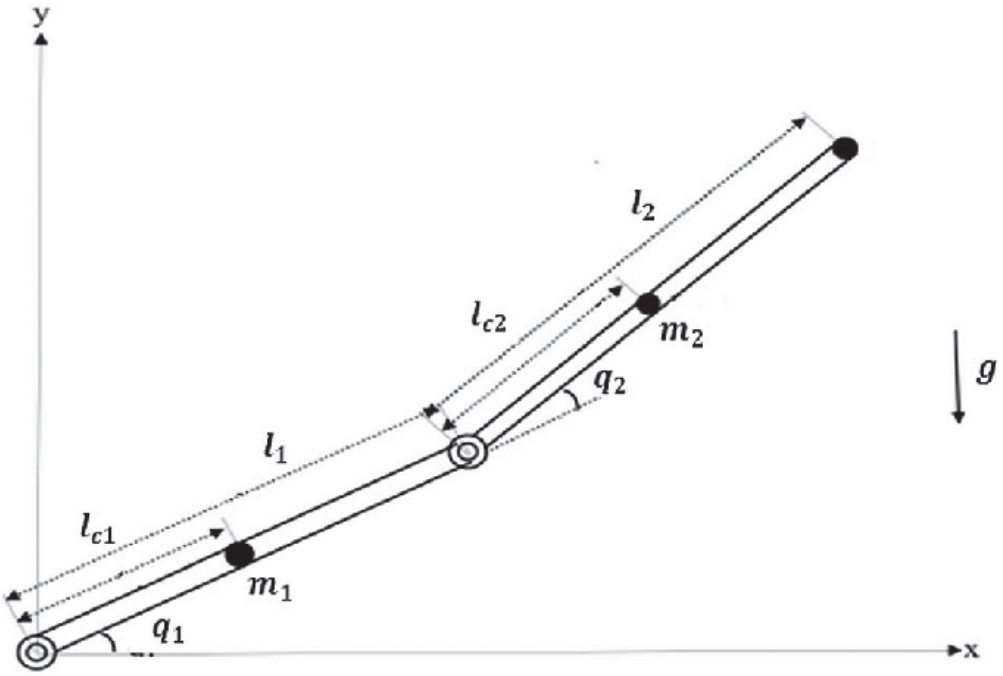
SCIENTIFIC THINKERS PROJECT REPORT:

SYSTEMS THINKING MINI PROJECT:-

2 LINK MANIPULATOR:-

TEAM MEMEBRS:-

1) Chirag Goyal 2022102041
2) Manas Deshmukh 2022102040
3) Prakhar Raj 2022102066
4) Aditya Raj Singh 2022102067
5) Sreevidhu Yarra 2022102012
6) Aniket Verma 2022102019



BASIC INTUITION ABOUT 2 LINK MANIPULATOR AND REAL LIFE SYSTEMS BASED ON IT:-

A 2-link manipulator, often referred to as a 2-DOF (Degrees of Freedom) robot arm, is a fundamental concept in robotics and mechanical engineering. Imagine a robotic arm composed of two segments (links) connected by joints. Understanding the intuition behind a 2-link manipulator involves grasping the geometry, degrees of freedom, and its practical applications. Let's break down the key aspects:

1. Linkage and Joints:

- **Links:** The robotic arm consists of two rigid links. These links can be of varying lengths and are connected end-to-end.
- **Joints:** The links are connected by joints that allow relative motion. In a 2-link manipulator, there are typically two joints - one at the base (allowing rotation in a plane) and another at the elbow (allowing movement in another plane).

2. Degrees of Freedom (DOF):

- **DOF in Planes:** A 2-link manipulator has two degrees of freedom, meaning it can move in a 2D plane. The joint rotations enable the arm to reach any point within its 2D workspace.
- **Configuration Space:** The possible positions of the end-effector (the tip of the robotic arm) form its configuration space. For a 2-link manipulator, this space is 2D, representing all possible combinations of joint angles.

3. Intuition Behind Movement:

- **End-Effector Path:** By changing the joint angles, the end-effector can trace various paths in the 2D plane. Each combination of joint angles results in a unique position of the end-effector.
- **Reachability:** The manipulator can reach different points within its workspace. The lengths of the links and the joint angles determine the reachable area. Different combinations of link lengths and joint angles create different workspaces.

4. Practical Applications:

- **Assembly Line Robots:** 2-link manipulators are used in manufacturing for tasks like picking and placing objects on assembly lines due to their simple and precise movements.
- **Surgical Robotics:** In minimally invasive surgeries, robotic arms with 2 links can be used for precise movements within limited spaces.
- **Educational Purposes:** Understanding 2-link manipulators helps students and researchers grasp essential concepts in robotics, kinematics, and control theory.

5. Kinematics:

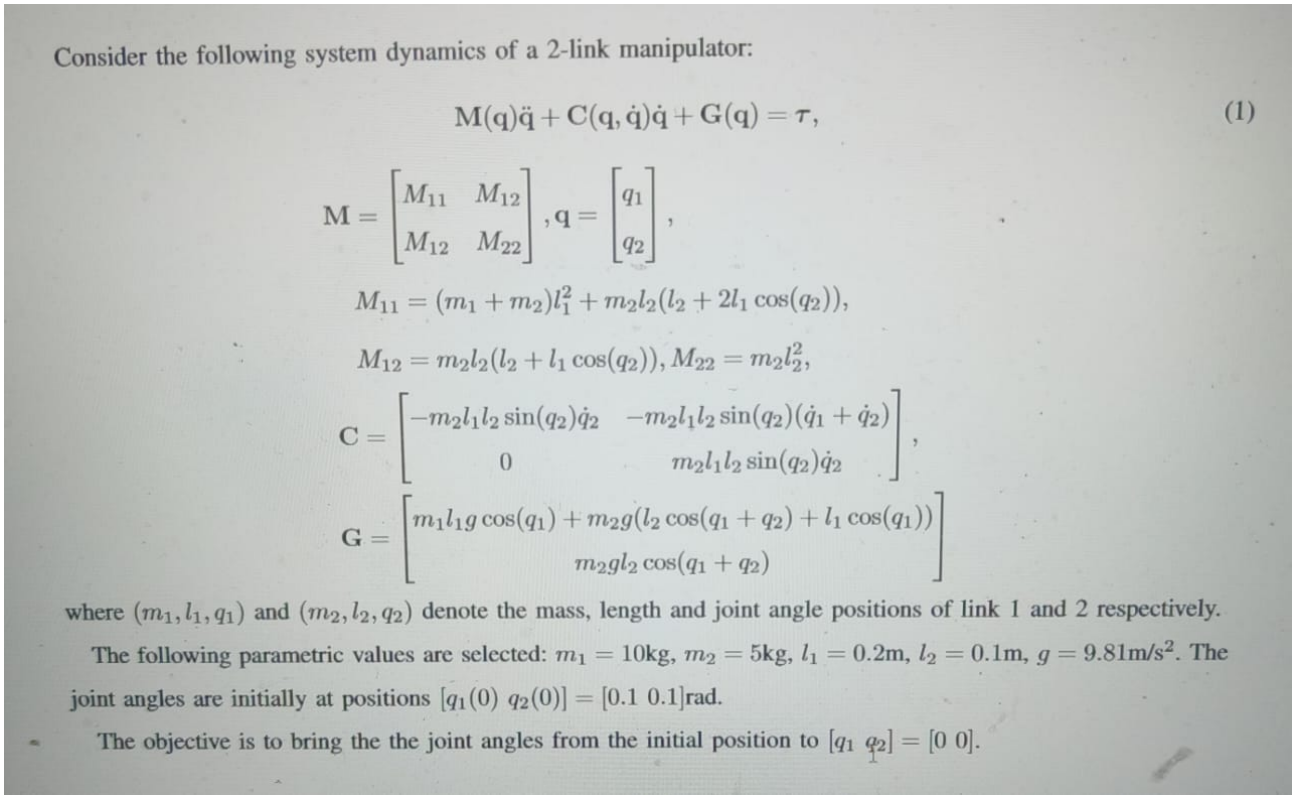
- **Forward Kinematics:** Describes how joint angles affect the position of the end-effector. It involves computing the (x, y) coordinates of the end-effector given the joint angles.
- **Inverse Kinematics:** Involves finding the joint angles required to position the end-effector at a specific (x, y) coordinate. Solving inverse kinematics problems is crucial for controlling the manipulator to reach desired points.

In summary, a 2-link manipulator provides a foundational understanding of robotic systems. It illustrates the relationship between joint motions and end-effector positions, forming the basis for more complex robotic systems. Mastering the concepts of 2-link manipulators is essential for anyone working in the fields of robotics, automation, or mechatronics or any other area related to the fields mentioned above.A 2-link manipulator, with its two interconnected segments and joints, represents a simplified yet powerful model in robotics. Its mathematical analysis involves solving kinematic equations to understand the arm's reachable workspace and the end-effector's trajectory. This simplicity makes it a fundamental teaching tool for kinematics, dynamics, and control theory, serving as a building block for more complex robotic systems. Mastering the principles of a 2-link manipulator is crucial for engineers and researchers, providing

insights into the foundational concepts that govern the movement and control of robotic arms in various real-world applications

BASED ON THE INTUITION OF 2 LINK MANIPULATOR APPROACHING THE GIVEN PROBLEM STATEMENT:-

PROBLEM STATEMENT:-



Consider the following system dynamics of a 2-link manipulator:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau}, \quad (1)$$

$$\mathbf{M} = \begin{bmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{bmatrix}, \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix},$$

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

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

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

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

where (m_1, l_1, q_1) and (m_2, l_2, q_2) denote the mass, length and joint angle positions of link 1 and 2 respectively.

The following parametric values are selected: $m_1 = 10$kg, $m_2 = 5$kg, $l_1 = 0.2$m, $l_2 = 0.1$m, $g = 9.81$m/s^2. The joint angles are initially at positions $[q_1(0)\ q_2(0)] = [0.1\ 0.1]$rad.

The objective is to bring the the joint angles from the initial position to $[q_1\ q_2] = [0\ 0]$.

ACCORDING TO THE PROBLEM STATEMENT THE MAIN OBJECTIVE IS TO BRING Q1 AND Q2 TO 0 RADIAN(BOTH) FROM Q1 AND Q2(0.1 RAD):-

Intuition:

1. **Dynamic Equations:** The system dynamics equation $M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=\tau$ $M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=\tau$ describes the motion of the manipulator. Here, $M(q)$ is the inertia matrix, $C(q,\dot{q})C(q,\dot{q})$ captures centrifugal and Coriolis effects, $G(q)G(q)$ represents gravitational forces, $\tau\tau$ is the joint torque,

and qq, q˙q˙, and q¨q¨ are joint positions, velocities, and accelerations respectively.

2. **Control Strategy:** To achieve the desired joint positions, a control strategy such as PID control, computed torque control, or trajectory planning algorithms can be applied. These strategies use feedback from the current joint positions and velocities to calculate the required joint torques to achieve the desired positions.

Possibilities of Solving the Problem:

1. **Inverse Dynamics Control:** Inverse dynamics control calculates the joint torques required to achieve a desired trajectory. By knowing the target joint positions and velocities, this method can compute the necessary torques using the given dynamic equations.
2. **Iterative Numerical Methods:** Numerical methods like the Runge-Kutta integration can be employed to numerically solve the differential equations derived from the dynamics. By discretizing the equations, the system's state (joint positions and velocities) can be updated iteratively until it converges to the desired positions.
3. **Optimal Control:** Optimal control techniques, such as Linear Quadratic Regulator (LQR) or Model Predictive Control (MPC), can be utilized to minimize a cost function representing the deviation from the desired positions. These methods find the optimal control inputs (joint torques) that drive the system to the desired positions while considering the system dynamics and constraints.

In summary, solving this problem involves understanding the system dynamics and applying suitable control strategies to compute the joint torques required to move the 2-link manipulator from the initial to the target joint positions. The choice of the specific method depends on factors like system constraints, accuracy requirements, and real-time computational capabilities

CODE FOR SYSTEM DYNAMIC FUNCTION WE USE RUNGE INTEGRATION METHOD IN:-

```
function [dydt] = system_dynamics(t, y, m1, m2, l1, l2, g, KP1, KP2, KD1,
KD2, KI1, KI2, q1f, q2f)
    q1 = y(1);
    dq1 = y(2);
    q2 = y(3);
    dq2 = y(4);
    int_e1=y(5);
    int_e2=y(6);

    e1 = q1f-q1;
    e2 = q2f-q2;

    %{
    %{
    persistent integral1 integral2

    if isempty(integral1)
        integral1 = 0;
    end

    if isempty(integral2)
        integral2 = 0;
    end
    %}
    integral1 = cumtrapz(t, e1);  % Integrate e1 over time
    integral2 = cumtrapz(t, e2);  % Integrate e2 over time
    integral1 = integral1 + e1;
    integral2 = integral2 + e2;
    %}
%{
    u1 = KP1 * e1 - KD1 * dq1 + KI1 * integral1;
    u2 = KP2 * e2 - KD2 * dq2 + KI2 * integral2;
%}
    ans1=KP1 * e1 - KD1 * dq1 + KI1 *(int_e1);
    ans2=KP2 * e2 - KD2 * dq2 + KI2 *(int_e2);

    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;

    C11 = -m2 * l1 * l2 * sin(q2) * dq2;
    C12 = -m2 * l1 * l2 * sin(q2) * (dq1 + dq2);
    C21 = 0;
    C22 = m2 * l1 * l2 * sin(q2) * dq2;

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

    M=[M11,M12;M12,M22];
    T=(M)*[ans1;ans2];
    C=[C11,C12;C21,C22];
    G=[G1;G2];
    dq=[dq1;dq2];
```

```
    M_inverse=inv(M);
    Q=(M_inverse)*(T-C*dq-G);
    ddq1=Q(1);
    ddq2=Q(2);

   % ddq1 = (1/(M21*M12-M11*M22))(M21(u1-C11*dq1-C12*dq2-G1)-M11*(u2-
C22*dq2-G2));
   % ddq2 = ((1/M22*M11-M12*M21))(M22(u1-C11*dq1-C12*dq2-G1)-M12*(u2-
C22*dq2-G2));

    dydt = [dq1; ddq1;dq2;ddq2;e1;e2];
end
```

CODE EXPLANATION:-

1. **Input Extraction:**
   - Extracts joint positions, velocities, and integral of errors from the input vector `y`.
2. **Error Calculation:**
   - Calculates errors in joint angles (`e1` and `e2`) by subtracting desired joint angles from the current joint angles.
3. **Integral of Errors Calculation:**
   - Integrates errors over time using numerical integration (cumulative trapezoidal method) to obtain `integral1` and `integral2`.
4. **Control Input Calculation:**
   - Computes control inputs (`ans1` and `ans2`) using PID control formulas, where proportional, derivative, and integral gains (`KP1`, `KP2`, `KD1`, `KD2`, `KI1`, `KI2`) are applied to the errors and integral of errors.
5. **Dynamic Model Computation:**
   - Computes the elements of the mass matrix (`M11`, `M12`, `M21`, `M22`) based on the given manipulator parameters.
   - Calculates Coriolis and centrifugal forces (`C11`, `C12`, `C21`, `C22`) and gravitational forces (`G1`, `G2`).
6. **Inverse Dynamics Calculation:**
   - Uses the computed mass matrix, Coriolis forces, gravitational forces, and control inputs to compute joint accelerations (`ddq1` and `ddq2`) using inverse dynamics.
7. **Output:**
   - Constructs the output vector `dydt` containing joint velocities, accelerations, errors, and integral of errors.

In summary, this function integrates the errors, applies PID control to calculate control inputs, computes the inverse dynamics of the 2-link manipulator, and provides the derivatives of the state variables for numerical simulations

CODE FOR MAIN FUNCTION:-

```
m1 = 10;
m2 = 5;
l1 = 0.2;
l2 = 0.1;
g = 9.81;

q10 = 0.1;
q20 = 0.1;
dq1_0 = 0;
dq2_0 = 0;

q1f = 0;
q2f = 0;

KP1 = 100;
KP2 = 300;
KI1 = 200;
KI2 = 200;
KD1 = 150;
KD2 = 200;

%odefun = @(t, x)[system_dynamics(t, x, m1, m2, l1, l2, g, KP1, KP2, KD1,
KD2, KI1, KI2, q1f, q2f)];

tspan = [0 10];

initial_conditions = [q10, dq1_0, q20, dq2_0,0,0];

options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);
[T, y] = ode45(@(t, x)system_dynamics(t, x, m1, m2, l1, l2, g, KP1, KP2,
KD1, KD2, KI1, KI2, q1f, q2f), tspan, initial_conditions,options);


q1 = y(:, 1);
q2 = y(:, 3);

e1 = q1f - q1;
e2 = q2f - q2;

figure;
subplot(4, 2, 1);
plot(T, q1, 'r');
xlabel('Time (s)');
ylabel('q1 (rad)');
title('Joint Angles vs. Time (PID Control)');

%{
subplot(4, 2, 2);
plot(T, e1, 'b');
xlabel('Time (s)');
ylabel('e1 (rad)');
title('Error in q1 vs. Time');
%}

subplot(4, 2, 3);
plot(T, q2, 'g');
xlabel('Time (s)');
ylabel('q2 (rad)');
```

```
%{
subplot(4, 2, 4);
plot(T, e2, 'p', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('e2 (rad)');
title('Error in q2 vs. Time');
%}
```

CODE EXPLANATION:-

1. **Setting Parameters:**
   - Masses, lengths, gravitational acceleration, and initial joint positions and velocities are defined.
2. **Controller Gains:**
   - Proportional (KP1, KP2), Integral (KI1, KI2), and Derivative (KD1, KD2) gains for PID control are specified.
3. **ODE Solver Setup:**
   - Time span for simulation (`tspan`) and initial conditions for the 2-link manipulator are set.
4. **ODE Solver Call:**
   - The ODE solver (`ode45`) is used to solve the system dynamics (`system_dynamics`) over the specified time span with given parameters and initial conditions.
5. **Error Calculation:**
   - Errors in joint angles (`e1` and `e2`) are computed by comparing desired joint positions with the simulated positions.
6. **Plotting Results:**
   - Plots joint angles (`q1` and `q2`) against time to visualize the manipulator's movement and tracks the errors over time.

The code essentially simulates the 2-link manipulator's behavior using PID control and visualizes the joint angles and errors over time

How we are solving it using ODE45 command

The equation of motion can be compactly written as

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

where

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{bmatrix},\ q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix},$$

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

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

$$G = \begin{bmatrix} m_1 l_1 g cos(q_1) + m_2 g(l_2 cos(q_1 + q_2) + l_1 cos(q_1)) \\ m_2 g l_2 cos(q_1 + q_2) \end{bmatrix},\ \mathrm{T} = \begin{bmatrix} \mathrm{T}_1 \\ \mathrm{T}_2 \end{bmatrix}$$

So , we want to calculate ddq,so we send all things to the right side,and left side there is only ddq

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

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

where $\hat{\mathrm{T}} = M^{-1}(q)\mathrm{T}.$

$$\hat{\mathrm{T}} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

$$\mathrm{T} = \begin{bmatrix} \mathrm{T}_1 \\ \mathrm{T}_2 \end{bmatrix} = M(q)\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}.$$

$$e(q_1) = q_{1f} - q_1,\ e(q_2) = q_{2f} - q_2,$$

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

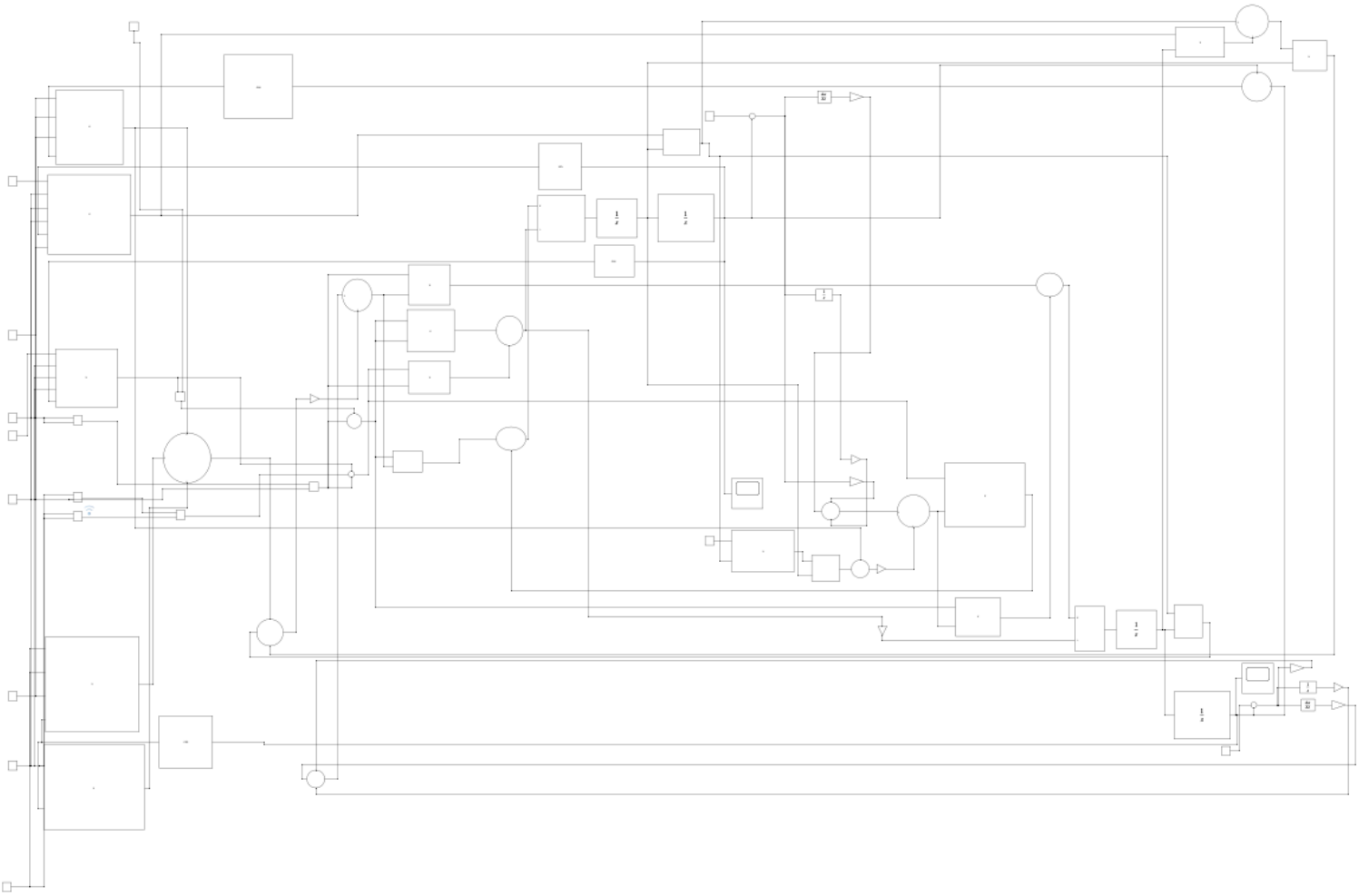
Here,I have assumed 6 state variable

$$u_1 = x_1,\ u_2 = x_2,\ u_3 = q_1,\ u_4 = q_2,\ u_5 = \dot{q}_1,\ u_6 = \dot{q}_2,$$

$$\dot{u}_1 = \dot{x}_1 = q_{1f} - u_3,\ \dot{u}_2 = \dot{x}_2 = q_{2f} - u_4,$$
$$\dot{u}_3 = \dot{q}_1 = u_5,\ \dot{u}_4 = \dot{q}_2 = u_6,$$
$$\dot{u}_5 = \ddot{q}_1 = \phi(t, u_1,\ u_2, u_3,\ u_4,\ u_5,\ u_6), \dot{u}_6 = \ddot{q}_2 = \psi(t, u_1,\ u_2, u_3,\ u_4,\ u_5,\ u_6)$$

$$\begin{bmatrix} \phi \\ \psi \end{bmatrix} = -M^{-1}(q)[C(q, \dot{q})\dot{q} + G(q)] + \begin{bmatrix} K_{P_1}(q_{1f} - u_3) - K_{D_1}u_5 + K_{I_1}u_1 \\ K_{P_2}(q_{2f} - u_4) - K_{D_2}u_6 + K_{I_2}u_2 \end{bmatrix},$$
$$\text{and } q = \begin{bmatrix} u_3 \\ u_4 \end{bmatrix}.$$

The **ode45 command** defined in MATLAB to solve the system of ordinary differential equations numerically. This command is based on the fourth-order Runge-Kutta method.



Simulink-

Here we have implements the equation in time domain using constant,product,adder,divison ,integartor,derivative blocks.

And we are plotting q1 and q2 on scope

EXPLANATION FOR THE PID CONTROL PART AFTER SIMULATIONS :-

1. System Modeling:
    - Equations of Motion: The first step involves deriving the equations of motion for the double pendulum system. This is done using the Lagrangian approach, resulting in a set of nonlinear ordinary differential equations (ODEs) that describe the dynamics of the system.
2. Control Objective:
    - Desired Position: The goal is to control the motion of the double pendulum system to achieve a specific position defined by the user. Due to the nonlinear nature of the system, achieving accurate control is challenging.
3. Control Strategy:
    - Computed Torque Control Method: The computed torque control method is chosen as the control strategy. This method involves calculating the required torque to achieve the desired motion. It typically compensates for the nonlinear dynamics of the system, making it suitable for controlling complex systems like the double pendulum.

Simulating Torque Application: The PID controller generates a control signal based on the combination of the proportional, integral, and derivative terms. This control signal is then used to simulate the application of torques to the double pendulum system.

Explanation of P,I,D components->

Proportional(P) Controller

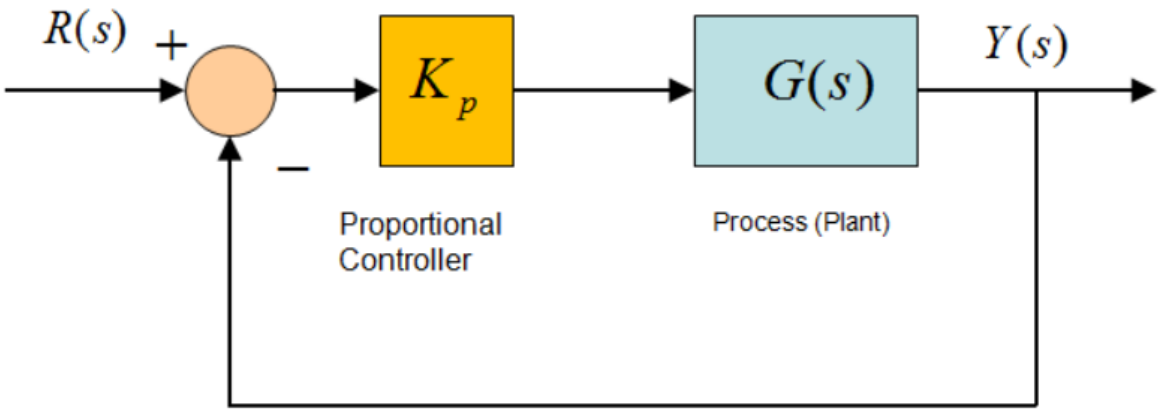
The proportional term is directly proportional to the current error, which is the difference between the desired setpoint and the actual process variable. The larger the error, the greater the proportional term.

The proportional term contributes to the controller's ability to reduce the steady-state error and respond quickly to changes in the system.

Impact on Stability:

- Pure P control can lead to steady-state error in the system.
- It may introduce oscillations and result in overshooting, especially in systems with significant inertia or delay.

The proportional controller produces an output, which is proportional to error signal.

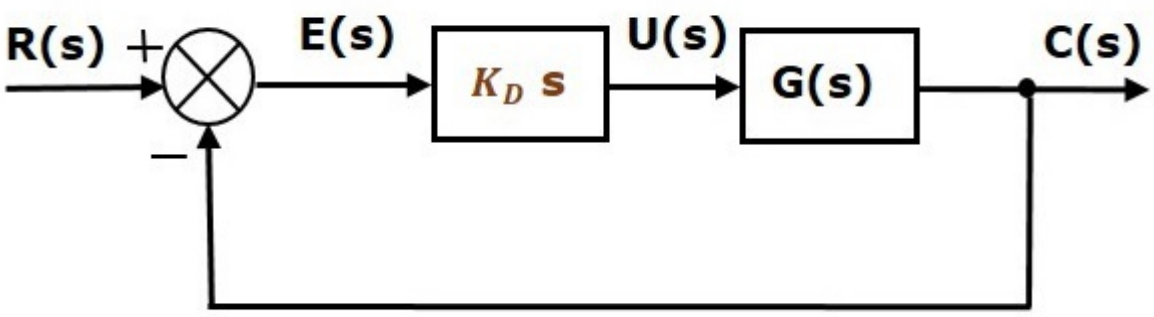


Derivative(D) Controller

- The derivative term is particularly useful in improving the stability of the system and reducing the effects of sudden changes or disturbances.

Unlike proportional and integral controllers, derivative controllers do not guide the system to a steady state. Because of this property, D controllers must be coupled with P, I or PI controllers to properly control the system.

- **Effect on System Response:**
  - The derivative term is proportional to the rate of change of the error.
  - It helps dampen oscillations, prevents overshooting, improving the transient response and stability.
- **Impact on Stability:**
  - A high derivative gain can lead to increased sensitivity to noise and disturbances.
  - It may introduce high-frequency noise in the control signal.



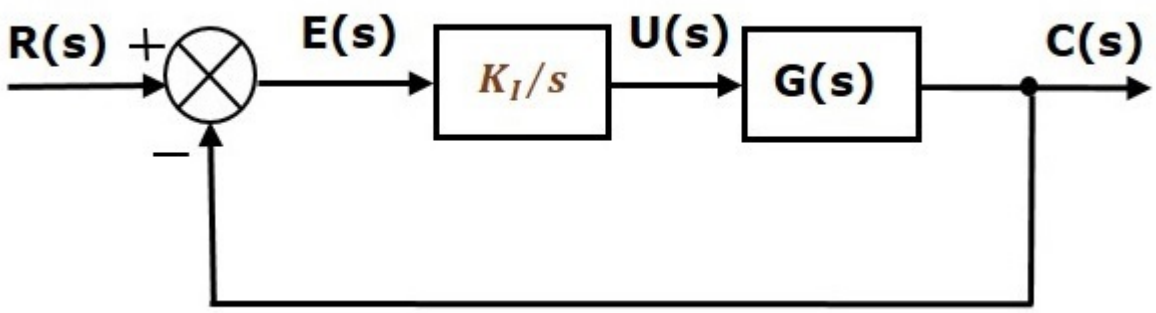
- **Integral(I) controller:**

  The integral term considers the accumulated error over time. It helps eliminate any residual steady-state error that may be present even when the proportional term is active.

Impact on Stability:

- While it eliminates steady-state error, an aggressive I term can introduce overshooting and oscillations.

The integral controller produces an output, which is integral of the error signal.



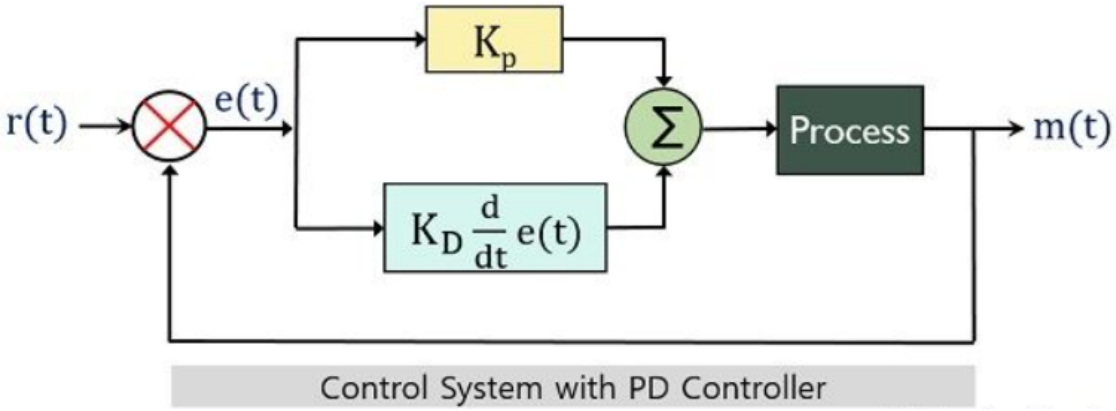
The integral controller is used to decrease the steady state error.

Let us now discuss about the combination of basic controllers.

Proportional-Derivative (PD) Controller:

- **Combined Effects:**
  - PD controllers produces an output, which is the combination of the outputs of proportional and derivative controllers.
  - Proportional action addresses steady-state error, while derivative action improves transient response.
- **Stability and Response:**
  - PD controllers can improve stability and damping, but they might not eliminate steady-state error.

The proportional derivative controller is used to improve the stability of control system without affecting the steady state error.

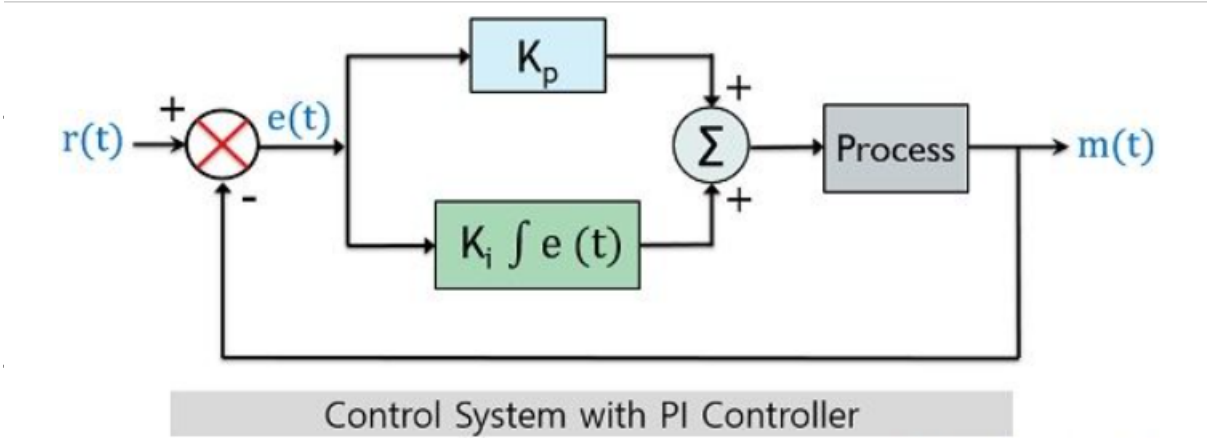


Control System with PD Controller

To *enhance the stability of the system without affecting the steady-state error*, a combination of proportional and derivative controllers is used.

Proportional Integral (PI) Controller

- **Combined Effects:**
  - PI controllers combine the benefits of the proportional and integral terms.
  - Proportional action provides a quick initial response, while integral action eliminates steady-state error.
- **Stability and Response:**
  - PI controllers are commonly used for stable control without introducing excessive overshooting or oscillations.



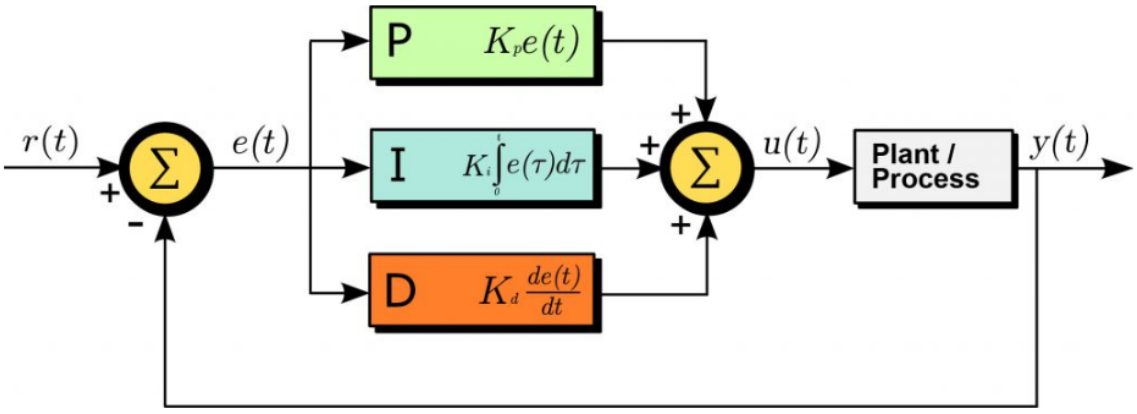
Control System with PI Controller

The proportional integral controller is used to decrease the steady state error without affecting the stability of the control system.

Proportional Integral Derivative (PID) Controller

- PID controllers combine proportional, integral, and derivative actions.
- They offer a balanced approach, addressing steady-state error, providing fast response, and improving stability.

The proportional integral derivative controller is used to improve the stability of the control system and to decrease steady state error.



Differences in responses-

1)PD-

i) KP is constant and increasing KD

KP1 = 10;

KP2 = 30;

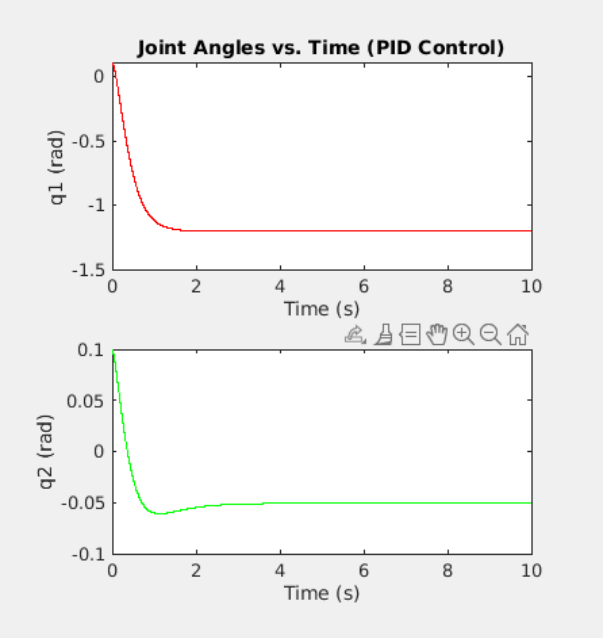
KI1 = 0;

KI2 = 0;

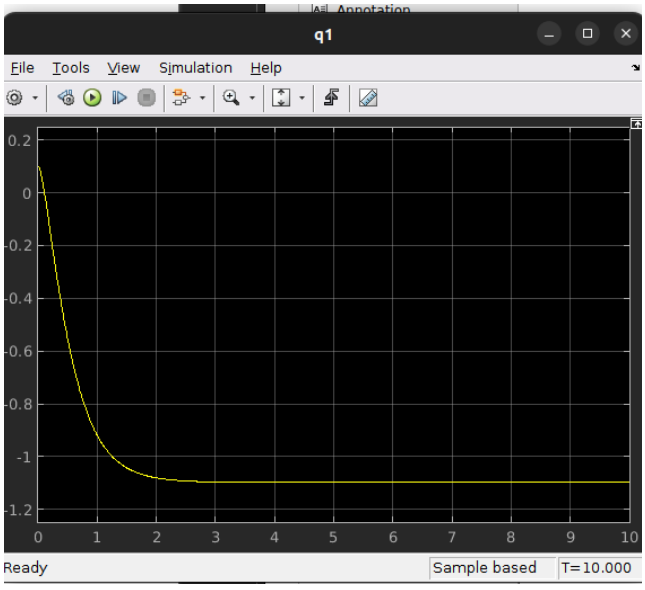
KD1 = 15;

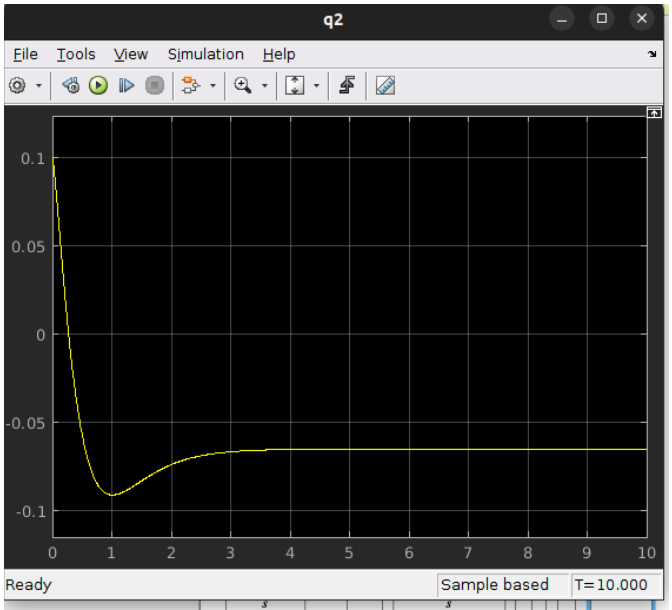
KD2 = 20;

Matlab Plots



Simulink Plots





KP1 = 10;

KP2 = 30;

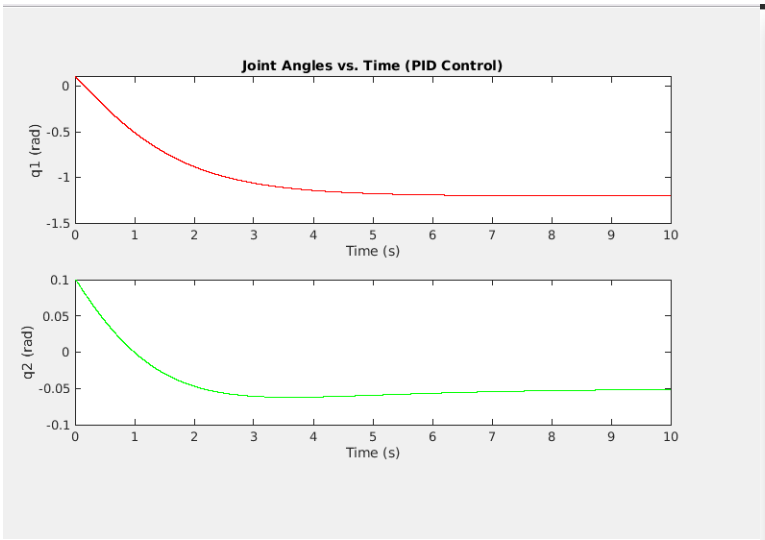
KI1 = 0;

KI2 = 0;

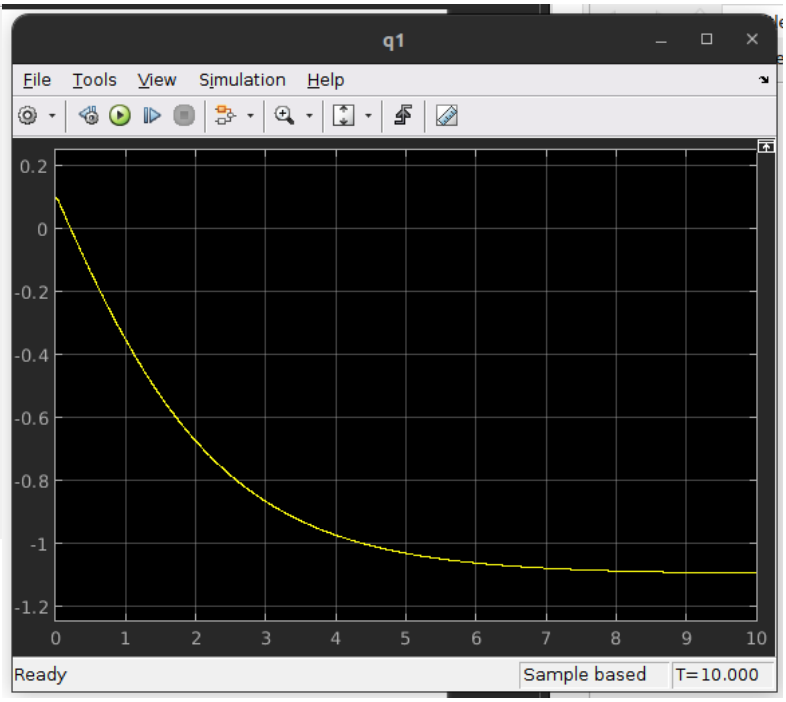
KD1 = 50;

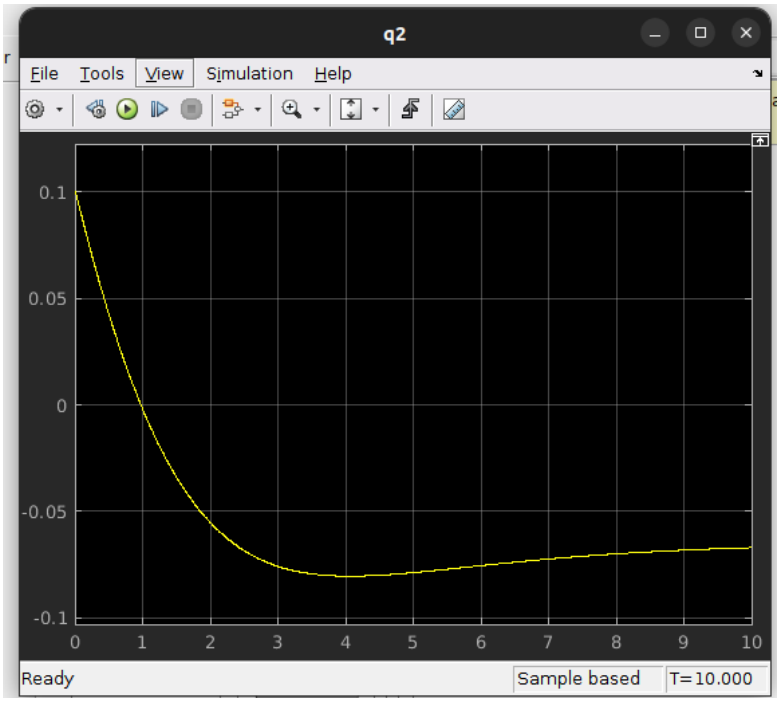
KD2 = 60

Matlab Plots



Simulink Plots





What is happening as we increasing KD keeping other thing fix?

The derivative gain, Kd, in a PID (Proportional-Integral-Derivative) controller plays a critical role in providing damping and controlling the rate of change of the error signal. Its primary function is to anticipate and counteract rapid changes in the error, which helps to achieve the following:

1. Damping: Kd provides damping to the system's response. It does this by looking at how fast the error signal is changing over time. When the error changes rapidly (i.e., when there's a sudden disturbance or a quick change in the setpoint), Kd responds proportionally to this change. By providing a damping effect, it reduces the tendency of the system to oscillate or overshoot the desired setpoint.
2. Overshoot Reduction: By reducing the rate of change of the error, Kd helps in minimizing overshoot. Overshoot occurs when the system's output exceeds the desired setpoint before settling down. Kd acts to counteract the acceleration of the error, thus limiting the extent to which the system overshoots.
3. Stability Enhancement: Kd contributes to the overall stability of the control system. It helps prevent excessive oscillations and ensures that the system settles smoothly and quickly when disturbances occur. Without the derivative term, systems can become more prone to oscillations and instability, especially in the presence of delays or rapid changes.

i) KD is constant and increasing KP

KP1 = 50;

KP2 = 60;

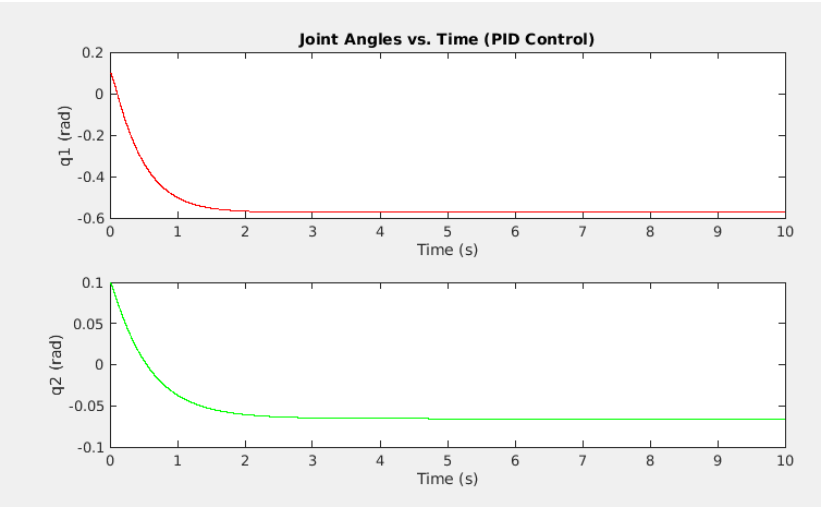
KI1 = 0;

KI2 = 0;

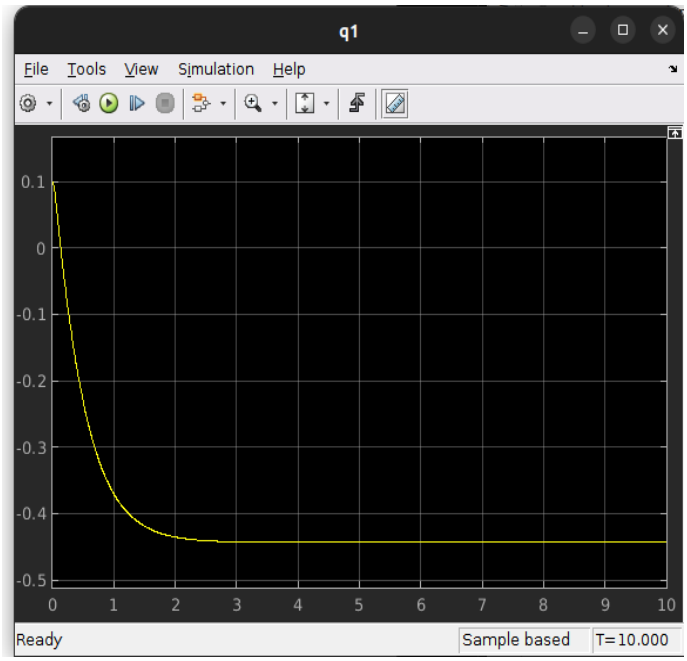
KD1 = 30;

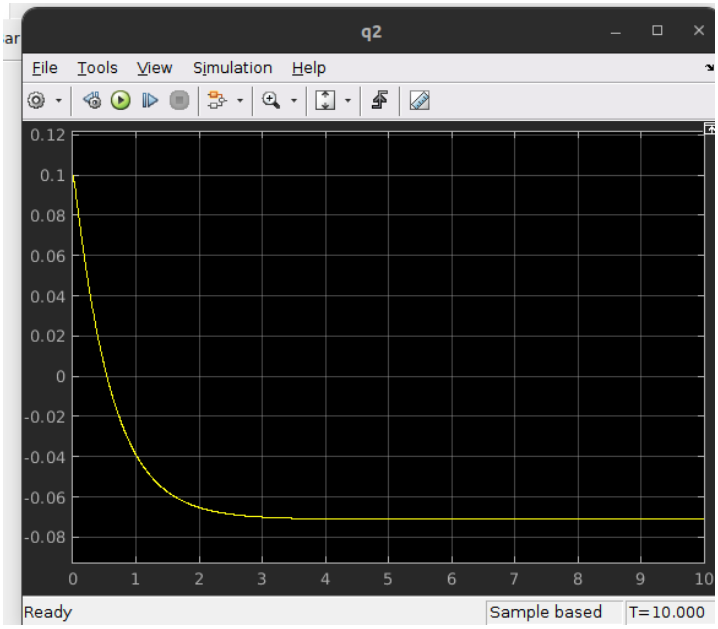
KD2 = 40;

Matlab Plots



Simulink Plots





KP1 = 150;

KP2 = 160;

KI1 = 0;

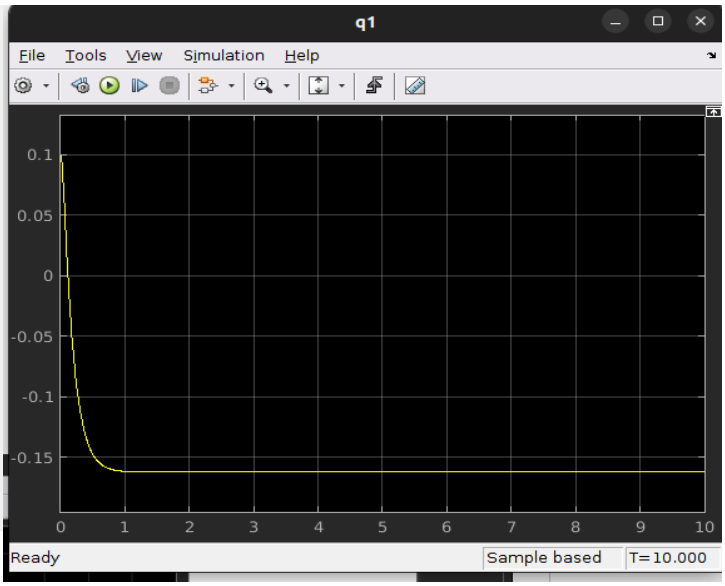
KI2 = 0;

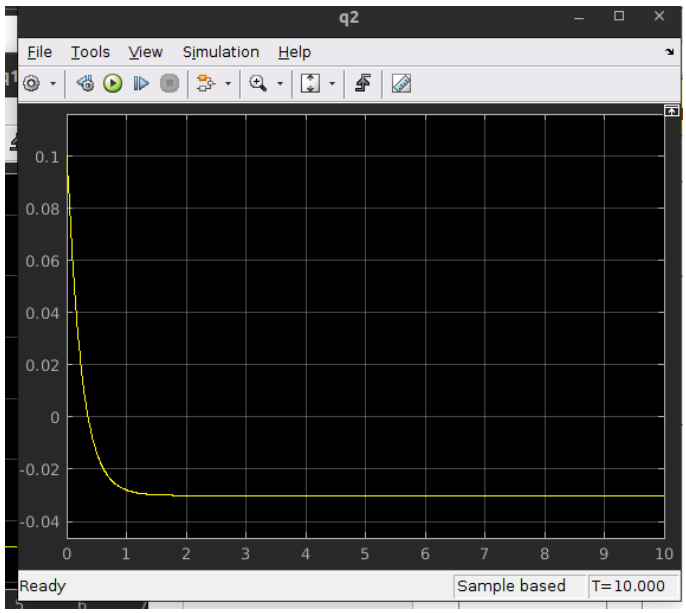
KD1 = 30;

KD2 = 40;

Matlab Plots

Simulink Plots





What is happening as we increasing KP keeping other thing fix?

The proportional gain, Kp, is one of the three parameters in a PID (Proportional-Integral-Derivative) controller, and its primary function is to determine the strength or influence of the proportional control action. Kp plays a crucial role in controlling a system's response to changes in error, which is the difference between the desired setpoint and the current process variable.

Here's what Kp does in a PID controller:

1. **Proportional Response:** Kp produces a control output that is directly proportional to the current error. The greater the error, the larger the proportional control action. In other words, Kp determines how much the controller responds to deviations from the setpoint. A higher Kp value results in a stronger proportional response.
2. **Error Correction:** The proportional term aims to bring the system closer to the desired setpoint by reducing the error. When the error is large, Kp causes the controller to take more significant corrective action, driving the process variable closer to the setpoint.
3. **Fast Response:** Increasing Kp makes the controller respond more quickly to changes in the error. It leads to a faster initial response when an error occurs, helping the system reach the setpoint more rapidly.
4. **Sensitivity to Error:** The sensitivity of the control system to error is directly affected by Kp. A higher Kp makes the system more sensitive and responsive to deviations from the setpoint, which can be beneficial for systems that require a quick response.

2)PI Response

i) KP is constant and increasing KI

KP1 = 50;

KP2 = 60;

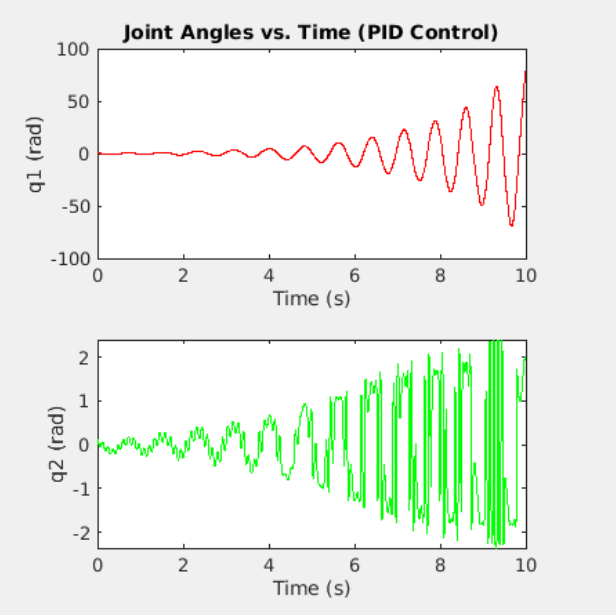
KI1 = 50;

KI2 = 60;

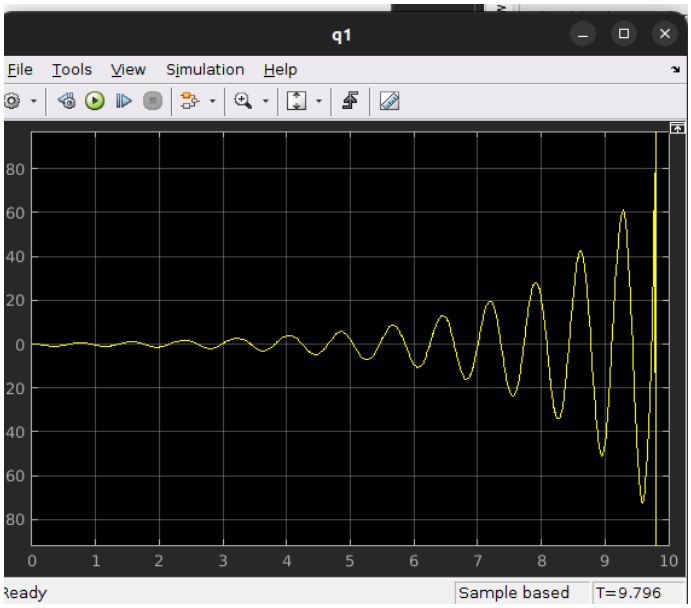
KD1 = 0;

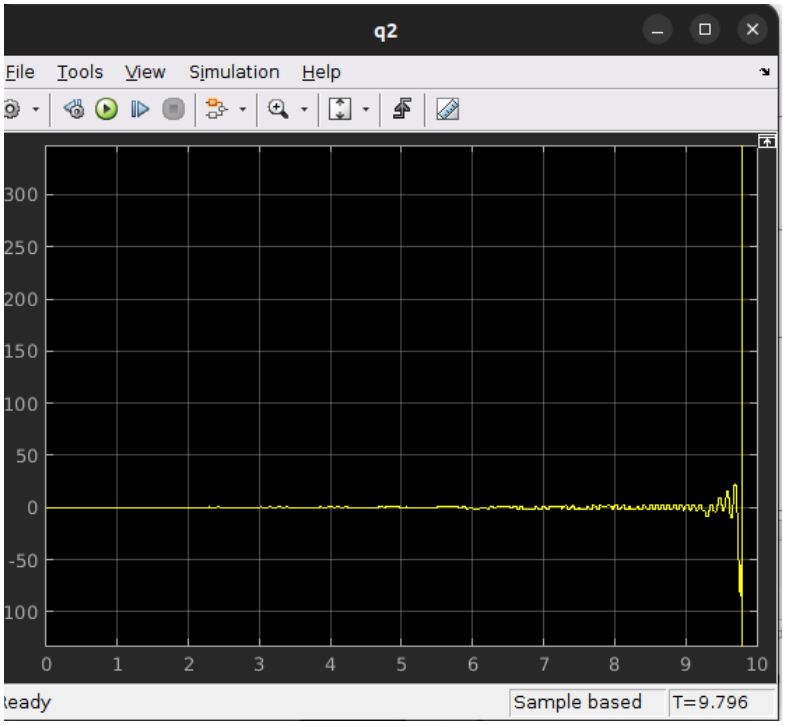
KD2 = 0;

Matlab Plots



Simulink Plots





KP1 = 50;

KP2 = 60;

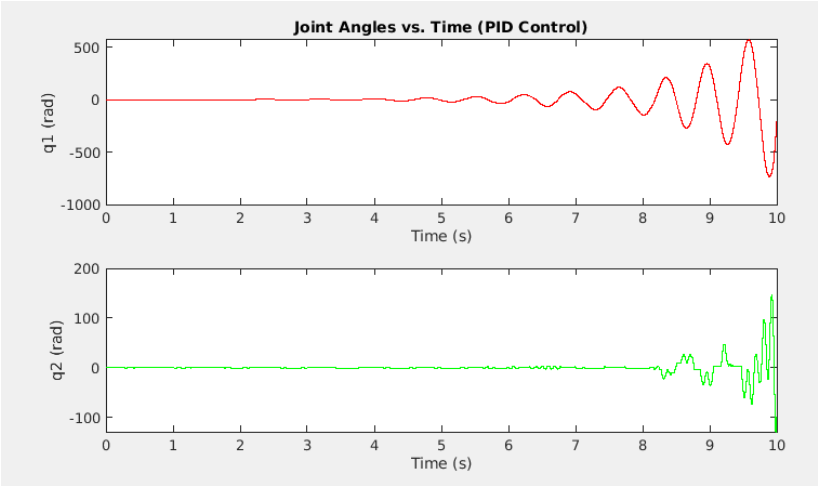
KI1 = 70;

KI2 = 80;

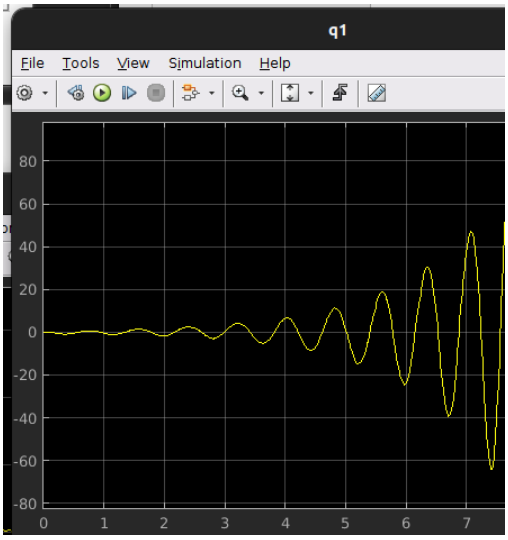
KD1 = 0;

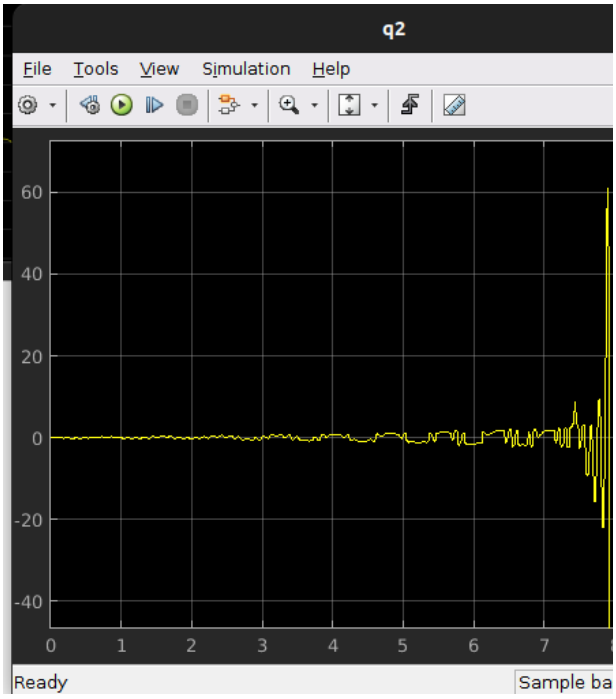
KD2 = 0;

Matlab Plots



Simulink Plots





What is happening as we increasing KI keeping other thing fix?

When you increase the integral gain (Ki) in a PID (Proportional-Integral-Derivative) controller while keeping other parameters (Kp and Kd) fixed, several things happen:

1. Increased Steady-State Correction: The primary role of the integral term (Ki) is to eliminate steady-state error by accumulating and integrating the error signal over time. Increasing Ki makes the integral term more influential in the control action. As a result:
    - The controller becomes more effective at correcting any long-term or persistent errors between the desired setpoint and the actual process variable.
    - Steady-state error is reduced more aggressively. Steady-state error is the error that remains after the system has settled to a new equilibrium. A higher Ki value can reduce this error faster.
2. Slower Transient Response: While increasing Ki reduces steady-state error, it can also slow down the transient response of the system. The transient response refers to how quickly the system reacts to changes or disturbances. With a higher Ki, the controller places more emphasis on correcting past errors, which may slow down the response to new errors.

ii) KI is constant and increasing KP

KP1 = 100;

KP2 = 200;

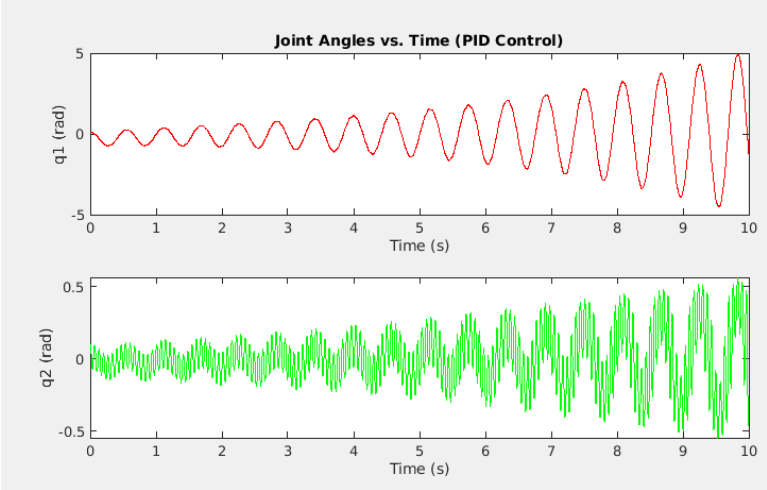
KI1 = 50;

KI2 = 60;

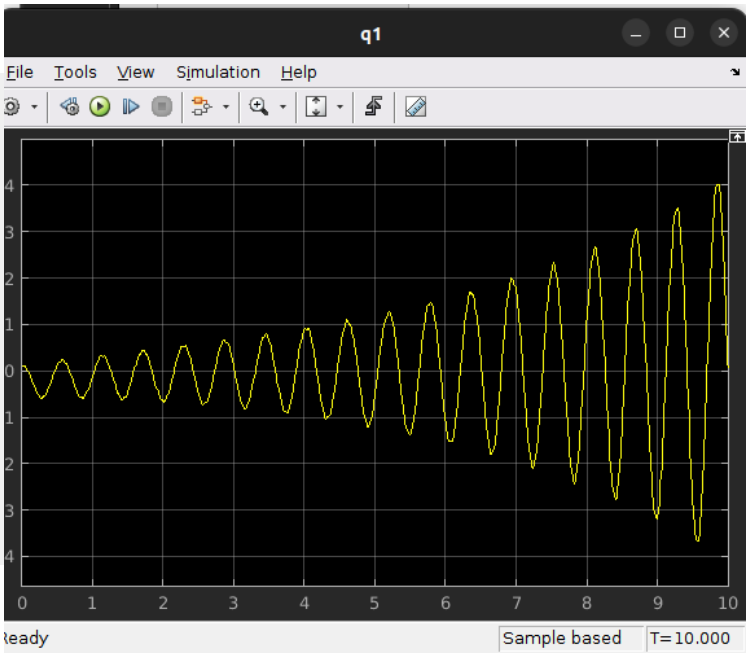
KD1 = 0;

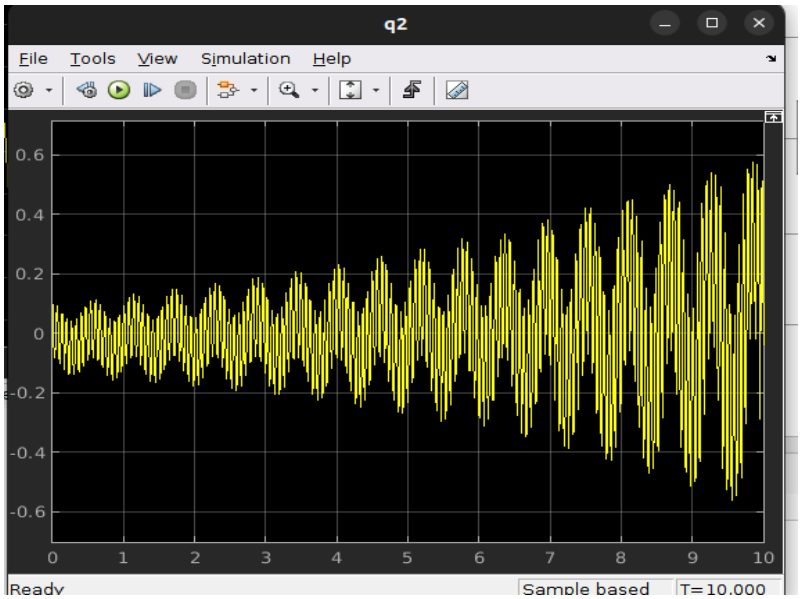
KD2 = 0;

Matlab Plots



Simulink Plots





KP1 = 300;

KP2 = 400;

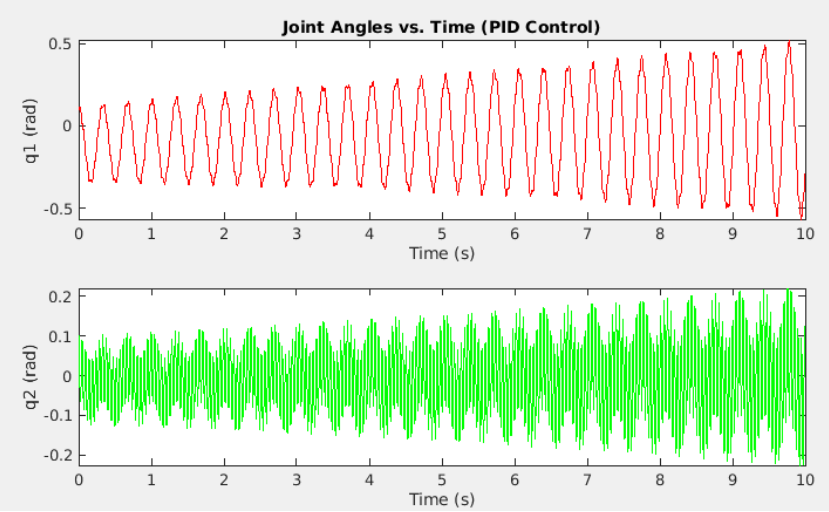
KI1 = 50;

KI2 = 60;

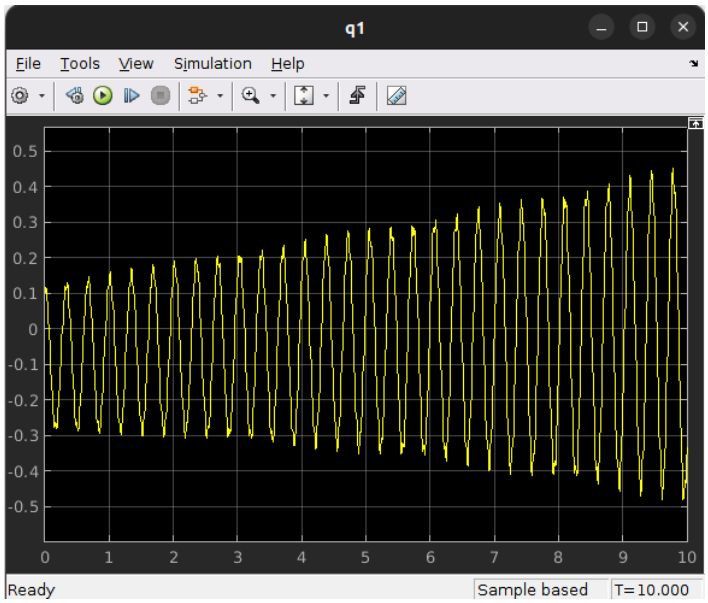
KD1 = 0;

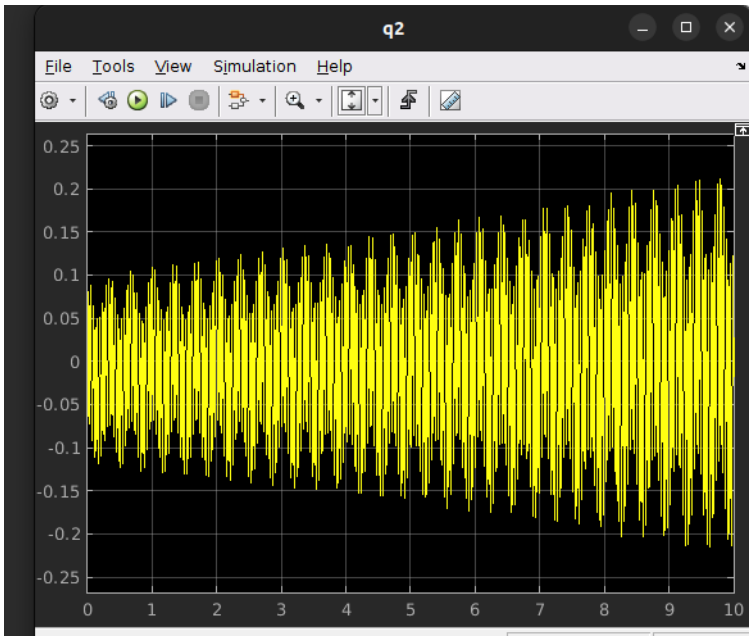
KD2 = 0;

Matlab Plots



Simulink Plots





What is happening as we increasing KP keeping other thing fix?

When you increase KP (Proportional Gain) while keeping KI (Integral Gain) fixed and KD (Derivative Gain) equal to zero in a PID (Proportional-Integral-Derivative) controller, you are essentially operating with a pure proportional controller. This means that the control action will be solely based on the current error signal, without considering the integral or derivative aspects of the error. Here's what will happen to the system's output:

1. Faster Response: Increasing KP will make the controller respond more aggressively to the current error. The larger the error, the stronger the control action applied. This results in a faster initial response to disturbances or setpoint changes.
2. Potential Overshoot: With a higher KP, there is a greater chance of overshooting the desired setpoint. The aggressive response can cause the system's output to go beyond the target value before settling down.
3. Reduced Stability: A purely proportional controller can lead to reduced system stability, especially in the presence of noise or system uncertainties. It may exhibit oscillatory behavior and might not be able to completely eliminate steady-state errors.
4. No Integral or Derivative Action: Since KI is fixed at a constant value, there is no integral action to eliminate steady-state errors over time, and since KD is set to zero, there is no damping effect provided by derivative action to reduce overshoot and oscillations.

III)PID Response

Here we are now taking plotting for different values of P,I,D

1) Kp=non-zero,KI=KD=0

KP1 = 100;

KP2 = 200;

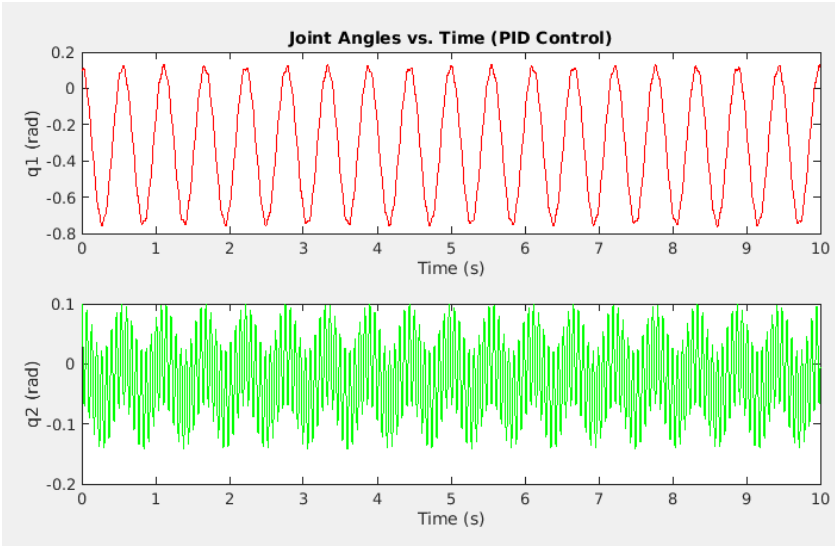
KI1 = 0;

KI2 = 0;

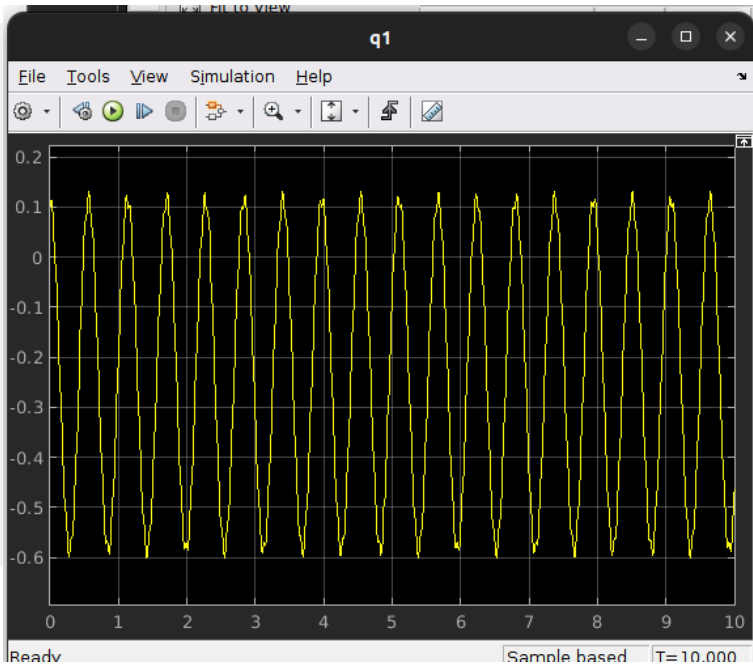
KD1 = 0;

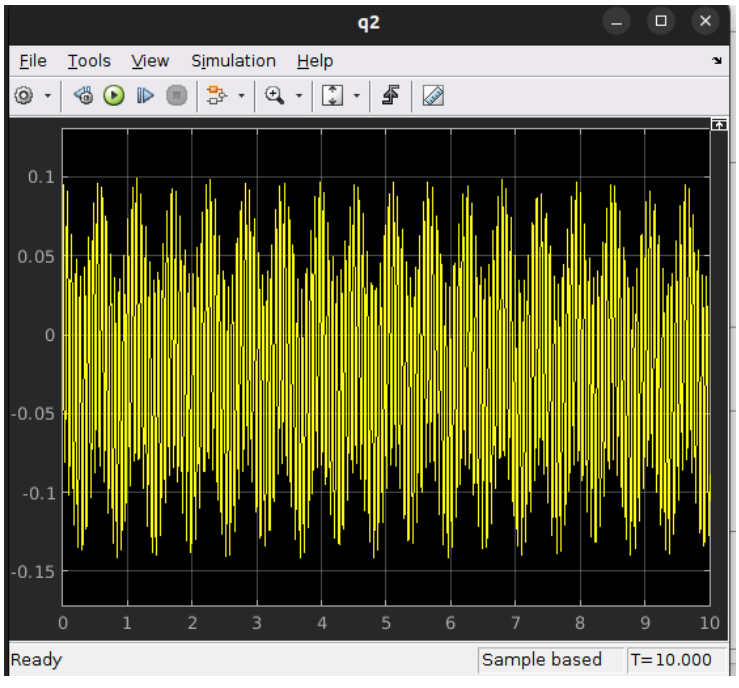
KD2 = 0;

Matlab Plots



Simulink Plots

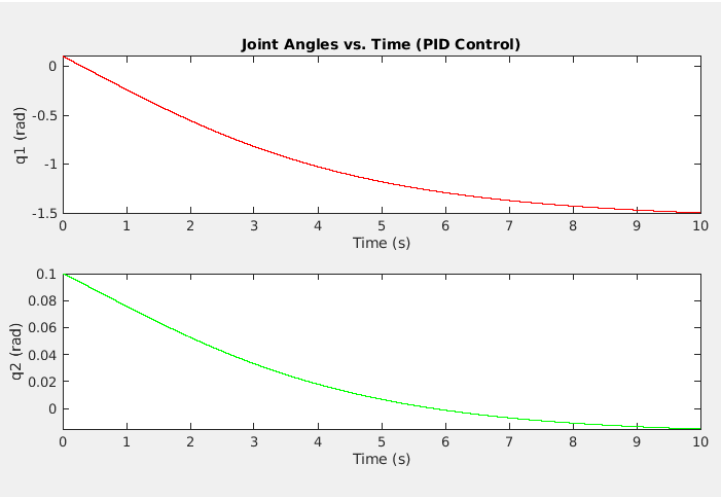




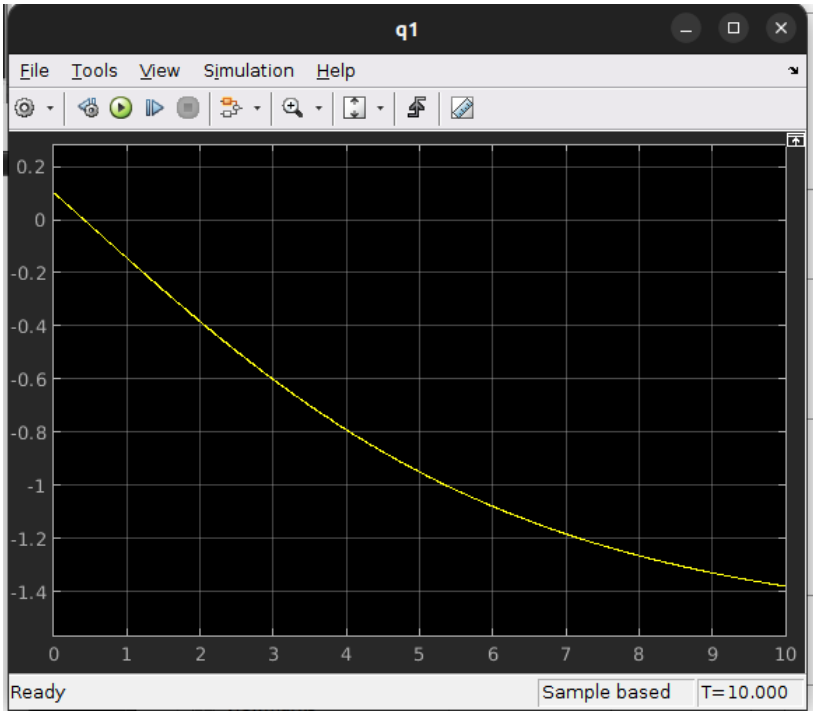
2) KD=non-zero,KI=KP=0

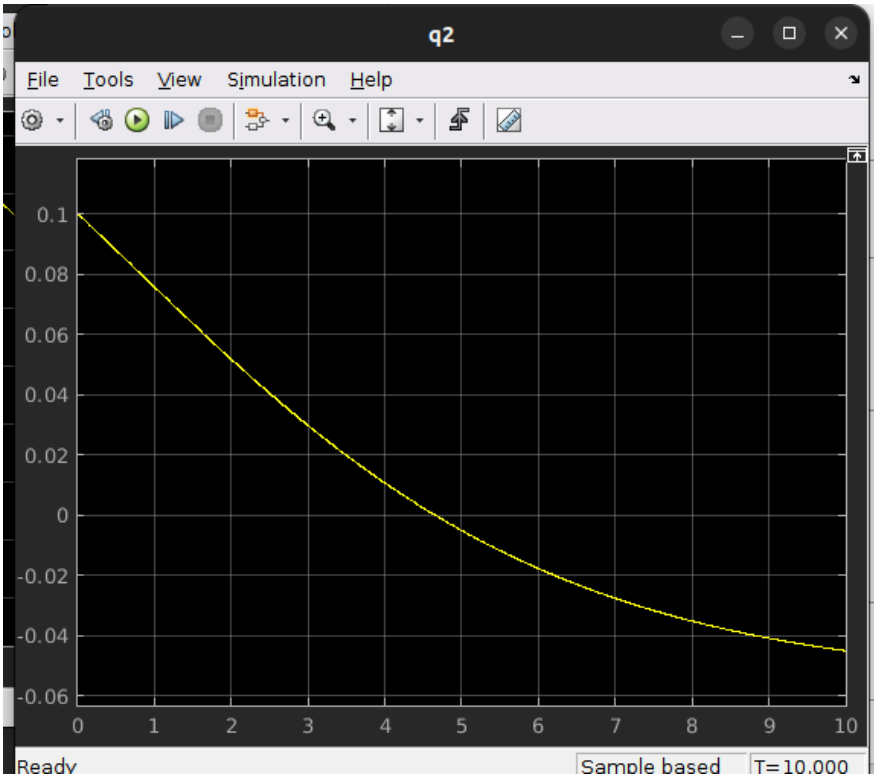
```
KP1 = 0;

KP2 = 0;

KI1 = 0;

KI2 = 0;

KD1 = 100;

KD2 = 200;
```

Matlab Plots



Simulink Plots





2) KI=non-zero,KI=KD=0

KP1 = 0;

KP2 = 0;

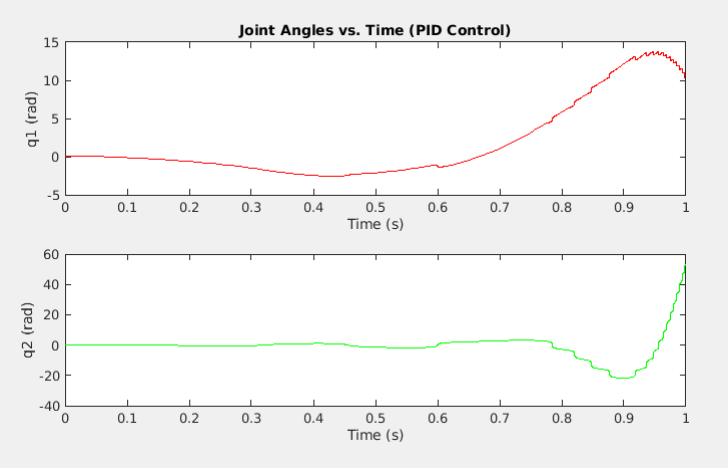
KI1 = 100;

KI2 = 200;

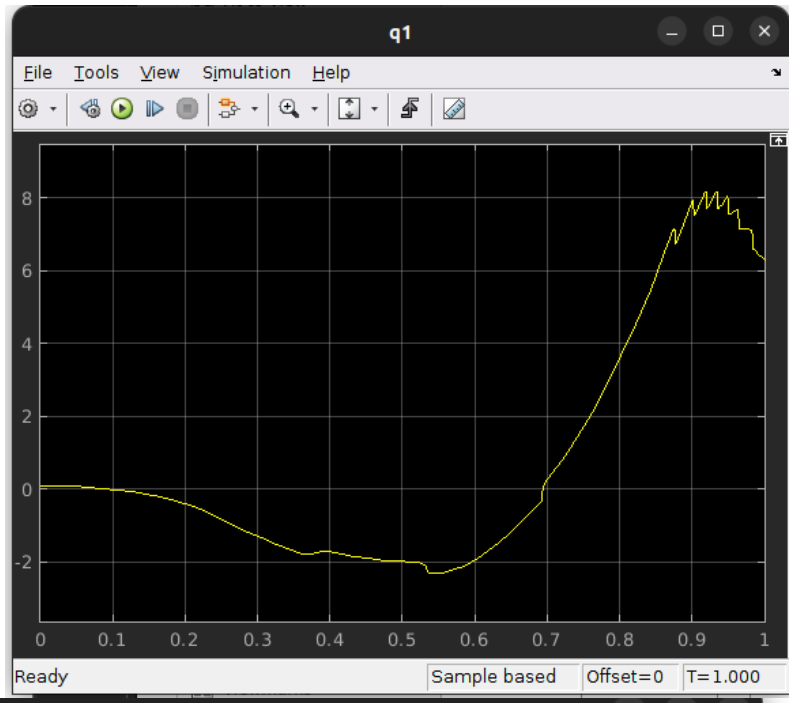
KD1 = 0;

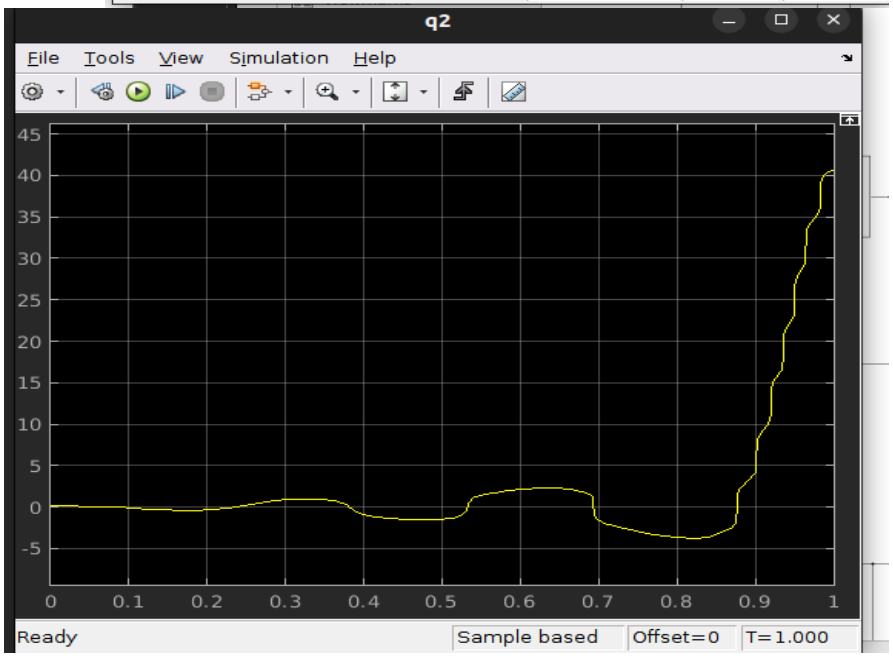
KD2 = 0;

Matlab Plots



Simulink Plots





4) ALL Kp,KD,Ki=non-zero

KP1 = 150;

KP2 = 300;

KI1 = 120;

KI2 = 100;

KD1 = 150;

KD2 = 130;