Final Project

ENPM667: Control of Robotic Systems
Munyaradzi P Antony (UID: 120482731)
Varad Nerlekar (UID: 120501135)

Under the guidance of Dr. Waseem Malik

Contents

1 First Component **3**
1.1 Assumptions . . . 3
1.2 Equations Of Motion And Corresponding Nonlinear State-space Representation . . . 4
1.3 Linearize The System Around An Equilibrium point . . . 7
1.4 Obtaining The Controllability And Stability Conditions On M, m_1, m_2, l_1, l_2 For The Linearized system. . . . 8
1.4.1 Checking That The System Is Controllable . . . 9
1.4.2 Obtain An LQR Controller . . . 9
1.4.3 Simulate The Resulting Response To Initial Conditions When The Controller Is Applied To The Linearized System . . . 10
1.4.4 Simulate The Resulting Response To Initial Conditions When The Controller Is Applied To The Original Nonlinear System . . . 11
1.4.5 Simulate The Resulting Response To Initial Conditions When The Controller Is Applied To The Original Nonlinear System . . . 12
1.4.6 Adjust The Parameters Of The LQR Cost Until You Obtain A Suitable Response. . . . 12

2 Second Component **13**
2.1 Observability . . . 13
2.1.1 Determine For Which Output Vectors The Linearized System Is Observable. . . . 13
2.2 Obtain Your "Best" Luenberger Observer For Each One Of The Output Vectors . . . 14
2.2.1 For The Linearized System . . . 14
2.2.2 For The Original nonlinear system . . . 15
2.3 Design An Output Feedback Controller For Your Choice Of The "Smallest" Output Vector - LQG Controller. . . . 15

3 Links **19**

4 Appendix **19**

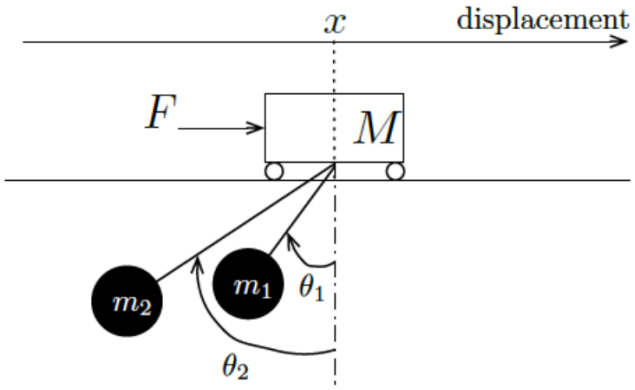
List of Figures

1 response plots - Linear system . . . 10
2 Response Plots - Nonlinear System . . . 11
3 Second Question (Component) . . . 12
4 Observability Check . . . 13
5 Luenberger Observer For Linear System(a). . . . 14
6 Luenberger Observer For Nonlinear System(b). . . . 15
7 LQG Controller For Non-Linear System . . . 16
8 LQG Controller For Non-Linear System . . . 16
9 LQG Controller For Non-Linear System . . . 17
10 LQG Controller For Non-Linear System . . . 18

1 First Component

Consider a crane that moves along an 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.

1. Obtain the equations of motion for the system and the corresponding nonlinear state-space representation.
2. Obtain the linearized system around the equilibrium point specified by $x = 0$ and $\theta_1 = \theta_2 = 0$. Write the state-space representation of the linearized system.
3. Obtain conditions on M, m_1, m_2, l_1, l_2 for which the linearized system is controllable.
4. Choose $M = 1000Kg$, $m_1 = m_2 = 100Kg$, $l_1 = 20m$ and $l_2 = 10m$. 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 also to the original nonlinear system. Adjust the parameters of the *LQR cost* until you obtain a suitable response. Use *Lyapunov's indirect method* to certify stability (locally or globally) of the closed-loop system



1.1 Assumptions

1. There is no friction between the cart and the ground.
2. The motion only happens along a single dimension.
3. The poles for the *Luenberger observer* are placed significantly far from the system's natural poles.
4. Damping is observed in the estimates, implying that the system will eventually reach an equilibrium point if given more time.
5. The acceleration due to gravity is assumed to be $9.8m/s^2$.
6. No collision occurs between the two loads in the system.
7. The length of the strings remains constant over time, with consistent tension in the strings.

1.2 Equations Of Motion And Corresponding Nonlinear State-space Representation

Defining the variables used to deriving the equations:

- $x(t)$: Position of the crane w.r.t. time
- $\theta_1(t)$: Angle of Load 1 with the vertical
- $\theta_2(t)$: Angle of Load 2 with the vertical
- m_1: Load 1′s mass
- m_2: Load 2′s mass
- M: Crane's mass
- l_1: Length of Cable 1
- l_2: Length of Cable 2
- g: Acceleration due to gravity
- $F(t)$: External force acting on the crane

—

We can express the position of Load 1 w.r.t. the variable θ_1 as:

$$\mathbf{x}_1 = (x - l_1 \sin(\theta_1))\hat{i} + (-l_1 \cos(\theta_1))\hat{j}$$

Differentiating x_{m_1} w.r.t. time gives us the expression for velocity as:

$$\frac{d}{dt}x_1 = \frac{d}{dt}(x - l_1 sin(\theta_1))\hat{i} + \frac{d}{dt}(-l_1 cos(\theta_1))\hat{j}$$

$$\vec{v_1} = (\dot{x} - l_1 \cos(\theta_1)\dot{\theta_1})\hat{i} + (l_1 \sin(\theta_1)\dot{\theta_1})\hat{j}$$

Similarly for Load 2, The expression for the position is given by:

$$\mathbf{x}_2 = (x - l_2 \sin(\theta_2))\hat{i} + (-l_2 \cos(\theta_2))\hat{j}$$

And the expression for the velocity is given by:

$$\vec{v_2} = (\dot{x} - l_2 \cos(\theta_2)\dot{\theta_2})\hat{i} + (l_2 \sin(\theta_2)\dot{\theta_2})\hat{j}$$

So, we have our equations for position for Load 1 and Load 2,

$$x_1 = (x - l_1 sin(\theta_1))\hat{i} + (-l_1 cos(\theta_1))\hat{j}$$

$$x_2 = (x - l_2 sin(\theta_2))\hat{i} + (-l_2 cos(\theta_2))\hat{j}$$

And for velocities,

$$\vec{v_1} = (\dot{x} - l_1 cos(\theta_1)\dot{\theta_1})\hat{i} + (l_1 sin(\theta_1)\dot{\theta_1})\hat{j}$$

$$\vec{v_2} = (\dot{x} - l_2 cos(\theta_2)\dot{\theta_2})\hat{i} + (l_2 sin(\theta_2)\dot{\theta_1})\hat{j}$$

We assumed that the displacement of the cart only occurs in the +ve x-direction, we can calculate the kinetic energy of the cart as:

$$K_M = \frac{1}{2}M\dot{x}^2$$

Similarly, the kinetic energy of Load 1 can be expressed as:

$$K_{m_1} = \frac{1}{2}m_1\dot{x_1}^2$$
$$K_{m_1} = \frac{1}{2}m_1(\dot{x} - l_1\dot{\theta_1}cos(\theta_1))^2 + \frac{1}{2}m_1(l_1\dot{\theta_1}sin(\theta_1))^2$$

And the kinetic energy of Load 2 as:

$$K_{m_2} = \frac{1}{2}m_2\dot{x_2}^2$$
$$K_{m_2} = \frac{1}{2}m_2(\dot{x} - l_2\dot{\theta_2}cos(\theta_2))^2 + \frac{1}{2}m_2(l_2\dot{\theta_2}sin(\theta_2))^2$$

$\therefore$ The Total Kinetic Energy of the system becomes:

$$K_{Total} = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m_1(\dot{x} - l_1\dot{\theta_1}cos(\theta_1))^2 + \frac{1}{2}m_1(l_1\dot{\theta_1}sin(\theta_1))^2 + \frac{1}{2}m_2(\dot{x} - l_2\dot{\theta_2}cos(\theta_2))^2 + \frac{1}{2}m_2(l_2\dot{\theta_2}sin(\theta_2))^2$$

Now, we calculate for the Potential Energy of the system:

$$P_{Total} = -m_1gl_1cos(\theta_1) - m_2gl_2cost(\theta_2)$$
$$P_{Total} = -g(m_1l_1cos(\theta_1) + m_2l_2cos(\theta_2))$$

With the obtained expressions for Kinetic and Potential Energies, we can write the Lagrange as:

$$\mathcal{L} = K_{Total} - P_{Total}$$

$$\mathcal{L} = \frac{1}{2}M\dot{x}^2+\frac{1}{2}m_1(\dot{x}-l_1\dot{\theta_1}cos(\theta_1))^2+\frac{1}{2}m_1(l_1\dot{\theta_1}sin(\theta_1))^2+\frac{1}{2}m_2(\dot{x}-l_2\dot{\theta_2}cos(\theta_2))^2+\frac{1}{2}m_2(l_2\dot{\theta_2}sin(\theta_2))^2+g(m_1l_1cos(\theta_1)+m_2l_2cos(\theta_2)$$

The Lagrange Equations of Motions [4]are given by:

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{q_i}} - \frac{\partial\mathcal{L}}{\partial q_i} = F$$

where, q_i are generalized coordinates (x, θ_1, θ_2) and Q_i are the generalized forces (external force F).

For $q_1 = x$,

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{x}} - \frac{\partial\mathcal{L}}{\partial x} = F$$

For $q_2 = \theta_1$,

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\theta_1}} - \frac{\partial\mathcal{L}}{\partial\theta_1} = 0$$

For $q_3 = \theta_2$,

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\theta_2}} - \frac{\partial\mathcal{L}}{\partial\theta_2} = 0$$

From the above equations, we can substitute with the values found from the above equations

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

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

Find the derivative w.r.t

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) = M\ddot{x} + (m_1 + m_2)\ddot{x} - \Big[m_1 l_1 \ddot{\theta}_1 \cos(\theta_1) - m_1 l_1 \dot{\theta}_1^2 \sin(\theta_1)\Big] - \Big[m_2 l_2 \ddot{\theta}_2 \cos(\theta_2) - m_2 l_2 \dot{\theta}_2^2 \sin(\theta_2)\Big]$$

Given that:

$$\frac{\partial \mathcal{L}}{\partial x} = 0$$

It implies that the equation becomes:

$$[M + m_1 + m_2]\ddot{x} - m_1 l_1 \ddot{\theta}_1 \cos(\theta_1) + m_1 l_1 \dot{\theta}_1^2 \sin(\theta_1) - m_2 l_2 \ddot{\theta}_2 \cos(\theta_2) + m_2 l_2 \dot{\theta}_2^2 \sin(\theta_2) = F$$

We have seen that:

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

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} = m_1 l_1^2 \dot{\theta}_1 - m_1 \dot{x}_1 l_1 \cos(\theta_1)$$

Find the derivative w.r.t:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1}\right) = m_1 l_1^2 \ddot{\theta}_1 - \Big[m_1 l_1 \ddot{x} \cos(\theta_1) - m_1 \dot{x}_1 l_1 \dot{\theta}_1 \sin(\theta_1)\Big] \quad (20)$$

Also here:

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = m_1 l_1 \dot{\theta}_1 \dot{x} \sin(\theta_1) - m_1 l_1 g \sin(\theta_1)$$

The two equations can be put together as:

$$m_1 l_1^2 \ddot{\theta}_1 - m_1 \dot{x}_1 l_1 \cos(\theta_1) + m_1 \dot{\theta}_1 \dot{x}_1 l_1 \sin(\theta_1) - m_1 \dot{\theta}_1 \dot{x}_1 l_1 \sin(\theta_1) + m_1 l_1 g \sin(\theta_1)$$

Addition and subtraction of like terms result in the following:

$$m_1 l_1^2 \ddot{\theta}_1 - m_1 \dot{x}_1 l_1 \cos(\theta_1) + m_1 l_1 g \sin(\theta_1) = 0$$

This is the equation derived from the second Lagrange Equation. Next, to obtain the third equation using the Lagrange Equation, we carry out the following calculations:

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

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}_2} = m_2 l_2^2 \dot{\theta}_2 - m_2 \dot{x}_1 l_2 \cos(\theta_2)$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_2}\right) = m_2 l_2^2 \ddot{\theta}_2 - \Big[m_2 \dot{x}_1 l_2 \cos(\theta_2) - m_2 \dot{\theta}_2 \dot{x}_1 l_2 \sin(\theta_2)\Big]$$

$$\left(\frac{\partial \mathcal{L}}{\partial \theta_2}\right) = m_2 \dot{x}_1 l_2 \dot{\theta}_2 \sin(\theta_2) - m_2 l_2 g \sin(\theta_2)$$

Therefore:

$$m_2 l_2^2 \ddot{\theta}_2 - m_2 \dot{x}_1 \ddot{x} \cos(\theta_2) + m_2 \dot{\theta}_2 \dot{x}_1 l_2 \sin(\theta_2) - m_2 \dot{\theta}_2 \dot{x}_1 l_2 \sin(\theta_2) + m_2 g l_2 \sin(\theta_2)$$

Addition and subtraction of like terms result in the following:

$$m_2 l_2^2 \ddot{\theta}_2 - m_2 l_2 \ddot{x} \cos(\theta_2) + m_2 g l_2 \sin(\theta_2) = 0$$

1.3 Linearize The System Around An Equilibrium point

From the equation derived above we can write the state governing the system as follows :

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

According to [1], linearization approximates a non-linear function near an equilibrium point using a first-order Taylor expansion. This simplifies nonlinear systems, allowing stability analysis with linear tools.The cart system with two pendulums exhibits nonlinear behavior due to trigonometric terms. Solving such systems directly is challenging.To simplify, we linearize around the equilibrium point $x = 0, \theta_1 = 0, \theta_2 = 0$, and use the approximations:

The approximations for linearization are given as:

$$\begin{aligned} \sin\theta_1 &\approx \theta_1, \\ \sin\theta_2 &\approx \theta_2, \\ \cos\theta_1 &\approx 1, \\ \cos\theta_2 &\approx 1, \\ \dot{\theta}_1^2 \approx \dot{\theta}_2^2 &\approx 0. \end{aligned}$$

and the Jacobian matrix utilized is as follows :

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial \dot{x}} & \frac{\partial F_1}{\partial \theta_1} & \frac{\partial F_1}{\partial \theta_2} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial \dot{x}} & \frac{\partial F_2}{\partial \theta_1} & \frac{\partial F_2}{\partial \theta_2} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial \dot{x}} & \frac{\partial F_3}{\partial \theta_1} & \frac{\partial F_3}{\partial \theta_2} \\ \frac{\partial F_4}{\partial x} & \frac{\partial F_4}{\partial \dot{x}} & \frac{\partial F_4}{\partial \theta_1} & \frac{\partial F_4}{\partial \theta_2} \\ \frac{\partial F_5}{\partial x} & \frac{\partial F_5}{\partial \dot{x}} & \frac{\partial F_5}{\partial \theta_1} & \frac{\partial F_5}{\partial \theta_2} \\ \frac{\partial F_6}{\partial x} & \frac{\partial F_6}{\partial \dot{x}} & \frac{\partial F_6}{\partial \theta_1} & \frac{\partial F_6}{\partial \theta_2} \end{bmatrix}$$

The resulting linearized equations of motion are:

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

From the above we can the following as the choice of the states

$$states = \left[x; \dot{x}; \theta_1; \dot{\theta}_1; \theta_2; \dot{\theta}_2\right]$$

where these can be written in the standard state space form [3]:

$$\mathbf{\dot{x}(t)} = \mathbf{Ax(t)} + \mathbf{Bu(t)}$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta_1} \\ \ddot{\theta}_1 \\ \dot{\theta_2} \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{m_1 g}{M} & 0 & -\frac{m_2 g}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{(M+m_1)g}{ML_1} & 0 & \frac{m_2 g}{ML_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{m_1 g}{ML_2} & 0 & \frac{(M+m_2)g}{ML_2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{ML_1} \\ 0 \\ \frac{1}{ML_2} \end{bmatrix} u$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta_1} \\ \ddot{\theta}_1 \\ \dot{\theta_2} \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{m_1 g}{M} & 0 & -\frac{m_2 g}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{(M+m_1)g}{ML_1} & 0 & \frac{m_2 g}{ML_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{m_1 g}{ML_2} & 0 & \frac{(M+m_2)g}{ML_2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{ML_1} \\ 0 \\ \frac{1}{ML_2} \end{bmatrix} F(t)$$

The output is given by the following equation :

$$\mathbf{y(t)} = \mathbf{Cx(t)} + \mathbf{Du(t)}$$

Expanding the C and D where $D = 0$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} F(t)$$

1.4 Obtaining The Controllability And Stability Conditions On M, m_1, m_2, l_1, l_2 For The Linearized system.

The following system properties are given: $M = 1000$ kg, $m_1 = m_2 = 100$ kg, $l_1 = 20$ m, and $l_2 = 10$ m.

A linear time-varying system ,according to [2] is deemed controllable over the interval $(0, T)$ if and only if the square Gramian matrix of controllability is nonsingular. For the Gramian matrix $C(A, B)$ to be nonsingular, the controllability matrix must meet the following conditions:

- The rank of $C(A, B)$ equals its full rank.
- The full rank is determined by the order of the matrix (n).

Defining the controllability Matrix:

$$R = [B\ AB\ A^2B\ A^3B\ A^4B\ A^5B]$$

To determine controllability, the determinant of matrix C is calculated as:

$$\text{rank} = |C| = \frac{g^6 l_1^2 - 2g^6 l_1 l_2 + g^6 l_2^2}{M^6 l_1^6 l_2^6}$$

This rank can be calculated as:

$$\text{rank} = ([B_k\ AB_k\ A^2B_k\ \ \ldots\ \ A^{n-1}B_k])_{n \times nm} = n$$

This results in the following constraints:

$$g^6 l_1^2 - 2g^6 l_1 l_2 + g^6 l_2^2 \neq 0$$
$$(g^3 l_1 - g^3 l_2)^2 \neq 0$$
$$(g^3 l_1)^2 \neq g^3 l_2$$
$$l_1 \neq l_2$$

This clearly shows that when $l_1 = l_2$, the determinant of the matrix becomes zero and when either l_1 or l_2 equals zero, controllability cannot be determined. The necessary conditions for controllability are:

$$l_1 \neq l_2,\ l_1 \neq 0,\ l_2 \neq 0$$

These conditions ensure that the system is controllable and enable the computation of controllability.

1.4.1 Checking That The System Is Controllable

Various controllers can be used to achieve system controllability and drive output values to desired levels. The controller type is determined by the system's model and attributes. PID (Proportional-Integral-Derivative) and LQR (Linear Quadratic Regulator) are two commonly used controllers, each with unique applications.

1.4.2 Obtain An LQR Controller

- PID Controller: The PID controller, which uses classical linear equations, is ideal for linear systems. It changes the control input based on the difference between the desired and actual outputs.
- LQR Controller: Non-linear models benefit especially from LQR. The optimal state-feedback law minimizes a specified quadratic objective function. LQR is considered one of the most effective controllers.

When A and B_k are stable, the LQR Controller seeks a control gain matrix k that minimizes the following cost function:

$$J\big(k, X(0)\big) = \int_0^\infty \Big(X^\top(t) Q X(t) + U^\top(t) R U(t)\Big)\, dt$$

Here:

- Q is a non-negative definite matrix that penalizes performance deviations.
- R is a positive-definite matrix that penalizes the effort or energy expended by control inputs.

Various Q and R values can be chosen to maximize system control. The output graphs validate the usefulness of the selected Q and R matrices.

- The Q value is determined by the system's ability to stabilize rapidly and function within an acceptable error range.
- Lowering the R value can speed up stabilization by reducing the energy required for control inputs.

1.4.3 Simulate The Resulting Response To Initial Conditions When The Controller Is Applied To The Linearized System

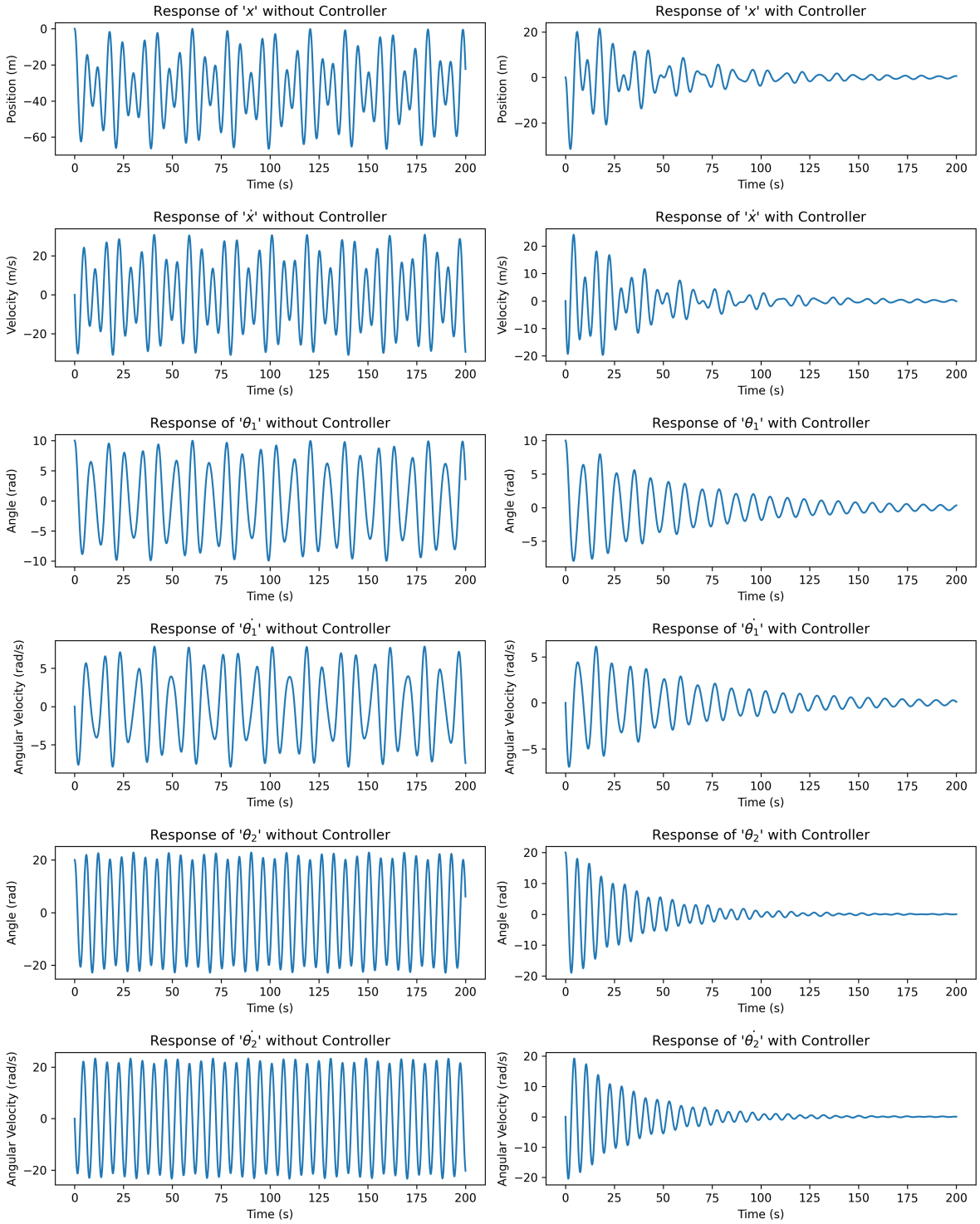


Figure 1: response plots - Linear system

1.4.4 Simulate The Resulting Response To Initial Conditions When The Controller Is Applied To The Original Nonlinear System

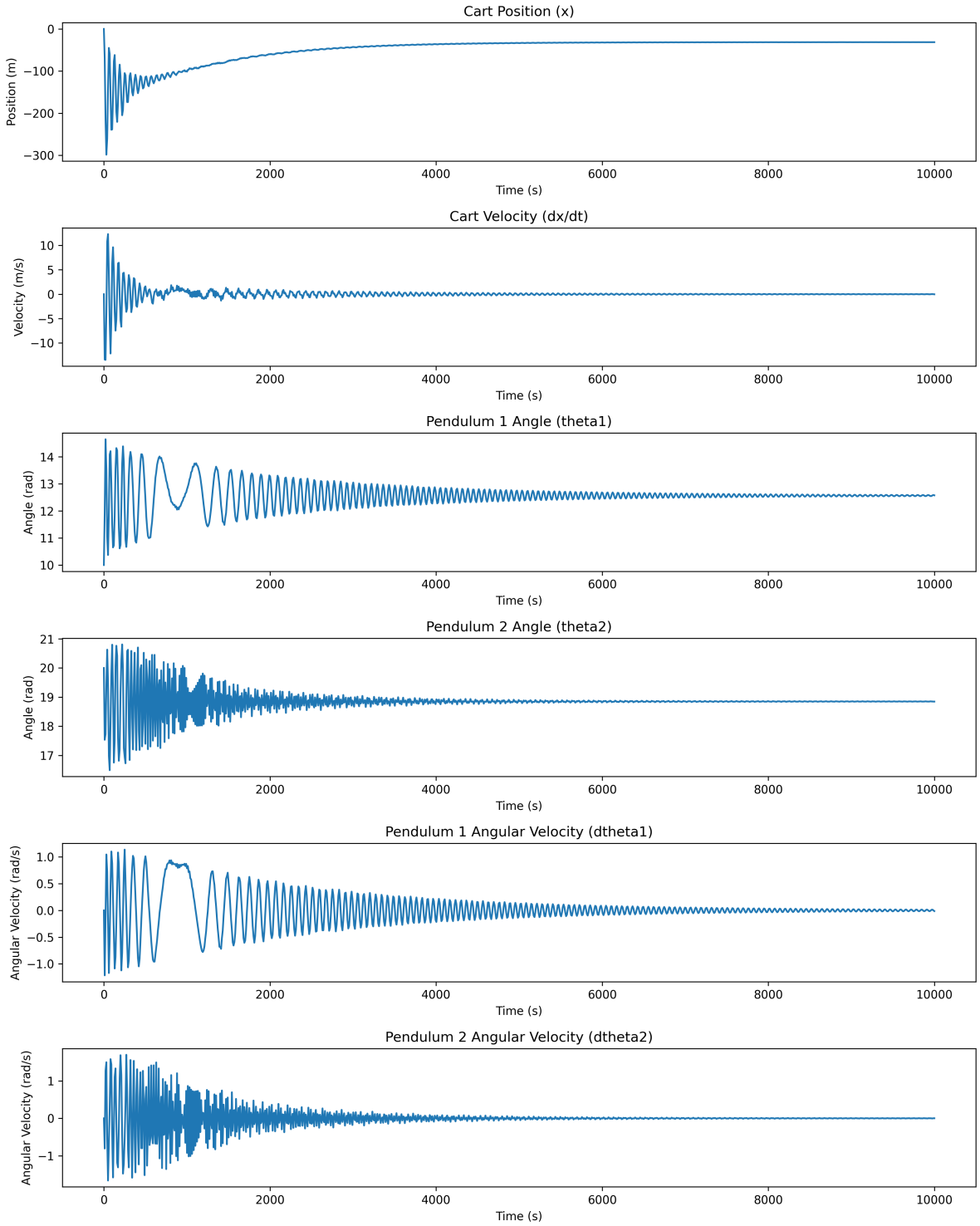


Figure 2: Response Plots - Nonlinear System

1.4.5 Simulate The Resulting Response To Initial Conditions When The Controller Is Applied To The Original Nonlinear System

Lyapunov's indirect approach of stability analysis, commonly known as the linearization method, examines the eigenvalues of the system's Jacobian matrix at an equilibrium point (in this case, matrix $A - BK$). The approach uses the following principle for continuous-time systems:

- If all eigenvalues have negative real portions, the equilibrium point is considered asymptotically stable.
- If an eigenvalue has a positive real portion, the equilibrium point becomes unstable.
- If some eigenvalues have zero real parts while the remainder have negative real parts, the system may be stable or unstable, requiring further investigation (the system is on the edge of stability).

1.4.6 Adjust The Parameters Of The LQR Cost Until You Obtain A Suitable Response.

The computed eigenvalues for the closed-loop system are as follows:

- $\lambda_1 = -0.383784665947216 + 0.354266878997317i$
- $\lambda_2 = -0.383784665947216 - 0.354266878997317i$
- $\lambda_3 = -0.0297862291376132 + 1.03473463598333i$
- $\lambda_4 = -0.0297862291376132 - 1.03473463598333i$
- $\lambda_5 = -0.0160677005129693 + 0.721304562445142i$
- $\lambda_6 = -0.0160677005129693 - 0.721304562445142i$

Observing the eigenvalues listed, we see that each has a negative real portion. We conclude that the system is asymptotically stable. This means that even minor deviations from the equilibrium state will eventually lead to a return to that condition. Complex eigenvalues suggest oscillatory activity as the system approaches convergence. The damping effect, represented by negative real parts, reduces oscillations over time and prevents uncontrolled growth.

The Lyapunov indirect stability study shows that the closed-loop system is stable with the specified LQR controller. The control approach successfully stabilizes the system near the equilibrium point.

Second Component (100 points): Consider the parameters selected in C) above.

E) Suppose that you can select the following output vectors: $x(t)$, $(\theta_1(t), \theta_2(t))$, $(x(t), \theta_2(t))$ or $(x(t), \theta_1(t), \theta_2(t))$. Determine for which output vectors the linearized system is observable.

F) 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.

G) 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 ?

Figure 3: Second Question (Component)

2 Second Component

Upon building an LQR controller for the system, the subsequent stage is to assess the system's observability with respect to specific output vectors. The metrics employed to determine controllability conditions are essential in this evaluation.

The project's continuance requires us to:

1. **Identify the optimal Luenberger Observer for each output vector, contingent upon their observability.** This entails selecting suitable observer structures to accurately estimate the system's state variables.
2. **Simulate the system's response to designated input circumstances, including a unit step input, for both the linearized and non-linearized versions of the system.** This phase enables us to evaluate the practical performance of the designed controllers and observers.
3. **Formulate an output feedback controller utilizing the LQG (Linear Quadratic Gaussian) approach for the minimal output vector.** This controller seeks to enhance the system's performance by integrating state estimates and control. The efficacy of this controller can be demonstrated via simulated outcomes.

These measures collectively guarantee the effectiveness of the designed controllers and observers in achieving the desired system performance.

2.1 Observability

Upon building an LQR controller for the system, the subsequent stage is to assess the system's observability with respect to specific output vectors. The metrics employed to determine controllability conditions are essential in this evaluation.

2.1.1 Determine For Which Output Vectors The Linearized System Is Observable.

The project's continuance requires us to:

1. **Identify the optimal Luenberger Observer for each output vector, contingent upon their observability.** This entails selecting suitable observer structures to accurately estimate the system's state variables.
2. **Simulate the system's response to designated input circumstances, including a unit step input, for both the linearized and non-linearized versions of the system.** This phase enables us to evaluate the practical performance of the designed controllers and observers.
3. **Formulate an output feedback controller utilizing the LQG (Linear Quadratic Gaussian) approach for the minimal output vector.** This controller seeks to enhance the system's performance by integrating state estimates and control. The efficacy of this controller can be demonstrated via simulated outcomes.

```
For state x:
Rank for x: 6
Observable for x: Yes

For θ_1 & θ_2:
Rank for θ_1 & θ_2: 4
Observable for θ_1 & θ_2: No

For x & θ_2:
Rank for x & θ_2: 6
Observable for x & θ_2: Yes

For x, θ_1 & θ_2:
Rank for x, θ_1 & θ_2: 6
Observable for x, θ_1 & θ_2: Yes
```

Figure 4: Observability Check

2.2 Obtain Your "Best" Luenberger Observer For Each One Of The Output Vectors

We have constructed a Luenberger observer for the input vectors, thereby guaranteeing the system's observability. The observer system can be articulated as follows:

- **Estimated State:**

$$\dot{\hat{X}}(t) = A\hat{X}(t) + Bu(t) + L\big(y(t) - \hat{y}(t)\big)$$

- **Estimated Output:**

$$\hat{y}(t) = C\hat{X}(t) + Du(t)$$

In this context:

- L denotes the observer gain,
- $\hat{X}(t)$ signifies our estimated state obtained from the output $y(t)$.

To ascertain the suitable L for our observer, we position the poles of $A - LC$ in the negative left-half plane. To improve convergence, we place them at a location roughly three times the distance from the eigenvalues of $A - BK$, where K represents our feedback gain.

This code will simulate both nonlinear and linear systems. It takes into account the initial conditions and applies a step input at $t = 200$ seconds:

2.2.1 For The Linearized System

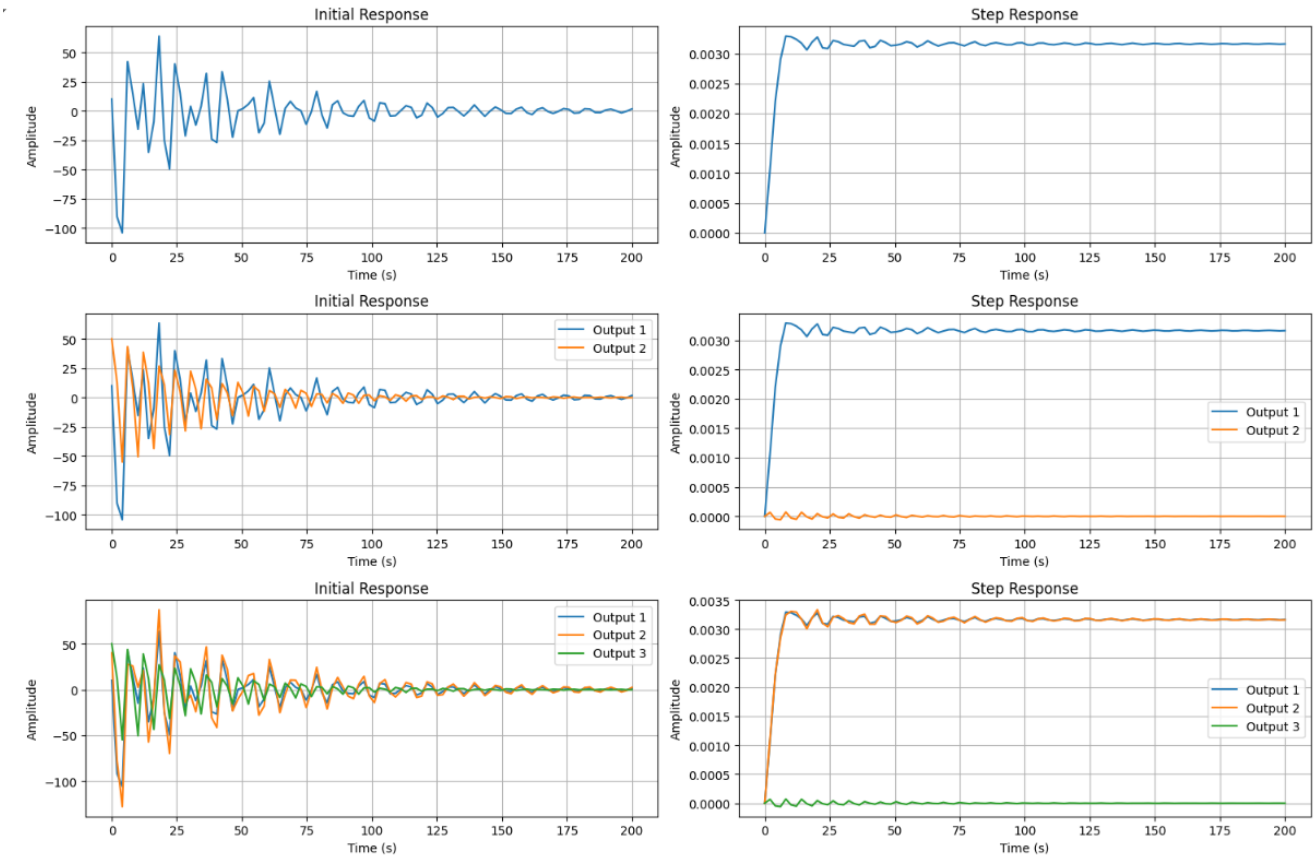


Figure 5: Luenberger Observer For Linear System(a).

2.2.2 For The Original nonlinear system

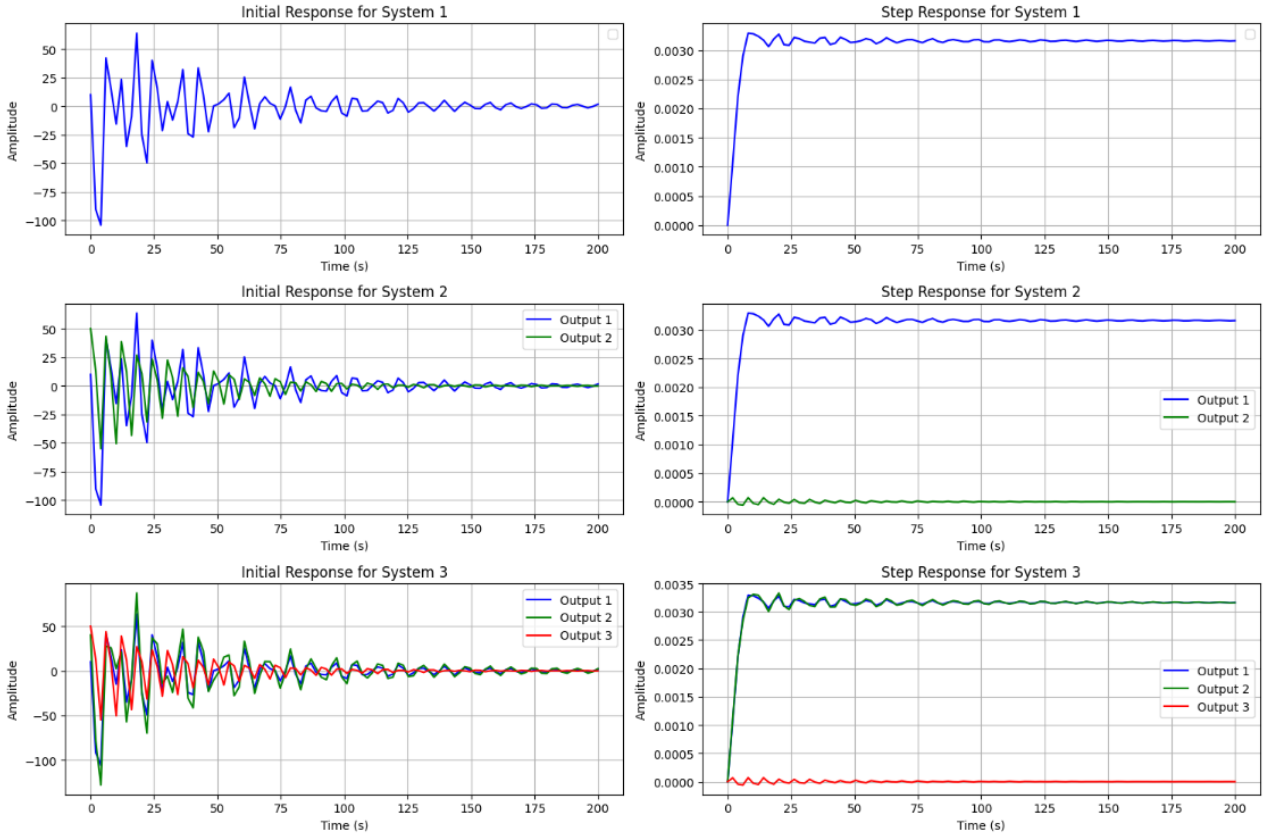


Figure 6: Luenberger Observer For Nonlinear System(b).

2.3 Design An Output Feedback Controller For Your Choice Of The "Smallest" Output Vector - LQG Controller.

Combining a Kalman filter with a linear quadratic regulator (LQR), the LQG (Linear Quadratic Gaussian) controller results. The separation principle holds that one may independently design the state feedback (LQR) and the state estimator (Kalman filter).

The ideal solution follows the conventional output feedback arrangement in which optimal gain matrices K (for control) and L (for estimate) are combined with the Luenberger observer. These gain matrices are generated separately using the Kalman-Bucy filter and the LQR technique, respectively.

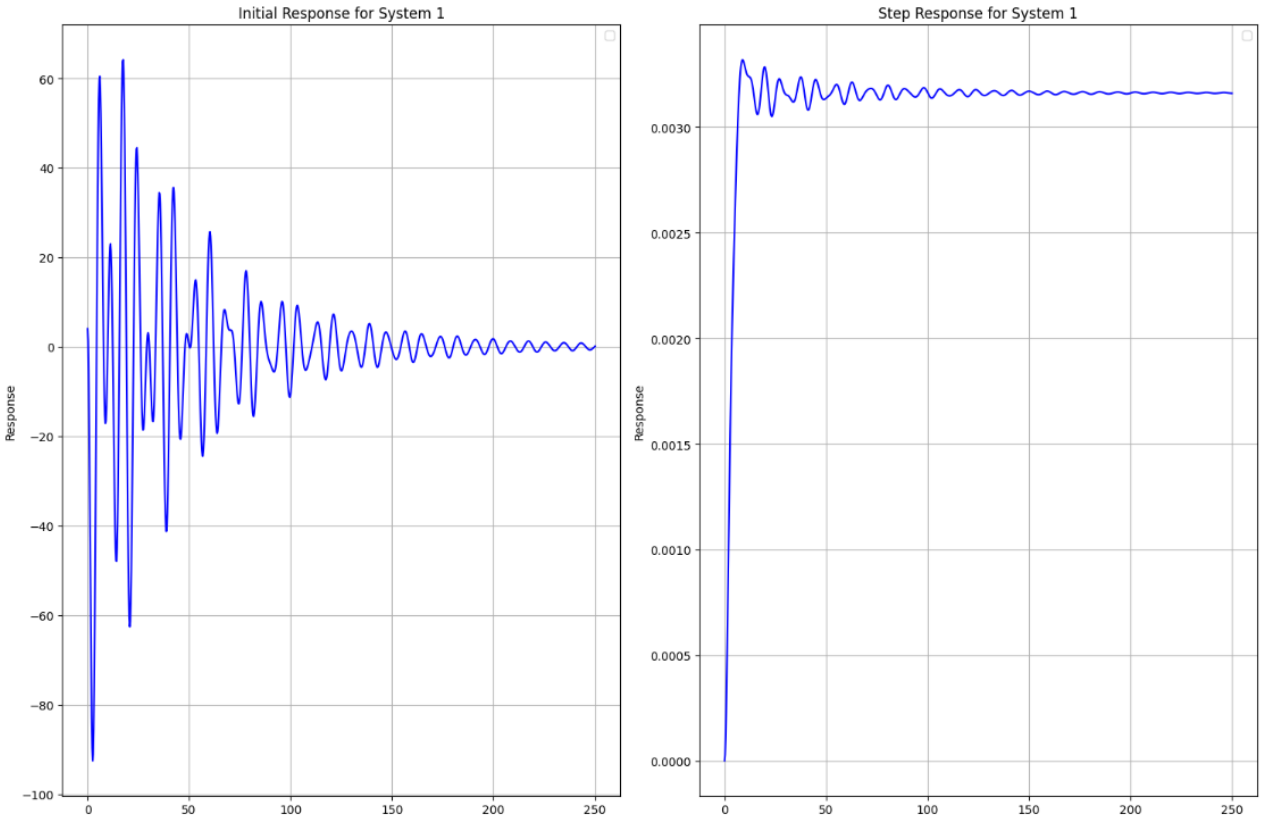


Figure 7: LQG Controller For Non-Linear System

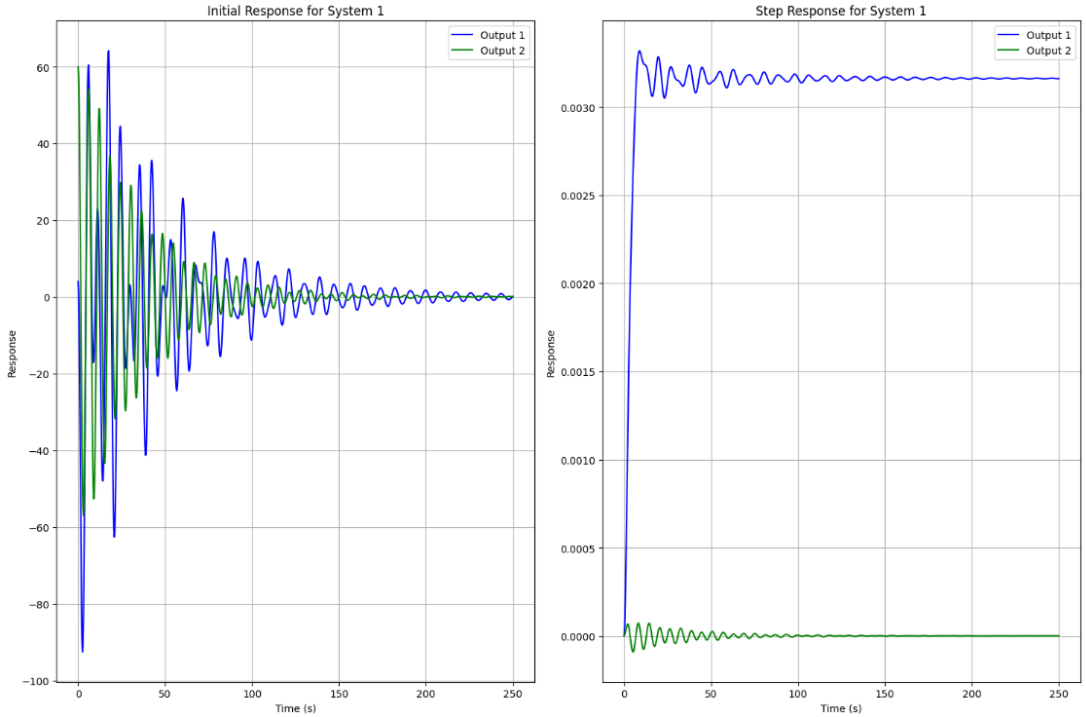


Figure 8: LQG Controller For Non-Linear System

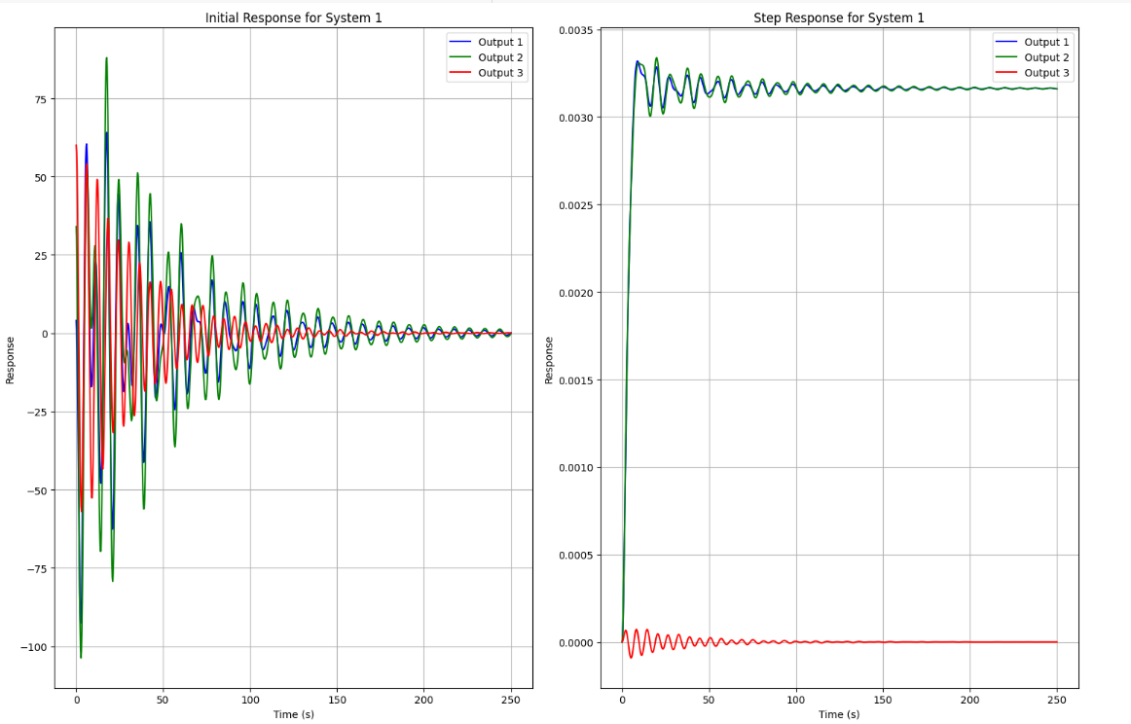


Figure 9: LQG Controller For Non-Linear System

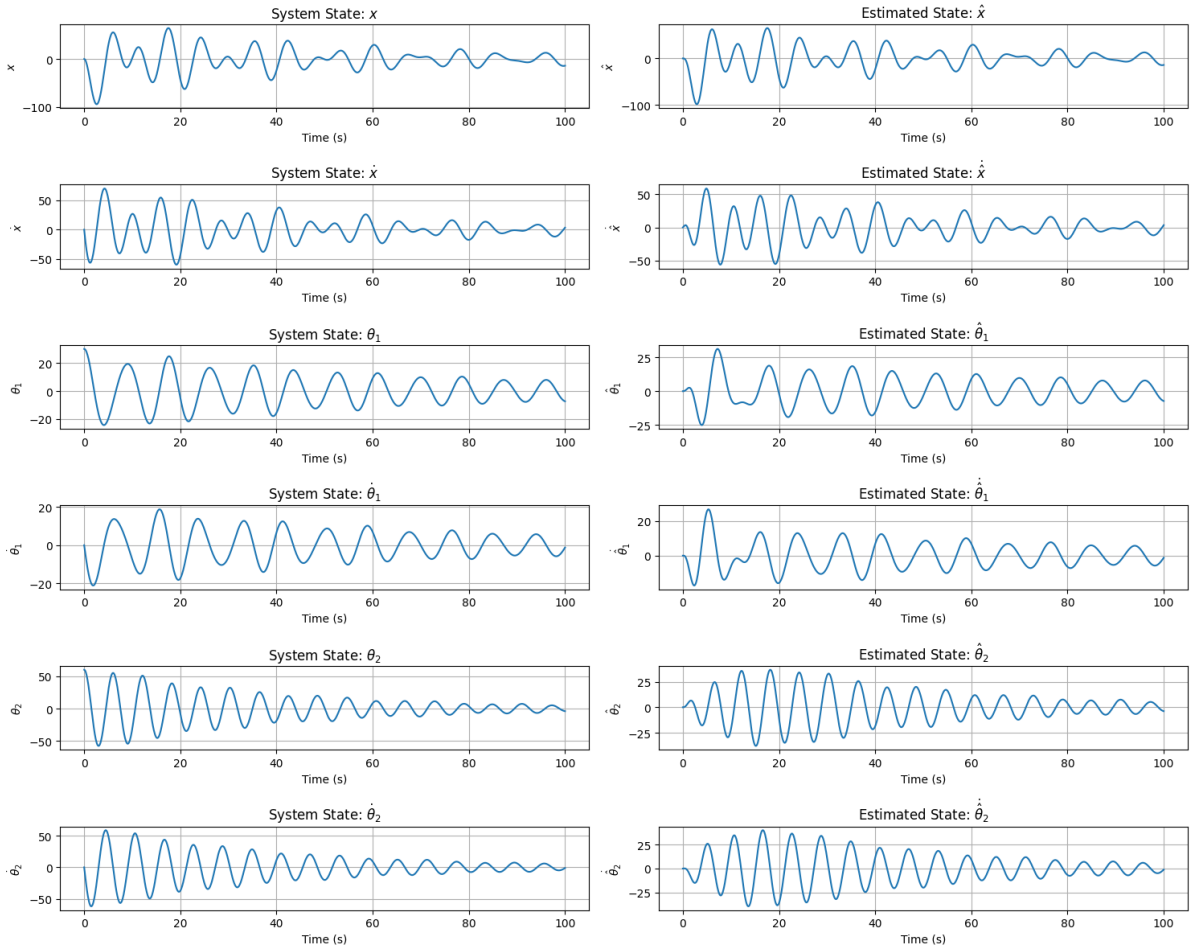


Figure 10: LQG Controller For Non-Linear System

To attain asymptotic tracking of a constant reference, we meticulously adjust the controller so that the variables x and u converge to the specified values as time t progresses. Throughout the reference tracking procedure, the system incurs expenses related to the existing deviations from the reference limitations.

The major aim is to reduce this cost, which can be articulated as:

$$\int_0^\infty \Big((X(t) - X_d)^\top Q (X(t) - X_d) + (U(t) - U_\infty)^\top R (U(t) - U_\infty) \Big)\, dt$$

This expense can be efficiently reduced if a solution U_∞ exists such that:

$$AX_\infty + BU_\infty = 0.$$

The best solution for U_∞ is expressed as:

$$U(t) = K(X(t) - X_d) + U_\infty,$$

where:

$$K = -R^{-1} B^\top P,$$

and P denotes the positive definite matrix solution of the Riccati equation:

$$A^\top P + PA - PBR^{-1} B^\top P = -Q.$$

By adhering to these limitations, we may guide the system to achieve the required output when a constant tracking reference X_{ref} fulfills the equation:

$$X_{\text{ref}} = AX_{\text{ref}} + BU_{\text{ref}}.$$

This design is capable of managing continuous force disturbances exerted on the cart. If these force disturbances adhere to a Gaussian distribution, the controller can efficiently accommodate and adjust to these system disturbances.

References

[1] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, 1999.

[2] João P. Hespanha. *Linear Systems Theory*. Princeton University Press, 2018.

[3] Waseem Malik. enpm 667 control of robotic systems notes, 2024.

[4] Mark W Spong, Seth Hutchinson, and M. (Mathukumalli) Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Inc., Hoboken, NJ, 2nd edition, 2020. Print.

3 Links

- Python Code With Google Colab

4 Appendix

```
"""
Final Project
Authors:

Munyaradzi Antony (UID: 120482731)
Varad Nerlekar (UID: 120501135)

"""
```

```
!pip install control
```

```
# Importing libraries
import sympy as sp
from sympy import simplify, symbols
from pprint import pprint


# Defining the masses, link lengths and gravity as symbols
M, m1, m2, l1, l2, g = sp.symbols('M m1 m2 l1 l2 g')


```

```

# Defining the linearized state-space equation
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]
])

# Printing the A Matrix
A
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{gm_1}{M} & 0 & -\frac{gm_2}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{g(M+m_1)}{Ml_1} & 0 & -\frac{gm_2}{Ml_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{gm_1}{Ml_2} & 0 & -\frac{g(M+m_2)}{Ml_2} & 0 \end{bmatrix}$$

```
B = sp.Matrix([
  0,
  1 / M,
  0,
  1 / (M * l1),
  0,
  1 / (M * l2)
]).reshape(6, 1)

# Printing the B Matrix
B
```

$$\begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml_1} \\ 0 \\ \frac{1}{Ml_2} \end{bmatrix}$$

```

# Controllability Matrix
C = sp.Matrix.hstack(
    B, A * B, A**2 * B, A**3 * B, A**4 * B, A**5 * B
)
C
```

$$\begin{bmatrix} 0 & \frac{1}{M} & 0 & -\frac{gm_2}{Ml_2} - \frac{gm_1}{Ml_1} & 0 & \frac{Mg^2l_1m_2+g^2l_1m_2^2+g^2l_2m_1m_2}{M^2l_1l_2^2} + \frac{Mg^2l_2m_1+g^2l_1m_1m_2+g^2l_2m_1^2}{M^2l_1^2l_2} \\ \frac{1}{M} & 0 & -\frac{gm_2}{Ml_2} - \frac{gm_1}{Ml_1} & 0 & \frac{Mg^2l_1m_2+Mg^2l_2m_2+g^2l_1m_2^2+g^2l_2m_1m_2}{M^2l_1l_2^2} + \frac{M^2g^2l_2+2Mg^2l_2m_1+g^2l_1m_1m_2+g^2l_2m_1^2}{M^2l_1^2l_2} & 0 \\ 0 & \frac{1}{Ml_1} & 0 & -\frac{gm_2}{Ml_1l_2} + \frac{-Mg-gm_1}{Ml_1^2} & 0 & \frac{Mg^2l_1m_2+Mg^2l_2m_2+g^2l_1m_2^2+g^2l_2m_1m_2}{M^2l_1^2l_2^2} + \frac{M^2g^2l_2+2Mg^2l_2m_1+g^2l_1m_1m_2+g^2l_2m_1^2}{M^2l_1^3l_2} \\ \frac{1}{Ml_1} & 0 & -\frac{gm_2}{Ml_1l_2} + \frac{-Mg-gm_1}{Ml_1^2} & 0 & \frac{M^2g^2l_1+2Mg^2l_1m_2+g^2l_1m_2^2+g^2l_2m_1m_2}{M^2l_1^2l_2^2} + \frac{Mg^2l_1m_1+Mg^2l_2m_1+g^2l_1m_1m_2+g^2l_2m_1^2}{M^2l_1^3l_2^2} & 0 \\ 0 & \frac{1}{Ml_2} & 0 & -\frac{gm_1}{Ml_1l_2} + \frac{-Mg-gm_2}{Ml_2^2} & 0 & \frac{M^2g^2l_1+2Mg^2l_1m_2+g^2l_1m_2^2+g^2l_2m_1m_2}{M^2l_1^2l_2^2} + \frac{Mg^2l_1m_1+Mg^2l_2m_1+g^2l_1m_1m_2+g^2l_2m_1^2}{M^2l_1^2l_2^3} \\ \frac{1}{Ml_2} & 0 & -\frac{gm_1}{Ml_1l_2} + \frac{-Mg-gm_2}{Ml_2^2} & 0 & \frac{M^2g^2l_1+2Mg^2l_1m_2+g^2l_1m_2^2+g^2l_2m_1m_2}{M^2l_1^2l_2^2} + \frac{Mg^2l_1m_1+Mg^2l_2m_1+g^2l_1m_1m_2+g^2l_2m_1^2}{M^2l_1^2l_2^3} & 0 \end{bmatrix}$$

```
# Taking the determinant of the Controllability Matrix
det_C = C.det()
det_C
```

$$-\frac{g^6 l_1^2 - 2g^6 l_1 l_2 + g^6 l_2^2}{M^6 l_1^6 l_2^6}$$

```
# Taking the rank of the Controllability Matrix
rank_C = C.rank()
rank_C
```

6

```
# Special case: When pendulum lengths are equal i.e l1 = l2
C_special = C.subs(l1, l2)
C_special
```

$$\begin{bmatrix} 0 & \frac{1}{M} & 0 & -\frac{gm_1}{M^2 l_2} - \frac{gm_2}{M^2 l_2} & 0 & \frac{Mg^2 l_2 m_1 + g^2 l_2 m_1^2 + g^2 l_2 m_1 m_2}{M^3 l_2^3} + \frac{Mg^2 l_2 m_2 + g^2 l_2 m_1 m_2 + g^2 l_2 m_2^2}{M^3 l_2^3} \\ \frac{1}{M} & 0 & -\frac{gm_1}{M^2 l_2} - \frac{gm_2}{M^2 l_2} & 0 & \frac{Mg^2 l_2 m_1 + g^2 l_2 m_1^2 + g^2 l_2 m_1 m_2}{M^3 l_2^3} + \frac{Mg^2 l_2 m_2 + g^2 l_2 m_1 m_2 + g^2 l_2 m_2^2}{M^3 l_2^3} & 0 \\ 0 & \frac{1}{Ml_2} & 0 & -\frac{gm_2}{M^2 l_2^2} + \frac{-Mg - gm_1}{M^2 l_2^2} & 0 & \frac{2Mg^2 l_2 m_2 + g^2 l_2 m_1 m_2 + g^2 l_2 m_2^2}{M^3 l_2^4} + \frac{M^2 g^2 l_2 + 2Mg^2 l_2 m_1 + g^2 l_2 m_1^2 + g^2 l_2 m_1 m_2}{M^3 l_2^4} \\ \frac{1}{Ml_2} & 0 & -\frac{gm_2}{M^2 l_2^2} + \frac{-Mg - gm_1}{M^2 l_2^2} & 0 & \frac{2Mg^2 l_2 m_2 + g^2 l_2 m_1 m_2 + g^2 l_2 m_2^2}{M^3 l_2^4} + \frac{M^2 g^2 l_2 + 2Mg^2 l_2 m_1 + g^2 l_2 m_1^2 + g^2 l_2 m_1 m_2}{M^3 l_2^4} & 0 \\ 0 & \frac{1}{Ml_2} & 0 & -\frac{gm_1}{M^2 l_2^2} + \frac{-Mg - gm_2}{M^2 l_2^2} & 0 & \frac{2Mg^2 l_2 m_1 + g^2 l_2 m_1^2 + g^2 l_2 m_1 m_2}{M^3 l_2^4} + \frac{M^2 g^2 l_2 + 2Mg^2 l_2 m_2 + g^2 l_2 m_1 m_2 + g^2 l_2 m_2^2}{M^3 l_2^4} \\ \frac{1}{Ml_2} & 0 & -\frac{gm_1}{M^2 l_2^2} + \frac{-Mg - gm_2}{M^2 l_2^2} & 0 & \frac{2Mg^2 l_2 m_1 + g^2 l_2 m_1^2 + g^2 l_2 m_1 m_2}{M^3 l_2^4} + \frac{M^2 g^2 l_2 + 2Mg^2 l_2 m_2 + g^2 l_2 m_1 m_2 + g^2 l_2 m_2^2}{M^3 l_2^4} & 0 \end{bmatrix}$$

```
# Taking the rank of the Controllability Matrix for the Special case
rank_C_special = C_special.rank()
rank_C_special
```

4

```
# The Controllability Condition for the system
if det_C == 0:
  print("The system is not controllable as it is not full rank.")
else:
  # print("The system may be controllable if is controllable for all conditions.")
  if rank_C == rank_C_special:
      print("The system is controllable as the ranks are equal for all conditions.")
  else:
      print("The system is not controllable as the ranks are not equal for all conditions.")


```

```
# Importing Libraries
import scipy
import numpy as np
import matplotlib.pyplot as plt

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


```

```
# Defining the linearized state-space equation
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]
])

# Printing the A Matrix
simplify(A)


```

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.981 & 0.0 & -0.981 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.53955 & 0.0 & -0.04905 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & -0.0981 & 0.0 & -1.0791 & 0.0 \end{bmatrix}$$

```
B = np.array([
  0,
  1 / M,
  0,
  1 / (M * l1),
  0,
  1 / (M * l2)
]).reshape(6, 1)

# Printing the B Matrix
simplify(B)


```

$$\begin{bmatrix} 0.0 \\ 0.001 \\ 0.0 \\ 5.0 \cdot 10^{-5} \\ 0.0 \\ 0.0001 \end{bmatrix}$$

```

# Controllability Matrix
C = np.column_stack([
    B, A @ B, A@A @ B, A@A@A @ B, A@A@A@A @ B, A@A@A@A@A @ B
])

```

```

# Taking the rank of the Controllability Matrix to check its controllability
rank_C = np.linalg.matrix_rank(C)
if rank_C == C.shape[0]:
  print("The system is controllable.")
else:
  print("The system is not controllable.")

```

```
# Setting the initial conditions
x_initial = np.array([0, 0, 10, 0, 20, 0])
Q = np.diag([100, 100, 100, 100, 100, 100])
R = np.array([[0.001]])


```

```

# Implementing the LQR Controller (scipy.linalg.solve_continuous_are) using the Ricatti Equation
P = scipy.linalg.solve_continuous_are(A, B, Q, R)
simplify(P)

```

$$\begin{bmatrix} 293.104392177231 & 379.55092356792 & -216.133955681951 & -605.569584766415 & -105.703878321283 & -330.446783127645 \\ 379.55092356792 & 1145.28754618159 & -27.928532323754 & -2215.59112338585 & 20.6240730662794 & -1076.30518533009 \\ -216.133955681951 & -27.928532323754 & 13614.739651211 & 183.569433993754 & 72.689320596692 & -229.526566948364 \\ -605.569584766415 & -2215.59112338585 & 183.569433993754 & 29065.7011334585 & 457.988540473508 & 788.304870562619 \\ -105.703878321283 & 20.6240730662794 & 72.689320596692 & 457.988540473508 & 6550.09209215496 & 5.86654946079844 \\ -330.446783127645 & -1076.30518533009 & -229.526566948364 & 788.304870562619 & 5.86654946079844 & 7026.24928793402 \end{bmatrix}$$

```
# Computing the Feedback Gain Matrix
K = np.dot(np.linalg.inv(R), np.dot(B.T, P))
simplify(K)


```

```
[[316.227766016835  926.877471479285  −41.7027173189027  −683.475579656664  44.1101550360346  −334.265013008559]]
```

```
# Setting the output Matrices C & D
C = np.eye(6)
D = np.zeros((6, 1))

# Setting Input Signal U
t = np.linspace(0, 200, 1000)
u = np.zeros_like(t)
```

```
# System Response without the controller
system_without_controller = scipy.signal.StateSpace(A, B, C, D)
_, y_without_controller, x_without_controller = scipy.signal.lsim(system_without_controller, u, t, x_initial)
```

```
# System with LQR Controller
A_closed_loop = A - np.dot(B, K)
simplify(A_closed_loop)
```

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -0.316227766016835 & -0.926877471479285 & -0.939297282681097 & 0.683475579656664 & -1.02511015503603 & 0.334265013008559 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ -0.0158113883008418 & -0.0463438735739642 & -0.537464864134055 & 0.0341737789828332 & -0.0512555077518017 & 0.0167132506504279 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ -0.0316227766016835 & -0.0926877471479285 & -0.0939297282681097 & 0.0683475579656664 & -1.0835110155036 & 0.0334265013008559 \end{bmatrix}$$

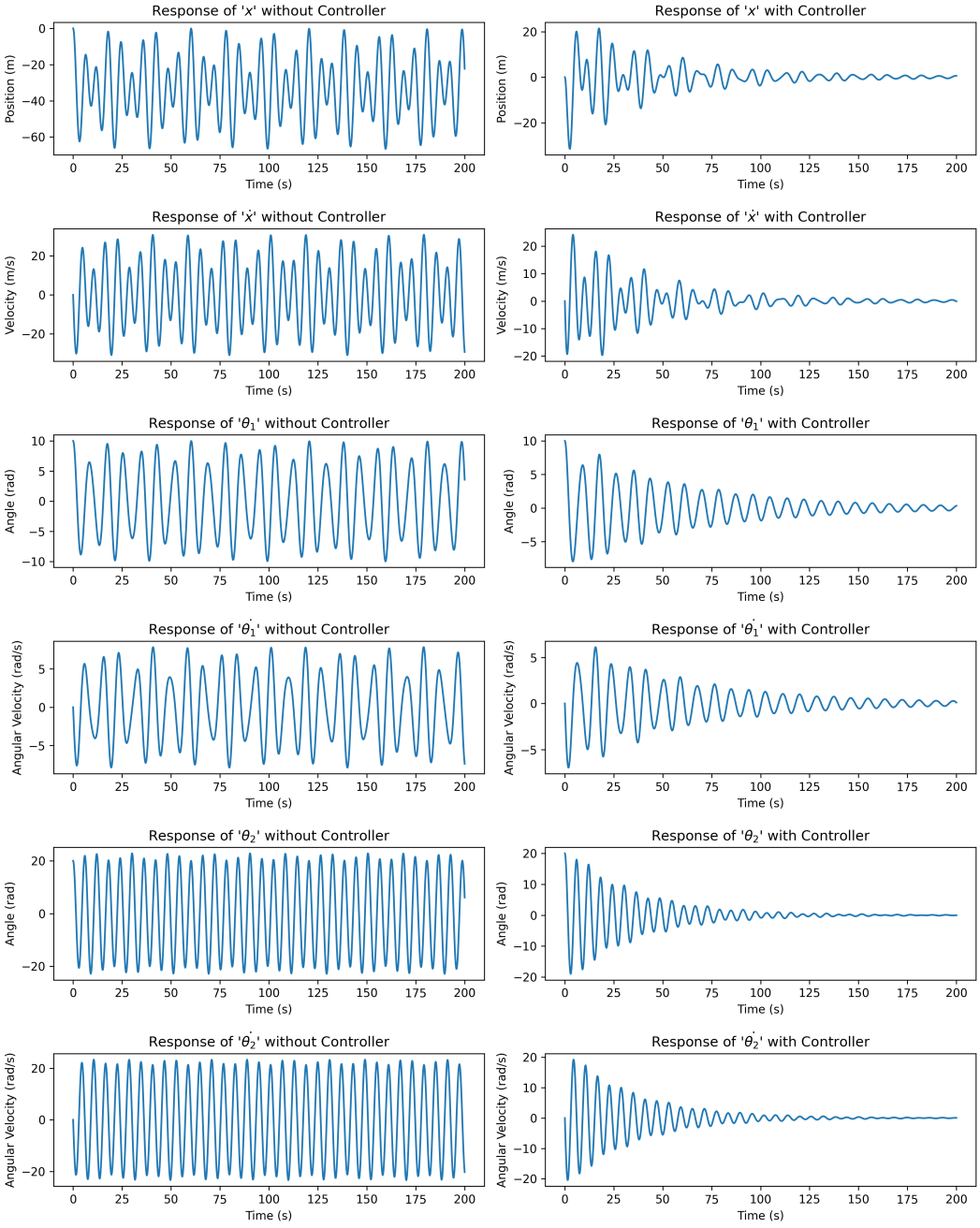
```
# Getting the Eigenvalues of the Closed Loop System
eigenvalues_closed_loop = np.linalg.eigvals(A_closed_loop)
simplify(eigenvalues_closed_loop)
```

$$\left[-0.383784665947216 + 0.354266878997317i \quad -0.383784665947216 - 0.354266878997317i \quad -0.0297862291376132 + 1.03473463598333i \quad -0.0297862291376132 - 1.03473463598333i \quad -0.0160677005129693 + 0.721304562445142i \quad -0.0160677005129693 - 0.721304562445142i\right]$$

```
# System Response with the controller
system_with_controller = scipy.signal.StateSpace(A_closed_loop, B, C, D)
_, y_with_controller, x_with_controller = scipy.signal.lsim(system_with_controller, u, t, x_initial)
```

```

# Plotting the responses
fig, ax = plt.subplots(6, 2, figsize=(12, 15))

# Plot position x without controller
ax[0, 0].plot(t, x_without_controller[:, 0])
ax[0, 0].set_title(r"Response of '$x$' without Controller")
ax[0, 0].set_xlabel('Time (s)')
ax[0, 0].set_ylabel('Position (m)')

# Plot position x with controller
ax[0, 1].plot(t, x_with_controller[:, 0])
ax[0, 1].set_title(r"Response of '$x$' with Controller")
ax[0, 1].set_xlabel('Time (s)')
ax[0, 1].set_ylabel('Position (m)')

# Plot velocity dx/dt without controller
ax[1, 0].plot(t, x_without_controller[:, 1])
ax[1, 0].set_title(r"Response of '$\dot{x}$' without Controller")
ax[1, 0].set_xlabel('Time (s)')
ax[1, 0].set_ylabel('Velocity (m/s)')

# Plot velocity dx/dt with controller
ax[1, 1].plot(t, x_with_controller[:, 1])
ax[1, 1].set_title(r"Response of '$\dot{x}$' with Controller")
ax[1, 1].set_xlabel('Time (s)')
ax[1, 1].set_ylabel('Velocity (m/s)')

# Plot the first pendulum angle (without and with controller)
ax[2, 0].plot(t, x_without_controller[:, 2])
ax[2, 0].set_title(r"Response of '$\theta_1$' without Controller")
ax[2, 0].set_xlabel('Time (s)')
ax[2, 0].set_ylabel('Angle (rad)')

ax[2, 1].plot(t, x_with_controller[:, 2])
ax[2, 1].set_title(r"Response of '$\theta_1$' with Controller")
ax[2, 1].set_xlabel('Time (s)')
ax[2, 1].set_ylabel('Angle (rad)')

# Plot the first pendulum angular velocity (without and with controller)
ax[3, 0].plot(t, x_without_controller[:, 3])
ax[3, 0].set_title(r"Response of '$\dot{\theta_1}$' without Controller")
ax[3, 0].set_xlabel('Time (s)')
ax[3, 0].set_ylabel('Angular Velocity (rad/s)')

ax[3, 1].plot(t, x_with_controller[:, 3])
ax[3, 1].set_title(r"Response of '$\dot{\theta_1}$' with Controller")
ax[3, 1].set_xlabel('Time (s)')
ax[3, 1].set_ylabel('Angular Velocity (rad/s)')

# Plot the second pendulum angle (without and with controller)
ax[4, 0].plot(t, x_without_controller[:, 4])
ax[4, 0].set_title(r"Response of '$\theta_2$' without Controller")
ax[4, 0].set_xlabel('Time (s)')
ax[4, 0].set_ylabel('Angle (rad)')

ax[4, 1].plot(t, x_with_controller[:, 4])
ax[4, 1].set_title(r"Response of '$\theta_2$' with Controller")
ax[4, 1].set_xlabel('Time (s)')
ax[4, 1].set_ylabel('Angle (rad)')

# Plot the second pendulum angular velocity (without and with controller)
ax[5, 0].plot(t, x_without_controller[:, 5])
ax[5, 0].set_title(r"Response of '$\dot{\theta_2}$' without Controller")
ax[5, 0].set_xlabel('Time (s)')
ax[5, 0].set_ylabel('Angular Velocity (rad/s)')

ax[5, 1].plot(t, x_with_controller[:, 5])
```

```
ax[5, 1].set_title(r"Response of '$\dot{\theta_2}$' with Controller")
ax[5, 1].set_xlabel('Time (s)')
ax[5, 1].set_ylabel('Angular Velocity (rad/s)')

# Adjust layout to avoid overlapping
plt.tight_layout()

# Save each plot as a PNG image
fig.savefig('response_plots - Linear system.png', dpi=300)

# Show the plot
plt.show()
```



```python
# Importing Libraries
import scipy
import numpy as np
import matplotlib.pyplot as plt

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


```

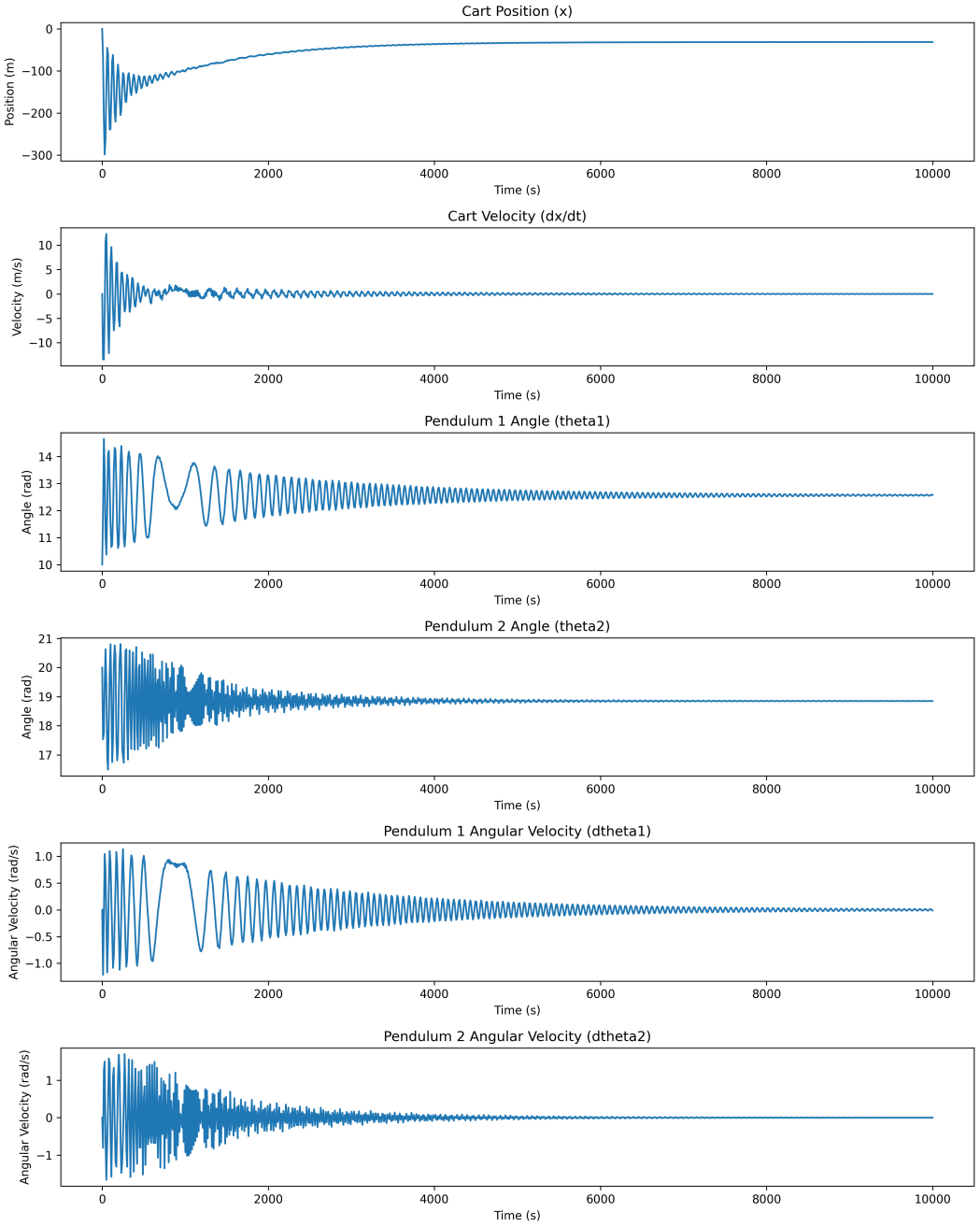
```python

# Define the nonlinear system (state = [x, dx, theta1, dtheta1, theta2, dtheta2])
def nonlinear_system(t, y):
    # Control input (LQR feedback law)
    F = -np.dot(K_val, y)

    # Initialize the dydt vector (6 state variables)
    dydt = np.zeros(6)

    # Cart position and velocity dynamics (x, dx)
    dydt[0] = y[1]  # dx/dt = velocity of the cart
    dydt[1] = (F - (g/2) * (m1*np.sin(2*y[2]) + m2*np.sin(2*y[4])) - (m1*l1*y[3]**2)) / (M + m1)

    # Pendulum 1 angle and angular velocity dynamics (theta1, dtheta1)
    dydt[2] = y[3]  # dtheta1/dt = angular velocity of the first pendulum
    dydt[3] = (dydt[1] * np.cos(y[2]) - g * np.sin(y[2])) / l1  # d^2theta1/dt^2

    # Pendulum 2 angle and angular velocity dynamics (theta2, dtheta2)
    dydt[4] = y[5]  # dtheta2/dt = angular velocity of the second pendulum
    dydt[5] = (dydt[1] * np.cos(y[4]) - g * np.sin(y[4])) / l2  # d^2theta2/dt^2

    return dydt

```

```python

# Setting the initial conditions
y_initial = np.array([0, 0, 10, 0, 20, 0])
t = np.linspace(0, 10000, 1000)
u = 0

# Control Gain Matrix (K_val)
K_val = np.array([10, 10, 10, 10, 10, 10])

```

```python

# Solve the system using ODE solver
solution = scipy.integrate.solve_ivp(nonlinear_system, (0, 10000), y_initial, t_eval=t).y

# Extract the state variables
x = solution[0]
dx = solution[1]
theta1 = solution[2]
dtheta1 = solution[3]
```

```
theta2 = solution[4]
dtheta2 = solution[5]

```

```

# Plotting the responses
fig, ax = plt.subplots(6, 1, figsize=(12, 15))

# Plot position x
plt.subplot(6, 1, 1)
plt.plot(t, x)
plt.title('Cart Position (x)')
plt.xlabel('Time (s)')
plt.ylabel('Position (m)')

# Plot velocity dx/dt
plt.subplot(6, 1, 2)
plt.plot(t, dx)
plt.title('Cart Velocity (dx/dt)')
plt.xlabel('Time (s)')
plt.ylabel('Velocity (m/s)')

# Plot pendulum 1 angle (theta1)
plt.subplot(6, 1, 3)
plt.plot(t, theta1)
plt.title('Pendulum 1 Angle (theta1)')
plt.xlabel('Time (s)')
plt.ylabel('Angle (rad)')

# Plot pendulum 2 angle (theta2)
plt.subplot(6, 1, 4)
plt.plot(t, theta2)
plt.title('Pendulum 2 Angle (theta2)')
plt.xlabel('Time (s)')
plt.ylabel('Angle (rad)')

# Plot pendulum 1 angular velocity (dtheta1)
plt.subplot(6, 1, 5)
plt.plot(t, dtheta1)
plt.title('Pendulum 1 Angular Velocity (dtheta1)')
plt.xlabel('Time (s)')
plt.ylabel('Angular Velocity (rad/s)')

# Plot pendulum 2 angular velocity (dtheta2)
plt.subplot(6, 1, 6)
plt.plot(t, dtheta2)
plt.title('Pendulum 2 Angular Velocity (dtheta2)')
plt.xlabel('Time (s)')
plt.ylabel('Angular Velocity (rad/s)')

# Adjust layout to avoid overlapping
plt.tight_layout()

# Save each plot as a PNG image
fig.savefig('response_plots - Nonlinear system.png', dpi=300)

# Show the plot
plt.show()


```




```python
# Importing Libraries
import numpy as np

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

```

```
# Observability matrix for a set of C matrices
def observability_matrix(A, C):
  obs_matrix = C
  for i in range(1, A.shape[0]):
    obs_matrix = np.vstack((obs_matrix, C @ np.linalg.matrix_power(A, i)))
  return obs_matrix

```

```
# Defining the linearized state-space equation
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]
])

# Printing the A Matrix
simplify(A)

```

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.981 & 0.0 & -0.981 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.53955 & 0.0 & -0.04905 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & -0.0981 & 0.0 & -1.0791 & 0.0 \end{bmatrix}$$

```
B = np.array([
  0,
  1 / M,
  0,
  1 / (M * l1),
  0,
  1 / (M * l2)
]).reshape(6, 1)

# Printing the B Matrix
simplify(B)

```

$$\begin{bmatrix} 0.0 \\ 0.001 \\ 0.0 \\ 5.0 \cdot 10^{-5} \\ 0.0 \\ 0.0001 \end{bmatrix}$$

```

# Defining the C Matrices
C_x = np.array([[1, 0, 0, 0, 0, 0]])
C_theta1 = np.array([[0, 0, 1, 0, 0, 0]])
C_theta2 = np.array([[0, 0, 0, 0, 1, 0]])
```

```

```

```
# Check observability for x
C_matrix = C_x
observability = observability_matrix(A, C_matrix)
rank = np.linalg.matrix_rank(observability)
observable = "Yes" if rank == A.shape[0] else "No"
print("For state x:")
print("Rank for x:", rank)
print("Observable for x:", observable)

# Check observability for _1 and _2
C_matrix = np.vstack([C_theta1, C_theta2])
observability = observability_matrix(A, C_matrix)
rank = np.linalg.matrix_rank(observability)
observable = "Yes" if rank == A.shape[0] else "No"
print("\nFor _1 & _2:")
print("Rank for _1 & _2:", rank)
print("Observable for _1 & _2:", observable)

# Check observability for x and _2
C_matrix = np.vstack([C_x, C_theta2])
observability = observability_matrix(A, C_matrix)
rank = np.linalg.matrix_rank(observability)
observable = "Yes" if rank == A.shape[0] else "No"
print("\nFor x & _2:")
print("Rank for x & _2:", rank)
print("Observable for x & _2:", observable)

# Check observability for x, _1 and _2
C_matrix = np.vstack([C_x, C_theta1, C_theta2])
observability = observability_matrix(A, C_matrix)
rank = np.linalg.matrix_rank(observability)
observable = "Yes" if rank == A.shape[0] else "No"
print("\nFor x, _1 & _2:")
print("Rank for x, _1 & _2:", rank)
print("Observable for x, _1 & _2:", observable)


```

```
For state x:
Rank for x: 6
Observable for x: Yes

For θ_1 & θ_2:
Rank for θ_1 & θ_2: 4
Observable for θ_1 & θ_2: No

For x & θ_2:
Rank for x & θ_2: 6
Observable for x & θ_2: Yes

For x, θ_1 & θ_2:
Rank for x, θ_1 & θ_2: 6
Observable for x, θ_1 & θ_2: Yes
```

```
# Importing Libraries
import control as ctrl
import numpy as np
import matplotlib.pyplot as plt

# Defining the system parameters
M = 1000  # (Kg): Mass of the cart
m1 = 100  # (Kg): Mass of Pendulum 1
```

```
m2 = 100  # (Kg): Mass of Pendulum 2
l1 = 20   # (m): Length of link 1
l2 = 10   # (m): Length of Link 2
g = 9.81  # (m/s^2): Acceleration due to gravity

```

```
# Defining state space matrices
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)
]).reshape(6, 1)

D = np.zeros((1, 1))

```

```
# Setting LQR parameters
Q = np.diag([1000, 1000, 1000, 1000, 1000, 1000])
R = np.array([[0.01]])
K, S, eigen_values = ctrl.lqr(A, B, Q, R)

```

```
# Setting initial conditions for the observer
# [x1, x1_dot, x2, x2_dot, theta1, theta1_dot, x_observer, x_dot_observer, observer_angle_error, observer_angular_velocity_error, er
initial_conditions = [10, 10, 30, 10, 50, 45, 10, 0, 0, 0, 0, 0]

```

```

# Define observer output matrices and corresponding L matrices
C_matrices = [np.array([[1, 0, 0, 0, 0, 0]]),   # Observing x component
              np.array([[1, 0, 0, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0]]),   # Observing x and theta2
              np.array([[1, 0, 0, 0, 0, 0],     # Observing x, theta1 and theta2
                        [1, 0, 1, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0]])]

poles = -np.linspace(1, 6, 6)
L_matrices = [ctrl.place(A.T, C.T, poles).T for C in C_matrices]
```

```
# Construct and analyze each system
def create_augmented_system(A, B, K, L, C):
  """Create the augmented system (observer design)."""
  A_q = np.block([[A - B @ K, B @ K], [np.zeros(A.shape), A - L @ C]])
  B_q = np.block([[B], [np.zeros(B.shape)]])
```

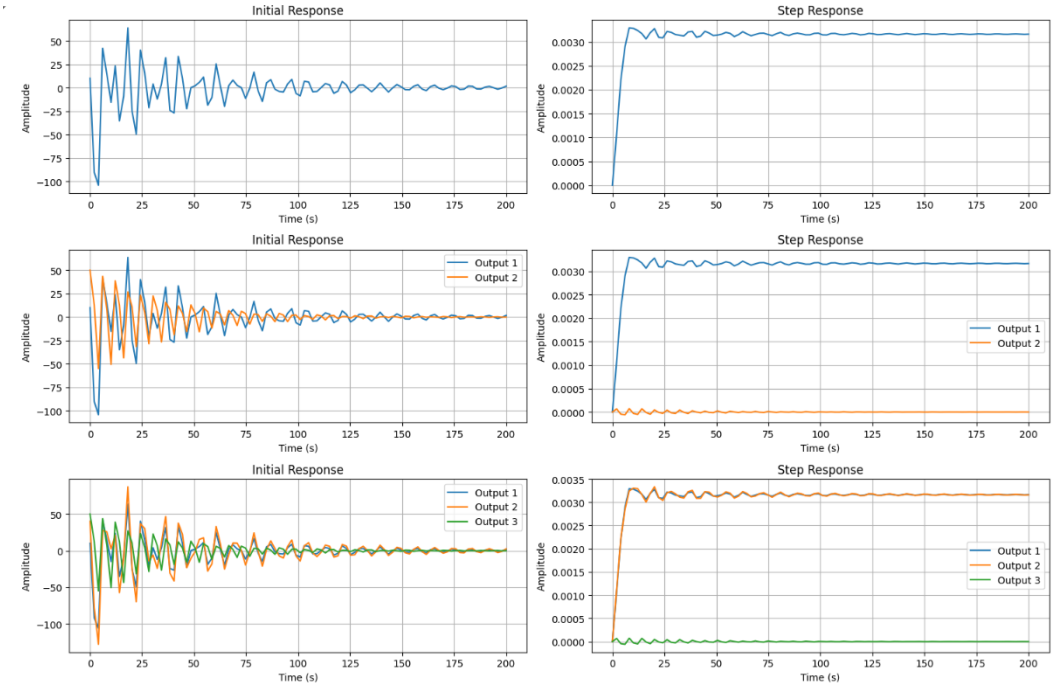
```
  C_q = np.block([C, np.zeros(C.shape)])
  D_q = np.zeros((C.shape[0], B.shape[1]))
  sys = ctrl.ss(A_q, B_q, C_q, D_q)
  return sys

```

```

def plot_initial_response(sys, initial_conditions, plot_index):
  """Plot the initial response for a system."""
  T, yout = ctrl.initial_response(sys, T=np.linspace(0, 200, 100), X0=initial_conditions)
  yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output

  plt.subplot(len(C_matrices), 2, plot_index)
  if yout.ndim == 1:
      plt.plot(T, yout)  # Default color
  else:
      for j in range(yout.shape[0]):
          plt.plot(T, yout[j, :], label=f'Output {j+1}')
      plt.legend()
  plt.title(f'Initial Response')
  plt.xlabel('Time (s)')
  plt.ylabel('Amplitude')
  plt.grid(True)
```

```

def plot_step_response(sys, plot_index):
  """Plot the step response for a system."""
  T, yout = ctrl.step_response(sys, T=np.linspace(0, 200, 100))
  yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output

  plt.subplot(len(C_matrices), 2, plot_index)
  if yout.ndim == 1:
      plt.plot(T, yout)  # Default color
  else:
      for j in range(yout.shape[0]):
          plt.plot(T, yout[j, :], label=f'Output {j+1}')
      plt.legend()
  plt.title(f'Step Response')
  plt.xlabel('Time (s)')
  plt.ylabel('Amplitude')
  plt.grid(True)
```

```
def plot_step_response(sys, plot_index):
  """Plot the step response for a system."""
  T, yout = ctrl.step_response(sys, T=np.linspace(0, 200, 100))
  yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output

  plt.subplot(len(C_matrices), 2, plot_index)
  if yout.ndim == 1:
      plt.plot(T, yout)  # Default color
  else:
      for j in range(yout.shape[0]):
          plt.plot(T, yout[j, :], label=f'Output {j+1}')
      plt.legend()
  plt.title(f'Step Response')
  plt.xlabel('Time (s)')
  plt.ylabel('Amplitude')
  plt.grid(True)

```

```

def plot_responses(A, B, K, L_matrices, C_matrices, initial_conditions):
  """Generate and plot the responses of the system."""
  plt.figure(figsize=(15, 10))

  for i, (C, L) in enumerate(zip(C_matrices, L_matrices)):
    sys = create_augmented_system(A, B, K, L, C)

    # Plot initial response of the system
    plot_initial_response(sys, initial_conditions, 2*i + 1)

    # Plot step response of the system
    plot_step_response(sys, 2*i + 2)

  plt.tight_layout()
  plt.show()

# Example usage
plot_responses(A, B, K, L_matrices, C_matrices, initial_conditions)
```




```
# Importing Libraries
import control as ctrl
import numpy as np
import matplotlib.pyplot as plt

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

```

```
# Defining state space matrices
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)
]).reshape(6, 1)

D = np.zeros((1, 1))

```

```

# Setting LQR parameters
Q = np.diag([1000, 1000, 1000, 1000, 1000, 1000])
R = np.array([[0.01]])
K, S, eigen_values = ctrl.lqr(A, B, Q, R)
```

```

# Setting initial conditions for the observer
# [x1, x1_dot, x2, x2_dot, theta1, theta1_dot, x_observer, x_dot_observer, observer_angle_error, observer_angular_velocity_error, er
initial_conditions = [10, 10, 30, 10, 50, 45, 10, 0, 0, 0, 0, 0]
```

```

# Define observer output matrices and corresponding L matrices
C_matrices = [np.array([[1, 0, 0, 0, 0, 0]]),   # Observing x component
              np.array([[1, 0, 0, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0]]),   # Observing x and theta2
              np.array([[1, 0, 0, 0, 0, 0],     # Observing x, theta1 and theta2
                        [1, 0, 1, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0]])]

poles = -np.linspace(1, 6, 6)
L_matrices = [ctrl.place(A.T, C.T, poles).T for C in C_matrices]
```

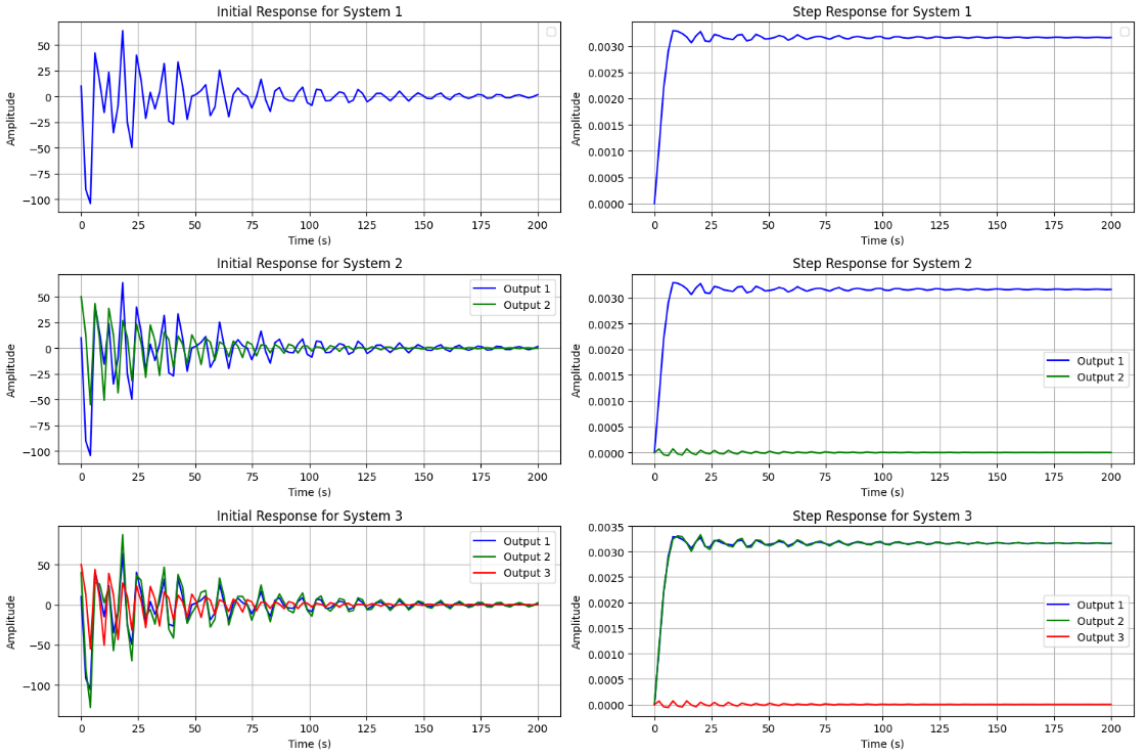
```

# Function to create the system for each observer configuration
def create_system(A, B, C, poles, K, L):
    """
    Create the system with the given observer configuration.

    Parameters:
    - A, B, C: System matrices
    - poles: Desired poles for the controller
    - K: State feedback gains
    - L: Observer gains

    Returns:
```

```
    - sys: The state-space system object
    """
    # Construct the augmented system with state feedback and observer
    A_q = np.block([[A - B @ K, B @ K], [np.zeros_like(A), A - L @ C]])
    B_q = np.block([[B], [np.zeros(B.shape)]])
    C_q = np.block([C, np.zeros(C.shape)])
    D = np.zeros((C.shape[0], B.shape[1]))

    # Create the state-space system
    sys = ctrl.ss(A_q, B_q, C_q, D)
    return sys
```

```
# Function to plot the initial response
def plot_initial_response(sys, i):
    """
    Plot the initial response of the system.

    Parameters:
    - sys: The state-space system
    - i: Index for the current system to determine the color
    """
    T, yout = ctrl.initial_response(sys, T=np.linspace(0, 200, 100), X0=initial_conditions)
    yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output

    # Assign colors within the function
    colors = ['b', 'g', 'r']
    plt.subplot(len(C_matrices), 2, 2 * i + 1)
    if yout.ndim == 1:
        plt.plot(T, yout, color=colors[i])
    else:
        for j in range(yout.shape[0]):
            plt.plot(T, yout[j, :], label=f'Output {j+1}', color=colors[j])
    plt.legend()
    plt.title(f'Initial Response for System {i+1}')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)

```

```
# Function to plot the step response
def plot_step_response(sys, i):
    """
    Plot the step response of the system.

    Parameters:
    - sys: The state-space system
    - i: Index for the current system to determine the color
    """
    T, yout = ctrl.step_response(sys, T=np.linspace(0, 200, 100))
    yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output

    # Assign colors within the function
    colors = ['b', 'g', 'r']
    plt.subplot(len(C_matrices), 2, 2 * i + 2)
    if yout.ndim == 1:
        plt.plot(T, yout, color=colors[i])
    else:
        for j in range(yout.shape[0]):
            plt.plot(T, yout[j, :], label=f'Output {j+1}', color=colors[j])
    plt.legend()
    plt.title(f'Step Response for System {i+1}')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)

```

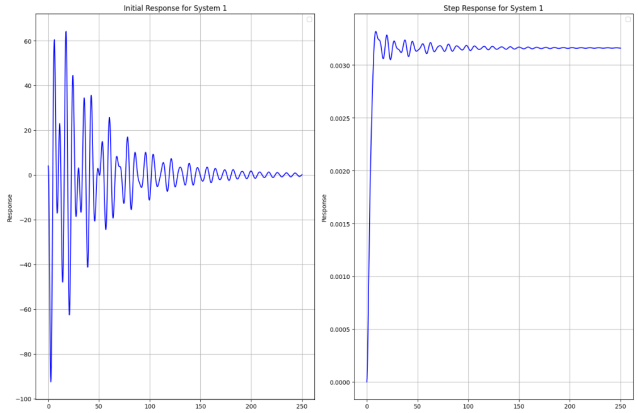
```python

# Main function to generate the plots
def generate_plots(A, B, C_matrices, poles, K, L, initial_cond):
    """
    Generate initial and step response plots for each system configuration.

    Parameters:
    - A, B: System matrices
    - C_matrices: List of C matrices for observer configurations
    - poles: Desired poles for the controller
    - K: State feedback gains
    - L: Observer gains
    - initial_cond: Initial condition for the initial response
    """
    plt.figure(figsize=(15, 10))  # Create a figure for the plots

    for i, C in enumerate(C_matrices):
        # Create system for the current observer configuration
        sys = create_system(A, B, C, poles, K, L_matrices[i])

        # Plot initial response
        plot_initial_response(sys, i)

        # Plot step response
        plot_step_response(sys, i)

    # Finalize the layout and display the plots
    plt.tight_layout()
    plt.show()

# Example of how to use the functions
generate_plots(A, B, C_matrices, poles, K, L_matrices, initial_conditions)
```

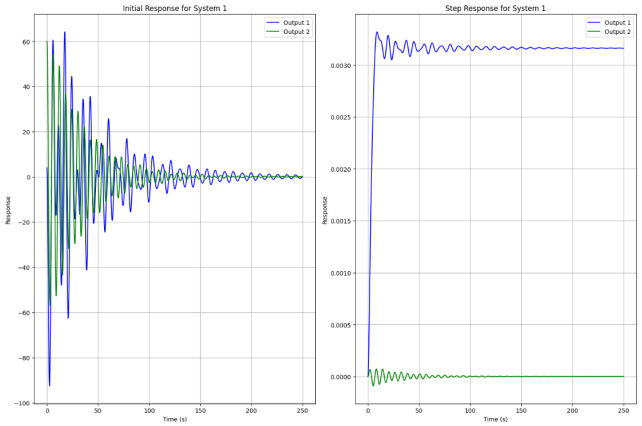


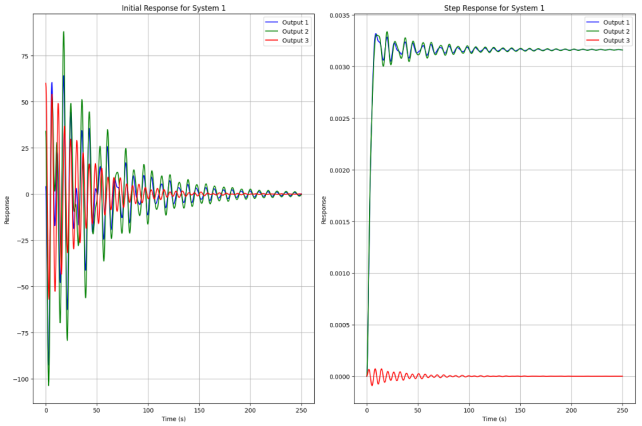
```

import numpy as np
import control as ctrl
import matplotlib.pyplot as plt

def define_system_parameters():
    """
    Define system parameters and return the system matrices A, B, and D.
    """
    M = 1000        # Mass of the cart (kg)
    mass_1 = 100    # Mass of Pendulum 1 (kg)
    mass_2 = 100    # Mass of Pendulum 2 (kg)
    length_1 = 20   # Length of pendulum 1 (m)
    length_2 = 10   # Length of pendulum 2 (m)
    g = 9.81        # Gravitational acceleration (m/s^2)

    A = np.array([[0, 1, 0, 0, 0, 0],
                  [0, 0, -(mass_1*g)/M, 0, -(mass_2*g)/M, 0],
                  [0, 0, 0, 1, 0, 0],
                  [0, 0, -((M+mass_1)*g)/(M*length_1), 0, -(mass_2*g)/(M*length_1), 0],
                  [0, 0, 0, 0, 0, 1],
                  [0, 0, -(mass_1*g)/(M*length_2), 0, -(g*(M+mass_2))/(M*length_2), 0]])

    B = np.array([[0], [1/M], [0], [1/(M*length_1)], [0], [1/(M*length_2)]])
    D = np.zeros((1, 1))  # No direct feedthrough

    return A, B, D

def define_output_matrices():
    """
    Define the different output matrices (C_matrices) for different observables.
    """
    C_matrices = [
        np.array([[1, 0, 0, 0, 0, 0]]),  # Observing x component
        np.array([[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0]]),  # Observing x and theta2
        np.array([[1, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0]])  # Observing x, theta1, and theta2
    ]

    return C_matrices

def design_lqr_controller(A, B, Q, R):
    """
    Design the LQR controller.
    """
    K, S, eigen_values = ctrl.lqr(A, B, Q, R)
    return K

def design_kalman_filter(A, C, v_d, v_n):
    """
    Design the Kalman filter for a given system using LQR method.
    """
    R_kalman = v_n * np.eye(C.shape[0])  # Measurement noise covariance
    K_pop = ctrl.lqr(A.T, C.T, v_d, R_kalman)[0].T
    return K_pop

def construct_lqg_system(A, B, C, K, K_pop):
    """
    Construct the state-space model for the LQG system (state feedback + Kalman observer).
    """
    sys = ctrl.ss(
        np.block([
            [A - np.dot(B, K), np.dot(B, K)],  # State feedback part
            [np.zeros_like(A), A - np.dot(K_pop, C)]  # Observer part (Kalman filter)
        ]),
        np.block([[B], [np.zeros_like(B)]]),
        np.block([C, np.zeros_like(C)]),
        np.zeros((C.shape[0], B.shape[1]))
    )
    return sys

```

```python
def plot_responses(sys, C_matrices, initial_conditions, colors):
    """
    Plot the initial and step responses for each system.
    """
    plt.figure(figsize=(15, 10))

    for i, C in enumerate(C_matrices):
        # Initial response
        plt.subplot(len(C_matrices), 2, 2*i + 1)
        T, yout = ctrl.initial_response(sys, T=np.linspace(0, 250, 1000), X0=initial_conditions)
        yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output
        if yout.ndim == 1:
            plt.plot(T, yout, color=colors[i])
        else:
            for j in range(yout.shape[0]):
                plt.plot(T, yout[j, :], label=f'Output {j+1}', color=colors[j])
        plt.title(f'Initial Response for System {i+1}')
        plt.xlabel('Time (s)')
        plt.ylabel('Response')
        plt.legend()
        plt.grid(True)

        # Step response
        plt.subplot(len(C_matrices), 2, 2*i + 2)
        T, yout = ctrl.step_response(sys, T=np.linspace(0, 250, 1000))
        yout = np.squeeze(yout)  # Ensure yout is a 1D array if it has only one output
        if yout.ndim == 1:
            plt.plot(T, yout, color=colors[i])
        else:
            for j in range(yout.shape[0]):
                plt.plot(T, yout[j, :], label=f'Output {j+1}', color=colors[j])
        plt.title(f'Step Response for System {i+1}')
        plt.xlabel('Time (s)')
        plt.ylabel('Response')
        plt.legend()
        plt.grid(True)

    plt.tight_layout()
    plt.show()

def run_lqg_design(A, B, C_matrices, initial_conditions, Q, R, v_d, v_n, colors):
    """
    Main function to perform LQG design and plot responses for different C_matrices.
    """
    # Design LQR controller
    K = design_lqr_controller(A, B, Q, R)

    # Loop through each C_matrix and plot responses
    for i, C in enumerate(C_matrices):
        # Design the Kalman filter for each system
        K_pop = design_kalman_filter(A, C, v_d, v_n)

        # Construct the state-space model for LQG
        sys = construct_lqg_system(A, B, C, K, K_pop)

        # Plot the initial and step responses for this system
        plot_responses(sys, [C], initial_conditions, colors)

# 2. Define initial conditions and noise parameters
initial_conditions = np.array([4, 0, 30, 0, 60, 0, 0, 0, 0, 0, 0, 0])
colors = ['b', 'g', 'r']  # Colors for plotting

# 3. Define system parameters
A, B, D = define_system_parameters()

# 4. Define output matrices
C_matrices = define_output_matrices()

# 5. Define LQR and Kalman filter parameters
Q = np.diag([100, 100, 100, 100, 100, 100])  # State weighting
R = np.array([[0.001]])  # Control effort weighting
```

```
v_d = 0.3 * np.eye(6)     # Process noise covariance
v_n = 1                   # Measurement noise covariance

# 6. Run LQG Design and Plot Responses
run_lqg_design(A, B, C_matrices, initial_conditions, Q, R, v_d, v_n, colors)

```



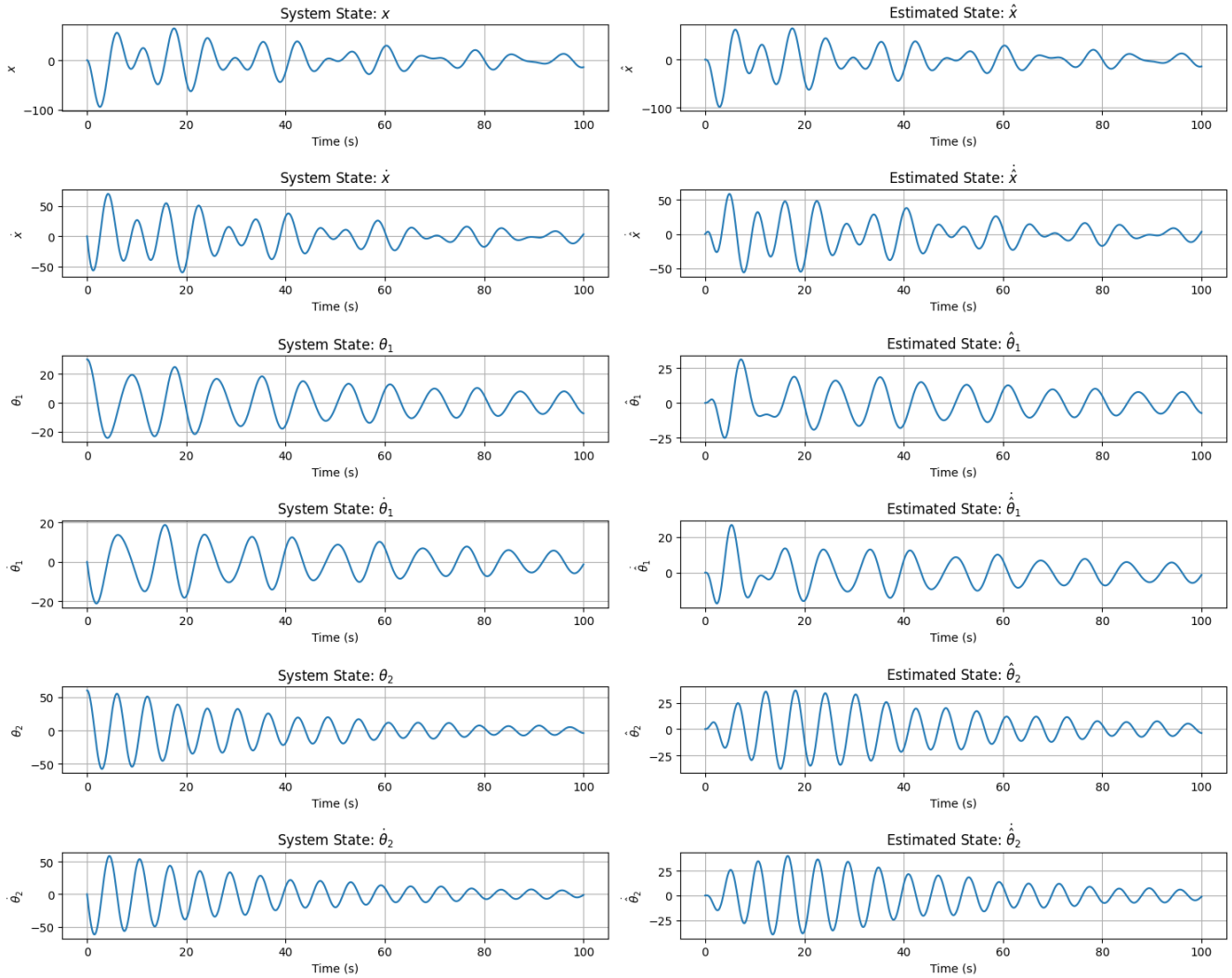





```

import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.integrate import odeint
import control as ctrl

# Part 1: Define System Parameters and State-Space Matrices

def define_system_parameters():
    # Define system parameters
    M = 1000
    mass_1 = 100
    mass_2 = 100
    length_1 = 20
    length_2 = 10
    g = 9.81
    return M, mass_1, mass_2, length_1, length_2, g

def construct_state_space(M, mass_1, mass_2, length_1, length_2, g):
    # Construct the state-space matrices A and B
    A = np.array([[0, 1, 0, 0, 0, 0],
                  [0, 0, -(mass_1 * g) / M, 0, -(mass_2 * g) / M, 0],
                  [0, 0, 0, 1, 0, 0],
                  [0, 0, -((M + mass_1) * g) / (M * length_1), 0, -(mass_2 * g) / (M * length_1), 0],
                  [0, 0, 0, 0, 0, 1],
                  [0, 0, -(mass_1 * g) / (M * length_2), 0, -((M + mass_2) * g) / (M * length_2), 0]])

    B = np.array([[0], [1 / M], [0], [1 / (M * length_1)], [0], [1 / (M * length_2)]])
    C_matrix = np.array([[1, 0, 0, 0, 0, 0]])   # C for x
    return A, B, C_matrix

# Part 2: LQR Design

def design_lqr(A, B):
    Q = np.diag([1000, 1000, 100, 10, 1000, 100])
    R = 0.01
    K, _, _ = ctrl.lqr(A, B, Q, R)
    return K

# Part 3: Kalman Filter Design

def design_kalman_filter(A, C_matrix):
    v_d = 0.3 * np.eye(6)
    v_n = 1
    K_pop = ctrl.lqr(A.T, C_matrix.T, v_d, v_n)[0].T
    return K_pop

# Part 4: Define Nonlinear LQG Control Function

def lqg_dynamics(y, t, A, B, K, C_matrix, K_pop):
    # Control input
    controlled_force = -K @ y[:6]
    # Estimator dynamics
    dx_hat = A @ y[6:12] + B @ controlled_force + K_pop @ (C_matrix @ y[:6] - C_matrix @ y[6:12])
    # System dynamics (controlled system)
    dx = A @ y[:6] + B @ controlled_force
    return np.concatenate((dx, dx_hat))

# Part 5: Run Simulation

def run_simulation(lqg_dynamics, A, B, K, C_matrix, K_pop, initial_conditions, time_span):
    # Solving the system using the odeint solver
    x = odeint(lqg_dynamics, initial_conditions, time_span, args=(A, B, K, C_matrix, K_pop))
    return x

# Part 6: Plot Results

def plot_results(time_span, x):
    plt.figure(figsize=(15, 12))

    # First six plots: System States
    for i, label in enumerate(['$x$', '$\dot{x}$', '$\\theta_1$', '$\dot{\\theta}_1$', '$\\theta_2$', '$\dot{\\theta}_2$']):
        plt.subplot(6, 2, 2*i + 1)  # Plot on odd indices (1, 3, 5, 7, 9, 11)
        plt.plot(time_span, x[:, i])
```

```
        plt.title(f'System State: {label}')
        plt.xlabel('Time (s)')
        plt.ylabel(label)
        plt.grid(True)

    # Next six plots: Estimated States
    for i, label in enumerate(['$\hat{x}$', '$\dot{\hat{x}}$', '$\hat{\\theta}_1$', '$\dot{\hat{\\theta}}_1$', '$\hat{\\theta}_2$',
        plt.subplot(6, 2, 2*i + 2)  # Plot on even indices (2, 4, 6, 8, 10, 12)
        plt.plot(time_span, x[:, i + 6])
        plt.title(f'Estimated State: {label}')
        plt.xlabel('Time (s)')
        plt.ylabel(label)
        plt.grid(True)

    plt.tight_layout()
    plt.show()



# Step 1: Define system parameters
M, mass_1, mass_2, length_1, length_2, g = define_system_parameters()

# Step 2: Construct state-space matrices
A, B, C_matrix = construct_state_space(M, mass_1, mass_2, length_1, length_2, g)

# Step 3: Design LQR controller
K = design_lqr(A, B)

# Step 4: Design Kalman Filter
K_pop = design_kalman_filter(A, C_matrix)

# Step 5: Set initial conditions and time span for the simulation
initial_conditions = np.array([0, 0, 30, 0, 60, 0, 0, 0, 0, 0, 0, 0])
time_span = np.linspace(0, 100, 1001)

# Step 6: Run the simulation
x = run_simulation(lqg_dynamics, A, B, K, C_matrix, K_pop, initial_conditions, time_span)

# Step 7: Plot results
plot_results(time_span, x)
```


System State: x
Estimated State: $\hat{x}$
System State: $\dot{x}$
Estimated State: $\dot{\hat{x}}$
System State: θ_1
Estimated State: $\hat{\theta}_1$
System State: $\dot{\theta}_1$
Estimated State: $\dot{\hat{\theta}}_1$
System State: θ_2
Estimated State: $\hat{\theta}_2$
System State: $\dot{\theta}_2$
Estimated State: $\dot{\hat{\theta}}_2$
Time (s)