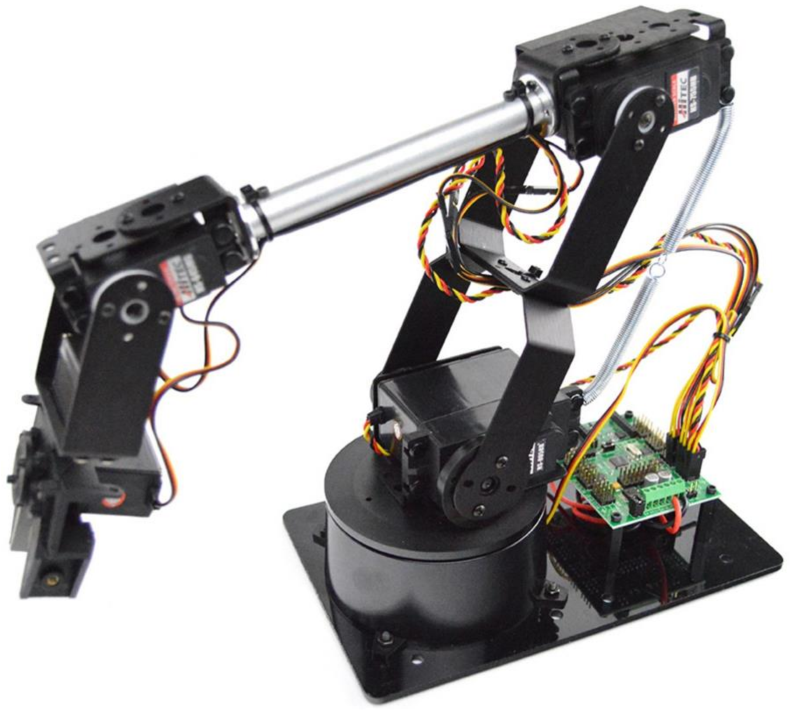
Lab 1 – Kinematics and the DH Convention

March 8, 2020

ME 439



Author:

Dean Rogers
Department of Mechanical Engineering
University of North Dakota
Grand Forks, North Dakota 58202
e-mail: dean.rogers@und.edu

Instructor:

Jeremiah Neubert
Department of Mechanical Engineering / Associate Professor
University of North Dakota
Grand Forks, North Dakota 58202
e-mail: jeremiah.neubert@und.edu

Executive Summary

The Lynxmotion AL5D 5 degree of freedom robot was able to complete Hanoi Tower puzzles of heights up to 7 disks successfully using inverse kinematics and controlled with MATLAB. There were a couple of challenges including jerky movements, inaccurate positioning due to changes in torque on the servos, and servo resolution. Although these issues were not completely fixed, they were controlled sufficiently to complete Hanoi Towers with 3 and 7 disks.

Table of Contents

Introduction

With the increasing availability and capability of robots combined with decreasing costs, robots are becoming commonplace in many aspects of life. Often used to automate tasks or perform dangerous tasks, the performance of these robots must be accurate, consistent, and reliable. In order to meet those criteria, robots must be designed, programmed and tested thoroughly. This lab provides an introduction to the programming and testing steps of that process by using a small-scale robotic arm to solve a Hanoi Tower puzzle. Although this specific application has no direct real-world value, the movement, orientation and positioning accuracy needed in this application adequately represent the requirements of a real-world robotics application.

Objectives

The objective of this lab is to become familiar with robots and programming them to perform certain functions. This includes modelling the robot using the DH convention and inverse kinematics, and using that to program the robot, using MATLAB, to solve a Hanoi Tower puzzle.

Theory

The first step to solving a Hanoi Tower with a robotic arm is model the robot. Two methods of modeling were used in this lab, forward kinematics using the Denavit–Hartenberg (DH) convention, and inverse kinematics using the geometric method.

Forward kinematics is used when the joint angles of the robot are known, and we want to know the location of the robot's end effector in the base reference frame. The DH convention specifies a certain method of establishing the coordinate frames for each link of the robot to make the forward kinematics process standardized. Although forward kinematics wasn't used in solving the Hanoi Tower, it is important to set up to at least organize positive joint directions and link lengths and offsets. The concept behind the DH convention is that each link's reference frame is related to the previous link's reference frame using only four parameters:

1. Rotation about z_{n-1} to align x_{n-1} with x_n
2. Translation about z_{n-1} to align x_{n-1} with x_n
3. Translation along x_{n-1} to align o_{n-1} with o_n
4. Rotation about x_{n-1} to align z_{n-1} with z_n

A DH table can then be created to summarize those four parameters for all frame relations. Then, using only the DH table, the robot's position can be modeled given the joint angles.

Inverse kinematics is a more difficult process, but generally a much more useful one, as it provides the joint angles required to obtain a certain end effector pose. Depending on the number of degrees of freedom in the robot, certain assumptions may have to be made in order to obtain a single solution. The geometric inverse kinematics method used in this lab uses geometry, as the name suggests, to determine the required joint angles. The most common equations used in geometric inverse kinematics are:

$$\sin(\theta) = \frac{o}{h} \tag{1}$$

$$\cos(\theta) = \frac{a}{h} \tag{2}$$

$$\tan(\theta) = \frac{o}{a} \tag{3}$$

$$h = \sqrt{a^2 + o^2} \tag{4}$$

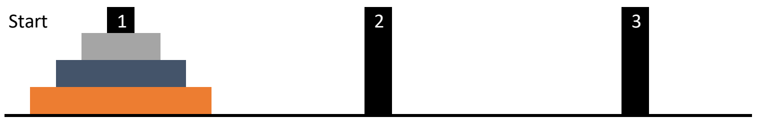
Equations 1-4 are only applicable to right triangles, where *θ* is an angle less than 90 degrees, *o* is the side opposite the angle, *h* is the hypotenuse of the triangle, and *a* is the side adjacent to the angle.

$$k = i^2 + j^2 - 2ij\cos(\theta_k) \tag{5}$$

$$\frac{i}{\sin(\theta_i)} = \frac{j}{\sin(\theta_j)} = \frac{k}{\sin(\theta_k)} \tag{6}$$

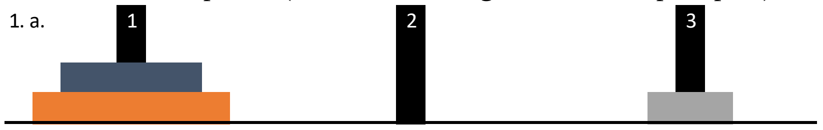
Equations 5 and 6 apply to any triangle, where *i*, *j*, and *k* are the lengths of the three sides and θ_i, θ_j, and θ_k are the angles opposite those sides respectively.

The next step is creating a program to solve the Hanoi Tower so that the program can instruct the robot where to move. The Hanoi Tower puzzle is a game where a stack of disks of different radii on one of three posts must be transferred to another post, subject to two rules: only one disk can be moved at a time, and no disk can be placed on a smaller disk. An example of a Hanoi Tower puzzle is shown in Figure 1.

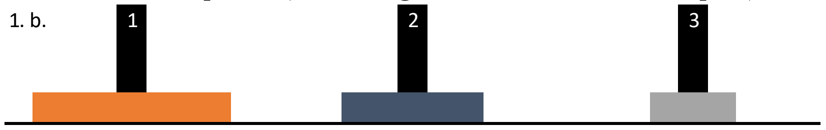


The minimum number of moves required to complete the puzzle is 2^n-1 where *n* is the number of disks, meaning a tower of 7 disks requires a minimum of 2^7-1, or 127, moves. Although this may seem like a complex problem, it can be simplified by working backwards from the desired position to the starting position. To simplify it we can look at a tower of 3 disks. We start by considering that we need to move the whole tower from post 1 to post 3. In order to do that, disks 1 and 2 must be on post 2 to allow disk 3 to move. To move disks 1 and 2 from post 1 to post 2, disk 1 must be on post 3 to allow disk 2 to move. Now, to find our moves, we go back through our analysis in reverse, which can be separated into:

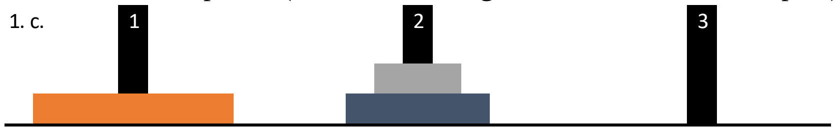
1. Move disks 1 and 2 to post 2 (Move all but largest disk to spare post)
    a. Disk 1 moves to post 3 (Move all but largest-1 disk to spare post)



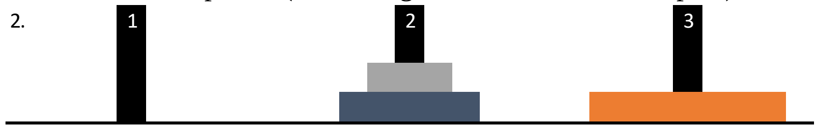
    b. Disk 2 moves to post 2 (Move largest-1 disk to destination post)



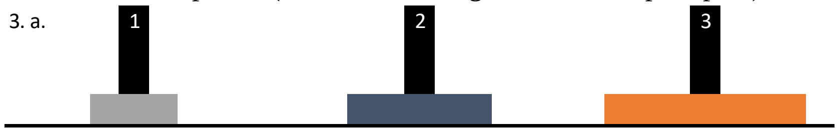
    c. Disk 1 moves to post 1 (Move all but largest-1 disk to destination post)



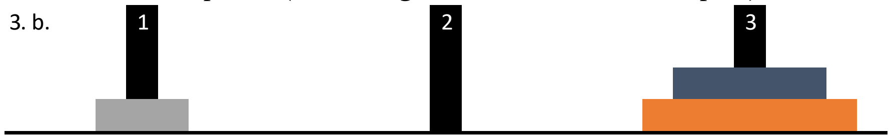
2. Disk 3 moves to post 3 (Move largest disk to destination post)



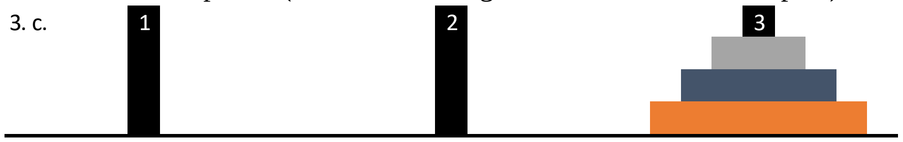
3. Move disks 1 and 2 to post 3 (Move all but largest disk to destination post)
    a. Disk 1 move to post 1 (Move all but largest-1 disk to spare post)



    b. Disk 2 moves to post 3 (Move largest-1 disk to destination post)



    c. Disk 1 moves to post 3 (Move all but largest-1 disk to destination post)



4. Done!

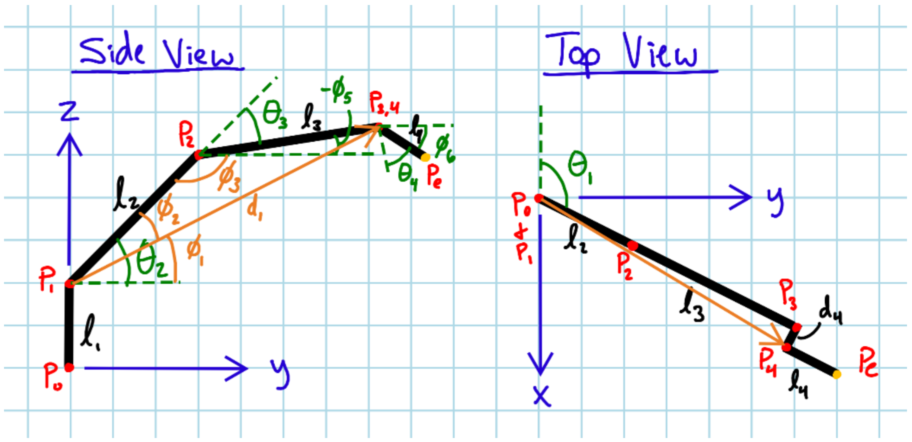
When programming the solution to a Hanoi Tower, the simplest method is to perform a recursive analysis where a function is defined that calls itself with different input parameters. The pseudocode for the recursive function is:

```
function MoveTower(diskNum, sourcePost, destPost, sparePost)
    if diskNum == 1
        move disk from sourcePost to destPost
    else
        MoveTower(diskNum - 1, sourcePost, sparePost, destPost)     % Step 1 above
        move diskNum from sourcePost to destPost                    % Step 2 above
        MoveTower(diskNum - 1, sparePost, destPost, sourcePost)     % Step 3 above
    end
end
```

where the input diskNum is the total number of disks and disk number 1 is the smallest disk.

Experimental Approach

The first step in programming the Lynxmotion AL5D robot is to complete the DH table to summarize link lengths, offsets, and joint angle directions. The full DH process can be found in the appendix of this report. The next step is to use geometry to find the joint angles given a position and orientation of the end effector. Starting with two diagrams, one of the top view of the robot and one of the side view, and separating them into different triangles, I was able to use combinations of equations 1-6 to determine the required joint angles. Initially, my plan was to have the end effector stay pointed straight down the whole time, but that only works up to a certain desired end effector height, then the 180° rotation limit of the servo makes the position impossible. To remedy that issue, I decided to keep the end effector positioned at 45° down to increase the max height that the arm can reach. This position does have issues if the arm needs to be positioned at a point close to the base of the robot since some of the mechanical parts of the robot collide before the servo reaches its limit, which just requires that the Hanoi Tower stacks be placed at least ~5 inches in front of the robot base. With those two factors in mind, I performed the inverse kinematics calculations for both scenarios, as shown in the appendix of this report, and the following equations describe the joint angles given the desired end effector x, y, and z positions as well as the end effector angle, ϕ_6. (The definitions of all the following variables can be found in the Appendix)



$$P_{4_{xy}} = \sqrt{P_{e_x}^2 + P_{e_y}^2} - l_4 \cos(\phi_6) \tag{7}$$

$$P_{4_z} = P_{e_z} + l_4 \sin(\phi_6) \tag{8}$$

$$d_1 = \sqrt{P_{4_{xy}}^2 + \left(P_{4_z} - l_1\right)^2} \tag{9}$$

$$\theta_3 = 180° - \cos^{-1}\left(\frac{l_2^2 + l_3^2 - d_1^2}{2 * l_2 * l_3}\right) \tag{10}$$

$$\theta_2 = \sin^{-1}\left(\frac{l_3 \sin(180° - \theta_3)}{d_1}\right) + \sin^{-1}\left(\frac{P_{4_z} - l_1}{d_1}\right) \tag{11}$$

$$\theta_1 = \cos^{-1}\left(\frac{P_{e_x}}{-\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right) - \cos^{-1}\left(\frac{l_2\cos(\theta_2) + l_3\cos(\theta_2 - \theta_3) + l_4\cos(\phi_6)}{\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right) \tag{12}$$

$$\theta_4 = 90° + \theta_3 - \theta_2 - \phi_6 \tag{13}$$

The next step in the process was to map the joint angles from degrees to servo ‘counts’, which is used to convert the angles obtained from the previous step into values that will be passed to the servos. For servos 1 through 5, the Pendant MATLAB app was used to specify an angle in ‘counts’, then a sliding T bevel was used to record the angle and transfer it to a piece of paper where it could be measured in degrees. Servo six, the clamp servo, was mapped from clamp opening in inches to servo rotation in counts using a ruler. Five different angles, or positions, were measured to provide data points which were then entered into an Excel spreadsheet. For servos 1 through 5, the LINEST() Excel function was used on the data from each servo to determine the slope and y-intercept of the linear trend line to map the joint angle in degrees to servo counts. For servo 6, the data points were plotted on a scatter plot and a polynomial best fit curve was added to determine the equation of the line. The actual equation to describe the relation between rotational movement and linear movement in the clamp is likely a sine function, however, that is not available as a best fit curve, so a cubic polynomial was chosen as the next best option. This file can be found in the appendix of this report. Once these values were found, they were tested on the robot by finding a point on the board attached to the base of the robot, calculating the required joint angles, converting them to counts, and then checking if the arm reached the correct point. Some small adjustments were made to the slope and intercept of servo 1 and the intercept of servo 5.

The next step was to make the pieces for the Hanoi Tower. I used a box made of 2-ply cardboard, a box made of 1-ply cardboard, and a 7/8” diameter wooden dowel. I first used a spray adhesive to bond 3 large pieces of the 2-ply cardboard together to effectively make 6-ply cardboard. Once this was dry, I used a jigsaw to cut out seven rings with outer diameters starting at 2” and going to 5”, increasing the outer diameter by ½” for each ring. The inside diameter of the rings needed to be at least 1-1/4” to provide enough wiggle room for the small piece of dowel that will be attached to the top of each disk. I increased the inside diameter with the ring size to save on weight, but that is not necessary. Next, disks of the 1-ply cardboard were cut out with diameters corresponding to the seven outer diameters of the rings. Wood glue was then used, although any Elmer’s type glue would work, to attach each 1-ply disk to the ring with the corresponding outer diameter. Seven pieces of the 7/8” dowel were then cut to 5/8” lengths, just slightly shorter than the thickness of the rings to prevent the disks above from resting on the dowel, and were glued to the center of the 1-ply disk on the opposite side from the 6-ply ring.

Since the disks used in this Hanoi Tower don’t fit over a post, the locations of each stack were measured and drawn on the board attached to the base of the robot. Using the center of the base of the robot as the origin for the x, y, and z axes, as specified in the DH table, stack 1 was set at (-5.75, 8.5), stack 2 was set at (0, 8.5), and stack 3 was set at (5.75, 8.5). The measurements were performed using a try square against the base of the robot to ensure

Writing the MATLAB program was the next step. The first section prompts the user to enter the height of the tower to allow for gradual testing as the size of the tower increases. Next, the Hanoi Tower is solved using the recursive function mentioned previously (moveTower) as well as another function which logs each move in an array (moveDisk). The program then establishes a serial connection with the robot at 115200 baud and moves the robot to the home position specified. Note that the baud rate of the robot was changed from the default to give a smoother movement and thus increase speed and accuracy. The baud rate of the SSC-32 controller on the robot must also be changed to match that baud rate, which is done by turning on the robot, pressing and holding the small white button the SSC-32 controller that says BAUD until the green and red lights next to it start alternating. Let go of the button and the green light should stay lit, which means it’s at 9600 baud. Press the button twice so that both the red and green light are on and leave it for 10 seconds to set the baud rate to 115200. To set it back to 9600, press and hold again and click the button once so that just the green light shows and wait 10 seconds. The next section of the MATLAB program loops through each step in the solution and generates a path for each move based on via points determined by the disk location and stack height. The path is then passed to a function which performs the inverse kinematics for the robot (invKin) which uses equations 7 through 13 to determine the required joint angles at each point in the path and then converts the joint angles to servo counts using the linear fit lines determined previously. The results of this function are then passed back the main script and each servo position is written to the serial in the format required to be read by the SSC-32 controller, and then the program waits for that move to finish before sending the next command. Once all moves are complete, the robot moves back to its home position. The MATLAB code is in the appendix of this report and attached as an *.m file.

Conclusion and Recommendations

Both objectives, completing towers with a height of three and seven disks, were achieved successfully. The videos can be found here: https://youtu.be/A5mwtjL2hJg for three disks and here: https://youtu.be/4ws7j_N2Bho for seven disks. One of the main challenges with getting the larger towers to work was the fact that the weight of the disks added enough torque to some of the servos to change their position slightly. This slight change was magnified due to the length of the arm and was causing the robot to place the disks up to a quarter inch off the mark. Then, when they were picked back up and set down another quarter inch off, they wouldn't stack correctly, or the robot couldn't pick them up the next time because they were too far off their mark. It seemed that the larger changes in height caused a slightly larger error, likely due to the change in the shape of the arm and thus changing the moment arm. In order to remedy this, a correction was added to the MATLAB code where a multiplier removed or added a small displacement to the tower location for that move only. With this correction, there is still some room for improvement, but the robot can now complete a tower of seven disks with no problems. Another issue that was found was, trying to use the RobotArm object provided, the movements of the arm were very jerky if anything more than endpoints were specified. Instead, I created my own serial connection which was better but blended everything together. I added a function to wait until the previous move was complete, but that caused it to be slightly jumpy, so I increased the baud rate from 9600 to 115200 to decrease the lag time between receiving the ok to send a new command, and the new command being sent. Additionally, because some of the paths generated for moving the disks was cause collisions with other disks or stacks, conditions were added to check that it was clearing the middle tower if it moved between stacks 1 and 3, and to limit the angle of the path moving between two stacks so the disk being moved didn't collide with the tower it was moving from. The final observation is that only so much precision can be acquired from a certain servo motor, and that the motors are probably one of the most important parts of a robot because the are the main determining factor for the precision, payload and speed of the robot.

As for future experiments, I would recommend using disks on posts, as just stacking them can lead to difficulty in accuracy as the stacks get higher since each inaccuracy is compounded, whereas using posts puts each disk back to a specific location after each move. Additionally, I would be interested in performing the same experiment but including the use of machine vision to determine the number of disks in the initial stack, as well as helping with actual disk location.

Appendices:

Nomenclature

d_1	= The distance between joint 2 and joint 4
l_1	= The length of link 1
l_2	= The length of link 2
l_3	= The length of link 3
l_4	= The length of link 4
P_{ex}	= The x coordinate of the end effector location
P_{ey}	= The y coordinate of the end effector location
P_{ez}	= The z coordinate of the end effector location
P_{4xy}	= The distance from the base of the robot to the wrist in the x-y plane
P_{4z}	= The z coordinate of the wrist location
θ_1	= The angle of joint 1 in degrees
θ_2	= The angle of joint 2 in degrees
θ_3	= The angle of joint 3 in degrees
θ_4	= The angle of joint 4 in degrees
ϕ_6	= The angle of the end effector with reference to the base x-y plane in degrees

Servo Mapping Spreadsheet

Servo 1 Count	Servo 1 angle
1000	37.5
1250	63
1500	87.5
1750	112
2000	137

Servo 2 Count	Servo 2 angle
1000	40
1250	66.5
1500	92
1750	117
2000	143

Servo 3 Count	Servo 3 angle
1000	39
1250	66
1500	91
1750	118
2000	146

Servo 4 Count	Servo 4 angle
750	18
1000	43
1250	67
1750	113.5
2000	136

Servo 5 Count	Servo 5 angle
1000	37
1250	65
1500	90
1750	113
2000	131

Slope	y-intercept
10.080	618.99

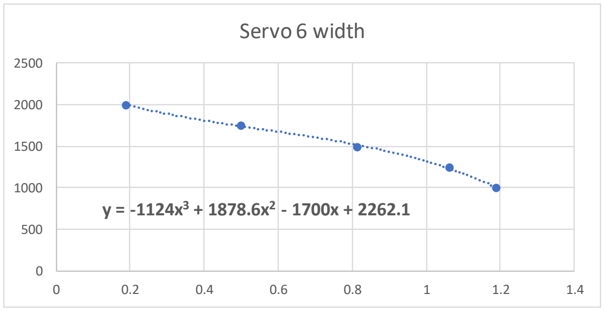
Slope	y-intercept
9.746	606.32

Slope	y-intercept
9.395	635.63

Slope	y-intercept
10.614	548.62

Slope	y-intercept
10.527	582.07

Servo 6 Count	Servo 6 width
1000	1.1875
1250	1.0625
1500	0.8125
1750	0.5
2000	0.1875



The DH Convention

Step 1: Locate and label the joint axes $z_0, ..., z_{n-1}$. Remember link i rotates about or translates along axes z_{i-1}.

Step 2: Establish a base frame. Chose x_0 and y_0 axes to ensure a right handed coordinate frame. Note: it is often helpful to consider the joint 2 in order to make generating the DH table easier.

Step 3: Do the following steps for frame 1 to $n-1$.

1. Locate the origin o_i and determine the direction of x_i based on which of the following cases is true:

   z_{i-1} **and** z_i **are parallel:** Locate o_i a convenient point on z_i —usually inside the link. Set x_i to point along the common normal between z_{i-1} and z_i.

   z_{i-1} **and** z_i **are not coplanar and do not intersect:** Locate where the common normal to z_{i-1} and z_i intersects z_i. Place the origin of frame i, o_i at this location. Set x_i point along the common normal between z_{i-1} and z_i.

   z_{i-1} **intersects** z_i**:** Place o_i at the intersection of z_{i-1} and z_i. Set the location of x_i so that it is normal to the z_{i-1} - z_i plane.

2. Place y_i to complete a right-handed coordinate frame.

Step 4: Establish the end effector frame n. Assuming the n^{th} joint is revolute, set z_n parallel to z_{n-1}. Place the origin o_n conveniently along the z_n-axis (between the fingers of the gripper or at the tool tip are common choices). The y-axis should point in the direction of the closing motion of the manipulator. The x-axis is chosen to make the n frame a right handed coordinate system.

Step 5: Create the the DH table. This requires that you remember the phrase that will impress anyone, "Rotate translate translate rotate." Once you utter these words people know you are a roboticist and must be respected and loved.

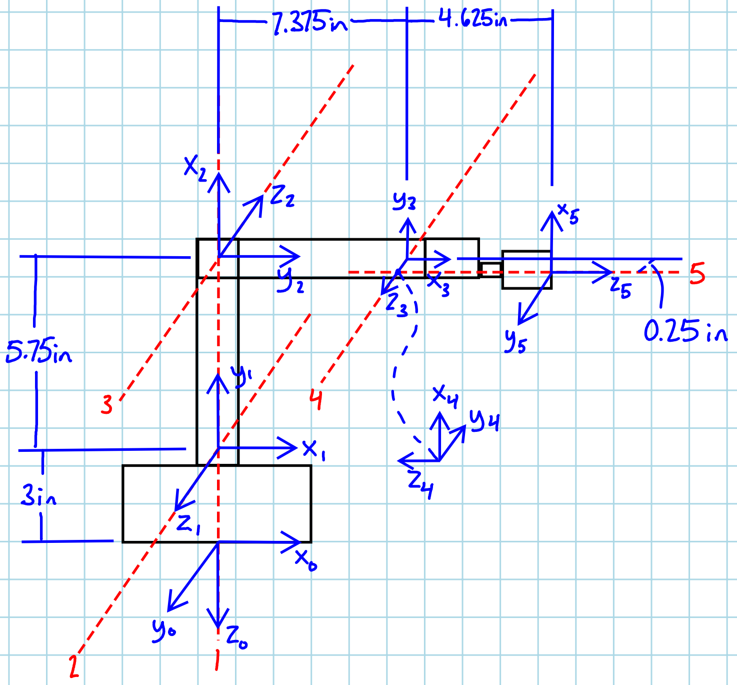
1. Rotate about the current z-axis to align the x_i with x_{i-1}. That rotation is the value θ_i.
2. Then translate along the current z-axis until the origin o_i is at the point where the x_i-axis intersects z_{i-1}-axis. The value of d_i is the distance translated along the z-axis.
3. Now translate along the current x-axis to the final location for o_i. The translation along x is a_i.
4. Then rotate about the current x-axis to align the z-axis of the current frame to the desired z_i. This rotation is α_i.

Dean Rogers

Lab 1

ME 439

①



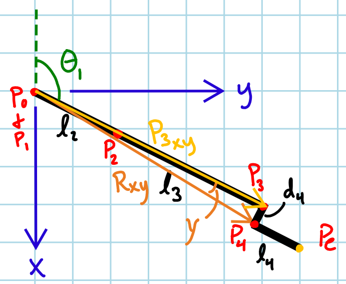
#	θ	d	a	α	var
$0\to1$	0	-3in	0	$-\pi/2$	θ_1
$1\to2$	$\pi/2$	0	5.75in	π	θ_2
$2\to3$	$\pi/2$	0	7.375in	π	θ_3
$3\to4$	$\pi/2$	0.25in	0	$-\pi/2$	θ_4
$4\to5$	0	-4.625in	0	π	θ_5

Dean Rogers

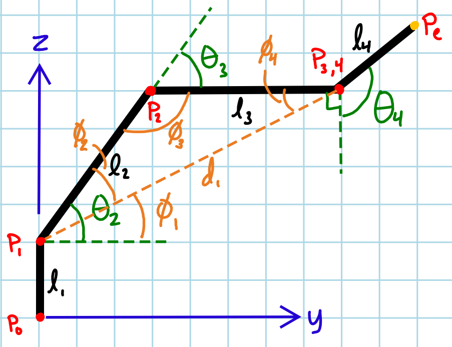
Lab 1

Inverse Kinematics - For wrist position (P_4)

Top View Fig.1



Side View (at $\theta_1 = 90°$) Fig.2



$$\phi_3 = 180° - \theta_3$$

$$\theta_2 = \phi_1 + \phi_2$$

$$180° = \phi_2 + \phi_3 + \phi_4$$

$$R_{xy} = \sqrt{P_{4x}^2 + P_{4y}^2}$$

$$P_{4x} = -R_{xy}\cos(\theta_1 + \gamma)$$

$$\gamma = \sin^{-1}\left(\frac{d_4}{R_{xy}}\right)$$

$$\theta_1 = \cos^{-1}\left(\frac{P_{4x}}{-R_{xy}}\right) - \sin^{-1}\left(\frac{d_4}{R_{xy}}\right)$$

$$= \cos^{-1}\left(\frac{P_{4x}}{-\sqrt{P_{4x}^2 + P_{4y}^2}}\right) - \sin^{-1}\left(\frac{d_4}{\sqrt{P_{4x}^2 + P_{4y}^2}}\right)$$

$$P_{4xy} = \sqrt{R_{xy}^2 - d_4^2} = \sqrt{P_{4x}^2 + P_{4y}^2 - d_4^2}$$

$$d_1 = \sqrt{P_{4xy}^2 + P_{4z}^2} = \sqrt{P_{4x}^2 + P_{4y}^2 - d_4^2 + (P_{4z} - l_1)^2}$$

$$P_{4xy} = d_1 \cos\phi_1$$

$$P_{4z} = d_1 \sin\phi_1 + l_1$$

$$d_1^2 = l_2^2 + l_3^2 - 2 l_2 l_3 \cos(180° - \theta_3)$$

$$\cos(180° - \theta_3) = \frac{d_1^2 - l_2^2 - l_3^2}{-2 l_2 l_3}$$

$$180° - \theta_3 = \cos^{-1}\left(\frac{l_2^2 + l_3^2 - d_1^2}{2 l_2 l_3}\right)$$

$$\theta_3 = 180° - \cos^{-1}\left(\frac{l_2^2 + l_3^2 - d_1^2}{2 l_2 l_3}\right)$$

$$\frac{d_1}{\sin\phi_3} = \frac{l_3}{\sin\phi_2}$$

$$\sin\phi_2 = \frac{l_3 \sin\phi_3}{d_1}$$

$$\phi_2 = \theta_2 - \phi_1 = \sin^{-1}\left(\frac{l_3 \sin\phi_3}{d_1}\right)$$

$$\theta_2 = \sin^{-1}\left(\frac{P_{4z} - l_1}{d_1}\right) + \sin^{-1}\left(\frac{l_3 \sin(180° - \theta_3)}{d_1}\right)$$

Dean Rogers

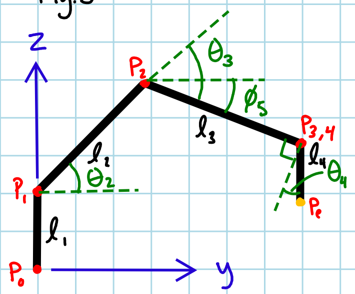
Lab 1

Inverse Kinematics cont...

There are three possible configurations of link 4 to give the desired end effector position. We will try to position link 4 straight down if possible.

Link 4 points down
(When θ_4 required is > 0)

Fig.3



$$\phi_5 = \theta_3 - \theta_2 = \theta_4$$

$$P_{4_x} = P_{e_x}$$

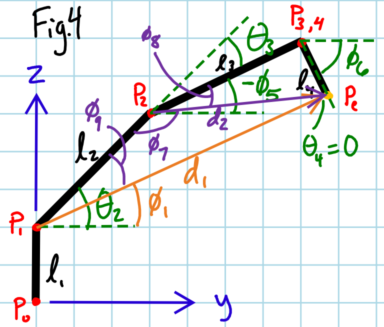
$$P_{4_y} = P_{e_y}$$

$$P_{4_z} = P_{e_z} + l_4$$

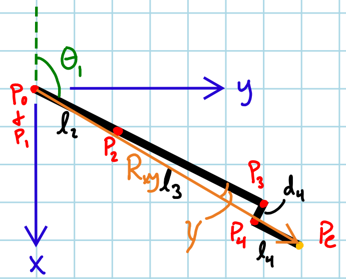
Then inverse kinematics to wrist position can be used

Link 4 stays at minimum angle
(When θ_4 required < 0)

Fig.4



Top View Fig.5



$$\phi_6 = 90° + \phi_5 - \theta_4 = 90° + \theta_3 - \theta_2 - \theta_4$$

$$R_{xy} = \sqrt{P_{e_x}^2 + P_{e_y}^2}$$

$$d_1 = \sqrt{R_{xy}^2 + (P_{e_z} - l_1)^2} = \sqrt{P_{e_x}^2 + P_{e_y}^2 + (P_{e_z} - l_1)^2}$$

$$\phi_1 = \sin^{-1}\left(\frac{P_{e_z} - l_1}{d_1}\right)$$

When $\theta_4 = 0°$

$$d_2 = \sqrt{l_3^2 + l_4^2}$$

$$d_1^2 = l_2^2 + d_2^2 - 2 l_2 d_2 \cos(\phi_7)$$

$$\phi_7 = \cos^{-1}\left(\frac{l_2^2 + d_2^2 - d_1^2}{2 l_2 d_2}\right)$$

$$\phi_8 = \tan^{-1}\left(\frac{l_4}{l_3}\right)$$

$$\theta_3 = 180° - (\phi_7 + \phi_8) = 180° - \left[\cos^{-1}\left(\frac{l_2^2 + d_2^2 - d_1^2}{2 l_2 d_2}\right) + \tan^{-1}\left(\frac{l_4}{l_3}\right)\right]$$

$$\frac{d_1}{\sin\phi_7} = \frac{d_2}{\sin\phi_9}$$

$$\phi_9 = \theta_2 - \phi_1 = \sin^{-1}\left(\frac{d_2 \sin\phi_7}{d_1}\right)$$

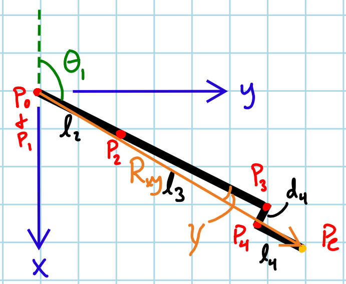
Dean Rogers

Lab 1

Inverse Kinematics cont...

When $\theta_4 = 0$ cont...

Top View Fig.5



$$\theta_2 = \phi_1 + \sin^{-1}\left(\frac{d_2 \sin \phi_7}{d_1}\right)$$

$$\phi_6 = 90^\circ + \theta_3 - \theta_2$$

$$P_{3xy} = l_2 \cos\theta_2 + l_3 \cos(\theta_2 - \theta_3)$$

$$R_{4e_{xy}} = l_4 \cos\phi_6$$

$$R_{xy} = \frac{P_{3xy} + R_{4e_{xy}}}{\cos\gamma} = \sqrt{P_{ex}^2 + P_{ey}^2}$$

$$\gamma = \cos^{-1}\left(\frac{P_{3xy} + R_{4e_{xy}}}{\sqrt{P_{ex}^2 + P_{ey}^2}}\right) = \cos^{-1}\left(\frac{l_2\cos\theta_2 + l_3\cos(\theta_2-\theta_3) + l_4\cos\phi_6}{\sqrt{P_{ex}^2+P_{ey}^2}}\right)$$

$$P_{ex} = -R_{xy}\cos(\theta_1 + \gamma)$$

$$\theta_1 = \cos^{-1}\left(\frac{P_{ex}}{-R_{xy}}\right) - \cos^{-1}\left(\frac{l_2\cos\theta_2 + l_3\cos(\theta_2-\theta_3) + l_4\cos\phi_6}{\sqrt{P_{ex}^2+P_{ey}^2}}\right)$$

$$= \cos^{-1}\left(\frac{P_{ex}}{-\sqrt{P_{ex}^2+P_{ey}^2}}\right) - \cos^{-1}\left(\frac{l_2\cos\theta_2 + l_3\cos(\theta_2-\theta_3) + l_4\cos\phi_6}{\sqrt{P_{ex}^2+P_{ey}^2}}\right)$$

When ϕ_6 is specified

Side View

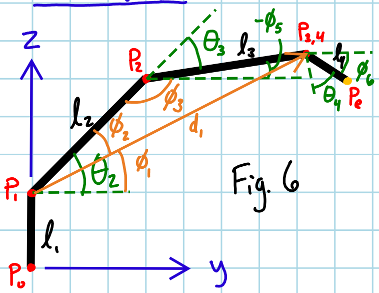
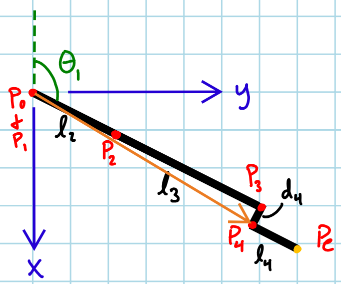


Fig. 6

Top View Fig. 7



$$P_{4_{xy}} = P_{e_{xy}} - l_4\cos\phi_6$$

$$P_{4_z} = P_{e_z} + l_4 \sin\phi_6$$

$$d_1 = \sqrt{P_{4_{xy}}^2 + (P_{4_z} - l_1)^2}$$

$$d_1^2 = l_2^2 + l_3^2 - 2 l_2 l_3 \cos(180^\circ - \theta_3)$$

$$\theta_3 = 180^\circ - \cos^{-1}\left(\frac{l_2^2 + l_3^2 - d_1^2}{2 l_2 l_3}\right)$$

$$\frac{d_1}{\sin\phi_3} = \frac{l_3}{\sin\phi_2}$$

$$\phi_2 = \theta_2 - \phi_1 = \sin^{-1}\left(\frac{l_3 \sin\phi_3}{d_1}\right)$$

$$\theta_2 = \sin^{-1}\left(\frac{l_3\sin\phi_3}{d_1}\right) + \sin^{-1}\left(\frac{P_{4_z} - l_1}{d_1}\right)$$

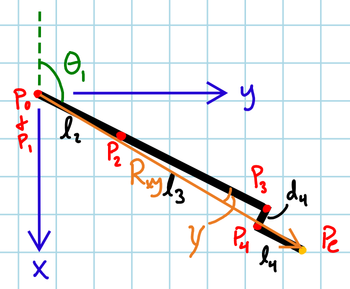
$$P_{e_{xy}} = \sqrt{P_{ex}^2 + P_{ey}^2}$$

$$\phi_1 = \sin^{-1}\left(\frac{P_{4_z}}{d_1}\right)$$

Inverse Kinematics cont...

When ϕ_6 is specified cont...

Top View Fig.5



$$P_{3xy} = l_2 \cos\theta_2 + l_3 \cos(\theta_2 - \theta_3)$$

$$R_{4e_{xy}} = l_4 \cos\phi_6$$

$$R_{xy} = \frac{P_{3xy} + R_{4e_{xy}}}{\cos\gamma} = \sqrt{P_{ex}^2 + P_{ey}^2}$$

$$\gamma = \cos^{-1}\left(\frac{P_{3xy} + R_{4e_{xy}}}{\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right) = \cos^{-1}\left(\frac{l_2 \cos\theta_2 + l_3\cos(\theta_2-\theta_3) + l_4 \cos\phi_6}{\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right)$$

$$P_{ex} = -R_{xy}\cos(\theta_1 + \gamma)$$

$$\theta_1 = \cos^{-1}\left(\frac{P_{ex}}{-R_{xy}}\right) - \cos^{-1}\left(\frac{l_2 \cos\theta_2 + l_3\cos(\theta_2-\theta_3) + l_4 \cos\phi_6}{\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right)$$

$$= \cos^{-1}\left(\frac{P_{ex}}{-\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right) - \cos^{-1}\left(\frac{l_2 \cos\theta_2 + l_3\cos(\theta_2-\theta_3) + l_4 \cos\phi_6}{\sqrt{P_{e_x}^2 + P_{e_y}^2}}\right)$$

Dean Rogers

Lab 1

Inverse Kinematics Summary

1) When link 4 can point straight down

$$\theta_1 = \cos^{-1}\left(\frac{P_{e_x}}{-\sqrt{P_{e_x}^2+P_{e_y}^2}}\right) - \sin^{-1}\left(\frac{d_4}{\sqrt{P_{e_x}^2+P_{e_y}^2}}\right)$$

$$d_1 = \sqrt{P_{e_x}^2+P_{e_y}^2-d_4^2+(P_{e_z}+l_4-l_1)^2}$$

$$\theta_3 = 180^\circ - \cos^{-1}\left(\frac{l_2^2+l_3^2-d_1^2}{2l_2l_3}\right)$$

$$\theta_2 = \sin^{-1}\left(\frac{P_{e_z}+l_4-l_1}{d_1}\right) + \sin^{-1}\left(\frac{l_3\sin(180^\circ-\theta_3)}{d_1}\right)$$

$$\theta_4 = \theta_3 - \theta_2$$

2) When $\theta_3-\theta_2<0$ and link 4 stays at its minimum angle, $\theta_4=0$

$$d_2 = \sqrt{l_3^2+l_4^2}$$

$$d_1 = \sqrt{P_{e_x}^2+P_{e_y}^2+(P_{e_z}-l_1)^2}$$

$$\theta_3 = 180^\circ - \left[\cos^{-1}\left(\frac{l_2^2+d_2^2-d_1^2}{2l_2d_2}\right) + \tan^{-1}\left(\frac{l_4}{l_3}\right)\right]$$

$$\theta_2 = \sin^{-1}\left(\frac{d_2\sin\phi_7}{d_1}\right) + \sin^{-1}\left(\frac{P_{e_z}-l_1}{d_1}\right)$$

$$\theta_1 = \cos^{-1}\left(\frac{P_{e_x}}{-\sqrt{P_{e_x}^2+P_{e_y}^2}}\right) - \cos^{-1}\left(\frac{l_2\cos\theta_2+l_3\cos(\theta_2-\theta_3)+l_4\cos\phi_6}{\sqrt{P_{e_x}^2+P_{e_y}^2}}\right)$$

$$\theta_4 = 0$$

3) When ϕ_6 is specified

$$P_{4_{xy}} = \sqrt{P_{e_x}^2+P_{e_y}^2} - l_4\cos\phi_6$$

$$P_{4_z} = P_{e_z} + l_4\sin\phi_6$$

$$d_1 = \sqrt{P_{4_{xy}}^2+(P_{4_z}-l_1)^2}$$

$$\theta_3 = 180^\circ - \cos^{-1}\left(\frac{l_2^2+l_3^2-d_1^2}{2l_2l_3}\right)$$

$$\theta_2 = \sin^{-1}\left(\frac{l_3\sin(180^\circ-\theta_3)}{d_1}\right) + \sin^{-1}\left(\frac{P_{4_z}-l_1}{d_1}\right)$$

$$\theta_1 = \cos^{-1}\left(\frac{P_{e_x}}{-\sqrt{P_{e_x}^2+P_{e_y}^2}}\right) - \cos^{-1}\left(\frac{l_2\cos\theta_2+l_3\cos(\theta_2-\theta_3)+l_4\cos\phi_6}{\sqrt{P_{e_x}^2+P_{e_y}^2}}\right)$$

$$\theta_4 = 90^\circ + \theta_3 - \theta_2 - \phi_6$$

```
clc
close all
clear all

global towerHeight;

% Get desired Hanoi Tower height
promptHeight = 'Enter the height of the Hanoi Tower:';
towerHeight = inputdlg(promptHeight);
towerHeight = str2double(towerHeight{1});

% Check that input is an integer
if mod(towerHeight, 1) ~= 0
    error('Height of the tower must be an integer')
end

% Declare Variables
global stacks height moveSeq seqNum;

% Initialize Variables
numOfMoves = 2^towerHeight - 1  % Number of moves to complete the tower
stacks = zeros(towerHeight,3);  % Visualization of stacks
stacks(:,1) = 1:towerHeight;    % Initial setup all disks on stack 1
height = [towerHeight 0 0];     % Track tower height
moveSeq = zeros(numOfMoves,5);  % Has columns
                    %[fromPostNum toPostNum post1Height post2Height post3Height]
seqNum = 1;                     % Track move numbers in moveDisk function
diskHeight = 0.975;             % Disk height (in)
nubHeight = 0.625;              % Nub height (in)
postLoc = [                     % Post locations (in) [x y]
    -5.75 8.5;
    0 8.5;
    5.75 8.5];
homeLoc = [0 3 2];              % Home location (in) [x y z]
mult = 4;                       % Nub Height multiplier to get height to raise
                                    % disk before moving
trqOffset = [0.01 0.035];       % Offset due to torque (in) [x y]
tStep = 0.08;                   % Trajectory time step
accT = 1.5;                     % Acceleration Time
qdMax = [2 2 2];                % Max axis speed
sp = '900';                     % Servo speed
spJaw = '2000';                 % Jaw servo speed
jawOC = [1.15 0.73];            % Jaw open & closed positions (in)
jawMap = [                      % Equation to map jaw width to servo count
    -1124
    1878.6
    -1700
    2262.1];

% Display initial Setup
stacks
```

```
% Run recursive function to move disks
moveTower(towerHeight, 1, 3, 2)

% Display move sequence for robot
moveSeq

% Initialize robot arm and move to home position
% NOTE: The 'serialport' object only works with R2019b.  The 'serial' object
% used in R2018b stopped working with my robot
s = serialport("COM3",115200);
configureTerminator(s,'CR');
p = invKin(homeLoc,'Home');
for k=1:numrows(p)
    writeline(s,strcat('#0P',int2str(p(k,1)),'S',sp,'#1P',int2str(p(k,2)),...
        'S',sp,'#2P',int2str(p(k,3)),'S',sp,'#3P',int2str(p(k,4)),'S',sp,...
        '#4P',int2str(p(k,5)),'S',sp))
end
writeline(s,strcat('#5P',int2str(polyvalm(jawMap,jawOC(1))),'S300'))

% Move Robot to solve tower
for i=1:numOfMoves
    fromPost = moveSeq(i,1);
    toPost = moveSeq(i,2);

    % Get via points to move to next disk
    %------------------------------
    % If just starting
    if i == 1
        % Set via to move directly from home location to first post
        via = [
            homeLoc(1) homeLoc(2) homeLoc(3)
            postLoc(fromPost,1) postLoc(fromPost,2) (moveSeq(i,fromPost + 2)...
                * diskHeight) + (mult * nubHeight)
            postLoc(fromPost,1) postLoc(fromPost,2) moveSeq(i,fromPost + 2)...
                * diskHeight
        ];
    else
        % Get the post at the current arm location
        prevPost = moveSeq(i-1,2);

        % If moving over middle post and middle post too high
        if prevPost + fromPost == 4 && (moveSeq(i,prevPost + 2) + ...
                moveSeq(i,fromPost + 2)) / 2 < moveSeq(i,4)
            % Set via points to avoid middle post
            via = [
                postLoc(prevPost,1) postLoc(prevPost,2) ...
                    moveSeq(i,prevPost + 2) * diskHeight
                postLoc(prevPost,1) postLoc(prevPost,2) ...
                    (moveSeq(i,4) * diskHeight) + (mult * nubHeight)
                postLoc(fromPost,1) postLoc(fromPost,2) ...
                    (moveSeq(i,4) * diskHeight) + (mult * nubHeight)
                postLoc(fromPost,1) postLoc(fromPost,2) ...
                    moveSeq(i,fromPost + 2) * diskHeight
            ];
        else
            % Set via to move directly from start to finish
            via = [
                postLoc(prevPost,1) postLoc(prevPost,2) ...
                    moveSeq(i,prevPost + 2) * diskHeight
                postLoc(prevPost,1) postLoc(prevPost,2) ...
                    (moveSeq(i,prevPost + 2) * diskHeight) + (mult * nubHeight)
                postLoc(fromPost,1) postLoc(fromPost,2) ...
                    (moveSeq(i,fromPost + 2) * diskHeight) + (mult * nubHeight)
                postLoc(fromPost,1) postLoc(fromPost,2) ...
                    moveSeq(i,fromPost + 2) * diskHeight
            ];
        end
    end

    % Generate trajectory to move arm to next disk
    dTraj = mstraj(via,qdMax,[],[],tStep,accT);

    % Apply inverse kinematics
    if i==1
        dTraj = invKin(dTraj,'Start');
    else
        dTraj = invKin(dTraj,'Move');
    end

    % Move robot arm
    for j=1:numrows(dTraj)

        % Write command to serial
        writeline(s,strcat('#0P',int2str(dTraj(j,1)),'S',sp,'#1P',...
            int2str(dTraj(j,2)),'S',sp,'#2P',int2str(dTraj(j,3)),'S',sp,...
            '#3P',int2str(dTraj(j,4)),'S',sp,'#4P',int2str(dTraj(j,5)),'S',sp))

        % Wait for previous move to finish
        qs='+';
        while qs=='+'
            writeline(s,"Q");
            qs = read(s,1,'char');
            %qs=char(qs)
        end
    end

    % Close jaw
    writeline(s,strcat('#5P',int2str(polyvalm(jawMap,jawOC(2))),'S',spJaw))
    qs='+';
    while qs=='+'
        writeline(s,"Q");
```

```
        qs = read(s,1,'char');
        %qs=char(qs)
    end

    % Get via points to move the disk to its new location
    %-----------------------------
    % If moving over middle post and middle post too high
    if fromPost + toPost == 4 && (moveSeq(i,3) + moveSeq(i,5))/2 < moveSeq(i,4)
        % Set via points to avoid middle post
        via = [
            postLoc(fromPost,1) postLoc(fromPost,2) ...
                moveSeq(i,fromPost + 2) * diskHeight
            postLoc(fromPost,1) postLoc(fromPost,2) ...
                (moveSeq(i,4) * diskHeight) + (mult * nubHeight)
            postLoc(toPost,1) postLoc(toPost,2) ...
                ((moveSeq(i,4) + 1)* diskHeight) + (mult * nubHeight)
            postLoc(toPost,1) postLoc(toPost,2) ...
                (moveSeq(i,toPost + 2) + 1) * diskHeight
        ];
    else
        % Set via to move directly from start to finish
        via = [
            postLoc(fromPost,1) postLoc(fromPost,2) ...
                moveSeq(i,fromPost + 2) * diskHeight
            postLoc(fromPost,1) postLoc(fromPost,2) ...
                (moveSeq(i,fromPost + 2) * diskHeight) + (mult * nubHeight)
            postLoc(toPost,1) postLoc(toPost,2) ...
                ((moveSeq(i,toPost + 2) + 1) * diskHeight) + (mult * nubHeight)
            postLoc(toPost,1) postLoc(toPost,2) ...
                (moveSeq(i,toPost + 2) + 1) * diskHeight
        ];

        % Limit path angle between stacks
        if abs(moveSeq(i,fromPost + 2) - (moveSeq(i,toPost + 2) + 1)) > 3
           via(3,3) = ((moveSeq(i,toPost + 2) + 4) * diskHeight) + ...
               (mult * nubHeight);
        elseif abs(moveSeq(i,fromPost + 2) - (moveSeq(i,toPost + 2) + 1)) < -3
            via(2,3) = ((moveSeq(i,fromPost + 2) + 3) * diskHeight) + ...
                (mult * nubHeight);
        end
    end

    % Correct x-y position inaccuracy due to torque
    if via(3,1) > 0 && (moveSeq(i,toPost + 2) + 1) <= moveSeq(i,fromPost + 2)
        via(3:4,1) = via(3:4,1) - ((abs(moveSeq(i,toPost + 2) + 1 - ...
            moveSeq(i,fromPost + 2)) + 1) * trqOffset(1)) - trqOffset(1);
    elseif via(3,1) < 0 && (moveSeq(i,toPost + 2) +1) <= moveSeq(i,fromPost + 2)
        via(3:4,1) = via(3:4,1) + ((abs(moveSeq(i,toPost + 2) + 1 - ...
            moveSeq(i,fromPost + 2)) + 1) * trqOffset(1)) - trqOffset(1);
    elseif via(3,1) > 0 && (moveSeq(i,toPost + 2) + 1) > moveSeq(i,fromPost + 2)
        via(3:4,1) = via(3:4,1) + ((abs(moveSeq(i,toPost + 2) + 1 - ...
            moveSeq(i,fromPost + 2)) + 1) * trqOffset(1));
    elseif via(3,1) < 0 && (moveSeq(i,toPost + 2) + 1) > moveSeq(i,fromPost + 2)
        via(3:4,1) = via(3:4,1) - ((abs(moveSeq(i,toPost + 2) + 1 - ...
            moveSeq(i,fromPost + 2)) + 1) * trqOffset(1));
    end

    if via(3,2) > 0 && (moveSeq(i,toPost + 2) + 1) <= moveSeq(i,fromPost + 2)
        via(3:4,2) = via(3:4,2) - ((abs(moveSeq(i,toPost + 2) + 1 - ...
            moveSeq(i,fromPost + 2)) + 1) * trqOffset(2));
    elseif via(3,2) > 0 && (moveSeq(i,toPost + 2) + 1) > moveSeq(i,fromPost + 2)
        via(3:4,2) = via(3:4,2) + ((abs(moveSeq(i,toPost + 2) + 1 - ...
            moveSeq(i,fromPost + 2)) + 1) * trqOffset(2));
    end

    % Generate disk move trajectory
    dTraj = mstraj(via,qdMax,[],[],tStep,accT);

    % Apply inverse kinematics
    dTraj = invKin(dTraj,'Move');

    % Move robot arm
    for j=1:numrows(dTraj)

        % Write command to serial
        writeline(s,strcat('#0P',int2str(dTraj(j,1)),'S',sp,'#1P',...
            int2str(dTraj(j,2)),'S',sp,'#2P',int2str(dTraj(j,3)),'S',sp,...
            '#3P',int2str(dTraj(j,4)),'S',sp,'#4P',int2str(dTraj(j,5)),'S',sp))

        % Wait for previous move to finish
        qs='+';
        while qs=='+'
            writeline(s,"Q");
            qs = read(s,1,'char');
        end
    end

    % Open jaw
    writeline(s,strcat('#5P',int2str(polyvalm(jawMap,jawOC(1))),'S',spJaw))
    qs='+';
    while qs=='+'
        writeline(s,"Q");
        qs = read(s,1,'char');
    end

    % If last move, move back to home position
    if i==numOfMoves
        % Set via to move directly from last post to home
        via = [
            postLoc(toPost,1) postLoc(toPost,2) ...
                (moveSeq(i,toPost + 2) + 1) * diskHeight
            postLoc(toPost,1) postLoc(toPost,2) ...
```

```
                (moveSeq(i,toPost + 2) * diskHeight) + (mult * nubHeight)
            homeLoc(1) homeLoc(2) homeLoc(3)
        ];

        % Generate return trajectory
        dTraj = mstraj(via,qdMax,[],[],tStep,accT);

        % Apply inverse kinematics
        dTraj = invKin(dTraj,'End');

        % Move robot arm
        for j=1:numrows(dTraj)

            % Write command to serial
            writeline(s,strcat('#0P',int2str(dTraj(j,1)),'S',sp,'#1P',...
                int2str(dTraj(j,2)),'S',sp,'#2P',int2str(dTraj(j,3)),'S',sp,...
                '#3P',int2str(dTraj(j,4)),'S',sp,'#4P',int2str(dTraj(j,5)),...
                'S',sp))

            % Wait for previous move to finish
            qs='+';
            while qs=='+'
                writeline(s,"Q");
                qs = read(s,1,'char');
            end
        end
    end
end

% Close serial port
clear s

%--------------------------------- Functions ----------------------------------

function srvCount = invKin(pose, moveType)
    % The input, 'pose', is a Nx3 matrix of the form [x y z] which represents
    % the desired location of the end effector of the Lynxmotion AL5D robot,
    % where N is the number of steps.
    % The input, 'moveType' is used to specify the angle, phi6, of the end
    % effector of the robot.
    %------------------------------------------------------------------------
    % The output, 'srvCount', is a Nx5 matrix of the form
    % [rot1 rot2 rot3 rot4 rot5] which represents the servo rotation count
    % which corresponds to an angle of rotation. N is the same as the
    % input N; the number of steps.

    % Initialize variables
    l1 = 3.5;
    l2 = 5.75;
    l3 = 7.375;
    l4 = 4.625;
    d4 = 0.25;
    srvCount = zeros([numrows(pose) 5]);
    countMap = [
        10.200  567
        9.746   606.32
        9.395   635.63
        10.614  548.62
        10.527  630
        ];

    % Determine end effector angle, phi6
    if strcmp(moveType,'Start')
        phi6(1:30,1) = 90:-45/(29):45;
        phi6(31:numrows(pose),1) = 45;
    elseif strcmp(moveType,'End')
        phi6(:,1) = 45:45/(numrows(pose)-1):90;
    elseif strcmp(moveType,'Move')
        phi6 = 45;
    elseif strcmp(moveType,'Home')
        phi6 = 90;
    end

    % Calculate variables
    P4xy = sqrt(pose(:,1).^2 + pose(:,2).^2) - l4 * cosd(phi6);
    P4z = pose(:,3) + l4 * sind(phi6);
    d1 = sqrt(P4xy.^2 + (P4z - l1).^2);

    % Calculate joint angles in degrees
    srvCount(:,3) = 180 - acosd((l2^2 + l3^2 - d1.^2)/(2*l2*l3));
    srvCount(:,2) = asind((l3 * sind(180 - srvCount(:,3)))./d1) + ...
        asind((P4z - l1)./d1);
    srvCount(:,1) = acosd(pose(:,1)./(-1*sqrt(pose(:,1).^2 + pose(:,2).^2)))...
        - acosd((l2*cosd(srvCount(:,2)) + l3*cosd(srvCount(:,2) - ...
        srvCount(:,3)) + l4*cosd(phi6))./sqrt(pose(:,1).^2 + pose(:,2).^2));
    srvCount(:,4) = max(0, 90 + srvCount(:,3) - srvCount(:,2) - phi6);
    srvCount(:,5) = 90;

    % Check for angles out of limits
    if min(min(srvCount(:,1:4))) < 0.0 || max(max(srvCount(:,1:4))) > 180.0
        error("The robot cannot reach that position")
    end

    % Convert angles to servo rotation counts
    srvCount(:,1) = (srvCount(:,1) .* countMap(1,1)) + countMap(1,2);
    srvCount(:,2) = (srvCount(:,2) .* countMap(2,1)) + countMap(2,2);
    srvCount(:,3) = (srvCount(:,3) .* countMap(3,1)) + countMap(3,2);
    srvCount(:,4) = (srvCount(:,4) .* countMap(4,1)) + countMap(4,2);
    srvCount(:,5) = (srvCount(:,5) .* countMap(5,1)) + countMap(5,2);
end

function moveDisk(disk, fromCol, toCol)
```

```
    global stacks height moveSeq towerHeight seqNum;

    % Log the move sequence for robot
    moveSeq(seqNum,1) = fromCol;
    moveSeq(seqNum,2) = toCol;
    moveSeq(seqNum,3:5) = height;

    % Update disk locations in visual array
    stacks(towerHeight - height(fromCol) + 1,fromCol) = 0;
    stacks(towerHeight - height(toCol),toCol) = disk

    % Update stack heights
    height(fromCol) = height(fromCol) - 1;
    height(toCol) = height(toCol) + 1;

    % Increment sequence number
    seqNum = seqNum + 1;
end

function moveTower(disk, source, dest, spare)
    if disk == 1
        % If current disk is the smallest disk, move to the destination stack
        moveDisk(disk, source, dest);
    else
        % Step into next smallest disk and move whole tower to spare stack
        moveTower(disk - 1, source, spare, dest);

        % Move current disk to the destination stack
        moveDisk(disk,source, dest);

        % Step into next smallest disk and move whole tower to destination stack
        moveTower(disk - 1, spare, dest, source);
    end
end
```