M23EC5.202 : Systems Thinking
Mini Project 1
Dynamic Analysis of 2-Link Manipulator
Group : Zenith

Adithi Samudrala
2022102065
International Institute of Information Technology Hyderabad
adithi.samudrala@students.iiit.ac.in

Aniruth Suresh
2022102055
International Institute of Information Technology Hyderabad
aniruth.suresh@students.iiit.ac.in

Aryan Garg
2022102074
International Institute of Information Technology Hyderabad
aryan.garg@students.iiit.ac.in

Asrith Reddy Patolla
2022102050
International Institute of Information Technology Hyderabad
asrith.reddy@students.iiit.ac.in

Autrio Das
2022112007
International Institute of Information Technology Hyderabad
autrio.das@research.iiit.ac.in

Pearl Shah
2022102073
International Institute of Information Technology Hyderabad
pearl.shah@students.iiit.ac.in

Priya Mandot
2022102053
International Institute of Information Technology Hyderabad
priya.mandot@students.iiit.ac.in

Raagav Ramakrishnan
2022102018
International Institute of Information Technology Hyderabad
raagav.ramakrishnan@students.iiit.ac.in

***Abstract*—In this paper we propose and analyse a proportional-integral-derivarive (PID) controller for a classic 2-link robotic manipulator. First we put forth equations of motion of the two link robotic manipulator. We focus mainly on control of the robot manipulator to get the desired position using the computed torque control method. Control simulations is performed using MATLAB. Several computer simulations are used to verify the performance of the controller. Specifically we are interested in simulating the system performance under proportional-derivative (PD), proportional-integral (PI) and proportional-integral-derivarive (PID) control for differing values of the respective control gains (k_p, k_i, k_d). Our findings and analysis is also verified using simulations run on MATLAB's SIMULINK.**

I. Introduction

Robotic manipulators are a major component in the manufacturing industry. They are used for many reasons including speed, accuracy, and repeatability. In robotics, one of the most difficult tasks is to perform a precise and fast movement of a robotic arm In this paper, we propose a PID controller for a double pendulum system. We derive the equations of motion for a two-link robot manipulator based on Jacobian Formulation. After deriving the equation of motion, control simulation is represented using MATLAB.

II. System Dynamics Of 2-Link Manipulator

A Two-Link manipulator is an example of a double pendulum –an inherently chaotic system that is extremely sensitive to intitial conditions. This manipulator, shown in Fig. 1 example studied in introductory robotic courses.

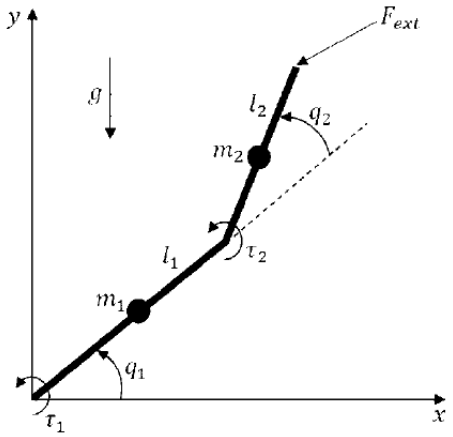


Fig. 1. Two-Link Robotic Manipulator

To analyse the system, we determine the Mass matrix, Coriolis matrix and the Gravitational matrix. This is derived by first determining the forward kinematics, then the Jacobian. KE and PE are used to determine the mass and gravitational matrices.

The joint Torques are given by the equation:

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

which when expanded gives us :

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix}M_{11} & M_{12}\\ M_{21} & M_{22}\end{bmatrix}\begin{bmatrix}\ddot{q}_1\\ \ddot{q}_2\end{bmatrix} + \begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix} + \begin{bmatrix}G_{11}\\ G_{12}\end{bmatrix}$$

The equations for the x-position and the y-position of m_1 are given by

$$x_1 = l_1\cos(q_1)$$
$$y_1 = l_1\sin(q_1)$$

Similarly, the equations for the x-position and the y-position of m_2 are given by

$$x_2 = l_1\cos(q_1) + l_2\cos(q_2+q_1)$$
$$y_2 = l_1\sin(q_1) + l_2\sin(q_2+q_1)$$

let r_1 and r_2 be the distances from the joints to the center of mass of the links. Thus COM coordinates for the two links will be:

$$c_1 = r_1\cos(q_1), r_1\sin(q_1)$$
$$c_2 = r_1\cos(q_1) + r_2\cos(q_1+q_2), r_1\sin(q_1) + r_2\sin(q_1+q_2)$$

Thus the Forward kinematic matrix for the second link system will be:

$$\begin{bmatrix}r_1\cos(q_1) + r_2\cos(q_1+q_2)\\ r_1\sin(q_1) + r_2\sin(q_1+q_2)\end{bmatrix}$$

Thus the Jacobian is derived using the formulation

$$\dot{X} = J\dot{q}$$
$$\implies J_{v_2}\begin{bmatrix}-l_1\sin(q_1) + r_2\sin(q_1+q_2) & r_2\sin(q_1+q_2)\\ l_1\cos(q_1) + r_2\cos(q_1+q_2) & r_2\cos(q_1+q_2)\end{bmatrix}$$

KE for link 1

$$KE_1 = \frac{1}{2}m_1r_1\dot{q}_1^2 + \frac{1}{2}I_1\dot{q}_1^2$$
$$\implies \frac{1}{2}\begin{bmatrix}\dot{q}_1 & \dot{q}_2\end{bmatrix}\begin{bmatrix}m_1r_1^2 + I_1 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix}$$

KE for link 2 using the Jacobian

$$KE_2 = \frac{1}{2}\begin{bmatrix}\dot{q}_1 & \dot{q}_2\end{bmatrix}\left[J_{v_2}^T J_{v_2} m_2 + I_2\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\right]\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix}$$

expanding these terms using the Location of the COM being $r = \frac{l}{2}$ we get

$$M_{11} = (m_1+m_2)l_1^2 + m_2l_2(l_2 + 2l_1\cos(q_2)),$$
$$M_{12} = m_2l_2(l_2 + l_1\cos(q_2)),$$
$$M_{22} = m_2l_2^2$$

Finding the coriolis matrix involves the calculations:

$$C\dot{q} = \begin{bmatrix}c_{111}\dot{q}_1 + c_{121}\dot{q}_2 & c_{211}\dot{q}_1 + c_{221}\dot{q}_2\\ c_{112}\dot{q}_1 + c_{122}\dot{q}_2 & c_{212}\dot{q}_1 + c_{221}\dot{q}_2\end{bmatrix}$$

where

$$c_{ijk} = \frac{1}{2}\left[\frac{\partial M_{ij}}{\partial\dot{q}_i} + \frac{\partial M_{jk}}{\partial\dot{q}_j} - \frac{\partial M_{ki}}{\partial\dot{q}_k}\right]$$

Some of the coriolis terms evaluate to be zero leaving us with the final coriolis matrix as:

$$\mathbf{C} = \begin{bmatrix}-m_2l_1l_2\sin(q_2)\dot{q}_2 & -m_2l_1l_2\sin(q_2)(\dot{q}_1+\dot{q}_2)\\ 0 & m_2l_1l_2\sin(q_2)\dot{q}_2\end{bmatrix}$$

To evaluate the G matrix we use the potential energy from the formula

$$PE = m_1gh_1 + m_2gh_2$$

to get the G matrix we take the derivative of these terms wrt q_1 and q_2.

$$\frac{\partial PE_1}{\partial q_1} = m_1gl_1\cos(q_1) + m_2g((l_1\cos(q_1) + r_2\cos(q_1+q_2)))$$
$$\frac{\partial PE_1}{\partial q_2} = m_2gl_2\cos(q_1+q_2)$$

resulting in

$$\mathbf{G} = \begin{bmatrix}m_1l_1g\cos(q_1) + m_2g(l_2\cos(q_1+q_2) + l_1\cos(q_1))\\ m_2gl_2\cos(q_1+q_2)\end{bmatrix}$$

In our analysis we consider

$$m_1 = 10kg \quad m_2 = 5kg$$
$$l_1 = 0.2m \quad l_2 = 0.1m$$
$$g = 9.81$$

The joint angles are initially at positions

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

The objective during the period of analysis is to get the angles to the final positions

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

III. Design Of PID Control

A general analysis that can be performed on the system in section II is under PID. PI and PD control analysis can be thought of as special cases of PID control and thus will be tackled in later sections.

A. *PID Control*

The system described in section II may be analysed subject to PID control by solving for $\ddot{q}$ matrix

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

where

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

thus we have the system takng new inputs

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

where the actual physical torque inputs are

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

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.

$$u_1 = \int e_1 dt$$
$$u_2 = \int e_2 dt$$
$$u_3 = q_1 \quad u_4 = q_2$$
$$u_5 = \dot{q}_1 \quad u_6 = \dot{q}_2$$

This system of linear ODEs is solved by the `ode45` function in MATLAB `ode45` requires the formulation of the derivative of these variables. Thus writing

$$\dot{u}_1 = e_1 \quad \dot{u}_2 = e_2$$
$$\dot{u}_3 = \dot{q}_1 \quad \dot{u}_4 = \dot{q}_2$$
$$\dot{u}_5 = \ddot{u}_1 = \ddot{q}_1 = \phi(t,u_1,u_2,u_3,u_4,u_5,u_6)$$
$$\dot{u}_6 = \ddot{u}_2 = \ddot{q}_2 = \psi(t,u_1,u_2,u_3,u_4,u_5,u_6)$$

Listing 1. PIDmain.m

```
% Assign the given masses and length

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
x10= 0 ;
x20=0 ;
q10= 0.1 ;
q20 = 0.1 ;
q1dot0 =0 ;
q2dot0 =0 ;
t0 = 0 ;
tf = 10;

q1_fin = 0;
q2_fin = 0;


% Final q = 0
kp1 = 100 ; kd1 = 100 ; ki1 = 200;
kp2 = 300 ; kd2 = 200 ; ki2 = 200;


%time span
tspan = [t0 , tf];

%Initial conditions
IC = [x10 , x20, q10, q20 ,q1dot0,q2dot0];

options = odeset('RelTol', 1e-3, 'AbsTol',
    1e-6);

[time , state_values] =
    ode45(@(t,s)pid(t,s,q1_fin ,q2_fin
    ,m1,m2 ,l1,l2
    ,g,kp1,kp2,kd1,kd2,ki1,ki2), tspan,
    IC, options);


q1 = state_values(:,3);
q2 = state_values(:,4);


figure;
subplot(2, 1, 1);
plot(time, q1, 'r');
xlabel('Time (s)');
ylabel('q1 (rad)');
title('Joint Angles vs. Time (PID
    Control)');
```

```
subplot(2, 1, 2);
plot(time, q2, 'g');
xlabel('Time (s)');
ylabel('q2 (rad)');

e1 = q1_fin - q1;
e2 = q2_fin - q2;

figure;
subplot(2, 1, 1);
plot(time, e1, 'r');
xlabel('Time (s)');
ylabel('e1 (rad)');
sgtitle(' Error in Joint Angles vs. Time
    (PID Control)');

subplot(2, 1, 2);
plot(time, e2, 'g');
xlabel('Time (s)');
ylabel('e2 (rad)');
```

The `ode45` function solves the ODEs of the form $y' = f(t, y)$ 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.

Implementing the function `pid` to get the general function:

Listing 2. pid.m

```
function output = pid(~,s, q1f, q2f, m1,
    m2, l1, l2, g, kp1,kp2,kd1,kd2,ki1,ki2)

% Declare the State variables [ x1 ,x2 ,
    q1 , q2 , q1. , q2. ]
x1 = s(1);
x2 = s(2);
q1 = s(3);
q2 = s(4);
q1dot = s(5);
q2dot = s(6);

x1dot = q1f - q1;
x2dot = q2f - q2;


f1 = kp1 * x1dot - kd1 * q1dot + ki1 *x1;
f2 = kp2 * x2dot - kd2 * q2dot + ki2 *x2;


%Definitions
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
M = [M11, M12; M21, M22];

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

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

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

% Create the Gravitatinal matrix G
G = [G1;G2];

f = [f1;f2];
Tau_matrix = M * f;

q_dot_matrix = [q1dot ; q2dot];


q_double_dot_matrix = (inv(M))* ( -
    C*q_dot_matrix-G) + f; % Quicker than
    inv (M) ( same operation as inverse )

q_dd_1 = q_double_dot_matrix(1); % first
    row
q_dd_2 = q_double_dot_matrix(2); % Second
    row

output = [x1dot ; x2dot ; q1dot ; q2dot ;
    q_dd_1 ; q_dd_2];

end
```

Reference to plots IV-A

B. *PD Control*

Under PD control we need now only four new variables due to the absence of integral terms.

the system is decribed again as such

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

where

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

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

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

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

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

Modeling the PD controller as

$$f = K_p e - K_D \dot{e}$$

we get

$$\begin{aligned} f_1 &= K_{p_1} e(q_1) + K_{D_1} \dot{e}(q_1) \\ \implies f_1 &= K_{P_1}(q_{1f} - q_1) - K_{D_1} \dot{q}_1 \\ f_2 &= K_{p_2} e(q_2) + K_{D_2} \dot{e}(q_2) \\ \implies f_2 &= K_{P_2}(q_{2f} - q_2) - K_{D_2} \dot{q}_2 \end{aligned}$$

thus we have

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

Out 4 state variables are thus

$$\begin{aligned} u_1 = q_1 \quad u_2 = q_2 \\ u_3 = \dot{u}_1 = \dot{q}_1 \\ u_4 = \dot{u}_2 = \dot{q}_2 \end{aligned}$$

`ode45` requires the formulation of the derivative of these variables. Thus writing

$$\begin{aligned} \dot{u}_1 = \dot{q}_1 \quad \dot{u}_2 = \dot{q}_2 \\ \dot{u}_3 = \dot{u}_1 = \ddot{q}_1 = \phi(t, u_1, u_2, u_3, u_4) \\ \dot{u}_4 = \dot{u}_2 = \ddot{q}_2 = \psi(t, u_1, u_2, u_3, u_4) \end{aligned}$$

Solving through MATLAB:

Listing 3. PDmain.m

```
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

q10 = 0.1;
q20 = 0.1;
q1dot0 = 0;
q2dot0 = 0;
t0 = 0;
tf = 10;

q1_fin = 0;
q2_fin = 0;

kp1 = 100;
kd1 = 100;
kp2 = 300;
kd2 = 200;

tspan = [t0 tf];

IC =[q10, q20, q1dot0, q2dot0];

options = odeset('RelTol', 1e-3, 'AbsTol',
    1e-6);

[time, state_values] = ode45(@(t, s) pd(t,
    s, m1, m2, l1, l2, g, kp1, kd1, kp2,
    kd2,q1_fin, q2_fin),tspan,IC, options);

q1 = state_values(:, 1);
q2 = state_values(:, 2);

figure;
subplot(2, 1, 1);
plot(time, q1, 'r');
xlabel('Time (s)');
ylabel('q1 (rad)');
sgtitle('Joint Angles vs. Time (PD
    Control)');

subplot(2, 1, 2);
plot(time, q2, 'g');
xlabel('Time (s)');
ylabel('q2 (rad)');

e1 = q1_fin - q1;
e2 = q2_fin - q2;

figure;
subplot(2, 1, 1);
plot(time, e1, 'r');
xlabel('Time (s)');
ylabel('e1 (rad)');
sgtitle('Error in Joint Angles vs. Time
    (PD Control)');

subplot(2, 1, 2);
plot(time, e2, 'g');
xlabel('Time (s)');
ylabel('e2 (rad)');
```

The user-defined function `pd` is implemented as

Listing 4. pd.m

```
function [output] = pd(~, s, m1, m2, l1,
    l2, g, kp1, kd1, kp2, kd2,q1_fin,
    q2_fin)
q1 = s(1);
q2 = s(2);
q1dot = s(3);
q2dot = s(4);

e1 = q1_fin - q1;
e2 = q2_fin - q2;

f1 = kp1 * e1 - kd1 * q1dot;
f2 = kp2 * e2 - kd2 * q2dot;

 %Definitions
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
M = [M11, M12; M21, M22];

% Coriolis and Centrifugal Matrix
```

```
    c11 = -m2 * l1 * l2 * sin(q2) * q1dot;
    c12 = -m2 * l1 * l2 * sin(q2) * (q1dot +
        q2dot);
    c21 = 0;
    c22 = m2 * l1 * l2 * sin(q2) * q2dot;

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

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

    % Create the Gravitatinal matrix G
    G = [G1;G2];

    f = [f1; f2];

    tau_matrix = (M) * f;

    dq = [q1dot; q2dot];

    m_inverse = inv(M);
    q_double_dot_matrix = ((m_inverse) * (-C *
        dq - G)) + f;
    ddq1 = q_double_dot_matrix(1);
    ddq2 = q_double_dot_matrix(2);

    output = [q1dot; q2dot; ddq1; ddq2];
end
```

Reference to plots IV-B

C. PI Control

As opposed to the case of PD control, PI control requires six state variables in spite the absence of derivative terms. This is so that we can represent the two error-integral terms present.

The system is decribed again as such

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

where

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

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

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

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

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

Modeling the PI controller as

$$f = K_p e + k_i \int e dt$$

we get

$$f_1 = K_{p_1} e(q_1) + K_{I_1} \int e(q_1) dt$$
$$\implies f_1 = K_{P_1}(q_{1f} - q_1) + K_{I_1} \int (q_{1f} - q_1) dt$$
$$f_2 = K_{p_2} e(q_2) + K_{I_2} \int e(q_2) dt$$
$$\implies f_2 = K_{P_2}(q_{2f} - q_2) + K_{I_2} \int (q_{2f} - q_2) dt$$

thus we have

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

Out 6 state variables are thus

$$u_1 = q_1 \quad u_2 = q_2$$
$$u_3 = \int e_1 dt$$
$$u_4 = \int e_2 dt$$
$$u_5 = \dot{q}_1 \quad u_6 = \dot{q}_2$$

`ode45` requires the formulation of the derivative of these variables. Thus writing

$$\dot{u}_1 = \dot{q}_1 \quad \dot{u}_2 = \dot{q}_2$$
$$\dot{u}_3 = e_1 \quad \dot{u}_4 = e_2$$
$$\dot{u}_5 = \ddot{u}_1 = \ddot{q}_1 = \phi(t, u_1, u_2, u_3, u_4, u_5, u_6)$$
$$\dot{u}_6 = \ddot{u}_2 = \ddot{q}_2 = \psi(t, u_1, u_2, u_3, u_4, u_5, u_6)$$

Solving through MATLAB

Listing 5. PImain.m

```
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

q10 = 0.1;
q20 = 0.1;
x10 = 0;
x20 = 0;
q1dot0 = 0;
q2dot0 = 0;
t0 = 0;
tf = 10;

q1_fin = 0;
q2_fin = 0;

kp1 = 5;
ki1 = 10;
```

```
kp2 = 5;
ki2 = 10;

tspan = [t0 tf];

IC =[q10, q20, x10, x20, q1dot0, q2dot0];

options = odeset('RelTol', 1e-3, 'AbsTol',
    1e-6);

[time, state_values] = ode45(@(t, s)
    pi_k(t, s, m1, m2, l1, l2, g, kp1,
    ki1, kp2, ki2,q1_fin,
    q2_fin),tspan,IC, options);

q1 = state_values(:, 1);
q2 = state_values(:, 2);

figure;
subplot(2, 1, 1);
plot(time, q1, 'r');
xlabel('Time (s)');
ylabel('q1 (rad)');
sgtitle('Joint Angles vs. Time (PI
    Control)');

subplot(2, 1, 2);
plot(time, q2, 'g');
xlabel('Time (s)');
ylabel('q2 (rad)');

e1 = q1_fin - q1;
e2 = q2_fin - q2;

figure;
subplot(2, 1, 1);
plot(time, e1, 'r');
xlabel('Time (s)');
ylabel('e1 (rad)');
sgtitle('Error in Joint Angles vs. Time
    (PI Control)');

subplot(2, 1, 2);
plot(time, e2, 'g');
xlabel('Time (s)');
ylabel('e2 (rad)');
```

The user-defined function `pi_k` is implemented as

Listing 6. pi_k.m

```
function [output] = pi_k(~, s, m1, m2, l1,
    l2, g, kp1, ki1, kp2, ki2,q1_fin,
    q2_fin)
q1 = s(1);
q2 = s(2);
x1 = s(3);
x2 = s(4);
q1dot = s(5);
q2dot = s(6);

e1 = q1_fin - q1;
e2 = q2_fin - q2;

f1 = kp1 * e1 + ki1 * x1;
f2 = kp2 * e2 + ki2 * x2;

 %Definitions
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
M = [M11, M12; M21, M22];

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

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

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

% Create the Gravitatinal matrix G
G = [G1;G2];

tau = (M) * [f1; f2];
f = [f1; f2];
dq = [q1dot; q2dot];

m_inverse = inv(M);
q_double_dot_matrix = ((m_inverse) * (-C *
    dq - G)) + f;
ddq1 = q_double_dot_matrix(1);
ddq2 = q_double_dot_matrix(2);

output = [q1dot;q2dot;e1; e2;ddq1; ddq2];
end
```

Reference to plots IV-C

IV. Simulation Results

The goal of this paper is to simulate the the error dynamics of a two-link manipulator in its motion from a described initial position to final position.

The start and end positions are captured by specifying the joint angles as:

$$q_{initial} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} rad$$

$$q_{final} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} rad$$

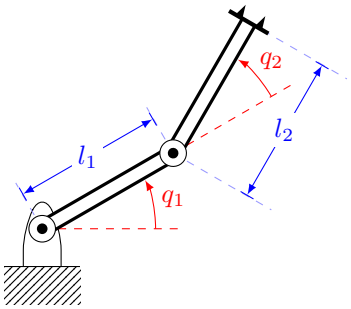


Fig. 2. initial position

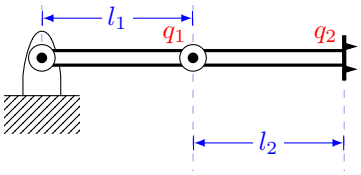


Fig. 3. final position

the parameters for mass and lengths of the links are

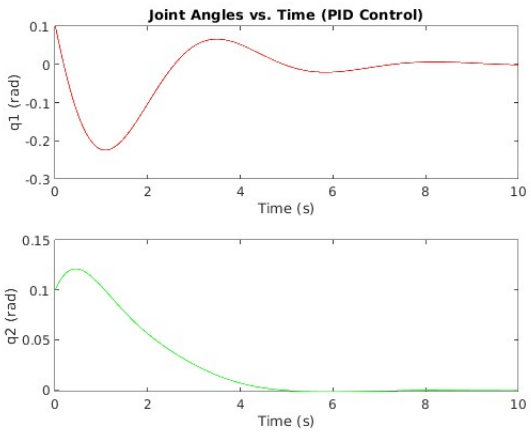
$$m_1 = 10kg \quad m_2 = 5kg$$
$$l_1 = 0.2m \quad l_2 = 0.1m$$

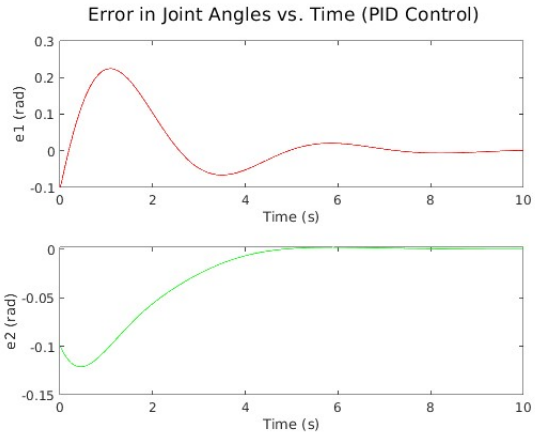
We observe independently the response of PID, PD and PI control

A. *PID Response*

The PID gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 100 \quad K_{I_1} = 200$$
$$K_{P_2} = 300 \quad K_{D_2} = 200 \quad K_{I_2} = 200$$



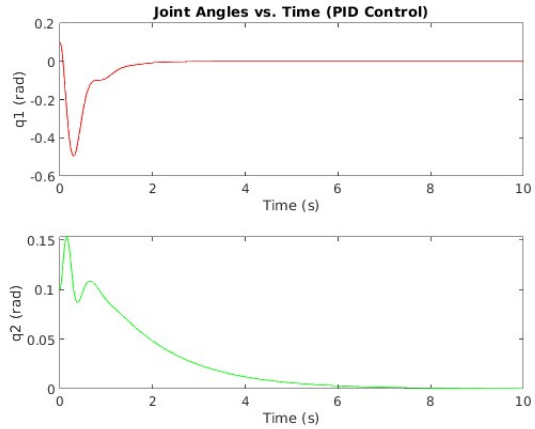


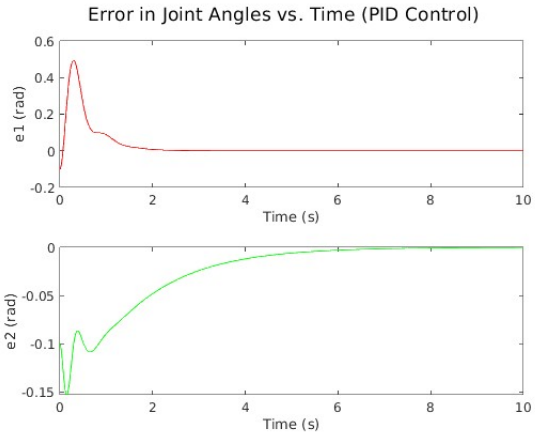
We observe that the joint angles q_1 and q_2 reach their final values in a very small time (about 7 seconds)

Furthermore the error in the joint angles also reaches 0 around 8 second after starting.

Analysing again the PID gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 10 \quad K_{I_1} = 200$$
$$K_{P_2} = 300 \quad K_{D_2} = 20 \quad K_{I_2} = 200$$

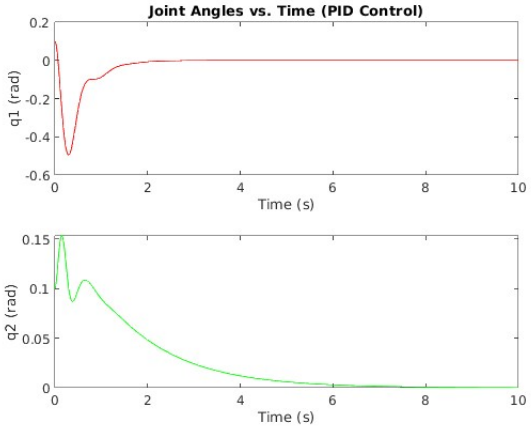


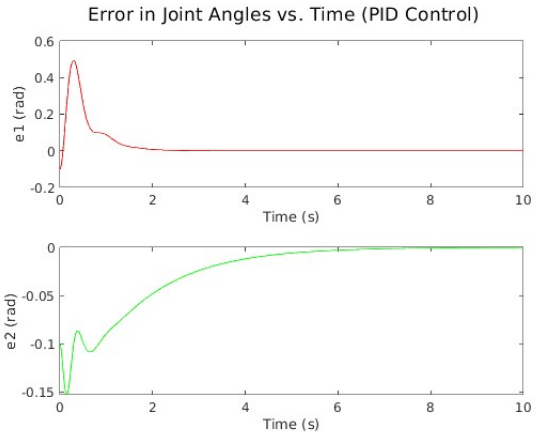


In this case we observe that q_1 and e_1 take much lesser time to saturate but q_2 and e_2 takes a little longer to settle down. This can be attributed to the lower K_D values.

Analysing again the PID gain parameters for this analysis are taken as:

$$K_{P_1} = 90 \quad K_{D_1} = 90 \quad K_{I_1} = 90$$
$$K_{P_2} = 90 \quad K_{D_2} = 90 \quad K_{I_2} = 90$$

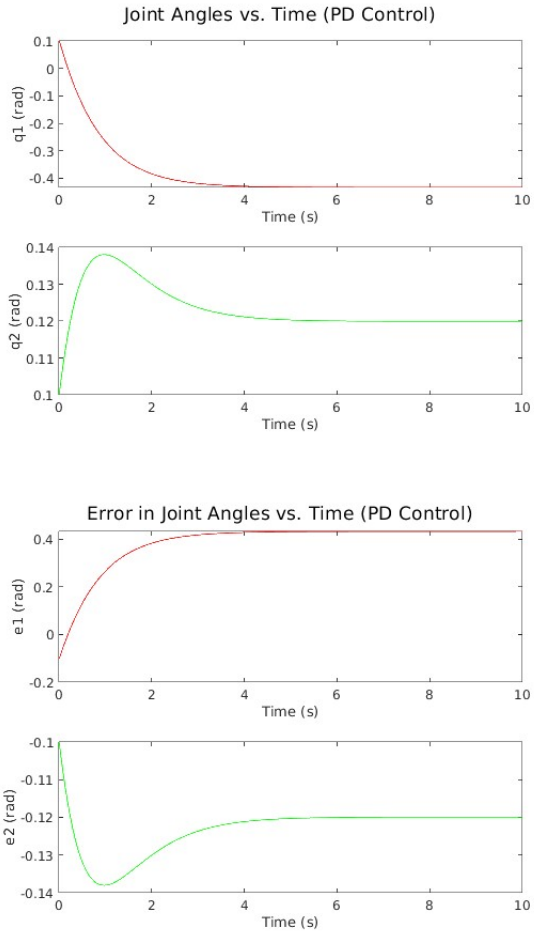




B. *PD Response*

The PD gain parameters for this analysis are taken as:

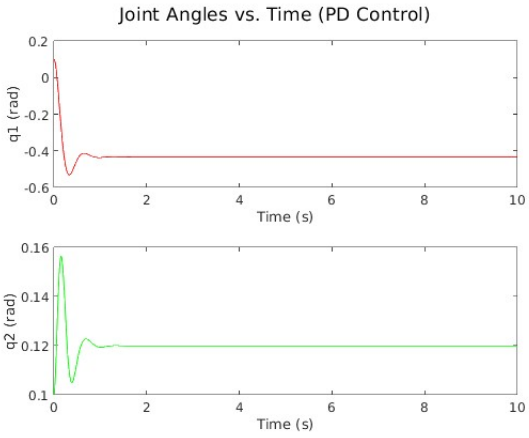
$$K_{P_1} = 100 \quad K_{D_1} = 100$$
$$K_{P_2} = 300 \quad K_{D_2} = 200$$

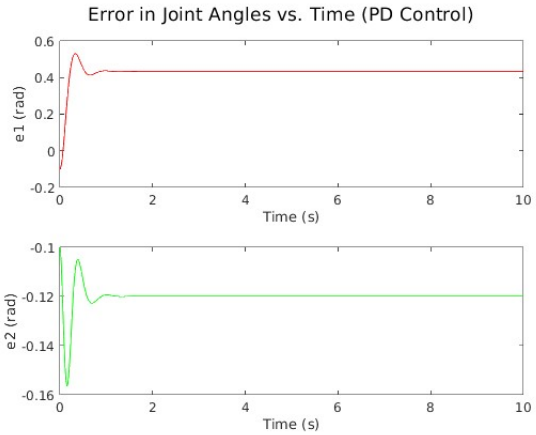


Here we observe that the system takes much lesser time to stabilize (around 3-4 seconds), however there is significant error in the final joint angles. This is observed in the second plot where the error signal does not settle at 0.

Analysing again the PD gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 10$$
$$K_{P_2} = 300 \quad K_{D_2} = 20$$



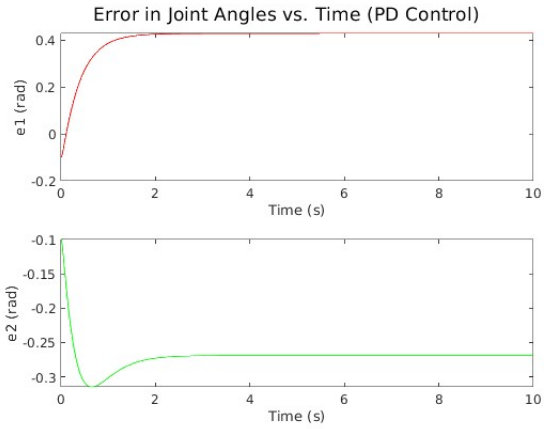


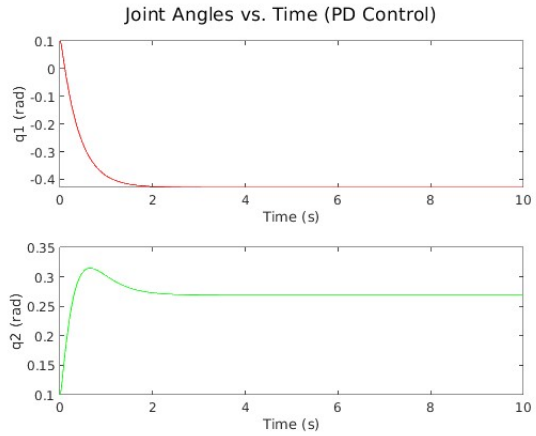
The lower values of K_D and higher values of K_P causes the the signals to saturate much faster as compared to the previous case. There is also considerably more offshoot

---

Analysing again the PD gain parameters for this analysis are taken as:

$$K_{P_1} = 98 \quad K_{D_1} = 48$$
$$K_{P_2} = 98 \quad K_{D_2} = 48$$



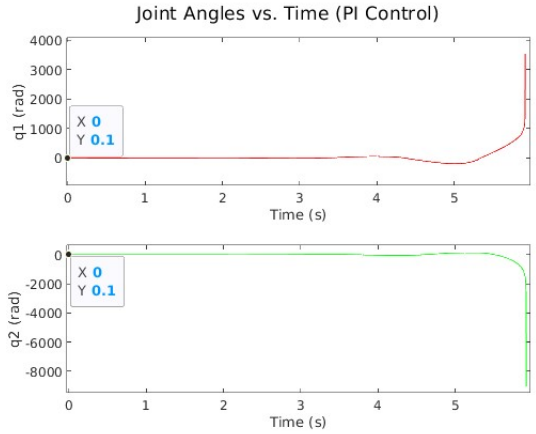


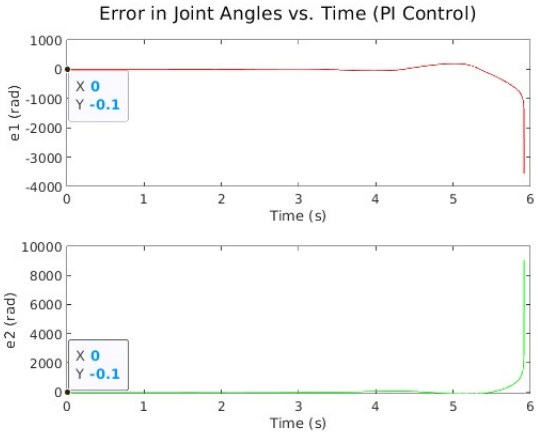
Here we see the offshoot is reduced due to higher relative values of K_D wrt K_P

C. *PI Response*

The PI gain parameters for this analysis are taken as:

$$K_{P_1} = 5 \quad K_{I_1} = 10$$
$$K_{P_2} = 5 \quad K_{I_2} = 10$$

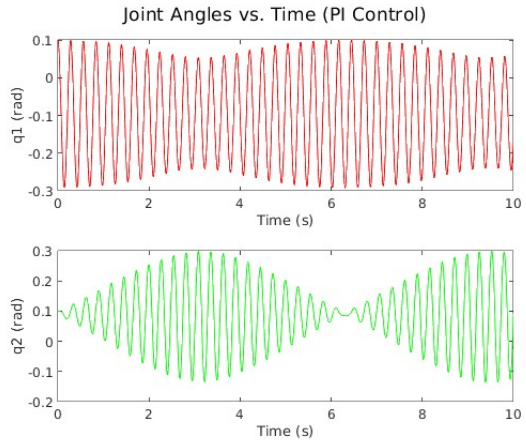


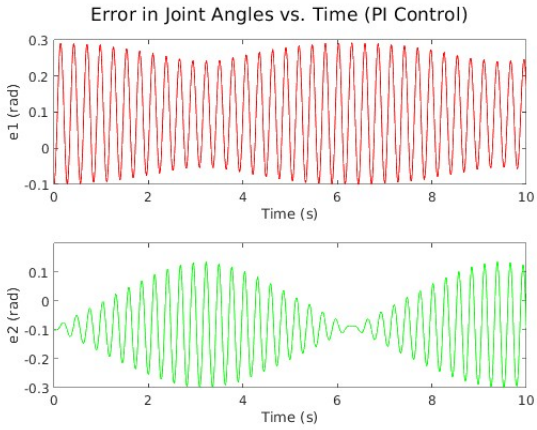


Here we see that the joint angles never actually reach the desired output and behaves chaotically. The error signal also diverges and oscillates.

One way to fix this is to have a very high Proportional gain and a small integral gain

$$K_{P_1} = 500 \quad K_{I_1} = 0.5$$
$$K_{P_2} = 500 \quad K_{I_2} = 0.5$$

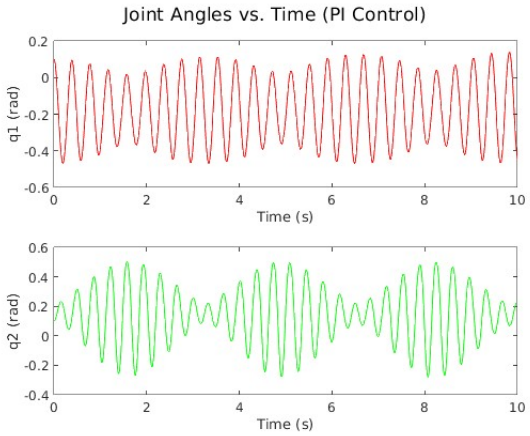


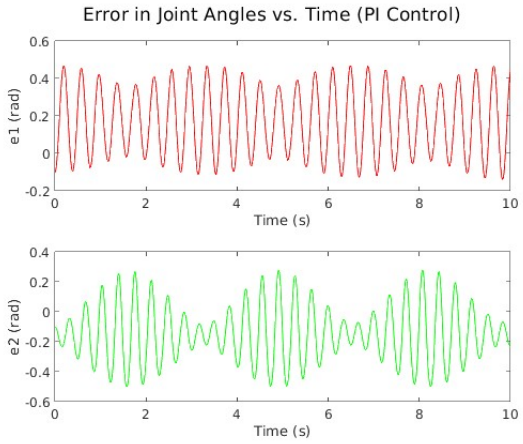


Now we see, although unstable and oscillatory, the joint angles and error signal both oscillate in the vicinity of the desired final value.

Analysing again the PI gain parameters for this analysis are taken as:

$$K_{P_1} = 250 \quad K_{I_1} = 3$$
$$K_{P_2} = 250 \quad K_{I_2} = 3$$





Here we observe as in the previous case the signals are oscillatory. The lower values of K_I and higher K_P means that the outputs deviate from the desired values more.

Further simulations on SIMULINK has been done in section V

V. Simulations On SIMULINK –

The same system as described in section II is implemented in SIMULINK to get the following block diagrams

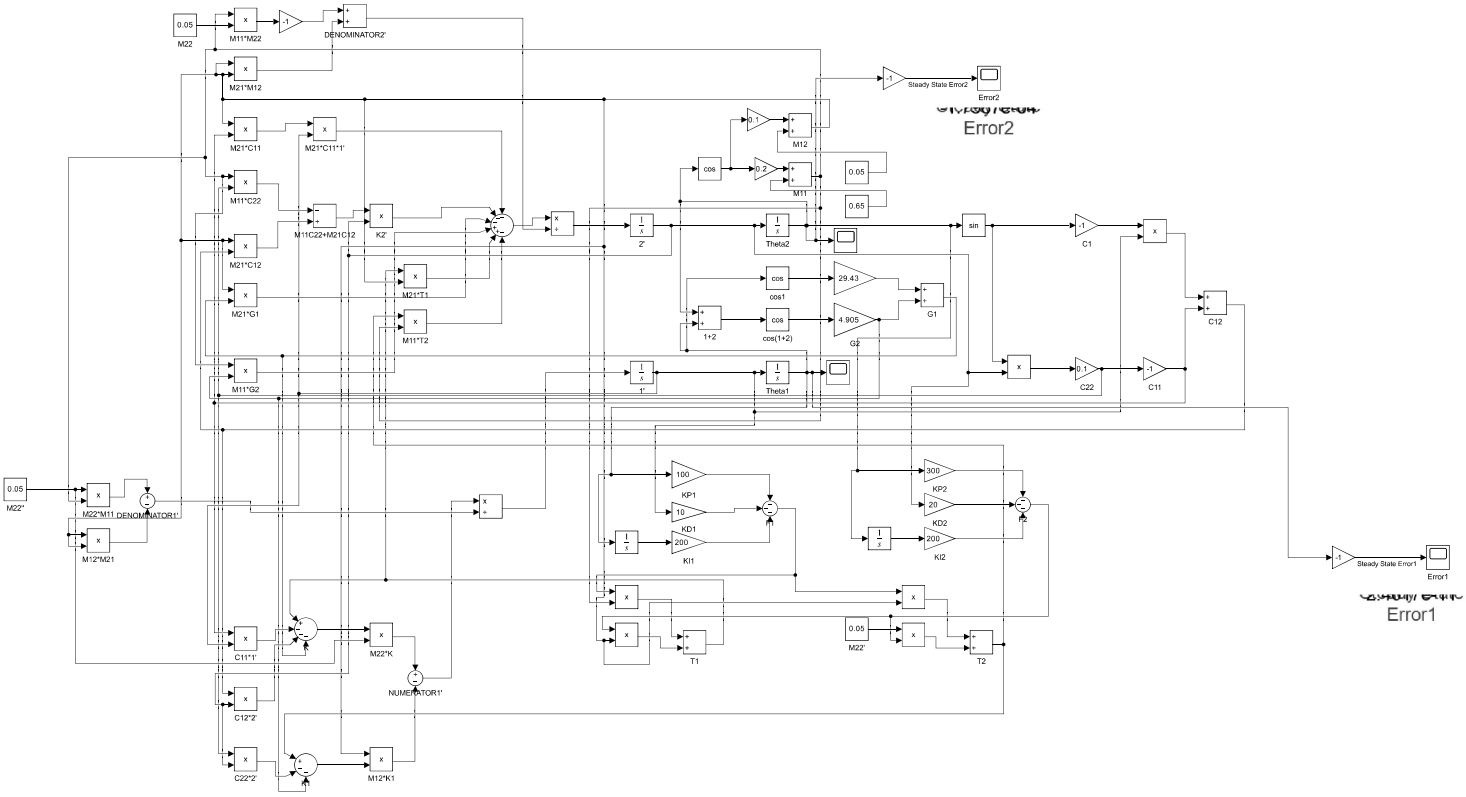


Fig. 4.
PID controller SIMULINK
K_{P_1} = 100 K_{D_1} = 10 K_{I_1} = 200
K_{P_2} = 300 K_{D_2} = 20 K_{I_2} = 200
Steady State Error 1 = 2.4617e-11
Steady State Error 2 = -1.7987e-04

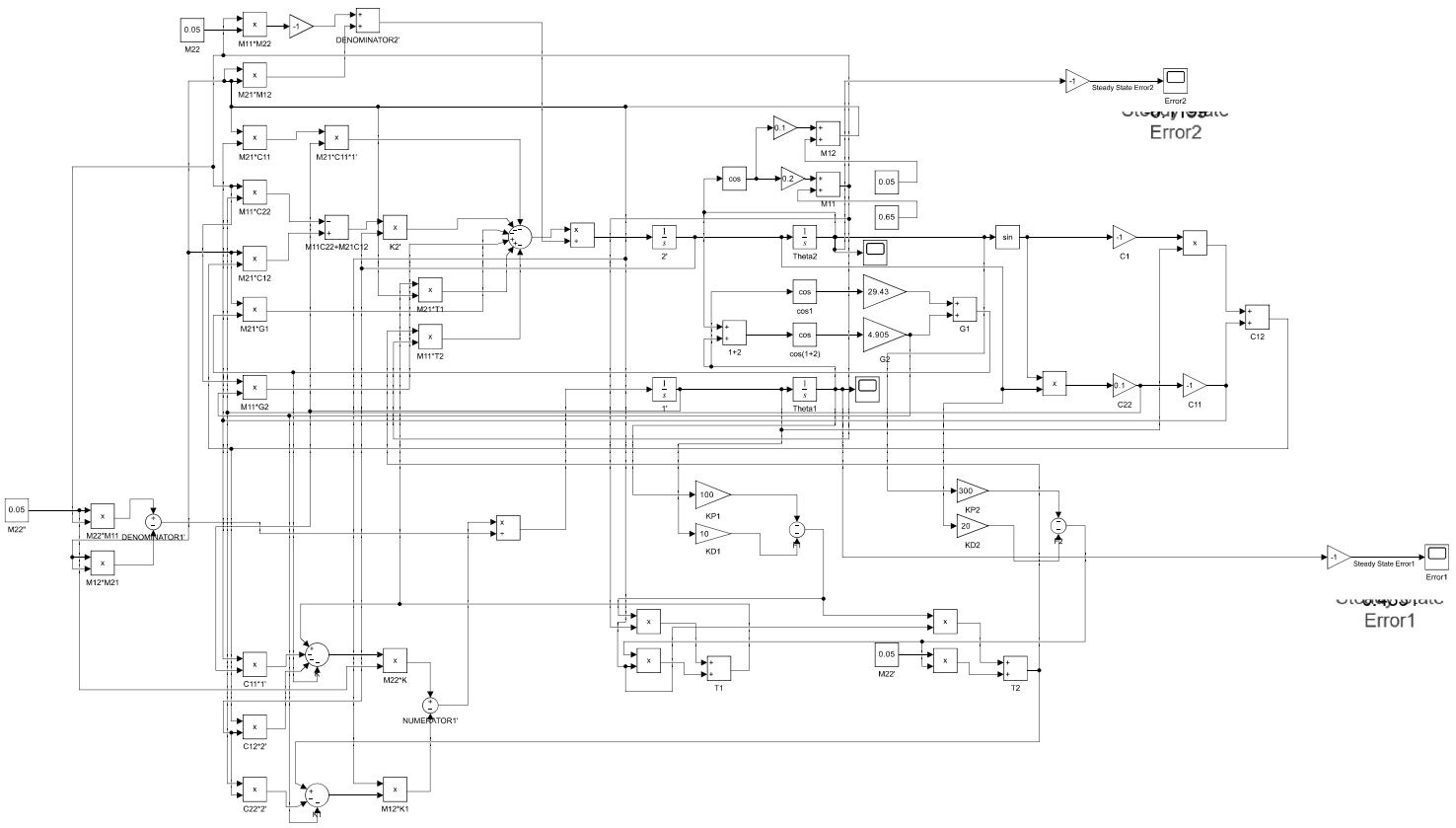


Fig. 5.
PD controller SIMULINK
K_{P_1} = 100 K_{D_1} = 100
K_{P_2} = 300 K_{D_2} = 200
Steady State Error 1 = 0.4331
Steady State Error 2 = -0.1199

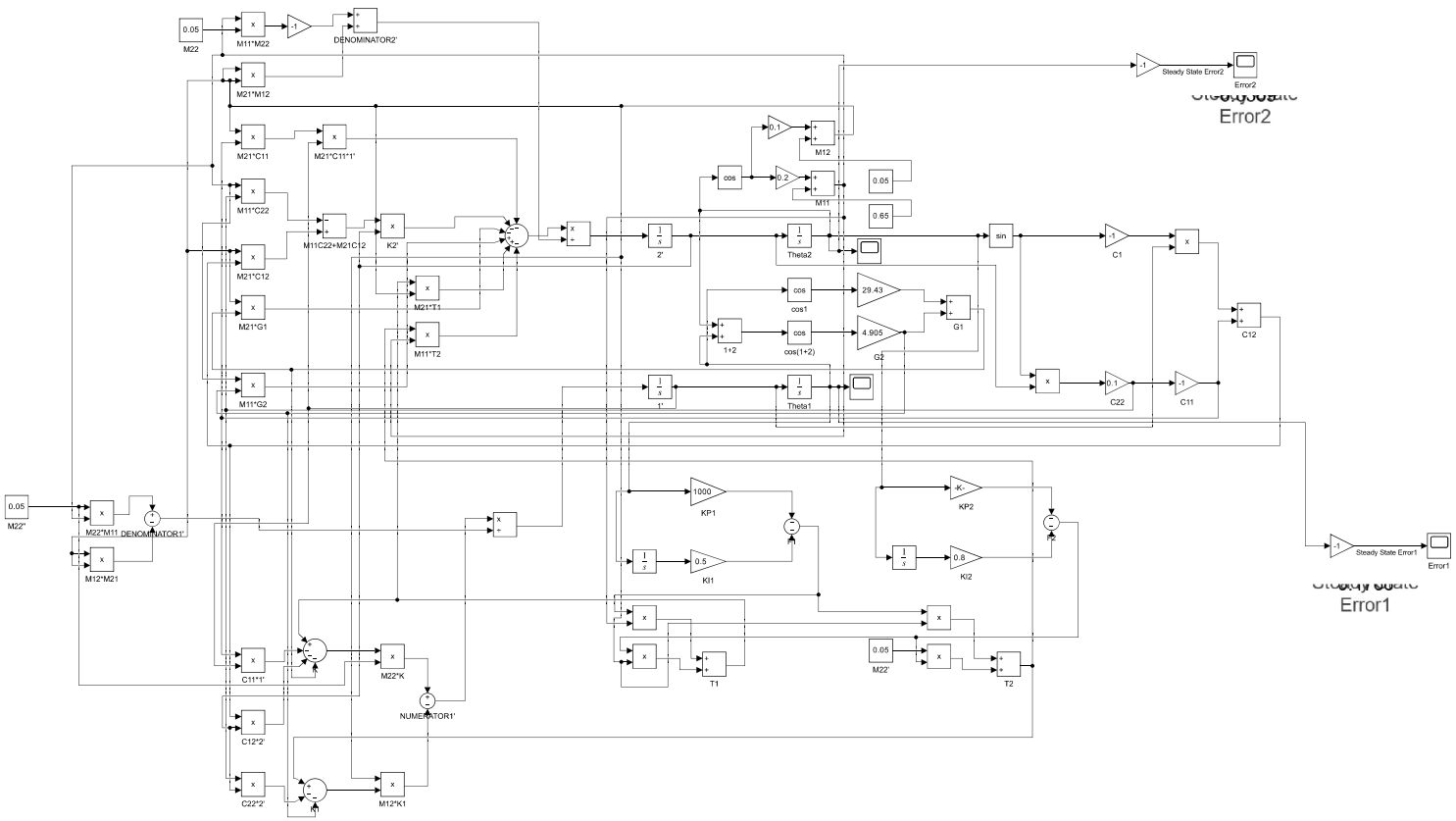


Fig. 6.
PI controller SIMULINK
K_{P_1} = 100 K_{I_1} = 100
K_{P_2} = 300 K_{I_2} = 200
Steady State Error 1 = 4.1055e+03
Steady State Error 2 = 1.4346

VI. Simulink Simulation Results

Running the systems we obtain the following joint angle and error plots

Again the gain values are taken as in section IV

A. PID Control

The PID gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 100 \quad K_{I_1} = 200$$
$$K_{P_2} = 300 \quad K_{D_2} = 200 \quad K_{I_2} = 200$$

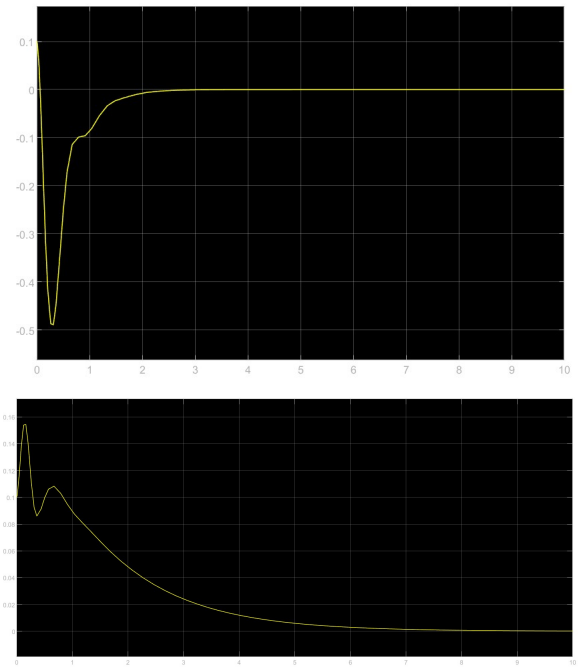


Fig. 7. q1 and q2 vs time plots for PID control resp.

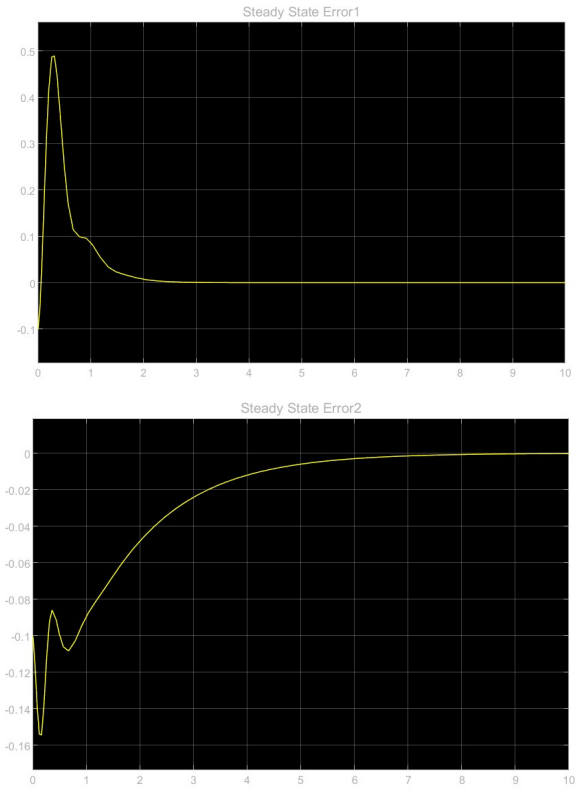


Fig. 8. error1,2 vs time plots for PID control resp

Analysing again the PID gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 10 \quad K_{I_1} = 200$$
$$K_{P_2} = 300 \quad K_{D_2} = 20 \quad K_{I_2} = 200$$

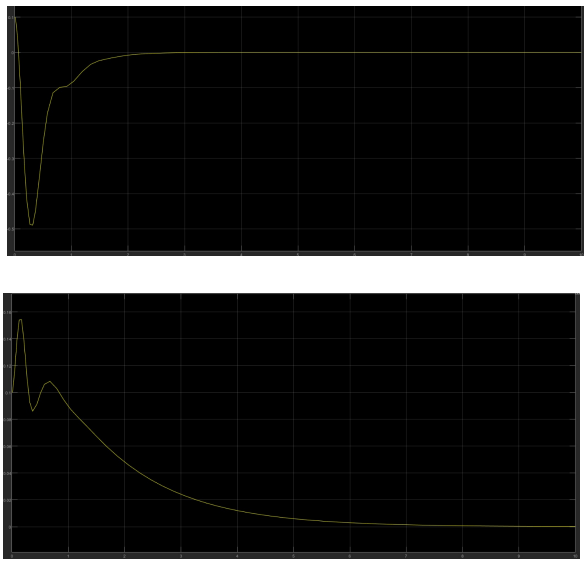


Fig. 9. q1 and q2 vs time plots for PID control resp.

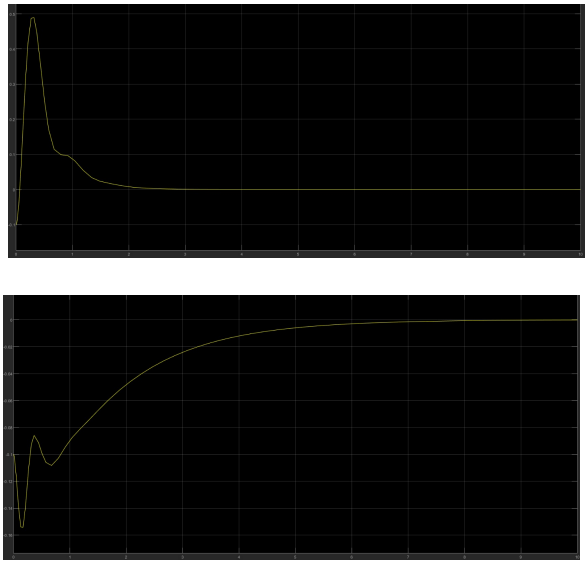


Fig. 10. error1,2 vs time plots for PID control resp

Analysing again the PID gain parameters for this analysis are taken as:

$$K_{P_1} = 90 \quad K_{D_1} = 90 \quad K_{I_1} = 90$$
$$K_{P_2} = 90 \quad K_{D_2} = 90 \quad K_{I_2} = 90$$

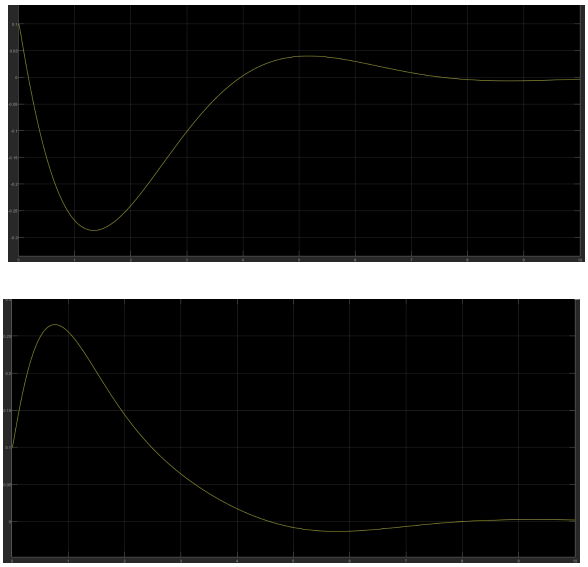


Fig. 11. q1 and q2 vs time plots for PID control resp.

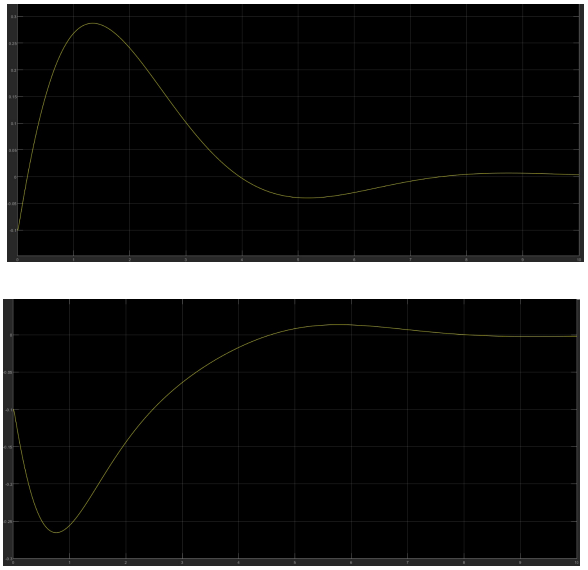


Fig. 12. error1,2 vs time plots for PID control resp

B. PD Control

The PD gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 100$$
$$K_{P_2} = 300 \quad K_{D_2} = 200$$

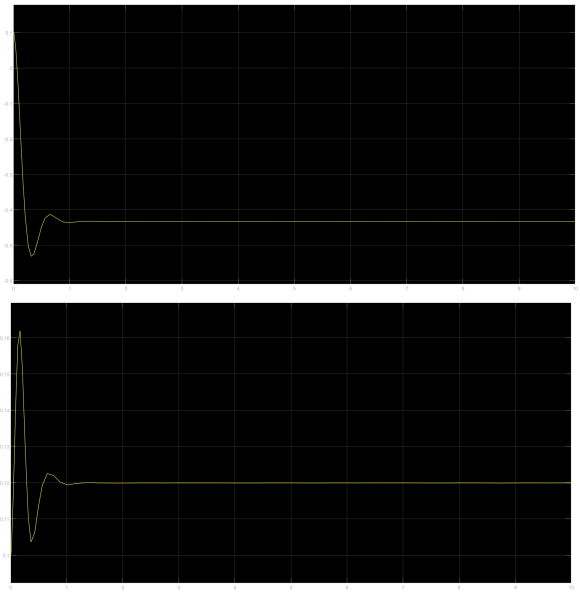


Fig. 13. q1 and q2 vs time plots for PD control resp.

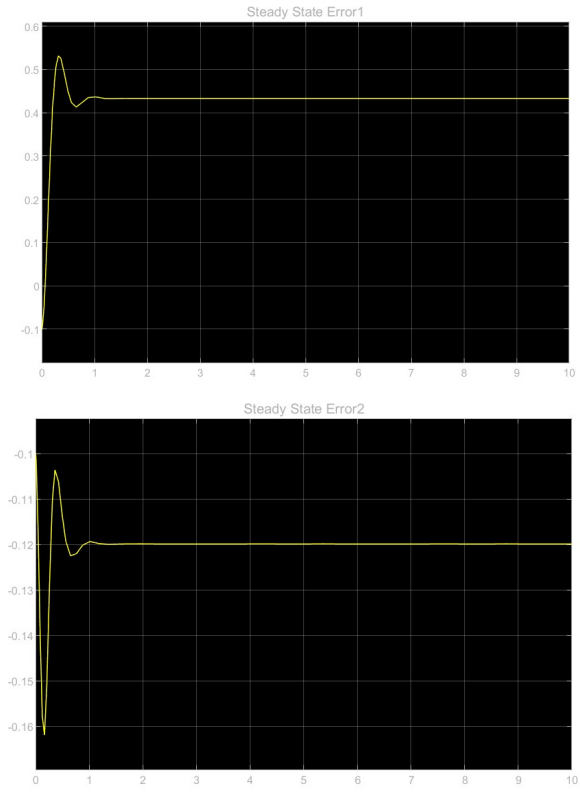


Fig. 14. error1,2 vs time plots for PD control resp

Analysing again the PD gain parameters for this analysis are taken as:

$$K_{P_1} = 100 \quad K_{D_1} = 10$$
$$K_{P_2} = 300 \quad K_{D_2} = 20$$

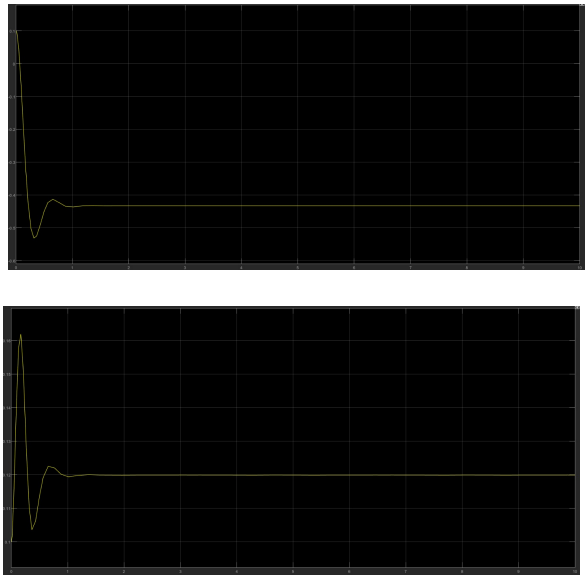


Fig. 15. q1 and q2 vs time plots for PD control resp.

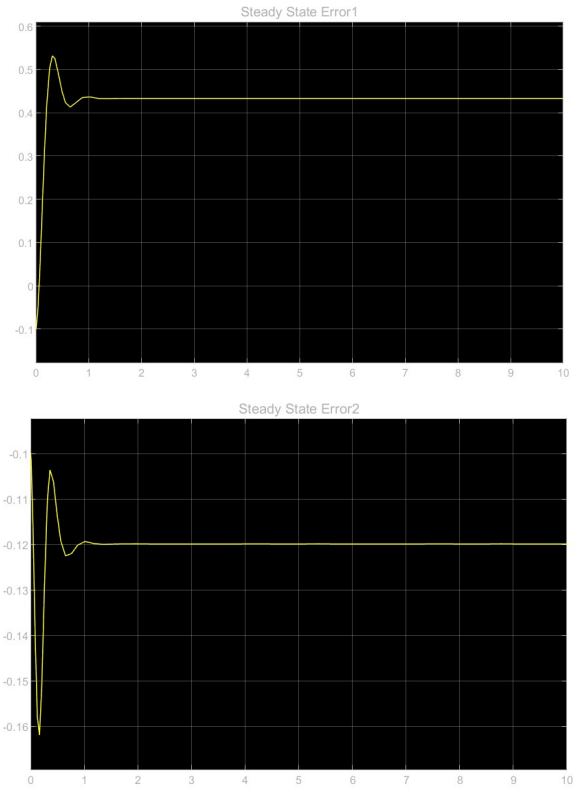


Fig. 16. error1,2 vs time plots for PD control resp

Analysing again the PD gain parameters for this analysis are taken as:

$$K_{P_1} = 98 \quad K_{D_1} = 48$$
$$K_{P_2} = 98 \quad K_{D_2} = 48$$

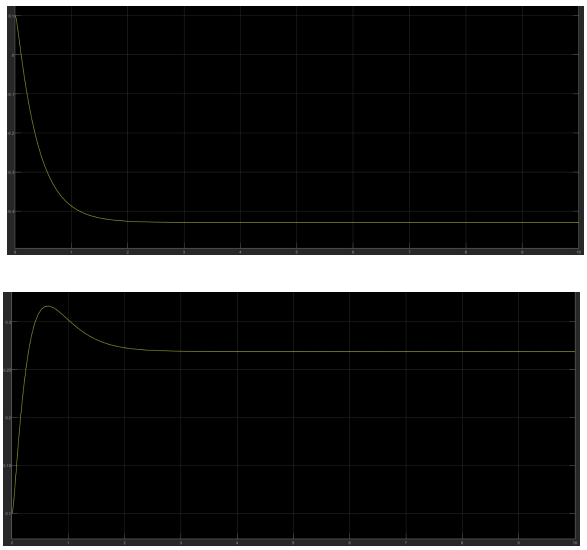


Fig. 17. q1 and q2 vs time plots for PD control resp.

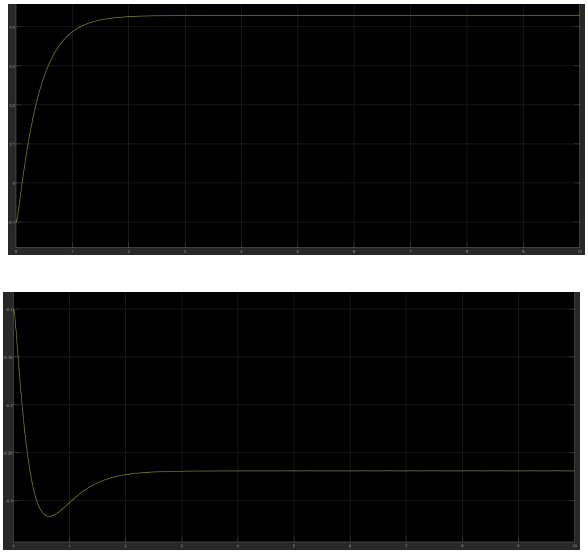


Fig. 18. error1,2 vs time plots for PD control resp

C. PI Control

The PI gain parameters for this analysis are taken as:

$$K_{P_1} = 5 \quad K_{I_1} = 10$$
$$K_{P_2} = 5 \quad K_{I_2} = 10$$

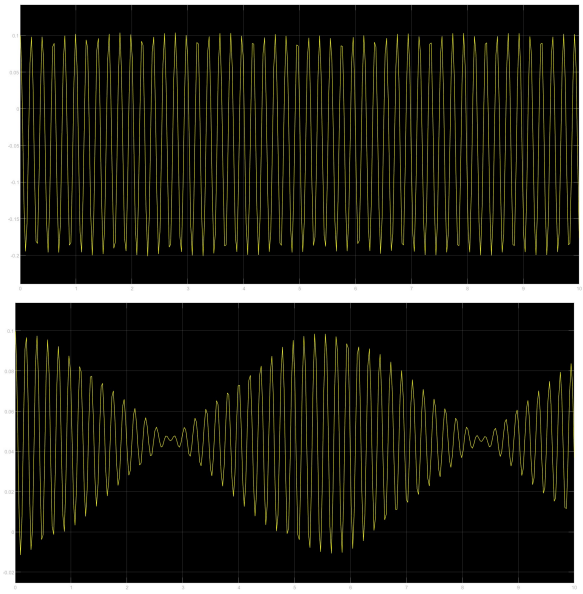


Fig. 19. q1 and q2 vs time plots for PI control resp.

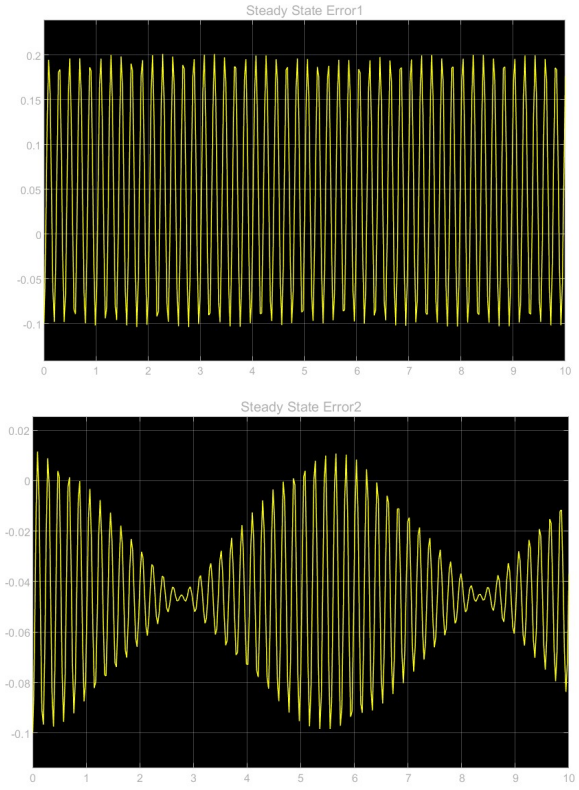


Fig. 20. error1,2 vs time plots for PI control resp

Analysing again the PI gain parameters for this analysis are taken as:

$$K_{P_1} = 250 \quad K_{I_1} = 3$$
$$K_{P_2} = 250 \quad K_{I_2} = 3$$

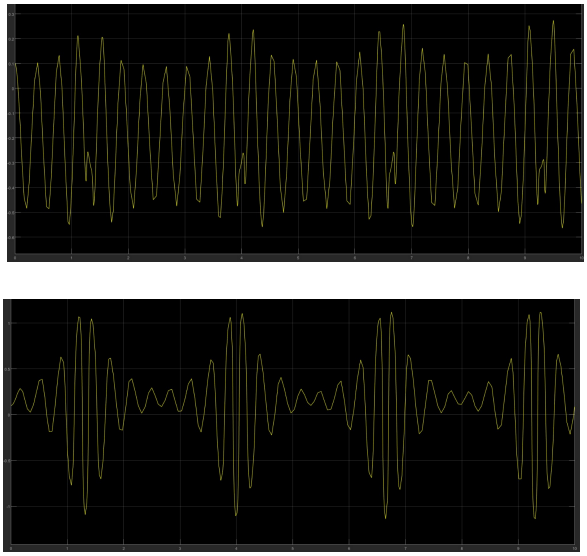


Fig. 21. q1 and q2 vs time plots for PI control resp.

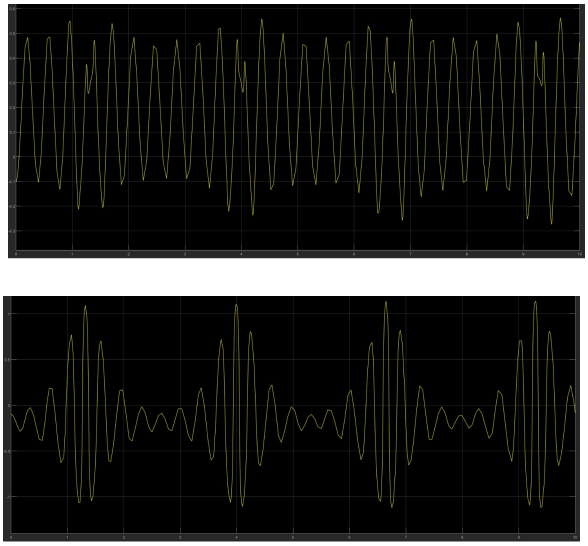


Fig. 22. error1,2 vs time plots for PI control resp

Analysing again the PI gain parameters for this analysis are taken as:

$$K_{P_1} = 500 \quad K_{I_1} = 0.5$$
$$K_{P_2} = 500 \quad K_{I_2} = 0.5$$

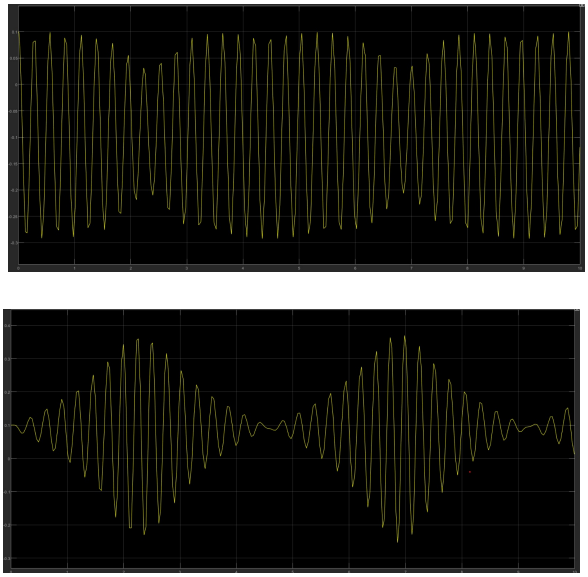


Fig. 23. q1 and q2 vs time plots for PI control resp.

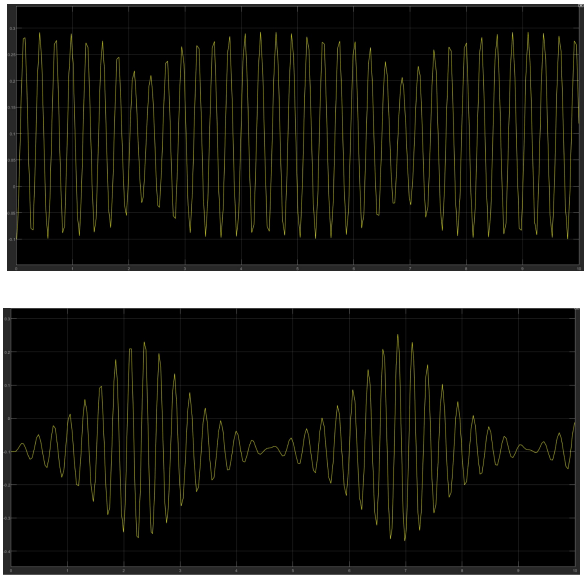


Fig. 24. error1,2 vs time plots for PI control resp

VII. Conclusion

For PD

A. *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 to 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 we need rapid adjustments or dynamic movements.

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 the speed of the response 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 oscillate excessively. This can result in less stability and accuracy in control.

B. 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 we need precise and controlled movements without oscillations.

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 you need a quicker initial movement and can tolerate some overshoot.

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.

C. 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 we require rapid and accurate positioning.

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 we need quick, dynamic responses from the arm.

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 you prioritize the initial speed of the response.

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.

D. 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.

E. 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.

For PI

F. Effect of Proportional Gain (K_P):

High K_P (Pros):

- Increasing K_P leads to a stronger response to errors in joint angles. The higher the kp, the more aggressively the controller corrects deviations from the desired positions.
- High K_P values can result in faster convergence to the desired joint angles even when errors are present.

High K_P (Cons):

- Excessively high K_P values can lead to overshooting the desired positions, which can result in oscillations and instability in the system, making it challenging to achieve precise control.
- High kp values may also lead to increased sensitivity to noise and disturbances, causing erratic behavior from our arm.

Low K_P (Pros):

- Low K_P values provide a more stable and less aggressive response to errors. This is helpful when we prioritize stability over speed.
- Lower K_P values are less likely to lead to overshoot and oscillations, making the system smoother and more predictable.

Low K_P (Cons):

- Very low K_P values may result in slow convergence to the desired positions, especially in the presence of significant errors. This can lead to delays in achieving the target joint angles.

G. Effect of Integral Gain (K_I)

High K_I (Pros):

- Increasing K_I increases the ability of the controller to eliminate steady-state errors. It effectively integrates error over time, making the controller more effective in maintaining precise joint angles.

- High K_I values can enhance the system's accuracy, ensuring that it settles very close to the desired positions.

High K_I (Cons):

- Excessive K_I values can lead to instability, especially if the system has significant noise or disturbances. It can result in a system that overshoots, oscillates, or exhibits erratic behavior.
- High K_I values require careful tuning and may not be suitable for all systems or environments.

Low K_I (Pros):

- Low K_I values provide a more stable response with less sensitivity to disturbances and noise. The system is less likely to exhibit excessive overshoot or oscillations.
- Lower K_I values are helpful when the system does not require strong correction of steady-state errors.

Low K_I (Cons):

- Very low K_I values may result in persistent steady-state errors, making it challenging to maintain precise joint angles over time. This can affect applications requiring high accuracy

H. Steady-State Error Reduction:

High K_I (Pros):

- High K_I values effectively eliminate steady-state errors, ensuring that the system settles very close to the desired joint angles without any long-term deviations.

High K_I (Cons):

- Excessive K_I values can lead to instability and a risk of overshooting or oscillations, especially in the presence of disturbances.

Low K_I (Pros):

- low K_I values result in a more stable response but may allow small steady-state errors to persist.

Low K_I (Cons):

- Very low K_I values may not effectively correct steady-state errors, potentially compromising the system's accuracy.

I. Conclusion

In practice, tuning the PI controller involves finding the right combination of K_P and K_I to achieve the desired control performance while avoiding instability, overshooting values or oscillations. The optimal values will depend on the specific characteristics of our robotic arm system and the control requirements of your application (refer simulink)

For PID

J. Effect of Proportional Gain (K_P):

High K_P (Pros):

- Increasing K_P reduces steady-state error quickly as high K_P scales the control error based on the current error. This means that when there is a significant error, the control action is strong, quickly reducing the error and bringing the system closer to the desired position.
- High K_P ensures high accuracy and precision which results in fine control, leading to precise positioning of the robotic arm.

High K_P (Cons):

- Excessively high K_P values can lead to oscillations and instability in the system, making it challenging to achieve precise control.
- High K_P values may also lead to overshooting and erratic behavior as it may cause the system to overshoot before settling.

Low K_P (Pros):

- Low K_P values provide a more stable response to errors as it is less likely to cause oscillations and instability.

Low K_P (Cons):

- Very low K_P values may result in slower error reduction due to a less aggressive control system, this leads to an increased time for the system to reach the desired position.

K. Effect of Proportional Gain (K_D):

High K_D (Pros):

- High K_D values can contribute to a dampening oscillations. High K_D values are effective at reducing the rate of change of error, damping oscillations, and resulting in a smoother control response
- A high K_D value gives a highly stable system when tuned correctly which is ideal for applications requiring precision and stability.

High K_D (Cons):

- High 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 result in a rapid initial response, allowing the system to quickly approach the desired position.

Low K_D (Cons):

- Very low K_D values may not effectively dampen oscillations, potentially compromising stability.

L. Effect of Integral Gain (K_I)

High K_I (Pros):

- Increasing K_I gives an aggresive response to long-term errors which makes the system responsive to persistent disturbances. This ensures that the system approaches the desired system accurately.

High K_I (Cons):

- Excessive K_I values can lead to instability, especially if the system has significant noise or disturbances, potentially making the control system unpredictable.

Low K_I (Pros):

- Low K_I values avoid excessive accumulation of error and maintain stability in the system reducing the risk of oscillations and instability. This provide a smoother and more gradual control response.

Low K_I (Cons):

- Very low K_I values may not eliminate steady-state errors potentially resulting in persistent errors.

M. Conclusion

In practice, tuning the PID controller involves finding the right combination of K_P , K_I and K_D to achieve the desired control performance while avoiding instability, overshooting values or oscillations. The optimal values will depend on the specific characteristics of our robotic arm system and the control requirements of your application (refer simulink)

REFERENCES

[1] Mahboub Baccouch and Stephen Dodds, " A Two-Link Robot Manipulator: Simulation and Control Design "
[2] Matlab ode45 (and Similar) Tutorial Part 1: The Basics
[3] Matlab ode45 Tutorial Part 2A: Multidimensional Problems
[4] Two Link Robot Dynamics
[5] Two Link Manipulator Jacobian Example - Basic Method