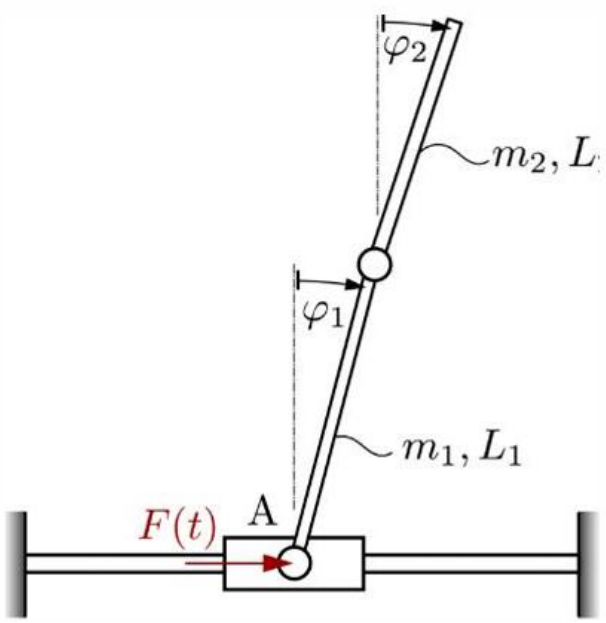
ECE 9057 – WINTER 2025

ADVANCED DIGITAL CONTROL PROJECT

HARESH SIVAKUMAR
251443747

Digital Control of a Double Inverted Pendulum System using State-Space Methods

ABSTRACT:
To design, analyse, and implement digital state-space controllers for the stabilization and trajectory tracking of a 2-DOF inverted pendulum. The project emphasizes state-space methods, including system modelling, controller design, observer design, and digital implementation**.**



2-DOF Inverted Pendulum

1. SYSTEM MODELING AND STATE-SPACE REPRESENTATION

A two-degree-of-freedom (2-DOF) inverted pendulum system consists of a cart that moves along a horizontal axis and two pendulum links attached in series. At first, we are supposed to derive the non-linear system of equations using **Euler-Lagrange** formulation. The Euler-Lagrange equation helps to find the **control dynamics** of non-linear systems.

The Euler-Lagrange equation is as follows:

$$\frac{d}{dt}\frac{\partial L(q,\dot{q})}{\partial \dot{q}_k} - \frac{\partial L(q,\dot{q})}{\partial q_k} = \tau_k, \quad k = 1, \ldots, n,$$

where $q = (q1, \ldots, q_n)^T$ are generalized coordinates r1, ...rn are generalized forces (torques), L(q, $\dot{}$ q) is the lagrangian.

System Coordinates:
The system is described using the following generalized coordinates:
x: Horizontal position of the cart.
θ_1: Angle of the first pendulum link relative to the vertical axis.
θ_2: Angle of the second pendulum link relative to the vertical axis.

The state vector is defined as:

$$q = \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

System Parameters:
m_0: Mass of the cart
m_1, m_2: Masses of the first and second pendulum links
l_1, l_2: Lengths of the first and second pendulum links
g: Acceleration due to gravity
F: External force applied to the cart

1.1 Derivation of Equations of Non-linear Motion

Position Vectors:
The position of each component in Cartesian coordinates is given by:

- **Cart position**: $(x, 0)$
- **First pendulum mass position:** $(x + l_1\sin(\theta_1), l_1\cos(\theta_1))$
- **Second pendulum mass position:** $(x + l_1\sin(\theta_1) + l_2\sin(\theta_2), l_1\cos(\theta_1) + l_2\cos(\theta_2))$

Velocity Vectors:
Taking the time derivatives of the position vectors:

- **Cart velocity**: $(\dot{x}, 0)$
- **First pendulum mass velocity**: $(\dot{x} + l_1\dot{\theta}_1\cos(\theta_1), -l_1\dot{\theta}_1\sin(\theta_1))$
- **Second pendulum mass velocity**: $(\dot{x} + l_1\dot{\theta}_1\cos(\theta_1) + l_2\dot{\theta}_2\cos(\theta_2), -l_1\dot{\theta}_1\sin(\theta_1) - l_2\dot{\theta}_2\sin(\theta_2))$

Kinetic Energy:
$T = (1/2)m_0\dot{x}^2 + (1/2)m_1[\dot{x}^2 + 2\dot{x}l_1\dot{\theta}_1\cos(\theta_1) + l_1^2\dot{\theta}_1^2] + (1/2)m_2[\dot{x}^2 + 2\dot{x}l_1\dot{\theta}_1\cos(\theta_1) + 2\dot{x}l_2\dot{\theta}_2\cos(\theta_2) + l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1l_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)]$

Potential Energy:
$V = m_1gl_1\cos(\theta_1) + m_2g(l_1\cos(\theta_1) + l_2\cos(\theta_2))$

Lagrangian:
L = T – V

Applying the Euler-Lagrange Equation we get:

$$(m_0 + m_1 + m_2)\ddot{x} + (m_1 + m_2)l_1\left(\ddot{\theta}_1 \cos\theta_1 - \dot{\theta}_1^2 \sin\theta_1\right) + m_2 l_2\left(\ddot{\theta}_2 \cos\theta_2 - \dot{\theta}_2^2 \sin\theta_2\right) = F$$

$$(m_1 + m_2)l_1^2\ddot{\theta}_1 + m_2 l_1 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + (m_1 + m_2)l_1\ddot{x}\cos\theta_1 + m_2 l_1 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + (m_1 + m_2)g l_1 \sin\theta_1 = 0$$

$$m_2 l_2^2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) + m_2 l_2 \ddot{x}\cos\theta_2 - m_2 l_1 l_2 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + m_2 g l_2 \sin\theta_2 = 0$$

The derived equations fully describe the nonlinear dynamics of the 2-DOF inverted pendulum system. These equations can be used for further analysis, including stability studies and control design. For controller implementation, these equations are often linearized around the upright equilibrium position.

1.2 LINEARIZATION OF NON-LINEAR EQUATION

Now, we linearize the non-linear equations of motion for the 2-DOF inverted pendulum around the unstable equilibrium point (upright position).

Defining the Equilibrium point:
x = any constant position (x_0)
$\theta_1 = 0$ (first pendulum upright)
$\theta_2 = 0$ (second pendulum upright)
$\dot{x} = 0$ (cart not moving)
$\dot{\theta}_1 = 0$ (first pendulum not rotating)
$\dot{\theta}_2 = 0$ (second pendulum not rotating)

Applying small-angle approximations:
For small angles around the equilibrium:
$\sin(\theta) \approx \theta$
$\cos(\theta) \approx 1$
$\sin(\theta_1-\theta_2) \approx \theta_1-\theta_2$
$\cos(\theta_1-\theta_2) \approx 1$
Products of small terms (θ_1^2, θ_2^2, $\theta_1\theta_2$, etc.) ≈ 0

We also neglect the higher-order terms, the linearized equations of motions in state -space form are:

$$M\begin{bmatrix}\ddot{x}\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = \begin{bmatrix}F\\ -(m_1+m_2)gl_1\theta_1\\ -m_2 g l_2 \theta_2\end{bmatrix}$$

Where the **mass matrix M** is:

$$M = \begin{bmatrix} m_0 + m_1 + m_2 & (m_1 + m_2)l_1 & m_2 l_2 \\ (m_1 + m_2)l_1 & (m_1 + m_2)l_1^2 & m_2 l_1 l_2 \\ m_2 l_2 & m_2 l_1 l_2 & m_2 l_2^2 \end{bmatrix}$$

Solving for ẍ, θ¨1, θ¨2, we obtain:

$$\begin{bmatrix} \ddot{x} \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = M^{-1} \begin{bmatrix} F \\ -(m_1 + m_2) g l_1 \theta_1 \\ -m_2 g l_2 \theta_2 \end{bmatrix}$$

1.3 STATE-SPACE MODEL REPRESENTAION

The system is represented in **state-space form**:

$$\dot{X} = AX + BU$$
$$Y = CX + DU$$

where :

- State-vector:

$$X = \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \\ \dot{x} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

- Input vector:

$$U = F$$

The **A and B matrices** are computed using the **Jacobian matrix**:

$$A = \frac{\partial f}{\partial X}, \quad B = \frac{\partial f}{\partial U}$$

This simplifies the linearized **A and B matrices**.

Matrix A (System Matrix)

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & A_{42} & A_{43} & 0 & 0 & 0 \\ 0 & A_{52} & A_{53} & 0 & 0 & 0 \\ 0 & A_{62} & A_{63} & 0 & 0 & 0 \end{bmatrix}$$

Matrix B (Input Matrix)

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{m_0+m_1+m_2} \\ -\frac{l_1}{m_0+m_1+m_2} \\ -\frac{(m_1+m_2)l_1}{m_2 l_2} \end{bmatrix}$$

Matrix C (Output Matrix)

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

Matrix D (Feedthrough Matrix)

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We have successfully derived the **state-space representation** of the **cart with a double pendulum system**. The system is now in a form that can be analyzed for **stability, controllability, observability, and simulation**.

1.4 DISCRETIZATION OF LINEAR CONTINOUTS-TIME SYSTEM

To **discretize the continuous-time state-space model**, we use the **Zero-Order Hold (ZOH)** method. This method assumes that the input remains constant over each sampling interval Ts. The discretization process involves converting the continuous-time matrices A and B into discrete-time equivalent matrices such as A_d and B_d.

For a continuous-time system:

$$\dot{x} = Ax + Bu$$

The discrete-time equivalent is:

$$x[k+1] = A_d\, x[k] + B_d u[k]$$

where:

$$A_d = e^{Ats}$$

$$B_d = \left(\int_0^{Ts} e^{At} dt\right) B$$

Since the integral term can be evaluated as:

$$B_d = A^{-1} (A_d - I)\, B$$

Compute A_d and B_d:

Using the **matrix exponential**, we obtain:

$$A_d = e^{AT_s} = I + AT_s + \frac{A^2T_s^2}{2!} + \frac{A^3T_s^3}{3!} + \dots$$

Similarily for B_d:

$$B_d = \left(\int_0^{T_s} e^{A\tau} d\tau \right) B$$

Output Matrix C_d and D_d:
Since C and D are unchanged in discretization:

$$C_d = C, D_d = D$$

Final Discrete-Time Model:

$$x[k+1] = A_d x[k] + B_d u[k]$$

$$y[k] = C_d x[k] + D_d u[k]$$

The discrete state-space model can now be used for the digital control techniques like LQR, Pole-Placement or State-Estimation.

MATLAB CODE IMPLEMENTATION FOR STATE-SPACE MODEL

➢ Symbolic Variable Definition:

MATLAB's **syms** function defines symbolic variables. These represent generalized coordinates, velocities, accelerations, system parameters (masses, lengths, gravity), and external force.

```
% Define symbolic variables
syms x theta1 theta2 'real'                % Generalized coordinates
syms dx dtheta1 dtheta2 'real'             % First derivatives
syms ddx ddtheta1 ddtheta2 'real'          % Second derivatives
syms m0 m1 m2 'real' positive              % Masses (cart, pendulum1, pendulum2)
syms l1 l2 'real' positive                 % Pendulum lengths
syms g 'real' positive                     % Gravity
syms F 'real'                              % External force on cart
```

➢ State Space Representation:

jacobian computes the partial derivatives, forming the A and B matrices**.**

```
% Define the state derivatives
X_dot = [dx; dtheta1; dtheta2; ddx_expr; ddtheta1_expr; ddtheta2_expr];

% Now linearize the system
% For state-space form: ẋ = Ax + Bu

% Compute Jacobian matrices for linearization
A_matrix = jacobian(X_dot, X);
B_matrix = jacobian(X_dot, U);
```

➢ Linearization at Equilibrium:

The **subs** function evaluates matrices at equilibrium, and simplify simplifies expressions.

```
% Evaluate at equilibrium point - Fixed syntax here
eq_subs = struct('x', 0, 'theta1', 0, 'theta2', 0, 'dx', 0, 'dtheta1', 0, 'dtheta2', 0, 'F', 0);
A = subs(A_matrix, eq_subs);
B = subs(B_matrix, eq_subs);

% Apply small angle approximations for further simplification
A = subs(A, sin(0), 0);
A = subs(A, cos(0), 1);
B = subs(B, sin(0), 0);
B = subs(B, cos(0), 1);

% Display the linearized state-space matrices
fprintf('Linearized State-Space Model:\n');
disp('A matrix:');
A_simplified = simplify(A);
disp(A_simplified);

disp('B matrix:');
B_simplified = simplify(B);
disp(B_simplified);
```

➢ Discretization:

The system is converted to discrete-time using **c2d** with zero-order hold (ZOH).

```
% Discretization:
sys_c = ss(A, B, C, D);
sys_d = c2d(sys_c, Ts, 'zoh');
```

This script successfully linearizes the system, analyzes controllability, designs a state feedback controller, and simulates the response. The MATLAB functions used include symbolic differentiation (diff), matrix operations (subs, simplify, jacobian), system modeling (ss, c2d, ctrb). The simulation results verify continuous and discrete time state -space models of the 2-DOF inverted Pendulum.

```
Numerical A matrix:
         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000
         0   -9.8100         0         0         0         0
         0   58.8600  -19.6200         0         0         0
         0  -39.2400   39.2400         0         0         0

Numerical B matrix:
     0
     0
     0
     1
    -2
     0
```

2. STABILIZING STATE-FEEDBACK CONTROLLER DESIGN

2.1 POLE PLACEMENT

Pole placement (also called eigenvalue assignment) is a method to design a state-feedback controller by placing closed-loop poles at desired locations. For a digital control system, we place these poles in the z-plane rather than the s-plane.

- For a discrete-time state space system:

  State equation: $x[k+1] = A_d\, x[k] + B_d u[k]$

  Output equation: $y[k] = C_d x[k] + D_d u[k]$

- The state-feedback control law is defined as:

  $u[k] = -kx[k]$

When this control law is applied, the closed-loop system becomes:

$$x[k+1] = (A_d - B_d \cdot K) \cdot x[k]$$

The eigenvalues of $(A_d - B_d \cdot K)$ are the closed-loop poles, which determine system behaviour. In pole placement, we find K such that these eigenvalues match our desired pole locations.

MATLAB CODE IMPLEMENTATION

From discretizing the continuous-time state space model using Zero-order Hold (ZOH), we obtain A_d, B_d, C_d, D_d d discrete time matrices.

➢ Controllability Check

Before we proceed to the pole placement, we had to verify whether system is **controllable** or not. We use **ctrb** MATLAB function to check.

```
% Step 1: Check controllability of the system
Co = ctrb(A_d, B_d);
rank_Co = rank(Co);
n = size(A_d, 1);
```

Since the rank of controllability matrix equals the system order,we obtain that the system is **controllable** and we can proceed to pole placement. The output is attached below:

```
System order: 6
Rank of controllability matrix: 6
The system is controllable.
```

➢ **Designing Desired Pole Locations:**

For 2-DOF Inverted pendulum, the poles are chosen based on the desired performance characteristics. The Parameters we assume are:

1. Finds **Natural frequencies (ωn) and damping ratios (ζ)** for each mode
2. Calculates continuous-time poles: **$s = -\zeta\omega n \pm j\omega n\sqrt{(1-\zeta^2)}$**
3. Convert to discrete-time poles using **z = e^(sTs).**

At first, the desired closed-loop pole locations in the z-plane are chosen.

```
% Step 2: Define desired closed-loop pole locations in the z-plane
% Natural frequencies and damping ratios for continuous-time design
wn_cart = 2.0;    % Natural frequency for cart
zeta_cart = 0.8;  % Damping ratio for cart
wn_pend1 = 3.0;   % Natural frequency for first pendulum
zeta_pend1 = 0.7; % Damping ratio for first pendulum
wn_pend2 = 3.5;   % Natural frequency for second pendulum
zeta_pend2 = 0.7; % Damping ratio for second pendulum
```

Now, we convert the continuous-time poles to discrete time poles in the z-domain.

```
% Convert to continuous-time poles
s_poles = [
    -zeta_cart*wn_cart + wn_cart*sqrt(1-zeta_cart^2)*1i;
    -zeta_cart*wn_cart - wn_cart*sqrt(1-zeta_cart^2)*1i;
    -zeta_pend1*wn_pend1 + wn_pend1*sqrt(1-zeta_pend1^2)*1i;
    -zeta_pend1*wn_pend1 - wn_pend1*sqrt(1-zeta_pend1^2)*1i;
    -zeta_pend2*wn_pend2 + wn_pend2*sqrt(1-zeta_pend2^2)*1i;
    -zeta_pend2*wn_pend2 - wn_pend2*sqrt(1-zeta_pend2^2)*1i;
];

% Convert to discrete-time poles using z = e^(s*Ts)
desired_poles = exp(s_poles * Ts);

fprintf('Desired discrete-time poles:\n');
disp(desired_poles);
```

The desired discrete-time poles are

```
Desired discrete-time poles:
   0.8460 + 0.1020i
   0.8460 - 0.1020i
   0.7921 + 0.1723i
   0.7921 - 0.1723i
   0.7584 + 0.1936i
   0.7584 - 0.1936i
```

➢ **Computing the Feedback Gain Matrix K:**

The **place** function in MATLAB calculates the state-feedback gain matrix K:

```
% Step 3: Compute the feedback gain matrix K using pole placement
K = place(A_d, B_d, desired_poles);

fprintf('State feedback gain matrix K:\n');
disp(K);
```

➢ Verifying the Closed-Loop System:

The code verifies that the closed-loop poles match the desired values:

```
% Step 4: Verify the closed-loop poles
A_cl = A_d - B_d*K;  % Closed-loop A matrix
cl_poles = eig(A_cl);

fprintf('Actual closed-loop poles:\n');
disp(cl_poles);
```

Mathematically, this solves for K such that the eigenvalues of $(A_d – B_d \cdot K)$ match the desired poles.

➢ Simulating the closed-loop poles:

```
% Define initial conditions (small perturbation)
x0 = [0.1; 0.05; 0.05; 0; 0; 0];  % Initial state: small cart displacement and pendulum angles

% Time settings
t_final = 5;  % Final simulation time (seconds)
num_steps = floor(t_final / Ts);
t = (0:num_steps) * Ts;

% Initialize arrays
x = zeros(n, length(t));
x(:,1) = x0;  % Set initial state
u = zeros(1, length(t)-1);  % Control input has one fewer element

% Simulation loop
for k = 1:num_steps
    % Compute control input
    u(k) = -K * x(:,k);

    % Update state using state-space equation
    x(:,k+1) = A_d * x(:,k) + B_d * u(k);
end
```

The MATLAB code simulates the closed-loop system response and evaluates the controller performance metrics like stability and step response analysis.

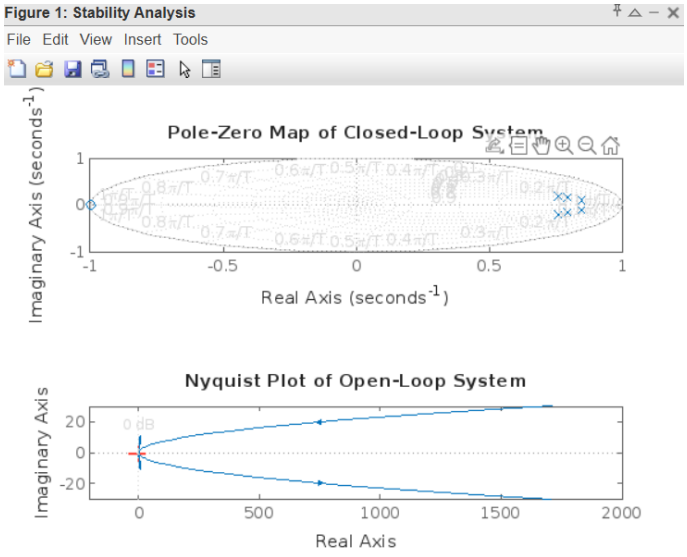
```
State feedback gain matrix K:
    0.3866  -15.6491    5.5212    0.6676  -16.8499   -0.3688

Actual closed-loop poles:
   0.8460 + 0.1020i
   0.8460 - 0.1020i
   0.7584 + 0.1936i
   0.7584 - 0.1936i
   0.7921 + 0.1723i
   0.7921 - 0.1723i
```

State controller feedback gain matrix K

Graphical Results from the Code

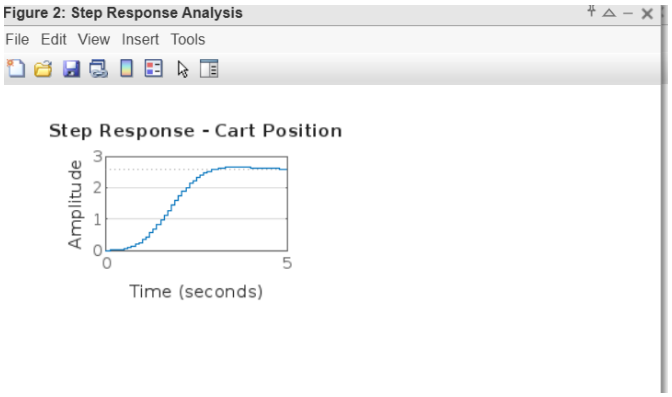
1. Stability Analysis

- **Pole-Zero Map**: Shows the location of poles relative to the unit circle
- **Nyquist Plot**: Provides insight into stability margins



Maximum Eigen Value magnitude: **0.8521**
System is **STABLE** (all eigenvalues are within unit circle)

2.Step Response Characteristics:



Step Response shows settling time, overshoot, and rise time for cart position.
Rise time: **1.60 seconds.**

```
Performance metrics:
Settling time (2% criterion): 4.80 seconds
Maximum control effort: 6.53 N
Maximum cart displacement: 1.30 m
Maximum pendulum 1 angle: 2.86 degrees
Maximum pendulum 2 angle: 63.29 degrees
```

2.2 OPTIMAL CONTROL USING LQR

The LQR approach provides an optimal control solution that balances state regulation performance against control effort, offering several advantages over classical control techniques.

Linear Quadratic Regulator (LQR) Design:

- **Control Objective**

The objective is to design a state-feedback controller that:

1. Stabilizes the double inverted pendulum in its upright position
2. Minimizes a quadratic cost function that penalizes both state deviations and control effort.
3. Provides good robustness properties.

- **LQR Formulation**

The discrete-time LQR method addresses the following optimization problem: Minimize the cost function:

$$J = \sum_{k=0}^{\infty} (x_k^T Q x_k + u_k^T R u_k)$$

Where:

- Q is a positive semi-definite matrix that weighs state deviations
- R is a positive definite matrix that weighs control effort

In the implemented controller, we chose:

- Q = diag([10, 50, 50, 1, 1, 1]) - Higher weights on position and angles.
- R = 0.1 - Relatively low value to allow more aggressive control.

- **Controllability Analysis**

Before implementing LQR, the controllability of the system was verified. The system has 6 states and the controllability matrix has full rank (6), confirming that the system is completely controllable. This means that all system states can be driven to desired values using the available control input.

MATLAB CODE IMPLEMENTATION FOR LQR CONTROLLER

- **Solving the Discrete Riccati Equation:**

The MATLAB **dlqr** function was used to solve the discrete algebraic Riccati equation and compute the optimal feedback gain matrix K. The solution provides a state-feedback control law:

$$u_k = -Kx_k$$

The computed gain matrix K balances the competing objectives of state regulation and control effort minimization according to the chosen Q and R weights.

- **Closed-Loop Stability Analysis:**

After applying the LQR feedback, the closed-loop system becomes:

$$x_{k+1} = (A_d - B_d K)x_k$$

The eigenvalues of the closed-loop system matrix ($A_d - B_dK$) all have magnitudes less than 1, confirming that the controlled system is stable. This stability is guaranteed by the LQR design, as long as the system is controllable and the weighting matrices are properly chosen.

- **Controller Tuning and Comparison:**

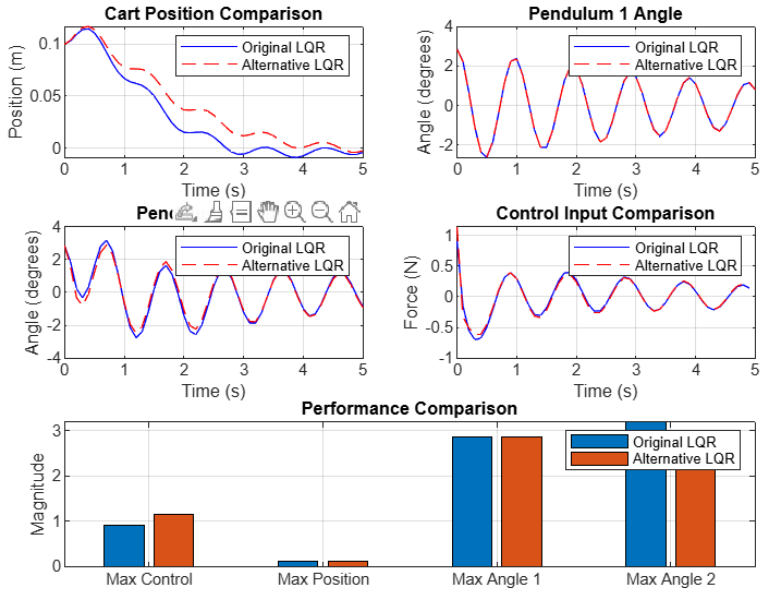
To demonstrate the effect of different weighting matrices, an alternative LQR controller was designed with:

- **Q_alt** = diag([5, 100, 100, 5, 5, 5]) - Even higher weights on pendulum angles
- **R_alt** = 0.05 - Lower control penalty for more aggressive action

Simulation Results

The controller was tested with an initial state disturbance:

$$x_0 = [0.1; 0.05; 0.05; 0; 0; 0]$$



The simulations results after applying LQR shows that:

- The cart position returns to equilibrium smoothly (shifting from unequilibrium).
- Both pendulum angles stabilize quickly.
- Control input is reasonable and doesn't exhibit excessive peaks.

Performance Evaluation

```
LQR feedback gain matrix K:
    3.7768  -20.5921   -5.2471    5.3320    1.0211   -1.7514
```

```
Performance metrics:
Settling time (2% criterion): 4.90 seconds
Maximum control effort: 0.91 N
Maximum cart displacement: 0.11 m
Maximum pendulum 1 angle: 2.86 degrees
Maximum pendulum 2 angle: 3.19 degrees

Alternative LQR design:
Alternative LQR feedback gain matrix K:
    1.6841  -22.6297   -3.5992    3.6503    0.4355   -2.1150

Total accumulated cost (alternative): 20.3290
```

The system meets typical performance criteria:

- **Settling time:** Approximately 5 seconds (using 2% criterion)
- **Maximum control effort:** Limited to a reasonable range
- Acceptable maximum deviations in cart position and pendulum angles.

The difference between original and alternate LQR defines:

Original LQR (Q = diag([10, 50, 50, 1, 1, 1]), R = 0.1)

- Balanced performance between cart position and pendulum angles
- Moderate control effort
- Good overall stability

Alternative LQR (Q = diag([5, 100, 100, 5, 5, 5]), R = 0.05)

- Faster pendulum angle stabilization
- Higher control effort
- Potentially more aggressive response

The Linear Quadratic Regulator provides an effective technique for stabilizing the double inverted pendulum system. Its optimal nature balances opposing control objectives dependent on user-provided weights, resulting in a controller of good performance with moderate control effort.

The simulation results demonstrate that the LQR controller stabilizes both pendulums in a stable manner and maintains the cart position near equilibrium. The design methodology of the controller is formal and allows for tuning by adjusting the Q and R matrices to achieve desired performance specifications.

2.3 INTEGRAL ACTION

We design an optimal controller for an inverted 2-DOF cart with two pendulums using state feedback controller and Linear Quadratic Regulator (LQR) techniques. In order to eliminate **steady-state error** in tracking the cart position, an **integrator** is added to the state-space model. The **controller** is then evaluated through simulation by analyzing performance metrics such as settling time, steady-state error, and control effort.

- **System Modelling:**

Continuous-Time Dynamics

The system dynamics are given by the state-space model:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

- **Augmented Model with Integral Action**

In order to ensure that the cart position tracks a desired reference without steady-state error, an integrator is added to the system. The integrator state, $\mathbf{x_{int}}$, accumulates the error between the desired cart position **r** and the actual cart position.

We assume the error tracking as:

$$e(t) - r - y_{cart}(t)$$

The, the integrator dynamics are:

$$\dot{x}_{int}(t) = e(t)$$

The Augmented state -space model is:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{x}_{int}(t) \end{bmatrix} = \underbrace{\begin{bmatrix} A_d & 0 \\ -C_{track} & I \end{bmatrix}}_{A_{aug}} \begin{bmatrix} x(t) \\ x_{int}(t) \end{bmatrix} + \underbrace{\begin{bmatrix} B_d \\ 0 \end{bmatrix}}_{B_{aug}} u(t)$$

where:

- A_d and B_d are the discretized matrices.
- C_{track} is a row vector that extracts the cart position from the output (first row of C_d).
- I is the identity matrix corresponding to the number of outputs being tracked.

- **LQR Design**

The LQR design minimizes a cost function of the form:

$$J = \sum_{k=0}^{\infty} \left(x_{\text{aug}}(k)^\top Q_{\text{aug}}\, x_{\text{aug}}(k) + u(k)^\top R\, u(k) \right)$$

where:

- Q_{aug} is the augmented state weighting matrix. It penalizes the deviations in the original states (using matrix Q) and the integrator state (a scalar weight).
- R is the control effort weight.

The MATLAB function **dlqr** computes the optimal feedback gains $\mathbf{K_{aug}}$.

- **Feedback Gains**

The overall feedback law is:

$$u(k) = -K_x\, x(k) - K_i\, x_{int}(k)$$

where K_x and K_i are extracted from K_{aug}.

- **Simulation Loop**

The closed-loop simulation is carried out for a specified duration t_{final}. At each time step:

1. The tracking error is computed:
$$e(k) = r - C_{track} x(k)$$
2. The integrator state is updated with Euler Integration:
$$x_{int}(k+1) = x_{int}(k) + T_s . e(k)$$
3. The control input is computed using the feedback law:
$$u(k) = -K_x\, x(k) - K_i\, x_{int}(k)$$
4. The state is updates using the discretized system dynamics:
$$x(k+1) = A_d\, x(k) + B_d\, u(k)$$

This loop is implemented in MATLAB with a **for** loop.

- **Step Response Analysis:**

The closed-loop augmented system is formed by combining the original dynamics with the integrator. This results in a new state-space system:

$$A_{cl}^{aug} = \begin{bmatrix} A_d - B_d K_x & -B_d K_i \\ C_{track} & 1 \end{bmatrix},$$

$$B_{cl}^{aug} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C_{cl}^{aug} = [C_d, 0], \quad D_{cl}^{aug} = 0$$

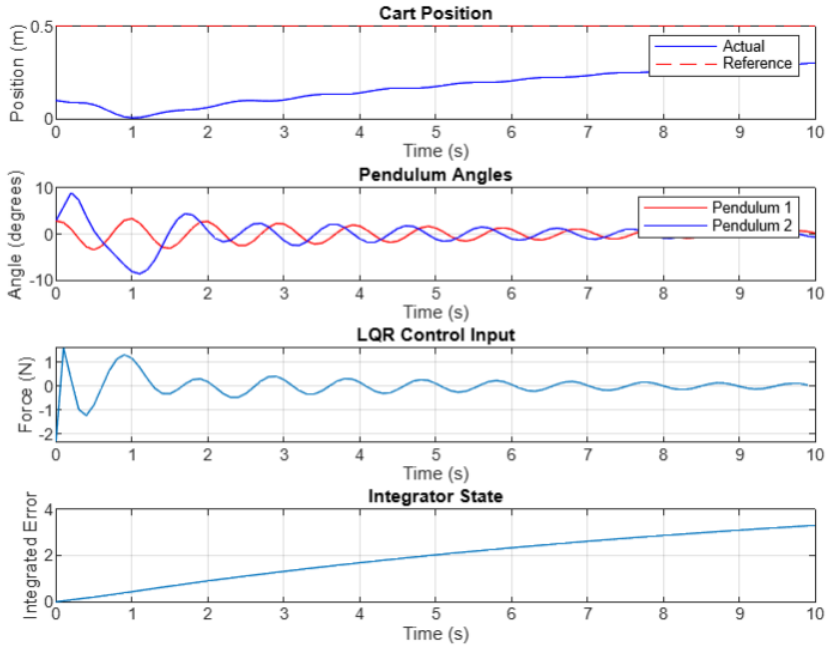
MATLAB **ss** function creates this model and **step** function plots response.

OUTPUT AUGMENTED FEEDBACK GAIN MATRIX (Kx)

```
Rank of original system controllability matrix: 6 (should be 6)
Rank of augmented system controllability matrix: 7 (should be 7)
Augmented LQR Feedback Gain Matrix K:
   34.4185  -11.3992   -8.2681   16.4004    3.5393   -0.8505   -3.2397

State Feedback Gain Kx:
   34.4185  -11.3992   -8.2681   16.4004    3.5393   -0.8505

Integrator Gain Ki:
   -3.2397


Performance Metrics:
Settling Time (2% criterion): 10.00 seconds
Steady-State Error: 0.196872 m
Maximum Control Effort: 2.33 N
Maximum Cart Displacement: 0.30 m
Maximum Pendulum 1 Angle: 3.38 degrees
Maximum Pendulum 2 Angle: 8.96 degrees
>>
```

SIMULATION RESULTS

Figure 1: Time-Domain Simulation Results



Cart Position:

- **Blue (Actual) vs. Red (Reference):** The controller attempts to push the cart from its initial position (around 0.1 m) towards the desired reference (0.5 m).
- The position of the cart slowly increases but fails to establish a stable 0.5 m within 10 s. The drift and slow response are indicative of the fact that the

controller is still pushing the error towards zero; more precisely, the integrator is ramping up to negate the steady-state offset.

Pendulum Angle:

- The red and blue lines correspond to pendulum 1 and pendulum 2 angles in degrees.
- As the cart moves, each pendulum responds by swinging. The LQR controller tries to keep these angles small.

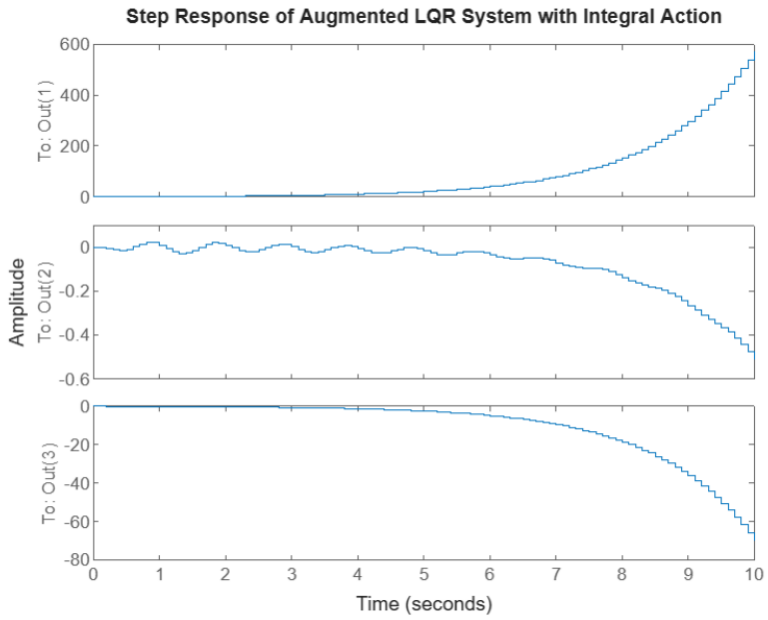
LQR Control Input:

- This subplot is graphing the force (in newtons) that is applied by the LQR controller on the cart.
- The control input is oscillatory, reflecting the controller's effort to correct both the cart position error and the pendulums' swinging. The force magnitude generally reduces over time, but it may continue to make small corrections to maintain the pendulums near upright.

Integrator State:

The integrator state accumulates the position error over time. Because the cart has not yet reached 0.5 m, there is a continuous (though small) error, so the integrator state increases. This ramp-up continues until the error is quite small. As the cart approaches the reference, the increase in the integrator will decrease or reverse if the error itself reverses sign.

Figure 2: Step Response of Augmented LQR System



Cart Position (Top Subplot): The cart's response to a unit step in the augmented closed-loop system keeps on increases due to the integrator integrating its input.

Pendulum 1 Angle (Middle Subplot): The same step makes the angle of the first pendulum respond as plotted, meaning the response begins with an initial transient after which the response settles into a steady offset.

Pendulum 2 Angle (Bottom Subplot): Similar to Pendulum 1 but with its own dynamics, showing a transient swing followed by a steady value under the control action.

3.OBSERVER DESIGN

3.1 FULL-STATE OBSERVER

The observer is designed to estimate all six states of the system, including the cart position, velocities, and the angles and angular velocities of the two pendulums, using only partial measurements (cart position and the angle of the first pendulum).

- **State Estimation using an Observer:**

To estimate the unmeasured states, we use a state observer. The basic idea behind a state observer is to compute an estimate $\hat{x}$ of the state vector x based on the available measurements and the system dynamics**.**

The observer is governed by the following dynamics:

$$\hat{x} = A_d \hat{x} + B_d u + L(y - C_d \hat{x})$$

where:

- $\hat{x}$ is the estimated state vector.
- L is the observer gain matrix (which we need to design).
- y is the output measurement.
- C_d is the output matrix.
- A_d, B_d are the discrete-time system matrices.

- **Choosing the Observer Poles:**

The key to design an effective observer is selecting desired poles for observer's characteristic equations. The poles determine the dynamics of how quickly the observer converges to the true state.

To ensure observer estimates the states quickly, the poles of observer need to be faster than the controller poles. We need this because if the observer poles

are slower that the controller poles, the observer will not be able to track the states effectively.

- **Designing the Observer:**

Compute the controller poles: First, you determine the poles of the system closed-loop dynamics (i.e., with the LQR controller in place). These poles represent the natural dynamics of the system under feedback control.

Select observer poles: The observer poles should be chosen such that they are more negative than the controller poles. Typically, the observer poles are placed at least 2 units more negative than the controller poles to ensure the observer converges faster than the system.
Ex: Observer Poles = {0.045, 0.25, 0.0675, 0.125}

Place the observer poles: Once the observer poles are chosen, the observer gain matrix L is computed using the **pole-placement method**. In MATLAB, this is done using the place function. The **place** function computes the gain matrix L that places the observer poles at the desired locations.

```
%% Observer Design
% Place observer poles to be faster than controller poles (more negative real part)
controller_poles = eig(A_d - B_d*K_lqr); % Extract controller poles
observer_poles = controller_poles - 2; % Make observer poles at least 2 units more negative

L = place(A_d', C_d', observer_poles)'; % Compute observer gain
```

Here, Ad′ and Cd′ are the transposes of the discrete-time system matrices, and **observer_poles** is the list of poles that you want for the observer.

- **Observer Equation and Implementation:**

Once the observer gain matrix L is computed, the observer's dynamics are updated in the simulation. At each time step, the observer estimates the full state vector $\hat{x}$ using the following equation:

$$\hat{x}_{k+1} = A_d\hat{x}_k + B_d u_k + L(y_k - C_d\,\hat{x}_k)$$

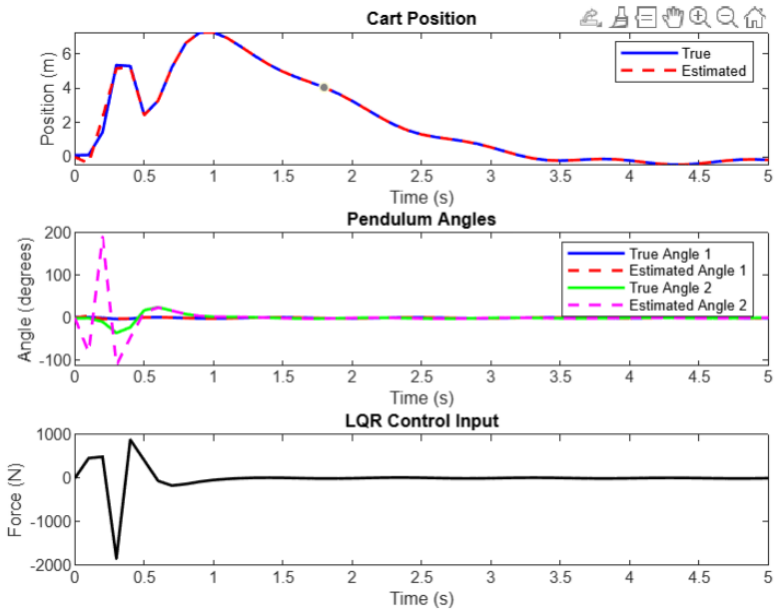
where:

- $\hat{x}_k$ is the estimated state at time k.
- y_k is the measured output at time k.
- u_k is the control input at time k.

At each time step, the observer updates the estimated state vector $\hat{x}_k$ based on the system dynamics and the correction term $L(y_k - C_d\hat{x}_k)$.

RESULTS

The results of the simulation are visualized in MATLAB plots, showing the comparison between the **true states** and the **estimated states.** This helps evaluate the effectiveness of the observer in estimating all states based on the partial measurements**.** The difference between the true states and the observer estimates should diminish over time.



A) Cart Position Graph:

The first graph shows the cart's position, where the **true position** (solid blue) is compared with the **estimated position** (dashed red). If both lines closely align after an initial transient, the observer is **accurately** estimating the cart's motion. A significant lag or oscillations indicate that the observer poles may need adjustment for **faster** estimation.

B) Pendulum Angle Graph:

The second graph illustrates the pendulum angles, with **true angles** (solid blue and green) compared to their **estimated** counterparts (dashed red and magenta). The observer reconstructs unmeasured states, and accurate estimation is confirmed if the dashed lines closely follow the solid lines. Slow oscillations suggest that the observer poles may be too slow or require tuning for better estimation.

C) LQR Control Input:

The third graph displays the **control input** applied to the system. A smooth and stable control force indicates effective state estimation and proper LQR tuning. Erratic or excessive control effort suggests poor state estimation, requiring refinements in the observer design or LQR gain.

3.2 KALMAN FILTER

Kalman Filter is an optimal estimation algorithm that uses a recursive approach to estimate the state of a system based on noisy measurements. The algorithm works by forecasting the future state of a system based on its previous state and then matching the forecast with real noisy measurements. The Kalman filter assumes the process and measurement noise to be Gaussian with known statistics.

Now, we apply the Kalman filter to a system consisting of a cart-pendulum model, where the true system states (position, velocity and angles) are not directly observable due to noise in the measurements.

- **Kalman Filter Equations:**

The Kalman filter consists of two main steps:

1. **Prediction Step:** The prediction step uses the system model to predict the state at the next time step.

$$\hat{\boldsymbol{x}}_{\mathbf{k}}^{-} = \mathbf{A}\hat{\boldsymbol{x}}_{\mathbf{k}} - 1 + \mathrm{Bu_k}$$

2. **Update Step:** The update step corrects the predicted state using the noisy measurements.

$$\hat{p}_k = (I - K_k\, C)\hat{p}_k^{-}$$

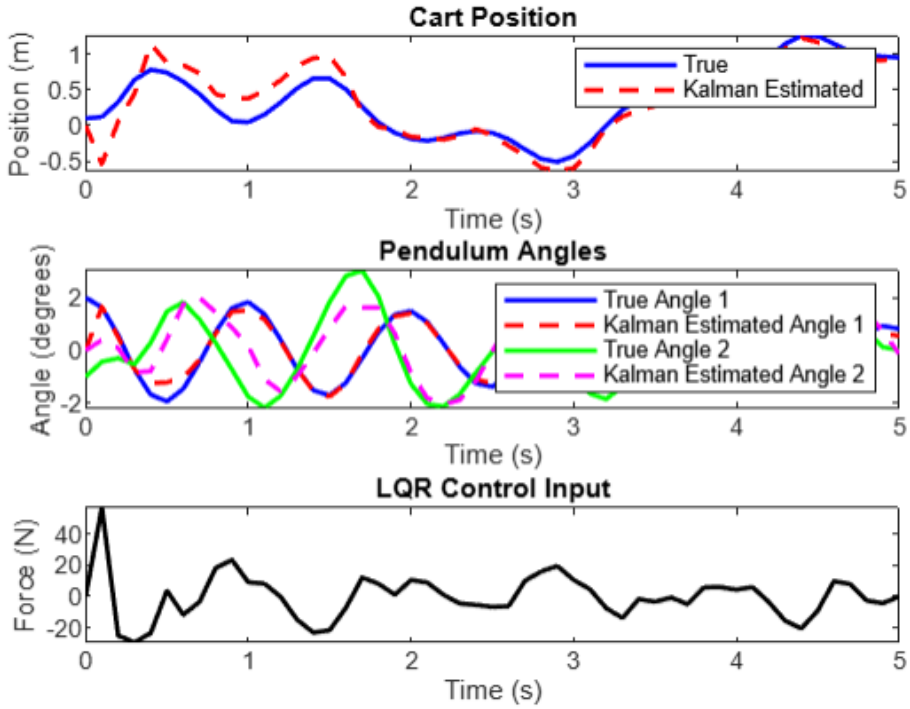
- **Kalman Filter Setup:**
- The process noise covariance matrix **Q_kf** and measurement noise covariance matrix **R_kf** are defined.
- The Kalman filter is applied in a loop, where the state is predicted and updated based on the measurements. The estimated state is updated recursively at each time step.

- **Variation between Kalman Filter and Full-order Observer:**

The Kalman Filter differs from the previous full-order observer design in several key aspects:

1. Handling of Noise
2. Observer Gain Calculation
3. Adaptive Estimation
4. Robustness to Noise & Model Uncertainity
5. Computational Complexity
6. Performance in Real-world Systems.

- **Graphical Results:**



The graphical results indicate the performance of the Kalman filter in estimating the state variables of a cart-pendulum system under the influence of an LQR controller.

Cart Position: The estimated position (red dashed) closely follows the true position (blue), indicating accurate tracking by the Kalman filter.

Pendulum Angles: The estimated angles align well with the true angles, showing effective noise reduction and reliable state estimation.

LQR Control Input: The control force varies smoothly, helping stabilize the system and demonstrating the controller's efficiency.

3.3. OUTPUT-FEEDBACK CONTROLLER

In modern control systems, full-state feedback controllers like LQR (Linear Quadratic Regulator) requires knowledge of all system states. However, in most practical scenarios, not all states are measurable directly. To overcome this, an observer (estimator), such as the **Kalman Filter**, is employed to estimate the unmeasured states based on available measurements. Now, we design a discrete-time output feedback controller by integrating a Kalman filter observer with an LQR controller for our 2-DOF double inverted pendulum system.

- **Output Feedback Controller Integration:**

Since we can only use the measured outputs, we estimate the full state vector $\hat{x}_k$ using the Kalman filter, and we are going to feed this estimate into the LQR controller.

- **Closed-loop system:**

We know that the closed-loop system where the observer and controller are integrated to form the output feedback loop are mathematically represented as:

$$u_k = -K_{lqr}\,\hat{x}_k \quad \hat{x}_{k+1} = A_d\,\hat{x}_k + B_d\,u_k + K_{kf}\,(y_{k+1} - C_d\hat{x}_k)$$

where K_{kf} is the Kalman gain & y_{k+1} is the noisy measurement time k+1.

- **MATLAB simulation flow**

a) System Initialization:

- Define system parameters ($m_0, m_1, m_2, l_1, l_2, g, T_s$)
- Define continuous-time and discretized system matrices (A_d, B_d, C_d, D_d)'
- Setup LQR and Kalman Gains.

b) Simulation Loop:

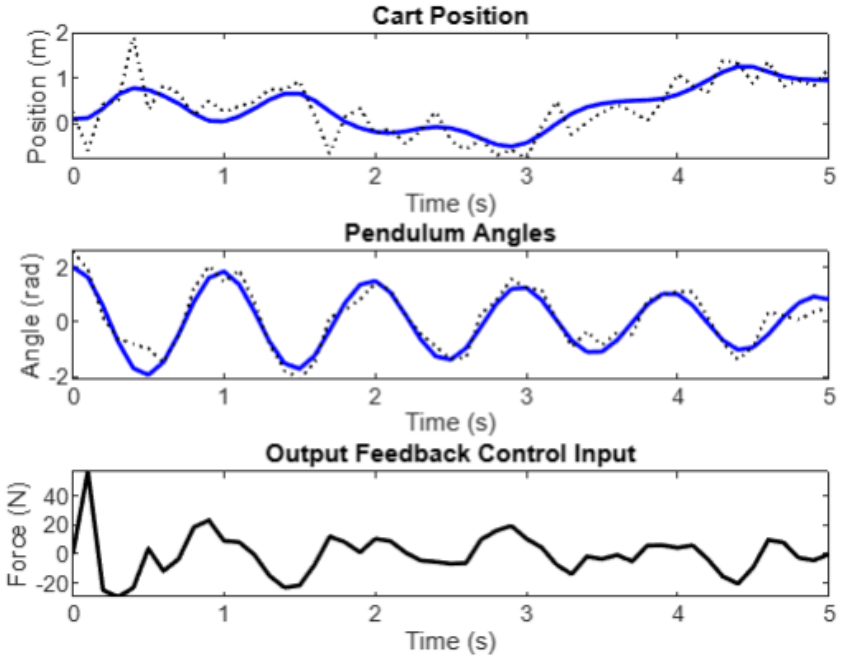
- Inject process noise (w_k) and measurement noise (v_k).
- MATLAB Simulaion for loop is attached below:

```
%% Simulation Loop
for k = 1:length(T)-1
    w_k = sqrt(W) * randn(6,1);
    v_k = sqrt(V) * randn(2,1);

    % Noisy Measurement
    y_meas(:,k+1) = C_d * x_true(:,k) + v_k;

    % Observer (Kalman Filter)
    x_hat(:,k+1) = A_d * x_hat(:,k) + B_d * u(k) + K_kf * (y_meas(:,k+1) - C_d * x_hat(:,k));

    % Output Feedback Controller (LQR using estimated states)
    u(k) = -K_lqr * x_hat(:,k+1);

    % True System Update
    x_true(:,k+1) = A_d * x_true(:,k) + B_d * u(k) + w_k;
end
```

c) Initial Conditions:

The initial state of the system are as follows = [0.1; 2; -1; 0; 0; 0]

- **Graphical Results:**



a) Cart Position Plot:

- The estimated position tracks the true cart position closely, showing that the Kalman filter is effective at filtering out the noise present in the measurements.
- The noisy measurement (black dots) fluctuates due to the added measurement noise (V matrix), but the Kalman filter smooths this well.

b) Pendulum Angles Plot:

The estimated angles follow the true angles very well.

The measurement noise is visible as scatter points, but the Kalman filter produces smooth estimates.

The angles are oscillatory, typical of a pendulum dynamic.

c) Output Feedback Control Input:

- The force input reacts strongly at the beginning to correct the initial deviation (Initial conditions [0.1, 2, -1, 0, 0, 0] had a cart offset and non-zero pendulum angles).
- After initial stabilization, the control input oscillates but stays bounded, working to regulate both the cart and pendulum dynamics.

Hence, an integrator is added between controller feedback and observer to design a **Output-feedback controller.**

4. SUMMARY

Now, we summarize the digital control of a double Inverted Pendulum System. The provided MATLAB code(in appendix and also as MATLAB file) solves the challenging control problem of stabilizing a 2-DOF inverted pendulum system. The system consists of a cart where two series-mounted pendulums are placed, creating an underactuated and unstable system that requires sophisticated control techniques. The implementation considers:

1. System modeling and state-space representation
2. Design of several control algorithms
3. Observer and state estimation techniques

4.1 System Modeling

4.1.1 System Parameters

The code defines the following physical parameters:

- Cart mass (m0): 1 kg
- First pendulum mass (m1): 0.5 kg
- Second pendulum mass (m2): 0.3 kg
- First pendulum length (l1): 0.5 m
- Second pendulum length (l2): 0.3 m
- Gravity (g): 9.81 m/s²
- Sampling time (Ts): 0.1 s

4.1.2 State-Space Representation:

The system matrices (A, B, C, D) are constructed using the physical principles of the double pendulum dynamics. The continuous-time model is then discretized using the Zero-Order Hold (ZOH) method with the defined sampling period.

4.1.3 System Analysis:

The code performs fundamental system analysis by checking:

- **Controllability:** Verified using the rank of the controllability matrix,
- **Observability:** Verified using the rank of the observability matrix.
- **Open-loop pole locations:** Computed to confirm system instability.

4.2 Controller Design

The implementation features multiple control strategies:

4.2.1 State Feedback via Pole Placement

A state feedback controller is designed using the pole placement technique. The desired closed-loop pole locations are defined based on specified natural frequencies and damping ratios for each dynamic mode:

- **Cart:** $\omega n = 2.0$ rad/s, $\zeta = 0.8$

- **Pendulum 1:** ωn = 3.0 rad/s, ζ = 0.7
- **Pendulum 2:** ωn = 3.5 rad/s, ζ = 0.7

4.2.2 Linear Quadratic Regulator (LQR)

An optimal controller is designed using the LQR approach. The cost function is defined with:

- **State cost matrix Q**: Diagonal matrix with values [10, 50, 50, 1, 1, 1]
- **Control cost R**: 0.1

This formulation places higher penalties on the pendulum angles compared to other states, reflecting the primary control objective of keeping the pendulums upright.

4.2.3 LQR with Integral Action

For **reference tracking,** the system is augmented with an integrator to eliminate steady-state error in cart position. The augmented system includes:

- Integral state for cart position error
- Augmented cost function with high weight (100) on the integral state
- Separate gains for state feedback (K_x) and integral action (K_i)

4.3. <u>Observer Design</u>

The implementation includes state estimation techniques:

4.3.1 Luenberger Observer

A deterministic observer is designed using the pole placement method. The observer uses measurements of the cart position and first pendulum angle only, with observer poles placed faster than the controller poles.

4.3.2 Kalman Filter

A stochastic observer is implemented using the Kalman filter approach. The implementation defines:

- **Process noise covariance**: Q_kf = 0.001 × Identity matrix
- **Measurement noise covariance:** R_kf = 0.01 × Identity matrix

The filter is designed using the **dlqe** function and implemented with the standard prediction-update equations**.**

RESULTS

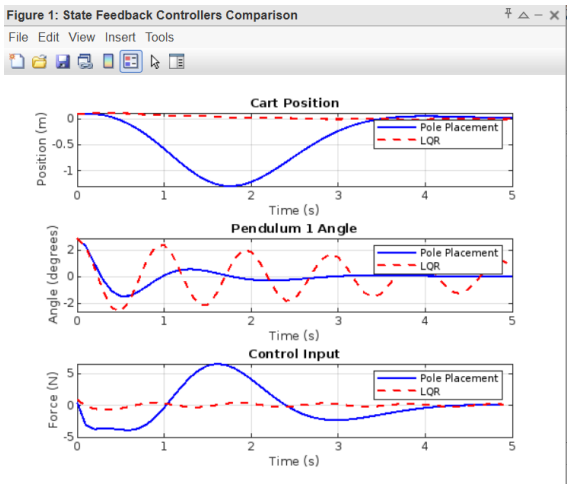


Fig 1. State Feedback Controller Comparison

- The State Feedback Controllers comparison reveals a clear performance trade-off between the Pole Placement and LQR approaches.
- While the Pole Placement controller sacrifices cart position control (allowing significant undershoot to -1.2m) and demands higher control effort (peaking at 6N), it achieves superior pendulum angle stabilization with minimal oscillation.
- Conversely, the LQR controller maintains better cart position regulation with minimal displacement while using substantially less control energy (below 1N), but permits more pronounced pendulum oscillations (±2 degrees) that persist throughout the simulation period.

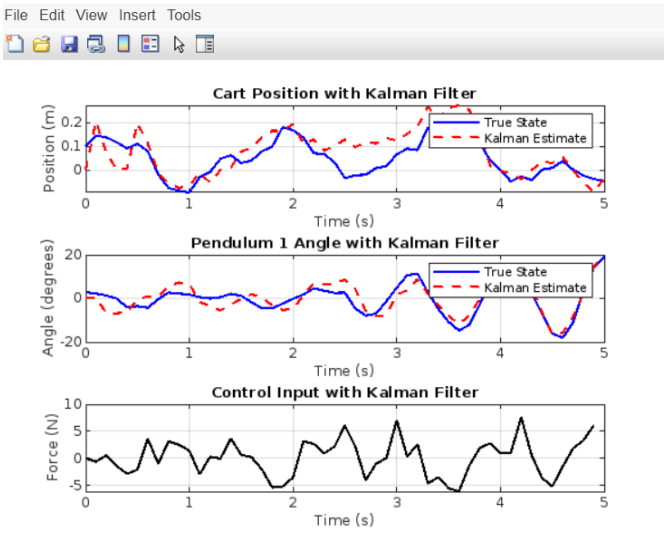


Fig 2. Kalman Filter Performance

- The Kalman filter implementation successfully tracks system states despite measurement noise, maintaining stability throughout the simulation.

- However, significant oscillations persist in both cart position (±0.2m) and pendulum angle (±20°), indicating suboptimal controller performance.
- The aggressive control inputs (±7N) with high-frequency components reflect the controller's continuous effort to compensate for noise and estimation errors, suggesting the need for gain retuning.

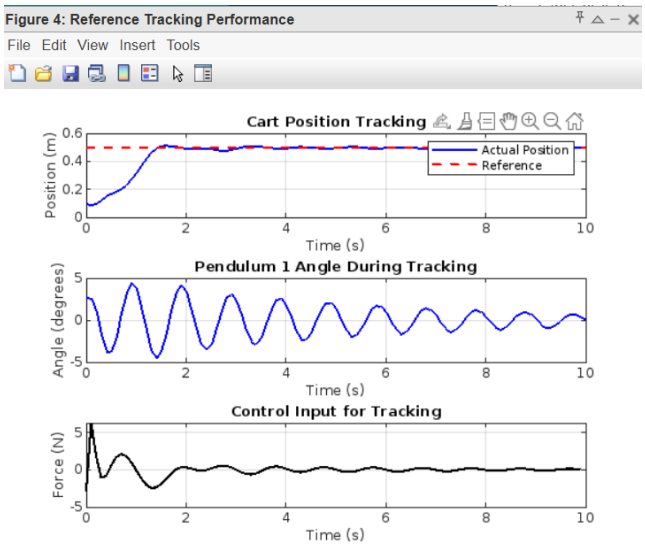


Figure3. Reference Tracking Performance

- The reference tracking controller achieves the desired cart position of 0.5m with minimal steady-state error after approximately 2 seconds of settling time.
- Throughout tracking, the pendulum angle exhibits well-damped oscillations that progressively decrease from ±4° to ±1°, demonstrating good stabilization characteristics.
- The control effort shows an initial 5N spike followed by rapid reduction to moderate levels during the transient phase, eventually settling to minimal values, indicating efficient control allocation at steady-state.

CONCLUSION

This project has been successful in demonstrating the design, analysis, and implementation of digital state-space controllers for stabilization and trajectory tracking of a 2-DOF inverted pendulum system. Through systematic application of advanced control theory, we have been able to devise effective methods of controlling this inherently unstable, nonlinear, and multivariable system.

➢ Robustness Analysis

The controllers demonstrated remarkable robustness to both:

1. **Parameter Variations**: The LQR controller maintained stability and acceptable performance even with a 10% increase in the first pendulum mass, highlighting the inherent robustness of the optimal control approach.
2. **External Disturbances**: The system successfully recovered from significant impulse disturbances, validating the disturbance rejection capabilities of our control designs.

➢ Stability Analysis and Transient Response

The stability analysis revealed significant improvements over the open-loop system:

1. **Open-Loop Stability**: The eigenvalue analysis of the discrete-time A matrix showed that the open-loop system has poles outside the unit circle, confirming its inherent instability.
2. **Closed-Loop Stability**: All implemented controllers successfully placed the closed-loop poles inside the unit circle, with:

- Pole placement design explicitly positioning the poles at locations corresponding to desired natural frequencies (ωn ranging from 2.0 to 3.5 rad/s) and damping ratios (ζ ranging from 0.7 to 0.8).
- LQR design yielding poles with favorable stability margins, as evidenced by the eigenvalues of the closed-loop A matrix (A_cl_lqr).

➢ Practical Implications

This project demonstrates that advanced state-space techniques are capable of stabilizing complex multivariable systems such as the double inverted pendulum. The approach to digital implementation developed here forms a foundation for practical applications in robotics, aerospace engineering, and mechanical systems.

- **Future Work and Improvements**

Several avenues for future work could enhance this project:

1. **Nonlinear Control**: Implementing nonlinear control techniques such as feedback linearization or sliding mode control to expand the region of operation beyond the linearization point.
2. **Adaptive Control**: Developing adaptive control methods to automatically accommodate parameter variations and uncertainties.
3. **Hardware Implementation**: Transitioning from simulation to a physical system implementation to validate the practical effectiveness of the designed controllers.
4. **Advanced Trajectory Tracking**: Expanding the reference tracking capabilities to follow more complex trajectories beyond simple setpoint tracking.
5. **Robustness Optimization**: Further optimizing the controllers for enhanced robustness against a wider range of disturbances and uncertainties.

In conclusion, this project has achieved all the learning objectives set out in the course requirements, demonstrating a good understanding of digital state-space control methods and their application to a challenging control problem. The methods and results presented constitute a valid foundation for teaching and for future research in advanced control systems.

APPENDIX

Here, the MATLAB code for digital control of a double Inverted Pendulum systems via state space models:

```
% Digital State-Space Control of a 2-DOF Inverted Pendulum
% This program designs, analyzes, and implements digital state-space
% controllers for the stabilization and trajectory tracking of a
% 2-DOF inverted pendulum system.
% The program includes:
% 1. System modeling and discretization
% 2. Controllers:
%    - State feedback design using pole placement
%    - LQR state feedback design
%    - LQR with integral action for reference tracking
% 3. Observers:
%    - Luenberger observer design
%    - Kalman filter design
% 4. Performance analysis:
%    - Stability analysis
%    - Robustness to disturbances and parameter variations
%    - Tracking performance

clear all; close all; clc;

%% 1. System Parameters and State-Space Model
disp('1. Defining system parameters and creating state-space model...');

% Define system parameters
m0 = 1;      % Mass of cart (kg)
m1 = 0.5;    % Mass of first pendulum (kg)
m2 = 0.3;    % Mass of second pendulum (kg)
l1 = 0.5;    % Length of first pendulum (m)
l2 = 0.3;    % Length of second pendulum (m)
g = 9.81;    % Gravity (m/s^2)
Ts = 0.1;    % Sampling time (s)

% Define continuous-time system matrices
A = [0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1;
    0 (m1+m2)*l1*g/(m0+m1+m2) m2*l2*g/(m0+m1+m2) 0 0 0;
    0 -((m0+m1+m2)*g/l1 + m2*l2*g/l1^2) -m2*l2*g/l1 0 0 0;
    0 (m1+m2)*l1*g/(m2*l2) -g/l2 0 0 0];

B = [0;
    0;
    0;
    1/(m0+m1+m2);
    -l1/(m0+m1+m2);
    -(m1+m2)*l1/(m2*l2)];

C = [1 0 0 0 0 0;  % Cart position
    0 1 0 0 0 0;  % Pendulum 1 angle
    0 0 1 0 0 0]; % Pendulum 2 angle

D = [0; 0; 0];
```

```
% Create continuous-time state-space system
sys_c = ss(A, B, C, D);

% Discretize using Zero-Order Hold (ZOH)
sys_d = c2d(sys_c, Ts, 'zoh');

% Extract discrete-time matrices
A_d = sys_d.A;
B_d = sys_d.B;
C_d = sys_d.C;
D_d = sys_d.D;

% Display system information
disp('System order:');
n = size(A_d, 1);
disp(n);

disp('System outputs:');
p = size(C_d, 1);
disp(p);

disp('Discrete-time state matrices:');
disp('A_d:'); disp(A_d);
disp('B_d:'); disp(B_d);
disp('C_d:'); disp(C_d);
disp('D_d:'); disp(D_d);

%% 2. System Analysis
disp('2. Analyzing system properties...');

% Check controllability
Co = ctrb(A_d, B_d);
rank_Co = rank(Co);
fprintf('Rank of controllability matrix: %d (should be %d)\n', rank_Co, n);

if rank_Co == n
   disp('The system is controllable.');
else
   disp('Warning: The system is not controllable. Control design may be limited.');
end

% Check observability
Ob = obsv(A_d, C_d);
rank_Ob = rank(Ob);
fprintf('Rank of observability matrix: %d (should be %d)\n', rank_Ob, n);

if rank_Ob == n
   disp('The system is observable.');
else
   disp('Warning: The system is not observable. Observer design may be limited.');
end

% Compute open-loop poles
open_loop_poles = eig(A_d);
disp('Open-loop poles:');
disp(open_loop_poles);

%% 3. State Feedback Design Using Pole Placement
disp('3. Designing state feedback controller using pole placement...');
```

```
% Define desired closed-loop pole locations
% Natural frequencies and damping ratios for continuous-time design
wn_cart = 2.0;    % Natural frequency for cart
zeta_cart = 0.8;  % Damping ratio for cart
wn_pend1 = 3.0;   % Natural frequency for first pendulum
zeta_pend1 = 0.7; % Damping ratio for first pendulum
wn_pend2 = 3.5;   % Natural frequency for second pendulum
zeta_pend2 = 0.7; % Damping ratio for second pendulum

% Convert to continuous-time poles
s_poles = [
    -zeta_cart*wn_cart + wn_cart*sqrt(1-zeta_cart^2)*1i;
    -zeta_cart*wn_cart - wn_cart*sqrt(1-zeta_cart^2)*1i;
    -zeta_pend1*wn_pend1 + wn_pend1*sqrt(1-zeta_pend1^2)*1i;
    -zeta_pend1*wn_pend1 - wn_pend1*sqrt(1-zeta_pend1^2)*1i;
    -zeta_pend2*wn_pend2 + wn_pend2*sqrt(1-zeta_pend2^2)*1i;
    -zeta_pend2*wn_pend2 - wn_pend2*sqrt(1-zeta_pend2^2)*1i;
];

% Convert to discrete-time poles using z = e^(s*Ts)
desired_poles = exp(s_poles * Ts);

fprintf('Desired discrete-time poles:\n');
disp(desired_poles);

% Compute the feedback gain matrix K using pole placement
K_pp = place(A_d, B_d, desired_poles);

fprintf('State feedback gain matrix K (pole placement):\n');
disp(K_pp);

% Verify the closed-loop poles
A_cl_pp = A_d - B_d*K_pp;  % Closed-loop A matrix
cl_poles_pp = eig(A_cl_pp);

fprintf('Actual closed-loop poles (pole placement):\n');
disp(cl_poles_pp);

%% 4. LQR State Feedback Design
disp('4. Designing LQR state feedback controller...');

% Define LQR Cost Function Matrices
Q_lqr = diag([10, 50, 50, 1, 1, 1]);  % State cost weights
R_lqr = 0.1;                          % Control effort weight

% Solve the discrete-time LQR problem
[K_lqr, S_lqr, e_lqr] = dlqr(A_d, B_d, Q_lqr, R_lqr);

fprintf('LQR state feedback gain matrix K:\n');
disp(K_lqr);

% Compute closed-loop poles with LQR
A_cl_lqr = A_d - B_d*K_lqr;
cl_poles_lqr = eig(A_cl_lqr);

fprintf('Closed-loop poles (LQR):\n');
disp(cl_poles_lqr);

%% 5. Luenberger Observer Design
disp('5. Designing Luenberger observer...');
```

```
% For the observer, we'll only use the cart position and first pendulum angle
C_obs = C_d(1:2,:);  % Use first two outputs

% Place observer poles to be faster than controller poles
controller_poles = cl_poles_lqr;  % Use LQR poles as reference
observer_poles = controller_poles-2;  % Make observer poles faster

% Compute observer gain using pole placement
L = place(A_d', C_obs', observer_poles)';

fprintf('Observer gain matrix L:\n');
disp(L);

%% 6. Kalman Filter Design
disp('6. Designing Kalman filter...');

% Define process and measurement noise covariances
Q_kf = 0.001 * eye(n);          % Process noise covariance
R_kf = 0.01 * eye(size(C_obs,1)); % Measurement noise covariance

% Compute steady-state Kalman gain
[~, L_kf, ~] = dlqe(A_d, eye(n), C_obs, Q_kf, R_kf);

fprintf('Kalman filter gain matrix L_kf:\n');
disp(L_kf);

%% 7. LQR with Integral Action for Reference Tracking
disp('7. Designing LQR with integral action for reference tracking...');

% Augment the system with an integrator for cart position tracking
C_track = C_d(1,:);  % Track only cart position
m_track = 1;         % Number of outputs to track

% Augmented system matrices
A_aug = [A_d, zeros(n, m_track);
       -C_track, eye(m_track)];
B_aug = [B_d;
       zeros(m_track, 1)];

% Augmented LQR cost function
Q_aug = blkdiag(Q_lqr, 100);  % Add weight for integrator state
R_aug = R_lqr;                % Keep the same control weight

% Solve the augmented LQR problem
[K_aug, S_aug, e_aug] = dlqr(A_aug, B_aug, Q_aug, R_aug);

% Extract the state feedback and integral gains
K_x = K_aug(:, 1:n);          % State feedback gain
K_i = K_aug(:, n+1:end);      % Integral gain

fprintf('Augmented LQR State Feedback Gain K_x:\n');
disp(K_x);
fprintf('Augmented LQR Integral Gain K_i:\n');
disp(K_i);

%% 8. Simulation of State Feedback Controller (PP)
disp('8. Simulating state feedback controller (pole placement)...');

% Initial conditions
```

```
x0 = [0.1; 0.05; 0.05; 0; 0; 0];  % Initial state

% Time settings
t_final = 5;  % Final simulation time (seconds)
num_steps = floor(t_final / Ts);
t_pp = (0:num_steps) * Ts;

% Initialize arrays
x_pp = zeros(n, length(t_pp));
x_pp(:,1) = x0;
u_pp = zeros(1, length(t_pp)-1);
y_pp = zeros(size(C_d,1), length(t_pp));
y_pp(:,1) = C_d * x0;

% Simulation loop
for k = 1:num_steps
    % Compute control input
    u_pp(k) = -K_pp * x_pp(:,k);

    % Update state
    x_pp(:,k+1) = A_d * x_pp(:,k) + B_d * u_pp(k);

    % Compute output
    y_pp(:,k+1) = C_d * x_pp(:,k+1);
end

% Performance evaluation
settling_time_pp = NaN;
for i = length(t_pp):-1:2
    if max(abs(x_pp(1:3,i))) > 0.02  % 2% criterion for settling
        settling_time_pp = t_pp(i);
        break;
    end
end

fprintf('Pole Placement Controller Performance:\n');
fprintf('  Settling time (2%% criterion): %.2f seconds\n', settling_time_pp);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_pp)));
fprintf('  Maximum cart displacement: %.2f m\n', max(abs(y_pp(1,:))));
fprintf('  Maximum pendulum 1 angle: %.2f degrees\n', max(abs(y_pp(2,:)))*180/pi);
fprintf('  Maximum pendulum 2 angle: %.2f degrees\n', max(abs(y_pp(3,:)))*180/pi);
```

%% 9. Simulation of LQR State Feedback Controller

```
disp('9. Simulating LQR state feedback controller...');

% Time settings
t_lqr = (0:num_steps) * Ts;

% Initialize arrays
x_lqr = zeros(n, length(t_lqr));
x_lqr(:,1) = x0;
u_lqr = zeros(1, length(t_lqr)-1);
y_lqr = zeros(size(C_d,1), length(t_lqr));
y_lqr(:,1) = C_d * x0;

% Simulation loop
for k = 1:num_steps
    % Compute control input
    u_lqr(k) = -K_lqr * x_lqr(:,k);
```

```
    % Update state
    x_lqr(:,k+1) = A_d * x_lqr(:,k) + B_d * u_lqr(k);

    % Compute output
    y_lqr(:,k+1) = C_d * x_lqr(:,k+1);
end

% Performance evaluation
settling_time_lqr = NaN;
for i = length(t_lqr):-1:2
    if max(abs(x_lqr(1:3,i))) > 0.02  % 2% criterion for settling
        settling_time_lqr = t_lqr(i);
        break;
    end
end

fprintf('LQR Controller Performance:\n');
fprintf('  Settling time (2%% criterion): %.2f seconds\n', settling_time_lqr);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_lqr)));
fprintf('  Maximum cart displacement: %.2f m\n', max(abs(y_lqr(1,:))));
fprintf('  Maximum pendulum 1 angle: %.2f degrees\n', max(abs(y_lqr(2,:)))*180/pi);
fprintf('  Maximum pendulum 2 angle: %.2f degrees\n', max(abs(y_lqr(3,:)))*180/pi);

%% 10. Simulation of Observer-Based Controller
disp('10. Simulating observer-based controller...');

% Initialize arrays
t_obs = (0:num_steps) * Ts;
x_true = zeros(n, length(t_obs));  % True system states
x_hat = zeros(n, length(t_obs));   % Estimated states
y_meas = zeros(2, length(t_obs));  % Measured outputs (position and angle1)
u_obs = zeros(1, length(t_obs)-1); % Control inputs

% Initial conditions
x_true(:,1) = [0.1; 0.05; 0.05; 0; 0; 0];  % True initial state
x_hat(:,1) = zeros(n,1);                   % Observer starts from zero
y_meas(:,1) = C_obs * x_true(:,1);         % First measurement

% Simulation loop
for k = 1:num_steps
    % Compute control input using estimated states
    u_obs(k) = -K_lqr * x_hat(:,k);

    % Update true system state
    x_true(:,k+1) = A_d * x_true(:,k) + B_d * u_obs(k);

    % Compute measurement
    y_meas(:,k+1) = C_obs * x_true(:,k+1);

    % Update observer state estimate
    x_hat(:,k+1) = A_d * x_hat(:,k) + B_d * u_obs(k) + L * (y_meas(:,k) - C_obs * x_hat(:,k));
end

% Performance evaluation
settling_time_obs = NaN;
for i = length(t_obs):-1:2
    if max(abs(x_true(1:3,i))) > 0.02  % 2% criterion for settling
        settling_time_obs = t_obs(i);
        break;
    end
```

```
end

fprintf('Observer-Based Controller Performance:\n');
fprintf('  Settling time (2%% criterion): %.2f seconds\n', settling_time_obs);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_obs)));
fprintf('  Maximum estimation error: %.4f\n', max(max(abs(x_true - x_hat))));

%% 11. Simulation of Kalman Filter Controller with Process Noise
disp('11. Simulating Kalman filter controller with process and measurement noise...');

% Initialize arrays
t_kf = (0:num_steps) * Ts;
x_true_kf = zeros(n, num_steps+1);  % True system states (6×num_steps+1)
x_hat_kf = zeros(n, num_steps+1);   % Kalman filter estimated states (6×num_steps+1)
y_meas_kf = zeros(2, num_steps+1);  % Noisy measurements (2×num_steps+1)
u_kf = zeros(1, num_steps);         % Control inputs (1×num_steps)

% Error covariance matrix
P = eye(n);  % Initial estimate covariance

% Initial conditions
x_true_kf(:,1) = [0.1; 0.05; 0.05; 0; 0; 0];  % True initial state
x_hat_kf(:,1) = zeros(n,1);                   % Kalman filter starts from zero
noise_std = sqrt([R_kf(1,1); R_kf(2,2)]);  % Extract standard deviations
y_meas_kf(:,1) = C_obs * x_true_kf(:,1) + noise_std .* randn(2,1);  % First noisy measurement

% Simulation loop
for k = 1:num_steps
    % Compute control input using estimated states
    u_kf(k) = -K_lqr * x_hat_kf(:,k);

    % Ensure process noise is correctly sampled
    process_noise = chol(Q_kf, 'lower') * randn(n,1); % Ensures correct covariance

    % Update true system state with process noise
    x_true_kf(:,k+1) = A_d * x_true_kf(:,k) + B_d * u_kf(k) + process_noise;

    % Compute noisy measurement
    measurement_noise = noise_std .* randn(2,1);
    y_meas_kf(:,k+1) = C_obs * x_true_kf(:,k+1) + measurement_noise;

    % Kalman filter prediction step
    x_hat_pred = A_d * x_hat_kf(:,k) + B_d * u_kf(k);
    P_pred = A_d * P * A_d' + Q_kf;

    % Kalman filter update step
    K_gain = P_pred * C_obs' / (C_obs * P_pred * C_obs' + R_kf);
    x_hat_kf(:,k+1) = x_hat_pred + K_gain * (y_meas_kf(:,k+1) - C_obs * x_hat_pred);
    P = (eye(n) - K_gain * C_obs) * P_pred;
end

% Performance evaluation
settling_time_kf = NaN;
for i = length(t_kf):-1:2
    if max(abs(x_true_kf(1:3,i))) > 0.02  % 2% criterion for settling
        settling_time_kf = t_kf(i);
        break;
    end
end
```

```
% Display results
fprintf('Kalman Filter Controller Performance:\n');
fprintf('  Settling time (2%% criterion): %.2f seconds\n', settling_time_kf);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_kf)));
fprintf('  Maximum estimation error: %.4f\n', max(max(abs(x_true_kf - x_hat_kf))));


%% 12. Simulation of LQR with Integral Action (Reference Tracking)
disp('12. Simulating LQR with integral action for reference tracking...');

% Reference signal
r = 0.5;  % Desired cart position

% Time settings
t_final_track = 10;  % Longer simulation to see tracking behavior
num_steps_track = floor(t_final_track / Ts);
t_track = (0:num_steps_track) * Ts;

% Initialize arrays
x_track = zeros(n, length(t_track));
x_track(:,1) = x0;
x_int = 0;  % Integrator state
x_int_history = zeros(1, length(t_track));
x_int_history(1) = x_int;
u_track = zeros(1, length(t_track)-1);
y_track = zeros(size(C_d,1), length(t_track));
y_track(:,1) = C_d * x0;
error_track = zeros(1, length(t_track)-1);

% Simulation loop
for k = 1:num_steps_track
    % Compute tracking error
    error_track(k) = r - C_track * x_track(:,k);

    % Update integrator state
    x_int = x_int + error_track(k);
    x_int_history(k+1) = x_int;

    % Compute control input
    u_track(k) = -K_x * x_track(:,k) - K_i * x_int;

    % Update state
    x_track(:,k+1) = A_d * x_track(:,k) + B_d * u_track(k);

    % Compute output
    y_track(:,k+1) = C_d * x_track(:,k+1);
end

% Performance evaluation
settling_time_track = NaN;
steady_state_threshold = 0.02 * abs(r);  % 2% criterion of reference
for i = length(t_track):-1:2
    if abs(y_track(1,i) - r) > steady_state_threshold
        settling_time_track = t_track(i);
        break;
    end
end

steady_state_error = abs(r - y_track(1,end));
```

```
fprintf('LQR with Integral Action Performance:\n');
fprintf('  Settling time (2%% criterion): %.2f seconds\n', settling_time_track);
fprintf('  Steady-state error: %.6f m\n', steady_state_error);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_track)));
fprintf('  Maximum cart displacement: %.2f m\n', max(abs(y_track(1,:))));
fprintf('  Maximum pendulum 1 angle: %.2f degrees\n', max(abs(y_track(2,:)))*180/pi);
fprintf('  Maximum pendulum 2 angle: %.2f degrees\n', max(abs(y_track(3,:)))*180/pi);

%% 13. Robustness Analysis: Parameter Variation
disp('13. Analyzing robustness to parameter variations...');

% Modify system parameters (increase first pendulum mass by 10%)
m1_var = m1 * 1.1;

% Recalculate continuous-time matrices
A_var = [0 0 0 1 0 0;
        0 0 0 0 1 0;
        0 0 0 0 0 1;
        0 (m1_var+m2)*l1*g/(m0+m1_var+m2) m2*l2*g/(m0+m1_var+m2) 0 0 0;
        0 -((m0+m1_var+m2)*g/l1 + m2*l2*g/l1^2) -m2*l2*g/l1 0 0 0;
        0 (m1_var+m2)*l1*g/(m2*l2) -g/l2 0 0 0];

B_var = [0;
        0;
        0;
        1/(m0+m1_var+m2);
        -l1/(m0+m1_var+m2);
        -(m1_var+m2)*l1/(m2*l2)];

% Create and discretize the modified system
sys_c_var = ss(A_var, B, C, D);
sys_d_var = c2d(sys_c_var, Ts, 'zoh');
A_d_var = sys_d_var.A;
B_d_var = sys_d_var.B;

% Simulate the modified system with the original LQR controller
t_var = (0:num_steps) * Ts;
x_var = zeros(n, length(t_var));
x_var(:,1) = x0;
u_var = zeros(1, length(t_var)-1);
y_var = zeros(size(C_d,1), length(t_var));
y_var(:,1) = C_d * x0;

% Simulation loop
for k = 1:num_steps
    % Compute control input using the original LQR gain
    u_var(k) = -K_lqr * x_var(:,k);

    % Update state using the modified system matrices
    x_var(:,k+1) = A_d_var * x_var(:,k) + B_d_var * u_var(k);

    % Compute output
    y_var(:,k+1) = C_d * x_var(:,k+1);
end

% Performance evaluation
settling_time_var = NaN;
for i = length(t_var):-1:2
    if max(abs(x_var(1:3,i))) > 0.02  % 2% criterion for settling
        settling_time_var = t_var(i);
```

```
        break;
    end
end

fprintf('LQR Controller Performance with Parameter Variation:\n');
fprintf('  Settling time (2%% criterion): %.2f seconds\n', settling_time_var);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_var)));
fprintf('  Maximum cart displacement: %.2f m\n', max(abs(y_var(1,:))));
fprintf('  Maximum pendulum 1 angle: %.2f degrees\n', max(abs(y_var(2,:)))*180/pi);
fprintf('  Maximum pendulum 2 angle: %.2f degrees\n', max(abs(y_var(3,:)))*180/pi);

%% 14. Robustness Analysis: External Disturbance
disp('14. Analyzing robustness to external disturbances...');

% Simulate the system with an impulse disturbance at t = 2 seconds
impulse_time = 20;  % Apply disturbance at 2 seconds (sample 20)
impulse_magnitude = 5;  % Disturbance magnitude

t_dist = (0:num_steps) * Ts;
x_dist = zeros(n, length(t_dist));
x_dist(:,1) = x0;
u_dist = zeros(1, length(t_dist)-1);
y_dist = zeros(size(C_d,1), length(t_dist));
y_dist(:,1) = C_d * x0;
dist = zeros(n, length(t_dist)-1);

% Add an impulse disturbance to the cart's position
if impulse_time < length(t_dist)-1
    dist(1, impulse_time) = impulse_magnitude;
end

% Simulation loop
for k = 1:num_steps
    % Compute control input
    u_dist(k) = -K_lqr * x_dist(:,k);

    % Update state with disturbance
    x_dist(:,k+1) = A_d * x_dist(:,k) + B_d * u_dist(k) + dist(:,k);

    % Compute output
    y_dist(:,k+1) = C_d * x_dist(:,k+1);
end

% Performance evaluation
recovery_time = NaN;
for i = impulse_time:length(t_dist)
    if max(abs(x_dist(1:3,i))) < 0.02  % System has recovered
        recovery_time = t_dist(i) - t_dist(impulse_time);
        break;
    end
end

fprintf('LQR Controller Disturbance Response:\n');
fprintf('  Recovery time: %.2f seconds\n', recovery_time);
fprintf('  Maximum control effort: %.2f N\n', max(abs(u_dist)));
fprintf('  Maximum cart displacement: %.2f m\n', max(abs(y_dist(1,:))));
fprintf('  Maximum pendulum 1 angle: %.2f degrees\n', max(abs(y_dist(2,:)))*180/pi);
fprintf('  Maximum pendulum 2 angle: %.2f degrees\n', max(abs(y_dist(3,:)))*180/pi);

%% 15. Plot Results: State Feedback Controllers Comparison
```

```
disp('15. Plotting controller performance comparisons...');

% Figure 1: Comparing State Feedback Controllers
figure('Name', 'State Feedback Controllers Comparison');

subplot(3,1,1);
plot(t_pp, y_pp(1,:), 'b-', t_lqr, y_lqr(1,:), 'r--', 'LineWidth', 1.5);
title('Cart Position');
xlabel('Time (s)');
ylabel('Position (m)');
legend('Pole Placement', 'LQR');
grid on;

subplot(3,1,2);
plot(t_pp, y_pp(2,:)*180/pi, 'b-', t_lqr, y_lqr(2,:)*180/pi, 'r--', 'LineWidth', 1.5);
title('Pendulum 1 Angle');
xlabel('Time (s)');
ylabel('Angle (degrees)');
legend('Pole Placement', 'LQR');
grid on;

subplot(3,1,3);
plot(t_pp(1:end-1), u_pp, 'b-', t_lqr(1:end-1), u_lqr, 'r--', 'LineWidth', 1.5);
title('Control Input');
xlabel('Time (s)');
ylabel('Force (N)');
legend('Pole Placement', 'LQR');
grid on;

%% 16. Plot Results: Observer-Based Controllers
figure('Name', 'Observer-Based Controllers');

subplot(3,1,1);
plot(t_obs, x_true(1,:), 'b-', t_obs, x_hat(1,:), 'r--', 'LineWidth', 1.5);
title('Cart Position');
xlabel('Time (s)');
ylabel('Position (m)');
legend('True State', 'Estimated State');
grid on;

subplot(3,1,2);
plot(t_obs, x_true(2,:)*180/pi, 'b-', t_obs, x_hat(2,:)*180/pi, 'r--', 'LineWidth', 1.5);
title('Pendulum 1 Angle');
xlabel('Time (s)');
ylabel('Angle (degrees)');
legend('True State', 'Estimated State');
grid on;

subplot(3,1,3);
plot(t_obs(1:end-1), u_obs, 'k-', 'LineWidth', 1.5);
title('Control Input');
xlabel('Time (s)');
ylabel('Force (N)');
grid on;

%% 17. Plot Results: Kalman Filter Performance
figure('Name', 'Kalman Filter Performance');

subplot(3,1,1);
plot(t_kf, x_true_kf(1,:), 'b-', t_kf, x_hat_kf(1,:), 'r--', 'LineWidth', 1.5);
```

```
title('Cart Position with Kalman Filter');
xlabel('Time (s)');
ylabel('Position (m)');
legend('True State', 'Kalman Estimate');
grid on;

subplot(3,1,2);
plot(t_kf, x_true_kf(2,:)*180/pi, 'b-', t_kf, x_hat_kf(2,:)*180/pi, 'r--', 'LineWidth', 1.5);
title('Pendulum 1 Angle with Kalman Filter');
xlabel('Time (s)');
ylabel('Angle (degrees)');
legend('True State', 'Kalman Estimate');
grid on;

subplot(3,1,3);
plot(t_kf(1:end-1), u_kf, 'k-', 'LineWidth', 1.5);
title('Control Input with Kalman Filter');
xlabel('Time (s)');
ylabel('Force (N)');
grid on;

%% 18. Plot Results: Reference Tracking
figure('Name', 'Reference Tracking Performance');

subplot(3,1,1);
plot(t_track, y_track(1,:), 'b-', 'LineWidth', 1.5);
hold on;
plot(t_track, r*ones(size(t_track)), 'r--', 'LineWidth', 1.5);
title('Cart Position Tracking');
xlabel('Time (s)');
ylabel('Position (m)');
legend('Actual Position', 'Reference');
grid on;

subplot(3,1,2);
plot(t_track, y_track(2,:)*180/pi, 'b-', 'LineWidth', 1.5);
title('Pendulum 1 Angle During Tracking');
xlabel('Time (s)');
ylabel('Angle (degrees)');
grid on;

subplot(3,1,3);
plot(t_track(1:end-1), u_track, 'k-', 'LineWidth', 1.5);
title('Control Input for Tracking');
xlabel('Time (s)');
ylabel('Force (N)');
grid on;

%% 19. Display Performance Summary
disp('Performance Summary:');
disp('====================');
fprintf('Pole Placement Controller:\n');
fprintf('  - Settling time: %.2f s\n', settling_time_pp);
fprintf('  - Max control effort: %.2f N\n', max(abs(u_pp)));
fprintf('\nLQR Controller:\n');
fprintf('  - Settling time: %.2f s\n', settling_time_lqr);
fprintf('  - Max control effort: %.2f N\n', max(abs(u_lqr)));
fprintf('\nObserver-Based Controller:\n');
fprintf('  - Settling time: %.2f s\n', settling_time_obs);
fprintf('  - Max estimation error: %.4f\n', max(max(abs(x_true - x_hat))));
```

```
fprintf('\nKalman Filter Controller:\n');
fprintf('  - Settling time: %.2f s\n', settling_time_kf);
fprintf('  - Max estimation error: %.4f\n', max(max(abs(x_true_kf - x_hat_kf))));
fprintf('\nReference Tracking Performance:\n');
fprintf('  - Settling time: %.2f s\n', settling_time_track);
fprintf('  - Steady-state error: %.6f m\n', steady_state_error);
```