Robotics 1: Tutorial #2 Submission

Completed by: Jenna Kelly Luchak
CID: 01429938

October 31, 2017

Question 1

My work is written and attached below via the app CamScanner. My final answer is boxed in RED ink.
For reference, the final answer should be equivalent to what is typed directly as follows.

$$\tau_B = H\ddot{q} + \dot{H}\dot{q} - \frac{dT}{dq}$$

$$\tau_B = \begin{bmatrix} \alpha & \beta \\ \beta & \alpha \end{bmatrix} * \begin{bmatrix} \ddot{q_L} \\ \ddot{q_R} \end{bmatrix} + \begin{bmatrix} 0 & -\gamma \\ \gamma & 0 \end{bmatrix} * \begin{bmatrix} \dot{q_L^2} \\ \dot{q_R^2} \end{bmatrix}$$

where,

$$\alpha = 2ml_m^2 + ml^2 + 2I$$
$$\beta = 2mll_m \cos\left(q_R - q_L\right)$$
$$\gamma = 2mll_m \sin\left(q_R - q_L\right)$$

Robotics 1: Tutorial #2 Oct. 17, 2017

- Given: $m = 1.0\ kg$ $l = 0.2\ [m]$ $l_m = 0.1\ [m]$ $I = 0.01\ kg\,m^2$

$$H(q) = \text{mass matrix} = \begin{bmatrix} \alpha & \beta \\ \beta & \alpha \end{bmatrix}, \quad \alpha = 2ml_m^2 + ml^2 + 2I$$
$$\beta = 2mll_m\cos(q_R - q_L)$$

- The lagrange equations are:

$$\tau_B = \frac{d}{dt}\left(\frac{dL}{d\dot{q}}\right) - \frac{dL}{dq} \quad \text{where } L \equiv T - U \text{ and } T = \text{Kinetic energy}$$
U = Potential energy.

- Kinetic energy of a robot $\rightarrow$ $T = \frac{1}{2}\dot{q}^T H(q)\dot{q} = \frac{1}{2}H(q)\dot{q}^2$
- let $G(q)$ = gravity and $G \equiv \frac{dU}{dq}$
- However, assume that the 4 link parallel robot acts in 2D, planar, therefore gravity is negligible and change in potential energy is equal to zero.

$$\therefore \frac{d}{dt}\left(\frac{dL}{d\dot{q}}\right) - \frac{dL}{dq} = \tau_B \implies \frac{d}{dt}\left(\frac{dT}{d\dot{q}}\right) - \frac{dT}{dq} = \tau_B - \cancel{G}^{\,0}, \quad G = \frac{\cancel{dU}^{\,0}}{dq}$$

* Gravity is negligible

$$\frac{d}{dt}\left(\frac{d}{d\dot{q}}\left(\frac{1}{2}H(q)\dot{q}^2\right)\right) = \frac{d}{dt}\left(H(q)\dot{q}\right) = \dot{H}(q)\dot{q} + H(q)\ddot{q}$$

$$\implies \dot{H}\dot{q} + H\ddot{q} - \frac{dT}{dq} = \tau_B \quad \text{—— (1)} \qquad \text{where } \dot{q} = \begin{bmatrix} \dot{q}_L \\ \dot{q}_R \end{bmatrix} \quad \ddot{q} = \begin{bmatrix} \ddot{q}_L \\ \ddot{q}_R \end{bmatrix}$$

- Solve $\dot{H}$

$$H = \begin{bmatrix} 2ml_m^2 + ml^2 + 2I & 2mll_m\cos(q_R - q_L) \\ 2mll_m\cos(q_R - q_L) & 2ml_m^2 + ml^2 + 2I \end{bmatrix}$$

$$\frac{dH}{dt} = \begin{bmatrix} 0 & -2mll_m\sin(q_R - q_L)(\dot{q}_R - \dot{q}_L) \\ -2mll_m\sin(q_R - q_L)(\dot{q}_R - \dot{q}_L) & 0 \end{bmatrix}$$

- Solve $\frac{dT}{dq}$

- Solve dT/dq

$$T = \frac{1}{2}\dot{q}^T H(q)\dot{q} = \frac{1}{2}\begin{bmatrix}\dot{q}_L & \dot{q}_R\end{bmatrix}\begin{bmatrix} 2ml_m^2 + ml^2 + 2I & 2mll_m\cos(q_R - q_L) \\ 2mll_m\cos(q_R - q_L) & 2ml_m^2 + ml^2 + 2I \end{bmatrix}\dot{q}$$

$$= \frac{1}{2}\begin{bmatrix} \dot{q}_L(2ml_m^2 + ml^2 + 2I) + \dot{q}_R(2mll_m\cos(q_R - q_L)) \\ \dot{q}_L(2mll_m\cos(q_R - q_L)) + \dot{q}_R(2ml_m^2 + ml^2 + 2I) \end{bmatrix}^T \begin{bmatrix}\dot{q}_L \\ \dot{q}_R\end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} \dot{q}_L\alpha + \dot{q}_R\beta, & \dot{q}_L\beta + \dot{q}_R\alpha \end{bmatrix}\begin{bmatrix}\dot{q}_L \\ \dot{q}_R\end{bmatrix}$$

$$= \frac{1}{2}\left(\dot{q}_L^2\alpha + \dot{q}_R\dot{q}_L\beta + \dot{q}_L\dot{q}_R\beta + \dot{q}_R^2\alpha\right)$$

$$T = \frac{1}{2}\left(\dot{q}_L^2\alpha + 2\dot{q}_R\dot{q}_L\beta + \dot{q}_R^2\alpha\right)$$

$\therefore$

Note that $\frac{d\dot{q}_L}{dq} = 0$ $\frac{d\dot{q}_R}{dq} = 0$, $q = [q_L\ q_R]^T$

$$\therefore \frac{dT}{dq} = \dot{q}_R\dot{q}_L\frac{d\beta}{dq} = -2mll_m\dot{q}_R\dot{q}_L\sin(q_R - q_L)\begin{bmatrix}-1 \\ 1\end{bmatrix}$$

- Recall equation 1 $\dot{H}\dot{q} + H\ddot{q} - \frac{dT}{dq} = \tau_B$ — (1)

$\therefore$ by plugging in the terms found we can further simplify this...

$$\dot{H}\dot{q} - \frac{dT}{dq} = -2ml_m l \sin(q_R - q_L)(\dot{q}_R - \dot{q}_L)\begin{bmatrix}\dot{q}_R \\ \dot{q}_L\end{bmatrix} + \left(2mll_m \dot{q}_R \dot{q}_L \sin(q_R - q_L)\right)\begin{bmatrix}-1 \\ 1\end{bmatrix}$$

$$\dot{H}\dot{q} - \frac{dT}{dq} = \begin{bmatrix} -2ml_m l \sin(q_R - q_L)\dot{q}_R^2 \\ 2ml_m l \sin(q_R - q_L)\dot{q}_L^2 \end{bmatrix}$$

$$\dot{H}\dot{q} - \frac{dT}{dq} = \begin{bmatrix} 0 & -\gamma \\ \gamma & 0 \end{bmatrix}\begin{bmatrix}\dot{q}_L^2 \\ \dot{q}_R^2\end{bmatrix}$$ where $\gamma = 2ml_m l \sin(q_R - q_L)$

$\therefore$

$$\tau_B = H\ddot{q} + \dot{H}\dot{q} - \frac{dT}{dq}$$

$$\Rightarrow \tau_B = \begin{bmatrix}\alpha & \beta \\ \beta & \alpha\end{bmatrix}\begin{bmatrix}\ddot{q}_L \\ \ddot{q}_R\end{bmatrix} + \begin{bmatrix}0 & -\gamma \\ \gamma & 0\end{bmatrix}\begin{bmatrix}\dot{q}_L^2 \\ \dot{q}_R^2\end{bmatrix}$$

where $\alpha = 2ml_m^2 + ml^2 + 2I$

$\beta = 2mll_m \cos(q_R - q_L)$

$\gamma = 2mll_m \sin(q_R - q_L)$

Question 2

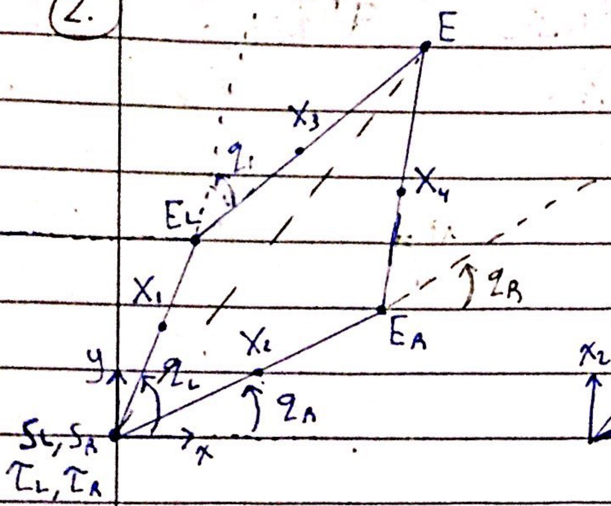
a) /b)

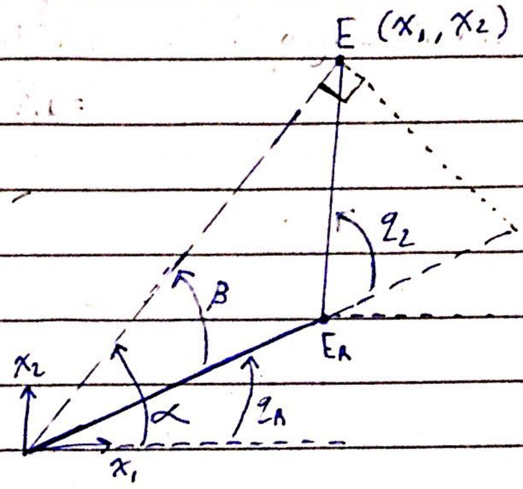
The first step to computing this question was to geometrically determine what the initial angles of q_L and q_R were. The equations to determine these angles as well as the End point trajectories of the robot were determined by hand and then written into a MatLab script to compute the quantitative values. The initial angles were found to be $q_L \approx 60^\circ$ and $q_R \approx 30^\circ$

The hand calculations for Question 2 are attached via the app CamScanner, as follows.

Robotics 1 Oct. 31, 2017

(2.)





Assume
- All links are the same length "l"

- End point "E" coordinates $E = (x_1, x_2)$

$x_1 = l\cos(q_A) + l\cos(q_A + q_2)$ or $x_1 = l\cos(q_L) + l\cos(q_L - q_1)$

$x_2 = l\sin(q_A) + l\sin(q_A + q_2)$ $\quad x_2 = l\sin(q_L) + l\sin(q_L - q_1)$

- From cos law

$$\vec{E} = x_1^2 + x_2^2 = 2l^2 - 2l^2\cos(\pi - q_2) = 2l^2(1 + \cos(q_2))$$

$$\Rightarrow q_2 = \cos^{-1}\left(-1 + \frac{x_1^2 + x_2^2}{2l^2}\right) = \cos^{-1}\left(\frac{x_1^2 + x_2^2 - 2l^2}{2l^2}\right)$$

$$\Rightarrow \alpha = \tan^{-1}\left(\frac{x_2}{x_1}\right)$$

$$\Rightarrow 0 = x_1^2 + x_2^2 - 2l\sqrt{x_2^2 + x_1^2}\cos\beta$$

$$\Rightarrow \beta = \cos^{-1}\left(\frac{\sqrt{x_1^2 + x_2^2}}{2l_1}\right)$$

$$\Rightarrow q_A = \alpha - \beta$$

$$\therefore q_L = q_A + 2\beta = \alpha - \beta + 2\beta = \alpha + \beta$$

$q_R = \alpha - B$

$q_L = \alpha + B$

Based on geometry assume $q_2 + q_A = q_L$ and $q_L - q_1 = q_R$

$$\therefore \quad x_1 = l\cos(q_A) + l\cos(q_L)$$

$$x_2 = l\sin(q_A) + l\sin(q_L)$$

$$E = (x_1, x_2)$$

Question 2 continued…

The MatLab code for Question 2 contains one script and two functions. The code and resulting plots are attached after the following observations to the questions posed in the tutorial assignment.

Observations

2.a) As the gain, K, was increased from 0.01 s, the error in the actual feedback plots was reduced and it agreed more closely with the desired plots. The actual feedforward plots did not change as K increased. However, if K was increased too much, i.e. K~0.06 s, the feedback controller became unstable. The actual feedback plots began to oscillate about the desired plots.

2.b) the feed forward controller improves the controller performance relative to the feedback controller because it takes into consideration the effects of the dynamics of the robot and tries to reduce their effect. The dynamics of the robot cause a disturbance in the process output, for example, there was an error between the desired and actual feedback angles. In general, the feed forward controller tries to minimize these effects (reduce the error) by measuring the error and then calculating a compensation factor. More specifically to this question, the error between the desired and actual feedback angles was calculated, a compensational torque was determined and then new, more accurate angles were identified.

The Matlab script, the Matlab functions, and then the resulting plots for 2A and 2B are as follows respectively.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File name Tutorial_2_question_2_working.m
%% Clear Workspace and Command Window
clc
clear

%% Given Constants and Variables
m=1.0; % Mass of link [kg]
l=0.2; % Length of link [m]
l_m=0.1; % Distance from the joint to the centre of mass of link [m]
I=0.01; % Moment of Inertia [kg*m^2]
T=2; % Total movement duration [s]
ts=0.01; % Timestep
t=0:ts:T; % time vector
w=t/T; % Dimensionless time
K=0.02; % Control Gain [s]
k=100; % Control Gain [Nm]

%% Given and determined Equations for desired position, velocity and
acceleration of the Robot Endpoint
x_d=[0.273-(0.2*(6*w.^5-15*w.^4+10*w.^3));0.273-(0.1*(6*w.^5-
15*w.^4+10*w.^3))]; % Position/trajectory of endpoint
x_d_dot=[-0.2/T^3*(30*t.^4/T^2-60*t.^3/T+30*t.^2);-0.1/T^3*(30*t.^4/T^2-
```

```
60*t.^3/T+30*t.^2)]; % velocity of endpoint
x_d_dot_dot=[-0.2/T^3*(120*t.^3/T^2-180*t.^2/T+60*t.^1);-
0.1/T^3*(120*t.^3/T^2-180*t.^2/T+60*t.^1)]; % Acceleration of endpoint

%% Determine Initial Angles of the desired and actual Shoulder Joints

x1=x_d(1); % x coordinate of endpoint
x2=x_d(2); % y coordinate of endpoint

q_d=zeros(2,T/ts+1); % Empty matrix for desired shoulder joint angles
[left;right]
q_FB=zeros(2,T/ts+1); % Empty matrix for actual shoulder joints [left;right]
q_FF=zeros(2,T/ts+1); % Empty matrix for actual shoulder joints [left;right]

A=atan(x2(1)/x1(1)); % Intermediate step from geometry of question
B=acos(sqrt(x1(1)^2+x2(1)^2)/(2*l)); % Intermiediate step from geometry of
question

q_d(1,1)=A+B; % Initial angle of left shoulder joint [rad]
q_d(2,1)=A-B; % Initial angle of right shoulder joint [rad]

q_FB(1,1)=q_d(1,1); % Initialize actual feedback left angle as equal to
desired
q_FB(2,1)=q_d(2,1); % Initialize actual feedback right angle as equal to
desired

q_FF(1,1)=q_d(1,1); % Initialize actual feedforward left angle as equal to
desired
q_FF(2,1)=q_d(2,1); % Initialize actual feedforward right angle as equal to
desired

%% Create and Initialize empty matrices for the printing of variable data

angular_acceleration_FB=zeros(2,T/ts+1); % Empty matrix for feedback angular
acceleration
angular_velocity_FB=zeros(2,T/ts+1); % Empty matrix for feedback angular
velocity

angular_acceleration_FF=zeros(2,T/ts+1); % Empty matrix for feedforward
angular acceleration
angular_velocity_FF=zeros(2,T/ts+1); % Empty matrix for feedforward angular
velocity

angular_velocity_desired=zeros(2,T/ts+1); % Empty matrix for desired angular
velocity
angular_acceleration_desired=zeros(2,T/ts+1); % Empty matrix for desired
angular acceleration

e=zeros(2,T/ts+1); % Empty matrix for feedback Error function
e_dot=zeros(2,T/ts+1); % Empty matrix for derivative of feedback error
function

eFF=zeros(2,T/ts+1); % Empty matrix for feed forward Error function
eFF_dot=zeros(2,T/ts+1); % Empty matrix for derivative of feedforward error
function
```

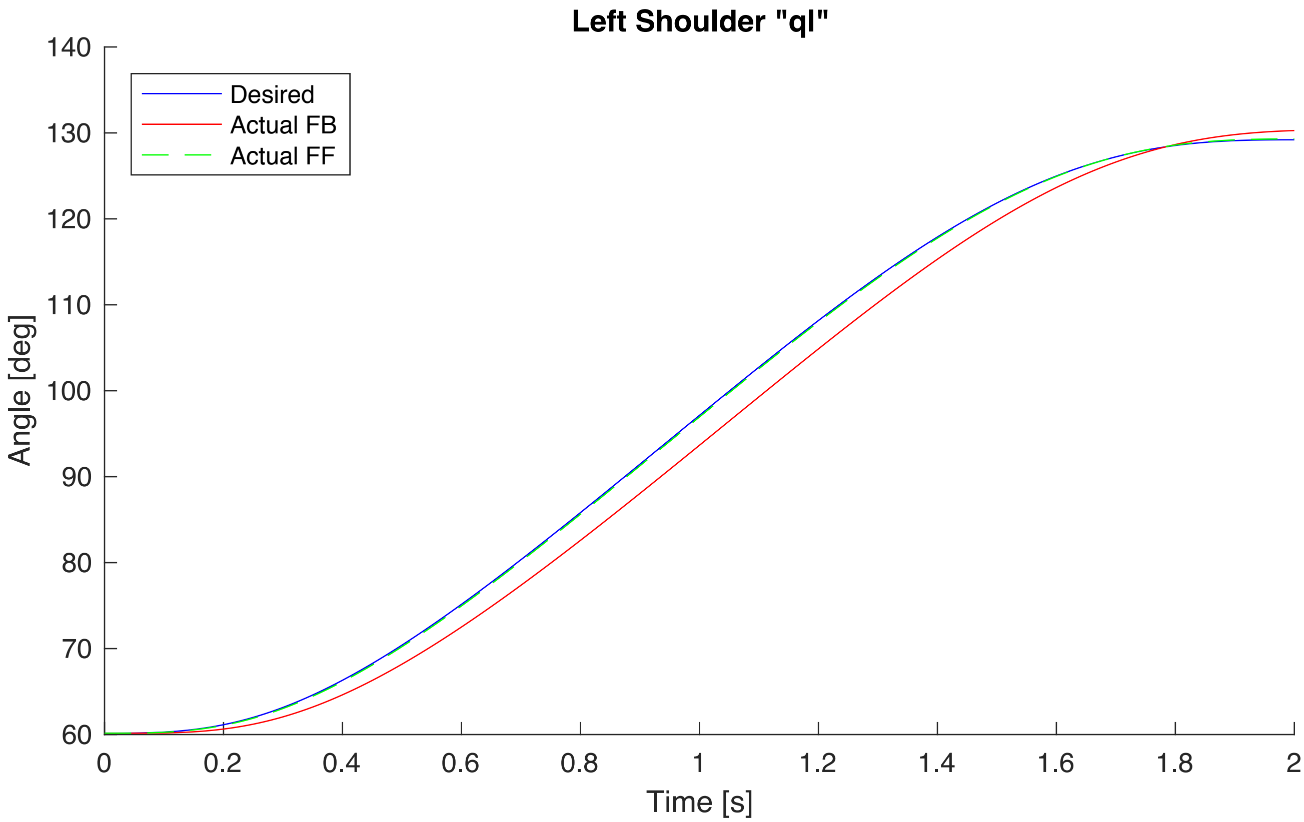
```matlab
%% Create a "for" loop for the feedback controller

for i=2:1:(T/ts+1)

    % Calculate and Print Desired angles

    q_d(1,i)=q_d(1,i-1)+angular_velocity_desired(1,i-1)*ts; % Desired Left
shoulder angles printed into row 1 of matrix q_d
    q_d(2,i)=q_d(2,i-1)+angular_velocity_desired(2,i-1)*ts; % Desired right
shoulder angles printed into row 2 of matrix q_d

    J=Jacobian2(q_d(1,i),q_d(2,i)); % Call the Jacobian Function
    angular_velocity_desired(:,i)=inv(J)*x_d_dot(:,i); % Print desired
angular velocity
    angular_acceleration_desired(:,i)=inv(J)*x_d_dot_dot(:,i); % Print
desired angular acceleration

    % Calculate and Print Actual angles
    q_FB(1,i)=q_FB(1,i-1)+angular_velocity_FB(1,i-1)*ts; % Actual left
shoulder angles
    q_FB(2,i)=q_FB(2,i-1)+angular_velocity_FB(2,i-1)*ts; % Actual right
shoulder angles

    e(:,i)=q_d(:,i)-q_FB(:,i); % Error function between desired and actual
feedback angles
    e_dot(:,i-1)=(e(:,i)-e(:,i-1))/ts; % Derivative of Error function

    torque_FB(:,i-1)=K*(e(:,i-1)+(k*e_dot(:,i-1))); % Linear Feedback
controller torque

    alpha=2*m*l_m^2+m*l^2+2*I; % Intermediate equation "alpha" used in matrix
H calculation
    beta_FB(i)=2*m*l*l_m*cos(q_FB(2,i)-q_FB(1,i)); % Intermediate equation
"beta" used in matrix H calculation

    H=[alpha beta_FB(i);beta_FB(i) alpha]; % Mass matrix "H" of parallel
robot

    gamma_FB(i)=2*m*l*l_m*sin(q_FB(2,i)-q_FB(1,i)); % Intermediate equation
"gamma" used in Matrix V calculation

    V_FB(:,i)=[0 -gamma_FB(i);gamma_FB(i)
0]*[angular_velocity_FB(1,i)^2;angular_velocity_FB(2,i)^2]; % Velocity Matrix

    angular_acceleration_FB(:,i)=inv(H)*(torque_FB(:,i-1)-V_FB(:,i)); %
Calculate angular acceleration
    angular_velocity_FB(:,i)=angular_velocity_FB(:,i-
1)+angular_acceleration_FB(:,i-1)*ts; % Calculate angular velocity
end

%% Create "for" loop for the Feedforward Controller

for i=2:1:(T/ts+1)
```

```
    q_FF(1,i)=q_FF(1,i-1)+angular_velocity_FF(1,i-1)*ts; % Actual left
shoulder angles printed into row 1 of matrix q
    q_FF(2,i)=q_FF(2,i-1)+angular_velocity_FF(2,i-1)*ts; % Actual right
shoulder angles printed into row 2 of matrix q

    angular_velocity_FF(1,i) = angular_velocity_FF(1,i-1) +
angular_acceleration_FF(1,i-1) * ts;
    angular_velocity_FF(2,i) = angular_velocity_FF(2,i-1) +
angular_acceleration_FF(2,i-1) * ts;

    J=Jacobian2(q_FF(1,i),q_FF(2,i)); % Call the Jacobian Function
    J_dot =
Jacobian_dot(q_FF(1,i),q_FF(2,i),angular_velocity_FF(1,i),angular_velocity_FF
(2,i));

    angular_acceleration_desired_FF(:,i) = inv(J)*x_d_dot_dot(:,i) -
inv(J)*J_dot*angular_velocity_FF(:,i);

    % Calculate feedforward angles
    beta_FF(i)=2*m*l*l_m*cos(q_FF(2,i)-q_FF(1,i)); % Intermediate equation
"beta" used in matrix H
    H_FF=[alpha beta_FF(i);beta_FF(i) alpha]; % Mass matrix "H" of parallel
robot

    gamma_FF(i)=2*m*l*l_m*sin(q_FF(2,i)-q_FF(1,i)); % Intermediate equation
"gamma" used in Matrix V
    V_FF(:,i)=[0 -gamma_FF(i);gamma_FF(i)
0]*[angular_velocity_FF(1,i)^2;angular_velocity_FF(2,i)^2]; % Velocity Matrix

    % Calculate Feedforward torque and superpositioned torque
    torque_FF(:,i)=H_FF*angular_acceleration_desired_FF(:,i)+V_FF(:,i);

    eFF(:,i)=q_d(:,i)-q_FF(:,i); % Error function between desired and actual
angles
    eFF_dot(:,i-1)=(eFF(:,i)-eFF(:,i-1))/ts; % Derivative of Error function

    torqueFF_FB(:,i)=K*(eFF(:,i)+(k*eFF_dot(:,i))); % Linear Feedback
controller torque
    torqueTotal(:,i)=torque_FF(:,i)+torqueFF_FB(:,i); % Total torque of
feedback plus feedforward

    angular_acceleration_FF(:,i)=inv(H_FF)*(torqueTotal(:,i)-V_FF(:,i)); %
Calculate angular acceleration
    angular_velocity_FF(:,i)=angular_velocity_FF(:,i-
1)+angular_acceleration_FF(:,i-1)*ts; % Calculate angular velocity
end
%% Calculate Trajectory of Endpoint
% Feedback actual
FB_actual_x_traj=l*cos(q_FB(2,:))+l*cos(q_FB(1,:));
FB_actual_y_traj=l*sin(q_FB(2,:))+l*sin(q_FB(1,:));
% Feedforward actual
FF_actual_x_traj=l*cos(q_FF(2,:))+l*cos(q_FF(1,:));
FF_actual_y_traj=l*sin(q_FF(2,:))+l*sin(q_FF(1,:));

%% Change calculated angles from radians to degrees
% Desired
```

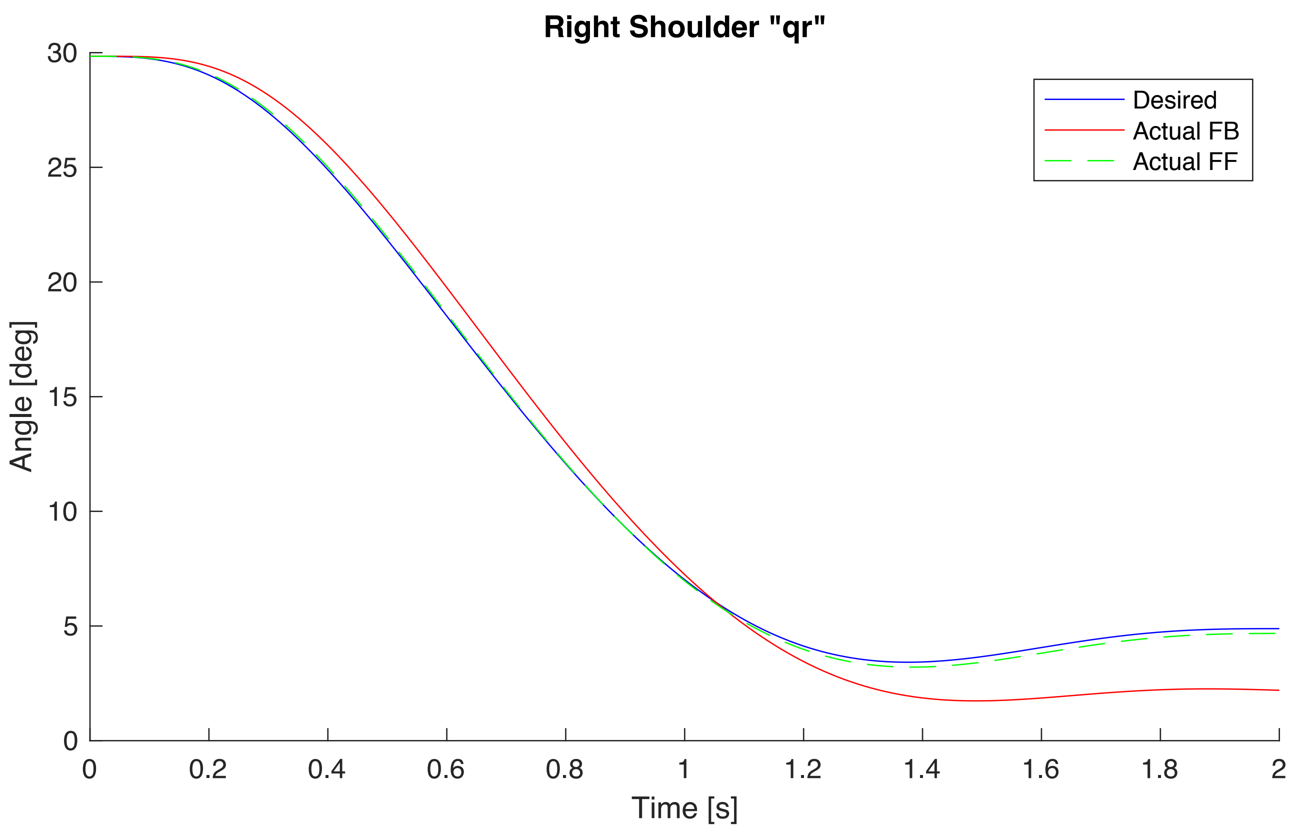
```
Desired_L =rad2deg(q_d(1,:));
Desired_R =rad2deg(q_d(2,:));
% Feedback Actual
Actual__FB_L =rad2deg(q_FB(1,:));
Actual__FB_R =rad2deg(q_FB(2,:));
% Feedforward Actual
Actual_FF_L =rad2deg(q_FF(1,:));
Actual_FF_R =rad2deg(q_FF(2,:));

%% Plot Desired and Actual Angles against time
figure(1)
% ql
subplot(2,2,1:2)
hold on
plot(t,Desired_L,'b'); % Desired plot
plot(t,Actual__FB_L,'r');
plot(t,Actual_FF_L,'g--');
legend('Desired','Actual FB','Actual FF','location','northwest');
title('Left Shoulder "ql"') % Subplot title
xlabel('Time [s]');
ylabel('Angle [deg]');
hold off

% qr
subplot(2,2,3:4)
hold on
plot(t,Desired_R,'b'); % Desired right plot
plot(t,Actual__FB_R,'r'); % Actual right plot
plot(t,Actual_FF_R,'g--');
legend('Desired','Actual FB','Actual FF','location','northeast');
title('Right Shoulder "qr"') % Subplot title
xlabel('Time [s]');
ylabel('Angle [deg]');
hold off

%% Plot Desired and Actual endpoint positions against time
figure(2)
% X position against time
subplot(2,2,1:2)
hold on
plot(t,x_d(1,:),'b'); % Desired
plot(t,FB_actual_x_traj,'r'); % Actual feedback
plot(t,FF_actual_x_traj,'g--'); % Actual feedforward
legend('Desired','Actual FB','Actual FF','location','northeast');
xlabel('Time [s]'); % Time in seconds
ylabel('Position "x1" [m]'); % x coordinate
title(' X Coordinate against Time');
hold off

% Y position against time
subplot(2,2,3:4)
hold on
plot(t,x_d(2,:),'b'); % Desired
plot(t,FB_actual_y_traj,'r'); % Actual feedback
plot(t,FF_actual_y_traj,'g--'); % Actual feedforward
legend('Desired','Actual FB','Actual FF','location','northeast');
xlabel('Time [s]'); % Time in seconds
```

```
ylabel('Position "x2" [m]'); % y coordinate
title(' Y Coordinate against Time');
hold off

%% Plot Desired and Actual Endpoint Trajectories
figure(3)
hold on
plot(x_d(1,:),x_d(2,:),'b'); % Desired
plot(FB_actual_x_traj,FB_actual_y_traj,'r'); % Actual feedback
plot(FF_actual_x_traj,FF_actual_y_traj,'g--'); % Actual feedforward
legend('Desired','Actual FB','Actual FF','location','southeast');
title('Desired and Actual Endpoint Trajectory of Robot');
xlabel('Position "x1" [m]'); % x coordinate
ylabel('Position "x2" [m]'); % y coordinate
hold off

% End of Script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File name Jacobian2.m
% This function is to calculate the Jacobian matrix, and will be called
%  in the script Tutorial_2_question_2_working
function J=Jacobian2(q_l,q_r)
l=0.2;
J=l*[-sin(q_l) -sin(q_r); cos(q_l) cos(q_r)];
end
% End of function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File name Jacobian_dot.m
% This function is to calculate the Jacobian_dot matrix, and will be called
%  in the script Tutorial_2_question_2_working
function J_dot=Jacobian_dot(q_l,q_r,q_l_dot,q_r_dot)
l=0.2;
J_dot = [-l*cos(q_l)*q_l_dot, -l*cos(q_r)*q_r_dot;
         -l*sin(q_l)*q_l_dot, -l*sin(q_r)*q_r_dot];
end
% End of function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2. i)

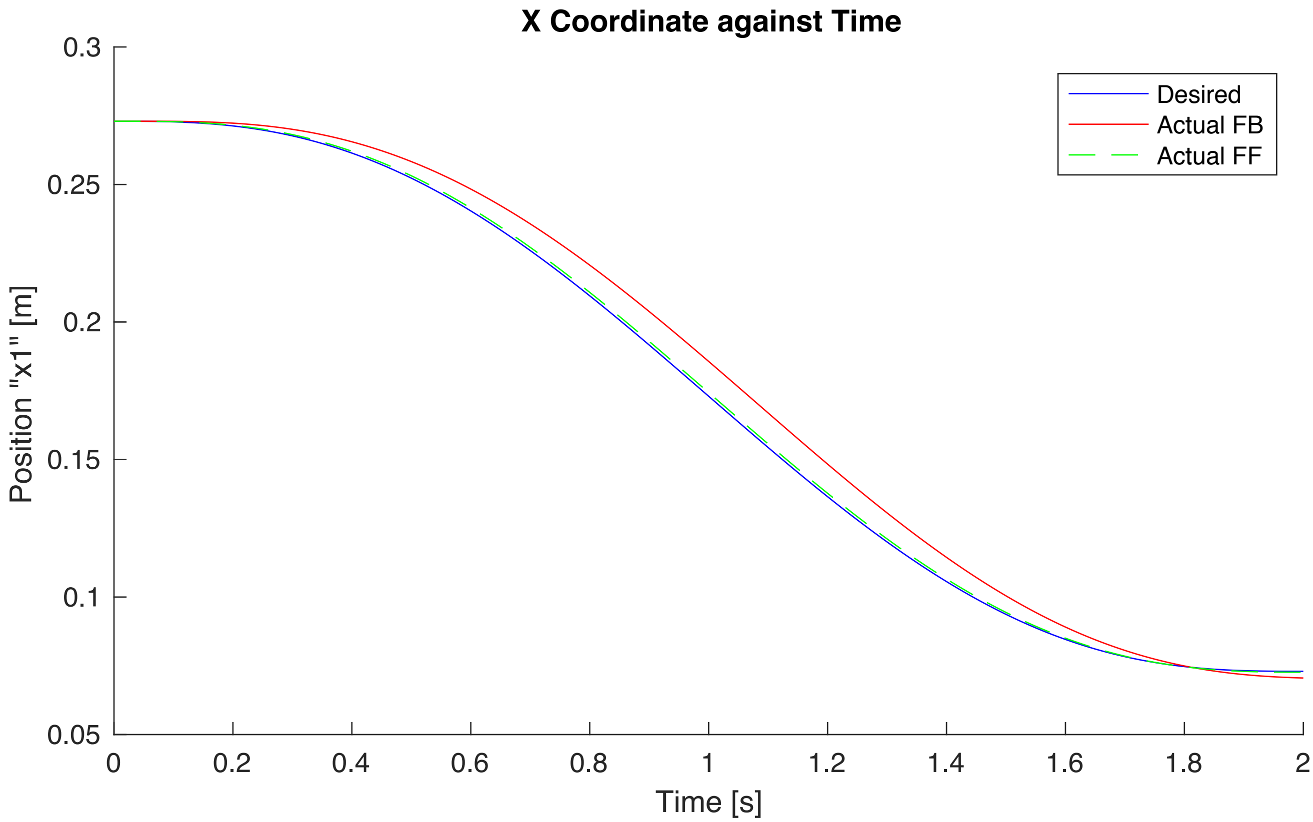


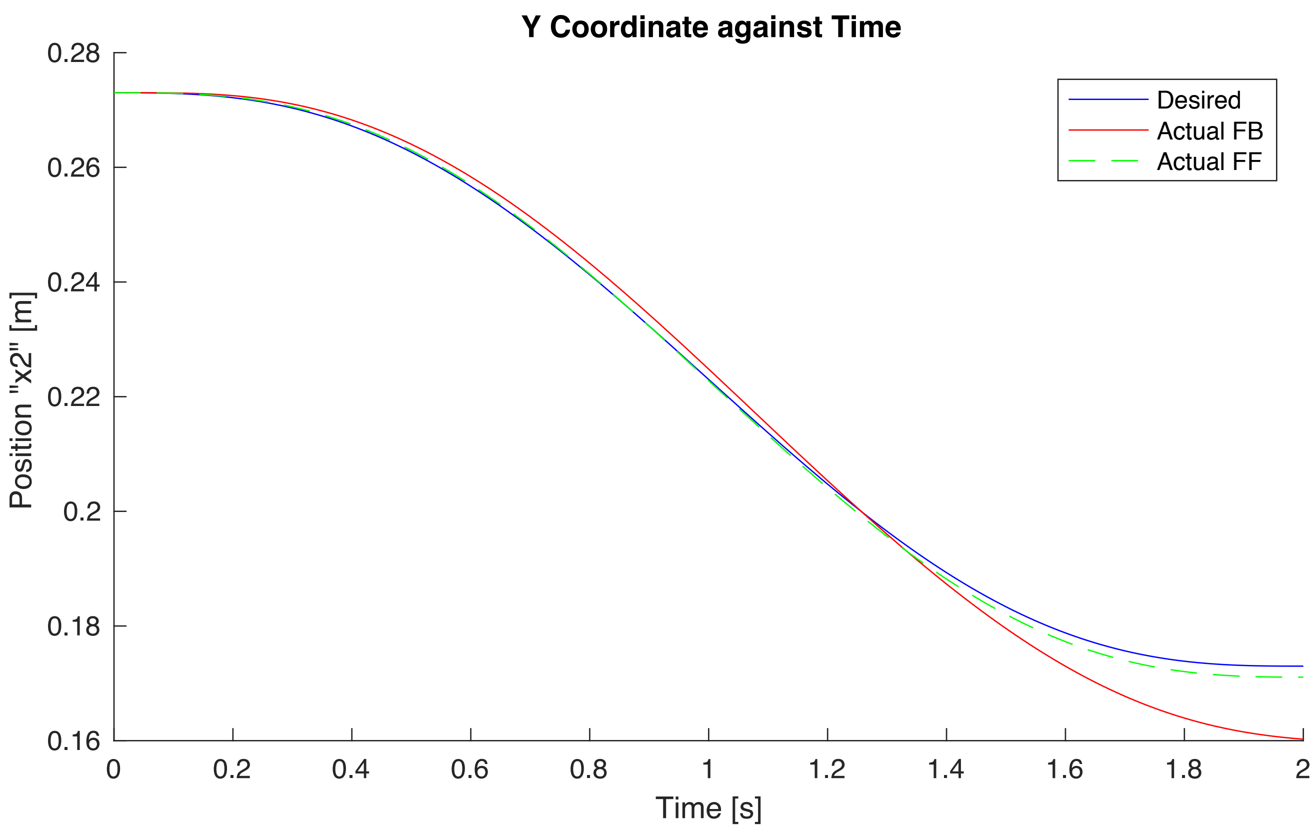
Left Shoulder "ql"
Desired
Actual FB
Actual FF
Angle [deg]
Time [s]
140
130
120
110
100
90
80
70
60
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2



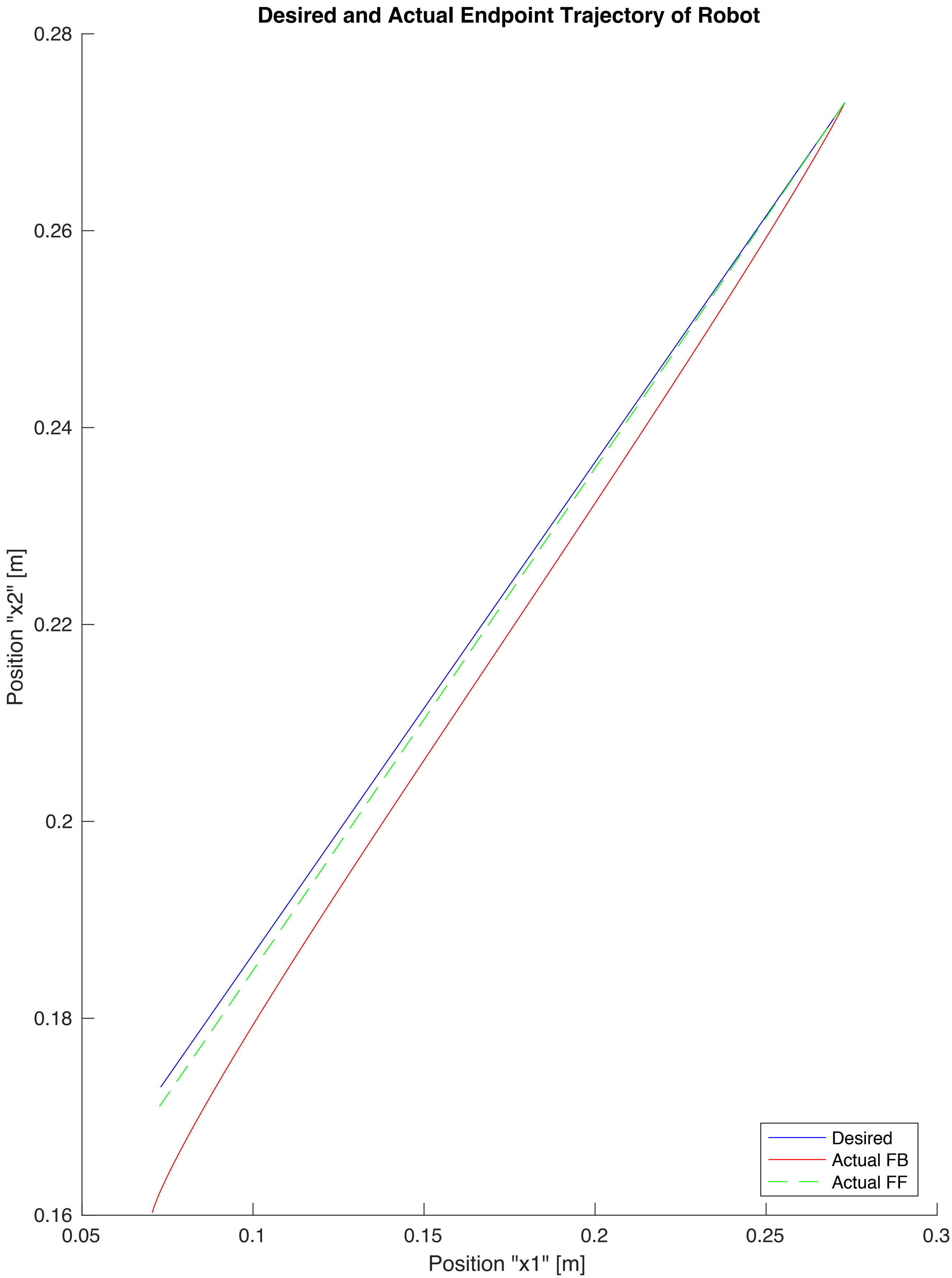
Right Shoulder "qr"
Desired
Actual FB
Actual FF
Angle [deg]
Time [s]
30
25
20
15
10
5
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

2. ii)


X Coordinate against Time
Desired
Actual FB
Actual FF
Position "x1" [m]
0.3
0.25
0.2
0.15
0.1
0.05
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Time [s]


Y Coordinate against Time
Desired
Actual FB
Actual FF
Position "x2" [m]
0.28
0.26
0.24
0.22
0.2
0.18
0.16
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Time [s]

2. iii)



Desired and Actual Endpoint Trajectory of Robot
0.28
0.26
0.24
0.22
0.2
0.18
0.16
Position "x2" [m]
0.05
0.1
0.15
0.2
0.25
0.3
Position "x1" [m]
Desired
Actual FB
Actual FF