Laboratory 1, Robots - 4U4

You will be required to write programs for a computer to send instructions over a serial interface to a six-axis robot. The robot will be required to search for an unknown point in space at the end of its arm's reach. At the end of the search, your program must indicate that it has found the point, display and then move to its coordinates. Time taken from beginning the search, until the hand has moved to the unknown point must be displayed as well.

For the robot to find this point, you will use a set of transducers: a photoresistor, and a set of light-emitting-diodes. One of the transducers will be placed at the coordinates of the unknown point. The light-emitting diodes can be turned on or off under computer control. The output voltage of the photoresistor can be read via an analog-to-digital converter inside the computer.

Scope of this lab

This lab is not about interfacing a computer to other hardware. We will make every effort to make the interfaces as transparent as possible. Your efforts should mostly involve choosing algorithms, calculating hand position or where to move it next, and interpreting data from your transducers.

This lab *is* about choosing good algorithms. You will also learn something about servo systems; using transducer data to direct the robots' search. This is a complex system: you should learn how to break it down into manageable blocks, each with inputs and outputs. Unlike many labs, you will not be given cookie-cutter techniques to achieve your goal. You should continually evaluate your methods - be prepared to try others. This lab is complex enough that dividing the workload and communication among lab partners will be critical to your success.

Your program design should employ good engineering principles. Your search should take a minimum amount of time. Your search should be able to find any point as efficiently as any other point. Your search should be robust in the face of deteriorating conditions: stray reflections, motor cable slippage, etc.

Lab Report

The lab write-up must describe your programs well enough that your teaching assistant can understand them.

Where to begin?

You must learn the command language of the robot so that your program can direct the robot's motors to move the arm to a desired position. Appendix A includes the command set. Appendix B describes the mechanical range of the robot's joints, and physical dimensions.
At least one of the lab partners should begin writing simple programs to move all the joints a desired amount under computer command. Work up to being able to move the arm to desired three-dimensional cartesian coordinates.

The characteristics of the transducers should be examined early on. Spend some time moving one of the transducers around while collecting data on the other. Be prepared to try a few different algorithms: your first approach will likely have flaws.

Appendix C shows in detail how the photoresistor can be read by the computer's analog-to-digital converter and how the LEDs can be turned on and off.

The BASIC programming language is adequate when dealing with both the robot's serial interface, and the analog/digital interface. QBASIC is available on the hard drive. However, you may use any computer language.

Robot Basics

The robot's motors as well as the microcontroller under the robot arm base need a twelve volt D.C. power supply. A hand-held controller, capable of moving all the joints in both directions is also used to enter a programmed sequence of arm movements into the robot's microcontroller. The on-board microcontroller may not have enough programming steps to be useful in this lab.

Configuration	5 revolution axes and integral hand
Drive	Electric stepper motors - Open loop control
Controller	6502A microprocessor with 4 K bytes EPROM and 1 K bytes RAM
Interface	Dual RS-232 asynchronous serial communications interface. Baud rate is switch-selectable between 110, 150, 300, 600, 1200, 2400, 4800, 9600 baud.
Teach Control	14-key 13-function keyboard for real-time motion; 5 output and 7 input bits under computer control.
Power	12 to 14 Volts, 4.5 amps D.C.

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. (14 Newtons) maximum
Reach	17.5 in (444 mm) partial sphere with center about shoulder joint
Velocity	0 - 7 in/s (0 - 178 mm/s) with controlled acceleration

Joint	Max range of motion	Speed (full load)	Speed (no load)
Base	+- 90 degrees	0.37 rad/s	0.42 rad/s
Shoulder	+144, -35 degrees	0.15 rad/s	0.36 rad/s
Elbow	+0, -149 degrees	0.23 rad/s	0.82 rad/s
Wrist Roll	+-270 degrees	1.31 rad/s	2.02 rad/s
Wrist Pitch	+- 90 degrees	1.31 rad/s	2.02 rad/s
Hand	0 - 3 inches	8 lb/sc (35 N/s)	(25 mm/s)
Arm weight	8 lbs. (4 kg)		

All six stepper drive motors are located in the body beneath the shoulder joint. The motors drive their corresponding arm members through gears and a cable/pulley system. Although the stepper motors are capable of very accurate and repeatable motion, gear backlash and slack cable tension will not allow repeatability to be as good as resolution. Cable slip may mean that the arm is not where you (or your computer program) think it is. This is what is known as an open-loop system. A closed loop system corrects for progressive errors in positioning.

Stepper motors require a complex sequence of current switches to rotate: the robot's microcontroller takes care of the sequencing and timing of all six motors. Motor rotation requests by the controlling computer through the serial port require specifying speed and number of half-steps.

Choosing a speed too high will cause the steppers to balk, resulting in less motor rotation than requested. No errors are reported when mis-stepping occurs - a consequence of an open loop controller. Maximum speed is dependent on motor load...heavy objects must be lifted slowly to avoid balking.

The hand has two fingers, which can be closed on an object. A micro-switch sensing cable tension lets the microcontroller know when the fingers have closed either against each other, or upon an object. Requesting more hand closure will increase the finger's grip, up to 14 Newtons - see Appendix A.

Transducer/computer Interface

An eight-channel 12-bit analog-to-digital (AD) converter and a four-channel 12 bit digital-to-analog (DA) converter are located on a card plugged into the computers' expansion bus. Data is read and written to the card using input/output instructions. Connections are made via a 25-pin connector on the computer's rear panel. The same card also has twenty four lines of digital inputs or outputs. A few of these bits are used to turn the light emitting diodes on or off. The robot also has some digital inputs and outputs, interfaced to the computer through the serial interface. Appendix A shows how to read and write these bits.

An optical sensors/transmitter will be made available along with their interface to the AD/DA converter card. The transducer pairs provided will be capable of detecting each other over the limits of the arm's reach, although if the transmitter's radiation is not directed in the appropriate direction, the receiver may not detect a signal. When placed next to each other, the transducers will not overload, although the analog-to-digital converter might. Under program control, the analog input can be scaled over an 8-1 range:

Smallest detectable signal is +/- 0.152 mV.....scale factor of 8
Largest signal before overload is +/- 5V.....scale factor of 1
Allowable scale factors are 1,2,4,8

Your transducer may generate more noise than the smallest detectable signal - you may need to use noise reduction techniques.

The AD and DA converters are very fast. Maximum speed of data collection will be determined more by the program than by the hardware.

Appendix A - Computer to Robot Interface

Communication between computer and robot is carried over a 9600 baud bi-directional serial link. Standard ASCII characters are accepted by the microcontroller inside the robot. After each command is completed, the robot's microcontroller sends a simple ASCII number code back to the computer in acknowledgment. Your computer program should read this acknowledgment for every command given, or the serial link may indicate to the robot's microcontroller that a serial-stream overflow condition exists, causing communication to cease.

You may choose to use any computer language you wish. Your language should be capable of setting up the computer's serial port to 9600 Baud, eight bit, no parity, with one stop bit. Your language must be able to:

-send a string of ascii characters out the serial port
-receive a string of ascii characters from the serial port with a termination <CR>
-convert integers to an ASCII character string
-parse and convert ASCII characters into an integer.

The following routines are probably useful, but not necessary:

-Test the serial port to see if characters have been received -or- set a flag when characters have been received via an interrupt service routine.

QBASIC (installed in the lab's computer) has built-in commands to perform all these functions.

Command Set

All ten commands begin with the ASCII "@" symbol. You may abbreviate the command to the first three characters -- @CLO for @CLOSE, etc. Some commands are followed by numeric parameters. All numerical values are integers composed of a string of decimal ASCII characters.

◆1 @STEP <sp>,<J1>,<J2>,<J3>,<J4>,<J5>,<J6>,<OUT>,<CR>

The STEP command causes all six stepper motors to move, simultaneously. <sp> is a numeric parameter that sets the speed of motion, <J1> to <J6> are numerical parameters indicating how many half-steps each stepper motor is to rotate, <OUT> is a numerical parameter that sets the bit pattern of the digital user outputs, and <CR> is the ASCII symbol for carriage return.

After successfully interpreting the command, and after the motors have finished their rotation, a response of <1><CR> is sent back to the computer over the serial link. A syntax error will respond with <0><CR>, with no motor rotation. If the STOP key on the hand controller is pressed before motion is complete, the response is <2><CR> instead. A STOP keypress will result in the motors having partially completing their required rotation, leaving the arm in an unknown position.

<sp>, ranging from 0 to 245, sets the maximum rotation rate. Motor speed in half-steps per second is given by the formula:

$$\text{Motor speed} = \frac{1843200}{256 \mid sp - 255 \mid}$$

Maximum no-slip motor speed is 400 steps per second, corresponding to a <sp> value of 238.
A load of 8 oz. reduces the no-slip speed to 206 steps/sec., for a <sp> value of 221.
Maximum load of 16 oz. reduces the no-slip speed to 99 steps/sec., for a <sp> value of 183.

<J1>...<J6> indicate the number of half-steps each motor will be driven. The sign of each number indicates direction:

Table A-1 @STEP joint parameters and direction of motion

parameter	Joint	positive	negative
J1	BASE swivel	counter-clockwise	clockwise
J2	SHOULDER	bend downward	bend upward
J3	ELBOW	bend downward	bend upward
J4	right WRIST	bend downward	bend upward
J5	left WRIST	bend downward	bend upward
J6	HAND	open	close

Unlike operation with the hand-held teach controller, 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>. If the 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>

<OUT> is a positive decimal integer (0 to 511) that gets translated into a binary number that specifies which of the nine user output bits are to be set to "one" once the joint motions are completed. For example, the parameter 129 would specify that user outputs 0 and 7 (which turn on the hand-controller's MODE and RUN lights) should be set to logic 1, and all the rest set to logic 0.

Table A-2 User programmable outputs

OUTPUT bit	Function Controlled
2^0	MODE light
2^1	User output 1
2^2	User output 2
2^3	User output 3
2^4	User output 4
2^5	User output 5
2^6	TRAIN light
2^7	RUN light
2^8	ENTER light

Not all parameters need be included in the STEP command string. For example, if only the base is to be swiveled 50 half-steps, with all other joints left unperturbed, the command

@STEP 200,50 is equivalent to the command @STEP 200,50,0,0,0,0,0

If the <OUT> parameter is excluded from the parameter list, OUTput bits remain unchanged.

◆2 @CLOSE <sp><CR>

This command is analogous to the GRIP button on the hand-held controller, except that instead of closing the hand 32 half-steps past where the grip switch closes, it closes the hand *just* to where the grip switch closes. Objects may not be gripped tightly enough to move: an @STEP command could follow to increase gripping force:

```
@CLOSE 235
@STEP 180,0,0,0,0,0,200
```

Maximum gripping force once the fingerpads close on an object is about 14 Newtons, achieved with about 200 half-steps. Gripping force is roughly proportional to the number of half-steps, but is quite dependent on object shape.

Once again a serial port response of <1><CR> is sent from the robot's microcontroller to the computer after the CLOSE command completes. A <0><CR> indicates a syntax error, and <2><CR> indicates that the hand-controller's STOP key was pressed mid-execution.

◆3 @SET <sp><CR>

This command activates the keys on the hand-held teach control, putting the unit into TRAIN mode. This means that you can program arm positions directly from the hand-held teach controller 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 the hand-held controller keys than by programming joint motor steps via the @STEP 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.

◆4 @RESET <CR>

This command is similar to the hand-controller's ZERO command, in that it sets the contents of the microcontroller's six internal position registers to zero. The RESET command also turns off the current to all six stepper motors, allowing you to manipulate the arm manually via the large plastic gears at the back of the base. You may use the RESET command to initialize the arm as well.

Once again, a serial port response of <1><CR>, or <0><CR> is sent from the robot's microcontroller to the computer.

◆5 @READ <CR>

This command has no equivalent on the hand-held controller. The robot microcontroller's position registers are sent over the serial link to the controlling computer. The robot's microcontroller responds through the serial port with <1><CR>, or <0><CR> as with all commands, followed by a string of ASCII numbers corresponding to the STEP command parameters:

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

where <K1> to K6> are the values of the microcontroller's internal position registers. The last number, I is a decimal integer that can be decoded to yield two things:

- the logic values of the eight input bits
- the last key that was pressed on the hand-held controller.

I = 256(Last key) + Input byte

where "Last key" is zero unless a key has been pressed on the hand-held controller since the last READ command; otherwise "Last key" is a numeral from 1 to 14:

Table A-3 User programmable inputs

Last key	Key action	Key programming action
1	Base counterclockwise	Train
2	Shoulder down	Pause
3	Elbow down	Grip
4	Pitch down	Out
5	Roll counterclockwise	Free
6	Hand grip (close)	Move
7	Mode	Stop
8	Base clockwise	Step
9	Shoulder up	Point
10	Elbow up	Jump
11	Pitch up	Clear
12	Roll clockwise	Zero
13	Hand grip (open)	Speed
14	Record	Run

"Input byte" is a decimal integer that, when translated to binary, indicates which of the eight input bits are set (logic 1). The least significant bit shows the status of the gripper switch, and bits 1 to 7 are user inputs that can be set by external switches, the host computer, etc.

Example: If I is 1800:
"Last key" must be 7, or the STOP key
"Input byte " must be 8 (because 1800 = 7 * 256 + 8)
Decimal 8 = binary 00001000, which means user input 3 is on, and all other bits are off.

As mentioned earlier, a response of <2><CR> indicates that the STOP key was pressed during the execution of a STEP or CLOSE command. In this state, the robot's internal microcontroller will have position values different from those expected by the remainder of the program, likely resulting in incorrect motions thereafter. Upon receipt of a <2><CR> response, your program should READ the robot's position values, so that it can make up the difference with a STEP command.

◆6 @ARM<char><CR>

This command changes the pre-command "arming" character (default @ symbol). One computer could then command multiple daisy-chained robots, each having a unique arming character. If another device is daisy-chained onto the serial port, the robot's microcontroller passes all characters not matching its arming character to the next serial device. All commands (and all parameters belonging to those commands) having the matching arming character are not passed to the next serial device. Once again a response of <1><CR> or <0><CR> is sent from robot to computer.

◆7 @DELAY <N><CR>

This command affects the data rate on the serial port for data transmitted from robot to computer. Some computers will experience serial port data overflow, especially at high baud rates. Characters are lost by the computer when it cannot process them fast enough. The symptom of this problem is having less than seven valid parameters after a READ command.

A lower baud rate would cure the problem at the expense of slow data transmission from computer to robot. Since more data is sent from computer to robot than the other way, a lower baud rate is undesirable.

The DELAY command puts a time delay between characters sent from robot to computer of approximately 0.5 millisecond times N. N may take on values including 0 to 255.
Once again a response of <1><CR> or <0><CR> is sent from robot to computer.

◆8 @QDUMP <CR>

This command uploads a microcontroller program from the TeachMover to host computer, allowing you to generate programs on the hand-held controller and then save them on a disk for later use.

After acknowledge of <1><CR>, the arm sends a long character string which represents the entire set of 53 program steps (or 126 steps if expanded memory is installed). The data is coded into a terse format having no simple relation to command codes. Each program 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 is the step number (0-52 or 0-125). The other seven values are not described in detail here, since the codes can be re-loaded into the TeachMover with the following QWRITE command. The eighth value is followed by a <CR>.

◆9 @QWRITE <L1>,<L2>,<L3>,<L4>,<L5>,<L6>,<L7>,<L8><CR>

<L1> corresponds to the step number 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. A QWRITE command with no syntax errors results in an acknowledgment response from robot to computer of <1><CR>. Unlike the QDUMP command, you must use QWRITE for each program step, rather than just once for the entire program.

◆10 @RUN <N><CR>

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

N is a decimal integer representing the program step at which you wish to begin the program 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 with no syntax errors, the arm responds with <1><CR> before beginning to execute the TeachMover program.

Program Example in BASIC

This program moves only the hand, demonstrating how QBASIC may be used to communicate over the serial port, command the robot, and read data from the robot. Thickness of an object placed between the arm's fingers is measured in inches.

```
5  REM Initialize the serial port, ignoring CLEAR-TO-SEND,DATA-SET-
READY,CARRIER-DETECT
10 OPEN "COM2:9600,N,8,1,CS,DS,CD"AS #1
30 REM close the hand, so that the fingers are just touching:
40 PRINT #1,"@CLOSE 238":INPUT#1,ST:JAW = 0
50 REM Reset arm's internal registers, making this the zero-set
calibration position:
60 PRINT #1,"@RESET":INPUT#1,ST
70 REM Now open the hand, to allow object to be placed between fingers:
80 PRINT #1,"@STEP 238",0,0,0,0,0,500-JAW:INPUT#1,ST
90 INPUT "Hit <RET> to measure object",A$
100 REM now close the fingers on the object:
110 PRINT #1,"@CLOSE 238":INPUT#1,ST
120 REM now read the robot's internal position registers:
130 PRINT #1,"@READ":INPUT#1,ST:INPUT#1,T,U,V,W,X,JAW,Y
140 PRINT "Object is ";JAW/371;" INCHES thick"
150 PRINT #1,"@STEP 238",0,0,0,0,0,90:INPUT#1,ST:REM release object
160 END
```

Appendix B - Coordinate and Joint Definitions

It is often advantageous to describe the configuration of a robot arm in more than one coordinate system:

Joint Coordinates (the joint angles of the arm). These are most convenient for controlling the arm directly from the computer, because joint angles are proportional to the expressions used in computer commands controlling those joints.

Cartesian Coordinates (X, Y, Z, pitch, and roll). These are more convenient for describing an assembly task above a flat table top. It is advisable to use cartesian coordinates in this lab.

Workspace Coordinates. Often, the object upon which the robot arm must perform some task is rotated or moved to allow access by the robot's hand. In this case, the workstation's coordinate system must be conjoined with the robot's.

Moving Coordinates. This coordinate system might be located on, and aligned with a continually moving platform such as a conveyor belt or a turntable.

Each of the five joints on the robot arm (base, shoulder, elbow, right wrist, left wrist) is rotated by a stepper motor via gears and/or cables. The joint expressions J1, J2, J3, J4 and J5 sent from the computer to robot are directly proportional to the angular rotation of each of the five joints.

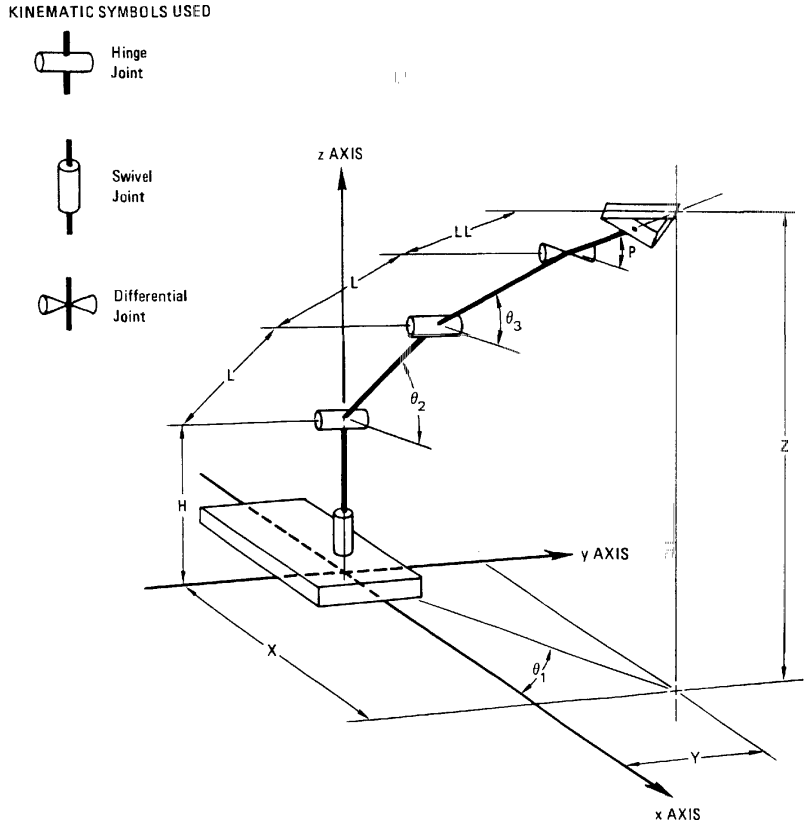


Figure B-1 Geometry of joints of the Robot Arm

Table B-1 Conversion Factor between Motor Steps and Revolute Joint Angles

Motor	Joint	Steps in one rev.	Steps/radian	Steps/degree
1	Base	7072	1125	19.64
2	Shoulder	7072	1125	19.64
3	Elbow	4158	661.2	11.55
4	Right Wrist	1536	244.4	4.27
5	Left Wrist	1536	244.4	4.27

Distances between joints (length of arm members) are indicated by the constants H, L, and LL shown in figure B-1 .

Table B-2 Length of Robot Arm Members

Segment	Description	Length(in.)	Length(mm.)
H	table top to shoulder joint centerline	7.68	195.0
L	shoulder joint to elbow joint	7.00	177.8
L	elbow joint to wrist joint	7.00	177.8
LL	wrist joint to center point of fingerpads with fingers separated by 1.5 inches	3.8	96.5

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

$$P = 0.5(\theta_5 + \theta_4) \quad (1)$$

$$R = 0.5(\theta_5 - \theta_4) \quad (2)$$

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 plane. It is difficult to distinguish between positive and negative roll angles (as +90 vs. -90, or +45 vs.-135) by looking at the hand. It may be helpful to mark the top of the hand when it is at zero to eliminate this ambiguity. The zero position corresponds to the hand orientation when the wrist cable turnbuckles (hidden in the member between elbow and wrist) are aligned with each other.

Knowing the angles of the joints (with respect to the horizontal plane), calculating the X, Y, Z cartesian coordinates is easy:

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

Horizontal distance from coordinate (0, 0, 0) to a point directly underneath the fingertips is:

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

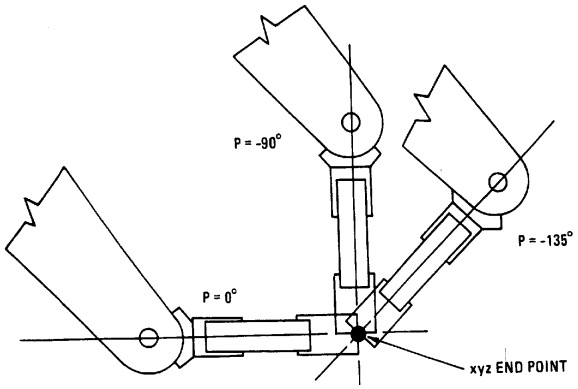
from which X and Y are:

$$X = RR \cos \theta_1 \quad (5)$$

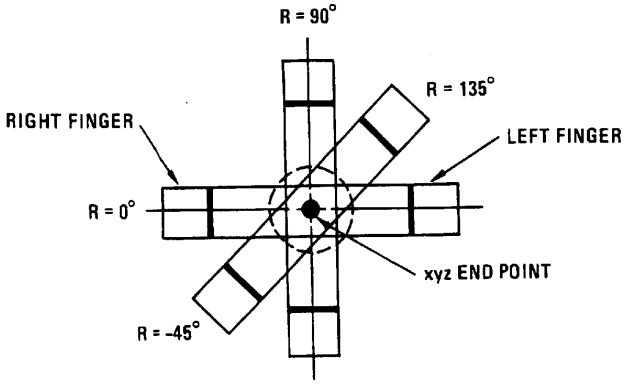
$$Y = RR \sin \theta_1 \quad (6)$$

Backward Arm Solutions

This section shows how to determine the joint angles θ_1, θ_2, θ_3, θ_4, and θ_5 (and thus the stepper motor joint parameters J1, J2, J3, J4, and J5) required to position the end point at a desired X, Y, and Z position, with a desired pitch and roll.



(a) Different Pitch Angles at Same Endpoint



(b) Different Roll Angles at Same Endpoint
View looking into front of hand along pitch vector

Figure B-2 Different Hand Orientations

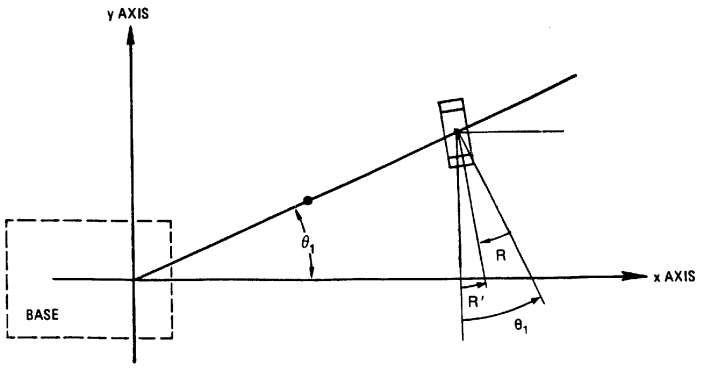


Figure B-3 Defining the frame of reference for Roll

Sometimes it is desirable to express "roll" with respect to a cartesian frame rather than with respect to the arm. For example, you may need to pick up an object with a know X, Y, Z and angular orientation. One way to do this is to use P = -90 degrees (hand pointing down) as a reference orientation, and then measure the "Cartesian roll" with respect to the x-axis, as shown in figure B-3. Roll measured with respect to the arm (R) and roll measured with respect to the cartesian frame (R') is related by:

$$R' = R - \theta_1 \qquad (7)$$

In the backward solution, we introduce a special variable, R1, 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:

R1 = 1 if roll is with respect to Cartesian frame
R1 = 0 if roll is with respect to arm frame

With this new variable, Equation (7) can be modified to express both normal and Cartesian roll as follows:

$$R' = R - \theta_1 R1 \qquad (8)$$

and

$$R = R' + \theta_1 R1 \qquad (9)$$

Next, determine the base angle θ_1, and the radius vector RR, from the base to the end point:

$$RR = \text{sqrt}(X^2 + Y^2) \qquad (10)$$

$$\theta_1 = \tan^{-1}(Y/X) \qquad (11)$$

Next, find θ_4 and θ_5 from P and R. Using Equation (1) and (2) of the wrist differential, and substituting $(R' + \theta_1 R1)$ for R using Equation (9) gives:

$$\theta_5 = P + R' + \theta_1 R1 \quad (12)$$

$$\theta_4 = P - R' - \theta_1 R1 \quad (13)$$

[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 frame, and set R1 = 1 when roll is measured with respect to the Cartesian frame.]

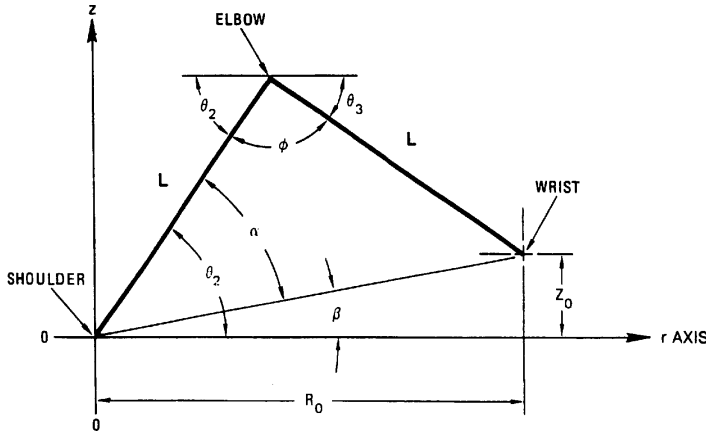


Figure B-4 Partial robot arm geometry - from shoulder to wrist

The wrist, elbow and shoulder joints all lie in the same plane, defined vertically by the Z axis and horizontally along the vector RR. Letting Re and Ze be the coordinates of the end point in this plane, we can calculate the coordinates of the wrist (Rw and Zw):

$$Rw = Re - LL \cos P \quad (14)$$

$$Zw = Ze - LL \sin P \quad (15)$$

Now let us 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 B-4 is used. The origin (0, 0) is at the shoulder and the coordinates of the wrist are now (Ro, Zo). The distance from the shoulder to the wrist, Ro, is the same as Rw previously determined in Equation (14):

$$Ro = Re - LL \cos P \quad (16)$$

The height of the wrist above the shoulder, Zo, is just the height of the wrist above the table-top, Zw, less the height of the shoulder, H. Thus,

$$Zo = Zw - H \quad (17)$$

Substituting for Zw using Equation (15) gives

$$Zo = Ze - LL \sin P - H \quad (18)$$

The last two joint angles θ_2 and θ_3 involve the shoulder-elbow-wrist triangle, shown in figure B-4. Three new angles, α, β, and ϕ, are introduced to simplify the solution of angles θ_2 and θ_3. Let us now find expressions for α, β, and ϕ .

Since tan β = (Zo/Ro), we obtain:

$$\beta = \tan^{-1} (Zo/Ro). \quad (19)$$

Pivoting the shoulder-elbow-wrist triangle about the shoulder joint by the angle β lets the triangle base lie along the horizontal plane. The length of the base of the simplified triangle is given by sqrt($Zo^2 + Ro^2$). This simplified triangle can be partitioned into two, similar, congruent right triangles. The base b, of each of these smaller triangles is then given by:

$$b = .5 \text{ sqrt } (Zo^2 + Ro^2) \quad (20)$$

The height, h, of each of these same two triangles is

$$h = \text{sqrt } (L^2 - b^2) \quad (21)$$

Since the tangent of α is h/b,

$$\alpha = \tan^{-1} (h/b) \quad (22)$$

Substituting for h in Equation (22) by using Equation (21) gives

$$\alpha = \tan^{-1} (\text{sqrt}(L^2 - b^2)/ b) \quad (23)$$

Substituting for b in Equation (23) using Equation (20) gives

$$\alpha = \tan^{-1} \text{sqrt}(4 L^2/(Ro^2 + Zo^2) -1) \quad (24)$$

Now we can 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 B-4),

$$\theta_2 = \alpha + \beta \quad (25)$$

At the elbow apex,

$$\theta_2 + \phi + \theta_3 = 180 \quad (26)$$

Summing the internal angles of the simplified triangle gives $\phi + \alpha + \alpha = 180$, or

$$\phi = 180 - 2\alpha \quad (27)$$

Substituting the value of θ_2 from Equation (25) and the value of ϕ from Equation (27) into Equation (26) gives

$$\theta_3 = \alpha - \beta \ . \quad (28)$$

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

A summary of the backward solution is given in Table B-3 . A sample backward solution QBASIC program follows.

Table B-3 Summary of Backward Solution of θ_1, θ_2, θ_3, θ_4, θ_5

Determine arm length constants H, L, LL.
Determine the desired **X**, **Y**, **Z**, **R**, **P**, and **R1** coordinates of the endpoint.

$$\theta_1 = \tan^{-1}(Y/X)$$
$$RR = \text{sqrt}(X^2 + Y^2)$$
$$\theta_5 = P + R + R1\theta_1$$
$$\theta_4 = P - R - R1\theta_1$$
$$Ro = RR - LL \cos P$$
$$Zo = Z - LL \sin P - H$$
$$\beta = \tan^{-1}(Zo/Ro)$$
$$\alpha = \tan^{-1} \text{sqrt}(4L^2/(Ro^2 + Zo^2) - 1)$$
$$\theta_2 = \alpha + \beta$$
$$\theta_3 = \beta - \alpha$$

```
REM *********************************
REM *                               *
REM *          Teachmover           *
REM *  Cartesian Coordinate Control *
REM *           Program             *
REM *                               *
REM *********************************

CLS
OPEN "COM2:9600,N,8,1,CS,DS,CD" FOR RANDOM AS #1
REM INPUT #1, ST

REM Define the Robot Arm Constants

H = 7.625: REM Shoulder Height above Table
L = 7: REM Shoulder to Elbow and Elbow to Wrist Length
LL = 3.8: REM Wrist to Fingertip Length

REM Define other constants

PI = 3.14159265#
C = 57.2957795#: REM Degrees in 1.00 radian
R1 = 1: REM Flag for world coordinates

REM Define robot arm scale factors

S1 = 1125: REM steps/radian, joints 1 & 2
S2 = -S1
S3 = -661.2: REM steps/radian, joint 3
S4 = -244.4
S5 = S4: REM steps/radian, joints 4 & 5
S6 = 371: REM steps/inch, hand

REM Initialization

DIM UU(7, 40): REM Room for 40 steps
```

```
P1 = 0
P2 = -508
P3 = 1162
P4 = 384
P5 = P4
P6 = 0

REM The six lines above are the number of joint steps from
REM T1=0, T2=0, T3=0, T4=0, T5=0, and J=0, To X=5, Y=0, Z=0,
REM P=-90, R=0, and J=0

REM Read in the first line for initialization

READ X, Y, Z, P, R, GP, S

CLS

PRINT "Set arm to the following position and orientation"
PRINT "using keyboard."
PRINT
PRINT "     X="; X; " inches"
PRINT "     Y="; Y; " inches"
PRINT "     Z="; Z; " inches"
PRINT " PITCH="; P; " degrees"
PRINT "  ROLL="; R; " degrees"
PRINT "  HAND="; GP / S6; "inches"

PRINT #1, "@STEP 200": INPUT #1, ST: REM Move Arm
PRINT #1, "@RESET": INPUT #1, ST
U = 0

170 :

PRINT
INPUT "Enter the new X position: "; X
INPUT "Enter the new Y position: "; Y
INPUT "Enter the new Z position: "; Z
INPUT "Enter the new P position: "; P
INPUT "Enter the new R position: "; R
INPUT "Enter the new J position: "; GP
GP = GP * S6
INPUT "Enter the new SP position: "; S


REM READ X, Y, Z, P, R, GP, S
REM INPUT "Hit return to go on:"; A$

CLS

REM Display coordinates
PRINT "Robot arm is moving to the following"
PRINT "coordinates:"
PRINT
```

```
PRINT "     X = "; X; " inches"
PRINT "     Y = "; Y; " inches"
PRINT "     Z = "; Z; " inches"
PRINT " Pitch = "; P; " degrees"
PRINT "  Roll = "; R; " degrees"
PRINT "  Hand = "; GP / S6; " inches"

REM
REM
REM Routine to convert cartesian coordinates
REM to number of joints steps away from the start position
REM
REM

REM Backward solution calculations

P = P / C
R = R / C
IF X = 0 THEN T1 = SGN(Y) * PI / 2
IF X <> 0 THEN T1 = ATN(Y / X)
IF T1 < -90 THEN PRINT : PRINT "Base out of range. T1= "; T1: GOTO 170
IF T1 > 90 THEN PRINT : PRINT "Base out of range. T1 = "; T1: GOTO 170
RR = SQR(X * X + Y * Y)
IF RR < 4 AND Z < 15 THEN PRINT : PRINT "Hand too close to body. RR = "; RR: GOTO 170
IF RR > 17.8 THEN PRINT : PRINT "Reach out of range. RR = "; RR: GOTO 170
RO = RR - LL * COS(P)
IF X < 2.25 AND Z < 1.25 AND RO < 3.5 THEN IF P < -90 / C THEN PRINT : PRINT "Hand
interference with base.": GOTO 170
REM Note that the above statement may be altered to accommodate movement close to the base
ZO = Z - LL * SIN(P) - H
IF RO = 0 THEN B = (SGN(ZO)) * PI / 2
IF RO <> 0 THEN B = ATN(ZO / RO)
A = RO * RO + ZO * ZO
A = 4 * L * L / A - 1
IF A < 0 THEN PRINT : PRINT "Reach out of range for shoulder and elbow.": GOTO 170

A = ATN(SQR(A))
T2 = A + B
T3 = B - A
IF T2 > 144 / C OR T2 < -35 / C THEN PRINT : PRINT "Shoulder out of range. T2 = "; T2 * C:
GOTO 170
IF (T2 - T3) < 0 OR (T2 - T3) > (149 / C) THEN PRINT : PRINT "Elbow out of range. T3 = ";
T3 * C: GOTO 170
IF (R > (180 / C) OR R < (-180 / C)) THEN IF (P > ((90 / C + T3) - (R + 180 / C)) OR P <
((-90 / C + T3) + (R - 180 / C))) THEN PRINT : PRINT "Pitch out of range. Pitch = "; P *
C: GOTO 170
IF P > (90 / C + T3) OR P < (-90 / C + T3) THEN PRINT : PRINT "Pitch out of range. Pitch =
"; P * C: GOTO 170
IF (R > (270 / C - ABS(P - T3)) OR R < (-270 / C + ABS(P - T3))) THEN PRINT : PRINT "Roll
out of range. Roll = "; R * C: GOTO 170
IF GP > (3 * S6) THEN PRINT : PRINT "Hand cannot open that wide": GOTO 170

T4 = P - R - R1 * T1
T5 = P + R + R1 * T1
```

```
REM Correct Coordinates

W1 = INT(S1 * T1 + .5) - P1
W2 = INT(S2 * T2 + .5) - P2
W3 = INT(S3 * T3 + .5) - P3
W4 = INT(S4 * T4 + .5) - P4
W5 = INT(S5 * T5 + .5) - P5


PRINT #1, "@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): INPUT #1, ST

TMP = INT(GP - UU(6, U))
U = U + 1

UU(1, U) = W1
UU(2, U) = W2
UU(3, U) = W3
UU(4, U) = W4
UU(5, U) = W5
UU(6, U) = GP
UU(7, U) = S

IF GP <= 0 THEN GOTO 300

REM Open hand if JAW > 0

PRINT #1, "@STEP 240", 0, 0, 0, 0, 0, TMP: INPUT #1, ST
GOTO 170

REM Close hand and squeeze if JAW < 0

300 :
PRINT #1, "@CLOSE 245": INPUT #1, ST
GOTO 170

DATA 5,0,0,-90,0,0,225
```

Appendix C - 12-bit Analog/Digital Converter Program Interface

A card in the PC's expansion slot has a 12-bit analog-to-digital converter, four 12-bit digital-to-analog converters, and twenty-four bits of digital input/output. The analog inputs and outputs are available on a 25-pin D-connector at the PC's rear panel. Since we are dealing with a card on an eight-bit I/O bus, twelve bits of data must be broken into two parts - eight least significant bits, and the most significant four bits (high nibble).

Coding of analog-to-digital and digital-to-analog conversions is OFFSET BINARY, where zero volts is represented by the most-significant bit being set, while all others are zero: 800h or 2048d. With the analog-to-digital gain set to one, the most positive voltage of +5v is represented by all twelve bits being set: 0FFFh or 4095d. The most negative voltage of -5v is represented by all twelve bits being zero.

I/O Port		Port READ	Port WRITE
768	bit 7	when low, the A/D is busy doing a conversion	---
768	bit 6	---	---
768	bit 5	---	---
768	bit 4	along with bit 3: read gain	with bit 3: sets gain,
768	bit 3	00=gain 1, 01=gain 2, 10=gain 4, 11=gain 8	least significant gain set.
768	bit 2	MSB of multiplexer address	with bit 1, bit 0, set multiplexer
768	bit 1	000=channel 0, 111= channel 7	address for analog-to-digital
768	bit 0	LSB of multiplexer address	converter.
770	bit 7	analog-to-digital data, 8th significant bit	---
770	bit 6	analog-to-digital data, 7th significant bit	---
770	bit 5	analog-to-digital data, 6th significant bit	---
770	bit 4	analog-to-digital data, 5th significant bit	---
770	bit 3	analog-to-digital data, 4th significant bit	---
770	bit 2	analog-to-digital data, 3th significant bit	---
770	bit 1	analog-to-digital data, 2nd significant bit	---
770	bit 0	analog-to-digital data, least significant bit	---
771	bit 7	---	---
771	bit 6	---	---
771	bit 5	---	---
771	bit 4	---	---
771	bit 3	analog-to-digital data, 12th significant bit	---
771	bit 2	analog-to-digital data, 11th significant bit	---
771	bit 1	analog-to-digital data, 10th significant bit	---
771	bit 0	analog-to-digital data, 9th significant bit	---
772		8255 Parallel Interface chip	8255 Parallel Interface chip
773		8255 Parallel Interface chip	8255 Parallel Interface chip
774		8255 Parallel Interface chip	8255 Parallel Interface chip
775		8255 Parallel Interface chip	8255 Parallel Interface chip
776		transfer 12-bit data to DAC0	High nibble DAC0 holding register
777		transfer 12-bit data to DAC1	High nibble DAC1 holding register
778		transfer 12-bit data to DAC2	High nibble DAC2 holding register
779		transfer 12-bit data to DAC3	High nibble DAC3 holding register
780		---	Low byte DAC0 holding register
781		---	Low byte DAC1 holding register
782		---	Low byte DAC2 holding register
783		---	Low byte DAC3 holding register

Rather than have eight analog-to-digital converters, an eight-way switch allows one analog-to-digital converter to read eight different voltages. Both the switch and the analog to digital converter's scaling are set with the same I/O port. Channels 0 through 7 can be selected by writing to an output port:

```
REM Set the gain to 1
REM Always access channel 0 when getting input
GA = 0
CH = 0
OUT 768, CH + 8 * GA
```

will connect the analog-to-digital converter to the first of eight inputs. GA has been set to the lowest gain (0=gain of 1, 1=gain of 2, 2=gain of 4, 3=gain of 8). Port address of 768 is the base address to which the analog/digital card has been set. The voltage can now be read with:

```
AD = INP(770)
REM Wait until A/D is finished doing a conversion
WHILE INP(768) < 128
WEND
REM Input the data from 771(MSB) and 770(LSB) ports
AD = 256 * INP(771)
AD = AD + INP(770)
```

The first conversion here is required to clear old data in the analog to digital converter. Port 770 always contains the eight least significant bits of the previous conversion. *Reading* from port 770 also begins an analog to digital conversion cycle. After the first instruction, AD will hold eight bits of old data: while the instruction is executing, the converter will be acquiring new data. A status byte at port 768 shows when the AD conversion has completed. A WHILE statement loops until the most significant bit of status (the **not busy** flag) goes high. The next line will read the most significant four bits of the new conversion from port 771, and the eight least significant bits from port 770.

Currently, a photoresistor circuit is wired to analog input channel 0. No light will give zero volts, with more light causing voltage to increase positively This circuit is nonlinear: output voltage is not directly proportional to incident light intensity.

Digital-to-Analog Conversions

The twelve-bit data to be output as an analog voltage (between -5.0v to +5.0v) must first be written to one of four holding registers. Once again, the holding register is broken into two parts: the least significant byte and most significant four bits (high nibble).

Once data is written to the appropriate I/O ports, the digital-to-analog converter is activated by READING from port 776, 777, 778 or 779. It is important to note that writing new data to the holding registers will not result in the analog output voltage changing to the new value. The voltage output from the four digital-to-analog converters will remain fixed until new data is loaded into the holding register(s) followed by a READ from ports 776, 777, 778, or 779.

All four outputs are buffered by unity-gain op-amps, which can supply about +/- 10ma. of current before the output voltage is affected.

```
DA0= 2048+409.4*V1:REM V1 is voltage (between -5 to +5) to be output
OUT 780,DA0
OUT 776,(DA0/256)
A=INP(776)
```

Parallel Port input/output

The three eight-bit parallel ports at addresses 772, 773, 774 are wired to an Intel 8255 parallel interface chip. This chip is configurable for a wide variety of applications - this description will be limited to inputting and outputting logic signals.

On power-up, all three ports default to being input ports, simply reading any of these ports will input logic data to your program. Changing these ports to output data involves writing a "mode" byte to port address 775:

Port 775 Mode definition

Control Byte	Port A(772)	Port B(773)	Port C(774) High nibble	Port C(774) Low nibble
128	Out	Out	Out	Out
129	Out	Out	Out	In
130	Out	In	Out	Out
131	Out	In	Out	In
136	Out	Out	In	Out
137	Out	Out	In	In
138	Out	In	In	Out
139	Out	In	In	In
144	In	Out	Out	Out
145	In	Out	Out	In
146	In	In	Out	Out
147	In	In	Out	In
152	In	Out	In	Out
153	In	Out	In	In
154	In	In	In	Out
155	In	In	In	In

Currently, four light-emitting diodes are switched on and off with control signals from Port C(774), high nibble. A logic 1 turns the LED on, a logic 0 turns the LED off. Port C must be set to output data in order to control the LEDs. Here is a QBASIC code segment that writes the mode, and then turns all the LEDs off:

```
OUT 775,147: REM set the mode to output data on the four MSB of port C
OUT 774,0: REM turn off all four LEDs
```

Code segment to turn on each LED in sequence:

```
OUT 774,128: REM LED 1 only
OUT 774,64: REM LED 2 only
OUT 774,32: REM LED 3 only
OUT 774,16: REM LED 4 only
```

Code segment to turn on all four LEDs:

```
OUT 774, (128+64+32+16)
```

Here is a sample program that initializes the Analog-to-digital converter and parallel port chip, and then prints continuously the data values input from the AD converter. This program should work with no external connections to the converter card (the AD card *is* required to be installed in one of the expansion slots). *Useful* data will be printed when the photoresistor is powered (and connected to the DB25 plug on the back of the card), and the four-LED array is connected to the 8-pin header labeled "PORT C".

```
REM Program: ADTEST.BAS
REM Set the mode to output data on the four MSB of port C
OUT 775, 147
REM Turn on all four LEDs
OUT 774, 240
REM Set the gain to 1
REM Always access channel 0 when getting input
GA = 0:CH = 0
OUT 768, CH + 8 * GA
REM Reading port 770 begins an analog to digital cycle
AD = INP(770)
bck:
REM Wait until A/D is finished doing a conversion
WHILE INP(768) < 128
WEND
REM Input the data from 771(MSB) and 770(LSB) ports
AD = 256 * INP(771)
AD = AD + INP(770)
PRINT AD
GOTO bck
```