**EE6225 Assignment-2**

Wu Tianwei

Matriculation No. G2101446F

e-mail: WU0008EI@e.ntu.edu.sg

Consider a Consider a 3×3 process given by Ogunnaike and Ray (1979); the transfer function matrix is obtained by empirical modeling technique as:



Try to design following controllers using ETF based methods

1. Decentralized control
2. Sparse control
3. Decoupling control

Compare the performance with the control system based on classical decentralized BLT control method by simulating the closed-loop performances for step response R1(t)=1 for t>0, R2(t)=1 for t>50 and R3(t)=1 for t>100, respectively. (Try to adjust the simulation time frame if the simulation horizons are not suitable).

**Solution:**

**1. Calculate the Equivalent Transfer Functions for**

**1) Loop pairing**

Using given transfer function matrix, steady-state gain and Relative Gain Array (RGA) can be calculated as:

According to the loop pairing rules, select 1-1、2-2、3-3.

Since , the loop pairing for the system is stable.

**2) Equivalent Transfer Function Based on RNGA Approximation**

Normalized Gain and Relative Normalized Gain Array (RNGA) :

Relative Average Residence Time Array (RARTA) :

**3) Determine Equivalent Transfer Function without Integrity Rules**

For FOPTD, we have

Which results

Therefore

手机屏幕截图

描述已自动生成**4) Determine Equivalent Transfer Function with Integrity Rules**

**Table Ⅰ ETFS and PI parameters for FOPTD process**

According to table Ⅰ, we have

**5) Matlab Code**

|  |
| --- |
| % Define the transfer function matrix obtained by empirical modeling technique  G = [tf(-0.98, [12.5, 1], 'inputDelay', 17), tf(-0.36, [15, 1], 'inputDelay', 27), tf(-0.14, [15.2, 1], 'inputDelay', 32);  tf(-0.43, [14.7, 1], 'inputDelay', 25), tf(-0.92, [13, 1], 'inputDelay', 16), tf(-0.11, [15.6, 1], 'inputDelay', 33);  tf(-0.12, [15, 1], 'inputDelay', 31), tf(-0.16, [15, 1], 'inputDelay', 34), tf(-1.02, [11.8, 1], 'inputDelay', 16)];    [rows, cols] = size(G);  K = zeros(rows, cols);  L = zeros(rows, cols);  T = zeros(rows, cols);  K\_N = zeros(rows, cols);  for i = 1 : rows  for j = 1 : cols  K(i, j) = G(i, j).num{1}(end);  L(i, j) = G(i, j).InputDelay + G(i, j).ioDelay;  T(i, j) = G(i, j).den{1}(1);  K\_N(i,j) = K(i, j) / ( T(i, j) + L(i, j));  end  end    % Calculate RGA  Lambda = K .\* (K^-1)';  NI = det(K) / (K(1, 1) \* K(2, 2) \* K(3, 3));    % Calculate RNGA  Lambda\_N = K\_N .\* (K\_N^-1)';  Gamma = Lambda\_N ./ Lambda;    % Without Integrity Rules  ETF = etf(G, Lambda, Gamma);    % With Integrity Rules  [ETF\_IR, PID\_IR] = eft\_PID\_IR(G, Lambda, Gamma); |
| % ETFs without Integrity Rules  function G\_hat = etf(G, Lambda, Gamma )  [rows, cols] = size(G);  G\_hat = [tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1)];  K = zeros(rows, cols);  L = zeros(rows, cols);  T = zeros(rows, cols);  for i = 1 : rows  for j = 1 : cols  K(i,j) = G(i,j).num{1}(end);  L(i,j) = G(i,j).InputDelay + G(i,j).ioDelay;  T(i, j) = G(i, j).den{1}(1);  G\_hat(i,j) = tf(K(i,j) / Lambda(i,j), [Gamma(i,j) \* T(i,j), 1], 'inputdelay', Gamma(i,j) \* L(i,j));  end  end  end |
| % ETFs and PID parameters without Integrity Rules  function [G\_hat, PID] = eft\_PID\_IR(G, Lambda, Gamma)  Am = 3;  [rows, cols] = size(G);  G\_hat = [tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1)];  K = zeros(rows, cols);  L = zeros(rows, cols);  T = zeros(rows, cols);  PID = cell(rows, cols);  for i = 1 : rows  for j = 1 : cols  K(i,j) = G(i,j).num{1}(end);  L(i,j) = G(i,j).InputDelay + G(i,j).ioDelay;  T(i, j) = G(i, j).den{1}(1);  if abs(Lambda(i, j)) < 1  if Gamma(i, j) > 1  G\_hat(i, j) = tf(K(i, j) / Lambda(i, j), [Gamma(i, j) \* T(i, j), 1], 'inputDelay', Gamma(i, j) \* L(i, j));  PID{i, j} = [pi \* Lambda(i, j) \* T(i, j) / 2 / Am / L(i, j) / K(i, j), pi \* Lambda(i, j) / 2 / Am / Gamma(i, j) / L(i, j) / K(i, j)];  elseif Gamma(i, j) > 0 && Gamma(i, j) <= 1  G\_hat(i, j) = tf(K(i, j) / Lambda(i, j), [T(i, j), 1], 'inputDelay', L(i, j));  PID{i, j} = [pi \* Lambda(i, j) \* T(i, j) / 2 / Am / L(i, j) / K(i, j), pi \* Lambda(i, j) / 2 / Am / L(i, j) / K(i, j)];  end  else  if Gamma(i, j) > 1  G\_hat(i, j) = tf(sign(Lambda(i, j)) \* K(i, j), [Gamma(i, j) \* T(i, j), 1], 'inputDelay', Gamma(i, j) \* L(i, j));  PID{i, j} = [pi \* T(i, j) / 2 / Am / L(i, j) / K(i, j), pi / 2 / Am / Gamma(i, j) / L(i, j) / K(i, j)];  elseif Gamma(i, j) > 0 && Gamma(i, j) <= 1  G\_hat(i, j) = tf(sign(Lambda(i, j)) \* K(i, j), [T(i, j), 1], 'inputDelay', L(i, j));  PID{i, j} = [pi \* T(i, j) / 2 / Am / L(i, j) / K(i, j), pi / 2 / Am / L(i, j) / K(i, j)];  end  end  end  end  end |

**2. Design Decentralized Control using ETF**

**1) Design Decentralized PID controller**

For FOPTD loop transfer functions and for different combination of and modes, the ETF has the form

So, the ETFs are

The ETFs together with PID controller parameters based on gain and phase margin method for FOPTD loop transfer functions are shown in Table Ⅱ.

According to Table Ⅱ, with and , the decentralized PID controller is

表格

描述已自动生成**Table Ⅱ ETFS and PI controllers for gain and phase margin method**

图表, 折线图

描述已自动生成**2) Simulation**

图示, 示意图

描述已自动生成

As shown in the figure above, all three outputs converge to 1 responding to the unit step inputs. Therefore, all three PID controllers are designed properly and successfully control the three outputs.

**3. Design Sparse Control using ETF**

**1) Design the structure of Sparse Control**

Based on the Interaction Index method, the index matrix is calculated as

With criterion , the sparse control should have the following structure

**2) Design PID controller based on Gain and Phase Margin method**

According to Table Ⅱ, with and , the sparse PID controller is

**3) Simulation**

图示, 示意图

描述已自动生成

图表, 折线图

描述已自动生成

As shown in the figure above, all three outputs converge to 1 responding to the unit step inputs. Therefore, all five PID controllers are designed properly and successfully control the three outputs.

**4. Design Decoupling Control using ETF with Integrity Rules**

**1) Normalized Decoupling**

According to the normalized decoupling control system design rules, the diagonal forward transfer matrix is selected as

The decoupler can be calculated by

**2) Design PID Controllers based on Gain and Phase Margin Method**

With and , the decoupling PID controller is

**3) Matlab Code**

|  |
| --- |
| %Decoupling Control with Integrity Rules  G\_R = [tf([ETF\_IR(1,1).num{1}], [ETF\_IR(1,3).den{1}], 'inputDelay', ETF\_IR(1,3).InputDelay + ETF\_IR(1,3).ioDelay), 0, 0;  0, tf([ETF\_IR(2,2).num{1}], [ETF\_IR(2,3).den{1}], 'inputDelay', ETF\_IR(2,3).InputDelay + ETF\_IR(2,3).ioDelay), 0;  0, 0, tf([ETF\_IR(3,3).num{1}], [ETF\_IR(3,2).den{1}], 'inputDelay', ETF\_IR(3,2).InputDelay + ETF\_IR(3,2).ioDelay)];    G\_hat\_I = [tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1)];    for i = 1 : rows  for j = 1 : cols  G\_hat\_I(i, j) = tf(G\_R(j, j).num{1}(end) / ETF\_IR(j, i).num{1}(end) \* ETF\_IR(j, i).den{1}, G\_R(j,j).den{1}, ...  'inputDelay', abs(G\_R(j, j).InputDelay) + abs(G\_R(j, j).ioDelay) - abs(ETF\_IR(j, i).InputDelay)- abs(ETF\_IR(j, i).ioDelay));  end  end    PID\_decoupling = PID\_controller(G\_R);  simout\_Decoupling\_IR = sim('Decoupling\_PID\_Controller\_IR');  plot(simout\_Decoupling\_IR); |
| function PID = PID\_controller(G)  Am = 3;  [rows, cols] = size(G);  K = zeros(rows, cols);  L = zeros(rows, cols);  T = zeros(rows, cols);  PID = cell(rows, cols);  for i = 1 : rows  K(i,i) = G(i,i).num{1}(end);  L(i,i) = G(i,i).InputDelay + G(i,i).ioDelay;  T(i,i) = G(i,i).den{1}(1);  PID{i,i} = {pi \* T(i, i) / 2 / Am / L(i, i) / K(i, i), pi / 2 / Am / L(i, i) / K(i ,i), 0};  end  end |

**4) Simulation**

图示, 示意图

描述已自动生成

图表, 折线图

描述已自动生成

As shown in the figure above, all three outputs can’t perfectly converge to 1 responding to 3 unit step inputs. Obviously, the outputs all fluctuate around 1 but can’t stabilize at 1. So, the decoupling controller with integrity rules are not designed properly and I’ll try to redesign it using ETF without integrity rules.

**5. Design Decoupling Control using ETF without Integrity Rules**

**1) Normalized Decoupling**

According to the normalized decoupling control system design rules, the diagonal forward transfer matrix is selected as

The decoupler can be calculated by

**2) Design PID Controllers based on Gain and Phase Margin Method**

With and , the decoupling PID controller is

**3) Matlab Code**

|  |
| --- |
| %Decoupling Control without Integrity Rules  G\_R = [tf([ETF(1,1).num{1}], [ETF(1,1).den{1}], 'inputDelay', ETF(1,3).InputDelay + ETF(1,3).ioDelay), 0, 0;  0, tf([ETF(2,2).num{1}], [ETF(2,2).den{1}], 'inputDelay', ETF(2,2).InputDelay + ETF(2,2).ioDelay), 0;  0, 0, tf([ETF(3,3).num{1}], [ETF(3,3).den{1}], 'inputDelay', ETF(3,3).InputDelay + ETF(3,3).ioDelay)];    G\_hat\_I = [tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1);  tf(0, 1), tf(0, 1), tf(0, 1)];    for i = 1 : rows  for j = 1 : cols  G\_hat\_I(i, j) = tf(G\_R(j, j).num{1}(end) / ETF(j, i).num{1}(end) \* ETF(j, i).den{1}, G\_R(j,j).den{1}, ...  'inputDelay', abs(G\_R(j, j).InputDelay) + abs(G\_R(j, j).ioDelay) - abs(ETF(j, i).InputDelay)- abs(ETF(j, i).ioDelay));  end  end    PID\_decoupling = PID\_controller(G\_R);  simout\_Decoupling = sim('Decoupling\_PID\_Controller');  plot(simout\_Decoupling); |
| function PID = PID\_controller(G)  Am = 3;  [rows, cols] = size(G);  K = zeros(rows, cols);  L = zeros(rows, cols);  T = zeros(rows, cols);  PID = cell(rows, cols);  for i = 1 : rows  K(i,i) = G(i,i).num{1}(end);  L(i,i) = G(i,i).InputDelay + G(i,i).ioDelay;  T(i,i) = G(i,i).den{1}(1);  PID{i,i} = {pi \* T(i, i) / 2 / Am / L(i, i) / K(i, i), pi / 2 / Am / L(i, i) / K(i ,i), 0};  end  end |

**4) Simulation**

图示, 示意图

描述已自动生成

图表, 折线图

描述已自动生成

As shown in the above figure, all the outputs can converge to 1 responding to the unit step inputs. The curves look much better than the decoupling PID controller using ETF with integrity rules, which means the decoupler and PID controller is designed properly.

**6. Classical Decentralized Control based on Biggest Log Modulus Tuning Method**

**1) Compute the Ziegler-Nichols tuning parameters for Each Individual Loop**

For each loop, the Bode Diagram is shown in the following plots. Using Bode plot, the gain margin and ultimate frequency can be easily determined.

图表, 图示

描述已自动生成

For , the ultimate gain and the ultimate frequency .

图表, 图示

描述已自动生成

For , the ultimate gain and the ultimate frequency .

图表

描述已自动生成

For , the ultimate gain and the ultimate frequency .

According to the Ziegler–Nichols method, and for PI control. So, and are calculated as

**2) Design PID controller based on BLT**

Choose the range of as 2 to 5 and the step of adjustment is 0.05. The order of the system , so .

After tuning, is finally set to 2. Therefore,

**3) Matlab Code**

|  |
| --- |
| % Bode plot  figure(1), margin(-G(1, 1));  figure(2), margin(-G(2, 2));  figure(3), margin(-G(3, 3));    % Calculate Ku and Wu  all\_margin = [allmargin(-G(1, 1)), allmargin(-G(2, 2)), allmargin(-G(3, 3))];  Ku = [-all\_margin(1).GainMargin(1), -all\_margin(2).GainMargin(1), -all\_margin(3).GainMargin(1)];  Wu = [all\_margin(1).GMFrequency(1), all\_margin(2).GMFrequency(1), all\_margin(3).GMFrequency(1)];    K\_ZN = Ku ./ 2.2;  T\_ZN = 2 \* pi ./ (1.2 \* Wu);    min\_error = 10^9;  N = 3;  for F = 2 : 0.05 : 5  K\_C = K\_ZN / F;  T\_I = F \* T\_ZN;  K\_I = K\_C ./ T\_I;  max\_Lc = 0;  Gc = [tf([K\_C(1), K\_I(1)], [1, 0]), 0, 0;  0, tf([K\_C(2), K\_I(2)], [1, 0]), 0;  0, 0, tf([K\_C(3), K\_I(3)], [1, 0])];  for w = 0.1 : 0.01 : 1  W = -1 + det(eye(3) + freqresp(G \* Gc, w));  Lc = 20 \* log10(abs(W / (1 + W)));  if Lc > max\_Lc  max\_Lc = Lc;  end  end  error = abs(max\_Lc - 2 \* N);  if error < min\_error  min\_error = error;  F\_match = F;  end  end    K\_C = K\_ZN / F\_match; % P  T\_I = F\_match \* T\_ZN;  K\_I = K\_C ./ T\_I; % I  simout\_Decentralized\_BLT = sim('Decentralized\_PID\_Controller\_BLT');  plot(simout\_Decentralized\_BLT); |

**4) Simulation**

图示, 示意图

描述已自动生成

图表

描述已自动生成

As shown in the figure above, all three outputs converge to 1 responding to the unit step inputs, but the time it takes to get stable is much longer than previous ones.

**7. Comparison**

**1) Output**

图表

低可信度描述已自动生成

**2) Output**

图表

描述已自动生成

**3) Output**

图示

描述已自动生成

According to the three figures above, it is obvious that decentralized control method and sparse control method using ETF have similar results and they are acceptable. The classical decentralized control method based on BLT performs much worse than the previous two methods. The decoupling method using ETF without Integrity Rules is harder to obtain a good result than decentralized control method and sparse control method using ETF, but the three results are not so bad.