My ECE/ME 739 Introduction to Robotics Spring 2015, University of Wisconsin, Madison

Nasser M. Abbasi

spring 2015 Compiled on May 3, 2022 at 11:07pm [public]

Contents

Chapter 1

Introduction

Took this course in Spring 2015 to learn a little about dyanmics of robotics. Instructor: professor Michael Zinn

1.1 Syllubus

ME/ECE 739 Introduction to Robotics

Course Description:	A first course in robotics analysis and design, focusing on the analytical fundamentals specific to robotic manipulators. Topics to be covered included serial chain robotic manipulator kinematics, dynamics, motion planning, and controls.
Prerequisites:	ME 446 or ECE 332 or equivalent 1st semester controls course, Math 320 or 340 or equivalent basic linear algebra course. In addition to the formal prerequisites, familiarity with the following topic areas will be helpful There are no required prerequisites for the class - but exposure to the following topic areas will be helpful ▪ Kinematics and dynamics (at ME 240 level) ▪ System dynamics (at ME 340 level) ▪ Working knowledge of Matlab (*important for homework*)
Lectures:	Posted weekly to the course Learn@UW course page
Instructor:	Prof. Michael Zinn 2242 ME bldg. 608.263.2893 / mzinn@wisc.edu
Textbook:	*Robot Modeling and Control,* by Spong, Hutchinson, and Vidyasagar, published by John Wiley & Sons, Inc., 2006 ISBN-10: 0-471-64990-2 / ISBN-13: 978-0-471-64990-8
Learn@UW:	The course Learn@UW web page will be used to post course material throughout the semester. The page will have links to the recorded lectures, course schedule/syllabus, the PowerPoint lecture notes, assigned homework, and miscellaneous supporting material. I recommend you check this periodically to make sure you have the up-to-date course material.
Homework:	Homework will be assigned approximately every two weeks and will be due at the beginning of class on the specified due date.
Final Project / Paper Review:	Students will select a robotics design problem of their choosing and use a combination of the tools learned in the course to develop a final design. A final report, including working Matlab code, is required. The chosen design problem must be approved by the instructor
Grading:	Grades are based on your performance on the assigned homework (75%) and the final project (25%).

ME/ECE 739 Introduction to Robotics

Course Schedule (Approximate):

Approx. Date		Topic	Text
January	1/20 (T)	1 . Introduction	Spong 1.1 - 1.4
	1/22 (Th)	2. Rigid Body Motion Representation	Spong 2.1 - 2.8
	1/27 (T)		
	1/29 (Th)		
February	2/3 (T)		
	2/5 (Th)	3. Forward and Inverse Kinematics	Spong 3.1 - 3.4
	2/10 (T)		
	2/12 (Th)		
	2/17 (T)	4. Velocity Kinematics	Spong 4.1 - 4.13
	2/19 (Th)		
	2/24 (T)		
	2/26 (Th)		
March	3/3 (T)	5. Dynamics	Spong 7.1 - 7.6
	3/5 (Th)		
	3/10 (T)		
	3/12 (Th)		
	3/17 (T)	6. Motion Planning / Trajectory Generation	Spong 5.1 - 5.5
	3/19 (Th)		
	3/24 (T)		
	3/26 (Th)	7. Independent Joint Control	Spong 6.1 - 6.5
April	4/7 (T)		
	4/9 (Th)		
	4/14 (T)	8. Multivariable Control	Spong 8.1 - 8.2
	4/16 (Th)		
	4/21 (T)		
	4/23 (Th)		
	4/28 (T)	9. Force Control and Miscellaneous Topics	Spong 9.1 - 9.4
	4/30 (Th)		
May	5/5 (T)		
	5/7 (Th)		

Chapter 2

HWs

2.1 HW 1

2.1.1 Problem 1

problem description

(1) [Spong 2-15] Suppose that three coordinate frames $o_1x_1y_1z_1$, $o_2x_2y_2z_2$, and $o_3x_3y_3z_3$ are given, and suppose

$$R_2^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & -\sqrt{3}/2 \\ 0 & \sqrt{3}/2 & 1/2 \end{bmatrix}, \quad R_3^1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

►Find the matrix R_3^2

solution

Starting with the relation

$$R_3^1 = R_2^1 R_3^2$$

Pre-multiplying both sides by $(R_2^1)^{-1}$ which exists since R is a rotation matrix and hence invertible results in

$$R_3^2 = (R_2^1)^{-1} R_3^1$$

For a rotation matrix the following relation holds

$$(R_2^1)^{-1} = (R_2^1)^T$$

Therefore

$$
\begin{aligned}
R_3^2 &= (R_2^1)^T R_3^1 \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{-\sqrt{3}}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix}^T \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \\ 0 & \frac{-\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & -1 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{-\sqrt{3}}{2} & 0 \end{pmatrix}
\end{aligned}
$$

2.1.2 Problem 2

► Find the matrix R_3^2

(2) [Spong 2-38] Consider the adjacent diagram. Find the homogeneous transformations T_1^0, T_2^0, T_2^1 representing the transformations among the three frames shown.
► Show that $T_2^0 = T_1^0 T_2^1$

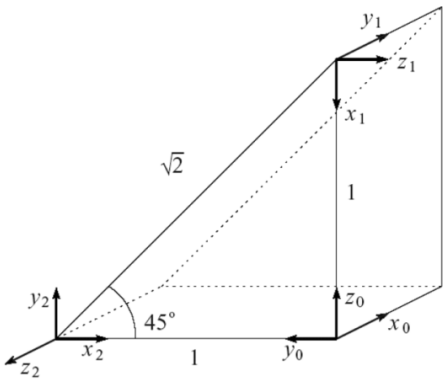


Figure 2.1: problem 2 description

The goal is to determine T_1^0, T_2^0, T_3^0 and T_3^2. T_i^{i-1} is the homogeneous transformation from frame $\{i-1\}$ to frame $\{i\}$ given by

$$
T_i^{i-1} = \begin{pmatrix} R_i^{i-1} & d \\ 0 & 1 \end{pmatrix}
$$

Where d is the position vector from the origin of frame $\{i-1\}$ to the origin of frame $\{i\}$ expressed in frame $\{i-1\}$, and R_i^{i-1} is the rotation matrix.

By direct inspection of the above diagram the following transformations are obtained

$$T_1^0 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_2^0 = \begin{pmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_2^1 = \begin{pmatrix} 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Given the above transformations $T_1^0 T_2^1$ is found and checked to be the same as T_2^0

$$T_1^0 T_2^1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Which is the same as T_2^0 as expected.

2.1.3 Problem 3

(3) [Spong 2-39] Consider the diagram below. A robot is set up 1 meter from a table. The table top is 1 meter high and 1 meter square. A frame $o_1x_1y_1z_1$ is fixed to the edge of the table as shown. A cube measuring 20 cm on a side is placed in the center of the table with the frame $o_2x_2y_2z_2$ established at the center of the cube as shown. A camera is situated directly above the center of the block 2 meters above the table top with frame $o_3x_3y_3z_3$ attached as shown. Find the homogeneous transformations relating each of these frames to the base frame $o_0x_0y_0z_0$.

► Find the homogeneous transformation relating the frame $o_2x_2y_2z_2$ to the camera frame $o_3x_3y_3z_3$.

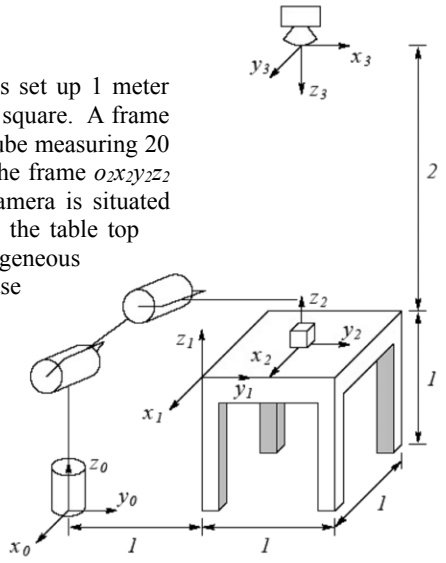


Figure 2.2: problem 3 description

The distance from the table surface to the center of the small cube is 0.1 meter. The goal is to determine T_1^0, T_2^0, T_3^0 and T_3^2. By direct inspection of the given figure the following

transformations are obtained

$$T_1^0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_2^0 = \begin{pmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & 1.5 \\ 0 & 0 & 1 & 1.1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_3^0 = \begin{pmatrix} 0 & 1 & 0 & -0.5 \\ 1 & 0 & 0 & 1.5 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In each of the above, the first column of T_i^{i-1} is the projection of $\hat{x}$ in frame i into frame $i-1$ and the second is the projection of $\hat{y}$ in frame i into frame $i-1$ and the third column the projection of $\hat{z}$ in frame i into frame $i-1$. The fourth column of T is the position vector of the center of frame i expressed in frame $i-1$. By inspection T_3^2 is found to be

$$T_3^2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1.9 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.1.4 Problem 4

(4) Coordinate frames {A} and {B} are fixed with respect to ground and are related by the homogeneous transformation matrix

$$T_B^A = \begin{bmatrix} -\frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} & -2 \\ 0 & 1 & 0 & 1 \\ \frac{\sqrt{3}}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The velocity of a point expressed in frame {A} is given as

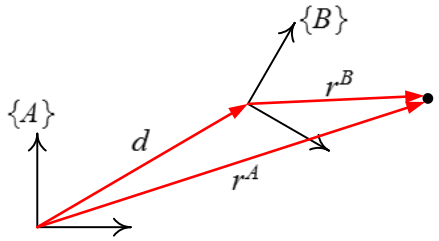
$$v^A = \begin{bmatrix} -2 & 4 & 2 \end{bmatrix}^T$$

- Evaluate the velocity of the point expressed in frame {B}, v^B
- Calculate the magnitude of v^A and v^B. Are they equal and why?

Figure 2.3: problem 4 description

PART 1:

Let d be the position vector of the origin of frame $\{B\}$ relative to frame $\{A\}$. Let r^A be the position vector of a point relative to frame $\{A\}$, and r^B be the position vector of the point relative to frame $\{B\}$ as shown in the following diagram



From the above diagram r^A is found as

$$r^A = R_B^A r^B + d \tag{1}$$

From the problem statement, $R_B^A = \begin{pmatrix} \frac{-1}{2} & 0 & \frac{-\sqrt{3}}{2} \\ 0 & 1 & 0 \\ \frac{\sqrt{3}}{2} & 0 & \frac{-1}{2} \end{pmatrix}$ and $d = \begin{pmatrix} -2 \\ 4 \\ 2 \end{pmatrix}$

Taking time derivative of (1) and using the chain rule results in

$$v^A = \frac{dR_B^A}{dt} r^B + R_B^A v^B + \frac{d}{dt} d \tag{2}$$

R_B^A does not depend on time, therefore $\frac{dR_B^A}{dt} = 0$. Since frame $\{B\}$ does not move relative to frame $\{A\}$, therefore $\frac{d}{dt} d = 0$. Using these results (2) simplifies to

$$v^A = R_B^A v^B$$

Solving for v^B from the above, and noting that $\left(R_B^A\right)^{-1} = \left(R_B^A\right)^T$ since it is a rotation matrix gives

$$v^B = \left(R_B^A\right)^T v^A$$

Substituting the values given in the problem in the above results in

$$\begin{aligned} v^B &= \begin{pmatrix} \frac{-1}{2} & 0 & \frac{-\sqrt{3}}{2} \\ 0 & 1 & 0 \\ \frac{\sqrt{3}}{2} & 0 & \frac{-1}{2} \end{pmatrix}^T \begin{pmatrix} -2 \\ 4 \\ 2 \end{pmatrix} \\ &= \begin{pmatrix} -\frac{1}{2} & 0 & \frac{1}{2}\sqrt{3} \\ 0 & 1 & 0 \\ -\frac{1}{2}\sqrt{3} & 0 & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} -2 \\ 4 \\ 2 \end{pmatrix} \end{aligned}$$

Therefore

$$v^B = \begin{pmatrix} \sqrt{3}+1 \\ 4 \\ \sqrt{3}-1 \end{pmatrix}$$

PART 2:

The norm of the velocity vectors are

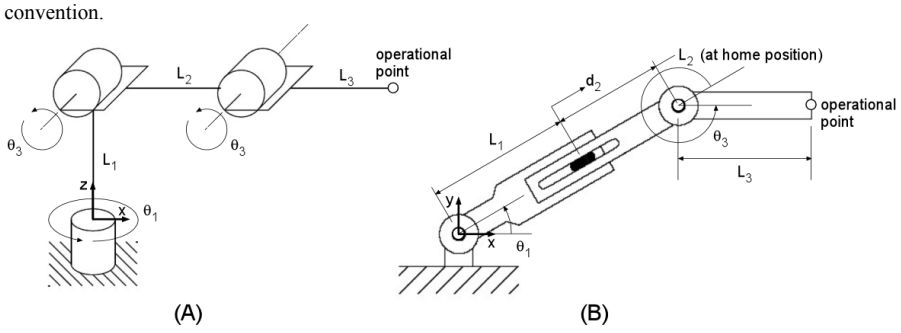
$$\left\| v^A \right\| = \left\| \begin{pmatrix} -2 \\ 4 \\ 2 \end{pmatrix} \right\| = 2\sqrt{6}$$

$$\left\| v^B \right\| = \begin{pmatrix} \sqrt{3}+1 \\ 4 \\ \sqrt{3}-1 \end{pmatrix} = 2\sqrt{6}$$

They have the same magnitude. The reason is that frame $\{B\}$ itself does not move nor rotate relative to $\{A\}$. Therefore frame B is fixed relative from frame $\{A\}$. Hence the velocity of a point relative to frame $\{A\}$ will have the same magnitude relative to $\{B\}$. The velocity vector has different representation depending on the frame of reference, but has the same magnitude.

2.1.5 Problem 5

(5) For the two manipulators shown below derive the forward kinematics equations using the DH convention.



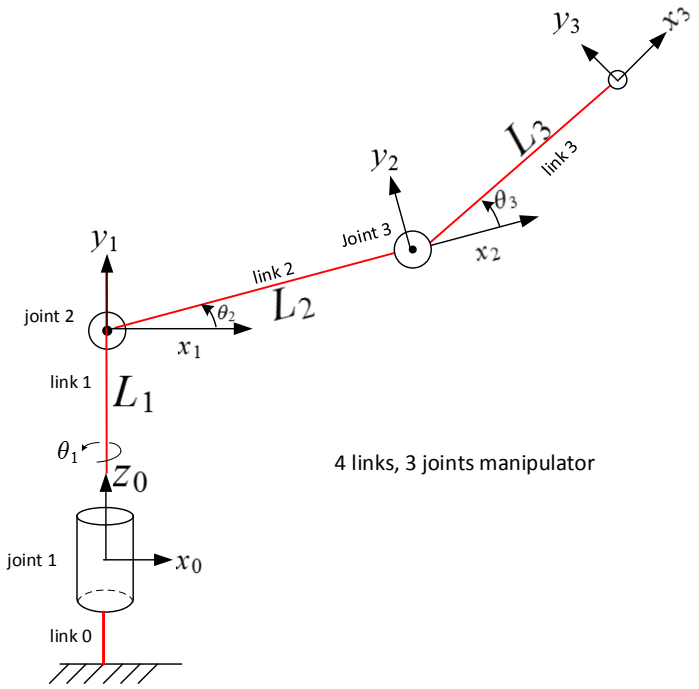
- Assign frames per the DH convention and build the DH table for each manipulator
- Derive the homogeneous transforms that relate successive frames (i.e. frame {i} to {i-1}) as a function of the joint variables and manipulator geometric parameters.
- Derive the homogeneous transform that relates the position and orientation of the end-effector to the base frame as a function of the joint variables and manipulator geometric parameters. Use the base frame and end-effector operation point defined in the figures.

Figure 2.4: problem 5 description

solution

PART (A)

The first step is to assign the z_i axes for each link as follows



The four Denavit-Hartenberg parameters are defined as follows[1]

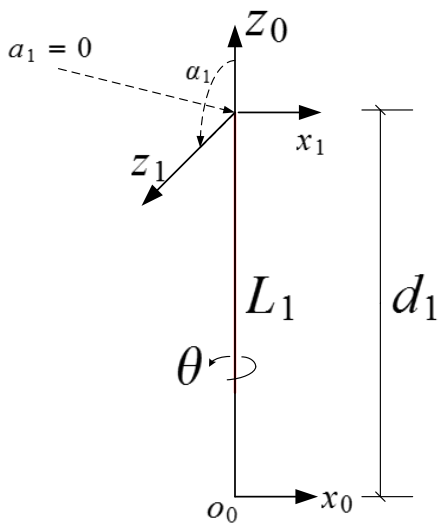
1. a_i (link length). The distance between axis z_{i-1} and z_i measured along x_i.
2. α_i (link twist angle). The angle between z_{i-1} and z_i measured in a plane normal to x_i using the right hand rule, around x_i (not x_{i-1}) to determine the positive sense of this angle.
3. d_i (link offset). The distance from origin o_{i-1} to the intersection of the x_i axis with z_{i-1} measured along z_{i-1} axis.
4. θ_i (Joint angle). The angle from x_{i-1} to x_i measured in plane normal to z_{i-1}.

When assigning the frames using the above rules, we need to insure[2] that $x_{i+1} \perp z_i$ and x_{i+1} intersects z_i. Using the above rules the DH table is written down. There is one row in the table for each link. Hence there will be three rows. Link 0 is the base link and attached to the ground and does not show in the table.

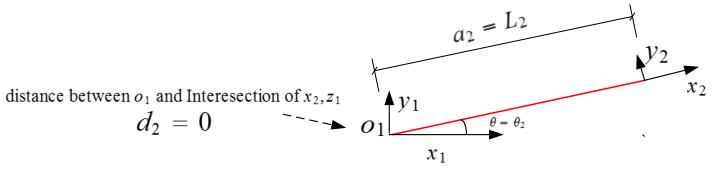
	a (link length)	α (link twist angle)	d (link offset)	θ (Joint angle)
link 1	0	90^0	L_1	θ_1
link 2	L_2	0	0	θ_2
link 3	L_3	0	0	θ_3

[1]Textbook, page 80.
[2]Text book, page 78.

The following diagram shows the parameters for the first link

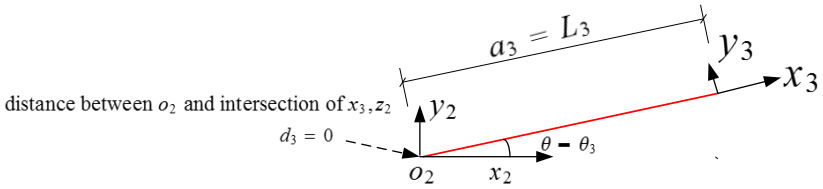


The following diagram shows the parameters for the second link



since $z_1 \| z_1$ then $\alpha_2 = 0$

The following diagram shows the parameters for the third link



since $z_3 \| z_2$ then $\alpha_3 = 0^0$

Now the forward transformations using equation (3.10) on page 77 of the textbook is found

$$A = \begin{pmatrix} C_\theta & -S_\theta C_\alpha & S_\theta S_\alpha & aC_\theta \\ S_\theta & C_\theta C_\alpha & -C_\theta S_\alpha & aS_\theta \\ 0 & S_\alpha & C_\alpha & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hence

$$A_1 = \begin{pmatrix} \cos\theta_1 & -\sin\theta_1\cos\frac{\pi}{2} & \sin\theta_1\sin\frac{\pi}{2} & 0 \\ \sin\theta_1 & \cos\theta_1\cos\frac{\pi}{2} & -\cos\theta_1\sin\frac{\pi}{2} & 0 \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} \cos\theta_2 & -\sin\theta_2\cos 0 & \sin\theta_2\sin 0 & L_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2\cos 0 & -\cos\theta_2\sin 0 & L_2\sin\theta_2 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & L_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & L_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3\cos 0 & \sin\theta_3\sin 0 & L_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3\cos 0 & -\cos\theta_3\sin 0 & L_3\sin\theta_3 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & L_3\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Using the above, T_3^0 is found

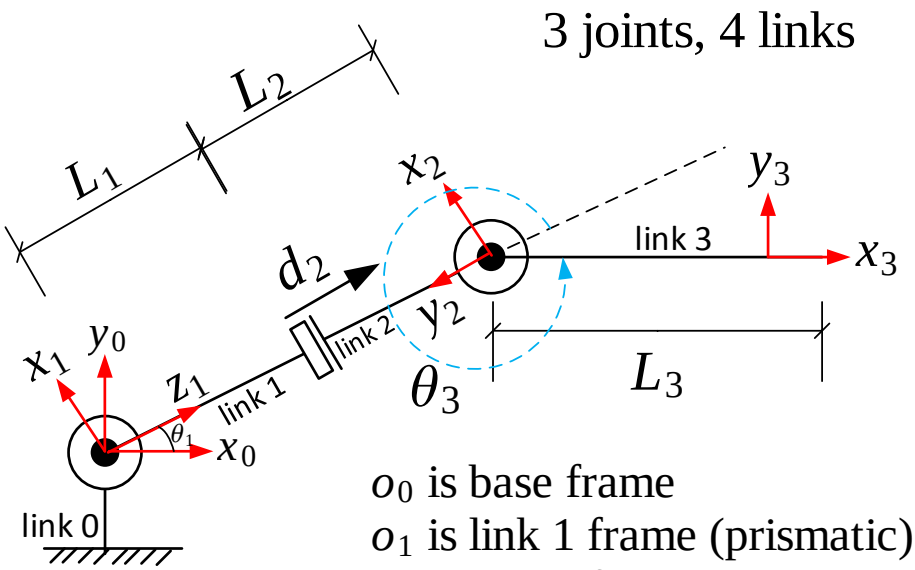
$$\begin{aligned} T_3^0 &= A_1A_2A_3 \\ &= \begin{pmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & L_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & L_2\sin\theta_2 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & L_3\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

For verification of the above, let $\theta_1 = 0, \theta_2 = 0, \theta_3 = 0$ then the fourth column of T_3^0 gives the position vector of the end effector relative to the base when the manipulator is in the position in the problem. Substituting these values for the angles gives

$$T_3^0 = \begin{pmatrix} 1 & 0 & 0 & L_2 + L_3 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The above says that the end effector is at position vector $p^0 = (L_2 + L_3, 0, L_1)$ which means $x_3 = L_2 + L_3, y_3 = 0, z_3 = L_1$. From the diagram this result is correct.

PART (B) For the second manipulator, the same steps were repeated. The first step is to assign the axes for each link as follows



The DH table is written down

	a (link length)	α (link twist angle)	d (link offset)	θ (Joint angle)
link 1	0	90^0	0	$\theta_1 + 90^0$
link 2	0	-90^0	$L_1 + L_2 + d_2$	0
link 3	L_3	0	0	$\theta_3 - 90^0$

The forward transformations using equation (3.10) on page 77 of the textbook gives

$$A = \begin{pmatrix} C_\theta & -S_\theta C_\alpha & S_\theta S_\alpha & aC_\theta \\ S_\theta & C_\theta C_\alpha & -C_\theta S_\alpha & aS_\theta \\ 0 & S_\alpha & C_\alpha & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Using the above, and noting that $\cos(x + 90^0) = -\sin x$,$\sin(x + 90^0) = \cos x$,$\cos(x - 90^0) = \sin x$ and $\sin(x - 90^0) = -\cos(x)$ results in

$$A_1 = \begin{pmatrix} \cos(\theta_1+90) & -\sin(\theta_1+90)\cos\frac{\pi}{2} & \sin(\theta_1+90)\sin\frac{\pi}{2} & 0 \\ \sin(\theta_1+90) & \cos(\theta_1+90)\cos\frac{\pi}{2} & -\cos(\theta_1+90)\sin\frac{\pi}{2} & 0 \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & 0 \\ \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} \cos 0 & -\sin 0\cos(-90^0) & \sin 0\sin(-90^0) & 0\cos 0 \\ \sin 0 & \cos 0\cos(-90^0) & -\cos 0\sin(-90^0) & 0\sin 0 \\ 0 & \sin(-90^0) & \cos(-90^0) & L_1+L_2+d_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & L_1+L_2+d_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} \cos(\theta_3-90^0) & -\sin(\theta_3-90^0)\cos 0 & \sin(\theta_3-90^0)\sin 0 & L_3\cos(\theta_3-90^0) \\ \sin(\theta_3-90^0) & \cos(\theta_3-90^0)\cos 0 & -\cos(\theta_3-90^0)\sin 0 & L_3\sin(\theta_3-90^0) \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \sin\theta_3 & \cos\theta_3 & 0 & L_3\sin\theta_3 \\ -\cos\theta_3 & \sin\theta_3 & 0 & -L_3\cos\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

From the above

$$\begin{aligned} T_3^0 &= A_1A_2A_3 \\ &= \begin{pmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & 0 \\ \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & L_1+L_2+d_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \sin\theta_3 & \cos\theta_3 & 0 & L_3\sin\theta_3 \\ -\cos\theta_3 & \sin\theta_3 & 0 & -L_3\cos\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos\theta_1\cos\theta_3-\sin\theta_1\sin\theta_3 & -\cos\theta_1\sin\theta_3-\cos\theta_3\sin\theta_1 & 0 & \cos\theta_1(L_1+L_2+d_2)+L_3\cos\theta_1\cos\theta_3-L_3\sin\theta_1\sin\theta_3 \\ \cos\theta_1\sin\theta_3+\cos\theta_3\sin\theta_1 & \cos\theta_1\cos\theta_3-\sin\theta_1\sin\theta_3 & 0 & \sin\theta_1(L_1+L_2+d_2)+L_3\cos\theta_1\sin\theta_3+L_3\cos\theta_3\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

To verify the above, let $\theta_1 = 0, \theta_3 = 360^0, d_2 = 0$ then the fourth column of T_3^0 gives the position vector of the end effector relative to the base when the manipulator is in a straight horizontal position

$$T_3^0 = \begin{pmatrix} 1 & 0 & 0 & L_1+L_2+L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

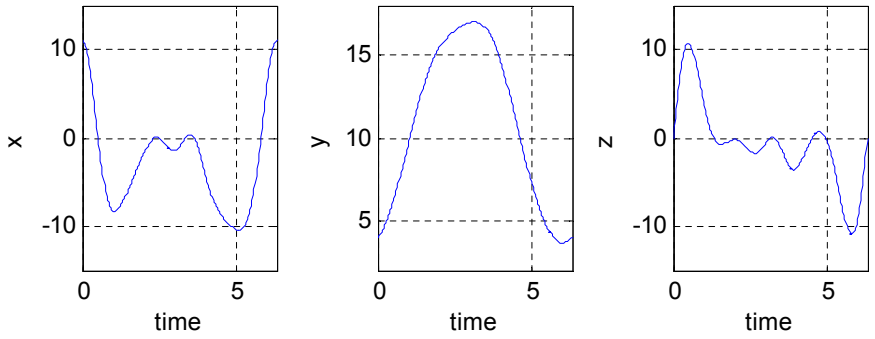
The above results show that $x_3 = L_1 + L_2 + L_3$ which is the expected result.

2.1.6 Problem 6

problem description

- Write Matlab code to plot the position (x, y, and z coordinates) of the end-effector (point E) as a function of time. Your plots should match the plots shown below. The joint motion, as a function of time, is given below. Make sure to include all supporting functions, including any custom plotting routines, with your homework submission. Your homework submission must provide clear, easy instructions to run you Matlab code

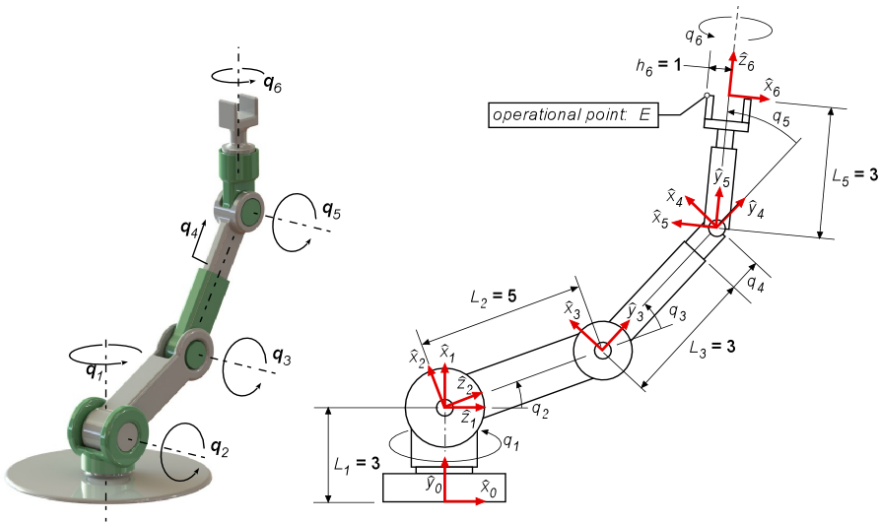
  Joint variable inputs:
  $q_1 = -\pi\sin(t),\ q_2 = \frac{\pi}{4}(1-\cos(t)),\ q_3 = \frac{\pi}{4}\sin(t),\ q_4 = \frac{1}{2}L_3(1-\cos(t)),\ q_5 = -\frac{\pi}{4}\sin(t),\ q_6 = \frac{\pi}{4}\sin(t)$
  Please animate the system over the time interval $t = 0 : 2\pi$.



- Write Matlab code to animate the manipulator using the Matlab scripts provided on the Learn@UW course page. The joint motion, as a function of time, is given above. Make sure to include all supporting functions, including any custom plotting routines, with your homework submission. Your homework submission must provide clear, easy instructions to run you Matlab code. To maintain consistency, please use the following rendering window view parameters.

```
%----set rendering window view parameters

% figure handle
f_handle = 1;

% axis limits
axis_limits = [-10 10 0 10 -10 10];

% camera position
render_view = [-1 1 -1];

% vertical orientation
view_up = [0 1 0];

% initialize rendering view
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
```

(6) For the manipulator shown in the adjacent figure



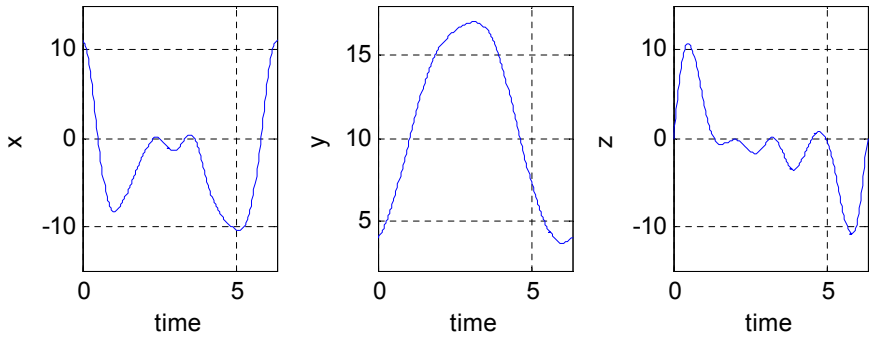
► Derive the homogeneous transforms that relate the successive frames as a function of the joint variable and manipulator geometric parameters (i.e. find: $T_1^0, T_2^1, T_3^2, T_4^3, T_5^4, T_6^5$). Use the coordinate frames as defined in the figure. Note that the frames are fixed to the links at various locations (e.g. proximal end, distal end). Assume that q_1 equals zero when the manipulator lies in the plane of the page. Attendee

► Write Matlab code to calculate the homogeneous transforms derived above as a function of the joint variables q_1, q_2, q_3, q_4, and q_5.

► Write Matlab code to calculate the homogeneous transformation matrices that describe the frame displacements relative to the ground frame {0} (i.e. numerically evaluate $T_1^0, T_2^0, T_3^0, T_4^0, T_5^0, T_6^0$).

► Write Matlab code to plot the position (x, y, and z coordinates) of the end-effector (point E) as a function of time. Your plots should match the plots shown below. The joint motion, as a function of time, is given below. Make sure to include all supporting functions, including any custom plotting routines, with your homework submission. Your homework submission must provide clear, easy instructions to run you Matlab code

Joint variable inputs:
$q_1 = -\pi \sin(t),\ q_2 = \frac{\pi}{4}(1-\cos(t)),\ q_3 = \frac{\pi}{4}\sin(t),\ q_4 = \frac{1}{2}L_3(1-\cos(t)),\ q_5 = -\frac{\pi}{4}\sin(t),\ q_6 = \frac{\pi}{4}\sin(t)$
Please animate the system over the time interval $t = 0 : 2\pi$.



► Write Matlab code to animate the manipulator using the Matlab scripts provided on the Learn@UW course page. The joint motion, as a function of time, is given above. Make sure to include all supporting functions, including any custom plotting routines, with your homework submission. Your homework submission must provide clear, easy instructions to run you Matlab code. To maintain consistency, please use the following rendering window view parameters.

```
%----set rendering window view parameters

% figure handle
f_handle = 1;

% axis limits
axis_limits = [-10 10 0 10 -10 10];

% camera position
render_view = [-1 1 -1];

% vertical orientation
view_up = [0 1 0];

% initialize rendering view
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
```

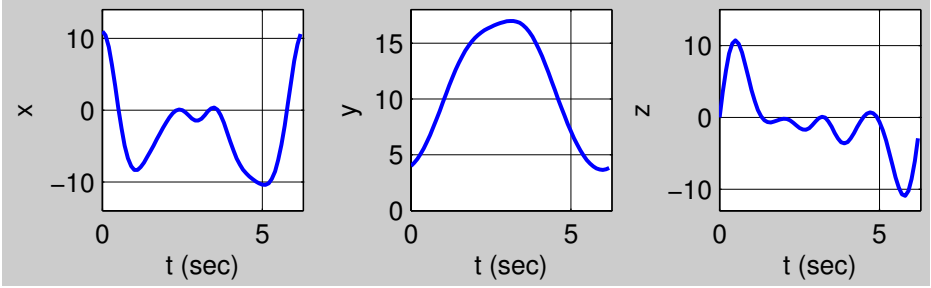
solution

The homogeneous transformation T_i^{i-1} was derived by inspection giving the following results

$$T_1^0 = \begin{pmatrix} 0 & \sin q_1 & \cos q_1 & 0 \\ 1 & 0 & 0 & L_1 \\ 0 & \cos q_1 & -\sin q_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_2^1 = \begin{pmatrix} \cos q_2 & 0 & \sin q_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin q_2 & 0 & \cos q_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_3^2 = \begin{pmatrix} \cos q_3 & \sin q_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -\sin q_3 & \cos q_3 & 0 & L_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_4^3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_3 + q_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_5^4 = \begin{pmatrix} \cos q_5 & \sin q_5 & 0 & 0 \\ -\sin q_5 & \cos q_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_6^5 = \begin{pmatrix} -\cos q_6 & \sin q_6 & 0 & 0 \\ 0 & 0 & 1 & L_5 \\ \sin q_6 & \cos q_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The following is plot of the x, y, z coordinates of the end effector E



The following Matlab script `problem_6_part_1.m` calculates the homogeneous transformation T_6^0 and plots the above figures

```
%This scripts plots the x,y,z coordinates of the end effector E
%for problem 5, HW1 , ME 739.
%to run, type this script name on the Matlab console
%    problem_6_part_1
%The matlab path must include the ME 739 rendering software
%Nasser M. Abbasi 2/16/15

clear all; close all;
%define syms to use to build the T matrices
syms q1 q2 q3 q4 q5 q6 L1 L2 L3 L5 t
L1 = 3;
L2 = 5;
L3 = 3;
L5 = 3;
h6 = 1;
q1 = -pi*sin(t);
q2 = pi/4*(1-cos(t));
q3 = pi/4*sin(t);
q4 = 1/2*L3*(1-cos(t));
q5 = -pi/4*sin(t);
q6 = pi/4*sin(t);
```

```
%define the 6 transformation matrices T01 to T56 in syms
T01 = [0        sin(q1) cos(q1)   0;
       1        0       0         L1;
       0        cos(q1) -sin(q1)  0;
       0        0       0         1];

T12 = [cos(q2)  0       sin(q2)  0;
       0        1       0        0;
       -sin(q2) 0       cos(q2)  0;
       0        0       0        1];

T23 = [cos(q3)  sin(q3) 0        0;
       0        0       -1       0;
       -sin(q3) cos(q3) 0        L2;
       0        0       0        1];

T34 = [1        0       0        0;
       0        1       0        L3+q4;
       0        0       1        0;
       0        0       0        1];

T45 = [cos(q5)  sin(q5) 0        0;
       -sin(q5) cos(q5) 0        0;
       0        0       1        0;
       0        0       0        1];

T56 = [-cos(q6) sin(q6) 0        0;
       0        0       1        L5;
       sin(q6)  cos(q6) 0        0;
       0        0       0        1];

%Now obtain T06 to allow finding the end effector coordinates
T06 = T01*T12*T23*T34*T45*T56;

%handle to function to evaluate T06 at each instance of time
endPos = @(t0) subs(T06,t,t0)

%set up time scale, and evaluate the end effector coordinates
%saving result in a matrix for plotting later.
timeScale = 0:.1:2*pi;
coords    = zeros(length(timeScale),3);

%generate the coordinates of the end effector
for i = 1:length(timeScale)
    p = endPos(timeScale(i))*[-h6 0 0 1]';
    coords(i,:) = p(1:3);
```

```
end

%now plot the result. First col is the x-coordinates,
%second col is y-coord, and third col is z-coordinate.
subplot(1,3,1);
plot(timeScale,coords(:,1), 'LineWidth',1.5);
xlabel('t (sec)'); ylabel('x');
grid; axis square; xlim([0 2*pi]);ylim([-14 14]);
set(gca, 'GridLineStyle', '-');

subplot(1,3,2);
plot(timeScale,coords(:,2), 'LineWidth',1.5);
xlabel('t (sec)'); ylabel('y');
grid; axis square; xlim([0 2*pi]);ylim([0 18]);
%axis square; axis tight
set(gca, 'GridLineStyle', '-');

subplot(1,3,3);
plot(timeScale,coords(:,3),'LineWidth',1.5);
xlabel('t (sec)'); ylabel('z');
grid; axis square; xlim([0 2*pi]);ylim([-13 15]);
set(gca, 'GridLineStyle', '-');
%export_fig(gcf,'problem_6_part_1.pdf');
```

The following matlab script calculates $T_1^0, T_2^0, T_3^0, T_4^0, T_5^0, T_6^0$, where $T_2^0 = T_1^0 T_2^1, T_3^0 = T_2^0 T_3^2, T_4^0 = T_3^0 T_4^3, T_5^0 = T_4^0 T_5^4, T_6^0 = T_5^0 T_6^5$

For example, for T_2^0 the result is

$$T_1^0 = \begin{pmatrix} 0 & \sin q_1 & \cos q_1 & 0 \\ 1 & 0 & 0 & L_1 \\ 0 & \cos q_1 & -\sin q_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2^0 = T_1^0 T_2^1 = \begin{pmatrix} 0 & \sin q_1 & \cos q_1 & 0 \\ 1 & 0 & 0 & L_1 \\ 0 & \cos q_1 & -\sin q_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos q_2 & 0 & \sin q_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin q_2 & 0 & \cos q_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -\cos q_1 \sin q_2 & \sin q_1 & \cos q_1 \cos q_2 \\ \cos q_2 & 0 & \sin q_2 \\ \sin q_1 \sin q_2 & \cos q_1 & -\cos q_2 \sin q_1 \\ 0 & 0 & 0 \end{pmatrix}$$

The complete calculation was done in the following Matlab script `problem_6_part_2.m`. The script is run by typing its name on the Matlab console.

```
%This calculates T01,T02,T03,T04,T05,T06 numerically
%for problem 5, HW1 , ME 739.
%to run, just type this script name on the Matlab console
%    problem_6_part_2
```

```
%
%The matlab path must include the ME 739 rendering software
%Nasser M. Abbasi 2/16/15

clear all; close all;
%define syms to use to build the T matrices
syms q1 q2 q3 q4 q5 q6 L1 L2 L3 L5 t
L1 = 3;
L2 = 5;
L3 = 3;
L5 = 3;
h6 = 1;
q1 = -pi*sin(t);
q2 = pi/4*(1-cos(t));
q3 = pi/4*sin(t);
q4 = 1/2*L3*(1-cos(t));
q5 = -pi/4*sin(t);
q6 = pi/4*sin(t);

%define the 6 transformation matrices T01 to T56 in syms
%define the 6 transformation matrices T01 to T56 in syms
T01 = [0        sin(q1) cos(q1)   0;
       1        0       0         L1;
       0        cos(q1) -sin(q1)  0;
       0        0       0         1];

T12 = [cos(q2)  0       sin(q2)  0;
       0        1       0        0;
       -sin(q2) 0       cos(q2)  0;
       0        0       0        1];

T23 = [cos(q3)  sin(q3) 0        0;
       0        0       -1       0;
       -sin(q3) cos(q3) 0        L2;
       0        0       0        1];

T34 = [1        0       0        0;
       0        1       0        L3+q4;
       0        0       1        0;
       0        0       0        1];

T45 = [cos(q5)  sin(q5) 0        0;
       -sin(q5) cos(q5) 0        0;
       0        0       1        0;
       0        0       0        1];

T56 = [-cos(q6) sin(q6) 0        0;
```

```
       0        0       1        L5;
       sin(q6)  cos(q6) 0        0;
       0        0       0        1];

%Now calculate T02,T03,T04,T05,T06
T02 = T01*T12;
T03 = T02*T23;
T04 = T03*T34;
T05 = T04*T45;
T06 = T05*T56;

%handle to function to evaluate each Tij at each instance of time
calcT = @(T,t0) double(subs(T,t,t0));

%now calculate all the T's at some specific time. The problem
%does not says what time instance to use, so we use t=0 for
%illustration

timeToShow = 1; %change this to different time as needed

fprintf('T01 at t=1 is \n'); calcT(T01,timeToShow)
fprintf('T02 at t=1 is \n'); calcT(T02,timeToShow)
fprintf('T03 at t=1 is \n'); calcT(T03,timeToShow)
fprintf('T04 at t=1 is \n'); calcT(T04,timeToShow)
fprintf('T05 at t=1 is \n'); calcT(T05,timeToShow)
fprintf('T06 at t=1 is \n'); calcT(T06,timeToShow)
```

The above script calculates numerically all the transformation matrices using the joint variable inputs given in the problem. At the end it prints each matrix. The problem did not indicate for which value of t to use to calculate the matrices, hence for illustration these are displayed for $t = 0$ and $t = 1$ second. A variable inside the script can be used to change the time instance. The following is the output from running the above script for illustration

At $t = 0$ the output is

```
T01 at t=0 is
     0     0     1     0
     1     0     0     3
     0     1     0     0
     0     0     0     1

T02 at t=0 is
     0     0     1     0
     1     0     0     3
     0     1     0     0
     0     0     0     1

T03 at t=0 is
     0     1     0     5
     1     0     0     3
     0     0    -1     0
     0     0     0     1

T04 at t=0 is
     0     1     0     8
     1     0     0     3
     0     0    -1     0
     0     0     0     1

T05 at t=0 is
     0     1     0     8
     1     0     0     3
     0     0    -1     0
     0     0     0     1

T06 at t=0 is
     0     0     1    11
    -1     0     0     3
     0    -1     0     0
     0     0     0     1
```

At $t = 1$ the output is

```
T01 at t=1 is
            0      -0.4777      -0.87852            0
            1            0            0            3
            0      -0.87852       0.4777            0
            0            0            0            1

T02 at t=1 is
      0.31034      -0.4777      -0.82188            0
      0.93553            0      0.35325            3
     -0.16875      -0.87852       0.4469            0
            0            0            0            1

T03 at t=1 is
      0.74949      -0.45834       0.4777      -4.1094
      0.52172       0.85312            0       4.7663
     -0.40753       0.24922      0.87852       2.2345
            0            0            0            1

T04 at t=1 is
      0.74949      -0.45834       0.4777      -5.8005
      0.52172       0.85312            0       7.9139
     -0.40753       0.24922      0.87852        3.154
            0            0            0            1

T05 at t=1 is
      0.31034      -0.82188       0.4777      -5.8005
      0.93553       0.35325            0       7.9139
     -0.16875        0.4469      0.87852        3.154
            0            0            0            1

T06 at t=1 is
     0.048223       0.56761      -0.82188      -8.2661
     -0.73855       0.57425      0.35325       8.9736
      0.67247       0.58997       0.4469       4.4947
            0            0            0            1
```

The manipulator was animated using the UW software. The following script `problem_6_part_3.m` written for this purpose. Typing the name of the script in Matlab starts the animation.

This script `problem_6_part_3.m` assumes the Matlab path is set to include the UW rendering software.

```
%This calculates T01,T02,T03,T04,T05,T06 numerically
%for problem 5, HW1 , ME 739.
%to run, just type this script name on the Matlab console
%
%    problem_6_part_3
%The matlab path must include the ME 739 rendering software
%
%Nasser M. Abbasi 2/17/15
```

```
clear all; close all; clc;

%----set rendering window view parameters
f_handle = 1; % figure handle
axis_limits = [-10 10 0 13 -10 10]; %needed little bit more space
render_view = [-1 1 -1]; % camera position
view_up = [0 1 0]; % vertical orientation
% initialize rendering view
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);

ADD_BASE = false; %set to TRUE to see base rendered, does not move.
DO_MOVIE = false; %set true to make frames for movie
ANIMATION_TIME = 10; %10 seconds to animation
DEL_T          = 0.05; %time betweeb each animation loop. smaller time
                       %make it run slower but more accurate

%define syms to use to build the T matrices
syms q1 q2 q3 q4 q5 q6 L1 L2 L3 L5 t
L1 = 3;
L2 = 5;
L3 = 3;
L5 = 3;
h6 = 1;
q1 = -pi*sin(t);
q2 = pi/4*(1-cos(t));
q3 = pi/4*sin(t);
q4 = 1/2*L3*(1-cos(t));
q5 = -pi/4*sin(t);
q6 = pi/4*sin(t);

%define the 6 transformation matrices T01 to T56 in syms
%define the 6 transformation matrices T01 to T56 in syms
T01 = [0        sin(q1) cos(q1)   0;
       1        0       0         L1;
       0        cos(q1) -sin(q1)  0;
       0        0       0         1];

T12 = [cos(q2)  0       sin(q2)  0;
       0        1       0        0;
       -sin(q2) 0       cos(q2)  0;
       0        0       0        1];

T23 = [cos(q3)  sin(q3) 0        0;
       0        0       -1       0;
       -sin(q3) cos(q3) 0        L2;
       0        0       0        1];
```

```
T34 = [1        0       0        0;
       0        1       0        L3+q4;
       0        0       1        0;
       0        0       0        1];

T45 = [cos(q5)  sin(q5) 0        0;
       -sin(q5) cos(q5) 0        0;
       0        0       1        0;
       0        0       0        1];

T56 = [-cos(q6) sin(q6) 0        0;
       0        0       1        L5;
       sin(q6)  cos(q6) 0        0;
       0        0       0        1];

   %Now calculate T02,T03,T04,T05,T06
T02 = T01*T12;
T03 = T02*T23;
T04 = T03*T34;
T05 = T04*T45;
T06 = T05*T56;

%handle to function to evaluate each Tij at each instance of time
%this is called during running the animation
calcT = @(T,t0) double(subs(T,t,t0));

%base, does not move
if ADD_BASE
    linkColor = [0 0 0]; plotFrame=0; normalized_location=-1;
    nSides = 4; radius = 4; r = L1;  axis_aligned = 2;
    d0 = CreateLinkRendering(r ,radius, nSides, axis_aligned ,normalized_location, ...
        linkColor,plotFrame, f_handle);
end

%now create all the links
linkColor = [1 .1 0]; plotFrame=0; normalized_location=-1;
nSides=20; radius=2; r=L1;  axis_aligned = 1;
d1 = CreateLinkRendering(r ,radius, nSides, axis_aligned ,normalized_location, ...
    linkColor,plotFrame, f_handle);

linkColor = [.5 .2 1]; plotFrame=0; normalized_location=-1;
nSides=4; r=L2;  radius=1.2; axis_aligned=3;
d2 = CreateLinkRendering(r ,radius, nSides,axis_aligned ,normalized_location, ...
    linkColor,plotFrame,  f_handle);

linkColor = [1 .1 1]; plotFrame=0; normalized_location=-1;
```

```
nSides=4; r=L3;  radius=1.15; axis_aligned=2;
d3 = CreateLinkRendering(r ,radius, nSides,axis_aligned ,normalized_location,...
    linkColor,plotFrame,f_handle);

linkColor = [0 0 1]; plotFrame=0; normalized_location=1;
nSides=4;   r=L5;  radius=.9; axis_aligned=2;
d4 = CreateLinkRendering(r ,radius, nSides,axis_aligned,normalized_location,...
    linkColor,plotFrame,  f_handle);

linkColor = [.3 .5 .3]; plotFrame=0; normalized_location=-1;
nSides=4;   r=L5;  radius=.7; axis_aligned=2;
d5 = CreateLinkRendering(r ,radius, nSides,axis_aligned ,normalized_location,...
    linkColor,plotFrame,  f_handle);

linkColor = [.5 1 .5]; plotFrame=0; normalized_location=-1;
nSides=4;   r=L5;  radius=0.5; axis_aligned=3;
d6 = CreateLinkRendering(r ,radius, nSides,axis_aligned,normalized_location,...
    linkColor,plotFrame,  f_handle);

%now start the animation
timeScale = 0: DEL_T :ANIMATION_TIME;
k         = 0; %for screen shots counting, to make movie
for i = 1:length(timeScale)
    T = calcT(T01,timeScale(i));
    UpdateLink(d1,T);

    T = calcT(T02,timeScale(i));
    UpdateLink(d2,T);

    T = calcT(T03,timeScale(i));
    UpdateLink(d3,T);

    T = calcT(T04,timeScale(i));
    UpdateLink(d4,T);

    T = calcT(T05,timeScale(i));
    UpdateLink(d5,T);

    T = calcT(T06,timeScale(i));
    UpdateLink(d6,T);
    title(sprintf('time = %3.3f (sec)',timeScale(i)));

    if DO_MOVIE
       k = k+1;
       I = getframe(gcf);
       imwrite(I.cdata, sprintf('frame%d.png',k));
    end
```

```
    %p=T*[-h6 0 0 1]';  to show the end effector path if needed
    %plot3(p(1),p(2),p(3),'o');
    drawnow
end
```

The following is a movie of the first few seconds of the Matlab run. (posted as animation gif in my site).

2.1.7 key solution for HW 1

-SOLUTION-

ME / ECE 739: Advanced Robotics Homework #1

Due: February 19th (Thursday)

Please submit your answers to the questions and all supporting work including your Matlab scripts, and, where appropriate, program results (plots, explanations). Your Matlab scripts should be readable, with comments, sensible variable names, indentation of code-block, etc. In addition to the hardcopy (pdf format), you must also submit your Matlab scripts electronically to the Learn@UW course page dropbox (e.g. Homework #1) using a zip archive file format. Please name your zip files using your last name and hw# (e.g. zinn_hw1.zip)

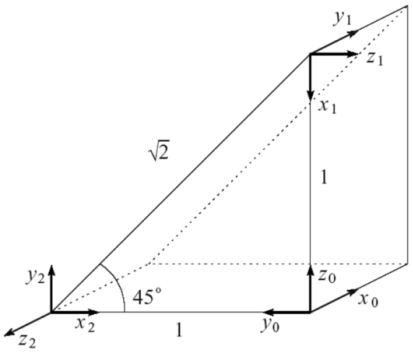
(1) [Spong 2-15] Suppose that three coordinate frames $o_1x_1y_1z_1$, $o_2x_2y_2z_2$, and $o_3x_3y_3z_3$ are given, and suppose

$$R_2^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & -\sqrt{3}/2 \\ 0 & \sqrt{3}/2 & 1/2 \end{bmatrix}, \quad R_3^1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

►Find the matrix R_3^2

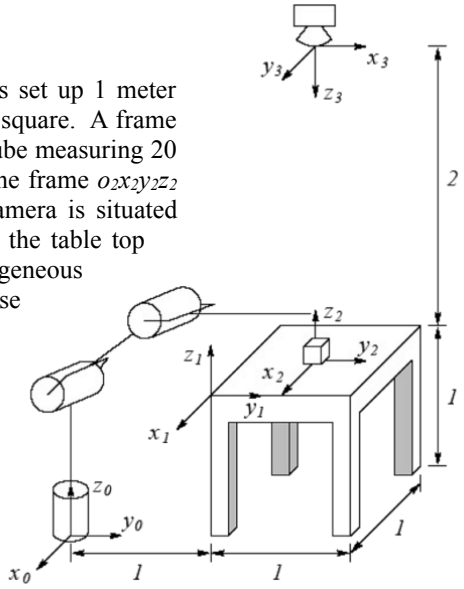
(2) [Spong 2-38] Consider the adjacent diagram. Find the homogeneous transformations T_1^0, T_2^0, T_2^1 representing the transformations among the three frames shown.

►Show that $T_2^0 = T_1^0 T_2^1$



(3) [Spong 2-39] Consider the diagram below. A robot is set up 1 meter from a table. The table top is 1 meter high and 1 meter square. A frame $o_1x_1y_1z_1$ is fixed to the edge of the table as shown. A cube measuring 20 cm on a side is placed in the center of the table with the frame $o_2x_2y_2z_2$ established at the center of the cube as shown. A camera is situated directly above the center of the block 2 meters above the table top with frame $o_3x_3y_3z_3$ attached as shown. Find the homogeneous transformations relating each of these frames to the base frame $o_0x_0y_0z_0$.

► Find the homogeneous transformation relating the frame $o_2x_2y_2z_2$ to the camera frame $o_3x_3y_3z_3$.



ME / ECE 739: Advanced Robotics Homework #1

Due: February 19th (Thursday)

(4) Coordinate frames {A} and {B} are fixed with respect to ground and are related by the homogeneous transformation matrix

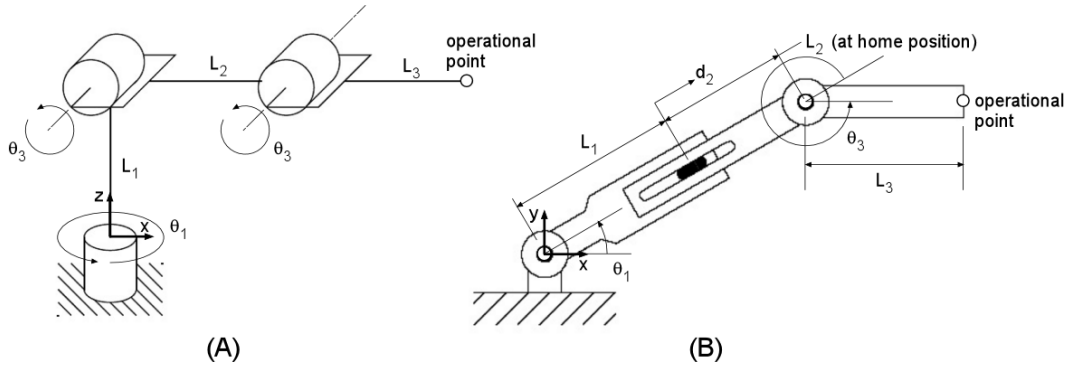
$$T_B^A = \begin{bmatrix} -\frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} & -2 \\ 0 & 1 & 0 & 1 \\ \frac{\sqrt{3}}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The velocity of a point expressed in frame {A} is given as

$$v^A = \begin{bmatrix} -2 & 4 & 2 \end{bmatrix}^T$$

- ► Evaluate the velocity of the point expressed in frame {B}, v^B
- ► Calculate the magnitude of v^A and v^B. Are they equal and why?

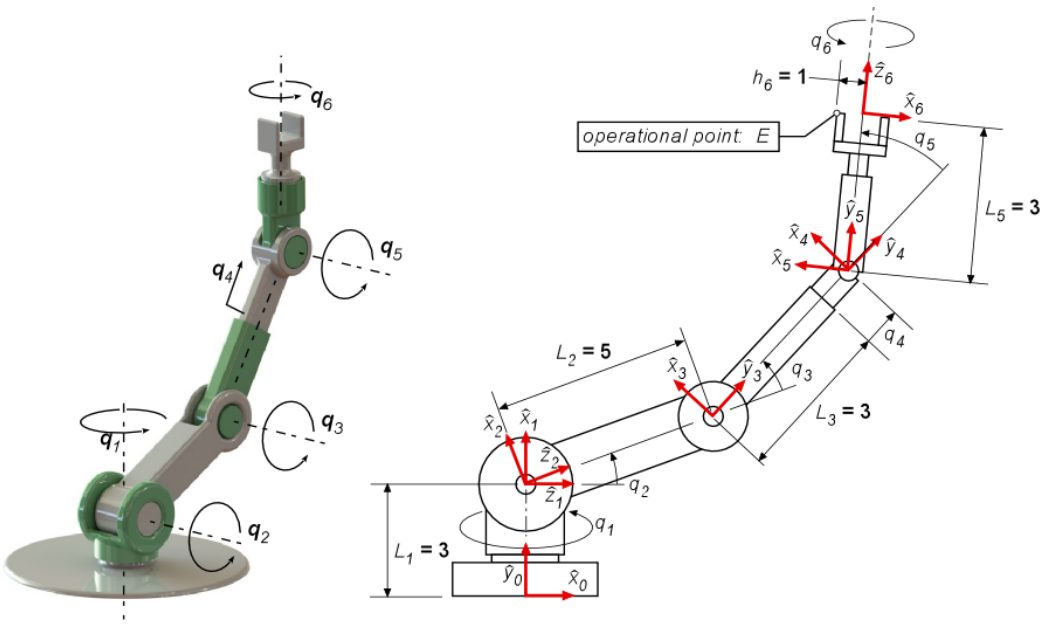
(5) For the two manipulators shown below derive the forward kinematics equations using the DH convention.



- ► Assign frames per the DH convention and build the DH table for each manipulator
- ► Derive the homogeneous transforms that relate successive frames (i.e. frame {i} to {i-1}) as a function of the joint variables and manipulator geometric parameters.
- ► Derive the homogeneous transform that relates the position and orientation of the end-effector to the base frame as a function of the joint variables and manipulator geometric parameters. Use the base frame and end-effector operation point defined in the figures.

ME / ECE 739: Advanced Robotics Homework #1
Due: February 19th (Thursday)

(6) For the manipulator shown in the adjacent figure



► Derive the homogeneous transforms that relate the successive frames as a function of the joint variable and manipulator geometric parameters (i.e. find: $T_1^0, T_2^1, T_3^2, T_4^3, T_5^4, T_6^5$). Use the coordinate frames as defined in the figure. Note that the frames are fixed to the links at various locations (e.g. proximal end, distal end). Assume that q_1 equals zero when the manipulator lies in the plane of the page. Attendee

► Write Matlab code to calculate the homogeneous transforms derived above as a function of the joint variables q_1, q_2, q_3, q_4, and q_5.

► Write Matlab code to calculate the homogeneous transformation matrices that describe the frame displacements relative to the ground frame {0} (i.e. numerically evaluate $T_1^0, T_2^0, T_3^0, T_4^0, T_5^0, T_6^0$).

(continued)

ME / ECE 739: Advanced Robotics Homework #1

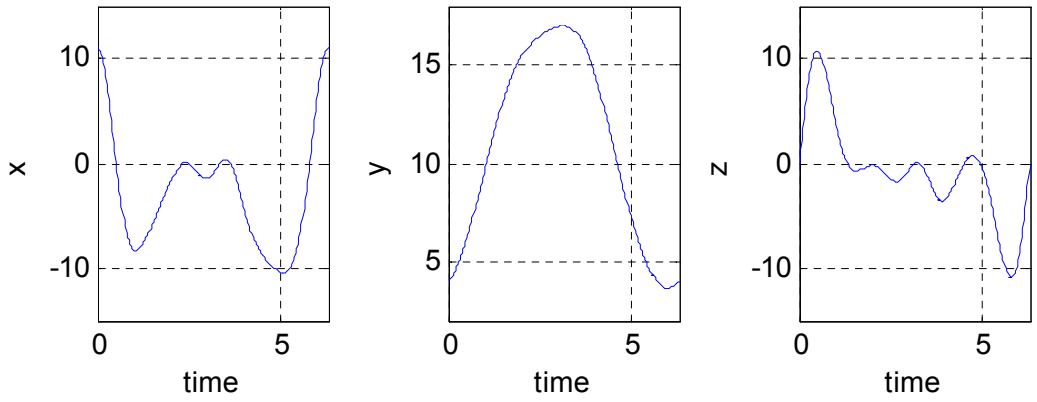
Due: February 19th (Thursday)

- Write Matlab code to plot the position (x, y, and z coordinates) of the end-effector (point E) as a function of time. Your plots should match the plots shown below. The joint motion, as a function of time, is given below. Make sure to include all supporting functions, including any custom plotting routines, with your homework submission. Your homework submission must provide clear, easy instructions to run you Matlab code

  Joint variable inputs:

  $q_1 = -\pi \sin(t),\ q_2 = \frac{\pi}{4}(1-\cos(t)),\ q_3 = \frac{\pi}{4}\sin(t),\ q_4 = \frac{1}{2}L_3(1-\cos(t)),\ q_5 = -\frac{\pi}{4}\sin(t),\ q_6 = \frac{\pi}{4}\sin(t)$

  Please animate the system over the time interval $t = 0 : 2\pi$.



- Write Matlab code to animate the manipulator using the Matlab scripts provided on the Learn@UW course page. The joint motion, as a function of time, is given above. Make sure to include all supporting functions, including any custom plotting routines, with your homework submission. Your homework submission must provide clear, easy instructions to run you Matlab code. To maintain consistency, please use the following rendering window view parameters.

```
%----set rendering window view parameters

% figure handle
f_handle = 1;

% axis limits
axis_limits = [-10 10 0 10 -10 10];

% camera position
render_view = [-1 1 -1];

% vertical orientation
view_up = [0 1 0];

% initialize rendering view
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
```

Question #1.1

$$R_2^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & -\sqrt{3}/2 \\ 0 & \sqrt{3}/2 & 1/2 \end{bmatrix}$$

find R_3^2

$$R_3^1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$R_3^2 = R_1^2 R_3^1$$

$$\uparrow = R_2^{1\,T}$$

$$R_3^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{\sqrt{3}}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

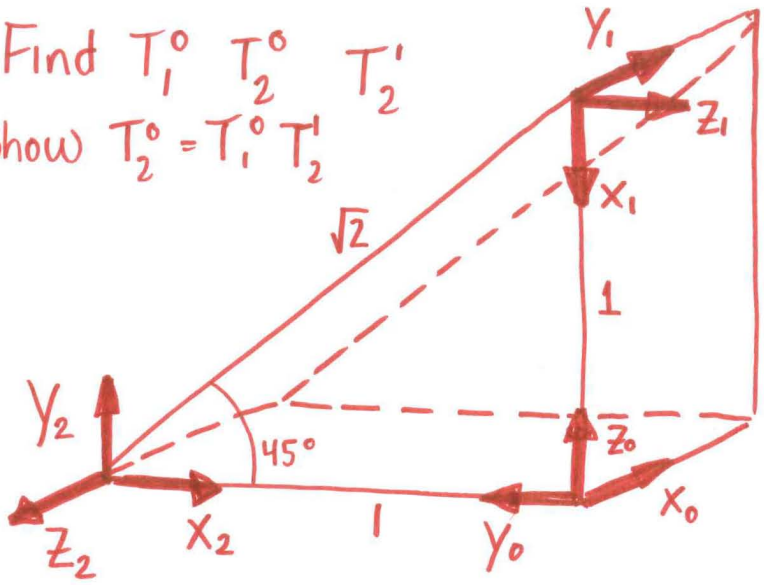
(1)

$$R_3^2 = \begin{bmatrix} 0 & 0 & -1 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \end{bmatrix}$$

(2)

Question #2

- Find T_1^0 T_2^0 T_2^1
- Show $T_2^0 = T_1^0 T_2^1$



use $$T_a^b = \left[\begin{array}{c|c} R_a^b & O_a^b \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$$

$$R_a^b = \left[\hat{x}_a^b \quad \hat{y}_a^b \quad \hat{z}_a^b\right]$$

$$T_1^0 = \left[\begin{array}{c|c} R_1^0 & O_1^0 \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$$

$$R_1^0 = \begin{bmatrix} \hat{x}_1^0 & \hat{y}_1^0 & \hat{z}_1^0 \end{bmatrix} \qquad O_1^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

$$\Rightarrow T_1^0 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^0 = \left[\begin{array}{c|c} R_2^0 & O_2^0 \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$$

$$R_2^0 = \begin{bmatrix} \hat{x}_2^0 & \hat{y}_2^0 & \hat{z}_2^0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$O_2^0 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

(4)

$$\Rightarrow T_2^0 = \left[\begin{array}{ccc|c} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^1 = \left[\begin{array}{c|c} R_2^1 & O_2^1 \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$$

$$R_2^1 = \left[\hat{x}_2^1 \ \ \hat{y}_2^1 \ \ \hat{z}_2^1\right] = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$O_2^1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

$$\Rightarrow T_2^1 = \left[\begin{array}{ccc|c} 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Show $T_2^0 = T_1^0 T_2^1$

$$T_2^0 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]\left[\begin{array}{ccc|c} 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$\Rightarrow T_2^0 = \left[\begin{array}{ccc|c} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

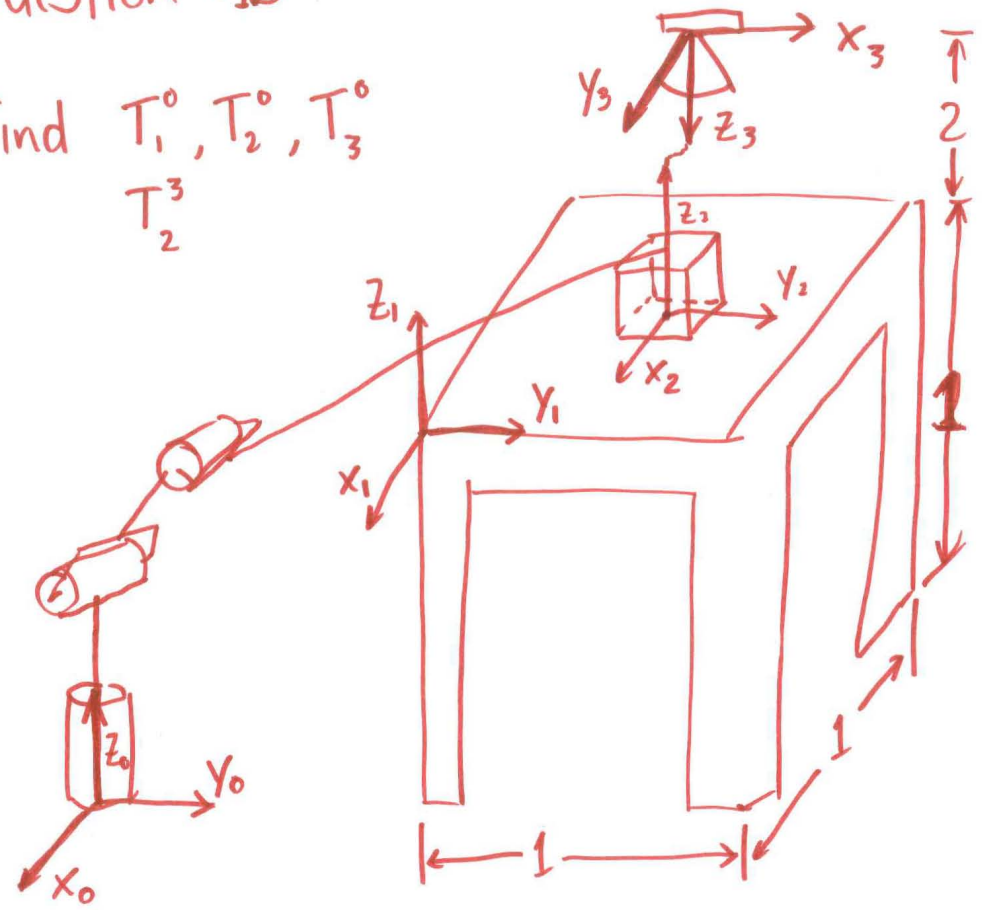
← equals result on page ⑤

⑥

Question #13

Find T_1^0, T_2^0, T_3^0

T_2^3



assuming that {2} origin is at center of bottom face of the cube

$$T_1^0 = \left[\begin{array}{c|c} R_1^0 & O_1^0 \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$$

$$R_1^0 = \left[\hat{X}_1^0 \quad \hat{Y}_1^0 \quad \hat{Z}_1^0\right] \qquad O_1^0 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$R_1^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow T_1^0 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^0 = \begin{bmatrix} R_2^0 & O_2^0 \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

$$R_2^0 = \left[\hat{X}_2^0 \quad \hat{Y}_2^0 \quad \hat{Z}_2^0\right] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(8)

$$O_2^0 = \begin{bmatrix} -\frac{1}{2} \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

$$\Rightarrow T_2^0 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & -1/2 \\ 0 & 1 & 0 & 3/2 \\ 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_3^0 = \begin{bmatrix} R_3^0 & O_3^0 \\ 000 & 1 \end{bmatrix}$$

$$R_3^0 = \begin{bmatrix} \hat{x}_3^0 & \hat{y}_3^0 & \hat{z}_3^0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$O_3^0 = \begin{bmatrix} -1/2 \\ 3/2 \\ 3 \end{bmatrix}$$

(9)

$$\Rightarrow T_3^0 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & -1/2 \\ 1 & 0 & 0 & 3/2 \\ 0 & 0 & -1 & 3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^3 = \begin{bmatrix} R_2^3 & O_2^3 \\ 000 & 1 \end{bmatrix}$$

$$R_2^3 = \begin{bmatrix} \hat{x}_2^3 & \hat{y}_2^3 & \hat{z}_2^3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$O_2^3 = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

(10)

$$\Rightarrow T_2^3 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_B^A = \left[\begin{array}{c|c} R_B^A & O_B^A \\ \hline 0\ 0\ 0 & 1 \end{array}\right] \qquad R_B^A = \begin{bmatrix} -\frac{1}{2} & 0 & -\sqrt{3}/2 \\ 0 & 1 & 0 \\ \sqrt{3}/2 & 0 & -1/2 \end{bmatrix}$$

given $v^A = [-2\ 4\ 2]^T$

evaluate $v^B = R_A^B v^A$

$$R_A^B = \left(R_B^A\right)^T = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \\ -\frac{\sqrt{3}}{2} & 0 & -\frac{1}{2} \end{bmatrix}$$

$$v^B = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \\ \frac{\sqrt{3}}{2} & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} -2 \\ 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 1+\sqrt{3} \\ 4 \\ \sqrt{3}-1 \end{bmatrix} \cong \begin{bmatrix} 2.732 \\ 4 \\ 0.732 \end{bmatrix}$$

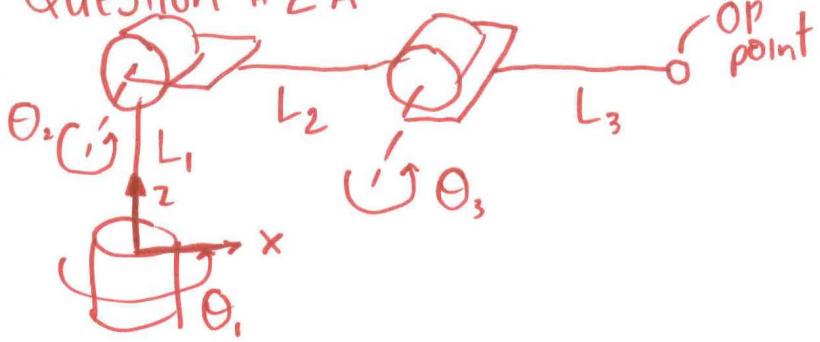
$$\|v^A\| = \sqrt{(-2)^2 + 4^2 + 2^2} = \sqrt{24}$$

$$\|v^B\| = \sqrt{(1+\sqrt{3})^2 + 4^2 + (\sqrt{3}-1)^2} = \sqrt{24}$$

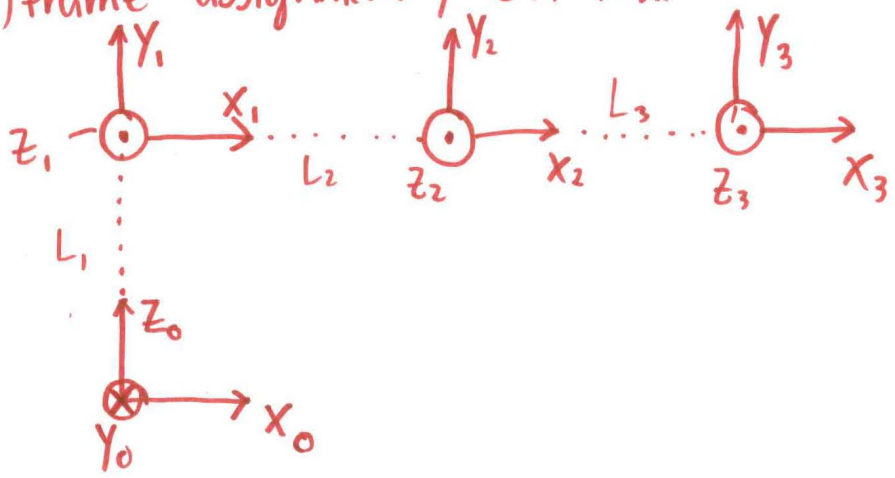
equal

magnitude of velocity vector is invariant to rotation

Question #2 A



(1) frame assignment / DH table



link	θ_i	d_i	a_i	α_i
1	θ_1	L_1	ϕ	$\pi/2$
2	θ_2	0	L_2	0
3	θ_3	0	L_3	0

(12)

(2)

$$T_i^{i-1} = \left[\begin{array}{ccc|c} C_{\theta_i} & -S_{\theta_i} C_{\alpha_i} & S_{\theta_i} S_{\alpha_i} & a_i C_{\theta_i} \\ S_{\theta_i} & C_{\theta_i} C_{\alpha_i} & -C_{\theta_i} S_{\alpha_i} & a_i S_{\theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_1^0 = \left[\begin{array}{ccc|c} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & L_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^1 = \left[\begin{array}{ccc|c} C_2 & -S_2 & 0 & L_2 C_2 \\ S_2 & C_2 & 0 & L_2 S_2 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_3^2 = \left[\begin{array}{ccc|c} C_3 & -S_3 & 0 & L_3C_3 \\ S_3 & C_3 & 0 & L_3S_3 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(3)

$$T_3^0 = T_1^0\, T_2^1\, T_3^2$$

$$= \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_2 & -S_2 & 0 & L_2C_2 \\ S_2 & C_2 & 0 & L_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T_3^2$$

$$= \left[\begin{array}{ccc|c} C_1C_2 & -C_1S_2 & S_1 & L_2C_1C_2 \\ S_1C_2 & -S_1S_2 & -C_1 & L_2S_1C_2 \\ S_2 & C_2 & 0 & L_2S_2 + L_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] T_3^2$$

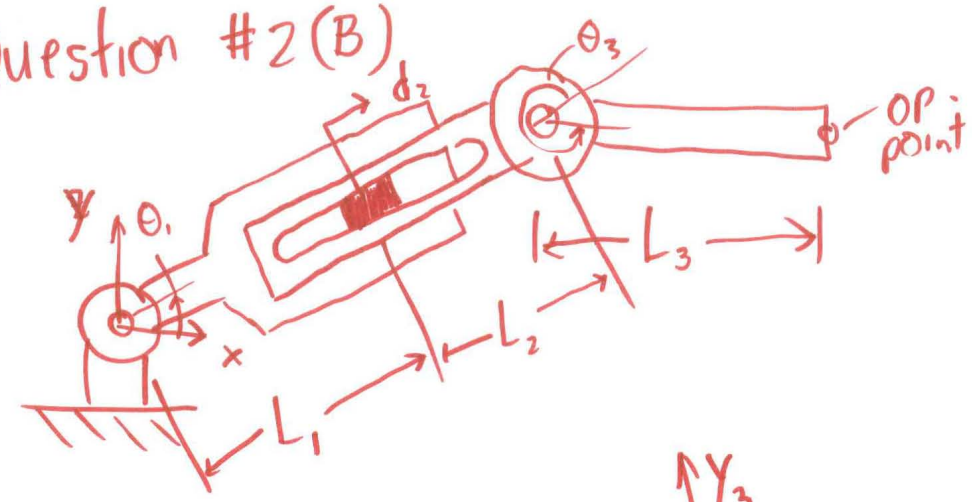
(14)

$$T_3^0 = \begin{bmatrix} C_1C_2 & -C_1S_2 & S_1 & L_2C_1C_2 \\ S_1C_2 & -S_1S_2 & -C_1 & L_2S_1C_2 \\ S_2 & C_2 & 0 & L_2S_2 + L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_3 & -S_3 & 0 & L_3C_3 \\ S_3 & C_3 & 0 & L_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

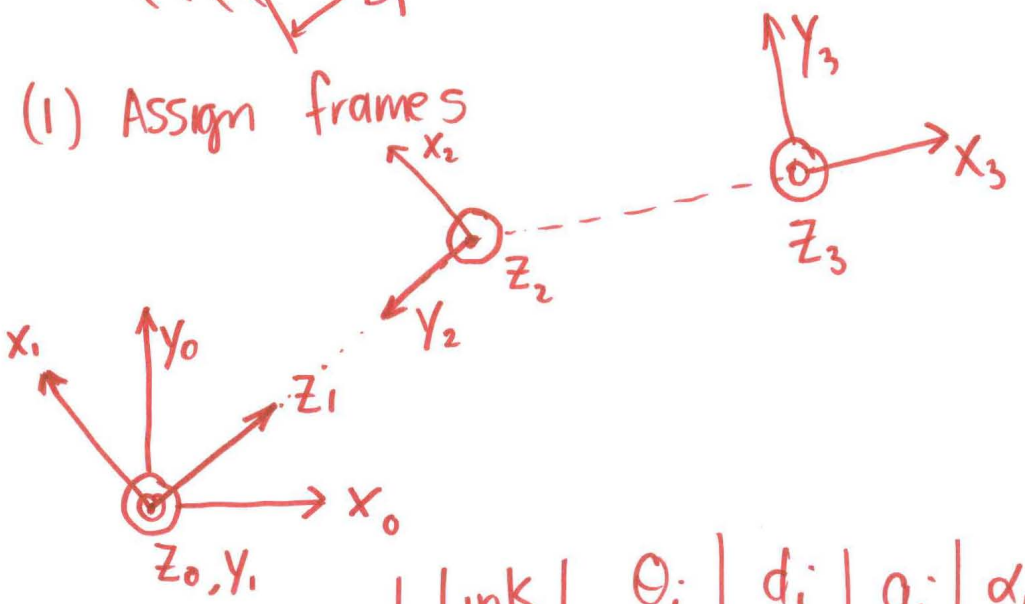
$$T_3^0 = \left[\begin{array}{ccc|c} C_1C_{23} & -C_1S_{23} & S_1 & L_3C_1C_{23} + L_2C_1C_2 \\ S_1C_{23} & -S_1S_{23} & -C_1 & L_3S_1C_{23} + L_2S_1C_2 \\ S_{23} & C_{23} & 0 & L_3S_{23} + L_2S_2 + L_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(15)

Question #2(B)



(1) Assign frames



Link	θ_i	d_i	a_i	α_i
1	$\theta_1 + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
2	0	$L_1 + L_2 + d_2$	0	$-\frac{\pi}{2}$
3	$\theta_3 - \frac{\pi}{2}$	0	L_3	0

(16)

(2)

T_1^0 :

$$\theta_i = \theta_1 + \frac{\pi}{2}$$

$$C_{\theta_i} = -S_1$$

$$S_{\theta_i} = C_1$$

$$S_{\alpha_i} = 1$$

$$C_{\alpha_i} = 0$$

$$T_1^0 = \left[\begin{array}{ccc|c} -S_1 & 0 & C_1 & 0 \\ C_1 & 0 & S_1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^1 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & L_1+L_2+d_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$T_3^2 \Rightarrow$

$$\theta_i = \theta_3 - \frac{\pi}{2}$$

$$C_{\theta_i} = S_{\theta_3} = S_3$$

$$S_{\theta_i} = -C_{\theta_3} = -C_3$$

$$C_{\alpha_i} = 1$$

$$S_{\alpha_i} = 0$$

(8)

$$T_3^2 = \left[\begin{array}{ccc|c} S_3 & C_3 & 0 & L_3 S_3 \\ -C_3 & S_3 & 0 & -L_3 C_3 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(3)

$$T_3^0 = T_1^0 T_2^1 T_3^2$$

$$= \left[\begin{array}{ccc|c} -S_1 & 0 & C_1 & 0 \\ C_1 & 0 & S_1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & L_1+L_2+d_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] T_3^2$$

$$= \left[\begin{array}{ccc|c} -S_1 & -C_1 & 0 & C_1(L_1+L_2+d_2) \\ C_1 & -S_1 & 0 & S_1(L_1+L_2+d_2) \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] T_3^2$$

(19)

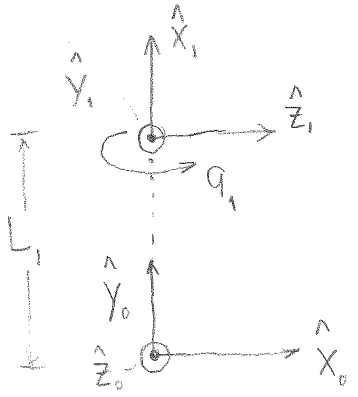
$$T_3^0 = \left[\begin{array}{ccc|c} -S_1 & -C_1 & 0 & C_1 L \\ C_1 & -S_1 & 0 & S_1 L \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc|c} S_3 & C_3 & 0 & L_3 S_3 \\ -C_3 & S_3 & 0 & -L_3 C_3 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$L = L_1 + L_2 + d_2$$

$$T_3^0 = \left[\begin{array}{ccc|c} C_{13} & -S_{13} & 0 & L_3 C_{13} + (L_1+L_2+d_2) C_1 \\ S_{13} & C_{13} & 0 & L_3 S_{13} + (L_1+L_2+d_2) S_1 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Find $T_1^0, T_2^1, \ldots, T_6^5$

T_1^0:

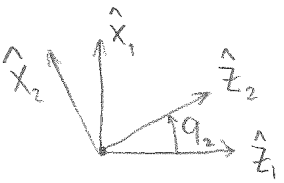


$$\hat{x}_1^0 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \hat{y}_1^0 = \begin{bmatrix} S_1 \\ 0 \\ C_1 \end{bmatrix} \quad \hat{z}_1^0 = \begin{bmatrix} C_1 \\ 0 \\ -S_1 \end{bmatrix}$$

$$O_1^0 = \begin{bmatrix} 0 \\ L_1 \\ 0 \end{bmatrix}$$

$$T_1^0 = \begin{bmatrix} R_1^0 & O_1^0 \\ 0\,0\,0 & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} 0 & S_1 & C_1 & 0 \\ 1 & 0 & 0 & L_1 \\ 0 & C_1 & -S_1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \Leftarrow$$

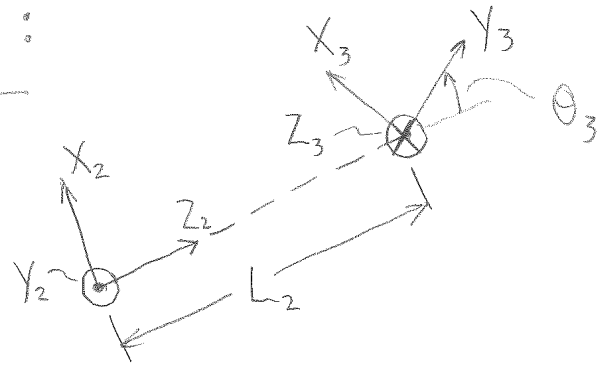
T_2^1:



$$\hat{x}_2^1 = \begin{bmatrix} C_2 \\ 0 \\ -S_2 \end{bmatrix} \quad \hat{y}_2^1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \hat{z}_2^1 = \begin{bmatrix} S_2 \\ 0 \\ C_2 \end{bmatrix} \quad O_2^1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$T_2^1 = \left[\begin{array}{ccc|c} C_2 & 0 & S_2 & 0 \\ 0 & 1 & 0 & 0 \\ -S_2 & 0 & C_2 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \Leftarrow$$

T_3^2:



Unit vectors

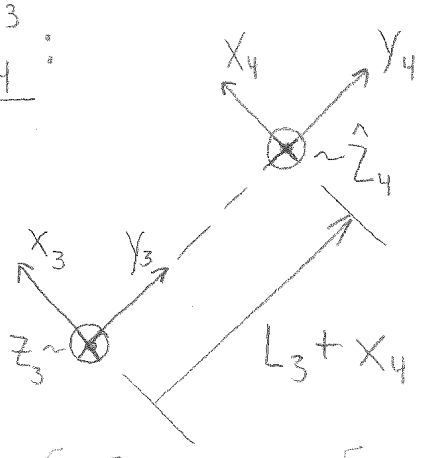
$$\hat{X}_3^2 = \begin{bmatrix} C_3 \\ 0 \\ -S_3 \end{bmatrix} \quad \hat{Y}_3^2 = \begin{bmatrix} S_3 \\ 0 \\ C_3 \end{bmatrix} \quad \hat{Z}_3^2 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

origin:

$$O_3^2 = \begin{bmatrix} 0 \\ 0 \\ L_2 \end{bmatrix}$$

$$T_3^2 = \left[\begin{array}{ccc|c} C_3 & S_3 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -S_3 & C_3 & 0 & L_2 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

T_4^3:



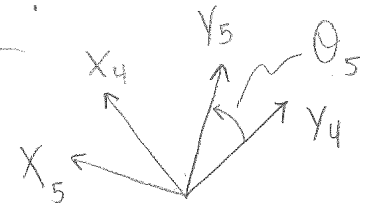
$$\hat{X}_4^3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \hat{Y}_4^3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad Z_4^3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Leftarrow \text{unit vectors}$$

origin of $\{4\}$ in $\{3\}$

$$O_4^3 = \begin{bmatrix} 0 \\ L_3 + X_4 \\ 0 \end{bmatrix}$$

$$T_4^3 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_3 + X_4 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

T_5^4:



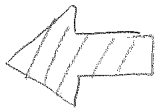
Unit vectors:

$$\hat{X}_5^4 = \begin{bmatrix} C_5 \\ -S_5 \\ 0 \end{bmatrix} \quad \hat{Y}_5^4 = \begin{bmatrix} S_5 \\ C_5 \\ 0 \end{bmatrix} \quad \hat{Z}_5^4 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

origin of {5} in {4}

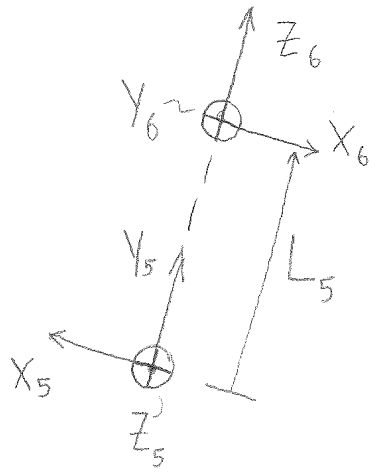
$$O_5^4 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$T_5^4 = \left[\begin{array}{ccc|c} C_5 & S_5 & 0 & 0 \\ -S_5 & C_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$



(8)

T_6^5:



Unit vectors:

$$\hat{X}_6^5 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \quad \hat{Y}_6^5 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \hat{Z}_6^5 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

origin of {6} in {5}

$$O_6^5 = \begin{bmatrix} 0 \\ L_5 \\ 0 \end{bmatrix}$$

$$T_6^5 = \left[\begin{array}{ccc|c} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & L_5 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(9)

to form the homogenous transformation matricies describing the frame displacements relative to ground

- frame {2}

$$T_2^0 = T_1^0 T_2^1$$

- frame {3}

$$T_3^0 = T_1^0 T_2^1 T_3^2 = T_2^0 T_3^2$$

- frame {4}

$$T_4^0 = T_1^0 T_2^1 T_3^2 T_4^3 = T_3^0 T_4^3$$

- frame {5}

$$T_5^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 = T_4^0 T_5^4$$

- frame {6}

$$T_6^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 = T_5^0 T_6^5$$

(10)

```
%-------------------------------------------------------------------
%       Homework #1 - Problem #6
%-------------------------------------------------------------------

clear all; close all; clc

% Parameters
L1 = 3;
L2 = 5;
L3 = 3;
L4 = 3;
L5 = 3;
h6 = 1;

%----set rendering window view parameters
% figure handle
f_handle = 1;
% axis limits
axis_limits = [-10 10 0 10 -10 10];
% camera position
render_view = [-1 1 -1];
% vertical orientation
view_up = [0 1 0];
% initialize rendering view
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);

%----initialize rendering

%  base (link 0) rendering initialization
%    (doesn't move - created for aesthetic reasons)
r = 3; L = 0.2;  sides = 4; Axis = 2; norm_L = 1.0;
    linkColor = [.5 .5 .5]; plotFrame = .1 ;
d0 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);

% link 1 rendering initialization
r = 1.0; L = L1;  sides = 10; Axis = 1; norm_L = 1.0;
    linkColor = [0.75 0 0]; plotFrame = .1;
d1 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);

% link 2 rendering initialization
r = 0.75; L = L2;  sides = 10; Axis = 3; norm_L = -1.0;
    linkColor = [0 0.75 0]; plotFrame = .1;
d2 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);
```

```
% link 3 rendering initialization
r = 0.75; L = L3;  sides = 10; Axis = 2; norm_L = -1.0;
    linkColor = [0 0 0.75]; plotFrame = .1;
d3 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);

% link 4 rendering initialization
r = 0.6; L = L4;  sides = 10; Axis = 2; norm_L = 1.0;
    linkColor = [0.75 0 0]; plotFrame = .1;
d4 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);

% link 5 rendering initialization
r = 0.5; L = L5;  sides = 10; Axis = 2; norm_L = -1.0;
    linkColor = [0 0.75 0]; plotFrame = .1;
d5 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);

% link 6 rendering initialization (to visualize end effector motion)
r = 0.2; L = 2;  sides = 10; Axis = 1; norm_L = 0.0;
    linkColor = [0 0 0.75]; plotFrame = .1;
d6 =  CreateLinkRendering(L,r,sides,Axis,norm_L,linkColor,...
                          plotFrame,f_handle);

%----joint variables as a function of time

%  create time vector
tEnd = 2*pi; % animation run time
SamplesPerSimulation = 200;
t = linspace(0,tEnd,SamplesPerSimulation)';

% create joint position vectors
q1 =  -pi*sin(t);
q2 =  (pi/4)*(1-cos(t));
q3 =  (pi/4)*sin(t);
q4 =   L3*(1-cos(t))/2;
q5 = -(pi/4)*sin(t);
q6 =  (pi/4)*sin(t);

%----evaluate the homogeneous transformation matrices that describe the
%----displacement of sucessive frames.
for i = 1:size(t,1)
    %transformation from {1} to {0}
    s = sin(q1(i)); c = cos(q1(i));
    T10(:,:,i) = [0  s   c  0
```

```
             1  0   0  L1
             0  c  -s  0
             0  0   0  1];
%transformation from {2} to {1}
s = sin(q2(i)); c = cos(q2(i));
T21(:,:,i) = [ c  0  s  0
               0  1  0  0
              -s  0  c  0
               0  0  0  1];
%transformation from {3} to {2}
s = sin(q3(i)); c = cos(q3(i));
T32(:,:,i) = [ c  s   0  0
               0  0  -1  0
              -s  c   0  L2
               0  0   0  1];
%transformation from {4} to {3}
T43(:,:,i) = [ 1  0   0  0
               0  1   0  L3+q4(i)
               0  0   1  0
               0  0   0  1];
%transformation from {5} to {4}
s = sin(q5(i)); c = cos(q5(i));
T54(:,:,i) = [ c  s   0  0
              -s  c   0  0
               0  0   1  0
               0  0   0  1];
%transformation from {5} to {6}
s = sin(q6(i)); c = cos(q6(i));
T65(:,:,i) = [-c  s  0   0
               0  0  1  L5
               s  c  0   0
               0  0  0   1];
%transformations to the ground frame {0}
T20(:,:,i) = T10(:,:,i)*T21(:,:,i);
T30(:,:,i) = T20(:,:,i)*T32(:,:,i);
T40(:,:,i) = T30(:,:,i)*T43(:,:,i);
T50(:,:,i) = T40(:,:,i)*T54(:,:,i);
T60(:,:,i) = T50(:,:,i)*T65(:,:,i);

%evaluate the position of the end-effector point, E, relative to {0}
h6 = 1;
p6 = [-h6 0 0]';  P6 = [p6; 1];
XE(:,:) = T60(:,:,i)*P6;
xE(:,i) = XE(1);
yE(:,i) = XE(2);
zE(:,i) = XE(3);
```

```
end

%----update the manipulator rendering
for i = 1:size(t,1)
    % update the link rendering
    UpdateLink(d1,T10(:,:,i));
    UpdateLink(d2,T20(:,:,i));
    UpdateLink(d3,T30(:,:,i));
    UpdateLink(d4,T40(:,:,i));
    UpdateLink(d5,T50(:,:,i));
    UpdateLink(d6,T60(:,:,i));
    if i == 1
        pause; % to allow resizing of graphics window
    end
    %pause;
    pause(0.1); % pause time adjusted to give a smooth response
end

%----plote the end effector point, E, position as a function of time
figure;
subplot(1,3,1);plot(t,xE);xlabel('time');ylabel('x')
a = axis; axis([a(1) tEnd a(3) a(4)])
subplot(1,3,2);plot(t,yE);xlabel('time');ylabel('y')
a = axis; axis([a(1) tEnd a(3) a(4)])
subplot(1,3,3);plot(t,zE);xlabel('time');ylabel('z')
a = axis; axis([a(1) tEnd a(3) a(4)])
```

2.2 HW 2

2.2.1 Problem 1

Problem 1.

Frame {A} and frame {B} are initially coincident. Frame {B} is rotated through the following sequence of rotations:

1. $+\alpha$ degree rotation about $\hat{z}_A$
2. -β degree rotation about $\hat{x}_B$
3. -α degree rotation about $\hat{z}_A$
4. $+\gamma$ degree rotation about $\hat{y}_B$
5. $+\beta$ degree rotation about $\hat{x}_A$

where α = +45 degrees; β = -30 degrees; γ = +90 degrees

Evaluate the rotation transformation (matrix) that describes the orientation of frame {B} relative to frame {A} following this sequence of rotations.

When the rotation R_i is around a fixed frame, it is pre-multiplied by the current sequence of rotations. If the rotation R_i is around the current frame, it is post-multiplied by the current sequence of rotations.

1. $R = R_{z,\alpha}$
2. Since rotation is around current frame, it is post multiplied giving $R = R_{z,\alpha}R_{x,-\beta}$
3. Since rotation is around fixed frame, it is pre-multiplied, giving $R = R_{z,-\alpha}R_{z,\alpha}R_{x,-\beta}$
4. Since rotation now is about current frame, it is post multiplied giving $R = R_{z,-\alpha}R_{z,\alpha}R_{x,-\beta}R_{y,\gamma}$
5. Since rotation now is about fixed frame, it is pre-multiplied giving $R = R_{x,\beta}R_{z,-\alpha}R_{z,\alpha}R_{x,-\beta}R_{y,\gamma}$

Now that the rotation sequence is completed, the sequence of rotations are evaluated. Before that, some simplification is made as follows

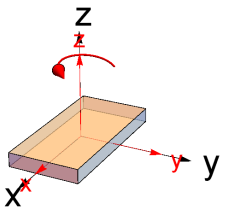
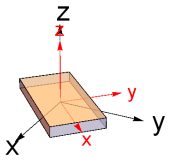
$$\begin{aligned} R &= R_{x,\beta}\overbrace{R_{z,-\alpha}R_{z,\alpha}}^{I}R_{x,-\beta}R_{y,\gamma} \\ &= \overbrace{R_{x,\beta}R_{x,-\beta}}^{I}R_{y,\gamma} \\ &= R_{y,\gamma} \end{aligned}$$

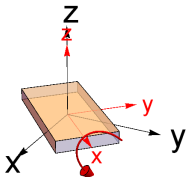
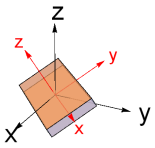
Therefore

$$R = R_{y,\gamma} = \begin{pmatrix} \cos\gamma & 0 & \sin\gamma \\ 0 & 1 & 0 \\ -\sin\gamma & 0 & \cos\gamma \end{pmatrix} = \begin{pmatrix} \cos 90^0 & 0 & \sin 90^0 \\ 0 & 1 & 0 \\ -\sin 90^0 & 0 & \cos 90^0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

Only angle γ was needed in finding the final result. The above result shows that the final orientation is as if one rotation of $+90$ degree was made around the fixed y axes. The following diagram shows the result after each rotation, which confirms the above result.[3]

[3]Source code that generated these plot is in the appendix if needed.

+45 around fixed z	after rotation	Final rotation matrix
		$\begin{pmatrix} 0.707107 & -0.707107 & 0. \\ 0.707107 & 0.707107 & 0. \\ 0. & 0. & 1. \end{pmatrix}$

30° around current x	after rotation	Final rotation matrix
		$\begin{pmatrix} 0.707107 & -0.612372 & 0.353553 \\ 0.707107 & 0.612372 & -0.353553 \\ 0. & 0.5 & 0.866025 \end{pmatrix}$

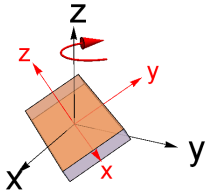
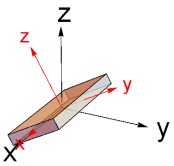
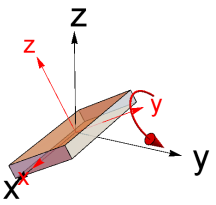
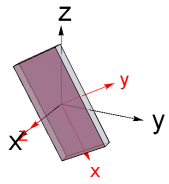
-45° around fixed z	after rotation	Final rotation matrix
		$\begin{pmatrix} 1. & 0. & 0. \\ 0. & 0.866025 & -0.5 \\ 0. & 0.5 & 0.866025 \end{pmatrix}$

Figure 2.5: Graphical representation of problem 1 rotations

+90 around current y	after rotation	Final rotation matrix
		$\begin{pmatrix} 0. & 0. & 1. \\ 0.5 & 0.866025 & 0. \\ -0.866025 & 0.5 & 0. \end{pmatrix}$

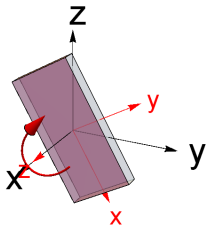
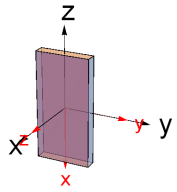
-30° around fixed x	after rotation	Final rotation matrix
		$\begin{pmatrix} 0 & 0 & 1. \\ 0 & 1. & 0 \\ -1. & 0 & 0 \end{pmatrix}$

Figure 2.6: Graphical representation of problem 1 rotations

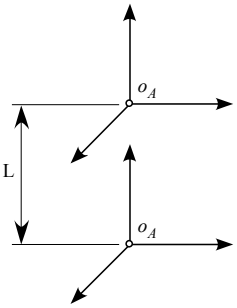
2.2.2 Problem 2

Problem 2.

The D-H parameters (d, a, α, and θ) and the homogeneous transformation which results (see below – also in Kinematics lecture notes) cannot be used to represent a *general* rigid-body transformation.

Homogeneous transformation matrix using DH convention:

$$T = \begin{bmatrix} c_\theta & -s_\theta c_\alpha & s_\theta s_\alpha & a c_\theta \\ s_\theta & c_\theta c_\alpha & -c_\theta s_\alpha & a s_\theta \\ 0 & s_\alpha & c_\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- ► Explain why this is the case. You can use physical and/or mathematical arguments.
- ► For the rigid-body transformation shown to the right, label the unit vectors of the two reference frames such that the D-H parameters cannot describe their relative transformation.

Part a

The homogeneous transformation matrix based on the use of the DH convention contains only 4 parameters d, a, α, θ. Since a general rigid body transformation requires 6 parameters (3 angles for orientations, and 3 for translation), this implies the DH homogeneous transformation matrix can not be used to represent any arbitrary rigid body transformation. However, the DH convention can be used to represent any rigid body transformation that meets two conditions, as specified on page 78 of the text book. These are

1. The x_i axis of the i^{th} frame is $\perp$ to the z_{i-1} axis of the $i-1$ frame.
2. The x_i axis of the i^{th} frame intersects the z_{i-1} axis of the $i-1$ frame

Part b

At least one of the above two DH constraints need to be violated in order to come up with a configuration that cannot be described using the DH transformation matrix. This is done by making x_1 axis not perpendicular to z_0 axis. The following diagram illustrates this

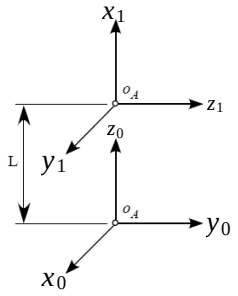
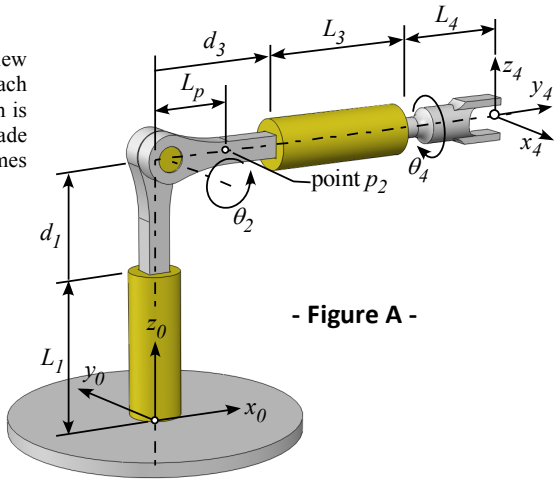


Figure 2.7: Problem 2 setup of axis

2.2.3 Problem 3

Problem 3.

► Sketch the DH frames on the planar view of the manipulator given below. For each frame, state whether the frame definition is unique and describe the choices you made in the table below. Use the defined frames {0} and {4} shown in Figure A.



- Figure A -

Planar view of the manipulator:

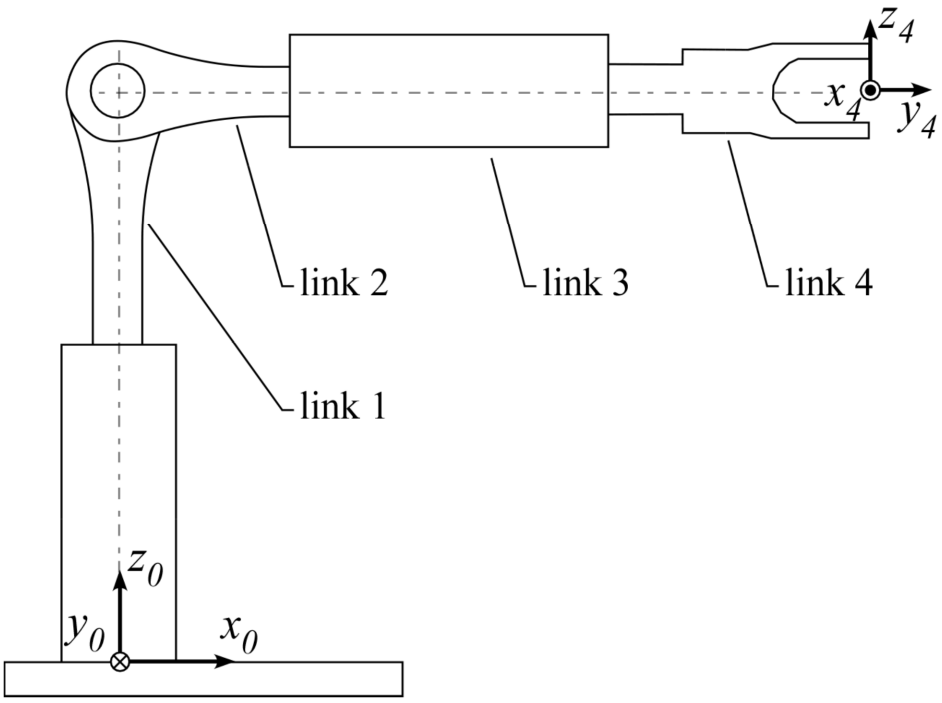


Figure 2.8: Problem 3 description

The first step is to assign the z_i axis for each link. There are 5 links numbered $0, 1, 2, 3, 4$ and four joints numbered $1, 2, 3, 4$. Link 0 is fixed and does not move. Frame i is attached to link i but its z_i axis is oriented along the line of motion of joint $i+1$. Once all the z axis are established, then the rest of the frames are configured by insuring that that each x axis is perpendicular to the last frame z axis.

So the rule to follow is that x_i axis must be perpendicular to z_{i-1} axis.

The frames are drawn below. The following choices were made. z_0 intersects z_1 (case 3 in the textbook), hence x_1 is arbitrary. The origin o_1 is the point of intersection of z_1 and z_0 as shown in the problem statement.

Next, z_1 and z_2 also intersect (case 3). The choice of x_2 is arbitrary. The origin o_2 could be located also where z_1 intersect z_2. But any point along z_2 will also work. Here o_2 was placed at point p_2.

Next, z_2 is parallel to z_3. This is case (2) in the textbook. There are infinitely many common normals. Here origin o_3 can be anywhere along z_3. It is placed at start of the end effector as shown.

The following diagram shows the frames locations.

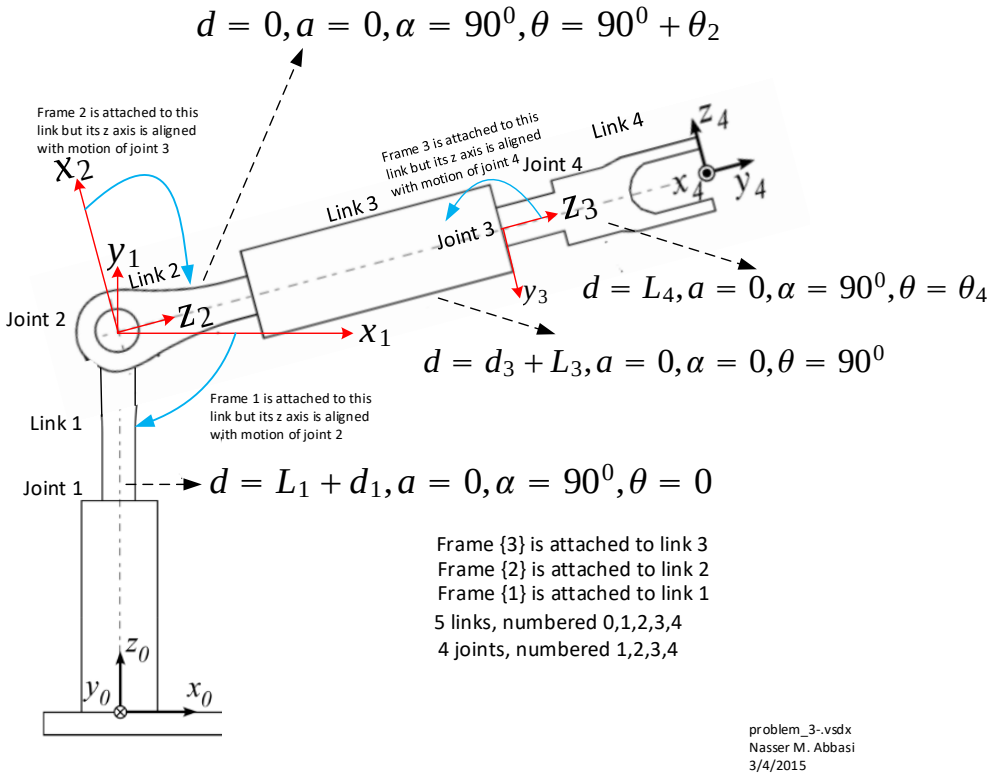


Figure 2.9: Frames assignments for problem 3

2.2.4 Problem 4

Evaluate the DH parameters for the manipulator and frame definitions developed in the previous prob

i	θ_i	d_i	a_i	α_i
1	0	$L_1 + d_1$	0	90^0
2	$90^0 + \theta_2$	0	0	90^0
3	90^0	$L_3 + d_3$	0	0
4	θ_4	L_4	0	90^0

The homogeneous transformation matrices T_i^{i-1} are now evaluated, and then T_4^0 is found in order to verify that the location of the end effector. To verify, when $\theta_2 = 0, \theta_4 = 0$, then the end effector x, y, z position relative to the base frame should be located at

$$
\begin{aligned}
x &= d_3 + L_3 + L_4 \\
y &= 0 \\
z &= L_1 + d_1
\end{aligned}
$$

These are The homogeneous transformation matrices T_i^{i-1}

$$
T_1^0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & d_1 + L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
T_2^1 = \begin{pmatrix} -\sin\theta_2 & 0 & \cos\theta_2 & 0 \\ \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
T_3^2 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_3 + L_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$$
T_4^3 = \begin{pmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & L_4 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

$T_4^0 = T_1^0 T_2^1 T_3^2 T_4^3$ was found and evaluated for $\theta_2 = 0$. The result is

$$
\begin{pmatrix} 0 & 1 & 0 & d_3 + L_3 + L_4 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_1 + L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Which shows the correct position vector of the end effector frame.

2.2.5 Problem 5

Problem 5

▶ Write a Matlab function which takes as its input the DH parameters and returns the associated 4x4 homogeneous transformation matrix, T. A possible function prototype is given below. To learn how to write a `function` in Matlab, type `help function` in the Matlab workspace.

```
function T = DH2T(d, a, alpha, theta);
```

▶ Write a Matlab script to verify that your function is working properly. The script should evaluate the T matrix for the set of DH parameters listed in the adjacent table. When your script is run, the T matrices should be printed to the workspace. This script, along with your function, are to be handed in to the course dropbox on the Learn@UW course page. As a possible starting point, your script might look like:

Link	d_i	a_i	α_i	θ_i
Case 1:	0	10	0	0
Case 2:	10	0	0	0
Case 3:	10	0	π	0
Case 4:	0	0	π	π

```
clear all; close all; clc
% Case 1:
d = 0;  a = 10; alpha = 0;  theta = 0;
T1 = DH2T(a,alpha,d,theta)
% Case 2:
d = 10; a = 0;  alpha = 0;  theta = 0;
T2 = DH2T(a,alpha,d,theta)
% Case 3:
d = 10; a = 0;  alpha = pi; theta = 0;
T3 = DH2T(a,alpha,d,theta)
% Case 4:
d = 0;  a = 0;  alpha = pi; theta = pi;
T4 = DH2T(a,alpha,d,theta)
```

Figure 2.10: Problem 5 description

part a

The following is the Matlab function `DH2T`

```
function T = DH2T(a,alpha,d,theta)
%generated the transformation matrix for a DH table row
%ME 739, Univ. Of Wisconsin, Madison. Spring 2015
T=[cos(theta)  -sin(theta)*cos(alpha)  sin(theta)*sin(alpha)   a*cos(theta);
   sin(theta)  cos(theta)*cos(alpha)   -cos(theta)*sin(alpha)  a*sin(theta);
   0           sin(alpha)              cos(alpha)              d;
   0           0                       0                       1
   ];
end
```

part b

The following is the script used and below it is the output generated.

```
clear all; close all; clc
%case 1
d=0; a=10; alpha=0; theta=0;
T1=DH2T(a,alpha,d,theta)

%case 2
d=10; a=10; alpha=0; theta=0;
T2=DH2T(a,alpha,d,theta)

%case 3
d=10; a=10; alpha=pi; theta=0;
T3=DH2T(a,alpha,d,theta)

%case 4
d=0; a=10; alpha=pi; theta=pi;
T4=DH2T(a,alpha,d,theta)
```

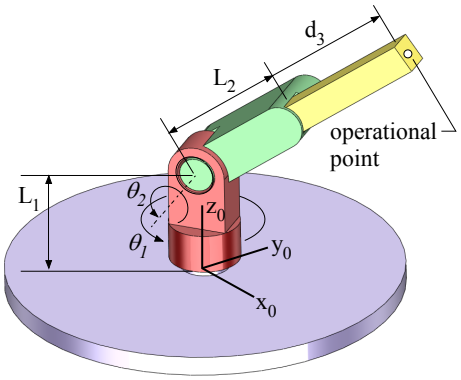
```
T1 =
   1     0     0    10
   0     1     0     0
   0     0     1     0
   0     0     0     1

T2 =
   1     0     0    10
   0     1     0     0
   0     0     1    10
   0     0     0     1

T3 =
   1             0            0           10
   0            -1  -1.2246e-16            0
   0   1.2246e-16           -1           10
   0             0            0            1

T4 =
   -1   1.2246e-16   1.4998e-32          -10
   1.2246e-16             1   1.2246e-16   1.2246e-15
   0   1.2246e-16           -1            0
   0             0            0            1
```

2.2.6 Problem 6

Problem 6.

For the manipulator shown, the forward kinematics are given as:

$$x_e = \cos\theta_1 \cos\theta_2 (L_2 + d_3)$$
$$y_e = \sin\theta_1 \cos\theta_2 (L_2 + d_3)$$
$$z_e = L_1 + \sin\theta_2 (L_2 + d_3)$$



- Develop the linear velocity Jacobian, J_v, using direct differentiation of the forward kinematics. Assume that the task and joint variable vectors are defined as

$$q = \begin{bmatrix} \theta_1 & \theta_2 & d_3 \end{bmatrix}^T$$
$$x = \begin{bmatrix} x_e & y_e & z_e \end{bmatrix}^T$$

- For the manipulator configuration defined by the joint vector, $q = \begin{bmatrix} \pi & \frac{1}{2}\pi & L_2 \end{bmatrix}^T$, evaluate the linear velocity of the end-effector given the joint velocity vector $\dot{q} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$.

- For what values of the joint variables is the linear velocity Jacobian, J_v, singular? Use physical and/or mathematical arguments to support your answer.

Figure 2.11: Problem 6 description

Part a

$$J_v = \begin{pmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial d_3} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial d_3} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial d_3} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial}{\partial \theta_1} \cos\theta_1 \cos\theta_2 (L_2 + d_3) & \frac{\partial}{\partial \theta_2} \cos\theta_1 \cos\theta_2 (L_2 + d_3) & \frac{\partial}{\partial d_3} \cos\theta_1 \cos\theta_2 (L_2 + d_3) \\ \frac{\partial}{\partial \theta_1} \sin\theta_1 \cos\theta_2 (L_2 + d_3) & \frac{\partial}{\partial \theta_2} \sin\theta_1 \cos\theta_2 (L_2 + d_3) & \frac{\partial}{\partial d_3} \sin\theta_1 \cos\theta_2 (L_2 + d_3) \\ \frac{\partial}{\partial \theta_1} (L_1 + \sin\theta_2 (L_2 + d_3)) & \frac{\partial}{\partial \theta_2} (L_1 + \sin\theta_2 (L_2 + d_3)) & \frac{\partial}{\partial d_3} (L_1 + \sin\theta_2 (L_2 + d_3)) \end{pmatrix}$$

$$= \begin{pmatrix} -\sin\theta_1 \cos\theta_2 (L_2 + d_3) & -\cos\theta_1 \sin\theta_2 (L_2 + d_3) & \cos\theta_1 \cos\theta_2 \\ \cos\theta_1 \cos\theta_2 (L_2 + d_3) & -\sin\theta_1 \sin\theta_2 (L_2 + d_3) & \sin\theta_1 \cos\theta_2 \\ 0 & \cos\theta_2 (L_2 + d_3) & \sin\theta_2 \end{pmatrix}$$

Part b

$$
\begin{aligned}
v &= J_v \dot{q} \\
&= \begin{pmatrix} -\sin\theta_1 \cos\theta_2 \left(L_2 + d_3\right) & -\cos\theta_1 \sin\theta_2 \left(L_2 + d_3\right) & \cos\theta_1 \cos\theta_2 \\ \cos\theta_1 \cos\theta_2 \left(L_2 + d_3\right) & -\sin\theta_1 \sin\theta_2 \left(L_2 + d_3\right) & \sin\theta_1 \cos\theta_2 \\ 0 & \cos\theta_2 \left(L_2 + d_3\right) & \sin\theta_2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} -\sin\pi \cos\frac{\pi}{2} \left(L_2 + L_2\right) & -\cos\pi \sin\frac{\pi}{2} \left(L_2 + L_2\right) & \cos\pi \cos\frac{\pi}{2} \\ \cos\pi \cos\frac{\pi}{2} \left(L_2 + L_2\right) & -\sin\pi \sin\frac{\pi}{2} \left(L_2 + L_2\right) & \sin\pi \cos\frac{\pi}{2} \\ 0 & \cos\frac{\pi}{2} \left(L_2 + L_2\right) & \sin\frac{\pi}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 2L_2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}
\end{aligned}
$$

Part(c)

Mathematically, J_v is singular when its determinant is zero, or when J_v has a zero eigenvalue. The determinant of the Jacobian [4] was found to be

$$\boxed{|J_v| = \left(d_3 + L_2\right)^2 \cos\theta_2}$$

The above is zero when $\cos\theta_2 = 0$. This occurs when $\theta_2 = \pm\frac{\pi}{2}$ (and any odd integer multiple of this value).

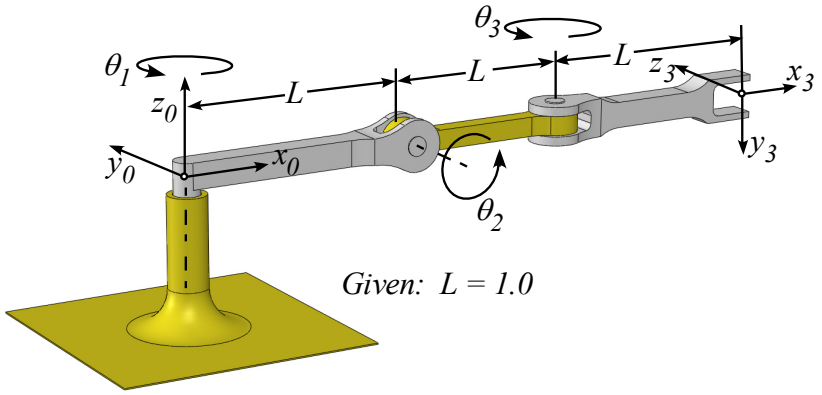
This implies that the singularity occurs when the arm is fully extended vertically in the upward position, or if physically possible, when the arm is fully extended vertically but in the downwards position.

2.2.7 Problem 7

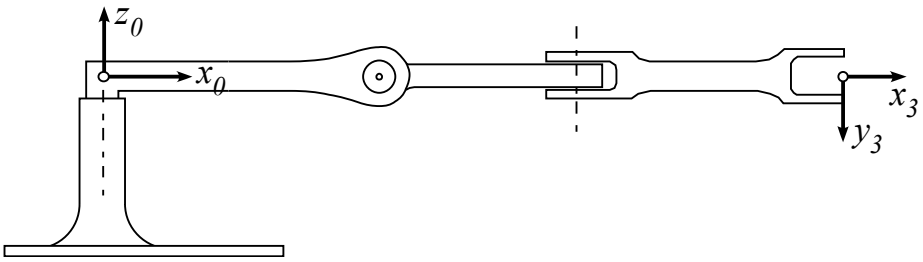
[4] Using syms and simplification

Problem 7.

Consider the three axis RRR manipulator shown in the figure below



► Derive the forward kinematics, T_3^0, of this manipulator as a function of the joint displacements and the geometric parameters shown in the figure. Sketch your intermediate frame definitions on the plane view of the manipulator shown below. Keep in mind that your frame definitions should be consistent with the conventions assumed when constructing the explicit form of the basic Jacobian.



► Evaluate the full basic Jacobian, J_0, for this manipulator. In this case, the basic Jacobian relates the joint space velocities, $\dot{q}$, and task space velocities $\dot{X}$.

$\dot{X} = J_o \dot{q}$ where $\dot{X} = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \omega_x \quad \omega_y \quad \omega_z]^T$ and $\dot{q} = [\dot{\theta}_1 \quad \dot{\theta}_2 \quad \dot{\theta}_3]^T$

- For the manipulator configurations listed below, evaluate the required joint torques to react the applied end-effector forces and torques: $F = [f_x \quad 0 \quad 0 \quad 0 \quad 0 \quad \tau_z]^T$. Discuss you results.

Configuration	θ_1	θ_2	θ_3
1	0	30°	0
2	0	0	30°
3	0	90°	90°

- Evaluate the linear velocity Jacobian, J_v^1, expressed in frame {1}
- Using J_v^1 find the singularities of the manipulator (with respect to the end-effector's linear velocity)
- For each type of singularity that you identified explain the physical interpretation of the singularity - by sketching the arm in a singular configuration and describing the resulting limitation on its movement.
- For the manipulator above, a new task position representation has been defined as

  $u = 2x + 3y$

  $v = x + y - z$

  $w = z$

  Evaluate the *linear* velocity analytical Jacobian, J_a, for this new representation when the manipulator configuration is given as $\vec{q} = [\theta_1 \quad \theta_2 \quad \theta_3] = \left[0 \quad \frac{\pi}{2} \quad \frac{\pi}{2}\right]$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = J_a \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}$$

Figure 2.12: Problem 7 description

Derive the forward kinematics

The frames are first assigned to each link by insuring that that the z axis of each frame follows the DH convention, which means z_i axis attached to link i is aligned in the direction of motion of joint $i + 1$.

For a revolute joint, this will be its axis of spin, and for prismatic joint, this will be its direction of sliding. Once this is done the forward kinematics matrices are found using either the DH table method or using homogeneous transformation by direct inspection. In this case, the homogeneous transformation by direct inspection method was used. The frames are assigned as follows

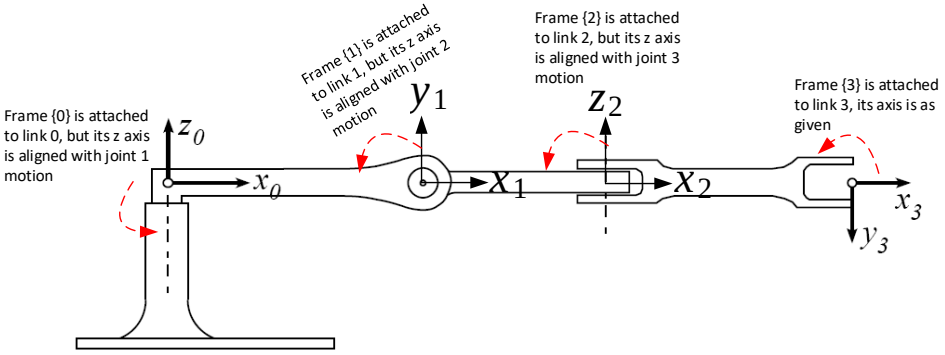


Figure 2.13: Frames assignments for problem 7

Determine forward kinematics matrices and evaluate full basic Jacobian J_0

Define the matrices T_1^0, T_2^1, T_3^2 using homogeneous transformation. Since $x' = J_0 q'$ then to find J_0 we need to first find forward kinematics. By inspection, we find each transformation matrix to be

$$T_1^0 = \begin{pmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & L\cos(\theta_1) \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & L\sin(\theta_1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2^1 = \begin{pmatrix} \cos(\theta_2) & 0 & -\sin(\theta_2) & L\cos(\theta_2) \\ \sin(\theta_2) & 0 & \cos(\theta_2) & L\sin(\theta_2) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_3^2 = \begin{pmatrix} \cos(\theta_3) & 0 & -\sin(\theta_3) & L\cos(\theta_3) \\ \sin(\theta_3) & 0 & \cos(\theta_3) & L\sin(\theta_3) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Therefore

$$T_2^0 = T_1^0 T_2^1 = \begin{pmatrix} \cos(\theta_1)\cos(\theta_2) & -\sin(\theta_1) & -\cos(\theta_1)\sin(\theta_2) & L\cos(\theta_1) + L\cos(\theta_2)\cos(\theta_1) \\ \cos(\theta_2)\sin(\theta_1) & \cos(\theta_1) & -\sin(\theta_1)\sin(\theta_2) & L\sin(\theta_1) + L\cos(\theta_2)\sin(\theta_1) \\ \sin(\theta_2) & 0 & \cos(\theta_2) & L\sin(\theta_2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_3^0 = T_2^0 T_3^2$$

$$= \begin{pmatrix} \cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_3) & \cos(\theta_1)\sin(\theta_2) & -\cos(\theta_3)\sin(\theta_1) - \cos(\theta_1)\cos(\theta_2)\sin(\theta_3) & L(\cos(\theta_1)(\cos(\theta_2)(\cos(\theta_3)+1)+1) - \sin \\ \cos(\theta_2)\cos(\theta_3)\sin(\theta_1) + \cos(\theta_1)\sin(\theta_3) & \sin(\theta_1)\sin(\theta_2) & \cos(\theta_1)\cos(\theta_3) - \cos(\theta_2)\sin(\theta_1)\sin(\theta_3) & L((\cos(\theta_2)(\cos(\theta_3)+1)+1)\sin(\theta_1) + \cos \\ \cos(\theta_3)\sin(\theta_2) & -\cos(\theta_2) & -\sin(\theta_2)\sin(\theta_3) & L(\cos(\theta_3)+1)\sin(\theta_2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In short notation the above can be written as

$$T_2^0 = \begin{pmatrix} C_1C_2 & -S_1 & -C_1S_2 & LC_1\left(C_2+1\right) \\ C_2S_1 & C_1 & -S_1S_2 & L\left(C_2+1\right)S_1 \\ S_2 & 0 & C_2 & LS_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_3^0 = \begin{pmatrix} C_1C_2C_3 - S_1S_3 & C_1S_2 & -C_3S_1 - C_1C_2S_3 & L\left(C_1\left(C_2\left(C_3+1\right)+1\right) - S_1S_3\right) \\ C_2C_3S_1 + C_1S_3 & S_1S_2 & C_1C_3 - C_2S_1S_3 & L\left(\left(C_2\left(C_3+1\right)+1\right)S_1 + C_1S_3\right) \\ C_3S_2 & -C_2 & -S_2S_3 & L\left(C_3+1\right)S_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Now we find the base Jacobian J_0 where $x' = J_0 q'$

$$\boxed{J_0 = \begin{pmatrix} J_{v_1} & J_{v_2} & J_{v_3} \\ J_{\omega_1} & J_{\omega_2} & J_{\omega_3} \end{pmatrix}}$$

where

$$J_{v_i}^0 = \epsilon_i z_{i-1}^0 + \hat{\epsilon}\left(z_{i-1}^0 \times \left(o_n^0 - o_{i-1}^0\right)\right)$$
$$J_{\omega_i}^0 = \hat{\epsilon} z_{i-1}^0$$

z_0^0 is given by $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ and

$$
\begin{aligned}
J_{v_1} &= z_0^0 \times (o_3^0 - o_0^0) \\
&= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \times \begin{pmatrix} L\left(\cos\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)-\sin\left(\theta_1\right)\sin\left(\theta_3\right)\right) \\ L\left(\sin\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)+\sin\left(\theta_3\right)\cos\left(\theta_1\right)\right) \\ L\sin\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right) \end{pmatrix} \\
&= \begin{pmatrix} -L\left(\sin\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)+\sin\left(\theta_3\right)\cos\left(\theta_1\right)\right) \\ L\left(\cos\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)-\sin\left(\theta_1\right)\sin\left(\theta_3\right)\right) \\ 0 \end{pmatrix} \\
J_{v_2} &= z_1^0 \times (o_3^0 - o_1^0) \\
&= \begin{pmatrix} \sin\left(\theta_1\right) \\ -\cos\left(\theta_1\right) \\ 0 \end{pmatrix} \times \left(\begin{pmatrix} L\left(\cos\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)-\sin\left(\theta_1\right)\sin\left(\theta_3\right)\right) \\ L\left(\sin\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)+\sin\left(\theta_3\right)\cos\left(\theta_1\right)\right) \\ L\sin\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right) \end{pmatrix} - \begin{pmatrix} L\cos\left(\theta_1\right) \\ L\sin\left(\theta_1\right) \\ 0 \end{pmatrix} \right) \\
&= \begin{pmatrix} -L\sin\left(\theta_2\right)\cos\left(\theta_1\right)\left(\cos\left(\theta_3\right)+1\right) \\ -L\sin\left(\theta_1\right)\sin\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right) \\ L\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right) \end{pmatrix} \\
J_{v_3} &= z_2^0 \times (o_3^0 - o_2^0) \\
&= \begin{pmatrix} \sin\left(\theta_2\right)\left(-\cos\left(\theta_1\right)\right) \\ -\sin\left(\theta_1\right)\sin\left(\theta_2\right) \\ \cos\left(\theta_2\right) \end{pmatrix} \times \left(\begin{pmatrix} L\left(\cos\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)-\sin\left(\theta_1\right)\sin\left(\theta_3\right)\right) \\ L\left(\sin\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)+\sin\left(\theta_3\right)\cos\left(\theta_1\right)\right) \\ L\sin\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right) \end{pmatrix} - \begin{pmatrix} L\cos\left(\theta_1\right)\left(\cos\left(\theta_2\right)+1\right) \\ L\sin\left(\theta_1\right)\left(\cos\left(\theta_2\right)+1\right) \\ L\sin\left(\theta_2\right) \end{pmatrix} \right) \\
&= \begin{pmatrix} -L\left(\sin\left(\theta_1\right)\cos\left(\theta_3\right)+\sin\left(\theta_3\right)\cos\left(\theta_1\right)\cos\left(\theta_2\right)\right) \\ L\left(\cos\left(\theta_1\right)\cos\left(\theta_3\right)-\sin\left(\theta_1\right)\sin\left(\theta_3\right)\cos\left(\theta_2\right)\right) \\ -L\sin\left(\theta_2\right)\sin\left(\theta_3\right) \end{pmatrix} \\
J_{w_1} &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
J_{w_2} &= \begin{pmatrix} \sin\left(\theta_1\right) \\ -\cos\left(\theta_1\right) \\ 0 \end{pmatrix} \\
J_{w_3} &= \begin{pmatrix} \sin\left(\theta_2\right)\left(-\cos\left(\theta_1\right)\right) \\ -\sin\left(\theta_1\right)\sin\left(\theta_2\right) \\ \cos\left(\theta_2\right) \end{pmatrix}
\end{aligned}
$$

Hence the Jacobian becomes

$$J_0 = \begin{pmatrix} -L\left(\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)\sin\left(\theta_1\right)+\cos\left(\theta_1\right)\sin\left(\theta_3\right)\right) & -L\cos\left(\theta_1\right)\left(\cos\left(\theta_3\right)+1\right)\sin\left(\theta_2\right) & -L\left(\cos\left(\theta_3\right)\sin\left(\theta_1\right)+\cos\left(\theta_1\right)\cos\left(\theta_2\right)\right. \\ L\left(\cos\left(\theta_1\right)\left(\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right)+1\right)-\sin\left(\theta_1\right)\sin\left(\theta_3\right)\right) & -L\left(\cos\left(\theta_3\right)+1\right)\sin\left(\theta_1\right)\sin\left(\theta_2\right) & L\left(\cos\left(\theta_1\right)\cos\left(\theta_3\right)-\cos\left(\theta_2\right)\sin\left(\theta_1\right)\right. \\ 0 & L\cos\left(\theta_2\right)\left(\cos\left(\theta_3\right)+1\right) & -L\sin\left(\theta_2\right)\sin\left(\theta_3\right) \\ 0 & \sin\left(\theta_1\right) & -\cos\left(\theta_1\right)\sin\left(\theta_2\right) \\ 0 & -\cos\left(\theta_1\right) & -\sin\left(\theta_1\right)\sin\left(\theta_2\right) \\ 1 & 0 & \cos\left(\theta_2\right) \end{pmatrix}$$

Or in short notation

$$\boxed{J_0 = \begin{pmatrix} -L\left(\left(C_2\left(C_3+1\right)+1\right)S_1+C_1S_3\right) & -LC_1\left(C_3+1\right)S_2 & -L\left(C_3S_1+C_1C_2S_3\right) \\ L\left(C_1\left(C_2\left(C_3+1\right)+1\right)-S_1S_3\right) & -L\left(C_3+1\right)S_1S_2 & L\left(C_1C_3-C_2S_1S_3\right) \\ 0 & LC_2\left(C_3+1\right) & -LS_2S_3 \\ 0 & S_1 & -C_1S_2 \\ 0 & -C_1 & -S_1S_2 \\ 1 & 0 & C_2 \end{pmatrix}}$$

When $L = 1$ the above becomes

$$\boxed{J_0 = \begin{pmatrix} -\left(C_2\left(C_3+1\right)+1\right)S_1-C_1S_3 & -C_1\left(C_3+1\right)S_2 & -C_3S_1-C_1C_2S_3 \\ C_1\left(C_2\left(C_3+1\right)+1\right)-S_1S_3 & -\left(C_3+1\right)S_1S_2 & C_1C_3-C_2S_1S_3 \\ 0 & C_2\left(C_3+1\right) & -S_2S_3 \\ 0 & S_1 & -C_1S_2 \\ 0 & -C_1 & -S_1S_2 \\ 1 & 0 & C_2 \end{pmatrix}}$$

Evaluate required joint torques for the configurations

Using the duality property where

$$x' = J_0 q'$$
$$\tau = J^T f$$

We can determine τ for each Jacobian at each configuration. The following table gives the result of this computation

#	J_0	$J^T f$	
$\begin{pmatrix} 0 \\ 30 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -L & 0 \\ \left(1+\sqrt{3}\right) L & 0 & L \\ 0 & \sqrt{3}\, L & 0 \\ 0 & 0 & -\frac{1}{2} \\ 0 & -1 & 0 \\ 1 & 0 & \frac{\sqrt{3}}{2} \end{pmatrix}$	$\begin{pmatrix} 0 & \left(1+\sqrt{3}\right) L & 0 & 0 & 0 & 1 \\ -L & 0 & \sqrt{3}\, L & 0 & -1 & 0 \\ 0 & L & 0 & -\frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{pmatrix}$	$\begin{pmatrix} \tau \\ -L \\ \frac{\sqrt{3}}{2} \end{pmatrix}$
$\begin{pmatrix} 0 \\ 0 \\ 30 \end{pmatrix}$	$\begin{pmatrix} -\frac{L}{2} & 0 & -\frac{L}{2} \\ \left(2+\frac{\sqrt{3}}{2}\right) L & 0 & \frac{\sqrt{3}\, L}{2} \\ 0 & \left(1+\frac{\sqrt{3}}{2}\right) L & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{L}{2} & \left(2+\frac{\sqrt{3}}{2}\right) L & 0 & 0 & 0 & 1 \\ 0 & 0 & \left(1+\frac{\sqrt{3}}{2}\right) L & 0 & -1 & 0 \\ -\frac{L}{2} & \frac{\sqrt{3}\, L}{2} & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{pmatrix}$	$\begin{pmatrix} \tau_z - \\ \\ \tau_z - \end{pmatrix}$
$\begin{pmatrix} 0 \\ 90 \\ 90 \end{pmatrix}$	$\begin{pmatrix} -L & -L & 0 \\ L & 0 & 0 \\ 0 & 0 & -L \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -L & L & 0 & 0 & 0 & 1 \\ -L & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -L & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{pmatrix}$	$\begin{pmatrix} \tau_z - \\ -L \\ \end{pmatrix}$

Discussion of result

Since

$$f = \begin{pmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{pmatrix}$$

then only entries in the J^T matrix which are not zero at location $(i, 1)$ and $(i, 6)$ where i is the row number of J^T will contribute to the joint torques. So for configuration 1 above, we see that J^T has non zero value in one of these locations in each row. But for configuration 2, the second row of J^T has zero in both the first column and the last column. This means no matter what joint torque is applied to joint 2, it will have no effect on end effector forces produced.

Similarly, for the third configuration, we see that the last row of J^T also has zero entry in the first and last column. This means no matter what joint torque is applied to joint 3, it will have no effect on end effector forces produced when in this configuration.

Evaluate linear velocity Jacobian J_v^1 expressed in frame 1

To find J_0^1 we need to transform J_0 to frame 1 using $J_0^i = \begin{pmatrix} R_0^i & 0 \\ 0 & R_0^i \end{pmatrix} J_0$ where in this case $i = 1$. But $R_0^1 = (R_1^0)^{-1} = (R_1^0)^T$. We found T_1^0 from above, so we can extract R_1^0 part from it

$$T_1^0 = \begin{pmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & L\cos(\theta_1) \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & L\sin(\theta_1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_1^0 = \begin{pmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) \\ \sin(\theta_1) & 0 & -\cos(\theta_1) \\ 0 & 1 & 0 \end{pmatrix}$$

Hence

$$\begin{pmatrix} R_0^i & 0 \\ 0 & R_0^i \end{pmatrix} = \begin{pmatrix} \cos(\theta_1) & \sin(\theta_1) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \sin(\theta_1) & -\cos(\theta_1) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(\theta_1) & \sin(\theta_1) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \sin(\theta_1) & -\cos(\theta_1) & 0 \end{pmatrix}$$

mutliplying the above with J_0 found earlier gives

$$\begin{aligned} J_0^1 &= \begin{pmatrix} R_0^i & 0 \\ 0 & R_0^i \end{pmatrix} J_0 \\ &= \begin{pmatrix} -L\sin(\theta_3) & -L(\cos(\theta_3)+1)\sin(\theta_2) & -L\cos(\theta_2)\sin(\theta_3) \\ 0 & L\cos(\theta_2)(\cos(\theta_3)+1) & -L\sin(\theta_2)\sin(\theta_3) \\ -L(\cos(\theta_2)(\cos(\theta_3)+1)+1) & 0 & -L\cos(\theta_3) \\ 0 & 0 & -\sin(\theta_2) \\ 1 & 0 & \cos(\theta_2) \\ 0 & 1 & 0 \end{pmatrix} \end{aligned}$$

Hence $J_{0_v}^1$ is the top three rows given by

$$\boxed{J_{0_v}^1 = \begin{pmatrix} -L\sin(\theta_3) & -L(\cos(\theta_3)+1)\sin(\theta_2) & -L\cos(\theta_2)\sin(\theta_3) \\ 0 & L\cos(\theta_2)(\cos(\theta_3)+1) & -L\sin(\theta_2)\sin(\theta_3) \\ -L(\cos(\theta_2)(\cos(\theta_3)+1)+1) & 0 & -L\cos(\theta_3) \end{pmatrix}}$$

When $L = 1$ the above becomes

$$J_{0_v}^1 = \begin{pmatrix} -\sin(\theta_3) & -\cos(\theta_3)\sin(\theta_2) - \sin(\theta_2) & -\cos(\theta_2)\sin(\theta_3) \\ 0 & \cos(\theta_3)\cos(\theta_2) + \cos(\theta_2) & -\sin(\theta_2)\sin(\theta_3) \\ -\cos(\theta_3)\cos(\theta_2) - \cos(\theta_2) - 1 & 0 & -\cos(\theta_3) \end{pmatrix}$$

Find the singularities in J_v^1 and sketch the arm

The singularity of J_v^1 can be found mathematically by finding the conditions under which the determinant of J_v^1 is zero, or the conditions under which one of the eigevalues become zero. Or we can use geometry and consider the cases where the arm is in singular direction. Mathematically, the determinant of J_v^1 is

$$|J_v^1| = -L^3 \sin(\theta_3)(\cos(\theta_2) + 1)(\cos(\theta_3) + 1)$$

When $L = 1$ the above becomes

$$|J_v^1| = -(1 + \cos\theta_2)(1 + \cos\theta_3)\sin\theta_3$$

To make $|J_v^1|$, the above implies the corresponding joint angles have to be the following

$$\theta_2 = \pm\pi$$
$$\theta_3 = \{\pm\pi, 0\}$$

The joint angle θ_1 can be any value since it does not contribute to making the determinant zero since the determinant does not depend on θ_1. The above shows there are a total of 5 configurations that will result in singularity.

The following diagram illustrates the above singularites found and also sketches the the singular direction.

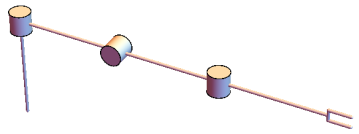
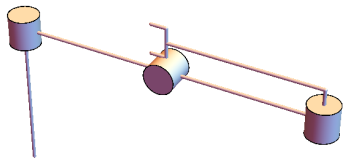
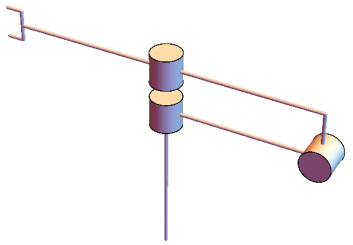
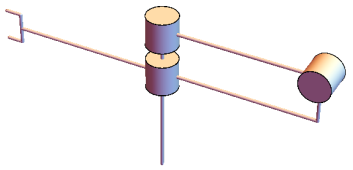
θ_1	θ_2	θ_3	J_v^1	configuration showing singular direction
any	any	0	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 2L & 0 \\ -3L & 0 & -L \end{pmatrix}$	
any	any	$\pm\pi$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -L & 0 & L \end{pmatrix}$	
any	π	any	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -2L & 0 \\ L & 0 & -L \end{pmatrix}$	
any	$-\pi$	any	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -2L & 0 \\ L & 0 & -L \end{pmatrix}$	

Figure 2.14: Singular directions for problem 7

Find the linear analytic Jacobian J_a for new representation

The analytical linear velocity Jacobian J_a is given by

$$J_a = E_p(x_p) J_0$$

Where $E_p(x_p)$ is the representation matrix found from

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = E_p(x_p) \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Since we are given the expressions for u, v, w, then we differentiate them w.r.t time to determine $E_p(x_p)$ and obtain

$$\begin{aligned} \begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} &= \begin{pmatrix} 2x' + 4y' \\ x' + y' - z' \\ z' \end{pmatrix} \\ &= \begin{pmatrix} 2 & 3 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \end{aligned}$$

Hence

$$E_p(x_p) = \begin{pmatrix} 2 & 3 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore $J_a = E_p(x_p) J_0$ becomes, at the required angles

$$\boxed{J_a = \begin{pmatrix} L & -2L & 0 \\ 0 & -L & L \\ 0 & 0 & -L \end{pmatrix}}$$

And at $L = 1$

$$J_a = \begin{pmatrix} 1 & -2 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

2.2.8 Problem 8

Problem 8

► Evaluate the inverse kinematics to provide a functional relationship between the defined task and joint space displacements

$$\left(\text{i.e. } \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = f(x_e, y_e, z_e) \right)$$

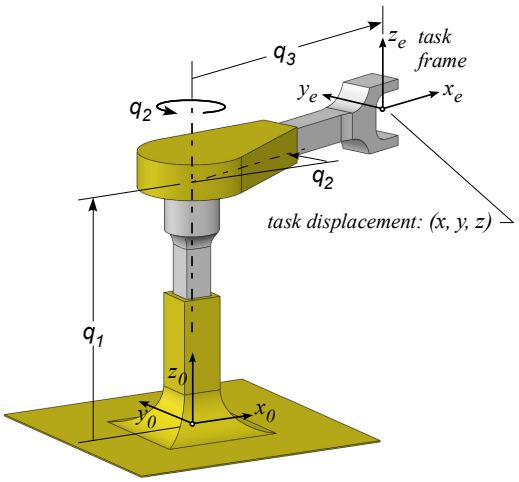
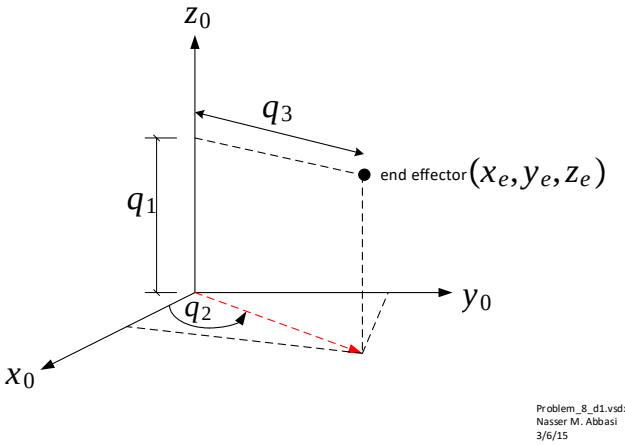


Figure 2.15: Problem 8 description

Using geometry as illustrated below



Hence based on the above diagram q_1 is found to be

$$q_1 = z_e$$

Taking the projection of the end effector vector on the xy plane gives

$$x_e = q_3 \cos(q_2)$$
$$y_e = q_3 \sin(q_2)$$

Dividing the second equation above by the first one gives $\frac{y_e}{x_e} = \tan(q_2)$. Hence $q_2 = \text{atantwo}(x_e, y_e)$. If $x = 0$ then there is no solution (singular direction). A second solution is

$q_2 = \text{atantwo}(x_e, y_e) + \pi$ and finally $q_3 = \sqrt{x_e^2 + y_e^2}$. Therefore the two solutions are

$$\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}_{\{1\}} = \begin{pmatrix} z_e \\ \text{atantwo}(x_e, y_e) \\ \sqrt{x_e^2 + y_e^2} \end{pmatrix} \qquad \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}_{\{2\}} = \begin{pmatrix} z_e \\ \text{atantwo}(x_e, y_e) + \pi \\ \sqrt{x_e^2 + y_e^2} \end{pmatrix}$$

2.2.9 Appendix

Source code for problem 1

```
axes[x_, y_, z_, f_, a_] :=
  Graphics3D[Join[{Arrowheads[a]},
    Arrow[{{0, 0, 0}, #}] & /@ {{x, 0, 0}, {0, y, 0}, {0, 0, z}},
     {Text[Style["x", FontSize -> Scaled[f]], {1.2*x, 0.1*y, 0.1*z}],
     Text[Style["y", FontSize -> Scaled[f]], {0.1 x, 1.2*y, 0.1*z}],
     Text[Style["z", FontSize -> Scaled[f]], {0.1*x, 0.1*y, 1.2*z}]}]];

rotZ[a_] := {{Cos[a], -Sin[a], 0}, {Sin[a], Cos[a], 0}, {0, 0, 1}};
rotX[a_] := {{1, 0, 0}, {0, Cos[a], -Sin[a]}, {0, Sin[a], Cos[a]}};
rotY[a_] := {{Cos[a], 0, Sin[a]}, {0, 1, 0}, {-Sin[a], 0, Cos[a]}};
obj = Cuboid[{1, -.5, -.1}, {-1, .5, .1}];

show[mat_, arrow_, fixed_] := Show[Graphics3D[
    {
     GeometricTransformation[{
       {Opacity[0.7], obj},
       {Red, Arrow[{{0, 0, 0}, {1.2, 0, 0}}]},
       {Red, Arrow[{{0, 0, 0}, {0, 1.2, 0}}]},
       {Red, Arrow[{{0, 0, 0}, {0, 0, 1.2}}]},
       Text[Style["x", Red, FontSize -> Scaled[0.07]], {1.4, 0, 0}],
       Text[Style["y", Red, FontSize -> Scaled[0.07]], {0, 1.4, 0}],
       Text[Style["z", Red, FontSize -> Scaled[0.07]], {0, 0, 1.4}],
       If[fixed, Sequence @@ {}, arrow]
       }, mat
      ],
     If[fixed, arrow, Sequence @@ {}]
     },
    Boxed -> False,
    Axes -> None,
    ViewPoint -> {3, 1.5, 1.5},
    SphericalRegion -> True,
    ImageSize -> 250, ImageMargins -> 0, ImagePadding -> 0, PlotRangePadding -> None],
   axes[1.6, 1.6, 1.5, 0.1, 0.04]];

show[mat_] := Show[Graphics3D[
   {
```

```
    GeometricTransformation[{
      {Opacity[0.7], obj},
      {Red, Arrow[{{0, 0, 0}, {1.2, 0, 0}}]},
      {Red, Arrow[{{0, 0, 0}, {0, 1.2, 0}}]},
      {Red, Arrow[{{0, 0, 0}, {0, 0, 1.2}}]},
      Text[Style["x", Red, FontSize -> Scaled[0.07]], {1.4, 0, 0}],
      Text[Style["y", Red, FontSize -> Scaled[0.07]], {0, 1.4, 0}],
      Text[Style["z", Red, FontSize -> Scaled[0.07]], {0, 0, 1.4}]
      }, mat
     ]
    },
   Boxed -> False,
   Axes -> None,
   ViewPoint -> {3, 1.5, 1.5},
   SphericalRegion -> True,
   ImageSize -> 200, ImageMargins -> 0, ImagePadding -> 0, PlotRangePadding -> None],
  axes[1.6, 1.6, 1.5, 0.1, 0.04]];

p1 = Grid[{
   {Style["+45 around fixed z", Bold, 14], Style["after rotation ", Bold, 14], Style["Final r
   {
    show[rotZ[0], makeCurvedArrow[.5, "z", 1], True],
    show[rotZ[45 Degree]],
    MatrixForm[N@rotZ[45 Degree]]
    }
   }, Spacings -> {1, 1}, Frame -> All, FrameStyle -> LightGray
  ]

p2 = Grid[{
   {Style["30 around current x", Bold, 14], Style["after rotation ", Bold, 14],
    Style["Final rotation matrix", Bold, 14]},
   {
    show[rotZ[45 Degree], makeCurvedArrow[.5, "x", 1], False],
    show[rotZ[45 Degree].rotX[30 Degree]],
    MatrixForm[N@rotZ[45 Degree].rotX[30 Degree]]
    }}, Spacings -> {1, 1}, Frame -> All, FrameStyle -> LightGray
  ]

p3 = Grid[{
   {Style["-45 around fixed z", Bold, 14], Style["after rotation ", Bold, 14],
    Style["Final rotation matrix", Bold, 14]},
   {
    show[rotZ[45 Degree].rotX[30 Degree], makeCurvedArrow[.5, "z", -1], True],
    show[rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree]],
    MatrixForm[N@rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree]]
    }}, Spacings -> {1, 1}, Frame -> All, FrameStyle -> LightGray
  ]
```

```
p4 = Grid[{
   {Style["+90 around current y", Bold, 14], Style["after rotation ", Bold, 14], Style["Final
   {
    show[rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree], makeCurvedArrow[.5, "y", 1], False
    show[rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree].rotY[90 Degree]],
    MatrixForm[N@rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree].rotY[90 Degree]]
    }}, Spacings -> {1, 1}, Frame -> All, FrameStyle -> LightGray
  ]

p5 = Grid[{
   {Style["-30 around fixed x", Bold, 14], Style["after rotation ", Bold, 14],
    Style["Final rotation matrix", Bold, 14]},
   {
    show[rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree].rotY[90 Degree], makeCurvedArrow[.5
    show[rotX[-30 Degree].rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree].rotY[90 Degree]],
    MatrixForm[Chop[N@rotX[-30 Degree].rotZ[-45 Degree].rotZ[45 Degree].rotX[30 Degree].rotY[
    }
   }, Spacings -> {1, 1}, Frame -> All, FrameStyle -> LightGray]
```

2.2.10 key solution for HW 2

-SOLUTION-

ME / ECE 739: Advanced Robotics Homework #2

Due: March 5th (Thursday)

Please submit your answers to the questions and all supporting work including your Matlab scripts, and, where appropriate, program results (plots, explanations). Your Matlab scripts should be readable, with comments, sensible variable names, indentation of code-block, etc. In addition to the hardcopy (pdf format), you must also submit your Matlab scripts electronically to the Learn@UW course page dropbox (e.g. Homework #2) using a zip archive file format. Please name your zip files using your last name and hw# (e.g. zinn_hw2.zip)

Problem 1.

Frame {A} and frame {B} are initially coincident. Frame {B} is rotated through the following sequence of rotations:

1. $+\alpha$ degree rotation about $\hat{z}_A$
2. $-\beta$ degree rotation about $\hat{x}_B$
3. $-\alpha$ degree rotation about $\hat{z}_A$
4. $+\gamma$ degree rotation about $\hat{y}_B$
5. $+\beta$ degree rotation about $\hat{x}_A$

where α = +45 degrees; β = -30 degrees; γ = +90 degrees

Evaluate the rotation transformation (matrix) that describes the orientation of frame {B} relative to frame {A} following this sequence of rotations.

Problem 3 [10 points]

Frame {A} and frame {B} are initially coincident. Frame {B} is rotated through the following sequence of rotations:

1. $+\alpha$ degree rotation about $\hat{z}_A$
2. $-\beta$ degree rotation about $\hat{x}_B$ current axis
3. $-\alpha$ degree rotation about $\hat{z}_A$ fixed axis
4. $+\gamma$ degree rotation about $\hat{y}_B$ current axis
5. $+\beta$ degree rotation about $\hat{x}_A$ fixed axis

where α = +45 degrees; β = -30 degrees; γ = +90 degrees

Evaluate the rotation transformation (matrix) that describes the orientation of frame {B} relative to frame {A} following this sequence of rotations.

(1) $R_{z,\alpha}$

(2) $R_{x,-\beta} = R_{x,\beta}^{-1} = R_{x,\beta}^{T}$

(3) $R_{z,-\alpha} = R_{z,\alpha}^{T}$

(4) $R_{y,\gamma}$

(5) $R_{x,\beta}$ (5th, 3rd, 1st, 2nd, 4th)

$$\left(R_{x,\beta}\right)\left(R_{z,\alpha}^{T}\right)\left(R_{z,\alpha}\right)\left(R_{x,\beta}^{T}\right)\left(R_{y,\gamma}\right)$$

$$\left(R_{x,\beta}\, I\, R_{x,\beta}^{T}\right)\left(R_{y,\gamma}\right)$$

$$I\, R_{y,\alpha} = R_{y,\gamma} = \begin{bmatrix} C_\gamma & 0 & S_\gamma \\ 0 & 1 & 0 \\ -S_\gamma & 0 & C_\gamma \end{bmatrix}$$

for $\gamma = 90°$

$$R_{y,\alpha} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} = R_B^A$$

ME / ECE 739: Advanced Robotics — Homework #2

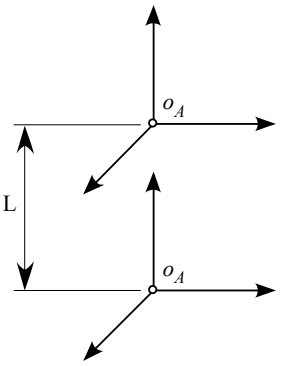
Due: March 5th (Thursday)

Problem 2.

The D-H parameters (d, a, α, and θ) and the homogeneous transformation which results (see below – also in Kinematics lecture notes) cannot be used to represent a *general* rigid-body transformation.

Homogeneous transformation matrix using DH convention:

$$T = \begin{bmatrix} c_\theta & -s_\theta c_\alpha & s_\theta s_\alpha & a c_\theta \\ s_\theta & c_\theta c_\alpha & -c_\theta s_\alpha & a s_\theta \\ 0 & s_\alpha & c_\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



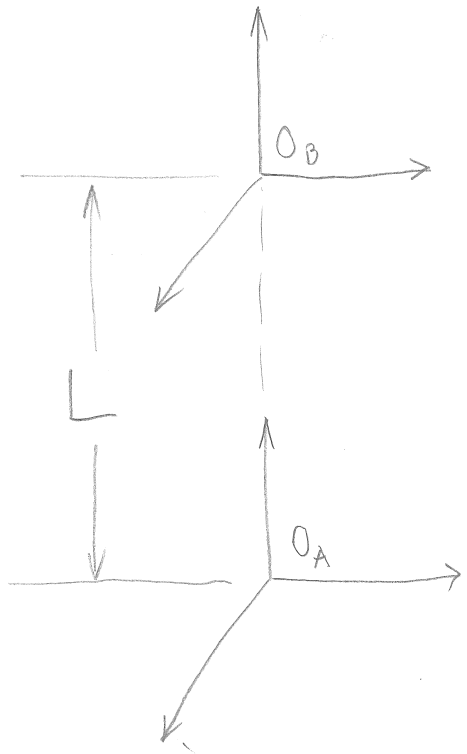
► Explain why this is the case. You can use physical and/or mathematical arguments.

► For the rigid-body transformation shown to the right, label the unit vectors of the two reference frames such that the D-H parameters cannot describe their relative transformation.

Problem #2

▷ The homogenous transformation is derived from four discrete motions ($Z_{rot} \rightarrow Z_{trans} \rightarrow X_{trans} \rightarrow X_{rot}$) while a general displacement requires 6 parameters to describe it

▷



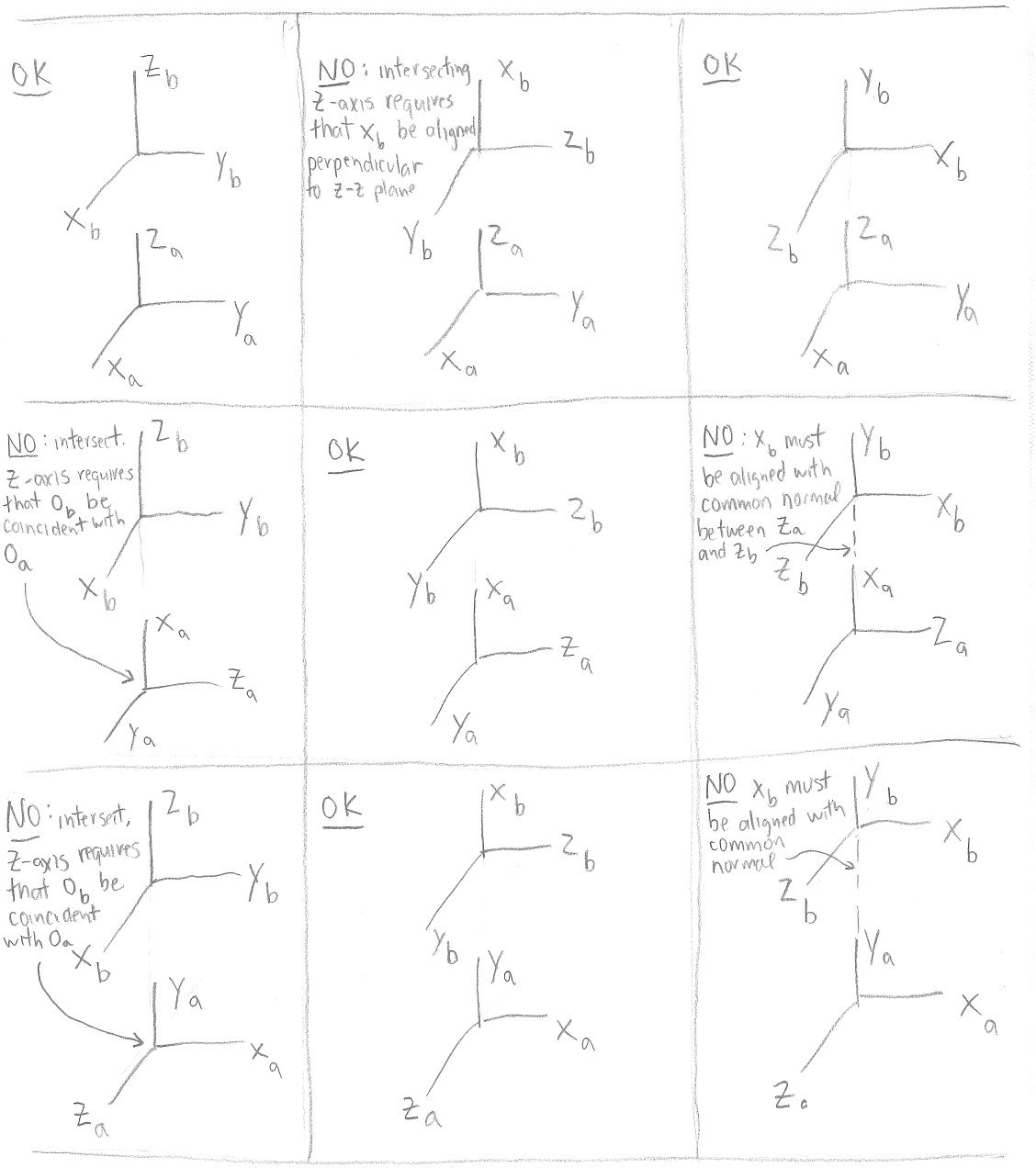
find a frame assignment which results in a situation where the transformation shown can't be expressed by the D-H convention

Recall that the frame assignment rules given for the D-H frames are required (constraints) such that the transformat. can be described with only 4 parameters.

Thus, we only need to find unit vector assignments which violate the rules

(2)

There are 9 possible cases to examine

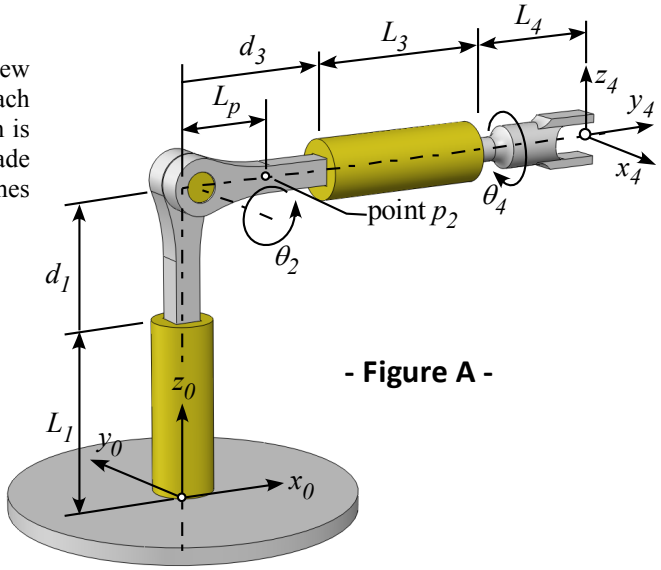


ME / ECE 739: Advanced Robotics Homework #2

Due: March 5th (Thursday)

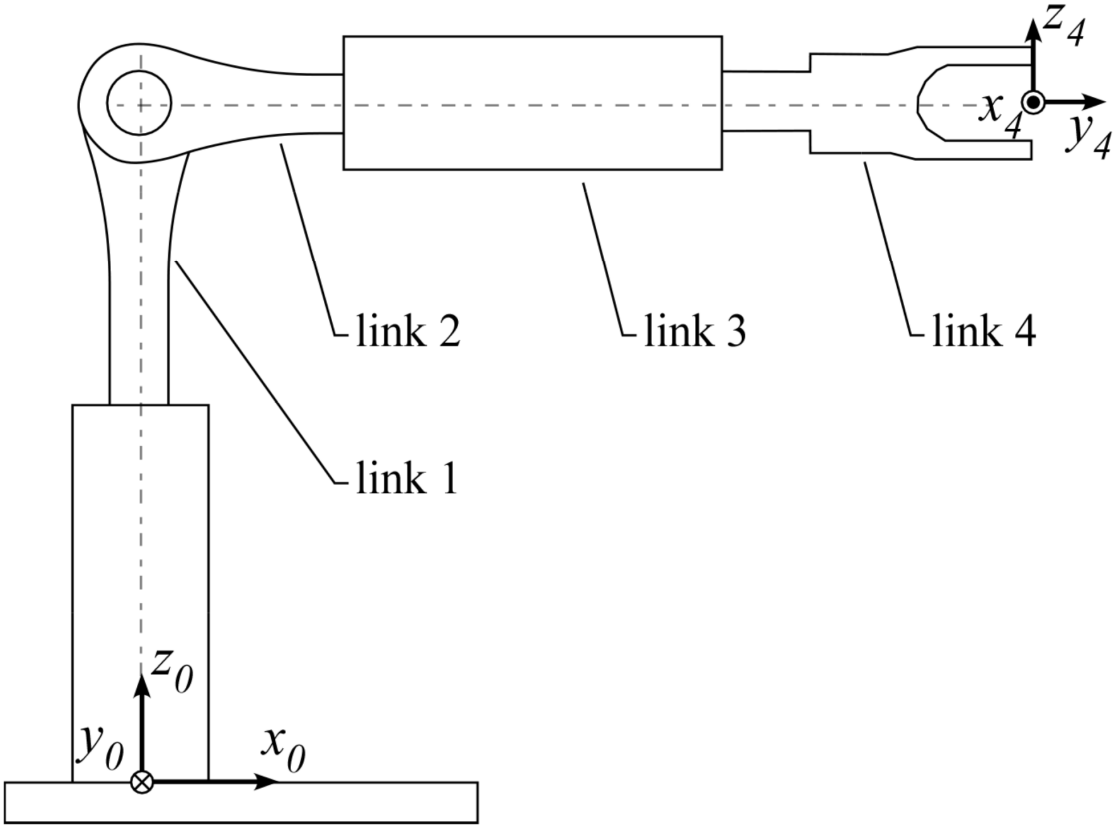
Problem 3.

► Sketch the DH frames on the planar view of the manipulator given below. For each frame, state whether the frame definition is unique and describe the choices you made in the table below. Use the defined frames {0} and {4} shown in Figure A.



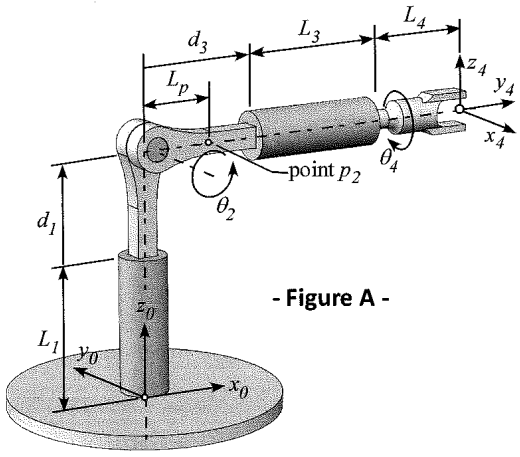
- Figure A -

Planar view of the manipulator:



Problem 5 [10 points]

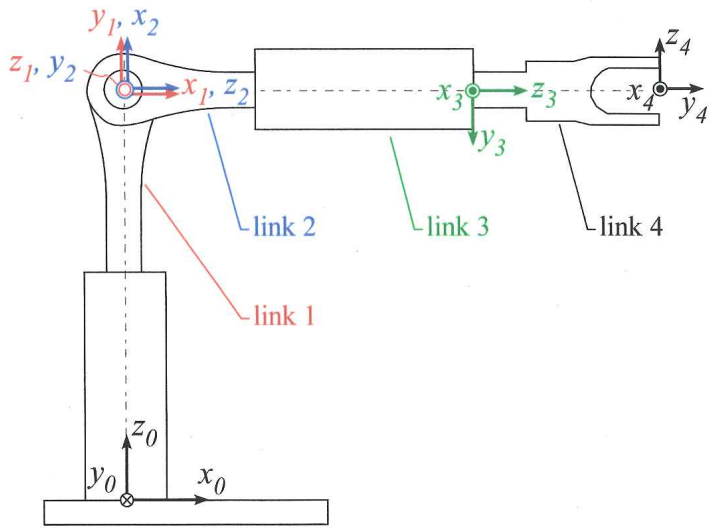
Sketch the DH frames on the planar view of the manipulator given on the *next* page. For each frame, state whether the frame definition is unique and describe the choices you made in the table below. Use the defined frames {0} and {4} shown in Figure A.



- Figure A -

	State whether the frame definition is unique and describe the choices you made.
Frame 1	▪ z_1-axis aligned with revolute joint of link 2. Can be directed into or out of the page. ▪ origin located at intersection of z_0 and z_1 ▪ x_1-axis perpendicular to z_0 and z_1. Can be directed to the right or to the left
Frame 2	▪ z_2-axis aligned with prismatic joint of link 3. Can be directed to the left or to the right. ▪ origin located at intersection of z_1 and z_2 ▪ x_2-axis perpendicular to z_1 and z_2. Can be directed up or directed down.
Frame 3	▪ z_3-axis aligned with revolute joint of link 4. Can be directed to the left or to the right. ▪ origin can be located anywhere along the z_3-axis (because z_3 and z_2 are collinear). In this case, it was located at the distal end of link 3. ▪ x_3-axis is perpendicular to z_3. – otherwise the direction is arbitrary. In this case it was directed out of the page (and perpendicular to x_4.

DH Frames:
Sketch the DH frames on the planar view of the manipulator given below



Problem 6 [15 points]

Evaluate the DH parameters for the manipulator and frame definitions developed in the previous problem. List the DH parameters in the table below.

i	θ_i	d_i	a_i	α_i
1	0	$L_1 + d_1$	0	$\pi/2$
2	$\pi/2 + \theta_2$	0	0	$\pi/2$
3	$\pi/2$	$L_3 + d_3$	0	0
4	θ_4	L_4	0	$\pi/2$

ME / ECE 739: Advanced Robotics Homework #2

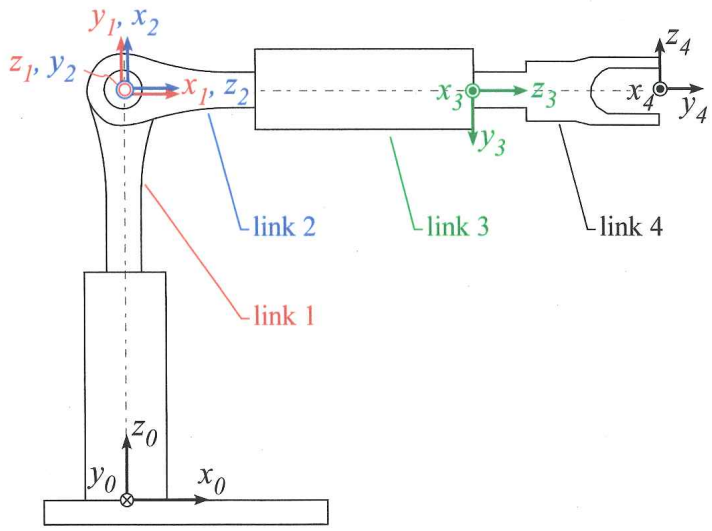
Due: March 5th (Thursday)

Problem 4.

► Evaluate the DH parameters for the manipulator and frame definitions developed in the previous problem. List the DH parameters in the table below.

i	θ_i	d_i	a_i	α_i
1				
2				
3				
4				

DH Frames:
Sketch the DH frames on the planar view of the manipulator given below



Problem 6 [15 points]

Evaluate the DH parameters for the manipulator and frame definitions developed in the previous problem. List the DH parameters in the table below.

i	θ_i	d_i	a_i	α_i
1	0	$L_1 + d_1$	0	$\pi/2$
2	$\pi/2 + \theta_2$	0	0	$\pi/2$
3	$\pi/2$	$L_3 + d_3$	0	0
4	θ_4	L_4	0	$\pi/2$

ME / ECE 739: Advanced Robotics — Homework #2

Due: March 5th (Thursday)

Problem 5

► Write a Matlab function which takes as its input the DH parameters and returns the associated 4x4 homogeneous transformation matrix, *T*. A possible function prototype is given below. To learn how to write a `function` in Matlab, type `help function` in the Matlab workspace.

```
function T = DH2T(d, a, alpha, theta);
```

► Write a Matlab script to verify that your function is working properly. The script should evaluate the *T* matrix for the set of DH parameters listed in the adjacent table. When your script is run, the *T* matrices should be printed to the workspace. This script, along with your function, are to be handed in to the course dropbox on the Learn@UW course page. As a possible starting point, your script might look like:

Link	d_i	a_i	α_i	θ_i
Case 1:	0	10	0	0
Case 2:	10	0	0	0
Case 3:	10	0	π	0
Case 4:	0	0	π	π

```
clear all; close all; clc
% Case 1:
d = 0;  a = 10; alpha = 0;  theta = 0;
T1 = DH2T(a,alpha,d,theta)
% Case 2:
d = 10; a = 0;  alpha = 0;  theta = 0;
T2 = DH2T(a,alpha,d,theta)
% Case 3:
d = 10; a = 0;  alpha = pi; theta = 0;
T3 = DH2T(a,alpha,d,theta)
% Case 4:
d = 0;  a = 0;  alpha = pi; theta = pi;
T4 = DH2T(a,alpha,d,theta)
```

Problem #3:

Testing Script:

```
clear all; close all; clc

% Case 1:
d = 0;  a = 10; alpha = 0;  theta = 0;
T1 = DH2T(a,alpha,d,theta)

% Case 2:
d = 10; a = 0;  alpha = 0;  theta = 0;
T2 = DH2T(a,alpha,d,theta)

% Case 3:
d = 10; a = 0;  alpha = pi; theta = 0;
T3 = DH2T(a,alpha,d,theta)

% Case 4:
d = 0;  a = 0;  alpha = pi; theta = pi;
T4 = DH2T(a,alpha,d,theta)
```

DH function (DH2T.m)**:**

```
function T = DH2T(a,alpha,d,theta)

sinTheta = sin(theta); cosTheta = cos(theta);
sinAlpha = sin(alpha); cosAlpha = cos(alpha);

T = [cosTheta  -sinTheta*cosAlpha   sinTheta*sinAlpha   a*cosTheta
     sinTheta   cosTheta*cosAlpha  -cosTheta*sinAlpha   a*sinTheta
        0           sinAlpha             cosAlpha            d
        0              0                    0                1];
```

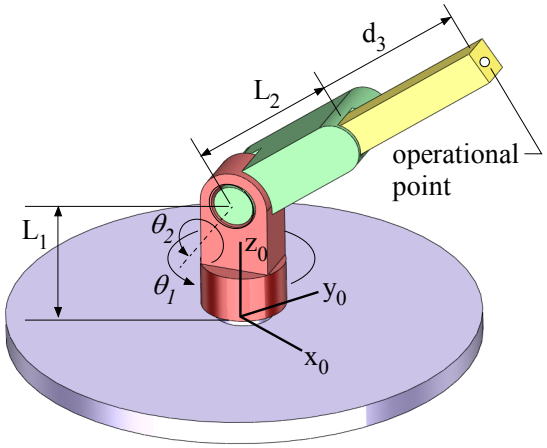
ME / ECE 739: Advanced Robotics — Homework #2

Due: March 5th (Thursday)

Problem 6.

For the manipulator shown, the forward kinematics are given as:

$$x_e = \cos\theta_1 \cos\theta_2 (L_2 + d_3)$$
$$y_e = \sin\theta_1 \cos\theta_2 (L_2 + d_3)$$
$$z_e = L_1 + \sin\theta_2 (L_2 + d_3)$$



► Develop the linear velocity Jacobian, J_v, using direct differentiation of the forward kinematics. Assume that the task and joint variable vectors are defined as

$$q = \begin{bmatrix} \theta_1 & \theta_2 & d_3 \end{bmatrix}^T$$
$$x = \begin{bmatrix} x_e & y_e & z_e \end{bmatrix}^T$$

► For the manipulator configuration defined by the joint vector, $q = \begin{bmatrix} \pi & \frac{1}{2}\pi & L_2 \end{bmatrix}^T$, evaluate the linear velocity of the end-effector given the joint velocity vector $\dot{q} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$.

► For what values of the joint variables is the linear velocity Jacobian, J_v, singular? Use physical and/or mathematical arguments to support your answer.

Problem #4

Fwd Kinematics (given)

$$x_e = \cos\theta_1 \cos\theta_2 (L_2 + d_3)$$
$$y_e = \sin\theta_1 \cos\theta_2 (L_2 + d_3)$$
$$z_e = L_1 + \sin\theta_2 (L_2 + d_3)$$

- linear velocity Jacobian

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{z}_e \end{bmatrix} = J \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{d}_3 \end{bmatrix}$$

where

$$J = \begin{bmatrix} \frac{\partial x_e}{\partial \theta_1} & \frac{\partial x_e}{\partial \theta_2} & \frac{\partial x_e}{\partial d_3} \\ \frac{\partial y_e}{\partial \theta_1} & \frac{\partial y_e}{\partial \theta_2} & \frac{\partial y_e}{\partial d_3} \\ \frac{\partial z_e}{\partial \theta_1} & \frac{\partial z_e}{\partial \theta_2} & \frac{\partial z_e}{\partial \theta_3} \end{bmatrix}$$

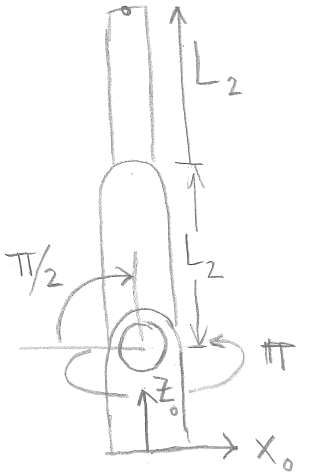
$$J = \left[\begin{array}{c|c|c} -c_2 s_1 (L_2 + d_3) & -c_1 s_2 (L_2 + d_3) & c_1 c_2 \\ \hline c_1 c_2 (L_2 + d_3) & -s_1 s_2 (L_2 + d_3) & c_2 s_1 \\ \hline 0 & c_2 (L_2 + d_3) & s_2 \end{array}\right]$$

note: I used Matlab's symbolic toolbox to evaluate the derivatives (see code attached)

▷ when $q = \begin{bmatrix} \pi \\ \frac{\pi}{2} \\ L_2 \end{bmatrix}$ the Jacobian is evaluated as

$$J = \begin{bmatrix} 0 & 2L_2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

note: the Jacobian is singular in this configuration



To evaluate the velocity

$$\dot{x} = J\dot{q}$$

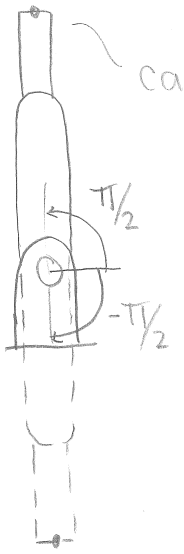
$$\dot{x} = \begin{bmatrix} 0 & 2L_2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

▷ The manipulator linear velocity Jacobian is singular when

$$\theta_2 = \pm \frac{\pi}{2} \left(\text{or } \frac{\pi}{2} + K\pi \right)$$

$$d_3 = -L_2$$

when $\theta_2 = \pm \frac{\pi}{2}$



can't move in/out of page

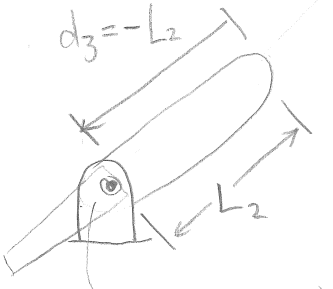
$$J = \begin{bmatrix} 0 & -C_1(L_2+d_3) & 0 \\ 0 & -S_1(L_2+d_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

↑ singular (rank = 2)

when $d_3 = -L_2$

$$J = \begin{bmatrix} 0 & 0 & C_1 C_2 \\ 0 & 0 & S_1 C_2 \\ 0 & 0 & S_2 \end{bmatrix}$$

↑ singular (rank = 1)



can't move in/out of page
or ⊥ to d_3 motion

```
% HW #2 - Problem $

clear all; close all; clc

% define variables as symbolic
syms theta1 theta2 d3 L1 L2 L3 xe ye ze

% forward kinematics
xe = cos(theta1)*cos(theta2)*(L2 + d3);
ye = sin(theta1)*cos(theta2)*(L2 + d3);
ze = L1 + sin(theta2)*(L2 + d3);

% evaluate the Jacobian (using diff function <- to differentiate)
J11 = diff(xe,theta1);
J12 = diff(xe,theta2);
J13 = diff(xe,d3);
J21 = diff(ye,theta1);
J22 = diff(ye,theta2);
J23 = diff(ye,d3);
J31 = diff(ze,theta1);
J32 = diff(ze,theta2);
J33 = diff(ze,d3);
J = [J11  J12  J13
     J21  J22  J23
     J31  J32  J33];
% print J to the workspace
pretty(J)

% evaluate the Jacobian - substituting in for the given values of q
J11e = subs(J11,{theta1,theta2,d3},[pi,pi/2,L2]);
J12e = subs(J12,{theta1,theta2,d3},[pi,pi/2,L2]);
J13e = subs(J13,{theta1,theta2,d3},[pi,pi/2,L2]);
J21e = subs(J21,{theta1,theta2,d3},[pi,pi/2,L2]);
J22e = subs(J22,{theta1,theta2,d3},[pi,pi/2,L2]);
J23e = subs(J23,{theta1,theta2,d3},[pi,pi/2,L2]);
J31e = subs(J31,{theta1,theta2,d3},[pi,pi/2,L2]);
J32e = subs(J32,{theta1,theta2,d3},[pi,pi/2,L2]);
J33e = subs(J33,{theta1,theta2,d3},[pi,pi/2,L2]);
Je = [J11e  J12e  J13e
      J21e  J22e  J23e
      J31e  J32e  J33e];
pretty(Je)

% evaluate the Jacobian for the singular configuration: theta2 = pi/2
J11e = subs(J11,{theta2},[pi/2]);
J12e = subs(J12,{theta2},[pi/2]);
```

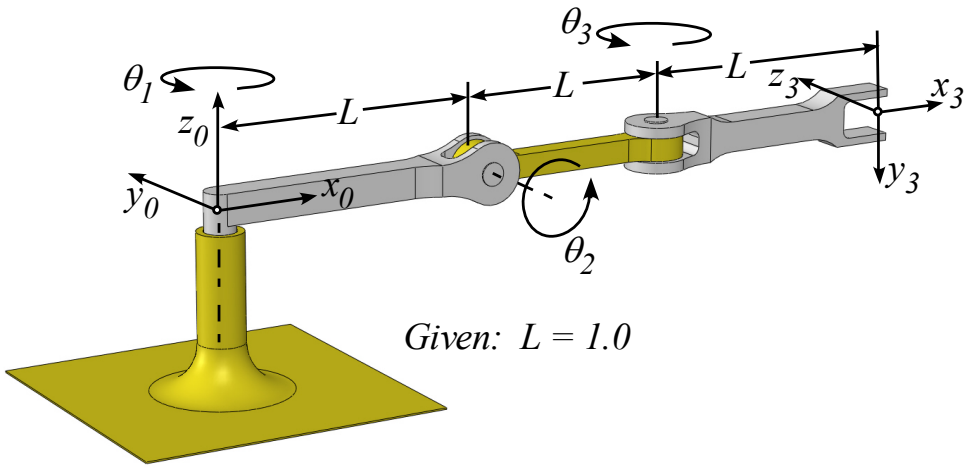
```
J13e = subs(J13,{theta2},[pi/2]);
J21e = subs(J21,{theta2},[pi/2]);
J22e = subs(J22,{theta2},[pi/2]);
J23e = subs(J23,{theta2},[pi/2]);
J31e = subs(J31,{theta2},[pi/2]);
J32e = subs(J32,{theta2},[pi/2]);
J33e = subs(J33,{theta2},[pi/2]);
Je = [J11e  J12e  J13e
      J21e  J22e  J23e
      J31e  J32e  J33e];
pretty(Je)

% evaluate the Jacobian for the singular configuration: d3 = -L2
J11e = subs(J11,{d3},[-L2]);
J12e = subs(J12,{d3},[-L2]);
J13e = subs(J13,{d3},[-L2]);
J21e = subs(J21,{d3},[-L2]);
J22e = subs(J22,{d3},[-L2]);
J23e = subs(J23,{d3},[-L2]);
J31e = subs(J31,{d3},[-L2]);
J32e = subs(J32,{d3},[-L2]);
J33e = subs(J33,{d3},[-L2]);
Je = [J11e  J12e  J13e
      J21e  J22e  J23e
      J31e  J32e  J33e];
pretty(Je)
```

ME / ECE 739: Advanced Robotics Homework #2

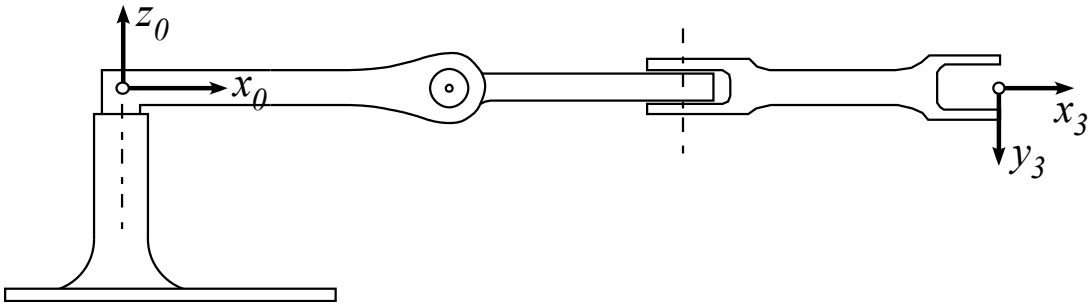
Due: March 5th (Thursday)

Problem 7.

Consider the three axis RRR manipulator shown in the figure below



► Derive the forward kinematics, T_3^0, of this manipulator as a function of the joint displacements and the geometric parameters shown in the figure. Sketch your intermediate frame definitions on the plane view of the manipulator shown below. Keep in mind that your frame definitions should be consistent with the conventions assumed when constructing the explicit form of the basic Jacobian.



► Evaluate the full basic Jacobian, J_0, for this manipulator. In this case, the basic Jacobian relates the joint space velocities, $\dot{q}$, and task space velocities $\dot{X}$.

$$\dot{X} = J_o \dot{q} \quad \text{where } \dot{X} = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \omega_x \quad \omega_y \quad \omega_z]^T \text{ and } \dot{q} = [\dot{\theta}_1 \quad \dot{\theta}_2 \quad \dot{\theta}_3]^T$$

(Problem continued on next page)

ME / ECE 739: Advanced Robotics Homework #2

Due: March 5th (Thursday)

► For the manipulator configurations listed below, evaluate the required joint torques to react the applied end-effector forces and torques: $F = [f_x \quad 0 \quad 0 \quad 0 \quad 0 \quad \tau_z]^T$. Discuss you results.

Configuration	θ_1	θ_2	θ_3
1	0	30°	0
2	0	0	30°
3	0	90°	90°

► Evaluate the linear velocity Jacobian, J_v^1, expressed in frame {1}

► Using J_v^1 find the singularities of the manipulator (with respect to the end-effector's linear velocity)

► For each type of singularity that you identified explain the physical interpretation of the singularity - by sketching the arm in a singular configuration and describing the resulting limitation on its movement.

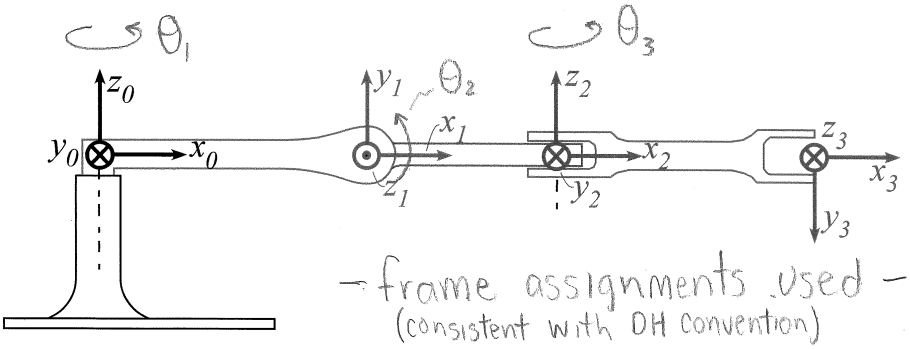
► For the manipulator above, a new task position representation has been defined as

$$u = 2x + 3y$$
$$v = x + y - z$$
$$w = z$$

Evaluate the *linear* velocity analytical Jacobian, J_a, for this new representation where

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = J_a \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}$$

Problem 1



- frame assignments used -
(consistent with DH convention)

by inspection we can derive the transformations between frames

$$T_1^0 = \left[\begin{array}{ccc|c} C_1 & 0 & S_1 & LC_1 \\ S_1 & 0 & -C_1 & LS_1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^1 = \left[\begin{array}{ccc|c} C_2 & 0 & -S_2 & LC_2 \\ S_2 & 0 & C_2 & LS_2 \\ 0 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad T_3^2 = \left[\begin{array}{ccc|c} C_3 & 0 & -S_3 & LC_3 \\ S_3 & 0 & C_3 & LS_3 \\ 0 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

To evaluate the basic Jacobian we need

T_3^0 , T_2^0 and T_1^0

$$T_2^0 = T_1^0\, T_2^1 = \left[\begin{array}{ccc|c} C_1C_2 & -S_1 & -C_1S_2 & LC_1(1+C_2) \\ S_1C_2 & C_1 & -S_1S_2 & LS_1(1+C_2) \\ S_2 & 0 & C_2 & LS_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_3^0 = T_2^0\, T_3^2$$

note: I used Matlab's symbolic toolbox for many of these calculations

$$= \left[\begin{array}{ccc|c} C_1C_2C_3 - S_1S_3 & C_1S_2 & -S_1C_3 - C_1C_2S_3 & L(C_1 + C_1C_2 - S_1S_3 + C_1C_2C_3) \\ C_1S_3 + S_1C_2C_3 & S_1S_2 & C_1C_3 - S_1C_2S_3 & L(S_1 + S_1C_2 + C_1S_3 + S_1C_2C_3) \\ S_2C_3 & -C_2 & -S_2S_3 & L(S_2 + S_2C_3) \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The basic Jacobian for this system is evaluated as

$$J^0 = \left[\begin{array}{c|c|c} \hat{Z}_0^0 \times (O_3^0 - O_0^0) & \hat{Z}_1^0 \times (O_3^0 - O_1^0) & \hat{Z}_2^0 \times (O_3^0 - O_2^0) \\ \hat{Z}_0^0 & \hat{Z}_1^0 & \hat{Z}_2^0 \end{array}\right]$$

evaluating terms

$$\hat{Z}_0^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \hat{Z}_1^0 \Rightarrow \underbrace{\left[\begin{array}{ccc|c} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ \hline 0 & 0 & 0 & 1 \end{array}\right]}_{T_1^0} = \begin{bmatrix} S_1 \\ C_1 \\ 0 \end{bmatrix}$$

$$\hat{Z}_2^0 = \underbrace{\left[\begin{array}{ccc|c} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ \hline 0 & 0 & 0 & 1 \end{array}\right]}_{T_2^0} = \begin{bmatrix} -C_1 S_2 \\ -S_1 S_2 \\ C_2 \end{bmatrix}$$

$$O_0^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad O_1^0 \Rightarrow \left.\left[\begin{array}{ccc|c} & R_1^0 & & x \\ & & & x \\ & & & x \\ \hline 0 & 0 & 0 & 1 \end{array}\right]\right\} T_1^0$$

$$O_1^0 = \begin{bmatrix} L C_1 \\ L S_1 \\ 0 \end{bmatrix}$$

$$O_2^0 \Rightarrow \underbrace{\left[\begin{array}{ccc|c} & R_2^0 & & x \\ & & & x \\ & & & x \\ \hline 0 & 0 & 0 & 1 \end{array}\right]}_{T_2^0} \qquad O_2^0 = \begin{bmatrix} L(C_1 + C_1 C_2) \\ L(S_1 + S_1 C_2) \\ L S_2 \end{bmatrix}$$

$$O_3^0 \Rightarrow \left.\left[\begin{array}{ccc|c} & R_3^0 & & x \\ & & & x \\ & & & x \\ \hline 0 & 0 & 0 & 1 \end{array}\right]\right\} T_3^0 \qquad O_3^0 = \begin{bmatrix} L(C_1 + C_1 C_2 - S_1 S_3 + C_1 C_2 C_3) \\ L(S_1 + S_1 C_2 + C_1 S_3 + S_1 C_2 C_3) \\ L(S_2 + S_2 C_3) \end{bmatrix}$$

evaluating J^0

$$J^0 = \left[\begin{array}{c|c|c} \hat{Z}_0^0 \times (O_3^0 - O_0^0) & \hat{Z}_1^0 \times (O_3^0 - O_1^0) & \hat{Z}_2^0 \times (O_3^0 - O_2^0) \\ \hat{Z}_0^0 & \hat{Z}_1^0 & \hat{Z}_2^0 \end{array}\right]$$

$$J^0 = \left[\begin{array}{c|c|c} -L(s_1 + s_1c_2 + c_1s_3 + s_1c_2c_3) & -Lc_1s_2(c_3 + 1) & -L(s_1c_3 + c_1c_2s_3) \\ L(c_1 + c_1c_2 - s_1s_3 + c_1c_2c_3) & -Ls_1s_2(c_3 + 1) & L(c_1c_3 - s_1c_2s_3) \\ 0 & Lc_2(c_3 + 1) & -Ls_2s_3 \\ \hline 0 & s_1 & -c_1s_2 \\ 0 & -c_1 & -s_1s_2 \\ 1 & 0 & c_2 \end{array}\right]$$

To evaluate the required joint torques given the applied end-effector forces

$$\tau = J^T F \qquad F = [f_x \; 0 \; 0 \; 0 \; 0 \; \tau_z]^T$$

at configuration 1 $(\theta_1 = 0, \theta_2 = 30^\circ, \theta_3 = 0)$

$$J = \left[\begin{array}{c|c|c} 0 & -L & 0 \\ (\sqrt{3}+1)L & 0 & L \\ 0 & \sqrt{3}L & 0 \\ \hline 0 & 0 & -1/2 \\ 0 & -1 & 0 \\ 1 & 0 & \sqrt{3}/2 \end{array}\right]$$

evaluating $\tau = J^T F$

$$\tau = \begin{bmatrix} 0 & (\sqrt{3}+1)L & 0 & 0 & 0 & 1 \\ -L & 0 & \sqrt{3}L & 0 & -1 & 0 \\ 0 & L & 0 & -1/2 & 0 & \sqrt{3}/2 \end{bmatrix} \begin{bmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{bmatrix} = \begin{bmatrix} \tau_z \\ -f_x L \\ \frac{\sqrt{3}}{2}\tau_z \end{bmatrix}$$

these columns don't factor into the calculation

at configuration 2 $(\theta_1 = 0, \theta_2 = 0, \theta_3 = 30°)$

$$J = \left[\begin{array}{c|c|c} -\frac{1}{2}L & 0 & -\frac{1}{2}L \\ L\left(\frac{\sqrt{3}}{2}+2\right) & 0 & \frac{\sqrt{3}}{2}L \\ 0 & L\left(\frac{\sqrt{3}}{2}+1\right) & 0 \\ \hline 0 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \end{array}\right]$$

evaluating $\tau = J^T F$

$$\tau = \begin{bmatrix} -\frac{1}{2}L & \cdots & 1 \\ 0 & & 0 \\ -\frac{1}{2}L & & 1 \end{bmatrix} \begin{bmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}L f_x + \tau_z \\ 0 \\ -\frac{1}{2}L f_x + \tau_z \end{bmatrix}$$

at configuration 3 ($\theta_1 = 0, \theta_2 = 90, \theta_3 = 90$)

$$J = \begin{bmatrix} -L & -L & 0 \\ L & 0 & 0 \\ 0 & 0 & -L \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

evaluating $\tau = J^T F$

$$\tau = \left[\begin{array}{c|c|c} -L & & 1 \\ -L & \cdots & 0 \\ 0 & & 0 \end{array}\right] \begin{bmatrix} f_x \\ 0 \\ 0 \\ 0 \\ 0 \\ \tau_z \end{bmatrix} = \begin{bmatrix} -L f_x + \tau_z \\ -L f_x \\ 0 \end{bmatrix}$$

▲ To find $J_v^1 = R_0^1 J_v^0$

$$R_0^1 = R_1^{0\,T} \Rightarrow T_1^0 \left[\begin{array}{c|c} R_1^0 & \begin{matrix} x \\ y \\ z \end{matrix} \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$$

$$R_0^1 = \begin{bmatrix} C_1 & S_1 & 0 \\ 0 & 0 & 1 \\ S_1 & -C_1 & 0 \end{bmatrix}$$

evaluating $J_v^1 = R_0^1 J_v^0$ leads to:

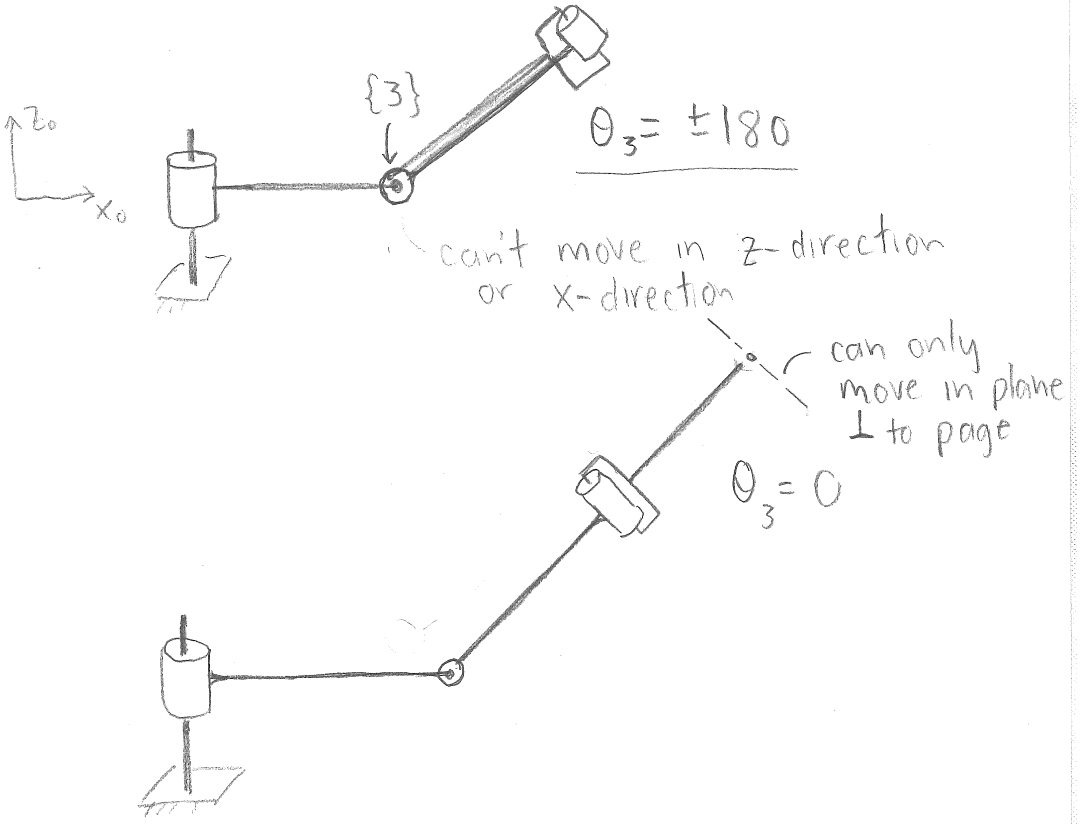
$$J_v^1 = \begin{bmatrix} -Ls_3 & -Ls_2(c_3 + 1) & -Lc_2s_3 \\ 0 & Lc_2(c_3 + 1) & -Ls_2s_3 \\ -L(c_2 + c_2c_3 + 1) & 0 & -Lc_3 \end{bmatrix}$$

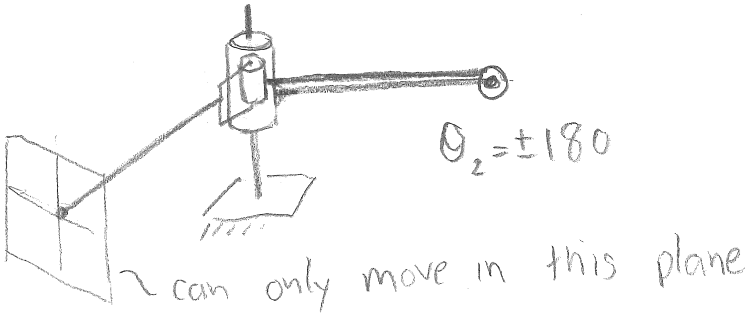
the singular configurations can be found by examining the determinant of J_v^1

$$\det(J_v^1) = -L^3 s_3(c_2 + 1)(c_3 + 1) = 0$$

the singularities occur at

- $S_3 = 0 \Rightarrow \theta_3 = 0°, 180°$
- $C_2 + 1 = 0$
  $C_2 = -1 \Rightarrow \theta_2 = \pm 180°$
- $C_3 + 1$
  $C_3 = -1 \Rightarrow \theta_3 = \pm 180°$


z_0
x_0
{3}
$\theta_3 = \pm 180$
can't move in z-direction or x-direction
can only move in plane ⊥ to page
$\theta_3 = 0$


$\theta_2 = \pm 180$
can only move in this plane

task description

$$u = 2x + 3y \quad (1)$$
$$v = x + y - z \quad (2)$$
$$w = z \quad (3)$$

Analytical Jacobian

$$J_a = E \cdot J_0$$

↑ basic Jacobian
representation Jacobian

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = J_a \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_n \end{bmatrix} = E J_0 \dot{q}$$

$$E = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial z} \end{bmatrix}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = E \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

↑ evaluate from (1), (2), and (3)

$$E = \begin{bmatrix} 2 & 3 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

the basic Jacobian (reduced - 1st 3 rows), where $\theta_1 = 0$ and $\theta_2 = \theta_3 = \pi/2$ evaluates (using result from earlier)

$$J_0 = \begin{bmatrix} -L & -L & 0 \\ L & 0 & 0 \\ 0 & 0 & -L \end{bmatrix} \quad \text{and} \quad J_a = E J_0 = \begin{bmatrix} L & -2L & 0 \\ 0 & -L & L \\ 0 & 0 & -L \end{bmatrix}$$

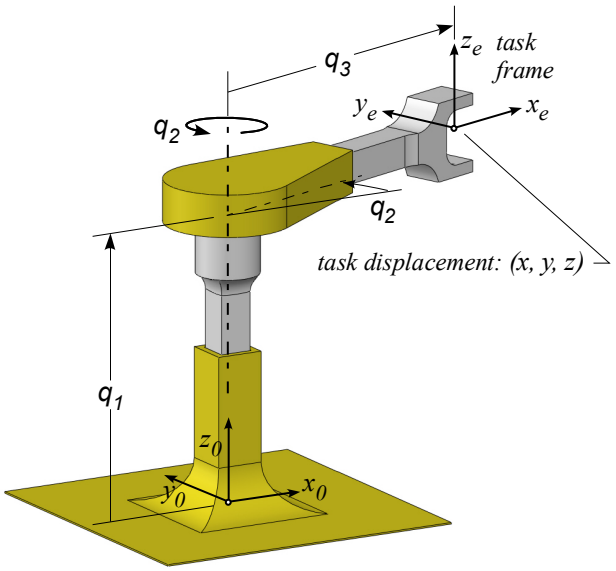
ME / ECE 739: Advanced Robotics Homework #2

Due: March 5th (Thursday)

Problem 8

► For the three degree-of-freedom PRP manipulator shown, evaluate the inverse kinematics to provide a functional relationship from the defined task to the joint space displacements

(i.e. $\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = f(x_e, y_e, z_e)$)



Forward kinematics:

$$x_e = q_3 \cos q_2 \quad (1)$$

$$y_e = q_3 \sin q_2 \quad (2)$$

$$z_e = q_1 \quad (3)$$

Inverse kinematics

from (3) $q_1 = z_e$ ⇦

from (1) $\cos q_2 = x_e / q_3$

from (2) $\sin q_2 = y_e / q_3$

thus $q_2 = \text{atan2}(x_e, y_e)$ ⇦

square (1) and (2) and sum

$$x_e^2 + y_e^2 = q_3^2 \cos^2 q_2 + q_3^2 \sin^2 q_2 = q_3^2$$

$$q_3 = \sqrt{x_e^2 + y_e^2}$$ ⇦

$$q_1 = z_e$$

$$q_2 = \text{atan2}(x_e, y_e)$$

$$q_3 = \sqrt{x_e^2 + y_e^2}$$

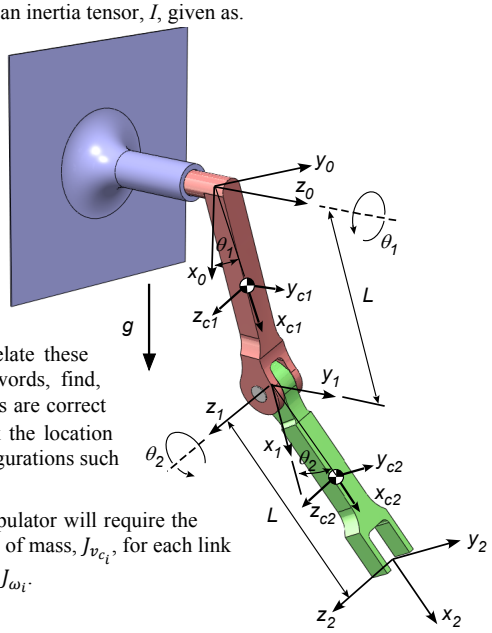
2.3 HW 3

2.3.1 Problem 1

Problem 1 [20 points]

Following the steps outlined below, derive the equations of motion of the two degree-of-freedom manipulator depicted. The links have mass, m, and an inertia tensor, I, given as.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}I_a & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_a \end{bmatrix}$$



The location of the center of mass is located in the middle of each link.

Note: you are strongly advised to use the Matlab symbolic toolbox (or equivalent) for your algebraic and differential operations.

- On each link we have attached frames at the joints and at the center of mass of each link. Calculate the homogeneous transforms that relate these frames to the inertial frame $\{0\}$. In other words, find, T_1^0, T_2^0, $T_{C_1}^0$, and $T_{C_2}^0$. Verify that your matrices are correct before proceeding to the next step (e.g. check the location and orientation of your frames for simple configurations such as $\theta_1 = \theta_2 = 90°$)
- The evaluation of the mass matrix of the manipulator will require the computation of the linear Jacobian of the center of mass, $J_{v_{c_i}}$, for each link and the angular velocity Jacobian for each link, J_{ω_i}.
  - Find $J_{v_{c_1}}$ and $J_{v_{c_2}}$.
  - Find J_{ω_1} and J_{ω_2}.

  Verify that your matrices are correct before proceeding to the next step.
- Evaluate the mass matrix, $D(q)$, of the manipulator in terms of mass and geometric properties and its configuration.
- Evaluate the centrifugal and Coriolis inertial terms of the manipulator in terms of mass and geometric properties and its configuration.
  - Find $B(q)[\dot{q}\dot{q}]$
  - Find $C(q)[\dot{q}^2]$
- Evaluate the gravity vector, $G(q)$. In frame $\{0\}$ the gravity vector is given as: $\mathbf{g} = g[1 \quad 0 \quad 0]^T$.
- Form the complete equations of motion in the form

$$D(q)\ddot{q} + B(q)[\dot{q}\dot{q}] + C(q)[\dot{q}^2] + G(q) = \tau$$

Figure 2.16: Problem 1 description

Part 1

The homogeneous transformation matrices T_i^{i-1} are found by direct inspection.

$$T_1^0 = \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 & L\cos\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & L\sin\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2^1 = \begin{pmatrix} [1.3]\cos\theta_2 & -\sin\theta_2 & 0 & L\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & L\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hence

$$\begin{aligned} T_2^0 &= T_1^0 T_2^1 \\ &= \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 & L\cos\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & L\sin\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} [1.3]\cos\theta_2 & -\sin\theta_2 & 0 & L\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & L\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 & L\cos\theta_1 + L\cos\theta_2\cos\theta_1 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 & L\sin\theta_1 + L\cos\theta_2\sin\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & L\sin\theta_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

$T_{c_i}^{i-1}$ are now found. These are the same as T_i^{i-1} but with the last column adjusted for the center of mass position which is given as being in the middle of the link, therefore

$$T_{c_1}^0 = \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 & \frac{L}{2}\cos\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & \frac{L}{2}\sin\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{c_2}^1 = \begin{pmatrix} [1.3]\cos\theta_2 & -\sin\theta_2 & 0 & \frac{L}{2}\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & \frac{L}{2}\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hence

$$
\begin{aligned}
T_{c_2}^0 &= T_1^0 T_{c_2}^1 \\
&= \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 & L\cos\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & L\sin\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} [1.3]\cos\theta_2 & -\sin\theta_2 & 0 & \frac{L}{2}\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & \frac{L}{2}\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 & L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 & L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & \frac{1}{2}L\sin\theta_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
$$

VERIFICATION:

The end effector position (origin of frame (2)), relative to the inertial frame is given by

$$
\begin{aligned}
r_2^0 &= T_2^0 r_2^2 \\
&= \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 & L\cos\theta_1 + L\cos\theta_2\cos\theta_1 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 & L\sin\theta_1 + L\cos\theta_2\sin\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & L\sin\theta_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} [1.3]0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} [1.3]L\cos\theta_1 + L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + L\cos\theta_2\sin\theta_1 \\ L\sin\theta_2 \\ 1 \end{pmatrix} \qquad (1)
\end{aligned}
$$

When $\theta_1 = \theta_2 = 0$ the above becomes

$$
r_2^0 = \begin{pmatrix} [1.3]2L \\ 0 \\ 0 \\ 1 \end{pmatrix}
$$

Which is what expected. Now it is verified for $\theta_1 = 90^0, \theta_2 = 0$, Substituting these angle values in (1) gives

$$
\begin{pmatrix} [1.3]0 \\ 2L \\ 0 \\ 1 \end{pmatrix}
$$

as expected. At $\theta_1 = 0, \theta_2 = 90^0$ and substituting these angle values in (1) gives

$$\begin{pmatrix} [1.3]L \\ 0 \\ L \\ 1 \end{pmatrix}$$

as expected. Finally, at $\theta_1 = \theta_2 = 90^0$ and substituting these angle values in (1) gives

$$\begin{pmatrix} [1.3]0 \\ L \\ L \\ 1 \end{pmatrix}$$

as expected. The result of part (1) have been verified.

Part 2

To determine the linear Jacobian for the center of the mass, the following quantities needs to be determined

$$z_0^0, z_1^0, o_0^0, o_1^0, o_{c_1}^0, o_{c_2}^0$$

These are obtained from the result of part (1)

$$z_0^0 = \begin{pmatrix} [1.3]0 \\ 0 \\ 1 \end{pmatrix}$$

$$z_1^0 = \begin{pmatrix} [1.3]\sin\theta_1 \\ -\cos\theta_1 \\ 0 \end{pmatrix}$$

$$o_0^0 = \begin{pmatrix} [1.3]0 \\ 0 \\ 0 \end{pmatrix}$$

$$o_1^0 = \begin{pmatrix} [1.3]L\cos\theta_1 \\ L\sin\theta_1 \\ 0 \end{pmatrix}$$

$$o_{c_1}^0 = \begin{pmatrix} [1.3]\frac{L}{2}\cos\theta_1 \\ \frac{L}{2}\sin\theta_1 \\ 0 \end{pmatrix}$$

$$o_{c_2}^0 = \begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \frac{1}{2}L\sin\theta_2 \end{pmatrix}$$

Therefore

$$J_{v_{c_1}} = \begin{pmatrix} [1.3]z_0^0 \times \left(o_{c_1}^0 - o_0^0\right) & \mathbf{0} \end{pmatrix}$$
$$= \begin{pmatrix} [1.3] \begin{pmatrix} [1.3]0 \\ 0 \\ 1 \end{pmatrix} \times \left(\begin{pmatrix} [1.3]\frac{L}{2}\cos\theta_1 \\ \frac{L}{2}\sin\theta_1 \\ 0 \end{pmatrix} - \begin{pmatrix} [1.3]0 \\ 0 \\ 0 \end{pmatrix} \right) & \mathbf{0} \end{pmatrix}$$
$$= \boxed{\begin{pmatrix} [1.3] - \frac{1}{2}L\sin\theta_1 & 0 \\ \frac{1}{2}L\cos\theta_1 & 0 \\ 0 & 0 \end{pmatrix}}$$

$$J_{v_{c_2}} = \begin{pmatrix} [1.3]z_0^0 \times o_{c_2}^0 & z_1^0 \times \left(o_{c_2}^0 - o_1^0\right) \end{pmatrix}$$
$$= \begin{pmatrix} [1.3] \begin{pmatrix} [1.3]0 \\ 0 \\ 1 \end{pmatrix} \times \begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \frac{1}{2}L\sin\theta_2 \end{pmatrix} & \begin{pmatrix} [1.3]\sin\theta_1 \\ -\cos\theta_1 \\ 0 \end{pmatrix} \times \left(\begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\mathrm{co} \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta \\ \frac{1}{2}L\sin\theta_2 \end{pmatrix} \right. \end{pmatrix}$$
$$= \boxed{\begin{pmatrix} [1.3] - L\sin\theta_1 - \frac{1}{2}L\cos\theta_2\sin\theta_1 & -\frac{1}{2}L\cos\theta_1\sin\theta_2 \\ L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 & -\frac{1}{2}L\sin\theta_1\sin\theta_2 \\ 0 & L\cos\theta_2 \end{pmatrix}}$$

Now the angular velocity Jacobians are found

$$J_{\omega_1} = \begin{pmatrix} [1.3]\bar{\varepsilon}_1 z_0^0 & 0 \end{pmatrix} = \boxed{\begin{pmatrix} [1.3]0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}}$$

$$J_{\omega_2} = \begin{pmatrix} [1.3]\bar{\varepsilon}_1 z_0^0 & \bar{\varepsilon}_2 z_1^0 \end{pmatrix} = \boxed{\begin{pmatrix} [1.3]0 & \sin\theta_1 \\ 0 & -\cos\theta_1 \\ 1 & 0 \end{pmatrix}}$$

VERIFICATION:

To verify the linear velocity Jacobians the following relations are used

$$J_{v_{c_1}} = \begin{pmatrix} [1.3]\frac{\partial r_{c_1}}{\partial\theta_1} & \frac{\partial r_{c_1}}{\partial\theta_2} \end{pmatrix} \tag{2}$$

$$J_{v_{c_2}} = \begin{pmatrix} [1.3]\frac{\partial r_{c_2}}{\partial\theta_1} & \frac{\partial r_{c_2}}{\partial\theta_2} \end{pmatrix} \tag{3}$$

Result obtained from the above is compared to part (2) result. In the above, r_{c_i} is the position of the center of mass of link i measured from the origin of the inertial frame. Therefore[5]

$$\begin{aligned} r^0_{c_1} &= T^0_1 r^1_{c_1} \\ &= \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 & L\cos\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & L\sin\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} [1.3] - \frac{L}{2} \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} [1.3]\frac{1}{2}L\cos\theta_1 \\ \frac{1}{2}L\sin\theta_1 \\ 0 \\ 1 \end{pmatrix} \end{aligned}$$

Therefore

$$r^0_{c_1} = \begin{pmatrix} [1.3]\frac{1}{2}L\cos\theta_1 \\ \frac{1}{2}L\sin\theta_1 \\ 0 \end{pmatrix}$$

Using the above in (2) gives

$$\begin{aligned} J_{v_{c_1}} &= \begin{pmatrix} [1.3]\frac{\partial}{\partial\theta_1}\begin{pmatrix} [1.3]\frac{1}{2}L\cos\theta_1 \\ \frac{1}{2}L\sin\theta_1 \\ 0 \end{pmatrix} & \frac{\partial}{\partial\theta_2}\begin{pmatrix} [1.3]\frac{1}{2}L\cos\theta_1 \\ \frac{1}{2}L\sin\theta_1 \\ 0 \end{pmatrix} \end{pmatrix} \\ &= \boxed{\begin{pmatrix} [1.3] - \frac{1}{2}L\sin\theta_1 & 0 \\ \frac{1}{2}L\cos\theta_1 & 0 \\ 0 & 0 \end{pmatrix}} \end{aligned}$$

The above is the same result obtained in part (2). Similarly $J_{v_{c_2}}$ is verified

$$\begin{aligned} r^0_{c_2} &= T^0_2 r^2_{c_2} \\ &= \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 & L\cos\theta_1 + L\cos\theta_2\cos\theta_1 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 & L\sin\theta_1 + L\cos\theta_2\sin\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & L\sin\theta_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} [1.3] - \frac{L}{2} \\ 0 \\ 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \frac{1}{2}L\sin\theta_2 \\ 1 \end{pmatrix} \end{aligned}$$

[5] we can also use $r^0_{c_1} = T^0_{c_1} r^{c_1}_{c_1}$ where in this case $r^{c_1}_{c_1} = [0,0,0,1]^T$

Hence

$$r_{c_2}^0 = \begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \frac{1}{2}L\sin\theta_2 \end{pmatrix}$$

Using the above in (3) gives

$$J_{v_{c_2}} = \begin{pmatrix} [1.3]\frac{\partial}{\partial\theta_1}\begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \frac{1}{2}L\sin\theta_2 \end{pmatrix} & \frac{\partial}{\partial\theta_2}\begin{pmatrix} [1.3]L\cos\theta_1 + \frac{1}{2}L\cos\theta_1\cos\theta_2 \\ L\sin\theta_1 + \frac{1}{2}L\cos\theta_2\sin\theta_1 \\ \frac{1}{2}L\sin\theta_2 \end{pmatrix} \end{pmatrix}$$

$$= \boxed{\begin{pmatrix} [1.3] - L\sin\theta_1 - \frac{1}{2}L\sin\theta_1\cos\theta_2 & -\frac{1}{2}L\cos\theta_1\sin\theta_2 \\ L\cos\theta_1 + \frac{1}{2}L\cos\theta_2\cos\theta_1 & -\frac{1}{2}L\sin\theta_2\sin\theta_1 \\ 0 & \frac{1}{2}L\cos\theta_2 \end{pmatrix}}$$

The above is the same as found in part(2).

2.3.2 Part 3

The mass matrix $D(q)$ is evaluated. By definition

$$D(q) = \sum_{i=1}^{2} m_i J_{v_{c_i}}^T J_{v_{c_i}} + J_{\omega_i}^T R_i^0 I_{c_i}^i \left(R_i^0\right)^T J_{\omega_i} \tag{4}$$

The following is found in part (1)

$$R_1^0 = \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$R_2^0 = \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 \end{pmatrix}$$

Therefore (4) becomes

$$
\begin{aligned}
D_1(q) = m_1 & \begin{pmatrix} [1.3] - \frac{1}{2}L\sin\theta_1 & 0 \\ \frac{1}{2}L\cos\theta_1 & 0 \\ 0 & 0 \end{pmatrix}^T \begin{pmatrix} [1.3] - \frac{1}{2}L\sin\theta_1 & 0 \\ \frac{1}{2}L\cos\theta_1 & 0 \\ 0 & 0 \end{pmatrix} \\
& + \begin{pmatrix} [1.3]0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}^T \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} [1.3]\frac{1}{2}I_a & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_a \end{pmatrix} \\
& \begin{pmatrix} [1.3]\cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{pmatrix}^T \begin{pmatrix} [1.3]0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \\
& = m_1 \begin{pmatrix} [1.3]\frac{1}{4}L^2 + I_a & 0 \\ 0 & 0 \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
D_2(q) = m_2 & \begin{pmatrix} [1.3] - L\sin\theta_1 - \frac{1}{2}L\sin\theta_1\cos\theta_2 & -\frac{1}{2}L\cos\theta_1\sin\theta_2 \\ L\cos\theta_1 + \frac{1}{2}L\cos\theta_2\cos\theta_1 & -\frac{1}{2}L\sin\theta_2\sin\theta_1 \\ 0 & \frac{1}{2}L\cos\theta_2 \end{pmatrix}^T \begin{pmatrix} [1.3] - L\sin\theta_1 - \frac{1}{2}L\sin\theta_1\cos\theta_2 & - \\ L\cos\theta_1 + \frac{1}{2}L\cos\theta_2\cos\theta_1 & - \\ 0 & \end{pmatrix} \\
& + \begin{pmatrix} [1.3]0 & \sin\theta_1 \\ 0 & -\cos\theta_1 \\ 1 & 0 \end{pmatrix}^T \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 \end{pmatrix} \begin{pmatrix} [1.3]\frac{1}{2}I_a & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_a \end{pmatrix} \\
& \begin{pmatrix} [1.3]\cos\theta_1\cos\theta_2 & -\cos\theta_1\sin\theta_2 & \sin\theta_1 \\ \cos\theta_2\sin\theta_1 & -\sin\theta_1\sin\theta_2 & -\cos\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 \end{pmatrix}^T \begin{pmatrix} [1.3]0 & \sin\theta_1 \\ 0 & -\cos\theta_1 \\ 1 & 0 \end{pmatrix} \\
& = m_2 \begin{pmatrix} [1.3]\frac{1}{8}\left(6I_a + 9L^2 + 8L^2\cos\theta_2 + (2I_a + L^2)\cos(2\theta_2)\right) & 0 \\ 0 & I_a + \frac{L^2}{4} \end{pmatrix}
\end{aligned}
$$

Hence the $D(q)$ becomes

$$
\begin{aligned}
D(q) &= \begin{pmatrix} [1.3]m_1\left(\frac{1}{4}L^2 + I_a\right) & 0 \\ 0 & 0 \end{pmatrix} + m_2 \begin{pmatrix} [1.3]\frac{1}{8}\left(6I_a + 9L^2 + 8L^2\cos\theta_2 + (2I_a + L^2)\cos(2\theta_2)\right) & 0 \\ 0 & I_a + \frac{L^2}{4} \end{pmatrix} \\
&= \boxed{\begin{pmatrix} [1.3]m_1\left(I_a + \frac{1}{4}L^2\right) + \frac{1}{8}m_2\left(6I_a + 9L^2 + 8L^2\cos\theta_2 + (2I_a + L^2)\cos(2\theta_2)\right) & 0 \\ 0 & m_2\left(I_a + \frac{L^2}{4}\right) \end{pmatrix}}
\end{aligned}
$$

Part 4

The Coriolis term $B(q)\left[\dot{q}\dot{q}\right]$ is now evaluated

$$B\left(q\right)\left[\dot{q}\dot{q}\right]=\begin{bmatrix}2b_{1,12}\\2b_{2,12}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\dot{\theta}_2\end{bmatrix} \tag{1}$$

Where $b_{i,jk}$ is the Christoffel symbol of first kind defined as

$$b_{i,jk}=\frac{1}{2}\left(d_{ijk}+d_{ikj}-d_{jki}\right)$$

Where $d_{ijk}=\frac{\partial d_{ij}}{\partial q_k}$. Using these in (1) gives

$$\begin{aligned}B\left(q\right)\left[\dot{q}\dot{q}\right]&=\begin{bmatrix}d_{112}+d_{121}-d_{121}\\d_{212}+d_{221}-d_{122}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\dot{\theta}_2\end{bmatrix}\\&=\begin{bmatrix}d_{112}\\d_{212}+d_{221}-d_{122}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\dot{\theta}_2\end{bmatrix}\\&=\begin{bmatrix}\frac{\partial d_{11}}{\partial q_2}\\\frac{\partial d_{21}}{\partial q_2}+\frac{\partial d_{22}}{\partial q_1}-\frac{\partial d_{12}}{\partial q_2}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\dot{\theta}_2\end{bmatrix}\end{aligned}$$

The mass matrix was found in part (3) as

$$D\left(q\right)=\begin{bmatrix}m_1\left(I_a+\frac{1}{4}L^2\right)+\frac{1}{8}m_2\left(6I_a+9L^2+8L^2\cos\theta_2+\left(2I_a+L^2\right)\cos\left(2\theta_2\right)\right) & 0\\0 & m_2\left(I_a+\frac{L^2}{4}\right)\end{bmatrix}$$

Hence

$$\begin{aligned}\frac{\partial d_{11}}{\partial q_2}&=\frac{\partial d_{11}}{\partial\theta_2}\\&=\frac{\partial}{\partial\theta_2}\left[m_1\left(I_a+\frac{1}{4}L^2\right)+\frac{1}{8}m_2\left(6I_a+9L^2+8L^2\cos\theta_2+\left(2I_a+L^2\right)\cos\left(2\theta_2\right)\right)\right]\\&=\frac{1}{8}m_2\left(-8L^2\sin\theta_2-2\left(2I_a+L^2\right)\sin\left(2\theta_2\right)\right)\\&=-m_2L^2\sin\theta_2-\frac{1}{4}m_2\left(2I_a+L^2\right)\sin\left(2\theta_2\right)\end{aligned}$$

And

$$\begin{aligned}\frac{\partial d_{21}}{\partial q_2}&=\frac{\partial d_{21}}{\partial\theta_2}\\&=0\end{aligned}$$

And

$$\begin{aligned}\frac{\partial d_{22}}{\partial q_1}&=\frac{\partial d_{22}}{\partial\theta_1}\\&=0\end{aligned}$$

And

$$\begin{aligned}\frac{\partial d_{12}}{\partial q_2} &= \frac{\partial d_{12}}{\partial \theta_2} \\ &= 0\end{aligned}$$

Hence

$$\begin{aligned}B\left(q\right)\left[\dot{q}\dot{q}\right] &= \begin{bmatrix}-m_2 L^2 \sin\theta_2 - \frac{1}{4} m_2 \left(2I_a + L^2\right) \sin\left(2\theta_2\right) \\ 0\end{bmatrix} \begin{bmatrix}\dot{\theta}_1 \dot{\theta}_2\end{bmatrix} \\ &= \boxed{\begin{bmatrix}-m_2 \dot{\theta}_1 \dot{\theta}_2 L^2 \sin\theta_2 - \frac{1}{4} \dot{\theta}_1 \dot{\theta}_2 m_2 \left(2I_a + L^2\right) \sin\left(2\theta_2\right) \\ 0\end{bmatrix}}\end{aligned}$$

The centrifugal term is now evaluated

$$C\left(q\right)\begin{bmatrix}\dot{q}^2\end{bmatrix} = \begin{bmatrix}b_{1,11} & b_{1,22} \\ b_{2,11} & b_{2,22}\end{bmatrix}\begin{bmatrix}\dot{\theta}_1^2 \\ \dot{\theta}_2^2\end{bmatrix}$$

Where

$$\begin{aligned}b_{1,11} &= \frac{1}{2}\left(d_{111} + d_{111} - d_{111}\right) \\ &= \frac{1}{2} d_{111} \\ &= \frac{1}{2}\frac{\partial d_{11}}{\partial q_1} \\ &= 0\end{aligned}$$

And

$$\begin{aligned}b_{1,22} &= \frac{1}{2}\left(d_{122} + d_{122} - d_{221}\right) \\ &= d_{122} - \frac{1}{2} d_{221} \\ &= \frac{\partial d_{12}}{\partial \theta_2} - \frac{1}{2}\frac{\partial d_{22}}{\partial \theta_1} \\ &= 0\end{aligned}$$

And

$$\begin{aligned}b_{2,11} &= \frac{1}{2}\left(d_{211} + d_{211} - d_{112}\right) \\ &= d_{211} - \frac{1}{2} d_{112} \\ &= \frac{\partial d_{21}}{\partial \theta_1} - \frac{1}{2}\frac{\partial d_{11}}{\partial \theta_2} \\ &= 0 - \frac{1}{2}\left[-m_2 L^2 \sin\theta_2 - \frac{1}{4} m_2 \left(2I_a + L^2\right) \sin\left(2\theta_2\right)\right] \\ &= \frac{1}{2} m_2 L^2 \sin\theta_2 + \frac{1}{8} m_2 \left(2I_a + L^2\right) \sin\left(2\theta_2\right)\end{aligned}$$

And

$$\begin{aligned} b_{2,22} &= \frac{1}{2}\left(d_{222}+d_{222}-d_{222}\right) \\ &= \frac{1}{2}d_{222} \\ &= \frac{1}{2}\frac{\partial d_{21}}{\partial \theta_2} \\ &= 0 \end{aligned}$$

Therefore

$$\begin{aligned} C\left(q\right)\left[\dot{q}^2\right] &= \begin{bmatrix} 0 & 0 \\ \frac{1}{2}m_2L^2\sin\theta_2+\frac{1}{8}m_2\left(2I_a+L^2\right)\sin\left(2\theta_2\right) & 0 \end{bmatrix}\begin{bmatrix}\dot{\theta}_1^2 \\ \dot{\theta}_2^2\end{bmatrix} \\ &= \boxed{\begin{bmatrix} 0 \\ \frac{1}{2}m_2\dot{\theta}_1^2L^2\sin\theta_2+\frac{1}{8}m_2\dot{\theta}_1^2\left(2I_a+L^2\right)\sin\left(2\theta_2\right)\end{bmatrix}} \end{aligned}$$

Part (5)

The gravity vector $G(q)$ is now evaluated

$$G\left(q\right) = -\left[J_{v_{c_1}}^T m_1\mathbf{g}+J_{v_{c_2}}^T m_2\mathbf{g}\right] \tag{1}$$

From part(2) we found

$$J_{v_{c_1}} = \begin{bmatrix} -\frac{1}{2}L\sin\theta_1 & 0 \\ \frac{1}{2}L\cos\theta_1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$J_{v_{c_2}} = \begin{bmatrix} -L\sin\theta_1-\frac{1}{2}L\sin\theta_1\cos\theta_2 & -\frac{1}{2}L\cos\theta_1\sin\theta_2 \\ L\cos\theta_1+\frac{1}{2}L\cos\theta_2\cos\theta_1 & -\frac{1}{2}L\sin\theta_2\sin\theta_1 \\ 0 & \frac{1}{2}L\cos\theta_2 \end{bmatrix}$$

Hence (1) becomes

$$\begin{aligned} G\left(q\right) &= -m_1g\begin{bmatrix} -\frac{1}{2}L\sin\theta_1 & 0 \\ \frac{1}{2}L\cos\theta_1 & 0 \\ 0 & 0 \end{bmatrix}^T\begin{bmatrix}1\\0\\0\end{bmatrix} - m_2g\begin{bmatrix} -L\sin\theta_1-\frac{1}{2}L\sin\theta_1\cos\theta_2 & -\frac{1}{2}L\cos\theta_1\sin\theta_2 \\ L\cos\theta_1+\frac{1}{2}L\cos\theta_2\cos\theta_1 & -\frac{1}{2}L\sin\theta_2\sin\theta_1 \\ 0 & \frac{1}{2}L\cos\theta_2 \end{bmatrix}^T\begin{bmatrix}1\\0\\0\end{bmatrix} \\ &= \begin{bmatrix}\frac{1}{2}gm_1L\sin\theta_1 \\ 0\end{bmatrix} - \begin{bmatrix} -Lgm_2\sin\theta_1-\frac{1}{2}Lgm_2\cos\theta_2\sin\theta_1 \\ -\frac{1}{2}gm_2L\cos\theta_1\sin\theta_2\end{bmatrix} \\ &= \boxed{\begin{bmatrix}\frac{1}{2}Lgm_1\sin\theta_1+Lgm_2\sin\theta_1+\frac{1}{2}Lgm_2\cos\theta_2\sin\theta_1 \\ \frac{1}{2}Lgm_2\cos\theta_1\sin\theta_2\end{bmatrix}} \end{aligned}$$

Part (6)

The equation of motion is now found using above results. Using the notation [ ] for a matrix and { } for a vector, it is written as

$$\overbrace{[D]\{\ddot{q}\}}^{\text{mass matrix}} + \overbrace{[B]\{\dot{q}\dot{q}\}}^{\text{Coriolis}} + \overbrace{[C]\left\{\dot{q}^2\right\}}^{\text{centrifugal}} + \overbrace{[G]}^{\text{gravity}} = \overbrace{\{\tau\}}^{\text{torques}}$$

Since there is no applied external torques or forces, the right hand side is zero. The equation of motion becomes

$$\begin{bmatrix} m_1\left(I_a+\frac{1}{4}L^2\right)+\frac{1}{8}m_2\left(6I_a+9L^2+8L^2\cos\theta_2+(2I_a+L^2)\cos(2\theta_2)\right) & 0 \\ 0 & m_2\left(I_a+\frac{L^2}{4}\right) \end{bmatrix}\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}$$
$$+\begin{bmatrix} -m_2L^2\sin\theta_2-\frac{1}{4}m_2\left(2I_a+L^2\right)\sin(2\theta_2) \\ 0 \end{bmatrix}\begin{bmatrix}\dot{\theta}_1\dot{\theta}_2\end{bmatrix}+\begin{bmatrix} 0 & 0 \\ \frac{1}{2}m_2L^2\sin\theta_2+\frac{1}{8}m_2\left(2I_a+L^2\right)\sin(2\theta_2) & 0 \end{bmatrix}\begin{bmatrix}\dot{\theta}_1^2\\ \dot{\theta}_2^2\end{bmatrix}$$
$$+\begin{bmatrix} \frac{1}{2}Lgm_1\sin\theta_1+Lgm_2\sin\theta_1+\frac{1}{2}Lgm_2\cos\theta_2\sin\theta_1 \\ \frac{1}{2}Lgm_2\cos\theta_1\sin\theta_2 \end{bmatrix}=\begin{bmatrix}\tau_1\\ \tau_1\end{bmatrix}$$

Looking at each equation of motion on its own, for θ_1 the equation of motion is

$$\ddot{\theta}_1\left[m_1\left(I_a+\frac{1}{4}L^2\right)+\frac{1}{8}m_2\left(6I_a+(\cos(2\theta_2))\left(L^2+2I_a\right)+8L^2\cos\theta_2+9L^2\right)\right]$$
$$-\left(m_2L^2\sin\theta_2+\frac{1}{4}m_2\left(2I_a+L^2\right)\sin(2\theta_2)\right)\dot{\theta}_1\dot{\theta}_2+\frac{1}{2}Lgm_1\sin\theta_1+Lgm_2\sin\theta_1+\frac{1}{2}Lgm_2\cos\theta_2\sin\theta_1=0$$

And the equation of motion for θ_2 is

$$\ddot{\theta}_2m_2\left(\frac{1}{4}L^2+I_a\right)+\left(\frac{1}{2}m_2L^2\sin\theta_2+\frac{1}{8}m_2\left(2I_a+L^2\right)\sin(2\theta_2)\right)\dot{\theta}_1^2+\frac{1}{2}Lgm_2\cos\theta_1\sin\theta_2=0$$

These are coupled differential equations since θ_2,θ_1 appears in both equations. They are also nonlinear due to the $\dot{\theta}_2^2,\dot{\theta}_1^2$ terms.

2.3.3 Problem 2

Problem 2. [20 points]

Using your results from the previous question, write a Matlab numerical simulation of the manipulator described in Question 2. Implement a 4th order Runge-Kutta numerical integration of your equations of motion. The model and simulation parameters are given below.

Geometric parameters:	L = 2 meters
Mass properties:	m = 10 kg
	I_a = 5 kg/m^2
	g = 9.81 m/s^2
Simulation parameters:	Δt = 0.01 seconds (integration time step)
	T_{final} = 20 seconds (simulated time duration)

Note, to help you develop your code, some example code has been posted to the course Learn@UW page. Please use these scripts as a starting point. I would advise you to review all of the posted examples before you start working on your code.

- Simulate (*and animate*) the release of the manipulator from a vertical position (i.e. $q_1 = 180°, q_2 = 0$) with small initial joint velocity ($\dot{q}_1 = \dot{q}_2 = 0.1\ r/s$). You may want to add a small amount of damping to the joint torque vector such that the system starts to settle after a few major oscillations (suggested c_i = 1 N-m/r/s where $\tau_i = -c_i \dot{q}_i$).
- To check that you simulation (and equations of motion) are correct, calculate and plot the system energy for the case where the damping coefficients are set equal to zero. Specifically, please plot the total system kinetic energy, T, the total system potential energy, V, and the total system energy ($T + V$) as a function of time. If your simulation and underlying equations of motion are correct, the total energy should stay constant.

Figure 2.17: Problem 2 description

2.3.4 Part 1

The equations of motions for the 2 link serial manipulator were simulated by modifying the learn UW Matlab code using results of problem 1.

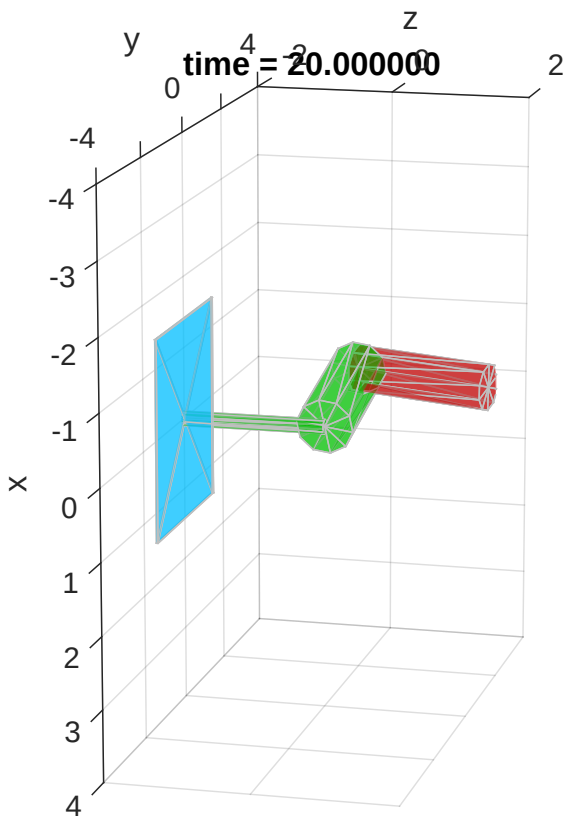


Figure 2.18: screen shot of simulation

A small amount of damping $c = 1$ N-m/r/s was added to both joints in the file `zDot2dof.m` and the simulation was run for 20 seconds.

When adding damping, the equation of motion becomes

$$\overbrace{[D]\{\ddot{q}\}}^{\text{mass matrix}} + \overbrace{c\{\dot{q}\}}^{\text{damping}} + \overbrace{[B]\{\dot{q}\dot{q}\}}^{\text{Coriolis}} + \overbrace{[C]\left\{\dot{q}^2\right\}}^{\text{centrifugal}} + \overbrace{[G]}^{\text{gravity}} = \overbrace{\{\tau\}}^{\text{torques}}$$

Where c above is the damping constant used.

The diagram below shows that total energy decreased as would be expected.

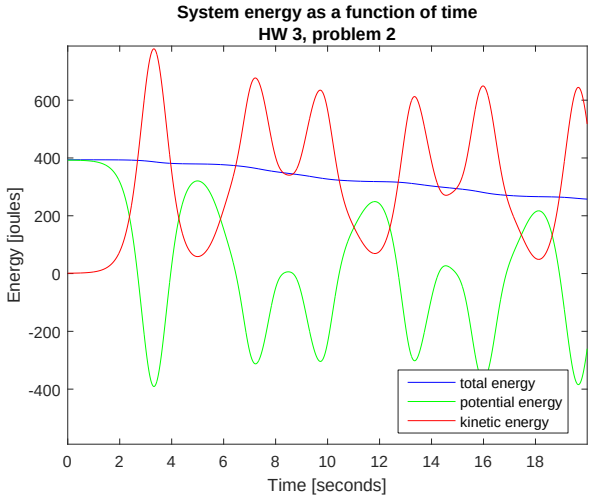


Figure 2.19: Damping $c = 1$ N-m/r/s for 20 seconds

The damping c was now increased to 5 N-m/r/s and the simulation time increased to 50 seconds. Now the total energy decreased to almost zero by the end of the simulation and the robot arm came close to being at rest. This was done as an additional verification yo verify the equations of motion.

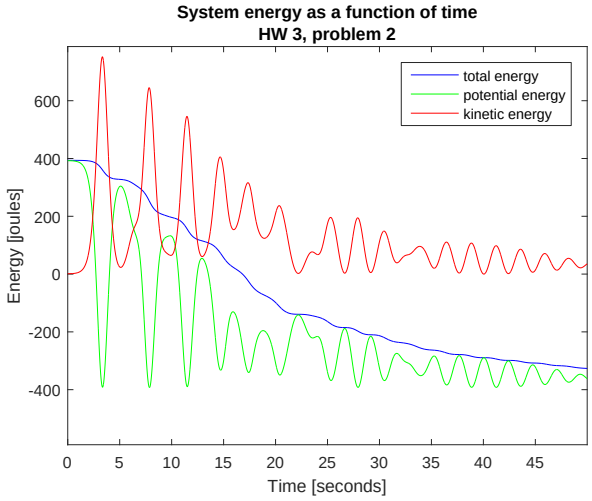


Figure 2.20: Damping $c = 5$ N-m/r/s for 50 seconds

Part 2

The damping constant was removed and the simulation run again for 20 seconds. The total energy remained constant as would be expected

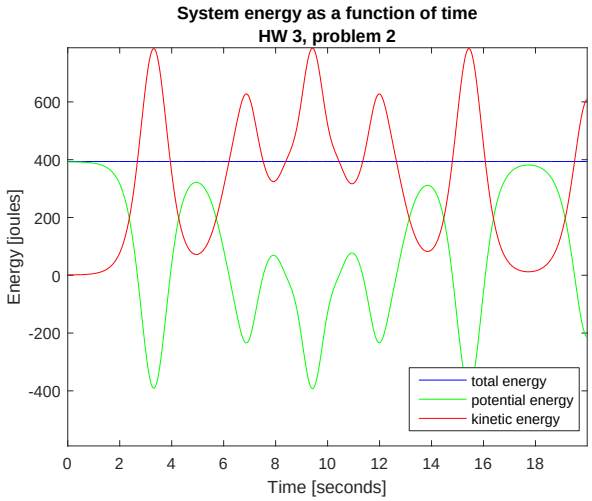


Figure 2.21: Damping $c = 0$ for 20 seconds

Animation

The following is a movie of the first 10 seconds of the simulation for the zero damping case.

source code

The following are the Matlab source code listings of all the files used used for the implementation of this problem.

```
%nma_HW3_problem.m
%This is modified version of the UW learn script used to solve
%probvlem 2, HW 3. ME 739, Univ. Wisconsin Madison
%Nasser M. Abbasi
%--------------------------------------------------------------------------
%  NUMERICAL INTEGRATION OF DYNAMIC EQUATIONS
%--------------------------------------------------------------------------
clear all; close all; clc;

DO_MOVIE = false; %make true to generate movie frames

% model parameters
modelParameters.g  = 9.81; % gravitational constant [m/s^2]
modelParameters.m  = 10;   % link mass [kg]
modelParameters.L  = 2;    % link length [m]
modelParameters.Ia = 5;   % inertia (_|_ to  link's CL) [kg/m^2]

% initialize integration variables
dT      = .01;                %integration step size
tend    = 20;                 %simulation run time
numPts  = floor(tend/dT);
q       = zeros(2,numPts);    %pre-allocate array
dq      = zeros(2,numPts);
t       = zeros(1,numPts);
q(:,1)  = [pi; 0];            %initial position
qd(:,1) = [0.1; 0.1];         %initial velocity
z       = [q(:,1); qd(:,1)];  %initialize the state variables

% integrate equations of motion
for i = 1:numPts-1
    % Runge-Kutta 4th order
    k1 = zDot2dof(z,modelParameters);
    k2 = zDot2dof(z + 0.5*k1*dT,modelParameters);
    k3 = zDot2dof(z + 0.5*k2*dT,modelParameters);
    k4 = zDot2dof(z + k3*dT,modelParameters);
    z  = z + (1/6)*(k1 + 2*k2 + 2*k3 + k4)*dT;

    % store joint position and velocity for post processing
    q(:,i+1)  = z(1:2);
```

```
    qd(:,i+1) = z(3:4);
    t(1,i+1)  = t(1,i) + dT;
end

%-------------------------------------------------------------------------
%  RENDERING INITIALIZATION
%-------------------------------------------------------------------------

%----set rendering window view parameters
L           = modelParameters.L;
L1          = L;
L2          = L;
f_handle    = 1;
axis_limits = L*[-2 2 -2 2 -1 1];
render_view = [-.4 -.8 .5]; view_up = [-1 0 0];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective % turns on perspective

%----initialize rendering

% link 00 rendering initialization, this is the base on the wall
% fixed, used for illustration
r00 = L; sides00 = 4; axis00 = 3; norm_L00 = 1.0;
linkColor1 = [.6 0.75 0]; plotFrame00 = 0;
d00 =  CreateLinkRendering(0.01*L,r00,sides00,axis00,norm_L00,...
    linkColor1,plotFrame00,f_handle);

% link 0 this is the rod holding the arm, fixed does not move
r0 = L/10; sides0 = 10; axis0 = 3; norm_L = 1.0;
linkColor0 = [0.75 1 1]; plotFrame0 = 0;
d0 =  CreateLinkRendering(L,r0,sides0,axis0,norm_L,linkColor0,...
    plotFrame0,f_handle);

% link 1 rendering initialization
r1 = L1/5; sides1 = 10; axis1 = 1; norm_L1 = 1.0;
linkColor1 = [0 0.75 0]; plotFrame1 = 0;
d1 =  CreateLinkRendering(L1,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);

% link 2 rendering initialization
r2 = L2/6; sides2 = 10; axis2 = 1; norm_L2 = 1.0;
linkColor2 = [0.75 0 0]; plotFrame2 = 0;
d2 =  CreateLinkRendering(L2,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

%-------------------------------------------------------------------
%  DISPLAY INTERATION RESULTS
```

```
%-----------------------------------------------------------------
%since these links are fixed, they are set outside the loop
T00  = [1  0   0  0
    0  1   0   0
    0  0   1   -L
    0  0   0  1];

T0  = [1  0   0  0
    0  1   0  0
    0  0   1  0
    0  0   0  1];
UpdateLink(d00,T00);
UpdateLink(d0,T0);
k = 0;

for i = 1:numPts
    % Update frame {1}
    c = cos(q(1,i)); s = sin(q(1,i)); L = L1;
    T10 = [c  0   s  L*c
        s  0  -c  L*s
        0  1   0  0
        0  0   0  1];

    % Update frame {2}
    c = cos(q(2,i)); s = sin(q(2,i)); L = L2;
    T21 = [c  -s  0  L*c
        s   c  0  L*s
        0   0  1   0
        0   0  0   1];
    T20 = T10*T21;

    UpdateLink(d1,T10);
    UpdateLink(d2,T20);

    title(sprintf('time = %f',i*dT));
    if i == 1;  %pause at start of simulation rendering
        pause;
    end

    if DO_MOVIE
       k = k+1;
       I = getframe(gcf);
       imwrite(I.cdata, sprintf('frame%d.png',k));
    end

    pause(dT);
end
```

```
%--------------------------------------------------------------------
%  ENERGY BALANCE - TO CHECK SIMULATION RESULTS
%--------------------------------------------------------------------

V = zeros(numPts,1);
T = V;
E = V;

for i = 1:numPts
    % kinetic energy
    Q    = q(:,i);
    Qd   = qd(:,i);
    D    = Dmatrix2dof(Q,modelParameters);
    T(i) = (1/2)*Qd'*D*Qd;

    % potential energy
    %   evaluate position of links center of mass
    %   Link 1:
    c    = cos(q(1,i));
    s    = sin(q(1,i)); L = L1;
    T10  = [c  0   s   L*c
            s  0  -c  L*s
            0  1   0  0
            0  0   0  1];
    L    = L1/2;
    T10c = [c  0   s   L*c
            s  0  -c  L*s
            0  1   0  0
            0  0   0  1];

    %   Link 2:
    c    = cos(q(2,i));
    s    = sin(q(2,i)); L = L2;
    T21  = [c  -s  0  L*c
            s   c  0  L*s
            0   0  1   0
            0   0  0   1];
    L    = L2/2;
    T21c = [c  -s  0  L*c
            s   c  0  L*s
            0   0  1   0
            0   0  0   1];

    T20  = T10*T21;
    T20c = T10*T21c;
```

```
    % assign center of mass position vectors
    rc1 = T10c(1:3,4);
    rc2 = T20c(1:3,4);
    g = modelParameters.g*[1; 0; 0];
    m = modelParameters.m;

    % calculate the gravitational potential energy
    V(i) = -(m*g'*rc1 + m*g'*rc2 );

    % calculate the total system energy
    E(i) = V(i) + T(i);
end

% plot the energy terms as a function of time
figure; plot(t,E,'b',t, V,'g',t,T,'r');
legend('total energy', 'potential energy', 'kinetic energy')
xlabel('Time [seconds]');ylabel('Energy [joules]');
title({'System energy as a function of time','HW 3, problem 2'});
axis([0 max(t) -1.5*max(E) 2*max(E)])
```

```
%zDot2dof.m
%problem 2, HW 3. ME 739, Univ. Wisconsin Madison
%called from RK-4 numerical integration method
%Nasser M. Abbasi
function [zDot] = zDot2dof(z,modelParameters);

    % assign joint displacements / velocities from state variables
    q = [z(1); z(2)];
    qd = [z(3); z(4)];

    % calculate D, B, D, and G matrices
    D = Dmatrix2dof(q,modelParameters);
    B = Bmatrix2dof(q,modelParameters);
    C = Cmatrix2dof(q,modelParameters);
    G = Gvector2dof(q,modelParameters);

    %viscous friction in joints
    tau_friction = 0; %5*qd;  %5*qd;

    %acceleration
    qdqd = [qd(1)*qd(2)];
    qd2 =  [qd(1)*qd(1); qd(2)*qd(2)];
    qdd = D\(-B*qdqd - C*qd2 - G - tau_friction);

    % assign state variable derivatives
    zDot = [qd; qdd];
```

```
end
```

```
function G = Gvector2dof(q,modelParameters)

    % assign model parameters to local variables
    g = modelParameters.g;
    m = modelParameters.m;
    L = modelParameters.L;

    G=zeros(2,1);

    G(1,1) = 1/2*L*m*g*sin(q(1))+L*g*m*sin(q(1))+1/2*L*g*m*cos(q(2))*sin(q(1
    G(2,1) = 1/2*L*g*m*cos(q(1))*sin(q(2));

end
```

```
%Dmatrix2dof.m
%probvlem 2, HW 3. ME 739, Univ. Wisconsin Madison
%build the mass matrix D
%Nasser M. Abbasi
function D = Dmatrix2dof(q,modelParameters)

    % assign model parameters to local variables

    m = modelParameters.m;
    Ia = modelParameters.Ia;
    L = modelParameters.L;

    D=zeros(2,2);
    D(1,1) = m*(Ia+1/4*L^2)+1/8*m*(6*Ia+9*L^2+8*L^2*cos(q(2))...
        +(2*Ia+L^2)*cos(2*q(2)));
    D(1,2) = 0;
    D(2,1) = D(1,2);
    D(2,2) = m*(Ia+L^2/4);

end
```

```
%Cmatrix2dof.m
%probvlem 2, HW 3. ME 739, Univ. Wisconsin Madison
%build the C matrix
%Nasser M. Abbasi
function C = Cmatrix2dof(q,modelParameters)

    % assign model parameters to local variables
```

```
    m = modelParameters.m;
    Ia = modelParameters.Ia;
    L = modelParameters.L;

    C=zeros(2,2);

    C(1,1) = 0;
    C(1,2) = 0;
    C(2,1) = 1/2*m*L^2*sin(q(2))+1/8*m*(2*Ia+L^2)*sin(2*q(2));
    C(2,2) = 0;

end
```

```
%Bmatrix2dof.m
%probvlem 2, HW 3. ME 739, Univ. Wisconsin Madison
%build the B matrix
%Nasser M. Abbasi
function B = Bmatrix2dof(q,modelParameters)

    % assign model parameters to local variables

    m = modelParameters.m;
    Ia = modelParameters.Ia;
    L = modelParameters.L;

    B=zeros(2,1);

    B(1,1) = -m*L^2*sin(q(2))-1/4*m*(2*Ia+L^2)*sin(2*q(2));
    B(2,1) = 0;

end
```

2.3.5 key solution for HW 3

ME/ECE 739: Advar Homework #3

Due: April 1st (Wednesd

Please submit your answers to th ncluding your Matlab scripts, and, where appropriate, program results (p ripts should be readable, with comments, sensible variable names, inden addition to the hardcopy (pdf format), you must also submit your Matlab scrip Learn@UW course page dropbox (e.g. Homework #3) using a zip archive file format. zip files using your last name and hw# (e.g. zinn_hw3.zip).

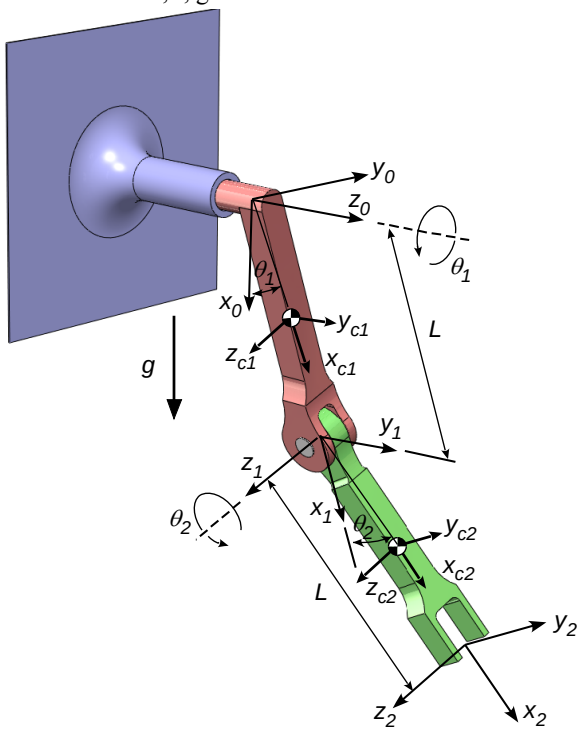
Problem 1 [20 points]

Following the steps outlined below, derive the equations of motion of the two degree-of-freedom manipulator depicted. The links have mass, m, and an inertia tensor, I, given as.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}I_a & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_a \end{bmatrix}$$

The location of the center of mass is located in the middle of each link.

Note: you are strongly advised to use the Matlab symbolic toolbox (or equivalent) for your algebraic and differential operations.



► On each link we have attached frames at the joints and at the center of mass of each link. Calculate the homogeneous transforms that relate these frames to the inertial frame $\{0\}$. In other words, find, T_1^0, T_2^0, $T_{C_1}^0$, and $T_{C_2}^0$. Verify that your matrices are correct before proceeding to the next step (e.g. check the location and orientation of your frames for simple configurations such as $\theta_1 = \theta_2 = 90°$)

SOLUTION:

The homogeneous transformation matrices relating successive frames are given as

$$T_1^0 = \left[\begin{array}{ccc|c} c_1 & 0 & s_1 & Lc_1 \\ s_1 & 0 & -c_1 & Ls_1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad T_{C_1}^0 = \left[\begin{array}{ccc|c} c_1 & 0 & s_1 & \frac{1}{2}Lc_1 \\ s_1 & 0 & -c_1 & \frac{1}{2}Ls_1 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_2^1 = \left[\begin{array}{ccc|c} c_2 & -s_2 & 0 & Lc_2 \\ s_2 & c_2 & 0 & Ls_2 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad T_{C_2}^1 = \left[\begin{array}{ccc|c} c_2 & -s_2 & 0 & \frac{1}{2}Lc_2 \\ s_2 & c_2 & 0 & \frac{1}{2}Ls_2 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The homogeneous transformation matrices relating the link frames to the ground frame are calculated as:

$$T_2^0 = T_1^0 T_2^1 = \left[\begin{array}{ccc|c} c_1 & 0 & s_1 & Lc_1 \\ s_1 & 1 & -c_1 & Ls_1 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]\left[\begin{array}{ccc|c} c_2 & -s_2 & 0 & Lc_2 \\ s_2 & c_2 & 0 & Ls_2 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} c_1c_2 & -c_1s_2 & s_1 & Lc_1(1+c_2) \\ s_1c_2 & -s_1s_2 & -c_1 & Ls_1(1+c_2) \\ s_2 & c_2 & 0 & Ls_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$T_{C_2}^0 = T_1^0 T_{C_2}^1 = \left[\begin{array}{ccc|c} c_1 & 0 & s_1 & Lc_1 \\ s_1 & 1 & -c_1 & Ls_1 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]\left[\begin{array}{ccc|c} c_2 & -s_2 & 0 & \frac{1}{2}Lc_2 \\ s_2 & c_2 & 0 & \frac{1}{2}Ls_2 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} c_1c_2 & -c_1s_2 & s_1 & Lc_1(1+\frac{1}{2}c_2) \\ s_1c_2 & -s_1s_2 & -c_1 & Ls_1(1+\frac{1}{2}c_2) \\ s_2 & c_2 & 0 & \frac{1}{2}Ls_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

► The evaluation of the mass matrix of the manipulator will require the computation of the linear Jacobian of the center of mass, $J_{v_{c_i}}$, for each link and the angular velocity Jacobian for each link, J_{ω_i}.
  - Find $J_{v_{c_1}}$ and $J_{v_{c_2}}$.
  - Find J_{ω_1} and J_{ω_2}.

  Verify that your matrices are correct before proceeding to the next step.

SOLUTION:

The linear and angular velocity Jacobians (associated with the center-of-mass of each link) can be evaluated using the explicit form. In this case, the link frame definitions must be consistent with our D-H frame rules. Specifically, the z-axis of link *i* must be aligned with joint axis of link *i*+1. The frame definitions shown in the figure above conform to this requirement.

Linear velocity Jacobian: center of mass of link 1:

$$J_{v_{C_1}} = \left[\begin{array}{c|c} \hat{z}_0^0 \times o_{C_1}^0 & \begin{matrix}0\\0\\0\end{matrix} \end{array}\right]$$

Linear velocity Jacobian: center of mass of link 2:

$$J_{v_{C_2}} = \left[\begin{array}{c|c} \hat{z}_0^0 \times o_{C_2}^0 & \hat{z}_1^0 \times \left(o_{C_2}^0 - o_1^0\right) \end{array}\right]$$

Angular velocity Jacobian of link 1:

$$J_{\omega_1} = \left[\begin{array}{c|c} \hat{z}_0^0 & \begin{matrix}0\\0\\0\end{matrix} \end{array}\right]$$

Angular velocity Jacobian of link 2:

$$J_{\omega_1} = \left[\begin{array}{c|c} \hat{z}_0^0 & \hat{z}_1^0 \end{array}\right]$$

where:

$$\hat{z}_0^0 = \begin{bmatrix}0\\0\\1\end{bmatrix}, \quad \hat{z}_1^0 = \begin{bmatrix}s_1\\-c_1\\1\end{bmatrix} \text{ (3rd column, rows 1-3 of } T_1^0\text{)},$$

$$o_{C_1}^0 = \begin{bmatrix}\frac{1}{2}Lc_1\\ \frac{1}{2}Ls_1\\0\end{bmatrix} \text{ (4th column, rows 1-3 of } T_{C_1}^0\text{)}, \quad o_{C_2}^0 = \begin{bmatrix}Lc_1\left(1+\frac{1}{2}c_2\right)\\ Ls_1\left(1+\frac{1}{2}c_2\right)\\ \frac{1}{2}Ls_2\end{bmatrix} \text{(4th column, rows 1-3 of } T_{C_2}^0\text{)}$$

$$o_1^0 = \begin{bmatrix}Lc_1\\Ls_1\\0\end{bmatrix} \text{ (4th column, rows 1-3 of } T_1^0\text{)}$$

► Evaluate the mass matrix, $D(q)$, of the manipulator in terms of mass and geometric properties and its configuration.

SOLUTION:

The mass matrix for the two link manipulator is evaluated as

$$D(q) = \underbrace{D_{v_1} + D_{v_2}}_{\substack{\text{due to linear motion} \\ \text{of links COM}}} + \underbrace{D_{\omega_1} + D_{\omega_2}}_{\substack{\text{due to rotational} \\ \text{motion of links}}} = \begin{bmatrix} \frac{1}{4}mL^2 + \frac{1}{4}mL^2\left(c_2+2\right)^2 + 2I_a - \frac{1}{2}I_a {s_2}^2 & 0 \\ 0 & \frac{1}{4}mL^2 + I_a \end{bmatrix}$$

where

$$D_{v_1} = m_1 J_{v_{C_1}}^T J_{v_{C_1}} = \begin{bmatrix} \frac{1}{4}mL^2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$D_{v_2} = m_2 J_{v_{C_2}}^T J_{v_{C_2}} = \begin{bmatrix} \frac{1}{4}mL^2\left(c_2+2\right)^2 & 0 \\ 0 & \frac{1}{4}mL^2 \end{bmatrix}$$

$$D_{\omega_1} = J_{\omega_1}^T R_1^0 I_{C_1} R_1^{0T} J_{\omega_1} = \begin{bmatrix} I_a & 0 \\ 0 & 0 \end{bmatrix}$$

$$D_{\omega_2} = J_{\omega_2}^T R_2^0 I_{C_2} R_2^{0T} J_{\omega_2} = \begin{bmatrix} \frac{1}{2}I_a\left(2-{s_2}^2\right) & 0 \\ 0 & I_a \end{bmatrix}$$

The inertia tensors, I_{C_1} and I_{C_2}, are defined in the problem statement and the linear and angular velocity Jacobians were evaluated in the previous question. The rotation matrices, R_1^0 and R_2^0, are given as

$$R_1^0 = \begin{bmatrix} c_1 & 0 & s_1 \\ s_1 & 0 & -c_1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{(row and columns 1-3 of } T_1^0\text{)}$$

$$R_2^0 = \begin{bmatrix} c_1c_2 & -c_1s_2 & s_1 \\ s_1c_2 & -s_1s_2 & -c_1 \\ s_2 & c_2 & 0 \end{bmatrix} \quad \text{(row and columns 1-3 of } T_2^0\text{)}$$

► Evaluate the centrifugal and Coriolis inertial terms of the manipulator in terms of mass and geometric properties and its configuration.
  - Find $B(q)[\dot{q}\dot{q}]$
  - Find $C(q)[\dot{q}^2]$

SOLUTION:

For the two-degree-of-freedom manipulator shown, the Coriolis and centrifugal terms are evaluated as

$$B(q)[\dot{q}\dot{q}] = \begin{bmatrix} 2b_{1,12} \\ 2b_{2,12} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1\dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -\left(\tfrac{1}{2}I_a + \tfrac{1}{4}mL^2\right)\sin(2\theta_2) - mL^2\sin(\theta_2) \\ 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1\dot{\theta}_2 \end{bmatrix}$$

$$C(q)\left[\dot{q}^2\right] = \begin{bmatrix} b_{1,11} & b_{1,22} \\ b_{2,11} & b_{2,22} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1^2 \\ \dot{\theta}_2^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \left(\tfrac{1}{4}I_a + \tfrac{1}{8}mL^2\right)\sin(2\theta_2) + \tfrac{1}{2}mL^2\sin(\theta_2) & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1^2 \\ \dot{\theta}_2^2 \end{bmatrix}$$

where the Christoffel symbols are defined as $b_{i,jk} = \frac{1}{2}\left(\frac{\partial d_{ij}}{\partial q_k} + \frac{\partial d_{ik}}{\partial q_j} - \frac{\partial d_{jk}}{\partial q_i}\right) = \frac{1}{2}\left(d_{ijk} + d_{ikj} - d_{jki}\right)$. The evaluation of the elements of the *B* and *C* matrices are given below.

$$\begin{aligned}
b_{1,11} &= \tfrac{1}{2}\left(d_{111} + d_{111} - d_{111}\right) &&= \tfrac{1}{2}d_{111} &&= 0 \\
b_{1,12} &= \tfrac{1}{2}\left(d_{112} + d_{121} - d_{121}\right) &&= \tfrac{1}{2}d_{112} &&= -\left(\tfrac{1}{4}I_a + \tfrac{1}{8}mL^2\right)\sin(2\theta_2) - \tfrac{1}{2}mL^2\sin(\theta_2) \\
b_{1,22} &= \tfrac{1}{2}\left(d_{122} + d_{122} - d_{221}\right) &&= d_{122} - \tfrac{1}{2}d_{221} &&= 0 \\
b_{2,11} &= \tfrac{1}{2}\left(d_{211} + d_{211} - d_{112}\right) &&= d_{211} - \tfrac{1}{2}d_{112} &&= \left(\tfrac{1}{4}I_a + \tfrac{1}{8}mL^2\right)\sin(2\theta_2) + \tfrac{1}{2}mL^2\sin(\theta_2) \\
b_{2,12} &= \tfrac{1}{2}\left(d_{212} + d_{221} - d_{122}\right) &&&&= 0 \\
b_{2,22} &= \tfrac{1}{2}\left(d_{222} + d_{222} - d_{222}\right) &&= \tfrac{1}{2}d_{222} &&= 0
\end{aligned}$$

where the partial derivatives of the mass matrix elements are evaluated below.

$$d_{111} = \frac{\partial d_{11}}{\partial \theta_1} = 0 \qquad d_{112} = \frac{\partial d_{11}}{\partial \theta_2} = -\tfrac{1}{2}I_a\sin(2\theta_2) - mL^2\sin(\theta_2) - \tfrac{1}{4}mL^2\sin(2\theta_2)$$

$$d_{122} = \frac{\partial d_{12}}{\partial \theta_2} = 0$$

$$d_{211} = \frac{\partial d_{21}}{\partial \theta_1} = 0 \qquad d_{212} = \frac{\partial d_{21}}{\partial \theta_2} = 0$$

$$d_{221} = \frac{\partial d_{22}}{\partial \theta_1} = 0 \qquad d_{222} = \frac{\partial d_{22}}{\partial \theta_2} = 0$$

► Evaluate the gravity vector, $G(q)$. In frame $\{0\}$ the gravity vector is given as: $\mathbf{g} = g[1 \quad 0 \quad 0]^T$.

SOLUTION:

For the two-degree-of-freedom manipulator shown, the gravity vector is evaluated as

$$G(q) = -\left(J_{v_{C_1}}^T m_1 g + J_{v_{C_2}}^T m_2 g \right) = \begin{bmatrix} \frac{1}{2} mgLs_1 (c_2 + 3) \\ \frac{1}{2} mgLc_1 s_2 \end{bmatrix}$$

The linear velocity Jacobians for the link COMs were evaluated in the previous question. The gravitational vector g is defined in the problem statement (figure) and is given as $g = \begin{bmatrix} g & 0 & 0 \end{bmatrix}^T$.

► Form the complete equations of motion in the form

$$D(q)\ddot{q} + B(q)[\dot{q}\dot{q}] + C(q)[\dot{q}^2] + G(q) = \tau$$

SOLUTION:

The assembled equations of motion are given as

$$\begin{bmatrix} \frac{1}{4} mL^2 + \frac{1}{4} mL^2 (c_2 + 2)^2 + 2I_a - \frac{1}{2} I_a {s_2}^2 & 0 \\ 0 & \frac{1}{4} mL^2 + I_a \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \cdots$$

$$\begin{bmatrix} -\left(\frac{1}{2} I_a + \frac{1}{4} mL^2 \right) \sin(2\theta_2) - mL^2 \sin(\theta_2) \\ 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \dot{\theta}_2 \end{bmatrix} + \cdots$$

$$\begin{bmatrix} 0 & 0 \\ \left(\frac{1}{4} I_a + \frac{1}{8} mL^2 \right) \sin(2\theta_2) + \frac{1}{2} mL^2 \sin(\theta_2) & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1^2 \\ \dot{\theta}_2^2 \end{bmatrix} + \begin{bmatrix} \frac{1}{2} mgLs_1 (c_2 + 3) \\ \frac{1}{2} mgLc_1 s_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_1 \end{bmatrix}$$

SOLUTION:

The symbolic solution presented above was derived (in part) using the Matlab symbolic toolbox. The Matlab script used to evaluate the solution has been posted to the Learn@UW course page.

Problem 2. [20 points]

Using your results from the previous question, write a Matlab numerical simulation of the manipulator described in Question 2. Implement a 4th order Runge-Kutta numerical integration of your equations of motion. The model and simulation parameters are given below.

Geometric parameters:	$L = 2$ meters
Mass properties:	$m = 10$ kg
	$I_a = 5$ kg/m^2
	$g = 9.81$ m/s^2
Simulation parameters:	$\Delta t = 0.01$ seconds (integration time step)
	$T_{final} = 20$ seconds (simulated time duration)

Note, to help you develop your code, some example code has been posted to the course Learn@UW page. Please use these scripts as a starting point. I would advise you to review all of the posted examples before you start working on your code.

- Simulate (*and animate*) the release of the manipulator from a vertical position (i.e. $q_1 = 180°, q_2 = 0$) with small initial joint velocity ($\dot{q}_1 = \dot{q}_2 = 0.1\, r/s$). You may want to add a small amount of damping to the joint torque vector such that the system starts to settle after a few major oscillations (suggested $c_i = 1$ N-m/r/s where $\tau_i = -c_i \dot{q}_i$).
- To check that you simulation (and equations of motion) are correct, calculate and plot the system energy for the case where the damping coefficients are set equal to zero. Specifically, please plot the total system kinetic energy, T, the total system potential energy, V, and the total system energy ($T + V$) as a function of time. If your simulation and underlying equations of motion are correct, the total energy should stay constant.

SOLUTION:

The Matlab scripts used to evaluate the solution have been posted to the Learn@UW course page. To simulation is executed by running the m-file `TwoDOFDynamicsHW3.m`. The rendering functions introduced earlier in the semester are also required to display the results. A video of the simulation output is included with the posted archive.

2.4 HW 4

2.4.1 Problem 1

Problem 1 [20 points]

► Write a Matlab function which constructs a quintic polynomial, for the purposes of trajectory generation, given the following input parameters

t_0:	Initial time	t_f:	Final time
y_0:	Initial position	y_f:	Final position
$\dot{y}_0$:	Initial velocity	$\dot{y}_f$:	Final velocity
$\ddot{y}_0$:	Initial acceleration	$\ddot{y}_f$:	Final acceleration

An example function prototype is shown below

$$[t,\ y,\ \dot{y},\ \ddot{y}] = \texttt{QuinticPolynomial}(t_0,\ t_f,\ y_0,\ y_f,\ \dot{y}_0,\ \dot{y}_f,\ \ddot{y}_0,\ \ddot{y}_f,\ n)$$

where n is the number of elements in the time vector and output arrays ($[y,\ \dot{y},\ \ddot{y}]$)

t	:	n-dimensional array of trajectory time
y	:	n-dimensional array of trajectory position
$\dot{y}$	:	n-dimensional array of trajectory velocity
$\ddot{y}$	:	n-dimensional array of trajectory acceleration

► To check your function, create and plot the trajectory for the input parameters given below:

t_0:	0	t_f:	5
y_0:	-10	y_f:	10
$\dot{y}_0$:	-50	$\dot{y}_f$:	-50
$\ddot{y}_0$:	0	$\ddot{y}_f$:	0

e.g.
```
[t, y, dy, ddy] = QuinticPolynomial(0,5,-10,10,-50,50,0,0,1000);
figure; plot(t,y);
figure; plot(t, dy);
figure; plot(t, ddy);
```

Figure 2.22: Problem 1 description

A Matlab function called `QuinticPolynomial` was implemented. The following shows the call made and the three plots generated. The output matched the required output in the problem statement.

```
[t,y,dy,dyy] = QuinticPolynomial(0,5,-10,10,-50,-50,0,0,1000);
figure;
plot(t,y);
title('HW4, problem1, y(t) solution')

figure;
plot(t,dy);
title('HW4, problem1, dy(t) solution')

figure;
plot(t,dyy);
title('HW4, problem1, ddy(t) solution')
```

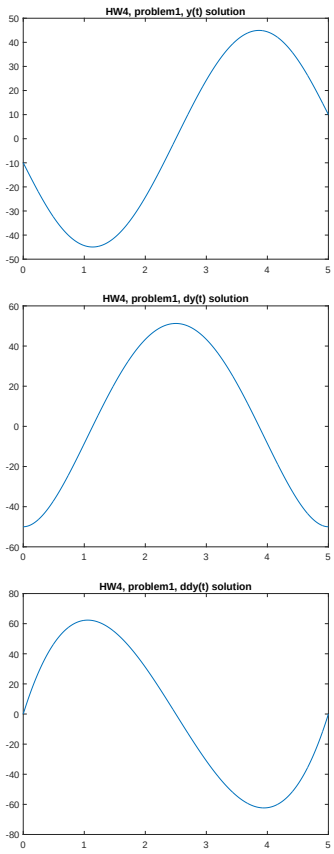


Figure 2.23: Problem 1 plots

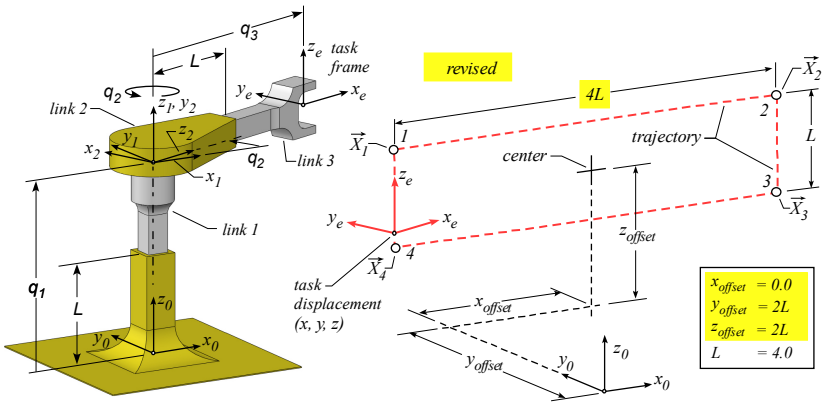
The following is the Matlab source code listings of the above function.

```
function [t,y,dy,ddy] = QuinticPolynomial(t0,tf,y0,yf,dy0,dyf,ddy0,ddyf,n)
%function QuinticPolynomial to solve trajectory generation using quintic
%polynomial method
%ME 739, UW Madison, Spring 2015
%by Nasser M. Abbasi

%INPUT:
%t0   : initial time
%tf   : end time
%y0   : initial position
%yf   : final position
%dy0  : initial speed
%dyf  : final speed
%ddy0 : initial acceleration
%ddyf : final acceleration
%n    : number of samples
%
%OUTPUT
%t    : time vector
%y    : position vector
%dy   : speed vector
%ddy  : acceleration vector

syms t a0 a1 a2 a3 a4 a5;
%set up the polynomial
y   = a0+a1*t+a2*t^2+a3*t^3+a4*t^4+a5*t^5;
dy  = diff(y,t);
ddy = diff(dy,t);

%setup the 6 contraints
eq1 = subs(y,t,t0)==y0;
eq2 = subs(y,t,tf)==yf;
eq3 = subs(dy,t,t0)==dy0;
eq4 = subs(dy,t,tf)==dyf;
eq5 = subs(ddy,t,t0)==ddy0;
eq6 = subs(ddy,t,tf)==ddyf;

%solve for the unknowns
[a0,a1,a2,a3,a4,a5]=solve(eq1,eq2,eq3,eq4,eq5,eq6);

%set up time vector
t = linspace(t0,tf,n);

%use subs to replace all unknowns and time in the polynomials
y   = double(subs(y));
```

```
dy  = double(subs(dy));
ddy = double(subs(ddy));
end
```

2.4.2 Problem 2

Problem 2. [40 points]



Overview of 3-DOF manipulator **Task-space trajectory (rectangular)**

You are to generate the motion trajectory for the three degree of freedom manipulator shown in the above figure such that the manipulator's task frame (origin) moves between the four points that describe a square in space (see Figure). The forward and inverse kinematics are given below.

Forward kinematics:	*Inverse kinematics:*
$x_e = q_3 \cos(q_2)$	$q_1 = z_e$
$y_e = q_3 \sin(q_2)$	$q_2 = \tan^{-1}(y_e / x_e)$
$z_e = q_1$	$q_3 = \sqrt{x_e^2 + y_e^2}$

For reference, the homogeneous transformations between the successive link frames (defined in the figure above) are given below. These may be useful when animating your results in Problem 3.

$$T_1^0 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]; \quad T_2^1 = \left[\begin{array}{ccc|c} -s_2 & 0 & c_2 & 0 \\ c_2 & 0 & s_2 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]; \quad T_e^2 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & q_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Problem 2. *continued*

Scenario 1: Joint-space trajectory generation

- Calculate the joint space displacements that correspond to the given task-space waypoints $\left[\vec{X}_1, \vec{X}_2, \vec{X}_3, \text{and } \vec{X}_4\right]$ (using the inverse kinematics). The task space Cartesian coordinates for each waypoint can be determined from the adjacent figure.
- Generate the four part trajectory, moving from waypoint *1* to *2*, *2* to *3*, *3* to *4*, and *4* to *1*. The time required to move between successive waypoint is 5 seconds. Generate the trajectory in joint-space, using the joint-space displacements calculated above. The trajectories should follow a quintic polynomial. At each waypoint, the joint-space velocity and acceleration should equal zero.
- Plot your results to include:
  - Joint space displacements as a function of time.
  - Task space displacements as a function of time (using the forward kinematics to calculate the task space displacements).
  - Task space displacements in 3-D space (using the Matlab `plot3` command).

Scenario 2: Task-space trajectory generation

- Generate the four part trajectory, moving from waypoint *1* to *2*, *2* to *3*, *3* to *4*, and *4* to *1*. The time required to move between successive waypoint is 5 seconds. Generate the trajectory in task-space, using the given task-space waypoints. The trajectories should follow a quintic polynomial. At each waypoint, the task-space velocity and acceleration should equal zero.
- Plot your results to include:
  - Joint space displacements as a function of time (using the inverse kinematics to calculate the joint-space displacements).
  - Task space displacements as a function of time.
  - Task space displacements in 3-D space (using the Matlab `plot3` command).
- Comment on the differences between the two approaches (advantages and disadvantages).

Figure 2.24: Problem 2 description

The following 2D diagram illustrates the task space trajectory which is the path that the end effector will travel over.

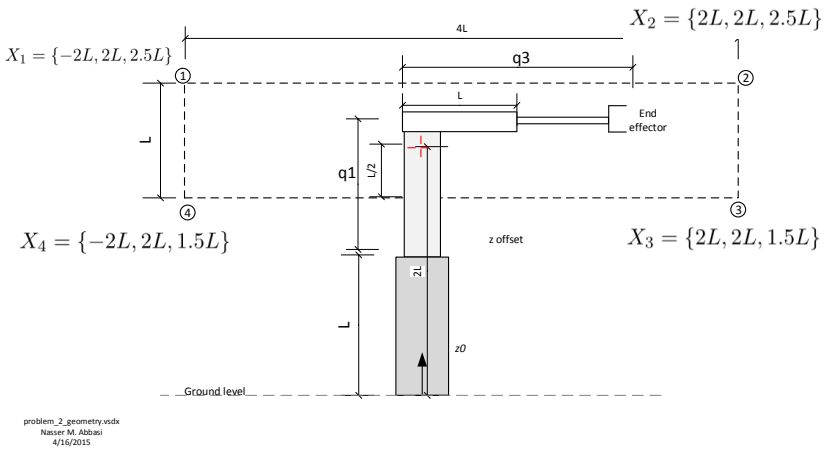


Figure 2.25: Problem 2 task path

First scenario

PART ONE:

The waypoints have the following coordinate values by inspection from the above diagram

$$
\begin{aligned}
X_1 &= \{-2L, 2L, 2.5L\} \\
X_2 &= \{2L, 2L, 2.5L\} \\
X_3 &= \{2L, 2L, 1.5L\} \\
X_4 &= \{-2L, 2L, 1.5L\}
\end{aligned}
$$

Inverse kinematics was used to determine the joint space displacements that corresponds to the above task space. For X_1 this results in

$$
\begin{aligned}
q_1 &= z_e = 2.5L \\
q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{-2L}\right) = 135^0 \\
q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2}
\end{aligned}
$$

For X_2

$$
\begin{aligned}
q_1 &= z_e = 2.5L \\
q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{2L}\right) = 45^0 \\
q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2}
\end{aligned}
$$

For X_3

$$
\begin{aligned}
q_1 &= z_e = 1.5L \\
q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{2L}\right) = 45^0 \\
q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2}
\end{aligned}
$$

And for X_4

$$
\begin{aligned}
q_1 &= z_e = 1.5L \\
q_2 &= \tan^{-1}\left(\frac{y_e}{x_e}\right) = \tan^{-1}\left(\frac{2L}{-2L}\right) = 135^0 \\
q_3 &= \sqrt{x_e^2 + y_e^2} = \sqrt{4L^2 + 4L^2} = 2L\sqrt{2}
\end{aligned}
$$

PART TWO

The trajectory in joint space is now generated. The joint space coordinates was found above and illustrated in the following diagram

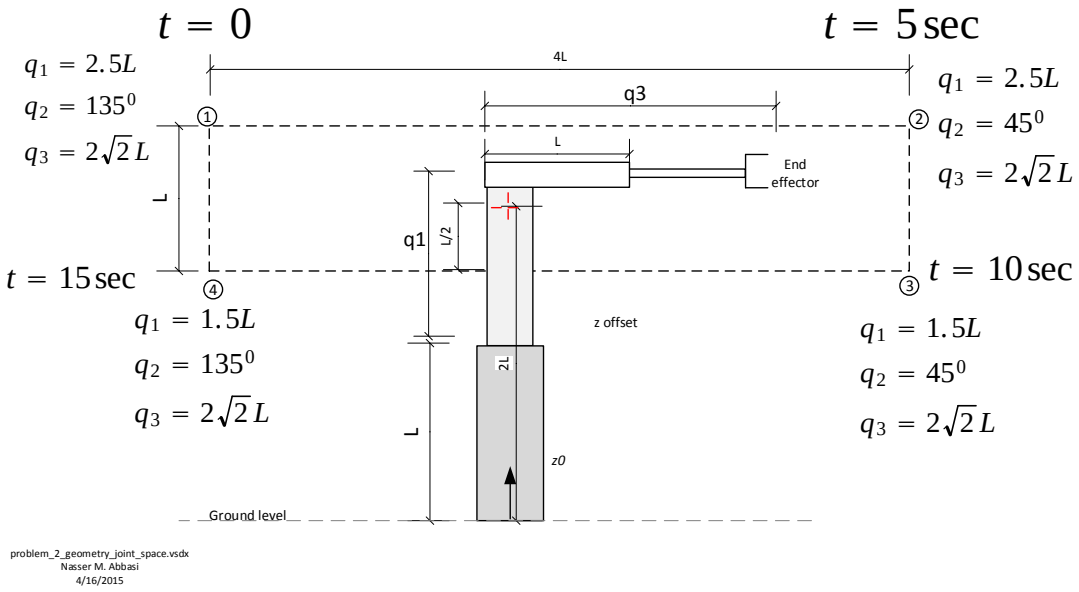


Figure 2.26: Problem 2 task path expressed in joint coordinates

A quintic polynomial was used with the restriction that at each waypoint $\dot{q} = 0$ and also $\ddot{q} = 0$.

The above was done for each joint space path between two points. There are 3 joints. For each joint the full path was generated. For example, for joint q_1, the polynomial between X_1 and X_2 was found by solving 6 equations in 6 unknowns. The polynomial is defined as

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

The 6 equations are

$$
\begin{aligned}
q(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 = 2.5L \\
\dot{q}(t_0) &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 = 0 \\
\ddot{q}(t_0) &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 = 0 \\
q(t_f) &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 = 2.5L \\
\dot{q}(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 = 0 \\
\ddot{q}(t_f) &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 = 0
\end{aligned}
$$

Where $t_0 = 0$ and $t_f = 5$. The above 6 equations are solved for $a_0, a_1, a_2, a_3, a_4, a_5$ which now gives the polynomial $q(t)$ in order to obtain the joint coordinate at any time $0 < t < 5$.

$$
\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\
1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\
0 & 1 & 2t_f & 3t_f^2 & 4t_0^3 & 5t_f^4 \\
0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}
=
\begin{bmatrix} q(t_0) \\ \dot{q}(t_0) \\ \ddot{q}(t_0) \\ q(t_f) \\ \dot{q}(t_f) \\ \ddot{q}(t_f) \end{bmatrix}
=
\begin{bmatrix} 2.5L \\ 0 \\ 0 \\ 2.5L \\ 0 \\ 0 \end{bmatrix}
$$

Similarly, these 6 equations are solved for the next segment between X_2, X_3

$$
\begin{aligned}
q(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 = 2.5L \\
\dot{q}(t_0) &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 = 0 \\
\ddot{q}(t_0) &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 = 0 \\
q(t_f) &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 = 1.5L \\
\dot{q}(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 = 0 \\
\ddot{q}(t_f) &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 = 0
\end{aligned}
$$

Where in the above $t_0 = 5$ and $t_f = 10$.

The above was done for all the segments. The same process was repeated for joints q_2 and q_3.

A Matlab program listed below was written to implement the above. The result generated is given below the source code listing.

```
function  nma_HW4_problem_2_part1()
%nma_HW4_problem_2_part1()
%This function implements problem 2, HW 4, ME 739, scenario ONE
%by Nasser M. Abbasi
%
close all;

L=4;
```

```
%generate the joint space (q) displacements, velocity and acceleration
%the value of q_1  at each X1,X2,X3,X4 have been determined from the
%inverse kinmatics as shown in the HW report above. These are now used
%to generate the polynomial

%generate polynomial for q1 trajectory and position, speed and acc. plots
figure;
[t1,q1]=process_q(2.5*L,2.5*L,1.5*L,1.5*L,'q1',1.4*L,2.6*L,...
    'meter','m/s','m/s^2');

%generate polynomial for q2 trajectory and position, speed and acc. plots
figure;
[t2,q2]=process_q(135*pi/180,45*pi/180,45*pi/180,135*pi/180,...
    'q2',40*pi/180,155*pi/180,'angle(rad)','rad/sec','rad/sec^2');

%generate polynomial for q3 trajectory and position, speed and acc. plots
figure;
z=2*L*sqrt(2);
[t3,q3]=process_q(z,z,z,z,'q3',0,1.2*2*L*sqrt(2),'meter','m/s','m/s^2');

%generate the task space X displacement using the forward kinematics
xe = q3.*cos(q2);
ye = q3.*sin(q2);
ze = q1;

%generate task space displacements
figure;
h1 = subplot(3,1,1);
plot(t1,xe);
title(h1,{'Task space displacements','Xe'});
xlabel(h1,'time (sec)');
ylabel(h1,'meter');
axis(h1,[0 20 -2.2*L 2.2*L]);

h2 = subplot(3,1,2);
plot(t2,ye);
title(h2,'Ye');
xlabel(h2,'time (sec)');
ylabel(h2,'meter');
axis(h2,[0 20 1.8*L 3*L]);

h3 = subplot(3,1,3);
plot(t3,ze);
title(h3,'Ze');
xlabel(h3,'time (sec)');
ylabel(h3,'meter');
axis(h3,[0 20 1.4*L 2.6*L]);
```

```
%generate 3D plot of the task space trajectory
figure;
plot3(xe,ye,ze);
hold on;
plot3(-2*L,2*L,2.5*L,'o');
text(-2*L,2*L,2.5*L,'X1');

plot3(2*L,2*L,2.5*L,'o');
text(2*L,2*L,2.5*L,'X2');

plot3(2*L,2*L,1.5*L,'o');
text(2*L,2*L,1.5*L,'X3');

plot3(-2*L,2*L,1.5*L,'o');
text(-2*L,2*L,1.5*L,'X4');

plot3(0,2*L,2*L,'+');
text(.1*L,2*L,2.1*L,'center');

title('3D task space displacement');
xlabel('X'); ylabel('Y'); zlabel('Z');
zlim([0 3*L]);

end
%-----------------------
function [t,q]=process_q(x1,x2,x3,x4,the_name,lim0,lim1,y1,y2,y3)
%
%Function to generate joint space trajectory for problem 2 for specific
%joint q
%x1: first point coordinate in this joint  space
%x2: second point coordinate in this joint  space
%x3: third point coordinate in this joint  space
%x4: fourth point coordinate in this joint  space
%the_name: title to put on the plot, for example 'q1' or 'q2' etc..
%lim0 and lim1 are the y-axis limits to use for the plot
%y1: position y label
%y2: speed y label
%y3: acceleration y label

%RETURNS
%q: which is vector of q joint coordinates in joint space over
%the full time space to be used later to generate the task space
%trajectory using forward kinemetics
%
%t: the corresponding time vector
```

```
%call gen_path to obtain the polynomial coefficients
a              = gen_path(x1,x2,0,5);
[q1,dq,ddq,t1] = find_q(0,5,a); %use the coefficients to make polynomials
[h1,h2,h3]     = make_first_plot(t1,q1,dq,ddq);

title(h1,{the_name,'position'});
title(h2,'velocity');
title(h3,'acceleration');
xlabel(h1,'time (sec)'); ylabel(h1,y1);
xlabel(h2,'time (sec)'); ylabel(h2,y2);
xlabel(h3,'time (sec)'); ylabel(h3,y3);

a              = gen_path(x2,x3,5,10);
[q2,dq,ddq,t2] = find_q(5,10,a);
make_other_plot(t2,q2,dq,ddq,h1,h2,h3);


a              = gen_path(x3,x4,10,15);
[q3,dq,ddq,t3] = find_q(10,15,a);
make_other_plot(t3,q3,dq,ddq,h1,h2,h3);

a              = gen_path(x4,x1,15,20);
[q4,dq,ddq,t4] = find_q(15,20,a);
make_other_plot(t4,q4,dq,ddq,h1,h2,h3);

axis(h1,[0 20 lim0 lim1]);
q=[q1 q2 q3 q4];
t=[t1 t2 t3 t4];
end

%-------------------------------------------
function a = gen_path(q0,qf,t0,tf)
%this function generates the polynomial coefficients by solving for the
%constraints given

dq0  = 0;
dqf  = 0;
ddq0 = 0;
ddqf = 0;

C = [ 1  t0  t0^2  t0^3    t0^4     t0^5;
      0  1   2*t0  3*t0^2  4*t0^3   5*t0^4;
      0  0   2     6*t0    12*t0^2  20*t0^3;
      1  tf  tf^2  tf^3    tf^4     tf^5;
      0  1   2*tf  3*tf^2  4*tf^3   5*tf^4;
      0  0   2     6*tf    12*tf^2  20*tf^3];
```

```
q  = [q0 dq0 ddq0 qf dqf ddqf]';
a  = C\q;

end

%-------------------------------------------------
function [q,dq,ddq,t]=find_q(t0,tf,a)
%This function takes the coefficients of the polynomial and
%return back the polynomials for position, speed and acceleration
a0  = a(1); a1 = a(2); a2 = a(3); a3 = a(4); a4 = a(5); a5 = a(6);
t   = linspace(t0,tf,1000);
q   = a0   + a1*t   + a2*t.^2    + a3*t.^3    + a4*t.^4     + a5*t.^5;
dq  = a1   + 2*a2*t + 3*a3*t.^2  + 4*a4*t.^3  + 5*a5*t.^4;
ddq = 2*a2 + 6*a3*t + 12*a4*t.^2 + 20*a5*t.^3;

end

%-------------------------------------------------------
function [h1,h2,h3]=make_first_plot(t,q,dq,ddq)
%this function plots the position in joint space
h1 = subplot(3,1,1);
plot(t,q)

h2 = subplot(3,1,2);
plot(t,dq);

h3 = subplot(3,1,3);
plot(t,ddq);
end

%-----------------------------------------------
function make_other_plot(t,q,dq,ddq,h1,h2,h3)
%this function plots the speed and acceleration trajetory in joint space
hold(h1,'on');
subplot(3,1,1);
plot(t,q);
plot(t(1),q(1),'o');

hold(h2,'on');
subplot(3,1,2);
plot(t,dq);
plot(t(1),dq(1),'o');

hold(h3,'on');
subplot(3,1,3);
plot(t,ddq);
plot(t(1),ddq(1),'o');
```

```
end
```

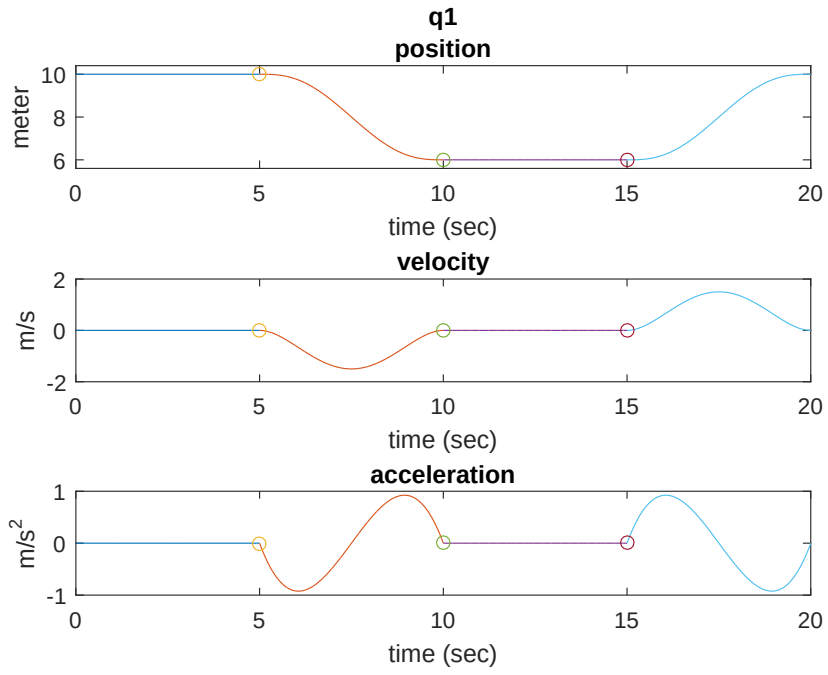


Figure 2.27: Problem 2, part 1, q_1 joint space trajectory

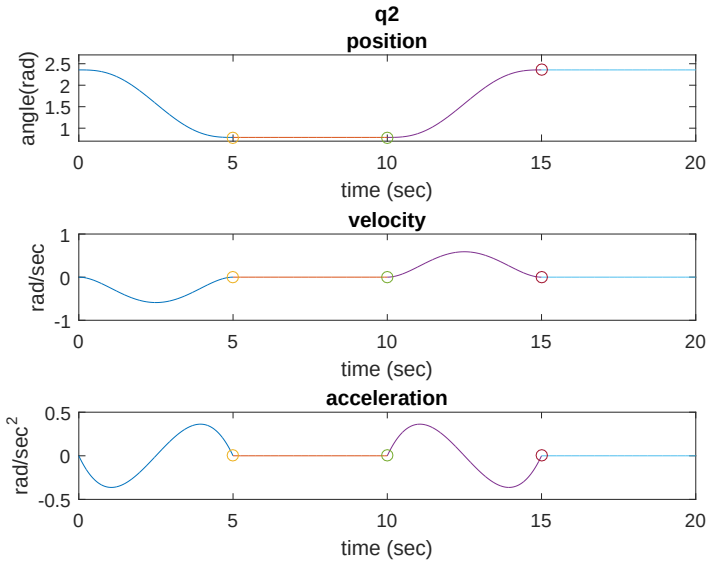


Figure 2.28: Problem 2, part 1, q_2 joint space trajectory

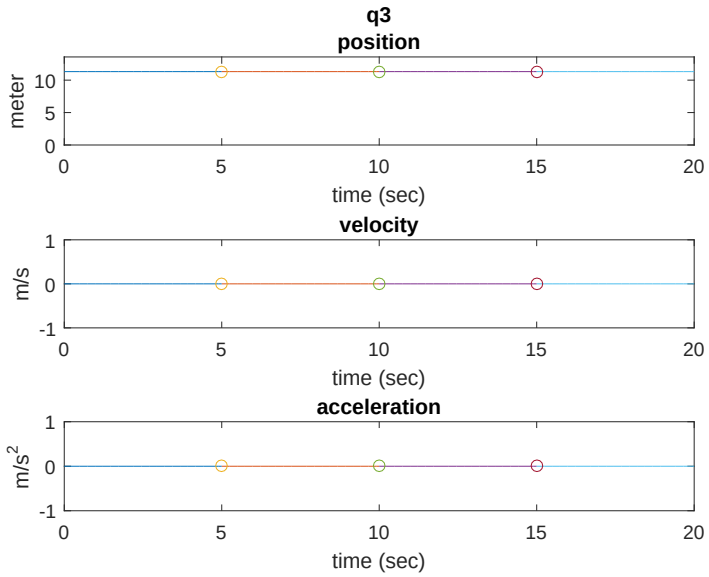


Figure 2.29: Problem 2, part 1, task space x_e, y_e, z_e trajectory

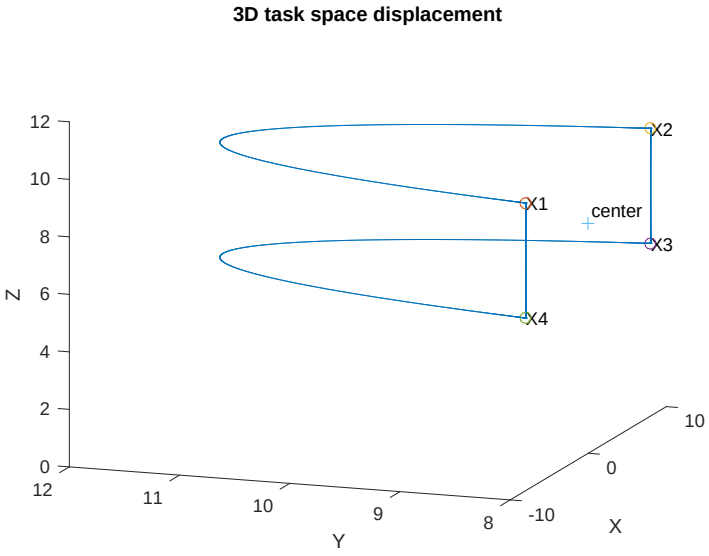


Figure 2.30: Problem 2, part 1, 3D plot of task space trajectory

Second scenario

The four part trajectory was generated again, using the same constraints as in first scenario, but now the polynomial was generated in task space as required for this part. The polynomial between waypoint X_1 and X_2 was found by solving 6 equations in 6 unknowns. The polynomial is defined for each coordinate x_e, y_e, z_e. For example, for x_e

$$x_e(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

The 6 equations are

$$\begin{aligned}
x_e(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 = 2.5L \\
\dot{x}_e(t_0) &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 = 0 \\
\ddot{x}_e(t_0) &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 = 0 \\
x_e(t_f) &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 = 2.5L \\
\dot{x}_e(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 = 0 \\
\ddot{x}_e(t_f) &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 = 0
\end{aligned}$$

Where $t_0 = 0$ and $t_f = 5$. The above 6 equations are solved for $a_0, a_1, a_2, a_3, a_4, a_5$ which gives the polynomial $x_e(t)$ used obtain the joint coordinate at any time $0 < t < 5$.

$$\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\
1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\
0 & 1 & 2t_f & 3t_f^2 & 4t_0^3 & 5t_f^4 \\
0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}
=
\begin{bmatrix} x_e(t_0) \\ \dot{x}_e(t_0) \\ \ddot{x}_e(t_0) \\ x_e(t_f) \\ \dot{x}_e(t_f) \\ \ddot{x}_e(t_f) \end{bmatrix}
=
\begin{bmatrix} 2.5L \\ 0 \\ 0 \\ 2.5L \\ 0 \\ 0 \end{bmatrix}$$

Similarly, these 6 equations were solved for the next segment between waypoint X_2, X_3. The same process was repeated for y_e and z_e.

The following gives the new Matlab implementation with the plots generated. Discussion on differences between the two approaches is given at the end.

```
function  nma_HW4_problem_2_part2()
%nma_HW4_problem_2_part2()
%This function implements problem 2, HW 4, ME 739, scenario TWO
%by Nasser M. Abbasi
%
close all;

L=4;
%generate the task space (X,Y,Z) displacements, velocity and acceleration
%the value of x,y,z  at each X1,X2,X3,X4 are given in the problem from
%the diagram shown.
```

```
%generate polynomial for x-coodinate of each X waypoint trajectory
figure;
[t1,x_coordinate]=process_X(-2*L,2*L,2*L,-2*L,'x',-2.6*L,2.6*L,...
    'meter','m/s','m/s^2');

%generate polynomial for y-coordinate of each X waypoint trajectory
figure;
[t2,y_coordinate]=process_X(2*L,2*L,2*L,2*L,...
    'y',1.9*L,2.1*L,'meter','m/sec','m/sec^2');

%generate polynomial for z-coordinate of each X waypoint trajectory
figure;
[t3,z_coordinate]=process_X(2.5*L,2.5*L,1.5*L,1.5*L,'z',1.4*L,2.6*L,...
    'meter','m/s','m/s^2');

%generate the corresponding joint space q displacement, speed and
%acceleration from the above using forward kinematics
q1 = z_coordinate;
q2 = atan2(y_coordinate,x_coordinate);
q3 = sqrt(x_coordinate.^2+y_coordinate.^2);

%generate joint space displacements
figure;
h1 = subplot(3,1,1);
plot(t1,q1);
title(h1,{'Joint space displacements','q1'});
xlabel(h1,'time (sec)');
ylabel(h1,'meter');
axis(h1,[0 20 0.9*L 2.8*L]);

h2 = subplot(3,1,2);
plot(t2,q2);
title(h2,'q2');
xlabel(h2,'time (sec)');
ylabel(h2,'radian');
axis(h2,[0 20 30*pi/180 140*pi/180]);

h3 = subplot(3,1,3);
plot(t3,q3);
title(h3,'q3');
xlabel(h3,'time (sec)');
ylabel(h3,'meter');
axis(h3,[0 20 1.9*L 2.9*L]);

%generate 3D plot of the task space trajectory
figure;
```

```
plot3(x_coordinate,y_coordinate,z_coordinate);
hold on;
plot3(-2*L,2*L,2.5*L,'o');
text(-2*L,2*L,2.5*L,'X1');

plot3(2*L,2*L,2.5*L,'o');
text(2*L,2*L,2.5*L,'X2');

plot3(2*L,2*L,1.5*L,'o');
text(2*L,2*L,1.5*L,'X3');

plot3(-2*L,2*L,1.5*L,'o');
text(-2*L,2*L,1.5*L,'X4');

plot3(0,2*L,2*L,'+');
text(.1*L,2*L,1.6*L,'center');

title('3D task space displacement');
xlabel('X'); ylabel('Y'); zlabel('Z');
zlim([0 3*L]);

end
%-----------------------
function [t,x]=process_X(x1,x2,x3,x4,the_name,lim0,lim1,y1,y2,y3)
%
%Function to generate task space trajectory for problem 2
%x1: first point coordinate in this task  space
%x2: second point coordinate in this task  space
%x3: third point coordinate in this task  space
%x4: fourth point coordinate in this task  space
%the_name: title to put on the plot, for example 'x' or 'y' or 'z'
%lim0 and lim1 are the y-axis limits to use for the plot
%y1: position y label
%y2: speed y label
%y3: acceleration y label

%RETURNS
%q: which is vector of q joint coordinates in joint space over
%the full time space to be used later to generate the task space
%trajectory using forward kinemetics
%
%t: the corresponding time vector

%call gen_path to obtain the polynomial coefficients
a              = gen_path(x1,x2,0,5);
[x_segment_1,dx,ddx,t1] = find_x(0,5,a); %use the coefficients to make polynomials
[h1,h2,h3]     = make_first_plot(t1,x_segment_1,dx,ddx);
```

```
title(h1,{the_name,'position'});
title(h2,'velocity');
title(h3,'acceleration');
xlabel(h1,'time (sec)'); ylabel(h1,y1);
xlabel(h2,'time (sec)'); ylabel(h2,y2);
xlabel(h3,'time (sec)'); ylabel(h3,y3);

a              = gen_path(x2,x3,5,10);
[x_segment_2,dx,ddx,t2] = find_x(5,10,a);
make_other_plot(t2,x_segment_2,dx,ddx,h1,h2,h3);


a              = gen_path(x3,x4,10,15);
[x_segment_3,dx,ddx,t3] = find_x(10,15,a);
make_other_plot(t3,x_segment_3,dx,ddx,h1,h2,h3);

a              = gen_path(x4,x1,15,20);
[x_segment_4,dx,ddx,t4] = find_x(15,20,a);
make_other_plot(t4,x_segment_4,dx,ddx,h1,h2,h3);

axis(h1,[0 20 lim0 lim1]);
x=[x_segment_1 x_segment_2 x_segment_3 x_segment_4];
t=[t1 t2 t3 t4];
end

%---------------------------------------------
function a = gen_path(x0,xf,t0,tf)
%this function generates the polynomial coefficients by solving for the
%constraints given

dx0  = 0;
dxf  = 0;
ddx0 = 0;
ddxf = 0;

C = [ 1  t0  t0^2  t0^3    t0^4     t0^5;
      0  1   2*t0  3*t0^2  4*t0^3   5*t0^4;
      0  0   2     6*t0    12*t0^2  20*t0^3;
      1  tf  tf^2  tf^3    tf^4     tf^5;
      0  1   2*tf  3*tf^2  4*tf^3   5*tf^4;
      0  0   2     6*tf    12*tf^2  20*tf^3];

x  = [x0 dx0 ddx0 xf dxf ddxf]';
a  = C\x;

end
```

```
%-------------------------------------------------
function [x,dx,ddx,t]=find_x(t0,tf,a)
%This function takes the coefficients of the polynomial and
%return back the polynomials for position, speed and acceleration
a0  = a(1); a1 = a(2); a2 = a(3); a3 = a(4); a4 = a(5); a5 = a(6);
t   = linspace(t0,tf,1000);
x   = a0   + a1*t   + a2*t.^2    + a3*t.^3    + a4*t.^4     + a5*t.^5;
dx  = a1   + 2*a2*t + 3*a3*t.^2  + 4*a4*t.^3  + 5*a5*t.^4;
ddx = 2*a2 + 6*a3*t + 12*a4*t.^2 + 20*a5*t.^3;

end

%-----------------------------------------------------
function [h1,h2,h3]=make_first_plot(t,x,dx,ddx)
%this function plots the position in task space
h1 = subplot(3,1,1);
plot(t,x)

h2 = subplot(3,1,2);
plot(t,dx);

h3 = subplot(3,1,3);
plot(t,ddx);
end

%-------------------------------------------------
function make_other_plot(t,x,dx,ddx,h1,h2,h3)
%this function plots the speed and acceleration trajetory in task space
hold(h1,'on');
subplot(3,1,1);
plot(t,x);
plot(t(1),x(1),'o');

hold(h2,'on');
subplot(3,1,2);
plot(t,dx);
plot(t(1),dx(1),'o');

hold(h3,'on');
subplot(3,1,3);
plot(t,ddx);
plot(t(1),ddx(1),'o');
end
```

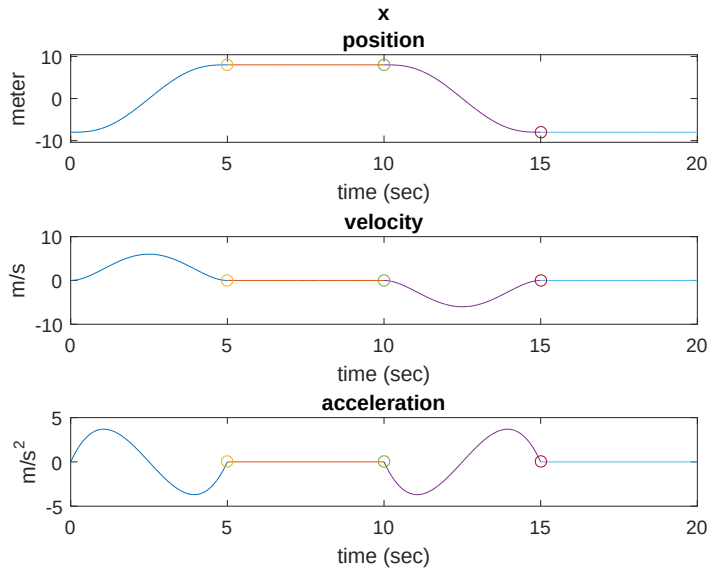


Figure 2.31: Problem 2, part 2, x_e task space trajectory

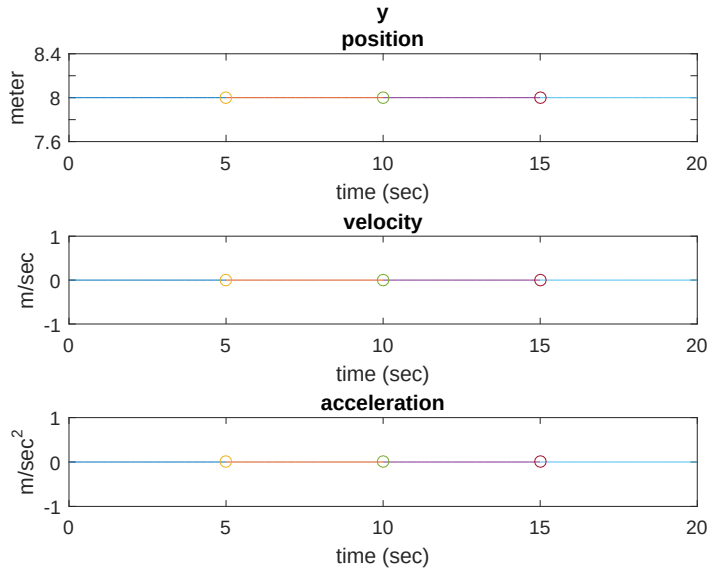


Figure 2.32: Problem 2, part 2, y_e task space trajectory

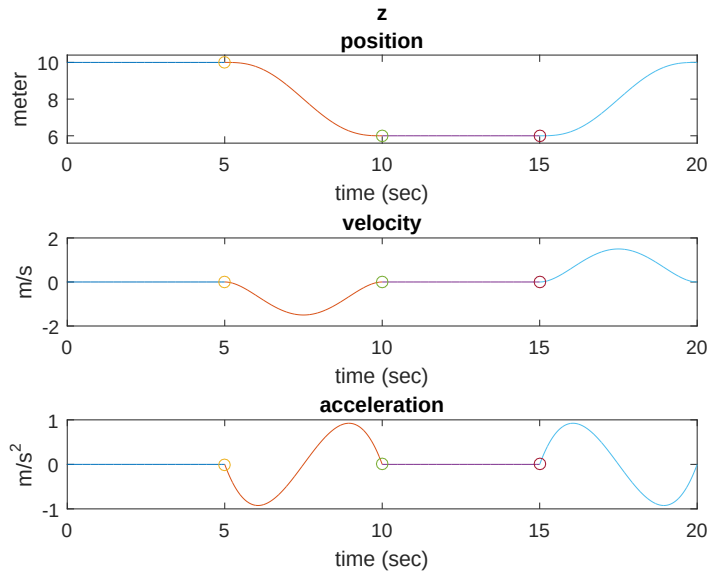


Figure 2.33: Problem 2, part 2, z_e task space trajectory

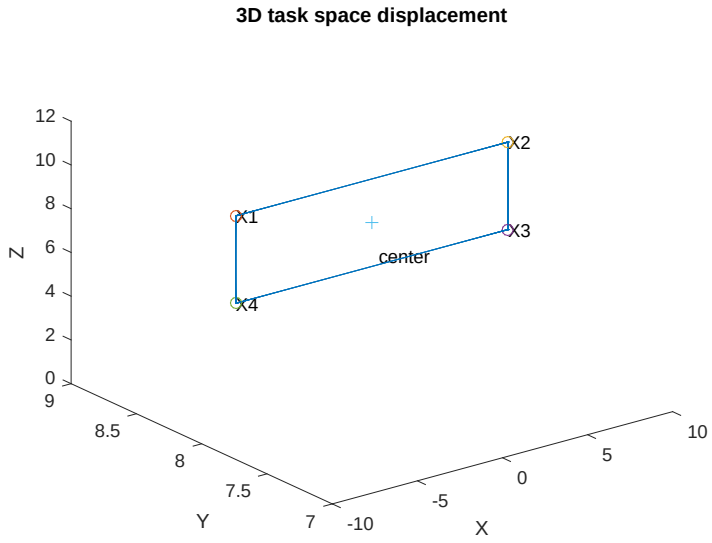


Figure 2.34: Problem 2, part 2, 3D plot of task space trajectory

discussion

The following diagram gives an overview of the difference of the algorithm used for first and second scenarios.

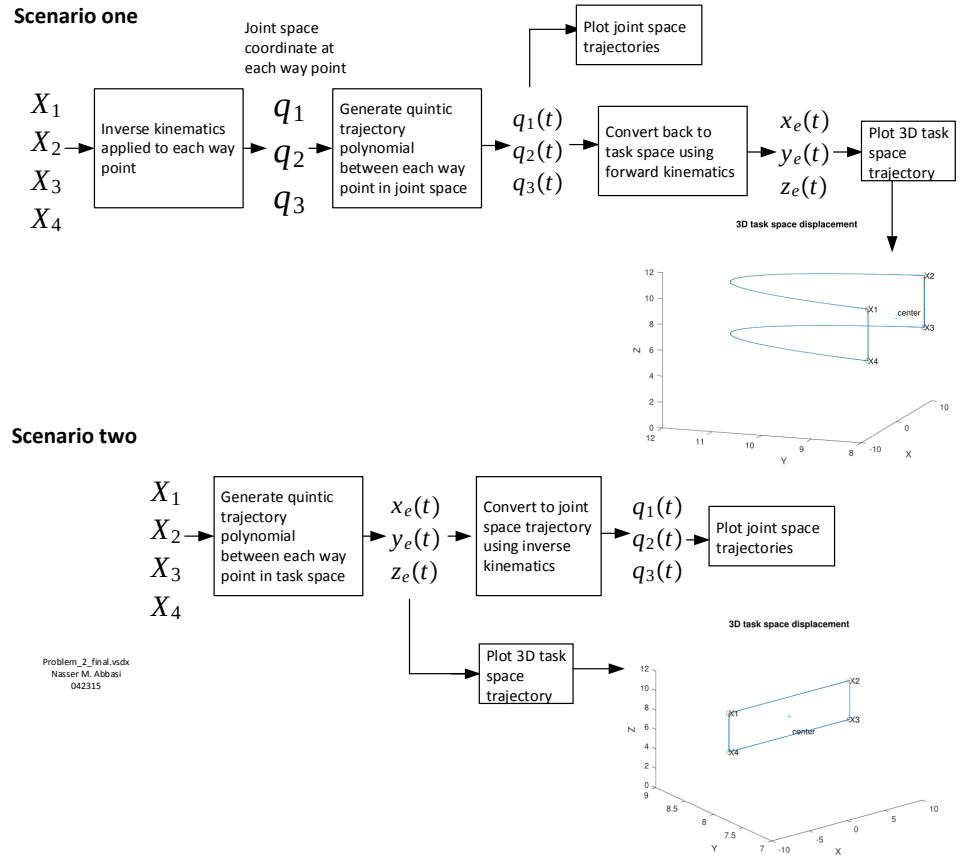


Figure 2.35: Problem 2 algorithm difference scenario one and two

There are two main differences observed between the two methods.

1. In the first scenario, joint space trajectories $q_1(t)$ and $q_2(t)$ were the same as in the second scenario, but $q_3(t)$ was not the same.

   In first scenario, $q_3(t)$ was constant with value $2\sqrt{2}\,L$ for the whole path. This is becuase at each way point q_3 had the same value $2\sqrt{2}\,L$, therefore the polynomial generated was straight line connecting all the way points. In second scenario, $q_3(t)$ was not constant, since $q_3(t) = \sqrt{x_e(t)^2 + y_e(t)^2}$ and $x_e(t)$ polynomial was not constant (even though $y_e(t)$ was constant over the whole path).

2. In the first scenario, $y_e(t)$ was generated by inverse kinematics using $y_e(t) = q_3(t)\sin(q_2(t))$ and even though $q_3(t)$ was constant, $q_2(t)$ was not. Hence the path along the y direction is task space was changing as can be seen from the above plot. In the second scenario, since the y was fixed at each way point, the trajectory generated in task space shows that $y(t)$ is not changing. This is why the 3D plot for the second scenario do not show the same curved path in the y dimension as in the first scenario.

The result from first scenario seems to be more realistic 3D task space path that the robot arm end effector would take. Computationally, there is little difference between the two scenarios.

2.4.3 Problem 3

Problem 3. [40 points]

For the manipulator described in Problem 2

- Use a parabolic well potential to define an attractive field applied to the origin of frame {e}. Use Matlab to implement a gradient descent algorithm to find a path from the specified initial configuration to the specified final configuration given below.

  Initial configuration: $q_0 = \begin{bmatrix} L & 0 & L \end{bmatrix}^T$

  Final configuration: $q_f = \begin{bmatrix} 2L & \frac{1}{2}\pi & 2L \end{bmatrix}^T$

- Animate your results using the Matlab rendering functions posted to the Learn@UW course page.

In general, you can vary ζ_i (the scaling factor applied to the attractive force, f_i) to modify the resulting path. In this case, the forces are applied at only one point so there is only one value of ζ. You can also vary the gradient descent scaling factor, α. Larger values of α can speed up the gradient descent algorithm, but can also cause numerical instabilities in the solution. In addition, you can use different values of α (α_i for joint i) to scale the joint torques and thus modify the path solution.

Gradient descent:

$$\begin{bmatrix} \text{while } \left\| q^k - q_f \right\| > \varepsilon \\ \quad q^{k+1} = q^k + \alpha \nabla U_\tau \quad \text{where } \nabla U_\tau = \dfrac{\tau(q^k)}{\left\| \tau(q^k) \right\|} \text{ and } \alpha = \begin{bmatrix} \alpha_1 & & 0 \\ & \ddots & \\ 0 & & \alpha_n \end{bmatrix} \\ \quad k = k+1 \end{bmatrix}$$

- Vary α_i and comment on the resulting solution behavior (both in regards to the shape of the path and the numerical stability of the solution.

Figure 2.36: Problem 3 description

In this problem we need to determine only the attractive virtual force on origin of end effector frame {e}. The repulsive forces are not involved. The attractive virtual forces are approximated with a parabolic well potential. We have only one location where the force is applied to (which is the origin of frame {e}, which is the end effector). The first step is to determine the virtual force from the gradient of the parabolic potential field given by

$$\begin{aligned} \mathbf{F}_e &= -\nabla U_e \\ &= -\zeta \left(\mathbf{o}_e \left(q_{current} \right) - \mathbf{o}_e \left(q_f \right) \right) \end{aligned}$$

Where in the above $\mathbf{o}_e(q)$ is the position vector of the origin of frame {e} at current configuration q and $\mathbf{o}_e(q_f)$ is position vector of the origin of frame {e} at final configuration q_f. Hence $\mathbf{o}_e(q_f)$ is constant position vector and only $\mathbf{o}_e(q)$ will change as the robot arm moves.

We are given the final configuration as $q_f = \left[2L, \frac{\pi}{2}, 2L \right]^T$, and we now use this to obtain $\mathbf{o}_e(q_f)$. We first need to obtain T_e^0 which we obtained forward kinematics given in problem 2. Hence

$$
\begin{aligned}
T_e^0 &= T_1^0 T_2^1 T_e^2 \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\sin q_2 & 0 & \cos q_2 & 0 \\ \cos q_2 & 0 & \sin q_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & q_3 \cos q_2 \\ \sin q_2 & \cos q_2 & 0 & q_3 \sin q_2 \\ 0 & 0 & 1 & q_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Therefore, $\mathbf{o}_e(q)$ from the above is the last column. Hence

$$
\mathbf{o}_e(q_{current}) = \begin{bmatrix} q_3 \cos q_2 \\ q_3 \sin q_2 \\ q_1 \end{bmatrix}
$$

We now replace the values of q_1, q_2, q_3 in the above, with the final configuration values given

$$
\mathbf{o}_e(q_{final}) = \begin{bmatrix} 2L \cos \frac{\pi}{2} \\ 2L \sin \frac{\pi}{2} \\ 2L \end{bmatrix} = \begin{bmatrix} 0 \\ 2L \\ 2L \end{bmatrix}
$$

And the initial configuration (at time $t = 0$) is

$$
\mathbf{o}_e(q_{initial}) = \begin{bmatrix} L \cos 0 \\ L \sin 0 \\ L \end{bmatrix} = \begin{bmatrix} L \\ 0 \\ L \end{bmatrix}
$$

Using the above, we need determine the virtual force F_e

$$
\begin{aligned}
\mathbf{F}_e &= -\nabla U_e \\
&= -\zeta_e \left(\mathbf{o}_e(q_{current}) - \mathbf{o}_e(q_f) \right) \\
&= -\zeta_e \left(\begin{bmatrix} q_3 \cos q_2 \\ q_3 \sin q_2 \\ q_1 \end{bmatrix} - \begin{bmatrix} 0 \\ 2L \\ 2L \end{bmatrix} \right) \\
&= -\zeta_e \begin{bmatrix} q_3 \cos(q_2) \\ q_3 \sin q_2 - 2L \\ q_1 - 2L \end{bmatrix}
\end{aligned}
$$

In the above, all the q_i variables are the current joint variables at the current time. In simulation, there will change with time. Now that we found the virtual force, we convert it to joint toque using the linear velocity Jacobian. We obtain J_{v_e} by direct differentiation method

$$J_{v_e} = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} \end{bmatrix}$$

From problem 2, we are given that

$$\begin{aligned} x_e &= q_3 \cos q_2 \\ y_e &= q_3 \sin q_2 \\ z_e &= q_1 \end{aligned}$$

Hence J_{v_e} becomes

$$J_{v_e} = \begin{bmatrix} \partial \frac{q_3 \cos q_2}{\partial q_1} & \partial \frac{q_3 \cos q_2}{\partial q_2} & \partial \frac{q_3 \cos q_2}{\partial q_3} \\ \partial \frac{q_3 \sin q_2}{\partial q_1} & \partial \frac{q_3 \sin q_2}{\partial q_2} & \partial \frac{q_3 \sin q_2}{\partial q_3} \\ \frac{\partial q_1}{\partial q_1} & \frac{\partial q_1}{\partial q_2} & \frac{\partial q_1}{\partial q_3} \end{bmatrix} = \begin{bmatrix} 0 & -q_3 \sin q_2 & \cos q_2 \\ 0 & q_3 \cos q_2 & \sin q_2 \\ 1 & 0 & 0 \end{bmatrix}$$

Therefore, using the duality relation that $\tau = \sum J_{v_i}^T F_i$ and since we have forces on one joint only, this simplifies to $\tau = J_{v_e}^T F_e$

$$\begin{aligned} \boldsymbol{\phi} &= -\zeta_e \begin{bmatrix} 0 & -q_3 \sin q_2 & \cos q_2 \\ 0 & q_3 \cos q_2 & \sin q_2 \\ 1 & 0 & 0 \end{bmatrix}^T \begin{bmatrix} q_3 \cos (q_2) \\ q_3 \sin q_2 - 2L \\ q_1 - 2L \end{bmatrix} \\ &= -\zeta_e \begin{bmatrix} q_1 - 2L \\ q_3 (\cos q_2) (q_3 \sin q_2 - 2L) - q_3^2 \cos q_2 \sin q_2 \\ q_3 \cos^2 q_2 + (\sin q_2) (q_3 \sin q_2 - 2L) \end{bmatrix} \end{aligned} \tag{1}$$

The above is the actual torque/force applied at the joints 1,2 and 3. Therefore, the force applied to joint 1 is $q_1 - 2L$ and the torque to apply to joint 2 is $q_3 (\cos q_2) (q_3 \sin q_2 - 2L) - q_3^2 \cos q_2 \sin q_2$ and the force to apply to end effector joint is $q_3 \cos^2 q_2 + (\sin q_2) (q_3 \sin q_2 - 2L)$.

For example, at initial configuration, where $q_{initial} = [L, 0, L]^T$ we find

$$\boldsymbol{\phi}_{initial} = -\zeta_e \begin{bmatrix} -L \\ -2Lq_3 \\ L \end{bmatrix}$$

And at final configuration where $q_{final} = q_f = \left[2L, \frac{\pi}{2}, 2L\right]^T$ we find

$$\boldsymbol{\phi}_{initial} = -\zeta_e \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Which is what we would expect. At the final configuration, the applied forces at the joints should vanish since we have arrived at the final configuration.

Now that we have equation (1), we can use it to implement the gradient descent algorithm in order to obtain the sequence of q_i positions to use for the animation.

Part one

In this part, a single α value was used for all the joints. This value was changed in order to observe the affect. In second part below, different α for each joint will be used.

In this part, and in all runs, ζ was set to 100. This is the attractive force scaling value.

Below is the source code written to implement this part of the problem. The following diagram shows the initial configuration for one example run.

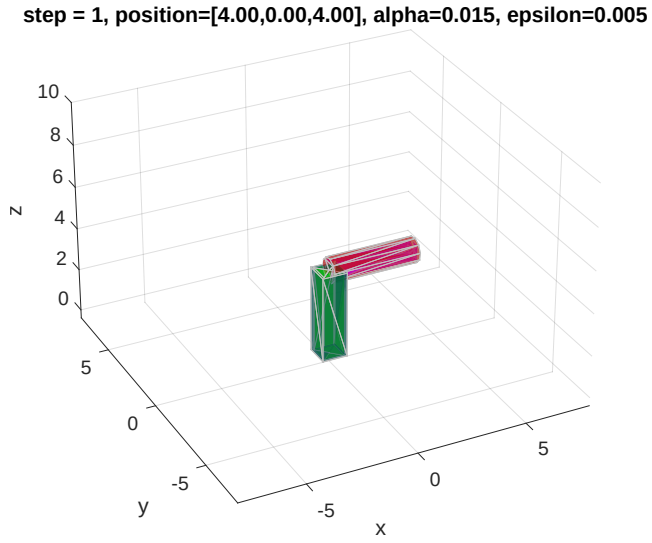


Figure 2.37: Initial configuration, problem 3

And the following diagram shows the final configuration reached. It shows that it took 535 steps to reach the final configuration using $\alpha = 0.015$ and $\epsilon = 0.005$

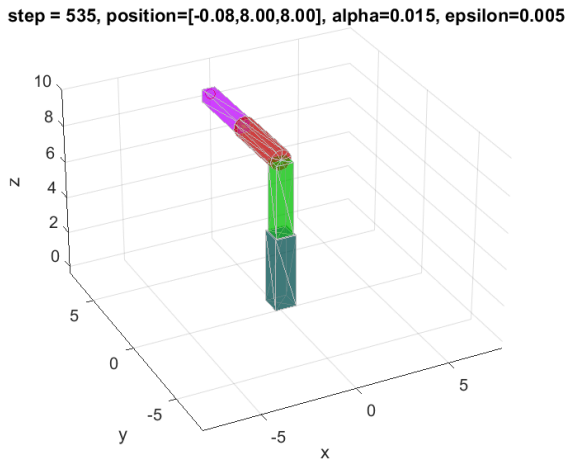


Figure 2.38: Final configuration, problem 3

As α was made smaller the convergence became slower, but the animation became more accurate and ran more smoothly. As the end effector came very close to the target, more vibration around that region started to show as the end effector oscillated around the target point as it converged to the exact target and the error became smaller and smaller.

To run the program, the command is `nma_HW4_problem_3`

```
function  nma_HW4_problem_3()
%function nma_HW4_problem_3()
%This function implements path planning for the 3 links robot arm in
%HW4, ME 739, problem 3, spring 2015, Univ. Wiscosin, Madison.
%
%This version is for part 1, which uses one alpha for all joints
%
%Parabolic attractive field is used to model the attractive virtual
%force on the end effector. The linear velocity Jacobian is used to
%map this force to torque forces at the three joints.
%Then gradient descent is used to obtain the joint coordinates sequence
%which is then used to simulate the motion.
%
%By Nasser M. Abbasi

close all; clear all;
L        = 4;  %length of link 1 and 3
%epsilon  = 0.005; %error tolerance
%alpha    = 0.015; %smaller value, slows down convergence but more accurate

epsilon  = 0.05; %error tolerance
zeta     = 100;  %attractive force scaling
```

```
alpha    = 0.05; %smaller value, slows down convergence but more accurate

q        = [L;0;L];  %initial joint configuration
qf       = [2*L;pi/2;2*L]; %find joint configuration
maxIter  = 1200;  %max iterations allowed

Q            = zeros(3,maxIter);  %where to save the sequence of joint q's
k            = 1;
keep_running = true;

%start of gradient descemt

while keep_running
    Q(:,k) = q;
    tau = -zeta*[q(1)-2*L;
        q(3)*cos(q(2))*(q(3)*sin(q(2))-2*L)-q(3)^2*cos(q(2))*sin(q(2));
        q(3)*(cos(q(2)))^2+sin(q(2))*(q(3)*sin(q(2))-2*L)
        ];
    q      = q + alpha*tau/norm(tau);
    if k+1>maxIter || norm( q-qf ) <= epsilon
        keep_running = false;
    else
        k = k +1;
    end
end

DO_MOVIE = false; %make true to generate movie frames
frameNumber = 0;

f_handle    = 1;
axis_limits = L*[-2 2 -2 2 -0.1 2.5];
render_view = [-.4 -.8 .5]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective % turns on perspective

% link 0 rendering initialization
r0 = L/5; sides0 = 4; axis0 = 3; norm_L0 = -1.0;
linkColor0 = [0 .3 .3]; plotFrame0 = 0;
d0 =  CreateLinkRendering(L,r0,sides0,axis0,norm_L0,linkColor0,...
    plotFrame0,f_handle);


% link 1 rendering initialization
r1 = L/6; sides1 = 4; axis1 = 3; norm_L1 = 1.0;
linkColor1 = [0 0.75 0]; plotFrame1 = 0;
d1 =  CreateLinkRendering(L,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);
```

```
% link 2 rendering initialization
r2 = L/7; sides2 = 10; axis2 = 3; norm_L2 = -1.0;
linkColor2 = [0.75 0 0]; plotFrame2 = 0;
d2 =  CreateLinkRendering(L,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

% link 3 rendering initialization
r3 = L/8; sides3 = 4; axis3 = 1; norm_L3 = 1.0;
linkColor2 = [0.75 0 1]; plotFrame2 = 0;
d3 =  CreateLinkRendering(L,r3,sides3,axis3,norm_L3,linkColor2,...
    plotFrame2,f_handle);

T00 = [1  0  0  0
    0  1  0  0
    0  0  1  0
    0  0   0  1];
UpdateLink(d1,T00);

for i = 1:k
    % Update frames
    q1=Q(1,i); q2=Q(2,i); q3=Q(3,i);
    T01 = [1  0  0  0
           0  1  0  0
           0  0  1  q1
           0  0   0  1];

    T12 = [-sin(q2)  0   cos(q2)   0
           cos(q2)   0   sin(q2)   0
           0         1   0         0
           0         0   0         1];

    T2e = [0   1  0  0
           0   0  1  0
           1   0  0  q3
           0   0  0  1];

    T02 = T01*T12;
    T0e = T02*T2e;

    UpdateLink(d1,T01);
    UpdateLink(d2,T02);
    UpdateLink(d3,T0e);

    title(sprintf('step = %d, position=[%3.2f,%3.2f,%3.2f], alpha=%3.3f, epsilon=%3.3f',i,...
        q3*cos(q2),q3*sin(q2),q1,alpha,epsilon));
    %if i == 1;  %pause at start of simulation rendering
```

```
    %    pause;
    %end

    if DO_MOVIE
        frameNumber = frameNumber+1;
        I = getframe(gcf);
        imwrite(I.cdata, sprintf('frame%d.png',frameNumber));
    end
    hold on;
    plot3(0,2*L,2*L,'marker','o','color','r');
    drawnow;
    hold off;
    pause(0.01);
end
end
```

Part two

In this part, different α_i values were used for each joint. The code to implement this part is similar to the above with the only change is in using a diagonal matrix for α instead of justone scalar value. Different α_i was used for each joint to see the effect on the path of the end effector.

The following vector of values gave the best solution

```
alpha    = diag([0.05,0.01,0.1]);
```

The above produced the least amount of oscillation as the end effector came close to the target. Making α_i smaller caused the joint i to move the slowest during the animation.

The following value of vector α caused large oscillation as the end effector came close to the target.

```
alpha    = diag([0.05,0.1,0.01]);
```

These values of α are kept in the code below to allow one to try them. The source code for this part is listed below. Using different α_i value for each joint is more flexible and allowed finding better solution than using the same *alpha* for all joints. The command to run the Matlab script is `nma_HW4_problem_3_part2`

```
function  nma_HW4_problem_3_part2()
%function nma_HW4_problem_3_part2()
%This function implements path planning for the 3 links robot arm in
%HW4, ME 739, problem 3, spring 2015, Univ. Wiscosin, Madison.
%
%This version is for part 2, which uses different alpha for each joint
%
```

```
%Parabolic attractive field is used to model the attractive virtual
%force on the end effector. The linear velocity Jacobian is used to
%map this force to torque forces at the three joints.
%Then gradient descent is used to obtain the joint coordinates sequence
%which is then used to simulate the motion.
%
%By Nasser M. Abbasi

close all; clear all;
L        = 4;  %length of link 1 and 3
%epsilon  = 0.005; %error tolerance
%alpha    = 0.015; %smaller value, slows down convergence but more accurate

epsilon  = 0.01; %error tolerance
zeta     = 100;  %attractive force scaling
%alpha    = diag([0.05,0.1,0.01]); %Causes large errors in solution
alpha    = diag([0.05,0.01,0.1]);  %Produces best solution

q        = [L;0;L];  %initial joint configuration
qf       = [2*L;pi/2;2*L]; %find joint configuration
maxIter  = 1000;  %max iterations allowed

Q            = zeros(3,maxIter);  %where to save the sequence of joint q's
k            = 1;
keep_running = true;

%start of gradient descemt

while keep_running
    Q(:,k) = q;
    tau = -zeta*[q(1)-2*L;
        q(3)*cos(q(2))*(q(3)*sin(q(2))-2*L)-q(3)^2*cos(q(2))*sin(q(2));
        q(3)*(cos(q(2)))^2+sin(q(2))*(q(3)*sin(q(2))-2*L)
        ];
    q      = q + alpha*tau/norm(tau);
    if k+1>maxIter || norm( q-qf ) <= epsilon
        keep_running = false;
    else
        k = k +1;
    end
end

DO_MOVIE = false; %make true to generate movie frames
frameNumber = 0;

f_handle    = 1;
axis_limits = L*[-2 2 -2 2 -0.1 2.5];
```

```
render_view = [-.4 -.8 .5]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective % turns on perspective

% link 0 rendering initialization
r0 = L/5; sides0 = 4; axis0 = 3; norm_L0 = -1.0;
linkColor0 = [0 .3 .3]; plotFrame0 = 0;
d0 =  CreateLinkRendering(L,r0,sides0,axis0,norm_L0,linkColor0,...
    plotFrame0,f_handle);


% link 1 rendering initialization
r1 = L/6; sides1 = 4; axis1 = 3; norm_L1 = 1.0;
linkColor1 = [0 0.75 0]; plotFrame1 = 0;
d1 =  CreateLinkRendering(L,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);

% link 2 rendering initialization
r2 = L/7; sides2 = 10; axis2 = 3; norm_L2 = -1.0;
linkColor2 = [0.75 0 0]; plotFrame2 = 0;
d2 =  CreateLinkRendering(L,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

% link 3 rendering initialization
r3 = L/8; sides3 = 4; axis3 = 1; norm_L3 = 1.0;
linkColor2 = [0.75 0 1]; plotFrame2 = 0;
d3 =  CreateLinkRendering(L,r3,sides3,axis3,norm_L3,linkColor2,...
    plotFrame2,f_handle);

T00 = [1  0  0  0
    0  1  0  0
    0  0  1  0
    0  0   0  1];
UpdateLink(d1,T00);

for i = 1:k
    % Update frames
    q1=Q(1,i); q2=Q(2,i); q3=Q(3,i);
    T01 = [1  0  0  0
           0  1  0  0
           0  0  1  q1
           0  0   0  1];

    T12 = [-sin(q2)  0   cos(q2)   0
           cos(q2)   0   sin(q2)   0
           0         1   0         0
           0         0   0         1];
```

```
    T2e = [0   1  0  0
           0   0  1  0
           1   0  0  q3
           0   0  0  1];

    T02 = T01*T12;
    T0e = T02*T2e;

    UpdateLink(d1,T01);
    UpdateLink(d2,T02);
    UpdateLink(d3,T0e);

    title(sprintf('step = %d, position=[%3.2f,%3.2f,%3.2f], epsilon=%3.3f',i,...
        q3*cos(q2),q3*sin(q2),q1,epsilon));
    %if i == 1;  %pause at start of simulation rendering
    %    pause;
    %end

    if DO_MOVIE
        frameNumber = frameNumber+1;
        I = getframe(gcf);
        imwrite(I.cdata, sprintf('frame%d.png',frameNumber));
    end
    hold on;
    plot3(0,2*L,2*L,'marker','o','color','r');
    drawnow;
    hold off;
    pause(0.01);
end
end
```

2.4.4 key solution for HW 4

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

Please submit your answers to the questions ... work including your Matlab scripts, and, where appropriate, program results (pl... Your Matlab scripts should be readable, with comments, sensible variable names, ...ode-block, etc. In addition to the hardcopy (pdf format), you must also submit your M... ...pts electronically to the Learn@UW course page dropbox (e.g. Homework #4) using a zip archive ... format. Please name your zip files using your last name and hw# (e.g. zinn_hw4.zip).

Problem 1 [20 points]

► Write a Matlab function which constructs a quintic polynomial, for the purposes of trajectory generation, given the following input parameters

t_0:	Initial time	t_f:	Final time
y_0:	Initial position	y_f:	Final position
$\dot{y}_0$:	Initial velocity	$\dot{y}_f$:	Final velocity
$\ddot{y}_0$:	Initial acceleration	$\ddot{y}_f$:	Final acceleration

An example function prototype is shown below

$$[t,\ y,\ \dot{y},\ \ddot{y}] = \texttt{QuinticPolynomial}(t_0,\ t_f,\ y_0,\ y_f,\ \dot{y}_0,\ \dot{y}_f,\ \ddot{y}_0,\ \ddot{y}_f,\ n)$$

where n is the number of elements in the time vector and output arrays ($[y,\ \dot{y},\ \ddot{y}]$)

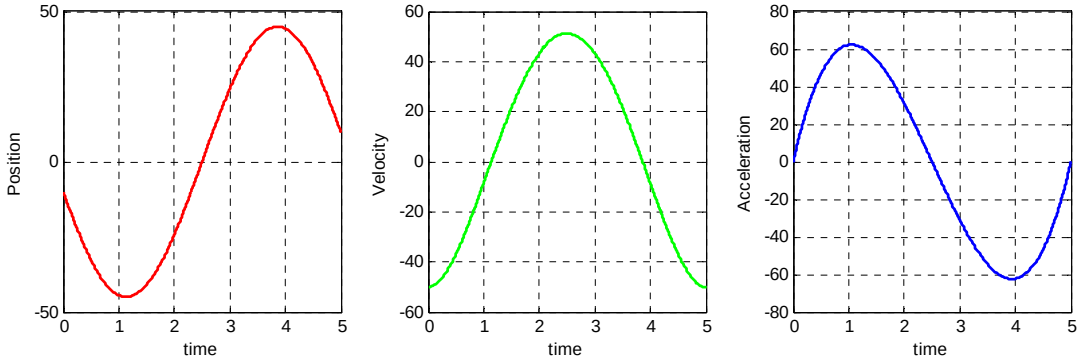
- t : n-dimensional array of trajectory time
- y : n-dimensional array of trajectory position
- $\dot{y}$: n-dimensional array of trajectory velocity
- $\ddot{y}$: n-dimensional array of trajectory acceleration

► To check your function, create and plot the trajectory for the input parameters given below:

t_0:	0	t_f:	5
y_0:	-10	y_f:	10
$\dot{y}_0$:	-50	$\dot{y}_f$:	-50
$\ddot{y}_0$:	0	$\ddot{y}_f$:	0

e.g.
```
[t, y, dy, ddy] = QuinticPolynomial(0,5,-10,10,-50,-50,0,0,1000);
figure; plot(t,y);
figure; plot(t, dy);
figure; plot(t, ddy);
```

Your output plots should like those shown below.



ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

Test of quintic polynomial function:

```
% number of elements in the time vector and output arrays
n = 1000;

% input parameters
t0   =   0;    tf   =   5;
y0   = -10;    yf   =  10;
dy0  = -50;    dyf  = -50;
ddy0 =   0;    ddyf =   0;

% test use of function
[t, y, dy, ddy] =
QuinticPolynomial(t0,tf,y0,yf,dy0,dyf,ddy0,ddyf,n);

% plot results
subplot(1,3,1);
plot(t, y,'r','LineWidth',2)
xlabel('time')
ylabel('Position');
grid on;

subplot(1,3,2);
plot(t, dy,'g','LineWidth',2)
xlabel('time')
ylabel('Velocity');
grid on;

subplot(1,3,3);
plot(t, ddy,'b','LineWidth',2)
xlabel('time')
ylabel('Acceleration');
grid on;
```

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

QuinticPolynomial.m:

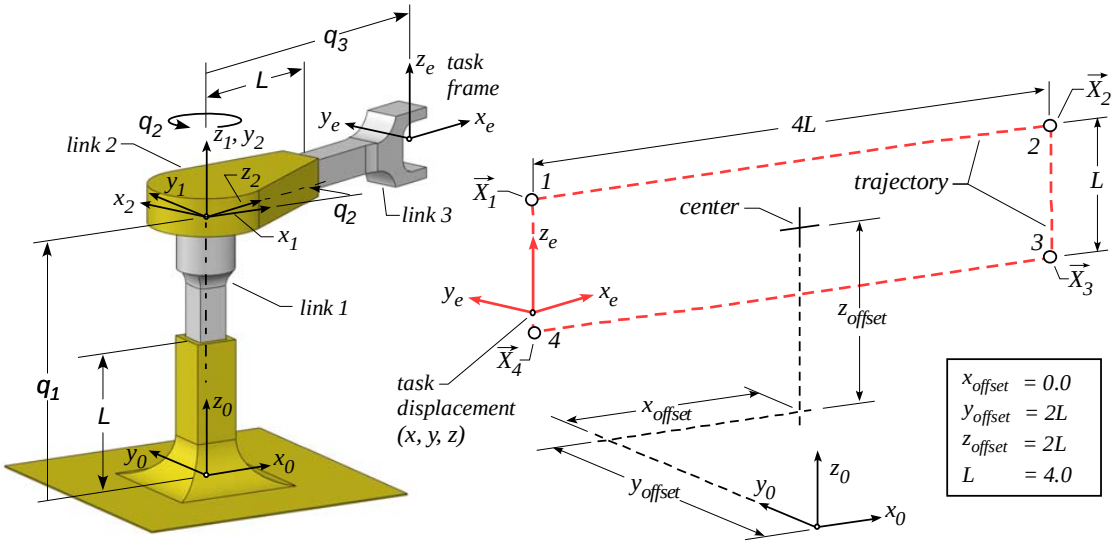
```
function [t, y, dy, ddy] = QuinticPolynomial(t0, tf, y0, ...
                                yf, dy0, dyf, ddy0, ddyf, n)

% coefficient matrix
C = [1    t0      t0^2     t0^3    t0^4      t0^5
     0     1    2*t0     3*t0^2   4*t0^3    5*t0^4
     0     0      2      6*t0    12*t0^2   20*t0^3
     1    tf      tf^2     tf^3     tf^4      tf^5
     0     1    2*tf     3*tf^2   4*tf^3    5*tf^4
     0     0      2      6*tf    12*tf^2   20*tf^3];

% specified positions and velocities
ystates = [y0 dy0 ddy0 yf dyf ddyf]';

% calculate quintic polynomial coefficients
a = C\ystates;

% construct time vector
t = linspace(t0,tf,n)';

% assisgn coefficient values
a0 = a(1); a1 = a(2); a2 = a(3);
a3 = a(4); a4 = a(5); a5 = a(6);

% construct output arrays
y   =   a0 +   a1*t +    a2*t.^2  +  a3*t.^3 +  ...
                         a4*t.^4  +  a5*t.^5;
dy  =   a1 + 2*a2*t +  3*a3*t.^2 +  4*a4*t.^3 + ...
                                    5*a5*t.^4;
ddy = 2*a2 + 6*a3*t + 12*a4*t.^2 + 20*a5*t.^3;
```

ME/ECE 739: Advanced Robotics

Homework #4

Due: April 26th (Sunday, 11:59 pm)

Problem 2. [40 points]



Overview of 3-DOF manipulator **Task-space trajectory (rectangular)**

You are to generate the motion trajectory for the three degree of freedom manipulator shown in the above figure such that the manipulator's task frame (origin) moves between the four points that describe a square in space (see Figure). The forward and inverse kinematics are given below.

Forward kinematics:	*Inverse kinematics:*
$x_e = q_3 \cos(q_2)$	$q_1 = z_e$
$y_e = q_3 \sin(q_2)$	$q_2 = \tan^{-1}(y_e / x_e)$
$z_e = q_1$	$q_3 = \sqrt{x_e^2 + y_e^2}$

For reference, the homogeneous transformations between the successive link frames (defined in the figure above) are given below. These may be useful when animating your results in Problem 3.

$$T_1^0 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]; \quad T_2^1 = \left[\begin{array}{ccc|c} -s_2 & 0 & c_2 & 0 \\ c_2 & 0 & s_2 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]; \quad T_e^2 = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & q_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

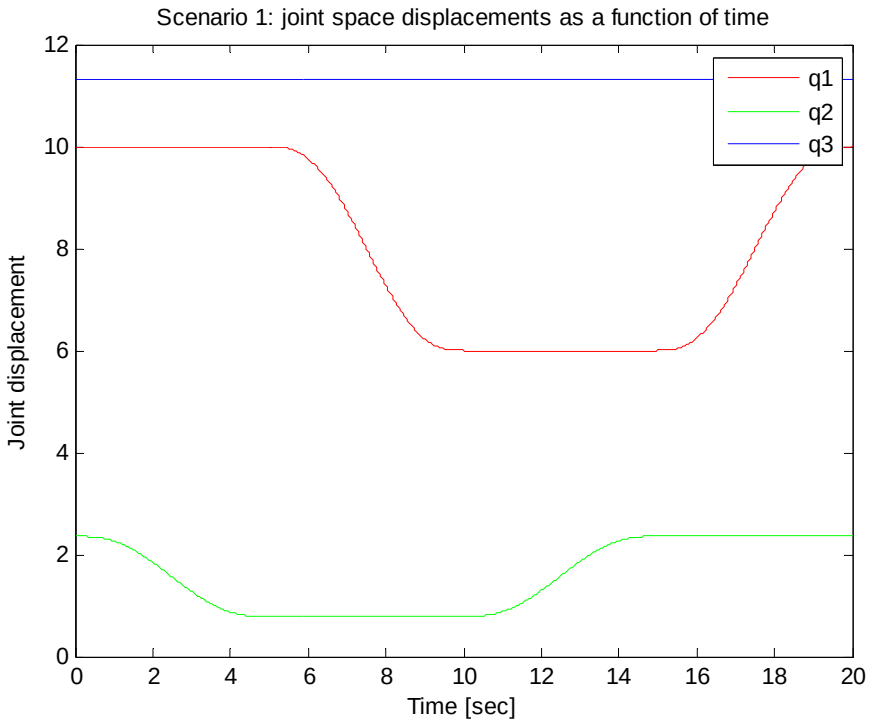
ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

Problem 2. *continued*

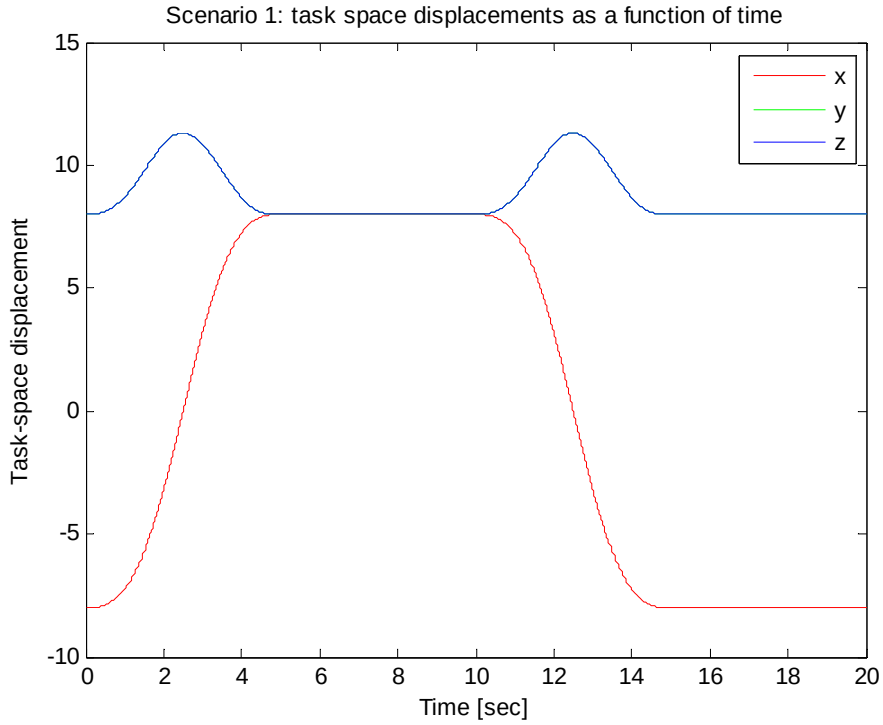
Scenario 1: Joint-space trajectory generation

- Calculate the joint space displacements that correspond to the given task-space waypoints $\left[\vec{X}_1, \vec{X}_2, \vec{X}_3, \text{and } \vec{X}_4 \right]$ (using the inverse kinematics). The task space Cartesian coordinates for each waypoint can be determined from the adjacent figure.
- Generate the four part trajectory, moving from waypoint *1* to *2*, *2* to *3*, *3* to *4*, and *4* to *1*. The time required to move between successive waypoint is 5 seconds. Generate the trajectory in joint-space, using the joint-space displacements calculated above. The trajectories should follow a quintic polynomial. At each waypoint, the joint-space velocity and acceleration should equal zero.
- Plot your results to include:
  - Joint space displacements as a function of time.
  - Task space displacements as a function of time (using the forward kinematics to calculate the task space displacements).
  - Task space displacements in 3-D space (using the Matlab `plot3` command).

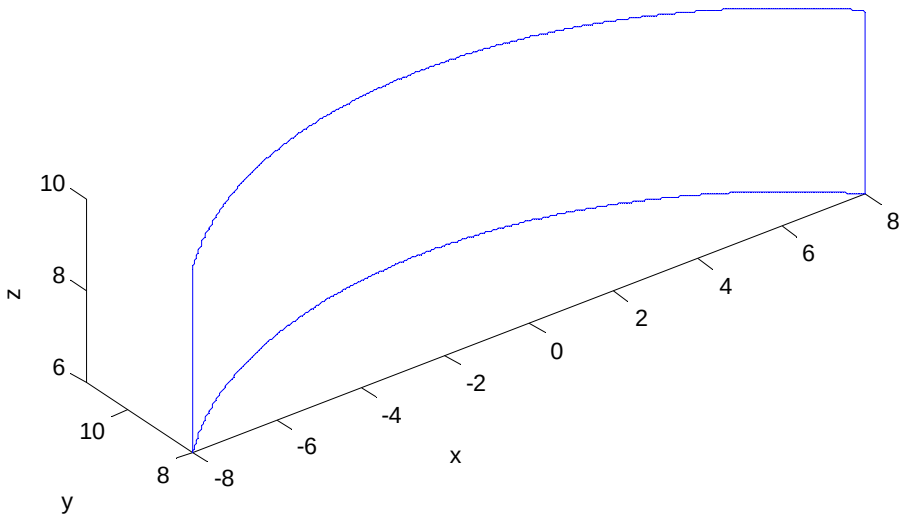


ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)



Scenario 1: task space displacements in 3-D space



ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

Scenario 1: Joint-space trajectory generation

```
% number of points in the trajectory
n = 1000;

% waypoint locations
L = 4; xOffset = 1; zOffset = 1; yOffset = 2;
xWP = [xOffset - L/2;  xOffset + L/2;  xOffset + L/2;  xOffset - L/2];
yWP = [      yOffset;        yOffset;        yOffset;        yOffset];
zWP = [zOffset + L/2;  zOffset + L/2;  zOffset - L/2;  zOffset - L/2];

% waypoint times
tWP = [0; 5; 10; 15; 20];

%-------------------------------------------------------
% Scenario 1
%-------------------------------------------------------

% calculate joint displacements at waypoint locations
q_WP1 = IK_PRP_manipulator([xWP(1); yWP(1); zWP(1)]);
q_WP2 = IK_PRP_manipulator([xWP(2); yWP(2); zWP(2)]);
q_WP3 = IK_PRP_manipulator([xWP(3); yWP(3); zWP(3)]);
q_WP4 = IK_PRP_manipulator([xWP(4); yWP(4); zWP(4)]);
q_WP = [q_WP1' q_WP2' q_WP3' q_WP4'];
q1WP = q_WP(1,:);  % joint 1 waypoints
q2WP = q_WP(2,:);  % joint 2 waypoints
q3WP = q_WP(3,:);  % joint 3 waypoints

% Calculate move from 1 to 2
[t1, q1_1, dq1_1, ddq1_1] = QuinticPolynomial(tWP(1), tWP(2), q1WP(1), ...
                                              q1WP(2), 0, 0, 0, 0, n);
[t1, q2_1, dq2_1, ddq2_1] = QuinticPolynomial(tWP(1), tWP(2), q2WP(1), ...
                                              q2WP(2), 0, 0, 0, 0, n);
[t1, q3_1, dq3_1, ddq3_1] = QuinticPolynomial(tWP(1), tWP(2), q3WP(1), ...
                                              q3WP(2), 0, 0, 0, 0, n);

% Calculate move from 2 to 3
[t2, q1_2, dq1_2, ddq1_2] = QuinticPolynomial(tWP(2), tWP(3), q1WP(2), ...
                                              q1WP(3), 0, 0, 0, 0, n);
[t2, q2_2, dq2_2, ddq2_2] = QuinticPolynomial(tWP(2), tWP(3), q2WP(2), ...
                                              q2WP(3), 0, 0, 0, 0, n);
[t2, q3_2, dq3_2, ddq3_2] = QuinticPolynomial(tWP(2), tWP(3), q3WP(2), ...
                                              q3WP(3), 0, 0, 0, 0, n);

% Calculate move from 3 to 4
[t3, q1_3, dq1_3, ddq1_3] = QuinticPolynomial(tWP(3), tWP(4), q1WP(3), ...
                                              q1WP(4), 0, 0, 0, 0, n);
[t3, q2_3, dq2_3, ddq2_3] = QuinticPolynomial(tWP(3), tWP(4), q2WP(3), ...
                                              q2WP(4), 0, 0, 0, 0, n);
[t3, q3_3, dq3_3, ddq3_3] = QuinticPolynomial(tWP(3), tWP(4), q3WP(3), ...
                                              q3WP(4), 0, 0, 0, 0, n);
```

```
% Calculate move from 4 to 1
[t4, q1_4, dq1_4, ddq1_4] = QuinticPolynomial(tWP(4), tWP(5), q1WP(4), ...
                                                q1WP(1), 0, 0, 0, 0, n);
[t4, q2_4, dq2_4, ddq2_4] = QuinticPolynomial(tWP(4), tWP(5), q2WP(4), ...
                                                q2WP(1), 0, 0, 0, 0, n);
[t4, q3_4, dq3_4, ddq3_4] = QuinticPolynomial(tWP(4), tWP(5), q3WP(4), ...
                                                q3WP(1), 0, 0, 0, 0, n);

% Assemble into a single trajectory
q1 = [q1_1; q1_2; q1_3; q1_4];
q2 = [q2_1; q2_2; q2_3; q2_4];
q3 = [q3_1; q3_2; q3_3; q3_4];
t = [t1; t2; t3; t4];

% Plot joint space displacements as a function of time.
figure;
plot(t,q1,'r',t,q2,'g',t,q3,'b');
xlabel('Time [sec]');
ylabel('Joint displacement')
legend('q1','q2','q3')
title('Scenario 1: joint space displacements as a function of time');

% Calculate task-space displacements using the forward kinematics
nPts = size(q1,1);
for i = 1:nPts
   X = FK_PRP_manipulator([q1(i) q2(i) q3(i)]);
   x(i) = X(1);
   y(i) = X(2);
   z(i) = X(3);
end

% Plot the task space displacements as a function of time
figure;
plot(t,x,'r',t,y,'g',t,y,'b');
xlabel('Time [sec]');
ylabel('Task-space displacement')
legend('x','y','z')
title('Scenario 1: task space displacements as a function of time');

% Plot the task space displacements in 3-D space
figure;
plot3(x,y,z);
axis equal
xlabel('x');
ylabel('y');
zlabel('z');
title('Scenerio 1: task space displacements in 3-D space');
```

FK_PRP_manipulator.m

```
function [x] = FK_PRP_manipulator(q)

xe = q(3)*cos(q(2));
ye = q(3)*sin(q(2));
ze = q(1);

x = [xe; ye; ze];

end
```

IK_PRP_manipulator.m

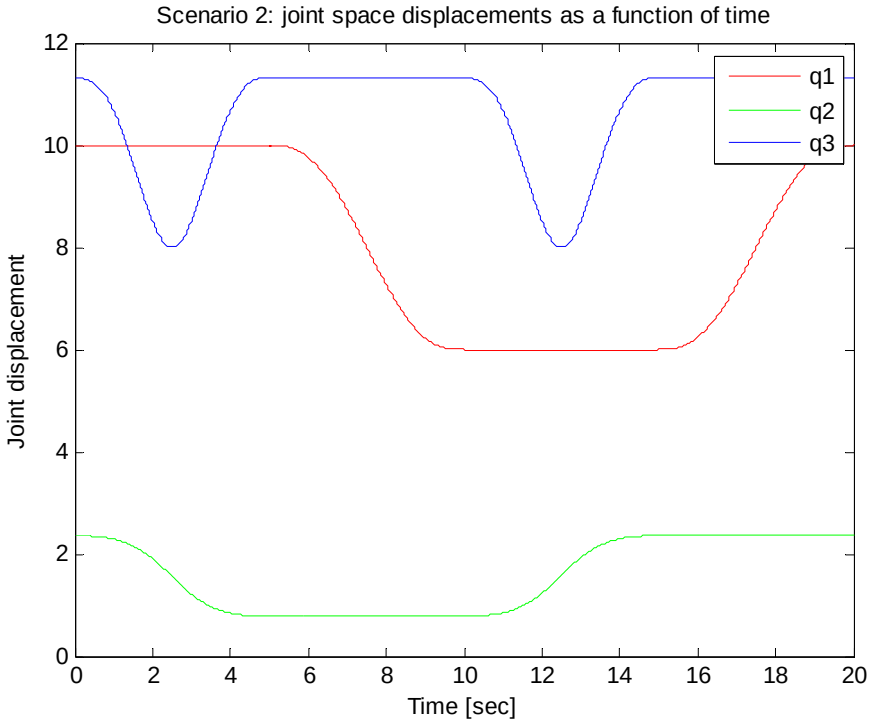
```
function [q] = IK_PRP_manipulator(x)

xe = x(1); ye = x(2); ze = x(3);

q(1) = ze;
q(2) = atan2(ye,xe);
q(3) = sqrt(xe^2 + ye^2);

end
```

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

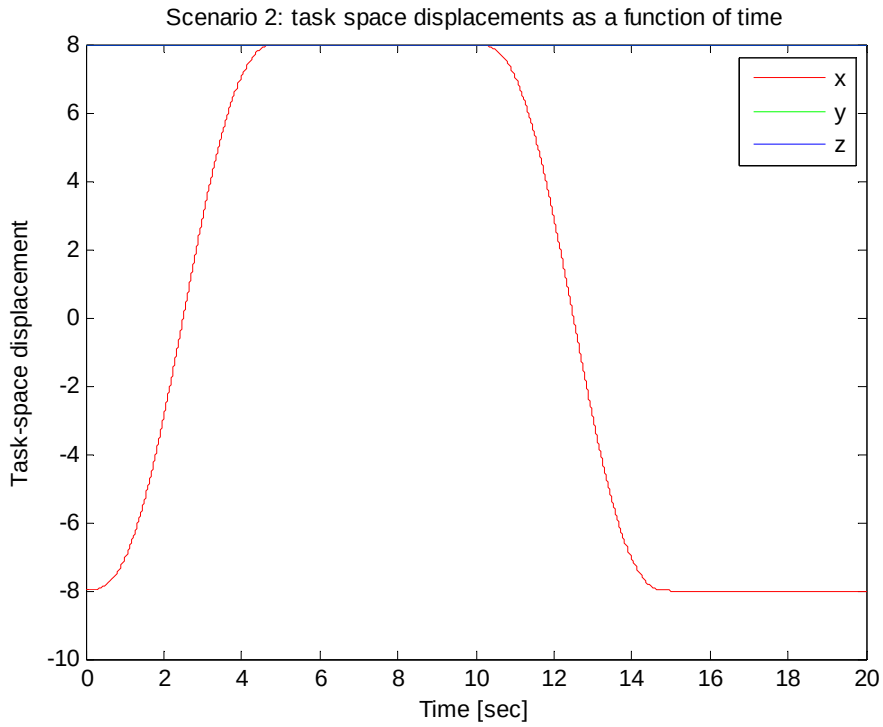
Scenario 2: Task-space trajectory generation

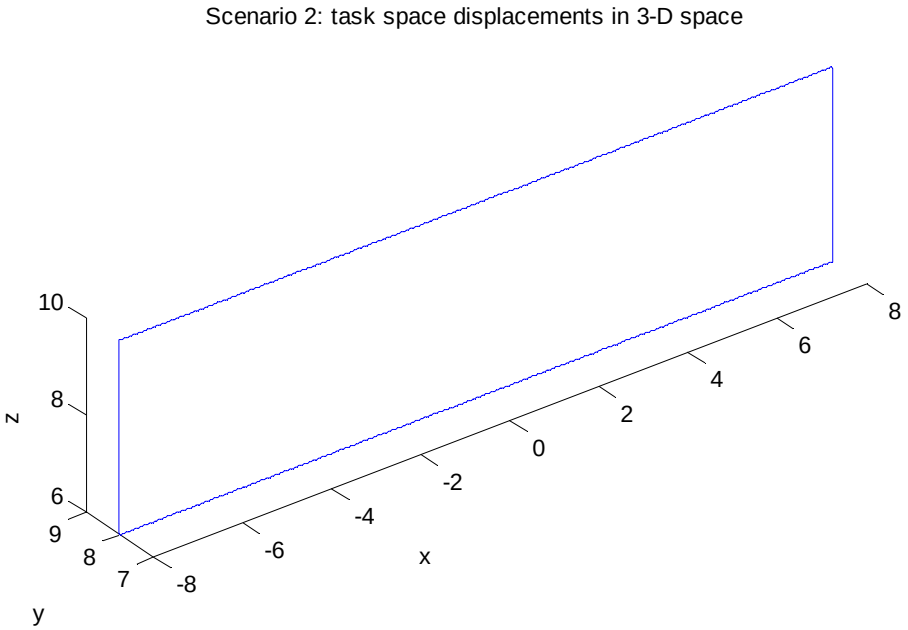
- Generate the four part trajectory, moving from waypoint *1* to *2*, *2* to *3*, *3* to *4*, and *4* to *1*. The time required to move between successive waypoint is 5 seconds. Generate the trajectory in task-space, using the given task-space waypoints. The trajectories should follow a quintic polynomial. At each waypoint, the task-space velocity and acceleration should equal zero.
- Plot your results to include:
  - Joint space displacements as a function of time (using the inverse kinematics to calculate the joint-space displacements).
  - Task space displacements as a function of time.
  - Task space displacements in 3-D space (using the Matlab `plot3` command).
- Comment on the differences between the two approaches (advantages and disadvantages).



ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)





Scenario 2: Joint-space trajectory generation

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

```
% Calculate move from 1 to 2
[t1, x1, dx1, ddx1] = QuinticPolynomial(tWP(1), tWP(2), xWP(1), ...
                                        xWP(2), 0, 0, 0, 0, n);
[t1, y1, dy1, ddy1] = QuinticPolynomial(tWP(1), tWP(2), yWP(1), ...
                                        yWP(2), 0, 0, 0, 0, n);
[t1, z1, dz1, ddz1] = QuinticPolynomial(tWP(1), tWP(2), zWP(1), ...
                                        zWP(2), 0, 0, 0, 0, n);

% Calculate move from 2 to 3
[t2, x2, dx2, ddx2] = QuinticPolynomial(tWP(2), tWP(3), xWP(2), ...
                                        xWP(3), 0, 0, 0, 0, n);
[t2, y2, dy2, ddy2] = QuinticPolynomial(tWP(2), tWP(3), yWP(2), ...
                                        yWP(3), 0, 0, 0, 0, n);
[t2, z2, dz2, ddz2] = QuinticPolynomial(tWP(2), tWP(3), zWP(2), ...
                                        zWP(3), 0, 0, 0, 0, n);

% Calculate move from 3 to 4
[t3, x3, dx3, ddx3] = QuinticPolynomial(tWP(3), tWP(4), xWP(3), ...
                                        xWP(4), 0, 0, 0, 0, n);
[t3, y3, dy3, ddy3] = QuinticPolynomial(tWP(3), tWP(4), yWP(3), ...
                                        yWP(4), 0, 0, 0, 0, n);
[t3, z3, dz3, ddz3] = QuinticPolynomial(tWP(3), tWP(4), zWP(3), ...
                                        zWP(4), 0, 0, 0, 0, n);

% Calculate move from 4 to 1
[t4, x4, dx4, ddx4] = QuinticPolynomial(tWP(4), tWP(5), xWP(4), ...
                                        xWP(1), 0, 0, 0, 0, n);
[t4, y4, dy4, ddy4] = QuinticPolynomial(tWP(4), tWP(5), yWP(4), ...
                                        yWP(1), 0, 0, 0, 0, n);
[t4, z4, dz4, ddz4] = QuinticPolynomial(tWP(4), tWP(5), zWP(4), ...
                                        zWP(1), 0, 0, 0, 0, n);


% Assemble into a single trajectory
x = [x1; x2; x3; x4];
y = [y1; y2; y3; y4];
z = [z1; z2; z3; z4];
t = [t1; t2; t3; t4];

% Calculate joint-space displacements using the inverse kinematics
nPts = size(q1,1);
for i = 1:nPts
   Q = IK_PRP_manipulator([x(i) y(i) z(i)]);
   q1(i) = Q(1);
   q2(i) = Q(2);
   q3(i) = Q(3);
end
```

```
% Plot joint space displacements as a function of time.
figure;
plot(t,q1,'r',t,q2,'g',t,q3,'b');
xlabel('Time [sec]');
ylabel('Joint displacement')
legend('q1','q2','q3')
title('Scenario 2: joint space displacements as a function of time');

% Plot the task space displacements as a function of time
figure;
plot(t,x,'r',t,y,'g',t,y,'b');
xlabel('Time [sec]');
ylabel('Task-space displacement')
legend('x','y','z')
title('Scenario 2: task space displacements as a function of time');

% Plot the task space displacements in 3-D space
figure;
plot3(x,y,z);
axis equal
xlabel('x');
ylabel('y');
zlabel('z');
title('Scenario 2: task space displacements in 3-D space');
```

ME/ECE 739: Advanced Robotics — Homework #4

Due: April 26th (Sunday, 11:59 pm)

Problem 3. [40 points]

For the manipulator described in Problem 2

- Use a parabolic well potential to define an attractive field applied to the origin of frame {e}. Use Matlab to implement a gradient descent algorithm to find a path from the specified initial configuration to the specified final configuration given below.

  Initial configuration: $q_0 = \begin{bmatrix} L & 0 & L \end{bmatrix}^T$

  Final configuration: $q_f = \begin{bmatrix} 2L & \frac{1}{2}\pi & 2L \end{bmatrix}^T$

- Animate your results using the Matlab rendering functions posted to the Learn@UW course page.

In general, you can vary ζ_i (the scaling factor applied to the attractive force, f_i) to modify the resulting path. In this case, the forces are applied at only one point so there is only one value of ζ. You can also vary the gradient descent scaling factor, α. Larger values of α can speed up the gradient descent algorithm, but can also cause numerical instabilities in the solution. In addition, you can use different values of α (α_i for joint i) to scale the joint torques and thus modify the path solution.

Gradient descent:

$$\left[\begin{array}{l} \textit{while } \left\| q^k - q_f \right\| > \varepsilon \\ \qquad q^{k+1} = q^k + \alpha \nabla U_\tau \quad \text{where } \nabla U_\tau = \dfrac{\tau\left(q^k\right)}{\left\| \tau\left(q^k\right) \right\|} \text{ and } \alpha = \begin{bmatrix} \alpha_1 & & 0 \\ & \ddots & \\ 0 & & \alpha_n \end{bmatrix} \\ \qquad k = k+1 \end{array}\right.$$

- Vary α_i and comment on the resulting solution behavior (both in regards to the shape of the path and the numerical stability of the solution.

Problem 3:

```
% attractive force scaling
zeta = 1;

% initial and goal position (joint space)
L = 5;
q0 = [L 0 L]';
o1g = [0; 2*L; 2*L];
qGoal = IK_PRP_manipulator(o1g)';

%gradient descent step size
alpha = 1*[0.1; 0.01; 0.1];

%gradient descent error tolerance
epsilon = 0.0001;

%max gradient descent iterations
maxIterations = 10000;

% initialize joint positions
q = q0;

%initialize counter
k = 1;
```

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

```
%------------------------------
% Gradient Descent
%------------------------------

while(1)


    % trigometric evaluations
    c2 = cos(q(2)); s2 = sin(q(2));
    % forward kinematics
    o1 = [q(3)*c2
          q(3)*s2
          q(1)];

    % attractive forces
    Fa1 = -zeta*(o1 - o1g);


    % point Jacobians
    Jo1 = [0  -q(3)*s2  c2
           0   q(3)*c2  s2
           1      0      0];

     %joint torques
    tau = Jo1'*Fa1;

    %update q
    q = q + alpha.*tau/sqrt(tau'*tau);

    % store variables
    Q(:,k) = q;
    FA1(:,k) = Fa1;
    TAU(:,k) = tau;

    %check convergence
    error = (q - qGoal)'*(q - qGoal);
    if error < epsilon
        break;
    end
    if k > maxIterations;
        break
    end
    k = k + 1;

end
```

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

```
%------------------------------
% Render result
%------------------------------

% Rendering parameters

dT = 0.001;
f_handle = 1;
axis_limits = L*[-2 2 -2 2 -2 2];
render_view = [-.5 -1 1]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,...
                           view_up,f_handle);


% Initialize link rendering

% link 0 rendering initialization
L0 = L; r0 = L0/4; sides0 = 4; axis0 = 3; norm_L0 = -1.0;
    linkColor0 = [0.5 0.5 0.5]; plotFrame0 = 0;
d0 =  CreateLinkRendering(L0,r0,sides0,axis0,norm_L0,linkColor0,...
                          plotFrame0,f_handle);

% link 1 rendering initialization
L1 = L; r1 = L1/5; sides1 = 4; axis1 = 3; norm_L1 = 1.0;
    linkColor1 = [0 0.75 0]; plotFrame1 = 0;
d1 =  CreateLinkRendering(L1,r1,sides1,axis1,norm_L1,linkColor1,...
                          plotFrame1,f_handle);

% link 2 rendering initialization
L2 = L; r2 = L2/6; sides2 = 4; axis2 = 3; norm_L2 = -1.0;
    linkColor2 = [0.75 0 0]; plotFrame2 = 0;
d2 =  CreateLinkRendering(L2,r2,sides2,axis2,norm_L2,linkColor2,...
                          plotFrame2,f_handle);

% link 3 rendering initialization
L3 = L; r3 = L3/8; sides3 = 4; axis3 = 1; norm_L3 = 1.0;
    linkColor3 = [0 0 0.75]; plotFrame3 = 0;
d3 =  CreateLinkRendering(L3,r3,sides3,axis3,norm_L3,linkColor3,...
                          plotFrame3,f_handle);
```

ME/ECE 739: Advanced Robotics Homework #4

Due: April 26th (Sunday, 11:59 pm)

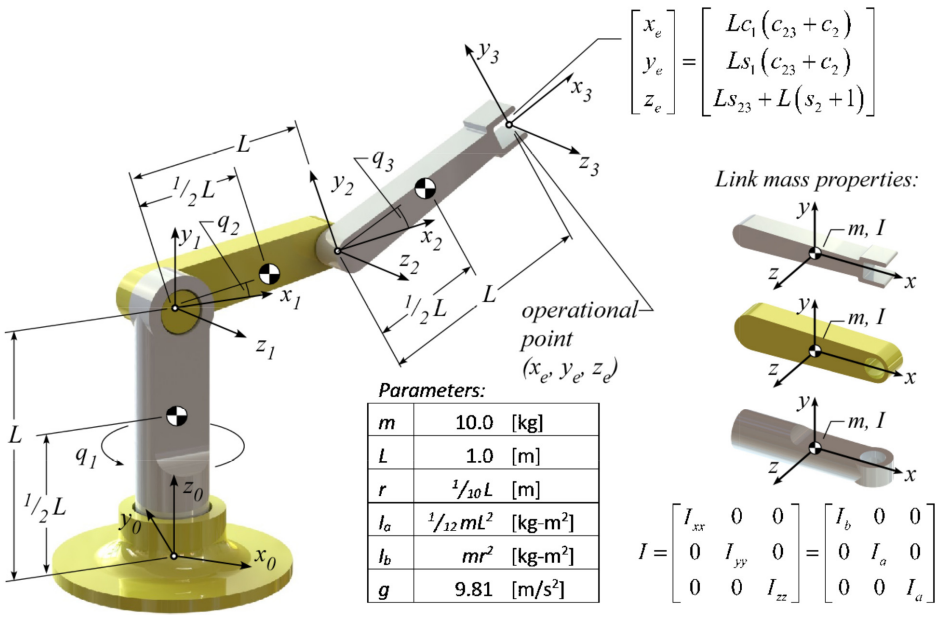
```
% Update rendering
for i = 1:k
    q1 = Q(1,i);  q2 = Q(2,i);  q3 = Q(3,i);

    % Update frame {1}
    T10 = [1  0  0    0
           0  1  0    0
           0  0  1   q1
           0  0  0    1];
     % Update frame {2}
    s = sin(q2); c = cos(q2);
    T21 = [-s  0  c  0
            c  0  s  0
            0  1  0  0
            0  0  0  1];
    % Update frame {e}
    Te2 = [0  1  0   0
           0  0  1   0
           1  0  0  q3
           0  0  0   1];
    T20 = T10*T21;
    Te0 = T20*Te2;
    UpdateLink(d1,T10);
    UpdateLink(d2,T20);
    UpdateLink(d3,Te0);

    if i == 1;  %pause at start of simulation rendering
        pause;
    end
    pause(dT);

end
```

2.5 HW 5

2.5.1 Problem description

Problem 1. [100 points]

You are to design a series of position controllers for the three degree of freedom manipulator shown in the figure below. The equations of motion and corresponding Matlab simulation code are given in the lecture notes (4-9 – Dynamics – EOM Simulation: slides 4-136 to 4-147). The simulation code has been posted on the course Learn@UW page (Simulation Example #2: Three DOF RRR Manipulator). For completeness, the equations of motion are given on the next page.



Equations of motion: $D\ddot{q} + B[\dot{q}\dot{q}] + C[\dot{q}^2] + G = \tau$ where the terms are given as:

Mass matrix:

$$D = \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & D_{23} \\ 0 & D_{32} & D_{33} \end{bmatrix} \quad \text{where} \quad \begin{aligned} D_{11} &= \tfrac{1}{4}mL^2c_2{}^2 + \tfrac{1}{4}mL^2\left(c_{23} + 2c_2\right)^2 + \cdots \\ &\quad I_b + I_a\left(c_2{}^2 + c_{23}{}^2\right) + I_b\left(s_2{}^2 + s_{23}{}^2\right) \\ D_{22} &= \tfrac{3}{2}mL^2 + mL^2c_3 + 2I_a \\ D_{23} &= \tfrac{1}{4}mL^2 + \tfrac{1}{2}mL^2c_3 + I_a \\ D_{32} &= d_{23} \\ D_{33} &= \tfrac{1}{4}mL^2 + I_a \end{aligned}$$

Coriolis matrix:

$$B(q)[\dot{q}\dot{q}] = \begin{bmatrix} B_{11} & B_{12} & 0 \\ 0 & 0 & B_{23} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1\dot{q}_2 \\ \dot{q}_1\dot{q}_3 \\ \dot{q}_2\dot{q}_3 \end{bmatrix} \quad \text{where} \quad \begin{aligned} B_{11} &= \left(I_b - I_a - \tfrac{1}{4}mL^2\right)\sin\left(2q_2 + 2q_3\right) + \cdots \\ &\quad \left(I_b - I_a - \tfrac{5}{4}mL^2\right)\sin\left(2q_2\right) - mL^2\sin\left(2q_2 + q_3\right) \\ B_{12} &= -\tfrac{1}{2}\Big(\sin\left(q_2 + q_3\right)\Big(\left(4I_a - 4I_b + mL^2\right)\cdots \\ &\quad \cos\left(q_2 + q_3\right) + 2mL^2\cos\left(q_2\right)\Big)\Big) \\ B_{23} &= -mL^2\sin\left(q_3\right) \end{aligned}$$

Centrifugal matrix:

$$\mathrm{C}(q)[\dot{q}^2] = \begin{bmatrix} 0 & 0 & 0 \\ C_{21} & 0 & C_{23} \\ C_{31} & C_{32} & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1{}^2 \\ \dot{q}_2{}^2 \\ \dot{q}_3{}^2 \end{bmatrix} \quad \text{where} \quad \begin{aligned} C_{21} &= \tfrac{1}{2}\left(I_a - I_b + \tfrac{1}{4}mL^2\right)\sin\left(2q_2 + 2q_3\right) + \cdots \\ &\quad \tfrac{1}{2}\left(I_a - I_b + \tfrac{5}{4}mL^2\right)\sin\left(2q_2\right) + \tfrac{1}{2}mL^2\sin\left(2q_2 + q_3\right) \\ C_{23} &= -\tfrac{1}{2}mL^2\sin\left(q_3\right) \\ C_{32} &= \tfrac{1}{2}mL^2\sin\left(q_3\right) \\ C_{31} &= \tfrac{1}{4}\Big(\sin\left(q_2 + q_3\right)\Big(\left(4I_a - 4I_b + \tfrac{1}{4}mL^2\right)\cdots \\ &\quad \cos\left(q_2 + q_3\right) + 2mL^2\cos\left(q_2\right)\Big)\Big) \end{aligned}$$

Gravity vector:

$$G = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \quad \text{where} \quad \begin{aligned} g_1 &= 0 \\ g_2 &= \tfrac{1}{2}mgL\left(3\mathrm{c}_2 + c_{23}\right) \\ g_3 &= \tfrac{1}{2}mgLc_{23} \end{aligned}$$

► ***Joint-space Control***

In parts (a) – (c) design and implement a joint-space inverse dynamics controller. Use a simple proportional-derivative (PD) controller to control the decoupled system. Set the controller gains such that the closed-loop position controller has an undamped natural frequency, ω_n, equal to 2 Hz and a damping ratio, ζ, equal to 1.0. Simulate the response of the system to a step input position command. The initial joint positions of the manipulator are given as $q_0 = [0 \quad \pi/4 \quad -\pi/2]^T$. The final (or desired) joint positions of the manipulator are given as $q_f = [\pi \quad \pi \quad \pi/2]^T$. Note, when simulating your system, you may need to adjust the integration step size of the Runge-Kutta algorithm to ensure that the system does not become ill-conditioned.

For parts (a) – (c) below, plot/animate the following:

- Plot the joint space displacements and velocities as a function of time
- Plot the operational-point (task frame origin) displacements and velocities as a function of time
- Plot the operational-point displacements in three-dimensions [i.e. `plot3(x, y, z)`]
- Animate the motion of the manipulator

(a) Design and implement a joint-space inverse dynamics controller, where the nonlinear Coriolis, centrifugal, and gravity terms are compensated and the equations are completely decoupled using the joint-space mass matrix.

(b) Design and implement a modified joint-space inverse dynamics controller. In this case, compensate for the gravitational terms but do not compensate for the nonlinear Coriolis or centrifugal terms.

(c) Design and implement a decentralized joint-space controller. In this case, compensate for the nonlinear Coriolis, centrifugal, and gravity terms but decouple the system using an *average* mass matrix. Derive an average mass matrix using the true mass matrix. Explain and justify your choice of an average (constant) mass matrix.

(d) Compare the response of the three controllers from parts (a) – (c) and comment on the differences, advantages, and disadvantages.

► ***Operational-space Control***

In parts (a) – (c), design and implement an operational-space inverse dynamics controller. Use a simple proportional-derivative (PD) controller to control the decoupled system. Set the controller gains such that the closed-loop position controller has an undamped natural frequency, ω_n, equal to 2 Hz and a damping ratio, ζ, equal to 1.0.

For parts (a) – (c) below, plot/animate the following:

- Plot the joint space displacements and velocities as a function of time
- Plot the operational-point (task frame origin) displacements and velocities as a function of time
- Plot the operational-point displacements in three-dimensions [i.e. `plot3(x, y, z)`]
- Animate the motion of the manipulator

(a) Design and implement an operational-space inverse dynamics controller, where the nonlinear Coriolis, centrifugal, and gravity terms are compensated and the equations are completely decoupled using the operational-space mass matrix[1]. Simulate the response of the system to a step input position command. The initial joint positions of the manipulator are given as $q_0 = [0 \quad \pi/4 \quad -\pi/2]^T$. The final operational-space position of the manipulator is given as $x_f = [-L \quad -L \quad 0]^T$.

(b) Using the controller designed in part (a), simulate the response of the system to a step input position command where the final operational-space position of the manipulator is given as $x_f = [-L \quad -L/10 \quad 0]^T$ (the initial joint positions of the manipulator are still given as $q_0 = [0 \quad \pi/4 \quad -\pi/2]^T$.

(c) Using the controller designed in part (a), implement the operational-space linear velocity limiting heuristic described in the lecture notes. Set the maximum linear velocity equal to 5 m/s. Simulate the response of the system to a step input position command where the initial joint positions are given as $q_0 = [0 \quad \pi/4 \quad -\pi/2]^T$ and the final operational-space positions is given as $x_f = [-L \quad -L \quad 0]^T$.

(d) Compare the response of the controllers from parts (a) and (b) comment on the differences, advantages, and disadvantages.

2.5.2 Joint space control

The main goal is to decouple the nonlinear equation of motion of the robotic arm which is given by the equation below, where [ ] indicates a matrix and { } indicates a column vector. Notice that $[D(q)]$ means that that matrix $[D]$ is a function of q. It does not mean the matrix $[D]$ is multiplied by q.

$$[D(q)]\{\ddot{q}\} + [B(q)][\dot{q}\dot{q}] + [C(q)]\{\dot{q}^2\} + [G(q)] = \{\tau\}$$

The sizes of each of above quantities in terms of n which is the number of generalized coordinates, or the degrees of freedom, or the number of joints, which is 3 in this example, are given by

$$\overbrace{[D(q)]}^{n\times n}\overbrace{\{\ddot{q}\}}^{n\times 1} + \overbrace{[B(q)]}^{n\times \frac{(n-1)n}{2}}\overbrace{[\dot{q}\dot{q}]}^{\frac{(n-1)n}{2}\times 1} + \overbrace{[C(q)]}^{n\times n}\overbrace{\{\dot{q}^2\}}^{n\times 1} + \overbrace{[G(q)]}^{n\times 1} = \overbrace{\{\tau\}}^{n\times 1}$$

The velocity terms and the gravity terms and the mass terms are indicated below

$$\overbrace{[D(q)]\{\ddot{q}\}}^{\text{Mass or inertia term}} + \overbrace{[B(q)][\dot{q}\dot{q}] + [C(q)]\dot{q}^2\}}^{\text{velocity nonlinear terms}} + \overbrace{[G(q)]}^{\text{gravity nonlinear term}} = \overbrace{\{\tau\}}^{\text{forces and torques}}$$

The above equation of motion can be written in simpler form for analysis by letting $V = [B(q)][\dot{q}\dot{q}] + [C(q)]\{\dot{q}^2\}$. Therefore the above becomes

$$\begin{aligned} [D]\{\ddot{q}\} + V + G &= \tau \\ [D]\{\ddot{q}\} &= \tau - V - G \end{aligned} \tag{1}$$

Setting $\tau = \tilde{D}\tau' + \tilde{V} + \tilde{G}$ where $\tilde{D}, \tilde{V}, \tilde{G}$ are estimates of the the actual D, V, G. The estimates are computed in real time. τ' is the actuating signal generated by the propertional derivative (P.D.) controller which is given by

$$\tau' = k_d(\dot{q}_{\text{desired}} - \dot{q}_{\text{actual}}) + k_p(q_{\text{desired}} - q_{\text{actual}})$$

Using the above equation (1) becomes

$$\begin{aligned} D[\ddot{q}] &= \tilde{D}\tau' + \tilde{V} + \tilde{G} - V - G \\ &= \tilde{D}\tau' + (\tilde{V} - V) + (\tilde{G} - G) \end{aligned} \tag{2}$$

Assuming the estimates are perfect with no noise and no delay, then $\tilde{V} = V$, $\tilde{G} = G$ and $\tilde{D} = D$ and (2) reduces to

$$\begin{aligned} [I]\{\ddot{q}\} &= D^{-1}\tilde{D}\tau' \\ &= \{\tau'\} \end{aligned} \tag{3}$$

Where $[I]$ is the identity matrix. Therefore the equations have been decoupled.

τ was determined based on the control being implemented as follows

part(a)	$\tau = \tilde{D}\tau' + \tilde{G} + \tilde{V}$	full compensation
part(b)	$\tau = \tilde{D}\tau' + \tilde{G}$	no velocity terms compensation, only gravity is compensated for. This means the V term is not estimated at run time hence the equations of motion will not be fully decoupled and some cross joints motion effects will result.
part (c)	$\tau = D_{\text{average}}\tau' + \tilde{G} + \tilde{V}$	decentralized control with full compensation. Mass matrix is constant which represents the average mass matrix.

For part (c), the average mass matrix D_{average} was found by setting the joint angles to zero $q_i = 0$ for the three joints in the original $[D]$ mass matrix. Therefore all the $\cos(q)$ terms were replaced by one and all $\sin(q)$ terms were replaced by zero.

Damping was not used as the problem did not specify one but this can be easily added in the Matlab code. In addition, the gear ratio N was not used as the problem did not specify a value for N.

The end-effector position in task space was found to be

$$X_{\text{end}} = T_3^0 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} L\cos(q_1(t))\cos(q_2(t)) + L\cos(q_1(t))\cos(q_2(t))\cos(q_3(t)) - L\cos(q_1(t))\sin(q_2(t))\sin(q_3(t)) \\ L\cos(q_2(t))\sin(q_1(t)) + L\cos(q_2(t))\cos(q_3(t))\sin(q_1(t)) - L\sin(q_1(t))\sin(q_2(t))\sin(q_3(t)) \\ L\left(\sin(q_2(t)) + 1\right) + L\cos(q_2(t))\sin(q_3(t)) + L\cos(q_3(t))\sin(q_2(t)) \end{bmatrix}$$

The above was evaluated at each time instance and used to plot the path of the end-effector in 3D.

The following diagram shows the controller layout taken from the class handout, page 6-120 which shows the controller with full compensation.

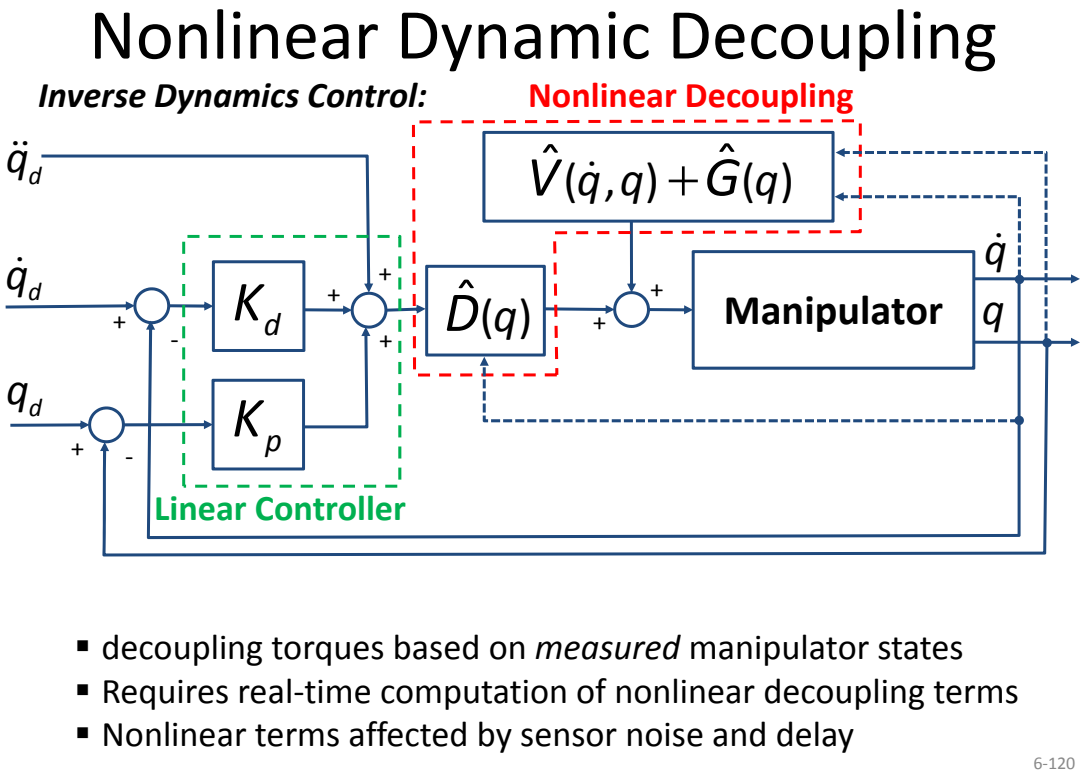


Figure 2.39: Diagram of controller, inverse dynamics control, full compensation. From class lecture notes

In the Matlab simulation, it was assumed that the measured manipulator states are exact and no noise was present as mentioned above. In practice this will not be the case and there will be an error in the estimates.

The following diagrams gives a high level overview of the Matlab software design for the implementation of the joint space control and the M files used.

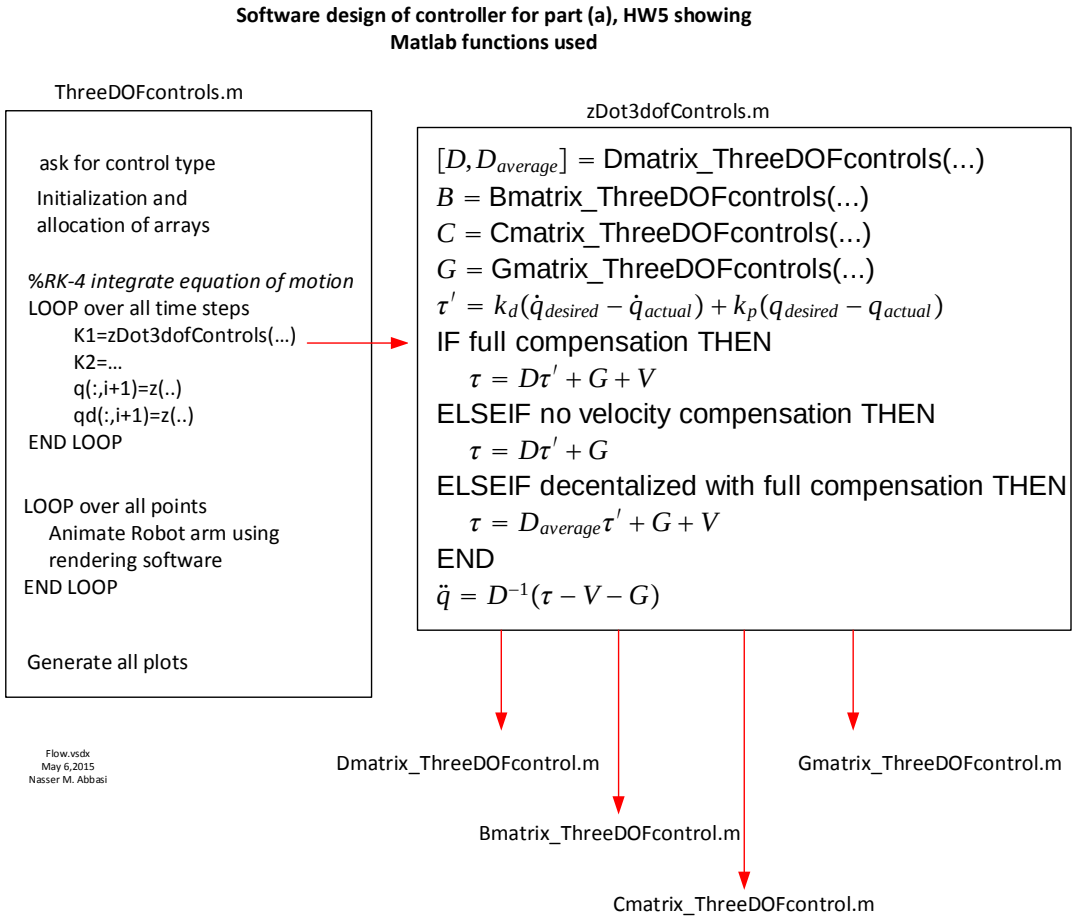


Figure 2.40: Matlab program design for joint space controller

The output for each part is now given followed by discussion of the results.

Part (a) Coriolis, centrifugal, and gravity terms are compensated

In this part, full compensation was made for the nonlinear Coriolis, centrifugal and gravity terms. This gave complete decoupling between the joints. Becuase of this one expects no oscillation in the plots of the joints displacements over the time of the simulation. This was verified from the plots generated by the simulation.

In addition to the required plots, an additional plot was made showing the end-effector speed over time. This was found by finding the end-effector linear speed using $\dot{X} = J\dot{q}$ where J is the end-effector Jacobian. The Jacobian was found using symbolic matlab in the file `ThreeDOFcontrols_symbolic.m` and the output was used in the the file `ThreeDOFcontrol.m` in order to calculate $|\dot{X}|$ over each time step.

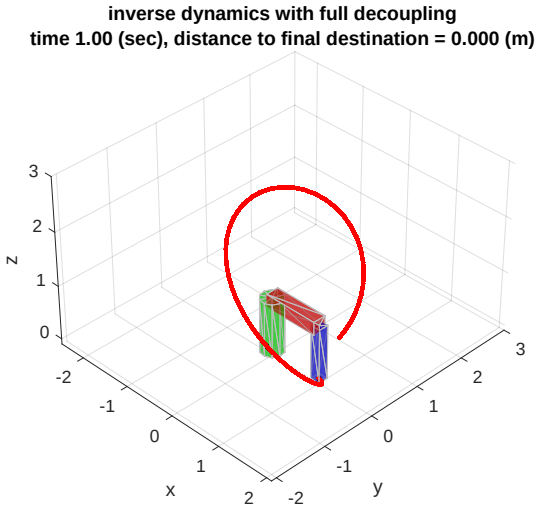


Figure 2.41: Final position of robot arm, Joint space control, part(a)

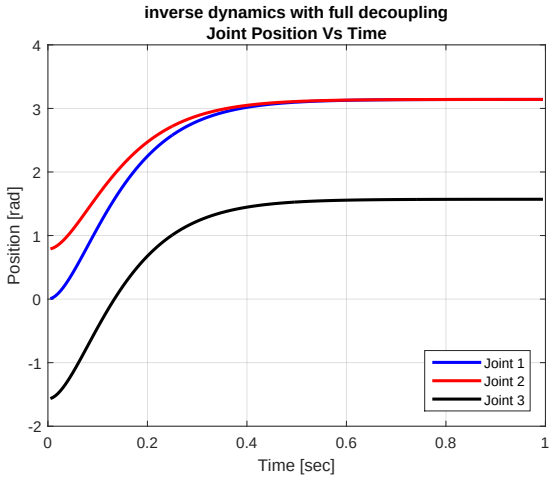


Figure 2.42: Joint position vs. time, Joint space control, part(a)

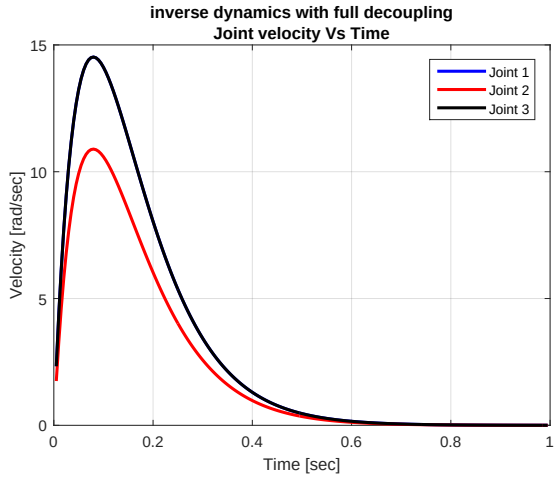


Figure 2.43: Joint velocity vs. time, Joint space control, part(a)

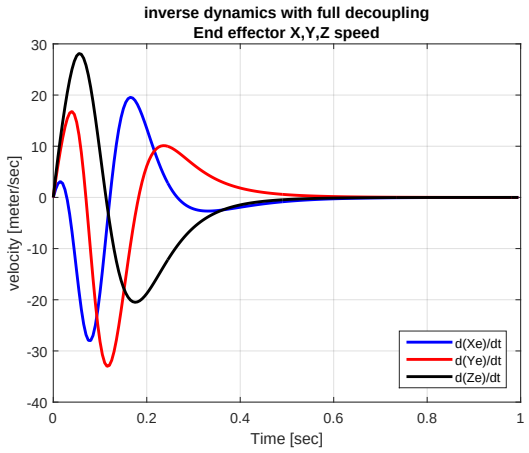


Figure 2.44: End effector $\frac{dX_e}{dt}, \frac{dY_e}{dt}, \frac{dZ_e}{dt}$ Joint space control, part(a)

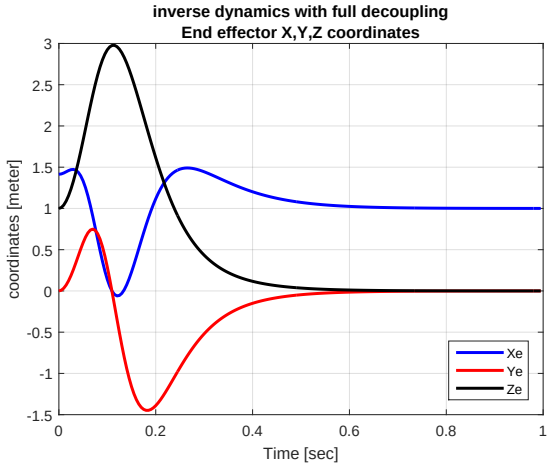


Figure 2.45: End effector X_e, Y_e, Z_e vs. time, Joint space control, part(a)

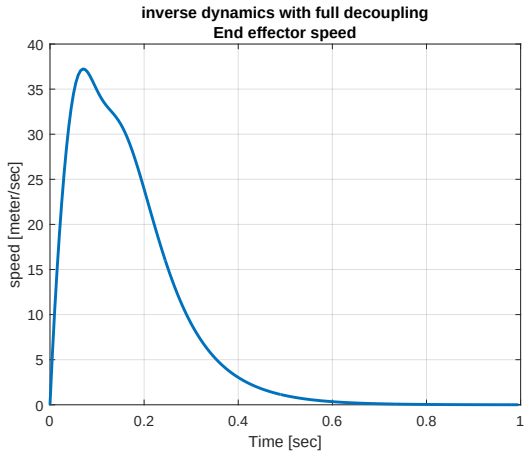


Figure 2.46: End effector linear speed vs. time, Joint space control, part(a)

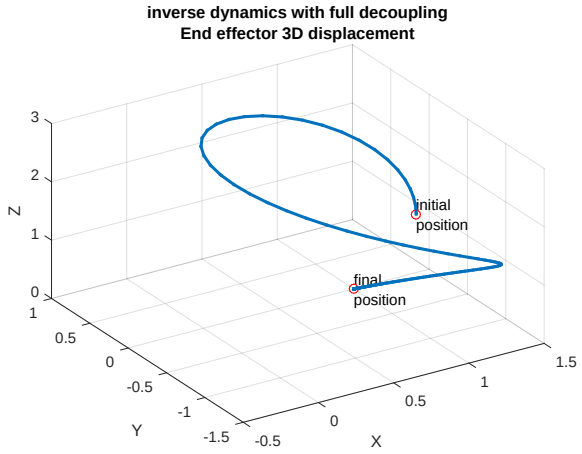


Figure 2.47: End effector plot3 displacement, Joint space control, part(a)

Part (b) No velocity compensation. Only gravity compensation

In this part, only the gravity terms were compensated for. Since there is coupling that remains between the joints, one expects to see some oscillation in the joints speeds as they are no longer independent from each other as with part (a).

This was verified by the plots generated from the simulation.

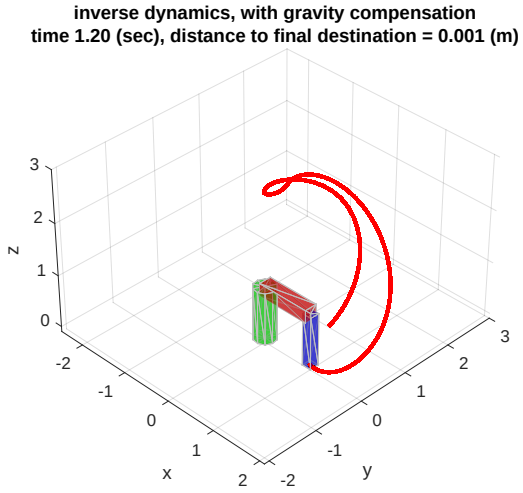


Figure 2.48: Final position of robot arm, Joint space control, part(b)

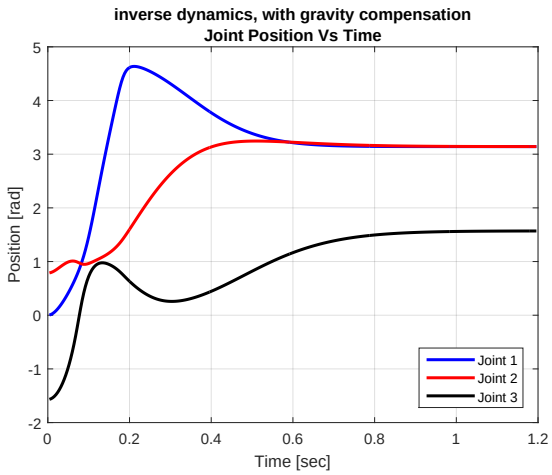


Figure 2.49: Joint position vs. time, Joint space control, part(b)

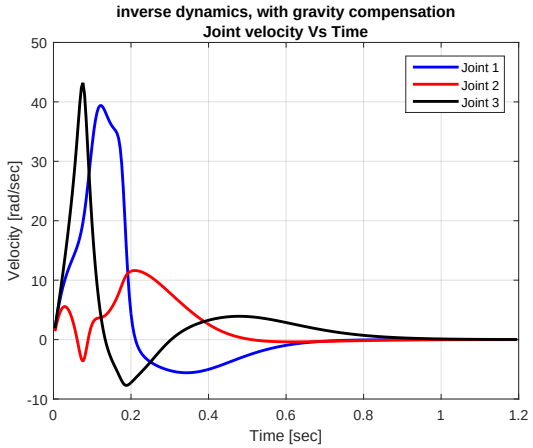


Figure 2.50: Joint velocity vs. time, Joint space control, part(b)

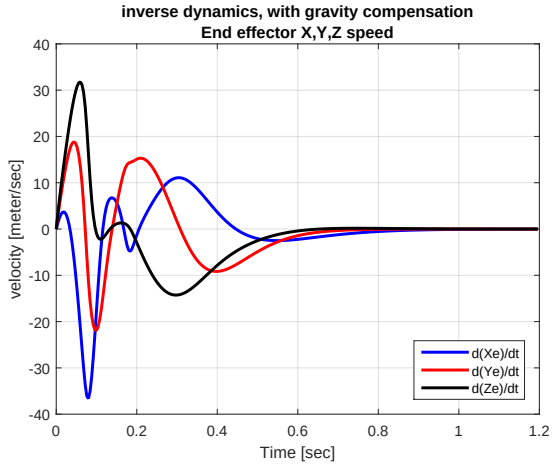


Figure 2.51: End effector $\frac{dX_e}{dt}$, $\frac{dY_e}{dt}$, $\frac{dZ_e}{dt}$ Joint space control, part(b)

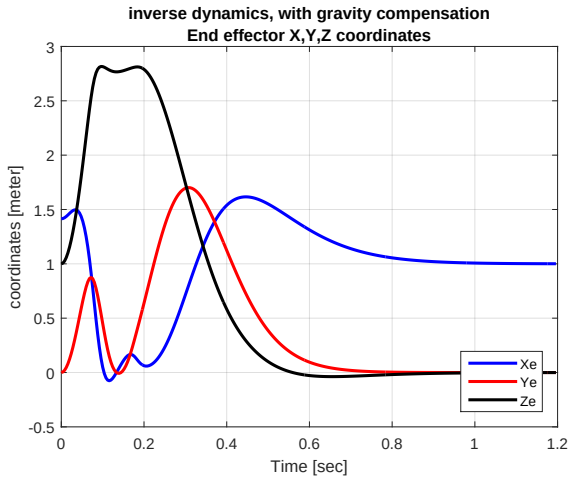


Figure 2.52: End effector X_e, Y_e, Z_e vs. time, Joint space control, part(b)

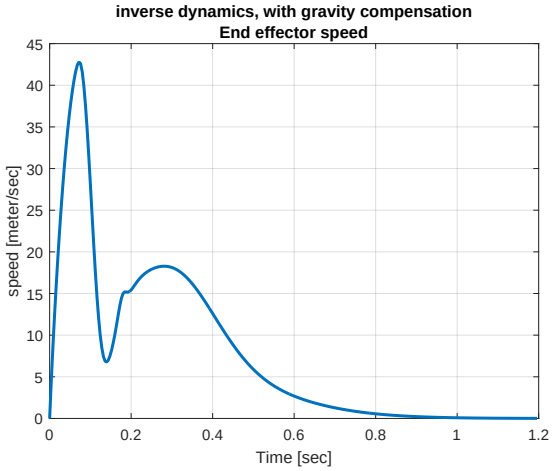


Figure 2.53: End effector linear speed vs. time, Joint space control, part(b)

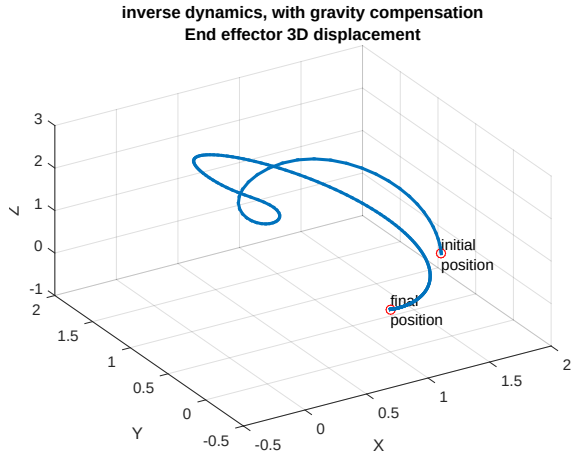


Figure 2.54: End effector plot3 displacement, Joint space control, part(b)

Part (c) Decentralized joint-space controller, full compensation using average D matrix

In this part, full compensation were made for the nonlinear Coriolis, centrifugal, and gravity terms, however the mass matrix D used was a constant matrix which represented the average of the original mass matrix.

The approximate mass matrix is given by

$$\tilde{D} = D_{\text{average}} + J_m^T I_m J_m$$

Where I_m is the actuator moment of inertia and J_m is the actuator Jacobian. In this problem these were not used as they were not specified, and only the average mass matrix needed to be determined.

The average mass matrix D_{average} was obtained from D by setting each joint position given by q to zero and by also setting the off diagonal elements to zero. Setting the off diagonal elements to zero was needed as the average mass matrix needs to be diagonal to produce the decoupling effect.

Therefore all the cosine terms were set to one and all the sine terms were set to zero. The original D matrix is

$$[D] = \begin{bmatrix} \frac{1}{4}mL^2c_2^2 + \frac{1}{4}mL^2(c_{23}+2c_2)^2 + I_b + I_a(c_2^2+c_{23}^2) + I_b(s_2^2+s_{23}^2) & 0 & 0 \\ 0 & \frac{3}{2}mL^2 + mL^2c_3 + 2I_a & \frac{1}{4}mL^2 + \frac{1}{2}mL^2 \\ 0 & \frac{1}{4}mL^2 + \frac{1}{2}mL^2c_3 + I_a & \frac{1}{4}mL^2 + \end{bmatrix}$$

Hence the average D matrix becomes

$$[D_{\text{average}}] = \begin{bmatrix} \frac{1}{4}mL^2 + \frac{1}{4}mL^2(1+2)^2 + I_b + I_a(1+1) + I_b(0+0) & 0 & 0 \\ 0 & \frac{3}{2}mL^2 + mL^2 + 2I_a & 0 \\ 0 & 0 & \frac{1}{4}mL^2 + I_a \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{4}mL^2 + \frac{9}{4}mL^2 + I_b + 2I_a & 0 & 0 \\ 0 & \frac{3}{2}mL^2 + mL^2 + 2I_a & 0 \\ 0 & 0 & \frac{1}{4}mL^2 + I_a \end{bmatrix}$$

Using the above $[D_{\text{average}}]$ the following plots shows the output obtained.

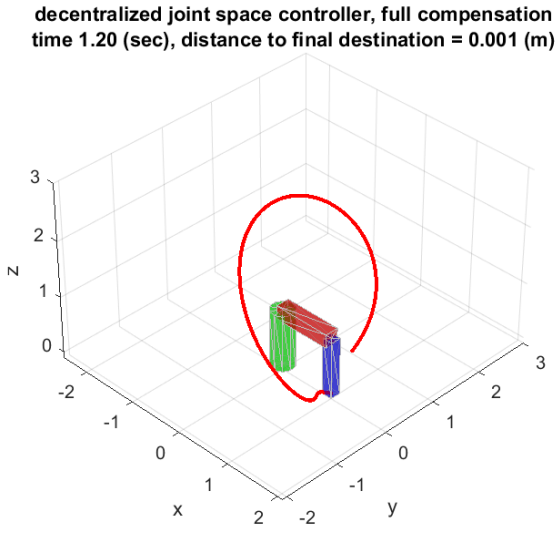


Figure 2.55: Final position of robot arm, Joint space control, part(c)

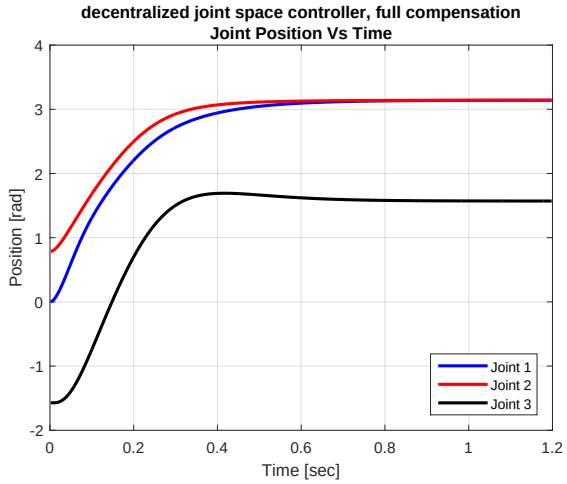


Figure 2.56: Joint position vs. time, Joint space control, part(c)

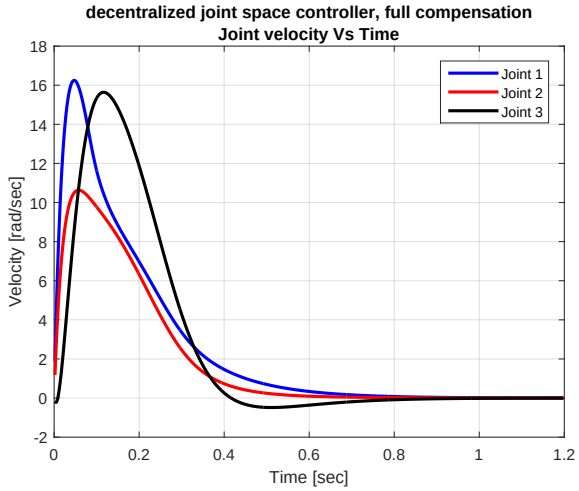


Figure 2.57: Joint velocity vs. time, Joint space control, part(c)

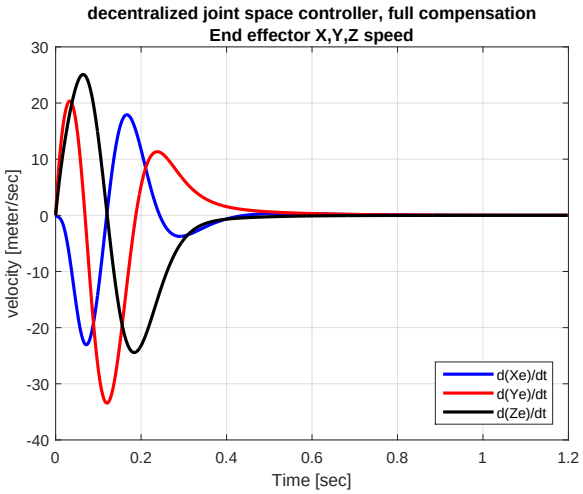


Figure 2.58: End effector $\frac{dX_e}{dt}, \frac{dY_e}{dt}, \frac{dZ_e}{dt}$ Joint space control, part(c)

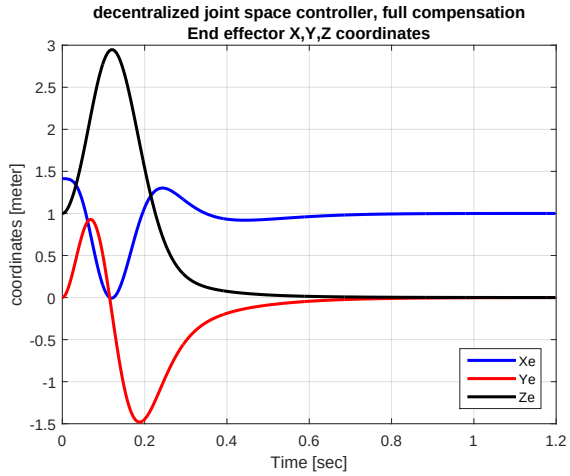


Figure 2.59: End effector X_e, Y_e, Z_e vs. time, Joint space control, part(c)

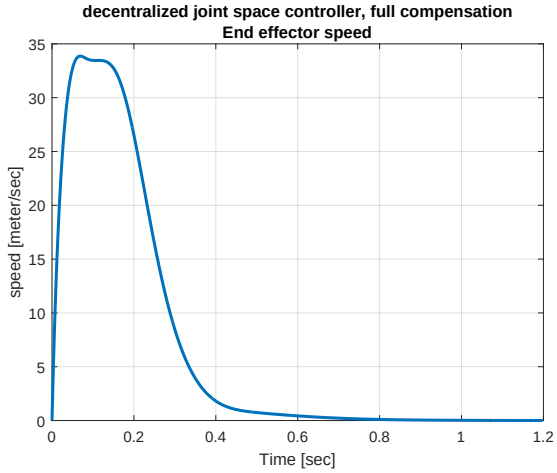


Figure 2.60: End effector linear speed vs. time, Joint space control, part(c)

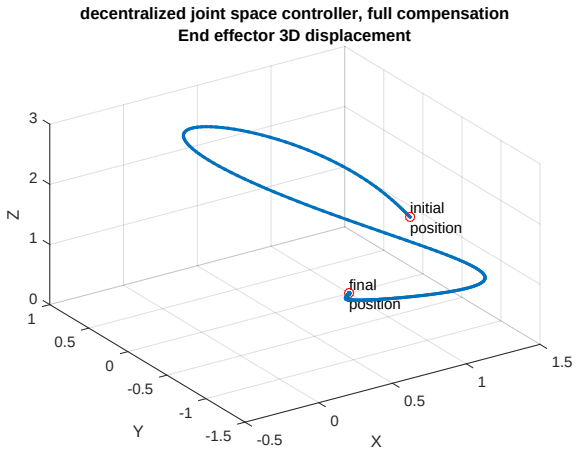


Figure 2.61: End effector plot3 displacement, Joint space control, part(c)

Part (d) discussion of result, compare control methods

There are two main methods for nonlinear dynamics decoupling. These are the computed torque method and the inverse dynamics control. In this problem the inverse dynamics control method was used. In this method, decoupling is based on measured or estimated manipulator states as described above.

The advantages and disadvantages of each controller are given below followed by a discussion on the output.

Full compensation

advantages Becuase all the non-linear terms were compensated for, this produced a smooth motion with no overshoot. In addition, the time step for the integration during simulation was not required to be too small.

disadvantages In practice, this requires measurements in real time of all the states in order to compute and estimate the current Mass, Coriolis, centrifugal and gravity matrices at each sample time. This can be expensive and require a fast CPU. Also compensating for noise and delay in measurements makes this more complicated and there will always be some error in the estimates made.

No velocity terms compensation, only gravity compensation

advantages The main advantage is that in practice this controller will be less complicated as the Coriolis and centrifugal terms do not need to be measured and computed at each time step. This will reduce the cost and make it faster to operate.

disadvantages By not compensating for velocity terms, coupling between the joints motion remains. This can be seen by the overshoot in joints motion from the desired value and the oscillation in motion, even with the use of critical damping which should produce no overshoot. This can be severe disadvantage for an end-effector which is required not to overshoot and possibly hit the target as it approaches it.

Decentralized controller with full compensation

advantages Since the mass matrix is constant, this reduces the computation in real time, as the mass matrix do not have to evaluated at each time sample as with the other controllers. This makes the controller simpler to implement.

disadvantages Since the mass matrix is the average, it is an approximation of the real mass matrix. This can produce errors. In the simulation it was found that a smaller time step was needed for the numerical integration to reduce the overshoot. Even with a much smaller time step, one joint had very small amount of overshoot. In practice this might require a small sampling time and faster CPU to implement.

The following discussion gives a review of the output of each controller.

In part(a), full decoupling was made, which means each joint motion was independent of the other joints. Comparing the joint position vs. time plot generated in part (a) shows that the joints motion was smooth and had no oscillation since it was not affected by other joints motions. There was no overshoot in the joint positions since the damping ratio is set to be critical.

While in part(b), where no velocity compensation was made (these are the centrifugal and Coriolis terms) but only gravity was compensated for, the joint motion that resulted had oscillation in it which showed as well in the speed profile which was not as smooth compared to speed profile of the full decoupling case in part (a).

In addition, there was an overshoot in the joint position which was most apparent in joint 1 motion. This can cause problems in applications where the end-effector must approach the target without hitting it.

Part(c) initially showed some small oscillation in the motion of the joints when compared to part(a). However, this turned out to be due to using a large time step for the Runge Kutta integration. By reducing the time step to smaller value than that used in part (a) and part (b), the result improved and showed no oscillation in the joint positions nor in the joint velocities.

The time step used for part(c) was 0.002 seconds, while for part(a) it was 0.005 seconds. The simulation time to converge to the final destination did not changed compared to part (a). The only change needed was to reduce the time step.

However in part (c), there was a very small overshoot in the motion of joint 3 around 0.35 second as can be seen in the plot.

The mass matrix for part (c), which is the average mass matrix D_{average}, is a constant matrix as described above, found by eliminating all the variable terms in the entries of the original mass matrix D by setting q to zero and updating the corresponding cosine and sine terms accordingly and by also setting the off-diagonal elements to zero to insure decoupling of the equations.

Of the above three controllers, the first one (part(a)) produced the best result (fast convergence to target and no oscillation in joints motion). Part(c) was similar to part(a), except for need to use much smaller integration time step. However, part (c) is the simplest controller and can make the implementation faster since the mass matrix used is not as complicated as the other two methods as it is a constant and hence no need to estimate it at run time. Therefore it is simpler method to execute and can be faster in practice.

The following diagram shows the joint position vs. time for the three controllers next to each others to make it easier to compare and contrast. Part (a) is similar to part(c), except for the small overshoot in joint 3 using part (c). Part(b) clearly did not produce good joint motion with large overshoot and large oscillations.

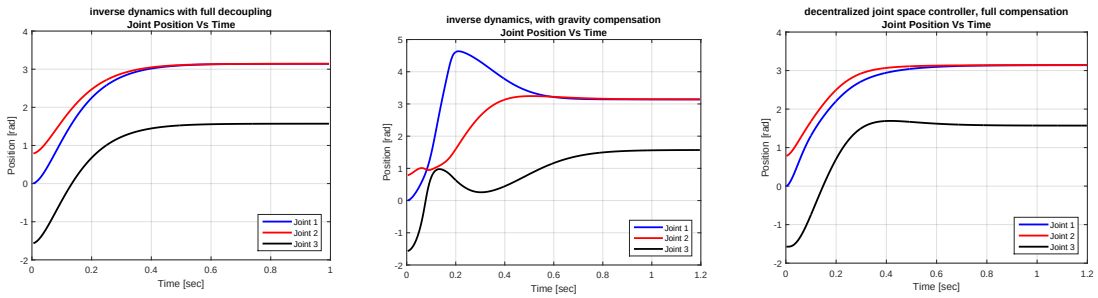


Figure 2.62: Joints positions vs. time, part(a),(b),(c) side by side

The following diagram shows the joint velocity vs. time for the three controller next to each others to make it easier to compare. Part(b) had the most oscillations. In Part(a), joint 1

had the same joint velocity as joint 2, and that is why the blue line in the plot did not show as it is below the black line. For part(c) this was not the case.

Even though each joint had smooth velocity profile, joint 1 and joint 2 did not have the same velocity profile as with the full decoupling case of part(a).

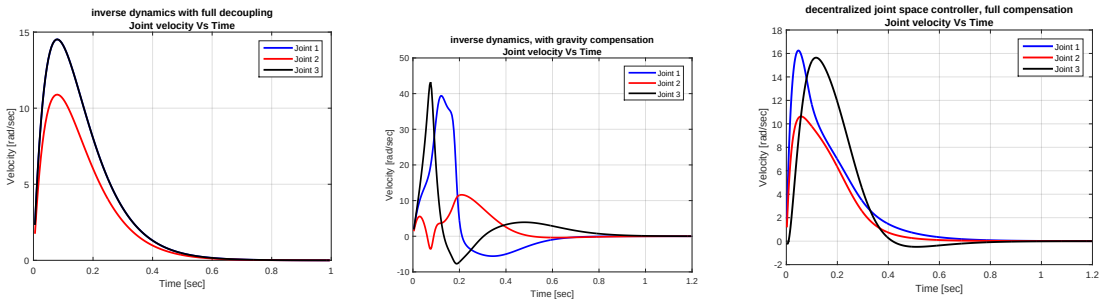


Figure 2.63: Joints velocity vs. time, part(a),(b),(c) side by side

The following diagram shows the end effector speed for the three control methods side by side. As discussed above, this was generated using $\dot{X} = J\dot{q}$ where J is the basic Jacobian for the end effector. This shows the end-effector speed profile in part (c) was similar to part (a), while Part(b) end-effector speed profile showed the effect of the coupling that remained between the joints where there was a number of places where an accelerations and deceleration showed up over the full time of the simulation. The motion was not as smooth as the other two methods.

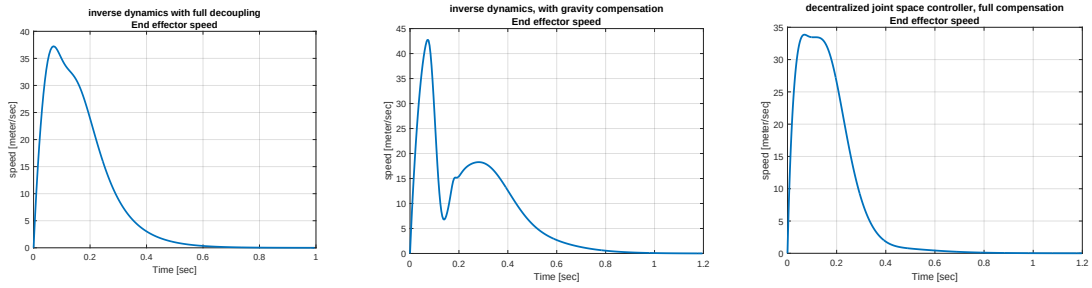


Figure 2.64: end effector speed, part(a),(b),(c) side by side

source code listing for joint-space control

The following is the Matlab source code listing which implements the joint based control part of the HW.

To run the script for this part, the command is

```
ThreeDOFcontrols
```

ThreeDOFcontrols.m

```
%file ThreeDOFcontrols.m
%This the main script used to implement HW5, part a.
%Modified original code from ME739 UW learn@UW, Madison by Professor Zinn
%
%Nasser M. Abbasi  5/10/2015

%-------------------------------------------------------------------------
%  NUMERICAL INTEGRATION OF DYNAMIC EQUATIONS
%-------------------------------------------------------------------------

%clear all;
close all; clc;

% set the model parameters per the HW problem.
modelParameters = InitializeThreeDOFmodel();

% Ask use for which part to run. There are 3 types of controllers
disp('Specify control method:')
disp('   1 = option(a), inverse dynamics with full decoupling')
disp('   2 = option(b), inverse dynamics - with gravity compensation')
disp('   3 = option(c), decentralized joint space controller')
modelParameters.controlMethod = input(' ');

%depending on the control type, set different values. The decentralized
%was found to require a smaller step size for RK-4 to behave well.
if modelParameters.controlMethod ==1
    tend    = 1;          %simulation run time
    dT      = .005;       %integration step size
    title_add_on = 'inverse dynamics with full decoupling';
elseif modelParameters.controlMethod ==2
    tend    = 1.3;
    dT      = .005;
    title_add_on = 'inverse dynamics, with gravity compensation';
else
    tend    = 1.25;
    dT      = .002;
    title_add_on = 'decentralized joint space controller, full compensation';
end;

numPts  = floor(tend/dT);

%-------------------------------------------------------------------------
%  RENDERING INITIALIZATION
%-------------------------------------------------------------------------
L           = modelParameters.L;
L1          = L;
L2          = L;
```

```
L3          = L;
f_handle    = 1;
axis_limits = L*[-2.5 2.1 -2.1 3 -.1 3];
render_view = [1 -1 1]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective

%----initialize rendering

% link 1 rendering initialization
r1         = L1/5;
sides1     = 10;
axis1      = 2;
norm_L1    = 1.0;
linkColor1 = [0 0.75 0];
plotFrame1 = 0;
d1         =  CreateLinkRendering(L1,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);

% link 2 rendering initialization
r2         = L2/6;
sides2     = 4;
axis2      = 1;
norm_L2    = 1.0;
linkColor2 = [0.75 0 0];
plotFrame2 = 0;
d2         =  CreateLinkRendering(L2,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

% link 3 rendering initialization
r3         = L2/8;
sides3     = 4;
axis3      = 1;
norm_L3    = 1.0;
linkColor3 = [0 0 0.75];
plotFrame3 = 0;
d3         =  CreateLinkRendering(L3,r3,sides3,axis3,norm_L3,linkColor3,...
    plotFrame3,f_handle);

q       = zeros(3,numPts);
dq      = zeros(3,numPts);
t       = zeros(1,numPts);
q(:,1)  = [0; pi/4; -pi/2];   %initial position
%q(:,1) = [0; 0.01; 0];   %initial position
qd(:,1) = [0; 0; 0];          %initial velocity
z       = [q(:,1); qd(:,1)];  %initialize the state variables
qDes    = [pi;pi;pi/2];       %desired final joint space position
```

```
%qDes   = [pi/3;pi/2;pi/4];       %desired final joint space position
qdDes   = [0; 0; 0];          %desired final joint velocity
zDes    = [qDes; qdDes];
Xend    = zeros(3,numPts);    %end effector coordinates
Xvend   = zeros(3,numPts);    %end effector linear velocity

% integrate equations of motion,     % Runge-Kutta 4th order
for i = 1:numPts-1
    k1 = zDot3dofControls(z,zDes,modelParameters);
    k2 = zDot3dofControls(z + 0.5*k1*dT,zDes,modelParameters);
    k3 = zDot3dofControls(z + 0.5*k2*dT,zDes,modelParameters);
    k4 = zDot3dofControls(z + k3*dT,zDes,modelParameters);
    z = z + (1/6)*(k1 + 2*k2 + 2*k3 + k4)*dT;

    % store joint position and velocity for post processing
    q(:,i+1)  = z(1:3);
    qd(:,i+1) = z(4:6);
    t(1,i+1)  = t(1,i) + dT;
end
%---------------------------------------------------------------------
%  DISPLAY INTERATION RESULTS
%---------------------------------------------------------------------

for i = 1:numPts
    q1 = q(1,i);
    q2 = q(2,i);
    q3 = q(3,i);

    % Update frame {1}
    c   = cos(q1);
    s   = sin(q1);
    L   = L1;
    T10 = [c  0   s  0
        s  0  -c  0
        0  1   0  L
        0  0   0  1];

    % Update frame {2}
    c   = cos(q2);
    s   = sin(q2);
    L   = L2;
    T21 = [c  -s  0  L*c
        s   c  0  L*s
        0   0  1   0
        0   0  0   1];

    % Update frame {3}
```

```
    c   = cos(q3);
    s   = sin(q3);
    L   = L3;
    T32 = [c  -s  0  L*c
        s   c  0  L*s
        0   0  1   0
        0   0  0   1];
    T20 = T10*T21;
    T30 = T20*T32;

    % end-effector position
    Xend(:,i) = T30(1:3,4);

    %to find end effector velocity, we use X' = J* q' where the Jacobian
    %for the end effector is found in the Three_DOF_symbolic.m script
    %as part of this HW, in the same folder as this file.

    J=zeros(3,3);
    J(1,1)=L*sin(q1)*sin(q2)*sin(q3) - L*cos(q2)*cos(q3)*sin(q1) - L*cos(q2)*sin(q1);
    J(1,2)=-cos(q1)*(L*(sin(q2) + 1) - L + L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(1,3)=-cos(q1)*(L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(2,1)=L*cos(q1)*cos(q2) + L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3);
    J(2,2)=-sin(q1)*(L*(sin(q2) + 1) - L + L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(2,3)=-sin(q1)*(L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(3,1)=0;
    J(3,2)=cos(q1)*(L*cos(q1)*cos(q2) + L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3)
    J(3,3)=cos(q1)*(L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3)) + sin(q1)*(L*cos(q

    Xvend(:,i) = J*qd(:,i);

    % update rendering
    figure(f_handle);
    UpdateLink(d1,T10);
    UpdateLink(d2,T20);
    UpdateLink(d3,T30);
    title(sprintf('%s\ntime %3.2f (sec), distance to final destination = %4.3f (m)',...
        title_add_on,i*dT,norm(Xend(:,i)-[1;0;0])));
    hold on;
    plot3(Xend(1,1:i),Xend(2,1:i),Xend(3,1:i),'r','LineWidth',2);

    if i == 1;  %pause at start of simulation rendering
        pause;
    end
    drawnow;
end

%-----------------------------------------------------------------------
```

```
%  PLOT JOINT POSITIONS
%-----------------------------------------------------------------------

figure(2);
plot(t(2:end),q(1,2:end),'b','LineWidth',2); hold on
plot(t(2:end),q(2,2:end),'r','LineWidth',2); hold on
plot(t(2:end),q(3,2:end),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'Joint Position Vs Time'));
xlabel('Time [sec]'); ylabel('Position [rad]');
grid on
legend('Joint 1','Joint 2','Joint 3','Location','southeast');

%-----------------------------------------------------------------------
%  PLOT JOINT Velocities
%-----------------------------------------------------------------------

figure(3);
plot(t(2:end),qd(1,2:end),'b','LineWidth',2); hold on
plot(t(2:end),qd(2,2:end),'r','LineWidth',2); hold on
plot(t(2:end),qd(3,2:end),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'Joint velocity Vs Time'));
xlabel('Time [sec]'); ylabel('Velocity [rad/sec]');
grid on
legend('Joint 1','Joint 2','Joint 3','Location','northeast');

%-----------------------------------------------------------------------
%  PLOT end effector X,Y,Z velocites
%-----------------------------------------------------------------------

figure(4);
plot(t,Xvend(1,:),'b','LineWidth',2); hold on
plot(t,Xvend(2,:),'r','LineWidth',2); hold on
plot(t,Xvend(3,:),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'End effector X,Y,Z speed'));
xlabel('Time [sec]'); ylabel('velocity [meter/sec]');
grid on
legend('d(Xe)/dt','d(Ye)/dt','d(Ze)/dt','Location','southeast');

%-----------------------------------------------------------------------
%  PLOT end effector X,Y,Z positions
%-----------------------------------------------------------------------

figure(5);
plot(t,Xend(1,:),'b','LineWidth',2); hold on
plot(t,Xend(2,:),'r','LineWidth',2); hold on
plot(t,Xend(3,:),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'End effector X,Y,Z coordinates'));
```

```
xlabel('Time [sec]'); ylabel('coordinates [meter]');
grid on
legend('Xe','Ye','Ze','Location','southeast');

%-----------------------------------------------------------------------
%  PLOT end effector speed (magnitude)
%-----------------------------------------------------------------------

figure(6);
plot(t, sqrt(sum(Xvend.^2,1)),'LineWidth',2);
title(sprintf('%s\n%s',title_add_on,'End effector speed'));
xlabel('Time [sec]'); ylabel('speed [meter/sec]');
grid on

%-----------------------------------------------------------------------
%  PLOT end effector displacement in 3D
%-----------------------------------------------------------------------

figure(7);
plot3(Xend(1,:),Xend(2,:),Xend(3,:),'LineWidth',2); hold on;
plot3(Xend(1,1),Xend(2,1),Xend(3,1),'ro');
text(Xend(1,1),Xend(2,1),Xend(3,1),{'initial','position'});
plot3(Xend(1,end),Xend(2,end),Xend(3,end),'ro');
text(Xend(1,end),Xend(2,end),Xend(3,end),{'final','position'});
title(sprintf('%s\n%s',title_add_on,'End effector 3D displacement'));
xlabel('X'); ylabel('Y'); zlabel('Z');
grid on
```

InitializeThreeDOFmodel.m

```
function modelParameters = InitializeThreeDOFmodel

    % set model parameters

    % gravitational constant [m/s^2]
    g = 9.81;

    % link mass [kg]
    m = 10;

    % link length [m]
    L = 1;

    % link COM location [m]
    Lc = L/2;

    % link radius [m]
    r = 0.1*L;
```

```
    % link inertia (_|_ to  link's CL) [kg/m^2]
    Ia = (1/12)*m*L^2;
    Ib = m*r^2;

    % assign values of model parameter structure
    modelParameters.g = 9.81; % gravitational constant [m/s^2]
    modelParameters.m = m;   % link mass [kg]
    modelParameters.L = L;    % link length [m]
    modelParameters.Lc = Lc;  % link COM location [m]
    modelParameters.Ia = Ia;   % inertia (_|_ to  link's CL) [kg/m^2]
    modelParameters.Ib = Ib;   % inertia (colinear to link's CL) [kg/m^2]
    modelParameters.controlMethod = 1;
end
```

zDot3dofControls.m

```
function [zDot] = zDot3dofControls(z,zDes,modelParameters)

    % assign joint displacements / velocities from state variables
    q      = z(1:3);
    qd     = z(4:end);
    qDes   = zDes(1:3);
    qdDes  = zDes(4:end);

    % mass and "average" mass matrix calculation
    [D,Davg] = Dmatrix_ThreeDOFcontrols(q,modelParameters);

    % calculate D, B, D, and G matrices
    B = Bmatrix_ThreeDOFcontrols(q,modelParameters);
    C = Cmatrix_ThreeDOFcontrols(q,modelParameters);
    V = B*[qd(1)*qd(2);qd(1)*qd(3);qd(2)*qd(3)]...
        +C*[qd(1)^2;  qd(2)^2; qd(3)^2];

    % gravity vector
    G = Gvector_ThreeDOFcontrols(q,modelParameters);

    % decoupled system PD-controller torques
    wn       = 2*2*pi; %using 2Hz per HW problem specs
    zeta     = 1;  %critical damping, per HW problem specs
    Kp       = wn^2;
    Kd       = 2*zeta*wn;
    tauPrime = Kd*(qdDes - qd) + Kp*(qDes - q);  %PD controller


    % calculate total control torques (PD control + nonlinear decoupling)
    % make up perfect estimate for simulation only
    G_estimate = G;
```

```
    V_estimate = V;
    D_estimate = D;
    if modelParameters.controlMethod == 1
         % inverse dynamics with full decoupling
         tau = D_estimate*tauPrime + G_estimate + V_estimate;
    elseif modelParameters.controlMethod == 2
        %inverse dynamics - with gravity compensation
        tau = D_estimate*tauPrime + G_estimate;
    elseif modelParameters.controlMethod == 3
        %decentralized joint space controller, full compensation but
        %use average D matrix, a constant matrix
        tau = Davg*tauPrime + G_estimate + V_estimate;
    end

    %form joint acceleration vector
    qdd = D\(tau -V - G);
    % assign state variable derivatives
    zDot = [qd; qdd];

end
```

Three_DOF_symbolic.m

```
%file Three_DOF_symbolic.m
%used for solving HW5, ME 739.
%finds the Jacobian also finds the derivative of the Jacobian,
%needed for part(b) of the HW5 problem
%
%Modifield slightly from original code from class web site.
%I changed the notation to T_0_1 instead of T_1_0, since this makes
%it more clear to me.

%Nasser M. Abbasi

clear all; clc;

MAKE_FUNCTION = 1;

syms q1 q2 q3 L1 L2 L3 Lc1 Lc2 Lc3 m1 m2 m3 g dq1 dq2 dq3
syms c1 c2 c3 s1 s2 s3
syms Ixx1 Ixx2 Ixx3 Iyy1 Iyy2 Iyy3 Izz1 Izz2 Izz3
syms Ia Ib L m

% simplifying assumptions
Ixx1 = Ia;  Iyy1 = Ib;  Izz1 = Ia;
Ixx2 = Ib;  Iyy2 = Ia;  Izz2 = Ia;
Ixx3 = Ib;  Iyy3 = Ia;  Izz3 = Ia;
```

```
L1 = L;  L2 = L;  L3 = L;
Lc1 = L/2;  Lc2 = L/2;  Lc3 = L/2;
m1 = m;  m2 = m;  m3 = m;

gVector = [0; 0; -g];
Ic1 = [Ixx1   0    0
        0   Iyy1   0
        0     0  Izz1];
Ic2 = [Ixx2   0    0
        0   Iyy2   0
        0     0  Izz2];
Ic3 = [Ixx3   0    0
        0   Iyy3   0
        0     0  Izz3];


disp('Evaluating kinematics')
% calculate homogeneous transformation matrices to the link center of mass
%   => Link 1
c = cos(q1); s = sin(q1);
T_0_1 = [c  0  s  0
         s  0 -c  0
         0  1  0  L1
         0  0  0  1];
T_0_c1 = [c  0  s   0
          s  0 -c   0
          0  1  0  Lc1
          0  0  0   1];
%   => Link 2
c = cos(q2); s = sin(q2);
T_1_2 = [c  -s  0  L2*c
         s   c  0  L2*s
         0   0  1   0
         0   0  0   1];
T_1_c2 = [c  -s  0  Lc2*c
          s   c  0  Lc2*s
          0   0  1    0
          0   0  0    1];
%   => Link 3
c = cos(q3); s = sin(q3);
T_2_3 = [c  -s  0  L3*c
         s   c  0  L3*s
         0   0  1   0
         0   0  0   1];
T_2_c3 = [c  -s  0  Lc3*c
          s   c  0  Lc3*s
          0   0  1    0
```

```
          0   0  0    1];
T_0_2 = T_0_1*T_1_2;        T_0_2 = simplify(T_0_2);
T_0_c2 = T_0_1*T_1_c2;      T_0_c2 = simplify(T_0_c2);
T_0_c3 = T_0_2*T_2_c3;      T_0_c3 = simplify(T_0_c3);
T_0_3 = T_0_2*T_2_3;

% calculate the linear and angular velocity Jacobian of each link (COM)
z0 = [0 0 1]';  z1 = T_0_1(1:3,3);  z2 = T_0_2(1:3,3);
o0 = [0 0 0]';  o1 = T_0_1(1:3,4);  o2 = T_0_2(1:3,4);
o3 = T_0_3(1:3,4);
oc1 = T_0_c1(1:3,4);  oc2 = T_0_c2(1:3,4);  oc3 = T_0_c3(1:3,4);

%find the Jacobian for end effector first. This is needed to find
%x' = J * q' for solving part (a)

Jv3 = [cross(z0,o3)  cross(z1,(o3 - o1))  cross(z2,(o3 - o2))];
Jw3 = [z0 z1 z2];
Jacobian = [Jv3;Jw3];

%now we need to find time derivative of the above Jacobian

tmp = subs(Jacobian,{q1,q2,q3},{'q1(t)','q2(t)','q3(t)'});
syms t;
der_Jacobian = diff(tmp,t);
der_Jacobian = subs(der_Jacobian,{'diff(q1(t),t)','diff(q2(t),t)','diff(q3(t),t)'},...
    {'qd(1)','qd(2)','qd(3)'});
der_Jacobian = subs(der_Jacobian,{'q1(t)','q2(t)','q3(t)'},...
    {q1,q2,q3});

der_Jacobian = subs(der_Jacobian,{'sin(q1)','sin(q2)',...
    'sin(q3)','cos(q1)','cos(q2)','cos(q3)'},...
    {s1,s2,s3,c1,c2,c3});


%find the end effector position vector using forward kinematics

Xend = T_0_3 * [0;0;0;1];
Xend = subs(Xend,{'cos(q1)','cos(q2)','cos(q3)','sin(q1)',...
    'sin(q2)','sin(q3)'},{c1,c2,c3,s1,s2,s3});

%   => Jvc1
Jvc1 = [cross(z0,(oc1 - o0))  [0; 0; 0]  [0; 0; 0]];
Jvc2 = [cross(z0,oc2)  cross(z1,(oc2 - o1))  [0; 0; 0]];
Jvc3 = [cross(z0,oc3)  cross(z1,(oc3 - o1))  cross(z2,(oc3 - o2))];
Jw1 = [z0 [0; 0; 0] [0; 0; 0]];
Jw2 = [z0 z1 [0; 0; 0]];
Jw3 = [z0 z1 z2];
```

```
% extract rotation matrices
R10 = T_0_c1(1:3,1:3);
R20 = T_0_c2(1:3,1:3);
R30 = T_0_c3(1:3,1:3);


% calculate mass matrix
disp('Evaluating mass matrix');

Dv1 =  m1*transpose(Jvc1)*Jvc1;
Dv1 = simplify(Dv1);

Dv2 =  m2*transpose(Jvc2)*Jvc2;
Dv2 = simplify(Dv2);

Dv3 =  m3*transpose(Jvc3)*Jvc3;
Dv3 = simplify(Dv3);

Dw1 = transpose(Jw1)*R10*Ic1*transpose(R10)*Jw1;
Dw1 = simplify(Dw1);

Dw2 = transpose(Jw2)*R20*Ic2*transpose(R20)*Jw2;
Dw2 = simplify(Dw2);

Dw3 = transpose(Jw3)*R30*Ic3*transpose(R30)*Jw3;
Dw3 = simplify(Dw3);

D = Dv1 + Dv2 + Dv3 + Dw1 + Dw2 + Dw3;
D = simplify(D);

% after examining solution - try to get further simplification
D = subs(D,{cos(q2)^2 - 1},{-sin(q2)^2});
D = subs(D,{cos(q2 + q3)^2 - 1},{-sin(q2 + q3)^2});

% calculate B and C maatrices
disp('Evaluating Coriolis and centrifual terms')

% form partial derivatives
for i = 1:3
    for j = 1:3
        for k = 1:3
            if k == 1
                d(i,j,k) = diff(D(i,j),q1);
            elseif k == 2
                d(i,j,k) = diff(D(i,j),q2);
            elseif k == 3
```

```
                d(i,j,k) = diff(D(i,j),q3);
            end
        end
    end
end
d = simplify(d);

% form Christofel symbols
for i = 1:3
    for j = 1:3
        for k = 1:3
            b(i,j,k) = 0.5*(d(i,j,k) + d(i,k,j) - d(j,k,i));
        end
    end
end
b = simplify(b);

% assemble C and B matrices
B = [2*b(1,1,2)  2*b(1,1,3)  2*b(1,2,3)
     2*b(2,1,2)  2*b(2,1,3)  2*b(2,2,3)
     2*b(3,1,2)  2*b(3,1,3)  2*b(3,2,3)];
C = [b(1,1,1)  b(1,2,2)  b(1,3,3)
     b(2,1,1)  b(2,2,2)  b(2,3,3)
     b(3,1,1)  b(3,2,2)  b(3,3,3)];

% form G vector
disp('Evaluating gravity vector')
G1 = -(transpose(Jvc1)*m1*gVector);
G2 = -(transpose(Jvc2)*m3*gVector);
G3 = -(transpose(Jvc3)*m3*gVector);
G = G1 + G2 + G3;
G = simplify(G);

% Auto-generate Matlab functions to evaluate D, B, D, and G matrices
if MAKE_FUNCTION == 1
    disp('Auto-generating Matlab functions');
    disp('    => generating D matrix function');
    matlabFunction(D,'file','EvaluateDmatrix');
    disp('    => generating B matrix function');
    matlabFunction(B,'file','EvaluateBmatrix');
    disp('    => generating C matrix function');
    matlabFunction(C,'file','EvaluateCmatrix');
    disp('    => generating G vector function');
    matlabFunction(G,'file','EvaluateGvector');
    disp('    => generating derivative of analytical Jacobian function');
    matlabFunction(der_Jacobian,'file','EvaluateJd');
```

```
end
```

Bmatrix_ThreeDOFcontrols.m

```
function B = Bmatrix_ThreeDOFcontrols(q,modelParameters)

    % assign model parameters to local variables
    g = modelParameters.g;
    m = modelParameters.m;
    Ia = modelParameters.Ia;
    L = modelParameters.L;
    Ib = modelParameters.Ib;

    q1 = q(1);  q2 = q(2);  q3 = q(3);
    B(1,1) = (Ib-Ia-(1/4)*m*L^2)*sin(2*q2+2*q3)+...
             (Ib-Ia- (5/4)*m*L^2)*sin(2*q2)-m*L^2*sin(2*q2+q3);
    B(1,2) = -(1/2)*(sin(q2+q3)*((4*Ia-4*Ib+m*L^2)*cos(q2+q3)+...
             2*m*L^2*cos(q2)));

    B(1,3) = 0;
    B(2,1) = 0;
    B(2,2) = 0;
    B(2,3) = -m*L^2*sin(q3);
    B(3,1) = 0;
    B(3,2) = 0;
    B(3,3) = 0;

end
```

Cmatrix_ThreeDOFcontrols.m

```
function C = Cmatrix_ThreeDOFcontrols(q,modelParameters)

    % assign model parameters to local variables
    g = modelParameters.g;
    m = modelParameters.m;
    Ia = modelParameters.Ia;
    L = modelParameters.L;
    Lc = modelParameters.Lc;
    Ib = modelParameters.Ib;

    q1 = q(1);  q2 = q(2);  q3 = q(3);
    C(1,1) = 0;
    C(1,2) = 0;
    C(1,3) = 0;
    C(2,1) = (1/2)*(Ia-Ib+(1/4)*m*L^2)*sin(2*q2+2*q3)+...
```

```
            (1/2)*(Ia-Ib+(5/4)*m*L^2)*sin(2*q2)+(1/2)*m*L^2*sin(2*q2+q3);
    C(2,2) = 0;
    C(2,3) = -(1/2)*m*L^2*sin(q3);
    C(3,1) = (1/4)*(sin(q2+q3)*((4*Ia-4*Ib+(1/4)*m*L^2)*...
              cos(q2+q3)+2*m*L^2*cos(q2)));
    C(3,2) = (1/2)*m*L^2*sin(q3);
    C(3,3) = 0; %verify with my derivation, I got something not zero

end
```

Dmatrix_ThreeDOFcontrols.m

```
function [D,Davg] = Dmatrix_ThreeDOFcontrols(q,modelParameters)

    % assign model parameters to local variables
    % model parameters
    m = modelParameters.m;
    L = modelParameters.L;
    Ia = modelParameters.Ia;
    Ib = modelParameters.Ib;

    s2   = sin(q(2));
    c2   = cos(q(2));
    c3   = cos(q(3));
    s23  = sin(q(2) + q(3));
    c23  = cos(q(2) + q(3));

    D    = zeros(3,3);
    Davg = D;

    %see ThreeDOFcontrols_symbolic.m for derivation of these
    D(1,1)= 1/4*m*L^2*c2^2+1/4*m*L^2*(c23+2*c2)^2+...
        Ib+Ia*(c2^2+c23^2)+Ib*(s2^2+s23^2);
    D(1,2) = 0;
    D(1,3) = 0;

    D(2,1) = 0;
    D(2,2) = 2*Ia + (3/2)*m*L^2 + L^2*m*c3;
    D(2,3) = 1/4*m*L^2 + 1/2*m*L^2*c3 + Ia;

    D(3,1) = 0;
    D(3,2) = D(2,3);
    D(3,3) = 1/4*m*L^2 + Ia;

    % "average" mass matrix calculation,
    % trigometric identities - assume q equals zero and set off diagonal
    % to zero
```

```
    c2 = 1.0; c3 = 1.0; c23=1; s2=0;s23=0;

    Davg(1,1)= 1/4*m*L^2*c2^2+1/4*m*L^2*(c23+2*c2)^2+...
        Ib+Ia*(c2^2+c23^2)+Ib*(s2^2+s23^2);
    Davg(1,2) = 0;
    Davg(1,3) = 0;

    Davg(2,1) = 0;
    Davg(2,2) = 2*Ia + (3/2)*m*L^2 + L^2*m*c3;
    Davg(2,3) = 0;  %  1/4*m*L^2 + 1/2*m*L^2*c3 + Ia;  Notice, set to zero

    Davg(3,1) = 0;
    Davg(3,2) = 0; %notice, set to zero
    Davg(3,3) = 1/4*m*L^2 + Ia;

end
```

Gvector_ThreeDOFcontrols.m

```
function G = Gvector_ThreeDOFcontrols(q,modelParameters)

    % assign model parameters to local variables
    g = modelParameters.g;
    m = modelParameters.m;
    L = modelParameters.L;

    q1 = q(1);  q2 = q(2);  q3 = q(3);


    G(1,1) = 0;
    G(2,1) = (1/2)*m*g*L*(3*cos(q2)+cos(q2+q3));
    G(3,1) = (1/2)*m*g*L*cos(q2+q3);


end
```

2.5.3 Operational space control

The difficult part of this implementation was finding the analytical Jacobian and its derivative with respect to time. Matlab symbolic was used for this and the file `Three_DOF_symbolic.m` contains the implementation of this.

The operational space controller using proportional derivative was implemented in the files `ThreeDOFcontrols_OP.m` and `zDot3dofControls_OP.m`. The following shows the plots generated for parts (a,b,c) followed by discussion comparing the results.

The following diagram shows the operational space controller layout taken from the class

handout, page 6-163 which shows the controller with full compensation.

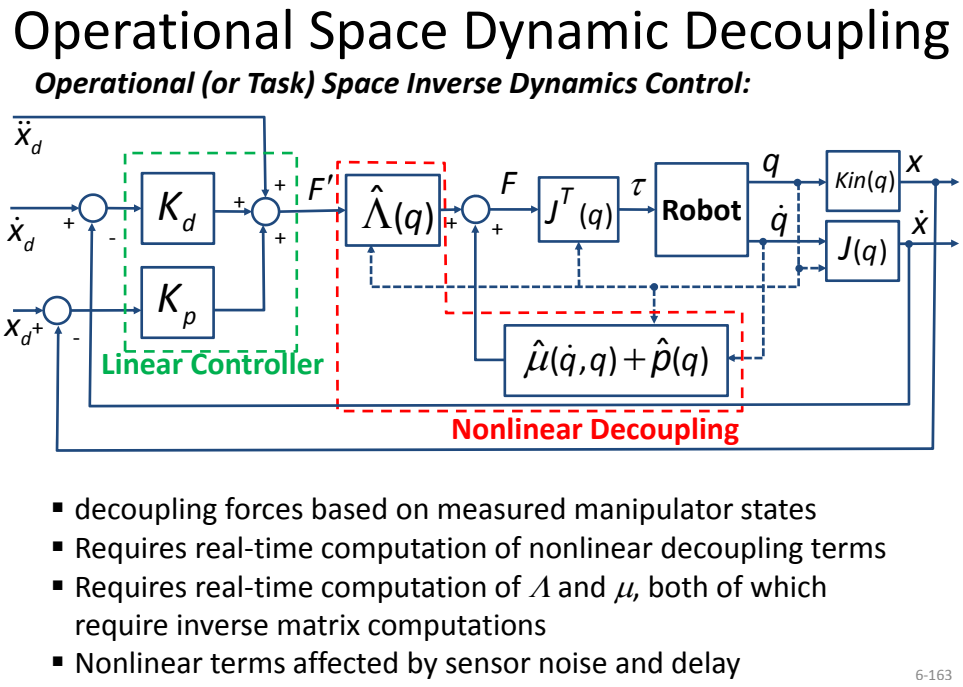


Figure 2.65: Diagram of controller, task space inverse dynamics control, full compensation. From class lecture notes

The following diagrams gives a high level overview of the Matlab software design for the implementation of the task space control. The Jacobian use in this controller is much more critical than with joint space controller due to the mapping between joint space and task space needed in the implementation.

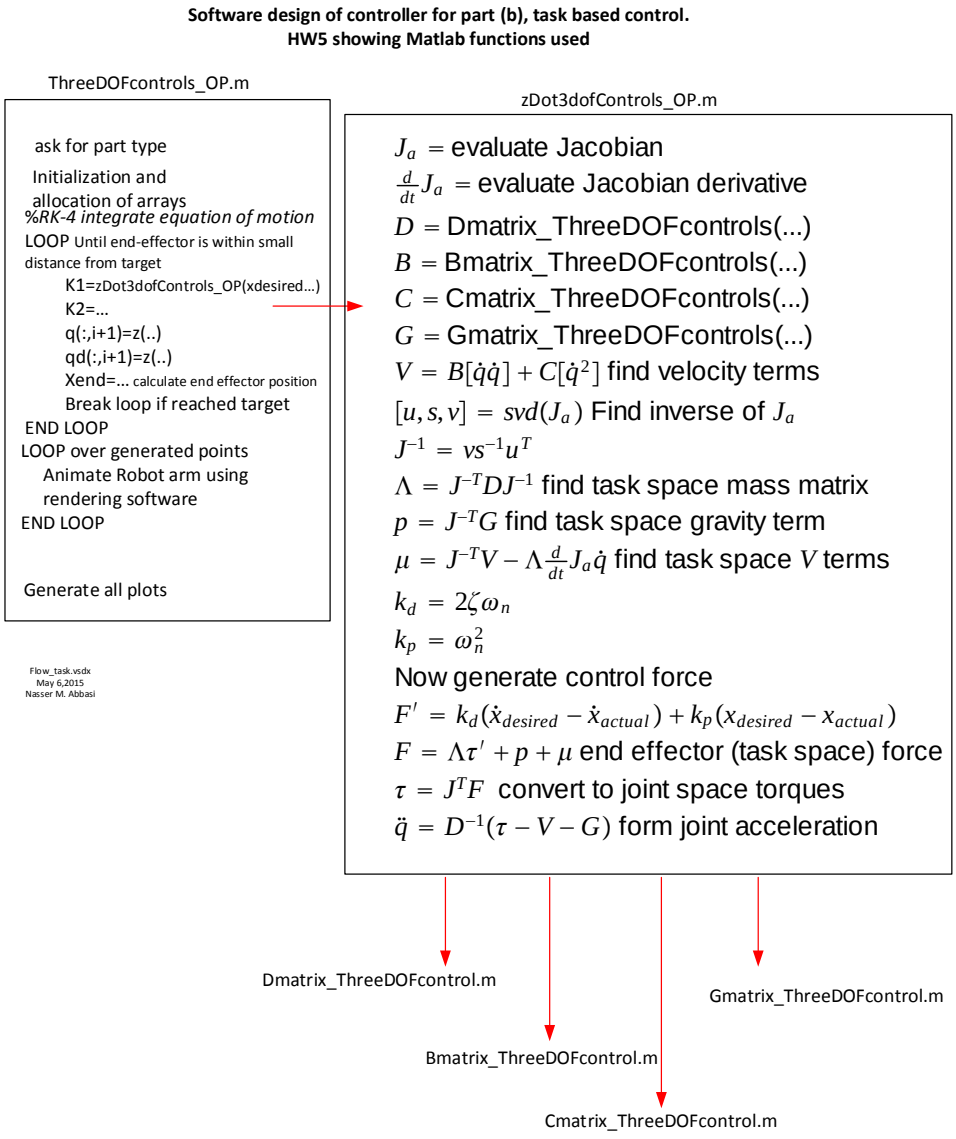


Figure 2.66: Matlab program design for task space controller

Part (a) $x_f = [-L, -L, 0]^T$

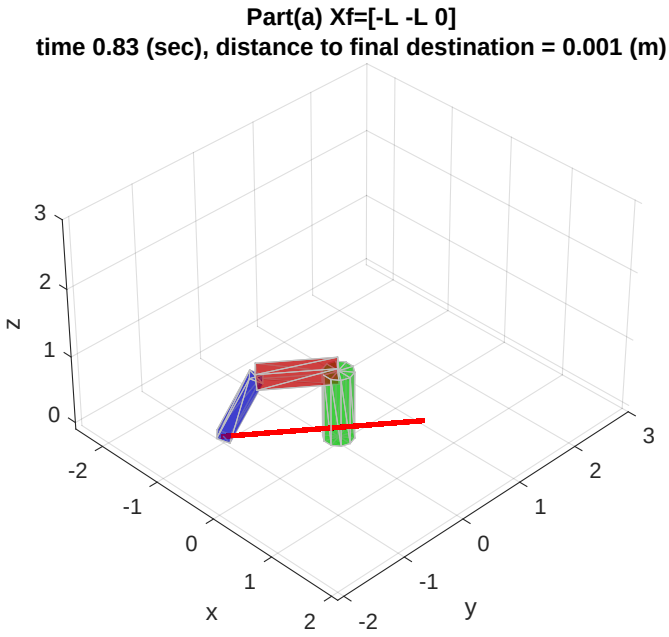


Figure 2.67: Final position of robot arm, Operational space control, part(a)

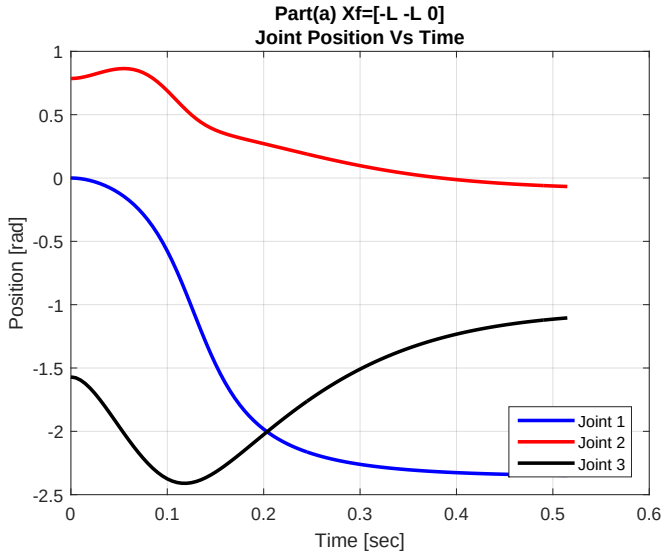


Figure 2.68: Joint position vs. time, Operational space control, part(a)

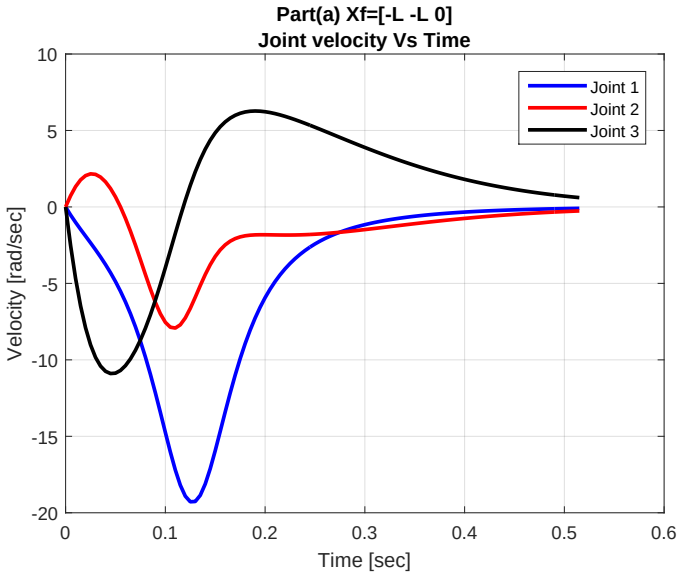


Figure 2.69: Joint velocity vs. time, Operational space control, part(a)

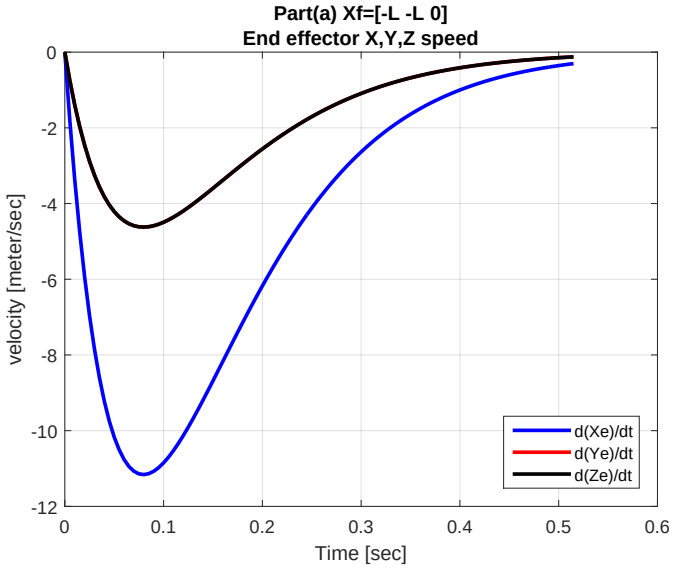


Figure 2.70: End effector $\frac{dX_e}{dt}, \frac{dY_e}{dt}, \frac{dZ_e}{dt}$ Operational space control, part(a)

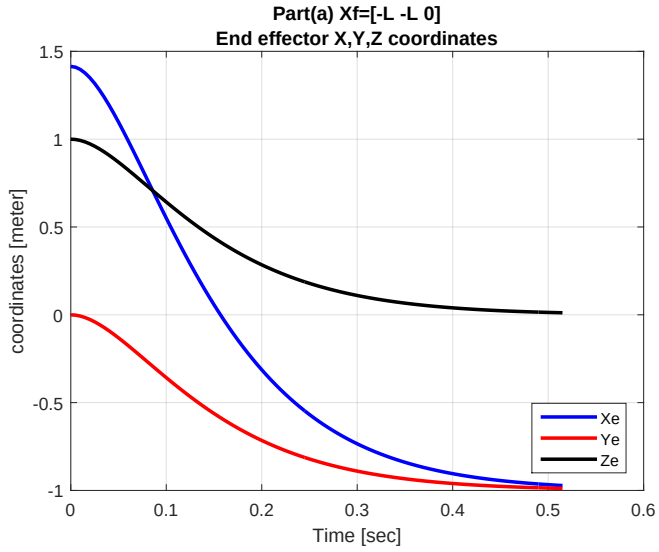


Figure 2.71: End effector X_e, Y_e, Z_e vs. time, Operational space control, part(a)

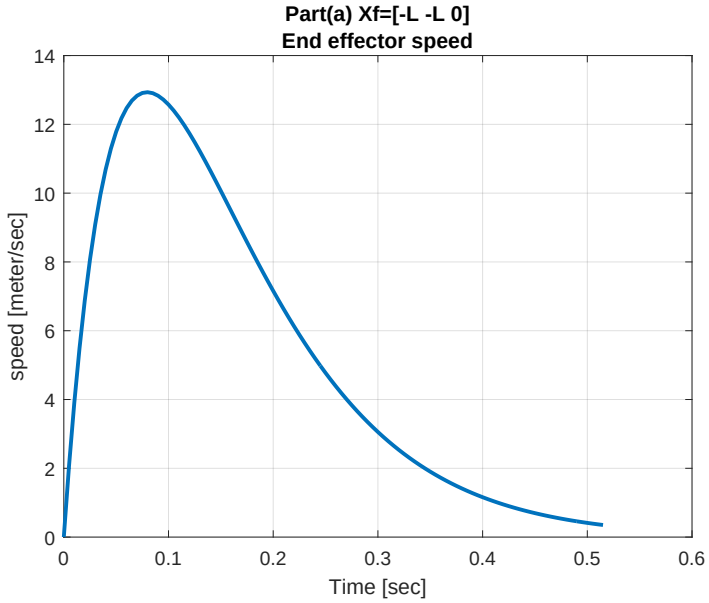


Figure 2.72: End effector linear speed vs. time, Operational space control, part(a)

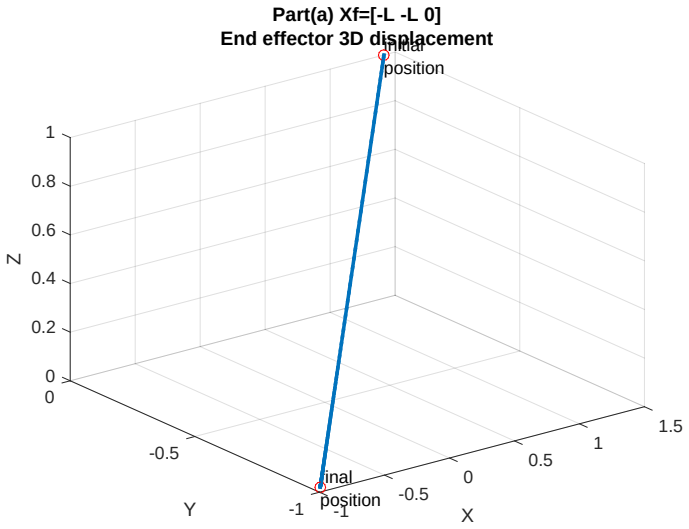


Figure 2.73: End effector plot3 displacement, Operational space control, part(a)

Part (b) $x_f = [-L, -\frac{L}{10}, 0]^T$

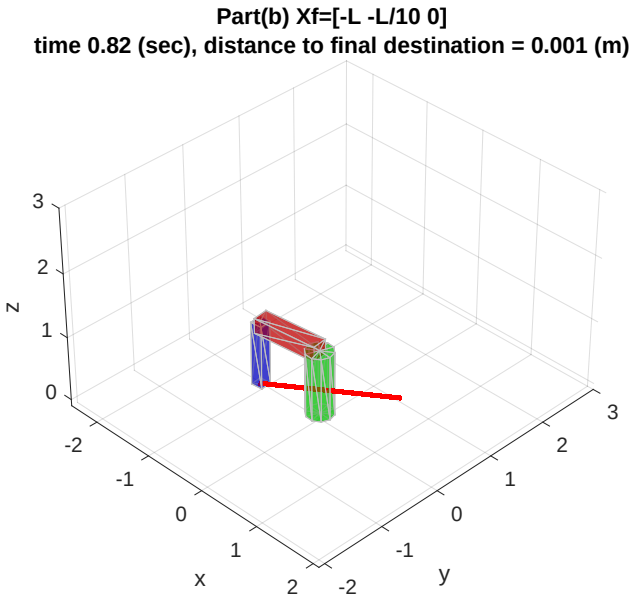


Figure 2.74: Final position of robot arm, Operational space control, part(b)

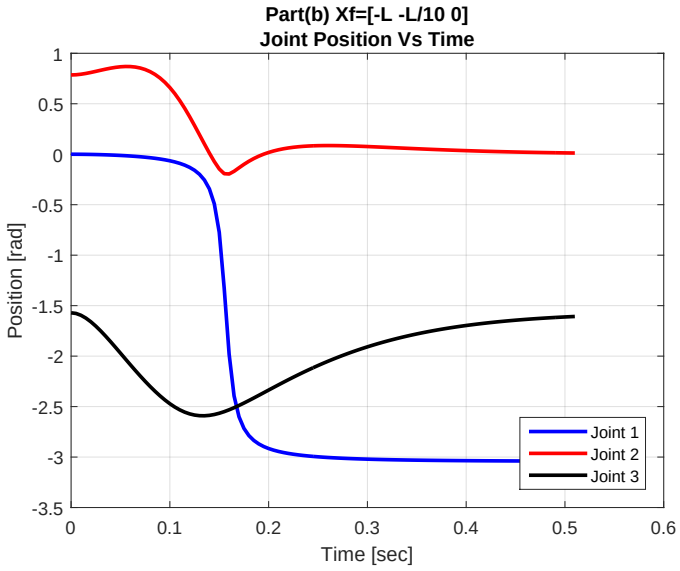


Figure 2.75: Joint position vs. time, Operational space control, part(b)

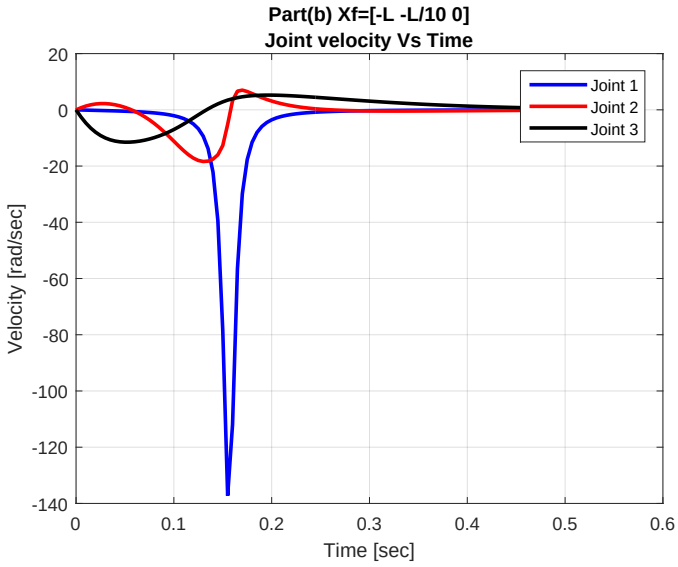


Figure 2.76: Joint velocity vs. time, Operational space control, part(b)

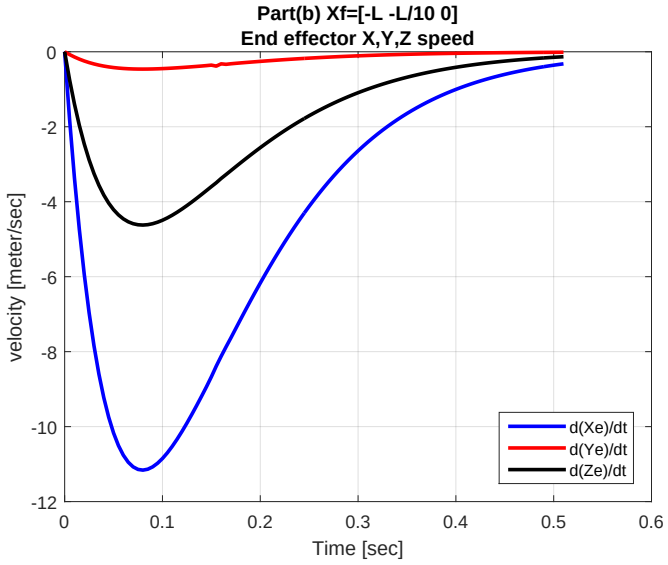


Figure 2.77: End effector $\frac{dX_e}{dt}, \frac{dY_e}{dt}, \frac{dZ_e}{dt}$ Operational space control, part(b)

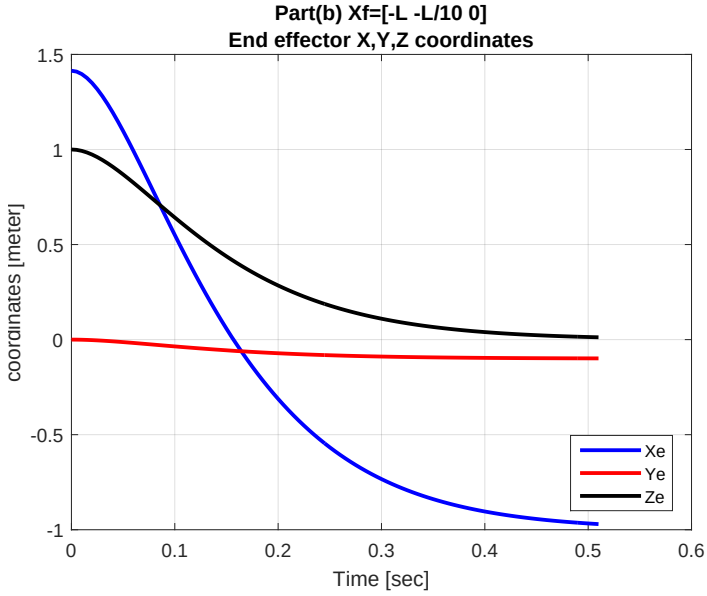


Figure 2.78: End effector X_e, Y_e, Z_e vs. time, Operational space control, part(b)

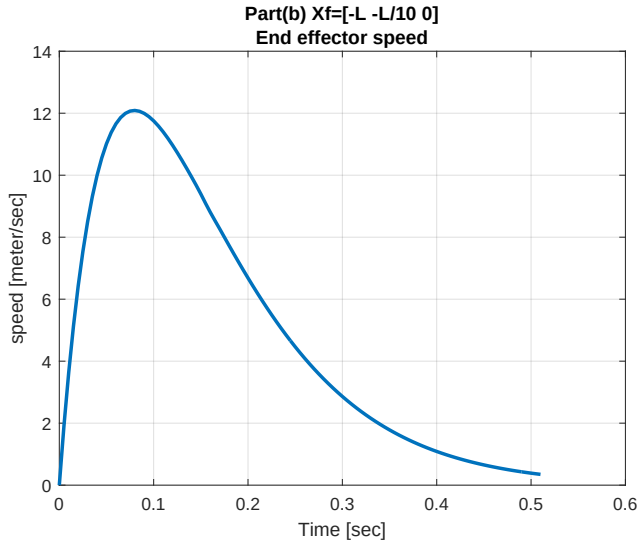


Figure 2.79: End effector linear speed vs. time, Operational space control, part(b)

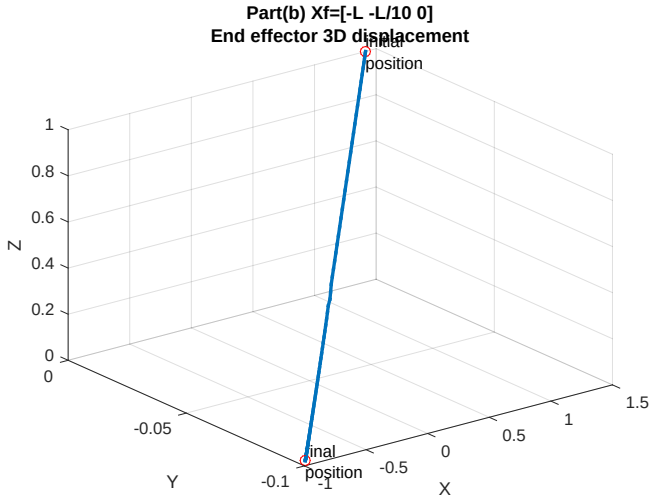


Figure 2.80: End effector plot3 displacement, Operational space control, part(b)

Part (c) $x_f = [-L, -L, 0]^T$ with velocity limiting heuristic

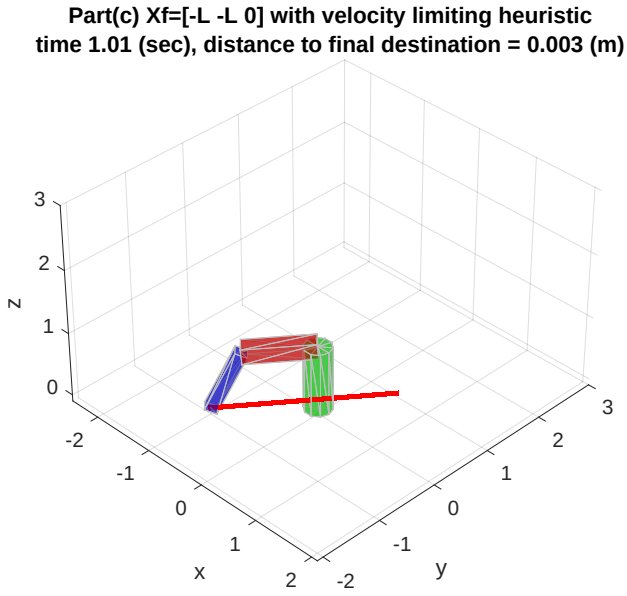


Figure 2.81: Final position of robot arm, Operational space control, part(c)

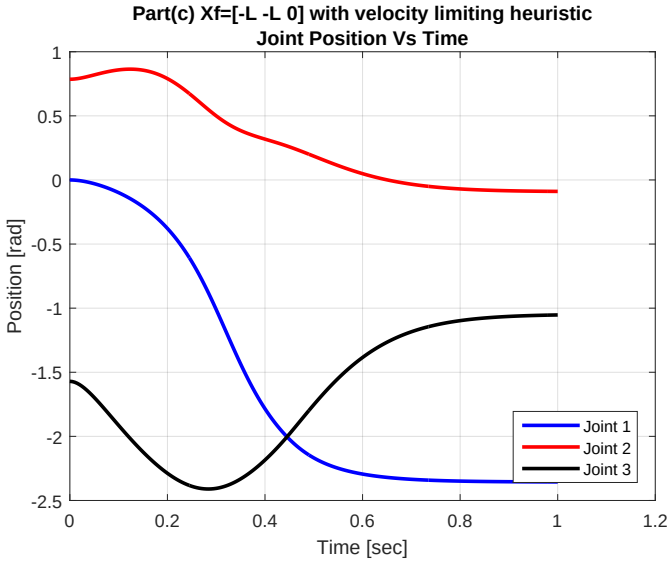


Figure 2.82: Joint position vs. time, Operational space control, part(c)

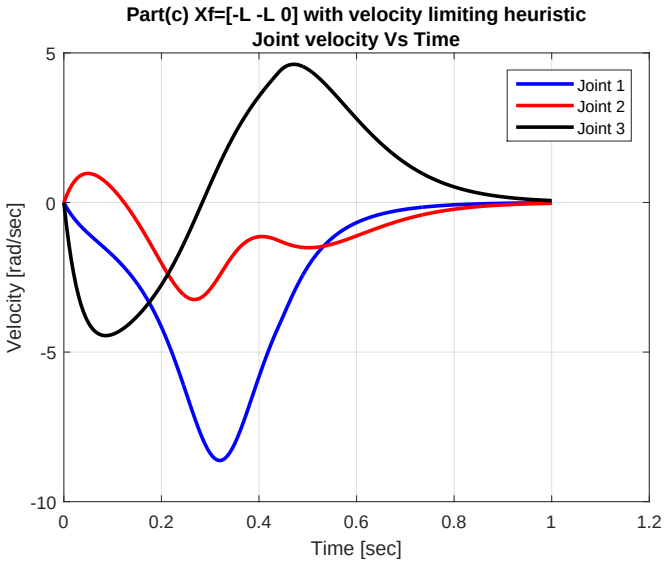


Figure 2.83: Joint velocity vs. time, Operational space control, part(c)

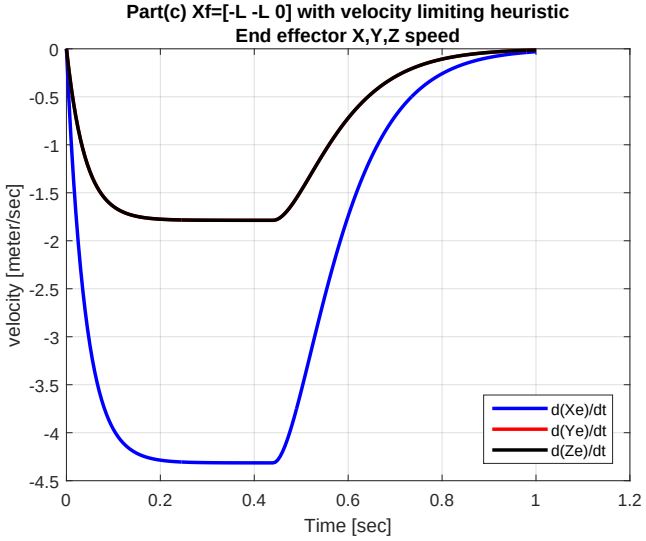


Figure 2.84: End effector $\frac{dX_e}{dt}, \frac{dY_e}{dt}, \frac{dZ_e}{dt}$ Operational space control, part(c)

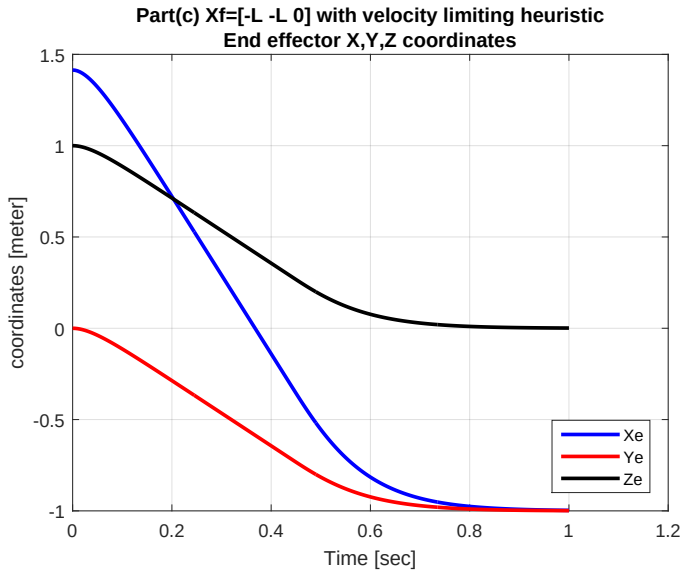


Figure 2.85: End effector X_e, Y_e, Z_e vs. time, Operational space control, part(c)

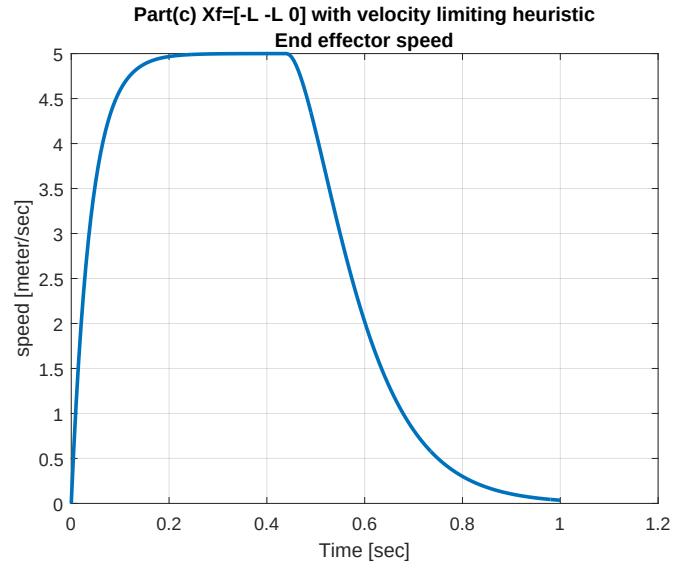


Figure 2.86: End effector linear speed vs. time, Operational space control, part(c)

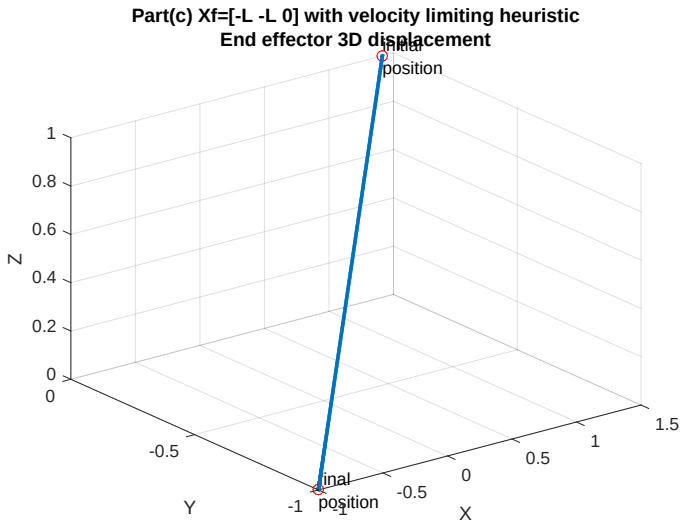


Figure 2.87: End effector plot3 displacement, Operational space control, part(c)

Part (d) discussion of result, compare control methods

In this part of the problem, the same controller was used for all parts, which is a P.D. controller with compensation for velocity terms (Coriolis and centrifugal terms) and the gravity terms.

Operational space based control is more intuitive since the target coordinates are given in operational space rather than in joint space and therefore it is easier to describe in terms of where the end-effector target should be.

One issue with Operational space is that the Jacobian can become ill-conditioned near singularities. Since the Jacobain and its derivatives are used in this control to map between joint space and operational space, this can cause a problem. This means the operational space control has to either avoid getting close to singularity region or be modified near singularities. In this simulation, no singularity was encountered.

In the three cases, the path traveled by the end effector showed a straight line from the initial position to the final position as desired. This was different from the joint space control, where the path traveled by the end effector was curved and had number of twists and turns.

The time used to reach the target for part (c) (with linear velocity limiting) was the longest, a little over one second. This is about 30% longer than the time taken by part(a) which did not have the velocity limiting heuristic and also had the same target coordinates in task space.

This is as expected, since the maximum speed the end effector can reach in part(c) was kept below 5 meter/sec. The end-effector speed profile for part(a) shows it reached maximum

speed of 12 meter/sec.

The advantage of speed limiting heuristic is that it eliminated large acceleration and deceleration of the end effector which can be important in some applications where joints speed have to be kept below some value.

In all cases, there was no overshoot in the operational space motion. But looking at joint space, case (b) showed sudden change in the joint one speed around 0.15 second.

Since control is based on operational space and not joint space, the joints positions and speeds that result can become much larger and exhibit large oscillation compared with joint based control. This is seen in part(b) where joint 1 had sudden change in speed. This can be a problem depending on the application where the joints actuators can not provide the required joint speed and will saturate.

The following plot shows side by side the joints space displacements that resulted for each part.

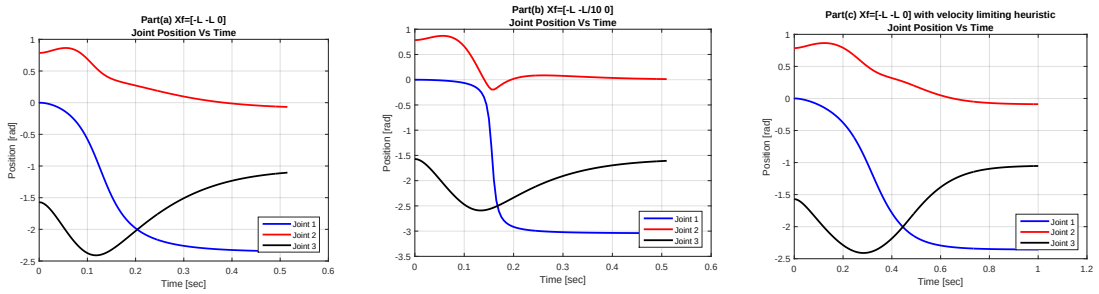


Figure 2.88: Joints positions vs. time. Operational space. part(a),(b),(c) side by side

In the above, joint 1 had sudden change in motion for part(b), this is due to the end target location. In the velocity profile below, one can see the corresponding sudden change in speed for this joint as well.

The following diagram shows the joint velocity vs. time for each case.

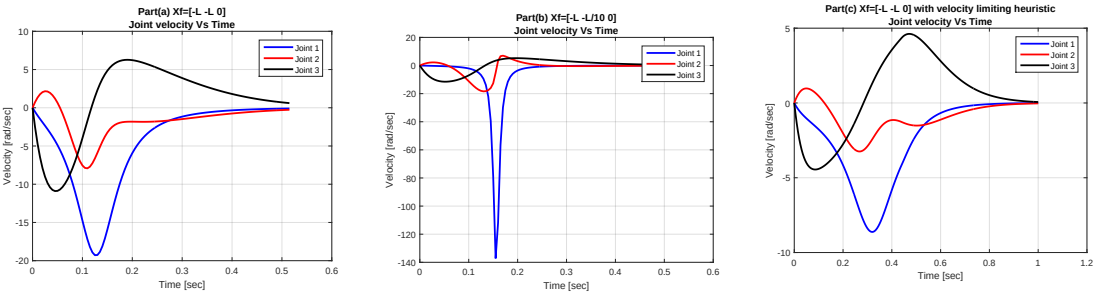


Figure 2.89: Joints velocity vs. time, operational space, part(a),(b),(c) side by side

The following diagram shows the end effector speed for the three cases side by side.

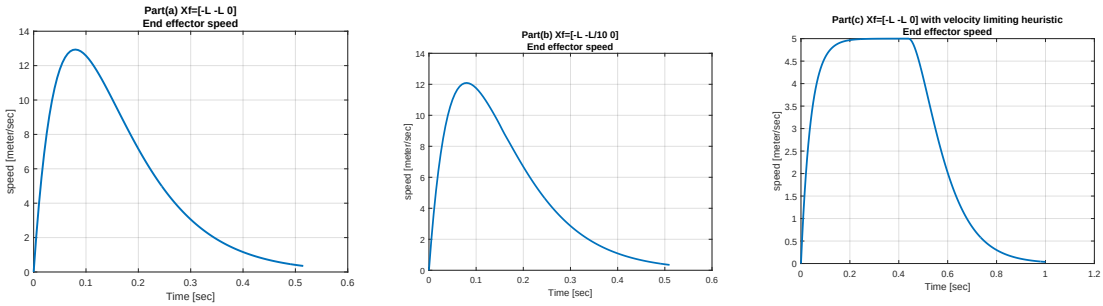


Figure 2.90: end effector speed. Operational space. part(a),(b),(c) side by side

Source code listing for operational space control

The following gives the Matlab source code listing that implements the operational space control part of the HW. Three new files were needed in addition of the files listed in part (a) above.

To run the script for this part, the command is

```
ThreeDOFcontrols_OP
```

ThreeDOFcontrols_OP.m

```
%file ThreeDOFcontrols_OP.m
%This the main script used to implement HW5, part b.
%Modified original code from ME739 UW learn@UW, Madison by Professor Zinn
%
%Nasser M. Abbasi  5/10/2015

%-------------------------------------------------------------------------
%  NUMERICAL INTEGRATION OF DYNAMIC EQUATIONS
%-------------------------------------------------------------------------

%clear all;
close all;
clc;

% set the model parameters per the HW problem.
modelParameters = InitializeThreeDOFmodel_OP();
L = modelParameters.L;

% Ask use for which part to run. There are 3 sections for this problem
disp('Specify part of problem to solve:')
disp('   1 = option(a)  Xf =[-L -L 0]')
disp('   2 = option(b)  Xf =[-L  -L/10 0]')
disp('   3 = option(c), Xf =[-L -L 0] with velocity limiting heuristic')
modelParameters.controlMethod = input('> ');
```

```
%depending on the section, set labels as needed
xdDes   = [0; 0; 0];   %desired final task space velocity
tend    = 1;     %max simulation run time, Actual time is determined below
dT      = .005;  %integration step size

switch modelParameters.controlMethod
  case 1
   title_add_on = 'Part(a) Xf=[-L -L 0]';
   xDes    = [-L;-L;0; xdDes];     %desired final task space states
  case 2
   title_add_on = 'Part(b) Xf=[-L -L/10 0]';
   xDes    = [-L;-L/10;0; xdDes];     %desired final task space states
  case 3
   title_add_on = 'Part(c) Xf=[-L -L 0] with velocity limiting heuristic';
   xDes    = [-L;-L;0; xdDes];     %desired final task space states
end;

%-------------------------------------------------------------------------
%  RENDERING INITIALIZATION
%-------------------------------------------------------------------------
L           = modelParameters.L;
L1          = L;
L2          = L;
L3          = L;
f_handle    = 1;
axis_limits = L*[-2.5 2.1 -2.1 3 -.1 3];
render_view = [1 -1 1]; view_up = [0 0 1];
SetRenderingViewParameters(axis_limits,render_view,view_up,f_handle);
camproj perspective

%----initialize rendering

% link 1 rendering initialization
r1         = L1/5;
sides1     = 10;
axis1      = 2;
norm_L1    = 1.0;
linkColor1 = [0 0.75 0];
plotFrame1 = 0;
d1         =  CreateLinkRendering(L1,r1,sides1,axis1,norm_L1,linkColor1,...
    plotFrame1,f_handle);

% link 2 rendering initialization
r2         = L2/6;
sides2     = 4;
axis2      = 1;
```

```
norm_L2    = 1.0;
linkColor2 = [0.75 0 0];
plotFrame2 = 0;
d2         =  CreateLinkRendering(L2,r2,sides2,axis2,norm_L2,linkColor2,...
    plotFrame2,f_handle);

% link 3 rendering initialization
r3         = L2/8;
sides3     = 4;
axis3      = 1;
norm_L3    = 1.0;
linkColor3 = [0 0 0.75];
plotFrame3 = 0;
d3         =  CreateLinkRendering(L3,r3,sides3,axis3,norm_L3,linkColor3,...
    plotFrame3,f_handle);

numPts  = floor(tend/dT);

q       = zeros(3,numPts);
dq      = zeros(3,numPts);
t       = zeros(1,numPts);
q(:,1)  = [0; pi/4; -pi/2];   %initial position
qd(:,1) = [0; 0; 0];          %initial velocity
z       = [q(:,1); qd(:,1)];  %initialize the state variables

% integrate equations of motion,     % Runge-Kutta 4th order
keep_running = true;
k0=1;  %counter. Loop updates this and terminates when arm reaches target

while keep_running
    k1 = zDot3dofControls_OP(z,xDes,modelParameters);
    k2 = zDot3dofControls_OP(z + 0.5*k1*dT,xDes,modelParameters);
    k3 = zDot3dofControls_OP(z + 0.5*k2*dT,xDes,modelParameters);
    k4 = zDot3dofControls_OP(z + k3*dT,xDes,modelParameters);
    z = z + (1/6)*(k1 + 2*k2 + 2*k3 + k4)*dT;

    % store joint position and velocity for post processing
    q(:,k0+1)  = z(1:3);
    qd(:,k0+1) = z(4:6);
    t(1,k0+1)  = t(1,k0) + dT;

    % check if goal position has been reached
    q1 = z(1); q2 = z(2);  q3=z(3);
    s1=sin(q1); c23=cos(q2+q3); s23= sin(q2+q3);
    c2=cos(q2); c1=cos(q1);
    s2=sin(q2); c3=cos(q3); s3=sin(q3);
```

```
    %This is end effector position vector found from forward kinematics
    %using the Three_DOF_symbolic.m symbolic computation script

    X  = [ L*c1*c2 + L*c1*c2*c3 - L*c1*s2*s3;
           L*c2*s1 - L*s1*s2*s3 + L*c2*c3*s1;
           L*(s2 + 1) + L*c2*s3 + L*c3*s2];
    xError = norm(X - xDes(1:3));
    if (xError < 1e-3 || k0 > numPts)
       keep_running = false;
    else
       k0=k0+1;
    end
end
%-----------------------------------------------------------------------
%  DISPLAY INTERATION RESULTS
%-----------------------------------------------------------------------

Xend    = zeros(3,k0);    %end effector coordinates
Xvend   = zeros(3,k0);    %end effector linear velocity

for i = 1:k0
    q1 = q(1,i);
    q2 = q(2,i);
    q3 = q(3,i);

    % Update frame {1}
    c   = cos(q1);
    s   = sin(q1);
    L   = L1;
    T10 = [c  0   s  0
        s  0  -c  0
        0  1   0  L
        0  0   0  1];

    % Update frame {2}
    c   = cos(q2);
    s   = sin(q2);
    L   = L2;
    T21 = [c  -s  0  L*c
        s   c  0  L*s
        0   0  1   0
        0   0  0   1];

    % Update frame {3}
    c   = cos(q3);
    s   = sin(q3);
    L   = L3;
```

```
    T32 = [c  -s  0  L*c
        s   c  0  L*s
        0   0  1   0
        0   0  0   1];
    T20 = T10*T21;
    T30 = T20*T32;

    % end-effector position
    Xend(:,i) = T30(1:3,4);

    %to find end effector velocity, we use X' = J* q' where the Jacobian
    %for the end effector is found in the Three_DOF_symbolic.m script
    %as part of this HW, in the same folder as this file.

    J=zeros(3,3);
    J(1,1)=L*sin(q1)*sin(q2)*sin(q3) - L*cos(q2)*cos(q3)*sin(q1) - L*cos(q2)*sin(q1);
    J(1,2)=-cos(q1)*(L*(sin(q2) + 1) - L + L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(1,3)=-cos(q1)*(L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(2,1)=L*cos(q1)*cos(q2) + L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3);
    J(2,2)=-sin(q1)*(L*(sin(q2) + 1) - L + L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(2,3)=-sin(q1)*(L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(3,1)=0;
    J(3,2)=cos(q1)*(L*cos(q1)*cos(q2) + L*cos(q1)*cos(q2)*cos(q3) - ...
        L*cos(q1)*sin(q2)*sin(q3)) + sin(q1)*(L*cos(q2)*sin(q1) + ...
        L*cos(q2)*cos(q3)*sin(q1) - L*sin(q1)*sin(q2)*sin(q3));
    J(3,3)=cos(q1)*(L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*...
        sin(q3)) + sin(q1)*(L*cos(q2)*cos(q3)*sin(q1) - ...
        L*sin(q1)*sin(q2)*sin(q3));

    Xvend(:,i) = J*qd(:,i);

    % update rendering
    figure(f_handle);
    UpdateLink(d1,T10);
    UpdateLink(d2,T20);
    UpdateLink(d3,T30);
    title(sprintf('%s\ntime %3.2f (sec), distance to final destination = %4.3f (m)',...
        title_add_on,i*dT,norm(Xend(:,i)-xDes(1:3))));
    hold on;
    plot3(Xend(1,1:i),Xend(2,1:i),Xend(3,1:i),'r','LineWidth',2);

    if i == 1;  %pause at start of simulation rendering
        pause;
    end
    drawnow;
end
```

```
%-------------------------------------------------------------------------
%  PLOT JOINT POSITIONS
%-------------------------------------------------------------------------

figure(2);
plot(t(1:k0),q(1,1:k0),'b','LineWidth',2); hold on
plot(t(1:k0),q(2,1:k0),'r','LineWidth',2); hold on
plot(t(1:k0),q(3,1:k0),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'Joint Position Vs Time'));
xlabel('Time [sec]'); ylabel('Position [rad]');
grid on
legend('Joint 1','Joint 2','Joint 3','Location','southeast');

%-------------------------------------------------------------------------
%  PLOT JOINT Velocities
%-------------------------------------------------------------------------

figure(3);
plot(t(1:k0),qd(1,1:k0),'b','LineWidth',2); hold on
plot(t(1:k0),qd(2,1:k0),'r','LineWidth',2); hold on
plot(t(1:k0),qd(3,1:k0),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'Joint velocity Vs Time'));
xlabel('Time [sec]'); ylabel('Velocity [rad/sec]');
grid on
legend('Joint 1','Joint 2','Joint 3','Location','northeast');

%-------------------------------------------------------------------------
%  PLOT end effector X,Y,Z velocites
%-------------------------------------------------------------------------

figure(4);
plot(t(1:k0),Xvend(1,:),'b','LineWidth',2); hold on
plot(t(1:k0),Xvend(2,:),'r','LineWidth',2); hold on
plot(t(1:k0),Xvend(3,:),'k','LineWidth',2); hold off
title(sprintf('%s\n%s',title_add_on,'End effector X,Y,Z speed'));
xlabel('Time [sec]'); ylabel('velocity [meter/sec]');
grid on
legend('d(Xe)/dt','d(Ye)/dt','d(Ze)/dt','Location','southeast');

%-------------------------------------------------------------------------
%  PLOT end effector X,Y,Z positions
%-------------------------------------------------------------------------

figure(5);
plot(t(1:k0),Xend(1,:),'b','LineWidth',2); hold on
plot(t(1:k0),Xend(2,:),'r','LineWidth',2); hold on
plot(t(1:k0),Xend(3,:),'k','LineWidth',2); hold off
```

```
title(sprintf('%s\n%s',title_add_on,'End effector X,Y,Z coordinates'));
xlabel('Time [sec]'); ylabel('coordinates [meter]');
grid on
legend('Xe','Ye','Ze','Location','southeast');

%-------------------------------------------------------------------------
%  PLOT end effector speed (magnitude)
%-------------------------------------------------------------------------

figure(6);
plot(t(1:k0), sqrt(sum(Xvend.^2,1)),'LineWidth',2);
title(sprintf('%s\n%s',title_add_on,'End effector speed'));
xlabel('Time [sec]'); ylabel('speed [meter/sec]');
grid on

%-------------------------------------------------------------------------
%  PLOT end effector displacement in 3D
%-------------------------------------------------------------------------

figure(7);
plot3(Xend(1,:),Xend(2,:),Xend(3,:),'LineWidth',2); hold on;
plot3(Xend(1,1),Xend(2,1),Xend(3,1),'ro');
text(Xend(1,1),Xend(2,1),Xend(3,1),{'initial','position'});
plot3(Xend(1,end),Xend(2,end),Xend(3,end),'ro');
text(Xend(1,end),Xend(2,end),Xend(3,end),{'final','position'});
title(sprintf('%s\n%s',title_add_on,'End effector 3D displacement'));
xlabel('X'); ylabel('Y'); zlabel('Z');
grid on
```

InitializeThreeDOFmodel_OP.m

```
function modelParameters = InitializeThreeDOFmodel_OP

    % set model parameters

    % gravitational constant [m/s^2]
    g = 9.81;

    % link mass [kg]
    m = 10;

    % link length [m]
    L = 1;

    % link COM location [m]
    Lc = L/2;
```

```
    % link radius [m]
    r = 0.1*L;

    % link inertia (_|_ to  link's CL) [kg/m^2]
    Ia = (1/12)*m*L^2;
    Ib = m*r^2;

    % assign values of model parameter structure
    modelParameters.g = 9.81; % gravitational constant [m/s^2]
    modelParameters.m = m;   % link mass [kg]
    modelParameters.L = L;    % link length [m]
    modelParameters.Lc = Lc;  % link COM location [m]
    modelParameters.Ia = Ia;   % inertia (_|_ to  link's CL) [kg/m^2]
    modelParameters.Ib = Ib;   % inertia (colinear to link's CL) [kg/m^2]
    modelParameters.controlMethod = 1;
    modelParameters.vMax=5; %meter/sec
end
```

zDot3dofControls_OP.m

```
function [zDot] = zDot3dofControls_OP(z,xDesired,modelParameters)

    % assign joint displacements / velocities from state variables
    q      = z(1:3);
    qd     = z(4:end);
    xDes   = xDesired(1:3);
    xdDes  = xDesired(4:end);
    L = modelParameters.L;
    % precalculate sin and cos terms
    s1=sin(q(1)); c23=cos(q(2)+q(3)); s23= sin(q(2)+q(3));
    c2=cos(q(2)); c1=cos(q(1));
    s2=sin(q(2)); c3=cos(q(3)); s3=sin(q(3));
    q1=q(1);q2=q(2);q3=q(3);
    J=zeros(3,3);
    J(1,1)=L*sin(q1)*sin(q2)*sin(q3) - L*cos(q2)*cos(q3)*sin(q1) - L*cos(q2)*sin(q1);
    J(1,2)=-cos(q1)*(L*(sin(q2) + 1) - L + L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(1,3)=-cos(q1)*(L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(2,1)=L*cos(q1)*cos(q2) + L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3);
    J(2,2)=-sin(q1)*(L*(sin(q2) + 1) - L + L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(2,3)=-sin(q1)*(L*cos(q2)*sin(q3) + L*cos(q3)*sin(q2));
    J(3,1)=0;
    J(3,2)=cos(q1)*(L*cos(q1)*cos(q2) + L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3)
    J(3,3)=cos(q1)*(L*cos(q1)*cos(q2)*cos(q3) - L*cos(q1)*sin(q2)*sin(q3)) + sin(q1)*(L*cos(q


    %the derivative of the Jacobian was derived in the Three_DOF_symbolic.m
    %file
```

```
Jd = zeros(3,3);
Jd(1,1)=L*s1*s2*qd(2) - L*c1*c2*qd(1) - L*c1*c2*c3*qd(1) + ...
    L*c1*s2*s3*qd(1) + L*c2*s1*s3*qd(2) + L*c3*s1*s2*qd(2) + ...
    L*c2*s1*s3*qd(3) + L*c3*s1*s2*qd(3);
Jd(1,2)=s1*qd(1)*(L*(s2 + 1) - L + L*c2*s3 + L*c3*s2) - ...
    c1*(L*c2*qd(2) - L*s2*s3*qd(2) - L*s2*s3*qd(3) + ...
    L*c2*c3*qd(2) + L*c2*c3*qd(3));
Jd(1,3)=c1*(L*s2*s3*qd(2) + L*s2*s3*qd(3) - L*c2*c3*qd(2) - ...
    L*c2*c3*qd(3)) + s1*qd(1)*(L*c2*s3 + L*c3*s2);
Jd(2,1)=L*s1*s2*s3*qd(1) - L*c1*s2*qd(2) - L*c2*c3*s1*qd(1) -...
    L*c1*c2*s3*qd(2) - L*c1*c3*s2*qd(2) - L*c1*c2*s3*qd(3) -...
    L*c1*c3*s2*qd(3) - L*c2*s1*qd(1);
Jd(2,2)=- s1*(L*c2*qd(2) - L*s2*s3*qd(2) - L*s2*s3*qd(3) + ...
    L*c2*c3*qd(2) + L*c2*c3*qd(3)) - c1*qd(1)*(L*(s2 + 1) -...
    L + L*c2*s3 + L*c3*s2);
Jd(2,3)=s1*(L*s2*s3*qd(2) + L*s2*s3*qd(3) - L*c2*c3*qd(2) - ...
    L*c2*c3*qd(3)) - c1*qd(1)*(L*c2*s3 + L*c3*s2);
Jd(3,1)=0;
Jd(3,2)=c1*qd(1)*(L*c2*s1 - L*s1*s2*s3 + L*c2*c3*s1) - ...
    s1*(L*s1*s2*qd(2) - L*c1*c2*qd(1) - L*c1*c2*c3*qd(1) +...
    L*c1*s2*s3*qd(1) + L*c2*s1*s3*qd(2) + L*c3*s1*s2*qd(2) +...
    L*c2*s1*s3*qd(3) + L*c3*s1*s2*qd(3)) - s1*qd(1)*(L*c1*c2 +...
    L*c1*c2*c3 - L*c1*s2*s3) - c1*(L*c2*s1*qd(1) + L*c1*s2*qd(2) +...
    L*c2*c3*s1*qd(1) + L*c1*c2*s3*qd(2) + L*c1*c3*s2*qd(2) +...
    L*c1*c2*s3*qd(3) + L*c1*c3*s2*qd(3) - L*s1*s2*s3*qd(1));
Jd(3,3)=- c1*(L*c2*c3*s1*qd(1) + L*c1*c2*s3*qd(2) + L*c1*c3*s2*qd(2)...
    + L*c1*c2*s3*qd(3) + L*c1*c3*s2*qd(3) - L*s1*s2*s3*qd(1)) - ...
    s1*(L*c1*s2*s3*qd(1) - L*c1*c2*c3*qd(1) + L*c2*s1*s3*qd(2) +...
    L*c3*s1*s2*qd(2) + L*c2*s1*s3*qd(3) + L*c3*s1*s2*qd(3)) -...
    s1*qd(1)*(L*c1*c2*c3 - L*c1*s2*s3) - c1*qd(1)*(L*s1*s2*s3 -...
    L*c2*c3*s1);

% mass matrix calculation
D = Dmatrix_ThreeDOFcontrols_OP(q,modelParameters);

% calculate D, B, D, and G matrices
B = Bmatrix_ThreeDOFcontrols(q,modelParameters);
C = Cmatrix_ThreeDOFcontrols(q,modelParameters);
V = B*[qd(1)*qd(2);qd(1)*qd(3);qd(2)*qd(3)]...
    +C*[qd(1)^2;  qd(2)^2; qd(3)^2];

% gravity vector
G = Gvector_ThreeDOFcontrols(q,modelParameters);

% evaluate inverese of the Jacobian
[u,s,v] = svd(J);
sInv = eye(size(J));
```

```
for i = 1: size(J,1)
     if s(i,i) < .01
         sInv(i,i) = 0;
     else
         sInv(i,i) = 1/s(i,i);
     end
 end
Jinv = v*sInv*u';

% evaluate operational space terms
L0  = Jinv'*D*Jinv;
p   = Jinv'*G;
mu = Jinv'*V - L0*Jd*qd;

% op space position and velocity. THis was found in Three_DOF_symbolic.m
x  = [ L*c1*c2 + L*c1*c2*c3 - L*c1*s2*s3;
       L*c2*s1 - L*s1*s2*s3 + L*c2*c3*s1;
       L*(s2 + 1) + L*c2*s3 + L*c3*s2];
xd = J*qd;

% decoupled system PD-controller torques
wn       = 2*2*pi; %using 2Hz per HW problem specs
zeta     = 1;  %critical damping, per HW problem specs
Kp       = wn^2;
Kd       = 2*zeta*wn;


% task-space velocity limiting heuristic
if modelParameters.controlMethod ==3
   vMax   = modelParameters.vMax;
   xError = norm(xDes - x);
   %xErrorMag = sqrt(xError(1)^2 + xError(2)^2);
   KpMax     = Kd*vMax/xError; %xErrorMag;
   if Kp > KpMax
      Kp = KpMax;
   end
end

Fprime = -Kd*(xd - xdDes) - Kp*(x - xDes);  %PD controller
F = L0*Fprime + p + mu;
tau = J'*F;

%form joint acceleration vector
qdd = D\(tau -V - G);
% assign state variable derivatives
zDot = [qd; qdd];
```

```
end
```