California AHMCT Research Center
University of California at Davis
California Department of Transportation

THE PREMARK MAKER USER'S MANUAL, VERSION 1.2

Phillip W. Wong

AHMCT Research Report
UCD-ARR-96-05-10-01

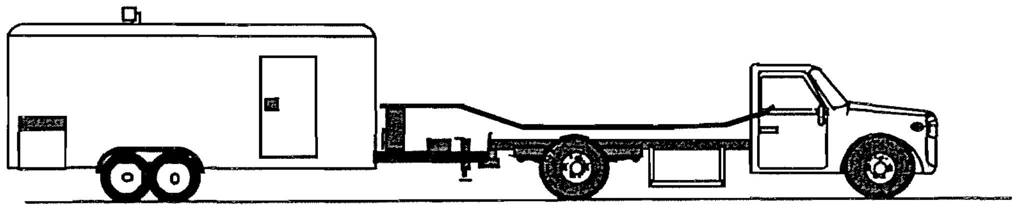
Final Report of Contract
IA 65Q168 (MOU 94-11)

May 10, 1996

* This work was supported by the Strategic Highway Research Program and the California Department of Transportation (Caltrans) Advanced Highway Maintenance and Construction Technology Center at UC, Davis

The Premark Maker

User's Manual
Version 1.2
May 10, 1996



Advanced Highway Maintenance and Construction Technologies Center
Department of Mechanical Engineering
University of California, Davis
Davis, CA 95616
Technical Support: (916) 752-4180

Table of Contents

READ CAREFULLY BEFORE ATTEMPTING TO ASSEMBLE, INSTALL, OPERATE OR MAINTAIN THE PRODUCT DESCRIBED. PROTECT YOURSELF AND OTHERS BY OBSERVING ALL SAFETY INFORMATION. FAILURE TO COMPLY WITH INSTRUCTIONS COULD RESULT IN PERSONAL INJURY AND/OR PROPERTY DAMAGE! RETAIN INSTRUCTIONS FOR FUTURE REFERENCE.

0. SAFETY INFORMATION

WARNING

Explosive Fuel
can cause fires and severe burns.

Stop engine before filling fuel tank.

Explosive Fuel!
Gasoline is extremely flammable and its vapors can explode if ignited. Store gasoline only in approved containers, in well-ventilated, unoccupied buildings, away from sparks or flames. Do not fill the fuel tank while the engine is hot or running, since spilled fuel could ignite if it comes in contact with hot parts or sparks from ignition. Do not start the engine near spilled fuel; wipe up spills immediately. Never use gasoline as a cleaning agent.

WARNING

Carbon Monoxide
can cause severe nausea, fainting or death.

Do not operate engine in closed or confined area.

Lethal Exhaust Gases!
Engine exhaust gases contain poisonous carbon monoxide. Carbon monoxide is odorless, colorless, and can cause death if inhaled. Avoid inhaling exhaust fumes, and never run the engine in a closed building or confined area.

WARNING

Rotating Parts
can cause severe injury.

Stay away while engine is in operation.

Rotating Parts!
Keep hands, feet, hair, and clothing away from all moving parts to prevent injury. Never operate the engine with covers, shrouds, or guards removed.

WARNING

Sulfuric Acid in batteries can cause severe injury or death.

Charge only in good ventilation.
Keep sources of ignition away.

Dangerous Acid, Explosive Gases!
Batteries contain sulfuric acid. To prevent acid burns, avoid contact with skin, eyes, and clothing. Batteries produce explosive hydrogen gas while being charged. To prevent a fire or explosion, charge batteries only in well ventilated areas. Keep sparks, open flames, and other sources of ignition away from the battery at all times. Keep batteries out of the reach of children. Remove all jewelry when servicing batteries.

Before disconnecting the negative ground cable, make sure all switches are OFF. If ON, a spark will occur at the ground cable terminal which *could cause an explosion if hydrogen gas or gasoline vapors are present.*

WARNING

Hot Parts
can cause severe burns.

Do not touch engine while operating or just after stopping.

Hot Parts!
The crankcase, cylinder head, exhaust system, and other components can get extremely hot from operation. To prevent severe burns, do not touch these areas while the engine is running or immediately after it is turned off. Never operate the engines with heat shields or guards removed.

WARNING

Accidental Starts
can cause severe injury or death.

Disconnect and ground spark plug lead before servicing.

Accidental Starts!
Before servicing the engine or equipment, always disconnect the spark plug lead to prevent the engine from starting accidentally. Ground the lead to prevent sparks that could cause fires.

On engines equipped with a 12-volt battery and/or electric start, disconnect the battery cables from the battery. Always disconnect the negative (-) cable first.

Before disconnecting the negative ground cable, make sure all switches are OFF. If ON, a spark will occur at the ground cable terminal which could cause an explosion if hydrogen gas or gasoline vapors are present.

High Voltage!
Never touch electrical wires or components while the engine is running. They can be sources of electrical shock which could cause severe injury or burns.

Overspeed is Hazardous!
Never tamper with the governor components or settings to increase the maximum speed. Severe personal injury and damage to the engine or equipment can result if operated at speed above maximum.

Engine Maintenance!
Do not attempt to crank or start engine until it has been properly serviced with recommended motor oil. Any attempt to crank or start the engine before it has been properly servicing with the recommended oil will result in engine failure.

Release Air pressure before Maintenance!
Before servicing the compressor package, make sure power source has been turned off and system air pressure has been released.

Do not Adjust Pressure Control Components!
Do not attempt to change the settings on control components. Pressure switch and pilot valve settings are preset at the factory for normal operating conditions. Altering the settings will result in compressor and motor damage.

1. System Overview and Description

1.1 Introduction

The UC Davis/CalTrans Premark Painting Trailer is an integrated system that can paint 2 different types of aerial surveying marks. The Trailer is self-contained with onboard power and painting systems. Figure 1.1 shows the basic layout of the Painting Trailer. All functions, except engine start and shutdown are accessible from the hand-held control unit.

1.2 Power Systems (Figure 1.2)

Electrical power is provided to the Trailer by a 6 kilowatt, Miller welder/generator. This generator provides electric service for the gantry robot and its controller. A 24 volt DC power transformer provides power for the air solenoids. Pneumatic power is provided by a 12HP 30 gallon Speedaire compressor. Pneumatic power is used for the paint pumps, as well as for all the actuators in the Trailer.

1.3 Control Systems

The entire system is controlled from the Hand-held Control Unit (Figure 1.3). This unit controls the gantry robot motion and activates all the actuators and solenoids necessary for operation. Interface boards are located on the top shelf of the Control cabinet (Figure 1.5-a). The gantry robot interface unit is also in the control cabinet on the third shelf (Figure 1.5-c). Manual controls to actuate all systems are located on a control panel on the second shelf of the control cabinet (Figure 1.5-b). Power supplies for all interface units are located on the bottom of the control cabinet (Figure 1.5-d).

1.3.1 Controller Software

The Hand Control Unit controls the Gantry in 3 different modes: 1) Automatic, 2) Incremental, and 3) Continuous.

In Automatic mode, the Gantry will paint a premark or a test mark. This mode is accessible by pressing the 'Auto Paint Sys' button to bring up the menu, and pressing 'F1' to activate the automatic premark painting mode or pressing 'F2' to activate the automatic test mark painting mode.

In Incremental mode, the gantry will move a preset distance in the direction chosen by the operator. To use this mode, choose gantry functions by pressing the 'Gant Funct/LFN Funct' button. Press the 'Jog' button to toggle between Continuous and Incremental mode. When the mode indicator (Figure 1.4) shows 'M:M', the unit is in Incremental mode. When the mode indicator shows "M:J", the unit is in Continuous mode. Pressing '+' or '-' increases or decreases the movement increment. Next, selection of the direction initiates the movement of the gantry. The orientation of the keypad is such that the keypad should be placed parallel to the ground, with arrow on the '2' key pointing to the back of the trailer.

In Continuous mode, the gantry will move in the direction chosen, until the 'Stop' key is pressed. The keypad orientation is as described above.

Pressing 'All Stop' will result in a software controlled emergency stop. If a more drastic emergency stop is required, the left red switch on the top of the hand unit should be pressed.

1.4 Paint Systems

Paint is pumped using 2 Binks pneumatic paint pumps. One each is provided to pump black and white paint. In addition, located on each paint can cover, are paint mixers to keep the paint well mixed. The mixers operation is controlled from the Hand Controller.

1.5 Premark Systems

A gantry robot is used to paint the large X-style premark (Figure 1.6). A custom designed Linear Free Nozzle system (Figure 1.7) is used to paint abbreviated premarks. All operations are fully automatic and are controlled from the Hand Controller.

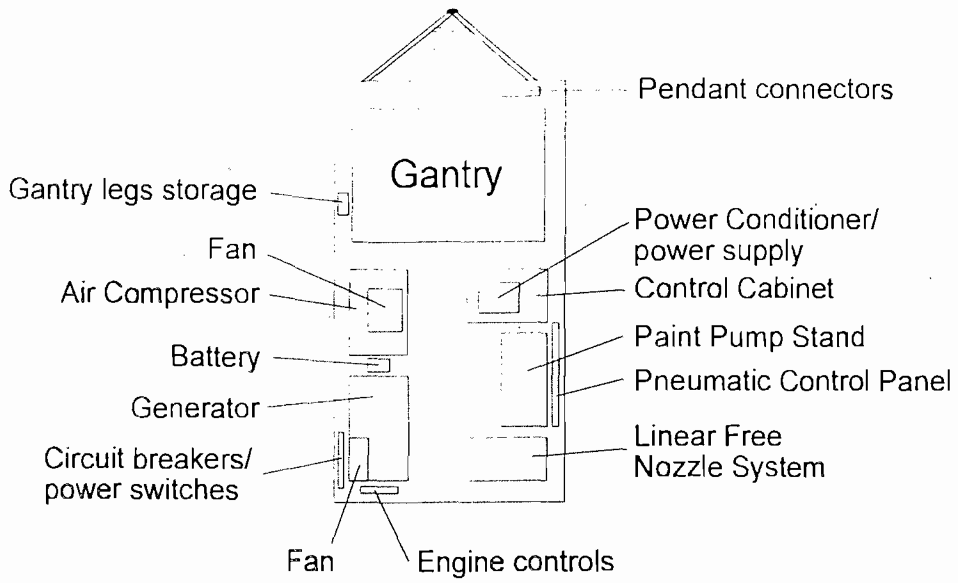


Figure 1.1: Trailer Layout

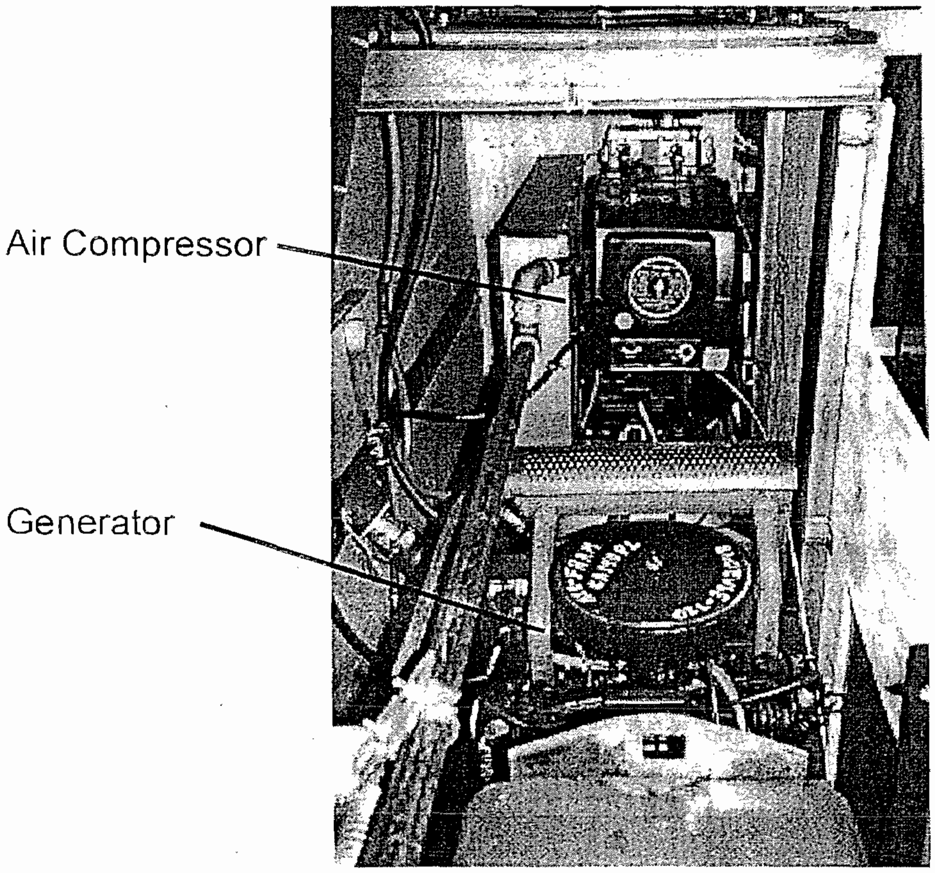


Figure 1.2: Power Systems

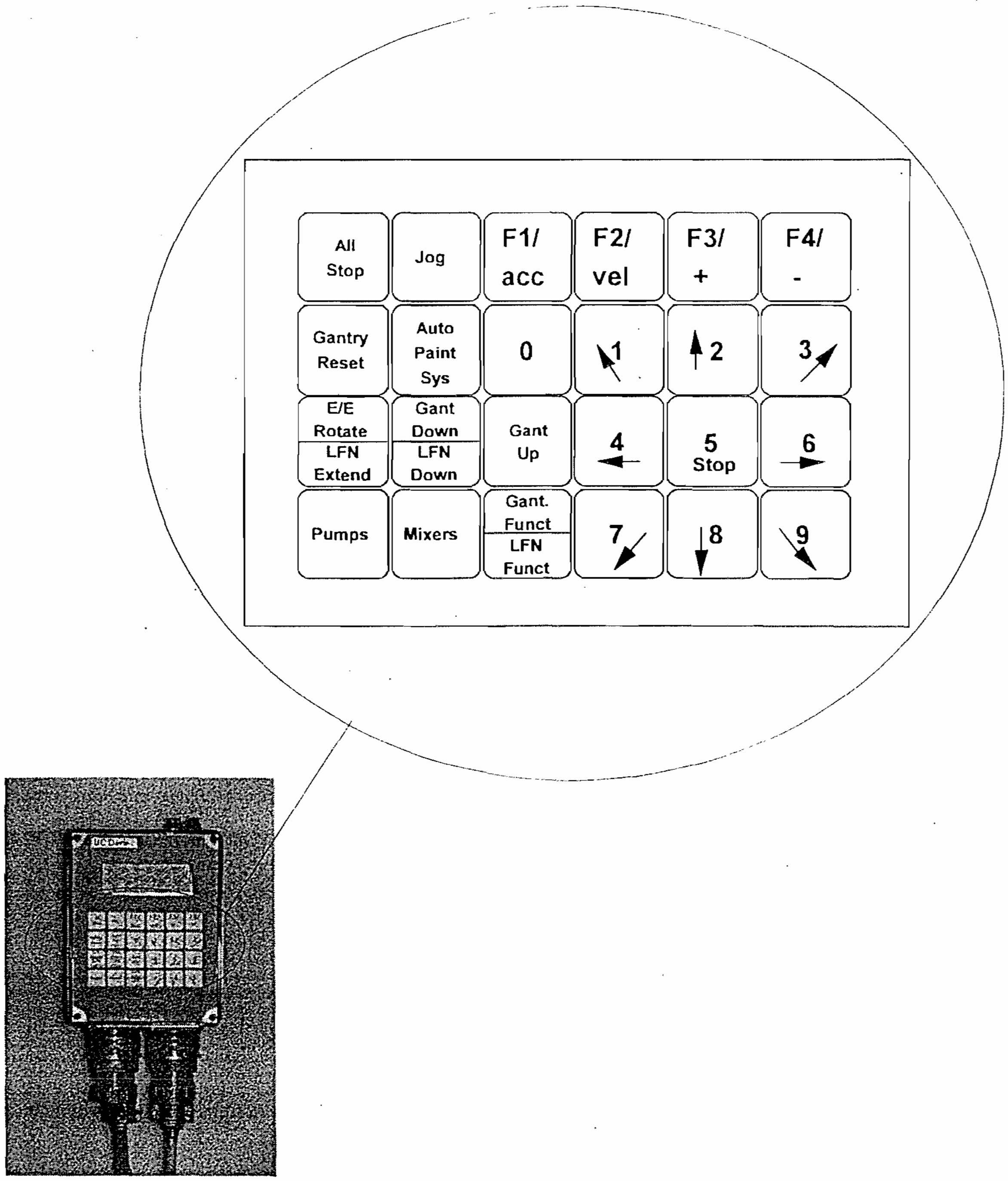
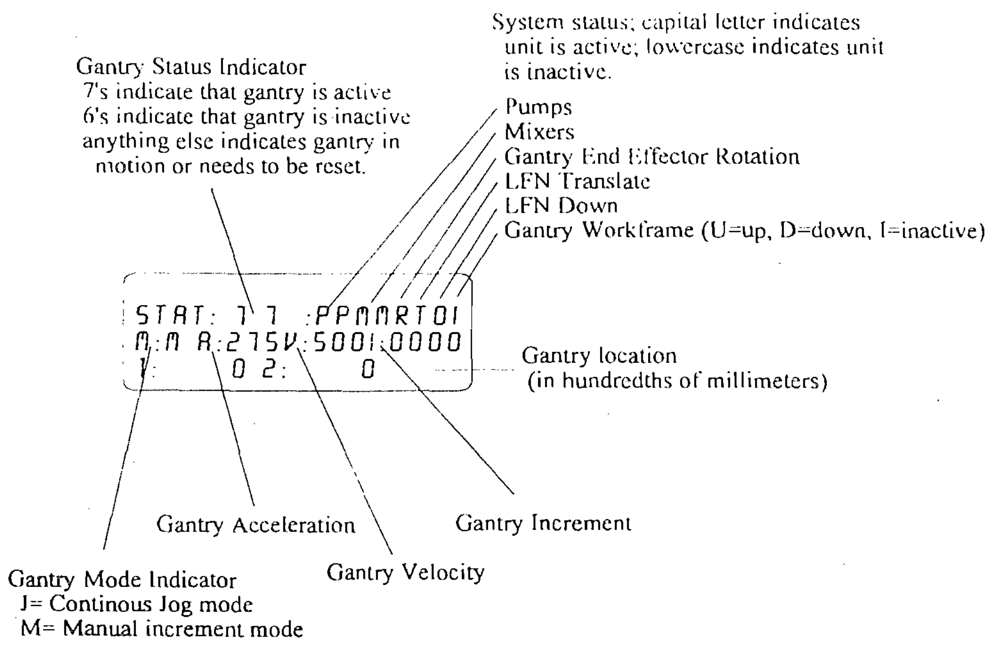
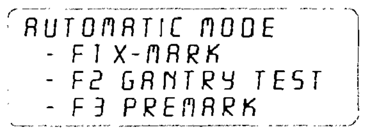


Figure 1.3: Hand Controller


System status; capital letter indicates unit is active; lowercase indicates unit is inactive.
Pumps
Mixers
Gantry End Effector Rotation
LFN Translate
LFN Down
Gantry Workframe (U=up, D=down, I=inactive)
Gantry Status Indicator
7's indicate that gantry is active
6's indicate that gantry is inactive
anything else indicates gantry in motion or needs to be reset.
STAT: 7 7 :PPMMRTOI
M:M A:275V:5001:0000
1: 0 2: 0
Gantry location
(in hundredths of millimeters)
Gantry Acceleration
Gantry Increment
Gantry Velocity
Gantry Mode Indicator
J= Continous Jog mode
M= Manual increment mode

Status Display



Automatic Menu

Figure 1.4: LCD Status Display

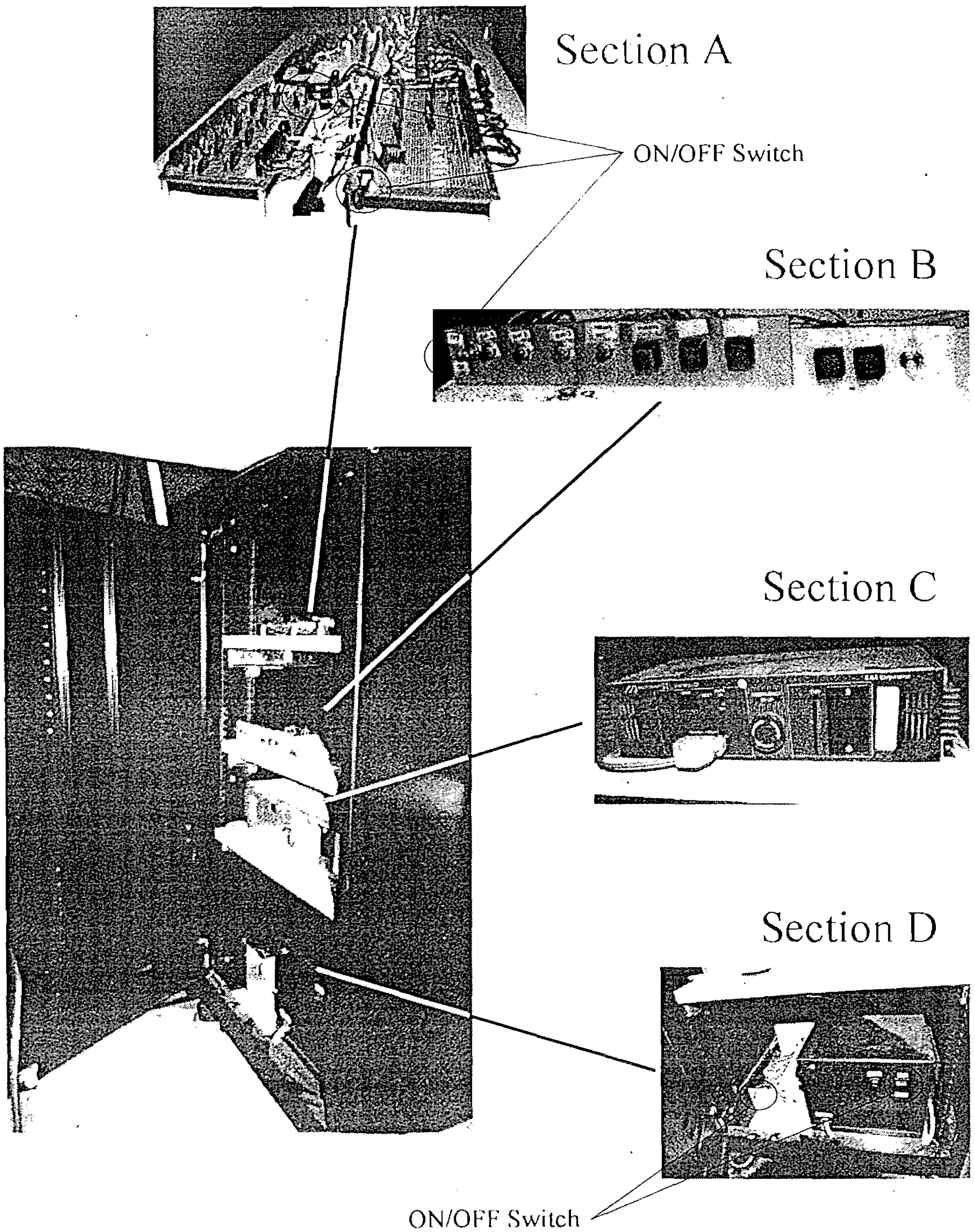


Figure 1.5: Controller Cabinet

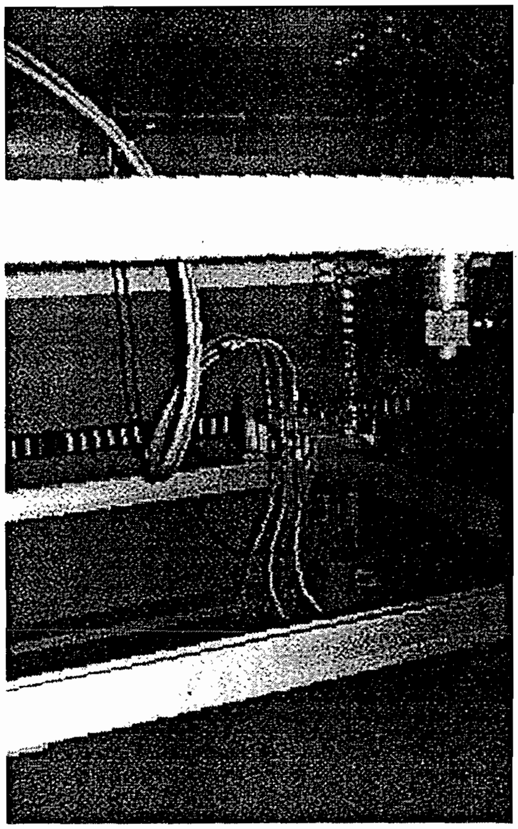


Figure 1.6: Gantry Robot

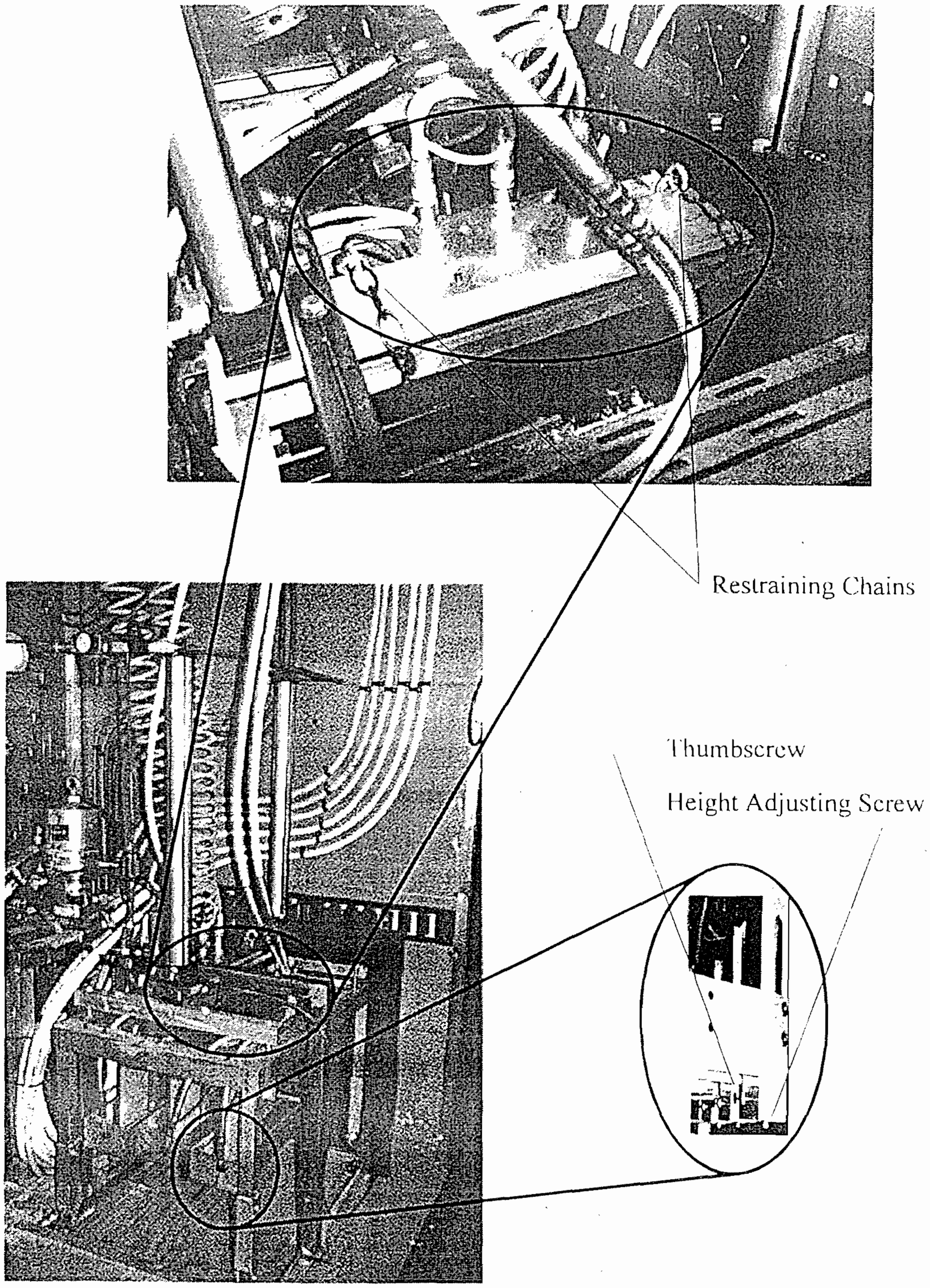


Figure 1 7: Linear Free Nozzle System

2. System Startup

2.1 Introduction

This section describes the startup procedures common to the operation of the Gantry Free Nozzle system and the Linear Free Nozzle system. To reduce the startup time, the operations described here can be accomplished in parallel with multiple workers. Please refer to Figure 1.1 for the location of equipment.

2.2 Preparing the Trailer for work

2.2.1 Open all doors. Be sure to secure all doors with the restraining latch to prevent inadvertent movement of doors.

2.2.2 If the gantry work frame legs are installed, remove them and place them in the leg storage container at the side of the trailer.

2.2.3 Unlatch floor covers. There are two slide latches on the aft edges of the floor covers.

2.2.4 Remove the left (facing forward) floor cover first. The cover is snug and form fitting and may need maneuvering to remove. Place the cover on the ground when removed from the frame.

2.2.5 Remove the right floor cover. The cover is snug and form fitting and may need maneuvering to remove. Place the cover on the ground when removed from the frame.

2.2.6 Place the removed covers in the back of the tow vehicle. If necessary, the covers maybe left at the maintenance station.

2.2.7 If gantry work frame legs are not installed, remove three (3) legs from the storage container and install them on the work frame. There should be 1 leg on the forward side and 2 legs on the aft side of the workframe. The leg receptacles are on the underside of the work frame. (Figure 2.2.1)

2.2.8 Check the oil level in the generator engine and the air compressor. (Figure 2.2.2a and 2.2.2b)

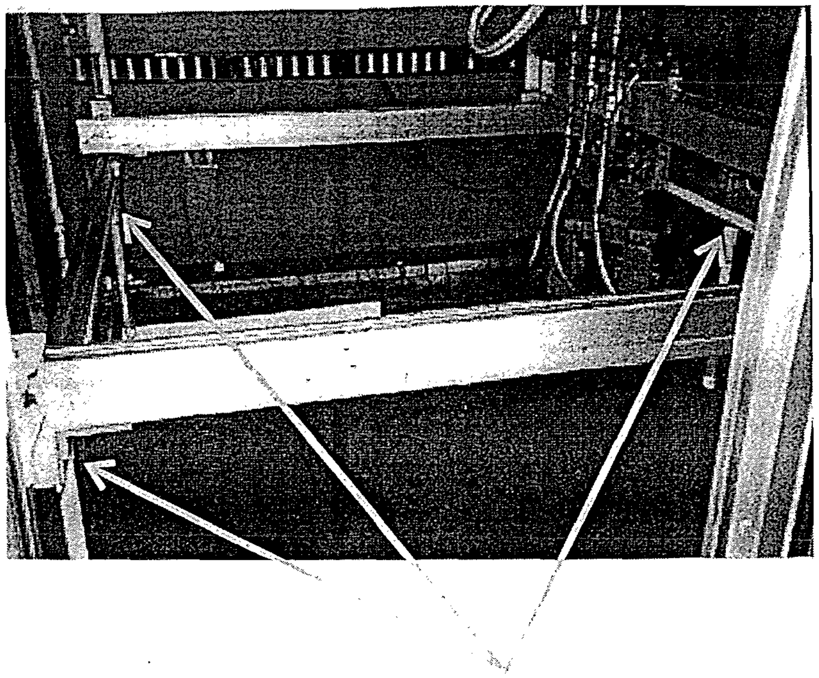


Figure 2.2.1: Gantry Leg Receptacle Locations

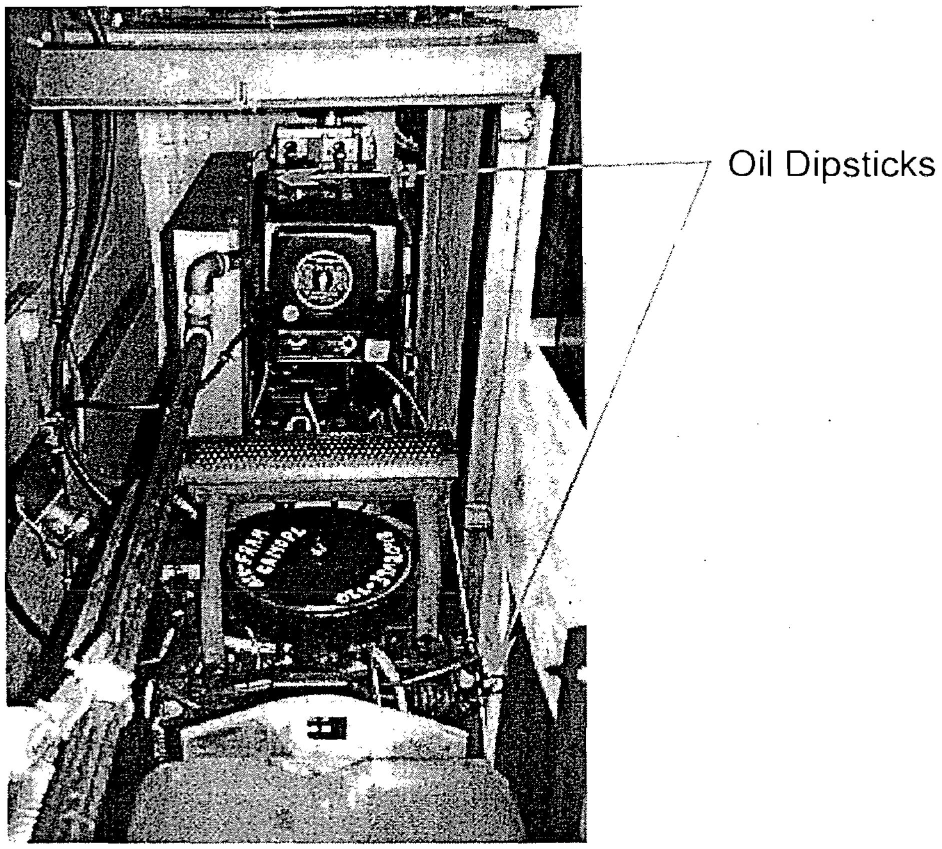


Figure 2.2.2: Oil level locations

2.3 Starting the Generator
General instructions are provided here for starting the Generator. If more details are required, please see the Generator's Manufacturer manual.

2.3.1 Remove all loads connected to generator by turning all circuit breakers to the 'OFF' position at the circuit breaker panel. (Figure 2.3.1) If external power is being used to heat the paint, disconnect the power from the trailer.

! Warning !

Failure to remove all loads connected to generator may result in damage to electrical loads and/or damage to generator unit.

2.3.2 Place the ignition switch to the 'ON' position.

2.3.3 Apply full choke. The choke actuation knob is located near the ignition switch on the engine control panels. (Figure 2.3.2)

2.3.4 Actuate the starter switch. Apply the starter a maximum of 10 seconds. Allow 1 minute between start attempts to allow starter to cool.

! Warning !

Failure to allow starter to cool may cause overheating and premature failure of the starter motor.

2.3.5 Once engine has started, push choke know in. Allow engine to warm up 2 minutes.

2.3.6 Place the fan and lights circuit breakers in the 'ON' position. The fan circuit breaker is #2 and the lights circuit breaker is #6. Refer to Figure 2.3.1 for locations.

! Warning !

After the fans have been turned on, do not cycle fan power while the robot and hand controller have power applied. Power surges from fan startup may damage the electronics.

! Warning !

Do not run fans without any of the trailer's ventilation ports open.

2.3.7 Briefly turn the fan on and then off using the power switch. Then turn the fan on. If the outside temperature is expected to be hot, use both fans; otherwise just use one fan. Turn the lights on.

2.3.8 Verify that all electrical equipment is off (Figure 1.5). Also check the Control Cabinet to ensure that the Gantry power supply and Power transformer is off. (Figure 1.5-d and Figure 2.3.3)

2.3.9 Place all power circuit breakers **EXCEPT #7** in the 'ON' position.

2.3.10 Turn the power conditioner on. Turn the power transformer (located below the power conditioner) on. (Figure 2.3.4)

1. Not used
2. Lights
3. Video Camera
4. Power Conditioner
5. Fan #1
6. Fan #2
7. Paint Heater
8. Not used

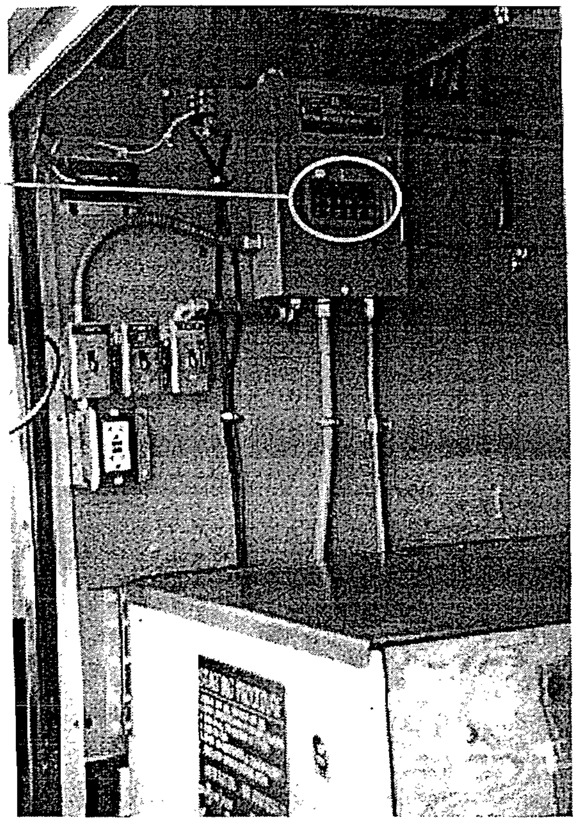


Figure 2.3.1 Circuit Breaker Locations

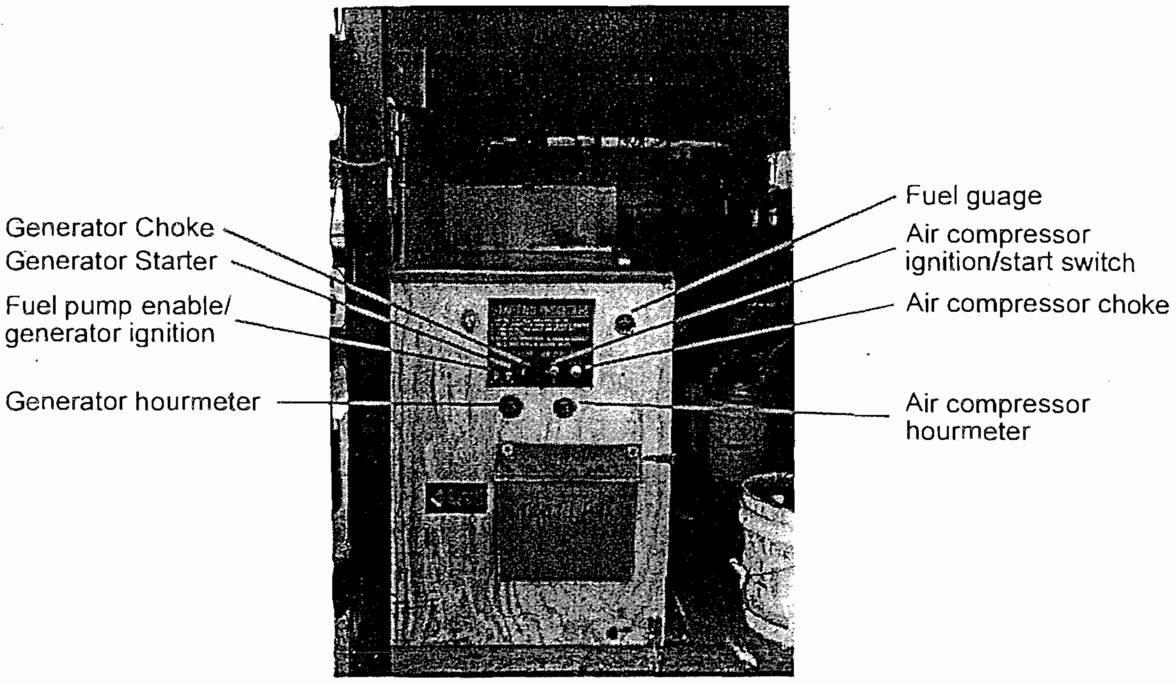


Figure 2.3.2: Engine Control Panel

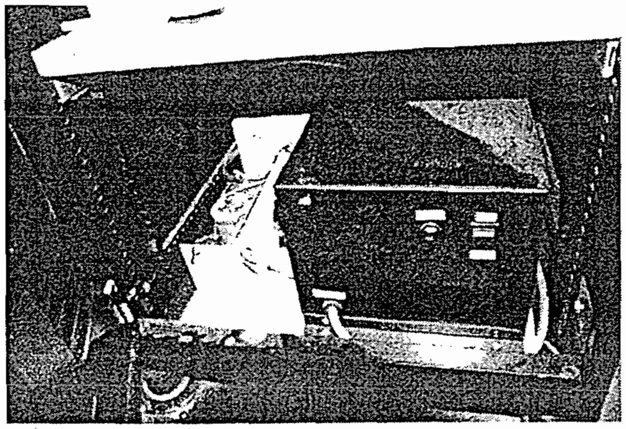


Figure 2.3.3: Gantry Power Supply and Power Transformer

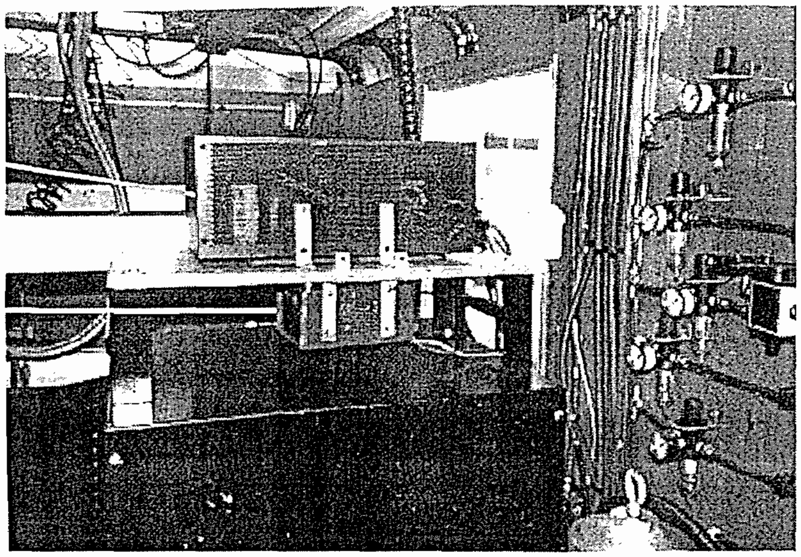


Figure 2.3.4: Power Conditioner and Power Transformer

2.4 Paint Heating System

2.4.1 Verify that water is in the paint heating tank. If water is cloudy or dirty, drain tank and refill with fresh water.

! Warning !

If external power is being used to preheat the paint tank, disconnect external power before proceeding. Failure to disconnect external power may result in electrical system damage and fire.

2.4.2 Place the water heater circuit breaker (#7) in the 'ON' position (Figure 2.3.1)

2.4.2 Verify that water heater power switch is in the blue position (Figure 2.4.1)

2.4.3 Verify that the water heater thermostat is set to the blue marker temperature (Figure 2.4.2) Disregard any other markings.

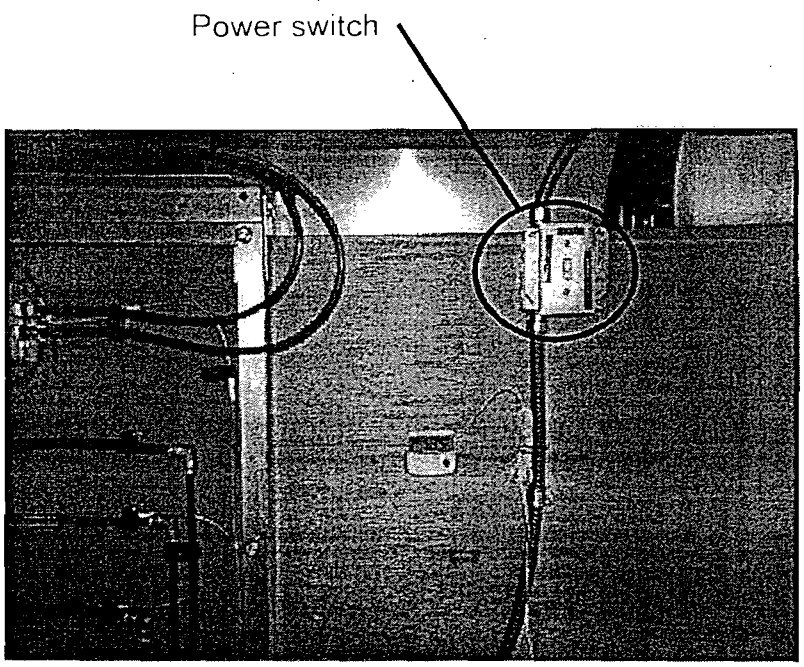


Figure 2.4.1: Heating Tank Power Switch

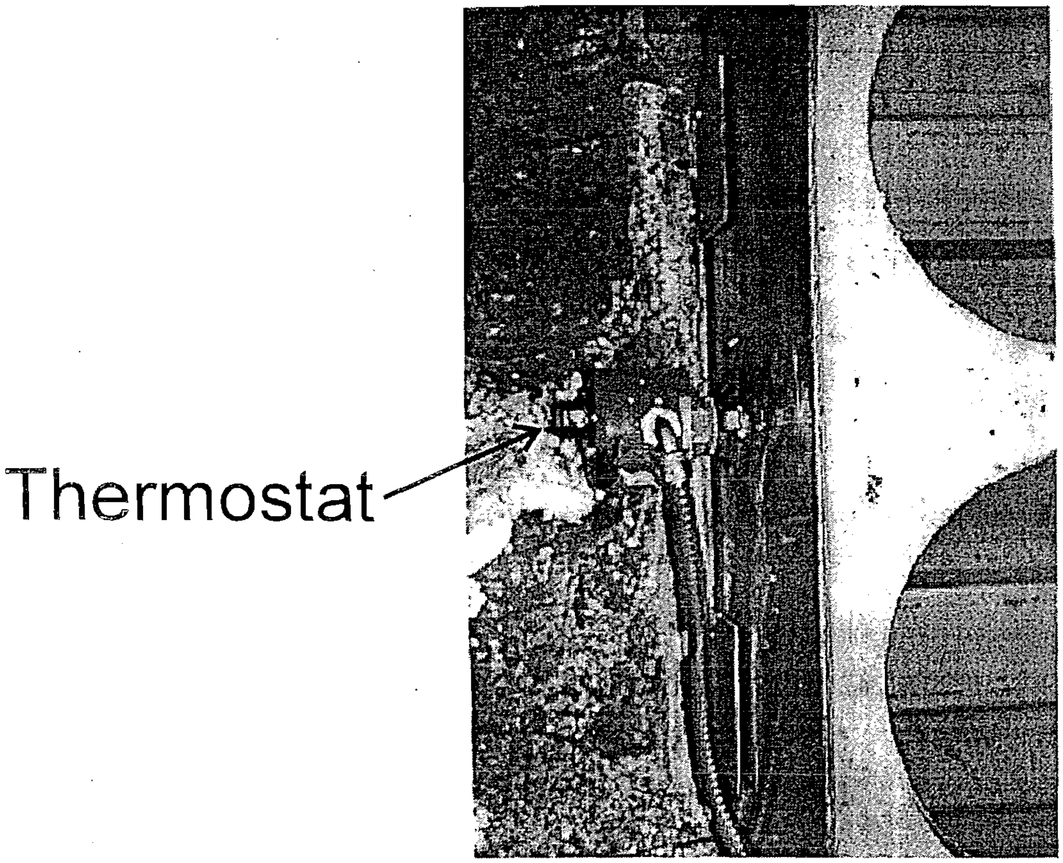


Figure 2.4.2: Heating Tank Thermostat

2.5 Starting the Air Compressor

General instructions are provided here for starting the Air Compressor. If more details are required, please see the Air Compressor's Manufacturer manual.

2.5.1 Apply full choke. The knob is located to the right of the ignition switch.

2.5.2 Crank engine by moving the toggle switch up to the 'crank' position. Crank engine a maximum of 10 seconds to prevent overheating of starter motor.

2.5.3 Once engine has started, remove choke. Allow 2 minutes for the engine to reach operating temperature.

! Warning !

The crankcase, cylinder head, exhaust system, and other components can get extremely hot from operation. To prevent severe burns, do not touch these areas while the engine is running or immediately after it is turned off. Never operate the engines with heat shields or guards removed.

! Warning !

Keep hands, feet, hair, and clothing away from all moving parts to prevent injury. Never operate the engine with covers, shrouds, or guards removed.

2.5.4 Briefly pull air tank water drain cord to purge water from the air compressor tank.

2.5.5 Allow 5 minutes for the air pressure to reach operating pressure.

2.6 Mixing the Paint

! Warning !

For best performance, pay particular attention to the addition of residual water to the paint. Small amounts of water will seriously degrade the quality of the premarks painted on the road.

2.6.1 Using a screwdriver, carefully remove the lid from the 5 gallon paint can. One can of black and one can of white traffic paint will be necessary to prime the system and begin initial calibration of the systems. Use Pervo Paint Fast-drying Water Based Road Paint, #4773A (white) and #4775A (black).

2.6.2 Using an approved mixing device, mix the paint. Pay particular attention to the sediment at the bottom of the cans, since the paint may have settled during shipment from the factory.

! Warning !

Failure to properly mix the paint to a smooth consistency will result in clogged pump siphon screens and a loss of paint pressure.

2.6.3 When all paint has been mixed, carefully place the paint containers in the pump stand. The black paint should be placed in the stand on the side nearest the front of the trailer. The white paint should be placed in the stand on the side nearest the back of the trailer. Carefully insert the siphon and agitator into the paint. Place the cover on top of the paint can.

2.7 Hand Controller Setup

! Warning !

Do not hold hand controller by cables. Do not pull on cables at the cable plug. Failure to observe these precautions will result in cable failure.

2.7.1 Retrieve the Hand Controller (Figure 2.7.1) and attach the cables. The larger cable attaches to the left plug.

2.7.2 Remove the covers from the pipe fittings at the front of the trailer. Place the covers in the cabinet at the front of the trailer.

2.7.2 Insert the other cable end into the pipe fittings. Attach the cable end to the mating plug inside the trailer. (Figure 2.7.2)

2.7.3 Insert the control cord for the warning lights through one of the pipe fittings. Insert cord plug into socket near the cable plug.

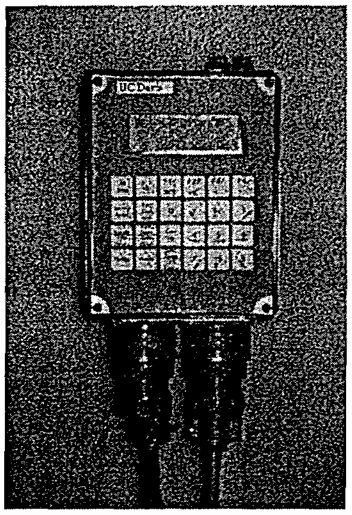


Figure 2.7.1: Hand Controller

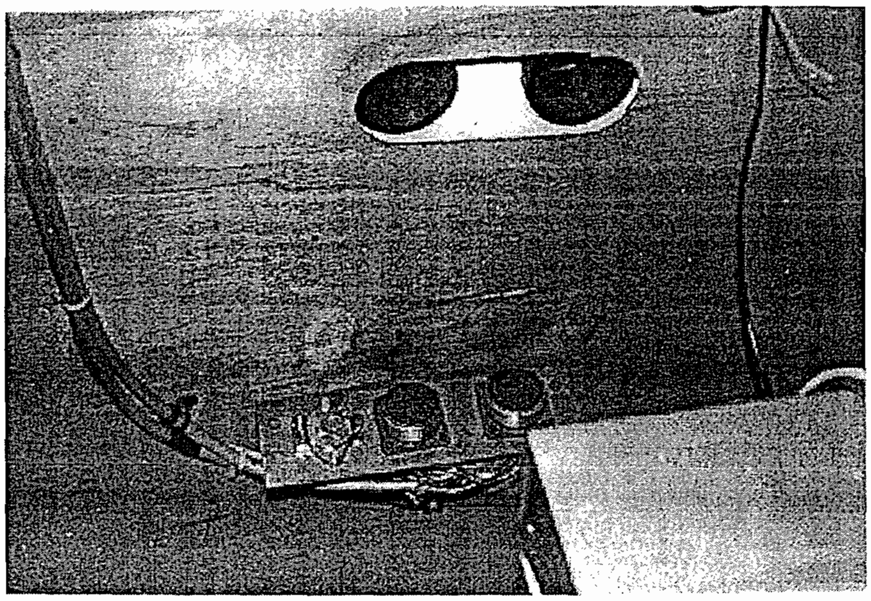


Figure 2.7.2: Cable Plugs Inside Trailer

2.8 System Power On

! Warning !

Before turning on the Gantry, confirm that no personnel are located within the workspace.

2.8.1 Turn on the power transformer, located on the bottom of the Control Cabinet. (Figure 2.8.1)

2.8.2 Turn on the Gantry power supply, located on the bottom of the Control Cabinet. (Figure 2.8.1) The Gantry LED display (Figure 1.5-c) should show 'EP'. If not, re-cycle power by turning off the Gantry power supply, waiting 5 seconds, and turning it on again.

2.8.3 Turn on the Solenoid Interface boards, located on the top shelf of the Control Cabinet. (Figure 2.8.2)

2.8.4 Turn on the Hand Controller, using the Red switch on the left. (Figure 2.8.3)

2.8.5 Manually move the gantry painting head to the center of the workspace.

2.8.6 Place the manual control panel control switch in the 'ON' position. (Figure 2.8.4)

2.8.7 Ensure that there are no obstructions within the gantry workspace. Lower the gantry by actuating the 'Gantry Up/Down' toggle switch on the manual control panel.

2.8.8 The gantry workframe legs should touch down on the ground before the air is automatically shutoff. If not, proceed to Step 2.8.9. The motion of the workframe should automatically shut off before the gantry reaches the top of the suspension bolts. If no space remains at the top of the suspension bolts, actuate the 'Gantry Up/Down' toggle switch to the 'OFF' (middle) position immediately to prevent workframe damage. Proceed to step 2.8.9, otherwise proceed to step 2.8.12.

2.8.9 Raise the workframe by actuating the 'Gantry Up/Down' toggle to the up position.

2.8.10 On the inside wall, near the cable plugs, are two cam actuated shutoff switches. Adjust the lower shutoff switch cam downward to allow the workframe more downward travel, or adjust it upwards to allow more upward travel. Adjust accordingly.

2.8.11 Lower the workframe using the manual switch until automatic shutoff. If no space remains at the top to the suspension bolts, actuate the 'Gantry Up/Down' toggle switch to the 'OFF' position immediately to prevent workframe damage. Readjust according to Steps 2.8.9 - 2.8.10.

2.8.12 Carefully observe the spacing of the gantry suspension bolts at the corners of the workframe. The workframe support should be at the middle of the travel of the suspension bolts. Otherwise, readjust according to Steps 2.8.9 - 2.8.10.

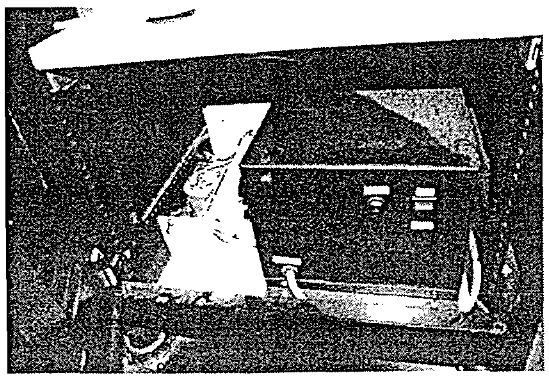


Figure 2.8.1: Gantry Power Supply and Power Transformer

Figure 2.8.2: Solenoid Interface Boards

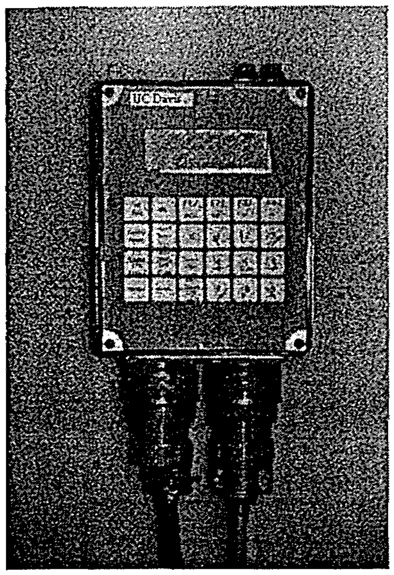


Figure 2.8.3: Hand Controller

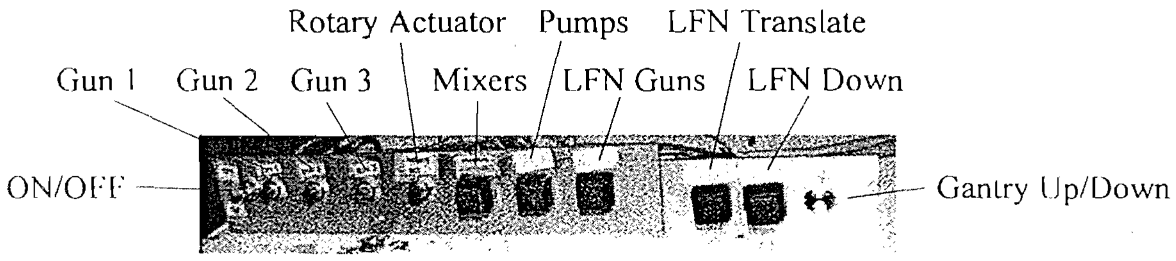


Figure 2.8.4: Manual Control Panel

2.9 Priming the Paint Pumps

2.9.1 Confirm that the main air inlet to the paint pumps are closed. The inlet is closed when the handle is in the horizontal position. (Figure 2.9.1)

2.9.2 Close the paint pump outlet valves. Also, close the paint pump bleed valve. (Figure 2.9.2 and Figure 2.9.3)

2.9.3 Place the manual control switch in the 'ON' position and toggle the 'PUMPS' switch to the 'ON' position (Figure 2.9.4). If desired at this time, the paint mixers can be activated to keep the paint mixed. To activate the mixers, depress the 'MIXERS' switch. (Figure 2.9.4)

2.9.4 If the mixers are to be used, adjust the mixing speed by turning the control valve to increase or decrease operating speed. (Figure 2.9.5)

2.9.5 Adjust the black and white pump air regulators, located on the Regulator Panel, to approximately 60 PSI (Figure 2.9.6)

2.9.6 Place a bucket at the end of the bleed valve hose.

2.9.7 Open the pump bleed valve. (Figure 2.9.2)

! Warning !

Exercise caution - pump may be under extremely high pressure. Secure hose to *prevent 'hose whip' when pressure is relieved.*

2.9.8 Open pump air inlet valve. (Figure 2.9.1)

2.9.9 Pump is fully primed when paint is flowing from bleed valve hose.

2.9.10 Close inlet air valve. (Figure 2.9.1)

2.9.11 Close bleed valve. (Figure 2.9.2)

2.9.12 Repeat steps 2.9.6 through 2.9.11 for the other paint pump.

2.9.13 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pumps.

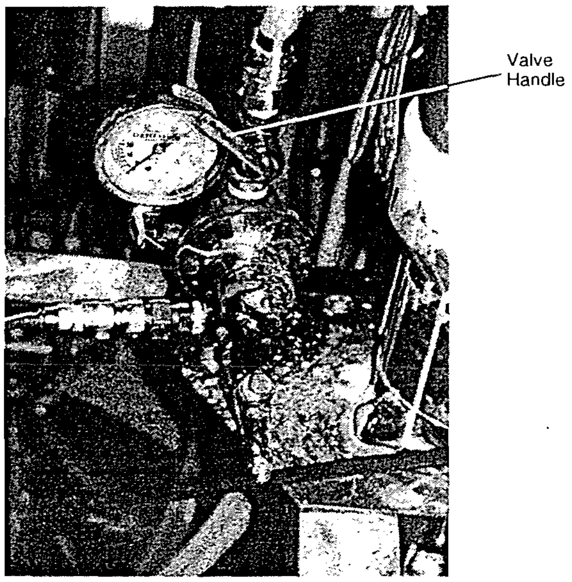


Figure 2.9.1: Pump Air Inlet Valve

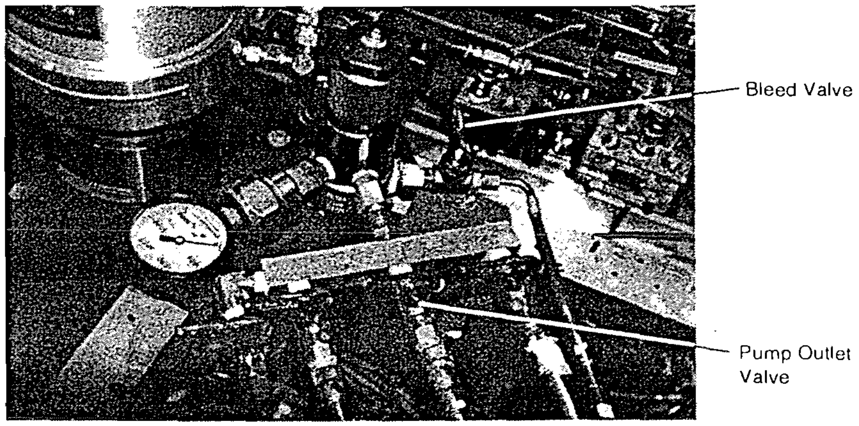


Figure 2.9.2: Pump Bleed Valve and Pump Outlet Valves

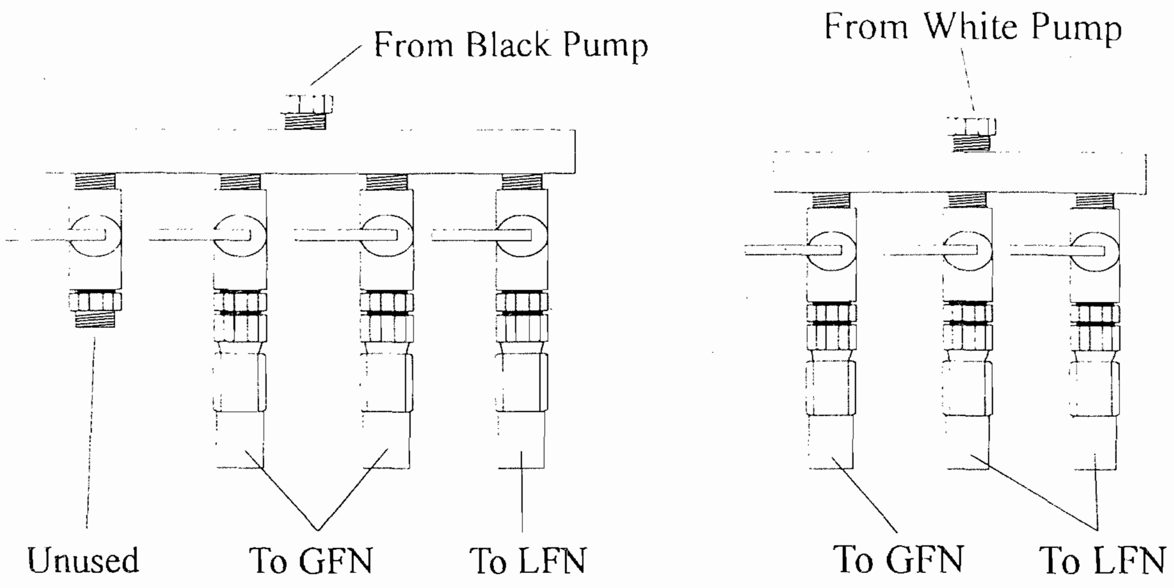


Figure 2.9.3: Pump Outlet Locations

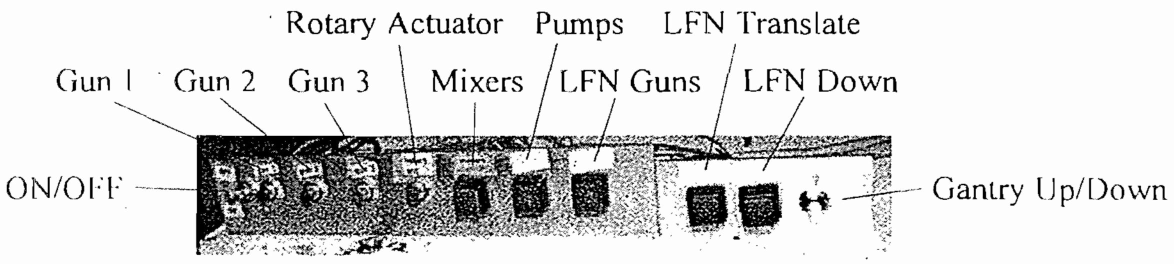


Figure 2.9.4: Manual Control Panel

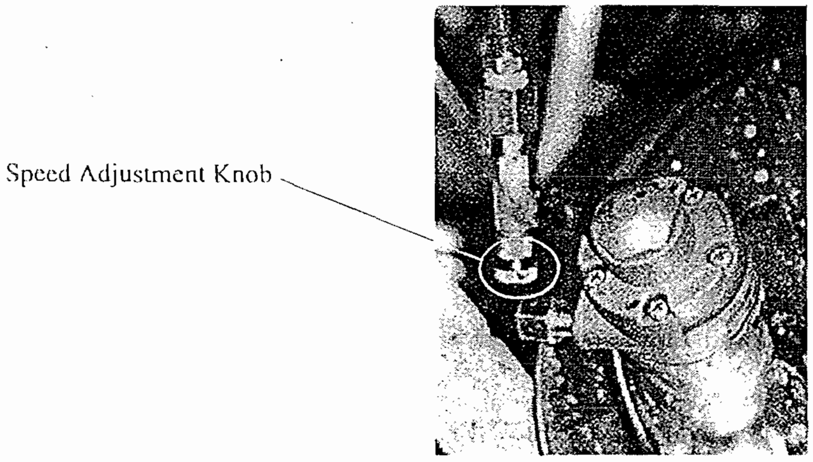


Figure 2.9.5: Mixer Speed Control

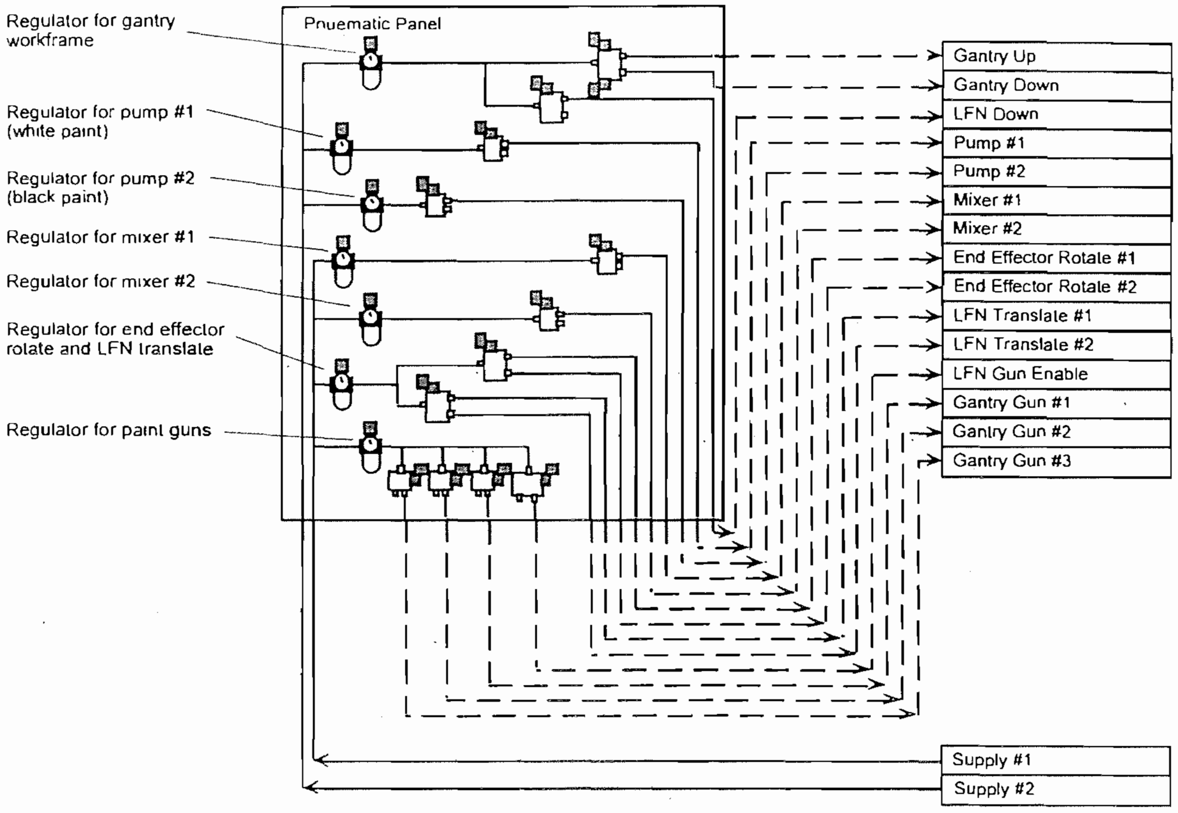


Figure 2.9.6: Regulator Panel

2.10 Priming the Paint Lines

! Warning !

The paint will flow from the gun outlet with considerable pressure. Skin damage may result if contact is made with any body part.

! Warning !

Before operating the gantry workframe, insure that there are no obstructions.

2.10.1 Position the Gantry workframe to a comfortable working height by actuating the 'UP/DOWN' switch on the Manual Control Panel.

2.10.2 Place a bucket underneath the gun outlets of the Gantry End Effector (Figure 2.10.1)

2.10.3 Open the black pump air inlet valve. (Figure 2.9.1)

2.10.4 Open the black paint outlet valves connected to the Gantry. There should be 2 black paint valves to open. Reference Figure 2.9.3 for the location.

2.10.5 Depress the 'GUN 1' switch to open the first black paint gun. With the 'GUN 1' switch depressed, toggle the 'PUMPS' switch to the 'ON' position to activate the paint pump. (Figure 2.9.4) Allow pump to run, until paint runs freely from the gun outlet.

2.10.6 Toggle the 'PUMPS' switch to the 'OFF" position to deactivate the pump. Release the 'GUN 1' switch. (Figure 2.9.4)

2.10.7 Depress the 'GUN 3' switch. With the 'GUN 3' switch depress, toggle the 'PUMPS' switch to the 'ON' position. (Figure 2.9.4) Allow pump to run, until paint runs freely from the gun outlet.

2.10.8 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pump. Release the 'GUN 3' switch. (Figure 2.9.4)

2.10.9 Close the black pump air inlet valve. (Figure 2.9.1)

2.10.10 Open the white pump air inlet valve. (Figure 2.9.1)

2.10.11 Open the white paint outlet valves connected to the Gantry. There should be 1 white paint valve to open (Figure 2.9.3).

2.10.12 Depress the 'GUN 2' switch to open the first white paint gun. With the 'GUN 2' switch depress, toggle the 'PUMPS' switch to the 'ON' position to activate the paint pump. (Figure 2.9.4) Allow pump to run, until paint runs freely from the gun outlet.

2.10.13 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pump. Release the 'GUN 2' switch (Figure 2.9.4)

2.10.14 Close the white pump air inlet.

2.10.15 The Gantry paint lines are now primed.

NOTE: If the LFN is not going to be use, please skip the rest of Section 2.10.

2.10.16 Position a bucket underneath the paint gun outlets of the Linear Free Nozzle (LFN) system.

2.10.17 Depress the 'Pre. Guns' button on the manual control panel (Figure 2.9.4)

2.10.18 Toggle the 'PUMPS' button to the 'ON' position. (Figure 2.9.4)

2.10.19 Open the pump air inlet valves for both paint pumps (Figure 2.9.1).

2.10.20 Open the black and white pump outlet valves. There should be 2 white outlet valves and 1 black outlet valve to open (Figure 2.9.3).

2.10.21 Locate the roller cam valve for each of the guns (Figure 2.10.2). Depressing the cam valve will open the paint gun. Depress each one until paint flows freely from the gun outlet.

2.10.22 Close the air inlet valve to both pumps.

2.10.23 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pumps. Depress the 'Pre. Guns' to deactivate the LFN guns.

2.10.24 Remove the paint buckets from underneath the Gantry and LFN systems.

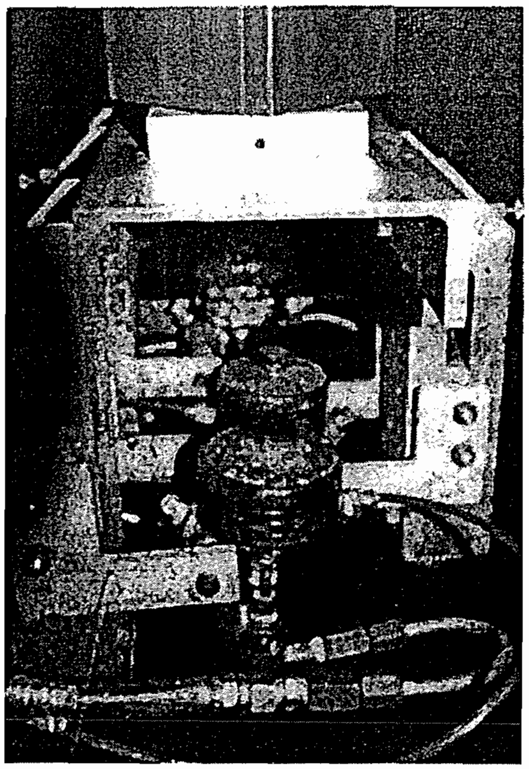


Figure 2.10.1: Gantry Guns

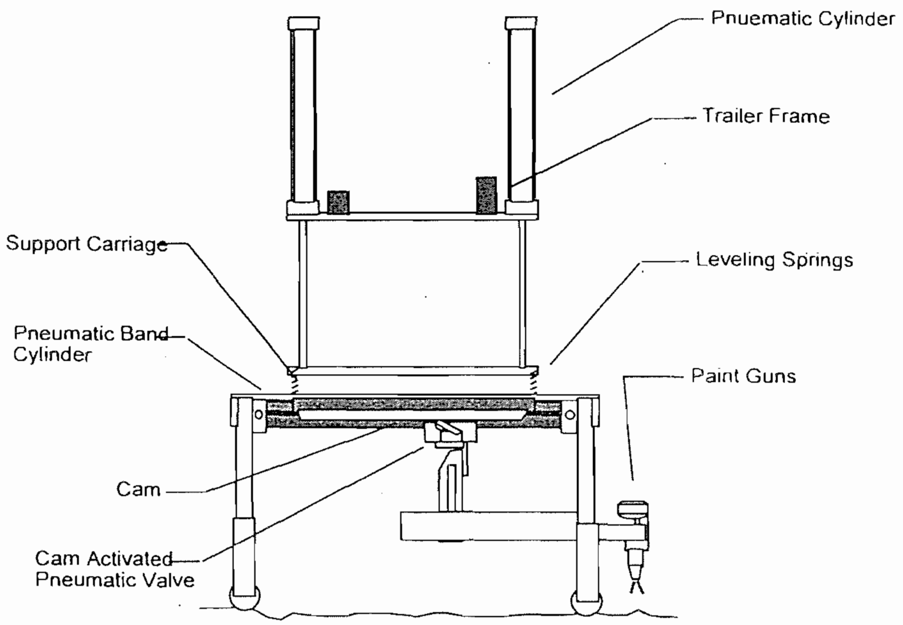


Figure 2.10.2: Roller Cam Location on LFN

2.11 Nozzle Installation

! Warning !

The paint will flow from the nozzle tip with considerable pressure. Skin damage may result if contact is made with any body part.

! Note !

Should any system malfunction during automatic control, depress the LEFT red switch on the hand controller to activate the EMERGENCY STOP. Note that gantry motion may not cease immediately.

2.11.1 Prepare 3 (three) size 1380 Binks paint nozzles. Also prepare 3 (three) nozzle filters. Pay particular attention to stray dirt and paint specks.

! Warning !

Lack of cleanliness will result in abnormal spray patterns and lack of performance from the spray nozzle.

2.11.2 Assemble the nozzle unit, as shown in Figure 2.11.1.

! Note !

The paint nozzle is keyed to fit into the orange spray tip.

2.11.3 Screw the nozzle unit onto the end of the Gantry spray gun. Using a 7/8" wrench, tighten the mount nut.

2.11.3 With the orange spray tip hand tight, turn the tip 90 degrees from the final desired position, perpendicular to the line of travel of the spray head. See Figure 2.11.2 for the orientation of the spray pattern.

2.11.4 Tighten the Nozzle nut by turning the nut 90 degrees. The orientation should be as close to perpendicular to the line of travel as possible.

2.11.5 Repeat Steps 2.11.2 through 2.11.4 for the rest of the guns.

NOTE: If the LFN is not going to be used, please skip to Step 2.11.8

2.11.6 Prepare 2 (two) size 1180 and 1 (one) size 1380 Binks spray tip. Also prepare 3 (three) nozzle filters. Pay particular attention to stray dirt and paint specks. Unchain the 2 chains restraining the LFN unit. (Figure 2.11.3)

! Warning !

Lack of cleanliness will result in abnormal spray patterns and lack of performance from the spray nozzle.

2.11.7 Repeat Steps 2.11.2 through 2.11.4. Note that the 2 (two) size 1180 spray tips are to be located on the outside guns of the LFN and the size 1380 tip on the center gun.

2.11.8 Place a piece of Simulated Road Material (i.e., Mineral Surface Rolled Roofing), approximately 1 foot wide by 3 foot long, under the Gantry end effector. (Figure 2.11.4) If the LFN is to be used, place a similar piece under the guns of the LFN.

2.11.9 Position the Gantry by actuating the 'UP/DOWN' switch on the manual control panel (Figure 2.9.4) so that the gantry gun tips are approximately 12 inches above the Simulated Road Material.

2.11.10 Depress the 'PUMPS' switch to activate the paint pumps.

2.11.11 Open the main air inlet for both pumps. (Figure 2.9.1) The valve handle should be in the vertical position.

2.11.12 Open the outlet valves for the Gantry system. There should be 2 black valves and 1 white valve to open. Reference 2.9.3 for valve location.

2.11.13 On the regulator panel, adjust the air regulator to 130 PSI for the black pump, and 115 PSI for the white pump. (Figure 2.9.6)

2.11.14 Depress the 'GUN 1' switch for approximately 1/10 second. Depress the 'GUN 3' switch for approximately 1/10 second. Depress the 'GUN 2' switch for approximately 1/10 second. The resulting spray pattern should look like Figure 2.11.5. Use a wrench to adjust the orientation of the orange spray tip to bring the spray pattern into alignment. Manually pivot the entire gun to adjust the start point of the spray pattern.

2.11.15 Repeat Step 2.11.14 with a new piece of Simulated Road Material until alignment is achieved.

NOTE: If the LFN is not going to be used, please skip to Step 2.11.20

2.11.16 Depress the 'Pre. Guns' button on the manual control panel to activate the LFN guns.

2.11.17 Depress the 'Pre. Down' button on the manual control panel to lower the LFN.

2.11.18 Depress each of the cam valves for approximately 1 seconds each. The spray pattern should look similar to Figure 2.11.5. Use a wrench to adjust the orientation of the tip to bring the spray pattern into alignment. Manually pivot the gun to adjust the start point of the spray pattern.

2.11.19 Depress the 'Pre. Down' button on the manual control panel to raise the LFN. Depress the 'Pre. Guns' button to deactivate the LFN guns.

2.11.20 Place the manual control panel master switch to 'OFF'.

2.11.21 Cut a 3' wide by 4' long section of Simulated Road Material. Place it into the Gantry workspace as shown in Figure 2.11.6.

2.11.22 Using the hand controller, press the 'mixers' button to activate the mixers under computer control. The LCD display should respond as shown in Figure 2.11.7.

2.11.23 Manually move the gantry end effector to the center of the workspace. Press the 'Gantry Reset' button. The gantry workframe should rise and the end effector should move to the corner. Reference Figure 2.11.6 for correct orientation.

2.11.24 Press the 'GFN Funct/LFN Funct' button to select Gantry functions. Press the 'Gantry Down' button. The gantry workframe should begin lowering. When it reaches the ground and shuts off, press the 'Gantry Down' button again.

2.11.25 Press the 'Pumps' button to activate the paint pumps. Press the 'Mixers' button to de-activate the paint mixers. Reference Figure 2.11.7 for correct LCD display indications.

2.11.26 Press the 'Auto Paint System' button. The menu (Figure 2.11.8) should be shown. Press 'F2' to activate the gantry test.

2.11.27 When the gantry motion is complete, deactivate the paint pumps by pressing the 'Pumps' button and re-activate the mixers by pressing the 'Mixers' button.

2.11.28 Observe the painted test mark. The black-white-black stripes should meet, without overlapping edges. If not, continue to Step 2.11.29, otherwise continue to Step 2.11.30.

! Note !

The quality of first few runs after initial system startup will be poor since the nozzle tips will contain water and entrained air.

It is recommended that the test mark be performed a few times before adjustment to the system is made.

2.11.29 Loosen the thumbscrew of the gun assembly that needs adjustment. (Figure 2.11.9) Using a 9/32" socket on the height adjustment screw (Figure 2.11.9) adjust the height of the gun assembly. **Raising the gun assembly increases the width of the spray pattern; lowering the gun assembly decreases the width of the spray pattern.** Adjust gun assembly accordingly. Retighten the thumbscrew when finished. Adjust all guns. Repeat Steps 2.11.25 through 2.11.28 with a new piece of Simulated Road Material until all stripe edges meet.

2.11.30 Press the 'Gant Up' button to raise the gantry workframe. Raise workframe approximately halfway. Press the 'Gant Up' again button to stop the workframe at the desired position.

NOTE: If the LFN is not going to be used, please skip to Step 2.11.36

2.11.31 Place a piece of Simulated Road Material underneath the LFN.

2.11.32 Press the 'Pumps' button to activate the pumps. Press the 'Mixers' button to deactivate the mixers.

2.11.33 Press the 'Auto Paint Sys' button. The menu should be displayed. Select 'F3' to paint an abbreviated premark. The LFN should cycle. Observe the mark painted. The edges of all stripes should meet, with no overlap. If not, proceed to Step 2.11.33, otherwise continue to Step 2.11.34.

! Note !

The quality of first few runs after initial system startup will be poor since the nozzle tips will contain water and entrained air.

It is recommended that the test mark be performed a few times before adjustment to the system is made.

2.11.34 Loosen the thumbscrew of the gun assembly that needs adjustment. (Figure 2.11.10) Using a 9/32" socket on the height adjustment screw (Figure 2.11.10) adjust the height of the gun assembly. **Raising the gun assembly increases the width of the spray pattern; lowering the gun assembly decreases the width of the spray pattern.** Adjust gun assembly accordingly. Retighten the thumbscrew when finished. Adjust all guns. Repeat Steps 2.11.33 through 2.11.34 with a new piece of Simulated Road Material until all stripe edges meet.

2.11.35 Press the 'Pumps' button to deactivate the pumps. Press the 'Mixers' button to activate the mixers.

2.11.36 All calibration procedures are now complete.

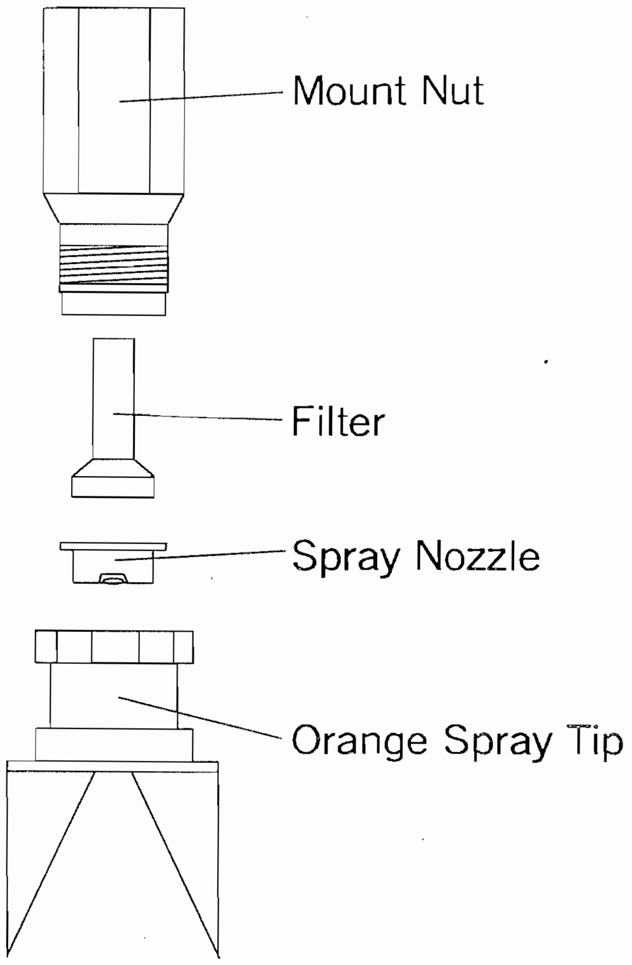


Figure 2.11.1: Nozzle Unit

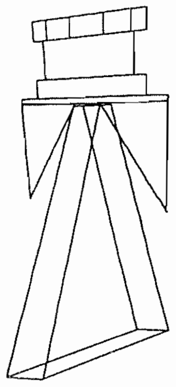


Figure 2.11.2: Spray Pattern

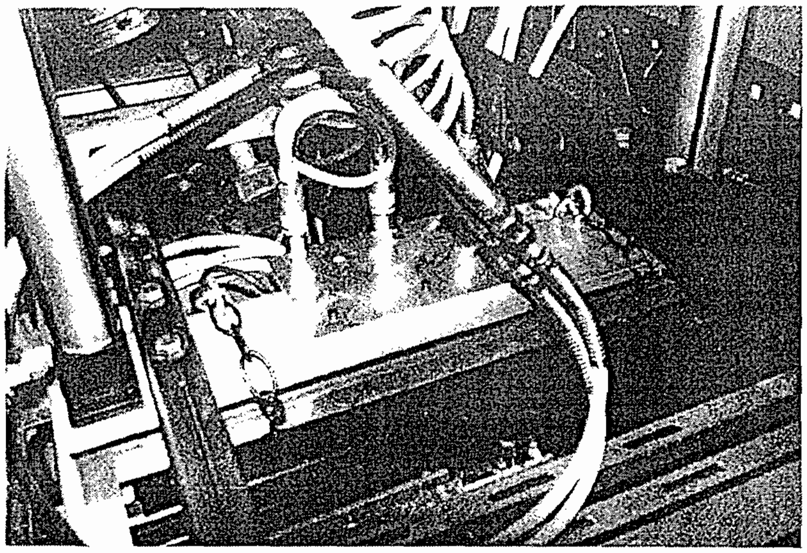


Figure 2.11.3: LFN Chains

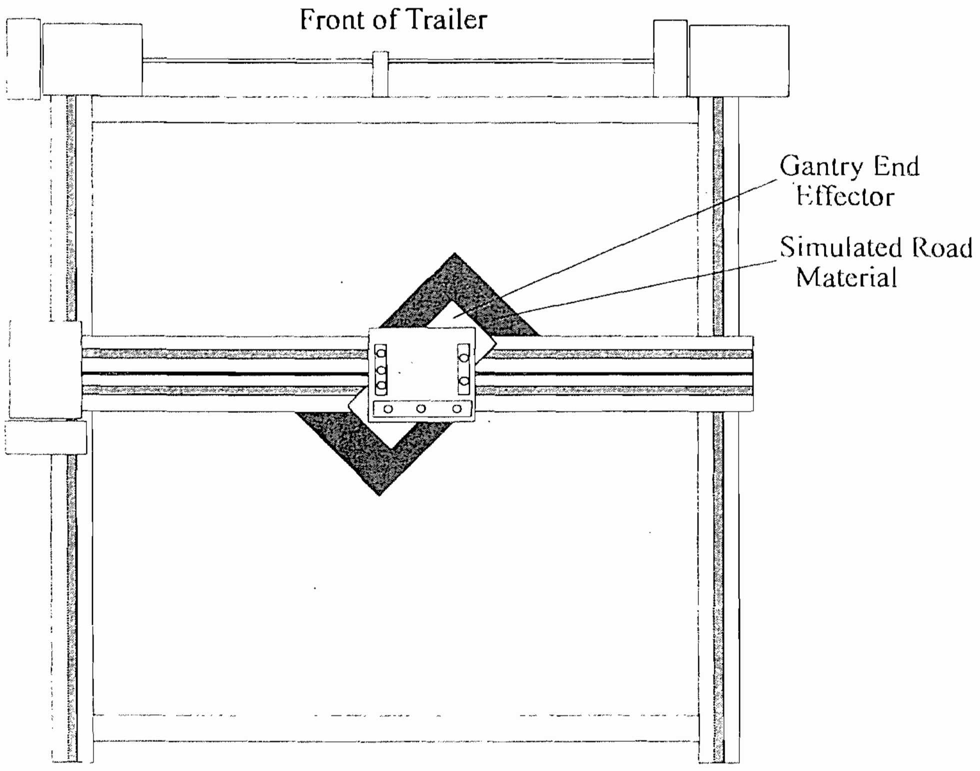


Figure 2.11.4: Material Installation

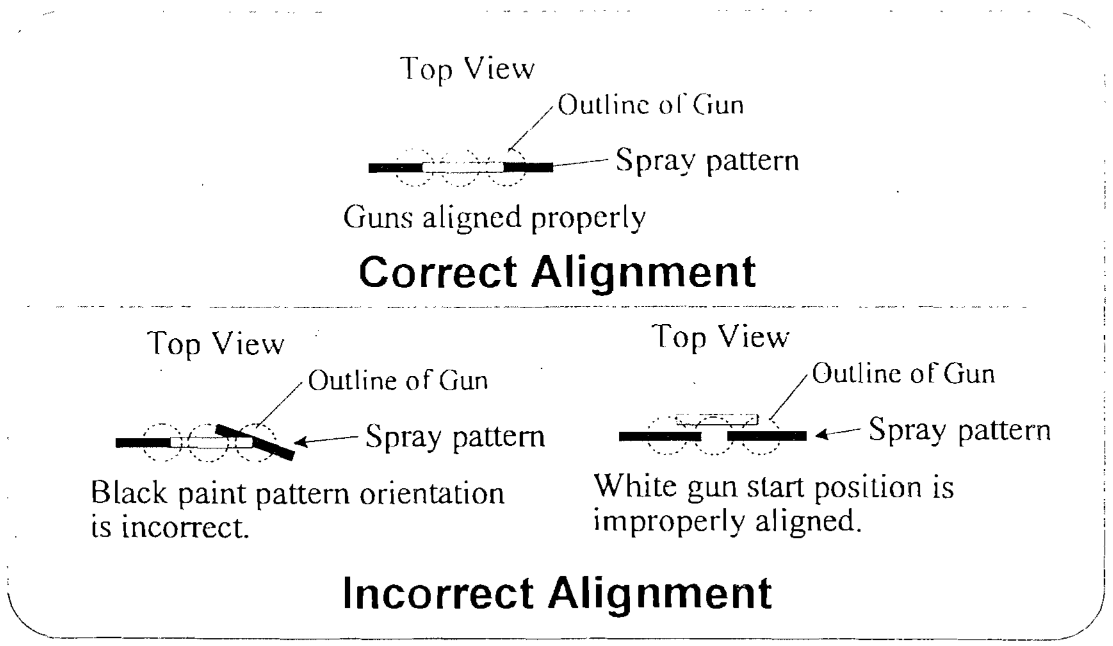


Figure 2.11.5: Spray Orientation

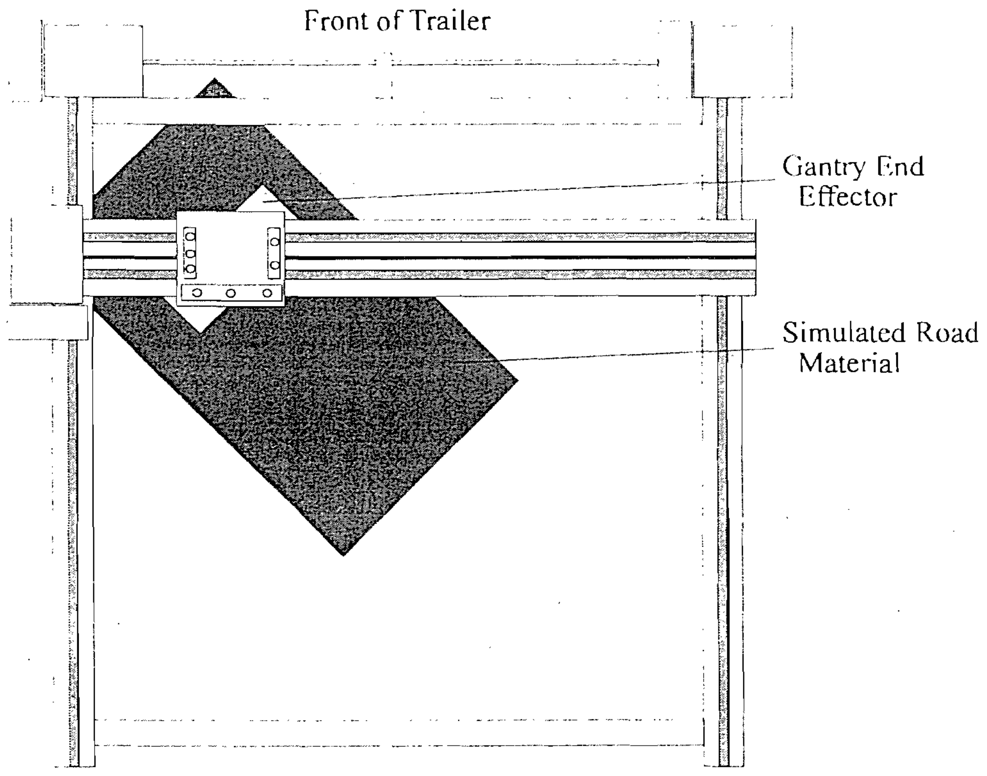
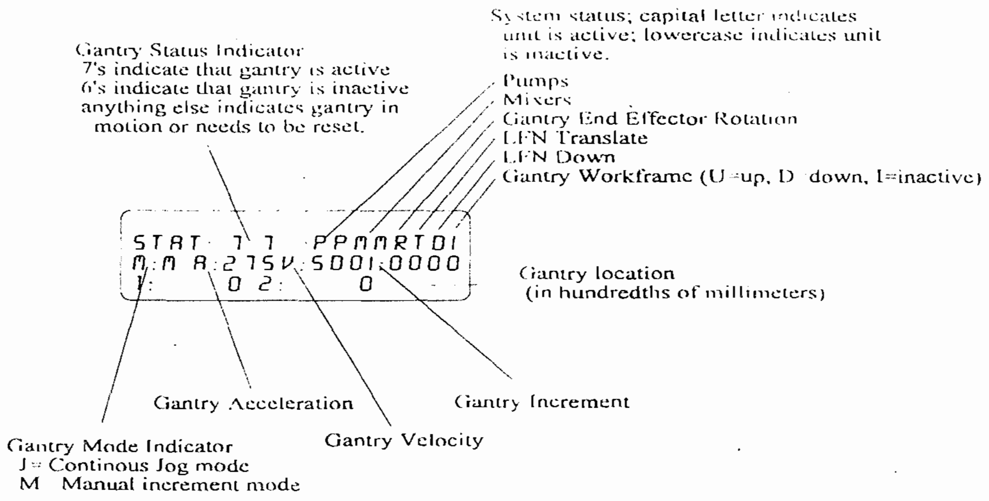
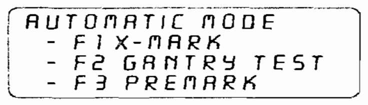


Figure 2.11.6: Material Orientation for Gantry Test


System status; capital letter indicates unit is active; lowercase indicates unit is inactive.
Pumps
Mixers
Gantry End Effector Rotation
LFN Translate
LFN Down
Gantry Workframe (U=up, D down, I=inactive)
Gantry Status Indicator
7's indicate that gantry is active
6's indicate that gantry is inactive
anything else indicates gantry in motion or needs to be reset.
Gantry location
(in hundredths of millimeters)
Gantry Acceleration
Gantry Increment
Gantry Velocity
Gantry Mode Indicator
J = Continous Jog mode
M Manual increment mode

```
STAT 7 7  PPMMRTDI
M:M A:275V:5001:0000
J:    0 2:    0
```

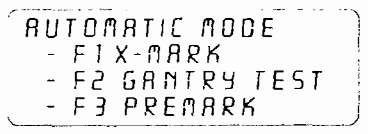
Status Display



```
AUTOMATIC MODE
  - F1 X-MARK
  - F2 GANTRY TEST
  - F3 PREMARK
```

Automatic Menu

Figure 2.11.7: LCD Status Display



```
AUTOMATIC MODE
  - F1 X-MARK
  - F2 GANTRY TEST
  - F3 PREMARK
```

Automatic Menu

Figure 2.11.8: LCD Menu Display

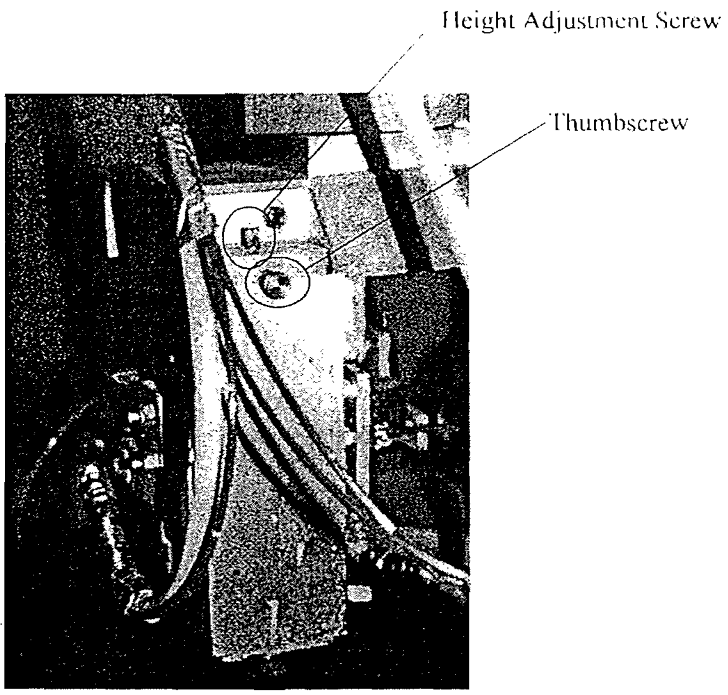


Figure 2.11.9: Gantry Gun Adjustment Locations

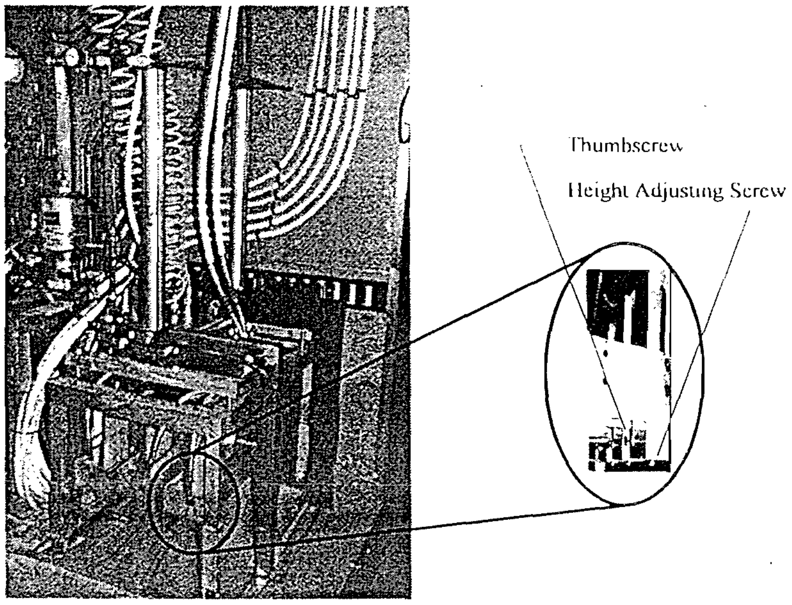


Figure 2.11.10: LFN Gun Adjustment Locations

3. Normal Operations

3.1 Introduction

This sections describes the normal operating procedure of the Premark trailer.

3.2 Placing a premark

! Note !

The surveying marks must be placed according to standard CalTrans operating procedures. Please reference the appropriate CalTrans standard for exact details.

! Note !

During normal operations, the hand control should have a blinking green light. If this light should cease blinking or the yellow low power light illuminates, cease operations immediately and correct the equipment fault before continuing operations.

3.2.1 When the appropriate location has been established, look at the status panel on the LCD of the Hand controller (Figure 3.2.1). If the paint mixers are active, press the 'Mixers' button on the Hand controller to deactivate the paint mixers.

3.2.2 Press the 'Pumps' button to activate the paint pumps.

3.2.3 Press the 'Auto Paint Sys' button to bring up the selection menu.

3.2.4 Press 'F1' to begin the process of painting the premark.

3.2.5 Wait approximately 2.5 minutes. When the LCD display indicates that the process has completed, disable the pumps by press the 'Pumps' button. Re-activate the paint mixers by pressing the 'Mixers' button.

3.2.6 Proceed to the next location.

3.3 Placing an abbreviated premark

! Note !

The surveying marks must be placed according to standard CalTrans operating procedures. Please reference the appropriate CalTrans standard for exact details.

! Note !

During normal operations, the hand control should have a blinking green light. If this light should cease blinking or the yellow low power light illuminates, cease operations immediately and correct the equipment fault before continuing operations.

3.3.1 When the appropriate location has been established, look at the status panel on the LCD of the Hand controller (Figure 3.2.1). If the paint mixers are active, press the 'Mixers' button on the Hand controller to deactivate the paint mixers.

3.3.2 Press the 'Pumps' button to activate the paint pumps.

3.3.3 Press the 'Auto Paint Sys' button to bring up the selection menu.

3.3.4 Press 'F3' to begin the process of painting the abbreviated premark.

3.3.5 Wait approximately 2.5 minutes. When the LCD display indicates that the process has completed, disable the pumps by press the 'Pumps' button. Re-activate the paint mixers by pressing the 'Mixers' button.

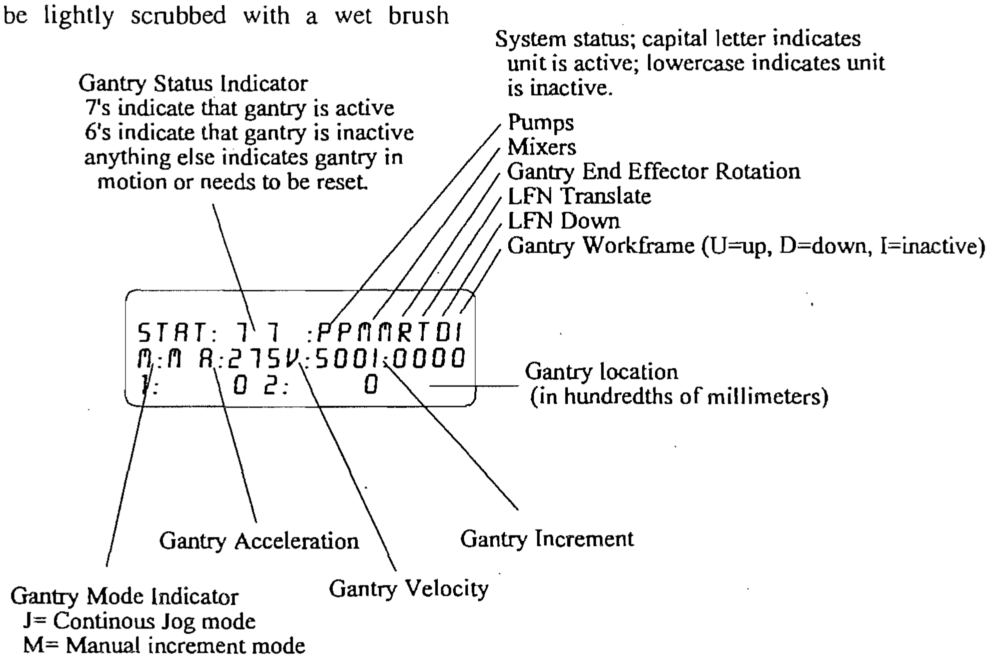
3.3.6 Proceed to the next location.

3.4 On Road Maintenance

3.4.1 Deactivate the gantry by pressing the 'Gant Funct/LFN Funct' button. Check the hand controller LCD panel to confirm that the gantry is deactivated (Figure 3.2.1).

3.4.2 Manually position the gantry end effector to allow for easy reach of the spray nozzles.

3.4.3 For optimum results, the spray nozzles should be lightly scrubbed with a wet brush every 2 hours. Also, the nozzles should be lightly scrubbed if the machine is not going to be used for a period of more than 30 minutes.


System status; capital letter indicates unit is active; lowercase indicates unit is inactive.
Gantry Status Indicator
7's indicate that gantry is active
6's indicate that gantry is inactive
anything else indicates gantry in motion or needs to be reset.
Pumps
Mixers
Gantry End Effector Rotation
LFN Translate
LFN Down
Gantry Workframe (U=up, D=down, I=inactive)
STAT: 7 7 :PPMMRTDI
M:M A:275V:5001:0000
1: 0 2: 0
Gantry location
(in hundredths of millimeters)
Gantry Acceleration
Gantry Increment
Gantry Velocity
Gantry Mode Indicator
J= Continous Jog mode
M= Manual increment mode

Status Display

Figure 3.2.1: Status Panel of the Hand Controller

4. Cleanup Procedures

! Warning !

Waste paint is toxic. Disposed of waste paint according to established toxic waste procedures. Consult the appropriate CalTrans standard for details.

! Warning !

The paint will flow from the nozzle tip with considerable pressure. Skin damage may result if contact is made with any body part.

4.1 Introduction

This section describes the cleanup procedures. Some sections maybe performed in parallel, if necessary, to speed up clean up time.

4.2 Nozzle Cleanup

4.2.1 Using a 7/8" wrench remove the complete nozzle unit from the end of the spray guns. Disassemble the unit.

4.2.2 Flush each component thoroughly under running water. Use a small scrub brush to remove deposits from the components. Pay particular attention to the spray tip orifice.

! Warning !

Failure to clean the spray tip thoroughly will result in a clogged tip. Decreased painting performance will result and tip replacement maybe necessary.

4.2.3 Allow all components to air dry.

4.3 Paint Pump Cleanup

4.3.1 Confirm that the paint mixers are off before removing the siphon hose covers. Remove the paint cans from the pump stand. Replace the paint lids.

! Warning !

Removal of siphon hose covers without turning off the paint mixers may result in extensive physical injury.

! Warning !

Improper replacement of paint lids will cause premature drying and thickening of paint.

4.3.2 Place the end of the siphon hoses in a clean bucket of water. Gently scrub the siphon strainer and siphon pipe to remove paint and paint deposits. Replace water with clean water.

4.3.3 Close the main air inlet to the paint pumps. The inlet is closed when the handle is in the horizontal position. (Figure 2.9.1)

4.3.4 Close the paint pump outlet valves. Also, close the paint pump bleed valve. (Figure 2.9.2 and Figure 2.9.3)

4.3.5 Place the manual control switch in the 'ON' position and toggle the 'PUMPS' switch to the 'ON' position.

4.3.6 Adjust the black and white pump air regulators, located on the Regulator Panel, to approximately 60 PSI (Figure 2.9.6)

4.3.7 Place a bucket at the end of the bleed valve hose.

4.3.8 Open the pump bleed valve. (Figure 2.9.2)

! Warning !

Exercise caution - pump may be under extremely high pressure. Secure hose to prevent 'hose whip' when pressure is relieved.

4.3.9 Open pump air inlet valve. (Figure 2.9.1)

4.3.10 Pump is fully clean when clear water flows from bleed valve hose.

4.3.11 Close inlet air valve. (Figure 2.9.1)

4.3.12 Close bleed valve. (Figure 2.9.2)

4.3.13 Repeat steps 4.3.7 through 4.3.12 for the other paint pump.

4.3.14 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pumps.

4.4 Cleaning the Paint Hoses

! Warning !

The water will flow from the gun outlet with considerable pressure. Skin damage may result if contact is made with any body part.

4.4.1 Lower the Gantry frame to a comfortable working height by actuating the 'UP/DOWN' switch on the Manual Control Panel.

4.4.2 Place a bucket underneath the gun outlets of the Gantry End Effector (Figure 2.10.1)

4.4.3 Open the black pump air inlet valve. (Figure 2.9.1)

4.4.4 Open the black paint outlet valves connected to the Gantry. There should be 2 black paint valves to open. Reference Figure 2.9.3 for the location.

4.4.5 Depress the 'GUN 1' switch to open the first black paint gun. With the 'GUN 1' switch depress, toggle the 'PUMPS' switch to the 'ON' position to activate the paint pump. (Figure 2.9.4)

4.4.6 Allow pump to run, until clean water runs freely from the gun outlet.

4.4.7 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pump. Release the 'GUN 1' switch. (Figure 2.9.4)

4.4.8 Depress the 'GUN 3' switch. With the 'GUN 3' switch depress, toggle the 'PUMPS' switch to the 'ON' position. (Figure 2.9.4)

4.4.9 Allow pump to run, until clean water runs freely from the gun outlet.

4.4.10 Toggle the 'PUMPS' switch to the 'OFF' position to deactivate the pump. Release the 'GUN 3' switch. (Figure 2.9.4)

4.4.11 Close the black pump air inlet valve. (Figure 2.9.1)

4.4.12 Open the white pump air inlet valve. (Figure 2.9.1)

4.4.13 Open the white paint outlet valves connected to the Gantry. There should be 1 white paint valve to open. Reference Figure 2.9.3 for the location.

4.4.14 Depress the 'GUN 2' switch to open the first white paint gun. With the 'GUN 2' switch depress, press the 'PUMPS' switch to activate the paint pump. (Figure 2.9.4)

4.4.15 Allow pump to run, until clean water runs freely from the gun outlet.

4.4.16 Depress the 'PUMPS' switch to deactivate the pump. Release the 'GUN 2' switch. (Figure 2.9.4)

4.4.17 Close the white pump air inlet.

4.4.18 The Gantry paint lines are now clean.

NOTE: If the LFN has not been used, please skip to Step 4.4.25.

4.4.19 Position a bucket underneath the paint gun outlets of the Linear Free Nozzle (LFN) system.

4.4.20 Depress the 'Pre. Guns' button on the manual control panel (Figure 2.9.4)

4.4.21 Depress the 'PUMPS' button. (Figure 2.9.4)

4.4.22 Open the pump air inlet valves for both paint pumps.

4.4.23 Open the black and white pump outlet valves. There should be 2 white outlet valves and 1 black outlet valve to open. Reference 2.9.3 for the location.

4.4.24 Locate the roller cam valve for each of the guns. (Figure 2.10.2) Depressing the cam valve will open the paint gun. Depress each one until clean water flows freely from the gun outlet.

4.4.25 Close the air inlet valve to both pumps.

4.4.26 Depress the 'PUMPS' switch to deactivate the pumps. Depress the 'Pre. Guns' to deactivate the LFN guns.

4.4.27 Open all paint line outlet valves. Relieve residual pressure in pump and lines by opening the bleed valves.

4.4.28 Close all paint line outlet valves. Close the bleed valves.

4.4.29 Remove the paint buckets from underneath the Gantry and LFN systems.

4.4.30 Using a wet brush, gently clean the ends of the paint guns to remove paint and other deposits.

4.5 System Power Down

4.5.1 Turn off the Gantry power supply, located on the bottom of the Control Cabinet. (Figure 2.8.1)

4.5.2 Turn off the Solenoid Interface boards, located on the top shelf of the Control Cabinet. (Figure 2.8.2)

4.5.3 Turn off the power transformer, located on the bottom of the Control Cabinet. (Figure 2.8.1)

4.5.4 Turn off the hand controller by pressing the left switch on the top of the unit.

4.5.5 Stop the compressor by placing the ignition switch in the 'OFF' position.

4.5.6 Turn the power conditioner off. Turn off the low voltage power supply. Turn off the fans. Turrn off the lights.

4.5.7 Place all circuit breakers in the 'OFF" position. Turn the generator ignition switch to the OFF position.

! Warning !

Failure to remove all loads connected to generator may result in damage to electrical loads and/or damage to generator unit.

4.5.8 Disconnect all cables from the Hand Controller. Stow the cables and Hand Controller.

! Warning !

Do not hold hand controller by cables. Do not pull on cables at the cable plug. Failure to observe these precautions will result in cable failure.

4.5.9 Remove gantry workframe legs. Replace all floor and wall panels. Close all doors.

4.5.10 System shutdown is complete.

5. Troubleshooting

5.1 Introduction

This section describes the troubleshooting procedures. For troubleshooting information pertaining to the electric generator and air compressor, please see the manufacturer's manual.

5.2 Troubleshooting Table

Symptom	Cause	Solution
Blinking green light on hand controller.	System is operating normally.	
The paint pump is making a 'chugging' sound.	The intake siphon screen is clogged.	Re-mix paint to remove solids and clean the intake siphon screen.
	Air is in the pump.	Re-prime the pump. See section 2.9.
	Air is in the paint lines.	Re-prime the paint lines. See section 2.10.
	Paint level is low.	Replenish paint.
	Pump is damaged and/or worn out.	Rebuild pump using approved Binks rebuild kit.
Paint pressure is inadequate.	The intake siphon screen is clogged.	Re-mix paint to remove solids and clean the intake siphon screen.
	Air is in the pump.	Re-prime the pump. See section 2.9.
	Air is in the paint lines.	Re-prime the paint lines. See section 2.10.
	Paint level is low.	Replenish paint.
Spray pattern is too wide and paint looks watery.	Water is in the paint.	Run the test pattern a few times to flush the water from the lines. If pattern does not improve, the paint should be replaced. Re-prime the pump (section 2.9) and re-prime the paint lines (section 2.10).
Improper spray pattern.	Paint pressure is too low.	Adjust paint pressure.
	Nozzle is clogged.	Clean the nozzle.

Yellow light on hand controller is illuminated.	Low power detected, operations may be unreliable. Momentary communications fault.	Reduce power load on generator. Readjust voltage levels on generator. Source of high power radio interference nearby. Relocate to another location or cease using radio transmissions Restart all electronic systems.
Error messages on hand controller and Yellow light on hand controller is illuminated.	Low power detected. Computer communications fault.	Reduce power load on generator. Readjust voltage levels on generator. Source of high power radio interference nearby. Relocate to another location or cease using radio transmissions.
Gantry does not respond. 'EP' is not displayed on gantry controller LED panel.	Gantry controller detected an operating fault.	Reset the gantry controller by pressing the 'RESET' button on the front panel. Power cycle the gantry controller and power cycle the hand controller.
Automatic modes unavailable.	Electronics/controller fault.	Contact Technical Support at (916) 752-4180.

Appendix A

System Checklists

The Premark Maker System Checklist

Supplies

Verify	Quantity	Item
◊	1 roll (20 feet)	Roofing material
◊	10 gallons	Water
◊	3	5 gallon buckets with lids
◊	1	2.5 gallon bucket (low height)
◊	as needed	gasoline
◊	5 gallons	White paint (Pervo #4773A) [for 25 targets, adjust accordingly]
◊	5 gallons	Black paint (Pervo #4775A) [for 25 targets, adjust accordingly]
◊	as needed	SAE 30 engine oil
◊	1	Paint mixing device
◊	1	Paint mixing device actuator
◊	3	Small scrub brushes
◊	1	Dental pick
◊	4	#1380 Binks spray tips
◊	2	#1180 Binks spray tips
◊	6	Binks nozzle filters
◊	6	Spray nozzle assemblies
◊	2	7/8" open wrenches
◊	1	9/32" socket
◊	1	Socket ratchet
◊	1	Retractable cutting knife
◊	as needed	towels or rags
◊	1	Measuring device (ruler)

Startup Checklist

Item Number	Item Complete	Manual Sections	Tasks
0	◊		Review all warnings and notices
1	◊	Section 1	Identify all equipment locations
2	◊		Check gasoline level.
3	◊	Section 2.2.1 - 2.2.7	Open all doors and install gantry legs.
4	◊	Section 2.2.8	Check oil levels
5	◊	Section 2.3	Start generator and turn circuit breakers on.
6	◊	Section 2.4	Turn on paint heating system.
7	◊	Section 2.5	Start the air compressor.
8	◊	Section 2.6	Mix the paint.
9	◊	Section 2.7	Setup the hand controller.
10	◊	Section 2.8	Turn all subsystems on.
11	◊	Section 2.9	Prime the paint pumps.
12	◊	Sections 2.10	Prime the paint lines.
13	◊	Section 2.11	Install and adjust the paint nozzle.

Operational Checklist for premark

Item Number	Item Complete	Manual Sections	Tasks
1	◊	Section 3.2.1	Establish proper target location.
2	◊	Section 3.2.1	Deactivate paint mixers.
3	◊	Section 3.2.2	Activate paint pumps.
4	◊	Section 3.2.3 - 3.2.4	Select target option from menu.
5	◊	Section 3.2.5	Deactivate paint pumps.
6	◊	Section 3.2.5	Reactivate paint mixers.
7	◊		Proceed to next target location.

Operational Checklist for abbreviated premark

Item Number	Item Complete	Manual Sections	Tasks
1	◊	Section 3.3.1	Establish proper target location.
2	◊	Section 3.3.1	Deactivate paint mixers.
3	◊	Section 3.3.2	Activate paint pumps.
4	◊	Section 3.3.3 - 3.3.4	Select target option from menu.
5	◊	Section 3.3.5	Deactivate paint pumps.
6	◊	Section 3.3.5	Reactivate paint mixers.
7	◊		Proceed to next target location.

Cleanup checklist

Item Number	Item Complete	Manual Sections	Tasks
1	◊	Section 4.2	Remove nozzles and clean components.
2	◊	Section 4.3.1	Remove paint cans.
3	◊	Section 4.3.2 - 4.3.14	Clean paint pumps.
4	◊	Section 4.4	Clean paint hoses.
5	◊	Section 4.5.1 - 4.5.7	System power down.
6	◊	Section 4.5.8	Disconnect all cables and stow.
7	◊	Section 4.5.9	Close all access ports.
8	◊		System shutdown complete.

Appendix B

Control System Program Code Listing

```
 /*=================================================================
PROGRAM
  premark.c

SYNPOSIS
  this program basically causes a z-world controller to act like
  a teach pendant, and a controller to an iai gantry robot.
  the teach pendant part cannot be taught. not yet at least.
  the program will do both a large premark and an abbreviated one.
  this program can also control the pumps and the mixers.

USES
  this program uses all global variables to minimize the stack usage.
  the important ones are commented below.

  - StatusWords: this contains the status word returned by the iai
                 controller. this is all text. see manual for meaning
                 of the characters.
  - ActPos: the actuator position in 0.01 mm units. this is a number.
  - WantedActPos: the requested actuator position in 0.01 mm units. this
                  is a number.
  - manual, jog: if in these modes they are defined as ON.
  - TotalPoints: total number of points uploaded to the controller.
  - acceleration, velocity: desired acceleration (0.01G units) and
                            desired velocity (mm/sec)
  - Input, Output: input and output string for serial communications
  - InCount, OutCount: the length of the input and output string.
  - increment: increment to move in increment mode (0.01 mm units)
  - AxesPattern, Direction: axes pattern to move and direction. see manual
                            for the exact character for the desired axis
                            pattern.
  - SolndPattern: expand to binary and negate and you'll get an idea of
                  which solenoids are open.


AUTHOR
  P.W. Wong

VERSION
  0.10 09/09/93  proved serial communications possible to iai controller
  0.20 09/10/93
  0.30 09/13/93  implemented keypad decoding
  0.40 09/14/93  implemented home command
  0.50 09/15/93  implemented control over acceleration and velocity
  0.60 09/16/93  implemented increment mode
  0.70 09/17/93  implemented jog mode
  0.80 09/20/93  cleaned up jog mode commands.
  0.90 09/22/93  implemented the automatic modes
  0.91 09/30/93  fixed automatic modes to use robot path planning
  0.99 12/15/93  final prototype code
  1.00 06/23/94  final revised code. incorporate usability changes.
  2.00 05/09/95  revised X coordinates and improved position tracking
                 algorithm
  2.10 05/10/95  added communication timeouts
  2.20 05/12/95  added watchdog timer
  2.30 05/24/95  added NMI power fail warning
  2.40 05/28/95  enhanced emergency stop

THINGS TO DO
  - fully implement downloadable paths to the iai memory card
  - get better coordinates for the premarks
  - tighten up the timing of the guns

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

#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 VERSION 2.4
#define MONTH 5
#define DAY 28
#define YEAR 95

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 */
  } Commands[] = {
     {"?99IS@@\r\n\0", 9, 24, 0},         /* 0: status, page 8 */
     {"?99RP@@\r\n\0", 9, 42, 0},         /* 1: actuator position, page 9 */
     {"?99/7  @@\r\n\0", 11, 9, 0},       /* 2: servo on/off, page 11 */
     {"?99/8 @@\r\n\0", 10, 9, 0},        /* 3: home, page 12 */
     {"?99/A30000000000000000@@\r\n\0", 25, 9, 0},
                                          /* 4: Linear motion, page 13 */
     {"?99/B@@\r\n\0", 10, 9, 0},         /* 5: stop actuator, page 17 */
     {"?99/900000@@\r\n\0", 14, 9, 0},    /* 6: jog actuator, page 18 */
     {"?99/D0000@@\r\n\0", 13, 9, 0},     /* 7: set acceleration, page 22 */
     {"?99/P@@\r\n\0", 0, 10, 1},         /* 8: set parameter, page 33 */
     {"?99PR@@\r\n\0", 10, 0, 1},         /* 9: get parameters, page 34 */
     {"?99!!@@\r\n\0", 9, 0, 0},          /* 10: reset controller, page 38 */
     {"/99  @@\r\n\0", 9, 0, 0},          /* 11: exec. command */
     {"?99/B @@\r\n\0", 10, 9, 0},        /* 12: stop motion */
     {"?99/H000000010000@@\r\n\0", 21, 9, 0},
                                          /* 13: path motion */
     {"?99/0000010000001000000000000@@\r\n\0", 34, 9, 0}};
                                          /* 14: load point data */

struct PlayList
  {
    int AccelWanted;         /* acceleration desired - 0.01G units */
    int VeloWanted;          /* velocity desired - mm/sec */
    long int Actuator1Pos;   /* actuator 1 position - 0.01 mm units */
    long int Actuator2Pos;   /* actuator 2 position - 0.01 mm units */
    char EEPosition;         /* end effector position - 0x4 or 0x0 */
    char gun1;               /* 0x1 to open - black gun */
    char gun2;               /* 0x8 to open - white gun */
    char gun3;               /* 0x2 to open - black gun */
  };


struct PlayList Xmark1[] = {
    {50, 275, 45577, 70123, 0x1, 0x0, 0x0, 0x0},
    {50, 275, 61756, 54943, 0x1, 0x0, 0x0, 0x0}, /* this is the bottom */
    {50, 100, 70824, 46075, 0x1, 0x0, 0x0, 0x0},
    {50, 100, 108220, 8480, 0x1, 0x1, 0x1, 0x1},
    {50, 100, 109220, 7480, 0x1, 0x0, 0x1, 0x1},
    {50, 100, 112220, 4480, 0x1, 0x0, 0x0, 0x0},
    {50, 100, 61756, 62156, 0x0, 0x0, 0x0, 0x0},
    {50, 100, 71824, 72224, 0x0, 0x0, 0x0, 0x0},
    {50, 100, 108220, 108620, 0x0, 0x1, 0x1, 0x1},
    {50, 100, 109220, 109620, 0x0, 0x0, 0x1, 0x1},
    {50, 100, 112220, 112620, 0x0, 0x0, 0x0, 0x0},
    {50, 100, 67224, 49475, 0x1, 0x0, 0x0, 0x0},
    {50, 275, 42575, 74524, 0x1, 0x0, 0x0, 0x0},  /* top half */
/*    {50, 275, 46975, 70124,   0x1, 0x1, 0x0, 0x1}, */
```

```
        {50, 100, 9479, 107620, 0x1, 0x1, 0x1, 0x1},
        {50, 100, 8479, 108620, 0x1, 0x0, 0x1, 0x1},
        {50, 100, 5479, 111620, 0x1, 0x0, 0x0, 0x0},
        {50, 100, 45577, 71123, 0x0, 0x0, 0x0, 0x0},
        {50, 100, 54568, 54968, 0x0, 0x0, 0x0, 0x0},
        {50, 100, 46375, 45875, 0x0, 0x0, 0x0, 0x0},
        {50, 100, 9479,   9480, 0x0, 0x1, 0x1, 0x1},
        {50, 100, 8479,   8480, 0x0, 0x0, 0x1, 0x1},
        {50, 100, 5479,   5480, 0x0, 0x0, 0x0, 0x0},
        {50, 275, 71000, 40000, 0x1, 0x0, 0x0, 0x0},  /* this is the center */
        {50, 275, 67224, 49475, 0x1, 0x0, 0x0, 0x0},
        {50, 275, 65756, 50943, 0x1, 0x0, 0x0, 0x0},
        {50, 300, 63756, 52943, 0x1, 0x1, 0x0, 0x1},
        {50, 300, 59868, 57831, 0x1, 0x1, 0x1, 0x1},
        {50, 300, 49577, 67123, 0x1, 0x1, 0x0, 0x1},
        {50, 300, 48577, 68123, 0x1, 0x0, 0x0, 0x0},
        {50, 200, 45577, 71123, 0x1, 0x0, 0x0, 0x0},
        {50, 200, 36000, 80000, 0x0, 0x0, 0x0, 0x0},
        {50, 200, 73224, 73624, 0x0, 0x0, 0x0, 0x0},
        {50, 200, 50577, 50977, 0x0, 0x1, 0x0, 0x1},
        {50, 200, 41577, 41977, 0x0, 0x0, 0x0, 0x0},
        {50, 200, 36177, 47377, 0x0, 0x0, 0x0, 0x0}, /* sides of the square */
/*      {50, 200, 62824, 74024, 0x0, 0x1, 0x0, 0x0}, */
        {50, 200, 63024, 75824, 0x0, 0x1, 0x0, 0x0},
        {50, 200, 64024, 76824, 0x0, 0x0, 0x0, 0x0},
        {50, 200, 73224, 73624, 0x0, 0x0, 0x0, 0x0},
        {50, 200, 73624, 63224, 0x0, 0x0, 0x0, 0x0},
/*      {50, 200, 46977, 36577, 0x0, 0x0, 0x0, 0x1}, */
        {50, 200, 45177, 34777, 0x0, 0x0, 0x0, 0x1},
        {50, 200, 44177, 33777, 0x0, 0x0, 0x0, 0x0},
        {50, 200, 80000, 44000, 0x1, 0x0, 0x0, 0x0},
        {50, 200, 71000, 40000, 0x1, 0x0, 0x0, 0x0},
/*      {50, 200, 55356, 50543, 0x1, 0x0, 0x0, 0x1}, */
        {50, 200, 54356, 51543, 0x1, 0x0, 0x0, 0x1},
        {50, 200, 44177, 61723, 0x1, 0x0, 0x0, 0x0},
        {50, 200, 40177, 65723, 0x1, 0x0, 0x0, 0x0},
        {50, 200, 50977, 76523, 0x1, 0x0, 0x0, 0x0},
/*      {50, 200, 72156, 55343, 0x1, 0x1, 0x0, 0x0}, */
        {50, 200, 73956, 53543, 0x1, 0x1, 0x0, 0x0},
        {50, 200, 76156, 51343, 0x1, 0x0, 0x0, 0x0},
        {50, 100, 60672, 59315, 0x0, 0x0, 0x0, 0x0},
        {00,  00,    00,    00, 0x0, 0x0, 0x0, 0x0}};


struct PlayList Xmark[] = {
        {50, 400, 80000, 30023,   0x1, 0x0, 0x0, 0x0},
        {50, 275, 71000, 40000, 0x1, 0x0, 0x0, 0x0},  /* this is the center */
/*      {50, 275, 67224, 49475, 0x1, 0x0, 0x0, 0x0}, */
        {50, 275, 69076, 47625, 0x1, 0x0, 0x0, 0x0},
        {50, 300, 65368, 51332, 0x1, 0x0, 0x1, 0x0},
        {50, 300, 49577, 67123, 0x1, 0x0, 0x0, 0x0},
/*      {50, 200, 47577, 68123, 0x1, 0x0, 0x0, 0x0}, */
        {50, 200, 45577, 71123, 0x1, 0x0, 0x0, 0x0},  /* 40996 diagonal length */

        {50, 275, 117220, 480, 0x1, 0x0, 0x0, 0x0}, /* this is the bottom */
        {50, 275, 84224, 33475, 0x1, 0x1, 0x1, 0x1},
/*      {50, 275, 70224, 47475, 0x1, 0x0, 0x0, 0x0},  */ /* 41996 */
        {50, 275, 63224, 54475, 0x1, 0x0, 0x0, 0x0},

/*      {50, 275, 61756, 54943,   0x1, 0x0, 0x0, 0x0},
        {50, 100, 70224, 49475,   0x1, 0x0, 0x0, 0x0},
        {50, 100, 111220, 6480,   0x1, 0x1, 0x1, 0x1},
        {50, 100, 112220, 5480,   0x1, 0x0, 0x1, 0x1},
        {50, 100, 117220, 480,   0x1, 0x0, 0x0, 0x0}, */

        {50, 100, 117220, 117620, 0x0, 0x0, 0x0, 0x0},  /* 41996 */
        {50, 100, 84224, 84624,   0x0, 0x1, 0x1, 0x1},
/*    {50, 100, 68224, 68624,   0x0, 0x0, 0x0, 0x0}, */
    {50, 100, 60224, 60624,   0x0, 0x0, 0x0, 0x0},

/*    {50, 100, 68224, 68624,   0x0, 0x0, 0x0, 0x0},
    {50, 100, 111220, 111620, 0x0, 0x1, 0x1, 0x1},
    {50, 100, 112220, 112620, 0x0, 0x0, 0x1, 0x1},
    {50, 100, 117220, 117620, 0x0, 0x0, 0x0, 0x0}, */

    {50, 100, 37224,  37475,  0x1, 0x0, 0x0, 0x0},

    {50, 275, 52075, 64124,   0x1, 0x0, 0x0, 0x0}, /* top half */
    {50, 100, 14479, 101620,   0x1, 0x1, 0x1, 0x1},
/*    {50, 100, 3479, 111620,   0x1, 0x0, 0x0, 0x0}, */
    {50, 100,  479, 115620,   0x1, 0x0, 0x0, 0x0},

    {50, 100, 45577, 71123,   0x0, 0x0, 0x0, 0x0},
    {50, 100, 54568,  54968,  0x0, 0x0, 0x0, 0x0},

    {50, 100, 52075,  51975,  0x0, 0x0, 0x0, 0x0},
    {50, 100, 13579,   13480,  0x0, 0x1, 0x1, 0x1},
    {50, 100, 3579,   3480,   0x0, 0x0, 0x0, 0x0},
    {50, 100, 579,   480,    0x0, 0x0, 0x0, 0x0},

    {50, 200, 36000, 80000, 0x0, 0x0, 0x0, 0x0},
    {50, 200, 72224, 72624, 0x0, 0x0, 0x0, 0x0},
    {50, 200, 53577, 53977, 0x0, 0x1, 0x0, 0x1},
    {50, 200, 41577, 41977, 0x0, 0x0, 0x0, 0x0},

    {50, 275, 85000, 30000, 0x1, 0x0, 0x0, 0x0},  /* this is the center */
    {50, 275, 78224, 38475, 0x1, 0x0, 0x0, 0x0},
/*    {50, 275, 65576, 51143, 0x1, 0x0, 0x0, 0x0}, */
/*    {50, 275, 60868, 56831, 0x1, 0x1, 0x0, 0x1}, */
    {50, 275, 53577, 63123, 0x1, 0x1, 0x0, 0x1},
/*    {50, 200, 48577, 67123, 0x1, 0x0, 0x0, 0x0}, */
    {50, 200, 44577, 72123, 0x1, 0x0, 0x0, 0x0},

    {50, 100, 60672, 59315, 0x0, 0x0, 0x0, 0x0},
    {00,  00,    00,    00, 0x0, 0x0, 0x0, 0x0}};

struct PlayList center[] = {
  {50, 250, 45575, 45575, 0x0, 0x0, 0x0, 0x0},
  {0,    0,    00,    00, 0x0, 0x0, 0x0, 0x0}};

struct PlayList premark[] = {
  {50, 400, 80000, 30023, 0x1, 0x0, 0x0, 0x0},
  {50, 275, 71000, 40000, 0x1, 0x0, 0x0, 0x0},  /* this is the center */
  {50, 275, 69076, 47625, 0x1, 0x0, 0x0, 0x0},
  {50, 300, 65368, 51332, 0x1, 0x0, 0x1, 0x0},
  {50, 300, 49577, 67123, 0x1, 0x0, 0x0, 0x0},
  {50, 200, 45577, 71123, 0x1, 0x0, 0x0, 0x0},  /* 40996 diagonal length */

  {50, 100, 71177, 71123,   0x0, 0x0, 0x0, 0x0},
  {50, 100, 54568,  54968,  0x0, 0x0, 0x0, 0x0},

  {50, 100, 52075,  51975,  0x0, 0x0, 0x0, 0x0},
  {50, 100, 13579,   13480,  0x0, 0x1, 0x1, 0x1},
  {50, 100, 3579,   3480,   0x0, 0x0, 0x0, 0x0},
  {50, 100, 579,   480,    0x0, 0x0, 0x0, 0x0},

  {50, 200, 36000, 80000, 0x0, 0x0, 0x0, 0x0},
  {50, 200, 72224, 72624, 0x0, 0x0, 0x0, 0x0},
  {50, 200, 53577, 53977, 0x0, 0x1, 0x0, 0x1},
  {50, 200, 41577, 41977, 0x0, 0x0, 0x0, 0x0},

  {50, 275, 85000, 30000, 0x1, 0x0, 0x0, 0x0},  /* this is the center */
  {50, 275, 78224, 38475, 0x1, 0x0, 0x0, 0x0},
  {50, 275, 53577, 63123, 0x1, 0x1, 0x0, 0x1},
    {50, 200, 44577, 72123, 0x1, 0x0, 0x0, 0x0},

    {50, 100, 60672, 59315, 0x0, 0x0, 0x0, 0x0},

    {00, 00,  00,    00,    0x0, 0x0, 0x0, 0x0}};

  char StatusWords[65];
  long int ActPos[4], WantedActPos[4], increment;
  int manual, jog, TotalPoints;
  int acceleration, velocity;
  shared char Input[101], Output[101], InCount, OutCount;
  char AxesPattern, Direction;
  char Solnd0Pattern, Solnd1Pattern;
  shared unsigned long int counter;
  shared unsigned int counter2;
  shared int CommFault;
  int led_mode;
  int ErrorNumber;
  int ErrorMode;
  int aindex, aindex1, resend;
  int index3;
  char anumber[9];

  main()

  {
    static char previous;
    static int index, index2;
    int lk_kxget();
    static int F1key, F2key, F3key, F4key;
    static int servos;

    ErrorMode = wderror();
    if (ErrorMode) CriticalError();

    outport (DCNTL, 0x30);
    counter = 0x0L;
    setdaisy (1);
    setctc (2, 1, 180, 1);

    Initialize();
    lk_printf("\x1b1");
    lk_printf("\x1be");

    F1key = F2key = F3key = F4key = OFF;
    jog = servos = OFF;
    WantedActPos[0] = WantedActPos[1] = increment = (long)0;
    acceleration = 475;
    velocity = 55;
    manual = ON;
    Solnd0Pattern = (char) 0;
    Solnd1Pattern = (char) 0;

    index = 0;
    while (index != 4)
      {
        ErrorNumber = 0;
        index = lk_kxget(0);

        if (index != -1) lk_setbeep(300);
                                       /* the switch pattern is as below:
                                          +---+---+---+---+---+---+
                                          + 4 + 8 + 12+ 16+ 20+ 24+
                                          +---+---+---+---+---+---+
                                          + 3 + 7 + 11+ 15+ 19+ 23+
                                          +---+---+---+---+---+---+
                                          + 2 + 6 + 10+ 14+ 18+ 22+
                                             +---+---+---+---+---+---+
                                             + 1 + 5 + 9 + 13+ 17+ 21+
                                             +---+---+---+---+---+---+
                                          */
switch (index) {
    case 0:
      break;
    case 1:  /* pumps on */
      Solnd1Pattern = Solnd1Pattern ^ (char)(1 << 1); /* pump 1 */
      Solnd1Pattern = Solnd1Pattern ^ (char)(1 << 2); /* pump 2 */
      outport (0x42, (char) Solnd1Pattern);
      break;
    case 2: /* end effector rotate */
      if (servos == ON)
        {
          Solnd1Pattern = Solnd1Pattern ^ (char)(1 << 0);
          outport (0x42, (char) Solnd1Pattern);
        }
      else  /* premark extend */
        {
          Solnd0Pattern = Solnd0Pattern ^ (char)(1 <<7);
          outport (0x40, (char) Solnd0Pattern);
        }
      break;
    case 3:  /* home the robot */
      Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
      outport (0x42, (char)Solnd1Pattern);
      lk_printf("\x1bp000\x1be");
      lk_printf("Please wait 1 \n");
      lk_printf("  minute to home \n");
      lk_printf("  the robot...\n");
     for (index3 = 0; index3 < 15; index3++)
       {
        lk_tdelay(1000);
        if (lk_kxget(0) == 4) EmergencyStop();
       }
      lk_printf("15..");
     for (index3 = 0; index3 < 15; index3++)
       {
        lk_tdelay(1000);
        if (lk_kxget(0) == 4) EmergencyStop();
       }
      lk_printf("30..");
     for (index3 = 0; index3 < 15; index3++)
       {
        lk_tdelay(1000);
        if (lk_kxget(0) == 4) EmergencyStop();
       }
      lk_printf("45..");
     for (index3 = 0; index3 < 15; index3++)
       {
        lk_tdelay(1000);
        if (lk_kxget(0) == 4) EmergencyStop();
       }
      servo(ON);
      Home();
      Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
      outport (0x42, (char)Solnd1Pattern);
      if (servos == OFF)
        servo(OFF);
      break;
    case 4:
     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 */
                outport (0x40, (char)0x1); /* disable the driver ports */
                outport (0x42, (char)(1<<7));
                  lk_printf("\x1b1");
                  lk_printf("\x1be");
                  lk_printf("Emergency Stop\n");
                  lk_printf("Emergency Stop\n");
                  lk_printf("Correct error. Cycle\n");
                  lk_printf("power to restart");
                  servo (OFF);
                  outport (CTC2, 3);
                  lk_led (0);
                while (1) hitwd();
                  exit(1);
                  break;
               case 5: /* mixers on */
                  Solnd0Pattern = Solnd0Pattern ^ (char)(1 << 5);
                  Solnd0Pattern = Solnd0Pattern ^ (char)(1 << 6);
                  outport (0x40, (char) Solnd0Pattern);
                  break;
               case 6:
/*                   Status(); */
                  if (servos == ON)
                    {  /* gantry down */
                      servo(ON);
                      LoadMagicMode (center);
                      AutoMagicMode (center);
                      if (servos == OFF)
                        servo(OFF);
                      Solnd1Pattern = Solnd1Pattern ^ (char) (1 << 6);
                      outport (0x42, (char)Solnd1Pattern);
                    }
                  else
                    { /* premark down */
                      Solnd1Pattern = Solnd1Pattern ^ (char)(1 << 3);
                      outport (0x42, (char)Solnd1Pattern);
                    }
                  break;
               case 7:
                  if (manual == 1)
                    {
                      lk_printf("\x1bp000");
                      lk_printf("\x1be");
                      lk_printf("Automatic Mode \n");
                      lk_printf("  - F1 X-mark\n");
                      lk_printf("  - F2 gantry test\n");
                      lk_printf("  - F3 premark");
                      manual = OFF;
                   }
                  else
                   {
                      lk_printf("\x1bp000");
                      lk_printf("\x1bc");
                      lk_printf("Manual Mode \n");
                      manual = ON;
                   }
                  break;
               case 8:
                  if (manual == ON)
                    {
                      if (jog == OFF)
                        {
                          jog = ON;
                          lk_printf("\x1b1\x1be");
                          lk_printf("Jog mode");
                    }
                  else
                    {
                      jog = OFF;
                      lk_printf("\x1b1\x1be");
                    }
                }
              break;
            case 9:
              if (servos == 1)
                {
                  servo(OFF);
                  lk_printf("\x1b1");
                  lk_printf("\x1be");
                  lk_printf("Servos disabled\n");
                  servos = OFF;
                  CommFault = 0;
                }
              else
                {
                  servo(ON);
                  lk_printf("\x1b1");
                  lk_printf("\x1be");
                  lk_printf("Servos enabled\n");
                  servos = ON;
                }
              break;
            case 10:
/*              lk_printf("\x1b1");    */ /* clear display */
/*              lk_printf("\x1be"); */
              if (servos == ON)
                {
                  servo(ON);
                  LoadMagicMode(center);
                  AutoMagicMode(center);
                  if (servos == OFF)
                    servo(OFF);
                  Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
                  outport (0x42, (char)Solnd1Pattern);
                }
              break;
            case 11:
              break;
            case 12:
              if (manual)
                {
                  if (F1key == OFF)
                    {
                      F1key = ON;
                      F2key = OFF;
                      lk_printf("\x1bp000\x1bc");
                      lk_printf("Set accl on");
                    }
                  else
                    {
                      F1key = OFF;
                      lk_printf("\x1bp000\x1bc");
                      SetAccel();
                    }
                }
              else
                {
                  lk_printf("\x1bp000");
                  lk_printf("\x1be");

                  Solnd1Pattern = Solnd1Pattern ^ (char) (1 << 6);
                  outport (0x42, (char)Solnd1Pattern);
```

```
          lk_printf("Gantry lowering \n");

          servo (ON);
          LoadMagicMode(Xmark);
          lk_printf("Loading points \n");
          lk_printf ("In motion  ");
          for (index2 = 0; index2 < 7; index2++)
            {
              lk_printf ("%d..",index2*4);
           for (index3 = 0; index3 < 4; index3++)
             {
              lk_tdelay(1000);
              if (lk_kxget(0) == 4) EmergencyStop();
             }
             }
          Solnd1Pattern = Solnd1Pattern ^ (char) (1 << 6);
          outport (0x42, (char)Solnd1Pattern);

          lk_printf("\x1bp000");
          lk_printf("\x1be");

          AutoMagicMode(Xmark);

          lk_printf("\x1bp000");
          lk_printf("\x1be");
          lk_printf("Gantry rising \n");
          Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
          outport (0x42, (char)Solnd1Pattern);
          lk_printf ("In motion ");
          for (index2 = 0; index2 < 6; index2++)
            {
              lk_printf ("%d..", index2*5);
           for (index3 = 0; index3 < 5; index3++)
             {
              lk_tdelay(1000);
              if (lk_kxget(0) == 4) EmergencyStop();
             }
            }
          Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
          outport (0x42, (char)Solnd1Pattern);
          if (servos == OFF)
            servo (OFF);
        }
      break;
    case 13:
      if (manual)
        {
          if (jog)
            {
              Direction = '1';
              AxesPattern = '2';
              JogGant();
              Direction = '0';
              AxesPattern = '1';
              JogGant();
            }
          else
            {
              WantedActPos[1] = ActPos[1] + increment;
              WantedActPos[0] = ActPos[0] - increment;
              if (WantedActPos[0] < (long)0)  WantedActPos[0] = (long)0;
              MoveGant();
            }
        }
      break;
    case 14:
      if (manual)
```

```
        {
          if (jog)
            {
              Direction = '1';
              AxesPattern = '2';
              JogGant();
            }
          else
            {
              WantedActPos[1] = ActPos[1] + increment;
              WantedActPos[0] = ActPos[0];
              MoveGant();
            }
        }
      break;
    case 15:
      if (manual)
        {
          if (jog)
            {
              Direction = '1';
              AxesPattern = '2';
              JogGant();
              Direction = '1';
              AxesPattern = '1';
              JogGant();
            }
          else
            {
              WantedActPos[1] = ActPos[1] + increment;
              WantedActPos[0] = ActPos[0] + increment;
              MoveGant();
            }
        }
      break;
    case 16:
      if (manual)
        {
          if (F2key == OFF)
            {
              F2key = ON;
              if (F1key)
                SetAccel();
              F1key = OFF;
              lk_printf("\x1bp000\x1bc");
              lk_printf("Set velo on");
            }
          else
            {
              F2key = OFF;
              lk_printf("\x1bp000\x1bc");
            }
        }
      else
        {
          LoadMagicMode(premark);
          AutoMagicMode(premark);
        }
      break;
    case 17:
      if (manual)
        {
          if (jog)
            {
              Direction = '0';
              AxesPattern = '1';
              JogGant();
          }
        else
          {
            WantedActPos[0] = ActPos[0] - increment;
            if (WantedActPos[0] < (long)0)  WantedActPos[0] = (long)0;
            WantedActPos[1] = ActPos[1];
            MoveGant();
          }
      }
    break;
  case 18:
    StopJogGant();
    break;
  case 19:
    if (manual)
      {
        if (jog)
          {
            Direction = '1';
            AxesPattern = '1';
            JogGant();
          }
        else
          {
            WantedActPos[0] = ActPos[0] + increment;
            WantedActPos[1] = ActPos[1];
            MoveGant();
          }
      }
    break;
  case 20:
    if (manual)
      {
        if (F1key)
          acceleration += 10;
        else
          if (F2key)
            velocity += 5;
          else
            increment += (long)1000;
      }
    else
      {
        MakePremark();
      }
    break;
  case 21:
    if (manual)
      {
        if (jog)
          {
            Direction = '0';
            AxesPattern = '2';
            JogGant();
            Direction = '0';
            AxesPattern = '1';
            JogGant();
          }
        else
          {
            WantedActPos[1] = ActPos[1] - increment;
            if (WantedActPos[1] < (long)0)  WantedActPos[1] = (long)0;
            WantedActPos[0] = ActPos[0] - increment;
            if (WantedActPos[0] < (long)0)  WantedActPos[0] = (long)0;
            MoveGant();
          }
      }
  break;
case 22:
  if (manual)
    {
      if (jog)
        {
          Direction = '0';
          AxesPattern = '2';
          JogGant();
        }
      else
        {
          WantedActPos[1] = ActPos[1] - increment;
          WantedActPos[0] = ActPos[0];
          if (WantedActPos[1] < (long)0)  WantedActPos[1] = (long)0;
          MoveGant();
        }
    }
  break;
case 23:
  if (manual)
    {
      if (jog)
        {
          Direction = '0';
          AxesPattern = '2';
          JogGant();
          Direction = '1';
          AxesPattern = '1';
          JogGant();
        }
      else
        {
          WantedActPos[1] = ActPos[1] - increment;
          if (WantedActPos[1] < (long)0)  WantedActPos[1] = (long)0;
          WantedActPos[0] = ActPos[0] + increment;
          MoveGant();
        }
    }
  break;
case 24:
  if (manual)
    {
      if (F1key)
        {
          acceleration -= 10;
          if (acceleration < 0) acceleration = 0;
        }
      else
        if (F2key)
          {
            velocity -= 5;
            if (velocity < 0) velocity = 0;
          }
        else
          {
            increment -= (long) 1000;
            if (increment < 0) increment = 0;
          }
    }
  else
    {
      Solnd1Pattern = Solnd1Pattern ^ (char) (1 << 6);
      outport (0x42, (char)Solnd1Pattern);
      LoadMagicMode(Xmark1);
      lk_tdelay(25000);
      Solnd1Pattern = Solnd1Pattern ^ (char) (1 << 6);
```

```
                outport (0x42, (char)Solnd1Pattern);

                AutoMagicMode(Xmark1);
                Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
                outport (0x42, (char)Solnd1Pattern);
                lk_tdelay(30000);
                Solnd1Pattern = Solnd1Pattern ^ (char) (1<<5);
                outport (0x42, (char)Solnd1Pattern);

               }
          }
         if (manual == ON)
          {
            ActPosition();
            DispPosition();
          }
         index = -1;
        }
   }

/*====================================================================
PROECUDURE
EmergencyStop

SYNPOSIS
This is for the emergency stop.
====================================================================*/

EmergencyStop()

{
  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 */
  outport (0x40, (char)0x1); /* disable the driver ports */
  outport (0x42, (char)(1<<7));
  lk_printf("\x1b1");
  lk_printf("\x1be");
  lk_printf("Emergency Stop\n");
  lk_printf("Emergency Stop\n");
  lk_printf("Correct error. Cycle\n");
  lk_printf("power to restart");
  servo (OFF);
  outport (CTC2, 3);
  lk_led (0);
  while (1) hitwd();
  exit(1);
}

/*====================================================================
PROCEDURE
NMI_int

SYNPOSIS
This intercepts the low power warning and makes the yellow light
come on.
====================================================================*/
#JUMP_VEC NMI_VEC NMI_int

interrupt retn NMI_int()
{
  lk_printf("power fail\n");
  CommFault = 1;
  while (1) if (!powerlo()) return;
)

/*=====================================================================
PROCEDURE
CriticalError

SYNPOSIS
This section is only active after a reset by the watchdog timer. It
only displays an error message and the error location
=====================================================================*/

CriticalError ()

{
  outport (CTC2, 3);
  lk_kxinit();
  lk_init();
  lk_init_keypad();

  if (CommFault)
    lk_led (1);
  else
    lk_led(0);

  lk_printf("\nController Critical\n");
  if (CommFault)
    {
      lk_printf ("  Error. Comm or pwr\n");
      lk_printf ("  failure.");
    }
  else
    lk_printf("  Error #%d Mode #%d\n", ErrorNumber, ErrorMode);

  /* lk_printf("Error2 #%d", ErrorNumber2); */
  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 */


  outport (0x40, (char)0x1); /* disable the driver ports */
  outport (0x42, (char)(1<<7));

  while (1)
    hitwd();
}

/*=====================================================================
PROCEDURE
ccc

SYNPOSIS
This is a heart beat timer.
=====================================================================*/

#INT_VEC CTC2_VEC ccc

interrupt reti int ccc()
{
  counter = counter + 1L;
  counter2 = counter2 + 1;
  if (counter2 == 50)
    {
      counter2 = 0;
      if (wderror() == 0) hitwd();
      if (CommFault == 1)
        {
          if (led_mode == 3)
            led_mode = 1;
          else
            led_mode = 3;
        }
      else
        {
          if (led_mode == 0)
            led_mode = 2;
          else
            led_mode = 0;
        }
      lk_led (led_mode);
    }
}


/*=====================================================================
PROCEDURE
MakePremark

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

MakePremark()

{
  ErrorNumber = 1;
  lk_printf("\x1be");
  lk_printf("Premark in progress \n");
  Solnd1Pattern = Solnd1Pattern | (1<<3);
  Solnd0Pattern = Solnd0Pattern | (1<<4);
  outport (0x40, (char)Solnd0Pattern);
  outport (0x42, (char)Solnd1Pattern);
  lk_printf("Guns on, frame down\n");
  lk_tdelay(3000);
  Solnd0Pattern = Solnd0Pattern | (1<<7);
  outport (0x40, (char)Solnd0Pattern);
  lk_printf("Guns in motion");
  lk_tdelay(6000);
  Solnd1Pattern = Solnd1Pattern ^ (1<<3);
  Solnd0Pattern = Solnd0Pattern ^ (1<<4);
  Solnd0Pattern = Solnd0Pattern ^ (1<<7);
  outport(0x40, (char)Solnd0Pattern);
  outport(0x42, (char)Solnd1Pattern);
  manual=ON;
  lk_printf("\x1be");
}

/*=====================================================================
PROCEDURE
  LoadMagicMode

SYNPOSIS
  this procedure loads the point data into the iai robot controller.
  basically, you need to pass a data structure of the points you want
  to move to. end the list with 0 acceleration

USES
  WhichOne - pointer to the data structure
=====================================================================*/
```

```
LoadMagicMode(WhichOne)

struct PlayList WhichOne[];

{
  static int index, index1, offset;
  static char value[10];

  index = 0;

  while (WhichOne[index].AccelWanted != 0)
    {
      ClearBuf(Output, 40);
      ClearBuf(Input, 40);
      strcpy(Output, Commands[14].Cmmnd);    /* load point command formation */

      ClearBuf(value, 6);
      itoa((index+1), value);                /* fill in point offset */
      index1 = itoan((index+1))-1;
      for (offset = 0; index1 >= 0; index1--, offset++)
        Output[8-offset] = value[index1];

      ClearBuf(value, 6);
      ltoa(WhichOne[index].Actuator1Pos, value); /* fill in actuator 1 part */
      index1 = ltoan(WhichOne[index].Actuator1Pos)-1;
      for (offset = 0; index1 >= 0; index1--, offset++)
        Output[15-offset] = value[index1];

      ClearBuf(value, 6);
      ltoa(WhichOne[index].Actuator2Pos, value); /* fill in actuator 2 part */
      index1 = ltoan(WhichOne[index].Actuator2Pos)-1;
      for (offset = 0; index1 >= 0; index1--, offset++)
        Output[22-offset] = value[index1];

      InCount = (char) Commands[14].RespLength;
      ser_rec_s0(Input, &InCount);
      OutCount = (char) Commands[14].ComLength;
      ser_send_s0(Output, &OutCount);
/*      printf("load mode: %s", Output); */
      while (OutCount != (char) 0);  /* wait for transmission to finish */

      counter = 0L;
      ErrorNumber = 2;
      do
        {
              if (counter > 20L)
                {
                  Input[0] = '%';
                  ser_kill_s0();
                  break;
                }
            }
      while (InCount != (char) 0);   /* wait for reception to finish */

      if(Input[0] == '%')
        {
          lk_printf("\x1be");
          lk_printf("Set Points:\n");
          lk_printf(" Transmission error\n");
          lk_printf(" %s", Input);
         /* printf("Set Points: Transmission error %s", Input); */
              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 */

            outport (0x40, (char)0x1); /* disable the driver ports */
            outport (0x42, (char)(1<<7));
            servo (OFF);
            outport (CTC2, 3);
         lk_led (0);

            while (1) hitwd();
         exit(1);
       }

     Output[0] = '/';             /* Convert to command sequence */
     Output[5] = '@';             /* and execute the command */
     Output[6] = '@';
     Output[7] = '\r';
     Output[8] = '\n';
     OutCount = (char) 9;

     InCount = (char) 9;
     ser_rec_s0 (Input, &InCount);
     ser_send_s0(Output, &OutCount);
     while (OutCount != (char) 0);  /* wait for transmission to finish */
     counter = 0L;
     ErrorNumber = 3;
     do
           {
            if (counter > 4L)
              {
                ser_kill_s0();
                break;
              }
           }
     while (InCount != (char) 0);
     index++;
    }
  TotalPoints = index;
}

/*======================================================================
PROCEDURE
  AutoMagicMode

SYNPOSIS
  this procedure executes the point data load into the iai robot
  controller previously by the LoadMagicMode procedure. you must,
  however,  pass a data structure of the points you want
  to move to in order to properly position the opening/closing of the
  guns and the rotation of the end effector. as always you must
  end the list with 0 acceleration

USES
  WhichOne - pointer to the data structure
======================================================================*/

AutoMagicMode(WhichOne)

struct PlayList WhichOne[];

{
  static int index, index1, offset;
  int lk_kxget(), itoan();
  long int labs();
  static char Old1SolndPattern, value[8];
  static float NewError, OldError;
  static float xerror, yerror;
```

```
index = 0;
NewError = 0.0;
OldError = 0.0;

ClearBuf(Output, 40);
ClearBuf(Input, 40);
strcpy(Output, Commands[13].Cmmnd);   /* execute point command formation */

ClearBuf(value, 6);
itoa(WhichOne[0].VeloWanted, value);  /* fill in velocity part */
index1 = itoan(WhichOne[0].VeloWanted)-1;
for (offset = 0; index1 >= 0; index1--, offset++)
  Output[8-offset] = value[index1];
ClearBuf(value, 6);
itoa(TotalPoints, value);            /* fill in point limits */
index1 = itoan(TotalPoints)-1;
for (offset = 0; index1 >= 0; index1--, offset++)
  Output[16-offset] = value[index1];

InCount = (char) Commands[13].RespLength;
ser_rec_s0(Input, &InCount);
OutCount = (char) Commands[13].ComLength;
ser_send_s0(Output, &OutCount);
while (OutCount != (char) 0);  /* wait for transmission to finish */
counter = 0L;
ErrorNumber = 4;
do
  {
    if (counter > 20L)
          {
            Input[0] = '%';
            ser_kill_s0();
            break;
          }
  }
while (InCount != (char) 0);   /* wait for reception to finish */

if(Input[0] == '%')
  {
    lk_printf("\x1be");
    lk_printf("Run Points:\n");
    lk_printf(" Transmission error\n");
    lk_printf(" %s", Input);
   /*  printf("Run Points: Transmission error %s", Input); */
    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 */

    outport (0x40, (char)0x1); /* disable the driver ports */
    outport (0x42, (char)(1<<7));
    servo (OFF);
    outport (CTC2, 3);
    lk_led(0);

    while (1) hitwd();
    exit(1);
  }

Output[0] = '/';              /* Convert to command sequence */
Output[5] = '@';              /* and execute the command */
Output[6] = '@';
```

```
Output[7] = '\r';
Output[8] = '\n';
OutCount = (char) 9;

ser_send_s0(Output, &OutCount);
while (OutCount != (char) 0);  /* wait for transmission to finish */
InCount = (char) 9;
ser_rec_s0 (Input, &InCount);
counter = 0L;
ErrorNumber = 5;
do
  {
    if (counter > 4L)
          {
            ser_kill_s0();
            break;
          }
  }
while (InCount != (char) 0);

lk_printf("\x1bp000\x1bc");
Old1SolndPattern = Solnd1Pattern;

while (WhichOne[index].AccelWanted != 0)
  {
    WantedActPos[0] = WhichOne[index].Actuator1Pos;
    WantedActPos[1] = WhichOne[index].Actuator2Pos;
    lk_printf("\x1bp000\x1bc");
    lk_printf("dErr: %6.2f \n", sqrt(NewError/10000.00));
    lk_printf("Move # %d....", index);

    Solnd1Pattern = (Old1SolndPattern | WhichOne[index].EEPosition);
    Solnd0Pattern = ((WhichOne[index].gun1 << 1) |
                      (WhichOne[index].gun2 << 2) |
                      (WhichOne[index].gun3 << 3));

    outport (0x40, (char)Solnd0Pattern);
    outport (0x42, (char)Solnd1Pattern);

    /*
      now loop until the minimum norm is reached. if the norm
      begins to increase, we are moving away from our target
      and the set point is assumed to have been reached.
    */

    ActPosition();
    xerror = (float)(WantedActPos[0] - ActPos[0]);
    yerror = (float)(WantedActPos[1] - ActPos[1]);
    NewError = (xerror * xerror) + (yerror * yerror);

    OldError = 1e10;  /* set old error to some large number */

    while (NewError <= OldError)
      {
        ActPosition();

            OldError = NewError;
            xerror = (float) (WantedActPos[0] - ActPos[0]);
            yerror = (float) (WantedActPos[1] - ActPos[1]);
            NewError = (xerror * xerror) + (yerror * yerror);

        if (lk_kxget(0) == 4)
          {
                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 */


                    outport (0x40, (char)0x1); /* disable the driver ports */
                    outport (0x42, (char)(1<<7));

                lk_printf("\x1bl");
                lk_printf("\x1be");
                lk_printf("Emergency Stop\n");
                lk_printf("Emergency Stop\n");
                lk_printf("Correct error. Cycle\n");
                lk_printf("power to restart");

                lk_led(0);

                servo(OFF);

                OutCount = (char) Commands[10].ComLength;
                ser_send_s0(Output, &OutCount);
                while (OutCount != (char) 0);  /* wait for transmission to
                                                  finish */
                outport (CTC2, 3);
                while (1) hitwd();
                    exit(1);
              }
         }

        index++;
      }
  manual = ON;
  outport (0x40, (char)Solnd0Pattern);
  outport (0x42, (char)Old1SolndPattern);
  Solnd1Pattern = Old1SolndPattern;
  lk_printf("\x1bp000\x1bc");
}

/*=====================================================================
PROCEDURE
  StopJogGant

USES
  nothing as far as I can tell

SYNPOSIS
  this procedure stops all axis of the gantry during the jog command
=====================================================================*/
StopJogGant()

{
  ClearBuf(Input, 50);
  ClearBuf(Output, 50);

  strcpy(Output, Commands[12].Cmmnd);     /* servo command formation */
  Output[5] = '3';

  InCount = (char) Commands[12].RespLength;
  ser_rec_s0(Input, &InCount);
  OutCount = (char) Commands[12].ComLength;
  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  counter = 0L;
  ErrorNumber = 6;
  do
    {
        if (counter > 20L)
              {
                Input[0] = '%';
                ser_kill_s0();
                break;
              }
      }
    while (InCount != (char) 0);   /* wait for reception to finish */

    if(Input[0] == '%')
      {
        lk_printf("\x1be");
        lk_printf("Stop Motion:\n");
        lk_printf("Transmission error\n");
        lk_printf(" %s", Input);
       /* printf("Stop Motion:Transmission error %s", Input); */
        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 */


        outport (0x40, (char)0x1); /* disable the driver ports */
        outport (0x42, (char)(1<<7));
        servo(OFF);
        outport (CTC2, 3);
        lk_led(0);

        while (1) hitwd();
        exit(1);
      }

    Output[0] = '/';              /* Convert to command sequence */
    Output[5] = '@';              /* and execute the command */
    Output[6] = '@';
    Output[7] = '\r';
    Output[8] = '\n';
    OutCount = (char) 9;

    ser_send_s0(Output, &OutCount);
    while (OutCount != (char) 0);   /* wait for transmission to finish */
    InCount = (char) 9;
    ser_rec_s0 (Input, &InCount);
    counter = 0L;
    ErrorNumber = 7;
    do
      {
        if (counter > 4L)
              {
                ser_kill_s0();
                break;
              }
      }
    while (InCount != (char) 0);
  }

  /*======================================================================
  PROCEDURE
    JogGant

  USES
    nothing as far as I can tell
```

```
      exit(1);
    }

  Output[0] = '/';              /* Convert to command sequence */
  Output[5] = '@';             /* and execute the command */
  Output[6] = '@';
  Output[7] = '\r';
  Output[8] = '\n';
  OutCount = (char) 9;

  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  InCount = (char) 9;
  ser_rec_s0 (Input, &InCount);
  counter = 0L;
  ErrorNumber = 9;
  do
    {
      if (counter > 4L)
            {
              ser_kill_s0();
              break;
            }
    }
  while (InCount != (char) 0);
}

/*=====================================================================
PROCEDURE
  SetAccel

USES
  this procedure uses the global variable acceleration to get the
   user's desired acceleration

SYNPOSIS
  this procedure sets the acceleration of the gantry according to the
  user's wishes.
=====================================================================*/
SetAccel()

{
  static char value[5];
  static int index, offset;
  int itoan();

  offset = 0;
  if (acceleration == 0) return(0);
  itoa(acceleration, value);
  index = itoan(acceleration) - 1;

  ClearBuf(Input, 50);
  ClearBuf(Output, 50);

  strcpy(Output, Commands[7].Cmmnd);     /* fill in accleration command */
  for (offset = 0; index >= 0; index--, offset++)
   {
     Output[8-offset] = value[index];
   }

  InCount = (char) Commands[7].RespLength;
  ser_rec_s0(Input, &InCount);
  OutCount = (char) Commands[7].ComLength;
  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  counter = 0L;
  ErrorNumber = 10;
  do
    {
      if (counter > 20L)
            {
              Input[0] = '%';
              ser_kill_s0();
              break;
            }
    }
  while (InCount != (char) 0);    /* wait for reception to finish */

  if(Input[0] == '%')
    {
      lk_printf("\x1be");
      lk_printf("Acceleration:\n");
      lk_printf(" Transmission error\n");
      lk_printf(" %s", Input);
     /* printf("Acceleration: Transmission error %s", Input); */

      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 */


      outport (0x40, (char)0x1);  /* disable the driver ports */
      outport (0x42, (char)(1<<7));
      servo (OFF);
      outport (CTC2, 3);
      lk_led(0);

      while (1) hitwd();
      exit(1);
    }

  Output[0] = '/';              /* Convert to command sequence */
  Output[5] = '@';              /* and execute the command */
  Output[6] = '@';
  Output[7] = '\r';
  Output[8] = '\n';
  OutCount = (char) 9;

  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);   /* wait for transmission to finish */
  InCount = (char) 9;
  ser_rec_s0 (Input, &InCount);
  counter = 0L;
  ErrorNumber = 11;
  do
    {
      if (counter > 4L)
            {
              ser_kill_s0();
              break;
            }
    }
  while (InCount != (char) 0);
}

/*======================================================================
PROCEDURE
  MoveGant
```

```
USES
  velocity - the desired velocity of the movement
  WantedActPos - array of wanted actuator positions

SYNPOSIS
  this procedure moves the gantry to the specified position.
  sets the acceleration of the gantry according to the
  user's wishes.
======================================================================*/
MoveGant()

{
  static char value[8];
  static int index, offset;
  int itoan();

  if (velocity == 0) return(0);

  ClearBuf(Input, 50);
  ClearBuf(Output, 50);
  offset = 0;

  strcpy(Output, Commands[4].Cmmnd);      /* get linear motion command */

  itoa(velocity, value);                  /* fill in velocity part */
  index = itoan(velocity)-1;
  for (offset = 0; index >= 0; index--, offset++)
    Output[8-offset] = value[index];

  ltoa(WantedActPos[0], value);           /* fill in 1st act. position */
  index = ltoan(WantedActPos[0]) - 1;

  for (offset = 0; index >= 0; index--, offset++)
    Output[14-offset] = value[index];

  ltoa(WantedActPos[1], value);           /* fill in 2nd act. position */
  index = ltoan(WantedActPos[1]) - 1;
  for (offset = 0; index >= 0; index--, offset++)
    Output[20-offset] = value[index];

  InCount = (char) Commands[4].RespLength;
  ser_rec_s0(Input, &InCount);
  OutCount = (char) Commands[4].ComLength;
  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  counter = 0L;
  ErrorNumber = 12;
  do
    {
     if (counter > 20L)
           {
             Input[0] = '%';
             ser_kill_s0();
             break;
           }
    }
  while (InCount != (char) 0);   /* wait for reception to finish */

  if(Input[0] == '%')
    {
     lk_printf("\x1be");
     lk_printf("Linear Motion:\n");
     lk_printf(" Transmission error\n");
     lk_printf(" %s", Input);
    /* printf("Linear motion: Transmission error %s", Input); */

     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 */


      outport (0x40, (char)0x1); /* disable the driver ports */
      outport (0x42, (char)(1<<7));
      servo (OFF);
      outport (CTC2, 3);
      lk_led(0);

      while (1) hitwd();
      exit(1);
    }

  Output[0] = '/';             /* Convert to command sequence */
  Output[5] = '@';             /* and execute the command */
  Output[6] = '@';
  Output[7] = '\r';
  Output[8] = '\n';
  OutCount = (char) 9;

  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  InCount = (char) 9;
  ser_rec_s0 (Input, &InCount);
  counter = 0L;
  ErrorNumber = 13;
  do
    {
      if (counter > 4L)
            {
              ser_kill_s0();
              break;
            }
    }
  while (InCount != (char) 0);
}

/*======================================================================
PROCEDURE
  servo

USES
  OnOff - the value whether on or off.

SYNPOSIS
  this procedure turns the servos on or off.
======================================================================*/
servo(OnOff)

int OnOff;

{
  ClearBuf(Input, 100);
  ClearBuf(Output, 100);

  strcpy(Output, Commands[2].Cmmnd);      /* servo command formation */
  Output[5] = '7';

  if (OnOff == 1)
    Output[6] = '1';                      /* servo on command */
  else
  Output[6] = '0';                     /* servo off command */

InCount = (char) Commands[2].RespLength;
ser_rec_s0(Input, &InCount);
OutCount = (char) Commands[2].ComLength;
ser_send_s0(Output, &OutCount);
while (OutCount != (char) 0);  /* wait for transmission to finish */
counter = 0L;
ErrorNumber = 14;
do
  {
    if (counter > 20L)
          {
            Input[0] = '%';
            ser_kill_s0();
            break;
          }
  }
while (InCount != (char) 0);   /* wait for reception to finish */

if(Input[0] == '%')
  {
    lk_printf("\x1be");
    lk_printf("Servo:\n");
    lk_printf(" Transmission error\n");
    lk_printf(" %s", Input);
   /* printf("Servo: Transmission error %s", Input); */
    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 */


    outport (0x40, (char)0x1); /* disable the driver ports */
    outport (0x42, (char)(1<<7));
    outport (CTC2, 3);
    lk_led(0);
    while (1) hitwd();
    exit(1);
  }

Output[0] = '/';             /* Convert to command sequence */
Output[5] = '@';             /* and execute the command */
Output[6] = '@';
Output[7] = '\r';
Output[8] = '\n';
OutCount = (char) 9;

ser_send_s0(Output, &OutCount);
while (OutCount != (char) 0);  /* wait for transmission to finish */
InCount = (char) 9;
ser_rec_s0 (Input, &InCount);
counter = 0L;
ErrorNumber = 15;
do
  {
    if (counter > 4L)
          {
            ser_kill_s0();
            break;
          }
  }
while (InCount != (char) 0);
```

```
)

/*=====================================================================
PROCEDURE
  DispPosition

USES
  StatusWords - the status word received from the gantry robot

SYNPOSIS
  this procedure gets the status word and interprets it. after it
  finishes interpreting it, it then displays the results on the LCD
  display.
====================================================================*/

DispPosition()

{
/*  char number[7];
  int index, index1; */
  int atoi();

/*
  printf("Status Word: %s", StatusWords);
  printf("Sup: %c Mde: %c Prg: %c Crd: %c Wrt: %c Axs: %c \n",
          StatusWords[5], StatusWords[6], StatusWords[7], StatusWords[8],
          StatusWords[9], StatusWords[10]);
  printf("Status: %c %c %c %c \n", StatusWords[14], StatusWords[15],
          StatusWords[16], StatusWords[17]);


  for (aindex = 0; aindex < 2; aindex++)
    {
      for (aindex1 = 0; aindex1 < 6; aindex1++)
        anumber[aindex1] = StatusWords[20+aindex1+(aindex*6)];
      anumber[aindex1] = '\0';
      ActPos[aindex] = atol(anumber);
    }

  printf("1:%d 2:%d 3:%d 4:%d", ActPos[0], ActPos[1],
         ActPos[2], ActPos[3]);
  printf("Mode: %d acc:%d vel:%d\n", manual, acceleration, velocity);
*/

  lk_printf("\x1b0");
  lk_printf("\x1bp100");
  lk_printf("\1bc");
  lk_printf("\x1bp100");
  lk_printf("Stat: %c %c ", StatusWords[14], StatusWords[15]);

  if (Solnd1Pattern & (char)(1<<1))
    lk_printf(":PP");
  else
    lk_printf(":pp");

  if (Solnd0Pattern & (char)(1<<5))
    lk_printf("MM");
  else
    lk_printf("mm");

  if (Solnd1Pattern & (char)(1<<0))
    lk_printf("R");
  else
    lk_printf("r");

  if (Solnd0Pattern & (char)(1<<7))
    lk_printf("T");
  else
    lk_printf("t");

  if (Solnd1Pattern & (char)(1<<3))
    lk_printf("D");
  else
    lk_printf("d");

  if ((Solnd1Pattern & (char)(1<<6)) && (Solnd1Pattern & (char)(1<<5)))
    lk_printf("E");
  else
    if (Solnd1Pattern & (char)(1<<6))
      lk_printf("D");
    else
      if (Solnd1Pattern & (char)(1 << 5))
        lk_printf("U");
      else
        lk_printf("I");

  lk_printf("\x1bp200");
  lk_printf("\1bc");
  lk_printf("\x1bp200");
  lk_printf("M:");
  if (manual)
    {
      if (jog)
        lk_printf("J ");
      else
        lk_printf("M ");
    }
  else
    lk_printf("A ");
  lk_printf("A:%3.0dV:%3.0dI:%4.0ld\n", acceleration, velocity, increment);
  lk_printf("\x1bp300");
  lk_printf("\x1bc");
  lk_printf("\x1bp300");
  lk_printf("1:%6.0ld 2:%6.0ld ", ActPos[0], ActPos[1]);
  lk_printf("\x1b1");
}

/*======================================================================
PROCEDURE
  ActPostion

USES
  StatusWords - the status word received from the gantry robot

SYNPOSIS
  this procedure gets the status word from the gantry controller.
======================================================================*/
root ActPosition()

{
  long atol();

/*  ClearBuf(Input, 100);
  ClearBuf(Output, 100);
*/
  resend = 0;
  InCount = (char) Commands[1].RespLength;   /* get status */
  ser_rec_s0(StatusWords, &InCount);
  OutCount = (char) Commands[1].ComLength;
  ser_send_s0(Commands[1].Cmmnd, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */

  counter = 0L;
  ErrorNumber = 16;
```

```
    do
      {
        if (counter > 20L)
              {
                resend = 1;
            CommFault = 1;
                ser_kill_s0();
                break;
              }
      }
    while (InCount != (char) 0);   /* wait for reception to finish */

    if (resend)
      {
        resend = 0;
        InCount = (char) Commands[1].RespLength;  /* get status */
        ser_rec_s0(StatusWords, &InCount);
        OutCount = (char) Commands[1].ComLength;
        ser_send_s0(Commands[1].Cmmnd, &OutCount);
        while (OutCount != (char) 0);  /* wait for transmission to finish */
        counter = 0L;
        ErrorNumber = 17;
        do
              {
                if (counter > 20L)
                  {
                    ser_kill_s0();
                    resend = 1;
                    break;
                  }
              }
        while (InCount != (char) 0);   /* wait for reception to finish */
      }
    if (resend)
      {
        lk_printf("\x1be");
        lk_printf("Serial\n");
        lk_printf("Receive error\n");

        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 */


        outport (0x40, (char)0x1); /* disable the driver ports */
        outport (0x42, (char)(1<<7));
        outport (CTC2, 3);
        lk_led(1);
        while (1) hitwd();
        exit(1);
      }

    for (aindex = 0; aindex < 2; aindex++)
      {
        for (aindex1 = 0; aindex1 < 6; aindex1++)
          anumber[aindex1] = StatusWords[20+aindex1+(aindex*6)];
        anumber[aindex1] = '\0';
        ActPos[aindex] = atol(anumber);
      }
  }

  /*======================================================================
PROCEDURE
  Home

USES
  nothing as far as I can tell

SYNPOSIS
  this procedure homes the robot
=====================================================================*/
Home()

{
  lk_printf("\x1bp000\x1bc\x1bp000");
  lk_printf("homing... ");
 /* printf("Homing gantry... beginning\n"); */

  ClearBuf(Input, 100);
  ClearBuf(Output, 100);

  strcpy(Output, Commands[3].Cmmnd);      /* form home command */
  Output[5] = '7';
  InCount = (char) Commands[3].RespLength;
  ser_rec_s0(Input, &InCount);
  OutCount = (char) Commands[3].ComLength;
  ser_send_s0(Output, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  counter = 0L;
  ErrorNumber = 18;
  do
    {
      if (counter > 20L)
            {
              Input[0] = '%';
              ser_kill_s0();
              break;
            }
    }
  while (InCount != (char) 0);   /* wait for reception to finish */

  if(Input[0] == '%')
    {
      lk_printf("\x1be");
      lk_printf("Home:\n");
      lk_printf(" Transmission error\n");
      lk_printf(" %s", Input);
     /* printf("Home: Transmission error %s", Input); */

      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 */


      outport (0x40, (char)0x1); /* disable the driver ports */
      outport (0x42, (char)(1<<7));
      servo (OFF);
      outport (CTC2, 3);
      lk_led(0);

      while (1) hitwd();
      exit(1);
    }
    Output[0] = '/';              /* Convert to command sequence */
    Output[5] = '@';              /* and execute the command */
    Output[6] = '@';
    Output[7] = '\r';
    Output[8] = '\n';
    OutCount = (char) 9;
    ser_send_s0(Output, &OutCount);
    while (OutCount != (char) 0);  /* wait for transmission to finish */
    InCount = (char) 9;
    ser_rec_s0(Input, &InCount);
    counter = 0L;
    ErrorNumber = 19;
    do
      {
        if (counter > 4L)
              {
                ser_kill_s0();
                break;
              }
      }
    while (InCount != (char) 0);

  /*   ClearBuf(Input, 100);
    ClearBuf(Output, 100); */

    ActPosition();
    while (!((StatusWords[14] == '7') && (StatusWords[15] == '7')
       && (StatusWords[16] == '7')))
      {
        DispPosition();
        ActPosition();
      }
    lk_printf("\x1bp000\x1bc\x1bp000");
    lk_printf("homing... done ");

   /* printf("Homing gantry... done\n"); */
  }

  /*====================================================================
  PROCEDURE
    ClearBuf

  USES
    in - input string
    count - number of nulls to use

  SYNPOSIS
    this procedure clears out the input string and makes it null.
  ====================================================================*/
  ClearBuf(in, count)

  char in[];
  int count;

  {
    static int index;

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

  Status()

  {
    static int index;
    int atoi();
  ClearBuf(Input, 50);
  ClearBuf(Output, 50);
  InCount = (char) Commands[0].RespLength;
  ser_rec_s0(Input, &InCount);
  OutCount = (char) Commands[0].ComLength;
  ser_send_s0(Commands[0].Cmmnd, &OutCount);
  while (OutCount != (char) 0);  /* wait for transmission to finish */
  counter = 0L;
  ErrorNumber = 20;
  do
    {
      if (counter > 20L)
           {
             ser_kill_s0();
             break;
           }
    }
  while (InCount != (char) 0);   /* wait for reception to finish */

  lk_printf("\x1bp000\x1be");
  if (Input[5] == '0')
    lk_printf("With memory card \n");
  else
    lk_printf("ROM - no memory card \n");

  lk_printf("Startup: ");

  switch (Input[6]) {
    case '0':
      lk_printf("Idle\n");
      break;
    case '1':
      lk_printf("Ext Strt\n");
      break;
    case '2':
      lk_printf("Teaching\n");
  }

  lk_printf("Md: ");
  if (Input[7] == '1')
    lk_printf("R P");

  switch (Input[8]) {
    case '0':
      lk_printf("No cd wrt on\n");
      break;
    case '1':
      lk_printf("Cd in wrt on\n");
      break;
    case '2':
      lk_printf("No cd wrt off\n");
      break;
    case '3':
      lk_printf("Cd in wrt off\n");
      break;
  }

  lk_printf("Axes: %c \n", Input[10]);
}

Initialize()

{
  static int index;
  int lk_kxget();

  ser_init_s0((char)4, (char) 8);  /* initialize serial port according
```

```
                                          to parameters on page 53         */
  lk_kxinit();
  lk_init();
  lk_init_keypad();
  lk_led (0);
  CommFault = 0;
  led_mode = 0;

  ClearBuf(Input, 98);
  ClearBuf(Output, 98);

  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 */
/*
  InCount = (char) Commands[10].RespLength;
  ser_rec_s0(Input, &InCount);

  OutCount = (char) Commands[10].ComLength;
  ser_send_s0(Commands[10].Cmmnd, &OutCount);
*/
/*  while (OutCount != (char) 0);*/  /* wait for transmission to finish */

  servo(OFF);

/*  while (InCount != (char) 0);*/    /* wait for reception to finish */
/*  printf("Resetting Controller \n"); */

  lk_printf("\x1bp000\x1be");
  lk_printf("The PreMark Maker\n");
  lk_printf("  Version %0.2f\n", VERSION);
  lk_printf(" (C)UCD %2d/%2d/%2d \n", MONTH, DAY, YEAR);
  lk_printf("F1 to continue! ");
  while (lk_kxget(0) != 12) hitwd();
}
```