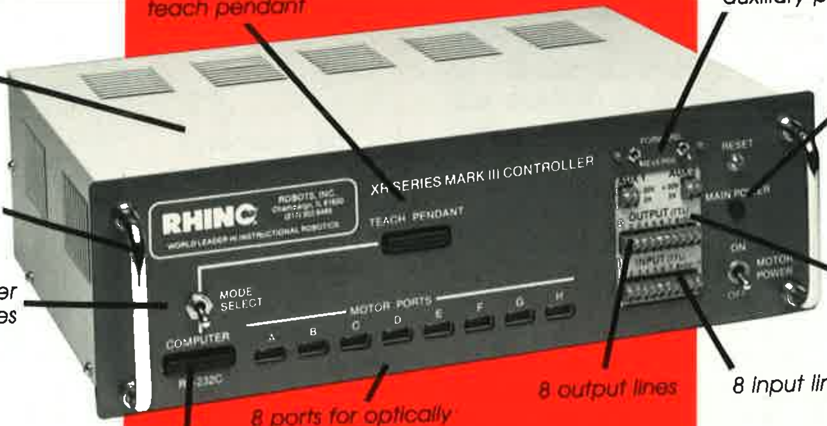
Rhino Robot's XR Owner's Manual

TABLE OF CONTENTS

XR-3 Series Mark III Controller


Optional connection for Rhino 32 key teach pendant
Switches to reverse polarity of the two auxiliary ports
Built-in power supplies
Motor power indicator light
Carrying handles for convenience
2 auxiliary, non-encoded ports
Optional switch for use of either computer or teach pendant modes
8 output lines
8 input lines
8 ports for optically encoded motors (6 for robotic arm and 2 for accessories)
RS-232C port for connection to any computer with RS-232C interface
RHINO
XR SERIES MARK III CONTROLLER
TEACH PENDANT
MODE SELECT
COMPUTER
RS-232C
MOTOR PORTS
RESET
MAIN POWER
MOTOR POWER

The increasing sophistication of automated manufacturing systems has stimulated the need for both existing and future workers to understand the concepts that make such systems work. Schools and training programs need to prepare students and trainees for use of these increasingly complex manufacturing technologies. As the leader in instructional robotics, Rhino Robots, Inc. recognizes the need for students and industrial trainees to become knowledgeable about input/output, sensor control and workcell operation in order to adequately prepare for work with real world industrial robotic workcells and automated systems.

The Rhino Mark III controller, like the robotic arm it controls, is specially designed for teaching, industrial training and research purposes. It provides unlimited potential for simulating robotic control in an industrial environment.

The built-in I/O of the Mark III controller affords Rhino users with tremendous opportunities for workcell experimentation.

Like all XR Series products, the Rhino Mark III controller is an ideal vehicle for introducing students at all levels to the subject of robotics by giving them opportunities for **hands-on** interaction with a small robotic system that costs a fraction of what an industrial model costs.

RHINO®
ROBOTS, INC.
world leader in instructional robotics

Specifications

XR-3 SERIES MARK III CONTROLLER

FEATURES

VALUE — Unlike most robotic controllers, the Mark III controls not just one type of robot, but two. Both the XR Series robotic arm and the Rhino SCARA robot can be controlled by the Mark III controller.

I/O CAPABILITIES — The built-in I/O of the Mark III controller affords Rhino users with unprecedented opportunities for workcell experimentation. It has 8 optically encoded ports, 6 to run the robot and 2 to run optically encoded motorized XR accessories for workcell experimentation. In addition, it has 2 DC motor ports with switches to reverse direction of the motors, as well as 8 additional output and 8 input lines which can be used to operate sensors and signals such as exist in the normal industrial environment.

VERSATILE — Together with Rhino's 5 axis, plus gripper, articulated robot arm, the Mark III controller can be used for a variety of applications to provide efficient training that transfers readily to the industrial setting without the costs of using a full scale robot. These applications include teaching of troubleshooting and maintenance techniques, learning about fundamentals of industrial automation and programming, conducting feasibility studies, experimenting with computer control of machinery, etc.

CIRCUIT BOARDS ARE READILY ACCESSIBLE — The accessibility of the circuit boards facilitates teaching of circuit design and troubleshooting. Full schematics down to chip level are available. Key ICs are socket mounted for easy testing and replacement. The Rhino is the only instructional controller on the market that is completely accessible for teaching and research purposes.

ONE OF THE LONGEST WARRANTIES IN THE ROBOTIC INDUSTRY — Rhino offers a 1 year mechanical warranty and 6 months electrical warranty. Unlike most other warranties for robotic controllers, the Rhino warranty is not voided if the user opens the controller.

Rhino Robots Inc. reserves the right to change any and all specifications and prices without prior notice.

Rhino Robots, Inc.
308 South State Street
P.O. Box 4010
Champaign, Illinois 61820 USA
Telephone 217-352-8485
TELEX: 3734731 RHINO ROBOTS C

Rhino has representatives throughout the USA, Canada and a large number of other countries. Please call or write for the name of the representative in your area.

Model	Mark III Controller
Part Number	FG0792
Applications	Education, industrial training, research, workcell modelling
Configuration	8 ports for optically encoded motors to operate the XR robot and accessories. Each port provides 1.6 amps at a nominal 20 volts for motor power, microswitch detection and optical encoder lines. 2 ports for non-encoded motors to operate workcell equipment. Each port provides 1.6 amps maximum at a nominal 20 volts DC. 8 inputs to read sensors for workcell development. 8 TTL level outputs to send signals for workcell development. Each line provides 24 milliamps low and 14 milliamps high.
Microprocessor	6502A
Communication Interface	RS232C using ASCII characters
Deceleration	Built in on encoded ports for smoother robot operation
Compatibility	Can be run with virtually any computer with a serial RS232C interface.
Commands	START to move motor QUESTION to check status of motor error register STOP to stop movement of a motor INQUIRY to read microswitches on motors C to H J to read status of microswitches A & B and inputs 4-1 K to read status of inputs 8-5 L to turn ON auxiliary port 1 M to turn OFF auxiliary port 1 N to turn ON auxiliary port 2 O to turn OFF auxiliary port 2 P to tell controller that next digit it receives addresses output line to be set high Q to reset controller R to tell controller that next digit it receives addresses output line to be set low
Software Available for Apple IIe or IBM PC	RoboTalk, robotic control language, Rhino-VAL emulation package, XYZ demo program, teach emulator demo program, extended version of Applesoft BASIC with Rhino Com Language Card (Apple only).
Warranty	6 months on electrical parts, 12 months on mechanical parts
Trademarks	VAL is a trademark of the Unimation Corporation. RoboTalk is a trademark of Rhino Robots Inc.

RHINO®
ROBOTS, INC.
world leader in instructional robotics

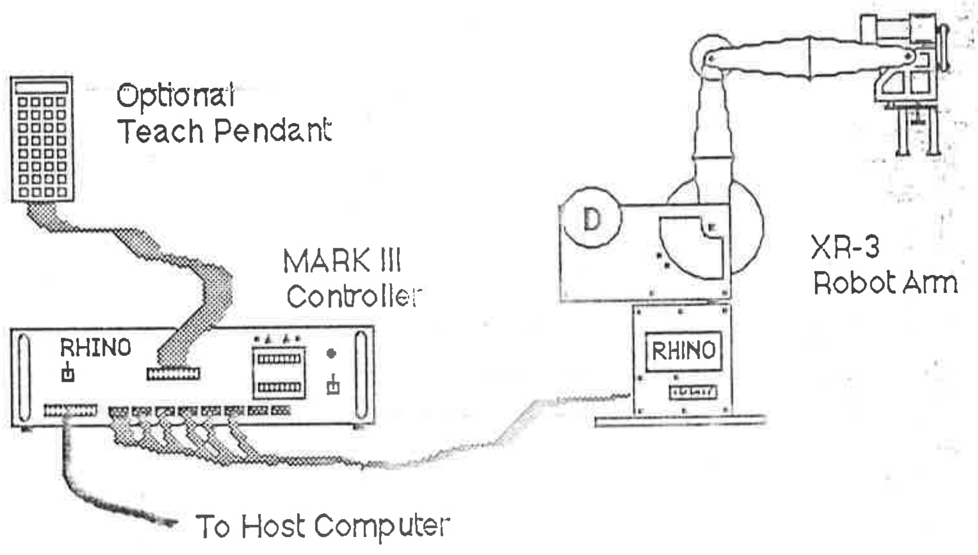
ENGINEERED AND MANUFACTURED IN THE USA

SS-0792

CHAPTER ONE

ABOUT THIS MANUAL AND THE XR SYSTEM

You are about to explore robotics the right way - with the XR Series robot and the MARK III controller from RHINO Robots Inc., the world leader in instructional robots.



Take a few minutes to read this section first. It provides a good map for following the rest of the manual. Because the XR is part of a system, we have used a loose leaf format. The manual assumes you have the basic system:

The XR-3 Series Robot Arm
The MARK III Controller

When you purchase any of the system accessories, you will receive documentation in a format that allows you to insert it into this basic manual.

HOW TO USE THIS MANUAL

This manual tells you everything you need to know to begin exploring robotics with the XR system.

1. **Start with Chapter Two.** It will guide you through unpacking, mounting, connecting and testing your XR.

2. **Move on to Chapter Four,** "Running The XR With A Host Computer." If you have one of the computers listed at the beginning of Chapter Four, go directly to **Chapter Five or Six**. Each section gives commands, programs, and programming techniques tailor-made for your computer.

   If you do not own one of the computers covered in Chapters Five or Six, use Chapter Four, "General Interface Instructions." Connect the controller to your computer according to the instructions in that section. Then, choose the section that covers **a computer most similar** to your own. For example, if you have an IBM compatible machine, choose the IBM section. Follow that section and adapt our sample programs to your machine as necessary.

3. **Read Chapter Three as a Reference Guide.** Chapter Three is used when complete information on each of the MARK III controller commands is needed. Example programs are given to help you understand the usage and format of the various commands.

4. **Read Chapter Seven after your first session with the XR.** Chapter Seven explains important maintenance checks and procedures that should be performed periodically.

5. **If you need help,** go to **Chapter Eight** for troubleshooting procedures. If you need our help, Chapter Eight gives information about the RHINO hotline, and how to send a component back to us if necessary.

6. **Chapter Nine** contains parts lists, exploded-view diagrams and ordering information.

7. **Chapter Ten** contains program listings for sample **hard home** routines.

8. **The Appendix** gives a short Tutorial on the principles of servos, closed loop control systems and the use of incremental encoders.

ABOUT THE XR SYSTEM

THE COMPLETE EXPERIMENTAL SYSTEM WITH ACCESSORIES.

The XR system is the complete EXPERIMENTAL ROBOTIC system; complete in that it contains

1. Robot Hardware
2. Controller Hardware
3. Software

the three basic components of any computer controlled system.

ROBOT HARDWARE:

The robot hardware consists of a five axis revolute coordinate robot arm with a motor driven gripper. All axes of the robot are controlled by DC servo motors using incremental optical encoders for feedback and contain limit switches which are used to position the robot to a hardware home position. At full extension (about 18 inches), the robot has a lifting capacity of one pound. Construction materials are aluminum which provides strength with low weight.

CONTROLLER HARDWARE

The MARK III is a microprocessor based eight axis motor controller which also provides two 20 VDC unencoded motor ports, eight TTL output lines and 16 TTL input lines (8 from the I/O panel and 8 from the axis limit switches). An RS-232c communication port is provided for use with a host computer allowing the computer to issue commands which operate the robot and the other features of the MARK III controller.

SOFTWARE:

Two types of software come with the Rhino system: diskette based for use in an Apple or IBM PC computer and the firmware which resides in permanent memory in the MARK III controller. The controller is designed to accept kernel commands via an RS-232c communication port. The robot and its I/O are controlled by these kernel commands which can be combined in an unlimited number of ways to create everything from simple robot movements to comprehensive robot languages.

The command structure of the XR system is designed to fully control the XR-3 robot and all the features of the MARK III controller and to interrogate the status of the controller-robot system. This ability to get information from the controller allows more sophisticated control strategies to be worked out by the investigator.

The diskette based software includes several programs which demonstrate how programs can be written in BASIC to accomplish various positioning tasks such as XYZ transformations. Also included is Rhino's generic robotic control language called RoboTalk™. This language allows the user to learn and understand the principles of robot control using software and provides a base for progressing on to higher level languages.

THE SYSTEM:

The system is designed so that the students, trainees, investigators and researchers can meet the system at the level that suits them best.

For initial familiarization and orientation, all that is needed is the basic teach pendant, controller and the robot arm. All aspects of five axis robot movement can be demonstrated and experienced by the user without ever writing a computer program.

For those users that have Apple IIe and IBM-PC personal computers available, Rhino provides full software support and teaching aids. Courseware suitable for factory floor personnel as well as students is also available from Rhino. The courses designed with the courseware may be geared for either one or two semester.

Using a host computer to control the XR system adds a completely new dimension to the study of robotics. This is best divided into two major sections, one for the academic community and one for the factory floor.

SUMMARY AND SPECIFICATIONS

That's the XR Series in a nutshell. The following tables give the specifications for the XR robotic arm.

BASIC SPECIFICATIONS FOR THE XR ARM

Vertical Reach	34 inches	86 cm
Radial Reach	24 inches	60 cm
Lifting Capability (arm extended)	1.0 lbs.	0.45 kg
Weight of XR-3	17 lbs.	7.7 kg
Weight of Mark III controller	27 lbs	12.3 kg

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

Repeatability (full extension , at grippers).........0.25 inches actual

MOTOR GEAR RATIOS AND FINAL REDUCTIONS

Axis	Motor Gear Ratio	Encoder steps /Degree of Axis Movement
Fingers	96/1	not applicable
Wrist Rotation	165.4/1	5.5
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 fully 1 sec. to close fully
Wrist Rotation	32
Wrist Flex	45
Elbow	30 — D
Shoulder	20 — E
Waist	60 — F

CHAPTER TWO

UNPACKING and SETTING UP

UNPACKING THE XR ARM

As you unpack, be careful not to tear the box or damage any of the packing materials. You can use them later to ship, store or move the XR system. You will use these containers if the robot has to be returned to the factory for service.

The XR arm comes cased between two foam packing shells. Located in the top of the packing shells is a cavity which contains two manuals; the Owner's Manual and the Software Manual. Set these aside for use later.

Now grip the foam shell, lift it out of the box and set it down in an upright position. You can most easily do this by having a helper hold the box while you pull the shell up and out; do not reach through to grasp the arm itself - you may reach in and damage the arm.

With the shell out of the box, you can see a cutout in each foam shell, see Fig 2.1. Through one of the cutouts locate two motors. Gently tip the shell on its side so that this cutout and the two motors face up.

NOTE: The encoders on each motor are part of the optical feedback system. This is the one relatively fragile component on the XR. Be careful not to grab the robot arm by the motors, encoders, or other small parts.

Now, carefully lift the top foam shell up and off, lift the arm from the bottom shell and set the arm base-down on your work surface. See Fig 2.2.

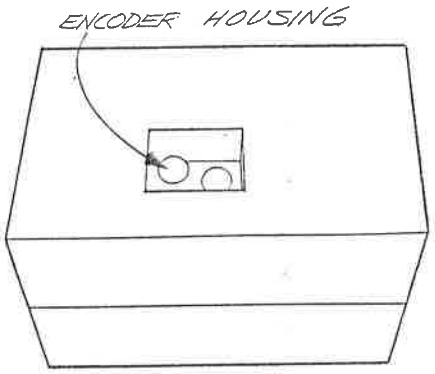


FIG 2.1

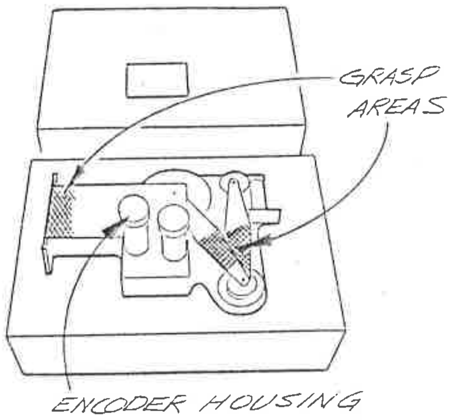


FIG 2.2

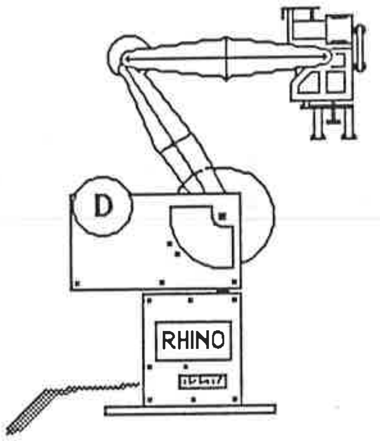


Figure 2-3
The XR-3 Robot

MOUNTING THE XR ARM ON A BASE

In this position , the XR appears deceptively short and stocky. However, when the arm is extended it becomes top-heavy and can tip over. The XR needs to be anchored to the work surface to prevent damage.

Use The Rhino Aluminum Base

If you have purchased a Rhino aluminum base, unpack it and lay it on your work surface. Included with the base is a small hardware package containing rubber feet, threaded studs and thumbscrews. First install one rubber foot in each of the corners of the base bottom (the bottom is the smooth side). Remove the two one inch long threaded Allen-head studs and screw them into the threaded holes in the base top so that 1/2" of each stud extends up from the base. Thread one of the thumbnuts onto a stud just enough to engage the threads, then tighten the thumbnut one turn.

Lift the robot arm and identify the slot at the front of the base. The front is the side with the chain sprocket near the base. Slide the slot at the front of the base around the stud and under the thumb nut, see Fig 2.4.

Align the slot at the rear of the robot base with the remaining stud. Slide the arm back until the back of the rear slot butts against the rear stud. Screw the other thumbnut onto the rear stud and tighten both thumbnuts to fasten the robot to its base, see Fig 2.5. Position the arm so that the front of the robot is to your left as you face the arm.

FIG. 2.4

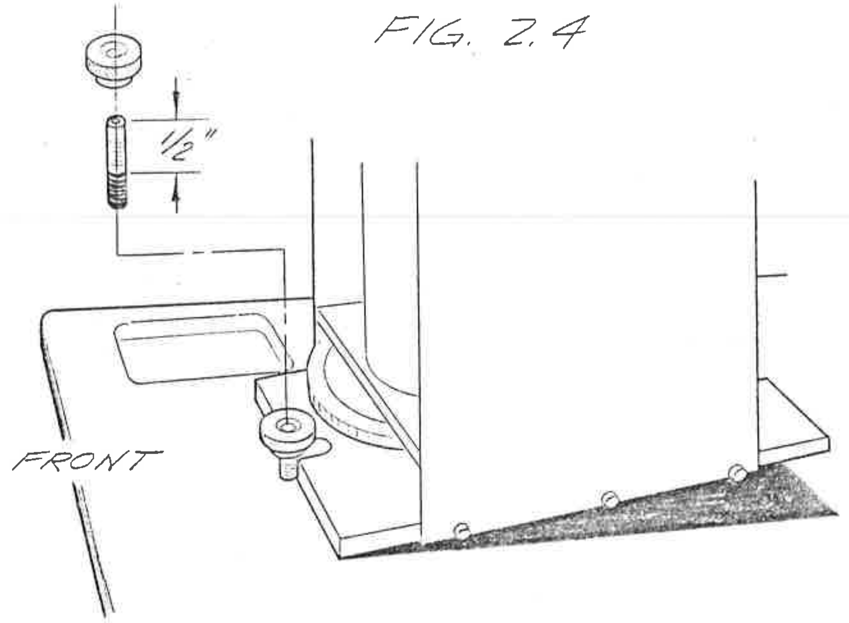
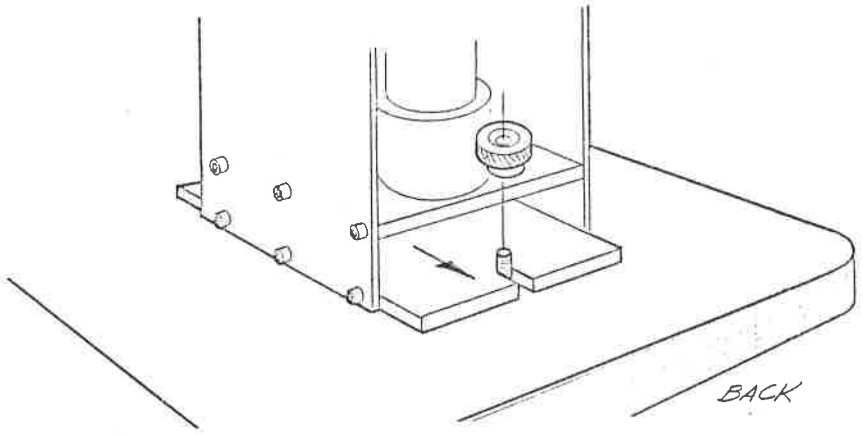


FIG. 2.5



Mounting The Arm On Your Own Base

If you did not order the RHINO-made base, you will still need to fasten the arm to something. You can clamp it to a table, bolt it to a table, or make your own base. To mount the arm, follow these guidelines:

1. Drill mounting holes at 6.50" centers.

2. Make your base about 12"x12". Any larger, and it may interfere with system components like the conveyor. Any smaller, and it will tend to be unstable. The base must also be heavy enough to hold the arm steady. As a guide, if you decide to use plywood, it should be at least 0.50" thick.

UNFOLDING THE ARM

Next, unfold the XR arm. To do so, gently apply even upward pressure on the joint that connects the lower arm to the upper arm. Do not use an abrupt movement since this may damage the high gear reduction motors. The application of steady but firm upward pressure on the joint will move the arm up and out of the nesting position and will clear the gripper from the two large 72 tooth sprockets on the robot body.

UNPACKING and CONFIGURING THE EIGHT AXIS CONTROLLER

Next, unpack the controller. Notice the two small padded paper bags; one contains the optional Teach Pendant hand unit, the other contains the Rhino data cable, the IBM adapter plug and the Tool kit. Your tool kit contains every tool you need to perform mechanical maintenance. Remove the two bags and set aside for now. You will also find the power cord in the controller carton. Lift the controller out with the two styrofoam shells on its ends. Remove the styrofoam but remember to save the packaging.

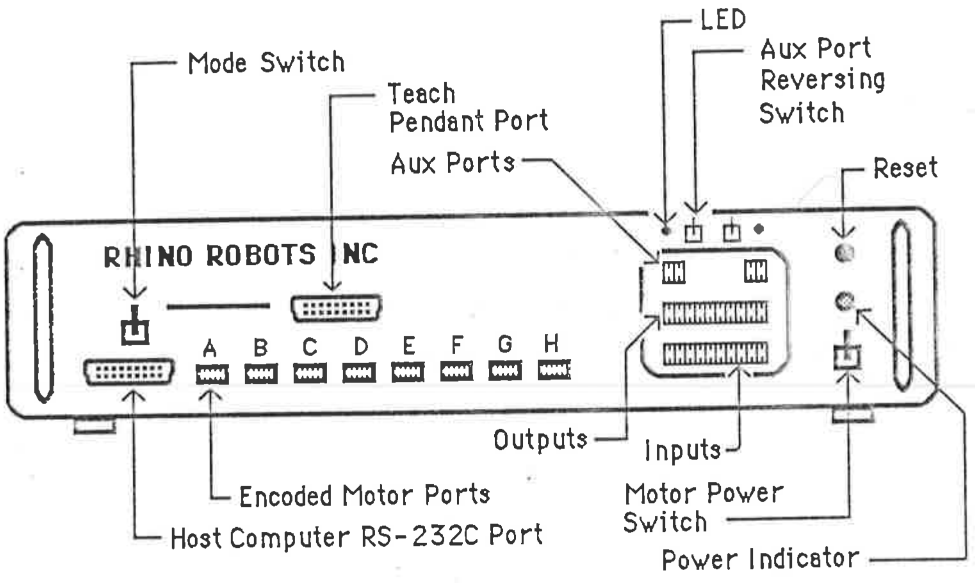


Figure 2-2
The front of the Mark III controller

Set your controller behind the arm (to the right as you face it), with the front panel facing you.

On the Mark III controller, locate the **main power** rocker switch on the back panel of the controller cabinet. Make sure it is **OFF (0)**. Locate the **motor power** switch on the right-hand side of the front panel and make sure that it too is **OFF**.

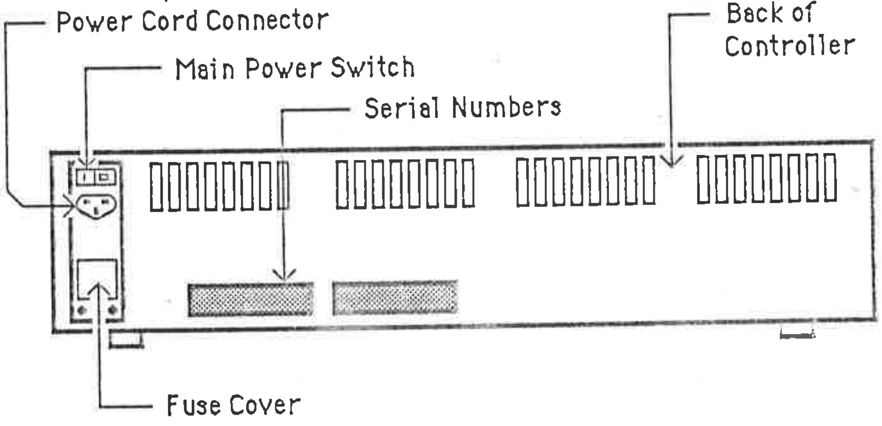


Figure2-3
Back of the controller

Now locate the power cord. Plug the female socket end of the cord into the male connector just below the main power switch, on the back panel of the controller. Plug the other end of the cord into a grounded 115 VAC outlet. If you must use a three-pronged adapter, make sure it is grounded.

NOTE: The Mark III has a switch to select power voltage. That switch is located below the main power switch on the back of the controller cabinet. As shipped, the switch is set to 120 volts. To use the Mark III with a 220 volt power source, this switch must be set to 220 and the internal power supply must be modified. Contact the factory if the higher voltage is to be used.

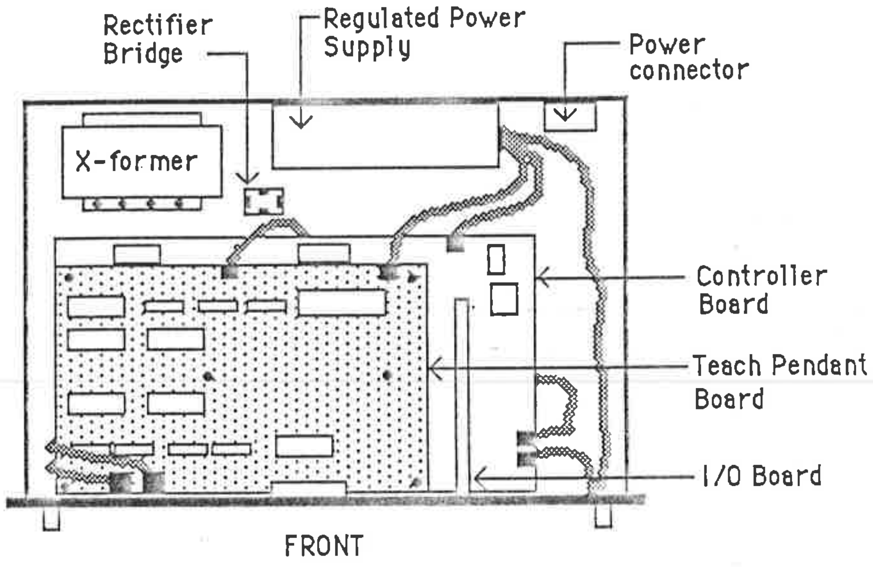
MARK III COMMUNICATIONS SETUP

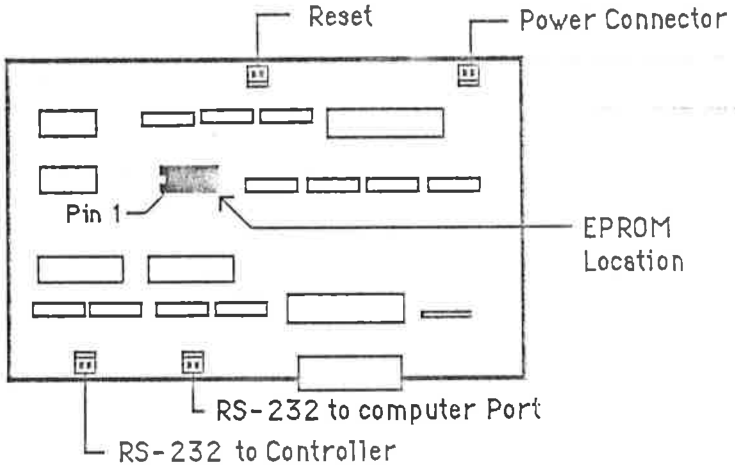
The MARK III controller comes from the factory with its communication ports configured for 9600 baud, 7 data bits, 2 stop bits and even parity. This settings allows compatability with all of Rhino Robots software offerings; if you plan to use only Rhino software or have host computers that can be configured for this communication setup, you can proceed without having to change the default settings.

However, if you need to change the communication settings, the controller can be configured for a lower baud rate or other parameters. To change these settings, open the controller cabinet by removing the top cover. Locate the 4 position DIP switch on the lower controller board; it is located in the rear right corner as you look from the front of the controller.

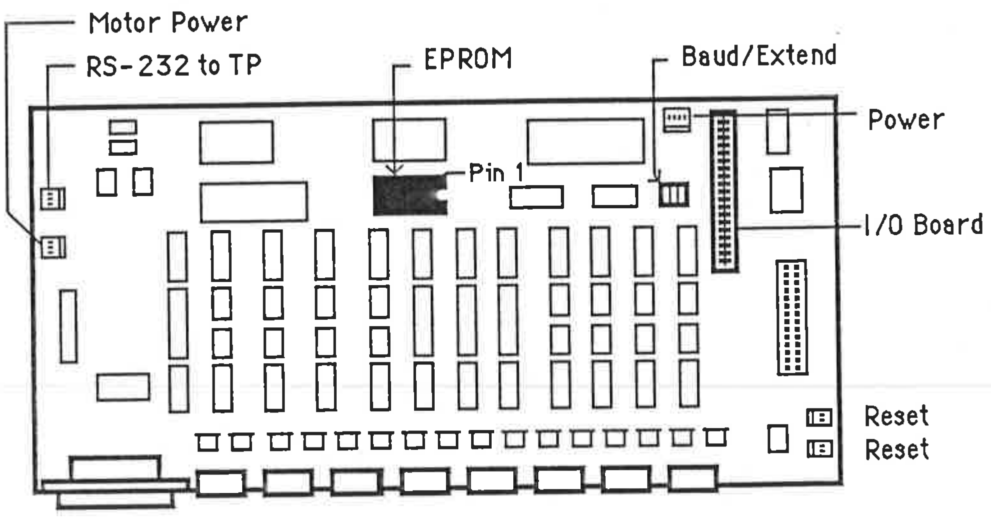
Switch number 1 should always be set ON (0 or Open). The remaining switches should be set as follows: (0=Open 1=Closed)

2-3-4 Setting	Data Bits	Stop Bits	Bit Parity	Baud Rate	
1 1 1	7	2	Even	9600	**Default**
1 1 0	7	2	None	9600	
1 0 1	7	2	Odd	9600	
1 0 0	7	2	None	300	
0 1 1	8	1	None	9600	
0 1 0	8	1	Even	9600	
0 0 1	8	1	Odd	9600	
0 0 0	8	1	None	300	





TEACH PENDANT COMPUTER CARD



CONTROLLER COMPUTER BOARD

CONNECTING THE MOTORS

Now you are ready to connect the motors to the controller. Start by locating the bundle of six flat ribbon cables at the rear of the robot body. Pull the bundle out and away from the robot body, remove the holding strap and straighten the cables.

All cables are labeled with the motor identifier to which they are connected; the letters correspond to the controller motor ports as labeled on the front panel. Insert the **A** motor cable first making sure that the plug is oriented such that the flat cable exits the plug body from the bottom. Continue inserting the remaining cables (**B-F**) in the same manner.

If you have the optional Teach Pendant, plug the large flat ribbon cable into the 25 pin connector labeled Teach Pendant.

The XR Robot system is now ready for operation. The following sections will describe the operation of the MARK III controller and how it is to be interfaced with a host computer.

CHAPTER THREE

THE COMMAND SET

THE MARK III CONTROLLER COMMANDS

The command set of the XR Robot - Mark III Controller system has been designed to allow the complete control of the Mark III 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 these kernel commands to create the higher level languages, such as RoboTalk and Rhino-VAL.

The MARK III controller has the following commands:

Command	**Description**
<return>	**Carriage return (initiate a move)**
?	**Return distance remaining**
A-H	**Set motor movement value**
I	**Inquiry command (read limit switches C-H)**
J	**Inquiry command (read limit switches A-B and inputs)**
K	**Inquiry command (read inputs)**
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 MARK III controller accepts commands as ASCII 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 the 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.

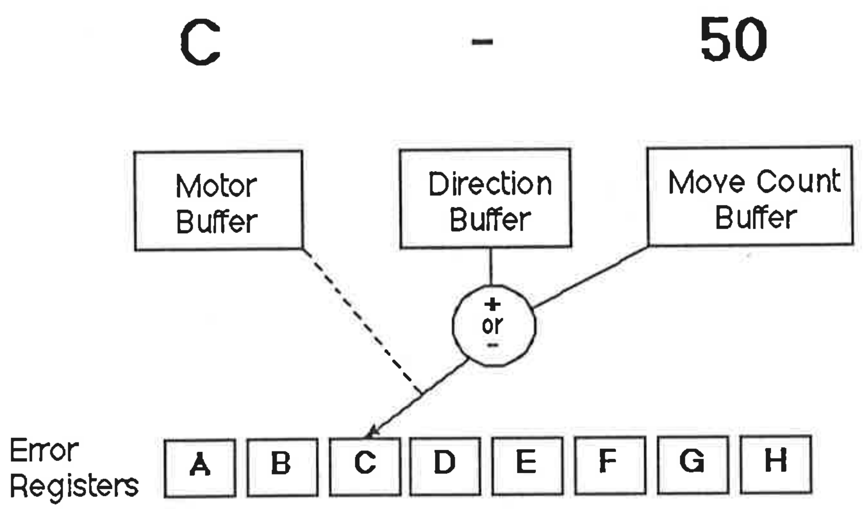


Figure 3.1
Processing of a Motor Move command

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 MARK III 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 specifier allows you to use the carriage return.

In the following command descriptions, the format and the examples of the commands

797-1099

will illustrate the use of the leading motor specifier. This will lead to a more intuitive understanding of the command set.

<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 the 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.

? 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 the controller. The command used to determine how far a motor has yet to move is the Question command.

The format of the command is

[<motor ID>]<?> <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.

A to H — Start motor commands

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

The start command is used to 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

<motor ID>[<sign>]<encoder counts><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 the 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 non-zero motor move command. It should be cleared before using commands when necessary.

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:

[<motor ID>]<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 BASIC program:

PRINT "I" <return>	
PRINT "AI" <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 the byte before transmitting it to the host computer so that no control codes will be transmitted to the 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 0 (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 controller another command. See detailed examples under the sections on running the robot with the Apple IIe and the IBM-PC.

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 the host computer, 32 is added to the returned value to ensure that no ASCII control character is returned to the computer. You use the **J-INQUIRY** command just like the **I-INQUIRY** command.

The format of the command is

[<motor ID>]<J> <return>

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

Examples of the command as issued from a BASIC program:

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

When interpreting the bits returned by the 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 the byte before transmitting it to the host computer so that no ASCII control codes will be transmitted to the 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 Apple IIe and the IBM-PC.

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 the **I and J** commands.

The format of the command is

[<motor ID>]<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 the byte before transmitting it to the host computer so that no ASCII control codes will be transmitted to the 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 Apple IIe and the IBM-PC.

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

[<motor ID>]<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)

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.

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

[<motor ID>]<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)

O — Turns Aux. Port #2 OFF

Turns Aux Port #2 OFF

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

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 MARK III controller output lines provide TTL level signals.

The format of the command is

[<motor ID>]<P><output line number> <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)
```

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 to ensure that the controller is at a known (reset) state when the program starts.

R Set Output Line Low

Sets an Output line LOW

The **R** command is similar to the **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 MARK III controller output lines provide TTL level signals.

The format of the command is

[<motor ID>]<R><output line number> <return>

Examples of command use as issued by a BASIC program:

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

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

<motor ID><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 ID 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.

USING THE COMMANDS

Detailed instruction and examples on how to use the commands are given in the sections on using the Apple IIe and the IBM-PC as the host computer. The examples given send the command in the context of a subroutine, although ordinary Print, Peek and Poke commands and machine code (assembly language) routines can also be used to send and receive information. The commands used will depend on the language that you are using. See the manual for your particular computer and the language that you are using to see how these commands can be used.

The fastest way to send commands and receive information is with the use of assembly language (machine code) programming. Sophisticated users interested in developing complicated command structures for the Rhino XR system will use machine code, however, the scope of this manual can not cover machine code programming.

Two fairly complicated **sample programs** are provided on disk with the XR system to show beginning users how to program complicated programs in machine code and/or in BASIC. Interested experimenters will do well to study these programs at length. One of the programs (the teach pendant emulator) actually emulates the teach pendant on the keyboard of the computer. The other (XYZ) is an XYZ program that controls the robot in XYZ coordinates. Of course your entire program does not have to be in machine code. You can take useful machine code routines out of the sample programs and incorporate them into your BASIC programs.

RoboTalk™

Those users wishing to use a **higher level language** to control the XR system can use **RoboTalk™** to do all their programming. RoboTalk is provided with each system. See your RoboTalk Manual.

Rhino-VAL

Those users wishing to use an **industrial robotic language** to control the XR system can use Rhino's emulation of **VAL™**, the Unimation language, to do their programming. Rhino-VAL is provided on disk and versions are available for the Apple IIe and the IBM-PC.

CHAPTER FOUR

RUNNING THE XR WITH ANY HOST COMPUTER

If you have one of the computers listed below, we suggest that you skip directly to one of the two following chapters that covers your computer.

COMPUTER	CHAPTER
Apple IIe	**Five**
IBM-PC	**Six**

Each chapter provides step-by-step instructions on how to connect the controller to the computer, a tutorial on the fundamental controller commands, and important programming techniques specific to the XR. Each section is specific for the computer covered.

If you own either of the computers listed above, you also received a software package designed for your machine. If your immediate concern is running the XR, and not programming, we recommend that you refer to the manual that came with the software. Then, if you want to learn more about programming, return to the section that covers your computer.

If you don't own one of the computers covered individually, go to the following pages for the general information you need to interface your computer to the RHINO system. We suggest that you read these pages and then skip to the section which covers a computer that most closely resembles your own. For example, if you have an IBM compatible machine, skip to the section on the IBM PC. Then adapt the sample programs to your own computer.

GENERAL INTERFACING INSTRUCTIONS

Electrical connections:

No matter what computer you use as a host, it must have an RS-232c serial interface, capable of both sending and receiving data. The Mark III Controller uses only three of the 25 communication lines on the DB 25 connector:

Line 2 carries data transmitted by the controller, received by the host computer

Line 3 carries data received by the controller, sent by the host computer

Line 7 is the common data ground line.

Therefore, your computer's RS-232c connections must be configured as follows:

Line 2 should be configured to receive data
Line 3 should be configured to transmit data
Line 7 should be configured to be the data ground

Handshaking:

The Mark III Controller does not use a handshake protocol. Therefore, the DB25 connectors must be modified. Jumper wires must be soldered to connector pins so that, essentially, your computer shakes hands with itself.

Usually, jumpers should be soldered between pins 4 and 5 and between pins 6, 8 and 20. To know exactly what to do, you will have to read the manual for the RS-232c port for your particular computer. You may be able to use the data cable supplied with the Rhino robot system.

Consult your computer manual to determine what signals it requires, you may have to make further modifications. Remember to treat the controller as a serial (ASCII) I/O device.

DATA FORMAT

Configure your computer's RS232c port for the following data format:

9600 Baud
7 data bits
even parity
2 stop bits

Although it depends on your computer, you'll probably have to set up the data format with software instructions at the start of each of your programs. You can find illustrations of how to do this in the sections covering the individual computers.

OPEN THE PORT

You also have to configure your computer so that commands are routed to the proper port. Using your computer manual, set up as if your computer were transmitting to an ASCII serial I/O device. This may require some switch setting, but more likely it will require some software instruction. Again, consult your computer manual.

You may be able to use the information provided in this manual for the IBM-PC and the Apple IIe as a guide to help you.

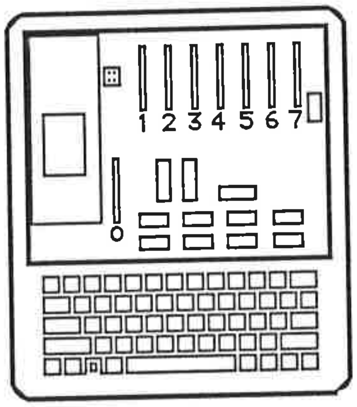
CHAPTER FIVE

RUNNING THE XR WITH AN APPLE IIe

INTERFACING

To interface your Apple IIe to the MARK III controller you will need the Apple Super Serial card. This card allows the computer to use the RS232c serial communication standard which the MARK III requires.

Install the Super Serial card in the **number two slot** of your Apple IIe (some higher level languages from Rhino require placement of the serial card to be in slot 1, refer to the appropriate software manual). To install the card, turn **off** the power and remove the top of your Apple according to the manual. Locate the peripheral board sockets at the back of the circuit board. The number 2 slot is the second socket from your left. Make sure you plug the board in with the component side facing toward your right as you look at the computer keyboard.



The Apple IIe

Rhino provides an RS 232C cable with DB25 connectors for a proper connection between your Apple IIe and the Mark III Controller. Install it as follows:

1. Be sure that the **main power** and **motor power** switches on the controller are **off**, and that the Apple power is **off**;

2. On the **Mark III** controller, plug one end of the cable into the port on the front of the controller labeled "**Computer**", plug the other end of the cable into the serial interface card in the computer.

Setting Up the Communications

1. Insert the system master disk (that came with your computer) into your disk drive.

2. Turn on your Apple.

3. Turn on the **controller main power** switch.

4. Turn on the **controller motor power** switch.

5. Press the **reset** button the Mark III controller.

Whenever you program your Apple to run the XR, you must begin by setting up communications between the computer and the robot controller. The first lines of any BASIC program must contain these initialization commands.

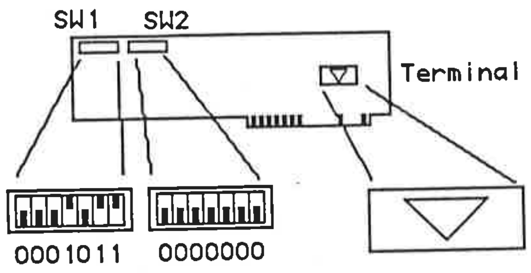
```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
```

In the above lines of codes, the variables are used as described below:

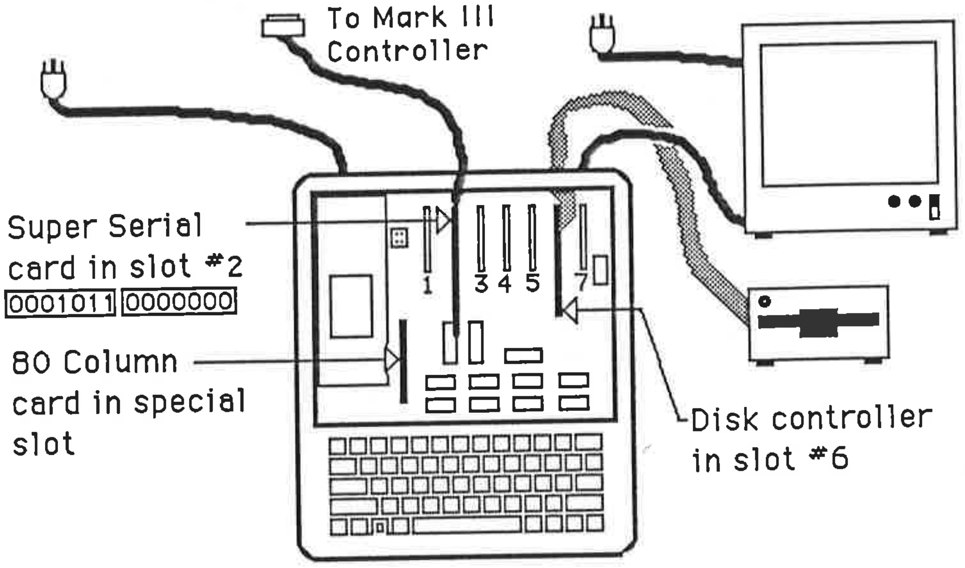
DA=49320	sets the memory address for the data (for slot 2) (use 49304 if you want to use slot 1)
ST=DA+1	sets the memory address for status register
CO=ST+1	addresses the command register
CL=CO+1	addresses the control register
S=16 and R=24	are the masks for bits 3 and 4
POKE ST,0	resets (clears) the UART
POKE CL,190	sets the BAUD rate at 9600
POKE CO,101	sets 7 data bits, even parity, 2 stop bits

These explanations are given for reference purposes. All you have to remember about the communication set up lines is that you must start all your BASIC programs with them.

The Super Serial card must also be configured for the correct BAUD rate, parity, etc. by the setting of its DIP switches. Refer to the following diagram to setup your Super Serial card.



Super serial card switch settings for
9600 BAUD, 7 data bits, 2 stop bits and even parity

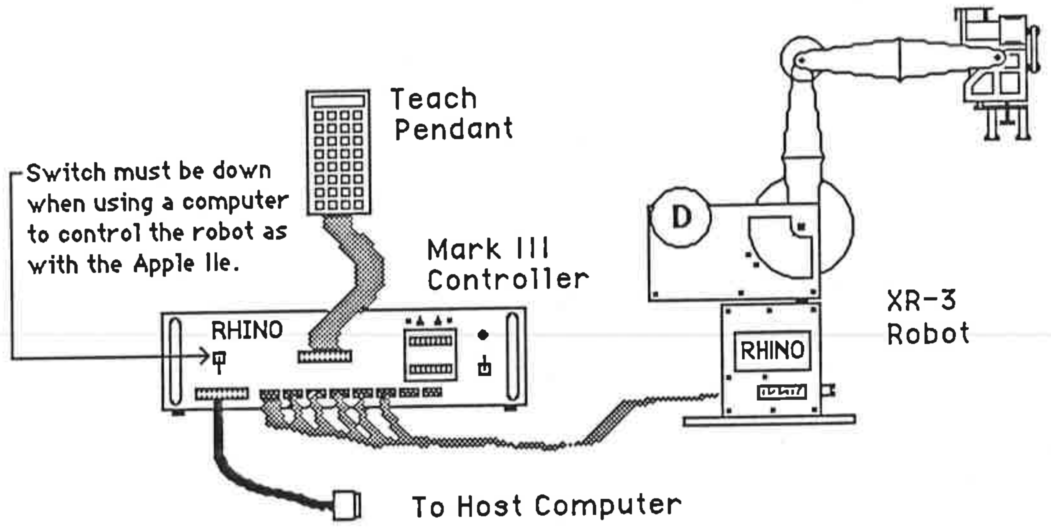

To Mark III Controller
Super Serial card in slot #2
0001011 0000000
80 Column card in special slot
1
3 4 5
7
Disk controller in slot #6

Typical set up for the Apple IIe

Moving The Motors With The START Command

Once you have initialized your computer you can use it to communicate with the robot controller. Type in the lines listed below.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  A$="F+50"+CHR$(13) :  GOSUB 100
40  END
```


Teach Pendant
Switch must be down when using a computer to control the robot as with the Apple IIe.
Mark III Controller
RHINO
XR-3 Robot
RHINO
To Host Computer

Typical set up of the robot and controller.

In the above lines:

"F+50" forms the START command, one of the fundamental commands you can send to the XR controller with your host computer. The START command tells the controller to start a designated motor and move it a specified number of steps in a specific direction.

The "F" designates that the **F** motor is to be moved

The "+" indicates the direction of motion

The "50" specifies the number of encoder positions the motor will turn -- the distance to be moved.

To send the START command, or any of the controller commands, it is best to use a subroutine. That's why in line 30 above we set the variable A$ equal to the command. A$ is the variable we will use to carry the command string to the subroutine.

The START command must always be followed by a carriage return, hence the command is followed by CHR$(13). CHR$(13) is the same as HEX (0D). GOSUB 100 simply calls the subroutine that sends the command to the controller.

Here is a standard subroutine that you can use, to send any of the fundamental commands to the controller. Type it in as it is written below.

```
100 FOR J=1 TO LEN(A$)
110    WAIT ST, S
120    POKE DA, ASC (MID$(A$,J,1))
130 NEXT
140 RETURN
```

Each line of the above subroutine functions as described below:

100	picks up the characters in the command one at a time
110	waits until the last character has been sent before sending the next character
120	calculates the ASCII value of the character and transmits it to the controller
130	loops back for the next character (until the last character)
140	goes back, after the last character is sent, to the next program line in the main program

Now, watch your XR arm as you run this short program (type RUN, then <return>). The XR will rotate at the waist and then stop. Now go back to line 30 and change it so it looks like this:

```
30 A$="F-50"+CHR$(13) :  GOSUB 100
```

Run the program again and observe that the XR arm now turns in the opposite direction. The "-" sign made the difference. We used a "+" sign in our first example. (Even if the + sign is omitted, the + direction will be assumed.)

You now have all you need to move each motor individually. Go back and change line 30 so that A$="E+50" (don't forget the rest of the line). Remember, the letter designates the motor to be moved, so you need only change the letter to specify a new motor. Run this and observe the **E** motor movement.

Do the same for the **D, C** and **B** motors. Don't use the **A** motor just yet because it has a very limited travel capability. Familiarize yourself with each axis movement as you go.

For now, limit the number of encoder steps in a move to 50.

More About The START Command--The Error Registers

Now that you have used the basic START command, you are ready to learn more about it.

Consider our first START command **"F+50"** as an example. When we sent that command (using the output subroutine), the motor "F" moved. But much more than that happened.

The MARK III controller maintains an 8 bit error register for each of the 8 motors. When the error register for a specific motor is zero, the controller removes all power to the motor and the motor remains stationary. When the error register is non-zero, the controller connects either a positive voltage or a negative voltage to the motor. The polarity of the power is a function of what is in the error register. If the error value is positive, the motor moves in one direction and if it is negative, the motor moves in the other direction. Before we sent a command, the error registers for all the motors were zero.

A START command adds the motor movement value to the error register for the motor specified. In our example, that motor was the **F** motor. The value added was 50.

As soon as a value is added to the error register for a motor, the controller starts that motor in the direction that will take the error register to zero. As the motor moves, the encoders are read and the count in the error register is decremented. When the error register reaches zero, the power is turned off. So, when our START command added 50 to the F motor register, the controller moved that motor 50 encoder steps in the right direction to return the error register to zero.

The command **"F-50"** adds a value of -50 to the F motor error register. When the controller receives a "-" sign, it moves the motor in the opposite direction from the + direction to reach the zero error position.

Remember:

1. A START command adds a value to the designated motor's error register.

2. The value added to the error register represents the number of steps the designated motor encoder must move to return the register to zero.

3. The controller software is designed so that motors seek the zero error condition.

4. The sign in the START command designates direction of motion.

With this in mind, we move on to the QUESTION command.

The QUESTION Command

After you have run each of the motors, clear your Apple's memory. Now type in the program listed below.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  A$="F+50"+CHR$(13) :  GOSUB 140
40  A$="F?" :  GOSUB 100
50  IF W>0 THEN 40
60  A$="F-50"+CHR$(13) :  GOSUB 140
70  A$="F?" :  GOSUB 100
80  IF W>0 THEN  70
90  END
100  GOSUB 140
110  WAIT ST, R
120  W=PEEK(DA)-32
130  RETURN
140  FOR J=1 TO LEN(A$)
150        WAIT ST, S
160        POKE DA,ASC(MID$(A$,J,1))
170  NEXT
180  RETURN
```

Run the program and you'll see that it simply turns the waist in one direction and then in the other. You might be wondering why we did not follow line

```
30  A$="F+50"+CHR$(13) : GOSUB 140
```

with

```
40  A$="F-50"+CHR$(13) :  GOSUB 140.
```

Instead, on line 40 we followed the "F+50" START command with "F?". Why did we do this?

Remember that a START command adds a value to a motor's error register. If we sent an "F-50" immediately after an "F+50", the -50 count in the second command would combine with the remains of the +50 command and send the F motor in the opposite direction before it completed the full "+50" steps in the first START command.

We had to be sure that all 50 original steps were executed before we sent the controller next 50 steps, which are, in the opposite direction. (If the steps are in the same direction, they can be added to the ongoing command as long as the error registers do not overflow. More about this later.)

How The QUESTION Command Works

The QUESTION command instructs the MARK III controller to return the current value of the error register for a specific motor. Specifically, it asks how much further the motor must travel before the error register is zero.

In our example, the START command on line 30 adds 50 steps to the F motor error register. Line 40 uses the QUESTION command to request the count in the error register for the F motor. The second half of line 40 calls the subroutine at line 100. The subroutine at line 100 uses the program you have already seen to send the START command (it begins on line 140). But it also incorporates a subroutine to receive the answer to the QUESTION command. You can use this subroutine, as written, whenever you use the QUESTION or similar inquiry commands in your own programs.

In our program, we want to be sure that all 50 steps sent in the original START command have been executed before we send the second START command (in the opposite direction). We use the QUESTION to determine how many steps are currently left in the move. With line 50 we loop through the QUESTION command until the answer is zero. We know that when the answer is zero, all 50 steps originally asked for have been executed. Only then does the program move on to send the second START command.

More About The QUESTION Command. Making A Longer Move

So far, our sample programs have limited the START commands to 50 steps. The reason has to do with the capacity and use of the error registers. Two factors create problems.

1. The largest signed decimal number we can store in an 8 bit error register is 127 (1 bit for the sign of the number, and seven bits to represent up to 127). Numbers larger than 127 will overflow the register. When this happens, the register overflows into the sign bit. This reverses the sign in

the error register and thus reverses the motor. In other words, the motor can start to move in the opposite direction to that specified in the START command if the register overflows.

2. When the controller receives a QUESTION command, it automatically adds 32 (hexadecimal 20) to the answer before it transmits it to the host computer. It does so because some computers treat values below decimal 32 (HEX 20) as commands instead of data. Note that in the subroutine, at line 120, 32 is subtracted from the answer received in response to the QUESTION command. This compensates for the 32 added by the controller. Since the communication line is set for 7 data bits, the largest value that can be returned is 127 which means that the largest value the controller can send is 127-32 or 95.

A motor's error register can be zero, but it can never be lower (because we use the sign bit for direction of travel of the motor). In other words, you can find how far a motor is from the zero position but you cannot find out what direction the zero position is in. The error count is always positive. By adding 32 to every value it transmits to the host computer, the controller ensures that the computer can never receive an answer less than decimal 32 (HEX 20).

EXAMPLE - Suppose you send a START command of 120 steps. The motor starts to seek zero immediately. If the controller receives a QUESTION when the motor has traveled 20 steps, the answer sent back (the number of steps until zero) will be 100. But the controller will add 32 to the 100 before it sends the answer back to the host. 132 is more than you can represent with 7 bits so the communication line will truncate the 132 to 7 bits and an error will occur.

All this may seem puzzling to you at first, however you will begin to understand the system as you work with it. If you are confused, just keep this rule in mind:

DON'T LET THE ERROR REGISTER EXCEED 95

"But," you may be asking, "how do I get the XR to execute a move longer than 95 steps?"

The answer is to first add to a motor's error register with a START command. Monitor the status of the error register with the QUESTION command. Then, add to the register with another START before the register reaches zero, but do not add so much as to exceed the 95 limit.

The following program shows a simple way to achieve a move of 500 steps.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  FOR I=1 TO 10
40          A$="F+50"+CHR$(13) :  GOSUB 140
50          A$="F?" :  GOSUB 100
60          IF W>45 THEN 50
80  NEXT
90  END
100  GOSUB 140
110  WAIT ST,R
120  W=PEEK(DA)-32
130  RETURN
140  FOR J=1 TO LEN(A$)
150        WAIT ST, S
160        POKE DA, ASC(MID$(A$,J,1))
170  NEXT
180  RETURN
```

This program uses two key techniques to accomplish the 500 step move.

1. The FOR / NEXT loop that begins on line 30 divides and conquers by adding 500 steps to the **F** motor error register, 50 steps at a time. It loops the computer through line 80 ten times, then exits the loop.

2. The IF / THEN statement on line 60 loops the computer through the QUESTION command on line 50 until the answer is 45 or less. When the answer is 45 or less it is safe to continue through the FOR loop and add the next 50 steps without exceeding the 95 limit.

Remember that in our simple program to rotate the **F** motor back and forth, the program looped until all of the first 50 steps were executed. This was necessary because the second move was in the opposite direction to the first, and would have reversed the motor before it completed the original 50 steps.

But for the long move above, we do not have to wait for the error register to reach zero before adding to it. We want to <u>avoid</u> letting the error register reach zero, because if it does, the motor will stop. And we want to keep adding to the register without exceeding 95. In fact we want to keep the error register as close to 95 as possible as long as we have encoder positions that need to be moved.

You might want to think of an error register as if it was a gas tank. You never want it to be empty, but you can't allow it to overflow either. Strategically this allows you to go the longest distance between gas stations if you don't know how far apart the stations are. More about this later.

The program above showed one way to do this. It added 50 to the register, and as soon as there was room for another 50, it added another 50.

The program worked as an illustration, but it has drawbacks. The most important drawback is that it waits for the register to come all the way down to 45 before it adds anything to it. This is not a problem here, but as you program more and more complex moves, and connect accessory motors to the controller, the computer will get busier and busier, and timing will become more critical. You will have less time to check and fill the registers.

A better way to do this is to always fill the register as full as possible. Instead of deciding beforehand when and how many steps to add, let the computer calculate how many steps it can add without exceeding 95, and automatically add that many steps. The program below does just that.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  N=500
40  H=95
50  IF N<=H THEN 100
60  A$="F+" + STR$(H) + CHR$(13)
70  GOSUB 160 :  N=N-H
80  A$="F?": GOSUB 120 :  H=95-W
90  GOTO 50
100  A$="F+" + STR$(N) + CHR$(13)
110  GOSUB 160 :  END
120  GOSUB 160
130  WAIT ST,R
140  W=PEEK(DA)-32
150  RETURN
160  FOR J=1 TO LEN(A$)
170        WAIT ST,S
180        POKE DA, ASC (MID$(A$,J,1))
190  NEXT
200  RETURN
```

In line 30 of the above program, N represents the total number of steps we want the motor to move. We used 500, but you could change the value of N to any number of encoder steps.

We use H as the variable for the number of steps we add to the error register. In line 40, we assign H an initial value of 95; the maximum number we can safely add to the error register. If your total move was less than 95, this would be taken into account in line 50.

In line 60, we send a START command that is slightly different from the ones we have used up to now. Instead of specifying the number of steps, we use the variable H. Line 60 will add H number of steps to the F motor error register. Since we have assigned H an initial value of 95, the computer will add 95 the first time it executes line 60.

Line 70 calls the subroutine to send the START command on line 60.

Line 80 sends a QUESTION command and calls the subroutine to send the command and receive the answer (W).

Line 80 also assigns H a new value, (95-W). Since W represents the number of steps the F motor must travel before the error register is zero, 95-W is the number of steps we can add to the error register without exceeding 95.

Line 90 loops back to line 50. Line 50 checks whether H is greater than N. If H is not greater than N, the computer goes to line 60.

Line 60 sends the START command with H steps.

The program will repeat this loop until H is greater than or equal to N. Remember, line 70 subtracts H from N each time H has been added to the error register. So N, our original number of steps, gets smaller each time through the loop.

When H is greater than N, the computer goes to line 100, another START command. This START command, however, adds N steps to the error register instead of H steps. At this point, sending N steps completes the original N (500) steps.

As you can see, this is one way of keeping the error register as full as possible. When you write programs to run several motors, this will be critical. Since the program will have to check more than one error register (by means of the QUESTION command), it is important to fill the error register completely each time the QUESTION command is used. Otherwise, the error register may get to zero (the motor stops) before the computer gets back to check and refill the register.

The STOP And INQUIRY Commands

In this section we introduce you to the STOP and INQUIRY commands. We'll talk about each separately and then provide a sample program that uses both.

The STOP Command

When you use a START command, a motor will travel the assigned number of steps and stop. But as you develop programs of your own, you'll need to stop motors under specified **conditions**.

For example, many applications present the possibility of running the robot arm into an object. If the robot crashes into an obstacle, you need a way of recovering from the accident without losing your program and positional information. A servo motor will try to keep moving as long as it has power (ie., as long as its error register is non-zero). The STOP command enables us to bring a specific error register to zero on command.

An example of the STOP command looks like this:

A$="FX"

As in the other commands, A$ carries the command through the output subroutine (the same subroutine as was used to send the START command).

F designates that the "F" motor error register is to be brought to zero.

X is the actual stop command.

With the STOP command, your programs can incorporate protections against stalls. You can use the QUESTION command to check the status of an error register. The program would send the controller a STOP command in the event that the (a non-zero) error register status condition did not change during a number of successive checks.

Another use of the STOP command has to do with the microswitches mounted on the XR. The next section, which covers the INQUIRY command, will illustrate the use of the STOP command with the INQUIRY command.

The INQUIRY Commands. Reading The Microswitches

The XR-3 is equipped with six microswitches, one for each of the six motors on the arm. Each axis closes it's switch at a defined point. Therefore, we have an identifiable and repeatable orientation point for each axis on the XR; the point at which the microswitch is closed.

The INQUIRY commands (**I** and **J**) gives us the means to determine whether any of the switches on the robot arm are closed at a given time. The Mark III controller can read switches on all its ports. So, as your XR is connected now, the controller reads the switches as follows:

Motor	**Port**	**Joint**	**Can controller read switch?**	**Command Used**
A	A	Fingers	Yes	J
B	B	Wrist	Yes	J
C	C	Wrist flex	Yes	I
D	D	Elbow	Yes	I
E	E	Shoulder	Yes	I
F	F	Waist	Yes	I
G	G	Accessory	Yes	I
H	H	Accessory	Yes	I

In the standard configuration, with motors **A-F** connected to ports **A-F** on the controller, the **I** INQUIRY command reads motors **C-F** on the XR-3 and **G** and **H** on the accessories.

In order to read the **A** and **B** port switches it is necessary to use the **J** INQUIRY command.

Using the INQUIRY Commands

The INQUIRY commands work much like the QUESTION command. Each requires the same subroutines to send the command and to receive the answer. The controller adds 32 to the answer before it transmits it to your computer to avoid ASCII command codes, and the receiving subroutine subtracts 32 to yield an accurate answer.

But instead of designating individual motors when you send the INQUIRY, you send the command and interpret the answer according to the port you wish to read. The command itself looks like this:

```
A$="I" : GOSUB XXX
```

or **A$="J" : GOSUB XXX**

where XXX is the number of the same subroutine that you've used for the QUESTION command.

Interpreting The Answers

The key to using the INQUIRY command is interpreting the answer sent back from the controller. Remember that the controller communicates by using one byte only, and that the single byte answer uses one bit in the byte for each switch.

The bits are assigned to the ports as follows in the "I" command:

Port	Bit		
C	0		
D	1		
E	2		
F	3		
G	4		
H	5		
-	6	not used	
-	7	not used	MSB.

A microswitch can either be open or be closed. When a switch is open, the corresponding bit contains a one. When all switches are open, all six bits contain a one, which has a decimal value of 63. (In fact, the controller sends a 95. Remember that the controller adds 32 to anything it sends to your computer to avoid command codes. Our subroutine to receive the answer automatically subtracts 32 to yield the real value.)

If a switch is closed, the corresponding bit contains a 0, and the answer to an I-INQUIRY will be 63 minus a value unique to the open switch and bit. The table below illustrates what happens when one switch is closed and all others are open.

IF THE SWITCH AT PORT X IS CLOSED, AND ALL OTHERS ARE OPEN	THE ANSWER RETURNED IN RESPONSE TO AN INQUIRY BECOMES. (Ignoring the 32 count offset)	
If X=C	62	(63-1)
If X=D	61	(63-2)
If X=E	59	(63-4)
If X=F	55	(63-8)
If X=G	47	(63-16)
If X=H	31	(63-32)

As you can see, when a switch is closed, a value unique to that switch is subtracted from 63. If we could be sure that only one switch is closed at any one time, things would be simple. We could just send the I-INQUIRY command and check the answer to determine if the switch we were checking for was closed.

The problem is that at any one time, more that one switch may be closed. The answer received will be 63 minus the sum of the values for all closed switches. For example, if switches **C**, **D**, and **E** were closed when we sent the I-INQUIRY, the answer returned would be 56 ((63-(1+2+4)).

To check for any one switch, we must mask the answer for any other switches that might be closed.

The following subroutine does just that. We would call this subroutine after sending an INQUIRY and receiving the answer. The variable W represents that answer, and we are checking to see whether the **D** microswitch is closed.

```
100 C=0 : D=0 : E=0 : F=0 : G=0 : H=0
110 IF W>31 THEN H=1 : W=W-32
120 IF W>15 THEN G=1 : W=W-16
130 IF W>7 THEN F=1 : W=W-8
140 IF W>3 THEN E=1 : W=W-4
150 IF W>1 THEN D=1 : W=W-2
160 IF W>0 THEN C=1
170 IF D=1 THEN XXX (go to desired portion of program).
```

Let's run through the subroutine with an actual value to see how it works. Assume that the **C** and **D** switches are closed, but that we only care about the D motor. Remember that if all switches are open, the answer returned in response to an I-INQUIRY command is 63.

Remember also that if a switch is closed, the answer returned will be 63 minus a unique value for that switch:

SWITCH CLOSED	CONTROLLER SUBTRACTS
C	1
D	2
E	4
F	8
G	16
H	32

Given this, we know that if switches **C** and **D** are closed, the answer returned in response to the INQUIRY command will be 60.

Let's see how our subroutine determines whether the **D** switch is closed.

Line 100 assigns a variable to each switch. It simply uses the letters assigned to each port. It assigns the value zero to each variable to initialize it to a known quantity. As discussed above, zero, corresponds to an open switch.

Line 110 checks to see whether the **H** switch is closed. We know that when W>31, the **H** switch is open. We know this because even if the **H** switch were closed and all others open, W would be at least 31. Therefore, if W>31, the **H** switch must be open. Since W is sixty the H switch is open, and we can now set H=1 to indicate that it is open.

If H is open, we know that 32 is part of W. Before checking the **G** switch, we must mask the 32. That's why before moving to line 120, we set W=W-32. So now W=28.

Line 120 checks the G switch. We know that if W>15, the G switch must be open because even if all other switches were open (remember we've masked the **H** switch),W would be only 15. Now G=1 and we can mask for 16 because we know it is open. Now W=28-16=12.

Understand how the routine checks for each switch and masks for its value before reading further. Run through line 130 with 12, then take the new W and run through 140.

After line 130 and line 140, W should equal 0. The next line, 150, checks for the D switch. Let's see what happens when a switch is closed.

```
150 IF W>1 THEN D=1:W=W-2
```

Since W now is 0, and 1 is not greater than 1, D remains zero. W remains 0, as there is no value to mask.

Line 160 checks for the only remaining switch, the C switch.

```
160 IF W>0 THEN C=1
```

Since W is still zero, and zero is not greater than zero, C remains zero. Since all switches have been checked, the subroutine ends here.

With the subroutine complete the variables are now as follows:

```
C=0   (C switch is closed)
D=0   (D switch is closed)
E=1   (E switch is open)
F=1   (F switch is open)
G=1   (G switch is open)
H=1   (H switch is open)
```

Since we are checking for the **D** switch, we have to use an IF... THEN ... statement to make the results of the subroutine useful. For example:

```
IF D=0 THEN XXX
```

where XXX represents the line we want to execute if the D switch is closed.

The STOP and INQUIRY Commands. A Sample Program

The program below supposes that an accessory (the carousel) is connected to the **G** port on the controller. It runs the carousel and uses the I-INQUIRY command to detect when the cam on the carousel closes the microswitch. When the microswitch is closed, the carousel (**G** motor) is sent a STOP command, bringing the **G** error register to zero to stop the motor.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  A$="G+50" + CHR$(13)
40  GOSUB 170
50  A$="I" :  GOSUB 130
60  GOSUB 220
70  IF G=0 THEN 110
```

```
80  A$="G?" :  GOSUB 130
90  IF  W>45 THEN 50
100  GOTO 30
110  A$="GX" :  GOSUB 170
120  END
130  GOSUB 170
140  WAIT ST,R
150  W=PEEK(DA)-32
160  RETURN
170  FOR J=1 TO LEN(A$)
180        WAIT ST,S
190        POKE DA, ASC(MID$(A$,J,1))
200  NEXT
210  RETURN
220  C=0 : D=0 : E=0 : F=0 : G=0 : H=0
230  IF W>31 THEN H=1 : W=W-32
240  IF W>15 THEN G=1 : W=W-16
250  IF W>7 THEN F=1 : W=W-8
260  IF W>3 THEN E=1 : W=W-4
270  IF W>1 THEN D=1 : W=W-2
280  IF W>0 THEN C=1
290  RETURN
```

Line 30 lists a START command, adding 50 steps to the **G** motor error register to get it started.

Line 40 calls the subroutine to send the START command.

Line 50 sends the INQUIRY command to immediately begin checking the switches. (Line 50 also calls the subroutine to transmit the command and receive the answer.)

Line 60 calls the subroutine to interpret the answer to the INQUIRY.

Line 70 checks whether the **G** switch is closed (G=0). If it is closed, it sends the computer to line 110, a STOP command.

Line 80 checks the status of the **G** motor error register and calls the subroutine to send the command and receive the answer.

Line 90 sends the computer back to the INQUIRY command if there is not enough room in the **G** motor error register for another 50 steps.

Line 100 loops back to line 30 to add 50 more steps to the error register and keep the **G** motor moving.

Line 110 is the STOP command executed when the **G** switch is found. It brings the **G** motor error register to zero, stopping the motor.

The rest of the program lists the subroutines that we have already discussed.

Note that we use 50 step increments in our START command. But as discussed earlier, in more complicated programs it is better to start with 95 steps and fill the error register as full as possible each time, just as illustrated earlier.

A More Efficient Subroutine To Interpret Answers To An Inquiry Command.

The routine we have used to interpret answers to an INQUIRY worked as an illustration. But it is long and awkward. Now that you know how to interpret an answer, here is a shortcut.

The following subroutine would replace lines 220 through 290 in the sample program above.

```
200  FOR T=5 TO 0 STEP -1 : A(T)=0
210  IF W>((2 ^ T)-1) THEN A(T)=1 : W=W-2 ^ T
220  NEXT T
```

This subroutine does essentially what our old one did, but it does it in terms of an array. It assumes all switches are closed (=0) until proven otherwise.

As with the other subroutine, you must check the specific switch at completion of the the subroutine. But for this subroutine, you must address the switch in terms of the array, not by letter.

A(5) corresponds to the **H** switch
A(4) corresponds to the **G** switch
A(3) corresponds to the **F** switch
A(2) corresponds to the **E** switch
A(1) corresponds to the **D** switch
A(0) corresponds to the **C** switch

In the last sample program, to find the G switch and stop, line 70 would change to:

```
IF A(4)=0 THEN 110
```

Hard Home

Now you have seen how the INQUIRY command works to read the microswitches. You may want to use the INQUIRY for any number of reasons, but an important reason will be to find a repeatable starting robot position.

You may want to write a program to move the robot from point A to point B. The program won't work more than once unless the robot starts from the same point each time you run it. The way to establish such a starting point is to send each axis to the position where it actuates the microswitch and stop it there.

We call this switch defined robot position the **hardware home** position, or **hard home** for short. We've written a hard home program for your Apple. This program finds the microswitches for motors **D**, **E**, and **F** and stops each motor in this position. In fact, it finds the center of each microswitch--that is, it stops at the midpoint between the point at which the switch closes and the point at which it reopens. This precision is vital if you wish to repeat programs accurately.

You'll find the program in Chapter 10.

In practice, you should call the hard home routine once before each of your programming sessions begins.

Our hard home routine sets only three switches. You may wish to modify it to set the other three. If you do, remember that ports **A** and **B** of the controller are read with the J-INQUIRY command. Address them accordingly in your commands.

Note: The fingers and wrist flex axes have readily identifiable visual references. The fingers can be opened as far as they will go. The hand should be perpendicular to the work surface, fingers down and parallel with the body.

SUMMARY OF COMMANDS AND SUBROUTINES

OPEN COMMUNICATIONS

Your programs should always begin with two lines that open communications between your Apple and the controller.

Use these two lines for the **SuperSerial** card:

```
10  DA=49320 :  ST=DA+1:  CO=ST+1:  CL=CO+1:  S=16 :  R=24
20  POKE ST, 0 :  POKE CL, 190 :  POKE CO, 101
```

The START command starts a motor and moves it a specific distance and direction by adding the given value to that motor's error register.

Sample Apple command:

A$="D+50"+CHR$(13)

"D" addresses the motor
"+50" assigns direction (+) and distance (50 encoder holes)
CHR$(13) is the carriage return required after a START command

A$ is the variable used to carry the command through the necessary output subroutine.

The QUESTION command checks the error register for a motor to find out how much further it must travel before its error register is zero.

Sample Apple command:

A$="D?"

"D" designates the error register to be checked
"?" is the actual QUESTION command

A$ is the variable used to carry the command through the input/output subroutine.

The I-INQUIRY command asks whether the microswitches for motors C through H are open or closed. J-INQUIRY is similar.

Sample Apple command:

A$="I"

"I" is the actual INQUIRY command
A$ is the variable used to carry the command through the necessary

input/output subroutine.

The STOP command stops a motor by bringing its error register to zero

Sample Apple command:

A$="DX"

"D" designates the motor
"X" is the actual STOP command

A$ is the variable used to carry the command through the necessary output subroutine

SUBROUTINES

Output only - call immediately after START and STOP commands to transmit them to the controller. (A$ contains the command string).

```
500 FOR J=1 TO LEN(A$)
510    WAIT ST, S
520    POKE DA, ASC(MID$(A$,J,1))
530 NEXT
540 RETURN
```

Output/input - call after QUESTION or INQUIRY commands to transmit the commands to the controller and receive the answer

```
400 GOSUB (output subroutine above)
410 WAIT ST,R
420 W=PEEK(DA)-32
430 RETURN
```

Subroutine to interpret answers to INQUIRY commands - call after INQUIRY has been sent and output/input subroutine has been called

```
100 FOR T=5 TO 0 STEP -1: A(T)=0
110    IF W>(2 ^ T)-1 THEN A(T)=1:W=W-2 ^ T
120 NEXT T
```

130 RETURN

At completion of the subroutine, the computer returns to the program. To check any one switch, you must address the switch in terms of an array, as follows:

```
A(5)=H switch
A(4)=G switch
A(3)=F switch
A(2)=E switch
A(1)=D switch
A(0)=C switch
```

Therefore, if you wanted to check the **D** switch, the statement would look like this:

```
IF A(1)=0 THEN XXX
```

where XXX is the line you want executed if the switch is closed.

The subroutine sets all switches closed (=0) until otherwise proven. An open switch equals one, a closed switch equals zero.

USING THE INPUT/OUTPUT COMMANDS

The MARK III controller has eight TTL inputs, eight TTL outputs and two unencoded motor power port called the AUX ports. Eight commands are available to completely control these features. The use of these commands follow closely the format already developed to control the motor movements; in fact they use the same subroutines.

The MARK III inputs and outputs are TTL compatable and should be treated as low current (less than 15ma at 5 VDC) devices. Do not apply an external voltages to these ports, use the 5 volt source provided on the I/O front panel. The inputs are internally pulled up by 10k ohm resistors and therefore appear to be ON when left unconnected. To control the inputs, a switch should be connected to the input port and the ground terminal. When the switch is closed the input will be OFF; with the switch open the input will be ON.

The outputs are active LOW which means that when the port is ON the output will be LOW and when the port is OFF the output will be HIGH.

The following pages will give examples on how to implement each of these commands.

J-Inquiry Command. Read Inputs 1 through 4

The **J** command, like the **I** command, returns a byte in which the first four bits represent the state of inputs 1 through 4 as follows:

Bit	Input
0	1
1	2
2	3
3	4

The same type of routine that was used with the **I** command can be used to set an arry variable to a 1 or a 0 dependent on the state of the input. Here we will use the variable **I** to represent the inputs, with I(0)=Input 1, I(1)=Input 2 etc.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  A$="G+50" + CHR$(13)
40  GOSUB 170
50  A$="J" :  GOSUB 130
60  GOSUB 220
70  IF I(1)=1 THEN 110
80  A$="G?" :  GOSUB 130
90  IF  W>45 THEN 50
100  GOTO 30
110  A$="GX" :  GOSUB 170
120  END
130  GOSUB 170
140  WAIT ST,R
150  W=PEEK(DA)-32
160  RETURN
170  FOR J=1 TO LEN(A$)
180        WAIT ST,S
190        POKE DA, ASC(MID$(A$,J,1))
200  NEXT
210  RETURN
220  FOR T=3 TO 0 STEP -1 : I(T)=0
230        IF W>((2 ^ T)-1) THEN I(T)=1 : W=W-2 ^ T
240  NEXT T
```

```
250  RETURN
```

This program will move the **G** axis motor, a belt conveyor for instance, until INPUT #2 is turned ON (high TTL level). In an actual workcell, a sensor which goes HIGH when a part is detected would be connected to the input port; when the part arrives at the sensor position, the conveyor will stop.

Lines 220 through 250 implement the routine that determines which inputs are ON by checking each of the first four bits in the byte returned by the controller in response to the **J** command. Line 70 checks if the input is ON; if it is the STOP command is executed and the **G** port is turned off.

K-Inquiry command. Read inputs 5 through 8.

The implementation of the **K** command is identical to the **J** command except that the returned byte contains the state of inputs 5 through 8 in the low four bits. The routine that determines the state of each input is identical to the example program above; instead of using a J in line 50 use a K.

Commands L, M, N, O. Sets the state of the AUX ports.

The two AUX ports on the MARK III controller provide + or - 20VDC motor power, the polarity of which is determined by the setting of the reversing switches near the AUX ports. The four commands (L, M, N, O) set the AUX ports on or off, they do not set the polarity of the output. These commands are simple one byte commands that do not invoke a response from the MARK III controller. The implementation of all four commands is identical, therefore the following example will show how to turn on AUX#2 and then turn it off.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  A$="N"
40  GOSUB 170
50  FOR J=1 TO 1000: NEXT
60  A$="O"
70  GOSUB 170
80  END
90  FOR J=1 TO LEN(A$)
100        WAIT ST,S
110        POKE DA, ASC(MID$(A$,J,1))
```

```
120  NEXT
130  RETURN
```

Lines 30 and 40 issue the **N** command which will turn on AUX port #2.

Line 50 implements a time delay to allow the user to see the effect of the AUX commands.

Lines 60 and 70 issue the **O** command which will turn off AUX port #2.

The MARK III controller has two indicator lights connected to the AUX ports to indicate if they are ON. Even of you have nothing connected to the AUX ports, this program can be run and verified since it will cause the light on AUX #2 to come on then go off.

P and R Commands. Set Output Lines High (OFF) and Low (ON)

Eight TTL level outputs are available on the MARK III controller, each of which can be set or reset upon command. The **P** command instructs the controller to turn OFF an output (make the output go high); the **R** command instructs the controller to turn ON an output (make the output go low). The format of both commands is the letter of the command followed by the port number that is to be changed.

Suppose it is desired to initialize output #3 by turning it off and then to turn on output #3 when the robot comes to rest to indicate that the material handling operation has been completed. The following program will initialize move the waist 240 counts and then turn ON the output.

```
10 DA=49320 :  ST=DA+1 :  CO=ST+1 :  CL=CO+1 :  S=16 :  R=24
20 POKE ST,0 :  POKE CL,190 :  POKE CO,101
30  A$="P3":   GOSUB 210
40  N=240
50  H=95
60  IF N<=H THEN 110
70  A$="F+" + STR$(H) + CHR$(13)
80  GOSUB 210 :  N=N-H
90  A$="F?": GOSUB 170 :  H=95-W
100  GOTO 60
110  A$="F+" + STR$(N) + CHR$(13)
120  GOSUB 210
130  A$="F?:  GOSUB 170
140  IF W>0 THEN 130
```

```
150  A$="R3":  GOSUB 210
160  END
170  GOSUB 210
180  WAIT ST,R
190  W=PEEK(DA)-32
200  RETURN
210  FOR J=1 TO LEN(A$)
220        WAIT ST,S
230        POKE DA, ASC (MID$(A$,J,1))
240  NEXT
250  RETURN
```

This program is familiar to you by now since it implements a long move as discussed previously. However, line 30 directs the controller to turn off output #3 before starting the programmed motor move. Lines 40 through 120 implements the long move. Upon completion of the move, line 130 outputs the command to turn on port 3.

CHAPTER SIX

RUNNING THE XR WITH AN IBM-PC

Interfacing with the IBM-PC

You need an IBM Asynchronous Serial Communications Card and an RS-232C cable with an IBM Adapter Plug for proper connection between your IBM-PC and the Mark III controller. RHINO provides the cable and the adapter plug with every Mark III Controller. The customer provides the serial card which can be installed in any of the expansion slots of the IBM PC; configure the card for COM1. Install the data cable as follows:

1. Be sure that the main power and motor power switches on the controller are **off**, and that the IBM-PC power is **off**;

2. Plug the female connector of the IBM adapter plug into the RS-232C port of your IBM-PC. Plug the data cable into the other end of the IBM adapter.

3. Connect the data cable into the port on the front of the controller labeled **"Computer"**.

Setting Up Communications

The following steps need to be taken to set up communications between your IBM-PC and the Mark III Controller

1. Insert your IBM system DOS master disk into the left drive and turn on the PC.

2. Place the Mode Switch on the front panel of the MARK III controller to the COMPUTER (down) position and turn on the **main power** on the controller.

3. Turn on the **motor power** on the controller.

4. Press the **RESET** button on the Mark III controller.

5. Load **BASICA** after the DOS disk is booted.

If you have to stop the arm because it is going to strike an obstacle, or for any emergency, press the reset button on the Mark III Controller. With this in mind, let's go on.

Whenever you program with your IBM-PC, you must start by setting up communications between your computer and the controller. Start by typing in the following line.

```
10 OPEN "COM1:9600,E,7,2,CS,DS,CD" AS #1
```

This line opens the communications port number 1 as file number 1 and sets the BAUD rate and data format. It also disables time out errors.

Move The Motors With The START Command

Now, to make your XR move. Type in:

```
20 PRINT #1,"F+50"
30 END
```

Watch your XR as you run the program. Your XR will move 50 encoder holes on it's waist axis and stop.

You used one of the four fundamental commands you can send to the controller; the START command.

The "**F**" designated the motor.

The "**+50**" designated the direction (+) and the distance (50 encoder holes).

Now change the sign in line 20 so it looks like this:

```
20 PRINT #1,"F-50"
```

By placing a - sign before the 50, you send the motor in the opposite direction. We included the + sign in the first example as an illustration, but absence of a sign is interpreted as a +. Run the program again and notice that the XR now moves in the opposite direction before stopping.

Now change line 20 so it looks like this:

```
20 PRINT #1,"E+50"
```

Run the program and observe the **E** motor (shoulder) movement.

Change line 20, one motor at a time, to move the **D**, **C**, and **B** motors. Don't use the **A** motor for now because of its limited travel. Just change the letter to that corresponding to the motor you want to move.

For now, do not use any number other than 50 for the number of steps.

More About The START Command--The Error Registers

Now that you have used the basic START command, you are ready to learn more about it.

Consider our first START command "**F+50**" as an example. When we sent that command (using the PRINT statement), the motor "F" moved. But much more than that happened.

The MARK III controller maintains an 8 bit error register for each of the 8 motors. When the error register for a specific motor is zero, the controller removes all power to the motor and the motor remains stationary. When the error register is non-zero, the controller connects either a positive voltage or a negative voltage to the motor. The polarity of the power is a function of what is in the error register. If the error value is positive, the motor moves in one direction and if it is negative, the motor moves in the other direction. Before we sent a command, the error registers for all the motors were zero.

A START command adds the motor movement value to the error register for the motor specified. In our example, that motor was the F motor. The value added was 50.

As soon as a value is added to the error register for a motor, the controller starts that motor in the direction that will take the error register to zero. As the motor moves, the encoders are read and the count in the error register is decremented. When the error register reaches zero, the power is turned off. So, when our START command added 50 to the F motor register, the controller moved that motor 50 encoder steps in the right direction to return the error register to zero.

The command "F-50" adds a value of -50 to the F motor error register. When the controller receives a "-" sign, it moves the motor in the opposite direction from the + direction to reach the zero error position.

Remember:

1. A START command adds a value to the designated motor's error register.

2. The value added to the error register represents the number of steps the designated motor encoder must move to return the register to zero.

3. The controller software is designed so that motors seek the zero error condition.

4. The sign in the START command designates direction of motion.

With this in mind, move on to the next command, the QUESTION command.

THE QUESTION COMMAND

Type in the program listed below.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"F+50"
30  PRINT #1,"F?";: GOSUB 90
40  IF W<>0 THEN 30
50  PRINT #1,"F-50"
60  PRINT #1,"F?";: GOSUB 90
70  IF W<>0 THEN 60
80  END
90  IF LOC(1)=0 THEN 90 ELSE W$=INPUT$(LOC(1),#1)
100 W=ASC(W$)-32
110 RETURN
```

Run the program and you will see that it simply turns the waist in one direction and then In the other. You might be wondering why we did not simply follow line 20

```
20 PRINT #1,"F+50"       with
50 PRINT #1,"F-50"
```

Instead, we send a QUESTION command on line 30. Why?

Remember that a START command adds a value to a motor's error register. If we sent an "F-50" immediately after the "F+50", the -50 count in the second command would combine with the remains of the +50 command and sent the F motor in the opposite direction before it completed the full "+50" steps in the first START command. The QUESTION command allows us to check the error register and make sure the first 50 steps have been executed before the next series of (-50) steps are sent.

How The QUESTION Command Works

The QUESTION command instructs the MARK III controller to return the current value of the error register for a specific motor. Specifically, it asks how much further the designated motor must travel until the register is again zero. In our sample program, after we sent the first START command, we immediately sent a QUESTION to check the status of the **F** error register.

We do so on line 30. Again "F" designates the motor. The QUESTION command is accomplished with the "**?**". Line 30 also calls a subroutine to allow your computer to receive the answer to the QUESTION. That subroutine, which begins on line 90, is written so you can use it any time you use a QUESTION or any inquiry command in your programs.

Line 40 loops through the QUESTION until the answer (W) is zero. When it is zero, we know that the error register is zero and all the original steps have been executed. Only then does the program move on to send the START command in the opposite direction.

More About The Question Command
Making A Longer Move

Thus far, our sample programs have limited START commands to 50 steps. The reason has to do with the error registers and communications. Two factors create problems.

1. The largest signed decimal number we can store in an 8 bit error register is 127 (1 bit for the sign of the number, and seven bits to represent up to 127). Numbers larger than 127 will overflow the register. When this happens, the register overflows into the sign bit. This reverses the sign in the error register and thus reverses the motor. In other words, the motor can start to move in the opposite direction to that specified in the START command if the register overflows.

2. When the controller receives a QUESTION command, it automatically adds

32 (hexadecimal 20) to the answer before it transmits it to the host computer. It does so because some computers treat values below decimal 32 (hex 20) as commands instead of data. Note that in subroutine 90, line 100 subtracts 32 from the answer received in response to the QUESTION command. This compensates for the 32 added by the controller. Since the communication line is set for 7 data bits, the largest value that can be returned is 127 which means that the largest value the controller can send is 127-32 or 95.

A motor's error register can be zero, but it can never be lower (because we use the sign bit for direction of travel of the motor). In other words, you can find how far a motor is from the zero position but you cannot find out what direction the zero position is in. The error count is always positive. By adding 32 to every value it transmits to the host computer, the controller ensures that the computer never receives an answer less than 32 (hex 20).

EXAMPLE - Suppose you send a START command of 120 steps. The motor starts to seek zero immediately. If the controller receives a QUESTION when the motor has traveled 20 steps, the answer sent back (the number of steps until zero) will be 100. But the controller will add 32 to the 100 before it sends the answer back to the host. 132 is more than you can represent with 7 bits so the communication line will truncate the 132 to 7 bits and an error will occur.

All this may seem puzzling to you at first, however you will begin to understand the system as you work with it. If you are confused, just keep this rule in mind:

DON'T LET THE ERROR REGISTER EXCEED 95

"But," you may be asking, "how do I get the XR to execute a move longer than 95 steps?"

The answer is to first add a value to a motor's error register with the START command. Monitor the status of the error register with the QUESTION command. Then, add to the register with another START before the register reaches zero, but do not add so much as to exceed the 95 limit.

The following program illustrates one way to make a move of 500 steps.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  FOR I = 1 TO 10
30      PRINT #1,"F+50"
40      PRINT #1,"F?";: GOSUB 90
50      IF W>45 THEN 40
```

```
60 NEXT
70 END
80 IF LOC(1)=0 THEN 80 ELSE W$=INPUT$(LOC(1),#1)
90 W=ASC(W$)-32
100 RETURN
```

This program uses two key techniques to accomplish the 500 step move.

1. The FOR/NEXT loop that begins on line 20 divides and conquers by adding 500 steps to the **F** motor error register, 50 steps at a time. It loops the computer through line 60 ten times, then exits the loop.

2. The IF / THEN statement on line 50 loops the computer through the QUESTION command on line 40 until the answer is 45 or less. When the answer is 45 or less it is safe to continue through the FOR loop and add the next 50 steps without exceeding the 95 limit.

Remember that in our simple program to rotate the **F** motor back and forth, the program looped until all of the first 50 steps were executed. This was necessary because the second move was in the opposite direction, and would have reversed the motor before it completed the original 50 steps.

But for the long move above, we do not have to wait for the error register to reach zero before adding to it. We want to avoid letting the error register reach zero; because if it does, the motor will stop. And we want to keep adding to the register without exceeding 95.

You might think of an error register as a gas tank. You never want it to get to empty, but you can't let it overflow, either.

The program above showed one way to do this. It added 50 steps to the register, and as soon as there was room for another 50, it added still another 50.

The program worked as an illustration, but it has drawbacks. The most important drawback is that it waits for the register to come all the way down to 45 before it adds more steps. This was not a problem here, but as you program more complex moves, and connect accessory motors to the controller, the computer will be busy, and timing will become more critical. You will have less time to check and fill the registers.

A better way is to always fill the register as full as possible. Instead of deciding beforehand when and how many steps to add, let the computer calculate how many steps it can add without exceeding 95, and automatically add that many steps. The following program does just that.

```
10 OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20 N=500
30 H=95
40 IF N≤H THEN 80
50 PRINT #1,"F+"; STR$(H): N=N-H
60 PRINT #1,"F?";:  GOSUB 90: H=95-W
70 GOTO 40
80 PRINT #1,"F+";STR$(N): END
90 IF LOC(1)=0 THEN 90 ELSE W$=INPUT$(LOC(1),#1)
100 W=ASC(W$)-32
110 RETURN
```

In line 20, N represents the total number of steps we want the motor to move. We used 500, but you could change N to any number of steps.

We use H as the variable for the number of steps we add to the error register. In line 30, we assign H an initial value of 95; the maximum number we can safely add to the error register.

In line 50, we send a START command that is slightly different than the ones we have used up to now. Instead of specifying the number of steps, we use the variable H. Line 50 adds H number of steps to the **F** motor error register. Since we have assigned H an initial value of 95, the computer will add 95 the first time it executes line 50. Line 50 also subtracts H from N, it decrements our original total steps by the number added to the error register.

Line 60 sends a QUESTION command and calls the subroutine to send the command and receive the answer (W).

Line 60 also assigns H a new value, 95-W. Since W represents the number of steps the **F** motor must travel before the error register is zero, 95-W is the number of steps we can add to the error register without exceeding 95.

Therefore, H now becomes whatever number required to bring the error register to 95.

Line 70 loops back to line 40. Line 40 checks whether N is less than or equal to H. If N is not less than or equal to H, the computer goes to line 50 to add H more steps to the error register to keep the motor moving.

The program will repeat this loop until N≤H. Remember, line 50 subtracts H from N each time H has been added to the error register. So N, our original number of steps, gets smaller each time through the loop.

When N≤H, the computer goes to line 80, another START command. This START command, however, adds N steps to the error register instead of H steps. At this point, sending N steps completes the original N=500 steps.

As you can see, this method keeps the error register as full as possible. When you write programs to run several motors, this will be critical. Since the program will have to check more than one error register (by means of the QUESTION command), it is important to fill the error register completely each time the QUESTION command is sent. Otherwise, the error register may run to zero (the motor stops) by the time the computer gets back around to check.

The STOP And INQUIRY Commands

In this section we introduce you to the STOP and INQUIRY commands. We'll talk about each separately and then provide a sample program that uses both.

The STOP Command

When you use a START command, a motor will travel the assigned number of steps and stop. But as you develop programs of your own, you'll need to stop motors under specified conditions.

For example, many applications present the possibility of running the robot arm into an object. If the robot crashes into an obstacle, you need a way of recovering from the accident without losing your program and positional information. A servo motor will try to keep moving as long as it has power (ie., as long as its error register is non-zero). The STOP command enables us to bring a specific error register to zero on command.

A STOP command looks like this for the IBM PC:

```
PRINT #1,"FX";
```

"F" designates the motor

"X" is the STOP command

With the STOP command, your programs can incorporate protections against stalls. You can use the QUESTION command to check the status of an error register. The program would send the controller a STOP command in the event that the (a non-zero) error register status did not change during a number of successive checks.

Another use of the STOP command has to do with the microswitches mounted on the XR. The next section, which covers the INQUIRY command, will illustrate the use of the STOP command with the INQUIRY command.

The INQUIRY Command. Reading The Microswitches

The XR is equipped with six microswitches, one for each of the six motors on the arm. Each axis closes its switch at a defined point. Therefore, we have an identifiable and repeatable orientation point for each axis on the XR; the point at which the microswitch is closed.

The INQUIRY commands (**I** and **J**) gives us the means to determine whether any of the switches on the robot arm are closed at a given time. The MARK III controller can read switches on all its ports. So, as your XR is connected now, the controller reads the switches as follows:

Motor	**Port**	**Joint**	**Can controller read switch?**	**Command Used**
A	A	Fingers	Yes	J
B	B	Wrist	Yes	J
C	C	Wrist flex	Yes	I
D	D	Elbow	Yes	I
E	E	Shoulder	Yes	I
F	F	Waist	Yes	I
G	G	Accessory	Yes	I
H	H	Accessory	Yes	I

In the standard configuration, with motors **A-F** connected to ports **A-F** on the controller, the **I**-INQUIRY command reads motors **C-F** on the XR-3 and **G** and **H** on the accessories.

In order to read the **A** and **B** port switches, it is necessary to use the **J**-INQUIRY command.

Using the I and J-INQUIRY Commands

The INQUIRY commands work much like the QUESTION command. Each requires the same subroutine to send the command and to receive the answer. The controller adds 32 to the answer before it transmits it to your computer to avoid ASCII command codes, and the receiving subroutine subtracts 32 to yield an accurate answer.

But instead of designating individual motors when you send the INQUIRY, you send the command and interpret the answer according to the port you wish to read. The command looks like this:

```
PRINT #1,"I";          (or "J")
GOSUB XXX
```

where XXX is the number of the same subroutine you've used for the QUESTION command. The subroutine enables the computer to receive the answer to an INQUIRY.

Interpreting The Answers

We will discuss the I-INQUIRY command only. The J-INQUIRY is similar but gives different information.

The key to using the INQUIRY command is interpreting the answer sent back from the controller. Remember that the controller communicates using one byte only, and that the single byte answer uses one bit in the byte for each switch.

The bits are assigned to the ports as follows:

Port	Bit		
C	0		
D	1		
E	2		
F	3		
G	4		
H	5		
-	6	not used	
-	7	not used	MSB

A microswitch can either be open or be closed. When a switch is open, the corresponding bit contains a one. When all switches are open, all six bits contain a one, which has a decimal value of 63. (In fact, the controller sends a 95. Remember that the controller adds 32 to anything it sends to your computer to avoid command codes. Our subroutine to receive the answer automatically subtracts 32 to yield the true value.)

If a switch is closed, the corresponding bit contains a 0, and the answer to an I-INQUIRY will be 63 minus a value unique to the open switch and bit. The table below illustrates what happens when one switch is closed and all others are open.

IF THE SWITCH AT PORT X IS CLOSED, AND ALL OTHERS ARE OPEN	THE ANSWER RETURNED IN RESPONSE TO AN INQUIRY BECOMES. (Ignoring the 32 count offset)	
If X=C	62	(63-1)
If X=D	61	(63-2)
If X=E	59	(63-4)
If X=F	55	(63-8)
If X=G	47	(63-16)
If X=H	31	(63-32)

As you can see, when a switch is closed, a value unique to that switch is subtracted from 63. If we could be sure that only one switch is closed at any one time, things would be simple. We could just send the I-INQUIRY command and check the answer to determine if the switch we were checking for was closed.

The problem is, at any one time, more that one switch may be closed. The answer received will be 63 minus the sum of the values for all closed switches. For example, if switches **C**, **D**, and **E** were closed when we sent the INQUIRY, the answer returned would be 56 ((63-(1+2+4)).

To check for any one switch, we must mask the answer for the other switches that might be closed.

The following subroutine does just that. We would call this subroutine after sending an INQUIRY and receiving the answer. The variable W represents that answer, and we are checking whether the **D** microswitch is closed.

```
100 C=0: D=0: E=0: F=0: G=0: H=0
110 IF W>31 THEN H=1: W=W-32
120 IF W>15 THEN G=1: W=W-16
130 IF W>7 THEN F=1: W=W-8
140 IF W>3 THEN E=1: W=W-4
150 IF W>1 THEN D=1: W=W-2
160 IF W>0 THEN C=1
```

170 IF D=1 THEN XXX (Go to desired point in the program).

Let's run through the subroutine with an actual value to see how it works. Assume that the **C** and **D** switches are closed, but that we only care about the D motor. Remember that if all switches are open, the answer returned in response to an **I**-INQUIRY command is 63.

Remember also that if a switch is closed, the answer returned will be 63 minus a unique value for that switch:

SWITCH CLOSED	CONTROLLER SUBTRACTS
C	1
D	2
E	4
F	8
G	16
H	32

Given this, we know that if switches **C** and **D** are closed, the answer returned in response to the INQUIRY command will be 60.

Let's see how our subroutine determines whether the **D** switch is closed.

Line 100 assigns a variable to each switch. It simply uses the letters assigned to each port. It assigns the value zero to each variable to initialize it to a known quantity. As discussed above, zero corresponds to an open switch.

Line 110 checks to see whether the **H** switch is closed. We know that when W>31 the **H** switch is open. We know this because even if the **H** switch were closed and all others open, W would be at least 31. Therefore, if W>31, the H switch must be open. Since W is sixty, the **H** switch is open, and we can now set H=1 to indicate that it is open.

If **H** is open, we know that 32 is part of W. Before checking the **G** switch, we must mask the 32. That's why before moving to line 120, we set W=W-32. So now W=28.

Line 120 checks the **G** switch. We know that if W>15, the **G** switch must be open because even if all other switches were open (remember we've masked the **H** switch),W would be only 15. Now **G**=1 and we again mask because we know it is open. Now W = 28-16 = 12.

Understand how the routine checks for each switch and masks for its value before reading further. Run through line 130 with 12, then take the new W and run through 140.

After line 130 and line 140, W should equal 0. The next line, 150, checks for the **D** switch. Let's see what happens when a switch is closed.

```
        150 IF W>1 THEN D=1:W=W-2
```

Since W now is 0, and 1 is not greater than 1, D remains zero. W remains 0, as there is no value to mask.

Line 160 checks for the only remaining switch, the **C** switch.

```
        160 IF W>0 THEN C=1
```

Since W is still zero, and zero is not greater than zero, **C** remains zero. Since all switches have been checked, the subroutine ends here.

With the subroutine complete the variables are now as follows:

C=0	(C switch is closed)
D=0	(D switch is closed)
E=1	(E switch is open)
F=1	(F switch is open)
G=1	(G switch is open)
H=1	(H switch is open)

Since we are checking for the **D** switch, we have to use an IF... THEN ... statement to make the results of the subroutine useful. For example:

```
        IF D=0 THEN XXX
```

where XXX represents the line we want to execute if the D switch is closed.

The STOP and INQUIRY Commands. A Sample Program

The program below supposes that an accessory, the carousel, is connected to the **G** port on the controller. It runs the carousel and uses the INQUIRY command to detect when the cam on the carousel closes the microswitch. When the microswitch is closed, the carousel (**G** motor) is sent a STOP command, bringing the **G** error register to zero to stop the motor.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"G+50"
30  PRINT #1,"I";: GOSUB 110
40  GOSUB 140
50  IF G=0 THEN 90
60  PRINT #1,"G?";: GOSUB 110
70  IF W>45 THEN 30
80  GOTO 20
90  PRINT #1,"GX";
100  END
110 IF LOC(1)=0 THEN 110 ELSE W$=INPUT$(LOC(1),#1)
120 W=ASC(W$)-32
130 RETURN
140 C=0: D=0: E=0: F=0: G=0: H=0
150 IF W>31 THEN H=1: W=W-32
160 IF W>15 THEN G=1: W=W-16
170 IF W>7 THEN F=1: W=W-8
180 IF W>3 THEN E=1: W=W-4
190 IF W>1 THEN D=1: W=W-2
200 IF W>0 THEN C=1
210 RETURN
```

Line 20 lists a START command, adding 50 steps to the **G** motor error register to get it started.

Line 30 sends the INQUIRY command to immediately begin checking the switches. (Line 30 also calls the subroutine to transmit the command and receive the answer.)

Line 40 calls the subroutine to interpret the answer to the INQUIRY.

Line 50 checks whether the **G** switch is closed (G=0). If it is closed, it sends the computer to line 90, a STOP command.

Line 60 checks the status of the **G** motor error register and calls the subroutine to send the command and receive the answer.

Line 70 sends the computer back to the INQUIRY command if there is not enough room in the **G** motor error register for another 50 steps.

Line 80 loops back to line 20 to add 50 more steps to the error register and keep the **G** motor moving.

Line 90 is the STOP command executed when the **G** switch is found. It brings the **G** motor error register to zero, stopping the motor.

The rest of the program lists the subroutines that we have already discussed.

Note that we use 50-step increments in our START command. But as discussed earlier, in more complicated programs it is better to start with 95 steps and fill the error register as full as possible each time, just as illustrated earlier.

A More Efficient Subroutine To Interpret Answers To An Inquiry Command

The routine we have used to interpret answers to an INQUIRY worked as an illustration. But it is long and awkward. Now that you know how to interpret an answer, here is a shortcut.

It uses the AND operator, which is part of IBM BASICA. It allows you to check specifically for the bit corresponding to the switch you are checking.

To show you just how much easier using the AND operator is, the line below would replace the entire subroutine--lines 140 through 200 in the sample program just given.

```
IF W AND 16 THEN XXX ELSE XXX
```

The AND operator is based on principles that require a lengthy explanation. You can find more information in your IBM BASIC manual.

But to use the AND operator, you really only need to know the following:

1. The integer values that correspond to each switch when it is open are:

SWITCH	INTEGER
C	1
D	2
E	4
F	8
G	16
H	32

2. Any time you use the AND operator to check the status of a single microswitch, just substitute the integer value for that switch for the 16. For example, if you want to check the **H** switch, the line should look like this:

IF W AND 32 THEN H=1 ELSE H=0

When the microswitch checked is closed, the computer executes the command listed after ELSE (ie. H=0) and when the microswitch checked is open, the computer executes the program line listed after THEN (ie. H=1).

Hard Home

Now you have seen how the INQUIRY command works to read the microswitches. You may want to use the INQUIRY for any number of reasons, but an important reason will be to find a repeatable starting robot position.

You may want to write a program to move the robot from point A to point B. That program won't work more than once unless the robot starts from the same point each time you run it. The way to establish such a starting point is to send each axis to the position where it actuates the microswitch and stop it there.

We call this switch defined robot position the **hardware home** position, or **hard home** for short. We've written a hard home program for your IBM-PC. This program finds the microswitches for motors **D**, **E**, and **F** and stops each motor in this position. In fact, it finds the center of each microswitch, that is, it stops at the midpoint between the point at which the switch closes and the point at which it reopens. This precision is vital if you wish to repeat programs accurately.

You'll find the program in the Appendices.

In practice, you should call the hard home routine once before each of your programming sessions begins.

Our hard home routine sets only three switches. You may wish to modify it to set the other three. If you do, remember that ports **A** and **B** of the controller are read with the **J**-INQUIRY command. Address them accordingly in your commands.

Note: The fingers and wrist flex, there are readily identifiable visual references. The fingers can be opened as far as they will go. The hand should be perpendicular to the work surface, fingers down and parallel with the body.

SUMMARY OF COMMANDS AND SUBROUTINES

The OPEN COMMUNICATIONS line should begin each program. It opens a communications file at 9600 BAUD, 7 data bits, even parity, 2 stop bits, and disables time out errors.

```
OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
```

The START command moves a specified motor a specified distance and direction by adding a value to the motor's error register.

Example:

```
PRINT #1,"F+50"
```

PRINT #1 sends the command through communications file #1

"F" designates the motor

"+50" designates the number of encoder holes (distance) to be moved and direction (+) to be moved. A - sign before the 50 would reverse the motor 50 steps.

The QUESTION command asks how much further a motor must travel until its error register is zero.

Example:

```
PRINT #1,"F?"
```

PRINT #1 sends the command through communications file #1

"F" designates the motor being checked

"?" is the QUESTION command

The QUESTION must always be followed by the input subroutine that allows the computer to receive the answer.

The STOP command brings a designated motor's error register to zero, stopping the motor.

Example:

PRINT #1,"FX"

PRINT #1 sends the command through communications file #1

"F" designates the motor

"X" is the actual STOP command

The INQUIRY command asks whether the microswitches are opened or closed.

Example:

PRINT #1,"I";

PRINT #1 sends the command through communications file #1

"I" is the command

An INQUIRY command must always be followed by the input subroutine that allow the computer to receive the answer. In addition, a subroutine to interpret the answer must follow the input subroutine. The semi-colon should follow the PRINT statement to suppress the sending of a carriage return which is interpreted by the MARK III controller as a motor move command.

Output/Input subroutine--use after every QUESTION and INQUIRY command.

```
nnn  IF LOC(1)=0 THEN nnn ELSE W$=INPUT$(LOC(1),#1)
     W=ASC(W$)-32
     RETURN
```

Subroutine to interpret answer to an INQUIRY command--call after command is sent and Input subroutine is called.

IF W AND N THEN XXX ELSE XXX

W is the answer received in response to an INQUIRY command

N is the integer value corresponding to the bit of the switch to be checked

THEN XXX will be executed if the switch is closed. XXX indicates the line to be executed.

ELSE YYY will be executed if the switch is OPEN. YYY indicates the line to be executed.

USING THE INPUT/OUTPUT COMMANDS

The MARK III controller has eight TTL inputs, eight TTL outputs and two unencoded motor power port called the AUX ports. Eight commands are available to completely control these features. The use of these commands follow closely the format already developed to control the motor movements; in fact they use the same subroutines.

The MARK III inputs and outputs are TTL compatable and should be treated as low current (less than 15ma at 5 VDC) devices. Do not apply an external voltages to these ports, use the 5 volt source provided on the I/O front panel. The inputs are internally pulled up by 10k ohm resistors and therefore appear to be ON when left unconnected. To control the inputs, a switch should be connected to the input port and the ground terminal. When the switch is closed the input will be OFF; with the switch open the input will be ON.

The outputs are active LOW which means that when the port is ON the output will be LOW and when the port is OFF the output will be HIGH.

The following pages will give examples on how to implement each of these commands.

J-Inquiry Command. Read Inputs 1 through 4

The **J** command, like the **I** command, returns a byte in which the first four bits represent the state of inputs 1 through 4 as follows:

Bit	Input
0	1
1	2
2	3
3	4

The same type of routine that was used with the **I** command can be used to set an arry variable to a 1 or a 0 dependent on the state of the input. Here we will use the variable **I** to represent the inputs, with I(0)=Input 1, I(1)=Input 2 etc.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"G+50"
30  PRINT #1,"J";:  GOSUB 110
40  GOSUB 140
50  IF I(1)=1 THEN 90
60  PRINT #1,"G?";:  GOSUB 110
70  IF  W>45 THEN 30
80  GOTO 20
90  PRINT #1,"GX";
100  END
110  IF LOC(1)=0 THEN 110 ELSE W$=INPUT$(LOC(1),#1)
120  W=ASC(W$)-32
130  RETURN
140  FOR T=0 TO 3
150        IF W AND (2 ^ T) THEN I(T)=1 ELSE I(T)=0
160  NEXT T
170  RETURN
```

This program will move the **G** axis motor, a belt conveyor for instance, until INPUT #2 is turned ON (high TTL level). In an actual workcell, a sensor which goes HIGH when a part is detected would be connected to the input port; when the part arrives at the sensor position, the conveyor will stop.

Lines 140 through 170 implement the routine that determines which inputs are ON by checking each of the first four bits in the byte returned by the controller in response to the **J** command. Line 50 checks if the input is ON; if it is, the STOP command is executed and the **G** port is turned off.

K-Inquiry command. Read inputs 5 through 8.

The implementation of the **K** command is identical to the **J** command except that the returned byte contains the state of inputs 5 through 8 in the low four bits. The routine that determines the state of each input is identical to the example program above; instead of using a J in line 50 use a K.

Commands L, M, N, O. Sets the state of the AUX ports.

The two AUX ports on the MARK III controller provide + or - 20VDC motor power, the polarity of which is determined by the setting of the reversing switches near the AUX ports. The four commands (L, M, N, O) set the AUX ports on or off, they do not set the polarity of the output. These commands are simple one byte commands that do not invoke a response from the MARK III controller. The implementation of all four commands is identical, therefore the following example will show how to turn on AUX#2 and then turn it off.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"N";
30  FOR J=1 TO 1000: NEXT
40  PRINT #1,"O";
50  END
```

Line 20 issues the **N** command which will turn on AUX port #2.

Line 30 implements a time delay to allow the user to see the effect of the AUX commands.

Line 40 issues the **O** command which will turn off AUX port #2.

The MARK III controller has two indicator lights connected to the AUX ports to indicate if they are ON. Even of you have nothing connected to the AUX ports, this program can be run and verified since it will cause the light on AUX #2 to come on then go off.

P and R Commands. Set Output Lines High (OFF) and Low (ON)

Eight TTL level outputs are available on the MARK III controller, each of which can be set or reset upon command. The **P** command instructs the controller to turn OFF an output (make the output go high); the **R** command instructs the controller to turn ON an output (make the output go low). The format of both commands is the letter of the command followed by the port number that is to be changed.

Suppose it is desired to initialize output #3 by turning it off and then to turn on output #3 when the robot comes to rest to indicate that the material handling operation has been completed. The following program will initialize move the waist 240 counts and then turn ON the output.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"P3";
30  N=240
40  H=95
50  IF N<=H THEN 90
60  PRINT #1,"F+" + STR$(H):  N=N-H
70  PRINT #1,"F?": GOSUB 140:  H=95-W
80  GOTO 50
90  PRINT #1,"F+" + STR$(N)
100  PRINT #1,"F?": GOSUB 140
110  IF W>0 THEN 100
120  PRINT #1,"R3";
130  END
140  IF LOC(1)=0 THEN 140 ELSE W$=INPUT$(LOC(1),#1)
150  W=ASC(W$)-32
160  RETURN
```

This program is familiar to you by now since it implements a long move as discussed previously. However, line 20 directs the controller to turn off output #3 before starting the programmed motor move. Lines 30 through 90 implements the long move. The program waits until the robot comes to a complete stop in lines 100 through 110. Upon completion of the move, line 120 outputs the command to turn on port 3.

CHAPTER SEVEN

MAINTAINING THE XR

This section shows you how to perform simple but important maintenance on the XR-3 arm. It describes and illustrates the drive train for each axis and then explains how to maintain it.

We suggest that you run through this section once after your first session with the XR-3. After that, keep in mind the points we cover as you use the robot.

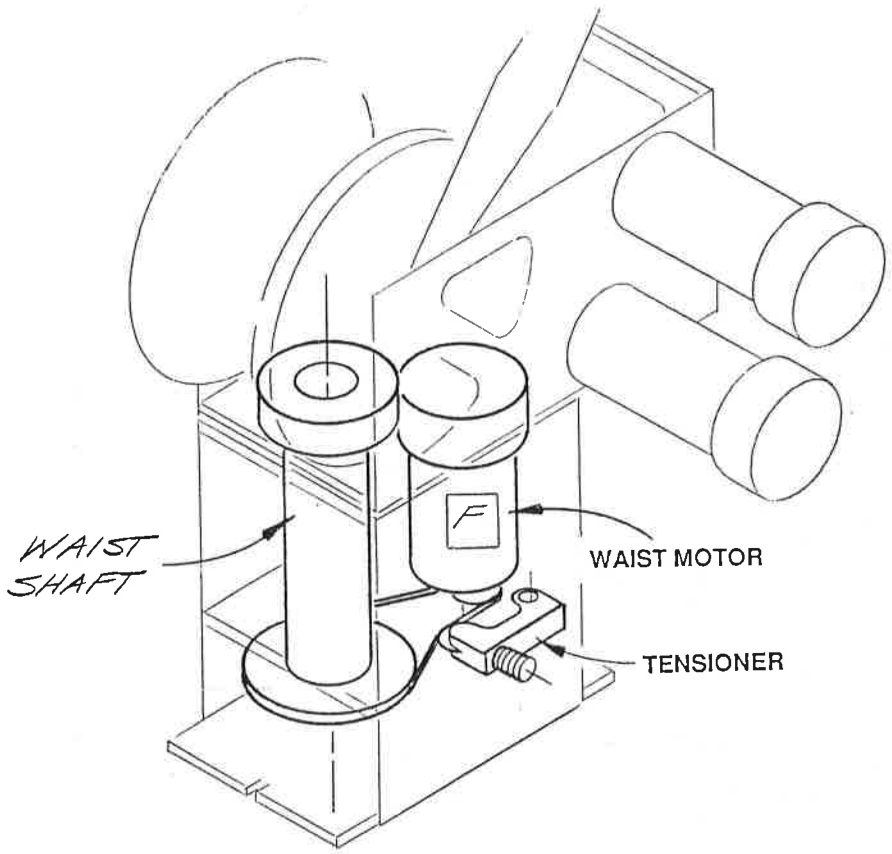


Figure 7.1
Waist (F Motor) Drive

THE WAIST (F MOTOR) DRIVE

Refer to figure 7-1 to identify the waist drive components. The **F** motor, which drives the waist, is mounted in the base housing below the actual waist pivot. A sprocket on the motor shaft drives a chain connected to the hollow waist shaft. The waist shaft extends through the base, and then connects to the robot body.

Maintenance

The only maintenance recommended for the waist drive system is the monthly oiling of the chain tensioner roller. Using a lightweight oil, apply one drop to the upper surface of the wheel allowing the oil to reach the roller shaft.

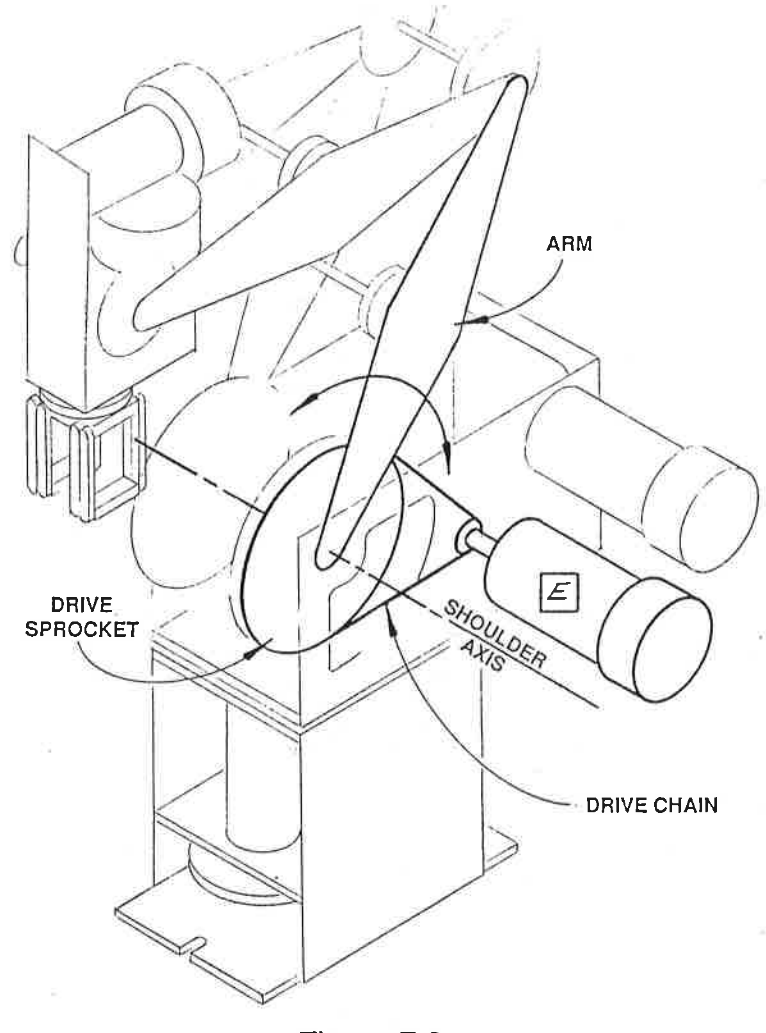


Figure 7.2
Shoulder (E Motor) Drive

THE SHOULDER (E MOTOR) DRIVE

Moving upward from the waist, the next joint (or axis) is the shoulder. Refer to figure 7-2 to identify the shoulder drive components.

The **E** motor mounts on the left side of the robot body, the side with two motors. The **E** motor is the lower of the two.

A sprocket mounted on the **E** motor shaft drives a chain. That chain drives a larger sprocket attached to the upper arm.

The large sprocket and upper arm pivot on an axle common to other sprockets, but they move independently.

Maintenance

First make sure that the sprocket on the motor shaft is fastened securely onto the motor shaft, and that it is aligned with the drive sprocket. Check that the chain runs a straight path from sprocket to sprocket. To adjust alignment, loosen the set screw on the motor sprocket and slide the sprocket along the motor shaft as necessary.

The set screw on the motor sprocket should be centered and tight over the flat portion of the motor shaft.

Next, test the drive chain tension. Applying no more than one pound of force in the middle of the chain span, there should be no chain deflection. If there is excessive play in the chain, adjustment will be needed.

To adjust chain tension, locate the two **E** motor mounting screws visible on the inside of the robot body. Loosen them and rotate the **E** motor to adjust the tension. Refer to figure 7-2 if necessary. Tighten the screws when the chain is in proper adjustment. Make sure that the chain has a slight amount of play in it at all positions of the arm. There may be points where it is tighter than at other points.

THE ELBOW (D MOTOR) DRIVE

The next axis up, moving away from the base along the arm, is the elbow. Refer to figure 7-3 to identify the elbow drive components.

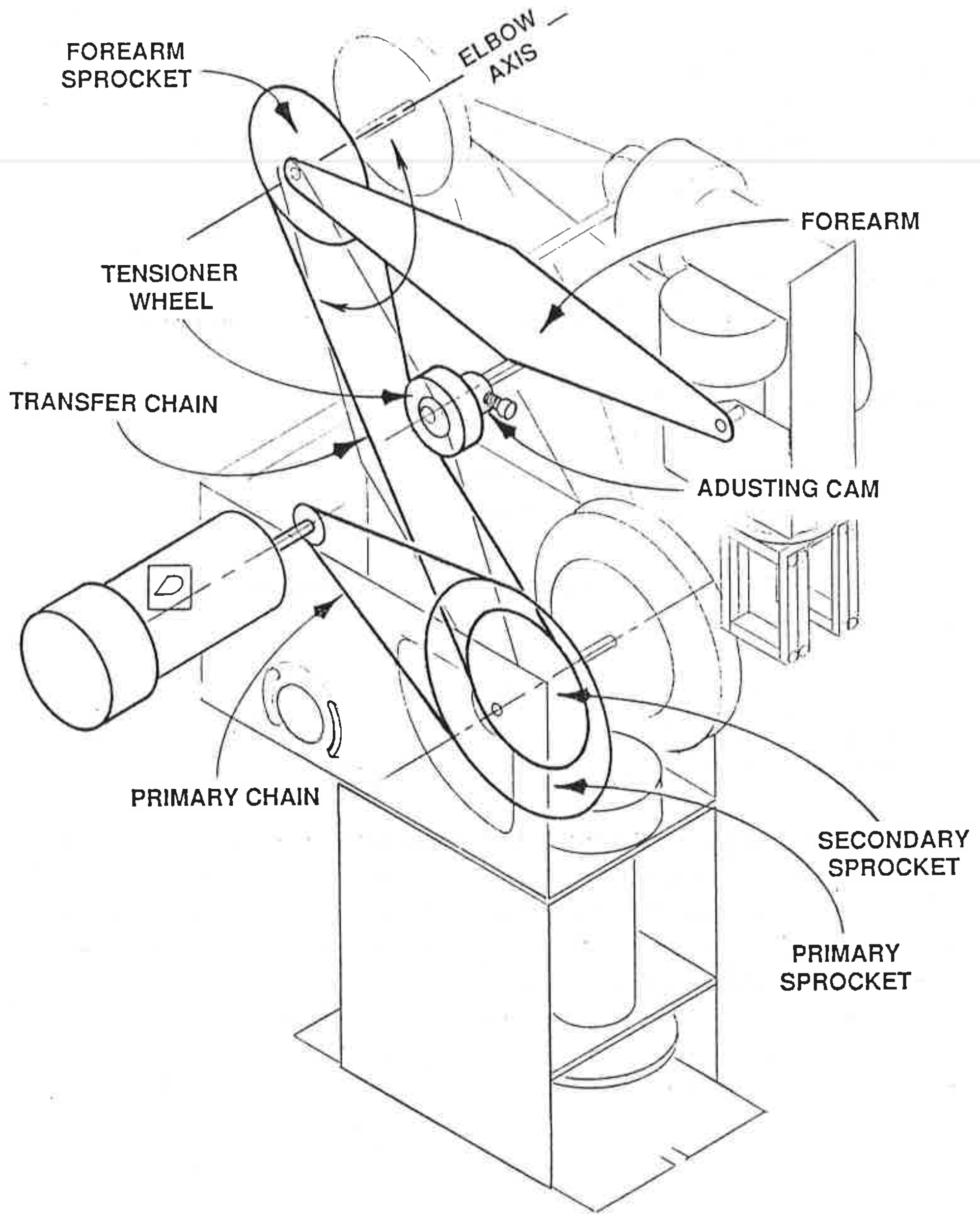


Figure 7-3
Elbow (D Motor) Drive

The **D** motor mounts on the right-hand side of the robot body, it is the only motor extending from that side.

The elbow uses a two part drive.

At the primary drive, a sprocket on the **D** motor drives a chain, which in turn drives a larger sprocket called the primary sprocket. In this way, the primary drive is basically the same as the **E** motor drive.

The secondary sprocket drives the secondary chain, which in turn drives the forearm sprocket.

Maintenance

Check the primary drive from the **D** motor as you did the **E** motor drive.

1. Check for alignment between the motor sprocket and primary sprocket. Make adjustments by moving the motor sprocket along the motor shaft.

2. Check chain tension. Adjust by rotating the **D** motor in its mounting slots.

Refer to the previous section on motor **E** if necessary.

Your main concern on the secondary drive is chain tension. Because the motor is not part of the secondary drive, an eccentric chain tensioner is used. Test the chain as before, but be sure to check the section that does not run over the tensioner.

If the chain is not snug, you must make an adjustment. To do so, locate the set screw that extends from the inner hub of the chain tensioner. Loosen the screw and use it as a lever to rotate the tensioner. Because it is a cam, turning the tensioner in one direction increases chain tension, moving it in the other decreases tension. Rotate the tensioner until the chain is snug when you test it. Then tighten the set screw on the adjusting hub. Make sure that the tensioner is able to rotate freely on the tensioner cam. If it can not, clean and lubricate it so that it can.

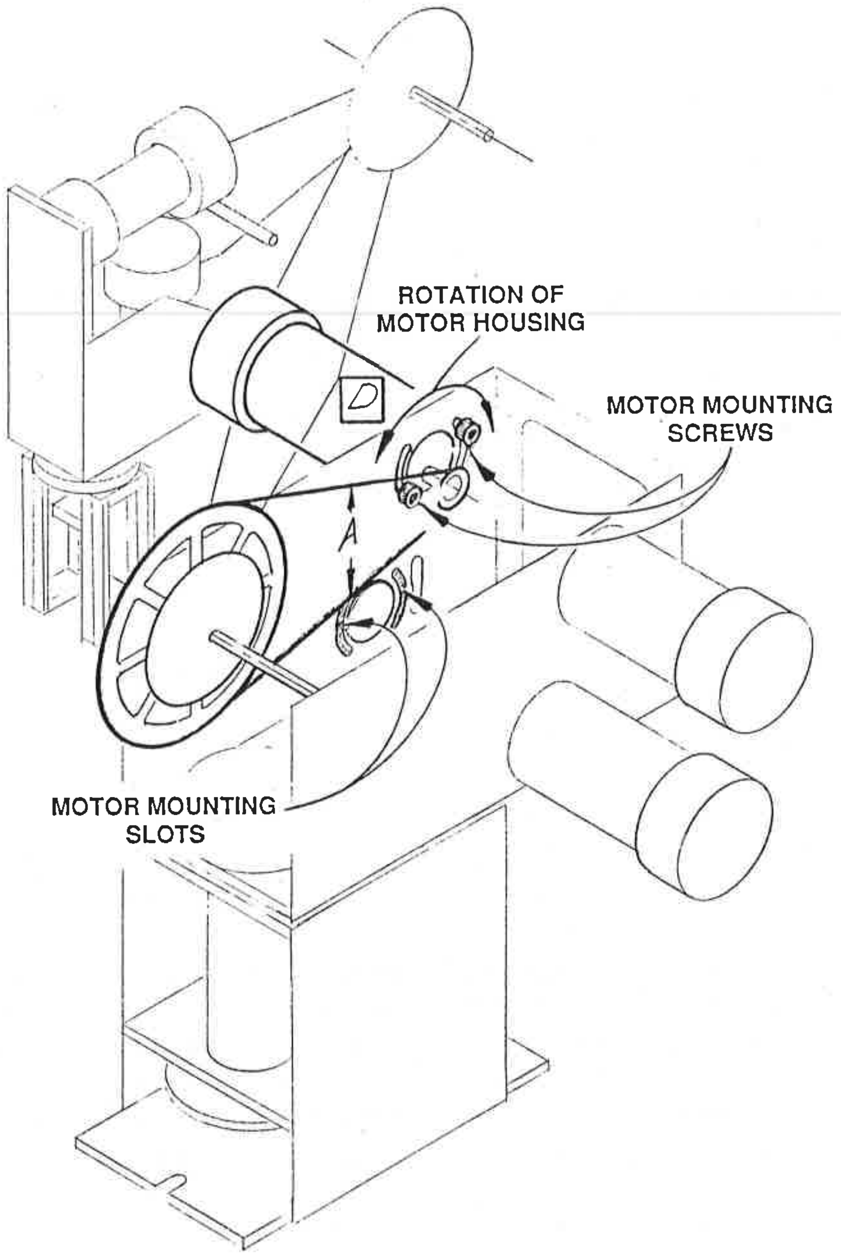


Figure 7.4
Chain Tension Adjustment

THE HAND (C MOTOR) DRIVE

The next axis up, moving away from the waist, is the wrist flex. Refer to figure 7.5 to identify the wrist flex drive components.

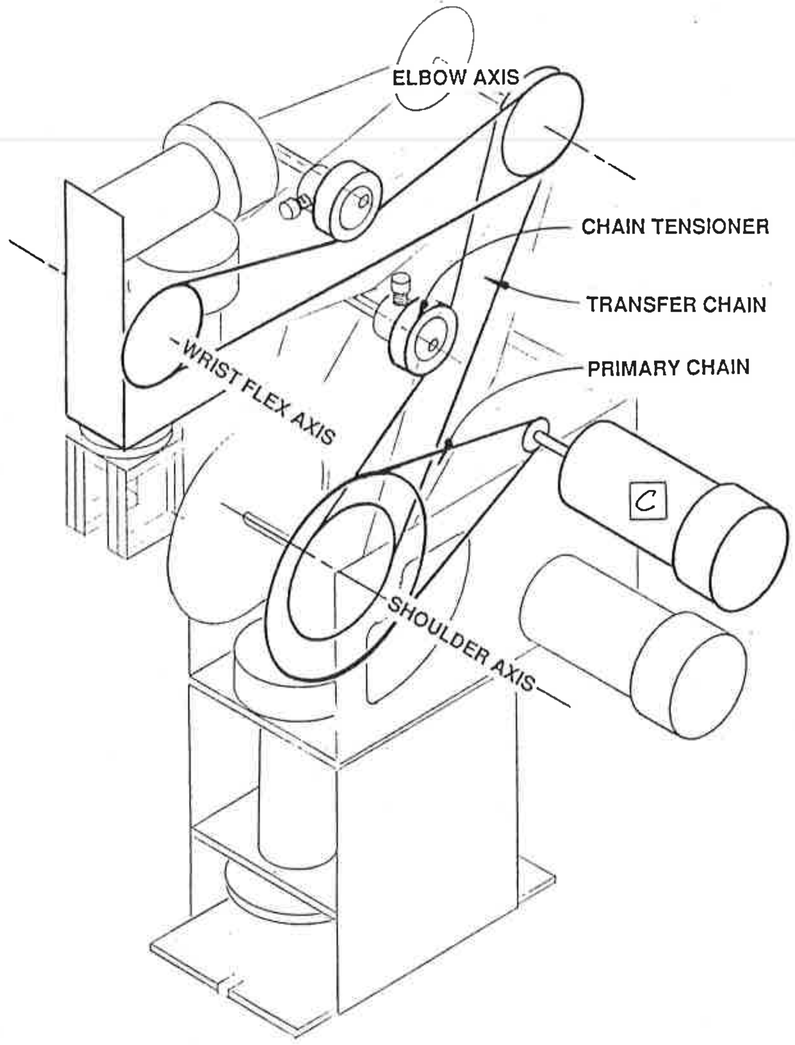


Figure 7.5
Hand (C Motor) Drive

The **C** motor mounts on the left-hand side of the robot--it is the upper of two motors mounted on that side.

The wrist flex uses a three-part drive.

The primary and secondary drives resemble the primary and secondary drives of the elbow.

A sprocket on the **C** motor drives a chain, which in turn drives a primary sprocket. As on the elbow, this sprocket attaches to and drives a third sprocket.

This secondary sprocket drives a chain and fourth sprocket. This fourth sprocket, however, attaches to and drives a sprocket called the elbow hub.

This sprocket drives a chain, which in turn drives the wrist sprocket. This second sprocket attaches to and drives the hand to produce wrist flex.

Maintenance

Maintain the primary and secondary drives as you would the drives for the elbow.

On the primary drive do the following:

1. Check alignment between the sprockets. Adjust by moving the motor sprocket along the motor shaft;

2. Check chain tension on all chains. Adjust the proimary chain's tension by rotating the **C** motor in its mounting slots. The remaining chains are tensioned by adjusting the appropriate chain tensioner.

THE WRIST ROTATION (B MOTOR) DRIVE

Moving away from the waist and toward the fingers, the next axis is the wrist (figure 7.6).

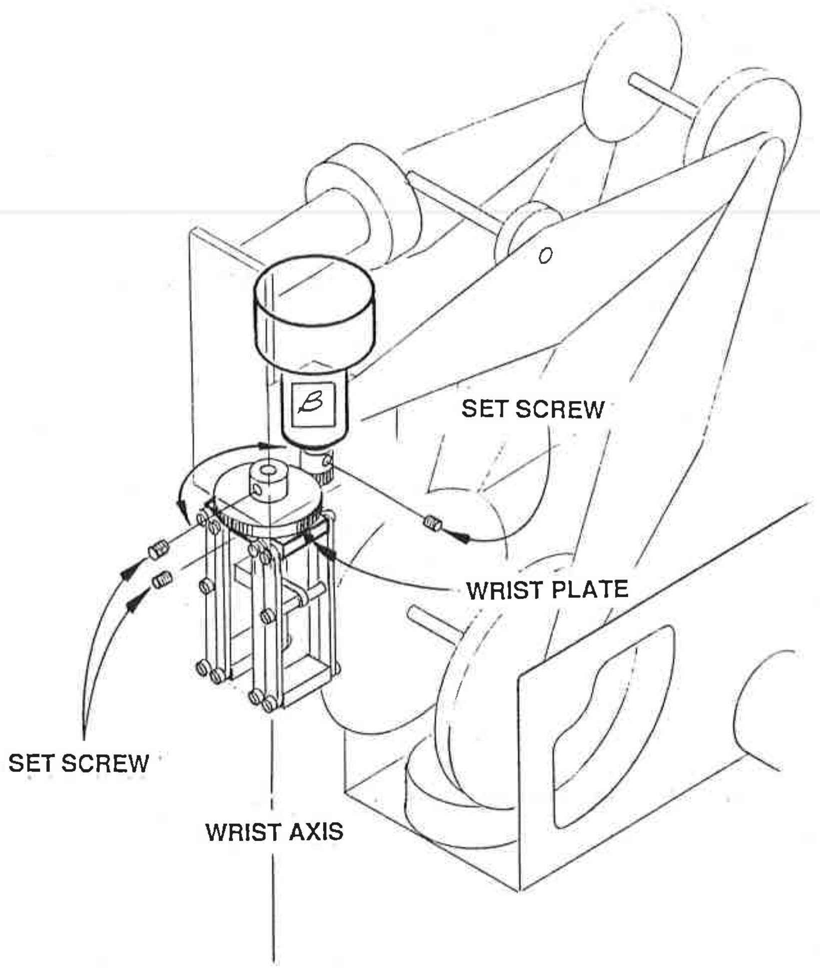


Figure 7.6
Wrist Rotation (B Motor) Drive

The wrist uses a very simple drive train. A pinion gear mounted on the **B** motor shaft drives a gear attached to the wrist plate.

Maintenance

The only maintenance required on the wrist rotation is checking the gears. Each gear has a set screw. Make sure that both are tight and in alignment with one another. Make sure that the small gear does not hit the wrist plate above the large gear when the hand rotates.

CAUTION: Overtightening the set screw on the larger plastic gear can strip the threads. As long as the gear does not slip, the set screw is sufficiently tight. If you must tighten the set screw, do so in small increments. Check the gear after each increment, and stop as soon as the gear no longer slips.

THE FINGER (A MOTOR) DRIVE

The **A** motor mounts on the top of the hand. Refer to figure 7.7

The push-pull arm mounts directly on the **A** motor shaft. This arm drives the long arm link, which mounts and pivots on the end of the push-pull arm.

The long arm link drives the long actuating arm. The long arm link also pivots at the end of the long actuating arm.

The long actuating arm mounts on the transfer shaft that extends inside the hand. There, the intermediate actuating arm mounts on the same transfer shaft, and must be parallel to the long actuating arm.

The intermediate actuating arm drives the short arm link. The short arm link drives (pushes or pulls) the actuating rod.

In turn the actuating rod drives the tendon nut, which opens or closes the finger depending on the direction the motor is turning.

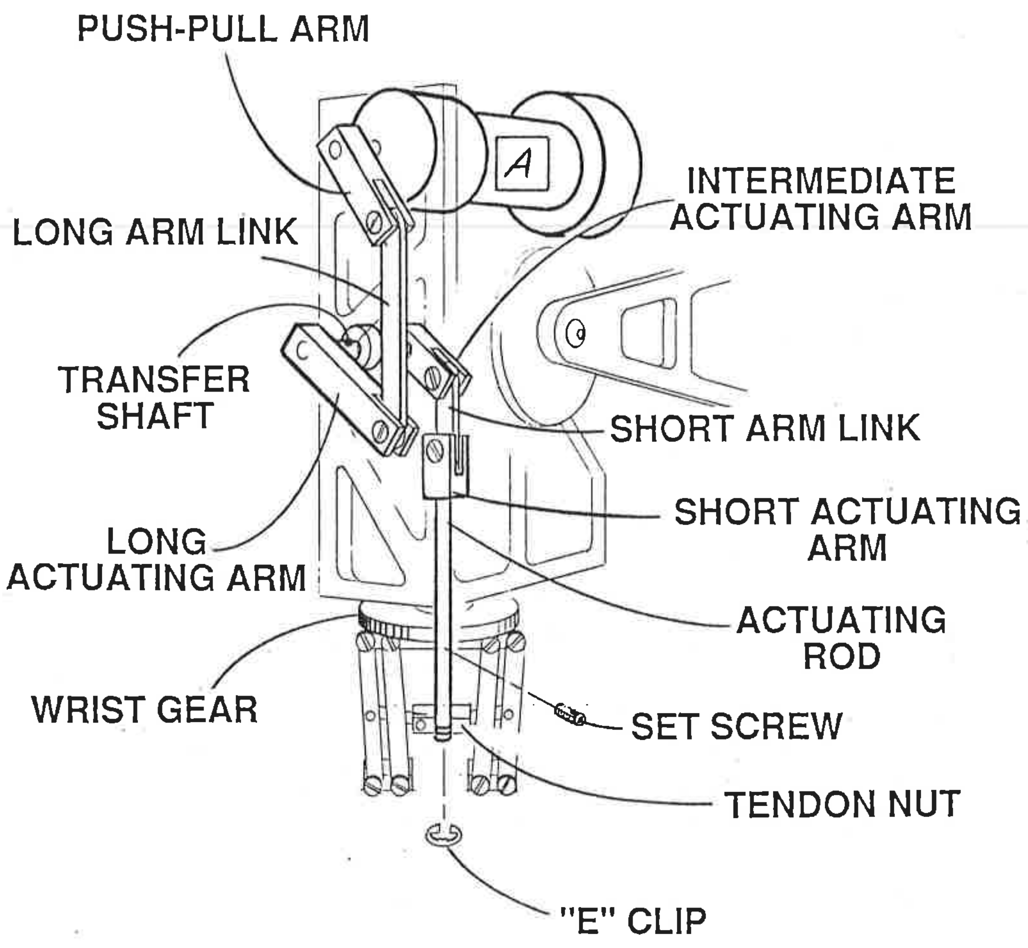


Figure 7.7
Finger (A Motor) Drive

REMOVING FINGERS

To remove the standard fingers, as you must to install any optional end-of-arm tooling, refer to figure 7.7. To remove the fingers, remove the E clip below the tendon nut and loosen the set screw. Carefully insert a small screwdirver between the wrist gear and the finger assembly to loosen. Once loose, the finger assembly can be removed by pulling the finger assembly away from the hand.

Reverse the procedure to install optional grippers.

GENERAL MAINTENANCE

Besides the adjustments just given, follow these procedures.

1. Make sure all large sprocket set screws are tight. You should have already checked the motor sprockets. Now check all points at which sprockets attach to other sprockets, arm section, or pulleys.

   NOTE: Loc-tite™ or a similar product helps prevent set screws from coming loose.

2. Check set screws that fasten the frame and tighten if necessary.

3. Lubricate the five drive chains. Use 3-in-1 oil. Apply a few drops from one sprocket, along one length of chain to the other sprocket. Run the robot to distribute the oil along the chain and sprockets.

4. Lubricate the chain guides. The guides are the hubs mounted on the tensioners. The guides should rotate along with the chain. If they do not rotate, tiny specks of aluminum will begin to flake from the guides.

5. Apply a drop of oil to the **B** motor gear and wrist gear.

6. Every moving part on the XR-3 can stand a drop of oil from time to time.

CHAPTER EIGHT

IF YOU HAVE A PROBLEM

This section describes some very basic trouble shooting procedures.

PROBLEMS DISCOVERED DURING SETUP

If a motor does not respond properly, first try the obvious.

1. Make sure the controller is plugged in.
2. Make sure the main power switch is on and the red pilot light on the front panel of the controller is lit.
3. Make sure the motor power switch is on.
4. Make sure that the motor cable is plugged firmly into its port.

If the motor still does not work, turn off the motor power on the controller and plug the suspect motor into a port that is working. Turn the motor power back on and try the motor again.

If the motor now responds properly, you know that the problem is in the original controller port.

If the motor still doesn't work, you know that the problem can be anywhere from the cable connector to the motor itself.

Whether the problem lies in the controller port or in the motor and its ribbon cable, we recommend that you don't try to repair it yourself unless you are a competent electronics technician, especially while the unit is under warranty. Instead, we recommend you call the **RHINO Hotline (217) 352-8485.**

When you call, describe the problem and we may be able to come up with a simple solution. If we cannot, we'll give you a return authorization number and instructions on how to return the faulty component.

PROBLEMS ENCOUNTERED AFTER CONNECTING TO A HOST COMPUTER

If all the motors checked out during setup, but the XR does not respond to commands you give the controller, there is probably an interface problem. If you own one of the computers covered individually in this manual, retrace the interfacing instructions from the beginning.

Whatever computer you're using, make sure that you have plugged your computer into the controller port labeled "Computer" on the Mark III controller.

If you are using a computer other than one covered specifically in this manual, refer back to the general interfacing chapter. Remember that the controller uses only three lines.

line 2 transmits to the host
line 3 receives data sent by the host
line 7 is a common data ground

If the XR is not responding, chances are your computer uses a protocol that requires a signal that the controller does not provide. Remember that on the DB25 connectors, pins 6, 8 and 20 and pins 4 and 5 must have a jumper wire soldered between them. Modifications to the connector for your computer depend on your computer. Recheck the manual for your machine. Contact a service representative if necessary, and you can always call the RHINO Hotline (217) 352-8485.

OPTICS PROBLEMS

If you send a START command to a motor and it starts but runs away (doesn't stop) in one direction, optical encoder or the 74LS14 encoder line buffer is probably defective.

Also, if you determine that a motor is missing counts--that is it travels more or fewer steps then you command, the encoders have a problem.

MECHANICAL PROBLEMS

If a motor turns but its axis does not, a sprocket or a chain is probably loose. Refer to the maintenance chapter on mechanical maintenance for a remedy.

CALL US

We built the XR to be rugged and dependable, and you probably will encounter few if any problems. But if you do, remember that you can always call us. Before calling, narrow the problem down as far as possible.

Be sure to have the serial number of your robot handy when you call. Slight changes during production runs can make a difference when we are diagnosing your problem.

Even if you are sure you need to send a component back to us, call first. For us to accept the component, you must have a return authorization number. And, if you do ship a component back to the factory, pack it in the same materials in which you received it.

Remember, if you have a problem, call us at **(217) 352-8485.** Monday through Friday, 9:00 a.m. to 5:00 p.m. Central Daylight Time.

CHAPTER NINE

PARTS--FIGURES AND LISTS

The following pages contain exploded-view diagrams for the XR-3 series robot. The figures are accompanied by parts lists.

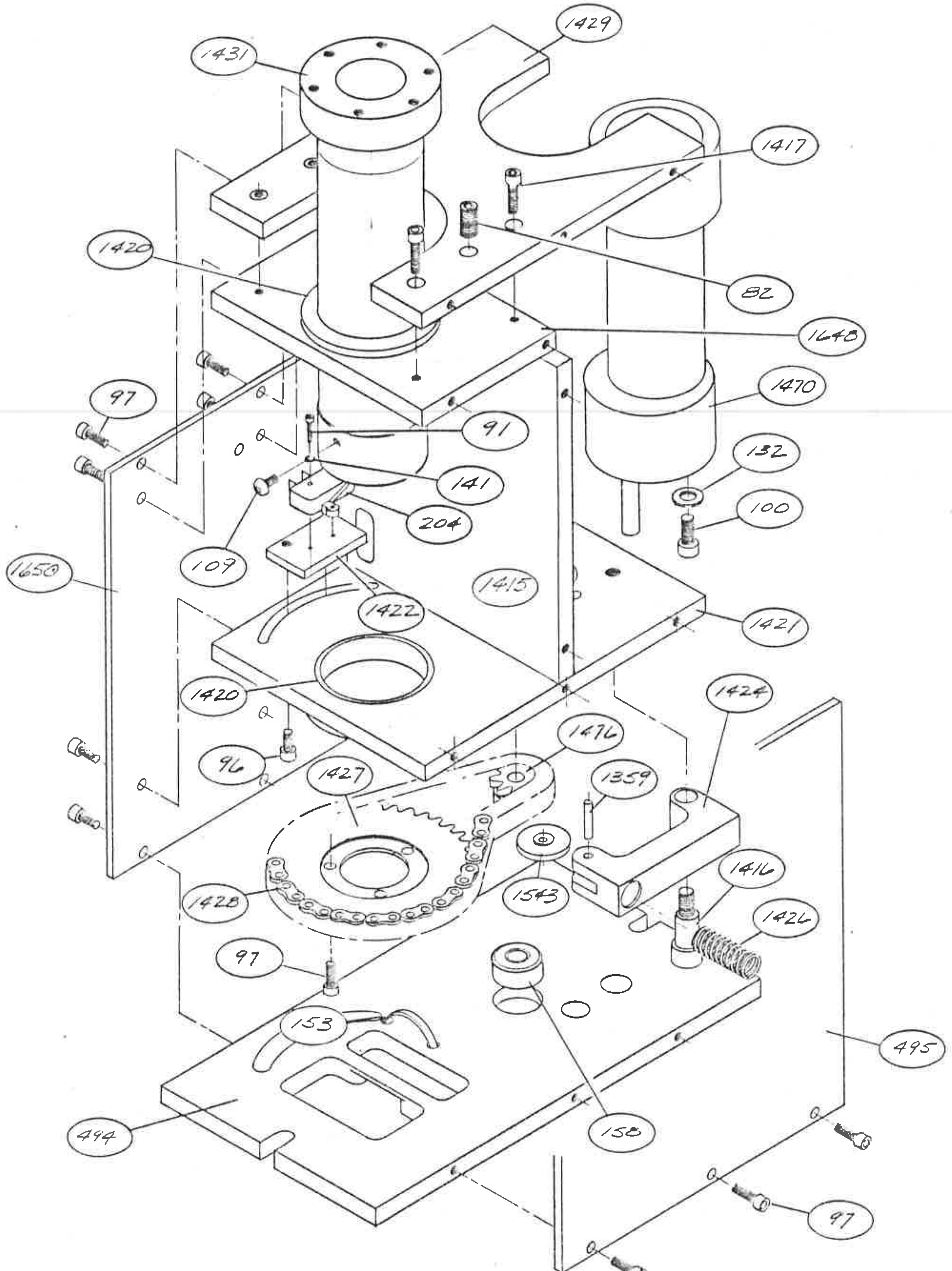
Although the diagrams were complete and accurate at the time of publication, minor changes may have been implemented on subsequent robots. Because of this, your robot may differ slightly from the following diagrams.

The reference numbers shown on the exploded drawings refer to the last four digits of the Rhino Part Number. When ordering parts, please specify the complete part number.

PARTS LISTS

FIGURE 9.1: BASE COMPONENTS

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
158	10-40-158	Ball bearing 1602 DS	1
204	30-140-204	Microswitch	1
494	50-301-494	Base bottom	1
495	50-301-495	Base side-A	1
1359	15-70-1359	.125 x 1/2 Dowel pin	1
1415	50-301-1415	Base Center Plate	1
1416	15-41-1416	1/4-20 x 1/2 Shoulder bolt	1
1420	50-307-1420	Waist Shaft Bearing	2
1421	50-307-1421	Base Motor Mount Plate	1
1422	50-307-1422	Switch Mounting Plate	1
1424	50-307-1424	Chain Tensioner	1
1426	10-76-1426	Tensioner spring	1
1427	50-301-1427	Waist Drive Sprocket	1
1428	10-90-1428	48P #25 Chain	1
1429	50-301-1429	Base Top	1
1431	50-302-1431	Hollow Waist Shaft	1
1470	99-308-1470	F motor, Version 5	1
1476	99-307-1476	9 Tooth sprocket	1
1543	50-307-1543	Roller	1
1648	50-301-1648	Top Bearing Plate	1
1650	50-301-1650	Base Side - B	1



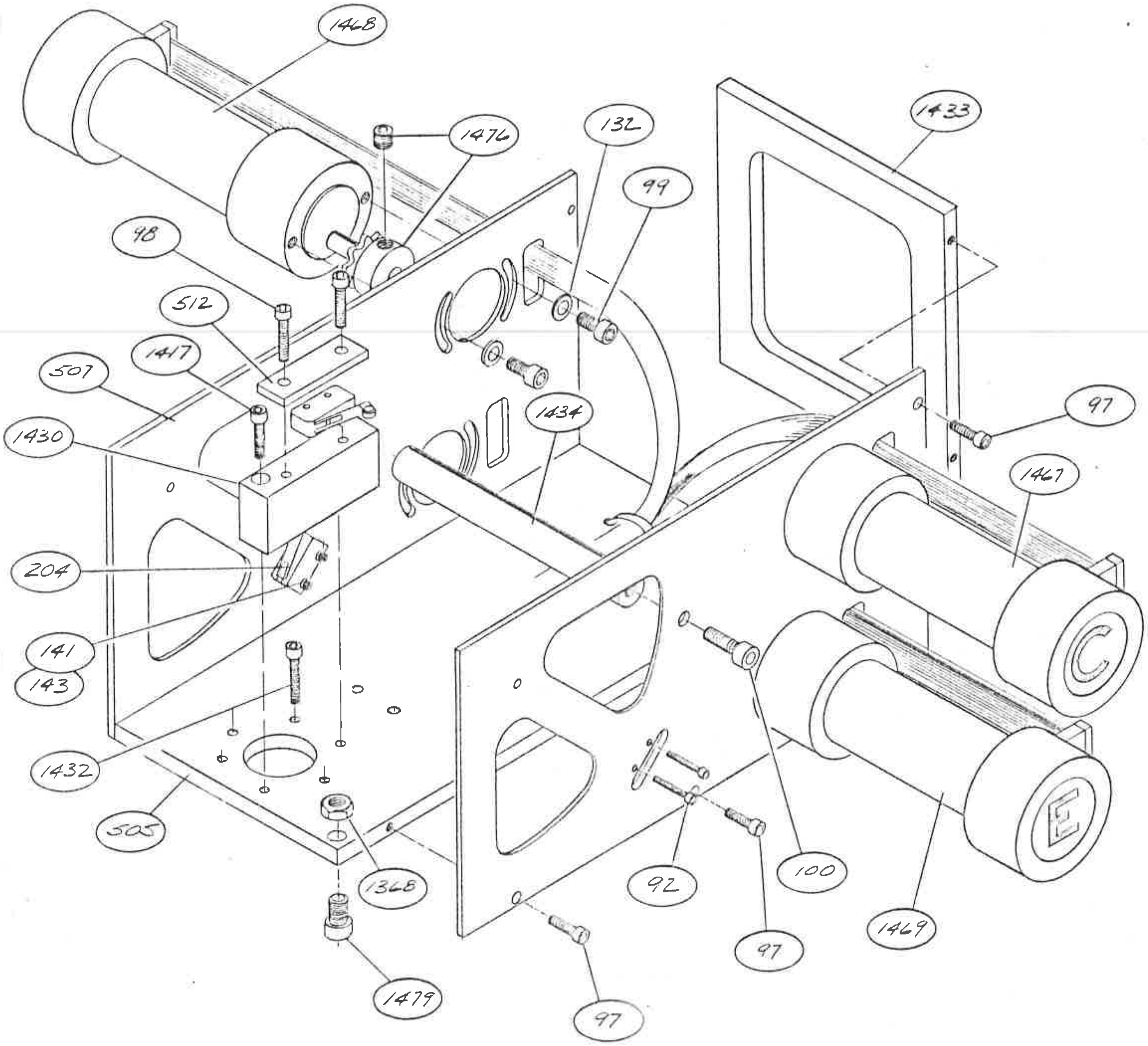
BASE ASSEMBLY
XR-3

FIGURE 9.1

PARTS LISTS

FIGURE 9.2: BODY COMPONENTS

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
204	30-140-204	Microswitch	3
505	50-302-505	body bottom	1
507	50-302-507	body side	2
512	50-307-512	microswitch clamp	1
1430	50-302-1430	Switch Block	1
1433	50-302-1433	Body Rear Plate	1
1434	50-302-1434	Body Side Spacer	1
1467	99-308-1467	C motor, Version 5	1
1468	99-308-1468	D motor, Version 5	1
1469	99-308-1469	E motor, Version 5	1
1476	99-307-1476	9 Tooth sprocket	3



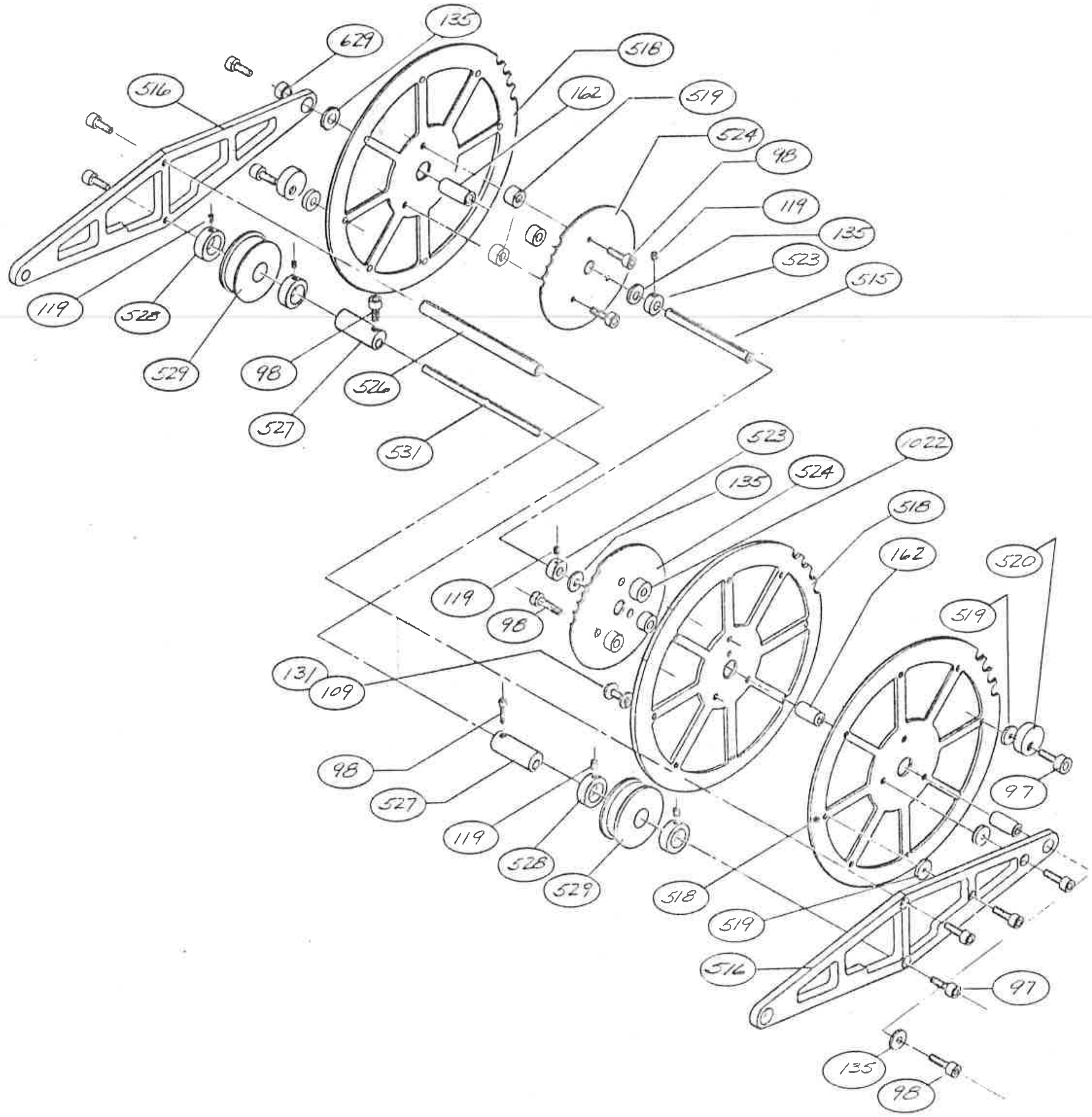
BODY ASSEMBLY
XR-3

FIGURE 9.2

PARTS LISTS

FIGURE 9.3: LOWER ARM COMPONENTS

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
135	15-23-135	1/4" nylon washer	6
162	10-44-162	5/8 x 1/4 x 7/16 bushing	3
515	50-307-515	1st section pivot shaft	1
516	50-303-516	lower right arm	2
518	50-307-518	72 T drive sprocket	3
519	50-307-519	drive sprocket spacer	4
520	50-307-520	1st section cam	2
523	50-307-523	collar 1/4"	2
524	50-307-524	36 T sprocket	2
526	50-303-526	lower arm spacer	1
527	50-307-527	tensioner eccentric	2
528	50-307-528	tensioner collar	4
529	50-307-529	tensioner wheel	2
531	50-307-531	2nd section pivot shaft	1
629	10-44-629	1/4" long bushing	1
1022	50-307-1022	1/4" sprocket spacers	6



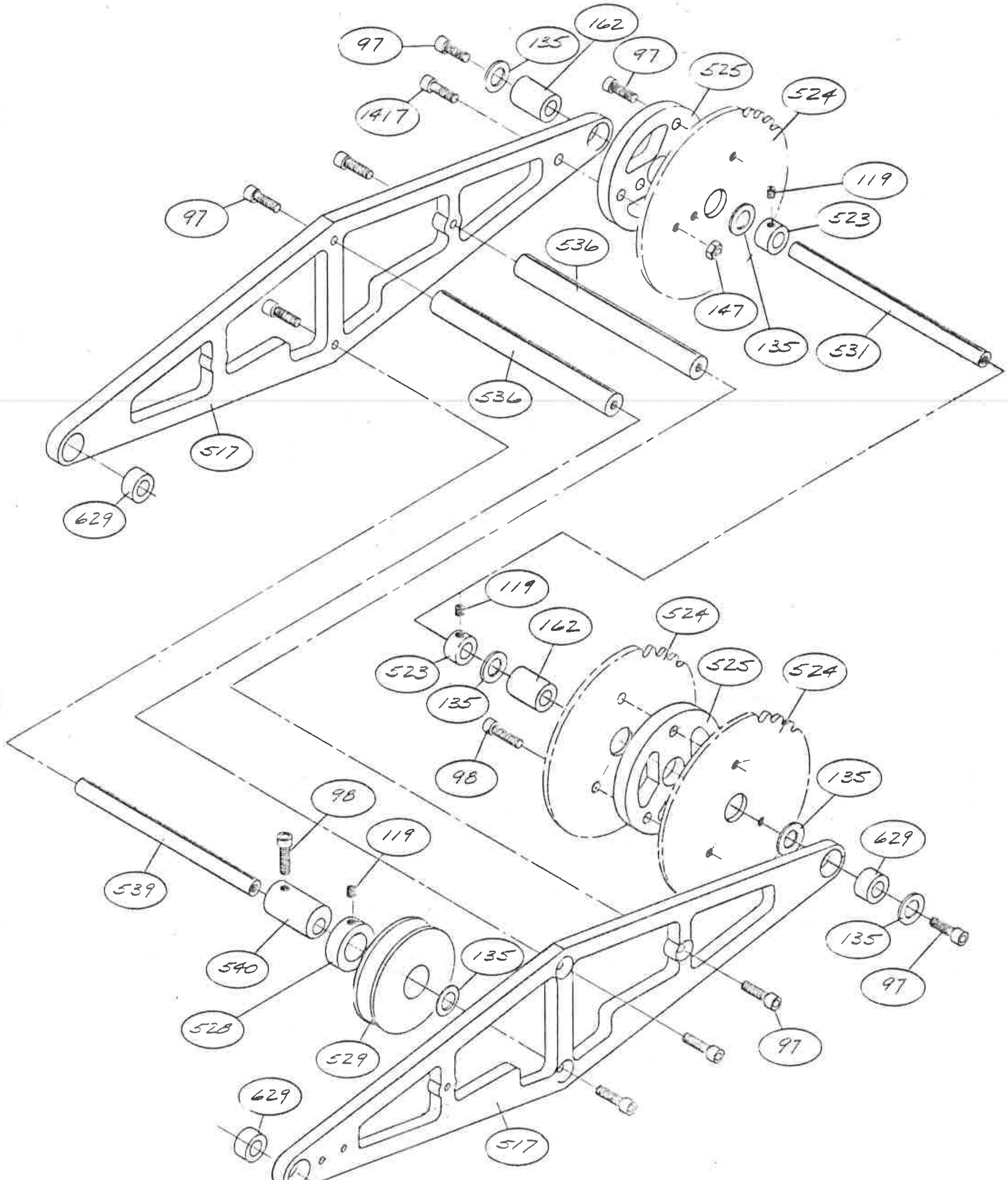
LOWER ARM ASSEMBLY
XR-3

FIGURE 9.3

PARTS LISTS

FIGURE 9.4: UPPER ARM COMPONENTS

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
135	15-23-135	1/4" nylon washer,thin	4
162	10-44-162	5/8 L 1/4 x 7/16 bush	2
517	50-303-517	Upper Arm	2
523	50-307-523	Collar 1/4"	2
524	50-307-524	36 T sprocket, unthreaded	3
525	50-307-525	Sprocket support hub	2
528	50-307-528	Tensioner collar	1
529	50-307-529	Tensioner wheel	1
531	50-307-531	2nd section pivot shaft	1
536	50-303-536	Upper arm spacer	2
539	50-307-539	Upper tensioner shaft	1
540	50-307-540	Upper tensioner eccentric	1
629	10-44-629	1/4" long bushing	3



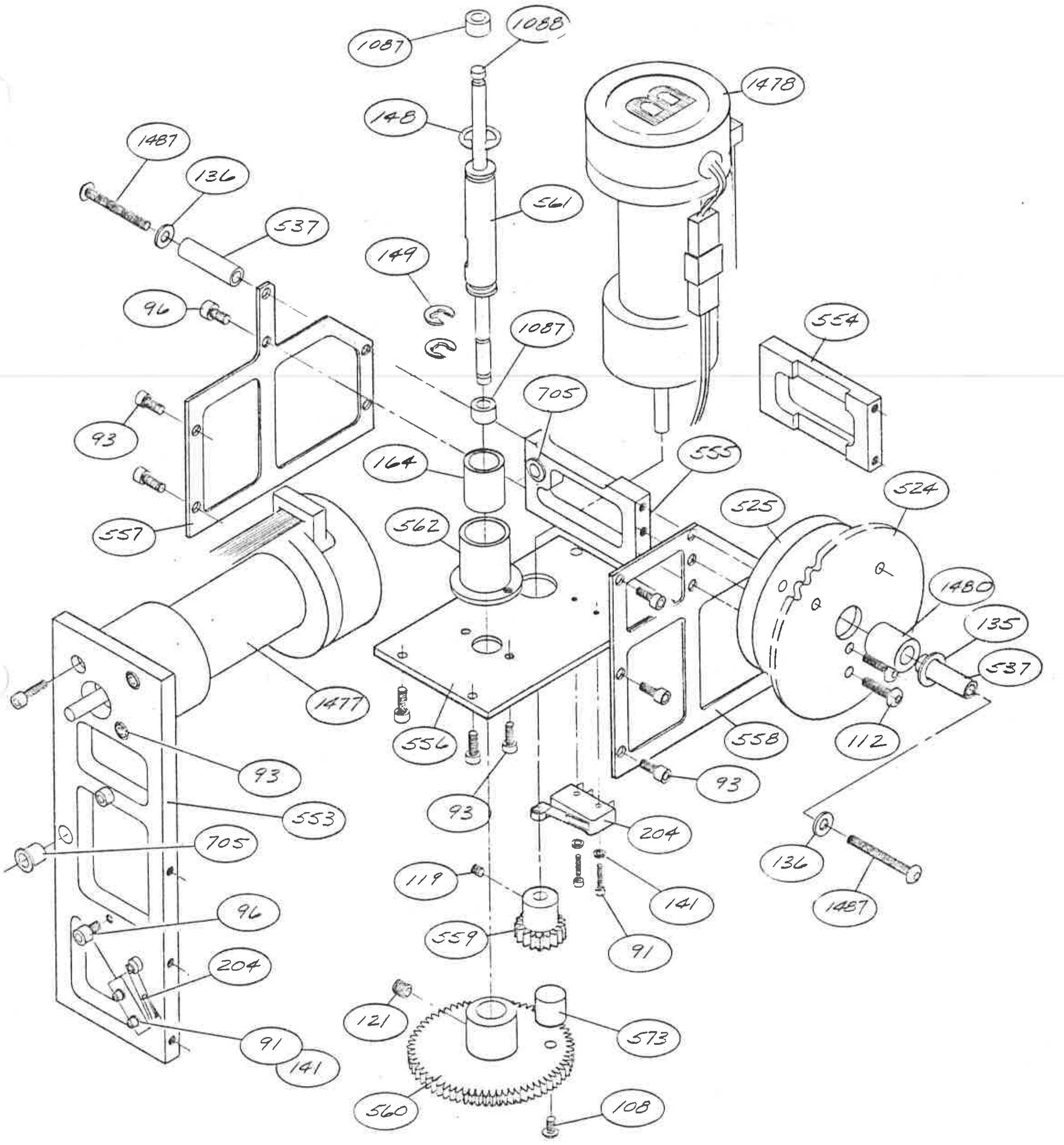
UPPER ARM ASSEMBLY
XR-3

FIGURE 9.4

PARTS LISTS

FIGURE 9.5: HAND & WRIST DRIVE COMPONENTS

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
135	15-23-135	1/4" nylon washer,thin	2
136	15-23-136	#6 nylon washer,thin	2
148	15-30-148	Large E ring	1
149	15-30-149	Small E ring	2
164	10-44-164	3/4 L x 3/8 x 9/16 bush	1
204	30-14-204	Microswitch	2
524	50-307-524	36T sprocket, unthreaded	1
525	50-307-525	Sprocket support hub	1
537	50-304-537	Hand mount spacer	2
553	50-304-553	Hand mount frame	1
554	50-304-554	Hand rear frame	1
555	50-304-555	Hand intermediate frame	1
556	50-304-556	Hand bottom frame	1
557	50-304-557	Hand right frame	1
558	50-304-558	Hand left frame	1
559	50-304-559	Hand gear, small	1
560	50-304-560	Hand gear, large	1
561	50307-561	Wrist shaft	1
562	50-304-562	Hand bearing block	1
573	50-304-573	Hand cam	1
705	10-44-705	Nylon bushing	2
1087	10-156-1087	7/32 Rubber spacer	2
1088	50-305-1088	Finger Actuator shaft	1
1477	99-308-1477	A motor, version 5	1
1478	99–308-1478	B motor, version 5	1
1480	10-44-1480	1/2L x 1/4 x 7/16 bushing	1



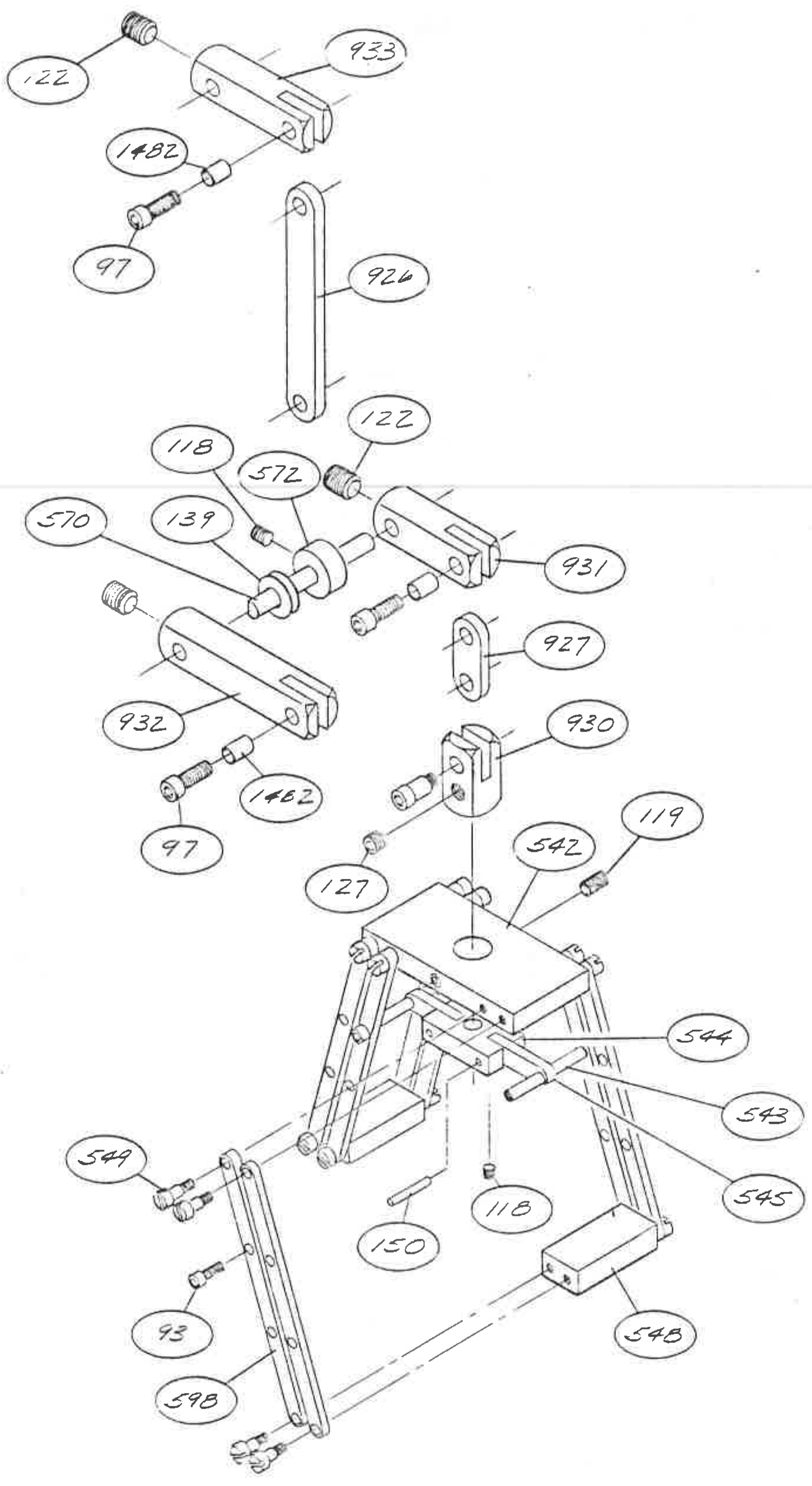
HAND & WRIST DRIVE ASSEMBLY
XR-3

FIGURE 9.5

PARTS LISTS

FIGURE 9.6: FINGER ASSEMBLY & LINKAGE

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
139	15-23-139	#10 nylon washer	3
150	15-70-150	steel pins 9Q32-0320	2
542	50-307-542	wrist plate	1
543	50-305-543	finger pivot post	2
544	50-305-544	finger tendon nut	1
545	50-305-545	finger actuating link	2
548	50-305-548	finger tip	2
549	50-305-549	finger shoulder screw	16
570	50-307-570	Cross shaft	1
572	50-307-572	Cross shaft collar	1
598	50-306-598	Finger, 1.5X	8
926	50-307-926	Long connecting link	1
927	50-307-927	Short connecting link	1
930	50-307-930	Pivot Head	1
931	50-307-931	Short cross shaft	1
932	50-307-932	Cross shaft arm long	1
933	50-307-933	Pull arm	1
1482	50-307-1482	Actuator sleeve	2



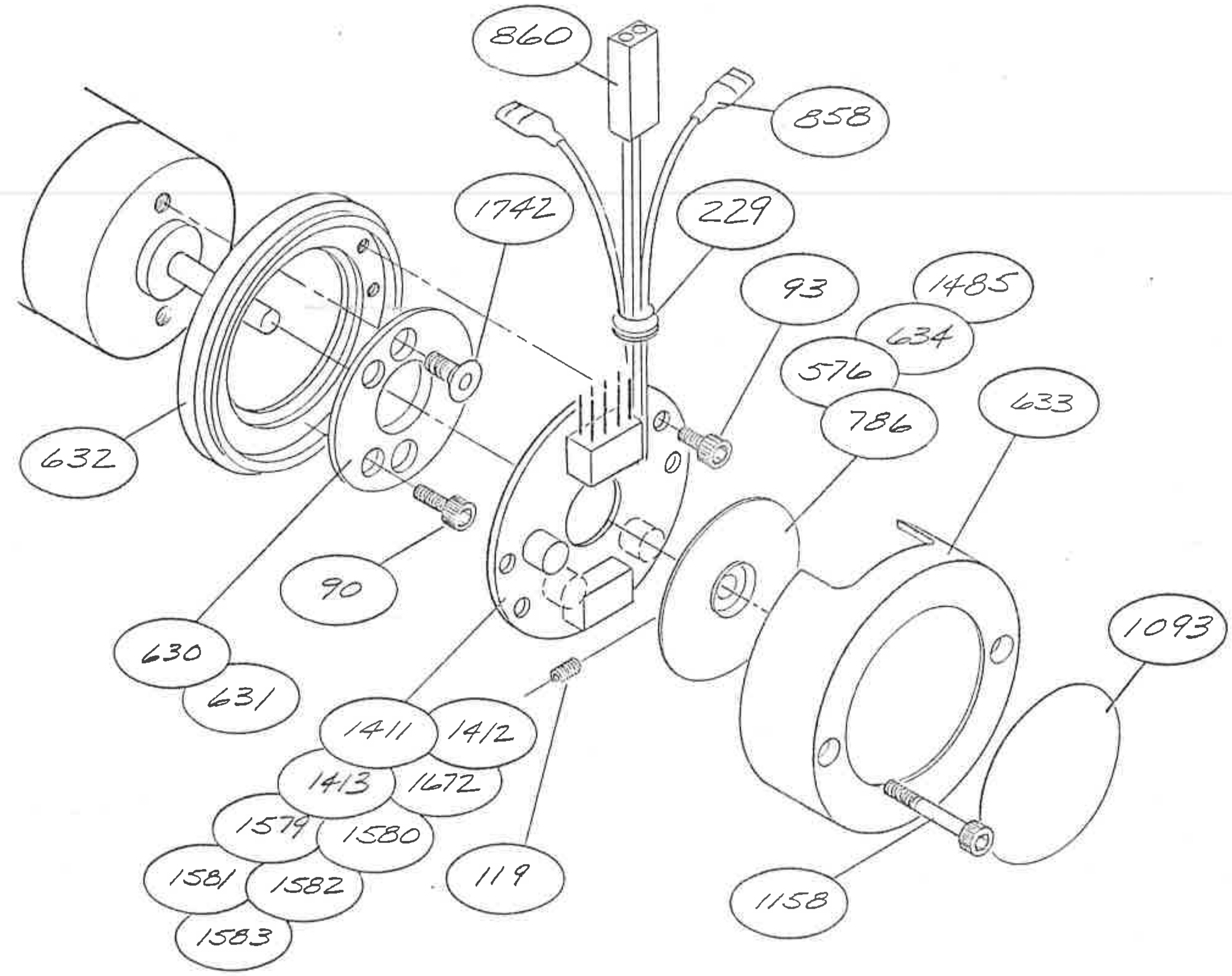
FINGER ASSEMBLY & LINKAGE
XR-3

FIGURE 9.6

PARTS LISTS

FIGURE 9.7: TYPICAL MOTOR COMPONENTS

FIGURE	PART NUMBER	PART DESCRIPTION	QUANTITY
186	30-100-186	D.C. servo motor, large	1
229	10-156-229	Rubber grommet	1
630	50-302-630	Encoder mounting plate	1
632	50-302-632	Encoder base ring	1
633	50-302-633	Encoder end cap	1
636	99-304-636	Disk assm. - 6 state	1
672	30-241-672	Reflective optic	2
843	30-210-843	LM393 Voltage comparator	1



TYPICAL MOTOR COMPONENTS
XR-3

FIGURE 9.7

CHAPTER TEN

HARD HOME ROUTINES

HARD HOME PROGRAMS

The following pages list hard home routines for the Apple IIe and the IBM PC. The routines find the microswitches for motors **D**, **E**, and **F**--the elbow, shoulder, and waist respectively--assuming that the motors are connected in the standard configuration. But you can use these routines as guides to finding the other switches. These routines are presented as samples--you will be able to improve and modify them to suit your particular needs.

Note that these routines do not simply close the switches and stop. They find the switch at full speed and then slow down. The robot travels slowly until it gets off the microswitch and the switch reopens. The axis motor then reverses, travels slowly to close the switch and then reopens it. The controller keeps track of how many steps the motor travels from the open point to the close point. The routine then sends the axis back slowly half that distance ensuring that the cam for that axis not only closes the switch, but also that it stops centered over that switch.

Even if you do not have one of the computers covered here, the routines provide a good model for your own hard home routines.

Hard Home Routine for the Apple

This routine uses the Apple Super Serial card installed in slot #1.

```
10    GOSUB 250: GOTO 170
20    A$=M$(N)+K$+"1"+CHR$(13)
30    FOR J=1 TO LEN(A$): WAIT ST, S: POKE DA, ASC(MID$(A$,J,1)):
           NEXT J: RETURN
40    WAIT ST,R: W=PEEK(DA)-32: RETURN
50    IF N=6 THEN 80
60    A$="I": GOSUB 30: GOSUB 40: FOR I=5 TO 0 STEP -1: A(I)=0: IF
           W>(2^I)-1 THEN A(I)=1: W=W-(2^I)
70    NEXT I: W=A(5-N): RETURN
80    A$="J": GOSUB 30: GOSUB 40: IF W>32 THEN W=1: RETURN
90    W=0: RETURN
100   IF K$="+" THEN K$="-": RETURN
110   K$="+": RETURN
115   A$=M$(N)+"X": GOSUB 30
120   GOSUB 20: GOSUB 50: IF W=0 THEN 120
130   GOSUB 100: CN=0
140   GOSUB 20: GOSUB 50: IF W<>0 THEN 140
150   GOSUB 20: GOSUB 50: CN=CN+1: IF W= 0 THEN 150
160   CN=INT(CN/2): GOSUB 100: A$=M$(N)+K$+STR$(CN)+CHR$(13):
           GOSUB 30: NEXT N: END
170   FOR N=3 TO 6: K$="-"
180   A$=M$(N)+K$+"40"+CHR$(13): GOSUB 30
190   GOSUB 50: IF W=0 THEN 115
200   A$=M$(N)+"?": GOSUB 30: GOSUB 40: IF W<40 THEN 180
210   IF W<=PW+1 AND W>+PW-1 THEN CN=CN+1 GOTO 230
220   PW=W: CN=0: GOTO 180
230   IF CN<10 THEN 190
240   GOSUB 100: CN=0: GOTO 180
250   DA=49304: ST=DA+1: CO=ST+1: CL=CO+1: S=16: R=24: POKE
           ST,0: POKE CL,190: POKE CO,101: M$(3)="E": M$(4)="D":
           M$(5)="C": M$(6)="B": RETURN
```

HARD HOME ROUTINE FOR THE IBM PC

```
10      OPEN "COM1:9600,E,7,2,CS,DS,CD" AS #1
20      M$(0)="E": M$(1)="D": M$(2)="F"
30      M(0)=4: M(1)=2: M(2)=8
40      CT=0
50      FOR N=0 TO 2
60              K$="-": I=0
70              I=I+1: PRINT #1,M$(N);K$;"50"
80              IF N=0 AND K$="-" THEN PRINT #1,"D+50": PRINT #1, "C+40":
                        CT=CT+1
90              IF N=1 AND K$="-" AND CT>0 THEN PRINT #1,"C-40":
                        CT=CT-1
100             GOSUB 310: IF W THEN 110 ELSE PRINT #1,M$(N);"X";:
                GOTO 180
110             PRINT #1,M$(N);"?";
120             GOSUB 330: IF W>27 OR W<0 THEN 160
130             CN=0
140             IF N=2 AND I=20 THEN GOSUB 290
150             GOTO 70
160             IF W=PW THEN CN=CN+1 ELSE PW=W: CN=0: GOTO 100
170             IF CN=10 THEN I=21: GOSUB 290: GOTO 70 ELSE 100
180             PRINT #1, M$(N);K$;"1": GOSUB 310: IF W THEN GOSUB 290:
                        CN=0 ELSE 180
190             PRINT #1, M$(N);K$;"1": GOSUB 310: IF W THEN 190
200             PRINT #1, M$(N);K$;"1": GOSUB 310: CN=CN+1
210             IF W THEN CN=INT(CN/2): GOSUB 290: PRINT #1, M$(N);K$;
                        STR$(CN) ELSE 200
220             IF N<>1 THEN 270
230             FOR L=1 TO CT
240                     PRINT #1,"C?";
250                     GOSUB 330: IF W>27 OR W<0 THEN
                        240 ELSE PRINT #1,"C-40"
260             NEXT L
270     NEXT N
280     END
290     IF K$="+" THEN K$="-" ELSE K$="+"
300     RETURN
310     PRINT #1,"I";: GOSUB 330: W=W AND M(N)
320     RETURN
330     IF LOC(1)=0 THEN 330 ELSE W=ASC(INPUT$(LOC(1),#1))-32
340     RETURN
```

APPENDIX

A TUTORIAL ON DC SERVOS and OPTICAL ENCODERS

SUMMARY

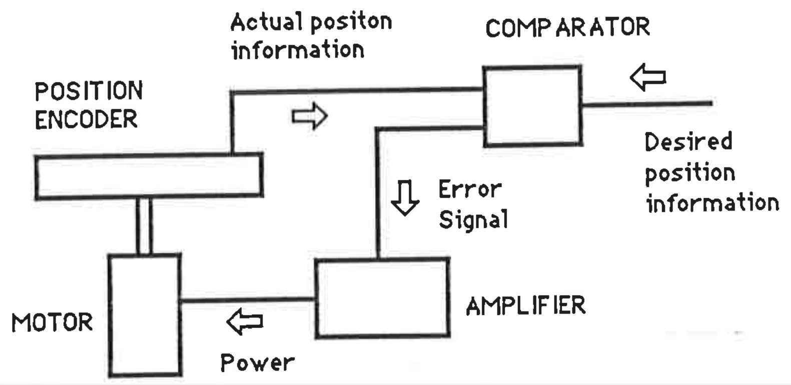
The heart of any robotic system is the servo motors that run it. This section describes servomotors and encoders and discusses the system used by the Rhino XR Robot.

SERVOMOTORS.

Servomechanism (sur-vo-mek-e-niz-em) n. A feedback system that consists of a sensing element, an amplifier and a servomotor used in the automatic control of a mechanical device.

So, in the broadest sense, a servo system is any mechanical device that is controlled by an error signal. By this we mean that a small signal is somehow used to cause a much larger device to be controlled. An amplifier is used to magnify the small signal. In this context, an electrically operated switching relay is probably the simplest device that can be described as a servo. The human being provides the error signal. The human brain knows when it is time to turn the signal on. This relatively weak signal in the brain actuates the human arm and hand. This is one stage of amplification. They in turn turn on the switch. The switch provides power to the coil that pulls in a set of contacts. The contacts control something much larger than the switch itself could handle. This is such a common servomechanism that we normally don't even think of it as a servo.

Another servo system that we have all seen used is the power steering system in an automobile. Though the system works even when the hydraulic system breaks down, the effort then required is very large, and the only reason that the system works without hydraulic assist is that it has to be designed for inherent safety. The design is such that it allows you to control the car even if the system fails. Actually the system is designed to follow the error signal that the steering wheel provides. A system of valves allows the hydraulic fluid in the system to assist the turning of the wheel. You provide the error signal when you start to turn the wheel. The hydraulic system provides the larger force needed to steer the car.



THE SERVO LOOP

Another servo system that, is in common use is the automatic speed (cruise) control system in a car. In this system, we set the desired speed and the cruise system then modulates (increases and decreases) the position of the throttle (gas pedal) to maintain the set speed. There is no need for human interference, except to start and stop the system.

The system is designed so that it will keep adding to the throttle position as long as the car is not going fast enough and will keep subtracting from the throttle position as long as the car is going faster than the desired speed. We call this an integrating control scheme. We are integrating the error signal over time to correct for the error. Integrating the signal gives us better control.

If we had linear or proportional control, we would fix the throttle setting at the given position, that gave us the desired speed, and then not move it. This would work well on a straight and level road if there was no wind. However, the car would slow down on every uphill stretch of road and speed up on every downhill stretch.

Integrating control is necessary to compensate for varying load conditions; as the load increases, we increase the power by responding to the error signal. Integrating the error signal over a period of time allows for better response because it effectively increases the magnitude of the error signals received.

We have a similar problem when we pick up loads of different weights with a robot. The electrical energy needed to pick up the arm itself is much less than the energy needed to pick up a heavy load at the same speed. Some scheme has to be devised to change the energy level as necessary.

The problem of moving a robot arm consists of starting, running and stopping the motor within a specified time-distance framework. Put another way, the motor has to do the following basic things:

1. Maintain its stopped position. (Provide a holding torque.)
2. Start smoothly and come up to speed.
3. Maintain the specified speed under varying load conditions.
4. Slow down when nearing the destination
5. Stop at the predetermined point
6. Maintain the stopped position.

Here we are discussing simple straight line motion on a simple robot. As geometries and sophisticated requirements are added, other factors come into play. Some of these can be enumerated as follows:

1. Calculate exact path requirements
2. Monitor load and encoders to determine if a stall condition or an overload condition has occurred.
3. Respond to speed changes in mid motion.
4. Do look ahead calculations to determine effect of robot geometry on motion path.
5. Merge one move with another.

Servo systems, and thus robots, use a number of techniques to meet the above requirements. The two basic requirements are to determine how fast a servo motor is spinning and how far it has spun. In order to determine these two pieces of information, we need time information and distance information.

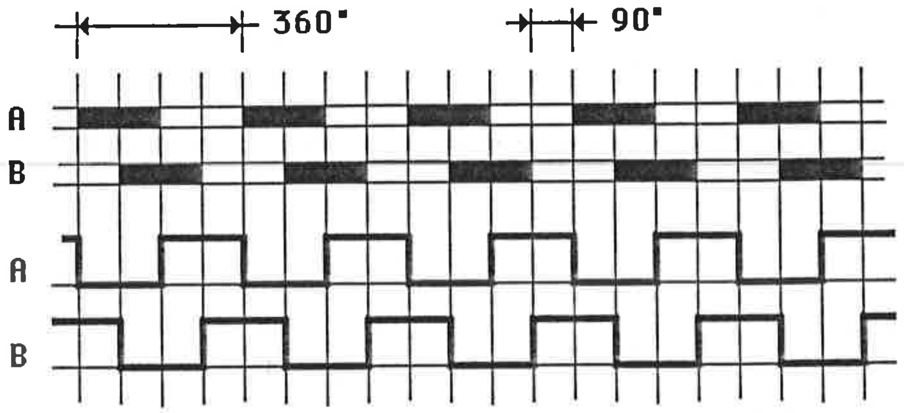
Time signals are generated within the robot/controller system by an internal clock that runs the microprocessors. The distance information is normally obtained from encoders that are attached to the mechanical robot system. The information is combined in the processor to determine the speed of the system.

ENCODERS

In general there are three types of encoders in use for obtaining positional information:

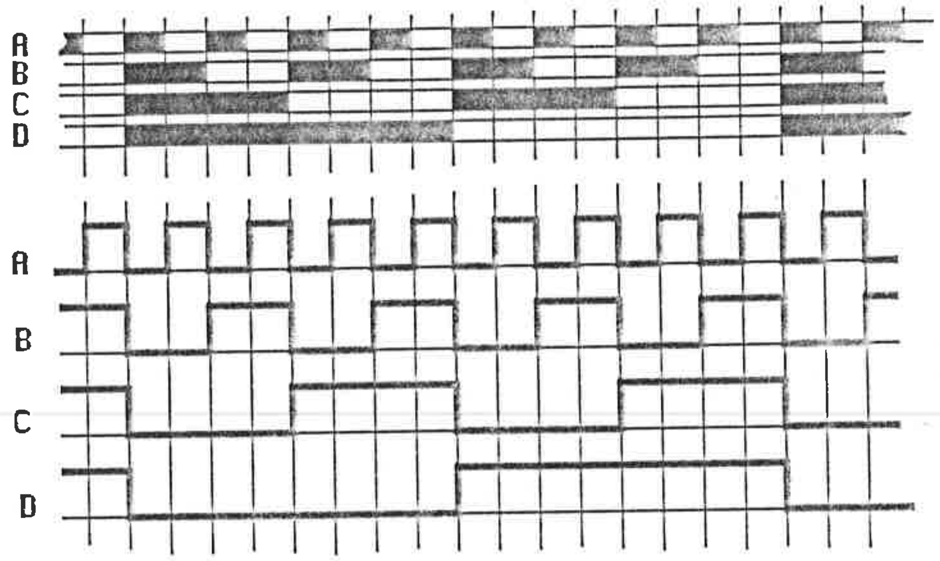
1. Incremental encoders
2. Absolute encoders
3. Combination of the above.

Incremental Encoders: The simplest encoder is the incremental encoder. This encoder consists of a series of transparent and opaque windows and a couple of sensors that provide a two phase signal as the encoder moves. The two phases "A" and "B" are 90 degrees out of phase with one another. When "A" leads "B", the motor is turning in one direction and when "B" leads "A", the motor-encoder system is turning in the other direction. Thus we have a way of determining direction. The actual cycles that one signal goes through over a period of time determines how far the encoder has moved. Incremental encoders can provide up to 1,000 pulses per revolution.



INCREMENTAL ENCODER SIGNALS

Absolute encoders: When the absolute position of a shaft is needed, an absolute encoder is used. Absolute encoders have a series of rings on the encoder plate with each ring having graduations that are twice a fine as the one inside it. The system has the advantage of giving the user the position of the shaft at all times without any calculations or doing a home routine. Absolute encoders with upto 16 rings of binary coding are available. These encoders can give the position of the shaft to within 1/63,536th of a revolution (1 part in 2^16).



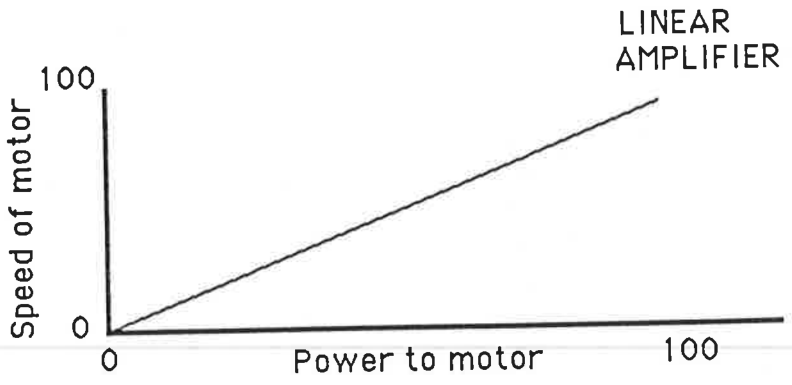
ABSOLUTE ENCODER SIGNALS

Combination units: Combination encoding schemes consist of using a coarse absolute encoder and then keeping track of how many times the unit has gone around. They provide the advantage of not having to look at an incremental encoder at all times. (ie less frequently without missing a count)

These three encoder schemes all provide positional information. That is how we determine distance. By dividing the distance by the time we get the speed of the system. By making the calculation often, we can keep close tabs on the system.

Now that we have a way of knowing how fast our servo is running and how far it has traveled, we need some way of controlling its speed. The speed of a motor can be controlled by changing the amount of energy that is sent to the system over a period of time.

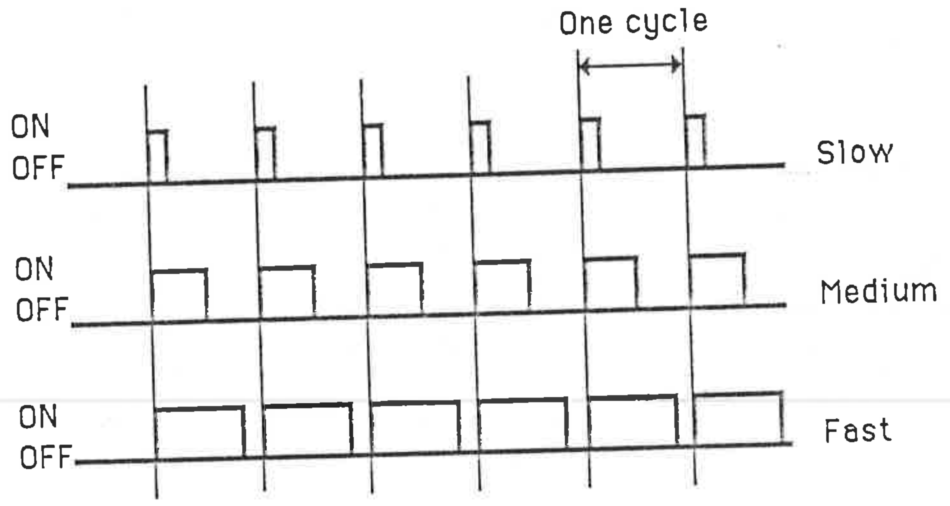
There are two basic ways to change the amount of energy being sent to the motor. We can do it either digitally or with a linear analog scheme. Linear modulation means that we will change the actual voltage that we apply to the terminals of the motor. The higher the voltage, the faster the motor will



turn.

Digital control usually means that we will apply a relatively fixed voltage to the system, but we will vary how long we apply the voltage to the motor terminals during an interval of time. Thus, if we wanted the motor to run at about half speed, we would apply power to it about half the time and turn the motor off the other half of the time. We would vary the width of the pulse that determines how much power we were applying. This is called **pulse width modulation.** Most modern servos, that are controlled by computers, use a pulse width modulation scheme (PWM) to control the energy to the motors.

When the power to a motor is applied and turned off rapidly, noise can be generated by loose parts in the motor. The magnetic fields move the loose parts back and forth. Most human beings can hear sounds that are at from 20 to 20,000 Hertz. In order to avoid this problem, most modulation schemes use a frequency that is between 14,000 and 20,000 Hertz. That means that the power to the motor is turned on and off 20,000 times during every second.

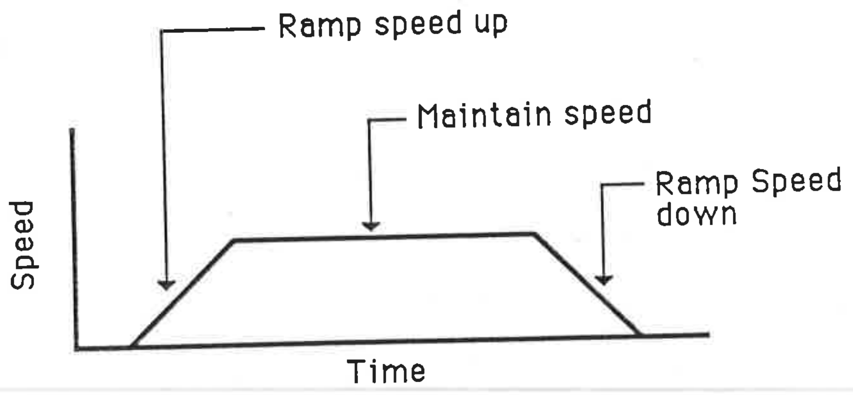


PULSE WIDTH MODULATION

In the computer, the firmware programmer creates a free running counter that takes about 1/20,000th of a second (50 microseconds) to go from 0 to its full value. The program compares the value in the counter to a constant in memory; if the value in the counter is higher than or equal to the constant, the power is turned off, if it is lower, the power is turned on. The energy to the motor is thus controlled by and directly proportional to the value of this constant.

As an example, let us assume that the counter counts from 0 to 255 over and over again at all times. Let us further assume that the constant has been set to 127. As the counter goes from 0 to 127, the power will be on. As soon as the counter reaches 128, the power will turn off. It will remain off till the counter reaches 255. As soon as the counter resets to 0, the power will come on again. This will give us a 50% duty cycle and will run the motor at about half speed. If we set the constant to 0, the power will remain off at all times because the count will never be less than the constant. If we set the constant at 255, the power will be on almost all the time. The counter will be less than 255 at all times except when the counter is actually at 255.

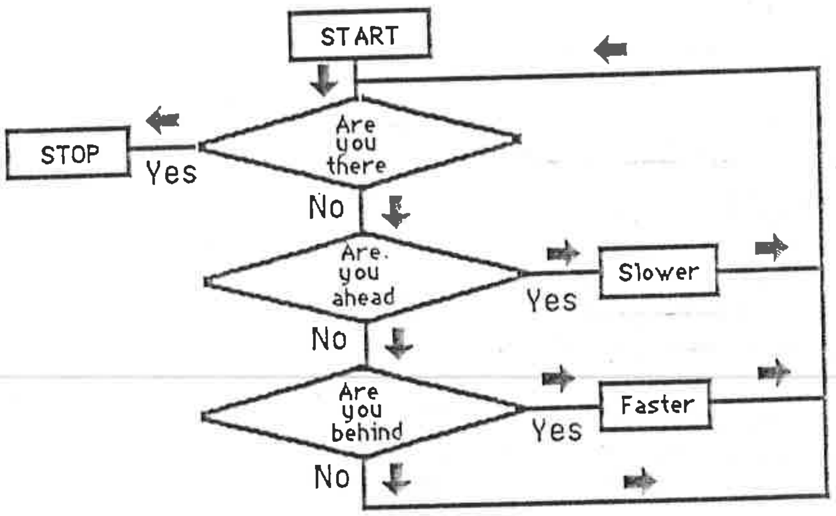
The circuits that are used to compare the values and switch the power are not within the scope of this manual. The intent of the manual is to give you an understanding of how these controls are implemented. You need to understand that there is a way that a number can control the power going to a motor. In the above example there are 256 (from 0 to 255) power settings for the motor. Each would give a free running motor a slightly different speed. The computer uses the numbers (settings) in the register to control the motor.



SPEED/TIME RELATIONSHIP

The most usual mode of operation for a motor is to first accelerate the motor uniformly till it reaches its target speed. Once the motor is at it's target speed, the computer maintains the speed till it is within a specified distance from it's target destination. The motor then slows down or decelerates so that it can stop at its destination. The acceleration and deceleration portions of the movement are said to be ramped. Ramping the motors allows them to start and stop without a jerking action. The ability to maintain a target speed allows all motors to start and stop each move at the same time.

The computer controls the motor speed at any time by estimating the distance the motor should have travelled in the allocated time. If the motor has travelled too far, the computer decreases the power to the motor by decreasing the number in the speed register. If the motor is falling behind, the computer increases the power to the motor by increasing the number in the speed register. Since the computer can check the motor position hundreds of times a second, very precise control can be maintained.



SIMPLIFIED LOGIC DIAGRAM

Other control functions can be tied into the control of the motor. The following examples show some typical applications:

1. If the speed register is set to full power (255) and the motor still cannot maintain its position, an overload condition is signaled. The robot stops. The error condition is annunciated.

2. If an encoder is not moving at all, a stall has occurred and the robot is stopped. The stall in annunciated.

Industrial robots usually use one microprocessor to control each motor. The Rhino XR system uses one microprocessor to control the entire system which consists of eight motors, the I/O and the communications. As a consequence of this, the XR microprocessor cannot perform all the functions that the more sophisticated industrial systems can provide. However, a limited ramping function is provided on all motors on the XR system. The motors start at full speed. Ramping occurs only as the motor is stopping. This allows the motors to come to a stop more effectively. Speed control is not implemented on the XR system.

THE USE OF ENCODERS ON THE XR SYSTEM.

The encoders used on the XR system are incremental encoders. Each consist 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.

$$\begin{tikzcd}
 & \boxed{00} \arrow[dl, shift left] \arrow[dr, shift left] & \\
\boxed{01} \arrow[ur, shift left] \arrow[dr, shift left] & & \boxed{10} \arrow[ul, shift left] \arrow[dl, shift left] \\
 & \boxed{11} \arrow[ul, shift left] \arrow[ur, shift left] &
\end{tikzcd}$$

ENCODER STATES

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 of the diagram. If we picked four positions, we would change the encoder count every time the state changed.

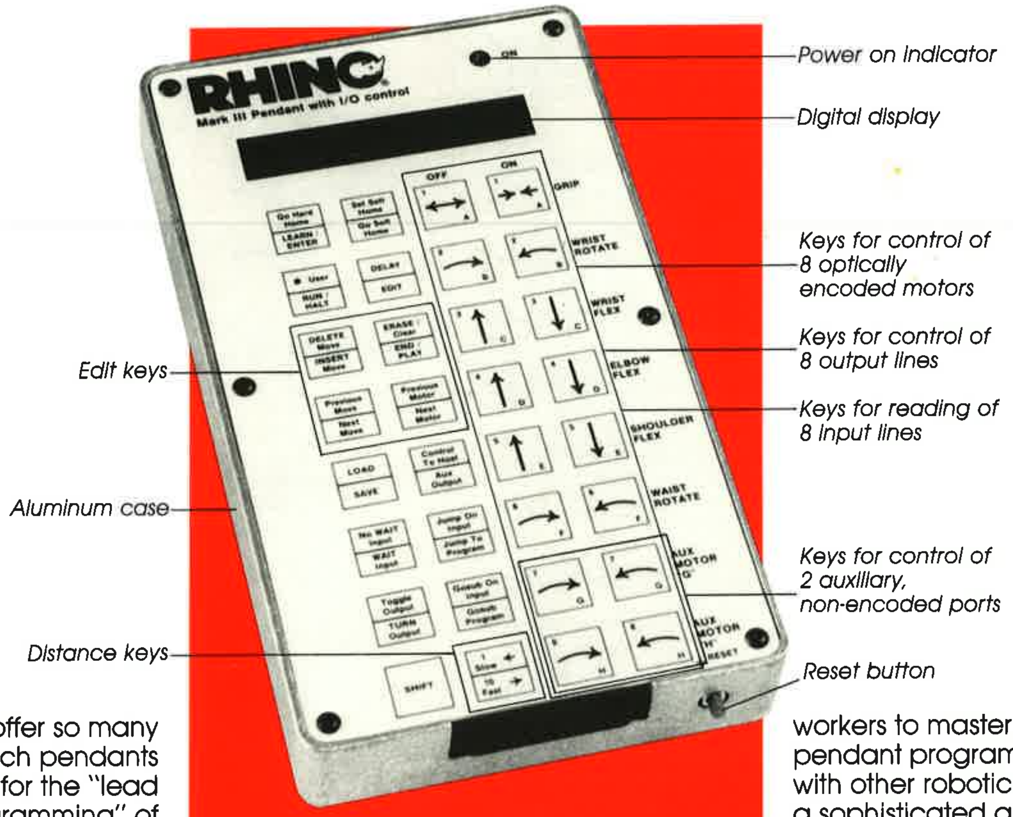
The controller uses eight registers in its memory as error registers for the motors. As long as a register is zero, the motor has no power applied to it. If an encoder turns, the register is added to or subtracted from as needed. As soon as the controller sees a number in the error register, it applies 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 make that error register go to zero. Each cycle of the encoder changes the error register by one. When the error goes 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 XR-3 robot, the large motors have 6 detectable states per motor revolution and the small motors have 3 states per revolution.

XR-3 Series

Teach Pendant


Power on indicator
Digital display
Keys for control of 8 optically encoded motors
Keys for control of 8 output lines
Keys for reading of 8 input lines
Keys for control of 2 auxiliary, non-encoded ports
Reset button
Edit keys
Aluminum case
Distance keys
RHINO
Mark III Pendant with I/O control

Because they offer so many advantages, teach pendants are widely used for the "lead through programming" of robots. One major advantage is the ease of use of the teach pendant as compared to any other form of programming. With a teach pendant, an operator can control a robot without having any knowledge of a formal programming language. This makes teach pendant use ideal for groups such as maintenance and repair personnel, beginning robotic students, and robotic operators who do not necessarily need to know how to program a robot. Rhino's Mark III teach pendant enables students and workers to master teach pendant programming along with other robotic concepts. It is a sophisticated and powerful device that enhances the instructional potential of the XR Series robotic system.

This latest version of the teach pendant is designed to provide Rhino users with exciting new functions, allowing full control of the I/O of the Mark III controller for varied applications. Many new commands have been added, so that all the keys, (with the exception of the shift key), now serve multiple functions. In addition, the number and variety of displays has been increased to give the user additional feedback.

RHINO®

ROBOTS, INC.

world leader in instructional robotics

Specifications

XR-3 SERIES TEACH PENDANT

The Rhino Mark III teach pendant has its own stand-alone computer located in a separate case or built into the XR controller as a combination unit. This hand held input device has many of the features of its industrial counterparts and additional features that are specifically designed for instructional purposes. Pictographs on the motor move keys are immediately understandable and help to introduce newcomers without prior programming experience to basic principles of robot operation. These and other features make the Rhino Mark III teach pendant the premier instructional model on the market today.

FEATURES

TRAINING TRANSFERS EASILY — Operation of the teach pendant is very similar to that of industrial teach pendants so that training will transfer easily.

I/O CAPABILITIES — Using the new teach pendant, Rhino users can control the I/O of the Mark III controller for extensive workcell experimentation. The teach pendant operator can control 8 ports for encoded motors, 2 ports for non-encoded motors, responds to 8 additional inputs, and controls 8 output lines.

VERSATILE — The Mark III teach pendant, with its I/O capabilities, is designed for a variety of applications. Because it enables students to operate a robot without getting involved in the complexities of off-line programming, it is an ideal introduction to robotics. Students may use the teach pendant to develop complex move strategies. It may also be used in conjunction with Rhino's RoboTalk™ or Rhino-VAL* software, to enter points into computer programs. These capabilities make it exceptionally useful for workcell development.

WIDE ARRAY OF EDITING FEATURES — Using the Mark III teach pendant, Rhino users can edit programs to delete moves, insert moves and step forward or backward through a program.

EASY TO USE — The many user friendly features of the teach pendant allow anyone to learn how to perform very basic move strategies after just a few minutes of instruction. Since the computer in the teach pendant is transparent to the user, operators do not need to know how to use a computer in order to operate the robot with the teach pendant.

SOPHISTICATED — Sharing many of the characteristics of a true industrial pendant, the Mark III teach pendant is the most complete educational teach pendant on the market today. Possibilities for experimentation with the teach pendant and its I/O capabilities are virtually endless.

DIGITAL DISPLAY — The 7 segment LED's on the Mark III teach pendant provide constant feedback to the student, emulating the feedback generally available on an industrial teach pendant. The display presents continuous updates on each motor by encoder counts. The display also provides helpful messages such as reminding the operator to do a hard home before beginning a learned move sequence, informing the operator if a motor has stalled during a move, and warning the operator when the move buffer is almost full.

COMPREHENSIVE MANUAL — The teach pendant manual is ideal for classroom use. It contains a lengthy section on the strategies and principles of teach pendant programming along with tutorial examples and a full discussion of teach pendant capabilities and feedback displays.

* VAL is a trademark of the Unimation Corporation.

Rhino has representatives throughout the USA, Canada and a large number of other countries. Please call or write for the name of the representative in your area.

Model	Mark III Teach Pendant
Part Number	FG1069 (with circuitry built into Mark III controller) and FG0649 (with circuitry in separate cabinet)
Construction	Machined from solid aluminum for strength without weight, hand held case is 4" x 7" x 1" (101.6 mm x 177.8 mm x 25.4 mm)
System	6502 microprocessor based
Power	110/220 volts, 50/60 Hz
Touch Panel	32 pressure sensitive keys, motor move keys with pictographs.
Display	Real time digital display. 7 segment LEDs display up to 7 characters simultaneously.
Continuous Feedback Features	Mode, motor, direction of movement, number of motor counts — current motor, current move, number of current move in move sequence, special messages, e.g. do hard home, buffer full.
Modes of Operation	PLAY for manual mode LEARN for programming a move sequence EDIT for modifying a learned move sequence RUN for executing a move sequence
Capacity	149 moves of 8 motors plus I/O
Special I/O and Program	AUX OUTPUT — Controls the 2 auxiliary motor ports (on/off).
Control Functions	JUMP ON INPUT — Directs the program to a given program sequence when or if an input occurs. (conditional jump). UNCONDITIONAL JUMP — Directs the system to a given program segment. WAIT INPUT — Directs the robot to WAIT for a given input signal before executing the next step. NO WAIT INPUT — Allows pulsing of the outputs of the Mark III controller. TOGGLE OUTPUT — Allows the pulsing of the outputs of the Mark III controller. TURN OUTPUT — Controls the level of the outputs of the Mark III controller (high/low). END — Ends the present mode, when in the LEARN, EDIT, and STALL modes. In the STALL mode, the END mode acknowledges the STALL message and takes you out of the stall trap.
Other Features	Includes upload/download software for Apple IIe and IBM PC on diskette. Teach pendant manual.

Rhino Robots Inc. reserves the right to change any and all specifications and prices without prior notice.

Rhino Robots, Inc.
308 South State Street
P.O. Box 4010
Champaign, Illinois 61820 USA
Telephone 217-352-8485
TELEX: 3734731 RHINO ROBOTS C

RHINO® ROBOTS, INC.

world leader in instructional robotics

ENGINEERED AND MANUFACTURED IN THE USA

MARK III
TEACH PENDANT

OWNER'S MANUAL

Version 3.00 September 1st, 1986

RHINO ROBOTS INC.

Manual Part Number
65-60-872

Authors
H. S. Sandhu
Carl Phillips
Tom Hendrickson

Documentation Services,
Rhino Robots Inc.
P. O. Box 4010,
Champaign, Illinois. 61820
U. S. of America
Tel: 217-352-8485.
Telex: 3734731 RHINO ROBOTS C

CREDITS

The hardware for the teach pendant was designed by Carl A. Phillips, BS Electrical Engineering, University of Illinois.

The firmware in the teach pendant was designed and written by Carl Phillips, BS University of Illinois. Tom Hendrickson helped with the firmware.

Tom Hendrickson checked the software and helped draft parts of the manual.

Harprit S. Sandhu wrote the manual.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Teach pendants are widely used for the "lead through" programming of robots because they offer many advantages. Prime among these is the ease of use of the teach pendant as compared to any other form of programming. All teach pendants are similar and usually use a readily understood pictorial language that can be mastered within a short time. Programming from a host computer often requires familiarity with manufacturer specific robotic languages such as VAL™ or Karel™. Each language contains many different commands and requires considerable study before an operator can become familiar enough to control a robot with ease.

With a teach pendant, an operator can control a robot without having any knowledge of a formal programming language. For example, a company wishing to automate a painting operation, could easily train an experienced painter to set up a move sequence with a teach pendant even though the painter may not be able to program the robot with its robotic language. On the other hand, a competent computer programmer may not know enough about painting to handle a complicated painting assignment even though he is very familiar with the language of the robot. Usually the skills of both individuals are necessary to do the job well.

Another important group that makes heavy use of teach pendants are maintenance and repair personnel. They need to conduct extensive tests of robot motion but do not necessarily need to create a complex move sequence from a host computer requiring formal programming skills. With a teach pendant, they can exercise all the axes and conduct most trouble shooting operations without being trained programmers.

Another advantage of a teach pendant system is that the pendant computer is transparent to the user. Since many people are still fearful of computers, it is often helpful to introduce them to robot operations without discussing the computer that is part of the teach pendant system. The Rhino Robots teach pendant system includes an 8-bit microprocessor but the user does not need to be aware of its presence.

The Rhino teach pendant is the ideal tool for a rapid, hands-on introduction to the XR robotic system in particular and to robots in general. With the teach pendant, novices are able to begin operating the robot after just a few minutes of preliminary instruction.

THE RHINO ROBOTS TEACH PENDANT SYSTEM

The Rhino Robots teach pendant system consists of the following major components:

1. A 32 key, hand held pendant in a metal case. The pendant is provided with a cable for connecting to the teach pendant computer.

2. A microprocessor based computer card that forms the heart of the teach pendant system. This can be mounted either in a separate cabinet or in a combination cabinet together with the XR controller.

3. A software disk containing the SAVE/LOAD program for saving and recalling move sequences to/from a host computer system.

4. This manual.

The SAVE/LOAD program is described in detail in Appendix II.

In addition to the above, you should also have the following items:

1. A cable to connect the teach pendant control computer (not the hand held part) to the robot controller. If you have the Mark III combination controller with a built in teach pendant system, you will not need this cable since a built in switch provides this function.

SETTING UP TO USE THE TEACH PENDANT

Unpack the teach pendant carefully and make sure that you have every item listed in the previous section.

Check that the power is off before hooking up the teach pendant system. This is a good general rule whenever you are working with electronic components.

Set up the teach pendant system according to the configurations shown below.

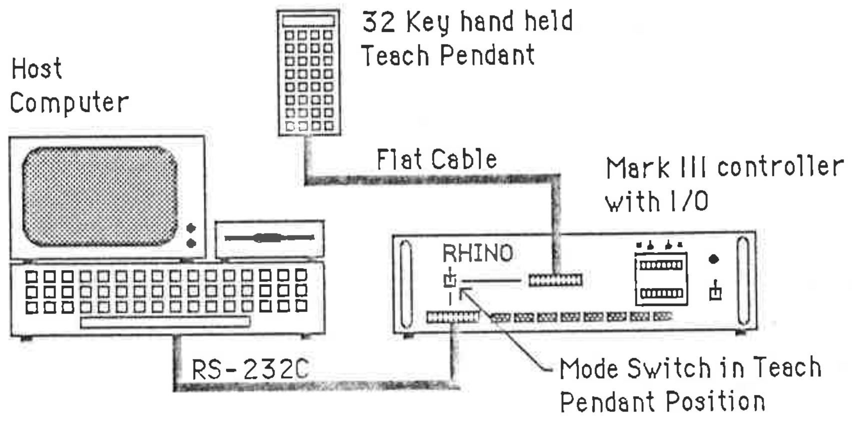

32 Key hand held Teach Pendant
Host Computer
Flat Cable
Mark III controller with I/O
RHINO
RS-232C
Mode Switch in Teach Pendant Position

FIGURE 1-1
Configuration of Mark III Combo Controller with I/O

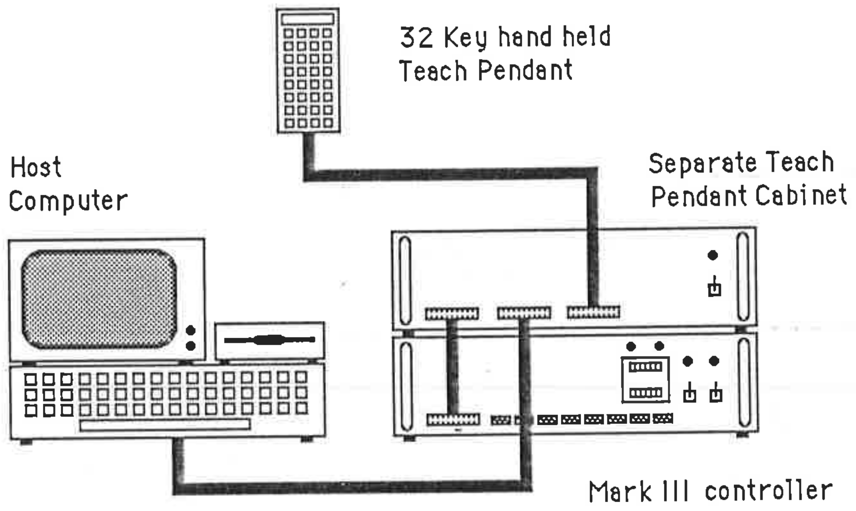


FIGURE 1-2
Mark III controller with separate Teach Pendant cabinet.

With the Mark III combo unit, you can keep both the teach pendant and host computer plugged in at all times. To change control from the teach pendant to the host computer simply flip the mode switch on the left hand side of the controller's front panel.

The teach pendant is a stand alone device. This means that it does not require a host computer to allow you to use the robot with it. However, if you want to save programs that you have "taught" the robot with the teach pendant, you will need a host computer to store the programs. Plug the host computer into the appropriate port of the controller. Use your disk with the SAVE/LOAD program (in the host computer) to save your teach pendant programs. The programs are saved on the disk of the host computer system.

THE TEACH PENDANT KEYBOARD

The teach pendant display panel is shown below.

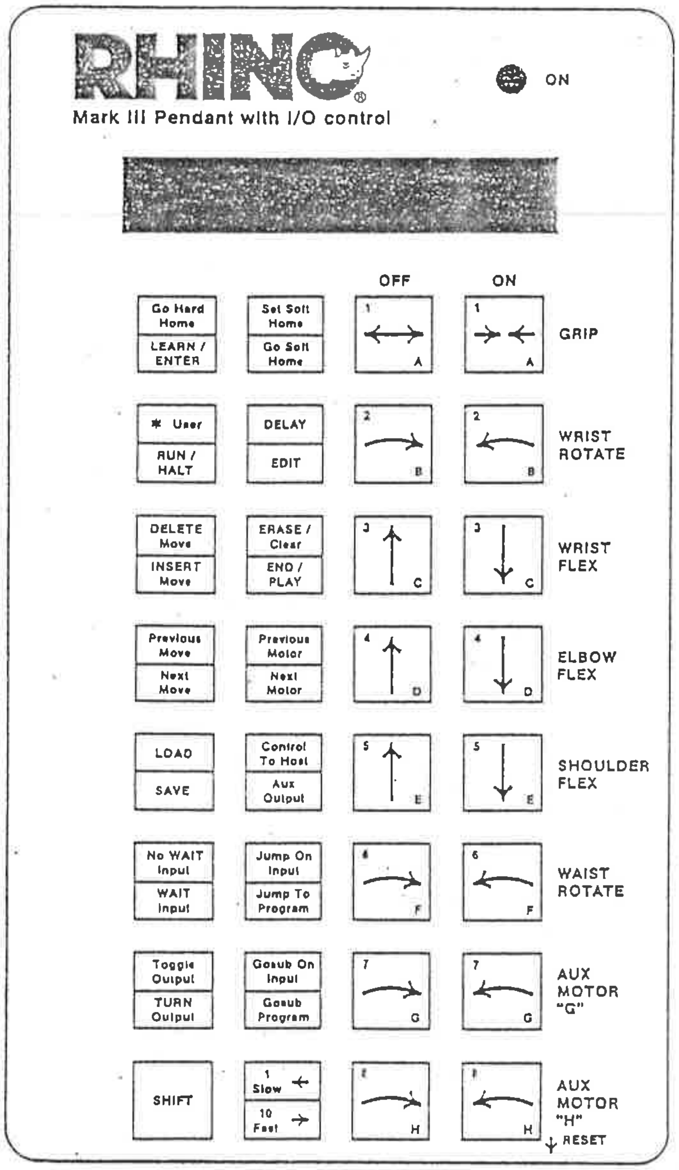


FIGURE 1-3
Teach Pendant Keyboard

The small, round light at the upper right corner of the teach pendant indicates that the power is on. Immediately below is a display consisting of seven 7-segment LEDs. This display provides information to the operator about what is happening in the system. As we describe the use of the teach pendant, we will tell you what the different displays mean. You will find the information in the displays especially helpful when you are editing a move sequence.

The 32 keys on the teach pendant keyboard can be divided into two parts. The two columns on the left (16 keys) are the **FUNCTION** keys. The two columns on the right (16 keys) are the **MOTOR MOVE** keys. However, notice that all keys except the SHIFT key serve more than one function.

The keyboard follows the structure of the XR robot, moving from the fingers to the waist. The keys correspond to the motors, designated A through H, on the controller. Specifically,

MOTOR	SERVICE
A	finger gripper.
B	wrist rotation.
C	wrist flex. (azimuthal)
D	elbow flex.
E	bicep flex.
F	waist rotation.
G	accessory
H	accessory

Keys that have a horizontal dividing bar in them are dual function keys. Pressing the key alone will activate the lower function. If the key is pressed while holding the shift key down, the upper function will be activated.

USING THE TEACH PENDANT

INITIAL DISPLAYS

Whenever you power up, the word **"init"** appears in the display momentarily. After the display blanks out, the gripper is closed and then set to it's fully open position. The system then displays the letter "P" at the left of the display. The system has now entered the PLAY mode. This process is repeated whenever the system is turned **on** or **reset**.

THE STRUCTURE OF THE TEACH PENDANT PROGRAM

The teach pendant has four modes of operation: PLAY, LEARN, EDIT, and RUN.

PLAY.	In PLAY you can try out various motor moves and get a feel for running a robot.
LEARN.	You may teach the robot a designated sequence of moves in the LEARN mode.
EDIT.	You use the EDIT mode to make modifications and fine tune the sequence you have taught.
RUN.	You run the taught cycle in the RUN mode.

You can save a program using the SAVE routine and reload a saved program with the LOAD command. (A host computer is needed for these functions). Finally, the teach pendant program has a special STALL routine which helps you to recover from a situation where the system is blocked from carrying out a designated move sequence. You can do this without losing the move sequence that the system was being taught.

Figure 1-4 below shows how the modes interact with each other. Notice that you always go through the PLAY mode when you switch from one mode to another.

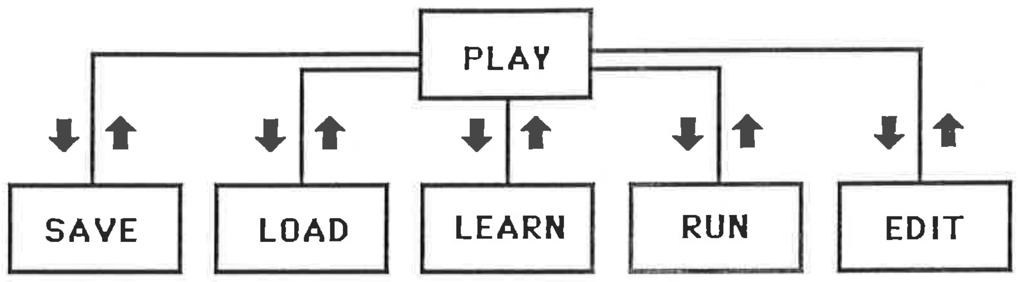


FIGURE 1-4
Mode transfer protocol

Figure 5 shows how the four main modes interact with the STALL TRAP routine. Except when the system is in RUN mode, it can always go back to its prior mode from a STALL. If a STALL occurs while you are in RUN mode, you have to go into PLAY mode before you can go back to RUN. This means that you must start your program from the beginning again.

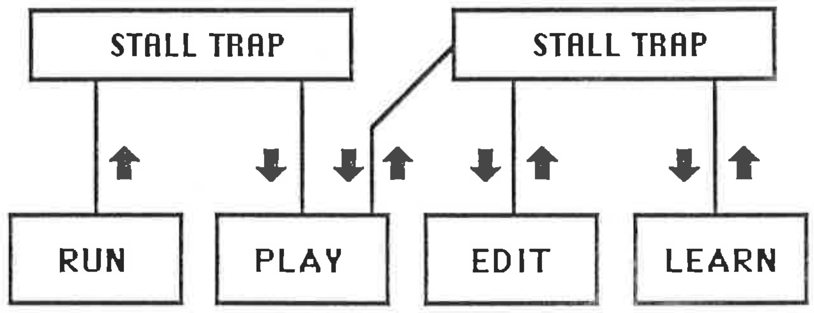


FIGURE 1-5
Stall trap interaction

SECTION 2

PLAY MODE

THE FUNCTION OF PLAY MODE

After you turn the system on and the "**init**" message disappears, the letter "**P**" will appear in the left digit of the display. This indicates that the system is now in the **PLAY** mode. PLAY mode is useful for practicing robot movements and positions. You can test the reach of the robot and find out how best to get to a designated location. This is the mode that you use to familiarize yourself with the robot and to work out operational schemes for the robot.

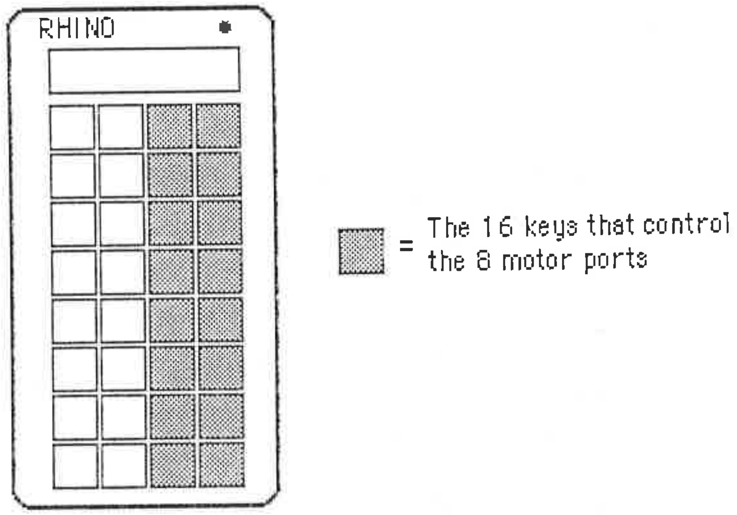


Figure 2-1
The 16 motor move keys

In the PLAY mode, all the motor move keys in the two right hand columns are active. By pressing any of these keys, you move the corresponding motor on the robot (or connected accessories). If a motor is not connected to a motor port, the teach pendant detects this condition upon power up and reset and automatically disables the two keys controlling that motor port. If a disabled key is pressed on the pendant, the pendant will display the "**OFF**" message.

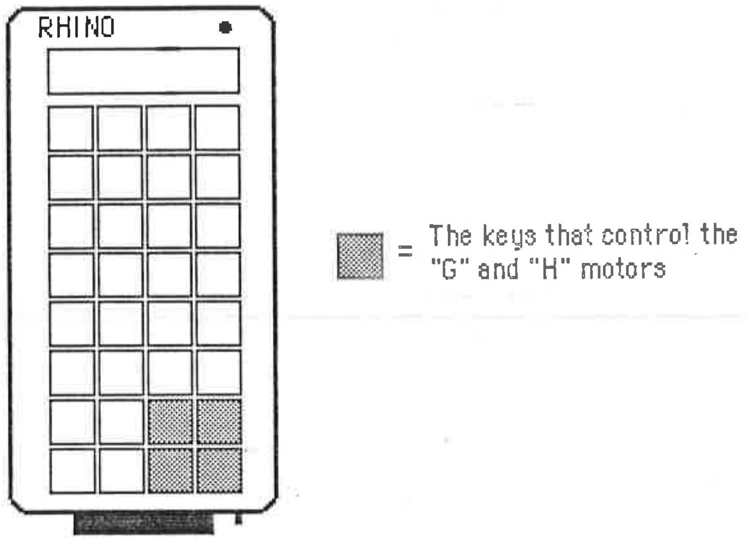


Figure 2-2
The "G" and "H" motor move keys

The four motor move keys at the bottom are used only if you have motors connected to ports "G" and "H" of the robot controller. Any of the motorized Rhino accessories such as the conveyor, carousel, slide base, or X-Y table, can be plugged into the "G" and/or "H" ports. Pressing the corresponding keys moves the accessories in the same way as the motors on the body of the XR robot. Keys for the "G" and "H" motors or any of the other motors that are not connected will put the "**OFF**" message in the display when they are pressed. This tells you that the motors are not active.

When you keep your finger pressed on a motor move key, the **AUTO REPEAT** function becomes active. The motor moves smoothly and continuously until you lift your finger from the key.

As you press the motor move keys, you will find the robot movement reflected on the keypad display. For instance, if you rotate the wrist joint (B motor) counter-clockwise 120 steps, the display will show Pb 0120 (assuming that the B motor started a 0 location); this means that the device is in PLAY mode and that the "B" motor has traveled 120 encoder counts in the positive direction. The display always shows **absolute** not relative position. The only exception is the gripper (end effector) which has only two settings, **open** or **closed**. This simulates the operation of a pneumatic gripper.

DISTANCE KEY

The key at the lower left of the pendant with the numbers 1 and 10 is used to set the distance moved by a motor with each key press. The 10 key means that a motor will move 10 encoder counts for each press of a motor move key. This is the most common setting and is the default value.

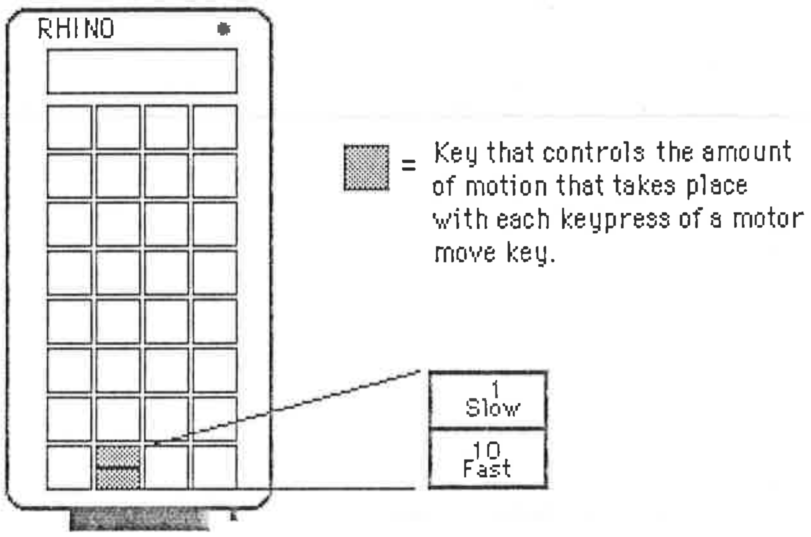

RHINO
= Key that controls the amount of motion that takes place with each keypress of a motor move key.
1 Slow
10 Fast

Figure 2-3
The distance key

The "1" setting is activated by pressing the "1" key while pressing the "SHIFT" key. Use the "1" key when you are fine tuning a move. For instance, you may want to use small increments together with the auto repeat feature to achieve exact placement of the robot gripper. The toggling of the motor can give the robot a slower, sometimes jerky action. It should be used any time you are within an inch or two of your pick up or release point.

On start up and after a RESET, the 10 key function is selected. This selection is suitable for most move functions and should be used most of the time.

Stalls: If, during a motor movement, the robot encounters an obstacle that prevents further movement, the teach pendant will display the "**Stall**" message. The system will automatically back away from the stall position. The system will clear the stall message when the motor move key is released.

LEARN MODE

PRELIMINARY STEPS

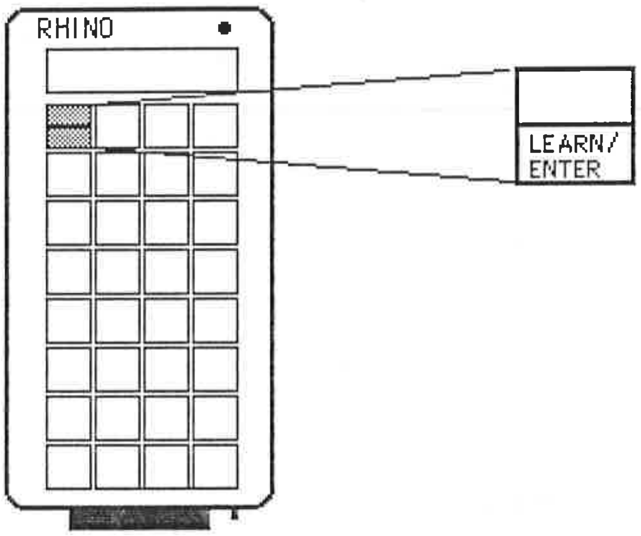


FIGURE 2-4
The LEARN Key

In the LEARN mode, the robot can be taught a set of moves that will be retained in memory. Two things must be done before you enter the LEARN mode: CLEAR the memory and send the robot to its HARD HOME position.

NOTE: CLEAR memory is only needed if a program is already resident and you don't want to append to it. HARDHOME needs to be done only once.

To CLEAR the memory, push the SHIFT and ERASE/CLEAR key. Pressing the RESET button does **not** erase a program already in memory.

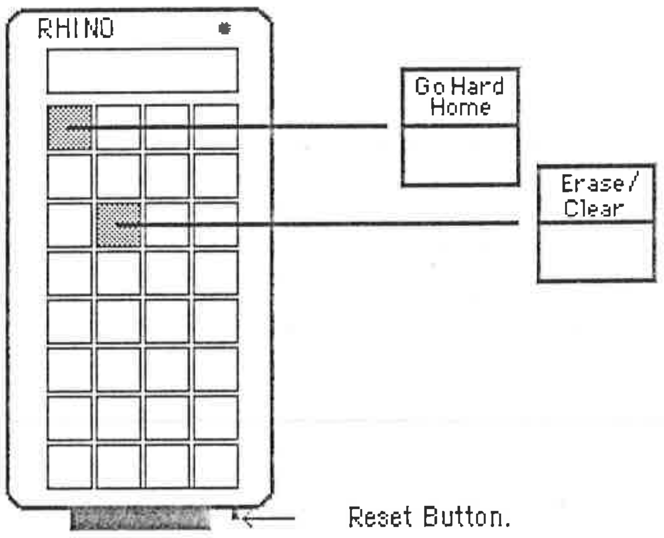


Figure 2-5
Keys used to initialize the robot/controller system
before entering the LEARN mode

A good hard home routine is one that successfully finds the B,C,D,E and F limit switches on the robot. If the robot is in an unusual position or encounters an obstacle, it might not be able to find hard home on the first try. If this happens, the display will show "**noHArd**" and you simply repeat the hard home request. (Note: Mark II users must manually home the B motor). SCARA users should see Appendix III for hard homing their robot.

"Hard" home means "Hardware" home. It is used for initializing the robot to a known position. After the GO HARD HOME command is given, the robot closes the gripper then moves to find the exact center of the microswitches on the "B", "C", "D", "E", and "F" motors and then fully opens the hand. These actions provide an absolute, repeatable position from which the robot can LEARN a move sequence. You must do a HARD HOME once after each RESET before the system will allow you to enter LEARN mode. In case you forget, the display will remind you by telling you to "**do HArd**".

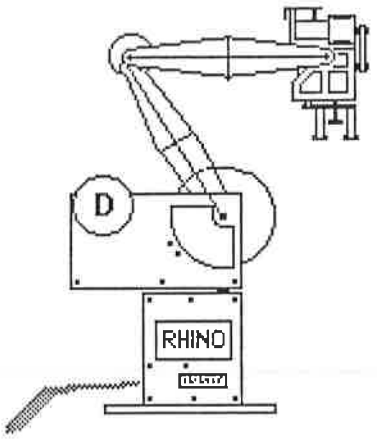


Figure 2-6
The "hard home" position for the robot.

SETTING THE "SOFT HOME" POSITION

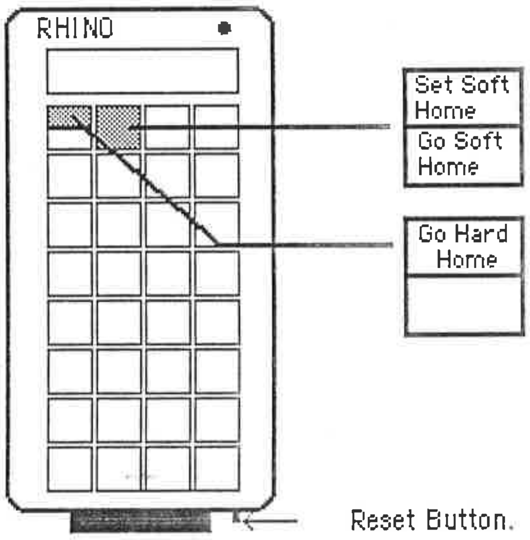


FIGURE 2-7
The SOFT and HARD HOME Keys

The Soft Home" position is a robot position that is a convenient starting position for creating a move sequence.

To establish a "Soft Home", or software home, move the robot into the desired configuration and press SET SOFT HOME (Press SHIFT and SET SOFT HOME). Thereafter, the robot will return to this position when ever the GO SOFT HOME key is pressed.

The "Soft Home" position will be retained until the power is turned off, a new soft home position is defined or the RESET key is pressed. From now on, when we refer to "Home", we will always mean the "Soft Home". The "Soft Home" is concerned with all eight axes, not just with the six that "Hard Home" refers to. On the "G" and "H" axes, the soft home position will be the position that they were in when the SET SOFT HOME key was pressed.

The use of "Soft Home" increases robot efficiency. It should be set at a location that is convenient to the initial "pick up" position, about two inches above it, in most cases. The "Soft Home" then becomes the actual starting point of the move sequence and the robot will not need to return all the way to its "Hard Home" position when it runs a move sequence.

The "Soft Home" position cannot be set unless the user has done a "Hard Home" first (once). If the user **chooses not to set** a "Soft Home" position, the robot will treat the "Hard Home" as the "Soft Home" and return to the "Hard Home" position whenever the GO SOFT HOME key is pressed. If you cannot remember whether you have created a "Soft Home" position or not, simply move the robot a few steps and then send it to SOFT HOME. If it goes to "Hard Home", it is assuming that "Soft Home" and "Hard Home" are the same position. Also, if the robot does not move when the GO SOFT HOME key is pressed, you can assume that it is already in its "Soft Home" position and the display will show "AtSoFt".

TEACHING SINGLE MOTOR MOVES

At this point you should be in the LEARN mode. If you do not have an **"L"** in the display, press the LEARN key. That will put you into the learn mode.

Single motor move sequences are good for initial training and practice. Very quickly you will be developing multiple motor move sequences, which are more like the real world.

For a single motor move, you simply press the key that corresponds to the motor you wish to move until the robot has reached the position you want. If you overshoot, press the key for the opposite direction until the robot is positioned correctly. Then, press ENTER to record the move. Any back and forth motions that you may have made as you position this motor will not be reflected in the recorded move. For instance, if you turned clockwise -1000 encoder positions on the waist, motor "F", and then moved back 470 encoder positions, your move would be registered as an aggregate of - 530.

After you ENTER the "F" motor move, your display changes. When you began, you had "L" on the left for LEARN and "0" on the right because no motor moves had yet been entered. After the waist rotation, your display will read

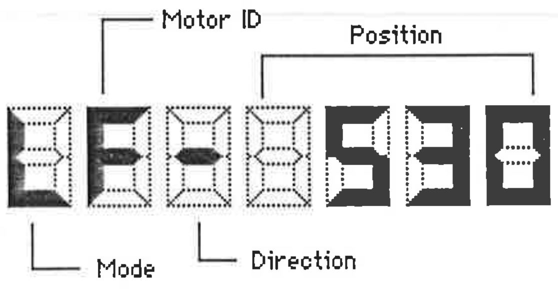


FIGURE 2-8
Learn, F motor, -530 position

Now when you record this move by pressing the ENTER key, the display will change to:

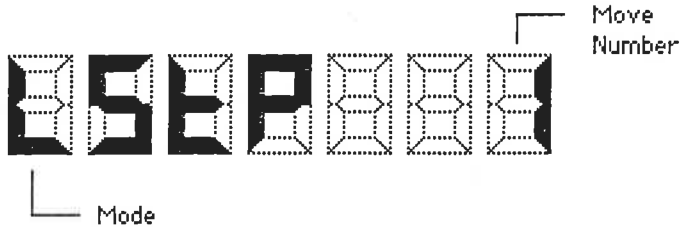


FIGURE 2-9
Learn, Stored position 1

NOTE: While the Enter key is being pressed, the display will show LEnt # (where # is the move number being recorded).

If this was the only movement you wished for your sequence, you could complete the cycle by pressing the GO SOFT HOME key, which sends the robot to the "Soft Home" position. If you did this, the display would change to: