

Stochastic Adaptive Control Assignment 3

Kristoffer Erbo Kjær - s203829

May 6, 2025



Question 1.1

To design the stationary ordinary Kalman filter the associated Discrete algebraic Riccati equation needs to be solved. For convenience in the Matlab implementation, the Riccati equation of the stationary covariance of the predictive Kalman filter is solved Eq. (1).

$$P_{\infty}^{pp} = AP_{\infty}^{pp}A^T + R_1 - AP_{\infty}^{pp}C^T(CP_{\infty}^{pp}C^T + R_2)^{-1}CP_{\infty}^{pp}A^T \quad (1)$$

This is done since the equations' structure is very similar to the matlab solver: $[\mathbf{X}, \mathbf{K}, \mathbf{L}] = \text{idare}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}, \mathbf{S}, \mathbf{E})$ Eq. (2).

$$A^T X A - E^T X E - (A^T X B + S)(B^T X B + R)^{-1}(A^T X B + S)^T + Q = 0 \quad (2)$$

Calling the **idare** function as follows: $[\mathbf{P}_{\infty}^{pp}, \sim, \sim] = \text{idare}(\mathbf{A}', \mathbf{C}', \mathbf{R1}, \mathbf{R2})$ solves the Discrete algebraic Riccati equation of the stationary covariance of the predictive Kalman filter.

The stationary covariance of the ordinary Kalman filter can then be found using Eq. (3).

$$(P_{\infty}^o)^{-1} = (P_{\infty}^p)^{-1} + C^T R_2^{-1} C \quad (3)$$

The Kalman filter gain can then be calculated:

$$K_f = P_{\infty}^o \cdot C'(C \cdot P_{\infty}^o \cdot C' + R_2)^{-1} = \begin{bmatrix} 0.4738 & 0.0008 \\ 0.0008 & 0.4738 \\ 0.1811 & 0.0034 \\ 0.0034 & 0.1811 \end{bmatrix} \quad (4)$$

Implementing a simulation with the decruded noise and no control input, the following simulation can then be obtained:

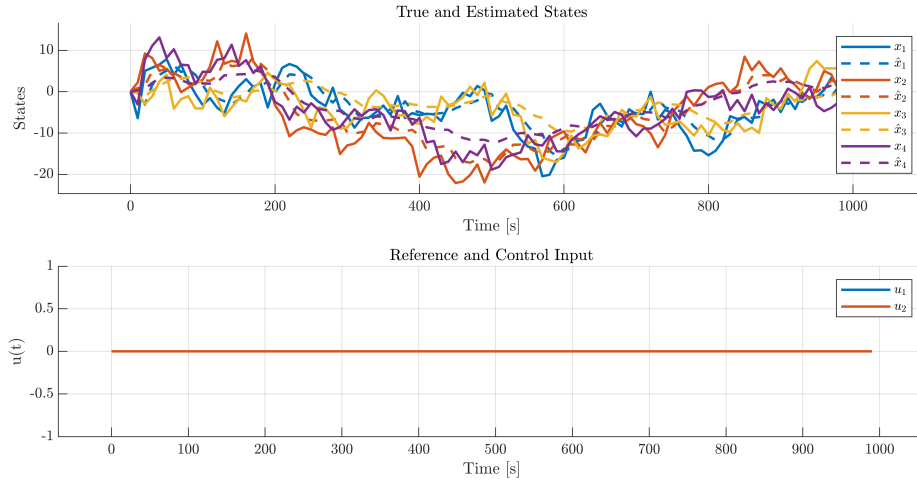


Figure 1: The uncontrolled system with a Kalman filter as a state estimator.

From the simulation it is clear that the state estimates are following the actual states to some degree, but that especially the unmeasured states are deviating from the actual value of the state. This is, however, to be expected since the system is in steady state with zero input. The expected value is therefore dependent on the noise term which has a mean value of 0. Any deviations from this value are caused by the noise and it is therefore reasonable that the Kalman filter state estimates are not more precise.

Question 1.2

In this section, the aim is to implement a general predictive controller following the control law described in Eq. (5).

$$U_N = -(\Gamma_{yu}^T Q_y \Gamma_{yu} + Q_u)^{-1} \Gamma_{yu}^T Q_y (\Phi_{yx} \mathbb{E}[x_0] - W_N) \quad (5)$$

Where Q_y and Q_u are weighting matrices from the cost function describing the cost of reference deviation and utilization of control input respectively.

For the prediction the following matrices in Eq. (6) are used.

$$\Gamma_{yu}^N = \begin{bmatrix} D & & & \\ CB & D & & \\ \vdots & \ddots & \ddots & \\ CA^{N-1}B & \dots & CB & D \end{bmatrix}, \Phi_{yx}^N = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^N \end{bmatrix} \quad (6)$$

The input prediction is updated for each iteration and the only control input utilized is therefore described in Eq. (7).

$$u_t = [I \quad 0 \quad \dots \quad 0] U_N \quad (7)$$

Implementing this together with the Kalman filter (Fig. 2), it is shown that the state estimation is satisfactory for the control problem. It can be seen that state/output 1 and 2 follow their respective references. The only significant deviations are when the setpoint is changed. The deviation is both before and after the reference change due to the predictive element of the controller with even weights of all steps. The prediction horizon is limited to 10 steps / 100 s but similar performance can be shown with an increased horizon.

The deviation from the reference could be minimized by lowering the values of Q_u . The values chosen (**diag**([100,100])) are to limit the control effort while keeping reasonable response time of the thermal system. The settling time is around 50 s which is around one deg per second. In addition to limiting the control utilization the temperature of state 3 and 4 is also kept in a reasonable range when using a lower control effort. This increases the chance of the linearization being an acceptable estimation.

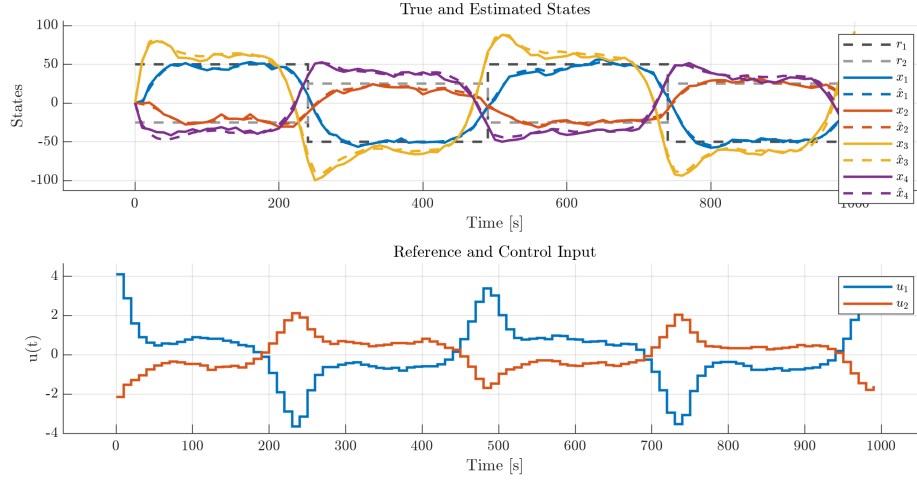


Figure 2: The system with a General predictive controller implimented and Kalman filter state estimation.

Question 1.3

Since the system has two independent measurements, it is possible to estimate two constant independent disturbances. Since a disturbance for each state cannot be estimated, a choice of where to model the disturbances must be made. Due to the actuator placement and the strong link between state 1 and 3, as well as state 2 and 4, the disturbances are modeled as entering state 3 and 4. This ensures that any offset in the actuators will effectively be counteracted by an integral action, ensuring that the system reaches steady state when control is implemented.

Ideally, a constant disturbance would be modeled as $\kappa = 1$, but as I understand it, classmates had problems with no solution to the Riccati equation existing and a value close to is therefore chosen. I can reproduce this issue when adding more than two disturbances, but since the system then becomes not observable, I would argue that adding more disturbances becomes meaningless. Especially since the system is a two input two output system and two disturbances should be enough for a future implimentation of a controller.

$\sigma_d^2 = 4$ is iteratively tuned until a satisfactory disturbance estimation is reached which will be descussed below.

$$A_{\text{aug}} = \begin{bmatrix} 0.7901 & 0.0238 & 0.1667 & 0.0081 & 0 & 0 \\ 0.0238 & 0.7901 & 0.0081 & 0.1667 & 0 & 0 \\ 0.1667 & 0.0081 & 0.7236 & 0.0014 & 1.0000 & 0 \\ 0.0081 & 0.1667 & 0.0014 & 0.7236 & 0 & 1.0000 \\ 0 & 0 & 0 & 0 & 0.9990 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9990 \end{bmatrix}, \quad (8)$$

$$B_{\text{aug}} = \begin{bmatrix} 1.4459 & 0.0622 \\ 0.0622 & 1.4459 \\ 13.3615 & 0.0077 \\ 0.0077 & 13.3615 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, C_{\text{aug}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, D_{\text{aug}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (9)$$

$$R_{1_{\text{aug}}} = \begin{bmatrix} 8.0209 & 0.2260 & 1.5334 & 0.0812 & 0 & 0 \\ 0.2260 & 8.0209 & 0.0812 & 1.5334 & 0 & 0 \\ 1.5334 & 0.0812 & 7.4082 & 0.0172 & 0 & 0 \\ 0.0812 & 1.5334 & 0.0172 & 7.4082 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.0000 \end{bmatrix} \quad (10)$$

To test the the Kalman filter disturbance estimation, the unforced system is simulated in Fig. 3. It is clear here that the disturbance estimation is working since the disturbance is tracked and the state estimation is still close to the actual values of the states. It is, however, clear that when the disturbance is changing rapidly, the state estimation of especially state 3 and 4 becomes more unreliable until the disturbance estimate is converged again. This could be done faster but since a constant disturbance is assumed a slower response is prioritized to minimize noise in the steady state of the disturbance.

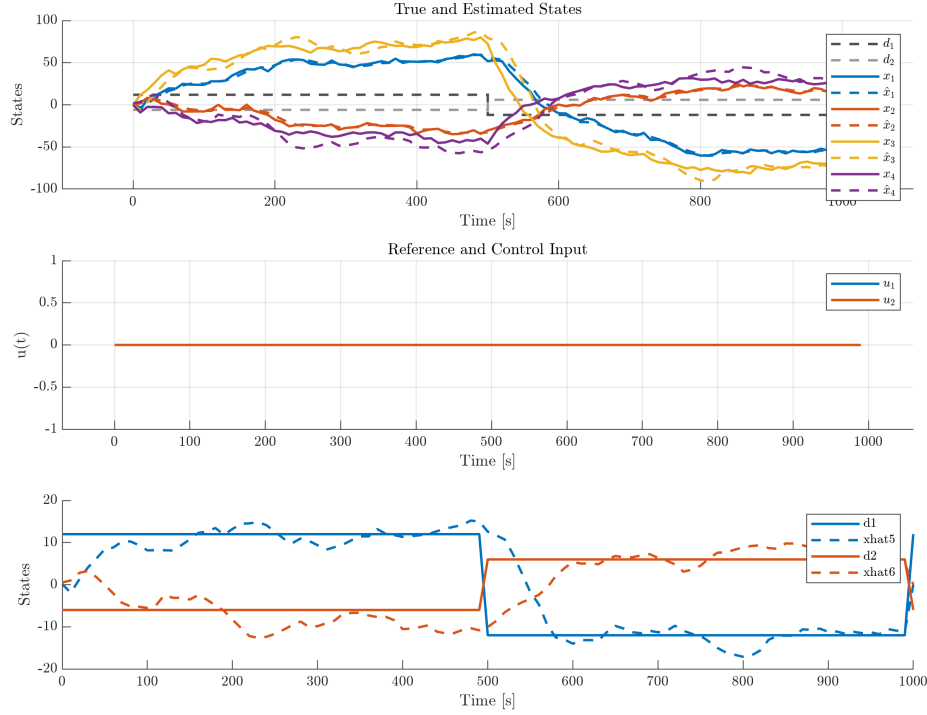


Figure 3: Disturbance estimation with no control input.

Question 1.4

Now combining the disturbance estimation Kalman filter and the general predictive controller, the results seen in Fig. 4 are achieved. Since the same seed as in question 1.3 is used, the system is subject to the same noise. Observing Fig. 3 and Fig. 4 it appears that the disturbance estimate is the same as with no control. This is reasonable since the Kalman filter accounts for the input; the only unexplained factors are therefore the noise and the disturbances which are the same in both cases (For confirmation it was checked that changing the seed results in different results).

The estimation performance is also similar to the one seen in question 1.3. After stepping the disturbance, the estimation of states 3 and 4 is worse. In this particular case, it appears to affect the step in state 1 around 500 s since the disturbance is lowered in the same instance. This results in the settling time being lower since the controller expects state 3 to be higher than it actually is.

Overall, the disturbance estimation and controller appear to function well when combined since the reference is followed and the disturbance is successfully estimated.

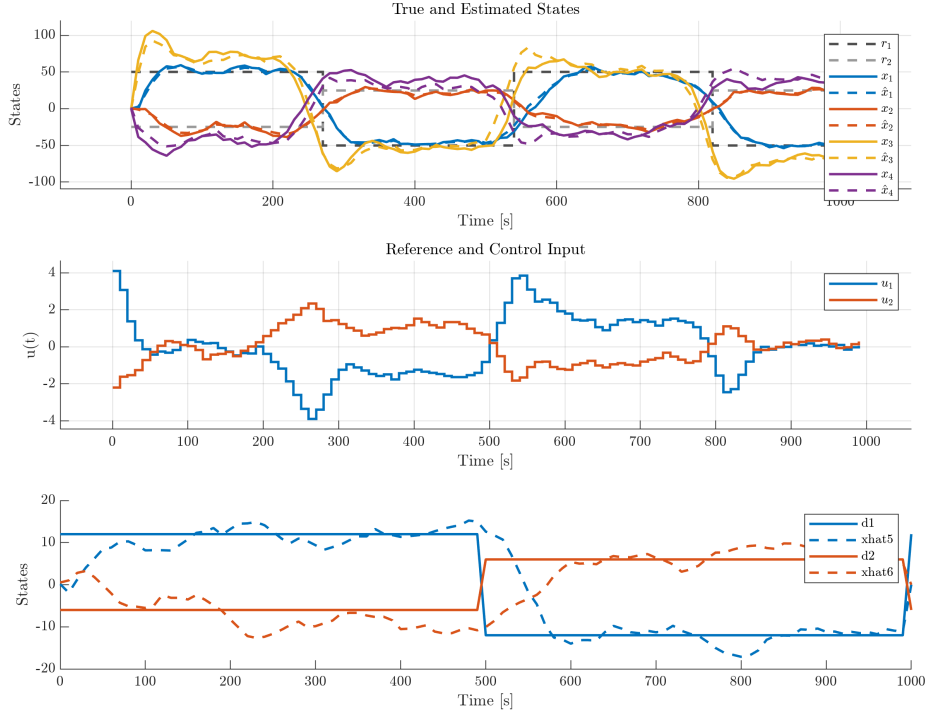


Figure 4: General predictive control with disturbance estimation with two disturbances.

To further test the performance of the closed loop system, a disturbance is now added to all states while the disturbance estimation still only assumes the disturbances enter through states 3 and 4. This can be seen in Fig. 5. The disturbance estimation is now significantly off for states 3 and 4, which are the ones modeled since the disturbances in states 1 and 2 dominate the effect on the output. The state estimates of states 3 and 4 are therefore also significantly off. However, since the disturbance estimation acts as integrators on the reference error, it can be observed that the reference tracking is still satisfactory when there are no rapid changes in the disturbances.

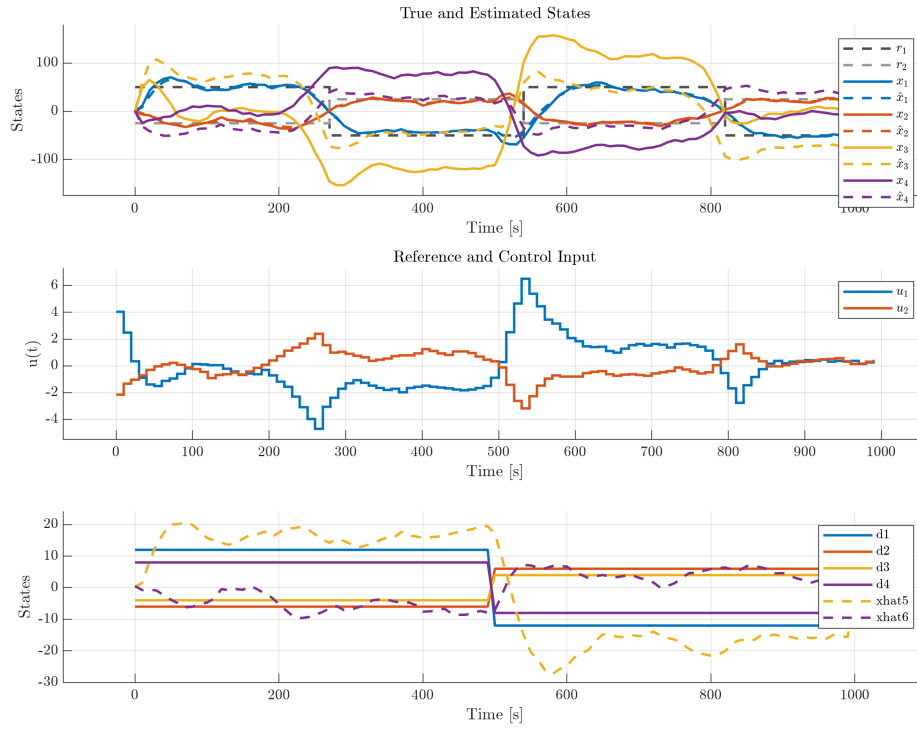


Figure 5: General predictive control with disturbance estimation with four disturbances.

Appendix

A init

```
1 clear
2 rng(203829)
3 addpath('functions')
4 set(groot, 'defaultTextInterpreter', 'latex');
5 set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
6 set(groot, 'defaultLegendInterpreter', 'latex');
7 plotPos = [100 100 800, 400];
8 plotPos2 = [100 100 800, 600];
```

B Question 1.1

```
1 Question 1.1
2 %% --- system definition ---
3 Ts = 10;    % sample time [s]
4
5 A = [0.7901, 0.0238, 0.1667, 0.0081;
6      0.0238, 0.7901, 0.0081, 0.1667;
7      0.1667, 0.0081, 0.7236, 0.0014;
8      0.0081, 0.1667, 0.0014, 0.7236];
9
10 B = [ 1.4459,  0.0622;
11      0.0622,  1.4459;
12      13.3615,  0.0077;
13      0.0077, 13.3615];
14
15 C = [1, 0, 0, 0;
16      0, 1, 0, 0];
17
18 % cost / noise-covariance matrices (as given)
19 R1= [8.0209, 0.2260, 1.5334, 0.0812;
20      0.2260, 8.0209, 0.0812, 1.5334;
21      1.5334, 0.0812, 7.4082, 0.0172;
22      0.0812, 1.5334, 0.0172, 7.4082];
23
24 R2 = eye(2);    % = R2
25
26 %% --- LQR controller design ---
27 ctrl0n = 0;
28 [K,~,~] = dlqr(A,B,diag([1,1,0,0]),R2*250);
29 dcG = (C*(eye(4) - (A-B*K))^( -1) * B)^-1;
```

```

30
31 %% --- Kalman filter design (steady-state) ---
32 % L is the state-estimator gain:  $\hat{x}_{k+1} = A\hat{x}_k + B*u_k + L*(y_k - C*\hat{x}_k)$ 
33 % Solve the discrete Riccati equation
34 [P,~,~] = dare(A', C', R1, R2);
35 [Pp,~,~] = idare(A', C', R1, R2);
36
37 Po = inv(inv(Pp) + C'*inv(R2)*C);
38
39 % Compute stationary Kalman gain
40 %  $K_f = P * C' / (C * P * C' + R2)$ ;
41 Kf = Po * C' / (C * Po * C' + R2);
42
43
44 %% --- simulation setup ---
45 Tfinal = 1000; % total simulation time [s]
46 N = Tfinal / Ts; % number of steps
47 x = zeros(4, N+1); % true state
48 xhat = zeros(4, N+1); % estimated state
49 y = zeros(2, N); % measurements
50 yhat = zeros(2, N); % predicted outputs
51 u = zeros(2, N); % control inputs
52
53 rng(0); % for reproducible noise
54
55 T = N; % Periode in sampels
56 ref1 = 50*sqrt(2*pi)/T*(1:N),50);
57 ref2 = zeros(1,N);
58 ref = [ref1;ref2];
59
60
61 for k = 1:N
62     % Simulate measurement
63     v = mvnrnd(zeros(size(R2,1),1), R2)';
64     y(:,k) = C*x(:,k) + v;
65
66     % Controller (if any)
67     if ctrlOn
68         u(:,k) = -K*xhat(:,k) + dcG * ref(:,k);
69     end
70     % True system evolution
71     w = mvnrnd(zeros(size(R1,1),1), R1)';
72     x(:,k+1) = A*x(:,k) + B*u(:,k) + w;
73
74     % Stationary Kalman filter update
75     innov = y(:,k) - C*xhat(:,k);

```

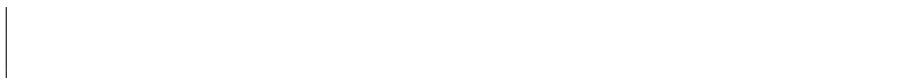
```

76     xhat(:,k + 1) = A*xhat(:,k) + B*u(:,k) + Kf*innov;
77     yhat(:,k + 1) = C*xhat(:,k);
78 end
79
80 time = 0:Ts:Tfinal;
81 colors = lines(4);
82 %% --- plotting ---
83 fig = figure();
84 fig.Position = plotPos;
85 tiledlayout(2,1, 'Padding', 'compact', 'TileSpacing', 'compact');
86
87 nexttile;
88 names = {'$x_1$', '$\hat{x}_1$', '$x_2$', '$\hat{x}_2$', '$x_3$', '$\hat{x}_3$', '$x_4$', '$\hat{x}_4$'};
89 if ctrlOn
90     stairs(time(1:end-1), ref(1,:), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
91     names = {'$r_1$', '$x_1$', '$\hat{x}_1$', '$x_2$', '$\hat{x}_2$', '$x_3$', '$\hat{x}_3$', '$x_4$', '$\hat{x}_4$'};
92 end
93 hold on
94 for i = 1:4
95     plot(time, x(i,:), '-', 'Color', colors(i,:), 'LineWidth', 1.5);
96     plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
97 end
98 xlabel('Time [s]');
99 ylabel('States');
100 legend(names, ...
101         'Location','northeast');
102 title('True and Estimated States');
103 ylim padded;
104 xlim padded;
105 grid on;
106 hold off
107
108 % Plot control input
109 nexttile;
110 hold on
111 stairs(time(1:end-1), u(:,:)', '-', 'LineWidth', 1.5);
112 xlabel('Time [s]');
113 ylabel('u(t)');
114 legend('$u_1$', '$u_2$');
115 title('Reference and Control Input');
116 ylim padded;
117 xlim padded;
118 grid on;
119 hold off
120 filePath = fullfile("output", 'question_1_1.png');
121 exportgraphics(fig, filePath, 'Resolution', 600);

```

122

123



C Question 1.2

```

1 Question 1.2
2 %% 1) System definition
3 Ts = 10; % sample time [s]
4 Tfinal = 1000; % total simulation time [s]
5 Nsim = Tfinal / Ts; % number of steps
6
7 A = [0.7901, 0.0238, 0.1667, 0.0081;
8      0.0238, 0.7901, 0.0081, 0.1667;
9      0.1667, 0.0081, 0.7236, 0.0014;
10     0.0081, 0.1667, 0.0014, 0.7236];
11 B = [ 1.4459, 0.0622;
12      0.0622, 1.4459;
13      13.3615, 0.0077;
14      0.0077, 13.3615];
15 C = [1, 0, 0, 0;
16      0, 1, 0, 0];
17
18 D = zeros(2,2);
19
20 [nx, nu] = size(B);
21 ny = size(C,1);
22
23 R1 = [8.0209, 0.2260, 1.5334, 0.0812;
24      0.2260, 8.0209, 0.0812, 1.5334;
25      1.5334, 0.0812, 7.4082, 0.0172;
26      0.0812, 1.5334, 0.0172, 7.4082]; % process-noise cov
27 R2 = eye(ny); % meas-noise cov
28 [Lf, P, ~] = dlqe(A, eye(nx), C, R1, R2);
29
30 % [P,~,~] = dare(A', C', Q, R);
31 [Pp,~,~] = idare(A', C', R1, R2);
32
33 Po = inv(inv(Pp) + C'*inv(R2)*C);
34
35 % Compute stationary Kalman gain
36 % Kf = P * C' / (C * P * C' + R2);
37 Kf = Po * C' / (C * Po * C' + R2);
38
39 %% 3) GPC tuning parameters
40 Np = 10; % prediction horizon
41 % Np = Nsim
42 Qy = kron(eye(Np), 1*eye(ny)); % output-tracking weight
43 Ru = 100 * eye(nu*Np); % input move weight

```

```

44
45 %% 4) Build prediction matrices
46 Phi = zeros(ny*Np, nx);
47 Gamma_y = zeros(ny*Np, nu*Np);
48
49 for i = 1:Np
50     Phi((i-1)*ny+1:i*ny, :) = C * A^(i-1);
51     for j = 1:i
52         idx_row = (i-1)*ny+1 : i*ny;
53         idx_col = (j-1)*nu+1 : j*nu;
54
55         if i == j
56             Gamma_y(idx_row, idx_col) = D; % Diagonal: D
57         else
58             Gamma_y(idx_row, idx_col) = C * A^(i-j) * B; % Lower-triangular: CA^(i-j)B
59         end
60     end
61 end
62 %% 5) Reference signal
63 ref1 = 50 * square(2*pi*(2/(Nsim))*(1:Nsim + Np), 50);
64 ref2 = -0.5*ref1; %zeros(1, Nsim + Np);
65 ref = [ref1; ref2];
66
67 %% 6) Preallocate
68 x = zeros(nx, Nsim+1); % true state
69 xhat = zeros(nx, Nsim+1); % estimated state
70 u = zeros(nu, Nsim);
71 y = zeros(ny, Nsim);
72 xhat_pred = xhat(:,1);
73
74 rng(0); % for reproducibility
75
76 %% 7) Main simulation loop
77
78 for k = 1:Nsim
79     %--- simulate measurement y(k) with noise
80     y(:,k) = C*x(:,k) + mvnrnd(zeros(ny,1), R2)';
81
82     %--- Kalman correct
83     innov = y(:,k) - C*xhat_pred;
84     xhat(:,k) = xhat_pred + Kf * innov;
85
86     %--- GPC: form QP linear term
87     r_win = reshape(ref(:, k:k+Np-1), [], 1); % stacked reference
88
89     %--- apply control law

```

```

90     U_N = -inv(Gamma_y' * Qy * Gamma_y + Ru) * Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);
91
92     %--- apply only first move
93     % u(:, k) = U_N(1:nu);
94     u(:, k) = [eye(nu), zeros(nu, nu*Np - nu)] * U_N;
95
96     %--- simulate true system with process noise
97     x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx, 1), R1)';
98
99     %--- Kalman predict
100    xhat_pred = A*xhat(:,k) + B*u(:, k);
101
102    end
103
104    %% 8) Plotting
105    time = 0:Ts:Tfinal;
106    colors = lines(4);
107
108    fig = figure();
109    fig.Position = plotPos;
110    tiledlayout(2,1, 'Padding', 'compact', 'TileSpacing', 'compact');
111
112    nexttile;
113    hold on
114    stairs(time(1:end-1), ref(1,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
115    stairs(time(1:end-1), ref(2,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.6 0.6 0.6]);
116    names = {'$r_1$', '$r_2$', '$x_1$', '$\hat{x}_1$', '$x_2$', '$\hat{x}_2$', '$x_3$', '$\hat{x}_3$'};
117
118
119    for i = 1:4
120        plot(time, x(i,:), '-', 'Color', colors(i,:), 'LineWidth', 1.5);
121        plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
122    end
123    xlabel('Time [s]');
124    ylabel('States');
125    legend(names, ...
126           'Location','northeast');
127    title('True and Estimated States');
128    ylim padded;
129    xlim padded;
130    grid on;
131    hold off
132
133    % Plot control input
134    nexttile;
135    hold on

```

```

136 stairs(time(1:end-1), u(:, :)', '-', 'LineWidth', 1.5);
137 xlabel('Time [s]');
138 ylabel('u(t)');
139 legend('$u_1$', '$u_2$');
140 title('Reference and Control Input');
141 ylim padded;
142 xlim padded;
143 grid on;
144 hold off
145 filePath = fullfile('output', 'question_1_2.png');
146 exportgraphics(fig, filePath, 'Resolution', 300);
147

```


D Question 1.3

```
1 Question 1.3
2
3 %% 1) System definition
4 Ts = 10; % sample time [s]
5 Tfinal = 1000; % total simulation time [s]
6 Nsim = Tfinal / Ts; % number of steps
7
8 A = [0.7901, 0.0238, 0.1667, 0.0081;
9      0.0238, 0.7901, 0.0081, 0.1667;
10     0.1667, 0.0081, 0.7236, 0.0014;
11     0.0081, 0.1667, 0.0014, 0.7236];
12 B = [ 1.4459, 0.0622;
13      0.0622, 1.4459;
14      13.3615, 0.0077;
15      0.0077, 13.3615];
16 C = [1, 0, 0, 0;
17      0, 1, 0, 0];
18
19
20
21 [nx, nu] = size(B);
22 ny = size(C,1);
23
24 %% 2) Kalman filter design (steady-state gain)
25 R1 = [8.0209, 0.2260, 1.5334, 0.0812;
26      0.2260, 8.0209, 0.0812, 1.5334;
27      1.5334, 0.0812, 7.4082, 0.0172;
28      0.0812, 1.5334, 0.0172, 7.4082]; % process-noise cov
29 R2 = eye(ny); % meas-noise cov
30
31
32 kappa = [0.999,0.999];
33 % kappa = [0.99999,0.99999,0.99999,0.99999];
34 sigma_d = 2*[1,1];
35 A_aug = [A,[zeros(2,2);eye(2,2)];zeros(2,4),diag(kappa)];
36 % A_aug = [A,eye(4,2);zeros(2,4),diag(kappa)]
37 B_aug = [B;zeros(2,2)];
38 C_aug = [C, zeros(2,2)];
39 D_aug = D;
40 R1_aug = [R1,zeros(4,2);zeros(2,4),diag(sigma_d.^2)];
41 rank(observ(A_aug,C_aug))
42
43
```

```

44 [nx, nu] = size(B_aug);
45 ny      = size(C_aug,1);
46
47
48 [ Kf, ~, ~ ] = dlqe(A_aug, eye(nx), C_aug, R1_aug, R2);
49 [Pp,~,~] = idare(A_aug', C_aug', R1_aug, R2);
50
51 Po = inv(inv(Pp) + C_aug'*inv(R2)*C_aug);
52
53 % Compute stationary Kalman gain
54 % Kf = P * C' / (C * P * C' + R2);
55 Kf = Po * C_aug' / (C_aug * Po * C_aug' + R2);
56
57
58 %% 3) GPC tuning parameters
59 Np = 10; % prediction horizon
60 Qy = kron(eye(Np), eye(ny)); % output-tracking weight
61 Ru = 100 * eye(nu*Np); % input move weight
62
63 %% 4) Build prediction matrices
64 Phi = zeros(ny*Np, nx);
65 Gamma_y = zeros(ny*Np, nu*Np);
66
67 for i = 1:Np
68     Phi((i-1)*ny+1:i*ny, :) = C_aug * A_aug^(i-1);
69     for j = 1:i
70         idx_row = (i-1)*ny+1 : i*ny;
71         idx_col = (j-1)*nu+1 : j*nu;
72
73         if i == j
74             Gamma_y(idx_row, idx_col) = D_aug; % Diagonal: D
75         else
76             Gamma_y(idx_row, idx_col) = C_aug * A_aug^(i-j) * B_aug; % Lower-triangular: C
77         end
78     end
79 end
80
81 % Precompute Hessian for QP
82 H = Gamma_y' * Qy * Gamma_y + Ru;
83 opts = optimoptions('quadprog','Display','off');
84
85 %% 5) Reference signal (±10 square wave on output 1)
86 ref1 = 50 * square(2*pi*(2/(Nsim+ Np))*(1:Nsim + Np), 50);
87 ref2 = -0.5*ref1; %zeros(1, Nsim + Np);
88 ref = [ref1; ref2];
89

```

```

90
91 %% 6) Preallocate
92 x      = zeros(nx-2, Nsim+1); % true state
93 xhat    = zeros(nx, Nsim+1); % estimated state
94 d      = [12* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
95           -6* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
96           ]; % estimated state
97 % d = 100*[ones(1,Nsim+1);
98 %          -ones(1,Nsim+1)];
99
100 u      = zeros(nu, Nsim);
101 y      = zeros(ny, Nsim);
102 xhat_pred = xhat(:,1);
103 Xpred   = zeros(nx, Np+1, Nsim); % store Np-step predictions
104 rng(0); % for reproducibility
105
106 %% 7) Main simulation loop
107 for k = 1:Nsim
108     %--- simulate measurement y(k) with noise
109     y(:,k) = C*x(:,k) + mvnrnd(zeros(ny,1), R2)';
110
111     %--- Kalman correct
112     innov      = y(:,k) - C_aug*xhat_pred;
113     xhat(:,k)  = xhat_pred + Kf * innov;
114
115     %--- GPC: form QP linear term
116     r_win = reshape(ref(:, k:k+Np-1), [], 1); % stacked reference
117     f = Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);
118
119     %--- apply control law
120     U_N = -inv(Gamma_y' * Qy * Gamma_y + Ru) * Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);
121
122     %--- apply only first move
123     % u(:, k) = U_N(1:nu);
124     u(:, k) = [eye(nu), zeros(nu,nu*Np - nu)] * U_N;
125     u(:, k) = [0;0];
126
127
128     % Compute Np-step ahead state predictions
129     Xpred(:,1,k) = xhat(:,k);
130     for j = 1:Np
131         Uj = U_N((j-1)*nu+1:j*nu);
132         Xpred(:,j+1,k) = A_aug*Xpred(:,j,k) + B_aug*Uj;
133     end
134
135

```

```

136     %--- simulate true system with process noise
137     %  $x(:, k+1) = A * x(:, k) + B * u(:, k) + \text{mvnrnd}(\text{zeros}(nx-2, 1), Qk\_f)' + [\text{eye}(2,2); \text{zeros}(2, nx-2)] * \text{zeros}(nx-2, 1)$ 
138      $x(:, k+1) = A * x(:, k) + B * u(:, k) + \text{mvnrnd}(\text{zeros}(nx-2, 1), R1)' + [\text{zeros}(2,2); \text{eye}(2, nx-2)] * \text{zeros}(nx-2, 1)$ 
139
140     %--- Kalman predict
141      $\hat{x}_{pred} = A_{aug} * \hat{x}(:, k) + B_{aug} * u(:, k)$ ;
142 end
143
144 %% 8) Plotting
145 time = 0:Ts:Tfinal;
146 colors = lines(4);
147
148 fig = figure();
149 fig.Position = plotPos2;
150 tiledlayout(3,1, 'Padding', 'compact', 'TileSpacing', 'compact');
151
152 nexttile;
153 hold on
154 stairs(time(1:end), d(1,1:end), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
155 stairs(time(1:end), d(2,1:end), '--', 'LineWidth', 1.5, 'Color', [0.6 0.6 0.6]);
156 names = {'$d_1$', '$d_2$', '$x_1$', '$\hat{x}_1$', '$x_2$', '$\hat{x}_2$', '$x_3$', '$\hat{x}_3$'};
157
158
159 for i = 1:4
160     plot(time, x(i,:), '-', 'Color', colors(i,:), 'LineWidth', 1.5);
161     plot(time,  $\hat{x}(i,:)$ , '--', 'Color', colors(i,:), 'LineWidth', 1.5);
162 end
163 xlabel('Time [s]');
164 ylabel('States');
165 legend(names, ...
166         'Location','northeast');
167 title('True and Estimated States');
168 ylim padded;
169 xlim padded;
170 grid on;
171 hold off
172
173 % Plot control input
174 nexttile;
175 hold on
176 stairs(time(1:end-1), u(:, :)', '-', 'LineWidth', 1.5);
177 xlabel('Time [s]');
178 ylabel('u(t)');
179 legend('$u_1$', '$u_2$');
180 title('Reference and Control Input');
181 ylim padded;

```

```

182 xlim padded;
183 grid on;
184 hold off
185 % filePath = fullfile("output", 'question_1_4.png');
186 % exportgraphics(fig, filePath, 'Resolution', 300);
187
188 % fig = figure();
189 % fig.Position = plotPos;
190 % tiledlayout(1,1, 'Padding', 'compact', 'TileSpacing', 'compact');
191 nexttile;
192 hold on
193 for i = 5:6
194     plot(time, d(i-4,:), '-', 'Color', colors(i-4,:), 'LineWidth', 1.5);
195     plot(time, xhat(i,:), '--', 'Color', colors(i-4,:), 'LineWidth', 1.5);
196 end
197 hold off
198 xlabel('Time [s]');
199 ylabel('States');
200 legend([strcat("d",string(1:2));strcat("xhat",string(5:6))], ...
201         'Interpreter','latex','Location','northeast');
202 filePath = fullfile("output", 'question_1_3.png');
203 exportgraphics(fig, filePath, 'Resolution', 300);
204
205

```

E Question 1.4

```

1 Question 1.4
2 %% 1) System definition
3 Ts = 10; % sample time [s]
4 Tfinal = 1000; % total simulation time [s]
5 Nsim = Tfinal / Ts; % number of steps
6
7 A = [0.7901, 0.0238, 0.1667, 0.0081;
8      0.0238, 0.7901, 0.0081, 0.1667;
9      0.1667, 0.0081, 0.7236, 0.0014;
10     0.0081, 0.1667, 0.0014, 0.7236];
11 B = [ 1.4459, 0.0622;
12      0.0622, 1.4459;
13      13.3615, 0.0077;
14      0.0077, 13.3615];
15 C = [1, 0, 0, 0;
16      0, 1, 0, 0];
17 D= zeros(2,2);
18
19 [nx, nu] = size(B);
20 ny = size(C,1);
21
22 %% 2) Kalman filter design (steady-state gain)
23 R1 = [8.0209, 0.2260, 1.5334, 0.0812;
24      0.2260, 8.0209, 0.0812, 1.5334;
25      1.5334, 0.0812, 7.4082, 0.0172;
26      0.0812, 1.5334, 0.0172, 7.4082]; % process-noise cov
27 R2 = eye(ny); % meas-noise cov
28
29
30 kappa = [0.999,0.999];
31 % kappa = [0.99999,0.99999,0.99999,0.99999];
32 sigma_d = 2*[1,1];
33 A_aug = [A,[zeros(2,2);eye(2,2)];zeros(2,4),diag(kappa)];
34 % A_aug = [A,eye(4,2);zeros(2,4),diag(kappa)]
35 B_aug = [B;zeros(2,2)];
36 C_aug = [C, zeros(2,2)];
37 D_aug = D;
38 R1_aug = [R1,zeros(4,2);zeros(2,4),diag(sigma_d.^2)];
39 rank(observ(A_aug,C_aug))
40
41
42 [nx, nu] = size(B_aug);
43 ny = size(C_aug,1);

```

```

44
45
46 [ Kf, ~, ~ ] = dlqe(A_aug, eye(nx), C_aug, R1_aug, R2);
47 [Pp,~,~] = idare(A_aug', C_aug', R1_aug, R2);
48
49 Po = inv(inv(Pp) + C_aug'*inv(R2)*C_aug);
50
51 % Compute stationary Kalman gain
52 % Kf = P * C' / (C * P * C' + R2);
53 Kf = Po * C_aug' / (C_aug * Po * C_aug' + R2);
54
55
56 %% 3) GPC tuning parameters
57 Np = 10; % prediction horizon
58 Qy = kron(eye(Np), eye(ny)); % output-tracking weight
59 Ru = 100 * eye(nu*Np); % input move weight
60
61 %% 4) Build prediction matrices
62 Phi = zeros(ny*Np, nx);
63 Gamma_y = zeros(ny*Np, nu*Np);
64
65 for i = 1:Np
66     Phi((i-1)*ny+1:i*ny, :) = C_aug * A_aug^(i-1);
67     for j = 1:i
68         idx_row = (i-1)*ny+1 : i*ny;
69         idx_col = (j-1)*nu+1 : j*nu;
70
71         if i == j
72             Gamma_y(idx_row, idx_col) = D_aug; % Diagonal: D
73         else
74             Gamma_y(idx_row, idx_col) = C_aug * A_aug^(i-j) * B_aug; % Lower-triangular: C
75         end
76     end
77 end
78
79 % Precompute Hessian for QP
80 H = Gamma_y' * Qy * Gamma_y + Ru;
81 opts = optimoptions('quadprog','Display','off');
82
83 %% 5) Reference signal (±10 square wave on output 1)
84 ref1 = 50 * square(2*pi*(2/(Nsim+ Np))*(1:Nsim + Np), 50);
85 ref2 = -0.5*ref1; %zeros(1, Nsim + Np);
86 ref = [ref1; ref2];
87
88
89 %% 6) Preallocate

```

```

90 x      = zeros(nx-2, Nsim+1); % true state
91 xhat   = zeros(nx, Nsim+1); % estimated state
92 d1     = [12* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
93          -6* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
94          ]; % estimated state
95
96 d2     = [12* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
97          -6* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
98          -4* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50);
99          8* square(2*pi*(2/(2*Nsim+ 1))*(1:Nsim + 1), 50)];
100 % d = 100*[ones(1,Nsim+1);
101 %         -ones(1,Nsim+1)];
102
103 u      = zeros(nu, Nsim);
104 y      = zeros(ny, Nsim);
105 xhat_pred = xhat(:,1);
106 Xpred   = zeros(nx, Np+1, Nsim); % store Np-step predictions
107 rng(0); % for reproducibility
108
109 %% 7) Main simulation loop
110 for k = 1:Nsim
111     %--- simulate measurement y(k) with noise
112     y(:,k) = C*x(:,k) + mvnrnd(zeros(ny,1), R2)';
113
114     %--- Kalman correct
115     innov      = y(:,k) - C_aug*xhat_pred;
116     xhat(:,k)  = xhat_pred + Kf * innov;
117
118     %--- GPC: form QP linear term
119     r_win = reshape(ref(:, k:k+Np-1), [], 1); % stacked reference
120     f = Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);
121
122     %--- apply control law
123     U_N = -inv(Gamma_y' * Qy * Gamma_y + Ru) * Gamma_y' * Qy * (Phi * xhat(:, k) - r_win);
124
125     %--- apply only first move
126     % u(:, k) = U_N(1:nu);
127     u(:, k) = [eye(nu), zeros(nu,nu*Np - nu)] * U_N;
128     % u(:, k) = [0;0];
129
130
131     % Compute Np-step ahead state predictions
132     Xpred(:,1,k) = xhat(:,k);
133     for j = 1:Np
134         Uj = U_N((j-1)*nu+1:j*nu);
135         Xpred(:,j+1,k) = A_aug*Xpred(:,j,k) + B_aug*Uj;

```



```

136     end
137
138
139     %--- simulate true system with process noise
140     x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx-2, 1), R1)' + [zeros(2,2);eye(2)];
141     % x(:, k+1) = A * x(:, k) + B * u(:, k) + mvnrnd(zeros(nx-2, 1), R1)' + eye(4)*d2(:,k);
142
143     %--- Kalman predict
144     xhat_pred = A_aug*xhat(:,k) + B_aug*u(:, k);
145 end
146
147 %% 8) Plotting
148 time = 0:Ts:Tfinal;
149 colors = lines(4);
150
151 fig = figure();
152 fig.Position = plotPos2;
153 tiledlayout(3,1, 'Padding', 'compact', 'TileSpacing', 'compact');
154
155 nexttile;
156 hold on
157 stairs(time(1:end-1), ref(1,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.3 0.3 0.3]);
158 stairs(time(1:end-1), ref(2,1:end-Np), '--', 'LineWidth', 1.5, 'Color', [0.6 0.6 0.6]);
159 names = {'$r_1$', '$r_2$', '$x_1$', '$\hat{x}_1$', '$x_2$', '$\hat{x}_2$', '$x_3$', '$\hat{x}_3$'};
160
161
162 for i = 1:4
163     plot(time, x(i,:), '-', 'Color', colors(i,:), 'LineWidth', 1.5);
164     plot(time, xhat(i,:), '--', 'Color', colors(i,:), 'LineWidth', 1.5);
165 end
166 xlabel('Time [s]');
167 ylabel('States');
168 legend(names, ...
169         'Location','northeast');
170 title('True and Estimated States');
171 ylim padded;
172 xlim padded;
173 grid on;
174 hold off
175
176 % Plot control input
177 nexttile;
178 hold on
179 stairs(time(1:end-1), u(:,:)', '-', 'LineWidth', 1.5);
180 xlabel('Time [s]');
181 ylabel('u(t)');

```

```

182 legend('$u_1$', '$u_2$');
183 title('Reference and Control Input');
184 ylim padded;
185 xlim padded;
186 grid on;
187 hold off
188 % filePath = fullfile('output', 'question_1_4.png');
189 % exportgraphics(fig, filePath, 'Resolution', 300);
190 % fig = figure();
191 % fig.Position = plotPos;
192 % tiledlayout(1,1, 'Padding', 'compact', 'TileSpacing', 'compact');
193 nexttile;
194 hold on
195 for i = 5:6
196     plot(time, d1(i-4,:), '-', 'Color', colors(i-4,:), 'LineWidth', 1.5);
197     plot(time, xhat(i,:), '--', 'Color', colors(i-4,:), 'LineWidth', 1.5);
198 end
199 hold off
200 xlabel('Time [s]');
201 ylabel('States');
202 ylim padded;
203 xlim padded;
204 grid on
205 legend([strcat("d",string(1:2));strcat("xhat",string(5:6))], ...
206         'Interpreter','latex','Location','northeast');
207 filePath = fullfile('output', 'question_1_4.png');
208 exportgraphics(fig, filePath, 'Resolution', 300);
209

```