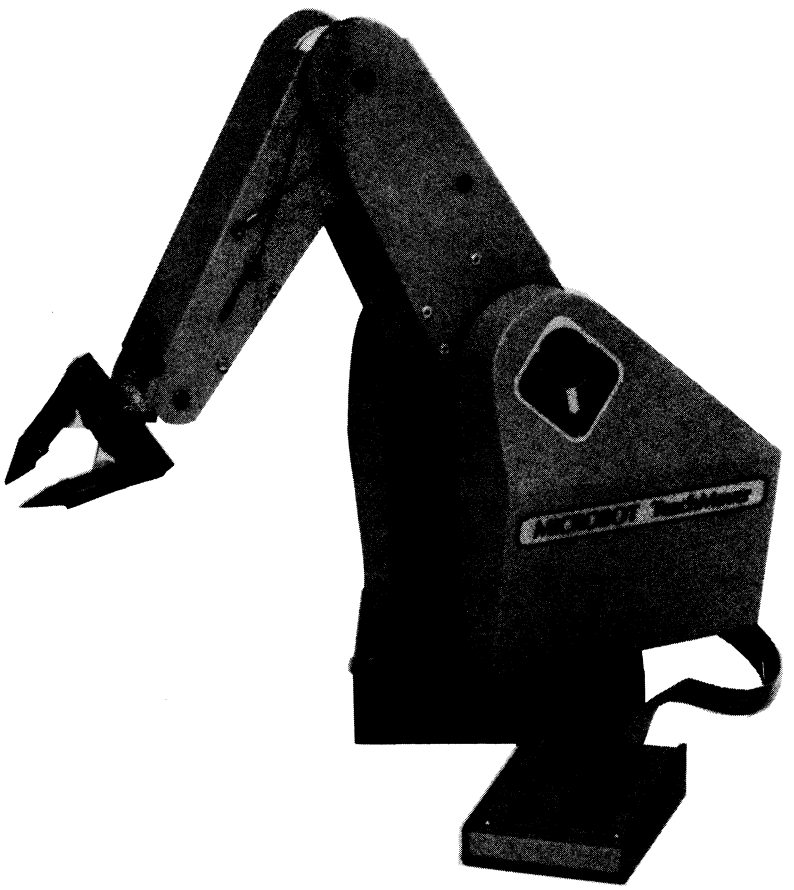
Operation of the Five-Axis Robot

Model TCM



MICROBOT

Microbot, Inc.
453-H Ravendale Drive
Mountain View, CA 94043

Part Number RRA-2

Edition 2

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

TABLE OF ILLUSTRATIONS

TABLE OF ILLUSTRATIONS

TABLE OF ILLUSTRATIONS

TABLE OF ILLUSTRATIONS

CHAPTER ONE
INTRODUCTION

The TeachMover robot arm is a microprocessor-controlled, six-jointed mechanical arm designed to provide an unusual combination of dexterity and low cost. The TeachMover arm can be used in either of the following modes:

Two operating modes

- Teach Control Mode , in which the hand-held teach control can be used to teach, edit, and run a variety of manipulation programs.
- Serial Interface Mode , in which the TeachMover arm can be controlled by a host computer or a computer terminal via one of two built-in RS-232C asynchronous serial communications lines.

The TeachMover arm can be used for an unlimited number of applications, including:

A myriad of applications

- Education in areas such as manipulation algorithms, coordinate transformations, planning strategies, and computer language development.
- Industrial Automation for light assembly, kitting of parts, palletizing, and operation of equipment and tools.
- Enjoyment by playing games such as chess, drawing with felt-tip pens or paints, and assembling structures with Lincoln Logs, blocks, etc.
- Experimentation with computer vision, electromechanical sensors, mobile robots, and more.

Overview Of Manual

Chapter-by-chapter overview

Chapter 2 of this manual tells you how to get started using the TeachMover. The unit's mechanical & electrical construction are explained in chapters 3-5. (It is important to understand this material in order to use the TeachMover properly and to full advantage.) Chapters 6 & 7 explain all the details of operation of teach control mode and serial interface mode, respectively. Suggested advanced applications are given in Chapter 8.

We have also incorporated a number of useful appendices. These include: a brief history of robotics, the mathematics of coordinate conversions, TeachMover electronic and cabling details, instructions for expanding the memory available for TeachMover programs, convenient reference tables, and a grid diagram to define the robot arm starting position.

Finally, we have provided a suggested list of references on robotics and artificial intelligence for those who wish to delve more deeply into this exciting field.

BACKGROUND

Developmental work with computer-controlled manipulators was limited, at first, to laboratories at a few research-oriented universities in the USA and overseas. Principal advances have come from artificial intelligence laboratories, particularly at Stanford University and the Massachusetts Institute of Technology. Work has concentrated on manipulation languages, manipulation planning strategies, coordinate transformations, use of sensors, and manipulator kinematics. Much of this work is highly mathematical and has been performed on large sophisticated computing systems.

Industry has applied a great deal of this research to factory situations, and as a result thousands of robot

arms now help to manufacture products we use daily. Industrial robots spot-weld automobile bodies, feed material into punch-presses, spray-paint metal & plastic components, empty injection-molding machines, and perform many other factory jobs [19]. Principal manufacturers of industrial robots include Unimation and Cincinnati Milacron in the USA, ASEA in Sweden, and several companies in Japan. Unfortunately, the price range of most industrial robots - $35,000 to $120,000 - hinders robot experimentation by individuals and schools, as do the carefully guarded, proprietary hardware and software details of these machines.

Microbot's contribution

Since 1979, Microbot, Inc. has been developing low-cost manipulating systems for education, industrial evaluation, and actual industrial use. Our objectives with the TeachMover arm have been to provide a complete self-contained system, incuding the manipulator, on-board computer, hand-held teach control, host computer interface, and the control language, so that the user would not have to undertake a major development effort or purchase a special computer in order to gain a sound working knowledge of robot operation.

CHAPTER TWO
GETTING STARTED

Minimum but necessary details

If you have just received your TeachMover, you probably will want to start operating it right away. You can - but there are a few simple things you must do first. This chapter will give you the minimum information you need to quickly get started using and programming the TeachMover arm. After you've completed this chapter, be sure to read chapters 3-5 (on mechanical & electrical construction of the arm) before proceeding to chapter 6, where all the teach control operating instructions are given in detail. Proper utilization of all the arm commands requires an understanding of how the arm itself is constructed.

A. MECHANICAL CHECK-OUT

Carefully unpack the TeachMover robot arm and place it on a convenient working surface. Visually inspect the arm, looking for any obvious signs of shipping damage. Remove any packaging material that has become lodged in the arm. Save the carton in case you ever need to ship your TeachMover. (Appendix E shows how to repack the unit properly.)

Learning the names of the arm members and joints

Refer to Figure 2-2 to learn the names of the five arm members and six joints. Now, operate the joints by manually turning each of the six large plastic drive gears which are accessible at the rear of the arm. The drive gears should turn with the application of a moderate force, and each of the joints should move. Now, hold the body and arm members, and move each member manually. There should be little play and the arm members should offer only moderate resistance (about 1 pound) to movement.

Check to see that the hand cable correctly runs over the shoulder idler pulley, the sense bracket pulley,and the elbow idler pulley (Figure 2-3); if the cable has slipped off any of these pulleys, simply replace it, and, if necessary, turn the drive gear to provide enough cable.

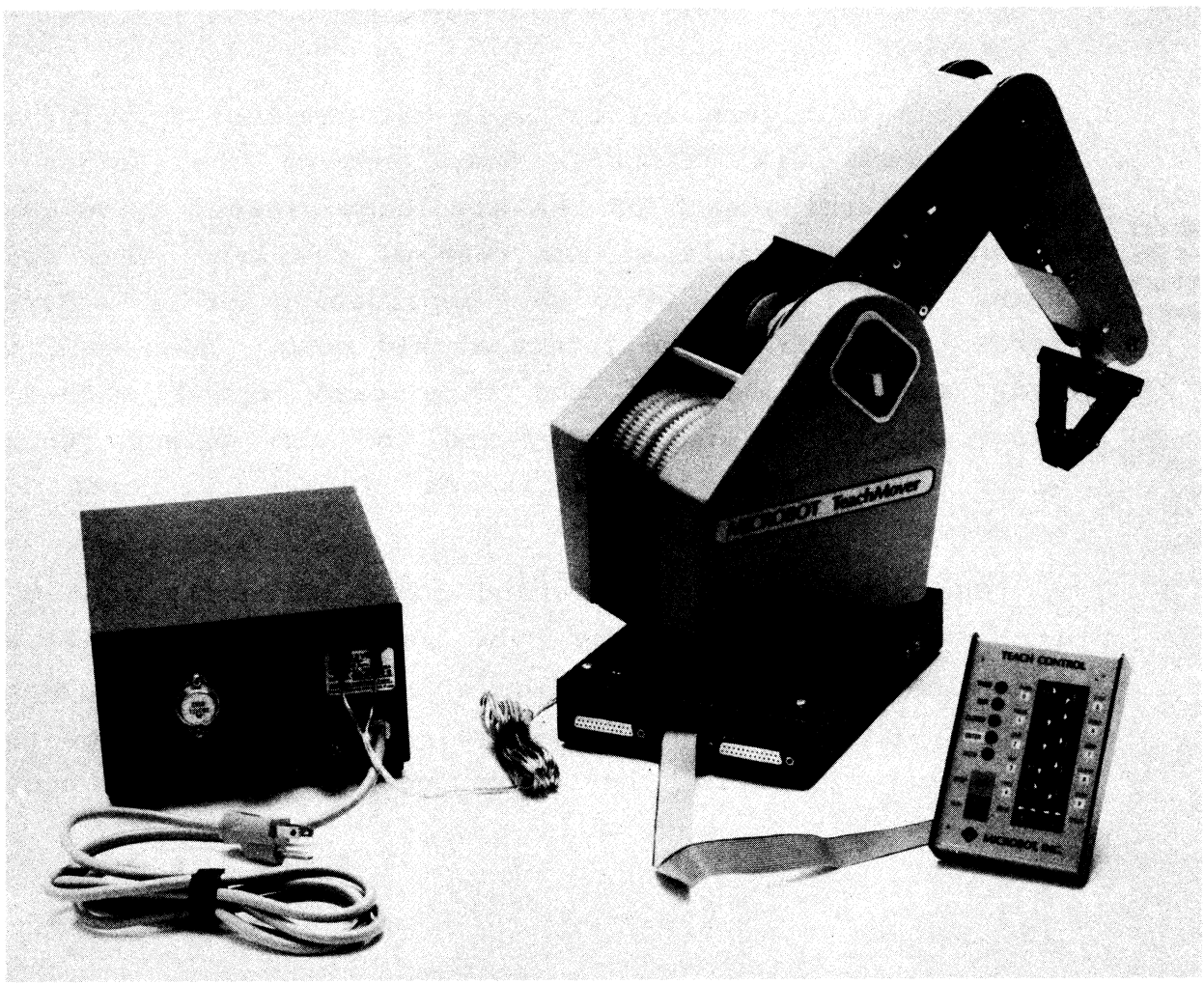


Figure 2-1 TeachMover Robot Arm with Teach Control and Power Supply

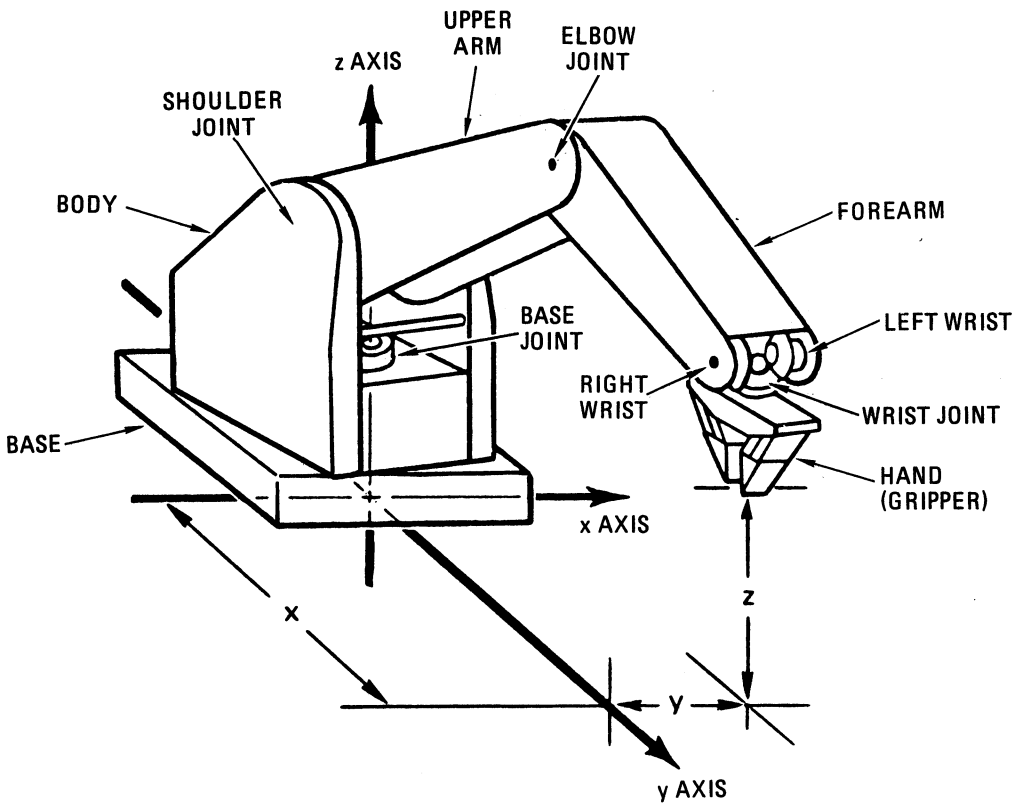


Figure 2-2 Major Structural Components

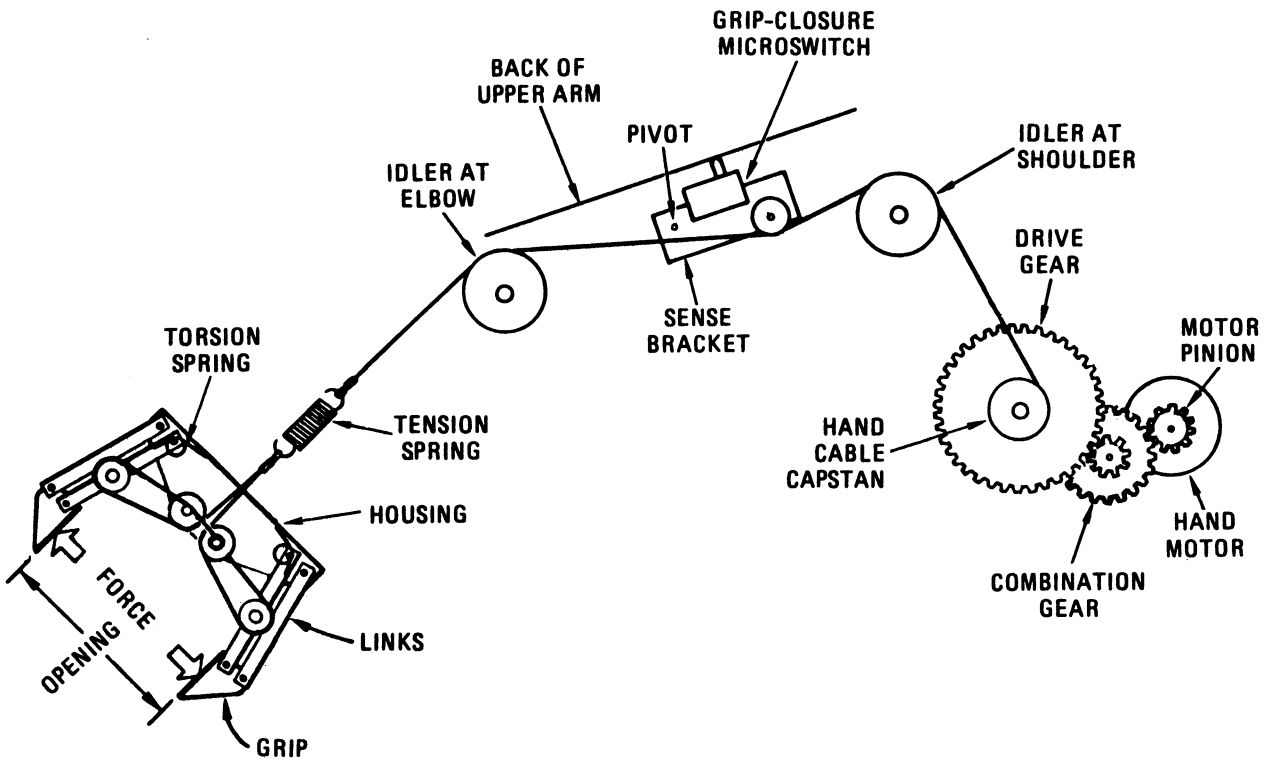


Figure 2-3 Hand Cabling Diagram

B. CONNECTING THE POWER SUPPLY

Connect the power cord at the back of the base of the arm to the power supply unit.

```
**************************************
*             IMPORTANT!             *
*    TERMINAL WITH RED TAG MUST      *
*  BE CONNECTED TO POSITIVE VOLTAGE  *
**************************************
```

Using your own power supply instead

The TeachMover requires a 12-volt DC power supply capable of at least 4.5 amperes with a ripple (peak-to-peak) voltage of 1 volt or less. Battery eliminator power supplies (providing 13.8 volts) can be used if higher arm speeds and lifting capacities are required; these power supplies make the TeachMover motors hot to the touch (120^{o}F), but do not cause damage.

C. PLUGGING IN THE ARM

The three-prong plug is used to avoid shock hazard and should be plugged into a grounded, three-prong socket only.

If a three-prong socket is unavailable, then use a two-prong adapter, and ground the third lead by wiring it to a pipe or other metal object that goes into the earth.

D. TRIAL OPERATION

Using the joint-control keys

When the power supply is connected and the TeachMover is first powered up, the green TRAIN light on the hand-held teach control (Figure 2-4) should be on. Pressing the B, E, P, R, and G keys activate the base, shoulder, and elbow joints, the hand's pitch and roll, and the grip, respectively. The right-hand column of keys move the joints in one direction, and the left-hand keys move the joints in the opposite direction. Use these arm-control keys to move each joint back and forth to familiarize yourself with how these keys operate.

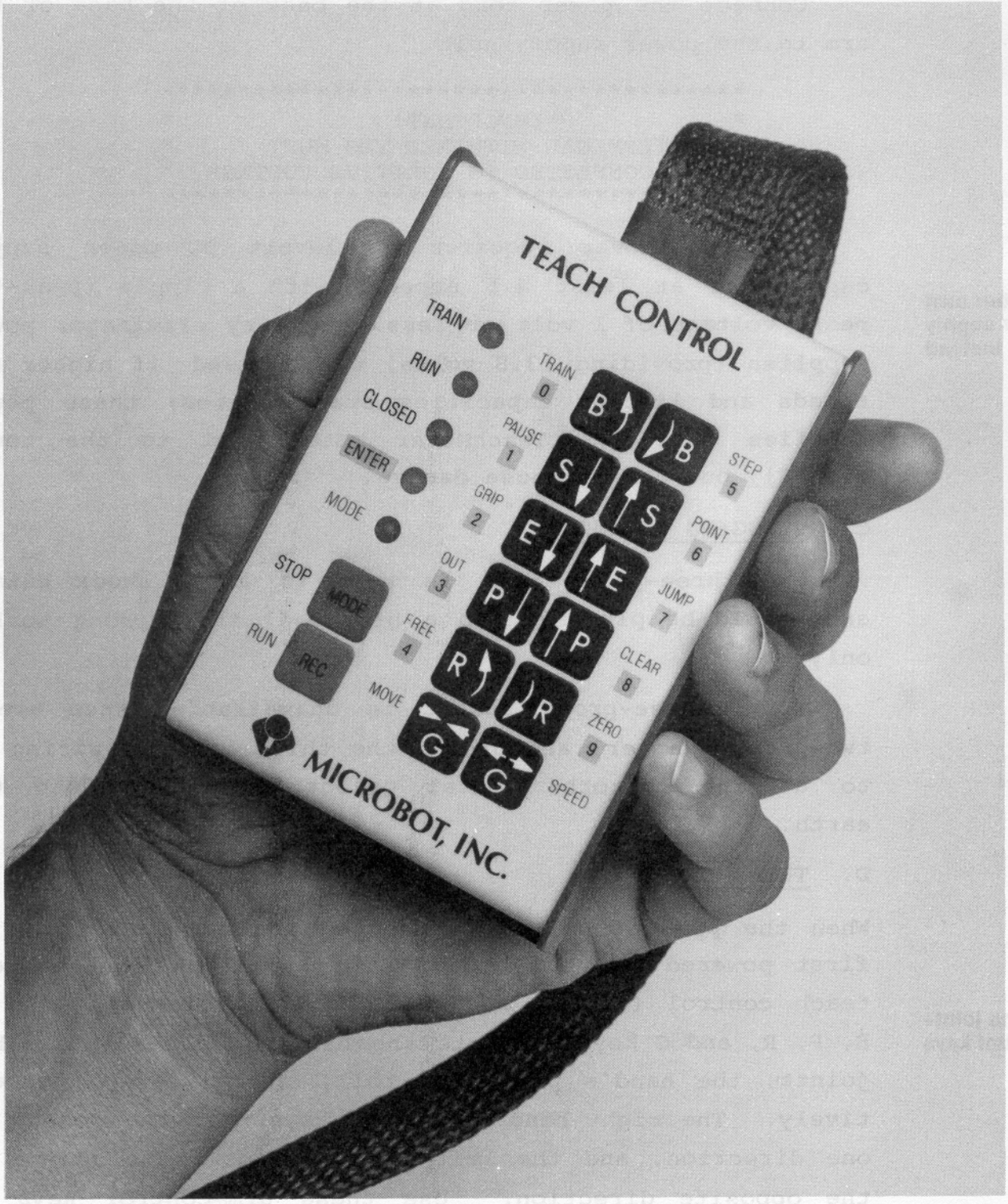


Figure 2-4 Hand-held Teach Control

Once you're familiar with moving the joints one at a time, try pressing two or more keys simultaneously to see what kind of motion results. After experimenting with this for awhile, try causing the hand to move along an approximately straight path. Now try causing the arm to pick up an object, move it to another location, and then release it.

E. TRIAL PROGRAMMING

Since the TeachMover is programmable, it may have stored some of the key entries you made during your trial operations. Each time you begin a new programming sequence, give the TeachMover a clean start with the following procedure:

Press the MODE key and, while holding it down,
press the CLEAR key; then release both keys,
Press the MODE key and, while holding it down,
press the ZERO key; then release both keys.
Now try the following sequence to program the unit.

Using the REC key

- Press TRAIN key.
- Swivel the base by pressing one of the B keys.
- Press the REC key.
- Swivel the base back by pressing the other B key.
- Press the REC key.
- Raise the elbow by pressing the E key.
- Press the REC key.
- Press the MODE key and then the RUN key. (The RUN key is physically the same as the REC key, but since you're in a different mode, it functions differently.)

At this point the green RUN light should go on, and the arm should move in a triangular pattern. Pressing the STOP key will immediately stop the robot. Pressing RUN again will resume the motion. (Pressing the REC key while the arm is moving will cause the arm to stop when it has reached the next one of your three programmed points.)

Now, if you press the MODE key, hold it down, and at the same time press the CLEAR key, the program is erased. Pressing RUN no longer causes the arm to move. Before starting a new program, you also need to press the MODE key with the CLEAR key to clear the internal registers. The TeachMover is again ready with a clean start.

After thus clearing the TeachMover, try creating a program of your own. To do so, first press the TRAIN key to get back into TRAIN mode. Then press the various arm-motion keys, remembering to press REC after the arm achieves each desired position. If you want to store the starting position, you will need to press REC before moving the arm anywhere else.

Note that you can use more than one arm-control key in between recorded positions. For example, you can press B, then E, then P, then REC. When you run the program (that is, when you press MODE, then RUN), the arm will simply take a direct path between your recorded points. Try it!

At this stage, it is also appropriate to try using the PAUSE command. Clear the TeachMover, put it into the TRAIN mode, then try this:

Using the PAUSE command

- Record any three successive arm positions you wish.
- Press the MODE key (to exit from TRAIN mode) and then press the PAUSE key.
- Notice that the yellow ENTER light goes on. This means that the yellow numerical labels now apply to the keys.
- Press the keys to enter the number of seconds from 0 to 255.
- Press MODE, then RUN, to start the program.

If all is well, the arm runs through the three programmed positions, pauses for the number of seconds you keyed in, runs through the three positions again, pauses again, and so on until you STOP it.

Later on, you'll learn how to use all the other commands (OUT, ZERO, STEP, etc). But first, it is necessary to learn something about how the arm is built and how it functions electrically.

Before going on to the next chapter, feel free to experiment with what you've learned so far. Now is a good time to get used to how the arm moves, how to position it to pick up an object, how to avoid obstructions, and so forth.

CHAPTER THREE
HOW THE TEACHMOVER IS BUILT

A. MAJOR COMPONENTS

The TeachMover's major structural components are shown in Figure 3-1 which is a duplicate of Figure 2-2. The microprocessor card is housed in the base. The teach control cable and the D.C. power cord extend from the rear of the base. The two RS-232C connectors are also located at the rear of the base. The body swivels relative to the base on a hollow shaft attached to the base. This shaft is called the base joint.

Members:
base
body
upper arm
forearm
hand (gripper)

Six stepper motors with gear assemblies are mounted on the body and control each of the six joints. The power wires for the motors pass from the computer card in the base through a hollow shaft to the body. This arrangement provides a direct cable-drive system.

Joints:
base joint
shoulder
elbow
right wrist
left wrist

The upper arm is attached to the top of the body and rotates relative to the body on a shaft called the shoulder joint. Similarly, the forearm is attached to the upper arm by another shaft known as the elbow joint.

Finally, the hand, also called the gripper, is attached to the forearm by two wrist joints. Two separate motors operate the wrist joints to control the pitch and roll of the hand.

The TeachMover arm has a lifting capacity of one pound when fully extended, and a resolution (the smallest amount the arm can be made to move) of 0.011 inches. The end of the hand can be positioned anywhere within a partial sphere with a radius of 17.5 inches, as shown in Figure 3-2. The maximum speed is from 2 to 7 inches per second, depending upon the load (weight of the object being handled.) Detailed performance characteristics of the TeachMover are given in Table 3-1; this table is reproduced in Appendix F for easy reference.

In general, the base, the body, and all the extension members are hollow sheet-metal parts which are light in weight but strong. All members are connected to each other by means of shafts, or axles, passing through bushings mounted on the members.

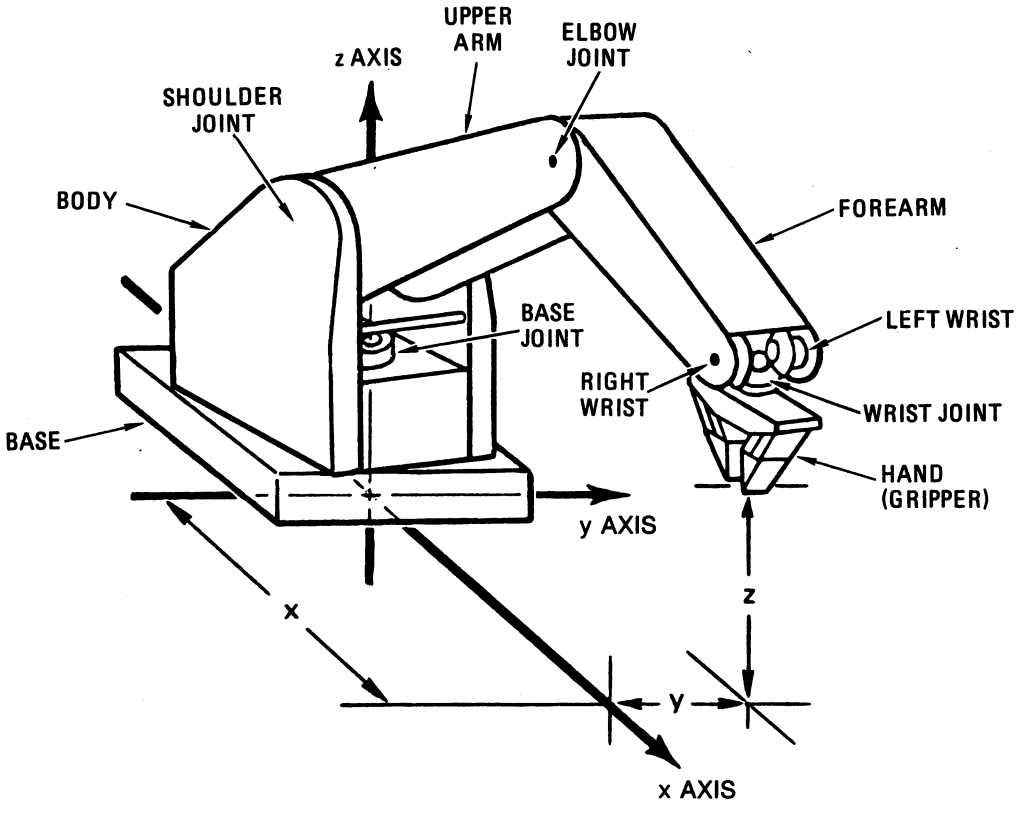
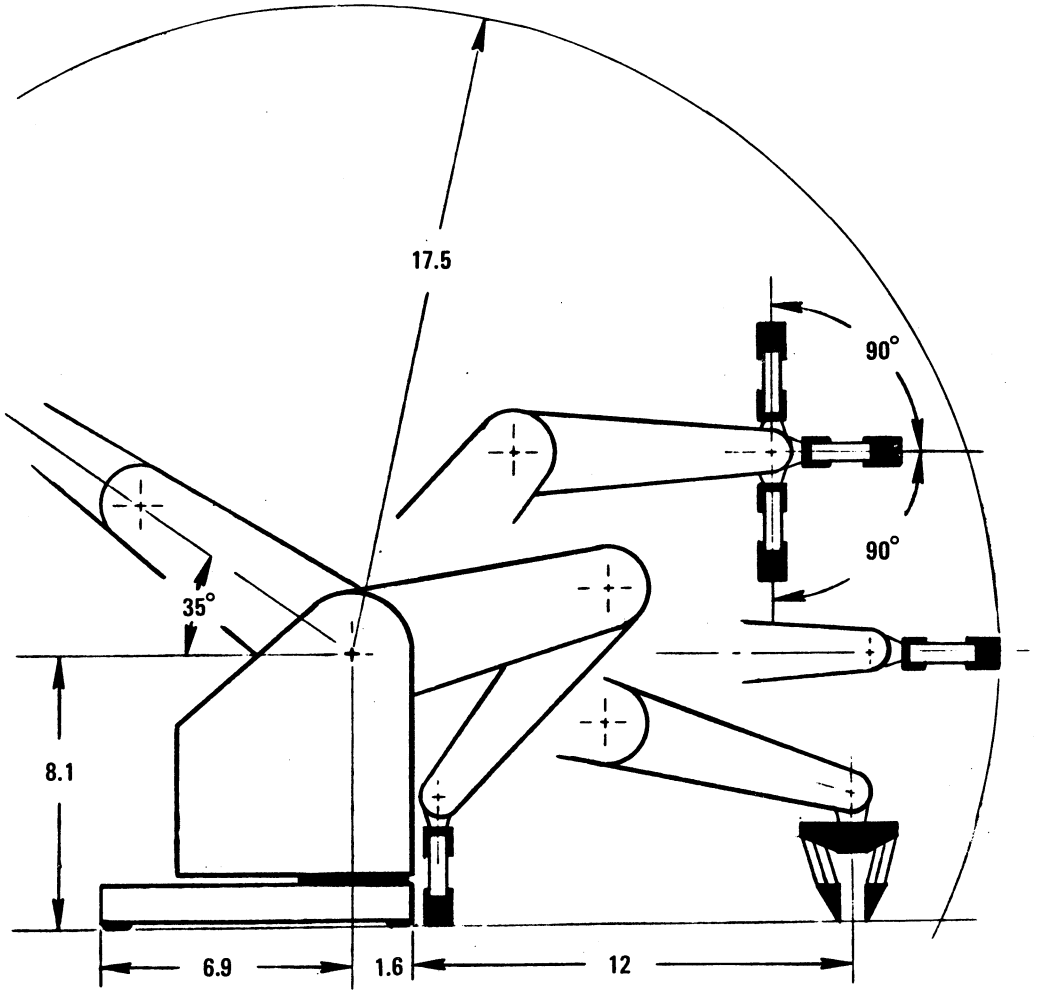


Figure 3-1 Major Structural Components



NOTE: Dimensions in inches.

Figure 3-2 Operating Envelope of the TeachMover Arm

TABLE 3-1

TEACHMOVER PERFORMANCE CHARACTERISTICS

GENERAL

Configuration	5 revolution axes and integral hand
Drive	Electric stepper motors- Open door control
Controller	6502A microprocessor with 4K bytes of EPROM and 1K bytes of RAM located in base of unit
Interface	Dual RS-232C asynchronous serial communications interfaces (baud rate is switch-selectable between 110, 150, 300, 600, 1200, 2400, 4800, and 9600 baud)
Teach Control	14-key 13-function keyboard; 5 output and 7 input bits under computer control
Power Requirement	12 to 14 volts, 4.5 amps DC

PERFORMANCE

Resolution	0.011 in (0.25 mm) maximum on each axis
Load Capacity	16 oz (445 gm) at full extension
Gripping Force	3 lbs (13 Newtons) maximum
Reach	17.5 in (444 mm)
Velocity	0-7 in/sec (0-178 mm/sec) with controlled acceleration

DETAILED PERFORMANCE

Motion	Max Range of Mtn	Speed (Full Load)	Speed (No Load
Base	$+90^{\circ}$	0.37 rad/sec	0.42 rad/sec
Shoulder	$\mp 144^{\circ}, -35^{\circ}$	0.15 rad/sec	0.36 rad/sec
Elbow	$+0^{\circ}, -149^{\circ}$	0.23 rad/sec	0.82 rad/sec
Wrist Roll	$+360^{\circ}$	1.31 rad/sec	2.02 rad/sec
Wrist Pitch	$\mp 90^{\circ}$	1.31 rad/sec	2.02 rad/sec
Hand	0-3 in.	8 lb/sec* (35 n/sec)	(20 mm/sec)

PHYSICAL CHARACTERISTICS

Arm Weight	8 lbs (4kg)
Teach Control Cable Length	3.75 ft. (1150 mm)

* This is given in lbs/sec rather than in./sec, because as the gripper closes it no longer moves, but instead builds up gripping force. It takes 0.37 sec to build up the maximum force of 3 lbs.

C. CABLE DRIVE SYSTEMS

Most robot arms have at least some of the drive motors mounted on the extension members (forearm, upper arm, hand). Unfortunately this adds to the weight of those members, and means that the other motors - those that drive the extension members - need to be larger and more expensive than would otherwise be required.

All six motors are in the body

If you look for motors on the TeachMover's extension members, you won't find any. That's because all six drive motors are mounted in the body. This minimizes the weight of the extension members and keeps the motor workload requirements as low as possible. To reduce the number of moving parts, all six drive gears are mounted on the same shaft.

Unique cable driver system

A unique system is employed to manipulate the arm members. From the drive system in the body, aircraft-quality cables extend to the base, upper arm, forearm, and hand, as you can see by examining the TeachMover or by looking at Figure 3-3. This cable design is an adaptation and refinement of the "tendon technology" used in aircraft, high speed printers, and other types of equipment. Note that each cable is wound around the hub of the drive gear. This serves not only to provide a take-up drum for the cable, but also gives the proper gear-reduction ratios for each of the six drives.

Now, let's look briefly at how each of the cable drives is constructed. As you read about each mechanical part, it's a good idea to locate that part on your TeachMover. You need not be an expert in all aspects of the cable drive system, but some basic knowledge of how the cables work can prove extremely valuable later on.

As you look down on the hubs of the drive gears, you can see a set screw on each hub pinning the cable tightly in a groove cut into the hub. As the drive gear turns,

the hub is pulling, or winding, one-half of the cable while unwinding the other half. (See Figure 3-3)

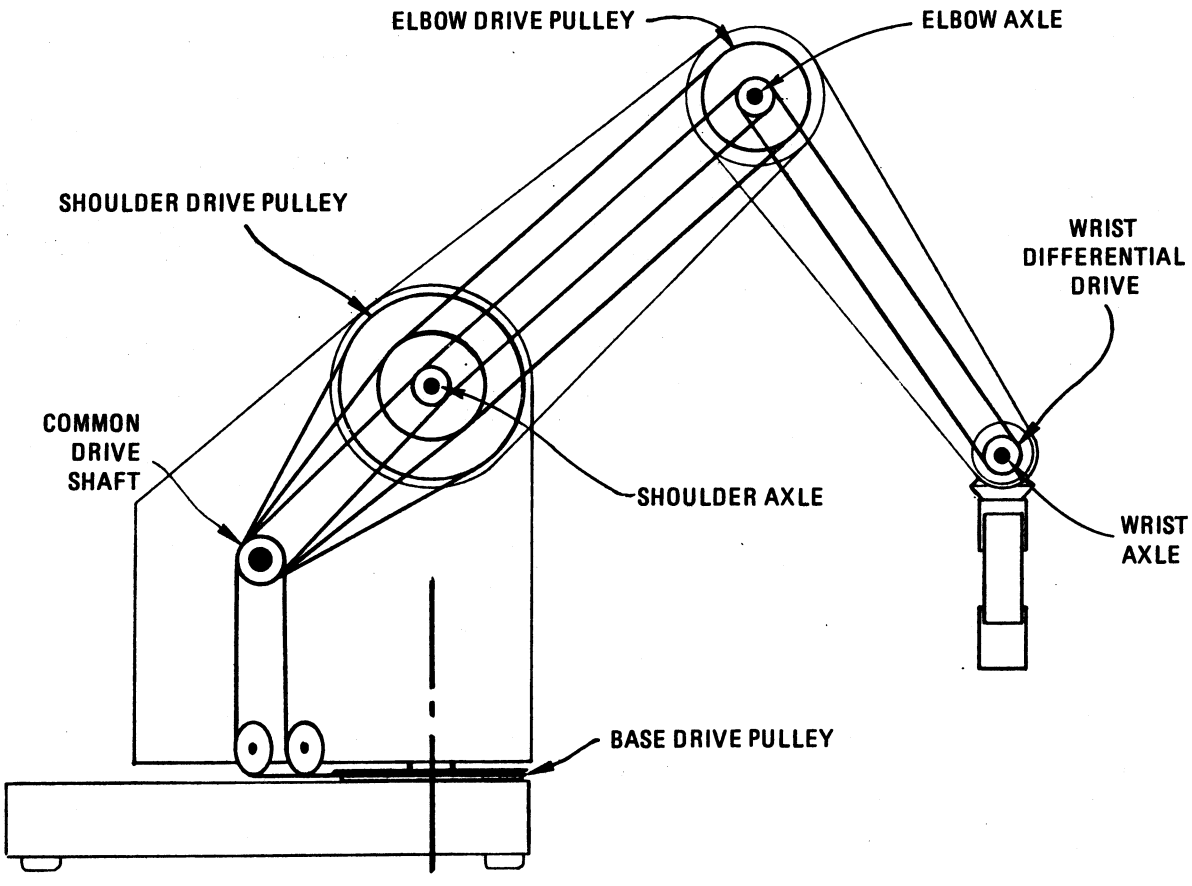


Figure 3-3 Cable Drive System

1. Base Drive

The base drive causes the body to rotate on the vertical base axle by driving a large pulley mounted on the base. Two small pulleys, located at the bottom of the body, change the base cable direction so that the cable feed is tangent to the surface of both the drive drum and the base pulley. You can see how these pulleys work if you rotate the body manually with the power turned off. Note the termination of the cable in a clamp fastened by two screws on one side of the base.

2. Shoulder Drive

Now hold the body in place and move the upper arm to see how the shoulder cable causes the upper arm to rotate on the horizontal shoulder axle. Note how this cable passes around a drive pulley on the shoulder shaft, then terminates on the upper arm housing. At the termination point, you'll notice two screws. These screws are used to maintain the cable under tension, as is explained in part C, below (Cable Adjustments).

Shoulder rotation maintains orientation of hand

Now rotate the shoulder joint again and notice that shoulder rotation always causes equal and opposite elbow and wrist rotation so that the orientation of the hand remains unchanged. This feature is built into the TeachMover's cabling design to make sure that the hand can hold a glass of liquid while the shoulder rotates without spilling the liquid.

3. Elbow Drive

The elbow cable causes the elbow to rotate on the horizontal elbow axle. Note that this cable first passes around an idler pulley on the shoulder axle, then around a drive pulley of the same diameter attached to the elbow axle. The cable terminates at a tension mechanism on the forearm housing. Rotate the elbow manually and you'll

Elbow rotation maintains orientation of hand

notice that the wrist rotates the same amount in the opposite direction, thus maintaining hand oreintation.

In rotating the elbow manually (that is with the power off), you may have noticed something else: when the elbow rotates, the hand opens and closes. Designing cabling to prevent this from happening mechanically would have added undesirable complexity. Instead, a built-in software routine automatically decouples elbow rotation from hand closure. To see this in action, turn the power on and press one of the E keys. Notice that now the elbow moves without opening or closing the hand.

Elbow rotation can affect grip opening

Note : When the TeachMover is in serial interface mode, this automatic decoupling is inoperative, and, once again, rotating the elbow will effect hand closure. It is, however, easy to provide for the decoupling yourself when you write a host computer program. A simple formula to accomplish this is given later with the discussion of the commands available for your use in serial interface mode.

4. Wrist Drive

The right and left wrist cables cause the hand to "roll" and "pitch" relative to the forearm. These cables together control the wrist joint (figure 3-4). Both cables pass around idler pulleys on the shoulder axle and the elbow axle, then around the hubs of bevel gears located on the wrist axle. Tension is maintained in both cables by means of turnbuckles (to find them, look inside the forearm housing.)

Note how the two bevel gears on the wrist axle mesh with the output gear on the hand axle. This configuration forms a differential gear set.

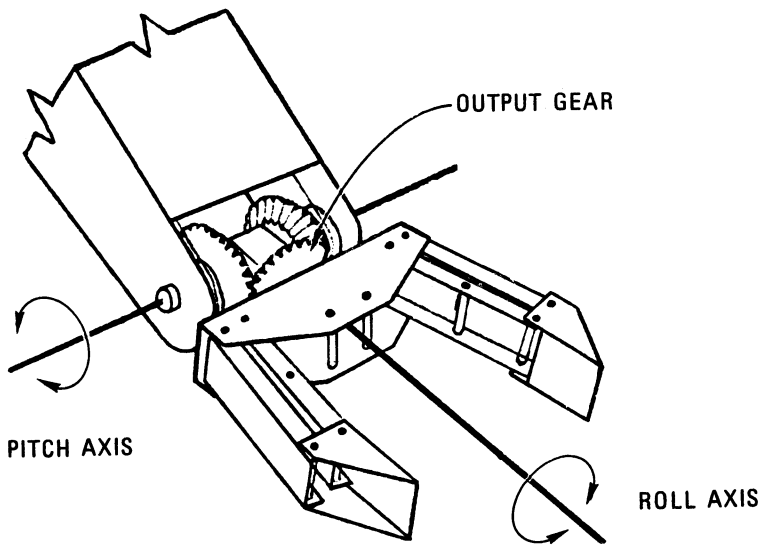


Figure 3-4 The Wrist Joint

Differential gear set allows for pitch and roll

To see how this differential works, turn the drive gears so the left and right wrist cables both move in the same direction. You can see the wrist gears control the pitch of the hand. Turning the drive gears so the wrist cables move in opposite directions controls the roll of the hand.

The TeachMover's built in software automatically coordinates left and right wrist motions to produce the amount of pitch or roll you specify with the P and R keys on the teach control. When the TeachMover is in serial interface mode, you cannot specify pitch and roll directly, but only the amount of motion of the left and right wrists. These motions can, however, be coordinated by a simple formula to produce the desired amount of pitch or roll.

5. Hand Drive

The hand cable system is shown in Figure 3-5. Attached to the output gear of the differential gear set, the hand housing holds two pairs of links, and each pair of links terminates in the gripper. The housing, the links, and the gripper are attached to each other by small pins. Torsion springs located on the pins attach the links to the hand housing and provide the return force to open the hand as the hand cable is slackened.

Note: The length of the hand varies slightly with hand opening. For most applications, the amount of variation is negligible. For high-precision work, however, it may be necessary to take the variation into account. A formula for this is given at the end of Appendix D.

The hand cable is attached to the hand drive drum located in the body. The cable passes over an idler pulley located on the shoulder axle, and then over an idler pulley mounted on a sensing bracket found inside

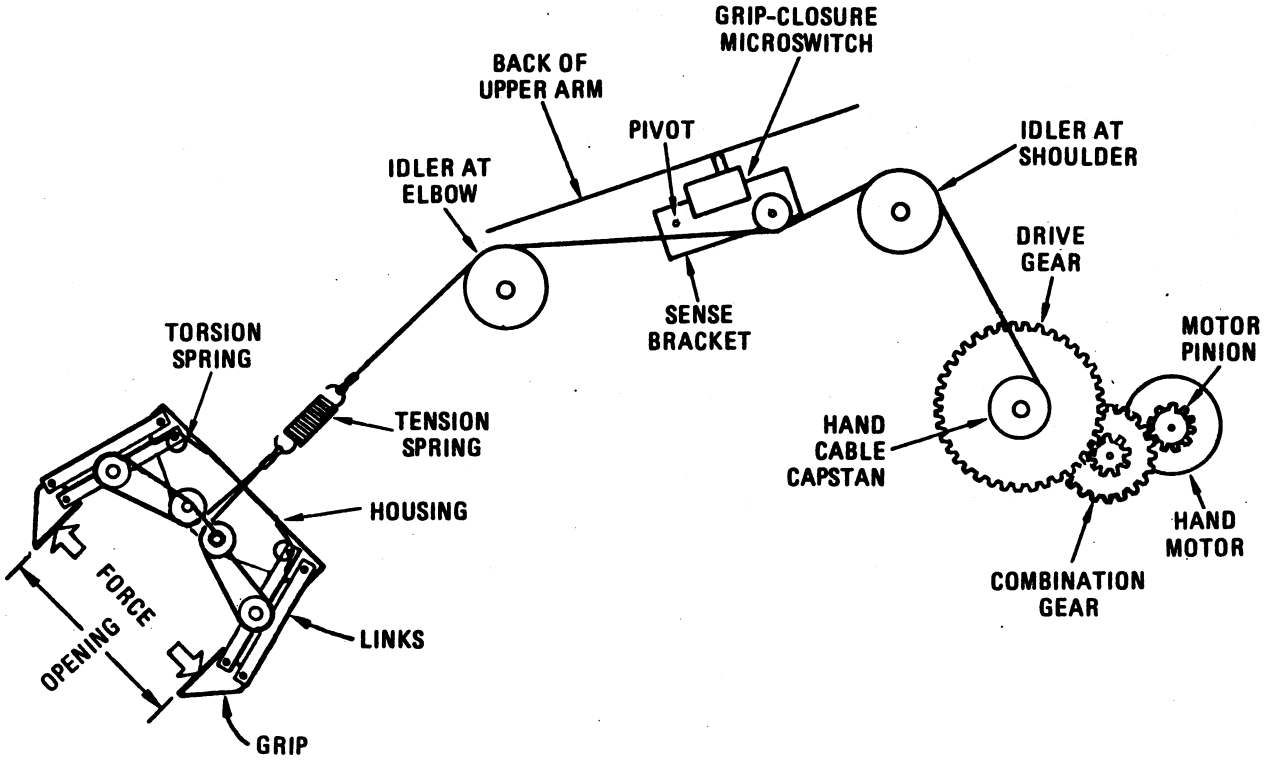


Figure 3-5 Hand Cabling Diagram

Location of grip switch

the upper arm housing. This sensing bracket is also called the grip micro switch. This switch goes on when the cable tension increases just beyond the point where the hand closes on an object (or on itself).

As we'll see later, the status of the grip switch can be used in programming for conditional branching. When the grip switch is activated, the green "CLOSED" light in the hand-held teach control goes on. Try it. If you close the gripper on an object and the green light does not go on, then check to see whether the hand cable has come off the shoulder idler pulley, the elbow idler pulley, or the sense bracket (grip switch) pulley. If it has, then simply replace it (referring, if necessary to the Hand Cabling Diagram on Page 3.13), and try again. If the green "CLOSED" light goes on before the gripper actually closes, then try tugging on the hand cable a few times, or adjusting the cable tension (see Section C, Cable Tension Adjustments, below.)

If grip switch doesn't work properly

Attached to the other end of the tension spring, and in line with the hand cable, you'll notice two separate link-drive cables. These cables pass over two guide pulleys in the wrist yoke and then through the center of the hollow hand axle. When the cables emerge from the hand axle, they pass over separate idler pulleys mounted in the base of the hand. They then pass around idler pulleys mounted on the inner links of the hand, and return to and terminate on the shafts of the two pulleys mounted on the hand base. This arrangement forms two block-and-tackle devices that augment the gripping force of the hand. The use of identical cabling on both links provides for symmetrical hand closure.

The tension spring mounted in series with the hand-cable drive assembly permits gripping force to be built up by the position-controlled drive motor once the hand has

closed. Figure 3-6 shows: (1) The relationship between hand opening and drive motor steps (this is the line sloping down to the right), and (2) the relationship between gripping force and drive motor steps once the hand has closed (this is the line sloping up to the right). Note that the maximum gripping force is 3 lbs; this occurs approximately 100 motor steps beyond hand closure. Figure 3-6 is reproduced in Appendix F for ready reference.

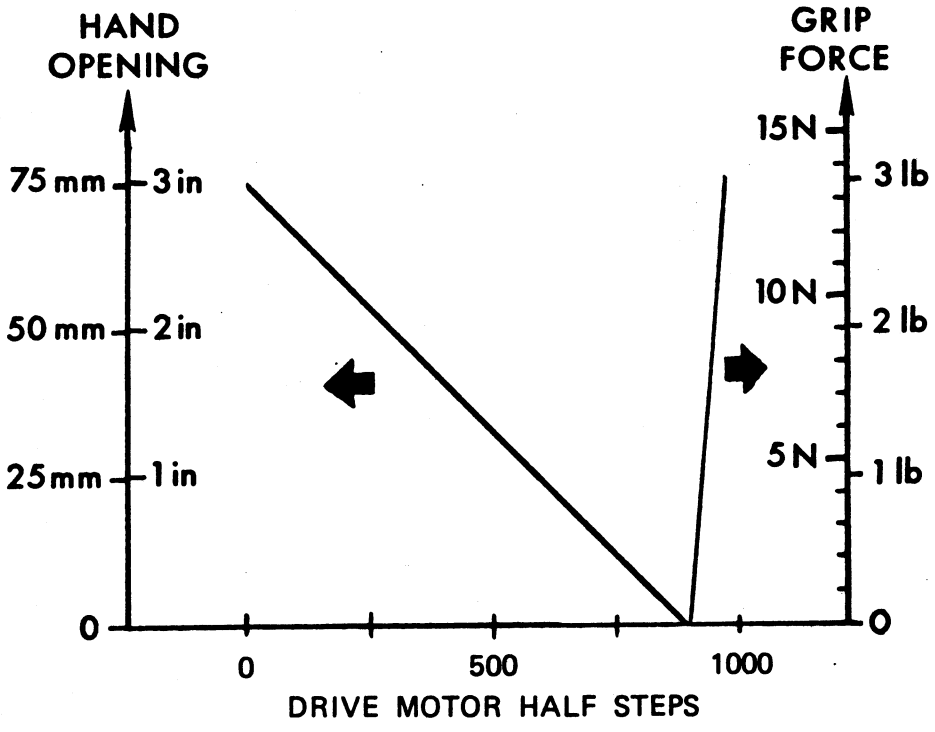


Figure 3-6 Hand Opening and Gripping Force Diagram

C. CABLE TENSION ADJUSTMENTS

After a period of extended use or after an extreme overload, tension adjustments may slip. The relaxed half of a cable should not have noticeable slack. If a cable does develop slack, then the cable needs to be tightened.

Tension adjustments are provided at the following locations:

DRIVE	LOCATION
Base	Right side of base (see figure 3-7)
Shoulder	Right side of upper arm
Elbow	Left side of forearm
Wrist	Turnbuckle inside forearm

The adjustment procedure is as follows:

1. Base, Shoulder, Elbow:
   - Loosen both tension screws.
   - Pull firmly (2-3 lbs.) on one tension screw to tension the cable as shown in Figure 3-7, and tighten the other screw with the other hand.
   - Release tension and tighten both screws.

Be careful not to put excessive tension in the cable. Tight cables can cause the motor to slip with a loss of orientation between the microprocessor and the arm position.

2. Wrist:
   Each turnbuckle may be tightened or loosened as required to achieve proper tension. As with the other adjustments, be careful not to tension the cables so much that the motors slip.

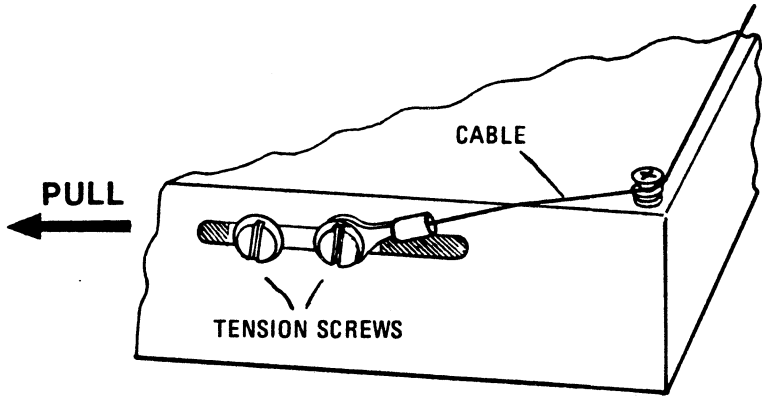


Figure 3-7 Base Tension Adjustment

CHAPTER FOUR
HOW THE STEPPER MOTORS OPERATE

A. FUNDAMENTALS

Each of the cable drives is controlled by a stepper motor. The motors used have 4 coils, each driven by a power transistor. The drive is digital, with the transistors either turned on or turned off to obtain the desired pattern of currents in the motor windings. By changing the pattern of currents, a rotating magnetic field is obtained inside the motor that causes the motor to rotate in small increments or steps. More information on stepper motor control can be found in references [3] and [5].

Stepper and servo motors compared

Stepper motors are not the only kind of motors used in robot arms. Some arms use servo motors with electronic feedback loops for precise position control. Unlike stepper motors, these servo motors cannot develop slippage. This advantage must be weighed against the servo motor's far greater cost.

Keeping cost as low as possible is one reason we chose to use stepper motors for the TeachMover. Another reason is that stepper motors are easier to control from a computer than are servo motors.

"Half-Stepping"

Now, in order to turn a stepper motor in the TeachMover, a particular sequence of binary phase patterns is output to the desired motor, one pattern per step. In order to change motor direction, the order in which the phase patterns are output is simply reversed. The particular phase patterns used in the TeachMover generate a sequence known as "half-stepping;" the steps are half the size specified by the motor manufacturer. (The motors used to drive the TeachMover are specified by the manufacturer at 48 steps per revolution, but are actually

stepped at 96 steps per revolution.) Compared to full stepping, half-stepping produces smoother slow-speed motions, reduces the power requirement, and improves the arm resolution by a factor of two.

The relationship between motor steps and actual joint rotation is given in Table 4-1. (The relationship between motor steps and hand opening was given in Figure 3-6.)

Table 4-1

MOTOR STEPS AND JOINT ROTATIONS

Motor	Joint	Steps per degree	Steps per radian
1	Base	19.64	1125
2	Shoulder	19.64	1125
3	Elbow	11.55	672
4	Right wrist	4.27	241
5	Left wrist	4.27	241

B. SPEED-TORQUE CONSIDERATIONS

The torque output (lifting capacity) of the stepper motors used on the TeachMover varies with their speed. At slow speeds, maximum torque is obtained. Above a critical high speed the motors suddenly slip, and no torque is obtained. (Motor slippage can cause a discrepancy between where the arm is and where the computer program thinks it is, and this may result in unpredictable performance.)

The torque required by the motors of the TeachMover also depends on the configuration of the arm and the load held in the hand. This relation is a complex trigonometric expression involving the lengths and weights of all the arm members. Instead of solving such an expression before each arm movement to determine the maximum allowable speed, it is simpler to program for the worst case.

The worst case is when the members of the arm are at maximum horizontal extension, requiring the maximum motor torque. All other configurations will require less motor torque. With the arm fully extended but with no load, the torque on all the motors is the same (by design) and motor speed can be as high as 400 half-steps per second, as indicated by the "no-load" point in Figure 4-1. Above this speed the motors will slip, and the torque will be zero. With the arm carrying the maximum rated load (that is, with the arm lifting 16 ounces) the torque on all the motors (except the base motor, which does not lift) is approximately equal, and maximum speed without slippage is 99 half-steps per second; this is shown as the "full-load point" in Figure 4-1. At half rated load (8 ounces), maximum speed without slippage is 206 half-steps per second. These figures will become important later, when we discuss the commands you can use to control the speed of the TeachMover arm.

Full-load, half-load, no-load points

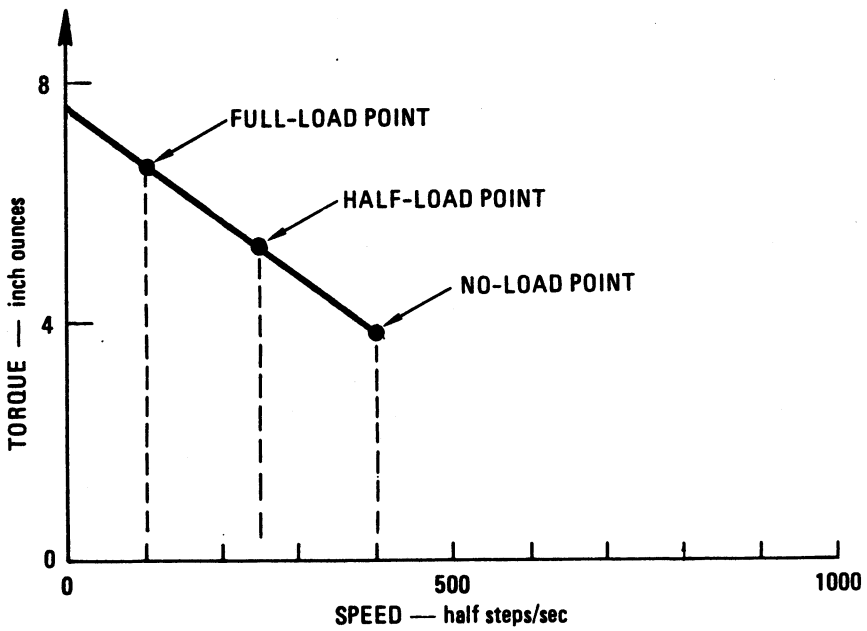


Figure 4-1 Stepping Motor Speed/Load Trade-off

Speed-Torque Considerations

To perform a task as fast as possible without risking slippage, the following suggestions may prove helpful:

Tips to avoid motor slippage

- Lowering a load may be done at no-load speed-even if the arm is holding a load - provided the shoulder, elbow and wrist do not raise.
- Swiveling a load about the base joint may always be done at no-load speed.
- Opening the hand, or closing the hand until the contact point is reached, may always be done at no-load speed.

However special care must be exercised in selecting the proper speed when:

- raising a load with any joint.
- developing a gripping force once the gripper has closed on an object or on itself.

C. MOTOR CONTROL

Sequential vs. simultaneous (coordinated joint movement)

Moving the arm from one position to another often requires rotation of more than one joint. In such cases, the motion can, in principle, be accomplished in either of two ways: the joints can be rotated sequentially or simultaneously. For example, as shown in Figure 4-2, motion of the arm from A to C can be accomplished through separate, sequential motions of the elbow and then the shoulder (A to B, then B to C), or through a coordinated motion in which the elbow and shoulder joints move simultaneously (A to C).

Coordinated motor timing

In general, coordinated motion is both smoother and faster than sequential motion. TeachMover firmware is programmed to produce coordinated motion whenever two or more motors are needed to move the arm from one recorded position to the next. To accomplish the coordination, the motor steps are timed so that each motor is pulsed at regular intervals during the full duration of the move. For example, if the shoulder motor is told to move 3 steps and the elbow motor is told to move 21 steps, the resultant timing will be as shown in Figure 4-3.

Controlled acceleration

The TeachMover's motor-control algorithm has another feature: it produces controlled acceleration and deceleration to minimize jerkiness when the arm starts and stops. The velocity profile for motion of a stepper motor at a speed of 450 half-steps per second is shown in Figure 4-4. Note that for relatively short motions, such as the 100-step motion shown in Figure 4-4, the motor might not actually reach the specified speed before it needs to begin decelerating for a smooth stop. (We'll describe how to specify motor speeds in Chapters 6 and 7.)

Approximating straight-line motion

Although the TeachMover's arm members move along curved paths, motion in a straight line may be approximated by a series of these curved motions. For

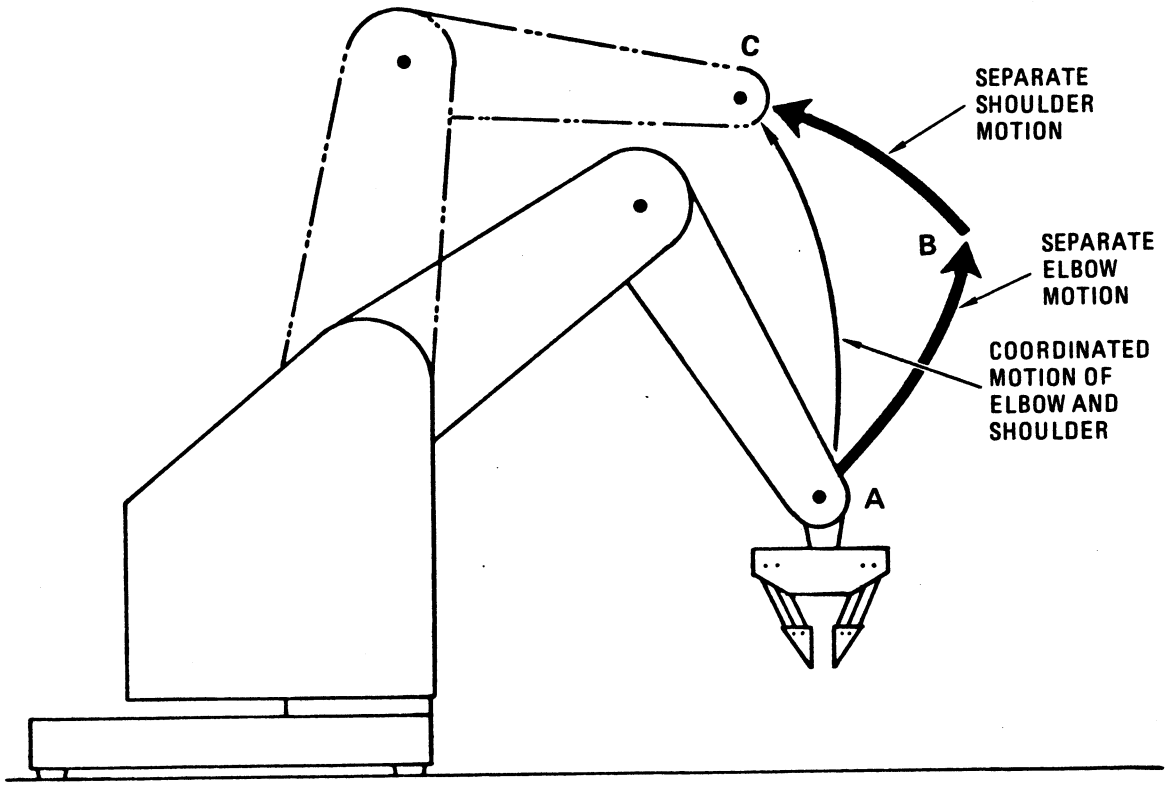


Figure 4-2 Coordinated versus Sequential Motions

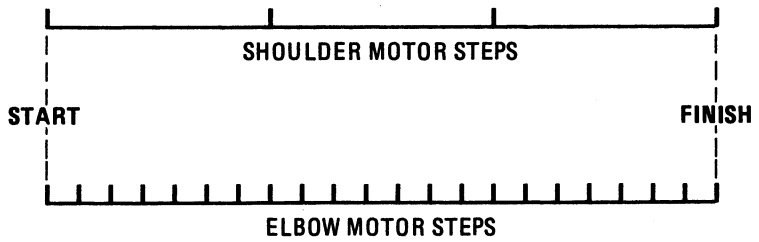


Figure 4-3 Step Timing Diagram

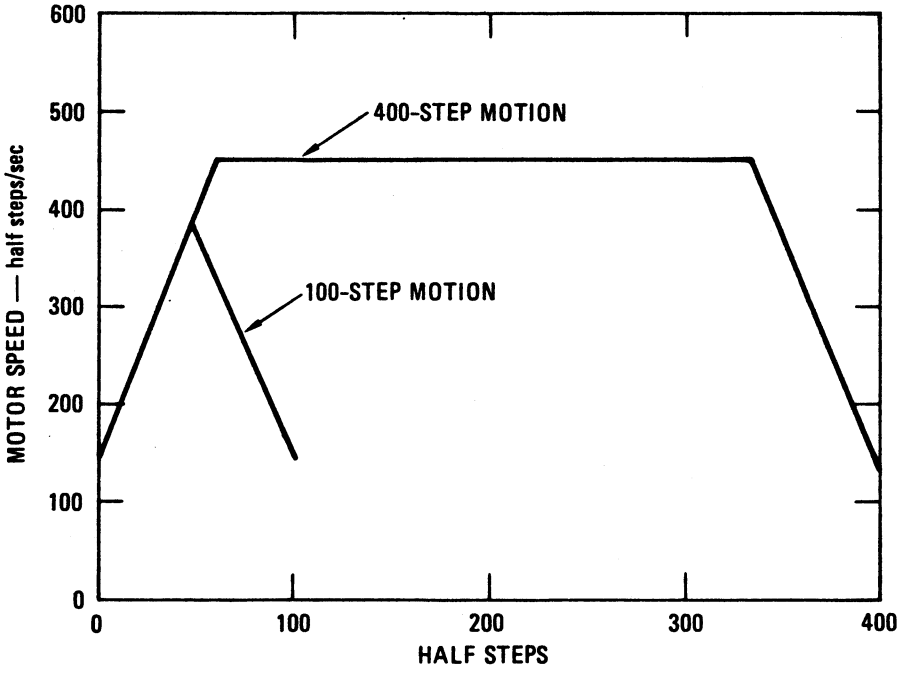


Figure 4-4 Trapezoidal Stepper Motor Velocity Profile

example, one of the TeachMover's built-in demonstration programs moves the arm along an approximately straight 10-inch line by means of 11 steps spaced one inch apart. (We'll explain how to run the demonstration programs later.) The segmented approximation has a theoretical error of only 0.018 inches in tracking the desired 10-inch line.

CHAPTER FIVE

INTERNAL ELECTRONICS AND INTERFACES

A. ON-BOARD COMPUTER AND MEMORY

If you open up the base of the TeachMover (by gently laying the unit on its side and removing the four screws you'll find on the bottom), you'll see the circuit card that houses all the internal electronics, including the 6502A Microprocessor (Figure 5-1). In technical terms, this microprocessor is an 8-bit, 2MHz chip. It's the same chip used in the Apple, Atari, & PET computers; it is used in the TeachMover to coordinate all joint motions and handle all input and output.

6502A microprocessor

TeachMover firmware (permanently built-in software) is contained in another chip housing 4K bytes of read-only memory (ROM); this firmware interprets the commands you give to the arm, converting these to electrical signals the arm can obey. Also contained in the 4K ROM are two built-in demonstration programs. More on these later.

4K ROM

The circuit card also includes chips containing 1K bytes of random-access memory (RAM); this is enough RAM to let you store an arm-motion program of up to 53 steps. It is possible to "piggy-back" a second set of RAMs on the first, thereby extending your program capacity to 126 steps. See Appendix C for instructions.

1K RAM

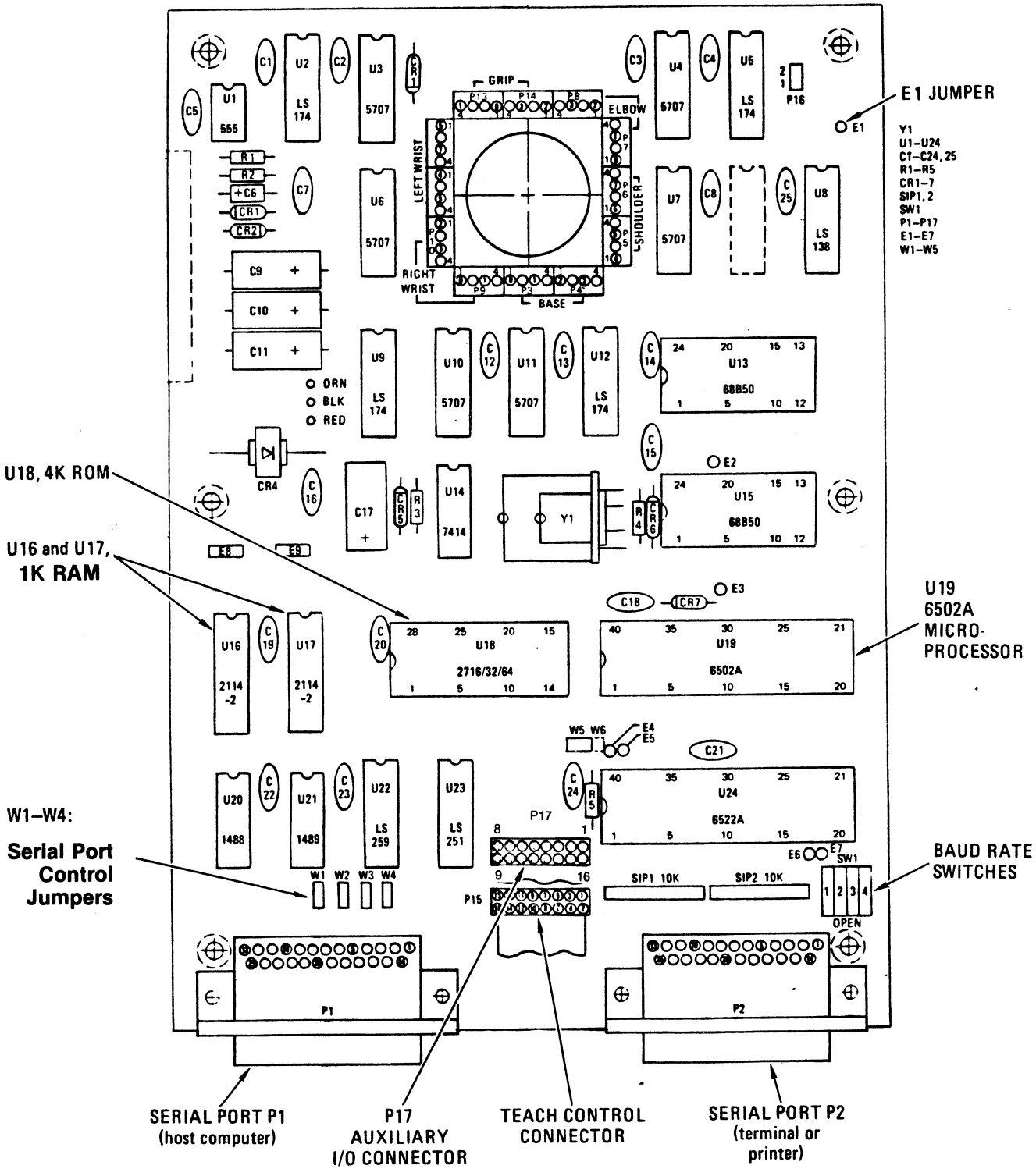


Figure 5-1 TeachMover Circuit Card (bottom view)

B. Serial Ports

In the rear of the base, on either side of the flat cable that goes to the hand-held teach control, you'll see two multi-pin connectors. These are the serial interface ports that allow you to connect the TeachMover to a host computer, printer, or terminal. A switch on the computer card allows you to select the serial transmission speed; eight standard speeds are available, from 110 to 9600 Baud. Further details of serial port operation are given in the chapter on serial interface mode.

C. User Inputs and Outputs

The computer card also contains an auxiliary parallel input/output port. This lets you interface the TeachMover to external equipment with a 16-conductor flat ribbon cable. Five TTL compatible user output bits can be set (to 1) or cleared (to 0) under program control to turn other equipment on or off when a given arm motion is complete. Seven TTL compatible user input bits can be used to control an arm sequence when a given external condition is met. The 16 pins and their functions are given in Figure 5-2 and Table 5-1. We'll explain how to use the input and output bits in Chapter 6.

A block diagram of the TeachMover's electronic circuitry is shown in Figure 5-3.

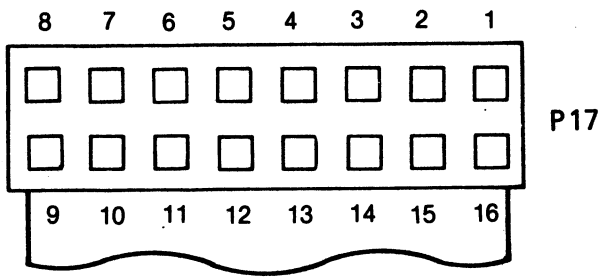


Figure 5-2 Pin Numbering for Auxiliary I/O Connector

Table 5-1

AUXILIARY I/O CONNECTOR

Pin No.	Function
1	+5V-User Power
2	Ground
3	Not Used
4	Input bit 1
5	Output bit 5
6	Input bit 2
7	Output bit 4
8	Input bit 3
9	Output bit 3
10	Input bit 4
11	Output bit 2
12	Input bit 5
13	Output bit 1
14	Input bit 6
15	Ground
16	Input bit 7

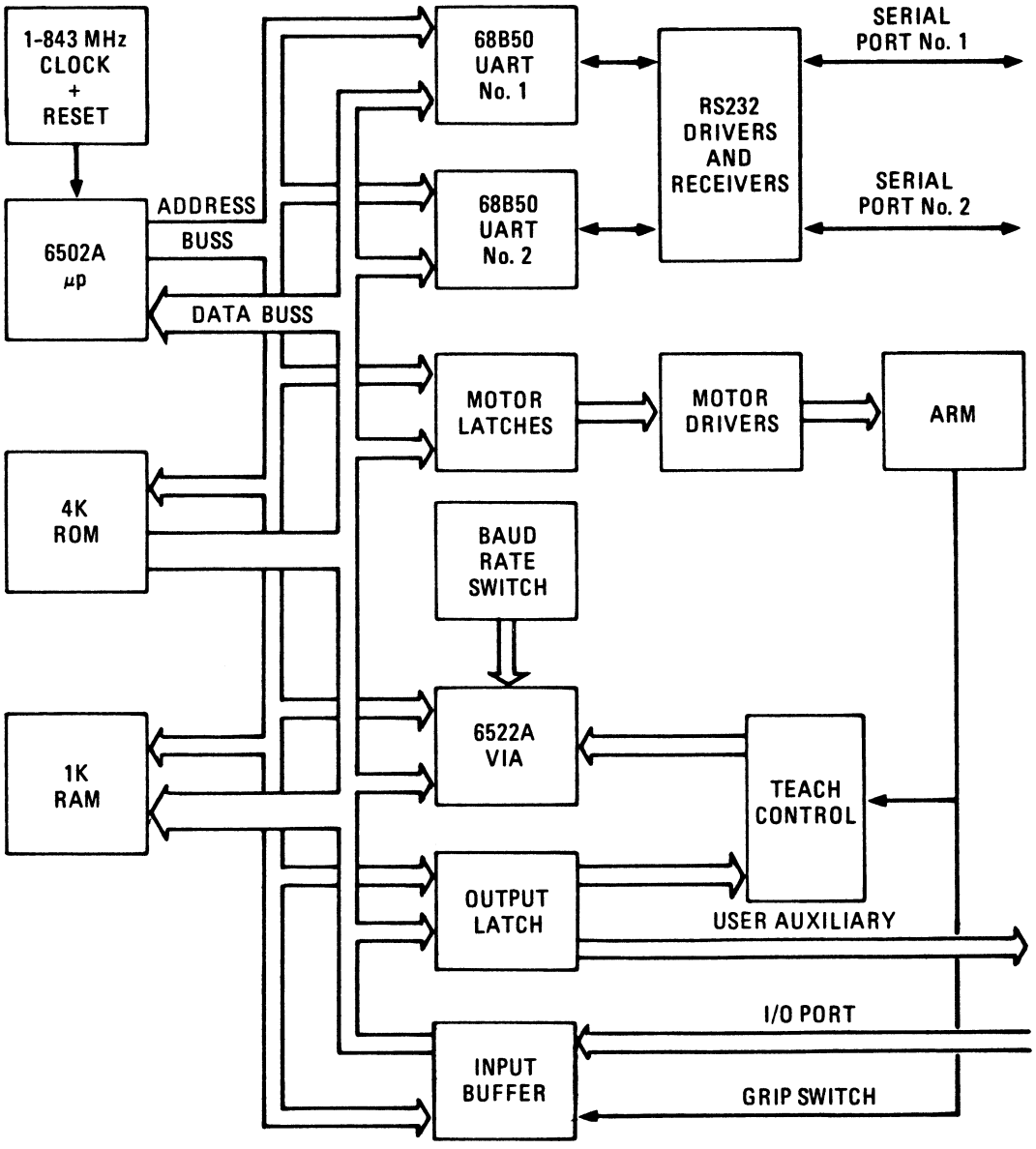


Figure 5-3 Block Diagram of TeachMover
Computer and Electronics

CHAPTER SIX

OPERATING THE HAND-HELD TEACH CONTROL

A. COLOR CODED CONTROLS

The TeachMover's hand-held teach control (see Figure 6-1) performs most of the same functions as do the teach controls on large-scale industrial robots. To provide a wide range of command options yet keep product cost to a minimum, we employed keyboard "overlays" that allow the same set of keys to provide three different kinds of functions. Rather than use an expensive alpha-numeric display to indicate which overlay is in use, we developed a simple system of color-coded lights and key labels.

For example, when you press the red MODE key, the red MODE light goes on, and the words printed in red (TRAIN, STEP, PAUSE, RUN, etc.) apply to the keys.

Three overlays

Pressing certain of these keys (PAUSE, OUT, POINT, JUMP, or SPEED) will cause the yellow ENTER light to go on. When this light is on, the yellow numerals next to the keys apply, and you can enter numericals values (exactly how, we'll explain later). When the ENTER light is on, pressing the REC button will clear the entered value, allowing you to then enter the correct value. Pressing the MODE button terminates enter mode.

Finally, the labels printed on the keys themselves (B, S, E, P, R, G, and REC) apply when the teach control is in TRAIN mode (or in MOVE mode, as you'll see).

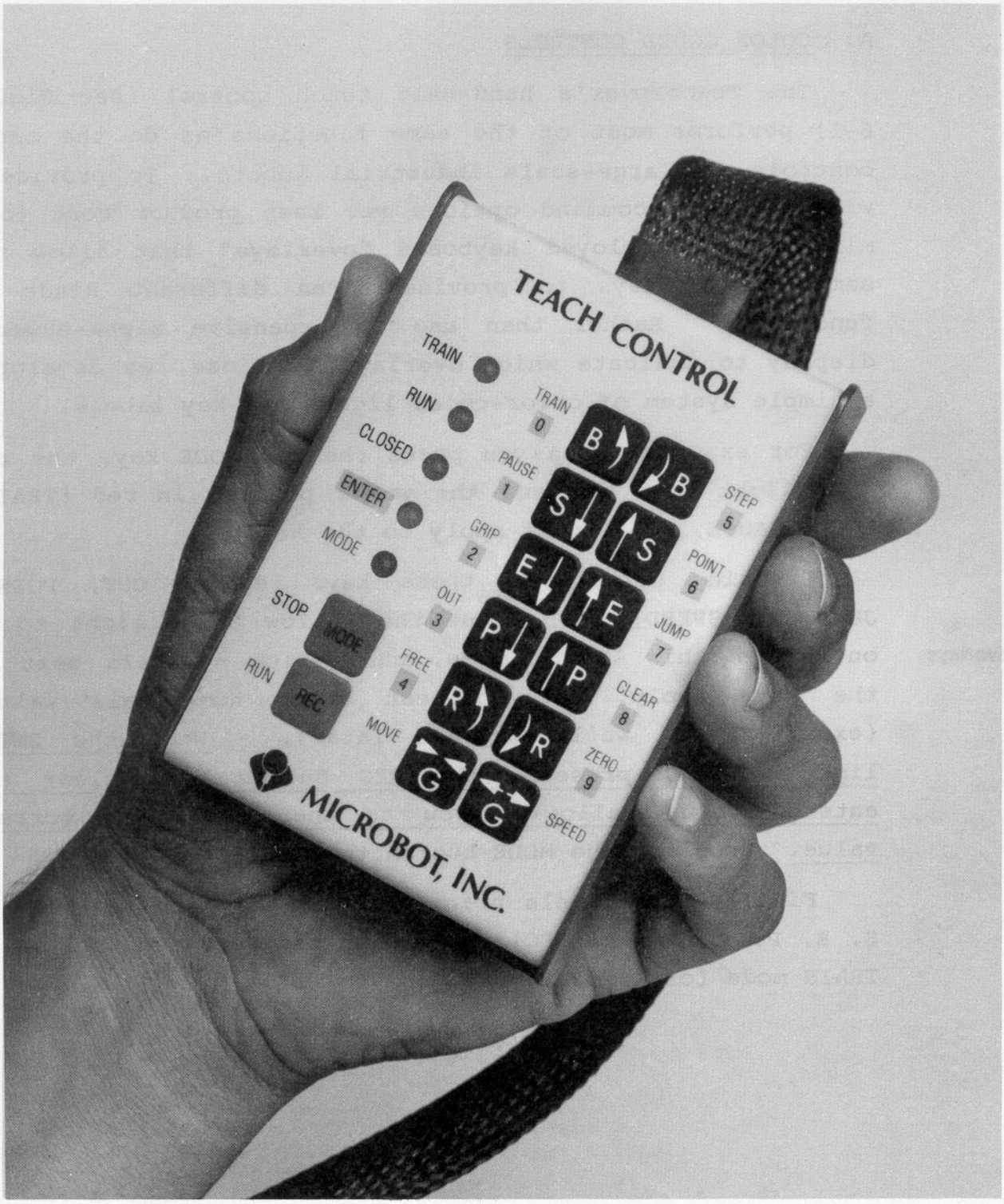


Figure 6-1 Hand-held Teach Control

B. The Thirteen Control Functions

There are 13 different control commands you can give with the teach control. (A concise summary of all teach control commands is given in Appendix F.)

1. TRAIN

You can put the teach control into TRAIN mode in two different ways.

Two ways to enter TRAIN mode

A. Simply turn the unit on. This puts the teach control into TRAIN mode automatically. (Note: turning the unit off erases the current program, and is therefore not recommended during arm training.)

or B. Press the red MODE key at any time to make the red MODE light go on, then press the key labeled, TRAIN.

Once the teach control is in the TRAIN mode, you can use the arm-motion (joint-control) keys and the REC key to manipulate the arm and record arm positions. (If necessary, refer back to Chapter 2, Section E, Trial Programming, to review how to do this.)

You can program up to 53 steps; these steps are internally numbered 0-52. (By adding additional RAM according to the instructions in Appendix C, you can extend program memory to 126 steps; these steps are internally numbered 0-125.)

Use of REC key

Important Point : Pressing the REC key writes the current program step and then increments an internal sequence pointer so the TeachMover memory is ready to record the next step.

2. RUN

To run a program, first press the MODE key to exit from TRAIN, then press RUN. To stop a program while it's

How to RUN and STOP

running, just press STOP. The MODE light will go on, and you can press RUN again, or TRAIN, or any other control key.

At this point, it's a good idea to begin writing down your programs so you can edit them and/or use them again. One way to do this is shown in Figure 6-2.

This is a listing of a simple "pick-and-place" program. The arm moves over an object, lowers, grasps the object, moves it to another location, puts it down, and releases it.

As compared with conventional computer program listings, this kind of listing may seem like a rather informal method of program documentation - and it is. But it is important to write something down for each step number, especially for when you want to jump from one part of a program to another (see JUMP and POINT commands, below.) You'll also find that writing your programs down is a must for program editing.

STEP	POSITION
0	HOME POSITION (GRIPPER OPEN)
1	MOVE RIGHT AND DOWN
2	CLOSE GRIPPER
3	MOVE UP AND LEFT
4	MOVE DOWN
5	OPEN GRIPPER
6	MOVE UP (TO CLEAR OBJECT)
7-52	(NULL)

Figure 6-2 Listing of "Pick-and-Place" Program

3. CLEAR

This command clears all recorded arm positions and operations from program memory, and sets the sequence pointer to Step 0. You must use this command before you start entering a new program sequence.

To operate the CLEAR command:

A. Press MODE and hold the key down.

B. Then press CLEAR at the same time.

A word of caution: Do not use the CLEAR command unless you mean it! When you use CLEAR as above, your program is erased. A program can be uploaded to a host computer and saved on a disk. We'll explain how to do this in the next chapter.

4. ZERO

The TeachMover keeps track of the arm position with a set of six internal motor position registers. Each register contains the number of steps one of the six motors has taken since the registers were initialized. These registers are automatically initialized, or set to zero, when power is turned on.

In addition to setting all six internal position registers to zero, the ZERO command also resets the sequence pointer to step zero. The ZERO command lets you initialize the position register at other times as well. As with the CLEAR command, you can activate the ZERO command only if you press the ZERO key while holding down the MODE key.

At the starting point for each program, the arm is first placed in a known starting position (such as the one described in Appendix G), then the ZERO command is entered. As the arm moves, the computer keeps a count of the number of steps each motor takes.

If you place the arm in the correct initial position, but forget to use the ZERO command prior to starting a recorded program, the arm first moves to reverse the count of all the internal position registers to zero. To avoid this problem, it's a good idea to get into the habit of using the ZERO command just before running a recorded program.

5. PAUSE

This command is introduced in Chapter 2, Section E (Trial Programming). To review: If you want to program a pause into a program, perform these steps:

- Press the MODE key (if the MODE light is off.)
- Press the PAUSE key. The yellow ENTER light will come on.
- Enter a number (0-255) corresponding to number of seconds you wish the arm to pause. If you make an error in the numerical entry, you can "erase" it by pressing the REC key while the ENTER light is still on. Then enter the correct value.
- Press the MODE key again. This terminates ENTER mode.

Important : The PAUSE command is saved as a program step, and the sequence pointer is incremented.

6. SPEED

This command lets you change the speed of the arm by causing all subsequent steps and manual teach control motions to be executed at the commanded speed. The SPEED command does not get recorded as a program step. Try this:

- Press the MODE key (if the MODE light is off).
- Press the SPEED key. Yellow ENTER light will come on.
- Enter a number from 0 to 15. Zero represents the slowest speed (it's not zero speed!) and 15 represents the fastest. (The TeachMover is always initialized to speed 5 when you turn it on.) If you make an error in entering the SPEED value, you can erase it by pressing the REC key while the ENTER light is still on. Then enter the correct value.
- Press the MODE key again.

The correspondence between the speed numbers and the number of steps per second of the drive motors is given in Table 6-1. (This Table is also included in Appendix F for easy reference.)

As discussed at the end of Chapter 4, there is a maximum speed you can drive a motor before causing it to slip. For the worst-case configuration (arm fully extended, therefore requiring maximum torque), the highest speed without slipping depends on the load the arm is carrying, as indicated in Table 6-2. (For easy reference, this table also appears in Appendix F).

TABLE 6-1
STEPPING RATES FOR THE SPEED COMMAND

Speed No.	Half-Steps Per Second	Speed No.	Half-Steps Per Second
0	28	8	400
1	50	9	424
2	74	10	450
3	99	11	480
4	141	12	514
5	206	13	554
6	300	14	600
7	360	15	655

TABLE 6-2
MAXIMUM NO-SLIP MOTOR SPEEDS

Load	Half-Steps Per Second	Equivalent Teach Control Speed No.
0	400	8
Half (8oz.)	206	5
Rated (16oz.)	99	3

Teach Control Command: SPEED

Under certain conditions, motors can be operated at higher speeds without slipping. In particular:

Tips for high-speed operation

A. If shoulder, elbow, and wrist all descend, the arm may be lowered at the no-load maximum speed (400 half-steps per second, teach control speed number 8) even if it is carrying a load. However, be careful not to exceed the speed given in the table whenever lifting a load with any joint when the arm is at or near full extension.

B. The base joint may be swiveled as fast as speed number 7 even if the arm is carrying a load.

C. The hand may always be opened as fast as speed number 12.

D. The hand may always be closed as fast as speed number 10 until the hand closure contact point is reached. However, once the grip has closed, it is best to stay at or below a speed number of 6 in order to build up gripping force without motor slippage.

To gain familiarity with the SPEED command, try this:

Use of SPEED command

A. Press MODE, if necessary, then SPEED.

B. Key in a speed number of 4.

C. Press MODE, then TRAIN.

D. Use the S↑ key to move the shoulder up, then press REC.

E. Use S↓ to move the shoulder back down, then press REC.

F. Repeat A-E, but this time use a speed number of 8. Use similar arm positions to those you used in Steps D & E.

G. Press MODE, then RUN.

You should now see the shoulder move up and down slowly, then rapidly, then slowly, then rapidly, then slowly, then

rapidly,...When you're tired of watching this, press STOP. You now know how to use the SPEED command.

Experimenting with motor slippage

To see what happens when you command the arm to move much too fast, first erase the above program (press MODE, then while pressing MODE, press CLEAR), and record a new program. Make this new program the same as Steps A-E, above, but use a speed of 14. Now press RUN. What happens? That's right, nothing! The shoulder motor hums and slips, but the arm doesn't move. (In fact, you probably didn't even get as far as pressing the REC button; the shoulder simply won't lift if you command it to move that fast.)

Before we go on, there's something else you should try. Sometimes it's possible that a motor will slip somewhat - the arm will move, but not exactly to where it should. When the arm isn't fully extended, this can happen at speeds above the maximum no-load speed. Try this:

A. Turn power off and position the arm manually so that the elbow is bent to about a 45^{o} angle. Turn power back on.
B. Again key in a program following steps A-E above. Use a speed number of 9. Then RUN the program. Chances are the motor won't slip.
C. Erase the program (MODE, then CLEAR), key in a new program using a speed number of 10, and RUN this new program.
D. If you aren't aware of any motor slippage, then erase the program and do it again. This time use a speed number of 11.
E. By the time you've tried this with a speed number of 12, you'll probably notice that although the motor does lift and lower the arm repeatedly, the arm doesn't always end up in exactly the same place. Whenever you find an arm position

"drifting" like this as a program is run over and over again, chances are you've programmed the arm at too high a speed.

Once you've gained more experience with the arm, you'll get a feeling for how fast you can run it for any given application. If you find out you've keyed in too high a speed, so that a slight amount of unwanted "drift" is taking place, it is possible to change the speed number without having to erase your entire program. We'll explain how to do this when we get to the STEP and POINT commands.

7. STEP

Once all or part of a program has been recorded, it is often useful to move the arm through the program one step at a time. As we saw in Chapter 2, Section E (Trial Operation), one way to do this is to RUN the program, then press REC before the arm reaches the next programmed position. However, two key presses are required for each step (RUN, then REC), and you can accidentally bypass a programmed arm position altogether.

A simpler method is to use the STEP command. First press the MODE key, if necessary, then press STEP. This moves the arm to the next programmed position. Press STEP again, and the arm moves to the next position again, and so forth.

Let's examine more closely how programs are stored. Try this:

- Press CLEAR while holding down the MODE key to clear the memory.
- Record any three arm positions you wish. List them in order on a piece of paper.
- RUN the program for a few cycles, then STOP the arm somewhere between the first and second positions.
- Now, press STEP. The arm continues to the second position and stops.
- Press STEP again. The arm moves to position 3.
- Press STEP a third time. Nothing happens! Why? Because your program only use 3 steps of the 53-step memory; you've just reached the fourth step, which is null. You'll have to press the STEP key 50 more times to get back to Step 0. (Try it - it won't take long.) Actually, this same phenomenon occurs whenever you RUN a program. The sequence pointer goes through all 53 steps, but the RUN

Unused program steps

command runs it through the null steps so fast that you don't notice a pause.

Using STEP for program editing

The other thing you should know about the STEP command is how to use it for program editing. To change an already-recorded arm position, simply STEP through the program until you reach the position you wish to change. Switch over to TRAIN mode, move the arm to the correct position, and then press the REC key. This overwrites the old position.

This editing procedure works because the STEP command increments the sequence pointer before executing a program step. In other words, when the STEP command brings the arm to a given position, the sequence pointer remains pointing at the program step for that position even while the arm position is being changed. Changing the arm position simply changes the contents of that step location in memory.

When STEP doesn't increment the sequence pointer

[Note: If you change an already-recorded arm position using the above procedure and then press STEP again, the arm will move to the next recorded position as usual, but this time without incrementing the sequence pointer. This is because the REC command already did the incrementing. As we'll see later, a similar situation occurs when you step through a JUMP command or when you use the STEP or RUN commands after stopping or immediately after a POINT command.]

The POINT command, which we'll discuss soon, provides an alternative method of accessing program steps for editing. Further details and a summary of both methods are given in Part C of this chapter.

8. JUMP

This command lets you write rather sophisticated arm-motion programs by allowing for conditional branching - one of the most powerful features of classical computer programming. The JUMP command tests the user input bits on the Auxiliary I/O connector (P17) discussed in Chapter 5, Section C. Here is how it works: When you press the JUMP key, the yellow ENTER light comes on, and you enter not one, but two numbers. The MODE key is pressed after each number to allow the computer to store each number separately. A number may have more than one digit requiring you to press more than one key to enter the number.

- The first number identifies the jump condition.
- The second number identifies the programmed step to jump to if the jump condition has been met.

The jump conditions are as follows (These are also given in a table in Appendix F for ready reference):

- Condition 0: Grip switch is open
- Condition 1: User input bit 1 is on (ie. set to one)
- Condition 2: " 2 "
- Condition 3: " 3 "
- . " . "
- . " . "
- . " . "
- Condition 7: " 7 "
- Condition 8: Never (an effective NULL)
- Condition 9: Always (unconditional jump)

This may sound complicated, but an example should make it clear. Let's say you want the arm to move to Step 5 in a program on the condition that the grip switch is open. All you need to do is:

Using the JUMP command

- Press MODE (if necessary), then JUMP. Yellow ENTER light will go on.
- Press 0. (Actually, it isn't really necessary to press 0. The ENTER value is preset to ZERO automatically. Any entry other than 0, however, would have to be entered.)
- Press MODE.
- Press 5.
- Press MODE.

From now on, instead of writing out all these steps, we'll use the short-hand notation: JUMP 0,5.

To jump to a step, say Step 23, regardless of the status of the grip switch or any other factor (this is called an unconditional JUMP), just key in:

JUMP 9, 23

The user input bits can be set in a variety of ways. For example, sensor switches might be mounted on the TeachMover's fingers or signals might be generated by external machinery.

Another example of when the STEP command doesn't increment the sequence pointer

Note : When you STEP through a JUMP command, the usual incrementing of the sequence pointer is slightly modified. For example, let's say you STEP to the command, JUMP 9,7. This unconditional jump sets the value of the sequence pointer to 7. If you press STEP again, step number 7 gets executed, but the sequence pointer is not incremented. If the pointer were incremented first , as it usually is with the STEP command, then step number 7 would be skipped, and step number 8 would be executed instead. On subsequent pressings of the STEP key, the sequence pointer is incremented first as usual.

Teach Control Command: JUMP

One of the two demonstration programs built into the TeachMover makes extensive use of the JUMP command. We'll discuss this program after we've outlined the remaining control commands.

9. POINT

In a way, the POINT command is similar to an unconditional JUMP. For example, POINT 12 means go to Step 12 in a program and proceed from there. However, unlike the JUMP command, POINT does not create a program step. POINT is used simply to move to a given step in an existing program. It can be invoked even in the middle of program execution by pressing the MODE (STOP) key first. Try it:

Use of POINT command

- Enter, say, a 5-step arm-motion program. Write down the steps.
- RUN the program.
- STOP just before the last position is reached.
- Press POINT. The yellow enter light goes on.
- Enter the number 2.
- Press MODE, then RUN.

Did the arm go to the position you expected it to? If not, did you remember that program Step Number 2 is really the third step, because the sequence pointer numbers start with 0? If necessary, try the above exercise again. The POINT command is very useful, but only if you POINT exactly where you want to!

Using POINT for program editing

One of the most useful applications of the POINT command is program editing. Instead of using the STEP command to go to a program step you want to change, just POINT to the corresponding program step number. Then press MODE and TRAIN (or other appropriate command), and enter the new program step. Keep in mind that the STEP command points to the step and executes it. Thus, when you use STEP for editing, the program step you change is the one the TeachMover just executed. However, the POINT command points to the step without executing it. Your written program list can remind you what the step does.

If you find it confusing to keep this distinction in mind, there's an alternative: You can press STEP immediately after a POINT command; this executes the step number pointed to, but does not increment the sequence pointer.

Use of POINT followed by STEP for program editing

Example: If you enter the command POINT 2, followed by STEP, the sequence pointer remains at 2 while step number 2 gets executed. If this did not occur - that is, if the STEP command were to increment the pointer first, as it usually does - then the STEP command would skip over step number 2 and cause step number 3 to be executed instead. Naturally, on subsequent pressings of the STEP key, the sequence pointer continues to be incremented first as usual.

Note: There is one case in which you do not want to use POINT followed by STEP for editing; this is when the step number being pointed to contains a JUMP command. If you point to a JUMP and then press STEP, the JUMP gets executed without first giving you a chance to modify it!

We'll review how to use STEP and POINT for editing in Part C of this chapter.

Executing multiple programs

The POINT command can also be used to execute multiple programs stored in memory. You could, for example, have three distinct programs recorded as shown in Figure 6-3. You can isolate the programs by means of unconditional JUMP commands returning the pointer to the beginning of the program.

The program listing would show:

```
Step 0 _________; start program 1
   .       .
   .       .
   .       .
Step 15 JUMP 9,0; return to start of program 1
   .       .
   .       .
   .       .
   .       .
Step 21 _________; start program 2
   .       .
   .       .
   .       .
Step 31 JUMP 9,21; return to start of program 2
Step 32 _________; start of program 3
   .       .
   .       .
   .       .
Step 38 JUMP, 9, 32; return to start of program 3
```

To execute Program 2, all you need to do is enter:

```
POINT 21
```

prior to pressing the RUN key. Similarly,

```
POINT 32
```

executes Program 3.

The POINT command has still another application. In creating a program, sometimes it is desirable to make an exact "copy" of a program step - for example, to key in a program step that causes the arm to move to a position it already achieved elsewhere in the program. Here's how to do this:

Using POINT to "copy" a program step

- STEP through the program until the desired arm position is achieved.
- POINT to the program step to contain the position to be copied.
- Press MODE, then TRAIN, then REC.

This duplicates the desired position at the desired program step. Try it.

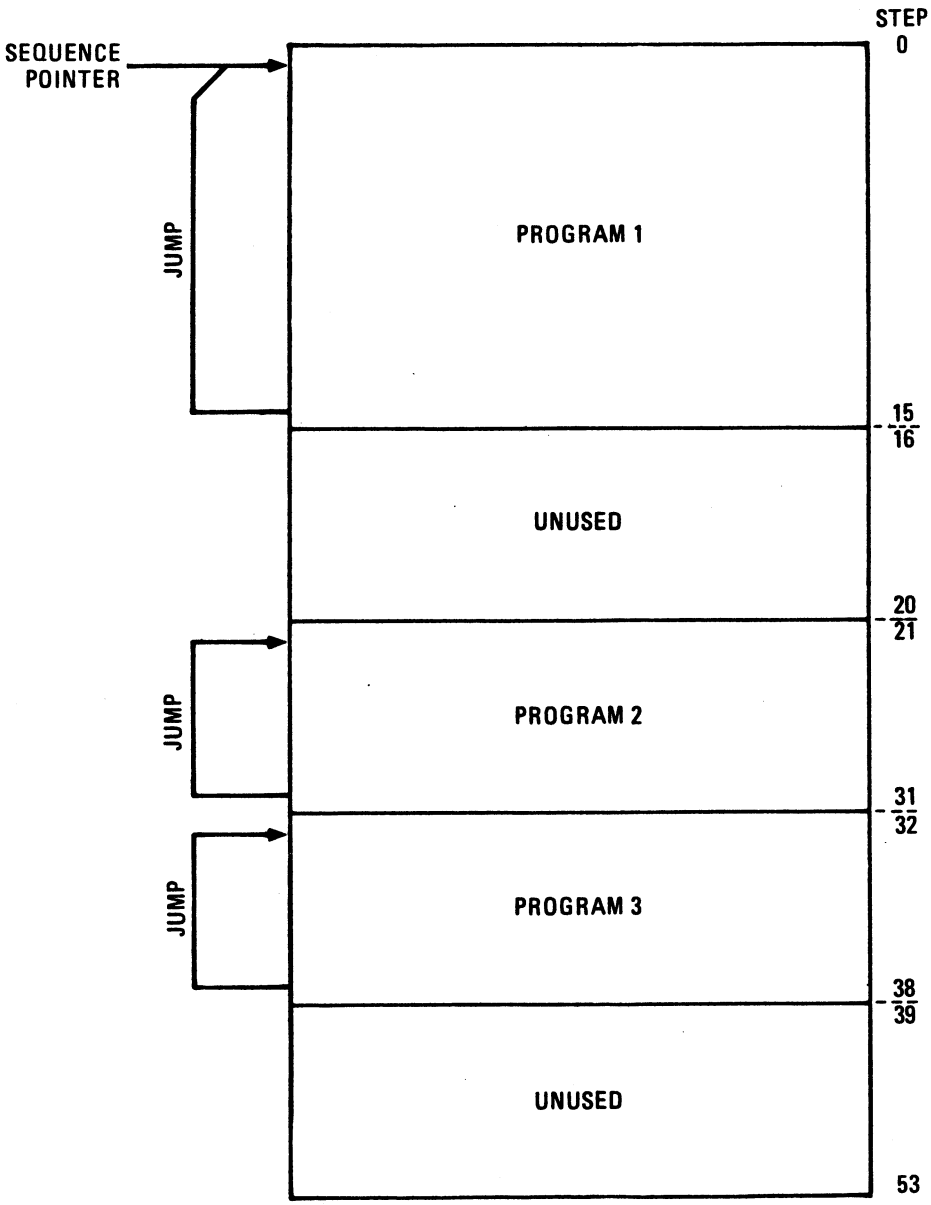


Figure 6-3 Multiple Program Storage

10. **GRIP**

This command will cause the gripper to close 32 half-steps of the motor past the point at which the grip switch is activated as the gripper first closes. This builds up about 1 lb. of gripping force.

Distinction between "G" keys and GRIP command

This command is different from using the G keys in TRAIN mode. The G keys will simply command the fingers to open or close to a particular spacing, regardless of whether the hand is holding an object. The GRIP command, on the other hand (sorry for the pun!), closes on an object and then builds up 1 lb. of gripping force regardless of the size of the object. Thus, the GRIP command is useful when you want the arm to pick up a variety of objects, whereas the G keys are a better choice when you want the arm to sense whether a particular object is present.

[Note: Once in a while, the grip switch may fail to operate, or the GRIP command opens the gripper instead of closing it. See page 3.14, Hand Drive, for simple remedies.]

11. MOVE

This activates the joint control keys used in TRAIN mode, but does not change the internal position registers or allow a position to be recorded. In other words, if you press MODE, then MOVE, then one of the joint control keys, then REC, you'll find that the REC key has no effect.

The MOVE command proves useful in moving the arm to a known initial position in the event of motor slippage or mechanical interference from an external obstacle. The procedure is:

Using MOVE to initialize the arm

- Use the ZERO command. (Remember, this sets the internal position registers and the sequence pointer to zero.)
- Use the MOVE command.
- Use the arm-motion keys to achieve the correct initial configuration. (For very precise position control, first use the SPEED key to specify a speed number of 0.)

Note that this does not change the recorded program in any way; it simply lets you start the program again with the arm in its correct initial position.

In some cases, you may want to reinitialize the arm by returning it to a known position other than the initial position. The ZERO command is not useful here. Instead, follow these steps:

Using MOVE to return arm to known position

- STEP (or POINT then STEP) until the arm reaches the position you wish to use for initialization.
- Use the MOVE command.
- Use the Arm-motion keys to achieve the correct position. For very precise position control, fist use the SPEED command to specify a speed number of 0.

When you subsequently RUN the program, all settings of the internal position registers are associated with the recalibrated new initialization position.

Using MOVE to tighten grip

A similar use for the MOVE command is if an object is slipping from the gripper because the gripper isn't holding it tight enough. Instead of reprogramming, you can just STOP the program, use the MOVE command, use the G key to close the gripper a bit, then RUN. All subsequent settings of the internal position registers will now be associated with the new grip setting.

12. **FREE**

FREE is similar to the MOVE command, except it turns off all motor currents to allow you to position the arm manually. In many cases, it is simply a matter of preference whether you use MOVE or FREE. However, as mentioned earlier, you will probably find that you get finer position control when you use the MOVE command with a speed setting of 0, rather than moving the arm manually in FREE mode. Try both and see.

13. OUT

This is the command that lets you turn external equipment on and off based on arm positions achieved or conditions met. You can also use it to turn on and off various lights on the hand-held teach control.

When you press OUT, the yellow ENTER light will come on, and you must give two numerical entries (pressing the MODE key in between). The first entry is an output number (see below), and the second is 0 or 1. (In the case of the teach control lights, 0 indicates "off", and 1 indicates "on".) The output numbers are as follows. (These are also given in a table in Appendix F for easy reference.)

TeachMover output numbers

Output 0: The MODE light.
Output 1, 2..., 5: User outputs on the I/O connector.
Output 6: The TRAIN light.
Output 7: The RUN light.
Output 8: The ENTER light.

Here's a sample program you can try:

Using the OUT command

Step No.	Operation
0	OUT 8, 1
1	PAUSE 2
2	OUT 8, 0
3	PAUSE 2
4-52	(NULL)

When you RUN this program, you should see the yellow ENTER light blink on and off at 2-second intervals. Didn't work? Then check the following and try again:

a. Did you press the MODE key between the two numerical entries for each OUT command? (For example, program step 0 should be keyed in as OUT, 8, MODE, 1).

b. Did you press MODE after each program step to exit from OUT and PAUSE? (In other words, each program step should be followed by a pressing of the MODE key.)

As you can see, the basic idea is simple, but you have to be careful with those buttons! Now is a good time to practice some more. See if you can create programs that will:

a. Turn the TRAIN light on and off at three-second intervals.

b. Turn the ENTER light on for three seconds, then off for one second.

c. Turn the MODE light on for 2 seconds, then turn the ENTER light on for 1 second without turning off the MODE light, then turn both lights off for one second.

d. Flash the lights in sequence from top to bottom, leaving each light on for one second, with no pause between lights.

C. DEMONSTRATION PROGRAMS

We have permanently recorded two demonstration programs in the TeachMover's read-only memory. One of these programs begins at program step 126. It exercises all the TeachMover joints, and, in fact, is used to test each TeachMover before shipping. The second demonstration program begins at step 174, and causes the arm to first determine which of two blocks is larger, then move the larger one to a new location and stack the smaller on top of it. To run these programs, you first need to initialize the arm in a particular way. Here are the details:

Remember to first initialize the arm

The proper initialization position for the TeachMover and a block to be manipulated is shown in Figure 6-4. Refer to Appendix B for a reference grid and details of the initialization procedure.

1. EXERCISER PROGRAM

A 1-1/2 inch cube is placed at P3 with its edges aligned with the x and y axes. Once the arm is in its starting position, and the block is at P3 position, set the internal position registers to zero by turning on the power, or, if power is already on, by using the ZERO command; then:

a. POINT to 126

b. RUN

A listing of the program is given in Figure 6-5. Note what happens if you don't position the cube at P3: The hand signals to you by waving a few times, then the arm tries again (see program Step 143 and Steps 167-173).

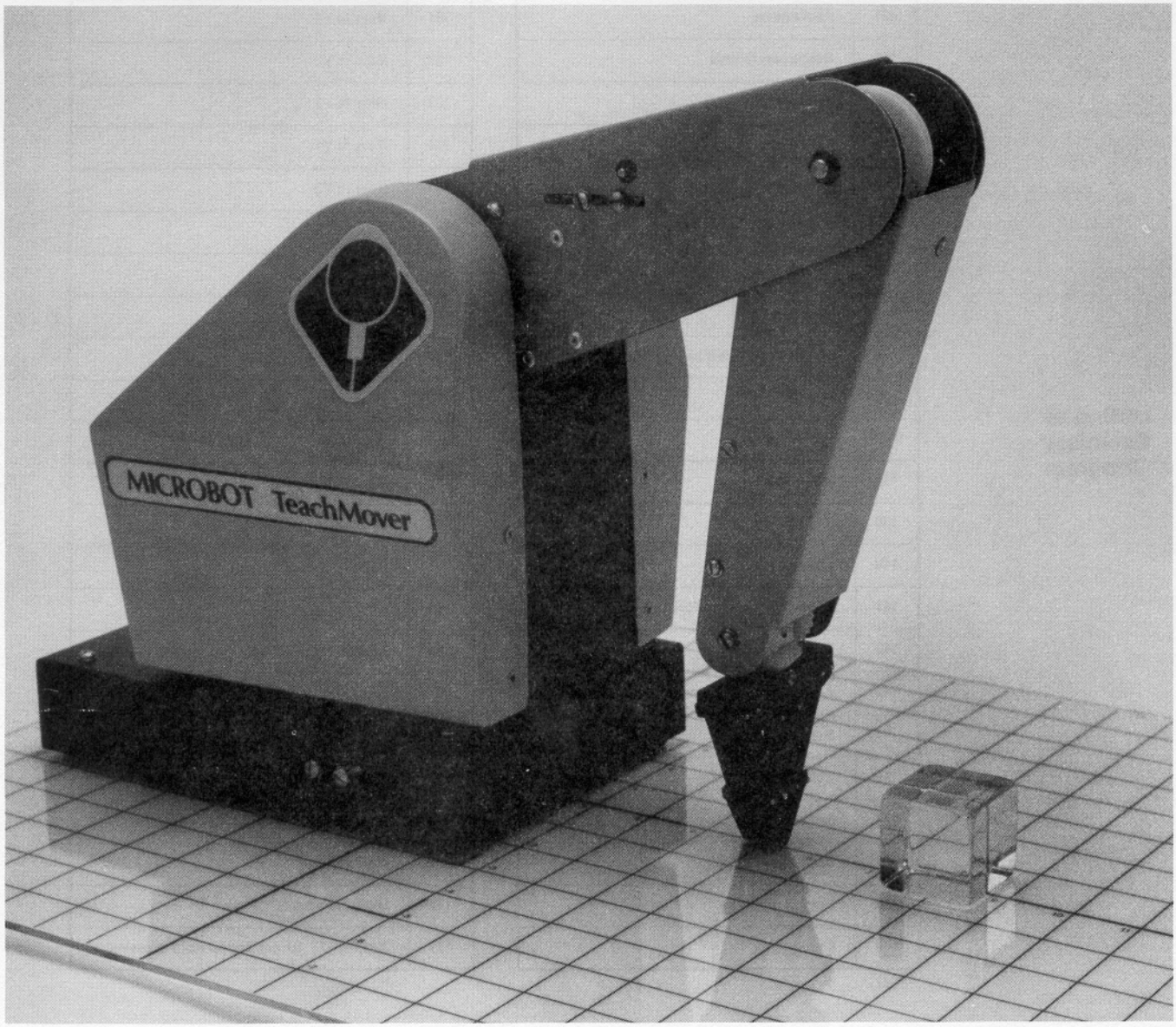


Figure 6-4 Initialization Configuration for Exerciser Program

Listing of Exerciser Program

STEP	POSITION
126	Home position; ZERO command
127	Above home
128	Rotate base to limit
129	Move hand out
130	Put hand in the air
131	Open hand
132	Close hand
133	Pause 2 seconds
134	Move hand back
135	Spiral down to other base limit
136	Hand in
137	Base to center
138	Above home
139	Home slowly
140	Up and open
141	Forward to P3; grip block
142	Close gripper
143	Jump 0, 167 (If grip switch is open, go to 167)
144	Move up
145	Roll and flip block
146	Set down block
147	Open
148	Pivot up
149	Close

STEP	POSITION
150	Move up
151	Move to Y1
152	Move to Y2
153	Move to Y3
154	Move to Y4
155	Move to Y5
156	Move to Y6
157	Move to Y7
158	Move to Y8
159	Move to Y9
160	Move to Y10
161	Move to Y11
162	Move up and rotate
163	Set down
164	Open
165	Move up
166	Jump 9, 126 (Unconditional jump to beginning)
167	Up to wave and close
168	Wave up
169	Wave down
170	Wave up
171	Wave down
172	Move above home
173	Jump 9, 126 (Unconditional jump to beginning)

Figure 6-5 Exerciser Program

Demonstration Program: Exerciser

Although this program is listed using the format for a teach control program listing, the program was, in fact, written on a host computer and then downloaded and permanently stored in the TeachMover's read-only memory. This is important for you to know if you try to duplicate the program on your own using the hand-held teach control. You will find one place where you won't be entirely successful. This is in steps 151-161, where motion along a straight 10-inch line segment is approximated by a series of 11 steps spaced one inch apart. When you run the program, notice that the arm doesn't accelerate or decelerate between each of the intermediate steps as it would if programmed via the hand-held teach control. Instead the arm maintains its speed throughout the 10-inch motion. This is accomplished using a special "PATH" operation that is available only in serial interface mode, and then only for programs that are being downloaded to the TeachMover. We will go into this further in Chapter 7.

"No Deceleration" feature

Except for this "no acceleration" feature, it is possible to duplicate - or at least approximate - the above Exerciser Program by recording program steps on the teach control, and you may wish to try doing so as an exercise. The Exerciser Program contains frequent changes in arm speed, that are not shown in the listing. Since this program is designed to exercise the arm to its various limits, each arm motion is programmed at the highest speed achievable without motor slippage. If you do try "copying" the Exerciser Program, use a speed number of 7 or less throughout; step 139 can be done at a speed number of 4.]

2. BLOCK STACKING PROGRAM

The initialization configuration for this program is shown in Figure 6-6. The arm is placed at PO and wrist-cable turn-buckles aligned as in the Exerciser Program. Two cubes - a 1-1/2 inch cube and a one-inch cube - are placed at Pl and P2 on the reference grid of Appendix G. The positions of the two cubes may be interchanged, and one or both cubes can even be omitted, but if they are present, their edges must be aligned with the X and Y axes.

To activate the program:

- POINT 174
- MODE
- RUN

A flowchart of the program is given in Figure 6-7. The numbers 174, 175, etc. refer to program steps. Note the extensive use of conditional branching.

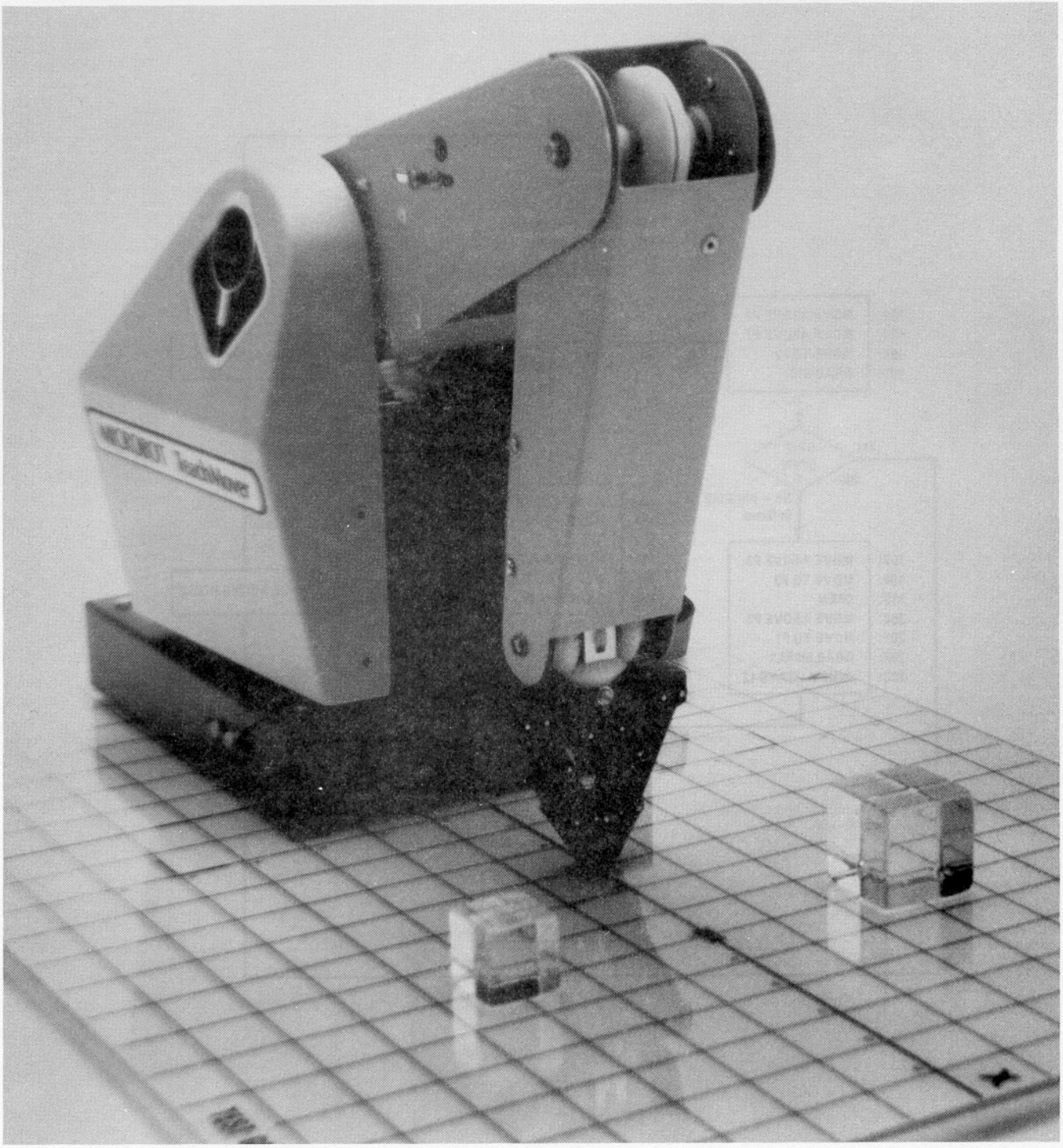


Figure 6-6 Initialization Configuration for Block Stacking Program

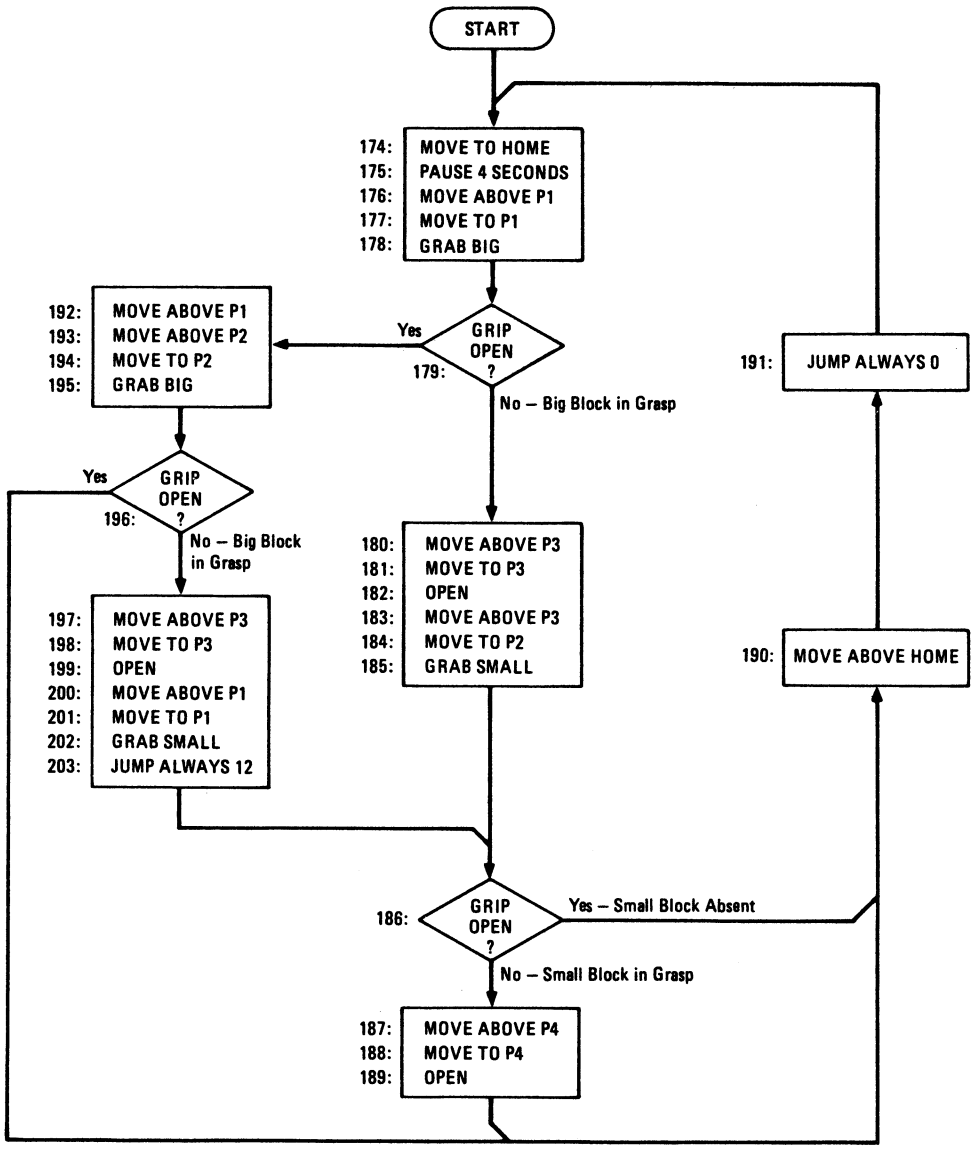


Figure 6-7 Flowchart of the Block Stacking Program

D. PROGRAM EDITING

As discussed under the STEP and POINT commands earlier, there two different methods for editing programs. To review, let's say you want to change the arm position to be achieved by the 23rd step of a recorded program. You can choose either:

Two methods for program editing

Method 1. STEP or, to save time, POINT, then STEP through the program until the 23rd step gets executed, then:

- Press TRAIN.
- Use the arm-motion keys to achieve the correct position.
- Press REC.

Method 2. POINT to program step number 22 (not 23! Remember, the first step is step 0.) Then follow steps a-c above.

If, instead of an arm position, the new program step is a PAUSE, OUT, GRIP, or JUMP, then in step a of the above procedure, use PAUSE, OUT, GRIP, or JUMP instead of TRAIN, followed by the correct numerical entry (or entries). Also, when editing a JUMP command, always use Method 2 above (POINT command) rather than Method 1 with the STEP command, because stepping to a JUMP will cause the JUMP to execute.

E. EXPERIMENTATION

You have now learned how to use all the teach control commands and how to write and edit teach control programs. Before going on to learn how to operate the TeachMover from a host computer, it is important to gain experience by writing several teach control programs of your own. Here are some suggestions:

1. Start with two stacked blocks and have the TeachMover unstack them.
2. Have the TeachMover pick up a block at P0 and move it either to P1 or P2, depending on its size.
3. Have the TeachMover pour water from one cup into another without spilling any.
4. Try programming some ideas of your own.
5.
6.

CHAPTER SEVEN
OPERATION FROM A HOST COMPUTER

Connecting the TeachMover arm to a host computer or a terminal greatly extends the unit's capabilities. As we will see, use of the TeachMover's serial interfaces allows you to write programs that specify arm positions by means of Cartesian coordinates, programs that actually measure the position and thickness of an object, and much more - all without losing the ability to program the arm from the hand-held teach control.

A. CONFIGURING THE SERIAL PORTS

"Configuring the serial ports" refers to making sure that your computer and the TeachMover can "talk" to one another. This requires taking care of the following:

Six items needed for configuring serial ports

1. electrical connections
2. transmission rate
3. data format
4. settings for standard interface signals
5. opening the port
6. testing the configuration

Depending on the computer you're using, configuring the serial ports may be straightforward and simple, or it may be complex. We'll start with the most basic steps first, then proceed to more intricate details.

If you follow the procedure given below, and yet the arm won't respond to serial port commands, chances are that some of the details of configuring your computer are not correct. If this seems to be the case, carefully review the user manual that came with your computer or call a customer service representative employed by the computer manufacturer. The configuration steps begin on the next page.

1. Electrical connections

In the back of the TeachMover's base you'll find two multi-pin connectors (Figure 7-1). These are the two serial ports.

Functions on left and right serial ports

- Signals that enter the left port (P2) always pass through to the right port (P1) unchanged.
- Signals that enter the right port pass through to the left port unchanged, unless the signals are a series of characters beginning with an "@" sign and terminating with a <CR> (carriage return); these signals are not passed through, but are interpreted as arm commands. (As we'll see later, one of the serial interface commands lets you specify a different recognition character in place of the "@" sign.)

Thus, to operate the arm from a host computer or a terminal, connect the computer or terminal to the TeachMover's right serial port (Figure 7-2 a. and b.). You would use the left port in either of the following two situations:

a.) TeachMover in Series with computer and other peripheral

Some host computers only have one serial port of their own and thus cannot be directly connected both to the TeachMover and to a peripheral unit (terminal or serial printer) simultaneously. To overcome this limitation, simply connect the computer, the TeachMover, and the other peripheral in series, as shown in Figure 7-2c. Signals coming from the peripheral will pass through to the computer unchanged, and signals coming from the computer will either pass through to the peripheral unchanged, or, if preceded by an "@" sign, will operate the TeachMover.

Configuring Serial Ports: Electrical Connections

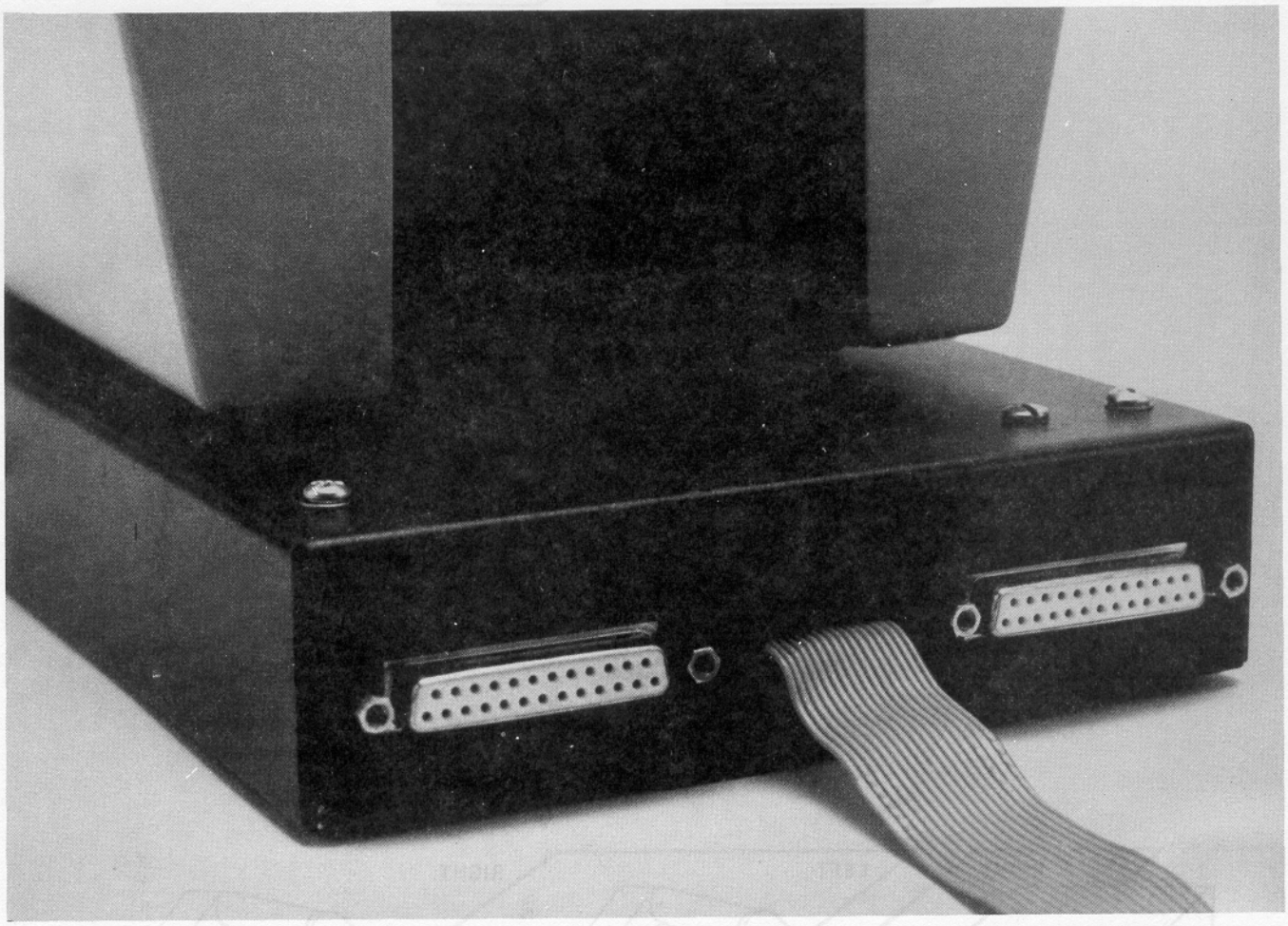


Figure 7-1 Two Serial Ports

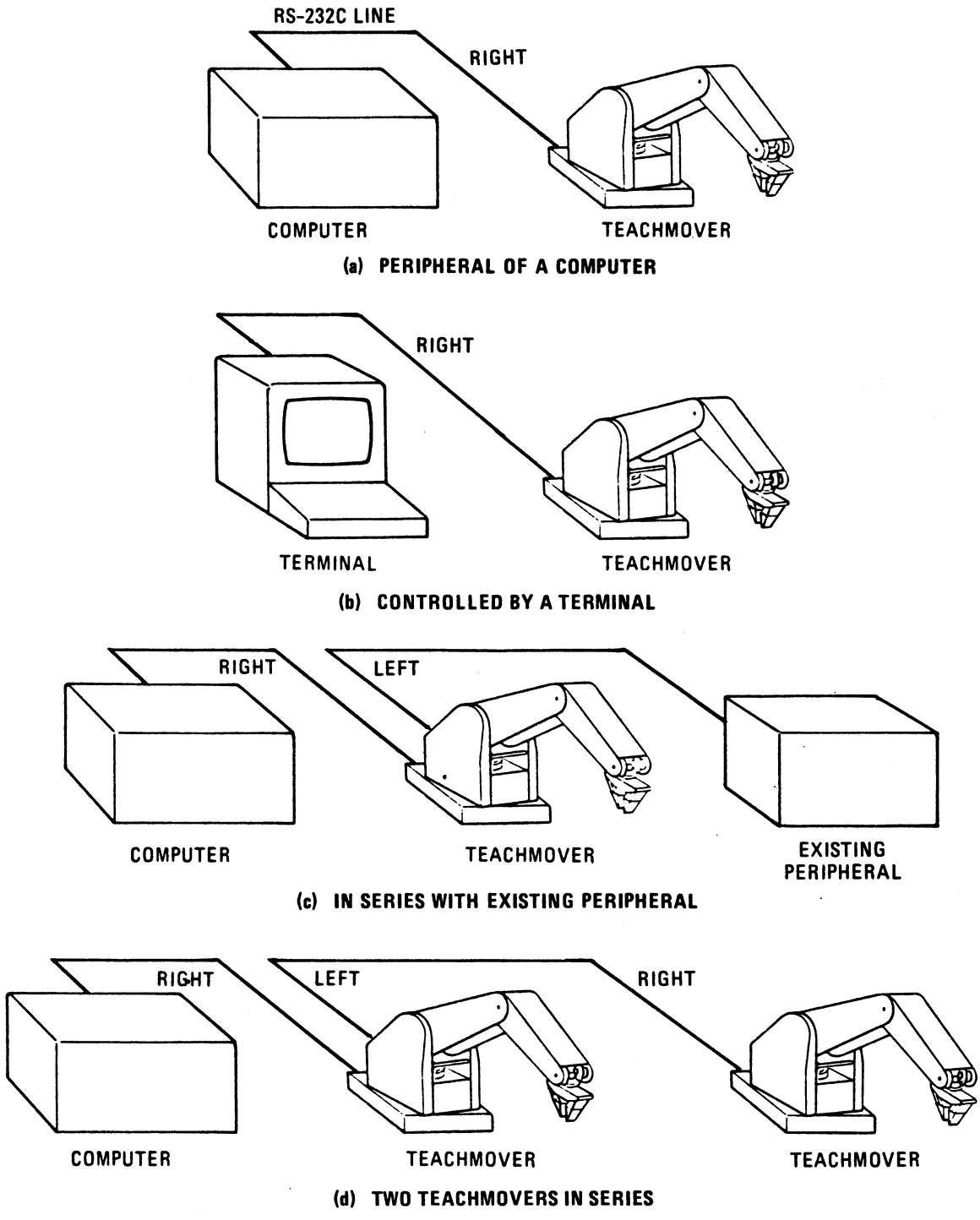


Figure 7-2 Connecting the TeachMover to a Computer and/or Peripheral

b.) Two or more TeachMovers in Series

You can also connect two or more TeachMovers in series and operate them from the same computer (Figure 7-2d). In such cases, it is useful to program each of the Teach-Movers to respond to a different recognition character; this can be accomplished by means of the @ARM command, as will be explained later.

Pins two and three transmit and receive

When you use a serial line to connect the TeachMover to a computer or peripheral, it is important that the transmit and receive lines be interfaced properly. On the right port of the TeachMover, the receive line is on pin 3, and the transmit line is on pin 2. (See Figure 7-3 for pin numberings.) Most computers are configured with pin 3 as transmit and pin 2 as receive, so in most cases a standard serial cable will provide the proper straight-through wiring; i.e., pin 2 to pin 2 and pin 3 to pin 3 (see Figure 7-4).

However, it is possible that the pin assignments have been reversed on your computer. This can happen if your computer has been configured to behave as a terminal; in other words, if it has been wired to another computer. This is because in order to wire two computers together, the transmit and receive lines must be crossed as shown in Figure 7-5. If your host computer has been configured to behave as a terminal, you must reverse its transmit and receive lines so they are as in Figure 7-4. You can accomplish this by modifying the cable or by reconfiguring the computer internally, as per instructions in the use manual that came with your computer.

[Note: On the TeachMover's left port, the transmit line is on pin 3 and the receive line on pin 2. This is because the TeachMover acts, in effect, as a host to whatever peripheral is connected to its left port, and most peripherals use pin 3 for receive and pin 2 for transmit.]

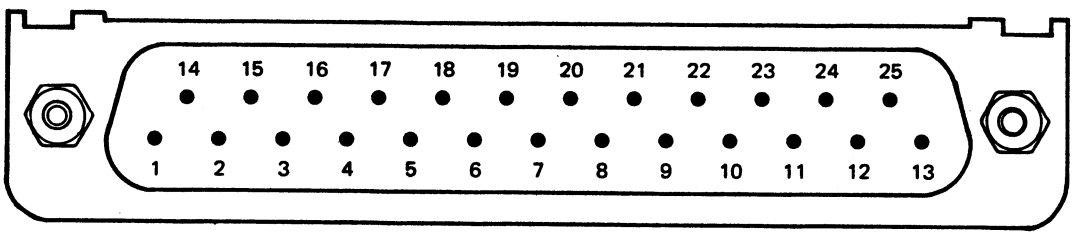


Figure 7-3 Pin Numbering for Serial Port Connectors

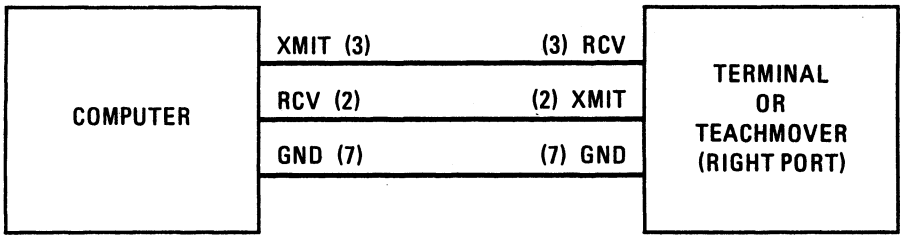


Figure 7-4 Computer-to-Terminal Serial Connection

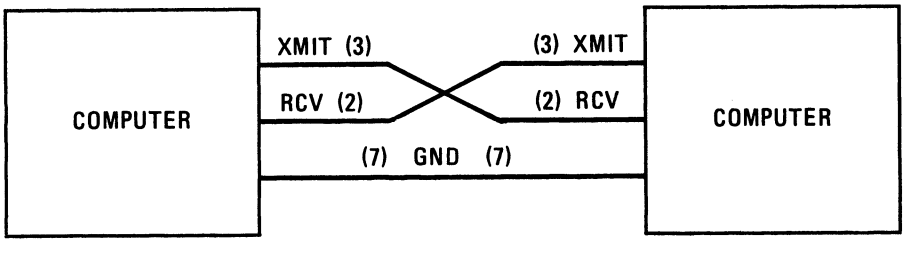


Figure 7-5 Computer-to-Computer Serial Connection

2. Transmission Rate

The TeachMover is shipped with both serial ports configured to operate at a transmission rate of 9600 baud (9600 bits per second), for both send and receive. You can change this rate to any of seven other standard rates by means of three switches located on the TeachMover computer card (Figure 7-6). The available rates and the corresponding switch settings are given in Table 7-1. (This table also appears in Appendix F for easy reference.) These switches should be changed when power is off, since the switch settings are read by TeachMover firmware on power-up only.

Baud rate switches on the TeachMover

As with the TeachMover, most computers have some means of setting baud rate - either through switches on a circuit card, or via commands that can be issued under the computer's disk operating system (DOS), or as part of BASIC. The main thing is that both the TeachMover and your computer be configured to operate at the same baud rate; otherwise communication between the two will be impossible.

Setting the baud rate

Table 7-1
BAUD RATE SELECTION

BAUD	SW1	SW2	SW3
110	ON	ON	ON
150	OFF	ON	ON
300	ON	OFF	ON
600	OFF	OFF	ON
1200	ON	ON	OFF
2400	OFF	ON	OFF
4800	ON	OFF	OFF
9600	OFF	OFF	OFF

NOTE: SW4 is not used.

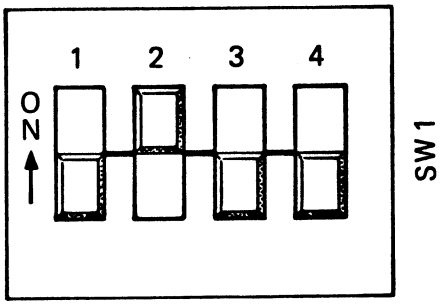


Figure 7-6 Switches for Selecting Serial Transmission Rate

3. Data Format

The TeachMover uses the following data format:

word length = 8 bits
1 start bit
1 stop bit
no parity bit
Full duplex

Word length, stopbit(s) and parity

Many computers have the above as their "default" format - that is, the format in which data will be transmitted if you do nothing special to configure the format. However, with other computers you will need to configure the format explicitly. Consult your computer manual to learn how to do this.

[Note: Configuring data format is not always a simple task. On at least one popular microcomputer, for example, rather than specify a word length of 8, it is necessary to specify a word length of 7 plus a parity bit equal to zero. This is because this computer uses a most significant bit equal to 1 when processing 8-bits words. (It is possible to use the @ARM command to allow the robot to recognize an "@" with the eighth bit = 1, but in order to execute this command the robot must recognize the first "@" in this @ARM command. To do this the robot must receive an "@" character with the most significant bit = 0.)]

4. Standard Interface Signals

Some computers and terminals require logic levels on certain pins to indicate the following status conditions:

Four common communication signals

Data Terminal Ready
Clear to Send
Carrier Detect
Request to Send

The TeachMover does not use these signals, but does pass them through when it is placed in series between a computer and a terminal.

However, when only a single computer or terminal is connected to the TeachMover (or in some cases even if the TeachMover is placed in Series between a computer and a terminal), you may need to modify the TeachMover in order to provide these signals.

To find out if this is necessary, consult the user manual for your computer to determine whether any of the four above-mentioned serial port signals are required. (With some computers, the user has control over whether these signals are required. If this is the case with your computer, then configure the computer so that these signals are not required.) If any of these signals are required, then, if you have a peripheral in series with the TeachMover, check the user manual that came with that peripheral to see whether the peripheral supplies the required signals for transmitting and receiving data. If it does, then all should be well. If not, then you will need to modify the TeachMover.

When to solder jumpers W1, W2, W3 and W4

The modification procedure is simply to solder a jumper across the appropriate terminals on the TeachMover circuit card as shown in Figure 7-7, using the information

given in Table 7-2; the jumpers are labelled W1, W2, W3, W4. (Note: soldering a jumper will have the effect of permanently setting the corresponding signal to logic level 1, or "on.")

TABLE 7-2

Auxiliary Control Lines

Left Port Pin No.	Description	Right Port Pin No.	Jumper
8	Data Carrier Detect	8	W1
1,7	Ground	1,7	--
3	Transmit from TeachMover	2	--
2	Receive by TeachMover	3	--
4	Request to Send	4	W4
5	Clear to Send	5	W3
20	Data Terminal Ready	20	W2

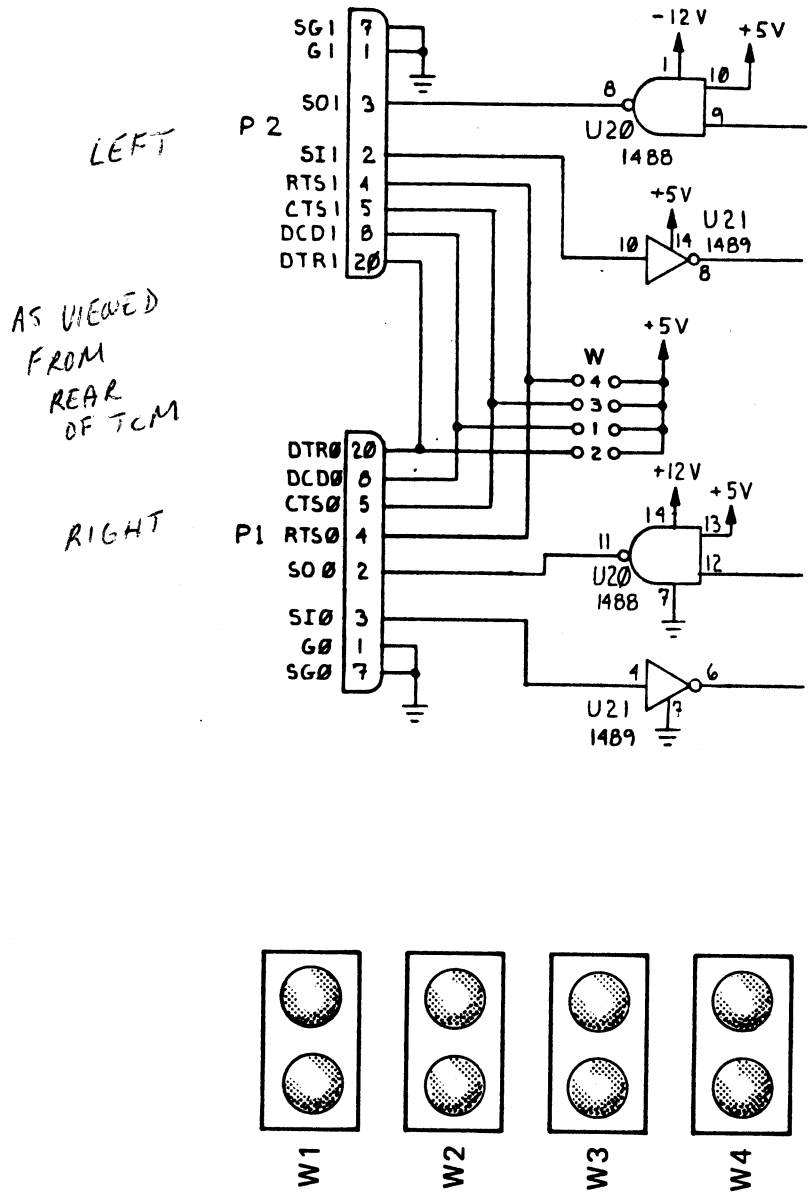


Figure 7-7 Location of Jumper Connections for Serial Operations

5. Opening the Port

This step refers to configuring your computer so that commands and data are properly routed from your computer to the TeachMover. If your computer has only one serial port, there may be nothing special to do other than use the proper BASIC commands for input and output through that port. On some computers, you might also have to use a special routing command or a switch setting to transmit to the TeachMover whatever would normally go to a parallel printer. (In such cases, you could not use the printer and the TeachMover at the same time.)* Some computers have several serial ports, and allow you a choice of transmission channels.

In all cases, consult your computer manual to find out what is required.

---

* This is true for at least one of the well-known personal computers. Morever, this computer does not have a command for transmitting more than one item of data over the serial line. Since the TeachMover's commands are strings of data items, special routines must be written to provide for this.

6. Testing the configuration

Once steps 1-5 above are completed, you are ready to test the serial configuration. This is best accomplished by issuing an "@CLOSE" command. This command closes the gripper until the grip switch is activated. We'll go into the details of this and all the other commands in the next section, but for now it is important to know how the TeachMover responds to arm commands in general.

When an "@" sign or other programmed recognition character is received at the TeachMover's right port, all subsequent characters are buffered (temporarily saved) inside the TeachMover until a carriage return, <CR>*, is received. The buffered characters are interpreted as an arm command.

"Handshaking" details:
[0 <CR>]
[1 <CR>]

- If the intercepted command is not syntactically correct, the TeachMover will return a zero (in industry-standard ASCII format) followed by a carriage return signal. We'll designate this as [0<CR>].
- If the intercepted command is syntactically correct, the arm will execute the command, then will signal completion by means of a 1 (ASCII) followed by a carriage return signal. We'll designate this as [1<CR>].

This sending back of [0<CR>] or [1<CR>] is called "handshaking." After every arm command, it is necessary to input the handshake character into the computer so the computer knows that the arm is ready for the next command. In addition to simply inputting the handshake character, it's a good idea to actually test whether it's a 0 or a 1 before issuing the next command. One way to do this is by means of the following subroutine.

---

*Sometimes labled "RETURN" or "ENTER" on the keyboard.

[Note: In this and all other examples in this chapter, we will assume that your computer uses a BASIC-like language, and that you know how to write programs in that language, including all input/output procedures. Since there are many variations of BASIC on the market, we will use the imaginary instructions "INPORT" and "OUTPORT"to represent the commands you normally would use for transmitting information between your computer and a peripheral unit over serial lines. INPORT causes the computer to receive information, and OUTPORT causes the computer to transmit information. In most cases, INPORT and OUTPORT are simply modified INPUT and PRINT statements. Typical commands might be INPUT #3, LPRINT, etc. When you enter one of our sample programs into your computer, be sure to replace INPORT and OUTPORT with the proper command syntaxes.]

INPORT and OUTPORT

Here is the "handshaking" subroutine you can use to test for a syntax error or for when an arm command has been executed:

"Handshaking" subroutine

```
6010      INPORT I
6020      IF I = 0 GO TO 6040
6030      RETURN
6040      PRINT "INVALID COMMAND"
6050      STOP
```

To call up this subroutine place the statement

```
GOSUB 6010
```

after each arm command.

If you do not wish to actually test for whether the returned character is a 0 or a 1, you still must input the character, and can do so simply by placing an INPORT I statement after each arm command. In all subsequent examples in this manual, we will simply use INPORT I, mainly to avoid clutter in our program listings.

Now here is the program you would use to issue an @CLOSE command to test your serial configuration. (Be sure the gripper is open before running the program.)

Serial port test program

```
10        OUTPORT "@CLOSE"
20        INPORT I
30        PRINT I
```

If all is well, then when you run the program two things should happen: the gripper should close, and a 1 should appear on your screen. If neither of these happens, then carefully review every step of the above procedure. The following checklist will help:

Serial port configuration checklist

1. Does your computer's transmit line connect to the TeachMover's receive line and vice versa?
2. Is the baud-rate setting of the Teach-Mover and your computer the same? (If you're using a high baud rate, try a lower one--300, for example. Sometimes too high a baud rate can cause problems. We'll discuss this further when we explain the "@DELAY" command later on.)
3. Have you configured your computer with the proper data format (1 start bit, one stop bit, 8 bits of data, no parity bit)?
4. Have you configured your computer so it doesn't require Data Terminal Ready, Clear to Send, Carrier Detect, or Request to Send signals? If these signals <u>are</u> required, is there a peripheral in series with the TeachMover that can provide these signals, or, alternatively, have you wired jumpers W1, W2, W3, and/or W4 on the Teach-Mover's circuit card?

5. Have you opened your computer's serial port properly?
6. Have you used the proper syntaxes for the commands that correspond to INPORT and OUTPORT on your computer?

If you feel you have done everything correctly and the arm still won't respond, then call your computer dealer or computer manufacturer's customer service representative. Because of the complexities of configuring a computer's serial port--and because the procedure for doing so varies greatly from computer to computer--chances are the difficulty is with your computer and not with the TeachMover.

Once you have the @CLOSE command working, you might try using the @STEP command to move the various joints of the arm. Details of @STEP and all the other commands are given in the next section.

B. SERIAL INTERFACE COMMANDS

Ten different commands can be issued to the TeachMover over the serial lines. (A concise summary of all ten commands is given in Appendix F.)

Note:

- It is a good idea to familiarize yourself with all the teach control commands (Chapter 6) before reading about the serial interface commands in this chapter.
- All commands can be abbreviated to an "@" sign plus the first three characters--@CLO for @CLOSE, etc.
- All characters and numeric values are decimal ASCII (industry-standard character format).
- Once a serial command is executed, the teach control is left in TRAIN mode, with two exceptions:
  - @RESET leaves it in MODE mode.
  - @RUN simply runs the arm until another command stops it.
- However, the indicator lights will remain as they were before the serial command was executed. (Example: If MODE light is on, and then, say, an @CLOSE command is executed, the Teach Control will then be in TRAIN mode but with the MODE light still on.)
- If you wish to change the status of the indicator lights, you can use the @STEP command with all parameters set to zero except the "OUT" value (see below). No other serial command affects the status of the lights (except the closed light which always indicates the state of the gripper switch).

The ten commands are as follows. Details of syntax and usage begin on the next page. Sample programs using these commands are given in part C of this chapter.

1. @STEP
2. @CLOSE
3. @SET
4. @RESET
5. @READ
6. @ARM
7. @DELAY
8. @QDUMP
9. @QWRITE
10. @RUN

1. @STEP

The @STEP command is analogous to using the teach control SPEED command, the TRAIN command, and the OUT command all at once.

The @STEP command causes all six of the stepper motors to move simultaneously. The syntax of this command is:

```
@STEP <SP>,<J1>,<J2>,<J3>,<J4>,<J5>,<J6>,<OUT><CR>
```

where:

<SP> gives the speed of motion,
<J1> to <J6> are the number of half-steps that each of the six motors is to be moved,
<OUT> specifies the bit pattern to go to the user outputs,
and <CR> signifies carriage return.

Arm returns [2 <CR>] if STOP key was pressed

As explained above, the arm responds with [0<CR>] if you have made a syntax error, and [1<CR>] if the arm executes the command. However, if the STOP key gets pressed before the specified motion is completed, the arm returns [2<CR>] instead. (This is discussed further at the end of the "@CLOSE" command, below).

Now here are the details:

a. Speed Value, SP

The speed value, SP, is related to the motor speed in half-steps per second by the formula:

$$\text{Motor Speed} = \frac{1843200}{|SP - 255| \cdot 256}$$

Speed value SP

Table 7-3 gives the correspondence between motor speeds, teach control speed numbers, and serial interface speed values (SP). Note that although only 16 speeds are possible using the hand-held teach control, you can specify 246 different speeds (SP = 0, 1, 2, ..., 245) in

serial interface mode. Table 7-3 is reproduced in Appendix F for ready reference.

It's a good idea before using the @STEP command to review the maximum speeds that you can drive the motors without causing them to slip. You will find these speeds in a table in Appendix F; the maximum speeds are given in three different ways: half-steps per second, teach control speed numbers, and serial interface speeds values (SP).

TABLE 7-3

Stepping Rates

Teach Control Speed Number	Serial Port Speed Value	Half-Steps Per Second
0	0	28
1	111	50
2	159	74
3	183	99
4	205	141
5	221	206
6	232	300
7	236	360
8	238	400
9	239	424
10	240	450
11	241	480
12	242	514
13	243	554
14	244	600
15	245	655

b. Motor Steps, J1-J6

The magnitude of each of the quantities J1-J6 indicates the number of half-steps the motor should be driven. The sign of each number indicates the direction; positive directions are indicated in parentheses as follows:

Sign conventions for motor half-steps J1-J6

J1 - Base Swivel (counter-clockwise)
J2 - Shoulder Bend (downwards)
J3 - Elbow Bend (downwards)
J4 - Right Wrist (downwards)
J5 - Left Wrist (downwards)
J6 - Hand (open)

Important Point : Unlike operation with the hand-held teach control, using the @STEP command does not uncouple the elbow, <J3>, from the hand, <J6>. Moreover, you cannot specify "pitch" and "roll" directly, but only the number of half-steps of the right and left wrist, <J4> and <J5>. However, if the desired number of half-steps for the base, shoulder, elbow, pitch, roll, and grip are given by B, S, E, P, R, and G respectively, then the motion command you would use is simply:

@STEP <SP>,B,S,E,(P-R),(P+R),(E+G),<OUT><CR>

Formula for specifying pitch, roll and elbow-gripper compensation

Coordinated motion of two or more motors

If you specify an unequal number of half-steps for each joint, then TeachMover firmware will automatically coordinate the timing so as to produce smooth simultaneous motion of the motors. For example, if the elbow motor is told to move 21 steps and the shoulder motor 3 steps, the resultant timing is as illustrated in Figure 7-8.

---

*NOTE: Roll angle is defined as positive clockwise, looking towards the hand along its centerline axis from the finger tips.

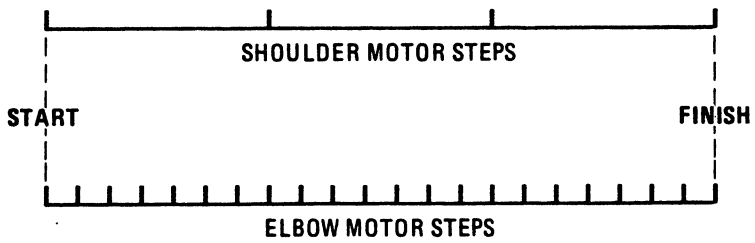


Figure 7-8 Step Timing Diagram

c. OUT Number

This is a decimal number that gets translated into a binary number that specifies which of the 9 user output bits are to be set to 1 when the specified joint motions have been accomplished. For example, the number 129 would specify that user outputs 0 and 7 (which turn on the MODE and RUN lights) should be set to 1, and all the rest to 0.

Example of OUT number

User Output Nos.:	8	7	6	5	4	3	2	1	0
Decimal:129=Binary :	0	1	0	0	0	0	0	0	1

In using the @STEP command, you won't always be moving all 6 joints at once. Often, many of the entries <J1>-<J6> and/or <OUT> will be zero. In such cases, you can omit the zeros, provided this won't alter the meaning of the command. For example, to rotate the base motor 50 half-steps at a speed value of 200 without moving any of the other members or setting any output bits*, you would not need to key in:

```
@STEP 200,50,0,0,0,0,0,0
```

but simply:

Use of zeros and blanks

```
@STEP 200,50
```

However, to hold the base fixed while moving all the other motors 50 half-steps, you would have to key in:

```
@STEP 200,0,50,50,50,50,50
```

or just:

```
@STEP 200, ,50,50,50,50,50
```

but you could not use:

```
@STEP 200,50,50,50,50,50
```

because this would be interpreted as a command to hold the hand fixed (J6=0) while moving all the other joint motors 50 half-steps.

*NOTE: Omitting the <OUT> value leaves the output bits unchanged. It does not set them all to zeros.

2. @CLOSE

This command is analogous to the GRIP command on the hand-held teach control, except that instead of closing the hand 32 steps past where the grip switch closes, it closes the hand just to where the grip switch closes.

Use of @STEP to tighten the grip

If, in practice, you find that the @CLOSE command does not cause the arm to grip an object tightly enough, you can insert an @STEP command right after the @CLOSE command. The @STEP command would specify that all the joints be held fixed while just the gripper is closed. Appendix F includes a figure showing the number of half-steps to specify in order to achieve a given gripping force.

The syntax of the @CLOSE command is simply:

```
@CLOSE <SP><CR>
```

where the optional <SP> gives the speed of closing as with the @STEP command. (NOTE: If you don't specify a speed, then the arm will use a speed equivalent to teach control speed number 5; this corresponds to 206 half-steps per second, or a serial interface speed number SP of 221.)

Default speed value

As with all serial interface commands, the arm responds with [0<CR>] if you've made a syntax error, or [1<CR>] when the command has been executed (in this case, as soon as the hand closes). As with the @STEP command, there is a third possibility: [2<CR>] will be returned if the STOP key on the teach control was pressed before the @CLOSE command finished executing. This is important to know because the STOP key will stop the current command only; any subsequent commands sent will still be executed. This means that if the STOP key is pressed during execution of an @STEP or @CLOSE command, the internal position registers will be at different settings from what the remainder of the program is expecting. To avoid this

How and why to test for [2 <CR>]

problem, you may wish to write a routine to test for a [2<CR>] and take appropriate action if [2<CR>] is, in fact, received. To do this, you will need to use the @READ command, so we'll defer further discussion on this point until we explain the @READ command in item 5 below.

3. @SET

This command activates the keys on the hand-held teach control, putting the unit into TRAIN mode. This means you can program arm positions directly from the Teach Control as well as via the @STEP command. This is an extremely useful feature, since some arm positions might be a lot easier to achieve by pressing teach control keys than by specifying joint motor steps.

The syntax of the @SET command is:

```
@SET <SP><CR>
```

Where <SP> is an optional speed value as in @CLOSE command.

Once the @SET command is given, the arm will remain in teach control TRAIN mode until you press either the MODE or REC key. At this point, a [1<CR>] will be returned to the computer or terminal. (NOTE: If the REC key is pressed to end an @SET command, the TeachMover does not record a step).

4. @RESET

This is similar to the teach control ZERO command, in that it sets the contents of the six internal position registers to zero. The @RESET command also turns off the current in all six drive motors, allowing you to manipulate the arm manually. The @RESET command is used for initializing the arm. The syntax is simply:

```
@RESET<CR>
```

The arm responds with a [0<CR>] or [1<CR>].

5. @READ

This command has no equivalent on the hand-held teach control.

The syntax is:

```
@READ<CR>
```

The arm responds with [0<CR>] or [1<CR>] followed by a string of numbers:

Values of the six internal position registers

<K1>,<K2>,<K3>,<K4>,<K5>,<K6>,<I><CR>

where <K1> to <K6> are the actual values of the internal position registers. In other words, the @READ command lets you read the position of the arm in terms of the positions of the stepper motors. As we'll see later, one use of the @READ command is to provide quantitative measurements of objects held in the gripper.

The seventh number, <I>, is a decimal integer that can be decoded to yield two things:

- the values of the 8 input bits
- the last key that was pressed on the teach control.

The formula is:

I = Last Key * 256 + Input Byte

Where "Last Key" is 0 unless a key has been pressed on the teach control since the last @READ command; otherwise "Last Key" is a numeral from 1 to 14. If "Last Key" is a non-zero numeral, its meaning is given in Figure 7-9. For example, if "Last Key" is 7, then the last key pressed was the REC key; if "Last Key" is 4, the last key pressed was the OUT key, and so forth.

"Last Key"

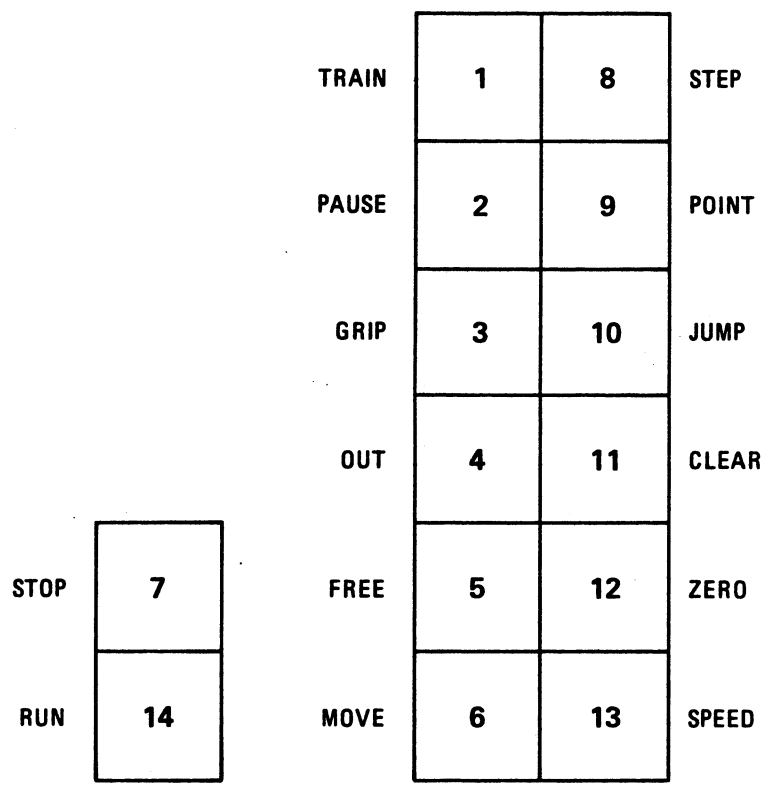


Figure 7-9 "Last Key" Numbers for the @READ Command

"Input Byte"

"Input Byte" is a decimal number that, when translated to binary, indicates which of the 8 input bits are set to 1. (Recall that bit 0 refers to the grip switch, and bits 1-7 are user inputs that can be set by external switches or by the host computer itself.)

Here's an example: If I is 264, it means that:

a. "Last Key" must be 1; in other words (see Figure 7-9), the last key pressed was the TRAIN key,

Example of determining "Last Key" and "Input Byte"

and b. "Input Byte" must be 8 (because 264 = 1 x 256 + 8). Decimal 8 equals binary 1000, which means user input Bit 3 is on, and the others (0, 1, 2, and 4-7) are off.

As mentioned earlier, the arm will respond with [2<CR>] if the STOP key was pressed during execution of an @STEP or @CLOSE command. If this does occur, it means that the internal position registers have not reached the values expected by the remainder of the program. If, in a given application, there is a chance that the STOP key will get pressed during an @STEP or @CLOSE command, then your program should include an @READ after the @STEP or @CLOSE, and then another @STEP or @CLOSE to make up the difference between the motion that should have occurred and the motion that actually occurred. (Of course, these extra program steps would only be used if the arm actually returned a [2<CR>] and you wanted to continue the program.)

What to do if arm returns [2 <CR>]

6. @ARM

If you have your computer connected over serial lines to both a video screen and your TeachMover at the same time, you might want to list a program on the screen--say, for editing--without actually causing the arm to run. To do so, you need to change the character to which the arm responds--that is, you need to instruct the arm to respond to a character other than the "@" sign, at least while you are listing and/or editing your program. If you key in:

```
@ARM <CHAR><CR>
```

where <CHAR> is any character other than a carriage return, the arm will treat as commands all signals that begin with that character, passing all other signals through unmodified (including those signals beginning with "@"). The arm responds with a [0<CR>] or [1<CR>].

Use of @ARM with multiple TeachMovers

The @ARM command can also be used to control multiple TeachMovers from a single serial port; simply connect the arms in series, and instruct each arm to respond to a different recognition character.

7. @DELAY

The above six commands handle all arm motions for most computers. However, some computers--certain personal computers in particular--cannot read characters as fast as the TeachMover transmits them. You'll know you have this problem if an @READ command generates fewer than seven numerals as a response. One cure is to use a low baud rate such as 300, but this will unnecessarily slow down transmission of commands to the arm as well. A better cure is to stay with a higher baud rate but use the @DELAY command. Keying in the following at the beginning of your program

When @DELAY is needed

```
@DELAY <N><CR>
```

where <N> is an integer, will set up a delay of approximately 1/2 millisecond times N between transmitted characters. The arm responds with a [0<CR>] or [1<CR>].

If you need to set up such a delay value for your computer, first use the teach control to key in a speed value of 0. Then use the following procedure. (The reason for setting SPEED to 0 is that the delay value set up by the @DELAY command is not actually a constant, but increases with arm speed. Since the delay is minimum at lowest arm speed, setting the delay value when SPEED is 0 will ensure that if the delay value is sufficient at that arm speed, it will be sufficient at all other arm speeds).

How to determine actual delay value by trial and error

- Use @DELAY command to set up a delay value. Use N=0.
- Do an @READ command to cause the computer to read the six position registers and the integer I (these are explained under the @READ command, above). It doesn't matter what the numerical values are that are being read; even zeros are acceptable for the purpose of this procedure.

- PRINT (that is, display on your screen) the values of the seven numerals.
- Make steps b. & c. into a loop, so the reading and displaying goes on repeatedly (sometimes everything will be all right for one or two cycles before a problem shows up).
- If the program stops, unable to display all seven numerals, it means that some of them got lost because the delay value was too short. If this is the case, return to step a., this time using a delay value of 1.
- Continue this procedure, each time adding 1 to the delay value, until all seven digits are being displayed continously. This tells you the proper delay value to use with your computer. Be sure to use this delay value in an @DELAY command at the beginning of every program you write.

8. @QDUMP

This command uploads a program from the TeachMover to a host computer, allowing you to generate programs on the hand-held teach control and then save them an a disk for later use. The syntax is simply;

```
@QDUMP<CR>
```

After handshaking the arm sends a long character string which represents the entire set of 53 program steps (or 126, if you're using extra RAM). These steps are not in quite the same format as teach control steps, nor are they the same as the serial interface commands. They are, in effect, building blocks which can be used by the computer to produce the same results as would be produced by a BASIC host computer program. Each step is coded as eight integer values, L1, L2,...L8, separated by commas; each of the eight values is two bytes in length. The first value, L1, is the step number (0-52, or 0-125). The second value, L2, contains an "opcode" that tells what kind of step is being uploaded. The opcode and the corresponding structure of the eight integer values for each kind of step is given in Table 7-4. Each step is terminated with a <CR>.

@QDUMP returns character string consistency of eight integer values for each program step

[Note: The information in Table 7-4 may seem complex, and it is. The complexity is partly due to the Teach-Mover's limited memory space; there just isn't room to store the routines that would be necessary to convert data from a more straightforward format to the packed format we had to use for internal storage. You'll be pleased to know that to use @QDUMP to upload programs, you do not need to know any of the details in Table 7-4. This is essentially true for downloading programs once they've been uploaded. The only times you'll need to know the details are when you wish to download a program that was not previously uploaded - that is, when you wish to down-load a program initially generated on the host computer -

or when you wish to manipulate the data itself rather than simply store and retrieve it. The notes below Table 7-4 provide information that will make these tasks easier.]

TABLE 7-4

OPCODES AND CHARACTER STRINGS FOR @DUMP COMMAND*

(SP = serial interface speed value)

(In all cases, L1 = program step number)

Type of step	Opcode	Description of the remaining seven 2-byte fields
MOVE	1	L2: opcode + [(255 - SP) * 256] L3: J1(low) + (J2(low) * 256) L4: J3(low) + (J4(low) * 256) L5: J5(low) + (J6(low) * 256) L6: J1(high)+(J2(high) * 256) L7: J3(high)+(J4(high) * 256) L8: J5(high)+(J6(high) * 256) where J1(low) and J1(high) are the low and high bytes, respectively, of the number of half-steps of motion specified for motor 1, and similarly for J2-J6; see Note 1 below. (The MOVE step is an encoded form of an @STEP command with the OUT value ommitted.)
GRIP	2	L2: opcode + [(255-SP) * 256]. L3-L8: not used (This has the same effect as the GRIP command on the hand-held teach coontrol; that is, it closes the gripper 32 steps past the point where the grip switch is activated, thereby building up 1 lb. of gripping force.)
JUMP	3	L2: opcode + [(jump condition) * 256]. See Appendix F, item 6 for jump conditions. L3: step no. to jump to L4-L8: not used (This is a direct encoding of the teach control JUMP command.)

*(See Note 1 for negative numbers and numbers greater than 32767)

Table 7-4 (cont.)

PAUSE	4	L2: opcode + [(pause time in sec.) * 256]. L3-L8: not used (encodes the teach control PAUSE command)
OUTPUT	5	L2: opcode + [(output bit no.) * 256. See Appendix F, item 6 for output numbers. L3: output value (0 or 1) L4-L8: not used (encodes the teach control OUT command)
SIGNAL	6	L2: opcode + [(1st character of a string) * 256] L3: 2nd character of string + [(3rd character) * 256] L4: 4th character of string + [(5th character) * 256] L5: 6th character of string + [(7th character) * 256] L6: 8th character of string + [(9th character) * 256] L7: 10th character of string + [(11th character) * 256] L8: 12th character of string + [(13th character) * 256] (Sends a character string to the host computer. The string sent can be from 0 to 12 characters long, and must be followed by a zero. See Notes 2 and 3 below.)
PATH	7	The same as MOVE, but without controlled acceleration. (PATH was used in generating steps 151-161 in the Exerciser Program listed at the end of chapter 6.) Also see Note 2 below.

Table 7-4 (cont.)

Note 1. Some low cost computers cannot deal with numbers greater than 32767 since they only use two bytes to represent numbers. Because of this, the TeachMover converts any larger number to a negative number. Any negative number sent by the TeachMover in response to an @QDUMP command can be converted to a positive number by adding 65536 to it. This is not necessary when transferring programs that do not need to be modified or decoded, or programs generated by the host computer.

The high and low bytes of, say, L3 can be calculated as follows:

```
If L3<0 then L3=L3+65536
L3(high) = integer quotient of L3/256 (that is,
           the quotient without the remainder)
L3(low) = remainder of (L3/256)
```

Example: If L3 = 259, then

```
J2(low)=L3(high) = 1
J1(low)=L3(low)  = 3
```

In practice, calculation of the high and low bytes of a number can be programmed in a variety of ways. On some computers there is a remainder function (REM) that you can use to calculate the remainder of J1/256 directly. Or, if your computer has logical functions (AND, OR, NOT, etc.), the remainder of J1/256 will be given by "ANDing" J1 with 255. (This works because 255 is equivalent to a low byte full of ones.) If none of these special functions is available, you can calculate the value of, say J1, using the following formulas (in integer arithmetic):

```
If L3<0 then L3=L3+65536

If L6<0 then L6=L6+65536
J1=L3-[(L3/256)*256]+[(L6-[(L6/256)*256])*256]
If J1>32767 then J1=J1-65536
```

Note 2. Neither SIGNAL steps nor PATH steps can be generated on the hand-held teach control. Thus, these steps can only be uploaded if they have previously been downloaded from the host.

Table 7-4 (cont.)

Note 3 . You can download a program in segments, using SIGNAL to tell the host when each segment is completed. Then while a given program segment is executing on the TeachMover, the host can be performing other tasks - including interacting with other TeachMovers. This cuts run time, because the host computer does not have to sit idle while a TeachMover is executing a given program segment.

The character string you send as a signal to the host can be [1<CR>], or [DONE<CR>], or almost anything you wish. (For most computers - at least those that employ BASIC - use of <CR> to signal the end of an input or an output sequence is mandatory.) Let's say you wish to use [R<CR>], to indicate "ready." The SIGNAL step would have to contain:

```
program step no.: (let's say it's step no. 21)
opcode: 6
1st character:  ASCII R (decimal 82)
2nd character:  ASCII <CR> (decimal 13)
3rd character:  0
```

The actual SIGNAL step would be coded as follows:

```
21, (7 + (82 * 256)), (13 + (0 * 256))
```

and sent as:

```
@QWRITE 21, 20999,13<CR>
See Note 5 after @QWRITE.
```

The next program step, number 22, would be coded as an unconditional jump to itself. This would cause the arm to wait until the host tested for the [R<CR>] and then downloaded another program segment.

Note 4 . To use SIGNAL to operate multiple TeachMovers simultaneously from a single serial port, program each TeachMover to send back a unique character string; this tells the host which arm is ready for its next command. (Without using SIGNAL in this way, multiple TeachMovers could only operate sequentially because if they were to operate simultaneously, the computer would not know which arm had sent back the standard "handshake" of [1<CR>].)

9. @QWRITE

This command is the reverse of the @QDUMP command, in that it allows you to download programs from your computer to the TeachMover. Unlike the @QDUMP command, you must use @QWRITE once for each program step, rather than just once for an entire program. The syntax is:

```
@WRITE <L1>,<L2>,<L3>,<L4>,<L5>,<L6>,<L7>,<L8><CR>
```

One @QWRITE for each program step

where <L1> is the step number in TeachMover memory to which you wish to write the program step, and L2-L8 are the values of seven two-byte data fields, structured exactly as in the @QDUMP command.

Down loading a previously uploaded program

To download a program that has previously been uploaded, all you need to do is use the @WRITE command once for each program step, retrieving L1, L2,..., L8 from memory/storage. (To store an uploaded program, it's best to use a 53x8 array [or 126x8 if you're using exta RAM]. The values of the array elements can be read in via one or more "INPORT" statements immediately following the @QDUMP command. Then, on downloading the program, the lines of the array can form the contents of successive @WRITE commands.)

Downloading a program not previously uploaded

Downloading a program that has not previously been uploaded is a more complex process, because you must encode each step individually. As an example, let's say you wanted to download the equivalent of:

@STEP 200,150,621,-323<CR> to TeachMover step number 42.

The command would be:

```
@QWRITE 42,(1+((255-200)*256),
(150(low)+(621(low)*256)),(-323(low)+(0*256)),(0+(0*256)),
(150(high)+(621(high)*256)),(-323(high)+(0*256)),
  (0+(0*256))<CR>
```

and sent as:

```
@QWRITE 42,14081,28054,189,0,27904,255<CR>
```

See Note 5 and 6.

Note 5. As in the @STEP command, trailing zero values may be omitted.

Note 6. The conversion to high and low bytes is the same as the conversion from high and low bytes in Note 1 except that it is not necessary to do the final conversion to negative if the result is greater than 32767. This is because the TeachMover accepts 32768 and -32768 as the number.

10. **@RUN**

This command will run any program stored in the TeachMover, whether the program was keyed in on the hand-held teach control, downloaded from a host computer, or one of the permanent demonstration programs.

The syntax is:

```
@RUN <N><CR>
```

where N is the program step at which you wish the program to begin running. (Example: @RUN 126 <CR> will run the demonstration program stored in TeachMover firmware beginning at program step 126. When you issue an @RUN command without any syntax errors, the arm responds with [1<CR>] before beginning to execute the TeachMover program.

[Note: Any valid serial command will stop the arm once it's running, but will not change the lights unless it is an @STEP command with an <OUT> value.]

C. SAMPLE PROGRAMS

Note : In the programs in this section, we will use the imaginary instructions OUTPORT and INPORT to represent the particular instructions you normally use with your computer for receiving and sending information over serial lines (that is, between the computer and a peripheral unit). These instructions are not to be confused with the more familiar instructions INPUT and PRINT. INPUT refers to information entered at the keyboard, while PRINT is a command to display information on the screen. See part A, section 6 of this chapter for further details on INPORT and OUTPORT.

Before running a program, remember to initialize the arm position according to the instructions given in Appendix G.

1. Use of @SET and @READ

In the program shown in Figure 7-10, the internal position registers are first initialized by means of an @RESET command, then control of the arm is turned over to the hand-held teach control by means of an @SET command (line number 80). The user moves the arm to a new position by means of the joint-control keys, then presses the REC or MODE key. The program reads the new joint positions, and uses this information to move the joints (except the gripper) back to their original configuration. Finally, the program causes the gripper to close. Notice how the @READ command (line 110), followed by the INPORT command on line 115 assigns variable names (A-F) to the six joint positions; this is so that the @STEP command (line 190) can move the arm back to exactly where it started.

[Note : In this program and those that follow, we are using the single statement INPORT I - rather that the subroutine discussed in part A, section 6 off this chapter - to accomplish the necessary "handshaking" with the TeachMover. As you know, the value of I will be 0, 1, or 2, and you should test to find out which it is; a 0 will mean a syntax error had been made and that the previous command could not be executed, while a 2 will mean that the STOP key was pressed while an @STEP or @CLOSE was executing.

Use of INPORT 1 in place of "handshaking" subroutine

The "handshaking" subroutine can be used to perform this test. We are omitting the subroutine from our program listings only to avoid clutter. When you're entering one of our programs into your computer and you come to INPORT I following an arm command, you can replace INPORT I with a call to the subroutine. (You can, if you wish, omit the subroutine and assume that the command syntax is correct; but you would still have to retain the

INPORT I statement after each arm command. This is because even if there is no syntax error, your computer needs to wait until an arm command has finished executing before going on to the next command, and the only way it can know this is by inputting the [1<CR>].)

Also, in some of our sample programs, we use IF... THEN...ELSE statements, statements involving logical expressions (AND, OR, etc.), and other kinds of statements that may not be available on your computer. If you run into a statement that your computer cannot interpret, you will need to modify the program using one or more statements that your computer can interpret and that will still yield the desired results.]

```
1   REM ****************************
2   REM *                          *
3   REM * SERIAL PORT DEMO PROGRAM *
4   REM *                          *
5   REM ****************************

10   OUTPORT "@RESET": INPORT I
20   REM
30   REM
40   PRINT "YOU CAN CONTROL THE ARM USING THE TEACH CONTROL"
50   PRINT "USE ONE OR MORE FOR THE JOINT-CONTROL KEYS"
70   PRINT "PRESS THE REC OR MODE KEY"
80   OUTPORT "@SET 200"
85   INPORT I
90   REM
100   REM
110   OUTPORT "@READ": INPORT I
115   INPORT A,B,C,D,E,F,G
120   REM THIS READS THE AMOUNT YOU MOVED
130   REM
140   REM
150   OUTPORT "@CLOSE",200
155   INPORT I
160   REM THIS CLOSES THE GRIPPER
170   REM
180   REM
190   OUTPORT "@STEP",200, - A, - B, - C, - D, - E
195   INPORT I
200   REM THIS MOVES IT BACK TO WHERE IT WAS
210   REM
220   REM
230   END
```

Figure 7-10 Serial Port Demonstration Program

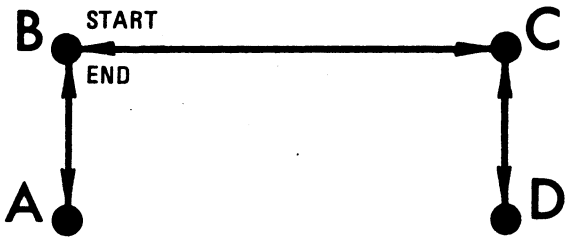
2. "Pick-and-place" program

The "pick-and-place" task is common in industrial robotics. A typical application is where parts from a feeder are placed in an assembly or on a moving conveyor. We will assume an unending supply of parts at the pick-up point, and continual removal of parts at the placement point.

The task is defined in Figure 7-11. Pickup is from A, and placement is at D. The approach and departure trajectories AB & CD are generally required for practical reasons: an object on a flat surface is best lifted before it is moved, and when the object is put down and released, the arm should be raised so as not to obstruct removal of the object.

The sequence of statements shown in Figure 7-12 will accomplish this task in a straightforward manner. However, for the program to actually work, you must specify the speed (S) and also define the various motor-step values (P1, P2, etc.) numerically, earlier in the program. In other words, you must know and specify in advance the number of motor steps that are required on each joint in order to accomplish each desired motion. To try to determine these step values by trial and error is impractical. Here is where the @SET and @READ commands can be of great benefit, for they let you position the arm via the hand-held teach control, and then cause the computer to count and remember the number of steps each motion takes.

Figure 7-13 shows one way to program this. The quantities A1, A2,..., A5 represent the actual values of the internal position registers when the arm is at point A of figure 7-11; B1, B2,..., B5 represent the values at point B; and so forth. The actual amounts of joint movement are given by subtraction. For example, D2-C2 is



1. Move Down to A
2. Close Grip
3. Move to B
4. Move to C
5. Move to D
6. Open Grip
7. Move to C
8. Move to B

Figure 7-11 Pick-and-Place Task

the number of steps the shoulder motor must have turned in moving the arm from point C down to Point D. This means that to return from D to C, the shoulder motor has to turn the number of steps given by the expression C2-D2. Note how this calculation is incorporated into lines 390 and 400 of the program. Also note how the required gripper movement is determined empirically (lines 165-205).

Line 290 represents the last line of the portion of the overall program that causes the computer to learn the joint positions and proper joint motions. Once this portion is accomplished, then the pick-and-place program itself can be run repeatedly and without human intervention. This portion of the overall program begins on line 1000. Note that although the hand is to remain closed while moving the part, it is necessary to program a non-zero value for the number of half-steps of the hand motor during this portion of the program in order to decouple elbow and grip movements, that is, in order to keep the gripper from opening when the elbow joint rotates. (See lines 1020, 1030, and 1040, in which the value J6 has been set equal to the elbow value J3, as per the equation given in the section on the @STEP command, above. This compensation was omitted from the listing in Figure 7-12 for simplicity.)

```
10    OUTPORT "@STEP",S,-P1,-P2,-P3,-P4,-P5
15    INPORT I
20    OUTPORT "@CLOSE"
25    INPORT I
30    OUTPORT "@STEP",S, P1, P2, P3, P4, P5
35    INPORT I
40    OUTPORT "@STEP",S, Q1, Q2, Q3, Q4, Q5
45    INPORT I
50    OUTPORT "@STEP",S, R1, R2, R3, R4, R5
55    INPORT I
60    OUTPORT "@STEP",S, 0, 0, 0, 0, 0, GR
65    INPORT I
70    OUTPORT "@STEP",S,-R1,-R2,-R3,-R4,-R5
75    INPORT I
80    OUTPORT "@STEP",S,-Q1,-Q2,-Q3,-Q4,-Q5
85    INPORT I
90    GOTO 10
```

Figure 7-12 Pick-and-Place Program Segment

```
 1  REM ************************
 2  REM *                      *
 3  REM * PICK & PLACE PROGRAM *
 4  REM *                      *
 5  REM ************************
 100  OUTPORT "@RESET": INPORT I: REM ZERO COUNTERS
 110  PRINT "PICK - AND - PLACE ROUTINE"
 120  PRINT "USE KEYBOARD TO SET SPEED"
 130  INPUT "SPEED=";S
 140  PRINT "POSITION HAND ON PART"
 150  PRINT "      ADJUST HAND OPENING TO CLEAR PART"
 155  PRINT "PRESS MODE WHEN DONE"
 160  OUTPORT "@SET",200: INPORT I
 165  OUTPORT "@READ": INPORT I
 170  INPORT A1,A2,A3,A4,A5,GO,V
 180  OUTPORT "@CLOSE",S: INPORT I: REM CLOSE HAND AND MEASURE PART
 185  OUTPORT "@READ": INPORT I
 190  INPORT A1,A2,A3,A4,A5,GC,V
 200  G = GO - GC: REM GRIP SIZE MINUS GRIP SIZE CLOSED
 205  OUTPORT "@STEP",S,0,0,0,0,0, - 50: INPORT I
 210  PRINT "POSITION PART ABOVE PICKUP SITE"
 215  PRINT "PRESS MODE WHEN DONE
 220  OUTPORT "@SET",200: INPORT I
 225  OUTPORT "@READ": INPORT I
 230  INPORT B1,B2,B3,B4,B5,GV,V
 240  PRINT "POSITION PART ABOVE PLACEMENT SITE"
 245  PRINT "PRESS MODE WHEN DONE"
 250  OUTPORT "@SET",200: INPORT I
 255  OUTPORT "@READ": INPORT I
 260  INPORT C1,C2,C3,C4,C5,GV,V
 270  PRINT "POSITION PART AT PLACEMENT SITE"
 275  PRINT "PRESS MODE WHEN DONE"
 280  OUTPORT "@SET",200: INPORT I
 285  OUTPORT "@READ": INPORT I
 290  INPORT D1,D2,D3,D4,D5,GV,V
 300  REM
 310  INPUT "TYPE G TO GO?";R$
 320  IF R$ = "G" THEN 330
 325  END
 330  REM
 340  REM  RUN THE PICK - AND - PLACE PROGRAM
 350  REM
 360  REM  RELEASE PART AND RETURN TO B
 370  REM
 380  OUTPORT "@STEP",S,0,0,0,0,0,G: INPORT I
 390  OUTPORT "@STEP",S,C1 - D1,C2 - D2,C3 - D3,C4 - D4,C5 - D5: INPORT I
 400  OUTPORT "@STEP",S,B1 - C1,B2 - C2,B3 - C3,B4 - C4,B5 - C5: INPORT I
 500  PRINT "TYPE G WHEN PART IS IN POSITION:";
 510  INPUT R$: IF R$ <  > "G" THEN  END
1000  OUTPORT "@STEP",S,A1 - B1,A2 - B2,A3 - B3,A4 - B4,A5 - B5: INPORT I
1010  OUTPORT "@CLOSE",S: INPORT I
1015  OUTPORT "@STEP",S,0,0,0,0,0, - 50: INPORT I
1020  OUTPORT "@STEP",S,B1 - A1,B2 - A2,B3 - A3,B4 - A4,B5 - A5,B3 - A3:
       INPORT I
1030  OUTPORT "@STEP",S,C1 - B1,C2 - B2,C3 - B3,C4 - B4,C5 - B5,C3 - B3:
       INPORT I
1040  OUTPORT "@STEP",S,D1 - C1,D2 - C2,D3 - C3,D4 - C4,D5 - C5,D3 - C3:
       INPORT I
1050  OUTPORT "@STEP",S,0,0,0,0,0,G: INPORT I
1060  OUTPORT "@STEP",S,C1 - D1,C2 - D2,C3 - D3,C4 - D4,C5 - D5: INPORT I
1070  OUTPORT "@STEP",S,B1 - C1,B2 - C2,B3 - C3,B4 - C4,B5 - C5: INPORT I
1080  GOTO 500
1090  END
```

Figure 7-13 Full Pick-and-Place Program

3. Thickness measuring program

Figure 7-14 shows a program that can be used to actually measure the thickness of various objects placed between the TeachMover's fingers.

When the program is first run, the @CLOSE command is used to close the hand with no object in its grasp (line 160). The internal position registers are set to zero (line 170) and the hand is then opened to a wide setting (line 190). This begins the actual measurement loop. Whenever you press the carriage return key, the hand will close on an object (line 220), the position registers will be read (line 230), and the value of the position register for the hand will be converted to inches (line 250).

This program also includes a determination of whether the object grasped was "thick" or "thin;" an arbitrary value of 0.015 inches is used as a cut-off. This kind of decision-making can be used to determine of an object was indeed gripped or if the fingers merely closed upon themselves.

```
100  PRINT "*****************************************";
110  PRINT "*   PROGRAM TO MEASURE THICKNESS OF     *";
120  PRINT "*  OBJECTS PLACED BETWEEN THE FINGERS   *";
130  PRINT "*              OF THE HAND.             *";
140  PRINT "*****************************************"
150  OUTPORT "@STEP 240",1, - 100,1,1,1,500: INPORT I
160  OUTPORT "@CLOSE 240": INPORT I: REM CALIBRATE HAND
170  OUTPORT "@RESET": INPORT I
180  REM RESTORE HAND TO PREVIOUS EXTENSION
190  OUTPORT "@STEP 240",0,0,0,0,0,1113 - JAW: INPORT I
200  PRINT "PRESS <RET> TO MEASURE OBJECT";
210  INPUT D$
220  OUTPORT "@CLOSE 240": INPORT I
230  OUTPORT "@READ": INPORT I
240  INPORT V,V,V,V,V,JAW,V
250  TH = JAW / 371
260  IF TH < 0.015 THEN  PRINT "THICKNESS IS 0.": GOTO 280
270  PRINT "THICKNESS = ";TH;" INCHES."
280  PRINT "----------------------------------------"
290  GOTO 180
```

Figure 7-14 Thickness Measuring Program

4. Cartesian "Pick-and-Place" Program

In some pick-and-place applications, programming arm positions on the hand-held teach control is not practical. For example, to move pieces on a chess or checker board would require manually teaching 64 different board locations and 64 placement points. However, since the 64 locations are in a simple geometric relationship with one another, a program that used Cartesian coordinates to specify positions would represent a more practical approach.

However, the TeachMover commands (@STEP, @READ, etc.) are in terms of joint coordinates (J1, J2,...,J6), not Cartesian coordinates in space. Thus, if you specify positions in Cartesian coordinates, you must convert those positions into joint coordinates in order to use the TeachMover's serial interface commands. Formulas for the required coordinate transformations--and derivations of those formulas--are given in Appendix D. These formulas are incorporated into a subroutine (Lines 5000 - 6010) in the sample Cartesian pick-and-place program shown in Figure 7-15.

In this program, Cartesian data is given beginning at line 10000. The Cartesian coordinates of four points are given; these correspond to points A, B, C, and D in Figure 7-11. The particular Cartesian values are as follows:

Position	X(in.)	Y(in.)	Z(in.)	P(pitch) (in.)	R(roll) (in.)
A	8	0	0.5	-90	0
B	8	0	2.0	-90	0
C	6	5	2.0	-90	90
D	6	5	0.5	-90	90

In other words, the X location of the object is changed from 8 inches to 6 inches, its Y location is changed from 0 to 5 inches, and before being moved laterally the object is lifted 1 1/2 inches (from 0.5 to 2 inches). The angular orientation of the hand is always straight down (Pitch = - 90 degrees), but the object is rotated through 90° of roll between pick-up and set-down (R changes from 0 to 90). You'll find these values in the data statements beginning on line 10010 (line 10000 gives the standard initialization configuration described in Appendix B) ; these data statements also include a sixth value governing the hand opening, and a seventh value governing arm speed.

In the first working phase of our program (lines 160-320), the arm is moved from position to position as the coordinate transformations are performed. The resultant joint steps are stored in matrix uu. After all the transformations are obtained, control is transferred to line 1000 and the pick-and-place cycle is run repeatedly from the data in the variables, without any further need for coordinate conversions.

The program in Figure 7-15 is written with data for just one pick-up point, but a straightforward extension of the programming could cause the arm to pick up and move pieces on a chess board, where all the pick-up positions (after the first one) are specified by simple formulas rather than by actual numerical data. You might wish to try this as an exercise.

```
10  REM ********************************
15  REM *                              *
20  REM *                              *
30  REM *          TEACHMOVER          *
40  REM *                              *
50  REM * CARTESIAN COORDINATE CONTROL *
60  REM *                              *
70  REM *            PROGRAM           *
80  REM *                              *
90  REM ********************************
100  REM DEFINE ROBOT ARM CONSTANTS
101 H = 7.625: REM SHOULDER HEIGHT ABOVE TABLE
102 L = 7.0: REM SHOULDER TO ELBOW AND ELBOW TO WRIST LENGTH
103 LL = 3.8: REM WRIST TO FINGERTIP LENGTH
104  REM
110  REM  DEFINE OTHER CONSTANTS
111 PI = 3.14159265
112 C = 57.2957795: REM DEGREES IN 1.00 RADIAN
113 R1 = 1: REM FLAG FOR WORLD COORDINATES
114  REM
120  REM  DEFINE ROBOT ARM SCALE FACTORS
121 S1 = 1125:S2 =-S1: REM STEPS/RADIAN, JOINTS 1 & 2
122 S3 =- 661.2: REM STEPS/RADIAN, JOINT 3
123 S4 =- 244.4:S5 = S4: REM STEPS/RADIAN, JOINTS 4 & 5
124 S6 = 371: REM STEPS/INCH, HAND
125  REM
130  REM  INITIALIZATION
131  DIM UU(7,40): REM ROOM FOR 40 STEPS
133 P1 = 0:P2 =- 508:P3 =    + 1162:P4 = +384 : P5 = P4 : P6 = 0
134 REM LINE 133 IS THE NUMBER OF JOINT STEPS FROM T1= 0,T2= 0,T3= 0,T4= 0,T5= 0,
    AND J=0, TO X=5,Y=0,Z=0,P=-90,R=0,AND J=0
136  REM
137  REM READ IN FIRST LINE FOR INITIALIZATION
138  READ X,Y,Z,P,R,GP,S
139 PRINT  "SET ARM TO THE FOLLOWING POSITION & ORIENTATION"
140 PRINT"   USING KEYBOARD, TYPE 0 WHEN FINISHED"
141 PRINT"      X = ";X;" INCHES"
142 PRINT"      Y = ";Y;" INCHES"
143 PRINT"      Z = ";Z;" INCHES"
144 PRINT" PITCH = ";P;" DEGREES"
145 PRINT"  ROLL = ";R;" DEGREES"
146 PRINT"  HAND = ";GP/S6;"INCHES"
150  OUTPORT "@SET 200": INPORT I: REM MOVE ARM
155  OUTPORT "@RESET": INPORT I
160 U = 0
170  READ X,Y,Z,P,R,GP,S
175 INPUT "HIT RETURN TO GO ON:";A$
180  IF X < 0 THEN 1000
190  GOSUB 4000: REM SHOW COORDINATES
200  GOSUB 5000
```

Figure 7-15 Cartesian Coordinate Program

```
210  OUTPORT "@STEP",S,W1 - UU(1,U),W2 - UU(2,U),W3 - UU(3,U),W4 - UU(4,U),
     W5 - UU(5,U),W3 - UU(3,U): INPORT I
220 U = U + 1
221 UU(1,U) = W1
222 UU(2,U) = W2
223 UU(3,U) = W3
224 UU(4,U) = W4
225 UU(5,U) = W5
226 UU(6,U) = GP
227 UU(7,U) = S
230  IF GP < 0 THEN 300
231  REM OPEN HAND IF JAW > 0
240  OUTPORT "@STEP",S,0,0,0,0,0,GP: INPORT I
250  GOTO 170
290  REM CLOSE HAND AND SQUEEZE IF JAW < 0
300  OUTPORT "@CLOSE 245": INPORT I
310  OUTPORT "@STEP",240,0,0,0,0,0,GP: INPORT I
320  GOTO 170
1000 PRINT"RUN THE PROGRAM"
1010  FOR I = 2 TO U
1020  OUTPORT "@STEP",UU(7,I),UU(1,I) - UU(1,I - 1),UU(2,I) - UU(2,I - 1),
      UU(3,I) - UU(3,I - 1),UU(4,I) - UU(4,I - 1),UU(5,I) - UU(5,I - 1),
      UU(3,I) - UU(3,I - 1): INPORT N
1030  IF UU(6,I) < 0 THEN 1060
1040  OUTPORT "@STEP",UU(7,I),0,0,0,0,0,UU(6,I): INPORT N
1050  GOTO 1100
1060  OUTPORT "@CLOSE 245": INPORT N
1080  OUTPORT "@STEP",240,0,0,0,0,0,UU(6,I): INPORT N
1100  NEXT I
1110 CY = CY + 1
1120 PRINT"CYCLE ";CY
1130  GOTO 1010
4000  REM
4010  REM DISPLAY COORDINATES
4020 PRINT"ROBOT ARM IS MOVING TO THE FOLLOWING"
4030 PRINT"COORDINATES:"
4050 PRINT:PRINT"      X = ";X;" INCHES"
4060 PRINT"      Y = ";Y;" INCHES"
4070 PRINT"      Z = ";Z;" INCHES"
4080 PRINT" PITCH = ";P;" DEGREES"
4090 PRINT"  ROLL = ";R;" DEGREES"
4100 PRINT"  HAND = ";GP/S6;"INCHES"
4110  RETURN
4120  REM
4130  REM
4140  REM ROUTINE TO CONVERT CARTESIAN COORDINATES
4150  REM TO NUMBER OF JOINT STEPS AWAY FROM START POSITION
4160  REM
5000  REM
5010  REM  BACKWARD SOLUTION CALCULATIONS
```

Figure 7-15 (Cont'd)

```
5020 P=P/C:R=R/C
5030  IF X = 0 THEN T1 =  SGN (Y) * PI / 2
5040  IF X <  > 0 THEN T1 =  ATN (Y / X)
5050  IF T1 < 0 THEN PRINT:PRINT"BASE OUT OF RANGE. T1= ";T1
5060 RR =  SQR (X * X + Y * Y)
5070  IF RR < 2.25 AND Z < 15 THEN PRINT:PRINT"HAND TOO CLOSE TO BODY.  RR = ";RR
5080  IF RR > 17.8 THEN PRINT: PRINT"REACH OUT OF RANGE.  RR = ";RR
5090 R0 = RR - LL *  COS (P)
5100  IF X < 2.25 AND Z < 1.25 AND R0 < 3.5 THEN  IF P <  - 90 / C
     THEN PRINT:PRINT"HAND INTERFERENCE WITH BASE."
5110  REM  NOTE THAT THE ABOVE STATEMENT MAY BE ALTERED TO ACCOMODATE
     MOVES CLOSE TO THE BASE
5120 Z0 = Z - LL *  SIN (P) - H
5130  IF R0 = 0 THEN B = ( SGN (Z0)) * PI / 2
5140  IF R0 <  > 0 THEN B =  ATN (Z0 / R0)
5150 A = R0 * R0 + Z0 * Z0
5160 A = 4 * L * L / A - 1
5170  IF A < 0 THEN PRINT:PRINT"REACH OUT OF RANGE FOR SHOULDER AND ELBOW.": GOTO 5500
5180 A =  ATN ( SQR (A))
5190 T2 = A + B
5200 T3 = B - A
5210  IF T2 > 144 / C OR T2 <  - 35 / C THEN PRINT:PRINT"SHOULDER OUT
     OF RANGE.  T2 = ";T2 * C
5220  IF T2 - T3 < 0 OR T2 - T3 > 149 / C THEN PRINT:PRINT"ELBOW OUT
     OF RANGE.  T3 = ";T3 * C
5230  IF (R > 270 / C OR R <  - 270 / C) THEN  IF (P > ((90 / C + T3) - (R + 270 / C))
     OR P < (( - 90 / C + T3) + (R - 270 / C))) THEN PRINT:PRINT"PITCH OUT
     OF RANGE.  PITCH= ";P * C
5240  IF P > (90 / C + T3) OR P < ( - 90 / C + T3) THEN PRINT:PRINT"PITCH OUT
     OF RANGE.  PITCH = ";P * C
5250  IF (R > (360 / C -  ABS (P - T3)) OR R < ( - 360 / C +  ABS (P - T3)))
     THEN PRINT:PRINT"ROLL OUT OF RANGE.  ROLL = ";R * C
5260 T4 = P - R - R1 * T1
5270 T5 = P + R + R1 * T1
6000  REM CORRECT COORDINATES
6010 W1 =  INT (S1 * T1 + .5) -P1
6020 W2 =  INT (S2 * T2 + .5) -P2
6030 W3 =  INT (S3 * T3 + .5) -P3
6040 W4 =  INT (S4 * T4 + .5) - P4
6050 W5 =  INT (S5 * T5 + .5) - P5
6060  RETURN
9999  REM       X   Y     Z     P    R     J    SP
10000  DATA     5,  0,    0,  -90,   0,    0,  200
10010  DATA     8,  0,    2,  -90,   0,  800,  221
10020  DATA     8,  0,  0.5,  -90,   0,  -50,  221
10030  DATA     8,  0,    2,  -90,   0,    0,  221
10040  DATA     6,  5,    2,  -90,  90,    0,  221
10050  DATA     6,  5,  0.5,  -90,  90,  300,  230
10060  DATA      6,  5,    2,  -90,  90,   -1,  230
10070  DATA     8,  0,    2,  -90,   0,  800,  221
10080  DATA -999,  0,    0,    0,   0,    0,    0
```

Figure 7-15 (Cont'd)

5. Cartesian Demonstration Program

This program is identical to the Cartesian pick-and-place program of Figure 7-15, except that line 1010 should be changed to:

```
1010 for I = 1 to U
```

and the data statements should be as in Figure 7-16.

This program manipulates a 2.6" x 1.6" x 1.6" block into many different positions and orientations, returns to the starting position, and then repeats the cycle. You can use this program to test the tension in all the TeachMover cables. If any of the cables are too tight, then the starting position of the block will drift over time, indicating that one or more of the stepper motors is slipping. If you do find that there is slippage, adjust the appropriate cable(s) according to the instructions in Chapter 3, part C (Cable Tension Adjustments).

```
10000  DATA 5,0,0,-90,0,0,225
10010  DATA 5,0,.5,-90,0,-1,225
10020  DATA 5,0,.5,-90,0,900,240
10030  DATA 8,0,.5,-90,0,-30,240
10040  DATA 8,0,1,-90,0,0,225
10050  DATA 8,0,1,-90,90,0,240
10060  DATA 8,-5,1,-90,90,0,240
10070  DATA 8,0,1,-90,90,0,240
10080  DATA 8,5,1,-90,90,0,240
10090  DATA 8,0,1,-90,90,0,240
10100  DATA 8,0,.5,-90,90,600,240
10110  DATA 8,0,3,-90,90,0,225
10120  DATA 8,0,3,-45,0,0,240
10130  DATA 8,0,.5,-45,0,-30,240
10140  DATA 8,0,2,-45,0,0,225
10150  DATA 8,0,10,-45,0,0,225
10160  DATA 8,0,4,-45,180,0,240
10170  DATA 8,0,.5,-45,180,300,200
10180  DATA 8,0,4,-45,180,0,225
10190  DATA 8,0,4,-135,0,0,240
10200  DATA 8,0,.5,-135,0,-30,240
10210  DATA 8,0,3,-135,0,-30,225
10220  DATA 8,0,3,-45,0,0,240
10230  DATA 8,0,.5,-45,0,300,240
10240  DATA 8,0,1.5,-90,0,0,225
10250  DATA 8,0,.5,-90,0,-30,240
10260  DATA 8,0,1,-90,0,0,225
10270  DATA 8,0,1,-90,-90,0,240
10280  DATA 8,0,.5,-90,-90,300,240
10290  DATA 8,0,3,-90,0,-1,225
10300  DATA 7,0,3,-90,0,0,240
10310  DATA 6,0,1,-90,0,0,240
10320  DATA 5,0,0,-90,0,0,200
10330  DATA -1000,0,0,0,0,0,0
```

Figure 7-16 Cartesian Demonstration Program

D. Experimentation

Careful study of the above sample programs will yield many pointers on how to write successful TeachMover programs using your computer. There is virtually no limit to the variety of tasks you can accomplish using serial interface mode. Once you're familiar with TeachMover programming, you might like to try some of the advanced applications suggested in the next chapter.

CHAPTER EIGHT
SUGGESTED ADVANCED APPLICATIONS

Some of the applications suggested in this chapter require additonal hardware - an extra cable, special sensors, or even a second TeachMover. Other applcations call for the development of new software routines. Still others are on the forefront of artificial intelligence research, and thus represent challenges that could take several man-years to overcome. Feel free to work on any of these applications you wish - or try venturing forth into other new areas on your own. In any event, we're sure you'll find one thing to be true: The more you use the TeachMover, the more ways you'll find of using it.

A. USING LOGIC FLAGS

If you wire one or more jumpers so as to connect the user input bits directly to the user output bits on the auxiliary I/O port, then you can use these bits as logic flags. One part of your program can set one of the bits to signify a certain condition. Other parts of the same program can test that bit to determine the appropriate action to take.

B. COORDINATING TWO TEACHMOVERS

In a similar manner, you can connect the output bits of one TeachMover to the input bits of another. Their actions can then be coordinated by using the I/O bits as "mailboxes." One TeachMover sets a bit when it desires that the other TeachMover perform some action. The other TeachMover can stay in a loop testing that bit. When the bit turns on, the second TeachMover can perform the desired action and then turn on another bit to signal completion. A flowchart of two arms cooperating to hand a block from one to the other is shown in Figure 8-1.

CHAPTER EIGHT
ADVANCED APPLICATIONS

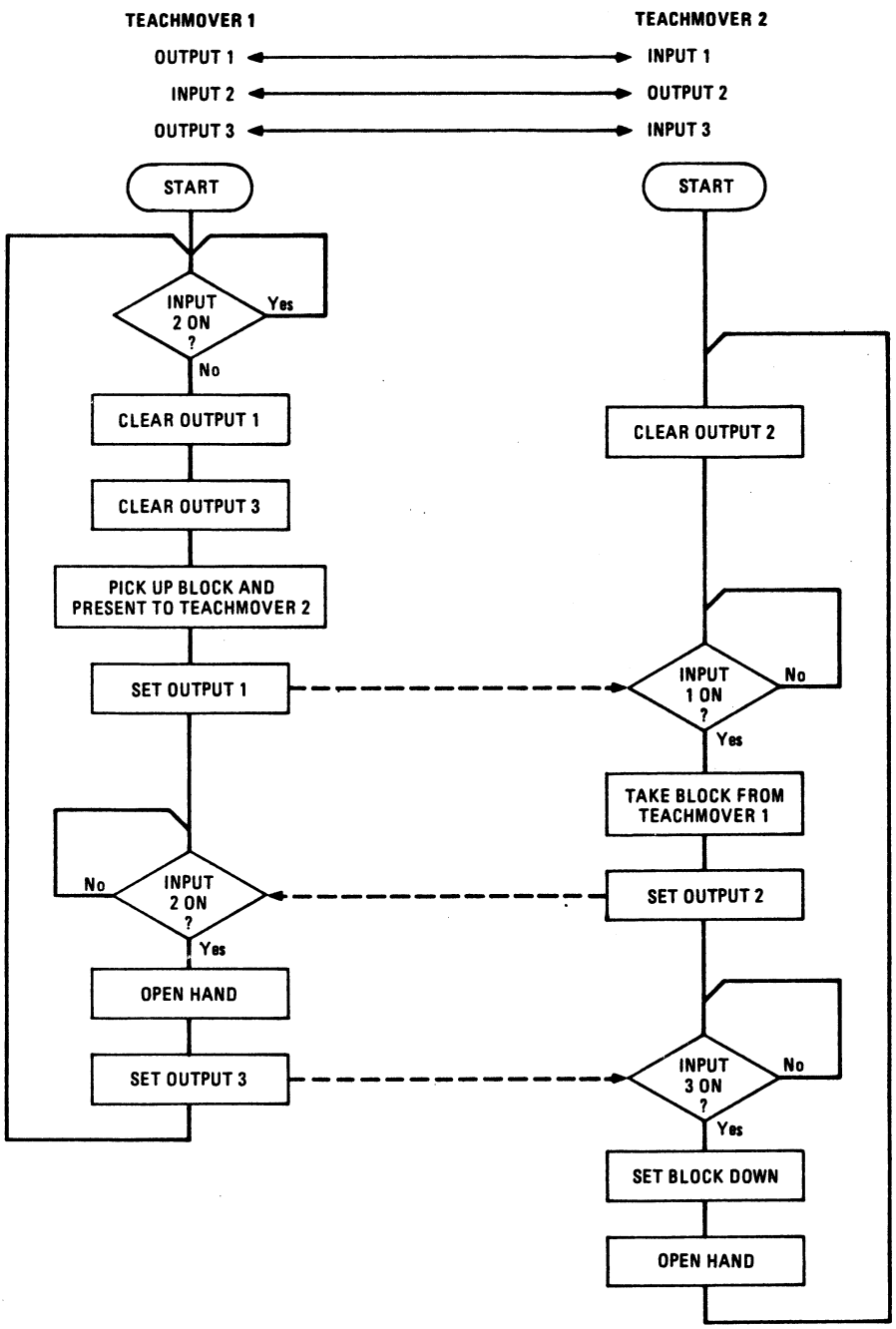


Figure 8-1 Two TeachMovers Cooperating

C. 3-D CARTESIAN POSITIONING

As we discussed in the last chapter, the TeachMover can be used to record and remember the Cartesian coordinates of various objects in its environment. To review the procedure: First initialize the arm (see Appendix G), then use the keys on the teach control to cause the tip of the hand to touch an object. Read the position of the arm (@READ command), then convert the readings into Cartesian coordinates (the formulas for this are given in Appendix D).

Once the Cartesian coordinates of an object are thus determined, it is a simple matter to command the arm to return to the object later - to move it elsewhere, change its orientation, etc. In this way, you could write a program to place a cup on a saucer, move both objects to a different position, perform some other tasks, and later retrieve the cup and saucer - all by specifying coordinates in space, that is, without having to manipulate the arm in TRAIN mode (except, of course, for determining the initial positions of both objects).

D. SENSOR CONTROL

Additional sensors can easily be interfaced to the TeachMover via the seven user inputs on the circuit card inside the base. These sensors could include:

1. Micro-switches placed on the far ends of the fingers that would sense the presence of an object in the environment. These switches would let you program the arm to reach out until it touched an object, then take a prescribed action.

2. A light-emitting diode on one finger and a photo transistor on the other to sense the presence of objects between the fingers without gripping.

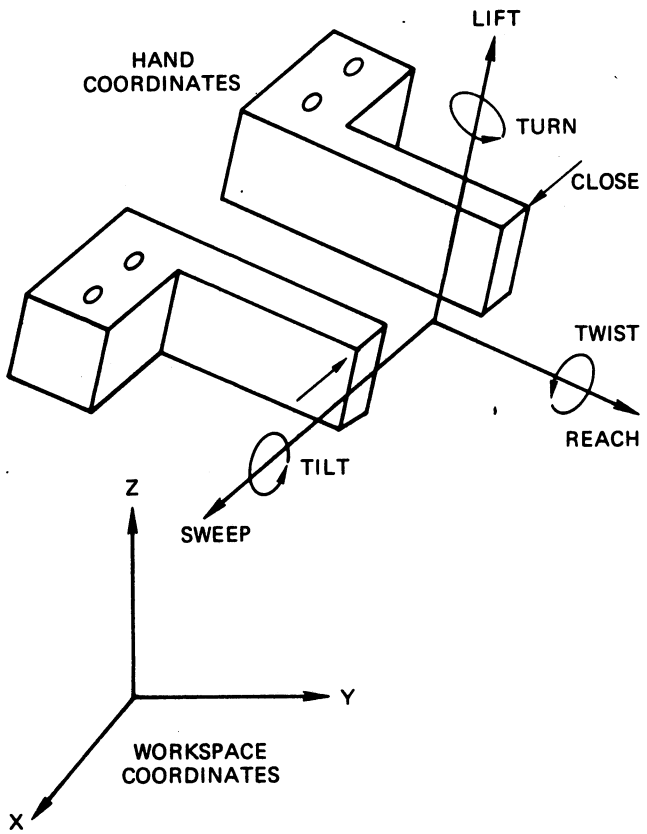


Figure 8-2 Definition of Hand Coordinate System

F. MATRIX ALGEBRA

Coordinate conversions of all kinds can be made much simpler using matrix algebra. Paul [13] discusses the use of 4 x 4 Denavit-Hartenburg matrices in robotics work. Instead of needing many program steps to convert, say, from joint coordinates to Cartesian coordinates, the use of matrix algebra lets you accomplish the conversion via a single multiplication statement. Matrix algebra also lets you incorporate several coordinate systems in one program - for example, "moving coordinates" for a conveyor belt, or modified hand coordinates to describe the motion of a hand-held tool in a simple, elegant manner.

G. LINEAR PATH CONTROL

Motion of the TeachMover arm is generally along curved paths defined by the rotation of the arm members around their joints. However, straight-line motion is often required - for example, for drawing with a hand-held pen, removing a peg from a hole, or sliding an object along a flat surface.

A system for linear path control has been proposed by Paul [13]. Paul's technique, which can be implemented in serial interface mode on the TeachMover, involves linear interpolation between the end-points of the desired trajectory. This interpolation will yield a number of "bench mark" points. If these points are close enough together, curvilinear motion between them will approximate a straight line (as in program steps 151-161 in the Exerciser Program listed at the end of Chapter 6).

H. HIGH-LEVEL, INTERACTIVE CONTROL LANGUAGE

It would, of course be more convenient to "talk" to your TeachMover in a high-level, English-like language rather than via the serial interface commands. A high-level language might let you use commands like:

- Pick up screwdriver
- Put screwdriver in box
- Pick up box
- Turn box over

Development of high-level languages is among the areas being explored today in artificial intelligence laboratories (see, for example, reference [4]). For an arm to be able to respond to high-level commands, software must be able to break up the commands into simpler tasks (for example, systematically search for and locate object, grasp object, determine by feel - or by artificial sight - if it's a screwdriver, find the box, see if it's open, and so forth). There is nothing in the design of the Teach-Mover to prevent you from developing and working with high-level languages in this way.

Keep us informed of innovative applications

We at Microbot are pleased that our low-cost Teach-Mover robot arm is so well-suited to leading-edge applications such as the above and more. Please keep us informed as to what new areas you are working on. We are committed to expanding the frontiers of robotics knowledge, and would love to hear from you.

APPENDIX A
BRIEF HISTORY OF ROBOTICS

1. MASTER-SLAVE MANIPULATORS

Some of the earliest mechanical manipulators were designed for use in radioactive "hot cells." Today, this type of manipulator (see Figure A-1) is in common use in laboratories throughout the U.S. The operator moves a "master" manipulator, and the "slave" manipulator inside the cell replicates its movements. (This terminology was originated by Ray C. Goertz of the Argonne National Laboratory in the late 1940's [7].)

The earliest master manipulators were connected to their slave manipulators via mechanical linkages. Later, flexible steel cables were employed. A more recent development is exemplified by the MA-11 master-slave manipulator, manufactured by LaCallene in France (Figure A-2). This type of manipulator is called bilateral, because it provides force feedback; in other words, the slave can affect the master. If the slave, for example, encounters an obstacle, the master becomes blocked. Without the use of force feedback, certain operations would be extremely difficult. (Imagine trying to grasp and lift a raw egg with a mechanical arm without being able to feel how tightly you are squeezing it!)

The next development in master-slave systems was the introduction of electrical control. This eliminated the need for a mechanical linkage betweeen master and slave. Instead, electric motors and potentiometers (or electrical position sensors) are mounted on both units. When the master manipulator is moved, electrical signals from its potentiometers are sent to the slave manipulator. These signals, in turn, cause the joint motors in the slave to move until the slave's potentiomenters indicate that the

APPENDIX A
HISTORY OF ROBOTICS

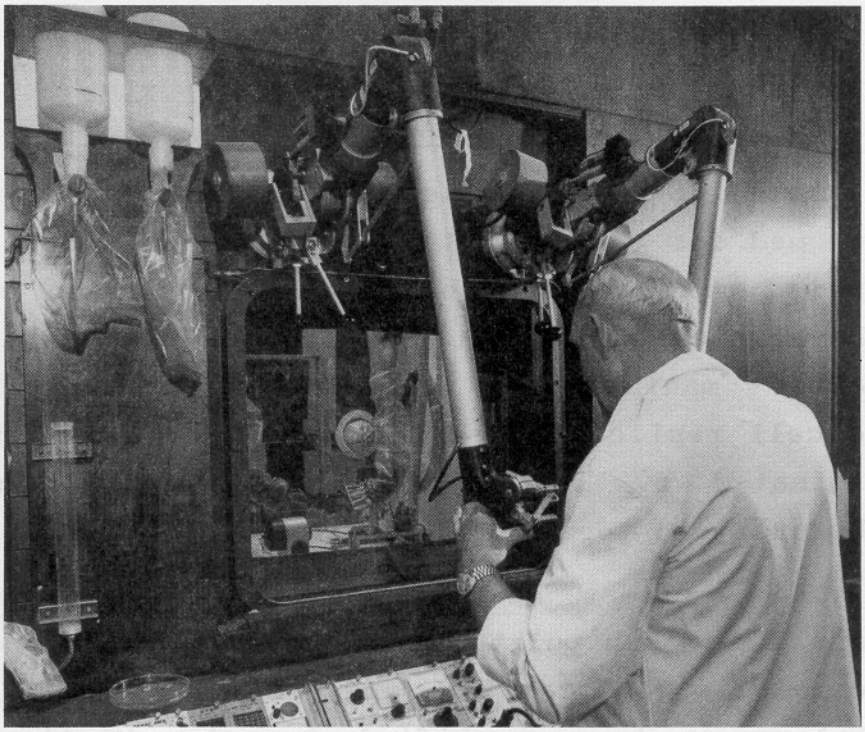


Figure A-1 Master-Slave Manipulator
in a Radioactive Environment

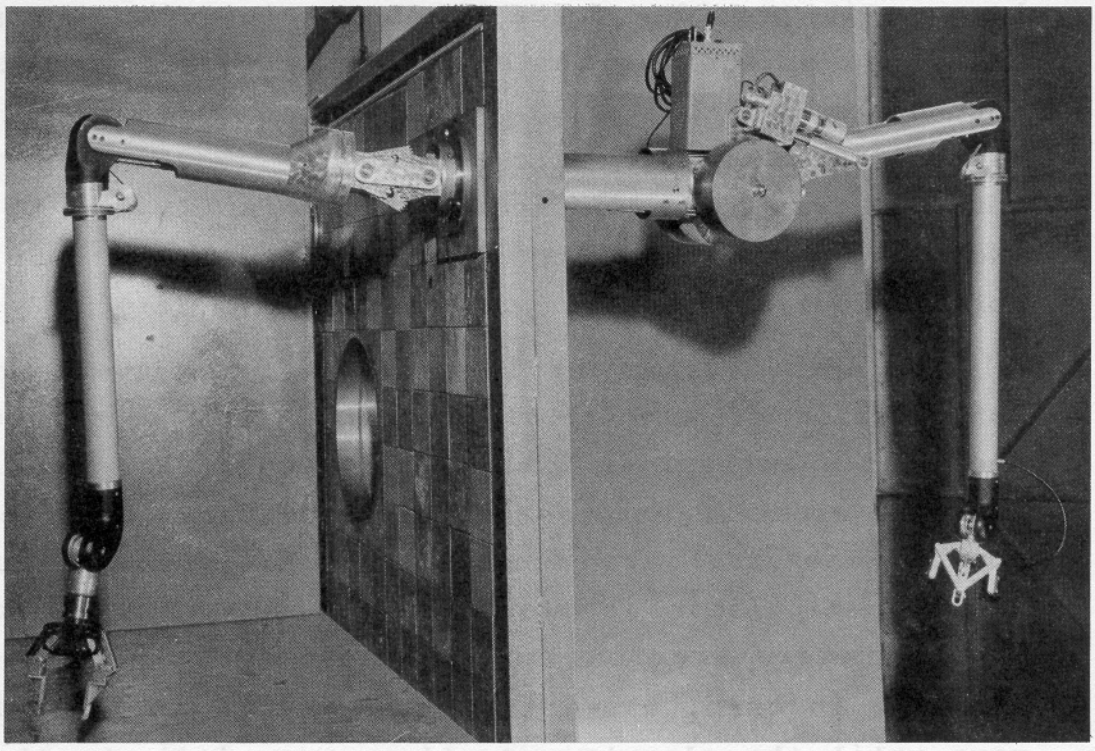


Figure A-2 The MA-11 Master-Slave Manipulator
with Force Feedback

slave's position is correct. Even though the manipulators are not mechanically coupled, force feedback can be provided. If the slave encounters an obstacle, the slave will no longer be able to follow the master exactly, and the discrepancy can be measured, converted to electrical signals, and fed back to the operator as a force.

The arm shown in Figure A-3 uses electrical control. This arm was developed from a prosthetic device by Rancho Los Amigos Hospital in Los Angeles, and is called the "Rancho Arm."

Another electrically linked manipulator system is shown in Figure A-4. This system was developed at NASA's Ames Research Center in Mountain View, California. It is based upon a "hard" space suit design originally developed for use by astronauts.

Electrically coupled systems allow master and slave manipulators to be separated by large distances rather than just a few feet. This factor, in turn, set the stage for the development of teleoperator systems.

2. TELEOPERATORS

A teleoperator system is essentially the same as an electrically coupled system, except that the communications link can be wireless, and the "master" need not be a manipulator at all; it might instead be a panel of buttons and dials. The word teleoperator is, in fact, defined rather broadly as "a general-purpose, dextrous, cybernetic machine" [7]. The characteristics of such a "machine" are given in Figure A-5.

The "barrier" can be a long distance, a task difficulty, or a hostile environment. In addition to force feedback, other kinds of information can be given from sensors located at the "remote" end of the system (TV

APPENDIX A
HISTORY OF ROBOTICS

Figure A-3 The Electrically Coupled "Rancho Arm"

Figure A-4 NASA Ames Master-Slave Manipulator

APPENDIX A
HISTORY OF ROBOTICS

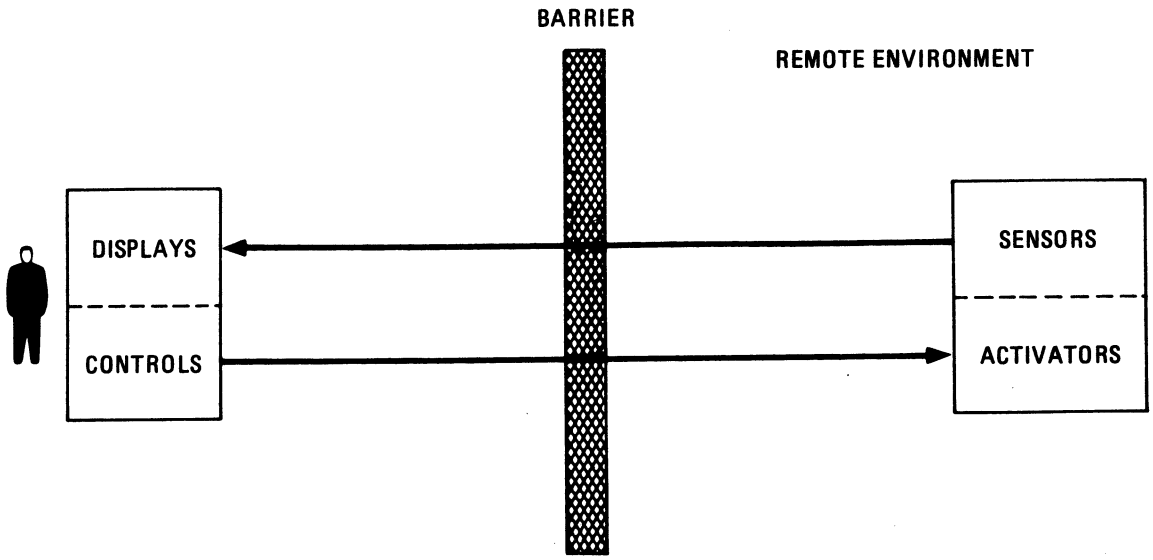


Figure A-5 Block Diagram of a Teleoperator System

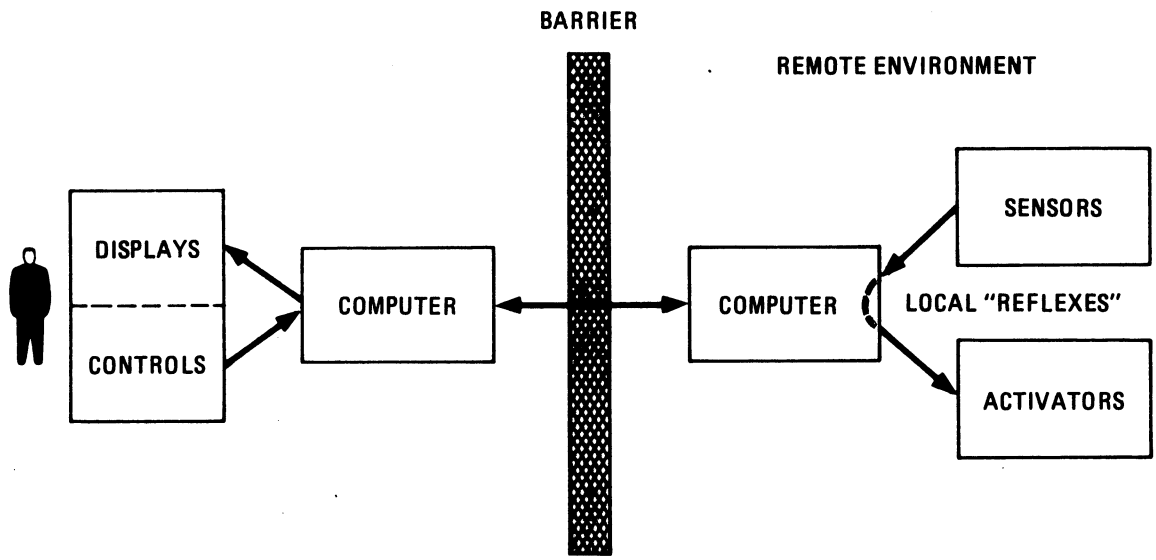


Figure A-6 Computer-Augmented Teleoperator System

cameras, for example). The operator's controls could be of a variety of forms. In addition to buttons and dials, there could be a joystick similar to the kind used in certain aircraft, or a "master replicate controller" (an arm with the same geometry as the slave, but usually scaled down in size).

Those interested in teleoperators will find reference [7] worth reading.

3. COMPUTER-AUGMENTED TELEOPERATORS

The development of computer-augmented teleoperator systems was primarily the result of research sponsored throughout the country by NASA. The essential element in these systems is the addition of "intelligence" in the form of one or more local computers (local means directly associated with the slave or with the master-see Figure A-6).

To understand the benefits of such a system, consider an application where an operator at an earth station is controlling a manipulator on the surface of the moon in order to collect lunar rock samples. In this case, approximately 1.4 seconds are required to transmit signals from the earth to the moon, and another 1.4 seconds to transmit them back. Thus, the operator's view of the remote work site is always 1.4 seconds late, and the remote manipulator will not even begin to respond to the operator's command until an additional 1.4 seconds has passed. Thus, if something unexpected happens at the remote work site, serious damage might occur before anything could be done about it. Another problem is that if something unexpected does happen, the operator will usually be forced to adopt a complex "move-and-wait" control strategy, moving the manipulator slightly, waiting

to see what has happened, and again moving it slightly [6]. Most operators find this procedure extremely tedious and tiring.

With a remote computer at the work site, local "autonomous" reactions can be programmed directly into the system. ("Autonomous" is used as in the human autonomous nervous system. When a person inadvertently touches something that is hot, the autonomous nervous system takes over and quickly pulls the hand back - in many cases even before the person is aware of any pain.) The lunar manipulator, for example, could be programmed to immediately stop moving if sensors on the manipulator indicate that the unit has encountered an unexpected obstacle. Furthermore, the remote computer could be programmed to aid the human operator by picking up lunar rocks and depositing them into a container automatically. All the human operator would have to do is to move the manipulator to the vicinity of a rock, then command the manipulator to pick the rock up. The remote computer would then take over control from the operator, carry out the various stages of the task, and then return control to the operator once again.

4. A ROBOT WITH EYES

After the introduction of computer-augmented tele-operator systems, it became increasingly clear that under certain circumstances a comletely automatic system would be feasible. In the mid 1960's, at Stanford University, researchers removed the master arm from the Rancho Arm system (Figure A-3), and placed the slave manipulator under computer control. Using early computer vision techniques, the positions and orientations of each of several blocks randomly placed upon a table top were determined. The Rancho manipulator was then commanded to

stack the blocks. The arm was affectionately called "Butterfingers" because it frequently dropped one or more of the blocks.

5. INDUSTRIAL ROBOTS

Once robots were shown to be feasible in the laboratory, they were initially put to work in factories on highly repetitive, boring, and/or dangerous tasks. Today thousands of industrial robots are used in factories in the U.S.A. and overseas, and help manufacture many of the products we use every day.

One of the early industrial robots was the Unimate Series-4000 manipulator shown in Figure A-7.* This unit, used on automobile assembly lines, resulted in a significant increase both in productivity and worker satisfaction. The Unimate could move faster and lift heavier loads than its human counterpart. Moreover, for such operations as placing workpieces in equipment such as punch presses, accidents were greatly reduced. Even when accidents did occur, they did not result in the loss of a worker's hands, but only in the loss of replaceable mechanical equipment.

A manipulator called the T^3, or The Tomorrow Tool (Figure A-8), was introduced by Cincinnati Milacron in the mid-1970's. The T^3 can lift up to 100 lbs. can move at a speed of up to 50 inches per second, has a working volume of 1000 cubic feet, can reach to a height of almost 13 feet, and has an accuracy of 0.050 inches.

*Figure A-7 and Figure A-10 are courtesy of Unimation, Inc.

APPENDIX A
HISTORY OF ROBOTICS

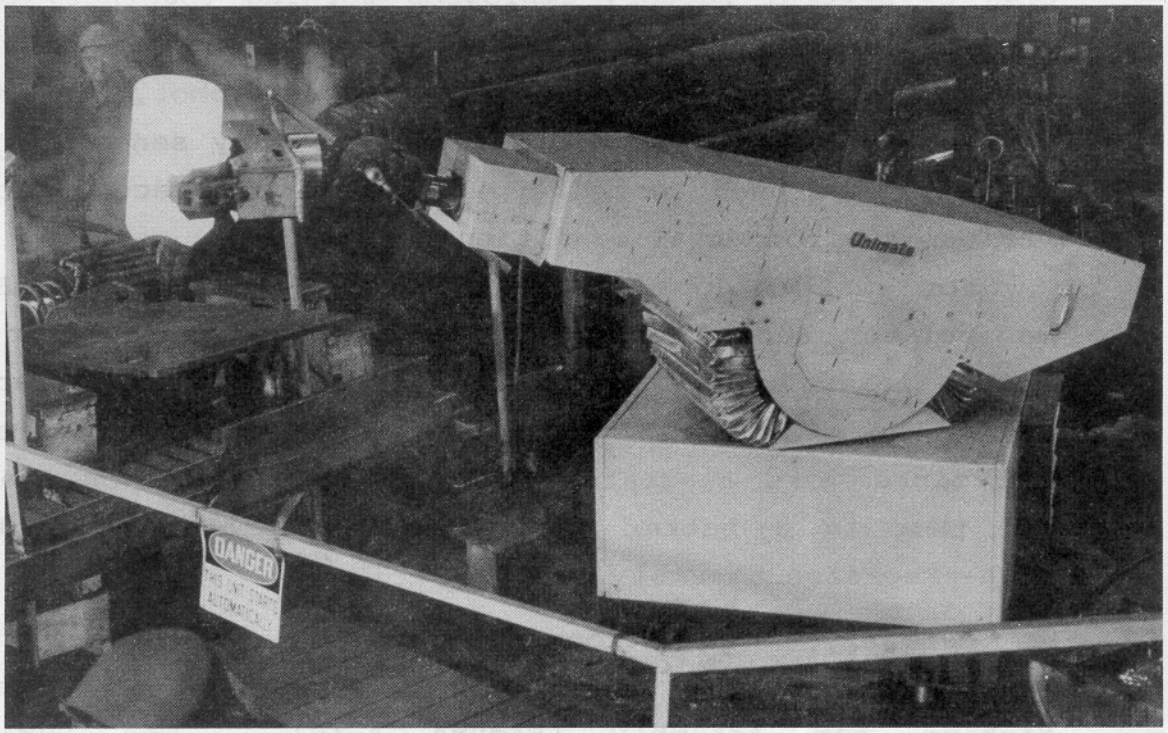


Figure A-7 Unimate Industrial Robot by Unimation, Inc.

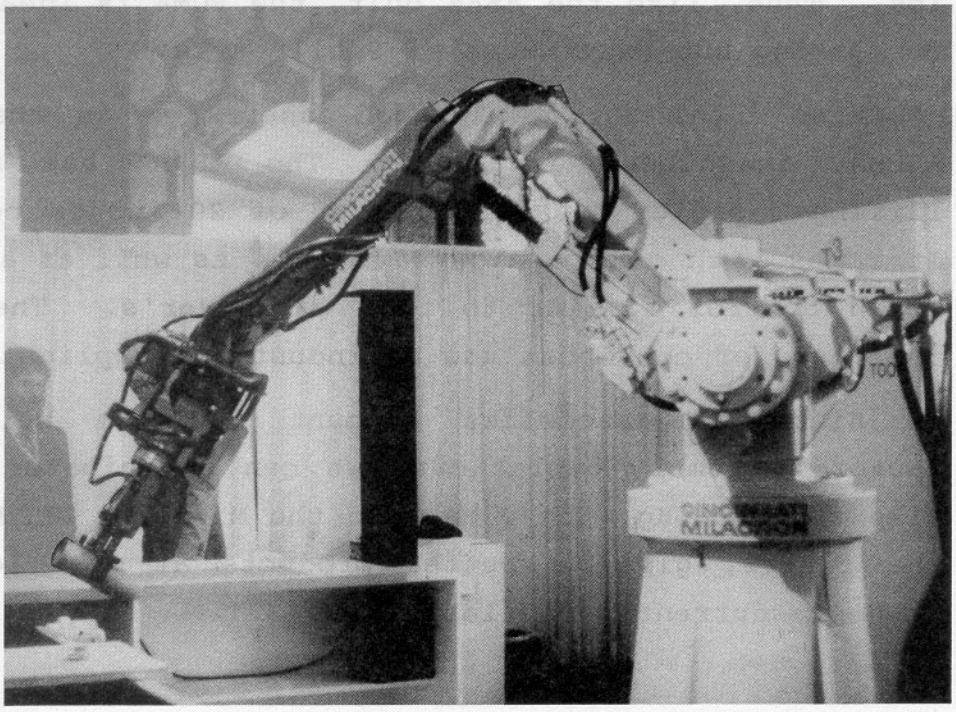


Figure A-8 Cincinnati Milacron "T^3" Industrial Robot

It can accomplish tasks that previously could be performed only by several factory employees working together, or by special purpose equipment costing much more than the T^3 costs. The T^3 is a general-purpose robot, as it is completely computer controlled and has many sensory inputs which can be used to modify its behavior to accomodate to various environmental conditions.

For smaller jobs, ASEA in Sweden, markets the IRB-6Kg manipulator shown in Figure A-9. This manipulator can lift a load of 13 lbs. and has a reach that is comparable to the human arm. However the IRB-6Kg is approximately ten times more accurate than the Series-4000 or the T^3, and thus is a better choice for applications involving high-precision assembly.

Recently, Unimation introduced a high-accuracy manipulator called the PUMA or Programmable Universal Machine for Assembly (Figure A-10). The PUMA is accurate to 0.005 inches and can lift approximately 7 lbs. Thus, like the ASEA unit, the PUMA is ideally suited for making sub-assemblies of small parts.

Also ideal for making sub-assemblies is the Microbot ALPHA, introduced in 1982. The ALPHA has a 1.5 lb. lifting capacity and a top speed of 20 inches per second. It comes with an operator's console as well as a hand-held teach control similar to the TeachMover's. The ALPHA is designed for continous use in industrial applications.

Since sub-assemblies account for the bulk of all factory assembly operations, we can anticipate that units like the IRB-6Kg, the PUMA, and the Microbot ALPHA will, in the future, give us better products at even lower costs than are currently available.

APPENDIX A
HISTORY OF ROBOTICS

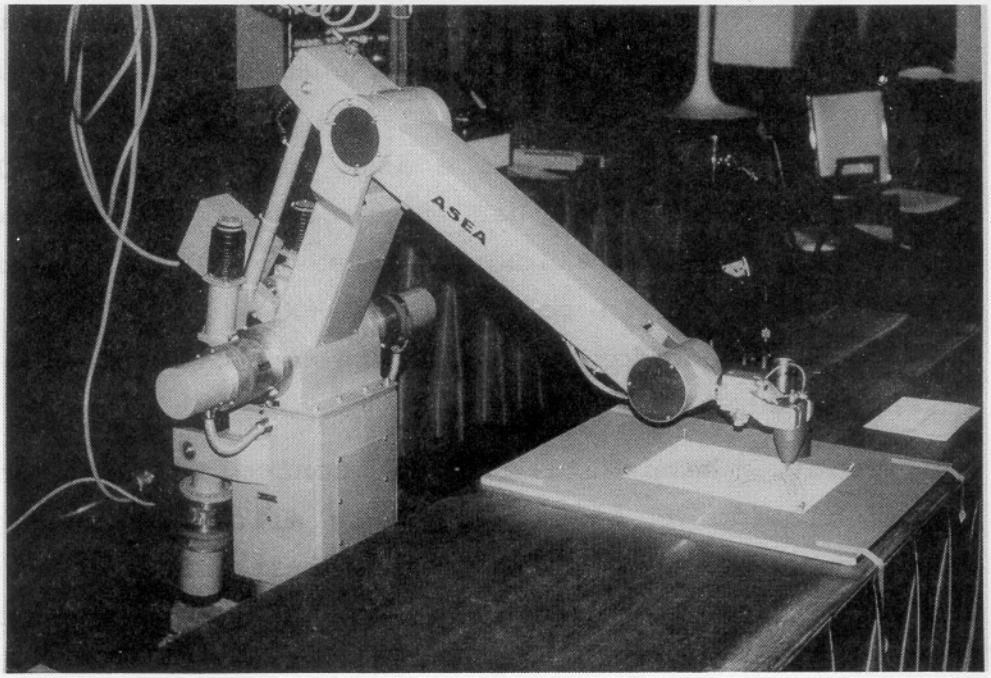


Figure A-9 The ASEA IRB-6Kg Industrial Robot

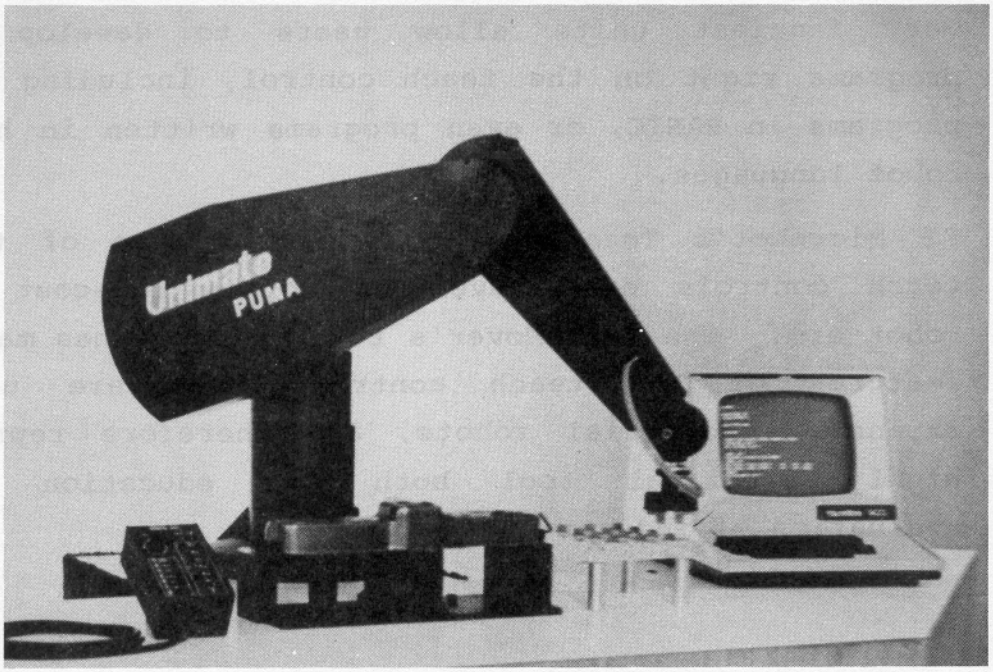


Figure A-10 Unimation PUMA Industrial Robot

6. INDUSTRIAL TEACH CONTROLS

Early hand-held teach controls were shaped like miniature versions of the arms themselves. The user would push a button on a member of the control unit to move the corresponding member on the arm. By the mid-1970's, hand-held teach controls had increased in sophistication to where some units allowed users to specify arm motions directly in "world coordinates" (X, Y, Z, etc.). By this time the need for a miniature version of the arm had disappeared.

Today, hand-held teach controls vary widely in sophistication. Some units only allow for moving the arm and recording its positions, other units have small LED screens that can display a limited number of canned messages. The most sophisticated units are really hand-held computer terminals that can display anything they are commanded to display; these same units often have multi-function programmable keys, with different overlays for different applications (welding, assembly, etc.). The very fanciest units allow users to develop computer programs right on the teach control, including Cartesian programs in BASIC, or even programs written in high-level robot languages.

Microbot's TeachMover incorporates one of the first teach controls ever developed for a low-cost tabletop robot arm. The TeachMover's teach control has many of the features of the teach controls that are used with expensive industrial robots, and therefore represents a highly practical tool both for education and for evaluation of industrial equipment.

APPENDIX A
HISTORY OF ROBOTICS

7. THE FUTURE OF MANIPULATORS

At many research laboratories throughout the country, research is being conducted in the fields of robotics and artificial intelligence. Areas currently being explored are: vision sensors [2], [8]; force and tactile sensors [11], [14]; proximity sensors [17]; compliant devices for assembly [18]; robot assembly [21]; trajectory calculations [12]; kinematics of arms [9]; manipulation languages [4]; and automatic task planning [10]. Microbot, Inc. is part of this research effort, and among our current projects is a new generation of industrial robots with increased sensory capability.

A survey of many of the current areas of robotics research is given by Abraham et al, reference [1].

APPENDIX B
ARM INITIALIZATION AND CALIBRATION

The computer in the robot keeps track of the arm by using the starting position as a reference. To run a program to operate the robot arm, you must know what the starting position was when the program was created. To make general programming more convenient, a "normal" starting position is defined. This initial position requires that the robot arm be placed exactly on a grid sheet marked off in a cartesian coordinate system with a scale of one inch per square. A grid sheet is inserted inside the back cover of this manual for your covenience. (Figure B-1)

With the origin of the coordinate system located at the axis of rotation of the base of the robot arm, the location of the front edge of the basse of the arm is defined as x = 1-5/8. The base is centered on the y-axis. The gripper is brought to rest, barely touching, on the spot P0, shown on the grid at X=5 and Y=0. (Points P1, P2, and P3 are positions at which blocks are placed for the demonstration programs discussed in Chapter 6 of this manual).

As shown in the illustration Figure B-2, the hand must be perpendicular to the work surface and parallel to the front edge of the base of the robot. Keep the gripper open as you bring the arm to this position, and then close the gripper as the last step in setting the initial position. The gripper can have a gripping force applied to it by the motor after the point of just closing. The initial position assumes no gripping force beyond closing.

The arm can be brought to this starting position by moving it manually with the power off or by using the teach control with the power on. Be sure to clear the memory and internal registers after achieving the starting

position. Refer to Chapter 2, Section E, for this procedure.

Specifically, the Cartesian coordinates of the initial configuration are:

X = 5 inches
Y = 0 inches
Z = 0 inches
Pitch = -90^{o}
Roll = 0^{o} (see Note 1, below)
Grip = Closed (see Note 2, below)

Note 1 : Because the hand can turn through many revolutions of "roll," it is difficult to tell by simply looking at the hand whether the "roll" has been set to 0^{o}. Yet, it is important that the roll initially be 0^{o} in order that the wrist cables be allowed their full range of motion. To accomplish the proper initialization of the wrist cables, turn the appropriate main drive gears until the turnbuckles on the left and right wrist cables are aligned as shown in Figure B-3. You'll find these turnbuckles inside the forearm housing.

Note 2 : For many programs, it is important that the initial position be very precise - for example, that the fingers be just touching (grip switch closed, but no gripping force built up), and that the fingertips be exactly horizontal and on the calibration mark at PO. To achieve this precision, it is best to key in a SPEED number of 0, then use the joint control keys in TRAIN or MOVE mode.

APPENDIX B
ARM INITIALIZATION

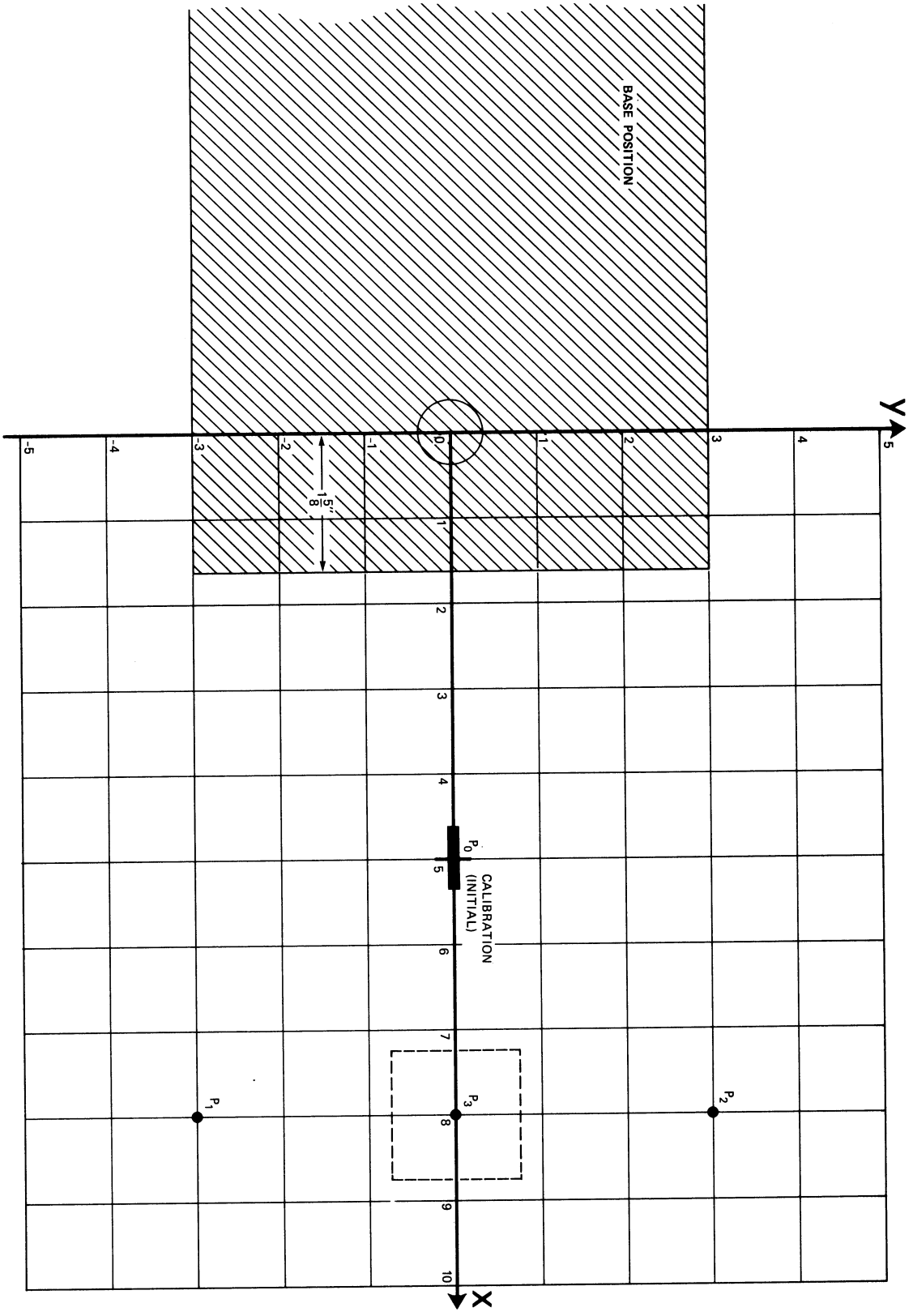


Figure B-1 Initialization and Calibration Grid

APPENDIX B
ARM INITIALIZATION

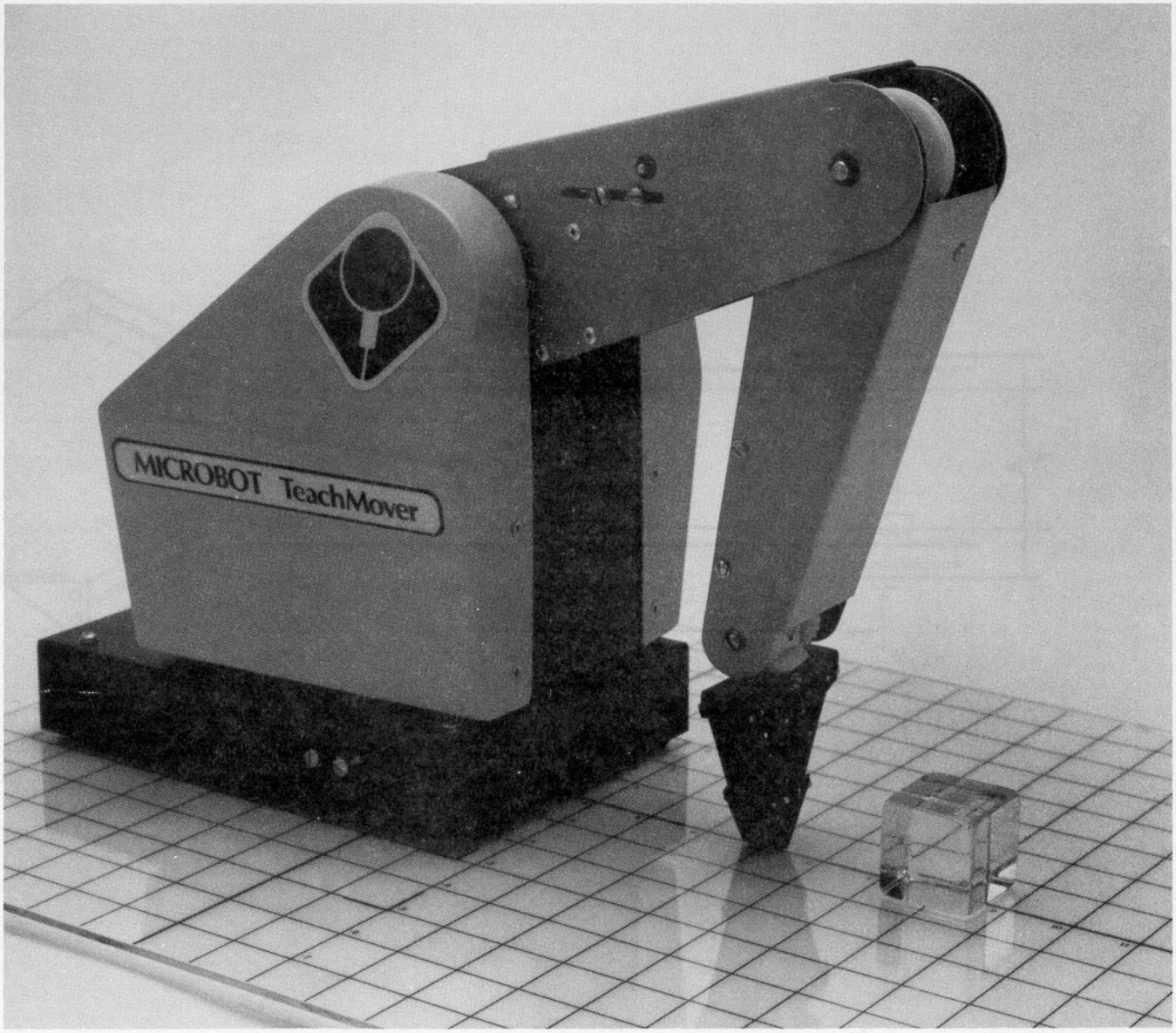


Figure B-2 Initialization Position

APPENDIX B
ARM INITIALIZATION

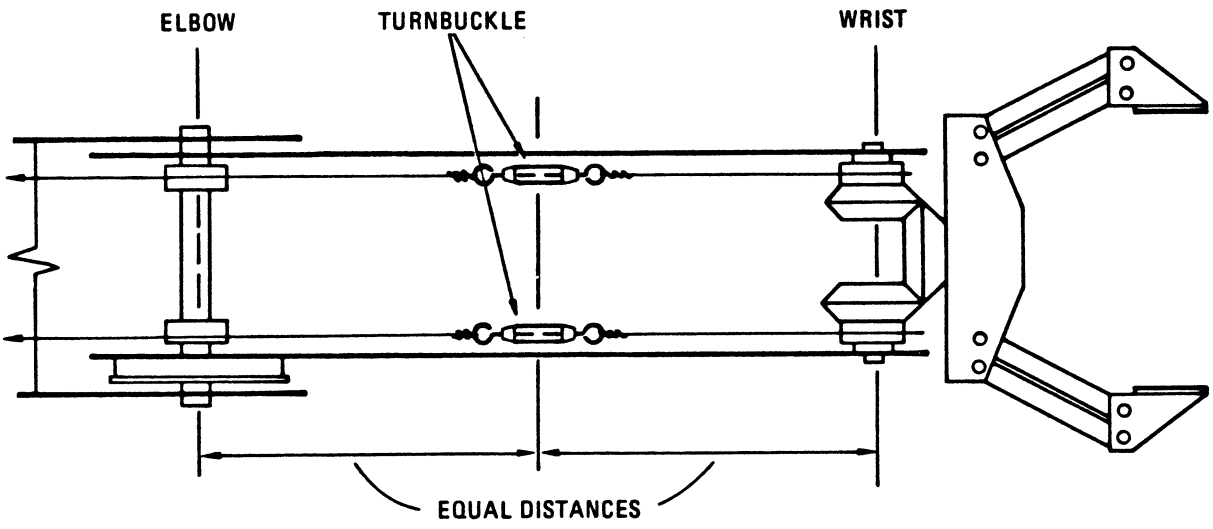


Figure B-3 Alignment of Turnbuckles for Initialization

APPENDIX C
ADDING ADDITIONAL RAM

The TeachMover comes supplied with enough Random Access Memory for 53 program steps. You can more than double this by adding more RAM.* This is done by "piggy backing" two RAM chips on top of the existing RAM chips, and jumpering the chip select lines of these chips onto pads on the circuit card. Briefly, the procedure is as follows:

1. Bend up Pin 8 of each of two 2114-2 1K x 4Bit RAMS
2. Carefully align the two additional RAM chips with U16 and U17 as shown on Figure 12, making sure that Pin 1 of each new chip aligns with Pin 1 of the corresponding chip on the circuit card.
3. Carefully solder all of the pins of the additional RAM chips to those of the chips on the board, with the exception of both Pin 8s.
4. Connect a wire between both Pin 8s and the jumper E1 shown in the upper right corner of Figure 12.
5. Remove any solder splashed on the board or between pairs of pins.

Figure C-1 shows the completed modification.

---

*This modification must be performed by a skilled technician experienced in circuit card modification techniques. Because of the potential for damage to the system electronics which might result from improper modification techniques, any malfunctions or damage resulting from this modification will be the responsibility of the user and will not be covered by the TeachMover warranty.

APPENDIX C
ADDITIONAL RAM

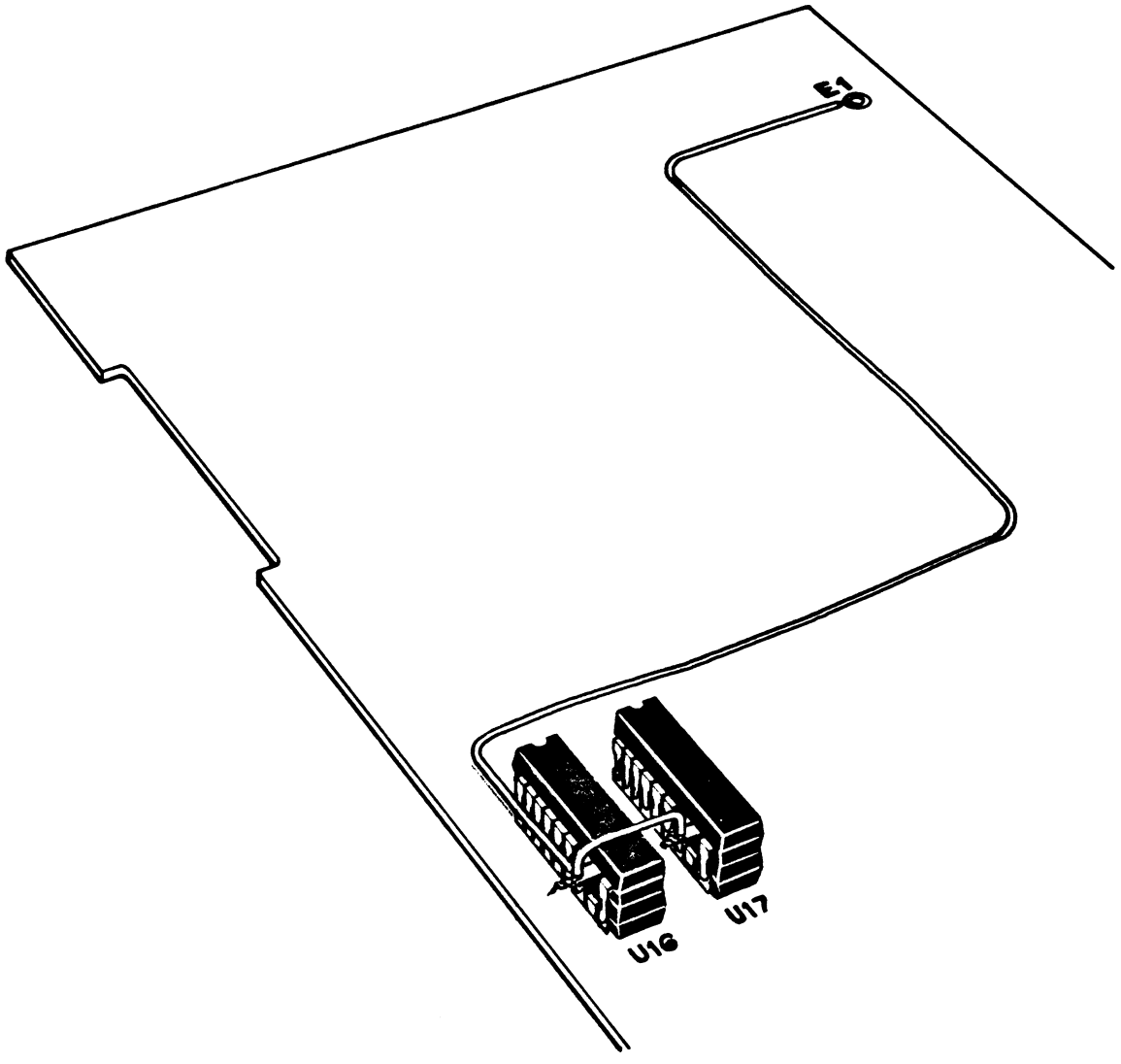


Figure C-1 Adding Extra RAM Chips

APPENDIX D
COORDINATE CONVERSIONS

It is often advantageous to be able to describe the configuration of a robot arm in more than one coordinate system. The two most commonly used systems are:*

Joint and cartesian coordinates

- Joint Coordinates (the joint angles of the arm). These are most convenient for controlling the arm directly from a computer.
- Cartesian Coordinates (X, Y, Z, pitch, and roll). These are more convenient for describing an assembly task on a flat table top.

For practical work, we need a set of formulas for mathematically converting from one coordinate system to the other.

Foreward and backward solutions defined

- The Forward Solution converts from joint angles to Cartesian coordinates.
- The Backward Solution converts from Cartesian coordinates to joint angles.

This appendix describes how both of these coordinate systems are defined, and how the forward and backward solutions may be derived and implemented.

To best understand the forward and backward solutions you should be familiar with basic trigonometry. It is not necessary to understand the solutions, however, to program the TeachMover. The material presented in part 1 of this

---

* Other coordinate systems commonly used are: hand coordinates with origin at the fingertips and axes aligned with the hand; workspace coordinates located on, and aligned with, a particular work station; moving coordinates located on, and aligned with, a conveyor belt or turntable.

appendix is sufficient for using the solution programs (in BASIC) given later (in Tables D-3 and D-4, and Figure D-12). An application program given in Chapter 7, part C, 3 shows how the backward solution may be incorporated into a program.

1. KINEMATIC MODEL OF ARM

Before we can formulate the arm solutions, the relationship between the different parts of the arm must be specified. This can be done in terms of the kinematic model shown in Figure D-1. The kinematic model indicates how each joint is articulated, how the joint angles are measured, and the distances between joints.

The Greek letter, θ, is frequently used to indicate joint angles in mathematical expressions. The symbols θ_1, θ_2, θ_3, θ_4, and θ_5, respectively, are proportional to the joint expressions J_1, J_2, J_3, J_4, and J_5 used in the computer command discussed in Chapter 7. The θs, measured in degrees or radians, are related to the Js, measured in motor steps, as shown in Table D-1. There are 360 degrees or 2π radians in one complete revolution.

Table D-1
Conversion Factors Between Motor Steps
and Revolute Joint Angles

Motor	Joint	Steps in one Revolution	Steps per Radian	Steps per Degree
1	Base	7072	1125	19.64
2	Shoulder	7072	1125	19.64
3	Elbow	4158	672	11.55
4	Right wrist	1536	241	4.27
5	Left wrist	1536	241	4.27

The distances between joints (lengths of arm members) are indicated by the constants, H, L, and LL shown in

Kinematic Model Of Arm

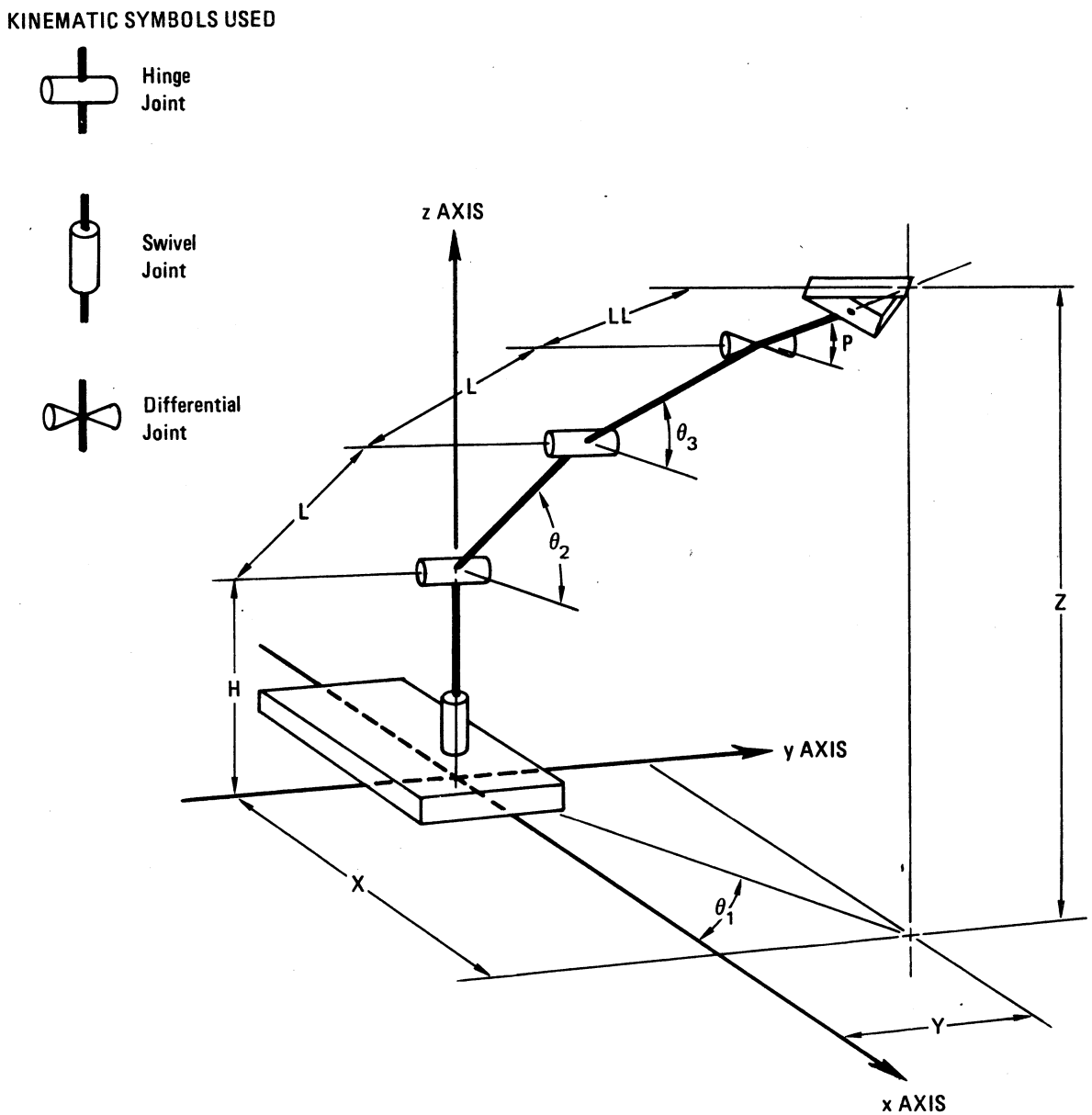


Figure D-1 Kinematic Model of the TeachMover Arm

Figure D-1. H is the distance from the table top to the shoulder joint centerline; L is the distance from shoulder joint to elbow joint, which equals the distance from elbow joint to wrist joint; and LL is the distance from the wrist joint to the center point between the two fingertips, with the fingertips separated by 1.5 inches. Values for these distances are given in Table D-2.

Table D-2

Lengths of TeachMover Arm Members

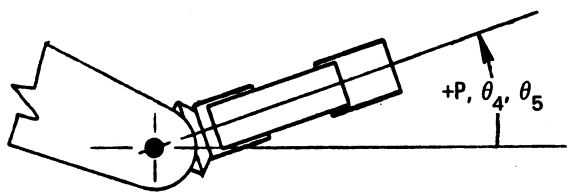
Segments	Length (inches)	Length (mm)
H	7.68	195.0
L	7.00	177.8
LL	3.80	96.5

The pitch angle, P, and the roll angle, R, are given by the following equations.

$$P = .5(\theta_5 + \theta_4) \quad (1)$$

$$R = .5(\theta_5 - \theta_4) \quad (2)$$

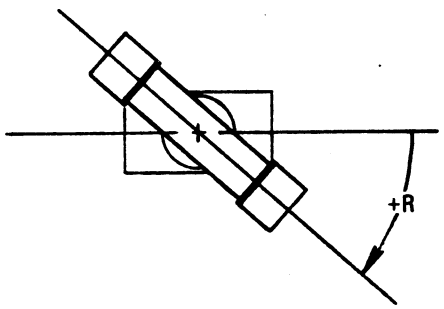
where θ_4 and θ_5 are right and left wrist angles. The angles P, θ_4, and θ_5 are all measured from the horizontal as shown in Figure D-2.



PITCH ANGLE is the orientation of the hand to the horizontal.

Upwards is positive.

SIDE VIEW



ROLL ANGLE is the orientation of the hand looking towards it along its centerline.

Clockwise rotation is positive.

END VIEW

Figure D-2 Definition of Roll and Pitch Angles

2. FORWARD ARM SOLUTION

This section shows how to determine P, R, and the X, Y, and Z coordinates of the end point* from the joint angles θ_1, θ_2, θ_3, θ_4, and θ_5. The coordinates and joint angles are defined in Figure D-1. This solution relies on the trigonometric relationships** given in Figure D-3 for reference.

The first step is to determine Z, the height of the end point above the table top, and an intermediate variable RR, the horizontal distance from the base pivot to the end point. The situation is summarized in Figure D-4. Summing the vertical contributions from each link gives the following expression for Z:

$$Z = H + L \sin \theta_2 + L \sin \theta_3 + LL \sin P \quad (3)$$

Summing the horizontal contributions gives:

$$RR = L \cos \theta_2 + L \cos \theta_3 + LL \cos P, \quad (4)$$

where pitch angle P is given by

$$P = .5(\theta_5 + \theta_4). \quad (5)$$

The second step is to determine the X and Y coordinates of the end point from the intermediate variable, RR, as shown in Figure D-5. By inspection, the coordinates are:

$$X = RR \cos \theta_1 \quad (6)$$

$$Y = RR \sin \theta_1 \quad (7)$$

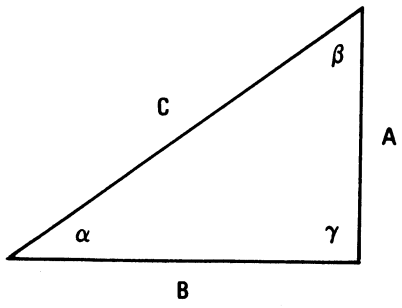
---

*The end point refers to the end point of the hand or, alternatively, the center point between the two fingertips.

**Readers unfamiliar with trigonometry will find this material covered in basic trigonometry textbooks.

Forward Arm Solution

A summary of this forward solution is given in Table D-3. A BASIC program implementing this solution is given in Figure D-12 (Statements 460 to 510). The programs variables T_1, T_2, ..., T_5 correspond to the angles θ_1, θ_2, ...,θ_5.



ANGLE FORMULAS:

$\alpha + \beta + \gamma = 180^\circ$

for a right triangle $\gamma = 90^\circ$

and $\alpha + \beta = 90^\circ$

PYTHAGOREAN THEOREM:

$C^2 = A^2 + B^2$, or

$C = \sqrt{A^2 + B^2}$ or $A = \sqrt{C^2 - B^2}$

RATIOS OF SIDES:

$\sin \alpha = \frac{A}{C}$ or $A = C \sin \alpha$

$\cos \alpha = \frac{B}{C}$ or $B = C \cos \alpha$

$\tan \alpha = \frac{A}{B}$ or $A = B \tan \alpha$

ANGLE DEFINED BY INVERSE FUNCTION:

$\alpha = \tan^{-1}\left(\frac{A}{B}\right)$

Figure D-3 Basic Trigonometric Relationships

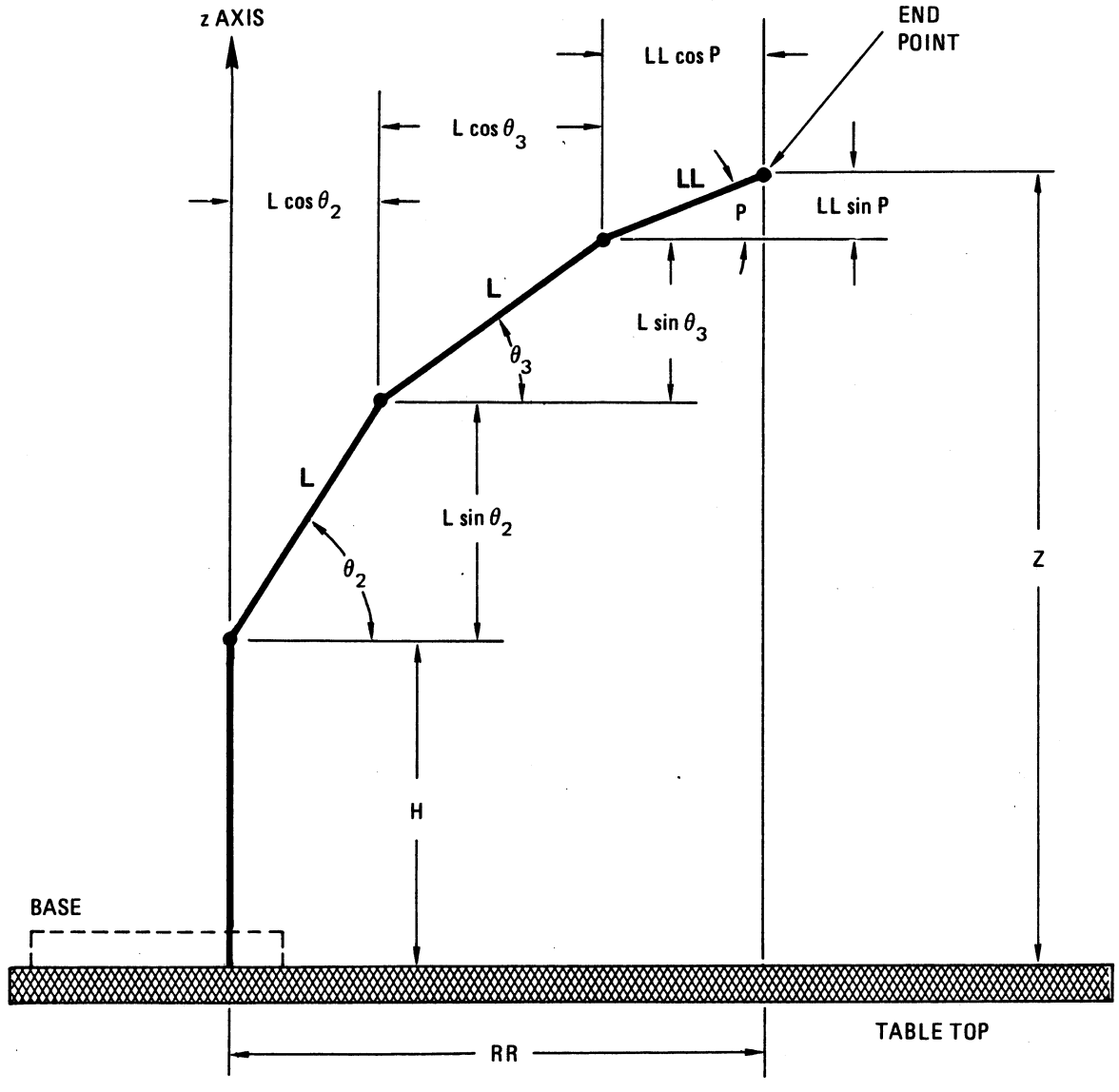


Figure D-4 Side View of Kinematic Model

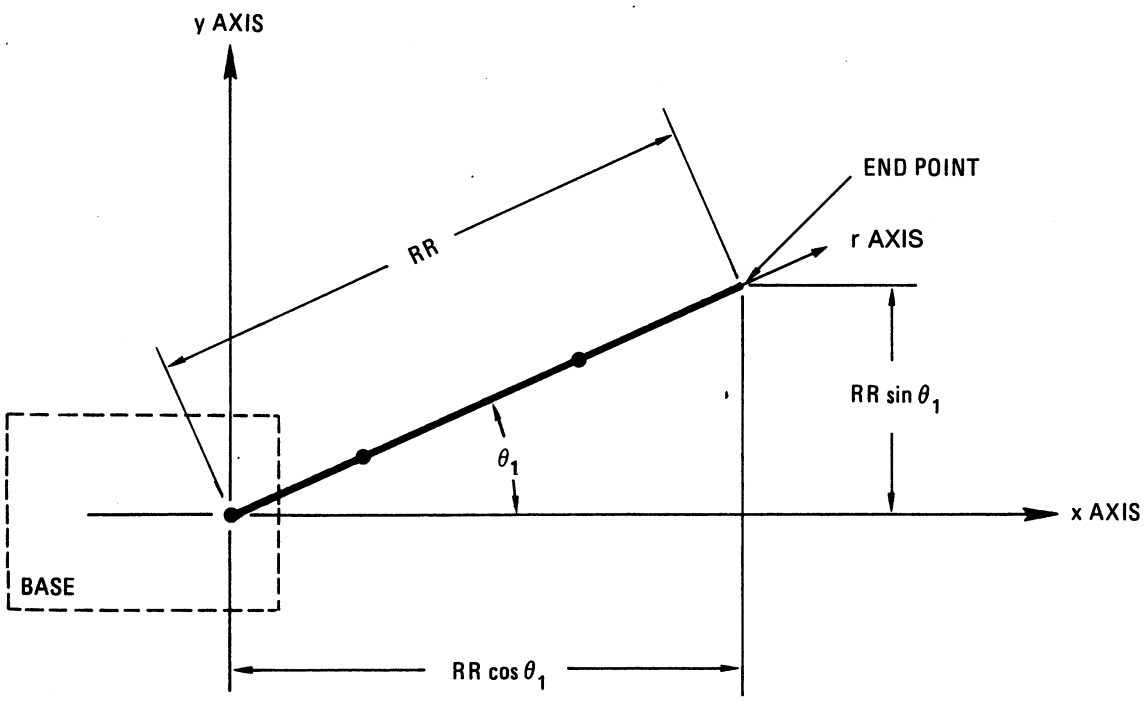


Figure D-5 Top View of Kinematic Model

Table D-3

Summary of Forward Solution

Step	Operation
1	$P = (\theta_5 + \theta_4)/2$
2	$R = (\theta_5 - \theta_4)/2$
3	$RR = L \cos \theta_2 + L \cos \theta_3 + LL \cos P$
4	$X = RR \cos \theta_1$
5	$Y = RR \sin \theta_1$
6	$Z = H + L \sin \theta_2 + L \sin \theta_3 + LL \sin P$

3. BACKWARD ARM SOLUTION

This section shows how to determine the joint angles θ_1, θ_2, θ_3, θ_4, and θ_5 required to position the end point at a desired X, Y, Z position and with desired values of pitch and roll. The coordinates referred to are shown in Figure D-1. A review of the formulas used is given in Figure D-3.

A. Specifying Position/Orientation - X, Y, Z, P, and R

Before starting the backward solution it is necessary to specify the desired position and orientation of the end point. The position of the end point is defined by the following three distances:

X: The distance of the desired end point in front of the arm, measured from the base pivot along the X-axis.

Y: The distance of the desired end point to the left of the arm, measured from the base pivot along the Y-axis.

Z: The vertical height of the desired end point above the table top.

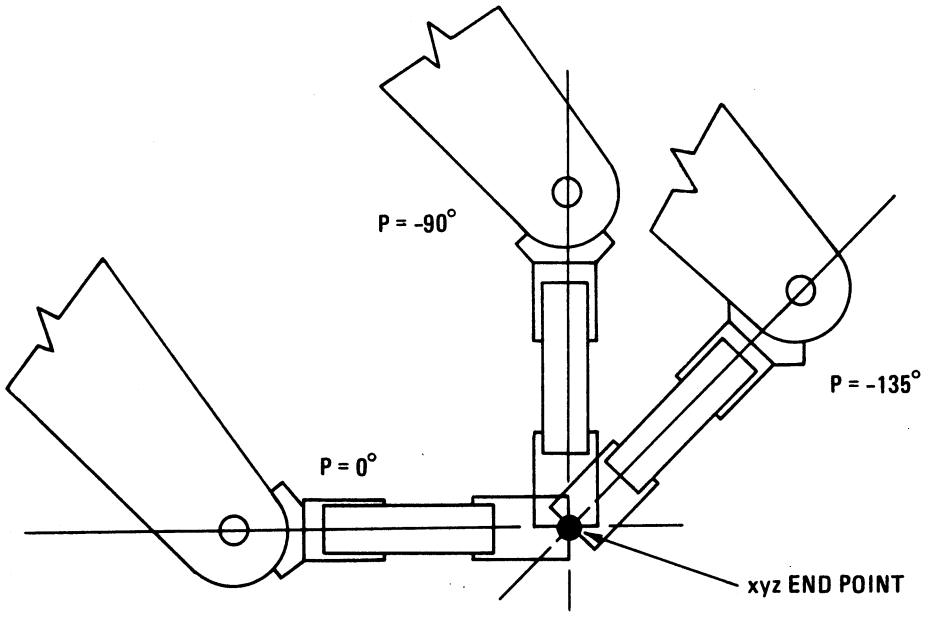
The units of these distances (inches or millimeters) should match the units of the segment lengths shown in Table D-2.

The orientation at the end point is defined by the following two angles (see Figure D-6):

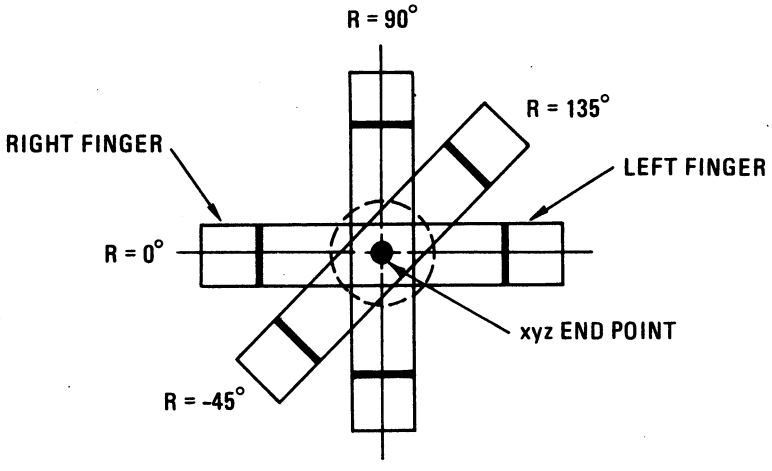
P: The desired pitch angle, measured in degrees

R: The desired roll angle, measured in degrees

In practice it is difficult to distinguish between positive and negative roll angles (as $+90^{\circ}$ and -90°, or $+45^{\circ}$ and -135°) by looking at the hand. It is helpful to mark the top of the hand when it is at 0° to



(a) DIFFERENT PITCH ANGLES AT SAME ENDPOINT



(b) DIFFERENT ROLL ANGLES AT SAME ENDPOINT.
View looking into front of hand along pitch vector.

Figure D-6 Different Hand Orientations

eliminate this ambiguity. The 0^{o} position corresponds to the orientation when the wrist cable turnbuckles are aligned, as discussed in Appendix B.

B. Specifying Roll in Cartesian Frame, R'

Sometimes it is useful to express "roll" with respect to a Cartesian frame rather than with respect to the arm. One way to do this is to use $P = -90^{o}$ (hand pointing down) as a reference orientation, and measure the "Cartesian roll" with respect to the x-axis, as indicated in Figure D-7. The formula relating the roll measured with respect to the arm (R) and the roll measured with respect to the Cartesian frame (R') is then simply:

$$R' = R - \theta_1$$

In the backward solution, we introduce a special variable, Rl, that enables us to write equations that are valid regardless of whether roll is measured with respect to the arm or with respect to the Cartesian frame.

Rl = 1 if roll is with respect to Cartesian frame.
Rl = 0 if roll is with respect to arm frame.

With this new variable, Equation (8) can be modified to express both normal and Cartesian roll as follows:

$$R' = R - \theta_1 Rl \qquad (9)$$

Solving for R gives:

$$R = R' + \theta_1 Rl \qquad (10)$$

C. Backward Solution, Step-by-Step

The first step of the backward solution is to determine the base angle, θ_1, and the radius vector, RR, from the base to the end point as shown in Figure D-8. Using the Pythagorean Theorem:

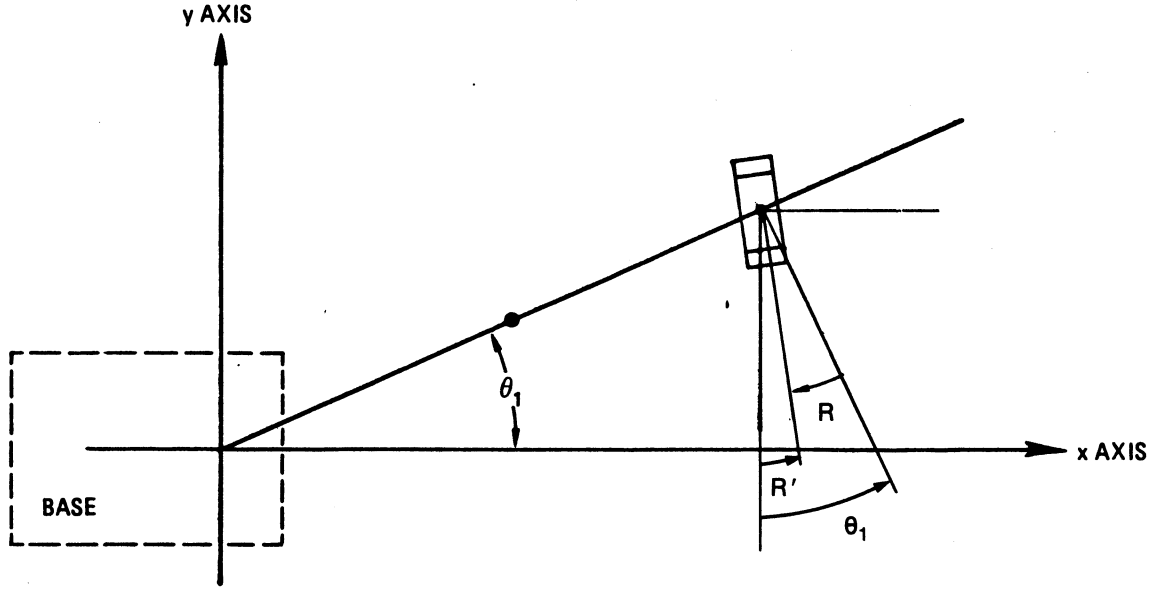


Figure D-7 Top view of arm with Pitch = 90^{o} showing roll in Cartesian frame (R') and roll with respect to the arm (R).

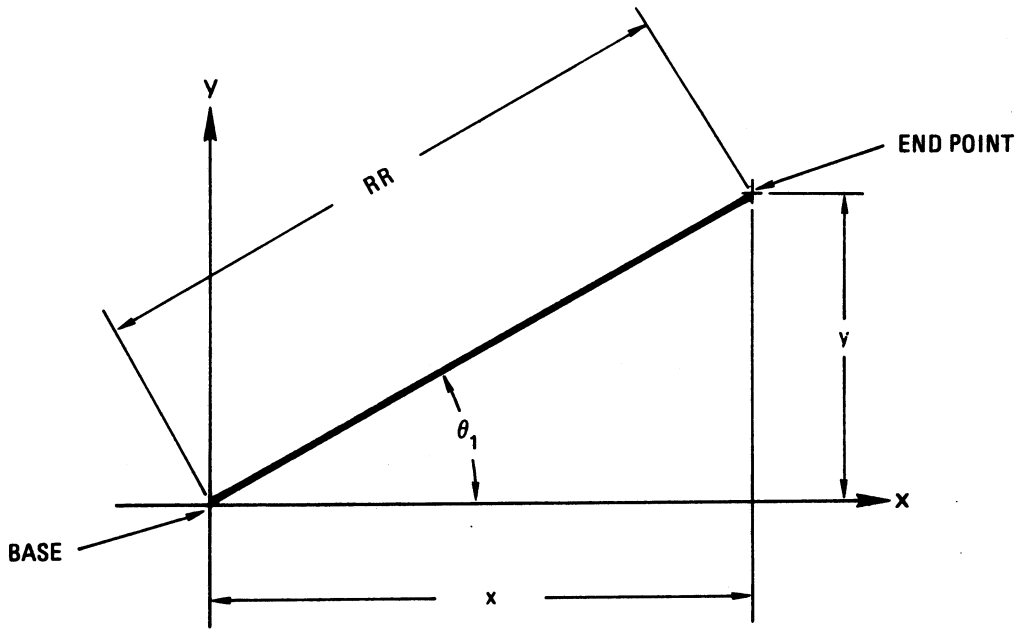


Figure D-8 Top View of Arm

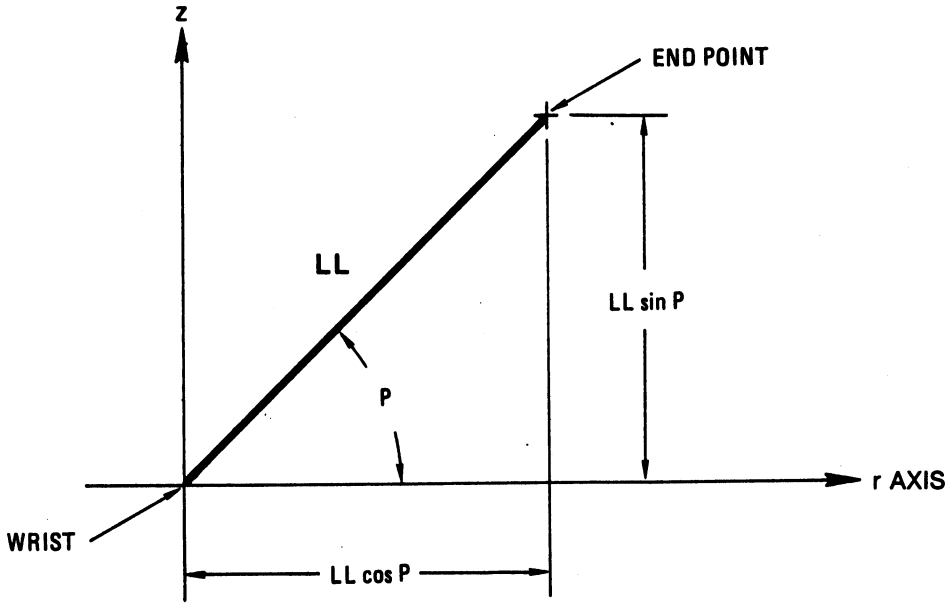


Figure D-9 Side View of Hand Triangle
in Kinematic Model

$$RR = \sqrt{X^2 + Y^2} \quad (11)$$

$$\theta_1 = \tan^{-1}(Y/X). \quad (12)$$

The second step is to find θ_4 and θ_5 from P and R. Using Equation (1) and Equation (2) of the wrist differential previously described, and substituting (R' + θ_1R1) for R using Equation (10) gives:

$$\theta_5 = P + R' + \theta_1 R1 \quad (13)$$

$$\theta_4 = P - R' - \theta_1 R1. \quad (14)$$

[Note: From here on, we will drop the prime and use R for roll in all cases, remembering to set R1 = 0 when roll is measured with respect to the arm, and R1 = 1 when roll is measured with respect to the Cartesian frame.]

The third step is to work back from the coordinates of the end point to those of the wrist. As in the forward solution, we use the side view of the kinematic model shown in Figure D-4. Distances in this view are measured vertically along the Z axis and horizontally along the radius from the base (r axis). Letting R_e and Z_e be the coordinates of the end point in this plane, we can calculate the coordinates of the wrist (R_w and Z_w) by using the triangle shown in Figure D-9. From this triangle the coordinates of the wrist are:

$$R_w = R_e - LL \cos P \quad (15)$$

$$Z_w = Z_e - LL \sin P \quad (16)$$

The fourth step is to define the shoulder-elbow-wrist triangle so that θ_2 and θ_3 can be determined. For this purpose, the translated coordinate system introduced in Figure D-10 is used. The origin (0, 0) is at the shoulder and the coordinates of the wrist are now (R_0, Z_0).

The distance from the shoulder to the wrist, R_0, is the same as R_w previously determined in Equation (15).

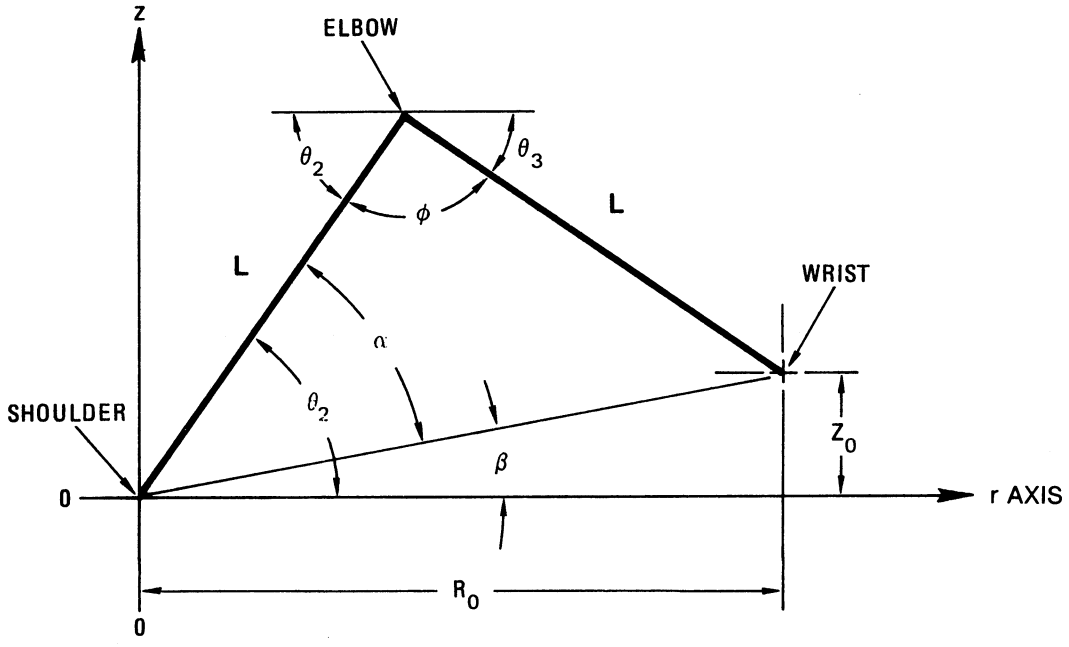


Figure D-10 Shoulder-Elbow-Wrist Triangle

This is expressed as:

$$R_0 = R_e - LL \cos P \qquad (17)$$

The height of the wrist above the shoulder, Z_0, is just the height of the wrist above the table top, Z_w, less the height of the shoulder, H. Thus,

$$Z_0 = Z_w - H \qquad (18)$$

Substituting for Z_w using Equation (16) gives

$$Z_0 = Z_e - LL \sin P - H \qquad (19)$$

The fifth step is to solve the shoulder-elbow-wrist triangle for θ_2 and θ_3. Three new angles, α, β, and ϕ, are introduced to simplify this solution. We first solve for α, β, and ϕ.

Since $\tan\beta = (Z_0/R_0)$, we obtain:

$$\beta = \tan^{-1}(Z_0/R_0). \qquad (20)$$

Pivoting the shoulder-elbow-wrist triangle about the shoulder by β gives the simplified triangle shown in Figure D-11. The length of the base of the simplified triangle is given by $\sqrt{Z_0^2 + R_0^2}$ (Phythagorean Theorem, using the right triangle at the bottom of Figure D-10). As shown in Figure D-11, the simplified triangle can be partitioned into two congruent right triangles. The base, b, of each of these smaller triangles is then given by:

$$b = .5\sqrt{Z_0^2 + R_0^2} \quad . \qquad (21)$$

The height, h, (again using the Pythagorean Theorem) is

$$h = \sqrt{L^2 - b^2} \quad . \qquad (22)$$

Since the tangent of α is h/b,

$$\alpha = \tan^{-1}(h/b) \quad . \qquad (23)$$

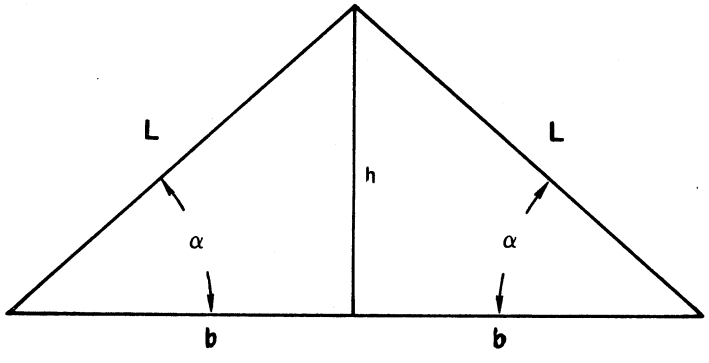


Figure D-11 Simplified Triangle

Substituting for h in Equation (23) by using Equation (22) gives

$$\alpha = \tan^{-1} \frac{\sqrt{L^2 - b^2}}{b} \quad . \qquad (24)$$

Substituting for b in Equation (24) using Equation (21) gives

$$\alpha = \tan^{-1} \sqrt{\frac{4\ L^2}{R_0^2 + Z_0^2} - 1} \quad . \qquad (25)$$

The sixth step is to use α and β to determine θ_2 and θ_3. The following three relations are first set up and then solved. At the shoulder (see Figure D-10),

$$\theta_2 = \alpha + \beta \quad . \qquad (26)$$

At the elbow apex (see Figure D-11),

$$\theta_2 + \phi + \theta_3 = 180^\circ \quad . \qquad (27)$$

Summing the internal angles of the simplified triangle (Figure D-11) gives $\phi + \alpha + \alpha = 180$, or

$$\phi = 180^\circ - 2\alpha \quad . \qquad (28)$$

Substituting the value of θ_2 from Equation (26) and the value of ϕ from Equation (28) into Equation (27) gives

$$\theta_3 = \alpha - \beta \, . \qquad (29)$$

Note however, that the elbow angle, θ_3, is defined as the angle above the horizontal and hence we must change the sign of θ_3.

In summary, the results of the sixth step are:

$$\theta_2 = \alpha + \beta \quad . \qquad (30)$$

$$\theta_3 = \beta - \alpha \quad . \qquad (31)$$

thus completing the backward solution. A summary of the backward solution is given in Table D-4.

Table D-4

Summary of Backward Solution

Step	Operation
1	Determine arm constants H, L, LL
2	Determine the desired X, Y, Z, R, P, and R1 coordinates of the endpoint
3	$\theta_1 = \tan^{-1}(Y/X)$
4	$RR = \sqrt{X^2 + Y^2}$
5	$\theta_5 = P + R + R1\ \theta_1$
6	$\theta_4 = P - R - R1\ \theta_1$
7	$R_0 = RR - LL \cos P$
8	$Z_0 = Z - LL \sin P - H$
9	$\beta = \tan^{-1}(Z_0/R_0)$
10	$\alpha = \tan^{-1} \sqrt{4L^2/(R_0^2 + Z_0^2) - 1}$
11	$\theta_2 = \alpha + \beta$
12	$\theta_3 = \beta - \alpha$

Backward Arm Solution

A BASIC implementation of the backward solution is given in Figure D-12 (Statements 130 to 370). Note that several tests have been included in the backward solution to determine if the requested position can be reached by the arm. Such tests or "software limits" are advisable in application programs to keep the arm from hitting its end stops and thereby losing calibration. You may wish to add other limits for your own applications. For example, when operating on a table top, requests for movement with Z < 0 (beneath the table top) should be refused. If obstacles in the working area can be described mathematically, collisions with them can also be avoided through the use of software limits.

```
500  REM ************************
501  REM *                      *
502  REM *         TCM          *
503  REM *      COORDINATE      *
504  REM *      CONVERSION      *
505  REM *                      *
506  REM ************************
999  REM                                   CONVERSION CONSTANTS
1000  REM
1001  REM  DEFINE CONSTANTS
1002  REM
1100 H = 7.625: REM   SHOULDER HEIGHT ABOVE TABLE
1110 L = 7.00: REM  SHOULDER-TO-ELBOW AND ELBOW-TO-WRIST LENGTH
1120 LL = 3.8: REM   WRIST TO FINGERTIP LENGTH (GRIPPER CLOSED)
1130 PI = 3.14159
1140 C = 180 / PI: REM  DEGREES TO 1.0 RADIAN
1190  HOME : VTAB 5
1200 PRINT"ENTER X, Y, Z, PITCH, ROLL, AND R1. "
1210  REM   R1 = 1 IF ROLL IS WRT CARTESIAN FRAME
1220  REM  R1 = 0 IF ROLL IS WRT ARM FRAME
1230  VTAB 7: INPUT X,Y,Z,P,R,R1
1499  REM
1500  REM  CONVERT TO RADIANS
1501  REM
1510 P = P / C:R = R / C
1530  GOTO 5000
1999  REM     BACKWARD SOLUTION          JOINT LIMIT TESTS
5000  REM
5010  REM  BACKWARD SOLUTION CALCULATIONS
5020  REM
5030  IF X = 0 THEN T1 =  SGN (Y) * PI / 2
5040  IF X <  > 0 THEN T1 =  ATN (Y / X)
5050  IF T1 < 0 THEN PRINT:PRINT"BASE OUT OF RANGE. T1= ";T1
5060 RR =  SQR (X * X + Y * Y)
5070  IF RR < 2.25 AND Z < 15 THEN PRINT:PRINT"HAND TOO CLOSE TO BODY. RR = ";RR
5080  IF RR > 17.8 THEN PRINT:PRINT"REACH OUT OF RANGE.  RR = ";RR
5090 R0 = RR - LL *  COS (P)
5100  IF X < 2.25 AND Z < 1.25 AND R0 < 3.5 THEN  IF P <  - 90 / C
      THEN PRINT:PRINT"HAND INTERFERENCE WITH BASE."
5110  REM  NOTE THAT THE ABOVE STATEMENT MAY BE ALTERED TO ACCOMODATE
      MOVES CLOSE TO THE BASE
5120 Z0 = Z - LL *  SIN (P) - H
5130  IF R0 = 0 THEN B = ( SGN (Z0)) * PI / 2
5140  IF R0 <  > 0 THEN B =  ATN (Z0 / R0)
```

Figure D-12 Basic Implementation of
Forward and Backward Solutions

```
5150 A = R0 * R0 + Z0 * Z0
5160 A = 4 * L * L / A - 1
5170  IF A < 0 THEN PRINT:PRINT"REACH OUT OF RANGE FOR SHOULDER
     AND ELBOW.": GOTO 5500
5180 A =  ATN ( SQR (A))
5190 T2 = A + B
5200 T3 = B - A
5210  IF T2 > 144 / C OR T2 <  - 35 / C THEN PRINT:PRINT"SHOULDER OUT
     OF RANGE.  T2 = ";T2 * C
5220  IF T2 - T3 < 0 OR T2 - T3 > 149 / C THEN PRINT:PRINT"ELBOW OUT
     OF RANGE.  T3 = ";T3 * C
5230  IF (R > 270 / C OR R <  - 270 / C) THEN  IF (P > ((90 / C + T3) - (R + 270 / C))
     OR P < (( - 90 / C + T3) + (R - 270 / C))) THEN PRINT:PRINT"PITCH OUT
     OF RANGE.  PITCH= ";P * C
5240  IF P > (90 / C + T3) OR P < ( - 90 / C + T3) THEN PRINT:PRINT"PITCH OUT
     OF RANGE.  PITCH = ";P * C
5250  IF (R > (360 / C -  ABS (P - T3)) OR R < ( - 360 / C +  ABS (P - T3)))
     THEN PRINT:PRINT"ROLL OUT OF RANGE.  ROLL = ";R * C
5260 T4 = P - R - R1 * T1
5270 T5 = P + R + R1 * T1
5280 PRINT:PRINT
5290 PRINT"BACKWARD SOLUTION RESULTS:  "
5300 PRINT:PRINT
5310 PRINT"T1 = ";T1 * C
5320 PRINT"T2 = ";T2 * C
5330 PRINT"T3 = ";T3 * C
5340 PRINT"T4 = ";T4 * C
5350 PRINT"T5 = ";T5 * C
5360  REM                                                    FORWARD SOLUTION
5370 PRINT: PRINT
5380 P1 = (T5 + T4) / 2
5390 R2 = (T5 - T4) / 2-R1*T1
5400 RR1 = L *  COS (T2) + L *  COS (T3) + LL *  COS (P1)
5410 X1 = RR1 *  COS (T1)
5420 Y1 = RR1 *  SIN (T1)
5430 Z1 = H + L *  SIN (T2) + L *  SIN (T3) + LL *  SIN (P1)
5440 PRINT"FORWARD SOLUTION RESULTS:  "
5450 PRINT:PRINT"X = ";X1
5460 PRINT"Y = ";Y1
5470 PRINT"Z = ";Z1
5480 PRINT"PITCH = ";P1 * C
5490 PRINT"ROLL = ";R2 * C
5500 PRINT: PRINT
5510 INPUT "TYPE A KEY WHEN READY TO CONTINUE  ";A$: GOTO 1190
```

Fig. D-12 (Cont'd.)

4. VARIATION OF HAND LENGTH WITH HAND OPENING

The opening of the hand is proportional to the number of steps of the hand drive motor. The constant of proportionality is:

$$S_6 = 371 \text{ steps/inch } (14.6 \text{ steps/mm}).$$

Although the length of the hand, LL, has been treated as a constant in the previous calculations, it varies slightly with hand opening, as shown in Figure D-13. The effect is small, ± 0.10in (± 2.5 mm), but for more precise work it may be necessary to take this into account.

The hand length, LL, may be expressed as the sum of a fixed length, L_1, and a varying length that depends on hand opening, G, by the following formula:

$$LL = L_1 + \sqrt{L_2^2 - \frac{(G - G_0)^2}{2}} \qquad (32)$$

where:

$L_1 = 1.884$ in (47.9 mm)
$L_2 = 1.700$ in (43.2 mm)
$G_0 = 1.520$ in (38.6 mm)

The hand opening, G, may be converted to motor steps and vice-versa by using the proportionality constant, S_6, given above.

Varying hand length may be taken into consideration in both the forward and backward solutions. Before starting either solution, the correct value of LL would be computer from the hand opening using Equation (32).

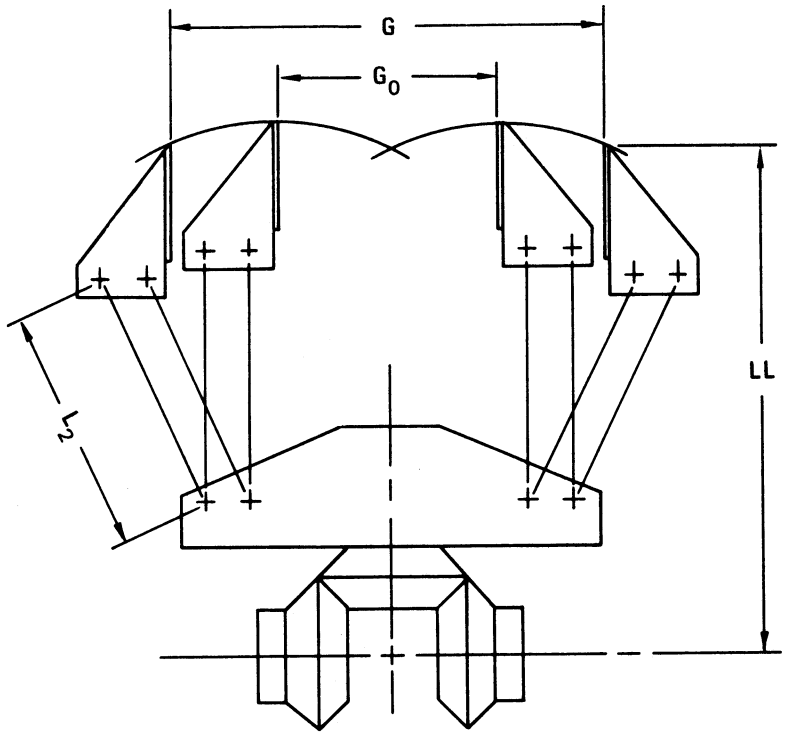


Figure D-13 Variation of Hand Length with Hand Opening

APPENDIX F
TABLES AND GRAPHS

Contents

APPENDIX F
TABLES AND GRAPHS

ITEM F-1

TEACHMOVER PERFORMANCE CHARACTERISTICS

GENERAL

Configuration	5 revolution axes and integral hand
Drive	Electric stepper motors- Open loop control
Controller	6502A microprocessor with 4K bytes of EPROM and 1K bytes of RAM located in base of unit
Interface	Dual RS-232C asynchronous serial communications interfaces (baud rates is switch-selectable between 110, 150, 300, 600, 1200, 2400, 4800, and 9600 baud)
Teach Control	14 key-13 function keyboard, 5 output and 7 input bits under computer control
Power Requirement	12 to 14 volts, 4.5 amps DC

PERFORMANCE

Resolution	0.011 in (0.25 mm) maximum on each axis
Load Capacity	16 oz (445 gm) at full extension
Gripping Force	3 lbs (13 Newtons) maximum
Reach	17.5 in (444 mm)
Velocity	0-7 in/sec (0-178 mm/sec) with controlled acceleration

DETAILED PERFORMANCE

Motion	Range	Speed (Full Load)	Speed (No Load)
Base	$\pm$90 deg	0.37 rad/sec	0.42 rad/sec
Shoulder	+144, -35 deg	0.15 rad/sec	0.36 rad/sec
Elbow	+0, -149 deg	0.23 rad/sec	0.82 rad/sec
Wrist Roll	$\pm$360° deg	1.31 rad/sec	2.02 rad/sec
Wrist Pitch	$\pm$90 deg	1.31 rad/sec	2.02 rad/sec
Hand	0-3 in (0-75mm)	8 lb/s* (35 n/sec)	(20 mm/sec)

PHYSICAL CHARACTERISTICS

Arm Weight	8 lbs (4 kg)
Teach Control Cable Length	3.75 ft. (1150 mm)

---

* This is given in lbs/sec rather than in./sec, because as the gripper closes, it no longer moves, but instead builds up gripping force. It takes 0.37 sec to build up the maximum force of 3 lbs.

APPENDIX F
TABLES AND GRAPHS

Item F-2

TEACHMOVER COMMAND SUMMARY

(Note: These tables are for reference and review. To learn how to use these commands for the first time, please refer to the appropriate sections of the manual.)

A. HAND HELD TEACH CONTROL COMMANDS

COMMAND	FUNCTION	SYNTAX/DETAILS OF OPERATION
CLEAR	Erases entire program	To activate, hold down MODE key, then press CLEAR
FREE	Turns off all motor currents	Allows for manual positioning of arm. Does not create a program step.
GRIP	Closes gripper	Builds up 1 lb. of gripping force. Moves hand motor 32 half-steps past the point where grip switch goes on.
JUMP	Conditional (or unconditional) branching	Two numerical entries (press MODE key in between): 1st entry-jump condition: 0: grip switch open 1, 2, ..., 7: user input bit 1, 2, ..., 7 is on 8: Never 9: Always 2nd entry - step number to jump to if jump condition is met.
MOVE	Activates joint-control (arm-motion) keys	Same as TRAIN mode, but does not change internal position registers, does not allow a position to be recorded, and does not create a program step.
MODE	Stops Arm	Used to exit TRAIN, MOVE, and ENTER modes.

APPENDIX F
TABLES AND GRAPHS

OUT	Designates which output bit to turn on	Two numerical entries (press MODE key in between) 1st entry - output number: 0: MODE light 1, 2, ..., 5: user output 1, 2, ..., 5 6: TRAIN light 7: RUN light 8: ENTER light 2nd entry - 0 or 1 (For indicator lights: 0 = off 1 = on)
PAUSE	Pauses arm for specified number of seconds	Numerical entry: 0 to 255
POINT	Sets program pointer to designated step	Numerical entry: step number to go to.
RUN	Runs current program	Or if running, stops at end of current step.
SPEED	Sets speed of subsequent arm motion	Numerical entry: 0 (slowest) to 15 (fastest). See items 4 & 5 of this appendix for stepping rates and maximum speeds.
STEP	Executes current program, one step at a time	Moves arm to next position.
TRAIN	Activates joint-control (arm-motion) keys	Press REC for each position to be saved. REC overwrites current step and increments sequence pointer. When power is turned on, unit is in TRAIN mode, with sequence pointer and internal position registers set to zero.
ZERO	Zeros the sequence pointer and the internal position registers	To activate, hold down MODE key, then press CLEAR.

APPENDIX F
TABLES AND GRAPHS

SERIAL INTERFACE COMMANDS		
Notes:	1. <CR> = Carriage Return 2. Arm returns [0<CR>] if command has a syntax error, [1<CR>] after command is executed (except for @ RUN), [2<CR>] if STOP button was pressed before execution was completed (@STEP and @CLOSE only)	
COMMAND	FUNCTION	SYNTAX/DETAILS OF OPERATION
@ARM	Specifies recognition character to use instead of "@" sign	@ARM <CHAR> <CR> where CHAR is any character except a carriage return
@CLOSE	Close gripper until grip switch is activated	@CLOSE <SP> <CR> where SP = optional speed value (see item 4 in this appendix)
@DELAY	Inserts a delay between transmitted characters	@DELAY <CR> Where N = proper delay value, determined by trial and error.
@QDUMP	Uploads entire current program from TeachMover to host computer	@QDUMP <CR> Returns character string comprising eight two-byte values for each program step. See Table 9 in chapter 7 for details.
@QWRITE	Downloads a program step from host computer to TeachMover	@QWRITE<N>,<L1>,<L2>, ...<L7><CR> where N = Step number to which program step is to be written. L1-L7 = two-byte values as in @QDUMP command. See chapter 7 for details.
@READ	Reads values of of the internal internal position registers, gives last key pressed on teach control, and tells which input bits are on.	@READ <CR> Arm returns: <K1>, <K2>, ..., <K6>, <I><CR> where K1-K6 = values of internal position registers I = Last key *256 + Input Byte where "Last key" values are defined below:

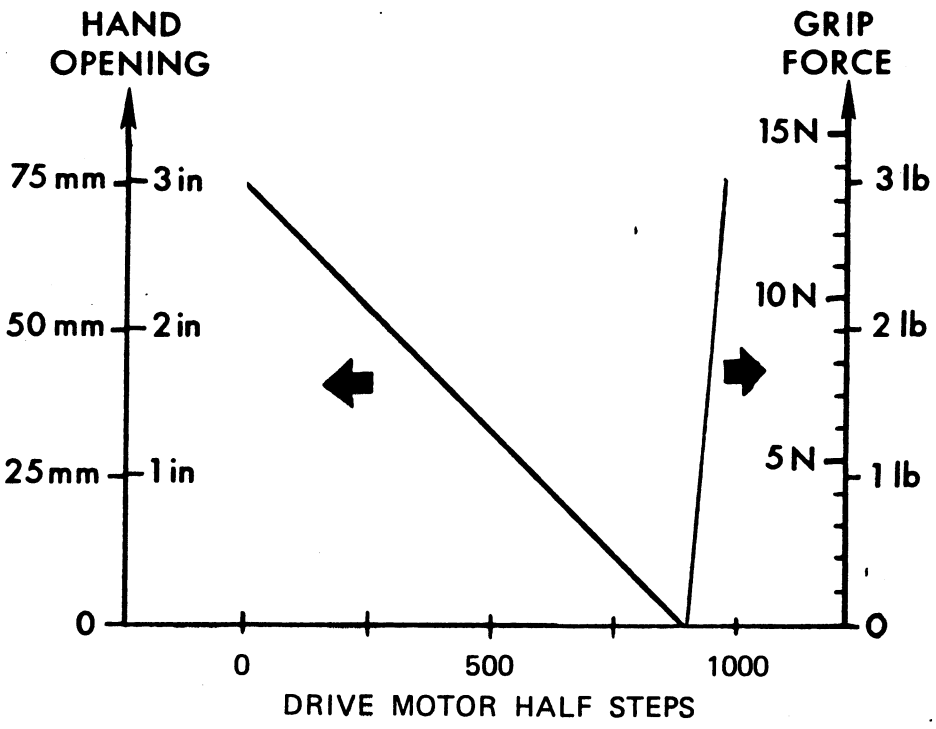
@READ (cont.)		"Last key" Value / Key Pressed 1 TRAIN 2 PAUSE 3 GRIP 4 OUT 5 FREE 6 MOVE 7 MODE 8 STEP 9 POINT 10 JUMP 11 CLEAR 12 ZERO 13 SPEED 14 REC and: "Input Byte" = decimal number whose binary equivalent specifies which of the eight input bits are set to 1 (see "jump condition" numbers under hand-held teach control JUMP command, above).
@RESET	Zeros the internal position registers and turns off motor currents	@RESET<CR>
@SET	Sets subsequent arm speed and activates joint control keys on hand-held teach control	@SET<SP><CR> where SP = optional speed value (see item 4 in this appendix). Control returns to host when REC or MODE key is pressed
@STEP	Sets arm speed, moves joints, sets output bits	@STEP<SP>,<J1>,<J2>,...,<J6>, <OUT><CR> where SP = speed value (see item 4 in this appendix)

@STEP (cont.)	J1-J6 = Number of motor half-steps J1 = Base swivel (positive counter-clockwise) J2 = Shoulder (positive downwards) J3 = Elbow (positive downwards) J4 = Right wrist (positive downwards) J5 = Left wrist (positive downwards) J6 = Hand (positive open) OUT = Optional decimal number whose binary equivalent specifies the value of the output bits (see Appendix F, item G).

APPENDIX F
TABLES AND GRAPHS

Item F-3

HAND OPENING AND GRIPPING FORCE DIAGRAM



APPENDIX F
TABLES AND GRAPHS

Item F-4

STEPPING RATES FOR THE SPEED COMMANDS

Teach Control Speed Number	Serial Port Speed Value	Half-Steps Per Second
0	0	28
1	111	50
2	159	74
3	183	99
4	205	141
5	221	206
6	232	300
7	236	360
8	238	400
9	239	424
10	240	450
11	241	480
12	242	514
13	243	554
14	244	600
15	245	655

APPENDIX F
TABLES AND GRAPHS

Item F-5

MAXIMUM NO-SLIP MOTOR SPEEDS

Load	Teach Control Speed No.	Serial Port Speed Value	Half-Steps Per Second
0	8	238	400
Half (8 oz.)	5	221	206
Rated (16 oz.)	3	183	99

APPENDIX F
TABLES AND GRAPHS

Item F-6

OUTPUT NUMBERS AND JUMP CONDITIONS

Output Number	Function Controlled
0	MODE light
1	User output 1
2	User output 2
3	User output 3
4	User output 4
5	User output 5
6	TRAIN light
7	RUN light
8	ENTER light

Jump Condition	Jump If
0	Grip switch is open
1	User input 1 is on
2	User input 2 is on
3	User input 3 is on
4	User input 4 is on
5	User input 5 is on
6	User input 6 is on
7	User input 7 is on
8	Never
9	Always

Item F-7

BAUD RATE SELECTION

Baud	SW1	SW2	SW3
110	ON	ON	ON
150	OFF	ON	ON
300	ON	OFF	ON
600	OFF	OFF	ON
1200	ON	ON	OFF
2400	OFF	ON	OFF
4800	ON	OFF	OFF
9600	OFF	OFF	OFF

PROGRAM_____

DATE_____

PROGRAMMER_____

PROGRAMMING WORKSHEET FOR MICROBOT TEACHMOVER

APPENDIX G

STEP #	OPERATION	STEP #	OPERATION
0		27	
1		28	
2		29	
3		30	
4		31	
5		32	
6		33	
7		34	
8		35	
9		36	
10		37	
11		38	
12		39	
13		40	
14		41	
15		42	
16		43	
17		44	
18		45	
19		46	
20		47	
21		48	
22		49	
23		50	
24		51	
25		52	
26			