

ACS6116 Assignment

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

Muzhaffar Maruf Ibrahim¹

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, $|x_3| \leq 0.5$, $|x_4| \leq 0.5$. 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 multi-variable 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.

II. PROBLEM STATEMENT

A. Two-mass-spring with uncertain parameters system

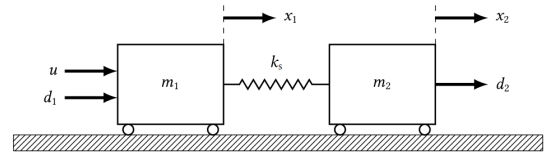


Fig. 1. The Wie–Bernstein two-cart control benchmark system

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:

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

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

^{*}This work was not supported by any organization

¹Muzhaffar 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

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:

$$|u| \leq 1, \quad |x_3| \leq 0.5, \quad |x_4| \leq 0.5. \quad (1)$$

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

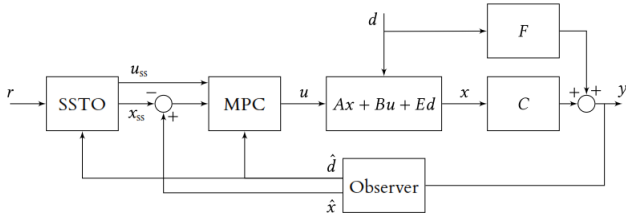


Fig. 2. Block diagram of a feedforward approach to tracking MPC

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:

$$x(k+1) = Ax(k) + Bu(k) + Ed(k), \quad (2)$$

$$y(k) = Cx(k) + Fd(k), \quad (3)$$

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

$$x_{ss} = Ax_{ss} + Bu_{ss} + Ed, \quad (4)$$

$$r = Cx_{ss} + Fd. \quad (5)$$

These yield the linear system:

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

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

Then, the control law defined as:

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

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

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

for $i = 1, \dots, N$, 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:

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

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:

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

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

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

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

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

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 $z = x - x_{ss}$, $v = u - u_{ss}$ 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 <code>predict_mats</code> (A , B , N) |
| 8. | Compute H , L , M using <code>cost_mats</code> (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 <code>constraint_mats</code> (F , G , q) |
| 12. | Set initial deviation state $z(0) = x(0) - x_{ss}$ |
| 13. | Solve control input $v(k)$ with <code>quadprog</code> at each step |

TABLE I
SIMPLIFIED MPC DESIGN PROCESS

TABLE II
PERFORMANCE METRICS OF LQ-MPC

| Scenario | Cost J | u_k | T_s | 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 | T_s | Overshoot |
|-----------------|----------|--------|-------|-----------|
| Offset-Tracking | 7.3865 | 7.3865 | 0.1 | 9.4244 |

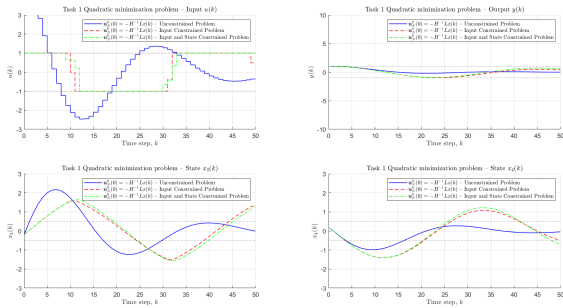


Fig. 3. Comparison of Unconstrained vs. Constrained MPC (Task.1 & 2)

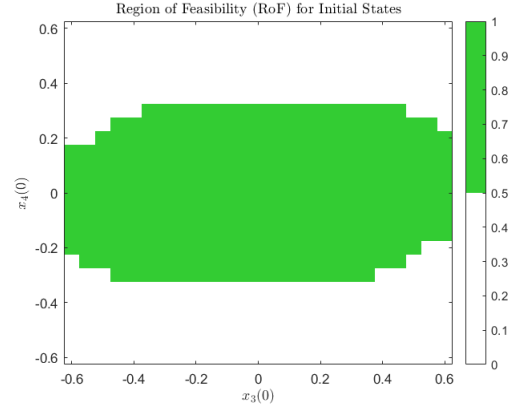


Fig. 4. Region of Feasibility (RoF) of Initial States (Task.3)

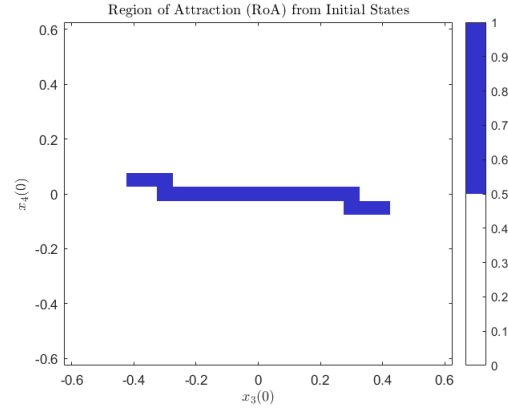


Fig. 5. Region of Attraction (RoA) of Initial States (Task.3)

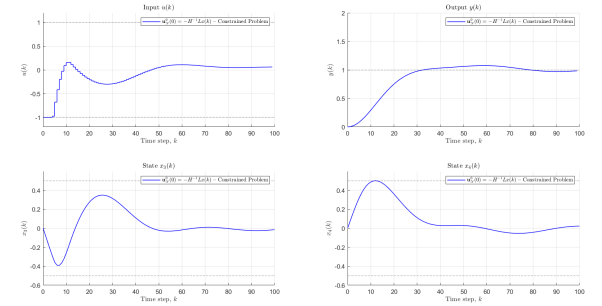


Fig. 6. Tracking MPC (Task.4)

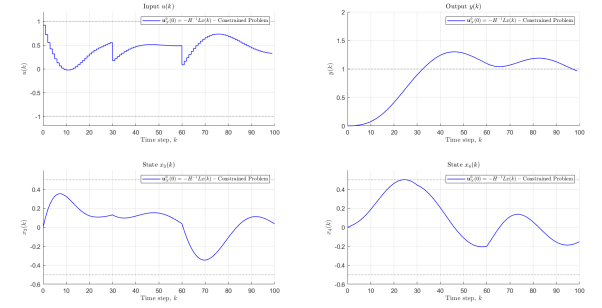


Fig. 7. Tracking MPC with Various Disturbances (Task.5)

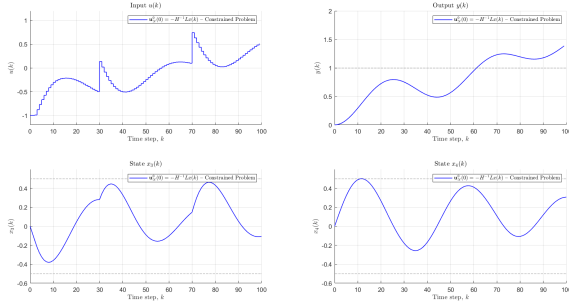


Fig. 8. Tracking MPC with Various References (Task.5)

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 $\mathbf{x}(0) = [1 \ 1 \ -0.2 \ 0.2]^T$. 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 $\mathbf{x}(0) = [1 \ 1 \ -0.2 \ 0.2]^T$ 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

$$\mathbf{x}(0) = [0 \ 0 \ 0 \ 0]^T,$$

and is subjected to an input disturbance given by

$$\mathbf{d} = [0.25 \ 0.7]^T.$$

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 characteristic, **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 | T_s | 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

```
1 %% ACS6116 MPC Computer Assignment
2 % (c) P.Trodden, 2023.
3
4 %% Problem setup
5 clear all
6 close all
7 clc
8
9 m1 = 1;
10 m2 = m1;
11 ks = 1;
12
13 ulimit = 1;
14 ylimit = 2.7;
15 x3limit = 0.5;
16 x4limit = 0.5;
17
18 N = 20;
19 Tsim = 50;
20
21 %% System matrices
22 Ac = [0 0 1 0; 0 0 0 1; -ks/m1 0 0 0; ks/m2 -ks/m2 0 0];
23 Bc = [0 0 1/m1 0]';
24 Cc = [0 1 0 0];
25 Ec = [0 0; 0 0; 1/m1 0; 0 1/m2];
26
27 %% Dimensions
28 n = size(Ac,1);
29 m = size(Bc,2);
30 p = size(Cc,1);
31
32 %% Initial state
33 Ts = 0.1;
34
35 sysc = ss(Ac, [Bc Ec], Cc, zeros(1,size(Bc,2)));
36 sysd = c2d(sysc,Ts);
37
38 A = sysd.A;
39 B = sysd.B(:,1);
40 C = sysd.C;
41 D = sysd.D;
42 E = sysd.B(:,2:3);
43
44 x0 = [-1; 1; -0.2; 0.2];
45 Q = C'*C;
46 R = 0.01;
47
48 [P, ~] = idare(A, B, Q, R, [], []);
49
50 %% Check system properties
51 check_ABQR(A,B,Q,R);
52
53 %% Constraints
54 Pu = [-1 1]';
```

```

55 qu = [1 1]';
56
57 Px = [ 0 0 1 0;
58        0 0 -1 0;
59        0 0 0 1;
60        0 0 0 -1 ];
61 qx = [x3limit; x3limit; x4limit; x4limit];
62
63 %% Task 1 (Unconstrained)
64 [F,G] = predict_mats(A,B,N);
65 [H,L,M] = cost_mats(F,G,Q,R,P);
66
67 x = x0;
68 for k = 0:Tsim-1
69     f = L * x;
70     zopt = quadprog(H, f);
71     u = zopt(1);
72     y = C*x;
73     xs(:,k+1) = x;
74     us(:,k+1) = u;
75     ys(:,k+1) = y;
76     x = A*x + B*u;
77 end
78
79 xs(:,Tsim+1) = x;
80 us(:,Tsim+1) = u;
81 ys(:,Tsim+1) = y;
82
83 %% Task 2 (Input Constrained)
84 [Pc_1,qc_1,Sc_1] = constraint_mats(F,G,Pu,qu,[],[],[],[]);
85
86 x = x0;
87 for k = 0:Tsim-1
88     f = L * x;
89     zopt = quadprog(H, f, Pc_1, qc_1);
90     u = zopt(1);
91     y = C*x;
92     xs_1(:,k+1) = x;
93     us_1(:,k+1) = u;
94     ys_1(:,k+1) = y;
95     x = A*x + B*u;
96 end
97
98 xs_1(:,Tsim+1) = x;
99 us_1(:,Tsim+1) = u;
100 ys_1(:,Tsim+1) = y;
101
102 %% Task 3 (Input and State Constrained)
103 [Pc_2,qc_2,Sc_2] = constraint_mats(F,G,Pu,qu,Px,qx,Px,qx);
104
105 x = x0;
106 for k = 0:Tsim-1
107     f = L * x;
108     zopt = quadprog(H, f, Pc_2, qc_2);
109     u = zopt(1);
110     y = C*x;
111     xs_2(:,k+1) = x;
112     us_2(:,k+1) = u;
113     ys_2(:,k+1) = y;

```

```

114     x = A*x + B*u;
115 end
116
117 xs_2(:,Tsim+1) = x;
118 us_2(:,Tsim+1) = u;
119 ys_2(:,Tsim+1) = y;
120
121 % 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

```

1
2
3 %% Problem setup
4 clear all
5 close all
6 clc
7
8 %% Define the each Variables
9
10 m1 = 1;
11 m2 = m1;
12 ks = 1;
13 N = 20;
14 Tsim = 100;
15
16 ulimit = 1;
17 ylimit = 2.7;
18 x3limit = 0.5;
19 x4limit = 0.5;
20 ylimit = 1;
21
22 %% System matrices
23 Ac = [0 0 1 0; 0 0 0 1; -ks/m1 0 0 0; ks/m2 -ks/m2 0 0];
24 Bc = [0 0 1/m1 0]';
25 Cc = [0 1 0 0];
26 Ec = [0 0; 0 0; 1/m1 0; 0 1/m2];
27
28 %% Dimensions
29 n = size(Ac,1); % number of states
30 m = size(Bc,2); % number of inputs
31 p = size(Cc,1); % number of outputs
32
33 %% User-configurable disturbance flag
34 % The input disturbance exist
35 flag = 1; % Set to 0 for zero disturbance, 1 for nonzero disturbance
36
37 if flag == 0
38     d1 = 0;
39     d2 = 0;
40 else
41     d1 = 0.25;
42     d2 = 0.7;
43 end
44
45 d = [d1;d2];

```

```

46
47 %% Initial state
48 Ts = 0.1;
49
50 sysc = ss(Ac, [Bc Ec], Cc, zeros(1,size(Bc,2)));
51 sysd = c2d(sysc,Ts);
52
53 %% Initiate from origins
54 flag_1 = 1; % Set to 0 for zero disturbance, 1 for nonzero disturbance
55
56 if flag_1 == 0
57     x0 = [-1; 1; -0.2; 0.2];
58 else
59     x0 = [0; 0; 0; 0];
60 end
61
62 x = zeros(n, Tsim+1); % state trajectory
63 x(:,1) = x0; % initial state
64
65 u = zeros(1, Tsim); % input trajectory
66 y = zeros(1, Tsim); % output trajectory
67
68 Q = Cc'*Cc;
69 R = 1;
70 r = 1;
71
72 A = sysd.A;
73 B = sysd.B(:,1);
74 C = sysd.C;
75 D = sysd.D;
76 E = sysd.B(:,2:3);
77
78 [P, ~] = idare(A, B, Q, R, [], []);
79
80 %% Check the Reachability, Observability, and R and Q Definiteness
81 check_ABQR(A,B,Q,R);
82
83 %% Constraints
84 % Input constraints
85 P_u = [1; -1]; % 2x1
86 q_u = [1; 1]; % 2x1
87
88 % State constraints
89 P_x = [0 0 1 0;
90        0 0 -1 0;
91        0 0 0 1;
92        0 0 0 -1];
93 q_x = [0.5; 0.5; 0.5; 0.5];
94
95 for k = 1:Tsim
96
97     %% Step 1: Solve for x_ss, u_ss
98     Tss = [eye(n)-A, -B; C, 0];
99     RHS = [E*d; r];
100     ans_ss = Tss \ RHS;
101     x_ss = ans_ss(1:n);
102     u_ss = ans_ss(n+1:end);
103
104

```



```

105 %% Step 2: Compute prediction and cost matrices
106 [F,G] = predict_mats(A,B,N);
107 cond_F = cond(F);
108 cond_G = cond(G);
109 fprintf('Condition number of F: %f\n', cond_F);
110 fprintf('Condition number of G: %f\n', cond_G);
111
112 [H,L,M] = cost_mats(F,G,Q,R,P);
113
114 %% Step 3: Shift constraints
115 qx_bar = q_x - P_x * x_ss;
116 qu_bar = q_u - P_u * u_ss;
117 [Pc, qc, Sc] = constraint_mats(F,G,P_u,qu_bar,P_x,qx_bar,P_x,qx_bar);
118
119 %% Step 4: Solve QP for deviation input sequence du
120 x_dev = x(:,k) - x_ss; % deviation state
121 f = L * x_dev;
122
123 % Solve QP
124 options = optimoptions('quadprog', 'Display', 'none');
125 [z, ~, exitflag, output] = quadprog(H, f, Pc, qc + Sc*x_dev, [], [], [], [],
126     [], options);
127
128 if exitflag ~= 1
129     disp("quadprog failed at step k = " + k);
130     disp(output.message);
131     z = zeros(N*m,1); % fallback zero input sequence
132 end
133
134 du = z(1); % first control increment
135 u(:,k) = du + u_ss; % actual control input
136
137 %% Step 5: Simulate system
138 x(:,k+1) = A * x(:,k) + B * u(:,k) + E * d;
139 y(:,k) = C * x(:,k);
140
141 end
142
143 %% Analyze Steady-State Error and Overshoot
144
145 % Reference value (used in tracking)
146 ref = r; % Should match the one used during simulation
147
148 % Steady-state error = |final y - reference|
149 steady_state_error = abs(y(end) - ref);
150
151 % Overshoot = max(y) - reference (if it exceeds reference)
152 overshoot = max(y) - ref;
153 if overshoot < 0
154     overshoot = 0; % no overshoot if response is always below reference
155 end
156
157 % Display results
158 fprintf('\n--- Performance Metrics ---\n');
159 fprintf('Reference value: %.4f\n', ref);
160 fprintf('Final output y(T): %.4f\n', y(end));
161 fprintf('Steady-state error: %.4f\n', steady_state_error);
162 fprintf('Overshoot: %.4f\n', overshoot);

```

```

163 %% [--- Plotting code unchanged, continues here ---]
164
165 % Plotting
166 time = 0:Tsim-1;
167
168 figure
169
170 %% Subplot 1: Output y(k)
171
172 subplot(2,2,1)
173 hold on
174 stairs(0:Tsim-1, u, 'b-', 'LineWidth', 1);
175 yline(ulimit, 'k:', 'LineWidth', 1); % Upper input constraint
176 yline(-ulimit, 'k:', 'LineWidth', 1); % Lower input constraint
177 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
178 ylabel('$u(k)$', 'Interpreter', 'latex');
179 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
180         'Location', 'northeast', 'Interpreter', 'latex');
181 title('Input $u(k)$', 'Interpreter', 'latex');
182 grid on
183 xlim([0 Tsim]);
184 ylim([-1.2*ulimit 1.2*ulimit]);
185
186
187 %% Subplot 2: Input u(k)
188 subplot(2,2,2)
189 hold on
190 plot(0:Tsim-1, y, 'b-', 'LineWidth', 1);
191 yline(ylim, 'k:', 'LineWidth', 1); % Output constraint
192 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
193 ylabel('$y(k)$', 'Interpreter', 'latex');
194 title('Output $y(k)$', 'Interpreter', 'latex');
195 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
196         'Location', 'northeast', 'Interpreter', 'latex');
197 grid on
198 xlim([0 Tsim]);
199 ylim([0 2 * ylim]);
200
201
202 %% Subplot 3: State x3(k)
203 subplot(2,2,3)
204 hold on
205 plot(0:Tsim, x(3,:), 'b-', 'LineWidth', 1);
206 yline(x3limit, 'k:', 'LineWidth', 1); % Upper state constraint
207 yline(-x3limit, 'k:', 'LineWidth', 1); % Lower state constraint
208 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
209 ylabel('$x_3(k)$', 'Interpreter', 'latex');
210 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
211         'Location', 'northeast', 'Interpreter', 'latex');
212 title('State $x_3(k)$', 'Interpreter', 'latex');
213 grid on
214 xlim([0 Tsim]);
215 ylim([-1.2*x3limit 1.2*x3limit]);
216
217 %% Subplot 4: State x4(k)
218 subplot(2,2,4)

```

```

219 hold on
220 plot(0:Tsim, x(4,:), 'b-', 'LineWidth', 1);
221 yline(x4limit, 'k:', 'LineWidth', 1); % Upper state constraint
222 yline(-x4limit, 'k:', 'LineWidth', 1); % Lower state constraint
223 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
224 ylabel('$x_4(k)$', 'Interpreter', 'latex');
225 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
226         'Location', 'northeast', 'Interpreter', 'latex');
227 title('State $x_4(k)$', 'Interpreter', 'latex');
228 grid on
229 xlim([0 Tsim]);
230 ylim([-1.2*x4limit 1.2*x4limit]);
231
232 %% Phase Poytrait Analysis for constrain case
233 figure;
234
235 %% First subplot: x3 vs x4
236 subplot(1,1,1);
237 grid on;
238 hold on;
239
240 % Unconstrained: blue line with blue circles
241 plot(x(3,:), x(4,:), 'bo-', 'LineWidth', 0.5, 'MarkerSize', 4, 'MarkerFaceColor', 'b');
242
243 % Initial and final points
244 % Input and State constrained
245 plot(x(3,1), x(4,1), 'gs', 'MarkerSize', 7, 'LineWidth', 1.2, 'DisplayName', '$x_0$ (Unconstrained)', 'MarkerFaceColor', 'w');
246 plot(x(3,end), x(4,end), 'r*', 'MarkerSize', 8, 'LineWidth', 1.2, 'DisplayName', '$x_f$ (Unconstrained)');
247
248 xline(-x3limit, 'k:', 'LineWidth', 1); % Vertical line at x1 = -10
249 xline(x3limit, 'k:', 'LineWidth', 1); % Vertical line at x1 = +10
250 yline(-x4limit, 'k:', 'LineWidth', 1); % Horizontal line at x2 = -2
251 yline(x4limit, 'k:', 'LineWidth', 1); % Horizontal line at x2 = +2
252 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Input\ and\ State\ Constrained\ Problem', ...
253         '$x_0$ (Input+State Constrained)', '$x_f$ (Input+State Constrained)', ...
254         'Location', 'northeast', 'Interpreter', 'latex');
255 title('Phase Portrait between $x_3$ vs $x_4$', 'Interpreter', 'latex');
256 xlim([-0.6 0.6]);
257 ylim([-0.6 0.6]);
258 xlabel('$x_3$', 'Interpreter', 'latex');
259 ylabel('$x_4$', 'Interpreter', 'latex');
260
261 % Define grid for initial x3 and x4 (velocity states)
262 x3_vals = linspace(-0.6, 0.6, 25);
263 x4_vals = linspace(-0.6, 0.6, 25);
264 [X3, X4] = meshgrid(x3_vals, x4_vals);
265
266 ROA = zeros(size(X3)); % Region of Attraction
267 ROF = zeros(size(X3)); % Region of Feasibility
268
269 % Loop over grid of initial states (x3, x4)
270 for i = 1:length(x3_vals)
271     for j = 1:length(x4_vals)
272         % Initial condition (assume other states zero)

```

```

273     x0_test = [0; 0; X3(j,i); X4(j,i)];
274     x = zeros(n, Tsim+1);
275     x(:,1) = x0_test;
276     feasible = true;
277
278     for k = 1:Tsim
279         % Step 1: solve x_ss, u_ss
280         Tss = [eye(n)-A, -B; C, 0];
281         RHS = [E*d; r];
282         ans_ss = Tss \ RHS;
283         x_ss = ans_ss(1:n);
284         u_ss = ans_ss(n+1:end);
285
286         % Step 2: predict and cost matrices
287         [F,G] = predict_mats(A,B,N);
288         [H,L,M] = cost_mats(F,G,Q,R,P);
289
290         % Step 3: shift constraints
291         qx_bar = q_x - P_x * x_ss;
292         qu_bar = q_u - P_u * u_ss;
293         [Pc, qc, Sc] = constraint_mats(F,G,P_u,qu_bar,P_x,qx_bar,P_x,qx_bar);
294
295         % Step 4: solve QP
296         x_dev = x(:,k) - x_ss;
297         f = L * x_dev;
298         options = optimoptions('quadprog','Display','off');
299         [z,~,exitflag] = quadprog(H, f, Pc, qc + Sc*x_dev, [], [], [], [], [],
300                                 options);
301
302         if exitflag ~= 1
303             feasible = false;
304             break;
305         end
306
307         du = z(1);
308         u_k = du + u_ss;
309         x(:,k+1) = A*x(:,k) + B*u_k + E*d;
310
311         % Check state and input constraints manually
312         if any(P_x * x(:,k+1) > q_x + 1e-3) || any(P_u * u_k > q_u + 1e-3)
313             feasible = false;
314             break;
315         end
316     end
317
318     % Check if system converged close to zero
319     if norm(x(:,end),2) < 1e-2 && feasible
320         ROA(j,i) = 1;
321     end
322     if feasible
323         ROF(j,i) = 1;
324     end
325 end
326
327 % Plot Region of Attraction
328 figure;
329 imagesc(x3_vals, x4_vals, ROA);
330 axis xy;

```

```

331 xlabel('$x_3(0)$', 'Interpreter', 'latex');
332 ylabel('$x_4(0)$', 'Interpreter', 'latex');
333 title('Convergence to Origin (Region of Attraction)', 'Interpreter', 'latex');
334 colorbar;
335
336 % Plot Region of Feasibility
337 figure;
338 imagesc(x3_vals, x4_vals, ROF);
339 axis xy;
340 xlabel('$x_3(0)$', 'Interpreter', 'latex');
341 ylabel('$x_4(0)$', 'Interpreter', 'latex');
342 title('Feasibility Region for Initial States', 'Interpreter', 'latex');
343 colorbar;
344
345 % === PERFORMANCE METRICS ===
346 metrics = struct();
347
348 compute_overshoot = @(y) max(0, (max(y) - y(end)) / max(abs(y(end)), 1e-6)); %
    Avoid divide-by-zero
349
350 labels = {'Constrained'};
351 xs_all = {x};
352 us_all = {u};
353 ys_all = {y};
354
355 for idx = 1:length(labels)
356     x_data = xs_all{idx};
357     u_data = us_all{idx};
358     y_data = ys_all{idx};
359
360     % === COST ===
361     J = 0;
362     for k = 1:Tsim
363         xk = x_data(:,k);
364         uk = u_data(:,k);
365         J = J + xk'*Q*xk + uk'*R*uk;
366     end
367
368     % === CONTROL EFFORT ===
369     Ueffort = sum(u_data.^2, 'all');
370
371     % === SETTLING TIME ===
372     final_x = x_data(:,end);
373     tol = 0.02 * abs(final_x + 1e-5);
374     settle_idx = find(all(abs(x_data - final_x) < tol, 1), 1);
375     settling_time = NaN;
376     if ~isempty(settle_idx)
377         settling_time = (settle_idx - 1) * Ts;
378     end
379
380     % === OVERSHOOT ===
381     overshoot = compute_overshoot(y_data(1,:));
382
383     scenario = labels{idx};
384     metrics.(scenario).Cost = J;
385     metrics.(scenario).ControlEffort = Ueffort;
386     metrics.(scenario).SettlingTime = settling_time;
387     metrics.(scenario).Overshoot = overshoot;
388 end

```

```

389
390 % === Display summary ===
391 summary_table = table(...
392     metrics.Constrained.Cost, ...
393     metrics.Constrained.ControlEffort, ...
394     metrics.Constrained.SettlingTime, ...
395     100 * metrics.Constrained.Overshoot, ...
396     'VariableNames', {'Cost', 'ControlEffort', 'SettlingTime', 'OvershootPercent',
397     }, ...
398     'RowNames', {'Constrained'});
399 disp('Performance Metrics Summary:');
400 disp(summary_table);

```

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

MATLAB Program for Tasks 4-5: Reference Variation

```

1
2 %% Problem setup
3 clear all
4 close all
5 clc
6
7 %% Define the each Variables
8
9 m1 = 1;
10 m2 = m1;
11 ks = 1;
12 N = 20;
13 Tsim = 100;
14
15 ulimit = 1;
16 ylimit = 2.7;
17 x3limit = 0.5;
18 x4limit = 0.5;
19 ylimit = 1;
20
21 %% System matrices
22 Ac = [0 0 1 0; 0 0 0 1; -ks/m1 0 0 0; ks/m2 -ks/m2 0 0];
23 Bc = [0 0 1/m1 0]';
24 Cc = [0 1 0 0];
25 Ec = [0 0; 0 0; 1/m1 0; 0 1/m2];
26
27 %% Dimensions
28 n = size(Ac,1); % number of states
29 m = size(Bc,2); % number of inputs
30 p = size(Cc,1); % number of outputs
31
32 %% User-configurable disturbance flag
33 % The input disturbance exist
34 flag = 1; % Set to 0 for zero disturbance, 1 for nonzero disturbance
35
36 if flag == 0
37     d1 = 0;
38     d2 = 0;
39 else
40     d1 = 0.25;

```

```

41     d2 = 0.7;
42 end
43
44 d = [d1;d2];
45
46 %% Initial state
47 Ts = 0.1;
48
49 sysc = ss(Ac, [Bc Ec], Cc, zeros(1,size(Bc,2)));
50 sysd = c2d(sysc,Ts);
51
52 %% Initiate from origins
53 flag_1 = 1; % Set to 0 for zero disturbance, 1 for nonzero disturbance
54
55 if flag_1 == 0
56     x0 = [-1; 1; -0.2; 0.2];
57 else
58     x0 = [0; 0; 0; 0];
59 end
60
61 x = zeros(n, Tsim+1); % state trajectory
62 x(:,1) = x0; % initial state
63
64 u = zeros(1, Tsim); % input trajectory
65 y = zeros(1, Tsim); % output trajectory
66
67 Q = Cc'*Cc;
68 R = 1;
69
70 r_vec = zeros(1, Tsim);
71 for k = 1:Tsim
72     if k <= 30
73         r_vec(k) = 0.5;
74     elseif k <= 70
75         r_vec(k) = 1.0;
76     else
77         r_vec(k) = 1.5;
78     end
79 end
80
81 A = sysd.A;
82 B = sysd.B(:,1);
83 C = sysd.C;
84 D = sysd.D;
85 E = sysd.B(:,2:3);
86
87 [P, ~] = idare(A, B, Q, R, [], []);
88
89 %% Check the Reachability, Observability, and R and Q Definiteness
90 check_ABQR(A,B,Q,R);
91
92 %% Constraints
93 % Input constraints
94 P_u = [1; -1]; % 2x1
95 q_u = [1; 1]; % 2x1
96
97 % State constraints
98 P_x = [0 0 1 0;
99         0 0 -1 0;

```

```

100     0 0 0 1;
101     0 0 0 -1];
102 q_x = [0.5; 0.5; 0.5; 0.5];
103
104 for k = 1:Tsim
105
106
107     %% Step 1: Solve for x_ss, u_ss
108     Tss = [eye(n)-A, -B; C, 0];
109     r = r_vec(k);
110     RHS = [E*d; r];
111     ans_ss = Tss \ RHS;
112     x_ss = ans_ss(1:n);
113     u_ss = ans_ss(n+1:end);
114
115     %% Step 2: Compute prediction and cost matrices
116     [F,G] = predict_mats(A,B,N);
117     cond_F = cond(F);
118     cond_G = cond(G);
119     fprintf('Condition number of F: %f\n', cond_F);
120     fprintf('Condition number of G: %f\n', cond_G);
121
122     [H,L,M] = cost_mats(F,G,Q,R,P);
123
124     %% Step 3: Shift constraints
125     qx_bar = q_x - P_x * x_ss;
126     qu_bar = q_u - P_u * u_ss;
127     [Pc, qc, Sc] = constraint_mats(F,G,P_u,qu_bar,P_x,qx_bar,P_x,qx_bar);
128
129     %% Step 4: Solve QP for deviation input sequence du
130     x_dev = x(:,k) - x_ss; % deviation state
131     f = L * x_dev;
132
133     % Solve QP
134     options = optimoptions('quadprog', 'Display', 'none');
135     [z, ~, exitflag, output] = quadprog(H, f, Pc, qc + Sc*x_dev, [], [], [], [],
136     [], options);
137
138     if exitflag ~= 1
139         disp("quadprog failed at step k = " + k);
140         disp(output.message);
141         z = zeros(N*m,1); % fallback zero input sequence
142     end
143
144     du = z(1); % first control increment
145     u(:,k) = du + u_ss; % actual control input
146
147     %% Step 5: Simulate system
148     x(:,k+1) = A * x(:,k) + B * u(:,k) + E* d;
149     y(:,k) = C * x(:,k);
150 end
151
152 %% Analyze Steady-State Error and Overshoot
153
154 % Reference value (used in tracking)
155 ref = r; % Should match the one used during simulation
156
157 % Steady-state error = |final y - reference|
158 steady_state_error = abs(y(end) - ref);

```



```

158
159 % Overshoot = max(y) - reference (if it exceeds reference)
160 overshoot = max(y) - ref;
161 if overshoot < 0
162     overshoot = 0; % no overshoot if response is always below reference
163 end
164
165 % Display results
166 fprintf('\n--- Performance Metrics ---\n');
167 fprintf('Reference value: %.4f\n', ref);
168 fprintf('Final output y(T): %.4f\n', y(end));
169 fprintf('Steady-state error: %.4f\n', steady_state_error);
170 fprintf('Overshoot: %.4f\n', overshoot);
171
172
173 %% [--- Plotting code unchanged, continues here ---]
174
175 % Plotting
176 time = 0:Tsim-1;
177
178 figure
179
180 %% Subplot 1: Output y(k)
181
182 subplot(2,2,1)
183 hold on
184 stairs(0:Tsim-1, u, 'b-', 'LineWidth', 1);
185 % yline(ulimit, 'k:', 'LineWidth', 1); % Upper input constraint
186 % yline(-ulimit, 'k:', 'LineWidth', 1); % Lower input constraint
187 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
188 ylabel('$u(k)$', 'Interpreter', 'latex');
189 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
190     'Location', 'northeast', 'Interpreter', 'latex');
191 title('Input $u(k)$', 'Interpreter', 'latex');
192 grid on
193 xlim([0 Tsim]);
194 ylim([-1.2*ulimit 1.2*ulimit]);
195
196
197 %% Subplot 2: Input u(k)
198 subplot(2,2,2)
199 hold on
200 plot(0:Tsim-1, y, 'b-', 'LineWidth', 1);
201 yline(ylimit, 'k:', 'LineWidth', 1); % Output constraint
202 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
203 ylabel('$y(k)$', 'Interpreter', 'latex');
204 title('Output $y(k)$', 'Interpreter', 'latex');
205 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
206     'Location', 'northeast', 'Interpreter', 'latex');
207 grid on
208 xlim([0 Tsim]);
209 ylim([0 2 * ylimit]);
210
211
212 %% Subplot 3: State x3(k)
213 subplot(2,2,3)
214 hold on

```

```

215 plot(0:Tsim, x(3,:), 'b-', 'LineWidth', 1);
216 yline(x3limit, 'k:', 'LineWidth', 1); % Upper state constraint
217 yline(-x3limit, 'k:', 'LineWidth', 1); % Lower state constraint
218 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
219 ylabel('$x_3(k)$', 'Interpreter', 'latex');
220 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
221         'Location', 'northeast', 'Interpreter', 'latex');
222 title('State $x_3(k)$', 'Interpreter', 'latex');
223 grid on
224 xlim([0 Tsim]);
225 ylim([-1.2*x3limit 1.2*x3limit]);
226
227 %% Subplot 4: State x4(k)
228 subplot(2,2,4)
229 hold on
230 plot(0:Tsim, x(4,:), 'b-', 'LineWidth', 1);
231 yline(x4limit, 'k:', 'LineWidth', 1); % Upper state constraint
232 yline(-x4limit, 'k:', 'LineWidth', 1); % Lower state constraint
233 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
234 ylabel('$x_4(k)$', 'Interpreter', 'latex');
235 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Constrained\ Problem', ...
236         'Location', 'northeast', 'Interpreter', 'latex');
237 title('State $x_4(k)$', 'Interpreter', 'latex');
238 grid on
239 xlim([0 Tsim]);
240 ylim([-1.2*x4limit 1.2*x4limit]);
241
242 %% Phase Poytrait Analysis for constrain case
243 figure;
244
245 %% First subplot: x3 vs x4
246 subplot(1,1,1);
247 grid on;
248 hold on;
249
250 % Unconstrained: blue line with blue circles
251 plot(x(3,:), x(4,:), 'bo-', 'LineWidth', 0.5, 'MarkerSize', 4, 'MarkerFaceColor',
252        'b');
253
254 % Initial and final points
255 plot(x(3,1), x(4,1), 'gs', 'MarkerSize', 7, 'LineWidth', 1.2, 'DisplayName', '$x_0$ (Unconstrained)', 'MarkerFaceColor', 'w');
256 plot(x(3,end), x(4,end), 'r*', 'MarkerSize', 8, 'LineWidth', 1.2, 'DisplayName', '$x_f$ (Unconstrained)');
257
258 xline(-x3limit, 'k:', 'LineWidth', 1); % Vertical line at x1 = -10
259 xline(x3limit, 'k:', 'LineWidth', 1); % Vertical line at x1 = +10
260 yline(-x4limit, 'k:', 'LineWidth', 1); % Horizontal line at x2 = -2
261 yline(x4limit, 'k:', 'LineWidth', 1); % Horizontal line at x2 = +2
262 legend('$\mathbf{u}^{\{0\}}_{\{N\}}(0) = -H^{\{-1\}} L x(k)$ \textendash\ Input\ and\ State\ Constrained\ Problem', ...
263         '$x_0$ (Input+State Constrained)', '$x_f$ (Input+State Constrained)', ...
264         'Location', 'northeast', 'Interpreter', 'latex');
265 title('Phase Portrait between $x_3$ vs $x_4$', 'Interpreter', 'latex');
266 xlim([-0.6 0.6]);
267 ylim([-0.6 0.6]);

```

```

268 xlabel('$x_3$', 'Interpreter', 'latex');
269 ylabel('$x_4$', 'Interpreter', 'latex');
270
271 figure;
272 plot(0:Tsim-1, r_vec, 'k--', 'LineWidth', 1.2); hold on;
273 plot(0:Tsim-1, y, 'b-', 'LineWidth', 1.2);
274
275 xlabel('$\mathrm{Time\ step,\ } k$', 'Interpreter', 'latex');
276 ylabel('$y(k)\ \mathrm{and\ reference\ } r(k)$', 'Interpreter', 'latex');
277 legend({'$r(k)$ \textendash Reference', '$y(k)$ \textendash Output'}, ...
278         'Interpreter', 'latex', 'Location', 'best');
279 title('Tracking Response with Varying Reference $r(k)$', 'Interpreter', 'latex');
280 grid on;
281
282 % === PERFORMANCE METRICS ===
283 metrics = struct();
284
285 compute_overshoot = @(y) max(0, (max(y) - y(end)) / max(abs(y(end)), 1e-6)); %
286     Avoid divide-by-zero
287
288 labels = {'Constrained'};
289 xs_all = {x};
290 us_all = {u};
291 ys_all = {y};
292
293 for idx = 1:length(labels)
294     x_data = xs_all{idx};
295     u_data = us_all{idx};
296     y_data = ys_all{idx};
297
298     % === COST ===
299     J = 0;
300     for k = 1:Tsim
301         xk = x_data(:,k);
302         uk = u_data(:,k);
303         J = J + xk'*Q*xk + uk'*R*uk;
304     end
305
306     % === CONTROL EFFORT ===
307     Ueffort = sum(u_data.^2, 'all');
308
309     % === SETTLING TIME ===
310     final_x = x_data(:,end);
311     tol = 0.02 * abs(final_x + 1e-5);
312     settle_idx = find(all(abs(x_data - final_x) < tol, 1), 1);
313     settling_time = NaN;
314     if ~isempty(settle_idx)
315         settling_time = (settle_idx - 1) * Ts;
316     end
317
318     % === OVERSHOOT ===
319     overshoot = compute_overshoot(y_data(1,:));
320
321     scenario = labels{idx};
322     metrics.(scenario).Cost = J;
323     metrics.(scenario).ControlEffort = Ueffort;
324     metrics.(scenario).SettlingTime = settling_time;
325     metrics.(scenario).Overshoot = overshoot;
326 end

```

```

326
327 % === Display summary ===
328 summary_table = table(...
329     metrics.Constrained.Cost, ...
330     metrics.Constrained.ControlEffort, ...
331     metrics.Constrained.SettlingTime, ...
332     100 * metrics.Constrained.Overshoot, ...
333     'VariableNames', {'Cost', 'ControlEffort', 'SettlingTime', 'OvershootPercent'
334         }, ...
335     'RowNames', {'Constrained'});
336 disp('Performance Metrics Summary:');
337 disp(summary_table);

```

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