5/13/2017

# Project Report

## MECH 564: Fundamentals of Robot Mechanics and Controls

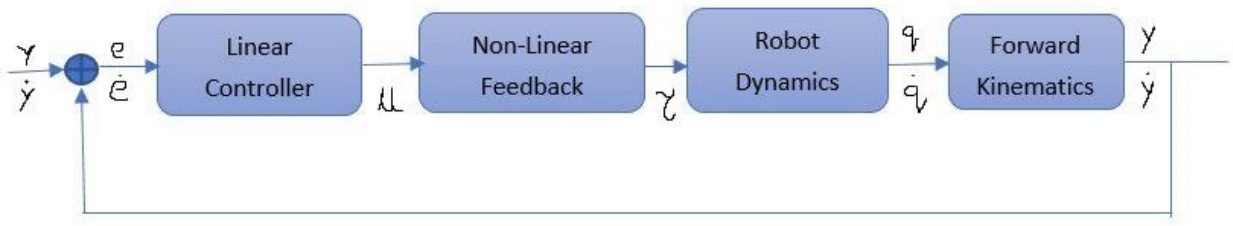
Satyanarayana,Shashank
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
COLORADO STATE UNIVERSITY

# TABLE OF CONTENTS

# 1. Objective:

Evaluate the task level robot control method for the first three joints of PUMA 560 robot arm by computer simulations.

# 2. Block Diagram:




# 3. Description:

This project aims to develop a MATLAB simulation for the first 3 joints (RRR) of a PUMA 560 robot, with 3 degrees of freedom, by implementing the mathematical model derived from the Euler-Lagrange equation.

$$L=K-P$$

Where; K refers to Kinetic Energy of the system

P refers to Potential Energy of the system

## 3.1. Robot Dynamics, Linearization, and Forward Kinematics

As given in *"Robot Dynamics and Control"* Mark W. Spong, Chapter 11, The general dynamic equation for a n-link robot is

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

The principle of inverse dynamics is to seek a nonlinear feedback control law

$$u = f(q, \dot{q}, t)$$

which, when substituted into the general dynamic equation, results in a linear closed loop system.

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

This equation is known as the double integrator system as it represents n uncoupled double integrators. The nonlinear control law is called the inverse dynamics control and achieves a rather remarkable result, namely that the "new" system is linear, and decoupled. This means that each input $\ddot{q_k}$ can be designed to control a scalar linear system. Moreover, if $\ddot{q_k}$ is a function only of $q_k$ and its derivatives, then $\ddot{q_k}$ will affect $q_k$ independently of the motion of the other links.
From the above equation, it can derived that torque and acceleration are related as:

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

The non-linear feedback for a Task level control is given by

$$\tau = D(q)J_h(q)^{-1}\left(u - \dot{J_h}(q)\dot{q}\right) + C(q,\dot{q})\dot{q} + g(q)$$

where $J_h$ is the Jacobian
The implementation of this control scheme requires the computation at each sample instant of the inertia matrix D(q) and the vector of Coriolis, centrifugal, and gravitational. Unlike the computation of torque scheme, however, the inverse dynamics must be computed on-line. In other words, as a feedback control law, it cannot be precomputed off-line and stored as can the computed torque. An important issue therefore in the control system implementation is the design of the computer architecture for the above computations. As processing power continues to increase the computational issues of real-time implementation become less important. An attractive method to implement this scheme is to use a dedicated hardware interface, such as a DSP chip, to perform the required computations in real time.

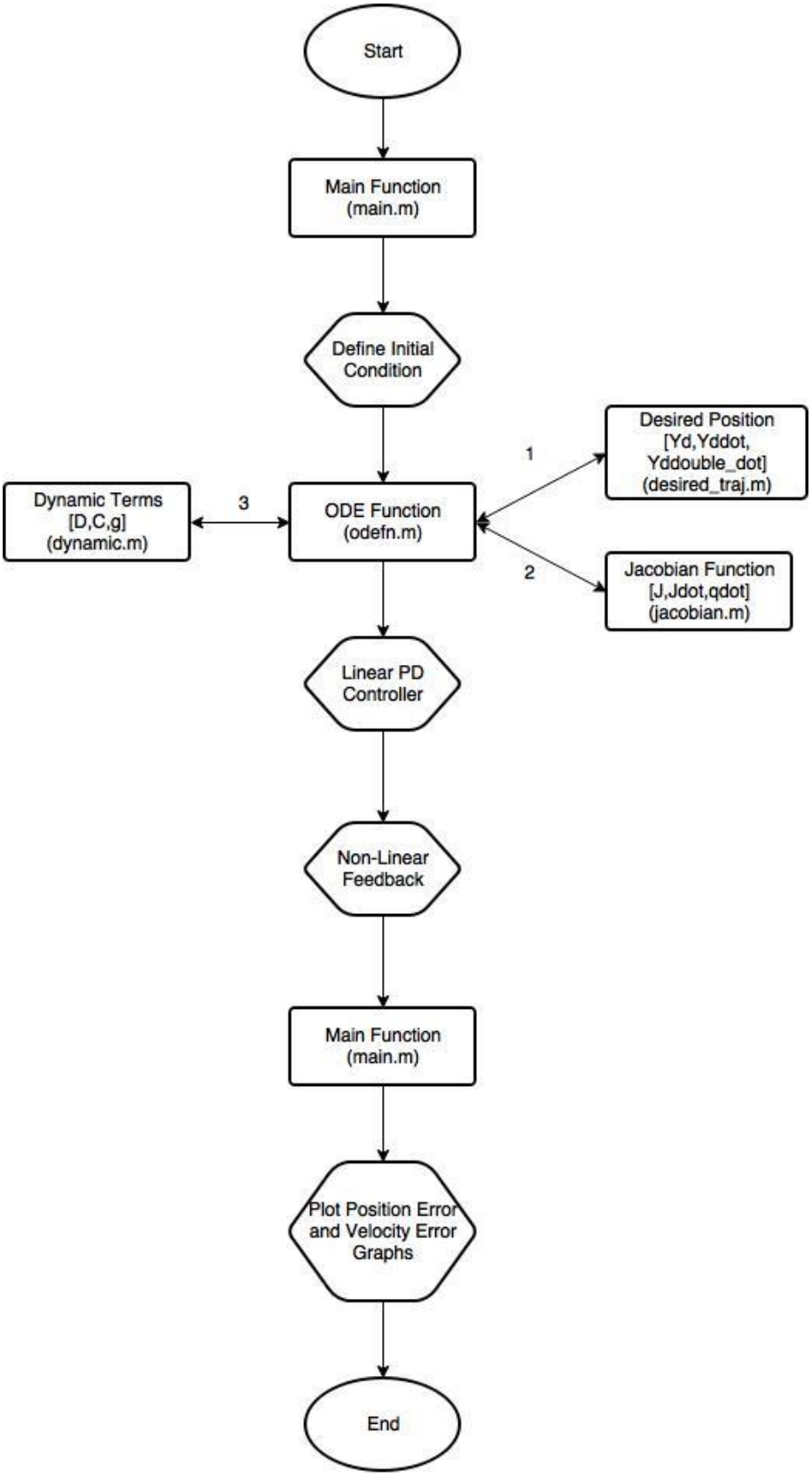
## 3.2. Linear PD (Proportional Differential) Controller:

The equation defines a linear independent joint PD-controller:

$$u = \ddot{Y_d} + k_d(\dot{Y_d} - \dot{Y}) + k_p(Y_d - Y)$$

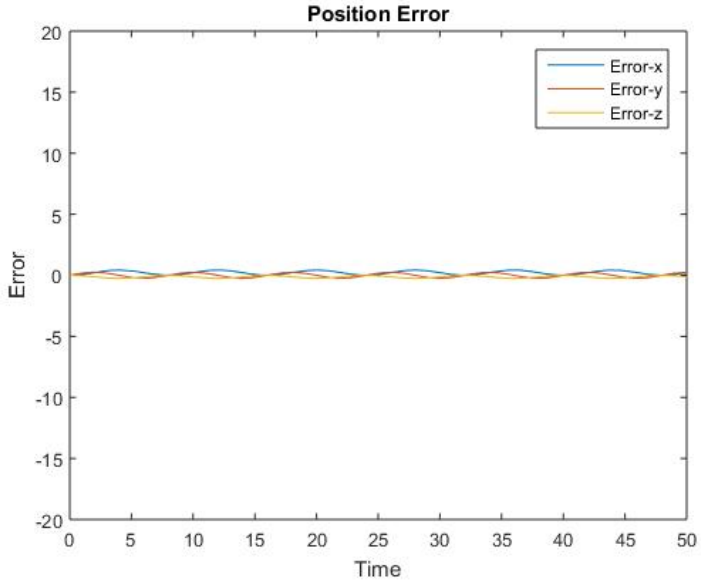
Where $Y_d$ is the desired joint displacement and Y is the actual joint displacement. $K_p$, $K_d$ are diagonal matrices of (positive) proportional and derivative gains, respectively. In the absence of gravity, the PD control law achieves asymptotic tracking of the desired joint positions. This, in effect, reproduces the result derived previously, but is more rigorous, in the sense that the nonlinear equations of motion are not approximated by a constant disturbance.

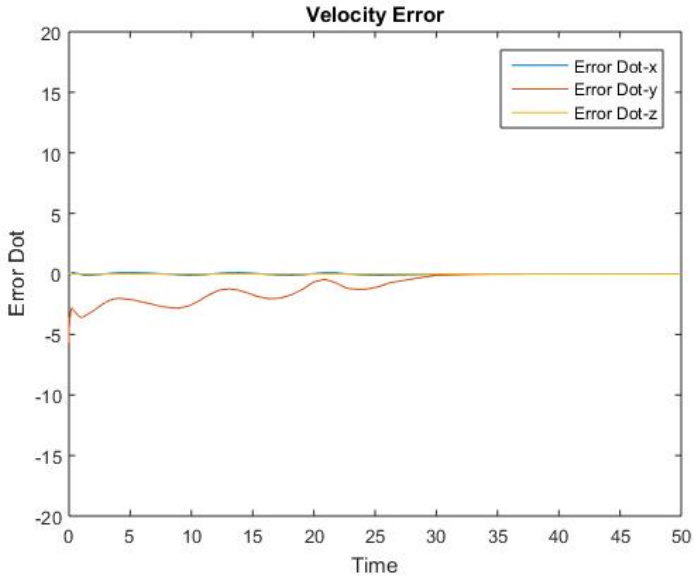
# 4. Algorithm / Flowchart



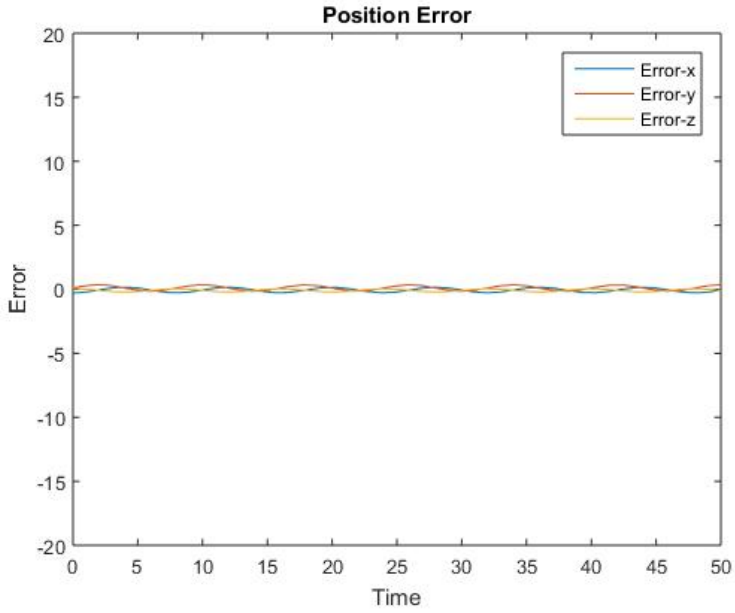
# 5. Results/Plots

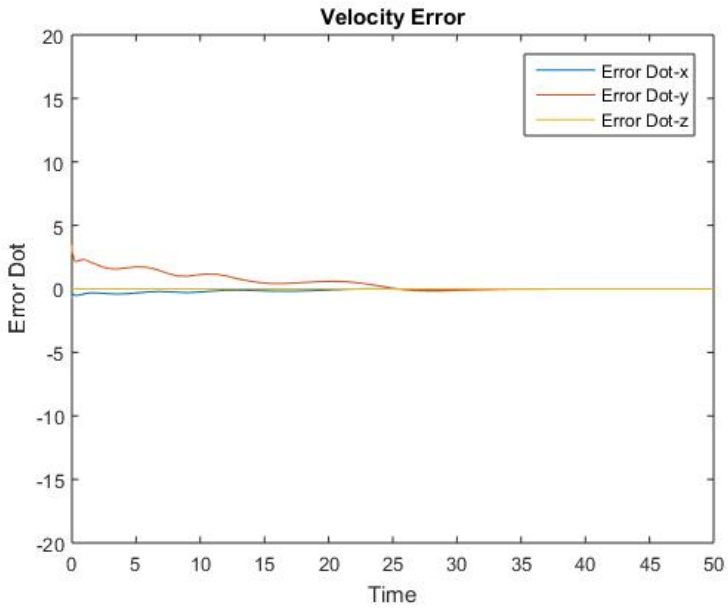
**Case 1:** ω=π/4 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 0%



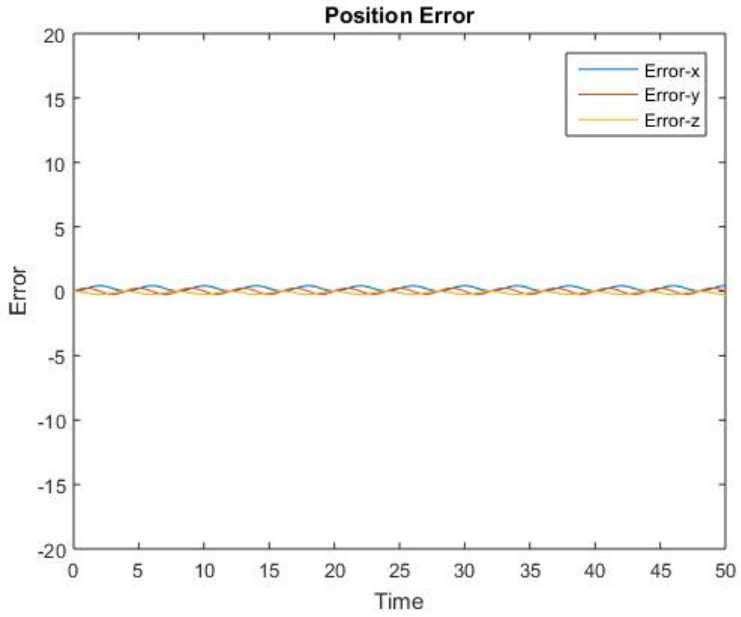



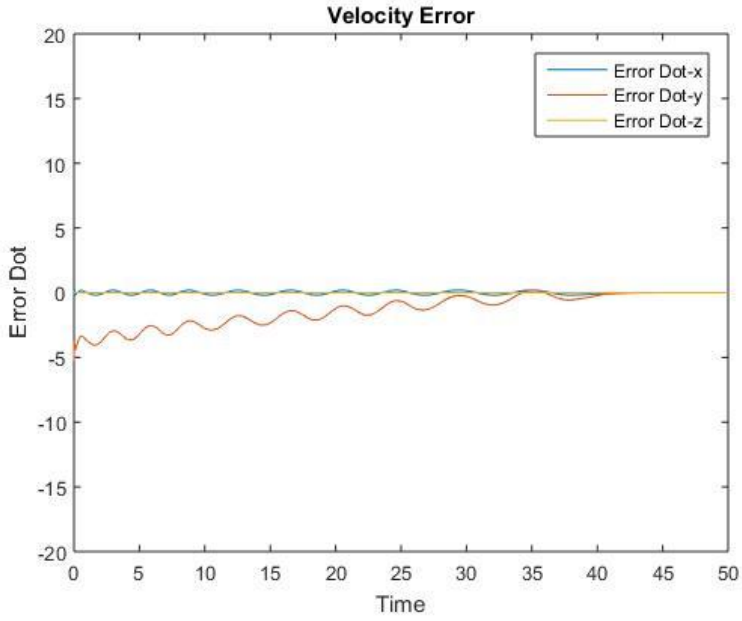
**Case 2:** ω=π/4 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 0%



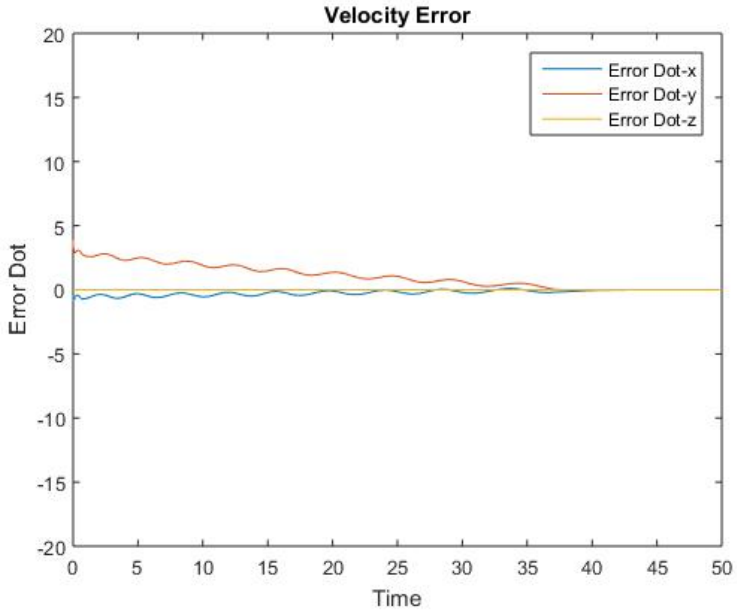



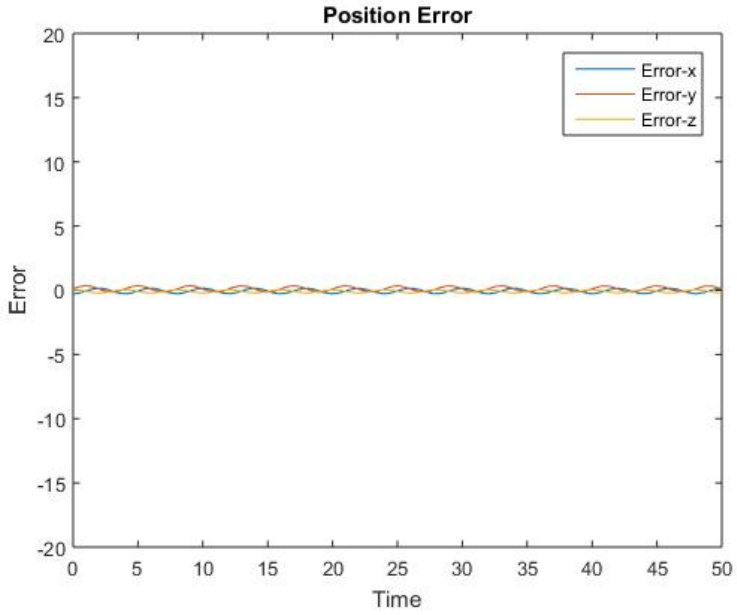
**Case 3:** ω=π/2 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 0%



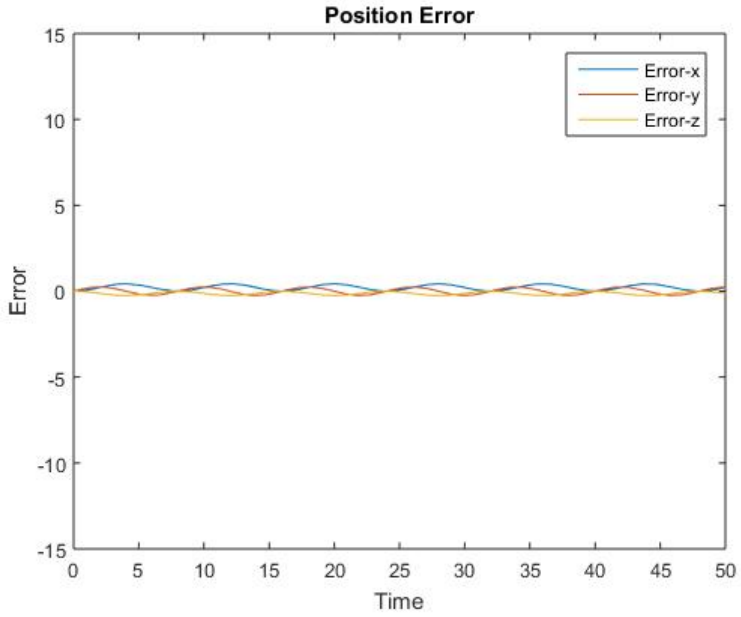



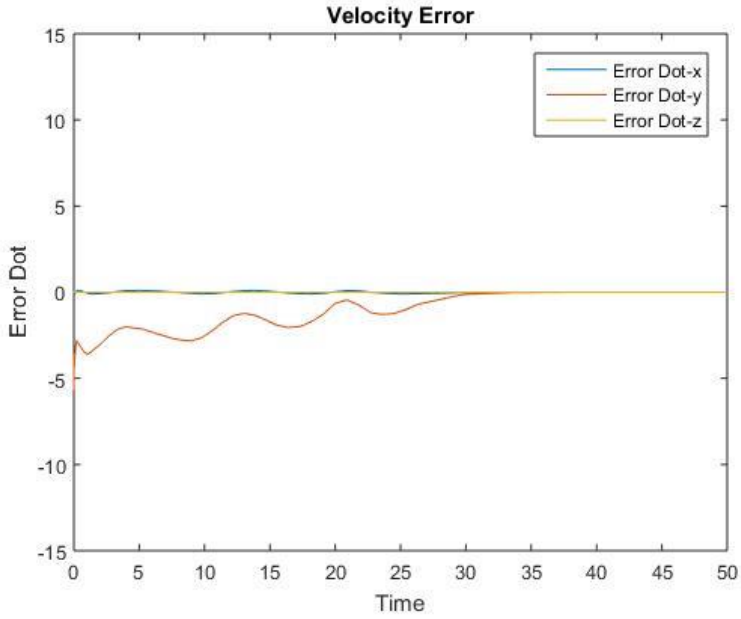
**Case 4:** ω=π/2 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 0%



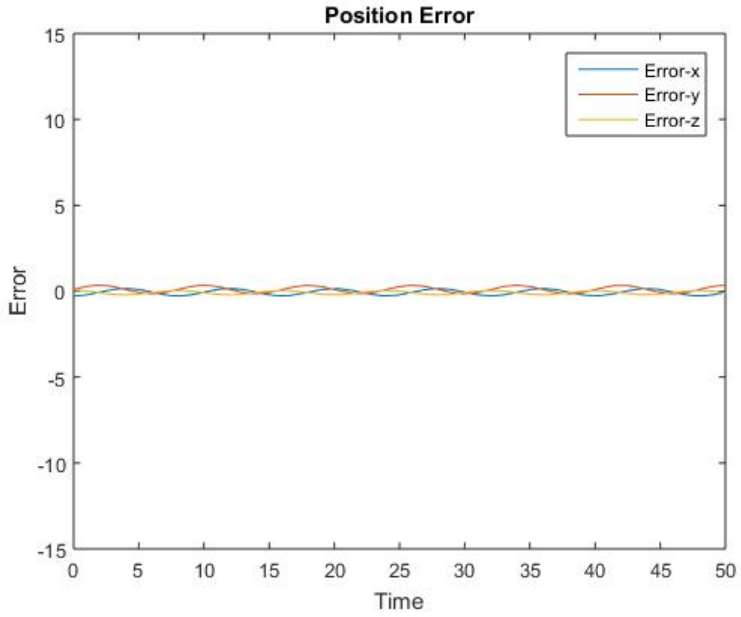



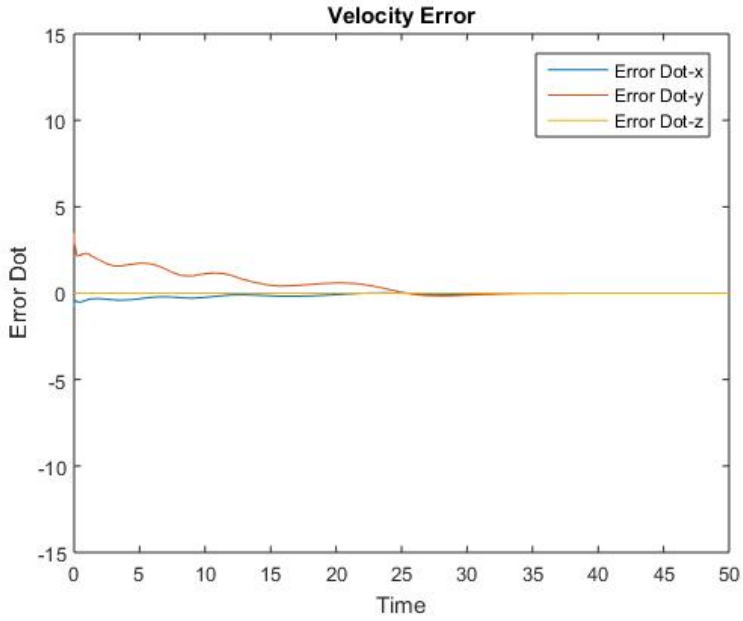
**Case 5:** ω=π/4 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 2%



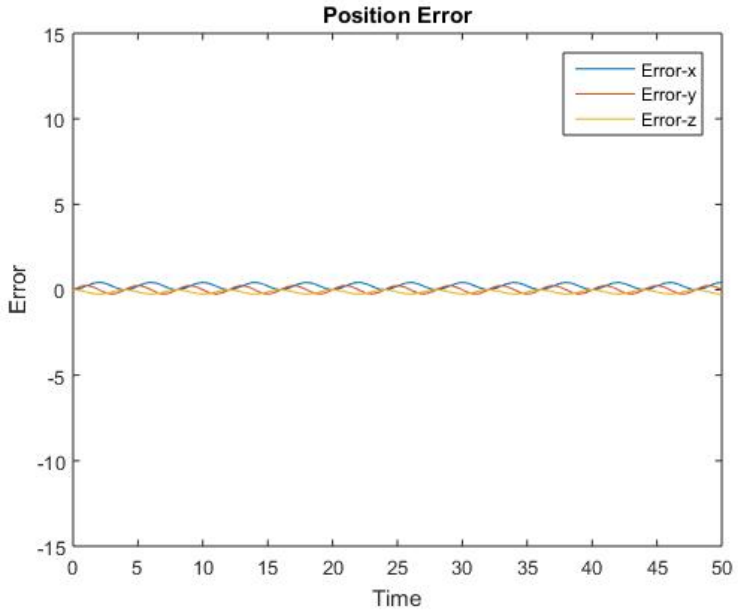



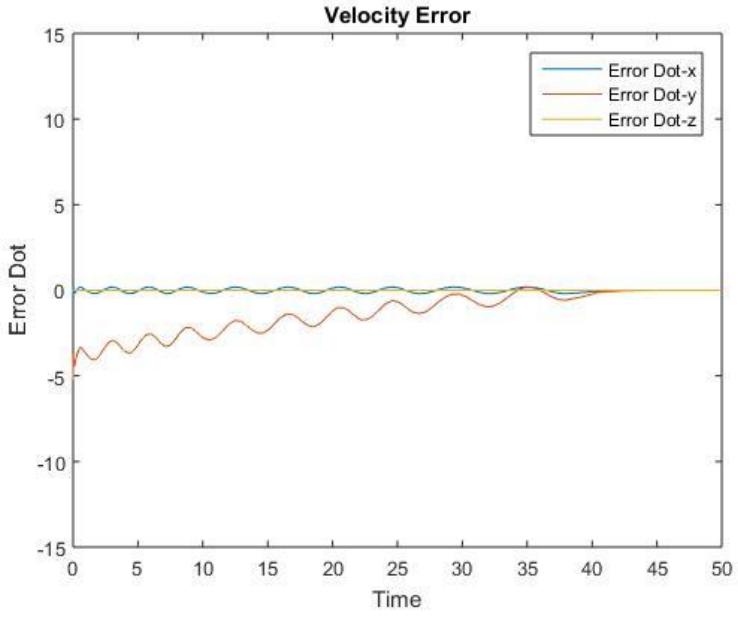
**Case 6:** ω=π/4 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 2%



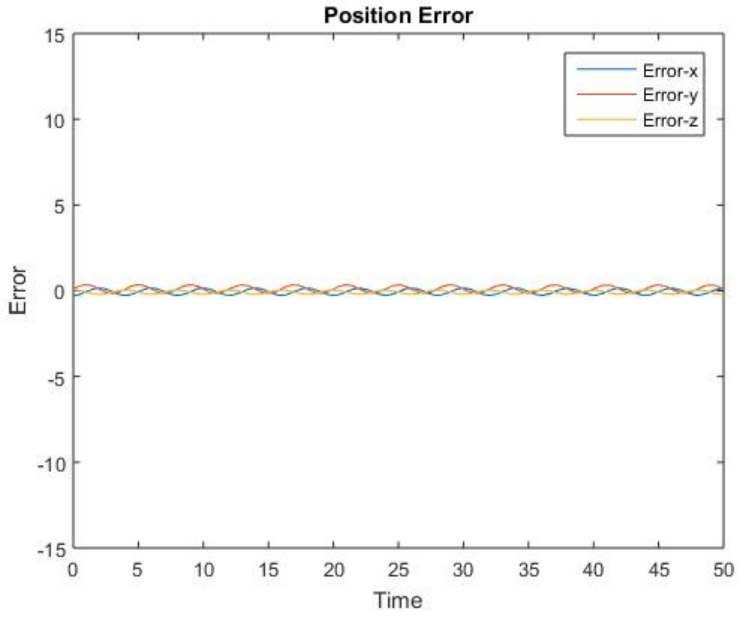



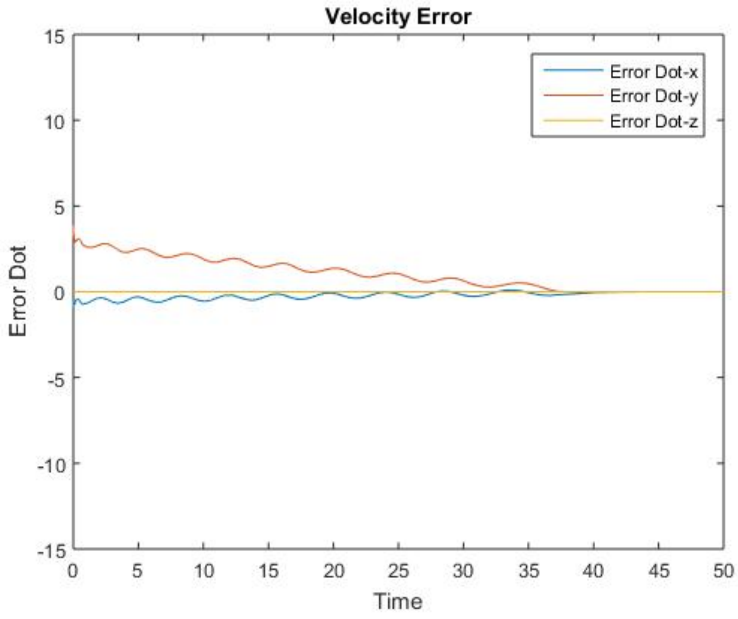
**Case 7:** ω=π/2 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 2%



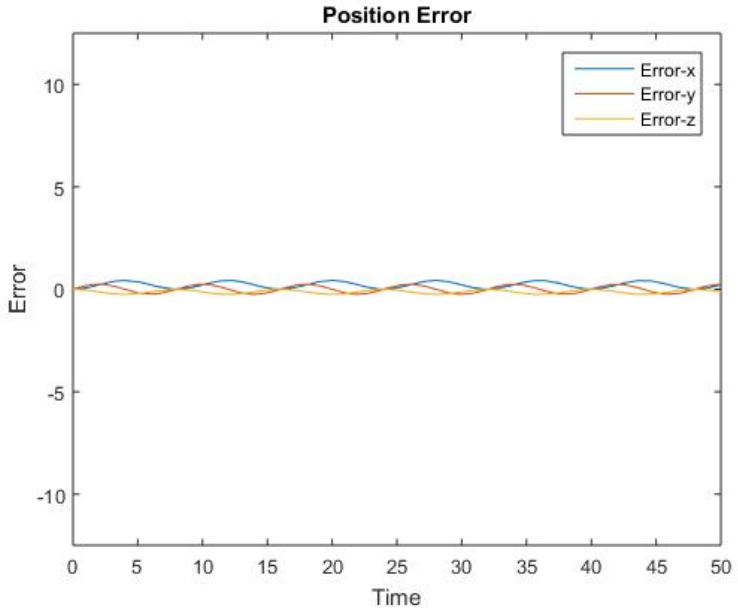



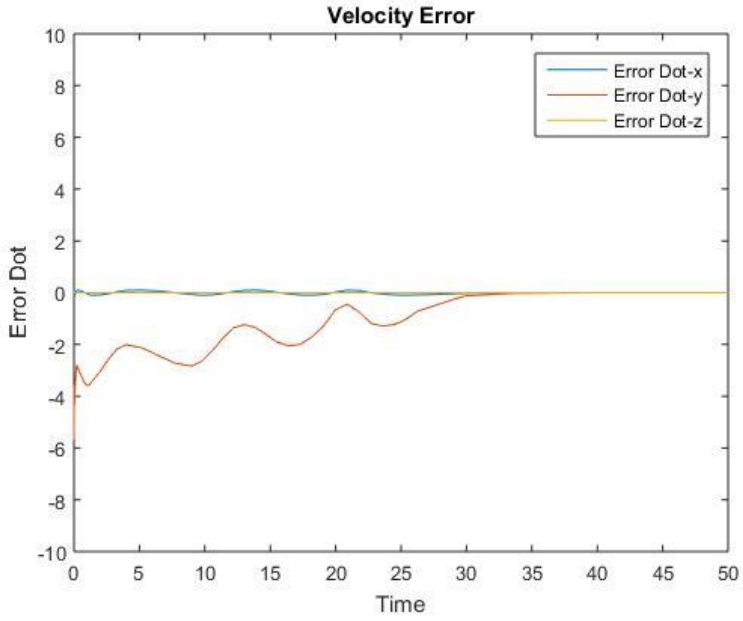
**Case 8:** ω=π/2 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 2%



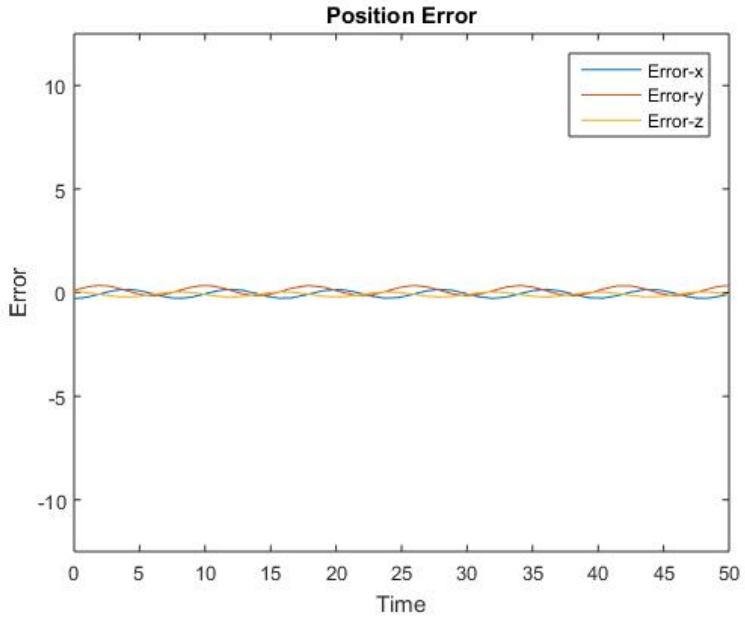



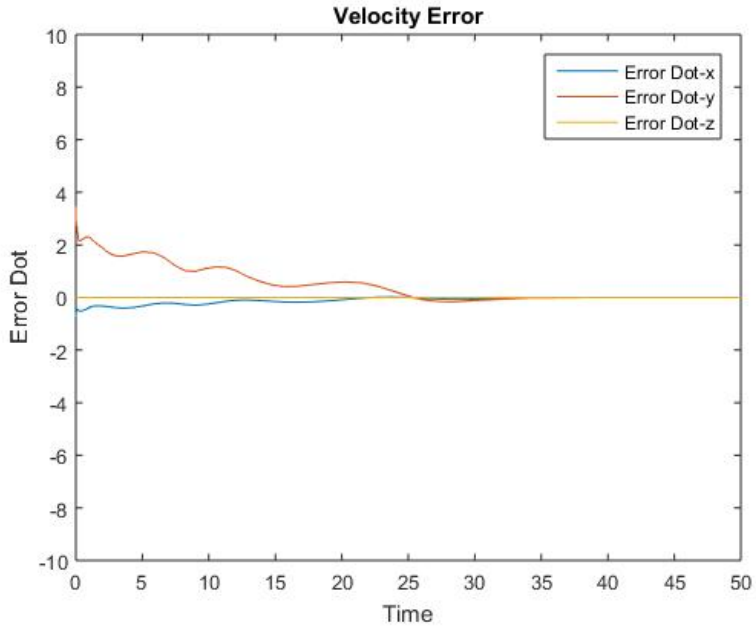
**Case 9:** ω=π/4 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 5%



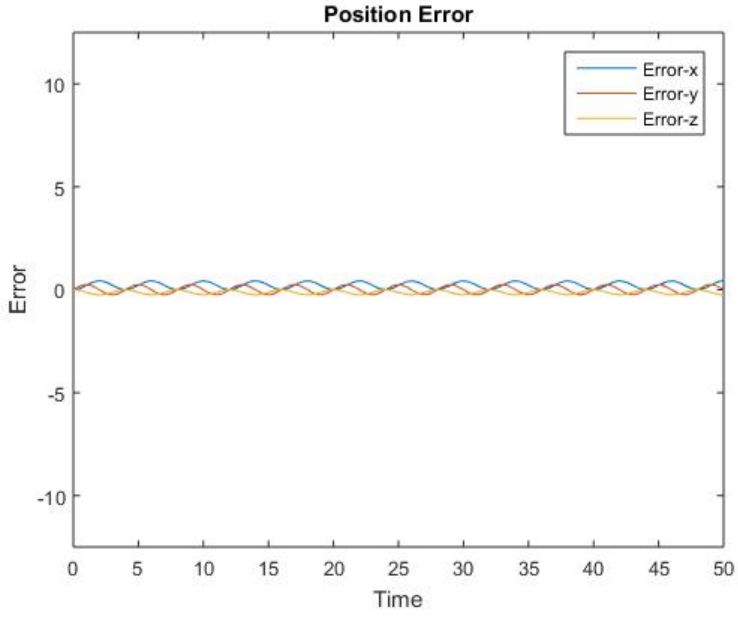



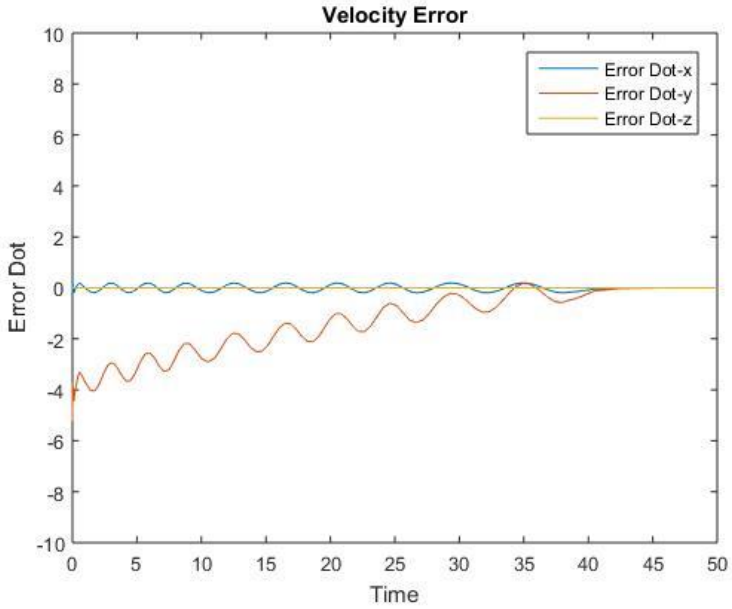
**Case 10:** ω=π/4 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 5%



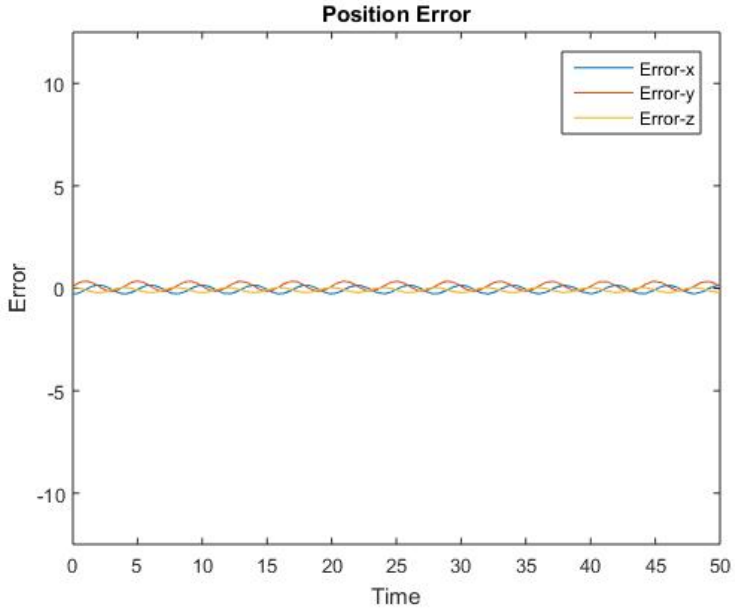



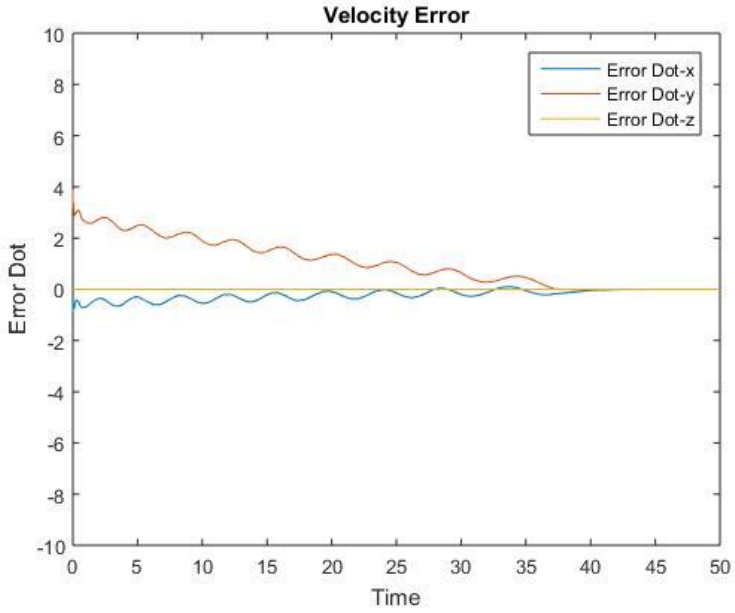
**Case 11:** ω=π/2 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 5%



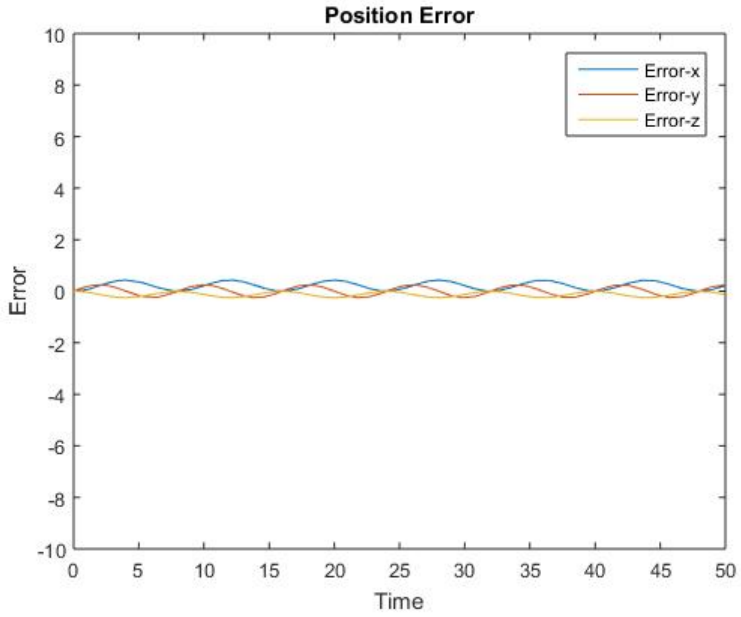



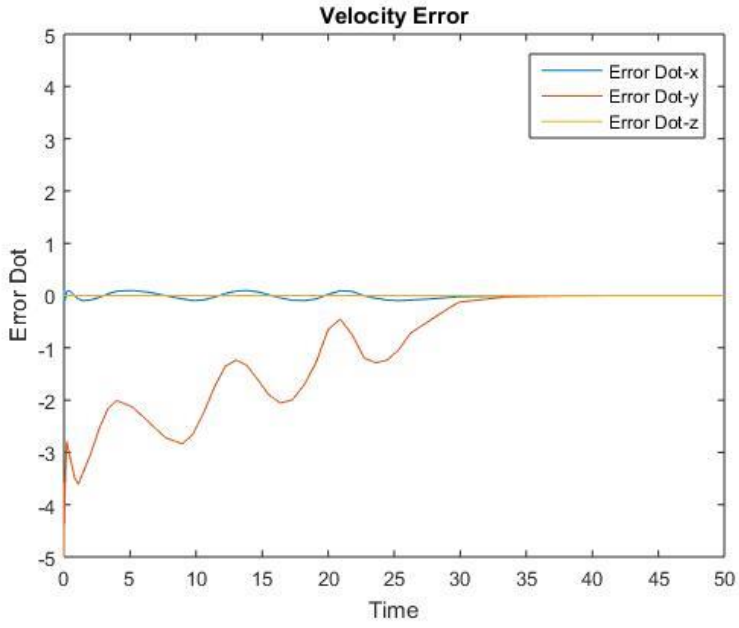
**Case 12:** ω=π/2 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 5%



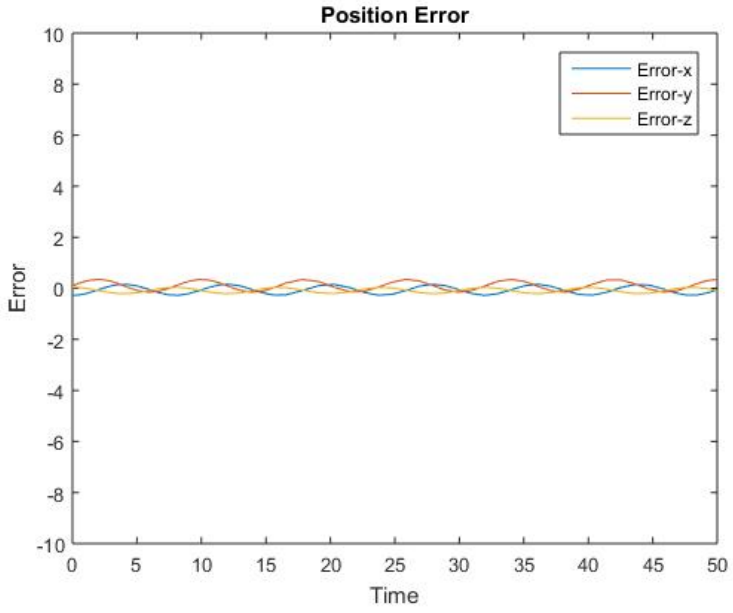



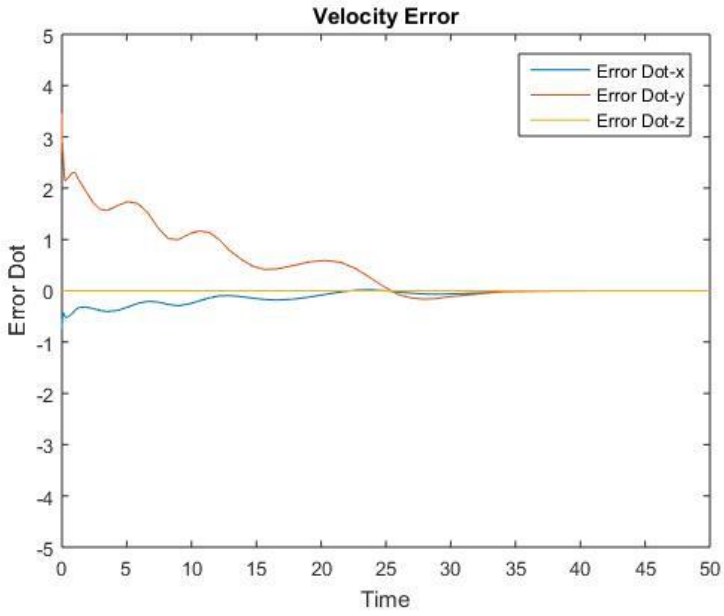
**Case 13:** ω=π/4 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 10%



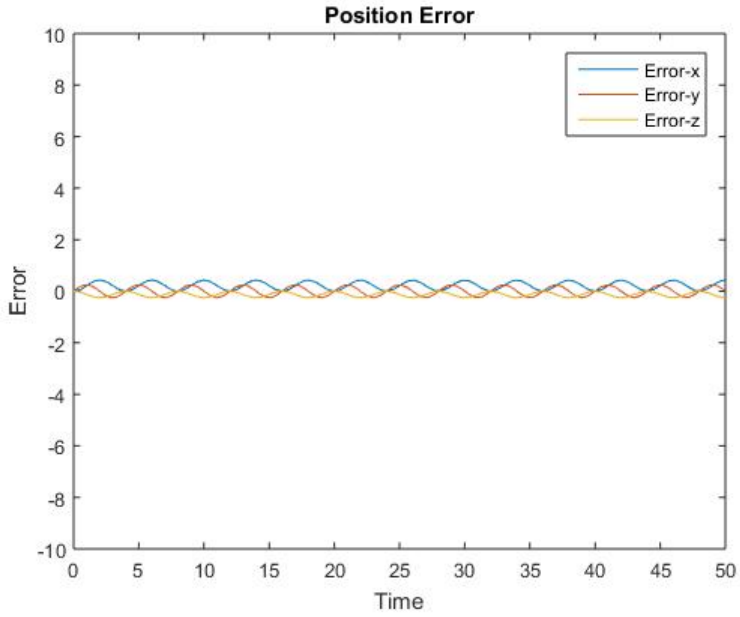



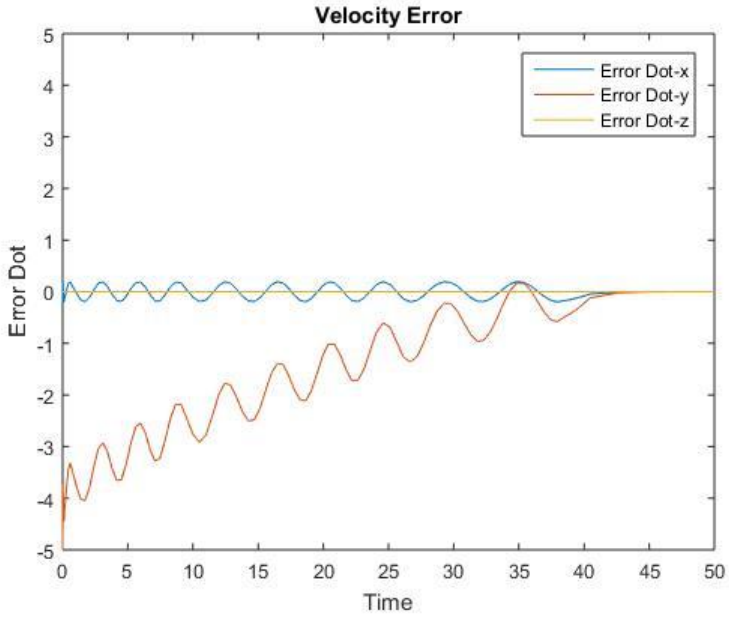
**Case 14:** ω=π/4 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 10%



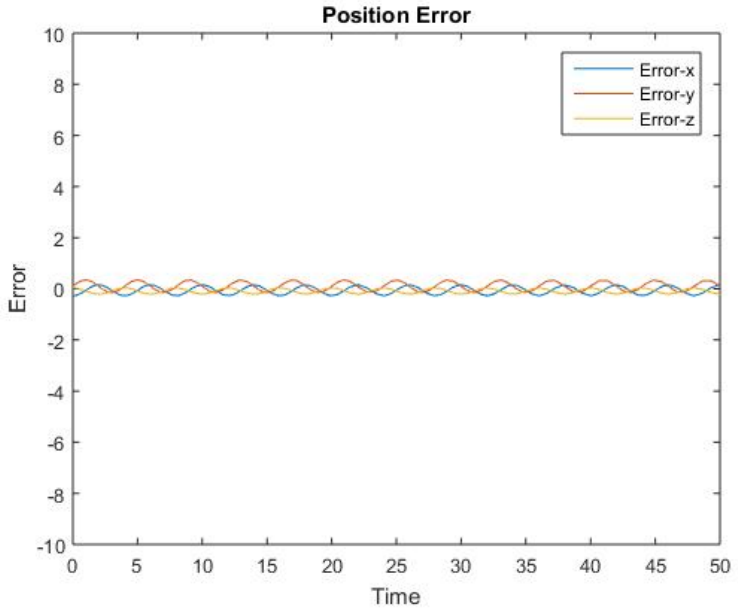



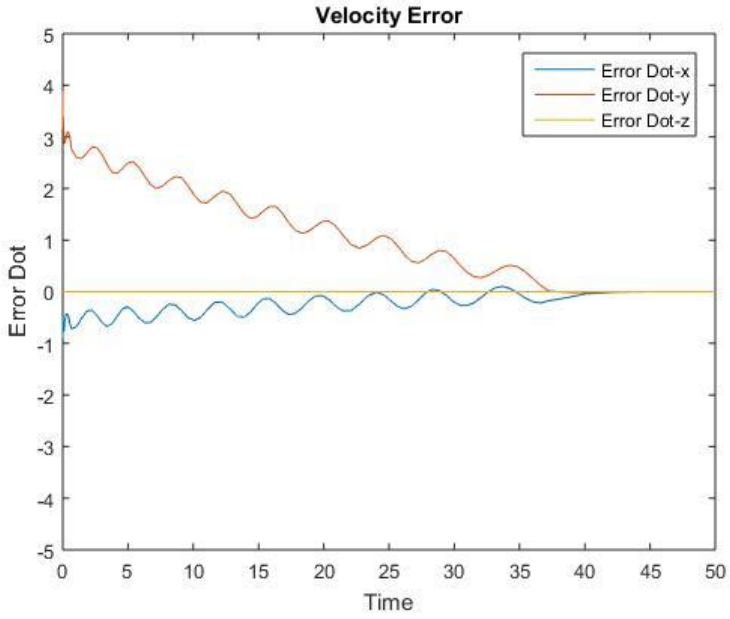
**Case 15:** ω=π/2 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 10%



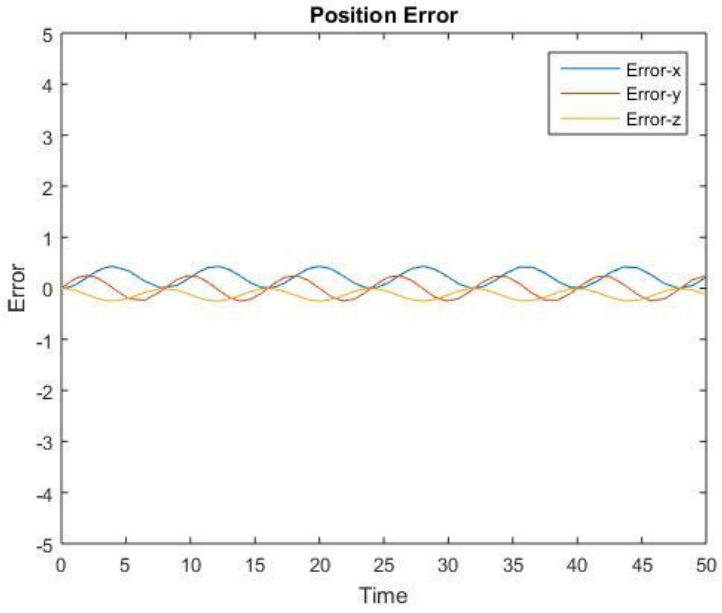



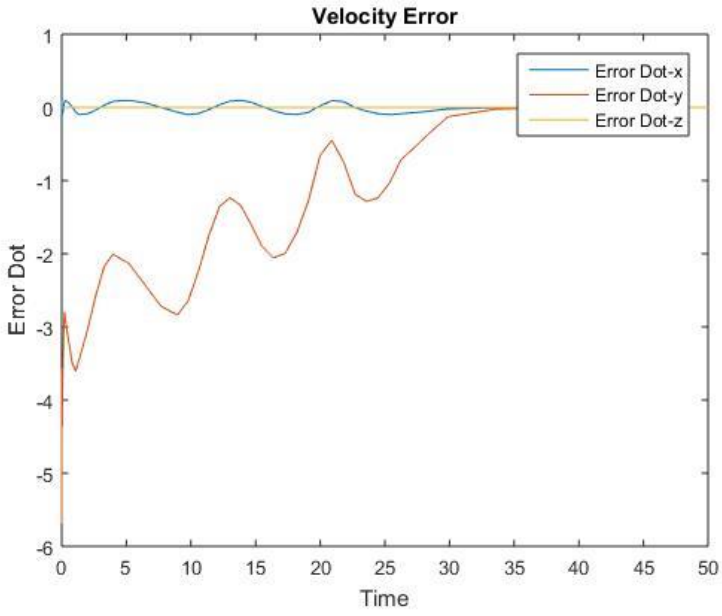
**Case 16:** ω=π/2 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 10%



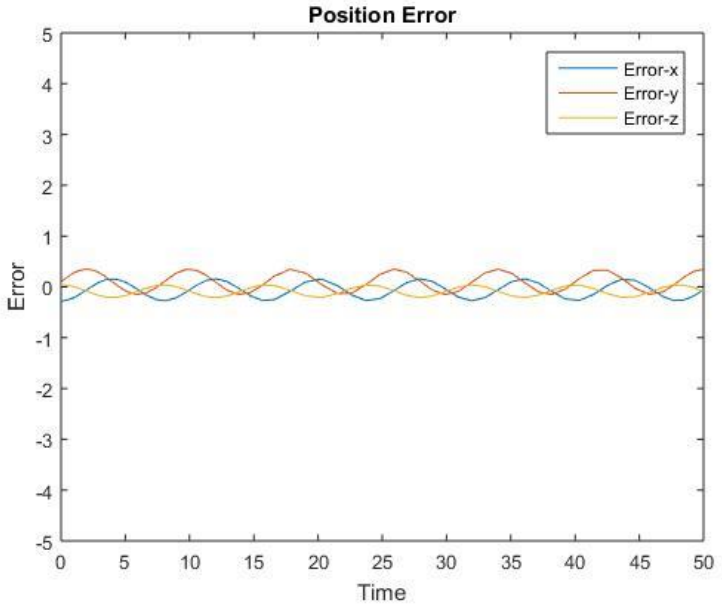



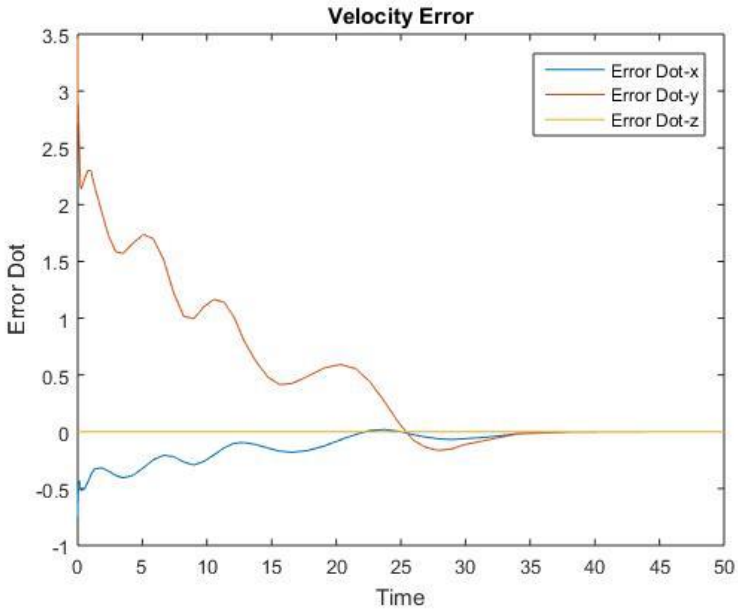
**Case 17:** ω=π/4 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 30%



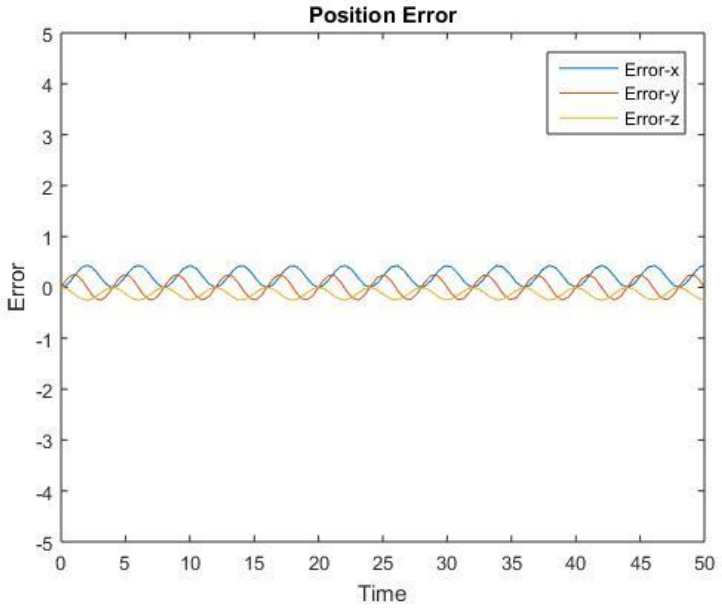



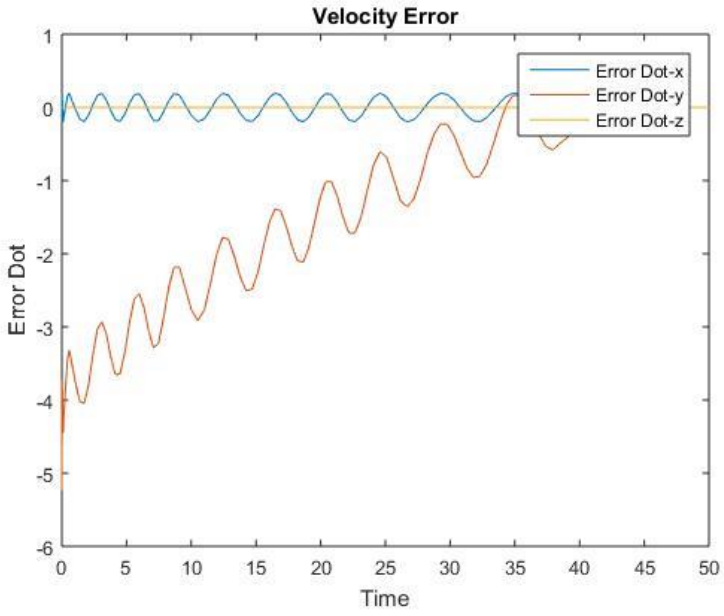
**Case 18:** ω=π/2 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 30%



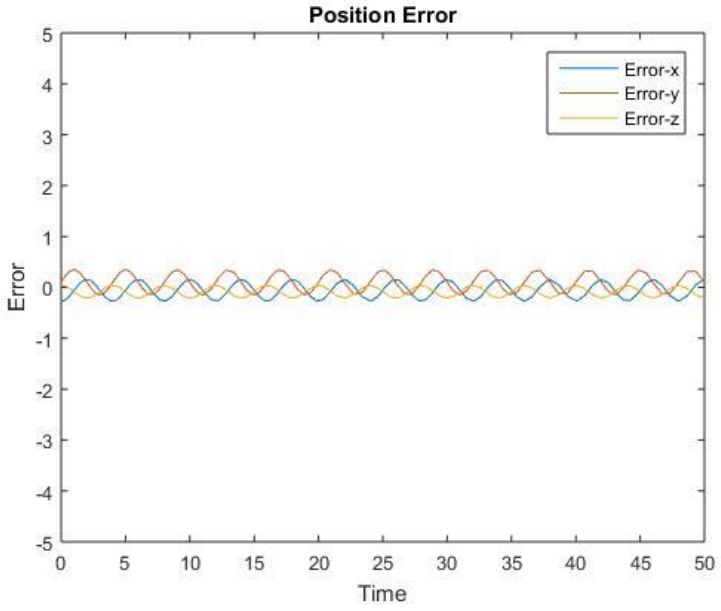



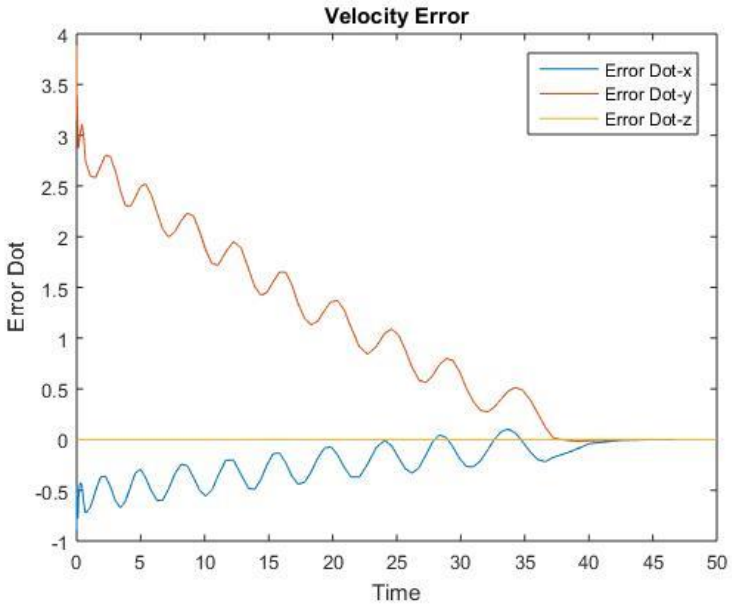
**Case 19:** ω=π/2 ; Y_int = [-0.7765 0 0.045] ; Parametric Variation = 30%



**Case 20:** ω=π/2 ; Y_int = [-0.5 -0.1 0] ; Parametric Variation = 30%



# 6. Code

## 6.1. Main Function

```
%---------------This is the Main Function---------%
%---------------Intialisation----------------------%
w=(pi/2); %Select Value of omega
if w==(pi/4)
    Y_int=[-0.7765 0 0.045 0 0 0];
    qint=[0.193 -2.59 0.637 0 0 0];
%     Y_int=[-0.5 -0.1 0 0 0 0];
%     qint=[0.49 -2.16 -0.32 0 0 0];
elseif w==(pi/2)
    Y_int=[-0.7765 0 0.045 0 0 0];
    qint=[0.193 -2.59 0.637 0 0 0];
%     Y_int=[-0.5 -0.1 0 0 0 0];
%     qint=[0.49 -2.16 -0.32 0 0 0];
end

tspan=[0 50];

%---------------Perform ODE----------------------%
[t,y]=ode45(@(t,y) odefn(t,y,Y_int,w),tspan,qint);

%---------------To Plot Error--------------------------%
q=y(:,1:3);
qdot=y(:,4:6);
[a,b]=size(q);
Y=zeros(a,b);
Yd=zeros(a,b);
for i=1:a
    q1=q(i,1);
    q2=q(i,2);
    q3=q(i,3);

    Y(i,1)=-0.02032*cos(q1)*cos(q2+q3) + 0.43307*cos(q1)*sin(q2+q3) +
0.4318*cos(q1)*cos(q2) - 0.14909*sin(q1);
    Y(i,2)=-0.02032*sin(q1)*cos(q2+q3) + 0.43307*sin(q1)*sin(q2+q3) +
0.4318*sin(q1)*cos(q2) + 0.14909*cos(q1);
    Y(i,3)=--0.02032*sin(q2+q3) + 0.43307*cos(q2+q3) - 0.4318*sin(q2);

    Yd(i,1)=-0.2165*cos(w*t(i))-0.56;
    Yd(i,2)=0.25*sin(w*t(i));
    Yd(i,3)=0.125*cos(w*t(i))-0.08;
end
error=Yd-Y;
figure(1)
plot(t,error)
axis([0 50 -5 5])
title('Position Error')
xlabel('Time')
ylabel('Error')
legend('Error-x','Error-y','Error-z')
```

```
%-------------Error dot plot--------------%
Ydot=zeros(a,b);
Yddot=zeros(a,b);
for i=1:a
[J]=jacobian_plot(q(i,:));
Ydot(i,:)=(J*qdot(i,:).').';

Yddot(i,1)= 0.2165*sin(w*t(i))*w;
Yddot(i,1)= 0.25*cos(w*t(i))*w;
Yddot(i,1)= -0.125*sin(w*t(i))*w;
end
error_dot=Yddot-Ydot;
e_dot=zeros(a,b);
figure(2)
plot(t,e_dot)
axis([0 50 -5 5])
title('Velocity Error')
xlabel('Time')
ylabel('Error Dot')
legend('Error Dot-x','Error Dot-y','Error Dot-z')
```

## 6.2. ODE Function

```
function dydt = odefn(t,y,Y_int,w)
dydt=zeros(6,1)
%Desired Position and Velocity
[Yd,Yddot,Yddot_double]=desired_traj(w,t);
%To Compute Jacobian
[J,Jdot,qdot]=jacobian(y(1:3),Y_int(4:6));
y(4:6)=qdot.';
%To Compute Dynamics
[D,C,g] = dynamic(y(1:3),qdot);
% Parametric Error
error_per=30/100*eye(3);
D=error_per*D;
C=error_per*C;
g=error_per*g;
%% Linear PD controller
e=Yd-Y_int(1:3).';
edot=Yddot-Y_int(4:6).';
kd=[3.5 3.0 4.0]*eye(3);
kp=[2.0 1.5 2.5]*eye(3);
u=Yddot_double+(kd*edot)+(kp*e);
%% Non-linear Feedback
Tou=(D*pinv(J)*u)-(D*pinv(J)*Jdot*qdot)+(C*qdot)+g;
%% Sysyem Dynamics
dydt(1)=y(4);%theta1-dot
dydt(2)=y(5);%theta2-dot
dydt(3)=y(6);%theta3-dot
qdot_double=pinv(D)*(Tou-(C*qdot)-g);
dydt(4)=qdot_double(1);
end
```

## 6.3. Linear Dynamic Function

```
function [D,C,g]=dynamic(y,ydot)
%-------------Initialising q values----------%
y1=y(1);
y2=y(2);
y3=y(3);
y4=ydot(1);
y5=ydot(2);
y6=ydot(3);

%-------------Parameter of D(q)--------------%
D=zeros(3,3);
D(1,1)=2.4574+1.7181*cos(y2)*cos(y2)+0.4430*sin(y2+y3)*sin(y2+y3)-
0.0324*cos(y2)*cos(y2+y3)-
0.0415*cos(y2+y3)*sin(y2+y3)+0.9378*cos(y2)*sin(y2+y3);
D(1,2)=2.2312*sin(y2)+0.0068*sin(y2+y3)-0.1634*cos(y2+y3);
D(1,3)=-0.0068*sin(y2+y3)-0.1634*cos(y2+y3);

D(2,1)=D(1,2);
D(2,2)=5.1285+0.9378*sin(y3)-0.0324*cos(y3);
D(2,3)=0.4424+0.4689*sin(y3)-0.0162*cos(y3);

D(3,1)=D(1,3);
D(3,2)=D(2,3);
D(3,3)=1.0236;

%------------Parameter of C(q)---------------%
C=zeros(3,3);
c111=0;
c121=0.0207-1.2752*cos(y2)*sin(y2)+0.4429*cos(y3)*sin(y3)-
0.8859*sin(y2)*sin(y3)*sin(y2+y3)+0.0325*cos(y2)*sin(y2+y3) +
0.4689*cos(y2)*cos(y2+y3)-0.4689*sin(y2)*sin(y2+y3)-0.0461*cos(y2+y2)-
0.0415*cos(y2+y3)*cos(y2+y3)-0.0163*sin(y3);
c131=0.0207 + 0.4429*cos(y2)*sin(y2) + 0.4429*cos(y3)*sin(y3)-
0.8859*sin(y2)*sin(y3)*sin(y2+y3)+0.0163*cos(y2)*sin(y2+y3) +
0.4689*cos(y2)*cos(y2+y3)-0.0415*cos(y2+y3)*cos(y2+y3);
c211=c121;
c221=1.8181*cos(y2)+0.1634*sin(y2+y3)-0.0068*cos(y2+y3);
c231=0.1634*sin(y2+y3)-0.0068*cos(y2+y3);
c311=c131;
c321=c231;
c331=0.1634*sin(y2+y3)-0.0068*cos(y2+y3);

c112=-c121;
c122=0;
c132=0;
c212=c122;
c222=0;
c232=0.4689*cos(y3)+0.0162*sin(y3);
c312=0;
c322=c232;
c332=0.4689*cos(y3)+0.0162*sin(y3);

c113=-c131;
```

```
c123=-c132;
c133=0;
c213=c123;
c223=-c232;
c233=0;
c313=c133;
c323=c233;
c333=0;

C(1,1)=c111*y4+c211*y5+c311*y6;
C(1,2)=c121*y4+c221*y5+c321*y6;
C(1,3)=c131*y4+c231*y5+c331*y6;
C(2,1)=c112*y4+c212*y5+c312*y6;
C(2,2)=c122*y4+c222*y5+c322*y6;
C(2,3)=c132*y4+c232*y5+c332*y6;
C(3,1)=c113*y4+c213*y5+c313*y6;
C(3,2)=c123*y4+c223*y5+c323*y6;
C(3,3)=c133*y4+c233*y5+c333*y6;

%----------------Parameters of g(q)---------------%
g=zeros(3,1);
g(1,1)=0;
g(2,1)=-48.5564*cos(y2)+1.0462*sin(y2)+0.3683*cos(y2+y3)-10.6528*sin(y2+y3);
g(3,1)=0.3683*cos(y2+y3)-10.6528*sin(y2+y3);
end
```

## 6.4. Desired Trajectory Function

```
%------------Function to compute Desired Trajectory-----------------%
function [Yd,Yddot,Yddot_double]=desired_traj(w,t)

Yd=zeros(3,1);
Yddot=zeros(3,1);
Yddot_double=zeros(3,1);

Yd(1,1)=-0.2165*cos(w*t)-0.56;
Yd(2,1)=0.25*sin(w*t);
Yd(3,1)=0.125*cos(w*t)-0.08;

Yddot(1,1)=0.2165*sin(w*t)*w;
Yddot(2,1)=0.25*cos(w*t)*w;
Yddot(3,1)=-0.125*sin(w*t)*w;

Yddot_double(1,1)=0.2165*cos(w*t)*w^2;
Yddot_double(2,1)=-0.25*sin(w*t)*w^2;
Yddot_double(3,1)=-0.125*cos(w*t)*w^2;
end
```

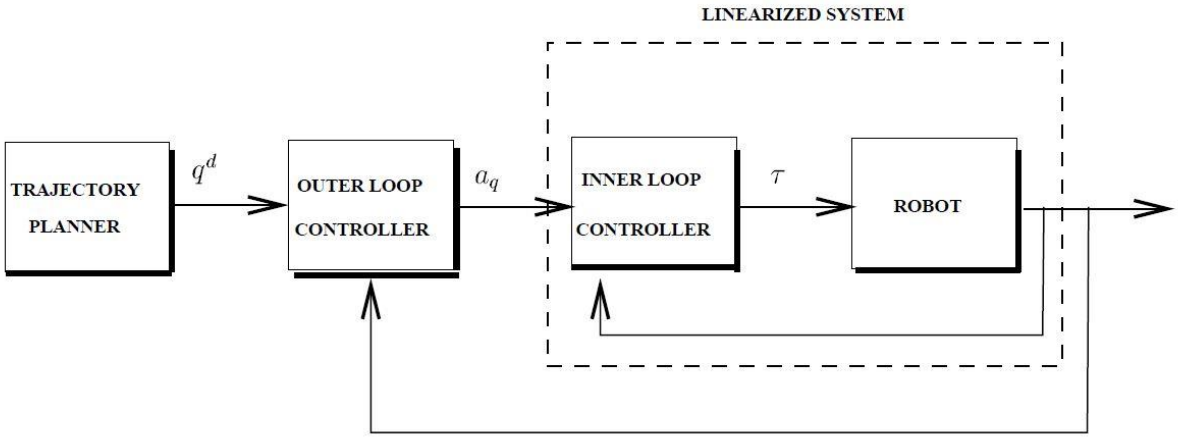
## 6.5. Jacobian Function

```
%--------Function for Jacobian and Derivative----------%
function [J,Jdot,qdot]=jacobian(q,Ydot)
Ydot=[0 0 0];
```

```
J=zeros(3,3);
Jdot=zeros(3,3);

J(1,1)= 0.02032*sin(q(1))*cos(q(2)+q(3)) - 0.43307*sin(q(1))*sin(q(2)+q(3)) -
0.4318*sin(q(1))*cos(q(2)) -0.14909*cos(q(1));
J(1,2)= 0.02032*cos(q(1))*sin(q(2)+q(3)) + 0.43307*cos(q(1))*cos(q(2)+q(3)) -
0.4318*cos(q(1))*sin(q(2));
J(1,3)= 0.02032*cos(q(1))*sin(q(2)+q(3)) + 0.43307*cos(q(1))*cos(q(2)+q(3));

J(2,1)= -0.02032*cos(q(1))*sin(q(2)+q(3)) + 0.43307*cos(q(1))*sin(q(2)+q(3))
+ 0.4318*cos(q(1))*cos(q(2)) - 0.14909*sin(q(1));
J(2,2)= 0.02032*sin(q(1))*sin(q(2)+q(3)) + 0.43307*sin(q(1))*cos(q(2)+q(3)) -
0.4318*sin(q(1))*sin(q(2));
J(2,3)= 0.02032*sin(q(1))*sin(q(2)+q(3)) + 0.43307*sin(q(1))*cos(q(2)+q(3));

J(3,1)= 0;
J(3,2)= 0.02032*cos(q(2)+q(3)) - 0.43307*sin(q(2)+q(3)) - 0.4318*cos(q(2));
J(3,3)= 0.02032*cos(q(2)+q(3)) - 0.43307*sin(q(2)+q(3));

Jdot(1,1)= 0.02032*cos(q(1))*cos(q(2)+q(3)) -
0.43307*cos(q(1))*sin(q(2)+q(3)) - 0.4318*cos(q(1))*cos(q(2))
+0.14909*sin(q(1));
Jdot(1,2)= 0.02032*cos(q(1))*cos(q(2)+q(3)) -
0.43307*cos(q(1))*sin(q(2)+q(3)) - 0.4318*cos(q(1))*cos(q(2));
Jdot(1,3)= 0.02032*cos(q(1))*cos(q(2)+q(3)) -
0.43307*cos(q(1))*sin(q(2)+q(3));

Jdot(2,1)= 0.02032*sin(q(1))*sin(q(2)+q(3)) -
0.43307*sin(q(1))*sin(q(2)+q(3)) - 0.4318*sin(q(1))*cos(q(2)) -
0.14909*cos(q(1));
Jdot(2,2)= 0.02032*sin(q(1))*cos(q(2)+q(3)) -
0.43307*sin(q(1))*sin(q(2)+q(3)) - 0.4318*sin(q(1))*cos(q(2));
Jdot(2,3)= 0.02032*sin(q(1))*cos(q(2)+q(3)) -
0.43307*sin(q(1))*sin(q(2)+q(3));

Jdot(3,1)= 0;
Jdot(3,2)= -0.02032*sin(q(2)+q(3)) - 0.43307*cos(q(2)+q(3)) +
0.4318*sin(q(2));
Jdot(3,3)= -0.02032*sin(q(2)+q(3)) - 0.43307*cos(q(2)+q(3));

qdot= inv(J)*Ydot.';
end
```

# 7. Discussion

(1) Explain the linearization and decoupling. How do you verify them?




The above figure from *"Robot Dynamics and Control"* Mark W. Spong, Chapter 11 illustrates the notion of inner-loop/outer-loop control. By this we mean that the computation of the nonlinear control is performed in an inner loop, with the vectors q, $\dot{q}$ (qd), and $\ddot{q}$ (aq) as its inputs and τ as output. The outer loop in the system is then the computation of the additional input term aq. Note that the outer loop control $\ddot{q}$ (aq) is more in line with the notion of a feedback control in the usual sense of being error driven. The design of the outer loop feedback control is in theory greatly simplified since it is designed for the plant represented by the dotted, which is now a linear or nearly linear system.
The process of converting a non-linear dynamic system into a linear dynamic system by use of non-linear feedback is known as Linearization and decoupling. We can verify by analyzing if each input designed to control the scalar linear system, affects the individual outputs without any dependency on the motion of other links

(2) What kind of effects do the robot's initial position and the velocity of the desired trajectory have on the tracking errors?
Given the initial position and velocity, we can calculate the values of q_initial from the above-mentioned equations. Additionally, higher the difference between q_actual and q_desired, higher will be the amplitudes of tracking error, and longer it takes for them to converge

(3) Explain the effects of modeling errors on the robot tracking control.
Modelling errors affect the calculation of q. By controlling the $K_p$ and $K_d$ values, the values of amplitude and tracking can be modified. Modelling errors induce inaccuracy in calculation of Jacobian, which further leads to errors in the differential equation solutions for the complete time span.

(4) Propose a method to reduce the effects of modeling errors and verify your ideas by the simulation.

According to *"Robot Dynamics and Control"* Mark W. Spong, methods to reduce the effects of modelling errors include Passivity based adaptive control and Robust Inverse Dynamics.

# 8. References

[1]. Robot Dynamics and Control, Second Edition, Mark W. Spong, Seth Hutchinson, and M. Vidyasagar

[2]. http://ece801.blogspot.com/

[3]. Mathworks Central