ISSN 0280-5316
ISRN LUTFD2/TFRT--5698--SE

Controller Synthesis by Linear Programming

Zhang Zhimin

Department of Automatic Control
Lund Institute of Technology
November 2002

Department of Automatic Control **Lund Institute of Technology** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* November 2002 *Document Number* ISRN LUTFD2/TFR--5698--SE
Author(s) Zhang Zhimin	*Supervisor* Anders Rantzer LTH *Sponsoring organization*

Title and subtitle
Controller Synthesis by Linear Programming (Regulatorsyntes via linjärprogrammering).

Abstract
Recent progress in parameterization of linear robust controllers has made it possible to investigate advanced controller specifications numerically using convex optimization. The main focus of this thesis project is to investigate how to design a LTI controller for a LTI plant using convex optimization, particularly Linear Programming technique. The concerned performance specification is defined as lower and upper bounds of time domain responses to step inputs. With the help of Q-parameterization, this control design requirement is formulated as a linear programming problem, which can be solved very easily and efficiently. As part of this thesis work, a Matlab toolbox is developed for the calculation of a numeric controller by using linear programming algorithm. Two practical design problems, double tank process and mass-spring system, are solved using this toolbox. The calculated controllers have been put into simulation for verification. Comparisons are also conducted between optimization based design and other control design methods. It has been found that optimization-based design is particularly suitable for the investigation of performance limitation and tradeoffs.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 59	*Recipient's notes*
Security classification		

Contents

ABSTRACT

Recent progress in parameterization of linear robust controllers has made it possible to investigate advanced controller specifications numerically using convex optimization. The main focus of this thesis project is to investigate how to design a LTI controller for a LTI plant using convex optimization, particularly Linear Programming technique. The concerned performance specification is defined as lower and upper bounds of time domain responses to step inputs. With the help of Q-parameterization, this control design requirement is formulated as a linear programming problem, which can be solved very easily and efficiently. As part of this thesis work, a Matlab toolbox is developed for the calculation of a numeric controller by using linear programming algorithm. Two practical design problems, double tank process and mass-spring system, are solved using this toolbox. The calculated controllers have been put into simulation for verification. Comparisons are also conducted between optimization based design and other control design methods. It has been found that optimization-based design is particularly suitable for the investigation of performance limitation and tradeoffs.

ACKNOWLEDGEMENT

I would like to thank my supervisor Prof. Anders Rantzer for his constant inspiration and valuable ideas and time for instructing of my thesis work!

I also would like to express my thanks to Prof. Karl Erik Arzen, Prof. Per Hangander, Prof. Rolf Johansson, Dr. Anders Robertsson and Agneta Tuszynski, Eva Schildt and Britt-Marie Mårtensson and all of the staffs in the department for their nice help during my stay in the department!

Finally, many thanks to my family for their continuous encouragement and support during my study!

Nov. 7, 2002 Lund
Zhang Zhimin

1. Introduction

1.1 Background

During the past few decades, control engineering enjoyed great advancement in both theoretic development and industrial applications. The purpose of control engineering is to devise a strategy to improve the performance of a system by adding sensors, control processors, and actuators. The sensors measure various signals in the system and command signals; the control processors process these signals and then drive the actuator which will affect the behavior of the system. A diagram of a general control system is shown as follows:

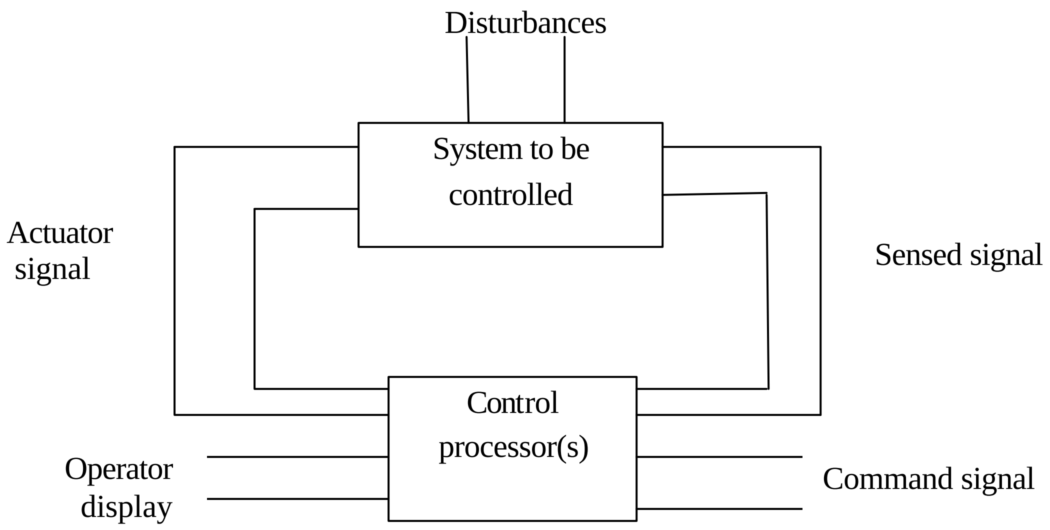


Figure 1-1: general control system diagram

Like any other theory in science and engineering, the establishment of control theory requires the simplification of plant and disturbance model. In control terminology, a framework has to be defined. Any design problem should be cast into this framework in order for the corresponding design method to be applicable. Complementary to the theoretic development, numeric/computational approaches have become a separate topic in many areas because of the growth of CPU power. Numerical methods, especially optimization theory, have achieved significant advances in recent years. Controller design based on numerical methods, especially convex optimization, has attracted more and more attentions. The controller design problem is converted to a pure numeric optimization problem and can be solved for a restricted set of systems and a restricted set of design specifications by combining theoretical results with numerical convex optimization techniques. The optimization problem is then solved numerically on a computer.

As mentioned above, we need make the restrictions on the systems and design specifications for applying the optimization algorithm to the controller design. The systems we studied here must be linear and time-invariant (LTI). And the specifications must be closed-loop convex. This restricted set of design specifications includes a wide class of performance specifications (how the closed-loop system should perform).

For controller design problems of restricted form, we can determine if the given specifications can be achieved or not, and the limits of performance can be easily studied for a given system and a given control configuration.

1.2 Optimization-based control design

Optimization-based control design is a very broad topic. The framework for the concerned control design problem is shown in figure 1-2. *G* is the model of the plant. It is often called augmented plant since extra models, inputs and outputs are often added to the original process for the purpose of imposing performance requirement. *C* is the controller. *y* is the measured signal, *u* is the controller output and also the input of the process. *w* is the disturbance or reference signal. *z* is the performance vector. The design requirement is specified on the closed-loop map from *w* to *z*. Note that every signal in this figure could be a vector.

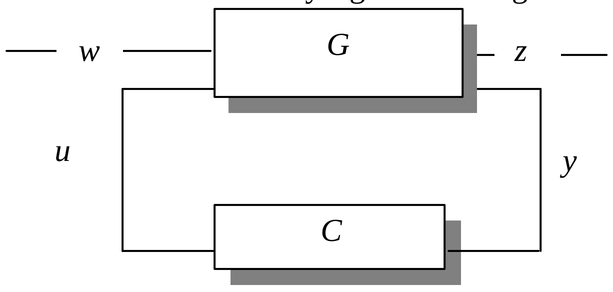


Figure 1-2 Controller design framework

In the most general sense, plant *P* and controller *C* both could be nonlinear and time varying. It is not difficult to convert the control design problem into numeric optimization problem for any given design specification. The biggest problem lies in the solvability of the derived optimization problem. Therefore, restrictions on the plant should be imposed. For a finite dimensional LTI plant, if a LTI controller is sought, then many design requirements can be cast into convex optimization problem with the help of Q parameterization.

1.3 Organization of the report

This thesis work is organized as follows:

Chapter 2 focuses on the theoretical derivation of optimization-based design method. In this chapter, the so-called Q parameterization and the controller calculation from the optimized Q will be introduced. In order to formulate the linear programming problem, the control framework for both stable and unstable plant will be formed. Finally, a simple example will be shown on how to formulate the linear programming problem from a design specification.

Chapter 3 demonstrates the optimization-based design method for two real processes: the stable double tank process and the unstable mass-spring system. Closed loop performance will be shown and verified through simulation. The results are also compared to those using other design methods.

Chapter 4 discusses the possible difficulties in the use of optimization-based design method. The focus is on the selection of bounds which will influence the closed-loop performance very sensitively. Performance specifications will be used to be the references of the bounds define.

Chapter 5 summarizes this thesis work. Suggestions for the future work will also be proposed.

The appendix will consist of simulation model, computer toolbox code and the bibliography.

2. Theoretical Part

2.1 Introduction

Consider the framework in figure 1-2. Let us assume the generalized plant G to be finite dimensional and linear time invariant, having the form of

$$\begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} G_{zw}(s) & G_{zu}(s) \\ G_{yw}(s) & G_{yu}(s) \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix} \tag{2-1}$$

where G_{zw}, G_{zu}, G_{yw}, G_{yu} are the open-loop transfer functions from inputs to outputs, or

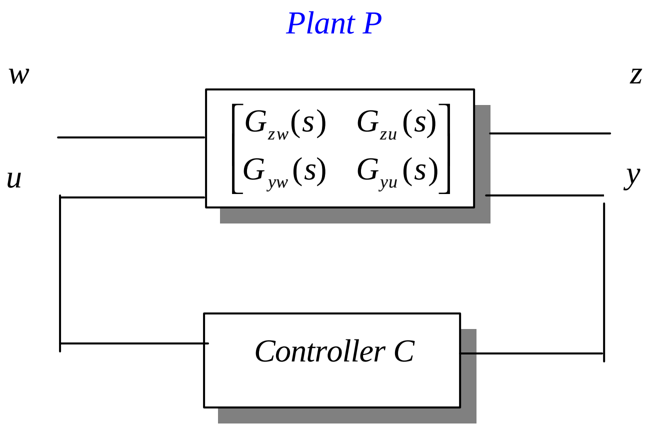
$$\begin{bmatrix} \dot{x} \\ z \\ y \end{bmatrix} = \begin{bmatrix} A & B_w & B_u \\ C_z & D_{zw} & D_{zu} \\ C_y & D_{yw} & D_{yu} \end{bmatrix} \begin{bmatrix} x \\ w \\ u \end{bmatrix} \tag{2-2}$$

and now suppose the controller is operating, u=C*y , we can solve for z in terms of w to get

$$z = (G_{zw} + G_{zu}(I - C * G_{yu})^{-1} * C * G_{yw}) * w \tag{2-3}$$

The closed-loop map from *w* to *z* can be readily written as

$$H_{clp} = G_{zw} + G_{zu}(I - C * G_{yu})^{-1} * C * G_{yw} \tag{2-4}$$



Plant P
w
z
$\begin{bmatrix} G_{zw}(s) & G_{zu}(s) \\ G_{yw}(s) & G_{yu}(s) \end{bmatrix}$
u
y
Controller C

Figure 2-1. framework for optimization-based design

The control design problem can be posed as to find the stabilizing controller C to make the closed loop system from w to z to satisfy certain convex constraints.

2.2 Design structure for stable plant

In equation (2-3), if we define S = $(I - C * G_{yu})^{-1} * C$, if S is used as optimization variables in linear programming problem, after performing optimization algorithm, the controller can be calculated from S as

$$C = (I + S * G_{yu})^{-1} * S \tag{2-5}$$

It is important that this design method is only valid for stable plant, for unstable plant, the so-call Q parameterization will be used for the design.

We will explain in detail how to build the framework for the stable plant and to convert it to linear programming problem

First, we will consider a stable plant in the following structure:

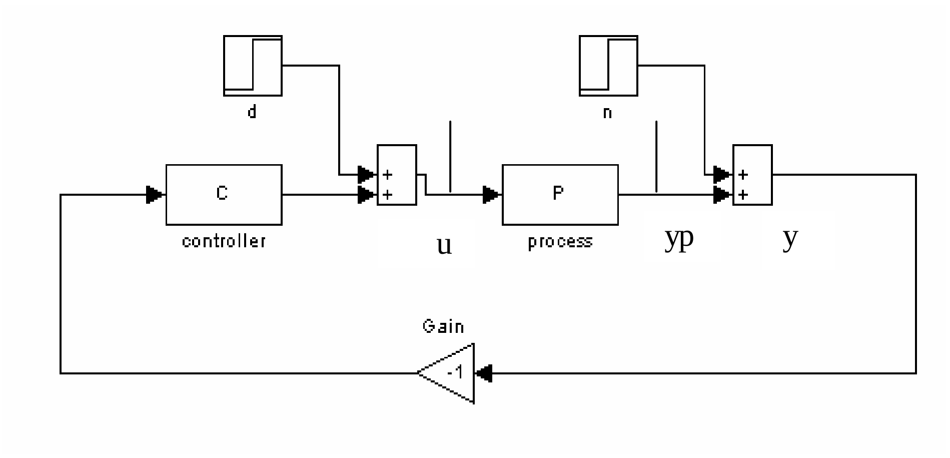


Figure 2-2. disturbance and noise rejection for a plant

The input to the system is the step process disturbance and step sensor noise. Then the following transfer function could be formulated

$$\begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} -cp/1+cp & -c/1+cp \\ p/1+cp & 1/1+cp \end{bmatrix} \begin{bmatrix} d \\ n \end{bmatrix} \tag{2-6}$$

In (2-6), z = $\begin{bmatrix} u \\ y \end{bmatrix}$ and w = $\begin{bmatrix} d \\ n \end{bmatrix}$, closed loop map from w to z is

$$H_{clp} = \begin{bmatrix} -cp/1+cp & -c/1+cp \\ p/1+cp & 1/1+cp \end{bmatrix} \tag{2-7}$$

Each transfer function in H_{clp} is very important, the second row consists of the closed loop transfer functions from the process disturbance and sensor noise to the output signal y. What we need do is to make these two transfer functions "small". The "size" of these two transfer functions told us something about the closed-loop control achieved by our control system. The first row consists of the closed-loop transfer functions from the process disturbance and sensor noise to the control signal u. We can notice that in H_{clp} all of the closed loop transfer functions of interest in our controller design has been included. Therefore, the specifications for a controller design could be expressed in terms of the four transfer functions in H_{clp}.

In equation (2-6), c is the controller, p is the plant, d is process disturbance, n is sensor noise, for simplicity, the d an n are all considered as the step inputs.

The performance requirement for the controller design is specified as step response bounds, that is,

Given d(k)=1(k), u(k) < ub_1(k) or u(k)>lb_1(k)
Given n(k)=1(k), u(k) < ub_2(k) or u(k) > lb_2(k)
Given d(k)=1(k), y(k) < ub_3(k) or y(k) > lb_3(k)
Given n(k)=1(k), y(k) < ub_4(k) or y(k) > lb_4(k)

Compared with equation (2-5), controller will be calculated from

$$c=Q/(1-Q*p) \tag{2-8}$$

which means Q=c/(1+c*p);
Then we got the following structure:

$$\begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} QP & Q \\ P - QP^2 & 1 - QP \end{bmatrix} \begin{bmatrix} d \\ n \end{bmatrix} \tag{2-9}$$

Until now, we have converted H_{clp} to the form in (2-9), as we can see, each element in H_{clp} has the form of R*Q+W which can be easily converted to the linear programming problem in the future.

2.3 Design structure for unstable plant.

As is shown in figure 2-1, the generalized plant G could be formulated as follows:

$$G=\begin{bmatrix} 0 & 0 & 1 \\ p & 0 & -p \\ p & 1 & -p \end{bmatrix} \text{ and } \begin{bmatrix} u \\ yp \\ y \end{bmatrix} = G*\begin{bmatrix} d \\ n \\ u \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ p & 0 & -p \\ p & 1 & -p \end{bmatrix} * \begin{bmatrix} d \\ n \\ u \end{bmatrix}$$

As we stated before, for unstable plant, Q parameterization should be used to perform optimization design.

Q-parameterization states that, any closed-loop transfer function H_{clp} achievable by a stabilizing controller C, no matter if the open-loop system *G* is stable or not, can be expressed in the linear form of some stable *Q*:

$$H_{clp} = T_{11} + T_{12} * Q * T_{21}. \tag{2-10}$$

Here T_{11}, T_{12}, T_{21} are only dependent on open-loop plant *G*, and can be calculated as follows:

Suppose that there exits matrices *K* and *L* such that $A - B_u L$ and $A - KC_y$ are stable (i.e. stability and detectability). Then T_{11}, T_{12}, T_{21} are given as

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{21} & 0 \end{bmatrix} = C_T (SI - A_T)^{-1} B_T + D_T \tag{2-11}$$

where

$$A_T = \begin{bmatrix} A - B_u L & B_u L \\ 0 & A - KC_y \end{bmatrix}, \quad B_T = \begin{bmatrix} B_w & B_u \\ B_u - KD_{yw} & 0 \end{bmatrix},$$

$$C_T = \begin{bmatrix} C_z - D_{zu} L & D_{zu} L \\ 0 & C_y \end{bmatrix}, \quad D_T = \begin{bmatrix} D_{zw} & D_{zu} \\ D_{yw} & 0 \end{bmatrix}.$$

A nice interpretation of *Q*-parameterization is given in figure 2-3, and all stabilizing controller can be represented based on an observer for the system in the following way:

$$\begin{bmatrix} \dot{\hat{x}} \\ u \\ e \end{bmatrix} = \begin{bmatrix} A\hat{x} + B_u u + Ke \\ r - L\hat{x} \\ y - C_y \hat{x} - D_{yu} u \end{bmatrix} \tag{2-12}$$

where $r = Q(l)e$ for some admissible, stable transfer function Q.

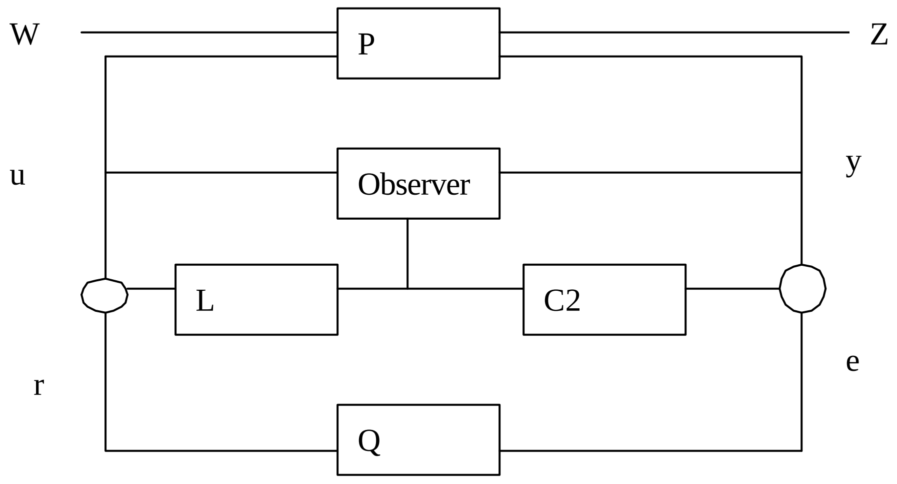


Figure 2-3: An interpretation of the Q-parametrizaiton(with D22=0).

In order to get the state space form of the controller, assume Q has the following state space representation

$$\begin{bmatrix} \dot{x}_Q \\ r \end{bmatrix} = \begin{bmatrix} A_Q & B_Q \\ C_Q & D_Q \end{bmatrix} \begin{bmatrix} x_Q \\ e \end{bmatrix} \tag{2-13}$$

Combining (2-12) and (2-13), the optimal controller can be derived as

$$C = C_k(lI - A_k)^{-1} B_k + D_k$$

with

$$A_k = \begin{bmatrix} A - B_u L - KC_y - B_u D_Q C_y & B_u C_Q \\ -B_Q C_y & A_Q \end{bmatrix} \qquad B_k = \begin{bmatrix} K + B_u D_Q \\ B_Q \end{bmatrix} \tag{2-14}$$

$$C_k = \lfloor -L - D_Q C_y \quad C_Q \rfloor, \qquad D_k = D_Q\,.$$

2.4 Linear programming formulation

In this section, we will derive the control design problem from above design structure. The design objective is to have good rejection of disturbance and measurement noise. The performance requirement is specified on step response. To make the problem simpler, the controller is assumed to be single input and single output. So Q or S is also a scale transfer function. The parameterization of Q is taking the form of

$$Q = q_0 + q_1 z^{-1} + q_2 z^{-2} + \cdots + q_m z^{-m}, \qquad (2\text{-}15)$$

And for Q we only take *m* terms.

Note that for above plants, as seen from the derived equation (2-9), All of the elements in the closed-loop transfer matrices are linear form of Q, with the following form:

$$H=R(z)*Q+W(z) \qquad (2\text{-}16)$$

Where $R(z) = r_0 + r_1 z^{-1} + r_2 z^{-2} + \cdots + r_n z^{-l}$

$$W(z) = w_0 + w_1 z^{-1} + w_2 z^{-2} + \cdots + w_n z^{-n}$$

are known, Please also note that at least n terms of the R should be used, i.e. The index of the last bounds point we defined. The controller should be designed to make sure that the responses of the system are within the bounds during the time instant which are defined by user. And the step response of the system should converge to the desired value which is also defined by the user.

Assum $H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + \cdots + h_n z^{-n}$, Then, by simple algebraic manipulations, we have

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_l \\ h_{l+1} \\ \vdots \\ h_n \end{bmatrix} = E \begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_{m-1} \\ q_m \end{bmatrix} + \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix} \qquad (2\text{-}17)$$

$$\text{where} \quad E = \begin{bmatrix} r_1 & 0 & 0 & 0 \\ r_2 & r_1 & 0 & 0 \\ r_3 & r_2 & r_1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ r_n & r_{n-1} & r_{n-3} & r_{n-m} \end{bmatrix}$$

As we can derive, for a plant $H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + \cdots + h_n z^{-n}$, its step response has z-transform:

$$Y(z) = h_0 + (h_0 + h_1) z^{-1} + (h_0 + h \;\; + h_2) z^{-2} + \cdots + (h_0 + h_1 + \cdots + h_n) z^{-n}$$

The bounds condition of its step response can be converted to the following LP problem:

*Af*x<bf*

And we form the problem in the following procedure:

First:

$$A_< = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}_{n1xn1} \qquad x = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_n \end{bmatrix}$$

$b_< =$ upper bounds

$b_> =$ lower bounds

for example, if an upper bound at the time instant i is defined, then the i row of the A should be the positive, and the row of the b should be the upper bound, otherwise, the row of the A should change the sign. And the lower bounds should also change the sign and be put into i row of the b, and then the following form will be formed:

A(Gq+w) < b ,it can be converted as (AG)q < b-Aw, and we denote it as:

$$\mathrm{Afl} * \mathrm{q} < \mathrm{bfl} \tag{2-18}$$

In our problem not only above equation should be satisfied, we also intend to minimize the bounds such that the optimal solution can be calculated, and at the same time, the response of the system should also converge to the desired value. Then a variable gamma will be introduced to get the following formula:

$$\mathrm{Afl} * \mathrm{q} - \mathrm{bdes} <= \mathrm{gamma} * (\mathrm{bfl} - \mathrm{bdes}) \tag{2-19}$$

Gamma will be one of the design variables, i.e. x= $\begin{bmatrix} q \\ gamma \end{bmatrix}$. After the translations, the final linear programming problem will be formulated in the following way:

$$[\mathrm{Af1}, \mathrm{des} - \mathrm{b}] * [\mathrm{q}; \mathrm{gamma}] <= \mathrm{des} - \mathrm{A} * \mathrm{w0} \tag{2-20}$$

From equation (2-19), the parameters for the LP problem is:.

$$\mathrm{A} = [\mathrm{Af1}, \mathrm{des} - \mathrm{b}], \mathrm{b} = [\mathrm{des} - \mathrm{A} * \mathrm{w0}]; \tag{2-21}$$

and the objective function is to minimize variable gamma.

2.5 An simple example

In order to have a rough idea on how to formulate the Linear programming problem from the performance specification, a simple example will be given in the following:
Consider the following control structure:

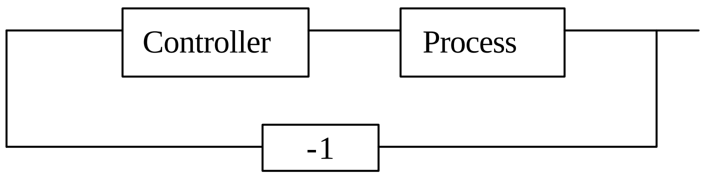


Figure 2-4: a closed loop structure for demonstrating the linear programming formulation from a control problem

The sensitivity function has been given in the form of

S(z) = (q0+q1*z^-1+q2*z^-2+q3*z^-3)/(z-0.5);

and the step response of the sensitivity function:

Y(z)=S(z)*z/(z-1)

has the following specifications at different time instant:

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \end{bmatrix} = \begin{bmatrix} 0 \\ [1,2] \\ [3,4] \\ [1,2] \\ [0,1] \end{bmatrix}$$

q0, q1, q2, q3 are design variables. The problem is that does such a S(z) exist such that the step response of S(z), i.e. Y(z) are within the bounds as given above at different time instant? To solve such problem, first,

$$\begin{aligned} Y(z) &= S(z) * z/(z-1) \\ &= z * (q0 + q1 * z^{\wedge}\text{-}1 + q2 * z^{\wedge}\text{-}2 + q3 * z^{\wedge}\text{-}3)/(z - 0.5) * (z-1) \\ &= (2/(z-1) - 1/(z-0.5))\ (q0 + q1 * z^{\wedge}\text{-}1 + q2 * z^{\wedge}\text{-}2 + q3 * z^{\wedge}\text{-}3), \end{aligned}$$

Mapping Y(z) to Y(k), the following formula could be obtained:

$$\begin{aligned} Y(k) &= q0 * (2 - 0.5^{\wedge}(k-1)) * u(k-1) + q1 * (2 - 0.5^{\wedge}(k-2) * u(k-2)) \\ &\quad + q2 * (2 - 0.5^{\wedge}(k-3) * u(k-3)) + q1 * (2 - 0.5^{\wedge}(k-4) * u(k-4)) \end{aligned}$$

k should be in the integer value of 0,1,2,3,4, insert value of k in above Y(k), then corresponding value of the Y(k) is

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \end{bmatrix} = \begin{bmatrix} 0 \\ q0 \\ 1.5*q0+q1 \\ 1.75*q0+1.5*q1+q2 \\ 1.875*\mathrm{q0}+1.75*\mathrm{q1}+1.5*\mathrm{q2}+\mathrm{q3} \end{bmatrix} = \begin{bmatrix} 0 \\ [1,2] \\ [3,4] \\ [1,2] \\ [0,1] \end{bmatrix} \tag{2-22}$$

To determine the existence of the feasible region, we can use a fake objective function in order to use linear programming function from Matlab:

F=q0+q1+q2+q3=F' *x;

$$x = \begin{bmatrix} q0 \\ q1 \\ q3 \\ q4 \end{bmatrix}$$

Linear programming problem is to minimize F' *x, under the condition A*x<b, BL=<x<=BU. In which

$$\mathrm{F}=\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix};\ \mathrm{AU}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1.5 & 1 & 0 & 0 \\ 1.75 & 1.5 & 1 & 0 \\ 1.875 & 1.75 & 1.5 & 1 \end{bmatrix};\ \mathrm{BL}=\begin{bmatrix} 1 \\ 3 \\ 1 \\ 0 \end{bmatrix};\ \mathrm{BU}=\begin{bmatrix} 2 \\ 4 \\ 2 \\ 1 \end{bmatrix}$$

Write in stack form:

$$\mathrm{A}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1.5 & 1 & 0 & 0 \\ 1.75 & 1.5 & 1 & 0 \\ 1.875 & 1.75 & 1.5 & 1 \\ -1 & 0 & 0 & 0 \\ -1.5 & -1 & 0 & 0 \\ -1.75 & -1.5 & -1 & 0 \\ -1.875 & -1.75 & -1.5 & -1 \end{bmatrix} *\mathrm{x}< \begin{bmatrix} 2 \\ 4 \\ 2 \\ 1 \\ -1 \\ -3 \\ -1 \\ 0 \end{bmatrix} =\mathrm{b} \tag{2-23}$$

Call Matlab function LINPROG (F,A,b,[],[]), then the solution will be calculated which mean there has such a solution such that the step response of the given sensitivity function are within the bounds defined.

3. Application results

3.1 Double tank process

3.1.1 Introduction of the process

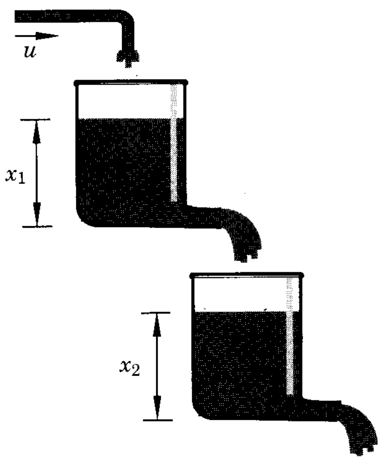


Figure 3-1: diagram for the double tank process

The inflow to the upper tank is generated by a pump which is controlled by an external voltage. The outflow from the upper tank is then in turn the inflow to the lower tank. Two level sensors make it possible to measure the levels in the two tanks. A tap placed on the upper tank to introduce the load disturbance. In our problem, the input signal is the flow to the first tank and the output is the level in the second tank.

The continuous transfer function of the double tank process is:

$$G(s) = \frac{0.0002456\, s + 0.06769}{s^2 + 0.3857\, s + 0.03627} \tag{3-1}$$

Using sampling time of 1 second, discrete transfer function of the double tank will be

$$H(q) = \frac{0.030q + 0.026}{q^2 - 1.65q + 0.68} \tag{3-2}$$

The pole, zeros map and bode diagram are shown in figure 3-2. it can be seen that this is a stable and slow process. The bandwidth of this process is 0.01rad/s.

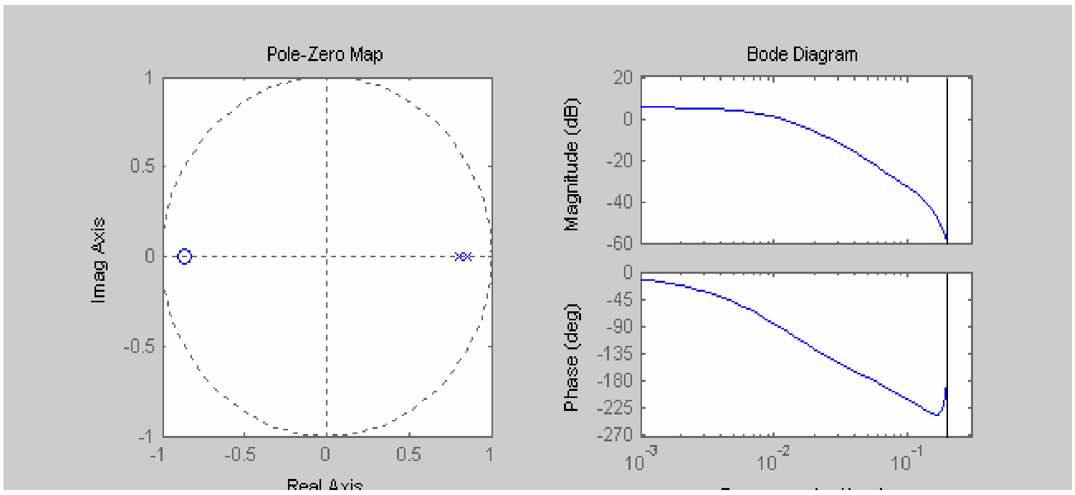


Figure 3-2: poles and bode diagram for double tank process

3.1.2 Design with PID controller

Using the simple PID design method, the following closed-loop step responses could be obtained (figure 3-3) with kp=2, ki=0.2, kd=0. Due to the existence of the integrator in the controller, the output y of the second tank settles to zero under the input disturbance d and sensor noise n. The setting time is about 40 seconds. In the steady state, the control action u, which is the water flowing into the first tank, will remain about 0.5. This can be verified from equation (3-1) by setting s=0 or equation (3-2) by setting q=1.

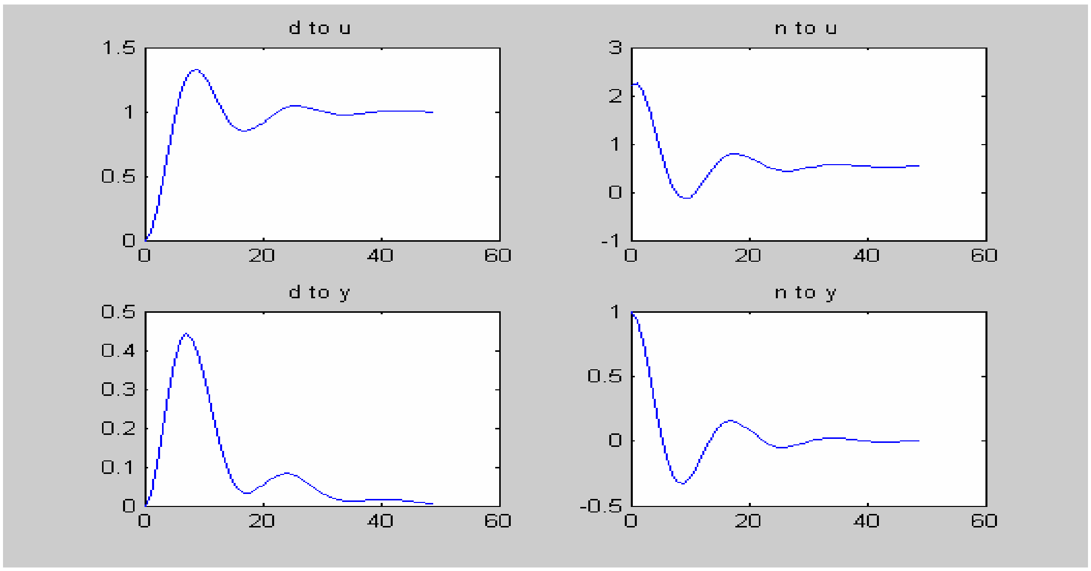


Figure 3-3: Step response of the u and y to the process disturbance d and sensor noise n

3.1.3 Optimization based design

Figure 3-4 shows the results from optimization-based design. Black ‘o’ represents the desired value. Blue one is the real response of the system. The red ‘*’ represents the bounds defined by the user, and the green ‘*’ represent the tightest bounds which can be achieved (the tightest bounds is defined as gamma*(bounds-des_value)). It can be seen that the user-defined bounds value can be reduced to the green one and the performance specification is still be fulfilled.

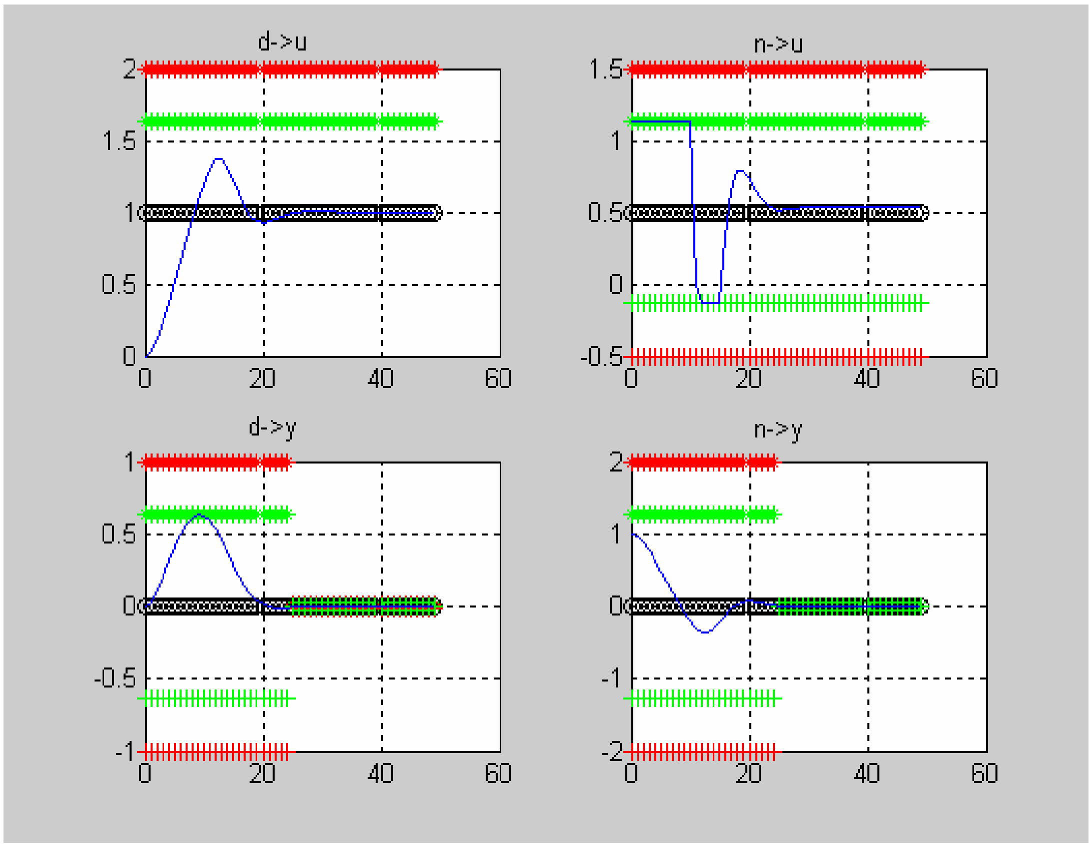


Figure 3-4: Step response of u and y to d and n based on the optimization design.

In terms of closed-loop performance, optimization-based design gives 25 seconds settling time, and close to zero steady state error, which is better than the PID design in figure 3-3. Controller calculated from optimization based design is of 17th order, Q is in order 15.

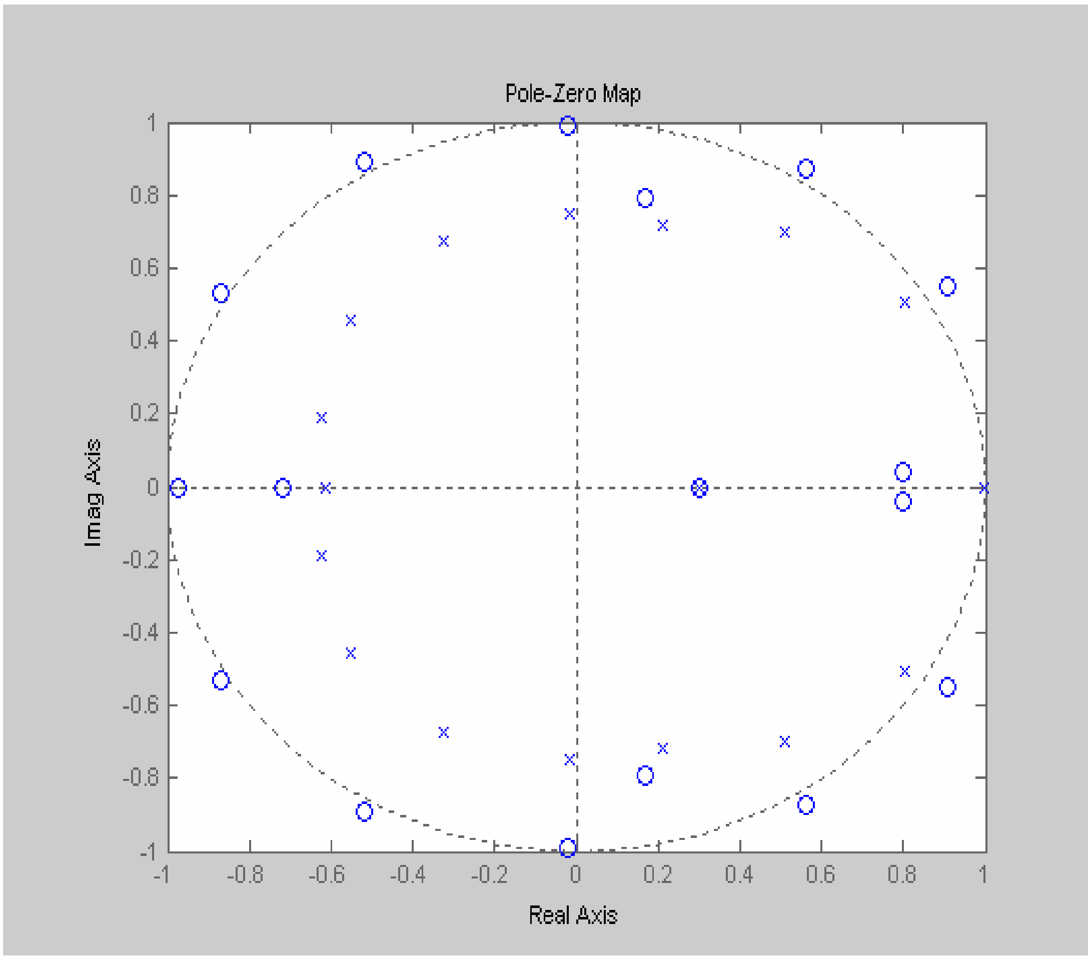


Figure 3-5: poles, zeros of the controller calculated from optimization-based design

It can be shown that there is a pole approaching to 1 in the designed controller, which corresponds to include an integrator in the controller. This explains why close to zero steady error can be achieved in figure 3-4. It can also be noticed that the exact integrator is not achieved due to the numeric round error and bound definition. Therefore, theoretically zero steady state error is not guaranteed.

After obtained the controller, a simulation model has been built to test the design, the results are show below, compared with the PID controller we used before, It can be shown that the computer tool box we have developed for the stable process works pretty well. The settling time has become much shorter compared with PID design.

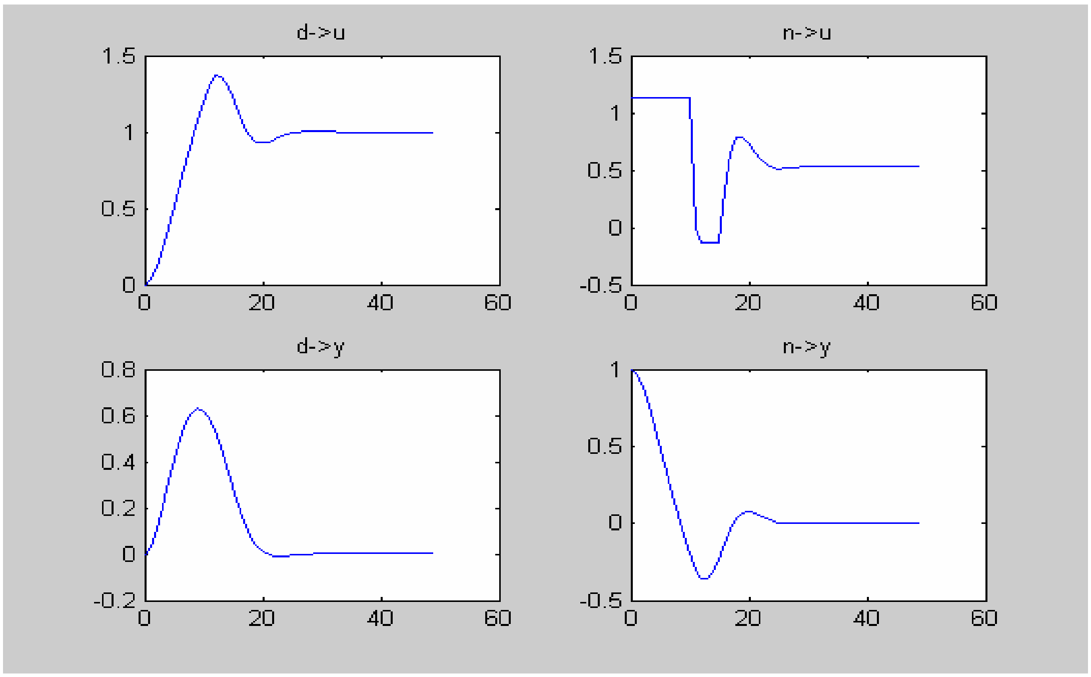


Figure 3-6: simulation result get from the optimization based design

3.2 Mass-spring system

3.2.1 Introduction of the process

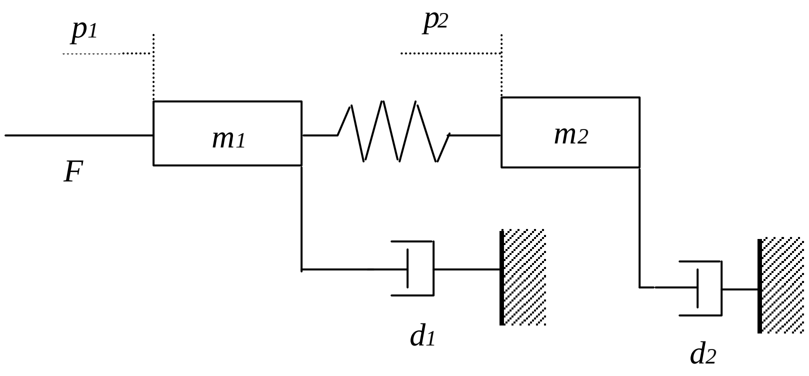


Figure 3-7: Diagram for the mass spring system

As we can see from above figure 3-7, there are two masses connected with a spring in the system. The mass on one side of the spring can be moved by a linear motor. This side is called the "motor side" and the other side is called the "load side". In two damping, only d2 can be seen as a part in the process. And d1 is modeled as in above figure. Our purpose is to control the position p2. i.e. the position of the mass on the load side, but in many applications, only the position on the motor side, p1 can be measured. Therefore, the control problem now becomes to control a signal which we can not measure. First, we will show how to get a linear model of the process: the weights for the two masses are m1 and m2, and the damps of the two masses are d1 and d2 respectively. The spring constant between them is k. one of the masses is controlled by a DC-motor, driven by a current controlled amplified. The dynamics of the motor and amplifier can be neglected. The relationship between the driving force of the motor F and the input voltage of the amplify u can be written as: F=Km*u. Finally, the dynamical model for the force balance equations can be obtained as follows:

$$\mathrm{m}_1 * \frac{d^2P_1}{dt^2} = \text{-d} - d_1 * \frac{dp1}{dt} - k*(p1-p2)+F(t)+l(t); \tag{3-3}$$

$$\mathrm{m}_2 * \frac{d^2P_2}{dt^2} = - d_2 * \frac{dp_2}{dt} + k*(p1-p2) \tag{3-4}$$

In which l(t) load disturbance and a state space representation with the state vector $x = \begin{bmatrix} p_1 \\ \dot{p}_1 \\ p_2 \\ \dot{p}_2 \end{bmatrix}$ can be written as:

$\dot{x}$ (t)=A*x(t)+B*u(t)+B$_l$*l(t)=

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ -k/m_1 & -d_1/m_1 & k/m_1 & 0 \\ 0 & 0 & 0 & 1 \\ k/m_2 & 0 & -k/m_2 & -d_2/m_2 \end{bmatrix} *x(t)+ \begin{bmatrix} 0 \\ k_m/m_1 \\ 0 \\ 0 \end{bmatrix} *u(t)+ \begin{bmatrix} 0 \\ 1/m_1 \\ 0 \\ 0 \end{bmatrix} *l(t) \tag{3-5}$$

$$y(t)=C*x(t)= \begin{bmatrix} C1 \\ C2 \end{bmatrix} *x(t)= \begin{bmatrix} k_{y1} & 0 & 0 & 0 \\ 0 & 0 & k_{y2} & 0 \end{bmatrix} x(t) \tag{3-6}$$

In our design, y2 is the variable which we want to control, and y1 is the variable that can be measured in practice. and k, d_1 m_1 k_{y2} k_{y1} m_2 are all constants that could be measured from the experiment.

3.2.2 Design with state feedback controller

The control law we will use here is state space feedback including an integrator. In addition, because in practice, only the variable y1 could be measured, so y2 will be observed from other states, the structure is as follows:

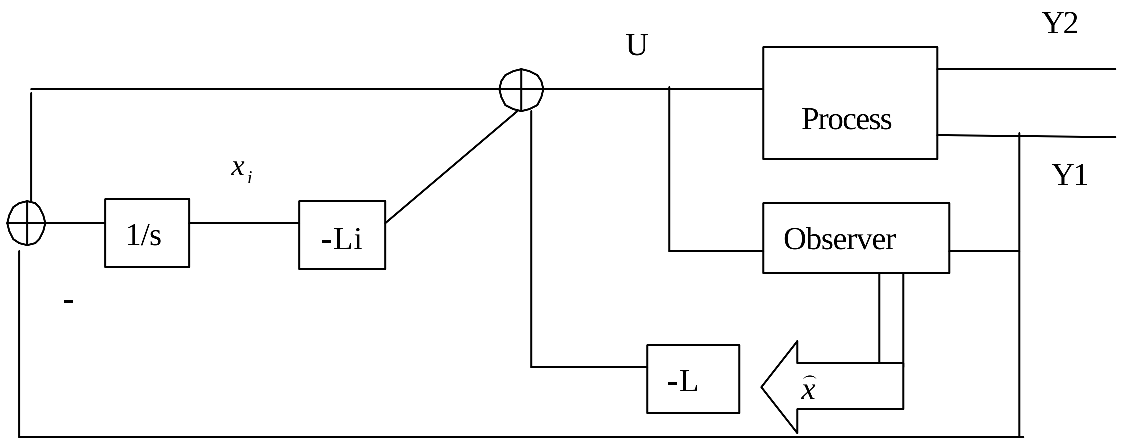


Figure 3-8: State feedback with integral action

<u>*Specifications:*</u>

Specifications are those requirements that we want the controlled system to fulfill. In our problem, we only study the closed loop system behavior in the time domain. A well damped step response with a rise time between 0.2 and 0.4 seconds is then specified. At the same time, the control signal should not be too ' large' because it will lead to undesirable damages on the motor.

<u>*Control strategy and design*</u>

It is difficult to control the process with feedback from only one state, so all of our four state will be used to do feedback, the system is controllable, because $\begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix}$ has full rank. So we can place all the poles freely by the state feedback. Due to only the y1 can be measured, so the estimation for other states is necessary. This is possible to do since the system is also observable.

<u>How to get L:</u>

Assume that all the states could be measured in our problem. We could place the poles for the closed loop system by using the control law

$\dot{x}$(t)=A*x(t)+B*u(t) (3-7)

u(t)=-Lx(t); (3-8)

Because the system is a 4th order system, so we just define four poles according to the specification, now we want to calculate L such that A-BL get designed properties. The function **PLACE** in Matlab will give a L-vector according to the pole defined by us. The poles we used in the design can be illustrated by the following figure:

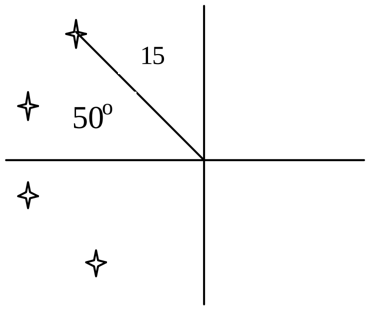


Figure 3-9: The location of the poles

<u>How to get K:</u> (Reconstruction of the states)

As we mentioned before, it is impossible to measure all the states in practice and only the position on the motor side, y1 can be measured. Therefore, we will build up a so called observed model (or called Kalman filter), what we should do is just copy the process model and feed it with the same input signal as the real process. The difference between the output of the real process and the model is used to correct the model states such that they should approach the process states. The observer can be written by:

$$\frac{d\,\hat{x}(t)}{dt} = A*\hat{x}(t) + B*u(t) + K*(y_1(t) - c_1 * \hat{x}(t)) \qquad (3\text{-}9)$$

In equation (3-9), $\hat{x}(t)$ is the states of the observer. K is the observer gain and can be set such that the observer states are approaching the real states with an arbitrary convergence speed. The observer states instead of the real states (which we can not measure) will be used in the state feedback design. Now we want to calculate K such that A-K*C1 gets designed properties. If we just write the transpose $(A-K*C_1)^T = A^T - C_1^T * K^T$; take K^T as the design variable, A^T, C_1^T as input parameters, then the function PLACE will solve the K. Normally, the observer dynamics will be chosen to be 1.5 to 2 times faster than the closed loop system.

<u>How to get Li:</u>

The pure state feedback works well when the model matches the real process and no disturbances entered into the system. However, a stationary error will be presented for a constant load disturbance. To overcome this problem, introducing the integral action into the controller is necessary. As shown in the figure 3-6, the controller states are the observer states $\hat{x}$ and the integrator x_i. Writing the following equations :

$$\frac{d\hat{x}(t)}{dt} = A*\hat{x} + B*u + K*(y1 - c1*\hat{x}); \qquad (3\text{-}10)$$

$$\frac{dx_i(t)}{dt} = -y1; \qquad (3\text{-}11)$$

$$u = -L*\hat{x} - Li*x_i \qquad (3\text{-}12)$$

Consider the u as the output from the controller, and y1 as the input to the controller, the sate space form of the controller could be formulated:

$$\frac{d\begin{bmatrix} x \\ x_i \end{bmatrix}}{dt} = \begin{bmatrix} A - B*L - K*C1 & -B*Li \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} \hat{x} \\ x_i \end{bmatrix} + \begin{bmatrix} K \\ -1 \end{bmatrix} *y1 \qquad (3\text{-}13)$$

$$u = [-L\ -Li]*\begin{bmatrix} \hat{x} \\ x_i \end{bmatrix} \qquad (3\text{-}14)$$

According to the obtained stated feedback gain L, observer gain K, set value for Li, then the controller could be obtained:

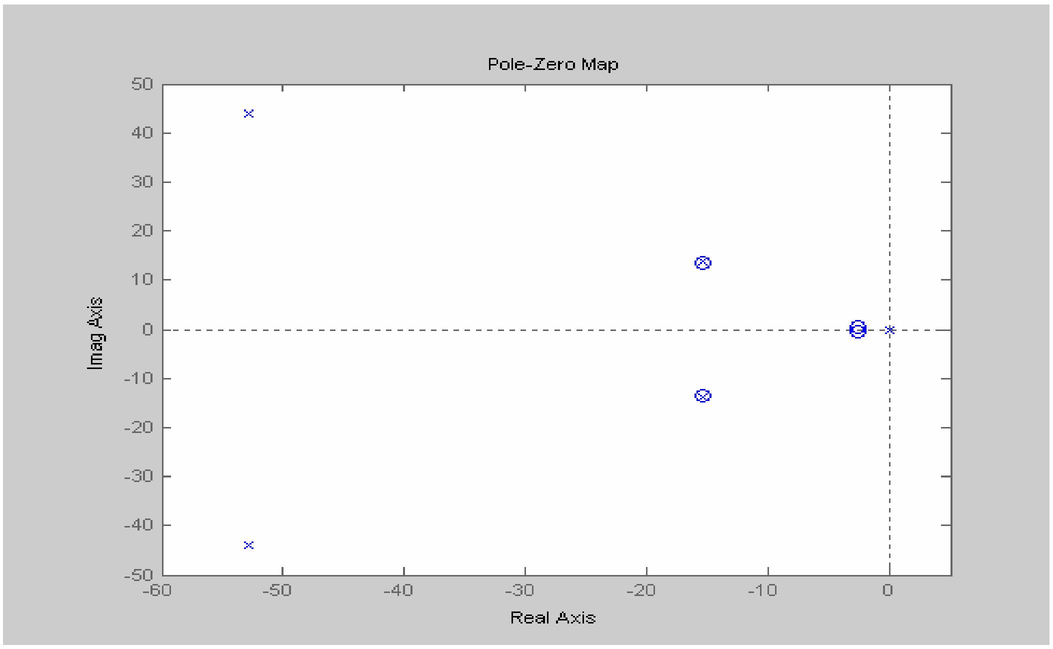


Figure 3-10: controller calculated from the state feedback design

Results from the L, K, Li design:

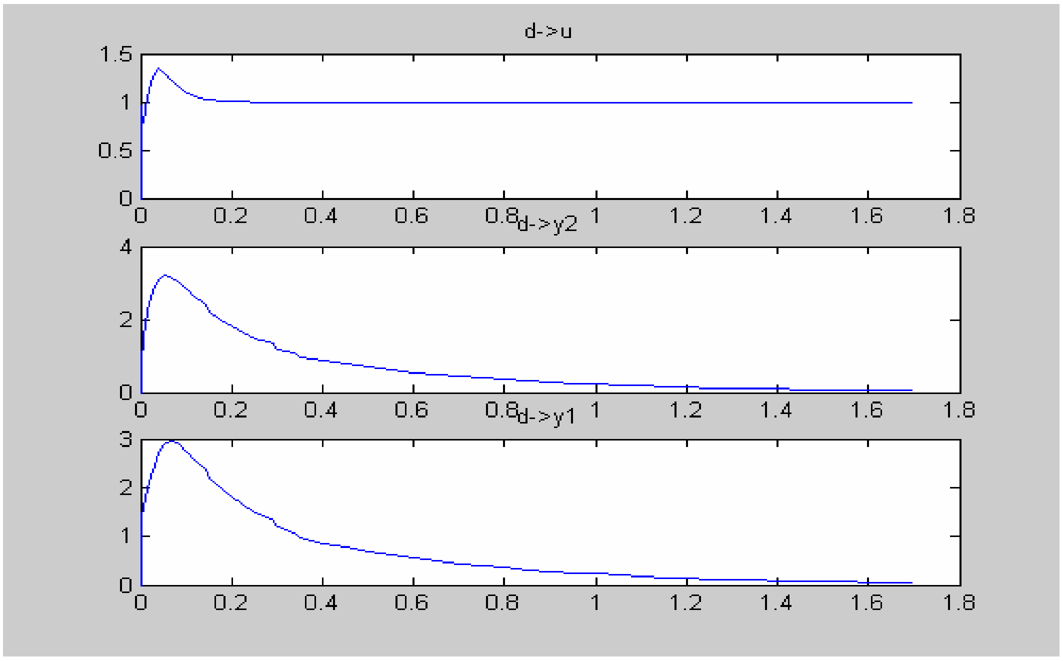


Figure 3-11: u, y2, y1 to the step input d. (h=0.05)

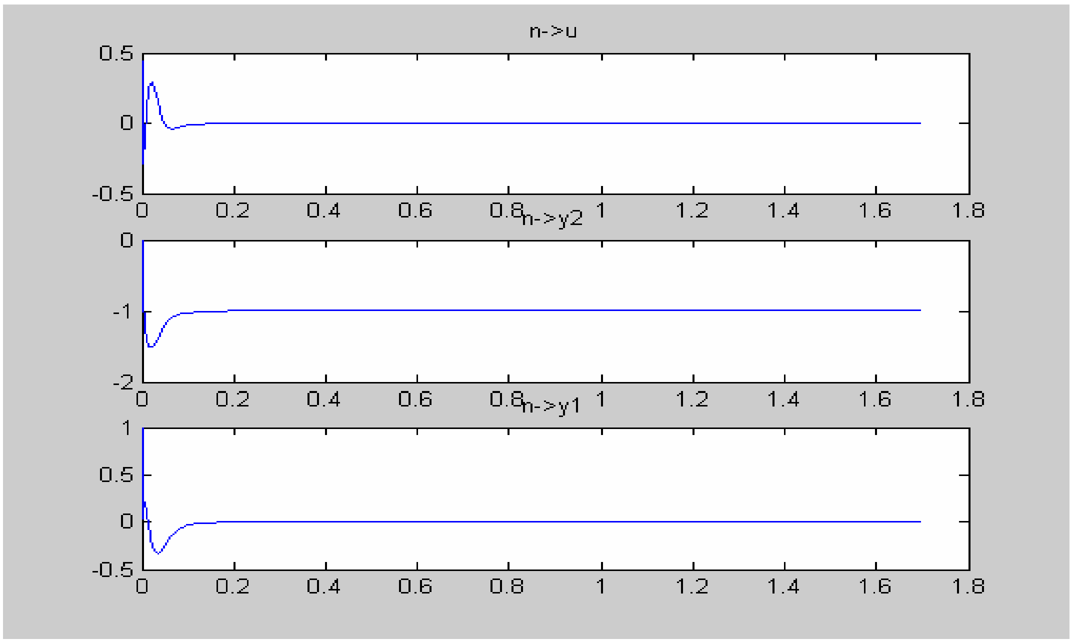


Figure 3-12: u, y2, y1 to step input n (h=0.05)

As we can notice from the above results, the setting time for u and y2, y1 to the process disturbance d are much longer.

3.2.3 Optimization based design

Now, we will explain how to formulate the Linear Programming for mass-spring process to calculate the controller, Q parameterization will be used in the design since this process is a potentially unstable process which can be shown in the following figure:

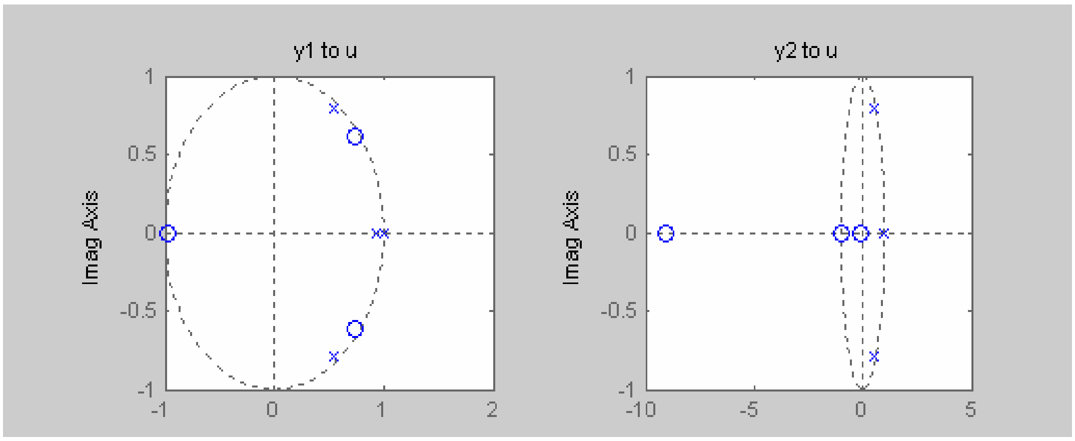


Figure 3-13: pole and zero maps for mass-spring process

As we know, y2- the position of the second mass is the variable which we want to control, y1- the position of the first mass is the variable we can measure, then we formulate the following closed loop transfer matrix to calculate the Q parameter using the Q parameterization. This system has three inputs and four outputs.

$$\begin{bmatrix} u \\ y2 \\ y1 \\ y1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ p2 & 0 & -p2 \\ p1 & 1 & -p1 \\ p1 & 1 & -p1 \end{bmatrix} \begin{bmatrix} d \\ n \\ u \end{bmatrix} \qquad (3\text{-}15)$$

Assuming Z= $\begin{bmatrix} u \\ y2 \\ y1 \end{bmatrix}$ and W= $\begin{bmatrix} d \\ n \end{bmatrix}$, G11= $\begin{bmatrix} 0 & 0 \\ p2 & 0 \\ p1 & 1 \end{bmatrix}$, G12= $\begin{bmatrix} 1 \\ -p2 \\ -p1 \end{bmatrix}$, G21= $[p1 \quad 1]$, G22= -p1

Use the developed Matlab Computer ToolBox, Q will be calculated, use (2-12) to (2-14), the optimized controller from Q parameterization can be calculated. The result of the system response has the following properties:

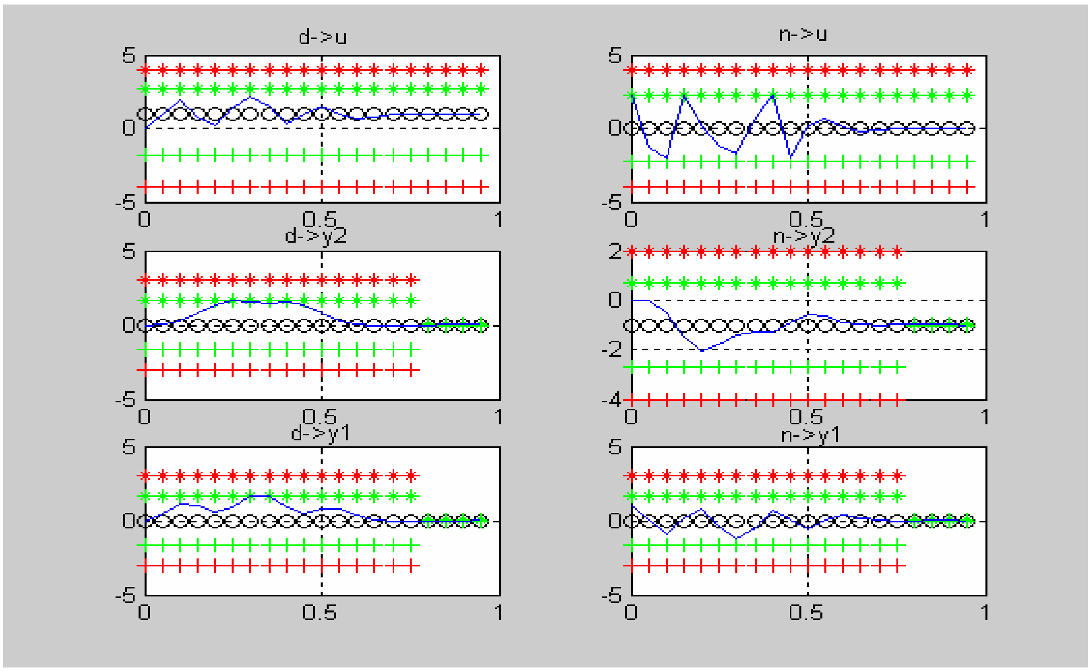


Figure 3-14: responses of the closed loop system to the step inputs d and n

The controller obtained from the optimization based design is as follows, it is a 12th order controller and there is an integrator has been included in the design. Q is in order 8.

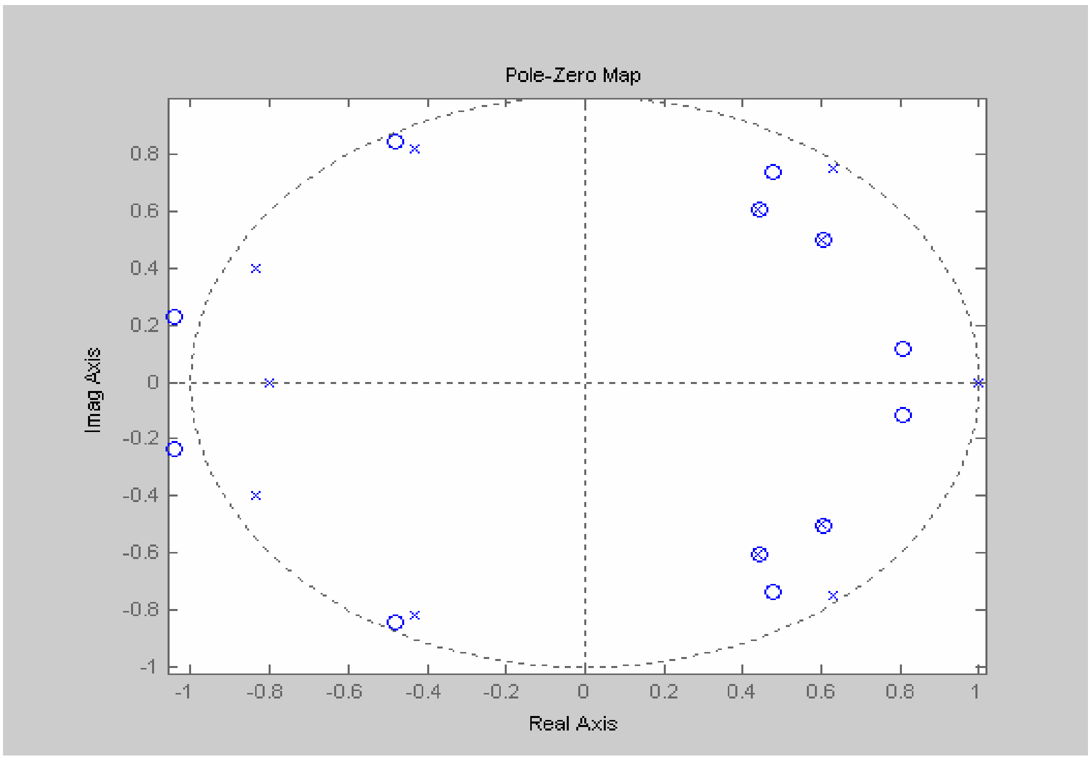


Figure 3-15: the poles and zeros of the controller calculated from optimization based design

The following plots are the simulation results obtained from the optimization-based design.

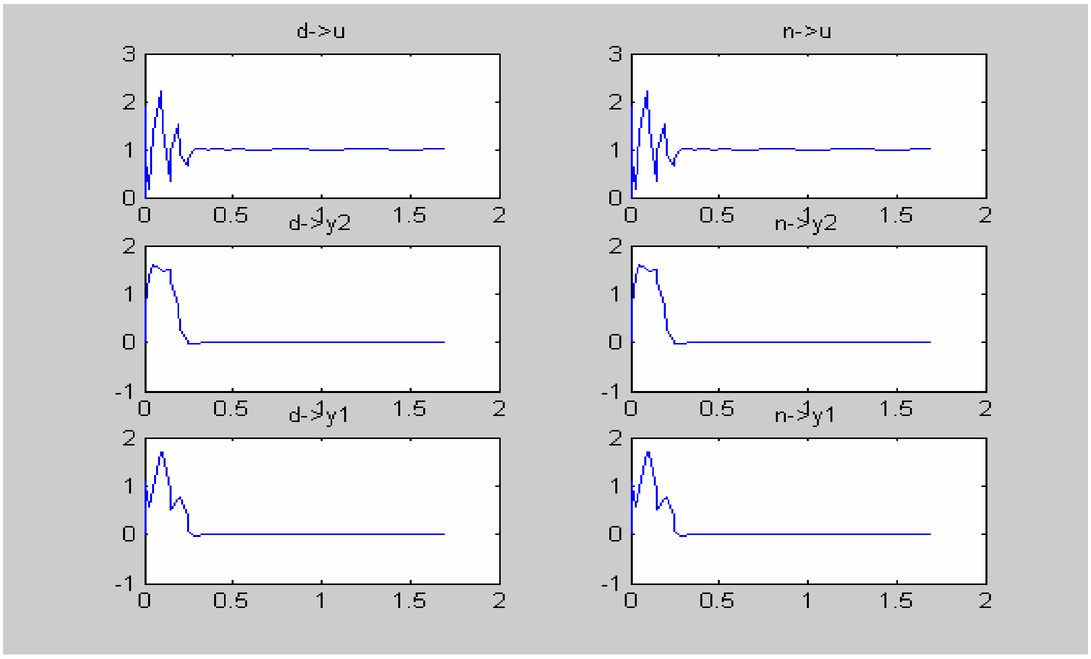


Figure 3-16: the responses of u, y2,y1 to step disturbance d and sensor noise n.

Compared with state feedback design, optimization-based design gives shorter setting time. From Figure 3-14, it also can be seen that the step response from n to u, d to y1, d to y2 reach critical bounds. This demonstrates which performance requirement is easier to be satisfied, and which one is not. If the limits of control action can be loosen, better output disturbance rejection can be achieved. In this sense, optimization-based design gives quantitative description on performance tradeoffs and limitations.

Compared to state feedback results, control action in figure 3-14 has more oscillations. The reason is that higher order controller in the optimization-based design gives potentially higher bandwidth to the closed-loop system. To reduce the oscillations in the output, one suggestion is to model the actuator as a low pass system, which means the process model has to be changed.

4. Bounds Definition

4.1 Bounds structure:

The bounds should be defined by the user. It is a critical parameter that will help us to get the correct optimized solution. The Matlab function STRUCT will help to form the structure of the bounds. The bounds structure is as follows:

B. T: the counter of the bounds, T*h (sampling time) equals the time instant(s) at which the bounds defined.

B. bound: the values that limit the system responses.

B. desired: the desired value that the system should converge to.

B. *output*: integer value, the row of the output signal in vector Z.

B. input: integer value, the row of the input signal in vector W.

4.2 bounds define instruction for the user

4.2.1 Generate bounds from performance specifications

For users, it is a little difficult to define the bounds if they are not aware how the system response will behave, improper definition of the bounds will not get the optimization solution or even the optimization will terminated successfully, the value of the gamma will be too large, which means we need to change the value of the bounds such that the designed specification be fulfilled. To solve this problem, performance specification will be used to be the references for the bounds define .Then our question has becomes how to automatically generate bounds from step response performance specification?

4.2.2 Parameters in performance specification

The performance specifications will be used as the tool to give the appropriate bounds value.

<u>For rejection:</u>

The parameters normally include: Limit, setTime, steadyError, (normally desired=0)

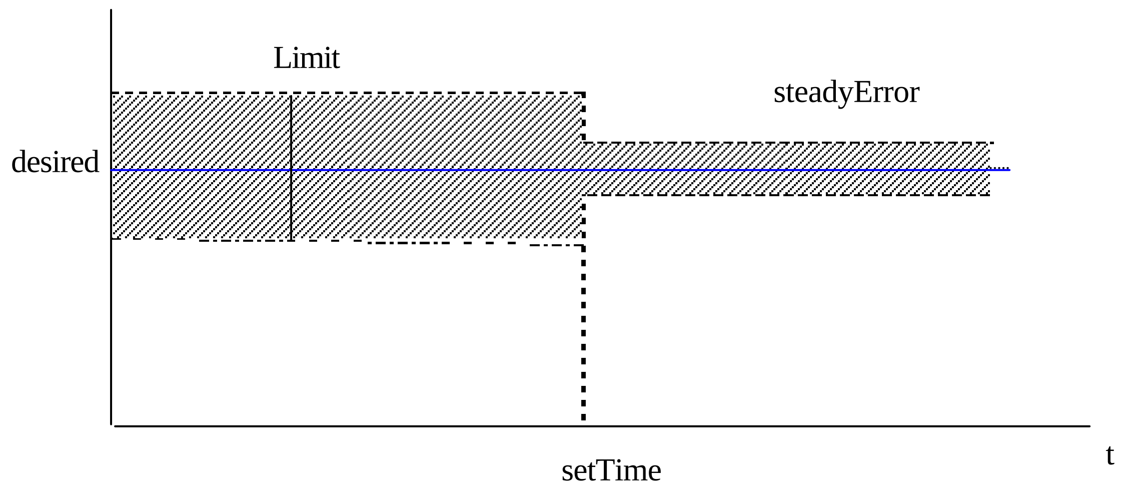


Figure 4-1: rejection response for bounds definition

<u>For control action or controller output:</u>

Only need to define the upper limit and lower limit, in order for the controller output not saturated.

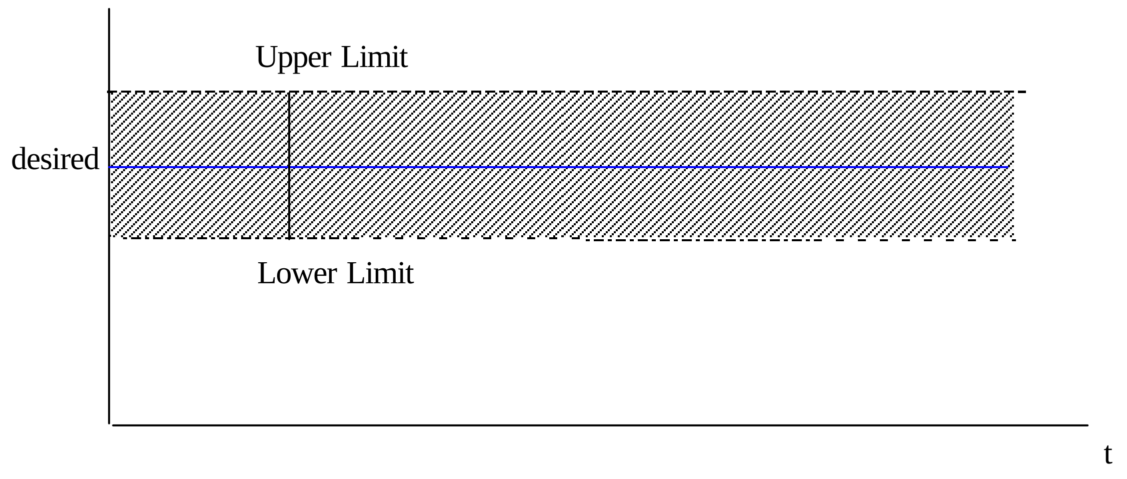


Figure 4-2: control action for bounds definition

5. Conclusions and Future work

5.1 Conclusions

The research area of this thesis project is on the optimization-based control design. Compared to other modern control design theories like LQR/LQG, optimization based design produces the so called numeric controller. In the thesis, we emphasized on the formulation of a general design problem into an optimization problem.

A disturbance rejection problem for a LTI plant under step inputs is then studied. The design goal is specified as step response setting time and steady state error etc. Derivation from original performance specification to the final linear programming form is performed and implemented as a set of Matlab functions.

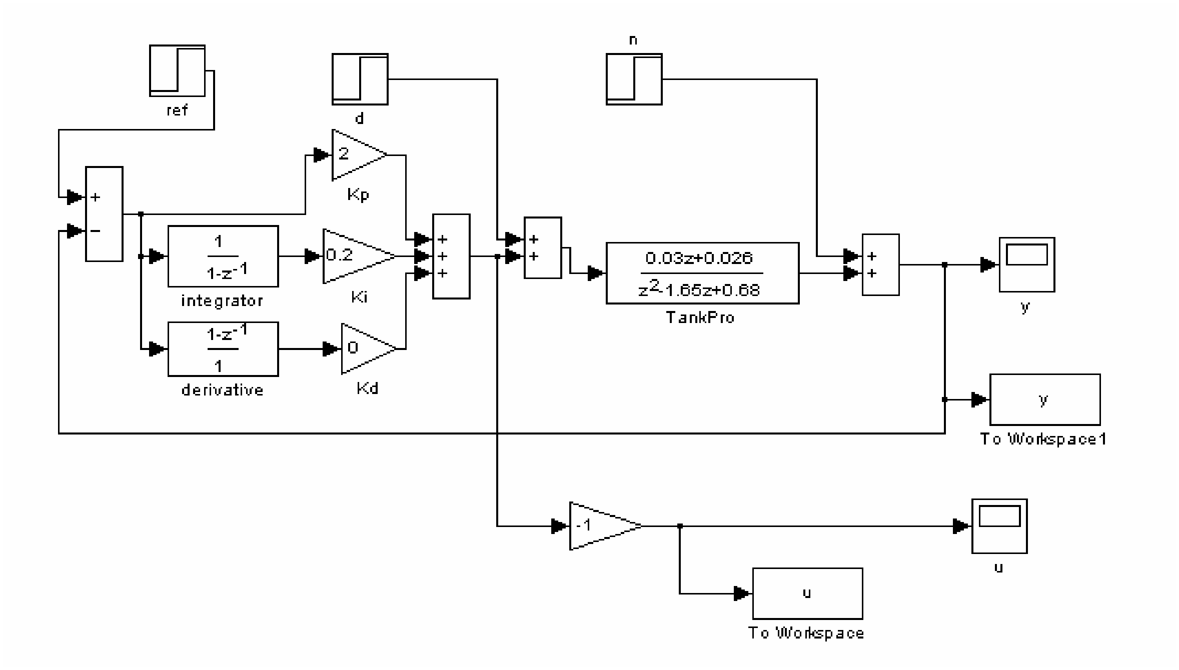
These Matlab functions have been applied for two real applications: double tank process and mass-spring system. Simulation model have been built to verify the correctness of the optimization formulation. Comparison is also conducted among optimization based control design, output feedback design and PID design for these two applications. The advantage of optimization based design is that it is quite easy to see the limits of the performance, and the tradeoffs between performance requirements. The disadvantage is that the designed controller often has very high order. So it is very important to perform model reduction.

5.2 Future Work

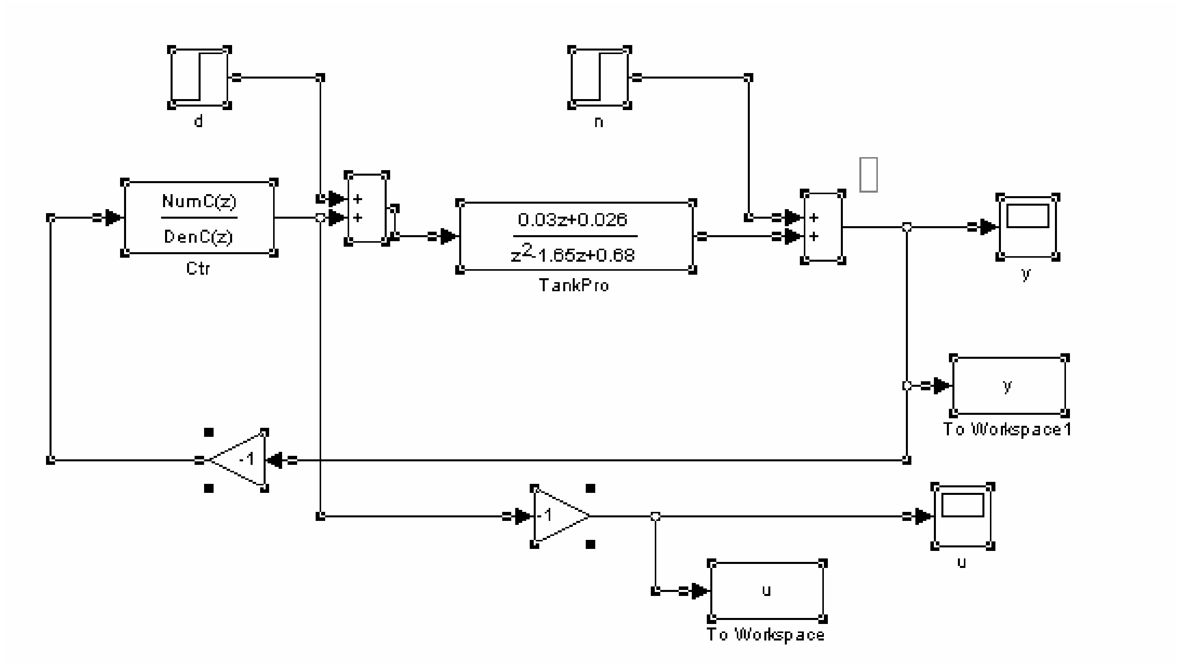
There are many things need to be improved. Due to the time limit, this thesis work only make the design for single input and single output system. The future work will of course to continue on the development of Matlab toolbox for MIMO system. The Q will be in matrix form, In this case, the algorithm will become more complex. And the controller order reduction should also be further studied.

6. Appendix

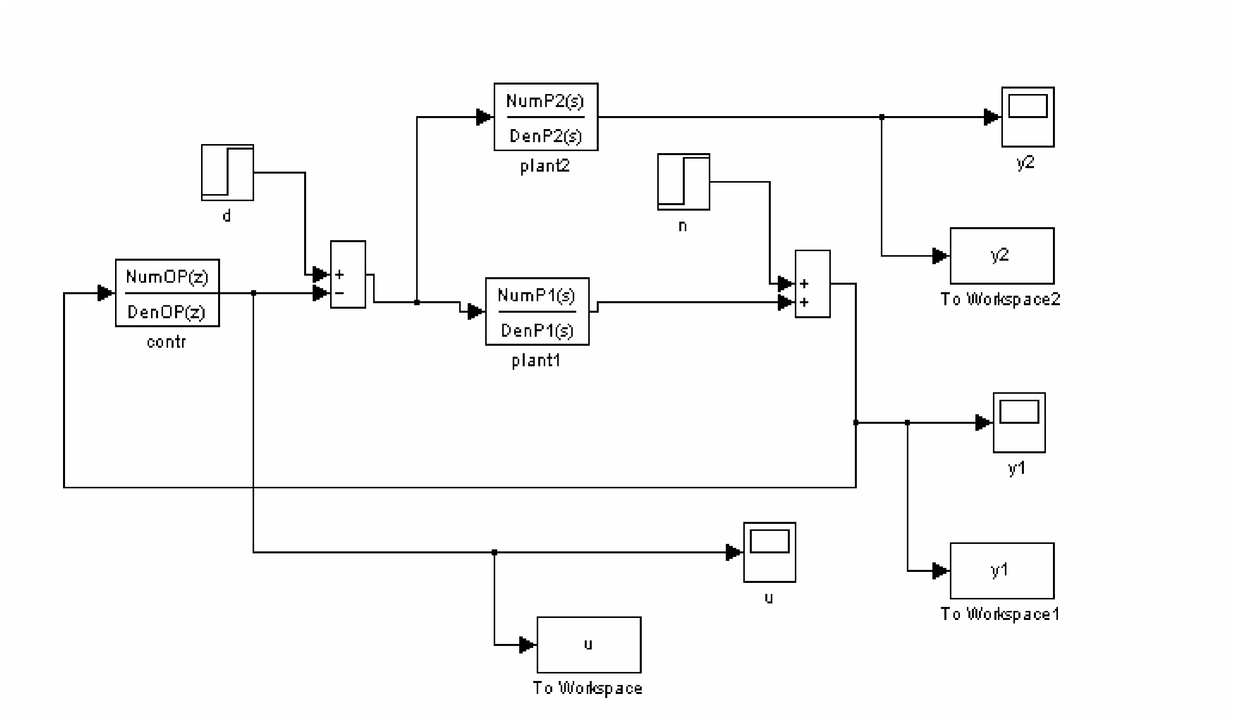
6.1.1 Simulation model for Double Tank process using PID controller



6.1.2 Simulation model for Double Tank process using optimization based controller



6.1.3 Simulation mode for mass-spring process using state feedback and integrator controller and optimization based controller



6.2 Matlab code for Computer Tool Box for Optimization based design.

6.2.1 List of the code

TestTank: design for the double tank process using Q parameterization.

TestMass: design for the mass-spring system using Q parameterization

qparlp: Get the Q parameters and calculate the controller

senslp: controller design using linear programming method

formlp: Linear Programming algorithm

getbounds: get the bounds value defined by users

plotres: plot responses of the closed loop system to the step inputs

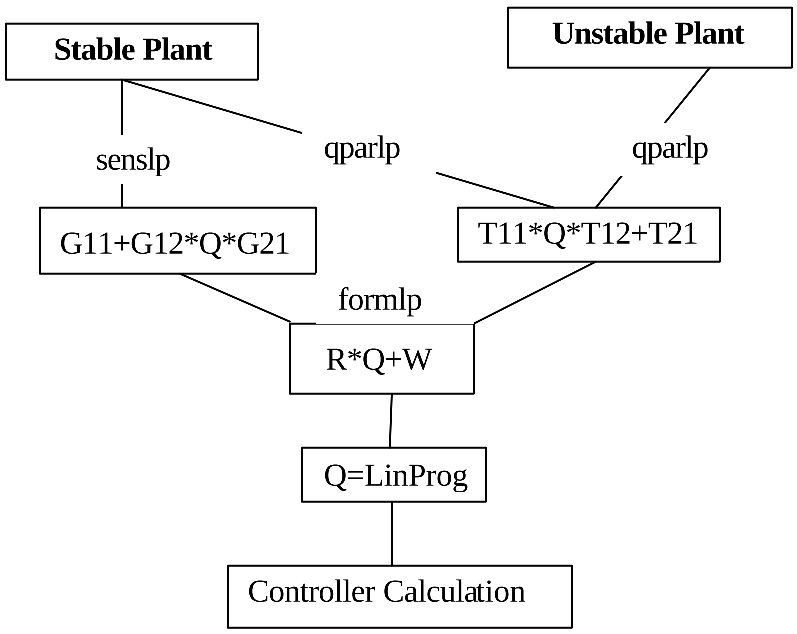
Actionbounds: define the bounds for the control signal to step input

Rejectionbounds: define the bounds for the measurement to step input

plotfigureTank: plot the results for simulink model of double tank process

plotfigureMass: plot the results for the simulink model of mass-spring system

6.2.2. Structure of the programming



6.3.2 Matlab Code

<u>TestTank</u>

```
clear all

% define the sampling time
h=1;

% stable double tank process
sys=tf([0.03 0.026],[1 -1.65 0.68],h); % discrete system
sysc=tf([0.0002456 0.06769],[1 0.3857 0.03627]);

% formulate the generalized open-loop plant
G11=[0 0; sys 1];
G12=[1; -sys];
```

```
G21=[sys 1];
G22=-sys;
G=[G11 G12;G21 G22];

% specify the bounds for closed loop system step response
B=[];
nsamples=50;
%nsamples=100;
Tf=(nsamples-1)*h;
tbnds=(0:nsamples-1);

% from input 1 to output1 (d to u)
% B=[B actionBounds(1,1,h,Tf,1,4,-2)];
ubnds=2*ones(1,50);
% lbnds=-0.5*ones(1,50);
for k=1:length(tbnds),
  B=[B struct('T',tbnds(k),'bound',ubnds(k),'desired',1,'output',1,'input',1)];
%  B=[B struct('T',tbnds(k),'bound',lbnds(k),'desired',1,'output',1,'input',1)];
end

% from input2 to output1 (n to u)
%B=[B actionBounds(1,2,h,Tf,0,3,-3)];
ubnds=1.5*ones(1,50);
lbnds=-0.5*ones(1,50);
for k=1:length(tbnds),
  B=[B struct('T',tbnds(k),'bound',ubnds(k),'desired',0.5,'output',1,'input',2)];
  B=[B struct('T',tbnds(k),'bound',lbnds(k),'desired',0.5,'output',1,'input',2)];
end

% define settle time
setTime=25;
```

```
% define steady Error
steadyError=0.01;

% get the couter for the whole bounds interval
Nf=ceil(Tf/h);
% get the counter for the bounds after the settling time
Ns=floor(setTime/h);

% from input1 to output2 (d to y)
%B=[B rejectBounds(2,1,h,Tf,0,1,15,0.01)];
%B=[B rejectBounds(2,1,h,Tf,0,1,20,0.01)];
for k=1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', 1, 'desired', 0, 'output', 2, 'input', 1);
    lB{k}=struct('T',tbnds(k),'bound', -1,'desired', 0, 'output', 2, 'input', 1);
end

for k=Ns+1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', steadyError, 'desired', 0, 'output', 2, 'input', 1);
    lB{k}=struct('T',tbnds(k),'bound', -steadyError, 'desired', 0, 'output', 2, 'input', 1);
end

for k=1:Nf+1
     B=[B uB{k} lB{k}];
end

%from input2 to output2 (n to y)
%B=[B rejectBounds(2,2,h,Tf,0,2,15,0.01)];
%B=[B rejectBounds(2,2,h,Tf,0,2,20,0.01)];
for k=1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', 2, 'desired', 0, 'output', 2, 'input', 2);
    lB{k}=struct('T',tbnds(k),'bound', -2,'desired', 0, 'output', 2, 'input', 2);
end

for k=Ns+1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', steadyError, 'desired', 0, 'output', 2, 'input', 2);
    lB{k}=struct('T',tbnds(k),'bound', -steadyError, 'desired', 0, 'output', 2, 'input', 2);
end

for k=1:Nf+1
    B=[B uB{k} lB{k}];
end



% use Q-parametrization to do optimal design
[contr,closeL,gama,AF,BF,Q,Ac,bc]=qparlp(G,B,15); % Q-parametrization based design
plotres(closeL,B,gama,Tf);
gama

% get the data from controller for simulation purpose
[NumC1,DenC1]=tfdata(contr);
NumC=NumC1{1};
DenC=DenC1{1};
```

TestMass

```
clear all

% system modeling
m1=2.29;
m2=2.044;
d1=3.12;
d2=3.73;
```

```
k=400;
km=2.96;
ky1=280;
ky2=280;
h=0.05;

A=[0 1 0 0; -k/m1 -d1/m1 k/m1 0; 0 0 0 1; k/m2 0 -k/m2 -d2/m2];
B=[0 km/m1 0 0]';
C1=[ky1 0 0 0];
C2=[0 0 ky2 0];
D=0;

% get the state space form from u to y1
G1=ss(A,B,C1,D);
% get the transfer function form from u to y1
Gf1=tf(G1);
% get the discrete form of the Gf1
Hf1=c2d(Gf1,h);
% get the state space form from u to y2
G2=ss(A,B,C2,D);
% get the transfer function form from u to y2
Gf2=tf(G2);
% get the discrete form of the Gf2
Hf2=c2d(Gf2,h);

% formulate the generalized open-loop plant
G11=[0 0; Hf2 0; Hf1 1];
G12=[1; -Hf2; -Hf1];
G21=[Hf1 1];
G22=-Hf1;
G=[G11 G12;G21 G22];
```

```
% specify the bounds for closed loop system step response
B=[];
nsamples=20;
Tf=(nsamples-1)*h;
Nf=nsamples-1;
tbnds=0:nsamples-1;
% from input 1 to output1
%B=[B actionbounds(1,1,h,Tf,1,4,-4)];
ubnds=4*ones(1,nsamples);
lbnds=-4*ones(1,nsamples);
for k=1:Nf+1
  B=[B struct('T',tbnds(k),'bound',ubnds(k),'desired',1,'output',1,'input',1)];
  B=[B struct('T',tbnds(k),'bound',lbnds(k),'desired',1,'output',1,'input',1)];
end

% from input2 to output1
%B=[B actionbounds(1,2,h,Tf,0,4,-4)];
ubnds=4*ones(1,nsamples);
lbnds=-4*ones(1,nsamples);
for k=1:Nf+1
  B=[B struct('T',tbnds(k),'bound',ubnds(k),'desired',0,'output',1,'input',2)];
  B=[B struct('T',tbnds(k),'bound', lbnds(k),'desired',0,'output',1,'input',2)];
end

% define the settling time
setTime=0.8;

% get the counter for the bounds after the settling time
Ns=floor(setTime/h);
```

```
% define the steadyError after the settling time
steadyError=0.01;

% from input1 to output2
%B=[B rejectbounds(2,1,h,Tf,0,3,0.8,0.01)];
for k=1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', 3, 'desired', 0, 'output', 2, 'input', 1);
    lB{k}=struct('T',tbnds(k),'bound', -3, 'desired', 0, 'output', 2, 'input', 1);
end

for k=Ns+1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', steadyError, 'desired', 0, 'output', 2, 'input', 1);
    lB{k}=struct('T',tbnds(k),'bound', -steadyError, 'desired', 0, 'output', 2, 'input', 1);
end

for k=1:Nf+1
     B=[B uB{k} lB{k}];
end

% from input2 to output2
%B=[B rejectbounds(2,2,h,Tf,-1,3,0.8,0.01)];
for k=1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', 2, 'desired', -1, 'output', 2, 'input', 2);
    lB{k}=struct('T',tbnds(k),'bound', -4,'desired', -1, 'output', 2, 'input', 2);
end

for k=Ns+1:Nf+1
    uB{k}=struct('T',tbnds(k),'bound', -1+steadyError, 'desired', -1, 'output', 2, 'input', 2);
    lB{k}=struct('T',tbnds(k),'bound', -1-steadyError, 'desired',-1, 'output', 2, 'input', 2);
end
```

```
for k=1:Nf+1
    B=[B uB{k} lB{k}];
end



% from input1 to output3
%B=[B rejectbounds(3,1,h,Tf,0,3,0.8,0.01)];
for k=1:Nf+1,
   uB{k}=struct('T',tbnds(k),'bound', 3, 'desired', 0, 'output', 3, 'input', 1);
   lB{k}=struct('T',tbnds(k),'bound', -3,'desired', 0,'output', 3, 'input', 1);
end

for k=Ns+1:Nf+1
   uB{k}=struct('T',tbnds(k),'bound', steadyError, 'desired', 0, 'output', 3, 'input', 1);
   lB{k}=struct('T',tbnds(k),'bound', -steadyError, 'desired', 0, 'output', 3, 'input', 1);
end

for k=1:Nf+1
    B=[B uB{k} lB{k}];
end

% from input2 to output3
%B=[B rejectbounds(3,2,h,Tf,0,3,0.8,0.01)];
for k=1:Nf+1,
   uB{k}=struct('T',tbnds(k),'bound', 3, 'desired', 0, 'output', 3, 'input', 2);
   lB{k}=struct('T',tbnds(k),'bound', -3,'desired', 0, 'output', 3, 'input', 2);
end

for k=Ns+1:Nf+1
   uB{k}=struct('T',tbnds(k),'bound', steadyError, 'desired', 0, 'output', 3, 'input', 2);
   lB{k}=struct('T',tbnds(k),'bound', -steadyError, 'desired', 0, 'output', 3, 'input', 2);
```

```
end

for k=1:Nf+1,
    B=[B uB{k} lB{k}];
end

% Q-parameterization based-design
[contr,closeL,gama,AF,BF,Q,Ac,bc]=qparlp(G,B,8);
plotres(closeL,B,gama,Tf);
gama

% get the data from u to y1
[Num1P,Den1P]=tfdata(Gf1);
NumP1=Num1P{1};
DenP1=Den1P{1};

% get the data from u to y2
[Num2P,Den2P]=tfdata(Gf2);
NumP2=Num2P{1};
DenP2=Den2P{1};

% get the data from the controller
[NumOP1,DenOP1]=tfdata(contr);
NumOP=NumOP1{1};
DenOP=DenOP1{1};
```

qparlp

```
%   Controller design using linear programming method and Q-parameterization.
%   G is the generalized open loop system as
%    [z]  [G11 G12][w]
%    [ ]= [       ][ ]   where w is disturbance, u is the controller output, z is controlled variable
%    [y]  [G21 G22][u]         y is the measurement for the controller input.
%    w has width of nw, u has width of nu, z has width of nz, y has width of ny.
%   Q parameterization tells that the closed loop system having the linear form of Q:
T11+T12*Q*T21,
%   where Q is any stable system.
%   This program ONLY deal with the case where Q is SISO, which means nu=1 and ny=1
%   The performance is specified as the step response bounds of the closed loop system, as
defined in
%   structure B.
%   m is the order of Q parameter. Q=q0+q1*z^(-1)+...+qm*z^(-m)
%   For the outputs:
%   contr---the designed controller
%   closeL--closed loop system
%   gama----how the bounds are satisfied
%   AF,BF,Q---LP problem parameter: AF*Q<BF
%   Ac,bc ----LP parameter for each input-output pair. Ac{i}{j} b{i}{j} are A and b from jth-
input to ith-output

function [contr,closeL,gama,AF,BF,Q,Ac,bc]=qparlp(G,B,m)
% get the size of G
[rT,cT]=size(G);
nz=rT-1;
nw=cT-1;

% call function dQpar, get the stable T
[T,A,L,K,B2,C2]=dQparState(G,nz,nw);
% to get T11, T12,T21 to formulate LP problem
T11=T(1:nz,1:nw);
T12=T(1:nz,(nw+1):cT);
T21=T((nz+1):rT,1:nw);
```

```
% closed loop transfer function Z=(T11+T12*Q*T21)*W

% to formulate the LP problem for each component of w to each component of z
AF=[]; BF=[];
for i=1:nz,
   for j=1:nw,
      [ubt,desU,ubs,lbt,desL,lbs]=getBounds(B,i,j);
      e1=zeros(1,nz); e1(i)=1;
      e2=zeros(nw,1); e2(j)=1;
      R=e1*T12*T21*e2;
      W=e1*T11*e2;
      [Ac{i}{j},bc{i}{j}]=formLP(m,R,W,ubt,desU,ubs,lbt,desL,lbs);
      AF=[AF;Ac{i}{j}];
      BF=[BF;bc{i}{j}];
   end
end

% get the size of AF
[r,c]=size(AF);

% to give the objective function, and to minimize gama
f=zeros(c,1);
f(c)=1;

% to solve the LP problem to get the minimized gama
QT=linprog(f,AF,BF,[],[]);

% to get the gama
gama=QT(m+2);

% to get the Q polynomials
```

```
Q=QT(1:(m+1));

% get the transfer function of Q
tmpNum=Q(1:m+1);
tmpDen=zeros(m+1,1);
tmpDen(1)=1;
QF=tf(tmpNum', tmpDen',G.ts);
closeL=T11+T12*T21*QF;

% get the state space form of controller from Q parameterization
% get state space form of the Q
[NumQ, DenQ]=tfdata(QF,'v');
[AQ,BQ,CQ,DQ]=tf2ss(NumQ,DenQ);

AAC=[A-B2*L-B2*DQ*C2-K*C2 B2*CQ; -BQ*C2 AQ];
BBC=[B2*DQ+K; BQ];
CCC=[-DQ*C2-L CQ];
DDC=DQ;

contr=ss(AAC,BBC,CCC,DDC,G.ts);
```

<u>senslp</u>

```
%  Control design using linear programming method.
%  The closed-loop system is formulated as linear form of Q
%  G is the generalized open loop system as
%    [z]  [G11 G12][w]
%    [ ]= [        ][ ]   where w is disturbance, u is the controller output, z is controlled variable
%    [y]  [G21 G22][u]    y is the measurement for the controller input.
%   w has width of nw, u has width of nu, z has width of nz, y has width of ny.
%   The controller is u=Cy, the closed loop system is then
%    G11+G12*(I-C*G22)^(-1)*C*G21
%   A simplest case is the regulation of a SISO plant P under the input disturbance and noise:
%   [ u ]   [ QP      Q   ]  [ d ]
%   [   ] = [             ]  [   ]
%   [ y ]   [ P-QPP  1-QP ]  [ n ]
%
%  If G is stable, let Q=(I-C*G22)^(-1)*C, we can formulate LP based on this Q
%  Q=q0+q1*z^(-1)+...+qm*z^(-1)
%  m---order of Q (terms taken from z^-1 series)
%  B---structure for the bounds definition.
%
%   For the outputs:
%    contr---the designed controller
%    closeL--closed loop system
%    gama---- how the bounds are satisfied
%    AF,BF,Q---LP problem parameter: AF*Q<BF
%    Ac,bc ---- LP parameter for each input-output pair
%
%   Limitation: this program ONLY deals with the case where controller is SISO and the plant is
stable

function [contr,closeL,gama,AF,BF,Q,Ac,bc]=sensLP(G,B,m)
% get the size of G
[rT,cT]=size(G);
nz=rT-1; % control input is only 1-dimensional
nw=cT-1; % control output is 1-dimensional

G11=G(1:nz,1:nw);
G12=G(1:nz,(nw+1):cT);
G21=G((nz+1):rT,1:nw);
G22=G((nz+1):rT,(nw+1):cT);
```

```
% closed loop transfer function Z=(G11+G12*Q*G21)*W

% to formulate the LP problem for each component of w to each component of z
AF=[]; BF=[];
for i=1:nz,
   for j=1:nw,
      [ubt,desU,ubs,lbt,desL,lbs]=getBounds(B,i,j);
      e1=zeros(1,nz); e1(i)=1;
      e2=zeros(nw,1); e2(j)=1;
      R=e1*G12*G21*e2;
      W=e1*G11*e2;
      [Ac{i}{j},bc{i}{j}]=formLP(m,R,W,ubt,desU,ubs,lbt,desL, lbs);
      AF=[AF;Ac{i}{j}];
      BF=[BF;bc{i}{j}];
   end
end

% get the size of AF
[r,c]=size(AF);

% to give the objective function, and to minimize gama
f=zeros(c,1);
f(c)=1;

% to solve the LP problem to get the minimized gama
QT=linprog(f,AF,BF,[],[]);

% to get the gama
gama=QT(m+2);

% to get the Q polynomials
```

```
Q=QT(1:(m+1));

% get the transfer function of Q
tmpNum=Q(1:m+1);
tmpDen=zeros(m+1,1);
tmpDen(1)=1;
QF=tf(tmpNum', tmpDen',G.ts);
closeL=G11+G12*G21*QF;

% get the controller from Q,Q=(I-C*G22)^(-1)*C, C=(G22*Q+I)^(-1)*Q
contr=QF/(G22*QF+1);
```

formlp

```
% [Af,bf]=formLP(m,R,W)
%  formulate LP problem Ax<b for linear form H=RQ+W as defined in the following
%  Q=q0+q1*z^(-1)+q2*z^(-2)+...+qm*z^(-m), where q0,q1,...,qm to be determined,
%  m is the max order of z^(-1)
%  R is SISO discrete system
%  W is SISO discrete system
%  the design parameter x=[q0 q1 ... qm]'.
%  the controller performance requirement is defined as the bounds of H's step response
%  desU,ubs,ubt---define upper bounds values, ubt is the sampling counter at which the desired
value
%  desU and upper bound ubs are specified for H's step response
%  desL,lbs,lbt---define lower bounds values, lbt is the sampling counter at which the desired
value
%  desL and lower bound lbs are specified for H's step response
%  Note:
%   (1) ubt,lbt are NOT time points, but sampling counters. That is, the real time for ubt is
ubt*R.ts
```

```
function [AfL,bfL,Af,bf]=formLP(m,R,W,ubt,desU,ubs,lbt,desL,lbs)

n1=length(ubt);
n2=length(lbt);
n=length(ubs)+length(lbs);
newN=max([ubt lbt]); % find out what is the max samping counter at which the bound is defined

% convert R and W into FIR form.
RI=impulse(R,newN*R.ts);
WI=impulse(W,newN*W.ts);

% define GG, H=Gq+W;
GG=zeros(newN+1,m+1);
for i=1:m+1
   GG(i:newN+1,i)=RI(1:(newN+1-i+1));
end

% formulate A and b, blt: upper bounds, bgt: lower bounds
AA=zeros(newN+1,newN+1);
for i=1:newN+1
   for j=1:i
      AA(i,j)=1;
   end
end

% row i of AA corresponds to sampling counter-1. For instance, row 1 is defined for sample 0
A=zeros(n,newN+1);
b=zeros(n,1);
des=zeros(n,1);

% get A and b for the upper bounds
```

```
for k=1:n1,
    tU=ubt(k);
    A(k,:)=AA(tU+1,:);
    b(k)=ubs(k);
    des(k)=desU(k);
end
% get A and b for the lower bounds
for k=1:n2
    tL=lbt(k);
    A(n1+k,:)=-AA(tL+1,:);
    b(n1+k)=-lbs(k);
    des(n1+k)=-desL(k);
end

% to formulate final Af and bf
Af=A*G;
bf=b-A*WI;

% introduce the gama, convert the problem to A(Gq+w)-des<gam*(b-des),
% x=[q;gama]' then get the final AfL and bfL, and to solve the LP problem
AfL=[Af,des-b]; bfL=des-A*WI;
```

getbounds

```
% get the bounds which are produced by the users
% Ubt: the sampling counter at which the upper bounds defined.
% Lbt: the sampling counter at which the lower bounds defined.

function [Ubt,DesU,Ubs,Lbt,DesL,Lbs]=getBounds(B,out,in)
DesU=[];
DesL=[];
```

```
Ubs=[];
Lbs=[];
Ubt=[];
Lbt=[];
for k=1:length(B)
if B(k).output==out & B(k).input==in,
      if B(k).desired<B(k).bound,
          Ubs=[Ubs B(k).bound];
          Ubt=[Ubt B(k).T];
          DesU=[DesU B(k).desired];
      else
          Lbs=[Lbs B(k).bound];
          Lbt=[Lbt B(k).T];
          DesL=[DesL B(k).desired];
      end

   end
end
```

plotres

```
% plot the step response of the closed loop system
function plotRes(closeL,bnds,gama,Tf)
[nz,nw]=size(closeL);
ts=closeL.ts;
[Y,T] = step(closeL,Tf);
figure;
for i=1:nz,
   for j=1:nw,
      [ubt,desU,ubs,lbt,desL,lbs]=getBounds(bnds,i,j);
      subplot(nz,nw,(i-1)*nw+j);
      % plot bounds
```

```
        plot(ubt*ts,desU,'ko',ubt*ts,ubs,'r*',ubt*ts,(ubs-desU)*gama+desU,'g*');
        hold on;
        plot(lbt*ts,desL,'ko',lbt*ts,lbs,'r+',lbt*ts,(lbs-desL)*gama+desL,'g+');
        hold on;
        % plot step response
        plot(T,Y(:,i,j),'-'); title(['input ' num2str(j) ' to output ' num2str(i)]);
        hold off;
        grid;
    end
end
```

Actionbounds:

```
% actionBounds: given step response specification, generate bounds for the optimization
% sampTime: the sample time of the system
% finalTime: the period during which bounds are defined
% upperLimit, lowerLimit: limit value, absolute value
% performance specification:
% for control action: just simple the constant lower/upper bounds

function B=actionBounds(whichOut, whichIn, sampTime, finalTime, desired, upperLimit,
lowerLimit)

B=[];
Nf=ceil(finalTime/sampTime);
sampleIdx=0:Nf;

for k=1:length(sampleIdx),
    uB{k}=struct('T',sampleIdx(k),'bound',upperLimit, 'desired', desired, 'output', whichOut,
'input', whichIn);
    lB{k}=struct('T',sampleIdx(k),'bound',lowerLimit, 'desired', desired, 'output', whichOut, 'input',
whichIn);
```

```
end

for k=1:length(sampleIdx),
   B=[B uB{k} lB{k}];
end
```

rejectbounds

```
% rejectBounds: given step response specification, generate bounds for the optimization
% sampTime: the sample time of the system
% finalTime: the period during which bounds are defined, it must be larger than setTime
% limit: in relative value
% steadyError: in relative value
% performance specification:
% for rejection:

function B=rejectBounds(whichOut, whichIn, sampTime, finalTime, desired, limit, setTime,
steadyError)

B=[];
Nf=ceil(finalTime/sampTime);
sampleIdx=0:Nf;

for k=1:length(sampleIdx),
   uB{k}=struct('T',sampleIdx(k),'bound',desired+limit, 'desired', desired, 'output', whichOut,
'input', whichIn);
   lB{k}=struct('T',sampleIdx(k),'bound',desired-limit, 'desired', desired, 'output', whichOut,
'input', whichIn);
end

if nargin >=8,
   if setTime > finalTime,
        error('In defBounds, setTime > finalTime');
    else,
       Ns=floor(setTime/sampTime);
       for k=Ns+1:Nf+1,
           uB{k}=struct('T',sampleIdx(k),'bound',desired+steadyError, 'desired', desired, 'output',
whichOut, 'input', whichIn);
           lB{k}=struct('T',sampleIdx(k),'bound',desired-steadyError, 'desired', desired, 'output',
whichOut, 'input', whichIn);
       end
    end
end

for k=1:length(sampleIdx),
    B=[B uB{k} lB{k}];
end

plotfiguretank:
subplot(2,2,1);plot(tout(1:50),u1(1:50));title('d>u');
subplot(2,2,2);plot(tout(1:50),u2(1:50));title('n>u');
subplot(2,2,3);plot(tout(1:50),y1(1:50));title('d->y');
subplot(2,2,4);plot(tout(1:50),y2(1:50));title('n->y');

plotfiguremass:
subplot(3,2,1); plot(tout(1:50),ud(1:50));title('d>u');
subplot(3,2,3);plot(tout(1:50),yd2(1:50));title ('d>y2');
subplot(3,2,5);plot(tout(1:50),yd1(1:50));title('d>y1');
subplot(3,2,2);plot(tout(1:50),un(1:50));title('n>u');
subplot(3,2,4);plot(tout(1:50),yn2(1:50));title('n->y2');
subplot(3,2,6);plot(tout(1:50),yn1(1:50));title('n->y1');
```

6.3 References used in the Thesis

1. S. P. Boyd and G. H. Barratt, *Linear Controller Design Limits of Performance*, 1991, Prentice-Hall
2. K. J. Åström and B. Wittenmark, *Computer controlled system,* Prentice-Hall
3. A. Rantzer and B. Bernhardsson, “A Simple Proof of the Q parameterization”, March 1994
4. M. Grundelius, C*ontrol of Servo with Flexible Axle*, Department of Automatic Control, Lund Institute of Technology, Sep. 1999, revised by Sven Hedlund, June 2000.
5. V. Balakrishnan and A.L.Tits, Numerical Optimization-Based Design, *The Control Handbook*, CRC Press, 1996
6. J. Oishi and V. Balakrishnan, "Linear Controller Design for the NEC Laser Bonder via LMI Optimization," *Recent Advances on LMI Methods in Control,* El Ghaoui and S. Niculescu, (Eds.), Society for Industrial and Applied Mathematics (SIAM), 1999.