The Reference about Rhino XR Robot System
ECE565 Robotics class
Version 2.0

Contents

1 Rhino Robot System : Overview

Rhino robot system consists of
XR-3 robotic arm,
Mark III controller,
Robot control computer,
Teach pendant.

Fig. 1 shows the total system of the Rhino XR-3 robot system.

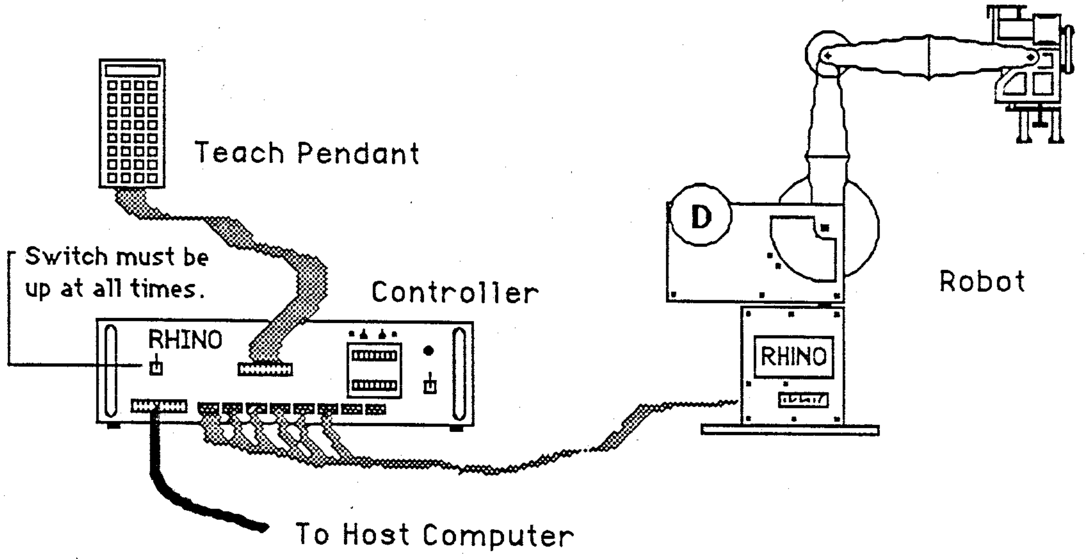


Figure 1: Overview of the Rhino XR-3 Robot system

1.1 XR-3 Robotic Arm

The XR-3 robotic arm in Fig. 2 has a jointed-spherical geometry with five degrees of freedom in the arm and wrist. the five axes of motion include rotation at the wrist, shoulder, and elbow for positioning the hand; then, flex and rotation motion in the wrist for orientation of the gripper. The motion is transferred from the axis drive motors to the joint by chain, and lever linkages. The rate of rotation of the axis drive motor is reduced by gears in the motor and the linkage between the motor and the joint.

The axis drive motors are dc servo devices with optical encoders attached to determine joint angles and arm position. The optical encoders are not visible but can be accessed by removal of the encoder covers on the ends of the motors. Each drive motors is electrically connected to the robot controller by a 10 conductor ribbon cable which supplies power to the motor and carries position data from the encoder. Each axis name and servo motors are shown in Fig. 2. A sixth servo drive motor and associated linkage to operate the gripper are mounted in the wrist assembly. The gripper servo drive does not cause motion in any arm axes.

1.2 MARK III Controller

The MARK III system in Fig. 3 is a complete manufacturing work cell controller with an internal microprocessor, operating system software, and interface electronics. The electronics provides interface for 8 dc servo motors, a 32 key hand-held teach pendant, and a standard RS-232C serial port for an external computer. The controller is shown in Fig. 3. The controller can communicate with either an external computer or the Rhino teach pendant for the arm control information. For example, the computer can send one of

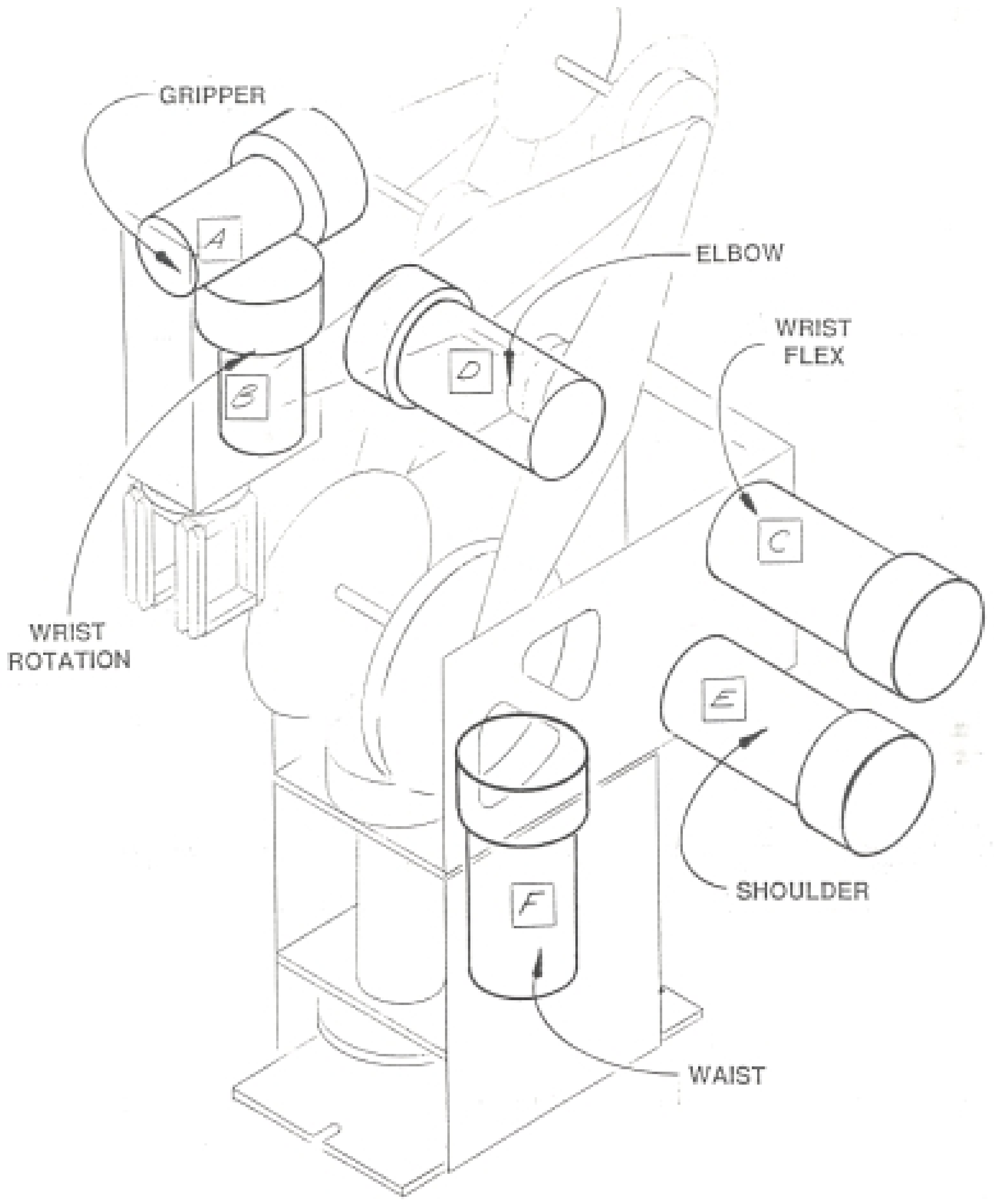
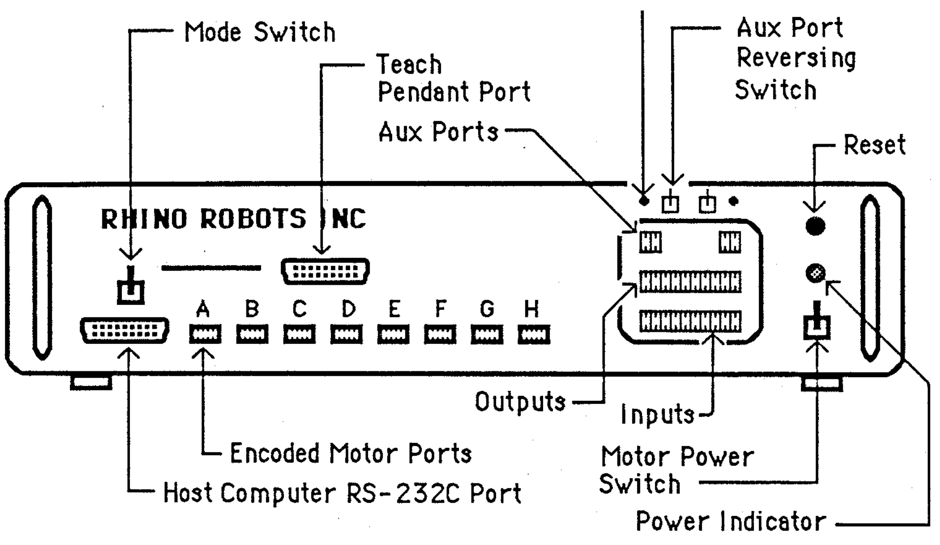
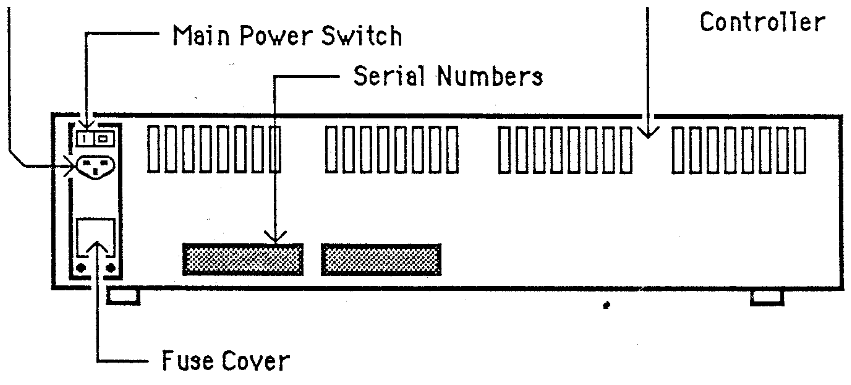


Figure 2: Joint and Servo motor name



(a) The front of the Rhino controller



(b) Back of the controller

Figure 3: The controller of Rhino XR system

the 13 work cell and robot commands to the controller over the RS-232C serial port. When the controller receives the command from the computer, the microprocessor inside the controller executes the command. The serial port and the commands included in the operating system software in the controller permit robot and work cell control from a remote computer. Using the same technique, the Rhino teach pendant can be used to control robot motion and develop work cell programs.

The controller has a *MODE SELECT* switch to put controller in either the teach pendant or remote computer mode. Two power switch allow separate control of the main power and servo motor power. A push button, labeled RESET, reestablishes the initial values in the controller memory and operating system. **Also, the reset button on both the controller and the teach pendant are your EMERGENCY stop buttons.** A pilot map on the front panel indicates when the main power switch is in the "on" position.

The pendant is a 32 key microprocessor controlled programming unit with a 7 character digital display and a reset button to restart the controller and act as an emergency stop.

If you do experiment with Rhino robot system. Please do first below in order.

- Check Motor Power switch is off.
- Turn on main controller power - Main power switch is at rear panel of the controller. (Fig. 3(b))
- Check the Motor Switch.
- Turn on the motor power switch.
- Push reset switch.
- Check 'init' is displayed in Teach pendant 7-segment LED panel.

And the power down sequence is the reverse of the power-up order. Use the following sequence to take the Rhino work cell in the teach pendant programming mode out of service.

1. Turn "off" arm motor power.
2. Turn controller power "off".
3. Turn the computer "off".

1.3 Teach pendant

The Rhino robot system has both a teach terminal and a teach pendant to program the robot arm and work cell hardware. The Rhino XR-3 teach pendant, illustrated in Fig. 7, is connected to the MARK III by a ribbon cable and connector on the front panel of the controller. The pendant is a 32 key microprocessor controlled programming unit with a 7 character digital display and a reset button to restart the controller and act as an emergency stop. Fifteen of the keys are used for program development and 16 provide motion control for the 5 arm axis servos, gripper servo, and 2 auxiliary servos. The teach pendant supports work cell program development in two ways: 1. complete work cell programs can be generated using only the teach pendant keys, 2. arm positions for programs written on the teach pendant terminal are taught using the teach pendant.

The teach pendant, the microcomputer in a Rhino robot system, is connected to the MARK III controller through the RS-232C serial port.

2 Specifications of Rhino XR System

In this section, specification of the Rhino XR system is presented. And, some components of the Rhino robot system are also explained.

2.1 Basic dimensions

This is the Rhino system in a nutshell. The following tables give the specifications for the Rhino system.

- Basic specifications for the Rhino robot arm
  - Vertical Reach : 68cm
  - Radical Reach : 60cm
  - Lifting Capability (arm extended) : 0.45Kg
  - Weight of Rhino : 7.7Kg
  - Weight of Controller : 12.3Kg
- Resolution at each axis

Axis	Motor	Resolution
Fingers	A	not applicable
Wrist Rotation	B	0.18 degrees theoretical
Wrist Flex	C	0.12 degrees theoretical
Forearm	D	0.12 degrees theoretical
Shoulder	E	0.12 degrees theoretical
Waist	F	0.23 degrees theoretical

- Motor gear ratios and final reductions

Axis	Motor Gear Ratio	Encoder steps
Fingers	96/1	not applicable
Wrist Rotation	165.4/1	5.51
Wrist Flex	66.1/1	8.8
Elbow	66.1/1	8.8
Shoulder	66.1/1	8.8
Waist	66.1/1	4.4

- Speed at each axis

Axis	Speed(degrees/second)
Fingers	1 sec. to open and close fully
Wrist Rotation	32
Wrist Flex	45
Elbow	30
Shoulder	20
Waist	60

- Link length for finding D-H parameters is shown in Fig. 4.
  The dimension is expressed in terms of inches.

2.2 Servo

The Rhino uses a closed loop encoder system that works well when adjusted correctly. First is the encoder wheel. As the wheel turns it feeds data to the controller. From this data the controller determines if the

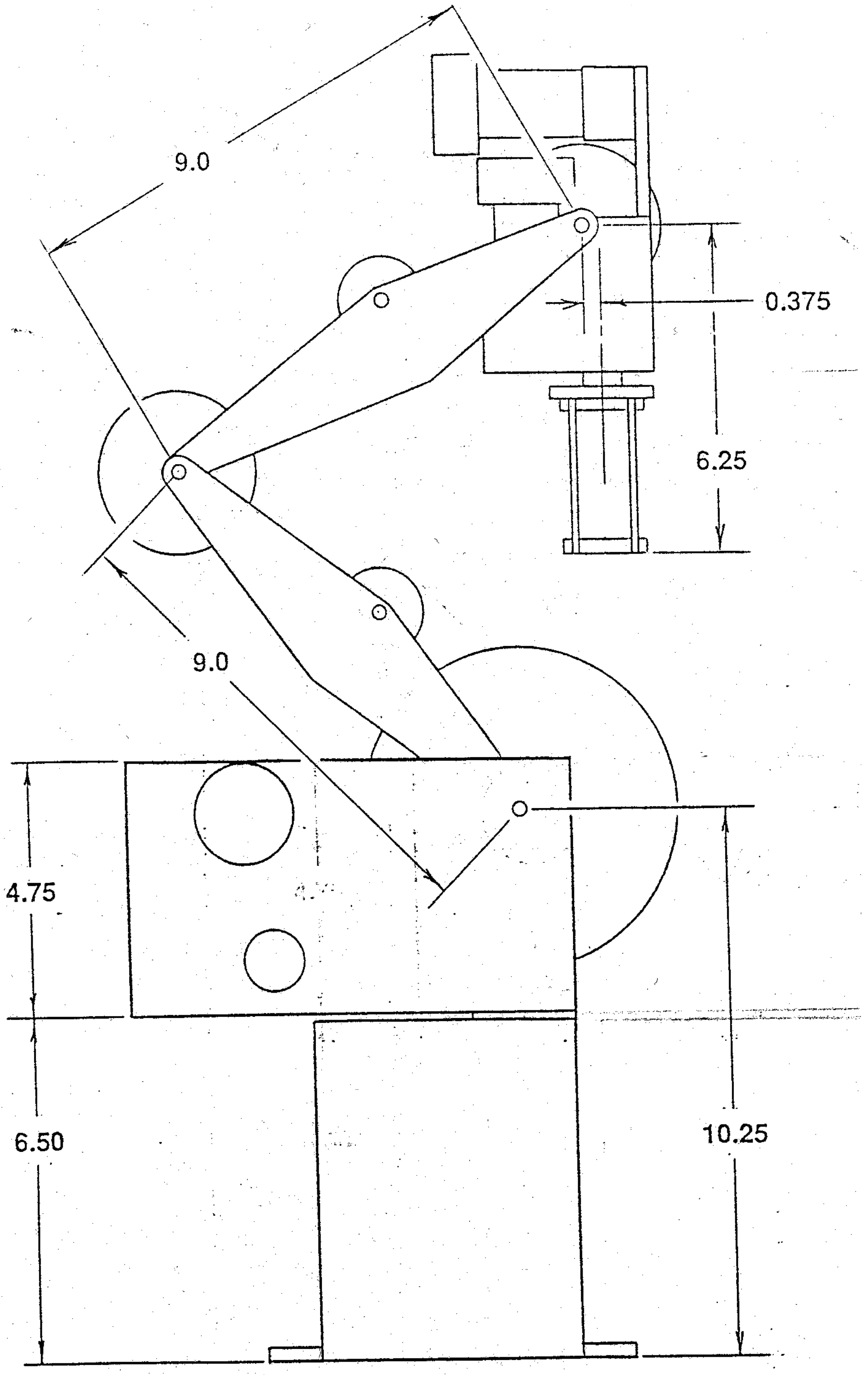


Figure 4: Rhino XR system : specification

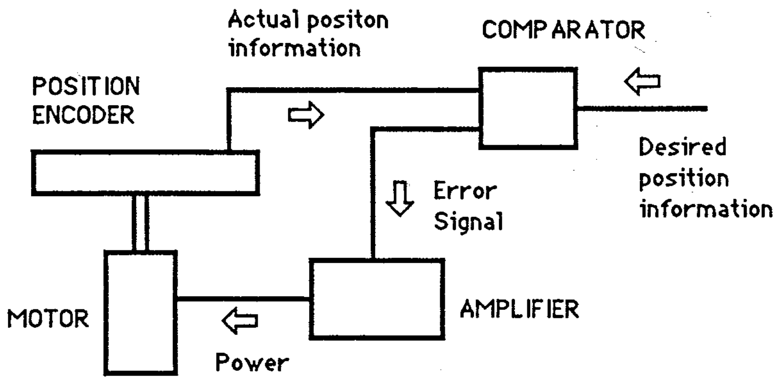


Figure 5: The servo control loop

correct position. Lets say the motor has turned when it was not supposed to. The controller would turn the motor on in the opposite direction to return it to the desired position This is fine expect the encoder now has gone to far in the other direction.

Fig. 5 shows basic control loop of servos. A feedback system that consists of a sensing element, an amplifier and a servomotor used in the automatic control of a mechanical device.

2.3 Encoder

Encoder is used to find the position of the servo motor. The encoders used on the Rhino system are incremental encoders. Each consists of an aluminum disk with light and dark bands placed radially on one side. Two reflective optical sensors are located to detect the different bands and placed so as to produce two signals (A and B) which are 90 degrees out of phase. The large motors (C-F) have six dark and six light bands per revolution; the remaining motors have 3 sets of bands.

When the A signal leads the B signal, the motor is moving in one direction and when the A signal trails the B signal, the motor is moving in the other direction. The logic in the controller is arranged to be able to make the necessary discrimination.

Given that there are two signals from the encoder, there are four states that the incremental encoder can provide. They are 00, 01, 11 and 10. Note that if these were interpreted as decimal values, the flow is from 0 to 1 to 3 to 2 and back to zero, not 1, 2, 3 and 4. Also note that it is not possible to go from 00 to 11 without going through one of the intermediate states and that it is not possible to go from 10 to 01 without going through an intermediate step. These are called forbidden states and if these changes occur, they indicate an error in operation. Fig. 6 show the transition of the encoder state.

A counter can be set up using either 1,2 or 4 positions of the encoder. If we pick one position, we would increment or decrement the counters only when the states went all the way around the above diagram. This method is used in the Mark III controller. If we picked two positions, we would increment or decrement the counters whenever the state changed from one side of the diagram to the other side on the diagram. If we picked four positions, we would change the encoder count every time the state changed.

The controller uses eight registers in its memory as error registers for the motors. As long as a register is zero, the motor has no power applied to it. If an encoder turns, the register is added to or subtracted from as needed. As soon as the controller sees a number in the error register, it applies power to the motor in a way that will turn it in the direction that will decrement the register as the motor turns. This is the holding algorithm that maintains motor position at all times.

When the controller receives a **START** command from the host computer, it adds the given number to the motor error register. This has the effect of making the controller start the motor in a direction that will

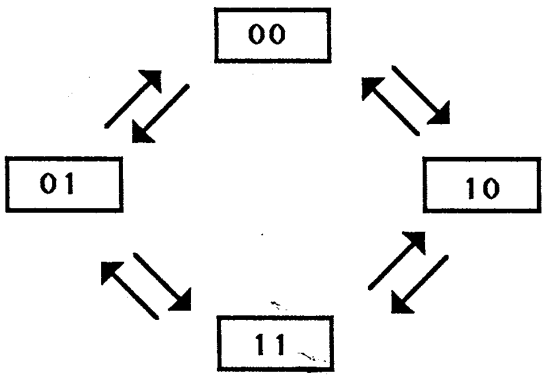


Figure 6: Encoder states

make that error register go to zero. Each cycle of the encoder changes the error register by one. When the error hoes to zero, the motor is turned off. If the motor overshoots, the power to the motor will be reversed automatically to bring the motor back to the zero position.

With the Rhino XR-3 robot, the large motors have 6 detectable state per motor revolution and the small motors have 3 states per revolution.

2.4 Optics

The optics system is the electronic circuitry that feeds;information back to me controller with regard to me position of the motor. The information needed by the controller is the position of the motor and the status of the microswitch. There are a few key assemblies used in the optics system to perform these functions. They follow, and each is very necessary to have the robot function correctly.

1. Optics board assembly
2. Encoder wheel assembly
3. Ribbon cable assembly

Optic Board Assembly

The Optic printed circuit board includes 2 infra-red LEDs, 2 infra-red Phototransistors,biasing resistors, a cable header, and an mounting bracket. There are two types of assemblies used with the XR-3 robot. They are exactly the same except they use different mounting1 brackets. The three hole bracket is for the small motors in the hand. The body brackets seem to have 2, 4, or 6 hole types. The cable header is a ten pin male header used to connect the ribbon cable into the optic board. The biasing resistors are used to determine the current that is allowed to go through the LEDs and the phototransistors. The optics printed circuit board is used to mount all of the components and provide electrical connections needed.

The LEDs are infra red so you cannot see the light. However the light is quite bright and is on any time 5 Volts is applied to the circuit board. The light is directed out of the LEDs towards the phototransistors. This will turn the phototransistor on. The phototransistors are turned off when they cannot see the infra red light. This condition occurs when the encoder wheel is between the LED and the phototransistor and they do not line up with one of the slots in the wheel. This prevents the infra red light from getting through.

As soon as the phototransistor sees infra red light,the base region of the phototransistor saturates, allowing current to flow thru the device. The collector drops from 4.80 volts to .70 volts. This change of state is noticed by the controller, which is constantly monitoring the optics. In order for the controller to monitor the optic, the signal must go thru the circuit board,thru me header, into the ribbon cable, and finally out the ribbon cable into the controller. If its path is broken in any of these places, a problem will occur. Also, the controller must read both of the phototransistors for the system to function properly.

There are three other electrical connections needed from each optic board. The first is the limit switch line which is connected from pin 6. This line is normally at a 5 volt level. When the limit switch is closed,the switch grounds pin 6 and goes to a buffer on the controller board.

Another function of the optic board is to feed motor power to the motors. This is done by pins 9 and 10 for positive and negative motor power and pin 7 and 8 for the motor ground. Across these two lines is a capacitor to ground spikes. The motor power lines are positioned away from the logic lines to prevent cross talk in the cable and also to prevent any noise from the motors being coupled into the logic.

2.5 Interface

Configure your computer's RS-232C port for the following:

9660 Baud
7 Data bits
even parity
2 stop bits

The Rhino controller does not use a handshaking protocol. Therefore, the DB 25 connectors must be modified.

Electric Connections : You must have an RS-232C serial interface, capable of both sending and receiving data. The Mark III controller uses only 3 of 25 communication lines on teh DB25 connector.

- Line 2 : carries data transmitted by controller, received by host computer.
- Line 3 : carries data received by the controller, sent by host computer.
- Line 7 : the common data ground line.

3 Teach pendant operation

The teach pendant on the Rhino system includes a separate microprocessor, system memory, and software. The version of software present in the teach pendant is displayed when the following command sequence is entered using teach pendant keys.

The Rhino teach pendant in Fig. 7 has 32 keys, a power "ON" indicator, a seven digit display, and a reset button. The pendant includes a "SHIFT" key the function or command on the upper half of the key is entered. The reset button serves as an **EMERGENCY** stop during program execution.

3.1 Home position

All industrial robots have a **HOME** position for the arm. Home is a defined or known position in the work envelope from which all new programs and stored programs start. A home position is necessary for the repeatability of the points taught in the program For examples, a robot loading a machine with parts from a parts feeder moves from the start position to the point in the work envelope where the tooling picks up a part. If the start position changes so will the position of the fooling as it tries to acquire a part. So unless the robot tooling starts from a known position every time, the programmed points will not be repeatable.

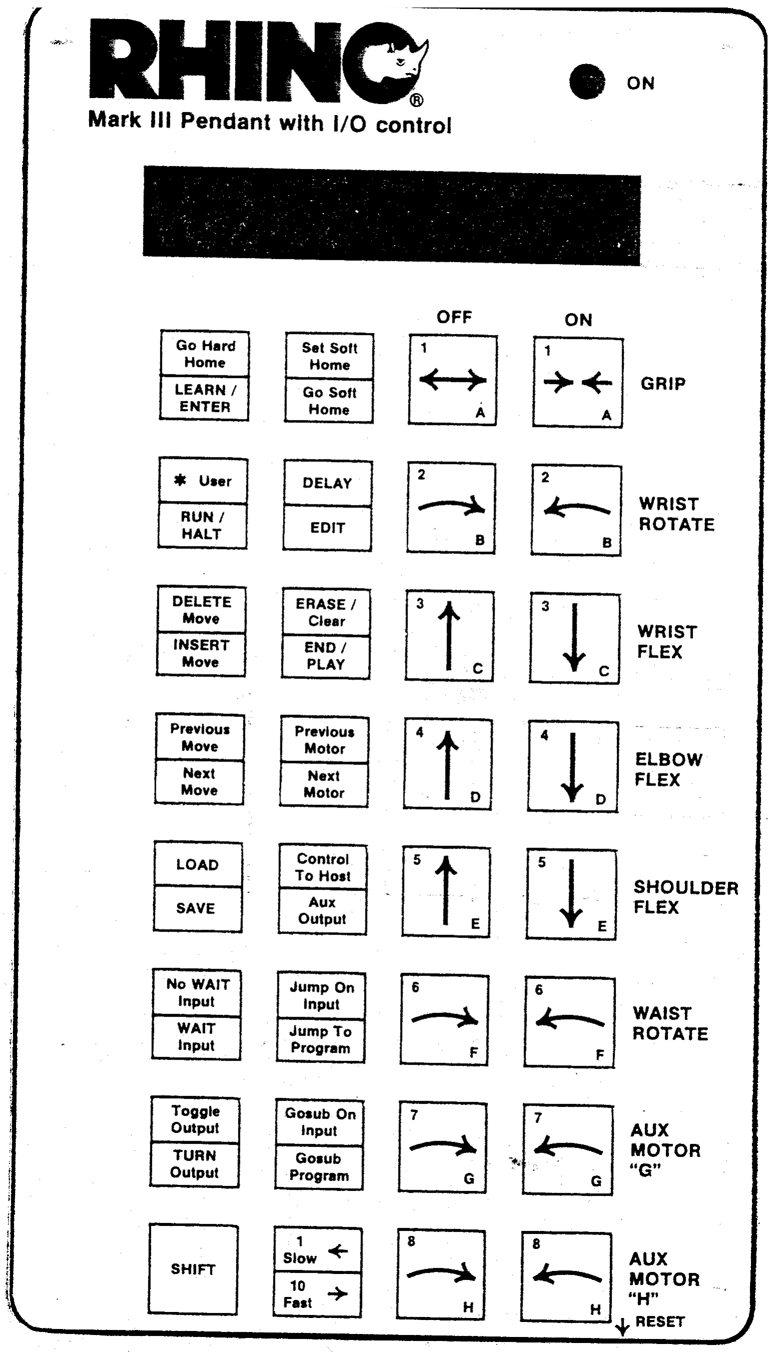


Figure 7: Rhino XR system teach pendant

The Cincinnati Milacron family of robots has a single home position which must be taught during the power-up sequence using the command “Home, Set”. The Rhino system has two types of HOME positions. The HARD HOME is a mechanical home position which is determined by limit switches located on the waist, shoulder, elbow, wrist flex, and wrist rotate axes. The SOFT HOME is a software home and can be set at any position in the work envelope. Hard home must be performed before a new move sequence is taught and before a stored sequence is loaded and executed. The soft home position is a point in the work envelope selected by the programmer because it is an efficient starting point for the desired move sequence. The hard home command command is that holding the SHIFT down while the function key is pressed causes the function described in the upper block of the key to be initiated. The function in the bottom block is activated without the SHIFT key. At this case you may see ”PGoHArd” in LCD panel.

3.2 Motion keys

The position axes are waist, shoulder, and elbow, while the orientation axes include the two wrist motions flex and rotation. In addition, the gripper open and close control is part of the motion key group on the Rhino pendant.

SYSTEM RESPONSE : When any of the axis motion keys, A through H, are pressed the axis motor moves 10 increments on the optical encoder which is about 1.2 degrees for the position axes. Each axis has a key for either the positive or negative direction. The teach pendant display indicates the current axis move with the following readout ”Pb- 276”. The first digit will be “P” to indicate the PLAY mode, the second lowercase character ’b’ indicates Motor label and third ‘-’ means direction of motor rotation, and last number means position.

If a motor port is open (motor disconnected) then the teach pendant displays “P off” when the motor key is pressed.

If a motor hits an object and cannot complete the move entered on the teach pendant then the teach pendant displays “PStALL” which indicates stalled motor. The motor on the stalled axis then reverses direction to move away from the stall point. The gripper “open” (axis heads pointing out) and “close” (arrow heads pointing in) keys cause the Rhino electric servo gripper to simulate the opening and closing of a pneumatic gripper.

4 Control and Programming

The command set of the Rhino robot controller system has been designed to allow the complete control of the Rhino controller. Through the use of only 14 basic commands, the user can control the position of all eight motors, read any of the 16 input bits, set any of the 8 output bits and control the AUX ports. All of Rhino Robot’s software uses there kernel commands to create the higher level languages, such as RoboTalk.

The Rhino controller has th following commands:

- <return> : Carriage return (initiate a move)
- ? : Return distance remaining
- A-H : Set motor movement value
- I : Inquiry command (read limit switched C-H)
- J : Inquiry command (read limit switched A-B and inputs)
- K : Inquiry command (read inputs)

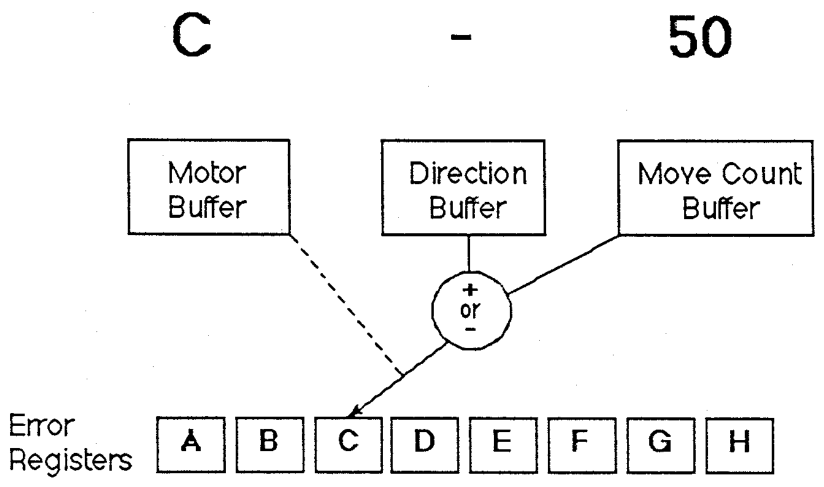


Figure 8: Processing of a motor move command

- L : Turn Aux #1 port ON
- M : Turn Aux #1 port OFF
- N : Turn Aux #2 port ON
- O : Turn Aux #2 port OFF
- P : Set output bits high
- Q : Controller reset
- R : Set output bits low
- X : Stop motor command

The Rhino Robot controller accepts commands as ASCll characters; each character is acted upon immediately upon receipt. Unlike most computer peripheral equipment, the controller does not wait for the receipt of a carriage return to signify a command completion; in fact the carriage return is considered a command itself.

When the motor move command letters A-H are received, the motor specifier is stored in the motor buffer over writing its previous contents. The receipt of a motor specifier also sets the direction buffer to the plus direction and clears (sets to zero) the move count buffer. Any sign character (+or-) is stored in the direction buffer over writing its previous contents. When a number digit is received, the contents of the move count buffer is multiplied by ten and the new digit is added to me product. In this way the move count buffer correctly accumulates a multi-digit number.

When a carriage return character is received, the controller adds or subtracts the amount in the move count buffer to the value in the error register pointed to by the motor buffer. If the direction buffer has been set to a plus the amount is added; if the direction buffer has been set to a minus the amount is subtracted.

The following illustration shows the relationships of the input buffers and the motor error registers. In this example, a C command was received followed by a -50. When a carriage return is received, the c motor error register will be decremented by 50.

The receipt of a carriage return can have no effect on the controller if the move count buffer is zero, as it is after a motor specifier(A-H) has been received. You can take advantage of this fact if you want to terminate all commands sent to the Rhino robot controller with a carriage return which would be the case if you were using standard PRINT statements in BASIC to control the robot. Preceding all commands with a motor specifer allows you m use the carriage return.

In the following command descriptions, the format and the examples of the commands will illustrate the use of the leading motor specifier. This will lead to a more intuitive understanding of the command set.

4.1 <return> Initiate a motor move

Whenever a carriage return is received from the host computer, the controller takes the move count in the motor move count buffer and adds it to (or subtracts it from depending on the sign of the direction buffer) the value in the error register of the motor that is addressed by the motor buffer.

Thus,if the command sequence **C-40** <**return**>is sent to me controller, a **C** will be stored in the motor buffer, a minus will be stored in the direction buffer and 40 will be stored in the move count buffer. Upon receipt of the <**carriage return**>, this data is transferred to the **C** error register and the controller will start the **C** motor in the negative direction with the intention of moving it 40 additional encoder steps in that direction. If the controller now receives a carriage return only, it will add another -40 to the error register for the **C** motor. With the above sequence of commands, the motor will eventually make a move of -80 total encoder counts.

4.2 ? Question command

Requests steps remaining to move on motors A thru H.

When making long moves it is necessary to determine how far a motor has to move before more move information can be sent to me controller. The command used to determine how far a motor has yet to move is the Question command.

The format of the command is

$$[< motorID >] <? >< return >$$

Examples of the command as issued from a BASIC program:

PRINT “D?” < return >

PRINT “A?” < return >

PRINT “C?” < return >

where the first letter identifies the motor error register to be interrogated and the question mark indicates that the remaining error count for that motor is to returned to the host computer. As always, the controller adds 32 to the error value before returning it to the host computer. Adding 32 to the count prevents the controller from sending ASCII command codes to the host computer. The controller always sends the absolute value of the error signal; it does not send the direction of the error signal. This means that you can determine how far a motor still has to move but cannot determine whether the move is to be in the positive or the negative direction.

NOTE : using the “?” command does not re-issue the move in the move buffer because the motor ID letter preceding the “?” always clears the move buffer.

4.3 A to H start motor commands

starts motors A to H and moves them a number of encoder steps

The start command is used m instruct the controller to start a given motor and to move it in a given direction by a given number of encoder steps. The value is added to the move in progress.

The format of the command is

$$< motorlD > [< sign >] < encodercounts >< return >$$

where < motor ID > is an uppercase letter A-H, <sign> is an optional + or - character (if no character is present a + is assumed) and <encoder counts> is a number from 0 to 127.

Example : for programming in BASIC: **PRINT "C-93"**

The above command will move the "C" motor in the negative direction by an additional 93 encoder positions. The positive sign is not needed for moves in the positive direction and may be omitted. The carriage return is used to execute the command. Only one motor can be addressed at one time. Other samples of the command are:

```
B-6 <return>
A+21 <return>
C33 <return>
D-125 <return>
H <return> (clears the move count buffer, therefore no move is made)
<return> (repeats the previous move)
```

Sending only a carriage return without a motor move, after a motor move command, repeats the last motor move command. Even carriage returns that are part of another command (discussed later) will re-issue the move command. In order to cancel a move command that would be carried out by the receipt of a carriage return, all commands should be preceded by a motor ID character (A thru H). Any motor ID letter sets the move buffer to zero if there are no numbers attached to it. Once the move buffer is cleared other commands with carriage returns will not activate the move instruction. For example the commands Similar to me following will clear the move buffer.

```
A <return>
CI <return>
EJ <return>
BL <return>
```

Once the move buffer is cleared, other commands may be issued without the motor ID characters. The move buffer is active after each nor-zero motor move command. It should be cleared before using commands when necessary.

4.4 I I-Inquiry command

Returns status of microswitches on motor ports C,D,E,F,G,and H

The INQUIRY command allows the user to interrogate the status of the 6 microswitches on the C,D,E,F,G and H motors. The format of the command is as follows:

$$[< motorID >] < I >< return >$$

Where the motor ID is included if the move buffer is to be cleared. Sample uses of the command as issued in a BASE program:

PRINT “r” <return>
PRINT “AL”<return> (Clears the move buffer first)

The command returns the status of the 6 microswitches in one byte. The returned byte is interpreted as follows **after subtracting 32:**

Bit	Motor
0 LSB	C
1	D
2	E
3	F
4	G
5	H
6	–
7 MSB	–

The controller adds decimal 32 to me byte before transmitting it to me host computer so that no control codes will be transmitted to me host computer. Upon receipt of the returned byte, it is the users responsibility to subtract 32 from the byte before using it. A closed microswitch is seen as a 1(one). An open microswitch is seen as a O(zero).

Bits 6 and 7, the most significant bits, are not used.

The host computer must be ready to receive the inquiry byte before sending the control another command. See detailed examples under the sections on running the robot with the IBM-PC.

4.5 J J-Inquiry command

Returns status of microswitches on motors A and B and input lines 1,2,3 and 4

The J-INQUIRY command tells the controller to send back the status of the microswitches on motors A and B and the status of input lines 1,2,3 and 4 of the 8 line input port. The returned data byte is of the form 00BA4321+32, where A and B are the levels of the A and B motor limit switches and 4,3,2, and 1 are the levels of the input lines 4 through 1. As with all other information returned to me host computer, 32 is added to the returned value measure that no ASCII control character is returned to the cmnputer. You use the J-INQUIRY command just like the J-INQUIRY command.

The format of the command is

$$[< motorlD >] < J >< return >$$

where, the <motor ID> has to be included if you want to clear the move buffer.

Examples of the command as issued form a BASE program:

```
PRINT"J" <return>
PRINT"CJ" <return>    (Clears the move buffer first)
```

When interpreting the bits returned by the motor controller, a value of 0 means that the input is low (or that a microswitch is closed). A value of 1 means that the input is high (or that a microswitch is open).

Bit	Meaning
0 LSB	Input 1
1	Input 2
2	Input 3
3	Input 4
4	Motor "A" Limit Switch
5	Motor "B" Limit Switch
6	–
7 MSB	–

Bits 6 and 7 are not used.

The controller adds decimal 32 to me byte before transmitting it to me host computer so that no ASCII control codes will be transmitted to me host computer.

The host computer must be ready to receive the inquiry byte before sending the controller another command. See detailed examples under the sections on running the robot with the IBM-PC.

4.6 K K-INQUIRY command

Returns the status of input lines 5,6,7 and 8.

The K-INQUIRY command tells the controller to send back the status of inputs 5 through 8 of the 8 line input port. The returned data is of the form 00008765+32, where 8,7,6 and 5 are the levels of input lines 8 through 5. The format and usage of the K command is similar to me I and J commands.

The format of the command is

$$[< motorlD >] < K >< return > .$$

where, the ¡motor ID¿ has to be included if you want to clear the move buffer. Examples of command as issued form a BASIC program:

```
PRINT "K" <return>
PRINT "EK" <return>    (Clears the move buffer first)
```

The command returns values that may be interpreted as follows after subtracting 32 from the byte received.

Bit	Meaning
0 LSB	Input 5
1	Input 6
2	Input 7
3	Input 8
4	–
5	–
6	–
7 MSB	–

Bits 4,5,6 and 7 are not used

The controller adds decimal 32 to me byte before transmitting it to me host computer so that no ASCII control codes will be transmitted to me host computer.

The host computer must be ready to receive the inquiry byte before sending the controller another command. See detailed examples under the sections on running the robot with the IBM-PC.

4.7 L Turn Aux. Port#1 ON

Turns Aux port#1 ON

The L command turns auxiliary port 1 ON. Auxiliary port 1 provides 1 amp at-20 volts DC. The forward/reverse switch above the aux. connector determines the polarity of the pins and can be used to reverse a PM DC motor connected to the port.

The format of the command is

$$[< motorlD >] < L >< return >$$

where, the <motor ID> has to be included if you want to clear the move buffer.

Examples of command use as issued in a BASIC program:

```
PRINT "L" <return>
PRINT "CL" <return>    (Clears the move buffer first)
```

4.8 M Turns Aux. Port#1 OFF

Turns Aux port#1 OFF

The M command turns auxiliary port 1 OFF. You use it just as you would the L command. See description of L command.

4.9 N Turns AuX. port#2 ON

Turns Aux. port#2 ON

The N command turns auxiliary port#2 ON. Auxiliary port#2 provides 1 amp at +20 volts DC. The forward reverse switch above the aux. connector determines the polarity of the pins and can be used to reverse a PM Dc motor connected to the port.

The format of the command is

$$[< motorID >] < N >< return >$$

where, the <motor ID> has to be included if you want to clear the move buffer. Examples of command use as issued by a BASIC Program:

```
PRINT "N" <return>
PRINT "CN" <return>    (Clears the move buffer first)
```

4.10 O — Turns Aux. Port#2 OFF

Turns Aux Port#2 OFF

The command turns auxiliary port 2 OFF. You use it just as you would the N command see description of N command.

4.11 P — set output Line High

Sets an output line HIGH

The P command tells the controller that the next digit it receives identifies the output line to be set high. The 8 output lines of the output port are numbered from 1 to 8. The output lines are set high during startup and after a reset. The Rhino controller output lines provide TTL level signals.

The format of the command is

$$[< motorID >] < P >< outputlinenumber >< return >$$

where the <motor ID> has to be included if you want to clear the move buffer. Examples of command use as issued by a BASIC program:

```
PRINT “P3” <return>
PRINT “AP6” <return>    (Clears the move buffer first)
```

4.12 Q — Reset

Resets the entire Controller

The Q command tells the controller to reset itself. The controller will clear all of its internal registers, turn off all motors, turn off all the auxiliary ports, set all output lines high and reset its communication port according to the BAUD switch in the controller.

The Q command is a convenient way of resetting the controller (in software) without having to press the reset button.

The format of the command is

$$< Q >< return >$$

Examples of command use as issued in a BASIC program:

```
PRINT “Q” <return>
```

complicated software programs often start with the “Q” command ensure that the controller is at a known (reset) state when the program starts.

4.13 R — set output Line Low

sets an output line LOW

The R command is similar to me P command but the next digit received after the R identifies the output line to be set low by the controller. The 8 output lines are numbered from 1 to 8. The output lines are set high during startup and after a reset. The Rhino controller output lines provide TTL level signals.

The format of the command is

$$[< motorID >] < R >< outputlinenumber >< return >$$

Examples of command use as issued by a BASIC program:

```
PRINT "R5" <return>
PRINT "CR2" <return>    (Clears the move buffer first)
```

4.14 X Stop motor command

Stops motors A thru H

It is often necessary to turn off a motor that is stalled one way to do this is to determine how far the motor is from completing it's move and then sending a move command that will reverse the motor far enough to cancel the remaining portion of the move. A faster way is to send the stop command.

The format of the stop command is

$$< motorID >< X >< return >$$

Examples of the command as issued from a BASIC program:

```
PRINT "BX" <return>
PRINT "DX" <return>
PRINT "HX" <return>
PRINT "AX" <return>
```

where the first character identifies the motor to be stopped and the "X" is the stop Command. The "X" command is followed by a carriage return and does not re-issue the preceding move command because the motor letter clears the buffer. When the "X" command is received, the remaining portion of the motor move, (the portion that was still to be moved,) is lost and cannot be recovered. If the information is important, the user should first determine how far the motor still has to go with the "?" command, store the information in the host computer and then send the "X" command to stop the motor.

5 Simulator : SIMULATR

In this section, we introduce simulator of Rhino XR system from [1]. It's name is SIMULATR. The simulator helps you to test your own program before you run with real Rhino robot. The simulator remain in ram if once the simulator is executed.

Basically, the simulator is hocking the signal of serial port, which is generated from your own control program. Therefore, you just execute your own program and check whether your code is correct or not.

We recommend to run simulator before experiment with Rhino robot. (If you do not run, your incorrect program may make the Rhino robot broken.)

You can get the simulator from Homepage of this class or TA (You can email to me).

We attach a few pages of usage of the simulator. From these, you can get all information about the simulator, SIMULATR.

Note. 1 Owner's manual, Service manual and Student's manual of Rhino XR system are available. If you want those, you can copy it. Come to Control Lab. at laboratory.
Note. 2 Please let me know errors in this material for other people.

References

[1] Rhino Robots, Inc., *Owner's Manual*.

[2] Rhino Robots, Inc., *Rhino XR-3 Robot Instructor's Manual*.

[3] Rhino Robots, Inc., *Rhino XR-3 Robot Service Manual*.

[4] Rhino Robots, Inc., *Rhino XR-3 Robot Software Manual*.

Chapter 2

A Graphics Robot Simulator

Robot control programs are often developed off-line using a simulated robot and work-cell. This way the actual robot does not have to be taken out of productive service during program development, it is only needed for the final debugging and testing of the software. In this chapter we introduce a three-dimensional color graphics robot simulator program called SIMULATR that simulates a robot and its environment (White et al., 1989). The objective of SIMULATR is to allow users to develop and test robot control programs off-line without a physical robot present using the computer programming language of their choice. SIMULATR is a terminate and stay resident (TSR) program that runs in the background on IBM PC/XT/AT computers and true compatibles. It intercepts commands sent to the robot controller through the COM1 device and responds to the commands as if it were the robot controller. SIMULATR supports a variety of robotic arms including the five-axis Rhino XR-3 educational robot. A copy of SIMULATR and supporting software is supplied on the distribution disk.

2.1 Installation

SIMULATR runs under the MS-DOS operating system, Version 2.18 through 3.30. The hardware requirements for SIMULATR include an IBM PC/XT/AT computer, or true compatible, with 512K of memory, a COM1 serial port, and a graphics adaptor card (CGA, EGA, Hercules). SIMULATR occupies approximately 75K of memory and therefore *can* be run on computers with less than 512K total memory. However at least 512K is recommended in order to provide adequate space for user programs and program development tools such as text editors, compilers and interpreters. The list of currently supported graphics boards can be found in the file README.DOC on the distribution disk.

SIMULATR has been designed to be applicable to several different types of robots and workcells. This is achieved through the use of data files which configure the robot simulator at installation time. The easiest way to install SIMULATR, using default values for the robot and workcell data files, is to issue the following batch file command in response to the operating system prompt:

This will install the Rhino XR-3 robot in its default workcell. During installation, two temporary data files are created. Consequently, the disk which contains SIMULATR must *not* be write-protected, and it must contain adequate free space (about 10K) for these files. A more general way to install SIMULATR is to use the INSTALL command with data file arguments as follows:

INSTALL <robot.dat> [workcell.dat]

Here <robot.dat> is a data file which specifies the robot to be simulated, while [workcell.dat] is an optional data file which specifies the workcell or environment within which that robot is to operate. Here upper case letters are literals that should be typed exactly as they appear, angle brackets, $< \cdots >$, denote an argument that *must* be supplied by the user, and square brackets, $[\cdots]$, denote an *optional* user-supplied argument.

2.1.1 Robot data file: RHINO.DAT

The first parameter in the INSTALL command is the data file that specifies the robot to be simulated. To simulate the Rhino XR-3 educational robot, the user specifies the file named RHINO.DAT. If a file extension is not supplied, the default extension (.DAT) is assumed. Additional robots such as the Adept One robot (ADEPT.DAT) and the Intelledex 660 robot (INTEL.DAT) can also be simulated. A list of the currently supported robots can be found in the disk file README.DOC. Although different robots can be simulated, they all use the same generic controller, a controller that is upward compatible with the Rhino XR Series controller (Hendrickson and Sandhu, 1986).

Each robot data file contains information which describes the kinematics and physical appearance of the manipulator. For each joint of the robot, the joint angle θ, joint distance d, link length a, and link twist angle α are specified. Next, the joint precision, the limits of travel for each joint, and the default joint speed are specified. This is followed by information about the location of the joint limit switch, the joint direction, and the joint label used for the movement commands. A description of the type of tool mounted at the end of the arm is also included. Finally, a physical description of the shape of each link of the arm is specified as a series of vectors which describes how to draw the link in a local coordinate frame. A color index is included for each link so that adjacent links can be distinguished from one another by using different colors or different shades of a color.

2.1.2 Workcell data file: RHINOCEL.DAT

The second parameter in the INSTALL command is an optional data file parameter that specifies the environment in which the robot is to operate. The default workcell data file supplied for the Rhino XR-3 robot is named RHINOCEL.DAT. Customized workcell data files can also be created off-line by the user with an interactive program called WORKCELL.EXE which prompts the user for a description of a work environment.

Each workcell data file contains information which specifies the size of the workspace, the nature of the objects which populate the workspace, and the sensors which can be used to locate and identify these objects. The first information specified in the workcell data file is the size of the workspace which is defined by three numbers (X, Y, Z) as follows:

$$W = \{(x, y, z) : 0 \leq x \leq X,\ -Y \leq y \leq Y,\ 0 \leq z \leq Z\} \tag{2.1}$$

The workspace W consists of the region in front of the robot $(0 \leq x \leq X)$, above it $(0 \leq z \leq Z)$, and to both sides of it $(-Y \leq y \leq Y)$. A narrow region behind the robot sufficient to show the back of the robot is also displayed. In order to make the picture of the robot as large as possible on the screen, the size of the workspace should be made as small as the reach of the robot permits.

Several workcell objects in the form of rectangular blocks can be placed in the workcell. The number of blocks, and their sizes, positions, orientations, and colors are specified in the workcell data file. These blocks can be manipulated by the simulated robot. Consequently tasks such as pick-and-place operations and stacking and unstacking of blocks can be performed. The blocks can also be sensed by a simulated overhead camera. The characteristics of the camera including its location, its field of view, and its resolution in pixels are specified in the workcell data file.

2.1.3 Program development restrictions

When SIMULATR is installed, neither the Turbo Pascal integrated environment (TP) nor the Turbo C integrated environment (TC) can be used due to hardware conflicts. Consequently, one must first remove SIMULATR using the following batch file in order to use an integrated program development environment.

```
REMOVE
```

Repeatedly removing SIMULATR for program development and reinstalling it for program testing is cumbersome at best. However, there is an alternative. The Turbo Pascal command-line compiler (TPC) and the Turbo C command-line compiler (TCC) can be used for program development *without* removing SIMULATR. This is the recommend procedure. The Microsoft BASIC interpreter (BASICA) can also be used when SIMULATR is installed.

During program testing, the user may have occasion to abort a program with the <Ctrl/C> key. If SIMULATR is in the graphics display mode when the user program is aborted, the screen will remain in the graphics mode. At this point the screen must be returned to the text mode to resume normal operation. This can be achieved by

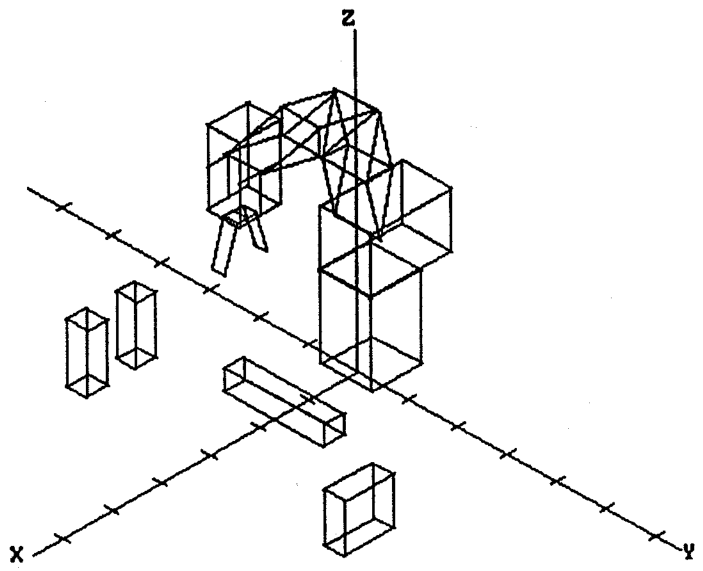


Figure 2.1: Rhino XR-3 in Home Position Using Perspective (1,1,1)

2. Change Perspective: <Alt/F2>

The change perspective option allows the user to alter the three-dimensional view of the simulated robot by entering a new perspective vector. The user is prompted for three integer coordinates, (x, y, z), which specify a point on the *line of sight* used to view the robot. For example, to view the robot from directly above, a perspective vector of (0,0,1) can be entered. This places the viewer's eye on the z axis looking down toward the origin. The coordinate values entered must be integers in the range $-9, \ldots, 9$, and they should be separated by spaces or commas. Pressing <Enter> after the third value completes the input. Only the relative values of the integers are important as the viewing distance, or scale, is adjusted automatically to display the entire workspace as specified in the file <workcell.dat>. The default perspective (1,1,1) represents an isometric projection. The robot can be viewed from the front, side, or top, respectively, by using the columns of the 3×3 identity matrix I for the perspective vector. For example, to view the Rhino XR-3 robot from its left side the perspective (0,1,0) can be used as shown in Figure 2.2.

3. Display status line: <Alt/F3>

The *status line* is a 1×80 text window which contains user-selectable robot status information. The <Alt/F3> key updates the status line and displays it at the top of the screen. It may be necessary to redisplay the status line if the user program has erased it by writing over it or by clearing the screen. This problem can be easily circumvented if the user program writes to a text window starting at line 2. A text window can be defined during initialization using, for example, the *Window* procedure in Turbo Pascal or the *window* function in Turbo C. This way the status line will be unaffected by the user program. The LOCATE command in BASIC also can be used to avoid writing in the status line area of the screen.

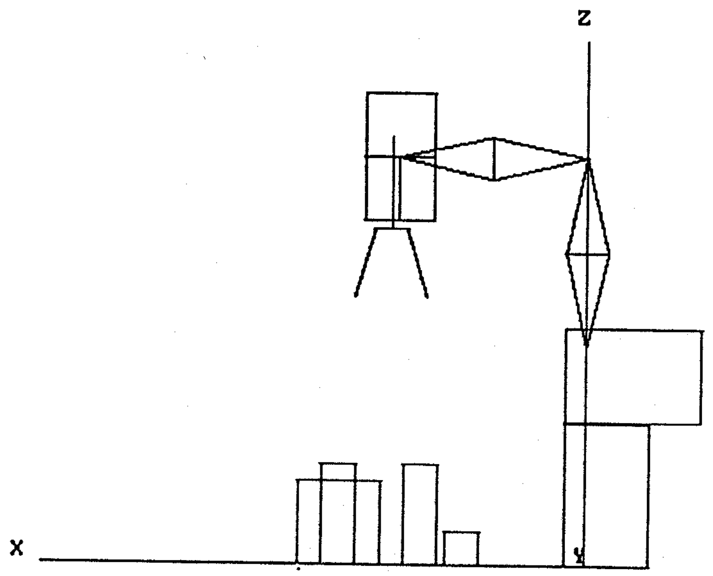


Figure 2.2: Rhino XR-3 in Home Position Using Perspective (0,1,0)

4. Select status Line: <Alt/F4>

When <Alt/F4> is repeatedly pressed, the contents of the status line *circulate* through the options listed in Table 2.2. The joint variable vector, q, specifies the joint angles of the revolute joints (degrees) and the joint distances of the prismatic joints (cm). The tool-configuration vector, w, specifies the position, p, and orientation, v, of the tool (Schilling 1990). The limit switch vector, s, specifies the state of each limit switch, open or closed. The error count vector, e, specifies the current values of the encoder error counter for each joint. Finally, the block configuration vectors consist of the (x, y, z) coordinates of the centroid of each block and the principal angle, β, of each block in the xy plane. All block configuration values are measured with respect to the robot base coordinate frame.

The status line at the top of the screen is periodically updated by SIMULATR as the robot moves. Consequently, if a user program writes to the line at the top of the screen, it will be overwritten by SIMULATR. However, if the blank line option is selected, SIMULATR will not overwrite any user data. In this way, the user program can have

Option	Display
1	Blank
2	Joint variable vector q
3	Tool-configuration vector w
4	Limit switch vector s
5	Error count vector e
6	Block 1 configuration vector
$\vdots$	$\vdots$
5+n	Block n configuration vector

Table 2.2: Status Line Options

access to the status line area of the screen although this is not recommended. Instead, it is recommended that the user program write to a text window below the status line.

The status line is a convenient tool for debugging and testing robot control programs because it requires little overhead and it is updated automatically as the robot moves. Once the robot has come to a halt, the graphics display option <Alt/F1> might then be used to examine the new robot status visually.

5. **Enable/disable path display: <Alt/F5>**

The enable/disable path display option is used in conjunction with the display robot command (#) discussed in Section 2.4. When the path display is enabled, the current picture of the simulated robot is not erased before an updated picture is drawn. In this way a multiple-exposure sequence of robot positions can be obtained to show the path taken. When SIMULATR is installed, it starts out with the path display option disabled.

6. **Enable/disable SIMULATR: <Alt/F6>**

The enable/disable SIMULATR option toggles the simulator between active and inactive states. When SIMULATR is enabled or active, all commands sent to the robot controller through the COM1 device are intercepted and processed by the simulator which resides in the background and responds as if it were the robot controller. When SIMULATR is disabled, the commands are passed directly through to the robot or whatever hardware is connected to the COM1 port. The default condition is enabled. Consequently, if SIMULATR is installed and a physical robot is to be controlled by a user program, then SIMULATR must *first* be disabled. SIMULATR can also be removed entirely by executing the batch file REMOVE.BAT. This will free up the memory occupied by SIMULATR and configure the COM1 device for communication with a physical Rhino XR-3 robot.

7. Enable/disable COM1 trace: <Alt/F7>

The enable/disable COM1 trace option is useful for *debugging* user programs. When the trace option is enabled, all communication through the COM1 serial port is echoed to the screen. Characters written to the COM1 device by a user program appear on the screen in white, while responses from SIMULATR appear in a nonwhite color. Responses from SIMULATR are expressed in one-byte two's complement form, so negative numbers appear as integers in the range 128 ... 255.

When the COM1 trace option is enabled, SIMULATR also enters a *single-step* mode in the sense that it pauses each time a complete command (carriage return) is sent to the COM1 device. Pressing any key will then erase the most recently echoed command and continue execution of the user program. The location on the screen where the information is displayed is determined by the cursor location at the time the trace option is enabled. When SIMULATR is installed, it starts out with the COM1 trace option disabled.

8. List command mode keys: <Alt/F8>

This is a help screen which summarizes the command mode keys <Alt/F1> ... < Alt/F10> and <Alt/Home>. Therefore, this option displays the information found in Table 2.1. Pressing any key returns control the screen active when <Alt/F8> was pressed.

9. List program mode controller commands: <Alt/F9>

This is a help screen which summarizes the program mode controller commands. These commands, which control the motion of the simulated robot, are discussed in detail in Section 2.3. Pressing any key returns control the screen active when <Alt/F9> was pressed.

10. List program mode environment commands: <Alt/F10>

This is a help screen which summarizes the program mode environment commands. These commands, which select SIMULATR display options and read workcell sensors, are discussed in detail in Section 2.4. Pressing any key returns control the screen active when <Alt/F10> was pressed.

11. Home Robot: <Alt/Home>

The home robot option returns the simulated robot to the *soft home* position specified in the file <robot.dat>. When SIMULATR is first installed, the robot starts out in the home position. This option can be used for initialization. It can also be used for recovery from the following error conditions created by a user program.

(a) Local stalls

If the user program attempts to drive a joint of the simulated robot past a hard limit specified in the file <robot.dat>, this creates a *local* stall condition because the corresponding motor on the physical robot would stall at this point. During a local stall condition, the stalled link turns red in the graphics display, and the status line shows an exclamation point next to the joint variable of the stalled joint. For convenience, the simulated link does not halt when it encounters a joint range limit (except for the jaws of the tool). Instead, when a joint is driven past its normal range of travel, a low-pitched sound is generated each time at attempt is made to move the joint. In addition, when the graphic display is selected with the <Alt/F1> key, an out-of-range error message is displayed at the bottom of the screen.

(b) Global stalls

A second type of error condition occurs when a user program attempts to move the tool tip outside its legal range: $x > 0$, $z > 0$, $r > 8$. These constraints prevent the tool tip from moving behind the robot base, below the work surface, or inside the robot body, respectively. When these constraints are violated, a *global* stall condition occurs which is indicated by having all the links turn red in the graphics display, and having an exclamation point appear at the end of the status line. Again, for convenience, the simulated robot will not stall. Instead it will move the tool into the forbidden region. However, when this occurs a low-pitched sound is generated each time at attempt is made to move the robot. In addition, when the graphic display is selected with the <Alt/F1> key, an out-of-range error message is displayed at the bottom of the screen.

(c) Abnormal program termination: <Ctrl/C>

On occasion, the user may want to terminate a program prematurely using the <Ctrl/C> key. If this is done when SIMULATR is in the graphics display mode (see the # command in Section 2.4), the screen with remain in the graphics mode. At this point the screen must be returned to the text mode to resume normal operation. This is achieved automatically with the <Alt/Home> key. Pressing the <Alt/Home> key will home the robot, clear the screen, and return to the text mode. Whenever a program is terminated with a <Ctrl/C> key, the user should then press the <Alt/Home> key to reinitialize SIMULATR.