MAE 163B PROJECT 2 PART 2 REPORT
DAVID (BLAKE) DEE
106045624

**Intro**

Part two of the trajectory project involved hard coding joint space trajectories for the SCARA arm based on the via points derived in the first phase. However, the via points first had to be translated from the task space to the work space via inverse kinematics. Finally, linear interpolation with parabolic blends were used to stitch the via points together and create a series of points for the joints to fluidly track.

**Inverse Kinematics**

Inverse kinematics was used to translate the via points from the task space to the joint space for later interpolation. In the previous section I had hand solved the inverse kinematics and written a function that would input cartesian position and Z-axis rotation (SCARA has no roll or pitch):

```
fprintf('* ========== Inverse Kinematics ========== \n')
TS = [0 395 0 208];
Rz = TS(1); Px = TS(2); Py = TS(3); Pz = TS(4);
t1 = atan2(Py*(225*cos(t2)+325)-Px*225*sin(t2),Px*(225*cos(t2)+325)+Py*225*sin(t2));
t2 = atan2((1-((Px^2+Py^2-156250)/146250)^2),(Px^2+Py^2-156250)/146250);
t3 = Rz-t1-t2;
d4 = -416+38+Pz;
IK = [t1 t2 t3 d4];

figure()
SCARA.plot(IK, 'workspace', [-1000 1000 -1000 1000 0 1000])
```

In order for loop through this function, I had to both convert the homogeneous matrices of the via points into a 1x4 condensed parameter, as well as transform the via points from the World View {S} to the Base View {B}:

```
% Defining via points in world frame {S}

T_BS = [1 0 0 395; 0 1 0 0; 0 0 1 208; 0 0 0 1]; % world frame {S} in base frame {B}
T_SI = [1 0 0 0; 0 1 0 0; 0 0 1 38; 0 0 0 1]; % initial configuration
T_SF = [1 0 0 60; 0 1 0 0; 0 0 1 0; 0 0 0 1]; % feeder
T_SL = [0 -1 0 45; 1 0 0 -45; 0 0 1 0; 0 0 0 1]; % L1
T_SLU = [0 -1 0 45; 1 0 0 -45; 0 0 1 3; 0 0 0 1] ;% via point above L1
T_SL2 = [1 0 0 -45; 0 1 0 -45; 0 0 1 0; 0 0 0 1]; % L2
T_SL2U = [1 0 0 -45; 0 1 0 -45; 0 0 1 3; 0 0 0 1]; % via point above L2
```

```
% Redefining via points to base frame {B}
T_BI = T_BS * T_SI; % initial configuration
T_BF = T_BS * T_SF; % feeder
T_BL = T_BS * T_SL; % L1
T_BLU = T_BS * T_SLU; % via point above L1
T_BL2 = T_BS * T_SL2; % L2
T_BL2U = T_BS * T_SL2U; % via point above L2

% Converting to IK solvable task space
TI = [395, 0 ,246, 0];
TF = [455, 0, 208, 0];
TL = [440, -45, 208, pi/2];
TLU = [440, -45, 211, pi/2, ];
TL2 = [350, -45, 208, 0];
TL2U = [350, -45, 211, 0];
```

Now that I had the via points arranged to be inputted by the IK function, I created a cell array “viaPoints” containing them and for looped through, generating joint angles/lengths and subsequently storing them in another cell array “vpIK”:

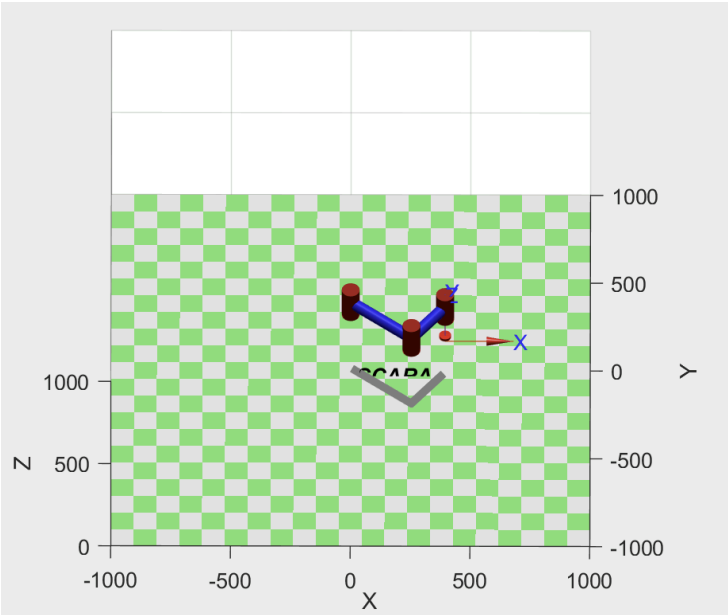
```
% Solve IK for each via point
viaPoints = {TI, TF, TL, TLU, TL2, TL2U};
vpIK = cell(1, length(viaPoints));

for idx = 1:length(viaPoints)
    TS = viaPoints{idx};
    Px = TS(1);Py = TS(2);Pz = TS(3); Rz = TS(4);
    t1 = atan2(Py*(225*cos(t2)+325)-Px*225*sin(t2),Px*(225*cos(t2)+325)+Py*225*sin(t2));
    t2 = atan2((1-((Px^2+Py^2-156250)/146250)^2),(Px^2+Py^2-156250)/146250);
    t3 = Rz-t1-t2;
    d4 = -416+38+Pz;
    IK = [t1 t2 t3 d4];

    vpIK{idx} = IK;
end
```

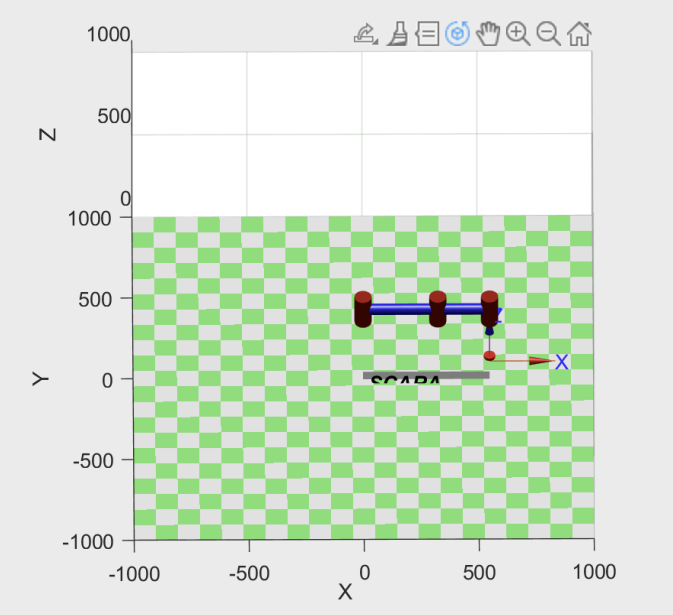
**Joints Kinematic Limits and Verification with 3D Inverse Kinematics Plotting**

Before I could move to interpolating between the joint angles, I needed to set limits on the position, velocity, and acceleration of each joint angle. First, I went back to the link initialization and added a parameter ‘qlim’ to links 1-3, setting minimum and maximum joint rotations per the limits provided in the spec sheet. Although the spec sheet did provide a limit for the fourth prismatic joint, it was causing issues with the 3D plotter, and because my via points did not demand a length outside of these limits, I decided to remove it:

```
fprintf('* ========== Generating SCARA DH Parameters ========== *')

L1 = Link('revolute','d', 416, 'a', 0,'alpha', 0, 'modified', 'qlim', [deg2rad(170) deg2rad(540)])
L2 = Link('revolute','d', 0, 'a', 325,'alpha', 0, 'modified', 'qlim', [deg2rad(145) deg2rad(435)])
L3 = Link('revolute','d', 0, 'a', 225,'alpha', 0, 'modified', 'qlim', [deg2rad(360) deg2rad(1080)])
L4 = Link('prismatic','a',0,'alpha',0,'theta',0,'modified')
tool = transl(0,0, -38)
```

In order to verify these limits, I hand inputted positions into my IK solver and plotted them in 3D. If everything was working correctly, the World Frame {S} position would form a right angle with the SCARA links (by design), and an x displacement of an entire arm's length would be straight:




**Linear Functions with Parabolic Blends**

Next, a linear function with parabolic blending was used to interpolate between the via point joint angles (now in the joint space). In order to do this, I created a separate function "traj.m" that input two joint space angles (1x4 matrices): an initial configuration and a target configuration. The function would interpolate between the configurations, and spit out a (3) N x 4 matrices containing joint positions, velocities, and accelerations at each nth time step (I chose 50):

```
function [q_full, qd_full, qdd_full] = traj(q_initial, q_final)
q0 = q_initial;
qf = q_final;
t0 = 0;
tf = 5;
N = 50;
t = linspace(t0,tf,N);
```

Next, a maximum acceleration was chosen to calculate the blending times. Per the spec sheet, we assumed 50 deg/s/s for the revolutes, and 50mm/s/s for the prismatic:

```
% defining max acceleration
maxA = deg2rad(50);
des_qdd = [maxA, maxA, maxA, 50];
```

Finally, equations of motion describing each of the three portions of the interpolation - the first parabolic blend, the linear section, and the second parabolic blend - were calculated. Given the initial joint positions for a single joint, as well as the assumption that each initial and final velocity was zero (to place the chip), I used a system of equations governing these blending relationships to solve for each of the constant coefficients for the equations of motion. With these equations, I could plug in the 50 time steps, append them to each other, and return (3) 50x4 matrices - “q_full, qd_full, and qdd_full” - describing the motion of each joint.

I then used a for-loop to iterate through all of the initial and final configurations for all the joints and generate position, velocity, and acceleration interpolations for all of them:

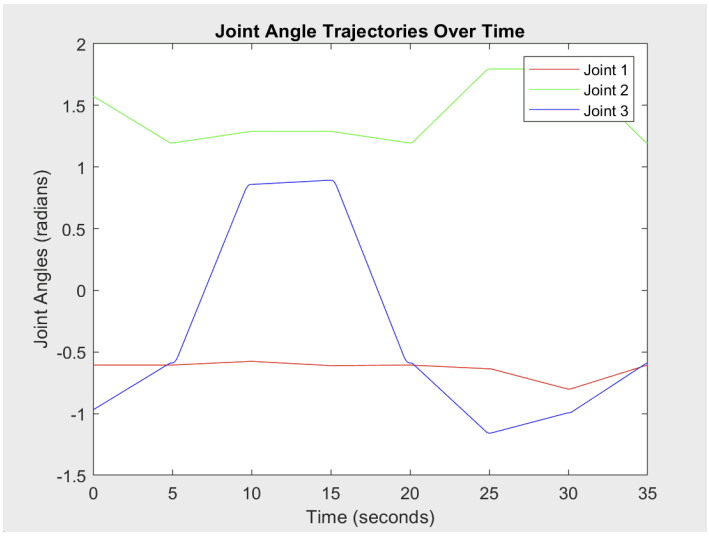
```
for i = 1:4 % We have 4 joints
    qdd = des_qdd(i);
    tb = 0.5*(tf-t0) - 0.5*sqrt(qdd^2*(tf-t0)^2 - 4*abs(qdd)*abs(qf(i)-q0(i)))/abs(qdd);
    if q0(i) < qf(i)
        ab0 = [1 t0 t0^2;0 1 2 * t0;0 0 2] \ [q0(i) 0 qdd]'; % Coefficient for first blend
        abf = [1 tf tf^2;0 1 2 * tf;0 0 2] \ [qf(i) 0 -qdd]'; % Coefficient for second blend
    else
        ab0 = [1 t0 t0^2;0 1 2 * t0;0 0 2] \ [q0(i) 0 -qdd]'; % Coefficient for first blend
        abf = [1 tf tf^2;0 1 2 * tf;0 0 2] \ [qf(i) 0 qdd]'; % Coefficient for second blend
    end
    qb1 = ab0(1) + ab0(2) * tb + ab0(3) * tb^2;
    qb2 = abf(1) + abf(2) * (tf - tb) + abf(3) * (tf - tb)^2;
    a = [1 tb;1 (tf - tb)] \ [qb1;qb2]; % Coefficient for linear region
    % first parabolic region
    t11 = t((t0<=t) & (t<=t0+tb));
    q = ab0(1) + ab0(2)*t11 + ab0(3)*t11.^2;
    qd  = ab0(2) + 2*ab0(3)*t11;
    qdd = 2*ab0(3)*ones(size(t11));
    % Linear region
    t22 = t((t0+tb<t) & (t<tf-tb)); % linear region
    q = [q, a(1) + a(2)*t22];
    qd = [qd, a(2).*ones(size(t22))];
    qdd = [qdd, zeros(size(t22))];
    % second parabolic region
    t33 = t((tf-tb<=t) & (t<=tf));
    q   = [q, abf(1) + abf(2)*t33 + abf(3)*t33.^2];
    qd  = [qd, abf(2) + 2*abf(3)*t33];
    qdd = [qdd, 2*abf(3)*ones(size(t33))];
    q_full = [q_full q'];
    qd_full = [qd_full qd'];
    qdd_full = [qdd_full qdd'];
end
```

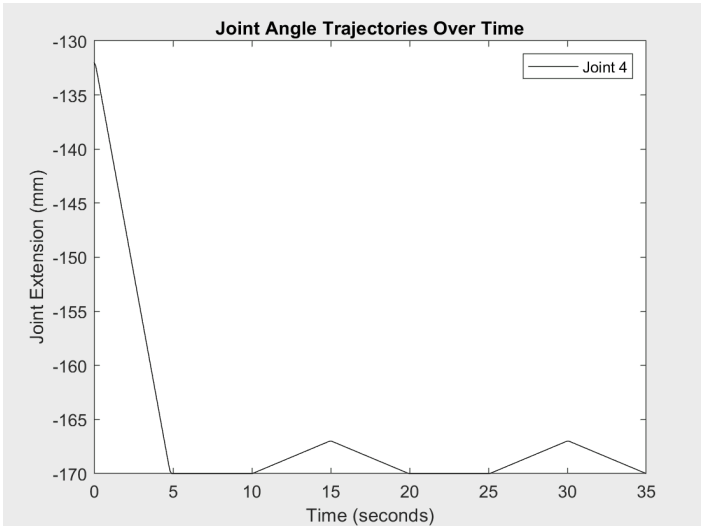
I then called “traj.m” on 7 pairs of via points, storing the results of each in their own defined matrix and then appending them all together in “q_animation”:

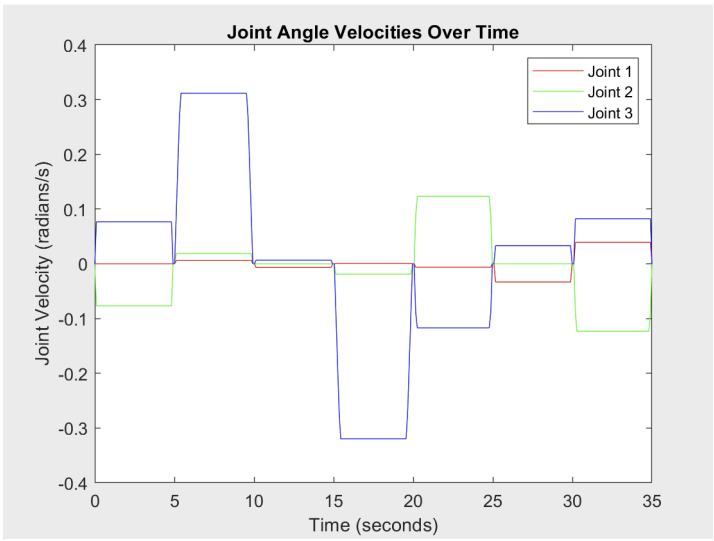
```
%% Solving trajectories

% initial configuration to feeder
[q_I_F, qd_I_F, qdd_I_F] = traj(cell2mat(vpIK(1)), cell2mat(vpIK(2)));

% feeder to L1
[q_F_L, qd_F_L, qdd_F_L] = traj(cell2mat(vpIK(2)), cell2mat(vpIK(3)));

% L1 up (dropping chip)
[q_L_U, qd_L_U, qdd_L_U] = traj(cell2mat(vpIK(3)), cell2mat(vpIK(4)));

% L1 up to feeder
[q_U_F, qd_U_F, qdd_U_F] = traj(cell2mat(vpIK(4)), cell2mat(vpIK(2)));

% feeder to L2
[q_F_L2, qd_F_L2, qdd_F_L2] = traj(cell2mat(vpIK(2)), cell2mat(vpIK(5)));

% L2 up (dropping chip)
[q_L2_U, qd_L2_U, qdd_L2_U] = traj(cell2mat(vpIK(5)), cell2mat(vpIK(6)));

% L2 up to feeder
[q_L2_F, qd_L2_F, qdd_L2_F] = traj(cell2mat(vpIK(6)), cell2mat(vpIK(2)));

q_animation = [q_I_F; q_F_L; q_L_U; q_U_F; q_F_L2; q_L2_U; q_L2_F];
qd_animation = [qd_I_F; qd_F_L; qd_L_U; qd_U_F; qd_F_L2; qd_L2_U; qd_L2_F];
qdd_animation = [qdd_I_F; qdd_F_L; qdd_L_U; qdd_U_F; qdd_F_L2; qdd_L2_U; qdd_L2_F];
```

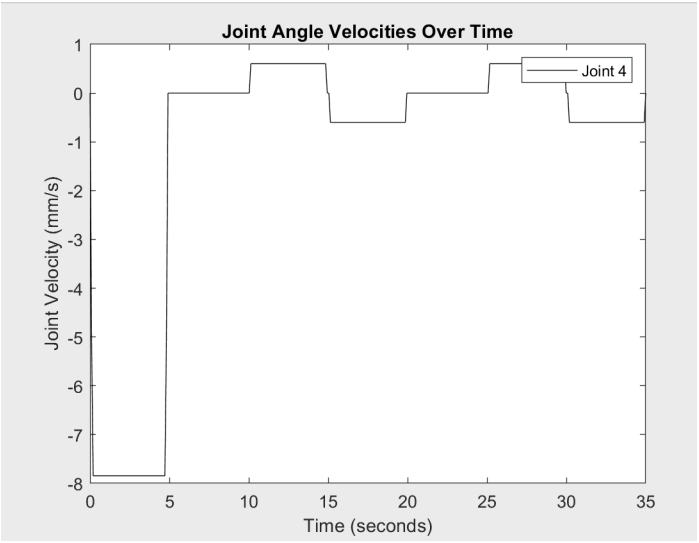
**Trajectory Results**

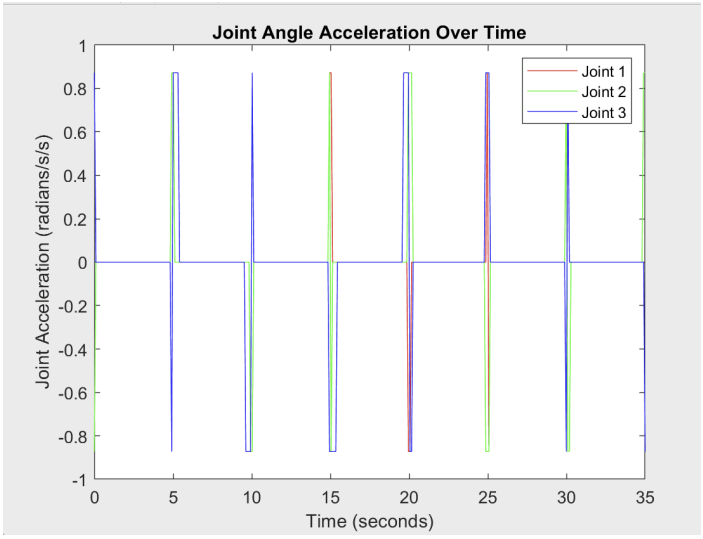
The joint angles for trajectories are shown below. Although on the 35 time second horizon the trajectories look sharp, on the 5 second time horizon they are indeed parabolic blends:

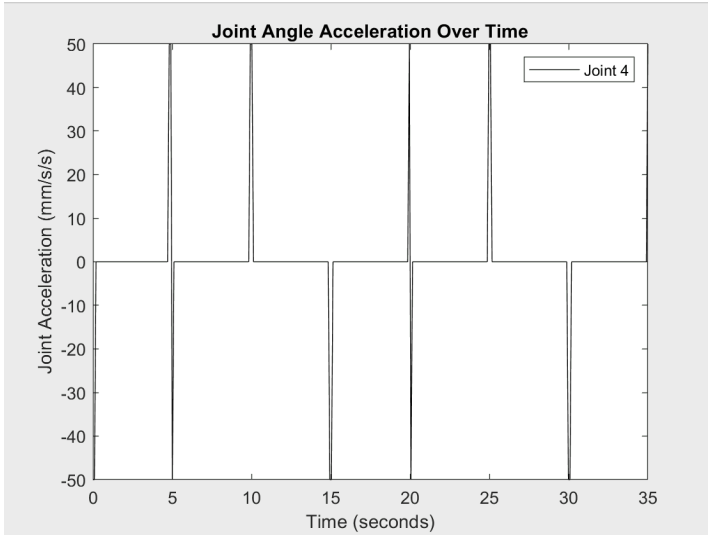

Joint Angle Trajectories Over Time
Joint 1
Joint 2
Joint 3
Joint Angles (radians)
Time (seconds)


Joint Angle Trajectories Over Time
Joint 4
Joint Extension (mm)
Time (seconds)


Joint Angle Velocities Over Time
Joint 1
Joint 2
Joint 3
Joint Velocity (radians/s)
Time (seconds)

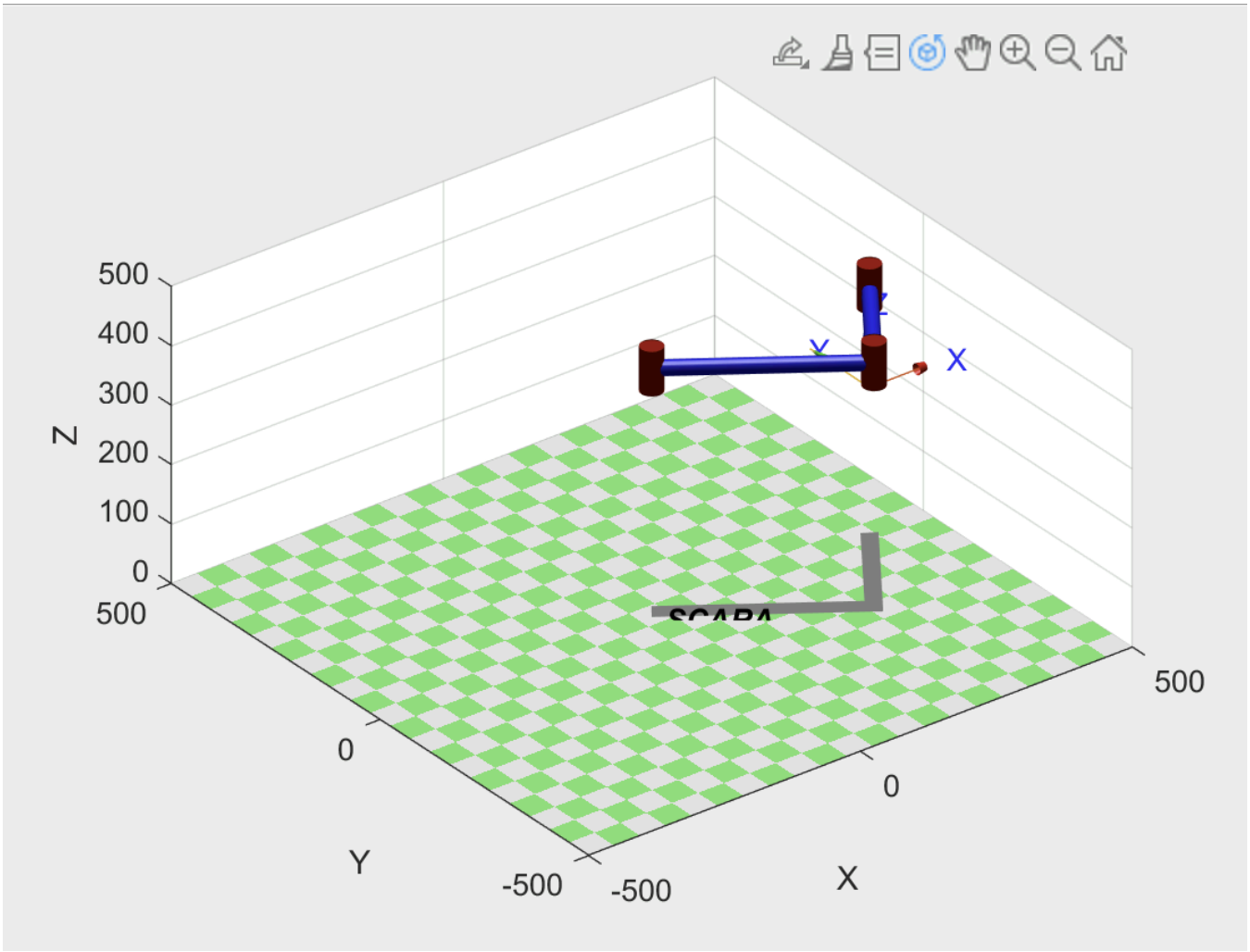

Joint Angle Velocities Over Time
Joint 4
Joint Velocity (mm/s)
Time (seconds)


Joint Angle Acceleration Over Time
Joint 1
Joint 2
Joint 3
Joint Acceleration (radians/s/s)
Time (seconds)


Joint Angle Acceleration Over Time
Joint 4
Joint Acceleration (mm/s/s)
Time (seconds)

**Full Animation**

Included in bruinlearn submission! Screenshots below. Although the motion is quite slow, you can clearly see the end effector hitting every via point and rotating to the correct orientation as necessary. If I wanted to make the motion faster, I could decrease the length of time between each interpolation. However, for the purposes of animation I chose to have each trajectory be 5 seconds.




**Discussion of Outcomes**

Overall, I'm very pleased with the results of the trajectory generation and animation. The graphs and animation came out relatively clean and intuitively made sense. If I were to change anything moving forward, I would increase the distance between the via points to exaggerate the motion of the animation (even though in real life this would be less efficient and undesirable). I would also speed up the motion of the end effector to show a more realistic velocity profile for the SCARA Mitsubishi and showcase it's speed.