Mech 564 Project Report

Paresh Bhambhani

May 14, 2016

Contents

1 Design of the Task Level Controller

1.1 Derivation of the nonlinear controller to linearize and decouple the robot dynamics model in the task space

The robot dynamic equation is:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{1}$$

$$\text{The forward kinematics equation is:} \quad Y = h(q) \tag{2}$$
$$\text{implies: } \dot{Y} = J(q)\dot{q} \tag{3}$$
$$\ddot{Y} = \dot{J}(q)\dot{q} + J(q)\ddot{q} \tag{4}$$
$$\text{implies } \ddot{q} = J^{-1}(q)[\ddot{Y} - \dot{J}(q)\dot{q}] \tag{5}$$

replacing $\ddot{q}$ in the dynamics equation we derive the task level dynamics model.

$$D(q)J^{-1}(q)[\ddot{Y} - \dot{J}(q)\dot{q}] + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{6}$$

Here the term $\ddot{Y}$ is the task level variable. To linearize the dynamics we need to design a feedback that will linearize the model. Hence the non linear feedback will be:

$$\tau = D(q)J^{-1}(q)U - D(q)J^{-1}\dot{J}\dot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{7}$$

Plugging this into the dynamics model,

$$D(q)J^{-1}(q)[\ddot{Y} - \dot{J}(q)\dot{q}] + C(q,\dot{q})\dot{q} + g(q) = D(q)J^{-1}(q)U - D(q)J^{-1}\dot{J}\dot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{8}$$

implies

$$D(q)J^{-1}(q)[\ddot{Y} - U] = 0 \tag{9}$$
$$\ddot{Y} = U \tag{10}$$

1.2 Linearized robot dynamics model in the state space form and designing of Linear controller

Y is represented as $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ and as derived in equation 10, $\ddot{Y} = U$.

For the state space representation let, X be $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$ $where x_1 = x, x_2 = \dot{x}, x_3 = y, x_4 = \dot{y}, x_5 = z, x_6 = \dot{z}.$

Hence $\dot{x_1} = x_2, \dot{x_2} = u_1, \dot{x_3} = x_4, \dot{x_4} = u_2, \dot{x_5} = x_6, \dot{x_6} = u_3$.

Comparing with $\dot{X} = AX + BU$,

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

A linear PD controller design can be used for designing $U = \ddot{Y}$. Let,

$$U = \ddot{Y_d} + K_D(\dot{Y_d} - \dot{Y}) + K_P(Y_d - Y) \tag{11}$$

The error dynamics will then be,

$$(\ddot{Y_d} - \ddot{Y}) + K_d(\dot{Y_d} - \dot{Y}) + K_p(Y_d - Y) = 0 \tag{12}$$

$$\ddot{e} + K_d\dot{e} + K_pe = 0 \tag{13}$$

K_d and K_p need to be designed in a way that $e \to 0$ as $t \to \infty$.

2 Simulation

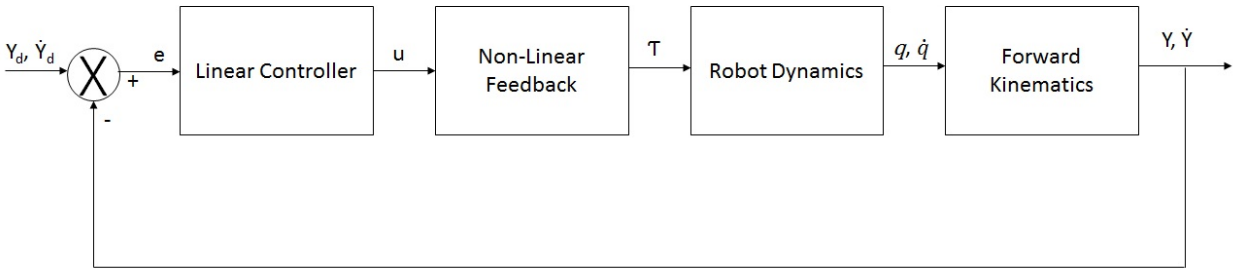

Figure 1: The control system

2.1 Description

Initializations The initial Y and $\dot{Y}$ were given using which the initial q and $\dot{q}$ were calculated using the "inverse_kinematics.m" script. The initial Y_d and $\dot{Y_d}$ were calculated using the "forward_kinematics.m" script using $t = 0$.

Desired Trajectory As per the given trajectory equation, the "traj.m" script takes in the ω and time t and produces Y_d and $\dot{Y}_d$.

Error The error between the desired trajectory and the output Y and $\dot{Y}_d$ is calculated in the "main.m" script to be passed on to the Linear controller.

Linear Control Linear control law is applied in the "main.m" script as per the law in equation 11.

Non-Linear feedback The non-linear feedback is calculated in the "non_linear_fb.m" script which takes in the u from the Linear controller and the previous q and $\dot{q}$ to generate τ.

Dynamics The robot dynamics equation 1 was implemented as per the code in Appendix, using the MATLAB function ODE45 since it makes use of Runge-Kutta 4^{th} and 5^{th} order methods.
The dynamics script takes in the torque τ and gives out $q, \dot{q}$

Forward Kinematics The "forward_kinmeatics.m" scripts generates the new Y and $\dot{Y}$ to be fed back to the comparator to generate the error and repeat the cycle.

3 Simulation Results

3.1 Trajectory Tracking

3.1.1 Trajectory Tracking Error Case 1

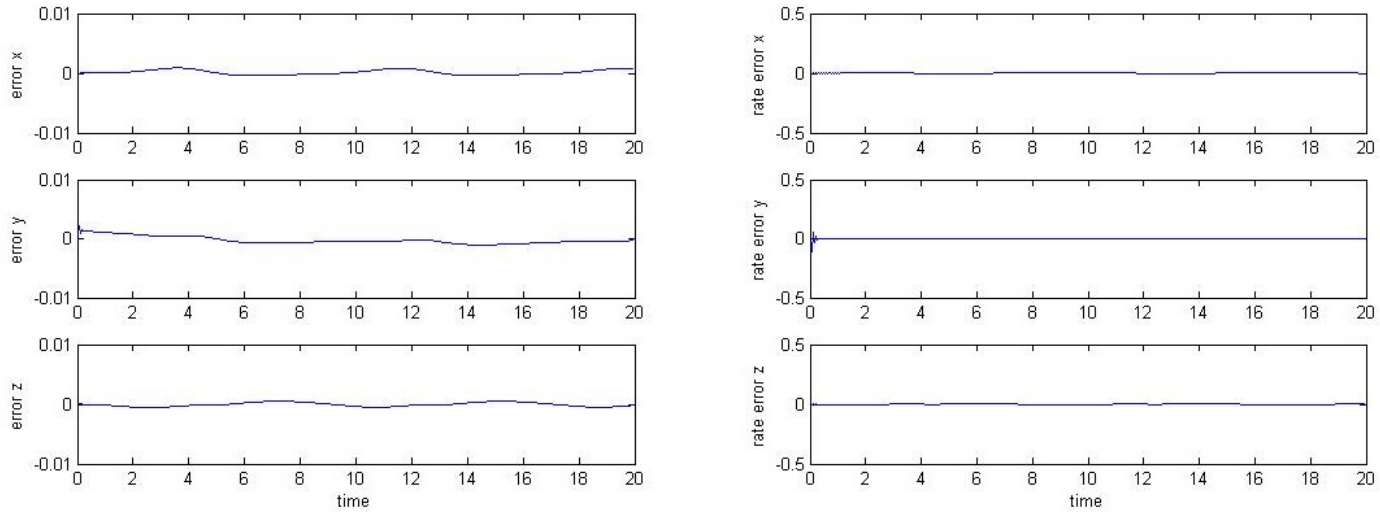

Figure 2: Error and Rate of Error

Figure 2 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/4, \dot{Y}(0) = 0$ and $Y(0) = [-0.7765m; 0.0m; 0.045m]^T$

3.1.2 Trajectory Tracking Error Case 2

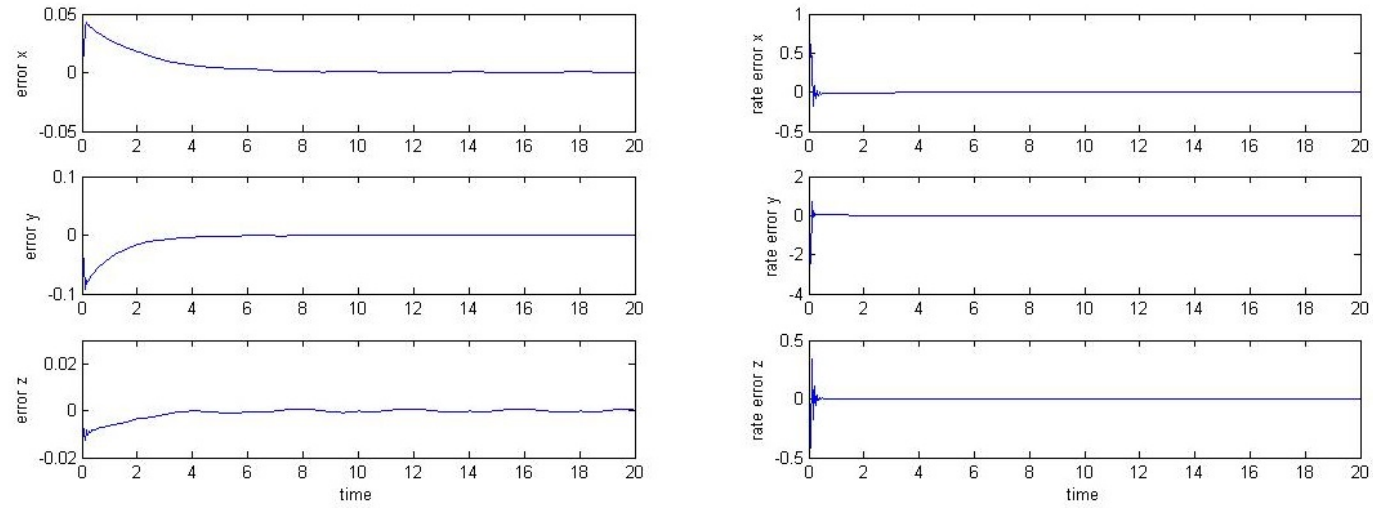

Figure 3: Error and Rate of Error

Figure 3 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/2, \dot{Y}(0) = 0$ and $Y(0) = [-0.5m; -0.1m; 0.0m]^T$

3.2 Effects of Variance in the dynamics model's $\mathbf{d_{kj}; c_{ijk}}$ and $\mathbf{g_i}$

3.2.1 Variance of 2% in Case 1

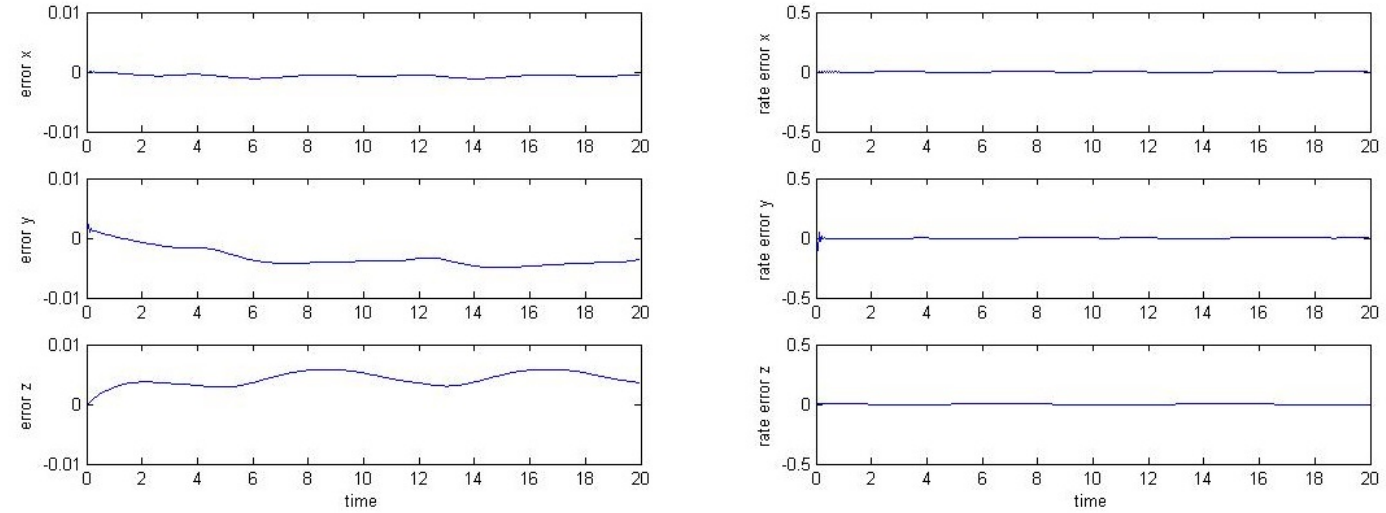

Figure 4: Error and Rate of Error

Figure 4 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/4, \dot{Y}(0) = 0$ and $Y(0) = [-0.7765m; 0.0m; 0.045m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 2%.

3.2.2 Variance of 5% in Case 1

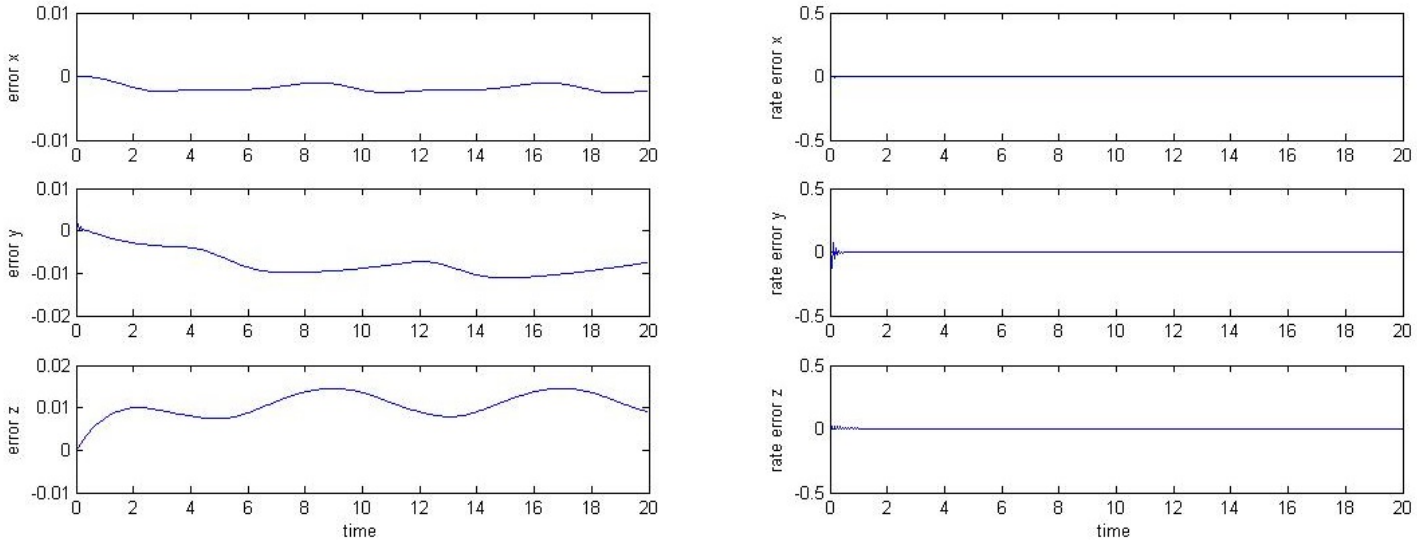

Figure 5: Error and Rate of Error

Figure 5 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/4, \dot{Y}(0) = 0$ and $Y(0) = [-0.7765m; 0.0m; 0.045m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 5%.

3.2.3 Variance of 10% in Case 1

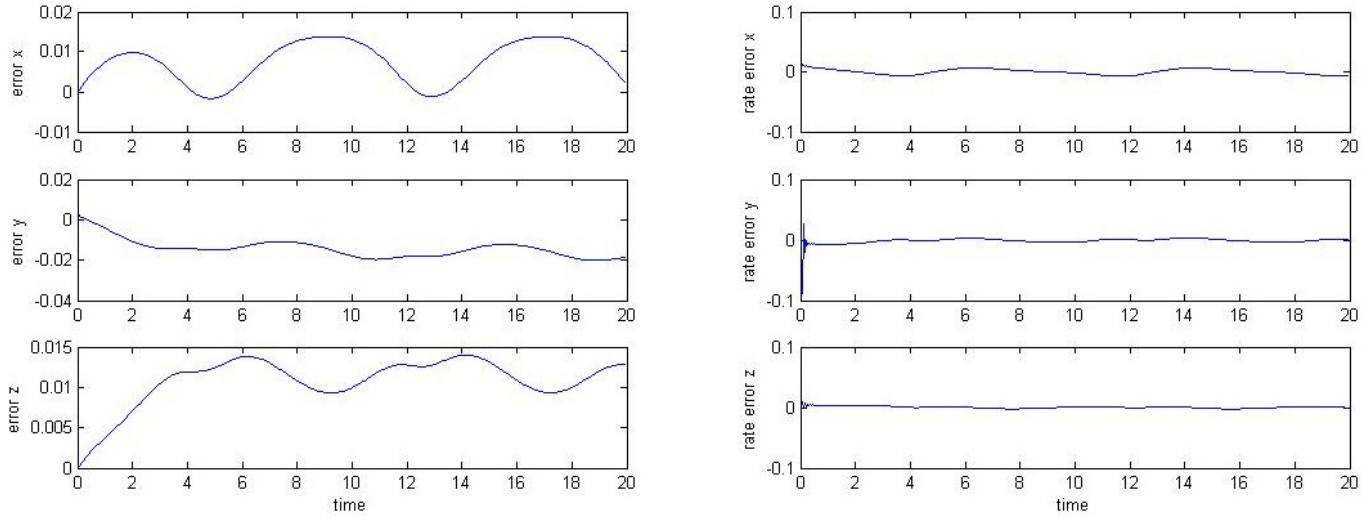

Figure 6: Error and Rate of Error

Figure 6 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/4, \dot{Y}(0) = 0$ and $Y(0) = [-0.7765m; 0.0m; 0.045m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 10%.

3.2.4 Variance of 30% in Case 1

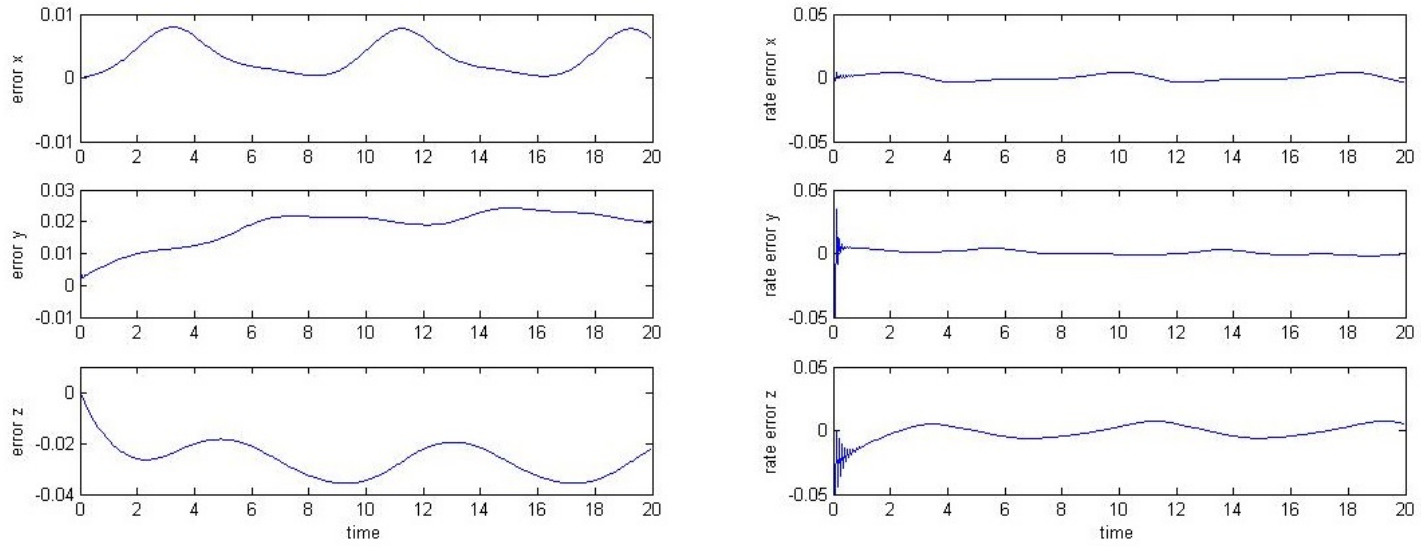

Figure 7: Error and Rate of Error

Figure 7 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/4, \dot{Y}(0) = 0$ and $Y(0) = [-0.7765m; 0.0m; 0.045m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 30%.

3.2.5 Variance of 2% in Case 2

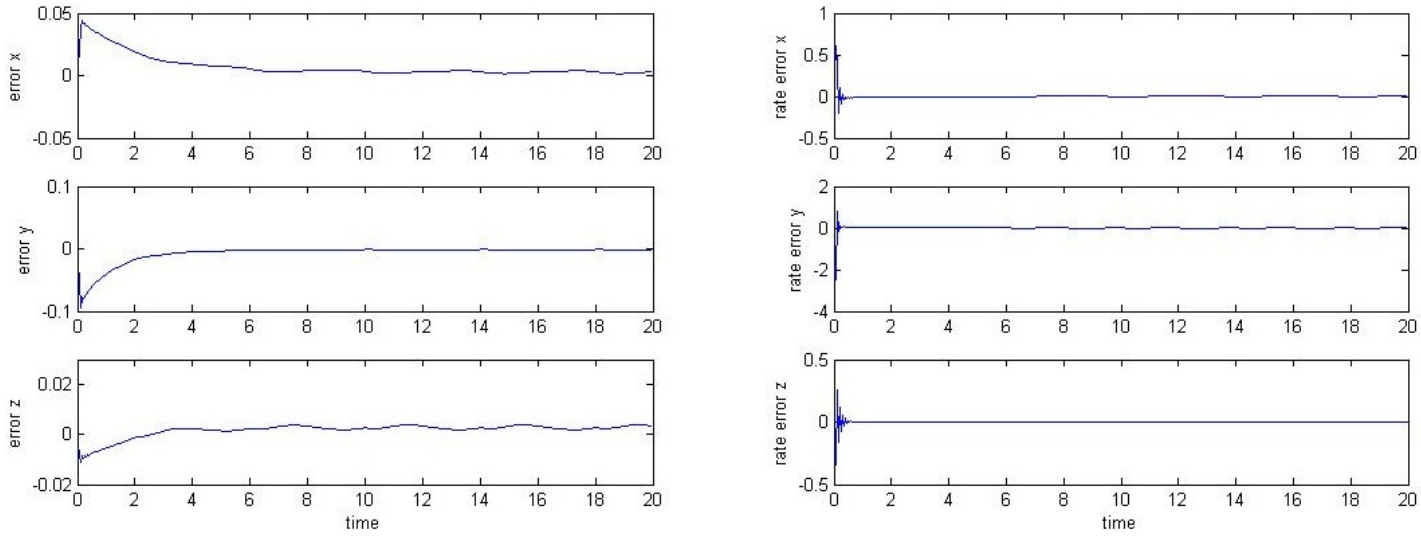

Figure 8: Error and Rate of Error

Figure 8 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/2, \dot{Y}(0) = 0$ and $Y(0) = [-0.5m; -0.1m; 0.0m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 2%.

3.2.6 Variance of 5% in Case 2

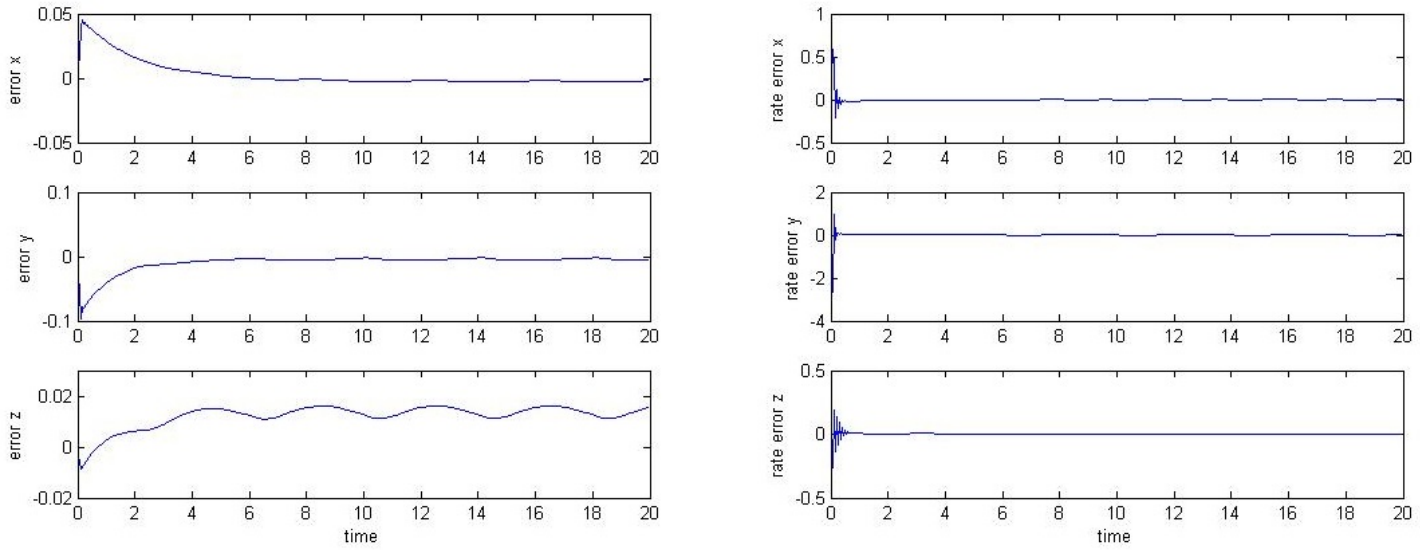

Figure 9: Error and Rate of Error

Figure 9 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/2, \dot{Y}(0) = 0$ and $Y(0) = [-0.5m; -0.1m; 0.0m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 5%.

3.2.7 Variance of 10% in Case 2

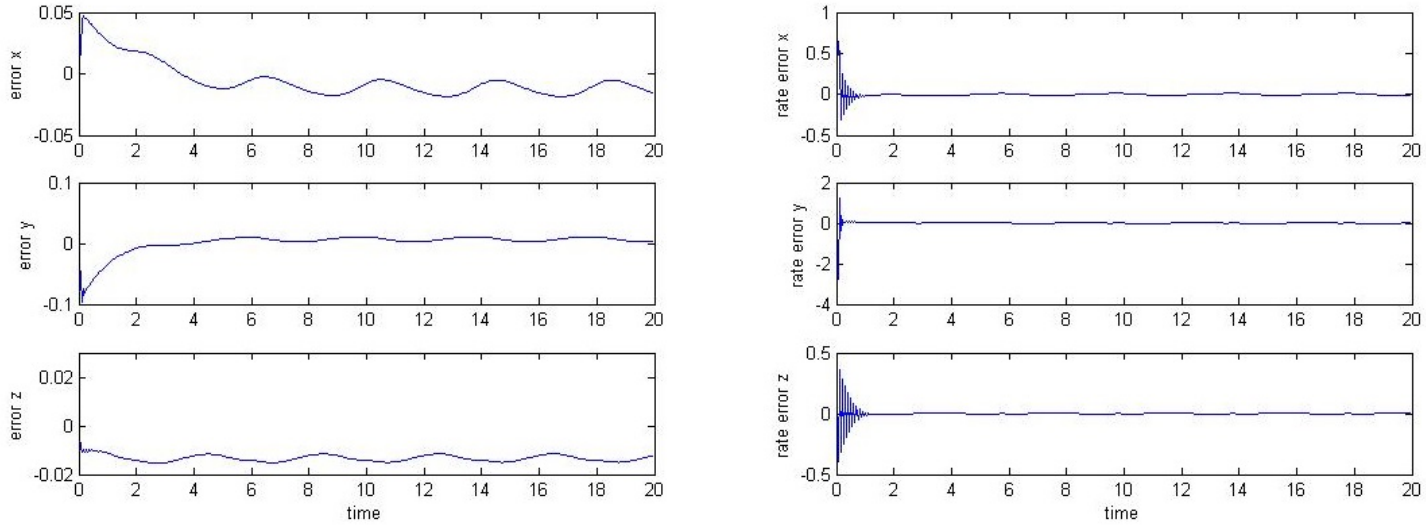

Figure 10: Error and Rate of Error

Figure 10 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/2, \dot{Y}(0) = 0$ and $Y(0) = [-0.5m; -0.1m; 0.0m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 10%.

3.2.8 Variance of 30% in Case 2

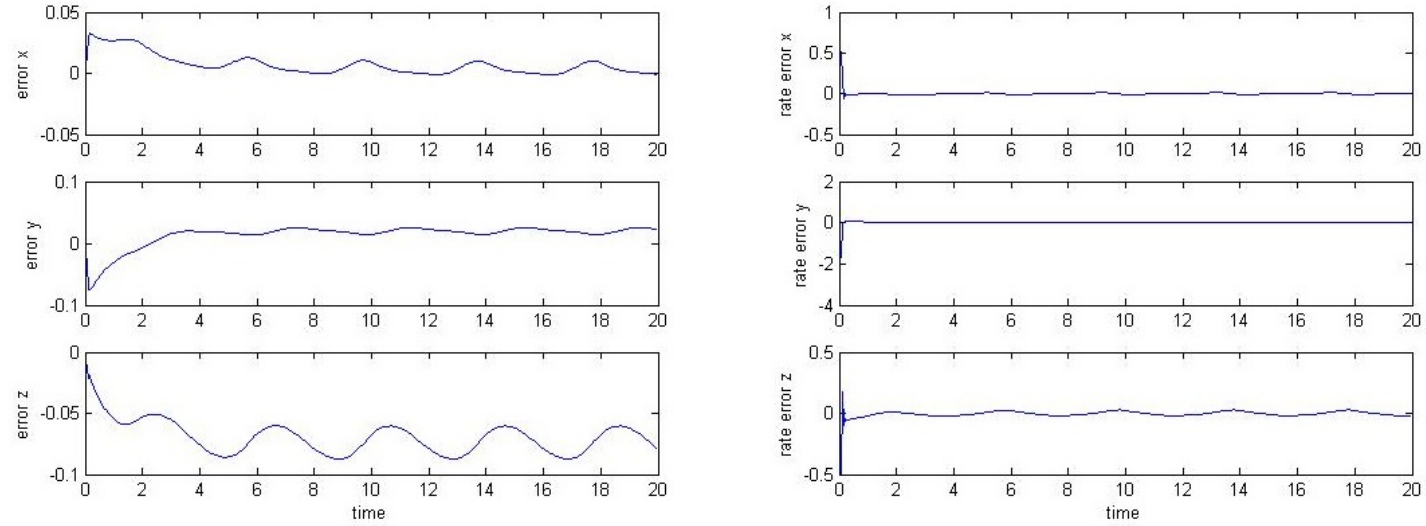

Figure 11: Error and Rate of Error

Figure 11 shows the plot of $\begin{bmatrix} Y_d - Y \\ \dot{Y}_d - \dot{Y} \end{bmatrix}$ for $\omega = \pi/2, \dot{Y}(0) = 0$ and $Y(0) = [-0.5m; -0.1m; 0.0m]^T$ when $d_{kj}; c_{ijk}$ and g_i vary by 30%.

4 Observations and Discussion

4.1 Linearization and Decoupling

In order to develop a controller for a system with non linear dynamics it is feasible and advisable to convert the dynamics to linear dynamics. Although it is an approximation of the original dynamics, it holds true in the region around the equilibrium point. There are multiple techniques out there which are used for linearizion depending on the system and requirements. Applying the non linear feedback on the dynamics achieves linearizion and decoupling of the system. Meaning each input can be designed to control a scalar linear system and each input will affect the individual output independently of the motion of other links.

4.2 Effects of robot's initial position and the velocity of the desired trajectory on the tracking errors

As seen from Figure 2 and Figure 3, when the initial position of the robot is away from the desired trajectory, the error is large initially. This is expected since the difference between Y_d and Y will be large at the start. Hence the setting time, t_s will be slightly larger for the case where the initial position is far off.

4.3 Effects of modeling errors on the robot tracking control

Plots from Section 3.2 show that as the error increases in the robot dynamics model, the tracking error increases and oscillates more and more. As can be seen from Figure 4 and Figure 8, the controller is robust for a small variation of 2% error in the dynamics model although minor oscillations start to appear. Subsequent increase in the error shows that the controller fails to perform and the error as well as the oscillations increase.

4.4 Methods to reduce the effects of modeling errors

A controller should be robust enough to compensate for errors in model. However a bound on the error is needed in order to achieve designing a robust controller. Various methods and techniques are available to tackle the problem of modeling errors. Few of them are Adaptive control, Lyapanov theory and Fuzzy control.
One of the methods is the Robust Inverse Dynamics as explained in Chapter 8 of the book. From Equation 1 we have

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

The inverse dynamics control input is

$$\tau = \hat{D}(q)J^{-1}(q)U - \hat{D}(q)J^{-1}\dot{J}\dot{q} + \hat{C}(q,\dot{q})\dot{q} + \hat{g}(q)$$

Here $\hat{(.)}$ notation is symbolic of the uncertainties in the system which can be due to the modeling or computation errors. Solving these two equation will not result in negation of the C and g terms as we had before. This retention of these terms accounts for the uncertainties in the system. Hence instead of $\ddot{q} = J^{-1}(q)[\ddot{Y} - \dot{J}(q)\dot{q}]$ we now have

$$\ddot{q} = J^{-1}(q)[\ddot{Y} - \dot{J}(q)\dot{q}] + \eta(q,\dot{q},[\ddot{Y} - \dot{J}(q)\dot{q}])$$

Here η is the uncertainty. Various methods are used to design the outer loop control for this system now which incorporates the uncertainty one of them being Lyapanov's second method.

Appendices

```
%---------------------------------- symbolic_derivations.m ------------------------------------%

%% Symbolic derivations
%% This script is to calculate the Jacobian and Yd symbolically to use in other scripts to avoid
    recalculation for each iteration

%Jacobian
% Robot Params
a = [0 431.8/1000 -20.32/1000 0]; %ai in m
d = [0 149.09/1000 0 433.07/1000]; %di in m

syms q1 q2 q3

q_sym = [q1; q2; q3];

h1 = a(3)*cos(q1)*cos(q2+q3) + d(4)*cos(q1)*sin(q2+q3) + a(2)*cos(q1)*cos(q2) - d(2)*sin(q1);
h2 = a(3)*sin(q1)*cos(q2+q3) + d(4)*sin(q1)*sin(q2+q3) + a(2)*sin(q1)*cos(q2) + d(2)*cos(q1);
h3 = - a(3)*sin(q2+q3) + d(4)*cos(q2+q3) - a(2)*sin(q2);
h = [h1; h2; h3];

Y = h % Y = h(q)

J = jacobian(h,q_sym)

%% Yd' and Yd"

syms t w R
yd = [-0.866*R*cos(w*t)-0.56 ;R*sin(w*t) ; 0.5*R*cos(w*t)-0.08];
yd_dot = diff(yd,t)
yd_dot_dot = diff(yd,t,2)
```

```
%---------------------------------- inverse_kinematics.m ------------------------------------%

%%Inverse Kinematics for q(0) and q'(0)

%function [q,q_dot] = inverse_kinematics(Yd,Yd_dot)

%Robot parameters
a = [0 431.8/1000 -20.32/1000 0]; %ai in m
d = [0 149.09/1000 0 433.07/1000]; %di in m

%Initial Y and Y'
Y_zero_1 = [-0.7765 ; 0 ; 0.045];
Y_zero_2 = [-0.5 ; -0.1 ; 0];
Y_dot_zero_1 = [0;0;0];
Y_dot_zero_2 = [0;0;0];

%options = optimoptions(@fsolve,'Display','iter','Jacobian','on');
kin_mod_1 = @root3d_init_1;
q0_1 = [0,0,0];
q_init_1 = fsolve(kin_mod_1,q0_1)
q0_2 = [pi/4,-pi/4,pi/4];
kin_mod_2 = @root3d_init_2;
q_init_2 = fsolve(kin_mod_2,q0_2)
```

```
q_dot_init_1 = [0;0;0]; %Since Y'_1(0) = [0;0;0]
q_dot_init_2 = [0;0;0]; %Since Y'_2(0) = [0;0;0]
```

```
%---------------------------------------- traj.m ----------------------------------------%

%% To generate Yd and Y'd
function [yd yd_dot yd_dot_dot] = traj(w,t)
R = 0.25;
yd = [-0.866*R*cos(w*t)-0.56 ;R*sin(w*t) ; 0.5*R*cos(w*t)-0.08];
yd_dot = [(433*R*w*sin(t*w))/500 ; R*w*cos(t*w); -(R*w*sin(t*w))/2]; %From symbolic derivation
    script
yd_dot_dot = [ (433*R*w^2*cos(t*w))/500; -R*w^2*sin(t*w); -(R*w^2*cos(t*w))/2]; %From symbolic
    derivation script
```

```
%------------------------------------ root3d_init_1.m ----------------------------------------%

%% Forward kinematics model to be used to calculate q(0) and q'(0) via
%% inverse kinematics when Y(0) = [-0.7765 ; 0 ; 0.045]

function F = root3d_init_1(q)
%Robot parameters
a = [0 431.8/1000 -20.32/1000 0]; %ai in m
d = [0 149.09/1000 0 433.07/1000]; %di in m
Y = [-0.7765 ; 0 ; 0.045];

F(1) = a(3)*cos(q(1))*cos(q(2)+q(3)) + d(4)*cos(q(1))*sin(q(2)+q(3)) + a(2)*cos(q(1))*cos(q(2)) -
    d(2)*sin(q(1)) - Y(1);
F(2) = a(3)*sin(q(1))*cos(q(2)+q(3)) + d(4)*sin(q(1))*sin(q(2)+q(3)) + a(2)*sin(q(1))*cos(q(2)) +
    d(2)*cos(q(1)) - Y(2);
F(3) = - a(3)*sin(q(2)+q(3)) + d(4)*cos(q(2)+q(3)) - a(2)*sin(q(2)) - Y(3);
```

```
%------------------------------------ root3d_init_2.m ----------------------------------------%

%% Forward kinematics model to be used to calculate q(0) and q'(0) via
%% inverse kinematics when Y(0) = [-0.5;-0.1; 0.0]

function F = root3d_init_2(q)
%Robot parameters
a = [0 431.8/1000 -20.32/1000 0]; %ai in m
d = [0 149.09/1000 0 433.07/1000]; %di in m
Y = [-0.5;-0.1; 0.0];

F(1) = a(3)*cos(q(1))*cos(q(2)+q(3)) + d(4)*cos(q(1))*sin(q(2)+q(3)) + a(2)*cos(q(1))*cos(q(2)) -
    d(2)*sin(q(1)) - Y(1);
F(2) = a(3)*sin(q(1))*cos(q(2)+q(3)) + d(4)*sin(q(1))*sin(q(2)+q(3)) + a(2)*sin(q(1))*cos(q(2)) +
    d(2)*cos(q(1)) - Y(2);
F(3) = - a(3)*sin(q(2)+q(3)) + d(4)*cos(q(2)+q(3)) - a(2)*sin(q(2)) - Y(3);
```

```
%------------------------------------ non_linear_fb.m ----------------------------------------%

% non linear feedback
% tau = D(q)*J^(-1)*u - D(q)*J^(-1)J'q' + C(q,q')q' + G(q)

function tau = non_linear_fb(u,q,q_dot)

q1 = q(1);
q2 = q(2);
q3 = q(3);
q1_dot = q_dot(1);
q2_dot = q_dot(2);
q3_dot = q_dot(3);


%J
J11 = 0.02032*cos(q2 + q3)*sin(q1) - 0.4318*cos(q2)*sin(q1) - 0.14909*cos(q1) - 0.43307*sin(q2 +
    q3)*sin(q1);
J12 = 0.43307*cos(q2 + q3)*cos(q1) - 0.4318*cos(q1)*sin(q2) + 0.02032*sin(q2 + q3)*cos(q1);
J13 = 0.43307*cos(q2 + q3)*cos(q1) + 0.02032*sin(q2 + q3)*cos(q1);
J21 = 0.4318*cos(q1)*cos(q2) - 0.14909*sin(q1) - 0.02032*cos(q2 + q3)*cos(q1) + 0.43307*sin(q2 +
    q3)*cos(q1);
J22 = 0.43307*cos(q2 + q3)*sin(q1) - 0.4318*sin(q1)*sin(q2) + 0.02032*sin(q2 + q3)*sin(q1);
J23 = 0.43307*cos(q2 + q3)*sin(q1) + 0.02032*sin(q2 + q3)*sin(q1);
J31 = 0;
J32 = 0.02032*cos(q2 + q3) - 0.43307*sin(q2 + q3) - 0.4318*cos(q2);
J33 = 0.02032*cos(q2 + q3) - 0.43307*sin(q2 + q3);
J = [J11 J12 J13; J21 J22 J23; J31 J32 J33];

%J' = d/dt(J)
J_dot_11 = 0.02032*cos(q2 + q3)*cos(q1)*q1_dot - 0.02032*sin(q2 + q3)*sin(q1)*(q2_dot+q3_dot) +
    0.4318*sin(q2)*sin(q1)*q2_dot - 0.4318*cos(q2)*cos(q1)*q1_dot + 0.14909*sin(q1)*q1_dot -
    0.43307*cos(q2 + q3)*sin(q1)*(q2_dot+q3_dot) - 0.43307*sin(q2 + q3)*cos(q1)*q1_dot;
J_dot_12 = -0.43307*sin(q2 + q3)*cos(q1)*(q2_dot + q3_dot) - 0.43307*cos(q2 + q3)*sin(q1)*q1_dot +
    0.4318*sin(q1)*sin(q2)*q1_dot - 0.4318*cos(q1)*cos(q2)*q2_dot + 0.02032*cos(q2 +
    q3)*cos(q1)*(q2_dot + q3_dot) - 0.02032*sin(q2 + q3)*sin(q1)*q1_dot;
J_dot_13 = -0.43307*sin(q2 + q3)*cos(q1)*(q2_dot + q3_dot) - 0.43307*cos(q2 + q3)*sin(q1)*q1_dot +
    0.02032*cos(q2 + q3)*cos(q1)*(q2_dot + q3_dot) - 0.02032*sin(q2 + q3)*sin(q1)*q1_dot;
J_dot_21 = -0.4318*sin(q1)*cos(q2)*q1_dot - 0.4318*cos(q1)*sin(q2)*q2_dot - 0.14909*cos(q1)*q1_dot
    + 0.02032*sin(q2 + q3)*cos(q1)*(q2_dot + q3_dot) + 0.02032*cos(q2 + q3)*sin(q1)*q1_dot +
    0.43307*cos(q2 + q3)*cos(q1)*(q2_dot + q3_dot) - 0.43307*sin(q2 + q3)*sin(q1)*q1_dot;
J_dot_22 = -0.43307*sin(q2 + q3)*sin(q1)*(q2_dot + q3_dot) + 0.43307*cos(q2 + q3)*cos(q1)*q1_dot -
    0.4318*cos(q1)*sin(q2)*q1_dot - 0.4318*sin(q1)*cos(q2)*q2_dot + 0.02032*cos(q2 +
    q3)*sin(q1)*(q2_dot + q3_dot) + 0.02032*sin(q2 + q3)*cos(q1)*q1_dot;
J_dot_23 = -0.43307*sin(q2 + q3)*sin(q1)*(q2_dot + q3_dot) + 0.43307*cos(q2 + q3)*cos(q1)*q1_dot +
    0.02032*cos(q2 + q3)*sin(q1)*(q2_dot + q3_dot) + 0.02032*sin(q2 + q3)*cos(q1)*q1_dot;
J_dot_31 = 0;
J_dot_32 = -0.02032*sin(q2 + q3)*(q2_dot + q3_dot) - 0.43307*cos(q2 + q3)*(q2_dot + q3_dot) +
    0.4318*sin(q2)*q2_dot;
J_dot_33 = -0.02032*sin(q2 + q3)*(q2_dot + q3_dot) - 0.43307*cos(q2 + q3)*(q2_dot + q3_dot);

J_dot = [J_dot_11 J_dot_12 J_dot_13; J_dot_21 J_dot_22 J_dot_23; J_dot_31 J_dot_32 J_dot_33];


%D(q) matrix
d11 = 2.4574 + 1.7181*cos(q2)*cos(q2) + 0.4430*sin(q2+q3)*sin(q2+q3) - 0.0324*cos(q2)*cos(q2+q3) -
    0.0415*cos(q2+q3)*sin(q2+q3) + 0.9378*cos(q2)*sin(q2+q3);
d12 = 2.2312*sin(q2) - 0.0068*sin(q2+q3) - 0.1634*cos(q2+q3);
```

```
d13 = -0.0068*sin(q2+q3) - 0.1634*cos(q2+q3);
d21 = d12;
d22 = 5.1285 + 0.9378*sin(q3) - 0.0324*cos(q3);
d23 = 0.4424 + 0.4689*sin(q3) - 0.0162*cos(q3);
d31 = d13;
d32 = d23;
d33 = 1.0236;
D = [d11 d12 d13; d21 d22 d23; d31 d32 d33];

%C(q,q') Matrix
c111 = 0;
c121 = 0.0207 - 1.2752*cos(q2)*sin(q2) + 0.4429*cos(q3)*sin(q3) -
    0.8859*sin(q2)*sin(q3)*sin(q2+q3) + 0.0325*cos(q2)*sin(q2+q3) + 0.4689*cos(q2)*cos(q2+q3) -
    0.4689*sin(q2)*sin(q2+q3) - 0.0461*cos(q2+q2) - 0.0415*cos(q2+q3)*cos(q2+q3) - 0.0163*sin(q3);
c131 = 0.0207 + 0.4429*cos(q2)*sin(q2) + 0.4429*cos(q3)*sin(q3) -
    0.8859*sin(q2)*sin(q3)*sin(q2+q3) + 0.0163*cos(q2)*sin(q2+q3) + 0.4689*cos(q2)*cos(q2+q3) -
    0.0415*cos(q2+q3)*cos(q2+q3);
c211 = c121;
c221 = 1.8181*cos(q2) + 0.1634*sin(q2+q3) - 0.0068*cos(q2+q3);
c231 = 0.1634*sin(q2+q3) - 0.0068*cos(q2+q3);
c311 = c131;
c321 = c231;
c331 = 0.1634*sin(q2+q3) - 0.0068*cos(q2+q3);
c112 = - c121;
c122 = 0;
c132 = 0;
c212 = c122;
c222 = 0;
c232 = 0.4689*cos(q3) + 0.0162*sin(q3);
c312 = 0;
c322 = c232;
c332 = 0.4689*cos(q3) + 0.0162*sin(q3);
c113 = - c131;
c123 = - c132;
c133 = 0;
c213 = c123;
c223 = - c232;
c233 = 0;
c313 = c133;
c323 = c233;
c333 = 0;

c11 = c111*q1_dot + c211*q2_dot + c311*q3_dot;
c12 = c121*q1_dot + c221*q2_dot + c321*q3_dot;
c13 = c131*q1_dot + c231*q2_dot + c331*q3_dot;
c21 = c112*q1_dot + c212*q2_dot + c312*q3_dot;
c22 = c122*q1_dot + c222*q2_dot + c322*q3_dot;
c23 = c132*q1_dot + c232*q2_dot + c332*q3_dot;
c31 = c113*q1_dot + c213*q2_dot + c313*q3_dot;
c32 = c123*q1_dot + c223*q2_dot + c323*q3_dot;
c33 = c133*q1_dot + c233*q2_dot + c333*q3_dot;

C = [c11 c12 c13; c21 c22 c23; c31 c32 c33];

%G matrix
g1 = 0;
g2 = - 48.5564*cos(q2) + 1.0462*sin(q2) + 0.3683*cos(q2+q3) - 10.6528*sin(q2+q3);
g3 = 0.3683*cos(q2+q3) - 10.6528*sin(q2+q3);
```

```
G = [g1 ; g2 ; g3];

tau = D*(inv(J))*u - D*(inv(J))*J_dot*q_dot + C*q_dot + G;
```

```
%----------------------------------- dynamics_var.m ----------------------------------------%

% Dynamics Puma 560
% D(q)q" + C(q,q')q' + g(q) = tau%%
% q(t) = x1
% q'(t) = x2
% xdot_1 = x1' = x2
% xdot_2 = x2' = inv(D(x1)){tau - g(x1) - C(x1,x2)x2}
% xdot = [xdot_1 xdot_2]

function xdot = dynamics_var(x,tau,R)

q = [x(1); x(2); x(3)];
x1 = q;
qdot = [x(4); x(5); x(6)];
x2 = qdot;

%D(q) matrix
d11 = 2.4574 + 1.7181*cos(q(2))*cos(q(2)) + 0.4430*sin(q(2)+q(3))*sin(q(2)+q(3)) -
    0.0324*cos(q(2))*cos(q(2)+q(3)) - 0.0415*cos(q(2)+q(3))*sin(q(2)+q(3)) +
    0.9378*cos(q(2))*sin(q(2)+q(3));
d12 = 2.2312*sin(q(2)) - 0.0068*sin(q(2)+q(3)) - 0.1634*cos(q(2)+q(3));
d13 = -0.0068*sin(q(2)+q(3)) - 0.1634*cos(q(2)+q(3));
d21 = d12;
d22 = 5.1285 + 0.9378*sin(q(3)) - 0.0324*cos(q(3));
d23 = 0.4424 + 0.4689*sin(q(3)) - 0.0162*cos(q(3));
d31 = d13;
d32 = d23;
d33 = 1.0236;
D = [R(1,1)*d11 R(1,2)*d12 R(1,3)*d13; R(2,1)*d21 R(2,2)*d22 R(2,3)*d23; R(3,1)*d31 R(3,2)*d32
    R(3,3)*d33];

%C(q,q') Matrix
c111 = 0;
c121 = 0.0207 - 1.2752*cos(q(2))*sin(q(2)) + 0.4429*cos(q(3))*sin(q(3)) -
    0.8859*sin(q(2))*sin(q(3))*sin(q(2)+q(3)) + 0.0325*cos(q(2))*sin(q(2)+q(3)) +
    0.4689*cos(q(2))*cos(q(2)+q(3)) - 0.4689*sin(q(2))*sin(q(2)+q(3)) - 0.0461*cos(q(2)+q(2)) -
    0.0415*cos(q(2)+q(3))*cos(q(2)+q(3)) - 0.0163*sin(q(3));
c131 = 0.0207 + 0.4429*cos(q(2))*sin(q(2)) + 0.4429*cos(q(3))*sin(q(3)) -
    0.8859*sin(q(2))*sin(q(3))*sin(q(2)+q(3)) + 0.0163*cos(q(2))*sin(q(2)+q(3)) +
    0.4689*cos(q(2))*cos(q(2)+q(3)) - 0.0415*cos(q(2)+q(3))*cos(q(2)+q(3));
c211 = c121;
c221 = 1.8181*cos(q(2)) + 0.1634*sin(q(2)+q(3)) - 0.0068*cos(q(2)+q(3));
c231 = 0.1634*sin(q(2)+q(3)) - 0.0068*cos(q(2)+q(3));
c311 = c131;
c321 = c231;
c331 = 0.1634*sin(q(2)+q(3)) - 0.0068*cos(q(2)+q(3));
c112 = - c121;
c122 = 0;
c132 = 0;
c212 = c122;
c222 = 0;
c232 = 0.4689*cos(q(3)) + 0.0162*sin(q(3));
c312 = 0;
```

```
c322 = c232;
c332 = 0.4689*cos(q(3)) + 0.0162*sin(q(3));
c113 = - c131;
c123 = - c132;
c133 = 0;
c213 = c123;
c223 = - c232;
c233 = 0;
c313 = c133;
c323 = c233;
c333 = 0;

c11 = c111*qdot(1) + c211*qdot(2) + c311*qdot(3);
c12 = c121*qdot(1) + c221*qdot(2) + c321*qdot(3);
c13 = c131*qdot(1) + c231*qdot(2) + c331*qdot(3);
c21 = c112*qdot(1) + c212*qdot(2) + c312*qdot(3);
c22 = c122*qdot(1) + c222*qdot(2) + c322*qdot(3);
c23 = c132*qdot(1) + c232*qdot(2) + c332*qdot(3);
c31 = c113*qdot(1) + c213*qdot(2) + c313*qdot(3);
c32 = c123*qdot(1) + c223*qdot(2) + c323*qdot(3);
c33 = c133*qdot(1) + c233*qdot(2) + c333*qdot(3);

C = [R(1,1)*c11 R(1,2)*c12 R(1,3)*c13; R(2,1)*c21 R(2,2)*c22 R(2,3)*c23; R(3,1)*c31 R(3,2)*c32
    R(3,3)*c33];

%G matrix
g1 = 0;
g2 = - 48.5564*cos(q(2)) + 1.0462*sin(q(2)) + 0.3683*cos(q(2)+q(3)) - 10.6528*sin(q(2)+q(3));
g3 = 0.3683*cos(q(2)+q(3)) - 10.6528*sin(q(2)+q(3));

G = [R(1,1)*g1 ; R(1,2)*g2 ; R(1,3)*g3];

xdot_1 = x2;
xdot_2 = (inv(D))*(tau - G - C*x2);
xdot = [xdot_1 ; xdot_2];

end
```

```
%------------------------------------ forward_kinematics.m ------------------------------------%

% Forward Kinematics
% Y = h(q)
% Y' = Jq'

function [Y Y_dot] = forward_kinematics(q,q_dot)

% Robot Params
a = [0 431.8/1000 -20.32/1000 0]; %ai in m
d = [0 149.09/1000 0 433.07/1000]; %di in m

%q(3x1),qdot(3x1)

q1 = q(1);
q2 = q(2);
q3 = q(3);
h1 = a(3)*cos(q1)*cos(q2+q3) + d(4)*cos(q1)*sin(q2+q3) + a(2)*cos(q1)*cos(q2) - d(2)*sin(q1);
h2 = a(3)*sin(q1)*cos(q2+q3) + d(4)*sin(q1)*sin(q2+q3) + a(2)*sin(q1)*cos(q2) + d(2)*cos(q1);
h3 = - a(3)*sin(q2+q3) + d(4)*cos(q2+q3) - a(2)*sin(q2);
```

```
h = [h1; h2; h3];

Y = h; % Y = h(q)
% Jacobian equation obtained from the Jacobian m-script where its calculated symbolically once to
% avoid symbolic recalculation in each loop (idea suggested by Dr. Zhao)
J = double([(127*cos(q2 + q3)*sin(q1))/6250-(2159*cos(q2)*sin(q1))/5000 - (14909*cos(q1))/100000 -
    (43307*sin(q2 + q3)*sin(q1))/100000, (43307*cos(q2 + q3)*cos(q1))/100000 -
    (2159*cos(q1)*sin(q2))/5000 + (127*sin(q2 + q3)*cos(q1))/6250, (43307*cos(q2 +
    q3)*cos(q1))/100000 + (127*sin(q2 + q3)*cos(q1))/6250 ;(2159*cos(q1)*cos(q2))/5000 -
    (14909*sin(q1))/100000 - (127*cos(q2 + q3)*cos(q1))/6250 + (43307*sin(q2 +
    q3)*cos(q1))/100000, (43307*cos(q2 + q3)*sin(q1))/100000 - (2159*sin(q1)*sin(q2))/5000 +
    (127*sin(q2 + q3)*sin(q1))/6250, (43307*cos(q2 + q3)*sin(q1))/100000 + (127*sin(q2 +
    q3)*sin(q1))/6250; 0,(127*cos(q2 + q3))/6250 - (43307*sin(q2 + q3))/100000 -
    (2159*cos(q2))/5000,(127*cos(q2 + q3))/6250 - (43307*sin(q2 + q3))/100000]);

Y_dot = J*q_dot;
```

```
%------------------------------------ main_case_1.m ------------------------------------%
% This is for the case when %omega = pi/4 and Y(0) = [-0.7765 ; 0 ; 0.045]
clear all; clc; close all;

%%Parameters
time_start = 0;
step = 0.05;
total_time = 20;
w = pi/4; %omega case 1 for Y(0) = [-0.7765 ; 0 ; 0.045]
iteration = 1;
Kd = [36;31;38];
Kp = [23;9;25];
err_x = zeros(1,400);
err_y = zeros(1,400);
err_z = zeros(1,400);
err_x_dot = zeros(1,400);
err_y_dot = zeros(1,400);
err_z_dot = zeros(1,400);

%For implementing variance in Dynamics , per=0 means no variance
per = 5; %Percentage variation needed
R = rand(3,3);
for i = 1:3
for j = 1:3
if R(i,j) >= 0.5
R(i,j) = (1 + (per/100));
else
R(i,j) = (1 - (per/100));
end
end
end

%Initial values for first run
q = [0.1932 ; -2.5912 ; 0.6370]; %Calculated using the inverse_kinematics script
q_dot = [0;0;0]; %Since Y'(0) = [0;0;0]
Y_init = [-0.7765 ; 0 ; 0.045];
Y_dot_init = [0;0;0];
[Yd, Yd_dot, Yd_dot_dot] = traj(w,0);
e = Yd - Y_init;
e_dot = Yd_dot - Y_dot_init;
u = Yd_dot_dot + Kp.*e + Kd.*e_dot; % u = Yd" + Kd(e') + Kp(e) and initially Yd' and Yd" = 0 and e
```

```
    = Yd
tau = non_linear_fb(u,q,q_dot);

while iteration*step < total_time

%%Dynamics Block%%
% takes in tau and gives out q and q'
[T,X]=ode45(@(t,x) dynamics_var(x,tau,R),[time_start step],[q,q_dot]);
q = X(length(X),1:3)';
q_dot = X(length(X),4:6)';


%%Forward Kinematics Block%%
%Takes in q,q' from dynamics block and gives out Y and Y'
[Y, Y_dot] = forward_kinematics(q,q_dot);

%%Desired trajectory%%
%Takes in omega and time and gives out Yd and Yd'
[Yd, Yd_dot, Yd_dot_dot] = traj(w,(iteration*step));


%%Error%%
%Calulate the error input to the linear controller
e = Yd - Y;
e_dot = Yd_dot - Y_dot;


%%Linear Controller%%
% u = Yd" + Kd(e') + Kp(e)
%Takes in the error and gives out u
u = Yd_dot_dot + Kd.*e_dot + Kp.*e;

%%Non-Linear feedback%%
%Takes in u from linear controller and gives out tau
tau = non_linear_fb(u,q,q_dot);

iteration = iteration + 1;
err_x(iteration) = e(1);
err_y(iteration) = e(2);
err_z(iteration) = e(3);
err_x_dot(iteration) = e_dot(1);
err_y_dot(iteration) = e_dot(2);
err_z_dot(iteration) = e_dot(3);
end
%Plot of error and error_dot
figure(01)
t=0:0.05:19.95;
A=subplot(3,2,1);
plot(t,err_x)
set( get(A,'YLabel'), 'String', 'error x' )
B=subplot(3,2,3)
plot(t,err_y);
set( get(B,'YLabel'), 'String', 'error y' )
C=subplot(3,2,5)
plot(t,err_z)
set( get(C,'YLabel'), 'String', 'error z' )
set( get(C,'XLabel'), 'String', 'time' )
D=subplot(3,2,2)
plot(t,err_x_dot)
```

```
set( get(D,'YLabel'), 'String', 'rate error x' )
E=subplot(3,2,4)
plot(t,err_y_dot)
set( get(E,'YLabel'), 'String', 'rate error y' )
F=subplot(3,2,6)
plot(t,err_z_dot)
set( get(F,'YLabel'), 'String', 'rate error z' )
set( get(F,'XLabel'), 'String', 'time' )
```

```
%------------------------------------ main_case_2.m ------------------------------------%
clear all; clc; close all;

% This is when %omega = pi/2 for Y(0) = [-0.5m; -0.1m; 0.0m]
%%Parameters
time_start = 0;
step = 0.05;
total_time = 20;
w = pi/2; %omega case 2 for Y(0) = [-0.5m; -0.1m; 0.0m]
iteration = 1;
Kd = [31;25;31];
Kp = [15;21;20];
err_x = zeros(1,400);
err_y = zeros(1,400);
err_z = zeros(1,400);
err_x_dot = zeros(1,400);
err_y_dot = zeros(1,400);
err_z_dot = zeros(1,400);

%For implementing variance in Dynamics, per=0 means no variance
per = 5; %Percentage variation needed
R = rand(3,3);
for i = 1:3
for j = 1:3
if R(i,j) >= 0.5
R(i,j) = (1 + (per/100));
else
R(i,j) = (1 - (per/100));
end
end
end

%Initial values for first run
q = [0.1932 ; -2.5912 ; 0.6370]; %Calculated using the inverse_kinematics script
q_dot = [0;0;0]; %Since Y'(0) = [0;0;0]
Y_init = [ 0.4941 ; -2.1664 ; -0.3267];
Y_dot_init = [0;0;0];
[Yd, Yd_dot, Yd_dot_dot] = traj(w,0);
e = Yd - Y_init;
e_dot = Yd_dot - Y_dot_init;
u = Yd_dot_dot + Kp.*e + Kd.*e_dot; % u = Yd" + Kd(e') + Kp(e) and initially Yd' and Yd" = 0 and e
    = Yd
tau = non_linear_fb(u,q,q_dot);

while iteration*step < total_time

%%Dynamics Block%%
% takes in tau and gives out q and q'
[T,X]=ode45(@(t,x) dynamics_var(x,tau,R),[time_start step],[q,q_dot]);
```

```
q = X(length(X),1:3)';
q_dot = X(length(X),4:6)';


%%Forward Kinematics Block%%
%Takes in q,q' from dynamics block and gives out Y and Y'
[Y, Y_dot] = forward_kinematics(q,q_dot);

%%Desired trajectory%%
%Takes in omega and time and gives out Yd and Yd'
[Yd, Yd_dot, Yd_dot_dot] = traj(w,(iteration*step));


%%Error%%
%Calulate the error input to the linear controller
e = Yd - Y;
e_dot = Yd_dot - Y_dot;


%%Linear Controller%%
% u = Yd" + Kd(e') + Kp(e)
%Takes in the error and gives out u
u = Yd_dot_dot + Kd.*e_dot + Kp.*e;

%%Non-Linear feedback%%
%Takes in u from linear controller and gives out tau
tau = non_linear_fb(u,q,q_dot);

iteration = iteration + 1;
err_x(iteration) = e(1);
err_y(iteration) = e(2);
err_z(iteration) = e(3);
err_x_dot(iteration) = e_dot(1);
err_y_dot(iteration) = e_dot(2);
err_z_dot(iteration) = e_dot(3);
end

%Plot of error and error_dot
figure(01)
t=0:0.05:19.95;
A=subplot(3,2,1);
plot(t,err_x)
set( get(A,'YLabel'), 'String', 'error x' )
B=subplot(3,2,3)
plot(t,err_y);
set( get(B,'YLabel'), 'String', 'error y' )
C=subplot(3,2,5)
plot(t,err_z)
set( get(C,'YLabel'), 'String', 'error z' )
set( get(C,'XLabel'), 'String', 'time' )
D=subplot(3,2,2)
plot(t,err_x_dot)
set( get(D,'YLabel'), 'String', 'rate error x' )
E=subplot(3,2,4)
plot(t,err_y_dot)
set( get(E,'YLabel'), 'String', 'rate error y' )
F=subplot(3,2,6)
plot(t,err_z_dot)
set( get(F,'YLabel'), 'String', 'rate error z' )
```

```
set( get(F,'XLabel'), 'String', 'time' )
```