Dov Cattan and Giovanni Michel
Florida Atlantic University

COT 4930 Robotic Applications (Fall 2021)

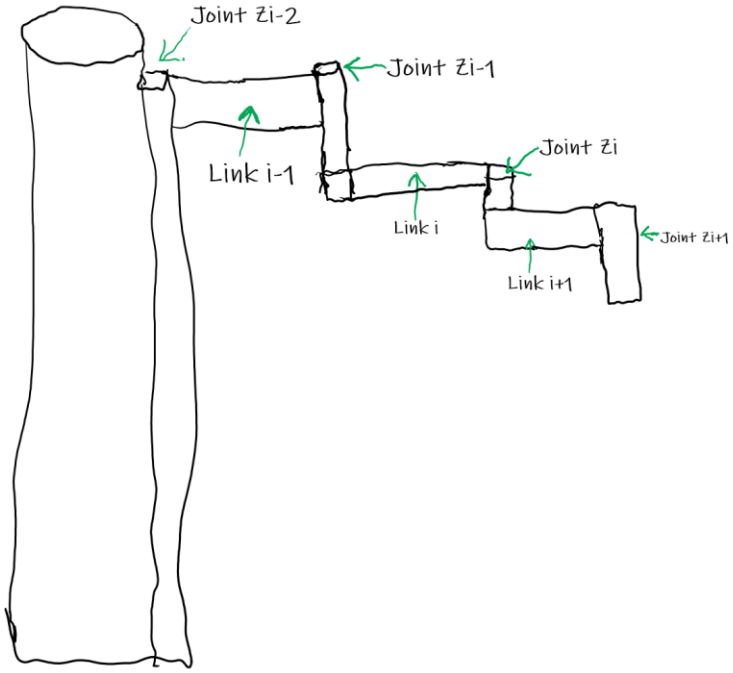
Homework 3

Problem 1: Manipulator DH Parameters and Forward Kinematic Model

Consider the 4-axis ZARM 1632 robot below.

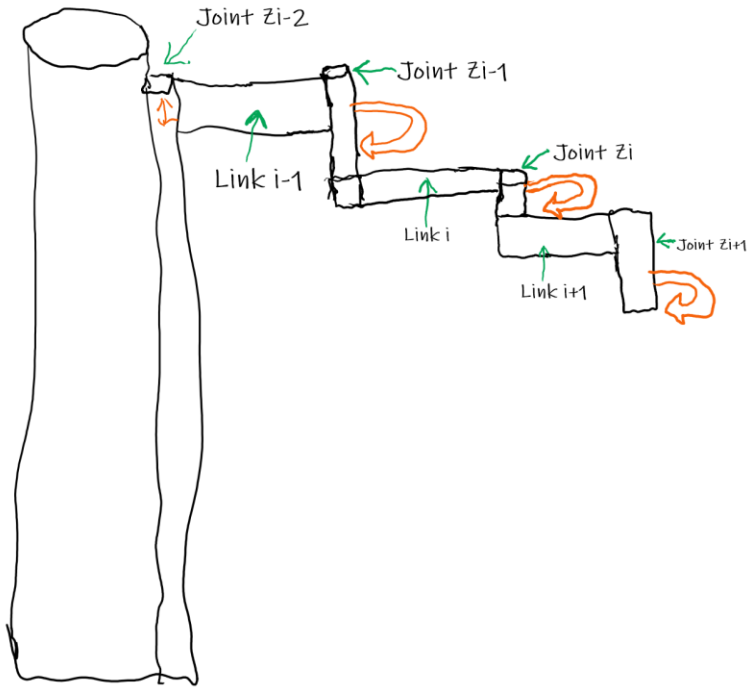
1.1. Classify the robot according to its joints type and order.

The base of the robot has an orthogonal joint as it allows the arms to move in a vertical motion up and down the base. Connected to the orthogonal joint, we have joint "Zi-2" that connects to link "i-1". On the other side of this link, we have a twisting joint which is called "Zi-1". This twisting joint connects to link "i" with a twisting joint. On the other side of link "i", we have twisting joint "Zi" which connects to link "i + 1". On the other side of link" i +1", we have another twisting joint which is called "Zi-1", which connects the robot to a tool of its choice. Please see the below diagram on the above joint parts and structure.



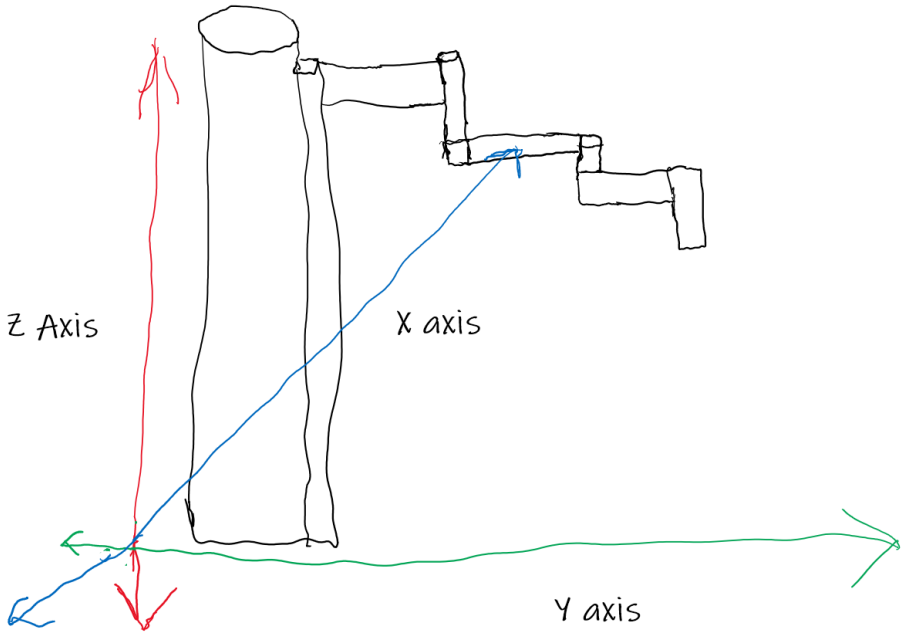
1.2. Show all the joints axes of motion (marked on the diagram above) and find the common normal lengths a_n and the distances d_n of the DH model.

D of n is considered the distance from link to another. If the joints that attach the links parallel D of n is if they prismatic i.e., long as they travel in the same axis it will a value correlated to D of n. With that said, we can see that joint Zi-1 is D of n since it travels along the Z axis using the orthogonal joint. D of n for Joints Zi-1, Zi and Zi+1 is 0 since they're parallel with the links they're attached to. For a of n it's the distance from each of the ends of the joints. When we look at the manufacturers site, we see that the length of each link is equal 160mm.



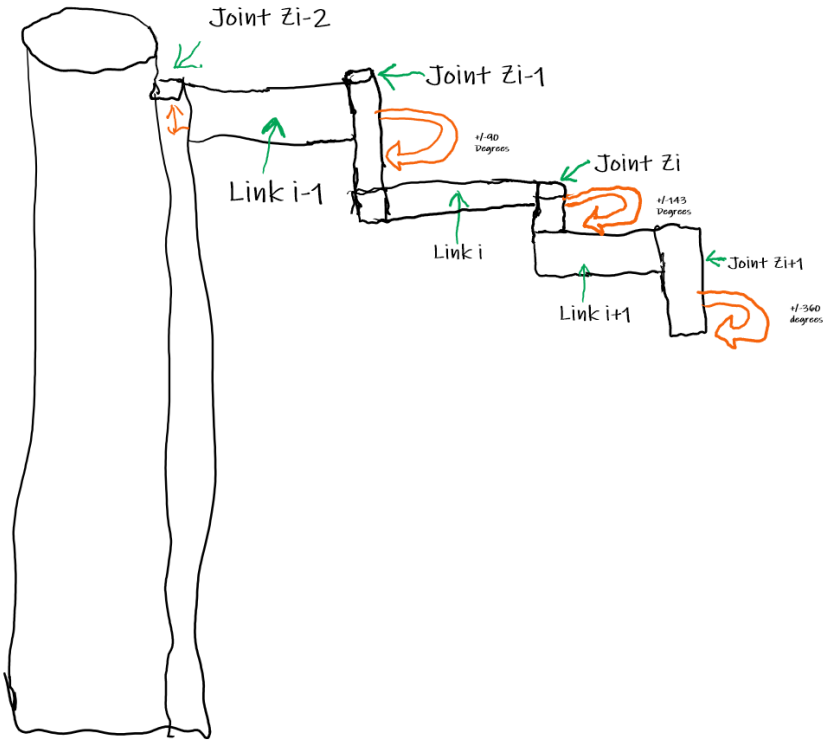
1.3. On a clean copy of the robot photo, show the links coordinate frames, and explain briefly.

We see from the base that the joint travels along Z axis. From there, links all rotate about the Z axis. Looking at this robot in comparison to a room, the Z Axis going from the floor to ceiling while the X axis is left most wall in comparison to the robot and the Y axis the wall right of it.



1.4. Find the rotation angles θ_n and the twist angles α_n and explain briefly.

Joint Zi-2 will travel up and down in a vertical motion. Joint Zi-1 will travel clockwise and counterclockwise 90 degrees around the horizontal axis. Joint Zi will travel clockwise and counterclockwise 143 degrees around the horizontal axis. Joint Zi+1 will travel clockwise and counterclockwise a full circle around the horizontal axis.



1.5. Find the A_n matrices, and then find the full kinematic model.

The A1 Matrix is the joint which is positioned between the orthogonal base and the link one. There is no twisting joint, and only a joint can travel up and down along the Z-Axis. The AZ matrix is the joint between link 1 and link 2. We kept the twist angle at Theta and the is absolute and rotates around the Z axis. Link 1 and Link 2 are both parallel from each other. A3 is the joint from link 2 to link 3 and all the information that was mentioned for A2 applies to this one as well. The same goes for A4 and this is the joint between link 3 and link 4.

Matrices:

A_n matrices:

$$A_n = \text{Rot}(z, \theta_n)\text{Trans}(0,0,d_n)\text{Trans}(a_n,0,0)\text{Rot}(x,\alpha_n)$$

$$= \begin{bmatrix} c\theta_n & -s\theta_n & 0 & 0 \\ s\theta_n & c\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_n & -s\alpha_n & 0 \\ 0 & s\alpha_n & c\alpha_n & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The twist angles α_n for all links is zero, therefore, $c\alpha_n = 1 \; and \; s\alpha_n = 0$.

$$= \begin{bmatrix} c\theta_n & -s\theta_n & 0 & 0 \\ s\theta_n & c\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_n = \begin{bmatrix} c\theta_n & -s\theta_n & 0 & a_n c\theta_n \\ s\theta_n & c\theta_n & 0 & a_n s\theta_n \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For The Full Kinematic Model:

Code:

```
%1.5
% create links using D-H parameters
% L = Link([theta, d, a, alpha]
L(1) = Link([0 100 160 0]);
L(2) = Link([0 -30 160 0]);
L(3) = Link([0 -30 132.5 0]);
L(4) = Link([0 0 0 0]);
% Joint limits
L(1).qlim = pi/180*[0 180];
L(2).qlim = pi/180*[0 150];
L(3).qlim = pi/180*[0 180];
% create the robot model
Robot = SerialLink(L);
Robot.name = 'Z-ARM 1632'
% starting position
qz = [9 4 6 8];
% ready position
qr = [0 0 0 0];
% generate a time vector
t = [0:0.056:2];
% computes the joint coordinate trajectory
q = jtraj(qz, qr, t);
% direct kinematics for each joint co-ordinate
T = Robot.fkine(q)
% plotvol9([-20,20,-20,20,0,15])
Robot.plot([0 0 0 0],'deg')
```

Output:

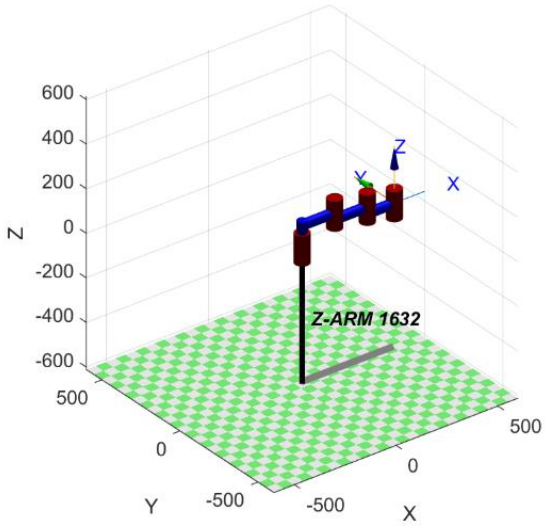
```
Robot =

Z-ARM 1632:: 4 axis, RRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|        100|        160|          0|          0|
|  2|         q2|        -30|        160|          0|          0|
|  3|         q3|        -30|      132.5|          0|          0|
|  4|         q4|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+



T(1) =
   -0.2921   -0.9564         0     130.4
    0.9564   -0.2921         0       153
         0         0         1        40
         0         0         0         1

T(2) =
   -0.2864   -0.9581         0     130.8
    0.9581   -0.2864         0     152.3
         0         0         1        40
         0         0         0         1

T(3) =
   -0.2477   -0.9688         0     133.5
    0.9688   -0.2477         0     147.8
         0         0         1        40
         0         0         0         1

T(4) =
   -0.1470   -0.9891         0     139.7
    0.9891   -0.1470         0     135.8
         0         0         1        40
         0         0         0         1

T(5) =
    0.0406   -0.9992         0     148.7
    0.9992    0.0406         0     113.3
         0         0         1        40
         0         0         0         1

T(6) =
    0.3256   -0.9455         0     156.6
    0.9455    0.3256         0     77.64
         0         0         1        40
         0         0         0         1

T(7) =
```

```
    0.6728   -0.7398         0     155.8
    0.7398    0.6728         0      28.7
         0         0         1        40
         0         0         0         1

T(8) =
    0.9543   -0.2988         0     135.4
    0.2988    0.9543         0    -28.23
         0         0         1        40
         0         0         0         1

T(9) =
    0.9385    0.3453         0     86.59
   -0.3453    0.9385         0    -78.77
         0         0         1        40
         0         0         0         1

T(10) =
    0.4222    0.9065         0     11.13
   -0.9065    0.4222         0    -99.82
         0         0         1        40
         0         0         0         1

T(11) =
   -0.4519    0.8921         0    -68.88
   -0.8921   -0.4519         0    -69.17
         0         0         1        40
         0         0         0         1

T(12) =
   -0.9967    0.0816         0    -111.8
   -0.0816   -0.9967         0     14.49
         0         0         1        40
         0         0         0         1

T(13) =
   -0.5027   -0.8645         0    -79.03
    0.8645   -0.5027         0     113.9
         0         0         1        40
         0         0         0         1

T(14) =
    0.6386   -0.7696         0     27.02
    0.7696    0.6386         0     162.6
         0         0         1        40
         0         0         0         1

T(15) =
    0.9129    0.4082         0     144.1
   -0.4082    0.9129         0     113.4
         0         0         1        40
         0         0         0         1

T(16) =
   -0.2100    0.9777         0     188.2
   -0.9777   -0.2100         0    -9.971
         0         0         1        40
```

```
         0         0         0         1

T(17) =
   -0.9988   -0.0489         0     130.4
    0.0489   -0.9988         0    -120.2
         0         0         1        40
         0         0         0         1

T(18) =
   -0.0855   -0.9963         0     29.17
    0.9963   -0.0855         0    -147.3
         0         0         1        40
         0         0         0         1

T(19) =
    0.9779   -0.2093         0     -31.7
    0.2093    0.9779         0    -109.1
         0         0         1        40
         0         0         0         1

T(20) =
    0.3386    0.9409         0    -36.36
   -0.9409    0.3386         0    -81.23
         0         0         1        40
         0         0         0         1

T(21) =
   -0.8737    0.4865         0    -45.73
   -0.4865   -0.8737         0    -100.5
         0         0         1        40
         0         0         0         1

T(22) =
   -0.6570   -0.7538         0    -113.7
    0.7538   -0.6570         0    -121.4
         0         0         1        40
         0         0         0         1

T(23) =
    0.5494   -0.8355         0    -218.1
    0.8355    0.5494         0    -72.92
         0         0         1        40
         0         0         0         1

T(24) =
    0.9736    0.2282         0      -283
   -0.2282    0.9736         0     61.19
         0         0         1        40
         0         0         0         1

T(25) =
    0.2131    0.9770         0    -251.9
   -0.9770    0.2131         0     228.5
         0         0         1        40
         0         0         0         1

T(26) =
```

```
   -0.7211    0.6928         0    -128.9
   -0.6928   -0.7211         0     357.3
         0         0         1        40
         0         0         0         1

T(27) =
   -0.9903   -0.1390         0     38.01
    0.1390   -0.9903         0       407
         0         0         1        40
         0         0         0         1

T(28) =
   -0.6044   -0.7966         0     196.3
    0.7966   -0.6044         0     380.6
         0         0         1        40
         0         0         0         1

T(29) =
    0.0070   -1.0000         0     315.5
    1.0000    0.0070         0     307.2
         0         0         1        40
         0         0         0         1

T(30) =
    0.5110   -0.8596         0     390.1
    0.8596    0.5110         0     218.7
         0         0         1        40
         0         0         0         1

T(31) =
    0.8091   -0.5876         0       429
    0.5876    0.8091         0     137.6
         0         0         1        40
         0         0         0         1

T(32) =
    0.9437   -0.3308         0     445.7
    0.3308    0.9437         0     74.93
         0         0         1        40
         0         0         0         1

T(33) =
    0.9889   -0.1484         0     451.2
    0.1484    0.9889         0     33.26
         0         0         1        40
         0         0         0         1

T(34) =
    0.9989   -0.0461         0     452.4
    0.0461    0.9989         0     10.32
         0         0         1        40
         0         0         0         1

T(35) =
    1.0000   -0.0060         0     452.5
    0.0060    1.0000         0     1.349
         0         0         1        40
```

```
     0     0     0     1

T(36) =
     1     0     0   452.5
     0     1     0     0
     0     0     1    40
     0     0     0     1
```

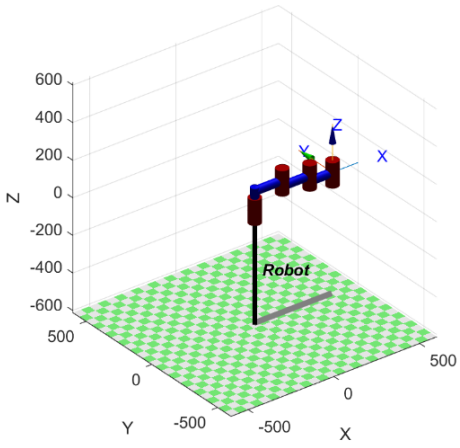


1.6. Using MATLAB Robotic Toolbox SerialLink, create a model of the robot. Plot a few robot configurations for several chosen joint position vectors.

```
%1.6
% create links using D-H parameters
% L = Link([theta, d, a, alpha]
L(1) = Link([0  100 160  0]);
L(2) = Link([0  -30  160  0]);
L(3) = Link([0  -30  132.5  0]);
L(4) = Link([0  0  0  0]);

% Joint limits
L(1).qlim = pi/180*[0 180];
L(2).qlim = pi/180*[0 150];
L(3).qlim = pi/180*[0 180];
L(4).qlim = pi/180*[0 180];

% create the robot model
Robot = SerialLink(L);
Robot.name = 'Robot'


 Robot.plot([0 0 0 0],'deg')
```

Model:

```
Robot =

Robot:: 4 axis, RRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|        100|        160|          0|          0|
|  2|         q2|        -30|        160|          0|          0|
|  3|         q3|        -30|      132.5|          0|          0|
|  4|         q4|          0|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

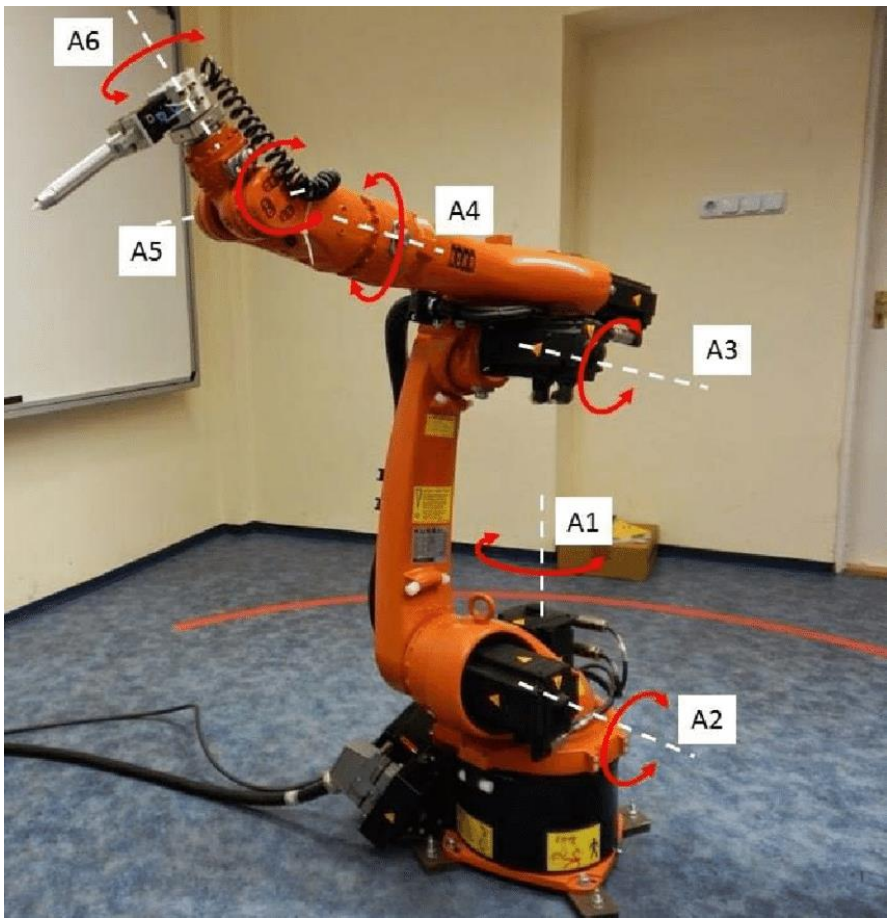


We can see in the above Serial Link model of the robot; all of the arms are facing at 0 degrees straight ahead. We have the different joints connected to the links as well as the orthogonal base which is at its highest base position.

Problem 2: Importing a Commercial Robot Model

From the robot models that are available in RTB, pick up a 6 DOF commercial robot manipulator. [Note: The PUMA 560, Stanford Arm and IRB 140 are excluded because these robots have been discussed in class].

2.1. Obtain a photo of the robot of your choice. Find its DH parameters following the 1.2-1.4 steps of Problem 1. Compare the DH parameters that you found manually to the ones listed in the model table of the robot.



Link i	Link Length (mm) – a_i	Link Twist (degrees) – a_i	Joint Offset (mm) -d_i	Joint Angle θ_i
0	180	90	400	θ_0
1	600	180	135	θ_1
2	120	-90	135	θ_2
3	0	90	620	θ_3
4	0	-90	0	θ_4
5	0	0	115	θ_5

Answer:

D_n is considered the distance from one link to another. If the joints that attach the links are parallel, the value of D_n is considered 0 while if they are prismatic, meaning that they travel along the same axis than it will be some value indicated as D_n

As for A_n it is the length from one end of the joint to the other.

Starting at the base, we can see that the orthogonal joint travels along the Z-axis. From there, the links all rotate about the Z-Axis. Looking at this robot in comparison to a room. The Z-Axis going

from the floor to the ceiling while the X-Axis is the left most wall compared to the robot and the Y-Axis is the wall to the right of the Robot.

Joint A1 will travel counterclockwise and clockwise 90 degrees in Z-Axis. Join A2, A3, A5 will travel counterclockwise and clockwise 90 degrees around the Y-Axis. Joint A4 will travel counterclockwise and clockwise 90 degrees around the X-Axis. Joint A6 will travel counterclockwise and clockwise 360 degrees around the Z-Axis

2.2. Demonstrate (with a few examples) how the "fkine" and "ikine" commands are used with the robot that you chose. Does "ikine6s" work for your robot? If so, demonstrate this command as well.

Code:

```
mdl_KR5
T = KR5.fkine(qk1)
L = KR5.fkine(qk2)
F = KR5.fkine(qk3)
K = KR5.ikine(T)
N = KR5.ikine(L)
E = KR5.ikine(F)
qi = KR5.ikine6s(T)
qi1 = KR5.ikine6s(L)
qi2 = KR5.ikine6s(F)
```

Result:

```
T =
    0.8262   -0.1643   -0.5388    0.4729
   -0.2192   -0.9749   -0.0388    0.5304
   -0.5190    0.1502   -0.8415     0.255
         0         0         0         1


L =
    0.6906   -0.6029   -0.3995    0.5862
   -0.4129   -0.7822    0.4665    0.6857
   -0.5937   -0.1572   -0.7891    0.3519
         0         0         0         1


F =
    0.1121   -0.9933   -0.0290    0.7708
   -0.8683   -0.1121    0.4833    0.5025
   -0.4833   -0.0290   -0.8750    0.3421
         0         0         0         1
K = 1×6
    0.7854    1.0472    0.7854    0.5236    0.7854    0.5236

N = 1×6
    0.7854    1.0472    0.5236    1.0472    0.7854    0.5236

E = 1×6
    0.5236    1.0472    0.5236    1.0472    0.5236    1.0472

qi = 1×6
    3.9270   -3.8176   -2.7909   -2.7788    1.4846    0.8785

qi1 = 1×6
    3.9270    2.5773   -2.4000   -2.4563    1.3156    1.2062

qi2 = 1×6
    3.6652    2.5773   -2.4000   -2.6390    1.1173    1.7937
```

2.3. For the configurations that you chose in 2.2, use MATLAB to find the Jacobian matrices of the robot, and find their rank. Can you find an example of a singular configuration?

Code:

```
mdl_KR5
Jacobian1 =KR5.jacob0(qk1)
Jacobian2 =KR5.jacob0(qk2)
Jacobian3 =KR5.jacob0(qk3)

S1 = det(Jacobian1)
S2 = det(Jacobian2)
S3 = det(Jacobian3)
R1 = rank(Jacobian1)
R2 = rank(Jacobian2)
R3 = rank(Jacobian3)
```

Results:

```
Jacobian1 = 6×6
    -0.5304    0.1025   -0.4699   -0.0220   -0.0917         0
     0.4729    0.1025   -0.4699    0.0776   -0.0342         0
    -0.0000    0.5294   -0.2294    0.0105    0.0603         0
     0.0000    0.7071   -0.7071    0.1830   -0.2709   -0.5388
     0.0000   -0.7071    0.7071    0.1830    0.9539   -0.0388
     1.0000    0.0000   -0.0000   -0.9659    0.1294   -0.8415

Jacobian2 = 6×6
    -0.6857    0.0340   -0.4014    0.0144   -0.1034         0
     0.5862    0.0340   -0.4014    0.0719   -0.0039         0
     0.0000    0.7194   -0.4194    0.0352    0.0501         0
    -0.0000    0.7071   -0.7071    0.3536    0.1768   -0.3995
     0.0000   -0.7071    0.7071    0.3536    0.8839    0.4665
     1.0000   -0.0000   -0.0000   -0.8660    0.4330   -0.7891

Jacobian3 = 6×6
    -0.5025    0.0502   -0.5002    0.0230   -0.1054         0
     0.7708    0.0290   -0.2888    0.0465    0.0388         0
    -0.0000    0.7388   -0.4388    0.0249    0.0249         0
     0.0000    0.5000   -0.5000    0.4330    0.3995   -0.0290
    -0.0000   -0.8660    0.8660    0.2500    0.8080    0.4833
     1.0000    0.0000   -0.0000   -0.8660    0.4330   -0.8750

S1 = 0.1679
S2 = 0.2264
S3 = 0.1601
R1 = 6
R2 = 6
R3 = 6
```

Problem 3: Joint Space Motion Planning

A vertical XZ planar positioner consists of two prismatic joints that move simultaneously for a variety of tool positioning tasks. The maximum velocity and acceleration of joint X are 0.8[m/s] and 2.9[m/s^2], respectively. The maximum velocity and acceleration of joint Z are 0.4[m/s] and 2.1[m/s^2], respectively. Need to plan a high-speed trapezoidal velocity profile for X and Z as the (X,Z) tool coordinates change from (4 , 1) to (-2 , 3.5) [m] (each).

3.1. Decide what should be the synchronized motion time T, whether each velocity profile be trapezoidal or triangular and what should be the acceleration/deceleration times. Also provide information about the chosen velocities and accelerations.

Answer:

Refer to the following equations for solving for the synchronized motion time, along with the acceleration and deceleration times. This will help us determine whether we should use triangular or trapezoidal inputs.

$$Given\ Vxmax = \frac{.8m}{s}, Axmax = \frac{2.1m}{s^2}, Vzmax = \frac{.4m}{s}, Azmax = \frac{2.1m}{s^2}$$

$$First\ find\ T_{acc}\ of\ both: T_{acc} = \frac{Vmax}{Amax}, taccx = .3809, taccz = .19047$$

We can observe that x has the larger T_{acc} thus we will use it to find T.

Find T_{mx} which equals $2T_{acc} = .76190$

and T_{mz} which equals $2T_{accz} = .38094$

We can now find Δq for z: $taccz \times Vzmax = .076188$

And Δq for x: $taccx \times Vxmax = .3809$

$$Voptimalz = \sqrt{Azmaz \times (\Delta q)} = .398\frac{m}{s}$$

3.2. Use MATLAB “jtraj” command to implement the needed motion.

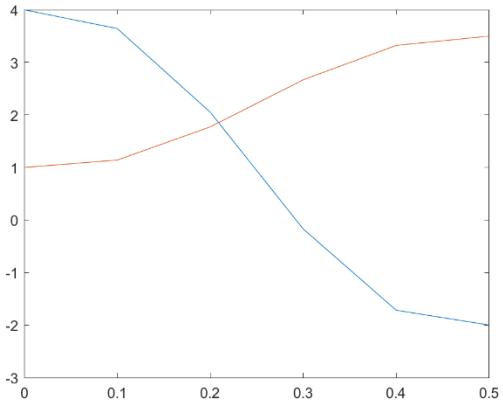
Code:

```
A = [4 1];
B = [-2 3.5]
T = 0:0.1:0.5516;
Vi =  [0 0];
Vf = [0.799 0.3986];
% Jtraj Function
[Q0,Q1] =jtraj(A,B,T,Vi,Vf);
%The X-Axis is blue while the Z-Axis is red.
plot(T,Q0);
plot(T,Q1);
```

Output:

Q0 =

4.0000	**1.0000**
3.6438	**1.1405**
2.0524	**1.7722**
-0.1713	**2.6685**
-1.7179	**3.3225**
-2.0000	**3.5000**



Q1 =

0	**0**

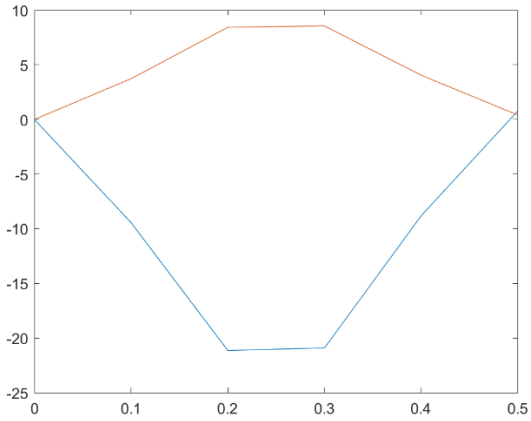
-9.4397 3.7284

-21.1451 8.4359

-20.9086 8.5539

-8.8069 4.0441

0.7990 0.3986



Q2 =

0 0

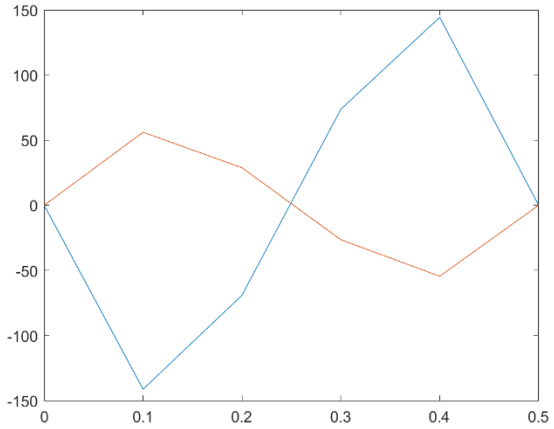
-141.3082 56.0694

-69.1200 28.8000

73.7222 -26.5041

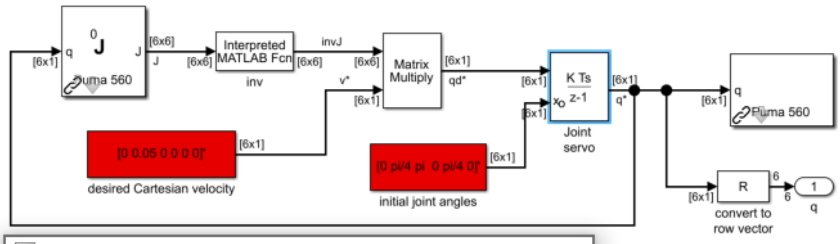
144.3763 -54.5388

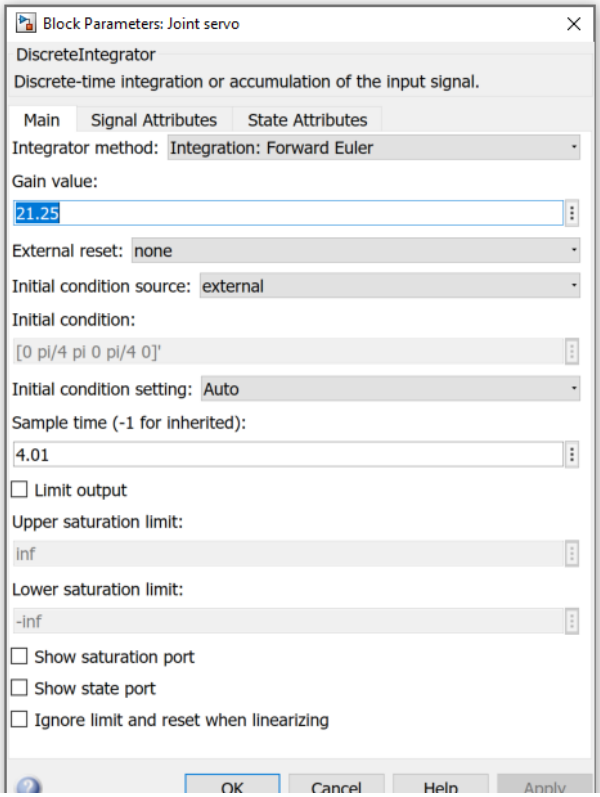
0 0.0000



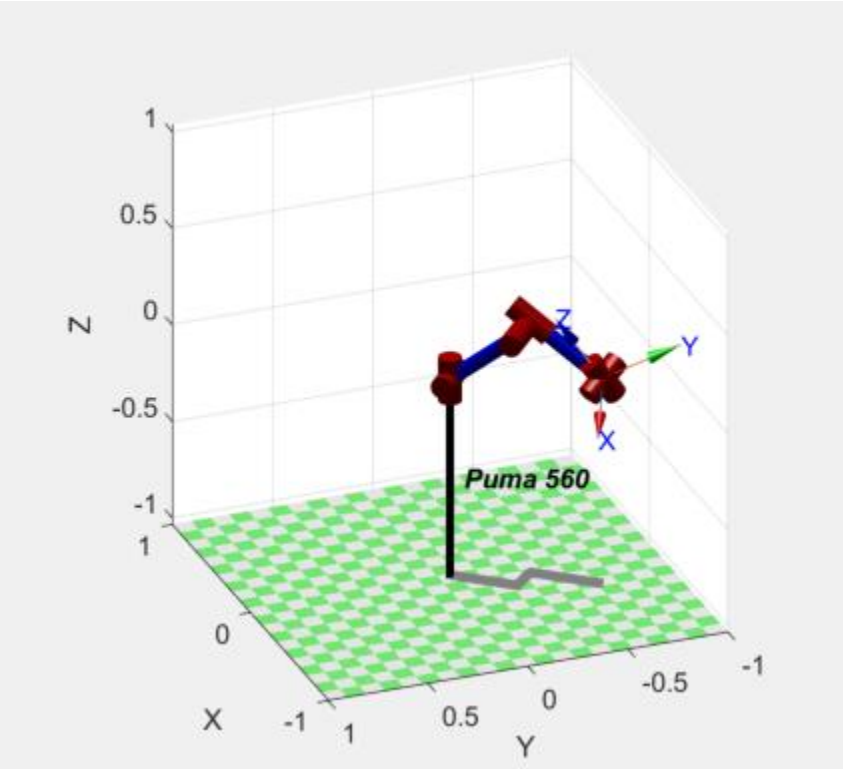
Problem 4: Resolved-Rate Motion Control

4.1. Explore the RRMC Simulink set up, for a different straight line motion, AND a different robot, AND various values of K and T. Demonstrate and explain. Are there any limitations on K and T?





Robot:



Using the Puma 560 robotic manipulator. Initial joint is set to default. The values of K = 21.25, T = 4.01. With these measurements for the gain and the time we resulted in the image above. However, Puma 560 cannot exceed the safety force limits of the robot in application use even if the robot has more power. Therefore, the values of K and T do have certain limitations, which negative values cannot be used to represent the gain and the time. Joint 0 which has a limitation of 12Nm, joint 1 of about 28 Nm, joint 2 56Nm, joint 3 150Nm, and joint 4 of 330Nm.

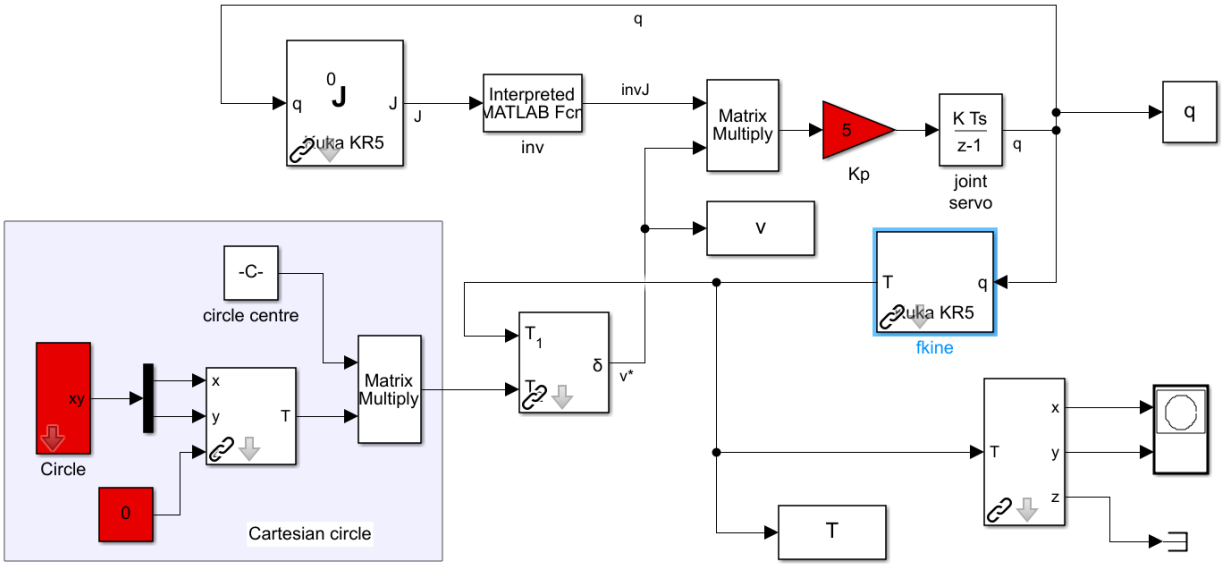
Code to Activate Model for RMMC

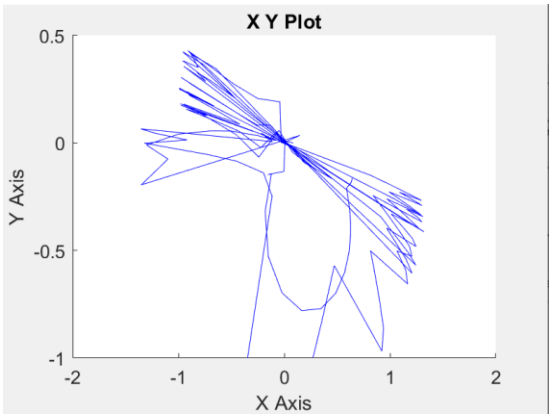
```
mdl_puma560
sl_rrmc
```

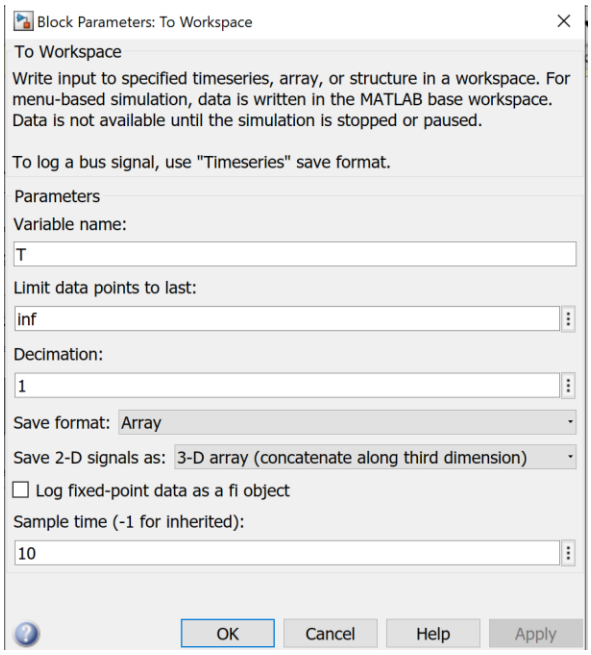
4.2. Explore the RRMC2 Simulink set up, for a robot other than the PUMA, AND a circle motion of different parameters and a variety of values for Kp and T. Demonstrate and explain. Are there any limitations on the Kp and T parameters?

Kp and T limited finite values including zero. If K is too high, it will either oscillate or miss its target. If time is too low it wont reach its gain or it will oscillate.

RRMC2 Model:







Code/Commands to Activate RRMC2 Model:

>> mdl_KR5

>> sl_rrmc2

Using the Puka KR5 model for sl_rrmc2 our values for KP = 5, T = 10. Initial joint value set to [0 pi/2 pi 0 pi/2 0] which yields the XY graph above. As far as limitations for the KR5 if K and T are too high the manipulator will miss its target or will either oscillate away from its desired position. Noting negative values of K and T, similar to 4.1 are also limitations of the KR5.

Problem 5: DC Motor Control

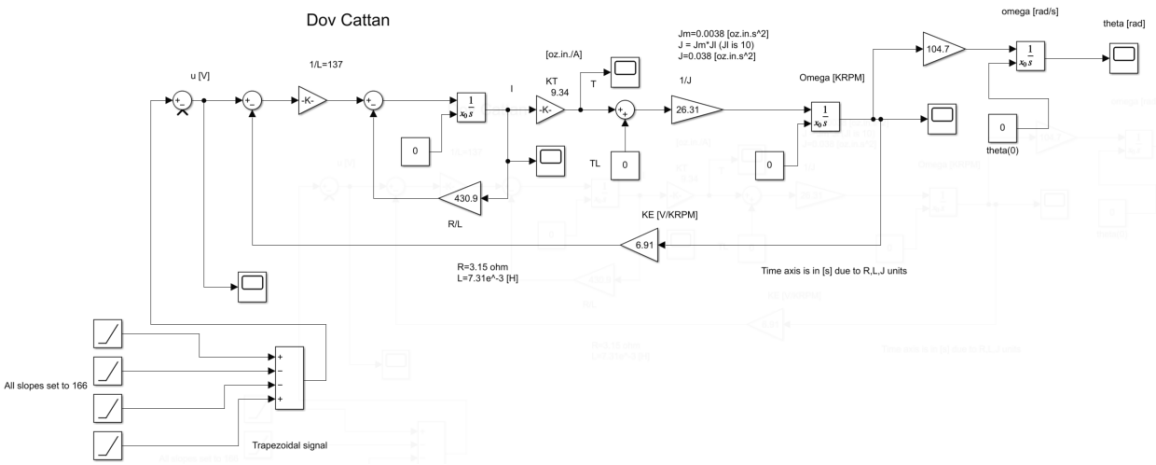
Use lectures 37 "DC Motor Simulation" and 38 "Simulation DC Motor PID Tuning" as references and use the posted Simulink models (in Canvas module "Fall 2021 Resource Material") so that you don't have to recreate the Simulink diagram, just to slightly modify existing diagrams. No need to use the motor's simplified transfer function – the provided Simulink diagrams provide exact models of the motor.

Replace the smaller motor E-508A parameters by a larger motor E-510C parameters (based on the data sheets posted in lecture 37).

In both assignments (below) **the goal is to advance the motor's angle θ from an initial angle θ(0) = 0 to a final angle $\theta(t_f) = 0.5$ [rad], in $t_f = 40$ms, or as fast as possible**. Assume that the total moment of inertia (motor plus load) is $J = 10J_m$, where J_m is listed in the data sheets. In each of the two tasks check that the motor's peak torque value and motor's maximum pulse current value are not exceeded. If needed slow the motor down a little. Also be sure that the output angle does not overshoot by more than 10%.

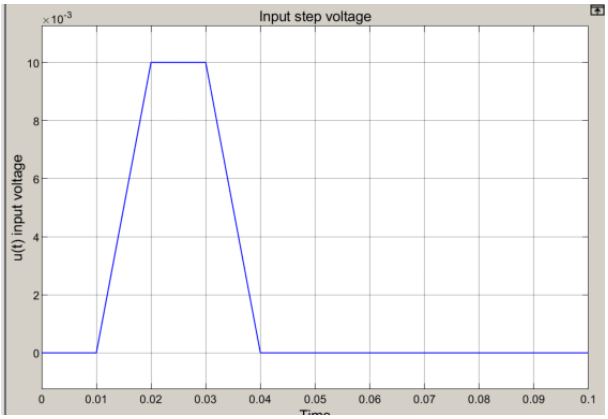
In each of the problems below, show the Simulink diagram, show all the relevant graphical results and add brief explanations.

5.1 Use the DC motor in open loop control. That is, leave the feedback of Ω via K_E, but do not create an additional unity feedback for Ω. No need to use any PID controller block. The task is to create a specific trapezoidal velocity profile input command signal, so that the motor's angle moves by the right amount by the specified timing.

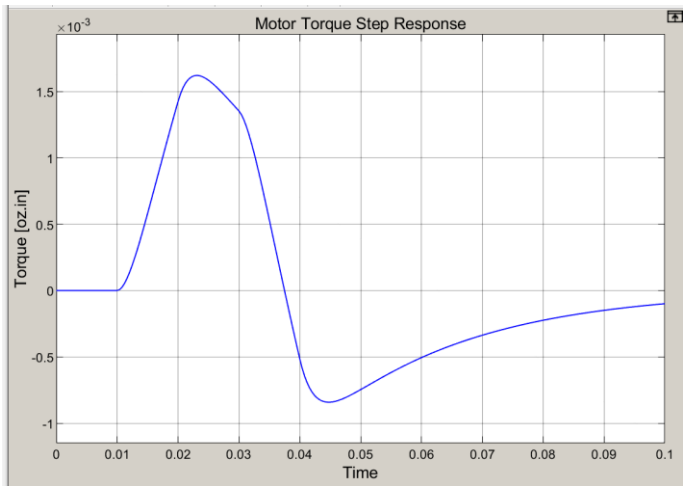


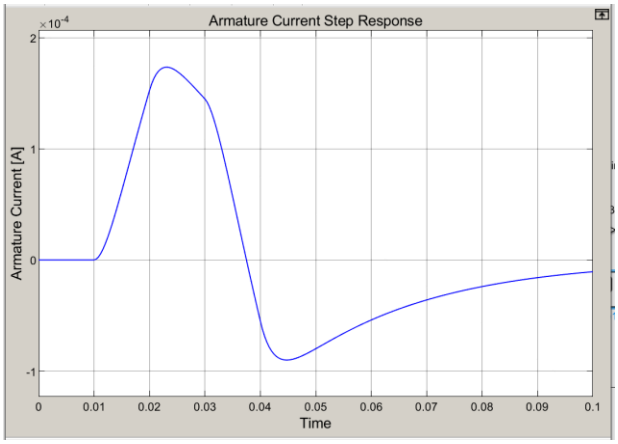
Both K's are 9.34, they both end up outputting an angular velocity prior to going to a motor shaft. The entire diagram uses the trapezoidal signal to output in a trapezoidal formation outputted by the slopes on the DC Motor, as shown below.

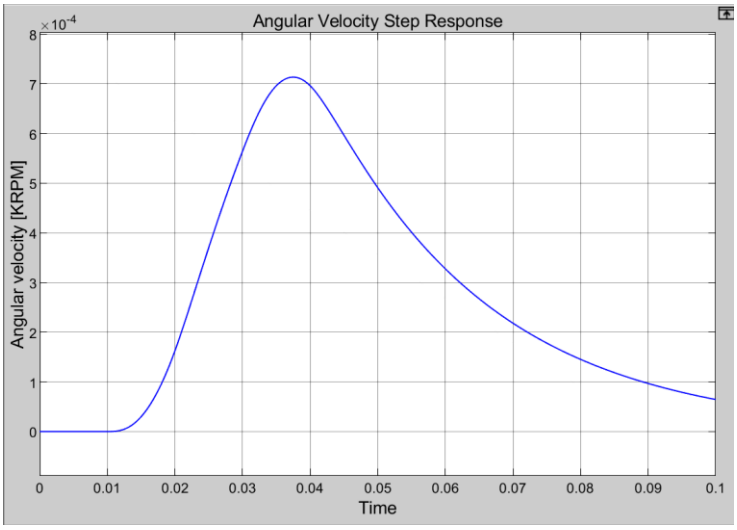
Once we had created the model, the whole point of this question was to find a trapezoidal input that moved the motor a certain angle. See the input below that moves this model .5 radians forward.



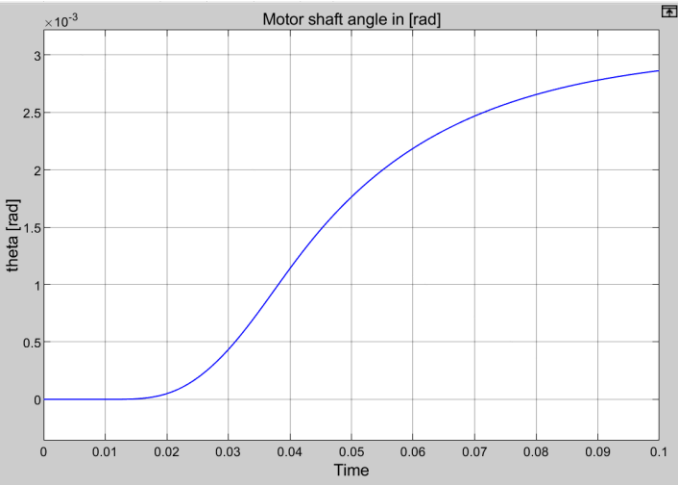
When running this motor with this voltage input we can observe the motor torque, motor velocity, and current through the motor. Notice how the peaks of all three of these graphs are below spec limits given for the E-510C, meaning this trapezoidal input will not break the motor.





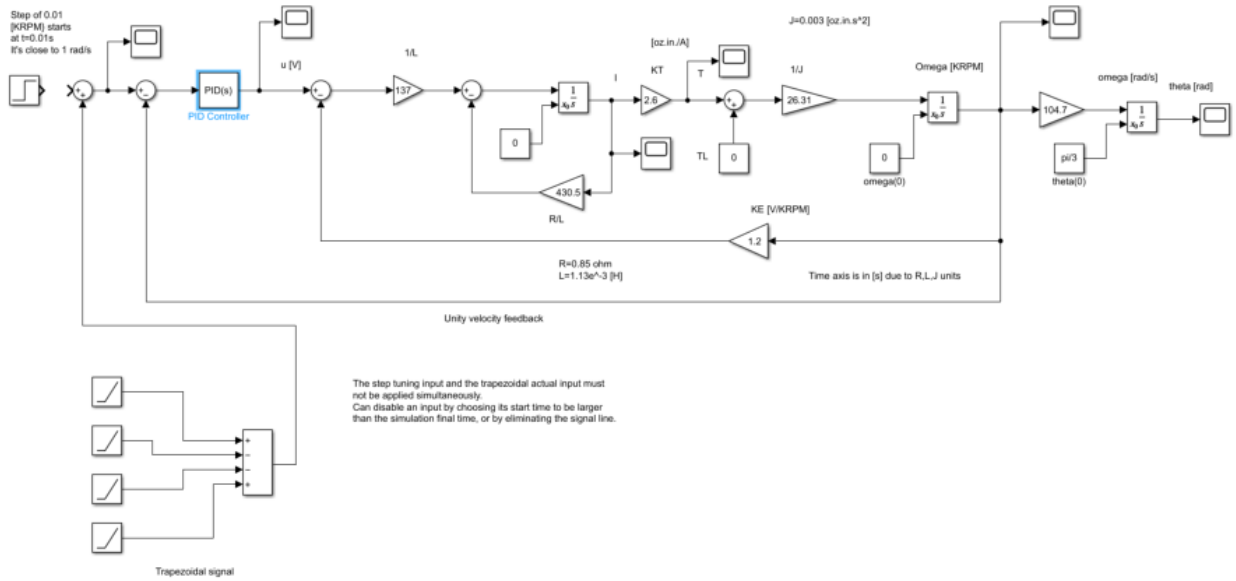


Lastly, we can observe the angle in radians that this trapezoidal input turns the motor, which is the goal of this question. Notice how in the graph below, once it turns it's .5 radians the motor stops turning. This time when the motor stops turning also lines up with angular velocity, torque and current going to 0 above.

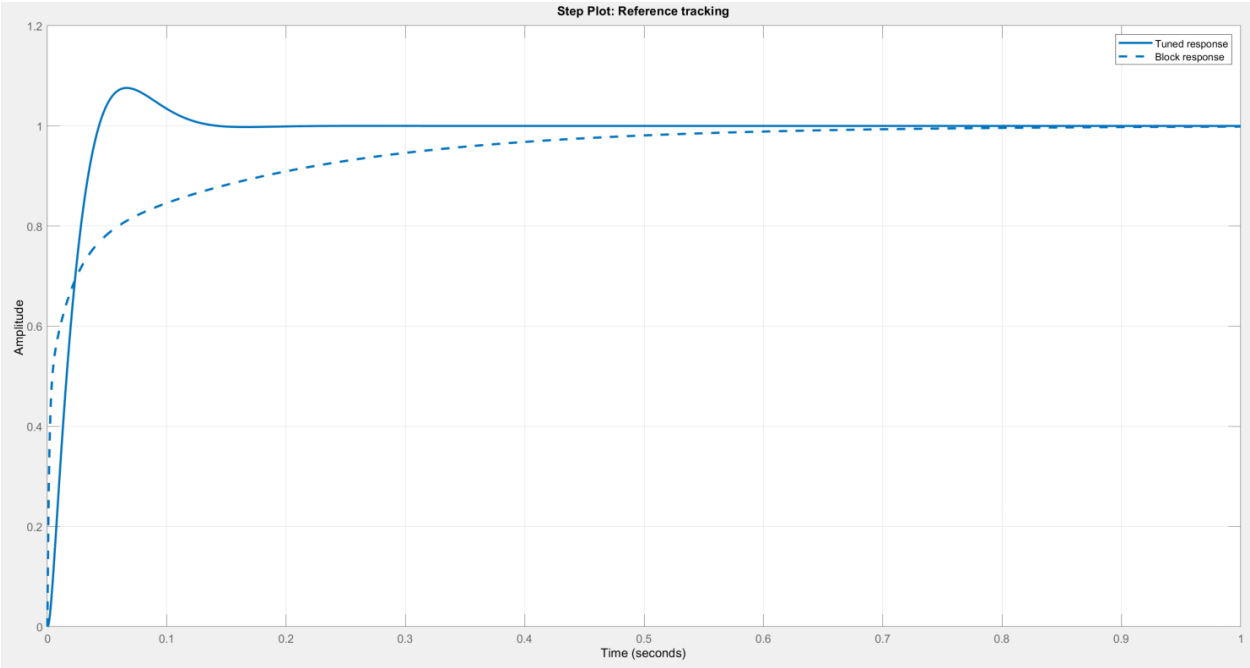


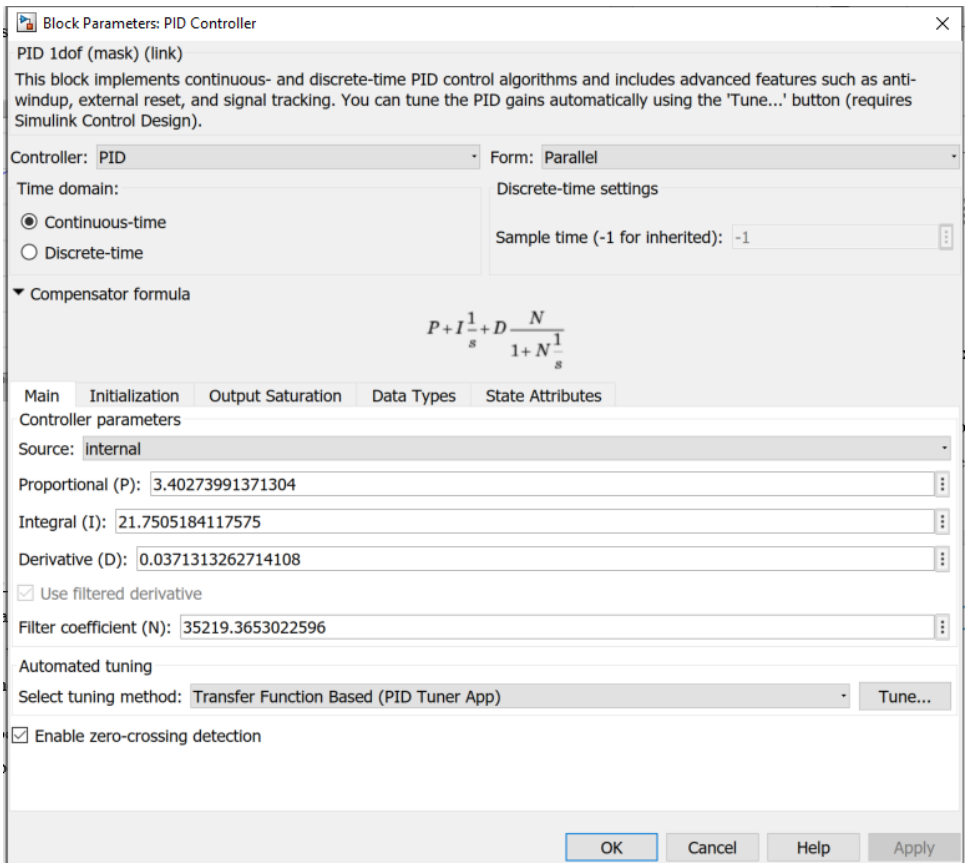
5.2 Use the DC motor in closed loop velocity control. That is, use the given diagram in which you leave the feedback of Ω via K_E, and add unity feedback for Ω. Need to tune a PID controller block. Again, the task is to create a specific trapezoidal velocity profile input command signal, so that the motor's angle moves by the right amount by the specified timing.

For this question we will edit the above motor model we created to include a PID controller and unity feedback. See the modified model below on how we added the PID and unity feedback.

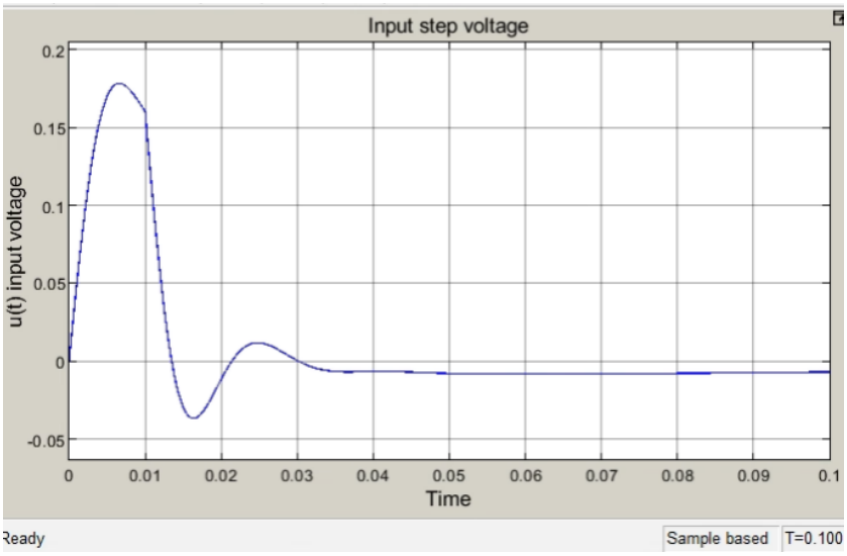


Now we will analyze the PID controller and using one of its tuning tools to get the best outcome for our design. I opted to use the transfer function based tuning app, rather than the frequency based one. I found much better results when tuning based on the transfer function. See the graph of my PID controllers' response below, along with an image of the settings used on the PID controller.

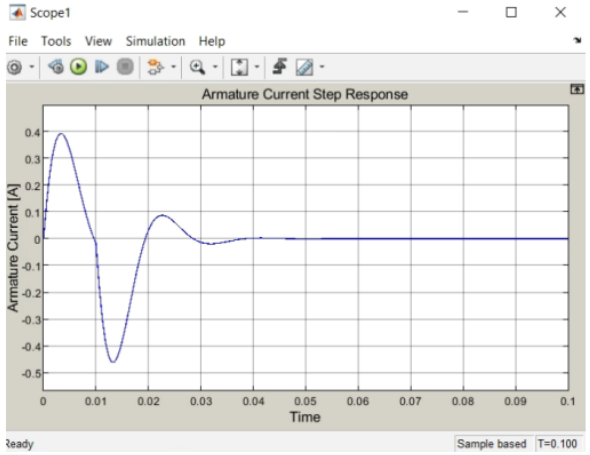


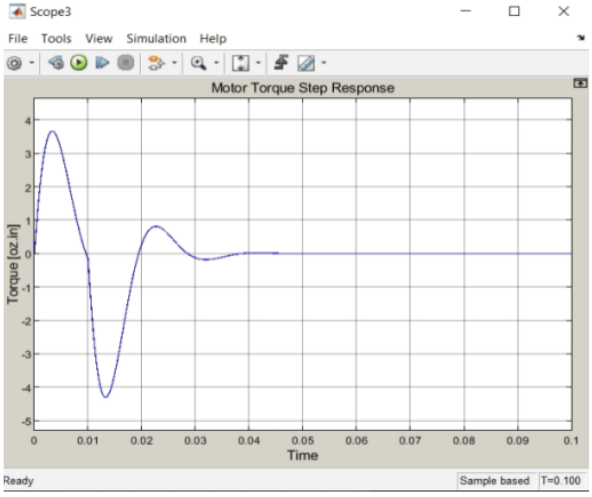


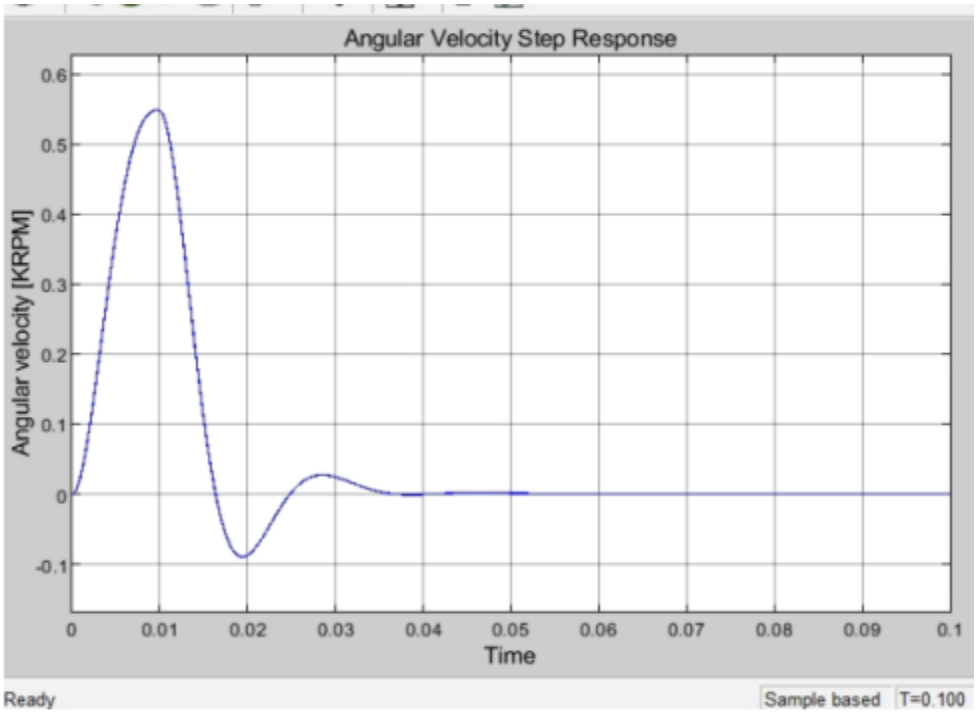
In the first image you can see how the auto tuned PID controller, tuned with a transfer based tuning app, is a better PID combination than the one I was using. The transfer function tuning gives us the shortest oscillation time while maintaining the final value we are looking for. In the second image I applied the values I obtained from the transfer function tuning application to my PID block. You can observe what these values are above. We will now run the motor again with the same trapezoidal input and observe how the input is distorted with the feedback and PID controller. See the new input below. This is the same trapezoidal input from 5.1, however the constant feedback changes it and mellows it out as seen below



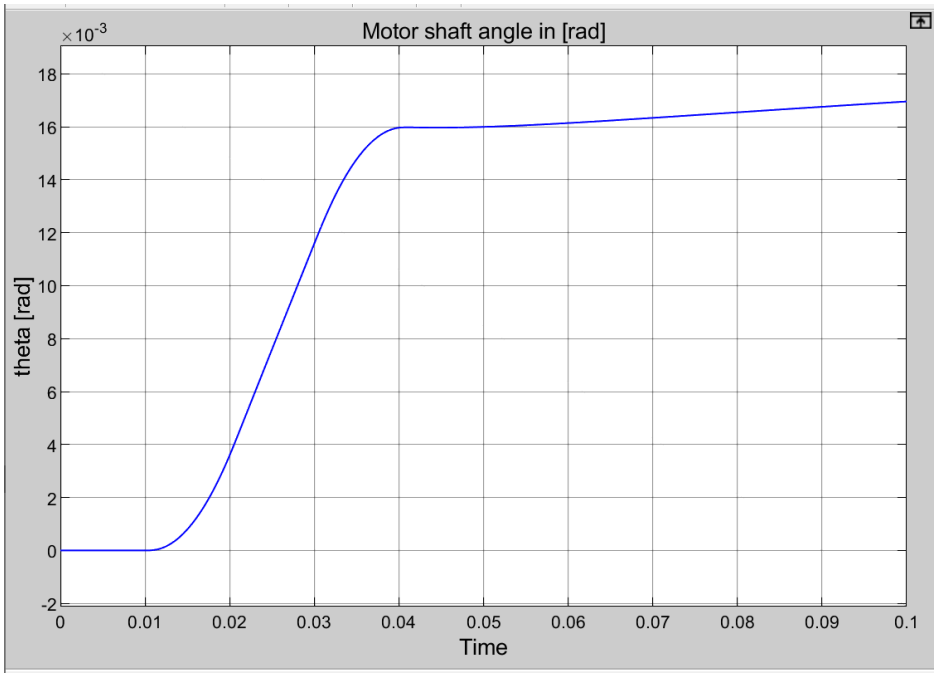
With our input noted, and our PID controller setup; let's look at our outputs. See the resulting current, torque and velocity of the motor using this setup. Notice how they are smoother than the previously obtained in 5.1 while remaining in spec.



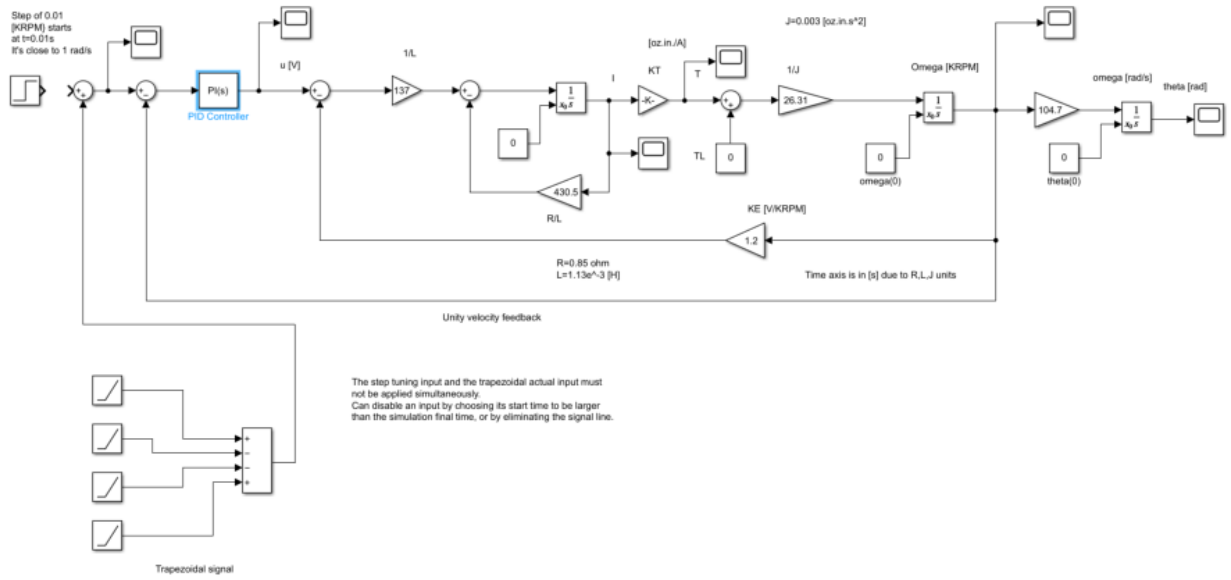




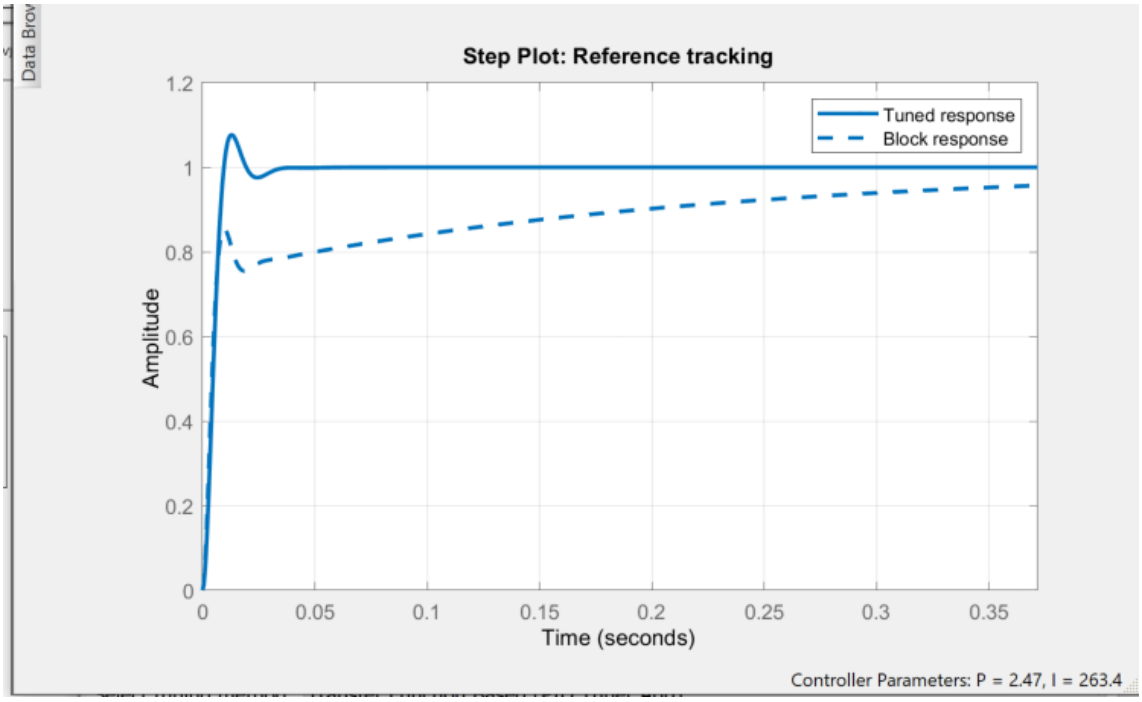
Lastly, let's look at the solution to this question, which is the motor angle using the trapezoidal input along with unity feedback and a PID controller. See the motor angle below. Notice how it achieves this angle about .01s faster than its 5.1 counterpart.



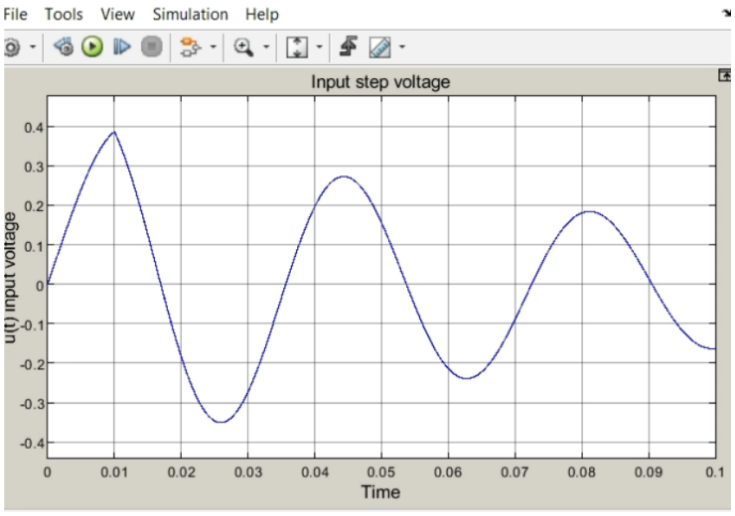
5.3 In 5.2, inspect the tuned PID control transfer function. If the D gain appear to be very small try to eliminate it altogether, leaving only a PI controller. Compare the performance to that obtained in 5.2.

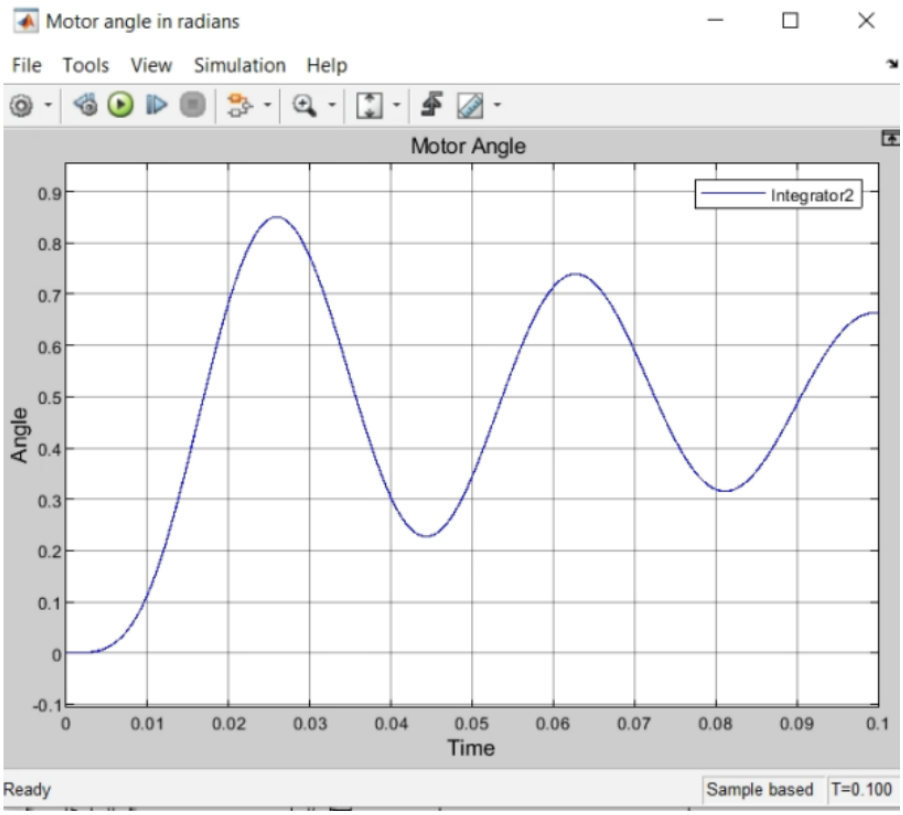


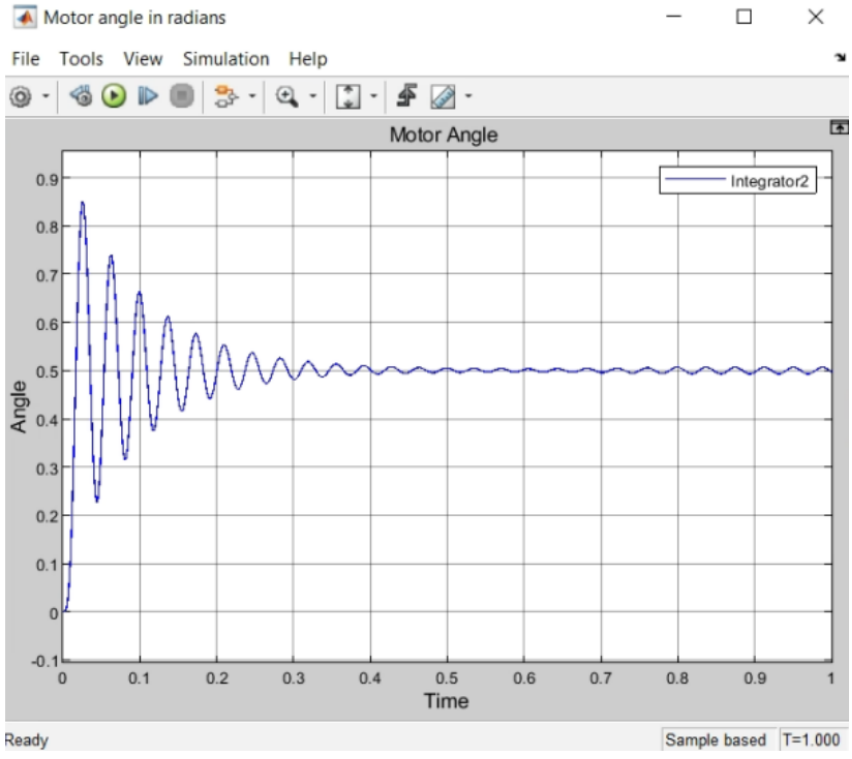
This proportional controller eliminates the derivative aspect of it, which makes rather a proportional-integral-controller (PI Controller) instead of a PID Controller. Because the derivative aspect is eliminated, the step plot amplitude is more unbalanced since the integral is in full control of the graph and not balanced by the derivative. If you look back at our PID controller setup in 5.2 we will be using the same numbers except we will forcibly turn the D value to be 0 as it is very small in 5.2. We can observe what this does to the PID controller by looking at the transfer function auto tuning application. See that graph below, note that the dotted line is what we are forcing the PID controller to do, and the solid line is what it suggests us to tune it to.



As you can see above, the D value is important and running this motor with just a PI controller will not be working well. Observe the rest of the values below and note that when compared to 5.2 they all fall very short. While they do remain in spec, they have settling times that reach almost 1 second; on top of that, the trapezoidal input doesn't even remain remotely trapezoidal looking when this is applied. While I believe this could be used on the motor, the motor would be shaking back and forth for quite some time before actually doing the .5 radian movement. If the goal was to use a trapezoidal signal train to control this motor like how most motors operate, this would not work.




Motor angle in radians
File Tools View Simulation Help
Motor Angle
Integrator2
Angle
Time
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1
Ready
Sample based
T=0.100


Motor angle in radians
File Tools View Simulation Help
Motor Angle
Integrator2
Angle
Time
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Ready
Sample based
T=1.000