ENPM 667 - Control of Robotic Systems

Final Project

Kunj Golwala
UID - 121118271

Shreya Kalyanaraman
UID - 121166647

Instructor : Waseem Ansar Malik
Semester: Fall 2024

Table of Contents

1 INTRODUCTION

This project focuses on designing and analyzing a control system for a crane that moves along a one-dimensional track. The system comprises a frictionless cart with mass M actuated by an external force F, and two suspended loads of masses m_1 and m_2, connected via cables of lengths l_1 and l_2, respectively. The primary goal of the project is to model, linearize, and control the system effectively, ensuring stability and robust performance.

The project is divided into two main components:

- The first component involves deriving the equations of motion for the crane, linearizing the system around its equilibrium point, and determining conditions for controllability. A Linear Quadratic Regulator (LQR) is designed for the linearized system, and simulations are conducted to validate the controller's performance and stability using Lyapunov's indirect method.
- The second component focuses on observability. Various output vectors are evaluated to determine their observability properties. Luenberger observers are designed for the observable cases, and their responses are simulated. Finally, an output feedback controller using the Linear Quadratic Gaussian (LQG) method is implemented and analyzed for its ability to track constant references and reject disturbances.

This report includes detailed calculations, state-space representations, simulation plots, and control designs, offering a comprehensive solution to the posed problem. All simulations are conducted for both the linearized and nonlinear systems, highlighting the practical applicability and limitations of the proposed control strategies.

2 FIRST COMPONENT (100 POINTS)

Consider a crane that moves along a one-dimensional track. It behaves as a frictionless cart with mass M, actuated by an external force F that constitutes the input of the system. There are two loads suspended from cables attached to the crane. The loads have mass m_1 and m_2, and the lengths of the cables are l_1 and l_2, respectively. The following figure depicts the crane and associated variables used throughout this project.

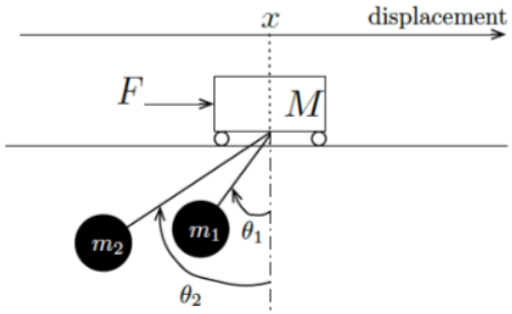

Figure 1. Enter Caption

2.1 Obtain the equations of motion for the system and the corresponding nonlinear state-space representation.

To calculate the kinetic and potential energy of the system, we first need to determine the position vectors of the crane and the suspended loads in terms of the generalized coordinates. For the crane, its position along the track is given directly by the coordinate x. The positions of the two suspended loads are determined by their respective

cable lengths and angles with the vertical. For the first load, the position vector is (x_1, y_1), where $x_1 = x - l_1 \sin(\theta_1)$ and $y_1 = l_1(-\cos(\theta_1))$. Similarly, for the second load, the position vector is (x_2, y_2), where $x_2 = x - l_2 \sin(\theta_2)$ and $y_2 = l_2(-\cos(\theta_2))$. These expressions for x_i and y_i $(i = 1, 2)$ are used to compute the velocities $\dot{x}_i$ and $\dot{y}_i$, which contribute to the kinetic energy, and the heights y_i, which determine the potential energy due to gravity.

To determine the velocities associated with each load and the crane, the equations of motion need to be differentiated with respect to time. The differentiated equations are given as follows:

$$x_1' = \dot{x} - l_1 C_1 \dot{\theta}_1, \quad y_1' = l_1 S_1 \dot{\theta}_1, \tag{1}$$

$$x_2' = \dot{x} - l_2 C_2 \dot{\theta}_2, \quad y_2' = l_2 S_2 \dot{\theta}_2, \tag{2}$$

$$x_M' = \dot{x}, \quad y_M' = 0. \tag{3}$$

The velocities v_1 and v_2 of the loads can be computed as the square root of the sum of the squares of their respective x- and y-velocities, using the Pythagorean theorem:

$$v_1^2 = {x_1'}^2 + {y_1'}^2, \tag{4}$$

$$= \left(\dot{x} - l_1 C_1 \dot{\theta}_1\right)^2 + \left(l_1 S_1 \dot{\theta}_1\right)^2, \tag{5}$$

$$= \dot{x}^2 + l_1^2 \dot{\theta}_1^2 - 2 l_1 C_1 \dot{x} \dot{\theta}_1. \tag{6}$$

$$v_2^2 = {x_2'}^2 + {y_2'}^2, \tag{7}$$

$$= \left(\dot{x} - l_2 C_2 \dot{\theta}_2\right)^2 + \left(l_2 S_2 \dot{\theta}_2\right)^2, \tag{8}$$

$$= \dot{x}^2 + l_2^2 \dot{\theta}_2^2 - 2 l_2 C_2 \dot{x} \dot{\theta}_2. \tag{9}$$

For the crane, the velocity v_M is computed as follows:

$$v_M^2 = {x_M'}^2 + {y_M'}^2, \tag{10}$$

$$= \dot{x}^2. \tag{11}$$

Using these equations, which provide the velocities of all masses in the system, and noting that there is no rotation about the center of mass (COM) for any of the masses, the total kinetic energy of the system is given by:

$$K.E. = \frac{1}{2} M v_M^2 + \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2. \tag{12}$$

The Euler-Lagrange equations are used to get the non-linear dynamics. The kinetic energy is given by: The kinetic energy is given by:

$$T = \frac{1}{2} M \dot{x}^2 + \frac{1}{2} m_1 (\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2} m_2 (\dot{x}_2^2 + \dot{y}_2^2). \tag{13}$$

where

$$x_i = x - l_i \sin(\theta_i), \quad y_i = l_i(-\cos(\theta_i)), \quad i = 1,2, \tag{14}$$

$$\dot{x}_i = \dot{x} - l_i \dot{\theta}_i \cos(\theta_i), \quad \dot{y}_i = l_i \dot{\theta}_i \sin(\theta_i). \tag{15}$$

Rewriting the kinetic energy:

$$\begin{aligned} T = {} & \frac{1}{2}(M + m_1 + m_2)\dot{x}^2 + \frac{1}{2} m_1 l_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 l_2^2 \dot{\theta}_2^2 \\ & - m_1 l_1 \cos(\theta_1)\dot{\theta}_1 \dot{x} - m_2 l_2 \cos(\theta_2)\dot{\theta}_2 \dot{x}. \end{aligned} \tag{16}$$

The potential energy is:

$$V = m_1 g l_1 (1 - \cos(\theta_1)) + m_2 g l_2 (1 - \cos(\theta_2)). \tag{17}$$

The Lagrangian is $L = T - V$. The system dynamics are governed by the equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = F, \tag{18}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) - \frac{\partial L}{\partial \theta_1} = 0, \tag{19}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) - \frac{\partial L}{\partial \theta_2} = 0. \tag{20}$$

From the first equation:

$$\frac{\partial L}{\partial x} = 0, \tag{21}$$

$$\frac{\partial L}{\partial \dot{x}} = (M + m_1 + m_2)\dot{x} - m_1 l_1 \cos(\theta_1)\dot{\theta}_1 - m_2 l_2 \cos(\theta_2)\dot{\theta}_2, \tag{22}$$

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) = {} & (M + m_1 + m_2)\ddot{x} + m_1 l_1 \sin(\theta_1)\dot{\theta}_1^2 + m_2 l_2 \sin(\theta_2)\dot{\theta}_2^2 \\ & - m_1 l_1 \cos(\theta_1)\ddot{\theta}_1 - m_2 l_2 \cos(\theta_2)\ddot{\theta}_2. \end{aligned} \tag{23}$$

The nonlinear system dynamics are:

$$\begin{aligned} (M + m_1 + m_2)\ddot{x} + m_1 l_1 \sin(\theta_1)\dot{\theta}_1^2 + m_2 l_2 \sin(\theta_2)\dot{\theta}_2^2 \\ - m_1 l_1 \cos(\theta_1)\ddot{\theta}_1 - m_2 l_2 \cos(\theta_2)\ddot{\theta}_2 = F, \end{aligned} \tag{24}$$

$$l_1 \ddot{\theta}_1 - \cos(\theta_1)\ddot{x} + g \sin(\theta_1) = 0, \tag{25}$$

$$l_2 \ddot{\theta}_2 - \cos(\theta_2)\ddot{x} + g \sin(\theta_2) = 0. \tag{26}$$

Rearranging:

$$\begin{aligned}\ddot{x} = &\frac{1}{M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2)} \\ &\times \left[F - m_1 l_1 \sin(\theta_1)\dot{\theta}_1^2 - m_2 l_2 \sin(\theta_2)\dot{\theta}_2^2 \right. \\ &\left. - m_1 g \cos(\theta_1)\sin(\theta_1) - m_2 g \cos(\theta_2)\sin(\theta_2)\right], \end{aligned} \tag{27}$$

$$\ddot{\theta}_1 = \frac{1}{l_1}\left[\cos(\theta_1)\ddot{x} - g\sin(\theta_1)\right], \tag{28}$$

$$\ddot{\theta}_2 = \frac{1}{l_2}\left[\cos(\theta_2)\ddot{x} - g\sin(\theta_2)\right]. \tag{29}$$

2.2 Linearization of Non-Linear System

Linearization is a technique used to approximate a nonlinear system with a linear one around a specific operating point, known as the *equilibrium point*. This is particularly useful because linear systems are easier to analyze, design, and control using well-established tools.

For a general nonlinear system:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{30}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input vector, and f represents the nonlinear dynamics, the system can be linearized around the equilibrium point $(\mathbf{x}_e, \mathbf{u}_e)$.

At equilibrium:

$$\begin{aligned} &f(\mathbf{x}_e, \mathbf{u}_e) = 0, \\ &\text{and small perturbations are:} \quad \delta\mathbf{x} = \mathbf{x} - \mathbf{x}_e, \quad \delta\mathbf{u} = \mathbf{u} - \mathbf{u}_e. \end{aligned} \tag{31}$$

Jacobian Linearization

The linearized system near the equilibrium point can be approximated as:

$$\delta\dot{\mathbf{x}} = A\delta\mathbf{x} + B\delta\mathbf{u}, \tag{32}$$

where:

$$A = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{(\mathbf{x}_e, \mathbf{u}_e)}, \quad B = \left.\frac{\partial f}{\partial \mathbf{u}}\right|_{(\mathbf{x}_e, \mathbf{u}_e)}. \tag{33}$$

The matrices A and B are the Jacobian matrices, which are obtained by computing the partial derivatives of f with respect to the state vector $\mathbf{x}$ and input vector $\mathbf{u}$, respectively. These derivatives are evaluated at the equilibrium point.

Steps for Jacobian Linearization

1. Identify the nonlinear system dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$. 2. Determine the equilibrium point $(\mathbf{x}_e, \mathbf{u}_e)$, where all derivatives are zero. 3. Compute the Jacobian matrices:

$$A = \frac{\partial f}{\partial \mathbf{x}}, \quad B = \frac{\partial f}{\partial \mathbf{u}}. \tag{34}$$

4. Substitute the equilibrium point into the Jacobians to evaluate A and B. 5. Write the linearized system as:

$$\delta\dot{\mathbf{x}} = A\delta\mathbf{x} + B\delta\mathbf{u}. \tag{35}$$

The resulting linearized model is valid in a small neighborhood around the equilibrium point.

Advantages of Linearization

- Simplifies the analysis and design of control systems.
- Enables the use of linear control techniques (e.g., pole placement, LQR, etc.).
- Provides a good approximation for small perturbations around equilibrium.

2.2.1 Equilibrium Conditions

The equilibrium conditions are:

$$x = 0, \quad \dot{x} = 0, \quad \theta_1 = 0, \quad \dot{\theta}_1 = 0, \quad \theta_2 = 0, \quad \dot{\theta}_2 = 0. \tag{36}$$

At equilibrium, we can make the following assumptions:

$$\theta_1 = 0, \quad \theta_2 = 0, \quad \sin\theta_1 \approx \theta_1, \quad \sin\theta_2 \approx \theta_2, \quad \cos\theta_1 \approx 1, \quad \cos\theta_2 \approx 1, \quad \dot{\theta}_1^2 \approx 0, \quad \dot{\theta}_2^2 \approx 0, \quad F = 0. \tag{37}$$

2.2.2 Jacobian Linearization

The linearized system is obtained using the Jacobian matrices:

$$A = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{(\mathbf{x}_e, \mathbf{u}_e)}, \quad B = \left.\frac{\partial f}{\partial \mathbf{u}}\right|_{(\mathbf{x}_e, \mathbf{u}_e)}. \tag{38}$$

The state vector $\mathbf{x}$ and input $\mathbf{u}$ are defined as:

$$\mathbf{x} = \begin{bmatrix} x & \dot{x} & \theta_1 & \dot{\theta}_1 & \theta_2 & \dot{\theta}_2 \end{bmatrix}^T, \quad \mathbf{u} = F. \tag{39}$$

Partial Derivatives for the System Dynamics

1. Partial derivatives for $\dot{x}$:

$$\frac{\partial \dot{x}}{\partial x} = 0, \quad \frac{\partial \dot{x}}{\partial \dot{x}} = 1, \quad \frac{\partial \dot{x}}{\partial \theta_1} = 0, \quad \frac{\partial \dot{x}}{\partial \dot{\theta}_1} = 0, \quad \frac{\partial \dot{x}}{\partial \theta_2} = 0, \quad \frac{\partial \dot{x}}{\partial \dot{\theta}_2} = 0. \tag{40}$$

2. Partial derivatives for $\ddot{x}$:

$$\frac{\partial \ddot{x}}{\partial x} = 0, \quad \frac{\partial \ddot{x}}{\partial \dot{x}} = 0, \quad \frac{\partial \ddot{x}}{\partial \theta_1} = -\frac{gm_1}{M}, \quad \frac{\partial \ddot{x}}{\partial \dot{\theta}_1} = 0, \quad \frac{\partial \ddot{x}}{\partial \theta_2} = -\frac{gm_2}{M}, \quad \frac{\partial \ddot{x}}{\partial \dot{\theta}_2} = 0. \tag{41}$$

3. Partial derivatives for $\dot{\theta}_1$:

$$\frac{\partial \dot{\theta}_1}{\partial x} = 0, \quad \frac{\partial \dot{\theta}_1}{\partial \dot{x}} = 0, \quad \frac{\partial \dot{\theta}_1}{\partial \theta_1} = 0, \quad \frac{\partial \dot{\theta}_1}{\partial \dot{\theta}_1} = 1, \quad \frac{\partial \dot{\theta}_1}{\partial \theta_2} = 0, \quad \frac{\partial \dot{\theta}_1}{\partial \dot{\theta}_2} = 0. \tag{42}$$

4. Partial derivatives for $\ddot{\theta}_1$:

$$\frac{\partial \ddot{\theta}_1}{\partial x} = 0, \quad \frac{\partial \ddot{\theta}_1}{\partial \dot{x}} = 0, \quad \frac{\partial \ddot{\theta}_1}{\partial \theta_1} = -\frac{gM + gm_1}{Ml_2}, \quad \frac{\partial \ddot{\theta}_1}{\partial \dot{\theta}_1} = 0, \quad \frac{\partial \ddot{x}}{\partial \theta_2} = -\frac{gm_2}{Ml_1}, \quad \frac{\partial \ddot{x}}{\partial \dot{\theta}_2} = 0. \tag{43}$$

5. Partial derivatives for $\dot{\theta}_2$:

$$\frac{\partial \dot{\theta}_2}{\partial x} = 0, \quad \frac{\partial \dot{\theta}_2}{\partial \dot{x}} = 0, \quad \frac{\partial \dot{\theta}_2}{\partial \theta_1} = 0, \quad \frac{\partial \dot{\theta}_2}{\partial \dot{\theta}_1} = 0, \quad \frac{\partial \dot{\theta}_2}{\partial \theta_2} = 0, \quad \frac{\partial \dot{\theta}_2}{\partial \dot{\theta}_2} = 1. \tag{44}$$

6. Partial derivatives for $\ddot{\theta}_2$:

$$\frac{\partial \ddot{\theta}_2}{\partial x} = 0, \quad \frac{\partial \ddot{\theta}_2}{\partial \dot{x}} = 0, \quad \frac{\partial \ddot{\theta}_1}{\partial \theta_1} = -\frac{gm_1}{Ml_2}, \quad \frac{\partial \ddot{\theta}_1}{\partial \dot{\theta}_1} = 0, \quad \frac{\partial \ddot{\theta}_2}{\partial \theta_2} = -\frac{gM + gm_2}{Ml_2}, \quad \frac{\partial \ddot{\theta}_2}{\partial \dot{\theta}_2} = 0. \tag{45}$$

The Jacobian matrix linearizes the system dynamics around the equilibrium point. Below, we show where each partial derivative contributes to the linearized A matrix.

The nonlinear equations are:

$$\mathbf{f}(\mathbf{x}, u) = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{bmatrix}. \tag{46}$$

We compute:

$$A = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{(\mathbf{x}_e, u_e)}, \quad B = \left.\frac{\partial \mathbf{f}}{\partial u}\right|_{(\mathbf{x}_e, u_e)}. \tag{47}$$

2.2.3 Linear State Space Representation

The state vector is:

$$\mathbf{x} = \begin{bmatrix} x & \dot{x} & \theta_1 & \dot{\theta}_1 & \theta_2 & \dot{\theta}_2 \end{bmatrix}^T. \tag{48}$$

The matrix A is:

$$A = \begin{bmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial \dot{x}} & \frac{\partial \dot{x}}{\partial \theta_1} & \frac{\partial \dot{x}}{\partial \dot{\theta}_1} & \frac{\partial \dot{x}}{\partial \theta_2} & \frac{\partial \dot{x}}{\partial \dot{\theta}_2} \\ \frac{\partial \ddot{x}}{\partial x} & \frac{\partial \ddot{x}}{\partial \dot{x}} & \frac{\partial \ddot{x}}{\partial \theta_1} & \frac{\partial \ddot{x}}{\partial \dot{\theta}_1} & \frac{\partial \ddot{x}}{\partial \theta_2} & \frac{\partial \ddot{x}}{\partial \dot{\theta}_2} \\ \frac{\partial \dot{\theta}_1}{\partial x} & \frac{\partial \dot{\theta}_1}{\partial \dot{x}} & \frac{\partial \dot{\theta}_1}{\partial \theta_1} & \frac{\partial \dot{\theta}_1}{\partial \dot{\theta}_1} & \frac{\partial \dot{\theta}_1}{\partial \theta_2} & \frac{\partial \dot{\theta}_1}{\partial \dot{\theta}_2} \\ \frac{\partial \ddot{\theta}_1}{\partial x} & \frac{\partial \ddot{\theta}_1}{\partial \dot{x}} & \frac{\partial \ddot{\theta}_1}{\partial \theta_1} & \frac{\partial \ddot{\theta}_1}{\partial \dot{\theta}_1} & \frac{\partial \ddot{\theta}_1}{\partial \theta_2} & \frac{\partial \ddot{\theta}_1}{\partial \dot{\theta}_2} \\ \frac{\partial \dot{\theta}_2}{\partial x} & \frac{\partial \dot{\theta}_2}{\partial \dot{x}} & \frac{\partial \dot{\theta}_2}{\partial \theta_1} & \frac{\partial \dot{\theta}_2}{\partial \dot{\theta}_1} & \frac{\partial \dot{\theta}_2}{\partial \theta_2} & \frac{\partial \dot{\theta}_2}{\partial \dot{\theta}_2} \\ \frac{\partial \ddot{\theta}_2}{\partial x} & \frac{\partial \ddot{\theta}_2}{\partial \dot{x}} & \frac{\partial \ddot{\theta}_2}{\partial \theta_1} & \frac{\partial \ddot{\theta}_2}{\partial \dot{\theta}_1} & \frac{\partial \ddot{\theta}_2}{\partial \theta_2} & \frac{\partial \ddot{\theta}_2}{\partial \dot{\theta}_2} \end{bmatrix}. \tag{49}$$

Substituting the partial derivatives:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{gm_1}{M} & 0 & -\frac{gm_2}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{gM+gm_1}{Ml_1} & 0 & -\frac{gm_2}{Ml_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{gm_1}{Ml_2} & 0 & -\frac{gM+gm_2}{Ml_2} & 0 \end{bmatrix}. \tag{50}$$

The input $u = F$ directly affects $\ddot{x}$, $\ddot{\theta}_1$, and $\ddot{\theta}_2$. The B matrix is:

$$B = \begin{bmatrix} \frac{\partial \dot{x}}{\partial u} \\ \frac{\partial \ddot{x}}{\partial u} \\ \frac{\partial \dot{\theta}_1}{\partial u} \\ \frac{\partial \ddot{\theta}_1}{\partial u} \\ \frac{\partial \dot{\theta}_2}{\partial u} \\ \frac{\partial \ddot{\theta}_2}{\partial u} \end{bmatrix}. \tag{51}$$

Substituting values:

$$B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml_1} \\ 0 \\ \frac{1}{Ml_2} \end{bmatrix}. \tag{52}$$

The linearized state-space representation is:

$$\dot{\mathbf{X}} = A\mathbf{X}(t) + BU(t), \quad \mathbf{Y} = C\mathbf{X}(t) + DU(t), \tag{53}$$

where:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{gm_1}{M} & 0 & -\frac{gm_2}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{g}{l_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\frac{g}{l_2} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml_1} \\ 0 \\ \frac{1}{Ml_2} \end{bmatrix}. \tag{54}$$

If the output $\mathbf{Y}$ consists of the states $x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2$, then:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{55}$$

2.3 Obtain conditions on M, m1, m2, l1, l2 for which the linearized system is controllable

The linear time-varying system is controllable on the interval $(0, T)$ if and only if the controllability Gramian matrix is invertible. The Gramian matrix of controllability, denoted $C(A, B)$, is considered invertible when the controllability matrix satisfies the following conditions:

- The rank of $C(A, B)$ must be full rank.
- The full rank is defined as the order of the matrix (denoted by n).

The controllability matrix is given by:

$$R = \begin{bmatrix} B & AB & A^2B & A^3B & A^4B & A^5B \end{bmatrix} \tag{56}$$

CODE IMPLEMENTATION

```
import sympy as sp

# Define the symbolic variables for system parameters
M, m1, m2, l1, l2, g = sp.symbols('M m1 m2 l1 l2 g')

# Define the A matrix for the linearized state-space representation
A = sp.Matrix([
    [0, 1, 0, 0, 0, 0],
    [0, 0, -(m1 * g) / M, 0, -(m2 * g) / M, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, -((M + m1) * g) / (M * l1), 0, -(m2 * g) / (M * l1), 0],
    [0, 0, 0, 0, 0, 1],
    [0, 0, -(m1 * g) / (M * l2), 0, -(g * (M + m2)) / (M * l2), 0]
])

# Display the A matrix
print("A matrix (system dynamics): ", A)

# Define the B matrix (input matrix) for the system
B = sp.Matrix([0, 1/M, 0, 1/(M*l1), 0, 1/(M*l2)]).reshape(6, 1)

# Display the B matrix
print("B matrix (input influence): ", B)

# Construct the controllability matrix by stacking powers of A with B
Ct = sp.Matrix.hstack(B, A*B, A**2*B, A**3*B, A**4*B, A**5*B)
```

```
27
28  # Display the controllability matrix
29  print("Controllability matrix (Ct): ", Ct)
30
31  # Simplify and calculate the determinant of the controllability matrix
32  det_Ct = sp.simplify(sp.det(Ct))
33
34  # Display the determinant of the controllability matrix
35  print("Determinant of the controllability matrix:", det_Ct)
36
37  # Calculate and display the rank of the controllability matrix
38  rank_controllability = Ct.rank()
39  print("Rank of the controllability matrix:", rank_controllability)
40
41  # Handle special case where pendulum lengths are equal (l1 = l2)
42  Ct1 = Ct.subs(l1, l2)
43
44  # Display the controllability matrix for the special case where l1 = l2
45  print("For l1 = l2, the controllability matrix is: ", Ct1)
46
47  # Calculate and display the rank of the new matrix (for l1 = l2)
48  print("Rank of the new controllability matrix:")
49  rank_Ct1 = Ct1.rank()
50  print(rank_Ct1)
51
52  # Check if the system remains controllable with l1 = l2
53  if rank_Ct1 == rank_controllability:
54      print("System is controllable because the ranks are equal.")
55  else:
56      print("System is not controllable because the ranks are not equal.")
```

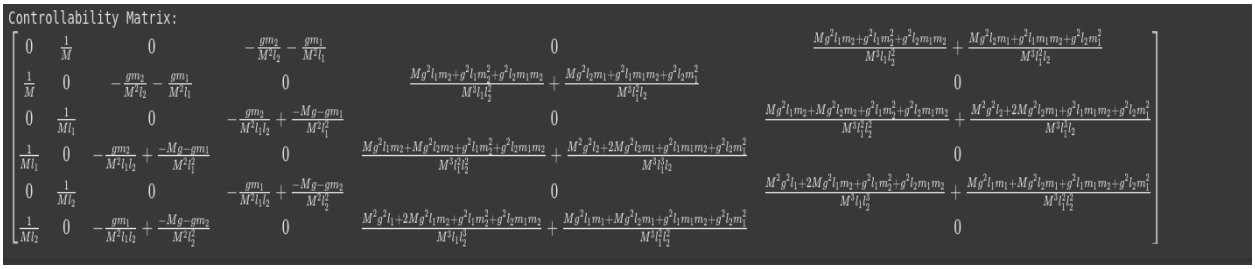

Figure 2. Controllability Matrix

$$\text{Determinant of the controllability matrix:} \quad \frac{g^6\left(-l_1^2+2l_1l_2-l_2^2\right)}{M^6l_1^6l_2^6} \tag{57}$$

Upon computation, it is observed that when $l_1 = l_2$, the determinant becomes zero, and when either $l_1 = 0$ or $l_2 = 0$, the controllability cannot be computed. Therefore, the conditions for achieving controllability are:

- $l_1 \neq l_2$
- $l_1 \neq 0$ and $l_2 \neq 0$

2.4 Choose M = 1000Kg, m1 = m2 = 100Kg, l1 = 20m and l2 = 10m. Check that the system is controllable and obtain an LQR controller. Simulate the resulting response to initial conditions when the controller is applied to the linearized system and the original nonlinear system. Adjust the parameters of the LQR cost until you obtain a suitable response. Use Lyapunov's indirect method to certify the stability (locally or globally) of the closed-loop system.

The Linear Quadratic Regulator (LQR) is an optimal control technique for linear systems. It computes a feedback control law that minimizes a quadratic cost function, balancing state deviation and control effort.

For a linear time-invariant (LTI) system:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}, \tag{58}$$

the goal of LQR is to minimize the cost function:

$$J = \int_0^\infty \left(\mathbf{x}^T Q\mathbf{x} + \mathbf{u}^T R\mathbf{u}\right) dt, \tag{59}$$

where:

- Q: A symmetric, positive semi-definite matrix penalizing state deviations.
- R: A symmetric, positive definite matrix penalizing control effort.

The optimal control law is given by:

$$\mathbf{u} = -K\mathbf{x}, \tag{60}$$

where the gain matrix K is:

$$K = R^{-1}B^T P. \tag{61}$$

Here, P is the solution to the continuous-time Algebraic Riccati Equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0. \tag{62}$$

Key Features

- Provides an optimal trade-off between performance and control effort.
- Ensures stability for properly chosen Q and R.
- Widely used in robotics, aerospace, and control systems.

2.4.1 LQR Controller applied to Linearized system

```
import numpy as np
from scipy import signal
from scipy.linalg import solve_continuous_are
import matplotlib.pyplot as plt
from scipy.signal import lsim

# Define system parameters
M = 1000  # Mass of the cart
```

```
m1 = 100  # Mass of Pendulum 1
m2 = 100  # Mass of Pendulum 2
l1 = 20   # Length of the string of Pendulum 1
l2 = 10   # Length of the string of Pendulum 2
g = 9.81  # Gravity

A = np.array([
    [0, 1, 0, 0, 0, 0],
    [0, 0, -(m1 * g) / M, 0, -(m2 * g) / M, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, -((M + m1) * g) / (M * l1), 0, -(m2 * g) / (M * l1), 0],
    [0, 0, 0, 0, 0, 1],
    [0, 0, -(m1 * g) / (M * l2), 0, -(g * (M + m2)) / (M * l2), 0]
])

B = np.array([[0], [1 / M], [0], [1 / (M * l1)], [0], [1 / (M * l2)]])
C = np.eye(6)
D = np.zeros((6, 1))

# LQR Weights
Q = np.diag([100, 100, 1000, 1000, 1000, 1000])
R = np.array([[0.01]])

# Solve LQR
P = solve_continuous_are(A, B, Q, R)
K = np.dot(np.linalg.inv(R), np.dot(B.T, P))
print(K)
# Closed-loop system
A_cl = A - np.dot(B, K)
# Closed-loop system
A_cl = A - np.dot(B, K)

# Calculate eigenvalues of the closed-loop system
eigenvalues = np.linalg.eigvals(A_cl)
print("Closed-loop eigenvalues:")
for i, eigenvalue in enumerate(eigenvalues, start=1):
    print(f"Eigenvalue {i}: {eigenvalue.real:.4f} + {eigenvalue.imag:.4f}j")



# Simulation
x_initial = np.array([0, 0, 20, 0, 40, 0])  # Initial state
t = np.linspace(0, 500, 1000)
u = np.zeros_like(t)

# Without Controller
sys1 = signal.StateSpace(A, B, C, D)
_, y1, x1 = lsim(sys1, U=u, T=t, X0=x_initial)

# With Controller
sys2 = signal.StateSpace(A_cl, B, C, D)
_, y2, x2 = lsim(sys2, U=u, T=t, X0=x_initial)

# Plotting
```

```
# Plotting each state variable in a separate figure
# Plotting each state variable in a separate figure
# Plotting each state variable in a separate figure
state_vars = ['x', '$\\dot{x}$', '$\\theta_1$', '$\\dot{\\theta}_1$', '$\\theta_2$', '$\\dot{\\
    theta}_2$']

for i, var in enumerate(state_vars):
    # Create a new figure for each state variable
    plt.figure(figsize=(10, 6))

    # Plot response without controller (blue) and with controller (orange)
    plt.plot(t, x1[:, i], color='blue', label='Without Controller')
    plt.plot(t, x2[:, i], color='orange', label='With Controller')

    # Add titles and labels
    plt.title(f'Response of {var}', fontsize=14)
    plt.xlabel('Time (s)', fontsize=12)
    plt.ylabel(f'{var} Response', fontsize=12)
    plt.grid(True)
    plt.legend(fontsize=10)

    # Save the plot for each state variable
    filename = f'response_var_{i + 1}.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight')

    # Download the plot in Colab
    from google.colab import files
    files.download(filename)
    plt.show()
```

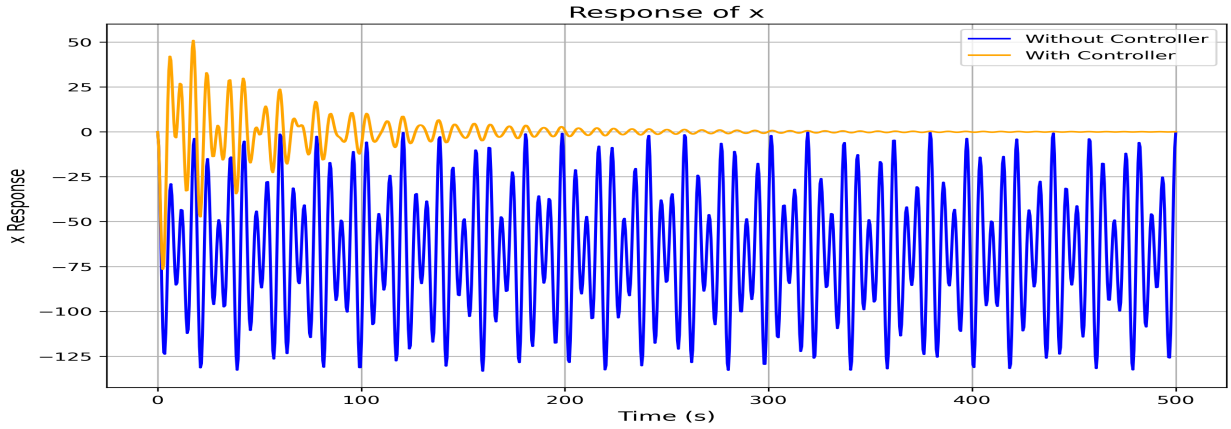

Figure 3. Variation of x with time

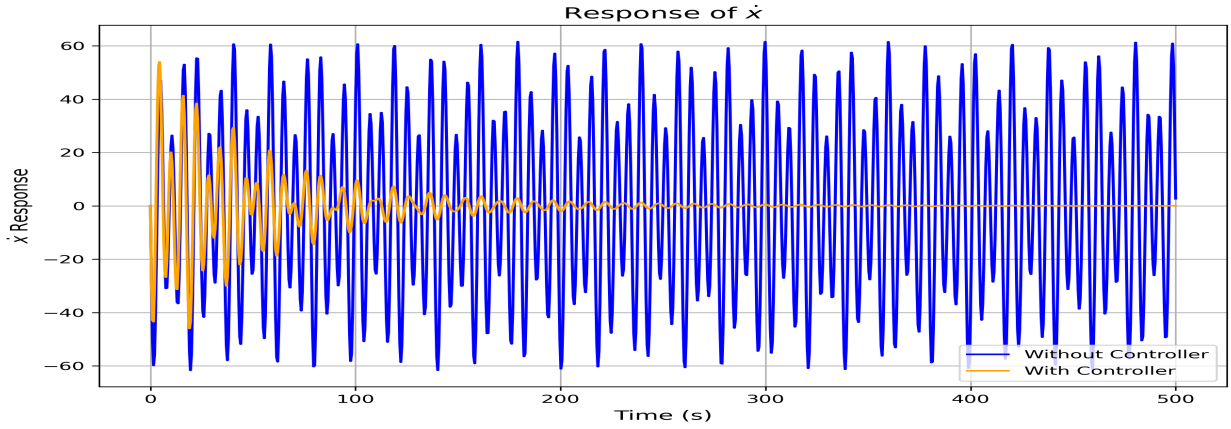

Figure 4. Variation of $\dot{x}$ with time

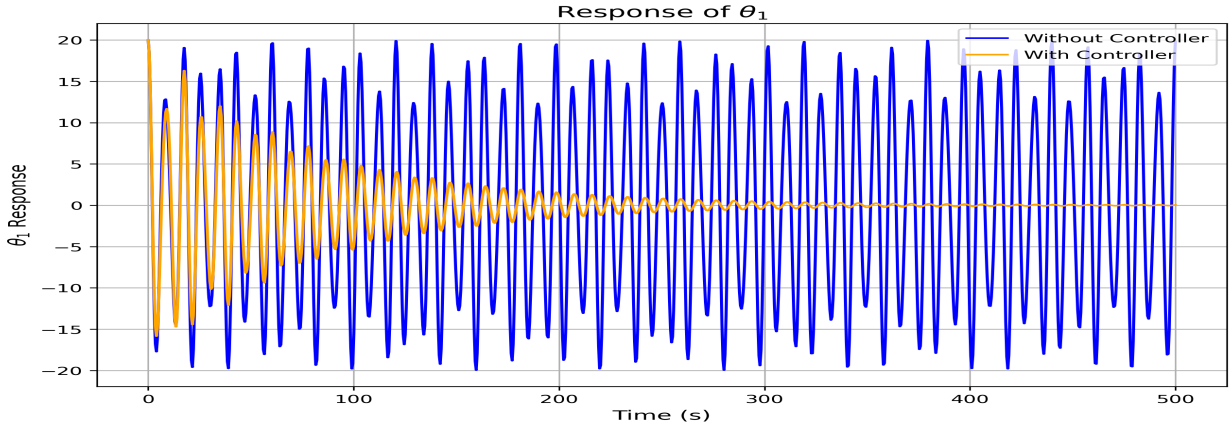

Figure 5. Variation of θ_1 with time

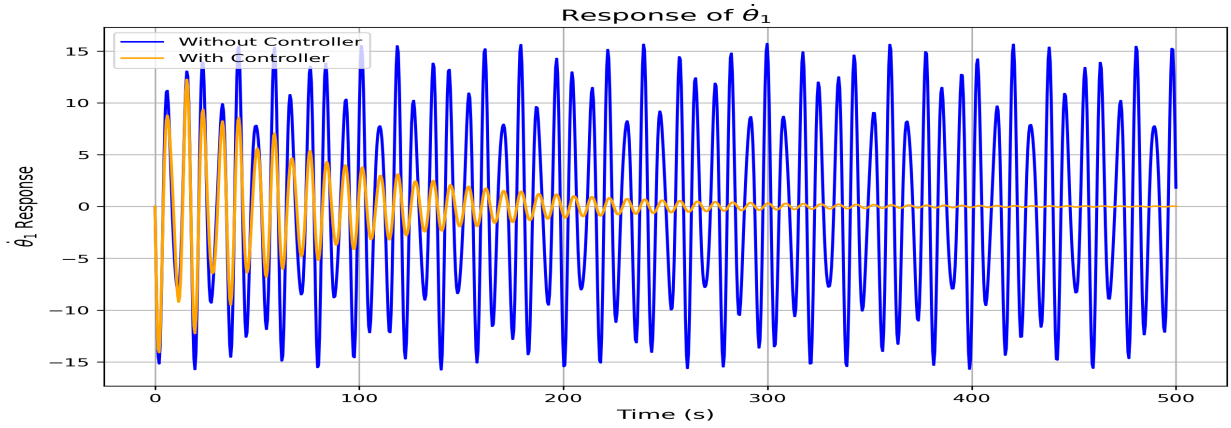

Figure 6. Variation of $\dot{\theta}_1$ with time

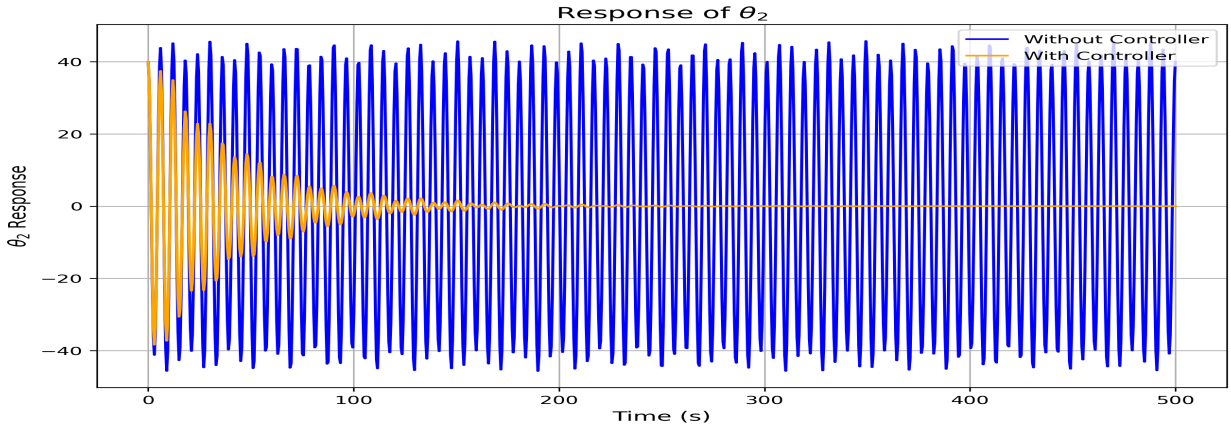

Figure 7. Variation of θ_2 with time

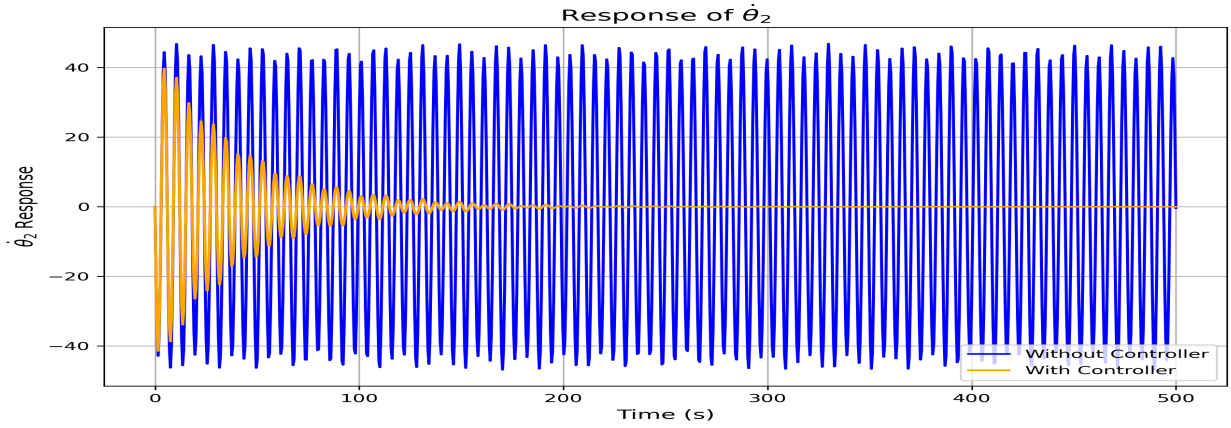

Figure 8. Variation of $\dot{\theta}_2$ with time

2.4.2 LQR Controller applied to Non - Linear system

```
import numpy as np
from scipy.integrate import solve_ivp
from scipy.linalg import solve_continuous_are
import matplotlib.pyplot as plt
# 1. Define system parameters
M = 1000 # Mass of the cart
m1 = 100 # Mass of Pendulum 1
m2 = 100 # Mass of Pendulum 2
l1 = 20 # Length of the string of Pendulum 1
l2 = 10 # Length of the string of Pendulum 2
g = 9.81 # Acceleration due to gravity (m/s^2)
# Define the LQR parameters
Q = np.diag([1000, 100, 1000, 1000, 100, 100])
R = np.array([[0.1]])
A = np.array([[0, 1, 0, 0, 0, 0],
 [0, 0, -(m1*g)/M, 0, -(m2*g)/M, 0],
 [0, 0, 0, 1, 0, 0],
 [0, 0, -((M+m1)*g)/(M*l1), 0, -(m2*g)/(M*l1), 0],
 [0, 0, 0, 0, 0, 1],
 [0, 0, -(m1*g)/(M*l2), 0, -(g*(M+m2))/(M*l2), 0]])
print("A = ")
A

B = np.array([[0], [1/M], [0], [1/(M*l1)], [0], [1/(M*l2)]])
print("B = ")
B
P = solve_continuous_are(A, B, Q, R)
print("P = ")
P
K_val = np.dot(np.linalg.inv(R), np.dot(B.T, P))
print("K = ")
K_val

y0 = np.array([5, 0, np.radians(30), 0, np.radians(60), 0])
print("Initial conditions = ")
y0
def double_pendulum_dynamics(t, y):
    """
    Computes the derivatives of the state vector for a double pendulum on a cart.

    Parameters:
    t : float
        Current time (not used in equations but required for solvers)
    y : ndarray
        State vector [x, x_dot, theta1, theta1_dot, theta2, theta2_dot]

    Returns:
    dydt : ndarray
        Derivatives of the state vector
    """
    # Compute the control force (using an LQR feedback gain)
    F = -np.dot(K_val, y)

```

```
    # Initialize the derivative array
    dydt = np.zeros(6)

    # Dynamics of the system
    dydt[0] = y[1]  # dx/dt = x_dot
    dydt[1] = (F
               - (g / 2) * (m1 * np.sin(2 * y[2]) + m2 * np.sin(2 * y[4]))
               - (m1 * l1 * y[3]**2 * np.sin(y[2]))
               - (m2 * l2 * y[5]**2 * np.sin(y[4]))) / (M + m1 * np.sin(y[2])**2 + m2 * np.sin(y
                   [4])**2)
    dydt[2] = y[3]  # dtheta1/dt = theta1_dot
    dydt[3] = (dydt[1] * np.cos(y[2]) - g * np.sin(y[2])) / l1  # dtheta1_dot/dt
    dydt[4] = y[5]  # dtheta2/dt = theta2_dot
    dydt[5] = (dydt[1] * np.cos(y[4]) - g * np.sin(y[4])) / l2  # dtheta2_dot/dt

    return dydt

# Time span
tspan = (0, 5000)
t_eval = np.arange(0, 5000, 0.01)
# Solve the differential equations
sol = solve_ivp(double_pendulum_dynamics, tspan, y0, t_eval=t_eval)
plt.figure(figsize=(12, 18))
state_labels = ['x', '$\\dot{x}$', '$\\theta_1$', '$\\dot{\\theta}_1$', '$\\theta_2$', '$\\dot{\\
    theta}_2$']
for i in range(sol.y.shape[0]):
 plt.subplot(len(state_labels), 1, i+1)
 plt.plot(sol.t, sol.y[i], label=state_labels[i])
 plt.title(f'State: {state_labels[i]}')
 plt.xlabel('Time')
 plt.ylabel(state_labels[i])
 plt.legend()
 plt.grid(True)
#  plt.tight_layout()
plt.show()
```

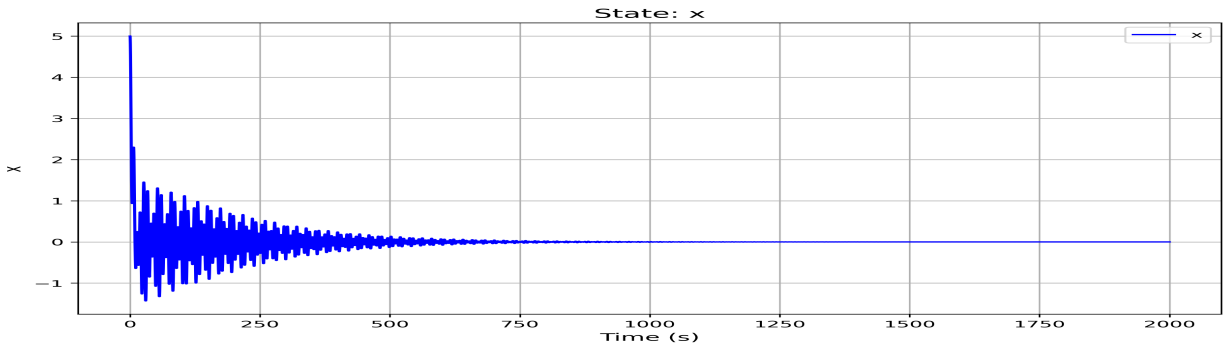

Figure 9. Variation of x with time

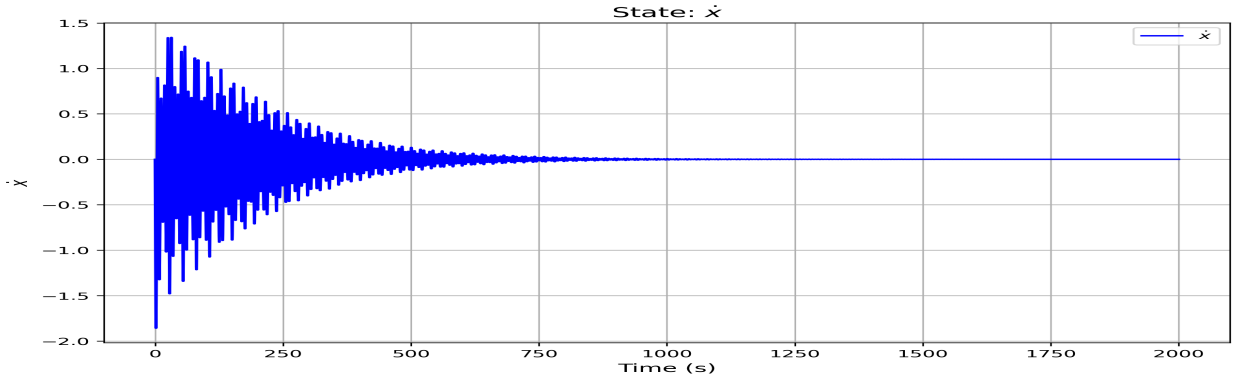

Figure 10. Variation of $\dot{x}$ with time

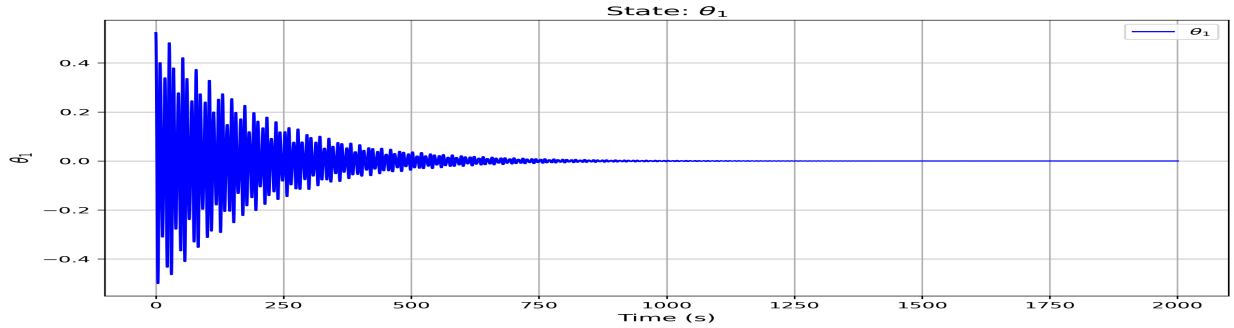

Figure 11. Variation of θ_1 with time

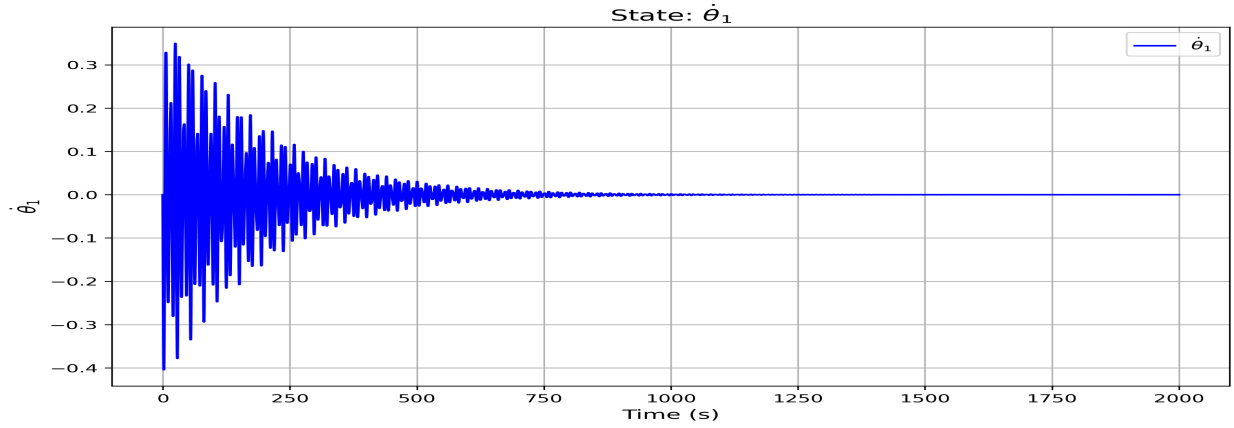

Figure 12. Variation of $\dot{\theta}_1$ with time

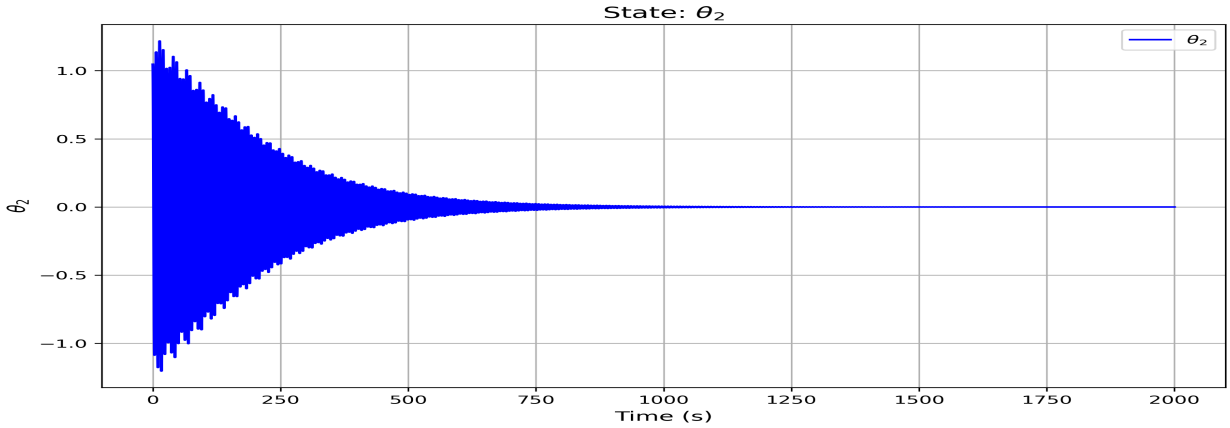

Figure 13. Variation of θ_2 with time

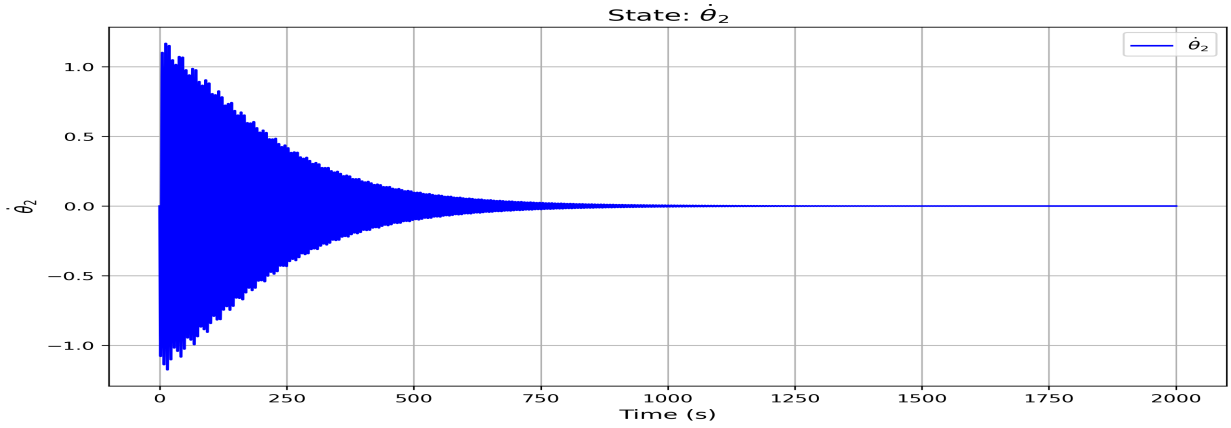

Figure 14. Variation of $\dot{\theta}_2$ with time

2.4.3 Lyapunov's Indirect Method

Lyapunov's indirect method also called the linearization technique, evaluates the stability of an equilibrium point by examining the eigenvalues of the Jacobian matrix. For a continuous-time system, consider the matrix $\mathbf{A} - \mathbf{BK}$:

- The equilibrium is **asymptotically stable** if all eigenvalues have **negative real parts**.
- The equilibrium is **unstable** if any eigenvalue has a **positive real part**.
- If eigenvalues include **zero real parts** alongside others with **negative real parts**, Lyapunov instability test is inconclusive.

This method offers a straightforward way to evaluate stability by focusing on the linearized system's eigenvalue behavior.

The eigenvalues of the controlled system are as follows:

Eigenvalue Index	Value
1	$-0.2066 + 0.2023\,j$
2	$-0.2066 - 0.2023\,j$
3	$-0.0245 + 1.0421\,j$
4	$-0.0245 - 1.0421\,j$
5	$-0.0120 + 0.7276\,j$
6	$-0.0120 - 0.7276\,j$

From the analysis of the eigenvalues, it is evident that all real parts are negative. This demonstrates that the system is asymptotically stable, meaning any small deviation from the equilibrium will decay over time, and the system will return to its steady state. The complex components of the eigenvalues suggest that the system's response involves oscillatory dynamics. Despite these oscillations, the negative real parts indicate sufficient damping, ensuring that the oscillations diminish over time without leading to instability or divergence.

In summary, the Lyapunov indirect method confirms the effectiveness of the designed LQR controller in stabilizing the system. The control strategy achieves robust stability around the equilibrium point, validating its suitability for the application.

3 SECOND COMPONENT (100 POINTS)

Consider the parameters selected in part C) above.

3.1 Suppose that you can select the following output vectors:

$$x(t), \quad (\theta_1(t), \theta_2(t)), \quad (x(t), \theta_2(t)), \quad \textbf{or} \quad (x(t), \theta_1(t), \theta_2(t)).$$

Determine for which output vectors the linearized system is observable.

Observability in a control system refers to the ability to infer the internal states of the system by analyzing its outputs over a finite period of time when subjected to certain inputs. For a Linear Time-Invariant (LTI) discrete system represented in state-space form as:

$$X(k+1) = AX(k), \quad X(0) = X_0 \tag{63}$$

the system's output, denoted by Y, is given by:

$$Y = CX + DU \tag{64}$$

To assess the observability of the system, we use the observability matrix $O(A,C)O(A,C) = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$

```python
import numpy as np
import sympy as sp

# Defining symbols for parameters
M, m1, m2, l1, l2, g = sp.symbols('M m1 m2 l1 l2 g')

M = 1000
m1 = 100
m2 = 100
l1 = 20
l2 = 10
g = 9.81

# Defining the A matrix for the linearized state-space representation
A = sp.Matrix([
    [0, 1, 0, 0, 0, 0],
    [0, 0, -(m1 * g) / M, 0, -(m2 * g) / M, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, ((M + m1) * g) / (M * l1), 0, -(m2 * g) / (M * l1), 0],
    [0, 0, 0, 0, 0, 1],
    [0, 0, -(m1 * g) / (M * l2), 0, -(g * (M + m2)) / (M * l2), 0]
])

# Creating the C matrices for different output vectors
C_matrices = {
    r'x': np.array([[1, 0, 0, 0, 0, 0]]),
    r'theta_1 and theta_2': np.array([[0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]]),
    r'x and theta_2': np.array([[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0]]),
    r'x, theta_1 and theta_2': np.array([[1, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1,
        0]])
}

# Function to compute the observability matrix Ct
def compute_Ct(A, C):
    C_matrix = sp.Matrix(C)  # Convert numpy array to sympy Matrix for compatibility
    O_matrix = C_matrix  # Start with the first C matrix
    for i in range(1, 6):  # Compute A^i * C and horizontally stack them
        O_matrix = O_matrix.col_join(C_matrix * A**i)
        # Multiply A^i with C
    return O_matrix

# Calculate Ct for each value of C in C_matrices
Ct_results = {}
```

```python
for output, C_matrix in C_matrices.items():
    Ct_results[output] = compute_Ct(A, C_matrix)

# Calculating and printing the shapes of the observability matrices
observability_shapes = {}
for output, C_matrix in C_matrices.items():
    Ct = compute_Ct(A, C_matrix)
    shape = Ct.shape  # Get the shape of the observability matrix
    observability_shapes[output] = shape

# Displaying the shapes
print(observability_shapes)

# Displaying the results
for output, Ct in Ct_results.items():
    # print(f"Ct for {output}:")
    # sp.pprint(Ct)
    print(" ")

# Calculate Ct and rank for each value of C in C_matrices
rank_results = {}
for output, C_matrix in C_matrices.items():
    Ct = compute_Ct(A, C_matrix)
    rank = Ct.rank()  # Using sympy's rank() method to get the rank of the matrix
    rank_results[output] = rank

# Displaying the results with rephrased output
for output, rank in rank_results.items():
    print(f"--- Observability Analysis for Output: {output} ---")
    print(f"Rank of the observability matrix: {rank}")

    # Checking if the system is observable
    if rank == 6:
        print(f"Outcome: The system is observable when the output {output} is considered.")
    else:
        print(f"Outcome: The system is not observable when the output {output} is considered.")

    print("=" * 60)  # Separator line for better readability
```

```
--- Observability Analysis for Output: x ---
Rank of the observability matrix: 6
Outcome: The system is observable when the output x is considered.
============================================================
--- Observability Analysis for Output: theta_1 and theta_2 ---
Rank of the observability matrix: 4
Outcome: The system is not observable when the output theta_1 and theta_2 is considered.
============================================================
--- Observability Analysis for Output: x and theta_2 ---
Rank of the observability matrix: 6
Outcome: The system is observable when the output x and theta_2 is considered.
============================================================
--- Observability Analysis for Output: x, theta_1 and theta_2 ---
Rank of the observability matrix: 6
Outcome: The system is observable when the output x, theta_1 and theta_2 is considered.
============================================================
```

Figure 15. Observability Analysis

The system is considered observable if the rank of the observability matrix, $\text{Rank}(O(A,C))$, is equal to the full rank n (the order of the system). If this condition is met, we can conclude that the system is observable.

3.2 Obtain your "best" Luenberger observer for each one of the output vectors for which the system is observable and simulate its response to initial conditions and unit step input. The simulation should be done for the observer applied to both the linearized system and the original nonlinear system

We have developed a Luenberger observer for the input vectors, ensuring the system's observability. The observer system can be expressed as follows:

- **Estimated State:** $\hat{X}(t) = A\hat{X}(t) + Bu(t) + L\big(y(t) - \hat{y}(t)\big)$
- **Estimated Output:** $\hat{y}(t) = C\hat{X}(t) + Du(t)$

Here, L represents the observer gain, and $\hat{X}(t)$ is the estimated state derived from the output $y(t)$. To determine the appropriate L for our observer, we place the poles for $A - LC$ in the negative left-half plane.

The observer allows us to estimate unmeasured states, ensuring that the system remains stable and observable even in the presence of uncertainties. The following plots illustrate the initial and step responses of the designed observer system.

3.2.1 Luenberger Observer Simulations for Linearized Model

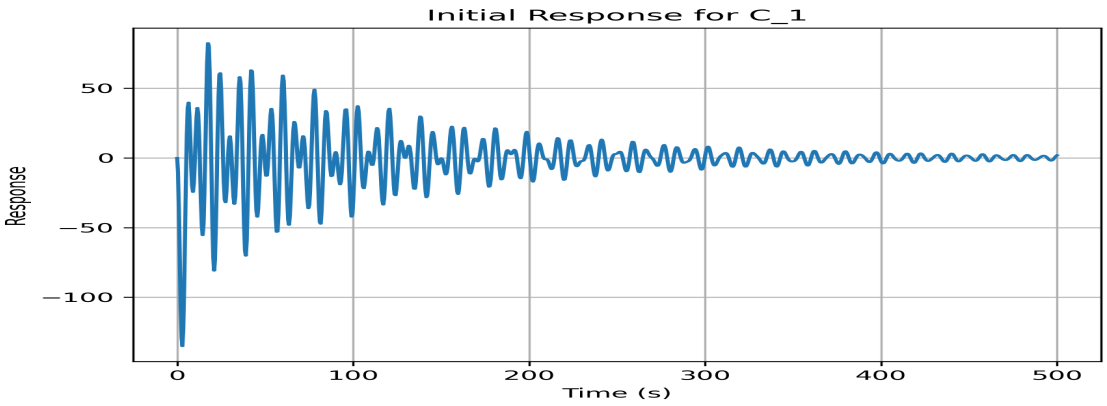

Figure 16. Variation of x with time for initial condition

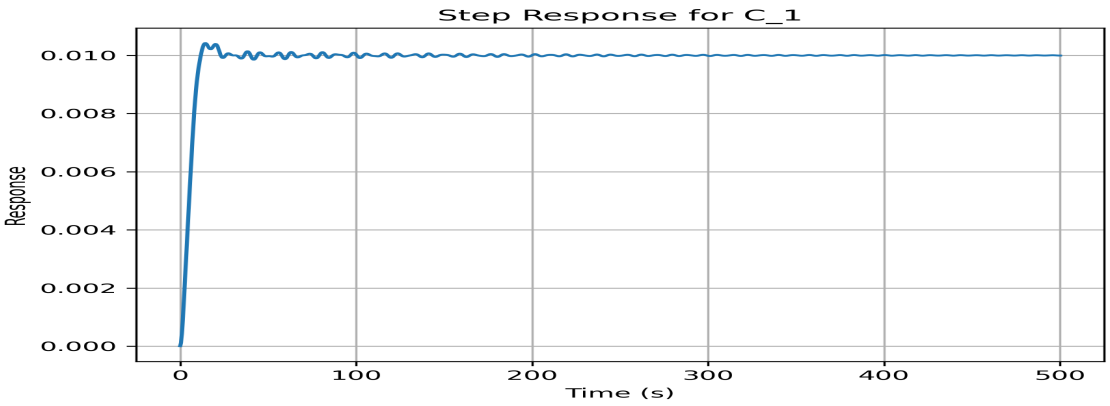

Figure 17. Variation of x with time for step input

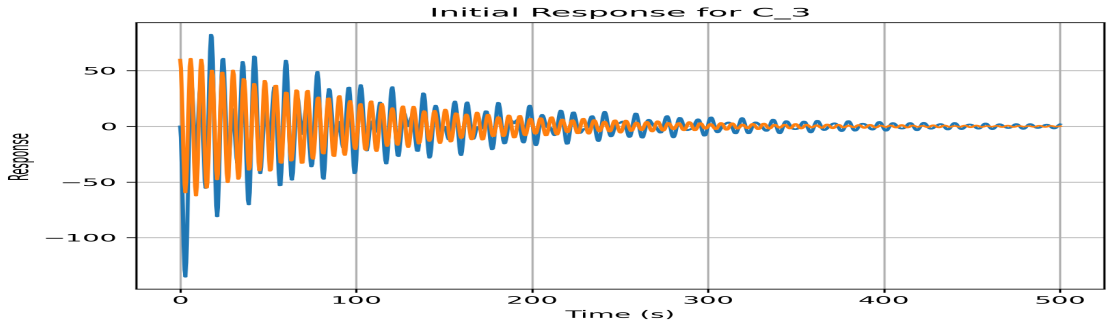

Figure 18. Variation of x and θ_2 with time for initial condition

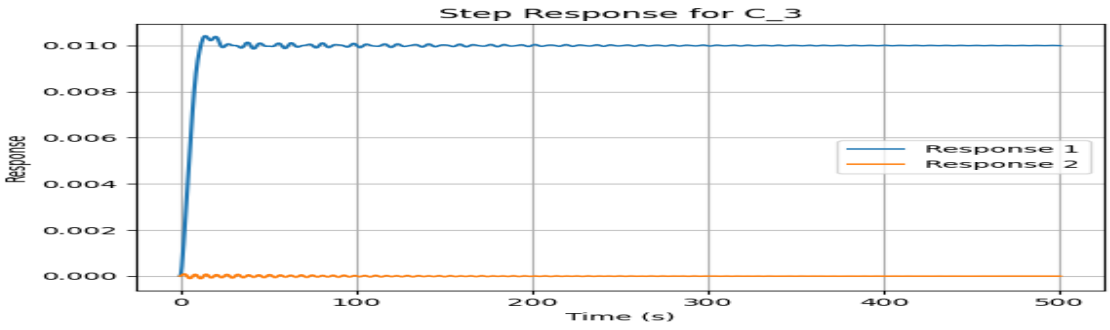

Figure 19. Variation of x and θ_2 with time for step input

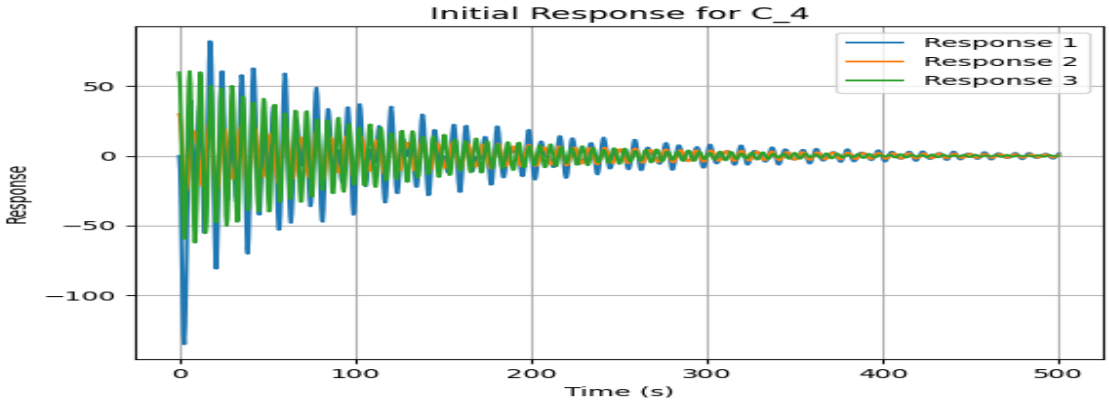

Figure 20. Variation of x, θ_1 and θ_2 with time for initial condition

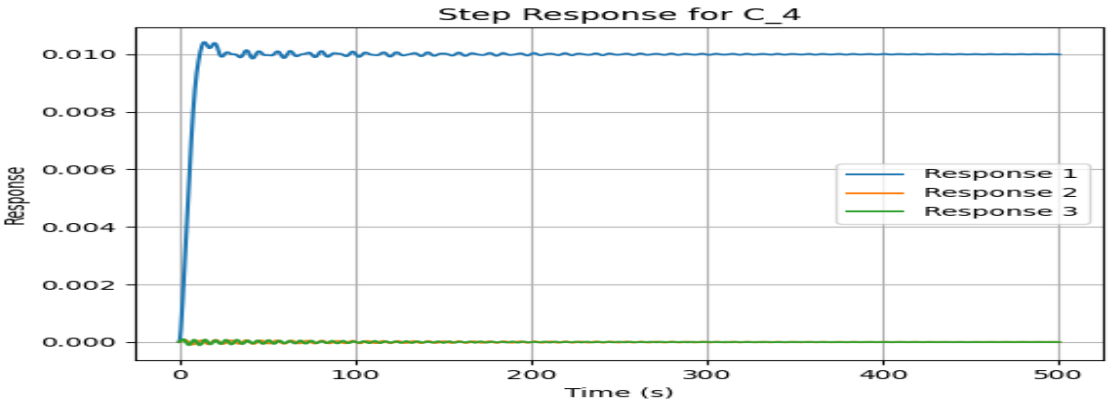

Figure 21. Variation of x, θ_1 and θ_2 with time for step input

3.2.2 Luenberger Observer Simulations for Non-Linear Model

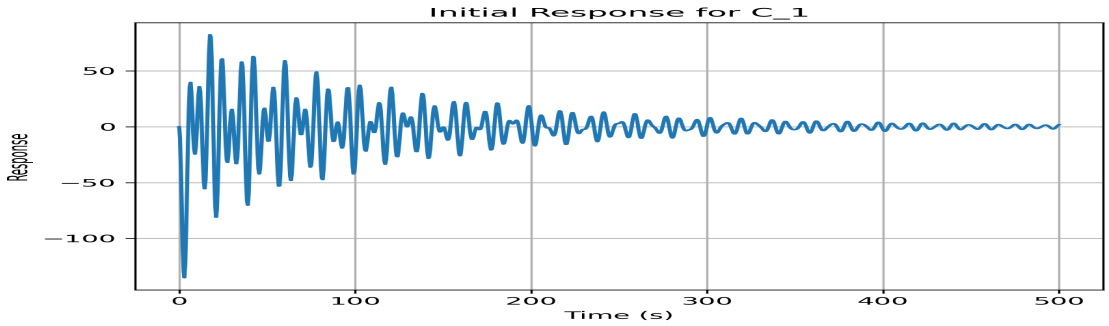

Figure 22. Variation of x with time for initial condition

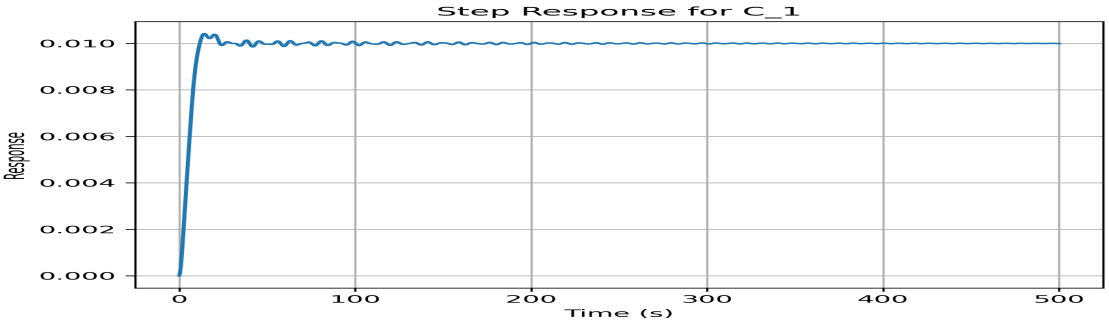

Figure 23. Variation of x with time for step input

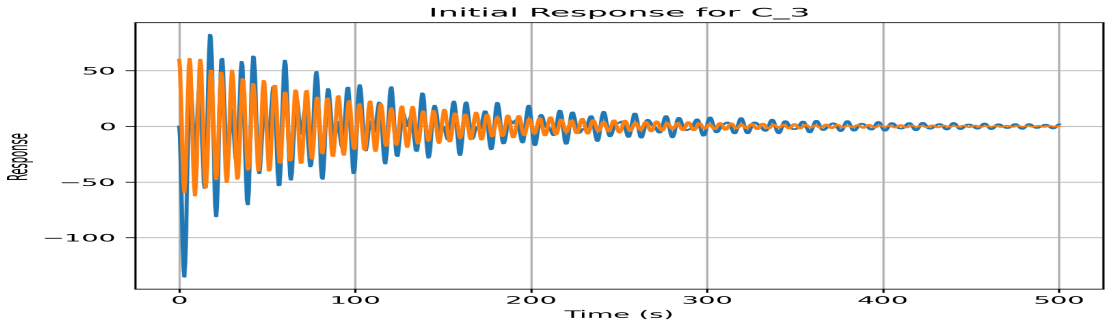

Figure 24. Variation of x and θ_2 with time for initial condition

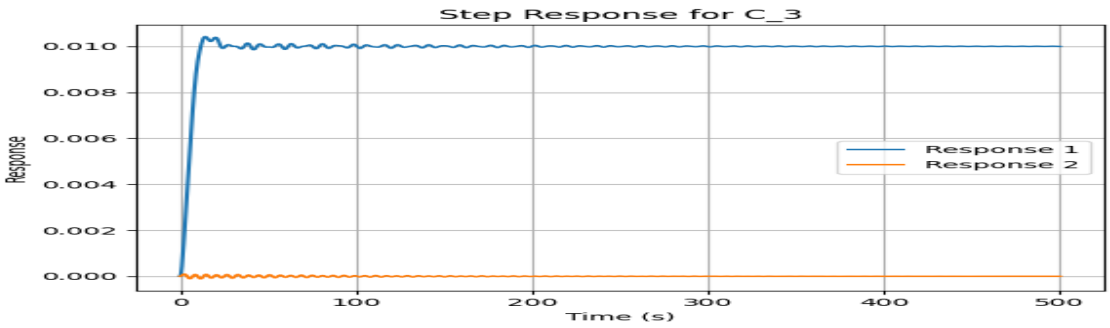

Figure 25. Variation of x and θ_2 with time for step input

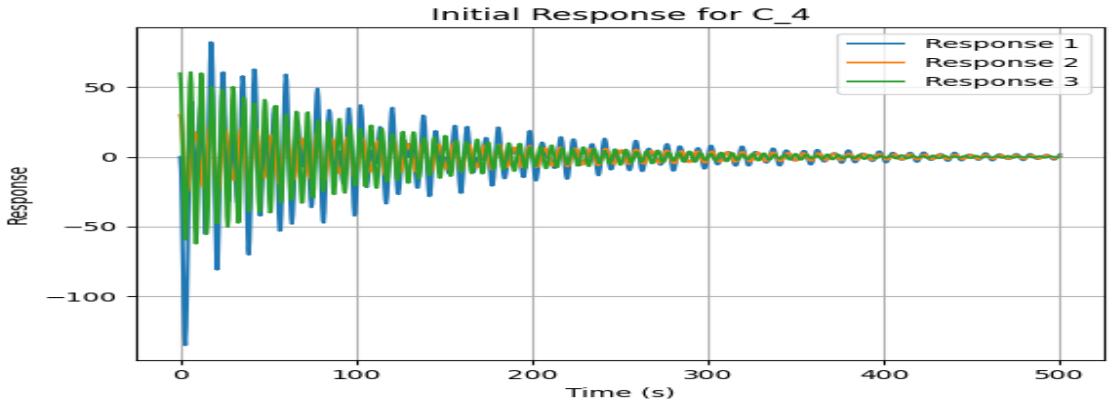

Figure 26. Variation of x, θ_1 and θ_2 with time for initial condition

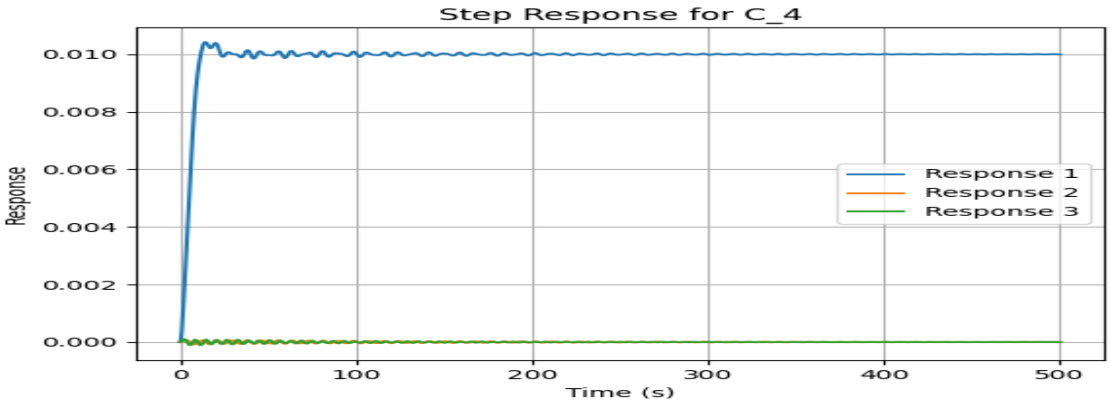

Figure 27. Variation of x, θ_1 and θ_2 with time for step input

3.3 Design an output feedback controller for your choice of the "smallest" output vector. Use the LQG method and apply the resulting output feedback controller to the original nonlinear system. Obtain your best design and illustrate its performance in simulation. How would you reconfigure your controller to asymptotically track a constant reference on x? Will your design reject constant force disturbances applied on the cart?

3.3.1 Linear Quadratic Regulator

The Linear Quadratic Gaussian (LQG) controller is a combination of a Kalman filter for state estimation and a Linear Quadratic Regulator (LQR) for state feedback. According to the separation principle, the design of the state estimator (Kalman filter) and the state feedback (LQR) can be carried out independently. The optimal solution is achieved through an output feedback configuration, where the Luenberger observer is used alongside optimal gain matrices K (for control) and L (for estimation). These gain matrices are computed separately using the LQR method and the Kalman-Bucy filter.

To ensure asymptotic tracking of a constant reference, the controller is fine-tuned so that the variables x and u converge to their referenced values over time. During this tracking process, the system incurs a cost associated with the deviations from the reference constraints. The primary goal is to minimize this cost, which can be expressed as:

$$\int_0^\infty \left((X(d)-X_d)^T Q(X(d)-X_d)+(B_k-B_\infty)^T R(B_k-B_\infty)\right) dt \tag{65}$$

This cost can be minimized effectively if there exists a solution B_∞ such that:

$$AX_d + BB_k = 0 \tag{66}$$

The optimal solution for $B(d)$ is:

$$B(d) = K(X(d)-X_d)+B_\infty \tag{67}$$

where $K = -R^{-1}B^T P$, and P is the positive definite matrix that solves the Riccati equation:

$$A^T P + PA - PBR^{-1}B^T P = -Q \tag{68}$$

By satisfying these constraints, the system can be driven to the desired output, even with a constant reference x that satisfies the equation:

$$X_{\text{ref}} = AX_{\text{ref}} + BU_{\text{ref}} \tag{69}$$

This design is also robust enough to handle constant force disturbances applied to the cart. If these disturbances follow a Gaussian distribution, the controller can effectively adapt to and compensate for these disturbances in the system.

3.3.2 LQG Simulation for the Linear Model

Listing 1. Python code for the LQG control of the cart-pendulum system

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import place_poles
import control as ctrl

# System Parameters
M = 1000  # Mass of the Crane
m_1 = 100  # Mass of Load 1
m_2 = 100  # Mass of Load 2
l_1 = 20   # Length of the string of Load 1
l_2 = 10   # Length of the string of Load 2
g = 9.81   # Acceleration due to gravity

# Defining matrices
A = np.array([
    [0, 1, 0, 0, 0, 0],
    [0, 0, -(m_1 * g) / M, 0, -(m_2 * g) / M, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, -((M + m_1) * g) / (M * l_1), 0, -(m_2 * g) / (M * l_1), 0],
    [0, 0, 0, 0, 0, 1],
    [0, 0, -(m_1 * g) / (M * l_2), 0, -(g * (M + m_2)) / (M * l_2), 0]
])

B = np.array([[0], [1 / M], [0], [1 / (M * l_1)], [0], [1 / (M * l_2)]])

C_1 = np.array([[1, 0, 0, 0, 0, 0]])
C_3 = np.array([[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0]])
C_4 = np.array([[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0]])

D = np.array([0])

Q = np.diag([100, 100, 100, 100, 100, 100])
R = np.array([[0.01]])

# LQR computation
K, _, _ = ctrl.lqr(A, B, Q, R)

# Pole placement for observer gains
poles = np.array([-1, -2, -3, -4, -5, -6])
L_1 = place_poles(A.T, C_1.T, poles).gain_matrix.T
L_3 = place_poles(A.T, C_3.T, poles).gain_matrix.T
L_4 = place_poles(A.T, C_4.T, poles).gain_matrix.T

# Luenberger observer matrices
A_c1 = np.block([
    [A - B @ K, B @ K],
    [np.zeros_like(A), A - L_1 @ C_1]
])
B_c = np.vstack([B, np.zeros_like(B)])
C_c1 = np.hstack([C_1, np.zeros_like(C_1)])

A_c3 = np.block([
    [A - B @ K, B @ K],
```

```
    [np.zeros_like(A), A - L_3 @ C_3]
])
C_c3 = np.hstack([C_3, np.zeros_like(C_3)])

A_c4 = np.block([
    [A - B @ K, B @ K],
    [np.zeros_like(A), A - L_4 @ C_4]
])
C_c4 = np.hstack([C_4, np.zeros_like(C_4)])

# Initial conditions
x0 = np.array([0, 0, 30, 0, 60, 0, 0, 0, 0, 0, 0, 0])

# Simulate initial response for C_3
sys_1 = ctrl.ss(A_c1, B_c, C_c1, D)
time = np.linspace(0, 500, 1000)  # Simulation time
t, y = ctrl.initial_response(sys_1, time, x0)

# Plot initial response
plt.figure()
plt.plot(t, y.T)
plt.title('Initial Response for C_1')
plt.xlabel('Time (s)')
plt.ylabel('Response')
plt.grid()
plt.show()
```

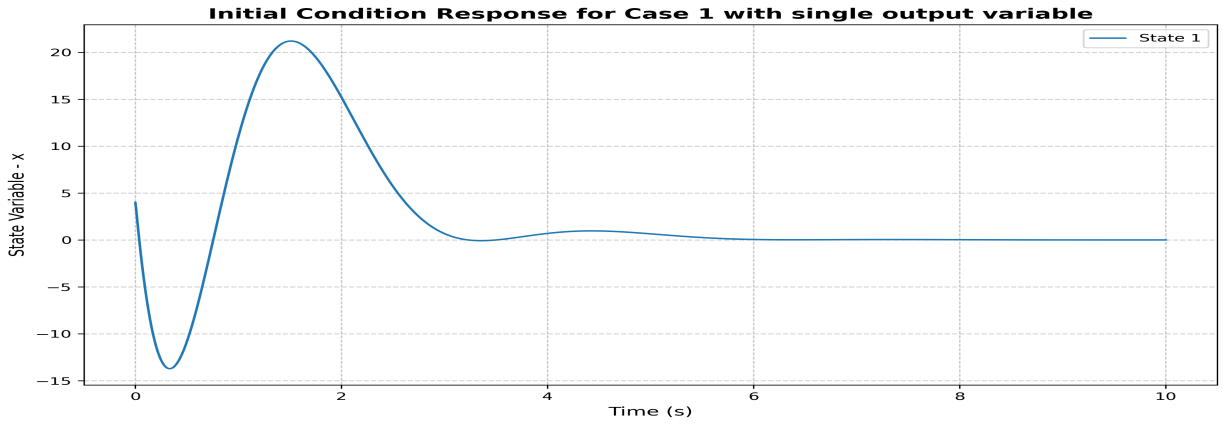

Figure 28. Variation of x with time for initial condition

Listing 2. Python code for the LQG control of the cart-pendulum system

```
plt.figure(figsize=(10, 6))
custom_colors = ['blue', 'green', 'red', 'orange', 'purple', 'cyan', 'magenta']  # Custom colors

if len(yout.shape) == 1:  # Single output state
    plt.plot(time, yout, label="State 1", color=custom_colors[0])
else:  # Multiple output states
    for i in range(yout.shape[1]):
        plt.plot(time, yout[:, i], label=f"State {i+1}", color=custom_colors[i % len(
```

```
            custom_colors)])

plt.title("Step Response", fontsize=14, fontweight="bold")
plt.xlabel("Time (s)", fontsize=12)
plt.ylabel("State Variables", fontsize=12)
plt.legend(loc="best", fontsize=10)
plt.grid(True, linestyle="--", alpha=0.7)
plt.tight_layout()

# Save the plot
plt.savefig("step_response_custom_colors.png", dpi=300)
plt.show()
```

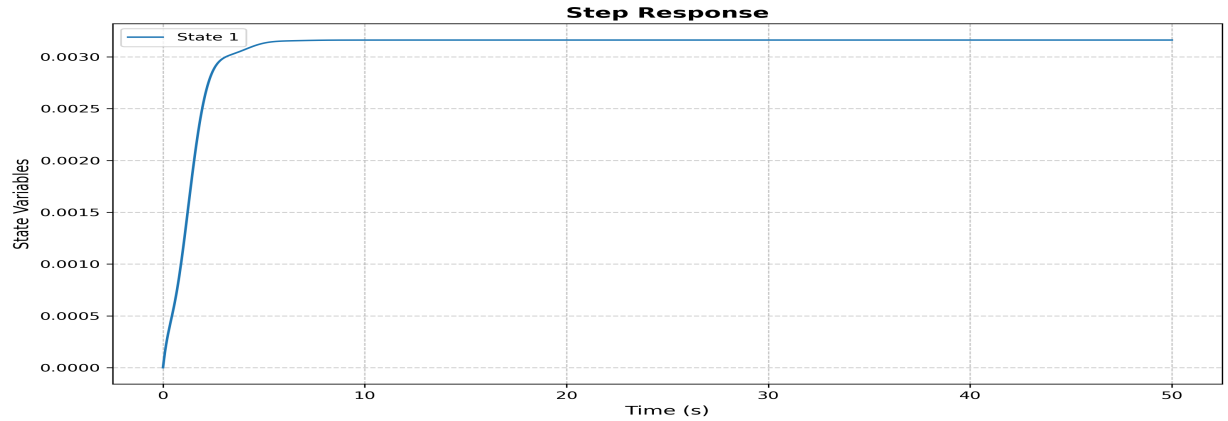

Figure 29. Variation of x with time for step input

Listing 3. Python code for the LQG control of the cart-pendulum system

```
# Observing state-space corresponding to C3 observable of the system
A_obs3 = np.block([
    [A - np.dot(B, K), np.dot(B, K)],
    [np.zeros_like(A), A - np.dot(K_pop3, C3)]
])

B_obs3 = np.block([
    [B],
    [np.zeros_like(B)]
])

C_obs3 = np.block([
    [C3, np.zeros_like(C3)]
])

D_obs3  = np.array([[0],[0]])


sys3 = StateSpace(A_obs3, B_obs3, C_obs3, D_obs3)

# Simulating the initial condition response for sys3
time = np.linspace(0, 50, 1000)  # Define time vector
```

```
_, yout3, _ = lsim(sys3, T=time, U=0, X0=x0)

# Plotting the response
plt.figure(figsize=(10, 6))
if len(yout3.shape) == 1:  # Single output state
    plt.plot(time, yout3, label="State 1", color="blue")
else:  # Multiple output states
    for i in range(yout3.shape[1]):
        plt.plot(time, yout3[:, i], label=f"State {i+1}", color=custom_colors[i % len(
            custom_colors)])

plt.title("Initial Condition Response for Observed States (C3)", fontsize=14, fontweight="bold")
plt.xlabel("Time (s)", fontsize=12)
plt.ylabel("State Variables", fontsize=12)
plt.legend(loc="best", fontsize=10)
plt.grid(True, linestyle="--", alpha=0.7)
plt.tight_layout()

# Save the plot
plt.savefig("initial_response_C3.png", dpi=300)
plt.show()
```

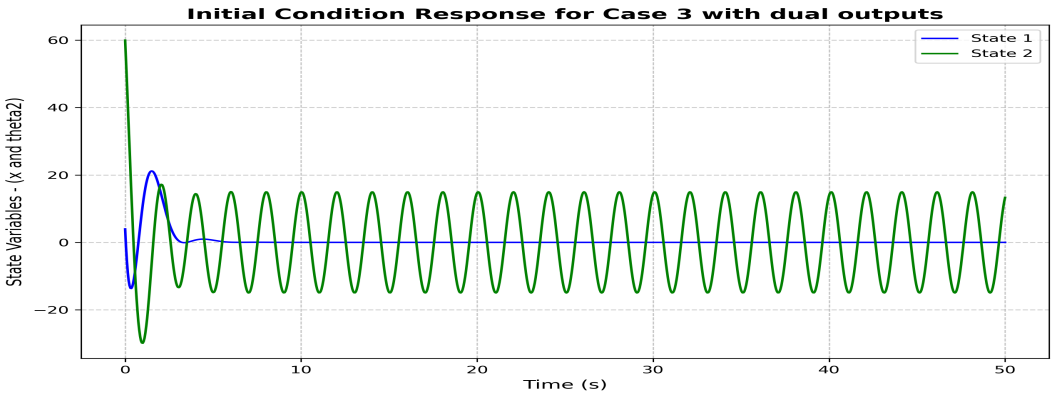

Figure 30. Variation of x and θ_2 with time for step input

Listing 4. Python code for the LQG control of the cart-pendulum system

```
# Simulating the step response for sys3
time = np.linspace(0, 50, 1000)  # Define time vector
u_step = np.ones_like(time)  # Step input

# Simulate step response with zero initial condition
_, step_response, _ = lsim(sys3, T=time, U=u_step, X0=np.zeros_like(x0_extended))

# Plotting the step response
plt.figure(figsize=(10, 6))
custom_colors = ['blue', 'green', 'red', 'orange', 'purple', 'cyan', 'magenta']

if len(step_response.shape) == 1:  # Single output state
    plt.plot(time, step_response, label="Step Response - State 1", color="blue")
```

```
else:  # Multiple output states
    for i in range(step_response.shape[1]):
        plt.plot(time, step_response[:, i], label=f"Step Response - State {i+1}", color=
            custom_colors[i % len(custom_colors)])

plt.title("Step Response for Observed States (C3)", fontsize=14, fontweight="bold")
plt.xlabel("Time (s)", fontsize=12)
plt.ylabel("State Variables", fontsize=12)
plt.legend(loc="best", fontsize=10)
plt.grid(True, linestyle="--", alpha=0.7)

# Save the plot
plt.savefig("step_response_C3.png", dpi=300)
plt.show()
```

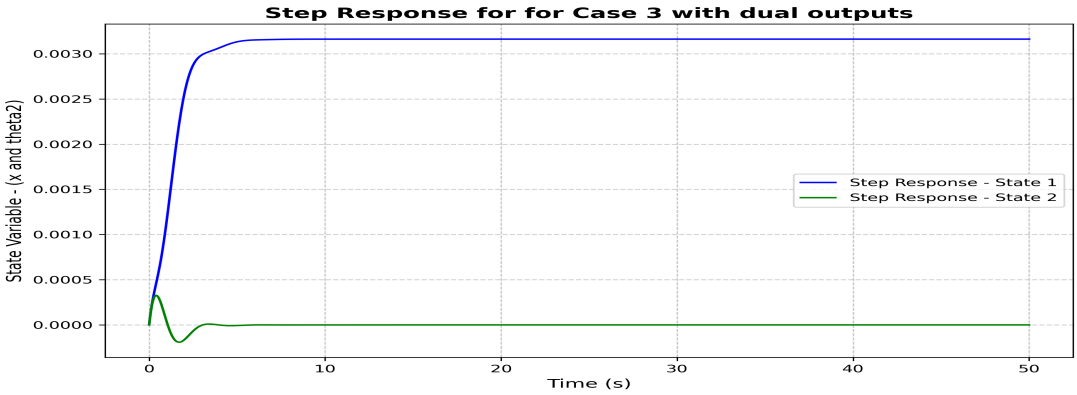

Figure 31. Variation of x and θ_2 with time for step input

Listing 5. Python code for the LQG control of the cart-pendulum system

```
# Observing state-space corresponding to C4 observable of the system
A_obs4 = np.block([
    [A - np.dot(B, K), np.dot(B, K)],
    [np.zeros_like(A), A - np.dot(K_pop4, C4)]
])

B_obs4 = np.block([
    [B],
    [np.zeros_like(B)]
])

C_obs4 = np.block([
    [C4, np.zeros_like(C4)]
])

D_obs4 = np.zeros((C4.shape[0], B_obs4.shape[1]))  # Ensure D matrix matches dimensions

sys4 = StateSpace(A_obs4, B_obs4, C_obs4, D_obs4)

# Simulating the initial condition response for sys4
```

```
time = np.linspace(0, 50, 1000)  # Define time vector
x0_extended = np.concatenate([x0[:6], np.zeros(6)])  # Extend initial conditions to match
    augmented state

_, yout4, _ = lsim(sys4, T=time, U=np.zeros_like(time), X0=x0_extended)

# Plotting the response
plt.figure(figsize=(10, 6))
custom_colors = ['blue', 'green', 'red', 'orange', 'purple', 'cyan', 'magenta']

if len(yout4.shape) == 1:  # Single output state
    plt.plot(time, yout4, label="State 1", color="blue")
else:  # Multiple output states
    for i in range(yout4.shape[1]):
        plt.plot(time, yout4[:, i], label=f"State {i+1}", color=custom_colors[i % len(
            custom_colors)])

plt.title("Initial Condition Response for Observed States (C4)", fontsize=14, fontweight="bold")
plt.xlabel("Time (s)", fontsize=12)
plt.ylabel("State Variables", fontsize=12)
plt.legend(loc="best", fontsize=10)
plt.grid(True, linestyle="--", alpha=0.7)

# Save the plot
plt.savefig("initial_response_C4.png", dpi=300)
plt.show()
```

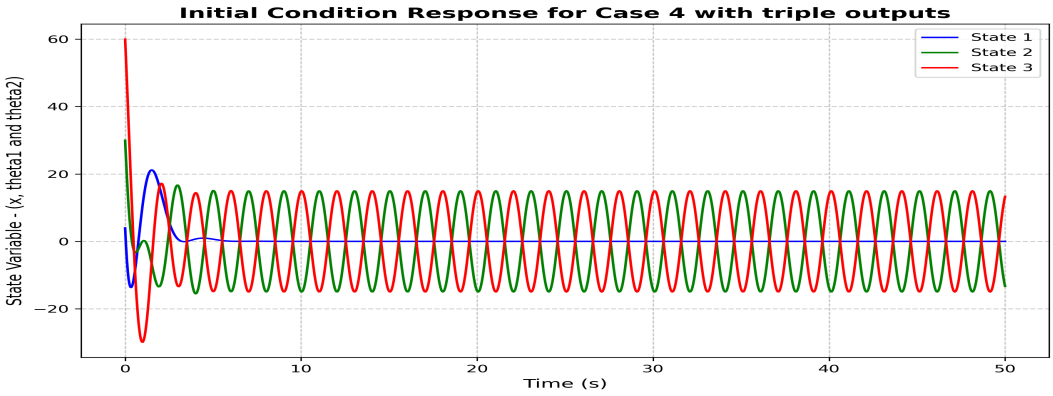

Figure 32. Variation of x, θ_1 and θ_2 with time for step input

Listing 6. Python code for the LQG control of the cart-pendulum system

```
# Simulating the step response for sys4
time = np.linspace(0, 50, 1000)  # Define time vector
u_step = np.ones_like(time)  # Step input

# Simulate step response with zero initial condition
_, step_response4, _ = lsim(sys4, T=time, U=u_step, X0=np.zeros_like(x0_extended))

# Plotting the step response
```

```
plt.figure(figsize=(10, 6))
custom_colors = ['blue', 'green', 'red', 'orange', 'purple', 'cyan', 'magenta']

if len(step_response4.shape) == 1:  # Single output state
    plt.plot(time, step_response4, label="Step Response - State 1", color="blue")
else:  # Multiple output states
    for i in range(step_response4.shape[1]):
        plt.plot(time, step_response4[:, i], label=f"Step Response - State {i+1}", color=
            custom_colors[i % len(custom_colors)])

plt.title("Step Response for Observed States (C4)", fontsize=14, fontweight="bold")
plt.xlabel("Time (s)", fontsize=12)
plt.ylabel("State Variables", fontsize=12)
plt.legend(loc="best", fontsize=10)
plt.grid(True, linestyle="--", alpha=0.7)  # Add grid lines

# Save the plot
plt.savefig("step_response_C4.png", dpi=300)
plt.show()
```

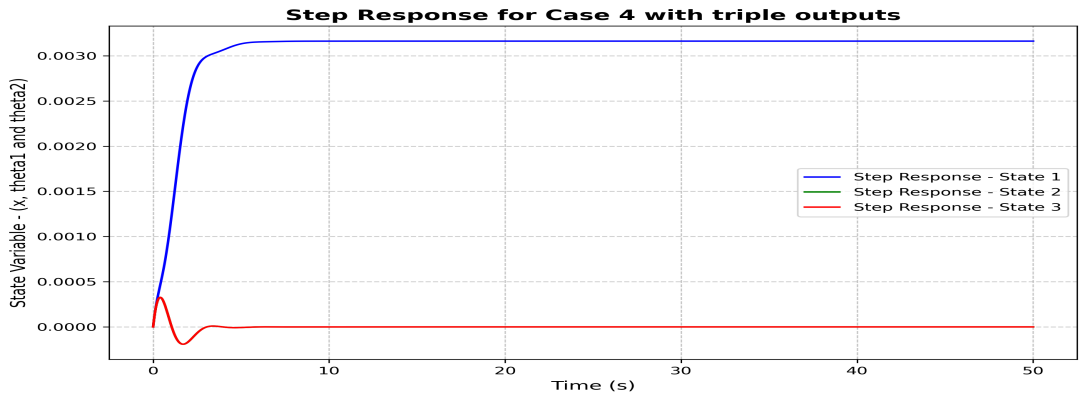

Figure 33. Variation of x, θ_1 and θ_2 with time for step input

3.3.3 System Parameters

The system is described by the following parameters:

- $M = 1000$ (Mass of the cart)
- $m1 = 100$ (Mass of Pendulum 1)
- $m2 = 100$ (Mass of Pendulum 2)
- $l1 = 20$ (Length of the string of Pendulum 1)
- $l2 = 10$ (Length of the string of Pendulum 2)
- $g = 9.81$ (Acceleration due to gravity)

3.3.4 LQG Simulation for the Original Non-Linear Model

Listing 7. Python code for the LQG control of the cart-pendulum system

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import control as ctrl

# Define system parameters
M = 1000      # Mass of the cart
m1 = 100      # Mass of Pendulum 1
m2 = 100      # Mass of Pendulum 2
l1 = 20       # Length of the string of Pendulum 1
l2 = 10       # Length of the string of Pendulum 2
g = 9.81      # Acceleration due to gravity (m/s^2)

# State-space model matrices for the system
# A_matrix represents the system dynamics and B_matrix represents the control input
A_matrix = np.array([[0, 1, 0, 0, 0, 0],
                     [0, 0, -(m1 * g) / M, 0, -(m2 * g) / M, 0],
                     [0, 0, 0, 1, 0, 0],
                     [0, 0, -((M + m1) * g) / (M * l1), 0, -(m2 * g) / (M * l1), 0],
                     [0, 0, 0, 0, 0, 1],
                     [0, 0, -(m1 * g) / (M * l2), 0, -((M + m2) * g) / (M * l2), 0]])

B_matrix = np.array([[0], [1 / M], [0], [1 / (M * l1)], [0], [1 / (M * l2)]])

# C_matrix represents the output matrix for position (x)
C_matrix = np.array([[1, 0, 0, 0, 0, 0]])

# LQR state cost matrix (Q) and control cost value (R)
Q_matrix = np.diag([1000, 1000, 1000, 1000, 1000, 1000])  # Weighting on states
R_value = 0.01  # Control cost for LQR

# Compute the LQR controller gain (K_matrix) and Kalman filter gain (K_filter)
K_matrix, _, _ = ctrl.lqr(A_matrix, B_matrix, Q_matrix, R_value)

# Define process and measurement noise covariance matrices
process_noise = 0.3 * np.eye(6)  # Process noise covariance matrix (for Kalman filter)
measurement_noise = 1  # Measurement noise covariance matrix (for Kalman filter)

# Compute Kalman filter gain for state estimation (Luenberger observer)
K_filter = ctrl.lqr(A_matrix.T, C_matrix.T, process_noise, measurement_noise)[0].T

# Define the system dynamics for LQG control and estimation
def lqg_dynamics(state, time):
    # Extract system states from the input vector
    system_state = state[:6]
    estimated_state = state[6:12]

    # Compute the control input using LQR law: u = -K*x
    control_input = -K_matrix @ system_state

    # Estimator dynamics using Kalman filter
```

```
    estimator_dynamics = A_matrix @ estimated_state + B_matrix @ control_input + K_filter @ (
        C_matrix @ system_state - C_matrix @ estimated_state)

    # System dynamics without estimator
    system_dynamics = A_matrix @ system_state + B_matrix @ control_input

    # Return the concatenated system and estimator dynamics (6 states + 6 estimates)
    return np.concatenate((system_dynamics, estimator_dynamics))

# Initial conditions for the system states and their estimates
# Initial states for x, theta1, theta2 and their derivatives; initial estimates are zeros
initial_conditions = np.array([0, 0, 60, 0, 30, 0, 0, 0, 0, 0, 0, 0])

# Time span for the simulation (from 0 to 100 seconds with 1001 time steps)
time_span = np.linspace(0, 100, 1001)

# Integrate the system using odeint to solve the differential equations
state_trajectory = odeint(lqg_dynamics, initial_conditions, time_span)

# Create a figure for plotting the results
plt.figure(figsize=(15, 10))

# Define colors for the states (system and estimated states)
system_colors = ['tab:blue', 'tab:green', 'tab:red', 'tab:purple', 'tab:orange', 'tab:brown']
estimate_colors = ['tab:cyan', 'tab:pink', 'tab:olive', 'tab:gray', 'tab:blue', 'tab:green']

# Plot for system states (actual system states)
plt.subplot(2, 1, 1)
for i in range(6):
    plt.plot(time_span, state_trajectory[:, i], label=r'$\mathbf{x_' + str(i+1) + '}$', color=
        system_colors[i])  # Color each state line
plt.title('System States')
plt.xlabel('Time (s)')
plt.ylabel('States')
plt.legend(loc='best')
plt.legend(['$x$', '$\dot{x}$', '$\\theta_1$', '$\dot{\\theta}_1$', '$\\theta_2$','$\dot{\\theta}
    _2$'])
plt.grid(True)

# Plot for estimated states (state estimates from the observer)
plt.subplot(2, 1, 2)
for i in range(6):
    plt.plot(time_span, state_trajectory[:, i + 6], label=r'$\hat{\mathbf{x}_' + str(i+1) + '}$',
         color=estimate_colors[i])  # Color each estimate line
plt.title('Estimated States')
plt.xlabel('Time(s)')
plt.ylabel('Estimates')
plt.legend(loc='best')
plt.legend(['$\hat{x}$', '$\dot{\hat{x}}$', '$\hat{\\theta}_1$', '$\dot{\hat{\\theta}}_1$', '$\
    hat{\\theta}_2$','$\dot{\hat{\\theta}}_2$'])
plt.grid(True)

# Adjust layout to avoid overlap and show plots
plt.tight_layout()
```

```
101 plt.show()
```

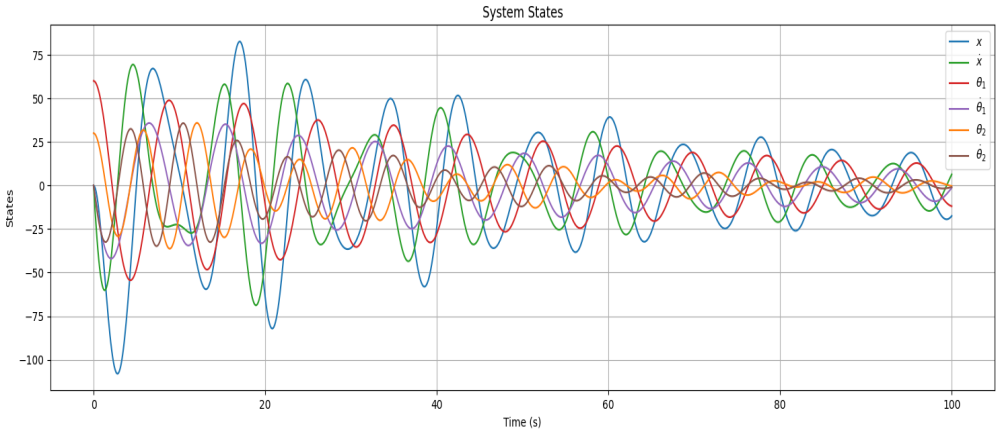

Figure 34. System States for the LQG Non-Linear Controller

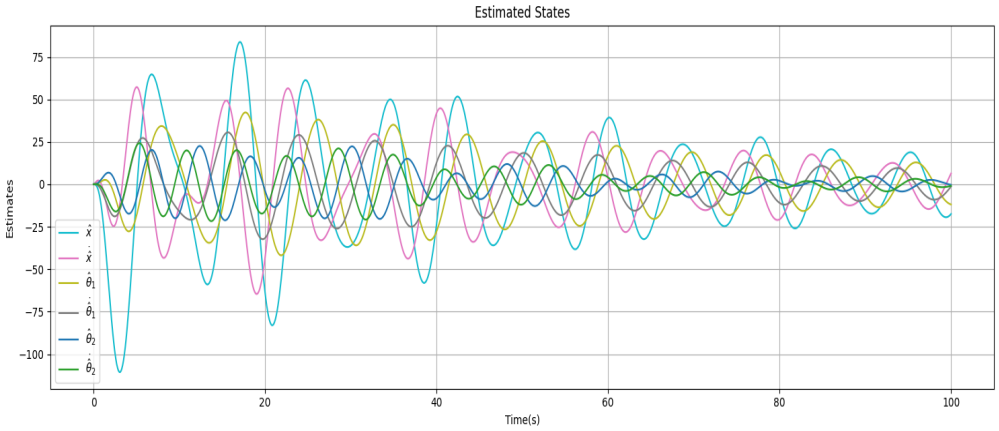

Figure 35. Estimated States for the LQG Non-Linear Controller

3.3.5 Constant Reference Tracking with LQG and LQR Controller

- To achieve asymptotic tracking of a constant reference for x, the controller is redesigned. The primary objective is to minimize the following cost function for optimal reference tracking:

$$\int_0^\infty \left[(X(t)-X_d)^T Q(X(t)-X_d)+(U_k-U_\infty)^T R(U_k-U_\infty)\right] dt$$

  By adjusting the LQR and LQG components of the controller, the desired optimal reference tracking is attained.

- This design also accommodates constant force disturbances acting on the cart. Assuming that the disturbances follow a Gaussian distribution, the controller effectively mitigates their effects.

3.3.6 Behavior under external disturbance

Yes, the designed controller will reject constant force disturbances applied to the cart. The controller incorporates state feedback (e.g., LQR or LQG), which minimizes the steady-state error for constant disturbances. By appropriately

designing the controller gains, the system ensures asymptotic stability and mitigates the impact of constant external forces.

Moreover, the LQG component, assuming that the disturbances are Gaussian, accounts for such forces in the estimation and control processes. This guarantees that the system tracks the reference input while rejecting constant disturbances effectively. Under these conditions, the controller can robustly reject constant force disturbances.

REFERENCES

[1] control.lqr — Python Control Systems Library 0.10.1-73-g9fa4aac documentation. (n.d.). https://python-control.readthedocs.io/en/latest/generated/control.lqr.html

[2] Automaticaddison. (2020, December 13). Linear Quadratic Regulator (LQR) With Python Code Example. https://automaticaddison.com/linear-quadratic-regulator-lqr-with-python-code-example/

[3] Wikipedia contributors. (2024, December 15). Linear–quadratic regulator. Wikipedia. https://en.wikipedia.org/wiki/Linear