTER OF SCIENCE
IN AERONAUTICS AND ASTRONAUTICS**

at the

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 6, 1988**

© Massachusetts Institute of Technology 1988

Signature of Author_____

Department of Aeronautics and Astronautics
May 6, 1988

Certified by_____

Professor David L. Akin
Thesis Supervisor

Accepted by_____

Professor Harold Y. Wachman
Chairman, Departmental Graduate Committee

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

MAY 24 1988

LIBRARIES

Aero



SUPERVISORY CONTROL ALGORITHMS FOR TELEROBOTIC SPACE STRUCTURE ASSEMBLY

by

DAVID ELLIOTT ANDERSON

Submitted to the Department of Aeronautics and Astronautics
on May 6, 1988 in partial fulfillment of the requirements for
the degree of Master of Science

ABSTRACT

A method for applying a task primitive supervisory control capability to a structural assembly teleoperator is presented and compared with manual control modes. The task of structural assembly is often repetitive. Several of the basic steps in an assembly can be completed by the computer faster with more accuracy and repeatability, lowering the workload on operator and increasing productivity. These results are demonstrated by example using the Beam Assembly Teleoperator.

The primary application of the Beam Assembly Teleoperator is the assembly of tetrahedral structures constructed of six beams and four nodes. The beam manipulation and joint assembly portions of the assembly are augmented by the task primitive supervisory control algorithm, Triad. Manual and supervisory modes are compared by repeated joint assembly and by full structure assembly.

Use of Triad proved to be significantly faster in repeated joint assembly, yielding times for a naïve subject using Triad equal or better than an experienced operator under manual control.

For full EASE structural assembly, Triad produced an estimated full assembly time 20% faster than the previously established manual time, despite increased times for the portions of assembly not under Triad control. The times for the portions assembly under Triad control were decreased 43% from the corresponding manual times.

Thesis Supervisor: Prof. David L. Akin

Title: Rockwell Assistant Professor of Aeronautics and Astronautics

ACKNOWLEDGEMENTS

For the correct combination of guidance and freedom, and for a unique research environment, I sincerely thank Professor Dave Akin.

Thanks also to Herb Viggh and John Spofford for bringing BAT into functionality and tolerating my questions about circuitry and 'Spofford Assembly Language'. Thanks to the BAT UROP'ers, Lisa, Jud, and Beth, for their skill with vacuum grease and devotion to WIF tests. Courage, Jud. BAT can function. When in doubt, read chapter 4.

For keeping me going through Multivariable, thanks to Sue and Rob. I am sure that if I put as much time into a marriage as we did into that class, it has got to work out.

For a truly eventful working environment, thanks to office mates Janice, Craig, Mary, Vicky, Wendy, Tony, and to Russ, even if he did make me sit in the corner for the last four months.

Thanks to my roommates, Chris, Win, Dana, and Jean±1, for good times, good food, a comfortable home and lots of cran-raspberry juice.

Thanks especially to my family out west in Oregon for never letting me forget that there is a world outside of Boston with good people, moderate weather, and decent pavement.

This research made possible by NASA's Office of Aeronautics and Space Technology under NASA Grant NAGW-21.

Table of Contents

1. INTRODUCTION.....	6
1.1 Teleoperation	6
1.2 The addition of supervisory control.....	7
1.3 EASE and the SSL loop group	8
1.4 The vehicle, BAT	10
1.5 The control station, ICS.....	13
1.6 Triad	15
1.7 Scope of thesis	16
2. UPGRADED CONTROL ALGORITHMS 5DOF ARM.....	17
2.1 Previous control system.....	17
2.1.1 Chain of command to move the arm.....	17
2.1.2 The role of BATSOFT.....	18
2.1.3 The role of the joint cards.....	18
2.1.4 Problems with the previous control system.....	19
2.2 Digital PID control	20
2.2.1 Digital approximations of PID control	20
2.2.2 Tuning	21
2.3 Application to BAT	22
2.3.1 Direct application	22
2.3.2 Addition of a separate integrator.....	23
3. TRIAD, TASK PRIMITIVE SUPERVISORY CONTROL.....	24
3.1 Theory.....	24
3.1.1 Supervisory Control.....	24
3.1.2 Trajectory control and path planning	24
3.2 Method	25
3.2.1 BAT's Limitations.....	25
3.2.2 Application to BAT.....	26
3.2.3 Triad trajectory development	28
3.3 Application to the EASE structure assembly.....	29
3.3.1 Breakdown of EASE joint assembly	29
3.3.2 Assembling a joint manually	30
3.3.3 Assembling a joint with Triad.....	30
4. MANUAL vs. TRIAD ASSEMBLY of the EASE JOINT	31
4.1 Manual assembly using P-type control law	31
4.2 Manual assembly using PD-type control law	31
4.3 Manual assembly using PID-type control law	34
4.4 Triad assembly	35
4.4.1 The procedure.....	35
4.4.2 Results	36
4.5 Comparing manual and Triad assemblies	37
4.5.1 Overall time.....	37
4.5.2 Joint completion time.....	38
4.5.3 Tasks covered by Triad	39

5. FULL EASE STRUCTURE ASSEMBLY	40
5.1 Results of NB-67	40
5.1.1 Test setup and procedure	40
5.1.2 Times for assembly and proportions	41
5.1.3 Comments on Fatigue.....	42
5.2 Results of NB-67C	42
5.2.1 Test setup and procedure	42
5.2.2 Mechanical setbacks.....	44
5.2.3 Estimation of Triad assembly times and proportions.....	44
5.3 Comparison of NB-67 and NB-67C data	45
5.3.1 Direct comparison of assembly times	45
5.3.2 Weighted comparison	46
5.4 Limitations of the present configuration.....	47
6. CONCLUSIONS AND RECOMMENDATIONS.....	48
6.1 Conclusions.....	48
6.2 The future of BAT	49
REFERENCES	52
APPENDIX I. BAT mobility using RMS.....	53
APPENDIX II. BATSOFT	55
APPENDIX III. JCS code.....	119
APPENDIX IV. Triad trajectories	132
APPENDIX V. EASE structure assembly schedule	150

1. INTRODUCTION

1.1 Teleoperation

In hazardous environments such as the containment vessel of a nuclear reactor, the high pressure depths of the ocean, or radiation-intense regions of space, it is dangerous or impossible to have a human on site. In these situations, the human must be kept remote from the site while a machine completes the task. One way to do this, with existing technology, is teleoperation, which involves an on-site machine controlled by a human in some safe remote location. A teleoperator enacts the human's commands, scaling them and enhancing them. From a safe distance, the human can use the enhancements of the system to move an object larger than he can move or make fine movements on a smaller scale.

In the case of the Beam Assembly Teleoperator (BAT) developed by the M.I.T. Space Systems Lab (SSL), the mobile machine has propellers to give the teleoperative base 6 degrees of freedom (DOF), a dextrous, anthropomorphic arm for manipulation, a claw for grappling and a claw for carrying. BAT operates in the neutrally buoyant environment of a swimming pool, controlled by an operator in the Integrated Control Station (ICS) on the surface.

The ultimate in teleoperation is called telepresence: "at the work site, the manipulators have the dexterity to allow the operator to perform normal human functions" and "at the control station, the operator receives sufficient quantity and quality of sensory feedback to provide a feeling of actual presence at the work site." [1] Telepresence allows the operator to use all the control strategies ingrained in the neural network of his brain from the long years of childhood and adulthood that allow him to use his body to interact with the environment and apply them to the remote work site. The control station and the teleoperator are "transparent" in the control scheme. This would be a tremendous savings in training time. Instead of the operator training to associate this joystick with that manipulator joint and this toggle with that thruster, he simply uses the training he has on

the use of his own body to associate closing his fingers with grasping an object.

Unfortunately, no truly telepresent system has yet been developed.

1.2 The addition of supervisory control

If a teleoperator that could transmit all the agility of a human could be built, it would equal human on-site productivity. But such a teleoperator must still be controlled manually through each step of the task at hand, limited by that same agility, and by the motivation and fatigue of the operator. One way to increase productivity is to hand off some portions of the task to a computer control system on the vehicle. The computer receives commands from the operator, and, by means of sensors and actuators, implements them. The operator monitors the system, intermittently adjusting or supplying new commands. This is called supervisory control.

Supervisory control is the middle ground between completely manual teleoperation and completely autonomous robotics. Depending on the sophistication of the system, the operator hands off some level of control, such as the major task level, task component level, or task primitive level. A major task is one requiring the orchestration of all of the system components. The computer must have the ability to plan, diagnose, and react to its environment. Operator commands to a major task level supervisory controller would be on the order of "diagnose and repair satellite telemetry module." Similarly, at the task component level, a controller may also require the use of all system components, but does not require planning or diagnoses. The computer implements a series of preplanned manipulations. An example of an operator command to a task component level supervisory controller is "detach and remove satellite telemetry module." At a more basic level, a task primitive involves a single system component. The computer implements a single manipulation such as "move freed module from point A to point B."

Supervisory control allows for quicker task completion, quicker automatic reflexes to disturbances and other unanticipated events, greater accuracy and repeatability, and, perhaps most importantly, increased duration [2]. Another opportunity afforded by supervisory control is multi-tasking. While the human operator may only be able to operate one remote machine manually, he may be able to supervise several, stepping in as required to solve problems the lower level computer had trouble with. The practicality of multi-tasking is discussed in [7].

1.3 EASE and the SSL loop group

The Experimental Assembly of Structures in EVA, or EASE, was a NASA sponsored Shuttle flight experiment to study the capabilities and limitations of humans in EVA (Extra-Vehicular Activity). Productivity of humans in space was considered key in determining the economic viability of any space system requiring assembly. Suited subjects assembled the EASE structure in neutral buoyancy at the NASA Marshall Space Flight Center Neutral Buoyancy Simulator (MSFC-NBS) and in EVA on STS flight 61-B in November and December 1985. Data returned from this experiment has already had some impact on space station design. Detailed results are found in [3].

A secondary objective of the EASE experiment was to verify neutral buoyancy as a simulation of sustained zero gravity. Preliminary studies concluded that neutral buoyancy was an accurate simulation for task involving manipulation of masses sufficiently large that inertia dominated water drag. Damping due to water drag was expected to facilitate alignment and body positioning in neutral buoyancy but not expected to affect task completion time. Times for full assembly in EVA were found to be 18% faster than those in neutral buoyancy. EVA and neutral buoyancy assembly repetitions showed similar learning and fatigue curves, though the onset of fatigue appeared earlier in neutral buoyancy. The overall conclusion was that, for the construction of the EASE structure,

there is a good correlation between working in space and simulating it under water in neutrally buoyant conditions.

The EASE structure is a tetrahedron with six beams and three nodes assembled in the space shuttle payload bay (see figure 1.1). One of the nodes, or clusters, is fixed to the payload bay. The clusters and beams are connected by the EASE joint (see figure 1.2). The cluster has a mushroom end which fits into a receptacle on the end of the beam. A sleeve on the beam end is then slid over the joint to hold it in place. The sleeve is held in place by a spring loaded button. The capture rocker shown in figure 1.2 was not used in the human assembly version of the EASE joint.

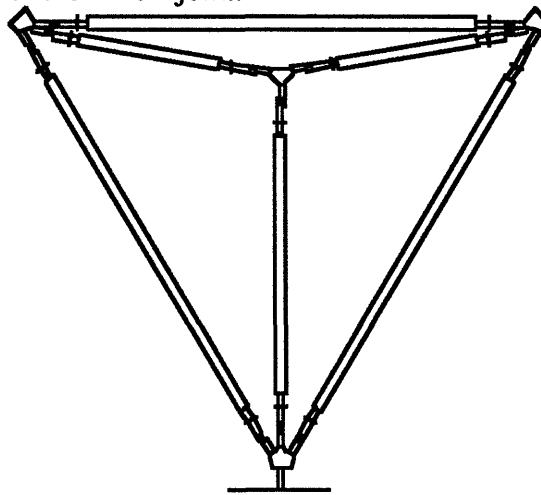


Figure 1.1 The EASE tetrahedron

The EASE tetrahedron was designed to be generic in nature rather than an operational prototype. The structure was designed to be challenging to assemble in order to assess learning, productivity and fatigue. As such, the allowable misalignment of the two ends to be joined is less than $\pm 2-3$ degrees for a successful joint completion.

The first teleoperator developed by the Space Systems Laboratory, the Beam Assembly Teleoperator or BAT, was designed to assemble the EASE structure to compare teleoperated assembly with human assembly. Developmental testing for BAT is performed at M.I.T.'s Alumni Swimming Pool. Full scale testing has been conducted at MSFC-

NBS, which is a cylindrical tank of water 23m across and 12m deep. Across the bottom of the tank is a mock-up of the space shuttle payload bay.

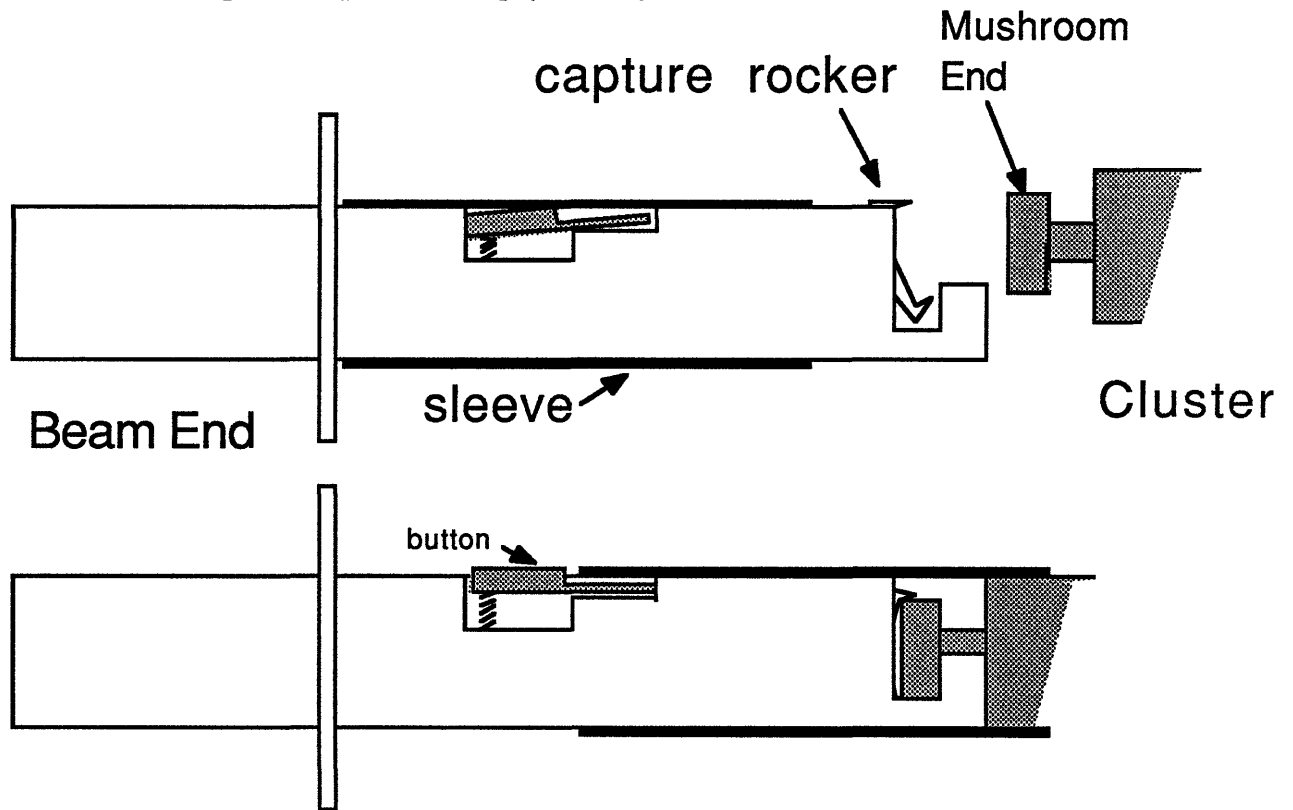


Figure 1.2 The EASE joint, unassembled and assembled

The EASE structure used by BAT is much the same as that used by human subjects in pressure suits with the addition of a capture latch, or rocker, to hold the mushroom in place while the sleeve is slid. BAT does not use a pair of "harpoons" used by the suited subjects to tether beams and clusters during assembly.

1.4 The vehicle, BAT

BAT is a neutral buoyancy simulation of a space based teleoperator. BAT's base is maneuverable in 6D use of 8 ducted propellers (see figure 1.3). Unlike vehicles designed specifically for underwater use, ca attention is paid in neutral buoyancy to placing the center of mass of the vehicle coincident with the center of buoyancy, so that BAT has no preferred attitude. This attitude freedom provides a

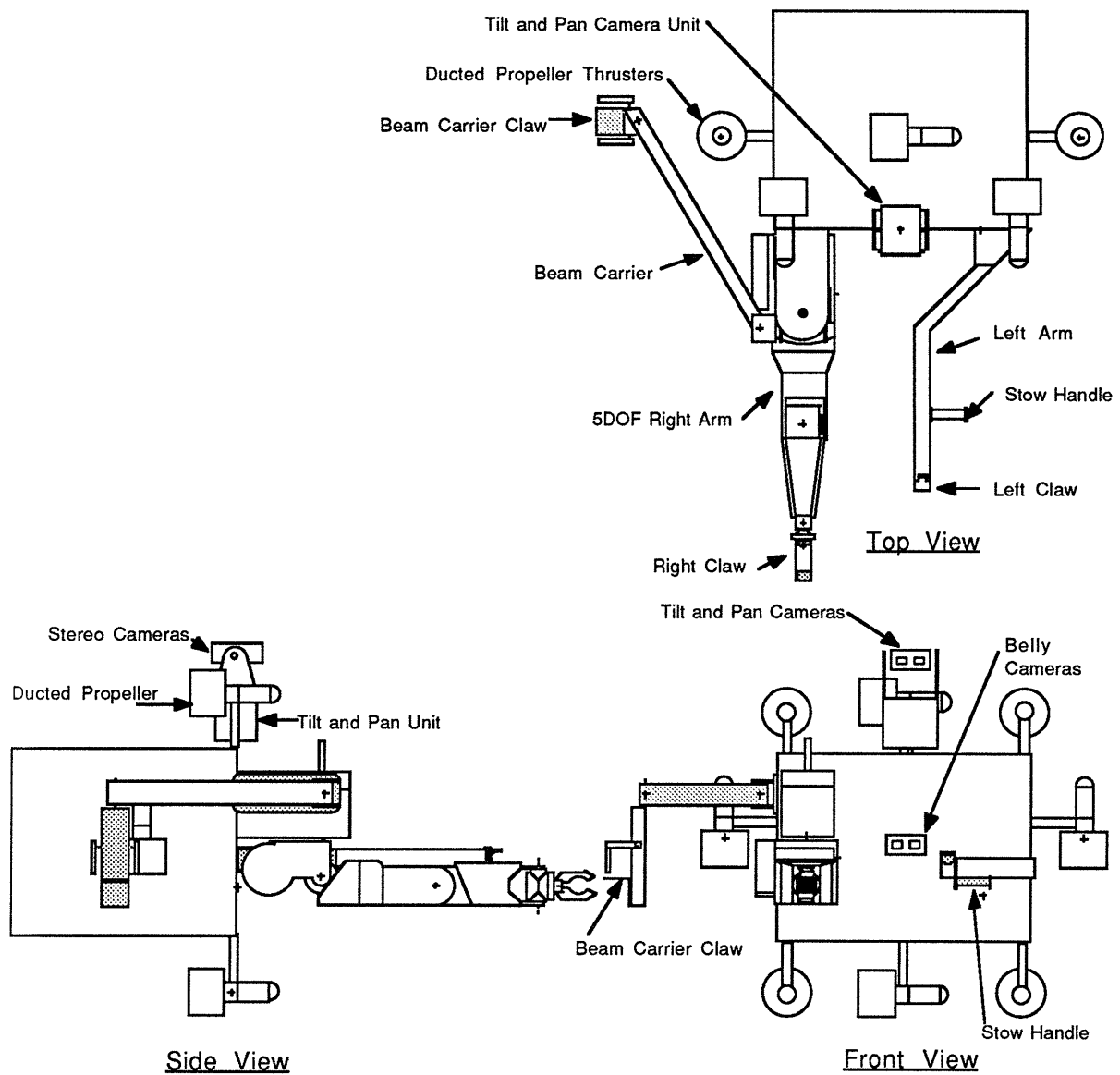


Figure 1.3 BAT

simulation of the rotational dynamics of a space vehicle, allowing BAT to 'fly' underwater.

BAT has one dextrous 5DOF arm which is proportionally similar to a human arm (see figure 1.4). The arm consists of a 2DOF differential wrist, a 1DOF elbow and a 2DOF shoulder. The motors for the upper three joints are held close to the upper frame origin to keep down inertia. Since the EASE joint has an axis of symmetry, the wrist did not need to have a degree of freedom in pitch. Each joint is also equipped with an angular position encoder.

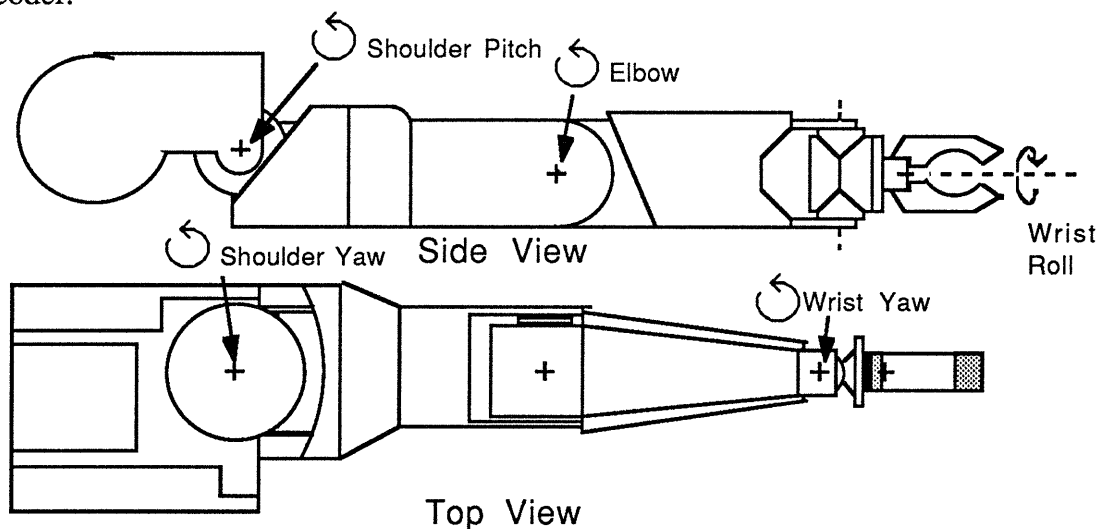


Figure 1.4 BAT's right arm

BAT has two other arms. The left arm is used for grappling, and includes a stowage handle for the right arm claw. The stowage handle is used to hold the right arm in place by its claw when the arm joint motors are not powered. The beam carrier arm is used to remove a beam from the beam rack, hold it back while BAT is flying, and then present it for assembly. The claws for all three arms as well as the extension and retraction of the beam carrier are controlled with pneumatic air cylinders.

There are two sets of stereo cameras on BAT for operator feedback. The cameras are mounted in pairs to allow the operator to have depth perception. One stereo pair is mounted on BAT's belly in the center of the vehicle. This pair has wide angle lenses and is

used for flying. The other pair has a more narrow field of view and is mounted on a tilt and pan base. The tilt and pan allows the camera to be commanded in 2DOF. The motion of the tilt and pan is normally slaved to the motion of the operator's head to allow him to look around. The magnification and aiming ability makes the tilt and pan cameras applicable to assembly tasks.

The motors, encoders, batteries and circuitry on BAT are enclosed either in water tight boxes or in waterproof potting. Most boxes are pressurized so that any leaks that occur will leak air out rather than water in. The battery boxes are not pressurized. Since batteries vent hydrogen as they are discharged, pressurization of the battery boxes would create a danger of ignition if hydrogen from the battery box were allowed to mix with air. The main battery box is purged with nitrogen before and after use.

1.5 The control station, ICS

ICS, the Integrated Control System, is the operator station for controlling BAT. (see figure 1.5) The operator interface consists of a master arm used to control BAT's dextrous right arm, a head controller used to control the tilt and pan cameras, two 3DOF joysticks, one translation and one rotation, used to control BAT's propulsion system, six monitors showing computer graphics and whatever external video might be available, four switches mounted on the master arm, four switches mounted on the rotational joystick, four foot switches, four panel switches, six pneumatic enable switches, one large switch to change between stereo camera pairs, and a keyboard, used mainly in system initialization.

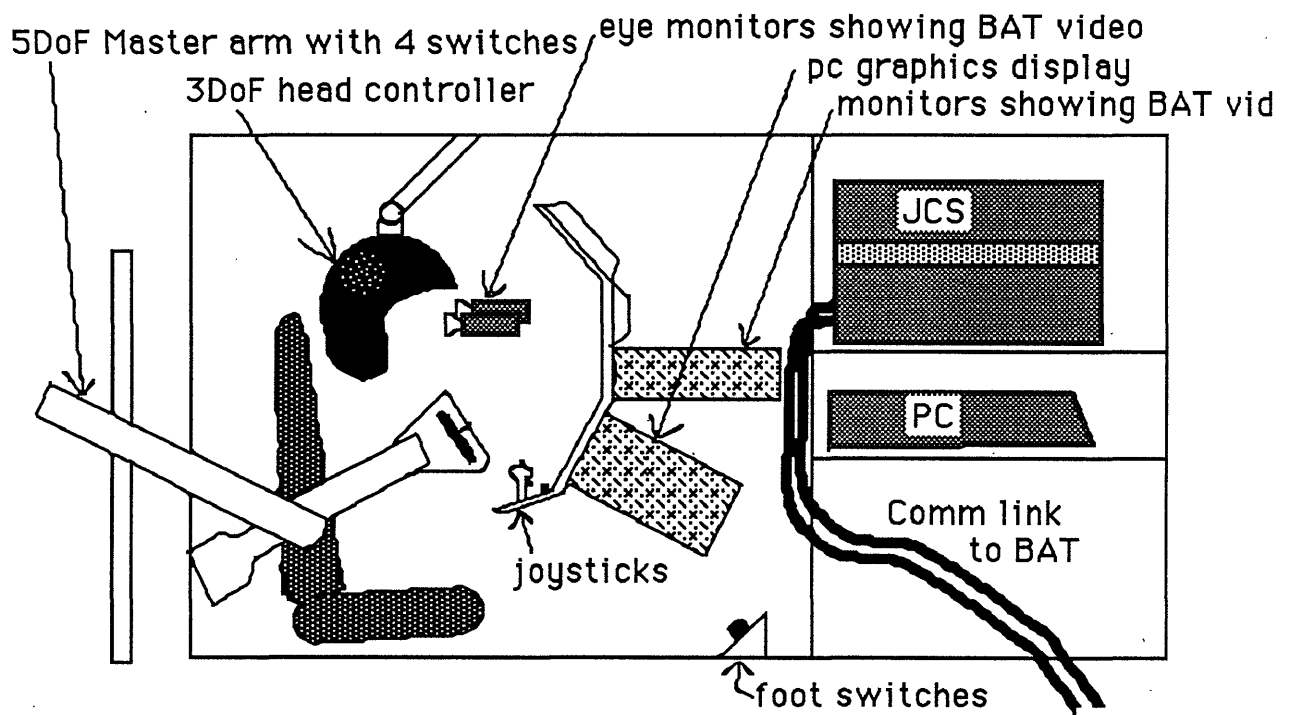


Figure 1.5 The Integrated Control Station



Figure 1.6 Photograph of the Integrated Control Station

The interface is monitored by an IBM PC running the program BATSOFT. BATSOFT reads the switches and the potentiometers on the arm and head controller, creates the graphic display, beeps the appropriate beeps, and gives commands to the Joint Control System (JCS), which controls the dextrous right arm, the pneumatics, the tilt and pan, and the propulsion system thrusters. The JCS contains four cards each driven by a Z80 microcomputer that calculates joint duty cycles, one Z80-based card that calculates thruster duty cycles, and a transceiver card which multiplexes all of the motor command signals together along with signals to control the state of the pneumatics, and sends it down the communications down link to BAT. The transceiver card also reads the up link from BAT containing mainly joint encoder positions for the joint cards and for BATSOFT.

Since the operator has primarily visual feedback with little oral and no tactile feedback, BAT and ICS are certainly not an example of telepresence. The operator interface, however, has been shown to be functional with the construction of the EASE structure detailed in [5] and chapter 5.

1.6 Triad

Triad is a set of task primitive supervisory control algorithms which use learned or established trajectories to move the dexterous arm through the motions of joint assembly. Triad reads in a command line from a file composed of angular positions for each joint plus commands to control the gains in the arm control system, the accuracy at which the trajectories are to be followed, and the state of the various pneumatics.

Triad temporarily replaces the master arm manual control. Triad works entirely in joint space, as the master arm does, providing the JCS with command set points for each joint of the arm.

Since the tilt and pan cameras are only for operator feedback and have no use to Triad, they remain in the control of the operator while Triad is running. In this way, the operator can monitor Triad.

1.7 Scope of thesis

Using TRIAD in addition to the already functional manual control system, the advantages of the addition of supervisory control are analyzed. Chapter 2 details upgrades made to the arm control system that allowed Triad to function. In chapter 3, the specifications for Triad and its application to BAT are listed. In chapter 4, the EASE joint is assembled repeatedly under manual and supervisory control situations for both experienced and naïve operators to discover the differences in productivity and its components in training and fatigue. In chapter 5, assembly of the full EASE tetrahedron is demonstrated and compared with a previous manual assembly.

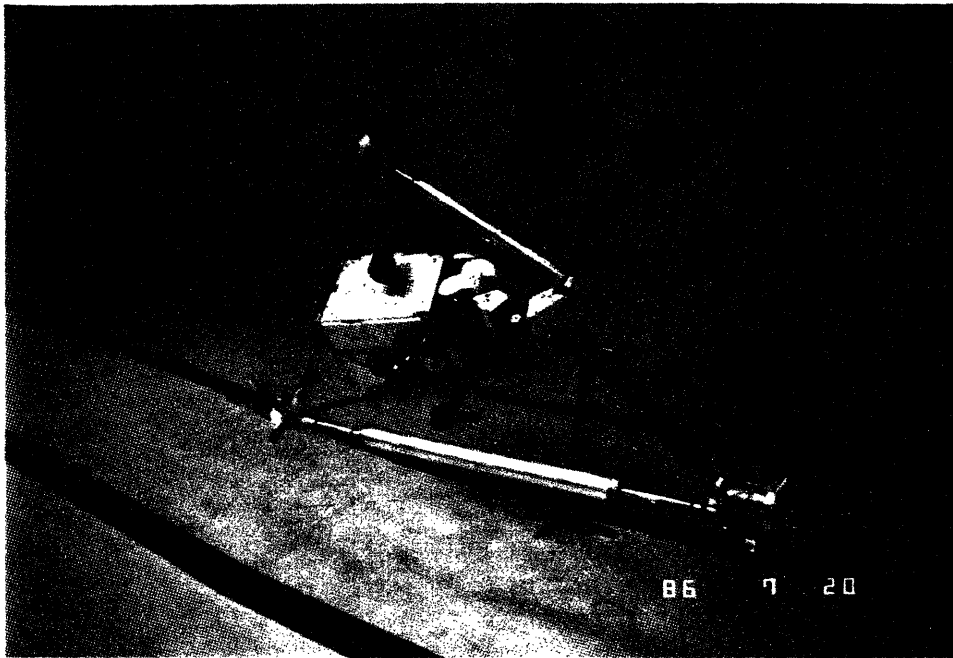


Figure 1.7 BAT assembles the mini-EASE tetrahedron in the M.I.T. Alumni Pool

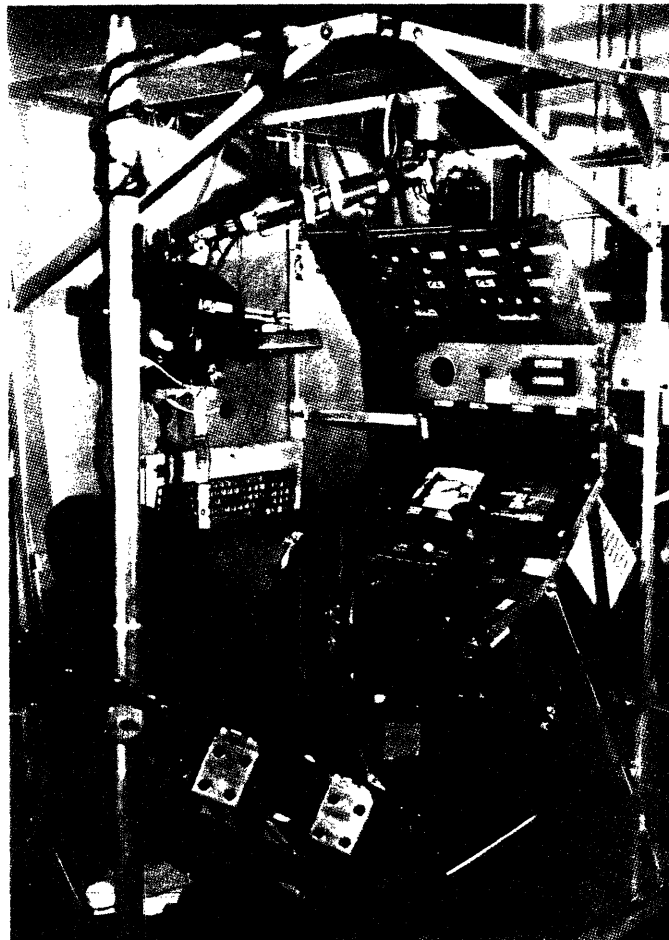


Figure 1.8 The Integrated Control Station

2. UPGRADED CONTROL ALGORITHMS 5DOF ARM

2.1 Previous control system

The prime objective for BAT was to assemble the EASE tetrahedron. This objective was accomplished in January 1987 at NASA Marshall Space Flight Center's Neutral Buoyancy Simulator during test series NB-67. The control of the 5DOF right arm was accomplished with a simple proportional or P-type control algorithm.

2.1.1 Chain of command to move the arm

The actuation of the right arm is accomplished by permanent magnet DC motors fed with a pulse width modulation (PWM) duty cycle. Gains for the P-type control system are input into BATSOFT which transmits them to the joint cards in the JCS via a dual port ram (DPR). Using these gains, a duty cycle is calculated for each joint by the joint cards and then multiplexed into a serial line for communication to BAT. On BAT, the serial data is demultiplexed and transmitted to power modules. The power modules contain H-bridge darlington transistors which reproduce the signal duty

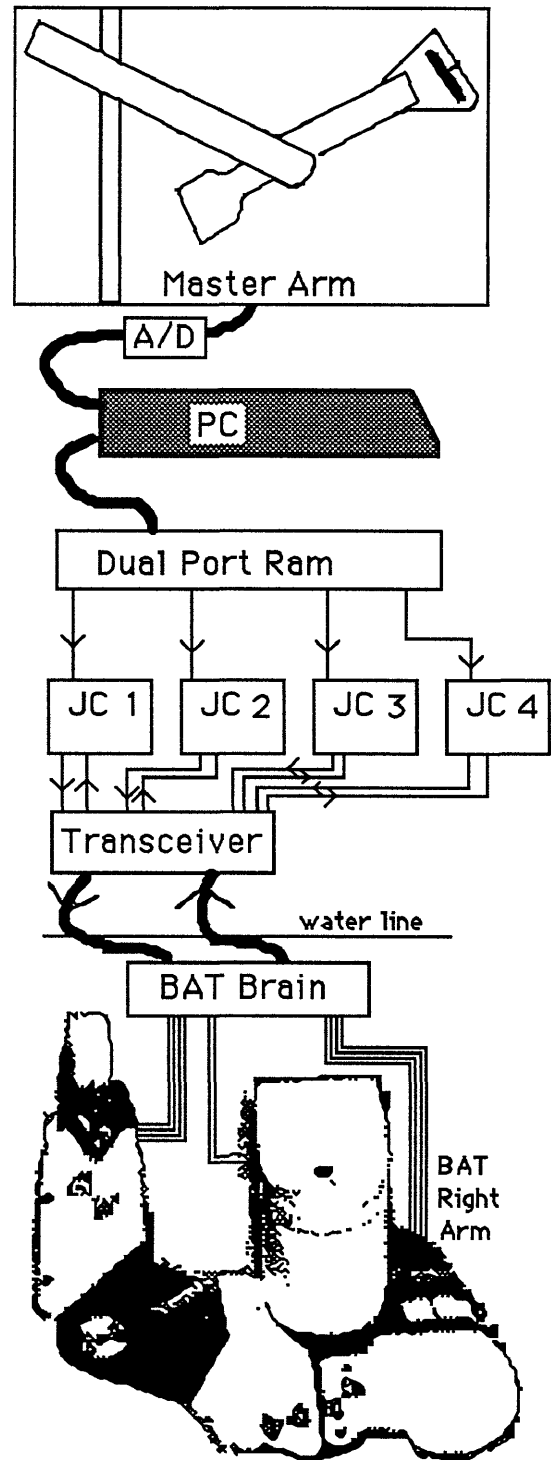


Figure 2.1 ICS-BAT Chain of Command

cycle at a current level capable of running the motors.

Each joint is also equipped with an optical incremental shaft angle encoder. Circuitry on BAT integrates the encoder quadratures and transmits the angular position of each joint back up the communications link to the JCS. In the JCS, the encoder positions are made available to the joint cards and to BATSOFT. Conversion of encoder value to joint angle in degrees is detailed in [4].

2.1.2 The role of BATSOFT

As mentioned above, the man-machine interface is monitored by an IBM PC running the C program, BATSOFT. BATSOFT reads the various potentiometers, switches and keys on ICS, and relays commands to the JCS. Code for BATSOFT can be found in Appendix II.

In the running of the right arm, BATSOFT sets the gains and provides command set points. The gains are set by the operator by loading prepared gain files or by adjusting each joint separately. The gains are then transmitted to the joint cards, via the DPR. When the operator turns on the master arm, BATSOFT reads the potentiometers at each joint of the master, adjusts them to produce a command set point for each joint, and transmits each set point to the appropriate joint card.

The tilt and pan is controlled in much the same way, with either the potentiometers on the head controller or those on a separate joystick being read for command set points.

As long as the master arm is connected, BATSOFT transmits a new position set point for each joint once per main loop. The frequency of the BATSOFT main loop is 8Hz.

2.1.3 The role of the joint cards

The frequency of the joint card main loop is 98Hz, an order of magnitude faster than the BATSOFT main loop. There are four joint cards resident in the JCS. Each calculates the duty cycle for two of BAT's motors. There are eight motors total, five in the right arm, two in the tilt and pan, and one for the cooling pump. The code for the joint cards is listed in Appendix III.

For the P-type control system, the joint cards read in the encoder position of a joint, subtract it from the command set point, and multiply by the constant gain supplied by BATSOFT. The result is then clipped to maximum duty cycle, if necessary, and sent to BAT in PWM form.

2.1.4 Problems with the previous control system

The P-type control law does not provide good tracking. Overcoming friction in the joints requires error integration. The P-type control law allows the arm to become stuck short of the mark, since its duty cycle diminishes as the actual joint position approaches the command set point. The operator is forced to account for this by overshooting a desired end point with the master arm and then coming back to stop the arm at the desired location. This is especially difficult for fine movements, in which the operator must use a large starting error to get the arm moving and come back just far enough to stop it again at the desired end point. The amount of leading required is variable with load and desired arm speed.

This problem is addressed here with the addition of integrated error and differentiated error to the control algorithm. The resulting controller is PID-type or proportional-integral-derivative type. The derivative term excites response in the applied torque due to quick changes in commanded speed. The integral term builds up applied torque in the presence of an error to get rid of steady state error and to overcome friction.

Another problem with the control system which is not addressed here is the frequency of the duty cycle used in the pulse width modulation. PWM is a method to approximate a DC voltage using a digital two-state signal. The premise of PWM theory is that the load has an electrical time constant sufficiently longer than the period of the PWM so that it sees the current it would otherwise receive from the corresponding DC voltage. Here, the PWM modulation has a frequency of 1KHz, but the motor response also about 1KHz. So, instead of the steady current draw expected by PWM, the motors on BAT draw current in

spikes in phase with the PWM duty cycle. The primary result of this problem is that there is no linear relationship between duty cycle and resultant motor torque.

A third problem with the existing control system is noise in the encoder signals. The wiring from the encoders in the present construction is laid along side the motor power cables for some distance. The highly oscillatory nature of the power current tends to induce noise in the adjacent, unshielded encoder cable. This is especially prevalent in the wrist, where an occasional twitch can be seen. The noise, together with nonlinearities produced by water currents, water drag, friction, stiction, and backlash makes a rather fuzzy picture of arm dynamics.

2.2 Digital PID control

2.2.1 Digital approximations of PID control

An advantage of a PID controller is that, when correctly tuned, it can give good system response without a resident model of the system dynamics. A standard continuous time PID-controller looks like:

$$U(s) = K \left(1 + \frac{1}{T_I s} + \frac{T_D s}{1 + T_D s/N} \right) E(s)$$

where $U(s)$ and $E(s)$ are Laplace transforms of control and error, K is the proportional gain, T_I is the integral time constant, T_D is the derivative time constant, and N is the number of sampling periods per rise time to a step input.

In order to implement a PID algorithm in the digital domain, the analog control law must be approximated. An extended discussion of digital approximations is found in [8]. There are three basic methods of approximation:

$$s = \frac{z-1}{h} \quad (\text{Euler Method})$$

$$s = \frac{z-1}{zh} \quad (\text{Backwards Difference})$$

$$s = \frac{2(z-1)}{h(z+1)} \quad (\text{Tustin Approximation})$$

where s is the Laplace operator, z is the forward shift operator such that $Z=e^{sh}$, and $z \times X_n = X_{n+1}$, where X_n is the value of X at $t = h \times n$, and h is the sampling period of the digital system.

According to [8], the most common way to approximate a PID controller is to make an Euler approximation of the integral part and a backwards difference approximation of the derivative part:

$$u(kh) = K \left(1 + \frac{h}{T_I(z-1)} + \frac{T_D}{h+T_D/N} \times \frac{z-1}{z-T_D/(Nh+T)} \right) e(kh)$$

where k is the sample number. For the purposes of tuning, the equation was streamlined to:

$$\frac{u(kh)}{e(kh)} = K_D \left(1 + \frac{h}{T_{ID}} \frac{1}{z-1} + \frac{T_{DD}}{h} \frac{z-1}{z+\gamma} \right)$$

where $T_{ID} = T_I$, $T_{DD} = T_D N$, and $\gamma = -e^{TN/T_D}$.

2.2.2 Tuning

With a short sampling period, a digital PID controller behaves much like an analog PID. Several tuning methods have been developed for the analog PID. Assuming a sufficiently short sampling period, these methods can be directly applied to the digital PID. The method used here is the transient-response method developed in [9].

The transient-response method models the system as a first order system with transportation lag. The step response to such a system is shown in figure 2.1.

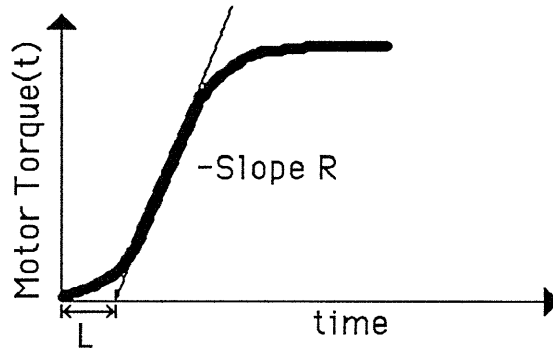


Figure 2.1 Open Loop Step Response of a System [8]

The lag and slope of the response can be used as a first approximation of PID parameters, using the following equations:

$$K_D = \frac{1.2}{RL} \quad T_{ID} = 2L \quad T_{DD} = 0.5L$$

Using these parameters, the controller can be tuned by trial and error for best response.

2.3 Application to BAT

2.3.1 Direct application

The first problem encountered is that listed in section 2.1.4, that of the failure of PWM for BAT. Instead of the smooth step response curve needed for the tuning, BAT's actual response is a series of spikes. The way chosen around this is to examine each of the spikes as a separate step response. The resulting values of N, the number of sampling periods per rise time, R, the torque response slope, and L, the torque response lag, are not guaranteed to provide the best starting point but were reasonably close.

Measuring the current output of each motor, which is proportional to output torque, values were found for R and L to construct a first iteration PID system for BAT's right arm. This was then implemented into the code running the joint cards. This code is listed in Appendix III.

The control law is arrived at as follows. Beginning with the equation above, solve for U_k , the control value for the present sampling period:

$$\frac{U(kh)}{E(kh)} = KS \left(1 + \frac{a}{(z-1)} + \frac{b(z-1)}{(z+c)} \right)$$

where $a = h/T_{ID}$, $b = T_{DD}/h$, $c = \gamma$, and S is the number of encoder pulses per joint angle degree. U here is the actual duty cycle sent to the arm, and E is the error between commanded and actual encoder positions.

$$\frac{U}{E} (1 + (c-1)z^{-1} - cz^{-2}) = KS (1+b) + KS(c+a-1-2b)z^{-1} + KS(ac-c+b)z^{-2}$$

$$U_k = (1-c)U_{k-1} + cU_{k-2} + KS(1+b)E_k + KS(a+c-1-2b)E_{k-1} + KS(ac-c+b)E_{k-2}$$

The coefficients of each of the control and error terms is calculated in BATSOFT and input to the JCS as a separate set of constants for each joint:

$$U_k = \{M_1 U_{k-1} + M_2 U_{k-2} + \alpha E_{k-1} + \beta E_{k-2}\} + G E_k$$

where the portion in the brackets can be precalculated at each step and added to the multiple of encoder error as soon as it is read.

Tuning of this algorithm failed to produce adequate tracking. Essentially, the integrator did not function correctly due to the slow sampling period. The use of a Tustin approximation instead of an Euler did not improve the response.

2.3.2 Addition of a separate integrator

A simple integrator was then added to the control law to force an integration of error to help zero out residual errors. The integrator simply sums a multiple of the error. The sum is then clipped, if necessary. The multiplier for each joint was determined by tuning for fastest response. The clipping level was determined from the duty cycle necessary to overcome starting friction. The integrator sum for each joint is reset every time the error passes through zero to avoid integrator windup.

This addition improved the tracking sufficiently to allow the use of Triad. Sets of parameters for each joint were then established for manual and Triad use.

3. TRIAD, TASK PRIMITIVE SUPERVISORY CONTROL

3.1 Theory

3.1.1 Supervisory Control

In the process of supervisory control, the operator issues commands to a computer which in turn closes its own loop with the vehicle being controlled. The computer produces a integrated status display to the operator to show response to these commands and the operator steps in when necessary to change commands or make adjustments.

Sheridan, in [2], goes into great detail concerning the roles of the computer and operator under supervisory control. He lists five major functions of the operator. The first is to plan the task to be done, decide a strategy for its completion including the appropriate roles of the operator and computer, and consider the initial conditions. The second is to teach the computer how to do the task, decide how the computer is to deal with unexpected occurrences, decide how far the computer is to go between commands, and give the command to begin. The third function is to monitor the computer, make minor adjustments when necessary, and diagnose failures. Fourth, the operator intervenes and takes manual control when the computer fails, in order to reconsider the plan and how it was taught to the computer. Last, the operator is to learn about the system in order to understand it and make best use of its characteristics.

3.1.2 Trajectory control and path planning

The object of planning trajectories is to move between points in an optimal way, primarily with regard to time. The path is constrained by its end points and obstacles that must be avoided. The trajectory, the time versus position sequence required to follow the path, is limited by joint torque maximums and allowable joint angles.

The output of a trajectory planning algorithm is a set of joint positions to be fed into the arm control system. The set points are typically required at a rate of 60Hz to 200Hz, depending on the control system, in order to provide a smooth arm motion as shown in [6].

The calculation of the set points may be accomplished by several methods. A set of intermediary or 'via' points can be added along the desired path to insure that the arm misses the obstacles in its path. The required arm velocities and accelerations at each point then become constraints to the path. The path can be calculated as an Nth order polynomial accounting for each of the constraints. But for even slightly complicated trajectories, the polynomial is of high enough order that checking for torque limit satisfaction becomes quite difficult. This type of trajectory is not optimal; due to the number of roots in the polynomial, it is likely to overshoot and wander.

A way around this problem is the use of knot points. Knot points are placed at the via points and where mathematically convenient. Given the arm state at each of the knot points, separate cubics are calculated to govern the motion from each knot point to the next.

A point of contention among the robotic community is whether these calculations should be conducted in cartesian space or joint space. A cartesian space trajectory is more easily visualized by the user of a system. As noted in Appendix I, it is difficult for a human operator to make a mental Jacobian transformation to move joints to go straight in a cartesian direction. But cartesian position and orientation are not enough for a trajectory planner. The inverse kinematics also require answers to questions such as whether the elbow should be up or down. The trajectory planner working in cartesian space has knowledge of end effector position but does not know the joint angle required to place the end effector. For this reason, it can assure obstacle avoidance for the end effector, but not necessarily for the upper portions of the arm.

3.2 Method

3.2.1 BAT's Limitations

Without major hardware changes, the application of supervisory control to BAT is limited. The propulsion system for the base is operated on a loop closed only through the visual system of the cameras and the human operator. Without any position sensing, BAT

has no way of knowing where it is inertially at any given time. BAT does know the state of its pneumatics and the encoder position of the right arm. With this information, BATSOFT could close a loop with the JCS to control the position of the right arm and the state of the pneumatics.

As noted in chapter 2, BAT has no exact control of torque. Joint velocity and acceleration are also not available. These limitations make a generalized trajectory planner impractical. But since the task primitives of the EASE joint assembly were repetitive, specialized trajectories could be developed for each portion.

3.2.2 Application to BAT

As stated in the introduction, Triad temporarily replaces the master arm control. Triad works entirely in joint space, as the master arm does, providing the JCS with command set points for the arm.

BATSOFT, with all its analog to digital conversions reading the potentiometers of the master arm, head controller and joysticks, has a main loop period of 8Hz. Triad, without the burden of the full human interface, can run much faster, approximately 1KHz. The JCS runs at 98Hz. This allows Triad to monitor each sampling period of the JCS.

Triad operates using a file of saved joint positions and commands. A command file can be generated during main loop manual control in Triad's learn mode. Triad saves the actual arm position to file once every main loop. The file can then be "played back" using triad's point-to-point mode, or modified off-line to smooth out the motion and get rid of errors. Files can be played back singly or concatenated together for one long trajectory. A stack of files can be set up for playback via a menu or a button software configured to load in a set of trajectories.

Triad's learn mode saves the actual arm position, as opposed to the command position, for two reasons. As will be shown in section 4.1, the operator often provides a higher order control system to pick up where the JCS leaves off. The commanded position may contain lead and dither not meant to appear in the actual arm position. The second reason is

that the Triad arm gains are higher than manual gains, and response to a command given using manual gains is different than that using Triad gains. Triad gains are tuned to provide the fastest possible tracking.

Manual gains are limited to allow for smooth arm motion under the control of the main loop of BATSOFT. The manual gains are sufficient to provide tracking of a single command set point in approximately one main loop of BATSOFT. If the tracking were significantly faster, the arm would arrive at the set point before a new set point could be supplied by BATSOFT, and the arm would slow toward a stop. The result would be discontinuous slave arm velocity in response to a constant master arm velocity.

After Triad has issued a set point to the JCS, it monitors joint positions and waits for the actual joint space position of the arm to come within a target region of the commanded position. The size of the target region is set by the command file being read, but is usually kept small near a target and large during a free move through open space. Triad also has control over the JCS control laws for each joint. If the arm takes too long to track to a new set point, Triad loads in higher control gains for each joint to get the arm going again. Friction is a large factor for BAT's right arm due to the fact that the arm is operated immersed in water, and is therefore difficult to lubricate. Once the actual position reaches inside the target region, Triad reads in a new command from the file, passes on any pneumatic changes in the command line, and gives the JCS a new set point.

Once the file or set of files is complete, Triad loads manual control gains to the arm if it is still in action, or loads zero gains to the arm if it was stowed, and returns control to the human interface. If the operator wishes to bail out of Triad in the event of an error or a change in plans, he can pull a finger switch. Triad responds immediately by loading in manual control gains, changing the master arm offsets so that the present master arm position is aligned with the arm position, and returning to main loop manual control.

Since the tilt and pan cameras are only for operator feedback and have no use to Triad, they remain in the control of the operator while Triad is running. In this way, the operator can monitor Triad's progress.

3.2.3 Triad trajectory development

Each trajectory developed for Triad started with points in joint space. Starting and ending points were taken as well as via points clear on approach to grappling positions, and clear of any obstacles. A path was then plotted between these points using linear joint interpolation, or a straight lines in joint space. Spacing between set points was determined by the tuning detailed in chapter 2. Commands were then added to the Triad file to change pneumatic states at the appropriate set points.

Commands to change arm gains were added. Three sets of gains are available: Triad 1, 2 and 3. Triad 1, the default, was used in guarded situations on approach to a grappling point. Triad 2, with slightly higher gains, was used in free portions of the trajectory when the arm was swinging across free space. Triad 3, with high gains, was used for arm movements with a load in the right claw.

In situations where a particularly large torque was required, such as getting a load started moving, a single set point was modified and moved beyond the desired position. The actual desired position remained within the target region of the modified set point. This would supply the large error to the joint cards to provide the torque required for the job without waiting for the integrator to build up. The load would then respond to the jolt, allowing the arm to move toward the desired set point. Once the actual position was within the target region of the modified set point, by design quite close to the desired set point, Triad would go on to the next set point. The next set point could then be back on the nominal path.

3.3 Application to the EASE structure assembly

3.3.1 Breakdown of EASE joint assembly

The assembly of the EASE joint used in the comparison of manual and supervisory control assembly is the configuration with BAT docked with its left claw to a cluster. A beam in the beam carrier is attached to the cluster. The assembly, called a beam joint assembly, consists of five steps:

1. Grab the beam. With a beam already in the retracted beam carrier claw, the beam is swung forward. The right arm is powered up and the right claw open. The right arm moves clear of the stow handle mounted on the side of the left arm, and moves clear of the left arm. The right arm waits for the beam carrier to extend and settle the beam to a stop and then moves up to close the right claw on the beam end. The beam carrier claw is then opened and the beam carrier swung back.
2. Make the joint. With the beam held in the right claw, the arm maneuvers the beam to place the cluster mushroom end into the receptacle on the beam end. The wrist must then be twisted to apply sufficient force to set the rocker in the beam end so that it holds the mushroom end in place.
3. Grab the sleeve. With the joint just completed, the right claw opens on the beam end, the arm backs off of the beam end, approaches the sleeve, and grabs the sleeve with the right claw.
4. Slide the sleeve. With the right claw closed on the sleeve, the right arm applies sufficient force to slide the sleeve over the cluster, holding the mushroom end in place. The sleeve is slid until the spring loaded button pops up to hold the sleeve in place.
5. Stow the right arm. With the right claw on the slid sleeve, the claw opens. The right arm backs the claw off of the sleeve and clear of the left arm. The arm approaches the stow handle so that the right claw can be closed on the stow handle. The gains to the right arm are then zeroed.

3.3.2 Assembling a joint manually

Manual assembly of the joint involves actuating switches to change pneumatic states. The arm movements are commanded with the master arm. Each time the master arm is brought into use, the operator hits a software switch to set the present master arm position equal to the present slave arm position. This process is called 'connecting' the master arm. When the master arm position becomes uncomfortable for the operator, he may disconnect the master arm and reconnect it in a more comfortable position. The arm is powered up and down with a button push. The operator uses the tilt and pan cameras slaved to the head controller to view the scene.

3.3.3 Assembling a joint with Triad

For each of the basic parts of the EASE joint assembly there is a Triad file. All five steps are reduced to button pushes with the exception of setting the rocker on the mushroom end. In the manual assembly, getting the rocker to set on the mushroom end requires wiggling the beam end until the rocker receives enough force to set. Due to problems in commanding torques, as discussed in section 3.2.1, together with inexact positioning of the cluster in the left claw, Triad only goes so far in making the joint as to place the beam end close to the mushroom end for easier manual completion.

For the purposes of assembling the EASE joint with Triad, the Triad trajectories are grouped. The operator hits a switch actuating GRABBEAM and JOINT trajectories, and watches the right arm grab the beam and move it close to the cluster. The operator manually gets the rocker to set, and hits the switch again. Triad runs GRABSLV and SLIDESLV. Once the operator sees the button pop on the sleeve, he hits the switch a third time. Triad runs STOW. During the process, if a Triad trajectory fails, the operator completes the failed task primitive manually, and restarts Triad at the beginning of the next task primitive.

4. MANUAL vs. TRIAD ASSEMBLY of the EASE JOINT

4.1 Manual assembly using P-type control law

In January of 1987 during test NB-67, BAT successfully completed the EASE structure at MSFC-NBS using manual control and a proportional (P-type) control law on the arm. The P-type control law forced the operator to use such tricks as supplying command set points beyond the desired location in order to overcome friction. Since assembly of the EASE joint is repetitive, the operator could learn the necessary master arm motions to assemble the joint.

During the testing, full size beams approximately 4m long were used. In the swimming pool facilities available at M.I.T., full size beams are difficult to accommodate, so mini-beams, approximately 1.5m long, are used instead. The mini-beams have the same beam ends as the full beams. The difference in joint assembly is contained in the setting of the rocker, where BAT must move the full beam. In this instance, the inertia of the full beams made them more difficult to maneuver.

4.2 Manual assembly using PD-type control law

For the purposes of comparing manual and Triad joint assembly, each operator completed several joints in succession. BAT was mounted to a frame at the bottom of the pool. Before each run, a diver would place a beam in the retracted beam carrier. The operator would then power up the arm, extend the beam carrier, and run through the five steps of assembly detailed in chapter three.

Data taking consisted of video of the operator's eye view through the BAT cameras, and a record of commanded and actual arm position and duty cycle. As soon as the data from one run was stored, the operator would start the next assembly. Data storage took approximately one minute per run.

There were three operators used in the testing. The experienced operator already had more than 25 hours on the arm as the primary operator during development and the NB-67

testing. His experience was exclusively with the P-type manual control system. The less experienced operator had only two hours of arm time at the start of the testing, but had more knowledge of the new control system and of Triad. The third operator was naïve, having no operational experience with the arm, and only basic training in its use.

The first testing was performed using a PD control system before the integrator detailed in chapter 2 was implemented. Assembly times are listed in table 4.1.

Subject:	Experienced PD-Control						seconds
Run #:	1	2	3	4	5	6	Average:
Grab Beam	15.0	10.0	15.0	13.0	11.0	9.0	12.2
Joint	49.0	29.0	30.0	19.0	25.0	28.0	30.0
Grab Sleeve	7.0	9.0	5.0	7.0	9.0	7.0	7.3
Slide Sleeve	8.0	5.0	5.0	3.0	4.0	4.0	4.8
Stow	17.0	28.0	17.0	10.0	16.0	9.0	16.2
Total Time	96.0	81.0	72.0	52.0	65.0	57.0	70.5

Subject:	Less Experienced PD-Control						seconds
Run #:	1	2	3	4	5	6	Average:
Grab Beam	19.0	84.0	39.0	23.0	30.0	14.0	34.8
Joint	37.0	96.0	50.0	7.0	30.0	13.0	38.8
Grab Sleeve	10.0	6.0	11.0	7.0	5.0	17.0	9.3
Slide Sleeve	10.0	26.0	46.0	5.0	5.0	8.0	16.7
Stow	20.0	26.0	10.0	15.0	13.0	14.0	16.3
Total Time	96.0	238.0	156.0	57.0	83.0	66.0	116.0

Subject:	Naïve PD-Control				Manual
Run #:	1	2	3	4	Average:
Grab Beam	215.0	150.0	133.0	116.0	153.5
Joint	568.0	90.0	107.0	57.0	205.5
Grab Sleeve	18.0	40.0	32.0	24.0	28.5
Slide Sleeve	29.0	27.0	40.0	25.0	30.3
Stow	48.0	54.0	16.0	21.0	34.8
Total Time	878.0	361.0	328.0	243.0	452.5

Table 4.1 Manual PD assembly times

The experienced operator had the shortest times, exhibiting learning through the first four runs. Run four for the experienced operator was the fastest manual assembly time encountered in the entire test. Since the addition of a differentiator to the control law does not significantly decrease steady state error, the experienced operator should continue to benefit from his experience with the P-type controller. This is quite evident in the graph of

command versus arm position. Figure 4.1 shows the shoulder pitch trace as an example. All the way through the trajectory, the operator's command leads position, overshoots to keep the arm moving, then comes back to stop the arm motion. The actual arm position is quite smooth, and as was already noted, quite fast. The commanded position is much more oscillatory, especially near the end as the arm approaches the stow handle. In this way, the operator has added his own lead to the system to compensate for the inherent lag in the PD controller.

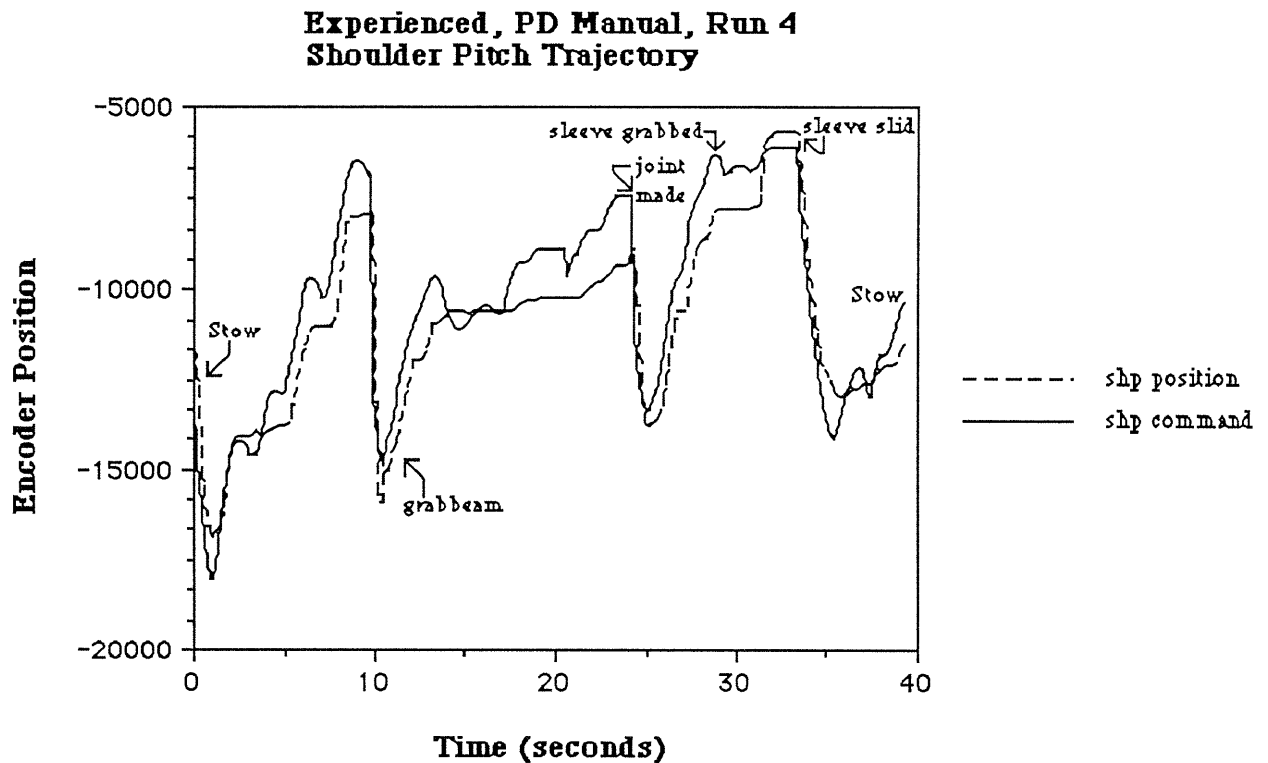


Figure 4.1 Example of Operator Addition of Higher Order Control

Traces from the less experienced and naïve operators do not show this kind of oscillation. The commanded position contains the same sort of oscillation, the larger command to move the arm, followed by a pull back. But in these cases, the pull back is not as well timed. The pull back is apparently actuated by the operator's view of the arm overshooting the desired position. This leads to actual arm motion that is much more oscillatory, and thereby slower.

The less experienced operator shows a much more erratic series of assembly times. This is caused by a series of operator errors such as missing the mushroom with the beam end and trouble aligning the claw with the sleeve while sliding it. Here the largest variation is in the completion of the joint. This task requires the application of the correct force to set the rocker, and the arm is not set up for force control. The result is that the operator must know the correct master arm motions to apply forces in the correct directions. Without any force feedback, this can be difficult.

The naïve operator has times much higher than the other two operators. His times do show a learning curve in overall time, though not consistently in each component of the assembly. Due to operator fatigue, the naïve operator only completed four runs before having to stop.

4.3 Manual assembly using PID-type control law

During a separate test, the experienced and the less experienced operators repeated the testing using PID arm control. The integrator added the feature of zero steady state error, balancing out the lag in the system. Assembly times are listed in table 4.2.

Subject: Experienced	PID-Control		Manual		seconds		
Run #:	1	2	3	4	5	6	Average:
Grab Beam	24.0	20.0	22.0	19.0	17.0	16.0	19.7
Joint	60.0	37.0	44.0	104.0	46.0	66.0	59.5
Grab Sleeve	12.0	11.0	13.0	18.0	9.0	8.0	11.8
Slide Sleeve	19.0	24.0	8.0	10.0	16.0	6.0	13.8
Stow	21.0	29.0	19.0	32.0	19.0	18.0	23.0
Total Time	136.0	121.0	106.0	183.0	107.0	114.0	127.8

Subject: Less Experienced	PID-Control		Manual		seconds		
Run #:	1	2	3	4	5	6	Average:
Grab Beam	22.0	18.0	19.0	22.0	36.0	27.0	24.0
Joint	86.0	24.0	53.0	24.0	14.0	8.0	34.8
Grab Sleeve	12.0	9.0	7.0	11.0	11.0	10.0	10.0
Slide Sleeve	15.0	30.0	5.0	4.0	4.0	3.0	10.2
Stow	20.0	16.0	17.0	12.0	16.0	19.0	16.7
Total Time	155.0	97.0	101.0	73.0	81.0	67.0	95.7

Table 4.2 Manual PID assembly times

The experienced operator shows an 80% increase in average assembly time. There is no clear learning curve. It is possible that the experienced operator had to unlearn the portion of the control law he was used to providing to the P-type controller.

The less experienced operator shows a 20% improvement in average assembly time. This is the expected result, considering that the arm response is faster and more accurate here than under PD control.

4.4 Triad assembly

4.4.1 The procedure

The testing was repeated using Triad to replace most of the manual portions of the assembly. The function of the operator through the Triad portions of the assembly was as an observer. On one occasion when Triad failed to grab the beam, the operator stepped in to complete the task manually.

With a beam ready in the beam carrier and the right arm on the stow handle, the operator began by pushing a button. Triad would extend the beam carrier, grab the beam with the right arm, release the beam from the beam carrier and move it close to the mushroom end on the cluster. The operator then moved the beam manually to insert the connector and set the rocker on the mushroom end. The operator next pushed the same button to get Triad to grab and slide the sleeve over the mushroom end. Verifying the sleeve to be slid, the operator would push the button a third time to get Triad to stow the arm.

The assembly was completed six times in succession. Operator's eye view was video taped and a trace of the arm command and positions was recorded. This led to the same one minute delay between runs for data storage.

4.4.2 Results

The same less experienced and naïve subjects were used for testing. The experienced operator was not available for this portion. The times for the Triad runs are listed in table 4.3.

Subject:	Less Experienced		Triad					seconds
Run #:	1	2	3	4	5	6	Average:	
Grab Beam	7.4	7.6	7.7	7.5	8.2	8.0	7.7	
Joint	31.1	27.9	27.9	47.9	17.1	82.6	39.1	
Grab Sleeve	6.6	5.5	5.1	3.0	5.9	5.4	5.2	
Slide Sleeve	1.7	5.5	1.5	7.6	11.2	1.4	4.8	
Stow	6.1	7.7	8.3	9.3	7.6	7.3	7.7	
Total Time	52.8	54.1	50.4	75.4	49.9	104.7	64.6	

Subject:	Naïve Triad							seconds
Run #:	1	2	3	4	5	6	Average:	
Grab Beam	8.4	7.6	26.1	8.3	8.6	8.1	11.2	
Joint	97.6	88.8	120.5	31.8	50.4	28.4	69.6	
Grab Sleeve	8.3	7.6	6.1	3.8	5.6	6.9	6.4	
Slide Sleeve	1.6	1.4	2.0	3.5	1.7	1.6	2.0	
Stow	6.5	7.9	7.8	6.2	6.6	6.5	6.9	
Total Time	122.4	113.3	162.5	53.7	72.8	51.5	96.0	

Table 4.3 Triad assembly times

The less experienced operator was one third faster than his manual PID average; half of the runs equal or exceed the experienced operator's record manual assembly time of 52 seconds. The times for the portions of the assembly competed by Triad are consistent. During runs 4 and 5, Triad failed to slide the sleeve all the way. The operator completed the task with little loss in time. The only real variation comes from the manual portion of the assembly, the completion of the joint with the operator showing fatigue in the final run.

The naïve operator shows an 80% decrease in average assembly time, including one time faster than the record manual assembly. Again the variability is contained mainly in the manual joint portion. Grabbeam failed in run 3. The operator recovered without a great loss in time.

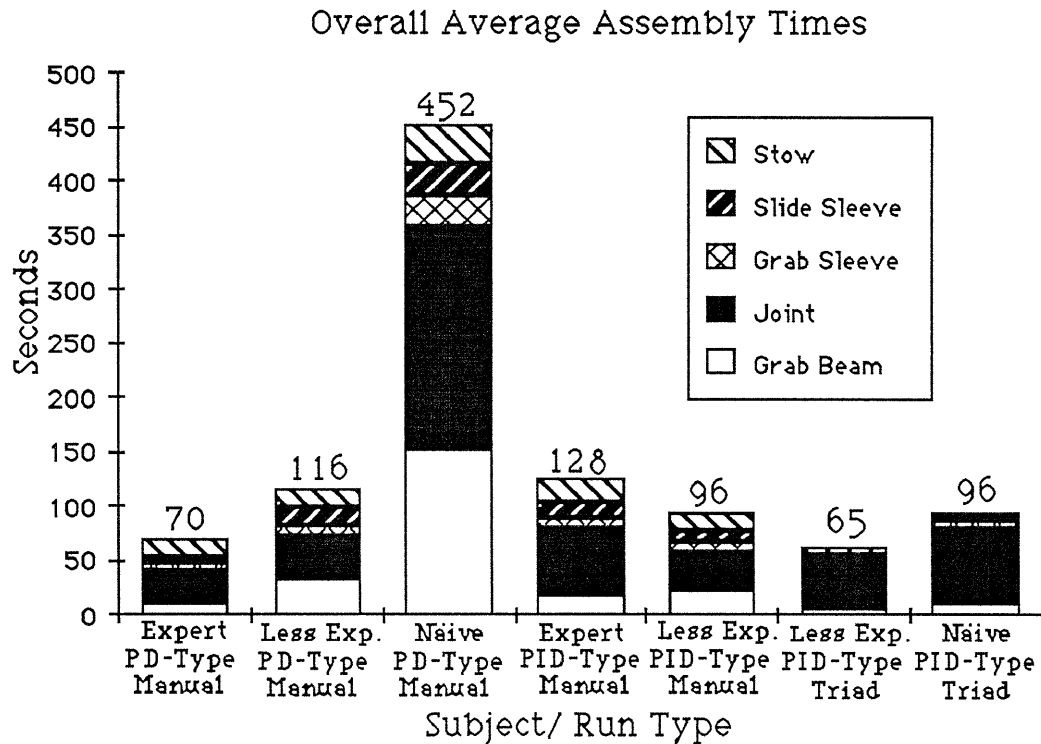


Figure 4.2 Overall Assembly Times

4.5 Comparing manual and Triad assemblies

4.5.1 Overall time

Table 4.4 shows the average assembly times of all the test runs. As stated earlier, the Triad times show a distinct improvement for the operator, comparable to the manual times of the next more experienced operator. Figure 4.2 shows the same information in graphical form.

Subject	Experienced	L. Exper.	Näive	Experienced	L. Exper.	L. Exper.	Näive
Control	Manual PD	Manual PD	Manual PD	Manual PID	Manual PID	Triad	Triad
Grab Beam	12.2	34.8	153.5	19.7	24.0	7.7	11.2
Joint	30.0	38.8	205.5	59.5	34.8	39.1	69.6
Grab Sleeve	7.3	9.3	28.5	11.8	10.0	5.2	6.4
Slide Sleeve	4.8	16.7	30.3	13.8	10.2	4.8	2.0
Stow	16.2	16.3	34.8	23.0	16.7	7.7	6.9
Total Time	70.5	116.0	452.5	127.8	95.7	64.6	96.0

Table 4.4 Overall Assembly Times

4.5.2 Joint completion time

As stated above, since Triad does not replace the entire joint-making portion of the assembly, the times for this portion of the assembly remain variable for the Triad runs. The task is still the same. The operator must move the master arm to build up a command position error on the slave arm to apply force on the beam sufficient to set the rocker. The amount of the force is not always the same due to the differences between rockers, the position of the right arm grip on the beam end, and the position of the cluster in the left arm.

For the less experienced operator, the joint completion portion is not significantly faster than for manual completion. But for the naïve operator, the time for joint completion is much improved, two thirds faster than his manual time. This is perhaps due to the lowering of workload on the operator. The naïve operator can now concentrate on learning and performing the joint completion. The lower workload and the faster completion times also lead to lower fatigue levels for the naïve operator allowing for better performance.

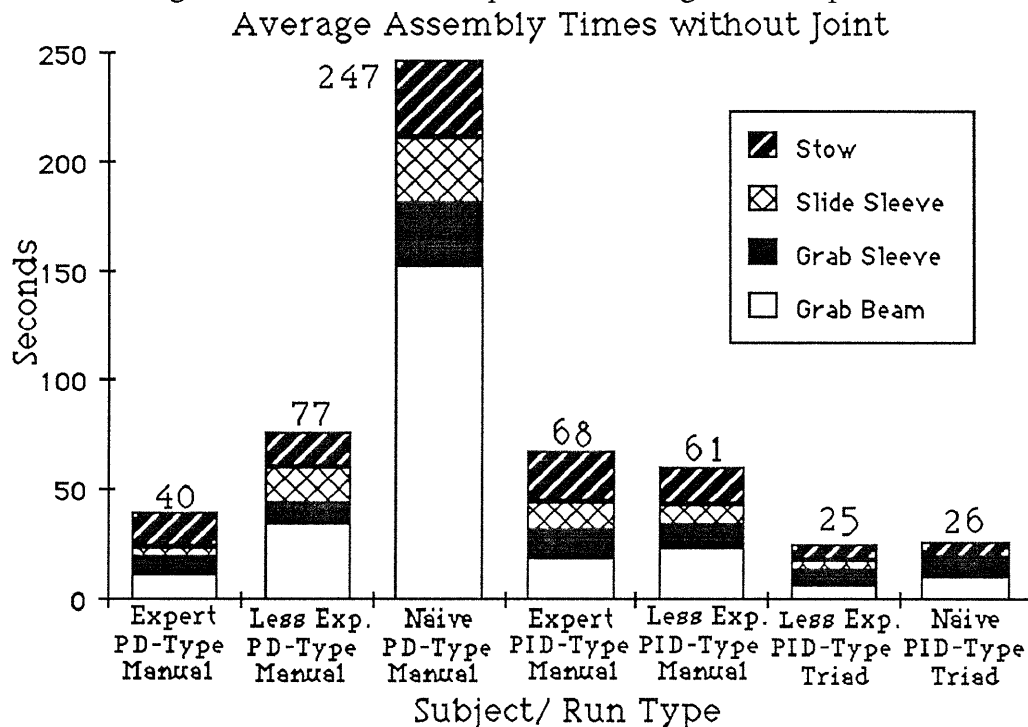


Figure 4.3 Overall assembly times showing Triad replaced portions only

4.5.3 Tasks covered by Triad

Figure 4.3 shows the overall assembly times only for the portions of the assembly replaced by Triad. There are two major points here.

- First, the decrease in assembly times are concentrated in these four portions of the assembly, so the differenced are more pronounced.
- Second, the Triad times for the less experienced and naïve operators is nearly the same, verifying that Triad assembly does not depend on experience.

5. FULL EASE STRUCTURE ASSEMBLY

5.1 Results of NB-67

5.1.1 Test setup and procedure

In January of 1987, during test NB-67, BAT successfully completed the EASE structure at MSFC-NBS using manual control and a P-type control law on the arm. This was the first full scale demonstration of teleoperated structural assembly in zero gravity. The intent of this demonstration was to generate data to be used in trade offs between using teleoperators and humans in space. The testing is detailed in [5].

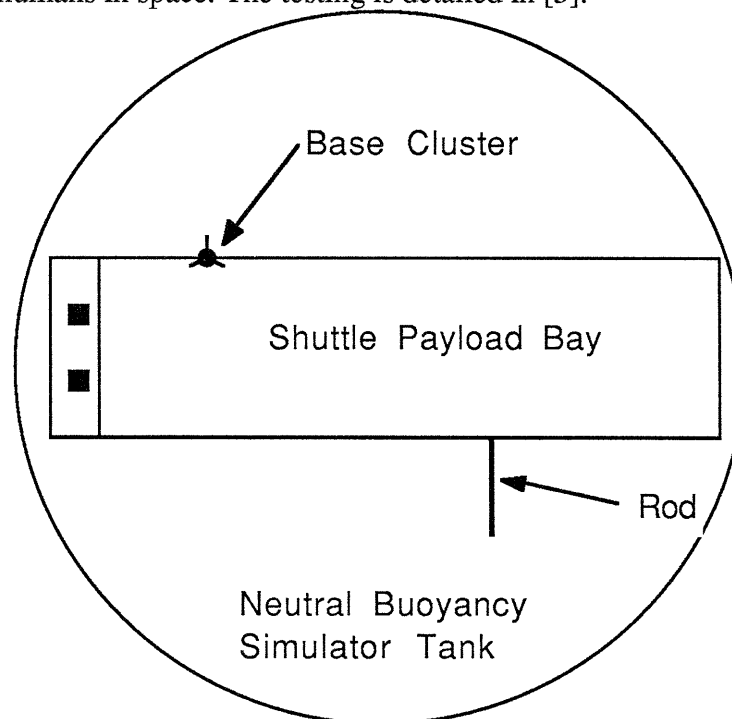


Figure 5.1 NB-67 Structural assembly setup

The MSFC-NBS consists of a cylindrical tank of water 12m deep and 23m across. A mockup of the space shuttle payload bay across the bottom of the tank was used as a base for BAT's EASE structure assembly. The EASE base cluster was attached to the side of the payload bay. Since the shuttle bay doors were not included in the mockup, BAT could approach the base cluster from outboard and below. Beam and cluster supplies were

simulated by a rod attached to the other side of the mock up. Docking to this rod, BAT would receive a beam or cluster from a support diver.

ICS was located along a walkway outside the tank. The operator had no direct view of BAT. Together with the video from the cameras onboard BAT, ICS was also equipped with video from cameras on the bulkhead of the payload bay and on the side of the tank. These external views were very useful for planning longer flying trajectories and for operator orientation.

Due to limitations on battery life and pressurization and pneumatic air supplies, BAT was able to operate for one hour at a time, insufficient for a full assembly of the EASE structure. Instead, the steps toward the completion were done separately and repeatedly over several runs. Video tape data from these runs were then used to determine a time for full structure assembly.

5.1.2 Times for assembly and proportions

Assembly of the Ease structure was divided into four major steps, each repeated three times. Each of the major steps were then divided into three or four tasks and each task divided into up to 16 primitive tasks. The full break down is listed in appendix V. The major tasks are

1. Attach a cluster to a beam in the beam rack.
2. Attach an upright beam to the base cluster.
3. Attach a cross beam to a top cluster.
4. Complete a triangle joint. Attach the free end of a cross beam to the appropriate top cluster. Structural constraints make the beam manipulation more difficult in this task.

Each major task is made up of four primitive task types. These are flying and docking, beam retrieval, video switching, and joint assembly.

Using these divisions, the first full assembly time for the EASE structure was found to be 89 minutes, slightly longer than the 71 minutes for the first neutral buoyancy assembly

by a pressure suited subject. The first BAT assembly was divided into 36% flying and docking, 4% beam retrieval, 4% video switching, and 56% joint assembly.

5.1.3 Comments on Fatigue

The high percentage of joint assembly time prompted the development of Triad with the intent of lowering the workload of the operator. One hour seemed to be the limit on continuous teleoperation before the operator began to fatigue significantly. With this figure it seems improbable that if BAT itself were capable of carrying sufficient consumables for the hour and a half necessary for a end to end manual assembly, that the operator would be capable of completing all 12 major tasks in one sitting. Decreased workload is another driving force behind the study of automation of whatever portion of the assembly is possible.

5.2 Results of NB-67C

5.2.1 Test setup and procedure

In January of 1988, during test NB-67C, BAT went back to MSFC-NBS for the purpose of assembling the EASE structure with the addition of supervisory control.

This time the setup was changed to attempt a more realistic configuration. The configuration used during STS flight 61B held the beams in a vertical rack with the clusters placed on top. Since BAT requires more elbow room, and to avoid getting tangled up in the payload bay, the beams and cluster were held in horizontal racks at the level of the top of the payload bay. The base cluster was placed on the same pallet at the center of the payload bay. See figure 5.2.

This set up is a more accurate approximation of an actual flight configuration. The extensive use of divers in the NB-67 set up was eliminated.

Due to mechanical problems, it was not possible to complete the entire EASE structure at MSFC-NBS. Repeated times for each task were recorded. Times for many of the tasks involving Triad use were verified at the M.I.T. Alumni Swimming Pool.

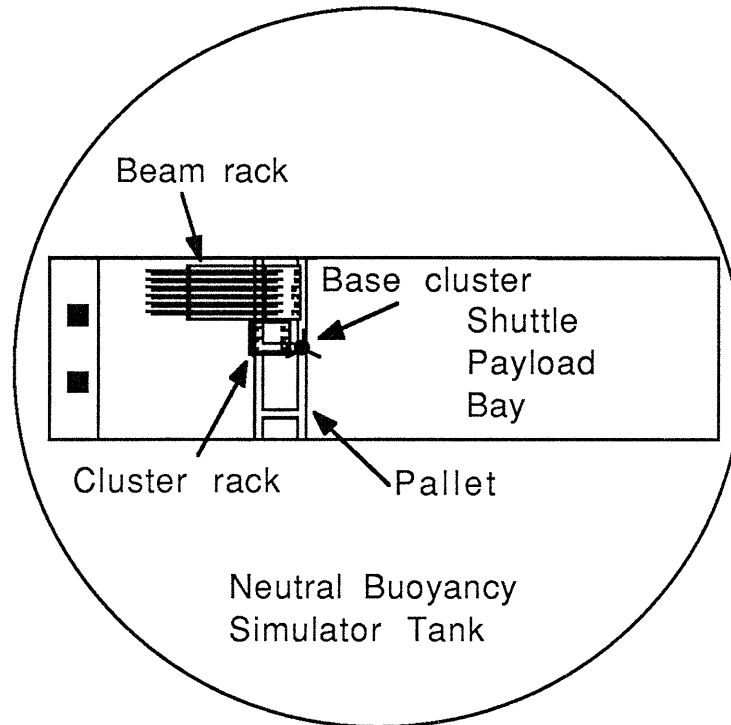


Figure 5.2 NB-67C Structural assembly setup

The procedure again contained four major tasks as follows:

1. Attach a cluster to a beam in the beam rack. This involves docking to one of the grapppling points on the cluster rack with the left claw, retrieving a cluster with the right arm, flying to the next available beam in the beam rack, docking and attaching the cluster to the beam end.
2. Attach an upright beam to the base cluster. Once the clusters are attached to the first three beams, the first two beams are attached to the base cluster as uprights. This involves flying to the beam rack, docking to the next available grapppling point, retrieving a beam from the rack with the beam carrier, flying over to the base cluster, docking, and attaching the beam to the base cluster.
3. Attach a cross beam to a top cluster. Once two riser beams are attached, the clusters on their free ends are available. Between these, a cross beam is attached. This involves the same procedure as in major task 2, except that the beam is attached to a top cluster instead of the base cluster.

4. Complete a triangle. This involves completing the second connection to a cross beam and immediately follows attaching the first connection to the cross beam. BAT translates along the cross beam, docks to the free end of the beam and maneuvers it close to the cluster. The operator then uses the right arm to grab the top cluster and bring it in to make the joint.

5.2.2 Mechanical setbacks

The PID control system, in the process of tightening up the response of the arm, also increases the wear. The point to point arm control system is very discontinuous in acceleration. As each new command comes in, the duty cycle changes instantly, changing the torque on the motor and giving a jerk to the arm. In comparison, the P-type motions were more smooth due to the slow and linear change in duty cycle with respect to command position error.

As a result, during the testing at MSFC-NBS, several of the arm components malfunctioned, bringing testing to a sudden halt. During the course of NB-67C, one of the wrist encoders failed due to moisture, the elbow bearings occasionally froze, the shoulder pitch motor coil melted, and a brush on the shoulder yaw motor failed. All of these problems were repairable. If, indeed, they had occurred simultaneously, they could have been dealt with without loss of too much testing time. They were not simultaneous, however. The more the arm was run using the high PID gains of Triad, the more breakdowns occurred. Only after three weeks of overhaul back at M.I.T. was the arm completely functional again.

5.2.3 Estimation of Triad assembly times and proportions

Using the task component times developed during NB-67C and subsequent pool tests at M.I.T., an assembly time of 72 minutes was established. Considering that the operator did not have extensive experience with the full EASE structure, the assembly can be considered a first assembly and is comparable to the naïve pressure suited subject time of

71 minutes. The time spent on assembly was broken down to 51% flying and docking, 5% beam retrieval, 5% video switching, and 39% joint assembly.

5.3 Comparison of NB-67 and NB-67C data

5.3.1 Direct comparison of assembly times

There are two major differences between the assemblies in NB-67 and NB-67C. One is the difference in setup, leading to flying times that are not directly comparable. The other difference is, of course, the joint assembly (manual or Triad). These tasks are exactly the same in result, and can thereby be compared directly. Table 5.1 shows the direct comparison of NB-67 assembly times and those of NB-67C.

EASE Assembly Breakdown

Manual	Triad	Major Task	(times in seconds)
NB-67	NB-67C	Task Component	
393	177	1. Attach a Cluster to a Beam in the Beam Rack	
97	46	A. Fly to cluster rack (from free beam end)	
62	31	B. Get cluster from cluster rack	
90	46	C. Fly to beam end	
144	54	D. Attach cluster to beam	
522	472	2. Attach an Upright Beam to the Base Cluster	
56	103	A. Fly to beam rack (from cluster rack, base cluster, top cluster)	
41	38	B. Load a beam in the beam carrier	
109	180	C. Fly to the base cluster	
316	151	D. Assemble beam to base cluster	
442	433	3. Attach a Cross Beam to a Top Cluster	
46	148	A. Fly to beam rack (from base cluster or top cluster)	
34	37	B. Load beam in beam carrier	
116	110	C. Fly to proper top cluster	
246	138	D. Assemble beam to top cluster	
427	364	4. Complete a Triangle Joint	
91	81	A. Fly to the free end of the beam	
32	18	B. Fly beam end close to cluster	
304	265	C. Assembly joint	
5352	4338	Total Assembly Time	
89.2	72.3	minutes	

	Breakdown	(times in minutes)
31.9	36.6	1. Fly and Dock
3.8	3.7	2. Beam Retrieval (from rack)
3.5	3.4	3. Video Switching
50.2	28.5	4. Joint Assembly

Table 5.1 Direct comparison of assembly times

The flying times differ by nearly five minutes overall. The significance of this will be discussed in the next section. The beam manipulation and video switching times were nearly the same. The big difference, as one might expect, is in joint assembly, where the Triad time shows a 43% reduction from the manual time. Again, the Triad assembly still includes the manual setting of the rocker, the most time consuming portion of the assembly. This shows a significant advantage to supervisory control, even in this primitive application.

5.3.2 Weighted comparison

The setup for NB-67 was very different from that of NB-67C. This leads to different proportions in flying times. The flying times during NB-67 do not have a defined beginning point, only a defined end point. When getting the beam, the operator would dock to the rod shown in figure 5.1, or receive the beam from a diver at an undefined location. The position of the base cluster was not constant, changing position depending on the other testing being performed at the time. In NB-67C, the process is exactly the same each time, starting at the same beam rack grappling point or the same top cluster. This is a more realistic simulation of an orbital assembly.

For the purpose of better comparing the overall times, table 5.2 shows the NB-67 joint assembly with NB-67C flying times. It is important to note that the operators during both test doubled or tripled their operation experience during the test. It is easy to accept that an operator with more experience free flying open loop could easily lower these times.

Amalgamation

Manual	Triad	Major Task	(times in seconds)
NB-67C	NB-67C	Task Component	
298	177	1. Attach a Cluster to a Beam in the Beam Rack	
46	46	A. Fly to cluster rack (from free beam end)	
62	31	B. Get cluster from cluster rack	
46	46	C. Fly to beam end	
144	54	D. Attach cluster to beam	
637	472	2. Attach an Upright Beam to the Base Cluster	
103	103	A. Fly to beam rack (from cluster rack, base cluster, top cluster)	
38	38	B. Load a beam in the beam carrier	
180	180	C. Fly to the base cluster	
316	151	D. Assemble beam to base cluster	
541	433	3. Attach a Cross Beam to a Top Cluster	
148	148	A. Fly to beam rack (from base cluster or top cluster)	
37	37	B. Load beam in beam carrier	
110	110	C. Fly to proper top cluster	
246	138	D. Assemble beam to top cluster	
403	364	4. Complete a Triangle Joint	
81	81	A. Fly to the free end of the beam	
18	18	B. Fly beam end close to cluster	
304	265	C. Assembly joint	
5637	4338	Total Assembly Time	
94.0	72.3	minutes	
Breakdown			(times in minutes)
36.6	36.6	1. Fly and Dock	
3.8	3.8	2. Beam Retrieval (from rack)	
3.5	3.5	3. Video Switching	
50.2	28.5	4. Joint Assembly	

Table 5.2 Comparison of manual and Triad assembly times with NB-67C flying

From Table 5.2, it is clear that the addition of Triad alone has lowered the estimated total assembly time by 23%, and the joint assembly time specifically by 43%.

5.4 Limitations of the present configuration

Triad, by comparison with other supervisory control methods, is an automation of only a limited portion of the assembly process. The present hardware and sensor configuration of BAT limits what can be done. These limits will be discussed in detail in chapter 6.

6. CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

Teleoperation is needed for task completion in dangerous environments. Supervisory control extends the abilities of manual control for better repeatability, speed and duration.

A method was presented for applying a task primitive supervisory control to the Beam Assembly Teleoperator, and compared with manual control modes. The supervisory routines, known as Triad, allowed the operator to relinquish control of BAT's dextrous arm to the computer for several portions of the joint assembly.

Manual and Triad enhanced control were compared by repeated assembly of the EASE joint by three operators of varied experience, and by assembly of the full EASE structure. Various types of manual control were also used. A PD-type manual control system proved best for the operator experienced in using a P-type control system. Command inputs by the experienced operator were shown to contain a higher order control strategy to optimize the performance of the PD control. A PID-type manual control system reduced assembly times for the less experienced operator by 20% due largely to improved tracking. The experienced operator's times, however increased by 80% from the PD- to the PID-type system, attributable mainly to unlearning of previous control techniques.

The addition of Triad produced a decrease of 80% average assembly time for the naïve operator and 30% for the less experienced operator. Half of the less experienced operator's times under Triad equaled or exceeded the experienced operator's fastest manual time. For the portions of assembly controlled by Triad, average assembly times were nearly the same for the less experienced and the naïve operators, showing an independence with respect to training.

For full EASE structural assembly, Triad produced an estimated full assembly time of 72 minutes, 20% faster than the previously established manual time of 98 minutes, despite

increased times for flying and docking. The times for the portions assembly under Triad control were decreased 43% from the corresponding manual times.

6.2 The future of BAT

The purpose of BAT as set out in Eric Shain's thesis [10] is to extend the Space Systems Lab's work in the area of human assembly of simulated space structures in the direction of man-machine systems. BAT was designed specifically for assembly of the EASE structure. Productivity results could then be directly compared with those produced by manual assembly. Shain envisioned a more dextrous arm, stereo vision systems, and innovative control schemes in the future.

One area yet to be explored is that of coordinated control of the body of BAT and its arm together. The need to coordinate is inherent in a zero gravity environment. In manipulation of some object of finite mass, the base of the teleoperator will also move, causing errors. John Spofford in [11] and Craig Carignan in [12] deal with this problem in detail. In the case of BAT, the manipulator arm is missing a degree of freedom in rotation. Coordinated control would allow for the use of the propulsion base to make up for this lost ability.

Presently, BAT gets around the problem by docking to some object defined as fixed and manipulating objects in a reference framed based on that object. In space, there are instances in which this will be unacceptable. One example is that of completing a triangle in the construction of an EASE structure. In order to complete a triangle, the operator docked to the top beam and flew it over to upright beam. On arrival, he maneuvered the base of BAT to align the cluster and the beam, in some cases actually moving the entire EASE structure. Only after the beam and cluster had been properly aligned and the cluster brought into the range of the right arm, could the operator power up the arm and complete the task.

Herb Viggh, in [5], tried several methods of combining control of the propulsion base and the arm at the same time, a total of eleven degrees of freedom. Using the head controller for rotation, he controlled three DOF with his head, one hand on the translation joystick, and the other in the master arm controlling the right arm. In another mode, he used a finger switch to use the master arm alternately to control the right arm and the rotational DOF of the propulsion base. With enough repetition these modes might have become natural to the operator, but the biggest barrier was that the propulsion system and the right arm positioning were independent. Using coordinated control, the operator need only be concerned with the position of end effector and allow the control systems to figure out what combination of propellers and servos are required to get it there. The barrier between BAT and coordinated control is a lack of sensor capability. The propulsion system currently runs open loop, without feedback. The Three Dimensional Attitude Positioning System, 3DAPS, under development by the SSL, shows promise for providing absolute positioning information.

Improvement of the right arm control involves modeling. Due to water drag, water currents, motor damping, backlash, encoder noise and non-linear duty cycle to torque ratio, modeling would be quite difficult. Implementation of a model based control system would be facilitated by joint velocity feedback and determination of joint torque.

Another direction is that of compliance control. With the addition of force sensors to the wrist of the right arm, elements of force control could be added. This would be especially advantageous for joint completion. A force controller could slide the beam end and mushroom together and control the force necessary to set the capture rocker.

The addition of proximity sensors in each of the claws would increase the information available to a program such as Triad. Many of the errors possible with Triad could be avoided if the system knew whether the right claw was approaching a target or whether the beam carrier actually held a beam to be retrieved.

Considering these possible directions of research, the preceding chapters should be considered a report on the progress of the Beam Assembly Teleoperator, not its completion.

REFERENCES

- [1] Akin,D.L., Minsky,M.L.,Theil,E.D.,Kurtzman,C.R.*Space Applications of Automation, Robotics and Machine Intelligence (ARIMIS) - Phase II*, NASA CR 3734, 1983.
- [2] Sheridan,T.B.,*Supervisory Control of Remote Manipulators, Vehicles, and Dynamic Processes: Experiments in Command and Display Aiding*. Cambridge: Man-Machine Systems Laboratory Report, 1983.
- [3] Akin, D.L., et al., EASE Final Report, in preparation.
- [4]Spofford, J.R., *Beam Assembly Teleoperator: Coordinates and Kinematics*. MIT Space Systems Laboratory Report #6-87, 1987.
- [5] Viggh, H.E.M, *Artificial Intelligence Applications in Teleoperated Robotic Assembly of the EASE Space Structure*, M.S. Thesis, M.I.T., 1987.
- [6] Brady, M. "Trajectory Planning", *Robot Motion: Planning and Control*, Cambridge: MIT Press, 1982, pp.221-243.
- [7] Marra, A., *Human Operator Performance in Dynamic Multi-Task Environments*, M.S. Thesis, M.I.T., 1988.
- [8] Åström, K.J., and Wittenmark, B, *Computer Controlled Systems, Theory and Design*, Prentice-Hall, 1984.
- [9] Ziegler, J.G., and Nichols, N.B., "Optimal Settings for Automatic Controllers," *Trans. ASME*, v64, pp759-768, 1942.
- [10] Shain, E.B., "*Design and Control of a Beam Assembly Teleoperator* ", M.S. Thesis, M.I.T., 1983.
- [11] Spofford, J.R., "*Coordinated Control of a Free-Flying Teleoperator* ", PhD Thesis, M.I.T., 1988.
- [12] Carignan, C.R., "*Control Strategies for Manipulating Payloads in Weightlessness with a Free-Flying Robot* ", ScD Thesis, M.I.T., 1987.

APPENDIX I. BAT mobility using RMS

In January of this year, during test NB-67C, an opportunity arose to use the neutral buoyancy version of the shuttle Remote Manipulator System (RMS) for BAT's mobility. The back side of BAT was attached to the RMS end effector and BAT's thrusters were disabled.

During the short test, the RMS operator maneuvered BAT to dock to a cross beam of a completed EASE tetrahedron, then translate along the beam and dock to the other end of the same beam. The first docking took 19 minutes. The translation and second docking took 7 minutes. These are compared with average free flying times of 1.5 minutes for each maneuver.

The operating conditions were difficult. The RMS operator at the MSFC-NBS is stationed on the second level on the outside of the tank with only a view through a porthole to work with. Distortion through the glass and a working distance of 10m made depth perception difficult. The ICS operator, with the assistance of BAT's belly and tilt and pan stereo cameras, gave directions to the RMS operator in vehicle coordinates. The operator then figured out which joints to move to produce the cartesian movement. The mechanical condition of the RMS was also not perfect. Potentiometers controlling several of the joints of the RMS were not steady, making station keeping non-trivial.

Still, the demonstration was successful. The RMS positioned BAT sufficiently for docking.

There are advantages to using an arm for base mobility. Since the RMS is mounted to a sufficiently large mass, station keeping and reaction forces during manipulation are not as important. BAT can swing a beam without the beam and BAT swinging toward each other. A by-product is that BAT need not dock to any other fixed target in order to interact

with it. This does assume an arm that provides rigid positioning, not completely true for the RMS.

This particular demonstration warrants repetition under better conditions. First, the RMS controls at MSFC-NBS are two 3DOF hand controllers, with each DOF directly controlling one joint of the RMS in rate. Using a resolved rate joystick control has been shown to be much more productive¹ in manipulation. The key is to make the control of the RMS as transparent as possible. If the operator is working from the vehicle cameras, the control system should take input in vehicle coordinates. Second, the RMS operator should have a good view of the scene, such as shuttle bay bulkhead cameras, RMS cameras, and vehicle cameras, perhaps under the control of the vehicle operator. A stereo image is more helpful than a single image². The sum of this is that the operator of the teleoperator should not feel the difference between free flight and RMS positioning, except for the advantages in the increased station keeping inertia afforded by the anchoring of the RMS.

¹Poulton, E.C., *Tracking Skill and Manual Control*, New York: Academic Press, 1974.

² Cole, R.E., Pepper, R.L., and Pinz, B.E., *The Influence of Head Movement Parallax on Perceptual Performance Under Direct and TV-Displayed Conditions*. San Diego NOSC Technical Report #678, May, 1981.

APPENDIX II. BATSOFT

```

/*-----*/
/* The following is a listing of the PC portion of the BAT software.
** It is a bit jumbled due to the number of authors, but I've tried to
** streamline it as much as possible. Nonfunctional subroutine have
** been commented out, but are keep as a record. TRIAD1.c and
** TRIAD2.c, found at the end, are entirely my own.    DEA 3/88 */
/*-----*/
/* -- BATSOFT.c -- Main Programs for BAT operation. */
/* Requires support files as follows: */
/* SUPPORT1, SUPPORT2, SUPPORT3, SUPPORT4, SUPPORT5, DATERS, PCIO
** Triad1, and Triad2 */
/* Beam Assembly Teleoperator Controller
   version 6.4
   October 1987
   (revisions: Finger Switch access to Triad)
   John Spofford, Tom Waschura, Tony Manganiello, Herbert Viggh
   & David E. Anderson
   Space Systems Laboratory
   Massachusetts Institute of Technology */

#include "DAT.H" /* defines data logging */
#include "PORTNUMS.H" /* defines io ports & commands */
#define RABKDZ          8 /* deadzone for backup arm pots */
#define RASTDZ          24 /* dz on stick control */
#define SWITCH_TC       12 /* 1.3 sec lockout at 9 samples/sec */
#define ACTCOLS 17 /* display actuator columns in screen */

/*-----*/
int testing; /* flag for development */
main()
{
    extern int testing;
    char tempc;

    scr_clr();
    initialize_bat();
    calibrate_bat();
    screen();
    run_bat();
    shutdown_bat();
}

/*-----*/
/* initialize_bat -- set up io ports, set initial values,
** */
initialize_bat()
{
    int i;

    scr_rowcol(23,0);
    printf("Initializing\n");
    bat_ports();

    output(P4C,0); /* initialize video switching - inactive since v5.4 */
    output(P0C,0xB8); /* cs high, master&puma stb hi */
    output(P0A,0);
    output(P1B,0);
    output(P1C,0XC0); /* 11000000 spare pnuc high default */
    output(P2C,0x1F); /* 00011111 JM-STB high, all pnueatics default */
    output(P3A,0);
    output(P3B,0);
    output(P3C,0);
    output(P0X,C6HI); /* reset JCS */
    output(P2X,C7HI); /* reset propulsion controller */
    for (i = 0; i < 20; i++); /* pulse width */
    output(P0X,C6LO);
    output(P2X,C7LO);
    for (i = 0; i < 36; i++)
        synchronize(); /* let JCS check memory */

    printf("Loading JCS software\n");
    if (load_control_system())
    {
        printf("\n%cError loading control system software\n",7);
        shutdown_bat();
        exit(0);
    }
    start_log(); /* init data recording buffers */
}

/*-----*/
/* calibrate_bat --
** */
/* triad preset variables */
int funnel_speed;
char triad_data;
/*-----*/

```

```

int stickoff[6];
int rbkoff[5],masoff[5]; /* backup & master offsets */
float rbkml[5],masml[5]; /* backup & master multipliers */
/*
This has been added to the current batsoft for the purpose
of slaving the tilt and pan unit to the head controller.
The head controller has five pots from which can pitch, yaw,
and roll can be calculated. The mux assignments are:
    pot name    pot label(x,y)    pot #
    -----
    top yaw      0,5              1
    top pitch     0,6              2
    roll         10,0             3
    bot pitch     11,0             4
    bot yaw      12,0             5

tilt = cpitch = bot pitch - top pitch
pan  = cyaw = bot yaw + top yaw
roll = roll      HV 2/6/85 */

int potlabelx[5] = { 0 0 10 11 12 };
int potlabeley[5] = { 5 6 0 0 0 };
int headoff[5],h_pot[5];
float cpit_mult = 1.0;
float cyaw_mult = 1.0;
int cpitch,cyaw;
int wra_off,wr_b_off,elb_off;
float p_mult,r_mult,y_mult; /* arm flying */
float ppit_mult,pyaw_mult,prol_mult; /* head flying */
float joy_mult[6];
int p_headoff[5];
int ppitch,pyaw,proll;

calibrate_bat()
{
    static int max_index[5] = { 9, 6, 5, 0, 2 };
    static int min_index[5] = { 7, 8, 4, 1, 3 };
    static int count_index[10] = { 6, 6, 7, 7, 5, 5, 4, 3, 4, 3 };
    extern int stickoff[6];
    extern int rbkoff[5],masoff[5]; /* backup & master offsets */
    extern float rbkml[5],masml[5]; /* backup & master multps */
    int i, dummy, word, fp, count, tdat;
    char tempc;
    float tempf;
    float bat_max[5], bat_min[5];
    int bat_temp[10];
    extern int max_jcs_val[8], min_jcs_val[8];
    extern char icnvt2[8], icnvt[8];
    extern float align_gain[8];
    extern int align_offset[8];
    extern int headoff[5];
    extern float cpit_mult,cyaw_mult;
    float gain[13];
    extern float p_mult,r_mult,y_mult; /* arm flying */
    extern float ppit_mult,pyaw_mult,prol_mult; /* head flying */
    extern int wra_off,wr_b_off,elb_off;
    extern int p_headoff[5],h_pot[5];
    extern int funnel_speed;
    extern char triad_data;

    funnel_speed = 0x400;
    triad_data = FALSE;

    scr_rowcol(23,0);
    scr_clr();
    printf("Calibrating, return joysticks to center.");
    for (i = 0; i <= 32000; i++)
        dummy = 1 * 2 * 3 * 4 * 5;
        dummy = 1 * 2 * 3 * 4 * 5;
        dummy = 1 * 2 * 3 * 4 * 5;
    stickoff[0] = convert(0,8) & 0xFFFF;
    stickoff[1] = convert(0,10) & 0xFFFF;
    stickoff[2] = convert(0,9) & 0xFFFF;
    stickoff[3] = convert(6,0) & 0xFFFF;
    stickoff[4] = convert(7,0) & 0xFFFF;
    stickoff[5] = convert(8,0) & 0xFFFF;
    for (i = 0; i <= 5; i++)
        stickoff[i] = 1024 - stickoff[i];

    scr_setmode(0);
    if ((fp = open("BAT_SET.ARM",0)) != -1)
    {
        for (count = 0; count < 10; count++)
        {
            while((tempc = getc(fp)) != '\n');
            fscanf(fp,"%f\n",&tempf);
            bat_temp[count] = tempf;
        }
        for (count = 1; count < 3; count++)
        {
            while ((tempc = getc(fp)) != '\n');
            fscanf(fp,"%f\n",&tempf);

```

```

        max_jcs_val[count] = tempf;
        while ((tempc =getc(fp)) != '\n');
        fscanf(fp,"%f\n",&tempf);
        min_jcs_val[count] = tempf;
    }
    for (count = 0; count < 13; count++)
    {
        while ((tempc =getc(fp)) != '\n');
        fscanf(fp,"%f\n",&tempf);
        gain[count] = tempf;
    }
    for (count = 0; count < 6; count++)
    {
        while ((tempc =getc(fp)) != '\n');
        fscanf(fp,"%f\n",&tempf);
        joy_mult[count] = tempf;
    }

    close(fp);
}

else
{
    scr_setmode(2);
    printf("\n\n%c==> BAT arm data not found <==\n",7);
    exit(1);
}

for (i=0; i<5; i++)
{
    align_gain[i+3] = gain[i];
}

cpit_mult = gain[5];
cyaw_mult = gain[6];
p_mult = gain[7];
r_mult = gain[8];
y_mult = gain[9];
ppit_mult = gain[10];
pyaw_mult = gain[11];
prol_mult = gain[12];
for (i = 3; i < 8; i++)
{
    tdat = (convert(icnvt2[i],icnvt[i]) & 0xFF) - 1024;
    tdat = adjust(tdat,i);
    bat_max[i-3] = (float) bat_temp[max_index[i-3]];
    bat_min[i-3] = (float) bat_temp[min_index[i-3]];
    word = (int) ((bat_max[i-3]+bat_min[i-3])/(2.0*align_gain[i]));
    align_offset[i] = word - tdat;
    max_jcs_val[i] = (int) bat_max[i-3];
    min_jcs_val[i] = (int) bat_min[i-3];
}

wra_off = convert(2,0) - 1024;
wrb_off = convert(1,0) - 1024;
elb_off = convert(3,0) - 1024;

read_head();
for (i=0; i<=4; i++) /* head controller offsets */
{
    headoff[i] = h_pot[i];
    p_headoff[i] = h_pot[i];
}

}

/*-----*/
/* run_bat -- */

/*
The following RAM is set aside for joint position
control. The arrays are used as follows:

max_p.....maximum positive position
min_p.....maximum negative position
div_or_mul...left or right shifts?
p_gain.....number of bits to shift
p_range_err..a range error occured on that joint

Note, all are indexed from 0 to 7 which correspond to
different joints. The actual correspondence is:

0.....left arm
1.....camera tilt
2.....camera pan
3.....wrist a
4.....wrist b
5.....elbow
6.....shoulder yaw
7.....shoulder pitch

These are software definable as to the addresses and
commands in iaddr and icmnd.

T.E.W. April 24, 1984 */

char *potname[8] = {"Left Arm\0", "Camera Tilt\0", "Camera Pan\0",

```

```

"Wrist A0", "Wrist B0", "Elbow0",
"Shoulder Yaw0", "Shoulder Pitch0");
/*----- Left, Cam T, Cam P, Wr A, Wr B, Elbow, Sh Y, Sh P)*/
char icnvt2[8]
= { 0, 0, 0, 2, 1, 3, 4, 5};
char icnvt[8]
= { 2, 5, 6, 0, 0, 0, 0, 0};
int max_p[8]
= { 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024};
int min_p[8]
= { -1024, -1024, -1024, -1024, -1024, -1024, -1024, -1024};
int div_or_mul[8]
= { FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE};
int p_gain[8]
= { 0, 0, 0, 0, 0, 0, 0, 0};
int p_range_err[8]
= { TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE};
static char iaddr[8]
= { 3, 4, 4, 1, 1, 3, 2, 2};
static char icmnd[8]
= { 0x90, 0x80, 0x90, 0x80, 0x90, 0x80, 0x90, 0x80};
int max_jcs_val[8]
= {0x07FF,0x0780,0x0780,0x0600,0x07A0,0x0780,0x0600,0x0140};
int min_jcs_val[8]
= {0xF800,0xF880,0xF880,0xF880,0xFA20,0xFC00,0xF960,0xFB80};
int align_offset[8]
= { 0, 0, 0,0xFE90,0x0213,0x03DF,0x0D48,0xFC06};
float align_gain[8]
= { 1.0, 1.0, 1.0, 3.00, 3.00, 1.75, -4.00, 1.25};
int card_order[8]
= { 6, 7, 8, 1, 2, 5, 4, 3};
/*----- Left, Cam T, Cam P, Wr A, Wr B, Elbow, Sh Y, Sh P)*/
/* dataset 4/13/84 JRS */
int mnf[8] = {3, 4, 7, 6, 5, 0, 1, 2};
int slave_toggle; /* variables for slaverest)*/
int curr_master[5], curr_h_pot[5];
int set_triad;
int Gain[9], alpha[9], beta[9], alg[9], pos[9], mult1[9], mult2[9];
int joint_pos[8], joint_com[8];
char comm_disp;
char mdat[30], mcom[30], madd[30];
char data_flag, boost_flag; /* logical flags */
char left_claw_state, master_connected;
char head_controller, puma_on, draw, right_claw;
char head_ctrl_connected, slave_arm_on;
char master_flying, head_ctrl_flying;
char beam_carrier, beam_claw;
char spare1, spare2; /* spare pneumatics toggles */

/* successive triad counters for forward and reverse assembly */
int tri_for, tri_rev;

DATATAG cmi[TAGSIZE] = {"AIAloop counter"};
DATATAG th0[TAGSIZE] = {"BIFjoystick thr"};
DATATAG th1[TAGSIZE] = {"CIFrimmed thr"};
DATATAG rawm[TAGSIZE] = {"DIEraw master"};
DATATAG jcom[TAGSIZE] = {"BIEjoint com"};
DATATAG jpos[TAGSIZE] = {"FIEjoint pos"};
DATATAG zcn[TAGSIZE] = {"GIAz80 loop"};
DATATAG adty[TAGSIZE] = {"HIEarm duty"};
/*-----*/
run_bat()
{
extern char comm_disp;
extern pos[9];
extern char mdat[30], mcom[30], madd[30]; /* master transmit buffer */
char pdat[10]; /* puma transmit buffer */
extern char cswitch[16]; /* ICS control switches */
char i,j,k,temp; /* fast loop index, temp */
char com_row, pos_row, thr_old_row[6], old_com_row[8], old_pos_row[8], col;
char row; /* display parameters */
int dvec[8][ACTCOLS+1]; /* display filter */
int thr_row[6], thr_trim[6], temp; /* x,y,z,p,r,y */
extern int max_jcs_val[8], min_jcs_val[8];
extern char icnvt2[8], icnvt[8];
extern int set_triad;
float spw;

int mlast,plast; /* master,puma buffer ends */
int p_roll,p_pitch,p_yaw,p_x,p_y,p_z;
int tdat,actdat[8]; /* temp & data */
int a_syaw,a_spit,a_elbo,a_wrol,a_wyaw,a_left;
extern int testing;

extern char data_flag; /* logical flags */
extern char left_claw_state, master_connected;
extern char head_controller, puma_on, draw, right_claw;
extern char head_ctrl_connected, slave_arm_on;
extern char master_flying, head_ctrl_flying;
extern char beam_carrier, beam_claw;
extern char keep[MAXGRPNUM]; /* select variables */
int raw_master[5];

```

```

int posm[8];
extern int joint_com[8], joint_pos[8];
int thr_boost[6];

int tran_num;
char jump; /* flag for arm jump */
extern int curr_master[5];
extern int stickoff[6];
extern int rtkoff[5], masoff[5]; /* backup & master offsets */
extern float rbkmul[5], masmul[5]; /* backup & master mults */
int counter; /* main loops */
extern int slave_toggle; /* toggle to reset slave arm gains */
extern int potlabelx[5], potlabely[5]; /* head controller muxes */
extern int headoff[5]; /* hc offsets */
extern float cpit_mult, cyaw_mult; /* hc multipliers */
int extern_h_pot[5];
int extern_cpitch, cyaw;
int old_headoff[5], old_h_pot[5];
int loopz80, arm_duty[5];
extern int curr_h_pot[5];

extern int wra_off, wrb_off, elb_off;
int wra_raw, wrb_raw, elb_raw;
int delta_a, delta_b, delta_c;
extern float p_mult, r_mult, y_mult;
extern float joy_mult[6];
extern ppitch, proll, pyaw;
/* successive triad counters for forward and reverse assembly */
extern int tri_for, tri_rev;

for (i = 0; i < 8; i++) /* set up display filters */
{
    spw = ((float) (max_jcs_val[i] - min_jcs_val[i])) / ((float) ACTCOLS);
    dvec[i][0] = max_jcs_val[i];
    dvec[i][ACTCOLS+1] = min_jcs_val[i];
    for (j = 1; j <= ACTCOLS; j++)
        dvec[i][j] = max_jcs_val[i] - (int)((float)j) * spw;
}

tri_for = 1;
tri_rev = 1;
comm_disp = FALSE;
puma_on = FALSE;
head_controller = TRUE;
draw = TRUE;
data_flag = FALSE;
left_claw_state = FALSE;
boost_flag = FALSE;
master_connected = TRUE;
head_ctrl_connected = TRUE;
slave_arm_on = TRUE;
master_flying = FALSE;
head_ctrl_flying = FALSE;
right_claw = FALSE;
slave_toggle = 0;
beam_carrier = TRUE;
beam_claw = FALSE;
spare1 = FALSE;
spare2 = FALSE;
jump = FALSE;
set_triad = 0;

output(P2X, C7HI); /* reset propulsion controller */
for (i = 0; i < 20; i++); /* pulse width */
output(P2X, C7LO);
for (i = 0; i < 6; i++)
    pdat[i] = 128; /* zero each command */
tran_puma(pdat, 5); /* Prevent thruster start-up */

screen2();
for (i = 1; i <= 4; i++)
    start_jc(i); /* start card #1,2,3,4 */
for (i = 0; i <= 5; i++)
{
    thr_old_row[i] = 13; /* avoid randomness */
    old_com_row[i] = 13;
    old_pos_row[i] = 13;
}
old_com_row[6] = 13;
old_com_row[7] = 13;
old_pos_row[6] = 13;
old_pos_row[7] = 13;

output(P2X, C0LO); /* ACITVATE POWER RELAY */

synchronize(); /* get in sync */
for (counter = 0; TRUE; counter++) /* as long as in sync... */
{
    stat();
    if (data_flag)
        index_log();
    output(P2X, C6HI); /* ... ** TEST TIMING: WATCH P2C6 ** */
}

```

```

/* --- set up loop parameters */
mlast = -1;
plast = -1;

/* nada in buffers */

check_switches(); /* updates offsets and flags */

/* --- convert appropriate BAT pots,
message & transmit to master card */
if (head_ctrl_connected && !head_ctrl_flying)
{
    read_head();
    for (i=0; i<5; i++)
        curr_h_pot[i] = h_pot[i];
    head_cam_slave1();
    tran_tp();
}
else
{
    read_head();
    for (i=0; i<5; i++)
        curr_h_pot[i] = h_pot[i];
}

if (master_connected && !master_flying)
{
    if (slave_toggle == 1) slaverest(2); /*reset gains if powering up*/
    for (i = 3; i <= 7; i++)
    {
        raw_master[i-3] = (convert(icnvt2[i],icnvt[i]) & 0xFFFF) - 1024;
        curr_master[i-3] = adjust(raw_master[i-3],i);
        pos[card_order[i]] = align(curr_master[i-3],i) & 0xFFFF;
    }
    tran_arm(); /* transmit command to arm */
}
else
{
    if (slave_toggle == 1)
    {
        tran_zero(); /* zero out residual commands */
        slaverest(2);
    }
    for (i = 3; i <= 7; i++)
    {
        raw_master[i-3] = (convert(icnvt2[i],icnvt[i]) & 0xFFFF) - 1024;
        curr_master[i-3] = adjust(raw_master[i-3],i);
    }
}

/* display command and arm position - DEA 11/86 */
if (draw)
{
    read_joints(joint_com,joint_pos);
    if(set_triad == 3) learn_record();
    for (i=0; i<=7; i++)
    {
        if ((joint_com[i] & 0x8000) != 0) /*negative command */
        {
            /*right shift 12 bit number and sign extend */
            joint_com[i] = (joint_com[i] >> 4) | 0xF000;
        }
        else
        {
            joint_com[i] = joint_com[i] >> 4;
            if (i == 3 || i == 6 || i == 4) /*elb, shy, ct counter intuitive*/
                joint_com[i] = -joint_com[i];
        }
        if ((joint_pos[i] & 0x8000) != 0) /*negative position*/
        {
            posm[i] = (joint_pos[i] >> 4) | 0xF000;
            if (i == 3 || i == 6 || i == 4) /*elb, shy, ct counter intuitive*/
                posm[i] = -posm[i];
        }
        else
        {
            posm[i] = joint_pos[i] >> 4;
            if (i == 3 || i == 6 || i == 4) /*elb, shy, ct counter intuitive*/
                posm[i] = -posm[i];
        }
    }
    com_row = 0;
    pos_row = 0;
    for (j=1; j <= ACTCOLS+1; j++)
    {
        if((joint_com[i] >= dvec[mnf[i]][j]) && (joint_com[i] <= dvec[mnf[i]][j-1]))
            com_row = j+1;
        if((posm[i] >= dvec[mnf[i]][j]) && (posm[i] <= dvec[mnf[i]][j-1]))
            pos_row = j+1;
    }
    col = (i < 1) + 24;
    if (com_row != old_com_row[i])
    {
        /*cover up old command position */
        scr_rowcol(old_com_row[i],col);
        old_com_row[i] = com_row;
    }
    scr_co(179);
}

```

```

        }
        if ((pos_row & 0x1F) != (old_pos_row[i] & 0x1F), col)
        {
            /* cover up old arm position */
            scr_rowcol(old_pos_row[i], col);
            old_pos_row[i] = pos_row;
            scr_co(179);
        }
        if (com_row == pos_row)
        {
            /* command and position coincide */
            scr_rowcol(com_row, col);
            scr_co(219);
        }
        else
        {
            /* show new command and arm positions */
            scr_rowcol(com_row, col);
            scr_rowcol(pos_row, col);
            scr_co(233);
            scr_co(4);
        }
    }
}

/* --- convert appropriate PUMA pots,
message & transmit to PUMA controller */

thr_raw[0] = -(convert(7,0) - 1024 + stickoff[4])*joy_mult[0]; /*X*/
thr_raw[1] = -(convert(8,0) - 1024 + stickoff[5])*joy_mult[1]; /*Y*/
thr_raw[2] = -(convert(6,0) - 1024 + stickoff[3])*joy_mult[2]; /*Z*/

if (master_flying)
{
    if (master_connected)
    {
        wra_raw = (convert(2,0) - 1024 - wra_off);
        wrb_raw = (convert(1,0) - 1024 - wrb_off);
        elb_raw = (convert(3,0) - 1024 - elb_off);
        delta_a = wra_raw;
        delta_b = wrb_raw;
        delta_c = delta_a - delta_b;
        thr_raw[3] = -(elb_raw * p_mult);
        thr_raw[4] = (delta_a - (delta_c * 0.5)) * r_mult;
        thr_raw[5] = (delta_c * 0.5) * y_mult;
    }
    else
    {
        thr_raw[3] = 0;
        thr_raw[4] = 0;
        thr_raw[5] = 0;
    }
}

else if (head_ctrl_flying)
{
    if (head_ctrl_connected)
    {
        read_head();
        head_puma_slave1();
        thr_raw[3] = ppitch;
        thr_raw[4] = -proll;
        thr_raw[5] = -pyaw;
    }
    else
    {
        thr_raw[3] = 0;
        thr_raw[4] = 0;
        thr_raw[5] = 0;
    }
}

else
{
    thr_raw[3] = -(convert(0,9) - 1024 + stickoff[2])*joy_mult[3]; /*P*/
    thr_raw[4] = -(convert(0,10) - 1024 + stickoff[1])*joy_mult[4]; /*R*/
    thr_raw[5] = (convert(0,8) - 1024 + stickoff[0])*joy_mult[5]; /*Y*/
}

for (i=0; i<6; i++)
{
    if (thr_raw[i] > 1024)
        thr_raw[i] = 1024;
    else if (thr_raw[i] < -1024)
        thr_raw[i] = -1024;
}

if (boost_flag)
{
    calc_boost(joint_com, joint_pos, thr_boost);
    for (i = 0; i <= 5; i++)
        thr_raw[i] += thr_boost[i];
}

trim_thrust(thr_raw, thr_trim);
for (i = 0; i <= 5; i++)
{
    temp = thr_trim[i];
    temp = (temp > 127) ? (127) : (temp); /* limit */
    temp = (temp < -127) ? (-127) : (temp);
    pdat[i] = (temp + 128);
}

```

```

        if (pdat[i] == 0xFF)
            pdat[i] = 0xFE;
    }

    /* DISPLAY THRUST LEVELS */
    if (draw)
    {
        for (i = 0; i <= 5; i++)
        {
            row = 17 - (pdat[i] >> 4);
            if (row != thr_old_row[i])
            {
                col = (i << 1) + 1;
                scr_rowcol(row,col);
                scr_co(219);
                scr_rowcol(thr_old_row[i],col);
                scr_co(179);
                thr_old_row[i] = row;
            }
        }
    }

    if (puma_on)
        tran_puma(pdat,5);
    else
    {
        for (i=0; i<=5; i++)
            pdat[i] = 128;
        tran_puma(pdat,5);
    }

    if(comm_disp)
        comm_display();

    output(P2X,C6LO); /* --- ** TEST TIMING: WATCH P2C6 ** */
    tempc = scr_csts();
    if (tempc == 0)
        synchronize; /* avoid sync loss from printing */
    else if (tempc == 9)
        break; /* shutdown on a tab */
    else
    {
        keyboard_command(tempc); /* process */
        synchronize();
    }

    if (data_flag && keep[1])
        login(cnt,counter);
    if (data_flag && keep[2])
        login_a(th0,thr_raw);
    if (data_flag && keep[3])
        login_a(th1,thr_trim);
    if (data_flag && keep[4])
        login_a(rawm,raw_master);
    if (data_flag && keep[5])
        login_a(jcom,joint_com);
    if (data_flag && keep[6])
        login_a(jpos,joint_pos);
    if (data_flag && keep[8])
    {
        arm_duty[0] = peek(0xC,0xE880);
        arm_duty[1] = peek(0xD,0xE880);
        arm_duty[2] = peek(0xC,0xE900);
        arm_duty[3] = peek(0xD,0xE900);
        arm_duty[4] = peek(0xC,0xE980);
        login_a(adty,arm_duty);
    }

    if (data_flag && keep[7])
    {
        loopz80 = peek(0x10,0xE900)/(peek(0x11,0xE900) << 8);
        login(zcnt,loopz80);
    }
}

scr_rowcol(10,0);
printf("\nLOST SYNCHRONIZATION\n"); /* go to shutdown */
}

/*-----*/
/* shutdown_bat --
** */
shutdown_bat()
{
    int i;

    output(P2C,7); /* DEACTIVATE POWER RELAY and claws */

    output(P0X,C6HI); /* reset JCS */
    output(P2X,C7HI); /* reset propulsion controller */
    for (i = 0; i < 20; i++) /* pulse width */
        output(P0X,C6LO);
    output(P2X,C7LO);
    scr_setmode(2);
}

```

```

/*-----*/
/* This function checks all of the switches and toggles the
proper logic flags and updates offsets. - HV */

int old_master[5], old_offset[5];
int old_h_pot[5], old_headoff[5];
int fold_headoff[5], fold_h_pot[5];
char triadfn[10][15];

check_switches()
{
    extern char data_flag, boost_flag;
    extern char left_claw_state, master_connected;
    extern char head_controller, puma_on, draw, right_claw;
    extern char head_ctrl_connected, slave_arm_on;
    extern char master_flying, head_ctrl_flying;
    extern char beam_carrier, beam_claw;
    extern char spare1, spare2; /* spare pneumatics */
    extern int align_offset[8];
    extern int curr_master[5], curr_h_pot[5];
    extern int joint_pos[8];
    extern int old_master[5], old_offset[5];
    extern int fold_master[5], fold_offset[5];
    extern int wrb_off, wrb_off, elb_off;
    extern int h_pot[5], headoff[5], p_headoff[5];
    extern int old_headoff[5], old_h_pot[5];
    extern int set_triad;
    extern char triadfn[10][15]; /* list of file names for triad */
    int i, dummy;
    extern char cswitch[8];
    extern int fold_headoff[5], fold_h_pot[5];
    extern int tri_for, tri_rev; /* for successive triad use */
    read_switches(); /* --- check ICS switches */

    if (cswitch[0]) /*-RED-*/
    {
        shutdown_bat();
        exit(2);
    }

    if (cswitch[1]) /* start/stop recording data */
    {
        if (set_triad==0)
        {
            data_flag = !data_flag;
            co(7);
            screen2();
        }
        else if (set_triad==1)
        {
            co(7);
            point_to_point();
        }
        else if (set_triad>=2)
        {
            co(7);
            learn();
        }
    }

    if (cswitch[9]) /*-BLUE-*/
    {
        /* thruster toggle */
        puma_on = !puma_on;
        co(7);
        screen2();
    }

    /*-----Removed for triad switching-----*/
    /*
    if (master_flying)
    {
        if (cswitch[7]) /* TURN OFF */
        {
            master_flying = FALSE;
            screen2();
            co(7);
            if (master_connected)
            {
                if (slave_arm_on)
                {
                    slaverest(1); /* realign master arm with slave arm */
                    slave_toggle = 0; /* keep slaverest from reloading gains */
                    reset_pid();
                }
            }
            else
            {
                for (i=0; i<5; i++)
                    align_offset[i+3] = old_offset[i] + old_master[i] - curr_master[i];
            }
        }
        else /* STILL ON */
        {
            {
            }
        }
    }
    else
    {
        if (cswitch[7]) /* TURN ON */

```

```

        {
            master_flying = TRUE;
            screen2();
            co(7);
            wra_off = convert(2,0) - 1024;
            wrb_off = convert(1,0) - 1024;
            elb_off = convert(3,0) - 1024;
            if (master_connected)
                for (i=0; i<5; i++)
                {
                    old_offset[i] = align_offset[i+3];
                    old_master[i] = curr_master[i];
                }
        }
    else
        * STILL OFF *
        {
        }
    }

if(head_ctrl_flying)
{
    if (cswitch[5])
        * TURN OFF *
        {
            head_ctrl_flying = FALSE;
            if (head_ctrl_connected)
            {
                for (i=0; i<5; i++)
                {
                    headoff[i] = fold_headoff[i] + curr_h_pot[i] - fold_h_pot[i];
                }
            }
            else
            {
                for (i=0; i<5; i++)
                {
                    old_headoff[i] = fold_headoff[i];
                    old_h_pot[i] = fold_h_pot[i];
                }
            }
            co(7);
            screen2();
        }
    else
        * STILL ON *
        {
        }
}

else
{
    if (cswitch[5])
        * TURN ON *
        {
            head_ctrl_flying = TRUE;
            read_head();
            for (i=0; i<5; i++)
                p_headoff[i] = h_pot[i];
            if (head_ctrl_connected)
            {
                for (i=0; i<5; i++)
                {
                    fold_headoff[i] = headoff[i];
                    fold_h_pot[i] = curr_h_pot[i];
                }
            }
            else
            {
                for (i=0; i<5; i++)
                {
                    fold_headoff[i] = old_headoff[i];
                    fold_h_pot[i] = old_h_pot[i];
                }
            }
            co(7);
            screen2();
        }
    else
        * STILL OFF *
        {
        }
}

-----*/
/*switching for easy triad access   DEA*/
if(cswitch[2]) /*initiate trajectory grab */
{
    strcpy(triadfn[0], "cgrab.trd");
    strcpy(triadfn[1], "0");
    point_to_point();
}

if(cswitch[4]) /*take arm from stow to general area of cluster */
{
    strcpy(triadfn[0], "ctri.trd");
    strcpy(triadfn[1], "0");
    point_to_point();
}

```

```

if(cswitch[5])/*index finger push does successive forward assembly*/
{
    if(tri_for == 1)
    {
        strcpy(triadfn[0],"grabbeam.trd");
        strcpy(triadfn[1],"joint.trd");
        strcpy(triadfn[2],"0");
        point_to_point();
        scr_rowcol(23,16); puts("Grabslv ");
    }

    if(tri_for == 2)
    {
        strcpy(triadfn[0],"grabslv.trd");
        strcpy(triadfn[1],"slideslv.trd");
        strcpy(triadfn[2],"0");
        point_to_point();
        scr_rowcol(23,16); puts("Stow ");
    }

    if(tri_for == 3)
    {
        strcpy(triadfn[0],"stow.trd");
        strcpy(triadfn[1],"0");
        point_to_point();
        scr_rowcol(23,16); puts("Grabbeam ");
    }

    tri_for = tri_for + 1;
    if(tri_for == 4) tri_for = 1;
}

if(cswitch[7])
{
    if(tri_rev == 1)
    {
        strcpy(triadfn[0],"cjoint.trd");
        strcpy(triadfn[1],"0");
        point_to_point();
        scr_rowcol(23,25); puts("Rgrabslv ");
    }

    if(tri_rev == 2)
    {
        strcpy(triadfn[0],"rgrabslv.trd");
        strcpy(triadfn[1],"rslideslv.trd");
        strcpy(triadfn[2],"0");
        point_to_point();
        scr_rowcol(23,25); puts("Rstow ");
    }

    if(tri_rev == 3)
    {
        strcpy(triadfn[0],"rstow.trd");
        strcpy(triadfn[1],"0");
        point_to_point();
        scr_rowcol(23,25); puts("Cjoint ");
    }

    tri_rev = tri_rev + 1;
    if(tri_rev == 4) tri_rev = 1;
}

if (cswitch[3]) /* increment next up triads*/
{
    tri_for = tri_for + 1;
    if(tri_for == 4) tri_for = 1;
    tri_rev = tri_rev + 1;
    if(tri_rev == 4) tri_rev = 1;
    screen2();
}

/*-----*/
if (cswitch[11]) /* toggle right claw */
{
    right_claw = !right_claw;
    /*-----for default right claw -----*/
    if (right_claw)
        output(P2X,C2LO);
    else
        output(P2X,C2HI);
    /*-----replace with pnuo 1-----*/
    if (right_claw)
        output(P1X,C6LO);
    else
        output(P1X,C6HI);
    co(7);
    screen2();
}

if (cswitch[8]|| cswitch[15]) /*left claw toggle(was p2c1,now using p1c7)*/
{
    left_claw_state = !left_claw_state;
    if (left_claw_state)
        output(P1X, C7LO);
    else
        output(P1X, C7HI);
    co(7);
    screen2();
}

```

```

if (master_connected)
{
    if (cswitch[6]) /* DISCONNECT */
    {
        master_connected = FALSE;
        screen2();
        co(7);
        if (!master_flying)
            for (i=0; i<5; i++)
            {
                old_offset[i] = align_offset[i+3];
                old_master[i] = curr_master[i];
            }
    }
    else /* still connected */
    {
        {
        }
    }
}

else
{
    if (cswitch[6]) /* CONNECT */
    {
        master_connected = TRUE;
        screen2();
        co(7);
        if (master_flying)
        {
            wra_off = convert(2,0) - 1024;
            wrb_off = convert(1,0) - 1024;
            clb_off = convert(3,0) - 1024;
        }
        if (!master_flying)
        {
            if (slave_arm_on)
            {
                slaverest(1); /*realign master with slave*/
                slave_toggle = 0; /*keep slaverest from reloading gains */
                reset_pid();
            }
            if (!slave_arm_on)
            {
                for (i=0; i<5; i++)
                    align_offset[i+3] = old_offset[i] + old_master[i] - curr_master[i];
            }
        }
    }
    else /* still disconnected */
    {
        {
        }
    }
}

if (cswitch[10]) /*slave toggle switch*/
{
    if (slave_arm_on)
        slaverest(0);
    else
        slaverest(1);
    co(7);
}

if (head_ctrl_connected)
{
    if (cswitch[13]) /* DISCONNECT */
    {
        head_ctrl_connected = FALSE;
        for (i=0; i<5; i++)
            old_headoff[i] = headoff[i];
        co(7);
        screen2();
    }
    else /* STILL CONNECTED */
    {
        {
            for (i=0; i<5; i++)
                old_h_pot[i] = curr_h_pot[i];
        }
    }
}

else
{
    if (cswitch[13]) /* CONNECT */
    {
        head_ctrl_connected = TRUE;
        for (i=0; i<5; i++)
            headoff[i] = old_headoff[i] + curr_h_pot[i] - old_h_pot[i];
        read_head();
        for (i=0; i<5; i++)
            p_headoff[i] = h_pot[i];
        co(7);
        screen2();
    }
    else /* STILL DISCONNECTED */
    {
        {
        }
    }
}

```

```

    }
    if (cswitch[12]) /*toggle beam carrier*/
    {
        beam_carrier = !beam_carrier;
        if(beam_carrier)
        {
            output(P2X,C3HI); /*extend beam carrier*/
        }
        else
        {
            output(P2X,C3LO); /*retract beam carrier*/
        }
        co(7);
        screen2();
    }
    if (cswitch[14]) /*toggle beam carrier claw (was also cswitch[3])*/
    {
        beam_claw = !beam_claw;
        if (beam_claw)
        {
            output(P2X,C4LO); /* close beam carrier claw*/
        }
        else
        {
            output(P2X,C4HI); /*open beam carrier claw*/
        }
        co(7);
        screen2();
    }
/*
    if (cswitch[]) *spare pneumatics 1
    {
        spare1 = !spare1;
        if (spare1)
        {
            output(P1X,C6LO); /*actuate spare pneumatics 1*
        }
        else
        {
            output(P1X,C6HI);
        }
        co(7);
    }
    if (cswitch[]) *spare pneumatics 2*
    {
        spare2 = spare2;
        if (spare2)
        {
            output(P1X,C7LO);
        }
        else
        {
            output(P1X,C7HI);
        }
        co(7);
    }
    if (cswitch[])
        stow();*/
}
/* End of Batsoft.c */
/*-----*/

```

```

/* SUPPORT1.C -- BAT support routines      version 6.4
** by John Spofford and Tom Waschura and David E. Anderson,
** M.I.T. Space Systems Laboratory      December 4, 1987
** revisions include Joint step input routine */
#include "PORTNUMS.H"
#include "DATA.H"
#define READ_CNT 21
/*-----*/
/* FEEDBACK.C -- interacts with the BAT/joint card/master card loop */
feedback()
{
    int not_done, selection, i;

    not_done = TRUE;
    while (not_done)
    {
        selection = get_sel2();      * light pen soon? *
        switch(selection)
        {
            case 1:
            case 2:
            case 3:
            case 4:
                shut_card_off(selection);
                work_card(selection);
                break;

            case 0:
                not_done = FALSE;
                break;

            case 9:
                test_routine();
                break;

            default:
                beep4(25);
                scr_rowcol(1,1);
                printf("Invalid Selection");
                pause(100);
        }

        shut_card_off(1);
        shut_card_off(2);
        shut_card_off(3);
        shut_card_off(4);
    }
}

*-----*
beep4(n)
{
    int n;
    {
        int i;

        output(0x61,0x43);
        for (i=0;i<n*25;i++);
        output(0x61,0x40);
        return(0);
    }
}

*-----*
paint_menu_3()
{
    scr_setmode(0);
    scr_clr();
    scr_rowcol(2,0);
    printf("      M.I.T. SSL\n");
    printf(" Beam Assembly Teleoperator\n\n");
    printf(" Hierarchical Control Evaluation\n\n");
    printf("  1. Wrist A + B\n");
    printf("  2. Shoulder Pitch + Yaw\n");
    printf("  3. Elbow + Left  \n");
    printf("  4. Camera Tilt + Pan\n");
    printf("  0. EXIT\n");
    printf("  9. Debugging test routine");
    scr_rowcol(22,0);
    printf(" Enter Selection: ");
    return(0);
}

*-----*
get_sel2()
{
    int query;

    paint_menu_3();
    scr_rowcol(22,19);
    printf(" ");
    scr_rowcol(22,19);
    scanf("%d",&query);
    return(query);
}

*-----*
char read_buffer[40];
*-----*
work_card(card_addr)
{
    int card_addr;
    {

```

```

extern char read_buffer[];
int i, posx, posy;

scr_cursorf();
paint_menu_4();
scr_rowcol(3,0);
switch(card_addr)
{
    case 1: printf("      Wrist A Wrist B");          break;
    case 2: printf("      Shoulder Pitch  Yaw");        break;
    case 3: printf("      Elbow    Left Arm");          break;
    case 4: printf("      Camera Tilt   Pan");          break;
    default: printf("ERROR");
}
while( !(scr_csts() == 13) ) * unless CR *
{
    rqst_mstr(card_addr);
    read_mast(READ_CNT);

    * posx = read_buffer[0] + read_buffer[1]*256;
    * posy = read_buffer[2] + read_buffer[3]*256;
    * scr_rowcol(14,10);
    * printf("%04x  %04x"); *

    scr_rowcol(14,0);
    for (i = 0; i < READ_CNT; i++)
        printf(" %02x", read_buffer[i]);
    while( scr_csts() == 32 ); * stop a while *
}
}
*-----*
read_mast(cnt)
int cnt;
{
    extern char read_buffer[];
    char i, waitc;

    for(i = 0; i < cnt; i++)
    {
        for (waitc = 0; !(import(P4B) & 1); waitc++) * JM-RDY hi *
            if (waitc > 250)
            {
                printf("R");
                break; * timeout *
            }

        read_buffer[i] = import(P4A);
        outport(P2X, C5H); * JM-STB high *
        for (waitc = 0; import(P4B) & 1; waitc++) * JM-RDY hi *
            if (waitc > 200)
            {
                printf("r");
                break; * timeout *
            }

        outport(P2X, C5LO); * JM-STB low *
    }
}
*-----*
paint_menu_4()
{
    scr_clr();
    scr_rowcol(0,0);
    printf("CARD:\n\n");
    printf("  Channel    X    Y    \n");
    printf("          \n");
    printf("          \n");
    printf("  Gain\n");
    printf("  Ref Pos\n");
    printf("Joint Pos\n");
}
*-----*
shut_card_off(card)
int card;
{
    unsigned jc_seg;

    jc_seg = SEG_OFF + 0x80 * card;
    poke('S', COMM_FLAG_jc_seg);
    while (peek(JC_STATUS_jc_seg) != 'W')
        scr_csts(); * no hangup *
    poke('G', COMM_FLAG_jc_seg);
    while (peek(JC_STATUS_jc_seg) != 'C')
        scr_csts(); * no hangup *
}
*-----*
char temp_ch;
*-----*
test_routine()
{
    char strobe, ready_var, jmd_data, buffer[150], i;
    extern char temp_ch;

```

```

scr_clr();
scr_rowcol(0,0);
printf("1= STB hi, 2= STB lo, 3= log, 4= cont\n\n");
printf("RDY STB DATA\n");
while(1)
{
    temp_ch = scr_csts();
    switch (temp_ch)
    {
        case '1':    strobe = 1;
                     output(P2X,C5HI);    * jm-stb hi *
                     break;
        case '2':    strobe = 0;
                     output(P2X,C5LO);    * jm-stb lo *
                     break;
        case '3':    * read 150 bytes *
                     output(P2X,C5LO);
                     for (i = 0; i < 150; i++)
                         buffer[i] = readjm(); * get data *
                     scr_rowcol(5,0);
                     for (i = 0; i < 150; i++)
                         printf(" %02x",buffer[i]);
                     while(!scr_csts());
        case '4':    * cont. read *
                     output(P2X,C5LO);
                     while(!scr_csts())
                     {
#asm
; ASM88 *
;=====
; Modified READJM.C for speed
;=====
CSEG

a_j1:
    mov     dx,311h    ;port 4B
    in      al,dx      ;get data
    mov     ah,0       ;char value
    and     ax,1
    cmp     ax,0
    jnz     a_j2
    jmp     a_j1

a_j2:
    mov     dx,310h    ;port 4A
    in      al,dx      ;get data
    mov     temp_ch,al  ;return in temp_ch
    mov     ax,11      ;C5 HI
    mov     dx,30Bh    ;Port 2X
    out     dx,al       ;put it out..

a_j3:
    mov     dx,311h    ;Port 4B
    in      al,dx      ;get data
    mov     ah,0       ;char value
    and     ax,1
    cmp     ax,0
    jz      a_j4
    jmp     a_j3

a_j4:
    mov     ax,10      ;C5 LO
    mov     dx,30Bh    ;Port 2X
    out     dx,al       ;put it out..
;=====
; *
;
    }

    case 13:    return;
    default:    break;
    }

    scr_rowcol(3,0);
    ready_var = (inport(P4B) & 1);
    jmd_data = inport(P4A);
    printf("%2x %2x %02x",ready_var,strobe,jmd_data);
}

}

*-----*
* readjm.c -- reads joint control system master card *
* produces assembly code *
readjm()
{
    char tempc;

    while(!(inport(P4B) & 1)); * wait for rdy *
    tempc = inport(P4A); * get data *
    output(P2X,C5HI);    * jm-stb hi *
    while(inport(P4B) & 1); * wait for rdy low *
    output(P2X,C5LO);    * jm-stb lo *
    return(tempc);
}

*-----*/
#define CAMERA_ADDR    4
#define WRIST_ADDR     1
#define SHOULDER_ADDR  2

```

```

#define LEFT_ARM_ADDR          3
#define CAMERA_TILT            0
#define CAMERA_PAN             1
#define WRIST_A                0
#define WRIST_B                1
#define SHOULDER_PITCH 0
#define SHOULDER_YAW          1
#define LEFT_ARM               1
#define ELBOW                  0
/*-----*/
char data_buffer[41];
static int rows[] = { 5, 6, 7, 8, 9 };
static int pot[] = { 0, 1, 2, 3, 4 };
char command_buffer[41] = { 0xE0, 0xC0, 0xA0, 0x80, 0x01,
                           0xF0, 0xD0, 0xB0, 0x90, 0x03,
                           0xE0, 0xC0, 0xA0, 0x80, 0x01,
                           0xF0, 0xD0, 0xB0, 0x90, 0x03,
                           0xE0, 0xC0, 0xA0, 0x80, 0x01,
                           0xF0, 0xD0, 0xB0, 0x90, 0x03,
                           0xE0, 0xC0, 0xA0, 0x80, 0x01,
                           0xF0, 0xD0, 0xB0, 0x90, 0x03,
                           0xE0, 0xC0, 0xA0, 0x80, 0x01,
                           0xF0, 0xD0, 0xB0, 0x90, 0x03, 0 };

char addr_buffer[41]      = { 4, 4, 4, 4, 4,
                              4, 4, 4, 4, 4,
                              1, 1, 1, 1, 1,
                              1, 1, 1, 1, 1,
                              2, 2, 2, 2, 2,
                              2, 2, 2, 2, 2,
                              3, 3, 3, 3, 3,
                              3, 3, 3, 3, 0 };

char *algorithm[6] = { "P-type \0", "PID-type\0", "SVC-type\0",
                      "O.L. *\0", "ERROR \0", "ERROR \0" };

char cur_command_buffer[41], cur_data_buffer[41];
/* WRA WRB SHP SHY ELB PMP CT CP*/
float scale[9] = {0.0, 50.272, 50.272, 67.28, 50.272, 63.104, 50.272, 50.272, 50.272};
int Gain[9], alpha[9], beta[9], mult1[9], mult2[9], alg[9], pos[9], mult3[9];
/*-----*/
setup_joints()
{
    int not_done,i;
    char selection;
    char fname[15];
    not_done = TRUE;
    while (not_done)
    {
        selection = get_selection(); /* light pen? */
        switch(selection)
        {
            case '1':
                tune_joint(CAMERA_ADDR, CAMERA_TILT,7);
                break;
            case '2':
                tune_joint(CAMERA_ADDR, CAMERA_PAN,8);
                break;
            case '3':
                tune_joint(WRIST_ADDR, WRIST_A, 1);
                break;
            case '4':
                tune_joint(WRIST_ADDR, WRIST_B, 2);
                break;
            case '5':
                tune_joint(SHOULDER_ADDR, SHOULDER_PITCH, 3);
                break;
            case '6':
                tune_joint(SHOULDER_ADDR, SHOULDER_YAW, 4);
                break;
            case '7':
                tune_joint(LEFT_ARM_ADDR, ELBOW, 5);
                break;
            case '8':
                tune_joint(LEFT_ARM_ADDR, LEFT_ARM, 6);
                break;
            case '9':
                printf(" Load filename [.BDT]: ");
                scanf("%s",fname);
                strcat(fname, ".BDT");
                load_from_file(fname);
                scr_rowcol(21,0);
                printf(" Data Transmitted to BAT");
                pause(100);
                break;
            case '0':
                not_done = FALSE;
                break;
            default:
                beep(25);
                scr_rowcol(1,1);
                printf("Invalid Selection, Dude! Try a number.");
                pause(200);
        }
    }
    printf(" Write filename [.BDT]: ");
    scanf("%s",fname);
}

```

```

        strcat(fname, ".BDT");
        write_out_data(fname);
    }
}

/*-----*/
beep(n)
{
    int n;
    {
        int i;

        output(0x61, 0x43);
        for (i=0; i<n*100; i++);
        output(0x61, 0x40);
        return(0);
    }
}

/*-----*/
paint_menu_1()
{
    scr_setmode(0);
    scr_clr();
    scr_rowcol(2,0);
    printf(" M.I.T. SSL\n");
    printf(" Beam Assembly Teleoperator\n\n");
    printf(" Joint Algorithm Tuning\n\n");
    printf(" 1. Camera Tilt\n");
    printf(" 2. Camera Pan\n");
    printf(" 3. Wrist A\n");
    printf(" 4. Wrist B\n");
    printf(" 5. Shoulder Pitch\n");
    printf(" 6. Shoulder Yaw\n");
    printf(" 7. Elbow\n");
    printf(" 8. Cooling Pump\n");
    printf(" 9. LOAD From File\n");
    printf(" 0. END JOINT TUNING");
    scr_rowcol(22,0);
    printf(" Enter Selection: ");
    return(0);
}

/*-----*/
get_selection()
{
    char query;

    paint_menu_1();
    scr_rowcol(22,19);
    printf(" ");
    scr_rowcol(22,19);
    scanf("%c", &query);
    return(query);
}

/*-----*/
tune_joint(card_addr, x_or_y, joint)
int card_addr, x_or_y, joint;
{
    extern float scale[9];
    extern int Gain[9], alpha[9], beta[9], mult1[9], mult2[9], alg[9], pos[9], mult3[9];
    float gainf, alphaf, betaf, mult1f, mult2f;
    float K, A, B, C;
    int i, done, transmit;
    int temp;
    char query, direct, inputt, dh, dl, conversion;
    int jsegm, joff;
    inputt = FALSE;
    transmit = FALSE;
    conversion = FALSE;
    jsegm = 0xE800 + card_addr*0x80;
    joff = x_or_y * 2;
    paint_menu_2();
    scr_rowcol(0,19);
    switch(joint)
    {
        case 1: printf("Wrist A"); break;
        case 2: printf("Wrist B"); break;
        case 3: printf("Shoulder Pitch"); break;
        case 4: printf("Shoulder Yaw"); break;
        case 5: printf("Elbow"); break;
        case 6: printf("Cooling Pump"); break;
        case 7: printf("Camera Tilt"); break;
        case 8: printf("Camera Pan"); break;
        default: printf("ERROR");
    }
    poke('U', TOZ80, jsegm);
    dl = peek((joff+0x84), jsegm);          dh = peek((joff+0x85), jsegm);
    temp = dl + (dh << 8);
    scr_rowcol(5,21);

```

```

printf("%4x ",temp);
dl = peek(joff+0x88,jsegm);
temp = dl + (dh << 8);
scr_rowcol(6,21);
printf("%4x ",temp);
dl = peek(joff+0x8C,jsegm);
temp = dl + (dh << 8);
scr_rowcol(7,21);
printf("%4x ",temp);
dl = peek(joff+ 0x80,jsegm);
temp = (dl >> 4) + (dh << 4);
scr_rowcol(8,21);
printf("%d ",(signed)(temp));
temp = peek(((joff/2)+0x90),jsegm);
scr_rowcol(9,21);
printf("%s",algorithm[temp]);
dl = peek(joff+0x9C,jsegm);
temp = dl + (dh << 8);
scr_rowcol(10,21);
printf("%4x ",temp);
dl = peek(joff+0xA0,jsegm);
temp = dl + (dh << 8);
scr_rowcol(11,21);
printf("%4x ",temp);
dl = peek(joff+0xA4,jsegm);
temp = dl + (dh << 8);
scr_rowcol(12,21);
printf("%4x ",temp);
poke('O',TOZ80,jsegm);
scr_rowcol(13,20);
printf("Transmit OFF");
scr_rowcol(13,00);
printf("type I to input");
scr_rowcol(14,20);
printf("Conversions C : OFF");
done = FALSE;
scr_rowcol(14,0);
printf("Input Variables");
scr_rowcol(15,0);
printf(" {000->FFF} ");
while (done == FALSE)
{
    if(inputt)
    {
        if(conversion)
        {
            scr_rowcol(17,20);
            printf("Set K: ");
            scanf("%f",&K);
            scr_rowcol(18,20);
            printf("Set A: ");
            scanf("%f",&A);
            scr_rowcol(19,20);
            printf("Set B: ");
            scanf("%f",&B);
            scr_rowcol(20,20);
            printf("Set C: ");
            scanf("%f",&C);
            scr_rowcol(21,20);
            printf("Position: ");
            scanf("%d",&pos[joint]);
            scr_rowcol(22,20);
            printf("Int Sat(hex): ");
            scanf("%4x",&mult2[joint]);
            scr_rowcol(23,20);
            printf("Int Mult(hex): ");
            scanf("%4x",&mult3[joint]);
            mult1f = C*65536.0;
            if (mult1f >= 65536.0) printf("mult1 overflow");
            mult1[joint] = (int) mult1f;
            gainf = scale[joint]*K*(B + 1.0);
            if (gainf >= 65536.0) printf("gain overflow");
            Gain[joint] = (int) gainf;
            alphaf = scale[joint]*K*(A - C - 1.0 - 2.0 * B);
            if(alphaf <= -65536.0) printf("alpha overflow");
            alpha[joint] = -(int) alphaf;
            betaf = scale[joint]*K*(B - A*C + C);
            if (betaf >= 65536.0) printf("beta overflow");
            beta[joint] = (int) betaf;
        }
        else
        {
            scr_rowcol(16,0);
            printf("Set G: ");
            scanf("%4x",&Gam[joint]);
            scr_rowcol(17,0);
            printf("Set Alpha: ");
            scanf("%4x",&alpha[joint]);
            scr_rowcol(18,0);
            printf("Set Beta: ");
            scanf("%4x",&beta[joint]);
            scr_rowcol(19,0);

```

```

        printf("Position: ");
        scanf("%d ", &pos[joint]);
        scr_rowcol(20,0);
        printf("Set Mult1: ");
        scanf("%4x", &mult1[joint]);
        scr_rowcol(21,0);
        printf("Set Mult2: ");
        scanf("%4x", &mult2[joint]);
        scr_rowcol(22,0);
        printf("Set Mult3: ");
        scanf("%4x", &mult3[joint]);
    }
    scr_rowcol(5,30);
    printf("%4x ", Gain[joint]);
    scr_rowcol(6,30);
    printf("%4x ", alpha[joint]);
    scr_rowcol(7,30);
    printf("%4x ", beta[joint]);
    scr_rowcol(8,30);
    printf("%d ", pos[joint]);
    scr_rowcol(10,30);
    printf("%4x ", mult1[joint]);
    scr_rowcol(11,30);
    printf("%4x ", mult2[joint]);
    scr_rowcol(12,30);
    printf("%4x ", mult3[joint]);
    scr_rowcol(9,30);
    printf("%s", algorithm[alg[joint]]);
    inputt = FALSE;
}

/* put rate_n onto screen */
while(peek(FROMZ80,jsegm) == 'U');
poke('R',FROMZ80,jsegm);
temp = peek(0x04+joff,jsegm)^(peek(0x05+joff,jsegm)<<8);
scr_rowcol(13,35);
printf("%4x",temp);
poke('O',FROMZ80,jsegm);
query = scr_csts();
switch(query)
{
    case 0:      break;
    case 'T':
    case 't':

        scr_rowcol(13,30);
        if (transmit)
        {
            transmit = FALSE;
            printf("OFF");
        }
        else
        {
            transmit = TRUE;
            printf("ON ");
        }
        break;

    case '0':
    case '1':
    case '2':
    case '3':

        alg[joint] = query - '0';
        scr_rowcol(9,30);
        printf("%s", algorithm[alg[joint]]);
        break;

    case 'i':
    case 't':

        inputt = TRUE;
        if(conversion)
        {
            scr_rowcol(17,20);
            printf(" ");
            scr_rowcol(18,20);
            printf(" ");
            scr_rowcol(19,20);
            printf(" ");
            scr_rowcol(20,20);
            printf(" ");
            scr_rowcol(21,20);
            printf(" ");
            scr_rowcol(22,20);
            printf(" ");
            scr_rowcol(23,20);
            printf(" ");
        }
        else
        {
            scr_rowcol(16,0);
            printf("      \n");
            printf("      \n");
            printf("      \n");
            printf("      \n");
            printf("      \n");
            printf("      \n");
        }
    }
}

```

```

        printf(" ");
    }
}

break;

scr_rowcol(14,37);
if(conversion)
{
    conversion = FALSE;
    printf("OFF");
    scr_rowcol(15,20);
    printf(" ");
    scr_rowcol(16,20);
    printf(" ");
    scr_rowcol(14,0);
    printf("Input Variables");
    scr_rowcol(15,0);
    printf(" {000->FFF}: ");
}

else
{
    conversion = TRUE;
    printf("ON ");
    scr_rowcol(15,20);
    printf("enter PID parameters");
    scr_rowcol(16,20);
    printf(" {00.000 - 99.999} ");
    scr_rowcol(14,0);
    printf(" ");
    scr_rowcol(15,0);
    printf(" ");
}

break;

break;

scr_rowcol(23,0);
printf("Invalid Key ");
pause(200);
scr_rowcol(23,0);
printf(" ");

case 'c':
case 'C':

case 13:    done = TRUE;

default:    beep(25);

}

if(transmit)
{
    tran_direct(jsegm, joff, joint);
}

}

/*-----*/
paint_menu_2()
{
    scr_clr();
    scr_rowcol(0,0);
    printf("Joint Tuning for:\n\n");
    printf("          Current  New\n\n");
    printf("Parameter  Value  Value\n\n");
    printf("Gain\n");
    printf("Alpha\n");
    printf("Beta\n");
    printf("Position\n");
    printf("Algorithm\n");
    printf("Mult1\n");
    printf("Mult2\n");
    printf("Mult3\n");
}

/*-----*/
load_from_file(fname)
char fname[15];
{
    int i,j,fn,seg;
    extern int Gain[9], alpha[9], beta[9], pos[9], alg[9], mult1[9], mult2[9], mult3[9];

    if((fn = open(fname,0)) == -1)
    {
        scr_rowcol(22,0);
        beep(25);
        printf("ERROR in loading");
        pause(250);
    }

    else
    {
        for(i=1; i<=8; i++)
        {
            Gain[i] = getw(fn);
            alpha[i] = getw(fn);
            beta[i] = getw(fn);
            pos[i] = getw(fn);
            alg[i] = getw(fn);
            mult1[i] = getw(fn);
            mult2[i] = getw(fn);
            mult3[i] = getw(fn);
        }
    }
}

```

```

close(fn);
seg = 0xE800;
for(i = 1; i <= 7; i = i + 2)
{
    seg = seg + 0x80;
    while(peek(TOZ80, seg) == 'R');
    poke('U', TOZ80, seg);
    /* put parameters to dual port ram jcl */
    poke(Gain[i] & 0xFF, 0x84, seg);      poke(Gain[i] >> 8, 0x85, seg);
    poke(Gain[i+1] & 0xFF, 0x86, seg);      poke(Gain[i+1] >> 8, 0x87, seg);
    poke(alpha[i] & 0xFF, 0x88, seg);      poke(alpha[i] >> 8, 0x89, seg);
    poke(alpha[i+1] & 0xFF, 0x8A, seg);      poke(alpha[i+1] >> 8, 0x8B, seg);
    poke(beta[i] & 0xFF, 0x8C, seg);      poke(beta[i] >> 8, 0x8D, seg);
    poke(beta[i+1] & 0xFF, 0x8E, seg);      poke(beta[i+1] >> 8, 0x8F, seg);
    /* no position input for anything but pump */
    if(i == 5)
    {
        poke((pos[i+1] << 4) & 0xFF, 0x82, seg);
        poke((pos[i+1] >> 4) & 0xFF, 0x83, seg);
    }
    poke(alg[i] & 0xFF, 0x90, seg);
    poke(alg[i+1] & 0xFF, 0x91, seg);
    poke(mult1[i] & 0xFF, 0x9C, seg);      poke(mult1[i] >> 8, 0x9D, seg);
    poke(mult1[i+1] & 0xFF, 0x9E, seg);      poke(mult1[i+1] >> 8, 0x9F, seg);
    poke(mult2[i] & 0xFF, 0xA0, seg);      poke(mult2[i] >> 8, 0xA1, seg);
    poke(mult2[i+1] & 0xFF, 0xA2, seg);      poke(mult2[i+1] >> 8, 0xA3, seg);
    poke(mult3[i] & 0xFF, 0xA4, seg);      poke(mult3[i] >> 8, 0xA5, seg);
    poke(mult3[i+1] & 0xFF, 0xA6, seg);      poke(mult3[i+1] >> 8, 0xA7, seg);
    poke('N', TOZ80, seg);
}
}

/*-----*/
pause(n)
{
    int n;
    {
        int i, j;
        for (i = 0; i < n * 100; i++)
            j = j + 1;
    }
}

/*-----*/
write_out_data(fname)
char fname[15];
{
    int fn, i, j;
    extern int Gain[9], alpha[9], beta[9], pos[9], alg[9], mult1[9], mult2[9], mult3[9];

    if((fn = creat(fname)) == -1)
    {
        scr_rowcol(0, 0);
        beep(25);
        printf("Unable to Write File");
        pause(250);
    }
    else
    {
        for(i = 1; i <= 8; i++)
        {
            putw(Gain[i], fn);
            putw(alpha[i], fn);
            putw(beta[i], fn);
            putw(pos[i], fn);
            putw(alg[i], fn);
            putw(mult1[i], fn);
            putw(mult2[i], fn);
            putw(mult3[i], fn);
        }
        close(fn);
    }
}

/*-----*/
tran_direct(jsegm, joff, joint)
int jsegm, joff, joint;
{
    extern int Gain[9], alpha[9], beta[9], pos[9], mult1[9], mult2[9], alg[9];
    while((peek(TOZ80, jsegm) == 'R');
    poke('U', TOZ80, jsegm);
    poke((mult1[joint] & 0xFF), joff + 0x9C, jsegm);
    poke(((mult1[joint] >> 8) & 0xFF), joff + 0x9D, jsegm);
    poke((mult2[joint] & 0xFF), joff + 0xA0, jsegm);
    poke(((mult2[joint] >> 8) & 0xFF), joff + 0xA1, jsegm);
    poke((mult3[joint] & 0xFF), joff + 0xA4, jsegm);
    poke(((mult3[joint] >> 8) & 0xFF), joff + 0xA5, jsegm);
    poke((alpha[joint] & 0xFF), (joff + 0x88), jsegm);
    poke(((alpha[joint] >> 8) & 0xFF), (joff + 0x89), jsegm);
    poke((beta[joint] & 0xFF), (joff + 0x8C), jsegm);
    poke(((beta[joint] >> 8) & 0xFF), (joff + 0x8D), jsegm);
    poke(alg[joint], ((joff/2) + 0x90), jsegm);
    poke((Gain[joint] & 0xFF), (joff + 0x84), jsegm);
    poke(((Gain[joint] >> 8) & 0xFF), (joff + 0x85), jsegm);
    poke(((pos[joint] << 4) & 0xFF), (joff + 0x80), jsegm);
    poke(((pos[joint] >> 4) & 0xFF), (joff + 0x81), jsegm);
}

```

```

        poke('N',TOZ80,jsegm);
    }

/*-----*/
tran_zero() /*zero out all commands in the arm joint*/
/* revision 8/4/87 -- for the purposes of jlt arm control algorithm,
this must include a zeroing of the upper byte of mult2 to reset the
integrator summation */
{
    /* zero out any residual commands jc1*/
    while(peek(FROMZ80,0xE880) == 'U');    poke('R',FROMZ80,0xE880);
    while(peek(TOZ80,0xE880) == 'R');    poke('U',TOZ80,0xE880);
    poke(peek(0x00,0xE880),0x80,0xE880);    poke(peek(0x01,0xE880),0x81,0xE880);
    poke(peek(0x02,0xE880),0x82,0xE880);    poke(peek(0x03,0xE880),0x83,0xE880);
    poke(0,0xA1,0xE880); poke(0,0xA3,0xE880);
    poke('N',TOZ80,0xE880);
    poke('O',FROMZ80,0xE880);
    /* zero out any residual commands jc2*/
    while(peek(FROMZ80,0xE900) == 'U');    poke('R',FROMZ80,0xE900);
    while(peek(TOZ80,0xE900) == 'R');    poke('U',TOZ80,0xE900);
    poke(peek(0x00,0xE900),0x80,0xE900);    poke(peek(0x01,0xE900),0x81,0xE900);
    poke(peek(0x02,0xE900),0x82,0xE900);    poke(peek(0x03,0xE900),0x83,0xE900);
    poke(0,0xA1,0xE900); poke(0,0xA3,0xE900);
    poke('N',TOZ80,0xE900);
    poke('O',FROMZ80,0xE900);
    /* zero out any residual commands jc3*/
    while(peek(FROMZ80,0xE980) == 'U');    poke('R',FROMZ80,0xE980);
    while(peek(TOZ80,0xE980) == 'R');    poke('U',TOZ80,0xE980);
    poke(peek(0x00,0xE980),0x80,0xE980);    poke(peek(0x01,0xE980),0x81,0xE980);
    poke(0,0xA1,0xE980);
    poke('N',TOZ80,0xE980);
    poke('O',FROMZ80,0xE980);
}

/*-----*/
tran_arm() /* transmits to commands to the arm */
/* due to pots, pos[i] is 12bits left justified but we need to */
/* poke 12 bits left justified into 16 bits */
{
    extern int pos[9];
    while(peek(TOZ80,0xE880) == 'R');    poke('U',TOZ80,0xE880);
    poke((pos[1] << 4) & 0xFF,0x80,0xE880);    poke((pos[1] >>4) & 0xFF, 0x81,0xE880);
    poke((pos[2] << 4) & 0xFF,0x82,0xE880);    poke((pos[2] >>4) & 0xFF, 0x83,0xE880);
    poke('N',TOZ80,0xE880);
    while(peek(TOZ80,0xE900) == 'R');    poke('U',TOZ80,0xE900);
    poke((pos[3] << 4) & 0xFF,0x80,0xE900);    poke((pos[3] >>4) & 0xFF, 0x81,0xE900);
    poke((pos[4] << 4) & 0xFF,0x82,0xE900);    poke((pos[4] >>4) & 0xFF, 0x83,0xE900);
    poke('N',TOZ80,0xE900);
    while(peek(TOZ80,0xE980) == 'R');    poke('U',TOZ80,0xE980);
    poke((pos[5] << 4) & 0xFF,0x80,0xE980);    poke((pos[5] >>4) & 0xFF, 0x81,0xE980);
    poke('N',TOZ80,0xE980);
}

/*-----*/
int pos1[8];
tran_arm1() /* transmits to commands to the arm in XXX0 form*/
/* pos[i] is 12 bits left justified into 16 bits */
{
    extern int pos1[8];
    while(peek(TOZ80,0xE880) == 'R');    poke('U',TOZ80,0xE880);
    poke(pos1[0] & 0xFF,0x80,0xE880);    poke((pos1[0] >>8) & 0xFF, 0x81,0xE880);
    poke(pos1[1] & 0xFF,0x82,0xE880);    poke((pos1[1] >>8) & 0xFF, 0x83,0xE880);
    poke('N',TOZ80,0xE880);
    while(peek(TOZ80,0xE900) == 'R');    poke('U',TOZ80,0xE900);
    poke(pos1[2] & 0xFF,0x80,0xE900);    poke((pos1[2] >>8) & 0xFF, 0x81,0xE900);
    poke('N',TOZ80,0xE900);    poke(pos1[3] & 0xFF,0x82,0xE900);    poke((pos1[3] >>8) & 0xFF, 0x83,0xE900);
    while(peek(TOZ80,0xE980) == 'R');    poke('U',TOZ80,0xE980);
    poke(pos1[4] & 0xFF,0x80,0xE980);    poke((pos1[4] >>8) & 0xFF, 0x81,0xE980);
    poke('N',TOZ80,0xE980);
}

/*-----*/
tran_tp() /* transmits to commands to the tilt and pan */
{
    extern pos[9];
    while(peek(TOZ80,0xEA00) == 'R');    poke('U',TOZ80,0xEA00);
    poke((pos[7] << 4) & 0xFF,0x80,0xEA00);    poke((pos[7] >>4) & 0xFF, 0x81,0xEA00);
    poke((pos[8] >>4) & 0xFF, 0x83,0xEA00);    poke((pos[8] << 4) & 0xFF,0x82,0xEA00);
    poke('N',TOZ80,0xEA00);
}

/*-----*/
signext(i)
{
    int i;
    {
        if((i & 0x800) != 0)
        {
            i=i ! 0xF000;
        }
    }
    return(i);
}

/*-----*/
comm_display() /* this just displays what duty cycle is being sent - DEA */
{
    int temp;
    scr_rowcol(16,24);
    printf("%2x",peek(0xC,0xE880));
    scr_rowcol(15,26);
}

```

```

printf("%2x",peek(0xD,0xE880));
scr_rowcol(16,28);
printf("%2x",peek(0xC,0xE900));
scr_rowcol(15,30);
printf("%2x",peek(0xD,0xE900));
scr_rowcol(16,32);
printf("%2x",peek(0xC,0xE980));
scr_rowcol(15,36);
printf("%2x",peek(0xC,0xEA00));
scr_rowcol(16,38);
printf("%2x",peek(0xD,0xEA00));
}
/*-----*/
/* Joint Step is meant to allow the operator to apply a step command to a joint
and record the results via the daters routines. I will use a more direct
method of addressing the joint cards than tran_jcs_dpr. The dual port ram that
acts as the two way closet between the pc and the joint card z80's can be
accessed simply by peeking and poking the variables by which they communicate.
No complex pointer/buffer system is required.

DATATAG onejnt[TAGSIZE] = {"GIAone joint pos"};
DATATAG onecom[TAGSIZE] = {"HIAone joint com"};
DATATAG cntt[TAGSIZE] = {"HIAstep counter"};
joint_step()
{
extern char data_flag;
extern float scale[9];
extern int Gain[9], alpha[9], beta[9], pos[9], mult1[9], mult2[9], alg[9];
int joint;
char testing, looking, transmit, dl, dh;
char written, direct, inputt;
int jsegm, counter, temp, jnt, algjc, joff;
int posjc, temp1;
float tempf, mult1f, mult2f, gainf, alphaf, betaf;
float K, A, B, C;
transmit = FALSE;
testing = TRUE;
looking = TRUE;
direct = FALSE;
inputt = FALSE;
while(testing)
{
while(looking)
{ * paint a joint selection menu *
paint_menu_1();
scr_rowcol(5,0);
printf(" PID Joint Step Algorithm ");
scr_rowcol(15,7);
printf("G. Daters Initialization\n");
printf(" T. Calibrate Control Loop Time \n");
printf(" 0. End Joint Step ");
scr_rowcol(22,19);
scanf("%c",&jnt);
paint_menu_2(); * paint the runtime menu *
scr_rowcol(0,0);
printf("Step Input For: ");
looking = FALSE;
joff = 0;
jnt = toupper(jnt);
switch(jnt)
{ * see which joint was selected *
case '1':
printf("Camera Tilt");
jsegm = 0xEA00;
joint = 7;
break;
case '2':
printf("Camera Pan");
joff = 2;
joint = 8;
jsegm = 0xEA00;
break;
case '3':
printf("Wrist A");
joint = 1;
jsegm = 0xE880;
break;
case '4':
printf("Wrist B");
joff = 2;
joint = 2;
jsegm = 0xE880;
break;
case '5':
printf("Shoulder Pitch");
jsegm = 0xE900;
joint = 3;
break;
case '6':
printf("Shoulder Yaw");
joff = 2;
joint = 4;

```

David E. Anderson, March 8, 1987 *

```

        jsegm = 0xE900;
        break;
    case '7':
        printf("Elbow");
        jsegm = 0xE980;
        joint = 5;
        break;
    case 'G':
        set_data0;
        joint = 6;
        looking = TRUE;
        break;
    case 'T':
        time_loop0;
        looking = TRUE;
        break;
    case '0':
        testing = FALSE;
        break;
    default:
        looking = TRUE;
        beep(25);
        scr_rowcol(1,1);
        printf(" Invalid Selection. Try a number.");
        pause(200);
    }
    if(!looking)
    {
        /*paint the rest of the runtime menu */
        scr_rowcol(10,12);
        printf("Runtime Position:");
        scr_rowcol(11,0);
        printf(" Direct inp: A\n");
        printf(" Start Step: T\n");
        printf(" Zero command: Z\n");
        printf(" Take input: N\n");
        printf(" Change Joints: J\n");
        printf(" Take Data: D\n");
        printf(" Exit: X");
        if(data_flag)
        {
            scr_rowcol(16,19);
            printf("SELECTED");
        }
    }
}
temp = scr_csts();
temp = toupper(temp);
switch(temp)
{
    /* check for commands */
    case 0:
        break;
    case 'X': /* return to main loop */
        testing = FALSE;
        break;
    case 'Z': /* quick power down */
        /* zero out any residual commands jc3 */
        while(peek(FROMZ80.jsegm) == 'U');
        while(peek(TOZ80.jsegm) == 'R');
        poke(peek(0x00+joff.jsegm),joff+0x80.jsegm);
        poke('N',TOZ80.jsegm);
        poke('O',FROMZ80.jsegm);
        break;
    case 'A':
        if (direct)
        {
            direct = FALSE;
            scr_rowcol(11,19);
            printf(" ");
        }
        else
        {
            direct = TRUE;
            input = TRUE;
            scr_rowcol(11,19);
            printf("ON");
        }
        break;
    case 'T':
        if(direct)
        {
            input = TRUE;
            scr_rowcol(18,0);
            printf(" ");
            printf(" ");
            printf(" ");
            printf(" ");
        }
        break;
    case 'D': /* take data during run */
        data_flag = !data_flag;
        scr_rowcol(16,19);
        if(data_flag)

```



```

        input = FALSE;
    }
}
else
{
    pos[joint] = convert(0,3) - 1024; *command*
    * and put them on the screen *
    scr_rowcol(5,29);
    printf(" %4x ",Gain[joint]);
    scr_rowcol(6,29);
    printf(" %4x ",alpha[joint]);
    scr_rowcol(7,29);
    printf(" %4x ",beta[joint]);
    scr_rowcol(8,29);
    signext(pos[joint]);
    printf(" %d ",pos[joint]);
    scr_rowcol(9,30);
    printf(" %s",algorithm[alg[joint]]);
    temp = peek((joff/2)+0xC,jsegm);
    scr_rowcol(15,36);
    printf(" %3x",temp);
    temp = peek((joff/2)+0xE,jsegm)&0x1;
    scr_rowcol(16,36);
    if(temp) printf("rev");
    else printf("for");
    temp = peek(joff+0x8,jsegm)/(peek(joff+0x9,jsegm) << 8);
    scr_rowcol(17,36);
    printf(" %4x",temp);
}

alg[joint] = 1;
* time to start the run *
if(transmit)
{
    scr_rowcol(12,20);
    printf("Transmitted*ANY KEY*");
    scr_rowcol(13,30);
    printf("TO EXIT*");
    logint(onecom,pos[joint]);
    scr_rowcol(15,28);
    printf("Duty:");
    scr_rowcol(16,28);
    printf("Dir:");
    scr_rowcol(17,28);
    printf("Cnt:");
    scr_rowcol(18,24);
    printf("count");
    scr_rowcol(19,24);
    printf("rate_x");
    scr_rowcol(20,24);
    printf("rate_y");
}
* input runtime variable to dual port ram, poking command position last *
tran_direct(jsegm,joff,joint);
* let it go until a key is hit *
while(!scr_csts())
{
    scr_rowcol(10,30); * put the present position on the screen *
    posjc = (peek(joff,jsegm) >> 4)+(peek(joff+1,jsegm) << 4);
    printf(" %d ",(signext(posjc)));
    temp = peek((joff/2)+0xC,jsegm);
    scr_rowcol(15,36);
    printf(" %3x",temp);
    temp = peek((joff/2)+0xE,jsegm)&0x1;
    scr_rowcol(16,36);
    if(temp) printf("rev");
    else printf("for");
    temp = peek(joff+0x8,jsegm)/(peek(joff+0x9,jsegm) << 8);
    scr_rowcol(17,36);
    printf(" %4x",temp);
    counter=peek((joff+0x10,jsegm)+(peek(joff+0x11,jsegm)<<8);
    scr_rowcol(18,32);
    printf(" %d",counter);
    scr_rowcol(19,32);
    * put rate_n onto screen *
    while(peek(FROMZ80,jsegm) == 'U');
    poke('R',FROMZ80,jsegm);
    temp = peek(0x04,jsegm)/(peek(0x05,jsegm)<<8);
    temp1 = peek(0x06,jsegm)/(peek(0x07,jsegm)<<8);
    poke('O',FROMZ80,jsegm);
    printf(" %d ",temp);
    scr_rowcol(20,32);
    printf(" %d ",temp1);
    ***data logging***
    if(data_flag)
    {
        logint(cntt,counter); * keep the counter *
        logint(onecom,pos[joint]); * keep commanded position *
        logint(onejnt,posjc); * and the output arm position *
    }
}
* zero out any residual commands jc3*
while(peek(FROMZ80,jsegm) == 'U');poke('R',FROMZ80,jsegm);
while(peek(TOZ80,jsegm) == 'R');    poke('U',TOZ80,jsegm);

```

```

        poke(peek(0x00+joff,jsegm),joff+0x80,jsegm);
poke(peek(0x01+joff,jsegm),0x81+joff,jsegm);
        poke(0,0x88+joff,jsegm);    poke(0,0x89+joff,jsegm);
        poke(0,0x84+joff,jsegm);    poke(0,0x85+joff,jsegm);
        poke(0,0x8C+joff,jsegm);    poke(0,0x8D+joff,jsegm);
        poke(0,0x9C+joff,jsegm);    poke(0,0x9D+joff,jsegm);
        poke(0,0xA0+joff,jsegm);    poke(0,0xA1+joff,jsegm);
        poke('N',TOZ80,jsegm);
        poke('O',FROMZ80,jsegm);
        transmit = FALSE;
        scr_rowcol(12,20);
        printf(" ");
        scr_rowcol(13,30);
        printf(" ");
        scr_rowcol(15,28);
        printf(" ");
        scr_rowcol(16,28);
        printf(" ");
    }
}

screen();
screen2();
}

time_loop(seg)
int seg;
{
    char time[9];
    long cnt, end;
    while(peek(FROMZ80,seg) == 'U');
    cnt = peek(0x10,seg) + (peek(0x11,seg) << 8) + (peek(0x12,seg) << 16);
    times(time);
    scr_clr;
    scr_rowcol(5,5);
    printf("loop counter:  %d",cnt,"\\n");
    printf("time:         %s",time,"\\n");
    end = cnt + 100000;
    while(cnt < end)
        cnt = peek(0x10,seg) + (peek(0x11,seg) << 8) + (peek(0x12,seg) << 16);
    times(time);
    scr_rowcol(8,5);
    printf("loop counter:  %d",cnt,"\\n");
    printf("time:         %s",time,"\\n");
    printf("type any key to exit");
    while(!scr_csts());
}*/

```

```

/* SUPPORT2.C -- BAT support routines    version 6.4
** by John Spofford, Herbie Viggh, and David Anderson
** M.I.T. Space Systems Laboratory
*/
#include "PORTNUMS.H"
/*#define MASBUFLIM 100 * # of buffer checks before timeout */
/*#define MASREDLIM 100 * # of ready checks before timeout */
/*-----*/
/* tran_mast -- transmit buffers to master control card */
tran_mast(mdata,mcomm,maddr,mplast)
char mdata[],mcomm[],maddr[];
int mlast;

{
    int retval, waitc; /* timeout variable */
    int i; /* loop index */

    puts("Using old tran_mast\n");          /* use tran_jcs_dpr! */
    return(0);
    retval = 0;
    /* master buffer full? if so, error */
    if (inport(P1A) & 0x40)
    {
        printf("*** MASTER BUFFER ERROR AT START ***\n");*
        retval = -1; /* -1 is start buffer full error */
    }

    /* master has hung joint card? if so, error */
    if (inport(P1A) & 0x80)
    {
        printf("*** MASTER HAS HUNG JOINT CARD ***\n");*
        retval = -2; /* -2 is hung joint card error */
    }

    for (i = 0; i <= mlast; i++) /* transmit to master */
    {
        /* master buffer full? try timeout */
        for (waitc = 0; inport(P1A) & 0x40; waitc++)
            if (waitc > MASBUFLIM)
            {
                printf("*** MASTER BUFFER TIMEOUT ERROR ***\n");*
                retval = -3; /* -3 is buffer full timeout error */
                break;
            }

        /* is master ready? try timeout */
        for (waitc = 0; !(inport(P1A) & 0x20); waitc++)
            if (waitc > MASREDLIM)
            {
                printf("*** MASTER NOT READY FOR TRANSMIT ***\n");*
                retval = -4; /* -4 is ready timeout error */
                break;
            }

        if (mcomm[i] == 0) /* don't pass a zero command */
        {
            retval = -5;
            continue;
        }

        /* output to master */
        output(P3A,mdata[i]); /* data */
        output(P3B,mcomm[i]); /* command */
        output(P3C,maddr[i]); /* address (low 3) */
        output(P0X,C5LO); /* strobe lo */
        output(P0X,C5HI); /* and hi */
        /* end of transmit to master */

        output(P0X,C5HI); /* leave it high */
        return(retval); /* all is okay */
    }
}

/*-----*/
/* tran_puma -- transmit buffers to puma control card */
#define PUMLIM 23
#define PUMSYNC 0xFF
/*-----*/
tran_puma(pdat,plast)
char pdat[];
int plast;

{
    int waitc; /* timeout variable */
    int i; /* loop index */

    /* is puma ready? try timeout */
    for (waitc = 0; !(inport(P1A) & 0x02); waitc++)
        if (waitc > PUMLIM)
        {
            printf("*** PUMA NOT READY FOR TRANSMIT ***\n");*
            return(-2); /* -2 is ready timeout error */
        }

    /* send PUMA sync code */
    output(P1B,PUMSYNC); /* send sync */
    output(P0X,C3LO); /* strobe lo */
    output(P0X,C3HI); /* and hi */
}

```

```

for (i = 0; i <= plast; i++) /* transmit to puma */
{
    /* is puma ready? try timeout */
    for (waitc = 0; !(import(P1A) & 0x02); waitc++)
        if (waitc > PUMLLIM)
        {
            printf("*** PUMA NOT READY FOR TRANSMIT **\n"); /*
            return(-2); /* -2 is ready timeout error */
        }

    /* output to puma */
    output(P1B, pdat[i]); /* data */
    output(P0X, C3LO); /* strobe lo */
    output(P0X, C3HI); /* and hi */
} /* end of transmit to puma */

return(0); /* all is okay */
}

/*-----*/
/* adctest.c -- convert analog inputs using AD364 */
/* convert -- returns signed integer (-2048...2047)
** 9999 on error */
convert(ais_ch, mux_ch)
int ais_ch, mux_ch; /* the two input multiplexers */
{
    int high8, low4, value;

    if ((ais_ch > 15 || ais_ch < 0) || (mux_ch > 15 || mux_ch < 0))
        return(9999); /* error return big num */

    output(P0A, (ais_ch << 4) | mux_ch); /* ais high, mux low */

    /* start conversion (need 15us delay after setting muxes) */
    output(P0X, C1LO); /* A0 select 12 bit conversion */
    output(P0X, C2LO); /* R/C set to convert */
    output(P0X, C0HI); /* CE trigger conversion */
    output(P0X, C0LO); /* CE drop trigger */
    output(P0X, C2HI); /* R/C set for read */

    /* wait for end-of-conversion */
    while (import(P1A) & 1); /* STATUS */

    /* read data */
    output(P0X, C0HI); /* set CE for read */
    high8 = import(P0B); /* A0 already low */
    output(P0X, C1HI); /* A0 select low 4 bits of data */
    low4 = import(P0B) & 0xF0; /* in upper 4 bits */
    output(P0X, C0LO); /* reset CE low */

    /* massage & return */
    value = ((high8 << 4) | (low4 >> 4)); /* adjust to form a number */
    return(value-2048); /* return centered at zero */
}

/*-----*/
rqst_mstr(cardnum)
char cardnum;
{
    int retval, waitc; /* timeout variable */

    retval = 0;
    /* master buffer full? if so, error */
    if (import(P1A) & 0x40)
    {
        printf("*** MASTER BUFFER ERROR AT START **\n"); /*
        retval = -1; /* -1 is start buffer full error */
    }

    /* master has hung joint card? if so, error */
    if (import(P1A) & 0x80)
    {
        printf("*** MASTER HAS HUNG JOINT CARD **\n"); /*
        retval = -2; /* -2 is hung joint card error */
    }

    /* is master ready? try timeout */
    for (waitc = 0; !(import(P1A) & 0x20); waitc++)
        if (waitc > MASREDLIM)
        {
            printf("*** MASTER NOT READY FOR TRANSMIT **\n"); /*
            retval = -4; /* -4 is ready timeout error */
            break;
        }

    /* output to master */
    output(P3A, 0); /* data */
    output(P3B, 0); /* command */
    output(P3C, cardnum); /* address (low 3) */
    output(P0X, C5LO); /* strobe lo */
    output(P0X, C5HI); /* and hi */
    /* end of transmit to master */
    return(retval); /* all is okay */
}

```

```

/*-----*/
/* sync.c -- waits for time interrupt and then returns.
** it returns -1 if there was a time fault error */

#define T_L_OFF 0x6C /* offset of PC BIOS timer_low counter */
#define T_L_SEG 0x40 /* segment of PC BIOS timer_low counter */
/*-----*/
synchronize()
{
    static char last_low;
    char peek();

    if (peek(T_L_OFF, T_L_SEG) != last_low) /* look for error */
    {
        last_low = peek(T_L_OFF, T_L_SEG); /* set to new */
        return(TRUE); /* error if flag already set */
    }

    while (peek(T_L_OFF, T_L_SEG) == last_low); /* wait for new count */

    last_low = peek(T_L_OFF, T_L_SEG); /* set it to new */
    return(FALSE); /* back to work */
}

/*-----*/
int addr, data;
/*-----*/
import(a)
int a;
{
    extern int addr, data;

    addr = a;

#asm
; ASM88 /*
                MOV            DX, WORD addr_
                IN              AL, DX                ;get data
                MOV            AH, 0                ;char value
                MOV            WORD data_, AX

;*/
#
    return(data);
}

/*-----*/
output(a,d)
int
{
    a, d;
    {
        extern int addr, data;

        addr = a;
        data = d;

#asm
; ASM88 /*
                MOV            DX, WORD addr_
                MOV            AX, WORD data_
                OUT            DX, AL                ;output char

;*/
#
    }

/*-----*/
unsigned offset16, segment16;
char byte8;
/*-----*/
char peek(o,s)
unsigned o,s;
{
    extern unsigned offset16, segment16;
    extern char byte8;

    offset16 = o;
    segment16 = s;

#asm
; ASM88 /*
                MOV            DI, WORD offset16_
                MOV            ES, WORD segment16_
                MOV            AL, ES:[DI]            ;peek
                MOV            byte8_, AL            ;return it

;*/
#
    return(byte8);
}

/*-----*/
poke(c,o,s)
char c;
unsigned o,s;
{
    extern unsigned offset16, segment16;
    extern char byte8;

    offset16 = o;
    segment16 = s;
    byte8 = c;

#asm

```

```

; ASM88 /*
        MOV             AL, byte8_
        MOV             DI, WORD offset16_
        MOV             ES, WORD segment16_
        STOSB           ;poke
;*/
#
}
/*-----*/
/*-----*/
adjust(tdat, i) /*clip & adjust position values*/
int tdat, i;
{
    if((tdat & 0x800) != 0) /*sign extend*/
        tdat = tdat | 0xF000;

    if (tdat > 1024)
        return(1024);
    else if (tdat < -1024)
        return(-1024);
    else
        return(tdat);
}
/*-----*/
align(dater, ix) /* offset and scale from master to real arm */
int dater, ix;
{
    int icy;
    extern int align_offset[8];
    extern float align_gain[8];
    extern int max_jcs_val[8], min_jcs_val[8];
    extern int masoff[5]; /* master offset */
    extern float masmul[5]; /* master mult */

    dater = align_gain[ix] * (dater + align_offset[ix]);
    if (dater > max_jcs_val[ix])
        return(max_jcs_val[ix]);
    else if (dater < min_jcs_val[ix])
        return(min_jcs_val[ix]);
    else
        return(dater);
}
/*-----*/
/* read control station switches - return via external cswitch[i]
   TRUE = switch activated. comps for swapped leads on blue */
/*-----*/
char cswitch[16];
char old8, old9, old10, old11;
char initflag = TRUE;
read_switches()
{
    char i, x, mask;
    int tempc;
    extern char cswitch[16];
    extern char old8, old9, old10, old11;
    extern char initflag;
}
/*


| I  | O (F) | 1 (T) | Location | Function (run_bat)                   |
|----|-------|-------|----------|--------------------------------------|
| 0  | on    |       | off      | panel: red shutdown & exit           |
| 1  | on    |       | off      | panel: white slave toggle            |
| 2  | off   |       | on       | panel: blue toggle PUMA enable       |
| 3  | on    |       | off      | panel: black master flying           |
| 4  | on    |       | off      | arm: out/pull(1B) connect head       |
| 5  | on    |       | off      | arm: out/push(1A) toggle left claw   |
| 6  | on    |       | off      | arm: in/pull(2B) master connect      |
| 7  | on    |       | off      | arm: in/push(2A) toggle right claw   |
| 8  | on    |       | off      | foot: far left Beam carrier toggle   |
| 9  | on    |       | off      | foot: near left Beam carrier claw    |
| 10 | on    |       | off      | foot: near right head flying toggle  |
| 11 | on    |       | off      | foot: far right                      |
| 12 | on    |       | off      | hand contr:right spare pnuematics 1  |
| 13 | on    |       | off      | hand contr:middle spare pnuematics 2 |
| 14 | on    |       | off      | hand contr:left                      |
| 15 | on    |       | off      | hand contr:trigger                   |


Modified from direct i/o port reading of switches to scanning through a 16 to
1 mux. - 10/86 DEA
*/
/* tempc = inport(P4C); ****direct i/o port code****
for (i = 0; i < 8; i++)
    cswitch[i] = !((tempc >> i) & 1);
if (cswitch[2]) cswitch[2] = 0;
else cswitch[2] = 1;
return(tempc); */
/*serial port reader*/
tempc = 0;
for (i=0; i<16; i++) /*scan through all 16 switches*/
{
    /* switch address, i, is sent to P3C3-6. Note that P3C0-2 are
    initialized to 0, but are used by the feedback() function.*/

    outport(P3C, ((i<<3) & 0x78)); /*now send addresses with strobe lowered */
    x = (inport(P1A) >> 2) & 0x01;
}

```

```

        if (i==2)
            x = !x;

        /*read the switch - output is at P1A2 so shift by 2 bits and mask to
        cut out garbage. Note that output is inversion of switch state.
        When switch i is grounded, output, x, is high. Switch 2 leads are
        backwards, so we invert x there.*/
        /* installed 11/86 due to high beam foot switches, which are not impulse
        switches like the rest - DEA*/
        if (initflag)
        {
            if (i==8) old8 = x;
            if (i==9) old9 = x;
            if (i==10) old10 = x;
            if (i==11)
            {
                old11 = x;
                initflag = FALSE;
            }
        }

        if (i==9)
        {
            if (x != old9)
            {
                old9 = x;
                x = 1;
            }

            else
            {
                old9 = x;
                x = 0;
            }
        }

        if (i==10)
        {
            if (x != old10)
            {
                old10 = x;
                x = 1;
            }

            else
            {
                old10 = x;
                x = 0;
            }
        }

        if (i==11)
        {
            if (x != old11)
            {
                old11 = x;
                x = 1;
            }

            else
            {
                old11 = x;
                x = 0;
            }
        }

        if (i==8)
        {
            if (x != old8)
            {
                old8 = x;
                x = 1;
            }

            else
            {
                old8 = x;
                x = 0;
            }
        }

        cswitch[i] = (x & 0x01); /*set appropriate cswitch */
        tempc |= (x & 0x01)<<i;
    }

    return(tempc);
}

/*-----*/
/* This function reads the five pots on the head controller. - HV */

read_head()
{
    int i, tdat;
    extern int potlabelx[5], potlabely[5];
    extern int h_pot[5];

    for (i=0; i<=4; i++)
    {
        tdat = (convert(potlabelx[i],potlabely[i]) & 0xFF) - 1024;
        h_pot[i] = tdat;
        if (h_pot[i] > 1024)
            h_pot[i] = 1024;
    }
}

```

```

                else if (h_pot[i] < -1024)
                    h_pot[i] = -1024;
            }
        }
    }
}

/*-----*/
/* This function calculates jcs commands for the tilt an pan from the
head controller pots. - HV revised DEA */

head_cam_slave1()
{
    int i, t_cpitch, t_cyaw;
    extern int h_pot[5], headoff[5];
    extern int cpitch, cyaw, pos[9];
    extern float cpit_mult, cyaw_mult;
    extern int max_jcs_val[8], min_jcs_val[8];

    for (i=0; i<5; i++)
        h_pot[i] = h_pot[i] - headoff[i];

    t_cpitch = h_pot[3] - h_pot[1];
    t_cyaw = h_pot[4] + h_pot[0];
    cpitch = t_cpitch * cpit_mult;
    cyaw = t_cyaw * cyaw_mult;

    if (cpitch > max_jcs_val[1])
        cpitch = max_jcs_val[1];
    else if (cpitch < min_jcs_val[1])
        cpitch = min_jcs_val[1];
    if (cyaw > max_jcs_val[2])
        cyaw = max_jcs_val[2];
    else if (cyaw < min_jcs_val[2])
        cyaw = min_jcs_val[2];
    pos[7] = (cpitch) & 0xFFFF;
    pos[8] = cyaw & 0xFFFF;
}

/*-----*/
/* This function calculates the PUMA commands for flying BAT rotations
from the head controller. - HV */

head_puma_slave1()
{
    int i, t_ppitch, t_pyaw, t_proll;
    extern h_pot[5], p_headoff[5];
    extern float ppit_mult, pyaw_mult, prol_mult;
    extern int ppitch, pyaw, proll;

    for (i=0; i<5; i++)
        h_pot[i] = h_pot[i] - p_headoff[i];

    t_ppitch = h_pot[3] - h_pot[1];
    t_pyaw = h_pot[4] + h_pot[0];
    t_proll = h_pot[2];

    ppitch = t_ppitch * ppit_mult;
    pyaw = t_pyaw * pyaw_mult;
    proll = t_proll * prol_mult;
}

/*-----*/

```

```

/* SUPPORT3.C -- BAT support routines      version 6.4
** by John Spofford, M.I.T. Space Systems Laboratory
** revisions by David E. Anderson
*/
#include "PORTNUMS.H"
#include "DATA.H"
#define PI 3.1415926536
#define PI2 1.5707963268
#define R2D 180.0/PI
#define D2R PI/180.0
/*-----*/
char jcs_code[2048];
unsigned load_addr;
unsigned total;
char jc_stat[8] = {FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE};
DATATAG jc2pc[TAGSIZE] = {"KIBJC2 pos com"};
DATATAG jc2pm[TAGSIZE] = {"MIBJC2 pos meas"};
/*-----*/
load_control_system()
{
    int i, j, k;
    extern char jc_stat[];

    for (i = 1; i <= 4; i++)
    {
        if (load_dualram(i))
        {
            printf("\n%c Dual-port RAM loading failed (JC-%d)\n",7,i);
            for (j = 0; j <= 10; j++)
                dawdle(' '); /* time to read message */
        }
        else
            jc_stat[i] = TRUE; /* card is OK */
    }

    return(FALSE);
}
/*-----*/
load_jcs_buffer(fp)
int fp;
{
    extern char jcs_code[2048];
    extern unsigned load_addr;
    extern unsigned total;
    int i, byte, count, namecnt, code_ptr;
    char last_block;

    r16(fp);
    namecnt = r8(fp)*256 + r8(fp);
    for (i = 0; i < namecnt; i++)
        r8(fp);

    r8(fp);
    if (r8(fp) != 0xA0)
    {
        printf("\nNot a Z80-type file.\n");
        close(fp);
        return(TRUE);
    }

    last_block = FALSE;
    code_ptr = 0;
    if ((count = r16(fp)) < 257)
        last_block = TRUE;

    count -= 5;
    total = count;
    r16(fp);
    load_addr = r16(fp);
    for (i = 0; i < count; i++)
        jcs_code[code_ptr++] = r8(fp);

    if (!last_block)
        while (!last_block)
        {
            r16(fp);
            if ((count = r16(fp)) < 257)
                last_block = TRUE;

            count -= 5;
            total += count;
            r16(fp);
            r16(fp);
            for (i = 0; i < count; i++)
                jcs_code[code_ptr++] = r8(fp);
        }

    if (code_ptr != total)
        printf("\ncode_ptr != total"); /* just checking */
    if ((byte = code_ptr % 16) != 0)
        for (i = 0; i < (16-byte); i++)
            jcs_code[code_ptr++] = 0; /* even 16-byte blocks */

    total = code_ptr;
    close(fp);
    return(FALSE);
}
/*-----*/

```

```

r8(fp)
int fp;

{
    int byte;

    if ((byte = getc(fp)) != -1)
        return(byte);
    printf("\n%cError reading file\n", 7);
    close(fp);
    exit(0);
}

/*-----*/
r16(fp)
int fp;

{
    return(r8(fp) + r8(fp)*256);
}

/*=====*/
load_dualram(jc)
int jc;

{
    int i, fp, errc;
    char *JC_file;
    unsigned jc_seg, u;
    extern char jcs_code[2048];
    extern unsigned load_addr;
    extern unsigned total;

    JC_file = "J1.OBJ";

    if ((fp = open(JC_file, 0)) == -1)
    {
        printf("\n--> joint code [J1.OBJ] not found\n");
        return(TRUE);
    }

    if (load_jcs_buffer(fp))
        return(TRUE);

    if (total > 0x7FC)
    {
        printf("\nCode > 7FC bytes\n");
        return(TRUE);
    }

    jc_seg = SEG_OFF + 0x80 * jc;
    printf("\nwaiting for JC ready...");
    errc = 0;
    while (peek(FROMZ80, jc_seg) != 'r') /* wait Z80 ready */
    {
        if (peek(FROMZ80, jc_seg) == 'm')
        {
            printf("\nJC-%d indicates bad memory\n", jc);
            return(TRUE);
        }
        dawdle('.'); /* check for BREAK */
        if (++errc > 10)
            return(TRUE);
    }

    printf("\nCopying into JCS memory\n");
    _lmove(total, jcs_code, _showds(0, 0, jc_seg); /* use canned move */
    for (u = 0; u < total; u++) /* test for errors */
    {
        errc = 0;
        for (i = 0; i < 3; i++) /* three tries */
            if (peek(u, jc_seg) != jcs_code[u])
                errc++;

        if (errc > 1) /* retry if too many errs */
        {
            poke(jcs_code[u], u, jc_seg);
            if (peek(u, jc_seg) != jcs_code[u])
            {
                /* give up */
                printf("*** Error at %04x\n", u);
                return(TRUE);
            }
        }
    }

    poke('c', TOZ80, jc_seg); /* code loaded flag */
    printf("\nwaiting for code running flag...");
    errc = 0;
    while (peek(FROMZ80, jc_seg) != 'f') /* wait code running flag */
    {
        dawdle('.'); /* check for BREAK */
        if (++errc > 5)
            return(TRUE);
    }

    return(FALSE);
}

/*=====*/
dawdle(ch)

```

```

char ch;
{
    putchar(ch);
    synchronize();
    synchronize();
    synchronize();
    synchronize();
    synchronize();
}

/*=====
static char tran_slate[14][2] =          * {old com, new offs} X,Y *
{{0x80, 0x80}, {0x90, 0x82}            * reference D *
{0xE0, 0x84}, {0xF0, 0x86}            * gain D *
{0xC0, 0x88}, {0xD0, 0x8A}            * alpha D *
{0xA0, 0x8C}, {0xB0, 0x8E}            * beta D *
{0x01, 0x90}, {0x03, 0x91}            * algorithm *
{0x04, 0x92}, {0x05, 0x94}            * pre-dither *
{0x06, 0x96}, {0x07, 0x98}}; * post-dither *
*-----*
tran_jcs_dpr(d,c,a,last)
char d[], c[], a[];
int last;
{
    int i, n, jc, dat;
    char com, dh, dl, jc2_hit;
    unsigned jc_seg, jc_offs;
    extern char jc_stat[];
    int jc2com[2];
    extern char keep[], data_flag;

    jc2_hit = FALSE;
    for (n = 0; n <= last; n++)
    {
        jc = a[n];
        if (!jc_stat[jc])
            continue;
        jc_seg = SEG_OFF + jc * 0x80;
        com = c[n] & 0xF0;
        for (i = 0; i <= 7; i++)
        {
            if (com == tran_slate[i][0])
                break;
        }
        if (i <= 7)
        {
            jc_offs = (unsigned) tran_slate[i][1];
            dh = d[n];
            dl = (c[n] << 4) & 0xF0;
            dat = dl + (dh << 8);
            if (i <= 1)
            {
                jc2com[i] = dat;
                jc2_hit = TRUE;
            }
            while (peek(TOZ80, jc_seg) == 'R'); * wait for JC read *
            poke('U', TOZ80, jc_seg); * set Update flag *
            poke(dl, jc_offs + jc_seg);
            poke(dh, jc_offs, jc_seg);
            poke('N', TOZ80, jc_seg); * set New data flag *
        }
        else
        {
            com = c[n] & 0x0F;
            for (i = 8; i <= 13; i++)
            {
                if (com == tran_slate[i][0])
                    break;
            }
            if (i <= 13)
            {
                jc_offs = (unsigned) tran_slate[i][1];
                while (peek(TOZ80, jc_seg) == 'R'); * wait for JC read *
                poke('U', TOZ80, jc_seg); * set Update flag *
                poke(dl, jc_offs + jc_seg);
                poke('N', TOZ80, jc_seg); * set New data flag *
            }
            * otherwise ignore command *
        }
    }
    * if (data_flag && jc2_hit && keep[11])
    logint_a(jc2pc, jc2com);
}

/*-----*/
start_jc(n)
int n;
{
    unsigned jc_seg;
    int count, i;
    char gg;

    if (!jc_stat[n])

```

```

        return(TRUE); /* can't start a dead JC */
jc_seg = SEG_OFF + 0x80 * n;
for (i = 0x80; i < COMM_FLAG; i++)
    poke(0,i,jc_seg); /* clear command area */
for (count = 0; peek(JC_STATUS_jc_seg) != 'W'; count++)
{
    /* JC should set Waiting flag */
    synchronize();
    if (count > 36) /* 2 sec limit */
    {
        shutdown_bat();
        printf("%c\nJC-%d not ready for start\n",7,n);
        exit(2);
    }
}
poke('G',COMM_FLAG_jc_seg); /* Go flag to JC */
if (peek(COMM_FLAG_jc_seg) != 'G')
{
    poke('G',COMM_FLAG_jc_seg); /* try again */
    gg = peek(COMM_FLAG_jc_seg);
}
for (count = 0; peek(JC_STATUS_jc_seg) != 'C'; count++)
{
    /* check for Controlling flag */
    synchronize();
    if (count > 18) /* 1 sec limit */
    {
        shutdown_bat();
        printf("%c\nJC-%d did not respond to start\n",7,n);
        dawdle(gg); dawdle(gg); dawdle(gg); dawdle(gg);
        exit(2);
    }
}
}

/*=====*/
log_joints()
{
    extern char keep[];
    int jc2meas[2];

    /* read back joint card */
    jc2meas[0] = peek(0,0xE900) + (peek(1,0xE900) << 8);
    jc2meas[1] = peek(2,0xE900) + (peek(3,0xE900) << 8);
    if (keep[13])
        logint_a(jc2pm,jc2meas);
}

/*=====*/
monitor_jc()
{
    int sel;

    while (TRUE)
    {
        sel = mon_jc_menu();
        switch (sel)
        {
            case 1:
            case 2:
            case 3:
            case 4:
                watch_jc(sel);
                break;

            case 0:
                return(TRUE);
                break;

            default:
                co(7); /* beep */
                break;
        }
    }
}

/*=====*/
watch_jc(n)
int n;
{
    unsigned jc_seg;
    char d, dl, dh, *oneb, *twob;
    long ld;

    oneb = "%02x";
    twob = "%02x%02x";
    scr_clr();
    scr_rowcol(0,0);
    puts("      - X - - Y - \n"); /* 0 */
    puts("\n"); /* 1 */
    puts("  COMMANDED POSITION  XXXX XXXX \n"); /* 2 */
    puts("  INDICATED POSITION  XXXX XXXX \n"); /* 3 */
    puts("  INDICATED RATE    XXXX XXXX \n"); /* 4 */
    puts("\n"); /* 5 */
    puts("      GAIN    XXXX XXXX \n"); /* 6 */
    puts("\n"); /* 7 */
    puts("\n"); /* 8 */
    puts("  CALCULATED CONTROL  XXXX XXXX \n"); /* 9 */
    puts("  DUTY CYCLE    XX  XX \n"); /* 10 */
}

```

```

puts("    DIRECTION    CCC CCC\n");/*11*/
puts("    LOOP COUNTER XXXXXX    \n");/*12*/
puts(" \n");/*13*/
puts(" \n");/*14*/
puts(" \n");/*15*/
puts("\n Hit any key to exit");

scr_rowcol(0,2);
printf("JC-%d",n);
jc_seg = SEG_OFF + 0x80 * n;
while (!scr_csts())
{
    /* com pos */
    dl = peek(0x80,jc_seg);          dh = peek(0x81,jc_seg);
    scr_rowcol(02,25);              printf(twob,dh,dl);
    dl = peek(0x82,jc_seg);          dh = peek(0x83,jc_seg);
    scr_rowcol(02,33);              printf(twob,dh,dl);

    /* ind pos */
    dl = peek(0x00,jc_seg);          dh = peek(0x01,jc_seg);
    scr_rowcol(03,25);              printf(twob,dh,dl);
    dl = peek(0x02,jc_seg);          dh = peek(0x03,jc_seg);
    scr_rowcol(03,33);              printf(twob,dh,dl);

    /* ind rat */
    dl = peek(0x04,jc_seg);          dh = peek(0x05,jc_seg);
    scr_rowcol(04,25);              printf(twob,dh,dl);
    dl = peek(0x06,jc_seg);          dh = peek(0x07,jc_seg);
    scr_rowcol(04,33);              printf(twob,dh,dl);

    /* gain */
    dl = peek(0x84,jc_seg);          dh = peek(0x85,jc_seg);
    scr_rowcol(06,25);              printf(twob,dh,dl);
    dl = peek(0x86,jc_seg);          dh = peek(0x87,jc_seg);
    scr_rowcol(06,33);              printf(twob,dh,dl);

    /* calc ctrl */
    dl = peek(0x08,jc_seg);          dh = peek(0x09,jc_seg);
    scr_rowcol(09,25);              printf(twob,dh,dl);
    dl = peek(0x0A,jc_seg);          dh = peek(0x0B,jc_seg);
    scr_rowcol(09,33);              printf(twob,dh,dl);

    /* duty cy */
    d = peek(0x0C,jc_seg);
    scr_rowcol(10,27);              printf(oneb,d);
    d = peek(0x0D,jc_seg);
    scr_rowcol(10,35);              printf(oneb,d);

    /* directn */
    d = peek(0x0E,jc_seg);          scr_rowcol(11,26);
    if (d)                          puts("REV");          else puts("FOR");
    d = peek(0x0F,jc_seg);          scr_rowcol(11,34);
    if (d)                          puts("REV");          else puts("FOR");

    /* loop ct */
    scr_rowcol(12,23);
    d = peek(0x12,jc_seg);          printf(oneb,d);
    d = peek(0x11,jc_seg);          printf(oneb,d);
    d = peek(0x10,jc_seg);          printf(oneb,d);
}

/*=====*/
mon_jc_menu()
{
    int query;

    scr_clr();
    scr_rowcol(2,0);
    printf("    MIT SSL BAT/ICS\n");
    printf("    Monitor Joint Card\n");
    printf("    1. WRIST A & B\n");
    printf("    2. SHOULDER P & Y\n");
    printf("    3. ELBOW & LEFT ARM\n");
    printf("    4. CAMERA T & P\n");
    printf("    0. EXIT\n");
    printf(" Enter Selection: ");
    scanf("%d",&query);
    return(query);
}

/*=====*/
read_joints(com,post)
/* reads positions and commands for all power modules. Note that output
is a 12 bit number left shifted into a 16 bit number. */
int com[8], post[8];
{
    char dl, dh;

    /* command pos */
    dl = peek(0x80,0xE880);          dh = peek(0x81,0xE880);
    com[0] = dl + (dh << 8);          /* wrist A */
    dl = peek(0x82,0xE880);          dh = peek(0x83,0xE880);
    com[1] = dl + (dh << 8);          /* wrist B */
    dl = peek(0x80,0xE900);          dh = peek(0x81,0xE900);
    com[2] = dl + (dh << 8);          /* sh pitch */
    dl = peek(0x82,0xE900);          dh = peek(0x83,0xE900);
    com[3] = dl + (dh << 8);          /* sh yaw */
    dl = peek(0x80,0xE980);          dh = peek(0x81,0xE980);
    com[4] = dl + (dh << 8);          /* elbow */
    dl = peek(0x82,0xE980);          dh = peek(0x83,0xE980);
}

```

```

com[5] = dl + (dh << 8); /* cooling pump */
dl = peek(0x80,0xEA00); dh = peek(0x81,0xEA00);
com[6] = dl + (dh << 8); /* camera tilt (pitch) */
dl = peek(0x82,0xEA00); dh = peek(0x83,0xEA00);
com[7] = dl + (dh << 8); /* camera pan (yaw) */

/* indicated pos */
dl = peek(0x00,0xE880); dh = peek(0x01,0xE880);
post[0] = dl + (dh << 8); /* wrist A */
dl = peek(0x02,0xE880); dh = peek(0x03,0xE880);
post[1] = dl + (dh << 8); /* wrist B */
dl = peek(0x00,0xE900); dh = peek(0x01,0xE900);
post[2] = dl + (dh << 8); /* sh pitch */
dl = peek(0x02,0xE900); dh = peek(0x03,0xE900);
post[3] = dl + (dh << 8); /* sh yaw */
dl = peek(0x00,0xE980); dh = peek(0x01,0xE980);
post[4] = dl + (dh << 8); /* elbow */
dl = peek(0x02,0xE980); dh = peek(0x03,0xE980);
post[5] = dl + (dh << 8); /* cooling pump */
dl = peek(0x00,0xEA00); dh = peek(0x01,0xEA00);
post[6] = dl + (dh << 8); /* camera tilt (pitch) */
dl = peek(0x02,0xEA00); dh = peek(0x03,0xEA00);
post[7] = dl + (dh << 8); /* camera pan (yaw) */
}

/*=====*/
display_arm()
{
    int allpos[8], armpos[5], dummy[8], i, v;
    float grip_x, grip_y, grip_z, angles[5];
    extern char *potname[];

    scr_clr();
    scr_rowcol(0,0);
    puts(" ENCODER\n");/*0*/
    puts(" HEX DEC JOINT\n");/*1*/
    for (i = 3; i < 8; i++)
    {
        scr_rowcol(i,18);
        puts(potname[i]);
        scr_rowcol(i + 7,5);
        printf("%c%c",233,i - 3 + '1');
    }
    scr_rowcol(19,4);
    puts("Hit any key to exit");
    scr_rowcol(9,10);
    puts("ANGLE (DEG)");

    while (!scr_csts())
    {
        read_joints(dummy,allpos);
        armpos[0] = allpos[0];
        armpos[1] = allpos[1];
        armpos[2] = allpos[4];
        armpos[3] = allpos[3];
        armpos[4] = allpos[2];
        enc2tht(armpos,angles);
        bat_for(angles,&grip_x,&grip_y,&grip_z);
        for (i = 0; i < 5; i++)
        {
            scr_rowcol(3+i,0);
            v = armpos[i];
            printf(" %04x %6d",v,v);
            scr_rowcol(10+i,10);
            printf("%6.2f",angles[i] * R2D);
        }
        scr_rowcol(16,4);
        printf("X = %5.1f Y = %5.1f Z = %5.1f",grip_x,grip_y,grip_z);
    }
}

/*=====*/

```

```

/* SUPPORT4.C -- BAT support routines    version 6.4
** by John Spofford and David E. Anderson, M.I.T. Space Systems Laboratory
***/
#include "PORTNUMS.H"
/*-----*
adjust_pots()
{
    int number, newval, i, temp, fn, iii;
    char tempc, done, done2;
    extern int div_or_mul[8], p_gain[8], max_p[8], min_p[8];
    extern char icnv1[8], icnv2[8];
    extern char *potname[8];
    extern float align_gain[8];
    extern int align_offset[8];

    done = FALSE;
    while(!done)
    {
        scr_clr();
        scr_rowcol(3,0);
        printf("    ICS Pot Adjustments\n\n");
        for (i = 0; i < 8; i++)
            printf("    %1d %s\n", i+1, potname[i]);
        printf(" S Save to file\n");
        printf(" L Load from file\n");
        printf("[CR] Exit\n");
        printf("\nEnter Selection: ");
        tempc = scr_ci();
        switch(tempc)
        {
            case '13':
                done = TRUE;
                break;

            case 'S':
            case 's':
                if ((fn = creat("ICS.DAT")) == -1)
                {
                    beep(25);
                    printf("UNABLE TO WRITE FILE\n");
                    pause(250);
                }
                else
                {
                    for (i = 0; i < 8; i++)
                    {
                        putw(div_or_mul[i], fn);
                        putw(p_gain[i], fn);
                        putw(max_p[i], fn);
                        putw(min_p[i], fn);
                        putc(icnv1[i], fn);
                        putc(icnv2[i], fn);
                    }

                    close(fn);
                    break;
                }

            case 'L':
            case 'l':
                if ((fn = open("ICS.DAT", 0)) == -1)
                {
                    beep(25);
                    printf("UNABLE TO OPEN FILE\n");
                    pause(250);
                }
                else
                {
                    for (i = 0; i < 8; i++)
                    {
                        div_or_mul[i] = getw(fn);
                        p_gain[i] = getw(fn);
                        max_p[i] = getw(fn);
                        min_p[i] = getw(fn);
                        icnv1[i] = getc(fn);
                        icnv2[i] = getc(fn);
                    }

                    close(fn);
                    break;
                }

            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
                iii = tempc - '1';
                scr_clr();
                scr_rowcol(1,0);
                printf("Joint: %s\n", potname[iii]);
                printf("Current Settings\n");
                printf("\n");
                printf("1--Maximum Position: %5d\n", max_p[iii]);
                printf("2--Minimum Position: %5d\n", min_p[iii]);
                printf("3--Pot Shifts: %5d\n", p_gain[iii]);
                printf("4--Shift Direction: ");

```

```

if(div_or_mul[iii])
    printf("RIGHT\n");
else
    printf("LEFT\n");
printf("\n\n");
printf("5--Pot: convert");
printf("%2d, %2d)\n", icnvt2[iii], icnvt[iii]);
printf("(CR)--Exit");
done2 = FALSE;
while(!done2)
{
    scr_rowcol(18,0);
    printf("Enter Selection: ");
    scr_rowcol(18,18);
    while((tempc = scr_csts()) == 0)
    {
        scr_rowcol(15,10);
        temp = (convert(icnvt2[iii], icnvt[iii]) & 0xFFF);
        printf("Adjusted pot: %5d", adjust(temp - 1024, iii));
    }
    scr_rowcol(18,17);
    printf("%c", tempc);
    scr_rowcol(21,0);
    printf(" ");
    switch(tempc)
    {
        case 13:
            done2 = TRUE;
            break;

        case '1':
            max_p[iii] = get_new_val(4,31);
            break;

        case '2':
            min_p[iii] = get_new_val(5,31);
            break;

        case '3':
            p_gain[iii] = get_new_val(6,31);
            break;

        case '4':
            get_new_val(21,16);
            div_or_mul[iii] = !div_or_mul[iii];
            scr_rowcol(7,31);
            if(div_or_mul[iii])
                printf("LEFT ");
            else
                printf("RIGHT");
            break;

        case '5':
            temp = get_new_val(10,31);
            newval = get_new_val(10,31);
            scr_rowcol(21,0);
            printf(" ");
            scr_rowcol(10,31);
            printf("( %2d, %2d )", temp, newval);
            icnvt2[iii] = temp;
            icnvt[iii] = newval;
            break;

        case '6':
            scr_rowcol(21,0);
            printf("Enter offset value (int)");
            scanf("%d\n", &align_offset[iii]);
            scr_rowcol(22,0);
            printf("off: %4d, ", align_offset[iii]);
            printf("gain: %8.5f", align_gain[iii]);
            break;

        case '7':
            scr_rowcol(21,0);
            printf("Enter gain value (float)");
            scanf("%f\n", &align_gain[iii]);
            scr_rowcol(22,0);
            printf("off: %4d, ", align_offset[iii]);
            printf("gain: %8.5f", align_gain[iii]);
            break;

        default:
            scr_rowcol(22,0);
            printf("%cInvalid Resp., Please Reenter", 7);
            pause(100);
            scr_rowcol(22,0);
            printf(" ");
            break;
    }
}
break;
default:
    scr_rowcol(22,0);
    printf("%cInvalid Entry, Please Re-Enter", 7);
    pause(100);
    scr_rowcol(22,0);
    printf(" ");
    scr_rowcol(15,17);
    printf(" ");
    scr_rowcol(15,17);

```

```

        break;
    }
}

*-----*
get_new_val(wor,loc)
int wor, loc;
{
    int ret_val;

    scr_rowcol(21,0);
    printf("Enter New Value");
    scr_rowcol(wor,loc);
    printf(" ");
    scr_rowcol(wor,loc);
    scanf("%d",&ret_val);
    return(ret_val);
}

*-----*/
/* stat --- check status */
/* JC_STATUS: C = controlling, W = waiting */
stat()
{
    char statword;
    char jc;
    unsigned jseg;
    extern char jc_stat[];

    /* is PUMA ready? */
    statword = inport(P1A);
    scr_rowcol(6,21);
    if (statword & 0x02)
        scr_co(2);
    else
        scr_co(25);

    for (jc = 0; jc <= 7; jc++)
        if (jc_stat[jc])
        {
            /* if jc loaded */
            jseg = SEG_OFF + jc * 0x80;
            statword = peek(JC_STATUS,jseg);
            scr_rowcol(5,16 + jc);
            scr_co(statword);
        }
}

*-----*/
/* batports.c -- initialize PC ports for BAT
** */

/* bat_ports.c -- sets the PPI ports for BAT control */
bat_ports()
{
    setport(0,0,1,0); /* 0A:out 0B:in 0C:out */
    setport(1,1,0,0); /* 1A:in 1B:out 1C:out */
    setport(2,1,1,0); /* 2A:in 2B:in 2C:out */
    setport(3,0,0,0); /* 3A:in/out 3B:out 3C:out */
    setport(4,1,1,0); /* 4A:in 4B:in 4C:out */
}

*-----*/
/* setport.c -- sets a 8255's ports to input or output mode */
setport(portnum,a,b,c)
int portnum,a,b,c;
{
    char cw; /* mode control word */
    int addr; /* port address */

    if (portnum < 0 || portnum > 4)
        return (-1); /* on an error */

    if (a == 0) /* port A */
        cw = 0x80;
    else
        cw = 0x90;

    if (b != 0) /* port B */
        cw = cw + 2;

    if (c != 0) /* port C */
        cw = cw + 4;

    addr = 0x303 + (portnum << 2); /* addr range is 300..313 hex */
    outport(addr,cw); /* control ports are 303,307,...,313 */
    return (0);
}

*-----*/
/* screen.c -- sets up BAT video background. 1/8/84 jrs */
screen()
{
    scr_setmode(0);
    scr_setup();
    scr_cursoroff();

    puts("");
    printf(" THRUSTERS %cffffffff%c ff ACTUATORS ff ",218,191); /* 0 */
}

```

```

puts("                >BATSOFT>                ");/* 1 */
puts("                > V 6.4 >                ");/* 2 */
printf("                > %cffffff%c                ");/* 3 */
puts("                >                ");/* 4 */
puts("                JC:....                ");/* 5 */
puts("                PUMA                ");/* 6 */
puts("                >                ");/* 7 */
puts("                >                ");/* 8 */
puts("                > DRAW ...                ");/* 9 */
puts("                > HEAD ...                ");/* 10 */
puts("                > MAST ...                ");/* 11 */
puts("                > SLAVE ...                ");/* 12 */
puts("                > FLYING:                ");/* 13 */
puts("                > .....                ");/* 14 */
puts("                > RCLW .....                ");/* 15 */
puts("                > LCLW .....                ");/* 16 */
puts("                > BMCR .....                ");/* 17 */
puts("                > BCLW .....                ");/* 18 */
printf(" X Y Z %c %c                ",232,233,234);/*19*/
puts("                W W S S E C C ");/* 20 */
puts("                A B P Y L T P ");/* 21 */
printf("                command:%c actual:%c ",233,4);/* 22 */
puts("                ");/* 23 */
puts(" MIT SSL Beam Assembly Teleoperator ");/* 24 */

}

/*-----*/
screen2()
{
extern char draw,puma_on,head_controller,data_flag;
extern char boost_flag,head_ctrl_connected;
extern char master_connected,slave_arm_on,master_flying;
extern char head_ctrl_flying,right_claw,left_claw_state;
extern char beam_carrier,beam_claw;
extern int set_triad;
extern int tri_for,tri_rev;/* for successive triad use */
scr_rowcol(21,3);
puts("thruster");

scr_rowcol(09,19);
if (draw) puts("ON ");
else puts("OFF");

scr_rowcol(1,4);
if (puma_on) puts("ON ");
else puts("OFF");

scr_rowcol(08,15);
if (data_flag) puts("DATA");
else if(set_triad == 0) puts(" ");

scr_rowcol(07,15);
if (boost_flag) puts("BOOST");
else if(set_triad == 0) puts(" ");

scr_rowcol(10,19);
if (head_ctrl_connected) puts("ON ");
else puts("OFF");

scr_rowcol(11,19);
if (master_connected) puts("ON ");
else puts("OFF");

scr_rowcol(12,19);
if (slave_arm_on) puts("ON ");
else puts("OFF");

scr_rowcol(14,15);
if (master_flying) puts("MASTER");
else if(head_ctrl_flying) puts("HEAD ");
else puts("STICK ");

scr_rowcol(15,18);
if (right_claw) puts("CLOSE");
else puts("OPEN ");

scr_rowcol(16,18);
if (left_claw_state) puts("CLOSE");
else puts("OPEN ");

scr_rowcol(17,18);
if (beam_carrier) puts("BACK ");/*counterintuitive for Huntsville*/
else puts("OUT ");

scr_rowcol(18,18);
if (beam_claw) puts("CLOSE");
else puts("OPEN ");

scr_rowcol(23,0);
puts(" TRIAD Up Next: ");
if(tri_for == 1) puts("Grabbeam ");
if(tri_for == 2) puts("Grabslv ");
if(tri_for == 3) puts("Stow ");
if(tri_rev == 1) puts("Cjoint ");
if(tri_rev == 2) puts("Rgrabslv ");

```

```

        if(tri_rev == 3) puts("Rstow  ");
    }
}
/*-----*/
strange()
{
    int val2, val1;
    char i, signc;
    extern int align_offset[5];
    extern float align_gain[5];
    extern char icnv[8], icnv2[8];
    extern char *potname[8];

    scr_clr();
    scr_cursoff();
    scr_rowcol(1,0);
    puts("  BAT Master Arm Monitor\n");
    puts("  Joint  pot adjust align\n");
    /*.....0123456789a123456789b123456789c123456789..line*/
    for (i = 0; i < 8; i++)
        printf("%s\n %6.3f %5d\n", potname[i], align_gain[i], align_offset[i]);
    while (scr_csts() != 13)
        for (i = 0; i < 8; i++)
        {
            val2 = adjust((convert(icnv2[i], icnv[i]) & 0xFFFF) - 1024, i);
            val1 = align(val2, i) & 0xFFFF;
            signc = '-';
            if (val2 < 0)
                val2 = -val2;
            else
                signc = '+';
            scr_rowcol(i*2 + 5, 13);
            printf("(%2d,%2d)%c%04x%03x", icnv2[i], icnv[i], signc, val2, val1);
        }
}
/*-----*/
keyboard_command(tempc)
char tempc;
{
    extern char comm_disp;
    extern char data_flag; /* logical flags */
    extern char master_flying, slave_arm_on;
    extern char left_claw_state, master_connected;
    extern char head_controller, puma_on, draw;
    extern char beam_carrier, beam_claw;
    extern char spare1, spare2; /* spare pneumatics 1 and 2 */
    extern int p_gain[8];
    extern int set_triad;
    extern char icnv[8], icnv2[8];
    int i;
    tempc = toupper(tempc);
    switch (tempc)
    {
        case 'C': /* change camera control drive */
            printf("%c", 7); /* beep */
            scr_rowcol(14, 15);
            if(head_controller)
            {
                head_controller = FALSE;
                icnv[1] = 9;
                icnv2[2] = 10;
                p_gain[1] = 0;
                p_gain[2] = 0;
                puts("STICK");
            }
            else
            {
                head_controller = TRUE;
                icnv[1] = 5;
                icnv2[2] = 6;
                p_gain[1] = 0;
                p_gain[2] = 0;
                puts("HEAD");
            }
            break;
        case 'P': /* transmit to puma */
            printf("%c", 7);
            scr_rowcol(1, 4);
            puma_on = !puma_on;
            if(puma_on)
                puts("ON ");
            else
                puts("OFF");
            break;
        case 'D': /* start or stop datalogging */
            data_flag = !data_flag;
            screen2();
            break;
        case 'E': /* draw to screen */
            printf("%c", 7);
            draw = !draw;
            scr_rowcol(9, 19);
    }
}

```

```

        if(draw)
            puts("ON ");
        else
            puts("OFF");
        break;
case 'R': /* reset cs */
    output(P0X,C6HI); /* reset JCS */
    output(P2X,C7HI); /* reset propulsion controller */
    for (i = 0; i < 20; i++); /* pulse width */
    output(P0X,C6LO);
    output(P2X,C7LO);
    for (i = 0; i < 36; i++)
        synchronize(); /* let JCS check memory */
    for (i = 1; i <= 4; i++)
        start_jc(i); /* start card #1,2,3,4 */
    break;
case 'T': /* tune joints */
    setup_joints();
    screen();
    screen2();
    break;
/* excess fat removed
case 'I': * adjust right arm *
    adjust_pots();
    screen();
    screen2();
    break;
case 'B': * test feedback from BAT *
    feedback();
    screen();
    screen2();
    break;
case 'X':
    strange();
    screen();
    screen2();
    break;
case 'L':
    left_claw_state = !left_claw_state; /* toggle */
    if (left_claw_state)
        output(P2X,C1LO); /* close claw */
    else
        output(P2X,C1HI); /* open claw */
    co(7);
    screen2();
    break;
/*
case 'U':
    joint_step();
    screen();
    screen2();
    break;*/
case '?':
    print_help();
    screen();
    screen2();
    break;
case 'W': /* supervisory control calls setup menu for TRIAD */
    triad();
    screen();
    screen2();
    break;
case 'F':
    set_trim();
    screen();
    screen2();
    break;
case 'G':
    set_data();
    screen();
    screen2();
    break;
case 'J':
    monitor_jc();
    screen();
    screen2();
    break;
case 'A':
    display_arm();
    screen();
    screen2();
    break;
case 'Z':
    set_boost();
    screen();
    screen2();
    break;
case 'M':
    beam_carrier = !beam_carrier;
    if(beam_carrier)
    {
        output(P2X,C3HI); /*extend beam carrier*/
    }

```

```

else
    {
        output(P2X,C3LO); /*retract beam carrier*/
    }
co(7);
screen2();
break;
case 'N':
    beam_claw = !beam_claw;
    if(beam_claw)
    {
        output(P2X,C4LO); /*close beam carrier claw*/
    }
    else
    {
        output(P2X,C4HI); /*open beam carrier claw*/
    }
    co(7);
    screen2();
    break;
case 'S':
    if (slave_arm_on)
        slaverest(0);
    else
        slaverest(1);
    break;
case 'O':
    spare1 = !spare1; /*toggle spare pneumatics 1*/
    if (spare1)
    {
        output(P1X,C6LO);
    }
    else
    {
        output(P1X,C6HI);
    }
    co(7);
    break;
case 'Y':
    comm_disp = !comm_disp;
    break;
case 'Q':
    spare2 = !spare2; /*toggle spare pneumatics 2*/
    if (spare2)
    {
        output(P1X,C7LO);
    }
    else
    {
        output(P1X,C7HI);
    }
    co(7);
    break;
case 'V':
    if(set_triad==1)
    {
        co(7);
        point_to_point();
    }
    else if(set_triad==2)
    {
        co(7);
        learn();
    }
    break;
case 'K':
    load_from_file("zero.bdt ");
    break;
case '1':
    load_from_file("fredpid.bdt ");
    break;
case '2':
    load_from_file("wrtst.bdt ");
    break;
case '3':
    load_from_file("eltst.bdt ");
    break;
case '4':
    load_from_file("shptst.bdt ");
    break;
case '5':
    load_from_file("shytst.bdt ");
    break;
case '6':
    load_from_file("tptst.bdt ");
    break;
case '7':
    load_from_file("triad1.bdt ");
    break;
case '8':
    load_from_file("triad2.bdt ");
    break;

```

```

        case '9':
            load_from_file("triad3.bdt ");
            break;

        case '0':
            load_from_file("fred.bdt ");
            break;
    }

}

/*-----*/
print_help()
{
    scr_clr();
    scr_rowcol(0,0);
    puts(" BAT / ICS Keyboard Assignments (Main) ");
    puts(" (6.4) ");
    puts("A: display arm pos ≥S: slave rest/reset ");
    puts("B: cam head/stick ≥T: set joint cntrl ");
    puts("C: toggle data log ≥U: TRIAD activate ");
    puts("D: dis/enable draw ≥W: TRIAD setup ");
    puts("E: thrust trim ≥X: monitor master ");
    puts("F: record data ≥Y: show commands ");
    puts("G: boost arm/thrust ");
    puts("H: ≥Z: print this list ");
    puts("I: monitor joint cd≥1: load fredpid ");
    puts("K: kill jc duty cyc≥2: load wrtst ");
    puts("L: use left claw ≥3: load eltst ");
    puts("M: deploy beam cr ≥4: load shptst ");
    puts("N: use beam cr claw≥5: load shyttst ");
    puts("O: use spare pnue 1≥6: load tptst ");
    puts("P: dis/enable PUMA ≥7: load triad1 ");
    puts("Q: use spare pnue 2≥8: load triad2 ");
    puts("R: reset JCS ! ≥tab: shutdown/exit ");
    puts(" ");
    puts(" Type (CR) to return to main loop ");
    while (scr_csts() != 13);
}

/*-----*/
/*Slaverest is a function designed to cutout excess use of the right
arm motors. slaverest can also be used to change the master arm offsets
for smooth power-up.
(0) Power down slave arm. Gains to zero.
(1) Prepare to power up. Set offsets to reconcile master and slave.
(2) Offsets have been digested. Reset gains.
Dave Anderson 5/20/87 */

slaverest(state)
int state;
{
    extern int align_offset[8], curr_master[5], slave_toggle;
    extern char master_connected, master_flying, slave_arm_on;
    extern float align_gain[8];
    int com[8], posr[8], i, fn, temp[5];
    int ord[5] = {0, 1, 4, 3, 2};
    switch (state)
    {
        case 0:
            slave_arm_on = FALSE;
            screen2();
            /* save and zero out slave arm gains */
            write_out_data("slaverest.BDT ");
            load_from_file("slavezero.BDT ");
            break;

        case 1:
            slave_arm_on = TRUE;
            screen2();
            read_joints(com,posr);
            for (i=0;i<5;i++)
            {
                if ((posr[ord[i]] & 0x8000) != 0)
                    posr[ord[i]] = (posr[ord[i]] >> 4) | 0xF000;
                else
                    posr[ord[i]] = posr[ord[i]] >> 4;
                align_offset[i+3] = posr[ord[i]]/align_gain[i+3] - curr_master[i];
            }
            slave_toggle = 1;
            break;

        case 2:
            /* load gains to arm. */
            slave_toggle = 0;
            load_from_file("slaverest.BDT ");
            break;

        default:
            scr_rowcol(12,19);
            puts("ERg");
            break;
    }
}

/*-----*/
reset_pid()
{
    int temp;
    /* reset integrators */
    while(peek(TOZ80,0xE900)=='R'); poke('U',TOZ80,0xE900);
    poke(0,0xA1,0xE900);
    poke(0,0xA3,0xE900);
}

```

```

poke('N',TOZ80,0xE900);
while(peek(TOZ80,0xE880)=='R'); poke('U',TOZ80,0xE880);
poke(0,0xA1,0xE880);
poke(0,0xA3,0xE880);
poke('N',TOZ80,0xE880);
while(peek(TOZ80,0xE980)=='R'); poke('U',TOZ80,0xE980);
poke(0,0xA1,0xE980);
poke('N',TOZ80,0xE980);
/* once the joint cards have read the zero (set toz80 to 'O')*/
/* replace original mult1 value */
temp = peek(0x10,0xE880); /* read counter on jc1 */
while(temp == peek(0x10,0xE880)); poke('U',TOZ80,0xE900);
poke(1,0xA1,0xE900);
poke(1,0xA3,0xE900);
poke('N',TOZ80,0xE900);
poke('U',TOZ80,0xE880);
poke(1,0xA1,0xE880);
poke(1,0xA3,0xE880);
poke('N',TOZ80,0xE880);
poke('U',TOZ80,0xE980);
poke(1,0xA1,0xE980);
poke('N',TOZ80,0xE980);
}
/*-----*/

```

```

/*=====*/
/* SUPPORT5.c -- fakes SUPERVIS.c & MOVE.c 10/23/85 */
/*=====*/
#include "PORTNUMS.H"
#define FACTOR 0.002 /* for pot trim */
#define PDEDZON 80 /* for pot trim */
#include <math.h>
#define PI 3.1415926536
#define PI2 1.5707963268
#define R2D 180.0/PI
#define D2R PI/180.0
/*=====*/
supervis()
{
    co(7); /* beep */
}

/*=====*/
float a[6][6] = { /* adjustment matrix */
    { 0.67,0.0,0.0,0.0,0.0,0.0 }, { 0.0,0.67,0.0,0.0,0.0,0.0 },
    { 0.0,0.0,0.33,0.0,0.0,0.0 }, { 0.0,0.0,0.0,0.67,0.0,0.0 },
    { 0.0,0.0,0.0,0.0,0.67,0.0 }, { 0.0,0.0,0.0,0.0,0.0,0.33 } };
/*=====*/
trim_thrust(j,x) /* adjust thruster commands */
int j[], x[];
{
    int i, k;
    float in[6], out;

    for (i = 0; i < 6; i++)
        in[i] = (float) j[i];

    for (i = 0; i < 6; i++)
    {
        out = 0.0;
        for (k = 0; k < 6; k++)
            out += a[i][k]*in[k];
        x[i] = (int) out;
    }
}

/*=====*/
set_trim() /* set thruster trim matrix */
{
    int sel, i, j;
    float v;
    char pdat[7];

    for (i = 0; i < 6; i++)
        pdat[i] = 128; /* zero each command */
    tran_puma(pdat,5); /* Prevent thruster start-up */
    while (TRUE)
    {
        sel = trim_select();
        switch (sel)
        {
            case 1:
                load_trim();
                break;

            case 2:
                save_trim();
                break;

            case 3:
                puts("\n\nX=0, Y=1, Z=2, P=3, R=4, Y=5\n");
                puts(" I is thruster output\n");
                puts(" J is joystick input\n");
                puts("Enter I, J: ");
                scanf("%d,%d\n",&i,&j);
                printf("A[I][J] is %6.2f\n",a[i][j]);
                puts("Enter new A[I][J]: ");
                scanf("%f\n",&v);
                if ((i >= 0) && (i < 6) && (j >= 0) && (j < 6))
                    a[i][j] = v;
                else
                    co(7); /* beep */
                break;

            case 4:
                puts("\n\nX=0, Y=1, Z=2, P=3, R=4, Y=5\n");
                puts(" I is thruster output\n");
                puts(" J is joystick input\n");
                puts("\n\n");
                for (i = 0; i < 6; i++)
                    printf(" %d ",i);
                puts("\n");
                for (i = 0; i < 6; i++)
                {
                    printf(" %d:",i);
                    for (j = 0; j < 6; j++)
                        printf(" %5.2f ",a[i][j]);
                    printf("\n");
                }
                puts("\nHit any key...");
                getchar();
        }
    }
}

```

```

        case 5:
            break;

            puts("\nSELECT THRUST AXIS TO TRIM");
            puts("\nX=0, Y=1, Z=2, P=3, R=4, Y=5: ");
            scanf("%d\n",&i);
            if ((i >= 0) && (i < 6))
                joystick(i);
            else
                co(7); /* beep */
            break;
        case 6:
            show_thrust();
            break;
        case 0:
            return(TRUE);
            break;
        default:
            co(7); /* beep */
            break;
    }
}

/*-----*/
joystick(ax)
int ax;
{
    int thr_ctrl, thr_in[6], thr_out[6], i, temp;
    extern int stickoff[6];
    float pot[5];
    char pdat[7];

    scr_clr();
    scr_rowcol(15,5);
    puts("Hit any key to exit...");

    for (i = 0; i < 6; i++)
        thr_in[i] = 0; /* clear */
    scr_rowcol(2,0);
    switch (ax)
    {
        case 0: /* X */
            printf("X -| Z -| P -| R -| Y -|");
            break;
        case 1: /* Y */
            printf("Y -| X -| Z -| P -| R -| Y -|");
            break;
        case 2: /* Z */
            printf("Z -| X -| Y -| P -| R -| Y -|");
            break;
        case 3: /* P */
            printf("P -| X -| Y -| Z -| R -| Y -|");
            break;
        case 4: /* R */
            printf("R -| X -| Y -| Z -| P -| Y -|");
            break;
        case 5: /* Y */
            printf("Y -| X -| Y -| Z -| P -| R -|");
            break;
        default:
            co(7); /* beep */
            return(FALSE);
            break;
    }
    scr_rowcol(8,0);
    printf("THRUSTERS ARE ENABLED");

    while (!scr_csts())
    {
        synchronize(); /* not too fast */
        scr_rowcol(4,0);
        for (i = 0; i < 5; i++)
        {
            temp = convert(0,i) - 1024;
            if (temp < 0) /* deadzone */
                if (temp > -PDEDZON)
                    temp = 0;
                else
                    temp = temp + PDEDZON;
            else
                if (temp < PDEDZON)
                    temp = 0;
                else
                    temp = temp - PDEDZON;
            pot[i] = (float) temp * FACTOR;
            printf("%7.2f ",pot[i]);
        }

        switch (ax)
        {
            case 0: /* X */
                thr_in[0] = -(convert(7,0) - 1024 + stickoff[4]);
                a[1][ax] = pot[0];

```

```

        a[2][ax] = pot[1];
        a[3][ax] = pot[2];
        a[4][ax] = pot[3];
        a[5][ax] = pot[4];
        break;
    case 1:
        /* Y */
        thr_in[1] = -(convert(8,0) - 1024 + stickoff[5]);
        a[0][ax] = pot[0];
        a[2][ax] = pot[1];
        a[3][ax] = pot[2];
        a[4][ax] = pot[3];
        a[5][ax] = pot[4];
        break;
    case 2:
        /* Z */
        thr_in[2] = -(convert(6,0) - 1024 + stickoff[3]);
        a[0][ax] = pot[0];
        a[1][ax] = pot[1];
        a[3][ax] = pot[2];
        a[4][ax] = pot[3];
        a[5][ax] = pot[4];
        break;
    case 3:
        /* P */
        thr_in[3] = -(convert(0,9) - 1024 + stickoff[2]);
        a[0][ax] = pot[0];
        a[1][ax] = pot[1];
        a[2][ax] = pot[2];
        a[4][ax] = pot[3];
        a[5][ax] = pot[4];
        break;
    case 4:
        /* R */
        thr_in[4] = -(convert(0,10) - 1024 + stickoff[1]);
        a[0][ax] = pot[0];
        a[1][ax] = pot[1];
        a[2][ax] = pot[2];
        a[3][ax] = pot[3];
        a[5][ax] = pot[4];
        break;
    case 5:
        /* Y */
        thr_in[5] = (convert(0,8) - 1024 + stickoff[0]);
        a[0][ax] = pot[0];
        a[1][ax] = pot[1];
        a[2][ax] = pot[2];
        a[3][ax] = pot[3];
        a[4][ax] = pot[4];
        break;
    }
    trim_thrust(thr_in,thr_out);
    for (i = 0; i <= 5; i++)
    {
        temp = thr_out[i];
        temp = (temp > 127) ? (127) : (temp); /* limit */
        temp = (temp < -127) ? (-127) : (temp);
        pdat[i] = (temp + 128);
        if (pdat[i] == 0xFF)
            pdat[i] = 0xFE;
    }
    tran_puma(pdat,5);
}
for (i = 0; i < 6; i++)
    pdat[i] = 128; /* zero each command */
tran_puma(pdat,5); /* Prevent thruster start-up */
}

/*-----*/
show_thrust()
{
    int thr_in[6], thr_out[6], i;
    extern int stickoff[6];

    scr_clr();
    scr_rowcol(3,15);
    puts("RAW TRIM");
    scr_rowcol(15,5);
    puts("Hit any key to exit...");
    scr_rowcol(13,5);
    puts("Need FF80 < TRIM < 0080");
    while (!scr_csts())
    {
        thr_in[0] = -(convert(7,0) - 1024 + stickoff[4]); /*X*/
        thr_in[1] = -(convert(8,0) - 1024 + stickoff[5]); /*Y*/
        thr_in[2] = -(convert(6,0) - 1024 + stickoff[3]); /*Z*/
        thr_in[3] = -(convert(0,9) - 1024 + stickoff[2]); /*P*/
        thr_in[4] = -(convert(0,10) - 1024 + stickoff[1]); /*R*/
        thr_in[5] = (convert(0,8) - 1024 + stickoff[0]); /*Y*/
        trim_thrust(thr_in,thr_out);
        for (i = 0; i < 6; i++)
        {
            scr_rowcol(5 + i,15);
            printf("%04x %04x",thr_in[i],thr_out[i]);
        }
    }
}

```

```

/*-----*/
/*=====*/
trim_menu()
{
    scr_clr();
    scr_rowcol(2,0);
    printf(" M.I.T. SSL\n");
    printf(" Beam Assembly Teleoperator\n\n");
    printf(" Thruster Adjustment\n\n");
    printf(" 1. LOAD FROM FILE\n");
    printf(" 2. SAVE TO FILE\n");
    printf(" 3. SET VALUE IN MATRIX\n");
    printf(" 4. DISPLAY MATRIX\n");
    printf(" 5. TRIM JOYSTICKS w/POTS\n");
    printf(" 6. SHOW JOYSTICK TRIM\n");
    printf(" 0. EXIT\n");
    scr_rowcol(22,0);
    printf(" Enter Selection: ");
    return(0);
}

/*-----*/
trim_select()
{
    int query;

    trim_menu();
    scr_rowcol(22,19);
    printf(" ");
    scr_rowcol(22,19);
    scanf("%d",&query);
    return(query);
}

/*-----*/
load_trim()
{
    int i, k, fn;
    char fname[20];

    printf(" Load filename [.TDT]: ");
    scanf("%s\n",fname);
    strcat(fname, ".TDT");
    if ((fn = open(fname,0)) == -1)
    {
        scr_rowcol(22,0);
        /* beep(25); */
        printf("ERROR in loading");
        /* pause(250); */
    }

    else
    {
        for (i = 0; i < 6; i++)
        {
            for (k = 0; k < 6; k++)
            {
                fscanf(fn,"%f",&a[i][k]);
                fprintf(fn,"%n");
            }
            close(fn);
        }
        return(0);
    }
}

/*-----*/
save_trim()
{
    int fn, i, k;
    char fname[20];

    printf(" Write filename [.TDT]: ");
    scanf("%s\n",fname);
    strcat(fname, ".TDT");
    if ((fn = creat(fname)) == -1)
    {
        scr_rowcol(0,0);
        /* beep(25); */
        printf("Unable to Write File");
        /* pause(250); */
    }

    else
    {
        for (i = 0; i < 6; i++)
        {
            for (k = 0; k < 6; k++)
            {
                fprintf(fn,"%7.4f ",a[i][k]);
                fprintf(fn,"%n");
            }
            close(fn);
        }
    }
}

/*=====*/
calc_boost(jc,jp,tb)
int jc[5], jp[5], tb[6];

```

```

    {
        float comtht[5], postht[5], xc,yc,zc, xp,yp,zp;
        int zero;
        extern float boost_x, boost_y, boost_z;

        zero = 0;
        enc2tht(jc,comtht);
        bat_for(comtht,&xc,&yc,&zc);
        enc2tht(jp,postht);
        bat_for(postht,&xp,&yp,&zp);
        tb[0] = (int) (boost_x * (xp - xc));
        tb[1] = (int) (boost_y * (yp - yc));
        tb[2] = (int) (boost_z * (zp - zc));
        tb[3] = zero;
        tb[4] = zero;
        tb[5] = zero;
    }

/*=====*/
bat_for(t,x,y,z)
float t[5];
float *x, *y, *z;
{
    float c1, s1, c2, s2, c3, s3, c23, s23, c4, s4;
    extern float l1, l2, l3;

    c1 = cos(t[0]);
    s1 = sin(t[0]);
    c2 = cos(t[1]);
    s2 = sin(t[1]);
    c3 = cos(t[2]);
    s3 = sin(t[2]);
    c4 = cos(t[3]);
    s4 = sin(t[3]);
    c23 = c2 * c3 - s2 * s3;
    s23 = s2 * c3 + c2 * s3;

    *x = l1*c1*c2 + l2*c1*c23 + l3*(c1*c23*c4 - s1*s4);
    *y = l1*s1*c2 + l2*s1*c23 + l3*(s1*c23*c4 + c1*s4);
    *z = l1*s2 + l2*s23 + l3*s23*c4;
}

/*=====*/
float l1 = 16.0;
float l2 = 14.1;
float l3 = 3.5;

int shyoff = 1692;
int shpoff = -3760;
int elboff = -12656;
int wraoff = 2992;
int wrboff = -3925;
float shy2th1 = 6.41e-5;
float shp2th2 = 4.80e-5;
float elb2th3 = -6.00e-5;
float wry2th4 = 4.80e-5;
float wr2th5 = 5.15e-5;

/*=====*/
enc2tht(enc,tht)
int enc[5]; /* wra, wrb, elb, shy, shp */
float tht[5]; /* shy, shp, elb, wry, wr2 */
{
    int i, wra, wrb;

    wra = enc[0] + wraoff;
    wrb = enc[1] + wrboff;
    tht[0] = shy2th1 * (float) (enc[3] + shyoff);
    tht[1] = shp2th2 * (float) (enc[4] + shpoff);
    tht[2] = elb2th3 * (float) (enc[2] + elboff);
    tht[3] = wry2th4 * (float) (wra - wrb);
    tht[4] = wr2th5 * (float) (wra + wrb);
}

/*=====*/
float boost_x, boost_y, boost_z;
set_boost()
{
    extern float boost_x, boost_y, boost_z;
    char ch;
    float val;

    scr_clr();
    scr_rowcol(5,0);
    printf(" CURRENT BOOST GAINS: X %5.1f\n",boost_x);
    printf("                Y %5.1f\n",boost_y);
    printf("                Z %5.1f\n",boost_z);
    printf("\n ENTER NEW BOOST GAIN LIKE THIS:\n");
    printf(" x ff.f[CR] x = {X,Y,Z}\n ?");
    scanf("%c %f\n",&ch,&val);
    ch = toupper(ch);
    switch (ch)
    {
        case 'X':

```

```

        boost_x = val;
        break;
    case 'Y':
        boost_y = val;
        break;
    case 'Z':
        boost_z = val;
        break;
    default:
        co(7);
        break;
    }
}
/*=====*/

```

```

/*=====*/
/*      DATERS.C - Data logging and timing functions      */
/*      Functions:                                         */
/*      start_log()          logchar()                    */
/*      index_log()          logchar_a()                  */
/*      logint()             logfloat()                   */
/*      logint_a()           logfloat_a()                 */
/*      dump_log()                                                  */
/*      John Spofford          June, 1985, Jan 1986      */
/*=====*/
/* revised July 1987 DEA                                     Version 6.2   */
#include "DAT.H"          /* data defs */
#define INSIZE 18000
#define CHSIZE 8
#define FLSIZE 2
#define GRPSIZE 6000

char l_index[TAGSIZE] = {"@IAIndex      "};
char keep[MAXGRPNUM]; /* init to FALSE */

static char *data_name[MAXGRPNUM];
static char grp_size[MAXGRPNUM];
static char data_type[MAXGRPNUM];
static char log_flag[MAXGRPNUM]; /* init to FALSE */

static int recdnt;
int intstor[INSIZE];
static int *intpnt;
float floatstor[FLSIZE];
static float *floatpnt;
char charstor[CHSIZE];
static char *charpnt;
char grpstor[GRPSIZE];
static char *grpnt;
/*=====*/
set_data()          /* set data logging */
{
    int sel;

    while (TRUE)
    {
        sel = data_menu();
        switch (sel)
        {
            case 1:
                start_log();
                break;

            case 2:
                save_data();
                break;

            case 3:
                select_keep();
                break;

            case 4:
                dump_log();
                while (!scr_csts()); /* wait for key */
                break;

            case 5:
                debug_log();
                while (!scr_csts()); /* wait for key */
                break;

            case 0:
                return(TRUE);
                break;

            default:
                co(7);          /* beep */
                break;
        }
    }
}
/*=====*/
select_keep()
{
    int sel;

    while (TRUE)
    {
        sel = keep_menu();
        if ((sel < 0) || (sel > MAXGRPNUM))
        {
            co(7);
            continue;
        }

        if (sel == 0)
            return(TRUE);
        keep[sel] = !keep[sel];
    }
}
/*=====*/
data_menu()

```

```

{
int query;

scr_clr();
scr_rowcol(2,0);
printf(" M.I.T. SSL\n");
printf(" Beam Assembly Teleoperator\n");
printf(" Data Recording Function\n");
printf(" 1. INITIALIZE DATA STORAGE\n");
printf(" 2. SAVE DATA TO FILE\n");
printf(" 3. SELECT VARIABLES TO RECORD\n");
printf(" 4. DISPLAY DATA LOG ON SCREEN\n");
printf(" 5. debug data log\n");
printf(" 0. EXIT\n");
printf(" Enter Selection: ");
scanf("%d",&query);
return(query);
}

/*=====*/
keep_menu()
{
int query;
char kp[MAXGRPNUM];

for (query = 0; query < MAXGRPNUM; query++)
if (keep[query])
kp[query] = 'Y';
else
kp[query] = '-';

scr_clr();
scr_rowcol(2,0);
printf(" MIT SSL BAT\n");
printf(" Data Recording Function\n");
printf(" Variable Selection\n");
printf("[%c] 1 LOOP COUNTER [%c]11 JC MEAS POS \n",kp[1],kp[11]);
printf("[%c] 2 JOYSTICKS [%c]12 JC MEAS RATE \n",kp[2],kp[12]);
printf("[%c] 3 THRUST COM [%c]13 JC COM POS \n",kp[3],kp[13]);
printf("[%c] 4 MASTER ARM [%c]14 \n",kp[4],kp[14]);
printf("[%c] 5 JOINT COM [%c]15 \n",kp[5],kp[15]);
printf("[%c] 6 JOINT POS [%c]16 \n",kp[6],kp[16]);
printf("[%c] 7 Z80 LOOP [%c]17 \n",kp[7],kp[17]);
printf("[%c] 8 ARM DUTY CYC [%c]18 \n",kp[8],kp[18]);
printf("[%c] 9 [%c]19 \n",kp[9],kp[19]);
printf("[%c]10 \n",kp[10]);
printf(" 0 - EXIT\n");
printf(" Enter Selection: ");
scanf("%d",&query);
return(query);
}

/*-----*/
save_data()
{
char fname[20];
int i, j, siz, fn;
int grplogcnt;
char ans, grp, ffmt[20];
int *indmpt = instor;
float *fldmpt = floatstor;
char *chdmpt = charstor;

printf(" Write filename [.DAT]: ");
scanf("%s",fname);
strcpy(fname, ".DAT");
if ((fn = creat(fname)) == -1)
{
scr_rowcol(0,0);
bcep(25);
printf("Unable to Write File");
pause(250);
}
else
{
fprintf(fn, "%s\n", fname);
for (grp = 0; grp < MAXGRPNUM; grp++)
if (log_flg[grp])
{
fprintf(fn, "\n%d: %s ", grp, data_name[grp]);
fprintf(fn, " %c %d\n", data_type[grp], grp_size[grp]);
}

fprintf(fn, "\n=====");
grplogcnt = grpnt - grpstor; /* # entries in group store */
for (i = 0; i < grplogcnt; i++)
{
grp = grpstor[i];
siz = grp_size[grp];
fprintf(fn, "\n%d:", grp);
switch (data_type[grp])
{
case FLDAT: /* float store */
for (j = 0; j < siz; j++)
fprintf(fn, " %c", *fldmpt++);
break;

```

```

        case INDAT: /* int store */
            for (j = 0; j < siz; j++)
                fprintf(fn, " %d", *indmpt++);
            break;
        case CHDAT: /* char store */
            for (j = 0; j < siz; j++)
                fprintf(fn, " %d", *chdmpt++);
            break;
        default:
            printf("data_type[%d] = %d ERROR", grp, data_type[grp]);
    }

    close(fn);
}

/*=====*/
start_log()
{
    int i;

    intpnt = intstor;
    floatpnt = floatstor;
    charpnt = charstor;
    grppnt = grpstor;
    for (i = 0; i < MAXGRPNUM; i++)
        log_flag[i] = FALSE;

    recdcnt = 0;
}

/*=====*/
index_log()
{
    logint(l_index, ++recdcnt);
}

/*=====*/
logint(tag, var)
int var;
char tag[];
{
    char grp;
    extern char data_flag;

    grp = tag[0] - '@';
    if (!log_flag[grp])
    {
        log_flag[grp] = TRUE;
        data_type[grp] = tag[1];
        grp_size[grp] = tag[2] - '@';
        data_name[grp] = &tag[3];
    }

    if (tag[1] != INDAT)
        printf("\n*** Wrong type to logint ***\n");
    if ((GRPSIZE - 1) >= (grppnt - grpstor))
        if ((INSIZE - 1) >= (intpnt - intstor))
        {
            *grppnt++ = grp;
            *intpnt++ = var;
        }
        else
        {
            scr_rowcol(24, 0);
            printf("**** INT STORAGE OVERFLOW ****");
            data_flag = FALSE;
            screen2();
        }
    else
    {
        scr_rowcol(24, 0);
        printf("**** GRP STORAGE OVERFLOW ****");
        data_flag = FALSE;
        screen2();
    }
}

/*=====*/
logint_a(tag, var)
int var[];
char tag[];
{
    int i, dim;
    char grp;
    extern char data_flag;

    grp = tag[0] - '@';
    if (!log_flag[grp])
    {
        log_flag[grp] = TRUE;
        data_type[grp] = tag[1];
        grp_size[grp] = tag[2] - '@';
        data_name[grp] = &tag[3];
    }
}

```

```

    }
    if (tag[1] != INDAT)
        printf("\n*** Wrong type to logint_a ***\n");
    dim = tag[2] - '@';
    if ((GRPSIZE - 1) >= (grpnt - grpstor))
        if ((INSIZE - dim) >= (intput - intstor))
        {
            *grpnt++ = grp;
            for (i = 0; i < dim; i++)
                *intput++ = var[i];
        }
        else
        {
            scr_rowcol(24,0); scr_clr();
            printf("**** INT STORAGE OVERFLOW ****");
            data_flag = FALSE;
            screen2();
        }
    else
    {
        scr_rowcol(24,0); scr_clr();
        printf("**** GRP STORAGE OVERFLOW ****");
        data_flag = FALSE;
        screen2();
    }
}

/*=====*/
logchar(tag,var)
char var;
char tag[];
{
    char grp;
    extern char data_flag;

    grp = tag[0] - '@';
    if (!log_flag[grp])
    {
        log_flag[grp] = TRUE;
        data_type[grp] = tag[1];
        grp_size[grp] = tag[2] - '@';
        data_name[grp] = &tag[3];
    }

    if (tag[1] != CHDAT)
        printf("\n*** Wrong type to logchar ***\n");
    if ((GRPSIZE - 1) >= (grpnt - grpstor))
        if ((CHSIZE - 1) >= (charpnt - charstor))
        {
            *grpnt++ = grp;
            *charpnt++ = var;
        }
        else
        {
            scr_rowcol(24,0); scr_clr();
            printf("**** CHAR STORAGE OVERFLOW ****");
            data_flag = FALSE;
            screen2();
        }
    else
    {
        scr_rowcol(24,0); scr_clr();
        printf("**** GRP STORAGE OVERFLOW ****");
        data_flag = FALSE;
        screen2();
    }
}

/*=====*/
logchar_a(tag,var)
char var[];
char tag[];
{
    int i, dim;
    char grp;
    extern char data_flag;

    grp = tag[0] - '@';
    if (!log_flag[grp])
    {
        log_flag[grp] = TRUE;
        data_type[grp] = tag[1];
        grp_size[grp] = tag[2] - '@';
        data_name[grp] = &tag[3];
    }

    if (tag[1] != CHDAT)
        printf("\n*** Wrong type to logchar ***\n");
    dim = tag[2] - '@';
    if ((GRPSIZE - 1) >= (grpnt - grpstor))
        if ((CHSIZE - dim) >= (charpnt - charstor))
        {
            *grpnt++ = grp;
            for (i = 0; i < dim; i++)

```

```

        *charpnt++ = var[i];
    }
    else
    {
        scr_rowcol(24,0);
        scr_clr0();
        printf("**** CHAR STORAGE OVERFLOW ****");
        data_flag = FALSE;
        screen2();
    }
}

else
{
    scr_rowcol(24,0);
    scr_clr0();
    printf("**** GRP STORAGE OVERFLOW ****");
    data_flag = FALSE;
    screen2();
}
}

/*=====*/
logfloat(tag,var)
float var;
char tag[];
{
    char grp;
    extern char data_flag;

    grp = tag[0] - '@';
    if (!log_flag[grp])
    {
        log_flag[grp] = TRUE;
        data_type[grp] = tag[1];
        grp_size[grp] = tag[2] - '@';
        data_name[grp] = &tag[3];
    }

    if (tag[1] != FLDAT)
        printf("\n*** Wrong type to logfloat ***\n");
    if ((GRPSIZE - 1) >= (grpnt - grpstor))
        if ((FLSIZE - 1) >= (floatpnt - floatstor))
        {
            *grpnt++ = grp;
            *floatpnt++ = var;
        }
        else
        {
            scr_rowcol(24,0);
            scr_clr0();
            printf("**** FLOAT STORAGE OVERFLOW ****");
            data_flag = FALSE;
            screen2();
        }
    else
    {
        scr_rowcol(24,0);
        scr_clr0();
        printf("**** GRP STORAGE OVERFLOW ****");
        data_flag = FALSE;
        screen2();
    }
}

/*=====*/
logfloat_a(tag,var)
float var[];
char tag[];
{
    int i, dim;
    char grp;
    extern char data_flag;

    grp = tag[0] - '@';
    if (!log_flag[grp])
    {
        log_flag[grp] = TRUE;
        data_type[grp] = tag[1];
        grp_size[grp] = tag[2] - '@';
        data_name[grp] = &tag[3];
    }

    if (tag[1] != FLDAT)
        printf("\n*** Wrong type to logfloat ***\n");
    dim = tag[2] - '@';
    if ((GRPSIZE - 1) >= (grpnt - grpstor))
        if ((FLSIZE - dim) >= (floatpnt - floatstor))
        {
            *grpnt++ = grp;
            for (i = 0; i < dim; i++)
                *floatpnt++ = var[i];
        }
        else
        {
            scr_rowcol(24,0);
            scr_clr0();
            printf("**** FLOAT STORAGE OVERFLOW ****");
            data_flag = FALSE;
            screen2();
        }
}

```

```

else
{
    scr_rowcol(24,0);
    printf("**** GRP STORAGE OVERFLOW ****");
    data_flag = FALSE;
    screen2();
}

/*=====*/
bailout(errstr);
char *errstr;
{
    printf(errstr);
    exit(2);
}

/*=====*/
dump_log()
{
    int i, j, siz;
    int grplogcnt;
    char ans, grp, frmt[20];
    int *indmpt = intstor;
    float *fldmpt = floatstor;
    char *chdmpt = charstor;

    printf("\nDisplay data log verbose? (Y/N): ");
    ans = getchar();
    if (ans == 'y')
        ans = 'Y';
    printf("\nHit any key to stop\n");
    grplogcnt = grppnt - grpstor;          /* # entries in group store */
    for (i = 0; i < grplogcnt; i++)
    {
        grp = grpstor[i];
        siz = grp_size[grp];
        j = 0;
        /* copy format string */
        if (ans == 'Y')
            printf("\n%s:", data_name[grp]);
        else
            printf("\n%d:", grp);
        switch (data_type[grp])
        {
            case FLDAT: /* float store */
                for (j = 0; j < siz; j++)
                    printf(" %e", *fldmpt++);
                break;
            case INDAT: /* int store */
                for (j = 0; j < siz; j++)
                    printf(" %d", *indmpt++);
                break;
            case CHDAT: /* char store */
                for (j = 0; j < siz; j++)
                    printf(" %d", *chdmpt++);
                break;
            default:
                printf("data_type[%d] = %d ERROR", grp, data_type[grp]);
        }
        if (scr_csts())
            break;
    }
    printf("Dump complete\n");
}

/*=====*/
debug_log()
{
    int indpth, fldpth, chdpth, grpdpth, i;

    printf("\nDEBUG: recdcnt = %d", recdcnt);
    indpth = input - intstor;
    fldpth = floatpnt - floatstor;
    chdpth = charpnt - charstor;
    grpdpth = grppnt - grpstor;

    printf("\nGRP: depth = %d\n", grpdpth);
    printf("\nCHR: depth = %d\n", chdpth);
    printf("\nINT: depth = %d\n", indpth);
    printf("\nFLT: depth = %d\n", fldpth);
    printf("\n--- GROUP ----\n");
    for (i = 0; i < grpdpth; i++)
        printf("%d ", grpstor[i]);
    puts("\n--- CHAR ----\n");
    for (i = 0; i < chdpth; i++)
        printf("%d", charstor[i]);
    puts("\n--- INT ----\n");
    for (i = 0; i < indpth; i++)
        printf("%d ", intstor[i]);
    puts("\n--- FLOAT ----\n");
    for (i = 0; i < fldpth; i++)
        printf("%e", floatstor[i]);
}

```

```
}  
/*=====*/
```



```

        case '0': /* exit */
            not_done = FALSE;
            break;
        default: /* unrecognized command */
            scr_rowcol(15,5);
            printf("MUDPIE command, try again");
            pause(250);
            scr_rowcol(15,5);
            printf(" ");
            break;
    }
}

/*-----
** Learn_set_up() saves a filename to save a learned trajectory to.
**-----*/
char fnamer[15];
learn_set_up()
{
    extern int set_triad;
    extern char learning;
    extern char fnamer[15];
    set_triad = 2; /* activate Triad ready to learn mode */
    scr_rowcol(15,5);
    printf("Set storage file [.TRD] -> ");
    gets(fnamer);
    strcat(fnamer, ".TRD");
}

/*-----
** Learn() is called by a button push. Learn starts and stops the process of saving actual arm positions.
**-----*/
int fntt;
learn()
{
    extern int fntt;
    extern int set_triad;
    extern char fnamer[15];
    extern joint_pos[8];
    if(set_triad==2) /* if Triad in ready to learn mode active */
    {
        if((fntt = creat(fnamer)) == -1) /* make a new trajectory file */
        {
            scr_rowcol(19,24);
            printf("Cannot learn");
            beep(25);
            pause(150);
            scr_rowcol(19,24);
            printf(" ");
        }
        else
        {
            set_triad = 3; /* activate Triad learn mode so save trajectory */
            scr_rowcol(19,24);
            printf("Learning");
        }
    }
    else /* if Triad leaving learn mode */
    {
        /* save another command with EOF code */
        fprintf(fntt, "%4x %4x %4x %4x H\n", joint_pos[0], joint_pos[1], joint_pos[2], joint_pos[3], joint_pos[4]);
        close(fntt); /* close trajectory file */
        set_triad = 2; /* set to Triad ready to learn mode */
        scr_rowcol(19,24);
        printf(" "); /* turn off learning sign */
    }
}

/*-----
** Learn_record() is called once per Batsoft main loop. It saves the actual arm joint ** positions.
**-----*/
learn_record()
{
    extern int fntt;
    extern int joint_pos[8];
    fprintf(fntt, "%4x %4x %4x %4x A\n", joint_pos[0], joint_pos[1], joint_pos[2], joint_pos[3], joint_pos[4]);
}

/*-----*/

```

APPENDIX III. JCS code

```
; Z80 code for the JCS joint cards. Each Z80 controls two joints
page 60,132
title BAT JOINT CONTROL CARD (w/DPR) J1u.asm 8/4/87
page 60,132
;*****
;   Downloaded code for JC cards
;   uses dual-port RAM (DPR) for commands & data
;*****
;   by John Spofford, Tom Waschura, & David E. Anderson
;   MIT Space Systems Laboratory
;
;   include z80.mac
;   include z80_math.mac
;   include joint.mac
;
;-----MACROS START
;
JPBIT      macro    b, reg, label
            bit      b, reg
            jpc      nz, label
            endm
;-----MACROS END
;
;---Set up all I/O Port Definitions
;
pio2a_data      equ    4
pio2a_cntl      equ    5
pio2b_data      equ    6
pio2b_cntl      equ    7
ctc0            equ    8
ctc1            equ    9
ctc2            equ    10
ctc3            equ    11
pio3a_data      equ    12
pio3a_cntl      equ    13
pio3b_data      equ    14
pio3b_cntl      equ    15
;
;---Set up I/O related constants
;
bitmode         equ    0CFH
bi_dirac_mode   equ    08FH
input_mode      equ    04FH
output_mode     equ    00FH
all_input_mask  equ    0FFH
all_output_mask equ    000H
interrupt_ctl    equ    087H
no_interrupt_ctl equ    007H
ctc_vector      equ    010H ;%(LOW ctc_vector_addr)
ctc0_ctl        equ    0D7H
ctc1and2_ctl    equ    057H
tc_follows      equ    057H
min_tc_size     equ    8
wait_count      equ    00AH ;for 10 ms loop should be 10
;
;---Set up Initial Value Constants for RAM Variables
;
dataram_lo      equ    0800h
dataram_hi      equ    0BFFh
stack_top       equ    dataram_hi + 0
var_page_size   equ    0100h
```

```

coderam_lo      equ      1000h
coderam_hi      equ      17FDh
dualram_lo      equ      1800h
dualram_hi      equ      1FFFh

to8088          equ      1FFEh  ;see JRS labbook
from8088         equ      1FFFh
wc              equ      1813h
jc_status       equ      1814h  ;to PC
comm_flag       equ      189Ah  ;from PC

;
;---Define RAM Variables (location, size and usage)
;
;      alpha_n      control algorithm coefficient
;      beta_n       control algorithm coefficient
;      gain_n        control algorithm coefficient
;      mult1_n       control algorithm coefficient
;      mult2_n       integrator on/off and integrator saturation
;      mult3_n       integrator multiplier
;      sigma_n       integration sum
;      sign_n        present sign of error, used to reset sigma_n
;      reference_n   reference position indicated by IBM PC
;      theta_n       indicated position read from decoders
;      control_n     control signal outputted from control
;                   algorithm. (Note: not to be confused
;                   with TC_n)
;      TC_n          CTC time constant
;      delta_n       error difference (reference - theta)
;      old_delta_n   OLD error difference
;      oold_delta_n  Old OLD error difference
;      temp_n        8 bit temporary RAM variable
;      temp16_n      16 bit temporary RAM variable
;      start_signal  start up FLAG
;      algorithm_n   control algorithm selection
;      old_theta_n   OLD indicated position
;      old_control_n OLD control value
;      oold_control_n Old OLD control value
;      direction_n   motor direction indicator
;      bits_2a       multi-use port output byte
;      n_abort       abort flags
;      interrupt_cnt count of CTC interrupt services
;
;      PID CONTROLLER:
;      The PID compensator
;       $(control)/(error) = k\{1 + a/(z-1) + b(z-1)/(z+c)\} + \sigma$ 
;      where k is gain, a is (fake) integrator multiplier, b is differential mult.
;      In batsoft, these are combined shifted to form a 0.16 fixed point binary
;      and sent to the dpr in the form of :
;
;      gain_n        k(b+1) multiplier of error
;      alpha_n       -k(a-1-c-2b) neg of mult of old error
;      beta_n        k(c-ac+b) mult of oold error
;      mult1_n       c mult of old control
;      mult2_n       upper byte: 1=integrator on, 0=integrator off
;                   lower byte: integrator saturation
;      mult3_n       fraction multiple of each loop delta to add to sigma
;      sigma_n       integrated sum. Upper byte added to control
;      sign_n        keeps sign of delta. resets sigma when delta changes sign
;
;===== START OF RAM DEFINITION
from_pc_vars     org      dataram_lo
                  equ      $          ;order of variables is important!

```

```

defun reference_x, 2 ;80
defun reference_y, 2 ;82
defun gain_x, 2 ;84
defun gain_y, 2 ;86
defun alpha_x, 2 ;88
defun alpha_y, 2 ;8A
defun beta_x, 2 ;8C
defun beta_y, 2 ;8E
defun algorithm_x, 1 ;90
defun algorithm_y, 1 ;91
defun pre_dither_x, 2 ;92
defun pre_dither_y, 2 ;94
defun post_dither_x, 2 ;96
defun post_dither_y, 2 ;98
defun nothing, 2 ;9A
defun mult1_x, 2 ;9C
defun mult1_y, 2 ;9E
defun mult2_x, 2 ;A0
defun mult2_y, 2 ;A2
defun mult3_x, 2 ;A4
defun mult3_y, 2 ;A6
;
to_pc_vars equ $ ;order of variables is important!
defun theta_x, 2 ;00
defun theta_y, 2 ;02
defun rate_x, 2 ;04
defun rate_y, 2 ;06
defun calc_ctrl_x, 2 ;08
defun calc_ctrl_y, 2 ;0A
defun TC_x, 1 ;0C
defun TC_y, 1 ;0D
defun direction_x, 1 ;0E
defun direction_y, 1 ;0F
defun loop_counter, 3 ;10
;
defun control_x, 2
defun control_y, 2
defun delta_x, 2
defun delta_y, 2
defun last_data_x, 2
defun last_data_y, 2
defun old_delta_x, 2
defun old_delta_y, 2
defun oold_delta_x, 2
defun oold_delta_y, 2
defun oold_control_x, 2
defun oold_control_y, 2
defun old_last_data_x, 2
defun old_last_data_y, 2
defun vel_gain_x, 2
defun vel_gain_y, 2
defun pos_gain_x, 2
defun pos_gain_y, 2
defun old_theta_x, 2
defun old_theta_y, 2
defun old_control_x, 2
defun old_control_y, 2
defun bits_2a, 1
defun y_abort, 1
defun x_abort, 1
defun interrupt_cnt, 1
defun temp_0, 1
defun temp_1, 1
defun temp16_0, 2

```

```

        defun    temp16_1, 2
        defun    status_2b, 1
        defun    sigma_x, 2
        defun    sigma_y, 2
        defun    sign_x, 1
        defun    sign_y, 1
        IF      $ GT dataram_hi
        %out    DATA RAM OVER-RUN!
        ENDIF

;===== END OF RAM DEFINITION
;
;===== START OF Z80 CODE
        org      coderam_lo
        jp      begin_routine
        $org     %(coderam_lo + 010H)
;
;
;---Place the Interrupt VECTOR TABLE here, then jump to NMI service
;
ctc_vector_addr    equ    $
                  dw      ctc_service
                  dw      00
                  dw      00
                  dw      00
;
                  $org    %(coderam_lo + 30h)
;
;-----<BEGIN MAIN CODE>----->
;
;
;---Initialize all PIO's and the CTC
;
begin_routine      equ    $
                  lri     a, 'f' ;1/2/86 jrs
                  sta     (to8088) ;free the PC

init_pio           %pio2a_cntl, %bitmode, %all_output_mask, %no_interrupt_ctl
init_pio           %pio2b_cntl, %output_mode, 0, %no_interrupt_ctl
init_pio           %pio3a_cntl, %bitmode, %all_input_mask, %no_interrupt_ctl
init_pio           %pio3b_cntl, %bitmode, %all_input_mask, %no_interrupt_ctl
init_ctc           %ctc0, %ctc0_ctl, ctc_vector, 0
init_ctc           %ctc1, %ctc1and2_ctl, 0, 25
init_ctc           %ctc2, %ctc1and2_ctl, 0, 25
;
;
;---Initialize CPU parameters
;
                  lrpi     hl, stack_top
                  lsphl
                  lri     a, %(HIGH ctc_vector_addr)
                  ldia
                  im2
                  ei
;
;---Initialize both motor flip-flops OFF
;
                  lri     a, 00000110B ;first set F-Flops
                  out     pio2a_data
                  lri     a, 11100000B
                  out     pio2a_data
                  sta     (bits_2a)
                  lri     a, tc_follows ;then reset them via
                  out     ctc1 ;the timers timing out
                  out     ctc2

```

```

        lri        a, 1
        out        ctc1
        out        ctc2
;
;---Assuming the PC will not write to Joints until after initializing
;---Initialize RAM Variables with their initial values
;
        lrpi       hl, 0
        sthl       (alpha_x)
        sthl       (alpha_y)
        sthl       (beta_x)
        sthl       (beta_y)
        sthl       (gain_x)
        sthl       (gain_y)
        sthl       (reference_x)
        sthl       (reference_y)
        sthl       (theta_x)
        sthl       (theta_y)
        sthl       (rate_x)
        sthl       (rate_y)
        sthl       (control_x)
        sthl       (control_y)
        sthl       (delta_x)
        sthl       (delta_y)
        sthl       (last_data_x)
        sthl       (last_data_y)
        sthl       (old_delta_x)
        sthl       (old_delta_y)
        sthl       (old_last_data_x)
        sthl       (old_last_data_y)
        sthl       (old_theta_x)
        sthl       (old_theta_y)
        sthl       (old_control_x)
        sthl       (old_control_y)
        sthl       (post_dither_x)
        sthl       (post_dither_y)
        sthl       (pre_dither_x)
        sthl       (pre_dither_y)
        sthl       (mult1_x)
        sthl       (mult1_y)
        sthl       (mult2_x)
        sthl       (mult2_y)
        sthl       (mult3_x)
        sthl       (mult3_y)
        sthl       (sigma_x)
        sthl       (sigma_y)
        sthl       (loop_counter)
        sthl       (oold_control_x)
        sthl       (oold_control_y)
        sthl       (oold_delta_x)
        sthl       (oold_delta_y)
;
        lri        a, wait_count
        sta        (wc)
        lri        a, 0
        sta        (TC_x)
        sta        (TC_y)
        sta        (direction_x)
        sta        (direction_y)
        sta        (x_abort)
        sta        (y_abort)
        sta        (interrupt_cnt)
        sta        (algorithm_x)
        sta        (algorithm_y)

```

```

        sta      (sign_x)
        sta      (sign_y)
        sta      %(loop_counter+2)
;
;
;---Now that all is initialized & the motors are OFF, sit and wait
;---for the start up code from the IBM PC.
;
        lri      a, 'W'
        sta      (jc_status)      ;ready to go flag
        lda      (bits_2a)
        andi     10111111B      ;b6=0 --> not started
        out      pio2a_data
sit_and_wait equ $
        lda      (comm_flag)
        cpi      'G'      ;look for start flag
        jpc      nz, sit_and_wait
;
;---And we're off. Now we'll work on motor x first. Read the x
;---position, and get a control_x based on the selected algorithm
;
main_loop equ $
        lda      (comm_flag)
        cpi      'S'      ;look for shutoff flag
        jpc      z, begin_routine ;reset
        lri      a, 'C'      ;'controlling' flag
        sta      (jc_status)
;
        READ_THETA 0      ;0 selects channel x
;
        VMOV theta_x, temp16_0
        VNEG temp16_0
        VADD temp16_0, reference_x, delta_x
;
;---Now check the algorithm type and dispatch accordingly
;
        lda      algorithm_x
        cpi      0
        jpc      nz, pd_algorithm_x
;
;---PROPORTIONAL CONTROL ALGORITHM
;
;
        MULTIPLY gain_x, delta_x, control_x
        jp      drain_x
;
;---PID CONTROL ALGORITHM
;
pd_algorithm_x equ $
        cpi      1
        jpc      nz, SVC_algorithm_x
        MULTIPLY gain_x, delta_x, temp16_0
        VADD rate_x, temp16_0, control_x
;
        jp      drain_x
;
;---SVC CONTROL ALGORITHM
;
SVC_algorithm_x equ $
        cpi      2
        jpc      nz, OL_algorithm_x
;
        MULTIPLY pos_gain_x, delta_x, control_x
        VMOV theta_x, temp16_0

```

```

        VADD    temp16_0, old_theta_x, temp16_0
        VADD    control_x, temp16_0, control_x
        jp      drain_x
;
;---OPEN LOOP CONTROLLER
;
OL_algorithm_x equ    $
        MULTIPLY gain_x, reference_x, control_x
;
;---Now filter the control value for saturation and direction
;
drain_x      equ    $
        VMOV     control_x, calc_ctrl_x
        filter x
;
;-----
; JOINT Y
;-----
;
;
        READ_THETA 1
;
        VMOV     theta_y, temp16_0
        VNEG     temp16_0
        VADD     temp16_0, reference_y, delta_y
;
;---Now check the algorithm type and dispatch accordingly
;
        lda      algorithm_y
        cpi      0
        jpc      nz, pd_algorithm_y
;
;---PROPORTIONAL CONTROL ALGORITHM
;
;
        MULTIPLY gain_y, delta_y, control_y
        jp      drain_y
;
;---PID CONTROL ALGORITHM
;
pd_algorithm_y equ    $
        cpi      1
        jpc      nz, SVC_algorithm_y
        MULTIPLY gain_y, delta_y, temp16_0
        VADD     rate_y, temp16_0, control_y
;
        jp      drain_y
;
;---SVC CONTROL ALGORITHM
;
SVC_algorithm_y equ    $
        cpi      2
        jpc      nz, OL_algorithm_y
;
        MULTIPLY pos_gain_y, delta_y, control_y
        VMOV     theta_y, temp16_0
        VADD     temp16_0, old_theta_y, temp16_0
        VADD     control_y, temp16_0, control_y
        jp      drain_y
;
;---OPEN LOOP CONTROLLER
;
OL_algorithm_y equ    $
        MULTIPLY gain_y, reference_y, control_y

```

```

;
;---Now filter the control value for saturation and direction
;
drain_y      equ      $
              VMOV     control_y, calc_ctrl_y
              filter y
;
;-----
;PRECALCULATE VALUES FOR NEXT LOOP if PID Controller used.
;
;      this updates the state variables for the algorithms that
;      need them.
;
;              lda      algorithm_x
;              cpi      1
;              jpc      nz, try_y_1
; update x variables
;              VMOV     old_delta_x, oold_delta_x
;              VMOV     delta_x, old_delta_x
;              VMOV     old_control_x, oold_control_x
;              VMOV     theta_x, old_theta_x
;              VMOV     control_x, old_control_x
;
;
; precalculate for PID controller for joint x
;
;              MULTIPLY mult1_x, old_control_x, temp16_0
;              MULTIPLY alpha_x, old_delta_x, temp16_1
;              VNEG     temp16_1
;              VADD     temp16_0, temp16_1, temp16_1
;              MULTIPLY beta_x, oold_delta_x, temp16_0
;              VADD     temp16_0, temp16_1, rate_x
;              MULTIPLY mult3_x, old_delta_x, temp16_0
;              VADD     temp16_0, sigma_x, sigma_x
;
; reset for the PID: mult2 is an on off switch
;
;              ldhl     mult2_x
;              bit      0,h      ;if bit is set (z not set) then go
;              jpc      nz,xxx
;              lri      h,0
;              lri      l,0
;              sthl     (sigma_x)
xxx          equ      $
;
; if the sign delta changes, reset sigma
;
;              ldhl     old_delta_x
;              bit      7, h ;check sign of error
;              jpc      z, psign_x      ;error is negative
;              lda      sign_x
;              bit      0, a      ;if bit 0 not set, reset sigma
;              jpc      nz, norst_x
;              lri      h, 0
;              lri      l, 0
;              sthl     %(sigma_x)
;              lri      a, 0      ;delta now positive, change sign
;              sta      (sign_x)
;              jp       norst_x
psign_x      equ      $
;              lda      sign_x      ;if bit 0 of a is set, reset sigma
;              bit      0, a
;              jpc      z, norst_x
;              lri      h, 0

```

```

        lri    1, 0
        sthl   %(sigma_x)
        lri    a, 1
        sta    (sign_x)
norst_x  equ    $
;
;clip sigma if necessary
        ldrp   de, sigma_x
        ldhl   mult2_x
        bit    7, d
        jpc    z, pos_x
        ld     a, 1
        neg
        cp     d        ;if a>d, bad, carry not set
        jpc    nc, nbad_x
        ld     e, d
        lri    d, 0FFh
        jp     good_x
nbad_x   equ    $
        ld     d, a
        lri    e, 0
        strp   de, %(sigma_x)
        ld     e, d
        lri    d, 0FFh
        jp     good_x
pos_x    equ    $
        ld     a, 1
        cp     d        ;if d> a, bad        carry set
        jpc    car, pbad_x
        ld     e, d
        lri    d, 0
        jp     good_x
pbad_x   equ    $
        ld     d, a
        lri    e, 0
        strp   de, %(sigma_x)
        ld     e, d
        lri    d, 0
        jp     good_x
good_x   equ    $
        strp   de, %(temp16_0)
        VADD   temp16_0, rate_x, rate_x
;
;
try_y_1  equ    $
        lda    algorithm_y
        cpi    1
        jpc    nz, try_x_SVC
; update y variables
        VMOV   old_delta_y, oold_delta_y
        VMOV   delta_y, old_delta_y
        VMOV   old_control_y, oold_control_y
        VMOV   theta_y, old_theta_y
        VMOV   control_y, old_control_y
;
; precalculate for PID controller for joint y
;
        MULTIPLY mult1_y, old_control_y, temp16_0
        MULTIPLY alpha_y, old_delta_y, temp16_1
        VNEG   temp16_1
        VADD   temp16_0, temp16_1, temp16_1
        MULTIPLY beta_y, oold_delta_y, temp16_0
        VADD   temp16_0, temp16_1, rate_y
        MULTIPLY mult3_y, old_delta_y, temp16_0
        VADD   temp16_0, sigma_y, sigma_y

```

```

;
; reset for the PID: mult2 is an on off switch
;
                ldhl    mult2_y
                bit     0,h ;if bit is set (z not set) then go
                jpc     nz,yyy
                lri     h,0
                lri     l,0
                sthl    (sigma_y)
yyy             equ     $
;
; if the sign delta changes, reset sigma
;
                ldhl    old_delta_y
                bit     7, h ;check sign of error
                jpc     z, psign_y ;error is negative
                lda     sign_y
                bit     0, a ;if bit 0 not set, reset sigma
                jpc     nz, norst_y
                lri     h, 0
                lri     l, 0
                sthl    %(sigma_y)
                lri     a, 0 ;delta now positive, change sign
                sta     (sign_y)
                jp      norst_y
psign_y         equ     $
                lda     sign_y ;if bit 0 of a is set, reset sigma
                bit     0, a
                jpc     z, norst_y
                lri     h, 0
                lri     l, 0
                sthl    %(sigma_y)
                lri     a, 1
                sta     (sign_y)
norst_y         equ     $
; clip sigma if necessary
                ldrp    de, sigma_y
                ldhl    mult2_y
                bit     7, d
                jpc     z, pos_y
                ld      a, 1
                neg
                cp      d ;if a>d, bad, carry not set
                jpc     nc, nbad_y
                ld      e, d
                lri     d, 0FFh
                jp      good_y
nbad_y          equ     $
                ld      d, a
                lri     e, 0
                strp    de, %(sigma_y)
                ld      e, d
                lri     d, 0FFh
                jp      good_y
pos_y           equ     $
                ld      a, 1
                cp      d ;if d> a, bad carry set
                jpc     car, pbad_y
                ld      e, d
                lri     d, 0
                jp      good_y
pbad_y          equ     $
                ld      d, a
                lri     e, 0

```

```

                strp    de, %(sigma_y)
                ld      e, d
                lri     d, 0
good_y          equ     $
                strp    de, %(temp16_0)
                VADD    temp16_0, rate_y, rate_y
;
;
;
;
; precalculate for SFC controller
;
try_x_SVC       equ     $
;               lda     algorithm_x
;               cpi     2
;               jpc     nz, try_y_SVC
;
try_y_SVC       equ     $
;               lda     algorithm_y
;               cpi     2
;               jpc     nz, end_precalcs
;               nop
;
;
end_precalcs    equ     $
;-----
; check for new info from PC
check_new       equ     $
                lda     (from8088)
                cpi     'N'      ;new info flag
                jpc     nz, dont_read
                lri     a, 'R'   ;reading flag
                sta     (from8088)
                lrpri    bc, 28h  ;setup for ldir
                lrpri    hl, %(dualram_lo + 80h) ;from pc area
                lrpri    de, from_pc_vars
                ldir
                lri     a, 'O'   ;info has been read flag
                sta     (from8088)
                jp      transfer
dont_read       equ     $
                cpi     'U'      ;info being updated flag
                ;do nothing, get it next time

; send data to PC
transfer        equ     $
                lda     (to8088)
                cpi     'R'      ;currently being read flag
                jpc     z, done_talking ;next time..
                lri     a, 'U'   ;updating flag
                sta     (to8088)
                lrpri    bc, 13h  ;setup for ldir
                lrpri    hl, to_pc_vars
                lrpri    de, %(dualram_lo + 0) ;to pc area
                ldir
                lri     a, 'N'   ;new data available flag
                sta     (to8088)
done_talking    equ     $
;-----
;now wait for interrupt_count to get to wait_count. This will permit
;a 10 ms calculation loop.
;
;--- set pio2-a7 zero while waiting
                lda     (bits_2a)
                res     7, a      ;b7=0 --> waiting

```

```

        sta      (bits_2a)
        out      pio2a_data
;---here in wait loop, increase loop count:
        lrpw     hl,(loop_counter)
        lri      a, 1          ;increment value = 1
        addm     ;first byte
        strm     a
        lri      a, 0          ;zero for carry
        incrp    hl
        adcm     ;second..
        strm     a
        lri      a, 0          ;zero for carry
        incrp    hl
        adcm     ;last byte
        strm     a

;---the loop
wait_here    lri      b, wait_count
            equ      $
            lda      (interrupt_cnt)
            cp       b
            jpc     nz, wait_here
;---leaving loop, reset count and status bit:
            lri      a, 0
            sta      (interrupt_cnt)
            lda      (bits_2a)
            set     7, a          ;b7=1 --> done waiting
            sta      (bits_2a)
            jp      main_loop
;
;
;*****
;*****
;
;       This is the CTC interrupt service routine:
;
;       This routine is designed to have fast response between the
;       pulse start signal and the timer start signals. The fastest
;       this can be done is in 34 cycles (@ 2 MHz = .017 ms). The
;       CTC counts every .004 ms, so the smallest count I can accurately
;       represent is 5 (i.e. 4.25), hence if the tc_n is less than
;       5, it is ignored, & then it is decreased by 4.
;       and if you understand that, congratulations!
;
ctc_service    equ     $
               save_state
;
               lda     (interrupt_cnt)
               inc     a
               sta     (interrupt_cnt)
;
               lda     (tc_x)          ;check for small
               ld      b, a
               lri      a, 0
               cp       b
               jpc     z, no_pulse_tc0
               lda     (x_abort)
               bit      0, a
               jpc     nz, no_pulse_tc0
               lri      a, ctc1and2_ctl ;shut it down
               out     ctc1
               lda     (bits_2a)
               set     1, a          ;start pulse
               out     pio2a_data
               res      1, a

```

```

        out    pio2a_data
        ld     a, b
        out    ctc1
;
no_pulse_tc0    equ    $
                lda    (tc_y)
                ld     b, a
                lri    a, 0
                cp     b
                jpc    z, no_pulse_tc1
                lda    (y_abort)
                bit    0, a
                jpc    nz, no_pulse_tc1
                lri    a, ctc1and2_ctl    ;shut 'em down
                out    ctc2
                lda    (bits_2a)
                set    2, a                ;start pulse
                out    pio2a_data
                res    2, a
                out    pio2a_data
                ld     a, b
                out    ctc2
;
no_pulse_tc1    equ    $
;
                restore_state
                ei
                reti
;
;
;
;*****
;
;  Multiply Subroutine (called by MULTIPLY macro)
;
mult_sub        equ    $
                lrpi    hl, 0
                irpc    x, <BC>
                irp     numb, <7,6,5,4,3,2,1,0>
                    mugn x, numb
                endm
                endm
                ret
;*****
;
                IF $ GT coderam_hi
                %OUT CODE EXCEEDS RAM LIMIT!
                ENDIF
;
end

```

APPENDIX IV. Triad trajectories

Grabbeam.trd: 3/17/88
 Start at stow, extend beam carrier, grab the beam
 with the right claw, retract beam carrier

```

28F4 5084 4F0 4BC0 DE80 D -- Open right claw
28F4 5084 432 4C76 DE32 M -- Extend beam carrier
28F4 5084 374 4D2C DDE4 A
28F4 5084 2B6 4DE2 DD96 A
28F4 5084 1F8 4E98 DD48 A
28F4 5084 13A 4F4E DCFA A
28F4 5084 07C 5004 DCAC A
28F4 5084 FFBE 50BA DC5E A
28F4 5084 FF00 5170 DC10 A
28F4 5084 FE42 5226 DBC2 A
28F4 5084 FD80 52E0 DB74 A -- Right arm clear of stow handle
28F4 5084 FC73 52E0 DB26 A
28F4 5084 FB66 52E0 DAD8 A
28F4 5084 FA59 52E0 DA8A A
28F4 5084 F94C 52E0 DA3C A
28F4 5084 F83F 52E0 D9EE A
28F4 5084 F732 52E0 D9A0 A
28F4 5084 F625 52E0 D952 A
28F4 5084 F518 52E0 D904 A
28F4 5084 F40B 52E0 D8B6 A
28F4 5084 F2FE 52E0 D868 A -- Right arm clear of left arm
28F4 5084 F1F0 52E0 D810 P -- Load Triad2 gains
2A84 4E44 F1F0 50CD D810 A
2C84 4C44 F1F0 4EBA D810 A
2E84 4A44 F1F0 4CA7 D810 A
3084 4844 F1F0 4A94 D810 A
3284 4644 F1F0 4881 D810 A
3484 4444 F1F0 466E D810 A
3684 4244 F1F0 445B D810 A
3884 4144 F1F0 4248 D810 A
3A84 4144 F1F0 4035 D810 A
3C84 4144 F1F0 3E22 D810 A
3E84 4144 F1F0 3C0F D810 A
4084 4144 F1F0 39FC D810 A
4284 4144 F1F0 37E9 D810 A
4484 4144 F1F0 35D6 D810 A
4684 4144 F1F0 33C3 D810 A
4884 4144 F1F0 31B0 D810 A
4A84 4144 F1F0 2F9D D810 A
4C84 4144 F1F0 2D8A D810 A
4E84 4144 F1F0 2B77 D810 A
5084 4144 F1F0 2964 D810 A
5284 4144 F1F0 2751 D810 A
5484 4144 F1F0 253E D810 A
5684 4144 F1F0 232B D810 A
5884 4144 F1F0 2118 D810 A
5884 4144 F1F0 1F05 D810 A
5884 4144 F1F0 1CF2 D810 A
5884 4144 F1F0 1ADF D810 A
5884 4144 F1F0 18CC D810 A

```

```

5884 4144 F1F0 16B9 D810 A -- Shy and wrist in position for grab
5884 4144 F2F0 15C0 D030 T -- Wait for beam carrier to settle
5884 4144 F3B0 15C0 D030 T
5884 4144 F3B0 15C0 D030 T
5884 4144 F3B0 15C0 D030 N -- Load Triad1 gains
5884 4144 F46B 15C0 D0BC T
5884 4144 F526 15C0 D148 A
5884 4144 F5E1 15C0 D1D4 A
5884 4144 F69C 15C0 D260 A
5884 4144 F757 15C0 D2EC A
5884 4144 F812 15C0 D378 A
5884 4144 F8CD 15C0 D404 A
5884 4144 F988 15C0 D490 A
5884 4144 FA43 15C0 D51C A
5884 4144 FAFE 15C0 D5A8 A
5884 4144 FBB9 15C0 D634 A
5884 4144 FC74 15C0 D6C0 A
5884 4144 FD2F 15C0 D74C A
5884 4144 FDEA 15C0 D7D8 A
5884 4144 FEA5 15C0 D864 A
5884 4144 FF60 15C0 D8F0 A
5884 4144 1B 15C0 D97C A
5884 4144 D6 15C0 DA08 A
5884 4144 191 15C0 DA94 A
5884 4144 24C 15C0 DB20 A
5884 4144 307 15C0 DBAC A
5884 4144 310 15C0 DBC0 E -- Close right claw on beam
5884 4144 310 15C0 DD40 F -- Open beam carrier claw
5884 4144 310 15C0 DD40 U -- Short pause while claw opens
5884 4144 310 15C0 DD40 L -- Retract beam carrier
5884 4144 310 15C0 DD40 H -- End of File

```

Joint.trd: 3/17/88

Start at grabbeam with beam.
position beam for easy manual
joint assembly.

5584	4574	3B0	15C0	DD40	T	-- Pause for beam to settle
5584	4574	3D0	15C0	DD90	Q	-- Load in heavy work gains
5584	4574	3F0	15C0	DDE0	A	
5584	4574	410	15C0	DE30	A	
5584	4574	430	15C0	DE80	A	
5584	4574	450	15C0	DED0	A	
5584	4574	470	15C0	DF20	A	
5584	4574	490	15C0	DF70	A	
5584	4574	4B0	15C0	DFC0	A	
5584	4574	4D0	15C0	E010	A	
5584	4574	4F0	15C0	E060	A	
5584	4574	510	15C0	E100	A	
5584	4574	530	1620	E150	A	
5584	4574	550	16A0	E1A0	A	
5584	4574	570	1720	E1F0	A	
5584	4574	590	17A0	E240	A	
5584	4574	5B0	1820	E290	A	
5584	4574	5D0	18A0	E2E0	A	
5584	4574	5F0	1920	E330	A	
5584	4574	610	19A0	E380	A	
5584	4574	638	1A20	E3D0	A	
5584	4574	660	1AA0	E3E0	T	-- Pause for beam to settle
5584	4574	660	1AA0	E3E0	H	-- End of trajectory

Grabslv.trd: 3/17/88
 Start at just-made-joint, release beam,
 grab sleeve.

5534	4454	AC0	1AE0	E430	D	--Open right claw
5534	4454	A16	1AE0	E3A6	A	
5534	4454	96C	1AE0	E31C	A	
5534	4454	8C2	1AE0	E292	A	
5534	4454	818	1AE0	E208	A	
5534	4454	76E	1AE0	E17E	A	
5534	4454	6C4	1AE0	E0F4	A	
5534	4454	61A	1AE0	E06A	A	
5534	4454	570	1AE0	DFE0	A	
5534	4454	4C6	1AE0	DF56	A	
5534	4454	41C	1AE0	DECC	A	
5534	4454	372	1AE0	DE42	A	
5534	4454	2C8	1AE0	DDB8	A	
5534	4454	21E	1AE0	DD2E	A	
5534	4454	174	1AE0	DCA4	A	
5534	4454	CA	1AE0	DC1A	A	
5534	4454	20	1AE0	DB90	A	
5534	4454	FF76	1AE0	DB06	A	
5534	4454	FECC	1AE0	DA7C	A	
5534	4454	FE22	1AE0	D9F2	A	
5534	4454	FD78	1AE0	D968	A	
5534	4454	FCCE	1AE0	D8DE	A	
5534	4454	FC24	1AE0	D854	A	
5534	4454	FB7A	1AE0	D7CA	A	
5534	4454	FAC0	1AE0	D730	P	--Claw clear of beam, load higher gains
5484	45B4	FAC0	1BE0	D730	A	
5284	45B4	FAC0	1C60	D730	A	
5084	45B4	FAC0	1CE0	D730	A	
4E84	45B4	FAC0	1D60	D730	A	
4D84	45B4	FAC0	1DE0	D730	A	
4D84	45B4	FAC0	1E60	D730	A	
4D84	45B4	FAC0	1EE0	D730	A	
4D84	45B4	FAC0	1F60	D730	A	
4D84	45B4	FAC0	1FE0	D730	A	
4D84	45B4	FAC0	2060	D730	A	
4D84	45B4	FAC0	20E0	D730	A	
4D84	45B4	FAC0	2160	D730	A	
4D84	45B4	FAC0	21E0	D730	A	
4D84	45B4	FAC0	2260	D730	A	
4D84	45B4	FAC0	22E0	D730	A	
4D84	45B4	FAC0	2360	D730	A	
4D84	45B4	FAC0	23E0	D730	A	
4D84	45B4	FAC0	2460	D730	A	
4D84	45B4	FAC0	24E0	D730	A	
4D84	45B4	FAC0	2500	D730	N	--Claw on approach to sleeve, load low gains
4D84	45B4	FB5D	2500	D7CE	A	
4D84	45B4	FBFA	2500	D86C	A	
4D84	45B4	FC97	2500	D90A	A	
4D84	45B4	FD34	2500	D9A8	A	
4D84	45B4	FDD1	2500	DA46	A	
4D84	45B4	FE6E	2500	DAE4	A	
4D84	45B4	FF0B	2500	DB82	A	
4D84	45B4	FFA8	2500	DC20	A	
4D84	45B4	45	2500	DCBE	A	

```

4D84 45B4    E2 2500 DD5C A
4D84 45B4   17F 2500 DDFA A
4D84 45B4   21C 2500 DE98 A
4D84 45B4   2B9 2500 DF36 A
4D84 45B4   356 2500 DFD4 A
4D84 45B4   3F3 2500 E072 A
4D84 45B4   490 2500 E110 A
4D84 45B4   52D 2500 E1AE A
4D84 45B4   5CA 2500 E24C A
4D84 45B4   667 2500 E2EA A
4D84 45B4   704 2500 E388 A
4D84 45B4   7A1 2500 E426 A
4D84 45B4   83E 2500 E4C4 A
4D84 45B4   8DB 2500 E562 A
4D84 45B4   978 2500 E600 A
4D84 45B4   A15 2500 E69E A
4D84 45B4   AB2 2500 E73C A
4D84 45B4   B4F 2500 E7DA A
4D84 45B4   BEC 2500 E878 A
4D84 45B4   C89 2500 E916 A
4D84 45B4   D26 2500 E9B4 A
4D84 45B4   DC3 2500 EA52 A
4D84 45B4   E60 2500 EAF0 A
4D84 45B4   EFD 2500 EB8E A
4D84 45B4   F9A 2500 EC2C A
4D84 45B4 1050 2500 ECE0 E --Close right claw
4D84 45B4 1050 2500 ECE0 H --End of file

```

Slideslv.trd: 3/17/88
Start grabbing unslid sleeve. Slide it.

4D84	4814	1050	2500	ECE0	E	--Make sure claw is closed
4D84	4814	1050	2500	ECE0	7	--Increase allowable position error
4D84	4814	1050	2500	ECE0	Q	--Load in heavy work gains
4D04	47D8	10A7	2567	ED6C	A	
4C84	479C	10FE	25CE	EDF8	A	
4C04	4760	1155	2635	EE84	A	
4B84	4724	11AC	269C	EF10	A	
4B04	46E8	1203	2703	EF9C	A	
4A84	46AC	125A	276A	F028	A	
4A04	4670	12B1	27D1	F0B4	A	
4984	4634	1308	2838	F140	A	
4904	45F4	1360	28A0	F1D0	A	
4814	4674	13F0	2910	F290	A	
4714	4774	1480	2980	F350	A	
4614	4874	1510	29F0	F410	A	
4514	4974	15A0	2A60	F4D0	A	
4414	4974	1630	2AD0	F590	A	
4414	4974	16C0	2B40	F650	A	
4414	4974	1750	2BB0	F710	A	
4414	4974	17E0	2C20	F7D0	A	
4414	4974	1850	2CC0	F8F0	A	
4414	4974	1850	32C0	F8F0	T	--Give Shy a kick and allow to settle
4414	4974	1850	2CC0	F8F0	N	--Reload normal gains
4414	4974	1850	2CC0	F8F0	H	--End of trajectory

Stow.trd: 3/17/88
Start at slid sleeve and go to stow

4414	4974	1850	2CC0	F8F0	D	--Open right claw
4414	4974	16EA	2CC0	F7AF	A	
4414	4974	1584	2CC0	F66E	A	
4414	4974	141E	2CC0	F52D	A	
4414	4974	12B8	2CC0	F3EC	A	
4414	4974	1152	2CC0	F2AB	A	
4414	4974	FEC	2CC0	F16A	A	
4414	4974	E86	2CC0	F029	A	
4414	4974	D20	2CC0	EEE8	A	
4414	4974	BBA	2CC0	EDA7	A	
4414	4974	A54	2CC0	EC66	A	
4414	4974	8EE	2CC0	EB25	A	
4414	4974	788	2CC0	E9E4	A	
4414	4974	622	2CC0	E8A3	A	
4414	4974	4BC	2CC0	E762	A	
4414	4974	356	2CC0	E621	A	
4414	4974	1F0	2CC0	E4E0	A	
4414	4974	8A	2CC0	E39F	A	
4414	4974	FF24	2CC0	E25E	A	
4414	4974	FDBE	2CC0	E11D	A	
4414	4974	FC58	2CC0	DFDC	A	
4414	4974	FAF2	2CC0	DE9B	A	
4414	4974	F98C	2CC0	DD5A	A	
4414	4974	F826	2CC0	DC19	A	
4414	4974	F6C0	2CC0	DAD8	A	
4414	4974	F55A	2CC0	D997	A	
4414	4974	F3F0	2CC0	D856	A	
4414	4974	F3F0	2CC0	D810	P	--Claw is clear of beam, load higher gains
443C	4A84	F3F0	2F00	D810	A	
42A4	4B84	F3F0	3100	D810	A	
410C	4C84	F3F0	3300	D810	A	
3F74	4D84	F3F0	3500	D810	A	
3DDC	4E8C	F3F0	3700	D810	A	
3C44	4F84	F3F0	3900	D810	A	
3AAC	508C	F3F0	3B00	D810	A	
3914	5084	F3F0	3D00	D810	A	
377C	5084	F3F0	3F00	D810	A	
35E4	5084	F3F0	4100	D810	A	
344C	5084	F3F0	4300	D810	A	
32B4	5084	F3F0	44C4	D810	A	
311C	5084	F3F0	4687	D810	A	
2F84	5084	F3F0	484A	D810	A	
2DEC	5084	F3F0	4A0D	D810	A	
2C54	5084	F3F0	4BD0	D810	A	
2ABC	5084	F3F0	4D93	D810	A	
2924	5084	F3F0	4F56	D810	A	
28FC	5084	F3F0	5119	D810	A	
27F4	5084	F3F0	52E0	D810	N	--On approach to stow handle, use low gains
26F4	5084	F4A7	52E0	D810	A	
25F4	5084	F55E	52E0	D810	A	
25F4	5084	F615	52E0	D810	A	
25F4	5084	F6CC	52E0	D810	A	
25F4	5084	F783	52E0	D810	A	
25F4	5084	F83A	52E0	D810	A	
25F4	5084	F8F1	52E0	D810	A	

```

25F4 5084 F9A8 52E0 D810 A
25F4 5084 FA5F 52E0 D810 A
25F4 5084 FB16 52E0 D810 A
25F4 5084 FBCD 52E0 D810 A
25F4 5084 FC84 52E0 D810 A
25F4 5084 FD3B 52E0 D810 A
25F4 5084 FD80 52E0 D810 U --Pause briefly on final approach to stow
25F4 5084 FDDA 5280 D862 A
25F4 5084 FE34 5220 D8B4 A
25F4 5084 FE8E 51C0 D906 A
25F4 5084 FEE8 5160 D958 A
25F4 5084 FF42 5100 D9AA A
25F4 5084 FF9C 50A0 D9FC A
25F4 5084 FFF6 5040 DA4E A
25F4 5084    50 4FE0 DA80 A
25F4 5084    AA 4F80 DAF2 A
25F4 5084   104 4F20 DB44 A
25F4 5084   15E 4EC0 DB96 A
25F4 5084   1B8 4E60 DBE8 A
25F4 5084   212 4E00 DC3A A
25F4 5084   26C 4DA0 DC8C A
25F4 5084   2C6 4D40 DCDE A
25F4 5084   320 4CE0 DD30 A
25F4 5084   37A 4C80 DD82 A
25F4 5084   3D4 4C20 DDD4 A
25F4 5084   42E 4BC0 DE26 A
25F4 5084   488 4BC0 DE80 A
25F4 5084   4F0 4BC0 DE80 S --Call for slaverest at end of trajectory
25F4 5084   4F0 4BC0 DE80 E --Close right claw
25F4 5084   4F0 4BC0 DE80 H --End of trajectory

```

Cgrab.trd: 3/17/88
 Start at stow, grab cluster from rack
 and retract to running position, leave live.

28F4	5084	4F0	4BC0	DE80	D	--Open right claw
28F4	5084	432	4C76	DDB4	A	
28F4	5084	374	4D2C	DD4E	A	
28F4	5084	2B6	4DE2	DCE8	A	
28F4	5084	1F8	4E98	DC82	A	
28F4	5084	13A	4F4E	DC1C	A	
28F4	5084	07C	5004	DBB6	A	
28F4	5084	FFBE	50BA	DB50	A	
28F4	5084	FF00	5170	DAEA	A	
28F4	5084	FE42	5226	DA80	A	
28F4	5084	FD80	52E0	DA00	P	--Clear of stow handle, load higher gains
28F4	5084	FC73	52E0	D966	A	
28F4	5084	FB66	52E0	D8CC	A	
28F4	5084	FA59	52E0	D832	A	
28F4	5084	F94C	52E0	D798	A	
28F4	5084	F83F	52E0	D6FE	A	
28F4	5084	F732	52E0	D664	A	
28F4	5084	F625	52E0	D5CA	A	
28F4	5084	F518	52E0	D530	A	
28F4	5084	F40B	52E0	D496	A	
28F4	5084	F2FE	52E0	D3FC	A	
28F4	5084	F1F0	52E0	D360	A	
2AD4	4E44	F1F0	50CD	D360	A	
2CD4	4C44	F1F0	4EBA	D360	A	
2ED4	4A44	F1F0	4CA7	D360	A	
30D4	4844	F1F0	4A94	D360	A	
32D4	4644	F1F0	4881	D360	A	
34D4	4444	F1F0	466E	D360	A	
36D4	4254	F1F0	445B	D360	A	
38D4	4054	F1F0	4248	D360	A	
3AD4	3E54	F1F0	4035	D360	A	
3CD4	3C54	F1F0	3E22	D360	A	
3ED4	3C54	F1F0	3C0F	D360	A	
40D4	3C54	F1F0	39FC	D360	A	
42D4	3C54	F1F0	37E9	D360	A	
44D4	3C54	F1F0	35D6	D360	A	
46D4	3C54	F1F0	33C3	D360	A	
48D4	3C54	F1F0	31B0	D360	A	
4AD4	3C54	F1F0	2F9D	D360	A	
4CD4	3C54	F1F0	2D8A	D360	A	
4ED4	3C54	F1F0	2B77	D360	A	
50D4	3C54	F1F0	2964	D360	A	
52D4	3C54	F1F0	2751	D360	A	
54D4	3C54	F1F0	253E	D360	A	
56D4	3C54	F1F0	232B	D360	A	
58D4	3C54	F1F0	2118	D360	A	
5AD4	3C54	F1F0	1F05	D360	A	
5CD4	3C54	F1F0	1CF2	D360	A	
5ED4	3C54	F1F0	1ADF	D360	A	
60D4	3C54	F1F0	18CC	D360	A	
62D4	3C54	F1F0	16B9	D360	A	
64D4	3C54	F1F0	14A6	D360	A	
66D4	3C54	F1F0	1293	D360	A	
6894	3C54	F1F0	1080	D360	A	

```

6894 3C54 F1F0 E6D D360 A
6894 3C54 F1F0 C5A D360 A
6894 3C54 F1F0 A47 D360 A
6894 3C54 F1F0 834 D360 A
6894 3C54 F2A6 740 D360 A
6894 3C54 F35C 740 D360 A
6894 3C54 F412 740 D360 A
6894 3C54 F4C8 740 D360 A
6894 3C54 F57C 740 D360 A
6894 3C54 F634 740 D360 A
6894 3C54 F6EA 740 D360 A
6894 3C54 F7A0 740 D360 A
6894 3C54 F856 740 D360 A
6894 3C54 F910 740 D360 A
6894 3C54 F910 740 D360 N --On approach to cluster, use lower gains
6894 3C54 F963 740 D3D9 A
6894 3C54 FA56 740 D452 A
6894 3C54 FAF9 740 D4CB A
6894 3C54 FB9C 740 D544 A
6894 3C54 FC3F 740 D5BD A
6894 3C54 FCE2 740 D636 A
68E4 3C54 FD85 740 D6AF A
6894 3C54 FE28 740 D728 A
6894 3C54 FECB 740 D7A1 A
6894 3C54 FF6E 740 D81A A
6894 3C54 11 740 D893 A
6894 3C54 B4 740 D90C A
6894 3C54 157 740 D985 A
6894 3C54 1FA 740 D9FE A
6894 3C54 29D 740 DA77 A
6894 3C54 340 740 DAF0 A
6894 3C54 3E3 740 DB69 A
6894 3C54 486 740 DBE2 A
6894 3C54 530 740 DC60 E --Close right claw
6894 3C54 530 740 DC60 Q --Increase gains to heavy
6894 3C54 530 740 DC60 7 --Increase allowable error
6000 3800 10 6A0 D170 A --Jerk the cluster off the rack
5F3E 38A2 10 6A0 D146 A
5E7C 3944 10 6A0 CFB0 A
5DBA 39E6 10 6A0 CE1A A
5CF8 3A88 10 6A0 CC84 A
5C36 3B2A FFF4 717 CAEE A
5B74 3BCC FF6E 78E C958 A
5AB2 3C6E FEE8 805 C7C2 A
59F0 3D10 FE62 87C C62C A
592E 3DB2 FDDC 8F3 C496 A
586C 3E54 FD56 96A C300 A
57AA 3EF6 FCD0 9E1 C16A A
56E8 3F98 FC4A A58 BFD4 A
5626 403A FBC4 ACF BE3E A
5564 40DC FB3E B46 BCA0 A
5494 4184 FAB0 0BC0 BCA0 A
5494 4184 FAB0 0BC0 BCA0 H --End of file

```

Ctri.trd: 3/17/88
 Start at stow and move arm into the workspace

28F4	5084	4F0	4BC0	DE80	D	--Open right claw
28F4	5084	432	4C76	DE32	A	
28F4	5084	374	4D2C	DDE4	A	
28F4	5084	2B6	4DE2	DD96	A	
28F4	5084	1F8	4E98	DD48	A	
28F4	5084	13A	4F4E	DCFA	A	
28F4	5084	07C	5004	DCAC	A	
28F4	5084	FFBE	50BA	DC5E	A	
28F4	5084	FF00	5170	DC10	A	
28F4	5084	FE42	5226	DBC2	A	
28F4	5084	FD80	52E0	DB74	A	
28F4	5084	FC73	52E0	DB26	A	
28F4	5084	FB66	52E0	DAD8	A	
28F4	5084	FA59	52E0	DA8A	A	
28F4	5084	F94C	52E0	DA32	A	
28F4	5084	F83F	52E0	D9EE	A	
28F4	5084	F732	52E0	D9A0	A	
28F4	5084	F625	52E0	D952	A	
28F4	5084	F518	52E0	D904	A	
28F4	5084	F40B	52E0	D8B6	A	
28F4	5084	F2FE	52E0	D868	A	
28F4	5084	F1F0	52E0	D810	P	--Claw clear of stow, load higher gains
2AD4	4E44	F1F0	50CD	D810	A	
2CD4	4C44	F1F0	4EBA	D810	A	
2ED4	4A44	F1F0	4CA7	D810	A	
30D4	4844	F1F0	4A94	D810	A	
32D4	4644	F1F0	4881	D810	A	
34D4	4444	F1F0	466E	D810	A	
36D4	4254	F1F0	445B	D810	A	
38D4	4184	F1F0	4248	D810	A	
3AD4	4184	F1F0	4035	D810	A	
3CD4	4184	F1F0	3E22	D810	A	
3ED4	4184	F1F0	3C0F	D810	A	
40D4	4184	F1F0	39FC	D810	A	
42D4	4184	F1F0	37E9	D810	A	
44D4	4184	F1F0	35D6	D810	A	
46D4	4184	F1F0	33C3	D810	A	
48D4	4184	F1F0	31B0	D810	A	
4AD4	4184	F1F0	2F9D	D810	A	
4CD4	4184	F1F0	2D8A	D810	A	
4ED4	4184	F1F0	2B77	D810	A	
50D4	4184	F1F0	2964	D810	A	
52D4	4184	F1F0	2751	D810	A	
5494	4184	F1F0	253E	D810	A	
5494	4184	F1F0	232B	D810	A	
5494	4184	F1F0	2118	D810	A	
5494	4184	F1F0	1F05	D810	A	
5494	4184	F1F0	1CF2	D810	A	
5494	4184	F1F0	1ADF	D810	A	
5494	4184	F1F0	18CC	D810	A	
5494	4184	F1F0	16B9	D810	A	
5494	4184	F1F0	14A6	D810	A	
5494	4184	F1F0	1293	D810	A	
5494	4184	F1F0	1080	D810	A	
5494	4184	F1F0	E6D	D810	A	

5494 4184 F1F0 C5A D810 A
5494 4184 F1F0 BC0 D810 H --End of trajectory

Cjoint.trd: 3/17/88
 Start at running position for carrying
 a cluster, and move cluster for easy
 manual joint assembly.

5494	4184	FAB0	BC0	BCA0	T	--Pause for cluster to settle
5494	4184	FAB0	BC0	BCA0	T	
5494	4184	FB64	C67	BE4D	P	--Load higher gains
5494	4184	FC18	D0E	BFFA	A	
5494	4184	FCCC	DB5	C1A7	A	
5494	4184	FD80	E5C	C354	A	
5494	4184	FE34	F03	C501	A	
5494	4184	FEE8	FAA	C6AE	A	
5494	4184	FF9C	1051	C85B	A	
5494	4184	50	10F8	CA08	A	
5494	4184	104	119F	CBB5	A	
5494	4184	1B8	1246	CD62	A	
5494	4184	26C	12ED	CF0F	A	
5494	4184	320	1394	D0BC	A	
5494	4184	3D4	143B	D269	A	
5494	4184	488	14E2	D416	A	
5494	4184	53C	1589	D5C3	A	
5494	4184	5F0	1630	D770	A	
5494	4184	6A4	16D7	D91D	A	
5494	4184	758	177E	DACA	A	
5494	4184	80C	1825	DC77	A	
5494	4184	8C0	18CC	DE24	A	
5494	4184	974	1973	DFD1	A	
5494	4184	A28	1A1A	E17E	A	
5494	4184	AF0	1AD0	E340	T	--Pause to allow cluster to settle
5494	4184	AF0	1AD0	E340	T	
5494	4184	AF0	1AD0	E340	H	--End of trajectory

Rgrabslv.trd: 3/17/88
 With claw on cluster just attached to
 existing beam, grab sleeve on beam end.

5574	44A4	840	1B20	E2C0	D	--Open right claw
5574	44A4	793	1B20	E232	A	
5574	44A4	6E6	1B20	E1A4	A	
5574	44A4	639	1B20	E116	A	
5574	44A4	58C	1B20	E088	A	
5574	44A4	4DF	1B20	DFFA	A	
5574	44A4	432	1B20	DF6C	A	
5574	44A4	385	1B20	DEDE	A	
5574	44A4	2D8	1B20	DE50	A	
5574	44A4	22B	1B20	DDC2	A	
5574	44A4	17E	1B20	DD34	A	
5574	44A4	D1	1B20	DCA6	A	
5574	44A4	24	1B20	DC18	A	
5574	44A4	FF77	1B20	DB8A	A	
5574	44A4	FECA	1B20	DAFC	A	
5574	44A4	FE1D	1B20	DA6E	A	
5574	44A4	FD70	1B20	D9E0	A	
5574	44A4	FCC3	1B20	D952	A	
5574	44A4	FC16	1B20	D8C4	A	
5574	44A4	FB69	1B20	D836	A	
5574	44A4	FB30	1B20	D7A0	P	--Claw clear of cluster, load higher gains
5484	4524	FB30	1B80	D7A0	A	
5384	4524	FB30	1C00	D7A0	A	
5284	4524	FB30	1C80	D7A0	A	
5184	4524	FB30	1D00	D7A0	A	
5084	4524	FB30	1D80	D7A0	A	
4F84	4524	FB30	1E00	D7A0	A	
4E84	4524	FB30	1E80	D7A0	A	
4D84	4524	FB30	1F00	D7A0	A	
4C84	4524	FB30	1F80	D7A0	A	
4B84	4524	FB30	2000	D7A0	A	
4A84	4524	FB30	2080	D7A0	A	
4984	4524	FB30	2100	D7A0	A	
4984	4524	FB30	2180	D7A0	A	
4984	4524	FB30	2200	D7A0	A	
4984	4524	FB30	2280	D7A0	A	
4984	4524	FB30	2300	D7A0	A	
4984	4524	FB30	2380	D7A0	A	
4984	4524	FB30	2400	D7A0	A	
4984	4524	FB30	2480	D7A0	A	
4984	4524	FB30	2500	D7A0	A	
4984	4524	FB30	2580	D7A0	A	
4984	4524	FB30	2600	D7A0	A	
4984	4524	FB30	2680	D7A0	A	
4984	4524	FB30	2700	D7A0	A	
4984	4524	FB30	2780	D7A0	A	
4984	4524	FB30	2800	D7A0	A	
4984	4524	FB30	2880	D7A0	A	
4984	4524	FB30	2900	D7A0	A	
4984	4524	FB30	2980	D7A0	N	--On approach to sleeve, load lower gains
4984	4524	FBE8	2980	D872	A	
4984	4524	FCA0	2980	D944	A	
4984	4524	FD58	2980	DA16	A	
4984	4524	FE10	2980	DAE8	A	

4984	4524	FEC8	2980	DBBA	A	
4984	4524	FF80	2980	DC8C	A	
4984	4524	38	2980	DD5E	A	
4984	4524	F0	2980	DE30	A	
4984	4524	1A8	2980	DF02	A	
4984	4524	260	2980	DFD4	A	
4984	4524	318	2980	E0A6	A	
4984	4524	3D0	2980	E178	A	
4984	4524	488	2980	E24A	A	
4984	4524	540	2980	E31C	A	
4984	4524	5F8	2980	E3EE	A	
4984	4524	6B0	2980	E4C0	A	
4984	4524	768	2980	E592	A	
4984	4524	820	2980	E664	A	
4984	4524	8D8	2980	E736	A	
4984	4524	990	2980	E808	A	
4984	4524	A48	2980	E8DA	A	
4984	4524	B00	2980	E9AC	A	
4984	4524	BB8	2980	EA7E	A	
4984	4524	C70	2980	EB50	A	
4984	4524	D28	2980	EC22	A	
4984	4524	DE0	2980	ECF4	A	
4984	4524	E98	2980	EDC6	A	
4984	4524	F50	2980	EE98	A	
4984	4524	1008	2980	EF6A	A	
4984	4524	10C0	2980	F03C	A	
4984	4524	1178	2980	F10E	A	
4984	4524	1240	2980	F1E0	E	--Close right claw
4984	4524	1240	2980	F1E0	H	--End of trajectory

Rsldslv.trd: 3/17/88
Slide the sleeve from existing beam
onto a cluster being attached.

4984	4524	1240	2980	F1E0	E	--Close right claw
4966	451A	11EE	2920	F144	Q	--Load heavy gains
4948	4510	119C	28C0	F0A8	7	--Increase allowable error size
492A	4506	114A	2860	F00C	A	
490C	44FC	10F8	2800	EF70	A	
48EE	44F2	10A6	27A0	EED4	A	
48D0	44E8	1054	2740	EE38	A	
48B2	44DE	1002	26E0	ED9C	A	
4894	44D4	FB0	2680	ED00	A	
48F8	44C4	F75	2634	ECA0	A	
495C	44B4	F3A	25E8	EC40	A	
49C0	44A4	EFF	259C	EBE0	A	
4A24	4494	EC4	2550	EB80	A	
4A88	4484	E89	2504	EB20	A	
4AEC	4474	E4E	24B8	EAC0	A	
4B50	4464	E13	246C	EA60	A	
4BB4	4454	DD8	2420	EA00	A	
4C18	4444	D9D	23D4	E9A0	A	
4C84	4444	D60	2380	E940	A	
4CC1	4434	D36	233A	E906	A	
4CFE	4424	D0C	22F4	E8CC	A	
4D3B	4414	CE2	22AE	E892	A	
4D78	4404	CB8	2268	E858	A	
4DB5	4404	C8E	2220	E81E	A	
4DF4	4404	C60	1E20	E7E0	T	--Jolt Shy and allow to settle
4DF4	4404	C60	2220	E7E0	H	--End of trajectory

Rstow.trd: 3/17/88
 While attaching a cluster to existing beam,
 leave slide sleeve and go to stow handle.

4DF4	4404	C60	2220	E7E0	D	--Open right claw
4DF4	4404	BEB	2220	E77B	A	
4DF4	4404	B76	2220	E716	A	
4DF4	4404	B01	2220	E6B1	A	
4DF4	4404	A8C	2220	E64C	A	
4DF4	4404	A17	2220	E5E7	A	
4DF4	4404	9A2	2220	E582	A	
4DF4	4404	92D	2220	E51D	A	
4DF4	4404	8B8	2220	E4B8	A	
4DF4	4404	843	2220	E453	A	
4DF4	4404	7CE	2220	E3EE	A	
4DF4	4404	759	2220	E389	A	
4DF4	4404	6E4	2220	E324	A	
4DF4	4404	66F	2220	E2BF	A	
4DF4	4404	5FA	2220	E25A	A	
4DF4	4404	585	2220	E1F5	A	
4DF4	4404	510	2220	E190	A	
4DF4	4404	49B	2220	E12B	A	
4DF4	4404	426	2220	E0C6	A	
4DF4	4404	3B1	2220	E061	A	
4DF4	4404	330	2220	DFF0	A	
4DF4	4404	1AA	2220	DF27	A	
4DF4	4404	24	2220	DE5E	A	
4DF4	4404	FE9E	2220	DD95	A	
4DF4	4404	FD18	2220	DCCC	A	
4DF4	4404	FB92	2220	DC03	A	
4DF4	4404	FA0C	2220	DB3A	A	
4DF4	4404	F886	2220	DA71	A	
4DF4	4404	F700	2220	D9A8	A	
4DF4	4404	F57A	2220	D8DF	A	
4DF4	4404	F3F0	2220	D810	P	--Clear of sleeve, load higher gains
4BF4	44A4	F3F0	23C0	D810	A	
49F4	45AC	F3F0	2560	D810	A	
47F4	46A4	F3F0	2700	D810	A	
45F4	47AC	F3F0	28A0	D810	A	
43F4	48A4	F3F0	2A40	D810	A	
41F4	49AC	F3F0	2BE0	D810	A	
3FF4	4AA4	F3F0	2D80	D810	A	
3DF4	4BAC	F3F0	2F20	D810	A	
3BF4	4CA4	F3F0	30C0	D810	A	
39F4	4DA4	F3F0	3260	D810	A	
37F4	4EA4	F3F0	3400	D810	A	
35F4	4FA4	F3F0	35A0	D810	A	
33F4	50A4	F3F0	3740	D810	A	
31F4	50A4	F3F0	38E0	D810	A	
2FF4	50A4	F3F0	3A80	D810	A	
2DF4	50A4	F3F0	3C20	D810	A	
2BF4	50A4	F3F0	3DC0	D810	A	
29F4	50A4	F3F0	3F60	D810	A	
27F4	50A4	F3F0	4100	D810	A	
25F4	50A4	F3F0	42A0	D810	A	
25F4	50A4	F3F0	4440	D810	A	
25F4	50A4	F3F0	45E0	D810	A	
25F4	50A4	F3F0	4780	D810	A	

```

25F4 50A4 F3F0 4920 D810 A
25F4 50A4 F3F0 4AC0 D810 A
25F4 50A4 F3F0 4C60 D810 A
25F4 50A4 F3F0 4E00 D810 A
25F4 50A4 F3F0 4FA0 D810 A
25F4 50A4 F3F0 5140 D810 A
25F4 50A4 F3F0 52E0 D810 A
25F4 5084 F3F0 52E0 D810 A
25F4 5084 F4A7 52E0 D810 A
25F4 5084 F55E 52E0 D810 A
25F4 5084 F615 52E0 D810 A
25F4 5084 F6CC 52E0 D810 A
25F4 5084 F783 52E0 D810 A
25F4 5084 F83A 52E0 D810 A
25F4 5084 F8F1 52E0 D810 A
25F4 5084 F9A8 52E0 D810 A
25F4 5084 FA5F 52E0 D810 A
25F4 5084 FB16 52E0 D810 A
25F4 5084 FBCD 52E0 D810 A
25F4 5084 FC84 52E0 D810 A
25F4 5084 FD3B 52E0 D810 A
25F4 5084 FD80 52E0 D810 A
25F4 5084 FD80 52E0 D810 A
25F4 5084 FD80 52E0 D810 N --On approach to stow, load lower gains
25F4 5084 FD80 5280 D862 A
25F4 5084 FD80 5220 D8B4 A
25F4 5084 FD80 51C0 D906 A
25F4 5084 FD80 5160 D958 A
25F4 5084 FD80 5100 D9AA A
25F4 5084 FD80 50A0 D9FC A
25F4 5084 FD80 5040 DA4E A
25F4 5084 FD80 4FE0 DA80 A
25F4 5084 FE12 4F80 DAF2 A
25F4 5084 FEA4 4F20 DB44 A
25F4 5084 FF36 4EC0 DB96 A
25F4 5084 FFC8 4E60 DBE8 A
25F4 5084 5A 4E00 DC3A A
25F4 5084 EC 4DA0 DC8C A
25F4 5084 17E 4D40 DCDE A
25F4 5084 210 4CE0 DD30 A
25F4 5084 2A2 4C80 DD82 A
25F4 5084 334 4C20 DDD4 A
25F4 5084 3C6 4BC0 DE26 A
25F4 5084 458 4BC0 DE80 A
25F4 5084 4F0 4BC0 DE80 S --Call for slaverest() at end of trajectory
25F4 5084 4F0 4BC0 DE80 E --Close right claw
25F4 5084 4F0 4BC0 DE80 H --End of trajectory

```

APPENDIX V. EASE structure assembly schedule

BAT EASE assembly progression

0. Startup

A. Set up right arm

- 1). Call for control power on
- 2). Run right arm through it's zeros
- 3). Enable Left Claw.
- 4). Enable Right Claw.
- 5). Enable Beam Carrier.
- 6). Have right arm placed on the stow handle.
- 7). Close right claw
- 8). Slaverest the arm

B. Tilt and Pan setup.

- 1). Run tilt and pan through it's zeros.
- 2). Center head controller
- 3). Disconnect head.

C. Enable Puma

- 1). Call for main power on.
- 2). Enable Puma
- 3). Power relay on.
- 4). Video switch to Belly

1. Attach the First Top Cluster to the first racked upright beam (C1a - B1b).

A. Dock to cluster Rack position 1.

- 1). Maneuver thrusters to place left claw on the first cluster rack dock location.
- 2). Close left claw.

B. Retrieve cluster C1

- 1). Video switch to Tilt and Pan (op.)
- 2). Connect head controller (op.)
- 3). Maneuver head to view stow handle (op.)
- 4). operators arm to master arm.
- 5). Press Button to initiate Triad trajectory "Cgrab" to retrieve cluster
- 6). Verify the cluster is free of the rack (arm will remain live)
- 7). Disconnect master arm.
- 8). video switch to Belly (op.)
- 9). Open left Claw

C. Dock to free end of beam B1b

- 1). Maneuver thrusters to place left claw on beam end B1b to the left side of the disk.
- 2). Close left Claw.

D. Attach cluster to Beam end (C1a - B1b)

- 1). Video switch to Tilt and Pan
- 2). Connect head controller
- 3). Operators arm to master arm
- 4). Verify "Cjoint" showing on next up reverse trajectory
- 5). Push inner finger switch to initiate "cjoint" trajectory.
- 6). Maneuver right arm with master arm to make the joint
- 7). Push inner finger switch to initiate "Rgrabslv" and "Rsldslv" trajectories
- 8). Verify button pop
- 9). Push inner finger switch to initiate "Rstow" trajectory
- 10). Verify Stow.
- 11). disconnect head
- 12). Video switch to Belly
- 13). Open left Claw.

2. Attach the Second Top Cluster to the second racked upright beam (C2a - B2b).

A. Dock to cluster Rack position 2.

- 1). Maneuver thrusters to place left claw on the second cluster rack dock location.
- 2). Close left claw.

B. Retrieve cluster C2

- 1). Video switch to Tilt and Pan (op.)
- 2). Connect head controller (op.)
- 3). Maneuver head to view stow handle (op.)
- 4). operators arm to master arm.
- 5). Press Button to initiate Triad trajectory "Cgrab" to retrieve cluster
- 6). Verify the cluster is free of the rack (arm will remain live)

- 7). Disconnect master arm.
- 8). video switch to Belly (op.)
- 9). Open left Claw
- C. Dock to free end of beam B2b
 - 1). Maneuver thrusters to place left claw on beam end B2b to the left side of the disk.
 - 2). Close left Claw.
- D. Attach cluster to Beam end (C2a - B2b)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Operators arm to master arm
 - 4). Verify "Cjoint" showing on next up reverse trajectory
 - 5). Push inner finger switch to initiate "cjoint" trajectory.
 - 6). Maneuver right arm with master arm to make the joint
 - 7). Push inner finger switch to initiate "Rgrabslv" and "Rsldslv" trajectories
 - 8). Verify button pop
 - 9). Push inner finger switch to initiate "Rstow" trajectory
 - 10). Verify Stow.
 - 11). disconnect head
 - 12). Video switch to Belly
 - 13). Open left Claw.
3. Attach the Third Top Cluster to the fourth rack upright beam (C3a - B4b).
 - A. Dock to cluster Rack position 3.
 - 1). Maneuver thrusters to place left claw on the third cluster rack dock location.
 - 2). Close left claw.
 - B. Retrieve cluster C3
 - 1). Video switch to Tilt and Pan (op.)
 - 2). Connect head controller (op.)
 - 3). Maneuver head to view stow handle (op.)
 - 4). operators arm to master arm.
 - 5). Press Button to initiate Triad trajectory "Cgrab" to retrieve cluster
 - 6). Verify the cluster is free of the rack (arm will remain live)
 - 7). Disconnect master arm.
 - 8). video switch to Belly (op.)
 - C. Dock to free end of beam B4b
 - 1). Maneuver thrusters to place left claw on beam end B4b to the left side of the disk.
 - 2). Close left Claw.
 - D. Attach cluster to Beam end (C3a - B4b)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Operators arm to master arm
 - 4). Verify "Cjoint" showing on next up reverse trajectory
 - 5). Push inner finger switch to initiate "cjoint" trajectory.
 - 6). Maneuver right arm with master arm to make the joint
 - 7). Push inner finger switch to initiate "Rgrabslv" and "Rsldslv" trajectories
 - 8). Verify button pop
 - 9). Push inner finger switch to initiate "Rstow" trajectory
 - 10). Verify Stow.
 - 11). disconnect head
 - 12). Video switch to Belly
 - 13). Open left Claw.
 - 14). Open left Claw.
4. Attach the first upright beam to the base cluster (B1a - CBa).
 - A. Dock to Beam Rack position 1.
 - 1). Maneuver thrusters to place left claw on the first beam rack dock location.
 - 2). Close left claw.
 - B. Retrieve Beam B1
 - 1). Extend Beam carrier
 - 2). Close Beam Carrier claw.
 - 3). Retract Beam Carrier
 - 4). Open left claw.
 - C. Dock to Base Cluster CBa.
 - 1). Maneuver thrusters to place left claw on Base Cluster CBa next to black tape.
 - 2). Close left Claw.
 - D. Attach beam to base cluster (B1a - CBa)

- 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Grabbeam" showing on next up forward trajectory
 - 6). Push outer finger switch to initiate "Grabbeam" and "joint" trajectory.
 - 7). Maneuver right arm with master arm to make the joint
 - 8). Push outer finger switch to initiate "grabslv" and "slidslv" trajectories
 - 9). Verify button pop
 - 10). Push inner finger switch to initiate "stow" trajectory
 - 11). Verify Stow.
 - 12). disconnect head
 - 13). Video switch to Belly
 - 14). Open left Claw.
5. Attach the second upright beam to the base cluster (B2a - CBb).
- A. Dock to Beam Rack position 2.
 - 1). Maneuver thrusters to place left claw on the second beam rack dock location.
 - 2). Close left claw.
 - B. Retrieve Beam B2
 - 1). Extend Beam carrier
 - 2). Close Beam Carrier claw.
 - 3). Retract Beam Carrier
 - 4). Open left claw.
 - C. Dock to Base Cluster CBb.
 - 1). Maneuver thrusters to place left claw on Base Cluster CBb next to black tape.
 - 2). Close left Claw.
 - D. Attach beam to base cluster (B2a - CBb)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Grabbeam" showing on next up forward trajectory
 - 6). Push outer finger switch to initiate "Grabbeam" and "joint" trajectory.
 - 7). Maneuver right arm with master arm to make the joint
 - 8). Push outer finger switch to initiate "grabslv" and "slidslv" trajectories
 - 9). Verify button pop
 - 10). Push inner finger switch to initiate "stow" trajectory
 - 11). Verify Stow.
 - 12). disconnect head
 - 13). Video switch to Belly
 - 14). Open left Claw.
6. Attach the first cross beam to the first top cluster (B3a - C1b).
- A. Dock to Beam Rack position 3.
 - 1). Maneuver thrusters to place left claw on the 3rd beam rack dock location.
 - 2). Close left claw.
 - B. Retrieve Beam B3
 - 1). Extend Beam carrier
 - 2). Close Beam Carrier claw.
 - 3). Retract Beam Carrier
 - 4). Open left claw.
 - C. Dock to top Cluster C1b.
 - 1). Maneuver thrusters to place left claw on top Cluster C1b on the edge of the silver tubing.
 - 2). Close left Claw.
 - D. Attach beam to top cluster (B3a - C1b)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Grabbeam" showing on next up forward trajectory
 - 6). Push outer finger switch to initiate "Grabbeam" and "joint" trajectory.
 - 7). Maneuver right arm with master arm to make the joint
 - 8). Push outer finger switch to initiate "grabslv" and "slidslv" trajectories
 - 9). Verify button pop
 - 10). Push inner finger switch to initiate "stow" trajectory

- 11). Verify Stow.
- 12). disconnect head
- 13). Video switch to Belly
- 14). Open left Claw.
7. Complete the triangle involving the first cross beam and the second top cluster (B3b - C2c).
 - A. Dock to the free end of beam B3b.
 - 1). Maneuver thrusters to place left claw on the beam end B3b next to the disk.
 - 2). Close left Claw.
 - B. Fly the beam end up to cluster C2c
 - C. Attach beam to top cluster (B3b - C2c)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Cjoint" showing on next up reverse trajectory
 - 6). Push inner finger switch to initiate "ctri" trajectory.
 - 7). maneuver right arm with master arm to position right claw over beam end next to the disk
 - 8). Close right claw
 - 9). Maneuver right arm with master arm to make the joint
 - 10). Push inner finger switch to initiate "Rgrabslv" and "Rsldslv" trajectories
 - 11). Verify button pop
 - 12). Push inner finger switch to initiate "Rstow" trajectory
 - 13). Verify Stow.
 - 14). disconnect head
 - 15). Video switch to Belly
 - 16). Open left Claw.
8. Attach the third upright beam to the base cluster (B4a - CBc).
 - A. Dock to Beam Rack position 4.
 - 1). Maneuver thrusters to place left claw on the 4th beam rack dock location.
 - 2). Close left claw.
 - B. Retrieve Beam B4
 - 1). Extend Beam carrier
 - 2). Close Beam Carrier claw.
 - 3). Retract Beam Carrier
 - 4). Open left claw.
 - C. Dock to Base Cluster CBc.
 - 1). Maneuver thrusters to place left claw on Base Cluster CBc next to black tape.
 - 2). Close left Claw.
 - D. Attach beam to base cluster (B4a - CBc)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Grabbeam" showing on next up forward trajectory
 - 6). Push outer finger switch to initiate "Grabbeam" and "joint" trajectory.
 - 7). Maneuver right arm with master arm to make the joint
 - 8). Push outer finger switch to initiate "grabslv" and "slidslv" trajectories
 - 9). Verify button pop
 - 10). Push inner finger switch to initiate "stow" trajectory
 - 11). Verify Stow.
 - 12). disconnect head
 - 13). Video switch to Belly
 - 14). Open left Claw.
9. Attach the second cross beam to the second top cluster (B5a - C2b).
 - A. Dock to Beam Rack position 5.
 - 1). Maneuver thrusters to place left claw on the 5th beam rack dock location.
 - 2). Close left claw.
 - B. Retrieve Beam B5
 - 1). Extend Beam carrier
 - 2). Close Beam Carrier claw.
 - 3). Retract Beam Carrier
 - 4). Open left claw.
 - C. Dock to top Cluster C2b.
 - 1). Maneuver thrusters to place left claw on top Cluster C2b on the edge of the silver tubing.

- 2). Close left Claw.
- D. Attach beam to top cluster (B5a - C2b)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Grabbeam" showing on next up forward trajectory
 - 6). Push outer finger switch to initiate "Grabbeam" and "joint" trajectory.
 - 7). Maneuver right arm with master arm to make the joint
 - 8). Push outer finger switch to initiate "grabslv" and "slidslv" trajectories
 - 9). Verify button pop
 - 10). Push inner finger switch to initiate "stow" trajectory
 - 11). Verify Stow.
 - 12). disconnect head
 - 13). Video switch to Belly
 - 14). Open left Claw.
10. Complete the triangle involving the second cross beam and the third top cluster (B5b - C3c).
 - A. Dock to the free end of beam B5b.
 - 1). Maneuver thrusters to place left claw on the beam end B5b next to the disk.
 - 2). Close left Claw.
 - B. Fly the beam end up to cluster C3c
 - C. Attach beam to top cluster (B5b - C3c)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Cjoint" showing on next up reverse trajectory
 - 6). Push inner finger switch to initiate "ctri" trajectory.
 - 7). maneuver right arm with master arm to position right claw over beam end next to the disk
 - 8). Close right claw
 - 9). Maneuver right arm with master arm to make the joint
 - 10). Push inner finger switch to initiate "Rgrabslv" and "Rsldslv" trajectories
 - 11). Verify button pop
 - 12). Push inner finger switch to initiate "Rstow" trajectory
 - 13). Verify Stow.
 - 14). disconnect head
 - 15). Video switch to Belly
 - 16). Open left Claw.
11. Attach the third cross beam to the third top cluster (B6a - C3b).
 - A. Dock to Beam Rack position 6.
 - 1). Maneuver thrusters to place left claw on the 6th beam rack dock location.
 - 2). Close left claw.
 - B. Retrieve Beam B6
 - 1). Extend Beam carrier
 - 2). Close Beam Carrier claw.
 - 3). Retract Beam Carrier
 - 4). Open left claw.
 - C. Dock to top Cluster C3b.
 - 1). Maneuver thrusters to place left claw on top Cluster C3b on the edge of the silver tubing.
 - 2). Close left Claw.
 - D. Attach beam to top cluster (B6a - C3b)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Grabbeam" showing on next up forward trajectory
 - 6). Push outer finger switch to initiate "Grabbeam" and "joint" trajectory.
 - 7). Maneuver right arm with master arm to make the joint
 - 8). Push outer finger switch to initiate "grabslv" and "slidslv" trajectories
 - 9). Verify button pop
 - 10). Push inner finger switch to initiate "stow" trajectory
 - 11). Verify Stow.
 - 12). disconnect head
 - 13). Video switch to Belly

- 14). Open left Claw.
12. Complete the triangle involving the third cross beam and the second top cluster (B6b - C1c).
 - A. Dock to the free end of beam B6b.
 - 1). Maneuver thrusters to place left claw on the beam end B6b next to the disk.
 - 2). Close left Claw.
 - B. Fly the beam end up to cluster C1c
 - C. Attach beam to top cluster (B6b - C1c)
 - 1). Video switch to Tilt and Pan
 - 2). Connect head controller
 - 3). Maneuver head to view stow handle (op.)
 - 4). Operators arm to master arm
 - 5). Verify "Cjoint" showing on next up reverse trajectory
 - 6). Push inner finger switch to initiate "ctri" trajectory.
 - 7). maneuver right arm with master arm to position right claw over beam end next to the disk
 - 8). Close right claw
 - 9). Maneuver right arm with master arm to make the joint
 - 10). Push inner finger switch to initiate "Rgrabslv" and "Rsldslv" trajectories
 - 11). Verify button pop
 - 12). Push inner finger switch to initiate "Rstow" trajectory
 - 13). Verify Stow.
 - 14). disconnect head
 - 15). Video switch to Belly
 - 16). Open left Claw.
13. Shut Down
 - A. Power relay off
 - B. Call for main power off
 - C. Call for Control power off
 - D. Disable Puma
 - E. Disable beam carrier
 - F. Disable left claw
 - G. Disable right claw