



HACETTEPE UNIVERSITY - COMPUTER ENGINEERING

Advanced Robust Control MMÜ 749

Homework-3

Student Name
Student ID:

Ayça KULA
202237285

Spring 2022

1 Problem 1:

The design specifications for this problem is:

- Unit DC gain from r to y
- Max %16 overshoot
- $\zeta = 0.5$ and $\omega_n = 0.5$. Therefore, the desired pole locations are

$$s = \zeta\omega_n \pm \omega_n\sqrt{1 - \zeta^2}j = 0.4 \pm 0.69282j$$

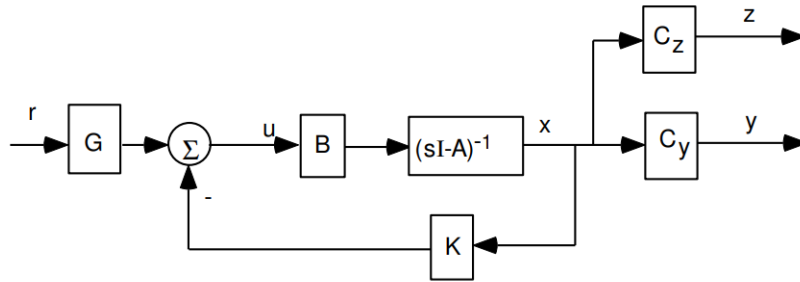


Figure 1: State Feedback with Two Sets of Outputs

And consider the transfer function as:

$$P_y(s) = \frac{1}{(0.5s + 1)(s^2 + 2(0.01)(0.8)s + (0.8)^2)} \quad (1)$$

1.1 Part A: LQR-1

For each part, you can examine the codes in Appendix and you can see each formulation for every question.

1.1.1 State space representation for the TF:

State-space is found by using the command "tf2ss" in MATLAB.

$$A = \begin{bmatrix} -2.0160 & -0.6720 & -1.2800 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad (2)$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (3)$$

$$C = \begin{bmatrix} 0 & 0 & 2 \end{bmatrix} \quad (4)$$

$$D = \begin{bmatrix} 0 \end{bmatrix} \quad (5)$$

1.1.2 Determine Q and R:

The MATLAB function `lqr` allows to use R and Q , which will balance the relative importance of the control effort (u) and error (deviation from 0), respectively, in the cost function that you are trying to optimize. The simplest case is to assume control weight has a form of $R = \rho R_0$ and state weighting matrix is $Q = C^T C$. In order to calculate the R matrix, I have used ρ values as given in $\rho = 0.6, 0.5, 0.4, 0.3, 0.2$. I have not chosen the ρ value as bigger than 1 due to overshoot problem and less than 0.1 due to cheap control situation. The LQR gains are shown in 1 for each ρ value. Since the lower ρ leads to cheap control and it is hard to control and big ρ leads to overshoot, I took the ρ value as 0.5.

1.1.3 Adding precompensation:

We need to compute what the steady-state value of the states should be, multiply that by the chosen gain K , and use a new value as our "reference" for computing the input. This can be done by adding a constant gain G after the reference [1]. We can find this G factor by employing the used-defined function `rscale.m` [2]. After adding a pre-compensator the step responses are obtained for each ρ value as seen in 2. For each ρ , pre-compensator design was made and shown in 1.

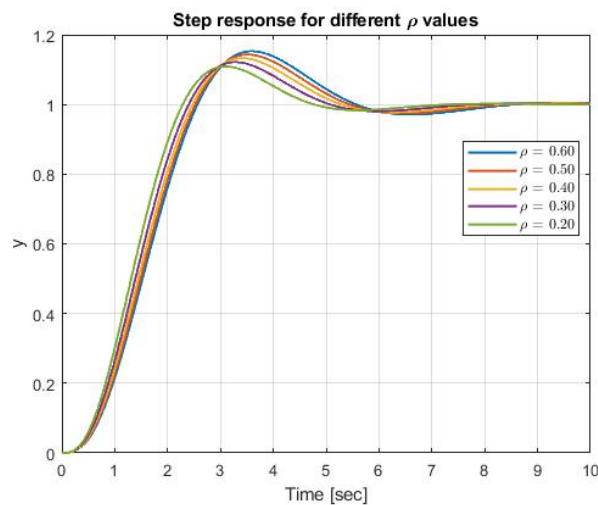


Figure 2: Step response for a system with pre-compensator

1.1.4 Draw SRL:

The symmetric root locus can be seen in figure 3. The SRL is obtained using the formulation in our lecture notes as 1. And, it can be gone through the the closed loop poles that satisfy the design requirements. This can be also seen in part-C figure 8.

```

%% Symmetric root locus plot
A_srl = [A zeros(size(A)); -C'*C -A'];
B_srl = [B; -C'*D];
C_srl = [D'*C B'];
D_srl = D'*D;
sys_srl = ss(A_srl,B_srl,C_srl,D_srl);
figure;
rlocus(sys_srl)
grid
legend('SRL')

```

Algorithm 1: Obtain SRL

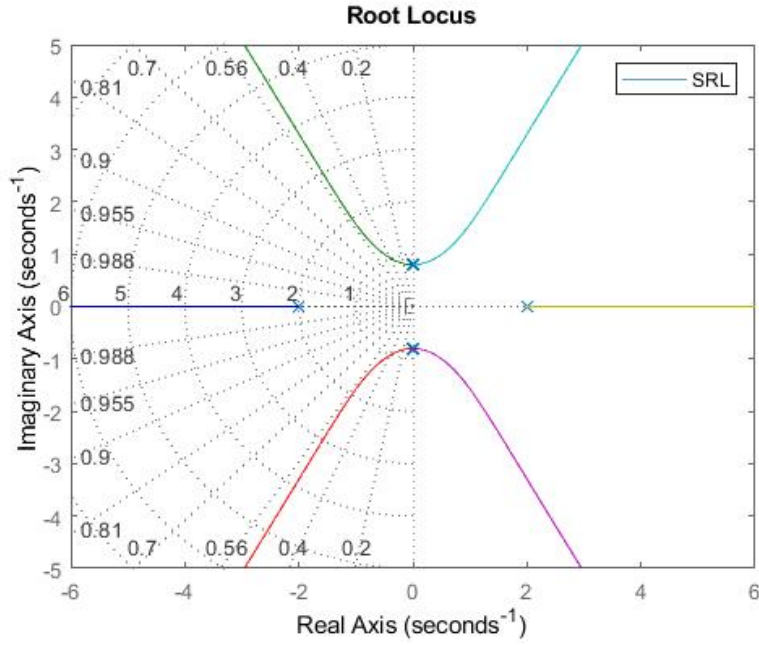


Figure 3: Symmetric Root Locus

1.1.5 Determine the loop gain L_{sf} :

The loop gain L_{sf} is found as:

$$L_{sf}(s) = K(sI - A)^{-1}B \quad (6)$$

And for each ρ value I have determined the L_{sf} which is shown in 1. It can be seen that increasing ρ value will increase the phase margin.

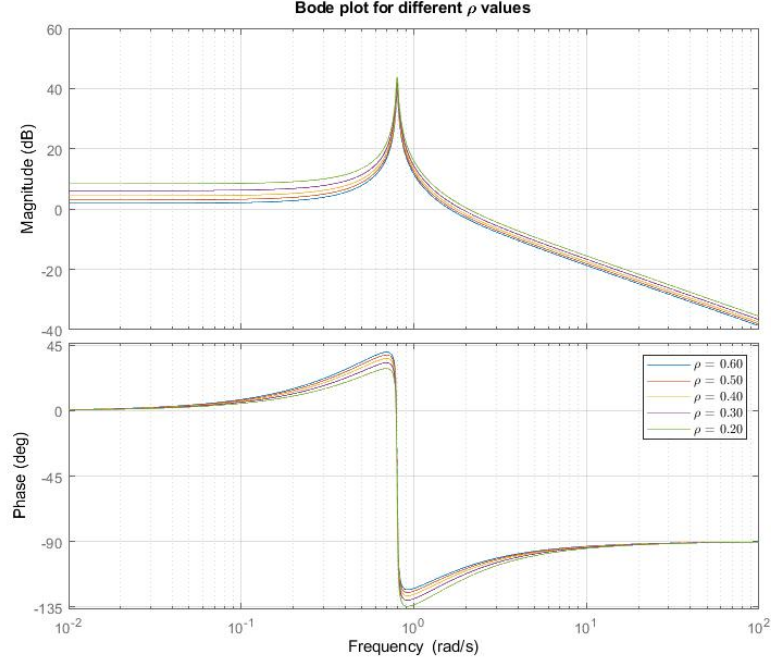


Figure 4: Bode plot

Table 1: The results for each ρ value

| rho | LQR Gain | Pre-Compensator | Gain Margin | Phase Margin |
|-----|------------------------|-----------------|-------------|--------------|
| 0.6 | 1.1616, 3.0166, 1.6019 | 1.4409 | - | 68.2063 |
| 0.5 | 1.2389, 3.2651, 1.8246 | 1.5523 | - | 67.7044 |
| 0.4 | 1.3375, 3.5908, 2.1315 | 1.7058 | - | 67.1349 |
| 0.3 | 1.4711, 4.0477, 2.5893 | 1.9347 | - | 66.4782 |
| 0.2 | 1.6721, 4.7688, 3.3717 | 2.3259 | - | 65.6797 |

1.2 Part B: LQR-2

1.2.1 State space representation for the TF:

State-space is found by using the command "tf2ss" in MATLAB.

$$A = \begin{bmatrix} -2.0160 & -0.6720 & -1.2800 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad (7)$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (8)$$

$$C = [2 \quad 1.6 \quad 1.28] \quad (9)$$

$$D = [0] \quad (10)$$

1.2.2 Determine Q and R:

The MATLAB function `lqr` allows to use R and Q , which will balance the relative importance of the control effort (u) and error (deviation from 0), respectively, in the cost function that you are trying to optimize. The simplest case is to assume control weight has a form of $R = \rho R_0$ and state weighting matrix is $Q = C^T C$. In order to calculate the R matrix, I have used ρ values as given in $\rho = 0.6, 0.5, 0.4, 0.3, 0.2$. I have not chosen the ρ value as bigger than 1 due to overshoot problem and less than 0.1 due to cheap control situation. The LQR gains are shown in 2 for each ρ value. Since the lower ρ leads to cheap control and it is hard to control and big ρ leads to overshoot, I took the ρ value as 0.5.

1.2.3 Adding precompensation:

We need to compute what the steady-state value of the states should be, multiply that by the chosen gain K , and use a new value as our "reference" for computing the input. This can be done by adding a constant gain G after the reference [1]. We can find this G factor by employing the used-defined function `rscale.m` [2]. After adding a pre-compensator the step responses are obtained for each ρ value as seen in 5. For each ρ , pre-compensator design was made and shown in 2.

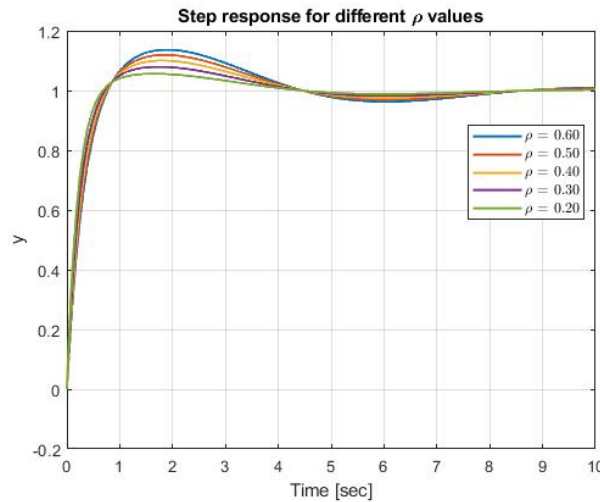


Figure 5: Step response for a system with pre-compensator

1.2.4 Draw SRL:

The symmetric root locus can be seen in figure 6. The SRL is obtained using the formulation in our lecture notes as 1. And, it can be gone through the the closed loop poles that satisfy the design requirements. This can be also seen in part-C figure 9. If we compare the part-A and part-B SRL, I can say that in part-A the poles go to infinity and in part-B it goes to the zero locations.

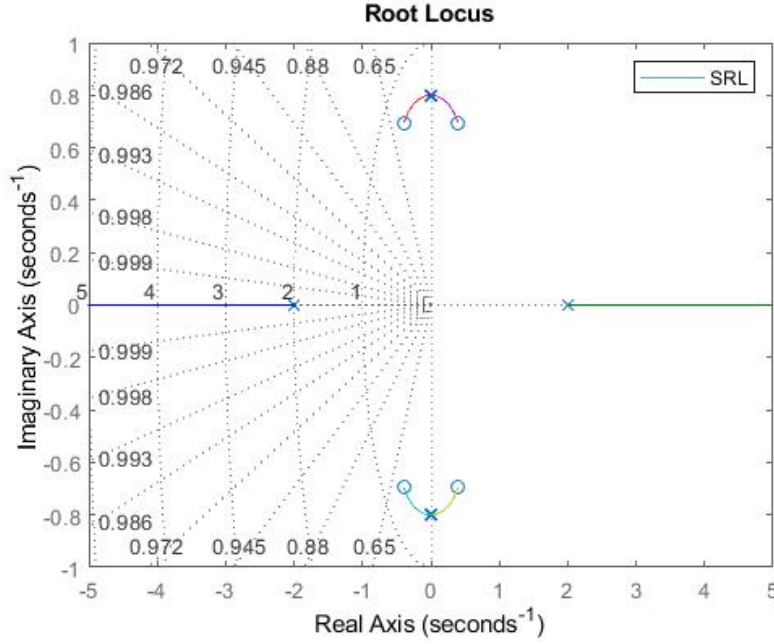


Figure 6: Symmetric Root Locus

Table 2: The results for each ρ value

| rho | LQR Gain | Pre-Compensator | Gain Margin | Phase Margin |
|-----|------------------------|-----------------|-------------|---------------------|
| 0.6 | 1.8177, 1.9833, 0.8102 | 1.6330 | - | -133.9272, 102.6751 |
| 0.5 | 2.0370, 2.1814, 0.9370 | 1.7321 | - | -141.2966, 103.8164 |
| 0.4 | 2.3388, 2.4501, 1.1147 | 1.8708 | - | -153.7585, 104.8312 |
| 0.3 | 2.7887, 2.8440, 1.3845 | 2.0817 | - | 105.2576 |
| 0.2 | 3.5584, 3.5048, 1.8553 | 2.4495 | - | 104.3113 |

1.2.5 Determine the loop gain L_{sf} :

The loop gain L_{sf} is found as:

$$L_{sf}(s) = K(sI - A)^{-1}B \quad (11)$$

And for each ρ value I have determined the L_{sf} which is shown in 2. It can be seen that decreasing ρ value will increase the phase margin (look at only positive values of PM). The bode plot is shown in figure 7.

And, different controller designs are shown in table 2 for decreasing ρ values.

1.3 Part C: LQG

1.3.1 Design a full order observer with the SRL approach

An observer is designed for PART-A and PART-B using the SRL approach.

1. PART A

The value for ρ has been chosen as 0.5. After choosing this value symmetric

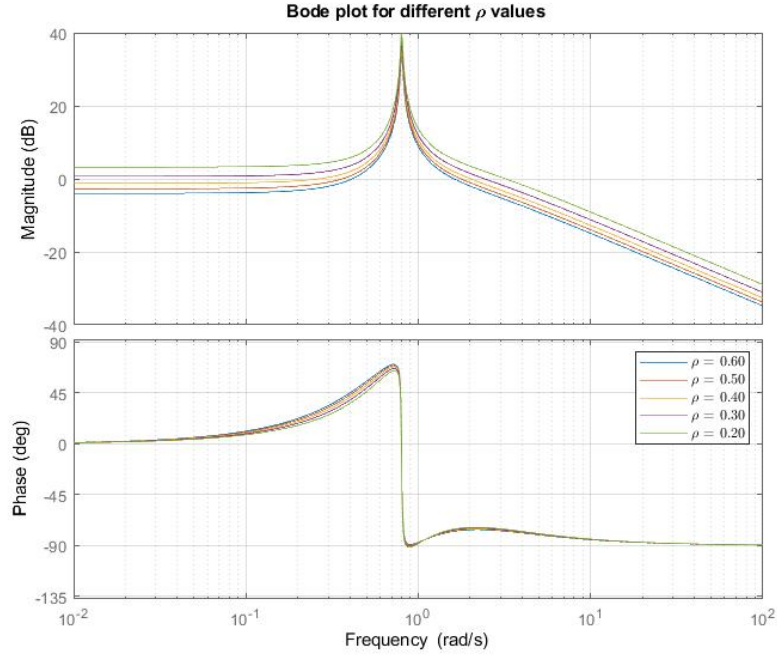


Figure 7: Bode plot

root locus has been plotted and the frequency has been seen as 1.23 rad/s. I tried to select the roots for estimation bandwidth as $BW = 1.23 \times 2.5 = 3.075 \text{ rad/s}$, however the gain was too big. And this can lead to problems. Therefore, I chose the bandwidth as 2.36 rad/s, which has the q value of 145. The corresponding estimator pole locations are:

$$pe = [-1.48 + 2.37i; -3.09; -1.48 - 2.37i] \quad (12)$$

And using the "place" command the corresponding observer gains are found as:

$$L = [1.8525; 4.0746; 2.0170] \quad (13)$$

The figure closed loop poles and observer poles can be seen in 8.

2. PART B

The value for ρ has been chosen as 0.5. After choosing this value symmetric root locus has been plotted and the frequency has been seen as 0.8 rad/s. I tried to select the roots for estimation bandwidth as $BW = 0.8 \times 2.5 = 2 \text{ rad/s}$, however 2 rad/s cannot be reached (it is out of bound). Therefore, I had to choose the bandwidth approximately similar. I have obtained my q value as 8.4. The corresponding estimator pole locations are:

$$pe = [-0.378 + 0.709i; -6.09; -0.378 - 0.709i] \quad (14)$$

And using the "place" command the corresponding observer gains are found

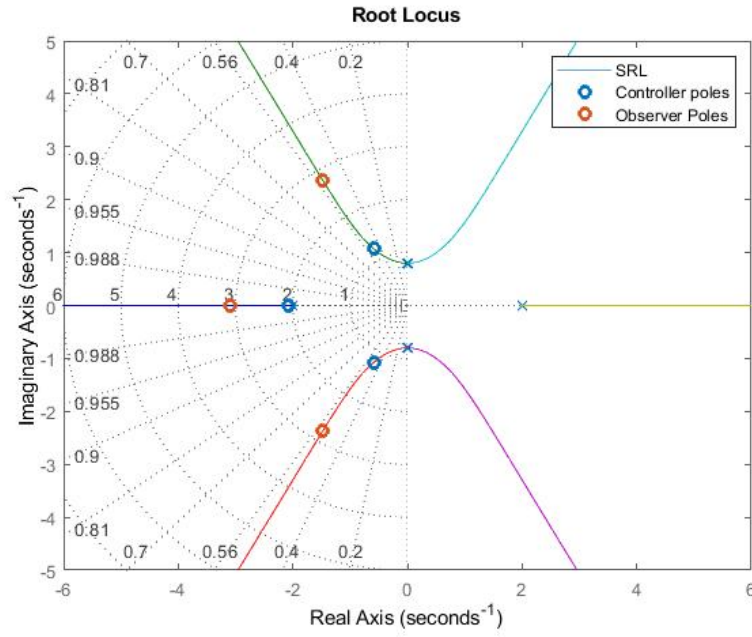


Figure 8: SRL plot for Observer for part-A

as:

$$L = [1.8525; 4.0746; 2.0170] \quad (15)$$

The figure closed loop poles and observer poles can be seen in 9.

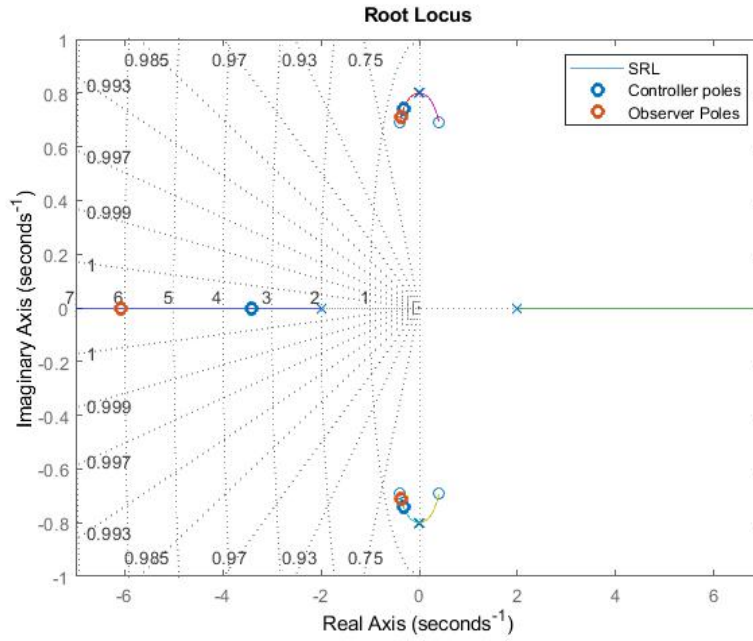


Figure 9: SRL plot for Observer for part-B

1.3.2 Test Performance in Simulink

The simulink model is obtained with the help of [3] and it is shown in figure 10. And, the simulation is made using 2.

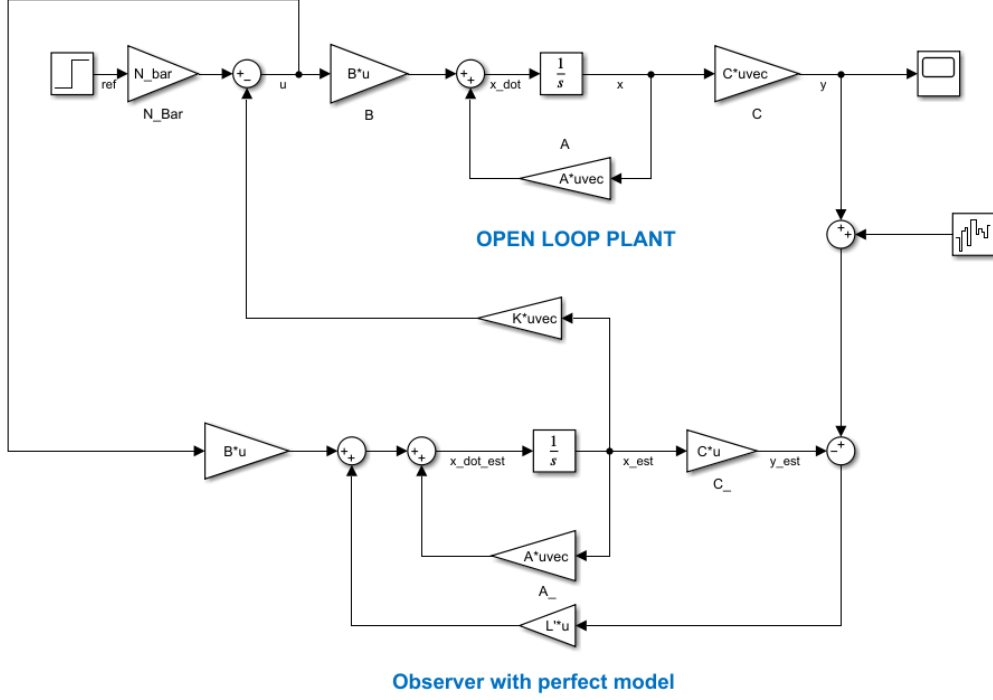


Figure 10: Simulink model

```
%% Simulink Part
N_bar = rscale(sys,K_Lqr);

noise_pow = [0.01, 0.1, 1, 10, 100];
figure;
for i = 1:length(noise_pow)
    set_param('LQR_model/Band-Limited White Noise','Cov',string(noise_pow(i)))
    y = sim('LQR_model')
    plot(y.yout{1}.Values.Time, y.yout{1}.Values.Data, 'linewidth', 1.25)
    hold on
end

legend('0.01', '0.1' , '1' , '10', '100' )

title('Increased Noise Power and Simulation Results')
```

Algorithm 2: Test Performance in Simulink

1.3.3 Discussions

- **How does the overall system perform as noise is gradually increased?**
The noise is gradually increased and the simulation results are shown in 11 for part-A and 12 for part-B. Since the q value is larger in Part A, this observer is more robust to noise compared to Part-b system.

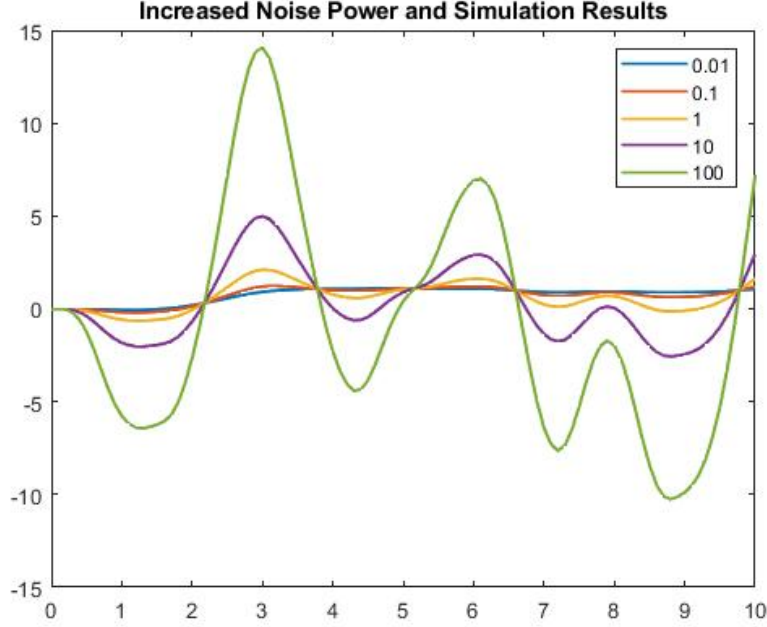


Figure 11: Increased noise and the simulation result for part-A

- **Are you able to satisfy performance specifications?** The step response of 1 and higher values of 1 is not appropriate. Therefore, systems are robust on smaller noise. However, Part-2 is more robust due to the big q value.
- **Do you feel the need to re-design the controller?** Yes, the design can be made for differently by choosing maybe different ρ values in the beginning such as 0.3 or 04.

1.3.4 Design an LQG/LTR controller

The LTR Controller is designed using [4]. LQG control inherently suffers from robustness issues. It is in fact less robust than LQR. While LQR have impressive Gain Margin(=infinity) and Phase margin(=60 degrees), there is no guaranteed stability margin for LQG control. The closed loop system can become unstable if our Kalman filter is not designed properly to take the non-linearities into account[3]. Again for the same, system parameters as before I have design an LTR Controller for part-A and part-B system as seen in figure 13 and 14 respectively. Also, the results are seen in table 3 and 4. And, the calculation can be seen in algorithm 3.

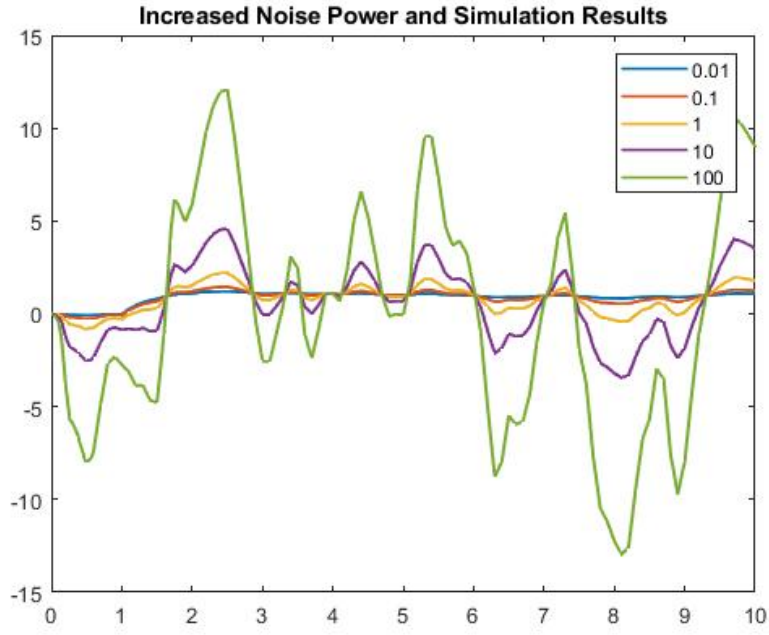


Figure 12: Increased noise and the simulation result for part-B

```

%% Design LTR
w=logspace(-1,3,1000);
rho_ltr = 0.5;
Q_ltr = rho_ltr*C'*C;
r = 1;
tf = K_Lqr*inv(s*eye(size(A))-A)*B;
[K] = lqr(A,B,Q_ltr,r);
sys1 = ss(A,B,K,0);
[magk1,phasgk1,w]=bode(sys1,w);

q = [1, 10, 100];
rv=1;
% MATLAB lqe
for i = 1:length(q)
    gam=q(i)*B;
    Q1=gam'*gam;
    LTR(i).L=lqe(A,gam,C,Q1,rv);

    % MATLAB bode MATLAB margin
    aa=A-B*K-(LTR(i).L)*C;
    bb=(LTR(i).L);
    cc=K;
    dd=0;
    sysk=ss(aa,bb,cc,dd);
    sysgk=series(sys,sysk);
    [magk,phsgk,w]=bode(sysgk,w);
    [gm,phm,wcg,wcp] = margin(magk,phsgk,w)
end

```

end

Table 3: LTR Controller Design for Part-A

| q | Observer Gain | Gain Margin | Phase Margin | Wcg | Wcp |
|----------|----------------------------|--------------------|---------------------|------------|------------|
| 1 | [-0.2408; 0.2308; 0.4804] | 3.5127 | 64.8465 | 0.0369 | 0.9668 |
| 10 | [47.3596; 24.1566; 4.9149] | 6.9243 | 61.9527 | 4.4065 | 1.1746 |
| 100 | [8602.9; 684; 26.2] | 28.7403 | 69.6066 | 19.4035 | 1.2401 |

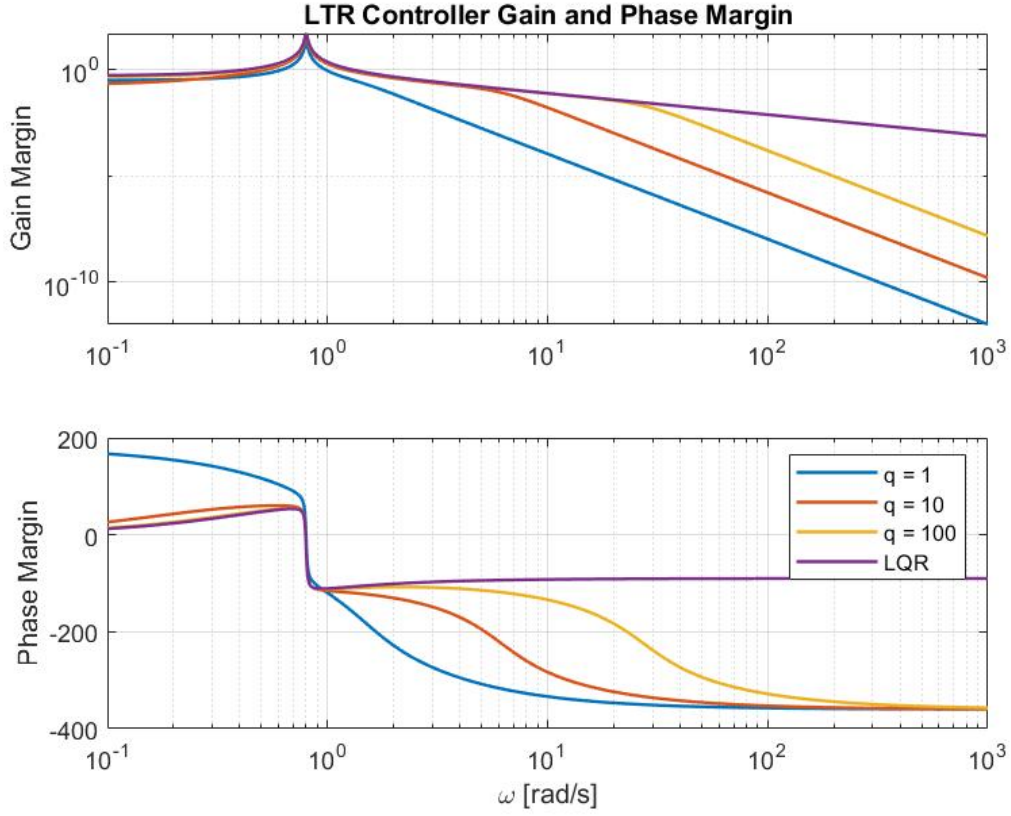


Figure 13: LTR Controller design for part A

Table 4: LTR Controller Design for Part-B

| q | Observer Gain | Gain Margin | Phase Margin | Wcg | Wcp |
|-----|---------------------------|-------------|--------------|-----|--------|
| 1 | [0.4334; 0.2603; 0.0251] | Inf | 84.3355 | NaN | 1.0219 |
| 10 | [98.9982; 0.4964; 0.0013] | Inf | 98.6645 | NaN | 1.3213 |
| 100 | [9999; 0.5; 0] | Inf | 99.0996 | NaN | 1.3270 |

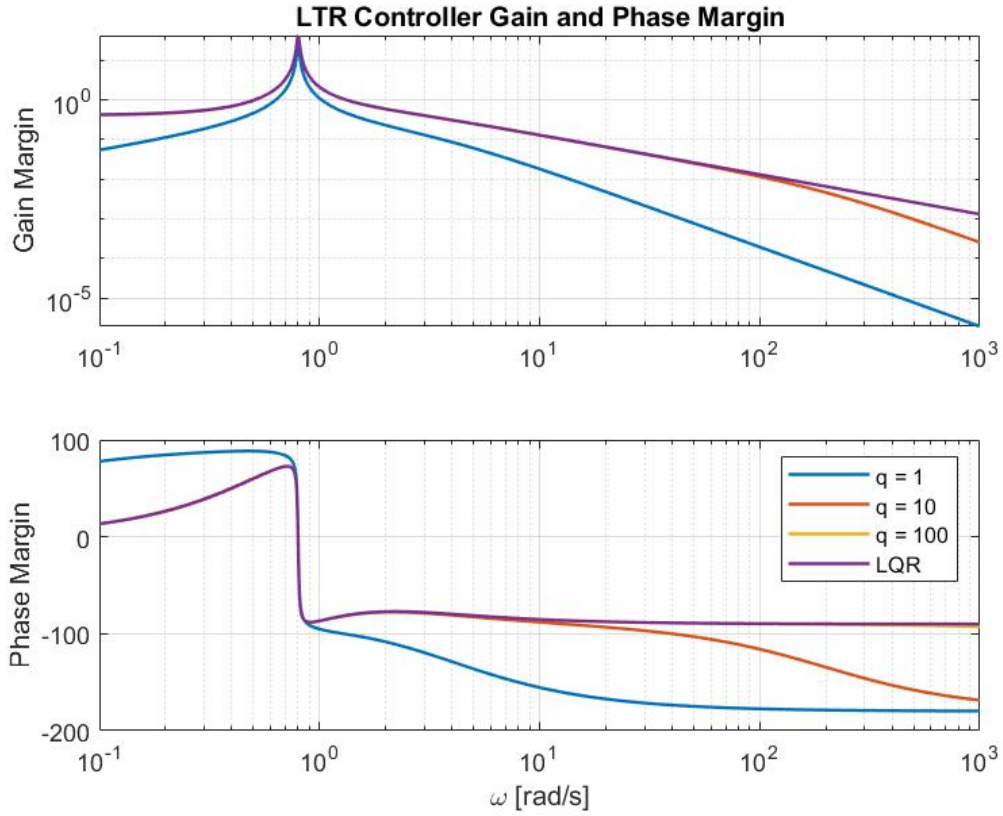


Figure 14: LTR Controller design for part B

1.4 Part D: Robustness

1.4.1 Part-A

The plots are obtained using the "ureal" command with the help of [5]. System in part-A is robustly stable for the modeled uncertainty as seen in 15. According to "robstab" command, the system can tolerate up to 433% of the modeled uncertainty. This perturbation causes an instability at the frequency 1.77 rad/seconds. Sensitivity with respect to each uncertain element is:

- 28% for p1. Increasing p1 by 25% decreases the margin by 7%.
- 8% for p2. Increasing p2 by 25% decreases the margin by 2%.
- 11% for p3. Increasing p3 by 25% decreases the margin by 2.75%.
- 0% for p4. Increasing p4 by 25% decreases the margin by 0%.
- 17% for p5. Increasing p5 by 25% decreases the margin by 4.25%.
- 33% for p6. Increasing p6 by 25% decreases the margin by 8.25%.

Lower bound on the actual robust stability margin of the model is 4.3263 whereas upper bound is 4.3353. Perturbations causing instability (wcu) are:

- p1: -1.1420
- p2: -0.3807
- p3: -1.8349
- p4: 1.4028
- p5: 1.4335
- p6: 0.5665

The algorithm can be obtained as seen in 4 for both part A and B by defining their state-space and LQR gains separately..

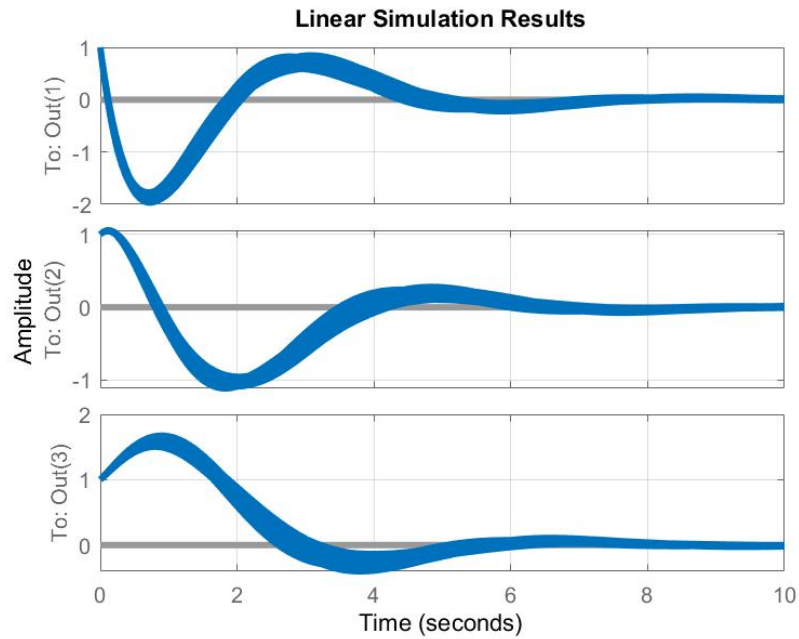


Figure 15: A and B matrices perturbed for part-A

```
%% Perurb separately the A and B matrices

p1 = ureal('p1',-2.0160,'Percentage',10);
p2 = ureal('p2',-0.6720,'Percentage',10);
p3 = ureal('p3',-1.2800,'Percentage',10);
p4 = ureal('p4',1,'Percentage',10);
p5 = ureal('p5',1,'Percentage',10);
p6 = ureal('p6',1,'Percentage',10);
A = [p1 p2 p3; p4 0 0 ;0 p5 0];
B = [p6; 0;0];
C = eye(3); % In order to reach full states
D = [0; 0; 0] % In order to reach full states
sys = ss(A,B,C,D) % Create uncertain state-space model
%% Uncertain closed loop system
Gcl = feedback(sys*K_Lqr, eye(3))

% Check robust stability margins for the uncertain system
opt = robOptions('Display','on','Sensitivity','on')
[stabmarg,wcu] = robstab(Gcl,opt)

% Step response for uncertain systems
T = 0:0.01:10
U = ones(size(T'))*[0 0 0]
x0 = [1 1 1]
lsim(Gcl, U, T, x0)
grid on

set(findall(gcf, 'type', 'line'), 'linewidth',3)
```


1.4.2 Part-B

Again the system is robustly stable for the modeled uncertainty as seen in figure 16. It can tolerate up to 445% of the modeled uncertainty. There is a destabilizing perturbation amounting to 454% of the modeled uncertainty. This perturbation causes an instability at the frequency 0.945 rad/seconds. Sensitivity with respect to each uncertain element is:

- 18% for p1. Increasing p1 by 25% decreases the margin by 4.5%.
- 9% for p2. Increasing p2 by 25% decreases the margin by 2.25%
- 12% for p3. Increasing p3 by 25% decreases the margin by 3
- 1% for p4. Increasing p4 by 25% decreases the margin by 0.25%.
- 15% for p5. Increasing p5 by 25% decreases the margin by 3.75%.
- 38% for p6. Increasing p6 by 25% decreases the margin by 9.5%.

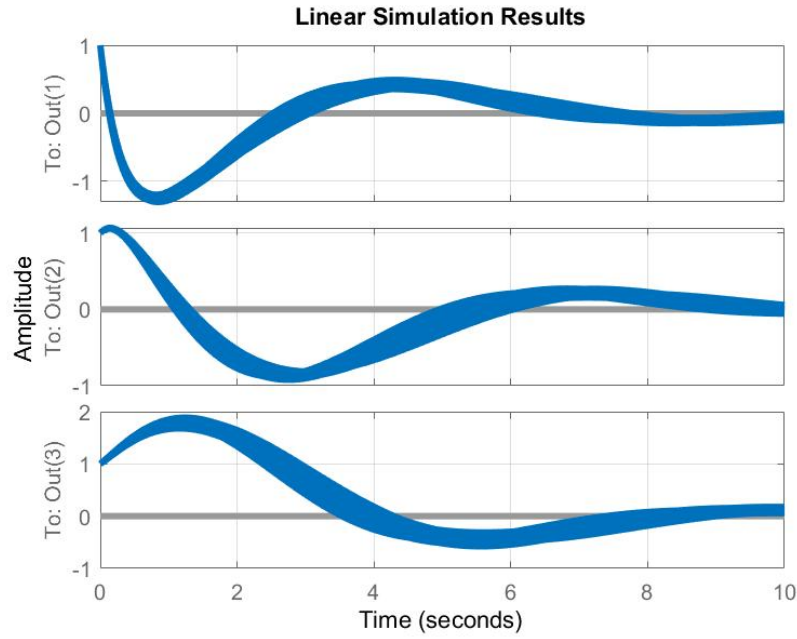


Figure 16: A and B matrices perturbed for part-B

Perturbations causing instability (wcu) are:

- p1: -1.1010
- p2: -0.3670
- p3: -1.8609
- p4: 0.5725

- p5: 1.4539
- p6: 0.5461

Appendix

LQR Design for Part-A and Part-B

```
1  clc;clear all;close all
2  %% Define systems (PART A & PART B)
3  % NOTE: RUN THE SYSTEM IN PART-A OR IN PART-B
4  s = tf('s');
5  %% First system --> Part A
6  Py = 1/((0.5*s + 1)*(s^2+2*(0.01)*(0.8)*s+(0.8)^2));
7
8  %% Second system --> Part B
9  % Py = (s^2+2*0.5*0.8*s+0.8^2)/((0.5*s + 1)*(s^2+2*(0.01)*(0.8)*s
    +(0.8)^2));
10
11 %% State-space representation
12 % Define A,B,C,D Matrices
13 [A,B,C,D] = tf2ss(cell2mat(Py.numerator),cell2mat(Py.denominator));
14 % State-space representation
15 sys = ss(A,B,C,D);
16 %% Test Controllability of the system
17 co = ctrb(sys);
18 controllability = rank(co);
19
20 %%
21 t = 0:0.01:10;
22 r = ones(size(t));
23 rho = [0.6, 0.5, 0.4, 0.3, 0.2];
24 % rho = [1, 0.1, 0.01];
25 disp('Full state feedback gain matrix(K), solution S of the
    associated algebraic Riccati equation(X) and the closed-loop
    poles(E)')
26 figure;
27 for i = 1:length(rho)
28     fprintf('For Rho value of %0.2f',rho(i))
29     %% Determine Q & R
30     Q = C'*C;
31     R(i) = rho(i);
32
33     [K,X,E] = lqr(A,B,Q,R(i))
34
35     % New states
36     states(i).Ac = [(A-B*K)];
37     states(i).Bc = [B];
38     states(i).Cc = [C];
39     states(i).Dc = [D];
40     stepSys.sys_cl(i) = ss(states(i).Ac,states(i).Bc,states(i).Cc,
        states(i).Dc)
41
42     %% Determine the precompensator G
43     fprintf('Pre-compansator:')
44     sys_ss = ss(A,B,C,D);
45     Gbar = rscale(sys_ss,K);
46     disp(Gbar);
47
```

```

48     %% Step response for a system with Precompansator --> Means
    that reference = Gbar*r
49     stepSys.sys_cl_precomp(i) = ss(states(i).Ac,states(i).Bc*Gbar,
    states(i).Cc,states(i).Dc);
50     [y,t,x] = lsim(stepSys.sys_cl(i),Gbar*r,t);
51     plot(t,y,'LineWidth',1.5);
52     legend_name(i) = sprintf("\rho$ = %.2f",rho(i));
53     hold on;
54
55     %% Lsf
56     Lsf(i) = K*inv(s.*eye(size(A))-A)*B;
57 end
58 legend(legend_name,'interpreter','latex')
59 ylabel('y');
60 xlabel('Time [sec]');
61 title("Step response for different \rho values");
62 grid;
63
64 %% Step response
65 % % No pre-compensator
66 % lsim(stepSys.sys_cl(1),stepSys.sys_cl(2),stepSys.sys_cl(3),
    stepSys.sys_cl(4),r,t)
67 % % With pre-compansator
68 % lsim(stepSys.sys_cl_precomp(1),stepSys.sys_cl_precomp(2),stepSys.
    sys_cl_precomp(3),stepSys.sys_cl_precomp(4),r,t)
69
70 %% Symmetric root locus plot
71 A_srl = [A zeros(size(A));-C'*C -A'];
72 B_srl = [B;-C'*D];
73 C_srl = [D'*C B'];
74 D_srl = D'*D;
75 sys_srl = ss(A_srl,B_srl,C_srl,D_srl);
76 figure;
77 rlocus(sys_srl)
78 grid
79 legend('SRL')
80
81 %% Loop Gain Lsf
82 % Compute the magnitude and phase of these responses
83 w = logspace(-2,2,1000);
84
85 figure;
86 for i = 1:length(rho)
87     bode(Lsf(i),w);
88     fprintf("GM,PM etc. for the \rho value = %.2f\n",rho(i));
89     S = allmargin(Lsf(i))
90     hold on;
91 end
92 grid minor;
93 legend(legend_name,'Interpreter','latex');
94 title('Bode plot for different \rho values');
95
96 % mag = squeeze(mag);
97 % magdb = 20*log10(mag)
98
99 %% Design LTR

```

```

100 w=logspace(-1,3,1000);
101 rho_ltr = 1;
102 Q_ltr = rho_ltr*C'*C;
103 r = 1;
104 tf = K_Lqr*inv(s*eye(size(A))-A)*B;
105 [K] = lqr(A,B,Q_ltr,r);
106 sys1 = ss(A,B,K,0);
107 [maggk1,phasgk1,w]=bode(sys1,w);
108
109 q = [1, 10, 100];
110 rv=1;
111 % MATLAB lqe
112 for i = 1:length(q)
113     gam=q(i)*B;
114     Q1=gam'*gam;
115     LTR(i).L=lqe(A,gam,C,Q1,rv);
116
117     % MATLAB bode MATLAB margin
118     aa=A-B*K-(LTR(i).L)*C;
119     bb=(LTR(i).L);
120     cc=K;
121     dd=0;
122     sysk=ss(aa,bb,cc,dd);
123     sysgk=series(sys1,sysk);
124     [maggk,phsgk,w]=bode(sysgk,w);
125     [gm,phm,wcg,wcp] = margin(maggk,phsgk,w)
126
127     subplot(2,1,1);
128     loglog(w, maggk(:),'linewidth',1.2); % loglog(w,[maggk1(:)
maggk(:)]);
129     grid;
130     hold all
131     subplot(2,1,2);
132     semilogx(w,phsgk(:),'linewidth',1.2); %semilogx(w,[phasgk1(:)
phsgk(:)]);
133     grid;
134     hold all;
135 end
136
137 subplot(2,1,1);
138 title('LTR Controller Gain and Phase Margin')
139 loglog(w, maggk1(:),'linewidth',1.2);
140 ylabel("Gain Margin")
141
142
143 subplot(2,1,2);
144 semilogx(w,phasgk1(:),'linewidth',1.2)
145 ylabel("Phase Margin")
146 xlabel('\omega [rad/s]')
147
148 legend('q = 1','q = 10','q = 100','LQR')

```

LTR Design for part-A

```

1 %%%%%%%%% PART C
2 clc;clear all;close all

```

```

3 %% Define systems (PART A & PART B)
4 % NOTE: RUN THE SYSTEM IN PART-A OR IN PART-B
5 s = tf('s');
6 %% First system --> Part A
7 Py = 1/((0.5*s + 1)*(s^2+2*(0.01)*(0.8)*s+(0.8)^2));
8
9 %% State-space representation
10 % Define A,B,C,D Matrices for PART-A
11 [A,B,C,D] = tf2ss(cell2mat(Py.numerator),cell2mat(Py.denominator));
12 % State-space representation
13 sys = ss(A,B,C,D);
14
15 %% Chosen LQR gains for PART-A
16 % % For rho = 0.3
17 % K_Lqr = [1.4711 4.0477 2.5893];
18
19 % For rho = 0.5
20 K_Lqr = [1.2389 3.2651 1.8246];
21
22 % Eigenvalues
23 % Or can be obtained from previous lqr func. result
24 % Closed loop pole locations
25 eigc = eig(A-B*K_Lqr);
26
27 % Check LQR Gain of the system
28 K_ack = acker(A,B,eigc);
29 %% Symmetric root locus plot PART-A
30 A_srl = [A zeros(size(A));-C'*C -A'];
31 B_srl = [B;-C'*D];
32 C_srl = [D'*C B'];
33 D_srl = D'*D;
34 sys_srl = ss(A_srl,B_srl,C_srl,D_srl);
35 figure;
36 rlocus(sys_srl)
37 % plot also the eigenvalues
38 hold all
39 plot(eigc,'o','Linewidth',2);
40 legend('SRL','Controller poles')
41 grid
42
43 %% Observer Poles & Find the Observer Gain
44 % The bandwidth is 1.23 rad/s. Therefore, choose pole loc as:
45 % estimator pole location
46 % Chose the gain as: G = 145
47 pe = [-1.48+2.37i; -3.09; -1.48-2.37i];
48 % Observer Gain
49 L = place(A',C',pe)
50
51 %% tf from process noise to sensor output
52 % Ge(s)
53 Ge = C * inv(s*eye(size(A)) - A)*B;
54 % Ge(-s)
55 Ge_neg = C * inv(-s*eye(size(A)) - A)*B;
56 figure;
57 rlocus(Ge_neg*Ge)
58 % Estimator SRL eqn: 1+q*Ge*Ge_neg

```

```

59
60 %% Symmetric root locus plot with Observer Gains
61 A_srl = [A zeros(size(A));-C'*C -A'];
62 B_srl = [B;-C'*D];
63 C_srl = [D'*C B'];
64 D_srl = D'*D;
65 sys_srl = ss(A_srl,B_srl,C_srl,D_srl);
66 figure;
67 rlocus(sys_srl)
68 % plot also the eigenvalues
69 hold on
70 plot(eigg,'o','Linewidth',2);
71 hold on
72 plot(pe,'o','Linewidth',2);
73 legend('SRL','Controller poles','Observer Poles')
74 grid
75
76 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
77 %% Simulink Part
78 N_bar = rscale(sys,K_Lqr);
79
80 noise_pow = [0.01, 0.1, 1, 10, 100];
81 figure;
82 for i = 1:length(noise_pow)
83     set_param('LQR_model/Band-Limited White Noise','Cov',string(
84         noise_pow(i)))
85     y = sim('LQR_model')
86     plot(y.yout{1}.Values.Time, y.yout{1}.Values.Data, 'linewidth',
87         1.25)
88     hold on
89 end
90
91 legend('0.01', '0.1' , '1' , '10', '100' )
92
93 title('Increased Noise Power and Simulation Results')
94
95 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
96 %% Design LTR
97 w=logspace(-1,3,1000);
98 rho_ltr = 0.5;
99 Q_ltr = rho_ltr*C'*C;
100 r = 1;
101 tf = K_Lqr*inv(s*eye(size(A))-A)*B;
102 [K] = lqr(A,B,Q_ltr,r);
103 sys1 = ss(A,B,K,0);
104 [maggk1,phasgk1,w]=bode(sys1,w);
105
106 q = [1, 10, 100];
107 rv=1;
108 % MATLAB lqe
109 for i = 1:length(q)
110     gam=q(i)*B;
111     Q1=gam'*gam;
112     LTR(i).L=lqe(A,gam,C,Q1,rv);
113
114 % MATLAB bode MATLAB margin

```

```

113     aa=A-B*K-(LTR(i).L)*C;
114     bb=(LTR(i).L);
115     cc=K;
116     dd=0;
117     sysk=ss(aa,bb,cc,dd);
118     sysgk=series(sys,sysk);
119     [magk,phsgk,w]=bode(sysgk,w);
120     [gm,phm,wcg,wcp] = margin(magk,phsgk,w)
121
122     subplot(2,1,1);
123     loglog(w, magk(:), 'linewidth',1.2);
124     grid;
125     hold all
126     subplot(2,1,2);
127     semilogx(w,phsgk(:), 'linewidth',1.2);
128     grid;
129     hold all;
130 end
131
132 subplot(2,1,1);
133 title('LTR Controller Gain and Phase Margin')
134 loglog(w, magk1(:), 'linewidth',1.2);
135 ylabel("Gain Margin")
136
137
138 subplot(2,1,2);
139 semilogx(w,phasgk1(:), 'linewidth',1.2)
140 ylabel("Phase Margin")
141 xlabel('\omega [rad/s]')
142
143 legend('q = 1', 'q = 10', 'q = 100', 'LQR')

```

LTR Design for part-B

```

1  %%%%%%%%% PART C -- system B
2  clc;clear all;close all
3  % NOTE:
4  s = tf('s');
5  %% Second system --> Part B
6  Py = (s^2+2*0.5*0.8*s+0.8^2)/((0.5*s + 1)*(s^2+2*(0.01)*(0.8)*s
    +(0.8)^2));
7
8  %% State-space representation
9  % Define A,B,C,D Matrices for PART-A
10 [A,B,C,D] = tf2ss(cell2mat(Py.numerator),cell2mat(Py.denominator));
11 % State-space representation
12 sys = ss(A,B,C,D);
13
14 %% Chosen LQR gains for PART-A and PART-B
15 % % For rho = 0.3
16 % K_Lqr = [2.7887 2.8440 1.3845];
17
18 % For rho = 0.5
19 K_Lqr = [2.0370 2.1814 0.9370];
20
21 % Eigenvalues

```



```

22 % Or can be obtained from previous lqr func. result
23 eiggg = eig(A-B*K_Lqr);
24
25 %% Symmetric root locus plot PART-B
26 A_srl = [A zeros(size(A));-C'*C -A'];
27 B_srl = [B;-C'*D];
28 C_srl = [D'*C B'];
29 D_srl = D'*D;
30 sys_srl = ss(A_srl,B_srl,C_srl,D_srl);
31 figure;
32 rlocus(sys_srl)
33 % plot also the eigenvalues
34 hold all
35 plot(eiggg,'o','Linewidth',2);
36 legend('SRL','Controller poles')
37 grid
38
39 %% tf from process noise to sensor output
40 % Ge(s)
41 Ge = C * inv(s*eye(size(A)) - A)*B;
42 % Ge(-s)
43 Ge_neg = C * inv(-s*eye(size(A)) - A)*B;
44 figure;
45 rlocus(Ge_neg*Ge)
46 % Estimator SRL eqn: 1+q*Ge*Ge_neg
47
48 %% Observer Poles & Find the Observer Gain
49 % The bandwidth is 1.23 rad/s. Therefore, choose pole loc as:
50 % estimator pole location
51 % Chose the gain as: G = 145
52 pe = [-0.378+0.709i; -6.09; -0.378-0.709i];
53 % Observer Gain
54 L = place(A',C',pe)
55
56 %% Symmetric root locus plot with Observer Gains
57 A_srl = [A zeros(size(A));-C'*C -A'];
58 B_srl = [B;-C'*D];
59 C_srl = [D'*C B'];
60 D_srl = D'*D;
61 sys_srl = ss(A_srl,B_srl,C_srl,D_srl);
62 figure;
63 rlocus(sys_srl)
64 % plot also the eigenvalues
65 hold on
66 plot(eiggg,'o','Linewidth',2);
67 hold on
68 plot(pe,'o','Linewidth',2);
69 legend('SRL','Controller poles','Observer Poles')
70 axis([-7 7 -1 1])
71 grid
72
73 %%%%%%%%%%%%%%%
74 %% Simulink Part
75 N_bar = rscale(sys,K_Lqr);
76
77 noise_pow = [0.01, 0.1, 1, 10, 100];

```

```

78 figure;
79 for i = 1:length(noise_pow)
80     set_param('LQR_model/Band-Limited White Noise','Cov',string(
        noise_pow(i)))
81     y = sim('LQR_model')
82     plot(y.yout{1}.Values.Time, y.yout{1}.Values.Data, 'linewidth',
        1.25)
83     hold on
84 end
85
86 legend('0.01', '0.1' , '1' , '10', '100' )
87
88 title('Increased Noise Power and Simulation Results')
89
90 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
91 %% Design LTR
92 w=logspace(-1,3,1000);
93 rho_ltr = 1;
94 Q_ltr = rho_ltr*C'*C;
95 r = 1;
96 tf = K_Lqr*inv(s*eye(size(A))-A)*B;
97 [K] = lqr(A,B,Q_ltr,r);
98 sys1 = ss(A,B,K,0);
99 [maggk1,phasgk1,w]=bode(sys1,w);
100
101 q = [1, 10, 100];
102 rv=1;
103 % MATLAB lqe
104 for i = 1:length(q)
105     gam=q(i)*B;
106     Q1=gam'*gam;
107     LTR(i).L=lqe(A,gam,C,Q1,rv);
108
109     % MATLAB bode MATLAB margin
110     aa=A-B*K-(LTR(i).L)*C;
111     bb=(LTR(i).L);
112     cc=K;
113     dd=0;
114     sysk=ss(aa,bb,cc,dd);
115     sysgk=series(sys,sysk);
116     [maggk,phsgk,w]=bode(sysgk,w);
117     [gm,phm,wcg,wcp] = margin(maggk,phsgk,w)
118
119     subplot(2,1,1);
120     loglog(w, maggk(:), 'linewidth', 1.2);
121     grid;
122     hold all
123     subplot(2,1,2);
124     semilogx(w, phsgk(:), 'linewidth', 1.2);
125     grid;
126     hold all;
127 end
128
129 subplot(2,1,1);
130 title('LTR Controller Gain and Phase Margin')
131 loglog(w, maggk1(:), 'linewidth', 1.2);

```

```

132 ylabel("Gain Margin")
133
134
135 subplot(2,1,2);
136 semilogx(w,phasgk1(:),'linewidth',1.2)
137 ylabel("Phase Margin")
138 xlabel('\omega [rad/s]')
139
140 legend('q = 1','q = 10','q = 100','LQR')

```

References

- [1] Inverted pendulum: State-space methods for controller design. URL <https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum\T1\textsectionion=ControlStateSpace>.
- [2] Function rscale: Finding the scale factor to eliminate steady-state error. URL https://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_rscale.
- [3] Introduction: State-space methods for controller design. URL <https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction\T1\textsectionion=ControlStateSpace>.
- [4] Gene F. Franklin, David J. Powell, and Abbas Emami-Naeimi. *Feedback control of Dynamic Systems*. Pearson, 2020.
- [5] Help center. URL <https://www.mathworks.com/support/search.html/videos/robust-control-part-4-working-with-parameter-uncertainty-1586760097653.html?fq%5B%5D=category%3Arobust%2Funcertain-models&page=1>.