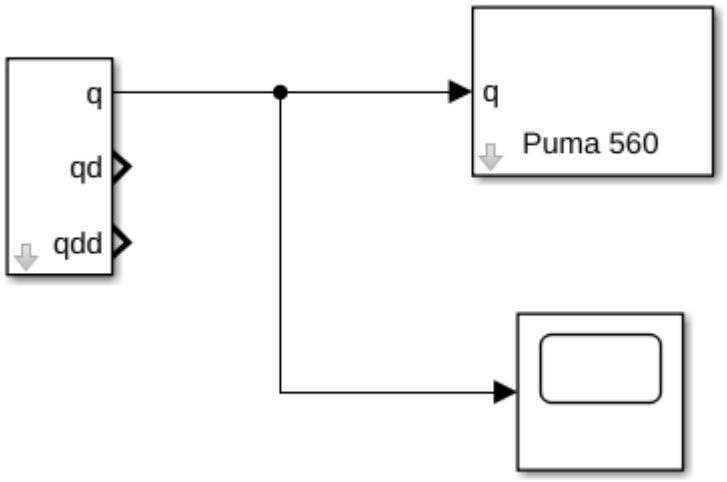
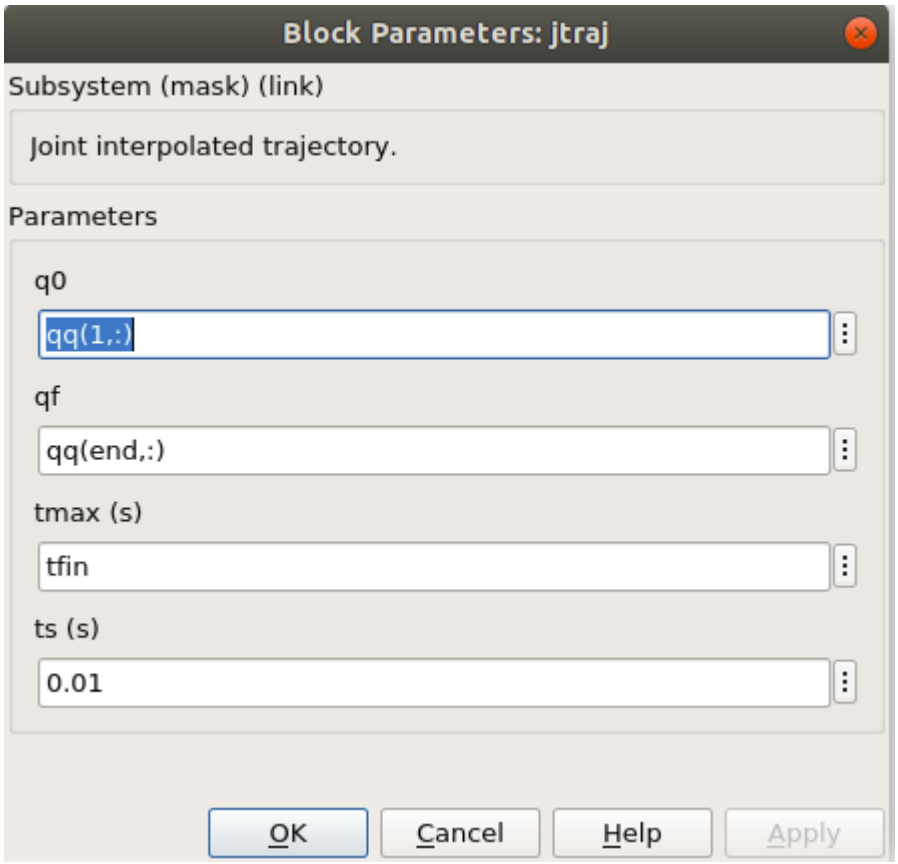
MATLAB EXERCISES
Sarim Mehdi

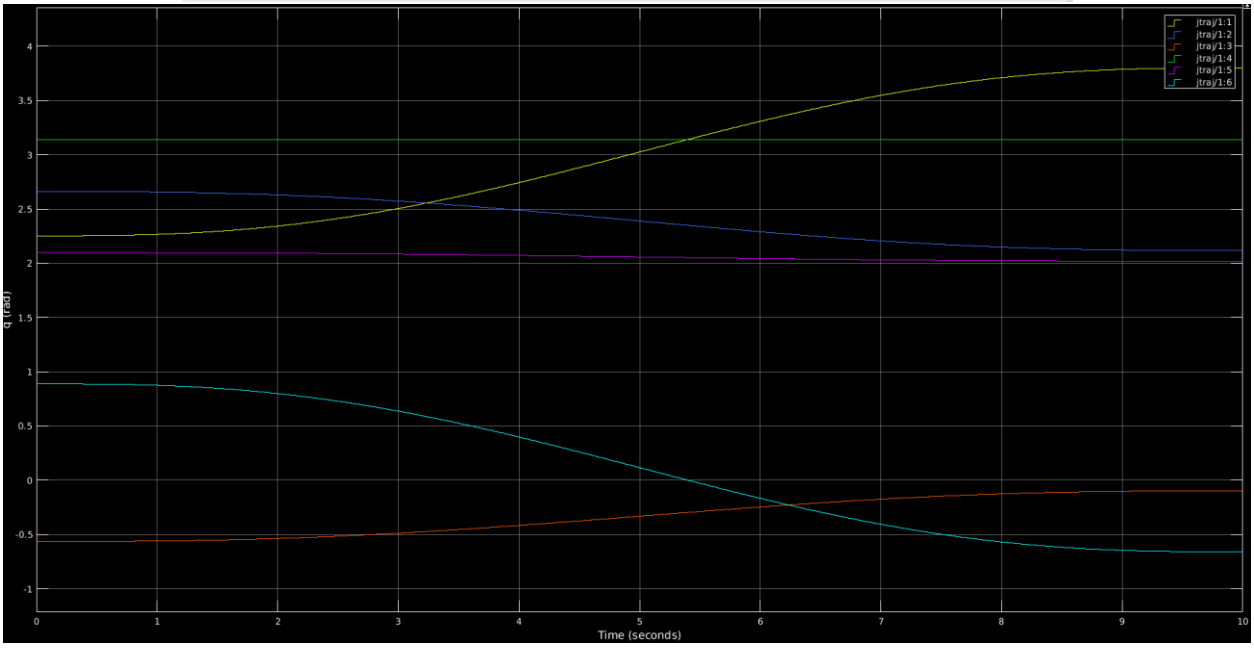
EXERCISE 1 (TRAJECTORY GENERATION & SINGULARITY)

For line trajectory, the following Simulink diagram was used:

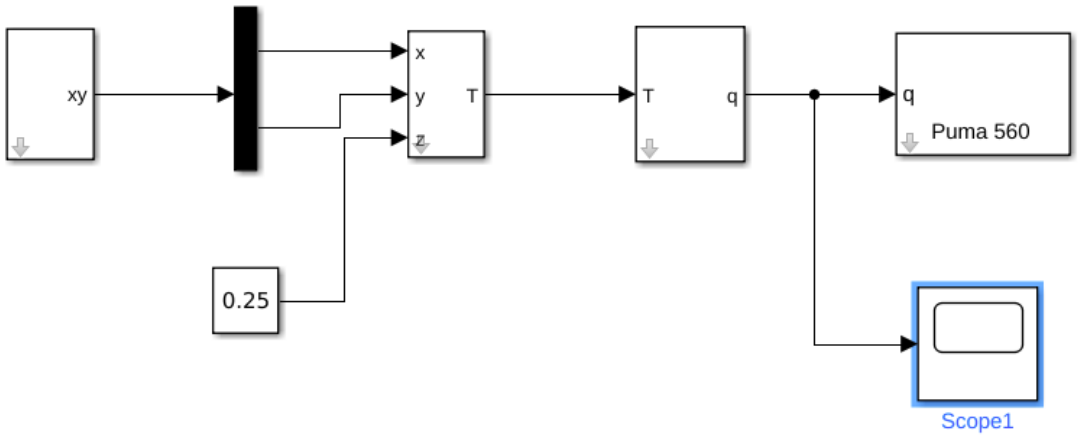


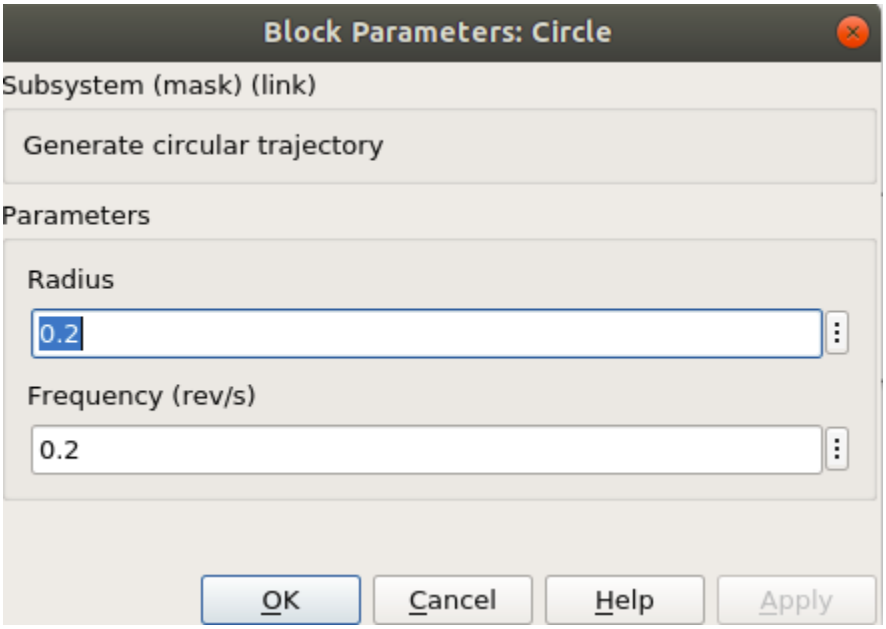

```
%MATLAB CODE
T1 = transl(0.6, -0.5, 0.0); % START
T2 = transl(0.4, 0.5, 0.2);    % DESTINATION
res=20;
TTl=ctraj(T1,T2,res);
qq=ikine6s(p560,TTl);
```

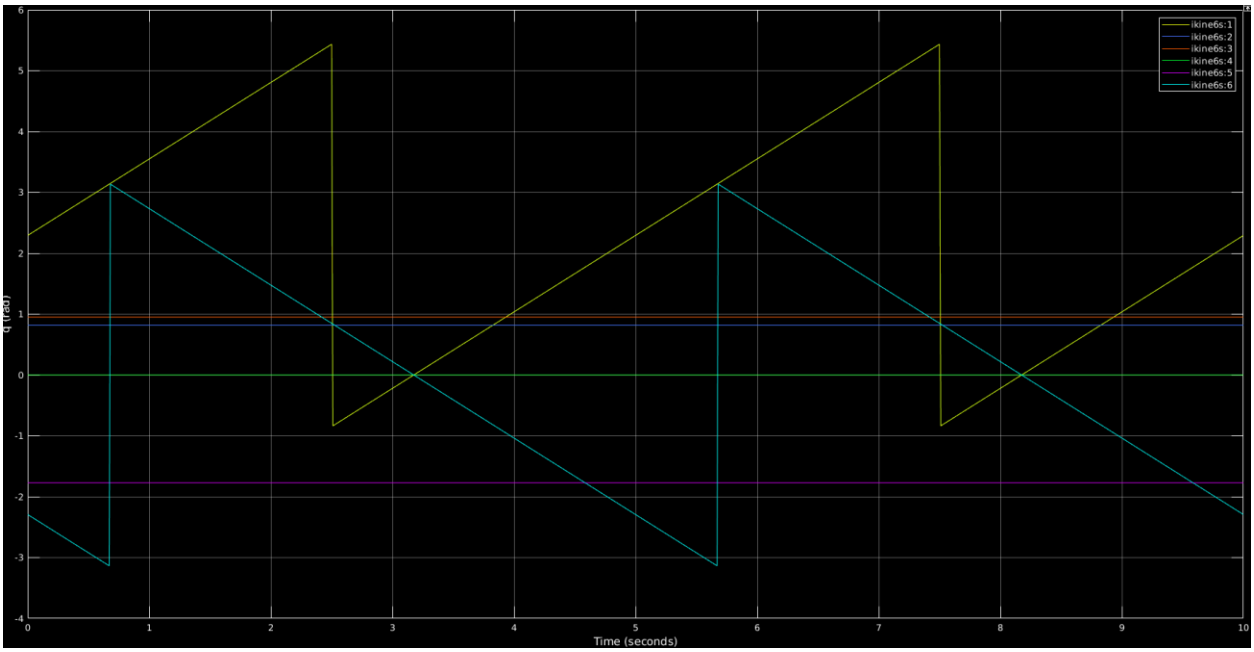

Block Parameters: jtraj
Subsystem (mask) (link)
Joint interpolated trajectory.
Parameters
q0
qq(1,:)
qf
qq(end,:)
tmax (s)
tfin
ts (s)
0.01
OK
Cancel
Help
Apply


jtraj/1:1
jtraj/1:2
jtraj/1:3
jtraj/1:4
jtraj/1:5
jtraj/1:6
q (rad)
Time (seconds)

For circle trajectory, the following Simulink diagram was used:







Convergence of inverse kinematics doesn’t happen in case of three singularities: wrist, elbow and shoulder singularity. To show singularity, I compute the manipulability measure.

```
%MATLAB CODE TO GENERATE SINGULARITY PLOTS (T1 AND T2 ARE INITIAL AND FINAL WORKSPACE
%COORDINATES. THEY ARE MENTIONED FOR EACH TYPE OF SINGULARITY BELOW)
t_res = 0.1;
t = [0:t_res:10];
Ts = ctraj(T1, T2, length(t));
q = p560.ikine6s(Ts);
m = p560.maniplty(q);
[rows, cols] = size(q);
colors = ['b', 'r', 'y', 'g', 'c', 'm'];
figure;
subplot(2,1,1);
xlabel('Time (s)');
ylabel('Joint coordinates (rad)');
hold on
plot(t, q(:,1))
for i=2:cols
   hold on; plot(t, q(:,i), colors(i))
end
legend('q1', 'q2', 'q3', 'q4', 'q5', 'q6')
xlabel('Time (s)');
ylabel('Joint coordinates (rad)');
subplot(2,1,2);
hold on; plot(t, m)
legend('manipulability')
```

Wrist singularity:

T1 = SE3(0.5, 0.3, 0.44) * SE3.Ry(pi/2);
T2 = SE3(0.5, -0.3, 0.44) * SE3.Ry(pi/2);

At time 3.4s, q5 is almost zero and q4 and q6 have a sudden increase. This is also at the exact time when the manipulability drops to a very low value. Axes of joints 4 and 6 become coincident. Robot cannot move in the direction of axis of joint 5 In order for the endpoint to follow a line through the singularity, joints 4 and 6 must simultaneously rotate 90 degrees in direction opposite to joint 5, at the singularity.

Elbow singularity:

T1 = SE3(0, -0.3, 0) * SE3.Ry(pi/2);
T2 = SE3(-0.823, -0.3, 0) * SE3.Ry(pi/2);

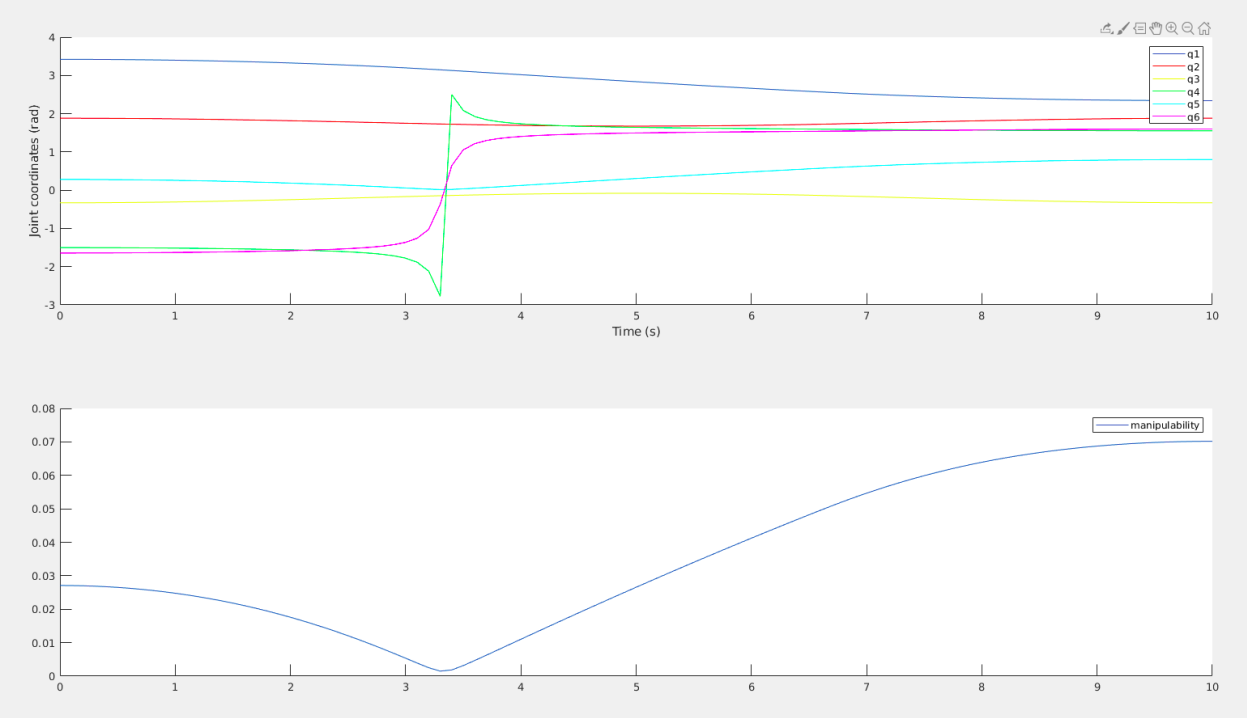
It occurs when the arm is fully stretched. We can see the manipulability drop as q3 drops to a value close to -1. In this singularity, the wrist center (where axes of joints 4, 5 and 6 intersect) lies on the plane passing through axes of joints 2 and 3.

Shoulder singularity:

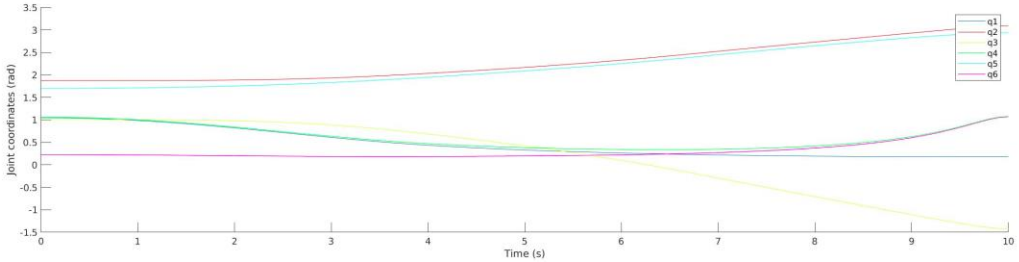
T1 = SE3(0, -0.3, 0.44) * SE3.Ry(0.42);
T2 = SE3(-0.154, -0.15, 0.741) * SE3.Ry(0.42);

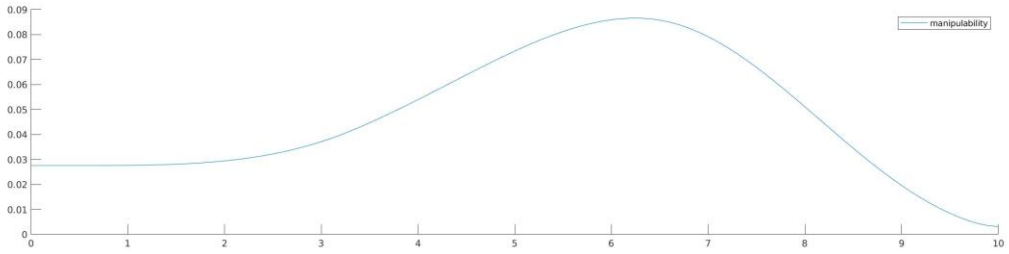
It occurs when the center of the robot wrist lies in the plane passing through the axes of joints 1 and 2. Joints 1 and 4 must rotate in opposite direction. Both joints 4 and 6 drop from 3.14 to -3.14 at singularity.

WRIST SINGULARITY GRAPH:

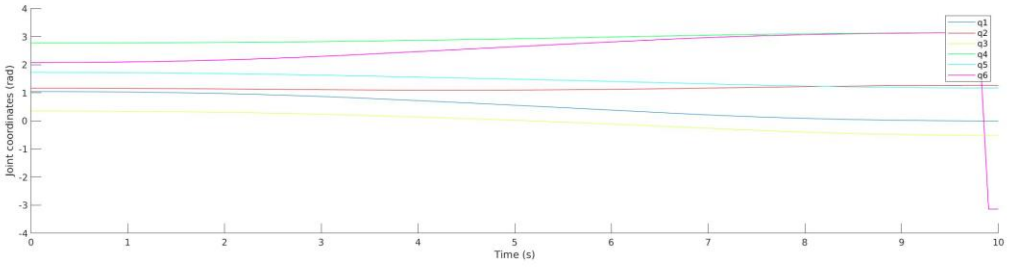


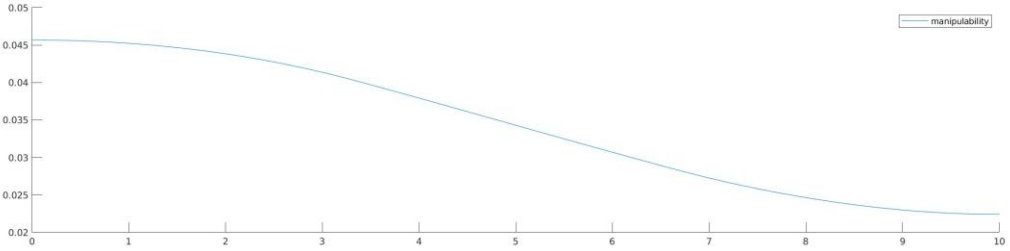
ELBOW SINGULARITY GRAPH:





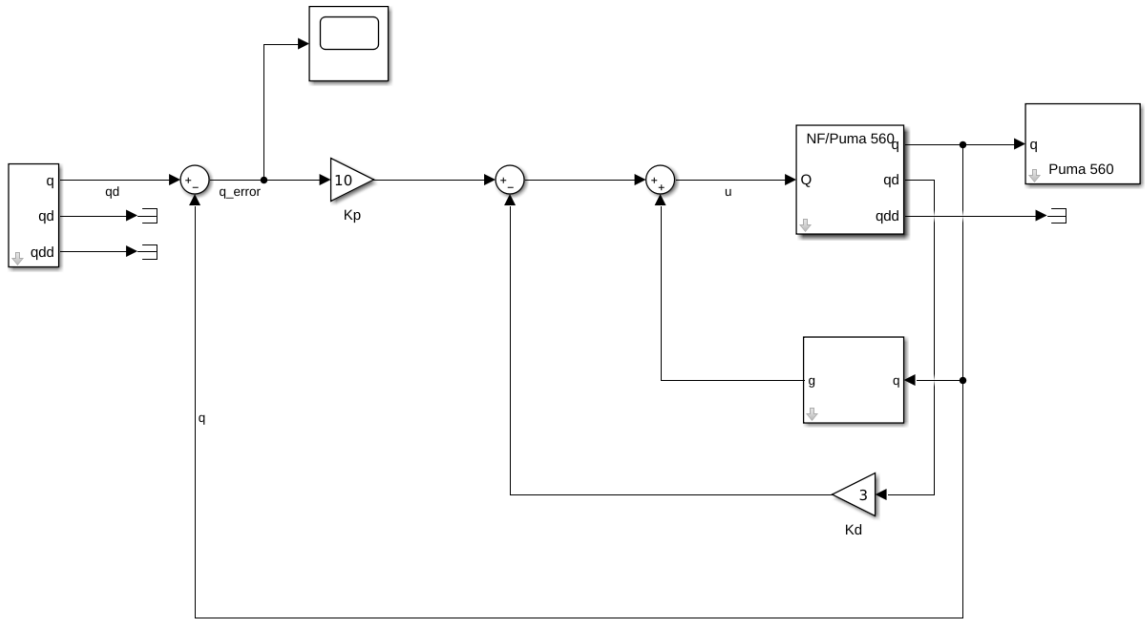
SHOULDER SINGULARITY GRAPH:





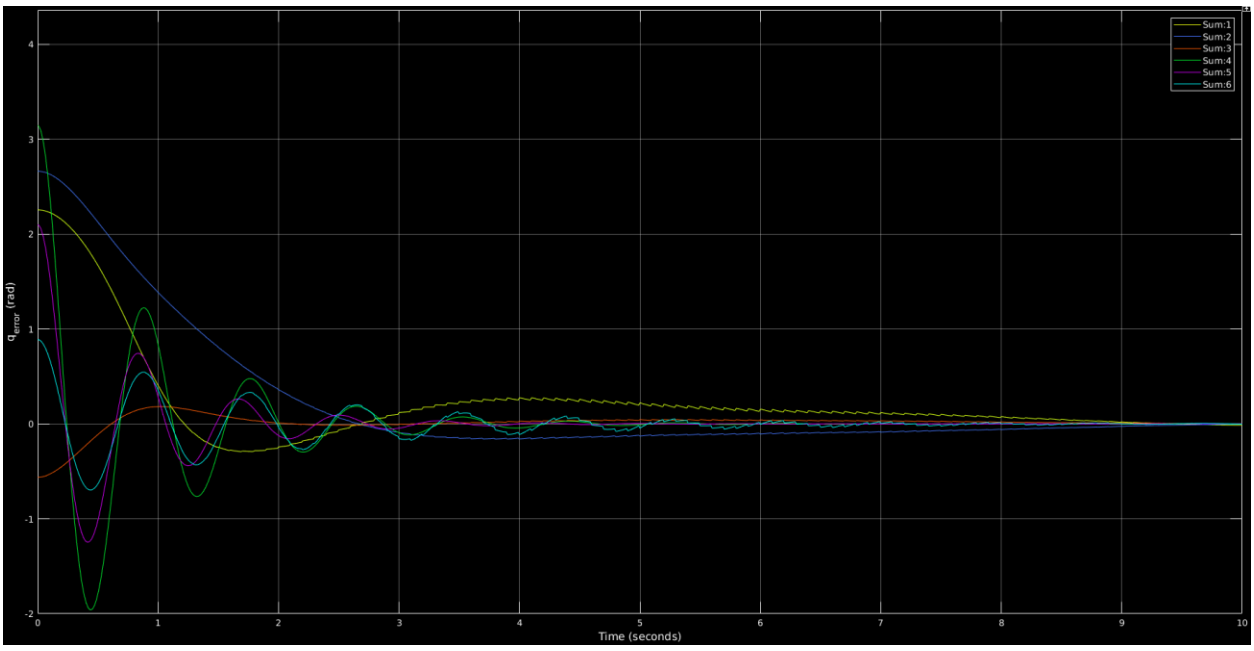
EXERCISE 2 (PD+GRAVITY COMPENSATION)

Following Simulink diagram was used:



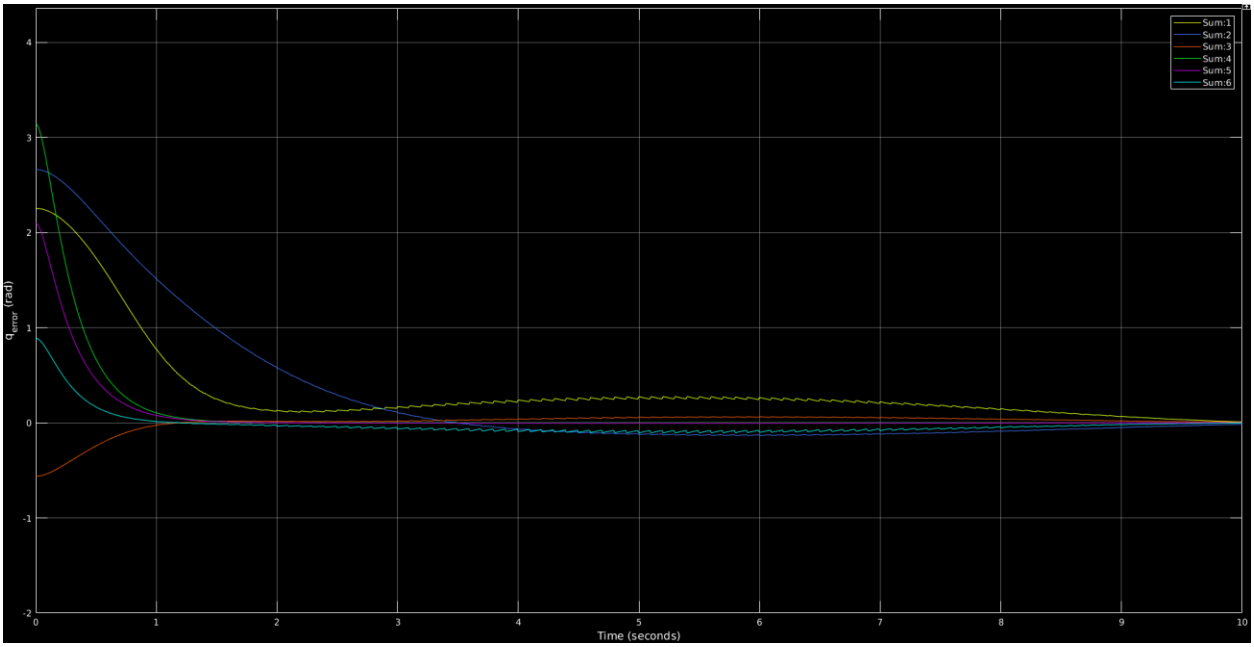
Kp = 10, Kd = 0

Joint Coordinate error:



Kp = 10, Kd = 3

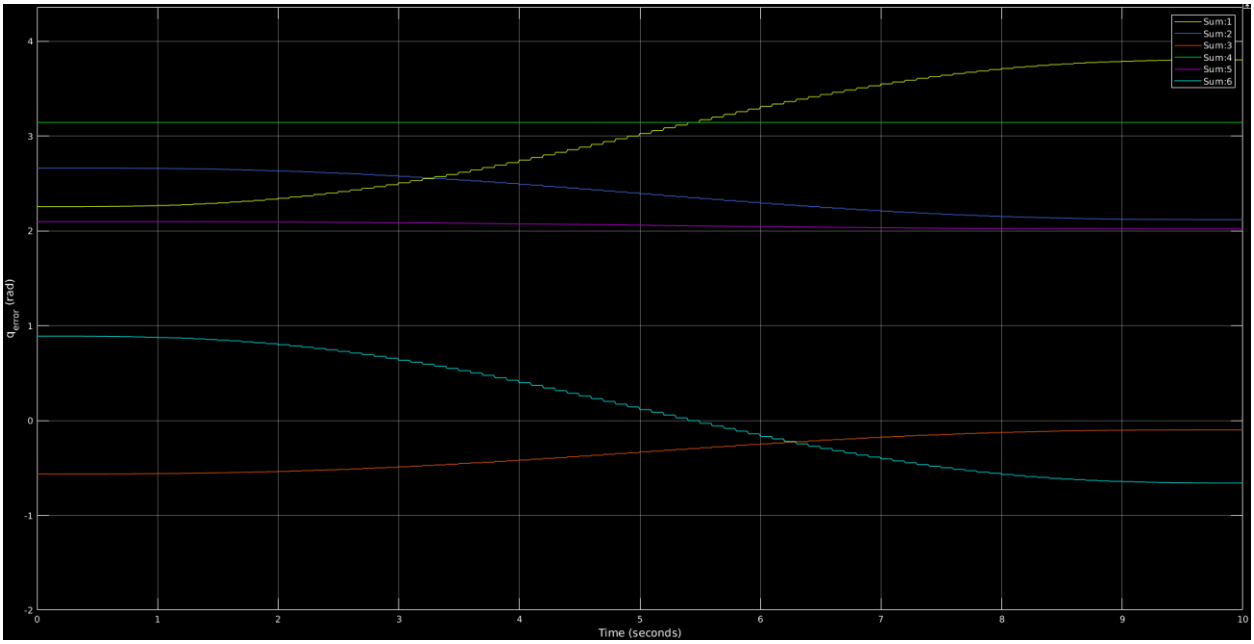
Joint Coordinate error:



We can see that adding Kd allows the graph to smooth out and increases the rate of convergence and decreases the amount of overshoot.

Kp = 0 Kd = 3

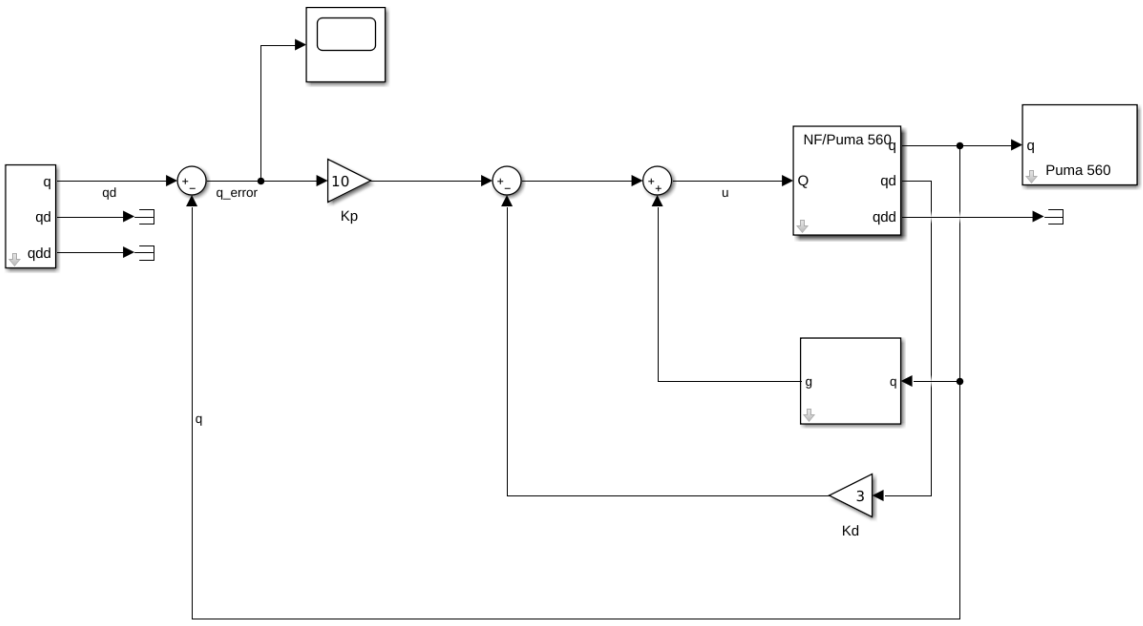
Joint Coordinate error:



We see that having no Kp means no convergence to zero error. This makes sense as Kp is responsible for convergence to zero error whereas Kd only has an effect on the rate of convergence. This is why, in the very first graph where Kp was 10 and Kd was 0, the graph still converged to zero error but it overshot the equilibrium a lot.

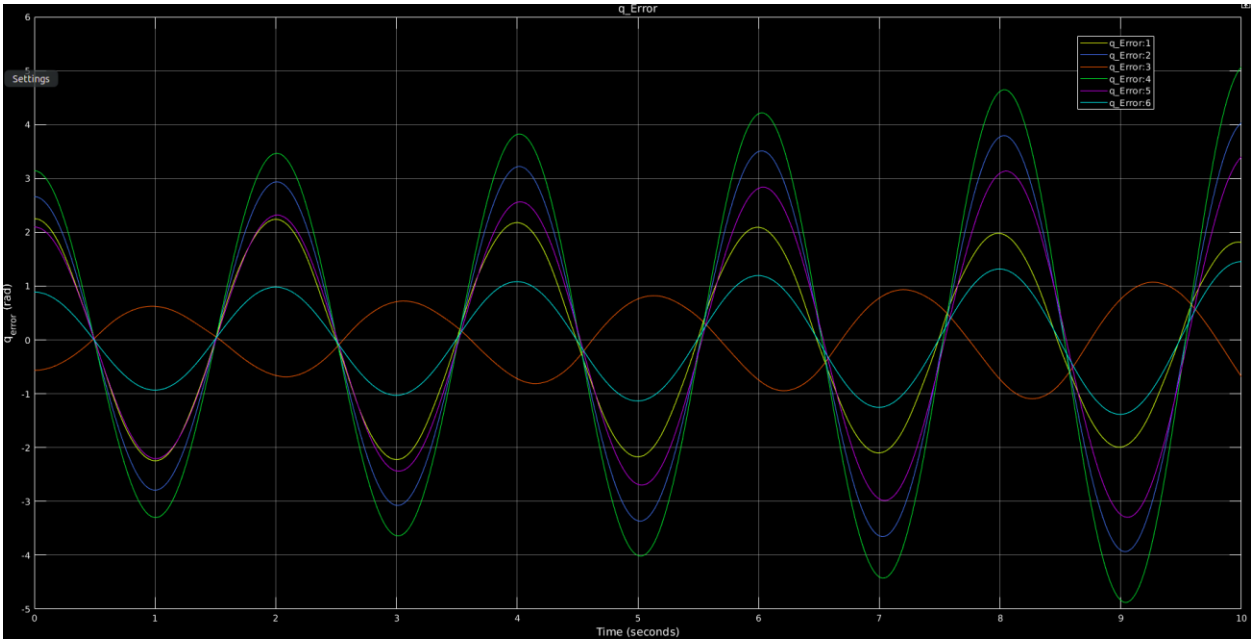
EXERCISE 3 (INVERSE DYNAMICS)

Following Simulink diagram was used:

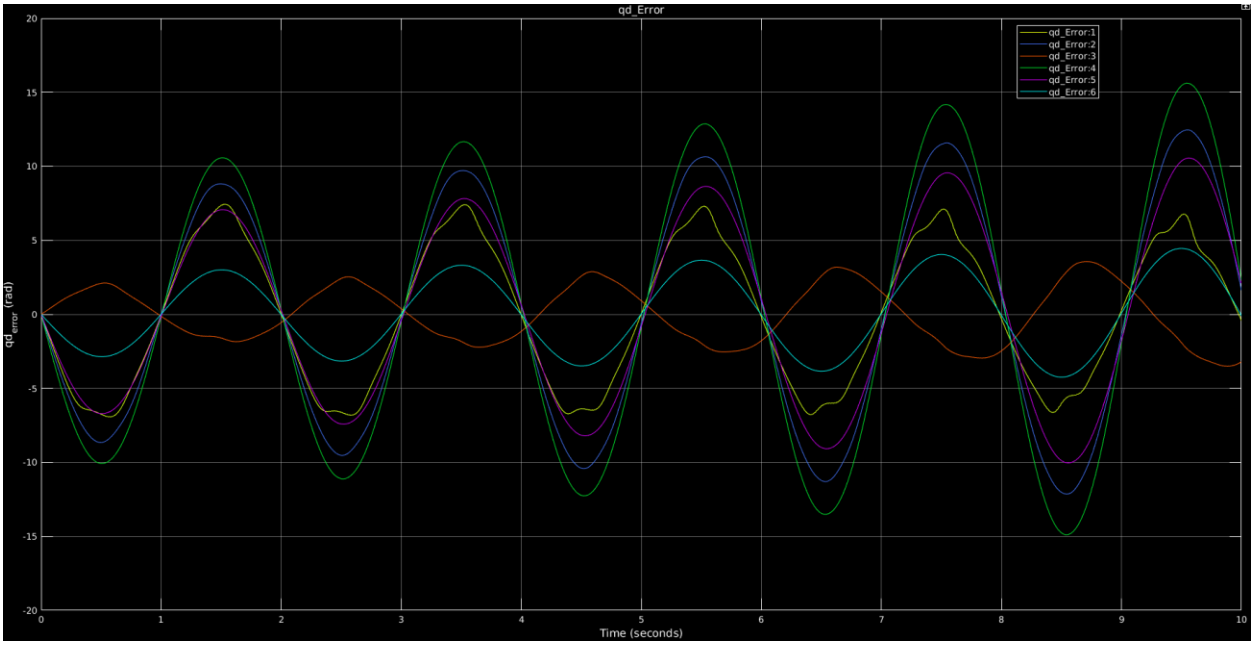


Kp = 10, Kd = 0

Joint Coordinate error:



Joint Speed error:

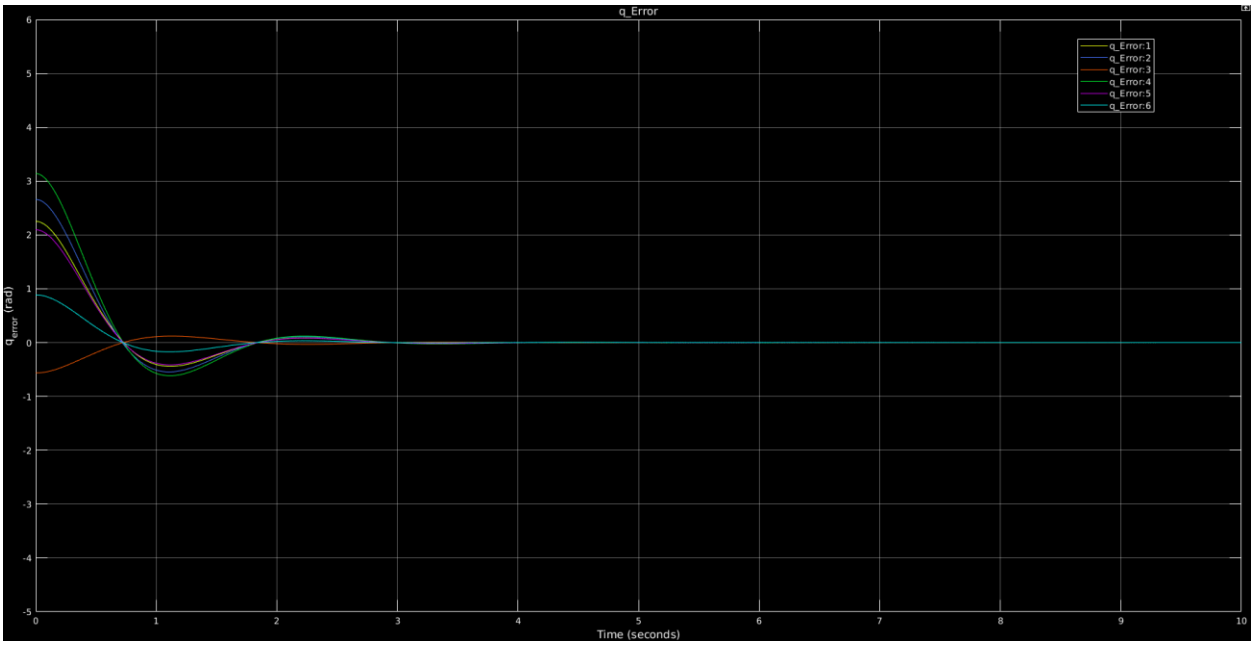


Here we can see that both Kp and Kd are necessary for convergence to equilibrium. This makes sense because the dynamics of the tracking error are defined by:

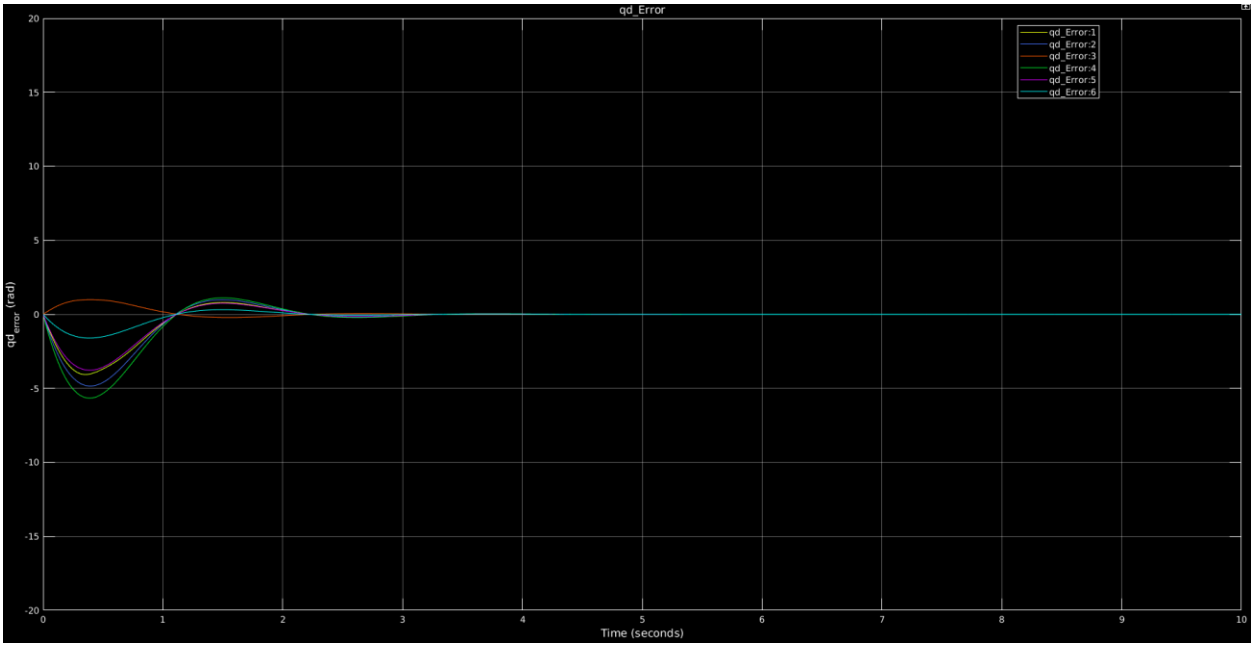
$$\ddot{\tilde{\mathbf{q}}} + \mathbf{K}_D\dot{\tilde{\mathbf{q}}} + \mathbf{K}_P\tilde{\mathbf{q}} = \mathbf{0}$$

Kp = 10, Kd = 3

Joint Coordinate error:

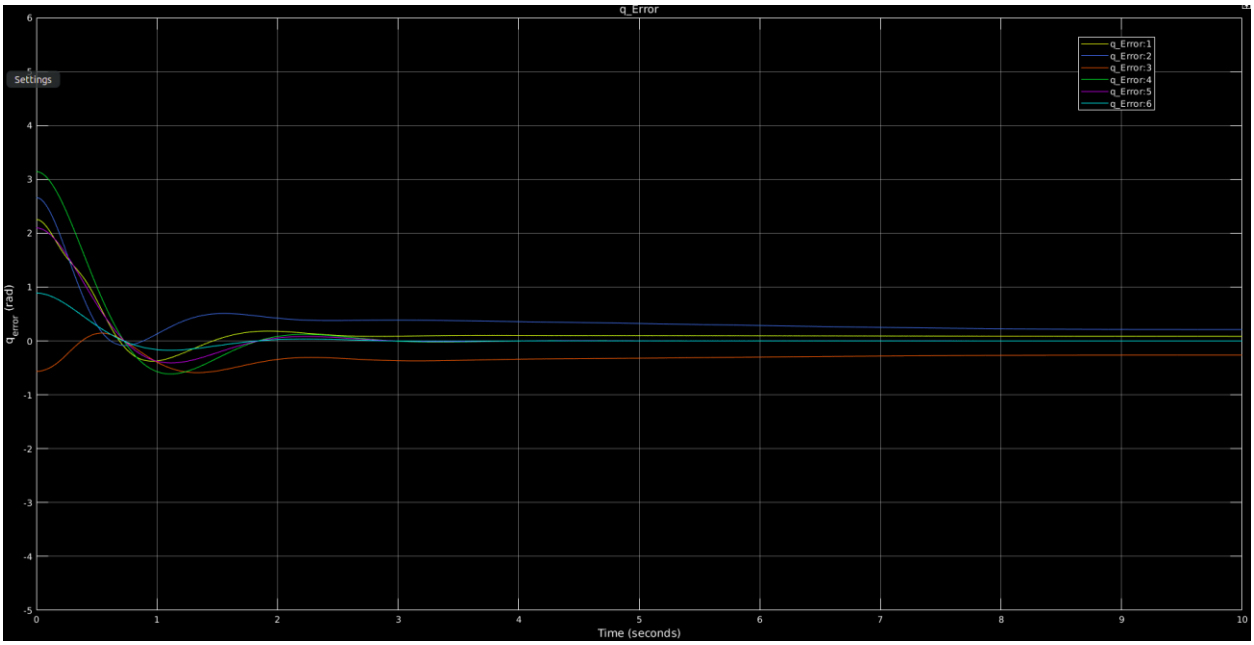


Joint Speed error:

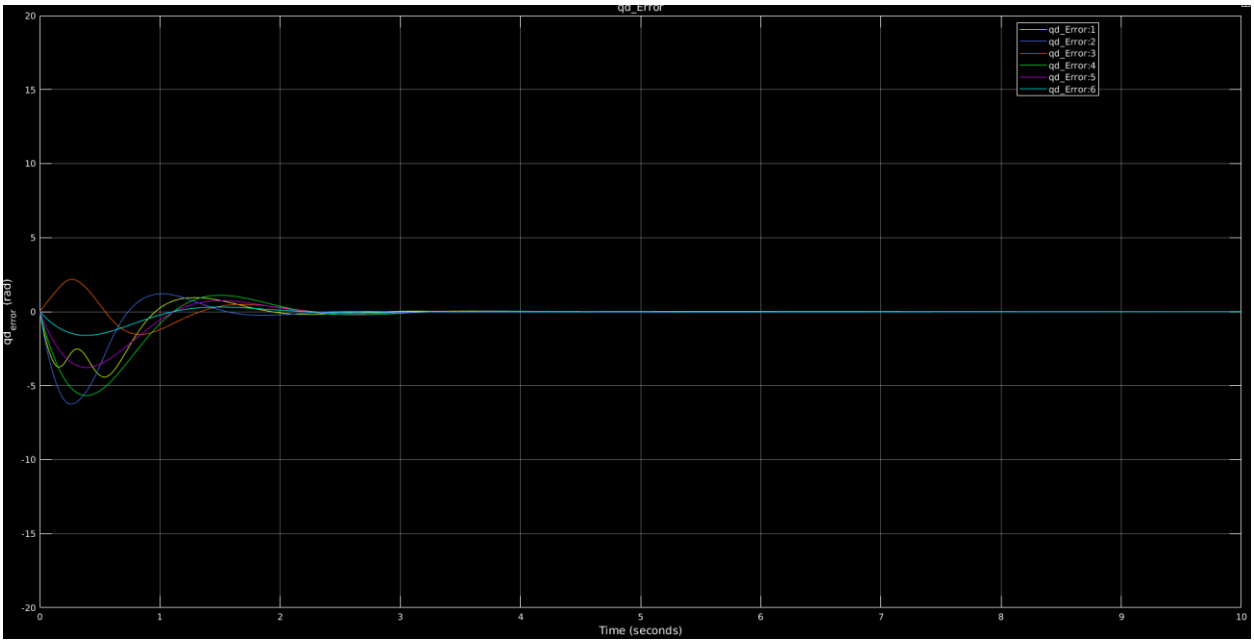


Kp = 10, Kd = 3 and parametric uncertainties present

Joint Coordinate error:



Joint Speed error:



We can see the joint position error no longer converges to zero but is always at a certain offset. This makes sense because the control law assumes perfect knowledge of the parameters of the manipulator. When we don't have perfect knowledge, we will not get zero error:

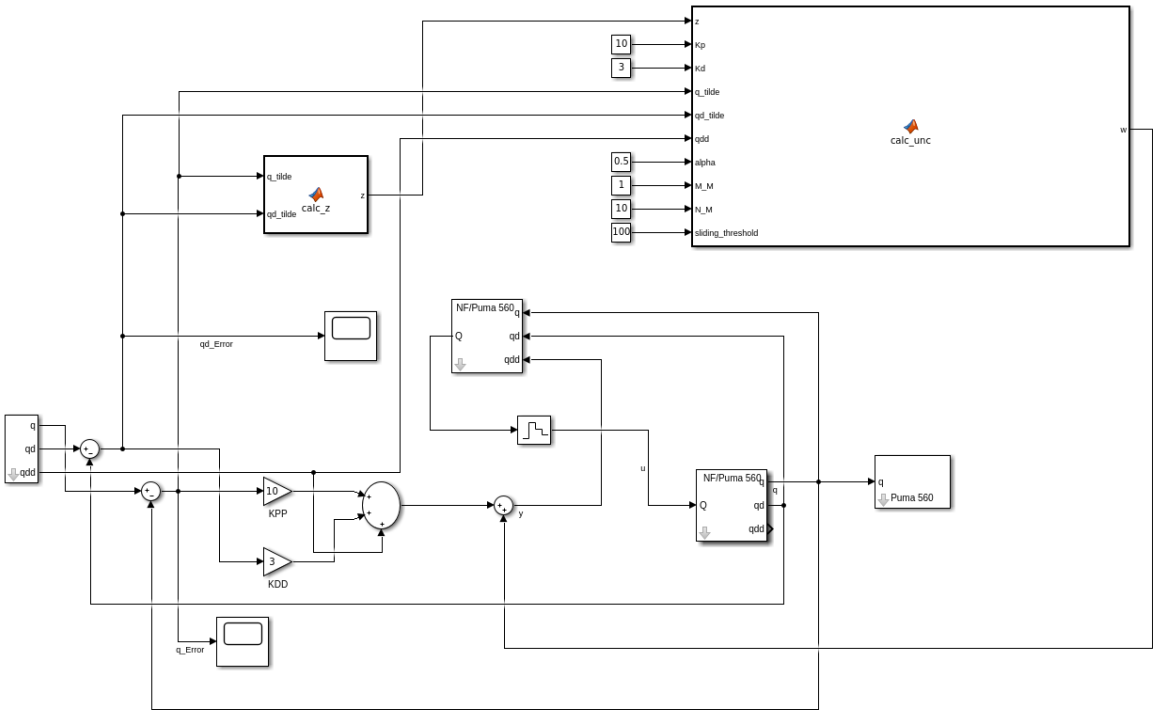
$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{n} = \mathbf{M}\mathbf{y} + \mathbf{n}$$

$$\mathbf{y} = -\mathbf{K}_P\mathbf{q} - \mathbf{K}_D\dot{\mathbf{q}} + \mathbf{r}$$

$$\ddot{\mathbf{q}} + \mathbf{K}_D\dot{\mathbf{q}} + \mathbf{K}_P\mathbf{q} = \mathbf{r}$$

EXERCISE 4 (ROBUST CONTROL)

To deal with uncertainties, we use Robust Control. Following Simulink diagram was used:

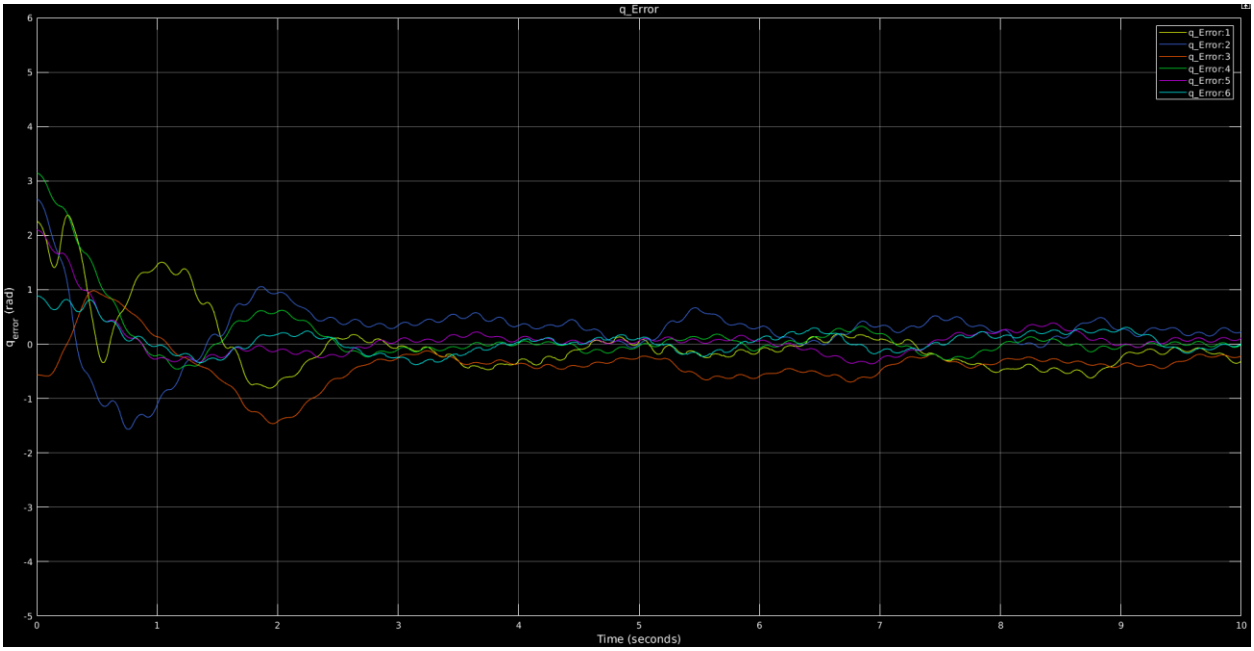



```
%MATLAB CODE FOR CALC_Z
function z = calc_z(q_tilde, qd_tilde)
D = [eye(6)*0; eye(6)];
matrixSize = 12;
while true
  A = rand (matrixSize, matrixSize);
  if rank (A) == matrixSize; break ; end % will be true nearly all the time
end
Q = A' * A; %positive definite matrix
epsilon = [q_tilde; qd_tilde];
z = transpose(D) * Q * epsilon;
end
```

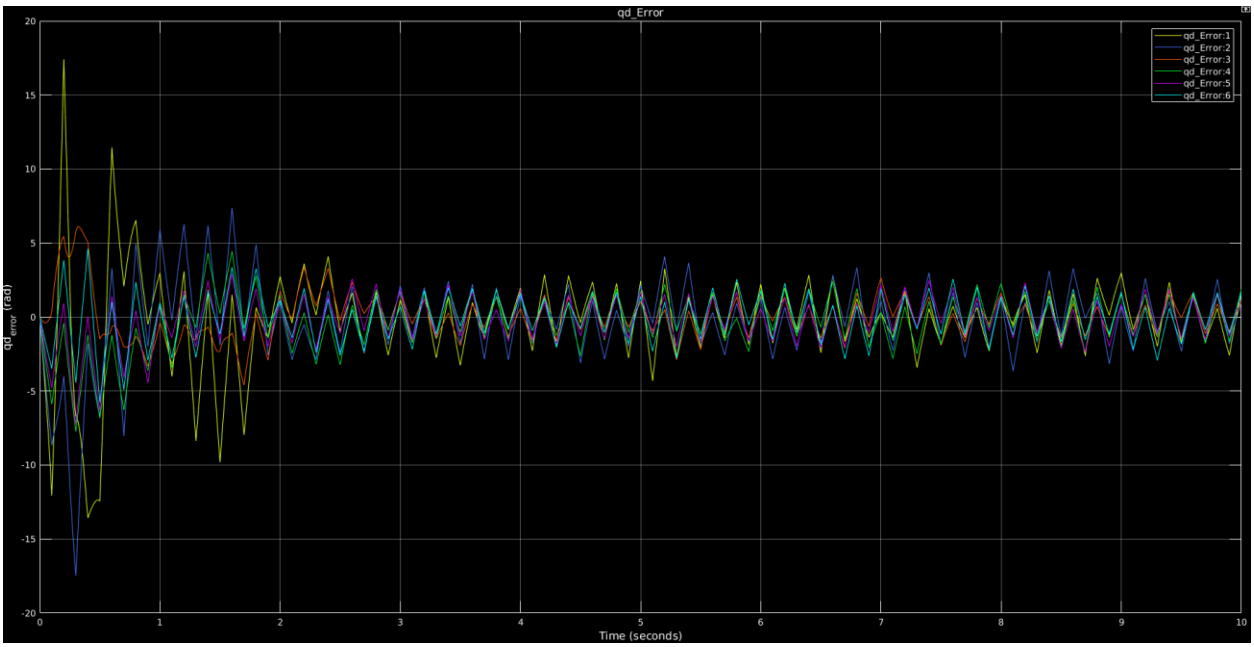
```
%MATLAB CODE FOR CALC_UNC
function w = calc_unc(z, Kp, Kd, q_tilde, qd_tilde, qdd, alpha, M_M, N_M, sliding_threshold)
   K = [eye(6)*Kp eye(6)*Kd];
   epsilon = [q_tilde; qd_tilde];
   Q_M = max(abs(qdd)) + 0.1;
   rho = (1/(1-alpha))*(alpha*Q_M + alpha*norm(K)*norm(epsilon) + M_M*N_M);
   z_unit_vector = z/norm(z);
   if norm(z) >= sliding_threshold
      w = rho * z_unit_vector;
   else
      w = rho * z * (1/sliding_threshold);
   end
end
```

Kp = 10, Kd = 3, alpha = 0.4, M_M = 5, N_M = 5, slide_threshold = 10

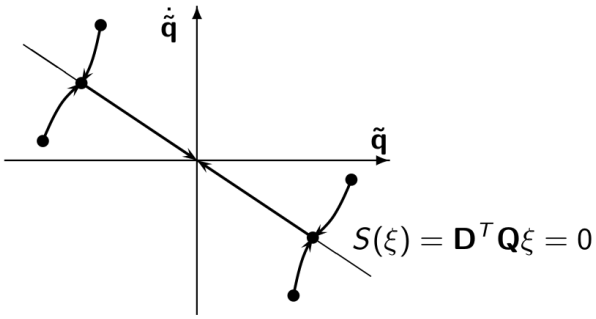
Joint Coordinate error:



Joint Speed error:

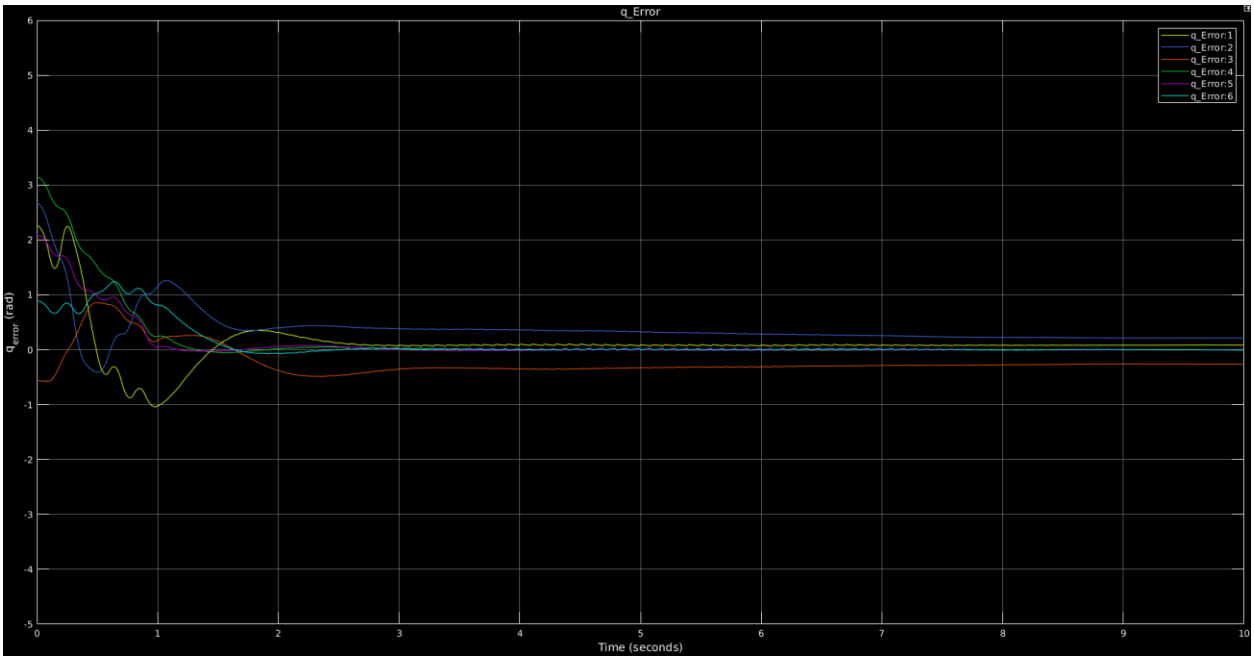


We immediately notice the chattering phenomenon. This is because the threshold of the boundary layer is not so far from the sliding surface which is defined as:

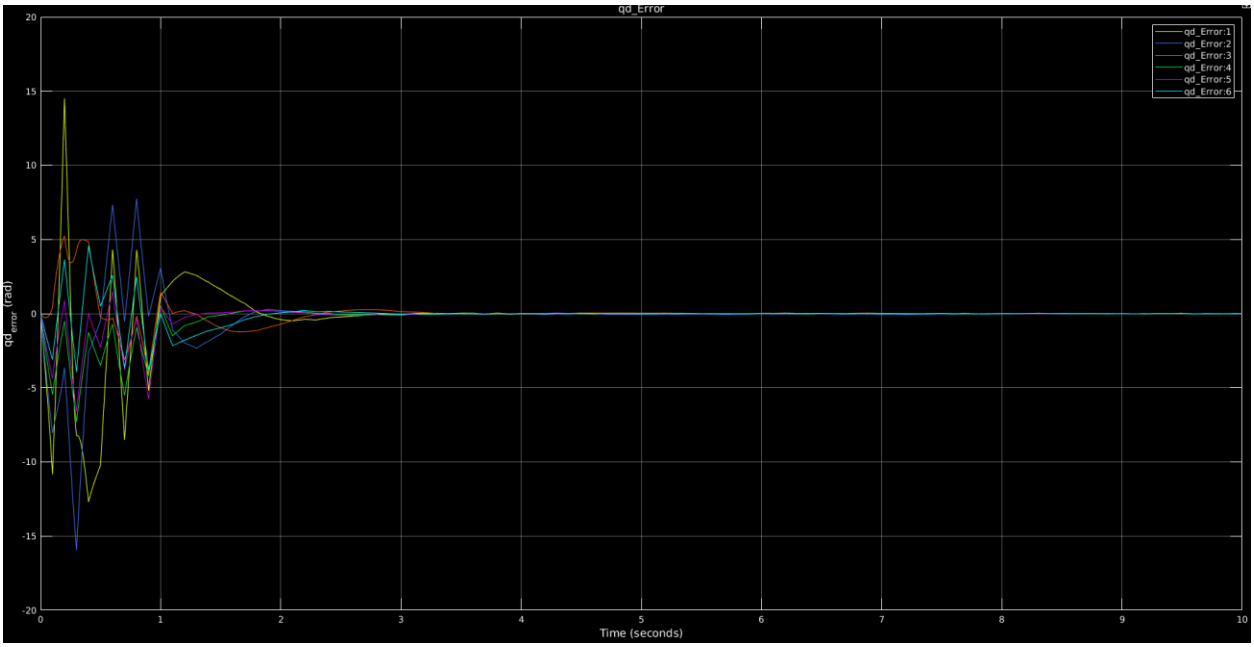


Kp = 10, Kd = 3, alpha = 0.4, M_M = 5, N_M = 5, slide_threshold = 100

Joint Coordinate error:



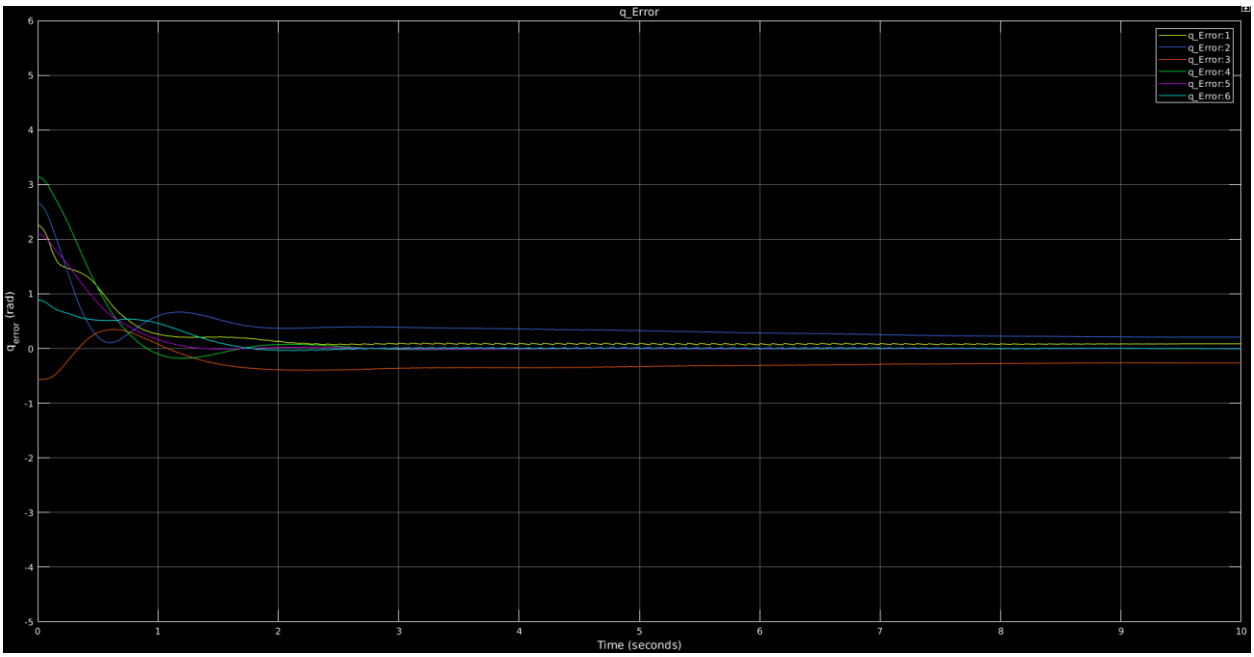
Joint Speed error:



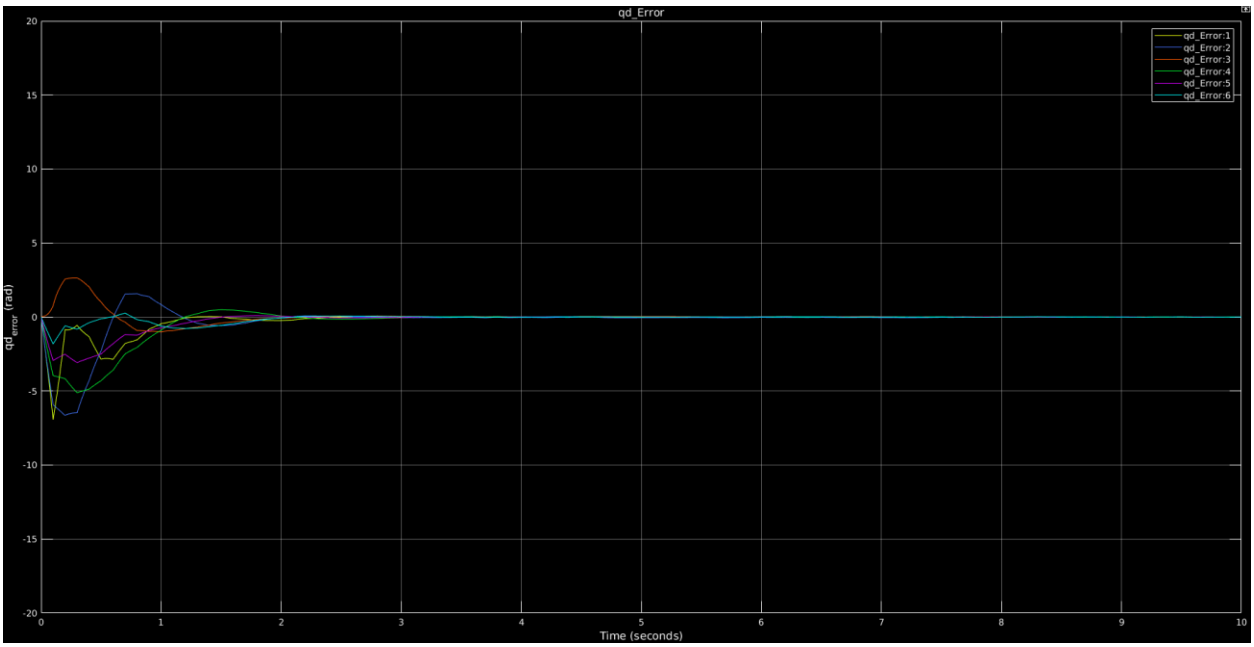
We no longer see any visible chattering throughout the graph (only in the beginning).

Kp = 10, Kd = 3, alpha = 0.1, M_M = 5, N_M = 5, slide_threshold = 100

Joint Coordinate error:



Joint Speed error:

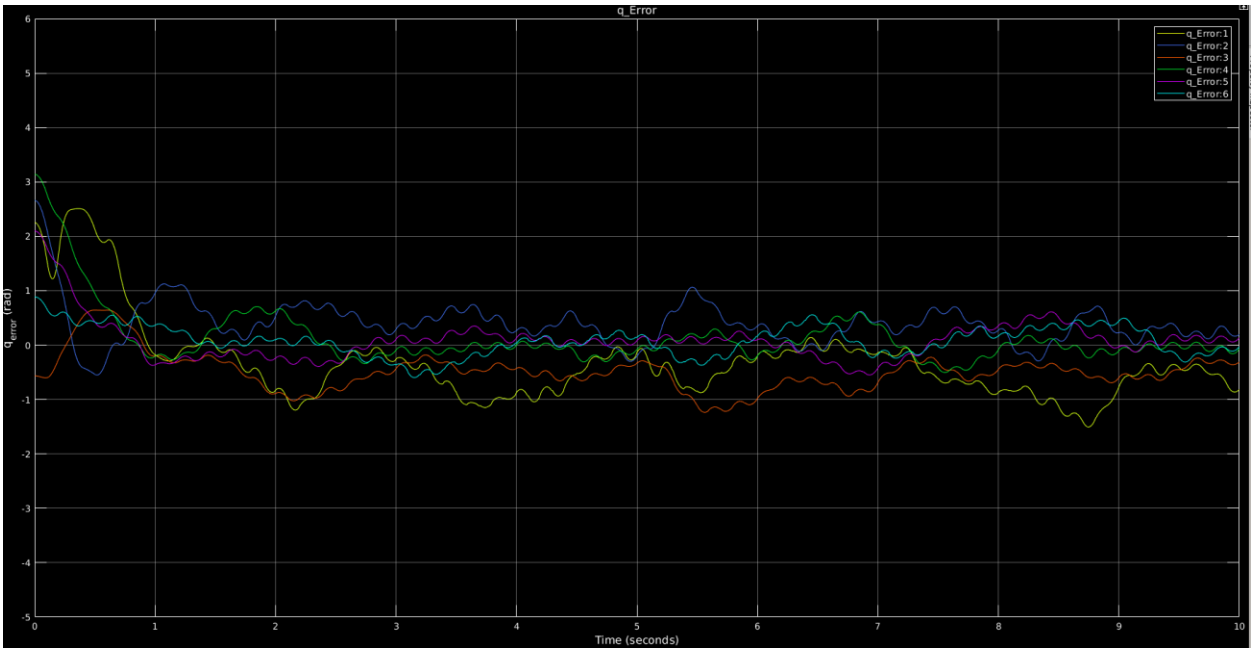


By increasing alpha, we are saying that our confidence in the estimated inertia matrix is higher meaning that it may be closer to the true value of the manipulator's inertia matrix. We also see much less chattering in the initial phase.

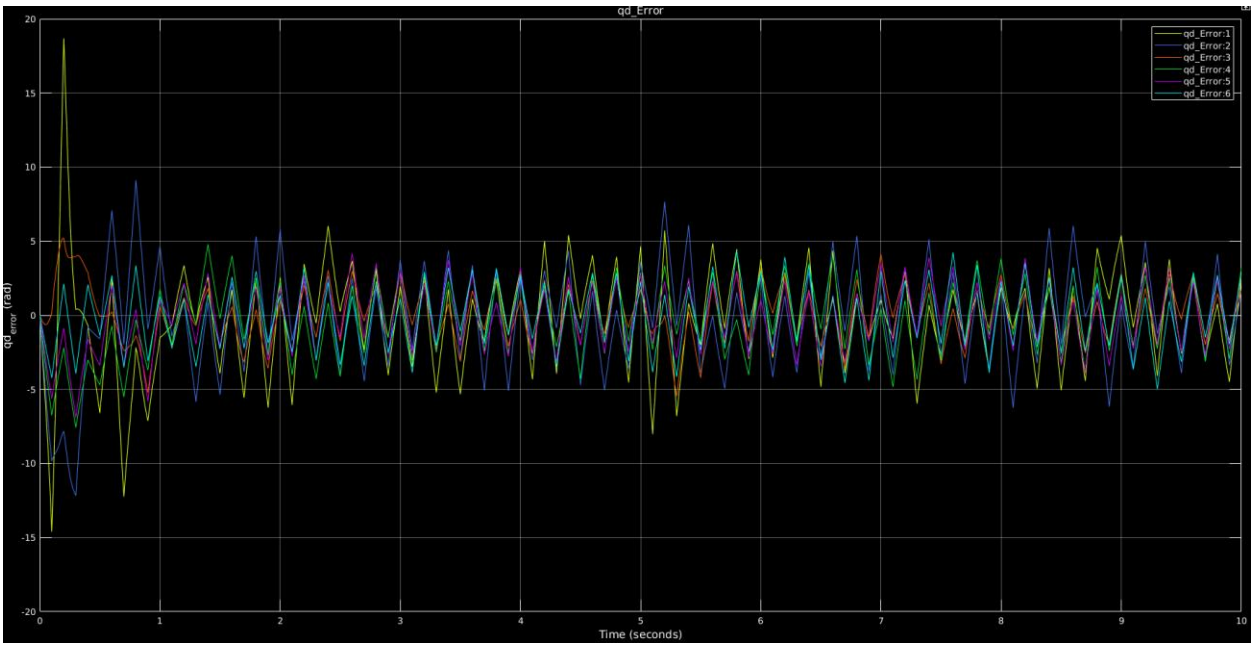
Kp = 10, Kd = 3, alpha = 0.1, M_M = 10, N_M = 10,

slide_threshold = 100

Joint Coordinate error:



Joint Speed error:



By increasing M_M and N_M (which provide upper bound on the parameter uncertainties) and keeping all other factors the same, we once again encounter chattering. M_M and N_M are defined as:

$$\|\mathbf{I} - \mathbf{M}^{-1}(\mathbf{q})\hat{\mathbf{M}}(\mathbf{q})\| \leq \alpha \leq 1 \qquad \forall \mathbf{q}$$

$$\|\tilde{\mathbf{n}}\| \leq N_M < \infty \qquad \forall \mathbf{q}, \dot{\mathbf{q}}$$

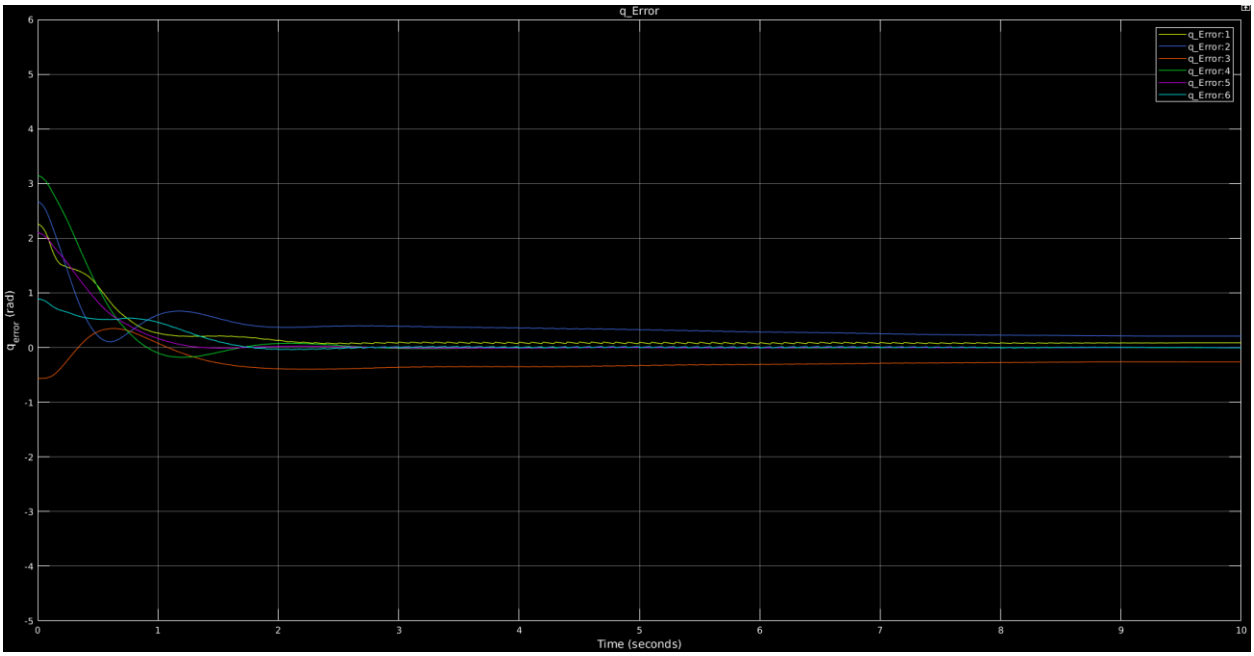
$$0 \leq M_m \leq \|\mathbf{M}^{-1}(\mathbf{q})\| \leq M_M \leq \infty, \qquad \forall \mathbf{q}$$

So, by increasing N_M we are saying that the difference between the true and estimated parameters is quite high and by increasing M_M we are estimating a higher value for the inverse of the true inertia matrix.

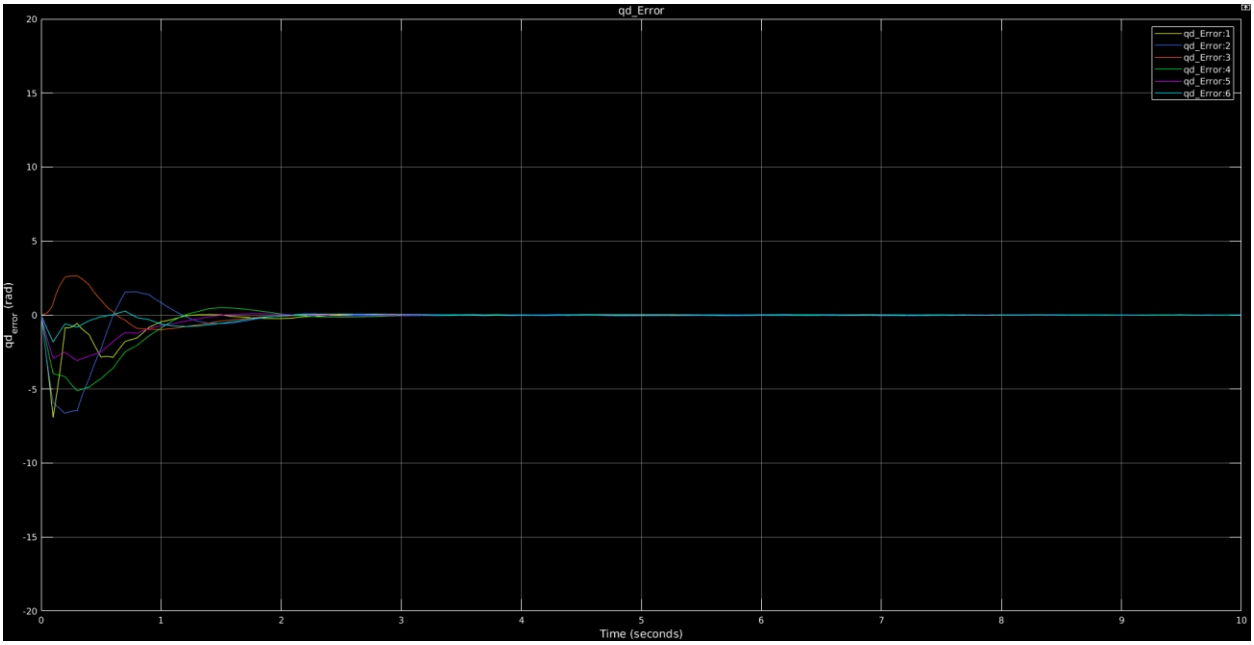
Kp = 10, Kd = 3, alpha = 0.1, M_M = 10, N_M = 10,

slide_threshold = 1000

Joint Coordinate error:



Joint Speed error:

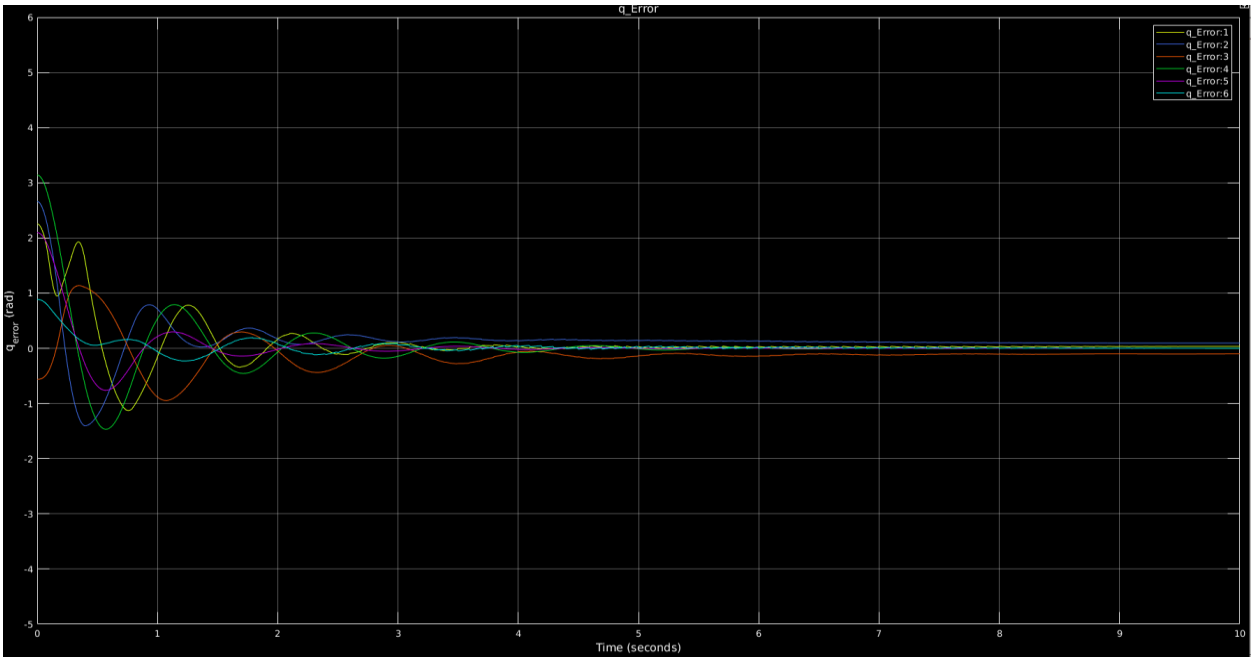


Once again, we were able to deal with chattering by further increasing the threshold of the boundary layer around the sliding region.

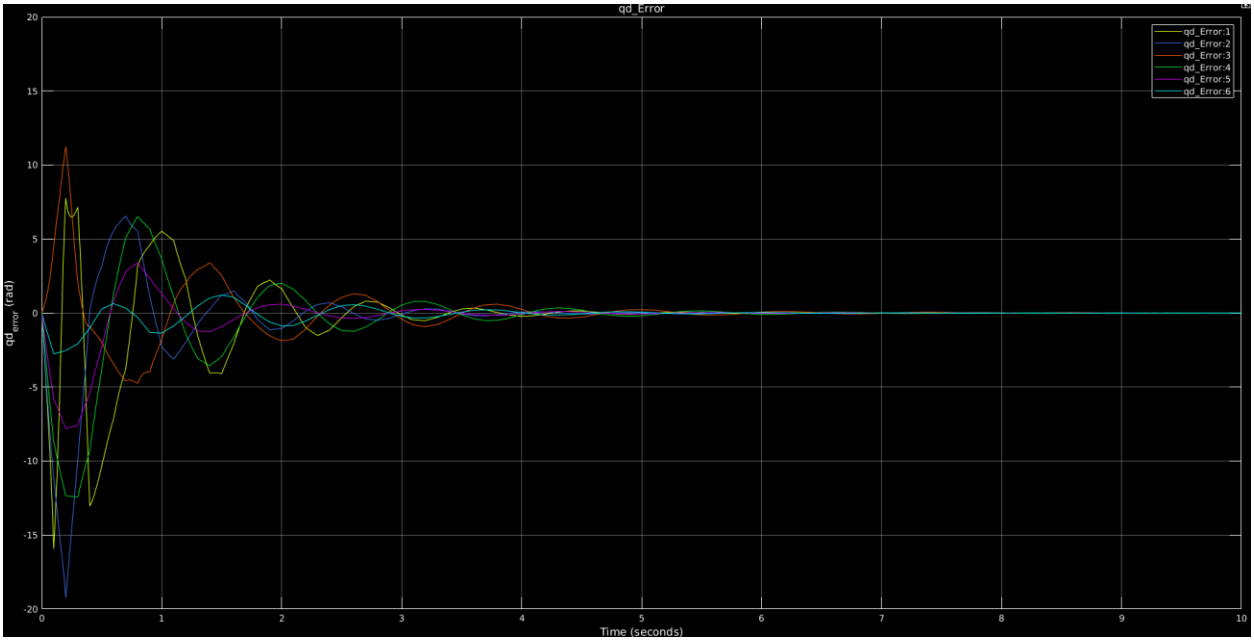
Kp = 30, Kd = 3, alpha = 0.05, M_M = 1, N_M = 1,

slide_threshold = 100:

Joint Coordinate error:



Joint Speed error:



For the above values, we are getting error values very close to 0.

EXERCISE 5 (PD + GRAVITY COMPENSATION IN WORKSPACE COORDINATES)

Following control law was used (with derivation):

$$V(x) = \frac{1}{2}\,\dot{q}^{\,T} M(q)\,\dot{q} + \frac{1}{2}\,\tilde{x}^{\,T} K_p\,\tilde{x}$$

$$\dot{V}(x) = \frac{1}{2}\,\dot{q}^{\,T}\,\dot{M}\,\dot{q} + \dot{q}^{\,T} M\,\ddot{q} + \dot{\tilde{x}}^{\,T} K_p\,\tilde{x}$$

$$\dot{V}(x) = \frac{1}{2}\,\dot{q}^{\,T}\,\dot{M}\,\dot{q} + \dot{q}^{\,T}(u - C\,\dot{q} - D\,\dot{q} - g) - \dot{x}^{\,T} K_p\,\tilde{x}$$

$$\dot{V}(x) = \frac{1}{2}\,\dot{q}^{\,T}\,\dot{M}\,\dot{q} - \dot{q}^{\,T} C\,\dot{q} + \dot{q}^{\,T}(u - D\,\dot{q} - g) - \dot{x}^{\,T} K_p\,\tilde{x}$$

$$\dot{V}(x) = \dot{q}^{\,T}(u - D\,\dot{q} - g) - \dot{x}^{\,T} K_p\,\tilde{x}$$

$$\left(\frac{1}{2}\,\dot{q}^{\,T}\,\dot{M}\,\dot{q} - \dot{q}^{\,T} C\,\dot{q} = 0 \quad \text{due to Christoffel Symbols}\right)$$

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

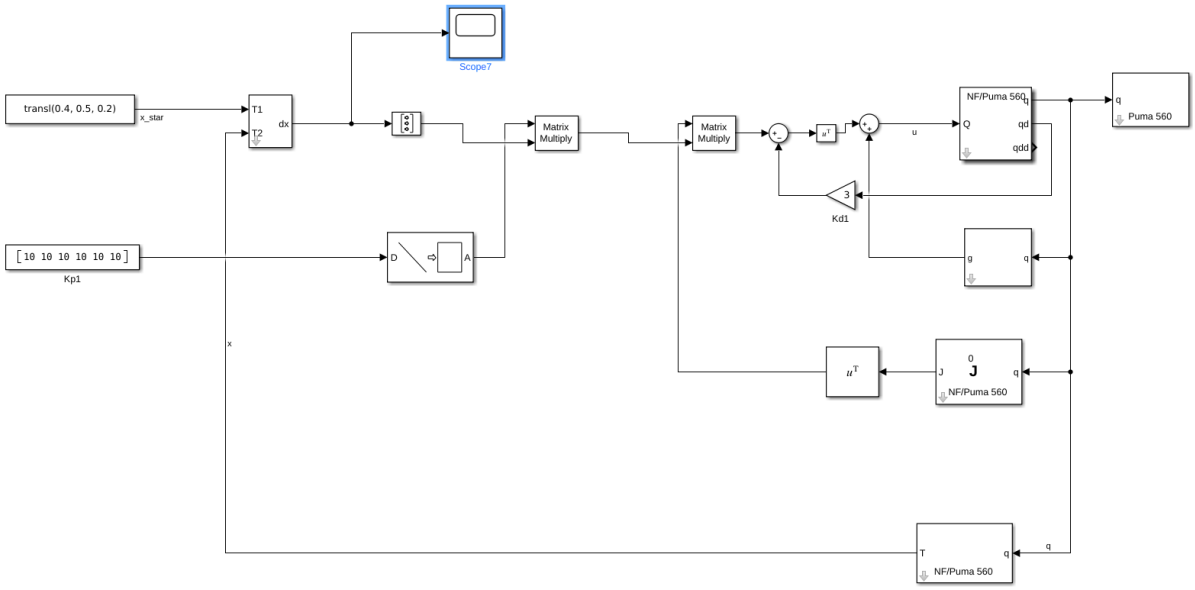
$$u = J^{T}(K_p\,\tilde{x}) - K_d\,\dot{q} + g$$

$$\dot{V}(x) = -\dot{q}\,D\,\dot{q} - \dot{q}\,K_d\,\dot{q} \le 0$$

$$\forall\,\dot{q} \text{ not equal to } 0$$

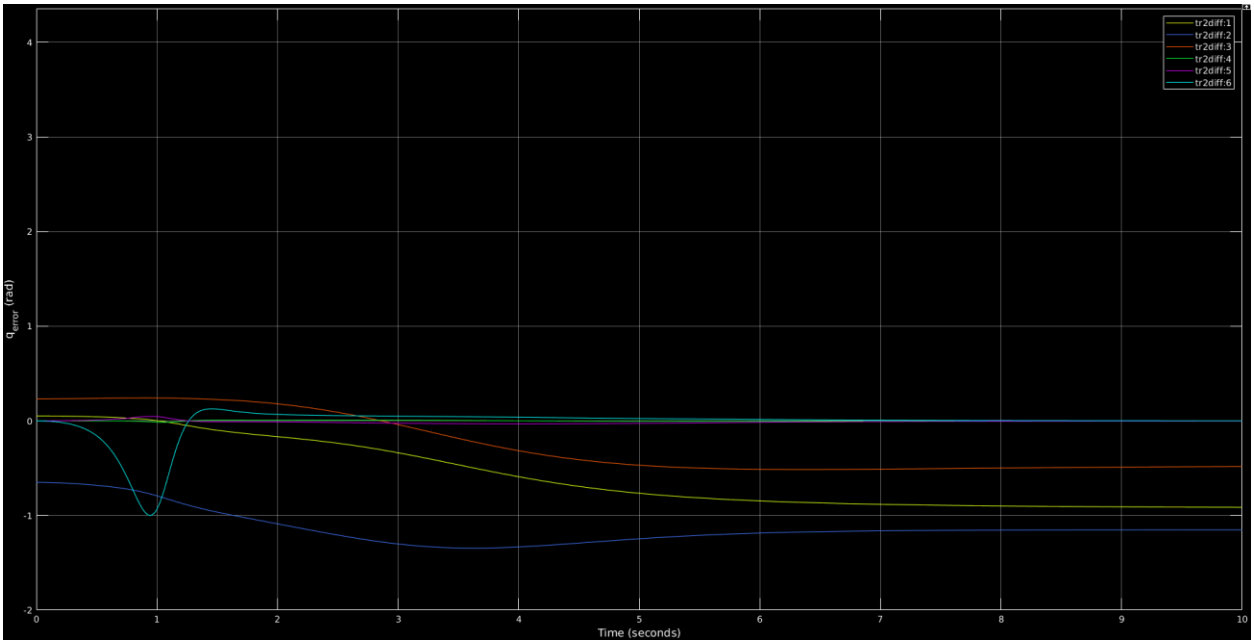
The choice of Lyapunov function is valid since it is 0 when joint space velocity and work space coordinate difference are 0 and it is positive definite when both are not equal to 0.

Following Simulink diagram was used:


Scope7
transl(0.4, 0.5, 0.2)
x_star
T1
T2
dx
Matrix Multiply
Matrix Multiply
NF/Puma 560
Q
qd
qdd
u
Puma 560
q
3
Kd1
[10 10 10 10 10 10]
Kp1
D
A
x
J
NF/Puma 560
T
NF/Puma 560

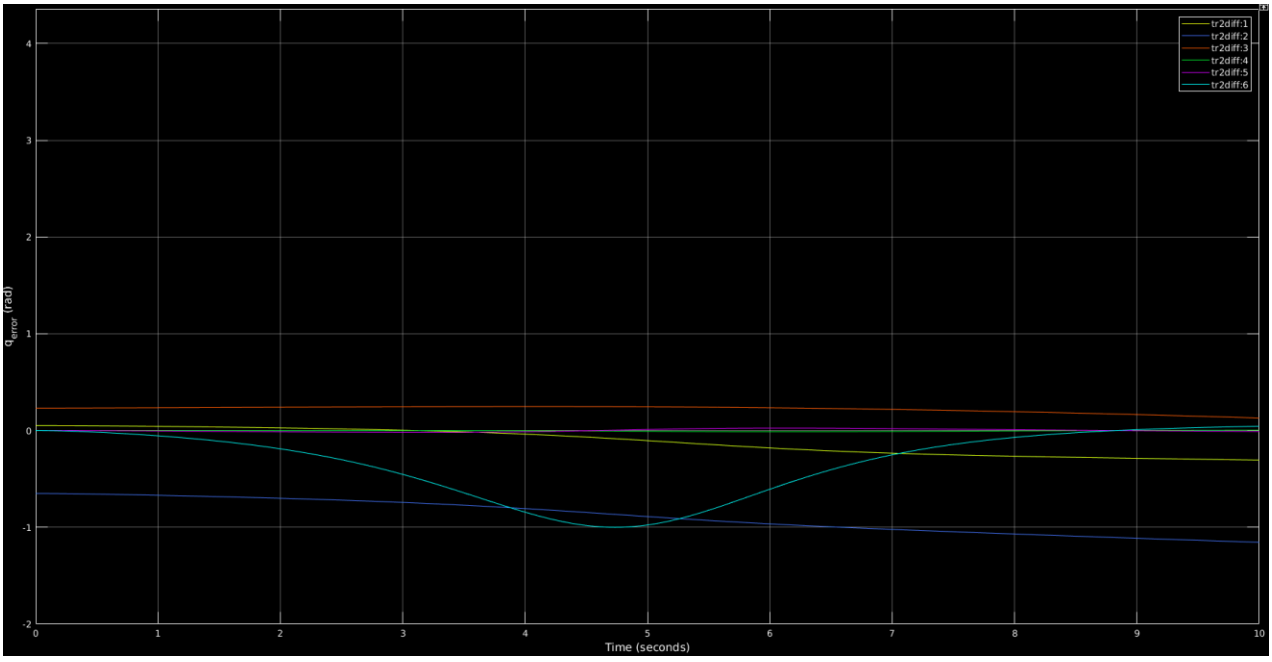
Kp = 10, Kd = 3

Joint Coordinate error:



Kp = 3, Kd = 10

Joint Coordinate error:



Increasing or decreasing either Kp or Kd still didn’t give perfect stability (zero error in joint space coordinates). This is because now we are using the Jacobian and singular configurations of J(q) coincide with singular configurations of the transpose of J(q). So, that means any singularities might create problems for us.