ACS6116 Assignment
Design of Model Predictive Control (MPC) for Wie–Bernstein two-cart control benchmark system

Muzhaffar Maruf Ibrahim[1]

***Abstract*— Regulating a two-mass spring system with uncertain parameters is essential for understanding analogous systems in real-world applications, such as coupled-mass systems, power grids, and spacecraft docking, which require precise control of interconnected components. This study introduces a refined Linear Quadratic Model Predictive Controller (LQ-MPC) designed to stabilize the applied force u_k while accommodating demand response within specified state and input constraints. The controller achieved a stable system response in fewer than 10 steps, maintaining the force within the range $-1 \leq u_k \leq 1$, while effectively limiting the demand response, $|x_3| \leq 0.5, \quad |x_4| \leq 0.5$. The effects of variations in the demand response time constant were analyzed and mitigated through adjustments to the terminal cost matrix.**

I. Introduction

A benchmark problem is formulated to represent key characteristics of real-world systems, which are often multivariable in nature and subject to structured uncertainties [1]. These problems typically incorporate various constraints—on both inputs and outputs—and include **clearly defined control objectives, such as reference tracking and disturbance rejection**. In addition, performance criteria are established to quantitatively evaluate the effectiveness of the controller under these conditions [2].

It is important to recognize that predictive control represents a general framework rather than a fixed algorithm. The tuning processes required to ensure robust performance in ill-conditioned MIMO systems are inherently complex [5].

In academic research, Model Predictive Control (MPC) has been demonstrated under controlled experimental conditions on systems such as a simple mixing tank and a heat exchanger [6], as well as on more complex processes like a coupled distillation column used for ternary mixture separation [7], [8].

This study presents the design, tuning, and validation of a Linear-Quadratic Model Predictive Controller (LQ-MPC). The analysis includes a formal assessment of closed-loop stability, formulation of constraint-handling through a quadratic programming (QP) framework, and evaluation of controller performance. Additionally, the investigation considers the impact of varying the time constant associated with the demand response, as well as the effectiveness and robustness of the disturbance rejection strategy under different levels of perturbation.

*This work was not supported by any organization

[1]Muzhaffar Maruf Ibrahim is a Master Student in School of Electrical and Electronic Engineering, University of Sheffield, Sheffield S10 2TN, United Kingdom mmibrahim2@sheffield.ac.uk

II. Problem Statement

A. Two-mass-spring with uncertain parameters system

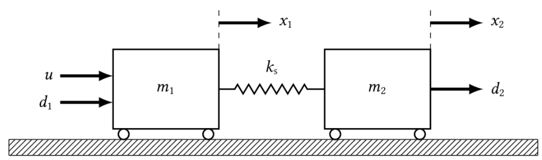

Fig. 1. The Wie–Bernstein two-cart control benchmark system

The Wie–Bernstein two-cart system is a well-known benchmark problem in the field of control design, particularly for evaluating robust and model predictive control methods which shown in Figure 1. This benchmark system consists of two carts, each with mass m_1 and m_2, connected by a linear spring of stiffness k_s, and placed on a frictionless surface. The first cart is directly actuated by a control input u, while the second cart is not actuated and instead serves as the system's performance output.

B. State-space Modelling and Constraints

The continuous-time dynamics of the system are described by a linear time-invariant state-space model:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = A^c \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + B^c u + E^c \begin{bmatrix} d_1 \\ d_2 \end{bmatrix},$$

where the state vector $x = [x_1\ x_2\ x_3\ x_4]^T$ contains the displacements of cart 1 and cart 2 (x_1, x_2) of the two carts and their respective velocities of cart 1 and cart 2 (x_3, x_4). The matrices A^c, B^c, and E^c are defined as:

$$A^c = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -k_s/m_1 & k_s/m_1 & 0 & 0 \\ k_s/m_2 & -k_s/m_2 & 0 & 0 \end{bmatrix}, \quad B^c = \begin{bmatrix} 0 \\ 0 \\ 1/m_1 \\ 0 \end{bmatrix}$$

$$E^c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m_1 & 0 \\ 0 & 1/m_2 \end{bmatrix}.$$

The disturbances d_1 and d_2 represent unknown external forces acting on carts 1 and 2, respectively. The performance output is defined as $y = x_2$, that is, the position of the second cart.

In addition to the above, the system is subject to physical and safety constraints, which include bounds on the control input and cart velocities:

$$|u| \leq 1, \quad |x_3| \leq 0.5, \quad |x_4| \leq 0.5. \tag{1}$$

These constraints reflect interconnected relationships of the actuator and the operating limits for the velocity of both carts and must be respected in any feasible control design.

III. Design

A. Controller design

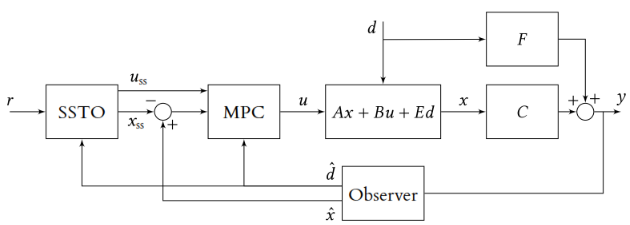

Fig. 2. Block diagram of a feedforward approach to tracking MPC

The Steady-State Target Optimization (SSTO) strategy requires computing the steady-state state x_{ss} and input u_{ss} that achieve the desired reference r in the presence of constant disturbance d, as shown in Figure 2. The discrete-time system is given by:

$$x(k+1) = Ax(k) + Bu(k) + Ed(k), \tag{2}$$

$$y(k) = Cx(k) + Fd(k), \tag{3}$$

with $d(k) = d$ and $r(k) = r$. Steady-state conditions are:

$$x_{ss} = Ax_{ss} + Bu_{ss} + Ed, \tag{4}$$

$$r = Cx_{ss} + Fd. \tag{5}$$

These yield the linear system:

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_{ss} \\ u_{ss} \end{bmatrix} = \begin{bmatrix} Ed \\ r - Fd \end{bmatrix}. \tag{6}$$

This ensures offset-free reference tracking in the presence of constant disturbances.

Then, the control law defined as:

$$u(k) = K_N x(k), \quad k = 0, 1, 2, \ldots$$

is stabilizing, where K_N is the feedback gain computed from the Riccati Difference Equation (RDE) as follows:

$$\begin{aligned} K_i &= -\left(R + B^\top \Pi_{i-1} B\right)^{-1} B^\top \Pi_{i-1} A \\ \Pi_i &= Q + A^\top \Pi_{i-1} A \\ &\quad - A^\top \Pi_{i-1} B \left(R + B^\top \Pi_{i-1} B\right)^{-1} B^\top \Pi_{i-1} A \end{aligned}$$

for $i = 1, \ldots, N$, with the initial condition:

$$\Pi_0 = P.$$

Under the above assumptions, the closed-loop system matrix $A + BK_N$ is stable for any prediction horizon $N \geq 0$. The weighting matrices Q, R, and P are selected to minimize the following finite-horizon quadratic cost function:

$$\sum_{k=0}^{N-1} \left[x^\top(k) Q x(k) + u^\top(k) R u(k)\right] + x^\top(N) P x(N) \tag{7}$$

Based on the system verification, it was confirmed that the matrices **A**, **B**, and the cost matrix **Q** satisfy the necessary controllability and observability conditions, as verified using the `check_ABQR(A,B,Q,R)` function.

B. Constraints formulation

To incorporate both input and state constraints into the MPC problem, we express them in stacked linear inequality form. Given:

$$P_u = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad q_u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{8}$$

$$P_x = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad q_x = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}, \tag{9}$$

To ensure that the input and state constraints are satisfied over the prediction horizon, we combine them as follows:

$$P_u u_N(k) \leq q_u \quad \text{(Input constraints)} \tag{1}$$

$$P_x x_N(k) \leq q_x \quad \text{(State constraints)} \tag{2}$$

where:

$$P_c = \begin{bmatrix} P_u \\ P_x G \end{bmatrix}, \quad q_c = \begin{bmatrix} q_u \\ q_x \end{bmatrix}, \quad S_c = \begin{bmatrix} 0 \\ -P_x F \end{bmatrix}$$

This formulation allows the MPC optimization problem to incorporate all constraints efficiently and solve for optimal control inputs $u_N(k)$ while accounting for current state $x(k)$.

IV. Results

A. Linear Quadratic Model Predictive Controller (LQ/MPC)

Based on the result of measurement analysis of Table IV, The **Unconstrained** controller achieves the *lowest cost* ($J = 9.3189$) but requires *substantially high control effort* ($\sum u_k = 132.95$) and exhibits a *very large overshoot* of approximately 71.17%. Although the settling time remains short ($T_s = 5$), the excessive overshoot reflects an aggressive control response due to the absence of constraints on inputs or states. In contrast, the **Input Constrained** controller demonstrates a *moderate cost* ($J = 20.946$) and significantly *lower control effort* ($\sum u_k = 47.617$), while the overshoot is markedly reduced to 1.54%. The **Input + State Constrained** controller offers the most balanced result, achieving the *lowest overshoot* (0.7635%) and further reduced control effort ($\sum u_k = 46.631$), with a *slightly higher cost* ($J = 22.347$) compared to the input-only constrained case.

No.	Description
1.	Define $x(k+1) = Ax(k) + B(u(k)+d)$, $y(k) = Cx(k)$
2.	Add state and input constraints: $P_x x \leq q_x$, $P_u u \leq q_u$
3.	Choose cost matrices Q, R, and terminal cost P
4.	Use $z = x - x_s$, $v = u - u_s$ to handle disturbances
5.	Reformulate as $z(k+1) = Az(k) + Bv(k)$
6.	Build cost and constraint matrices for optimization
7.	Compute F, G with `predict_mats(A, B, N)`
8.	Compute H, L, M using `cost_mats(F, G, Q, R)`
9.	Solve for x_{ss}, u_{ss} using steady-state equation
10.	Convert absolute constraints to deviation form: $q_x - P_x x_{ss}$, $q_u - P_u u_{ss}$
11.	Build with `constraint_mats(F, G, q)`
12.	Set initial deviation state $z(0) = x(0) - x_{ss}$
13.	Solve control input $v(k)$ with `quadprog` at each step

TABLE I
SIMPLIFIED MPC DESIGN PROCESS

TABLE II
PERFORMANCE METRICS OF LQ-MPC

Scenario	Cost J	u_k	Ts	Overshoot
Unconstrained	9.3189	132.95	5	71.172
Input Constrained	20.946	47.617	5	1.5423
Input + State Constrained	22.347	46.631	5	0.7635

B. LQ-MPC with Disturbance Rejection & Offset-Tracking

The LQ MPC controller minimises the quadratic cost function that balances state tracking and control effort. It achieves **fast convergence** ($T_s = 0.1$ s), **efficient control** ($u_k = 7.3865$), **and acceptable overshoot (9. 42%), demonstrating effective offset tracking under constraints shown in Table III**. These results confirm that the LQ MPC controller effectively handles system constraints and disturbances while maintaining robust performance in tracking the desired reference signal.

TABLE III
PERFORMANCE METRICS FOR OFFSET TRACKING

Scenario	Cost J	u_k	Ts	Overshoot
Offset-Tracking	7.3865	7.3865	0.1	9.4244

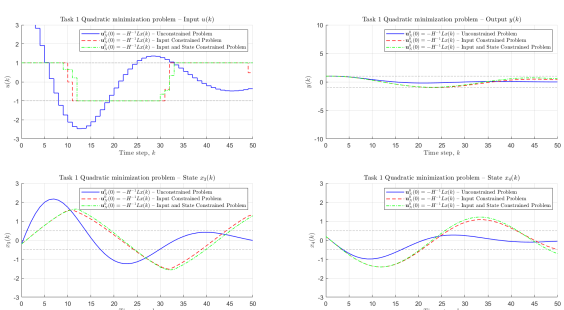

Fig. 3. Comparison of Unconstrained vs. Constrained MPC (Task.1 & 2)

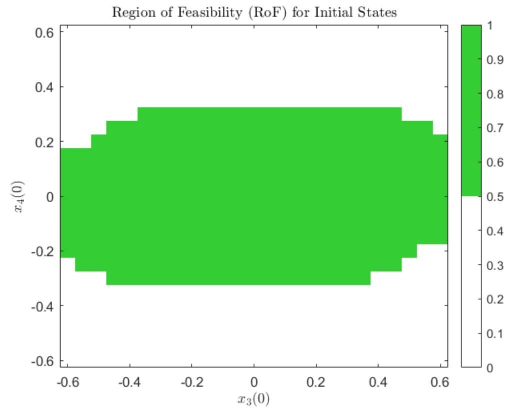

Fig. 4. Region of Feasibility (RoF) of Initial States (Task.3)

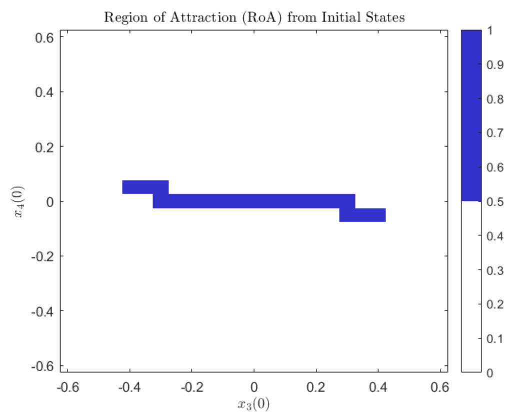

Fig. 5. Region of Attraction (RoA) of Initial States (Task.3)

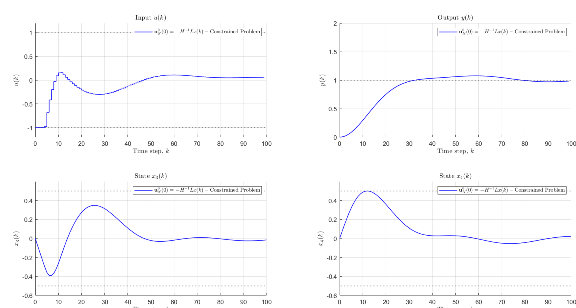

Fig. 6. Tracking MPC (Task.4)

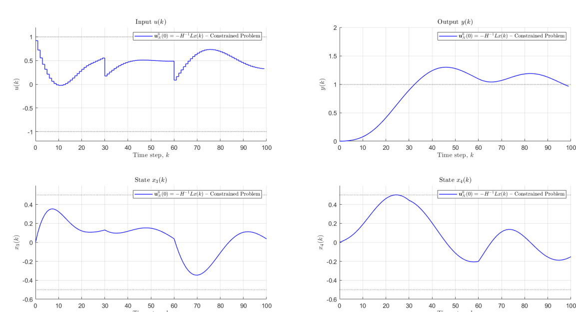

Fig. 7. Tracking MPC with Various Disturbances (Task.5)

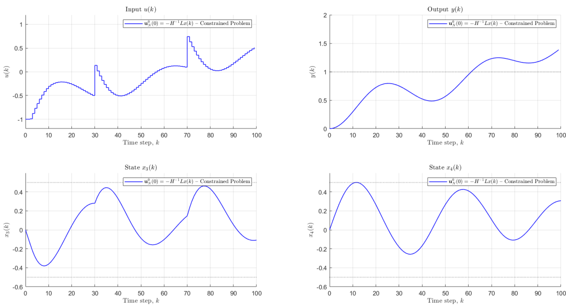

Fig. 8. Tracking MPC with Various References (Task.5)

V. Analysis and Discussion

A. Linear Quadratic Model Predictive Controller (LQ/MPC)

In this case, the input disturbance is absent, and the system is initialized at the initial state $\mathbf{x}(0) = \begin{bmatrix} 1 & 1 & -0.2 & 0.2 \end{bmatrix}^\top$. The simulation is performed using an incremental method under three scenarios: unconstrained, input-constrained, and state + input-constrained, with the constraint values specified in Equation 1.

Figure 4 represents the feasibility region for the initial conditions $x_3(0)$ and $x_4(0)$, the feasibility region of which spans $x_3(0) \in [-0.5, 0.5]$ and $x_4(0) \in [-0.4, 0.4]$, **implying that initial positions (x_1 & x_2) and velocities (x_3 & x_4) of $\mathbf{x}(0) = \begin{bmatrix} 1 & 1 & -0.2 & 0.2 \end{bmatrix}^\top$ within this region which will not cause violations of constraints during the operation of the system.**

The plot in Figure 5 illustrates the set of initial velocity conditions, specifically $x_3(0)$ and $x_4(0)$, from which only a narrow band of initial velocities around $x_4(0) \approx 0$ and $x_3(0) \in [-0.5, 0.5]$ leads to convergence.

B. LQ-MPC with Disturbance Rejection & Offset-Tracking

This section considers three distinct scenarios characterized by a single input disturbance, along with an additional scenario that introduces variations in both the disturbance vector and the reference signal. In the scenario corresponding to Task 4, the system is initialized with the state vector

$$\mathbf{x}(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^\top,$$

and is subjected to an input disturbance given by

$$\mathbf{d} = \begin{bmatrix} 0.25 & 0.7 \end{bmatrix}^\top.$$

In the first case from Table III, The LQ-MPC system was evaluated under constrained conditions, yielding the total cost J minimized to 7.3865, which effectively balancing state tracking and control effort. The control effort u_k was efficient, with a value of 7.3865. The system achieved a fast settling time of $T_s = 0.1$ seconds, and the overshoot was moderate at 9.42%, indicating acceptable transient behavior.

Furthermore, the LQ-MPC system was evaluated under constrained conditions to analyze its performance in handling repetitive disturbances and reference changes at $k = 0, 30, 60$. The controller minimized a quadratic cost function, balancing state tracking and control effort while ensuring feasibility and stability. **The results showed that the total cost increased in the disturbance scenario ($J = 117.57$) compared to the reference scenario ($J = 92.369$), reflecting the additional effort required to counteract disturbances**.

The control effort also increased ($u_k = 21.94$ vs. $u_k = 14.164$), while the settling time remained consistent at $T_s = 10$ seconds. However, the overshoot was significantly higher in the disturbance scenario (34.88%) compared to no overshoot in the reference scenario, indicating the impact of transient dynamics. Despite these charateristic, **the LQ-MPC system successfully adhered to the given constraints**, demonstrating its ability to maintain feasibility and stability under varying conditions.

TABLE IV

Performance Metrics of LQ-MPC with Various Disturbance & Reference

Scenario	Cost J	u_k	Ts	Overshoot
Reference Variety	92.369	14.164	10	0
Disturbance Variety	117.57	21.94	10	34.88

VI. Conclusion

An LQ-MPC controller was effectively developed for regulation within two-cart system, demonstrating stable performance—achieving convergence in fewer than 10 time steps—while adhering to the constraints outlined in Equations 1. Additionally, the controller exhibited robustness to variations in system time constants and demand response limitations through appropriate parameter tuning, and it managed external disturbances without significant degradation in performance.

References

[1] A. E. Bryson, “Some Connections Between Modern and Classical Control Concepts,” Journal of dynamic systems, measurement, and control, vol. 101, no. 2, pp. 91–98, 1979, doi: 10.1115/1.3426420.

[2] B. Wie and D. S. Bernstein, Benchmark problems for robust control design, Journal of Guidance, Control, and Dynamics, vol. 15, Art. no. 5, Sep. 1992.

[3] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Feedback control of dynamic systems, Eighth global edition. New York, NY: Pearson, 2020.

[4] J. A. Rossiter, A first course in predictive control, Second edition. Boca Raton; London; New York: CRC Press, Taylor & Francis Group, 2018.

[5] C. E. García, D. M. Prett, and M. Morari, “Model predictive control: Theory and practice—A survey,” Automatica (Oxford), vol. 25, no. 3, pp. 335–348, 1989, doi: 10.1016/0005-1098(89)90002-2.

[6] Y. Arkun, W. M. Canney, J. Hollett, and M. Morari, “Experimental study of internal model control,” Industrial & engineering chemistry process design and development, vol. 25, no. 1, pp. 102–108, 1986, doi: 10.1021/i200032a016.

[7] K. L. Levien, “Studies In The Design And Control Of Coupled Distillation Columns (Modeling, Collocation),” ProQuest Dissertations & Theses, 1985.

[8] K. L. Levien and M. Morari, “Internal model control of coupled distillation columns,” AIChE journal, vol. 33, no. 1, pp. 83–98, 1987, doi: 10.1002/aic.690330111.

[9] J. R. Parrish and C. B. Brosilow, “Inferential control applications,” Automatica (Oxford), vol. 21, no. 5, pp. 527–538, 1985, doi: 10.1016/0005-1098(85)90002-0.

APPENDIX

MATLAB Program for Tasks 1, 2, and 3

```
%% ACS6116 MPC Computer Assignment
% (c) P.Trodden, 2023.

%% Problem setup
clear all
close all
clc

m1 = 1;
m2 = m1;
ks = 1;

ulimit = 1;
ylimit = 2.7;
x3limit = 0.5;
x4limit = 0.5;

N = 20;
Tsim = 50;

%% System matrices
Ac = [0 0 1 0; 0 0 0 1; -ks/m1 0 0 0; ks/m2 -ks/m2 0 0];
Bc = [0 0 1/m1 0]';
Cc = [0 1 0 0];
Ec = [0 0;0 0;1/m1 0;0 1/m2];

%% Dimensions
n = size(Ac,1);
m = size(Bc,2);
p = size(Cc,1);

%% Initial state
Ts = 0.1;

sysc = ss(Ac, [Bc Ec], Cc, zeros(1,size(Bc,2)));
sysd = c2d(sysc,Ts);

A = sysd.A;
B = sysd.B(:,1);
C = sysd.C;
D = sysd.D;
E = sysd.B(:,2:3);

x0 = [-1; 1; -0.2; 0.2];
Q = C'*C;
R = 0.01;

[P, ~] = idare(A, B, Q, R, [], []);

%% Check system properties
check_ABQR(A,B,Q,R);

%% Constraints
Pu = [-1 1]';
```

```
qu = [1 1]';

Px = [ 0 0  1  0;
       0 0 -1  0;
       0 0  0  1;
       0 0  0 -1 ];
qx = [x3limit; x3limit; x4limit; x4limit];

%% Task 1 (Unconstrained)
[F,G] = predict_mats(A,B,N);
[H,L,M] = cost_mats(F,G,Q,R,P);

x = x0;
for k = 0:Tsim-1
    f = L * x;
    zopt = quadprog(H, f);
    u = zopt(1);
    y = C*x;
    xs(:,k+1) = x;
    us(:,k+1) = u;
    ys(:,k+1) = y;
    x = A*x + B*u;
end

xs(:,Tsim+1) = x;
us(:,Tsim+1) = u;
ys(:,Tsim+1) = y;

%% Task 2 (Input Constrained)
[Pc_1,qc_1,Sc_1] = constraint_mats(F,G,Pu,qu,[],[],[],[]);

x = x0;
for k = 0:Tsim-1
    f = L * x;
    zopt = quadprog(H, f, Pc_1, qc_1);
    u = zopt(1);
    y = C*x;
    xs_1(:,k+1) = x;
    us_1(:,k+1) = u;
    ys_1(:,k+1) = y;
    x = A*x + B*u;
end

xs_1(:,Tsim+1) = x;
us_1(:,Tsim+1) = u;
ys_1(:,Tsim+1) = y;

%% Task 3 (Input and State Constrained)
[Pc_2,qc_2,Sc_2] = constraint_mats(F,G,Pu,qu,Px,qx,Px,qx);

x = x0;
for k = 0:Tsim-1
    f = L * x;
    zopt = quadprog(H, f, Pc_2, qc_2);
    u = zopt(1);
    y = C*x;
    xs_2(:,k+1) = x;
    us_2(:,k+1) = u;
    ys_2(:,k+1) = y;
```

```
    x = A*x + B*u;
end

xs_2(:,Tsim+1) = x;
us_2(:,Tsim+1) = u;
ys_2(:,Tsim+1) = y;

% Plots and performance metrics follow (not shown here to save space)
```

Listing 1: MATLAB code for MPC Tasks 1–3

MATLAB Program for Tasks 4-5: Disturbance Variation

```


%% Problem setup
clear all
close all
clc

%% Define the each Variables

m1 = 1;
m2 = m1;
ks = 1;
N = 20;
Tsim = 100;

ulimit = 1;
ylimit = 2.7;
x3limit = 0.5;
x4limit = 0.5;
ylimit = 1;

%% System matrices
Ac = [0 0 1 0; 0 0 0 1; -ks/m1 0 0 0; ks/m2 -ks/m2 0 0];
Bc = [0 0 1/m1 0]';
Cc = [0 1 0 0];
Ec = [0 0; 0 0; 1/m1 0; 0 1/m2];

%% Dimensions
n = size(Ac,1);  % number of states
m = size(Bc,2);  % number of inputs
p = size(Cc,1);  % number of outputs

%% User-configurable disturbance flag
% The input disturbance exist
flag = 1;  % Set to 0 for zero disturbance, 1 for nonzero disturbance

if flag == 0
    d1 = 0;
    d2 = 0;
else
    d1 = 0.25;
    d2 = 0.7;
end

d = [d1;d2];
```

```matlab

%% Initial state
Ts = 0.1;

sysc = ss(Ac, [Bc Ec], Cc, zeros(1,size(Bc,2)));
sysd = c2d(sysc,Ts);

%% Initiate from origins
flag_1 = 1;  % Set to 0 for zero disturbance, 1 for nonzero disturbance

if flag_1 == 0
    x0 = [-1; 1; -0.2; 0.2];
else
    x0 = [0; 0; 0; 0];
end

x = zeros(n, Tsim+1);   % state trajectory
x(:,1) = x0;         % initial state

u = zeros(1, Tsim);      % input trajectory
y = zeros(1, Tsim);      % output trajectory

Q = Cc'*Cc;
R = 1;
r = 1;

A = sysd.A;
B = sysd.B(:,1);
C = sysd.C;
D = sysd.D;
E = sysd.B(:,2:3);

[P, ~] = idare(A, B, Q, R, [], []);

%% Check the Reachibility, Observability, and R and Q Definitness
check_ABQR(A,B,Q,R);

%% Constraints
% Input constraints
P_u = [1; -1];   % 2x1
q_u = [1; 1];    % 2x1

% State constraints
P_x = [0 0 1 0;
       0 0 -1 0;
       0 0 0 1;
       0 0 0 -1];
q_x = [0.5; 0.5; 0.5; 0.5];

for k = 1:Tsim


    %% Step 1: Solve for x_ss, u_ss
    Tss = [eye(n)-A, -B; C, 0];
    RHS = [E*d; r];
    ans_ss = Tss \ RHS;
    x_ss = ans_ss(1:n);
    u_ss = ans_ss(n+1:end);

```

```matlab
    %% Step 2: Compute prediction and cost matrices
    [F,G] = predict_mats(A,B,N);
    cond_F = cond(F);
    cond_G = cond(G);
    fprintf('Condition number of F: %f\n', cond_F);
    fprintf('Condition number of G: %f\n', cond_G);

    [H,L,M] = cost_mats(F,G,Q,R,P);

    %% Step 3: Shift constraints
    qx_bar = q_x - P_x * x_ss;
    qu_bar = q_u - P_u * u_ss;
    [Pc, qc, Sc] = constraint_mats(F,G,P_u,qu_bar,P_x,qx_bar,P_x,qx_bar);

    %% Step 4: Solve QP for deviation input sequence du
    x_dev = x(:,k) - x_ss;  % deviation state
    f = L * x_dev;

    % Solve QP
    options = optimoptions('quadprog', 'Display', 'none');
    [z, ~, exitflag, output] = quadprog(H, f, Pc, qc + Sc*x_dev, [], [], [], [],
        [], options);

    if exitflag ~= 1
        disp("quadprog failed at step k = " + k);
        disp(output.message);
        z = zeros(N*m,1);  % fallback zero input sequence
    end

    du = z(1);              % first control increment
    u(:,k) = du + u_ss;     % actual control input

    %% Step 5: Simulate system
    x(:,k+1) = A * x(:,k) + B * u(:,k) + E* d;
    y(:,k) = C * x(:,k);
end

%% Analyze Steady-State Error and Overshoot

% Reference value (used in tracking)
ref = r;  % Should match the one used during simulation

% Steady-state error = |final y - reference|
steady_state_error = abs(y(end) - ref);

% Overshoot = max(y) - reference (if it exceeds reference)
overshoot = max(y) - ref;
if overshoot < 0
    overshoot = 0;  % no overshoot if response is always below reference
end

% Display results
fprintf('\n--- Performance Metrics ---\n');
fprintf('Reference value: %.4f\n', ref);
fprintf('Final output y(T): %.4f\n', y(end));
fprintf('Steady-state error: %.4f\n', steady_state_error);
fprintf('Overshoot: %.4f\n', overshoot);
```

```
%% [--- Plotting code unchanged, continues here ---]

% Plotting
time = 0:Tsim-1;

figure

%% Subplot 1: Output y(k)

subplot(2,2,1)
hold on
stairs(0:Tsim-1, u, 'b-', 'LineWidth', 1);
yline(ulimit, 'k:', 'LineWidth', 1);     % Upper input constraint
yline(-ulimit, 'k:', 'LineWidth', 1);    % Lower input constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$u(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
title('Input $u(k)$', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([-1.2*ulimit 1.2*ulimit]);


%% Subplot 2: Input u(k)
subplot(2,2,2)
hold on
plot(0:Tsim-1, y, 'b-', 'LineWidth', 1);
yline(ylimit, 'k:', 'LineWidth', 1);  % Output constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$y(k)$', 'Interpreter', 'latex');
title('Output $y(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([0 2 * ylimit]);


%% Subplot 3: State x3(k)
subplot(2,2,3)
hold on
plot(0:Tsim, x(3,:), 'b-', 'LineWidth', 1);
yline(x3limit, 'k:', 'LineWidth', 1);    % Upper state constraint
yline(-x3limit, 'k:', 'LineWidth', 1);  % Lower state constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$x_3(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
title('State $x_3(k)$', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([-1.2*x3limit 1.2*x3limit]);

%% Subplot 4: State x4(k)
subplot(2,2,4)
```

```matlab
hold on
plot(0:Tsim, x(4,:), 'b-', 'LineWidth', 1);
yline(x4limit, 'k:', 'LineWidth', 1);   % Upper state constraint
yline(-x4limit, 'k:', 'LineWidth', 1);  % Lower state constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$x_4(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
title('State $x_4(k)$', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([-1.2*x4limit 1.2*x4limit]);

%% Phase Poytrait Analysis for constrain case
figure;

%% First subplot: x3 vs x4
subplot(1,1,1);
grid on;
hold on;

% Unconstrained: blue line with blue circles
plot(x(3,:), x(4,:), 'bo-', 'LineWidth', 0.5, 'MarkerSize', 4, 'MarkerFaceColor',
    'b');

% Initial and final points
% Input and State constrained
plot(x(3,1), x(4,1), 'gs', 'MarkerSize', 7, 'LineWidth', 1.2, 'DisplayName', '
    $x_0$ (Unconstrained)', 'MarkerFaceColor','w');
plot(x(3,end), x(4,end), 'r*', 'MarkerSize', 8, 'LineWidth', 1.2, 'DisplayName', '
    $x_f$ (Unconstrained)');

xline(-x3limit, 'k:','LineWidth', 1);  % Vertical line at x1 = -10
xline(x3limit,  'k:','LineWidth', 1);  % Vertical line at x1 = +10
yline(-x4limit, 'k:','LineWidth', 1);  % Horizontal line at x2 = -2
yline(x4limit,  'k:','LineWidth', 1);  % Horizontal line at x2 = +2
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Input\ and\ State\
    Constrained\ Problem',...
        '$x_0$ (Input+State Constrained)', '$x_f$ (Input+State Constrained)', ...
        'Location', 'northeast', 'Interpreter', 'latex');
title('Phase Portrait between $x_3$ vs $x_4$', 'Interpreter', 'latex');
xlim([-0.6 0.6]);
ylim([-0.6 0.6]);
xlabel('$x_3$', 'Interpreter', 'latex');
ylabel('$x_4$', 'Interpreter', 'latex');

% Define grid for initial x3 and x4 (velocity states)
x3_vals = linspace(-0.6, 0.6, 25);
x4_vals = linspace(-0.6, 0.6, 25);
[X3, X4] = meshgrid(x3_vals, x4_vals);

ROA = zeros(size(X3));   % Region of Attraction
ROF = zeros(size(X3));   % Region of Feasibility

% Loop over grid of initial states (x3, x4)
for i = 1:length(x3_vals)
    for j = 1:length(x4_vals)
        % Initial condition (assume other states zero)
```

```matlab
        x0_test = [0; 0; X3(j,i); X4(j,i)];
        x = zeros(n, Tsim+1);
        x(:,1) = x0_test;
        feasible = true;

        for k = 1:Tsim
            % Step 1: solve x_ss, u_ss
            Tss = [eye(n)-A, -B; C, 0];
            RHS = [E*d; r];
            ans_ss = Tss \ RHS;
            x_ss = ans_ss(1:n);
            u_ss = ans_ss(n+1:end);

            % Step 2: predict and cost matrices
            [F,G] = predict_mats(A,B,N);
            [H,L,M] = cost_mats(F,G,Q,R,P);

            % Step 3: shift constraints
            qx_bar = q_x - P_x * x_ss;
            qu_bar = q_u - P_u * u_ss;
            [Pc, qc, Sc] = constraint_mats(F,G,P_u,qu_bar,P_x,qx_bar,P_x,qx_bar);

            % Step 4: solve QP
            x_dev = x(:,k) - x_ss;
            f = L * x_dev;
            options = optimoptions('quadprog','Display','off');
            [z,~,exitflag] = quadprog(H, f, Pc, qc + Sc*x_dev, [], [], [], [], [],
                 options);

            if exitflag ~= 1
                feasible = false;
                break;
            end

            du = z(1);
            u_k = du + u_ss;
            x(:,k+1) = A*x(:,k) + B*u_k + E*d;

            % Check state and input constraints manually
            if any(P_x * x(:,k+1) > q_x + 1e-3) || any(P_u * u_k > q_u + 1e-3)
                feasible = false;
                break;
            end
        end

        % Check if system converged close to zero
        if norm(x(:,end),2) < 1e-2 && feasible
            ROA(j,i) = 1;
        end
        if feasible
            ROF(j,i) = 1;
        end
    end
end

% Plot Region of Attraction
figure;
imagesc(x3_vals, x4_vals, ROA);
axis xy;
```

```
xlabel('$x_3(0)$', 'Interpreter', 'latex');
ylabel('$x_4(0)$', 'Interpreter', 'latex');
title('Convergence to Origin (Region of Attraction)', 'Interpreter', 'latex');
colorbar;

% Plot Region of Feasibility
figure;
imagesc(x3_vals, x4_vals, ROF);
axis xy;
xlabel('$x_3(0)$', 'Interpreter', 'latex');
ylabel('$x_4(0)$', 'Interpreter', 'latex');
title('Feasibility Region for Initial States', 'Interpreter', 'latex');
colorbar;

% === PERFORMANCE METRICS ===
metrics = struct();

compute_overshoot = @(y) max(0, (max(y) - y(end)) / max(abs(y(end)), 1e-6));  %
    Avoid divide-by-zero

labels = {'Constrained'};
xs_all = {x};
us_all = {u};
ys_all = {y};

for idx = 1:length(labels)
    x_data = xs_all{idx};
    u_data = us_all{idx};
    y_data = ys_all{idx};

    % === COST ===
    J = 0;
    for k = 1:Tsim
        xk = x_data(:,k);
        uk = u_data(:,k);
        J = J + xk'*Q*xk + uk'*R*uk;
    end

    % === CONTROL EFFORT ===
    Ueffort = sum(u_data.^2, 'all');

    % === SETTLING TIME ===
    final_x = x_data(:,end);
    tol = 0.02 * abs(final_x + 1e-5);
    settle_idx = find(all(abs(x_data - final_x) < tol, 1), 1);
    settling_time = NaN;
    if ~isempty(settle_idx)
        settling_time = (settle_idx - 1) * Ts;
    end

    % === OVERSHOOT ===
    overshoot = compute_overshoot(y_data(1,:));

    scenario = labels{idx};
    metrics.(scenario).Cost = J;
    metrics.(scenario).ControlEffort = Ueffort;
    metrics.(scenario).SettlingTime = settling_time;
    metrics.(scenario).Overshoot = overshoot;
end
```

```
% === Display summary ===
summary_table = table(...
    metrics.Constrained.Cost, ...
    metrics.Constrained.ControlEffort, ...
    metrics.Constrained.SettlingTime, ...
    100 * metrics.Constrained.Overshoot, ...
    'VariableNames', {'Cost', 'ControlEffort', 'SettlingTime', 'OvershootPercent'
       }, ...
    'RowNames', {'Constrained'});

disp('Performance Metrics Summary:');
disp(summary_table);
```

Listing 2: MATLAB code for MPC Tasks 4–5: Disturbance Variation

MATLAB Program for Tasks 4-5: Reference Variation

```

%% Problem setup
clear all
close all
clc

%% Define the each Variables

m1 = 1;
m2 = m1;
ks = 1;
N = 20;
Tsim = 100;

ulimit = 1;
ylimit = 2.7;
x3limit = 0.5;
x4limit = 0.5;
ylimit = 1;

%% System matrices
Ac = [0 0 1 0; 0 0 0 1; -ks/m1 0 0 0; ks/m2 -ks/m2 0 0];
Bc = [0 0 1/m1 0]';
Cc = [0 1 0 0];
Ec = [0 0; 0 0; 1/m1 0; 0 1/m2];

%% Dimensions
n = size(Ac,1);  % number of states
m = size(Bc,2);  % number of inputs
p = size(Cc,1);  % number of outputs

%% User-configurable disturbance flag
% The input disturbance exist
flag = 1;  % Set to 0 for zero disturbance, 1 for nonzero disturbance

if flag == 0
    d1 = 0;
    d2 = 0;
else
    d1 = 0.25;
```

```
    d2 = 0.7;
end

d = [d1;d2];

%% Initial state
Ts = 0.1;

sysc = ss(Ac, [Bc Ec], Cc, zeros(1,size(Bc,2)));
sysd = c2d(sysc,Ts);

%% Initiate from origins
flag_1 = 1;  % Set to 0 for zero disturbance, 1 for nonzero disturbance

if flag_1 == 0
    x0 = [-1; 1; -0.2; 0.2];
else
    x0 = [0; 0; 0; 0];
end

x = zeros(n, Tsim+1);   % state trajectory
x(:,1) = x0;          % initial state

u = zeros(1, Tsim);      % input trajectory
y = zeros(1, Tsim);      % output trajectory

Q = Cc'*Cc;
R = 1;

r_vec = zeros(1, Tsim);
for k = 1:Tsim
    if k <= 30
        r_vec(k) = 0.5;
    elseif k <= 70
        r_vec(k) = 1.0;
    else
        r_vec(k) = 1.5;
    end
end

A = sysd.A;
B = sysd.B(:,1);
C = sysd.C;
D = sysd.D;
E = sysd.B(:,2:3);

[P, ~] = idare(A, B, Q, R, [], []);

%% Check the Reachibility, Observability, and R and Q Definitness
check_ABQR(A,B,Q,R);

%% Constraints
% Input constraints
P_u = [1; -1];    % 2x1
q_u = [1; 1];     % 2x1

% State constraints
P_x = [0 0 1 0;
       0 0 -1 0;
```

```
        0  0  0  1;
        0  0  0  -1];
q_x = [0.5; 0.5; 0.5; 0.5];

for k = 1:Tsim


    %% Step 1: Solve for x_ss, u_ss
    Tss = [eye(n)-A, -B; C, 0];
    r = r_vec(k);
    RHS = [E*d; r];
    ans_ss = Tss \ RHS;
    x_ss = ans_ss(1:n);
    u_ss = ans_ss(n+1:end);

    %% Step 2: Compute prediction and cost matrices
    [F,G] = predict_mats(A,B,N);
    cond_F = cond(F);
    cond_G = cond(G);
    fprintf('Condition number of F: %f\n', cond_F);
    fprintf('Condition number of G: %f\n', cond_G);

    [H,L,M] = cost_mats(F,G,Q,R,P);

    %% Step 3: Shift constraints
    qx_bar = q_x - P_x * x_ss;
    qu_bar = q_u - P_u * u_ss;
    [Pc, qc, Sc] = constraint_mats(F,G,P_u,qu_bar,P_x,qx_bar,P_x,qx_bar);

    %% Step 4: Solve QP for deviation input sequence du
    x_dev = x(:,k) - x_ss;  % deviation state
    f = L * x_dev;

    % Solve QP
    options = optimoptions('quadprog', 'Display', 'none');
    [z, ~, exitflag, output] = quadprog(H, f, Pc, qc + Sc*x_dev, [], [], [], [],
        [], options);

    if exitflag ~= 1
        disp("quadprog failed at step k = " + k);
        disp(output.message);
        z = zeros(N*m,1);  % fallback zero input sequence
    end

    du = z(1);              % first control increment
    u(:,k) = du + u_ss;     % actual control input

    %% Step 5: Simulate system
    x(:,k+1) = A * x(:,k) + B * u(:,k) + E* d;
    y(:,k) = C * x(:,k);
end

%% Analyze Steady-State Error and Overshoot

% Reference value (used in tracking)
ref = r;  % Should match the one used during simulation

% Steady-state error = |final y - reference|
steady_state_error = abs(y(end) - ref);
```

```matlab

% Overshoot = max(y) - reference (if it exceeds reference)
overshoot = max(y) - ref;
if overshoot < 0
    overshoot = 0;  % no overshoot if response is always below reference
end

% Display results
fprintf('\n--- Performance Metrics ---\n');
fprintf('Reference value: %.4f\n', ref);
fprintf('Final output y(T): %.4f\n', y(end));
fprintf('Steady-state error: %.4f\n', steady_state_error);
fprintf('Overshoot: %.4f\n', overshoot);


%% [--- Plotting code unchanged, continues here ---]

% Plotting
time = 0:Tsim-1;

figure

%% Subplot 1: Output y(k)

subplot(2,2,1)
hold on
stairs(0:Tsim-1, u, 'b-', 'LineWidth', 1);
% yline(ulimit, 'k:', 'LineWidth', 1);    % Upper input constraint
% yline(-ulimit, 'k:', 'LineWidth', 1);   % Lower input constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$u(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
title('Input $u(k)$', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([-1.2*ulimit 1.2*ulimit]);


%% Subplot 2: Input u(k)
subplot(2,2,2)
hold on
plot(0:Tsim-1, y, 'b-', 'LineWidth', 1);
yline(ylimit, 'k:', 'LineWidth', 1);  % Output constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$y(k)$', 'Interpreter', 'latex');
title('Output $y(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([0 2 * ylimit]);


%% Subplot 3: State x3(k)
subplot(2,2,3)
hold on
```

```matlab
plot(0:Tsim, x(3,:), 'b-', 'LineWidth', 1);
yline(x3limit, 'k:', 'LineWidth', 1);   % Upper state constraint
yline(-x3limit, 'k:', 'LineWidth', 1);  % Lower state constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$x_3(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
title('State $x_3(k)$', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([-1.2*x3limit 1.2*x3limit]);

%% Subplot 4: State x4(k)
subplot(2,2,4)
hold on
plot(0:Tsim, x(4,:), 'b-', 'LineWidth', 1);
yline(x4limit, 'k:', 'LineWidth', 1);   % Upper state constraint
yline(-x4limit, 'k:', 'LineWidth', 1);  % Lower state constraint
xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$x_4(k)$', 'Interpreter', 'latex');
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Constrained\ Problem
    ',...
    'Location', 'northeast', 'Interpreter', 'latex');
title('State $x_4(k)$', 'Interpreter', 'latex');
grid on
xlim([0 Tsim]);
ylim([-1.2*x4limit 1.2*x4limit]);

%% Phase Poytrait Analysis for constrain case
figure;

%% First subplot: x3 vs x4
subplot(1,1,1);
grid on;
hold on;

% Unconstrained: blue line with blue circles
plot(x(3,:), x(4,:), 'bo-', 'LineWidth', 0.5, 'MarkerSize', 4, 'MarkerFaceColor',
    'b');

% Initial and final points
% Input and State constrained
plot(x(3,1), x(4,1), 'gs', 'MarkerSize', 7, 'LineWidth', 1.2, 'DisplayName', '
    $x_0$ (Unconstrained)', 'MarkerFaceColor','w');
plot(x(3,end), x(4,end), 'r*', 'MarkerSize', 8, 'LineWidth', 1.2, 'DisplayName', '
    $x_f$ (Unconstrained)');

xline(-x3limit, 'k:','LineWidth', 1);  % Vertical line at x1 = -10
xline(x3limit,  'k:','LineWidth', 1);  % Vertical line at x1 = +10
yline(-x4limit, 'k:','LineWidth', 1);  % Horizontal line at x2 = -2
yline(x4limit,  'k:','LineWidth', 1);  % Horizontal line at x2 = +2
legend('$\mathbf{u}^{0}_{N}(0) = -H^{-1} L x(k)$ \textendash\ Input\ and\ State\
    Constrained\ Problem',...
        '$x_0$ (Input+State Constrained)', '$x_f$ (Input+State Constrained)', ...
        'Location', 'northeast', 'Interpreter', 'latex');
title('Phase Portrait between $x_3$ vs $x_4$', 'Interpreter', 'latex');
xlim([-0.6 0.6]);
ylim([-0.6 0.6]);
```

```matlab
xlabel('$x_3$', 'Interpreter', 'latex');
ylabel('$x_4$', 'Interpreter', 'latex');

figure;
plot(0:Tsim-1, r_vec, 'k--', 'LineWidth', 1.2); hold on;
plot(0:Tsim-1, y, 'b-', 'LineWidth', 1.2);

xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
ylabel('$y(k)\ \mathrm{and\ reference\ } r(k)$', 'Interpreter', 'latex');
legend({'$r(k)$ \textendash\ Reference', '$y(k)$ \textendash\ Output'}, ...
       'Interpreter', 'latex', 'Location', 'best');
title('Tracking Response with Varying Reference $r(k)$', 'Interpreter', 'latex');
grid on;

% === PERFORMANCE METRICS ===
metrics = struct();

compute_overshoot = @(y) max(0, (max(y) - y(end)) / max(abs(y(end)), 1e-6));  % Avoid divide-by-zero

labels = {'Constrained'};
xs_all = {x};
us_all = {u};
ys_all = {y};

for idx = 1:length(labels)
    x_data = xs_all{idx};
    u_data = us_all{idx};
    y_data = ys_all{idx};

    % === COST ===
    J = 0;
    for k = 1:Tsim
        xk = x_data(:,k);
        uk = u_data(:,k);
        J = J + xk'*Q*xk + uk'*R*uk;
    end

    % === CONTROL EFFORT ===
    Ueffort = sum(u_data.^2, 'all');

    % === SETTLING TIME ===
    final_x = x_data(:,end);
    tol = 0.02 * abs(final_x + 1e-5);
    settle_idx = find(all(abs(x_data - final_x) < tol, 1), 1);
    settling_time = NaN;
    if ~isempty(settle_idx)
        settling_time = (settle_idx - 1) * Ts;
    end

    % === OVERSHOOT ===
    overshoot = compute_overshoot(y_data(1,:));

    scenario = labels{idx};
    metrics.(scenario).Cost = J;
    metrics.(scenario).ControlEffort = Ueffort;
    metrics.(scenario).SettlingTime = settling_time;
    metrics.(scenario).Overshoot = overshoot;
end
```

```matlab

% === Display summary ===
summary_table = table(...
    metrics.Constrained.Cost, ...
    metrics.Constrained.ControlEffort, ...
    metrics.Constrained.SettlingTime, ...
    100 * metrics.Constrained.Overshoot, ...
    'VariableNames', {'Cost', 'ControlEffort', 'SettlingTime', 'OvershootPercent'
        }, ...
    'RowNames', {'Constrained'});

disp('Performance Metrics Summary:');
disp(summary_table);
```

Listing 3: MATLAB code for MPC Tasks 4–5: Reference Variataion