Final Project

Aim: This project aims to deploy nonlinear control algorithms for the trajectory tracking for the 4DOF Kinova arm. Our objective is to design and numerically implement 1) a Lyapunov-based controller,

2) an Adaptive Lyapunov-based controller

3) an Impedance controller.

Specs used of the Kinova Jaco Arm: These D.H parameters were used to compute the rotation and transformation matrices for the

Robot lenght values (meters)		
D1	0.2755	Base to elbow
D2	0.4100	Arm lengh
D3	0.2073	Front arm lengh
D4	0.0743	First wrist lengh
D5	0.0743	Second wrist lengh
D6	0.1687	Wrist to center of the hand
e2	0.0098	Joint 3-4 lateral offset

Alternate parameters	
aa	((11.0*PI)/72.0)
ca	(cos(aa))
sa	(sin(aa))
c2a	(cos(2*aa))
s2a	(sin(2*aa))
d4b	(D3 + sa/s2a *D4)
d5b	(sa/s2a*D4 + sa/s2a *D5)
d6b	(sa/s2a*D5 + D6)

1.1.1 Classic DH Parameters

DH Parameters				
i	**alpha(i-1)**	**a(i-1)**	**di**	**teta1**
1	pi/2	0	D1	q1
2	pi	D2	0	q2
3	pi/2	0	-e2	q3
4	2*aa	0	-d4b	q4
5	2*aa	0	-d5b	q5
6	pi	0	-d6b	q6

Mass used to calculate the Kinetic Energy.

Figure 5 : Inertial parameters

Inertial parameters	
m0	0.63 kg
m1	0.64 kg
m2	0.64 kg
m3	0.64 kg
m4	0.39 kg
m5	0.39 kg
m6	0.39 kg
mH	0.93 kg

Source for the images:
https://github.com/JenniferBuehler/jaco-arm-pkgs/blob/master/jaco_arm/jaco_description/doc/DH%20Parameters%20-%20Kinova%20-%201.1.6.pdf

Bezier Curves: To calculate the desired position, velocity and acceleration, have used the files provided by Dr. Kaveh which has coefficients precoded depending on the dimensions of alpha. In my case the dimensions of alpha is 4x4 since I am simulating a 4_DOF version of the Kinova Jaco Arm. The alpha is: a = [pi/6 pi/6 pi/2 pi/2; pi/4 pi/4 pi/3 pi/3; pi/6 pi/6 pi/5 pi/5; pi/8 pi/8 pi/5 pi/5];

1) Lyapunov-Based Control for Tracking

The mathematics is directly taken from the document given for the Problem Statement provided by Dr. Kaveh.

Equations of Motion:

$$D(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + G(q) = u,$$

Control Law:

$$u = D(q)\,\ddot{\zeta} + C(q,\dot{q})\,\dot{\zeta} + G(q) - K_D\,\sigma,$$

Where

$$\dot{\zeta} := \dot{q}_{\mathrm{d}}(t) - \Lambda\,\tilde{q}$$
$$\sigma := \dot{\tilde{q}} + \Lambda\,\tilde{q},$$

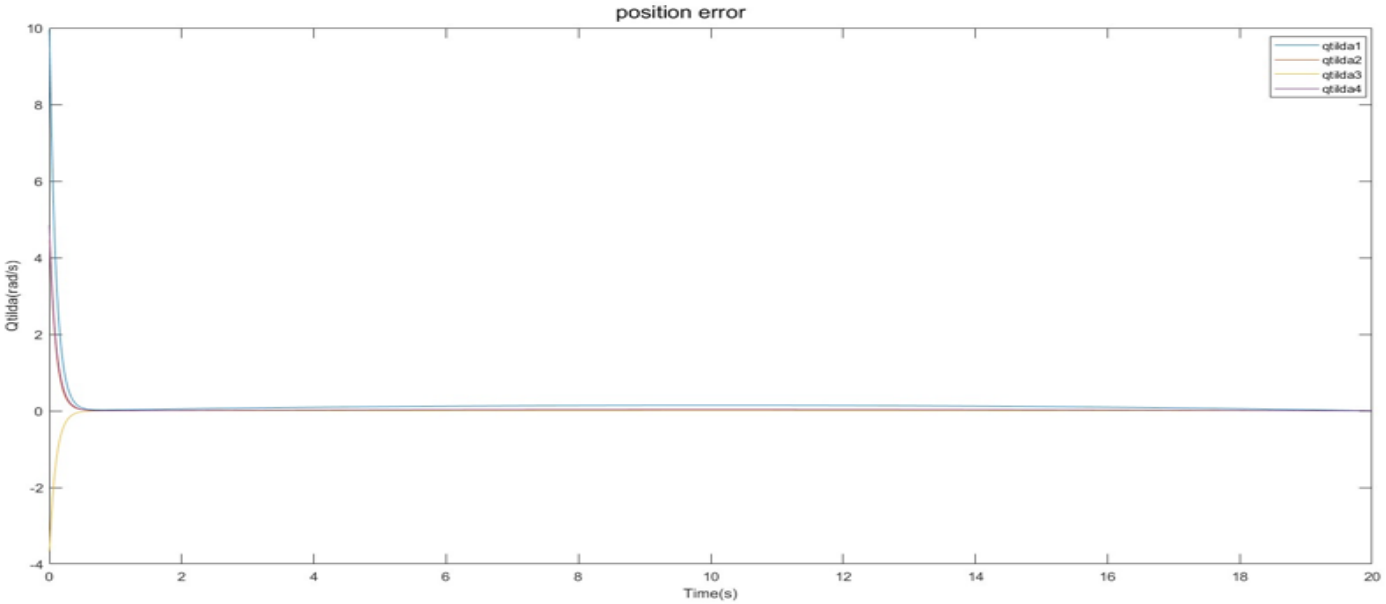
The augmented state space equation to feed the ode is

$$\dot{x} = \begin{bmatrix} \dot{q} \\ D^{-1}(q)\,(u(t,q,\dot{q}) - C(q,\dot{q})\,\dot{q} - G(q)) \end{bmatrix}.$$

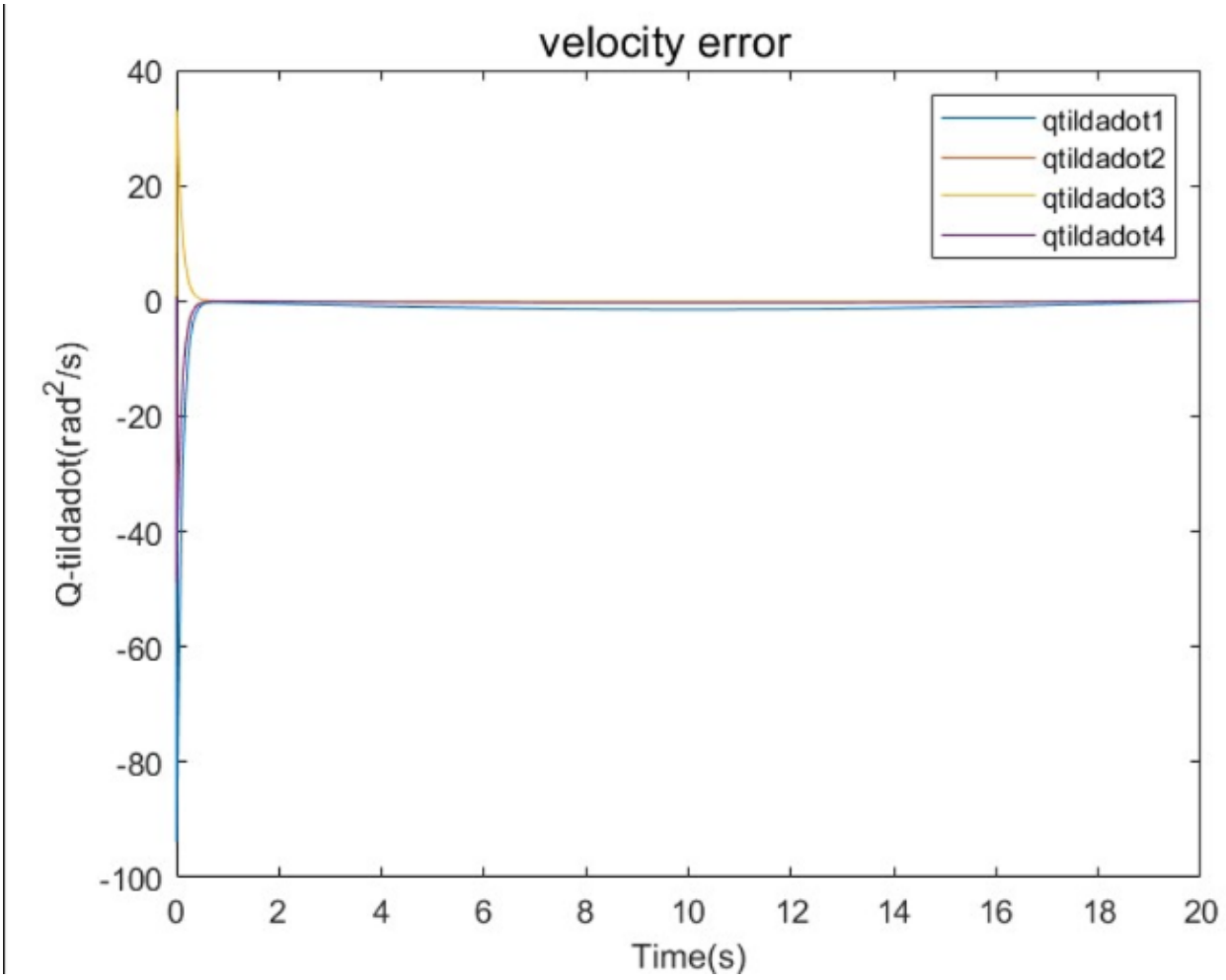
The initial conditions I chose have larger offset from the start of desired bezier function trajectories.
startpos = pi/3;
x0= [startpos*10;5*startpos;-3*startpos;5*startpos;0.2*5;3*0.2;0.1*3;8*0.1];

Plots for q(t),qtilda(t)) and u(t) versus time over [0,20] (s).

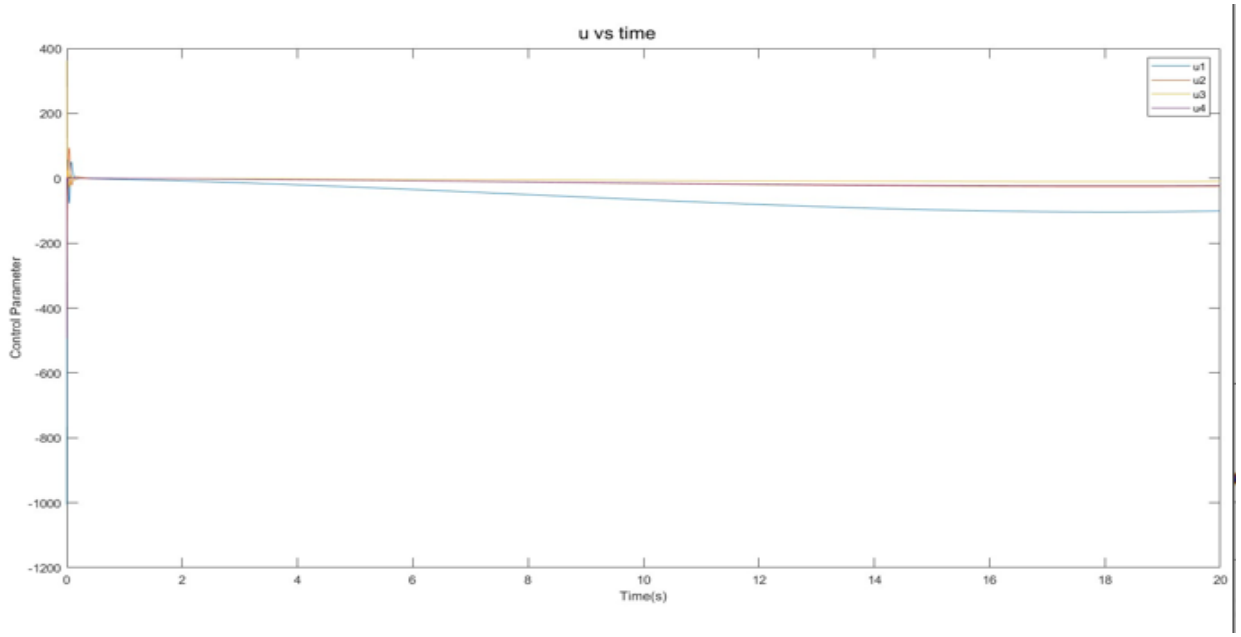
1) qtilda(t) vs Time(s)



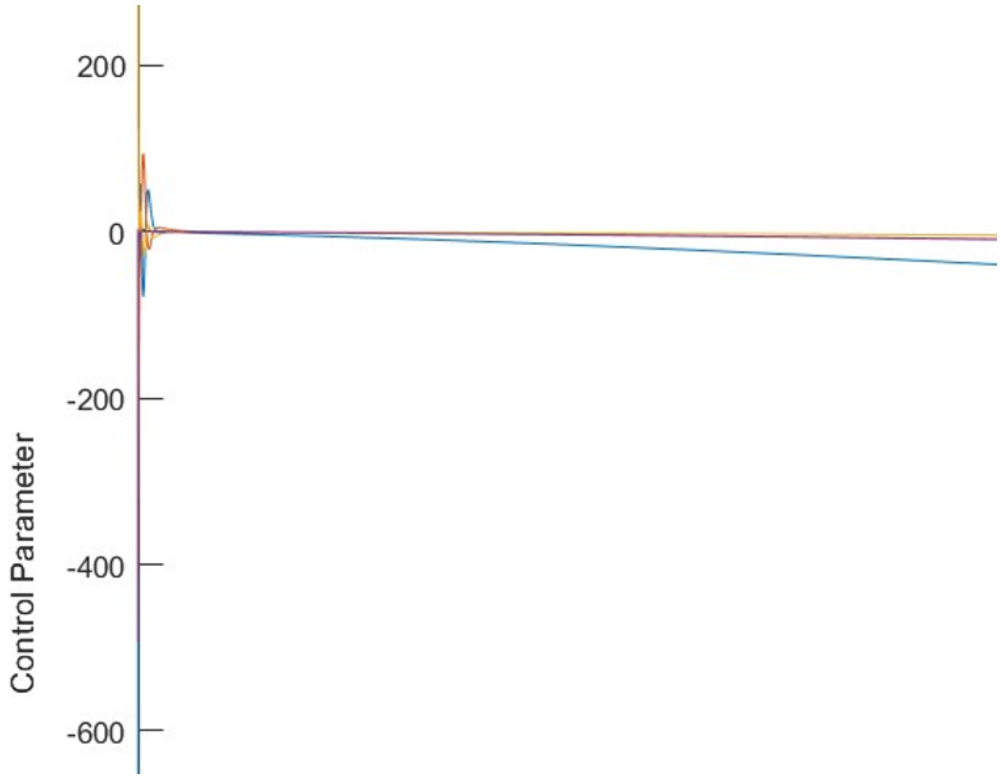
2) qtildadot(t) vs Time



3) u(t) vs TIme

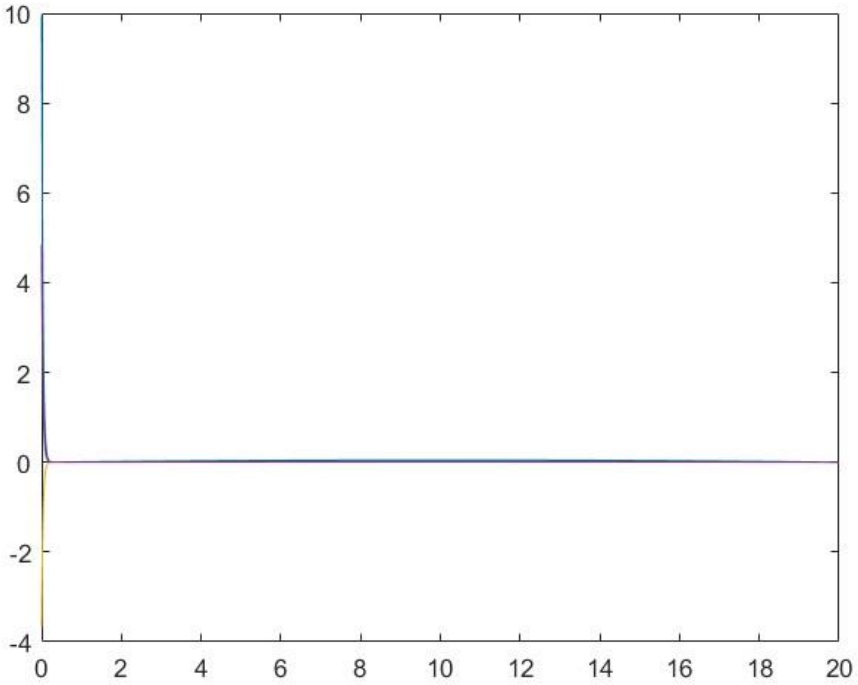


This is zoomed in version of the same plot above of the starting conditions.

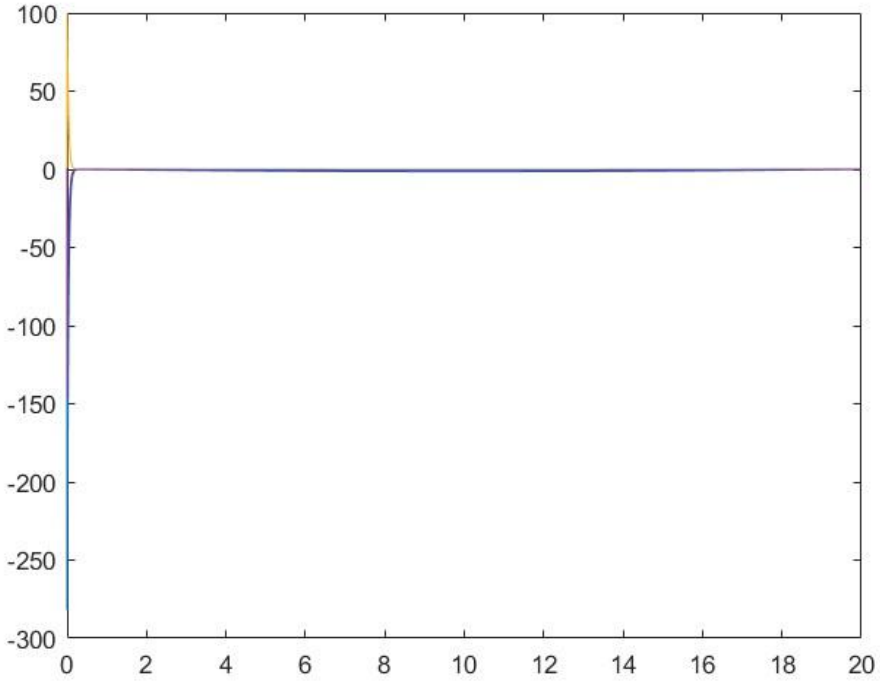


These plots are depicted for very high value of lambda and KD.(30)

1) Position error



2) Velocity Error



As expected, higher KD and lambda converge errors faster but take a lot of time to compute. **Conclusion:** I tested the algorithm for various values of KD and lambda for lower values of lambda and KD the convergence was slower but the control inputs were low as well. I did not know a practical enough value to set the KD and lambda such that the control input values are also in the feasible range. But as said the algorithm works for lower values as well but with a tradeoff with settling time.

2) Adaptive Lyapunov-Based Control for Unknown Payloads

Equations of Motion:

$$D(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + G(q) = u + J^{\top}(q)\,f,$$

Where the f represents unknown payload,

Control Law:

$$u = D(q)\,\ddot{\zeta} + C(q,\dot{q})\,\dot{\zeta} + G(q) - K_D\,\sigma - J^{\top}(q)\,\hat{f},$$

The adaptation law: From lecture 23 with some changes and calculation the adaptation law used with KI is

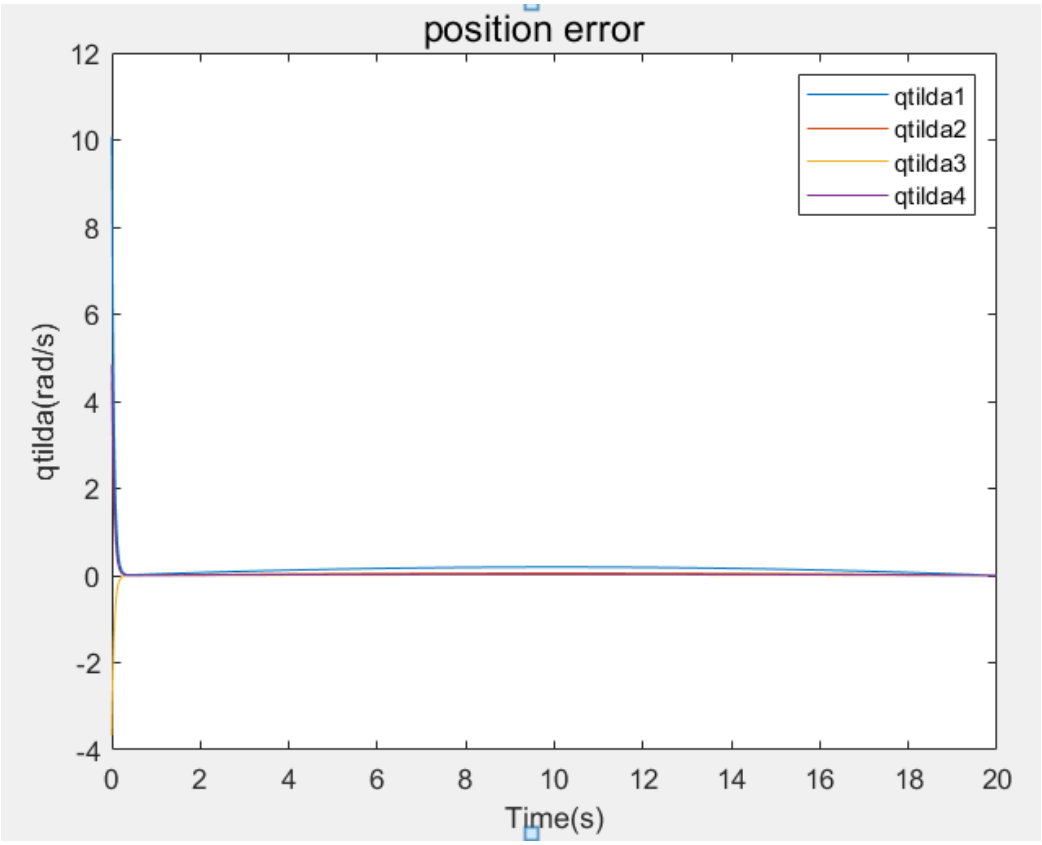
f_adapted = KI*J(q)*sigma;

The augmented state space equation to feed the ode is

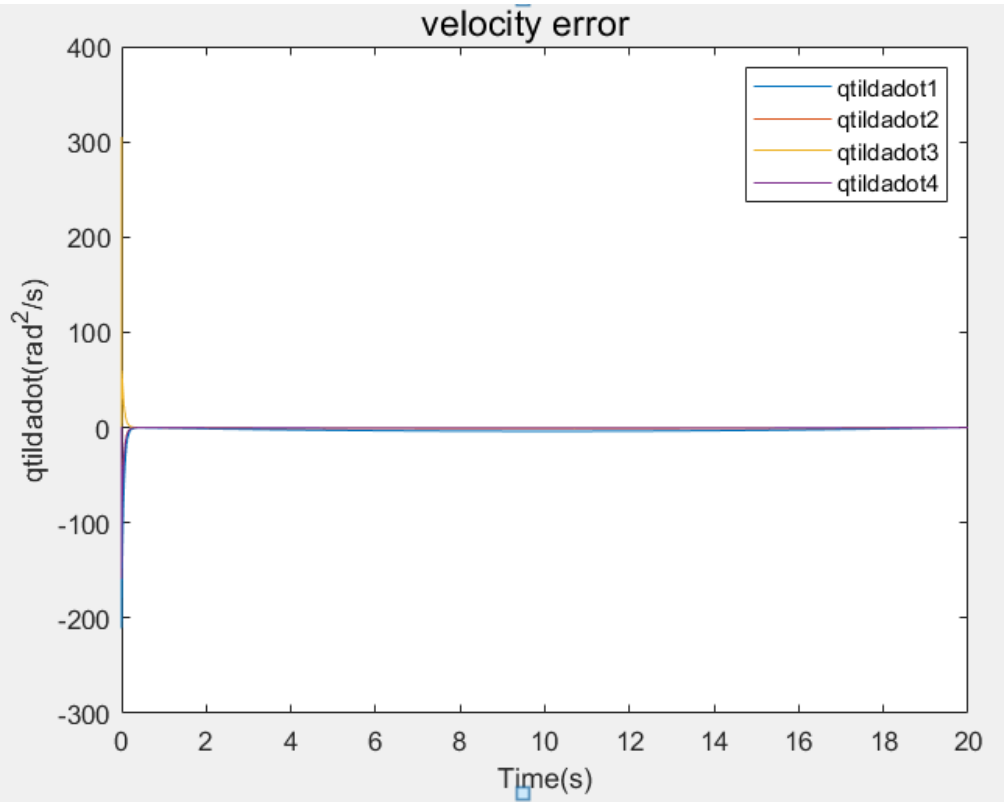
$$\dot{x}_a = \begin{bmatrix} \dot{q} \\ D^{-1}(q)\left(u(t,q,\dot{q},\hat{f}) + J^{\top}(q)\, f - C(q,\dot{q})\,\dot{q} - G(q)\right) \\ \text{Your adaptation law} \end{bmatrix}.$$

Plots for q(t),qtilda(t)), u(t) matilda(t) versus time over [0,20] (s).

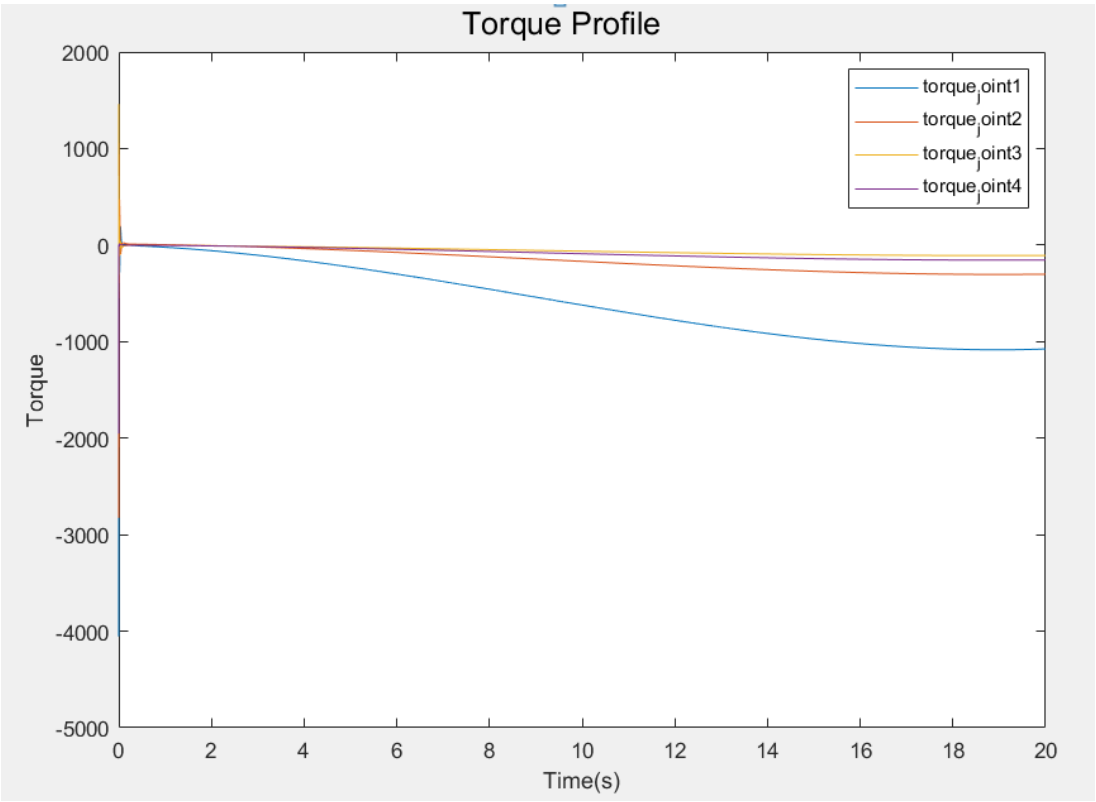
1) qtilda(t) vs Time



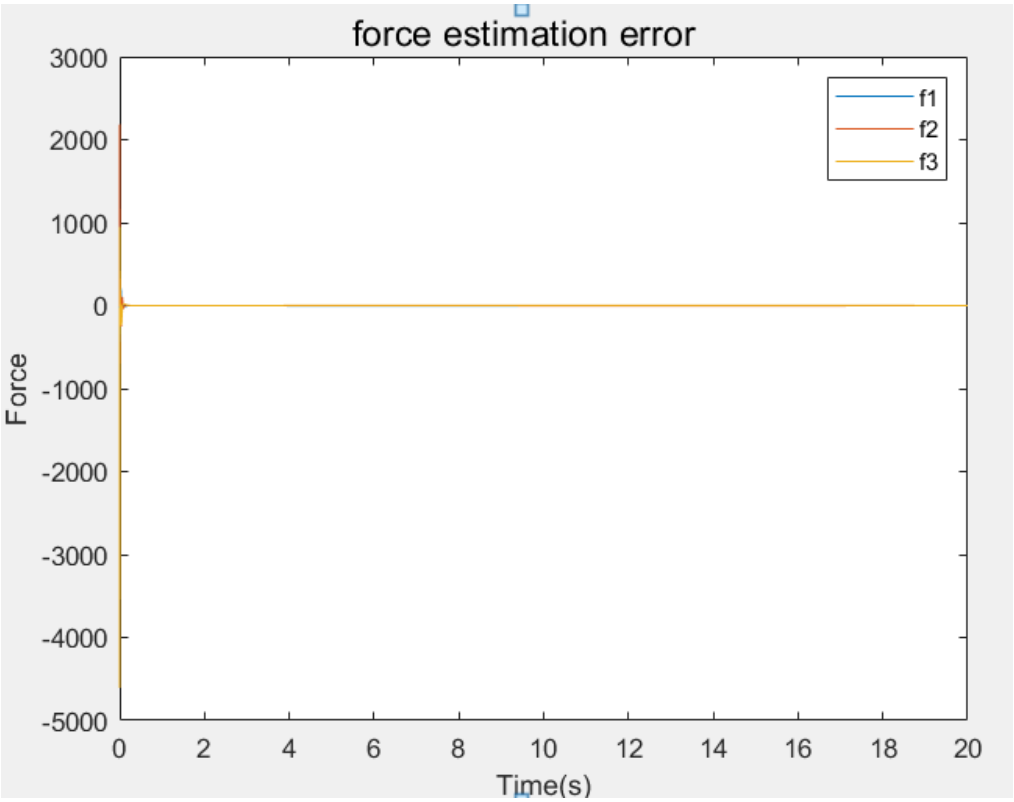
2) qtildadot(t) vs Time



3) u(t) vs TIme



4) ftilda(t) vs Time



The limit ftilda(t) → ∞ is 0

Conclusion: increasing the Kp and Kd values makes qtilda and qtildadot converge faster but the tradeoff is with the control inputs as they get high which is expected as higher gain means that the control input has to be higher. I tried the algorithm with lower values of Kp Kd and KI The difference is visible where the convergence is after some time of the start of the simulation but in the end all converge to 0. For very low values of Kp and Kd there is no convergence for the given time period of T=20 seconds. Some of the plots are included in the folder submitted along with this report starting with the name low_adaptive. I did not know a feasible/practical range for the control inputs to be hence the submission made depicts the force_tilda converging to 0 but the control input is higher as well. While tuning I found out that for lower Kp kd values the force does converge to 0 with some delay(<1s) with the same KI and lower control inputs. This may also be a valid case.
Please see the plots folder to review these graphs as well if necessary.

Impedance Control

Equations of Motion:

$$D(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + G(q) = u + J^{\top}(q)\,f.$$

Desired Dynamics: $$\ddot{x}_{\mathrm{ee}} = \ddot{x}_{\mathrm{d}} - K_P\left(x_{\mathrm{ee}} - x_{\mathrm{d}}(t)\right) - K_D\left(\dot{x}_{\mathrm{ee}} - \dot{x}_{\mathrm{d}}\right) + f,$$

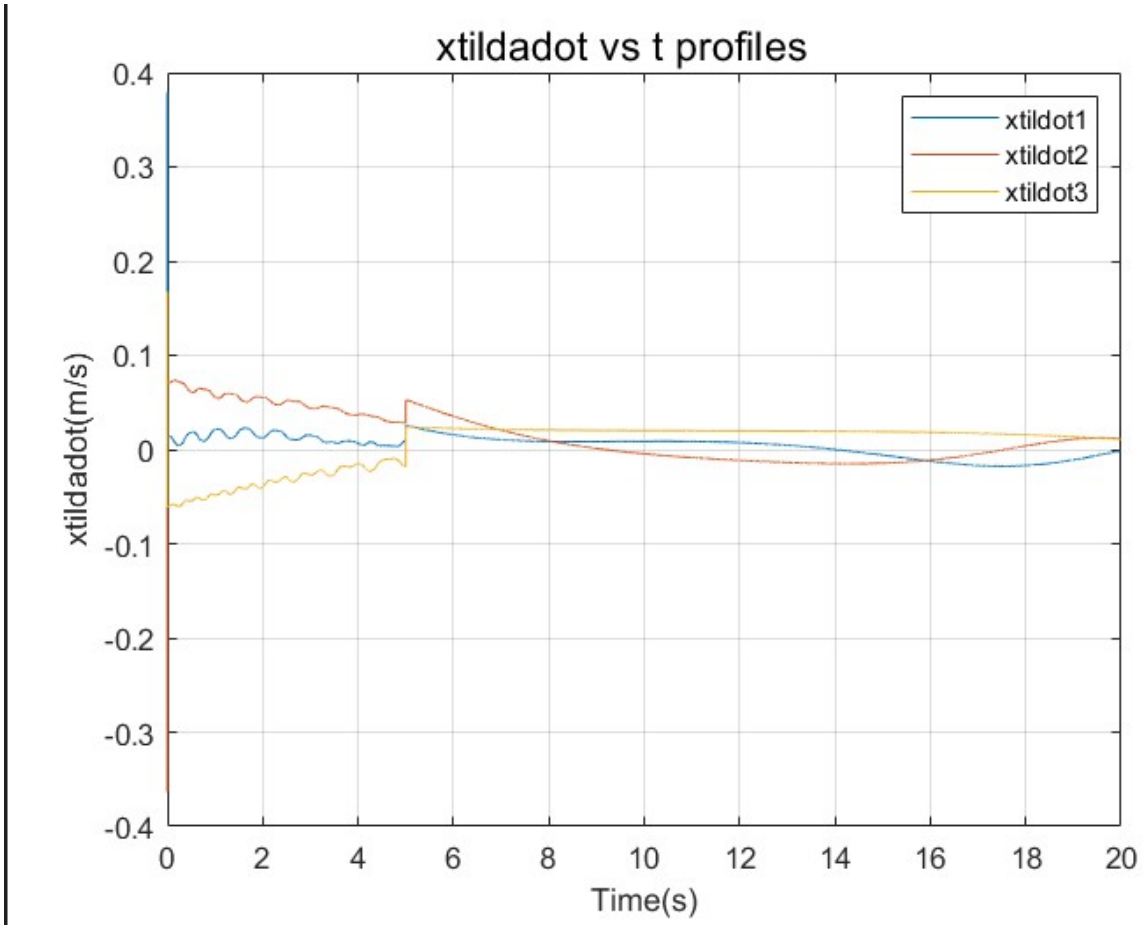
Control parameters: $$u = \left(J\,D^{-1}\right)^{\dagger}\left\{\ddot{x}_{\mathrm{d}} - K_P\,\tilde{x} - K_D\,\dot{\tilde{x}} + J\,D^{-1}\left(C\,\dot{q} + G\right) - \frac{\partial}{\partial q}\left(J\,\dot{q}\right)\dot{q} + \mathrm{proj}\,f\right\},$$

Impedance: $$f_x(t) = f_y(t) = f_z(t) = \begin{cases} 10 & \text{if} \quad 0 \le t \le 5 \\ 0 & \text{otherwise.} \end{cases}$$

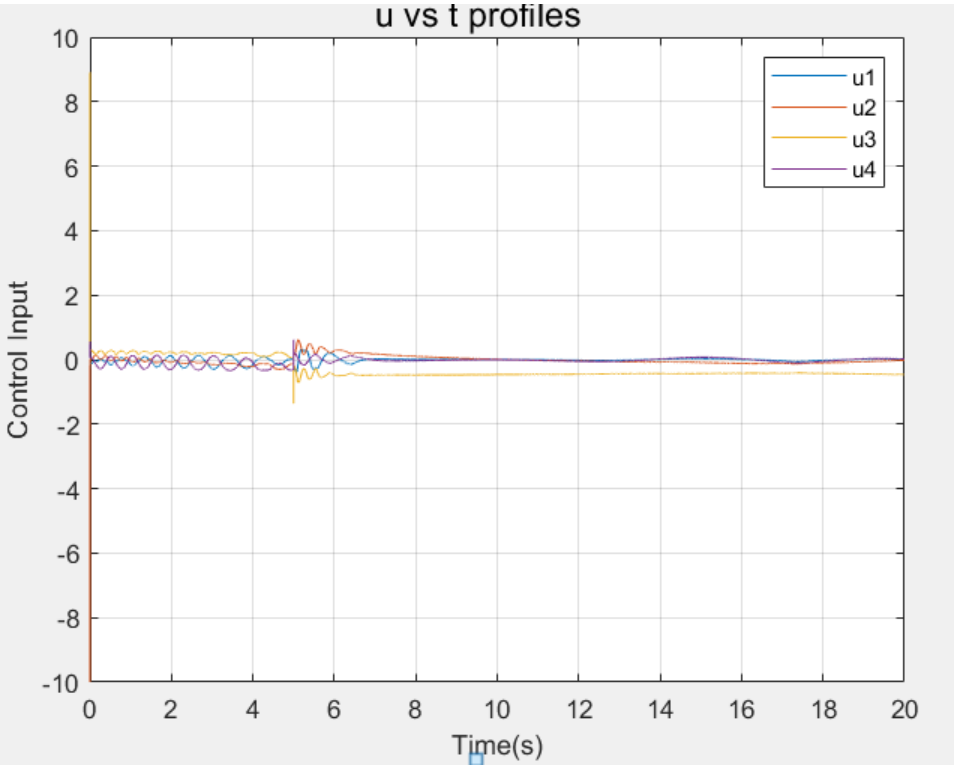
State Space Equation for the ODE: $$\dot{x} = \begin{bmatrix} \dot{q} \\ D^{-1}(q)\left(u(t,q,\dot{q},f) + J^{\top}(q)\,f - C(q,\dot{q})\,\dot{q} - G(q)\right) \end{bmatrix}$$

Plots for Impedance Control:

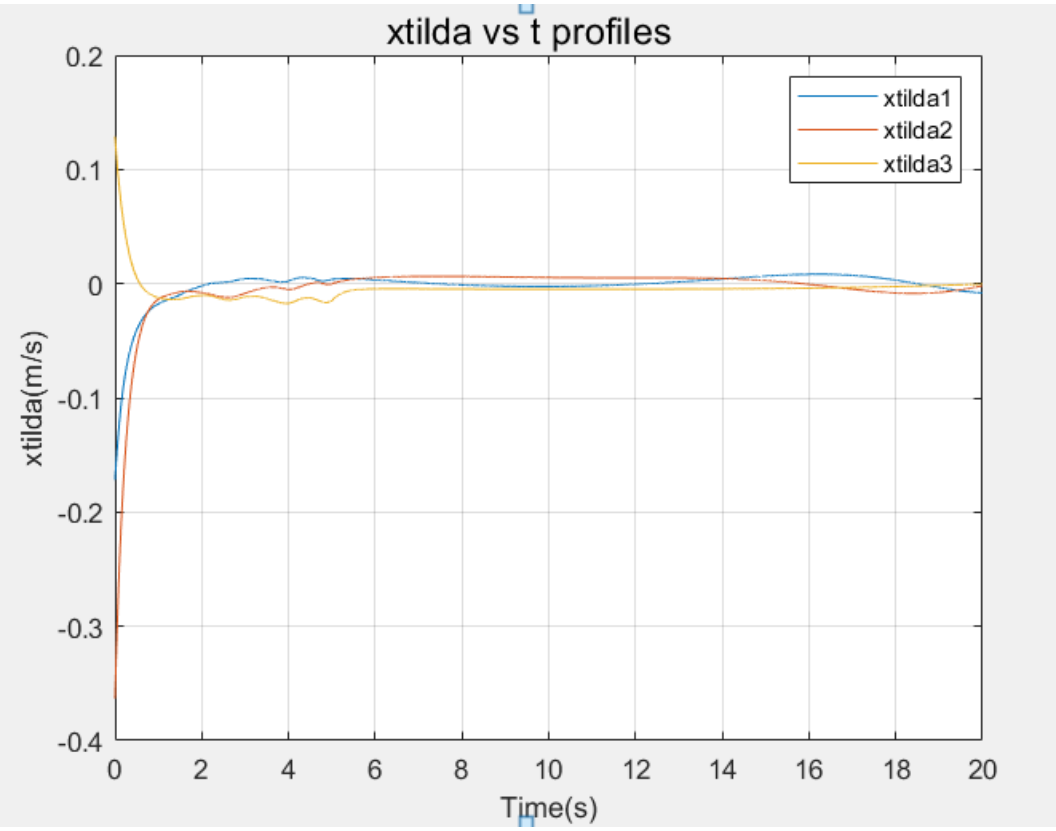
1) Xtildadot vs Time



2) U vs Time



3) Xtilda vs Time



Conclusion: The effect of the impedance is clearly visible for the first 5 seconds after which the curve settles to 0. The xtilda and xtildadot curves in the first 5 seconds can be seen to have some error in the values, which is expected because of the undesired impedance. The effect of which is seen for the increase in control input. For higher values of Kp and Kd the errors appear to converge faster for low values the curves may or may not converge at the end of 20 seconds.

P.S : The codes are structured in such a way that I call some functions(u,bezier) in a for loop for the length of the Time vector in main to get the values that I need to plot for all the three code files, this may take some time to execute(~5mins) for the first two codes and for the last problem it took me a lot of time to execute . I know that this is a very inefficient way of doing things but it was easier for me to debug my code this way and I am falling short of time to change the code to eliminate this for loop that takes majority of the time for the entire functioning of the code. The time to compute the trajectories is very less most of the time is taken by this for loop, but is necessary to get the points for the plot.

Sorry for the hassle.