Cristian Navarrete

Robotics - HW3

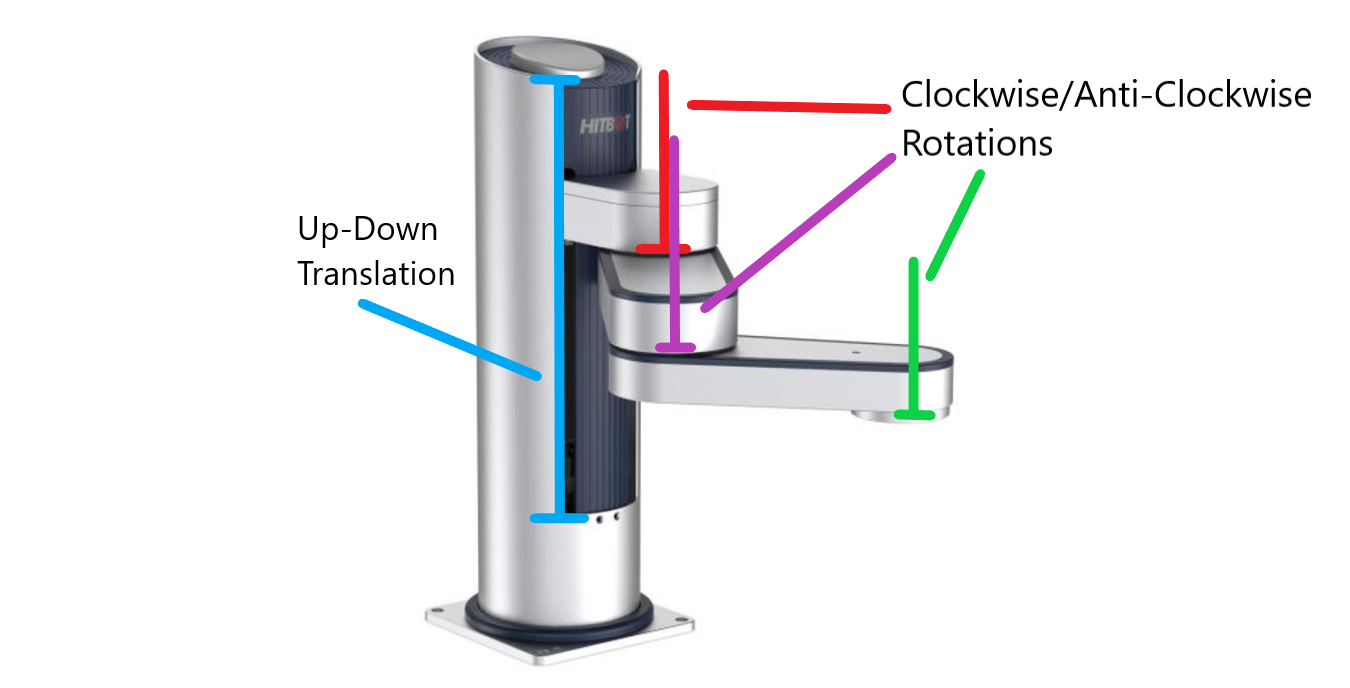
**Problem 1: Manipulator DH Parameters and Forward Kinematic Model**

* 1. **Classify the robot according to its joints type and order.**

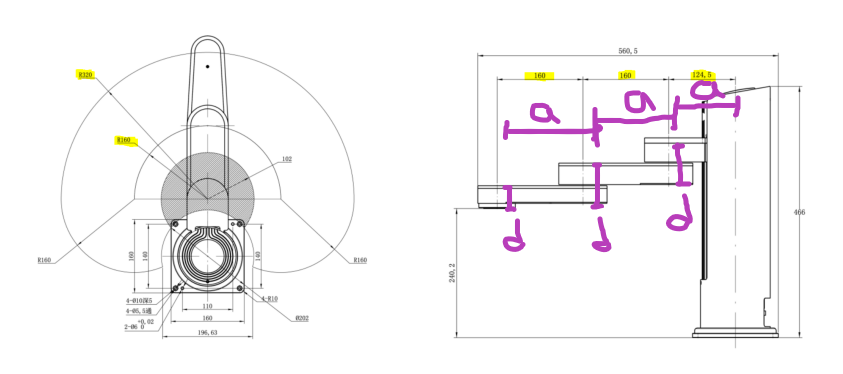
When starting from the base, the Zarm robot is a Prismatic, Revolute, Revolute, Revolute (P3R) robot.

* 1. **Show all the joints axes of motion (marked on the diagram above) and find the common normal lengths an and the distances dn of the DH model.**

Below is the prismatic axis of motion (in blue) of translation along the z axis, and the revolute axes of motion (in red, purple and green) of rotation around the z-axes (each joint’s z-axis frame is aligned).



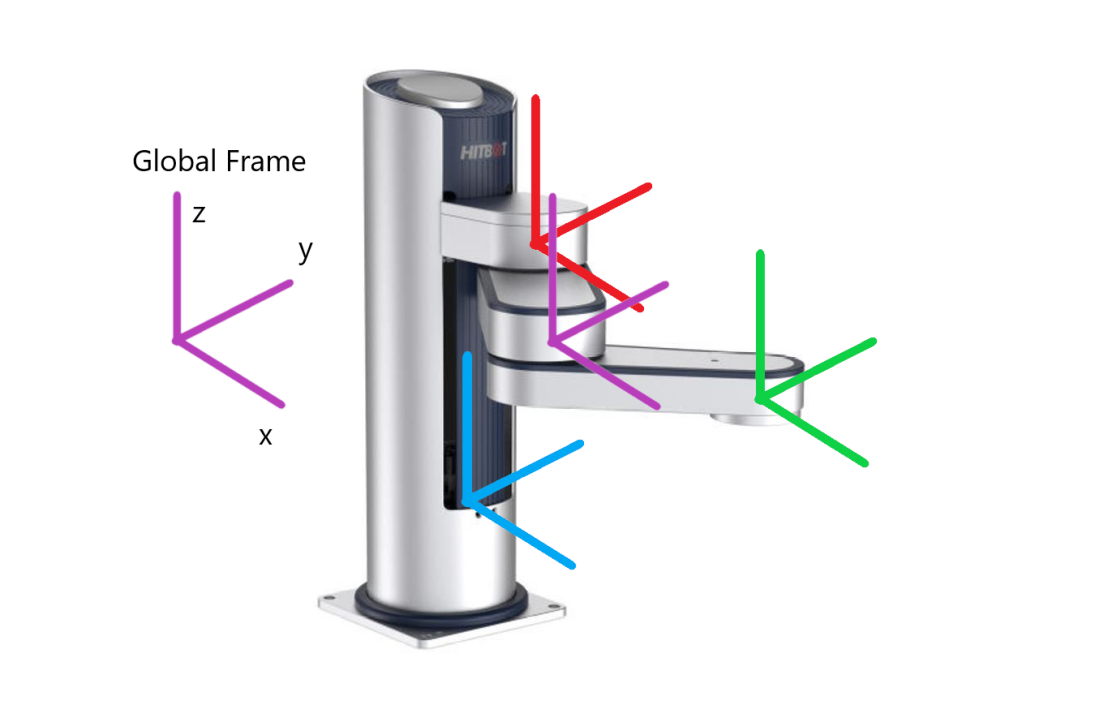
From the robot documentation, we can see the a and d distances visualized (the actual values will be represented as dx or ax, where x is the joint number):

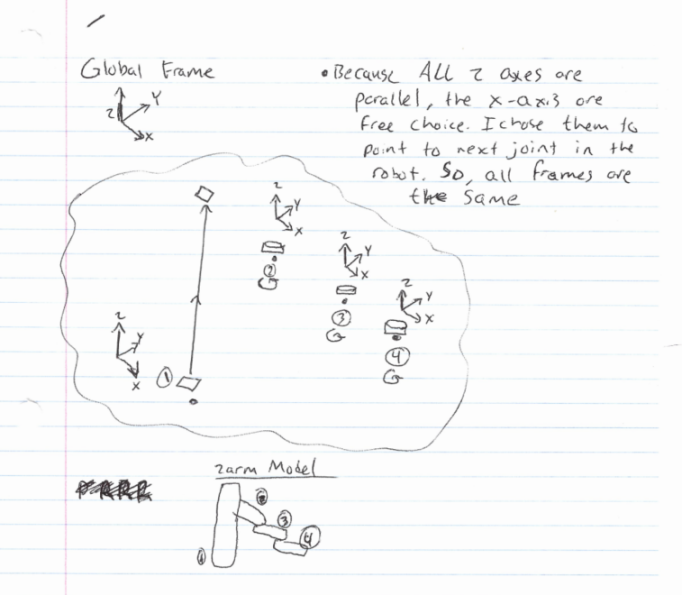


|  |  |  |
| --- | --- | --- |
| **Joint #** | **an** | **dn** |
| 1 | a1 | d1**\*** |
| 2 | a2 | d2 |
| 3 | a3 | d3 |
| 4 | 0 | d4 |

d1 is variable as joint 1 is prismatic and the distance the initial position and it’s position is dependent on how far it has translated up/down. I took each constant d{2-4} to be the translation along the positive direction of the z-axis to the point where each revolute joint touched the next joint. d4 is the distance to the tool. Each joint has a normal vector distance (an) to line up the z axes (except for joint 4 where the tool is directly below that joint).

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





Assuming the rest state of the robot at rest is with all joints straight out, the origin of the prismatic joint frame is set at the normal intersection between that joint and the first revolute joint. The origins of the revolute frames are centered around the point of rotation for the joint. All of the frames will have the same orientation when the arms are straight out and final arm has 0 rotation value.

* 1. **Find the rotation angles θn and the twist angles αn and explain briefly**

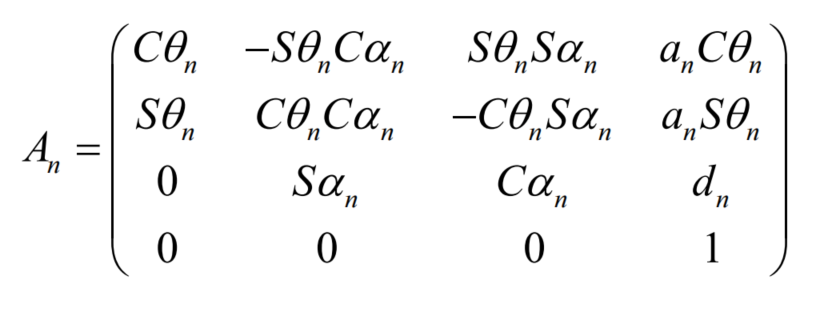
\* = variables

|  |  |  |
| --- | --- | --- |
| **Joint #** | **θn** | **αn** |
| 0-1 | 0 | 0 |
| 1-2 | **θ2\*** | 0 |
| 2-3 | **θ3\*** | 0 |
| 3-4 | **θ4\*** | 0 |

θ2, θ3, θ4 are variables because they are revolute joints and the theta parameter to align with the next joint is completely dependent on how far the joint has rotated. Prismatic joint 1 has rotation parameter set to 0 as it does not need to rotate since the x-axis of frames 1 and 2 are aligned. No twist angles needed for any joints as all frames have the same rotation/translation z-axis at all times (facing straight up).

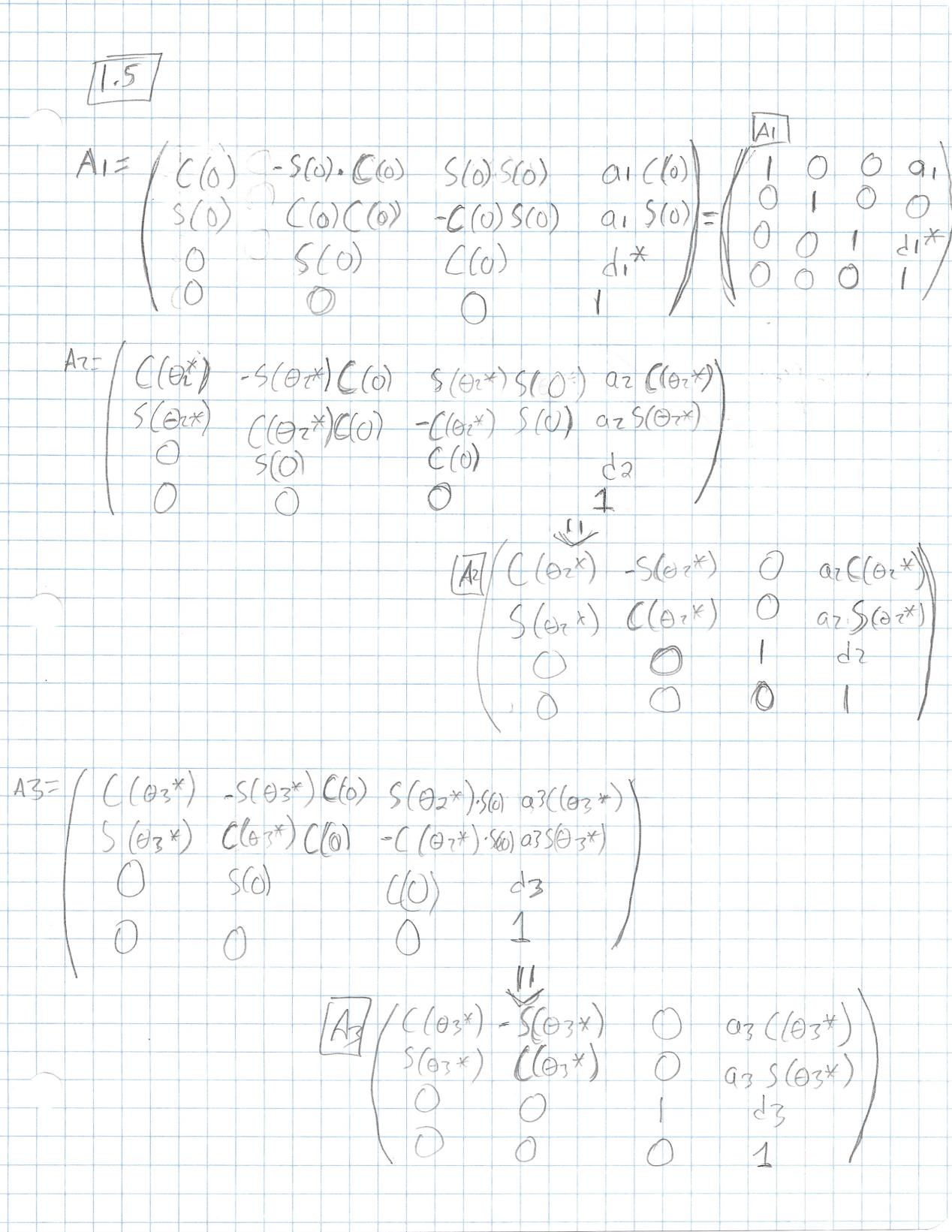
* 1. **Find the An matrices, and then find the full kinematic model.**

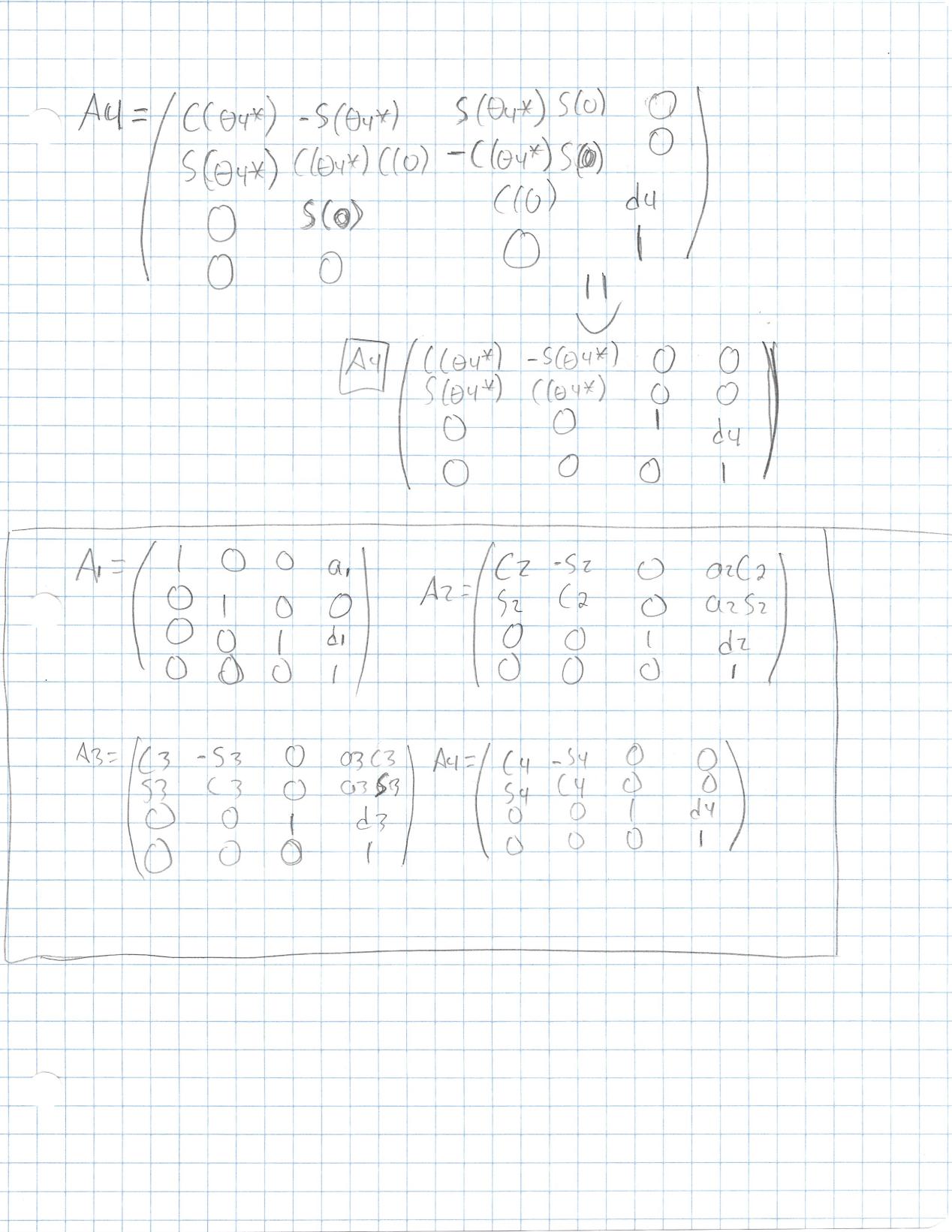
Using below form, I break down the DH table to create a transformation matrix for each joint. Then I multiply each transformation matrix in sequence in order to get the full kinematic model of the robot.

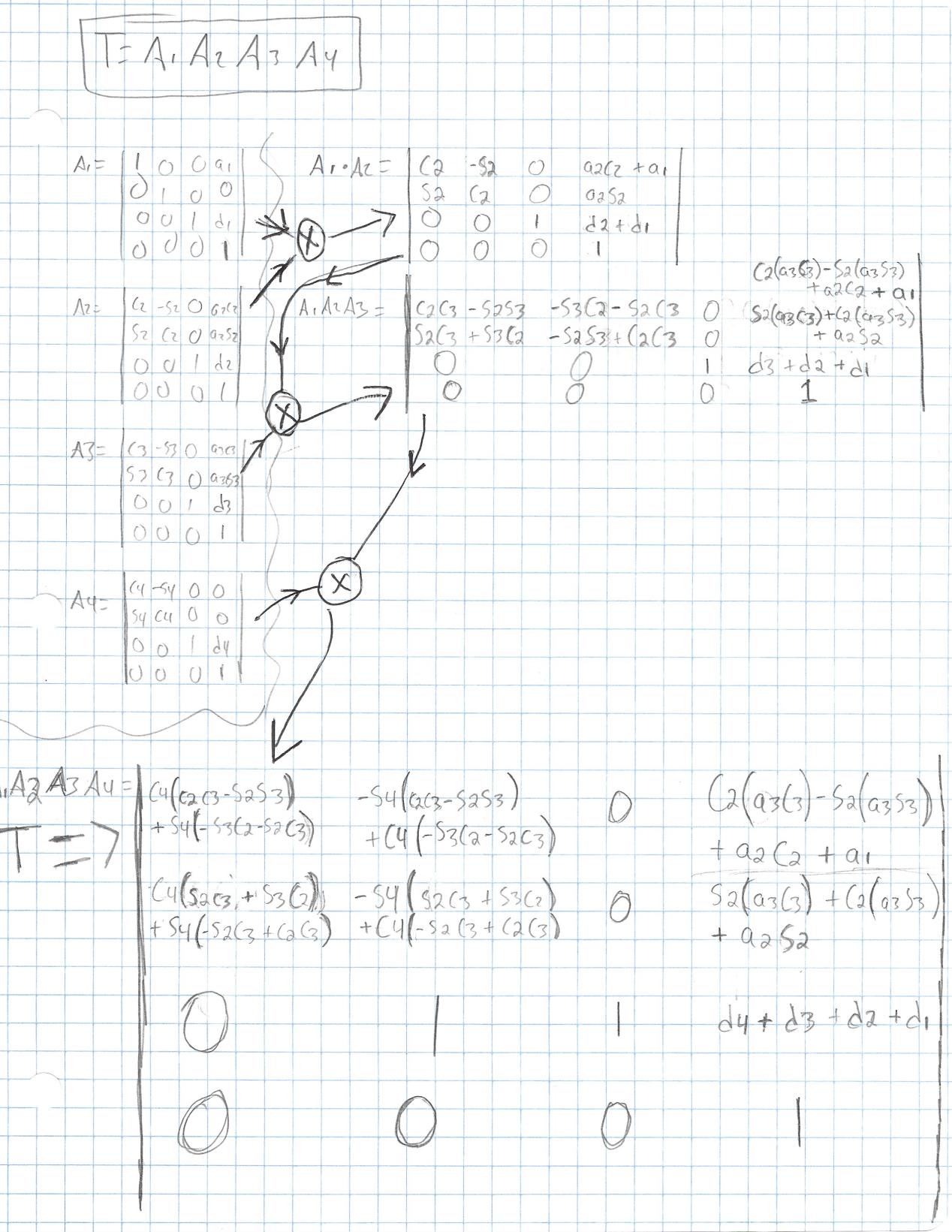


\* = variables

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Joint #** | **θn** | **dn** | **an** | **αn** |
| 0-1 | 0 | d1\* | a1 | 0 |
| 1-2 | **θ2\*** | d2 | a2 | 0 |
| 2-3 | **θ3\*** | d3 | a3 | 0 |
| 3-4 | **θ4\*** | d4 | 0 | 0 |



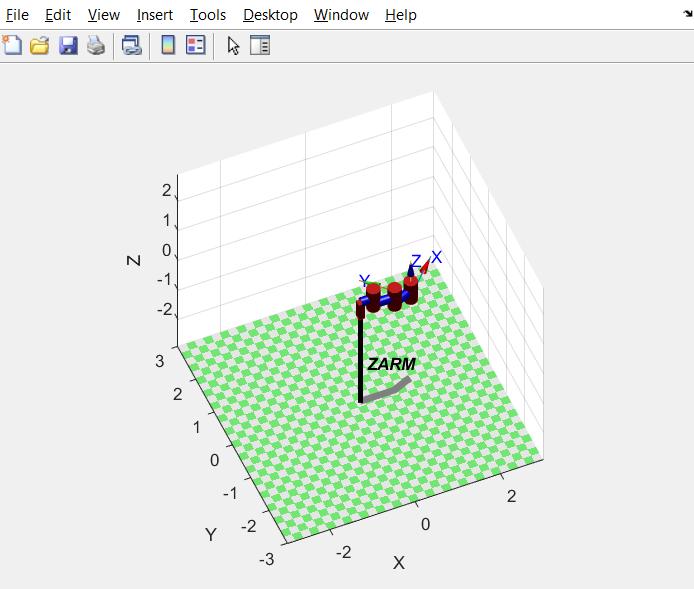




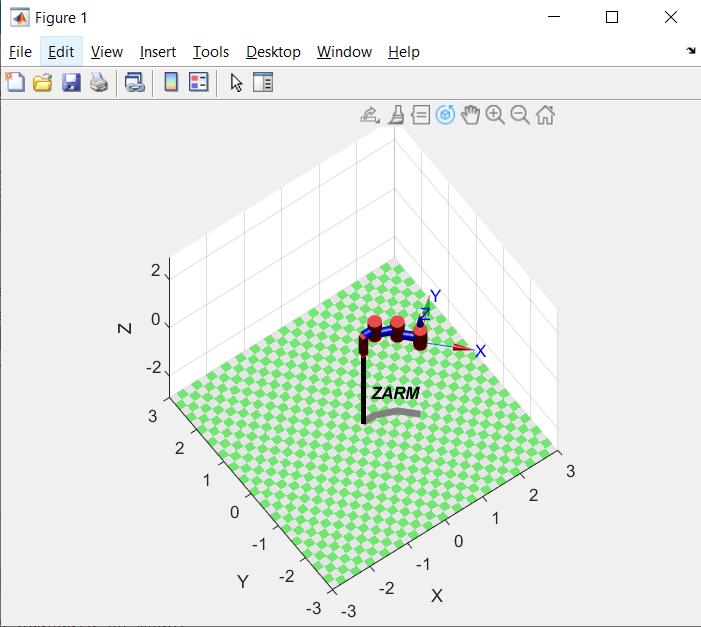
* 1. **Using MATLAB Robotic Toolbox SerialLink, create a model of the robot. Plot a few robot configurations for several chosen joint position vectors.**

For this problem, I provided arbitrary values for the constant a and d parameters to get a model that could be plotted at some scale (the values are in my code snippet below). I chose the below 3 poses to display. My point vectors are visible in the matlab code.

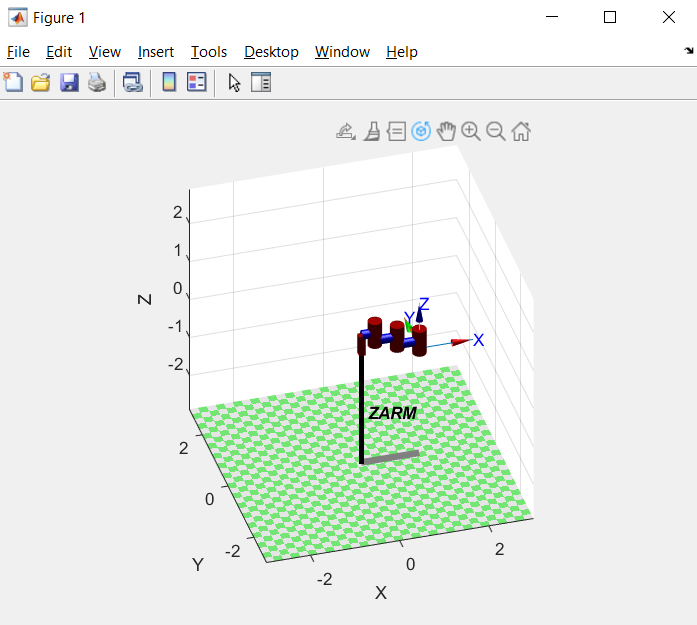
Extended Slightly to left



Extended to the right



Straight Out



Code:

ZARM=SerialLink([Prismatic('a', 0.3, 'theta', 0, 'alpha', 0, 'qlim', [0 1]) Revolute('a', 0.5, 'd', -0.2, 'alpha', 0) Revolute('a', 0.5, 'd', -0.2, 'alpha', 0) Revolute('a', 0, 'd', -0.2, 'alpha', 0)], 'name', 'ZARM');

ZARM.plot([0.5, 0, 0, 0], 'deg');

ZARM.plot([0.75, 75, 75, 0], 'deg');

ZARM.plot([0.5, 0, -75, -75], 'deg');

I used a Prismatic, 3 x Revolute SerialLink with the above parameters to plot out the different robot configurations.

**Problem 2: Importing a Commercial Robot Model**

**From the robot models that are available in RTB, pick up a 6 DOF commercial robot manipulator.**

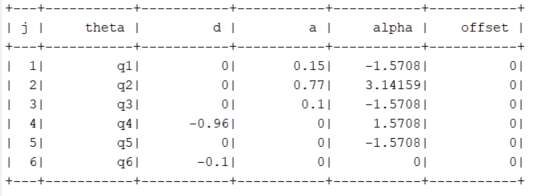
**[Note: The PUMA 560, Stanford Arm and IRB 140 are excluded because these robots have been discussed in class].**

**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.**

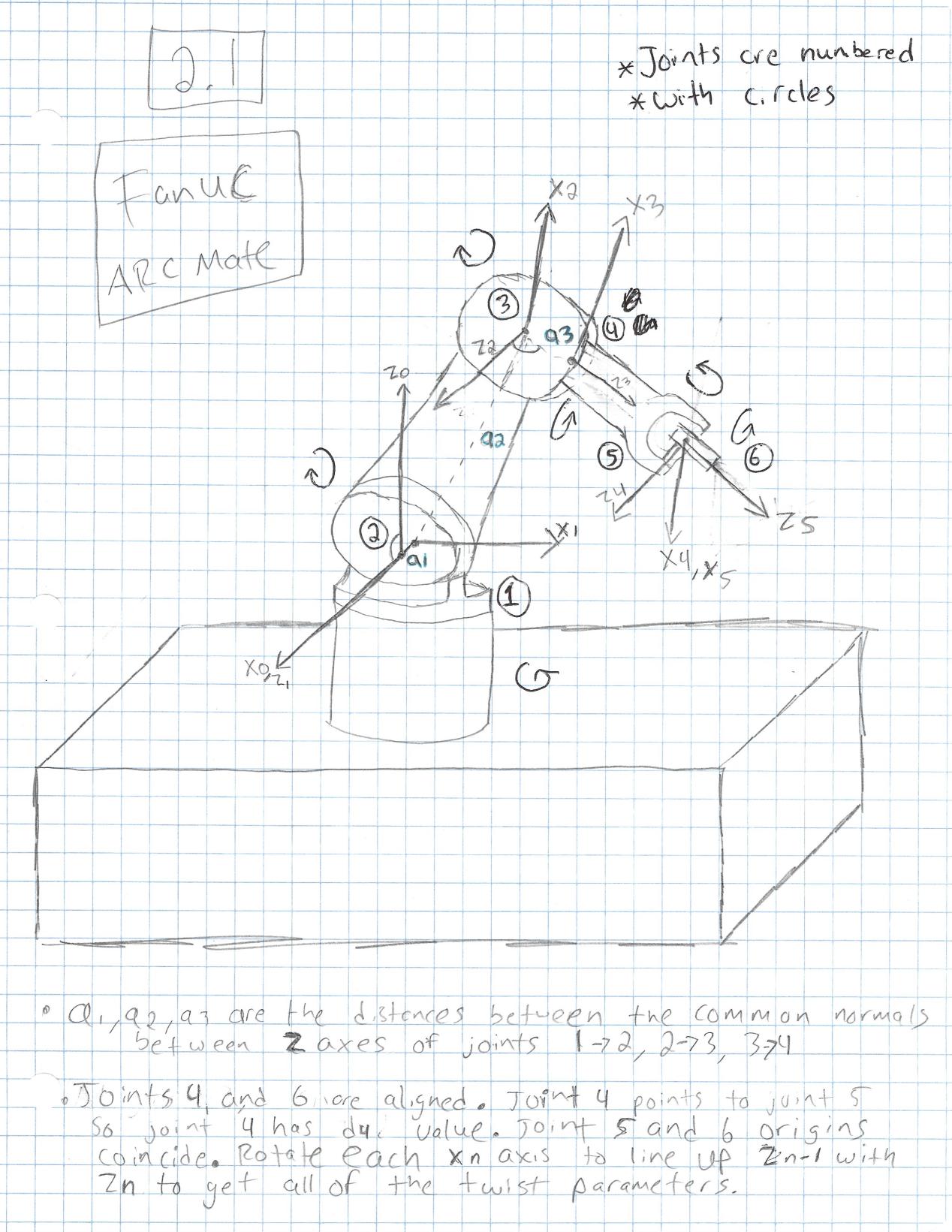
I chose the Fanuc ARC Mate 12 (mdl\_fanuc10L in RTB, which generates the robot as R).



Model DH Parameters



Manual DH Parameters



\* d4 and d6 line up with the z-axes of J4 and J6 respectively, so they are unmarked.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Joint #** | **θn** | **dn** | **an** | **αn (degrees)** |
| 1 | **θ1** | 0 | a1 | 90 |
| 2 | **θ2** | 0 | a2 | 0 |
| 3 | **θ3** | 0 | a3 | 90 |
| 4 | **θ4** | d4 | 0 | -90 |
| 5 | **θ5** | 0 | 0 | -90 |
| 6 | **θ6** | d6 | 0 | 0 |

With observation, I can calculate suitable DH parameters based on the picture of the robot. My parameters ended up being a bit different from RTB’s DH model parameters. While the dn and an match between both, my twist angles are slightly different (RTB decided to twist at for 180 degrees, but my J2 and J3 have same orientation of z, so I didn’t see the need).

**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.**

The fkine command is simply used by initializing the robot workspace parameter R, and then specifying poses in vector form, where each value is a rotation/translation value for the joint. This gives a transformation matrix for the end effector. ikine6s works in the reverse order by giving the joint values when given a valid transformation matrix.

Code:

mdl\_fanuc10L

% Set the fkine configurations

config1=[0,-0.75\*pi,pi/8,0,0,0]

config2=[pi/2,-pi/4,pi/4,0,0,0]

%fkine #1

T1=R.fkine(config1)

R.plot(config1)

%fkine #2

T2=R.fkine(config2)

R.plot(config2)

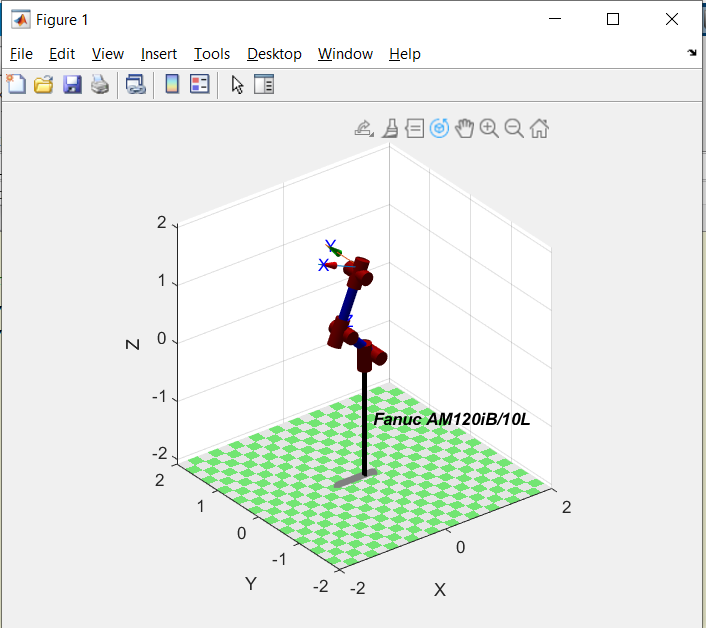
%ikine6s #1

q1=R.ikine6s(T1)

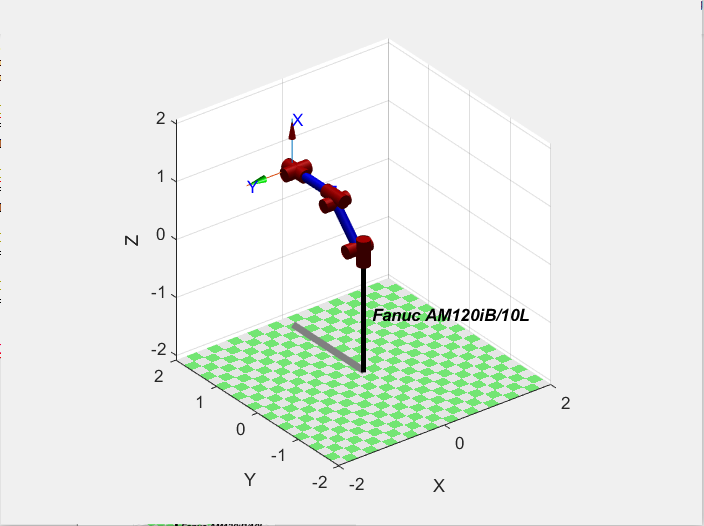
%ikine6s #2

q2=R.ikine6s(T2)

**fkine [0, -0.75\*pi, pi/8, 0, 0, 0] configuration**



**fkine [pi/2, -pi/4, pi/4, 0, 0, 0] configuration**



There is an error using the ikine6s in this case. This robot is not supported by the function.



**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?**

Chosen Configurations:

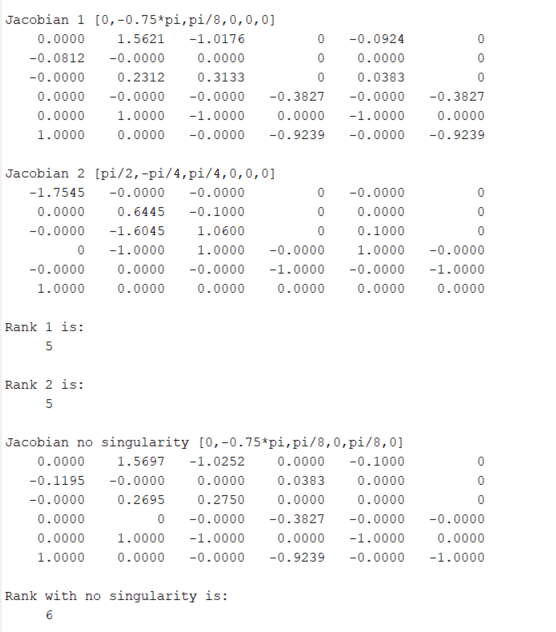
config1=[0,-0.75\*pi,pi/8,0,0,0]

config2=[pi/2,-pi/4,pi/4,0,0,0]

Both of my selected configurations (numbered as 1 and 2) have wrist singularities (so they have a rank of 5). Since the joints 4 and 6 rotate the same, when they are aligned, a degree of freedom is lost. To regain the DOF, it is necessary to make sure that the z-axes of joints 4 and 6 are not aligned. To do that, I can provide a small theta value to joint 5 to achieve this.

New Config with no singularity:

no\_sing\_config=[0,-0.75\*pi,pi/8,0,**pi/8**,0]



Code:

mdl\_fanuc10L;

config1=[0,-0.75\*pi,pi/8,0,0,0];

config2=[pi/2,-pi/4,pi/4,0,0,0];

J0=R.jacob0(config1);

J1=R.jacob0(config2);

r1=rank(J0);

r2=rank(J1);

disp('Jacobian 1 [0,-0.75\*pi,pi/8,0,0,0] ');

disp(J0);

disp('Jacobian 2 [pi/2,-pi/4,pi/4,0,0,0] ');

disp(J1);

disp('Rank 1 is: ');

disp(r1);

disp('Rank 2 is: ');

disp(r2);

no\_sing\_config=[0,-0.75\*pi,pi/8,0,pi/8,0];

J2=R.jacob0(no\_sing\_config);

r3=rank(J2);

disp('Jacobian no singularity [0,-0.75\*pi,pi/8,0,pi/8,0] ');

disp(J2);

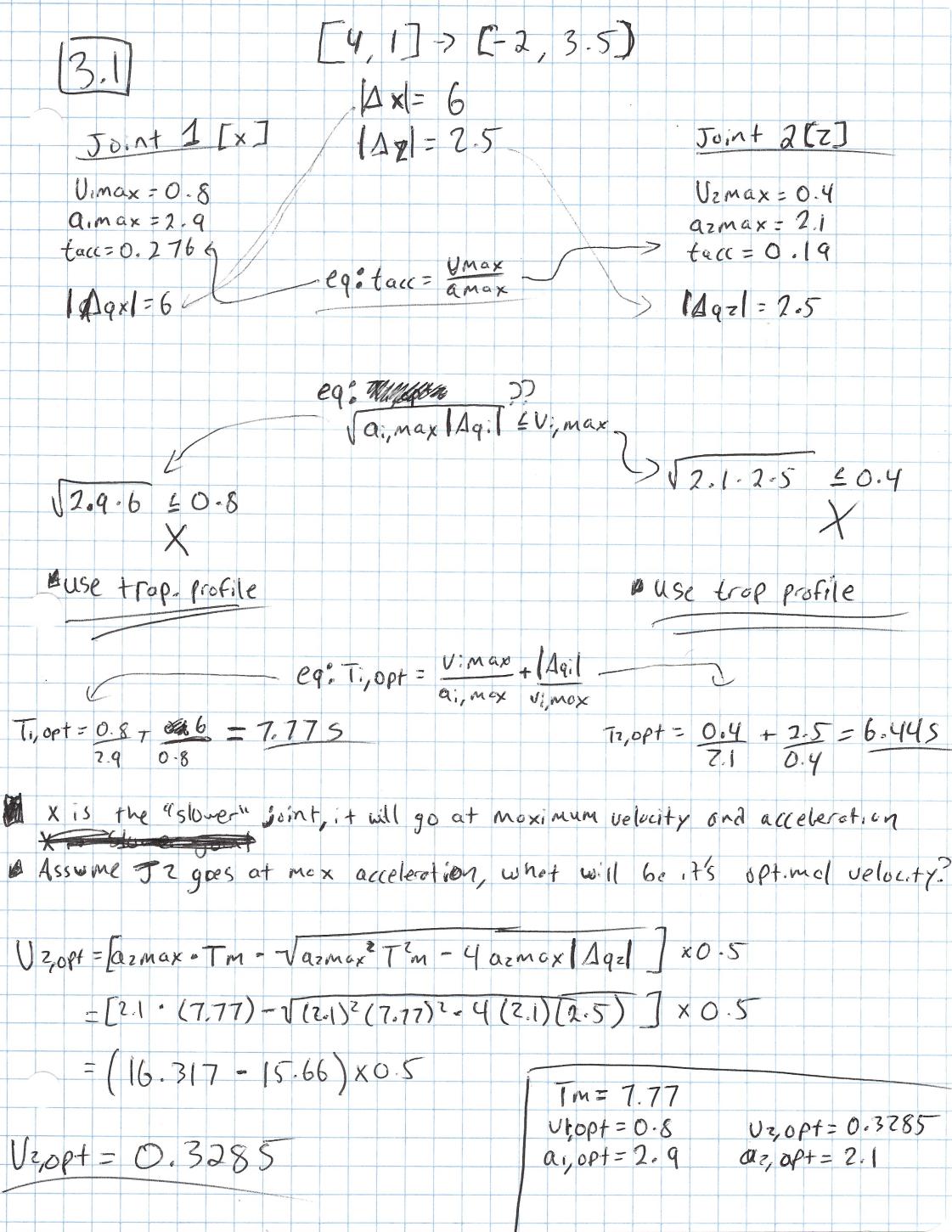
disp('Rank with no singularity is: ');

disp(r3);

**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/s2 ], respectively. The maximum velocity and acceleration of joint Z are 0.4[m/s] and 2.1[m/s2 ], 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.**

****

**Chosen Velocities/Accelerations**

**Xv,opt = 0.8 m/s**

**Xa,opt = 2.9 m/s2**

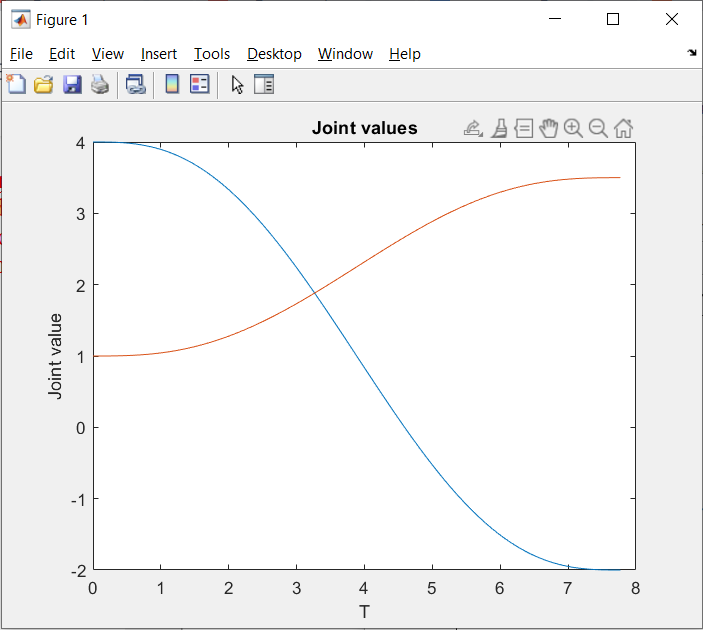
**Zv,opt = 0.3285 m/s**

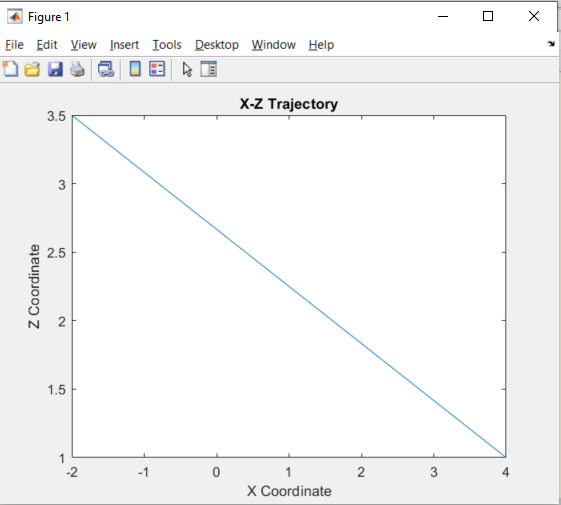
**Za,opt = 2.1 m/s2**

By solving for the “slower” joint, the faster joint parameters can be interpolated in order to sync the Tm parameter. The velocity profile chosen was the trapezoidal profile.

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

To the jtraj function, I provided the initial and final positions for both joints and specified the T value as 0 to 7.77 in 0.07 increments for 112 distinct time steps, each 0.07 seconds long. The result is the q variable which contains the q interpolated for each joint to sync. Below, I have both the joint value with respect to T and the mapping of the X-Z parameters in space as the X-Z trajectory. If looking straight at the X-Z plane, the two arms combined move to the up and left.





Code:

t=[0:0.07:7.77]';

q=jtraj([4 1], [-2 3.5], t);

figure(1);

plot(t, q(:,1:2));

xlabel('T');

ylabel('Joint value');

title('Joint values');

figure(2);

plot(q(:,1),q(:,2));

title('X-Z Trajectory');

xlabel('X Coordinate');

ylabel('Z Coordinate');

**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?**

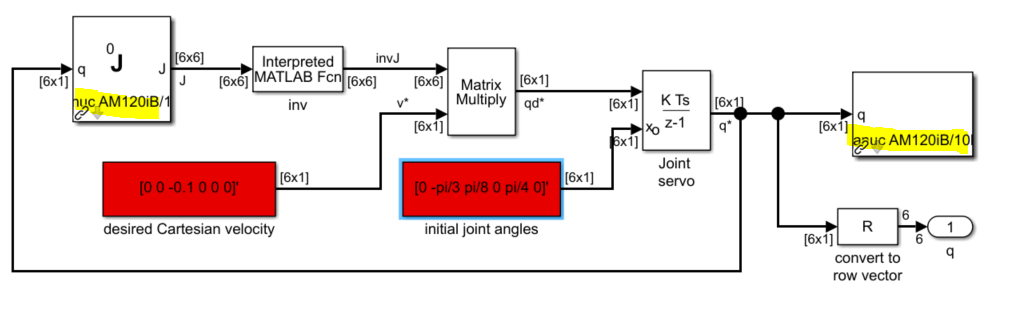
For my solution, I went with the FANUC robot (mdl\_fanuc10L) for a z-axis motion of velocity -0.1 m/s. I also used a stop time of 10 in the simulink model.

Robot : mdl\_fanuc10L (robot “**R**”)

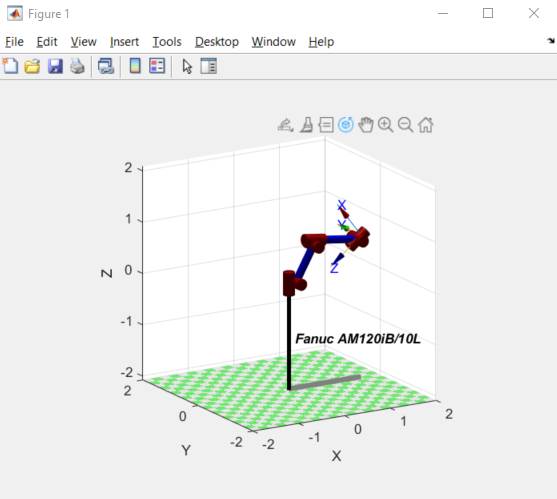
RRMC Cartesian Velocity : [0 0 -0.1 0 0 0] (z-axis motion of -0.1 m/s)

RRMC Initial Joint Angles : [0 -pi/3 pi/8 0 pi/4 0] (upright pose)

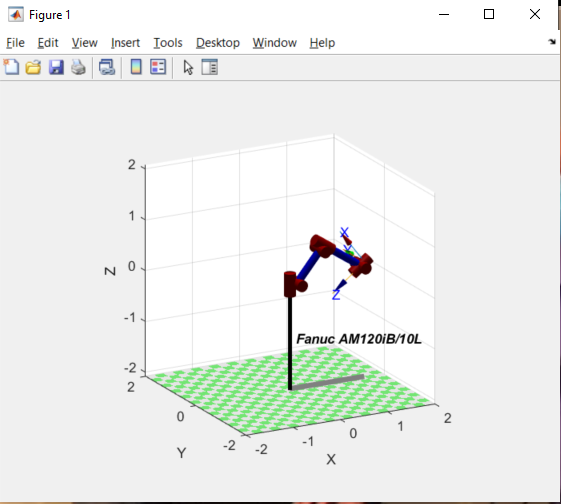
Initial K and T values : Gain of 0.5, Sample Time of 0.05



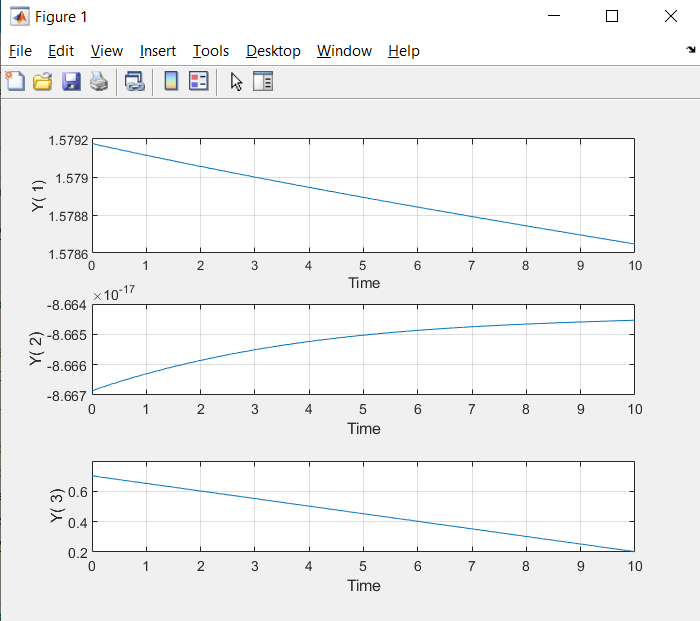
Initial Pose:



Final Pose:



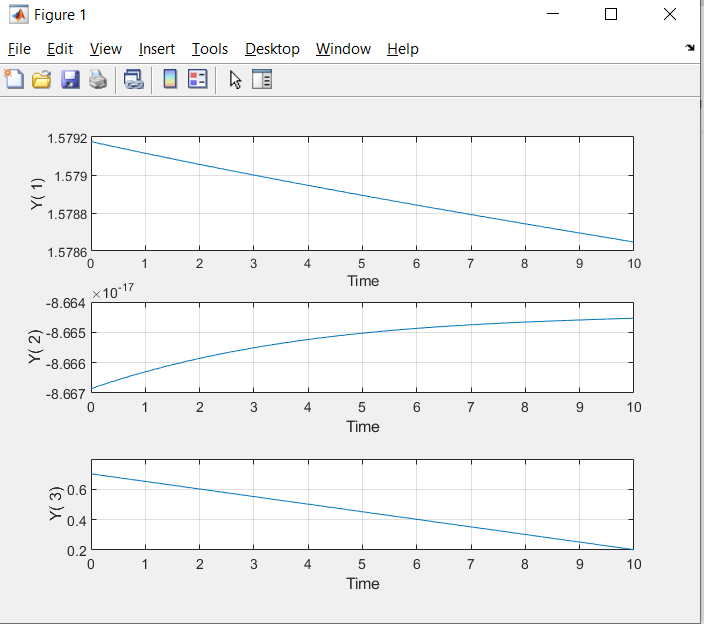
Joint Values:



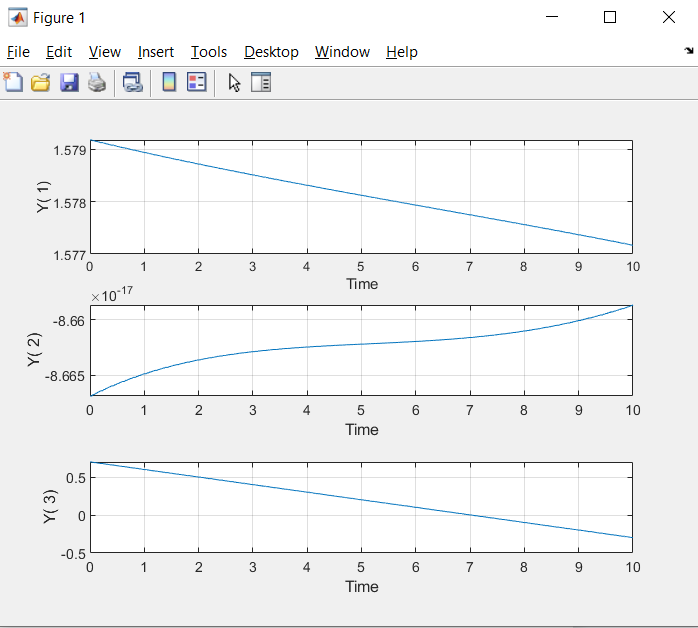
For this run, I can see that the major of the end-effector frame is in the Z-axis. The XY axes remained relatively stable with little changes. This makes sense since this problem was for a Z-axis move of 0.1 m/s.

**Experimenting with K/T Values**

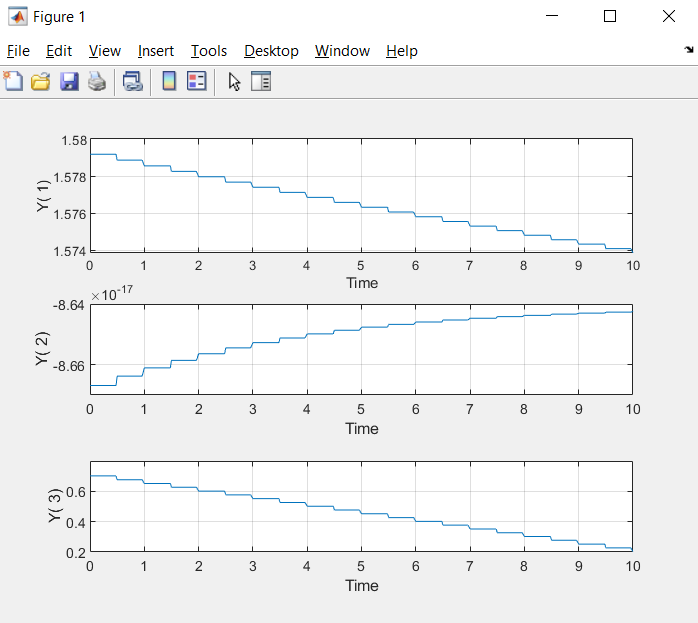
K=0.5 / T=0.05 (initial)



K=1 / T=0.05 (double gain)



K=0.5 / T=0.5 (lowered sample rate)



Observations:

Shown above are the three runs for the varying parametrs. As expected, increasing the gain by a factor of 2X (from 0.5 to 1), it practically doubled the z-axis velocity and allowed the robot to move farther over the same time period, giving us an entire meter change in the z-position over the 10 second time frame. Turning up the gain even further eventually results in glitchy and erratic behavior when the robot can no longer move anymore in that direction.

I also saw that changing the sampling period from 0.05 seconds to 0.5 seconds caused the joint position profile to become jagged as the position is not updated as often as when using a value of 0.05. Turning this value up even more causes the model to run much slower, but with more precision. For this example though, K=1 and T=0.05 are plenty for demonstrating the straight live movements.

Code:

%% Question 4.1

mdl\_fanuc10L

r=sim('sl\_rrmc\_Q3');

t=r.find('tout');

q=r.find('yout');

T=R.fkine(q);

xyz=transl(T);

mplot(t,xyz(:,1:3));

**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?**

For this solution, I again used with the FANUC robot (mdl\_fanuc10L) for an xy circle of radius 0.1 and rev/s of 0.1. I also used a stop time of 10 in the simulink model. Instead of the “qn” pose, I used a new pose that made more sense for this particular robot.

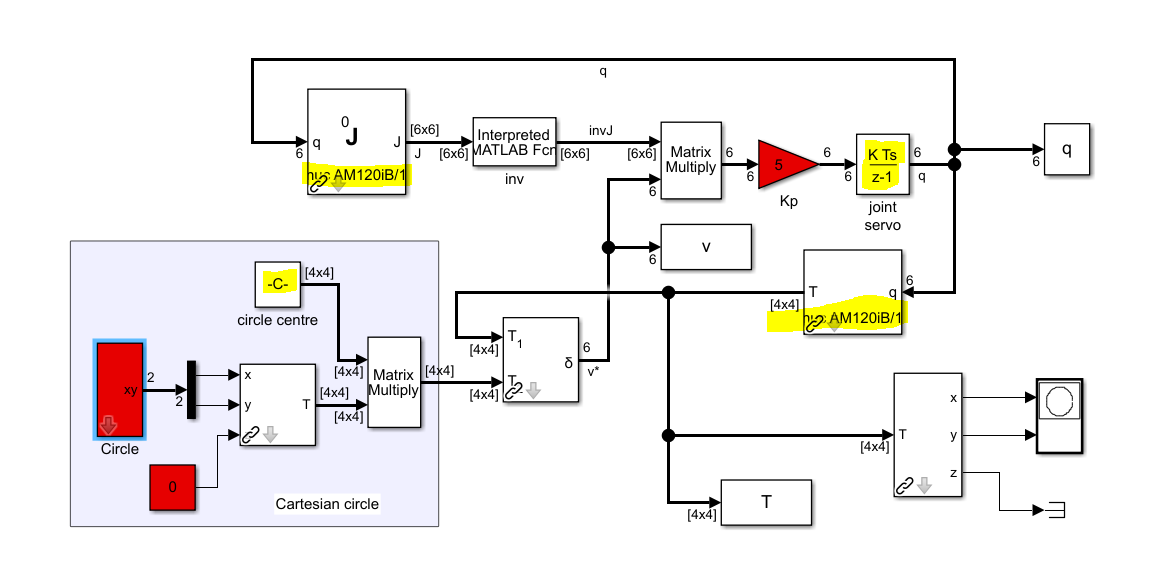
Robot : mdl\_fanuc10L (robot “**R**”)

Initial Configuration : [0 -pi/4 pi/8 0 pi/8 0]

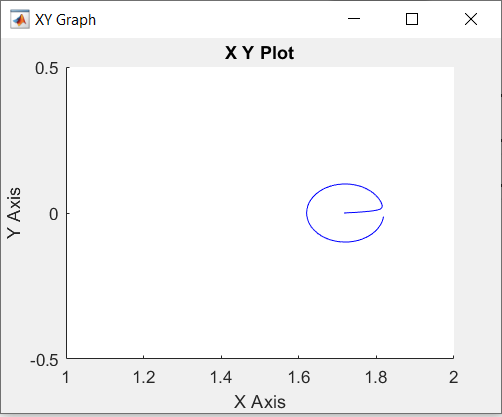
Radius : 0.1

Rev/s : 0.1

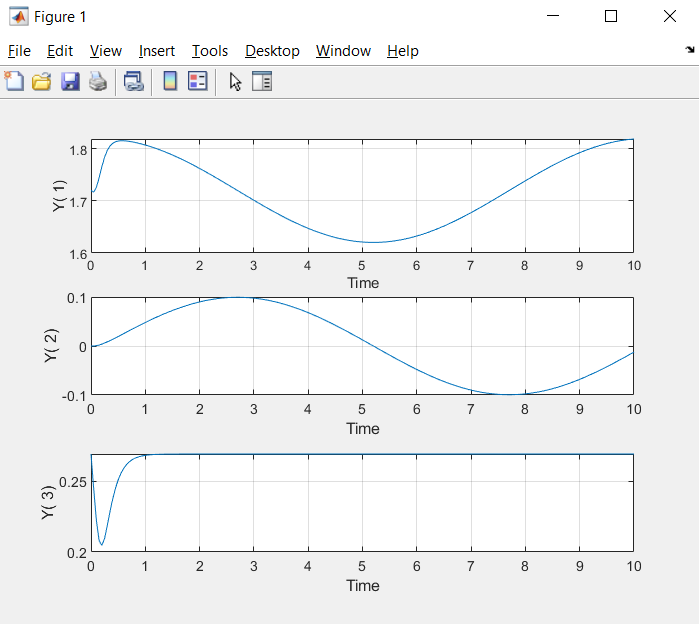
Initial Kp and T values : Gain of 5, Sample Time of 0.05



Initial Motion

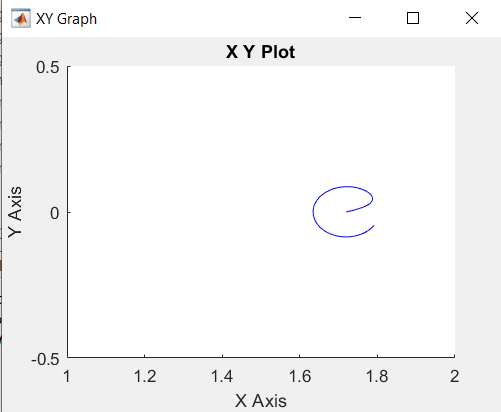
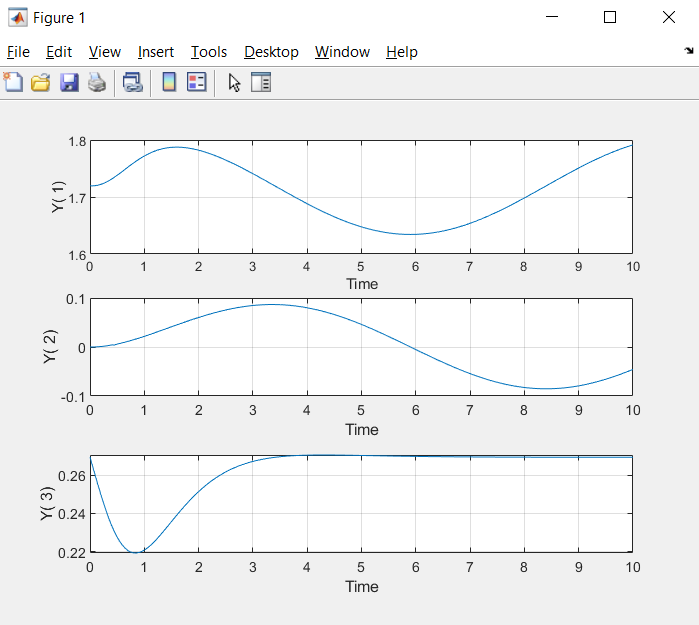


XYZ Values

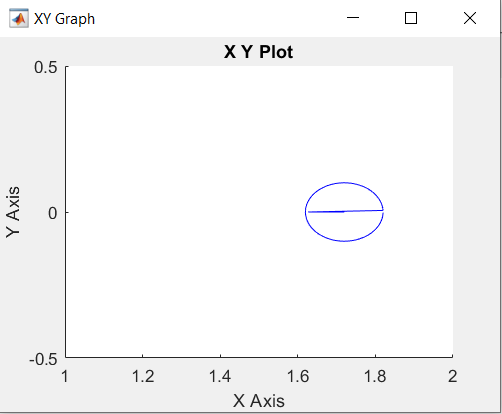
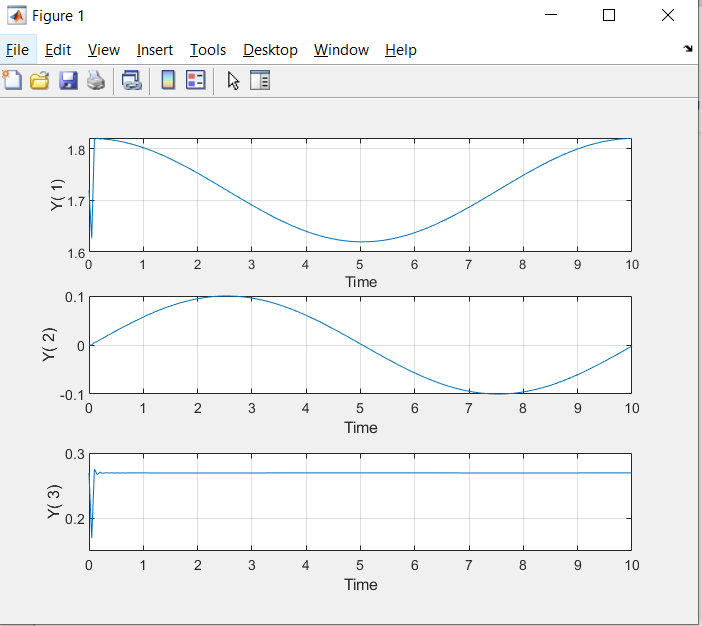


Above are the initial run results, with a circle being almost completed and the x,y,z coordinates shown above. As expected, the X-Y parameters take a sinusoidal path as the circle motion is happening and the z-parameter changes slightly as the arms align to the correct x-y plane for the circular motion but then remains constant after about 1 second. Next, I will try varying the Kp and T parameters:

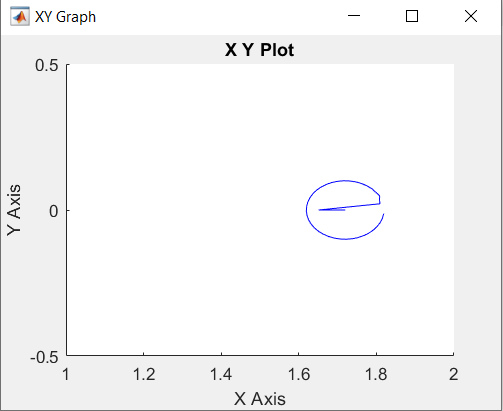
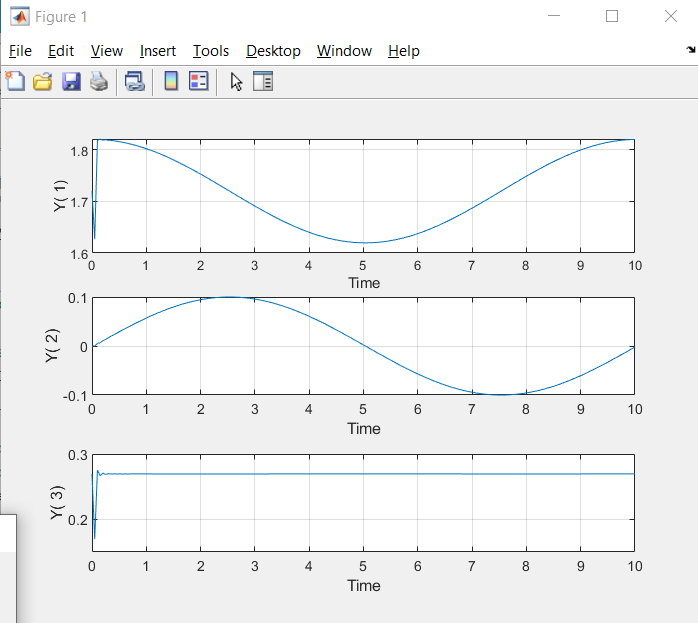
Kp=1 / T=0.05

Kp=30 / T=0.05

Kp=5 / T=0.25

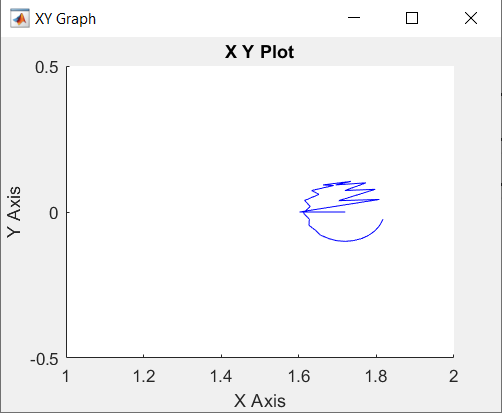
 

Observations:

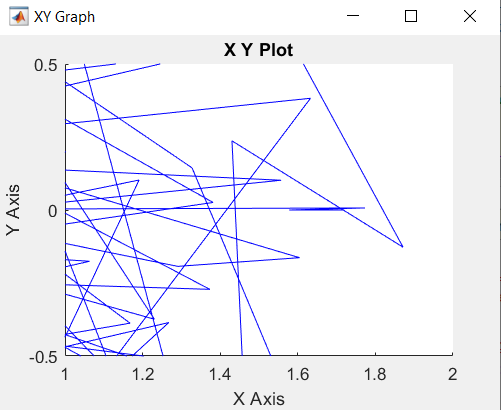
I notice in my above observations that increasing Kp or T decreases the amount of time that the robot takes to get to the first position to start the circle motion. A Kp of 30 or T of 0.25 both seem to make the first initial movement very quick. Reducing the Kp has the drawback of making that first initial motion slower and less of the circular motion is done after the 10 seconds are over.

Also, there does seem to be limits to Kp and T, as when they approach certain values (with the other value at the default value for this problem), there are some skewed plots that arise. Namely, the T after 0.35 starts to get very jagged and the Kp > 40 starts to go all over the place. Care should be taken to make sure that Kp and T values do not lead to these jagged motions.

T > 0.35



Kp > 40



Code:

%% Question 4.2

mdl\_fanuc10L

r=sim('sl\_rrmc2\_Q3');

t=[0:0.05:10];

T2=R.fkine(q);

xyz2=transl(T2);

mplot(t,xyz2(:,1:3));

**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 θ(tf) = 0.5 [rad], in tf = 40ms, or as fast as possible.**

**Assume that the total moment of inertia (motor plus load) is J = 10Jm, where Jm 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 KE, 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.**

Parameters:

Jm = 0.019 oz.in.s2 (taken from Jm = 10\*Jm / Jm = JL = 0.019)

JL = 0.019 oz.in.s2

KE = 6.91 V/KRPM

KT = 9.34 oz.in/A

R = 3.15 ohms

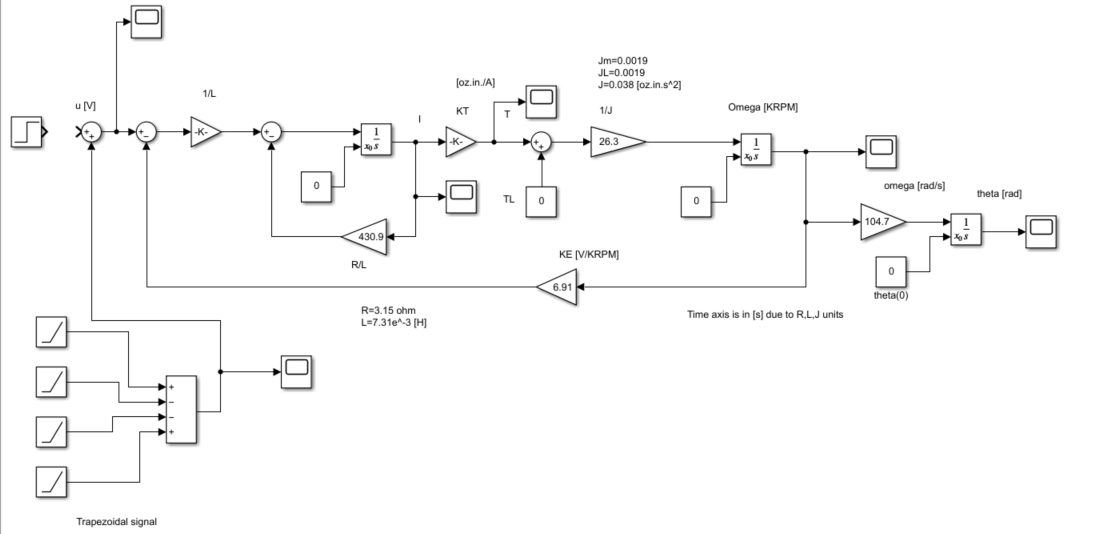
L = 0.00731 mH

Peak Torque=140 oz.in

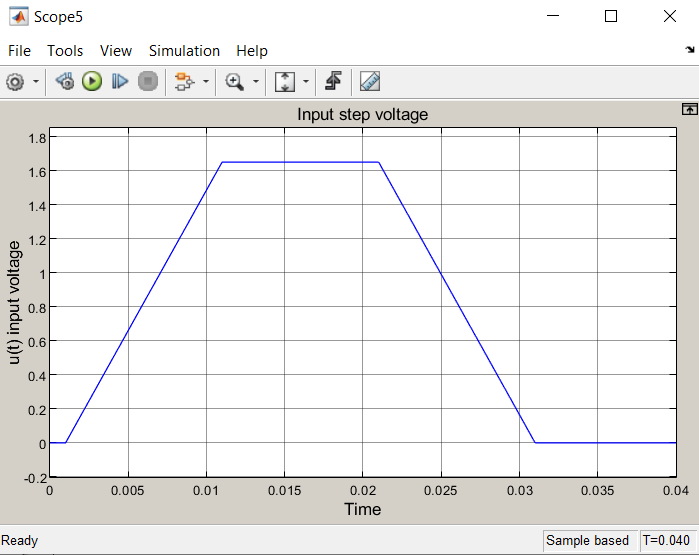
Max Current = 15 A

Stop Time = 0.04 s (40 ms)

Model

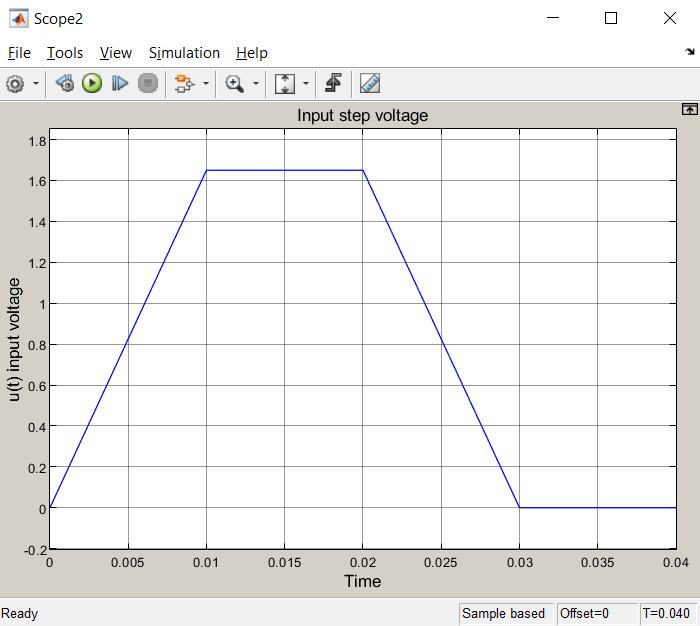


I used the trapezoidal signals with slope 165 at time steps: 0.001 (accelerate), 0.011 (constant speed), 0.021 (decelerate), 0.031 (plateau). There are 10 ms between each step and the acceleration starts at 1 ms. This provided the necessary voltage to accelerate, get to constant speed, decelerate and then stop to the turning angle of 0.5 radians within 40 ms (trapezoidal profile seen below).

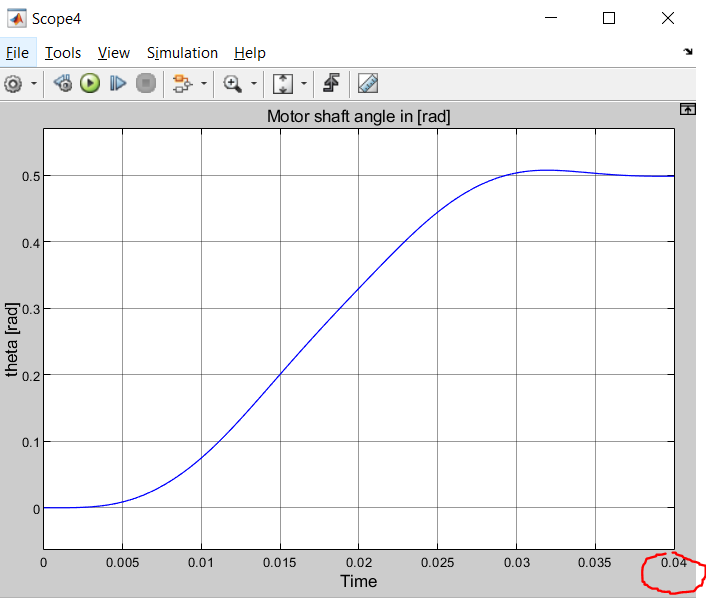


The turning angle overshoot is also very small, and the peak torque and max current values are not exceeded, so this result works as specified. This is open loop so no PID block is used.

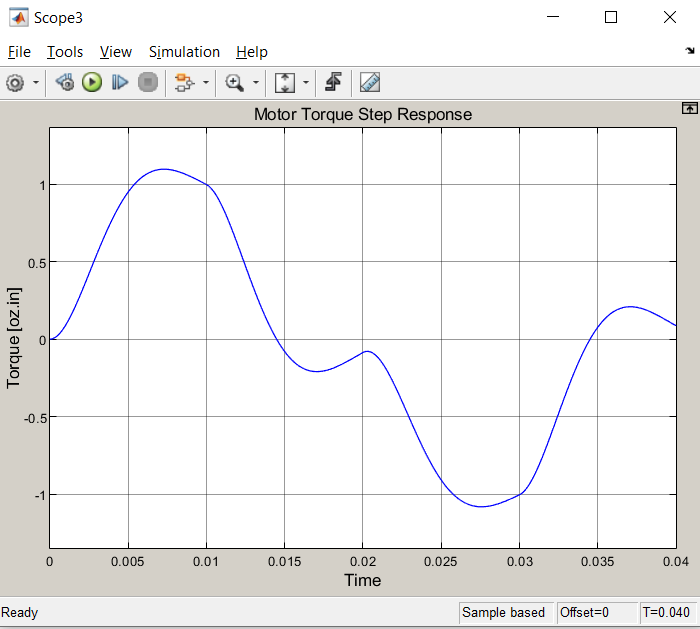
Voltage



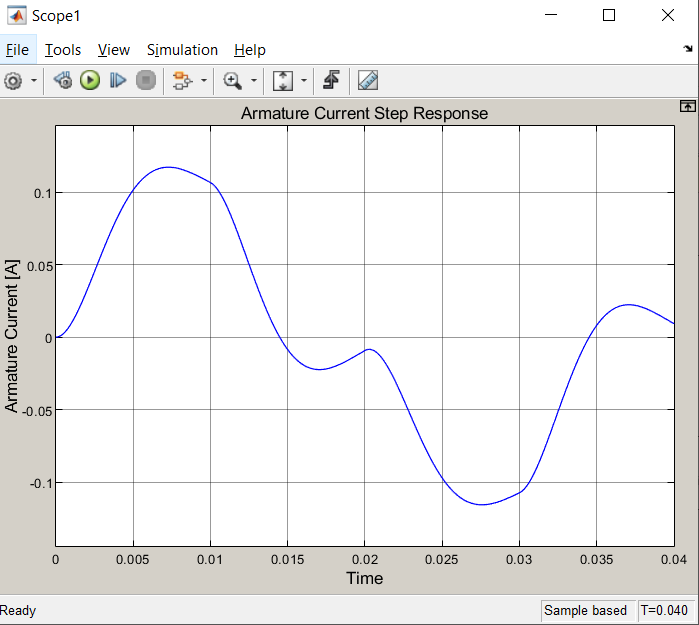
Turning Angle



Peak Torque



Max Current



**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 KE, 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.**

Parameters (same as before):

Jm = 0.019 oz.in.s2 (taken from Jm = 10\*Jm / Jm = JL = 0.019)

JL = 0.019 oz.in.s2

KE = 6.91 V/KRPM

KT = 9.34 oz.in/A

R = 3.15 ohms

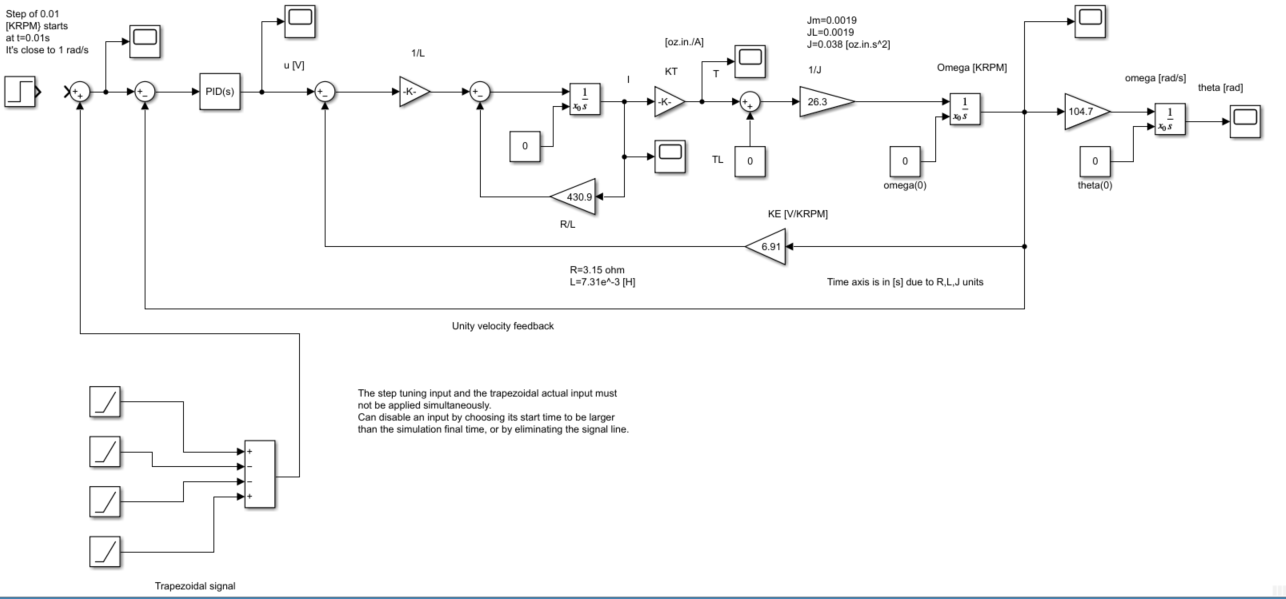
L = 0.00731 mH

Peak Torque=140 oz.in

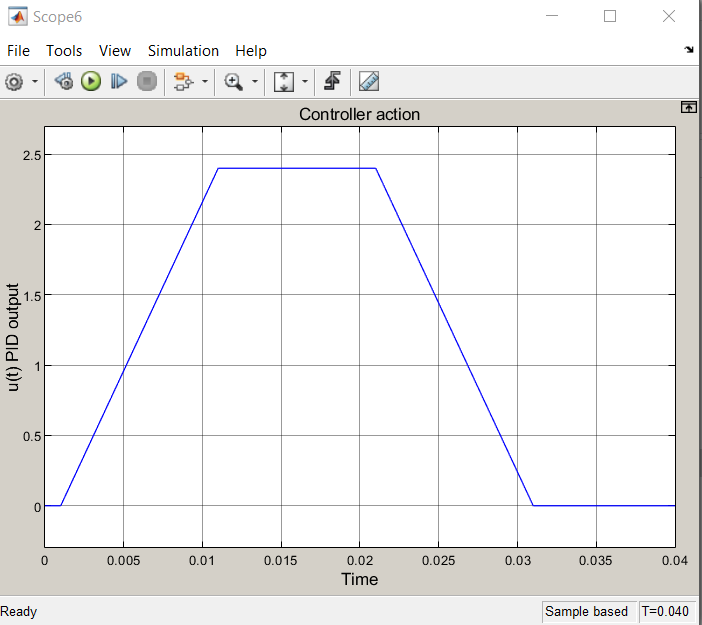
Max Current = 15000 mA

Stop Time = 0.04 s (40 ms)

Model

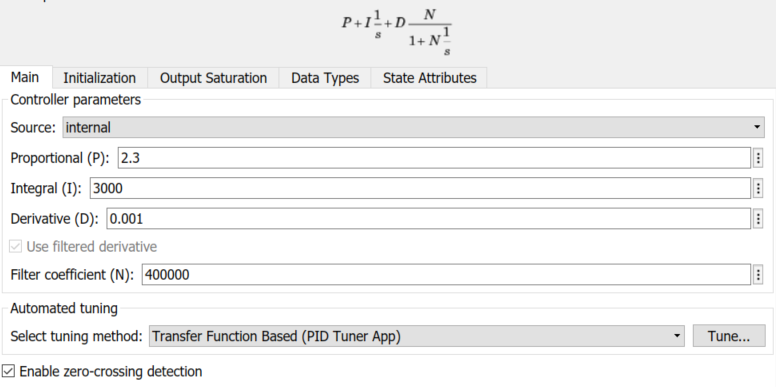


Since the parameters for this question are the same as 5.1, I just added the PID controller and velocity feedback in order to make this closed loop system. I used the trapezoidal signals with slope 240 at time steps: 0.001 (accelerate), 0.011 (constant speed), 0.021 (decelerate), 0.031 (plateau). There are 10 ms between each step and the acceleration starts at 1 ms. This provided the necessary voltage to accelerate, get to constant speed, decelerate and then stop to the turning angle of 0.5 radians within 40 ms (trapezoidal profile seen below).

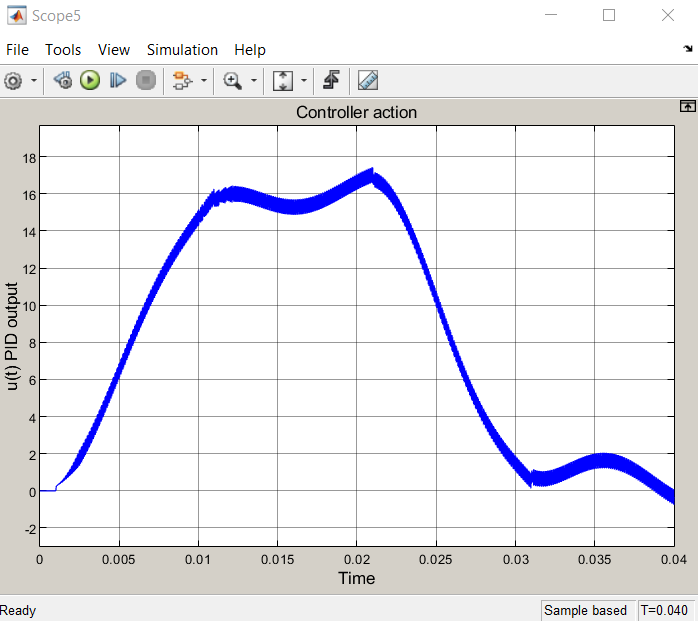


The turning angle overshoot is also very small, and the peak torque and max current values are not exceeded, so this result works as specified. For the PID controller I used values that were close to what was discussed in class. These values allowed for very little overshoot and settled on the final turning angle of 0.5 radians.

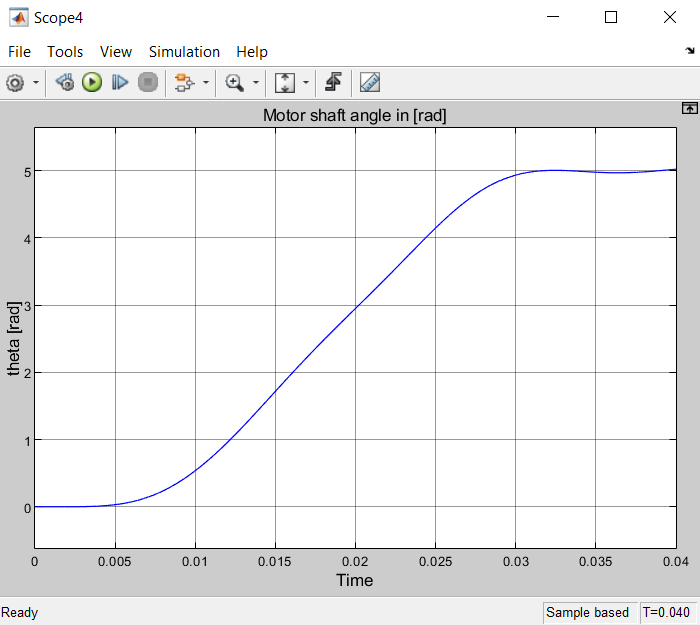
PID Parameters



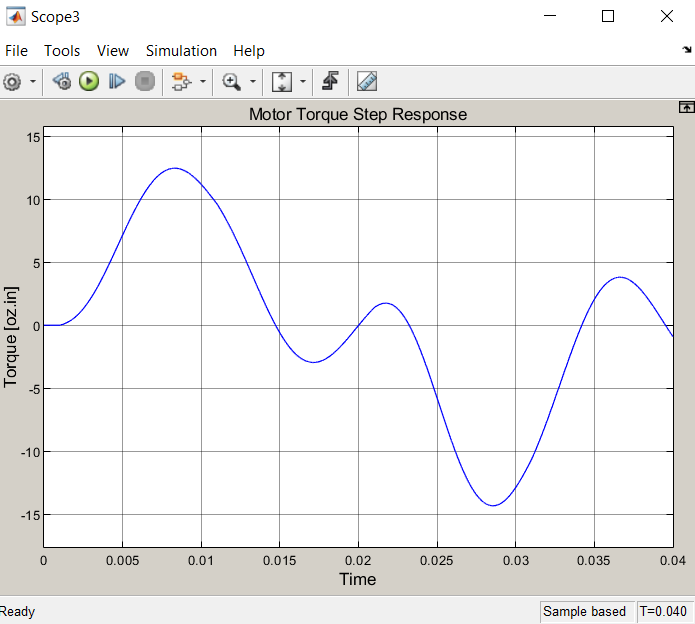
Voltage



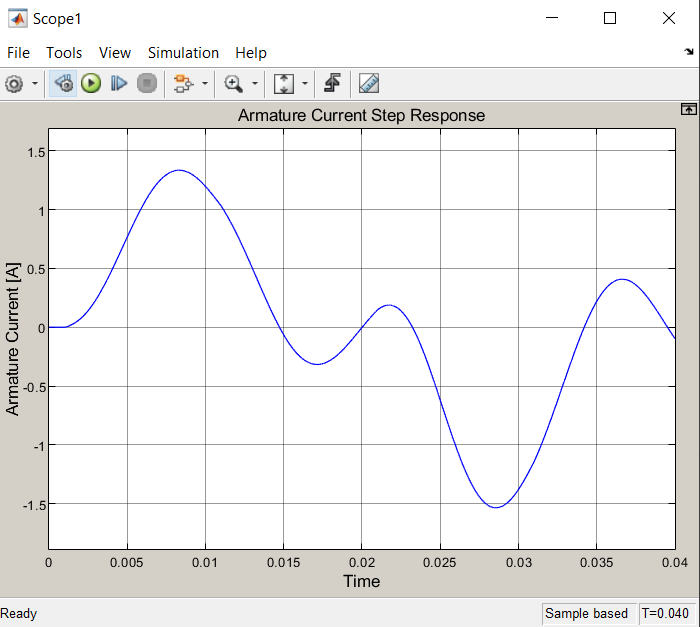
Turning Angle



Peak Torque

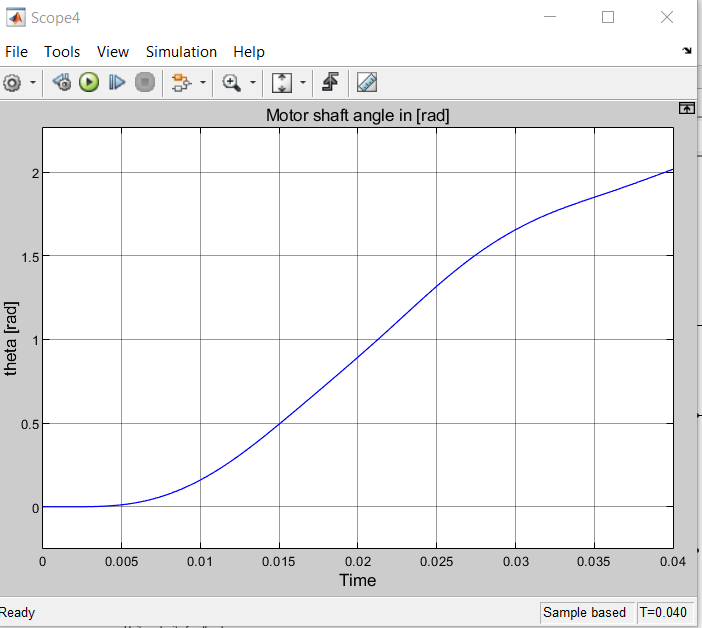


Max Current

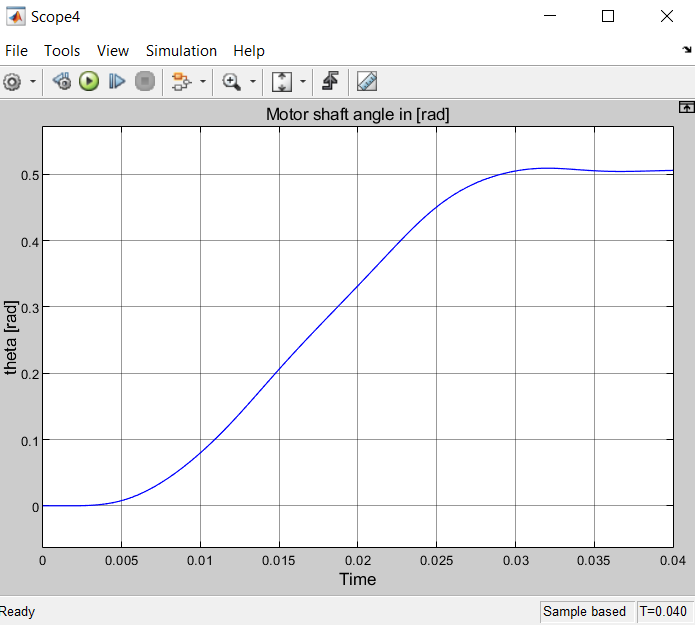


**5.3) In 5.2, inspect the tuned PID control transfer function. If the D gain appear to be very small try to eliminate it altogether, leaving only a PI controller. Compare the performance to that obtained in 5.1.**

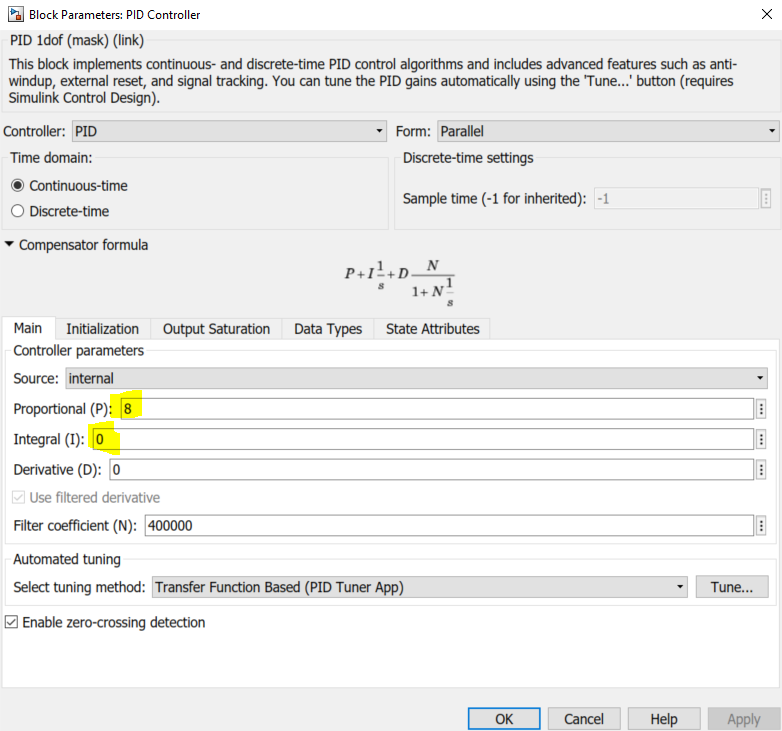
By only changing the D parameter to “0”, I get the below turning angle over time:

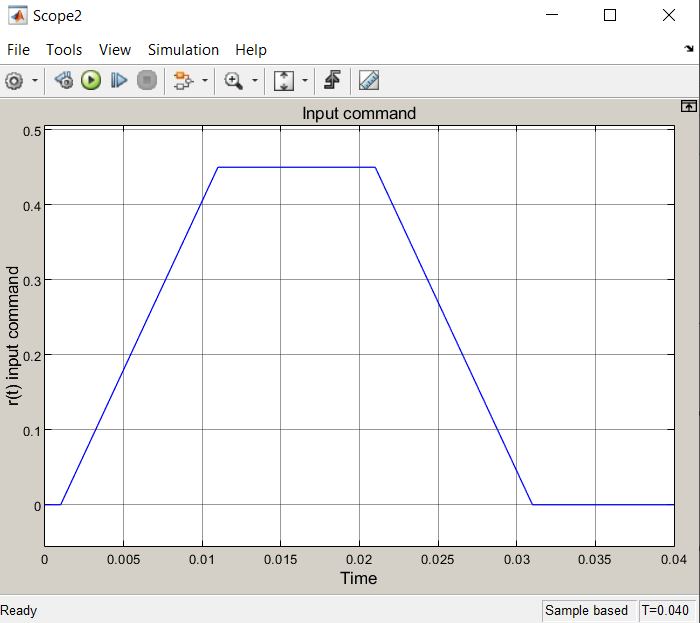


The motor now rotates faster and plateau’s later, making more modications necessary to fit the original request. By making some changes to the P and I parameters as well as the trapezoidal profile, I was able to get the motor working as expected again.



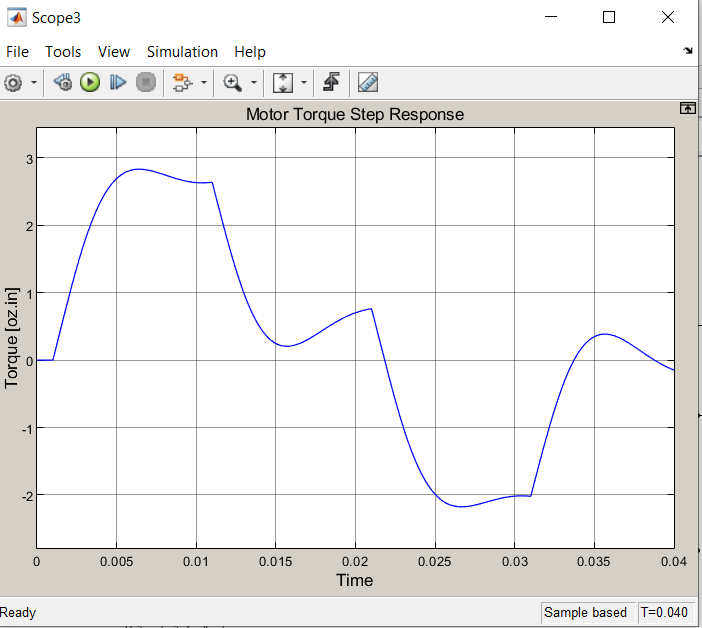
In order to get the above result again, I changed the P value to 8 and the integral to 0 to match the derivative of 0 specified in the problem. By pairing that with trapezoidal slope values of 45 for the same time steps, the motor begins to work as expected again.





This new model requires considerably less voltage than the model in 5.2 (it peaks at ~0.45). Both the peak torque and the max current for this run are scaled down to around 1/4 of the values for 5.2

Peak Torque



Max Current

