University of California at Davis
California Department of Transportation

THE BIG ARTICULATED STENCILING ROBOT (BASR)*

Volume I

Phillip W. Wong, P.E.
Professor Bahram Ravani
Richard Blank
Jeff Hemenway
Richard McGrew

AHMCT

Advanced Highway Maintenance and Construction Technology

Department of Mechanical, Aeronautical & Materials Engineering
Division of New Technology and Materials Research

University of California at Davis
California Department of Transportation

THE BIG ARTICULATED STENCILING ROBOT (BASR)*

Volume I

Phillip W. Wong, P.E.
Professor Bahram Ravani
Richard Blank
Jeff Hemenway
Richard McGrew
Ulrich Mueller
Dr. Walter Nederbragt
Robert Olshausen
Ken Sprott

AHMCT Research Report
UCD-ARR-98-01-15-01

Final Report of Contract
RTA-65X936

January 15, 1998

*This work was supported by the California Department of Transportation (Caltrans) Advanced Highway and Maintenance and Construction Technology Program at UC-Davis and by the Federal Highway Administration (FHWA).

Technical Documentation Page

1. Report No. FHWA/CA/NT-98/12	2. (PB98-155955	3. Recipient's Catalog No.
4. Title and Subtitle The Big Articulated Stenciling Robot (BASR), 4 volumes		5. Report Date January 15, 1998
		6. Performing Organization Code
7. Author(s) Phillip W. Wong, Bahram Ravani, Richard Blank, Jeff Hemenway, Richard McGrew, Ulrich Mueller, Walter Nederbragt, Robert Olshausen, Kenneth Sprott		8. Performing Organization Report No.
9. Performing Organization Name and Address AHMCT Research Center UCD Dept of Mech & Aero Engineering Davis, California 95616-5294		10. Work Unit No. (TRAIS)
		11. Contract or Grant RTA-65X936
12. Sponsoring Agency Name and Address California Department of Transportation P. O. Box 942873, MS 83 Sacramento, CA 94273-0001		13. Type of Report and Period Covered Final Report 1995-1997
		14. Sponsoring Agency Code

15. Supplementary Notes

This study was conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration, under the research project entitled "Development of a Robotic System for Stenciling of General Roadway Markings"

16. Abstract

This report provides a description of the Big Articulated Stenciling Robot (BASR) that was developed at the University of California at Davis under contract to the California Department of Transportation (Caltrans). This system is designed to paint markings on the roadway pavement with a primary emphasis on increasing highway worker safety by keeping personnel out of unprotected roadway areas. The system is completely integrated, with all normal operations controlled from a single control panel. Overall system descriptions are given in this report. Furthermore, detail design and operational descriptions are given. The report is contained in four (4) volumes, with each volume providing complete details of specific aspects about BASR. Volume I contains introductory material, theory of operation, system schematics, and source code listings. Volume II is a copy of Robert Olshausen's 1996 U.C. Davis Master's Thesis, "Development of an Articulating Robotic Arm for Spray Painting on Roadways". Volume III is a copy of Richard A. McGrew's 1996 U.C. Davis Master's Thesis, "A Robotic End-Effector for Roadway Stenciling". Volume IV is a copy of Richard Blank's 1996 U.C. Davis Master's Thesis, "Algorithms and Robotic Hardware Improvements for Painting of Roadway Markings".

17. Key Words	18. Distribution Statement
Highway, Maintenance, Construction, Pavement, Equipment, Robotics, Highway safety, Painting, New technology, Stenciling, Markings	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

20. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	614	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

ABSTRACT

This report provides a description of the Big Articulated Stenciling Robot (BASR) that was developed at the University of California at Davis under contract to the California Department of Transportation (Caltrans). This system is designed to paint markings on the roadway pavement with a primary emphasis on increasing highway worker safety by keeping personnel out of unprotected roadway areas. The system is completely integrated, with all normal operations controlled from a single control panel. Overall system descriptions are given in this report. Furthermore, detail design and operational descriptions are given. The report is contained in four (4) volumes, with each volume providing complete details of specific aspects about BASR. Volume I contains introductory material, theory of operation, system schematics, and source code listings. Volume II is a copy of Robert H. Olshausen's 1996 U.C. Davis Master's Thesis, " Development of an Articulating Robotic Arm for Spray Painting on Roadways" (report number UCD-ARR-96-09-30-01). Volume III is a copy of Richard A. McGrew's 1996 U.C. Davis Master's Thesis, " A Robotic End-Effector for Roadway Stenciling" (report number UCD-ARR-96-06-30-01). Volume IV is a copy of Richard Blank's 1996 U.C. Davis Master's Thesis, " Algorithms and Robotic Hardware Improvements for Painting of Roadway Markings" (report number UCD-ARR-97-06-15-01).

TABLE OF CONTENTS

Abstract i
Table of Contents ii
List of Figures iii
1. Introduction 1
2. System Description
  2.1 Truck 2
  2.2 Paint Subsystem 3
  2.3 Reflective Bead System 5
  2.4 Main Power Unit 6
  2.5 The Manipulator 7
  2.6 The End-Effector 7
  2.7 Supporting Subsystems
    2.7.1 The Computer Control Subsystem 7
    2.7.2 Hydraulic Supply Subsystem 9
    2.7.3 Electrical Subsystem 9
    2.7.4 Pneumatic Subsystem 10
    2.7.5 Embedded Software 10
3. System Operation
  3.1 System Startup 11
  3.2 Normal Operations 11
  3.3 Clean Up 11
4. Conclusions 11
Appendix A: Stabilizer Feet Data 12
Appendix B: Schematic of ESVCC Board 15
Appendix C: Joint SBC Commands 17
Appendix D: Connector Pinout Table 21
Appendix E: Source Code Listing for Transverse Joint SBC 28
Appendix F: Source Code Listing for Rotation Joint SBC 50
Appendix G: Source Code Listing for Extension Joint SBC 73
Appendix H: Source Code Listing for End Effector SBC 95
Appendix I: End Effector SBC Commands 116
Appendix J: Hydraulic System Schematic 119
Appendix K: Research Paper on BASR Motion Planning Software 123

LIST OF FIGURES

Figure 1: Manipulator Layout 1
Figure 2: Truck Layout.......... 3
Figure 3: Paint System Schematic 4
Figure 4: Paint Heating System 4
Figure 5: Temperature Control Schematic 5
Figure 6: Reflective Bead System Schematic 5
Figure 7: Main Power Unit Layout 6
Figure 8: MPU Control Panel 7
Figure 9: A Hydraulic Actuator Circuit 8
Figure 10: Control Pendant 8
Figure 11: Cable and Connector Routing 9
Figure 12: Pneumatic Subsystem 10

DISCLOSURE STATEMENT

Design information, processes and techniques discussed within this report may be patent pending. Do not disclose to other agencies, persons, companies, or entities.

The Contractor grants Caltrans and the FHWA a royalty-free, non-exclusive and irrevocable license to reproduce, publish or otherwise use, and to authorize others to use, the work and information contained herein for government purposes.

DISCLAIMER STATEMENT

The research report herein was performed as part of the Advanced Highway Maintenance and Construction Technology Program (AHMCT), within the Department of Mechanical And Aeronautical Engineering at the University of California, Davis and the Division of New Technology and Materials Research at the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, state and federal governments and universities.

The contents of this report reflect the views of the author(s) who is (are) responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the offical views or policies of the STATE OF CALIFORNIA or the FEDERAL HIGHWAY ADMINISTRATION and the UNIVERSITY OF CALIFORNIA. This report does not constitute a standard, specification, or regulation.

1. Introduction

Painting roadway markings on the road surface is a tedious and hazardous maintenance procedure. To create the markings, a work crew first must section off a lane area and then layout a set of stencils corresponding to the desired message. Once everything is in place, the crew uses a paint sprayer and coats the road surface and stencils with paint. Where there are open spots in the stencil is where the paint is deposited on the surface. After a suitable drying period, the stencils are removed and the lane reopened. Each time this process is repeated, the crew is exposed to traffic hazards since the crew must leave the safety of their vehicles and work on the open roadway.

At the University of California, Davis Advanced Highway Maintenance and Construction Technology (AHMCT) Center, we have developed a very long reach pantograph-type robot manipulator to accomplish the painting operations without the use of stencils. When the unit is fully extended, it has a reach of almost 4.575 meters [15 feet]. The base can rotate approximately 270 degrees. One of the unique features of this design is that all of the joint actuators are located at the base of the manipulator. This co-location leads to extremely high stiffness to weight ratios since the manipulator structure does not need to support the weight of the actuators.

As shown in Figure 1, the robot manipulator has two degrees of freedom: $\boldsymbol{R}$ and θ. The movement in the $\boldsymbol{R}$ direction is controlled by a linear hydraulic actuator, operating on the pivot of the pantograph. Motion is amplified by the pantograph mechanism according to an 8.3:1 ratio. Thus, for each 2.54 centimeters [1 inch] the hydraulic actuator moves, the tool center point moves linearly 21.082 centimeters [8.3 inches]. Rotation of the manipulator is controlled by a hydraulic motor mounted in the base. Position of the manipulator is determined by two optical encoders. One optical encoder is mounted on the output shaft of the hydraulic motor that rotates the manipulator and the other encoder is mounted on the pivot of one of the manipulator's link. Note that the extension length of the manipulator is indirectly sensed through the rotation angle of the upper manipulator link.

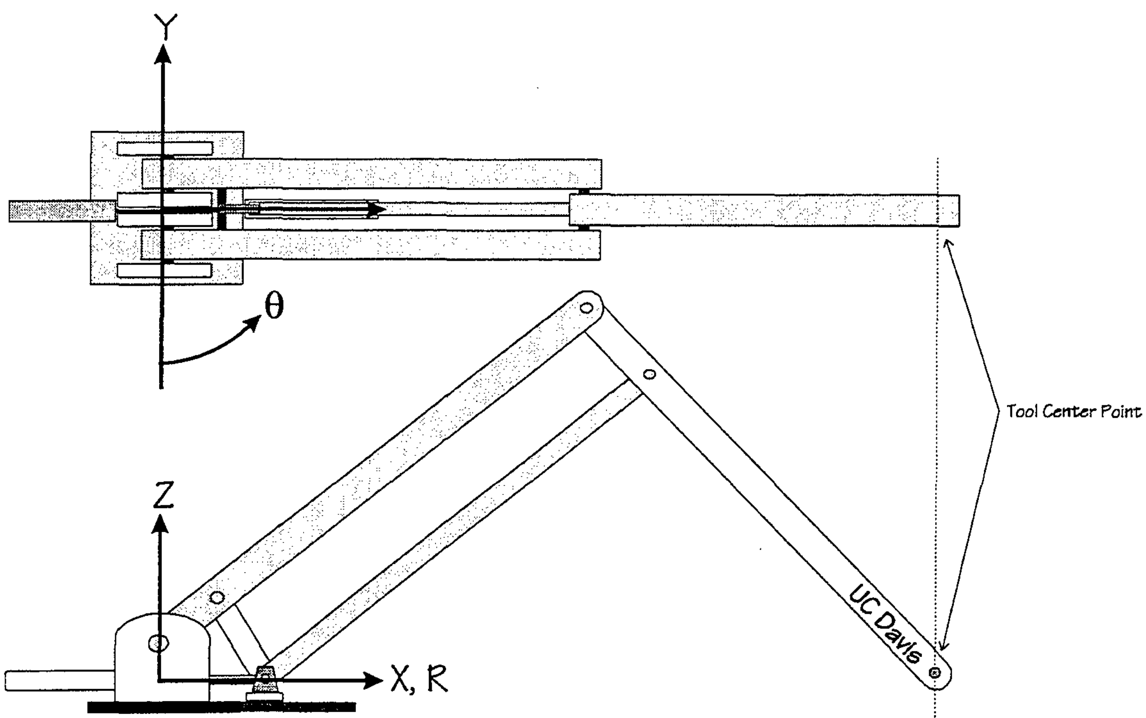


Figure 1: Manipulator Layout

In order to paint the roadway markings in a consistent fashion, the manipulator must move the tool center point (to which is mounted the painting end-effector) from point-to-point locations in an accurate way, as well as follow accurately a prescribed trajectory motion. The manipulator must follow a prescribed trajectory in order to create acceptable letter profiles with an evenly coated painted surface. The following sections describe the major components and subsystems necessary to accomplish the desired painting operations.

2. System Description

The BASR is composed of six (6) major components operated in conjunction with five (5) supporting subsystems. The six major components are:

1) the truck with its stabilizer feet,
2) the paint subsystem,
3) the reflective bead subsystem,
4) the main power unit,
5) the robot arm manipulator,
and 6) the manipulator end-effector

The five supporting subsystems are:

1) the hydraulic supply subsystem,
2) the electrical subsystem,
3) the pneumatic system,
4) the computer control system,
and 5) the embedded software.

Each component and subsystem are described in subsequent sections below.

2.1 Truck

The BASR support vehicle is a 1989 Ford F-350 9 ton flat bed truck modified by Lakeview Metal (Nice, CA). The location of major components on the truck are identified in Figure 2. For added vehicle stability, a set of stabilizer feet (Appendix A) at the front of the truck bed and a set of stabilizer feet (Appendix A) at the back of the bed have been installed. These feet are under computer control during automatic operations, but can be controlled manually (valves 2 through 4) from the hydraulic hand-valve stack located in the utility box on the side of the truck. Major utilities, such are hydraulic, pneumatic and electrical service, are distributed from the main power unit to the rest of the system through a trough located in the middle of the truck bed.

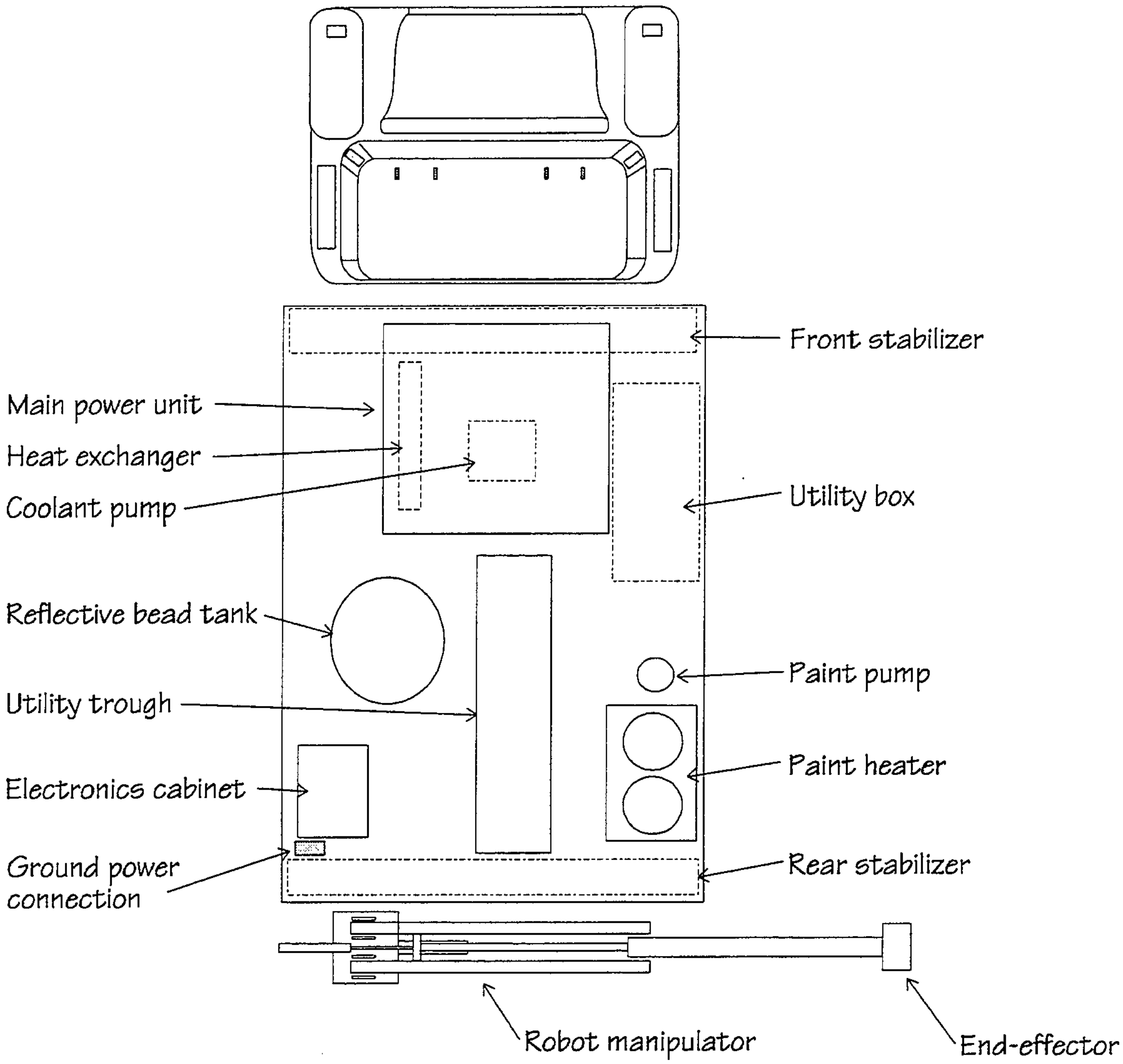


Figure 2: Truck Layout

2.2 Paint Subsystem

The paint subsystem is composed of a paint pump and associated plumbing to transport the paint from the paint supply bucket to the spray tip located at the end of the robot. The paint pump is a Graco Viscount I, operating on a 103.35 bar [1500 psi] hydraulic supply with a 2:1 compression ratio, thus producing an output pressure of 206.7 bar [3000 psi]. The paint system schematic is shown in Figure 3. The operation of the paint pump is under automatic control of the computer. The controlling hydraulic solenoid valve is located in the utility box. During cleaning and maintenance operations of the pump, the pump operation can be controlled by hand-valve number 1 on the hand-valve stack located in the utility box.

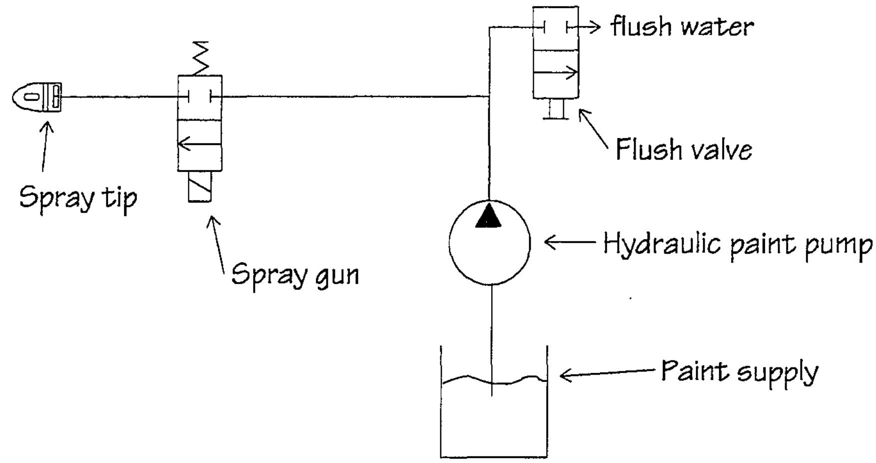


Figure 3: Paint System Schematic

In order to maintain optimal paint viscosity, the paint temperature is regulated by a paint heating system. The heating system schematic is shown in Figure 4. This heating system obtains its heat from an engine exhaust gas heat exchanger located in the main power unit. This heat exchanger operates by passing diesel engine exhaust around a tube containing coolant from the paint bucket reservoir. The coolant is circulated by an electrically driven water pump. A thermostat, which senses the coolant temperature, controls whether the pump recirculates the coolant or redirects the coolant to the heat exchanger. The temperature control schematic is shown in Figure 5. The coolant loop is sealed against outside contamination. The heated paint bucket receptacle has enough capacity for two (2) 18.9 liter [five gallon] containers.

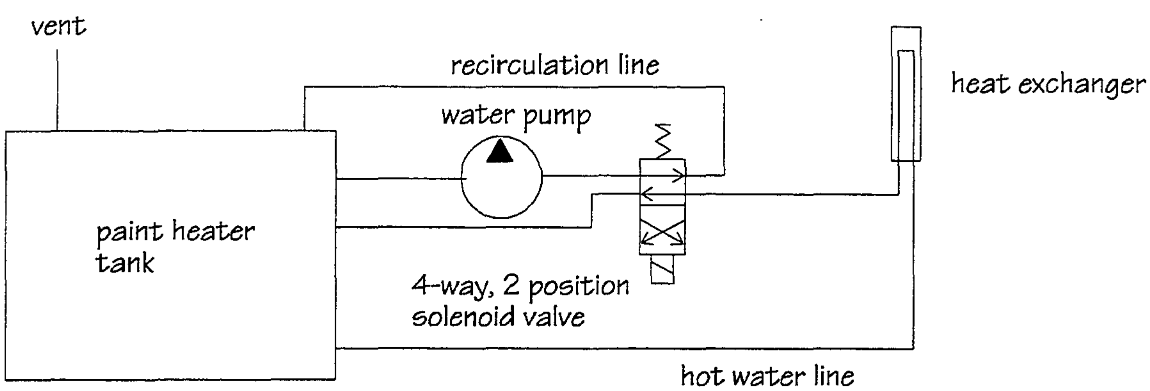


Figure 4: Paint Heating System

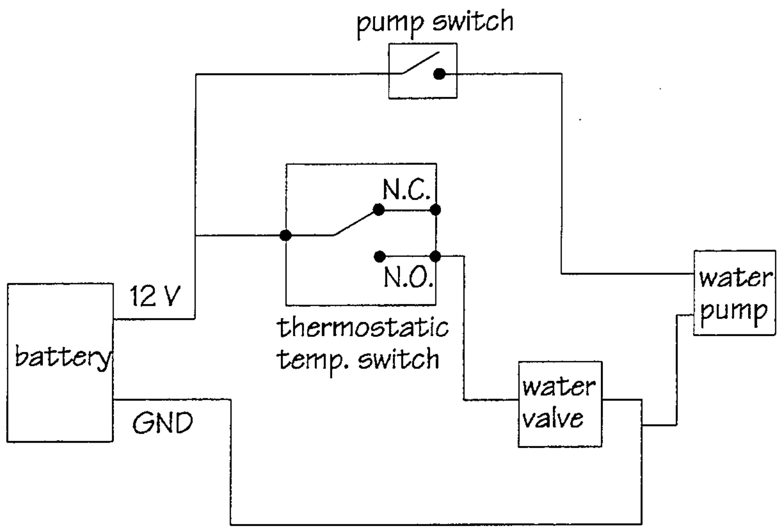


Figure 5: Temperature Control Schematic

After the paint has passed through the paint pump, it is transported to the paint gun by a 7.625 meter [25 feet] high pressure flexible hose. The paint flow then enters a Binks electrically controlled paint gun. This gun opens and closes under authority of an electrical signal from the computer. A Binks carbide spray tip (P/N 1380) converts the paint flow into a rectangular spray pattern on the ground.

2.3 Reflective Bead System

In order to increase reflectivity of the painted pavement markings, reflective glass spheres are placed in the pavement markings while it is still wet. These glass spheres are sprayed on using a pneumatically controlled applicator. This applicator contains a valve and a spreader nozzle. The spheres are supplied from a pressurized supply tank operating at 4.134 bar [60 PSI]. The operation of the applicator, as well as the pressurization of the supply tank, are computer controlled. A schematic of the system is shown in Figure 6.

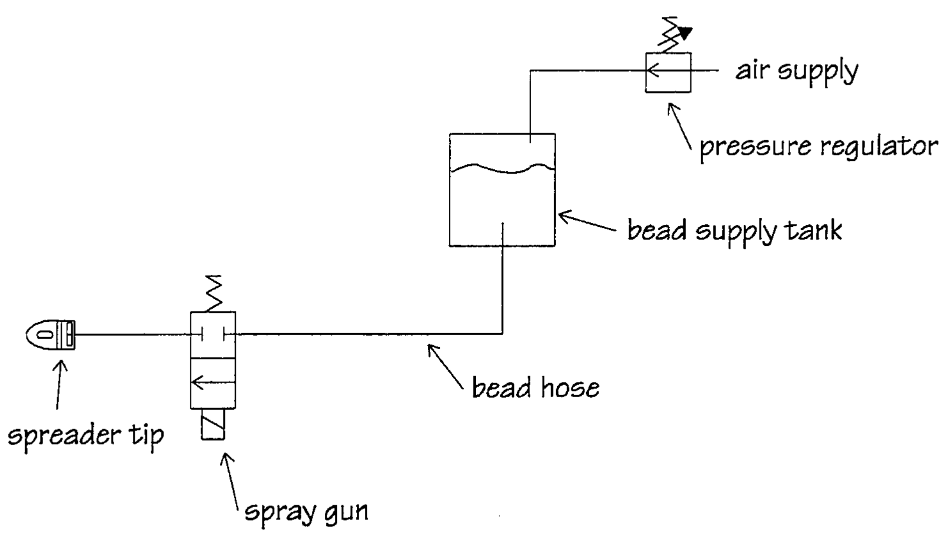


Figure 6: Reflective Bead System Schematic

2.4 Main Power Unit

The main power unit (MPU) provides electrical, pneumatic, and hydraulic power to the entire machine. The MPU layout is shown in Figure 7. The primary motive power for the MPU is a Deutcsh 4 cylinder air-cooled diesel engine. A maximum of 8000 watts, 220/120 volts is provided by a belt-driven alternator. A single-cylinder belt-driven air compressor produces 6.89 bar [100 PSI]. Finally, attached to the crankshaft of the engine are two (2) hydraulic pumps, both drawing hydraulic fluid from a common fluid reservoir. The hydraulic pump nearest to the engine is a pressure-compensated variable-displacement pump producing 37.9 liters [10 gallons] per minute (LPM) [(GPM)] at 206.7 bar [3000 PSI]. Pressure variations are suppressed by a gas-charged fluid accumulator. This pump provide primary motive power to the robotic manipulator and end-effector. The pump at the far-end is a constant displacement pump producing 37.9 LPM [10 GPM] at 103.35 bar [1500 PSI]. This pump provides motive power for the stabilizer feet and paint pump. Fluid output from both pumps is controlled by a toggle switch located under the main control panel for the MPU. Hydraulic fluid is filtered before it is returned to the reservoir.

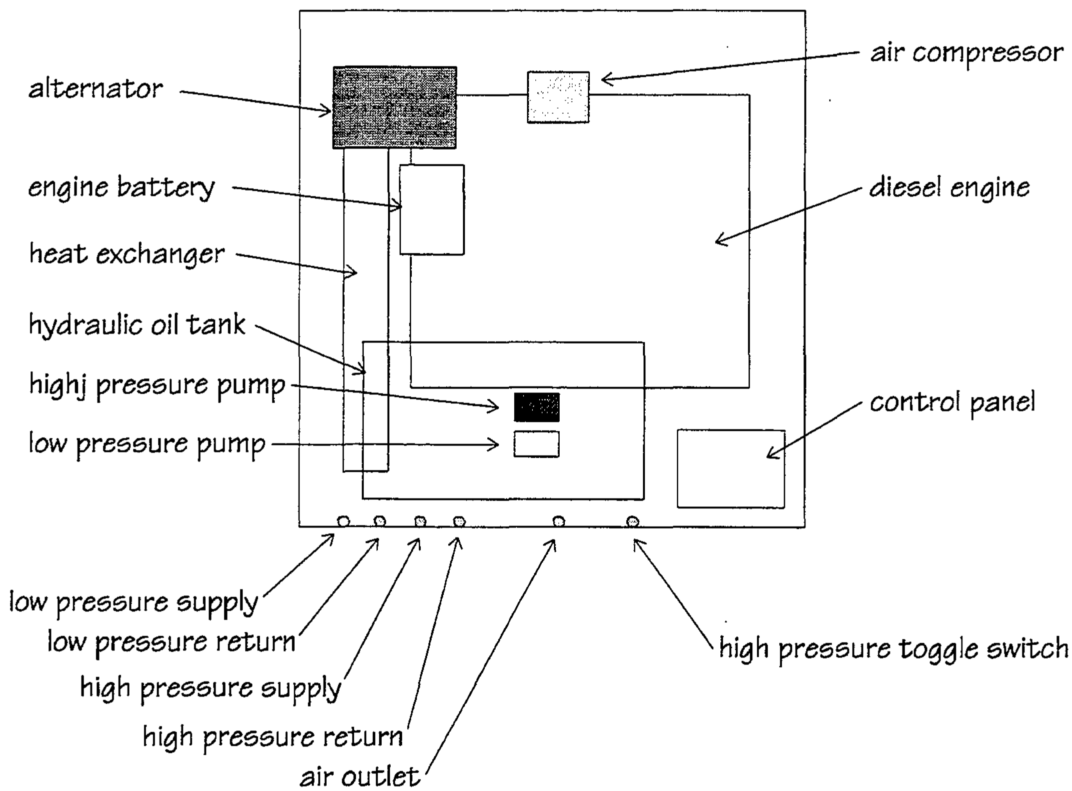


Figure 7: Main Power Unit Layout

Located on the main control panel (Figure 8) are electrical service outlets. Two outlets of 20 amps service and one outlet of 40 amps service are provided. Gauges to monitor MPU performance are located below the electrical outlets. At the bottom of the control panel is located the engine starter key switch and the engine pre-heat button. Both of these controls are used in starting the diesel engine.

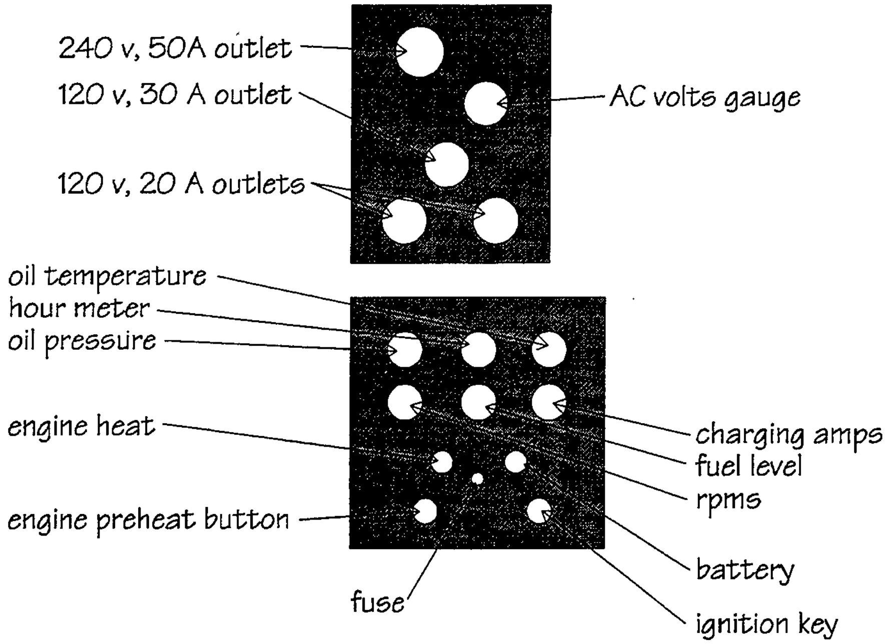


Figure 8: MPU Control Panel

2.5 The Manipulator

The manipulator is of a pantograph-style linkage type, custom built and designed at the University of California at Davis. Further descriptions and design information can be found in Robert H. Olshausen's 1996 U.C. Davis Master's Thesis, "Development of an Articulating Robotic Arm for Spray Painting on Roadways" (report number UCD-ARR-96-09-30-01). For convenience, a copy has been included as Volume II of this report.

2.6 The End-Effector

The manipulator end-effector was custom built and designed at the University of California at Davis. Further descriptions and design information can be found in Richard A. McGrew's 1996 U.C. Davis Master's Thesis, "A Robotic End-Effector for Roadway Stenciling" (report number UCD-ARR-96-06-30-01). For convenience, a copy has been included as Volume III of this report.

2.7 Supporting Subsystems

Five major subsystems are required to operate BASR. The five supporting subsystems are the hydraulic supply subsystem, the electrical subsystem, the pneumatic subsystem, the computer control subsystem, and the embedded software contained on each controller.

2.7.1 The Computer Control Subsystem

BASR is controlled by five (5) single board computers (SBC). Each SBC is manufactured by Z-World, Inc. (Davis, CA). The embedded software on the SBCs is described in Section 2.7.5. Four of the SBCs are Tiny Giants interfaced to a specially designed and manufactured encoder/servo valve control card (ESVCC). These four SBCs are directly associated with a hydraulic actuator. A schematic of the ESVCC is provided in Appendix B. The ESVCC is composed of a HP2020 quadrature decoder chip, a AD667 digital-to-analog converter (D/A), and associated logic control circuitry. The SBC receives position information from the quadrature decoder chip, computes the necessary servo valve position using a predetermined control law, and then outputs the necessary signal to the D/A chip. The output from the D/A chip then is conditioned by the voltage-to-current converter, which directly positions the servo valve, controlling the output of hydraulic flow to the actuator. Figure 9 is a schematic of the control architecture of one of the hydraulic actuator circuits.

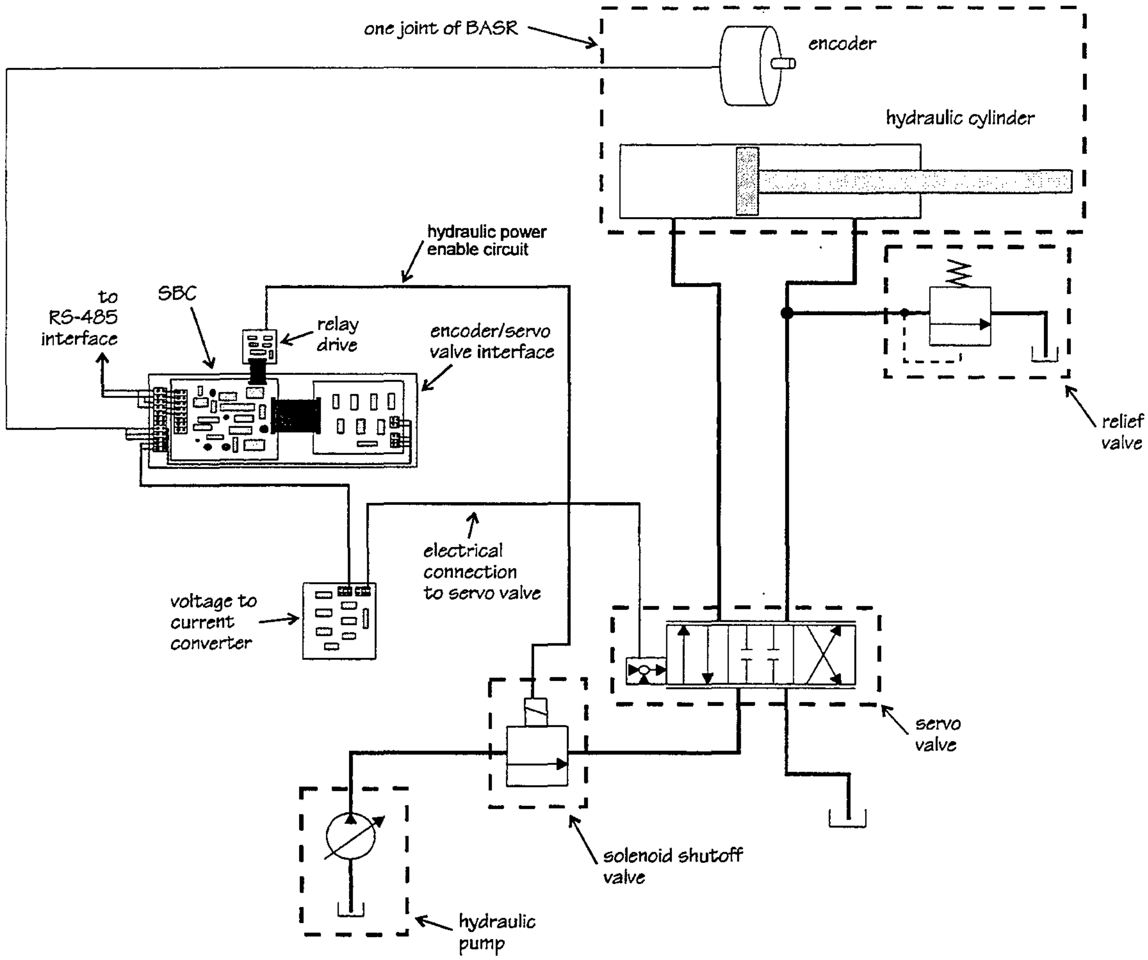


Figure 9: A Hydraulic Actuator Circuit

The fifth SBC is contained in the hand-held pendant. This pendant has a keypad interface, as well as a joystick and three (3) control knobs for positioning BASR. The keypad is used for entering textual information, such as the desired lettering to create on the pavement. A LCD screen displays status and informational messages to the operator.

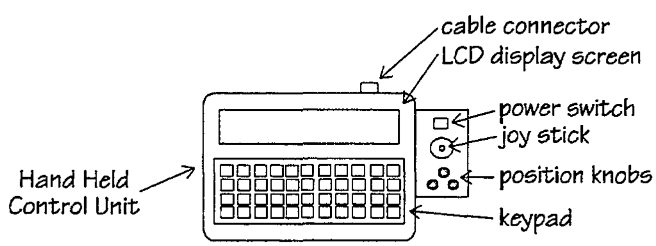
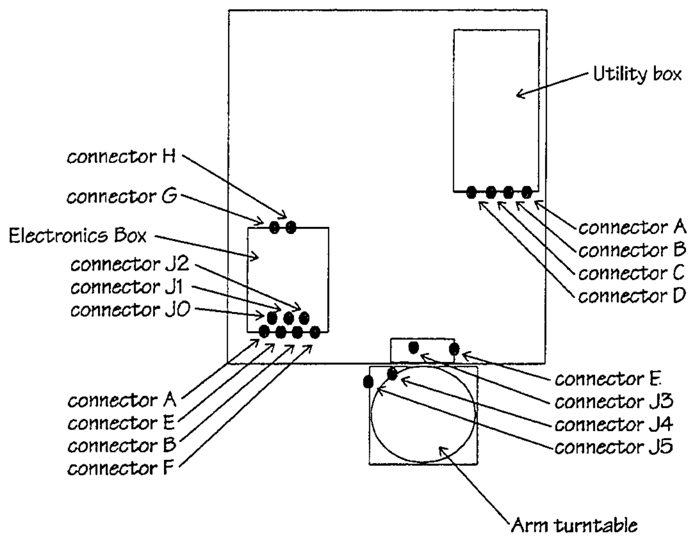


Figure 10: Control Pendant

All SBCs are linked together using RS-485 local area networking. Commands are passed through the network interface and are composed of text strings. Each command string is composed of five parts: a start character, an address, a command, the command option modifiers, and the end character. Appendix C has a list of acceptable commands.

In order to route the electrical signals from the SBCs to different parts of BASR, various cable interconnects are used. The cable connectors are keyed and different sizes and gender are used to prevent accidental connections to the wrong cable plug. Figure 11 shows the location of all connectors. A wiring diagram and pinout table is provided in Appendix D.



name	function
connector A	stabilizer control
connector B	transducer/communications
connector C	transducer/solenoid control
connector D	stabilizer limit switches
connector E	end effector accessories
connector F	accessory control
connector G	pendant
connector H	main power
connector J0	translation control/sensor
connector J1	rotation control/sensor
connector J2	extension control/sensor
connector J3	extension servo control
connector J4	extension sensor
connector J5	trans./rot/ control/sensor

Figure 11: Cable and Connector Routing

2.7.2 Hydraulic Supply Subsystem

Hydraulic power is supplied to BASR via two (2) engine driven hydraulic pumps routed through two separate hydraulic systems. One pump provides 103.35 bar [1500 PSI] and the other provides 206.7 bar [3000 PSI]. The 206.7 bar [3000 PSI] system operates the actuators related to the manipulator and end-effector. The 103.35 bar [1500 PSI] system operates the vehicle's stabilizer feet and the paint pump. Manual operation is provided for the operation of all accessories on the 103.35 bar [1500 PSI] system. The operating hand-valves are located in the utility box on the right side of the truck. Additionally, should it be desirable to use external hydraulic power, ground-service quick-disconnect return and supply ports are provided. The ports are located on the truck near the electronics cabinet. A pressure reducing valve is connected to the ground service port to automatically provide 103.35 bar [1500 PSI] from the 206.7 bar [3000 PSI] ground supply port. Various valves are provided through out the system in order to isolate different hydraulic circuits. The hydraulic system schematic is provided in Appendix K.

2.7.3 Electrical Subsystem

An 8000 watt belt-driven alternator provides the main AC power to BASR. AC power is used to power the paint heater coolant pump and the 12 volt power transformer for the SBCs.

2.7.4 Pneumatic Subsystem

Pneumatic power is supplied by a single-cylinder engine driven compressor. Maximum supply pressure is 6.89 bar [100 psi]. Air pressure is supplied via a manifold to four (4) valves operating the reflective bead pressurization system, the electronics vortex air conditioner, the end-effector stowage system, and the end-effector stowage system lock-out. Figure 13 is a schematic of the pneumatic system.

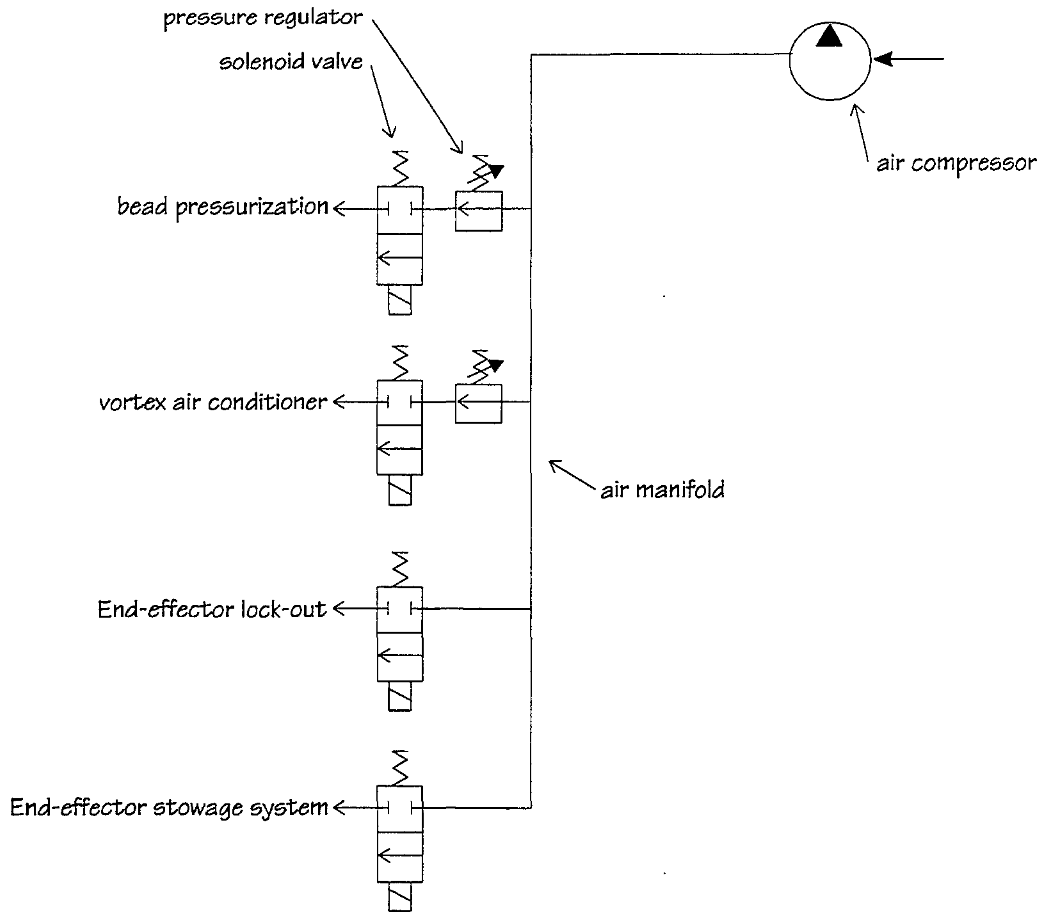


Figure 13: Pneumatic Subsystem

2.7.5 Embedded Software

Each of the SBCs contains embedded compiled program code written in the "C" programming language. Although each of the four manipulator joint SBCs respond to the same commands, the programming varies slightly to account for the kinematics of the mainpulator joint it is controlling. Complete source code listings are provided in Appendices I through L. As of this writing, a complete source code listing of the software for the pendant controller is not available. Operation of the trajectory planner and control module is described in the research paper provided in Appendix E.

Letter profiles for BASR are generated by the techniques discussed in the Master's Thesis of Richard Blank "Algorithms and Robotic Hardware Improvements for Painting of Roadway Markings" (report number UCD-ARR-97-06-15-01). A copy of the thesis has been included as Volume IV of this report. Once the letter profiles have been generated off-line, the data is encoded and programmed into the hand-held pendant control software. When the operator has selected a certain letter, its coordinates are retrieved from the pendant's memory and downloaded to the joint SBCs. These coordinates are then executed by the SBCs and move BASR accordingly.

3. System Operation

As of this writing, field testing has not been completed. As such, a complete description of field operational procedures cannot be provided. However, a description of laboratory operational procedures can be provided.

3.1 System Startup

BASR system startup consists of clearing the workspace of all obstacles and personnel. Then, all restraining chains and harnesses are removed. Finally, the main power unit can be started. MPU starting is accomplished by first depressing the engine preheat button, then turning the ignition key until the engine cranks. Once the engine has started, the engine preheat button and key is released. While the engine operation stabilizes and warms up, striping paint can be mixed and loaded into the paint bucket receptacle. The high pressure enable toggle switch on the MPU is then moved to the "enable" position, thus allowing the main hydraulic system to reach operating pressure. After allowing a few seconds for pressure to build, the paint pump can then be primed. This is accomplished by inserting the pump siphon hose into the paint bucket. The pump outlet valve is opened and the outlet hose placed in a suitable waste receptacle. Hand valve #1 is then operated to allow the paint pump to prime. Once air-bubble-free paint begins to flow from the outlet hose, the outlet valve can be shut off and the hydraulic hand-valve released to stop pump operations. The spray gun is then opened, and the paint pump is cycled once again to fill the paint hose. Once the paint hose is free of air, the spray gun and paint pump are deactivated. A spray tip is then installed on the spray gun. "Calibration mode" is then selected on the control pendant. The manipulator and end-effector then begin automatic check-up and calibration procedures. Once complete, BASR is ready to be used.

3.2 Normal Operations

Once the truck is located at the proper work site, from the control pendant, the operator selects the menu item to lower the stabilizer feet. By using the video screen to target the end-effector, the operator can position BASR by using the manual control joystick and knobs. Properly positioned, the operator types in the message on the control pendant to be painted on the ground. After presenting a confirmation message, the controller then pressurizes the paint system, moves BASR, thus marking the pavement. The operator then selects the "Stow" item from the control pendant. After BASR is stowed, the operator can then drive away from the work site.

3.3 Clean Up

After all operations are complete for the day, BASR cleanup merely involves removing the spray tip and cleaning off the paint deposits. The paint is replaced with water, and water is pumped through the system to remove the paint. The high pressure hydraulic system enable switch is then moved to the "disable" position. The controller power switch is then turned off, and all restraints and harnesses replaced. Finally, the MPU is turned off.

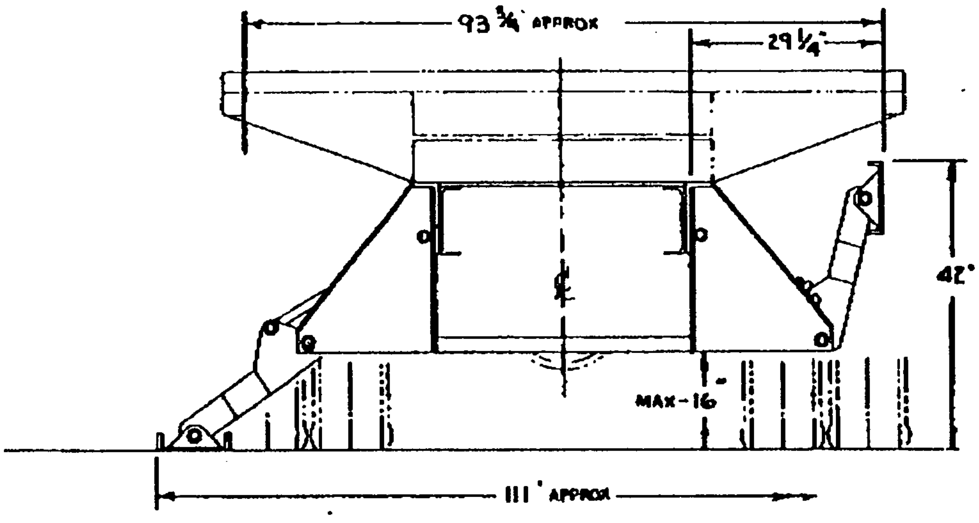
4. Conclusions

Although field testing of BASR is not yet complete, from the results of laboratory testing, a reliable and easy to use pavement marking system has been created. This system maximizes personnel safety, increases efficiency, and improves quality.

Appendix A

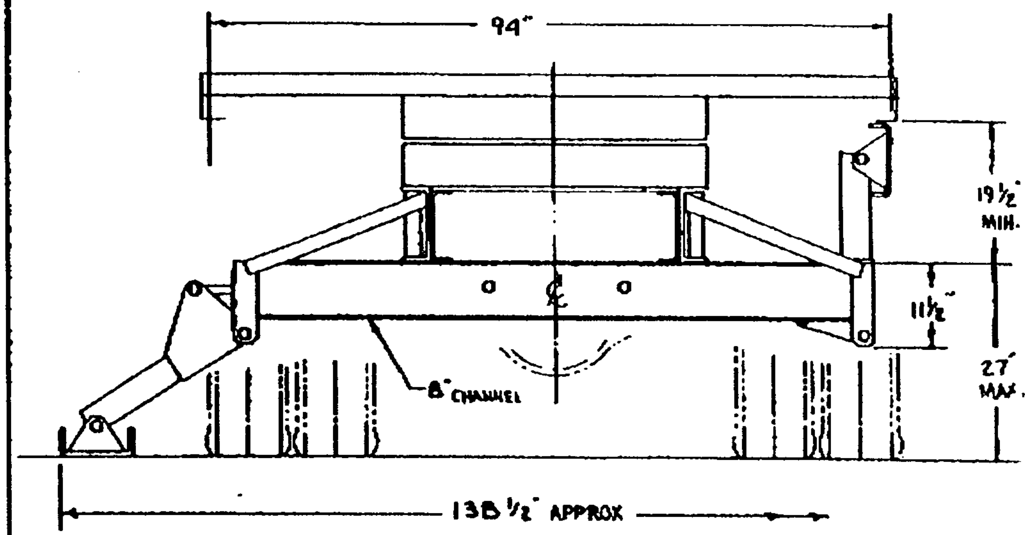
Stabilizer Feet Data

STANDARD ON HILIFTS



MODEL 4045

OPTIONAL
WIDER REACH FOR MORE STABILITY



MODEL 4046

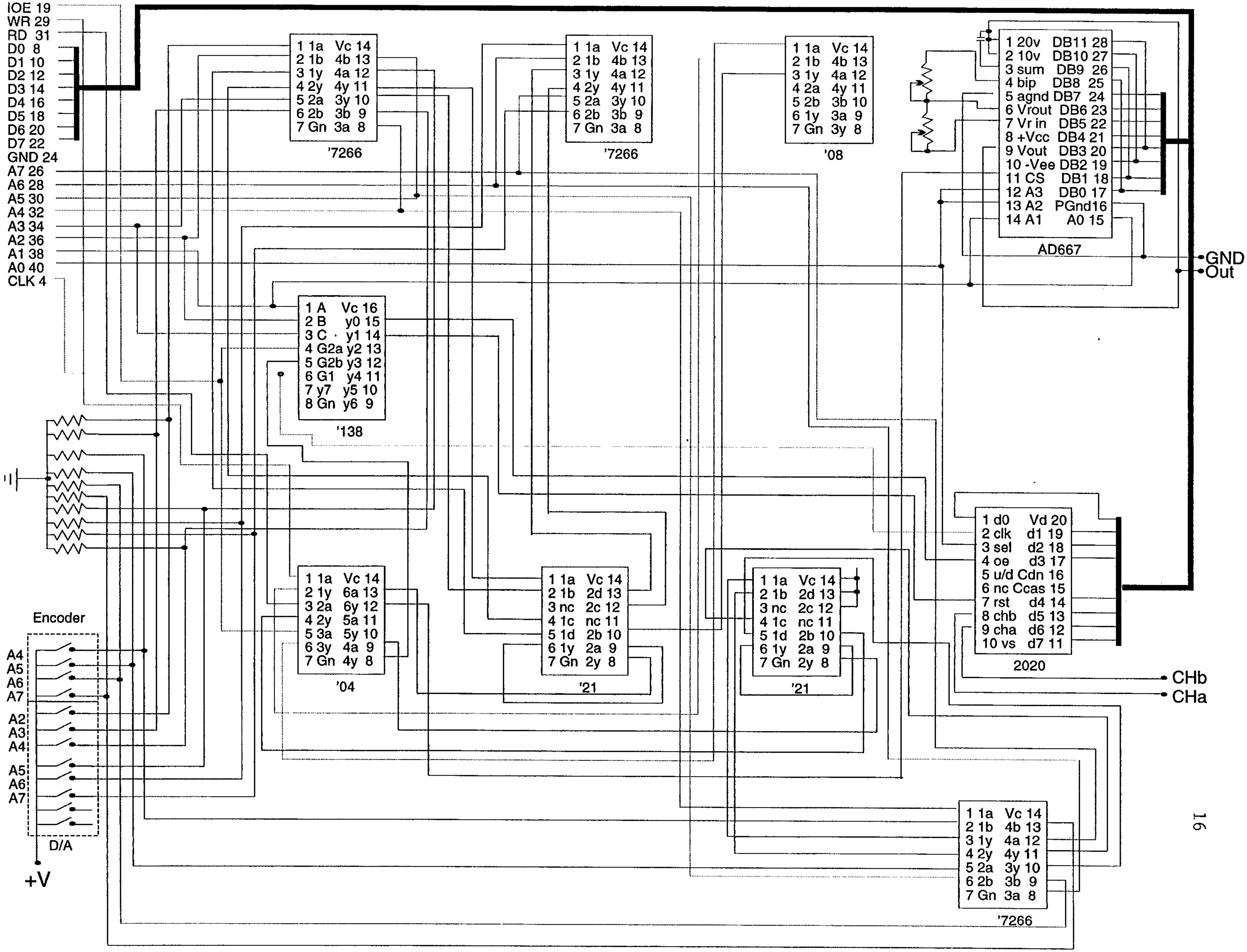
1. Lock Valves standard
2. Honed I.D. tubing
3. Centrally located grease zerks
4. Other models available upon request
5. Units shipped assembled

THS PRODUCTS, INC.
P.O. BOX 997
ELK GROVE, CA 95759-0997
916-684-9564 • 916-423-2088 • FAX 916-684-1634

THS 008

Appendix B

Schematic of ESVCC Board



IOE 19
WR 29
RD 31
D0 8
D1 10
D2 12
D3 14
D4 16
D5 18
D6 20
D7 22
GND 24
A7 26
A6 28
A5 30
A4 32
A3 34
A2 36
A1 38
A0 40
CLK 4
1 1a Vc 14
2 1b 4b 13
3 1y 4a 12
4 2y 4y 11
5 2a 3y 10
6 2b 3b 9
7 Gn 3a 8
'7266
1 1a Vc 14
2 1b 4b 13
3 1y 4a 12
4 2y 4y 11
5 2a 3y 10
6 2b 3b 9
7 Gn 3a 8
'7266
1 1a Vc 14
2 1b 4b 13
3 1y 4a 12
4 2a 4y 11
5 2b 3b 10
6 1y 3a 9
7 Gn 3y 8
'08
1 20v DB11 28
2 10v DB10 27
3 sum DB9 26
4 bip DB8 25
5 agnd DB7 24
6 Vrout DB6 23
7 Vr in DB5 22
8 +Vcc DB4 21
9 Vout DB3 20
10 -Vee DB2 19
11 CS DB1 18
12 A3 DB0 17
13 A2 PGnd16
14 A1 A0 15
AD667
GND
Out
1 A Vc 16
2 B y0 15
3 C y1 14
4 G2a y2 13
5 G2b y3 12
6 G1 y4 11
7 y7 y5 10
8 Gn y6 9
'138
1 d0 Vd 20
2 clk d1 19
3 sel d2 18
4 oe d3 17
5 u/d Cdn 16
6 nc Ccas 15
7 rst d4 14
8 chb d5 13
9 cha d6 12
10 vs d7 11
2020
CHb
CHa
Encoder
A4
A5
A6
A7
A2
A3
A4
A5
A6
A7
D/A
+V
1 1a Vc 14
2 1y 6a 13
3 2a 6y 12
4 2y 5a 11
5 3a 5y 10
6 3y 4a 9
7 Gn 4y 8
'04
1 1a Vc 14
2 1b 2d 13
3 nc 2c 12
4 1c nc 11
5 1d 2b 10
6 1y 2a 9
7 Gn 2y 8
'21
1 1a Vc 14
2 1b 2d 13
3 nc 2c 12
4 1c nc 11
5 1d 2b 10
6 1y 2a 9
7 Gn 2y 8
'21
1 1a Vc 14
2 1b 4b 13
3 1y 4a 12
4 2y 4y 11
5 2a 3y 10
6 2b 3b 9
7 Gn 3a 8
'7266

Appendix C

Joint SBC Commands

```
Outgoing message format
          |>|x|Command Data|.|

     > indicates command to follow
     . indicates end of message

Inbound message format
          |>|return data|.|
     -or-
          OK

     > indicates response to follow
     . indicates end of message

(x is unit number)
```

Outgoing	**Reponse**	**Comment**
>xLy.	>Lnnnn.	Request analog input on channel *y*. Value *nnnn* is returned.
>xH1.	OK	High voltage enable.
>xH0.	OK	High voltage disable.
>xVyyy.	OK	High voltage pattern *yyy*.
>xPAxxyy.	OK	Initialize PIO port A, mode *xx*, control word *yy*
>xPBxxyy.	OK	Initialize PIO port B, mode *xx*, control word *yy*
>xDOAyyy.	OK	output value *yyy* on PIO port A
>xDOByyy.	OK	output value *yyy* on PIO port B
>xDIA.	>DIAyyy.	return value *yyy* from PIO port A
>xDIB.	>DIByyy.	return value *yyy* from PIO port B
>xGPPyyyy.	OK >GPPyyyy.	Change positional proportional gain setting to *yyyy*. If *yyyy* = __-1, report current gain.
>xGPIyyyy.	OK >GPIyyyy.	Change positional integral gain setting to *yyyy*. If *yyyy* = __-1, report current gain.

>xGPDyyyy.	OK >GPDyyyy.	Change positional derivative gain setting to *yyyy*. If *yyyy* = __-1, report current gain.
>xGVPyyyy.	OK >GVPyyyy.	Change velocity proportional gain setting to *yyyy*. If *yyyy* = __-1, report current gain.
>xGVIyyyy.	OK >GVIyyyy.	Change velocity integral gain setting to *yYyy*. If *yYyy* = __-1, report current gain.
>xGVDyyyy.	OK >GVDyyyy.	Change velocity derivative gain setting to *yyyy*. If *yyyy* = __-1, report current gain.
>xELyyyyyyyzzzzzzzppp.	OK	Load encoder position *yyyyyyy*, velocity *zzzzzzz*, pio mode *ppp* and autoincrement point list. Move will our on X command.
>xESyyyyyyy.	>ESxxxxxxxzzzzzzz. >ESwwwwww.	If *yyyyyyy* =______-1, report current position *xxxxxxx* and velocity *zzzzzzz*. If *yyyyyyy* = ______-2, report control law actuation *wwwwww*.
>xERxxxxyyyyyyyzzzzzzzppp.	OK	Modify data point at *xxxx* with new encoder position *yyyyyyy*, velocity *zzzzzzz*, pio mode *ppp*.
>X.		Execute loaded moves.
>xS.	OK	Emergency shutdown.
>xIyyy.	OK	Change timing interval. if *yyy* = 0-1, report timing interval.
>xR.	>Ryyy.	Report software revision level.
>xMVyyyyy.	OK	Max velocity before auto shutdown.
>xMLyyyyy.	OK	Max extension limit before shutdown.
>xMSy.	OK	Gain scaling.

>xESxxxx.	>ESyyyyyyyzzzzzzzppp.	Show position *(yyyyyyy)*, velocity *(zzzzzzz)* and pio mode *(ppp)* for point xxxx.
>xERxxxxyyyyyyyzzzzzzzppp.	OK	Replace position *xxxx* with position *yyyyyyy*, velocity *zzzzzzz* and pio mode *ppp*.
>xELyyyyyyyzzzzzzzppp.	OK	Load position *yyyyyyy*, velocity *zzzzzzz*, pio mode *ppp* and auto-increment point list.
>xEC.	OK	Clear point counter.

Appendix D

Connector Pinout Table

Location	pin	color	function	Connects to:	pin	color	function
Moving carriage connector	A	orange	cable 0 servo +current	Servo valve	B, D	orange	+current
	B	blue	cable 0 servo -current		A, C	blue	-current
	C	gray	cable 0 encoder A channel	Encoder			
	D	purple	cable 0 encoder B channel	(1200 pulses/rev)			
	H	yellow	cable 0 encoder Z channel				
	F	green	N/C				
	J	red	cable 0 encoder +power				
	G	black	cable 0 encoder -power				
	E	white	N/C				
	Q	brown	N/C				
	Z	orange	cable 1 servo +current	Servo valve	B, D	orange	+current
	Y	blue	cable 1 servo -current		A, C	blue	-current
	X	gray	cable 1 encoder A channel	Encoder			
	W	purple	cable 1 encoder B channel	(88000 pulses/rev)			
	R	yellow	cable 1 encoder Z channel				
	S	green	cable 1 encoder case gnd				
	T	red	cable 1 encoder +power				
	U	black	cable 1 encoder -power				
	V	white	N/C				
	K	brown	N/C				
Cable 0 board end	L	orange	servo +current				
	M	blue	servo -current				
	P	gray	encoder A channel				
	N	purple	encoder B channel				
	D	yellow	encoder Z channel				
	E	green	N/C				
	F	red	encoder +power				
	G	black	encoder -power				
	H	white	N/C				
	J	brown	N/C				
Cable 1 board end	L	orange	servo +current				
	M	blue	servo -current				
	P	gray	encoder A channel				
	N	purple	encoder B channel				
	D	yellow	encoder Z channel				
	E	green	N/C				
	F	red	encoder +power				
	G	black	encoder -power				
	H	white	N/C				
	J	brown	N/C				
Cable 2 board end	L	orange	servo +current	Servo valve	B, D	red	+current
	M	blue	servo -current		A, C	black	-current
	P	gray	encoder A channel	Encoder			

	N	purple	encoder B channel	(88000 pulses/rev)		
	D	yellow	encoder Z channel			
	E	green	N/C			
	F	red	encoder +power			
	G	black	encoder -power			
	H	white	N/C			
	J	brown	N/C			
Pendant connector	A		CT014			
	B		CT114			
	C		CT214			
	D		CT314			
	E		CT +pwr 1			
	F		CT414			
	G		CT514			
	H		CT614			
	J		CT714			
	K		CT +pwr 2			
	L		GND			
	M		K			
	N		CT relay power			
	P		spare			
	R		spare			
	S		spare			
	T		+RS485 Chan. 0			
	U		-RS485 Chan. 0			
	V		+RS485 Chan. 1 (spare)			
	W		-RS485 Chan. 1 (spare)			
	X		+pwr			
	Z		GND (shield)			
	a		piob 1 (note: no piob 0!)			
	b		piob 2			
	c		piob 3			
	d		piob 4			
	e		piob 5			
	f		piob 6			
	g		piob 7			
	h		high voltage out 1			
	j		high voltage out 2			
	k		high voltage out 3			
	m		high voltage out 4			
	n		high voltage out 5			
	p		high voltage out 6			
	r		high voltage out 7			
	s		high voltage out 8			
End effector connector	A		+ 485	end effector	A	+ 485
(board end)	B		- 485		B	- 485
	C		spare		C	spare

	D	intr		D	intr
	E	- pwr		E	- pwr
	F	+ pwr		F	+ pwr
	G	spare		G	spare
	H	spare		H	spare
	J	spare		J	spare
	K	spare		K	spare

Connector A	A	ground
(utility box end)	B	spare
	C	front right foot up
	D	front right foot down
	E	front left foot up
	F	front left foot down
	G	rear right foot up
	H	rear right foot down
	J	rear left foot up
	K	rear left foot down
	L	paint pump +
	M	high pressure +
	N	spare
	P	spare

Connector A	A	ground		
(computer box end)	B	spare		
	C	front right foot up	relay #5 NC terminal	614
	D	front right foot down	relay #5 NO terminal	
	E	front left foot up	relay #2 NC terminal	314
	F	front left foot down	relay #2 NO terminal	
	G	rear right foot up	relay #3 NC terminal	414
	H	rear right foot down	relay #3 NO terminal	
	J	rear left foot up	relay #4 NC terminal	514
	K	rear left foot down	relay #4 NO terminal	
	L	paint pump +	relay #5 NO terminal	
	M	high pressure +	relay #4 NO terminal	
	N	spare		
	P	spare		

Connector B	A	low pressure transducer +
(utility box end)	B	low pressure transducer -
	C	high pressure transducer +
	D	high pressure transducer -
	E	rs-485 +
	F	rs-485 -
	G	spare
	H	shield

Connector C	A	high pressure transducer +
(utility box end)	B	high pressure transducer -

	C	high pressure solenoid +	
	D	high pressure solenoid -	
	E		
	F		
	G		
	H	shield	
Connector D	A	front right foot in	connector A #c
(utility box end)	B	front right foot out	f.r. limit switch
	C	front left foot in	connector A #E
	D	front left foot out	f.l. limit switch
	E	rear right foot in	connector A #g
	F	rear right foot out	r.r. limit switch
	G	rear left foot in	connector A #J
	H	rear left foot out	r.l. limit switch
Connector B	A	low pressure transducer +	
(computer box end)	B	low pressure transducer -	
	C	high pressure transducer +	
	D	high pressure transducer -	
	E	rs-485 +	
	F	rs-485 -	
	G	spare	
	H	shield	
Connector E	A	e/e retract power +	
(computer box end)	B	e/e retract power -	
	C	e/e brake power +	
	D	e/e brake power -	
	E	hyd extend valve power +	
	F	hyd extend valve power -	
	G	e/e gun power +	
	H	e/e gun power -	
	J	bead gun +	
	K	bead gun -	
	L	spare	
	M	spare	
	N	shield	
	P	no connection	
Connector F	A	refrigeration power +	
(computer box end)	B	bead pressure power +	
	C	strobe power +	
	D	spare	
	E	spare	
	F	spare	
	G	spare	
	H	ground	

Appendix E

Source Code Listing for Transverse Joint SBC

```
#use rtk.lib

/*
This program would run on the Little Giant, the Tiny Giant and the CPLC.
If port 0 is also used as the Dynamic C programming port, you have to
load the serial interrupt routine during run time by doing the following:

  (1) comment out:
#INT_VEC SER0_VEC Dz0_circ_int in z0232.lib
  (2) in the code, declare:
          extern void Dz0_circ_int();
  (3) load the routine with:
          reload_vec(14,Dz0_circ_int);

*/
/* #include <stdio.h> */
#define ON 1
#define OFF 0
#define FWD 1
#define REV 0
#define FALSE 0
#define TRUE 1

#define VERSION 4
#define SUBVERSION 01

#define CSAMPLE 900     /* clock periods: 512HZ */
#define CTIME 0.001953  /* clock time */

#define D2AOFFSET 2047
#define ACTSCALE 1

/* int task0(), task1(), task2(), backgnd();
int (*Ftask[])()={task0, task1, task2, backgnd}; */

int task1(), task2(), backgnd();
int (*Ftask[])() = {task1, task2, backgnd};

/*
#define NTASKS 4
#define TASK0 0
#define TASK1 1
#define TASK2 2 */

#define NTASKS 3
#define TASK1 0
#define TASK2 1

#define TASKSTORE_SIZE 500

void (*TaskDispatchTable[30])();

typedef union bytemode
 {
  int word;
  struct ByteStruct
   {
          char lsb;
          char msb;
   } byte;
 } BYTEMODE;

struct PointListPoint
{
  long ListPosition;
  long ListVelocity;
  int TimeSlices;
  char PIOmode;
};
```

```
struct PIOstruct
 {
  char PIOAmode;
  char PIOBmode;
 };

struct PointListPoint PointList[1001];
shared struct PIOstruct PIOstatus;

struct PositionData
 {
  int time[3];
  long position;   /* variables for data logging */
  long velocity;
  int point;
 };

struct PositionData data[410];

float VeloCorrFact[] =
{0.960719, 0.958142, 0.955486, 0.952749, 0.949934,
0.947038, 0.944064, 0.941011, 0.937879, 0.934669,
0.931381, 0.928015, 0.924572, 0.921051, 0.917453,
0.913779, 0.910029, 0.906202, 0.9023  , 0.898322,
0.89427 , 0.890142, 0.885941, 0.881665, 0.877316,
0.872893, 0.868397, 0.863829, 0.859189, 0.854477,
0.849694, 0.84484 , 0.839915, 0.83492 , 0.829855,
0.824721, 0.819518, 0.814247, 0.808907, 0.8035  ,
0.798026, 0.792486, 0.786879, 0.781206, 0.775468,
0.769665, 0.763798, 0.757868, 0.751874, 0.745817,
0.739698, 0.733517, 0.727275, 0.720972, 0.714609,
0.708186, 0.701704, 0.695163, 0.688564, 0.681908,
0.675195, 0.668426, 0.6616  , 0.65472 , 0.647784,
0.640795, 0.633752, 0.626656, 0.619508, 0.612308,
0.605057, 0.597755, 0.590404, 0.583003, 0.575553,
0.568056, 0.560511, 0.552919, 0.545281, 0.537597,
0.529869, 0.522096, 0.51428 , 0.50642 , 0.498519,
0.490575, 0.482591, 0.474566, 0.466502, 0.458399,
0.450258, 0.442079, 0.433863, 0.42561 , 0.417323,
0.409   , 0.400643, 0.392253, 0.38383 , 0.375375,
0.366889, 0.358372, 0.349825, 0.341248, 0.332644,
0.324011, 0.315352, 0.306666, 0.297954, 0.289218,
0.280457, 0.271673, 0.262867, 0.254038, 0.245188,
0.236318, 0.227428, 0.218519, 0.209591, 0.200646};

int datapointer, takedata;

int PointNumber, ExePointNumber;      /* place in point list */
shared long gActPosition, gActVelocity;  /* actual position and velocity */
shared long PseudoVelo;               /* IPC variable b/w pos and velo loop */
shared BYTEMODE Actuation;            /* system actuation */
float kpp, kpi, kpd;        /* system position gains 1 unit, I in .01 units */
int kvp, kvi, kvd;          /* system velocity gains 0.0001 units */
float kff;                  /* system feed forward gain 0.01 units */
int BeginMove;              /* change to TRUE to begin motion */
int NewMove;
int UseVelocityControl;     /* bypasses position controller */
int Acceleration;

BYTEMODE ServoNull;
shared long MaxVelo, MaxLimits, AbsLimits;
shared int ActScale;
int Control, Shutdown, Reset1, Reset2;
int UnitNumber, DumpSerial;
char InString[255], OutString[255];
char PowerFail[] = "!Power fail\0";
char SoftReset[] = "!Soft reset\0";
char tbuf[384], rbuf[384];
```

```
int AnalogInput (char *, char *);
int HighVoltage (char *, char *);
int ChangeHighVoltage (char *, char *);
int InitPIO (char *, char *);
int OutInPIO (char *, char *);
int SetGetGain (char *, char*);
int ControlLaw (char *, char *);
int DtoAOut (char *, char *);
int PosLoad (char *, char *);
int ExecuteMove (char *, char *);
int Diagnostic (char *, char *);
int ShutDown (char *, char *);
int RevisionLevel (char *, char *);
int MaxLimit (char *, char *);
void FatalErrorHandler (unsigned, unsigned);
float p2sin(float);
float p2cos(float);
void NotUsed(void);

/* extern void Dz0_circ_int(); */

/* 0x10 stores network node number, 0x12 stores null.lsb, 0x13 stores
 null.msb */

root main ()
{
 int index;

 if (wderror())
  {
   Shutdown = TRUE;
   Control = OFF;
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   op_kill_z1();
   op_init_z1 (19200/1200, InString, UnitNumber);
   while (1)
        {
         hitwd();
         if (check_opto_command() == 1)
           replyOpto22 (SoftReset, strlen (SoftReset), 0);
        }
  }

 _GLOBAL_INIT();

 ERROR_EXIT = FatalErrorHandler;
 BeginMove = FALSE;
 UseVelocityControl = OFF;
 Control = DumpSerial = OFF;
 Shutdown = takedata = OFF;
 NewMove = FALSE;
 Reset1 = Reset2 = ON;
 ExePointNumber = PointNumber = 0;
 datapointer = 0;
 gActVelocity = gActPosition = 0L;
 kpp = kpi = kpd = kff = 0;
 kvp = kvi = kvd = 0;
 MaxLimits = AbsLimits = 88000L;
 Acceleration = 5000;
 MaxVelo = 0L;
 ActScale = ACTSCALE;
 Actuation.word = 0;    /* zero! */
 PIOstatus.PIOAmode = 0x1;
 PIOstatus.PIOBmode = (char) 0;
 UnitNumber = ee_rd(0x10);
 ServoNull.byte.lsb = ee_rd(0x12);
 ServoNull.byte.msb = ee_rd(0x13);
```

```
 inport (0x82);                    /* clear the encoder */

 outport (0x81, ServoNull.byte.lsb);  /* zero the d-to-a board */
 outport (0x82, ServoNull.byte.msb);
 outport (PIOCA, 0xff);            /* pioa command */
 outport (PIOCA, 0x00);            /* set all bits for output */
 outport (0x41, (char)0x7);        /* disable interrupts */
 outport (PIODA, PIOstatus.PIOAmode);  /* pioa data */

 outport (PIOCB, 0xcf);            /* piob command */
 outport (PIOCB, 0x00);            /* set all bits for input */
 outport (PIOCB, PIOB_VEC);
 outport (PIOCB, 0x17);
 outport (PIOCB, 0xfe);
 outport (0x43, 0x7);              /* disable interrupts */
 /* outport (PIODB, PIOstatus.PIOBmode); */ /* piob data */
 outport (PIOCB, 0x7);

/*
 reload_vec (14, Dz0_circ_int);
 Dinit_z0(rbuf, tbuf, 384, 384, 4, 9600/1200, 0, 0);
*/

 op_init_z1 (19200/1200, InString, UnitNumber);

 for (index = 0; index < 26; index++)
   (TaskDispatchTable[index]) = NotUsed;

 TaskDispatchTable['L'-'A'] = AnalogInput;
 TaskDispatchTable['H'-'A'] = HighVoltage;
 TaskDispatchTable['V'-'A'] = ChangeHighVoltage;
 TaskDispatchTable['P'-'A'] = InitPIO;
 TaskDispatchTable['D'-'A'] = OutInPIO;
 TaskDispatchTable['G'-'A'] = SetGetGain;
 TaskDispatchTable['C'-'A'] = ControlLaw;
 TaskDispatchTable['A'-'A'] = DtoAOut;
 TaskDispatchTable['E'-'A'] = PosLoad;
 TaskDispatchTable['X'-'A'] = ExecuteMove;
 TaskDispatchTable['I'-'A'] = Diagnostic;
 TaskDispatchTable['M'-'A'] = MaxLimit;
 TaskDispatchTable['S'-'A'] = ShutDown;
 TaskDispatchTable['R'-'A'] = RevisionLevel;

 DI ();
 init_kernel ();
 run_every (TASK2, 100);
 run_every (TASK1, 8);  /* was 23  44.92 */ /* was 45 -or- 87.9 ms */
 /* run_every (TASK0, 5); */  /* 11.72 */ /* was 12 -or- 23.4 ms */
 init_timer0 (900);      /* 512 hz clock, 0.001953 seconds */
 EI ();
 backgnd ();
}

void NotUsed(void)
{
 return;
}

nodebug backgnd ()
{
 while (1)
  {
   if (Shutdown)
    {
        op_kill_z1();
        op_init_z1 (19200/1200, InString, UnitNumber);
        outport (0x81, ServoNull.byte.lsb);
        outport (0x82, ServoNull.byte.msb);
```

```
            while (1)
             {
              if (check_opto_command() == 1)
                replyOpto22 (PowerFail, strlen (PowerFail), 0);
              hitwd();
             }
     }

    if (check_opto_command() == 1)
     {
            strcpy (OutString, "!OK\0");

            InString[InString[1]] = '\0';

            if (isalpha (InString[4]))
             {
              (*TaskDispatchTable[toupper(InString[4]) - 'A'])
                (InString+2, OutString);
             }
            /*
             if (DumpSerial)
             {
             Dwrite_z0(InString+1, InString[0] - 2);
             Dwrite_z0(OutString, strlen (OutString));
             }
      */

            replyOpto22 (OutString, strlen (OutString), 0);
     }
  }
 while (1);
}


MaxLimit (char *In, char *Out)
{
 char chvalue[8];
 long value;

 switch (toupper(*(In+3)))
  {
   case 'V':  /* >1MV98765. */ /* velocity limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxVelo = atol (chvalue);
    sprintf (Out, "OK");
    break;
   case 'L':  /* >1ML12345. */ /* Extension limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxLimits = atol (chvalue);
    AbsLimits = MaxLimits + 200L;
    sprintf (Out, "OK");
    break;
   case 'S':  /* >1MS2. */  /* Actuator scaling */
    strncpy (chvalue, (In+4), 1);
    chvalue[1] = '\0';
    ActScale = atoi (chvalue);
    sprintf (Out, "OK");
    break;
   case 'N':  /* >1MN2047. */ /* set null point to 2047 */
    strncpy (chvalue, (In+4), 4);
    chvalue[4] = '\0';
    ServoNull.word = atoi(chvalue);
    if (ServoNull.word == -1)
          {
           ServoNull.byte.lsb = ee_rd (0x12);
           ServoNull.byte.msb = ee_rd (0x13);
```

```
            sprintf (Out, ">MN%4d.\0", ServoNull.word);
           }
     else
           {
            ee_wr (0x12, ServoNull.byte.lsb);
            ee_wr (0x13, ServoNull.byte.msb);
            sprintf (Out, "Servo null: %x %x", ee_rd (0x12), ee_rd(0x13));
            outport (0x81, ServoNull.byte.lsb);
            outport (0x82, ServoNull.byte.msb);
           }
     break;
    case 'U':   /* >1MU2. */ /* set unit number to 2 */
     strncpy (chvalue, (In+4), 1);
     chvalue[1] = '\0';
     ee_wr (0x10, atoi (chvalue));
     sprintf (Out, ">1MU%1d.\0", ee_rd (0x10));
     break;
  }
}

nodebug PosLoad (char *In, char *Out)
{
 char chvalue[8];
 char **pchvalue;
 long value;
 int value1;

 if (*(In+3) == 'S')
  {
   strncpy (chvalue, (In+4), 4);
   chvalue[4] = '\0';
   value1 = atoi (chvalue);

   if (value1 == -1)  /* >1ES__-1. */
          {
           sprintf (Out, ">ES%5ld%5ld.", gActPosition, gActVelocity);
           return;
          }
   else
          if (value1 == -2)  /* >1ES__-2. */
           {
            sprintf (Out, ">ES%5ld%5d.", gActPosition, Actuation.word);
            return;
           }
          else
           if ((value1 >= 0) && (value1 < PointNumber))
            {
             sprintf (Out, ">ES%5ld%5ld%3d.",
                        PointList[value1].ListPosition,
                        PointList[value1].ListVelocity,
                        PointList[value1].PIOmode);
             return;
            }
   sprintf (Out, "!ES ERROR");
   return;
  }

 if (*(In+3) == 'R')
  {                                /* replace pt 0000 */
   /* >1ER0000123456987654000. pos=123456,velo=987654,pio=000 */
   strncpy (chvalue, (In+4), 4);
   chvalue[4] = '\0';
   value1 = atoi (chvalue);

   strncpy (chvalue, (In+8), 6);
   chvalue[6] = '\0';
   value = atol (chvalue);

   if ((value > MaxLimits))
```

```
        PointList[value1].ListPosition = MaxLimits;
    else
        PointList[value1].ListPosition = value;

    stmcpy (chvalue, (In+14), 6);
    chvalue[6] = '\0';
    value = atol(chvalue);

    if ((value > MaxVelo) && (MaxVelo != 0L))
        PointList[value1].ListVelocity = MaxVelo;
    else
        if (value == 0)
         PointList[value1].ListVelocity = 200;
        else
         PointList[value1].ListVelocity = value;

    stmcpy (chvalue, (In+20), 3);
    chvalue[3] = 0;
    PointList[value1].PIOmode = atoi (chvalue);
    sprintf (Out, "OK");
    retum;
  }

if (*(In+3) == 'L')
 {        /* >1EL12345987600. pos=12345,velo=9876,pio=00, in hex */
   stmcpy (chvalue, (In+4), 5);
   chvalue[5]= '\0';
   value = atol (chvalue);

   if ((value > MaxLimits))
        PointList[PointNumber].ListPosition = MaxLimits;
   else
        PointList[PointNumber].ListPosition = value;

   stmcpy (chvalue, (In+9), 4);
   chvalue[4] = '\0';
   value = atol(chvalue);

   if ((value == 0) && (UseVelocityControl == OFF))
        PointList[PointNumber].ListVelocity = 200;
   else
        PointList[PointNumber].ListVelocity = value;

   stmcpy (chvalue, (In+13), 2);
   chvalue[2] = 0;
   PointList[PointNumber].PIOmode = (int)strtol (chvalue, pchvalue, 16);

   if (PointList[PointNumber].ListVelocity != 0)
        {
         if (PointNumber == 0)
          {
           PointList[PointNumber].TimeSlices =
                (PointList[PointNumber].ListPosition - gActPosition) * 64.0
                / PointList[PointNumber].ListVelocity;
          }
         else
          {
           PointList[PointNumber].TimeSlices =
                (PointList[PointNumber].ListPosition-
                 PointList[PointNumber-1].ListPosition) * 64.0
                / PointList[PointNumber].ListVelocity;
          }
        }
   else
        {
         PointList[PointNumber].TimeSlices = 0;
        }

   PointNumber++;
```

```
    sprintf (Out, "OK");
    return;
   }

  if (*(In+3) == 'C')
   {
    PointNumber = 0;
    ExePointNumber = 0;
    datapointer = 0;
    takedata = OFF;
    sprintf (Out, "OK");
    return;
   }

  if (*(In+3) == 'V')                    /* >1EV1. */
   {
    sprintf (Out, "OK");
    if (*(In+4) == '1')
           {
            UseVelocityControl = ON;
            /* outport (PIOCB, 0x87); */ /* enable interrupt driven */
           }
    else
           {
            UseVelocityControl = OFF;
            Reset1 = TRUE;
            /* outport (PIOCB, 0x7);  */ /* disable interrupt driven */
           }
    return;
   }
}

ExecuteMove (char *In, char *Out)
{
 datapointer = 0;
 takedata = ON;

 /*
   PIOstatus.PIOAmode =
   (PointList[ExePointNumber].PIOmode & 0xfe);  *//* lower status flag */
 /*  outport (PIODA, PIOstatus.PIOAmode);  */

/*
 if (PointList[0].ListVelocity != 0)
   PointList[0].TimeSlices =
     ((long)((float)(PointList[0].ListPosition - gActPos) /
               ((float)PointList[0].ListVelocity * 0.044919)));
NOTE: This is what i wanted, but a compiler error forced me into the below.
*/

 if (PointList[0].ListVelocity != 0)
   {
    PointList[0].TimeSlices =
           (PointList[0].ListPosition - gActPosition) * 64.0
           / PointList[0].ListVelocity;
   }

 ExePointNumber = 0;
 BeginMove = TRUE;
 NewMove = TRUE;
}

/* #INT_VEC PIOB_VEC intrExecuteMove */

/* interrupt reti int intrExecuteMove () */
/* { */
/*  takedata = ON; */
/*  datapointer = 0; */
```

```
/*  PIOstatus.PIOAmode =  */
/*    (PointList[ExePointNumber].PIOmode & 0xfe);  */
/*  outport (PIODA, PIOstatus.PIOAmode);  */

/*  ExePointNumber = 0; */
/*  BeginMove = TRUE; */
/*  UseVelocityControl = OFF; */
/* } */

Diagnostic (char *In, char *Out)
{
  static int indexer;

#GLOBAL_INIT
  {
   indexer = 0;
  }

  switch (*(In+3))
   {
    case '0':
          indexer = 0;
          if (BeginMove == FALSE)
            sprintf (Out, ">%4d.\0", datapointer);
          else
            sprintf (Out, "  -1\0");
          break;
    case '1':
          if (BeginMove == FALSE)
            {
             sprintf (Out, ">%u,%d,%ld,%ld.\0",
                      data[indexer].time[0], data[indexer].point,
                      data[indexer].position, data[indexer].velocity);
             indexer++;
            }
          break;
    case '2':
          if (DumpSerial == TRUE)
            DumpSerial = FALSE;
          else
            DumpSerial = TRUE;
          break;
    case '3':
          sprintf (Out, ">%d\0", PointNumber);
          break;
   }
}

ShutDown (char *In, char *Out)
{
  BYTEMODE index;

  DI();
  inport (0x82);            /* clear the encoder */
  outport (0x81, ServoNull.byte.lsb);       /* zero the d-to-a board */
  outport (0x82, ServoNull.byte.msb);
  outport (0x41, (char)0xf);  /* pioa command */
  outport (0x41, (char)0x0);  /* set all bits for output */
  outport (0x41, (char)0x7);  /* disable interrupts */
  outport (0x40, (char)0x0);  /* pioa data */
  outport (0x43, (char)0xf);  /* piob command */
  outport (0x43, (char)0x0);  /* set all bits for output */;
  outport (0x43, (char)0x7);  /* disable interrupts */
  outport (0x42, (char)0x0);  /* piob data */
  Shutdown = ON;
  UseVelocityControl = OFF;
  sprintf (Out, "OK");
}
```

```
RevisionLevel (char *In, char *Out)
{
 BYTEMODE index;

 Control = OFF;
 Shutdown = OFF;
 BeginMove = FALSE;
 NewMove = FALSE;
 UseVelocityControl = OFF;
 Reset1 = Reset2 = ON;
 gActVelocity = 0L;
 gActPosition = 0L;
 kpp = kpi = kpd = kff = 0;
 kvp = kvi = kvd = 0;
 PointNumber = ExePointNumber = 0;
 Actuation.word = 0;
 outport (0x81, ServoNull.byte.lsb);
 outport (0x82, ServoNull.byte.msb);
 inport (0x82);
 sprintf (Out, ">unit:%d Rev beta %d.%d.", UnitNumber, VERSION, SUBVERSION);
}

AnalogInput (char *In, char *Out)  /* >1L0. channel 0 */
{
 sprintf (Out, ">L%4d.", ad_rd8(atoi(In+3)));
}

HighVoltage (char *In, char *Out)   /* >1H1. on, >1H0. off */
{
 if (*(In+3) == '1')
   {
    hv_enb();
    sprintf (Out, "OK");
   }
 else
   {
    hv_dis();
    sprintf (Out, "OK");
   }
}

ChangeHighVoltage (char *In, char *Out)  /* >1V123.  port pattern 123 */
{
 char chvalue[4];

 strncpy (chvalue, (In+3), 3);
 chvalue[3] = 0;

 hv_wr(atoi(chvalue));
 sprintf (Out, "OK");
}

InitPIO (char *In, char *Out)  /* >1PA0103. mode 01, control 03, on PIO A */
{
 char InChar[3];
 char mode, control;

 strncpy (InChar, (In+4), 2);
 InChar[2] = 0;
 mode = atoi (InChar);

 strncpy (InChar, (In+6), 2);
 control = atoi (InChar);

 switch (*(In+3))
  {
   case 'A':
          outport (PIOCA, ((mode << 6) & 0xf0) | 0x0f);
          outport (PIOCA, (char)control);
```

```
            sprintf (Out, "OK");
            break;
      case 'B':
            outport (PIOCB, ((mode << 6) & 0xf0) | 0x0f);
            outport (PIOCB, (char)control);
            sprintf (Out, "OK");
            break;
    }
}

OutInPIO (char *In, char *Out)  /* >1DOB000. Output on PIO B value 000,
                                   >1DIA.  Input on PIO A returns 145 */
{
 char chvalue[4];
 int value;

 chvalue[3] = 0;
 switch (*(In+3))
  {
   case 'O':
         if (*(In+4) == 'A')
          {
           strncpy (chvalue, (In+5), 3);
           value = atoi(chvalue);
           PIOstatus.PIOAmode = (char)value | (PIOstatus.PIOAmode & 0x3);
           outport (PIODA, PIOstatus.PIOAmode);
           sprintf (Out, "OK");
          }
         else
          if (*(In+4) == 'B')
           {
            strncpy (chvalue, (In+5), 3);
            value = atoi (chvalue);
            PIOstatus.PIOBmode = (char)value;
            outport (PIODB, (char)value);
            sprintf (Out, "OK");
           }
         break;
   case 'I':
         if (*(In+4) == 'A')
          {
           value = inport (PIODA);
           sprintf (Out, ">DIA%3d.", value);
          }
         else
          if (*(In+4) == 'B')
           {
            value = inport (PIODB);
            sprintf (Out, ">DIB%3d.", value);
           }
          break;
  }
}

SetGetGain (char *In, char *out)  /* >1GPP1234., Pos. Proportional to 1234
                                     "  -1" to report */
{
 int value;
 char chvalue[5];

 strncpy (chvalue, (In+5), 4);
 chvalue[4] = '\0';

 value = atoi (chvalue);

 switch (*(In+3))
  {
   case 'V':
         switch (*(In+4))
```

```
              {
                case 'F':
                 if (value == -1)
                          {
                           sprintf (out, ">GVF%4d.", (int)kff*100);
                           return;
                          }
                 else
                          {
                           Reset1 = Reset2 = ON;
                           kff = (float)value/100.0;
                           sprintf (out, "OK");
                          }
                 break;
                case 'P':
                 if (value == -1)
                          {
                           sprintf (out, ">GVP%4d.", kvp);
                           return;
                          }
                 else
                          {
                           Reset1 = Reset2 = ON;
                           kvp = value;
                           sprintf (out, "OK");
                          }
                 break;
                case 'I':
                 if (value == -1)
                          {
                           sprintf (out, ">GVI%4d.", kvi);
                           return;
                          }
                 else
                          {
                           Reset1 = Reset2 = ON;
                           kvi = value;
                           sprintf (out, "OK");
                          }
                break;
               case 'D':
                if (value == -1)
                 {
                          sprintf (out, ">GVD%4d.", kvd);
                          return;
                 }
                else
                 {
                          Reset1 = Reset2 = ON;
                          kvd = value;
                          sprintf (out, "OK");
                 }
                break;
              }
             break;
case 'P':
        switch (*(In+4))
         {
          case 'P':
           if (value == -1)
                    {
                     sprintf (out, ">GPP%4d.", (int)kpp);
                     return;
                    }
           else
                    {
                     Reset1 = Reset2 = ON;
                     kpp = (float)value;
                     sprintf (out, "OK");
```

```
                    }
                break;
               case 'I':
                if (value == -1)
                    {
                     sprintf (out, ">GPI%4d.", (int)kpi);
                     return;
                    }
                else
                    {
                     Reset1 = Reset2 = ON;
                     kpi = (float)value;  /* WAS 10000 */
                     sprintf (out, "OK");
                    }
                break;
               case 'D':
                if (value == -1)
                    {
                     sprintf (out, ">GPD%4d.", (int)kpd);
                     return;
                    }
                else
                    {
                     Reset1 = Reset2 = ON;
                     kpd = (float)value;
                     sprintf (out, "OK");
                    }
                break;
             }
    }
}

ControlLaw (char *In, char *out)  /* >1C1. on, >1C0. off */
{
 if (*(In+3) == '0')
   {
    BeginMove = OFF;
    NewMove = FALSE;
    Control = OFF;
    takedata = OFF;
    UseVelocityControl = OFF;
    outport (0x81, ServoNull.byte.lsb);
    outport (0x82, ServoNull.byte.msb);
    sprintf (out, "OK");
   }
 else
   if (*(In+3) == '1')
    {
         Control = ON;
         Reset1 = Reset2 = ON;
         sprintf (out, "OK");
    }
}

DtoAOut (char *In, char *out)  /* >1A4321. 4321 on DtoA */
{
 char chvalue[5];
 BYTEMODE value;

 Control = OFF;

 strncpy (chvalue, (In+3), 4);
 chvalue[4] = '\0';

 value.word = atoi (chvalue);
 outport (0x81, value.byte.lsb);
 outport (0x82, value.byte.msb);
 sprintf (out, "OK");
}
```

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

Position loop!

This task generates the actuation using PID.

communicates with the velocity loop thru the shared variable PseudoVelo

=================================================================*/
nodebug task1()
{
 static long ActPos, ActVelocity;
 static float DeltaPErr0, DeltaPErr1, DeltaPErr2;
 static float ipActuation;
 static long DesVelo;
 static int TimePeriods;
 static float DesPos;
 static float DesPosIncrement;
 static int LastPoint;
 static long dResult, NewPos;
 static BYTEMODE RealActuation;
 static unsigned result, result0;
 static float dAbsPos[5];
 static long OldgActPosition;
 static int index, RollOver;
 static unsigned int lo0, lo1, hi0, hi1;

#GLOBAL_INIT
 {
  ActPos = DesPos = ActVelocity = 0L;
  DeltaPErr0 = DeltaPErr1 = DeltaPErr2 = 0;
  DesVelo = 0L;
  ipActuation = 0;
  TimePeriods = 0;
  DesPosIncrement = 0;
  LastPoint = FALSE;
  RollOver = 0;
  result = 0;
  gActPosition = 0;
  dAbsPos[0] = dAbsPos[1] = dAbsPos[2] = dAbsPos[3] = 0L;
  index = 0;
 }

 OldgActPosition = gActPosition;
 /* result = ((inport (0x80) << 8) & 0xff00) | inport (0x81); */

 do
  {
   DI();
   lo0 = inport (0x81);
   hi0 = inport (0x80);  /* stabilize results */
   lo1 = inport (0x81);
   hi1 = inport (0x80);
   EI();
  }
 while ((lo0 != lo1) || (hi0 != hi1));
 result = ((hi1 << 8) & 0xff00) | lo1;

 dResult= (long)((long)result - (long)result0);

 if (dResult > 32000)
  RollOver--;
 else
  if (-32000 > dResult)
   RollOver++;

 result0 = result;
 gActPosition = (long)result + (long)(RollOver * (long)65536);
```

```
dAbsPos[1] = dAbsPos[0];
dAbsPos[0] = gActPosition - OldgActPosition;
gActVelocity = (long)((dAbsPos[0] + dAbsPos[1]) * 32.0 );

if (gActPosition > AbsLimits)
 {
  outport (0x81, ServoNull.byte.lsb);
  outport (0x82, ServoNull.byte.msb);
  Control = OFF;
 }

if (Control == OFF)
 {
  PseudoVelo = 0;
  ipActuation = 0;
  return;
 }

ActPos = gActPosition;  /* transfer to local variables */
ActVelocity = gActVelocity;

if (Reset1)
 {
  DeltaPErr2 = DeltaPErr1 = DeltaPErr0 = 0;
  ipActuation = 0;
  DesPos = ActPos;
  NewPos = ActPos;
  DesVelo = 0;
  DesPosIncrement = 0;
  TimePeriods = 0;
  BeginMove = NewMove = FALSE;
  UseVelocityControl = OFF;
  Reset1 = OFF;
 }

if (BeginMove == TRUE)
 {
  LastPoint = FALSE;
  if (NewMove == TRUE)  /* allows for interrupted move */
        {
         NewMove = FALSE;
         TimePeriods = 0;
         if (UseVelocityControl == ON)
           DesVelo = PointList[0].ListVelocity;
        }
  if (((TimePeriods--) <= 1) && (UseVelocityControl == OFF))
        {
         if (ExePointNumber >= PointNumber)
           {
            ExePointNumber = PointNumber;
            NewPos = PointList[ExePointNumber-1].ListPosition;
            DesVelo = 0;
            DesPosIncrement = 0;
            BeginMove = FALSE;
           }
         else
           {
            if (PointList[ExePointNumber].TimeSlices != 0)
                  {
        NewPos = PointList[ExePointNumber].ListPosition;
                   DesPos = ActPos;
                   DesVelo = PointList[ExePointNumber].ListVelocity;
                   TimePeriods = (int)PointList[ExePointNumber].TimeSlices;

                   if (TimePeriods < 0)
                     {
                      TimePeriods = -TimePeriods;
                      DesVelo = -DesVelo;
```

```
                    }
                   DesPos = ActPos;
                  }
             else
                  {
                   DesPosIncrement = 0;
                   DesPos = PointList[ExePointNumber].ListPosition;
                   DesVelo = 0;
                  }
             }
           ExePointNumber++;
           if (ExePointNumber == PointNumber) LastPoint = TRUE;
          }
   if (TimePeriods > 0)
          {
           DesPos += (float)(NewPos - ActPos) / (TimePeriods);
          }
   if ((LastPoint == TRUE) && (TimePeriods < 5)) DesVelo = 0;
  }
 else
  {
   DesPosIncrement = 0;
   DesPos = NewPos;
  }

 DeltaPErr2 = DeltaPErr1; /* generate time history - 2 steps back */
 DeltaPErr1 = DeltaPErr0;
 DeltaPErr0 = DesPos - ActPos;

 ipActuation +=
  (((float)kpp * (DeltaPErr0 - DeltaPErr1)) +
   ((float)kpi * (DeltaPErr0)) +
   ((float)kpd * (DeltaPErr0 - (DeltaPErr1+DeltaPErr1) + DeltaPErr2)) +
   (kff * (DesVelo - ActVelocity)));

 if (UseVelocityControl == ON)
  PseudoVelo = DesVelo;
 else
  {
   PseudoVelo = ipActuation;

   if (PseudoVelo > (long)65504)
          PseudoVelo = 65504;
   else
          if (PseudoVelo < -65504)
           PseudoVelo = -65504;

   RealActuation.word = (int)(PseudoVelo >> 5);

   Actuation.word = RealActuation.word;

   RealActuation.word += D2AOFFSET;
   DI();
   k_lock();
   outport (0x81, (char)(RealActuation.byte.lsb));
   outport (0x82, (char)(RealActuation.byte.msb));
   k_unlock();
   EI();
  }
}

/*================================================================
This task takes sensor data.
================================================================*/

nodebug task2()
{
 static int index;
```

```
#GLOBAL_INIT
 {
  index = 0;
 }

 if (Shutdown == TRUE) return;
 hitwd();

 if (takedata == ON)
  {
   if (datapointer == 0) index = 0;

   gettimer (data[datapointer].time);
   data[datapointer].position = gActPosition;
   data[datapointer].velocity = gActVelocity;
   data[datapointer].point = ExePointNumber;

   if (BeginMove == FALSE)
         {
          if (index > 5)
           takedata = OFF;
          else
           index++;
         }

   if (datapointer++ > 395) takedata = OFF;
  }

 if ((PIOstatus.PIOAmode & 0x2) == 0x0)
  PIOstatus.PIOAmode |= 0x2;
 else
  PIOstatus.PIOAmode &= 0xfd;

 outport (PIODA, PIOstatus.PIOAmode);
}

/*==================================================================

Velocity loop!

This task generates the velocity actuation and also takes the sensor
data from the encoder.

uses the shared variables PseudoVelo for input,
and gActPosition, gActVelocity for output.

==================================================================*/

nodebug task0()
{
 static long pVelo, dResult;
 static long DeltaVErr0, DeltaVErr1, DeltaVErr2;
 static BYTEMODE RealActuation;
 static long vActuation;
 static unsigned int result0;
 static int result;
 static float dAbsPos[5];
 static long OldgActPosition;
 static int index, RollOver;
 static unsigned int lo0, lo1, hi0, hi1;

#GLOBAL_INIT
 {
  RollOver = 0;
  result = result0 = 0;
  gActPosition = 0;
  dAbsPos[0] = dAbsPos[1] = dAbsPos[2] = dAbsPos[3] = 0L;
  DeltaVErr0 = DeltaVErr1 = DeltaVErr2 = 0L;
  index = 0;
```

```
}

OldgActPosition = gActPosition;
/* result = ((inport (0x80) << 8) & 0xff00) | inport (0x81); */

do
 {
   DI();
   lo0 = inport (0x81);
   hi0 = inport (0x80);   /* stabilize results */
   lo1 = inport (0x81);
   hi1 = inport (0x80);
   EI();
 }
while ((lo0 != lo1) || (hi0 != hi1));
result = ((hi1 << 8) & 0xff00) | lo1;

gActPosition = 5588.0 *  p2sin (16.1125 + (0.00409091 * (float)result));

dAbsPos[1] = dAbsPos[0];
dAbsPos[0] = gActPosition - OldgActPosition;
gActVelocity = (long)((dAbsPos[0] + dAbsPos[1]) * 42.6667);

pVelo = PseudoVelo;  /* transfer variables to local storage */

if (Reset2)
 {
   DeltaVErr2 = DeltaVErr1 = DeltaVErr0 = 0L;
   vActuation = 0L;
   Reset2 = OFF;
 }

if (gActPosition > AbsLimits)
 {
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   Control = OFF;
 }

if (Control == OFF)
 {
   vActuation = 0;
   return;
 }

/*  DeltaVErr2 = DeltaVErr1; */
/*  DeltaVErr1 = DeltaVErr0; */
/*  DeltaVErr0 = pVelo - gActVelocity; */

/*  vActuation += (long) */
/*    ((((long)kvp*(long)(DeltaVErr0-DeltaVErr1)) + */
/*      ((long)kvi*(long)(DeltaVErr0)) + */
/*      ((long)kvd*(long)(DeltaVErr0 - (DeltaVErr1 + DeltaVErr1) + */
/*                                    DeltaVErr2)))); */

/*  if (vActuation > (long)268304384) */
/*    vActuation = 268304384; */
/*  else */
/*    if (vActuation < -268304384) */
/*      vActuation = -268304384; */

/*  RealActuation.word = ((int)(vActuation >> 17)); */

 if (pVelo > (long)2096128)
   pVelo = 2096128;
 else
   if (pVelo < -2096128)
     pVelo = -2096128;
```

```
  RealActuation.word = (int)(pVelo >> 10);

  Actuation.word = RealActuation.word;

  RealActuation.word += D2AOFFSET;
  DI();
  k_lock();
  outport (0x81, (char)(RealActuation.byte.lsb));
  outport (0x82, (char)(RealActuation.byte.msb));
  k_unlock();
  EI();
}

#JUMP_VEC NMI_VEC NMI_int

interrupt retn NMI_int()
{
 Shutdown = TRUE;
 Control = OFF;
 outport (0x81, ServoNull.byte.lsb);
 outport (0x82, ServoNull.byte.msb);
 while (1)
  {
   hitwd();
   if (!powerlo()) return;
  }
}

void FatalErrorHandler (unsigned code, unsigned address)
{
 outport (0x81, ServoNull.byte.lsb);
 outport (0x82, ServoNull.byte.msb);
 Shutdown = TRUE;
 Control = OFF;
 while (1);  /* stall until reset by watch dog */
}

float p2cos (float x)
{
 return (p2sin (90.0 - x));
}

/*==============================================================
PROCEDURE: p2sin

PARAMETERS: float Y

RETURNS: float

METHOD: computes sin(Y) by:
     sin (y + dy) = (sin y)(cos dy) + ([(cos x)/57.2958])(dy)
     where Y = y + dy, y = int(Y), dy = frac(Y), and
     the parameters are found from a lookup table. result is good
     to about 5 places.

VARIABLES:
     sinx are the values of sin(y) where y varies between 0->90
     cosdx are the values of cos(dy) where dy varies between
            0->1 in 0.01 increments
     cosxd are the values of [(cos y)/57.2958] where y varies
            between 0->90
     ALL fractional values have been shifted by multiplying by
            65535 to obtain integers.
================================================================*/

nodebug float p2sin (float y)

{
 static unsigned int sinx[] =
```

```
        {0x0000,0x0478,0x08EF,0x0D66,0x11DB,
         0x1650,0x1AC2,0x1F33,0x23A1,0x280C,
         0x2C74,0x30D9,0x3539,0x3996,0x3DEE,
         0x4242,0x4690,0x4AD9,0x4F1B,0x5358,
         0x578E,0x5BBE,0x5FE6,0x6407,0x681F,
         0x6C30,0x7039,0x7438,0x782F,0x7C1C,
         0x8000,0x83D9,0x87A8,0x8B6D,0x8F27,
         0x92D5,0x9679,0x9A10,0x9D9B,0xA11B,
         0xA48D,0xA7F3,0xAB4B,0xAE97,0xB1D4,
         0xB504,0xB826,0xBB39,0xBE3E,0xC134,
         0xC41B,0xC6F2,0xC9BA,0xCC73,0xCF1B,
         0xD1B3,0xD43B,0xD6B2,0xD919,0xDB6E,
         0xDDB3,0xDFE6,0xE208,0xE418,0xE616,
         0xE803,0xE9DD,0xEBA5,0xED5B,0xEEFE,
         0xF08F,0xF20D,0xF377,0xF4CF,0xF614,
         0xF746,0xF864,0xF96F,0xFA67,0xFB4B,
         0xFC1B,0xFCD8,0xFD81,0xFE17,0xFE98,
         0xFF06,0xFF5F,0xFFA5,0xFFD7,0xFFF5,
         0xFFFF};

static unsigned int cosdx[] =
        {0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
         0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
         0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
         0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
         0xFFFF,0xFFFF,0xFFFF,0xFFFE,0xFFFE,
         0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFE,
         0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFE,
         0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFD,
         0xFFFD,0xFFFD,0xFFFD,0xFFFD,0xFFFD,
         0xFFFD,0xFFFD,0xFFFD,0xFFFD,0xFFFD,
         0xFFFD,0xFFFC,0xFFFC,0xFFFC,0xFFFC,
         0xFFFC,0xFFFC,0xFFFC,0xFFFC,0xFFFC,
         0xFFFB,0xFFFB,0xFFFB,0xFFFB,0xFFFB,
         0xFFFB,0xFFFB,0xFFFB,0xFFFA,0xFFFA,
         0xFFFA,0xFFFA,0xFFFA,0xFFFA,0xFFFA,
         0xFFF9,0xFFF9,0xFFF9,0xFFF9,0xFFF9,
         0xFFF9,0xFFF8,0xFFF8,0xFFF8,0xFFF8,
         0xFFF8,0xFFF8,0xFFF7,0xFFF7,0xFFF7,
         0xFFF7,0xFFF7,0xFFF7,0xFFF6,0xFFF6,
         0xFFF6,0xFFF6,0xFFF6,0xFFF5,0xFFF5,
         0xFFF5};

static unsigned int cosxd[] =
        {0x478,0x478,0x477,0x476,0x475,0x473,0x472,
         0x46F,0x46D,0x46A,0x466,0x463,0x45F,0x45A,
         0x456,0x451,0x44B,0x446,0x440,0x439,0x433,
         0x42C,0x425,0x41D,0x415,0x40D,0x404,0x3FB,
         0x3F2,0x3E8,0x3DF,0x3D4,0x3CA,0x3BF,0x3B4,
         0x3A9,0x39D,0x391,0x385,0x379,0x36C,0x35F,
         0x352,0x345,0x337,0x329,0x31B,0x30C,0x2FD,
         0x2EE,0x2DF,0x2D0,0x2C0,0x2B0,0x2A0,0x290,
         0x280,0x26F,0x25E,0x24D,0x23C,0x22B,0x219,
         0x207,0x1F5,0x1E3,0x1D1,0x1BF,0x1AC,0x19A,
         0x187,0x174,0x161,0x14E,0x13B,0x128,0x115,
         0x101,0x0EE,0x0DA,0x0C7,0x0B3,0x09F,0x08B,
         0x078,0x064,0x050,0x03C,0x028,0x014,0x000};

static float fremainder, x;
static unsigned int whole;

x = y;
if (x < 0)
 {
  do
   {
        x = x + 360;
   }
  while (x < 0);
```

```
  }

 if (x <= 90.0)
  {
   whole = (int)x;
   fremainder = (x - whole);

   return ((float)((((unsigned long)sinx[whole]  *
                            (unsigned)cosdx[(int)(fremainder * 100.0)]) +
                           ((unsigned long)cosxd[whole] *
                            (unsigned)(fremainder * 65535)))
                          * 2.328377E-10));
  }
 else
  if (x <= 180)
   {
    x = 180.0 - x;
    whole = (int)x;
    fremainder = (x - whole);

    return ((float)((((unsigned long)sinx[whole]  *
                                  (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                                 ((unsigned long)cosxd[whole] *
                                  (unsigned long)(fremainder * 65535)))
                          * 2.328377E-10));
   }
 else
  if (x <= 270)
   {
    x = x - 180.0;
    whole = (int)x;
    fremainder = (x - whole);

    return ((float)((((unsigned long)sinx[whole]  *
                                  (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                                 ((unsigned long)cosxd[whole] *
                                  (unsigned long)(fremainder * 65535)))
                          * (-2.328377E-10)));
   }
 else
  if (x <= 360)
   {
    x = 360.0 - x ;
    whole = (int)x;
    fremainder = (x - whole);

    return ((float)((((unsigned long)sinx[whole]  *
                                  (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                                 ((unsigned long)cosxd[whole] *
                                  (unsigned long)(fremainder * 65535)))
                          * (-2.328377E-10)));
   }
 return (-2);
}
```

Appendix F

Source Code Listing for Rotation Joint SBC

```
#use rtk.lib

/*
This program would run on the Little Giant, the Tiny Giant and the CPLC.
If port 0 is also used as the Dynamic C programming port, you have to
load the serial interrupt routine during run time by doing the following:

  (1) comment out:
#INT_VEC SER0_VEC Dz0_circ_int in z0232.lib
  (2) in the code, declare:
        extern void Dz0_circ_int();
  (3) load the routine with:
        reload_vec(14,Dz0_circ_int);

*/

/*=====
For use on unit 3, rotation joint

Changes:
5/15/97 v5.01 Changed PIOmode to single char only.
7/15/97 v6.00 changed to burst serial reception. 8 at a time!
=====*/

#define SIMULATION 0

#define ON 1
#define OFF 0
#define FWD 1
#define REV 0
#define FALSE 0
#define TRUE 1

#define VERSION 6
#define SUBVERSION 00

#define CSAMPLE 900      /* clock periods: 512HZ */
#define CTIME 0.001953   /* clock time */

#define D2AOFFSET 2047
#define ACTSCALE 1

/* int task0(), task1(), task2(), backgnd();
int (*Ftask[])()={task0, task1, task2, backgnd}; */

int task1(), task2(), backgnd();
int (*Ftask[])() = {task1, task2, backgnd};

/*
#define NTASKS 4
#define TASK0 0
#define TASK1 1
#define TASK2 2 */

#define NTASKS 3
#define TASK1 0
#define TASK2 1

#define TASKSTORE_SIZE 500

void (*TaskDispatchTable[30])();

typedef union bytemode
 {
  int word;
  struct ByteStruct
   {
        char lsb;
        char msb;
```

```
    } byte;
  } BYTEMODE;

struct PointListPoint
 {
  int toY, toX;
  int Velocity;
  int TimeSlices;
  char PIOmode;
 };

struct PIOstruct
 {
  char PIOAmode;
  char PIOBmode;
 };

struct PointListPoint PointList[1001];
shared struct PIOstruct PIOstatus;

struct PositionData
 {
  int time[3];
  long position;   /* variables for data logging */
  long velocity;
  int point;
 };

struct PositionData data[410];

int datapointer, takedata;

int PointNumber, ExePointNumber;      /* place in point list */
shared long gActPosition, gActVelocity;  /* actual position and velocity */
shared long PseudoVelo;               /* IPC variable b/w pos and velo loop */
shared BYTEMODE Actuation;            /* system actuation */
float kpp, kpi, kpd;          /* system position gains 1 unit, I in .01 units */
int kvp, kvi, kvd;            /* system velocity gains 0.0001 units */
float kff;                    /* system feed forward gain 0.01 units */
int BeginMove;                /* change to TRUE to begin motion */
int NewMove;
int UseVelocityControl;       /* bypasses position controller */
int Acceleration;
int BegTime[3], EndTime[3];

BYTEMODE ServoNull;
shared long MaxVelo, MaxLimits, AbsLimits;
shared int ActScale;
int Control, Shutdown, Reset1, Reset2;
int UnitNumber, DumpSerial;
char InString[255], OutString[255];
char PowerFail[] = "!Power fail\0";
char SoftReset[] = "!Soft reset\0";
char tbuf[384], rbuf[384];

int AnalogInput (char *, char *);
int HighVoltage (char *, char *);
int ChangeHighVoltage (char *, char *);
int InitPIO (char *, char *);
int OutInPIO (char *, char *);
int SetGetGain (char *, char*);
int ControlLaw (char *, char *);
int DtoAOut (char *, char *);
int PosLoad (char *, char *);
int ExecuteMove (char *, char *);
int Diagnostic (char *, char *);
int ShutDown (char *, char *);
int RevisionLevel (char *, char *);
int MaxLimit (char *, char *);
```

```
void FatalErrorHandler (unsigned, unsigned);
float p2sin(float);
float p2cos(float);
void NotUsed(void);
long plant (int, int);

/* extern void Dz0_circ_int(); */

/* 0x10 stores network node number, 0x12 stores null.lsb, 0x13 stores
 null.msb */

root main ()
{
 int index;

 if (wderror())
  {
   Shutdown = TRUE;
   Control = OFF;
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   op_kill_z1();
   op_init_z1 (19200/1200, InString, UnitNumber);
   while (1)
         {
          hitwd();
          if (check_opto_command() == 1)
            replyOpto22 (SoftReset, strlen (SoftReset), 0);
         }
  }

 _GLOBAL_INIT();

 ERROR_EXIT = FatalErrorHandler;
 BeginMove = FALSE;
 UseVelocityControl = OFF;
 Control = DumpSerial = OFF;
 Shutdown = takedata = OFF;
 NewMove = FALSE;
 Reset1 = Reset2 = ON;
 ExePointNumber = PointNumber = 0;
 datapointer = 0;
 gActVelocity = gActPosition = 0L;
 kpp = kpi = kpd = kff = 0;
 kvp = kvi = kvd = 0;
 MaxLimits = AbsLimits = 88000L;
 Acceleration = 5000;
 MaxVelo = 0L;
 ActScale = ACTSCALE;
 Actuation.word = 0;    /* zero! */
 PIOstatus.PIOAmode = 0x1;
 PIOstatus.PIOBmode = (char) 0;
 UnitNumber = ee_rd(0x10);
 ServoNull.byte.lsb = ee_rd(0x12);
 ServoNull.byte.msb = ee_rd(0x13);

#if (SIMULATION == 0)
 inport (0x82);                  /* clear the encoder */
#else
 plant (2,0);
#endif

 outport (0x81, ServoNull.byte.lsb);  /* zero the d-to-a board */
 outport (0x82, ServoNull.byte.msb);
 outport (PIOCA, 0xff);              /* pioa command */
 outport (PIOCA, 0x00);              /* set all bits for output */
 outport (0x41, (char)0x7);          /* disable interrupts */
 outport (PIODA, PIOstatus.PIOAmode); /* pioa data */
```

```
  outport (PIOCB, 0xcf);              /* piob command */
  outport (PIOCB, 0x00);              /* set all bits for input */
  outport (PIOCB, PIOB_VEC);
  outport (PIOCB, 0x17);
  outport (PIOCB, 0xfe);
  outport (0x43, 0x7);                /* disable interrupts */
  /* outport (PIODB, PIOstatus.PIOBmode); */ /* piob data */
  outport (PIOCB, 0x7);

/*
 reload_vec (14, Dz0_circ_int);
 Dinit_z0(rbuf, tbuf, 384, 384, 4, 9600/1200, 0, 0);
*/

  op_init_z1 (19200/1200, InString, UnitNumber);

  for (index = 0; index < 26; index++)
    (TaskDispatchTable[index]) = NotUsed;

  TaskDispatchTable['L'-'A'] = AnalogInput;
  TaskDispatchTable['H'-'A'] = HighVoltage;
  TaskDispatchTable['V'-'A'] = ChangeHighVoltage;
  TaskDispatchTable['P'-'A'] = InitPIO;
  TaskDispatchTable['D'-'A'] = OutInPIO;
  TaskDispatchTable['G'-'A'] = SetGetGain;
  TaskDispatchTable['C'-'A'] = ControlLaw;
  TaskDispatchTable['A'-'A'] = DtoAOut;
  TaskDispatchTable['E'-'A'] = PosLoad;
  TaskDispatchTable['X'-'A'] = ExecuteMove;
  TaskDispatchTable['I'-'A'] = Diagnostic;
  TaskDispatchTable['M'-'A'] = MaxLimit;
  TaskDispatchTable['S'-'A'] = ShutDown;
  TaskDispatchTable['R'-'A'] = RevisionLevel;

  DI ();
  init_kernel ();
  run_every (TASK2, 100);
#if (SIMULATION == 0)
  run_every (TASK1, 8);
#endif
  init_timer0 (900);
  EI ();

  backgnd ();
}

void NotUsed(void)
{
  return;
}

nodebug backgnd ()
{
 while (1)
  {
#if (SIMULATION == 1)
  task1();
#endif

    if (Shutdown)
     {
          op_kill_z1();
          op_init_z1 (19200/1200, InString, UnitNumber);
          outport (0x81, ServoNull.byte.lsb);
          outport (0x82, ServoNull.byte.msb);

          while (1)
           {
            if (check_opto_command() == 1)
```

```
                replyOpto22 (PowerFail, strlen (PowerFail), 0);
               hitwd();
              }
      }

     if (check_opto_command() == 1)
      {
            strcpy (OutString, "!OK\0");

            InString[InString[1]] = '\0';

            if (isalpha (InString[4]))
             {
              (*TaskDispatchTable[toupper(InString[4]) - 'A'])
                (InString+2, OutString);
             }
            /*
             if (DumpSerial)
             {
             Dwrite_z0(InString+1, InString[0] - 2);
             Dwrite_z0(OutString, strlen (OutString));
             }
       */

            replyOpto22 (OutString, strlen (OutString), 0);
      }
   }
 while (1);
}


MaxLimit (char *In, char *Out)
{
 char chvalue[8];
 long value;

 switch (toupper(*(In+3)))
  {
   case 'V': /* >1MV98765. */ /* velocity limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxVelo = atol (chvalue);
    sprintf (Out, "OK");
    break;
   case 'L': /* >1ML12345. */ /* Extension limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxLimits = atol (chvalue);
    AbsLimits = MaxLimits + 200L;
    sprintf (Out, "OK");
    break;
   case 'S': /* >1MS2. */ /* Actuator scaling */
    strncpy (chvalue, (In+4), 1);
    chvalue[1] = '\0';
    ActScale = atoi (chvalue);
    sprintf (Out, "OK");
    break;
   case 'N': /* >1MN2047. */ /* set null point to 2047 */
    strncpy (chvalue, (In+4), 4);
    chvalue[4] = '\0';
    ServoNull.word = atoi(chvalue);
    if (ServoNull.word == -1)
           {
            ServoNull.byte.lsb = ee_rd (0x12);
            ServoNull.byte.msb = ee_rd (0x13);
            sprintf (Out, ">MN%4d.\0", ServoNull.word);
           }
    else
           {
```

```
            ee_wr (0x12, ServoNull.byte.lsb);
            ee_wr (0x13, ServoNull.byte.msb);
            sprintf (Out, "Servo null: %x %x", ee_rd (0x12), ee_rd(0x13));
            outport (0x81, ServoNull.byte.lsb);
            outport (0x82, ServoNull.byte.msb);
          }
      break;
    case 'U':   /* >1MU2. */ /* set unit number to 2 */
      strncpy (chvalue, (In+4), 1);
      chvalue[1] = '\0';
      ee_wr (0x10, atoi (chvalue));
      sprintf (Out, ">1MU%1d.\0", ee_rd (0x10));
      break;
  }
}

PosLoad (char *In, char *Out)
{
 char chvalue[8];
 char **pchvalue;
 long value;
 int value1, index, index2, offset;
 float CycTime;
 float TempFloat1, TempFloat2;

 if (*(In+3) == 'S')
  {
   strncpy (chvalue, (In+4), 4);
   chvalue[4] = '\0';
   value1 = atoi (chvalue);

   if (value1 == -1)  /* >1ES__-1. */
         {
          sprintf (Out, ">ES%5ld%5ld.", gActPosition, gActVelocity);
          return;
         }
   else
         if (value1 == -2)  /* >1ES__-2. */
          {
           sprintf (Out, ">ES%5ld%5d.", gActPosition, Actuation.word);
           return;
          }
         else
          if ((value1 >= 0) && (value1 < PointNumber))
           {
            sprintf (Out, ">ES%5d%5d%4d%4d%1d.",
                        PointList[value1].toX,
                        PointList[value1].toY,
                        PointList[value1].Velocity,
                        PointList[value1].TimeSlices,
                        PointList[value1].PIOmode);
            return;
           }
   sprintf (Out, "!ES ERROR");
   return;
  }

 if (*(In+3) == 'R')
  {                              /* replace pt 0000 */
   /* >1ER0000123456987654000. pos=123456,velo=987654,pio=000 */
   sprintf (Out, "OK");
   return;
  }

 if (*(In+3) == 'L')
  {        /* >1EL1234512345987600. pos=12345,velo=9876,pio=00, in hex */
   strncpy (chvalue, (In+4), 1);
   chvalue[1]= '\0';
   index = atoi (chvalue);
```

```
for (index2 = 0; index2 < index;)
        {
         offset = index2 * 15;
         strncpy (chvalue, In+(offset+5), 5);
         chvalue[5] = '\0';
         value = atoi (chvalue);

         if ((value > MaxLimits))
           PointList[PointNumber+index2].toX = MaxLimits;
         else
           PointList[PointNumber+index2].toX = (int)value;

         strncpy (chvalue, In + (offset+10), 5);
         chvalue[5] = '\0';
         value = atoi (chvalue);
         if (value > MaxLimits)
           PointList[PointNumber+index2].toY = MaxLimits;
         else
           PointList[PointNumber+index2].toY = (int)value;

         strncpy (chvalue, In + (offset + 15), 4);
         chvalue[4] = '\0';
         value = atoi(chvalue);

         if ((value == 0) && (UseVelocityControl == OFF))
           PointList[PointNumber+index2].Velocity = 200;
         else
           PointList[PointNumber+index2].Velocity = (int)value;

         PointList[PointNumber+index2].PIOmode = atoi (In+(offset+18));

         if (PointList[PointNumber+index2].Velocity != 0)
           {
      if (PointNumber == 0)
        {
         PointList[PointNumber].TimeSlices = 0;
        }
      else
        {
#if (SIMULATION == 1)
                    printf ("PosLoad Velocity: %d\n",
                            PointList[PointNumber].Velocity);
#endif
                    TempFloat1 = (float)(PointList[PointNumber+index2].toY -
                                         PointList[PointNumber-1+index2].toY);
                    TempFloat2 = (float)(PointList[PointNumber+index2].toX -
                                         PointList[PointNumber-1+index2].toX);
               CycTime =
                     sqrt((TempFloat1 * TempFloat1)+(TempFloat2 * TempFloat2))
                     / (float)PointList[PointNumber+index2].Velocity;
                    PointList[PointNumber+index2].TimeSlices = CycTime * 64;

#if (SIMULATION == 1)
          printf ("TimeSlices %d\n", PointList[PointNumber].TimeSlices);
#endif

                  }
           }
          else
           {
            PointList[PointNumber+index2].TimeSlices = 0;
           }
          index2++;
        }
    PointNumber += index2;
    sprintf (Out, "OK");
    return;
```

```
    }

  if (*(In+3) == 'C')
   {
    PointNumber = 0;
    ExePointNumber = 0;
    datapointer = 0;
    takedata = OFF;
    sprintf (Out, "OK");
    return;
   }

  if (*(In+3) == 'V')                    /* >1EV1. */
   {
    sprintf (Out, "OK");
    if (*(In+4) == '1')
          {
           UseVelocityControl = ON;
          /* outport (PIOCB, 0x87); */ /* enable interrupt driven */
          }
    else
          {
           UseVelocityControl = OFF;
           Reset1 = TRUE;
          /* outport (PIOCB, 0x7);  */ /* disable interrupt driven */
          }
    return;
   }
}

ExecuteMove (char *In, char *Out)
{
 datapointer = 0;
 takedata = ON;
 ExePointNumber = 1;
 BeginMove = TRUE;
 NewMove = TRUE;
}

/* #INT_VEC PIOB_VEC intrExecuteMove */

/* interrupt reti int intrExecuteMove () */
/* { */
/*   takedata = ON; */
/*   datapointer = 0; */

/*   PIOstatus.PIOAmode =  */
/*     (PointList[ExePointNumber].PIOmode & 0xfe); */
/*   outport (PIODA, PIOstatus.PIOAmode); */

/*   ExePointNumber = 0; */
/*   BeginMove = TRUE; */
/*   UseVelocityControl = OFF; */
/* } */

Diagnostic (char *In, char *Out)
{
 static int indexer;

#GLOBAL_INIT
 {
  indexer = 0;
 }

 switch (*(In+3))
  {
   case '0':
          indexer = 0;
          if (BeginMove == FALSE)
```

```
            sprintf (Out, ">%4d.\0", datapointer);
           else
            sprintf (Out, "  -1\0");
           break;
    case '1':
           if (BeginMove == FALSE)
            {
             sprintf (Out, ">%u,%d,%ld,%ld.\0",
                      data[indexer].time[0], data[indexer].point,
                      data[indexer].position, data[indexer].velocity);
             indexer++;
            }
           break;
    case '2':
           if (DumpSerial == TRUE)
            DumpSerial = FALSE;
           else
            DumpSerial = TRUE;
           break;
    case '3':
           sprintf (Out, ">%d\0", PointNumber);
           break;
  }
}

ShutDown (char *In, char *Out)
{
 BYTEMODE index;

#if (SIMULATION == 1)
 plant (2,0);
#endif

 DI();
 inport (0x82);           /* clear the encoder */
 outport (0x81, ServoNull.byte.lsb);      /* zero the d-to-a board */
 outport (0x82, ServoNull.byte.msb);
 outport (0x41, (char)0xf);  /* pioa command */
 outport (0x41, (char)0x0);  /* set all bits for output */
 outport (0x41, (char)0x7);  /* disable interrupts */
 outport (0x40, (char)0x0);  /* pioa data */
 outport (0x43, (char)0xf);  /* piob command */
 outport (0x43, (char)0x0);  /* set all bits for output */;
 outport (0x43, (char)0x7);  /* disable interrupts */
 outport (0x42, (char)0x0);  /* piob data */
 Shutdown = ON;
 UseVelocityControl = OFF;
 sprintf (Out, "OK");
}

RevisionLevel (char *In, char *Out)
{
 BYTEMODE index;

 Control = OFF;
 Shutdown = OFF;
 BeginMove = FALSE;
 NewMove = FALSE;
 UseVelocityControl = OFF;
 Reset1 = Reset2 = ON;
 gActVelocity = 0L;
 gActPosition = 0L;
 kpp = kpi = kpd = kff = 0;
 kvp = kvi = kvd = 0;
 PointNumber = ExePointNumber = 0;
 Actuation.word = 0;
 outport (0x81, ServoNull.byte.lsb);
 outport (0x82, ServoNull.byte.msb);
 inport (0x82);
```

```
 sprintf (Out, ">unit:%d Rev beta %d.%d.", UnitNumber, VERSION, SUBVERSION);
#if (SIMULATION == 1)
 plant (2,0);
#endif
}

AnalogInput (char *In, char *Out) /* >1L0. channel 0 */
{
 sprintf (Out, ">L%4d.", ad_rd8(atoi(In+3)));
}

HighVoltage (char *In, char *Out)   /* >1H1. on, >1H0. off */
{
 if (*(In+3) == '1')
  {
   hv_enb();
   sprintf (Out, "OK");
  }
 else
  {
   hv_dis();
   sprintf (Out, "OK");
  }
}

ChangeHighVoltage (char *In, char *Out)  /* >1V123.  port pattern 123 */
{
 char chvalue[4];

 strncpy (chvalue, (In+3), 3);
 chvalue[3] = 0;

 hv_wr(atoi(chvalue));
 sprintf (Out, "OK");
}

InitPIO (char *In, char *Out) /* >1PA0103. mode 01, control 03, on PIO A */
{
 char InChar[3];
 char mode, control;

 strncpy (InChar, (In+4), 2);
 InChar[2] = 0;
 mode = atoi (InChar);

 strncpy (InChar, (In+6), 2);
 control = atoi (InChar);

 switch (*(In+3))
  {
   case 'A':
          outport (PIOCA, ((mode << 6) & 0xf0) | 0x0f);
          outport (PIOCA, (char)control);
          sprintf (Out, "OK");
          break;
   case 'B':
          outport (PIOCB, ((mode << 6) & 0xf0) | 0x0f);
          outport (PIOCB, (char)control);
          sprintf (Out, "OK");
          break;
  }
}

OutInPIO (char *In, char *Out) /* >1DOB000. Output on PIO B value 000,
                                  >1DIA.  Input on PIO A returns 145 */
{
 char chvalue[4];
 int value;
```

```
   chvalue[3] = 0;
   switch (*(In+3))
    {
     case 'O':
             if (*(In+4) == 'A')
              {
               strncpy (chvalue, (In+5), 3);
               value = atoi(chvalue);
               PIOstatus.PIOAmode = (char)value | (PIOstatus.PIOAmode & 0x3);
               outport (PIODA, PIOstatus.PIOAmode);
               sprintf (Out, "OK");
              }
             else
              if (*(In+4) == 'B')
               {
                strncpy (chvalue, (In+5), 3);
                value = atoi (chvalue);
                PIOstatus.PIOBmode = (char)value;
                outport (PIODB, (char)value);
                sprintf (Out, "OK");
               }
             break;
     case 'I':
             if (*(In+4) == 'A')
              {
               value = inport (PIODA);
               sprintf (Out, ">DIA%3d.", value);
              }
             else
              if (*(In+4) == 'B')
               {
                value = inport (PIODB);
                sprintf (Out, ">DIB%3d.", value);
               }
              break;
    }
 }

 SetGetGain (char *In, char *out)  /* >1GPP1234., Pos. Proportional to 1234
                                              "  -1" to report */
 {
  int value;
  char chvalue[5];

  strncpy (chvalue, (In+5), 4);
  chvalue[4] = '\0';

  value = atoi (chvalue);

  switch (*(In+3))
   {
    case 'V':
            switch (*(In+4))
             {
              case 'F':
               if (value == -1)
                      {
                       sprintf (out, ">GVF%4d.", (int)kff*100);
                       return;
                      }
               else
                      {
                       Reset1 = Reset2 = ON;
                       kff = (float)value/100.0;
                       sprintf (out, "OK");
                      }
               break;
              case 'P':
               if (value == -1)
```

```
                {
                 sprintf (out, ">GVP%4d.", kvp);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kvp = value;
                 sprintf (out, "OK");
                }
           break;
          case 'I':
           if (value == -1)
                {
                 sprintf (out, ">GVI%4d.", kvi);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kvi = value;
                 sprintf (out, "OK");
                }
          break;
         case 'D':
          if (value == -1)
           {
                sprintf (out, ">GVD%4d.", kvd);
                return;
           }
          else
           {
                Reset1 = Reset2 = ON;
                kvd = value;
                sprintf (out, "OK");
           }
          break;
         }
        break;
case 'P':
        switch (*(In+4))
         {
          case 'P':
           if (value == -1)
                {
                 sprintf (out, ">GPP%4d.", (int)kpp);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kpp = (float)value/10.0;
                 sprintf (out, "OK");
                }
           break;
          case 'I':
           if (value == -1)
                {
                 sprintf (out, ">GPI%4d.", (int)kpi);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kpi = (float)value/10.0;  /* WAS 10000 */
                 sprintf (out, "OK");
                }
           break;
          case 'D':
```

```
            if (value == -1)
                {
                 sprintf (out, ">GPD%4d.", (int)kpd);
                 return;
                }
            else
                {
                 Reset1 = Reset2 = ON;
                 kpd = (float)value/10.0;
                 sprintf (out, "OK");
                }
            break;
          }
    }
}

ControlLaw (char *In, char *out)  /* >1C1. on, >1C0. off */
{
 if (*(In+3) == '0')
  {
   BeginMove = OFF;
   NewMove = FALSE;
   Control = OFF;
   takedata = OFF;
   UseVelocityControl = OFF;
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   sprintf (out, "OK");
  }
 else
  if (*(In+3) == '1')
   {
        Control = ON;
        Reset1 = Reset2 = ON;
        sprintf (out, "OK");
   }
}

DtoAOut (char *In, char *out)  /* >1A4321. 4321 on DtoA */
{
 char chvalue[5];
 BYTEMODE value;

 Control = OFF;

 strncpy (chvalue, (In+3), 4);
 chvalue[4] = '\0';

 value.word = atoi (chvalue);
 outport (0x81, value.byte.lsb);
 outport (0x82, value.byte.msb);
 sprintf (out, "OK");
}

/*================================================================

Position loop!

This task generates the actuation using PID.

communicates with the velocity loop thru the shared variable PseudoVelo

================================================================*/
nodebug task1()
{
 static long ActPos, ActVelocity;
 static float DeltaPErr0, DeltaPErr1, DeltaPErr2;
 static float ipActuation;
 static long DesVelo;
```

```
 static int TimePeriods;
 static float DesPos;
 static int LastPoint;
 static long dResult, NewPos, EndPos;
 static BYTEMODE RealActuation;
 static unsigned result, result0;
 static float dAbsPos[5];
 static long OldgActPosition;
 static int index, RollOver;
 static unsigned int lo0, lo1, hi0, hi1;
 static float yPosIncrement, xPosIncrement;
 static float yPosBegin, xPosBegin, PrevNewPos;

#GLOBAL_INIT
 {
  ActPos  = ActVelocity = 0L;
  DesPos = EndPos = 0;
  DeltaPErr0 = DeltaPErr1 = DeltaPErr2 = 0;
  DesVelo = 0L;
  ipActuation = 0;
  TimePeriods = 0;
  LastPoint = FALSE;
  RollOver = 0;
  result = result0 = 0;
  gActPosition = 0;
  dAbsPos[0] = dAbsPos[1] = dAbsPos[2] = dAbsPos[3] = 0L;
  index = 0;
 }

 OldgActPosition = gActPosition;

#if (SIMULATION == 0)
 do
  {
   DI();
   lo0 = inport (0x81);
   hi0 = inport (0x80);  /* stabilize results */
   lo1 = inport (0x81);
   hi1 = inport (0x80);
   EI();
  }
 while ((lo0 != lo1) || (hi0 != hi1));
 result = ((hi1 << 8) & 0xff00) | lo1;
#else
 result = plant (1, 0);
#endif

 dResult = (long)((long)result - (long)result0);

 if (dResult > 32000)
  RollOver--;
 else
  if (-32000 > dResult)
   RollOver++;

 result0 = result;
 gActPosition = (long)result + (long)(RollOver * (long)65536);

 dAbsPos[1] = dAbsPos[0];
 dAbsPos[0] = gActPosition - OldgActPosition;
 gActVelocity = (long)((dAbsPos[0] + dAbsPos[1]) * 32.0 );

 if (gActPosition > AbsLimits)
  {
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   Control = OFF;
  }
```

```
  if (Control == OFF)
   {
    PseudoVelo = 0;
    ipActuation = 0;
    return;
   }

  ActPos = gActPosition;  /* transfer to local variables */
  ActVelocity = gActVelocity;

  if (Reset1)
   {
    DeltaPErr2 = DeltaPErr1 = DeltaPErr0 = 0;
    ipActuation = 0;
    DesPos = ActPos;
    NewPos = ActPos;
    EndPos = ActPos;
    DesVelo = 0;
    TimePeriods = 0;
    BeginMove = NewMove = FALSE;
    UseVelocityControl = OFF;
    Reset1 = OFF;
    TimePeriods = 0;
   }

#if (SIMULATION == 1)
 gettimer (BegTime);
#endif

 if (BeginMove == TRUE)
  {
   LastPoint = FALSE;
   if (NewMove == TRUE)  /* allows for interrupted move */
          {
           NewMove = FALSE;
           TimePeriods = 0;
           if (UseVelocityControl == ON)
             DesVelo = PointList[0].Velocity;
          }
   if (((TimePeriods--) <= 1) && (UseVelocityControl == OFF))
          {
           if (ExePointNumber >= PointNumber)
             {
              ExePointNumber = PointNumber;
              yPosBegin = PointList[ExePointNumber-1].toY;
              xPosBegin = PointList[ExePointNumber-1].toX;
              EndPos = atan2 (PointList[ExePointNumber-1].toY,
                             PointList[ExePointNumber-1].toX) * 14005.6;
              DesVelo = 0;
              yPosIncrement = 0;
              xPosIncrement = 0;
              BeginMove = FALSE;
             }
           else
             {
              if (PointList[ExePointNumber].TimeSlices != 0)
                    {
                     TimePeriods = (int)PointList[ExePointNumber].TimeSlices;
                     yPosIncrement = (PointList[ExePointNumber].toY -
                                     PointList[ExePointNumber-1].toY) /
                             (float) TimePeriods;
                     xPosIncrement = (PointList[ExePointNumber].toX -
                                     PointList[ExePointNumber-1].toX)
                             / (float)TimePeriods;
                     yPosBegin = PointList[ExePointNumber-1].toY;
                     xPosBegin = PointList[ExePointNumber-1].toX;
                     DesPos = ActPos;
                    }
              else
```

```
                        {
                         TimePeriods = 0;
                         yPosBegin = PointList[ExePointNumber].toY;
                         xPosBegin = PointList[ExePointNumber].toX;
                         yPosIncrement = xPosIncrement = 0;
                         DesPos = ActPos;
                         DesVelo = 0;
                        }
                 }
                hv_wr (PointList[ExePointNumber].PIOmode);
                ExePointNumber++;
                if (ExePointNumber == PointNumber) LastPoint = TRUE;
               }

     yPosBegin += yPosIncrement;
     xPosBegin += xPosIncrement;
     PrevNewPos = DesPos;
     DesPos = atan2 (yPosBegin, xPosBegin) * 14005.6;
     DesVelo = (DesPos - PrevNewPos) * 64;
     if ((LastPoint == TRUE) && (TimePeriods < 5)) DesVelo = 0;
    }
  else
    {
     yPosIncrement = xPosIncrement = 0;
     DesPos = EndPos;
    }

#if (SIMULATION == 1)
 /*   printf ("time: %d Despos: %f\n", TimePeriods, DesPos); */
#endif

  DeltaPErr2 = DeltaPErr1;  /* generate time history - 2 steps back */
  DeltaPErr1 = DeltaPErr0;
  DeltaPErr0 = DesPos - ActPos;

  ipActuation +=
    (((float)kpp * (DeltaPErr0 - DeltaPErr1)) +
     ((float)kpi * (DeltaPErr0)) +
     ((float)kpd * (DeltaPErr0 - (DeltaPErr1+DeltaPErr1) + DeltaPErr2)) +
     (kff * (DesVelo - ActVelocity)));

#if (SIMULATION == 1)
  gettimer(EndTime);
#endif

  if (UseVelocityControl == ON)
    PseudoVelo = DesVelo;
  else
    {
     PseudoVelo = ipActuation;

     if (PseudoVelo > (long)65504)
            PseudoVelo = 65504;
     else
            if (PseudoVelo < -65504)
             PseudoVelo = -65504;

     RealActuation.word = (int)(PseudoVelo >> 5);

     Actuation.word = RealActuation.word;

     RealActuation.word += D2AOFFSET;
#if (SIMULATION == 0)
     DI();
     k_lock();
     outport (0x81, (char)(RealActuation.byte.lsb));
     outport (0x82, (char)(RealActuation.byte.msb));
     k_unlock();
     EI();
```

```
#else
    plant (0, RealActuation.word);
    printf ("D,V,A,A: %f %ld %ld %d %ld\n", DesPos, DesVelo, gActPosition,
                RealActuation.word, PseudoVelo);
#endif
   }
}

/*=================================================================
This task takes sensor data.
=================================================================*/

nodebug task2()
{
 static int index;

#GLOBAL_INIT
 {
  index = 0;
 }

 if (Shutdown == TRUE) return;
 hitwd();

 if (takedata == ON)
  {
   if (datapointer == 0) index = 0;

   gettimer (data[datapointer].time);
   data[datapointer].position = gActPosition;
   data[datapointer].velocity = gActVelocity;
   data[datapointer].point = ExePointNumber;

   if (BeginMove == FALSE)
         {
          if (index > 5)
           takedata = OFF;
          else
           index++;
         }

   if (datapointer++ > 395) takedata = OFF;
  }

 if ((PIOstatus.PIOAmode & 0x2) == 0x0)
   PIOstatus.PIOAmode |= 0x2;
 else
   PIOstatus.PIOAmode &= 0xfd;

 outport (PIODA, PIOstatus.PIOAmode);
}

/*=================================================================

Velocity loop!

This task generates the velocity actuation and also takes the sensor
data from the encoder.

uses the shared variables PseudoVelo for input,
and gActPosition, gActVelocity for output.

=================================================================*/

nodebug task0()
{
 static long pVelo, dResult;
 static long DeltaVErr0, DeltaVErr1, DeltaVErr2;
 static BYTEMODE RealActuation;
```

```
 static long vActuation;
 static unsigned int result0, result;
 static float dAbsPos[5];
 static long OldgActPosition;
 static int index, RollOver;
 static unsigned int lo0, lo1, hi0, hi1;

#GLOBAL_INIT
 {
  RollOver = 0;
  result = result0 = 0;
  gActPosition = 0;
  dAbsPos[0] = dAbsPos[1] = dAbsPos[2] = dAbsPos[3] = 0L;
  DeltaVErr0 = DeltaVErr1 = DeltaVErr2 = 0L;
  index = 0;
 }

 OldgActPosition = gActPosition;

 do
  {
   DI();
   lo0 = inport (0x81);
   hi0 = inport (0x80);  /* stabilize results */
   lo1 = inport (0x81);
   hi1 = inport (0x80);
   EI();
  }
 while ((lo0 != lo1) || (hi0 != hi1));
 result = ((hi1 << 8) & 0xff00) | lo1;

 dResult = (long)((long)result - (long)result0);

 if (dResult > 32000)
  RollOver--;
 else
  if (-32000 > dResult)
   RollOver++;

 result0 = result;
 gActPosition = (long)result + (long)(RollOver * (long)65536);

 dAbsPos[1] = dAbsPos[0];
 dAbsPos[0] = gActPosition - OldgActPosition;
 gActVelocity = (long)((dAbsPos[0] + dAbsPos[1]) * 42.6667);

 pVelo = PseudoVelo;  /* transfer variables to local storage */

 if (Reset2)
  {
   DeltaVErr2 = DeltaVErr1 = DeltaVErr0 = 0L;
   vActuation = 0L;
   Reset2 = OFF;
  }

 if (gActPosition > AbsLimits)
  {
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   Control = OFF;
  }

 if (Control == OFF)
  {
   vActuation = 0;
   return;
  }

/*  DeltaVErr2 = DeltaVErr1; */
```

```
/*  DeltaVErr1 = DeltaVErr0; */
/*  DeltaVErr0 = pVelo - gActVelocity; */

/*  vActuation += (long) */
/*    ((((long)kvp*(long)(DeltaVErr0-DeltaVErr1)) + */
/*      ((long)kvi*(long)(DeltaVErr0)) + */
/*      ((long)kvd*(long)(DeltaVErr0 - (DeltaVErr1 + DeltaVErr1) + */
/*                                   DeltaVErr2)))); */

/*  if (vActuation > (long)268304384) */
/*    vActuation = 268304384; */
/*  else */
/*    if (vActuation < -268304384) */
/*      vActuation = -268304384; */

/*  RealActuation.word = ((int)(vActuation >> 17)); */

  if (pVelo > (long)2096128)
    pVelo = 2096128;
  else
    if (pVelo < -2096128)
      pVelo = -2096128;

  RealActuation.word = (int)(pVelo >> 10);

  Actuation.word = RealActuation.word;

  RealActuation.word += D2AOFFSET;
  DI();
  k_lock();
  outport (0x81, (char)(RealActuation.byte.lsb));
  outport (0x82, (char)(RealActuation.byte.msb));
  k_unlock();
  EI();
}

#JUMP_VEC NMI_VEC NMI_int

interrupt retn NMI_int()
{
  Shutdown = TRUE;
  Control = OFF;
  outport (0x81, ServoNull.byte.lsb);
  outport (0x82, ServoNull.byte.msb);
  while (1)
    {
      hitwd();
      if (!powerlo()) return;
    }
}

void FatalErrorHandler (unsigned code, unsigned address)
{
  outport (0x81, ServoNull.byte.lsb);
  outport (0x82, ServoNull.byte.msb);
  Shutdown = TRUE;
  Control = OFF;
  while (1);  /* stall until reset by watch dog */
}

float p2cos (float x)
{
  return (p2sin (90.0 - x));
}

/*============================================================
PROCEDURE: p2sin

PARAMETERS: float Y
```

```
RETURNS: float

METHOD: computes sin(Y) by:
    sin (y + dy) = (sin y)(cos dy) + ([(cos x)/57.2958])(dy)
    where Y = y + dy, y = int(Y), dy = frac(Y), and
    the parameters are found from a lookup table. result is good
    to about 5 places.

VARIABLES:
    sinx are the values of sin(y) where y varies between 0->90
    cosdx are the values of cos(dy) where dy varies between
          0->1 in 0.01 increments
    cosxd are the values of [(cos y)/57.2958] where y varies
          between 0->90
    ALL fractional values have been shifted by multiplying by
          65535 to obtain integers.
==================================================================*/

nodebug float p2sin (float y)

{
 static unsigned int sinx[] =
            {0x0000,0x0478,0x08EF,0x0D66,0x11DB,
             0x1650,0x1AC2,0x1F33,0x23A1,0x280C,
             0x2C74,0x30D9,0x3539,0x3996,0x3DEE,
             0x4242,0x4690,0x4AD9,0x4F1B,0x5358,
             0x578E,0x5BBE,0x5FE6,0x6407,0x681F,
             0x6C30,0x7039,0x7438,0x782F,0x7C1C,
             0x8000,0x83D9,0x87A8,0x8B6D,0x8F27,
             0x92D5,0x9679,0x9A10,0x9D9B,0xA11B,
             0xA48D,0xA7F3,0xAB4B,0xAE97,0xB1D4,
             0xB504,0xB826,0xBB39,0xBE3E,0xC134,
             0xC41B,0xC6F2,0xC9BA,0xCC73,0xCF1B,
             0xD1B3,0xD43B,0xD6B2,0xD919,0xDB6E,
             0xDDB3,0xDFE6,0xE208,0xE418,0xE616,
             0xE803,0xE9DD,0xEBA5,0xED5B,0xEEFE,
             0xF08F,0xF20D,0xF377,0xF4CF,0xF614,
             0xF746,0xF864,0xF96F,0xFA67,0xFB4B,
             0xFC1B,0xFCD8,0xFD81,0xFE17,0xFE98,
             0xFF06,0xFF5F,0xFFA5,0xFFD7,0xFFF5,
             0xFFFF};

 static unsigned int cosdx[] =
            {0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFE,0xFFFE,
             0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFE,
             0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFE,
             0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFD,
             0xFFFD,0xFFFD,0xFFFD,0xFFFD,0xFFFD,
             0xFFFD,0xFFFD,0xFFFD,0xFFFD,0xFFFD,
             0xFFFD,0xFFFC,0xFFFC,0xFFFC,0xFFFC,
             0xFFFC,0xFFFC,0xFFFC,0xFFFC,0xFFFC,
             0xFFFB,0xFFFB,0xFFFB,0xFFFB,0xFFFB,
             0xFFFB,0xFFFB,0xFFFB,0xFFFA,0xFFFA,
             0xFFFA,0xFFFA,0xFFFA,0xFFFA,0xFFFA,
             0xFFF9,0xFFF9,0xFFF9,0xFFF9,0xFFF9,
             0xFFF9,0xFFF8,0xFFF8,0xFFF8,0xFFF8,
             0xFFF8,0xFFF8,0xFFF7,0xFFF7,0xFFF7,
             0xFFF7,0xFFF7,0xFFF7,0xFFF6,0xFFF6,
             0xFFF6,0xFFF6,0xFFF6,0xFFF5,0xFFF5,
             0xFFF5};

 static unsigned int cosxd[] =
            {0x478,0x478,0x477,0x476,0x475,0x473,0x472,
             0x46F,0x46D,0x46A,0x466,0x463,0x45F,0x45A,
```

```
        0x456,0x451,0x44B,0x446,0x440,0x439,0x433,
        0x42C,0x425,0x41D,0x415,0x40D,0x404,0x3FB,
        0x3F2,0x3E8,0x3DF,0x3D4,0x3CA,0x3BF,0x3B4,
        0x3A9,0x39D,0x391,0x385,0x379,0x36C,0x35F,
        0x352,0x345,0x337,0x329,0x31B,0x30C,0x2FD,
        0x2EE,0x2DF,0x2D0,0x2C0,0x2B0,0x2A0,0x290,
        0x280,0x26F,0x25E,0x24D,0x23C,0x22B,0x219,
        0x207,0x1F5,0x1E3,0x1D1,0x1BF,0x1AC,0x19A,
        0x187,0x174,0x161,0x14E,0x13B,0x128,0x115,
        0x101,0x0EE,0x0DA,0x0C7,0x0B3,0x09F,0x08B,
        0x078,0x064,0x050,0x03C,0x028,0x014,0x000};

static float fremainder, x;
static unsigned int whole;

x = y;
if (x < 0)
 {
  do
   {
        x = x + 360;
   }
  while (x < 0);
 }


if (x <= 90.0)
 {
  whole = (int)x;
  fremainder = (x - whole);

  return ((float)((((unsigned long)sinx[whole]  *
                    (unsigned)cosdx[(int)(fremainder * 100.0)]) +
                   ((unsigned long)cosxd[whole] *
                    (unsigned)(fremainder * 65535)))
                  * 2.328377E-10));
 }
else
 if (x <= 180)
  {
   x = 180.0 - x;
   whole = (int)x;
   fremainder = (x - whole);

   return ((float)((((unsigned long)sinx[whole]  *
                     (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                    ((unsigned long)cosxd[whole] *
                     (unsigned long)(fremainder * 65535)))
                  * 2.328377E-10));
  }
else
 if (x <= 270)
  {
   x = x - 180.0;
   whole = (int)x;
   fremainder = (x - whole);

   return ((float)((((unsigned long)sinx[whole]  *
                     (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                    ((unsigned long)cosxd[whole] *
                     (unsigned long)(fremainder * 65535)))
                  * (-2.328377E-10)));
  }
else
 if (x <= 360)
  {
   x = 360.0 - x ;
   whole = (int)x;
   fremainder = (x - whole);
```

```
      return ((float)((((unsigned long)sinx[whole]  *
                                   (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                                  ((unsigned long)cosxd[whole] *
                                   (unsigned long)(fremainder * 65535)))
                             * (-2.328377E-10)));
    }
  return (-2);
}

#if (SIMULATION == 1)

#define FLOWGAIN 0.1

long plant (int service, int input)

{
  static float position;

#GLOBAL_INIT
  {
    position = 0.0;
  }

  if (service == 1)
    return ((long)position);
  else
    if (service == 2)
      {
             position = 0.0;
             return ((long)position);
      }

  position += ((float)(input - 2047) * FLOWGAIN);
}

#endif
```

Appendix G

Source Code Listing for Extension Joint SBC

```
#use rtk.lib

/* this program to be used on EXTENSION joint only! */

/*
This program would run on the Little Giant, the Tiny Giant and the CPLC.
If port 0 is also used as the Dynamic C programming port, you have to
load the serial interrupt routine during run time by doing the following:

  (1) comment out:
#INT_VEC SER0_VEC Dz0_circ_int in z0232.lib
  (2) in the code, declare:
           extern void Dz0_circ_int();
  (3) load the routine with:
           reload_vec(14,Dz0_circ_int);

*/

/*======
For unit 4, translation joint only!

CHANGES:
5/15/97 v5.11 changed PIOmode to single char only. 1 turns on gun,
         0 turns it off again
7/15/97 v6.10 changed to burst point loading technique. loads 8 at a time.
======*/

#define SIMULATION 0

#define ON 1
#define OFF 0
#define FWD 1
#define REV 0
#define FALSE 0
#define TRUE 1

#define VERSION 6
#define SUBVERSION 10

#define CSAMPLE 900      /* clock periods: 512HZ */
#define CTIME 0.001953   /* clock time */

#define D2AOFFSET 2047
#define ACTSCALE 1

char HVreg[8];

#define HV0 HVreg[0]
#define HV1 HVreg[1]
#define HV2 HVreg[2]
#define HV3 HVreg[3]
#define HV4 HVreg[4]
#define HV5 HVreg[5]
#define HV6 HVreg[6]     /* hydraulic shutoff solenoid */
#define HV7 HVreg[7]     /* paint gun solenoid */

int task0(), task1(), task2(), backgnd();
int (*Ftask[])()={task0, task1, task2, backgnd};

#define NTASKS 4
#define TASK0 0
#define TASK1 1
#define TASK2 2

#define TASKSTORE_SIZE 500

void (*TaskDispatchTable[30])();

typedef union bytemode
```

```
 {
  int word;
  struct ByteStruct
   {
          char lsb;
          char msb;
   } byte;
 } BYTEMODE;

struct PointListPoint
 {
  int toY, toX;
  int Velocity;
  int TimeSlices;
  char PIOmode;
 };

struct PIOstruct
 {
  char PIOAmode;
  char PIOBmode;
 };

struct PointListPoint PointList[1001];
shared struct PIOstruct PIOstatus;

struct PositionData
 {
  int time[3];
  long position;    /* variables for data logging */
  long velocity;
  int point;
 };

struct PositionData data[410];

int datapointer, takedata;

int PointNumber, ExePointNumber;        /* place in point list */
shared long gActPosition, gActVelocity;  /* actual position and velocity */
shared long PseudoVelo;                /* IPC variable b/w pos and velo loop */
shared BYTEMODE Actuation;          /* system actuation */
float kpp, kpi, kpd;        /* system position gains 1 unit, I in .01 units */
int kvp, kvi, kvd;          /* system velocity gains 0.0001 units */
float kff;                /* system feed forward gain 0.01 units */
int BeginMove;              /* change to TRUE to begin motion */
int NewMove;
int UseVelocityControl;     /* bypasses position controller */
int Acceleration;
int BegTime[3], EndTime[3];

BYTEMODE ServoNull;
shared long MaxVelo, MaxLimits, AbsLimits;
shared int ActScale;
int Control, Shutdown, Reset1, Reset2;
int UnitNumber, DumpSerial;
char InString[255], OutString[255];
char PowerFail[] = "!Power fail\0";
char SoftReset[] = "!Soft reset\0";
char tbuf[384], rbuf[384];

int AnalogInput (char *, char *);
int HighVoltage (char *, char *);
int ChangeHighVoltage (char *, char *);
int InitPIO (char *, char *);
int OutInPIO (char *, char *);
int SetGetGain (char *, char*);
int ControlLaw (char *, char *);
int DtoAOut (char *, char *);
```

```
int PosLoad (char *, char *);
int ExecuteMove (char *, char *);
int Diagnostic (char *, char *);
int ShutDown (char *, char *);
int RevisionLevel (char *, char *);
int MaxLimit (char *, char *);
void FatalErrorHandler (unsigned, unsigned);
float p2sin(float);
float p2cos(float);
void NotUsed(void);
long plant (int, int);

/* extern void Dz0_circ_int(); */

/* 0x10 stores network node number, 0x12 stores null.lsb, 0x13 stores
  null.msb */

root main ()
{
 int index;

 if (wderror())
  {
    Shutdown = TRUE;
    Control = OFF;
    outport (0x81, ServoNull.byte.lsb);
    outport (0x82, ServoNull.byte.msb);
    op_kill_z1();
    op_init_z1 (19200/1200, InString, UnitNumber);
    while (1)
          {
           hitwd();
           if (check_opto_command() == 1)
             replyOpto22 (SoftReset, strlen (SoftReset), 0);
          }
  }

 _GLOBAL_INIT();

 ERROR_EXIT = FatalErrorHandler;
 BeginMove = FALSE;
 UseVelocityControl = OFF;
 Control = DumpSerial = OFF;
 Shutdown = takedata = OFF;
 HV0 = HV1 = HV2 = HV3 = HV4 = HV5 = HV6 = HV7 = 0;
 NewMove = FALSE;
 Reset1 = Reset2 = ON;
 ExePointNumber = PointNumber = 0;
 datapointer = 0;
 gActVelocity = gActPosition = 0L;
 kpp = kpi = kpd = kff = 0;
 kvp = kvi = kvd = 0;
 MaxLimits = AbsLimits = 88000L;
 Acceleration = 5000;
 MaxVelo = 0L;
 ActScale = ACTSCALE;
 Actuation.word = 0;    /* zero! */
 PIOstatus.PIOAmode = 0x1;
 PIOstatus.PIOBmode = (char) 0;
 UnitNumber = ee_rd(0x10);
 ServoNull.byte.lsb = ee_rd(0x12);
 ServoNull.byte.msb = ee_rd(0x13);

#if (SIMULATION == 0)
 inport (0x82);                /* clear the encoder */
#else
 plant (2,0);
#endif
```

```
  outport (0x81, ServoNull.byte.lsb);  /* zero the d-to-a board */
  outport (0x82, ServoNull.byte.msb);
  outport (PIOCA, 0xff);             /* pioa command */
  outport (PIOCA, 0x00);              /* set all bits for output */
  outport (0x41, (char)0x7);          /* disable interrupts */
  outport (PIODA, PIOstatus.PIOAmode); /* pioa data */

  outport (PIOCB, 0xcf);             /* piob command */
  outport (PIOCB, 0x00);              /* set all bits for input */
  outport (PIOCB, PIOB_VEC);
  outport (PIOCB, 0x17);
  outport (PIOCB, 0xfe);
  outport (0x43, 0x7);              /* disable interrupts */
 /* outport (PIODB, PIOstatus.PIOBmode); */ /* piob data */
  outport (PIOCB, 0x7);

/*
 reload_vec (14, Dz0_circ_int);
 Dinit_z0(rbuf, tbuf, 384, 384, 4, 9600/1200, 0, 0);
*/

  op_init_z1 (19200/1200, InString, UnitNumber);

  for (index = 0; index < 26; index++)
   (TaskDispatchTable[index]) = NotUsed;

  TaskDispatchTable['L'-'A'] = AnalogInput;
  TaskDispatchTable['H'-'A'] = HighVoltage;
  TaskDispatchTable['V'-'A'] = ChangeHighVoltage;
  TaskDispatchTable['P'-'A'] = InitPIO;
  TaskDispatchTable['D'-'A'] = OutInPIO;
  TaskDispatchTable['G'-'A'] = SetGetGain;
  TaskDispatchTable['C'-'A'] = ControlLaw;
  TaskDispatchTable['A'-'A'] = DtoAOut;
  TaskDispatchTable['E'-'A'] = PosLoad;
  TaskDispatchTable['X'-'A'] = ExecuteMove;
  TaskDispatchTable['I'-'A'] = Diagnostic;
  TaskDispatchTable['M'-'A'] = MaxLimit;
  TaskDispatchTable['S'-'A'] = ShutDown;
  TaskDispatchTable['R'-'A'] = RevisionLevel;

  DI ();
  init_kernel ();
  run_every (TASK2, 100);
#if (SIMULATION == 0)
  run_every (TASK1, 8);
  run_every (TASK0, 20);
#endif
  init_timer0 (900);
  EI ();

  backgnd ();
}

void NotUsed(void)
{
 return;
}

backgnd ()
{
 while (1)
  {
#if (SIMULATION == 1)
  task1();
#endif

    if (Shutdown)
     {
```

```
            op_kill_z1();
            op_init_z1 (19200/1200, InString, UnitNumber);
            outport (0x81, ServoNull.byte.lsb);
            outport (0x82, ServoNull.byte.msb);
            hv_wr (0);
            hv_dis();

            while (1)
             {
              if (check_opto_command() == 1)
                replyOpto22 (PowerFail, strlen (PowerFail), 0);
              hitwd();
             }
      }

   if (check_opto_command() == 1)
     {
            /* strcpy (OutString, "!OK\0"); */
       OutString[0] = '!';
       OutString[1] = 'O';
       OutString[2] = 'K';
       OutString[3] = 0;

            InString[InString[1]] = '\0';

/*          if (isalpha (InString[4])) */
             {
              /* (*TaskDispatchTable[toupper(InString[4]) - 'A']) */
              (*TaskDispatchTable[InString[4] - 'A'])
                (InString+2, OutString);
             }
            /*
             if (DumpSerial)
             {
             Dwrite_z0(InString+1, InString[0] - 2);
             Dwrite_z0(OutString, strlen (OutString));
             }
       */

            replyOpto22 (OutString, strlen (OutString), 0);
     }
  }
 while (1);
}


MaxLimit (char *In, char *Out)
{
 char chvalue[8];
 long value;

 switch (toupper(*(In+3)))
  {
   case 'V':  /* >1MV98765. */ /* velocity limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxVelo = atol (chvalue);
    sprintf (Out, "OK");
    break;
   case 'L':  /* >1ML12345. */ /* Extension limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxLimits = atol (chvalue);
    AbsLimits = MaxLimits + 200L;
    sprintf (Out, "OK");
    break;
   case 'S':  /* >1MS2. */  /* Actuator scaling */
    strncpy (chvalue, (In+4), 1);
    chvalue[1] = '\0';
```

```
      ActScale = atoi (chvalue);
      sprintf (Out, "OK");
      break;
    case 'N':  /* >1MN2047. */ /* set null point to 2047 */
      strncpy (chvalue, (In+4), 4);
      chvalue[4] = '\0';
      ServoNull.word = atoi(chvalue);
      if (ServoNull.word == -1)
            {
             ServoNull.byte.lsb = ee_rd (0x12);
             ServoNull.byte.msb = ee_rd (0x13);
             sprintf (Out, ">MN%4d.\0", ServoNull.word);
            }
      else
            {
             ee_wr (0x12, ServoNull.byte.lsb);
             ee_wr (0x13, ServoNull.byte.msb);
             sprintf (Out, "Servo null: %x %x", ee_rd (0x12), ee_rd(0x13));
             outport (0x81, ServoNull.byte.lsb);
             outport (0x82, ServoNull.byte.msb);
            }
      break;
    case 'U':  /* >1MU2. */ /* set unit number to 2 */
      strncpy (chvalue, (In+4), 1);
      chvalue[1] = '\0';
      ee_wr (0x10, atoi (chvalue));
      sprintf (Out, ">1MU%1d.\0", ee_rd (0x10));
      break;
  }
}

PosLoad (char *In, char *Out)
{
 char chvalue[8];
 char **pchvalue;
 long value;
 int value1, index, index2, offset;
 float CycTime;
 float TempFloat1, TempFloat2;

 if (*(In+3) == 'S')
  {
   strncpy (chvalue, (In+4), 4);
   chvalue[4] = '\0';
   value1 = atoi (chvalue);

   if (value1 == -1)  /* >1ES__-1. */
         {
          sprintf (Out, ">ES%5ld%5ld.", gActPosition, gActVelocity);
          return;
         }
   else
         if (value1 == -2)  /* >1ES__-2. */
          {
           sprintf (Out, ">ES%5ld%5d.", gActPosition, Actuation.word);
           return;
          }
         else
          if ((value1 >= 0) && (value1 < PointNumber))
           {
            sprintf (Out, ">ES%5d%5d%4d%4d%1d.",
                     PointList[value1].toX,
                     PointList[value1].toY,
                     PointList[value1].Velocity,
                     PointList[value1].TimeSlices,
                     PointList[value1].PIOmode);
            return;
           }
   sprintf (Out, "!ES ERROR");
```

```
    return;
   }

  if (*(In+3) == 'R')
   {                              /* replace pt 0000 */
    /* >1ER0000123456987654000. pos=123456,velo=987654,pio=000 */
    sprintf (Out, "OK");
    return;
   }

  if (*(In+3) == 'L')
   {        /* >1EL123451234598760. pos=12345,velo=9876,pio=0 */
    strncpy (chvalue, (In+4), 1);
    chvalue[1]= '\0';
    index = atoi (chvalue);

    for (index2 = 0; index2 < index;)
          {
           offset = index2 * 15;
           strncpy (chvalue, In + (offset+5), 5);
           chvalue[5] = '\0';
           value = atoi (chvalue);

           if ((value > MaxLimits))
             PointList[PointNumber+index2].toX = MaxLimits;
           else
             PointList[PointNumber+index2].toX = (int)value;

           strncpy (chvalue, In +(offset+10), 5);
           chvalue[5] = '\0';
           value = atoi (chvalue);

           if ((value > MaxLimits))
             PointList[PointNumber+index2].toY = MaxLimits;
           else
             PointList[PointNumber+index2].toY = (int)value;

           strncpy (chvalue, In + (offset+15), 4);
           chvalue[4] = '\0';
           value = atoi(chvalue);

           if ((value == 0) && (UseVelocityControl == OFF))
             PointList[PointNumber+index2].Velocity = 200;
           else
             PointList[PointNumber+index2].Velocity = (int)value;

           PointList[PointNumber+index2].PIOmode = atoi ((In + (offset+19)));

           if (PointList[PointNumber+index2].Velocity != 0)
             {
              if (PointNumber == 0)
                     {
                      PointList[PointNumber].TimeSlices = 0;
                     }
        else
          {
#if (SIMULATION == 1)
                     printf ("PosLoad Velocity: %d\n",
                             PointList[PointNumber].Velocity);
#endif
                     TempFloat1 = (float)(PointList[PointNumber+index2].toY -
                                         PointList[PointNumber-1+index2].toY);
                     TempFloat2 = (float)(PointList[PointNumber+index2].toX -
                                         PointList[PointNumber-1+index2].toX);
                CycTime =
                      sqrt((TempFloat1*TempFloat1) + (TempFloat2*TempFloat2))
                      / (float)PointList[PointNumber+index2].Velocity;
                     PointList[PointNumber+index2].TimeSlices = CycTime * 64;
```

```
#if (SIMULATION == 1)
          printf ("TimeSlices %d\n", PointList[PointNumber].TimeSlices);
#endif
              }
            }
           else
            {
              PointList[PointNumber+index2].TimeSlices = 0;
            }
           index2++;
          }

   PointNumber += index2;
   sprintf (Out, "OK");
   return;
  }

 if (*(In+3) == 'C')
  {
   PointNumber = 0;
   ExePointNumber = 1;
   datapointer = 0;
   takedata = OFF;
   sprintf (Out, "OK");
   return;
  }

 if (*(In+3) == 'V')                    /* >1EV1. */
  {
   sprintf (Out, "OK");
   if (*(In+4) == '1')
          {
           UseVelocityControl = ON;
           /* outport (PIOCB, 0x87); */ /* enable interrupt driven */
          }
   else
          {
           UseVelocityControl = OFF;
           Reset1 = TRUE;
           /* outport (PIOCB, 0x7);  */ /* disable interrupt driven */
          }
   return;
  }
}

ExecuteMove (char *In, char *Out)
{
 datapointer = 0;
 takedata = ON;
 ExePointNumber = 1;
 BeginMove = TRUE;
 NewMove = TRUE;
}

Diagnostic (char *In, char *Out)
{
 static int indexer;

#GLOBAL_INIT
 {
  indexer = 0;
 }

 switch (*(In+3))
  {
   case '0':
          indexer = 0;
          if (BeginMove == FALSE)
```

```
            sprintf (Out, ">%4d.\0", datapointer);
          else
            sprintf (Out, "  -1\0");
          break;
    case '1':
          if (BeginMove == FALSE)
            {
              sprintf (Out, ">%u,%d,%ld,%ld.\0",
                          data[indexer].time[0], data[indexer].point,
                          data[indexer].position, data[indexer].velocity);
              indexer++;
            }
          break;
    case '2':
          if (DumpSerial == TRUE)
            DumpSerial = FALSE;
          else
            DumpSerial = TRUE;
          break;
    case '3':
          sprintf (Out, ">%d\0", PointNumber);
          break;
  }
}

ShutDown (char *In, char *Out)
{
  BYTEMODE index;

#if (SIMULATION == 1)
  plant (2,0);
#endif

  hv_wr (0);
  hv_dis();
  DI();
  inport (0x82);              /* clear the encoder */
  outport (0x81, ServoNull.byte.lsb);       /* zero the d-to-a board */
  outport (0x82, ServoNull.byte.msb);
  outport (0x41, (char)0xf);  /* pioa command */
  outport (0x41, (char)0x0);  /* set all bits for output */
  outport (0x41, (char)0x7);  /* disable interrupts */
  outport (0x40, (char)0x0);  /* pioa data */
  outport (0x43, (char)0xf);  /* piob command */
  outport (0x43, (char)0x0);  /* set all bits for output */;
  outport (0x43, (char)0x7);  /* disable interrupts */
  outport (0x42, (char)0x0);  /* piob data */
  Shutdown = ON;
  UseVelocityControl = OFF;
  sprintf (Out, "OK");
}

RevisionLevel (char *In, char *Out)
{
  BYTEMODE index;


  Control = OFF;
  Shutdown = OFF;
  BeginMove = FALSE;
  NewMove = FALSE;
  UseVelocityControl = OFF;
  Reset1 = Reset2 = ON;
  gActVelocity = 0L;
  gActPosition = 0L;
  kpp = kpi = kpd = kff = 0;
  kvp = kvi = kvd = 0;
  PointNumber = ExePointNumber = 0;
  Actuation.word = 0;
```

```
 outport (0x81, ServoNull.byte.lsb);
 outport (0x82, ServoNull.byte.msb);
 inport (0x82);
 sprintf (Out, ">unit:%d Rev beta %d.%d.", UnitNumber, VERSION, SUBVERSION);
#if (SIMULATION == 1)
 plant (2,0);
#endif
}

AnalogInput (char *In, char *Out) /* >1L0. channel 0 */
{
 sprintf (Out, ">L%4d.", ad_rd8(atoi(In+3)));
}

nodebug HighVoltage (char *In, char *Out)   /* >1H1. on, >1H0. off */
{
 if (*(In+3) == '1')
  {
   hv_enb();
/*     sprintf (Out, "OK"); */
   /* strcpy (Out, "OK\0"); */
   *Out = 'O';
   *(Out + 1) = 'K';
   *(Out + 2) = 0;
  }
 else
  {
   hv_dis();
/*     sprintf (Out, "OK"); */
/*    strcpy (Out, "OK\0"); */
    *Out = 'O';
    *(Out+1) = 'K';
    *(Out+2) = 0;
  }
}

nodebug ChangeHighVoltage (char *In, char *Out)   /* >1V123.  port pattern 123 */
{
 char chvalue[4];
 char value;

/* strncpy (chvalue, (In+3), 3);
 chvalue[3] = 0;

 value = atoi(chvalue); */
 value = atoi (In+3);
 HVreg[0] = value & 0x1;
 HVreg[1] = (value >> 1) & 0x1;
 HVreg[2] = (value >> 2) & 0x1;
 HVreg[3] = (value >> 3) & 0x1;
 HVreg[4] = (value >> 4) & 0x1;
 HVreg[5] = (value >> 5) & 0x1;
 HVreg[6] = (value >> 6) & 0x1;
 HVreg[7] = (value >> 7) & 0x1;

/*  sprintf (Out, "OK"); */
/* strcpy (Out, "OK\0"); */
 *Out = 'O';
 *(Out+1) = 'K';
 *(Out+2) = 0;
}

InitPIO (char *In, char *Out)  /* >1PA0103. mode 01, control 03, on PIO A */
{
 char InChar[3];
 char mode, control;

 strncpy (InChar, (In+4), 2);
 InChar[2] = 0;
```

```
  mode = atoi (InChar);

  strncpy (InChar, (In+6), 2);
  control = atoi (InChar);

  switch (*(In+3))
   {
    case 'A':
           outport (PIOCA, ((mode << 6) & 0xf0) | 0x0f);
           outport (PIOCA, (char)control);
           sprintf (Out, "OK");
           break;
    case 'B':
           outport (PIOCB, ((mode << 6) & 0xf0) | 0x0f);
           outport (PIOCB, (char)control);
           sprintf (Out, "OK");
           break;
   }
}

OutInPIO (char *In, char *Out)  /* >1DOB000. Output on PIO B value 000,
                                   >1DIA.  Input on PIO A returns 145 */
{
 char chvalue[4];
 int value;

 chvalue[3] = 0;
 switch (*(In+3))
  {
   case 'O':
          if (*(In+4) == 'A')
           {
            strncpy (chvalue, (In+5), 3);
            value = atoi(chvalue);
            PIOstatus.PIOAmode = (char)value | (PIOstatus.PIOAmode & 0x3);
            outport (PIODA, PIOstatus.PIOAmode);
            sprintf (Out, "OK");
           }
          else
           if (*(In+4) == 'B')
            {
             strncpy (chvalue, (In+5), 3);
             value = atoi (chvalue);
             PIOstatus.PIOBmode = (char)value;
             outport (PIODB, (char)value);
             sprintf (Out, "OK");
            }
          break;
   case 'I':
          if (*(In+4) == 'A')
           {
            value = inport (PIODA);
            sprintf (Out, ">DIA%3d.", value);
           }
          else
           if (*(In+4) == 'B')
            {
             value = inport (PIODB);
             sprintf (Out, ">DIB%3d.", value);
            }
           break;
  }
}

SetGetGain (char *In, char *out)  /* >1GPP1234., Pos. Proportional to 1234
                                     " -1" to report */
{
 int value;
 char chvalue[5];
```

```
strncpy (chvalue, (In+5), 4);
chvalue[4] = '\0';

value = atoi (chvalue);

switch (*(In+3))
 {
  case 'V':
        switch (*(In+4))
         {
          case 'F':
           if (value == -1)
                {
                 sprintf (out, ">GVF%4d.", (int)kff*100);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kff = (float)value/100.0;
                 sprintf (out, "OK");
                }
           break;
          case 'P':
           if (value == -1)
                {
                 sprintf (out, ">GVP%4d.", kvp);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kvp = value;
                 sprintf (out, "OK");
                }
           break;
          case 'I':
           if (value == -1)
                {
                 sprintf (out, ">GVI%4d.", kvi);
                 return;
                }
           else
                {
                 Reset1 = Reset2 = ON;
                 kvi = value;
                 sprintf (out, "OK");
                }
          break;
         case 'D':
          if (value == -1)
           {
                sprintf (out, ">GVD%4d.", kvd);
                return;
           }
          else
           {
                Reset1 = Reset2 = ON;
                kvd = value;
                sprintf (out, "OK");
           }
          break;
         }
        break;
  case 'P':
        switch (*(In+4))
         {
          case 'P':
```

```
                if (value == -1)
                        {
                         sprintf (out, ">GPP%4d.", (int)kpp);
                         return;
                        }
                else
                        {
                         Reset1 = Reset2 = ON;
                         kpp = (float)value;
                         sprintf (out, "OK");
                        }
                break;
              case 'I':
                if (value == -1)
                        {
                         sprintf (out, ">GPI%4d.", (int)kpi);
                         return;
                        }
                else
                        {
                         Reset1 = Reset2 = ON;
                         kpi = (float)value;  /* WAS 10000 */
                         sprintf (out, "OK");
                        }
                break;
              case 'D':
                if (value == -1)
                        {
                         sprintf (out, ">GPD%4d.", (int)kpd);
                         return;
                        }
                else
                        {
                         Reset1 = Reset2 = ON;
                         kpd = (float)value;
                         sprintf (out, "OK");
                        }
                break;
            }
     }
}

nodebug ControlLaw (char *In, char *out)  /* >1C1. on, >1C0. off */
{
 if (*(In+3) == '0')
   {
    HV6 = 0;
    hv_dis ();
    BeginMove = OFF;
    NewMove = FALSE;
    Control = OFF;
    takedata = OFF;
    UseVelocityControl = OFF;
    outport (0x81, ServoNull.byte.lsb);
    outport (0x82, ServoNull.byte.msb);
/*      sprintf (out, "OK"); */
/*     strcpy (out, "OK\0"); */
    *out = 'O';
    *(out+1) = 'K';
    *(out+2) = 0;
   }
 else
   if (*(In+3) == '1')
    {
         hv_enb ();
         HV6 = 1;
         Control = ON;
         Reset1 = Reset2 = ON;
/*       sprintf (out, "OK"); */
```

```
/*    strcpy (out, "OK\0"); */
    *out = 'O';
    *(out+1) = 'K';
    *(out+2) = 0;
    }
}

DtoAOut (char *In, char *out)  /* >1A4321. 4321 on DtoA */
{
 char chvalue[5];
 BYTEMODE value;

 Control = OFF;

 strncpy (chvalue, (In+3), 4);
 chvalue[4] = '\0';

 value.word = atoi (chvalue);
 outport (0x81, value.byte.lsb);
 outport (0x82, value.byte.msb);
 sprintf (out, "OK");
}

/*================================================================

Position loop!

This task generates the actuation using PID.

communicates with the velocity loop thru the shared variable PseudoVelo

=================================================================*/
nodebug task1()
{
 static long ActPos, ActVelocity;
 static float DeltaPErr0, DeltaPErr1, DeltaPErr2;
 static float ipActuation;
 static long DesVelo;
 static int TimePeriods;
 static float DesPos;
 static int LastPoint;
 static long dResult, NewPos, EndPos;
 static BYTEMODE RealActuation;
 static int result;
 static float dAbsPos[5];
 static long OldgActPosition;
 static int index, RollOver;
 static unsigned int lo0, lo1, hi0, hi1;
 static float yPosIncrement, xPosIncrement;
 static float yPosBegin, xPosBegin, PrevNewPos;

#GLOBAL_INIT
 {
  ActPos  = ActVelocity = 0L;
  DesPos = EndPos = 0;
  DeltaPErr0 = DeltaPErr1 = DeltaPErr2 = 0;
  DesVelo = 0L;
  ipActuation = 0;
  TimePeriods = 0;
  LastPoint = FALSE;
  RollOver = 0;
  result = 0;
  gActPosition = 0;
  dAbsPos[0] = dAbsPos[1] = dAbsPos[2] = dAbsPos[3] = 0L;
  index = 0;
 }

 OldgActPosition = gActPosition;
```

```
#if (SIMULATION == 0)
 do
  {
   DI();
   lo0 = inport (0x81);
   hi0 = inport (0x80);  /* stabilize results */
   lo1 = inport (0x81);
   hi1 = inport (0x80);
   EI();
  }
 while ((lo0 != lo1) || (hi0 != hi1));
 result = ((hi1 << 8) & 0xff00) | lo1;
#else
 result = plant (1, 0);
#endif

 gActPosition = 5588.0 * p2sin (16.1125 + (0.00409091 * (float)result));
 dAbsPos[1] = dAbsPos[0];
 dAbsPos[0] = gActPosition - OldgActPosition;
 gActVelocity = (long)((dAbsPos[0] + dAbsPos[1]) * 32.0 );

 if (gActPosition > AbsLimits)
  {
   outport (0x81, ServoNull.byte.lsb);
   outport (0x82, ServoNull.byte.msb);
   Control = OFF;
  }

 if (Control == OFF)
  {
   PseudoVelo = 0;
   ipActuation = 0;
   return;
  }

 ActPos = gActPosition;  /* transfer to local variables */
 ActVelocity = gActVelocity;

 if (Reset1)
  {
   DeltaPErr2 = DeltaPErr1 = DeltaPErr0 = 0;
   ipActuation = 0;
   DesPos = ActPos;
   NewPos = ActPos;
   EndPos = ActPos;
   DesVelo = 0;
   TimePeriods = 0;
   BeginMove = NewMove = FALSE;
   UseVelocityControl = OFF;
   Reset1 = OFF;
   ExePointNumber = PointNumber = 0;
  }

#if (SIMULATION == 1)
 gettimer (BegTime);
#endif

 if (BeginMove == TRUE)
  {
   LastPoint = FALSE;
   if (NewMove == TRUE)  /* allows for interrupted move */
         {
          NewMove = FALSE;
          TimePeriods = 0;
          if (UseVelocityControl == ON)
            DesVelo = PointList[0].Velocity;
         }
   if (((TimePeriods--) <= 1) && (UseVelocityControl == OFF))
         {
```

```
            if (ExePointNumber >= PointNumber)
             {
              ExePointNumber = PointNumber;
              yPosBegin = PointList[ExePointNumber-1].toY;
              xPosBegin = PointList[ExePointNumber-1].toX;

              EndPos = sqrt ((yPosBegin * yPosBegin) +
                             (xPosBegin * xPosBegin));
              DesVelo = 0;
              yPosIncrement = 0;
              xPosIncrement = 0;
              BeginMove = FALSE;
             }
            else
             {
              if (PointList[ExePointNumber].TimeSlices != 0)
                   {
                    TimePeriods = (int)PointList[ExePointNumber].TimeSlices;
                    yPosIncrement = (PointList[ExePointNumber].toY -
                                      PointList[ExePointNumber-1].toY) /
                             (float) TimePeriods;

                    xPosIncrement = (PointList[ExePointNumber].toX -
                                      PointList[ExePointNumber-1].toX)
                             / (float)TimePeriods;
                    yPosBegin = PointList[ExePointNumber-1].toY;
                    xPosBegin = PointList[ExePointNumber-1].toX;
                    DesPos = ActPos;
                   }
              else
                   {
                    TimePeriods = 0;
                    yPosBegin = PointList[ExePointNumber].toY;
                    xPosBegin = PointList[ExePointNumber].toX;
                    yPosIncrement = xPosIncrement = 0;
                    DesPos = ActPos;
                    DesVelo = 0;
                   }
             }
            HV7 = (PointList[ExePointNumber].PIOmode);
            ExePointNumber++;
            if (ExePointNumber == PointNumber) LastPoint = TRUE;
           }

    yPosBegin += yPosIncrement;
    xPosBegin += xPosIncrement;
    PrevNewPos = DesPos;
    DesPos = sqrt ((xPosBegin * xPosBegin) + (yPosBegin * yPosBegin));
    DesVelo = (DesPos - PrevNewPos) * 64;
    if ((LastPoint == TRUE) && (TimePeriods < 5)) DesVelo = 0;
   }
  else
   {
    yPosIncrement = xPosIncrement = 0;
    DesPos = EndPos;
   }

#if (SIMULATION == 1)
 /*  printf ("time: %d Despos: %f\n", TimePeriods, DesPos); */
#endif

  DeltaPErr2 = DeltaPErr1; /* generate time history - 2 steps back */
  DeltaPErr1 = DeltaPErr0;
  DeltaPErr0 = DesPos - ActPos;

  ipActuation +=
   (((float)kpp * (DeltaPErr0 - DeltaPErr1)) +
    ((float)kpi * (DeltaPErr0)) +
    ((float)kpd * (DeltaPErr0 - (DeltaPErr1+DeltaPErr1) + DeltaPErr2)) +
```

```
    (kff * (DesVelo - ActVelocity)));

 #if (SIMULATION == 1)
  gettimer(EndTime);
 #endif

  if (UseVelocityControl == ON)
   PseudoVelo = DesVelo;
  else
   {
    PseudoVelo = ipActuation;

    if (PseudoVelo > (long)65504)
           PseudoVelo = 65504;
    else
           if (PseudoVelo < -65504)
            PseudoVelo = -65504;

    RealActuation.word = (int)(PseudoVelo >> 5);

    Actuation.word = RealActuation.word;

    RealActuation.word += D2AOFFSET;

#if (SIMULATION == 0)
    DI();
    k_lock();
    outport (0x81, (char)(RealActuation.byte.lsb));
    outport (0x82, (char)(RealActuation.byte.msb));
    k_unlock();
    EI();
#else
    plant (0, RealActuation.word);
    printf ("D,V,A,A: %f %ld %ld %d %d %d\n", DesPos, DesVelo, gActPosition,
               RealActuation.word, EndTime[0], BegTime[0]);
#endif
   }
}

/*=================================================================
This task takes sensor data.
=================================================================*/

nodebug task2()
{
 static int index;

#GLOBAL_INIT
 {
  index = 0;
 }

 if (Shutdown == TRUE) return;
 hitwd();

 if (takedata == ON)
  {
   if (datapointer == 0) index = 0;

   gettimer (data[datapointer].time);
   data[datapointer].position = gActPosition;
   data[datapointer].velocity = gActVelocity;
   data[datapointer].point = ExePointNumber;

   if (BeginMove == FALSE)
          {
           if (index > 5)
            takedata = OFF;
           else
```

```
              index++;
           }

     if (datapointer++ > 395) takedata = OFF;
    }

  if ((PIOstatus.PIOAmode & 0x2) == 0x0)
    PIOstatus.PIOAmode |= 0x2;
  else
    PIOstatus.PIOAmode &= 0xfd;

  outport (PIODA, PIOstatus.PIOAmode);
}

/*==================================================================

==================================================================*/

nodebug task0()
{
  char OldValue, NewValue;
#GLOBAL_INIT
  {
    OldValue = 0;
    NewValue = 0;
  }

  NewValue = HVreg[0] | (HVreg[1] << 1) | (HVreg[2] << 2) |
    (HVreg[3] << 3) | (HVreg[4] << 4) | (HVreg[5] << 5) |
    (HVreg[6] << 6) | (HVreg[7] << 7);

  if (OldValue != NewValue)
    {
     hv_wr (NewValue);
     OldValue = NewValue;
    }
}

#JUMP_VEC NMI_VEC NMI_int

interrupt retn NMI_int()
{
  Shutdown = TRUE;
  Control = OFF;
  outport (0x81, ServoNull.byte.lsb);
  outport (0x82, ServoNull.byte.msb);
  hv_dis();
  while (1)
    {
     hitwd();
     if (!powerlo()) return;
    }
}

void FatalErrorHandler (unsigned code, unsigned address)
{
  outport (0x81, ServoNull.byte.lsb);
  outport (0x82, ServoNull.byte.msb);
  Shutdown = TRUE;
  Control = OFF;
  hv_dis();
  while (1);  /* stall until reset by watch dog */
}

float p2cos (float x)
{
  return (p2sin (90.0 - x));
}
```

```
/*================================================================
PROCEDURE: p2sin

PARAMETERS: float Y

RETURNS: float

METHOD: computes sin(Y) by:
    sin (y + dy) = (sin y)(cos dy) + ([(cos x)/57.2958])(dy)
    where Y = y + dy, y = int(Y), dy = frac(Y), and
    the parameters are found from a lookup table. result is good
    to about 5 places.

VARIABLES:
    sinx are the values of sin(y) where y varies between 0->90
    cosdx are the values of cos(dy) where dy varies between
            0->1 in 0.01 increments
    cosxd are the values of [(cos y)/57.2958] where y varies
            between 0->90
    ALL fractional values have been shifted by multiplying by
            65535 to obtain integers.
=================================================================*/

nodebug float p2sin (float y)

{
 static unsigned int sinx[] =
            {0x0000,0x0478,0x08EF,0x0D66,0x11DB,
             0x1650,0x1AC2,0x1F33,0x23A1,0x280C,
             0x2C74,0x30D9,0x3539,0x3996,0x3DEE,
             0x4242,0x4690,0x4AD9,0x4F1B,0x5358,
             0x578E,0x5BBE,0x5FE6,0x6407,0x681F,
             0x6C30,0x7039,0x7438,0x782F,0x7C1C,
             0x8000,0x83D9,0x87A8,0x8B6D,0x8F27,
             0x92D5,0x9679,0x9A10,0x9D9B,0xA11B,
             0xA48D,0xA7F3,0xAB4B,0xAE97,0xB1D4,
             0xB504,0xB826,0xBB39,0xBE3E,0xC134,
             0xC41B,0xC6F2,0xC9BA,0xCC73,0xCF1B,
             0xD1B3,0xD43B,0xD6B2,0xD919,0xDB6E,
             0xDDB3,0xDFE6,0xE208,0xE418,0xE616,
             0xE803,0xE9DD,0xEBA5,0xED5B,0xEEFE,
             0xF08F,0xF20D,0xF377,0xF4CF,0xF614,
             0xF746,0xF864,0xF96F,0xFA67,0xFB4B,
             0xFC1B,0xFCD8,0xFD81,0xFE17,0xFE98,
             0xFF06,0xFF5F,0xFFA5,0xFFD7,0xFFF5,
             0xFFFF};

static unsigned int cosdx[] =
            {0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
             0xFFFF,0xFFFF,0xFFFF,0xFFFE,0xFFFE,
             0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFE,
             0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFE,
             0xFFFE,0xFFFE,0xFFFE,0xFFFE,0xFFFD,
             0xFFFD,0xFFFD,0xFFFD,0xFFFD,0xFFFD,
             0xFFFD,0xFFFD,0xFFFD,0xFFFD,0xFFFD,
             0xFFFD,0xFFFC,0xFFFC,0xFFFC,0xFFFC,
             0xFFFC,0xFFFC,0xFFFC,0xFFFC,0xFFFC,
             0xFFFB,0xFFFB,0xFFFB,0xFFFB,0xFFFB,
             0xFFFB,0xFFFB,0xFFFB,0xFFFA,0xFFFA,
             0xFFFA,0xFFFA,0xFFFA,0xFFFA,0xFFFA,
             0xFFF9,0xFFF9,0xFFF9,0xFFF9,0xFFF9,
             0xFFF9,0xFFF8,0xFFF8,0xFFF8,0xFFF8,
             0xFFF8,0xFFF8,0xFFF7,0xFFF7,0xFFF7,
             0xFFF7,0xFFF7,0xFFF7,0xFFF6,0xFFF6,
             0xFFF6,0xFFF6,0xFFF6,0xFFF5,0xFFF5,
```

```
            0xFFF5};

static unsigned int cosxd[] =
            {0x478,0x478,0x477,0x476,0x475,0x473,0x472,
             0x46F,0x46D,0x46A,0x466,0x463,0x45F,0x45A,
             0x456,0x451,0x44B,0x446,0x440,0x439,0x433,
             0x42C,0x425,0x41D,0x415,0x40D,0x404,0x3FB,
             0x3F2,0x3E8,0x3DF,0x3D4,0x3CA,0x3BF,0x3B4,
             0x3A9,0x39D,0x391,0x385,0x379,0x36C,0x35F,
             0x352,0x345,0x337,0x329,0x31B,0x30C,0x2FD,
             0x2EE,0x2DF,0x2D0,0x2C0,0x2B0,0x2A0,0x290,
             0x280,0x26F,0x25E,0x24D,0x23C,0x22B,0x219,
             0x207,0x1F5,0x1E3,0x1D1,0x1BF,0x1AC,0x19A,
             0x187,0x174,0x161,0x14E,0x13B,0x128,0x115,
             0x101,0x0EE,0x0DA,0x0C7,0x0B3,0x09F,0x08B,
             0x078,0x064,0x050,0x03C,0x028,0x014,0x000};

static float fremainder, x;
static unsigned int whole;

x = y;
if (x < 0)
 {
  do
   {
        x = x + 360;
   }
  while (x < 0);
 }


if (x <= 90.0)
 {
  whole = (int)x;
  fremainder = (x - whole);

  return ((float)((((unsigned long)sinx[whole]  *
                        (unsigned)cosdx[(int)(fremainder * 100.0)]) +
                       ((unsigned long)cosxd[whole] *
                        (unsigned)(fremainder * 65535)))
                      * 2.328377E-10));
 }
else
 if (x <= 180)
  {
   x = 180.0 - x;
   whole = (int)x;
   fremainder = (x - whole);

   return ((float)((((unsigned long)sinx[whole]  *
                           (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                          ((unsigned long)cosxd[whole] *
                           (unsigned long)(fremainder * 65535)))
                      * 2.328377E-10));
  }
else
 if (x <= 270)
  {
   x = x - 180.0;
   whole = (int)x;
   fremainder = (x - whole);

   return ((float)((((unsigned long)sinx[whole]  *
                           (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                          ((unsigned long)cosxd[whole] *
                           (unsigned long)(fremainder * 65535)))
                      * (-2.328377E-10)));
  }
else
```

```
   if (x <= 360)
     {
      x = 360.0 - x ;
      whole = (int)x;
      fremainder = (x - whole);

      return ((float)((((unsigned long)sinx[whole]  *
                                    (unsigned long)cosdx[(int)(fremainder * 100.0)]) +
                                   ((unsigned long)cosxd[whole] *
                                    (unsigned long)(fremainder * 65535)))
                              * (-2.328377E-10)));
     }
  return (-2);
 }

 #if (SIMULATION == 1)

 #define FLOWGAIN 0.1

 long plant (int service, int input)

 {
  static float position;

  if (service == 1)
    return ((long)position);
  else
    if (service == 2)
      {
             position = 0.0;
             return;
      }

  position += ((float)(input - 2047) * FLOWGAIN);
 }

 #endif
```

Appendix H

Source Code Listing for End Effector SBC

```
#include <stdio.h>

#define KEYPAD_SIZE 24
#define LK_LINES 4
#define LK_COLS 20
#define LK_BLINK 1
#define ON 1
#define OFF 0
#define FWD 1
#define REV 0
#define FALSE 0
#define TRUE 1

#define VERSION 2
#define SUBVERSION 7

#define CSAMPLE 900     /* clock periods: 512HZ */
#define CTIME 0.001953  /* clock time */

#define D2AOFFSET 2072
#define DEADBAND 300
#define ACTSCALE 1

/*** defines for RS232 communication ***/
#define IBAUD   1200        /* baud rate
                   without modem => 19200,9600, 4800, etc   */
#define TBUFSIZE 384        // size of transmit buffer
#define RBUFSIZE 384        // size of receive buffer
#define CR '\x0d'

int task0(), task1(), task2(), backgnd();
/* int (*Ftask[4])()={task0, task1, task2, backgnd}; */
int (*Ftask[2])()={task2, backgnd};


/*
#define NTASKS 4
#define TASK0 0
#define TASK1 1
#define TASK2 2

*/
#define NTASKS 2
#define TASK2 0

typedef union bytemode
 {
  int word;
  struct ByteStruct
   {
    char lsb;
    char msb;
   } byte;
 } BYTEMODE;

/*** Structures ***/
struct CmmndFormat
{
 char *Cmmnd;    /* command string - pad with 0 for fill-in-the-blanks */
 char ComLength; /* transmission length in bytes */
 char RespLength;/* command response length */
 char VarCommand;/* reserved for future use. Leave 0 */
};

struct CmmndFormat InitCmmnds[] = {

 {"",1,0,0},           /* 0: send two carriage returns to */
 {"",1,0,0},           /* 1: autobaud the SR233 */
```

```
 {"W00",4,0,0},          /* 2: send two carriage returns to */
 {"W00",4,0,0},          /* 3: autobaud both motor controllers */
 {"W00O 0A0H",10,0,0},    /* 4: set up handshaking */
 {"W00/B 0",8,0,0},       /* 5: initialize motor drivers */
 {"W00/B 2",7,0,0},       /* 6: set encoder initial value */
 {"W00B 3",8,0,0},        /* 7: HI,LO,HI,LO == 10, where bit 2 is */
 {"W00/B 4",8,0,0},       /* 8: the low bit and bit 5 is the high bit */
 {"W00B 5",8,0,0},        /* 9: possible values are 0 to 15. */
 {"W00/B 1",8,0,0},       /* 10: latch encoder value */
 {"W00B 1",7,0,0},        /* 11: */
 {"W00B 2",7,0,0},        /* 12: reset all user bits high before move */
 {"W00B 3",7,0,0},        /* 13: */
 {"W00B 4",7,0,0},        /* 14: */
 {"W00B 5",7,0,0}};       /* 15: */

struct PointListPoint
 {
  long ListPosition;
  long ListVelocity;
  char PIOmode;
  float ListpGainScale;
  float ListiGainScale;
 };

struct PIOstruct
 {
  char PIOAmode;
  char PIOBmode;
 };

struct PointListPoint PointList[100], MotPointList[100];
shared struct PIOstruct PIOstatus;
struct CmmndFormat MotHomList[100];

long data[500][2];
int datapointer, takedata;
int PointNumber, ExePointNumber;
shared int timeBeg[3], timeEnd[3];
shared long AbsPos;
shared long Velocity, PseudoVelo;
shared int VelocityControl;
int UseVelocityControl;
int kpp, kpi, kpd;
int kvp, kvi, kvd;
shared float pGainScale, iGainScale;
shared BYTEMODE Actuation, dActuation;
shared long MaxVelo, MaxLimits, AbsLimits;
shared int ActScale;
int Control, Shutdown, Reset1, Reset2;
int UnitNumber;
char InString[255], OutString[255];
int BeginMove;

/*** Global Variables for RS232 Communication ***/
char Out232[100], In232[100];
shared int ReadDelay;
char tbuf[TBUFSIZE];    // transmit buffer
char rbuf[RBUFSIZE];    // receive buffer
char buf[RBUFSIZE + 1]; // dummy buffer for receiving a complete command
char HomingCommands[50][15];// storage for homing commands

int AnalogInput (char *, char *);
int HighVoltage (char *, char *);
int ChangeHighVoltage (char *, char *);
int InitPIO (char *, char *);
int OutInPIO (char *, char *);
int SetGetGain (char *, char*);
int ControlLaw (char *, char *);
int DtoAOut (char *, char *);
```

```
int PosLoad (char *, char *);
int ExecuteMove (char *, char *);
int TimingChange (char *, char *);
int ShutDown (char *, char *);
int RevisionLevel (char *, char *);
int MaxLimit (char *, char *);
int FlipRelay8 (char *, char *);

/*** Function Prototypes for RS232 Communication ***/
int init_232();
int TurnMotor (char *, char *);
int ClearBuf(char *, int);
int Serial232Service(char *, int, int);
int EncoderService();
unsigned int CombineBits(char *, char *, char *, char *);

root main ()
{
 _GLOBAL_INIT();

 BeginMove = FALSE;
 Control = OFF;
 Shutdown = OFF;
 takedata = OFF;
 VelocityControl = OFF;
 UseVelocityControl = OFF;
 Reset1 = Reset2 = ON;
 ExePointNumber = 0;
 Velocity = AbsPos = 0L;
 kpp = kpi = kpd = 0;
 kvp = kvi = kvd = 0;
 MaxLimits = AbsLimits = MaxVelo = 0L;
 ActScale = ACTSCALE;
 Actuation.word = 0;    /* zero! */
 dActuation.word = 0;
 pGainScale = iGainScale = 1.0;
 PIOstatus.PIOAmode = 0x1;
 PIOstatus.PIOBmode = (char) 0;
 UnitNumber = ee_rd(0x10);

 inport (0x82);                  /* clear the encoder */
 outport (0x81, 0xff);             /* zero the d-to-a board */
 outport (0x82, 0x07);
 outport (PIOCA, 0xff);             /* pioa command */
 outport (PIOCA, 0x00);              /* set all bits for output */
/* outport (0x41, (char)0x7); */    /* disable interrupts */
 outport (PIODA, PIOstatus.PIOAmode); /* pioa data */

/* outport (PIOCB, 0xcf);          */   /* piob command */
/* outport (PIOCB, 0x00);          */   /* set all bits for input */;
/* outport (PIOCB, PIOB_VEC);      */
/* outport (PIOCB, 0x17);          */
/* outport (PIOCB, 0xfe);          */
 /* outport (0x43, 0x7); */         /* disable interrupts */
 /* outport (PIODB, PIOstatus.PIOBmode); */ /* piob data */
/* outport (PIOCB, 0x7); */

 op_init_z1 (19200/1200, InString, UnitNumber);

 Reset_PBus(); /* Initialize Relay8 board */
 Stall(5000);  /* 350 ms wait required after reset */
 _GLOBAL_INIT();

 DI ();
 init_kernel ();
 run_every (TASK2, 100);
/* run_every (TASK1, 23); */ /* 44.92 */ /* was 45 -or- 87.9 ms */
/* run_every (TASK0, 6); */ /* 11.72 */ /* was 12 -or- 23.4 ms */
 init_timer0 (900);     /* 512 hz clock, 0.001953 seconds */
```

```
  EI ();
  backgnd ();
 }


indirect backgnd ()
{
 while (1)
 {
  if (check_opto_command() == 1)
  {
   InString[InString[1]] = '\0';
   switch (toupper(InString[4]))
   {
   case 'L':  /* analog input requested */
    AnalogInput (InString+2, OutString);
    break;
   case 'H':  /* high voltage enable/disable */
    HighVoltage (InString+2, OutString);
    break;
   case 'V':  /* high voltage change */
    ChangeHighVoltage (InString+2, OutString);
    break;
   case 'P':  /* Initialize PIO */
    InitPIO (InString+2, OutString);
    break;
   case 'D':  /* Output/Input on PIO */
    OutInPIO (InString+2, OutString);
    break;
   case 'G':  /* Change gains */
    SetGetGain (InString+2, OutString);
    break;
   case 'C':  /* Control law */
    ControlLaw (InString+2, OutString);
    break;
   case 'A':  /* DtoA output */
    DtoAOut (InString+2, OutString);
    break;
   case 'E':  /* position load */
    PosLoad (InString+2, OutString);
    break;
   case 'X':  /* execute move */
    ExecuteMove (InString+2, OutString);
    break;
   case 'I':  /* change timing interval */
    TimingChange (InString+2, OutString);
   break;
   case 'M':  /* Maximum limits */
    MaxLimit (InString+2, OutString);
    break;
   case 'S':  /* Emergency shutdown */
    ShutDown (InString+2, OutString);
    break;
   case 'R':  /* Software revision level */
    RevisionLevel (InString+2, OutString);
    break;
   case 'F':  /* Flip relay 8 */
    FlipRelay8 (InString+2, OutString);
    break;
   case 'T':  /* Turn motor */
    TurnMotor (InString+2, OutString);
   }
   replyOpto22 (OutString, strlen (OutString), 0);
   if (Shutdown) break;
  }
 }
 while (1);
}
```

```
MaxLimit (char *In, char *Out)
{
 char chvalue[8];
 long value;

 switch (*(In+3))
  {
   case 'V':  /* >1MV98765. */ /* velocity limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxVelo = atol (chvalue);
    sprintf (Out, "OK");
    break;
   case 'L':  /* >1ML12345. */ /* Extension limits */
    strncpy (chvalue, (In+4), 5);
    chvalue[5] = '\0';
    MaxLimits = atol (chvalue);
    AbsLimits = MaxLimits + 200L;
    sprintf (Out, "OK");
    break;
   case 'S':  /* >1MS2. */ /* Actuator scaling */
    strncpy (chvalue, (In+4), 1);
    chvalue[1] = '\0';
    ActScale = atoi (chvalue);
    sprintf (Out, "OK");
    break;
  }
}

PosLoad (char *In, char *Out)
{
 char chvalue[8];
 long value;
 int value1;

 switch (*(In+3))
 {
  case 'S':
   strncpy (chvalue, (In+4), 7);
   chvalue[7] = '\0';
   value1 = atoi (chvalue);

   if (value1 == -1)  /* >1ES     -1. */
   {
    sprintf (Out, ">ES%7ld%7ld.", AbsPos, Velocity);
    return;
   }
   else
   if (value1 == -2)  /* >1ES     -2. */
   {
    /*    sprintf (Out, ">ES%6d.", Actuation.word);  */
    sprintf (Out, ">ES%6d.", timeEnd[0] - timeBeg[0]);
    return;
   }
   else
   if (value1 >= 0)
   {
    sprintf (Out, ">ES%7ld%7ld%3d.",
    PointList[value1].ListPosition,
    PointList[value1].ListVelocity,
    PointList[value1].PIOmode);
    return;
   }
   sprintf (Out, "OK");
   return;
   break;                              /* replace pt 0000 */
  case 'R': /* >1ER000012345679876543000. pos=1234567,velo=9876543,pio=0 */
   strncpy (chvalue, (In+4), 4);
```

```
chvalue[4] = '\0';
value1 = atoi (chvalue);

strncpy (chvalue, (ln+8), 7);
chvalue[7] = '\0';
value = atol (chvalue);

if ((value > MaxLimits) && (MaxLimits != 0L))
  PointList[value1].ListPosition = MaxLimits;
else
  PointList[value1].ListPosition = value;

strncpy (chvalue, (ln+15), 7);
chvalue[7] = '\0';
value = atol(chvalue);

if ((value > MaxVelo) && (MaxVelo != 0L))
  PointList[value1].ListVelocity = MaxVelo;
else
  PointList[value1].ListVelocity = value;

if (PointList[value1].ListVelocity != 0L)
{
  if (PointList[value1].ListVelocity < 20000L)
  {
    PointList[value1].ListpGainScale = 1.8;
    PointList[value1].ListiGainScale = 0.2;
  }
  else
  if (PointList[value1].ListVelocity < 80000L)
  {
    PointList[value1].ListpGainScale = 1.4;
    PointList[value1].ListiGainScale = 0.5;
  }
  else
  {
    PointList[value1].ListpGainScale = 1.0;
    PointList[value1].ListiGainScale = 1.0;
  }
}
else
{
  PointList[value1].ListpGainScale = 1.0;
  PointList[value1].ListiGainScale = 1.0;
}

strncpy (chvalue, (ln+22), 3);
chvalue[3] = 0;
PointList[value1].PIOmode = atoi (chvalue);
sprintf (Out, "OK");
break;
case 'L':  /* >1EL12345679876543000. pos=1234567,velo=9876543,pio=0 */
strncpy (chvalue, (ln+4), 7);
chvalue[7] = '\0';
value = atol (chvalue);

if ((value > MaxLimits) && (MaxLimits != 0L))
  PointList[PointNumber].ListPosition = MaxLimits;
else
  PointList[PointNumber].ListPosition = value;

strncpy (chvalue, (ln+11), 7);
chvalue[7] = '\0';
value = atol(chvalue);

if ((value > MaxVelo) && (MaxVelo != 0L))
  PointList[PointNumber].ListVelocity = MaxVelo;
else
  PointList[PointNumber].ListVelocity = value;
```

```
    if (PointList[PointNumber].ListVelocity != 0L)
    {
     if (PointList[PointNumber].ListVelocity < 20000L)
     {
      PointList[PointNumber].ListpGainScale = 1.8;
      PointList[PointNumber].ListiGainScale = 0.2;
     }
     else
     if (PointList[PointNumber].ListVelocity < 80000L)
     {
      PointList[PointNumber].ListpGainScale = 1.4;
      PointList[PointNumber].ListiGainScale = 0.5;
     }
     else
     {
      PointList[PointNumber].ListpGainScale = 1.0;
      PointList[PointNumber].ListiGainScale = 1.0;
     }
    }
    else
    {
     PointList[PointNumber].ListpGainScale = 1.0;
     PointList[PointNumber].ListiGainScale = 1.0;
    }

    strncpy (chvalue, (In+18), 3);
    chvalue[3] = 0;
    PointList[PointNumber].PIOmode = atoi (chvalue);
    sprintf (Out, "OK");
    PointNumber++;
    break;
   case 'C':
    PointNumber = 0;
    ExePointNumber = 0;
    datapointer = 0;
    sprintf (Out, "OK");
    break;
   case 'V':  /* >1EV1. */
    sprintf (Out, "OK");
    if (*(In+4) == '1')
    {
     /*    UseVelocityControl = ON; */
     outport (PIOCB, 0x87);  /* enable interrupt driven */
    }
    else
    {
     /*    UseVelocityControl = OFF; */
     outport (PIOCB, 0x7);  /* disable interrupt driven */
    }
    break;
  }
}

ExecuteMove (char *In, char *Out)
{
 takedata = ON;
 PIOstatus.PIOAmode =
   (PointList[ExePointNumber].PIOmode & 0xfe);  /* lower status flag */
 outport (PIODA, PIOstatus.PIOAmode);

 ExePointNumber = 0;
 BeginMove = TRUE;
}

#INT_VEC PIOB_VEC intrExecuteMove

interrupt reti int intrExecuteMove ()
{
```

```
  takedata = ON;

  PIOstatus.PIOAmode =
    (PointList[ExePointNumber].PIOmode & 0xfe);  /* lower status flag */
  outport (PIODA, PIOstatus.PIOAmode);

  ExePointNumber = 0;
  BeginMove = TRUE;
}


TimingChange (char *In, char *Out)
{
  int index;

  for (index=0; index < datapointer; index++)
    {
/*     printf ("\n%d,%ld,%ld", index, data[index][0], data[index][1]); */
    }
  sprintf (Out, "OK");
}

ShutDown (char *In, char *Out)
{
  BYTEMODE index;

  index.word = D2AOFFSET;
  inport (0x82);              /* clear the encoder */
  outport (0x81, index.byte.lsb);        /* zero the d-to-a board */
  outport (0x82, index.byte.msb);
  outport (0x41, (char)0xf);   /* pioa command */
  outport (0x41, (char)0x0);   /* set all bits for output */
  outport (0x41, (char)0x7);   /* disable interrupts */
  outport (0x40, (char)0x0);   /* pioa data */
  outport (0x43, (char)0xf);   /* piob command */
  outport (0x43, (char)0x0);   /* set all bits for output */;
  outport (0x43, (char)0x7);   /* disable interrupts */
  outport (0x42, (char)0x0);   /* piob data */
  Shutdown = ON;
  sprintf (Out, "OK");
}

RevisionLevel (char *In, char *Out)
{
  BYTEMODE index;

  Control = OFF;
  Shutdown = OFF;
  Reset1 = Reset2 = ON;
  Velocity = AbsPos = 0L;
  kpp = kpi = kpd = 0;
  kvp = kvi = kvd = 0;
  PointNumber = ExePointNumber = 0;
  Actuation.word = 0;
  index.word = D2AOFFSET;
  outport (0x81, index.byte.lsb);
  outport (0x82, index.byte.msb);
  inport (0x82);
  sprintf (Out, ">unit:%d Rev beta %d.%d.", UnitNumber, VERSION, SUBVERSION);
}

AnalogInput (char *In, char *Out)  /* >1L0. channel 0 */
{
  sprintf (Out, ">L%4d.", ad_rd8(atoi(In+3)));
}

HighVoltage (char *In, char *Out)   /* >1H1. on, >1H0. off */
{
  if (*(In+3) == '1')
```

```
    {
     hv_enb();
     sprintf (Out, "OK");
    }
   else
    {
     hv_dis();
     sprintf (Out, "OK");
    }
 }

ChangeHighVoltage (char *In, char *Out)   /* >1V123.  port pattern 123 */
{
 char chvalue[4];

 strncpy (chvalue, (In+3), 3);
 chvalue[3] = 0;

 hv_wr(atoi(chvalue));
 sprintf (Out, "OK");
}

InitPIO (char *In, char *Out)  /* >1PA0103. mode 01, control 03, on PIO A */
{
 char InChar[3];
 char mode, control;

 strncpy (InChar, (In+4), 2);
 InChar[2] = 0;
 mode = atoi (InChar);

 strncpy (InChar, (In+6), 2);
 control = atoi (InChar);

 switch (*(In+3))
  {
   case 'A':
    outport (PIOCA, ((mode << 6) & 0xf0) | 0x0f);
    outport (PIOCA, (char)control);
    sprintf (Out, "OK");
    break;
   case 'B':
    outport (PIOCB, ((mode << 6) & 0xf0) | 0x0f);
    outport (PIOCB, (char)control);
    sprintf (Out, "OK");
    break;
  }
}

OutInPIO (char *In, char *Out)  /* >1DOB000. Output on PIO B value 000,
                                   >1DIA.  Input on PIO A returns 145 */
{
 char chvalue[4];
 int value;

 chvalue[3] = 0;
 switch (*(In+3))
  {
   case 'O':
    if (*(In+4) == 'A')
     {
      strncpy (chvalue, (In+5), 3);
      value = atoi(chvalue);
   PIOstatus.PIOAmode = (char)value | (PIOstatus.PIOAmode & 0x3);
      outport (PIODA, PIOstatus.PIOAmode);
      sprintf (Out, "OK");
     }
    else
     if (*(In+4) == 'B')
```

```
        {
          strncpy (chvalue, (In+5), 3);
          value = atoi (chvalue);
        PIOstatus.PIOBmode = (char)value;
          outport (PIODB, (char)value);
          sprintf (Out, "OK");
        }
      break;
    case 'I':
      if (*(In+4) == 'A')
        {
          value = inport (PIODA);
          sprintf (Out, ">DIA%3d.", value);
        }
      else
        if (*(In+4) == 'B')
          {
            value = inport (PIODB);
            sprintf (Out, ">DIB%3d.", value);
          }
        break;
    }
}

SetGetGain (char *In, char *out)  /* >1GPP1234., Pos. Proportional to 1234
                                   " -1" to report */
{
 int value;
 char chvalue[5];

 strncpy (chvalue, (In+5), 4);
 chvalue[4] = '\0';

 value = atoi (chvalue);

 switch (*(In+3))
  {
   case 'V':
     switch (*(In+4))
       {
        case 'P':
          if (value == -1)
            {
              sprintf (out, ">GVP%4d.", kvp);
              return;
            }
          else
            {
              Reset1 = Reset2 = ON;
              kvp = value;
              sprintf (out, "OK", kvp);
            }
          break;
        case 'I':
          if (value == -1)
            {
              sprintf (out, ">GVI%4d.", kvi);
              return;
            }
          else
            {
              Reset1 = Reset2 = ON;
              kvi = value;
              sprintf (out, "OK");
            }
         break;
        case 'D':
         if (value == -1)
           {
```

```
              sprintf (out, ">GVD%4d.", kvd);
              return;
            }
          else
            {
              Reset1 = Reset2 = ON;
              kvd = value;
              sprintf (out, "OK");
            }
          break;
        }
      break;
    case 'P':
      switch (*(In+4))
        {
          case 'P':
            if (value == -1)
              {
                sprintf (out, ">GPP%4d.", kpp);
                return;
              }
            else
              {
                Reset1 = Reset2 = ON;
                kpp = value;
                sprintf (out, "OK", kpp);
              }
            break;
          case 'I':
            if (value == -1)
              {
                sprintf (out, ">GPI%4d.", kpi);
                return;
              }
            else
              {
                Reset1 = Reset2 = ON;
                kpi = value;
                sprintf (out, "OK", kpi);
              }
            break;
          case 'D':
            if (value == -1)
              {
                sprintf (out, ">GPD%4d.", kpd);
                return;
              }
            else
              {
                Reset1 = Reset2 = ON;
                kpd = value;
                sprintf (out, "OK", kpd);
              }
            break;
        }
    }
}

ControlLaw (char *In, char *out)  /* >1C1. on, >1C0. off */
{
  if (*(In+3) == '0')
    {
      Control = OFF;
      sprintf (out, "OK");
    }
  else
    if (*(In+3) == '1')
      {
        Control = ON;
```

```
      Reset1 = Reset2 = ON;
      sprintf (out, "OK");
    }
}

DtoAOut (char *In, char *out)  /* >1A4321. 4321 on DtoA */
{
 char chvalue[5];
 BYTEMODE value;

 Control = OFF;

 strncpy (chvalue, (In+3), 4);
 chvalue[4] = '\0';

 value.word = atoi (chvalue);
 outport (0x81, value.byte.lsb);
 outport (0x82, value.byte.msb);
 sprintf (out, "OK");
}

FlipRelay8 (char *In, char *Out) /* >1F51., relay 5, status 1*/
{                           /* 1=ON, 0=OFF */
 int board;
 board=7;

 /* check to see if relay board is alive */
 if(Poll_PBus_Node(Relay_Board_Addr(board))) sprintf(Out, "OK");
 else
 {
  sprintf(Out, "Error");
  return;
 }
 /* Set or clear relay */
 Set_PBus_Relay(board,*(In+3),*(In+4));
}

/*=====================================================================
This task generates the actuation using PID.
=====================================================================*/
indirect task1()
{
 static long ActPos, DesPos, DesVelo;
 static int DeltaPErr0, DeltaPErr1, DeltaPErr2;
 static long dpActuation, pActuation;
 static long ipActuation;
 static long DeltaPos;
 static int TimeIntervals;
 static float initialDesVelo;

#GLOBAL_INIT
 {
  ActPos = DesPos = DesVelo = 0L;
  DeltaPErr0 = DeltaPErr1 = DeltaPErr2 = 0;
  dpActuation = pActuation = 0L;
  ipActuation = DeltaPos = 0L;
 }

 gettimer (timeBeg);

 if (BeginMove == TRUE)
  {
   BeginMove = FALSE;
   DesPos = PointList[ExePointNumber].ListPosition;
   initialDesVelo = PointList[ExePointNumber].ListVelocity;
   pGainScale = PointList[ExePointNumber].ListpGainScale;
   iGainScale = PointList[ExePointNumber].ListiGainScale;

   if (UseVelocityControl)
```

```
 {
  if (ExePointNumber == PointNumber)
    VelocityControl = OFF;
  else
    VelocityControl = ON;
 }
   if (ExePointNumber++ >= PointNumber) ExePointNumber = PointNumber;
   DeltaPos = DesPos - ActPos;
   TimeIntervals = (int) ((labs(DeltaPos) * 22.2618) / initialDesVelo);
  }
else
  {
   DeltaPos = DesPos - ActPos;
  }

if (labs(DeltaPos) < 100)
  {
   PIOstatus.PIOAmode |= 0x1;    /* signal that we are done */
   outport (PIODA, PIOstatus.PIOAmode);
   takedata=OFF;
  }

if (Reset1)
  {
   DeltaPErr2 = DeltaPErr1 = DeltaPErr0 = 0;
   ipActuation = 0L;
   dpActuation = 0L;
   Reset1 = OFF;
  }

if (Control)
  {
   DeltaPErr2 = DeltaPErr1;  /* generate time history - 2 steps back */
   DeltaPErr1 = DeltaPErr0;
   DeltaPErr0 = DeltaPos;

   dpActuation = (long)( ( ((float)kpp*pGainScale)*
                          (float)(DeltaPErr0-DeltaPErr1)) +
                   ( ((float)kpi*iGainScale)*
                          (float)(DeltaPErr0)) +
                   ( (float)kpd*(float)(DeltaPErr0 -
                                (DeltaPErr1+DeltaPErr1) +
                                DeltaPErr2)) );
   ipActuation += dpActuation;

   if (VelocityControl)
{
 if (TimeIntervals == 0)
      {
    PseudoVelo = (long)((float)ipActuation * 0.001);
       BeginMove = TRUE;
      }
 else
   {
    PseudoVelo = (long)((float)DeltaPos * TimeIntervals * 22.2618);
    TimeIntervals--;
   }
}
  else
if (DesVelo != 0L)
 {
     pActuation = (long)((float)ipActuation * 0.001);

  if (pActuation > DesVelo)
       pActuation = DesVelo;
  else
    if (pActuation < -(DesVelo))
 pActuation = -DesVelo;
 }
```

```
   else
    PseudoVelo = (long)((float)ipActuation * 0.001);
   }
  else
   {
    pActuation = 0;
    ipActuation = dpActuation = 0;
   }
  gettimer (timeEnd);
}

/*=====================================================================
This task takes sensor data.
=====================================================================*/

indirect task2()
{
 if (takedata)
   {
    data[datapointer][0] = AbsPos;
    data[datapointer++][1] = Velocity;
    if (datapointer > 1000) datapointer = 1000;
   }

 if ((PIOstatus.PIOAmode & 0x2) == 0x0)
   PIOstatus.PIOAmode |= 0x2;
 else
   PIOstatus.PIOAmode &= 0xfd;

 outport (PIODA, PIOstatus.PIOAmode);
}

indirect task0()
{
 static long DesPos, pVelo;
 static long DeltaVErr0, DeltaVErr1, DeltaVErr2;
 static BYTEMODE RealActuation;
 static long dvActuation, vActuation;
 static unsigned int result0;
 static unsigned int result;
 static long dAbsPos[4], OldAbsPos;
 static int index, RollOver;
 static long dResult;

#GLOBAL_INIT
 {
   RollOver = 0;
   result = 0;
   result0 = 0;
   AbsPos = 0;
   dAbsPos[0] = 0;
   dAbsPos[1] = 0;
   dAbsPos[2] = 0;
   dAbsPos[3] = 0;
   DeltaVErr0 = 0;
   DeltaVErr1 = 0;
   DeltaVErr2 = 0;
   index = 0;
 }
/* gettimer (timeBeg); */
 OldAbsPos = AbsPos;
 result = ((inport (0x80) << 8) & 0xff00) | inport (0x81);
 dResult = (long) ((long)result - (long)result0);

 if (dResult > 32000)
   RollOver--;
 else
   if (-32000 > dResult)
  RollOver++;
```

```
  result0 = result;

  AbsPos = (long)result + (long)(RollOver * 65535);

  dAbsPos[index] = AbsPos - OldAbsPos;
  if ((index++) > 2) index = 0;
  Velocity = (long)((float)(dAbsPos[0] + dAbsPos[1] + dAbsPos[2]) * 42.67);

 /* if (labs(Velocity) > (MaxVelo + 500L)) Control = OFF; */
 /* if (AbsPos > AbsLimits) Control = OFF;  */

  pVelo = PseudoVelo; /* transfer variables to local storage */

  if (Reset2)
   {
    DeltaVErr2 = DeltaVErr1 = DeltaVErr0 = 0L;
    dvActuation = 0L;
    vActuation = 0L;
    Reset2 = OFF;
   }

  if (Control)
   {
    DeltaVErr2 = DeltaVErr1;
    DeltaVErr1 = DeltaVErr0;

    DeltaVErr0 = pVelo - Velocity;

    dvActuation = (long)(((kvp*(long)(DeltaVErr0-DeltaVErr1)) +
                  (kvi*(long)(DeltaVErr0)) +
                  (kvd*(long)(DeltaVErr0 -
                        (DeltaVErr1 + DeltaVErr1) +
                         DeltaVErr2))));
    vActuation += dvActuation;

    RealActuation.word = ((int)((float)vActuation * 0.0001));

    Actuation.word = RealActuation.word;

    RealActuation.word += D2AOFFSET;

    if (RealActuation.word > 4095)
      RealActuation.word = 4095;
    else
      if (RealActuation.word < 0)
        RealActuation.word = 0;

    outport (0x81, (char)(RealActuation.byte.lsb));
    outport (0x82, (char)(RealActuation.byte.msb));
   }
  else
   {
    dvActuation = vActuation = 0;
    RealActuation.word = D2AOFFSET;
   }
/* gettimer (timeEnd);    */
}


/*** Initialization Function ***/
int init_232()
{
 int i;
 int mode = 4;        /* 1 stop bit */
                  /* no parity */
                  /* 8 data bits */
                  /* even parity */
                  /* CTS/RTS disabled */
```

```
                /* set to 4 for CTS off */

 reload_vec(14,Dz0_circ_int); /* installs interrupt vector at runtime */
 Dreset_z0rbuf(); /* reset receive and transmit buffers */
 Dreset_z0tbuf();
 Dinit_z0(rbuf,tbuf,RBUFSIZE,TBUFSIZE,mode,IBAUD/1200,0,0);

 /* initialize the SMC */
 for(i=0;i<=15;i++)
 {
  sprintf(Out232,"%s%c",InitCmmnds[i].Cmmnd,CR);
  Dwrite_z0(Out232,InitCmmnds[i].ComLength);
 }
 ClearBuf(Out232,98);
 ClearBuf(In232,98);
 ReadDelay=IBAUD/2;
 lk_init_keypad();
 lk_setbeep(500);
}

int ClearBuf(char in[100], int cnt)
{
 static int index,count;

 count=atoi(cnt);
 for(index=0;index<count;index++) in[index] = '\0';
}

int Serial232Service(char *In, int ComLen, int RespLen)
{
 int err, cnt1;
 char cnt[5];

 ClearBuf(InString,40);
 if(ComLen) /* if writing */
 {
  if(Dwrite_z0(In,ComLen)); /* check for serial error */
  else return 0; /* write not successful */
  In[ComLen-1]='\0';
  In[ComLen]='\0';
 }
 if(RespLen) /* if reading */
 {
  if(ComLen) lk_tdelay(ReadDelay);
      /* delay if just wrote to smc */
  err=cnt1=0;
  do /* look for string length longer than one.  if looked more
      than 20 times, exit. */
  {
   if(cnt1>=20) /* took too long to get a response */
   {
    /*lk_setbeep(1000);
    lk_printf("Took too long\n");*/
    return 0;
   }
   if(err=Dread_z0(In,CR))
   {
    if(strlen(In)>1) err=1;
    else /* only carriage return in return */
    {
     err=0;
     /*lk_setbeep(300);
     lk_printf("Only CR\n");*/
    }
   }
   cnt1++;
  }
  while(err == 0);
 }
```

```
 return 1;
}

int EncoderService(char *Out, char mot[])
{
 unsigned int value;
 int i;
 char a[40],b[40],c[40],d[40];

 #GLOBAL_INIT
 {
  ClearBuf(a,40);
  ClearBuf(b,40);
  ClearBuf(c,40);
  ClearBuf(d,40);
 }

 sprintf(Out232,"W0%cY 200%c",mot,CR);
 if(Serial232Service(Out232, 9,0));
 else return 0;
 sprintf(Out232,"W0%cX%c",mot,CR);
 if(Serial232Service(Out232, 5,0));
 else return 0;
 lk_tdelay(ReadDelay);
 sprintf(Out232,"");
 if(Serial232Service(Out232, 0,8));
 else return 0;
 lk_printf("after first read\n");
 strncpy(a,(Out232+2),5);
 if(Serial232Service(Out232, 0,8));
 else return 0;
 lk_printf("after second read\n");
 strncpy(b,(Out232+2),5);
 if(Serial232Service(Out232, 0,8));
 else return 0;
 lk_printf("after third read\n");
 strncpy(c,(Out232+2),5);
 if(Serial232Service(Out232, 0,8));
 else return 0;
 lk_printf("after fourth read\n");
 strncpy(d,(Out232+2),5);
 value=CombineBits(a,b,c,d);
 sprintf(Out,"%u",value);
 lk_printf("E=%u\n",value);
 return 1;
}

/*** CombineBits *****************************************************
 This function uses the formula:
 value=(x>>(p+1-n))&~(~0<<n). It gets n bits from position
 p of x. For example, if (x,p,n)=(x,5,4), value would return
 the four bits in positions 5,4,3,2. */
/********************************************************************/
unsigned int CombineBits(char *a, char *b, char *c, char *d)
{
 unsigned int value, value1, value2, value3, value4;

 value1 = (unsigned)atoi(a);
 value2 = (unsigned)atoi(b);
 value3 = (unsigned)atoi(c);
 value4 = (unsigned)atoi(d);

 value1 = (value1 >> 2) & ~(~0 << 4);
 value2 = (value2 >> 2) & ~(~0 << 4);
 value3 = (value3 >> 2) & ~(~0 << 4);
 value4 = (value4 >> 2) & ~(~0 << 4);
 value2<<=4;
 value3<<=8;
 value4<<=12;
```

```
 value=value1|value2|value3|value4;
 return value;
}

TurnMotor (char *In, char *Out)
/*    V - >1T0VF010.
        Sets values F, R, or S on motor0
        >1T0VP10000.
        Sets P, A, or N on motor0, P = 10000
      C - >1T0CG.
        Sends a command to motor0
        G = Go
        I = Initialize
      L - >1T0LP11234987. Load Memory.
        Motor 0, Pos #1 = 1234, Vel #1 = 987
        >1T0LX 200.
        Motor 0, Execute memory at loc 200
        >1T0LH1R 100.
        Motor 0, Homing Routine command #1, R 100
      Q - >1T0QP.
        Queries for F,R,S,E,N */
{
 int i, j, value, ret, PointNum, length;
 static int index;
 long lvalue;
 char chvalue[10], mot, dir;

 #GLOBAL_INIT
 {
  for(i=0;i<50;i++) for(j=0;j<15;j++) HomingCommands[i][j]='\0';
 }
 ClearBuf(Out232,30); /* place null chars in Out232 */
 strncpy (chvalue, In+3, 1);
 chvalue[1]='\0';
 value = atoi(chvalue);
 if(value==0) mot = 'D';
 else if(value==1) mot = 'E';
 else mot = '0';

 switch (In[4])
 {
  case 'V':
   if((In[5]=='F')||(In[5]=='R')||(In[5]=='S'))
   { /* value changes of F,R,S */
    strncpy (chvalue, In+6, 3);
    chvalue[3] = '\0';
    value = atoi (chvalue);
    sprintf (Out232, "W0%c%c %d%c",mot,In[5],value,CR);
    Serial232Service (Out232, 9,0);
   }
   else if((In[5]=='P')||(In[5]=='A')||(In[5]=='N'))
   { /* value changes of P,A,N */
    strncpy (chvalue, In+6, 5);
    chvalue[5] = '\0';
    lvalue = atol (chvalue);
    sprintf (Out232, "W0%c%c %ld%c",mot,In[5],lvalue,CR);
    Serial232Service (Out232, 11,0);
   }
   break;
  case 'C': /* send a command to motor */
   switch (In[5])
   {
    case 'G':
     sprintf(Out232, "W0%cG%c",mot,CR);
     Serial232Service(Out232, 5,0);
     break;
    case 'I':
     init_232(); /* RS232 initialization routine */
```

```
      break;
     case 'R':
                                                          if (dir == '+') dir='-';
      else dir='+';
          sprintf(Out232, "W0%c%c%c",mot,dir,CR);
          Serial232Service(Out232, 5,0);
          break;
   }
  case 'L':
   switch (In[5])
   {
    case 'X':
      sprintf(Out232, "W0%cY 200%c",mot,CR);
      Serial232Service(Out232, 9,0);
      sprintf(Out232, "W0%cX%c",mot,CR);
      Serial232Service(Out232, 5,0);
      break;
    case 'P':
      sprintf(Out232, "W0%c0%c",mot,CR);
      Serial232Service(Out232, 5,0);
      sprintf(Out232, "W0%cQ%c",mot,CR);
      Serial232Service(Out232, 5,0);
      break;
    case 'H':
      strncpy (chvalue, In+6, 1);
      chvalue[1]='\0';
      PointNum = atoi(chvalue);
      if(PointNum==0) /* dump point list to smc */
      {
       sprintf(Out232, "W0%cY 10%c",mot,CR);
       Serial232Service(Out232, 8,0);
       sprintf(Out232, "W0%cE%c",mot,CR);
       Serial232Service(Out232, 5,0);
       for(i=1;i<=index;i++)
       {
        ClearBuf(Out232,30); /* place null chars in Out232 */
        sprintf(Out232, "W0%c%s\0%c",mot,
         HomingCommands[(PointNum-1)*15],CR);
        value=strlen(Out232);
        length=strlen(HomingCommands[(PointNum-1)*15])+4;
        lk_printf("%d,%d,",length,value);
        Serial232Service(Out232, length,0);
       }
       sprintf(Out232, "W0%c0%c",mot,CR);
       Serial232Service(Out232, 5,0);
       sprintf(Out232, "W0%cQ%c",mot,CR);
       Serial232Service(Out232, 5,0);
       return;
      }
      else
      {
       In[In[0]]='\0';
       In[strlen(In)-1]='\0';
       strcpy (HomingCommands[(PointNum-1)*15], In+7);
       lk_printf("%s,",HomingCommands[(PointNum-1)*15]);
       index=PointNum;
      }
      break;
    case 'E':
                                                          sprintf(Out232, "W0%cY 200%c",mot,CR);
      Serial232Service(Out232, 9,0);
      sprintf(Out232, "W0%cE%c",mot,CR);
      Serial232Service(Out232, 5,0);
      break;
   }
   break;
  case 'Q':
   if((In[5]=='F')||(In[5]=='R')||(In[5]=='S'))
   {
```

```
      sprintf(Out232, "W0%c? %c%c",mot,In[5],CR);
      Serial232Service(Out232, 7,7);
      lk_printf("%s\n",Out232);
     }
     if((In[5]=='P')||(In[5]=='N'))
     {
      sprintf(Out232, "W0%c? %c%c",mot,In[5],CR);
      Serial232Service(Out232, 7,10);
      lk_printf("%s\n",Out232);
     }
     if(In[5]=='E')
     {
      EncoderService(Out232,mot);
      lk_printf("E=%s\n",Out232);
     }
     break;
    }
   strncpy(Out,Out232,40);
  }
```

Appendix I

End Effector SBC Commands

The CY545 motion controller receives commands as a string of ASCII characters, of the form W0xy z, where "x" denotes the motor to receive the commands ("E" sends the command to motor controlling yaw, "D" sends the command to the motor controlling the pitch of the paint gun, and "0" sends the command to both motors), "y" denotes the command to be sent (a list of available commands appears in the CY545 users manual, but the most frequently used commands are listed below), and "z" is a number of either 8 bits or 24 bits, depending on the command (not all commands require a number following them, thus a value for "z" is not always used). For example, to move the paint gun motor to position 300 using previously defined values for speed, acceleration, and initial speed, the full command would be W0DP 300 (here "P" is the command used to send the motor to a specified position).

Commands sent to the motion controller may either be executed one at a time, or queued and written to the controller's EEPROM. The sending of commands to the motion controller is handled by the slave code. Each command issued by the slave code is of the form >1Twxyz, where "w" specifies the motor ("0" denotes motor "E", "1" denotes motor "D", and "2" denotes both motors), "x" specifies the type of command to be sent (the command types are as follows: "V" indicates a value to be sent (such as a change in motor speed, or a command to send the motor to a specific location), "C" indicates a command (such as the initialization routine, or a change in motor direction), "Q" queries the value of a defined parameter (speed, acceleration, position, etc.), and "L" indicates a command involving the loading of points into the EEPROM), "y" denotes the specific command to be sent (the commands are the same as those described above), and "z" is a numerical value of either 8- or 24-bits.

To load points into the EEPROM for subsequent execution, it is first necessary to initialize the motors to be used, with the command string >1TxCI, where "x" defines which motor is to be initialized. The values for "x" are described above. After initialization, the motion controller must be prepared to queue commands, i.e., it must be informed that the commands to followed are to be written to the EEPROM and not to be executed immediately. This is accomplished by the command string >1TxLP. Next, the commands to be queued are entered one at a time as detailed above. The commands may be parameter changes, positions to be moved to, or queries for parameter values, etc. When the commands to be queued have been entered, the controller must be informed that the queuing is complete. This is accomplished with the command string >1TxLX. After this command has been entered, the program will have been successfully written to the system's EEPROM, starting at value 200 in memory by default. In order to change the starting memory location, changes must be made to the slave code. To begin executing the commands thus written, the command string >1TxLX must be sent. The queued commands will then run in sequence until the end of the queue has been reached. At this point, the current memory location will be the location of the final executed command.

Example: To write a program for motor 1 that will set the speed value to 25, the acceleration to 30, the initial speed to 10, and then move the motor to position 1300, the following command strings would be sent:

>1T1CI	(initializes motor 1)
>1T1LP	(begins queuing sequence)
>1T1VR25	(sets speed of motor 1 to 25)
>1T1VS30	(sets acceleration of motor 1 to 30)
>1T1VF10	(sets initial speed of motor 1 to 10)
>1T1VP1300	(sends motor to position 1300)
>1T1LX	(ends queuing sequence)

To execute these commands, the following command string would be sent:

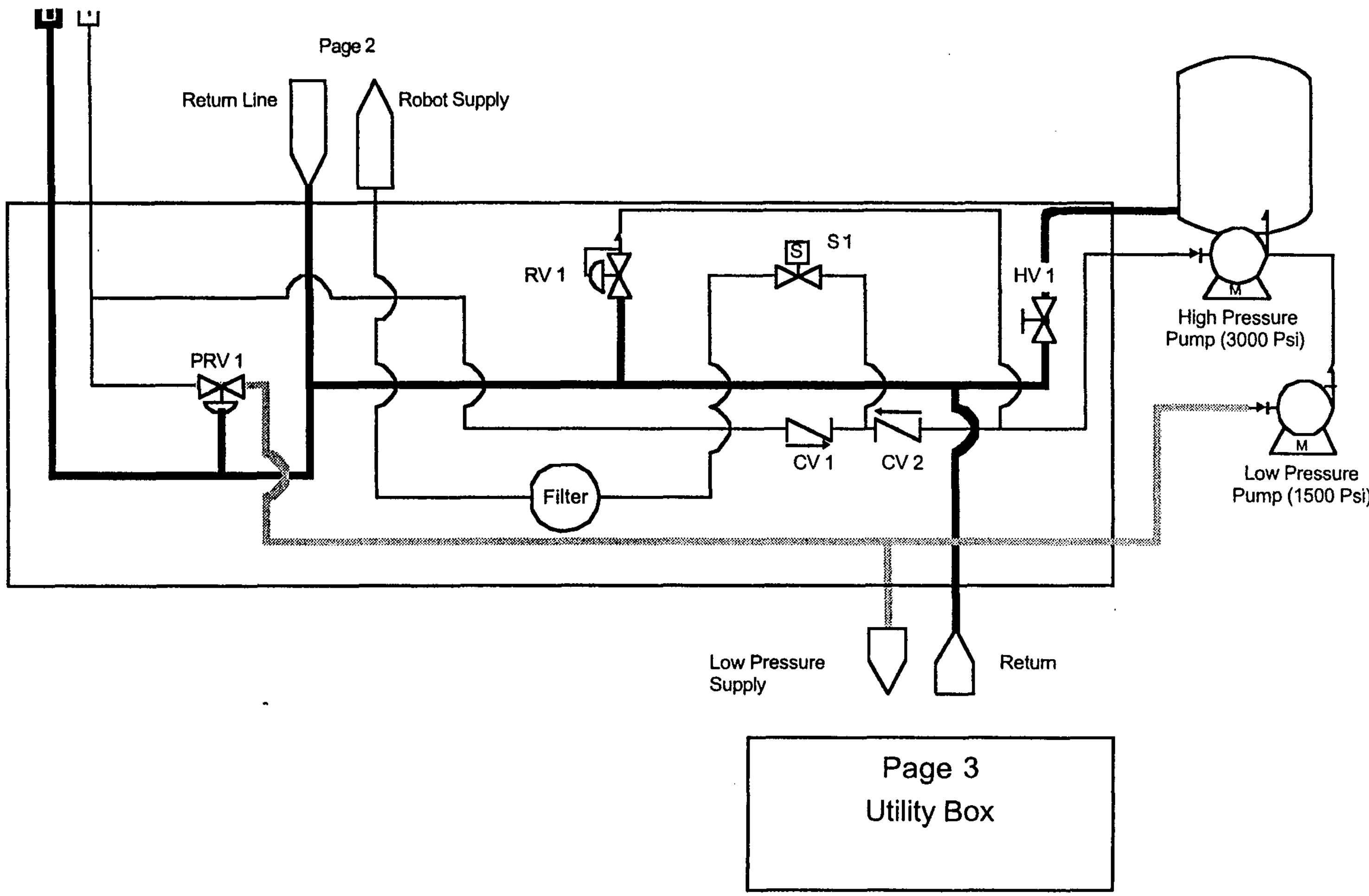
>1T1LX

After the commands had executed, the current memory location would be equal to that of the command >1T1VP1300.

Appendix J

Hydraulic System Schematic

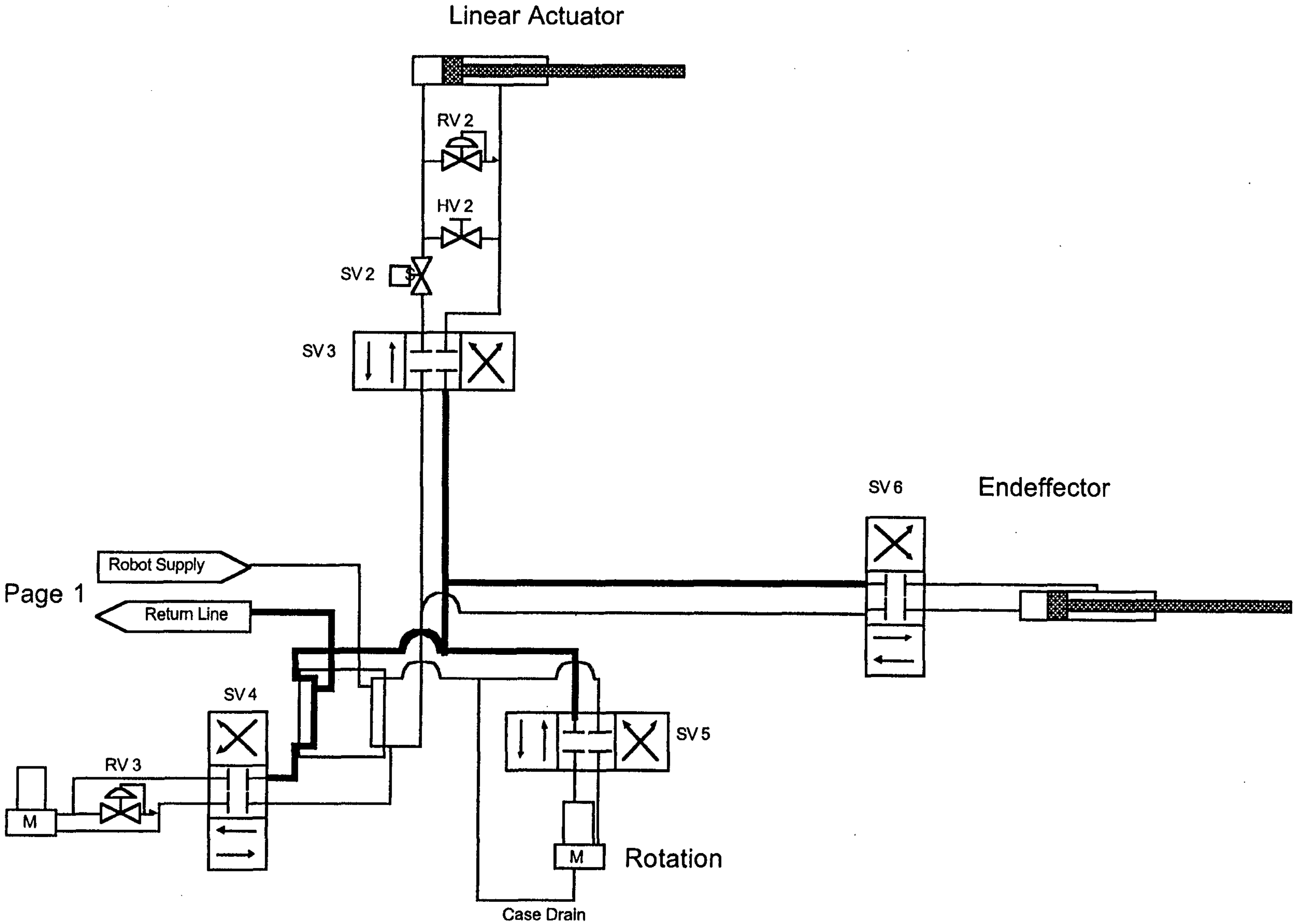
Hydraulic Schematic (Page 1 of 3)



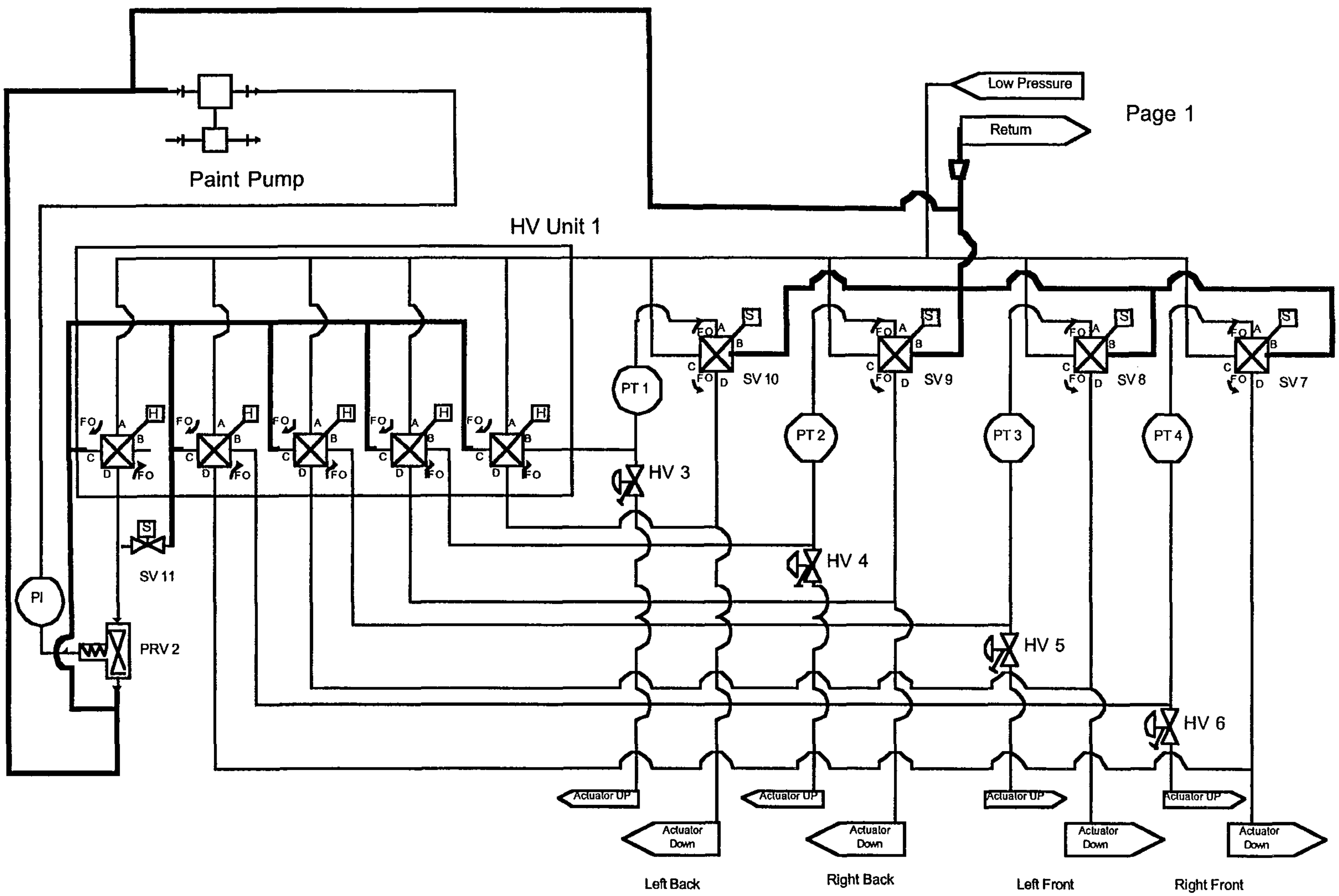
Abbreviations:

ACT	Actuator	RV	Return Valve
CV	Check Valve	SV	Solenoid Valve
HV	Hand Valve	PI	Pressure Indicator
PRV	Pressure Regulator Valve	PT	Pressure Transducer

Hydraulic Schematic (Page 2 of 3)



Hydraulic Schematic (Page 3 of 3)



Appendix K

Research Paper on
BASR Motion Planning Software

Control Structure of the U.C. Davis Stenciling Robot

B. Ravani and P.W. Wong
U.C. Davis, Davis, CA 95616

October 27, 1997

Abstract

At the University of California, Davis Advance Highway Maintenance and Construction Technology (AHMCT) Center, in conjunction with the California Department of Transportation (Caltrans), we have developed a very large pantograph-type robot. When a painting subsystem is mated to the robot's end-effector mounting plate, the robot system will be used to paint highway and roadway markings on the road surface. Each roadway marking consists of a message of eight (8) foot high alphanumeric characters, generally spanning the width (12 feet) of a traffic lane. This paper discusses the control system structure of the UCD/AHMCT robot, and introduces a unique way to decouple coupled kinematic motion in order to allow for simplied implementation of the robot controller.

1 Introduction

Painting roadway markings on the road surface is a tedious and hazardous maintenance procedure. To create the markings, a work crew first must section off a lane area and then layout a set of stencils corresponding to the desired message. Once everything is in place, the crew uses a paint sprayer and coats the road surface and stencils with paint. Where there are open spots in the stencil is where the paint is deposited on the surface. After a suitable drying period, the stencils are removed and the lane reopened. Each time this process is repeated, the crew is exposed to traffic hazards since the crew must leave the safety of their vehicles and work on the open roadway.

At the University of California, Davis Advanced Highway Maintenance and Construction Technology (AHMCT) Center, we have developed a very long reach pantograph-type robot (Figure 1) to accomplish the painting operations. When the unit is fully extended, it has a reach of almost 15 feet. The base can rotate approximately 270 degrees. One of the unique features of this design is that all the joint actuators are located at the base of the robot. This co-location

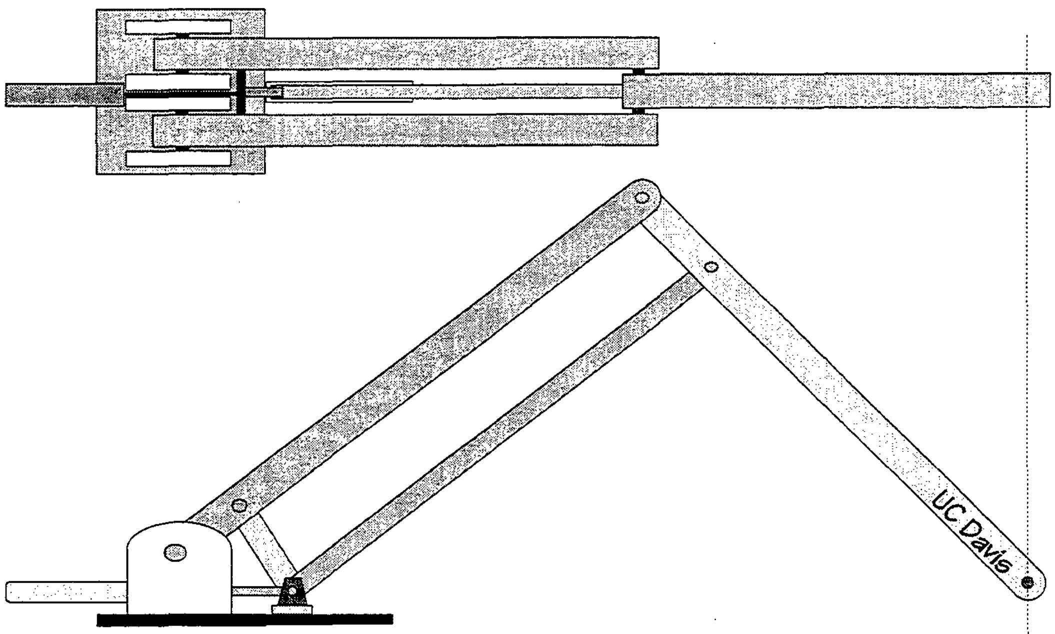


Figure 1: A Large Robot

leads to extremely high stiffness to weight ratios since the robot structure does not need to support the weight of the actuators.

As shown in Figure 2, the robot has two degrees of freedom: R and θ. The movement in the R direction is controlled by a linear hydraulic acutator, operating on the pivot of the pantograph. Motion is amplified by the pantograph mechanism according to an 8.3:1 ratio. Thus, for each inch the hydraulic actuator moves, the tool center point moves linearly 8.3 inches. Rotation of the robot arm is controlled by a hydraulic motor mounted in the base. Position of the arm is determined by two optical encoders mounted as shown in Figure 2. Note that the extension length of the arm is indirectly sensed through the rotation angle of the upper arm link.

In order to paint the roadway markings in a consistent fashion, the robot arm must move the tool center point (to which is mounted the painting mechanism) from point-to-point locations in an accurate way, as well as follow accurately a prescribed trajectory motion. The arm must follow a prescribed trajectory in order to create acceptable letter profiles with an evenly coated painted surface. The trajectory path planning is done in a separate software module that is discussed elsewhere.

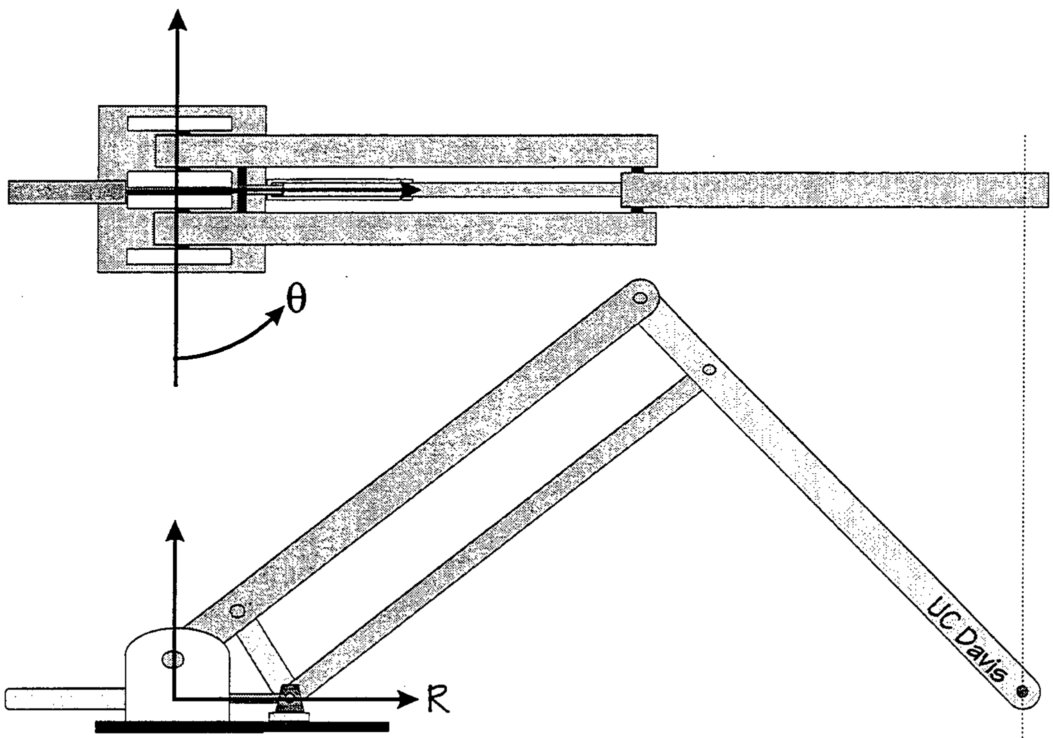


Figure 2: Degrees of Freedom

2 Arm Hardware Description

2.1 Sensor Configuration

The location of the tool center point is determined by two (2) rotary optical encoders. Each encoder is a quadrature type encoder capable of resolving forward and backward motion. The encoder resolution is 88,000 pulses per revolution or approximately 0.0041 degrees. One encoder is mounted on the shaft of the hydraulic motor and senses the rotation of the base of the arm. The other encoder is mounted on one joint to the pantograph parallelogram mechanism and senses the angle of the parallelogram. This angle is converted by software into a linear distance for the arm extension. Joint velocity is also computed by software using a finite difference approximation. Both encoders are interfaced with a custom designed encoder board. On this encoder board is an HP2020 which maintains an absolute encoder pulse count, and thus relieves the need for a computer or microcontroller to constantly monitor the encoders.

2.2 Kinematics

From Figure 3, the x, y coordinates of the tool center point is determined by the following:

$$x = Rcos(\theta)$$
$$y = Rsin(\theta)$$

where R is the length of extension and θ is the angle of rotation of the base of the arm.

Since trajectory control is necessary to create a consistently coated painted surface, accurate velocity control of motion is important. By taking time derivatives of the equations for the location of the tool center point and rearranging, we get the following:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} cos\theta & -Rsin\theta \\ sin\theta & Rcos\theta \end{bmatrix} \begin{bmatrix} \dot{R} \\ \dot{\theta} \end{bmatrix} \quad (1)$$

where the above $2x2$ matrix is known as the Jacobian.

From the above equations, it can be seen that in order to move the tool center point with a desired cartesian velocity ($\dot{x}$ and $\dot{y}$), the configuration of all the joints (θ and R) must be known to compute the necessary joint velocities ($\dot{\theta}$ and $\dot{R}$) to accomplish the desired trajectory. In other words, the joints are kinematically coupled together, since velocity profiles for individual joints cannot be computed independently of the other joint configuration information.

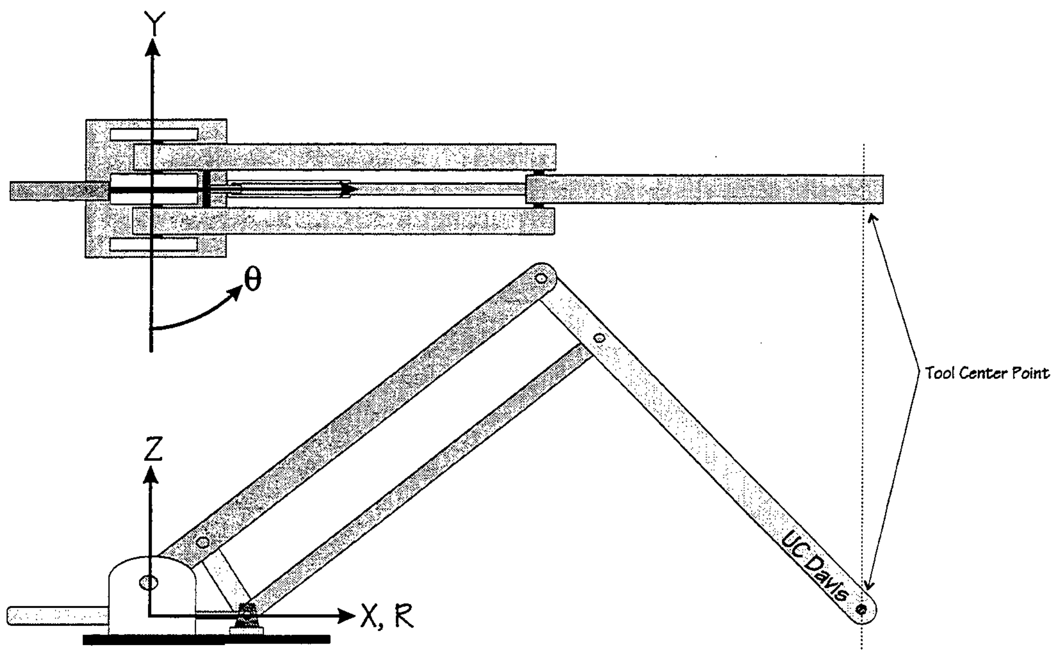


Figure 3: Arm Coordinate System

Figure Not Available.

Figure 4: Hydraulic System Schematic

2.3 Inverse Kinematics

By using the kinematic eqauations presented in Section 2.2, the inverse kinematics for the UC Davis arm mechanism can be derived. Dividing the equation for the y coordinate by the one for the x coordinate gives the following result:

$$\frac{y}{x} = tan(\theta) \tag{2}$$

or

$$\theta = tan^{-1}(\frac{y}{x}). \tag{3}$$

. From Figure 3, it can be seen that

$$R = \sqrt{(x^2 + y^2)}. \tag{4}$$

2.4 Hydraulic Control

Each arm joint motion is accomplished by using hydraulic motive power (Figure 4). For arm rotation, motion is accomplished by a hydraulic motor. For the arm extension, linear motion is created by using a hydraulic cylinder. The cylinder's motion is amplified by the pantograph mechanism. The hydraulic power to both the cylinder and the motor is controlled by a flow rate control hydraulic servo valve. By supplying electrical current to the servo valve, a directly proportional flow rate of hydraulic fluid is supplied from the hydraulic pump to the actuator. Electrical current to the servo valve is supplied from custom designed voltage-to-current (V-to-C) converters. The input to the V-to-C converter is a voltage signal from a computer controlled digital-to-analog (D-to-A) converter.

3 Arm Control System

The UC Davis arm control system is composed of a special microcontroller architecture. Instead of a uniprocessor computer control system, each joint is controlled by its own single-board microcontroller. Communiation and coordination between the joints is accomplished by low-cost and robust RS-485 serial communication local area networking.

3.1 Controller Arrangement

The UC Davis control system is a distributed processor system. However, instead of a dedicated joint processor residing in one main computer, each joint is independently controlled by its own microcontroller and signal conditioning board (Figure 5). Input/Output to the joint controllers is accomplished by a RS-485 Local Area Network (LAN) (see Section 3.2) The microcontrollers are from Z-World Engineering and are software programmable in the C language. The main CPU is from the Zilog Z-80 family and runs at 9.216 Mhz. The control software for each of the joints is contained on the microcontroller. Each of the signal conditioning boards are UC Davis custom designed and contain on-board a 12 bit digital-to-analog converter, an encoder control chip, and associated logic interface chips. Thus, to control the motion of the arm's tool center point, two (2) microcontrollers are required, one for each of the joints. Additional units are used to control the paint gun, as well as monitor the robot's hydraulic and electric subsystems. Furthermore, there is one microcontroller with a keypad that is used for user interaction and information display. Together with the joint controllers, a master/slave arrangement is formed. Commands are transferred from the keypad unit (master) to each of the joints (slave), which then execute the commands.

3.2 Networking

Master/slave controller communication is accomplished through the use of RS-485 serial communications (Figure 5). This protocol uses a two-wire interconnection scheme and has a maximum line length of approximately 5000 feet. Commands are transferred throughout the network at 19,200 bits per second. Full error correction and detection are provided on both the sending and the receiving side.

3.3 Communication Protocols

All joint controllers receive operating parameters through the RS-485 LAN interface. A command string of the form **>2A1090.** is sent through the network. The "2" in the command string is the address of the controller to which the command is directed. The letter "A" is the command to be executed, followed

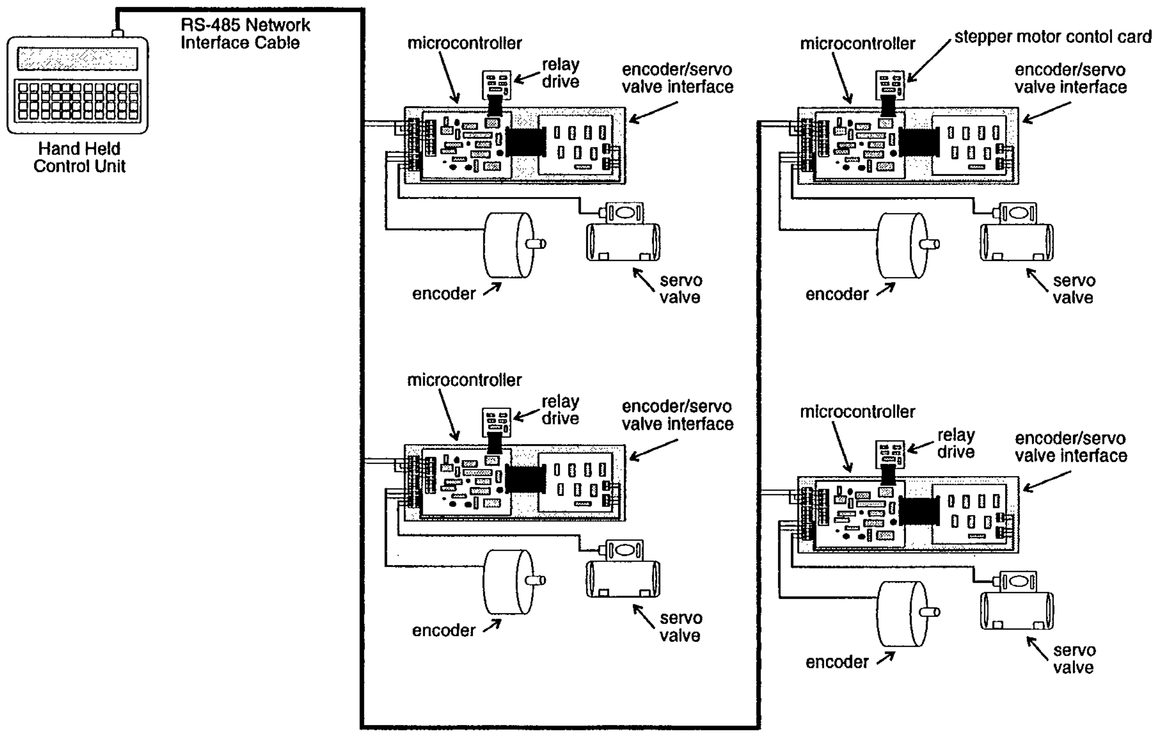


Figure 5: Control System Configuration

by "1090" which is the parameter for the "A" command. The "." marks the end of the command. Also appended automatically to the command string during network transmission is Error Correction Code (ECC) which informs the receiving controller of whether the command was corrupted in transit or not. If the command is corrupt, a retransmission is requested automatically. After the receiving controller has decoded and executed the command, it transmits the appropriate response back to the commanding unit. The response string generally looks like **>OK**. All operating parameters, from gain changes to relay driver status, is changable and obtainable through the network interface.

3.4 Control Software

Traditional robot path and trajectory planning generally involves the use of coordinate transformations and the Jacobian to transform Cartesian space coordinates (Figure 6, graph 1) and velocities (Figure 6, graph 2) into joint coordinates (Figure 6, graph 3) and velocities (Figure 6, section 4). The joint coordinates and velocities are then fed into the robot's joint controllers and executed.

However, due to the latencies involved in inter-joint communications over the LAN, the traditional techniques cannot be used on the UC Davis system. The reason is evident when the terms in the Jacobian are examined (Equation 1). In order to compute the output joint velocities, $\dot{\theta}$ and $\dot{R}$, from the input cartesian velocities, $\dot{x}$ and $\dot{y}$, both the θ and R values of the current configuration must be known in order to complete the calculation. The UC Davis *Motion Planner* uses a different approach. Instead of using the inverse kinematics and the Jacobian to compute the joint coordinates and velocities, respectively, the UCD joint software first uses inverse kinematics to compute the joint coordinates, then derives the necessary joint velocities by taking a backwards difference time derivative of the joint coordinates. Schematically, in Figure 6, the UCD software starts at graph 1, proceeds to graph 3 (by using inverse kinematics) and then arrives at graph 4 (by using a backwards difference approximation) (Figure 7).

The control system software necessary to operate the arm consists of three parts (Figure 8): the letter path generator, motion planner, and the control law itself. Due to the distributed computing nature of the UC Davis controller architecture, the different parts operate on different microcontrollers and coordinate their activities through the network. In addition to the software that directly controls the arm movement, fault tolerance and diagnostic software is also present on each joint microcontroller.

3.4.1 Letter Path Generator (LPG)

The Letter Path Generator (LPG) is a software module that runs on the handheld interface unit. This software is responsible for interpreting user inputs and generating the trajectory in cartesian space (x and y locations, as well as $\dot{x}$ and $\dot{y}$) that the arm's tool center point must move through in order to generate the

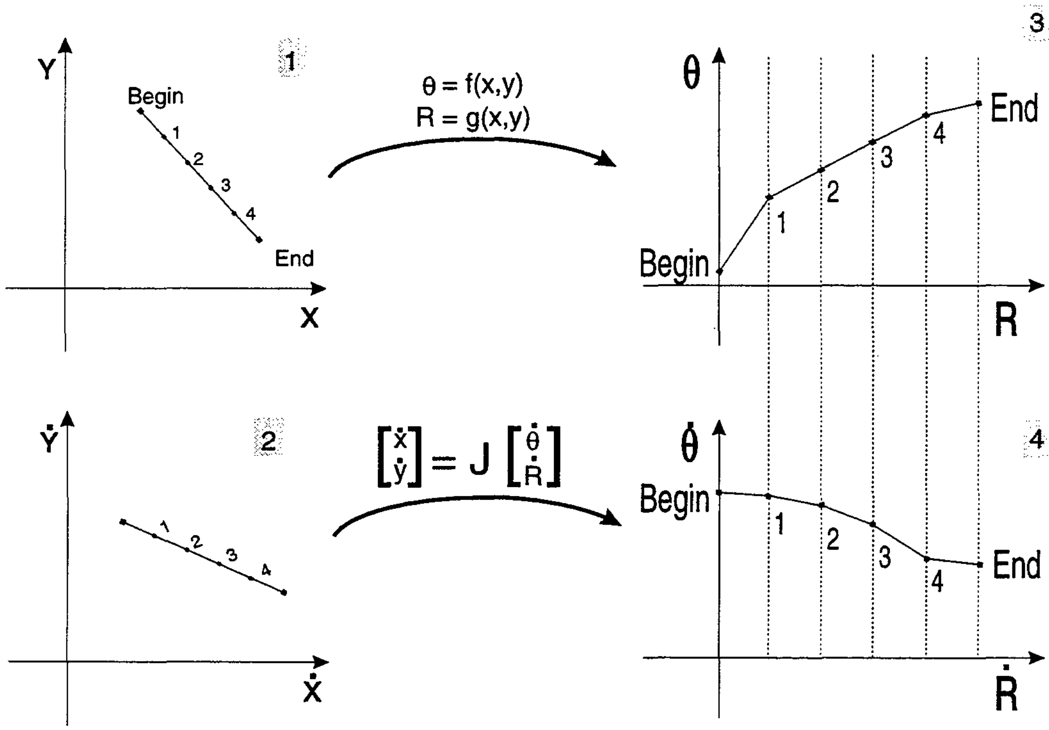


Figure 6: Cartesian to Joint Space Transformations

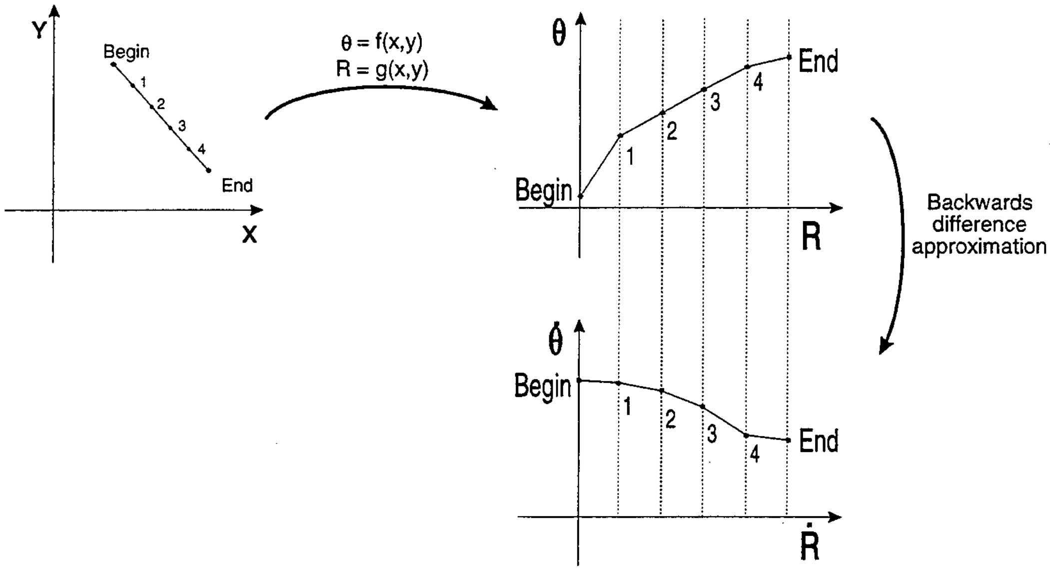


Figure 7: UC Davis Cartesian to Joint Space Transformations

desired letter shape. Once all the trajectory points have been computed, the x, y, $\dot{x}$, and $\dot{y}$ values are uploaded through the network to each of the joint microcontrollers.

3.4.2 Motion Planner

Once the joint microcontroller has received the trajectory points (Figure 9), the *Motion Planner* software on the joint controller comes into use. First, the *trajectory planner* calculates, based on the path length (s) between the trajectory points and the requested velocity (V), the cycle time for the interval. Using the cycle time for the interval, the *motion interpolator* then subdivides the path into straight line segments that correspond to a distance $\Delta s = Vx0.015625$. (Figure 10).

Since the control architecture does not allow inter-controller coummunications, individual joint microcontrollers do not have access to the other joints' configuration information, such as position and velocity. Thus, in order to complete the calculation of the inverse kinematics, the missing kinematic parameters must be estimated. The *kinematic parameter estimator (KPE)* module is used for this purpose. As the system runs, the *KPE* estimates the cartesian coordinates of the tool centerpoint. In effect, the system is generating an idealized trajectory profile for the tool centerpoint to follow. Once the estimated parameters are complete, the inverse kinematic transform (Section 2.3) equations are used to compute the joint coordinates. Depending on the joint microcontroller,

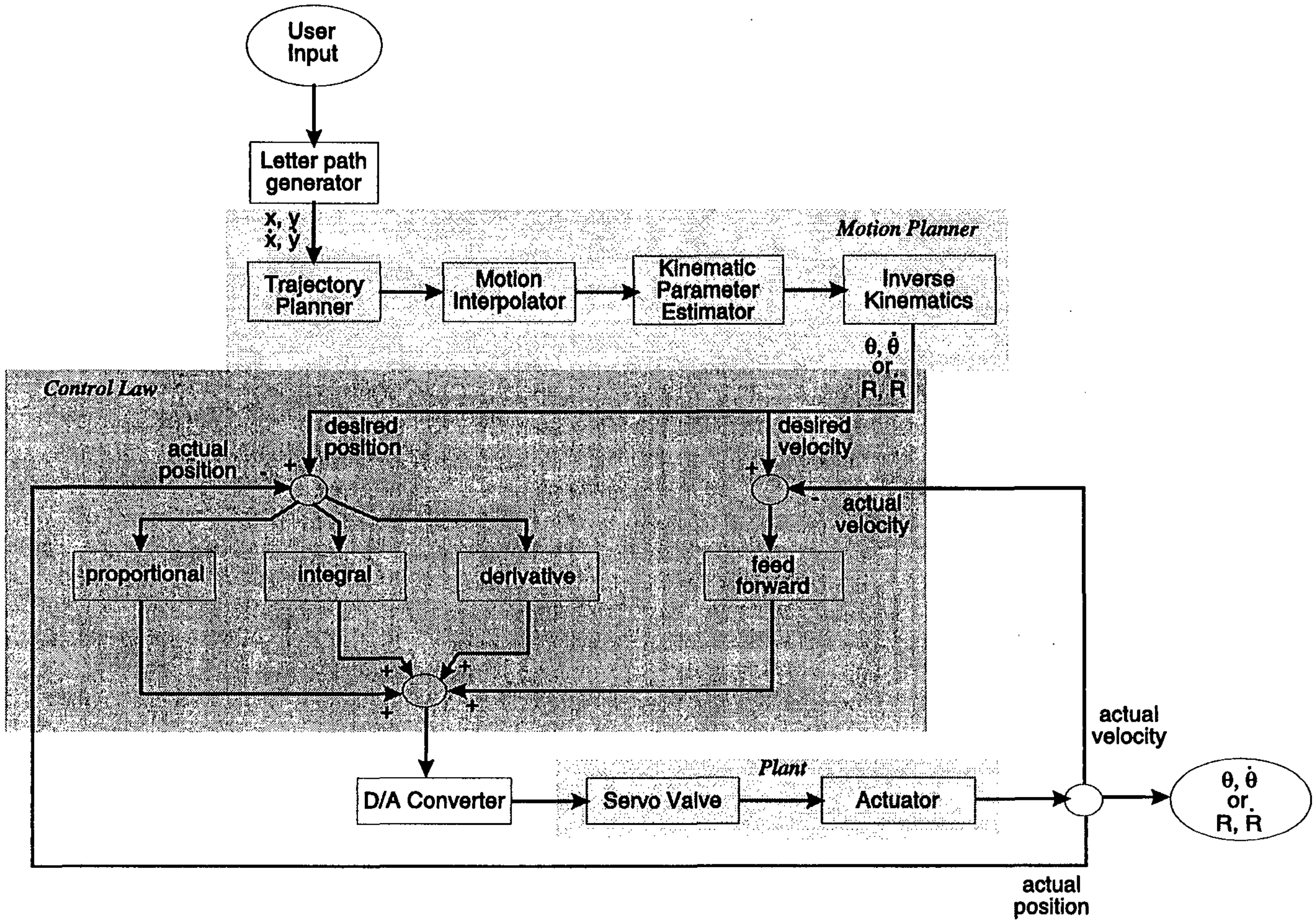


Figure 8: Control System Block Diagram

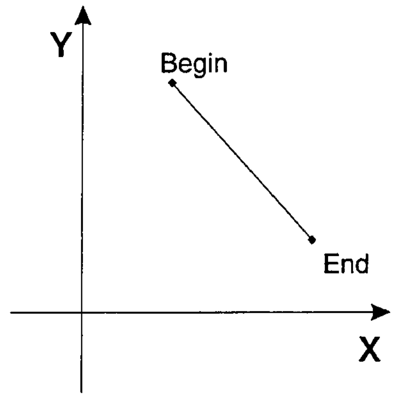


Figure 9: A Simple Two Point Path

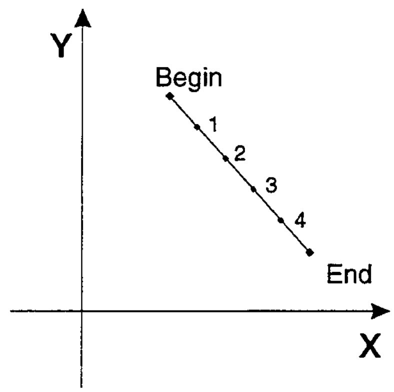


Figure 10: Path Subdivision by Motion Interpolator

either θ or R is utilized for control law input. In addition, $\dot{\theta}$ or $\dot{R}$ is computed as described earlier, and also utilized for control law input.

3.4.3 PID Control Structure

The control law used on all joint microcontrollers is the standard "velocity-form" of the descrete proportional-integral-derivative control law. However, for enhanced transcient startup and shutdown performance, the control law is modified by the addition of a velocity feedforward term. When high startup speed is required by the *motion planner*, this velocity feedforward term gives an additional actuator output boost to overcome the inertial and static friction effects within the arm mechanism. Additionally, the *motion planner* is designed so that near the end of the motion it inputs a large magnitude braking velocity

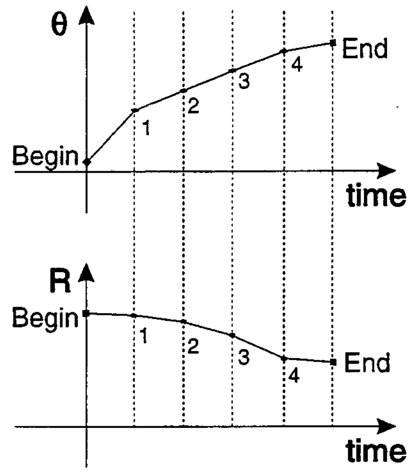


Figure 11: Joint Motion Profile After Using Estimator and Inverse Kinematics

to the control law to aid in deceleration of the arm mechanism.

3.5 Fault tolerance

Safety of the workers is paramount during operation of the robot. In addition to using engineering and design analysis to create a safe and reliable robot system design, system safety during adverse and unanticipated conditions must also be provided for. The UC Davis robot design incorporates four (4) different design features in order to create a safe design. The four design features are: power subsystem health monitoring, mechanical subsystem fault tolerance, electronic subsystem fault tolerance, and software fault tolerance.

3.5.1 Power Subsystem Health Monitoring

The UC Davis robot design incorporates a dedicated microcontroller whose sole purpose is to monitor the status of the power subsystem of the robot (Figure 12). Inputs to the microcontroller include pressure readings of the high pressure hydraulic system, the low pressure hydraulic system, differential hydraulic filter pressure, and system pneumatic pressure. Additionally, temperature readings of the engine coolant, paint heater and hydraulic oil tank are monitored by the microcontroller. Paint fluid levels are also reported to the microcontroller. If any parameter is out of specification, the microcontroller signals the hand-held control pendant that a fault condition exists. Using the RS-485 local area network, the hand controller then queries the power subsystem monitor for the fault that was detected. If the fault is serious, the hand controller then will initiate an auto-shutdown of the robot by broadcasting through the LAN that

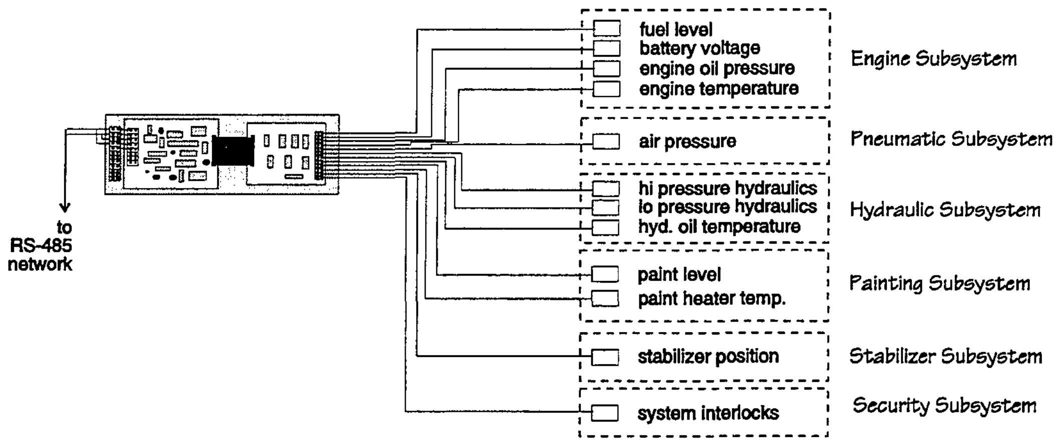


Figure 12: Health Monitoring

all joint controllers begin automatic shutdown procedures. The joint controllers will center the hydraulic servo valves, thereby stopping the flow of hydraulic oil to the robot actuators. Then the hand controller will close the main hydraulic supply valves. An informational message will flash on the display screen as to what the error condition was and what actions were taken. If the fault is not serious, but may indicate a future problem if preventive actions are not taken, a warning message will be displayed on the hand controller as a reminder to the operator that a condition exists that will provide degraded robot performance.

3.5.2 Mechanical Fault Tolerance

The UC Davis robot arm incorporates several mechanical design features to allow for fault tolerance in the event of any unanticipated mechanical failure. One of the major design rules is that all components be inactive or unpowered in the quiescent state and that a constantly supplied actuation signal be necessary to activitate the system. With the removal of the actuation signal, the system would revert back to its quiescent state. For example, although the arm links weigh several hundred pounds, the entire pantograph mechanism is gravity compensated with a counterbalance mechanism (Figure 13). With this counterbalance system, the mechanism cannot extend or retract by itself in the event that actuating force is lost due to a hydraulic hose rupture or other actuator failure. As a further backup to the counterbalance (FIgure 14), the hydraulic input to the actuator ports are closed automatically by a self-closing valve when electric power is lost. Thus, the self-closing valve traps the hydraulic oil within the actuator and locks its last position. Furthermore, the main hydraulic supply line is controlled by a spring loaded normally closed control valve. Thus, with no electric power applied to the valve, no hydraulic power is supplied to the

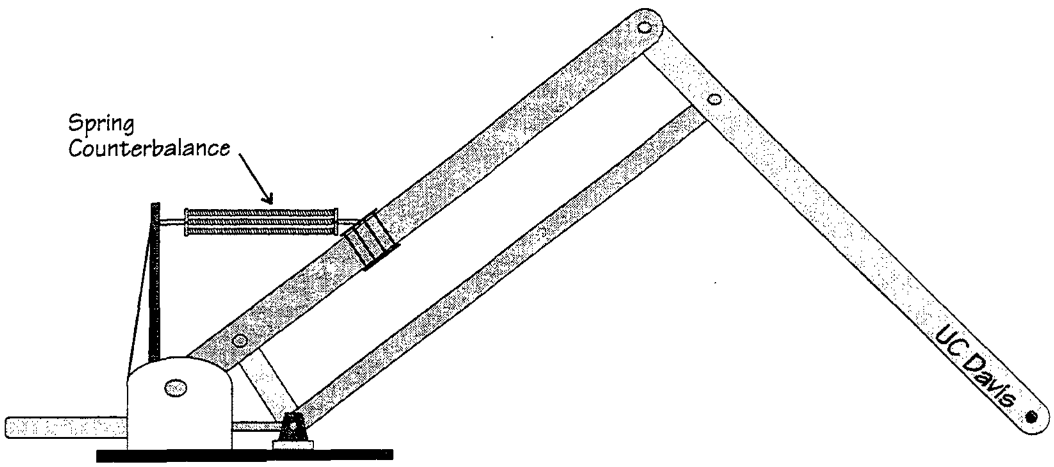


Figure 13: Mechanical Counterbalancing

system. In the event of a power failure for whatever reason to the valve, the hydraulic power will be shutoff automatically. Hydraulic relief valves are also located throughout the system. Should any hydraulic component experience any overpressure, rather than damage the component, the relief valves would open and relieve the pressure.

3.5.3 Electronic Fault Tolerance

All of the microcontrollers of the UC Davis robot are equipped with electronic fault tolerance features. The microcontroller power supplies have built in low power and power fail detection circuitry. If input power reaches a certain threshold, the detection circuitry notifies the microcontroller's main CPU. Once notified, the CPU has approximately 100 microseconds of residual power. During this time, the CPU executes a quick shutdown procedure to safe and secure the robot. This procedure involves closing the main hydraulic supply valve and centering the joint servo valves to stop the flow of hydraulic fluid to the actuators. If enough power remains, the microcontroller CPU attempts to continue to monitor the power supply to see if main power has returned to normal. If main power does return, the network command interface will refuse to execute any commands and return to the hand controller a message that a power failure or transcient had occured. If main power does not return to normal by the time the residual power has run out, when main power does eventually return, the microcontroller will re-initialize itself and become ready to accept commands in a normal fashion.

In addition to power supply monitoring, each microcontroller has independent circuitry to monitor the main CPU performance. On-board each microcon-

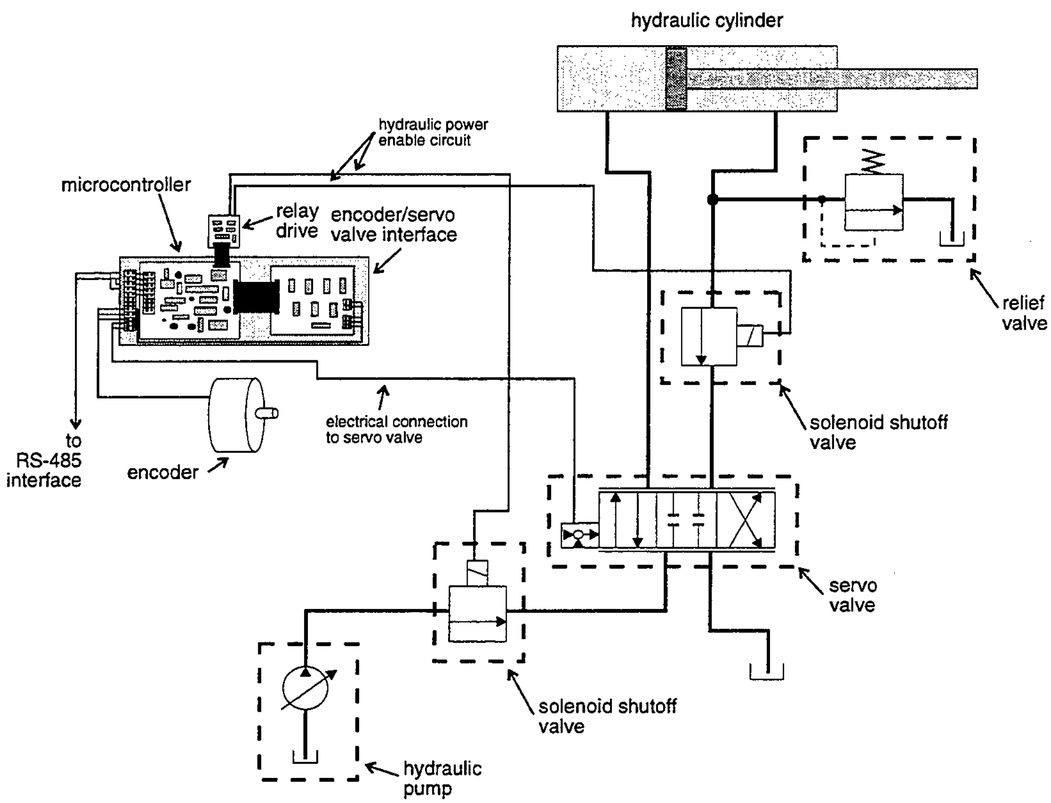


Figure 14: Hydraulic System Shutoff Valves

troller is a special "watch-dog" timer that must be reset every 2.6 milliseconds by special CPU instructions. Should the CPU be incapicitated or disabled due to memory faults, electrical interference, radiation, or other effects, the "watch-dog" will time-out, causing the CPU to be reset. Once the system has been reinitialized, it initiates an automatic shutdown of all hydraulic power. The network command interface will refuse to execute any commands, and return to the hand controller a message that a software induced reset had occured.

3.5.4 Software Fault Tolerance

The last feature for safe operation of the UC Davis robot incorporates software fault tolerance in an effort to guard against human error in coding software. Included with the control software are special software routines called "exeception handlers" that guard against unusual or abnormal software conditions. These conditions include invalid parameters to function calls or random access memory corrupted by stray memory pointers, radiation, or electromagnetic effects. Examples of invalid parameters to function calls are requesting the arc cosine of a value greater than 1.0, or division by zero. These errors occur, however infrequently, since in a real time system, it is impossible beforehand to analyse completely all the operating conditions the software may encounter. Thus, when any of the abnormal software conditions exist and the "exeception handler" is called, the system shuts down the the hydraulic power, and automatically reinitializes the system. The network command interface will refuse to execute any commands, and returns to the hand controller a message indicating a software fault had occured.

4 Conclusions

Due to careful design considerations, the AHMCT center at UC Davis has created a robust and low cost robotic stenciling system. In addition, highway worker safety is greatly increased since the workers are no longer exposed to hazardous situations on the open road.