UNIVERSITY OF MARYLAND

ENPM 667 : Control of Robotic Systems

Project -2 Report

Authors

Hitesh Kyatham - 120275364

Gautam Sreenarayanan Nair - 119543092

Submitted on: 12 - 20 - 2024

Contents

1 Component - 1

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 m1 and m2, and the lengths of the cables are l1 and l2, respectively. The following figure depicts the crane and associated variables used throughout this project.

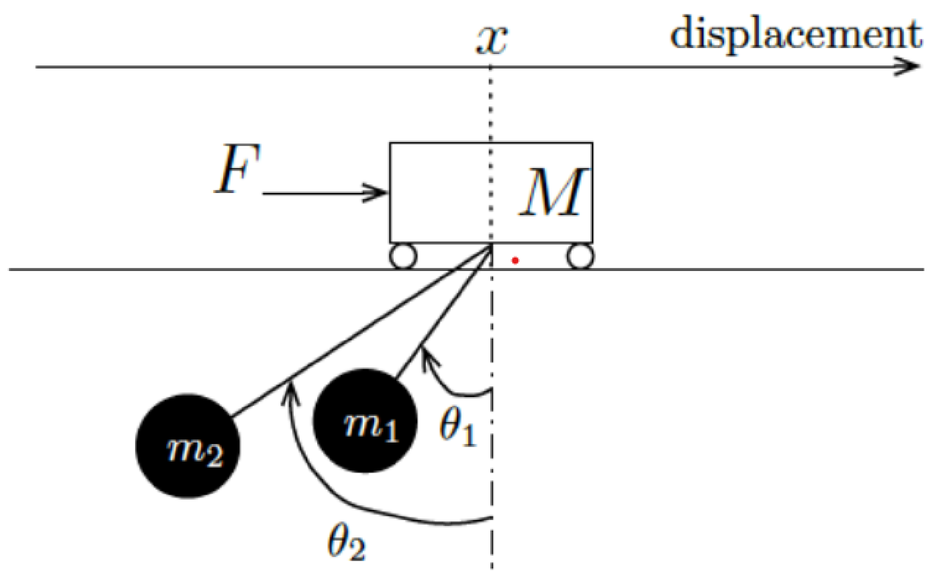


Figure 1: Two Pendulums on a Cart

The Figure 1 is a diagrammatic representation of the setup.

1.1 Equations of Motion

A) (25 points) Obtain the equations of motion for the system and the corresponding nonlinear state-space representation. For the purpose of solving this problem, let's assume that the mass of each pendulums are concentrated at the center of the loads and that the mass of cables are negligible. Below given are the list of variables used in this problem.

List of variables

- M: Mass of the cart
- m_1: Mass of pendulum-1

- m_2: Mass of pendulum-2
- l_1: Length of pendulum-1
- l_2: Length of pendulum-2
- (x, y): Reference position of the cart
- $\dot{x}$: Velocity of the cart
- $\dot{x}_1$: Horizontal velocity of the first pendulum bob
- $\dot{y}_1$: Vertical velocity of pendulum-1
- $\dot{x}_2$: Horizontal velocity pendulum-2
- $\dot{y}_2$: Vertical velocity of pendulum-1
- θ_1: Angle of pendulum-1 with respect to the vertical axis from (x,y)
- θ_2: Angle of pendulum-2 with respect to the vertical axis from (x,y)
- $\dot{\theta}_1$: Angular velocity of pendulum-1
- $\dot{\theta}_2$: Angular velocity of pendulum-1
- g: Gravitational acceleration constant

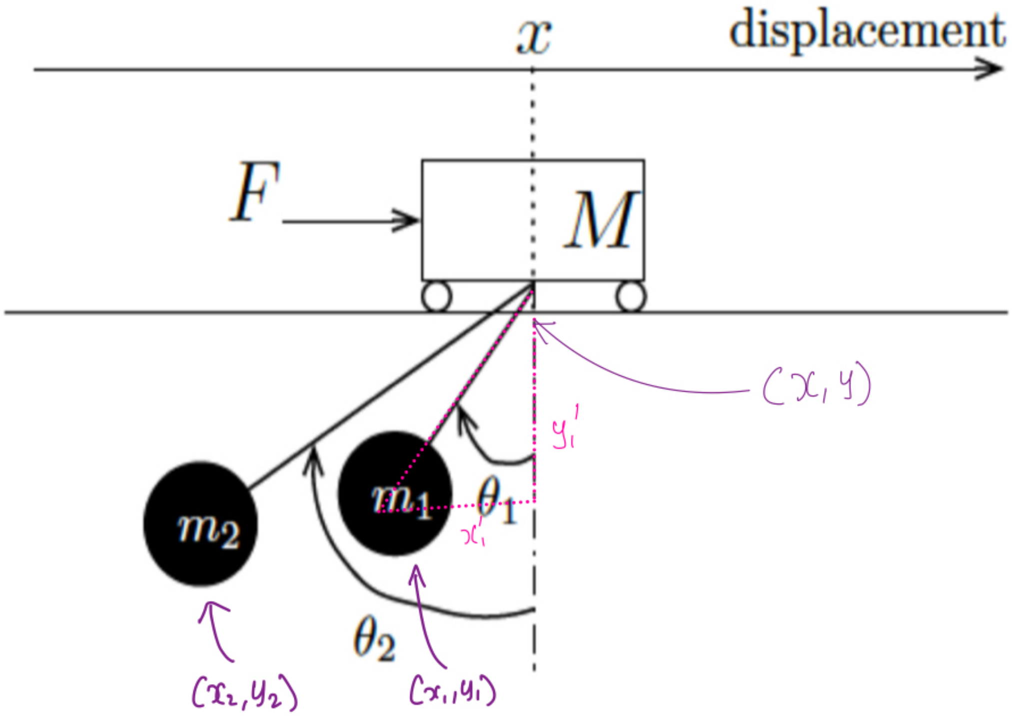


Figure 2: Two Pendulums with positions

It is to be noted that here after when a pendulum is referred it also implies the pendulum bob. Refer the Figure 2;

$$\begin{aligned} x_1' &= l_1 \sin(\theta_1) \\ x_2' &= l_2 \sin(\theta_2) \\ y_1' &= l_1 - l_1 \cos(\theta_1) \\ y_2' &= l_2 - l_2 \cos(\theta_2) \end{aligned} \tag{1}$$

$$\begin{aligned} x_1 &= x - x_1' = x - l_1 \sin(\theta_1) \\ x_2 &= x - x_2' = x - l_2 \sin(\theta_2) \\ y_1 &= y - y_1' = y - l_1(1 - \cos(\theta_1)) \\ y_2 &= y - y_2' = y - l_2(1 - \cos(\theta_2)) \end{aligned} \tag{2}$$

The equation (2) can be furthered simplified into a general expression:

$$\begin{aligned}
&x_i = x - l_i \sin(\theta_i) \\
&y_i = y - l_i(1 - \cos(\theta_i)) \\
&\quad \text{where } i = 1, 2, \ldots \text{ stands for pendulum number}
\end{aligned} \tag{3}$$

We get the velocities when we differentiate equation (2) with respect to time.

$$\begin{aligned}
\dot{x}_1 &= \frac{d}{dt}\left(x - l_1 \sin(\theta_1)\right) = \dot{x} - l_1\dot{\theta}_1 \cos(\theta_1) \\
\dot{x}_2 &= \frac{d}{dt}\left(x - l_2 \sin(\theta_2)\right) = \dot{x} - l_2\dot{\theta}_2 \cos(\theta_2) \\
\dot{y}_1 &= \frac{d}{dt}\left(y - l_1(1 - \cos(\theta_1))\right) = \dot{y} + l_1\dot{\theta}_1 \sin(\theta_1) \\
\dot{y}_2 &= \frac{d}{dt}\left(y - l_2(1 - \cos(\theta_2))\right) = \dot{y} + l_2\dot{\theta}_2 \sin(\theta_2)
\end{aligned} \tag{4}$$

The kinetic energy of the cart is given by:

$$T_{\text{cart}} = \frac{1}{2} M \dot{x}^2 \tag{5}$$

The kinetic energy of the pendulums are:

$$T_1 = \frac{1}{2} m_1 \left[\dot{x}_1^2 + \dot{y}_1^2\right] T_2 = \frac{1}{2} m_2 \left[\dot{x}_2^2 + \dot{y}_2^2\right] \tag{6}$$

The total kinetic energy of the system is the sum of the kinetic energies of the cart, pendulum-1, and pendulum-2 from equations (5) and (6).

$$T = T_{\text{cart}} + T_1 + T_2$$

Utilizing (5) and (6) we get,

$$T = \frac{1}{2} M \dot{x}^2 + \frac{1}{2} m_1 \left[\dot{x}_1^2 + \dot{y}_1^2\right] + \frac{1}{2} m_2 \left[\dot{x}_2^2 + \dot{y}_2^2\right] \tag{7}$$

Utilizing $\dot{x}_1$, $\dot{y}_1$, $\dot{x}_2$, and $\dot{y}_2$ from equation (4) in equation (7), we get:

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

This can be rearranged to give;

$$T = \frac{1}{2}(M + m_1 + m_2)\dot{x}^2 + \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2 - m_1 l_1\cos(\theta_1)\dot{\theta}_1\dot{x} - m_2 l_2\cos(\theta_2)\dot{\theta}_2\dot{x} \tag{8}$$

Now let's calculate the potential energy of the system. The potential energy of the system is the sum of potential energies of individual pendulums. The general equation of potential energy is:

$$V = \text{mass} \times \text{gravity} \times \text{height}$$

Thus, the potential energy of the pendulums are:

$$\begin{aligned} V_1 &= m_1 g l_1(1 - \cos(\theta_1)) \\ V_2 &= m_2 g l_2(1 - \cos(\theta_2)) \end{aligned} \tag{9}$$

Therefore, the total potential energy of the system is:

$$V = V_1 + V_2 = m_1 g l_1(1 - \cos(\theta_1)) + m_2 g l_2(1 - \cos(\theta_2)) \tag{10}$$

In case of non-linear dynamics, Euler-Lagrange equations are used to find the dynamics equations of the system. The Lagrangian for the system is L = T-V. Which can be express as

$$\begin{aligned} L = &\frac{1}{2}(M + m_1 + m_2)\dot{x}^2 + \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2 - m_1 l_1\cos(\theta_1)\dot{\theta}_1\dot{x} - m_2 l_2\cos(\theta_2)\dot{\theta}_2\dot{x} \\ &- \left(m_1 g l_1(1 - \cos(\theta_1)) + m_2 g l_2(1 - \cos(\theta_2))\right). \end{aligned} \tag{11}$$

The generalized momentum equation of Euler-Lagrangian is given as [1];

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

where q_i are the generalized coordinates (e.g., x, θ_1, θ_2 as in our case).

Thus the Lagrangian Equations capturing the non linear dynamics of this system are;

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = F \tag{12}$$

where F is the external force acting on the cart.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) - \frac{\partial L}{\partial \theta_1} = 0 \tag{13}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) - \frac{\partial L}{\partial \theta_2} = 0. \tag{14}$$

Now let's compute each term in equation (12).

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

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

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

Substituting these values into into the Euler-Lagrange equation for x in equation (12), we get:

$$(M + m_1 + m_2)\ddot{x} + m_1 l_1 \sin(\theta_1)\dot{\theta}_1^2 + m_2 l_2 \sin(\theta_2)\dot{\theta}_2^2 - m_1 l_1 \cos(\theta_1)\ddot{\theta}_1 - m_2 l_2 \cos(\theta_2)\ddot{\theta}_2 = F. \tag{15}$$

Now lets compute the individual term in equation (13).

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

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

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

Substituting these values into the Euler-Lagrange equation for θ_1 in equation (13), we get:

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

Which can be further simplified to;

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

The same can be repeated for θ_2 from equation (14) and we get;

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

The above equations can be further rearranged to get the following equations using which we can form the state space representation.

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

$$\ddot{\theta}_1 = \frac{1}{l_1}\left[\cos(\theta_1)\ddot{x} - g\sin(\theta_1)\right]. \tag{17}$$

$$\ddot{\theta}_2 = \frac{1}{l_2}\left[\cos(\theta_2)\ddot{x} - g\sin(\theta_2)\right]. \tag{18}$$

Using the equations (16), (17) and (18) we can get the state space representation;

The state vector X would be: $X = \begin{bmatrix} \dot{x} \\ \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{bmatrix}$, and its time derivative $\dot{X}$ would be: $\dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{bmatrix}$.

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

Since F is the external force applied. F would be the input and it can be taken out as input (u) in standard state space representation, Thus we get the state space representation.

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{bmatrix} = &\begin{bmatrix} \dot{x} \\ \frac{- m_1 l_1 \sin(\theta_1)\dot{\theta_1}^2 - m_2 l_2 \sin(\theta_2)\dot{\theta_2}^2 - m_1 g \cos(\theta_1)\sin(\theta_1) - m_2 g \cos(\theta_2)\sin(\theta_2)}{M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2)} \\ \dot{\theta}_1 \\ \frac{\cos(\theta_1)\left(- m_1 l_1 \sin(\theta_1)\dot{\theta_1}^2 - m_2 l_2 \sin(\theta_2)\dot{\theta_2}^2 - m_1 g \cos(\theta_1)\sin(\theta_1) - m_2 g \cos(\theta_2)\sin(\theta_2)\right)}{l_1(M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2))} - \frac{g \sin(\theta_1)}{l_1} \\ \dot{\theta}_2 \\ \frac{\cos(\theta_2)\left(- m_1 l_1 \sin(\theta_1)\dot{\theta_1}^2 - m_2 l_2 \sin(\theta_2)\dot{\theta_2}^2 - m_1 g \cos(\theta_1)\sin(\theta_1) - m_2 g \cos(\theta_2)\sin(\theta_2)\right)}{l_2(M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2))} - \frac{g \sin(\theta_2)}{l_2} \end{bmatrix} \\ &+ \begin{bmatrix} 0 \\ \frac{1}{M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2)} \\ 0 \\ \frac{F \cos(\theta_1)}{l_1(M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2))} \\ 0 \\ \frac{F \cos(\theta_2)}{l_2(M + m_1 \sin^2(\theta_1) + m_2 \sin^2(\theta_2))} \end{bmatrix} \cdot F \end{aligned} \tag{19}$$

1.2 Linearized System

B) (25 points) 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

To linearize the system, non-linearities in the state equation have to be removed. Here the non-linearities are due to trigonometric components (sine and cosine). The equilibrium point is specified by $x = 0$, $\theta_1 = 0$, and $\theta_2 = 0$. The system has to be linearized around the equilibrium and nearby points. At this point, we can assume that:

$$\sin(\theta_1) \approx \theta_1,\ \sin(\theta_2) \approx \theta_2,\ \cos(\theta_1) \approx 1,\ \cos(\theta_2) \approx 1$$

As the angles are $\theta_1 = \theta_2 = 0$ (or are close to zero near the equilibrium point), the square of these angles and their differentiations can also be assumed to be equal to zero. That is:

$$\theta_1^2 \approx 0, \theta_2^2 \approx 0, \dot{\theta}_1^2 \approx 0, \dot{\theta}_2^2 \approx 0$$

Using these approximations in the equations (16), we get,

$$\ddot{x} = \frac{1}{M}\Bigg[F - m_1 g\theta_1 - m_2 g\theta_2\Bigg]. \tag{20}$$

Using the same in equation (17), we get,

$$\ddot{\theta}_1 = \frac{1}{l_1}\left[\ddot{x} - g\theta_1\right].$$

Substituting equation (20) in the above equation,

$$\ddot{\theta}_1 = \frac{1}{l_1}\left[\frac{1}{M}\left(F - m_1 g\theta_1 - m_2 g\theta_2 - g\theta_1\right)\right].$$

Which can be rewritten as ;

$$\ddot{\theta}_1 = \frac{F - m_1 g\theta_1 - m_2 g\theta_2 - Mg\theta_1}{Ml_1} \tag{21}$$

As equation (18) is analogous to equation (17), we get;

$$\ddot{\theta}_2 = \frac{F - m_1 g\theta_1 - m_2 g\theta_2 - Mg\theta_2}{Ml_2} \tag{22}$$

The equations (20), (21) and (22) can be used in the original state equations to get linearized state equations as given below.

$$\dot{x} = \dot{x}$$

$$\ddot{x} = \frac{1}{M}\left(F - m_1 g\theta_1 - m_2 g\theta_2\right)$$

$$\dot{\theta}_1 = \dot{\theta}_1$$

$$\ddot{\theta}_1 = \frac{1}{Ml_1}\left(F - m_1 g\theta_1 - m_2 g\theta_2 - Mg\theta_1\right)$$

$$\dot{\theta}_2 = \dot{\theta}_2$$

$$\ddot{\theta}_2 = \frac{1}{Ml_2}\left(F - m_1 g\theta_1 - m_2 g\theta_2 - Mg\theta_2\right)$$

Using the above state equations the state space representation of the linearized system can be written as;

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\theta}_1\\ \ddot{\theta}_1\\ \dot{\theta}_2\\ \ddot{\theta}_2\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & \frac{-m_1 g}{M} & 0 & \frac{-m_2 g}{M} & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & \frac{-m_1 g - Mg}{Ml_1} & 0 & \frac{-m_2 g}{Ml_1} & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & \frac{-m_1 g}{Ml_2} & 0 & \frac{-m_2 g - Mg}{Ml_2} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \theta_1\\ \dot{\theta}_1\\ \theta_2\\ \dot{\theta}_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{M}\\ 0\\ \frac{1}{Ml_1}\\ 0\\ \frac{1}{Ml_2}\end{bmatrix}F \tag{23}$$

This is of the standard Linear Time Invariant (LTI) system as generally expressed as

$$\dot{X} = AX + BU$$

where U = F, in this case

1.3 Conditions for Controllability

C) (25 points) Obtain conditions on M, m1, m2, l1, l2 for which the linearized system is controllable. Since the system is a Linear Time Invariant system the rank test for controllability can be used to check if the system is controllable. Here the rank of controllability matrix (C) is calculated and if the rank is equal to the number of states (n), then the system is controllable.

The controllability matrix is given by;

$$\mathcal{C} = \begin{bmatrix}B & AB & A^2B & \cdots & A^{n-1}B\end{bmatrix}$$

The same was inputted in MATLAB code which is available in the Appendix Section. From Matlab we got the controllability matrix as given below.

$$C = \begin{bmatrix} 0 & \frac{1}{M} & 0 & -\frac{g(l_1m_2+l_2m_1)}{M^2l_1l_2} & 0 & \frac{g^2(l_1^2m_2^2+Ml_1^2m_2+2l_1l_2m_1m_2+l_2^2m_1^2+Ml_2^2m_1)}{M^3l_1^2l_2^2}\\ \frac{1}{M} & 0 & -\frac{g(l_1m_2+l_2m_1)}{M^2l_1l_2} & 0 & \frac{g^2(l_1^2m_2^2+Ml_1^2m_2+2l_1l_2m_1m_2+l_2^2m_1^2+Ml_2^2m_1)}{M^3l_1^2l_2^2} & 0\\ 0 & \frac{1}{Ml_1} & 0 & -\frac{g(Ml_2+l_1m_2+l_2m_1)}{M^2l_1^2l_2} & 0 & \frac{g^2(M^2l_2^2+Ml_1^2m_2+Ml_1l_2m_2+2Ml_2^2m_1+l_1^2m_2^2+2l_1l_2m_1m_2+l_2^2m_1^2)}{M^3l_1^3l_2^2}\\ \frac{1}{Ml_1} & 0 & -\frac{g(Ml_2+l_1m_2+l_2m_1)}{M^2l_1^2l_2} & 0 & \frac{g^2(M^2l_2^2+Ml_1^2m_2+Ml_1l_2m_2+2Ml_2^2m_1+l_1^2m_2^2+2l_1l_2m_1m_2+l_2^2m_1^2)}{M^3l_1^3l_2^2} & 0\\ 0 & \frac{1}{Ml_2} & 0 & -\frac{g(Ml_1+l_1m_2+l_2m_1)}{M^2l_1l_2^2} & 0 & \frac{g^2(M^2l_1^2+2Ml_1^2m_2+Ml_1l_2m_1+Ml_2^2m_1+l_1^2m_2^2+2l_1l_2m_1m_2+l_2^2m_1^2)}{M^3l_1^2l_2^3}\\ \frac{1}{Ml_2} & 0 & -\frac{g(Ml_1+l_1m_2+l_2m_1)}{M^2l_1l_2^2} & 0 & \frac{g^2(M^2l_1^2+2Ml_1^2m_2+Ml_1l_2m_1+Ml_2^2m_1+l_1^2m_2^2+2l_1l_2m_1m_2+l_2^2m_1^2)}{M^3l_1^2l_2^3} & 0 \end{bmatrix} \tag{24}$$

The rank of the controllability matrix would be equal to the number of states ($n = 6$) in most cases, as obtained from the symbolic rank conditions in MATLAB, as shown in Figure 3. Clearly, from the controllability matrix given in (24), the rank will be less than six when $M = 0$, $m_1 = 0$, $m_2 = 0$, $l_1 = 0$, and $l_2 = 0$, which are not practical cases. It was necessary to check what would be the rank when the lengths of the pendulums and masses are same. So multiple tests were done in Matlab to verify the rank when these variables are same in different combinations. The most significant ones are given in the code for brewity and the results are available in Figure 3. **Thus, it is clear that the system is controllable when $M > 0$, $m_1 > 0$, $l_1 > 0$, $l_2 > 0$, and when $l_1 \neq l_2$.**

```
Symbolic Rank of the Controllability Matrix: 6
Rank of the Controllability Matrix when m1 = m2: 6
Rank of the Controllability Matrix when  l1 = l2: 4
Rank of the Controllability Matrix when  M = m1 = m2: 6
```

Figure 3: Controllability checks - Output from MATLAB

1.4 LQR Controller

D) (25 points) Choose M = 1000Kg, m1 = m2 = 100Kg, l1 = 20m and l2 = 10m. Check that the system is controllable and obtain an LQR controller. Simulate the resulting response to initial conditions when the controller is applied to the linearized system and 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.

Substituting the values of the masses and lengths in equation (24) gives the below controllability matrix:

$$\mathrm{C} = 1.0 \times 10^{-3} \times \begin{bmatrix} 0 & 1.0000 & 0 & -0.1472 & 0 & 0.1419 \\ 1.0000 & 0 & -0.1472 & 0 & 0.1419 & 0 \\ 0 & 0.0500 & 0 & -0.0319 & 0 & 0.0227 \\ 0.0500 & 0 & -0.0319 & 0 & 0.0227 & 0 \\ 0 & 0.1000 & 0 & -0.1128 & 0 & 0.1249 \\ 0.1000 & 0 & -0.1128 & 0 & 0.1249 & 0 \end{bmatrix}$$

The system is said to be controllable if and only if the controllability matrix C has full rank (i.e., the rank of C is equal to the number of states n).The above matrix C has a rank equal to 6, and the given system is controllable. Additionally, it was proven in the previous section that the system is controllable when the parameters are greater than zero and when $l_1 \neq l_2$. Thus the system is controllable for the given parameter values.

We can control a system to reach a desired state using various controllers. Here, we employ the Linear Quadratic Regulator (LQR) controller to drive the system to the desired state.

The Linear Quadratic Regulator (LQR) is an optimal feedback control algorithm that minimizes a quadratic cost function. It utilizes all state variables to compute a control input, where each state variable is multiplied by a gain and summed to generate a single actuation value. The LQR controller is designed to provide the optimal state-feedback law that minimizes a quadratic cost function.

The LQR controller aims to minimize the following cost function [2]:

$$J(K, \bar{X}(0)) = \int_0^\infty \left(\bar{X}^T(t) Q \bar{X}(t) + \bar{U}_K^T(t) R \bar{U}_K(t)\right) dt \tag{25}$$

where:

- $\bar{X}(t)$: State vector at time t, representing the system's current states (e.g., position, velocity, angles).
- $\bar{U}_K(t)$: Control input at time t, computed using the state-feedback law.
- Q: Positive semi-definite matrix that penalizes deviations of the state $\bar{X}(t)$ from the desired state. It assigns weights to individual state variables.
- R: Positive definite matrix that penalizes the magnitude of the control input $\bar{U}_K(t)$. It reflects the cost of applying control effort.
- $J(K, \bar{X}(0))$: Quadratic cost function to be minimized over the time horizon.

1.4.1 Optimal LQR Controller

If the pair (A, B_K) is stabilizable, then the optimal state-feedback control law is given by [2]:

$$\bar{U}_K(t) = K\bar{X}(t), \quad \text{where } K = -R^{-1} B_K^T P \tag{26}$$

where:

- K: Optimal feedback gain matrix.
- P: Symmetric positive definite solution of the Algebraic Riccati Equation.

1.4.2 Algebraic Riccati Equation

The matrix P is obtained as the solution to the following stationary Riccati equation [2]:

$$A^T P + PA - PBR^{-1} B^T P = -Q \tag{27}$$

where:

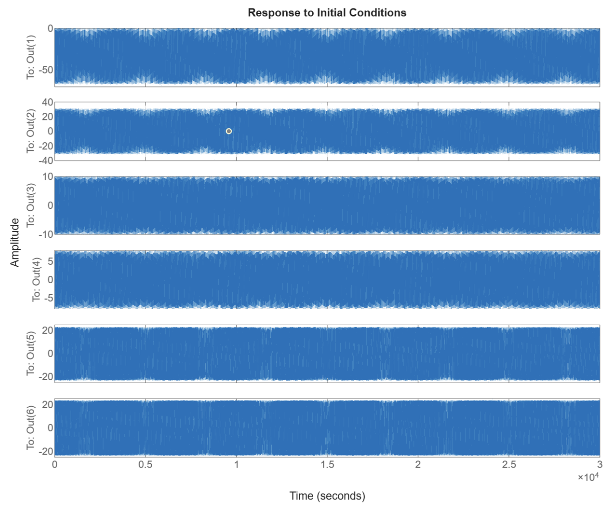


Figure 4: Initial Response of the system

- A: System dynamics matrix (state matrix).
- B: Input matrix that maps control inputs to the states.
- P: Symmetric positive definite matrix that characterizes the cost-to-go function for the optimal control.

1.4.3 Summary

The LQR controller computes an optimal feedback gain matrix K by solving the Riccati equation. This gain minimizes the quadratic cost function J, balancing the trade-off between state deviations (penalized by Q) and control effort (penalized by R).

To streamline the process, the rank of the controllability matrix was also calculated in MATLAB and was determined to be 6, which matches the number of state variables or the order of the matrix A.

1.4.4 Simulation of state

The simulations were conducted in MATLAB using the LQR function, The code used for the same is provided in Appendix Section. The results for the system's response to the specified initial conditions are presented below. These include:

The response of the linearized system is shown in Fig: 4. The system with initial conditions is not stable and the response oscillates over time as shown in the figure.

The response of the LQR-controlled system with the closed-loop dynamics $A + B_K$ is shown in Fig: 5. The system returns to equilibrium which verifies the stability ensured by the LQR controller. The gain matrix K is calculated using the lqr method provided by MATLAB, which

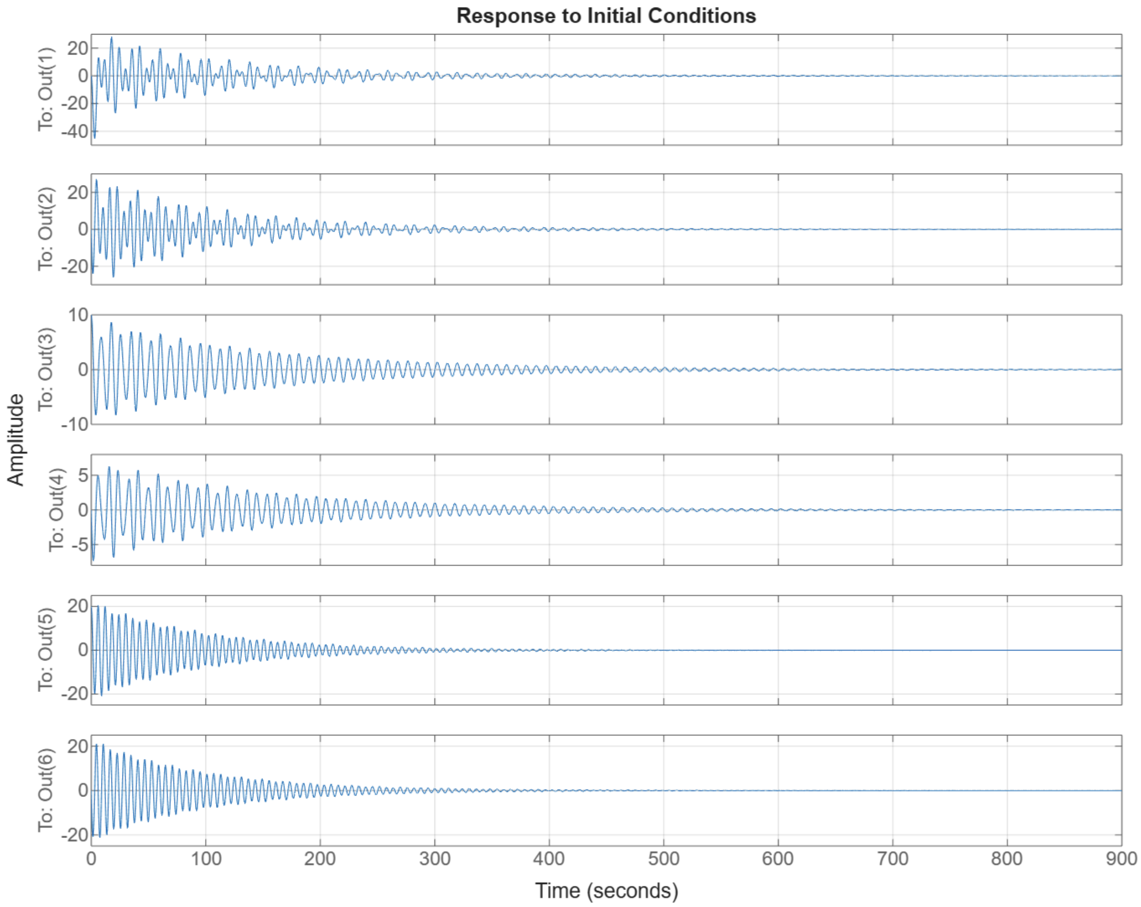


Figure 5: Response of the system with closed-llop dynamics

solves the ricatti equation and provides the values for gain matrix k and it's value was found to be:

$$K = \begin{bmatrix} 100.0000 & 498.2424 & -62.1098 & -520.5399 & -23.9485 & -259.8194 \end{bmatrix}$$

The response of the original nonlinear system is shown in Fig: 6. The nonlinear step response assesses the controller's performance when applied to the actual nonlinear system, contrasting with its design for the linearized model. As shown in the figure, the system stabilizes over time even though we designed the system with linearized assumptions.

we choose $Q = 1000 \cdot I_6$ and $R = 0.1$. Q is a diagonal matrix where larger values penalize deviations in the state variables more heavily. By setting $Q = 1000 \cdot I_6$ we ensure that all six state variables are equally weighted and deviations are strongly penalized, emphasizing accurate state regulation. The large value of 1000 ensures aggressive correction of state deviations, which is appropriate for systems requiring precise control.

R is a scalar (or diagonal matrix) that penalizes the control effort. A small value like

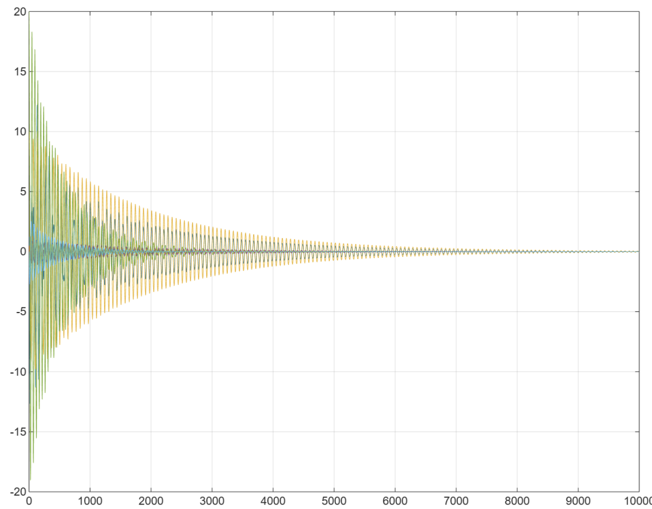


Figure 6: Response of the Non Linear System

$R = 0.1$ indicates that we are less concerned about minimizing control effort compared to achieving accurate state regulation. This choice prioritizes performance (state stability and tracking) over energy efficiency, We noticed that when the R value is increased the control effort is reduced.

We also tried various values for Q and R and plotted input to visualize how the control input is varying and settled with the above stated values.

1.4.5 Lyapunov Stability Analysis (Indirect Method)

Lyapunov stability analysis is a mathematical method used to determine the stability of equilibrium points in dynamical systems. It provides a rigorous framework to assess whether a system remains bounded and converges to a stable state over time. The method relies on constructing a scalar function, called the Lyapunov function, that behaves like an energy function for the system.

The stability of the system is analyzed using Lyapunov's stability criteria. This approach determines the stability of the equilibrium point of a system by analyzing the properties of the system matrix (using the eigen values of $A + B_K K$ Matrix). Specifically, for the LQR-controlled system with closed-loop dynamics given by $A + B_K$, the eigenvalues of the matrix were calculated.

The eigenvalues of $A + B_K$ were found to be:

$$
\begin{aligned}
\lambda_1 &= -0.2067 + 0.2024i, \\
\lambda_2 &= -0.2067 - 0.2024i, \\
\lambda_3 &= -0.0103 + 1.0421i, \\
\lambda_4 &= -0.0103 - 1.0421i, \\
\lambda_5 &= -0.0061 + 0.7277i, \\
\lambda_6 &= -0.0061 - 0.7277i.
\end{aligned}
$$

Since the real parts of all the eigenvalues are negative, the system is asymptotically stable according to criteria in Lyapunov's Indirect Method. This result confirms that the LQR controller effectively stabilizes the system.

2 Component - 2

Consider the parameters selected in C) above.

2.1 Observable Linearized System

E) Suppose that you can select the following output vectors: x(t), (1(t), 2(t)), (x(t), 2(t)) or(x(t), 1(t), 2(t)). Determine for which output vectors the linearized system is observable.

The output equation in state space representation is

$$y = Cx + Du$$

For a Linear Time Varying (LTV) system, the system is observable when the observability matrix has a rank that equals the number of states.

That is, the system is observable when:

$$\text{rank}[\mathcal{O}] = \text{rank}\begin{bmatrix} C^T & A^TC^T & \cdots & (A^T)^{n-1}C^T \end{bmatrix} = n$$

This can also be written as;

$$\text{rank}[O] = \text{rank}\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} = n \tag{28}$$

Given the output vectors:

$$y = x(t)$$
$$y = (\theta_1(t), \theta_2(t))$$
$$y = (x(t), \theta_2(t))$$
$$y = (x(t), \theta_1(t), \theta_2(t)).$$

We have to find if the system is observable with these output vectors. For this the rank test as given above can be utilized.

1. For achieving $y = x(t)$, the C vector would be:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then,

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

2. For achieving $y = (\theta_1(t), \theta_2(t))$, the C vector would be ,

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

where $\theta_1(t)$ is associated with the first row, and $\theta_2(t)$ is associated with the second row.

Then,

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \theta_1(t) \\ x_4(t) \\ \theta_2(t) \\ x_6(t) \end{bmatrix} = (\theta_1(t), \theta_2(t))$$

3. For achieving $y = (x(t), \theta_2(t))$, the C vector would be ,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

where $x(t)$ is associated with the first row, and $\theta_2(t)$ is associated with the second row.

Then,

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \theta_1(t) \\ x_4(t) \\ \theta_2(t) \\ x_6(t) \end{bmatrix} = (x(t), \theta_2(t))$$

4. For achieving $y = (x(t), \theta_1(t), \theta_2(t))$, the C vector would be ,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

where $x(t)$ is associated with the first row, $\theta_1(t)$ is associated with the second row, and $\theta_2(t)$ is associated with the third row.

Then,

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \theta_1(t) \\ x_4(t) \\ \theta_2(t) \\ x_6(t) \end{bmatrix} = (x(t), \theta_1(t), \theta_2(t))$$

The rank test of observability matrix was performed in MATLAB using the equation (28). The MATLAB code used for the same is available in the Appendix Section. The results of the same are given in Figure 7. It can be seen that the system is not observable when the output vector is not dependent on x(t), which is the horizontal position of the cart.

```
Number of states: 6
-------------------------------------------------
When output vector = x(t) :
Matrix C:
     1     0     0     0     0     0

Rank of the Observability Matrix: 6
The system is observable.
-------------------------------------------------
When output vector = (θ1(t), θ2(t)) :
Matrix C:
     0     0     1     0     0     0
     0     0     0     0     1     0

Rank of the Observability Matrix: 4
The system is not observable.
-------------------------------------------------
When output vector = (x(t), θ2(t)) :
Matrix C:
     1     0     0     0     0     0
     0     0     0     0     1     0

Rank of the Observability Matrix: 6
The system is observable.
-------------------------------------------------
When output vector = (x(t), θ1(t), θ2(t)) :
Matrix C:
     1     0     0     0     0     0
     0     0     1     0     0     0
     0     0     0     0     1     0

Rank of the Observability Matrix: 6
The system is observable.
-------------------------------------------------
```

Figure 7: Observability Checks - Output from MATLAB

2.2 Luenberger Observer

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.

The observable output vectors are $x(t)$, $(x(t), \theta_2(t))$, and $(x(t), \theta_1(t), \theta_2(t))$. Considering that an option is given between $(x(t), \theta_2(t))$ and $(x(t), \theta_1(t), \theta_2(t))$, we will be considering $(x(t), \theta_1(t), \theta_2(t))$. Thus, the output vectors in consideration are

$$x(t) \quad \text{and} \quad (x(t), \theta_1(t), \theta_2(t)).$$

The Luenberger Observer is given by the following state-space representation [2]:

$$\dot{\hat{X}}(t) = A\hat{X}(t) + B_K\tilde{U}_K(t) + L\left(\tilde{Y}(t) - C\hat{X}(t)\right), \quad \hat{X}(0) = 0 \tag{29}$$

where L is the observer gain matrix and $\left(\tilde{Y}(t) - C\hat{X}(t)\right)$ is the correction term.

The estimation error $\tilde{X}_e(t) = \tilde{X}(t) - \hat{\tilde{X}}(t)$ has the following state-space representation:

$$\dot{\tilde{X}}_e(t) = (A - LC)\tilde{X}_e(t) + B_D\tilde{U}_D(t)$$

The matrix $A - LC$ is stable if and only if $(A - LC)^T = A^T - C^TL^T$ is stable. We have established the observability of the output vectors in consideration earlier. Since the system is observable, $A - LC$ is also controllable. This means that the real parts of the eigenvalues of $A - LC$ lie in the left half of the complex plane. Therefore, the key factor in this system is the magnitude of pole placement in the left half-plane, which corresponds to the eigenvalues of $A - LC$. These eigenvalues can be adjusted by selecting an appropriate observer gain matrix L, allowing us to control the pole placement (i.e., the eigenvalues) of $A - LC$ by choosing L. MATLAB has an inbuilt function called place that can be used for pole placement [3] that can be used to find the observer gain L.The question does not specify whether the Luenberger Observer should be designed for an open-loop or closed-loop system. Since Section G of the question includes a closed-loop LQG controller, this question will be approached as an open-loop system, keeping in mind that the intend behind this question is for students to reason through the design of a Luenberger Observer. It is important to note that the approach for a closed-loop system would be similar to the one described in Section G.

Referring `Linear Systems Theory` by João Hespanha [4], it is stated that if the L is chosen such that A-LC is a stability matrix, then the state estimation error e converges to zero exponentially fast for every input signal u, as given in theorem 16.8 of the book as given in Figure 8. It is also to be noted that the equation 16.7 referred in the theorem below is as same as equation (29). This means that more negative the value of eigen values the faster the settling rate. However, in some practical cases convergence rate (or settling rate) is not only criteria of evaluation. The error value before settling is also crucial as in some cases higher values could

Theorem 16.8. *Consider the closed-loop state estimator (16.7). If the output injection matrix gain $L \in \mathbb{R}^{n \times m}$ is selected so that $A - LC$ is a stability matrix, then the state estimation error e converges to zero exponentially fast, for every input signal u.* □

Figure 8: Theorem from Linear Systems Theory by João Hespanha

affect results. Thus to get the "best" Luenberger Observer an optimization search was setup keeping the settling rate and estimation error till convergence in loss function. The optimizer would find the poles with minimum loss value. The optimization process followed is defined below.

2.2.1 Optimization Process to find the "best" Luenberger Observer

As explained earlier, optimization was performed to find the best pole placement based on a loss function. The loss function function is a weighted sum of three terms: settling time, sum of estimation errors, and pole penalty. This is done using an optimization algorithm (`fminsearch` in MATLAB) [5], with constraints enforced by the pole penalty term in loss function. `fminsearch` is a non-linear programming solver used to find minima without using any derivative based methods. The loss function was defined as;

$$\begin{aligned}\text{Loss function} = w_1 \cdot (\text{settling time}) + w_2 \cdot (\text{Estimation Error till convergence}) \\ + \text{penalty for out of bound pole values}\end{aligned}$$

Where:
- w_1 is the weight for settling time
- w_2 is the weight for the sum of errors

The weights were chosen as $w_1 = 1$ and $w_2 = 0.5$ to prioritize convergence rate while allowing a small trade-off in estimation error. This choice helps minimize oscillations around zero, ensuring a more stable convergence without excessive delay in reaching the desired state estimate. The penalty function for the poles ensures that they stay within the valid range between -1000 and 0. It was designed to penalize any pole that falls outside this range, ensuring that the optimization search remains within reasonable bounds. The penalty value is given as;

$$\text{pole\_penalty} = \sum_{i=1}^{6} \left[\begin{cases} (pole_i + 1000)^2 & \text{if } pole_i < -1000 \\ (pole_i + 0.1)^2 & \text{if } pole_i > 0 \\ 0 & \text{else} \end{cases} \right]$$

Where $pole_i$ is the i-th pole corresponding to each state. The magnitude of penalty value is not significant as long as the optimizer rejects out of bounds values.

The optimization process stops when either of the following criteria are met:

1. When the estimation error is less than a tolerance of 1×10^{-6}.
2. The change in the pole values is smaller than a threshold value.
3. When the iteration count reaches 10000.

This optimized pole set is used to place the observer poles using the inbuilt function in MATLAB called place [3] which returns the best observer gain matrix L. Using the best observer gain matrix, simulation was done for both linear and nonlinear systems.

2.2.2 Luenberger Observer Design and Simulation Results

The Figure 9 is the output from MATLAB (code available in Appendix Section) containing the optimization results and "best" observer gain matrix when the output vector is $(x(t))$

```
Optimized poles:
   -1.0550   -2.0664   -3.1316   -3.8872   -4.5093   -5.7996

Best observer gain matrix , L:
   1.0e+03 *

    0.0204
    0.1653
   -2.7689
    0.0365
    2.0969
   -1.3170
```

Figure 9: Optimal Observer Gain Matrix for output x(t) - Output from MATLAB

Using the best Luenberger Gain Matrix the system was simulated for unit step input for initial conditions for both linear and nonlinear systems. The estimation error for linear system is available in Figure 10. It can be seen that the estimation error is reaching to zero very quickly due to the choice of the observer gain matrix value. The optimization exercise played a crucial role in balancing the settling time and error variation of the estimator. The performance for the nonlinear system is available in Figure 11.In the case of a nonlinear system, the error value decreases exponentially over time, demonstrating the observer's effectiveness in achieving asymptotic state tracking. This behavior highlights the robustness of the Luenberger observer design, even in the presence of system nonlinearity. The exponential convergence ensures that the observer compensates for initial estimation errors and external disturbances thereby achieving the required observer performance.

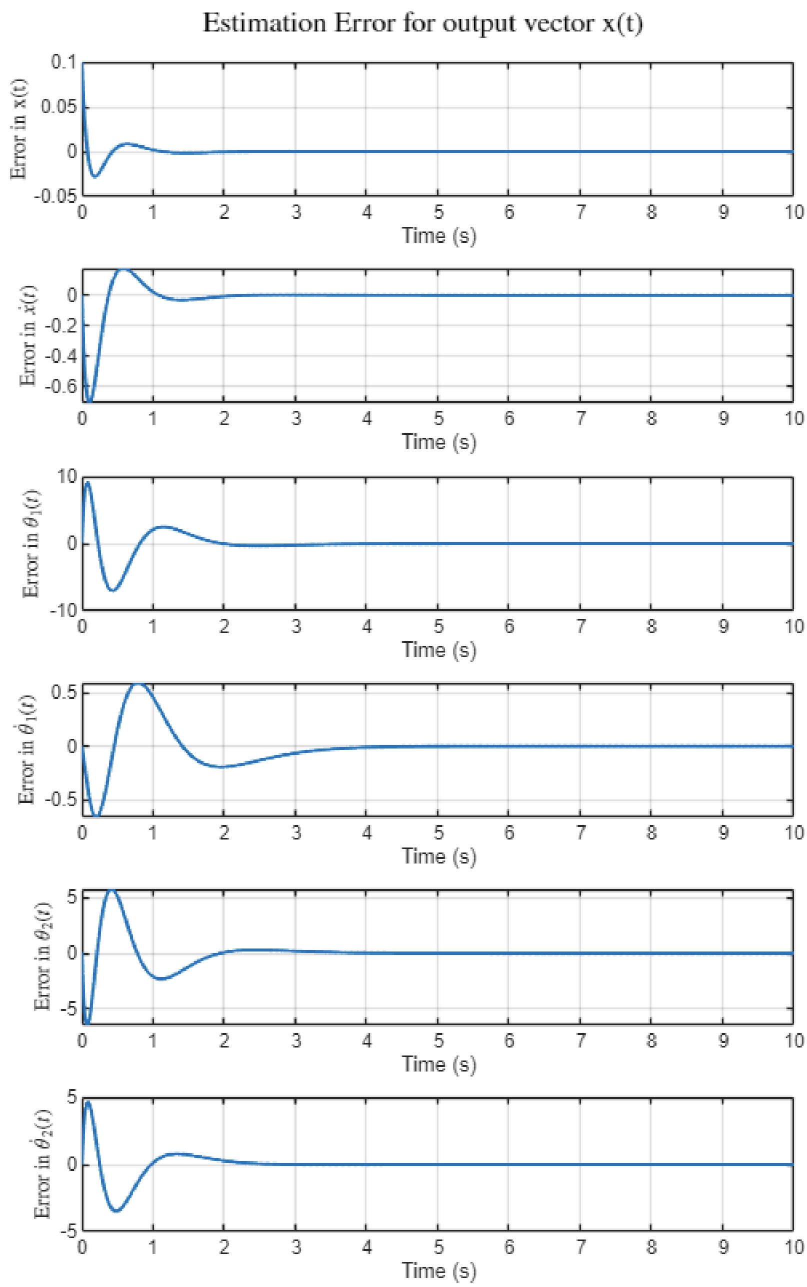


Figure 10: Linear system estimation performance for output x(t) - Output from MATLAB

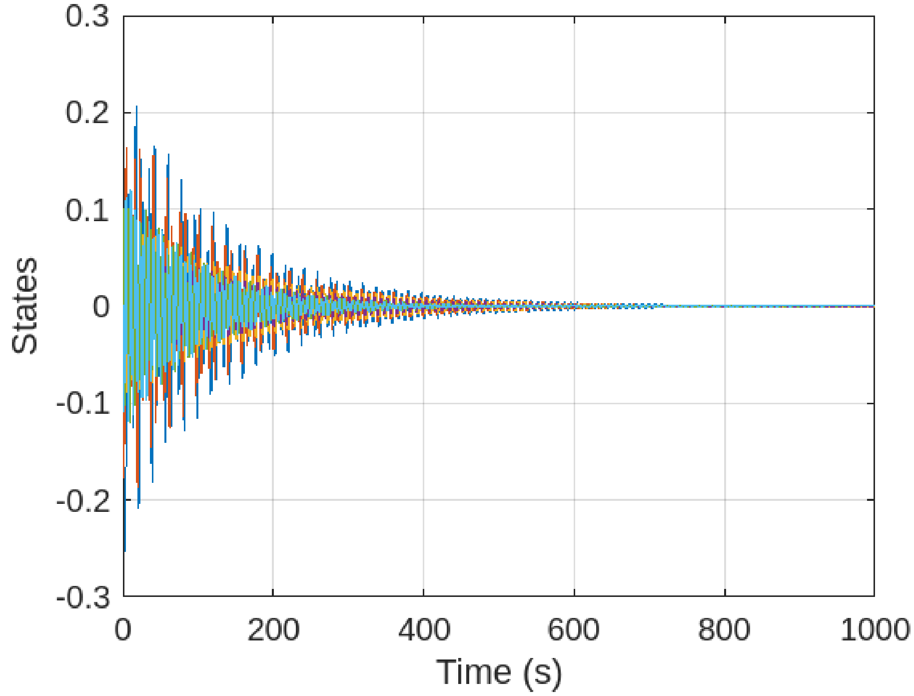


Figure 11: Nonlinear system estimation performance for output x(t) - Output from MATLAB

The optimization exercise was performed for output vector $(x(t), \theta_1(t), \theta_2(t))$, similar to how it was performed before. The optimal pole placement values and observer gain matrix values are available in Figure 12.Using this gain matrix the system was simulated for unit step input for initial conditions for both linear and nonlinear systems. The estimation error for linear system is available in Figure 13. It can be seen that the estimation error is reaching to zero very quickly due to the choice of the observer gain matrix value. For the nonlinear system, the estimation performance is also similar, demonstrating the robustness of the observer design.

```
Optimized poles:
   -1.0739   -1.9822   -1.7933   -4.2964   -4.9847   -6.7856

Best observer gain matrix , L:
    7.5272   -1.6356   -0.0000
   12.4604   -8.6677   -0.9810
   -1.6283   10.3329    0.0000
   -7.6474   25.0930   -0.0489
   -0.0000    0.0000    3.0562
   -0.0000   -0.0981    1.0497
```

Figure 12: Optimal Observer Gain Matrix for Outputs $(x(t), \theta_1(t), \theta_2(t))$ - Output from MATLAB

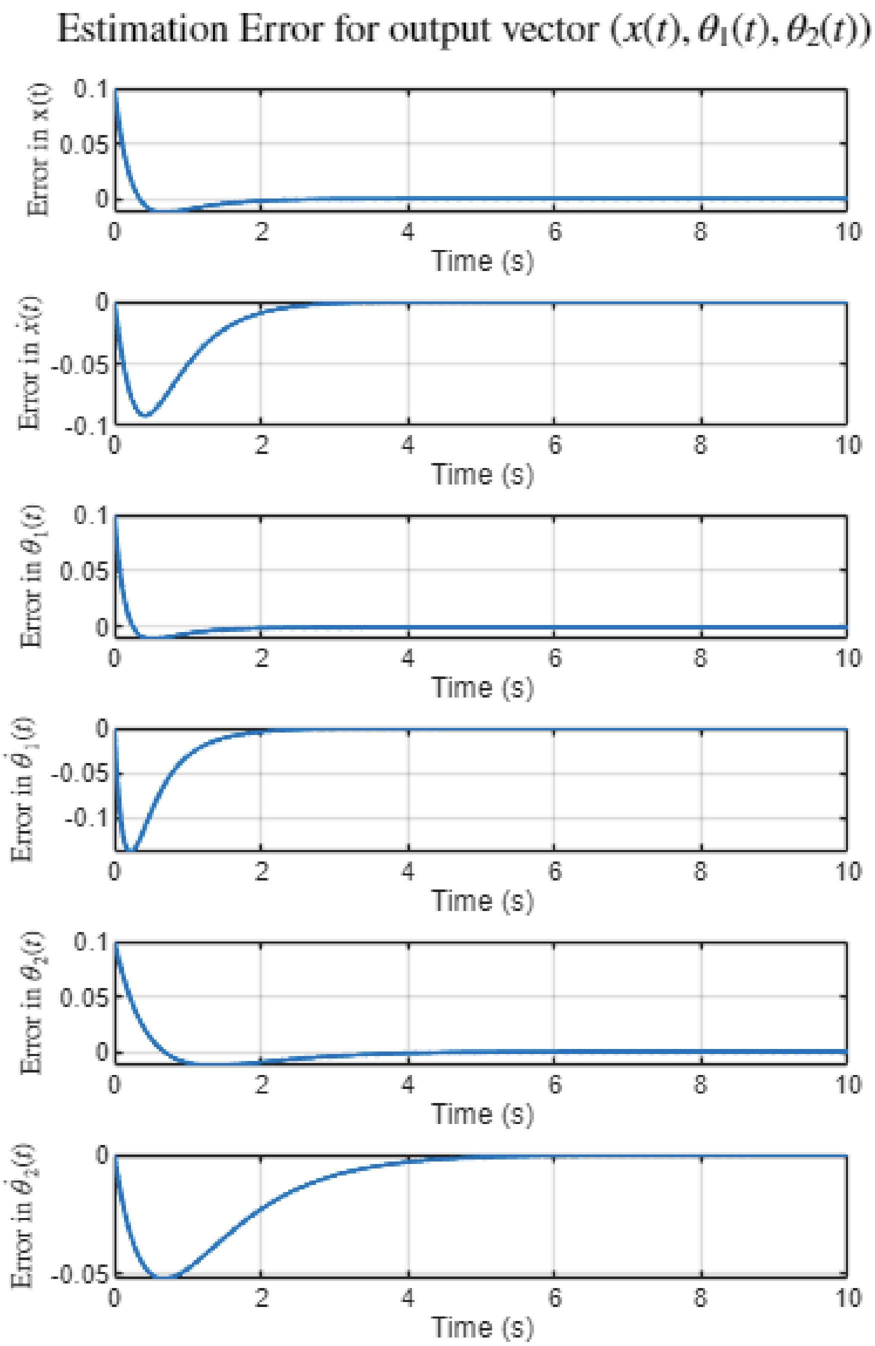
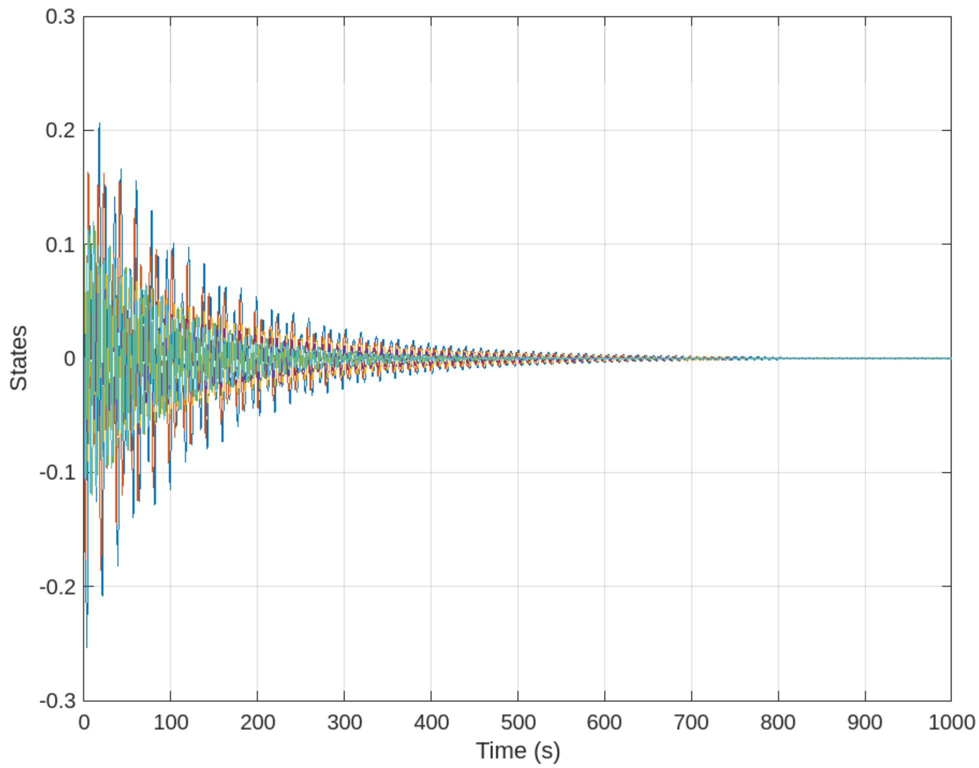


Figure 13: Linear system estimator performance for Outputs $(x(t), \theta_1(t), \theta_2(t))$ - Output from MATLAB



H

Figure 14: Nonlinear system estimator performance for Outputs $(x(t), \theta_1(t), \theta_2(t))$ - Output from MATLAB

2.3 LQG Controller

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 ?

2.3.1 Linear Quadratic Gaussian (LQG) Controller

The Linear Quadratic Gaussian (LQG) controller is a powerful framework for designing optimal control systems in the presence of process and measurement noise. It combines concepts from Linear Quadratic Regulator (LQR) for optimal state feedback control and Kalman filtering for optimal state estimation, resulting in a robust controller suitable for stochastic systems [2].

2.3.2 Components of the LQG Controller

The LQG controller has three main components:

1. **System Model:** The system dynamics are modeled using a state-space representation:

$$\dot{x}(t) = Ax(t) + Bu(t) + B_D U_D(t), \quad y(t) = Cx(t) + v(t),$$

   where:

   - $x(t)$ is the state vector,
   - $u(t)$ is the control input,
   - $y(t)$ is the measurement output,
   - $U_D(t)$ is the process noise,
   - $v(t)$ is the measurement noise.

2. **Optimal State Feedback (LQR):** The LQR provides an optimal control law by minimizing the following quadratic cost function:

$$J = \int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) dt,$$

   where Q and R are positive semi-definite and positive definite weighting matrices, respectively. The optimal control input is:

$$u(t) = -Kx(t),$$

   where $K = R^{-1}B^T P$, and P is the solution to the algebraic Riccati equation.

3. **State Estimation (Kalman Filter):** Since the state $x(t)$ is not always directly measurable, the Kalman filter provides an optimal estimate $\hat{x}(t)$ by minimizing the mean squared estimation error:

$$e(t) = x(t) - \hat{x}(t).$$

The Kalman filter gain L is calculated by solving:

$$L = PC^T V^{-1},$$

where P is the solution to the Riccati equation for the estimation error covariance.

2.3.3 Augmented Dynamics for LQG

The LQG controller integrates the state feedback and the Kalman filter into a single framework. The augmented dynamics for the closed-loop system are:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{\hat{x}}(t) \end{bmatrix} = \begin{bmatrix} A - BK & BK \\ LC & A - LC \end{bmatrix} \begin{bmatrix} x(t) \\ \hat{x}(t) \end{bmatrix}.$$

2.3.4 Advantages of LQG

The LQG controller has several advantages:

- Provides optimal control in the presence of noise.
- Separates control and estimation, simplifying design.
- Can handle multivariable systems efficiently.

2.3.5 Limitations of LQG

Despite its strengths, LQG has some limitations:

- Assumes linear dynamics and Gaussian noise, which may not hold in real systems.
- The design depends on accurate modeling of system parameters.

In summary, the LQG controller is a widely used framework for controlling stochastic linear systems, balancing performance and robustness through a systematic design process.

2.3.6 Smallest Output Vector

The smallest output vector corresponds to observing only $x(t)$, represented as:

$$C_1 = [1, 0, 0, 0, 0, 0].$$

2.3.7 Controller Design Using LQG

The controller is designed by combining:

- **LQR State Feedback:** Minimizes the cost function:

  $$J = \int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) dt,$$

  where $Q = 1000 \cdot I_6$ and $R = 0.1$. Q is a diagonal matrix where larger values penalize deviations in the state variables more heavily. By setting $Q = 1000 \cdot I_6$ we ensure that all six state variables are equally weighted and deviations are strongly penalized, emphasizing accurate state regulation. The large value of 1000 ensures aggressive correction of state deviations, which is appropriate for systems requiring precise control.

  R is a scalar (or diagonal matrix) that penalizes the control effort. A small value like $R = 0.1$ indicates that we are less concerned about minimizing control effort compared to achieving accurate state regulation. This choice prioritizes performance (state stability and tracking) over energy efficiency, We noticed that when the R value is increased the control effort is reduced.

  We also tried various values for Q and R and plotted input to visualize how the control input is varying and settled with the above stated values.

- **Kalman Filter:** Provides state estimation by minimizing the estimation error covariance. Noise covariances are $W = 0.01 \cdot I_6$ and $V_1 = 0.01$. W represents the covariance of process noise $w(t)$, which models the uncertainties in the system dynamics. A small value 0.01 indicates that the system dynamics are relatively well-known, with only minor uncertainties. By scaling W uniformly across all state variables I_6, we assume equal confidence in the dynamics of all state variables.

  V_1 represents the covariance of measurement noise $v(t)$, which models the noise in sensor readings. A small value 0.01 reflects high-quality sensors with minimal noise. Since the smallest output vector $C_1 = [1, 0, 0, 0, 0, 0]$ observes only $x(t)$, $V_1$is scalar and applies solely to x(t) measurements.

  We choose the values $W = 0.01 \cdot I_6$ and $V_1 = 0.01$ making assumptions that the state model pretty much models the dynamics of the linearized system accurately and the feedback is also very accurate.

The augmented system combines state feedback and state estimation dynamics.

2.3.8 Nonlinear System Simulation

The LQG controller is applied to the original nonlinear system. The system dynamics and controller implementation are shown in the Appendix Section.

2.3.9 Simulation Results

The initial response of the system using LQG Controller is shown in Fig: 15. The initial response of a system shows how the system evolves over time when it starts from a specific non-equilibrium initial state (i.e., an initial condition away from the desired state or trajectory) with no external inputs applied.

The step response of the LQG Controller is shown in Fig: 16. The step response measures how the system reacts. As shown in the figure the system initially had some disturbances but stabilizes over time.

The non linear system response using the LQG controller is shown in Fig: 17. The nonlinear step response analyzes the behavior of the original nonlinear system (instead of its linearized model) under the influence of the same step input.

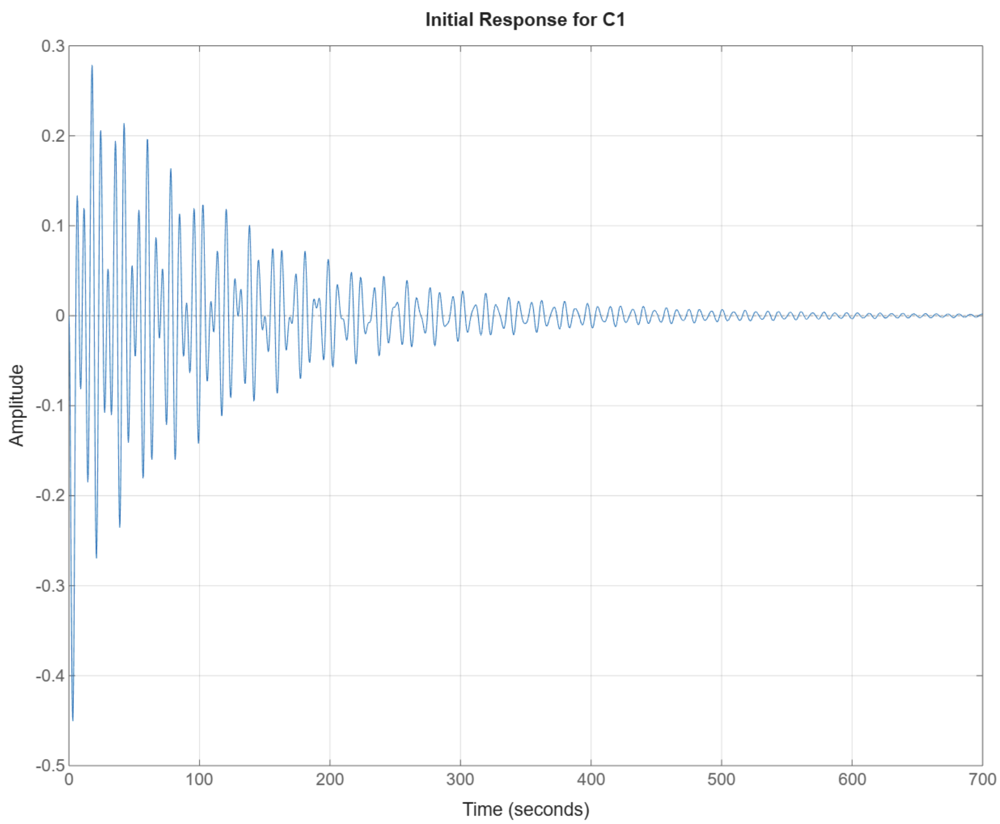


Figure 15: Initial Response for C_1.

UNIVERSITY OF MARYLAND

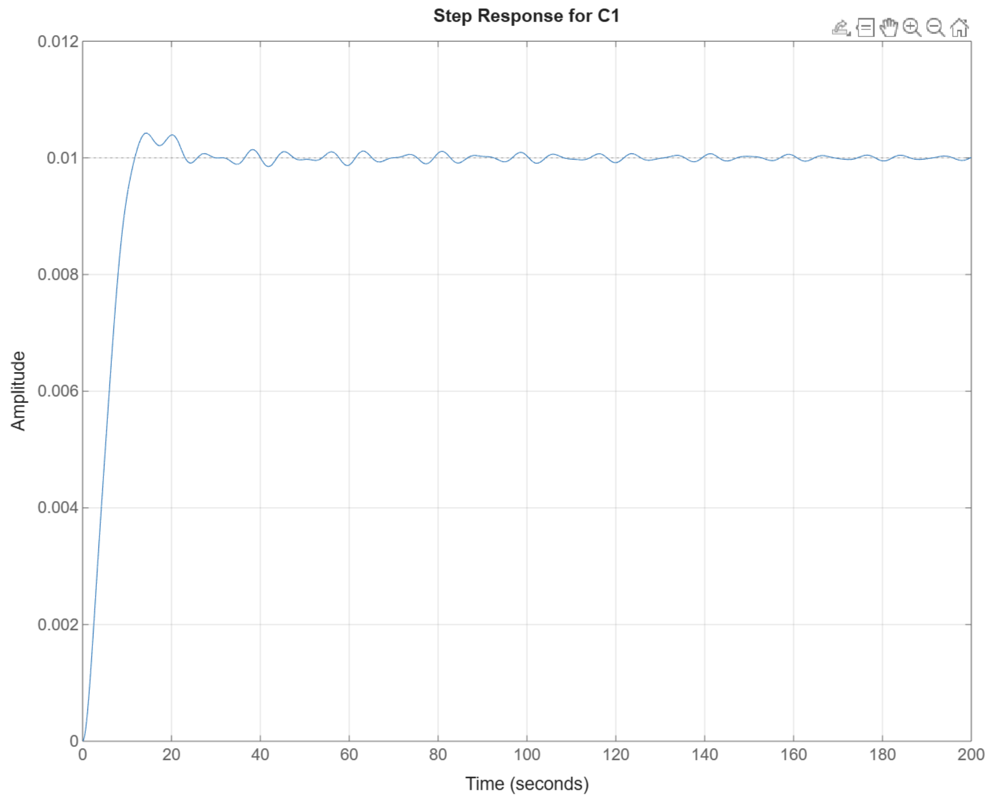


Figure 16: Step Response for C_1.

2.3.10 Reconfiguring the Controller

- **Tracking a Constant Reference on x :** To track a constant reference r on x, an integral action can be added to the system. The augmented state becomes:

$$\tilde{x}(t) = \begin{bmatrix} x(t) \\ \int_0^t (r - x(t))\, dt \end{bmatrix}.$$

Here, $x(t)$ denotes the original state vector, and the integral term $\int (r - x(\tau))d\tau$ represents the cumulative error up to time t. The LQR and Kalman filters were then redesigned to accommodate this augmented state.

- **Disturbance Rejection** The Linear Quadratic Gaussian (LQG) controller provides effective disturbance rejection due to its optimal design that combines state feedback control (via the Linear Quadratic Regulator, LQR) and robust state estimation (via the Kalman filter). This design inherently accounts for process noise and measurement noise, ensuring resilience to external perturbations such as constant force disturbances.

Constant force disturbances act as steady-state offsets in the dynamics. The feedback loop

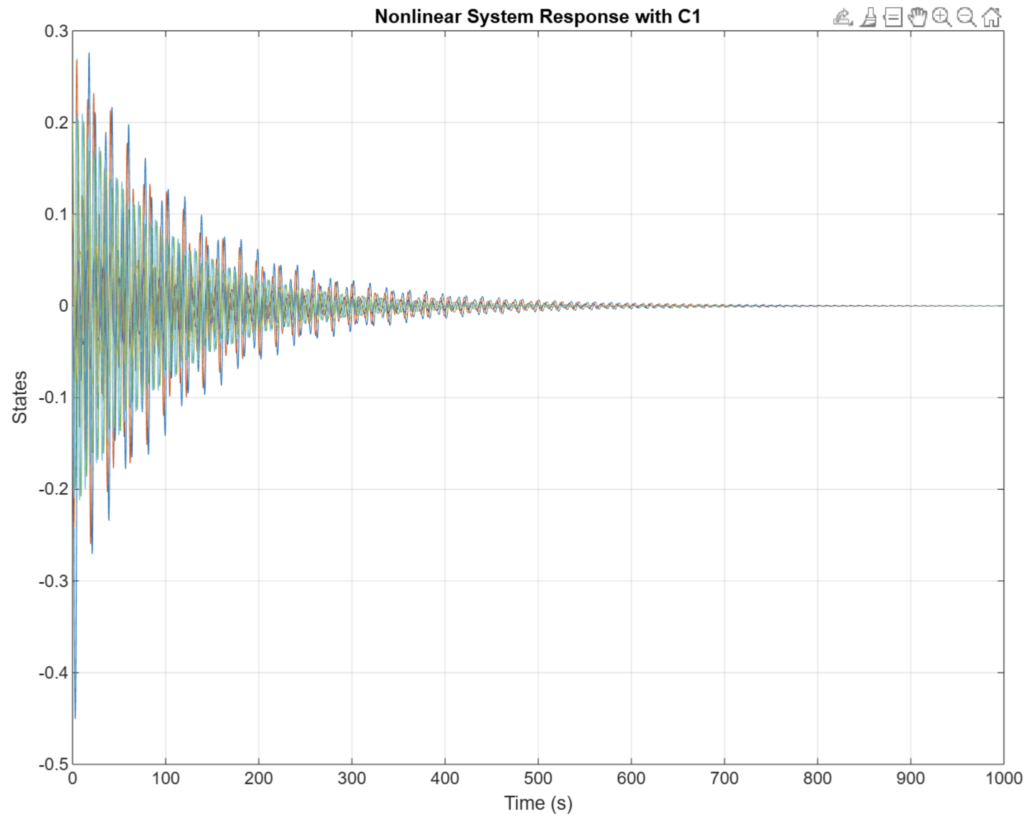


Figure 17: Nonlinear System Response with C_1.

in the LQG compensates for these by adjusting the control input, ensuring that the state converges to the desired trajectory. Moreover, if integral action is added to the controller, the system can asymptotically eliminate steady-state errors caused by such disturbances.

The LQG framework is optimal under the assumption that disturbances and measurement noise are Gaussian. If disturbances or noise are not Gaussian, the LQG controller can still work, but the Kalman filter may no longer provide the minimum mean-squared error estimate. The controller will still stabilize the system, but its performance may not be optimal and the performance of the LQG controller might degrade.

2.3.11 Conclusion

The LQG controller effectively stabilizes the nonlinear system, tracks constant references, and rejects disturbances. The smallest output vector C_1 provides sufficient observability while minimizing measurement requirements.

3 Conclusion

In this project, the equations motion for two pendulums on a cart was derived. A state space representation was formed using these equations. The system was then linearized around the equilibrium points to get a linearized state space representation. Then the controllability conditions of the system was arrived from rank tests after which an LQR controller was designed. Lyapunov's indirect method was used to evaluate the stability of this system. Then the observability of the system was evaluated for different output vectors and an optimal Luenberger observer was designed for each of the output vectors discussed. This Luenberger observer was then utilized to simulate the response of the system to initial conditions and unit step input for both linear and nonlinear system. Then, an LQG controller was designed for the smallest output vector and it's performance was discussed in detail.

The Linear Quadratic Gaussian (LQG) controller is the best choice for this project due to its optimal performance, robustness, and flexibility. By integrating Linear Quadratic Regulator (LQR) state feedback and Kalman filter-based state estimation, the LQG controller ensures optimal performance even in the presence of noise and uncertainties. Its ability to balance state tracking accuracy and control effort makes it well-suited for systems requiring precision. The Kalman filter provides robust state estimation, efficiently handling process and measurement noise, and allows the controller to function effectively with the smallest output vector ($C_1 = [1, 0, 0, 0, 0, 0]$), simplifying implementation and reducing computational complexity. Furthermore, the controller demonstrates strong compatibility with the original nonlinear system, achieving stabilization and control despite being designed for the linearized model. The LQG framework also allows for reconfiguration to track constant references by augmenting the state with an integral term, and inherently rejects constant force disturbances, showcasing its robustness to external perturbations. Additionally, the systematic separation of control and estimation simplifies the design process and ensures scalability to more complex systems. These qualities collectively make the LQG controller the most effective solution for this project, ensuring reliable and efficient performance under various conditions.

References

[1] P. Physics, "Lagrangian mechanics for beginners," 2024. www.profoundphysics.com/lagrangian-mechanics-for-beginners.

[2] W. A. Malik, "Advanced control methods," 2024. Lecture notes for ENPM667 - Control of Robotic Systems.

[3] MathWorks, *Pole Placement*, 2024. www.mathworks.com/help/control/getstart/pole-placement.html.

[4] J. P. Hespanha, *Linear Systems Theory: Second Edition*. Princeton University Press, 2018.

[5] I. The MathWorks, *fminsearch*, 2024. www.mathworks.com/help/matlab/ref/fminsearch.html.

4 Appendix

5 Code for Conditions of Controllability (Part-C).

```
% Controllability conditions check
% symbolic variables are defined here for solving the problem
syms m1 m2 l1 l2 g  M  real

% A matrix from state space representation
A = [0, 1, 0, 0, 0, 0;
     0, 0, -m1*g/M, 0, -m2*g/M, 0;
     0, 0, 0, 1, 0, 0;
     0, 0, (-m1*g-M*g)/(M*l1), 0, -m2*g/(M*l1), 0;
     0, 0, 0, 0, 0, 1;
     0, 0, -m1*g/(M*l2), 0, (-m2*g-M*g)/(M*l2), 0];

disp('Matrix A:');
disp(A);

% B matrix from state space representation
B = [0;
     1/M;
     0;
     1/(M*l1);
     0;
     1/(M*l2)];

disp('Matrix B:');
disp(B);


% Number of states
n = size(A, 1);
disp('Number of states:');
disp(n);

%
function C = controllability_matrix(A, B, n)
% Function to find controllability Matrix
% The controllability matrix  = [B, AB, A^2B, ..., A^(n-1)B].
%
% Inputs:
```


UNIVERSITY OF MARYLAND

```
%   A - (n x n matrix) A Matrix from state space representation
%   B - (n x m matrix) B Matrix from state space representation
%   n - (integer) Number of states.
%
% Outputs:
%   C - (n x n*m matrix) Controllability matrix.
    C = B; % Initialize
    for i = 1:n-1
        C = [C, A^i * B];
    end
end


Original_Controllability_Matrix = controllability_matrix(A, B, n);
% simplifying the expression. The simplifies expression is given in the
% report.
Simplified_Controllability_Matrix = simplify(Original_Controllability_Matrix);
disp('Controllability Matrix:');
disp(Simplified_Controllability_Matrix);

% Computing the rank of controllability matrix symbolically
rank_C = rank(Simplified_Controllability_Matrix);
fprintf('Symbolic Rank of the Controllability Matrix: %d\n', rank_C);

% Lets check the rank for specific conditions

% When the lengths of the pendulums are equal
% Applying substitution for l1 == l2
A_same_length = A;
B_same_length = B;
A_same_length = subs(A_same_length, l1, l2);  % Replace l1 with l2
B_same_length = subs(B_same_length, l1, l2);  % Replace l1 with l2

% When the mass of the pendulums are equal
% Applying substitution for m1 == m2
A_same_pendulum_mass = A;
B_same_pendulum_mass = B;
A_same_pendulum_mass = subs(A_same_pendulum_mass, m1, m2);  % Replace m1 with m2
B_same_pendulum_mass = subs(B_same_pendulum_mass, m1, m2);  % Replace m1 with m2


% When the mass of cart and mass of pendulums are same
% Applying substitution for m1 == m2 == M
A_same_all_mass = A_same_pendulum_mass;
```

```matlab
B_same_all_mass = B_same_pendulum_mass;
A_same_all_mass = subs(A_same_all_mass, M, m2);  % Replace M with m2
B_same_all_mass = subs(B_same_all_mass, M, m2);  % Replace M with m2

% Display the updated matrices   % Used for coding ad debugging. Commenting
% here
% disp('Updated A_same_pendulum_mass matrix:');
% disp(A_same_pendulum_mass);
% disp('Updated B_same_pendulum_mass matrix:');
% disp(B_same_pendulum_mass);

% Display the updated matrices
% disp('Updated A_same_all_mass matrix:');
% disp(A_same_all_mass);
% disp('Updated B_same_all_mass matrix:');
% disp(B_same_all_mass);

% Display the updated matrices
% disp('Updated A_same_length matrix:');
% disp(A_same_length);
% disp('Updated B_same_length matrix:');
% disp(B_same_length);

Controllability_Matrix = controllability_matrix(A_same_pendulum_mass, B_same_pendulum_mass, n);
% Computing the rank of controllability matrix symbolically
rank_C = rank(Controllability_Matrix);
fprintf('Rank of the Controllability Matrix when m1 = m2: %d\n', rank_C);


Controllability_Matrix = controllability_matrix(A_same_length, B_same_length, n);
% Computing the rank of controllability matrix symbolically
rank_C = rank(Controllability_Matrix);
fprintf('Rank of the Controllability Matrix when  l1 = l2: %d\n', rank_C);


Controllability_Matrix = controllability_matrix(A_same_all_mass, B_same_all_mass, n);
% Computing the rank of controllability matrix symbolically
rank_C = rank(Controllability_Matrix);
fprintf('Rank of the Controllability Matrix when  M = m1 = m2: %d\n', rank_C);
```

6 Code for LQR Controller(Part-D)

```
%% Initializing the Variables
M = 1000; % Mass of the cart
m1 = 100; % Mass of the 1st pendulum
m2 = 100; % Mass of the 2nd pendulum
l1 = 20;  % Length of the 1st pendulum
l2 = 10;  % Length of the 2nd pendulum
g = 9.81; % Acceleration due to gravity

%% Defining the A, B and C matrices.
A = [0 1 0 0 0 0;
    0 0 -m1*g/M 0 -m2*g/M 0;
    0 0 0 1 0 0;
    0 0 (-m1*g-M*g)/(M*l1) 0 -m2*g/(M*l1) 0;
    0 0 0 0 0 1;
    0 0 -m1*g/(M*l2) 0 (-m2*g - M*g)/(M*l2) 0];

B = [0;
    1/M;
    0;
    1/(M*l1);
    0;
    1/(M*l2)];
C = eye(6);
D = 0;
% Controllability Matrix
CtrlM = ctrb(A,B);
% Rank of the controllability matrix
if (rank(CtrlM) == 6)
    disp("Rank of the controllability matrix is equal to n, System " + ...
        "is Controllable")
else
    disp("Rank of the controllability matrix is less than n, System " + ...
        "is not Controllable")
end
%% Defining Q and R values
Q = 1000*eye(6);
R = 0.1;
x_0 = [0; 0; 0.1; 0; 0.2; 0]; % Initial condition as a column vector
% Defining the system
system = ss(A,B,C,D);
% Calculating K, Solution to Riccati equation and Poles
```


UNIVERSITY OF MARYLAND

```
[K,Solution,Poles] = lqr(system,Q,R);



figure;
initial(system, x_0); % Simulate states
grid on;

% Closed-loop system dynamics
Acl = A - B*K; % Closed-loop A matrix
system_cl = ss(Acl, B, C, D);

% Use initial() to simulate states
tspan = 0:0.01:30000; % Time vector
[y, t, x] = initial(system_cl, x_0, tspan); % Simulate states

% Compute the control input u(t)
u = -K * x'; % K * x' gives u for all time steps

% Plot states and control input
figure;
initial(system_cl, x_0)
grid on;
% Plot each state as a subplot
% for i = 1:6
%     subplot(7, 1, i); % 7 rows, 1 column, current subplot index
%     plot(t, x(:, i));
%     title(['State x_', num2str(i)]);
%     xlabel('Time (s)');
%     ylabel(['x_', num2str(i)]);
%     grid on;
% end
%
% % Plot control input
% subplot(7, 1, 7);
% plot(t, u);
% title('Control Input u(t)');
% xlabel('Time (s)');
% ylabel('u(t)');
% grid on;
%% Define Nonlinear System Dynamics with LQR Control
function dydt = nonLinear(t, y, K, M, m1, m2, l1, l2, g)
    % LQR Feedback Control
    F = -K * y; % Control input based on current state
```

```

    % Nonlinear dynamics of the system
    dydt = zeros(6,1);
    dydt(1) = y(2);
    %y(2)=xdot;
    dydt(2)=(F-(g/2)*(m1*sind(2*y(3))+m2*sind(2*y(5)))...
        -(m1*l1*(y(4)^2)*sind(y(3)))-(m2*l2*(y(6)^2)*sind(y(5))))...
        /(M+m1*((sind(y(3)))^2)+m2*((sind(y(5)))^2)); %X_DD
    %y(3)=theta1;
    dydt(3)= y(4);
    %y(4)=theta1dot;
    dydt(4)= (dydt(2)*cosd(y(3))-g*(sind(y(3))))/l1'; %theta 1 Ddot;
    %y(5)=theta2;
    dydt(5)= y(6); %theta 2D
    %y(6)=theta2dot;
    dydt(6)= (dydt(2)*cosd(y(5))-g*(sind(y(5))))/l2; %theta 2Ddot;
end
figure;
%% Simulate the Nonlinear System Using ODE45
tspan = 0:0.01:10000; % Time span for simulation
[t, y] = ode45(@(t, y) nonLinear(t, y, K, M, m1, m2, l1, l2, g), ...
    tspan, x_0);
%plotting the function output on a 2D graph
plot(t,y)
grid on
disp(eig(Acl))
```

7 Code for Observability Checks(Part-E).

```
% Observability check
% symbolic variables are defined here for solving the problem
syms m1 m2 l1 l2 g  M  real

% A matrix from state space representation
A = [0, 1, 0, 0, 0, 0;
     0, 0, -m1*g/M, 0, -m2*g/M, 0;
     0, 0, 0, 1, 0, 0;
     0, 0, (-m1*g-M*g)/(M*l1), 0, -m2*g/(M*l1), 0;
     0, 0, 0, 0, 0, 1;
     0, 0, -m1*g/(M*l2), 0, (-m2*g-M*g)/(M*l2), 0];

disp('Matrix A:');
```

```
disp(A);



% Number of states
n = size(A, 1);
fprintf('Number of states: %d\n', n);

function O = observability_matrix(A, C, n)
% Function to find the observability matrix.
%
% The observability matrix is  O = [C; C*A; C*A^2; ...; C*A^(n-1)].
%
% Inputs:
%   A - (n x n matrix) State matrix from the state-space representation.
%   C - (p x n matrix) Output matrix from the state-space representation.
%   n - (integer) Number of states.
%
% Outputs:
%   O - (p*n x n matrix) Observability matrix.

    O = C; % Initializing with first row
    for i = 1:n-1
        O = [O; C * A^i];
    end
end



function check_observability(A, C, n)
%Function to check observability
    Observability_Matrix = observability_matrix(A, C, n);
    % Display the observability matrix, commented out to make display
    % cleaner uncomment to see the observability matrix.
    % disp('Observability Matrix:');
    % disp(Observability_Matrix);

    % Compute the rank of the observability matrix
    rank_O = rank(Observability_Matrix);
    fprintf('Rank of the Observability Matrix: %d\n', rank_O);

    % Check observability
    if rank_O == n
        disp('The system is observable.');
```

```
    else
        disp('The system is not observable.');
    end
end


disp('--------------------------------------------------');

% When output vector = x(t)
% C matrix from state space representation
C = [1,0,0,0,0,0];
disp('When output vector = x(t) :')
disp('Matrix C:');
disp(C);
check_observability(A, C, n);

disp('--------------------------------------------------');

% When output vector = (1(t), 2(t))
% C matrix from state space representation
C = [0, 0, 1, 0, 0, 0;  % 1(t)
     0, 0, 0, 0, 1, 0]; % 2(t)

disp('When output vector = (1(t), 2(t)) :')
disp('Matrix C:');
disp(C);
check_observability(A, C, n);

disp('--------------------------------------------------');


% When output vector = (x(t), 2(t))
% C matrix from state space representation
C = [1, 0, 0, 0, 0, 0;  % x(t)
     0, 0, 0, 0, 1, 0]; % 2(t)
disp('When output vector = (x(t), 2(t)) :')
disp('Matrix C:');
disp(C);
check_observability(A, C, n);

disp('--------------------------------------------------');


% When output vector = (x(t), 1(t), 2(t))
```

```
% C matrix from state space representation
C = [1, 0, 0, 0, 0, 0;  % x(t)
     0, 0, 1, 0, 0, 0;  % 1(t)
     0, 0, 0, 0, 1, 0]; % 2(t)

disp('When output vector = (x(t), 1(t), 2(t)) :')
disp('Matrix C:');
disp(C);
check_observability(A, C, n);

disp('--------------------------------------------------');

```

8 Code for best Luenberger with output $(x(t))$ - Both linear and Nonlinear systems (Part-F)

```
%Luenberger design and simulations for output vector x(t)
% Parameters from section D
M = 1000;     % Mass of the cart
m1 = 100;     % Mass of pendulum 1
m2 = 100;     % Mass of pendulum 2
l1 = 20;      % Length of pendulum 1
l2 = 10;      % Length of pendulum 2
g = 9.81;     % Gravitational acceleration

% A matrix from state space representation
A = [0, 1, 0, 0, 0, 0;
     0, 0, -m1*g/M, 0, -m2*g/M, 0;
     0, 0, 0, 1, 0, 0;
     0, 0, (-m1*g-M*g)/(M*l1), 0, -m2*g/(M*l1), 0;
     0, 0, 0, 0, 0, 1;
     0, 0, -m1*g/(M*l2), 0, (-m2*g-M*g)/(M*l2), 0];

% B matrix from state space representation
B = [0;
     1/M;
     0;
     1/(M*l1);
     0;
     1/(M*l2)];

```

```
% Output vector
C = [1, 0, 0, 0, 0, 0];  % Output: x(t)

D = 0;

% Settings for simulation
t = 0:0.01:10;          % Time vector deom 0 to 10 seconds
u = ones(size(t));      % Unit step input
x0 = [0.1; 0 ; 0.1; 0; 0.1; 0];  % Initial condition for states
x_hat0 = [0; 0; 0; 0; 0; 0];    % Initial condition for observer
x0_nonlinear = [x0; zeros(6, 1)]; % Initial state for nonlinear system

tolerance = 1e-6;

% Objective function for optimization, it finds the settling time and
% sum of estimation error.end
function [settling_time, sum_estimation_error] = objective(poles, A, B, C, u, t, x0,
↪  x_hat0, tolerance)
    % Extracting poles for C
    poles_observer_C = poles;

    % Computing observer gains using pole placement method in mATLAB
    L1 = place(A', C', poles_observer_C)';

    % Simulating the actual system
    system_actual = ss(A, B, eye(6), zeros(6, 1));
    [x_actual, ~, x_states] = lsim(system_actual, u, t, x0);

    % Simulating the observer system
    A_augmented = A - L1*C;
    B_augmented = [B, L1];
    system_observer = ss(A_augmented, B_augmented, eye(6), zeros(6, 2));
    % input to the observer
    input_observer = [u(:), x_actual(:, 1)];
    [~, ~, x_hat_states_C] = lsim(system_observer, input_observer, t, x_hat0);

    % Calculating the estimation error
    error = x_states - x_hat_states_C;

    % Settling time calculation: find the when error is within tolerance of final
    ↪  value
    final_error = error(end, :);
    settling_times = zeros(1, 6);
    for i = 1:6
```

```
        error_each_state = error(:, i);
        threshold = abs(final_error(i)) * (1 + tolerance);  % Error tolerance

        % Find when the error first comes and stays within tolerance
        index_settle = find(abs(error_each_state) < threshold, 1, 'first');

        if ~isempty(index_settle)
            % Finding when the error stays within tolerance
            index_end = find(abs(error_each_state(index_settle:end)) > threshold, 1, 'first') + index_settle - 1;
            if isempty(index_end)  % If error stays within tolerance till end
                index_end = length(error_each_state);
            end
            settling_times(i) = t(index_end) - t(index_settle);  % Settling time in seconds
        end
    end
    settling_time = mean(settling_times);  % Average settling time across states

    % Sum of the estimation error values until settling
    sum_estimation_error = sum(sum(abs(error(1:index_end, :))));
end

% Sum error function for optimization (returns sum of estimation error)
function sum_estimation_error = sum_error(poles, A, B, C, u, t, x0, x_hat0, tolerance)
    [~, sum_estimation_error] = objective(poles, A, B, C, u, t, x0, x_hat0, tolerance);
end

% Randomly chosen initial poles
initial_poles = [-1, -2, -3, -4, -5, -6];

w1 = 1;  % Weight for settling time
w2 = 0.5;  % Weight for sum of estimatation errors


function penalty = pole_penalty(poles)
    penalty = 0;
    for i = 1:length(poles)
        if poles(i) < -1000
            penalty = penalty + (poles(i) + 1000)^2; % Penalty for being less than -1000
        elseif poles(i) > -0.0
```


UNIVERSITY OF MARYLAND

```
            penalty = penalty + (poles(i) + 0.1)^2; % Penalty for being greater than 0
        end
    end
end

loss_function = @(poles) w1 * objective(poles, A, B, C, u, t, x0, x_hat0, tolerance) + w2 * sum_error(poles, A, B, C, u, t, x0, x_hat0, tolerance) + pole_penalty(poles);

% Optimization settings
options = optimset('Display', 'iter', 'TolFun', 1e-6, 'TolX', 1e-6, 'MaxFunEvals', 10000, 'MaxIter', 5000);
optimized_poles = fminsearch(loss_function, initial_poles, options);



% Displaying the optimized poles
disp('Optimized poles:');
disp(optimized_poles);
% Simulating and plotting the observer's performance with the optimized poles
poles_observer_C = optimized_poles(1:6);
L1 = place(A', C', poles_observer_C)';
% Displaying the "best" Lueneberger matrix
disp('Best observer gain matrix , L:');
disp(L1)



% Augmenting the matrices
A_augmented = A - L1*C;
B_augmented = [B, L1];

% Simulating the actual system
system_actual_C = ss(A, B, eye(6), zeros(6, 1));
[x_actual_C, ~, x_states_C] = lsim(system_actual_C, u, t, x0);

% Simulating the observer system for C
system_observer = ss(A_augmented, B_augmented, eye(6), zeros(6, 2));
input_observer = [u(:), x_actual_C(:, 1)];  % Augmented input matrix
[x_hat_C, ~, x_hat_states_C] = lsim(system_observer, input_observer, t, x_hat0);

% Calculating the estimation error
error = x_states_C - x_hat_states_C;
```

```
Q = 1000*eye(6);
R = 0.1;
K = lqr(system_actual_C,Q,R);

% Plotting
figure;
set(gcf, 'Position', [100, 100, 1000, 1600]);

for i = 1:6
    subplot(6, 1, i);
    plot(t, error(:, i), 'LineWidth', 1.5);
    xlabel('Time (s)');
    if i == 1
        ylabel('Error in x(t)', 'Interpreter', 'latex');
    elseif i == 2
        ylabel('Error in $\dot{x}(t)$', 'Interpreter', 'latex');
    elseif i == 3
        ylabel('Error in $\theta_1(t)$', 'Interpreter', 'latex');
    elseif i == 4
        ylabel('Error in $\dot{\theta}_1(t)$', 'Interpreter', 'latex');
    elseif i == 5
        ylabel('Error in $\theta_2(t)$', 'Interpreter', 'latex');
    elseif i == 6
        ylabel('Error in $\dot{\theta}_2(t)$', 'Interpreter', 'latex');
    end
    grid on;
end
sgtitle('Estimation Error for output vector x(t)', 'Interpreter', 'latex');



%% Defining Nonlinear Dynamics Function
function dydt = nonLinear(~, y,K, A, C, LC, M, m1, m2, l1, l2, g)
    F = -K*y(1:6); % Control input
    dydt = zeros(12, 1);

    % Nonlinear system dynamics
    dydt(1) = y(2);
    dydt(2) = (F - (g/2) * (m1 * sin(2*y(3)) + m2 * sin(2*y(5))) ...
             - m1 * l1 * (y(4)^2) * sin(y(3)) - m2 * l2 * (y(6)^2) * sin(y(5))) ...
             / (M + m1 * (sin(y(3))^2) + m2 * (sin(y(5))^2));
    dydt(3) = y(4);
    dydt(4) = (dydt(2) * cos(y(3)) - g * sin(y(3))) / l1;
    dydt(5) = y(6);
```

```
    dydt(6) = (dydt(2) * cos(y(5)) - g * sin(y(5))) / l2;

    % Estimation error dynamics
    dydt(7:12) = (A - LC*C) * y(7:12);
end

%% Simulate Nonlinear System with C1
tspan = 0:0.1:1000;
[t, y] = ode45(@(t, y) nonLinear(t, y, K, A, C, L1, M, m1, m2, l1, l2, g), tspan, x0_nonlinear);

% Plot Results
figure;
plot(t, y(:, 1:6));
sgtitle('Nonlinear System Response with output vector $(x(t))$', 'Interpreter', 'latex');
xlabel('Time (s)');
ylabel('States');
grid on;

```

9 Code for Best Luenberger Observer with Output $(x(t), \theta_1(t), \theta_2(t))$ for Both Linear and Nonlinear Systems (Part-F)

```
%Luenberger design and simulations for output vector (x(t), 1(t) ,2(t))
% Parameters from section D
M = 1000;     % Mass of the cart
m1 = 100;     % Mass of pendulum 1
m2 = 100;     % Mass of pendulum 2
l1 = 20;      % Length of pendulum 1
l2 = 10;      % Length of pendulum 2
g = 9.81;     % Gravitational acceleration

% A matrix from state space representation
A = [0, 1, 0, 0, 0, 0;
     0, 0, -m1*g/M, 0, -m2*g/M, 0;
     0, 0, 0, 1, 0, 0;
     0, 0, (-m1*g-M*g)/(M*l1), 0, -m2*g/(M*l1), 0;
     0, 0, 0, 0, 0, 1;
     0, 0, -m1*g/(M*l2), 0, (-m2*g-M*g)/(M*l2), 0];

```

```
% B matrix from state space representation
B = [0;
     1/M;
     0;
     1/(M*l1);
     0;
     1/(M*l2)];



% Output: (x(t), 1(t) ,2(t))
C = [1, 0, 0, 0, 0, 0;    % x(t)
      0, 0, 1, 0, 0, 0;    % 1(t)
      0, 0, 0, 0, 1, 0];   % 2(t)


D = 0;

% Settings for simulation
t = 0:0.01:10;          % Time vector deom 0 to 10 seconds
u = ones(size(t));      % Unit step input
x0 = [0.1; 0 ; 0.1; 0; 0.1; 0];  % Initial condition for states
x_hat0 = [0; 0; 0; 0; 0; 0];    % Initial condition for observer
x0_nonlinear = [x0; zeros(6, 1)]; % Initial state for nonlinear system

tolerance = 1e-6;

% Objective function for optimization, it finds the settling time and
% sum of estimation error.
function [settling_time, sum_estimation_error] = objective(poles, A, B, C, u, t, x0, x_hat0, tolerance)
    % Extracting poles for C
    poles_observer_C = poles;

    % Computing observer gains using pole placement method in mATLAB
    L1 = place(A', C', poles_observer_C)';

    % Simulating the actual system
    system_actual = ss(A, B, eye(6), zeros(6, 1));
    [x_actual, ~, x_states] = lsim(system_actual, u, t, x0);

    % Augmenting the matrices
    A_augmented = A - L1*C;
    B_augmented = [B, L1];
```

```
    % Defining the combined input matrix for observer system (including x, 1, 2)
    input_observer = [u(:), x_actual(:, 1), x_actual(:, 3), x_actual(:, 5)];
    system_observer = ss(A_augmented, B_augmented, eye(6), zeros(6, 4));
    [~, ~, x_hat_states] = lsim(system_observer, input_observer, t, x_hat0);


    % Calculating the estimation error
    error = x_states - x_hat_states;


    % Settling time calculation: find the when error is within tolerance of final value
    final_error = error(end, :);
    settling_times = zeros(1, 6);
    for i = 1:6
        error_each_state = error(:, i);
        threshold = abs(final_error(i)) * (1 + tolerance);  % Error tolerance

        % Find when the error first comes and stays within tolerance
        index_settle = find(abs(error_each_state) < threshold, 1, 'first');

        if ~isempty(index_settle)
            % Finding when the error stays within tolerance
            index_end = find(abs(error_each_state(index_settle:end)) > threshold, 1, 'first') + index_settle - 1;
            if isempty(index_end)  % If error stays within tolerance till end
                index_end = length(error_each_state);
            end
            settling_times(i) = t(index_end) - t(index_settle);  % Settling time in seconds
        end
    end
    settling_time = mean(settling_times);  % Average settling time across states

    % Sum of the estimation error values until settling
    sum_estimation_error = sum(sum(abs(error(1:index_end, :))));
end

% Sum error function for optimization (returns sum of estimation error)
function sum_estimation_error = sum_error(poles, A, B, C, u, t, x0, x_hat0, tolerance)
    [~, sum_estimation_error] = objective(poles, A, B, C, u, t, x0, x_hat0, tolerance);
end
```

```
% Randomly chosen initial poles
initial_poles = [-1, -2, -3, -4, -5, -6];

w1 = 1;  % Weight for settling time
w2 = 0.5;  % Weight for sum of error



function penalty = pole_penalty(poles)
    penalty = 0;
    for i = 1:length(poles)
        if poles(i) < -1000
            penalty = penalty + (poles(i) + 1000)^2; % Penalty for being less than -1000
        elseif poles(i) > -0.0000
            penalty = penalty + (poles(i) + 0.1)^2; % Penalty for being greater than 0
        end
    end
end

loss_function = @(poles) w1 * objective(poles, A, B, C, u, t, x0, x_hat0, tolerance) + w2 * sum_error(poles, A, B, C, u, t, x0, x_hat0, tolerance) + pole_penalty(poles);

% Optimization settings
options = optimset('Display', 'iter', 'TolFun', 1e-6, 'TolX', 1e-6, 'MaxFunEvals', 10000, 'MaxIter', 5000);
optimized_poles = fminsearch(loss_function, initial_poles, options);



% Displaying the optimized poles
disp('Optimized poles:');
disp(optimized_poles);
% Simulating and plottting the observer's performance with the optimized poles
poles_observer_C = optimized_poles(1:6);
L1 = place(A', C', poles_observer_C)';
% Displaying the "best" Lueneberger matrix
disp('Best observer gain matrix , L:');
disp(L1)
```

```
%% Simulating observer
A_augmented = A - L1*C;
B_augmented = [B, L1];  % Augmenting B matrix to include observer feedback

% Simulating the actual system
system_actual_C = ss(A, B, eye(6), zeros(6, 1));
[x_actual_C, ~, x_states_C] = lsim(system_actual_C, u, t, x0);

% Defining the combined input matrix for observer system (including x, 1, 2)
input_observer = [u(:), x_actual_C(:, 1), x_actual_C(:, 3), x_actual_C(:, 5)];  % Augmented with x, 1, 2
% Simulating observer system for C2
sys_observer = ss(A_augmented, B_augmented, eye(6), zeros(6, 4));
[~, ~, x_hat_states] = lsim(sys_observer, input_observer, t, x_hat0);


% Calculating the estimation error
error = x_states_C - x_hat_states;
Q = 1000*eye(6);
R = 0.1;
K = lqr(system_actual_C,Q,R);

% Plotting
figure;
set(gcf, 'Position', [100, 100, 1000, 1600]);

for i = 1:6
    subplot(6, 1, i);
    plot(t, error(:, i), 'LineWidth', 1.5);
    xlabel('Time (s)');
    if i == 1
        ylabel('Error in x(t)', 'Interpreter', 'latex');
    elseif i == 2
        ylabel('Error in $\dot{x}(t)$', 'Interpreter', 'latex');
    elseif i == 3
        ylabel('Error in $\theta_1(t)$', 'Interpreter', 'latex');
    elseif i == 4
        ylabel('Error in $\dot{\theta}_1(t)$', 'Interpreter', 'latex');
    elseif i == 5
        ylabel('Error in $\theta_2(t)$', 'Interpreter', 'latex');
    elseif i == 6
        ylabel('Error in $\dot{\theta}_2(t)$', 'Interpreter', 'latex');
```

```
    end
    grid on;
end
sgtitle('Estimation Error for output vector $(x(t),\theta_1(t),\theta_2(t))$', 'Interpreter', 'latex');



%% Define Nonlinear Dynamics Function
function dydt = nonLinear(~, y, K, A, C, LC, M, m1, m2, l1, l2, g)
    F = -K*y(1:6); % Control input
    dydt = zeros(12, 1);

    % Nonlinear system dynamics
    dydt(1) = y(2);
    dydt(2) = (F - (g/2) * (m1 * sin(2*y(3)) + m2 * sin(2*y(5))) ...
              - m1 * l1 * (y(4)^2) * sin(y(3)) - m2 * l2 * (y(6)^2) * sin(y(5))) ...
              / (M + m1 * (sin(y(3))^2) + m2 * (sin(y(5))^2));
    dydt(3) = y(4);
    dydt(4) = (dydt(2) * cos(y(3)) - g * sin(y(3))) / l1;
    dydt(5) = y(6);
    dydt(6) = (dydt(2) * cos(y(5)) - g * sin(y(5))) / l2;

    % Estimation error dynamics
    dydt(7:12) = (A - LC*C) * y(7:12);
end

%% Simulate Nonlinear System with C1
tspan = 0:0.1:1000;
[t, y] = ode45(@(t, y) nonLinear(t, y, K, A, C, L1, M, m1, m2, l1, l2, g), tspan, x0_nonlinear);

% Plot Results
figure;
plot(t, y(:, 1:6));
sgtitle('Nonlinear System Response with output vector $(x(t),\theta_1(t),\theta_2(t))$', 'Interpreter', 'latex');
xlabel('Time (s)');
ylabel('States');
grid on;
```

10 Code for LQG Controller(Part-G).

```
%% Initializing the Variables
M = 1000; % Mass of the cart
m1 = 100; % Mass of the 1st pendulum
m2 = 100; % Mass of the 2nd pendulum
l1 = 20;  % Length of the 1st pendulum
l2 = 10;  % Length of the 2nd pendulum
g = 9.81; % Acceleration due to gravity

%% Defining the A, B and C matrices
A = [0 1 0 0 0 0;
    0 0 -m1*g/M 0 -m2*g/M 0;
    0 0 0 1 0 0;
    0 0 (-m1*g-M*g)/(M*l1) 0 -m2*g/(M*l1) 0;
    0 0 0 0 0 1;
    0 0 -m1*g/(M*l2) 0 (-m2*g-M*g)/(M*l2) 0];

B = [0;
    1/M;
    0;
    1/(M*l1);
    0;
    1/(M*l2)];

C1 = [1, 0, 0, 0, 0, 0]; % Observing only x(t)
C2 = [1, 0, 0, 0, 1, 0; 0, 0, 0, 0, 1, 0]; % Observing x(t) and theta_1(t)
C3 = [1, 0, 1, 0, 1, 0; 0, 0, 1, 0, 0, 0; 0, 0, 0, 0, 1, 0]; % Observing x(t), theta_1(t), theta_2(t)

D = 0;

%% Define noise covariances for Kalman filter
W = 0.01 * eye(size(A)); % Process noise
V1 = 0.01 * eye(size(C1, 1)); % Measurement noise for C1
V2 = 0.01 * eye(size(C2, 1)); % Measurement noise for C2
V3 = 0.01 * eye(size(C3, 1)); % Measurement noise for C3

%% Initial state
x_0 = [0; 0; 0.1; 0; 0.2; 0]; % Initial state for linear systems
x_0_nonlinear = [x_0; zeros(6, 1)]; % Initial state for nonlinear system

%% Design LQR Controller
```

```
Q = 1000 * eye(6); % State weights
R = 0.1; % Control input weight
K = lqr(A, B, Q, R);

%% Design Kalman Filter Gains
LC1 = lqe(A, W, C1, W, V1);
LC2 = lqe(A, W, C2, W, V2);
LC3 = lqe(A, W, C3, W, V3);

%% Combine LQR and Kalman Gains for C1
A_lqg_C1 = [A - B*K, zeros(size(A)); LC1*C1, A - LC1*C1];
B_lqg = [B; zeros(size(B))];
C_lqg_C1 = [C1, zeros(size(C1))];
sys_C1 = ss(A_lqg_C1, B_lqg, C_lqg_C1, D);

% Simulate Initial Response and Step Response for C1
figure;
initial(sys_C1, [x_0; zeros(size(x_0))]);
title('Initial Response for C1');
grid on;

figure;
step(sys_C1);
title('Step Response for C1');
grid on;

%% Combine LQR and Kalman Gains for C2
A_lqg_C2 = [A - B*K, zeros(size(A)); LC2*C2, A - LC2*C2];
C_lqg_C2 = [C2, zeros(size(C2))];
sys_C2 = ss(A_lqg_C2, B_lqg, C_lqg_C2, D);

% Simulate Initial Response and Step Response for C2
figure;
initial(sys_C2, [x_0; zeros(size(x_0))]);
title('Initial Response for C2');
grid on;

figure;
step(sys_C2);
title('Step Response for C2');
grid on;

%% Combine LQR and Kalman Gains for C3
A_lqg_C3 = [A - B*K, zeros(size(A)); LC3*C3, A - LC3*C3];
```

```
C_lqg_C3 = [C3, zeros(size(C3))];
sys_C3 = ss(A_lqg_C3, B_lqg, C_lqg_C3, D);

% Simulate Initial Response and Step Response for C3
figure;
initial(sys_C3, [x_0; zeros(size(x_0))]);
title('Initial Response for C3');
grid on;

figure;
step(sys_C3);
title('Step Response for C3');
grid on;

%% Define Nonlinear Dynamics Function
function dydt = nonLinear(t, y, A, C, LC, K, M, m1, m2, l1, l2, g)
    F = -K * y(1:6); % Control input
    dydt = zeros(12, 1);

    % Nonlinear system dynamics
    dydt(1) = y(2);
    dydt(2) = (F - (g/2) * (m1 * sin(2*y(3)) + m2 * sin(2*y(5))) ...
             - m1 * l1 * (y(4)^2) * sin(y(3)) - m2 * l2 * (y(6)^2) * sin(y(5))) ...
             / (M + m1 * (sin(y(3))^2) + m2 * (sin(y(5))^2));
    dydt(3) = y(4);
    dydt(4) = (dydt(2) * cos(y(3)) - g * sin(y(3))) / l1;
    dydt(5) = y(6);
    dydt(6) = (dydt(2) * cos(y(5)) - g * sin(y(5))) / l2;

    % Estimation error dynamics
    dydt(7:12) = (A - LC*C) * y(7:12);
end

%% Simulate Nonlinear System with C1
tspan = 0:0.1:100;
[t, y] = ode45(@(t, y) nonLinear(t, y, A, C1, LC1, K, M, m1, m2, l1, l2, g), tspan,
↪  x_0_nonlinear);

% Plot Results
figure;
plot(t, y(:, 1:6));
title('Nonlinear System Response with C1');
xlabel('Time (s)');
ylabel('States');
```

```
grid on;

```