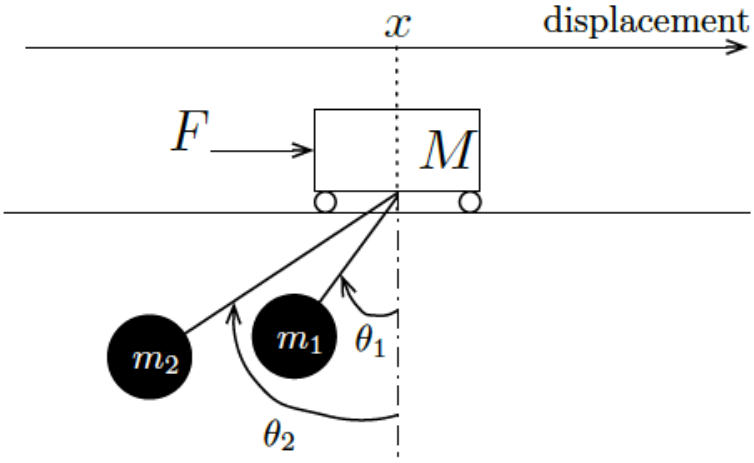
Final Project

ENPM667
Control of Robotic Systems

Vishnu Mandala - 119452608
Rohit Suresh - 119283684

Problem 1: First Component

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.



A. **Obtain the equations of motion for the system and the corresponding nonlinear state-space representation.**
B. **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.**
C. **Obtain conditions on M, m_1, m_2, l_1, l_2 for which the linearized system is controllable.**
D. **Choose M = 1000Kg, $m_1 = m_2 = 100$Kg, $l_1 = 20$m and $l_2 = 10$m. 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.**

Parameters referenced in the report:

a. **Masses:**
   - ⇨ m_1: Load1's mass
   - ⇨ m_2: Load2's mass
   - ⇨ M: Crane's mass

b. **Constants:**
   - ⇨ g: Acceleration due to gravity

c. **Lengths:**
   - ⇨ l_1: Cable 1's length
   - ⇨ l_2: Cable 2's length

d. **Angles:**
   - ⇨ θ_1: Load1's angle with the vertical
   - ⇨ θ_2: Load2's angle with the vertical

e. **Heights:**
   - ⇨ h_1: Effective height of load1 from the crane's base
   - ⇨ h_2: Effective height of load2 from the crane's base

f. **External Force:**
   - ⇨ F: External force

Assumptions

- There is no friction between the cart and the ground.
- The motion occurs along a single direction.
- The poles for the Luenberger observer are placed significantly far from the system's natural poles.

- Damping is observed in the estimates, implying that the system will eventually reach an equilibrium point if given more time.
- The acceleration due to gravity is assumed to be 9.8 m/s².
- No collision occurs between the two loads in the system.
- The length of the strings remains constant over time, with consistent tension in the strings.

Solution:

This project aims to develop Linear Quadratic Regulator (LQR) and Linear Quadratic Gaussian (LQG) controllers for a specialized crane system that supports two loads, each suspended on cables of differing lengths (l1, l2) and with different masses (m1, m2). The approach consists of several structured phases:

1. **Modeling the System Dynamics**: Initially, we employ the Lagrangian method to derive the equations of motion for the crane and its loads. This step is crucial for understanding the system's dynamics and lays the foundation for subsequent control design.

2. **State-Space Representation**: Following the dynamical modeling, we construct a non-linear state-space representation of the system. This representation captures the system's behavior and serves as a basis for further analysis and controller design.

3. **Linearization of the System**: Given the non-linear nature of the system, our next step involves linearizing it around a chosen equilibrium point. This simplification allows us to apply linear control theory techniques. The linearized model is represented in state-space form, facilitating the design and analysis of the proposed controllers.

4. **Assessing Controllability**: Before proceeding to controller design, we investigate the controllability of the system. This analysis, grounded in the physical parameters (M, m_1, m_2, l_1, l2), determines whether the system's states can be steered to a desired configuration under the influence of the control inputs.

5. **Designing the LQR Controller**: With the system confirmed to be controllable, we proceed to design an LQR controller. This controller aims to optimize a quadratic cost function while ensuring robust and efficient control of the crane and load system.

6. **Simulation and Analysis**: To validate our controller design, we conduct simulations under two distinct scenarios: applying the LQR controller to both the original non-linear system and its linearized counterpart. By adjusting the LQR parameters, we aim to achieve an optimal response from the system. The results of these simulations are critically analyzed to assess performance.

7. **Stability Verification**: Finally, we perform a Lyapunov analysis on the closed-loop system with the LQR controller implemented. This step is essential to certify the system's stability, ensuring that the designed controller reliably maintains the system in a safe and stable state under various operating conditions.

A. Equations of Motion

Expressing the position of mass m_1 as a function with respect to the variable θ_1

$$x_{m1} = (x - l_1 \sin(\theta_1))\hat{\imath} + (-l_1 \cos(\theta_1))\hat{\jmath}$$

Differentiating it with respect to time to obtain velocity

$$\overrightarrow{v_{m1}} = \left(\dot{x} - l_1 \cos(\theta_1)\, \dot{\theta_1}\right)\hat{\imath} + \left(l_1 \sin(\theta_1)\, \dot{\theta_1}\right)\hat{\jmath}$$

Similarly, position of mass m_2 and its velocity in parametric form is

$$x_{m2} = (x - l_2 \sin(\theta_2))\hat{\imath} + (-l_2 \cos(\theta_2))\hat{\jmath}$$

$$\overrightarrow{v_{m2}} = \left(\dot{x} - l_2 \cos(\theta_2)\, \dot{\theta_2}\right)\hat{\imath} + \left(l_2 \sin(\theta_2)\, \dot{\theta_2}\right)\hat{\jmath}$$

We consider only x directions as the displacement of the cart only occurs in the positive x-direction. We can now establish the system's kinetic energy using the two velocity equations we derived earlier

$$K = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m_1\left(\dot{x} - l_1\dot{\theta}_1\cos(\theta_1)\right)^2 + \frac{1}{2}m_1\left(l_1\dot{\theta}_1\sin(\theta_1)\right)^2 + \frac{1}{2}m_2\left(\dot{x} - l_2\dot{\theta}_2\cos(\theta_2)\right)^2 + \frac{1}{2}m_2\left(l_2\dot{\theta}_2\sin(\theta_2)\right)^2$$

We can write the Potential Energy considering the cart's height as reference as

$$P = -m_1 g l_1\cos(\theta_1) - m_2 g l_2\cos(\theta_2) = -g[m_1 l_1\cos(\theta_1) + m_2 l_2\cos(\theta_2)]$$

Using these, we can write the Lagrange Equation as a difference of Kinetic and Potential Energies

$$L = K - P = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m_1\dot{x}^2 + \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2\cos^2(\theta_1) - m_1 l_1\dot{x}\dot{\theta}_1\cos(\theta_1) + \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2\sin^2(\theta_1) + \frac{1}{2}m_2\dot{x}^2 + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2\cos^2(\theta_2) - m_2 l_2\dot{x}\dot{\theta}_2\cos(\theta_2) + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2\sin^2(\theta_2) + g[m_1 l_1\cos(\theta_1) + m_2 l_2\cos(\theta_2)]$$

$$\Rightarrow \quad L = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}(m_1 + m_2)\dot{x}^2 + \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2 - (m_1 l_1\dot{x}\cos(\theta_1) + m_2 l_2\dot{x}\cos(\theta_2)) + g(m_1 l_1\cos(\theta_1) + m_2 l_2\cos(\theta_2))$$

The Lyapunov Equations, pertaining to the considered state variables of the system, are defined as follows:

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

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

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

Based on the above equations, we can write

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

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

Here, $\frac{\partial L}{\partial x} = 0$

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

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

We also know that, for load 1

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

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

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

where, $\frac{\partial L}{\partial \theta_1} = -m_1 l_1\dot{x}\sin(\theta_1) - m_1 g l_1\sin(\theta_1)$

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

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

We use a similar method to find the third Lagrange Equation

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

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

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

$$\frac{\partial L}{\partial \theta_2} = m_2 l_2 \dot{\theta_2} \dot{x} \sin(\theta_2) - m_2 g l_2 \sin(\theta_2)$$

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

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

Prior to linearization around the specified equilibrium points, we express the equations for the second derivatives of certain state variables, as derived from the equations mentioned above:

$$\ddot{x} = \frac{1}{M + m_1 + m_2}\left[m_1 l_1 \ddot{\theta_1} \cos(\theta_1) + m_2 l_2 \ddot{\theta_2} \cos(\theta_2) - m_1 l_1 \dot{\theta_1}^2 \sin(\theta_1) - m_2 l_2 \dot{\theta_2}^2 \sin(\theta_2) + F\right]$$

$$\ddot{\theta_1} = \frac{\ddot{x} \cos(\theta_1) - g \sin(\theta_1)}{l_1}$$

$$\ddot{\theta_2} = \frac{\ddot{x} \cos(\theta_2) - g \sin(\theta_2)}{l_2}$$

When examining the system's states, we can formulate the nonlinear system's state-space representation as follows:

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

B. Linearized System about the Equilibrium Point

Linearization involves finding the linear approximation of a function at a specific point, typically through the first-order Taylor expansion. In the context of dynamical systems, it is a technique used to assess the local stability of an equilibrium point in a system of nonlinear differential equations or discrete dynamical systems. Linearization allows us to apply linear system analysis tools to understand the behavior of a nonlinear function near a particular point.

As previously discussed, we have derived the equations of motion for the cart system with two pendulums and represented them in state-space form. It is evident that these equations contain nonlinear components, particularly due to the presence of sine and cosine terms, making them challenging to solve directly.

To address this complexity, we employ linearization to approximate the system's behavior around the equilibrium point defined as $x = 0$, $\theta_1 = 0$, and $\theta_2 = 0$, as specified in the problem statement. The process of linearization around the state variables is accomplished by employing the Jacobian linearization matrix, as presented below:

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

The limiting condition at equilibrium, we set,

$$\sin\theta_1 \approx \theta_1$$
$$\sin\theta_2 \approx \theta_2$$
$$\cos\theta_1 \approx 1$$
$$\cos\theta_2 \approx 1$$
$$\dot{\theta_1}^2 = \dot{\theta_2}^2 \approx 0$$

The state space representation of linearized system can then be written as,

$$\dot{X} = AX(t) + BU(t)$$

$$\Rightarrow \dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta_1} \\ \ddot{\theta_1} \\ \dot{\theta_2} \\ \ddot{\theta_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-gm_1}{M} & 0 & \frac{-gm_2}{M} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{g+\frac{gm_1}{M}}{l_1} & 0 & \frac{-gm_2}{Ml_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-gm_1}{Ml_2} & 0 & -\frac{g+\frac{gm_2}{M}}{l_2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_1 \\ \dot{\theta_1} \\ \theta_2 \\ \dot{\theta_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml_1} \\ 0 \\ \frac{1}{Ml_1} \end{bmatrix} U(t)$$

C. Controllability

The linear time-varying system is considered controllable over the interval (0, T) if and only if the square Gramian matrix of controllability is invertible. For the Gramian matrix C(A, B) to be invertible, the controllability matrix must satisfy the following conditions:

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

The controllability matrix can be defined as follows:

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

Finding the determinant of Matrix C

$$rank = |C| = \frac{g^6l_1^2 - 2g^6l_1l_2 + g^6l_2^2}{M^6l_1^6l_2^6}$$

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

$$\Rightarrow g^6l_1^2 - 2g^6l_1l_2 + g^6l_2^2 \neq 0$$
$$(g^3l_1 - g^3l_2)^2 \neq 0$$
$$(g^3l_1)^2 \neq g^3l_2$$
$$l_1 \neq l_2$$

We observe that when $l_1 = l_2$, the determinant of this matrix becomes zero. Additionally, when either l_1 or l_2 is equal to zero, controllability cannot be determined. Therefore, the conditions required to achieve controllability are as follows:

$$\boxed{\begin{matrix} l_1 \neq l_2 \\ l_1 \neq 0 \\ l_2 \neq 0 \end{matrix}}$$

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

D. LQR Controller

To achieve controllability in the system, various controllers can be employed to drive the output to the desired values. The choice of controller depends on the system's model and characteristics. Two commonly used controllers are PID (Proportional-Integral-Derivative) and LQR (Linear Quadratic Regulator), each suited to different scenarios.

- PID Controller: This controller follows a classical linear equation approach and is well-suited for linear systems. It adjusts the control input based on the error between the desired and actual output.
- LQR Controller: LQR, on the other hand, is particularly effective for non-linear models. It provides an optimal state-feedback law that minimizes a specific quadratic objective function. In essence, LQR is regarded as one of the best controllers.

For the LQR Controller, when A and B_k are stabilizable, we can seek a control gain matrix 'k' that minimizes the following cost function:

$$J\left(k, \hat{X}(0)\right) = \int_0^\infty \widehat{X^T}(t) Q\, \hat{X}(t) + \widehat{U_k^T}(t) R \widehat{U_k}(t)\, dt$$

Here,

'Q' is a non-negative definite matrix, penalizing performance deviations.
'R' is a positive definite matrix, penalizing the effort or energy consumed by the control inputs.
Various values of 'Q' and 'R' can be selected to optimize the system's control. The resulting output graphs help validate the effectiveness of the chosen 'Q' and 'R' matrices.

The choice of 'Q' depends on how quickly the system should stabilize in a particular state and the allowable error band within which it should operate. A lower 'R' can lead to faster stabilization as it reduces the energy utilized by the control inputs.

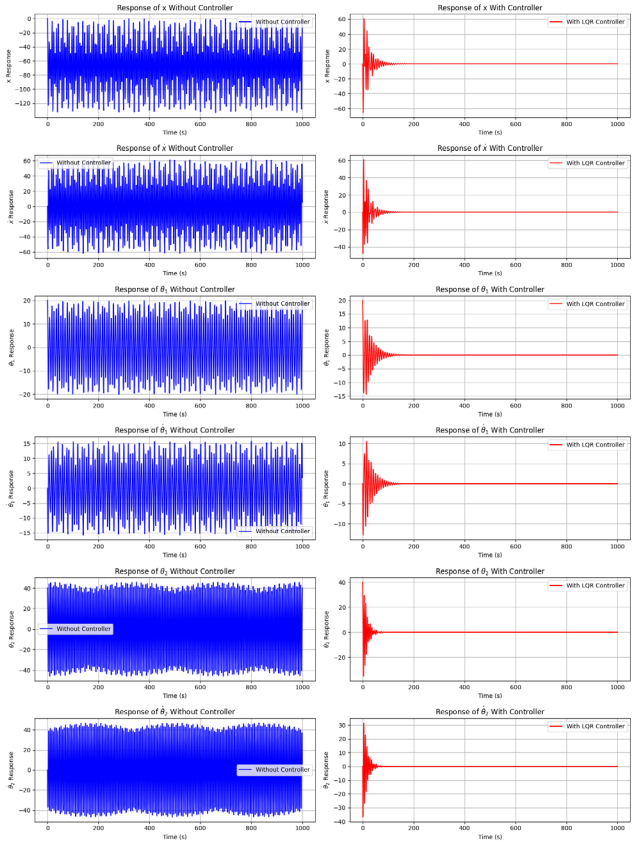


Figure 1. LQR Linear Output

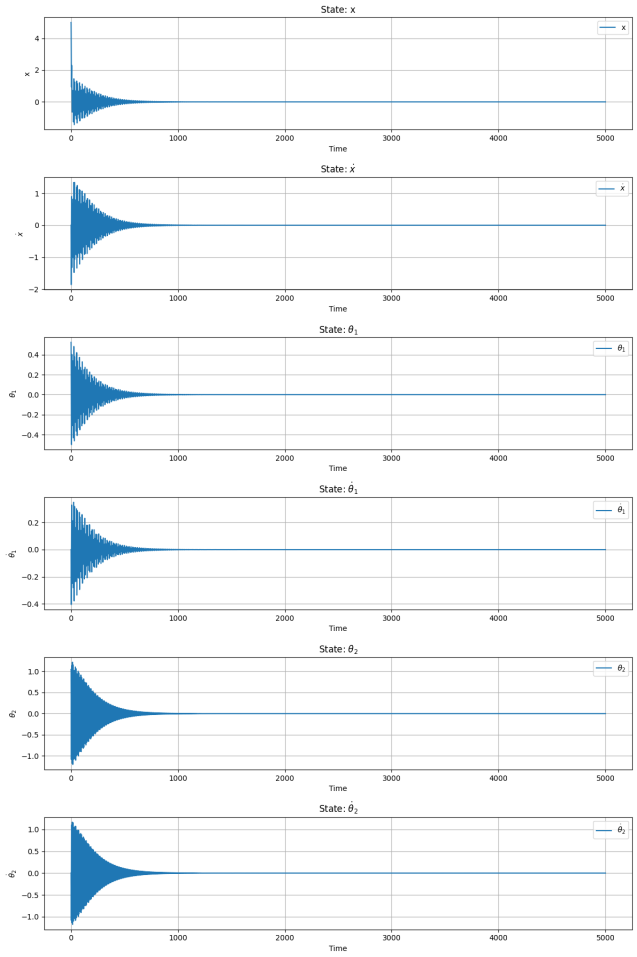


Figure 2. LQR Non-Linear Output

Lyapunov's indirect method of stability analysis, also known as the linearization method, involves examining the eigenvalues of the system's Jacobian matrix at an equilibrium point (in our case, the matrix A−BK). The method relies on the following principle for continuous-time systems:

- If all eigenvalues have negative real parts, the equilibrium point is asymptotically stable.
- If any eigenvalue has a positive real part, the equilibrium point is unstable.
- If some eigenvalues have zero real parts and the rest have negative real parts, the system may be stable or unstable, and further analysis is required (the system is on the boundary of stability).

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

- $\lambda_1 = -0.38230955 + 0.35462265j$
- $\lambda_2 = -0.38230955 - 0.35462265j$
- $\lambda_3 = -0.07230465 + 1.03416433j$
- $\lambda_4 = -0.07230465 - 1.03416433j$
- $\lambda_5 = -0.0327691 + 0.72083929j$
- $\lambda_6 = -0.0327691 - 0.72083929j$

Observing the eigenvalues listed, we note that each eigenvalue has a negative real part. This leads us to conclude that the system is indeed asymptotically stable. This implies that for any small perturbation from the equilibrium state, the system's states will converge back to the equilibrium over time.

The presence of complex parts in the eigenvalues also indicates that the system response will exhibit oscillatory behavior as it converges. However, the damping effect, as evidenced by the negative real parts, ensures that these oscillations diminish over time, and the system does not exhibit sustained oscillations or grow unbounded.

In conclusion, the Lyapunov indirect stability analysis confirms that the closed-loop system is stable under the designed LQR controller. As such, the control strategy successfully stabilizes the system around the equilibrium point.

Problem 2: Second Component (Consider the parameters selected in [C] above)

E. Suppose that you can select the following output vectors: x(t), (θ_1(t), θ_2(t)), (x(t), θ_2(t)) or (x(t), θ_1(t), θ_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?

Solution:

After designing an LQR controller for the system, the next step is to evaluate whether the system is observable with respect to certain output vectors. The parameters used to establish controllability conditions play a crucial role in this assessment.

The project's continuation instructs us to:

1. Determine the best Luenberger Observer for each of the output vectors if they are observable. This involves selecting appropriate observer structures to estimate the system's state variables accurately.
2. Simulate the system's response to specific input conditions, including unit step input, for both the linearized and non-linearized versions of the system. This step allows us to observe how well the designed controllers and observers perform in practice.
3. As the final step, design an output feedback controller using the LQG (Linear Quadratic Gaussian) method for the smallest output vector. This controller aims to optimize the system's performance by considering both state estimation and control. The performance of this controller can be illustrated through simulation results.

These steps collectively ensure a comprehensive evaluation of the system's observability, controller design, and performance assessment.

E. Observability

Observability in a control system pertains to the system's capability to deduce its internal states by examining the system's outputs within a finite time interval when subjected to inputs.

For a Linear Time-Invariant (LTI) discrete system represented in state-space form as: $X(k+1) = AX(k)\ X(0) = X$

The system's output, denoted as Y, can be expressed as: $Y = CX + DU$

To determine observability, we employ the observability matrix denoted as O(A, C), which is defined as:

$$O(A, C) = [C\ CA\ CA^2 \ldots CA^{n-1}]^T$$

Observability is confirmed when the rank of the observability matrix, Rank(O(A, C)), equals the full rank 's'. In such cases, we can assert that the considered system is indeed observable.

Outputs	C Matrix	Rank	Observable
x	[1 0 0 0 0 0]	6	Yes
theta_1 and theta_2	[0 0 1 0 0 0; 0 0 0 1 0 0]	4	No
x and theta_2	[1 0 0 0 0 0; 0 0 0 0 1 0]	6	Yes
x, theta_1 and theta_2	[1 0 0 0 0 0; 1 0 1 0 0 0; 0 0 0 0 1 0]	6	Yes

Figure 3. Observability Output

F. Luenberger Observer

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

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

Here, 'L' represents the observer gain, and $\hat{X}(t)$ is our 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. For enhanced convergence, we position them at a location approximately 3 times the distance from the eigenvalues of $A-BK$, where 'K' stands for our feedback gain.

The following code will simulate both the nonlinear and linear systems. It considers the initial conditions and introduces a step input at $t = 200$ seconds:

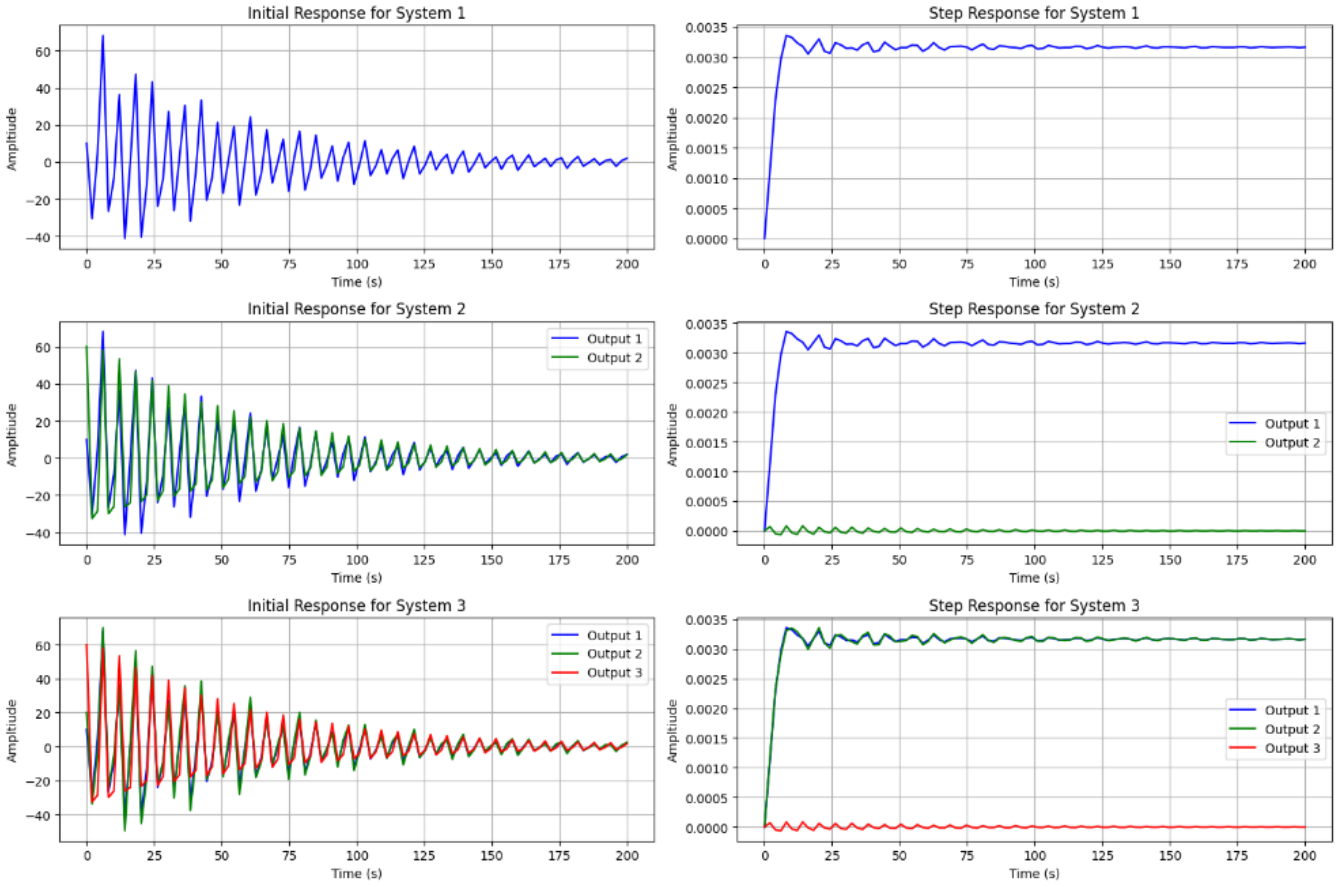


Figure 4. Luenberger Linear Output

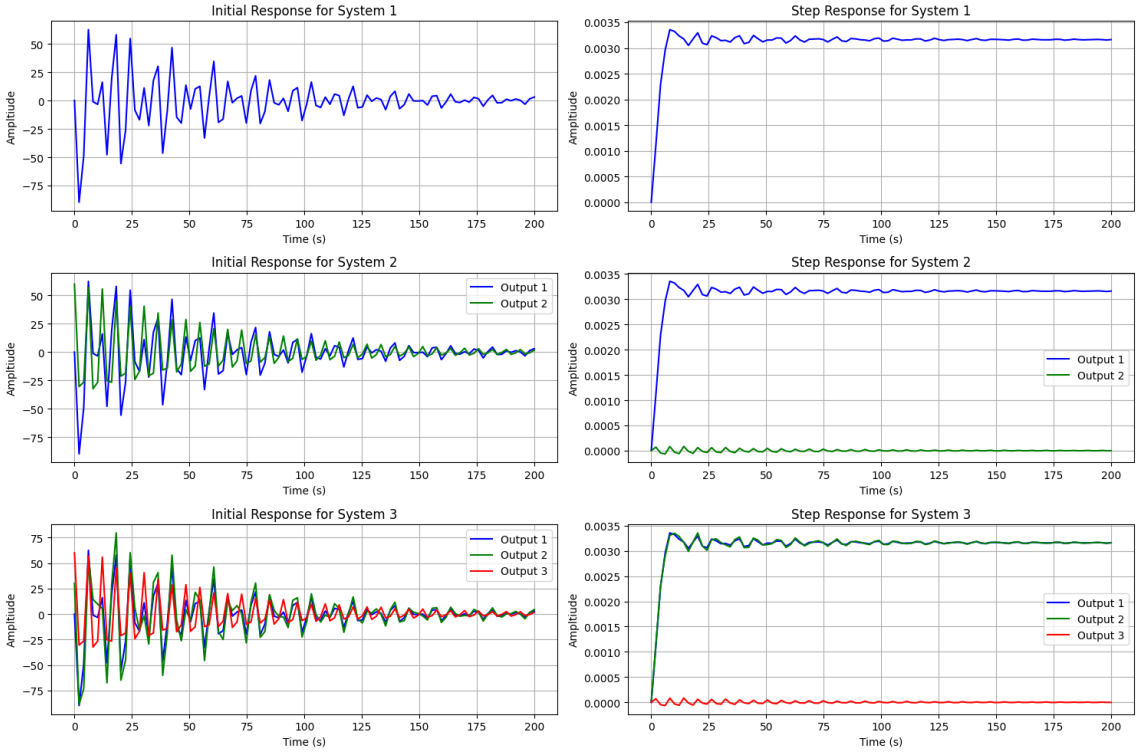


Figure 5. Luenberger Non-Linear Output

G. LQG Controller

The LQG (Linear Quadratic Gaussian) controller is a combination of a Kalman filter and a linear quadratic regulator (LQR). 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 follows the standard output feedback configuration, where the Luenberger observer is used in conjunction with 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, respectively.

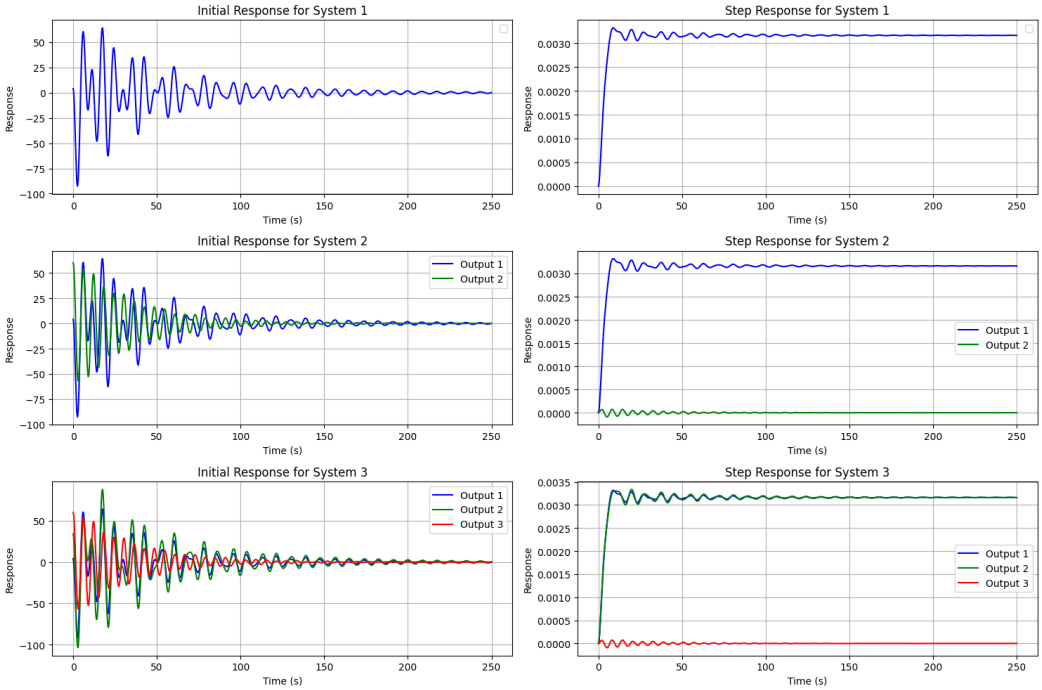


Figure 6. LQG Linear Controller

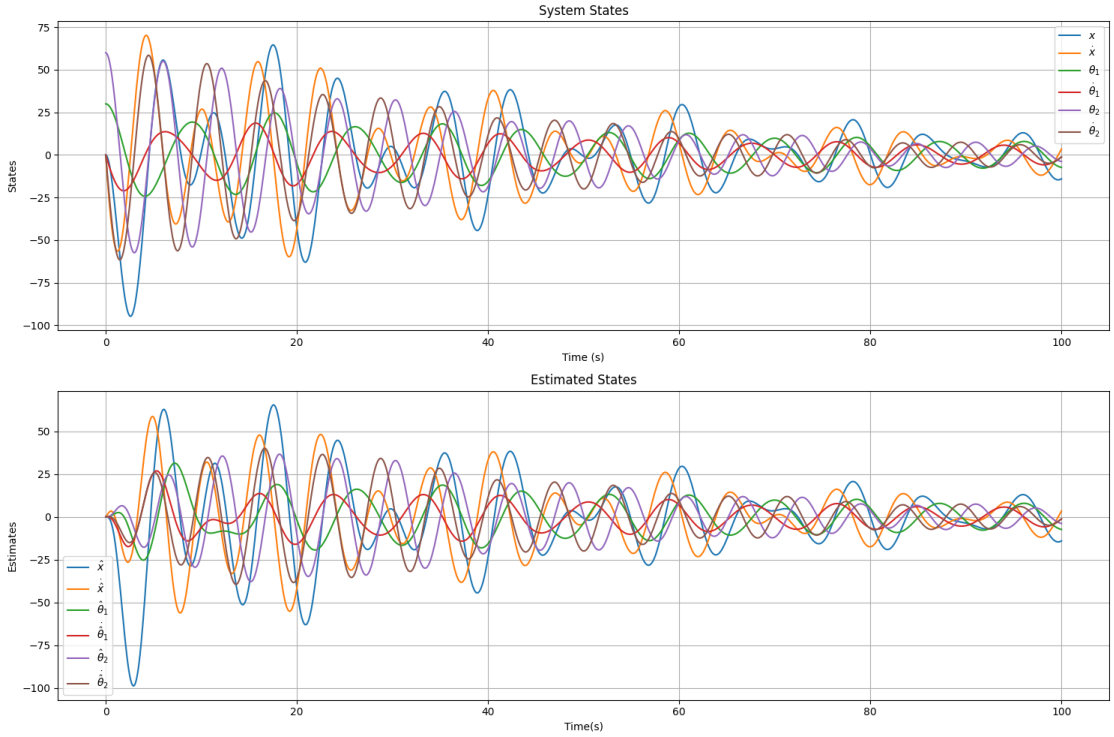


Figure 7. LQG Non-Linear Controller

To achieve asymptotic tracking of a constant reference, we fine-tune the controller in a way that the variables 'x' and 'u' converge to the referenced values over time 't'. During this reference tracking process, the system incurs a cost associated with the current deviations from the reference constraints.

The primary objective is to minimize this cost, which can be expressed as:

$$\int_0^\infty \left(\hat{X}(t) - X_d\right)^T Q \left(\hat{X}(t) - X_d\right) + \left(\widehat{U_k} - U_\infty\right)^T R\left(\widehat{U_k} - U_\infty\right) dt$$

This cost can be minimized effectively if there exists a solution U_∞ such that: $A\widehat{X_d} + B\widehat{U_k} = 0$

The optimal solution for U_∞ is given by: $U(t) = K(\hat{X}(t) - X_d) + U_\infty$ where $K = -R^{-1}B^TP$ and P is the positive definite matrix solution of the Riccati equation: $A^TP + PA - PBR^{-1}B^TP = -Q$

By satisfying these constraints, we can drive the system to reach the desired output when there is a constant tracking reference 'x' that satisfies the equation: $\mathbf{X}_{\text{ref}} = AX_{\text{ref}} + BU_{\text{ref}}$

This design is equipped to handle constant force disturbances applied to the cart. Assuming these force disturbances follow a Gaussian distribution, the controller can effectively account for and adapt to these disturbances in the system.

APPENDIX

Google Drive Link for Simulation:
https://drive.google.com/file/d/1l3Owrd25iIktncBgdGc_2vUhpnY4jZJi/view?usp=sharing

Part C: Controllability Check

In [ ]:
```
import sympy as sp
```

In [ ]:
```
# Define the symbolic variables
M, m1, m2, l1, l2, g = sp.symbols('M m1 m2 l1 l2 g')

# 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]
])
print("A: ")
A
```

A:

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

In [ ]:
```
# Defining the B matrix
B = sp.Matrix([0, 1/M, 0, 1/(M*l1), 0, 1/(M*l2)]).reshape(6, 1)
print("B: ")
B
```

B:

Out[ ]:
$$\begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml_1} \\ 0 \\ \frac{1}{Ml_2} \end{bmatrix}$$

In [ ]:
```
# Constructing the controllability matrix
Ct = sp.Matrix.hstack(B, A*B, A**2*B, A**3*B, A**4*B, A**5*B)
print("Controllability Matrix: ")
Ct
```

Controllability Matrix:

Out[ ]:

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

In [ ]:
```
# Simplifying and finding the determinant of the controllability matrix
det_Ct = sp.simplify(sp.det(Ct))
print("Determinant of the controllability matrix:")
det_Ct
```

```
Determinant of the controllability matrix:
```

Out[ ]:

$$\frac{g^6\left(-l_1^2+2l_1l_2-l_2^2\right)}{M^6l_1^6l_2^6}$$

In [ ]:
```
# Displaying the rank of the controllability matrix
rank_controllability = Ct.rank()
print("Rank of the controllability matrix:")
rank_controllability
```

```
Rank of the controllability matrix:
6
```

Out[ ]:

In [ ]:
```
# Checking for special case where pendulum_length1 equals pendulum_length2
Ct1 = Ct.subs(l1, l2)
print("For l1 = l2, the Controllability matrix is:")
Ct1
```

```
For l1 = l2, the Controllability matrix is:
```

Out[ ]:

$$\begin{bmatrix} 0 & \frac{1}{M} & 0 & -\frac{gm_1}{M^2l_2} - \frac{gm_2}{M^2l_2} & 0 \\ \frac{1}{M} & 0 & -\frac{gm_1}{M^2l_2} - \frac{gm_2}{M^2l_2} & 0 & \frac{Mg^2l_2m_1+g^2l_2m_1^2+g^2l_2m_1m_2}{M^3l_2^3} + \frac{Mg^2l}{} \\ 0 & \frac{1}{Ml_2} & 0 & -\frac{gm_2}{M^2l_2^2} + \frac{-Mg-gm_1}{M^2l_2^2} & 0 \\ \frac{1}{Ml_2} & 0 & -\frac{gm_2}{M^2l_2^2} + \frac{-Mg-gm_1}{M^2l_2^2} & 0 & \frac{2Mg^2l_2m_2+g^2l_2m_1m_2+g^2l_2m_2^2}{M^3l_2^4} + \frac{M^2g^2l_2+}{} \\ 0 & \frac{1}{Ml_2} & 0 & -\frac{gm_1}{M^2l_2^2} + \frac{-Mg-gm_2}{M^2l_2^2} & 0 \\ \frac{1}{Ml_2} & 0 & -\frac{gm_1}{M^2l_2^2} + \frac{-Mg-gm_2}{M^2l_2^2} & 0 & \frac{2Mg^2l_2m_1+g^2l_2m_1^2+g^2l_2m_1m_2}{M^3l_2^4} + \frac{M^2g^2l_2+}{} \end{bmatrix}$$

In [ ]:
```
print("Rank of the new matrix:")
rank_Ct1 = Ct1.rank()
rank_Ct1
```

```
Rank of the new matrix:
```

Out[ ]:
```
4
```

In [ ]:
```
# Displaying the system's controllability condition
if rank_Ct1 == rank_controllability:
    print("System is controllable as ranks are equal.")
else:
    print("System is not controllable as ranks are dissimilar.")
```

```
System is not controllable as ranks are dissimilar.
```

Part D(a): LQR Design for a Linear System

In [ ]:
```
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
```

In [ ]:
```
# 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)
```

Matrix A (State Matrix): This matrix represents the system dynamics. It defines how the current state of the system (e.g., position, velocity) affects the future state.

In [ ]:
```
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
```

Out[ ]:
```
A =
array([[ 0.     ,  1.     ,  0.     ,  0.     ,  0.     ,  0.     ],
       [ 0.     ,  0.     , -0.981  ,  0.     , -0.981  ,  0.     ],
       [ 0.     ,  0.     ,  0.     ,  1.     ,  0.     ,  0.     ],
       [ 0.     ,  0.     , -0.53955,  0.     , -0.04905,  0.     ],
       [ 0.     ,  0.     ,  0.     ,  0.     ,  0.     ,  1.     ],
       [ 0.     ,  0.     , -0.0981 ,  0.     , -1.0791 ,  0.     ]])
```

B matrix represents how the input to the system (e.g., control actions) affects its state. It's used along with matrix A in the state equation to describe the effect of the control input on the state.

In [ ]:
```
B = np.array([[0], [1/M], [0], [1/(M*l1)], [0], [1/(M*l2)]])
print("B = ")
B
```

Out[ ]:
```
B =
array([[0.e+00],
       [1.e-03],
       [0.e+00],
       [5.e-05],
       [0.e+00],
       [1.e-04]])
```

Controllability Matrix: It is used to determine whether the system's state can be fully controlled by its inputs. A system is controllable if its controllability matrix has full rank (i.e., the rank is equal to the number of states in the system).

In [ ]:
```
controllability_matrix = np.column_stack([B, A@B, A@A@B, A@A@A@B, A@A@A@A@B, A@A@A@
print ("Controllability matrix = ")
controllability_matrix
```

Out[ ]:
```
Controllability matrix =
array([[ 0.00000000e+00,  1.00000000e-03,  0.00000000e+00,
        -1.47150000e-04,  0.00000000e+00,  1.41948247e-04],
       [ 1.00000000e-03,  0.00000000e+00, -1.47150000e-04,
         0.00000000e+00,  1.41948247e-04,  0.00000000e+00],
       [ 0.00000000e+00,  5.00000000e-05,  0.00000000e+00,
        -3.18825000e-05,  0.00000000e+00,  2.27357786e-05],
       [ 5.00000000e-05,  0.00000000e+00, -3.18825000e-05,
         0.00000000e+00,  2.27357786e-05,  0.00000000e+00],
       [ 0.00000000e+00,  1.00000000e-04,  0.00000000e+00,
        -1.12815000e-04,  0.00000000e+00,  1.24866340e-04],
       [ 1.00000000e-04,  0.00000000e+00, -1.12815000e-04,
         0.00000000e+00,  1.24866340e-04,  0.00000000e+00]])
```

In [ ]:
```
# 3. Check system controllability
if np.linalg.matrix_rank(controllability_matrix) == 6:
    print("System is controllable. Rank = ", np.linalg.matrix_rank(controllability_
else:
    print("System is uncontrollable")
```

```
System is controllable. Rank =  6
```

Matrix C (Output Matrix): This matrix maps the state of the system to the output. It's used in the output equation: y = C *x* + *D* u, where y is the system output. Matrix D (Feedthrough Matrix): This matrix represents the direct effect of the input on the output. In many systems, D is a zero matrix (especially in systems where the input does not directly affect the output).

In [ ]:
```
# 4. Set initial conditions and define output matrix
x_initial = np.array([0, 0, 20, 0, 40, 0])  # Initial state
C = np.eye(6)  # Output matrix
D = np.zeros((6, 1))  # D matrix
```

Out[ ]:
```
C =
array([[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.]])
```

Matrix P (Solution to the Riccati Equation): In the context of LQR (Linear Quadratic Regulator) control, P is the solution to the algebraic Riccati equation. It's used to calculate the optimal state feedback gains.

In [ ]:
```
# 5. Implement and analyze LQR controller
Q = np.diag([100, 100, 1000, 1000, 1000, 1000])
# Redefine R as a 2D array
R = np.array([[0.001]])
# Implement and analyze LQR controller using solve_continuous_are
P = solve_continuous_are(A, B, Q, R)
print("P = ")
P
```

```
P =
```

Out[ ]:
```
array([[ 3.07237373e+02,  4.21974016e+02, -1.59102584e+01,
        -1.00945627e+03,  1.80640060e+01, -5.52734368e+02],
       [ 4.21974016e+02,  1.29624234e+03,  9.60574010e+02,
        -3.13386374e+03,  6.08233745e+02, -1.67979268e+03],
       [-1.59102584e+01,  9.60574010e+02,  2.98194336e+04,
        -4.98734318e+02,  2.65154203e+02, -3.50872508e+03],
       [-1.00945627e+03, -3.13386374e+03, -4.98734318e+02,
         6.09545594e+04,  3.50585105e+03,  3.58231173e+02],
       [ 1.80640060e+01,  6.08233745e+02,  2.65154203e+02,
         3.50585105e+03,  1.69613833e+04, -4.98368458e+02],
       [-5.52734368e+02, -1.67979268e+03, -3.50872508e+03,
         3.58231173e+02, -4.98368458e+02,  1.71900452e+04]])
```

Matrix K (Feedback Gain Matrix): This is the gain matrix used in LQR. It is calculated based on the system matrices (A, B) and the weighting matrices (Q and R) used in the LQR design. The matrix K is used to determine the control action as u = -K * x, aiming to minimize a cost function that balances state error and control effort.

In [ ]:
```
K = np.dot(np.linalg.inv(R), np.dot(B.T, P))
print("K = ")
K
```

Out[ ]:
```
K =
array([[316.22776602, 971.56988044, 584.7647864 , -50.31265473,
        733.68945161,  57.12340249]])
```

In [ ]:
```
# Time array for simulation
t = np.linspace(0, 1000, 1000)
# Zero input signal
u = np.zeros_like(t)
```

In [ ]:
```
# System without controller
sys1 = signal.StateSpace(A, B, C, D)
_, y1, x1 = lsim(sys1, U=u, T=t, X0=x_initial)
```

In [ ]:
```
# System with LQR controller
A_cl = A - np.dot(B, K)
print("A matrix of closed loop system defined by A_cl = A - BK = ")
A_cl
```

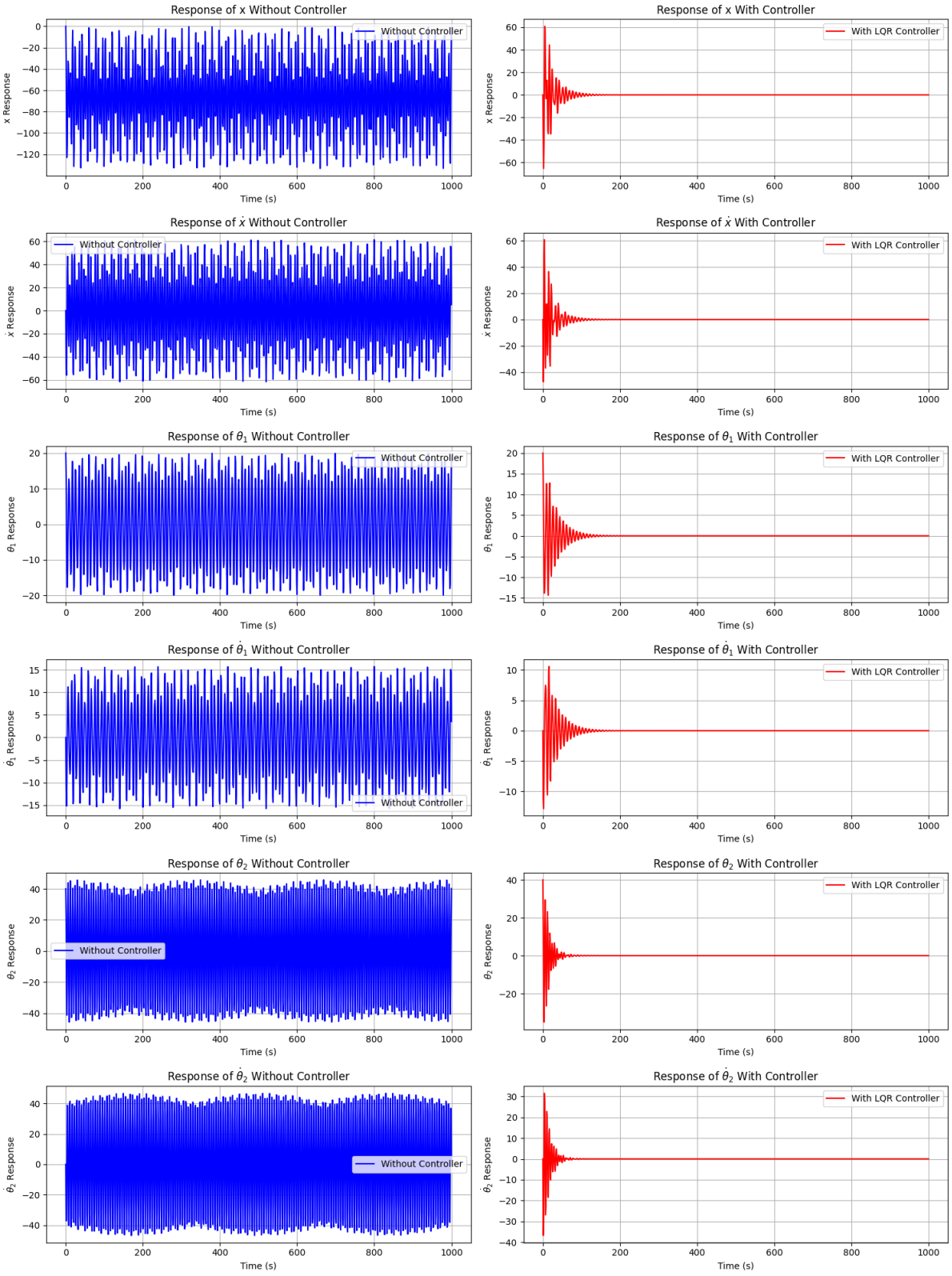
Out[ ]:
```
A matrix of closed loop system defined by A_cl = A - BK =
array([[ 0.        ,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [-0.31622777, -0.97156988, -1.56576479,  0.05031265, -1.71468945,
        -0.0571234 ],
       [ 0.        ,  0.        ,  0.        ,  1.        ,  0.        ,
         0.        ],
       [-0.01581139, -0.04857849, -0.56878824,  0.00251563, -0.08573447,
        -0.00285617],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [-0.03162278, -0.09715699, -0.15657648,  0.00503127, -1.15246895,
        -0.00571234]])
```

In [ ]:
```
closed_loop_eigenvalues = np.linalg.eigvals(A_cl)
print("Closed-Loop Eigenvalues:")
closed_loop_eigenvalues
```

Out[ ]:
```
Closed-Loop Eigenvalues:
array([-0.38230955+0.35462265j, -0.38230955-0.35462265j,
       -0.07230465+1.03416433j, -0.07230465-1.03416433j,
       -0.0327691 +0.72083929j, -0.0327691 -0.72083929j])
```

In [ ]:
```
sys2 = signal.StateSpace(A_cl, B, C, D)
_, y2, x2 = lsim(sys2, U=u, T=t, X0=x_initial)
```

In [ ]:
```
# State variable names
state_vars = ['x', '$\dot{x}$', '$\\theta_1$', '$\dot{\\theta}_1$', '$\\theta_2$',

# Set up the figure and axes for the subplots
fig, axes = plt.subplots(nrows=len(state_vars), ncols=2, figsize=(15, 20))

# Plotting the responses for each state variable
for i in range(len(state_vars)):
    # Response without controller
    axes[i, 0].plot(t, x1[:, i], 'b-', label='Without Controller')
    axes[i, 0].set_title(f'Response of {state_vars[i]} Without Controller')
    axes[i, 0].set_xlabel('Time (s)')
    axes[i, 0].set_ylabel(f'{state_vars[i]} Response')
    axes[i, 0].grid(True)
    axes[i, 0].legend()

    # Response with LQR controller
    axes[i, 1].plot(t, x2[:, i], 'r-', label='With LQR Controller')
    axes[i, 1].set_title(f'Response of {state_vars[i]} With Controller')
    axes[i, 1].set_xlabel('Time (s)')
    axes[i, 1].set_ylabel(f'{state_vars[i]} Response')
    axes[i, 1].grid(True)
    axes[i, 1].legend()

# Adjust layout
plt.tight_layout()
plt.show()
```



Observation: Decreasing the value of 'R' leads to quicker stabilization; however, it results in significantly higher energy consumption.

Part D(b): LQR Design for a Non-Linear System

In [ ]:
```
import numpy as np
from scipy.integrate import solve_ivp
from scipy.linalg import solve_continuous_are
import matplotlib.pyplot as plt
```

In [ ]:
```
# 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]])
```

In [ ]:
```
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
```

Out[ ]:
```
A =
array([[ 0.     ,  1.     ,  0.     ,  0.     ,  0.     ,  0.     ],
       [ 0.     ,  0.     , -0.981  ,  0.     , -0.981  ,  0.     ],
       [ 0.     ,  0.     ,  0.     ,  1.     ,  0.     ,  0.     ],
       [ 0.     ,  0.     , -0.53955,  0.     , -0.04905,  0.     ],
       [ 0.     ,  0.     ,  0.     ,  0.     ,  0.     ,  1.     ],
       [ 0.     ,  0.     , -0.0981 ,  0.     , -1.0791 ,  0.     ]])
```

In [ ]:
```
B = np.array([[0], [1/M], [0], [1/(M*l1)], [0], [1/(M*l2)]])
print("B = ")
B
```

Out[ ]:
```
B =
array([[0.e+00],
       [1.e-03],
       [0.e+00],
       [5.e-05],
       [0.e+00],
       [1.e-04]])
```

In [ ]:
```
# Compute LQR gain matrix K
P = solve_continuous_are(A, B, Q, R)
print("P = ")
P
```

```
P =
```

Out[ ]:
```
array([[   4882.91311849,   11871.42026134,   -5553.68647251,
         -18528.72264325,   -3270.95198186,   -9449.84129178],
        [  11871.42026134,   58866.66912716,   -8589.44588931,
        -101796.64331716,   -6521.93305037,  -49477.05776444],
        [  -5553.68647251,   -8589.44588931,  359821.06683642,
          14921.71671745,   -7970.72928199,    6267.44618594],
        [ -18528.72264325, -101796.64331716,   14921.71671745,
         836482.87294587,   11898.39558792,   44356.34944783],
        [  -3270.95198186,   -6521.93305037,   -7970.72928199,
          11898.39558792,  112410.07030794,    5299.56343381],
        [  -9449.84129178,  -49477.05776444,    6267.44618594,
          44356.34944783,    5299.56343381,  145497.20473458]])
```

In [ ]:
```
K_val = np.dot(np.linalg.inv(R), np.dot(B.T, P))
print("K = ")
K_val
```

Out[ ]:
```
K =
array([[ 100.        ,  488.29131185,  -72.16615435, -555.36864725,
         -53.97056928, -327.09519819]])
```

In [ ]:
```
# Define the double_pendulum_dynamics function
def double_pendulum_dynamics(t, y):
    F = -np.dot(K_val, y)
    dydt = np.zeros(6)
    dydt[0] = y[1]
    dydt[1] = (F - (g/2) * (m1*np.sin(2*y[2]) + m2*np.sin(2*y[4])) - (m1*l1*y[3]**2
    dydt[2] = y[3]
    dydt[3] = (dydt[1]*np.cos(y[2]) - g*np.sin(y[2])) / l1
    dydt[4] = y[5]
    dydt[5] = (dydt[1]*np.cos(y[4]) - g*np.sin(y[4])) / l2
    return dydt
```

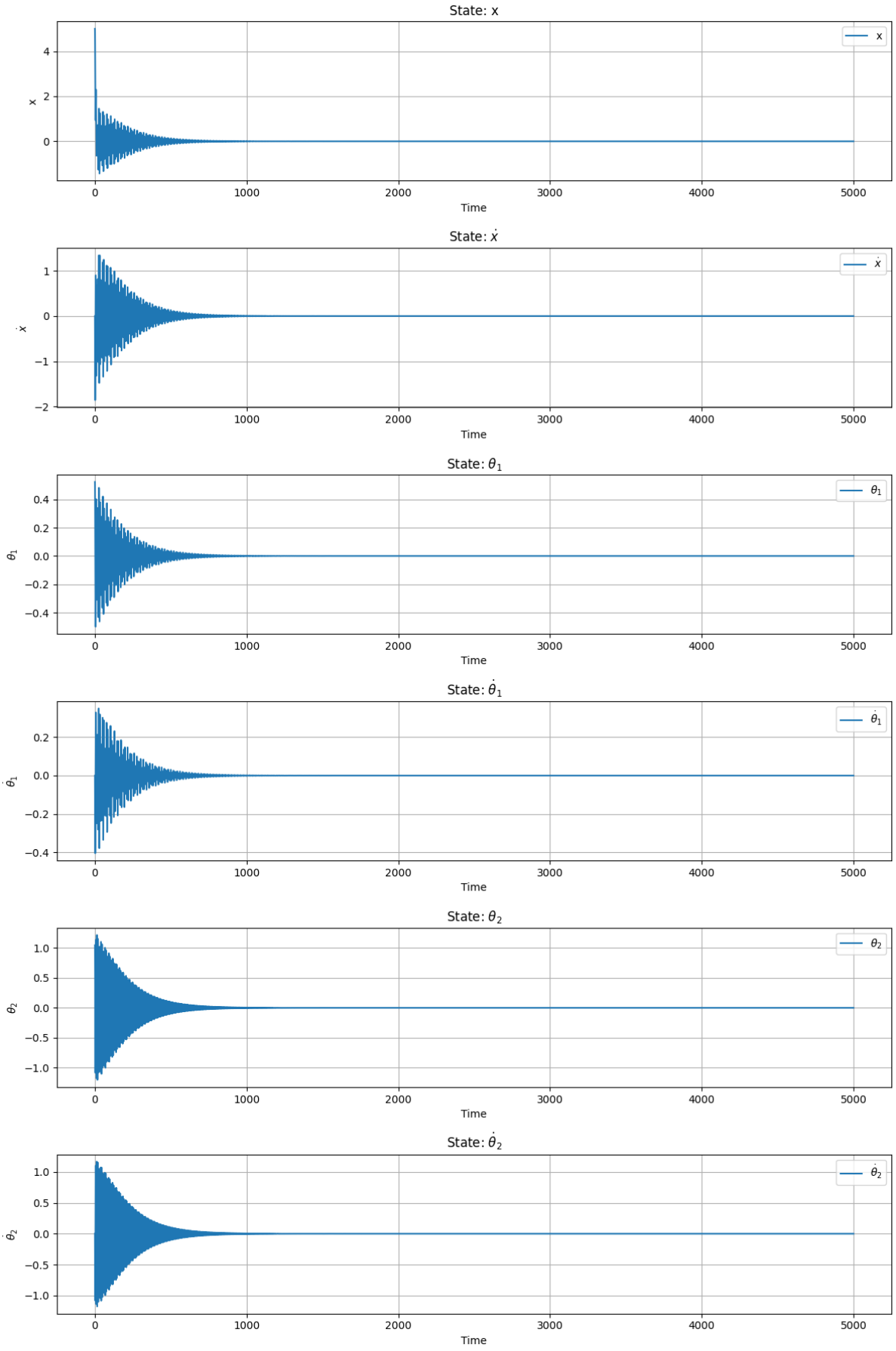
In [ ]:
```
# Initial conditions
y0 = np.array([5, 0, np.radians(30), 0, np.radians(60), 0])
print("Initial conditions = ")
y0
```

Out[ ]:
```
Initial conditions =
array([5.        , 0.        , 0.52359878, 0.        , 1.04719755,
       0.        ])
```

In [ ]:
```
# 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)
```

In [ ]:
```
# Plotting each state variable in a separate subplot
plt.figure(figsize=(12, 18))
state_labels = ['x', '$\dot{x}$', '$\\theta_1$', '$\dot{\\theta}_1$', '$\\theta_2$'

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()
```



Overall, the response of each state variable indicates that the control system is effective in stabilizing the cart and pendulums from an initial disturbance. The plots show the expected behavior of a controlled dynamic system where the effects of an initial disturbance are mitigated over time, leading to a steady state. The time it takes for the oscillations to decay

and the system to stabilize is indicative of the effectiveness of the LQR control strategy implemented.

Part E: Observability Check

In this notebook, we will perform an observability analysis for a system described by its state-space representation. We define observability for various output matrices and determine if the system states can be observed from these outputs.

In [ ]:
```
! pip install control
import numpy as np
import pandas as pd
from IPython.display import HTML
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: control in c:\users\manda\appdata\roaming\python\py
thon311\site-packages (0.9.4)
Requirement already satisfied: numpy in c:\program files\python311\lib\site-packag
es (from control) (1.23.5)
Requirement already satisfied: scipy>=1.3 in c:\users\manda\appdata\roaming\python
\python311\site-packages (from control) (1.10.1)
Requirement already satisfied: matplotlib in c:\users\manda\appdata\roaming\python
\python311\site-packages (from control) (3.6.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\manda\appdata\roaming
\python\python311\site-packages (from matplotlib->control) (1.0.7)
Requirement already satisfied: cycler>=0.10 in c:\users\manda\appdata\roaming\pyth
on\python311\site-packages (from matplotlib->control) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\manda\appdata\roaming
\python\python311\site-packages (from matplotlib->control) (4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\manda\appdata\roaming
\python\python311\site-packages (from matplotlib->control) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\manda\appdata\roaming\p
ython\python311\site-packages (from matplotlib->control) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\manda\appdata\roaming\pyt
hon\python311\site-packages (from matplotlib->control) (9.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\manda\appdata\roaming
\python\python311\site-packages (from matplotlib->control) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\manda\appdata\roam
ing\python\python311\site-packages (from matplotlib->control) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\manda\appdata\roaming\python\p
ython311\site-packages (from python-dateutil>=2.7->matplotlib->control) (1.16.0)
[notice] A new release of pip is available: 23.1.2 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [ ]:
```
def observability_matrix(A, C):
    """
    Compute the observability matrix for the system (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
```

In [ ]:
```
# Define system parameters
M = 1000
mass_1 = 100
mass_2 = 100
length_1 = 20
length_2 = 10
g = 9.81
```

```
# Define state space matrices A and B with the assigned values
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]
])
print("A: ")
A
```

Out[ ]:
```
A:
array([[ 0.     ,  1.     ,  0.     ,  0.     ,  0.     ,  0.     ],
       [ 0.     ,  0.     , -0.981  ,  0.     , -0.981  ,  0.     ],
       [ 0.     ,  0.     ,  0.     ,  1.     ,  0.     ,  0.     ],
       [ 0.     ,  0.     , -0.53955,  0.     , -0.04905,  0.     ],
       [ 0.     ,  0.     ,  0.     ,  0.     ,  0.     ,  1.     ],
       [ 0.     ,  0.     , -0.0981 ,  0.     , -1.0791 ,  0.     ]])
```

In [ ]:
```
# Defining the B matrix
B = np.array([[0], [1/M], [0], [1/(M*length_1)], [0], [1/(M*length_2)]])
print("B: ")
B
```

Out[ ]:
```
B:
array([[0.e+00],
       [1.e-03],
       [0.e+00],
       [5.e-05],
       [0.e+00],
       [1.e-04]])
```

Defining Output Matrices

We define different output matrices `C` to observe various combinations of the system's states. The observability of the system is determined for each case.

In [ ]:
```
# Define different C matrices for various outputs
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
}
```

In [ ]:
```
# Prepare data for the table
table_data = []

# Checking the observability for each C matrix and collecting data for the table
for output, C_matrix in C_matrices.items():
    O_matrix = observability_matrix(A, C_matrix)
    rank = np.linalg.matrix_rank(O_matrix)
    observable = "Yes" if rank == 6 else "No"
    matrix_string = '[' + '; '.join(' '.join(str(int(num)) for num in row) for row
    table_data.append({
        'Outputs': output,
        'C Matrix': matrix_string,
        'Rank': rank,
        'Observable': observable
    })
```

```
# Create a pandas DataFrame
df = pd.DataFrame(table_data)
# Convert the DataFrame to HTML and allow raw HTML and LaTeX rendering
html = df.to_html(escape=False, index=False)
display(HTML(html))
```

Outputs	C Matrix	Rank	Observable
x	[1 0 0 0 0 0]	6	Yes
theta_1 and theta_2	[0 0 1 0 0 0; 0 0 0 1 0 0]	4	No
x and theta_2	[1 0 0 0 0 0; 0 0 0 0 1 0]	6	Yes
x, theta_1 and theta_2	[1 0 0 0 0 0; 1 0 1 0 0 0; 0 0 0 0 1 0]	6	Yes

Part F(a) - Luenberger Observer for Linear Systems

In [ ]:
```
import numpy as np
import control as ctrl
import matplotlib.pyplot as plt
```

In [ ]:
```
# 1. Define system parameters
M = 1000
mass_1 = 100
mass_2 = 100
length_1 = 20
length_2 = 10
g = 9.81
```

In [ ]:
```
# Constructing state space matrices
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))
```

In [ ]:
```
# Setting LQR parameters
Q = np.diag([1000, 100, 1000, 1000, 100, 100])
R = np.array([[0.01]])
K, S, eigen_values = ctrl.lqr(A, B, Q, R)
```

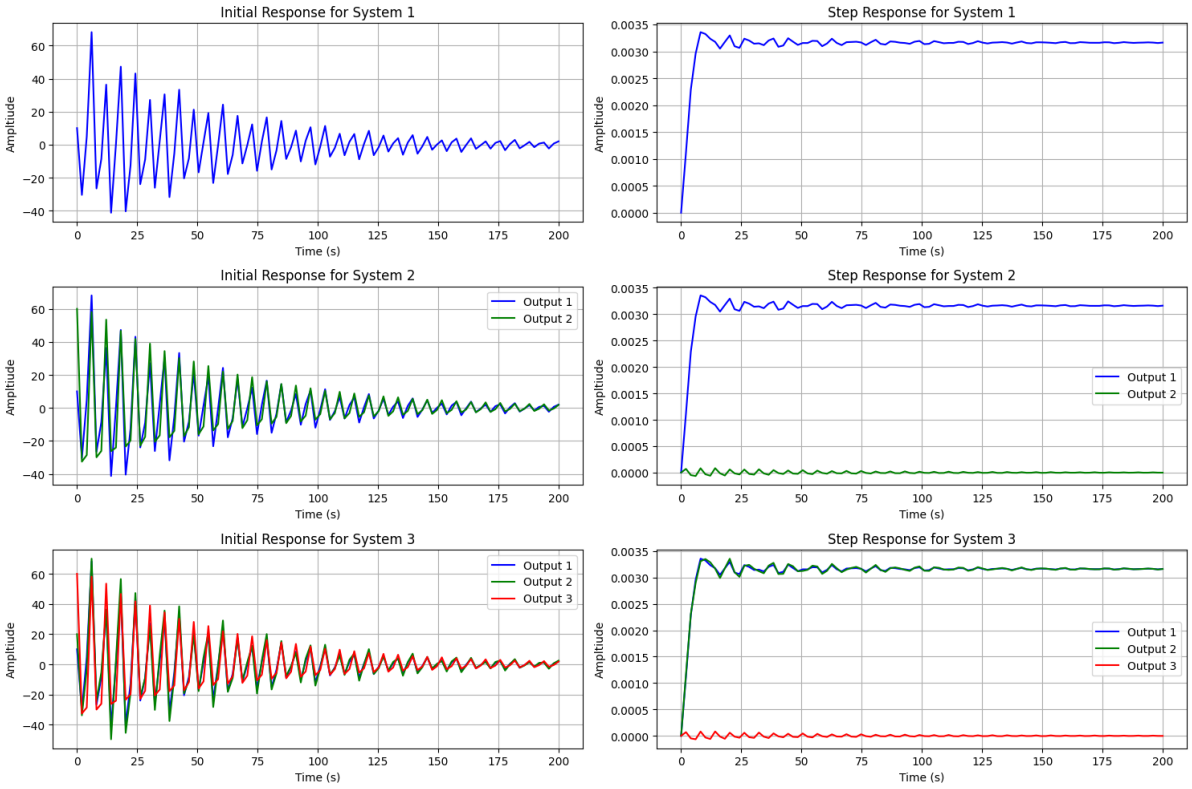
In [ ]:
```
# Define observer output matrices and corresponding L matrices
C_matrices = [np.array([[1, 0, 0, 0, 0, 0]]),  # Observing x component
              np.array([[1, 0, 0, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0]]), # Observing x and theta2
              np.array([[1, 0, 0, 0, 0,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

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

In [ ]:
```
# Setting initial conditions for the observer
initial_conditions = [10, 40, 10, 0, 60, 0, 0, 0, 0, 0, 0, 0]
```

In [ ]:
```
# Assign a color for each system
colors = ['b', 'g', 'r'] # Colors for plotting
# Construct and analyze each system
plt.figure(figsize=(15, 10))

for i, (C, L) in enumerate(zip(C_matrices, L_matrices)):
    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 = np.zeros((C.shape[0], B.shape[1]))
    sys = ctrl.ss(A_q, B_q, C_q, D)

    # Plot initial response of the system
```

```
    plt.subplot(len(C_matrices), 2, 2*i + 1)
    T, yout = ctrl.initial_response(sys, T=np.linspace(0, 200, 100), X0=initial_cor
    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.legend()
    plt.title(f'Initial Response for System {i+1}')
    plt.xlabel('Time (s)')
    plt.ylabel('Ampltiude')
    plt.grid(True)

    # Plot step response of the system
    plt.subplot(len(C_matrices), 2, 2*i + 2)
    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
    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('Ampltiude')
    plt.grid(True)

plt.tight_layout()
plt.show()
```



Part F(b) - Luenberger Observer for Non-Linear Systems

In [ ]:
```
import numpy as np
import control as ctrl
import matplotlib.pyplot as plt
```

In [ ]:
```
# 1. Define system parameters
M = 1000
mass_1 = 100
mass_2 = 100
length_1 = 20
length_2 = 10
g = 9.81
```

In [ ]:
```
# Constructing state space matrices
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))
```

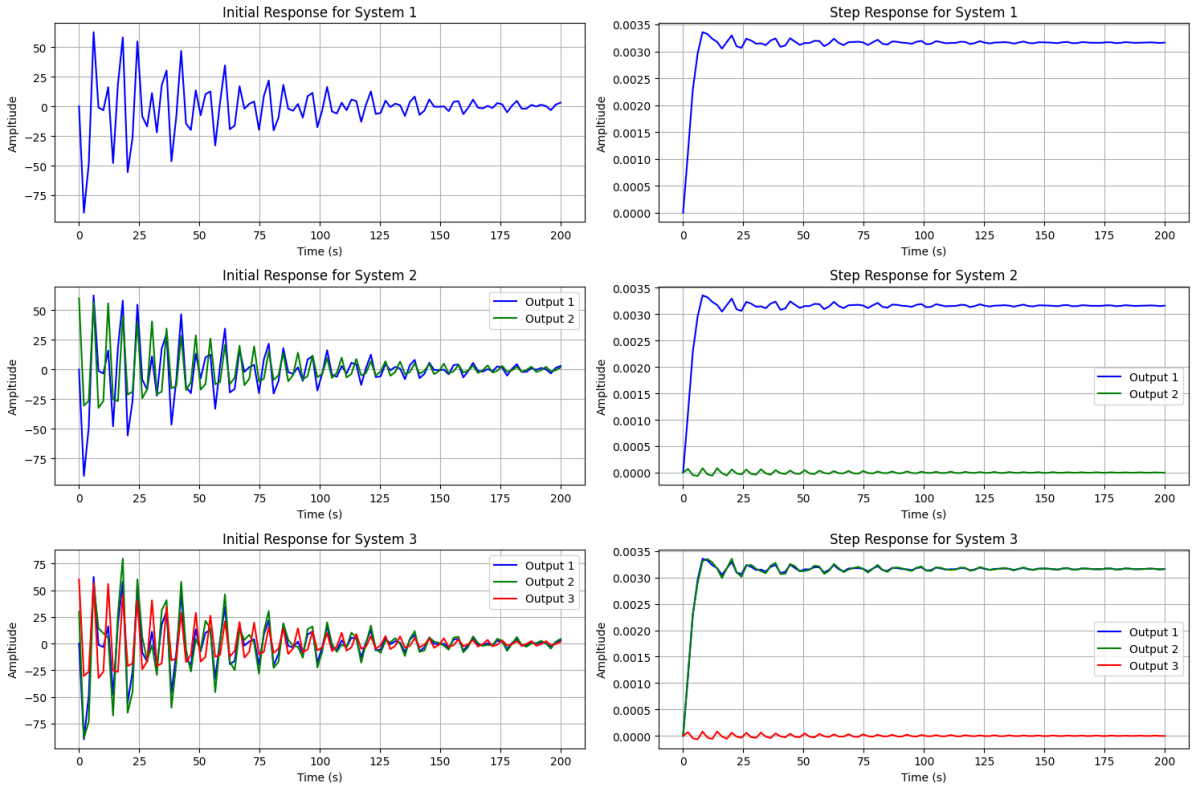
In [ ]:
```
# Define observer output matrices and corresponding L matrices
C_matrices = [np.array([[1, 0, 0, 0, 0, 0]]),  # Observing x component
              np.array([[1, 0, 0, 0, 0, 0],
                        [0, 0, 0, 0, 1, 0]]), # Observing x and theta2
              np.array([[1, 0, 0, 0, 0,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

In [ ]:
```
# Setting LQR parameters
Q = np.diag([1000, 100, 1000, 1000, 100, 100])
R = np.array([[0.01]])
# Solving LQR for gain matrix K
K, _, _ = ctrl.lqr(A, B, Q, R)
```

In [ ]:
```
# Initial conditions for the observer
initial_conditions = [0, 0, 30, 0, 60, 0, 0, 0, 0, 0, 0, 0]

# Desired poles for the observer
poles = [-1, -2, -3, -4, -5, -6]
```

In [ ]:
```
# Assign a color for each system
colors = ['b', 'g', 'r'] # Colors for plotting

# Creating figure for plots
plt.figure(figsize=(15, 10))

# Constructing and analyzing systems for each observer configuration
for i, C in enumerate(C_matrices):
    L = ctrl.place(A.T, C.T, poles).T
    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]))
    sys = ctrl.ss(A_q, B_q, C_q, D)

    # Plotting initial response (Left column)
    plt.subplot(len(C_matrices), 2, 2*i + 1)
    T, yout = ctrl.initial_response(sys, T=np.linspace(0, 200, 100), X0=initial_cor
    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.legend()
    plt.title(f'Initial Response for System {i+1}')
    plt.xlabel('Time (s)')
    plt.ylabel('Ampltiude')
    plt.grid(True)

    # Plotting step response (Right column)
    plt.subplot(len(C_matrices), 2, 2*i + 2)
    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
    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('Ampltiude')
    plt.grid(True)

plt.tight_layout()
plt.show()
```



Part G(a): LQG Design for a Linear System

In [ ]:
```
import numpy as np
import control as ctrl
import matplotlib.pyplot as plt
```

In [ ]:
```
# 1. Define system parameters
M = 1000
mass_1 = 100
mass_2 = 100
length_1 = 20
length_2 = 10
g = 9.81
```

In [ ]:
```
# Constructing state space matrices
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))
```

In [ ]:
```
# 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]])]
```

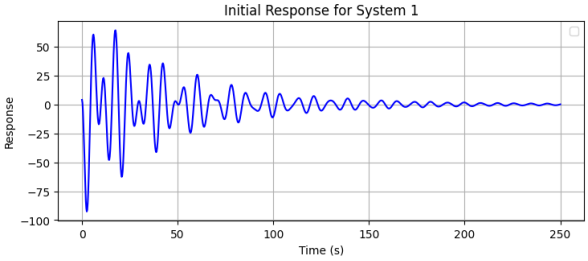
In [ ]:
```
# Defining LQR and Kalman filter parameters
Q = np.diag([100, 100, 100, 100, 100, 100])
R = np.array([[0.001]])
# Designing LQR controller
K, S, eigen_values = ctrl.lqr(A, B, Q, R)

# Defining noise covariance matrices for Kalman filter
v_d = 0.3 * np.eye(6)
v_n = 1
```

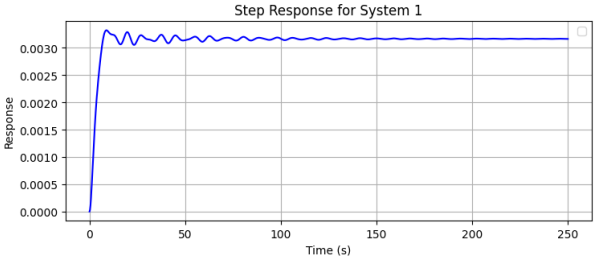
In [ ]:
```
# Setting initial conditions for the observer
initial_conditions = np.array([4, 0, 30, 0, 60, 0, 0, 0, 0, 0, 0, 0])
```

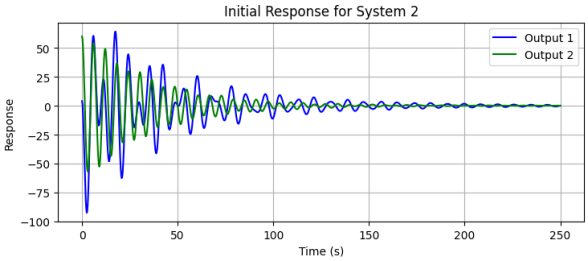
In [ ]:
```
# Assign a color for each system
colors = ['b', 'g', 'r'] # Colors for plotting

# Creating figure for plots
plt.figure(figsize=(15, 10))

# Analyzing systems for each observable configuration
for i, C in enumerate(C_matrices):
    # Adjusting the dimension of R for Kalman filter design
    R_kalman = v_n * np.eye(C.shape[0])

    # Designing Kalman filter
    K_pop = ctrl.lqr(A.T, C.T, v_d, R_kalman)[0].T
```

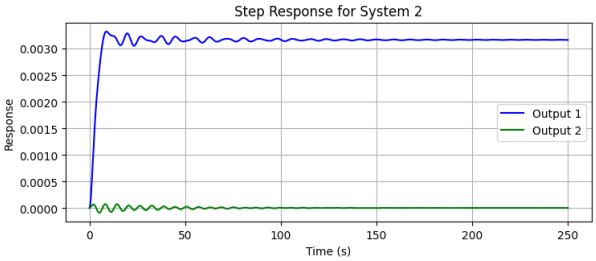
```
    # Constructing state-space model for LQG
    sys = ctrl.ss(np.block([[A - B @ K, B @ K],
                            [np.zeros_like(A), A - K_pop @ C]]),
                  np.block([[B], [np.zeros_like(B)]]),
                  np.block([C, np.zeros_like(C)]),
                  np.zeros((C.shape[0], B.shape[1])))

    # Plotting 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_cc
    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)

    # Plotting 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()
```

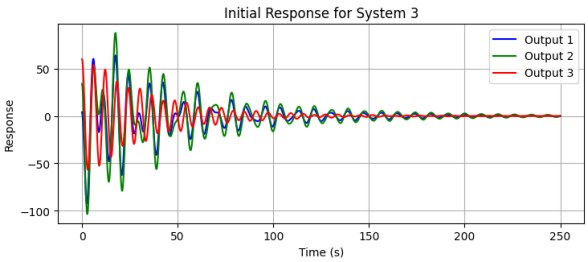
```
No artists with labels found to put in legend.  Note that artists whose label star
t with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label star
t with an underscore are ignored when legend() is called with no argument.
```

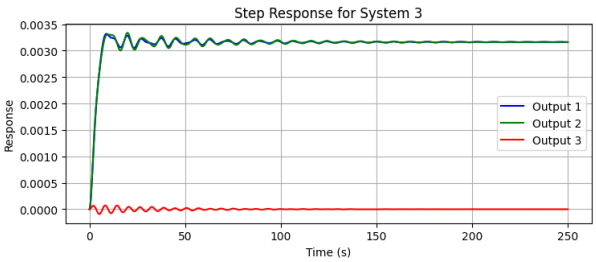

Initial Response for System 1
Response
Time (s)


Step Response for System 1
Response
Time (s)


Initial Response for System 2
Output 1
Output 2
Response
Time (s)


Step Response for System 2
Output 1
Output 2
Response
Time (s)


Initial Response for System 3
Output 1
Output 2
Output 3
Response
Time (s)


Step Response for System 3
Output 1
Output 2
Output 3
Response
Time (s)

Part G(b): LQG Design for a Non-Linear System

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import control as ctrl
```

In [ ]:
```
# Defining the system parameters
M = 1000
mass_1 = 100
mass_2 = 100
length_1 = 20
length_2 = 10
g = 9.81
```

In [ ]:
```
# Constructing state space matrices
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
```

In [ ]:
```
# Designing LQR parameters
Q = np.diag([1000, 1000, 100, 10, 1000, 100])
R = 0.01
K, _, _ = ctrl.lqr(A, B, Q, R)
```

In [ ]:
```
# Designing Kalman filter
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
```

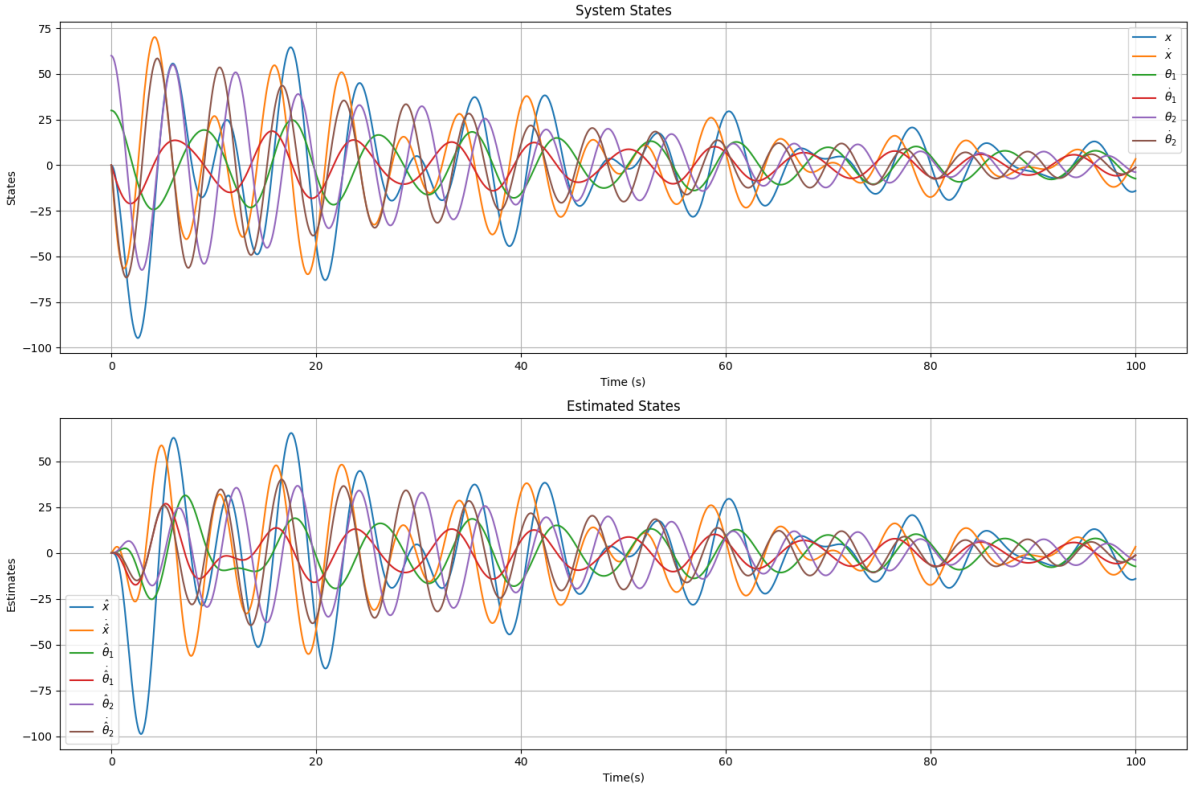
In [ ]:
```
# Defining the function for nonlinear LQG control
def lqg(y, t):
    # Control input
    controlled_force = -K @ y[:6]
    # Estimator dynamics
    dx_hat = A @ y[6:12] + B @ controlled_force + K_pop @ (C_matrix @ y[:6] - C_mat
    # System dynamics (controlled system)
    dx = A @ y[:6] + B @ controlled_force
    return np.concatenate((dx, dx_hat))
```

In [ ]:
```
# Setting 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)
```

In [ ]:
```
# Solving the system using the odeint solver
x = odeint(lqg, initial_conditions, time_span)
```

In [ ]:
```
# Creating figure for plots
plt.figure(figsize=(15, 10))
```

```
# Upper subplot for the system states
plt.subplot(2, 1, 1)
plt.plot(time_span, x[:, :6])
plt.title('System States')
plt.xlabel('Time (s)')
plt.ylabel('States')
plt.legend(['$x$', '$\\dot{x}$', '$\\theta_1$', '$\\dot{\\theta}_1$', '$\\theta_2$',
plt.grid(True)

# Lower subplot for the estimated states
plt.subplot(2, 1, 2)
plt.plot(time_span, x[:, 6:12])
plt.title('Estimated States')
plt.xlabel('Time(s)')
plt.ylabel('Estimates')
plt.legend(['$\hat{x}$', '$\dot{\hat{x}}$', '$\hat{\\theta}_1$', '$\dot{\hat{\\thet
plt.grid(True)

plt.tight_layout()
plt.show()
```



System States (Upper Plot):

- This plot shows six different system states over time, with time represented on the x-axis and the magnitude of the states on the y-axis.
- The states are oscillatory in nature, with no clear pattern of convergence or divergence, suggesting a dynamic system that is undergoing periodic motion or fluctuations.
- All states seem to have a similar frequency but varying amplitudes, which might indicate some form of coupling or interaction between them.

Estimated States (Lower Plot):

- Similar to the system states, the estimated states also exhibit an oscillatory behavior.
- The plot suggests that the estimator is tracking the system states with a similar pattern of oscillation.

- The amplitude and frequency of the estimated states seem to match closely with the actual system states, which imply a good performance of the estimation process.