Dynamic Analysis of 2-Link Manipulator

Team Name: System Busters

Pendela V P B Krishna Chaitanya 2023102029
Guru Shwadeep Dornadula 2023112020
Nunna Sri Abhinaya 2023102071
Aikya Oruganti 2023102062
Indira 2023102070
Hasitha Alamanda 2023102072
Sudershan Sarraf 2023102015
Astitva Ranjan 2019112025

September 25, 2024

1 Introduction

In this paper we will be analysing a basic 2-link manipulator with P,PD,PI, and PID controllers. Our goal is to make the initial angluar positions to 0.

2 System Dynamics of 2-Link Manipulator

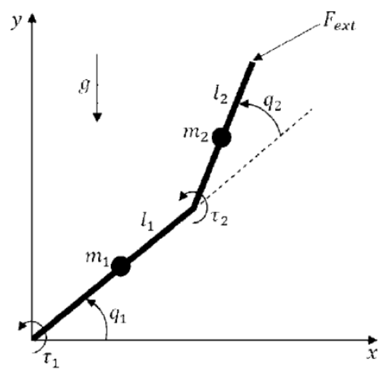


Fig. 1. Two-Link Robotic Manipulator

To analyse the system, we determine the Mass matrix, Coriolis matrix and the Gravitational matrix. KE and PE are used to determine the mass and gravitational matrices. The joint Torques are given by the equation:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

which when expanded gives:

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix}M_{11} & M_{12}\\ M_{21} & M_{22}\end{bmatrix}\begin{bmatrix}\ddot{q_1}\\ \ddot{q_2}\end{bmatrix} + \begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}\begin{bmatrix}\dot{q_1}\\ \dot{q_2}\end{bmatrix} + \begin{bmatrix}G_{11}\\ G_{12}\end{bmatrix}$$

In order to find Mass-Matrix we use the total KE of the system.
Let r_1 and r_2 be the distances of COM of links from the joints.
The equations are as follows:

$$C_{1x} = x_1\cos(q_1), \quad C_{2x} = l_1\cos(q_1) + l_2\cos(q_1+q_2)$$
$$C_{1y} = x_1\sin(q_1), \quad C_{2y} = l_1\sin(q_1) + l_2\sin(q_1+q_2)$$

The velocity squared term for v_j^2 is:

$$v_j^2 = (\dot{x}_i)^2 + (\dot{y}_i)^2$$

Expanding this:

$$= x_1^2\sin^2(q_1)\dot{q}_1^2 + x_1^2\cos^2(q_1)\dot{q}_1^2$$

which simplifies to:

$$v_1^2 = x_1^2 \dot{q}_1^2$$

The kinetic energy KE_{link1} is given by:

$$KE_{\text{link1}} = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} I_1 \dot{q}_1^2$$

$$KE_{\text{link1}} = \frac{1}{2} \begin{bmatrix} \dot{q}_1 & \dot{q}_2 \end{bmatrix} \begin{bmatrix} m_2 r_1^2 + I_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

For link 2:

Here:

$$C_{ax}(q_1, q_2), \quad C_{ay}(q_1, q_2)$$

Since q_1, q_2 are independent, from Jacobian theorem:

$$\frac{\partial(C_{ax}, C_{ay})}{\partial(q_1, q_2)} = \begin{bmatrix} \frac{\partial C_{ax}}{\partial q_1} & \frac{\partial C_{ax}}{\partial q_2} \\ \frac{\partial C_{ay}}{\partial q_1} & \frac{\partial C_{ay}}{\partial q_2} \end{bmatrix}$$

This is because

$$\frac{dC_x}{d(q_1, q_2)} = \frac{l_2 \dot{q}_2}{l_2 \dot{x}}$$

$$\frac{dC_{xy}}{d(q_1, q_2)} = \dot{v}_{2y}$$

$$v_2^2 = \dot{v}_{2x}^2 + \dot{v}_{2y}^2$$

Kinetic Energy link 2:

$$KE_{\text{link2}} = \frac{1}{2} m_2 v_2^2 + \frac{1}{2} I_2 (\dot{q}_2^2 + \dot{q}_1^2)$$

Kinetic Energy link 2 (expanded):

$$KE_{\text{link2}} = \frac{1}{2} (\dot{q}_1, \dot{q}_2) (\mathbf{J}^T \mathbf{J} m_2 + I_2 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

Jacobian matrix $\mathbf{J}$ is given by:

$$\mathbf{J} = \begin{bmatrix} -l_1 \sin(q_1) - r_2 \sin(q_1 + q_2) & -r_2 \sin(q_1 + q_2) \\ l_1 \cos(q_1) + r_2 \cos(q_1 + q_2) & r_2 \cos(q_1 + q_2) \end{bmatrix}$$

Transpose of Jacobian $\mathbf{J}^T$:

$$\mathbf{J}^T = \begin{bmatrix} l_1 \sin(q_1) + r_2 \sin(q_1 + q_2) & -l_1 \cos(q_1) - r_2 \cos(q_1 + q_2) \\ r_2 \sin(q_1 + q_2) & -r_2 \cos(q_1 + q_2) \end{bmatrix}$$

$$\mathbf{J}^T \cdot \mathbf{J} = \begin{bmatrix} (l_1 \sin(q_1) + l_2 \sin(q_1 + q_2))^2 + (l_1 \cos(q_1) + l_2 \cos(q_1 + q_2))^2 & J_{12} \\ J_{21} & J_{22} \end{bmatrix}$$

$$J_{11} = l_1^2 + l_2^2 + 2l_1l_2[\cos(q_2)]$$

$$J_{12} = -(l_1\sin(q_1)+l_2\sin(q_1+q_2))l_2\sin(q_1+q_2)+(l_1\cos(q_1)+l_2\cos(q_1+q_2))l_2\cos(q_1+q_2)$$

$$J_{12} = l_1l_2(\cos(q_2)) + l_2^2$$

$$J_{21} = J_{12}$$

$$J_{22} = l_2^2\sin^2(q_1+q_2) + l_2^2\cos^2(q_1+q_2) = l_2^2$$

Total Kinetic Energy:

$$KE = \frac{1}{2}\dot{q}^T m\dot{q}$$

The mass matrix m is given by:

$$m = \begin{bmatrix} m_1l_1^2 + m_2l_1^2 + m_2l_2^2 + 2m_2l_1l_2\cos(q_2) + I_1 + I_2 & m_2l_2^2 + m_2l_1l_2\cos(q_2) + I_2 \\ m_2l_2^2 + m_2l_1l_2\cos(q_2) + I_2 & m_2l_2^2 + I_2 \end{bmatrix}$$

$$M_{11} = m_1l_1^2 + m_2l_1^2 + m_2l_2^2 + 2m_2l_1l_2\cos(q_2) + I_1 + I_2$$
$$M_{12} = m_2l_2^2 + m_2l_1l_2\cos(q_2) + I_2$$
$$M_{22} = m_2l_2^2 + I_2$$

To get the desired M matrix:

$$r_1 = l_1, \quad r_2 = l_2, \quad I_1 = I_2 = 0$$

which is not practically possible.

Potential Energy (PE):

$$PE = m_1gh_1 + m_2gh_2$$

where

$$h_1 = \text{l.o.c. of com of link 1} = l_1\sin(q_1)$$
$$h_2 = \text{l.o.c. of com of link 2} = l_2\sin(q_1+q_2) + l_1\sin(q_1)$$

Thus, the potential energy becomes:

$$PE = m_1gl_1\sin(q_1) + m_2g\left(l_1\sin(q_1) + l_2\sin(q_1+q_2)\right)$$

The gravity matrix G:

$$G = \begin{bmatrix} \frac{\partial PE}{\partial q_1} \\ \frac{\partial PE}{\partial q_2} \end{bmatrix} = \begin{bmatrix} m_1gl_1\cos(q_1) + m_2g(l_1\cos(q_1) + l_2\cos(q_1+q_2)) \\ m_2gl_2\cos(q_1+q_2) \end{bmatrix}$$

For $q_1 = l_1, q_2 = l_2$ to satisfy requirements.
The Coriolis matrix C:

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

where

$$C_{kj} = \sum_{i=1}^{n} C_{ijk}\dot{q}_i$$

$$C = \begin{bmatrix} C_{111}\dot{q}_1 + C_{211}\dot{q}_2 & C_{121}\dot{q}_1 + C_{222}\dot{q}_2 \\ C_{112}\dot{q}_1 + C_{212}\dot{q}_2 & C_{122}\dot{q}_1 + C_{222}\dot{q}_2 \end{bmatrix}$$

$$C_{ijk} = \frac{1}{2}\left(\frac{\partial m_{kj}}{\partial q_i} + \frac{\partial m_{ki}}{\partial q_j} - \frac{\partial m_{ij}}{\partial q_k}\right)$$

$$C_{111} = \frac{1}{2}\left(\frac{\partial m_{11}}{\partial q_1} + \frac{\partial m_{11}}{\partial q_1} - \frac{\partial m_{11}}{\partial q_1}\right)$$

$$C_{111} = 0$$

$$C_{211} = \frac{1}{2}\left(\frac{\partial m_{11}}{\partial q_2} + \frac{\partial m_{12}}{\partial q_1} - \frac{\partial m_{21}}{\partial q_1}\right)$$

$$C_{211} = -m_2 l_1 r_2 \sin(q_2)$$

$$C_{121} = \frac{1}{2}\left(\frac{\partial m_{21}}{\partial q_1} + \frac{\partial m_{11}}{\partial q_2} - \frac{\partial m_{12}}{\partial q_1}\right)$$

$$C_{121} = C_{211} = -m_2 l_1 r_2 \sin(q_2)$$

$$C_{221} = \frac{1}{2}\left(\frac{\partial m_{21}}{\partial q_2} + \frac{\partial m_{12}}{\partial q_2} - \frac{\partial m_{12}}{\partial q_1}\right)$$

$$C_{221} = C_{211} = -m_2 l_1 r_2 \sin(q_2)$$

$$C_{112} = \frac{1}{2}\left(\frac{\partial m_{12}}{\partial q_1} + \frac{\partial m_{21}}{\partial q_1} - \frac{\partial m_{11}}{\partial q_2}\right)$$

$$C_{112} = -C211 = m_2 l_1 r_2 \sin(q_2)$$

$$C_{212} = \frac{1}{2}\left(\frac{\partial m_{12}}{\partial q_2} + \frac{\partial m_{22}}{\partial q_1} - \frac{\partial m_{21}}{\partial q_2}\right)$$

$$C_{212} = 0$$

$$C_{122} = \frac{1}{2}\left(\frac{\partial m_{22}}{\partial q_1} + \frac{\partial m_{21}}{\partial q_2} - \frac{\partial m_{12}}{\partial q_2}\right)$$

$$C_{122} = 0$$

$$C_{222} = \frac{1}{2}\left(\frac{\partial m_{22}}{\partial q_2} + \frac{\partial m_{22}}{\partial q_2} - \frac{\partial m_{22}}{\partial q_2}\right)$$

$$C_{222} = 0$$

$$C = \begin{bmatrix} -m_2 l_1 r_2 \sin(q_2)\dot{q}_2 & -m_2 l_1 r_2 \sin(q_2)(\dot{q}_1 + \dot{q}_2) \\ 0 & m_2 l_1 r_2 \sin(q_2)\dot{q}_1 \end{bmatrix}$$

If we put

$$r = l$$

$$I_1 = I_2 = 0$$

Final Matrices are:

$$M = \begin{bmatrix} (m_1 + m_2)l_1^2 + m_2 l_2(l_2 + 2l_1\cos(q_2) & m_2 l_2(l_2 + l_1\cos(q_2) \\ m_2 l_2(l_2 + l_1\cos(q_2) & m_2 l_2^2 \end{bmatrix}$$

$$C = \begin{bmatrix} -m_2 l_1 l_2 \sin(q_2)\dot{q}_2 & -m_2 l_1 l_2 \sin(q_2)(\dot{q}_1 + \dot{q}_2) \\ 0 & m_2 l_1 l_2 \sin(q_2)\dot{q}_1 \end{bmatrix}$$

$$G = \begin{bmatrix} m_1 g l_1 \cos(q_1) + m_2 g(l_1\cos(q_1) + l_2\cos(q_1 + q_2)) \\ m_2 g l_2 \cos(q_1 + q_2) \end{bmatrix}$$

In our analysis we consider

$$m_1 = 10kg m_2 = 5kg$$

$$l_1 = 0.2m l_2 = 0.1m$$

$$g = 9.81$$

Initial joint angles are:

$$\begin{bmatrix} q_1(0) \\ q_2(0) \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} rad$$

The objective is to get the angular position to

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} rad$$

3 P Controller

In this section we will be dealing with the effects of using only a proportional gain in the feedback loop. A proportional controller adjusts the control input proportionally to the error. The higher the gain, the faster the response, but too high a gain can cause oscillations.

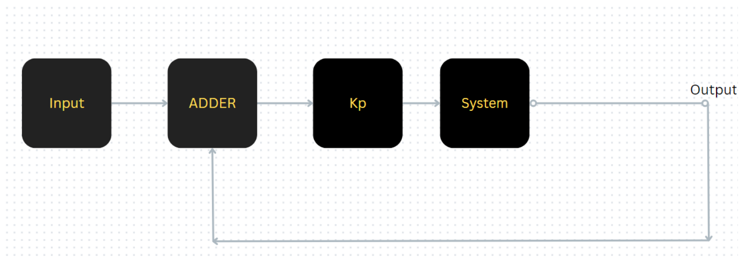


Figure 1: Proportional gain

$$\ddot{q} = -M^{-1}(q)\left[C(q,\dot{q})\dot{q} + G(q)\right] + \hat{\tau}$$

where

$$\hat{\tau} = M^{-1}(q)\tau$$

Thus we have the system taking new inputs

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

where the actual physical torque inputs are

$$\tau = M(q)\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

Let the error signals be denoted by e

$$e(q_1) = q_{1f} - q_1$$

$$e(q_2) = q_{2f} - q_2$$

where q_{1f} and q_{2f} are the desired final values of the angles. Assume the system has the initial positions

$$q_0 = \begin{bmatrix} q_1(0) \\ q_2(0) \end{bmatrix}$$

A common P controller follows the following equation:

$$f = K_P e$$

Therefore the corresponding f_1 and f_2 values will be

$$f_1 = K_{P_1} e(q_1)$$

$$f_2 = K_{P_2} e(q_2)$$

Thus we have

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} K_{P_1}(q_{1f} - q_1) \\ K_{P_2}(q_{2f} - q_2) \end{bmatrix}$$

State variables to be introduced:

$$x_1 = q_1 \quad x_2 = q_2$$

$$x_3 = \dot{q_1} \quad x_4 = \dot{q_2}$$

The function is implemented using 'ode45' but this requires us to give the inputs as the derivative matrix of the state variable. Hence we are required to convert these into derivatives of state variables.

$$\dot{x_1} = x_3 = \dot{q_1} \quad \dot{x_2} = x_4 = \dot{q_2}$$

$$\dot{x_3} = \ddot{q_1} = \phi(t, x1, x2, x3, x4)$$

$$\dot{x_4} = \ddot{q_2} = \psi(t, x1, x2, x3, x4)$$

Pmain.m

```
%P 2-link
%Assign the given masses and lengths

m1 = 10; % Mass of link 1
m2 = 5;  % Mass of link 2
l1 = 0.2; % Length of link 1
l2 = 0.1; % Length of link 2
g = 9.81; % Gravitational acceleration

%Initial values for joint angles and velocities
x10 = 0; % Initial integral error for joint 1
x20 = 0; % Initial integral error for joint 2
q10 = 0.1;  % Initial angle of link 1
q20 = 0.1;  % Initial angle of link 2
q1dot0 = 0;  % Initial angular velocity of link 1
q2dot0 = 0;  % Initial angular velocity of link 2

%Time span for the simulation
t0 = 0;
tf = 10;
```

```
tspan = [t0, tf];

%Desired final angles for all joints
q1_fin = 0;
q2_fin = 0;

%P gains for all joints

P_values = [1, 10, 50, 100];

for i = 1:length(P_values)
kp1 = P_values(i);
kp2 = P_values(i);

%Initial conditions for the ODE solver
IC = [x10, x20, q10, q20, q1dot0, q2dot0];

%Options for the ODE solver
options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);

%Solve the system of ODEs using ode45
[time, state_values] = ode45(@(t,s) p_2link(t, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, kp

%Extract the joint angles and velocities from the state values
q1 = state_values(:,1);
q2 = state_values(:,2);

%Plot the joint angles over time
figure;
subplot(2,1,1);
plot(time, q1, 'r', 'LineWidth', 1.5);
title('Joint Angle 1 vs Time');
xlabel('Time (s)');
ylabel('Angle (rad)');
subplot(2,1,2);
plot(time, q2, 'b', 'LineWidth', 1.5);
title('Joint Angle 2 vs Time');
xlabel('Time (s)');
ylabel('Angle (rad)');

%Plot error over time
e1 = q1_fin - q1;
e2 = q2_fin - q2;

figure;
subplot(2,1,1);
```

```
plot(time, e1, 'r', 'LineWidth', 1.5);
title('Error in Joint Angle 1 vs Time');
xlabel('Time (s)');
ylabel('Error (rad)');
subplot(2,1,2);
plot(time, e2, 'b', 'LineWidth', 1.5);
title('Error in Joint Angle 2 vs Time');
xlabel('Time (s)');
ylabel('Error (rad)');

end
```

The ode45 function solves the ODEs of the form $\dot{y}$ = f(t, y) and returns a vector $\begin{bmatrix} t & y \end{bmatrix}$. The inputs to this function are a general function that returns $\begin{bmatrix} t & u \end{bmatrix}$ where u is a vector containing the derivatives of the state variables, a vector tspan containing the time duration of analysis and another vector IC,containing the initial conditions of the state variables.

Below is the p-2link function file

$$p - 2link.m$$

```
    %P controller
function output = p_2link(~, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, kp2)
    %Declare the state variables [q1, q2, q1dot, q2dot]
    q1 = s(1); q2 = s(2);
    q1dot = s(3); q2dot = s(4);

    %P Control Errors
    e1 = q1_fin - q1;
    e2 = q2_fin - q2;

    %P control inputs
    f1 = kp1 * e1;
    f2 = kp2 * e2;

    %Mass matrix for the 2-link system
    M11 = (m1 + m2) * (l1*l1) + m2 * l2*(l2 + 2 * l1 * cos(q2));
    M22 = m2 * l2*l2;
    M12 = m2*l2*(l2+l1*cos(q2));
    M21 = M12;

    %Create the Mass matrix M (2x2)
    M = [M11, M12; M21, M22];

    %Coriolis and Centrifugal Matrix
    c11 = -m2 * l1 * l2 * sin(q2) * q2dot;
    c12 = -m2 * l1 * l2 * sin(q2) * (q1dot + q2dot);
    c22 = m2 * l1 * l2 * sin(q2) * q2dot;
    c21 = 0;

    %Create the Coriolis and Centrifugal matrix C (2x2)
    C = [c11, c12; c21, c22];

    %Gravitational Matrix
    G1 = m1 * g * l1 * cos(q1) + m2 * g * (l1 * cos(q1) + l2 * cos(q1 + q2));
    G2 = m2 * g * l2 * cos(q1 + q2);

    %Create the Gravitational matrix G (2x1)
    G = [G1; G2];

    %Control input vector (torques)
    f = [f1; f2];

    %Joint velocities and accelerations
    q_dot_matrix = [q1dot; q2dot];
    q_double_dot_matrix = inv(M) * (- C * q_dot_matrix - G) + f;

    %Convert double dot matrix to a row vector
    q_double_dot_matrix_row_1 = q_double_dot_matrix(1);
    q_double_dot_matrix_row_2 = q_double_dot_matrix(2);
    %Output the time derivatives of the state variables
    output = [q1dot; q2dot; q_double_dot_matrix_row_1; q_double_dot_matrix_row_2];
end
```

4 PI Controller

In this section we will be dealing with PI system to make sure that we dont have the steady state error we have been getting in the previous part.

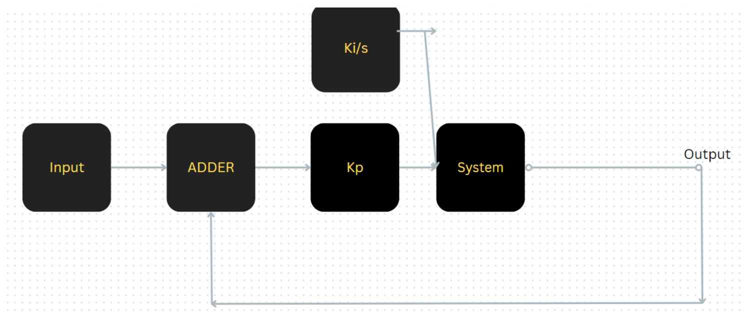


Figure 2: Proportional Integral gain

$$\ddot{q} = -M^{-1}(q)\left[C(q,\dot{q})\dot{q} + G(q)\right] + \hat{\tau}$$

where

$$\hat{\tau} = M^{-1}(q)\tau$$

Thus we have the system taking new inputs

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

where the actual physical torque inputs are

$$\tau = M(q)\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

Let the error signals be denoted by e

$$e(q_1) = q_{1f} - q_1$$

$$e(q_2) = q_{2f} - q_2$$

where q_{1f} and q_{2f} are the desired final values of the angles. Assume the system has the initial positions

$$q_0 = \begin{bmatrix} q_1(0) \\ q_2(0) \end{bmatrix}$$

A common PI controller follows the following equation:

$$f = K_P e + K_I \int e\, dt$$

Therefore the corresponding f_1 and f_2 values will be

$$f_1 = K_{P_1} e(q_1) + K_{I_1} \int e(q_1)\, dt$$

$$f_2 = K_{P_2} e(q_2) + +K_{I_2} \int e(q_2)\, dt$$

Thus we have

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} K_{P_1}(q_{1f} - q_1) + K_{I_1}\int(q_{1f} - q_1)\,dt \\ K_{P_2}(q_{2f} - q_2) + K_{I_2}\int(q_{2f} - q_2)\,dt \end{bmatrix}$$

State variables to be introduced:

$$x_1 = q_1 \quad x_2 = q_2$$

$$x_3 = \int e_1\,dt$$

$$x_4 = \int e_2\,dt$$

$$x_5 = \dot{q}_1 \quad x_6 = \dot{q}_2$$

$$\dot{x_1} = \dot{q_1} \quad \dot{x_2} = \dot{q_2}$$

$$\dot{x_3} = e_1 \quad \dot{x_4} = e_2$$

$$\dot{x_5} = \ddot{q_1} = \phi(t, x1, x2, x3, x4, x5, x6)$$

$$\dot{x_6} = \ddot{q_2} = \psi(t, x1, x2, x3, x4, x5, x6)$$

PImain.m

```
    %PI 2-link
%Assign masses and lengths
m1 = 10; % Mass of link 1
m2 = 5;  % Mass of link 2
l1 = 0.2; % Length of link 1
l2 = 0.1; % Length of link 2
g = 9.81; % Gravitational acceleration

%Initial values for joint angles and velocities
x10 = 0; % Initial integral error for joint 1
x20 = 0; % Initial integral error for joint 2
q10 = 0.1;  % Initial angle of link 1
q20 = 0.1;  % Initial angle of link 2
q1dot0 = 0;  % Initial angular velocity of link 1
q2dot0 = 0;  % Initial angular velocity of link 2

%Time span for the simulation
t0 = 0;
tf = 10;
tspan = [t0, tf];
```

```
%Desired final angles for all joints
q1_fin = 0;
q2_fin = 0;

%PI gains for all joints
% kp1 = 10; ki1 = 0;
% kp2 = 10; ki2 = 0;
% kp1 = 50; ki1 = 0;
% kp2 = 50; ki2 = 0;
% kp1 = 100; ki1 = 0;
% kp2 = 100; ki2 = 0;
% kp1 = 500; ki1 = 0;
% kp2 = 500; ki2 = 0;
% kp1 = 500; ki1 = 1;
% kp2 = 500; ki2 = 1;
kp1 = 500; ki1 = 10;
kp2 = 500; ki2 = 10;
% kp1 = 500; ki1 = 50;
% kp2 = 500; ki2 = 50;

%Initial conditions for the ODE solver
IC = [q10, q20, x10, x20, q1dot0, q2dot0];

%Options for the ODE solver
options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);

%Solve the system of ODEs using ode45
[time, state_values] = ode45(@(t,s) pi_2link(t, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, k

%Extract the joint angles and velocities from the state values
q1 = state_values(:,1);
q2 = state_values(:,2);
e1 = q1_fin - q1;
e2 = q2_fin - q2;
%Plot the joint angles over time
figure;
subplot(2,1,1);
plot(time, q1, 'b', 'LineWidth', 1.5);
title('Joint Angle q1 vs Time');
xlabel('Time (s)');
ylabel('Joint Angle q1 (rad)');
grid on;
subplot(2,1,2);
plot(time, q2, 'r', 'LineWidth', 1.5);
title('Joint Angle q2 vs Time');
xlabel('Time (s)');
```

```
ylabel('Joint Angle q2 (rad)');
grid on;

%Error Plots
figure;
subplot(2,1,1);
plot(time, e1, 'b', 'LineWidth', 1.5);
title('Error e1 vs Time');
xlabel('Time (s)');
ylabel('Error e1 (rad)');
grid on;
subplot(2,1,2);
plot(time, e2, 'r', 'LineWidth', 1.5);
title('Error e2 vs Time');
xlabel('Time (s)');
ylabel('Error e2 (rad)');
grid on;
```

Below is code for PI controller function

$$pi - 2link.m$$

```
    %PI code
function output = pi_2link(~, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, kp2, ki1, ki2)
    %Declare the state variables [q1, q2, u1, u2, q1dot, q2dot]
    q1 = s(1); q2 = s(2);
    u1 = s(3); u2 = s(4);
    q1dot = s(5); q2dot = s(6);

    %PI Control Errors
    e1 = q1_fin - q1;
    e2 = q2_fin - q2;

    %PI control inputs
    f1 = kp1 * e1 + ki1 * u1;
    f2 = kp2 * e2 + ki2 * u2;

    %Mass matrix for the 2-link system
    M11 = (m1 + m2) * (l1*l1) + m2 * l2*(l2 + 2 * l1 * cos(q2));
    M22 = m2 * l2*l2;
    M12 = m2*l2*(l2+l1*cos(q2));
    M21 = M12;

    %Create the Mass matrix M (2x2)
    M = [M11, M12; M21, M22];
    %Coriolis and Centrifugal Matrix
    c11 = -m2 * l1 * l2 * sin(q2) * q2dot;
    c12 = -m2 * l1 * l2 * sin(q2) * (q1dot + q2dot);
    c22 = m2 * l1 * l2 * sin(q2) * q2dot;
    c21 = 0;

    %Create the Coriolis and Centrifugal matrix C (2x2)
    C = [c11, c12; c21, c22];

    %Gravitational Matrix
    G1 = m1 * g * l1 * cos(q1) + m2 * g * (l1 * cos(q1) + l2 * cos(q1 + q2));
    G2 = m2 * g * l2 * cos(q1 + q2);

    %Create the Gravitational matrix G (2x1)
    G = [G1; G2];

    %Control input vector (torques)
    f = [f1; f2];

    %Joint velocities and accelerations
    q_dot_matrix = [q1dot; q2dot];
    q_double_dot_matrix = inv(M) * (- C * q_dot_matrix - G) + f;

    %Convert double dot matrix to a row vector
    q_double_dot_matrix_row_1 = q_double_dot_matrix(1);
    q_double_dot_matrix_row_2 = q_double_dot_matrix(2);
    %Output the time derivatives of the state variables
    output = [q1dot; q2dot; e1; e2; q_double_dot_matrix_row_1; q_double_dot_matrix_row_2];
end
```

5 PD Controller

In this section we will be checking out PD controller to find out the steady state error and transient response.

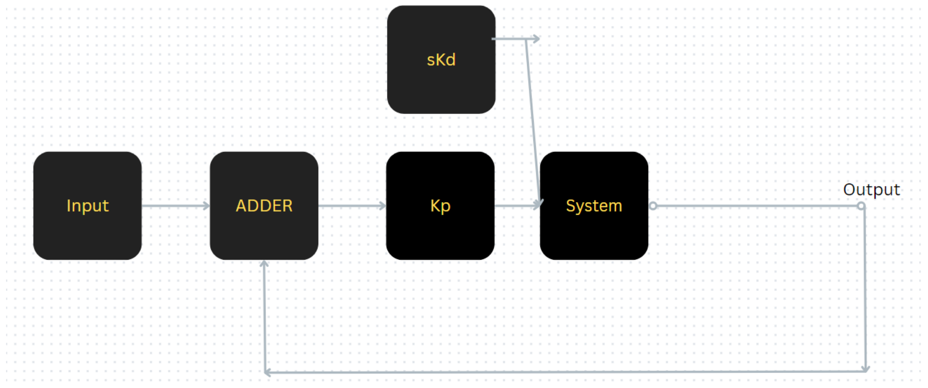


Figure 3: Proportional Differential

$$\ddot{q} = -M^{-1}(q)\left[C(q,\dot{q})\dot{q} + G(q)\right] + \hat{\tau}$$

where

$$\hat{\tau} = M^{-1}(q)\tau$$

Thus we have the system taking new inputs

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

where the actual physical torque inputs are

$$\tau = M(q)\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

Let the error signals be denoted by e

$$e(q_1) = q_{1f} - q_1$$

$$e(q_2) = q_{2f} - q_2$$

where q_{1f} and q_{2f} are the desired final values of the angles. Assume the system has the initial positions

$$q_0 = \begin{bmatrix} q_1(0) \\ q_2(0) \end{bmatrix}$$

A common PD controller follows the following equation:

$$f = K_P e + K_D \dot{e}$$

Therefore the corresponding f_1 and f_2 values will be

$$f_1 = K_{P_1} e(q_1) + K_{D_1}\dot{e}(q_1)$$

$$f_2 = K_{P_2} e(q_2) + K_{D_2}\dot{e}(q_2)$$

$$\implies f_1 = K_{P_1} e(q_1) + K_{D_1} \dot{q_1}$$
$$\implies f_2 = K_{P_2} e(q_2) + K_{D_2} \dot{q_2}$$

Thus we have

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} K_{P_1}(q_{1f} - q_1) + K_{D_1}\dot{q_1} \\ K_{P_2}(q_{2f} - q_2) + K_{D_2}\dot{q_2} \end{bmatrix}$$

State variables to be introduced:

$$x_1 = q_1 \quad x_2 = q_2$$
$$x_3 = \dot{q_1} \quad x_4 = \dot{q_2}$$

$$\dot{x_1} = \dot{q_1} = x_3 \quad \dot{x_2} = \dot{q_2} = x_4$$
$$\dot{x_3} = \ddot{q_1} = \phi(t, x1, x2, x3, x4)$$
$$\dot{x_4} = \ddot{q_2} = \psi(t, x1, x2, x3, x4)$$

PDmain.m

```
    %PD 2-Link
%Assign the given masses and lengths

m1 = 10; % Mass of link 1
m2 = 5;  % Mass of link 2
l1 = 0.2; % Length of link 1
l2 = 0.1; % Length of link 2
g = 9.81; % Gravitational acceleration

%Initial values for joint angles and velocities
q10 = 0.1;  % Initial angle of link 1
q20 = 0.1;  % Initial angle of link 2
q1dot0 = 0;  % Initial angular velocity of link 1
q2dot0 = 0;  % Initial angular velocity of link 2

%Time span for the simulation
t0 = 0;
tf = 10;
tspan = [t0, tf];

%Desired final angles for all joints
q1_fin = 0;
q2_fin = 0;
```

```
%PD gains for all joints
% kp1 = 500; kd1 = 10;
% kp2 = 500; kd2 = 10;
kp1 = 500; kd1 = 50;
kp2 = 500; kd2 = 50;
% kp1 = 500; kd1 = 100;
% kp2 = 500; kd2 = 100;
% kp1 = 500; kd1 = 500;
% kp2 = 500; kd2 = 500;

%Initial conditions for the ODE solver
IC = [q10, q20, q1dot0, q2dot0];

%Options for the ODE solver
options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);

%Solve the system of ODEs using ode45
[time, state_values] = ode45(@(t,s) pd_2link(t, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, k

%Extract the joint angles and velocities from the state values
q1 = state_values(:,1);
q2 = state_values(:,2);

%Plot the joint angles over time
figure;
subplot(2,1,1);
plot(time, q1, 'b', 'LineWidth', 1.5);
title('Joint Angle q1 vs Time');
xlabel('Time (s)');
ylabel('Joint Angle q1 (rad)');
grid on;
subplot(2,1,2);
plot(time, q2, 'r', 'LineWidth', 1.5);
title('Joint Angle q2 vs Time');
xlabel('Time (s)');
ylabel('Joint Angle q2 (rad)');
grid on;

%Error Plots
e1 = q1_fin - q1;
e2 = q2_fin - q2;

figure;
subplot(2,1,1);
plot(time, e1, 'b', 'LineWidth', 1.5);
title('Error e1 vs Time');
```

```
xlabel('Time (s)');
ylabel('Error e1 (rad)');
grid on;
subplot(2,1,2);
plot(time, e2, 'r', 'LineWidth', 1.5);
title('Error e2 vs Time');
xlabel('Time (s)');
ylabel('Error e2 (rad)');
grid on;
```

Below is the code for PD controller function

$$pd - 2link.m$$

```
    %PD 2-link
function output = pd_2link(~, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, kp2, kd1, kd2)
    %Declare the state variables [q1, q2, q1dot, q2dot]
    q1 = s(1); q2 = s(2);
    q1dot = s(3); q2dot = s(4);

    %PD Control Errors
    e1 = q1_fin - q1;
    e2 = q2_fin - q2;

    %PD control inputs
    f1 = kp1 * e1 - kd1 * q1dot;
    f2 = kp2 * e2 - kd2 * q2dot;

    %Mass matrix for the 2-link system
    M11 = (m1 + m2) * (l1*l1) + m2 * l2*(l2 + 2 * l1 * cos(q2));
    M22 = m2 * l2*l2;
    M12 = m2*l2*(l2+l1*cos(q2));
    M21 = M12;

    %Create the Mass matrix M (2x2)
    M = [M11, M12; M21, M22];

    %Coriolis and Centrifugal Matrix
    c11 = -m2 * l1 * l2 * sin(q2) * q2dot;
    c12 = -m2 * l1 * l2 * sin(q2) * (q1dot + q2dot);
    c22 = m2 * l1 * l2 * sin(q2) * q2dot;
    c21 = 0;

    %Create the Coriolis and Centrifugal matrix C (2x2)
    C = [c11, c12; c21, c22];
    %Gravitational Matrix
    G1 = m1 * g * l1 * cos(q1) + m2 * g * (l1 * cos(q1) + l2 * cos(q1 + q2));
    G2 = m2 * g * l2 * cos(q1 + q2);

    %Create the Gravitational matrix G (2x1)
    G = [G1; G2];

    %Control input vector (torques)
    f = [f1; f2];

    %Joint velocities and accelerations
    q_dot_matrix = [q1dot; q2dot];
    q_double_dot_matrix = inv(M) * (- C * q_dot_matrix - G) + f;

    %Convert double dot matrix to a row vector
    q_double_dot_matrix_row_1 = q_double_dot_matrix(1);
    q_double_dot_matrix_row_2 = q_double_dot_matrix(2);

    %Output the time derivatives of the state variables
    output = [q1dot; q2dot; q_double_dot_matrix_row_1; q_double_dot_matrix_row_2];
end
```

6 PID Controller

In this section we will be combining all the previous steps and make a complete PID controller which contains all the benefits of above and try to tune it for our requirement.

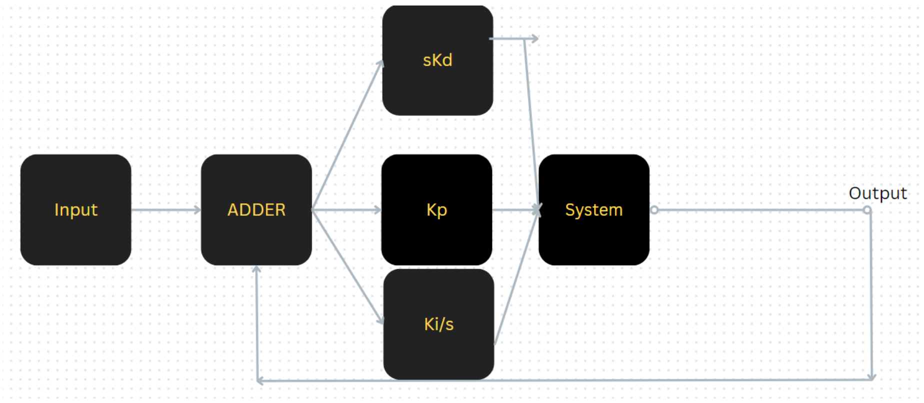


Figure 4: Proportional Differential Integral Gain

$$\ddot{q} = -M^{-1}(q)\left[C(q,\dot{q})\dot{q} + G(q)\right] + \hat{\tau}$$

where

$$\hat{\tau} = M^{-1}(q)\tau$$

Thus we have the system taking new inputs

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

where the actual physical torque inputs are

$$\tau = M(q)\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

Let the error signals be denoted by e

$$e(q_1) = q_{1f} - q_1$$

$$e(q_2) = q_{2f} - q_2$$

where q_{1f} and q_{2f} are the desired final values of the angles. Assume the system has the initial positions

$$q_0 = \begin{bmatrix} q_1(0) \\ q_2(0) \end{bmatrix}$$

A common PID controller follows the following design

$$f = K_P e - K_D \dot{e} + K_I \int e\,dt$$

Therefore the corresponding f_1 and f_2 values will be

$$f_1 = K_{P_1} e(q_1) - K_{D_1}\dot{e}(q_1) + K_{I_1}\int e(q_1)\,dt$$

$$\implies f_1 = K_{P_1}(q_{1f} - q_1) - K_{D_1}\dot{q}_1 + K_{I_1}\int (q_{1f} - q_1)\,dt$$

$$f_2 = K_{P_2} e(q_2) - K_{D_2}\dot{e}(q_2) + K_{I_2}\int e(q_2)\,dt$$

$$\implies f_2 = K_{P_2}(q_{2f} - q_2) - K_{D_2}\dot{q}_2 + K_{I_2}\int (q_{2f} - q_2)\,dt$$

Thus we have

$$\hat{\tau} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} K_{P_1}(q_{1f} - q_1) - K_{D_1}\dot{q}_1 + K_{I_1}\int (q_{1f} - q_1)\,dt \\ K_{P_2}(q_{2f} - q_2) - K_{D_2}\dot{q}_2 + K_{I_2}\int (q_{2f} - q_2)\,dt \end{bmatrix}$$

To implement the PID controller we introduce the following new state variables in our equation.

$$x_1 = \int e_1\,dt$$

$$x_2 = \int e_2 \, dt$$

$$x_3 = q_1 \quad x_4 = q_2$$

$$x_5 = \dot{q}_1 \quad x_6 = \dot{q}_2$$

Let's find out the derivatives

$$\dot{x_1} = \dot{q_1} \quad \dot{x_2} = \dot{q_2}$$

$$\dot{x_3} = e_1 \quad \dot{x_4} = e_2$$

$$\dot{x_5} = \ddot{q_1} = \phi(t, x1, x2, x3, x4, x5, x6)$$

$$\dot{x_6} = \ddot{q_2} = \psi(t, x1, x2, x3, x4, x5, x6)$$

$$PIDmain.m$$

```
    %PID 2-link
%Assign the given masses and lengths

m1 = 10; % Mass of link 1
m2 = 5;  % Mass of link 2
l1 = 0.2; % Length of link 1
l2 = 0.1; % Length of link 2
g = 9.81; % Gravitational acceleration

%Initial values for joint angles and velocities
x10 = 0; % Initial integral error for joint 1
x20 = 0; % Initial integral error for joint 2
q10 = 0.1;  % Initial angle of link 1
q20 = 0.1;  % Initial angle of link 2
q1dot0 = 0;  % Initial angular velocity of link 1
q2dot0 = 0;  % Initial angular velocity of link 2

%Time span for the simulation
t0 = 0;
tf = 10;
tspan = [t0, tf];

%Desired final angles for all joints
q1_fin = 0;
q2_fin = 0;

%PID gains for all joints
% kp1 = 500; kd1 = 50; ki1 = 10;
% kp2 = 500; kd2 = 50; ki2 = 10;
% kp1 = 500; kd1 = 50; ki1 = 50;
% kp2 = 500; kd2 = 50; ki2 = 50;
% kp1 = 500; kd1 = 50; ki1 = 100;
% kp2 = 500; kd2 = 50; ki2 = 100;
% kp1 = 500; kd1 = 50; ki1 = 500;
% kp2 = 500; kd2 = 50; ki2 = 500;
% kp1 = 500; kd1 = 100; ki1 = 500;
% kp2 = 500; kd2 = 100; ki2 = 500;
kp1 = 500; kd1 = 100; ki1 = 500;
kp2 = 1000; kd2 = 100; ki2 = 500;
% kp1 = 500; kd1 = 500; ki1 = 500;
% kp2 = 500; kd2 = 500; ki2 = 500;

%Initial conditions for the ODE solver
IC = [x10, x20, q10, q20, q1dot0, q2dot0];

%Options for the ODE solver
options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);

%Solve the system of ODEs using ode45
[time, state_values] = ode45(@(t,s) pid_2link(t, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1,

%Extract the joint angles and velocities from the state values
q1 = state_values(:,3);
q2 = state_values(:,4);

%Plot the joint angles over time
figure;
subplot(2,1,1);
plot(time, q1, 'b', 'LineWidth', 1.5);
title('Joint Angle q1 vs Time');
xlabel('Time (s)');
ylabel('Joint Angle q1 (rad)');
grid on;
subplot(2,1,2);
plot(time, q2, 'r', 'LineWidth', 1.5);
title('Joint Angle q2 vs Time');
xlabel('Time (s)');
ylabel('Joint Angle q2 (rad)');
grid on;

%Error Plots
e1 = q1_fin - q1;
e2 = q2_fin - q2;

figure;
```

```
subplot(2,1,1);
plot(time, e1, 'b', 'LineWidth', 1.5);
title('Error e1 vs Time');
xlabel('Time (s)');
ylabel('Error e1 (rad)');
grid on;
subplot(2,1,2);
plot(time, e2, 'r', 'LineWidth', 1.5);
title('Error e2 vs Time');
xlabel('Time (s)');
ylabel('Error e2 (rad)');
grid on;
```

The user implemented function for PID controller

$$pid - 2link.m$$

```
    %PID Code
function output = pid_2link(~, s, q1_fin, q2_fin, m1, m2, l1, l2, g, kp1, kp2, kd1, kd2, ki
    %Declare the stte variables [u1, u2, q1, q2, q1dot, q2dot]
    u1 = s(1); u2 = s(2);
    q1 = s(3); q2 = s(4);
    q1dot = s(5); q2dot = s(6);

    %PID Control Errors
    e1 = q1_fin - q1;
    e2 = q2_fin - q2;

    %PID control inputs
    f1 = kp1 * e1 - kd1 * q1dot + ki1 * u1;
    f2 = kp2 * e2 - kd2 * q2dot + ki2 * u2;

    %Mass matrix for the 2-link system
    M11 = (m1 + m2) * (l1*l1) + m2 * l2*(l2 + 2 * l1 * cos(q2));
    M22 = m2 * l2*l2;
    M12 = m2*l2*(l2+l1*cos(q2));
    M21 = M12;

    %Create the Mass matrix M (2x2)
    M = [M11, M12; M21, M22];

    %Coriolis and Centrifugal Matrix
    c11 = -m2 * l1 * l2 * sin(q2) * q2dot;
    c12 = -m2 * l1 * l2 * sin(q2) * (q1dot + q2dot);
    c22 = m2 * l1 * l2 * sin(q2) * q2dot;
    c21 = 0;

    %Create the Coriolis and Centrifugal matrix C (2x2)
    C = [c11, c12; c21, c22];

    %Gravitational Matrix
    G1 = m1 * g * l1 * cos(q1) + m2 * g * (l1 * cos(q1) + l2 * cos(q1 + q2));
    G2 = m2 * g * l2 * cos(q1 + q2);

    %Create the Gravitational matrix G (2x1)
    G = [G1; G2];

    %Control input vector (torques)
    f = [f1; f2];

    %Joint velocities and accelerations
    q_dot_matrix = [q1dot; q2dot];
    q_double_dot_matrix = inv(M) * (- C * q_dot_matrix - G) + f;

    %Convert double dot matrix to a row vector
    q_double_dot_matrix_row_1 = q_double_dot_matrix(1);
    q_double_dot_matrix_row_2 = q_double_dot_matrix(2);
    %Output the time derivatives of the state variables
    output = [e1; e2; q1dot; q2dot; q_double_dot_matrix_row_1; q_double_dot_matrix_row_2];
end
```

7 Simulation Results

7.1 Proportional Gain Controller

The values of P we are going to test are 1,10,50,100 and find a good value to stick on.

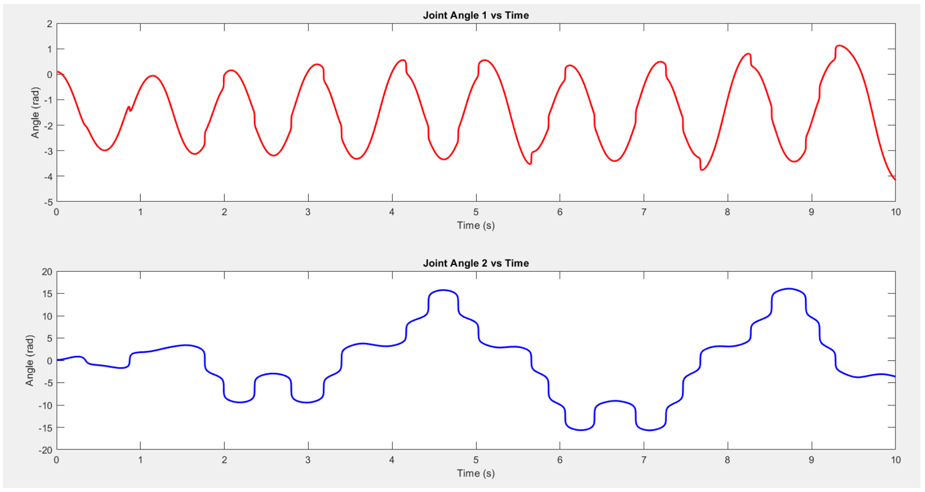


Figure 5: Kp = 1 Angles Plot

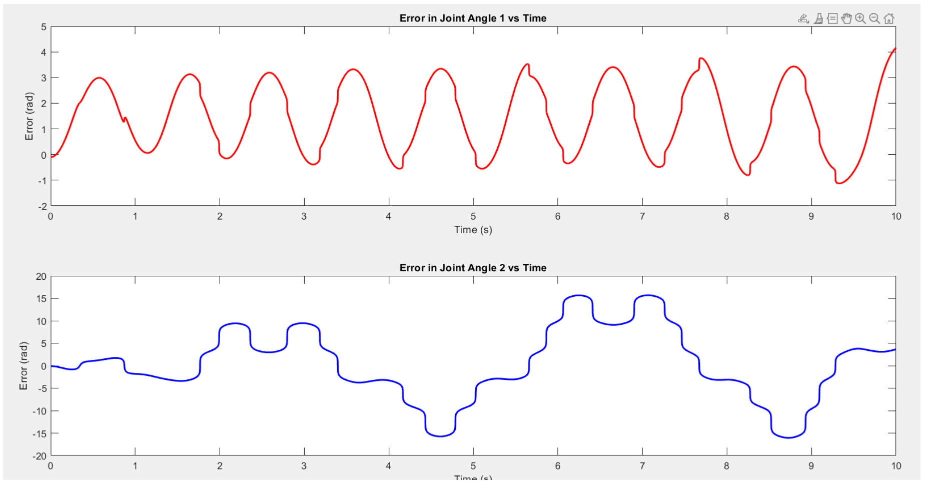


Figure 6: Kp = 1 Error Plot

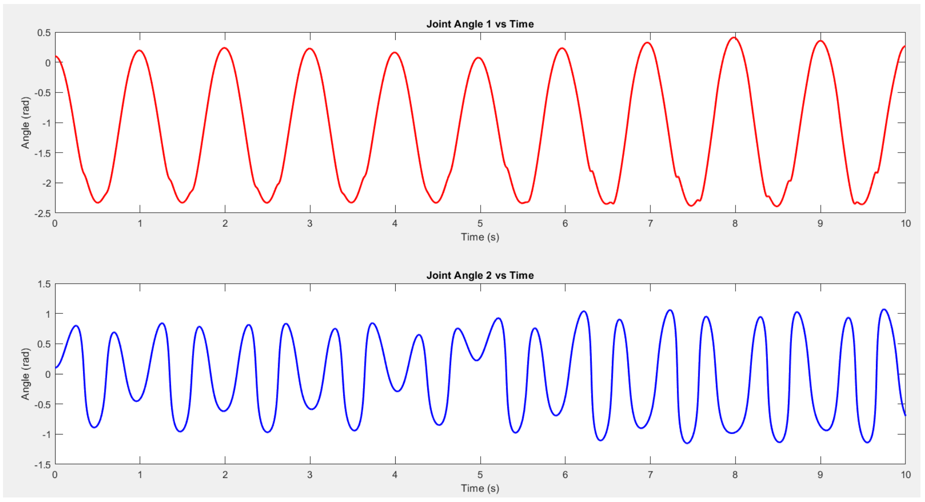


Figure 7: Kp = 10 Angles Plot

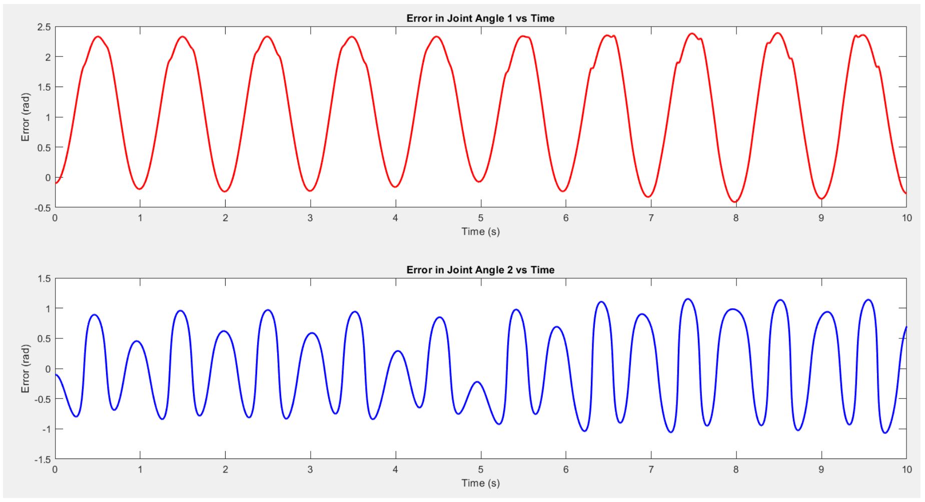


Figure 8: Kp = 10 Error Plot

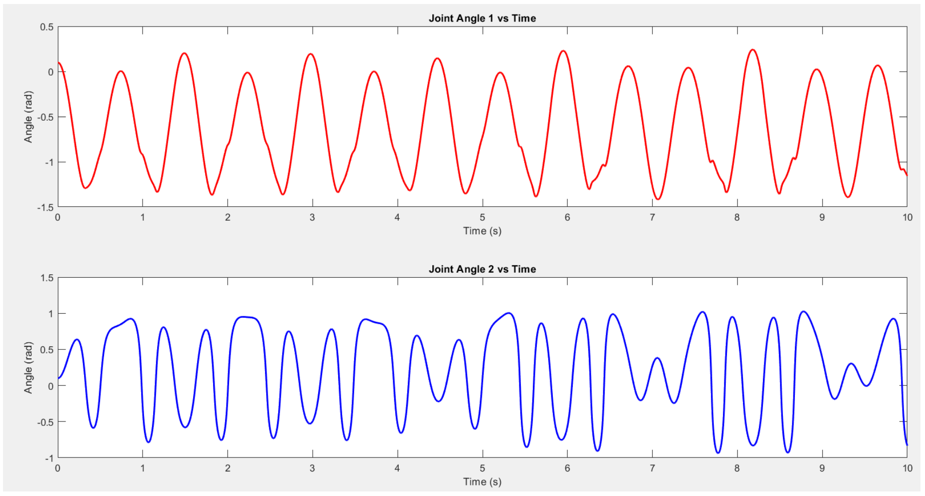


Figure 9: Kp = 50 Angles Plot

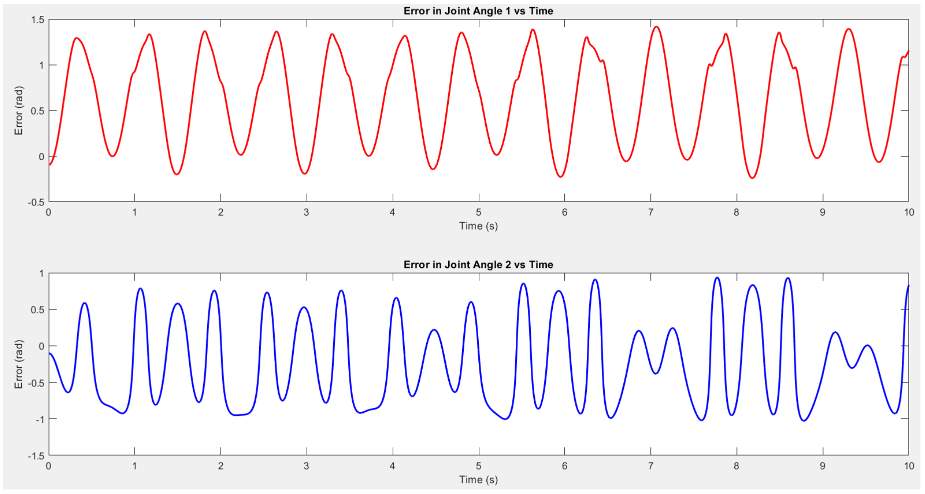


Figure 10: Kp = 50 Error Plot

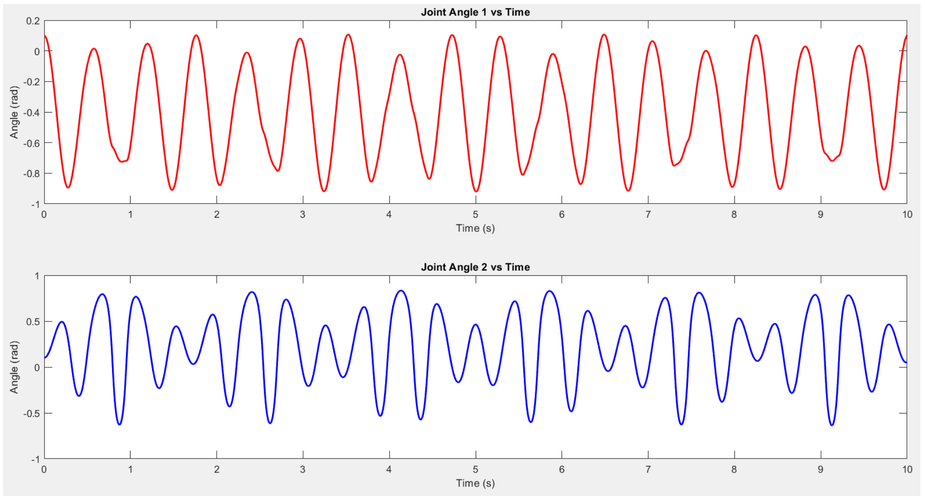


Figure 11: Kp = 100 Angles Plot

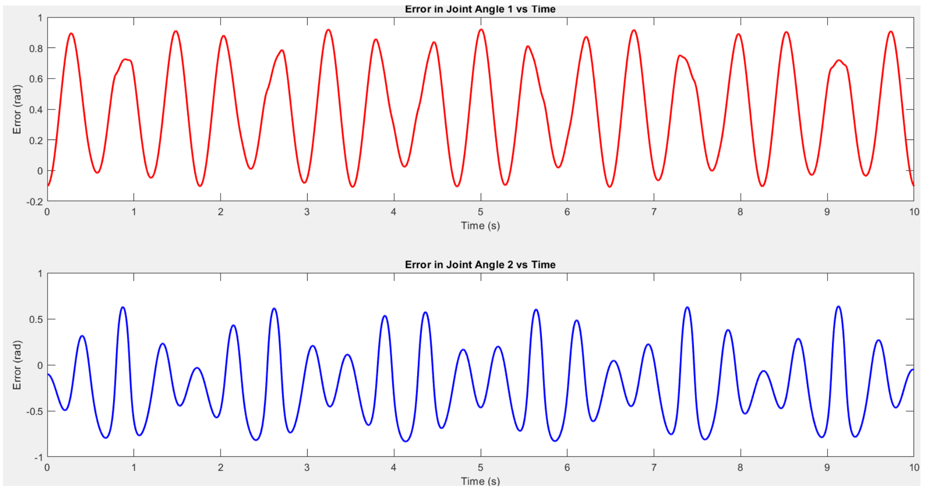


Figure 12: Kp = 100 Error Plot

Here we can observe that as K_p increases the error tradeoff is becoming less which will be useful in the next step for designing PI controller which trys to eliminate the steady state error.

7.2 Proportional Integrable Gain

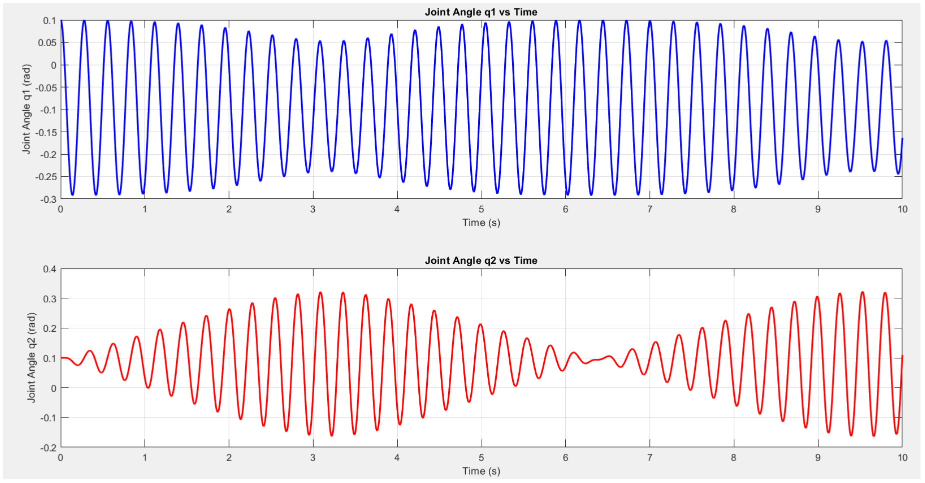


Figure 13: Kp = 500 Ki = 0 Angles Plot

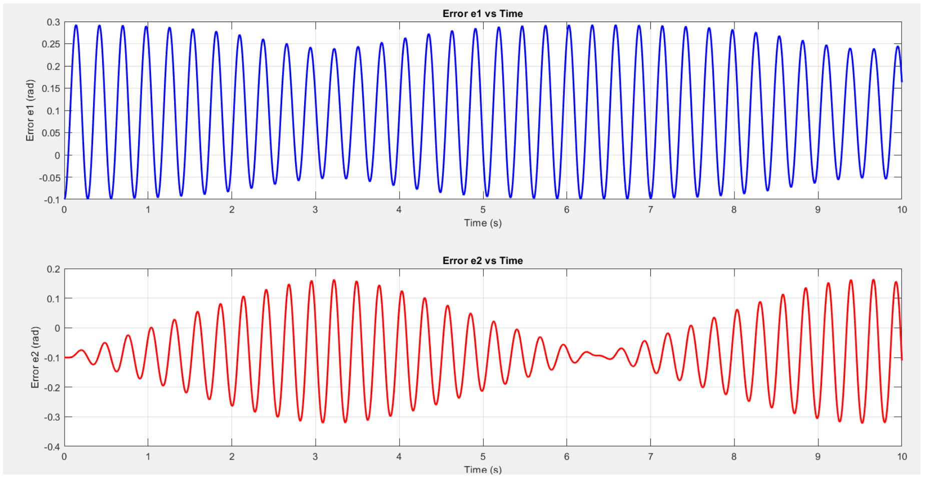


Figure 14: Kp = 500 Ki = 0 Error Plot

We can observe a good plot so lets go with $K_p = 500$ and then change value of K_i and check the results.

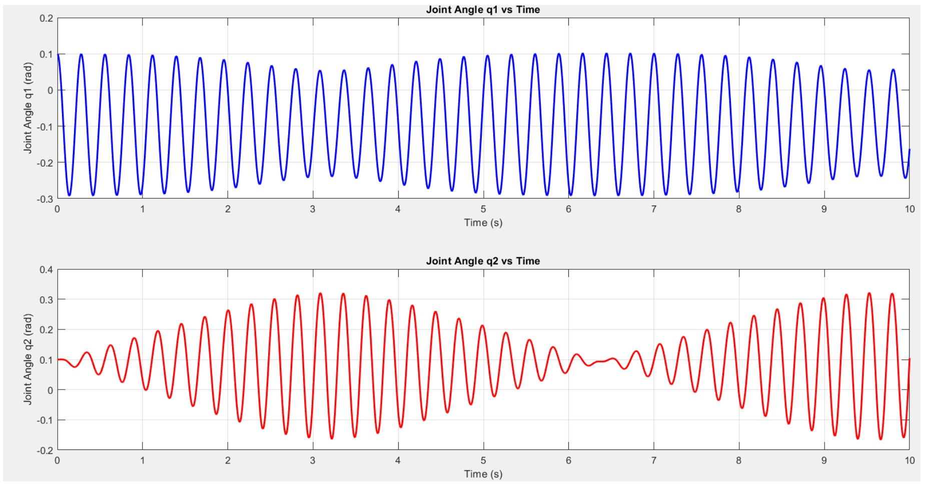


Figure 15: Kp = 500 Ki = 1 Angles Plot

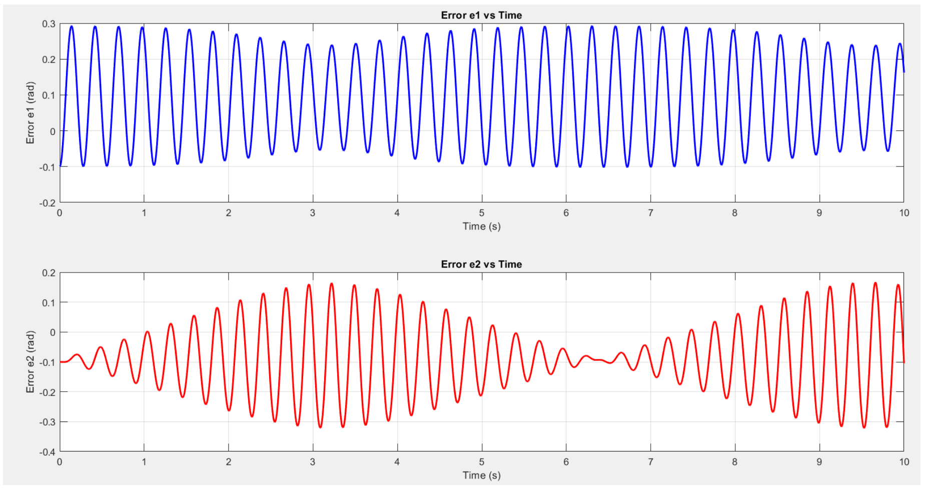


Figure 16: Kp = 500 Ki = 1 Error Plot

We werent able to observe any good change so lets keep increasing.

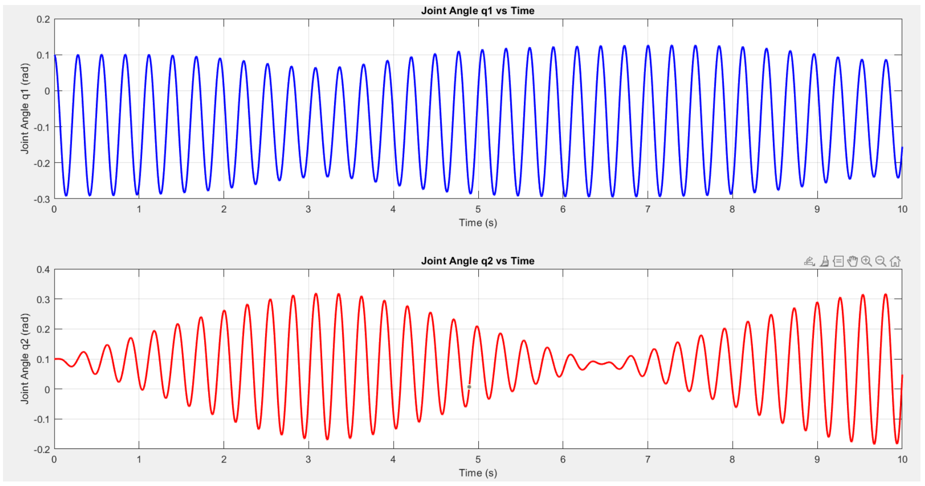


Figure 17: Kp = 500 Ki = 10 Angles Plot

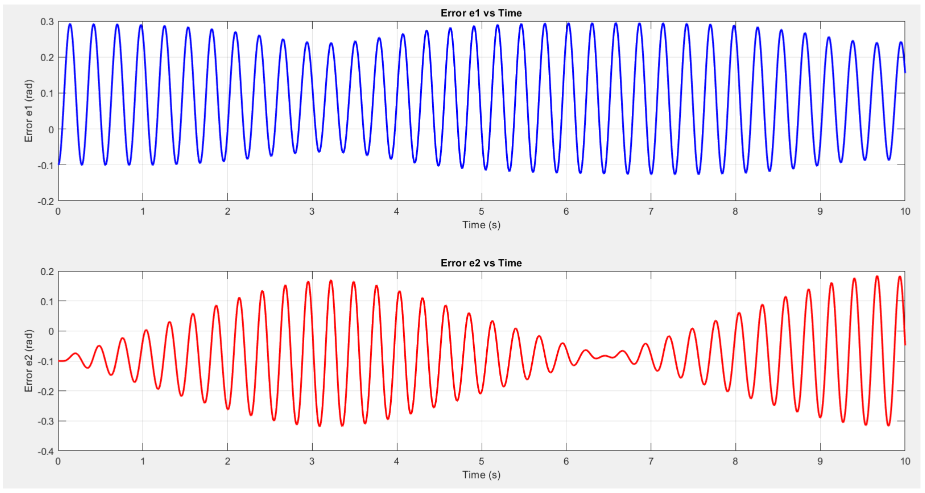


Figure 18: Kp = 500 Ki = 10 Error Plot

We can observe slight decrement in amplitude of oscilations

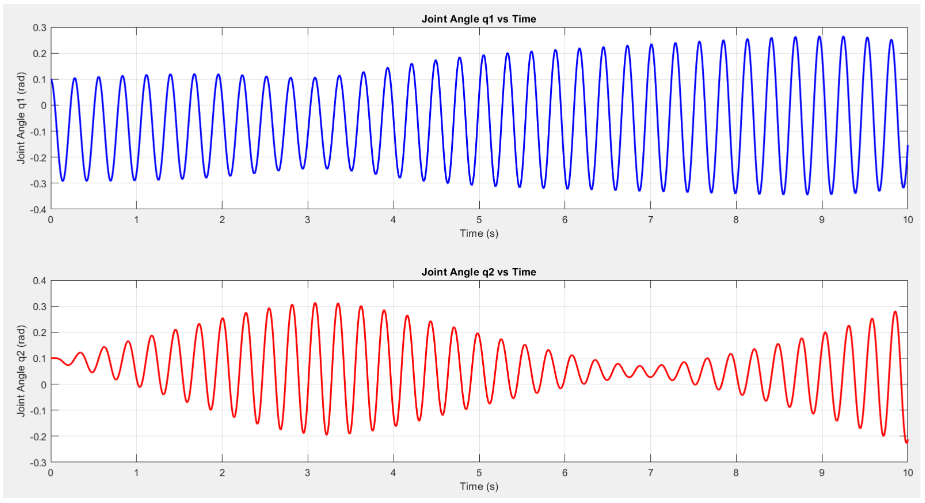


Figure 19: Kp = 500 Ki = 50 Angles Plot

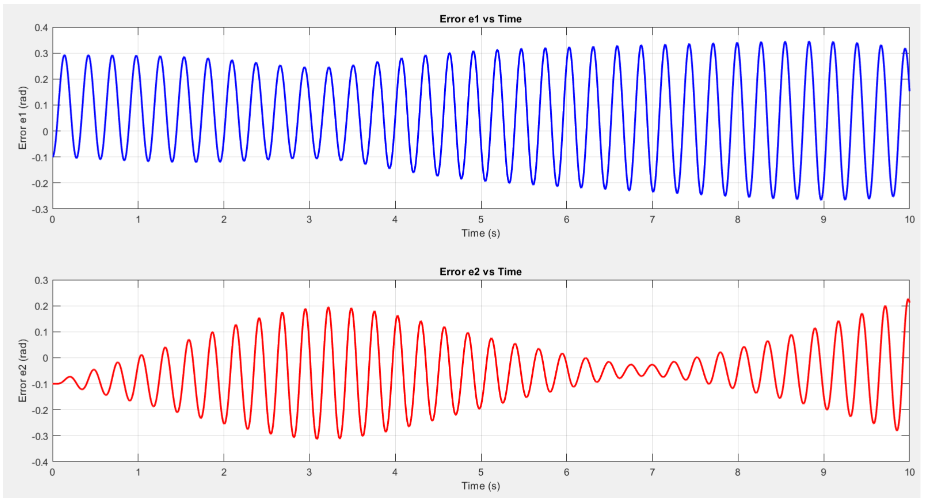


Figure 20: Kp = 500 Ki = 50 Error Plot

We can observe that the amplitude of oscillations increasing that means the angle is more deviating for required output. Hence $K_i = 10$ is a good value to start off.

7.3 Proportional Differentiable Gain

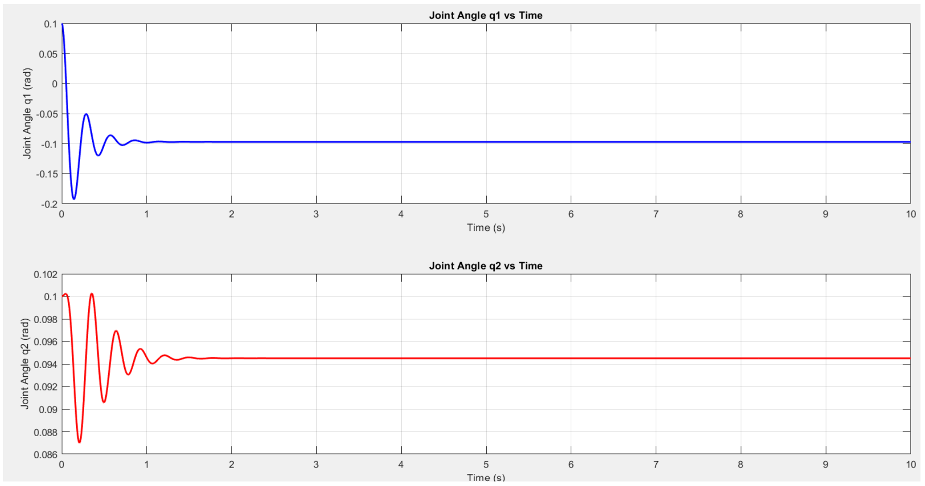


Figure 21: Kp = 500 Kd = 10 Angles Plot

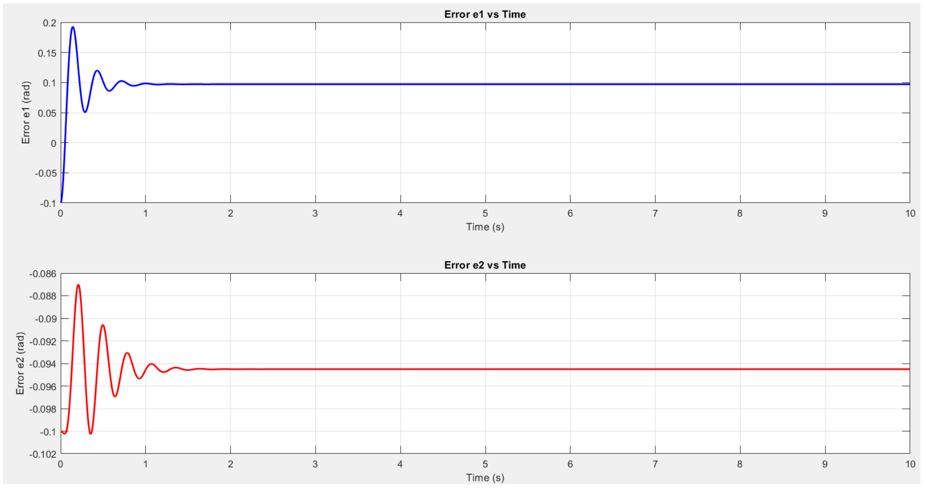


Figure 22: Kp = 500 Kd = 10 Error Plot

We can observe that steady state is achieved very fast means we are good to go for minimising the time taken to get to Steady State.

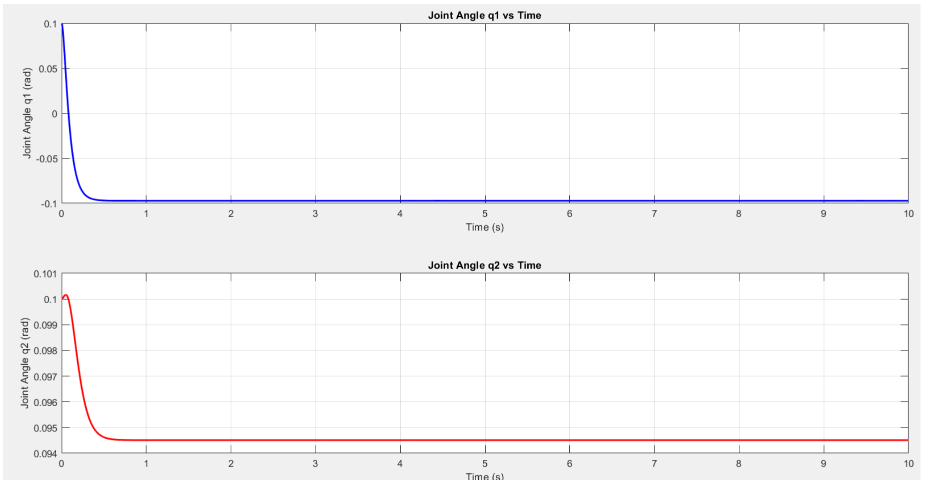


Figure 23: Kp = 500 Kd = 50 Angles Plot

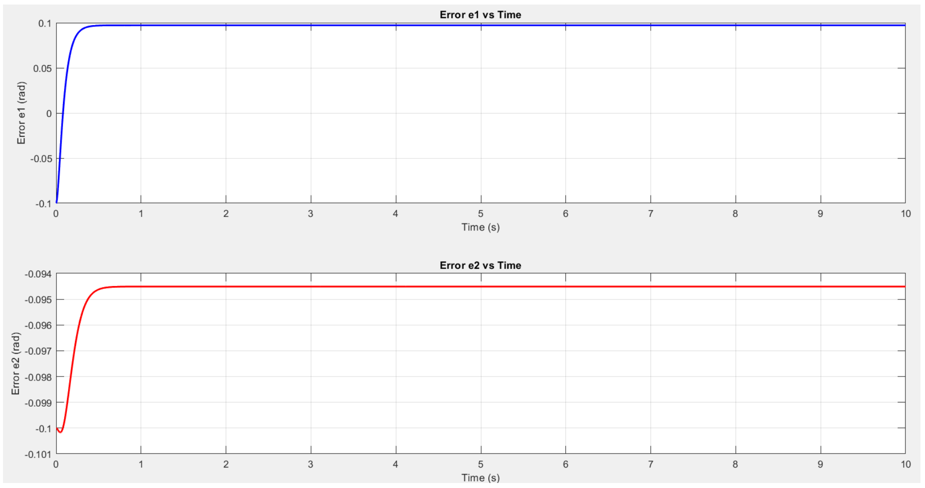


Figure 24: Kp = 500 Kd = 50 Error Plot

In these plots we can observe that the Steady State is achieved faster than the previous case so we are in right direction.

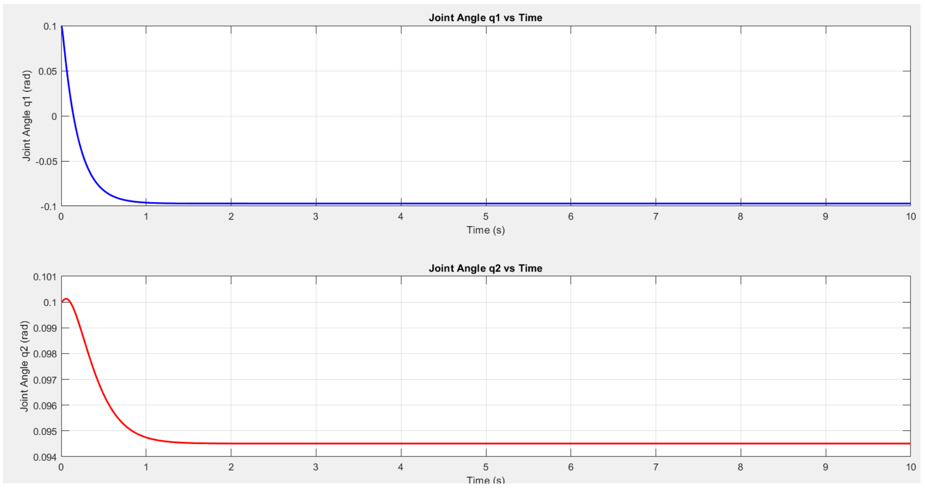


Figure 25: Kp = 500 Kd = 100 Angles Plot

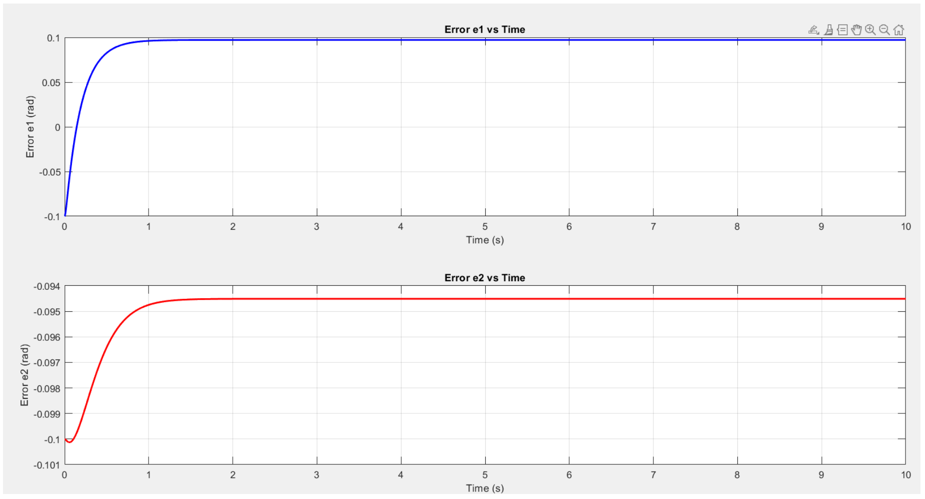


Figure 26: Kp = 500 Kd = 100 Error Plot

We can observe that there isnt much change but if we increase more it starts to go low first then high in error. So $K_d = 50$ to $K_d = 100$ are good ones.

7.4 Proportional Integrable Differential Gain (PID)

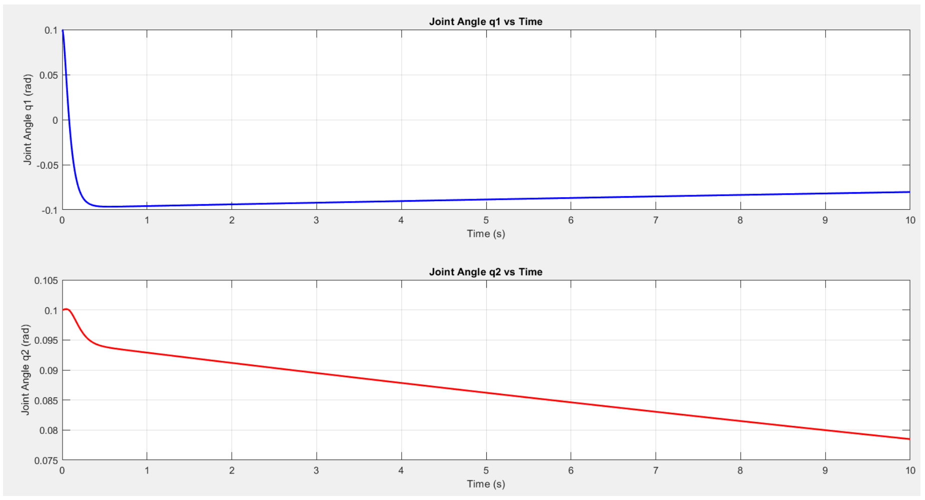


Figure 27: Kp = 500 Kd = 50 Ki = 10 Angles Plot

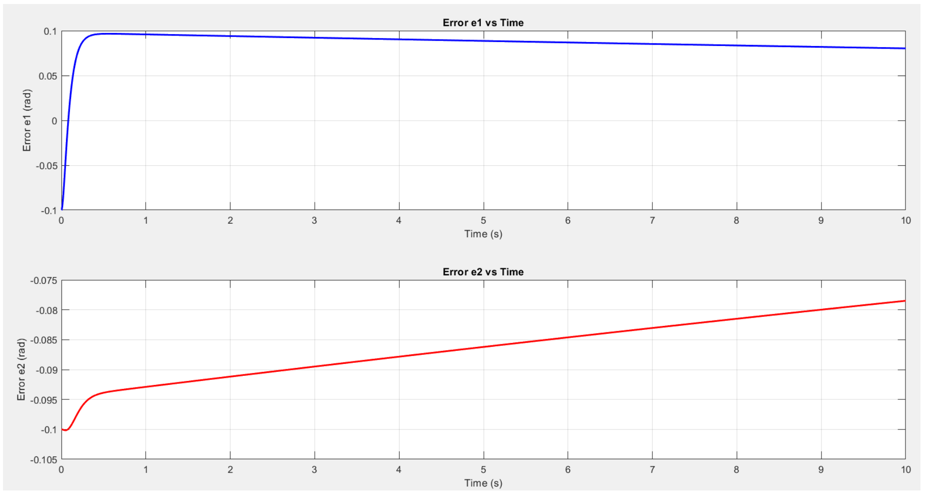


Figure 28: Kp = 500 Kd = 50 Ki = 10 Error Plot

We can observe a very different plot but error 2 seems decreasing so lets try some different values of gains and see outputs.

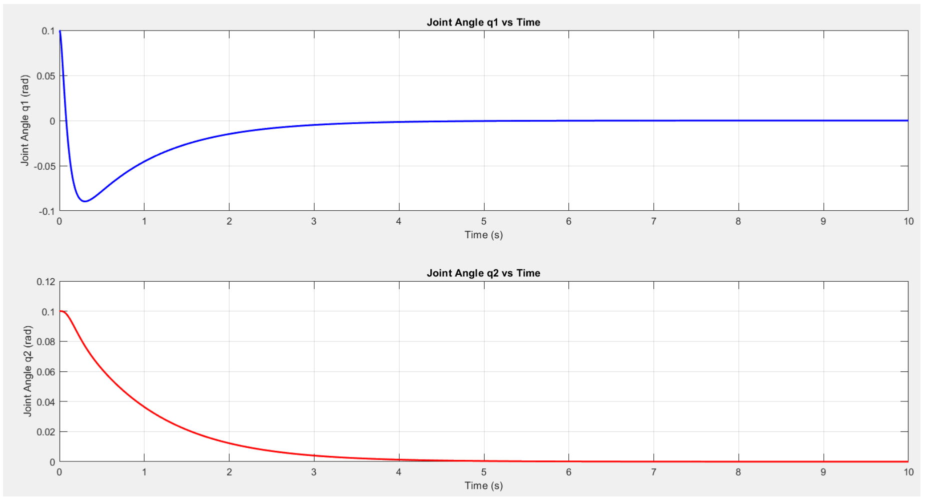


Figure 29: Kp = 500 Kd = 50 Ki = 500 Angles Plot

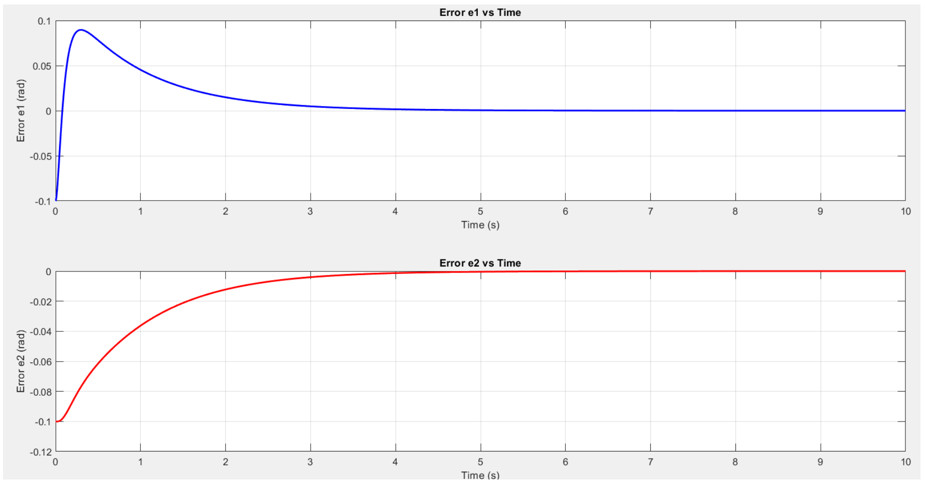


Figure 30: Kp = 500 Kd = 50 Ki = 500 Error Plot

We can observe that the steady state error is becoming 0 but as we can see this is at a very high integral gain.

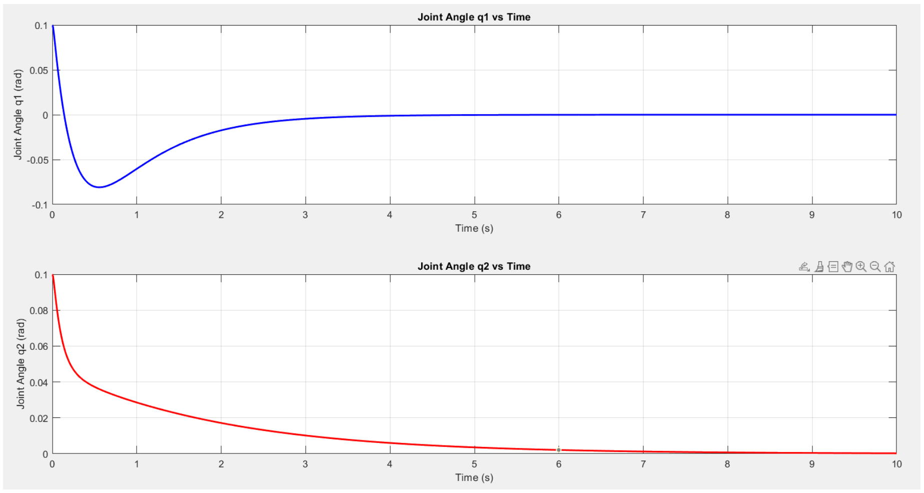


Figure 31: Kp1 = 500 Kd1 = 100 Ki1 = 500 Kp2 = 1000 Kd2 = 100 Ki2 = 500 Angles Plot

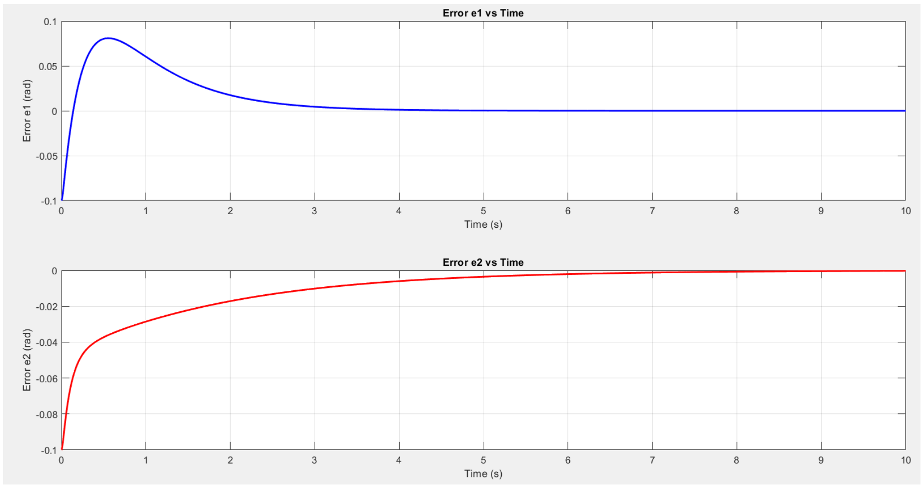
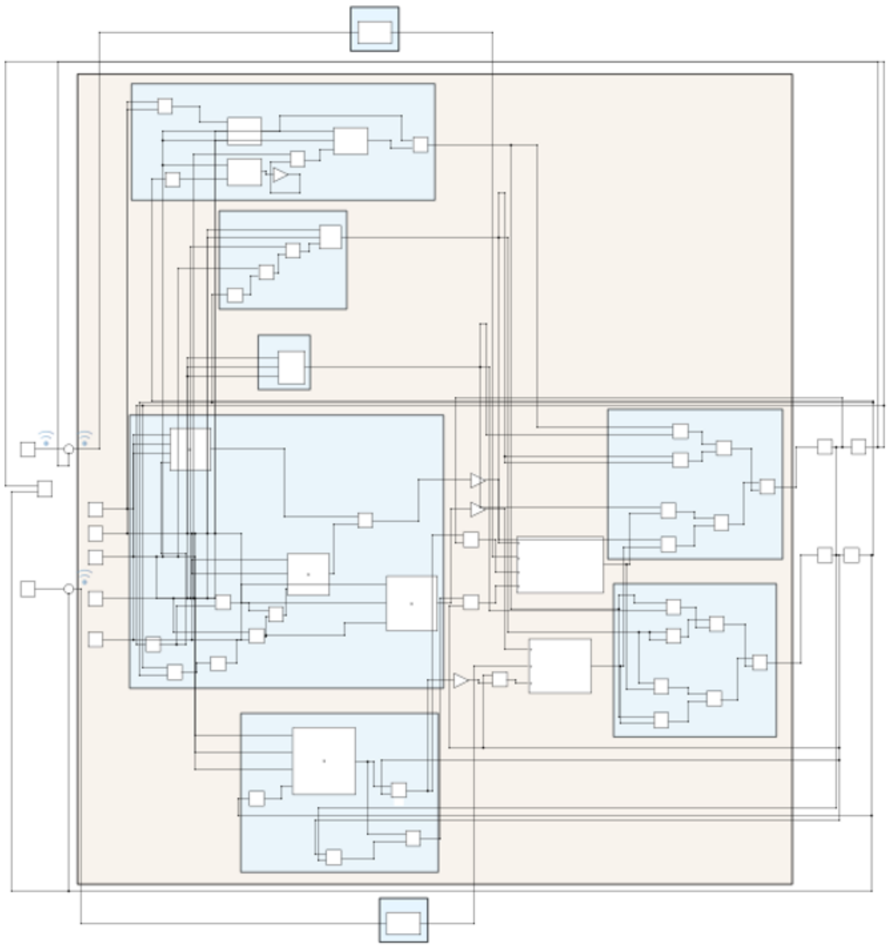


Figure 32: Kp1 = 500 Kd1 = 100 Ki1 = 500 Kp2 = 1000 Kd2 = 100 Ki2 = 500 Error Plot

Hence based on this we can observe a good plots at Kp1 = 500 Kd1 = 100 Ki1 = 500 Kp2 = 1000 Kd2 = 100 Ki2 = 500. But this can be finetuned more using Simulink and other automated tools and more practice.

8 Simulink Model

Reference step signals are used to compute errors in q_1, q_2 and feed it as input to the controllers.



8.1 P Controller

Starting from 1, we increase the value of K_p to obtain a faster transient response.

- $K_{p1} = K_{p2} = 1$: The response is very slow.

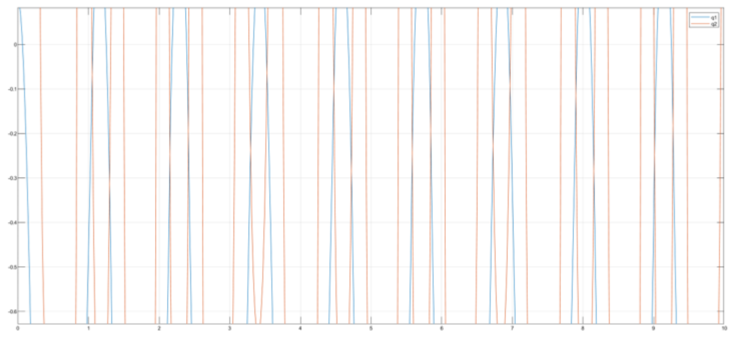


Figure 33: $K_{p1} = K_{p2} = 1$

- $K_{p1} = K_{p2} = 10$: The response is unstable and unpredictable. It may be problematic using this value.

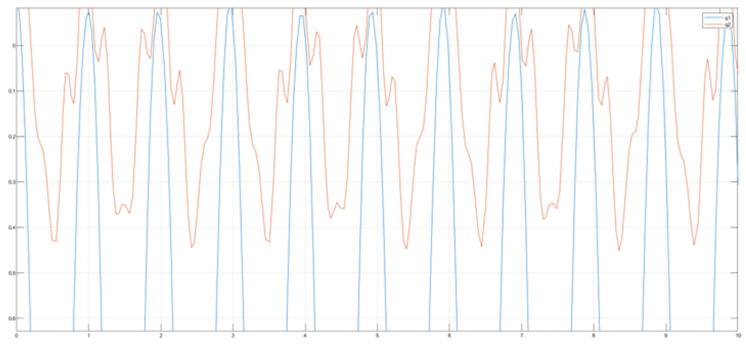


Figure 34: $K_{p1} = K_{p2} = 10$

- $K_{p1} = K_{p2} = 50$: Response is reasonably fast and stable. Steady state error is also minimized.

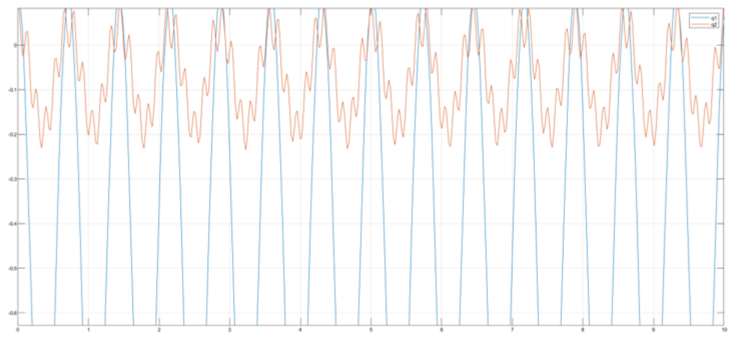


Figure 35: $K_{p1} = K_{p2} = 50$

- $K_{p1} = K_{p2} = 100$: The response is way too fast, not easy to deal with.

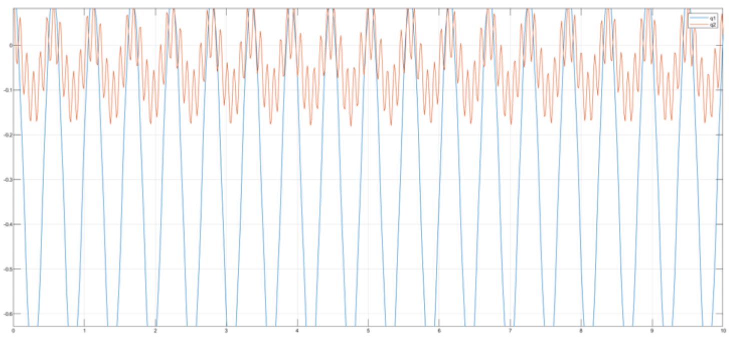


Figure 36: $K_{p1} = K_{p2} = 100$

Therefore, we prefer using $K_p = 50$.

8.2 PI Controller

Using the ideal value of K_p, we try to vary K_i to find the ideal values such that steady state error is minimized and the transient response is reasonably fast.

- $K_{i1} = K_{i2} = 1$: There is no significant effect of the integrator here.

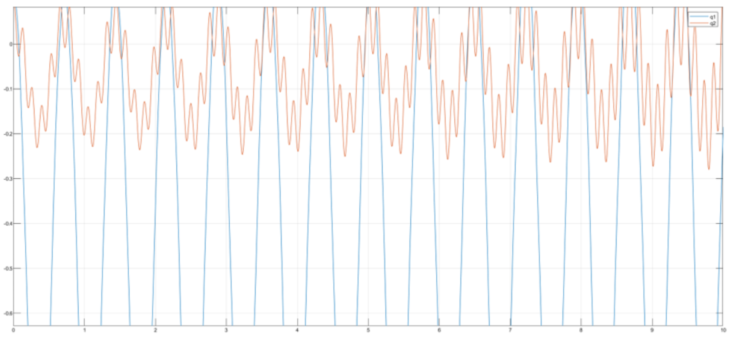


Figure 37: $K_{i1} = K_{i2} = 1$

- $K_{i1} = K_{i2} = 10$: The steady state is achieved for both but the increasing angle of oscillation makes it unstable.

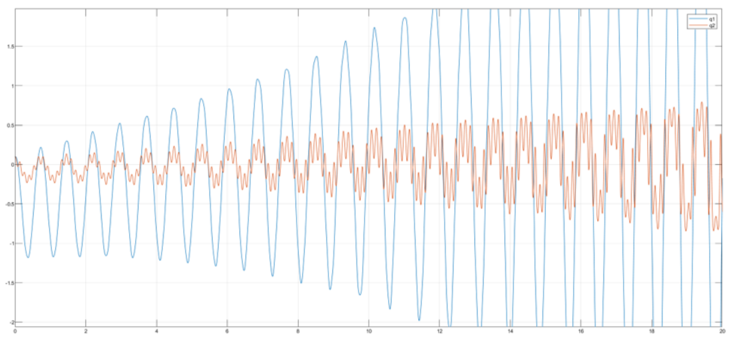


Figure 38: $K_{i1} = K_{i2} = 10$

- $K_{i1} = K_{i2} = 50$: Increasing K_i further makes the system highly unstable.

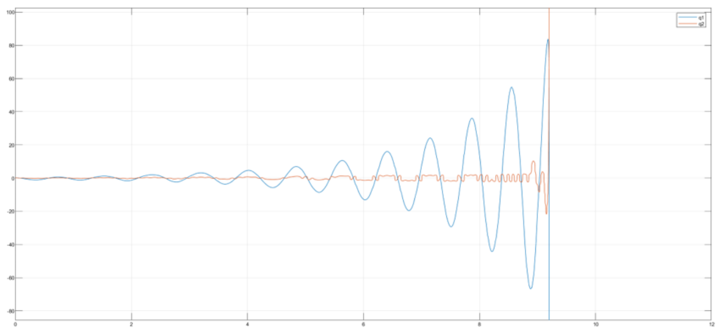


Figure 39: $K_{i1} = K_{i2} = 50$

To stabilize this, we can try increasing K_p:

$$K_{p1} = K_{p2} = 100, \quad K_{i1} = K_{i2} = 50$$

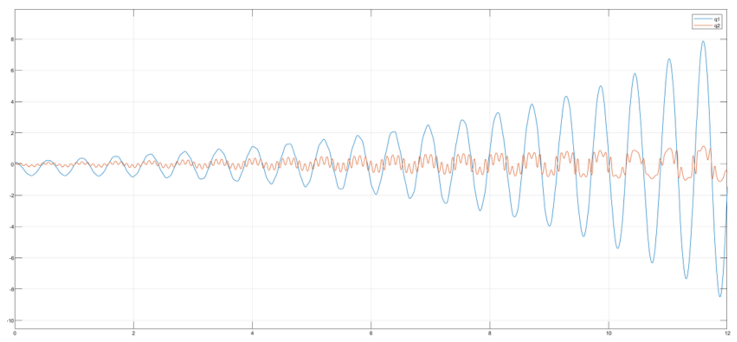


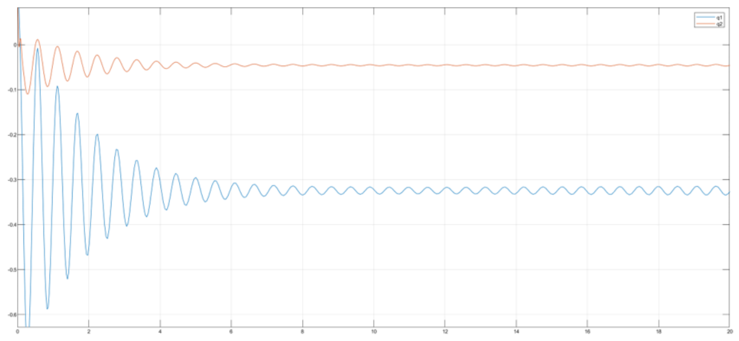
Figure 40: K_p:

$$K_{p1} = K_{p2} = 100, \quad K_{i1} = K_{i2} = 50$$

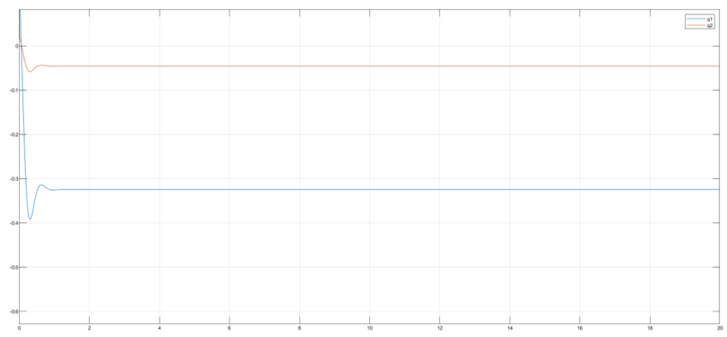
For these values, the error is minimized and the system is stable.

8.3 PD Controller

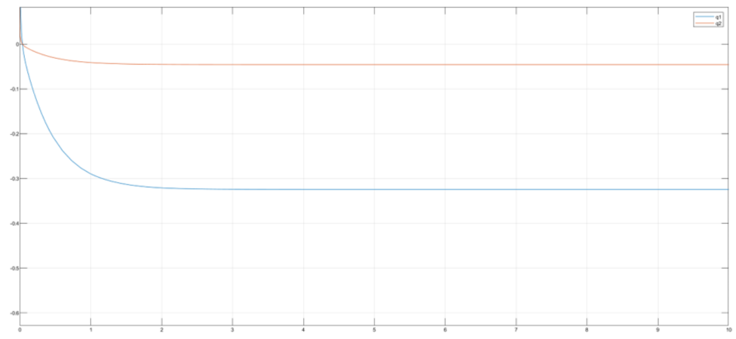
- $K_{p1} = K_{p2} = 100$, $K_{d1} = K_{d2} = 1$: The oscillations make the system problematic to work on.



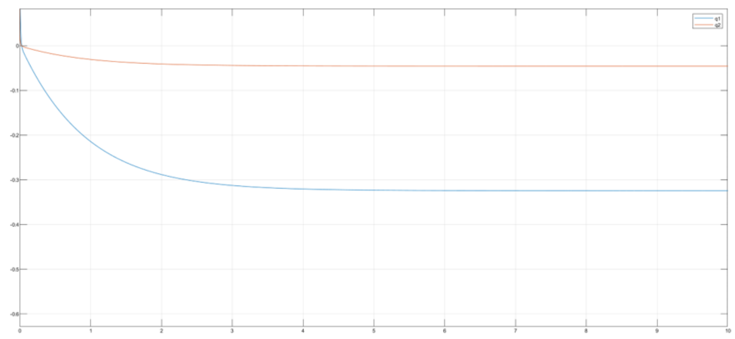
- $K_{d1} = K_{d2} = 10$: The system is still unstable.



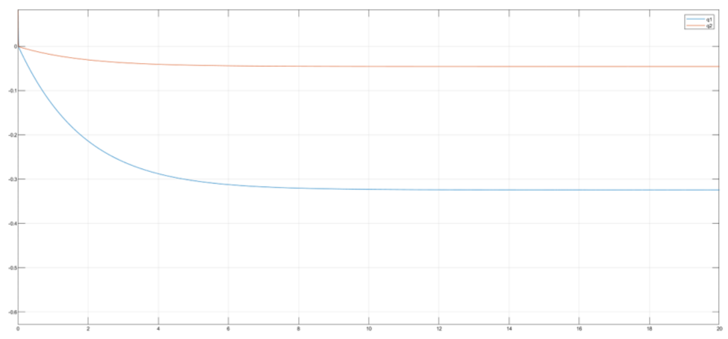
- $K_{d1} = K_{d2} = 50$: Oscillations are minimized almost completely, making the curve smooth.



- $K_{d1} = K_{d2} = 100$: Higher values of K_d reach steady state but the change is gradual.

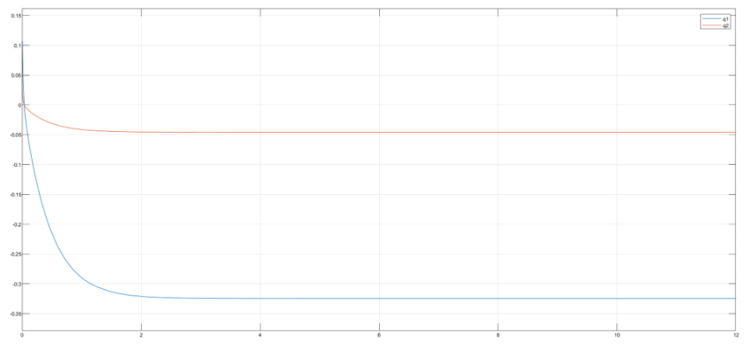


- $K_{d1} = K_{d2} = 200$



So, we prefer using $K_d = 50$. For a faster transient response, we increase K_p:

$$K_{d1} = K_{d2} = 50, \quad K_{p1} = K_{p2} = 100$$

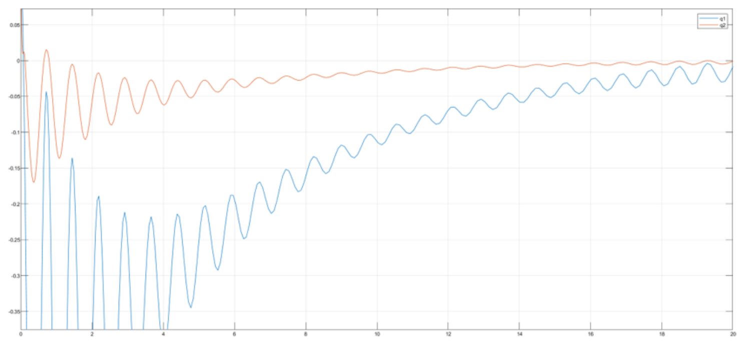


The steady state error is minimized further (observing the scale).

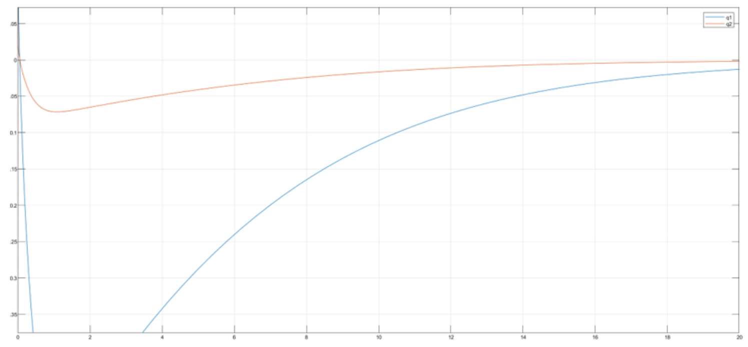
8.4 PID Controller

Let's start with:

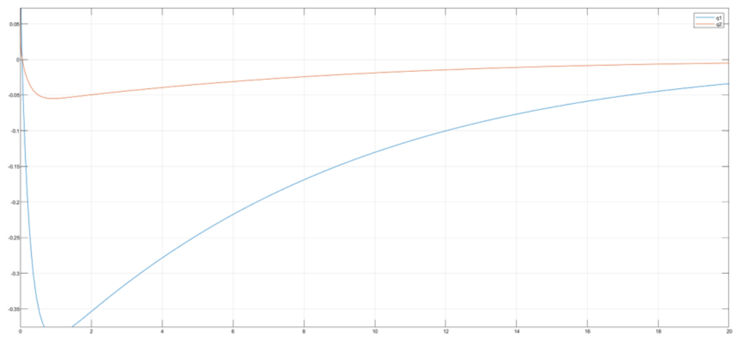
$$K_{p1} = K_{p2} = 50, \quad K_{d1} = K_{d2} = 1, \quad K_{i1} = K_{i2} = 10$$



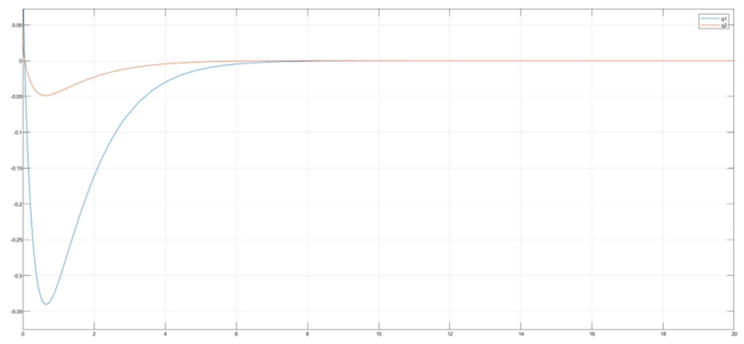
To reduce oscillations, we increase K_d. Let $K_d = 25$.



To reach the steady state faster, we increase K_p. Let $K_p = 75$.



To reduce the steady state error, we increase K_i. Let $K_i = 50$. Increasing K_d any further makes the system unstable.



Therefore, to get the desired transient response for a PID controller, we use:

$$K_{i1}, K_{i2} = 50, \quad K_{p1}, K_{p2} = 75, \quad K_{d1}, K_{d2} = 25$$

9 Conclusion

9.1 Effect on Damping

High K_D (Pros):

- High K_D values are effective at damping oscillations and reducing the amplitude of oscillatory behavior in the system, giving a smoother and more stable control response.
- The robotic arm is less prone to oscillations and vibrations, which helps when precise and controlled movements are needed.

High K_D (Cons):

- Excessive K_D can lead to over-damping, where the system becomes very slow to respond to changes in the desired positions. This results in slower movement and reduced overall performance.
- Over-damping can also lead to a lack of responsiveness, which is problematic when rapid adjustments or dynamic movements are needed.

Low K_D (Pros):

- Low K_D values maintain a more responsive system with less damping, allowing the robotic arm to move more quickly in response to changes. This is helpful in scenarios where we prioritize speed over stability.
- The system can be more agile and better suited for tasks that require rapid and precise movements.

Low K_D (Cons):

- Low K_D values may not effectively dampen oscillations, leading to a system that is more prone to overshoot and excessive oscillations. This can result in less stability and accuracy in control.

9.2 Reduction of Overshoot

High K_D (Pros):

- Increasing K_D can significantly reduce overshooting and oscillations, resulting in a smoother and more stable control response. This is beneficial when precise and controlled movements without oscillations are needed.

High K_D (Cons):

- If K_D is set too high, it can lead to over-damping, which could slow down the settling process and potentially make the system unresponsive. This could affect the system's ability to reach the desired position on time.

Low K_D (Pros):

- Low K_D values might result in a faster initial response but may allow some overshoot and oscillations to persist. This is very helpful if a quicker initial movement is needed and some overshoot is tolerable.

Low K_D (Cons):

- Very low K_D values may not adequately mitigate overshoot and oscillations, which can affect the precision of the robotic arm's movement. This might be problematic in applications where precise positioning is critical.

9.3 Effect on Response Time

High K_D (Pros):

- Higher K_D values lead to faster settling times, reducing the time it takes for the system to reach a stable state. This is helpful when rapid and accurate positioning is required.

High K_D (Cons):

- Excessive K_D can introduce overshoot or over-damping, causing the system to take longer to reach the desired position in some cases. This becomes a problem if quick, dynamic responses from the arm are needed.

Low K_D (Pros):

- Low K_D values maintain a more rapid initial response but may extend the settling time slightly. This can be helpful when the initial speed of the response is prioritized.

Low K_D (Cons):

- Very low K_D values may allow oscillations to persist and slow down the overall settling process. This can be problematic when precision and stability are crucial.

9.4 Stability

High K_D (Pros):

- High K_D values can contribute to a highly stable system when appropriately tuned, particularly in scenarios where precision and smoothness are crucial. The system is less likely to exhibit overshoot or oscillations.

High K_D (Cons):

- Excessive K_D values can lead to an over-damped response, resulting in slow settling and potentially causing instability. This may affect the system's ability to respond to dynamic changes (arm sensitivity is affected).

Low K_D (Pros):

- Low K_D values provide a more responsive system and reduce the risk of over-damping or instability. The robotic arm may be better suited for dynamic tasks.

Low K_D (Cons):

- Very low K_D values may not effectively dampen oscillations, making the system less stable and prone to overshoot. This can result in instability and unpredictable behavior.

9.5 Conclusion

In practice, tuning the derivative gain (K_D) involves finding the right balance between reducing overshoot, damping oscillations, and maintaining responsiveness based on the specific characteristics and requirements of our robotic arm control system. The optimal K_D value will depend on the following factors: mechanical properties of the system, desired performance, and the specific tasks it needs to perform.

We can observe that that for this 2-Link manipulator, PID controllers suits the best as it is the combination of advantages of all 3 controllers we discussed i.e. P, PI, PD controllers and each has its own benefit and cons. Only P controller gives a steady state error but also takes long time to reach there whereas the PI controller tries to eliminate the Steady State error but at the same time it introduces high starting overshoot, sensitivity to controller gains and sluggish response to sudden disturbances. In PD controller we can observe that the steady state is obtained quickly but there is still some error in the system. Hence with combinations of advantages and disadvantages of P,PI,PD controllers we have a PID controller which does controlling precisely and also quickly enabling us to be having more practical usage and a little less sensitive to noises. In practical we dont use high gain integral as it is very much sensitive to noise which is common in a practical use.

PID Controller is widely used in this case according to our team opinion.