Systems Thinking Project

2 Link Robotic Manipulator

Author: **Deepak Pandey**
ID: 2023102053
E-mail:
deepak.pandey@students.iiit.ac.in

Author: **Snehil Sanjog**
ID: 2023102051
E-mail:
snehil.sanjog@students.iiit.ac.in

Author: **Gautam Gandhi**
ID: 2023102059
E-mail:
gautam.gandhi@students.iiit.ac.in

Author: **Sai Rithvik**
ID: 2023102060
E-mail:
sairithvik.achutuni@students.iiit.ac.in

Author: **Roshan Kondabattini**
ID: 2023102061
E-mail: *kondabattini.roshan@students.iiit.ac.in*

Author: **Manas Inamdar**
ID: 2023102052
E-mail:
manas.inamdar@students.iiit.ac.in

Author: **Vedant Tejas**
ID: 2023112018
E-mail:
vedant.tejas@research.iiit.ac.in

Author: **Aryan Agrawal**
ID: 2023102050
E-mail:
aryan.agrawal@students.iiit.ac.in

September 30, 2024

Contents

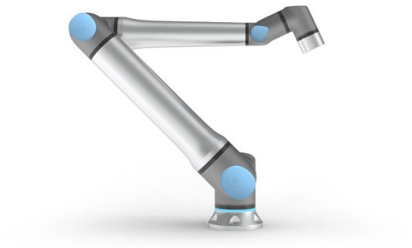

Figure 1: Robotic Arm

1 Introduction

The 2-link manipulator is a fundamental model in robotics, providing both simplicity and a foundation for comprehending more intricate systems.

1.1 Conceptual Modelling

The robotic arm can be modeled as two mass-less rods, each with a mass at its endpoint. In this representation, one rod-mass pair functions as the arm, while the other serves as the forearm. Additional masses are placed at the elbow joint and the end of the forearm. This configuration allows us to couple the two systems and examine the overall dynamics of the arm, which can oscillate freely within a plane.

1.2 Theoretical Framework

The dynamics of the manipulator can be thoroughly described by formulating its Lagrangian, which incorporates both potential and kinetic energies. By applying the Euler-Lagrange equation, we can derive the torques acting on each link. We subsequently utilize PI, PD, or PID controllers to simulate the system's behavior under varying torque applications.

1.3 Methodology

The motion equation for this 2-link manipulator is a non-linear differential equation. We solve it numerically and use MATLAB for simulation and visualization of the control mechanisms.

1.4 Objectives

- To model the 2-link robotic manipulator using state functions.
- To apply and compare PI, PD, and PID controllers for system control.
- To simulate the manipulator's dynamics using MATLAB.

2 Dynamics of a 2-link Manipulator

2.1 Overview

A two-link manipulator can be modeled as a system consisting of two massless rods, denoted by lengths L1 and L2, with masses M1 and M2 located at the ends of each rod, respectively. In this model, M1 represents the elbow joint, while M2 corresponds to the forearm's end.

If the robotic arm is mounted on a surface, let q1 be the angle formed between L1 and the surface, and q2 the angle between L2 and L1. This system can be depicted in a two-dimensional plane, where the x and y coordinates of the masses M1 and M2 can be defined accordingly.

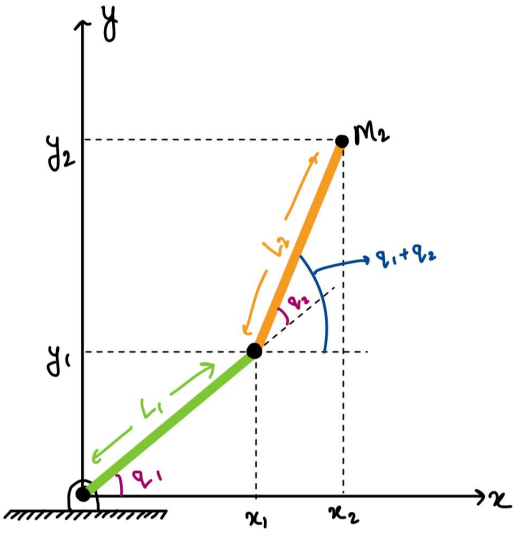

Figure 2: Simplified Model of a two-link planar robot manipulator

The above system can be described using a standard form for robotic systems :

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

Where:

- $M(q)$ is the mass/inertia matrix.
- $C(q,\dot{q})$ is the Coriolis and centrifugal force matrix.
- $G(q)$ is the gravitational force matrix.
- τ is the applied torque.
- $q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$ is the vector of joint angles.

2.1.1 Mass Matrix $M(q)$

$$M(q) = \begin{bmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{bmatrix}$$

Where the elements of $M(q)$ are given by:

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

2.1.2 Coriolis and Centrifugal Matrix $C(q, \dot{q})$

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

2.1.3 Gravitational Matrix $G(q)$

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

2.1.4 Parameter Values

The following parametric values are used for simulation:

$$m_1 = 10\,\text{kg}, \quad m_2 = 5\,\text{kg}, \quad l_1 = 0.2\,\text{m}, \quad l_2 = 0.1\,\text{m}, \quad g = 9.81\,\text{m/s}^2$$

The initial joint angles are given by:

$$q_1(0) = 0.1\,\text{rad}, \quad q_2(0) = 0.1\,\text{rad}$$

2.2 Objective

The objective is to bring the joint angles from their initial positions $q_1(0) = 0.1\,\text{rad}$ and $q_2(0) = 0.1\,\text{rad}$ to the final position $q_1 = 0$ and $q_2 = 0$.

3 State Space Equation

Let us define the following state variables :

$$x_1 = q_1(t), \quad x_2 = q_2(t), \quad x_3 = \dot{q}_1(t), \quad x_4 = \dot{q}_2(t).$$

We also have the following system dynamics equation, $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau$

Let us define the following matrices.

$$\dot{X} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix}$$

The input matrix (torque)

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

Let us define two more matrices as :

$$S = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}$$

Writing the system equation in matrix form, we can rearrange this equation

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

to

$$\dot{S}_1 = M^{-1}(q)(-C(S, S_1)S_1 - G(S) + \tau)$$

The above equation essentially represents the state-space model required to analyse the two-link system.

4 Control Design of a 2-link Manipulator

The objective of control of 2-link robotic manipulators is to reduce position and velocity errors. The most popular controllers for this purpose are :

- Proportional-Derivative (PI)
- Proportional-Integral (PD)
- Proportional-Integral-Derivative (PID)

We characterise the effect of each block in these controllers by the use of *gain constants*.

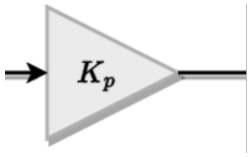

Figure 3: Proportional Block

- **Proportional Gain** (K_p): This is the constant which is multiplied to error as a control system.
- **Integral gain** (K_i): The integral term is concerned with the accumulation of past errors. This integral function is then multiplied by K_i.

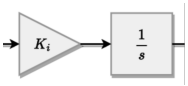

Figure 4: Integrator Block

- **Derivative Gain** (K_d): The derivative gain is used as some sort of a prediction of future based on the rate of change. The contribution of this block is characterised by K_d.

Figure 5: Derivative Block

By changing these values (K_p, K_d, K_i) we can create an effective controller to gives us the least error.

4.1 Using a PI controller

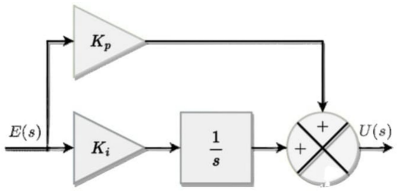

Figure 6: Block diagram for PI controller

From above relations, we can determine the equation for the controller as:

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

Using PI control, we get:

$$\tau = M(q)\left[k_p e + k_i \int e dt\right] \tag{2}$$

where $e = q_f - q$

Finally :

$$\begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix} = M(q)\begin{bmatrix}K_P(q_{1_f} - q_1) + K_I\int(q_{1_f} - q_1)dt \\ K_P(q_{2_f} - q_2) + K_I\int(q_{2_f} - q_2)dt\end{bmatrix} \tag{3}$$

4.2 Using a PD controller

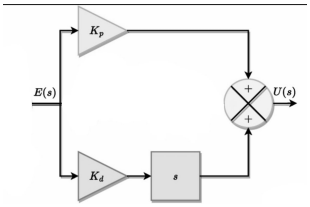

Figure 7: Block diagram for PD controller

We had,

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

Using PI control, we get:

$$\tau = M(q)\left[k_p e + k_d \dot{e}\right]$$

where $e = q_f - q$

Finally,

$$\begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix} = M(q)\begin{bmatrix}K_P(q_{1_f} - q_1) - K_d\dot{q_1} \\ K_P(q_{2_f} - q_2) - K_d\dot{q_2}\end{bmatrix} \tag{4}$$

4.3 Using a PID controller

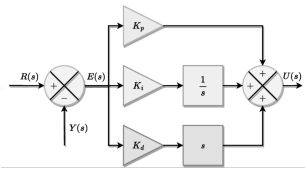

Figure 8: Block diagram for a PID Controller

From above relations, we can determine the equation for the controller as:

$$\ddot{q} = M^{-1}(q)\left[\tau - C(q,\dot{q})\dot{q} - G(q)\right] \tag{5}$$

Using PID Control, we get:

$$\tau = M(q)\left[k_p e + k_i \int e dt + k_d \dot{e}\right] \tag{6}$$

where $e = q_f - q$.

Finally :

$$\begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix} = M(q)\begin{bmatrix}K_P(q_{1_f} - q_1) + K_I\int(q_{1_f} - q_1)dt - K_d\dot{q_1} \\ K_P(q_{2_f} - q_2) + K_I\int(q_{2_f} - q_2)dt - K_d\dot{q_2}\end{bmatrix} \tag{7}$$

5 MATLAB Implementation and Simulink

5.1 Matlab Codes and Simulink Block Representation

Listing 1: PI Code

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

% Initial State Variables
e1 = 0;
e2 = 0;
positions = [0.1, 0.1]; % q1 and q2
velocities = [0, 0]; % q1_dot and q2_dot
initial_q = [e1, e2, positions, velocities];

% PID Controller gains
kp = 100;
ki = 1;

% Final angles
qfinal = [0; 0];

options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);
[time, s] = ode45(@(t, q) pid(t, q, qfinal, kp,ki), [0,30], initial_q,
    options);

figure;
% subplot(2, 1, 1);
plot(time, s(:, 3));
xlabel("Time");
ylabel("q_1");
hold on;
% subplot(2, 1, 2);
plot(time, s(:, 4), 'r');
xlabel("Time");
ylabel("q_2");

sgtitle('Joint Angles vs Time - PI Controller');

% PID Controller Function
function q_derivative = pid(~, q, qfinal, kp,ki)
    % Constants
    l1 = 0.2;
    l2 = 0.1;
    m1 = 10;
    m2 = 5;
    g = 9.81;

    % Error calculation
    e1 = qfinal(1) - q(3);
    e2 = qfinal(2) - q(4);

```

```
    % Mass Matrix
    M11 = (m1 + m2) * l1^2 + m2 * l2 * (l2 + 2 * l1 * cos(q(4)));
    M22 = m2 * l2^2;
    M12 = m2 * l2 * (l2 + l1 * cos(q(4)));
    M21 = M12;
    M = [M11, M12; M21, M22];

    % Torque
   f1 = kp * e1 + ki * q(1);
   f2 = kp * e2 + ki * q(2);
   f=[f1;f2];

    % Coriolis Matrix
    c11 = -m2 * l1 * l2 * sin(q(4)) * q(5);
    c12 = -m2 * l1 * l2 * sin(q(4)) * (q(5) + q(6));
    c21 = 0;
    c22 = m2 * l1 * l2 * sin(q(4)) * q(6);
    C = [c11, c12; c21, c22];

    % Gravitational Matrix
    G1 = m1 * l1 * g * cos(q(3)) + m2 * g * (l2 * cos(q(3) + q(4)) + l1
        * cos(q(3)));
    G2 = m2 * l2 * g * cos(q(3) + q(4));
    G = [G1; G2];

    % State Derivatives
    dq = [q(5); q(6)];
    M_inv = inv(M);
    W = (M_inv) * ((-C * dq - G)) + f;

    % Next state
    q_derivative = zeros(6, 1);
    q_derivative(1) = e1;
    q_derivative(2) = e2;
    q_derivative(3) = q(5);
    q_derivative(4) = q(6);
    q_derivative(5:6) = W;
end
```

Listing 2: PD Code

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

% Initial State Variables
e1 = 0;
e2 = 0;
positions = [0.1, 0.1]; % q1 and q2
velocities = [0, 0]; % q1_dot and q2_dot
initial_q = [e1, e2, positions, velocities];

% PID Controller gains
kp = 89;
```

```
kd = 89;

% Final angles
qfinal = [0; 0];

options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);
[time, s] = ode45(@(t, q) pid(t, q, qfinal, kp,kd), [0,30], initial_q,
    options);

figure;
% subplot(2, 1, 1);
plot(time, s(:, 3));
xlabel("Time");
ylabel("q_1");
hold on;
% subplot(2, 1, 2);
plot(time, s(:, 4), 'r');
xlabel("Time");
ylabel("q_2");

sgtitle('Joint Angles vs Time - PD Controller');

% PD Controller Function
function q_derivative = pid(~, q, qfinal, kp,kd)
    % Constants
    l1 = 0.2;
    l2 = 0.1;
    m1 = 10;
    m2 = 5;
    g = 9.81;

    % Error calculation
    e1 = qfinal(1) - q(3);
    e2 = qfinal(2) - q(4);

    % Mass Matrix
    M11 = (m1 + m2) * l1^2 + m2 * l2 * (l2 + 2 * l1 * cos(q(4)));
    M22 = m2 * l2^2;
    M12 = m2 * l2 * (l2 + l1 * cos(q(4)));
    M21 = M12;
    M = [M11, M12; M21, M22];

    % Torque
    f1 = kp * e1 - kd * q(5);
    f2 = kp * e2 - kd * q(6);
    f=[f1;f2];

    % Coriolis Matrix
    c11 = -m2 * l1 * l2 * sin(q(4)) * q(5);
    c12 = -m2 * l1 * l2 * sin(q(4)) * (q(5) + q(6));
    c21 = 0;
    c22 = m2 * l1 * l2 * sin(q(4)) * q(6);
    C = [c11, c12; c21, c22];

    % Gravitational Matrix
    G1 = m1 * l1 * g * cos(q(3)) + m2 * g * (l2 * cos(q(3) + q(4)) + l1
        * cos(q(3)));
```

```
    G2 = m2 * l2 * g * cos(q(3) + q(4));
    G = [G1; G2];

    % State Derivatives
    dq = [q(5); q(6)];
    M_inv = inv(M);
    W = (M_inv) * ((-C * dq - G)) + f;

    % Next state
    q_derivative = zeros(6, 1);
    q_derivative(1) = e1;
    q_derivative(2) = e2;
    q_derivative(3) = q(5);
    q_derivative(4) = q(6);
    q_derivative(5:6) = W;
end
```

Listing 3: PID Code

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

% Initial State Variables
e1 = 0;
e2 = 0;
positions = [0.1, 0.1]; % q1 and q2
velocities = [0, 0]; % q1_dot and q2_dot
initial_q = [e1, e2, positions, velocities];

% PID Controller gains
kp = 89;
ki = 89;
kd = 89;

% Final angles
qfinal = [0; 0];

options = odeset('RelTol', 1e-3, 'AbsTol', 1e-6);
[time, s] = ode45(@(t, q) pid(t, q, qfinal, kp,kd,ki), [0,30], initial_q
    , options);

figure;
% subplot(2, 1, 1);
plot(time, s(:, 3));
xlabel("Time");
ylabel("q_1");
hold on;
% subplot(2, 1, 2);
plot(time, s(:, 4), 'r');
xlabel("Time");
ylabel("q_2");

sgtitle('Joint Angles vs Time - PID Controller');
```

```
38
39 % PID Controller Function
40 function q_derivative = pid(~, q, qfinal, kp,kd,ki)
41     % Constants
42     l1 = 0.2;
43     l2 = 0.1;
44     m1 = 10;
45     m2 = 5;
46     g = 9.81;
47
48     % Error calculation
49     e1 = qfinal(1) - q(3);
50     e2 = qfinal(2) - q(4);
51
52     % Mass Matrix
53     M11 = (m1 + m2) * l1^2 + m2 * l2 * (l2 + 2 * l1 * cos(q(4)));
54     M22 = m2 * l2^2;
55     M12 = m2 * l2 * (l2 + l1 * cos(q(4)));
56     M21 = M12;
57     M = [M11, M12; M21, M22];
58
59     % Torque
60     f1 = kp * e1 - kd * q(5) + ki * q(1);
61     f2 = kp * e2 - kd * q(6) + ki * q(2);
62     f=[f1;f2];
63
64     % Coriolis Matrix
65     c11 = -m2 * l1 * l2 * sin(q(4)) * q(5);
66     c12 = -m2 * l1 * l2 * sin(q(4)) * (q(5) + q(6));
67     c21 = 0;
68     c22 = m2 * l1 * l2 * sin(q(4)) * q(6);
69     C = [c11, c12; c21, c22];
70
71     % Gravitational Matrix
72     G1 = m1 * l1 * g * cos(q(3)) + m2 * g * (l2 * cos(q(3) + q(4)) + l1
           * cos(q(3)));
73     G2 = m2 * l2 * g * cos(q(3) + q(4));
74     G = [G1; G2];
75
76     % State Derivatives
77     dq = [q(5); q(6)];
78     M_inv = inv(M);
79     W = (M_inv) * ((-C * dq - G)) + f;
80
81     % Next state
82     q_derivative = zeros(6, 1);
83     q_derivative(1) = e1;
84     q_derivative(2) = e2;
85     q_derivative(3) = q(5);
86     q_derivative(4) = q(6);
87     q_derivative(5:6) = W;
88 end
```

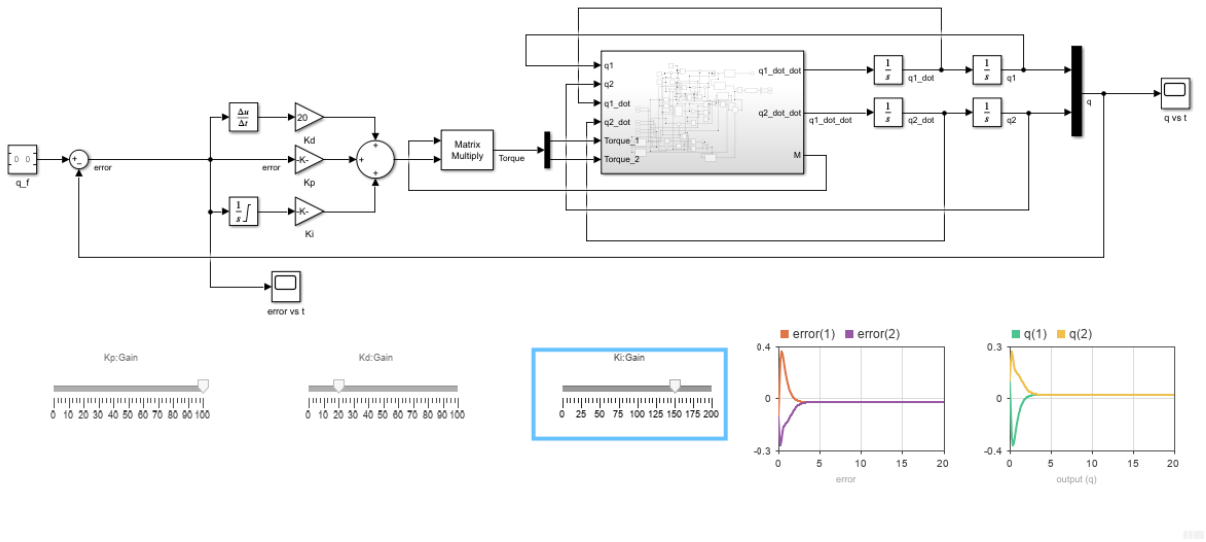

Figure 9: Block Representation of Link Robotic Manipulator in Simulink

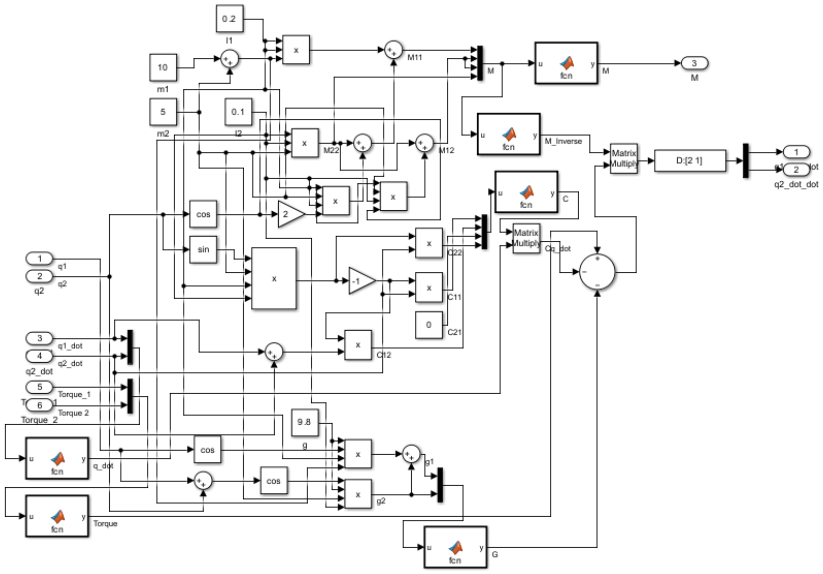

Figure 10: The subsystem to calculate $\ddot{q}$

5.2 Observations

5.2.1 PI Control Plots

(a) $K_p = 150$ $K_i = 0.8$

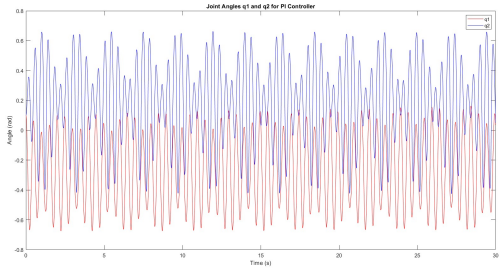

Figure 11: MATLAB Plot

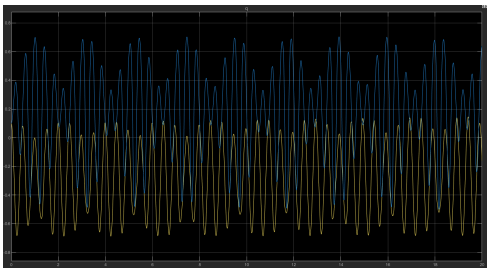

Figure 12: Simulink Plot

We observe exaggerated oscillations due to the lack of a derivative term. The Joint angles of q_1 and q_2 did not achieve a steady state at 0 as desired. The error occurred as there was nothing correcting the overshoot observed in the signal.

Even if there isn't an error at a given moment, the integrator may still be impacted by past errors as the integrator has a memory of all the errors in the past. Because of this, it keeps correcting for a little longer until it overshoots or undershoots. This leads to the oscillations observed in the graph.

However, the steady-state error is less when compared to other systems, like the PD system due to the integral term which adds up the error over time and reduces the steady-state errors.

(b) $K_p = 500$ $K_i = 1$

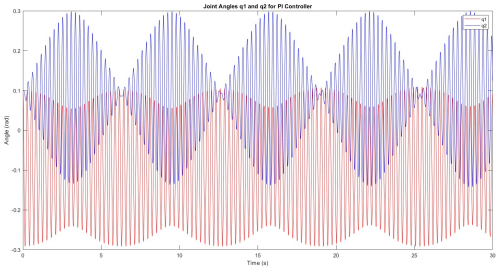

Figure 13: MATLAB Plot

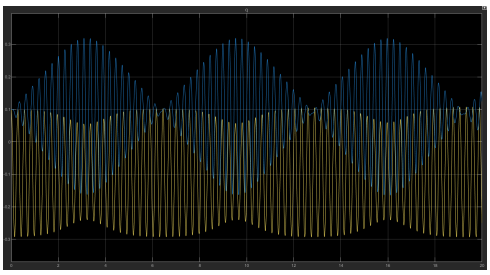

Figure 14: Simulink Plot

In the above plot, we see that the oscillating error increases due to increase in the value of K_p.

(c) $K_p = 1000$ $K_i = 0.5$

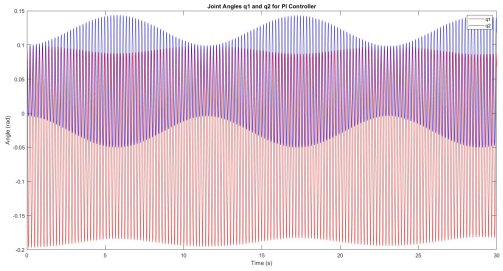

Figure 15: MATLAB Plot

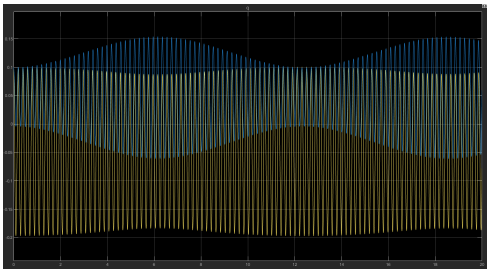

Figure 16: Simulink Plot

We see a similar variance in the above plot since both K_p and K_i are changed. The effect of change in K_i is observed in the faster rate of oscillations of the error in the final plot.

5.2.2 PD Control Plots

(a) $K_p = 100$ $K_d = 20$

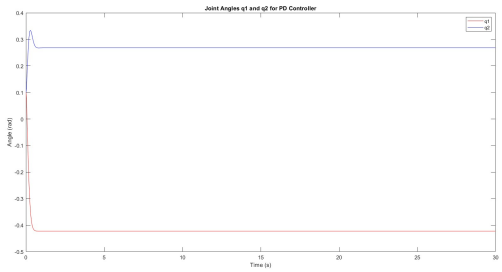

Figure 17: MATLAB Plot

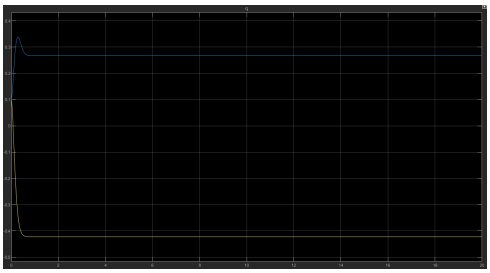

Figure 18: Simulink Plot

We don't observe any exaggerated oscillation as observed in the PI case. The PD controller helped in correcting the overshoot and decreasing the sudden changes in the system and improves the response time. K_d acts as a damping factor.

The joint angles of q_1 and q_2 did not achieve a steady state at 0 as desired. The error occurred as there was nothing to correct the steady-state errors. We observe a steady state error which is slightly away from the desired output and is usually corrected by integral terms.

(b) $K_p = 250$ $K_d = 100$

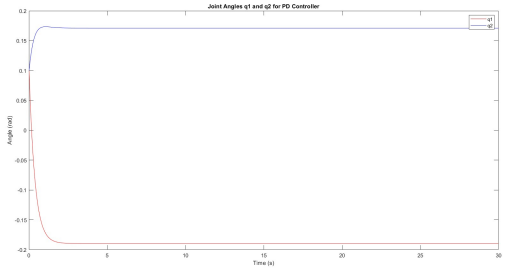

Figure 19: MATLAB Plot

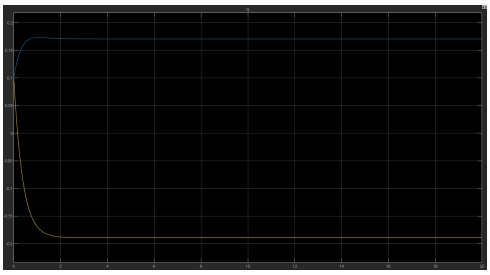

Figure 20: Simulink Plot

Increasing Kp results in a higher error from the desired output of the controller. Increasing Kd, we observe that the overshoot of the signal decreases which can observed in the above plot as compared to the first plot.

(c) $K_p = 250$ $K_d = 250$

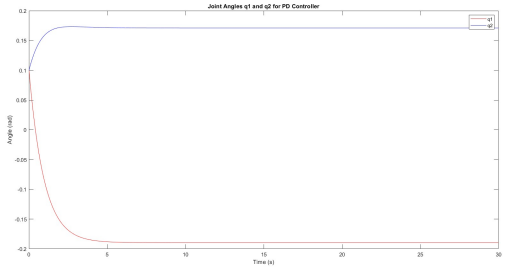

Figure 21: MATLAB Plot

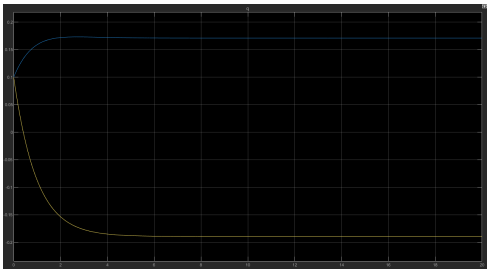

Figure 22: Simulink Plot

The same trend can be observed for the above plot as well.

5.2.3 PID Control Plots

(a) $K_p = 40$ $K_i = 40$ $K_d = 40$

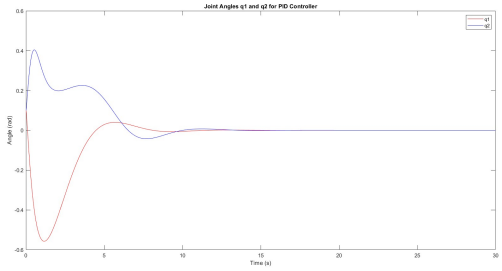

Figure 23: MATLAB Plot

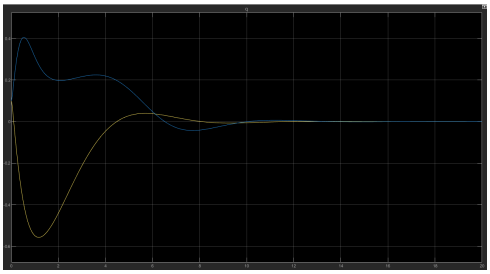

Figure 24: Simulink Plot

All the properties of the PID system make the system achieve the desired result. The System reaches the state (0,0). The PID has the least amount of error and settling when compared to PI and PD controllers.

We observe that the joint angle in q_1 falls sharply. The PD part of the controller Decreases overshoot and settling time. while no steady-state error is observed due to the PI part of the controller.

For q_2 We observe that the joint angle rises suddenly then falls sharply and starts to reach a steady state. The PD part of the controller decreases the overshoot. The rise time also decreases. The PI part of the controller decreases the steady state error in the system.

The proportional terms must be optimum. A higher K_p will give a snappier response, but the chances of overshooting increase and a lot of unwanted oscillations are generated. While a lower K_p will give a sluggish response. Which may lead it to never reach the steady state. The proportional term is selected carefully to avoid this error.

The derivative term helps in limiting the response of the system. This helps when the error is changing rapidly It helps in reducing the overshoot. Too much of this will make the system sluggish again. The correct Kd helped us reduce the discrepancies and

oscillations seen in a PI controller.

The integral term adds up the error over time and reduces the steady-state errors. A higher K_i is also not desirable as the possibility of overshooting increases maKing it unstable. Hence after the accurate value of K_i is taken the result is more accurate when compared to a PD controller.

Hence the K_p, K_i, k_d, are necessary to achieve the desired result

(b) $K_p = 90$ $K_i = 90$ $K_d = 90$

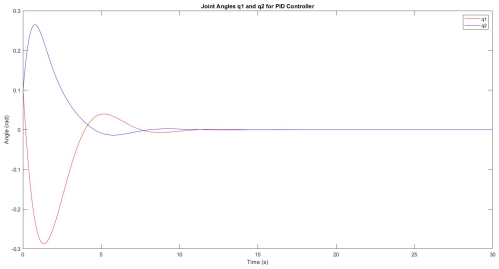

Figure 25: MATLAB Plot

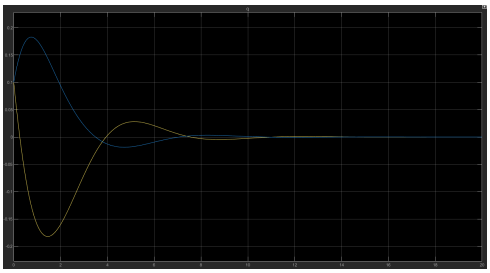

Figure 26: Simulink Plot

Increasing K_i decreases the minute error in the final output; increasing K_d decreases the overshoot of the overall plot and the rise in K_p tries to increase the initial error which is balanced out by the increased value of K_d.

(c) $K_p = 100$ $K_i = 150$ $K_d = 20$

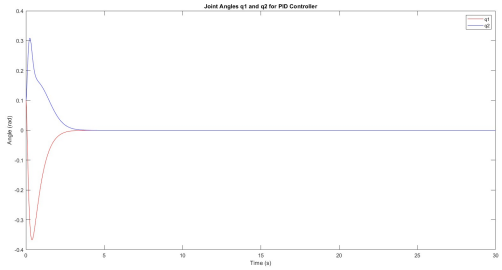

Figure 27: MATLAB Plot

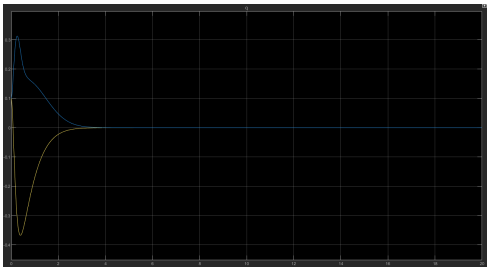

Figure 28: Simulink Plot

Here, we can see that q_1 and q_2 reaches 0 faster and with very less oscillations. While this might not be most optimal, we can manipulate k_p, k_d and k_i to get most optimal output.

6 Conclusion

Two link manipulators have many applications and constitute a basic building block in robotics. In this project, we have focused on understanding the dynamics and control of the system using mathematical and attempted to extrapolate our understanding by implementing the system in MATLAB and SIMULINK.

As shown in the project, control can be triggered with different variations of the proportional, derivative and integral responses. By observing the results, we were able to conclude that the use of the PID controller was the very effective and efficient.

However the experiments were carried out in a synthetic environment, and the output might differ in the real world. Our observations might be ideal but can be used to infer about PID control and be implemented in real world.