LabVIEW™

Control Design Toolkit User Manual

NATIONAL INSTRUMENTS™

September 2004 Edition
Part Number 371057B-01

Important Information

Contents

About This Manual

This manual contains information about the purpose of control design and the control design process. This manual also describes how to develop a control design system using the LabVIEW Control Design Toolkit.

This manual requires that you have a basic understanding of the LabVIEW environment. If you are unfamiliar with LabVIEW, refer to the *Getting Started with LabVIEW* manual before reading this manual.

Note This manual is not intended to provide a comprehensive discussion of control design theory. Refer to the following books for more information about control design theory: *Modern Control Systems*[1], *Feedback Control of Dynamic Systems*[2], *Digital Control of Dynamic Systems*[3], *Control Systems Engineering*[4], and *Modern Control Engineering*[5].

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

[1] Dorf, Richard C., and Robert H. Bishop. *Modern Control Systems*, 9th ed. Upper Saddle River, NJ: Prentice Hall, 2001.

[2] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[3] Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.

[4] Nise, Norman S. *Control Systems Engineering*, 3rd ed. New York: John Wiley & Sons, Inc., 2000.

[5] Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2001.

italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**
- *Getting Started with LabVIEW*
- *LabVIEW User Manual*
- *LabVIEW Simulation Module User Manual*
- *LabVIEW System Identification Toolkit User Manual*

1

Introduction to Control Design

Control design involves developing mathematical models that describe a physical system, analyzing the models to learn about their dynamic characteristics, and creating a controller to achieve certain dynamic characteristics.

Control systems in engineering contain an arrangement of physical components related in a way that commands, directs, or regulates the physical system. The controlled physical system also is known as the plant. In this manual, the control system refers to the sensors, the controller, and the actuators. The dynamic system refers to the control system and the plant, as shown in Figure 1-1.

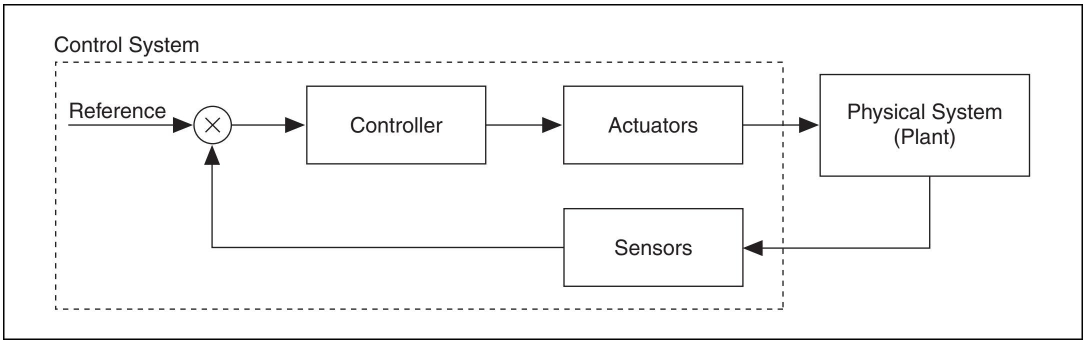


Figure 1-1. Dynamic System

The dynamic system in Figure 1-1 represents a closed-loop system. In a closed-loop system, the control system monitors the outputs of the plant and adjusts the inputs to the plant to make the actual response closer to the desired response. Closed-loop systems also are known as feedback systems.

To better understand a closed-loop system, consider a control system that regulates the temperature of a room. The thermometer reads the temperature of the room. Based on the desired temperature, the thermostat turns on the heater or the air conditioner. In this example, the room is the plant, the thermometer is the sensor, the thermostat is the controller, and the heater or air conditioner is the actuator.

Other common examples of control systems include the following:

- Automobile cruise control
- Robots in manufacturing
- Refrigerator temperature control system
- Hard drive head control

Model-Based Control Design Process

The model-based control design process involves plant modeling, analyzing and synthesizing a controller, simulating the system, and deploying the controller. While the LabVIEW Control Design Toolkit provides solutions for analyzing and synthesizing controllers, National Instruments also provides solutions for the other three components in the process, as shown in Figure 1-2.

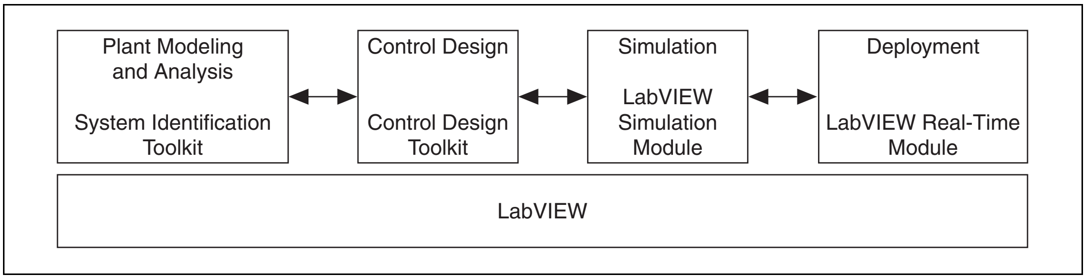


Figure 1-2. Plant Modeling, Control Design, Simulation, and Deployment

Analyzing and Modeling a Plant

In the initial phase of the model-based control design process, you must obtain a mathematical model of the plant you want to control. One way to obtain a model is by using a numerical process known as system identification. The system identification process involves acquiring data from a plant and then numerically analyzing stimulus and response data to estimate the parameters of the model.

The system identification process requires a combination of the following components:

- **Signal generation and data acquisition**—National Instruments provides software and hardware that you can use to stimulate and measure the response of the plant.
- **Mathematical modeling tools**—The LabVIEW System Identification Toolkit estimates and creates accurate mathematical models of dynamic systems. You can use the toolkit to create discrete, linear models of systems based on measured stimulus and response data.

Note This manual is not intended to provide a comprehensive discussion of modeling. Refer to the following books for more information about modeling: *Modern Control Systems*[1], *Feedback Control of Dynamic Systems*[2], *Digital Control of Dynamic Systems*[3], *Control Systems Engineering*[4], and *Modern Control Engineering*[5].

Designing a Controller

In the second phase of the model-based control design process, you analyze and synthesize a controller. The Control Design Toolkit provides a set of VIs for classical and modern linear control analysis and design techniques. With these VIs you can create and analyze linear time-invariant system models and design automatic control systems.

Using a model you identified in the plant modeling phase, identify the dynamic characteristics of the plant you want to control. Based on the dynamic characteristics of the plant, identify the actuators and sensors you want to use in the control system.

[1] Dorf, Richard C., and Robert H. Bishop. *Modern Control Systems*, 9th ed. Upper Saddle River, NJ: Prentice Hall, 2001.

[2] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[3] Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.

[4] Nise, Norman S. *Control Systems Engineering*, 3rd ed. New York: John Wiley & Sons, Inc., 2000.

[5] Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2001.

Analyze the dynamic behavior of the plant. Based on the dynamic behavior of the plant and/or control system, synthesize a controller based on the dynamic characteristics of the plant to achieve the desired performance criteria of the plant. Then, create a linear time-invariant model of the control system or the dynamic system. Use this model to analyze the dynamic response of the plant, controller, or entire closed-loop system to different stimuli. Figure 1-3 shows the major components involved in designing a controller.

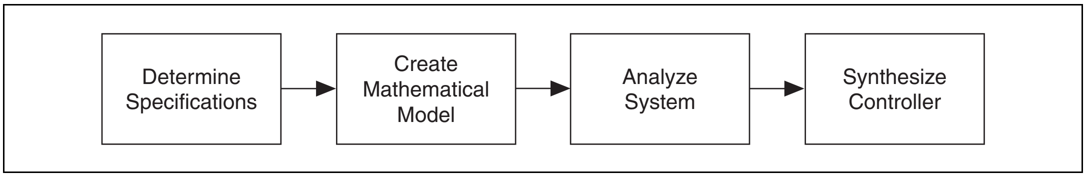


Figure 1-3. Control Design Process

You often iterate these steps to achieve an acceptable design that is physically realizable and meets the desired performance criteria.

Simulating the Dynamic System

In the third phase, you want to simulate the dynamic system. The LabVIEW Simulation Module allows you to simulate dynamic systems in LabVIEW. You can investigate the time response of the dynamic system to complex, time-varying inputs before deploying a controller. For this process, you can use a simple linear time-invariant model, a higher order model, or a nonlinear model of the plant.

Deploying the Controller

The last stage of the model-based control design process is to deploy the controller to a real-time target. LabVIEW and the LabVIEW Real-Time Module provide a common platform that you can use to implement the embedded control system.

National Instruments also provides products for I/O and signal conditioning that you can use to gather and process data. Using these tools, which are built on the LabVIEW platform, you can experiment with different approaches at each stage in the design process and quickly identify the optimal design solution for an embedded control system.

Refer to the National Instruments Web site at `ni.com` for information about these National Instruments products.

Designing a Controller with the Toolkit

The Control Design Toolkit provides a library of VIs and an assistant for designing a controller based on a model of a plant. Both tools enable you to complete the entire control design process from creating a model of the controller to synthesizing the controller.

Control Design Assistant

Without prior knowledge about programming in LabVIEW, you can use the Control Design Assistant to develop a model that reflects the behavior of a certain dynamic system. You access the Control Design Assistant through the NI Express Workbench. The Express Workbench is a new framework that can host multiple interactive National Instruments tools and assistants.

Using the Control Design Assistant, you can create a project that encompasses the whole control design process. In one project, you can load or create a model of a plant into the Control Design Assistant, analyze the time or frequency response, and then synthesize a controller. The Express Workbench provides windows in which you can immediately see the mathematical equation and graphical representation that describe the model. You also can view the response data and the configuration of the controller.

After creating this project in the Express Workbench, you can convert the project to a LabVIEW block diagram and customize the block diagram in LabVIEW. This conversion enables you to enhance the capabilities of the application. Refer to the *NI Express Workbench Help* for more information about using the Control Design Assistant to analyze models that describe a physical system and synthesize controllers to achieve certain dynamic characteristics.

Control Design VIs

The Control Design Toolkit also provides VIs that you can use to develop mathematical models, analyze the models to learn about their dynamic characteristics, and create controllers to achieve certain dynamic characteristics. The Model Construction VIs enable you to create models based on known parameters. After creating a model, you can use the Model Interconnection VIs to append multiple models or connect multiple models in series, in parallel, or in a feedback loop. The Model Conversion VIs enable you to change the representation of a model. The Time Response VIs and Frequency Response VIs analyze the response of the model. The State Feedback Design VIs and the State-Space Model Analysis VIs enable you to synthesize controllers based on modern control design techniques. Refer to the *LabVIEW Help* for information about the Control Design VIs.

These Control Design VIs enable you to customize a LabVIEW block diagram to achieve specific goals. You also can use other LabVIEW VIs and functions to enhance the functionality of the application. Unlike creating a project with the Control Design Assistant, creating a LabVIEW application using these VIs requires basic knowledge about programming in LabVIEW. Refer to the *LabVIEW User Manual* and the *Getting Started with LabVIEW* manual for more information about the LabVIEW programming environment.

2 Creating Dynamic System Models

Defining a mathematical model that captures the behavioral traits of a system is important in many engineering applications. A model describes the system you want to study and allows you to anticipate the behavior and analyze different properties of the system.

The purpose of modeling in control design theory is to build a succinct model that allows you to capture those features of the system you want to control. Therefore, a model needs to capture the minimum amount of information necessary to achieve a set of predefined goals. However, the model also must be accurate enough to allow you to build a controller for the system.

Most real-world systems include many different interacting components and are difficult to model from fundamental physical descriptions. You must consider many external factors such as temperature, altitude, and random interactions. You also must consider internal interacting structures and their fundamental descriptions. Such a general, accurate model is not only hard to build but provides very little insight about specific factors in the real-world system.

Usually you study a system with a specific purpose so you design a model that reflects that purpose. For example, in a mechanical system, if you want to control the interacting forces and friction, you do not need to include the thermodynamic effects of the system. These effects are complicated features of the system that do not directly affect the friction. If you create a model that incorporates these effects, the result is an unnecessarily complicated model.

This chapter provides information about modeling and then describes the three types of model representations—transfer function, zero-pole-gain, and state-space—you can create using the LabVIEW Control Design Toolkit.

Introduction to Modeling

To create a model of a system, conceptualize the system as a black box that continuously accepts inputs and continuously generates outputs. The control objectives are a function of these observed inputs and outputs. The system model needs to reflect the process of entering the inputs and generating the outputs. Figure 2-1 shows the basic black box model of a system.

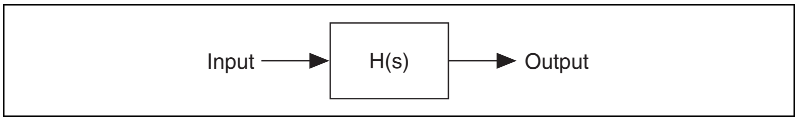


Figure 2-1. Black Box Model of a System

To make the model concrete, you must express the black box description of the system in a way that allows you to anticipate the system behavior and study control design properties. To achieve this goal, you must describe the relationship between the inputs and outputs of the system using mathematical expressions.

The model is a mathematical relationship that approximately captures the dynamics between the inputs and outputs of a system. This mathematical model also reveals properties of the system that are relevant to controlling it. The more accurate a model is, the more complex the mathematical relationship between inputs and outputs. However, such complexity might not be useful.

Real-world systems are subject to a variety of non-deterministic fluctuating conditions that prevent you from making an extremely detailed model. The usual control problem is to design a controller for a system and make the design robust enough to account for modeling inaccuracies. Inflexible designs might fail when you deploy the control system to a real-time target.

You can use physical laws or experimental data to develop a model. The laws of physics define the physical model of a system. The following sections describe various classifications and features of physical models.

Linear versus Nonlinear Models

A linear model obeys the principle of superposition. The following equations are true for linear models.

$$y_1 = f(x_1)$$

$$y_2 = f(x_2)$$

$$Y = f(x_1 + x_2) = y_1 + y_2$$

A differential equation is linear if the coefficients are constant or change only with the independent variable. Often this independent variable is time.

A nonlinear model does not obey the principle of superposition. Nonlinear effects in real-world systems include saturation, dead-zone, friction, backlash, and quantization effects; relays; switches; and rate limiters. All real-world systems are nonlinear, though you can linearize the model to simplify a design or analysis task.

Time-Variant versus Time-Invariant Models

Dynamic models are time-variant or time-invariant. If the parameters of a model do not change with time, the model is a time-invariant model. If the parameters change with time, the model is a time-variant model. For example, you can use a time-variant model to describe an automobile. As fuel burns, the mass of the vehicle changes with time.

Continuous versus Discrete Models

Dynamic models can be either continuous or discrete. Continuous models represent real-world signals that vary continuously with time. You can use differential equations to describe continuous systems. For example, a model that describes the orbital motion of a satellite is a continuous model. Discrete models represent signals that are sampled in time at discrete intervals. You can use difference equations to describe discrete systems. For example, a model that controls the altitude of the satellite is a discrete model. In either case, the equations that describe the system can be linear or nonlinear and time-invariant or time-variant.

LabVIEW Control Design Toolkit Model Representations

The Control Design Toolkit provides tools to study the dynamics of systems described by linear time-invariant continuous and discrete models. These system models take one of the forms shown in Table 2-1. You can use these representations to describe both single-input single-output (SISO) and multi-input multi-output (MIMO) systems.

Note The number of sensors or actuators determines whether a control system is a SISO, MIMO system, single-input multiple-output (SIMO), and multiple-input single-input (MISO) systems.

Continuous models use the *s* variable to define time, whereas discrete models use the *z* variable.

Table 2-1. Definitions of Continuous and Discrete Systems

Model Type	Continuous	Discrete
Transfer Function	$H(s) = \frac{b_0 + b_1 s + \ldots + b_{m-1} s^{m-1} + b_m s^m}{a_0 + a_1 s + \ldots + a_{n-1} s^{n-1} + a_n s^n}$ $\boldsymbol{H} = [H_{ij}]$	$H(z) = \frac{b_0 + b_1 z + \ldots + b_{m-1} z^{m-1} + b_m z^m}{a_0 + a_1 z + \ldots + a_{n-1} z^{n-1} + a_n z^n}$ $\boldsymbol{H} = [H_{ij}]$
Zero-Pole-Gain	$H(s) = \frac{k(s-z_1)(s-z_2)..(s-z_m)}{(s-p_1)(s-p_2)..(s-p_n)}$ $\boldsymbol{H} = [H_{ij}]$	$H(z) = \frac{k(z-z_1)(z-z_2)..(z-z_m)}{(z-p_1)(z-p_2)..(z-p_n)}$ $\boldsymbol{H} = [H_{ij}]$
Space-State	$\dot{x} = \boldsymbol{A}x + \boldsymbol{B}u$ $y = \boldsymbol{C}x + \boldsymbol{D}u$	$x(k+1) = \boldsymbol{A}x(k) + \boldsymbol{B}u(k)$ $y(k) = \boldsymbol{C}x(k) + \boldsymbol{D}u(k)$

You can create all these system models using the Model Construction VIs in the Control Design Toolkit. Refer to the *Transfer Function Model* section, the *Zero-Pole-Gain Model* section, and the *State-Space Model* section of this chapter for information about these system models and creating them in LabVIEW.

RLC Circuit Example

To illustrate how you can represent one system with different model types, consider the following example of an RLC circuit, shown in Figure 2-2. This simple circuit consists of an inductor *L*, a resistor *R*, a capacitor *C*, and an input voltage v_i.

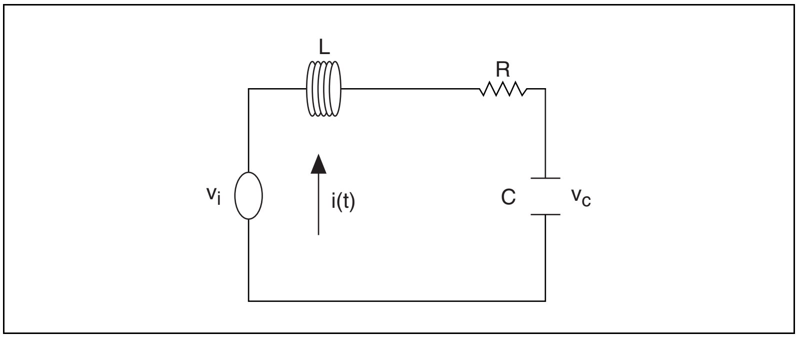


Figure 2-2. RLC Circuit

The *Transfer Function Model* section, the *Zero-Pole-Gain Model* section, and the *State-Space Model* section of this chapter explain how you can create different models to represent the same system.

Transfer Function Model

One way of representing a linear time-invariant system is with a transfer function model. Transfer functions define the dynamic relationship between inputs and outputs of a system. The following equations define continuous and discrete transfer functions where the numerator and denominator are polynomials.

Continuous Transfer Function Model

$$H(s) = \frac{numerator(s)}{denominator(s)} = \frac{b_0 + b_1 s + \ldots + b_{m-1}s^{m-1} + b_m s^m}{a_0 + a_1 s + \ldots + a_{n-1}s^{n-1} + a_n s^n}$$

Discrete Transfer Function Model

$$H(z) = \frac{numerator(z)}{denominator(z)} = \frac{b_0 + b_1 z + \ldots + b_{m-1}z^{m-1} + b_m z^m}{a_0 + a_1 z + \ldots + a_{n-1}z^{n-1} + a_n z^n}$$

Creating Transfer Function Models

Use the CD Construct Transfer Function Model VI to create SISO and MIMO system models in transfer function form. This VI creates a data structure that defines the transfer function model and contains additional information about the system, such as its sampling time, input or output delays, and input and output names. Refer to the *Model Information* section of this chapter for information about other properties of transfer function models.

SISO Transfer Function Model

For the example in the *RLC Circuit Example* section of this chapter, you can describe the voltage of the capacitor v_c using the following second order differential equation.

$$LC\ddot{v}_c + RC\dot{v}_c + v_c = v_i$$

Taking the Laplace transform and rearranging terms, you then can write the transfer function between the input voltage V_i and the capacitor voltage V_c using the following equation.

$$\frac{V_c(s)}{V_i(s)} = \frac{\frac{1}{LC}}{s^2 + \frac{Rs}{L} + \frac{1}{LC}} = H(s)$$

Using this transfer function, $H(s)$, you can study various dynamic properties of the RLC circuit.

If $R = 1$, $L = 2$, and $C = 3$, the corresponding transfer function is defined by the following equation.

$$H(s) = \frac{1.5}{s^2 + 0.5s + 0.1667}$$

Figure 2-3 shows how you create the continuous transfer function model in LabVIEW.

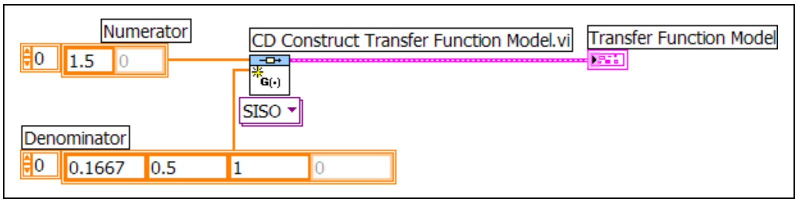


Figure 2-3. Creating a Continuous Transfer Function Model

The **Numerator** and **Denominator** inputs are zero-based arrays, and the i^{th} element of the array corresponds to the i^{th} order coefficient of the polynomial.

Note The transfer function model does not automatically cancel polynomial roots appearing in both the numerator and the denominator of a transfer function.

By using the coefficients of the continuous model and setting the **Sampling Time (s)** of the system to a value greater than zero, you do not create the discrete-time equivalent of the system. If you create a continuous transfer function model and want to convert the model to a discrete model, you must use the CD Convert Continuous to Discrete VI. You set the sampling time of the discrete model using the **Sampling Time (s)** parameter of the CD Convert Continuous to Discrete VI. Figure 2-4 illustrates how to convert the continuous transfer function model into a discrete transfer function model with a sampling time of 5 ms.

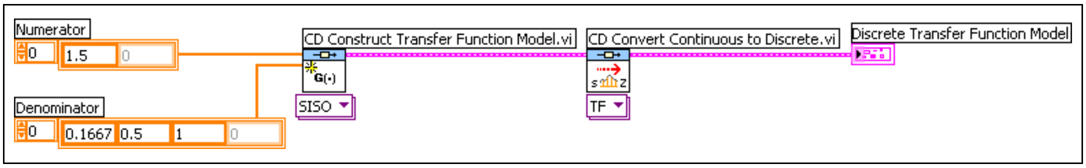

Numerator
0
1.5
0
CD Construct Transfer Function Model.vi
G(·)
SISO
CD Convert Continuous to Discrete.vi
TF
Discrete Transfer Function Model
Denominator
0
0.1667
0.5
1
0

Figure 2-4. Creating a Discrete Transfer Function Model from a Continuous Transfer Function Model

This conversion yields the following discrete transfer function model.

$$H(z) = \frac{1.873 \times 10^{-5} z + 1.872 \times 10^{-5}}{z^2 - 1.998z + 0.998}$$

If you already know the coefficients of a discrete transfer function model, you can create the model using the CD Construct Transfer Function Model. You can enter the coefficients for a discrete transfer function model in the **Numerator** and **Denominator** inputs and set the **Sampling Time (s)** of the system to a value greater than zero. Figure 2-5 illustrates how to create the transfer function of the following discrete system with a sampling time of 5 ms.

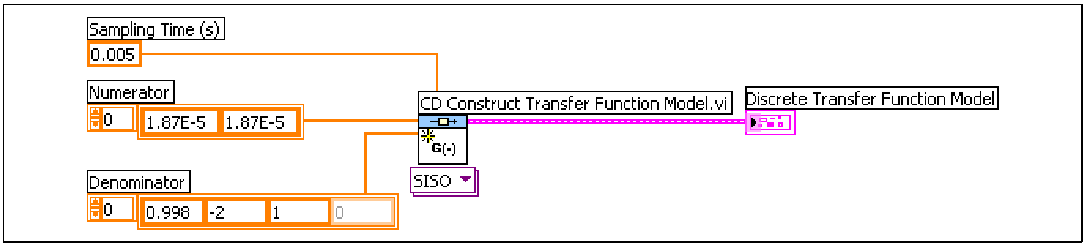


Figure 2-5. Creating a Discrete Transfer Function Model

The resulting **Discrete Transfer Function Model** is the same model as the one created in Figure 2-4.

MIMO Transfer Function Model

Consider the two-input two-output system shown in Figure 2-6.

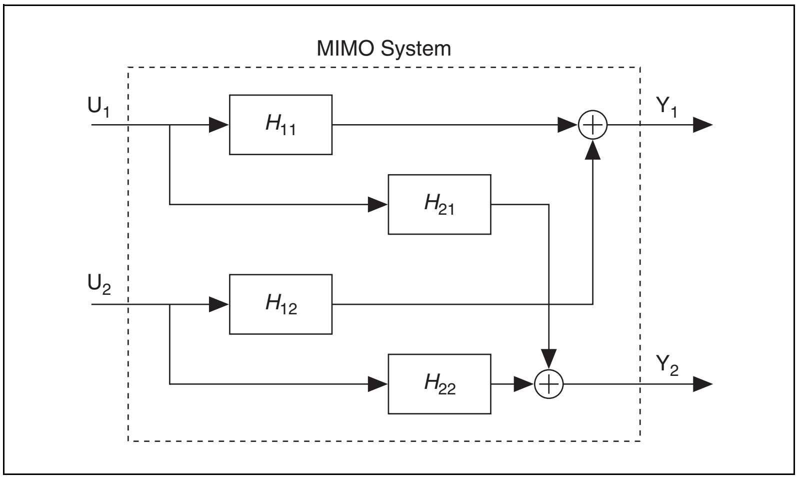


Figure 2-6. MIMO System with Two Inputs and Two Outputs

You can define the transfer function of this MIMO system by using the following transfer function matrix $\boldsymbol{H}$, where each element represents a SISO transfer function.

$$\boldsymbol{H} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$$

Suppose the following equations define the SISO transfer functions between each input-output pair.

$$H_{11} = \frac{1}{s} \qquad H_{12} = \frac{2}{s+1}$$

$$H_{21} = \frac{s+3}{s^2+4s+6} \qquad H_{22} = 4$$

To create a transfer function model of this MIMO system, you must select the MIMO instance of the CD Construct Transfer Function Model VI.

You then can specify each transfer function between the j^{th} input and the i^{th} output as the ij^{th} element of the two-dimensional **Transfer Function(s)** input array. Figure 2-7 illustrates that the numerator-denominator pair of the first row and first column corresponds to H_{11}, the numerator-denominator pair of the first row and second column corresponds to H_{12}, and so on.

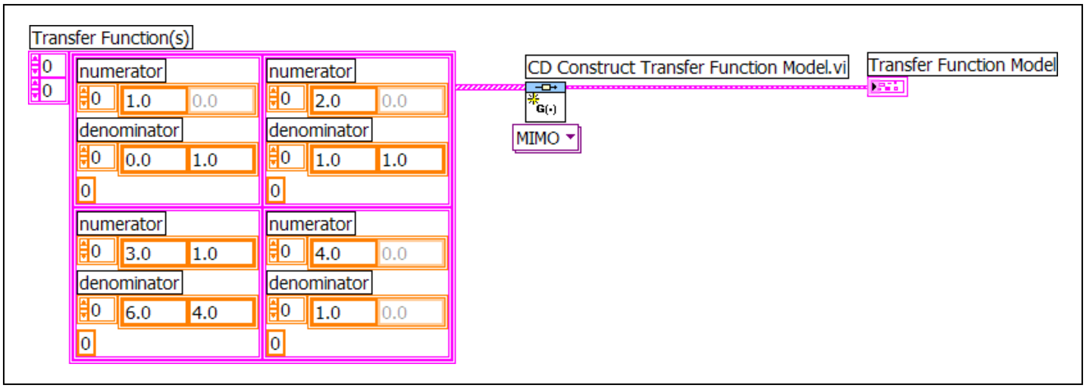


Figure 2-7. Creating a MIMO Transfer Function Model

The elements in the **numerator** and **denominator** arrays correspond to the coefficients, in ascending order, of the numerator and denominator in the H_{ij} transfer function model. For example, the numerator of H_{11} is 1, which corresponds to the zero-order coefficient. Therefore, the first element in the **numerator** array for H_{11} is 1. The denominator of H_{11} is s, which means the value 0 corresponds to the zero-order coefficient and the value 1 corresponds to the first-order coefficient. Therefore the first element in the **denominator** array for H_{11} is 0 and the second element is 1.

Symbolic Transfer Function Model

You also can create symbolic transfer function models and evaluate them by specifying numeric values for each variable you use.

Using the following transfer function, initially defined in the *SISO Transfer Function Model* section of this chapter, you can create a symbolic model of transfer function. The model is symbolic in that it allows you to use variables, rather than numerical values, to define the transfer function.

$$H(s) = \frac{\frac{1}{LC}}{s^2 + \frac{Rs}{L} + \frac{1}{LC}}$$

You must select the SISO (Symbolic) instance of the CD Construct Transfer Function Model VI. When specifying the **Symbolic Numerator** and **Symbolic Denominator** coefficients for the system, you use the actual variable names, R, L, and C. You then specify values of the numerator and denominator coefficients in the **variables** input, as shown in Figure 2-8.

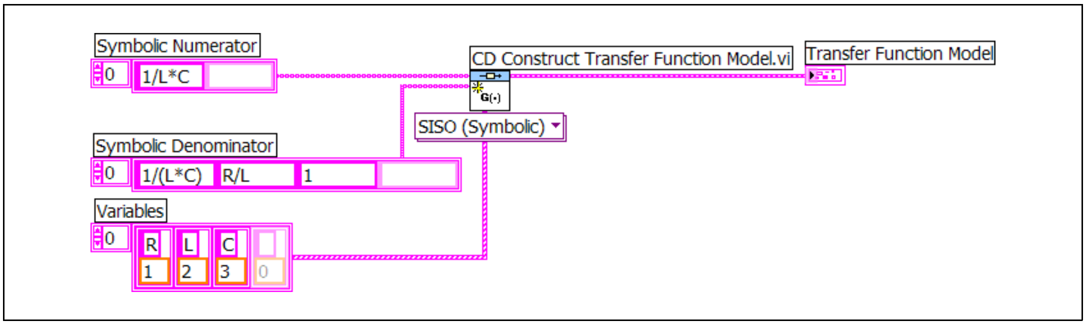


Figure 2-8. Creating a Symbolic Transfer Function

Zero-Pole-Gain Model

Another way of representing a linear time-invariant system is with a zero-pole-gain model. Zero-pole-gain models also define the dynamic relationship between inputs and outputs of a system. However, you represent the numerator and denominator polynomials in terms of their roots, thereby showing the poles and zeros of the system. The following equations define continuous and discrete zero-pole-gain models, where the numerators and denominators are first-order polynomials.

Continuous Zero-Pole-Gain Model

$$H_{ij}(s) = k\frac{\prod_{i=0}^{m} s + z_i}{\prod_{i=0}^{n} s + p_i} = \frac{k(s-z_1)(s-z_2)..(s-z_m)}{(s-p_1)(s-p_2)..(s-p_n)}$$

Discrete Zero-Pole-Gain Model

$$H_{ij}(z) = k\frac{\prod_{i=0}^{m} z + z_i}{\prod_{i=0}^{n} z + p_i} = \frac{k(z-z_1)(z-z_2)..(z-z_m)}{(z-p_1)(z-p_2)..(z-p_n)}$$

In these equations, k is a scalar quantity that represents the gain, z_i represents the zeros, and p_i represents the poles of the system model.

Creating Zero-Pole-Gain Models

Use the CD Construct Zero-Pole-Gain Model VI to create SISO and MIMO system models in zero-pole-gain form. This VI creates a data structure that defines the zero-pole-gain model and contains additional information about the system, such as its sampling time, input or output delays, and input and output names. Refer to the *Model Information* section of this chapter for information about other properties of zero-pole-gain models.

SISO Zero-Pole-Gain Model

For the example in the *RLC Circuit Example* section of this chapter, let $R = 1$, $L = 2$, and $C = 3$. The corresponding continuous zero-pole-gain model is defined by the following transfer function.

$$G(s) = \frac{1.5}{(s - 0.25 \pm 0.3228i)^2}$$

This equation defines a model with two complex conjugate poles. The complex conjugate poles are at $-0.25 \pm 0.3228i$.

Figure 2-9 shows how you create the continuous zero-pole-gain model in LabVIEW.

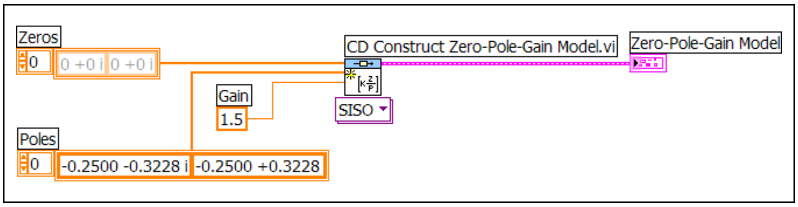


Figure 2-9. Creating a Zero-Pole-Gain Model from Poles and Zeros

For discrete systems, you have to specify the sampling time of the system in addition to the discrete gain, poles, and zeros. You create a discrete zero-pole-gain model the same way you create a discrete transfer function model. First set the **Sampling Time (s)** of the system to a value greater than zero, then use the CD Convert Continuous to Discrete VI to convert the continuous-time system to the discrete-time equivalent of the system. Refer to the *SISO Transfer Function Model* section of this chapter for information about creating a discrete system model.

MIMO Zero-Pole-Gain Model

You create MIMO zero-pole-gain system the same way you create a MIMO transfer function model. Refer to the *MIMO Transfer Function Model* section of this chapter for information about creating a MIMO system model.

Symbolic Zero-Pole-Gain Model

You create a symbolic zero-pole-gain model the same way you create a symbolic transfer function model. Refer to the *Symbolic Transfer Function Model* section of this chapter for information about creating a symbolic system model.

State-Space Model

Another way of representing a linear time-invariant system is with a state-space model. In state-space models, first-order differential (continuous) and difference (discrete) equations are represented as a set of state and output updates.

Continuous State-Space Model

$$\begin{aligned} \dot{x} &= \boldsymbol{A}x + \boldsymbol{B}u \\ y &= \boldsymbol{C}x + \boldsymbol{D}u \end{aligned}$$

Discrete State-Space Model

$$\begin{aligned} x(k+1) &= \boldsymbol{A}x(k) + \boldsymbol{B}u(k) \\ y(k) &= \boldsymbol{C}x(k) + \boldsymbol{D}u(k) \end{aligned}$$

The Table 2-2 describes the dimensions of the vectors and matrices of a state-space model.

Table 2-2. Dimensions and Names of State-Space Model Variables

Variable	Dimension	Name
k	—	discrete time
n	—	number of states
m	—	number of inputs
r	—	number of outputs
$\boldsymbol{A}$	$n \times n$ matrix	state matrix
$\boldsymbol{B}$	$n \times m$ matrix	input matrix
$\boldsymbol{C}$	$r \times n$ matrix	output matrix
$\boldsymbol{D}$	$r \times m$ matrix	direct transmission matrix

Table 2-2. Dimensions and Names of State-Space Model Variables (Continued)

Variable	Dimension	Name
x	n-vector	state vector
u	m-vector	input vector
y	r-vector	output vector

Creating State-Space Models

Use the CD Construct State-Space Model VI to create SISO and MIMO system models in state-space form. This VI creates a data structure that uses matrices to define the state-space model. The matrices are two-dimensional arrays of numbers where the *ij*th element of the array corresponds to the *ij*th element of the state-space matrix. The arrays are zero-based. An *n*th order system with *m* inputs and *r* outputs is assumed to have state, input, and output vectors, as defined in the following equations.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \qquad u = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{m-1} \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{r-1} \end{bmatrix}$$

State-space models also contain other information about the system model such as the sampling time, delays on the inputs or outputs, and names of the inputs and outputs. Refer to the *Model Information* section of this chapter for information about other properties state-space models contain.

SISO State-Space Model

For the example in the *RLC Circuit Example* section of this chapter, you can describe the system using a state-space model defined by the following equations.

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{LC} & -\frac{R}{L} \end{bmatrix} x + \begin{bmatrix} 0 \\ -\frac{1}{L} \end{bmatrix} u$$

$$y = \begin{bmatrix} -\frac{1}{C} & -R \end{bmatrix} x + \begin{bmatrix} 1 \end{bmatrix} u$$

In these equations, u equals the input voltage v_i and y equals the voltage of the capacitor v_c. If $R = 1$, $L = 2$, and $C = 3$, the corresponding state-space model is defined by the following matrices.

$$\boldsymbol{A} = \begin{bmatrix} -0.50 & -0.17 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & 1.5 \end{bmatrix} \qquad \boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix}$$

Figure 2-10 shows how you create the continuous state-space model in LabVIEW.

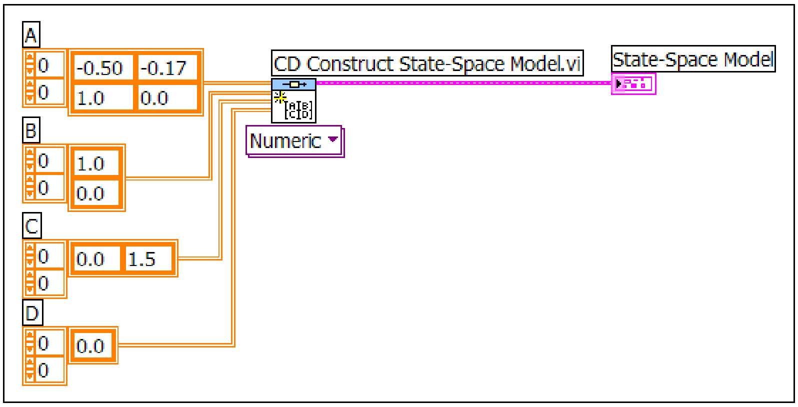


Figure 2-10. Creating a Continuous State-Space Model

For discrete systems, you have to specify the sampling time of the system in addition to the discrete $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ matrices of a state-space model. You create a discrete state-space model the same way you create a discrete transfer function model. Refer to the *SISO Transfer Function Model* section of this chapter for more information about creating a discrete system model.

MIMO State-Space Model

You create a MIMO state-space model by adding more states to the system. To add more states, add more rows and columns to the matrices $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$.

Symbolic State-Space Model

You create a symbolic state-space model the same way you create a symbolic transfer function model. Refer to the *Symbolic Transfer Function Model* section of this chapter for more information about creating a symbolic system model.

Model Information

The Model Construction VIs create data structures that define different system models. You use the data structures with every VI in the Control Design Toolkit that accepts a model as an input.

In addition to containing the mathematical representation of the model, each data structure also encapsulates a set of properties that provide information about the system. These properties are common in all three types of system models. Table 2-3 lists the properties and their corresponding data types.

Table 2-3. Model Properties

Property	Data Type	Description
Model Name	String	Assigns a name to a specific model.
Input Names	1D array of strings	The i^{th} element of the array defines the name of the i^{th} input to the model.
Output Names	1D array of strings	The i^{th} element of the array defines the name of the i^{th} output of the model.
Input Delays	1D array of double-precision, floating-point numeric values	The i^{th} element of the array defines the time delay of the i^{th} input of the model.
Output Delays	1D array of double-precision, floating-point numeric values	The i^{th} element of the array defines the time delay of the i^{th} output of the model.
Transport Delay	1D array of double-precision, floating-point numeric values	The ij^{th} element of the array defines the time delay between the i^{th} output and j^{th} input of the model.
Notes	String	A string for storing additional data. It can contain comments or other information that you want to store with the model.

Table 2-3. Model Properties (Continued)

Property	Data Type	Description
Sampling Time	Double-precision, floating-point numeric value	Represents the sampling time (in seconds) of the system. If a model represents a continuous system, the value of **Sampling Time** is zero. For discrete system models, the value must be greater than zero.
State Names	Array of strings	The i^{th} element of the array defines the name of the i^{th} state of the model. This property is available with state-space models only.

You can use the Model Information VIs to get and set various properties of the model. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about the Model Information VIs and how to use them to change the properties of a system model.

3

Model Conversion

The Model Conversion VIs enable you to convert system models from one representation to another, from continuous to discrete models, and from discrete to continuous models. You also can convert models you created using the LabVIEW Simulation Module into models you can use in the LabVIEW Control Design Toolkit and vice versa. Refer to the *LabVIEW Simulation Module User Manual* and the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about simulation models.

This chapter describes how you change the representation of a system and how you convert between continuous and discrete systems.

Changing System Representation

Control design uses transfer function, zero-pole-gain, and state-space models to represent systems. Certain model representations are better for certain analysis techniques in control design. Therefore, when you design control systems, you need the flexibility to switch between transfer function, zero-pole-gain, and state-space representations of systems. This conversion is necessary to create and understand the behavior of certain controls in a system model.

Converting State-Space Models to Transfer Function Models

Consider the continuous state-space model defined in the *State-Space Model* section of Chapter 2, *Creating Dynamic System Models*.

$$\begin{aligned} \dot{x} &= \boldsymbol{A}x + \boldsymbol{B}u \\ y &= \boldsymbol{C}x + \boldsymbol{D}u \end{aligned}$$

The Laplace transform is used to convert from the time domain to the Laplace domain model representation for continuous systems.

Note For discrete systems, you use the *z*-transform to convert from the time domain to the *z*-domain. This section describes changing the model type in the continuous domain. However, these equations also apply to models in the discrete domain.

The application of the Laplace transform to the state-space model results in the following equation, where *s* is the Laplace variable.

$$Y(s) = [\boldsymbol{C}(\boldsymbol{I}s - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}]U(s)$$

The matrix transfer function model *H*(*s*) is defined as the ratio between the output *Y*(*s*) and input *U*(*s*).

$$H(s) \equiv \frac{Y(s)}{U(s)} = \boldsymbol{C}(\boldsymbol{I}s - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}$$

Note When you convert a state-space model to a transfer function model, you lose the state information originally represented by the vector *x*.

Consider the following second-order MISO state-space system model.

$$\dot{x} = \begin{bmatrix} -1 & 2 \\ 0 & -1 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \end{bmatrix} u$$

Using the Laplace transform, you obtain the transfer function matrix *H*(*s*).

$$H(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{2}{s^2 + 2s + 1} \end{bmatrix}$$

Use the CD Convert to Transfer Function Model VI to convert a state-space model to a transfer function model. This VI also converts a zero-pole-gain model to a transfer function model.

Converting Transfer Function Models to Zero-Pole-Gain Models

To convert the transfer function matrix *H*(*s*) into the zero-pole-gain form, you calculate the numerator and denominator polynomial roots and the gain of each SISO transfer function in *H*(*s*).

When you convert the transfer function matrix from the *Converting State-Space Models to Transfer Function Models* section of this chapter, you obtain the following zero-pole-gain model.

$$H(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{2}{(s+1)^2} \end{bmatrix}$$

Use the CD Convert to Zero-Pole-Gain Model VI to convert a transfer function model to a zero-pole-gain model. This VI also converts a state-space model to a zero-pole-gain model.

Note To convert a state-space model to a zero-pole-gain model, the CD Convert to Zero-Pole-Gain Model VI internally converts a state-space model into a transfer function model first.

Converting Zero-Pole-Gain Models to State-Space Models

Use the CD Convert to State-Space Model VI to convert a zero-pole-gain model to a state-space model. This VI also converts a transfer function model to a state-space model.

Note To convert a zero-pole-gain model to a state-space model, the CD Convert to State-Space Model VI first converts the zero-pole-gain model into a transfer function model.

The CD Convert to State-Space Model VI uses the least common denominator roots of all elements in the transfer function model to determine the number of states necessary to convert the transfer function model to a state-space model. You can represent a transfer function model as a state-space model in an infinite number of ways. Therefore each representation is a state-space realization. A full state-space realization does not reduce the number of states determined by the least common denominator calculation. However, a minimal state-space realization reduces the number of states to produce a minimal representation of the original transfer function model.

Refer to the *Minimal Realization* section of Chapter 11, *Model Reduction*, for more information about state-space realizations. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the Model Conversion VIs.

Minimal Realization

When you convert the representation of a higher-order model, you often need a minimal realization of the model. A minimal realization consists of the minimum number of states necessary to describe a dynamic system. However, the minimal realization is not unique.

Using the example in the *Converting State-Space Models to Transfer Function Models* section of this chapter, the minimal realization when converting from a zero-pole-gain to a state-space model is given by the following equation.

$$\dot{x} = \begin{bmatrix} -0.33 & 0.94 \\ -0.47 & -1.67 \end{bmatrix} x + \begin{bmatrix} -0.41 & 0 \\ 0.29 & -0.87 \end{bmatrix} u$$

$$y = \begin{bmatrix} -2.45 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \end{bmatrix} u$$

This model numerically differs from the state-space model initially defined. However, from the input-output model perspective, the state-space models are identical.

Refer to the *Minimal Realization* section of Chapter 11, *Model Reduction*, for more information about minimal realization.

Converting between Continuous and Discrete Systems

Control systems are either analog or digital systems. Analog control systems are continuous systems that implement the controller using analog physical components. Digital control systems are discrete systems that interact with the plant using sensors and actuators. You implement a digital control system using computer equipment.

When designing a digital control system, you can convert a continuous model to a discrete model to approximate an accurate real-time solution. You can use a number of mathematical methods, listed in Table 3-1, to approximate the behavior of a continuous system in discrete time. You can use these same methods to convert a discrete model to a continuous model and a discrete model to another discrete model with a different sampling rate.

Table 3-1 summarizes the mathematical methods of approximation you can use as substitutions between the continuous Laplace-transform operator and the discrete z-transform operator.

Table 3-1. Mapping Methods for CD Convert to Discrete

Method of Approximation	Continuous to Discrete	Discrete to Continuous
Forward Rectangular Method	$s \rightarrow \frac{z-1}{T}$	$z \rightarrow 1 + sT$
Backward Rectangular Method	$s \rightarrow \frac{z-1}{zT}$	$z \rightarrow \frac{1}{1-sT}$
Tustin's Method	$s \rightarrow \frac{2(z-1)}{T(z+1)}$	$z \rightarrow \frac{1+\frac{sT}{2}}{1-\frac{sT}{2}}$
Prewarp Method	$s \rightarrow \frac{z(z-1)}{T^*(z+1)}$ $T^* = \frac{2\tan\left(\frac{w \times T}{2}\right)}{w}$	$z \rightarrow \frac{1+sT^*}{1-sT^*}$ $T^* = \frac{2\tan\left(\frac{w \times T}{2}\right)}{w}$

The following sections describe the Model Conversion VIs that you can use to perform continuous to discrete conversions, discrete to continuous conversions, and discrete to discrete conversions. The following sections also describe all the mathematical methods that the LabVIEW Control Design Toolkit uses to perform these conversions.

Continuous to Discrete Conversions

The CD Convert Continuous to Discrete VI provides the following mathematical methods for converting a continuous model to a discrete model.

- Forward Rectangular
- Backward Rectangular
- Tustin's
- Prewarp Conversion
- Zero-Order-Hold
- First-Order-Hold
- Z Transform
- Matched Pole Zero

To convert the model from a continuous differential equation to a discrete difference equation, first approximate the value of the derivative in the continuous equation over each change in time. Then find the area of the geometric region having width *dt* and height equal to the derivative.

Consider the following first-order differential equation.

$$\dot{y} = f(t)$$

To convert this differential equation into a difference equation, evaluate the derivative function $f(t)$ at different points to approximate $\dot{y}$ at time *t*. Figure 3-1 illustrates the function $f(t)$ between *t* and *t* + *T*, where *T* is the sampling time.

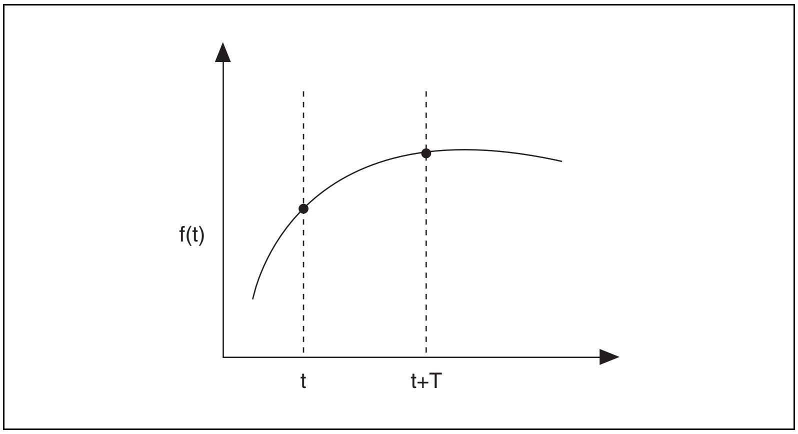


Figure 3-1. Discretizing a Differential Equation

Integrating between time *t* and *t* + *T* results in the following difference equation.

$$\int_{t}^{t+T} \dot{y}\,d\tau = y(t+T) - y(t) = \int_{t}^{t+T} f(\tau)\,d\tau$$

The right side of this equation represents the area under the curve between *t* and *t* + *T*.

The following sections describe the methods—Forward Rectangular, Backward Rectangular, Tustin's, Prewarp, Zero-Order-Hold, and First-Order-Hold—you can use to convert the model from a continuous differential equation to a discrete difference equation.

Forward Rectangular Method

You can approximate the area under the curve by considering $f(\tau)$ constant and equal to $f(t + T)$ along the integration range, which results in the following equation.

$$y(t+T) = y(t) + f(t+T)T$$

This integral approximation is also known as the Forward Rectangular method and considers the incremental area term between sampling times t and $t + T$ as a rectangle of width T and height equal to $f(t + T)$, as shown in Figure 3-2.

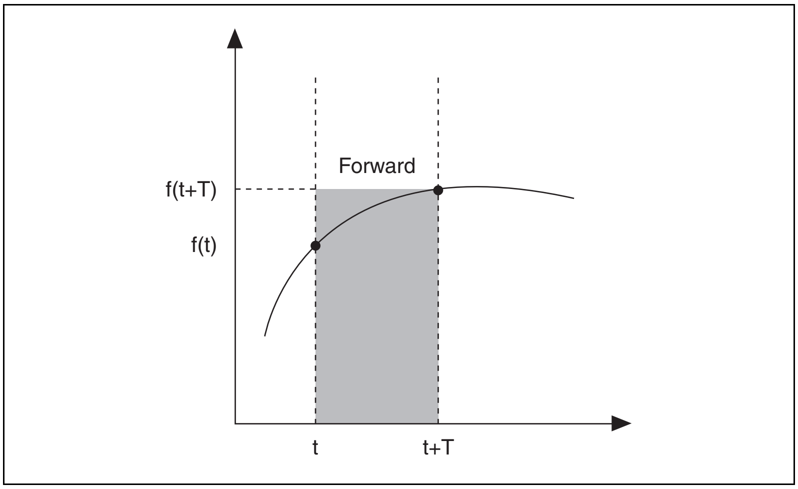


Figure 3-2. Forward Rectangular Method

Backward Rectangular Method

You can approximate the area under the curve by considering $f(\tau)$ constant and equal to $f(t)$ along the integration range, which results in the following equation.

$$y(t+T) = y(t) + f(t)T$$

This integral approximation is also known as the Backward Rectangular method. The Backward Rectangular method is similar to the Forward Regular method, except that the Backward Rectangular method considers the incremental area term between sampling times t and $t + T$ as a rectangle of width T and height equal to $f(t)$, as shown in Figure 3-3.

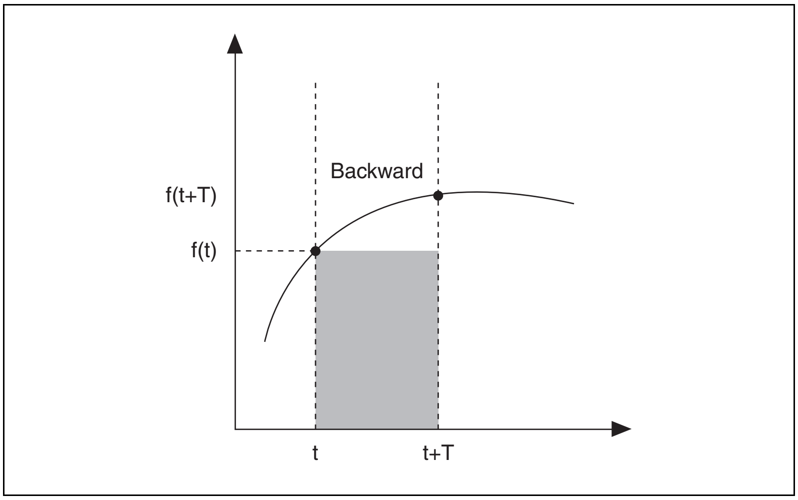


Figure 3-3. Backward Rectangular Method

Note The Forward Rectangular method tends to overestimate the incremental area and the Backward Rectangular method tends to underestimate it. To reduce the error in estimation, use a small sampling interval.

Tustin's Method

The Tustin's, or trapezoid, method provides a balance between the Forward Rectangular and Backward Rectangular methods by taking the average of the rectangles defined by these methods and using the average value as the incremental area to approximate the difference equation.

You can approximate the area under the curve is by considering $f(\tau)$ constant and equal to the average between $f(t)$ and $f(t + T)$ along the integration range, which results in the following equation.

$$y(t+T) = y(t) + \frac{[f(t) + f(t+T)]}{2}T$$

The last term in this equation is identical to the area of a trapezoid of height T and bases $f(t)$ and $f(t + T)$.

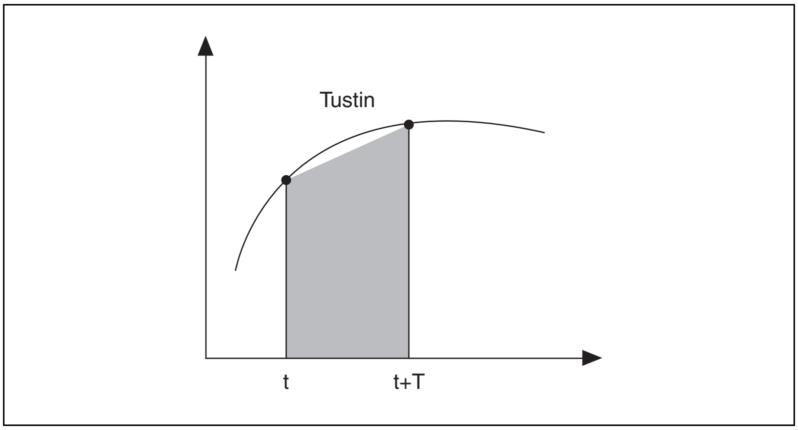


Figure 3-4. Tustin's Method

Prewarp Conversion Method

The Prewarp Conversion method is a trapezoidal type of transformation that uses the prewarp frequency (ω) to adjust the sampling time T. You can approximate the area under the curve by considering $f(\tau)$ constant and equal to the average between $f(t)$ and $f(t + T^*)$ along the integration range, which results in the following equation.

$$y(t+T) = y(t) + \frac{[f(t) + f(t+T^*)]}{2}T$$

The last term in this equation is identical to the area of a trapezoid of height T and bases $f(t)$ and $f(t + T^*)$.

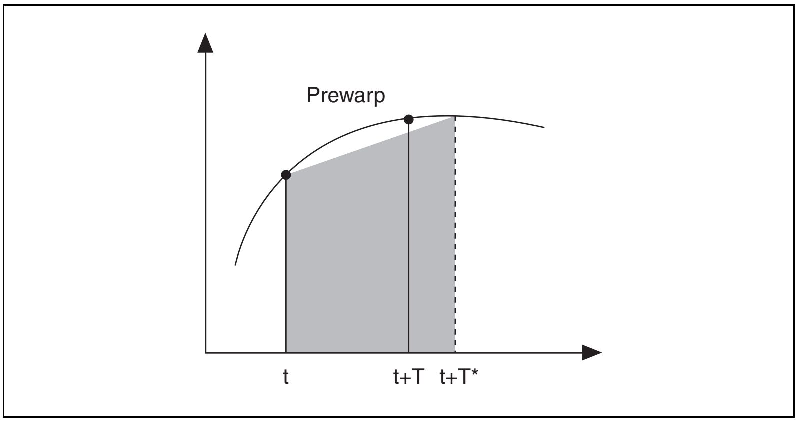


Figure 3-5. Prewarp Conversion Method

Zero-Order-Hold and First-Order-Hold Methods

The Zero-Order-Hold and First-Order-Hold discretization methods are different from other methods because they make assumptions of the derivative function structure, $f(t)$. These methods assume that $f(t)$ consists of an input that can be held constant (Zero-Order-Hold) or ramped over time (First-Order-Hold) during the integration period between t and $t + T$. The remaining terms of $f(t)$ not related to the input are integrated as these terms refer to the internal state dynamics.

With linear time invariant systems, such as state-space models, you obtain the following equation after integrating between t and $t + T$.

$$x(t+T) = e^{\boldsymbol{A}T}x(t) + \int_{t}^{t+T} e^{\boldsymbol{A}(t+T+\tau)}\boldsymbol{B}u(\tau)d\tau$$

$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

u is the input to the system and is not necessarily constant between t and $t + T$. The Zero-Order-Hold method approximates the input to a constant value $u(t)$ during the integration time.

$$x(t+T) = e^{\boldsymbol{A}T}x(t) + \int_{t}^{t+T} e^{\boldsymbol{A}(t+T+\tau)}\boldsymbol{B}d\tau\, u(\tau)$$

The First-Order-Hold method ramps the input values assuming a constant slope $[u(t + T) - u(t)]/T$ during integration time.

$$x(t+T) = e^{\boldsymbol{A}T}x(t) + \int_{t}^{t+T} e^{\boldsymbol{A}(t+T+\tau)}\boldsymbol{B}\left\{u(t) + [u(t+T) - u(t)]\frac{(\tau - t)}{T}\right\}d\tau$$

Refer to *Digital Control of Dynamic Systems*[1] for more information about the Zero-Order-Hold and First-Order-Hold methods.

[1] Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.

Z Transform Conversion Method

The Z Transform conversion method is defined such that the continuous and discrete impulse responses maintain major similarities. You calculate the impulse response of the discrete transfer function by multiplying the inverse Laplace transform of the continuous transfer function by *dt*.

Refer to *Discrete-Time Control Systems*[1] for more information about the Z Transform conversion method.

Matched Pole Zero Conversion Method

Similar to how you can represent the discrete sample time delay z^{-1} with e^{-st_d} in the continuous time domain, you can apply the following equivalents within the discrete and continuous time domains.

- You can map any pole or zero at $s = a + bi$ in a continuous transfer function to $z = e^{(a + bi)T} = e^{aT} \cdot e^{biT}$ in the discrete time domain. In polar coordinates, the location of the pole or zero in the discrete time domain is defined by $r = e^{aT}$ and $\phi = bT$.
- You must choose the gain of the discrete transfer function to match the gain of the continuous system at the limiting stability location in the complex plane. Therefore, set the gain of the continuous transfer function at $s = 0$ to match the discrete transfer function gain at $z = 1$.

Refer to *Digital Control of Dynamic Systems*[2] for more information about the Matched Pole Zero conversion method.

Discrete to Continuous Conversions

You can reverse many of the discretization techniques discussed in the *Continuous to Discrete Conversions* section of this chapter using the CD Convert Discrete to Continuous VI. You can reverse the Forward Rectangular, Backward Rectangular, Tustin's, Prewarp, and Zero-Order-Hold methods to map the *z*-plane to the *s*-plane. Refer to Table 3-1 for the equations for each mapping method.

[1] Ogata, Katsuhiko. *Discrete-Time Control Systems,* 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, 1995.

[2] Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.

The Z Transform conversion method is also a reverse calculation to map a model in the *z*-plane to the *s*-plane. You calculate the impulse response of the continuous transfer function by dividing the inverse *z*-transform of the discrete transfer function by *dt*. The CD Convert Discrete to Continuous VI does not support the First-Order-Hold or the Matched Pole Zero conversion methods.

Discrete to Discrete Conversions

The design of a digital control system often uses a specific sampling rate. However, you might want to modify the digital control system to use a different sampling rate. The process of adjusting a digital model to use a different sampling rate is known as model resampling. Model resampling requires the conversion of a discrete model to a continuous model and from a continuous model back to a new discrete model. This discrete to discrete conversion uses different sampling rates.

You can convert a discrete system model to another discrete system model by resampling the discrete system. The Control Design Toolkit internally performs two consecutive conversions to resample the model from T_1 to T_2. The first conversion uses the initial sampling time T_1 to convert the system model from discrete to continuous. The second conversion uses the final sampling time T_2 to convert the system model from continuous to discrete.

The CD Convert Discrete to Discrete VI resamples a discrete system using a different sampling time. You can specify one of the following methods of approximation you want to use when resampling a discrete system: Forward Rectangular, Backward Rectangular, Tustin's, Prewarp, Zero-Order-Hold, or Z Transform conversions methods. The CD Convert Discrete to Discrete VI does not support First-Order-Hold or Matched Pole Zero conversion methods.

4 Connecting Models

Dynamic systems are typically composed of many models, or subsystems, that are interconnected in various ways. The LabVIEW Control Design Toolkit provides Model Interconnection VIs that enable you to build a larger system from many subsystems. Therefore, modeling a complicated system is easier because you can describe the dynamics of individual pieces and connect them with the Model Interconnection VIs.

This chapter describes the four ways—in series, by appending, in parallel, and with feedback—in which you can interconnect models to create a larger model.

Connecting Models in Series

The CD Series VI joins outputs and inputs of consecutive systems. The outputs of the first model connect to inputs of the second model.

SISO Systems in Series

Consider a valve that controls the flow rate of water into a tank, as shown in Figure 4-1.

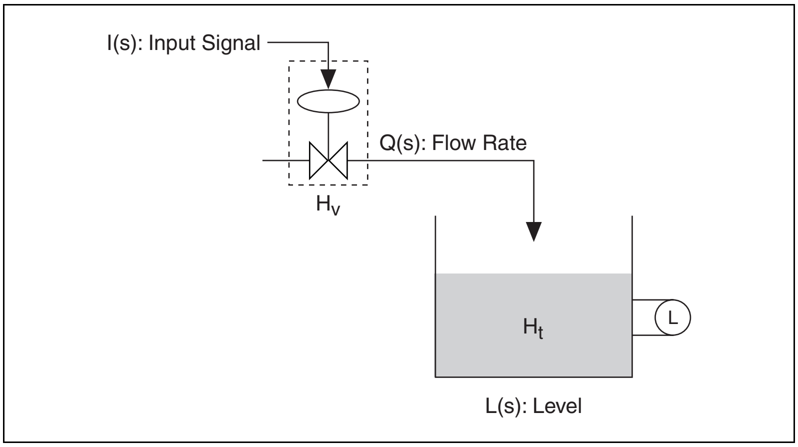


Figure 4-1. Flow of Water into a Tank

Assume that the incoming water pressure to the valve is constant. Therefore only the valve input signal affects the level of the water in the tank. You can model the flow rate of water into the tank using the following transfer functions. H_v is a model of the valve, and H_t is a model of the tank.

$$H_v \equiv \frac{Q(s)}{I(s)} = \frac{K_v}{\tau^2 s^2 + 2\zeta\tau s + 1} \qquad H_t \equiv \frac{L(s)}{Q(s)} = \frac{K_t}{s}$$

$I(s)$, $Q(s)$, and $L(s)$ represent the Laplace transform of the input signal, the flow rate, and level of water in the tank, respectively. The constants K_v, τ, ζ, and K_t are parameters of the models that describe the valve and tank. To obtain the effect of the input signal on the water level, place the two systems in series and multiply their transfer functions.

$$I(s) \equiv \frac{L(s)}{I(s)} = H_v \cdot H_t = \frac{K_v}{\tau^2 s^2 + 2\zeta\tau s + 1} \cdot \frac{K}{s}$$

Figure 4-2 illustrates how the two models become one by using a series connection, where the output of the first system, H_v, connects to the input of the second system, H_t.

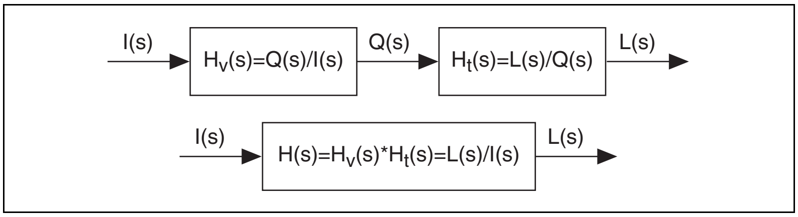


Figure 4-2. Valve Model and Tank Model in Series

The resulting SISO system $H(s)$ now represents the relationship between the input signal $I(s)$ and the level of water $L(s)$ in the tank.

MIMO Systems in Series

In a MIMO system, the connections between the two models can be arbitrary. For example, the first output of the first model can connect to the second input of the second model. Figure 4-3 shows two MIMO system models connected in series.

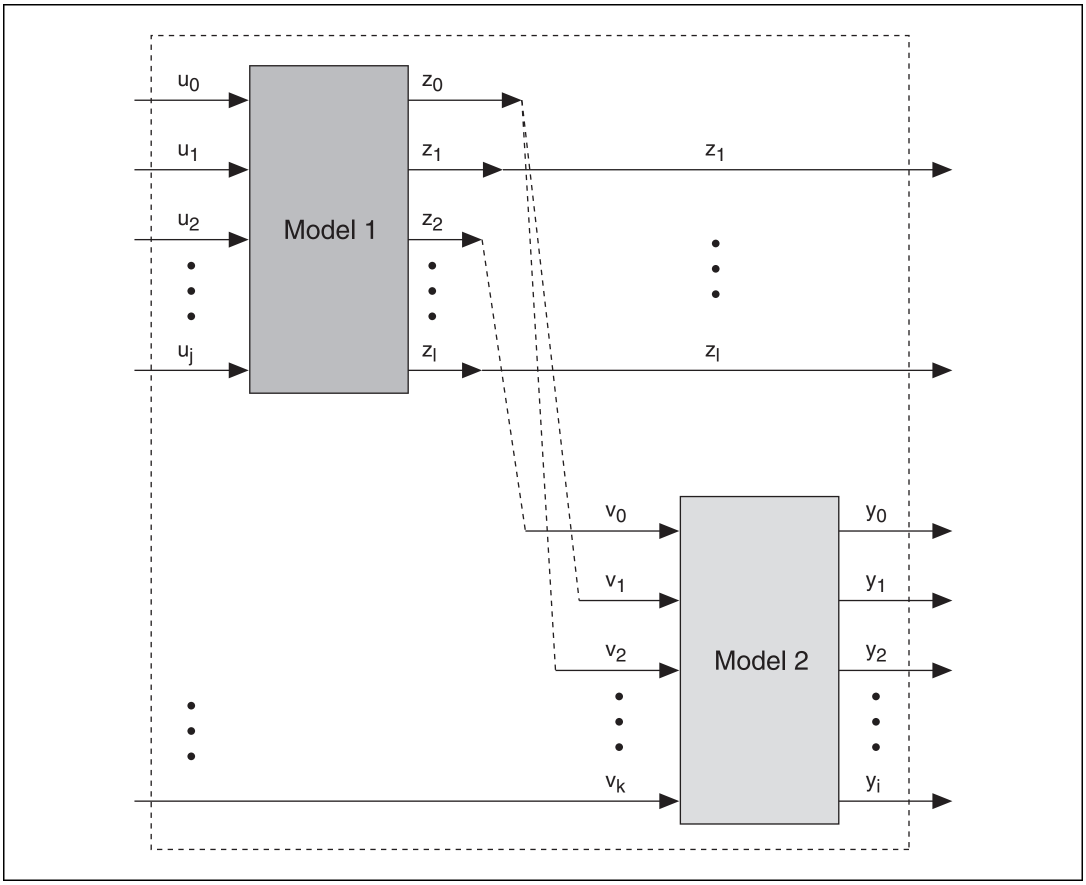


Figure 4-3. MIMO System Models in Series

Figure 4-3 shows how the outputs of Model 1 that are connected to the inputs of Model 2 do not appear as outputs of the resulting series model. For example, because z_0 connects to the Model 2 inputs v_1 and v_2, z_0 is no longer an output of the resulting series model. Similarly, z_2 is not an output of the resulting series model.

To define the relationship between the inputs and outputs of the models, use the CD Series VI. Figure 4-4 displays the **Connections** control you use to reflect the connections made in Figure 4-3.

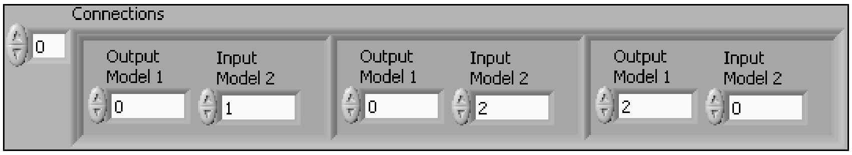


Figure 4-4. Connection Definitions for Models in Series

The control in Figure 4-4 indicates that the Model 1 output z_0 connects to the Model 2 inputs v_1 and v_2. You also can see how the Model 1 output z_2 connects to the Model 2 input v_0.

SIMO Systems in Series

Adding another valve and tank to the example in the *SISO Systems in Series* section of this chapter, you get the SIMO system represented in Figure 4-5. In this system, the flow rate is divided and sent to different tanks.

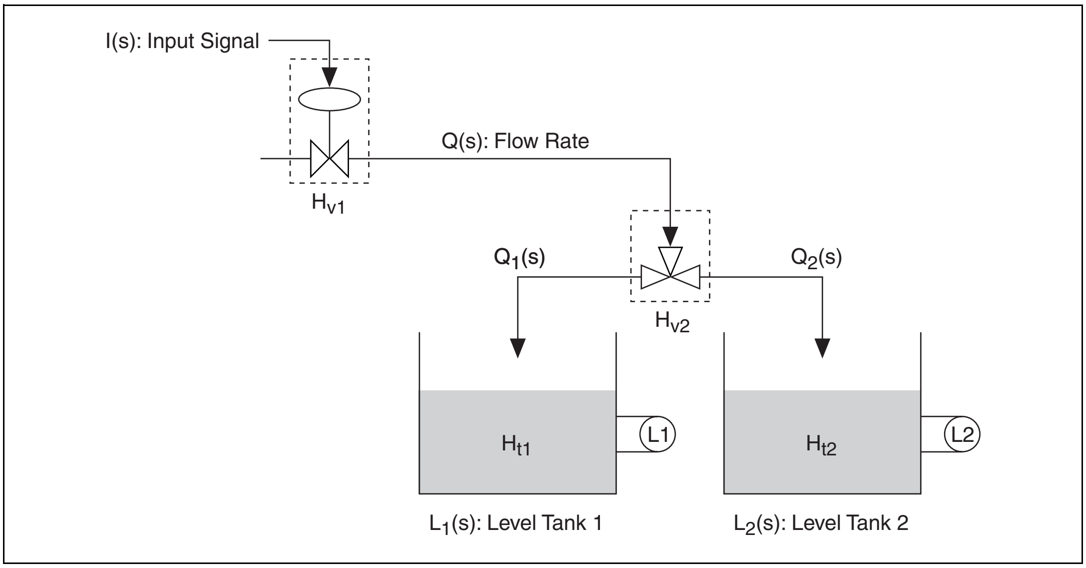


Figure 4-5. Flow of Water into Two Tanks

The transfer function matrix of the second valve H_{v2} represents the relationship of the flow rates. The total flow rate $Q(s)$ is equal to the sum of the parts, $Q_1(s)$ and $Q_2(s)$.

$$Q(s) = Q_1(s) + Q_2(s) = \lambda Q(s) + (1 - \lambda)Q(s)$$

The constant λ represents the fraction of flow going to the first tank, while (1 – λ) is the remaining fraction of flow sent to the second tank.

$$\boldsymbol{H}_{v2} = \begin{bmatrix} \lambda \\ 1 - \lambda \end{bmatrix}$$

Figure 4-6 illustrates how the models become one using a series connection, where the output of the first system H_{v1} connects to the input of the second system H_{v2}.

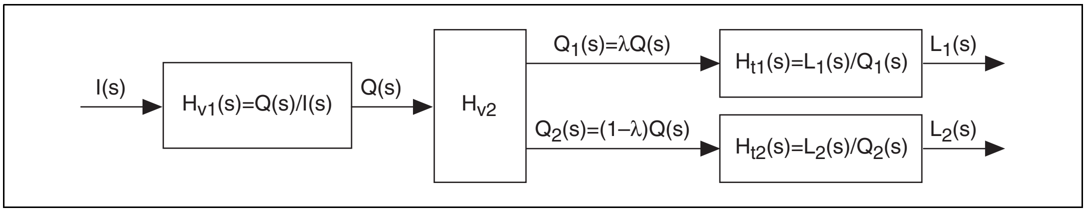


Figure 4-6. Two Valve Models and Two Tank Models in Series

The combined system consists of three blocks in series, and it has one input $I(s)$ and two outputs, $L_1(s)$ and $L_2(s)$.

The LabVIEW block diagram in Figure 4-7 illustrates the series connection where the models of Valve 1 and Valve 2 are connected in series. The two valves also are in series with the tanks, which are represented as a single model.

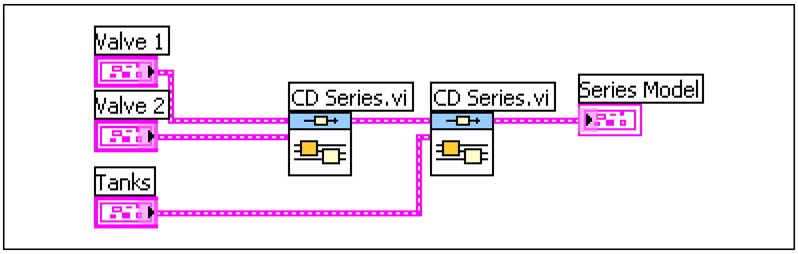


Figure 4-7. Block Diagram of the Two Valves and Tanks in Series

Appending Models

When you want to compare the time or frequency response of the two models in the same plot, you can append the models. The CD Append VI connects two models to produce an augmented model that contains all inputs and outputs of both models. With state-space models, states of the first model are combined with states of the second model.

Figure 4-8 shows two appended system models.

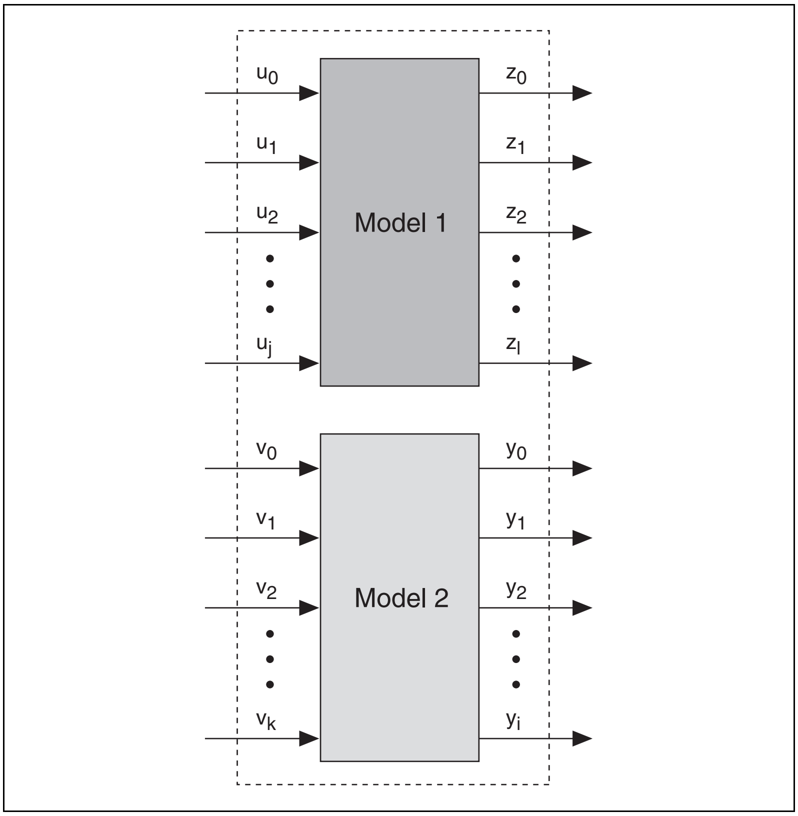


Figure 4-8. Appended Models

Consider the case of the two tanks from the *SIMO Systems in Series* section of this chapter. The transfer functions of the tanks are defined by the following equations.

$$H_{t1} = \frac{K_1}{s} \qquad H_{t2} = \frac{K_2}{s}$$

Appending the two transfer functions, H_{t1} and H_{t2}, results in the following appended matrix transfer function $\boldsymbol{H}_t$.

$$\boldsymbol{H}_t = \begin{bmatrix} H_{t1} & 0 \\ 0 & H_{t2} \end{bmatrix}$$

Figure 4-9 uses the block diagram from Figure 4-7 and replaces the **Tanks** input with the appended tank models. Again, the two valves are connected in series with each other and also in series with the two tanks.

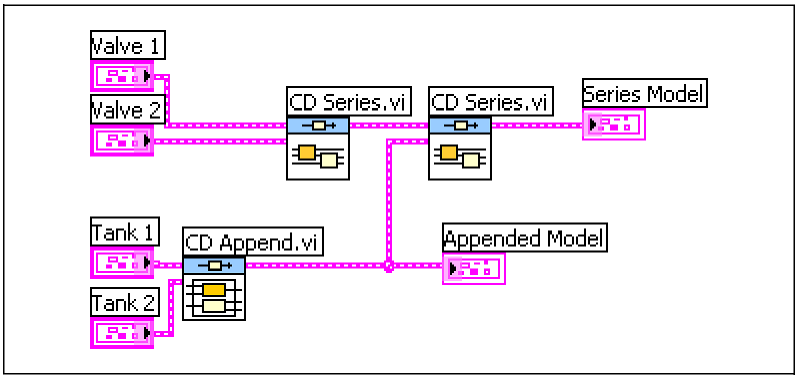


Figure 4-9. Appending the Two Tanks

Connecting Models in Parallel

If you want to create a single model from two separate subsystems that share common inputs, you must combine the subsystems in parallel. You also can use a parallel connection if you want to add or subtract outputs of two subsystems and represent them as a single output. To create a larger model of two subsystems in parallel, use the CD Parallel VI.

For example, consider the circuit system in Figure 4-10.

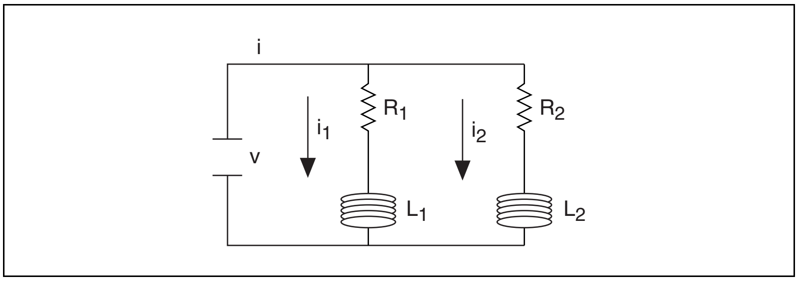


Figure 4-10. Circuit System

The input is the voltage *v*, and the output is the total current *i*. *i* is the sum of currents i_1 and i_2. The following equations describe the individual currents for the circuit system in Figure 4-10.

$$L_1\frac{di_1}{dt} + R_1 i_1 - v = 0$$

$$L_2\frac{di_2}{dt} + R_2 i_2 - v = 0$$

The resulting transfer functions for each circuit loop are given by the following equations.

$$H_1(s) = \frac{I_1(s)}{V(s)} = \frac{1}{L_1 s + R_1}$$

$$H_2(s) = \frac{I_2(s)}{V(s)} = \frac{1}{L_2 s + R_2}$$

In Figure 4-11, $H_1(s)$ and $H_2(s)$ represent the transfer functions defined in the previous equations, and $I_1(s)$ and $I_2(s)$ are their respective outputs.

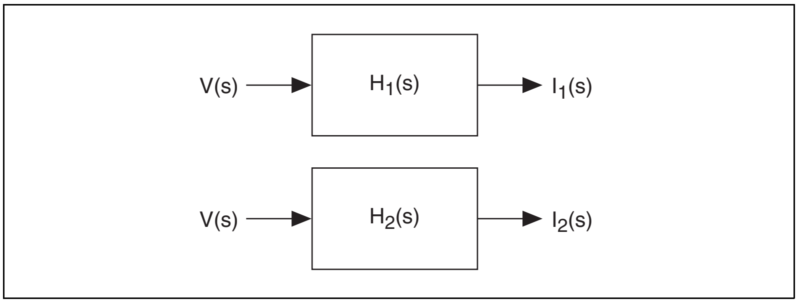


Figure 4-11. Each Circuit Loop in the Circuit System

Figure 4-12 illustrates the relationship between the voltage *v* and total current *i* by placing both models together in one larger system model. When the two models are in parallel, both models share the same input $V(s)$ and provide a total output $I(s)$ as shown in Figure 4-12.

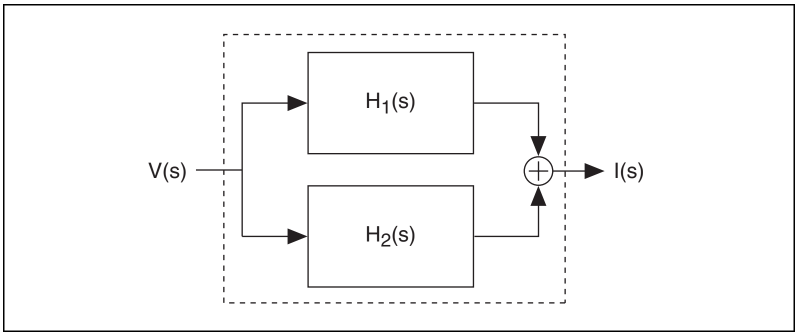


Figure 4-12. Entire Circuit System as a Parallel Model

The resulting transfer function is a second-order system described by the following equations.

$$I(s) = I_1(s) + I_1(s) = V(s)[H_1(s) + H_2(s)]$$

$$H(s) = \frac{I(s)}{V(s)} = H_1(s) + H_2(s)$$

Figure 4-13 illustrates how some inputs from Model 1 and Model 2 share the same inputs, and the outputs of Model 1 are added to or subtracted from the outputs of Model 2 to provide one combined parallel model.

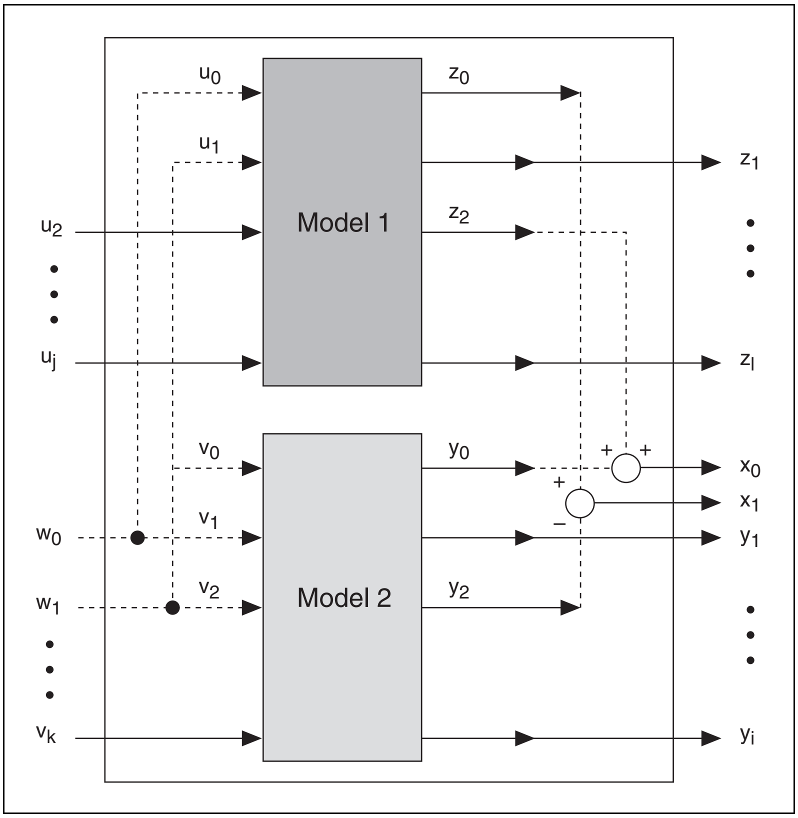
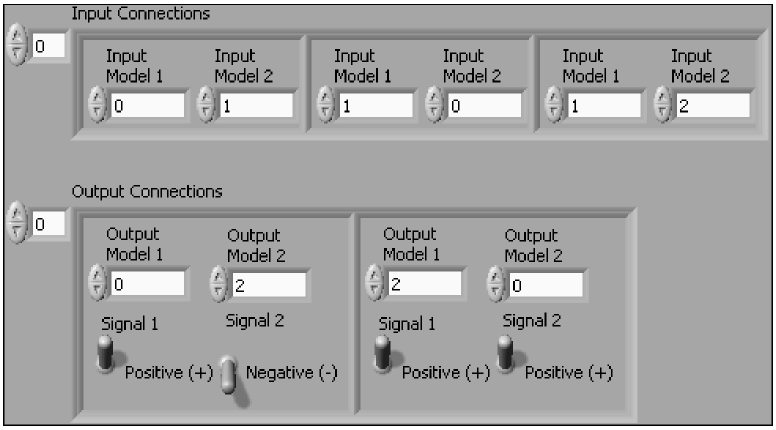


Figure 4-13. MIMO Models in Parallel

You can use the CD Parallel VI to define the relationship between the inputs and outputs of the models. Figure 4-14 displays the **Input Connections** and **Output Connections** controls needed to define the parallel interconnections shown in Figure 4-13.


Input Connections

	Input Model 1	Input Model 2
0	0	1
	1	0
	1	2

Output Connections

	Output Model 1	Output Model 2	Signal 1	Signal 2
0	0	2	Positive (+)	Negative (-)
	2	0	Positive (+)	Positive (+)

Figure 4-14. Connection Definitions for Models in Parallel

You can see how these controls indicate that the input for u_0 of Model 1 is the same as the input for v_1 of Model 2, and the input for u_1 of Model 1 is the same as the input for v_0 of Model 2, and so on. Furthermore, you can see how the y_2 output of Model 2 is subtracted from the z_0 output of Model 1. You also can see how the z_2 output of Model 1 is added to the y_0 output of Model 2. You define addition and subtraction by specifying the output as a **Positive (+)** or **Negative (–)** connection.

Placing Models in a Closed-Loop

Use the CD Feedback VI to place one or two models in a closed-loop configuration. The CD Feedback VI has two inputs, **Feedback Connections** and **Output Connections**, that define how to connect the outputs of a model to the inputs of the same model or a second model. **Feedback Connections** defines the connection between outputs of the first model and inputs of the second model. **Output Connections** defines the connection between outputs of the second model and inputs of the first model.

The following sections describe how the CD Feedback VI configures the closed-loop feedback when you have one or two models in the closed-loop configuration.

One Model in a Closed-Loop Configuration

When you only have one model in a closed-loop configuration, the CD Feedback VI connects the outputs to the inputs of the same model. The resulting model differs depending on the number of connections you define. The following sections describe the configuration of the model when you define and do not define connections.

Feedback Connections Undefined

In the first case, if you do not specify **Feedback Connections**, all outputs from Model 1 are fed back to the inputs of Model 1. When you do not specify **Feedback Connections**, the **Feedback Sign** input determines if these outputs are fed back negatively or positively. In the resulting model, shown in Figure 4-15, new reference inputs are created for each feedback connection.

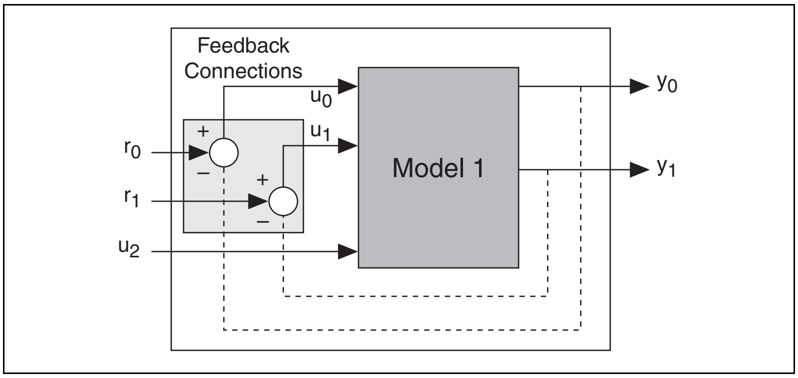


Figure 4-15. One Model with No Connections Defined

If you have an unequal number of inputs and outputs, the number of connections established is equal to the smaller number of inputs or outputs. The CD Feedback VI removes the remaining inputs or outputs. For example, if you have *m* inputs and *r* outputs in Model 1, where $m < r$, you have *m* number of reference inputs.

Similarly, if you have *m* inputs and *r* outputs in Model 1, where $r < m$, you have *r* number of reference inputs. The resulting model does not contain the the original inputs $u_{r+1} \ldots u_m$.

Feedback Connections Defined

If you specify **Feedback Connections**, each specified output in **Feedback Connections** is fed back to its corresponding input. You also define whether the connection is positive or negative. When you specify **Feedback Connections**, the CD Feedback VI ignores the **Feedback Sign** input, which specifies if the all outputs are fed back negatively or positively.

In the resulting model, shown in Figure 4-16, you can see how the CD Feedback VI creates new reference inputs for each feedback connection you specified.

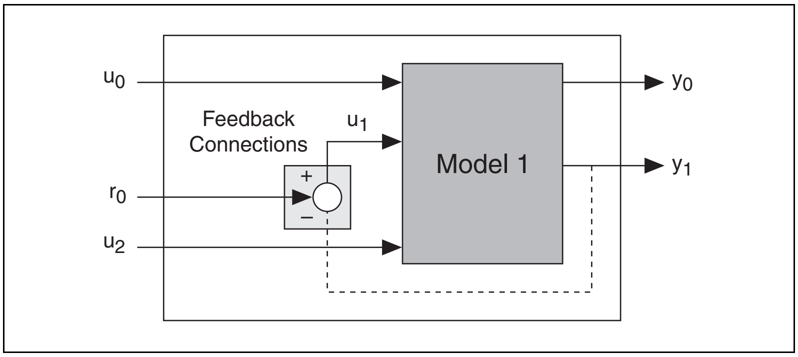


Figure 4-16. One Model with Connections Defined

If you have an unequal number of inputs and outputs, the number of connections established is equal to the smaller number of inputs or outputs. The CD Feedback VI removes the remaining inputs or outputs.

Two Models in a Closed-Loop Configuration

When you have two models in a closed-loop configuration, the first model is always in the open-loop path, while the second model is in the feedback path. Therefore, you now have the option to define both feedback and output connections when configuring a closed-loop system. By default, the CD Feedback VI uses negative feedback to connect the models.

The resulting model differs depending on the number of connections you define. The following sections describe the configuration of the models when you define and do not define connections.

Output and Feedback Connections Undefined

If you do not specify **Output Connections** or **Feedback Connections**, the CD Feedback VI tries to connect all the outputs of Model 1 to the inputs of Model 2. The CD Feedback VI also tries to connect all the outputs of Model 2 to the inputs of Model 1. When you do not specify **Feedback Connections**, the **Feedback Sign** input determines if these outputs are fed back negatively or positively.

In the resulting model, shown in Figure 4-17, you can see how the CD Feedback VI creates new reference inputs for each feedback connection you specified.

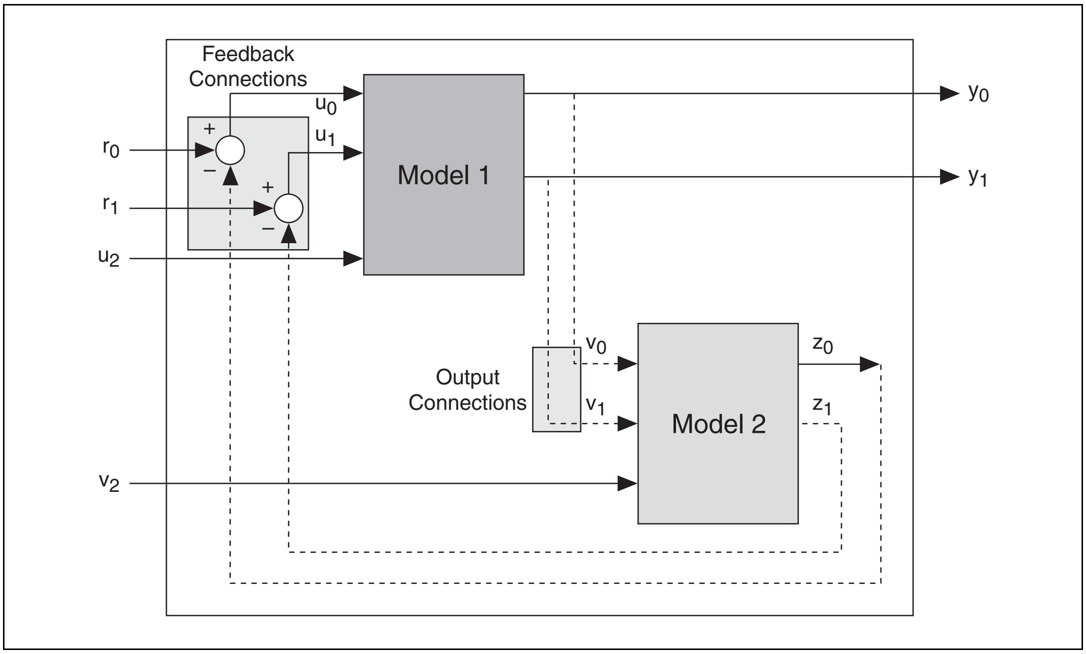


Figure 4-17. Two Models with No Connections Defined

If you have an unequal number of inputs and outputs, the number of connections established is equal to the smaller number of inputs or outputs. The CD Feedback VI removes the remaining inputs or outputs.

Output Connections Defined, Feedback Connections Undefined

If you specify a connection in **Output Connections** but not in **Feedback Connections**, the specified outputs for Model 1 are connected to the specified inputs for Model 2. You define whether the connection is positive or negative. Also, because you do not specify a connection in **Feedback Connections**, all outputs of Model 2 are connected to the inputs in Model 1 based on the **Feedback Sign**.

In the resulting model, shown in Figure 4-18, you can see how the CD Feedback VI creates new reference inputs for each feedback connection you specified.

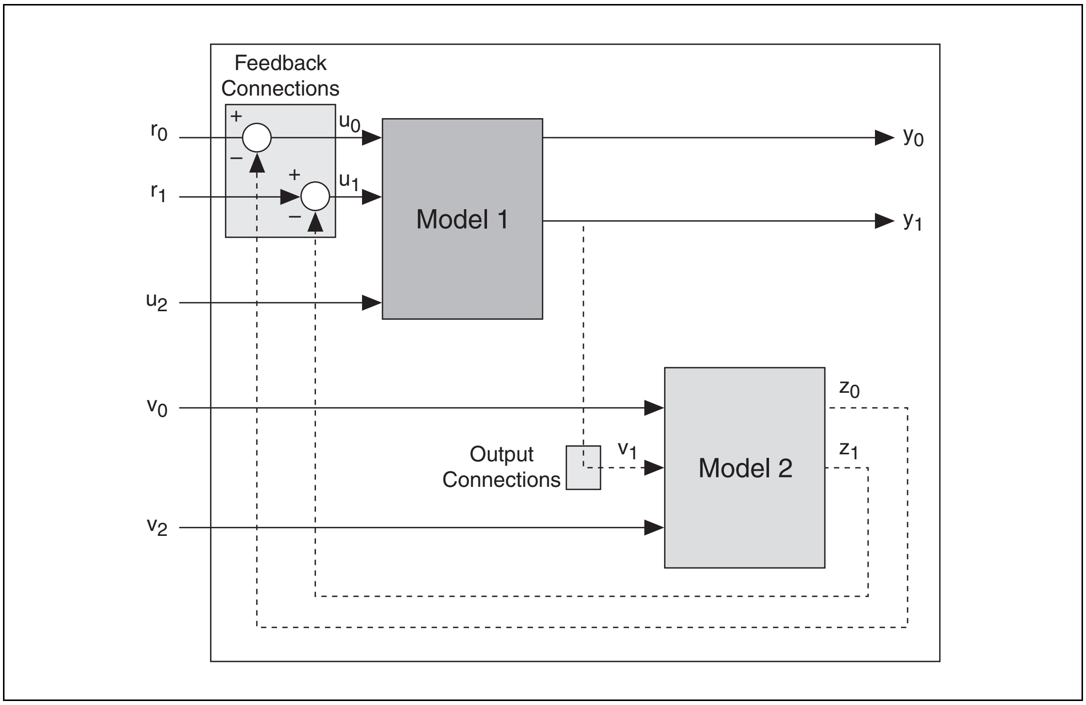


Figure 4-18. Two Models with Model 1 Connections Defined

If you have an unequal number of inputs and outputs, the number of connections established is equal to the smaller number of inputs or outputs. The CD Feedback VI removes the remaining inputs or outputs.

Output Connections Undefined, Feedback Connections Defined

If you specify a connection in **Feedback Connections** but not in **Output Connections**, the outputs specified for Model 2 are fed back to the specified inputs for Model 1. You define whether the feedback connection is positive or negative. Because you do not specify a connection in **Output Connections**, the CD Feedback VI tries to connect all outputs of Model 1 positively to the inputs in Model 2.

In the resulting model, shown in Figure 4-19, you can see how the CD Feedback VI creates new reference inputs for each feedback connection you specified.

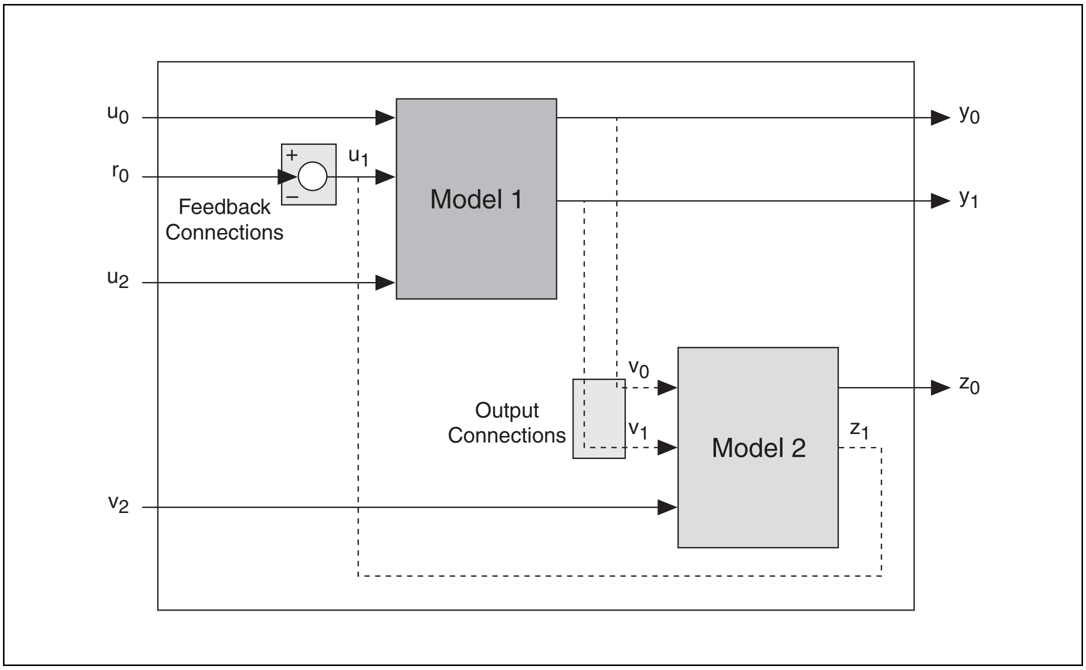


Figure 4-19. Two Models with Model 2 Connections Defined

If you have an unequal number of inputs and outputs, the number of connections established is equal to the smaller number of inputs or outputs. The CD Feedback VI removes the remaining inputs or outputs.

Both Output and Feedback Connections Defined

If you specify connections in both **Output Connections** and in **Feedback Connections**, you define all connections. Based on the connections you specified in **Output Connections**, the outputs specified for Model 1 are connected to the inputs specified for Model 2. You define whether the connection is positive or negative.

Based on the connections you specified in **Feedback Connections**, the outputs specified for Model 2 are fed back to the inputs specified for Model 1. You also define whether the feedback connection is positive or negative. Outputs of Model 2 not specified in **Feedback Connections** are removed from the resulting model. Again, because you specified connections using the **Feedback Connections**, the CD Feedback VI ignores the **Feedback Sign** input.

In the resulting model, shown in Figure 4-20, you can see how the CD Feedback VI creates new reference inputs for each feedback connection you specified.

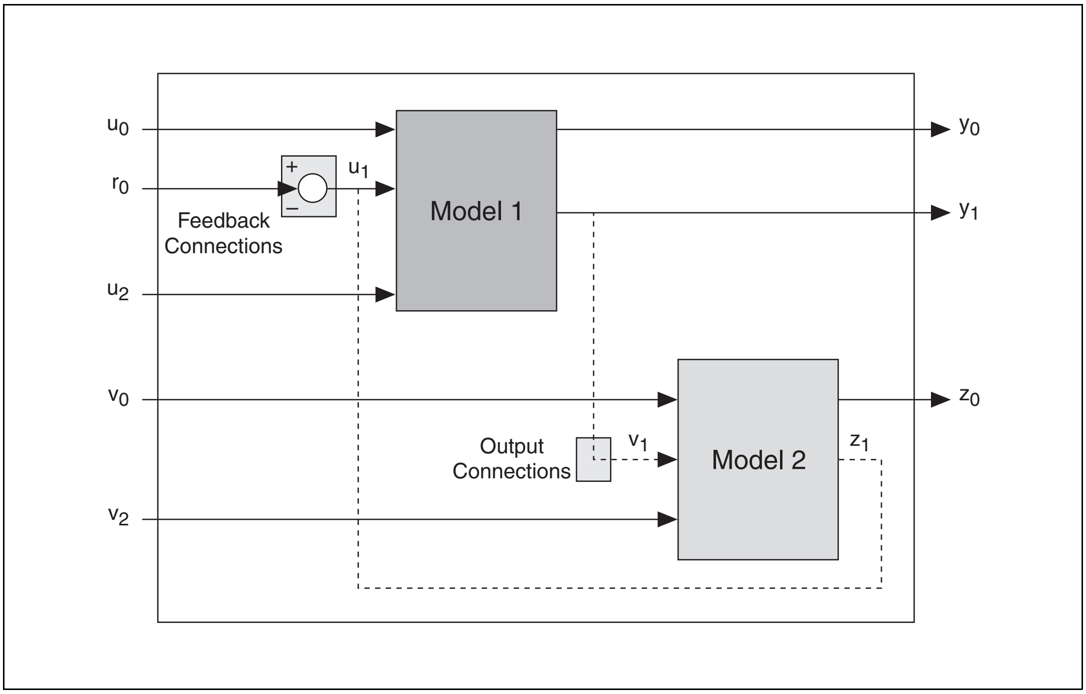


Figure 4-20. Both Model Connections Defined

If you have an unequal number of inputs and outputs, the number of connections established is equal to the smaller number of inputs or outputs. The CD Feedback VI removes the remaining inputs or outputs.

5 Creating a Model Delay

In control design, you must adjust the inputs of the plant to make the outputs promptly reach a setpoint. Delays in the plant strongly influence the performance of the controller by delaying the output response. If you want to design a controller that takes into account all factors that influence the output response, you must account for delays in the model.

For example, models of chemical plants have time delays representing the time involved in transporting fluids or materials between the process equipment, the actuators, and the sensors. Time delays can strongly influence the system and affect the stability of a system. Therefore, to account for all factors affecting the outputs of a system, you must include delays in the system model. By adding delay information to a model, you can see the effects of the delay in the frequency and time response graphs.

Use the CD Convert Delay with Pade Approximation VI for continuous models or the CD Convert Delay to Poles at Origin VI for discrete models to incorporate the delay information directly into the mathematical model. If you incorporate the delays in the model using one of these two VIs, the Dynamic Characteristics VIs and the State Feedback Design VIs account for the delays in their calculations. Refer to the *Delays in a Continuous System Model* section and the *Delays in a Discrete System Model* section of this chapter for information about adding delay information to a model.

Note Some VIs in the LabVIEW Control Design Toolkit support delays. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, to determine which VIs support delays.

Incorporating Delay Information into a Model

Each system model, as described in the *LabVIEW Control Design Toolkit Model Representations* section of Chapter 2, *Creating Dynamic System Models*, contains a mathematical model and a set of properties that provide information about the dynamic system. You can specify delay information about the dynamic system using these properties.

These properties include input, output, and transport delays. The input delay is the time it took a past input to affect the current output. The output delay is the time it takes the output to respond to the current input. The transport delay is a delay between the excitation of a system and the response of the system to that excitation. The total delay of the system includes the input, output, and transport delay.

The transfer function $H(s)$ represents the system. e^{-st_d} represents the delay factor. $H(s)e^{-st_d}$ defines a system that has a delay.

$$H(s)e^{-st_d} = \frac{Y(s)}{U(s)}$$

You also can represent the delay as an input delay or output delay. An input delay occurs if you apply the delay factor e^{-st_d} to the input $U(s)$. An output delay occurs if you apply the delay factor to the output $Y(s)$.

$$H(s) = \frac{Y(s)}{e^{-st_d}U(s)} \qquad \text{input delay}$$

$$H(s) = \frac{e^{st_d}Y(s)}{U(s)} \qquad \text{output delay}$$

Figure 5-1 shows how the delay can be an input delay or an output delay.

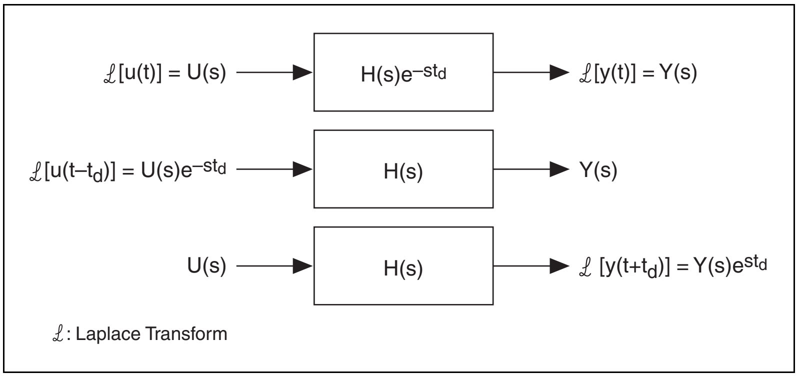


Figure 5-1. Mathematical Representation of Input and Output Delay Factors

Delays in a Continuous System Model

The mathematical representation of the continuous transfer function or zero-pole-gain model is a rational polynomial function $H(s)$. This function does not include the delay. The mathematical representation of a continuous state-space model does not include the delay as part of the state information.

To mathematically represent a delay in the system model, you must multiply the model with the delay factor e^{-st_d}. The delay factor e^{-st_d} is an exponential factor that represents the continuous model delay in the Laplace domain. Even though you multiply the model by the delay factor, the delay is not part of the rational polynomial function because e^{-st_d} is an exponential factor. Using the Padé approximation method, you can convert the delay factor into a rational polynomial function.

The Padé approximation method is a rational polynomial approximation of the delay factor. If you use the Padé approximation method to incorporate the delay directly into the model, the resulting mathematical model has no exponential terms. Connecting models that contain all rational polynomial functions is easier than connecting models that contain exponential factors and rational polynomial functions.

To calculate a Padé approximation, use the CD Convert Delay with Pade Approximation VI. When you use the CD Convert Delay with Pade Approximation VI, the Control Design Toolkit incorporates the delay information of the input model into the resulting mathematical model. Subsequently, the Control Design Toolkit removes the delay information from the resulting model properties.

Note The CD Convert Delay with Pade Approximation VI converts a state-space model into a transfer function representation before incorporating the delay information. Then, the CD Convert Delay with Pade Approximation VI converts the representation back to a state-space model. Therefore, the resulting states might not correspond directly to the original states of the model.

For example, consider the step response of a second-order system with a 25 second input delay, as shown in Figure 5-2. Refer to the *Step Response of a System* section of Chapter 6, *Time Responses*, for information about step responses.

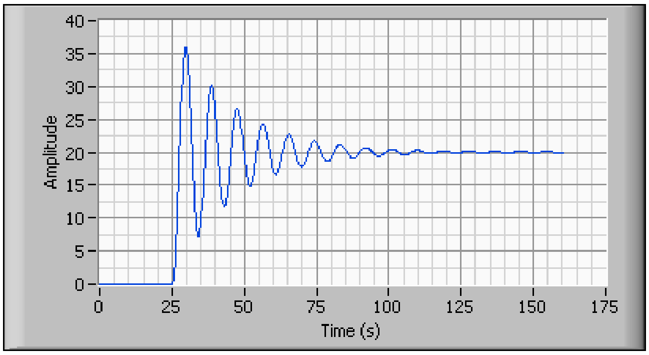


Figure 5-2. Step Response with a 25 Second Delay

The system in Figure 5-2 reflects a model whose mathematical representation includes a delay factor e^{-25s}.

When you use the Padé approximation method to incorporate the delay directly into the mathematical model, the CD Convert Delay with Pade Approximation VI transforms the delay to a rational polynomial function. The larger the polynomial order you specify, the more accurate the approximation, as shown in Figure 5-3.

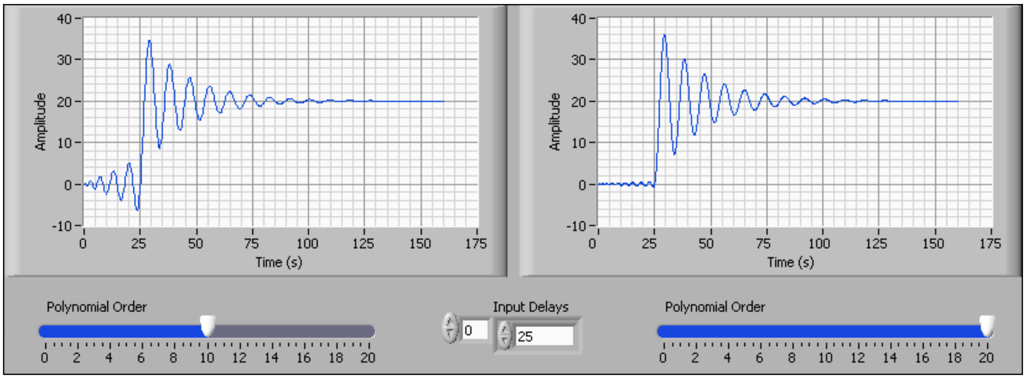


Figure 5-3. Effect of Polynomial Orders for a Padé Approximation

Delays in a Discrete System Model

To represent a delay in a discrete system model, you must apply the delay factor z^{-N_d} to the mathematical model. In transfer function models and zero-pole-gain models, incorporating delays means adding poles at the origin. By applying z^{-N_d} to a transfer function or zero-pole-gain model, you increase the order of the denominator polynomial with N_d poles at the origin. In state-space models, you can incorporate delays by creating N_d additional states.

Note N_d equals the delay divided by the sampling time. For example, if the sampling time specified in the model information equals 1 millisecond and the delay is equal to 10 milliseconds, N_d equals 10.

To incorporate delays in discrete models, use the CD Convert Delay to Poles at Origin VI. After incorporating the delay information of the input model into the resulting mathematical model, the Control Design Toolkit removes the delay information from the resulting model properties.

Figure 5-4 shows how you can add an input delay to a transfer function model and then use the CD Convert Delay to Poles at Origin VI to incorporate the input delay into the mathematical model.

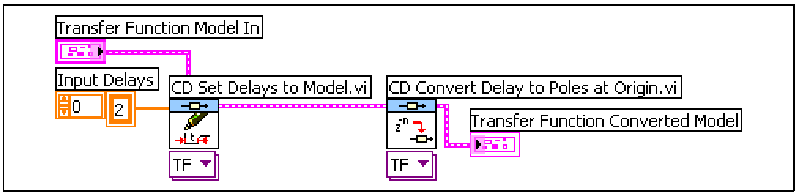
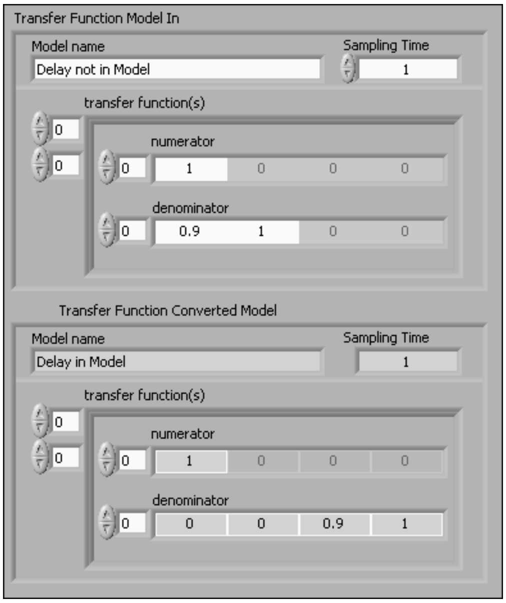

Transfer Function Model In
Input Delays
0
2
CD Set Delays to Model.vi
TF
CD Convert Delay to Poles at Origin.vi
TF
Transfer Function Converted Model

Figure 5-4. Adding Delays to a Transfer Function Model

Figure 5-5 shows the resulting transfer function model. The **Transfer Function Converted Model** has a larger order denominator than the **Transfer Function Model In** because the CD Convert Delay to Poles at Origin VI accounted for the input delay by increasing the number of poles at the origin in the model.



Transfer Function Model In

Model name: Delay not in Model
Sampling Time: 1

transfer function(s): 0, 0

numerator: 0 | 1 | 0 | 0 | 0

denominator: 0 | 0.9 | 1 | 0 | 0

Transfer Function Converted Model

Model name: Delay in Model
Sampling Time: 1

transfer function(s): 0, 0

numerator: 0 | 1 | 0 | 0 | 0

denominator: 0 | 0 | 0 | 0.9 | 1

Figure 5-5. Additional Poles Accounting for the Input Delay

The **Transfer Function Converted Model** expresses the additional poles at the origin with two additional zeros in the **denominator**.

Manipulating Delay Information

The Control Design Toolkit provides the CD Distribute Delay VI and CD Total Delay VI, which enable you to manipulate the delay properties of a dynamic system. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about these VIs.

Using the CD Distribute Delay VI and CD Total Delay VI, you can transfer delay values among the input, output, and transport delay properties. The CD Total Delay VI transfers all input and output delay values to the transport delay property. The CD Distribute Delay VI initially applies the total delay calculation and later distributes delays between inputs and outputs such that the non-zero elements of the transport delay matrix are minimized.

To illustrate how you manipulate the delay properties, consider the following equation.

$$\boldsymbol{Y}_d = \boldsymbol{H}_d \cdot \boldsymbol{U}_d$$

$\boldsymbol{H}_d$ is a 2 × 2 continuous matrix transfer function with delay information. $\boldsymbol{U}_d$ is the input vector with delay information. $\boldsymbol{Y}_d$ is the output vector with delay information. You also can represent the transfer function matrix $\boldsymbol{H}_d$ by an element-by-element product of $\boldsymbol{H}$ and $\boldsymbol{T}_d$.

$$\boldsymbol{H}_d = \begin{bmatrix} H_{11}e^{-st_{11}} & H_{12}e^{-st_{12}} \\ H_{21}e^{-st_{21}} & H_{22}e^{-st_{22}} \end{bmatrix} = \boldsymbol{H} \cdot \ \boldsymbol{T}_d$$

The following expressions represent the matrix transfer function $\boldsymbol{H}$ and the transport delay matrix $\boldsymbol{T}_d$.

$$\boldsymbol{H} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \quad \boldsymbol{T}_d = \begin{bmatrix} e^{-st_{11}} & e^{-st_{12}} \\ e^{-st_{21}} & e^{-st_{22}} \end{bmatrix}$$

The transport delay matrix $\boldsymbol{T}_d$ contains model delay information for each input-output pair. The following expression is the simplified way of representing the transport delay matrix.

$$\boldsymbol{T}_d \equiv \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}$$

Notice that the transport delay matrix $\boldsymbol{T}_d$ has the same dimension as the transfer function matrix $\boldsymbol{H}$.

The input delay intrinsically affects the time that each input takes to impact the system dynamics. Consider the following input vector with delay information $\boldsymbol{U}_d$.

$$\boldsymbol{U}_d \equiv \begin{bmatrix} U_1e^{-st_1} \\ U_2e^{-st_2} \end{bmatrix} = \boldsymbol{U} \cdot \ \boldsymbol{I}_d$$

You also can represent the input vector with delay information $\boldsymbol{U}_d$ by an element-by-element product of $\boldsymbol{U}$ and $\boldsymbol{I}_d$. The following expressions represent the input vector $\boldsymbol{U}$ and input delay vector $\boldsymbol{I}_d$.

$$\boldsymbol{U} \equiv \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \qquad \boldsymbol{I}_d \equiv \begin{bmatrix} e^{-st_1} \\ e^{-st_2} \end{bmatrix}$$

The input delay vector $\boldsymbol{I}_d$ contains input delay information for each input. The following expression is the simplified way of representing the input delay vector.

$$\boldsymbol{I}_d \equiv \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

Similarly, the output delay is the time that it takes the system dynamics to affect an output. Consider the following output vector with delay information $\boldsymbol{Y}_d$.

$$\boldsymbol{Y}_d \equiv \begin{bmatrix} Y_1 e^{st_a} \\ Y_2 e^{st_b} \end{bmatrix} = \boldsymbol{Y} \cdot \ \boldsymbol{O}_d$$

You also can represent $\boldsymbol{Y}_d$ by an element-by-element product of $\boldsymbol{Y}$ and $\boldsymbol{O}_d$. The following expressions represent the output vector $\boldsymbol{Y}$ and output delay vector $\boldsymbol{O}_d$.

$$\boldsymbol{Y} \equiv \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \qquad \boldsymbol{O}_d \equiv \begin{bmatrix} e^{st_a} \\ e^{st_b} \end{bmatrix}$$

The output delay vector $\boldsymbol{O}_d$ contains output delay information for each output. The following expression is the simplified way of representing the output delay vector.

$$\boldsymbol{O}_d \equiv \begin{bmatrix} t_a \\ t_b \end{bmatrix}$$

Because the number of rows and columns of $\boldsymbol{T}_d$ are the same as the dimension of vectors $\boldsymbol{I}_d$ and $\boldsymbol{O}_d$, you can represent all the delay information of a model using the following structure.

$$\begin{matrix} \begin{bmatrix} t_1 & t_2 \end{bmatrix} & \\ \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} & \begin{bmatrix} t_a \\ t_b \end{bmatrix} \end{matrix}$$

This representation is useful in understanding how the CD Total Delay VI and the CD Distribute Delay VI manipulate the delay information in a system. For example, consider a model with the following delay information.

$$\begin{matrix} \begin{bmatrix} t_1 & t_2 \end{bmatrix} & \\ \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} & \begin{bmatrix} t_a \\ t_b \end{bmatrix} \end{matrix} = \begin{matrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \\ \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{matrix}$$

The CD Total Delay VI transfers all the delay information into the transport delay matrix by adding the input and output delays to the delay in the transport delay matrix.

$$\begin{matrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \\ \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{matrix} \approx \begin{matrix} \begin{bmatrix} 0 & 0 \end{bmatrix} & \\ \begin{bmatrix} 3 & 3 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{matrix} \approx \begin{matrix} \begin{bmatrix} 0 & 0 \end{bmatrix} & \\ \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{matrix}$$

The CD Distribute Delay VI initially applies total delay and then uses a common delay factor to distribute the total delay between the inputs and outputs. The following equation represents the resulting delay information after distributing the total delay.

$$\begin{array}{cc} \begin{bmatrix}0 & 0\end{bmatrix} & \\ \begin{bmatrix}4 & 4\\ 4 & 4\end{bmatrix} & \begin{bmatrix}0\\ 0\end{bmatrix}\end{array} \approx \begin{array}{cc} \begin{bmatrix}4 & 4\end{bmatrix} & \\ \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} & \begin{bmatrix}0\\ 0\end{bmatrix}\end{array}$$

Figures 5-6 and 5-7 show you how to use the Control Design Toolkit to implement this example. The block diagram incorporates the delay information into a model and distributes the total delay into input, output, and transport delays.

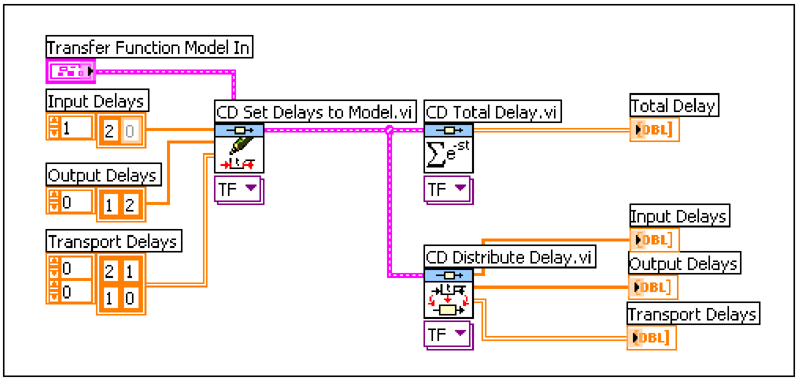
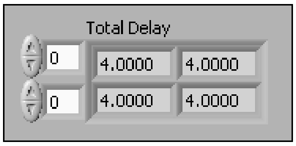


Figure 5-6. Incorporating Delay Information into a Model

Figure 5-7 displays the resulting total delay, which results when the CD Total Delay VI transfers the input and output delay information to the transport delay matrix.


Total Delay

0	4.0000	4.0000
0	4.0000	4.0000

Figure 5-7. Resulting Total Delay

Figure 5-8 displays the resulting input, output, and transport delays, which result when the CD Distribute Delay VI distributes the transport delay information to the input and output delay matrices.

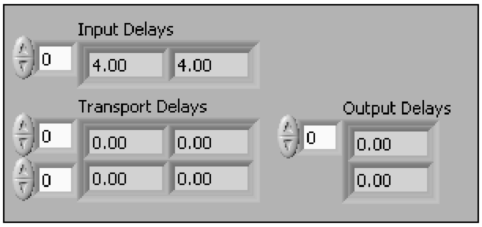


Figure 5-8. Resulting Delay Distribution

The CD Distribute Delay VI initially tries to distribute the delays to the inputs and then to the outputs. If a common delay factor does not exist among the inputs and outputs, then the CD Distribute Delay VI leaves some of the delay information in the transport delay matrix. For example, consider the following delay information.

$$\begin{matrix}\begin{bmatrix}0 & 0\end{bmatrix} & \\ \begin{bmatrix}5 & 3\\4 & 4\\3 & 5\end{bmatrix} & \begin{bmatrix}0\\0\\0\end{bmatrix}\end{matrix}$$

The CD Distribute Delay VI first distributes the delay in the transport delay matrix to the input delay vector by subtracting the minimum value, three, from each column in the transport delay matrix. Similarly, the CD Distribute Delay VI then distributes the delay to the output delay vector by subtracting the minimum value from each row in the resulting transport delay matrix.

$$\begin{matrix}\begin{bmatrix}0 & 0\end{bmatrix} & \\ \begin{bmatrix}5 & 3\\4 & 4\\3 & 5\end{bmatrix} & \begin{bmatrix}0\\0\\0\end{bmatrix}\end{matrix} \approx \begin{matrix}\begin{bmatrix}3 & 3\end{bmatrix} & \\ \begin{bmatrix}2 & 0\\1 & 1\\0 & 2\end{bmatrix} & \begin{bmatrix}0\\0\\0\end{bmatrix}\end{matrix} \approx \begin{matrix}\begin{bmatrix}3 & 3\end{bmatrix} & \\ \begin{bmatrix}2 & 0\\0 & 0\\0 & 2\end{bmatrix} & \begin{bmatrix}0\\1\\0\end{bmatrix}\end{matrix}$$

However, the CD Distribute Delay VI cannot fully distribute all the delays. So the transport delay matrix contains the residual delay information.

Note In multivariable systems, the total delay between an input and multiple outputs might be different. Therefore, a common input delay factor might not exist among outputs effected by the same input. In these cases, there might be residual transport delay information.

Some Control Design VIs distribute the delay information to preserve as much of the delay information as possible in the resulting model. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, to determine which VIs manipulate the transport delay matrix to preserve delay information.

6 Time Responses

Time-domain solutions, or time responses, provide information about the stability of a dynamic system and the performance of the controller. Also, in most control systems, you define the specifications in the time domain. Use the time-domain solutions to verify that the control system you design meets the specifications.

You can find the time-domain solution by integrating the system model. The LabVIEW Control Design Toolkit provides Time Response VIs to help you find these time-domain solutions. The Time Response VIs allow you to determine the response of a system to a standard set of inputs—step, impulse, and initial conditions. Also, you can determine the response of an arbitrary input by using the CD Linear Simulation VI. The Control Design Toolkit calculates either continuous or discrete time responses using an Euler solver.

This chapter describes time-domain solution and the standard set of inputs. This chapter also describes linear simulation, which is a general time-domain solution.

Time-Domain Solution

The Time Response VIs automatically convert transfer function and zero-pole-gain models to the state-space representation before calculating the time-domain solution. The following equation represents the continuous solution for a state-space model.

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}Bu(\tau)$$

x_0 denotes any initial conditions of the states in the model. $e^{At}x_0$ represents the solution of the model at the initial conditions. This solution is known as the free response.

$e^{A(t-\tau)}Bu(\tau)d\tau$ represents the solution of the model over time as the inputs $u(\tau)$ drive the dynamic system. This solution is known as the forced response.

The following equation represents the discrete solution for a state-space model.

$$(t_{i+1}) = e^{\boldsymbol{A}\Delta t}x(t_i) + \left[\int_0^{\Delta t} e^{\boldsymbol{A}\tau}\,d\tau\right]\boldsymbol{B}u(t_i$$

The first term denotes the discrete free response and the second term denotes the discrete forced response.

Spring-Mass Damper Example

To illustrate the different time responses you can obtain from a model, consider the following example of a spring-mass damper, shown in Figure 6-1.

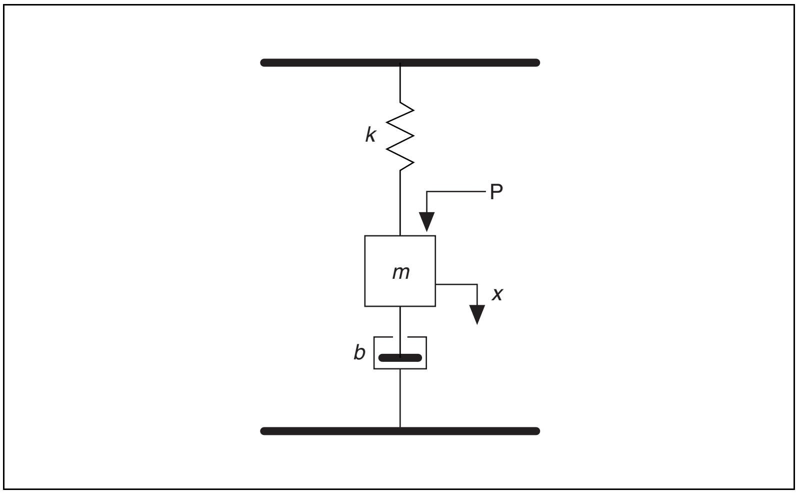


Figure 6-1. Spring-Mass Damper

In this example, k is the spring, m is the mass, and b is the damper coefficient. You can represent this spring-mass damper system with the following state-space model.

$$\dot{x} = \boldsymbol{A}x + \boldsymbol{B}u = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} x + \begin{bmatrix} 0 \\ -\frac{1}{m} \end{bmatrix} u$$

$$y = \boldsymbol{C}x + \boldsymbol{D}u = \begin{bmatrix} 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 \end{bmatrix} u$$

For this example, let $k = 50$, $m = 100$, and $b = 10$. The following matrices define the state-space model.

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -0.5 & -0.1 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad \boldsymbol{D} = \begin{bmatrix} 0 \end{bmatrix}$$

The following sections show how this system reacts to different inputs applied to the system.

Step Response of a System

Use CD Step Response VI to calculate the response of a dynamic system to a step input. The step response is how the dynamic system responds to this input signal. A unit step input is defined by the following equations.

$$y(t) = 0 \ \ if\ t < 0$$
$$y(t) = 1 \ \ if\ t \geq 0$$

The following list describes a set of common time-domain characteristics you can use to measure the performance of a dynamic system.

- Rise time (t_r)—The time required for the dynamic system response to rise from 10% of its final value to 90% of its final value.
- Maximum overshoot (M_p)—The dynamic system response value that most exceeds unity, expressed as a percent.
- Peak time (t_p)—The time required for the dynamic system response to reach the peak value of its first overshoot.
- Settling time (t_s)—The time required for the response to reach and stay within a 5% band of its final value.

The CD Step Response VI allows you to modify the limits for rise time and settling time. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about using the CD Step Response VI to calculate the step response of a dynamic system.

Figure 6-2 shows a standard step response and the corresponding response of the dynamic system.

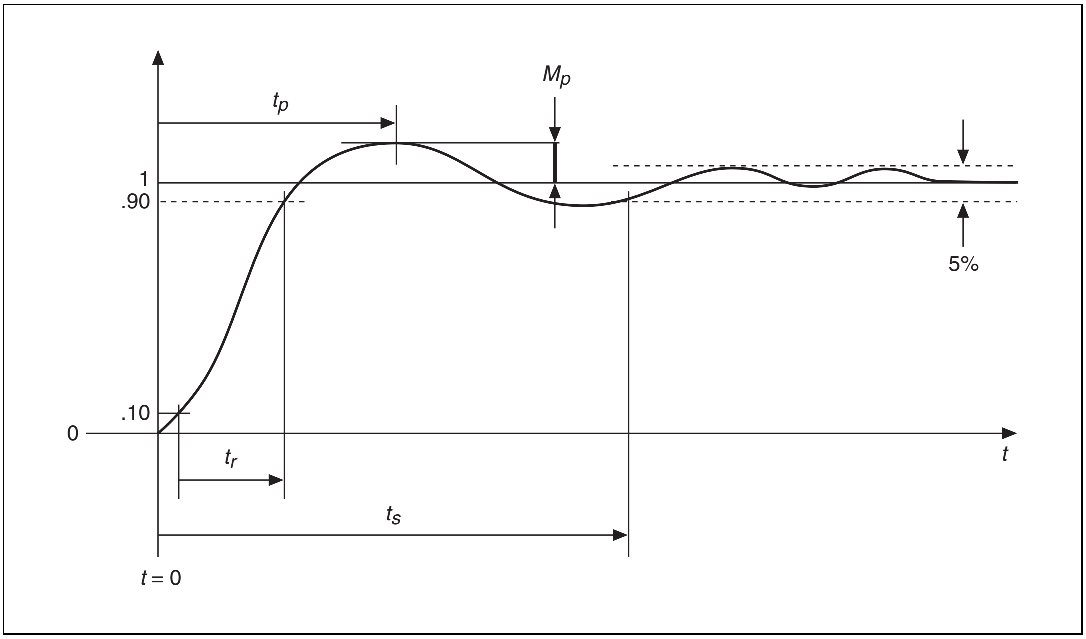


Figure 6-2. Step Response

By applying a step input to the system presented in the *Spring-Mass Damper Example* section of this chapter, you obtain the step response shown in Figure 6-3.

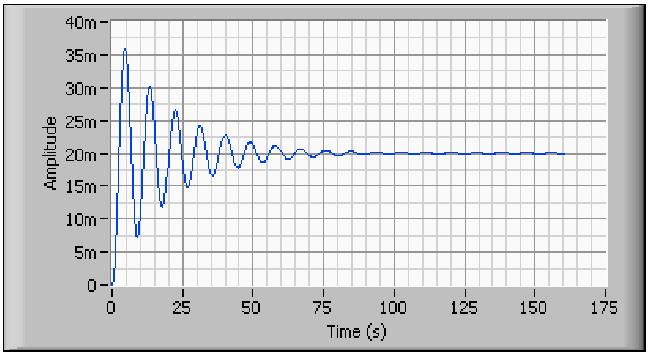


Figure 6-3. Step Response of the Spring-Mass Damper System

In Figure 6-3, you can see that the step input causes the system to settle at a new steady-state value of 20m. Based on the settling point, you can use the CD Parametric Time Response VI to calculate the rise time, maximum overshoot, peak time, and settling time. The rise time is 1.42 second, the percent overshoot is 79.90%, the peak time is 4.54 seconds, and the settling time is 89.89 seconds.

Impulse Response of a System

An impulse input to a dynamic system is defined differently depending on whether the system is discrete or continuous. For a continuous dynamic system, an impulse input is a unit-area signal with an infinite amplitude and infinitely small duration occurring at a specified time. At all other times, the input signal value is zero. For a discrete system, an impulse is a physical pulse that has unit amplitude at the first sample period and zero amplitude for all other times.

Use the CD Impulse Response VI to calculate the impulse response of a dynamic system to a standard impulse input, as shown in Figure 6-4.

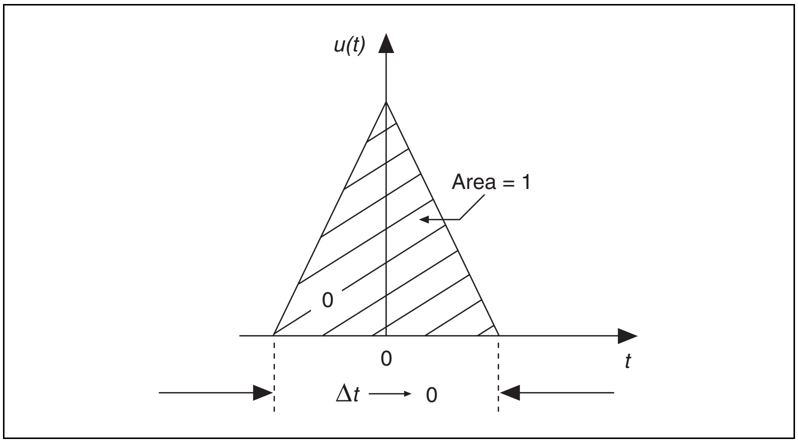


Figure 6-4. Impulse Response

Because the impulse signal excites all frequencies and the duration of this signal is infinitely small, you can see the natural response of the system.

By applying an impulse input to the system presented in the *Spring-Mass Damper Example* section of this chapter, you obtain the impulse response shown in Figure 6-5.

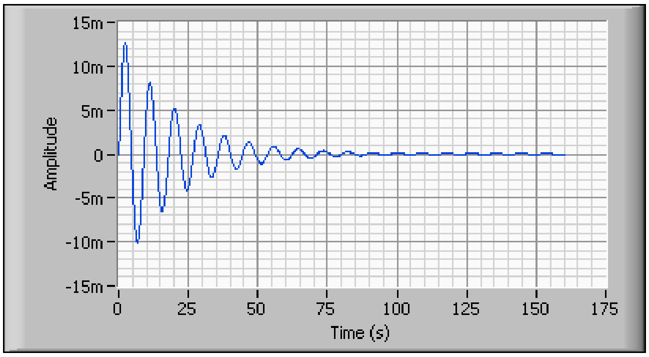


Figure 6-5. Impulse Response of the Spring-Mass Damper System

This graph shows the natural response of the spring-mass damper system. Because the impulse input is a unit-area signal with an infinite amplitude and infinitely small duration occurring at a specified time, the system eventually returns to its original steady-state value.

Initial Response of a System

You often assume that the states of a system have zero initial conditions. In many cases, however, you need to examine the response to a given set of non-zero initial conditions. A common control design goal is for this response to become zero or negligibly small as quickly as possible. The CD Initial Response VI determines this response.

By specifying a non-zero initial condition for the system presented in the *Spring-Mass Damper Example* section of this chapter, you obtain the initial response shown in Figure 6-6.

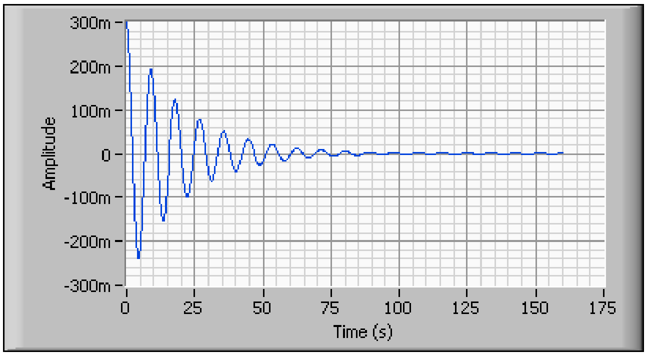


Figure 6-6. Initial Response of the Spring-Mass Damper System

The initial conditions start the system at an amplitude of 300m. Based on the response of the system to these initial conditions, you can see how long the system takes to return to the original steady-state value.

General Time-Domain Simulation

When creating a model of a dynamic system and trying to determine its response to input values, you generally want to simulate system behavior with more general input signals than the step, impulse, or initial inputs.

The CD Linear Simulation VI performs a general time-domain simulation by solving the following equations in response to an arbitrary input signal *u*.

Continuous Linear Simulation

$$x(t) = e^{\boldsymbol{A}t}x_0 + \int_0^t e^{\boldsymbol{A}(t-\tau)}\boldsymbol{B}u(\tau)$$

Discrete Linear Simulation

$$(t_{i+1}) = e^{\boldsymbol{A}\Delta t}x(t_i) + \left[\int_0^{\Delta t} e^{\boldsymbol{A}\tau}d\tau\right]\boldsymbol{B}u(t_i$$

This VI takes a continuous or discrete system model as input and the specified input signal. The VI then determines the response by integrating the dynamic equations at the specified time steps. If the system is discrete, the integration time step Δ*t* must equal the sampling period of the system. If the system is continuous, the Control Design Toolkit converts the continuous system to a discrete system using either an exponential zero-order-hold or first-order-hold method, with the sampling interval set equal to the integration time step.

Note For accurate results, verify that the sampling interval is small enough that the effects of converting the continuous system to a discrete system are negligible.

By using a square wave as the input signal to the system presented in the *Spring-Mass Damper Example* section of this chapter, you obtain the response shown in Figure 6-7.

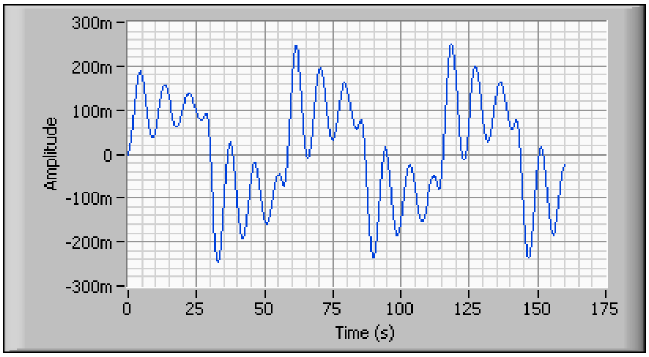


Figure 6-7. Linear Simulation of the Spring-Mass Damper System

This linear simulation of the system shows you how the system reacts to a square wave input. You can specify any input using the CD Linear Simulation VI and observe how the system behaves.

Time Response Data

The Time Response VIs return time response data, which contains information about the time response of all input-output pairs in the model. To access this information, use the CD Get Time Response Data VI. The CD Get Time Response VI enables you to customize how you want to view this information.

Using the CD Get Time Response Data VI, you can obtain the time response data of a certain input-output pair in an array or waveform format. You also can obtain the time response data of several input-output pairs in a table format or all the input-output pairs in an array format. For state-space models, the CD Get Time Response Data VI enables you to get the time response of the input-state pair(s). Transfer function and zero-pole-gain models do not have states, so the time response data for an input-state pair is an empty array.

Refer to the *LabVIEW Help* for more information about using the CD Get Time Response Data VI.

7

Frequency Response

The frequency response of a dynamic system is the output of a system given a unit-amplitude, zero-phase sinusoidal input. When applied to the system, a sinusoidal input with unit amplitude, zero phase, and frequency ω produces the following sinusoidal output.

$$H(j\omega) = A(\omega)e^{j\phi(\omega)}$$

A is the magnitude of the response as a function of ω, and ϕ is the phase. The magnitude and phase of the system output varies depending on the values of the system poles, zeros, and gain. In many engineering applications, the system poles and zeros are not precisely known. By experimentally determining the frequency response, you can determine the locations of the poles and zeros. Using information about the poles and zeros, you can design a compensator to improve undesirable parts of the frequency response.

You often use Bode, Nichols, and Nyquist plots to display the frequency response of a system. The LabVIEW Control Design Toolkit provides Frequency Response VIs that allow you to visualize the response using these frequency analysis methods.

Bode Frequency Analysis

Use Bode plots of system frequency responses to assess the relative stability of a closed-loop system given the frequency response of the open-loop system. The open-loop system must be stable and have a minimum phase for Bode frequency analysis. A stable system that has a minimum phase is a system that has no right-half plane poles or zeros. You can compute this response directly using the CD Bode VI, which provides insight into what the open- and closed-loop frequency responses of a system imply about the system behavior.

Note To determine the frequency response at specified values, use the CD Evaluate at Frequency VI.

You can separate the following complex frequency response equation into two parts—the magnitude, $A(\omega)$, and the phase, ϕ. Both parts are functions of the frequency.

$$H(j\omega) = A(\omega)e^{j\phi(\omega)}$$

You can obtain the magnitude from the absolute value of the response and the phase from the four-quadrant arctangent of the response.

The standard Bode plot has two subplots—the Bode magnitude plot and Bode phase plot. The Bode magnitude plot shows the decibel gain plotted against the logarithm of the frequency. The Bode phase plot shows the phase, in degrees, as a function of the logarithm of the frequency. For both the Bode magnitude and phase plots, you can use logarithmic frequency because you can display a wide range of frequency-dependent behaviors simultaneously.

Using logarithmic scales for the magnitude ratio makes manipulating the dynamic elements in a control loop easier. To obtain the magnitude ratio, you must multiply the individual magnitude ratios of the dynamic elements. When using a logarithmic scale, instead of multiplying the individual ratios, you can add the individual magnitude ratios to calculate the total magnitude response of the system. Similarly, because the phase information is on a linear scale, the phase angle is a sum of the individual phase elements.

Because you can add the magnitude and phase plots for systems in series, you can add Bode plots of an open-loop plant and potential compensators to determine the frequency-response characteristics of the complete system. Bode plots also illustrate system bandwidth as the frequency at which the output magnitude is reduced by three decibels or attenuated to approximately 70.7% of its original value. You also can use the CD Bandwidth VI to determine the system bandwidth.

You can measure how close a system is to instability by examining the value of the magnitude and phase at critical values. These values are the gain margin and the phase margin, which are important because real-life models are prone to uncertainties. Systems become unstable with gains that are too high or have too much phase lag.

Gain Margin

Bode plots provide an important aid to evaluate how stable or how close a closed-loop system is to instability. You often use this analysis on systems where $G(s)$ consists of a gain, K, and a dynamic model, $H(s)$, in series. For cases where increasing the gain leads to system instability, the system is stable for a given value of K only if the magnitude of $KH(s)$ is less than 0 dB at any frequency where the phase of $KH(s)$ is –180°.

The gain margin indicates how much you can increase the gain before the closed-loop system becomes unstable. This critical gain value, which causes instability, indicates the location of the closed-loop poles of the system on the imaginary axis. The Bode magnitude plot displays the gain margin as the number of decibels by which the gain exceeds zero when the phase equals –180°, as shown in Figure 7-1.

Phase Margin

The phase margin is the amount by which the phase exceeds –180° when the gain is equal to 0 dB. The phase margin also indicates how close a closed-loop system is to instability. A stable system must have a positive phase margin.

Figure 7-1 shows the Bode plots with corresponding gain and phase margins.

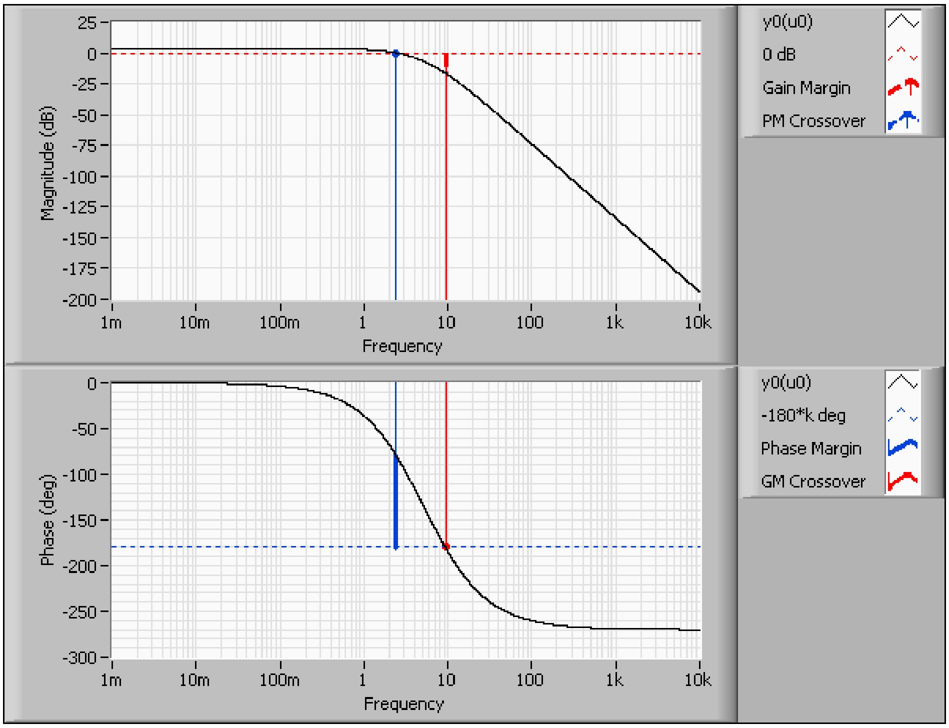


Figure 7-1. Gain and Phase Margins

Depending on the complexity of the system, a Bode plot might return multiple gain and/or phase margins.

Nichols Frequency Analysis

Use Nichols plots to obtain the closed-loop frequency response of a system from the open-loop response. Often open-loop response curves, or loci, of constant magnitude and phase augment a Nichols plot. Each point on the open-loop response curve corresponds to the response of the system at a given frequency. You then can read the closed-loop magnitude response at that frequency from the Nichols plot by identifying the value of the magnitude locus at which the point on the curve intersects. Similarly, you can determine the closed-loop phase by identifying the phase locus at which the open-loop curve crosses.

Use the CD Nichols VI to examine system performance in dynamic systems. The CD Nichols VI calculates and plots the open-loop frequency response against the gain and phase on the Nichols plot. Different points on the plot correspond to different values of the frequency, ω Using the CD Nichols VI, you can visually determine the gain and phase margins, bandwidth, and the effect of varying gains on the closed-loop system behavior.

Nyquist Stability Analysis

Use Nyquist stability analysis to examine the system performance of dynamic systems. Nyquist plots consist of the real part of the frequency response plotted against the imaginary part of the response. Nyquist plots indicate the stability of a closed-loop system, given an open-loop system, which includes a gain of *K*. Use the CD Nyquist VI to create a Nyquist plot.

The Nyquist stability criterion relates the number of closed-loop poles of the system to the open-loop frequency response. On the Nyquist plot, the number of encirclements of (–1, 0) is equal to the number of unstable closed-loop poles minus the number of unstable open-loop poles.

You can use this criterion to determine how many encirclements are required for closed-loop stability. For example, if the plant has all open-loop stable poles, there will be no encirclements. If the plant has one open-loop unstable pole, there will be one negative, counter-clockwise encirclement, as shown in Figure 7-2.

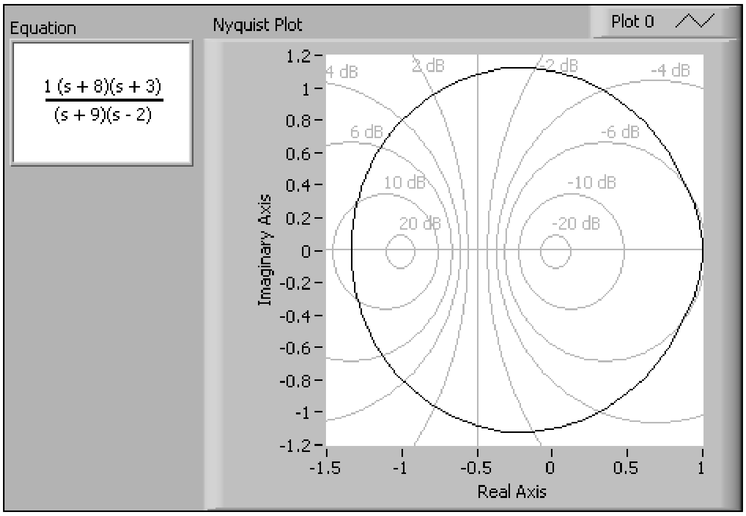


Figure 7-2. Nyquist Plot of One Unstable Pole

Often you want to determine a range of gain values for which the system is stable, rather than testing the stability of the system at a specific value of *K*. To determine the stability of a closed-loop system, you must determine how a range of gain values affect the stability of the system.

Consider the following closed-loop transfer function equation from *Y*(*s*) to *U*(*s*) where *K* is the gain value.

$$\frac{Y(s)}{U(s)} = \frac{KH(s)}{1 + KH(s)}$$

The closed-loop poles are the roots of the equation $1 + KH(s) = 0$. The complex frequency response of *KH*(*s*), evaluated for $s = j\omega$in continuous systems and $e^{j\omega T}$ for discrete systems, encircles (–1,0) in the complex plane if $1 + KH(s)$ encircles (0,0). If you examine the Nyquist plot of *H*(*s*), you can see that an encirclement of (–1/*K*,0) by *H*(*s*) is the same as an encirclement of (–1,0) by *KH*(*s*). Thus, you can use one Nyquist plot to determine the stability of a system for any and all values of *K*.

Frequency Response Data

The Frequency Response VIs return frequency response data that gives you information about the response of a certain input-output pair at a specific frequency. This response information varies depending on the VI you use to analyze the frequency response. The frequency response information for the CD Bode VI returns information about the Bode magnitude and Bode phase. The frequency response information for the CD Nichols VI returns information about the real and imaginary parts of the frequency response. Finally, the frequency response information for the CD Nyquist VI returns information about the open-loop gain and open-loop phase.

To access this information, use the CD Get Frequency Response Data VI. The CD Get Frequency Response Data VI enables you to obtain the frequency response data of a certain input-output pair in an array format. You also can obtain the frequency response data of several input-output pairs in a table format or all the input-output pairs in an array format. For state-space models, the CD Get Frequency Response Data VI enables you to get the frequency response of the input-state pair(s). Transfer function and zero-pole-gain models do not have states, so the frequency response data for an input-state pair is an empty array.

Refer to the *LabVIEW Help* for more information about using the CD Get Frequency Response Data VI.

8

Dynamic Characteristics of a System

The dynamic characteristics of a system include properties such as system stability, DC gain, damping ratio, natural frequency, and so on. This chapter describes the effects of poles and zeros on the stability and the response of a system to an input. This chapter also describes the Root Locus method that you can use to determine the stability of a system.

System Stability

The location of the system poles and zeros affects the stability of the system, so an effective feedback control design must take into account the closed-loop pole and zero locations. Consider the following equation that represents the open-loop transfer function, *H*(*s*).

$$H(s) = \frac{N(s)}{D(s)}$$

A stable system is one where the output is bounded for any bounded input or initial condition. This stability is known as bounded-input bounded-output stability. A continuous system is stable if and only if all poles of the system are negative, meaning all poles are in the left half of the complex plane. A discrete system is stable if and only if all poles are within the unit circle in the complex plane. A system also is stable if it does not contain any poles.

A continuous system is unstable if it contains at least one positive pole, meaning at least one pole is in the right half of the complex plane. A discrete system is unstable if at least one pole is outside of the unit circle in the complex plane. A system also is unstable if it contains more than one pole at the origin.

Continuous and discrete systems are marginally stable if they contain only one pole at the origin and no positive poles.

In terms of the dynamic response associated with the poles and zeros of a system, a pole is stable if the response it contributes decays over time. If the response becomes larger over time, the pole is unstable. If the response remains unchanged over time, the pole that causes the response is marginally stable. To describe a system as stable, all the closed-loop poles of a system must be stable.

You can use the CD Pole-Zero Map VI to obtain all the poles and zeros of a system and plot their corresponding locations on the complex plane. You can use the CD Stability VI to determine if a system is stable, unstable, or marginally stable.

Root-Locus Method

The Root-Locus method is based on an open-loop transfer function, which provides the closed-loop pole positions for all possible changes in a single variable K, or the loop gain.

You can rewrite the characteristic equation of a closed-loop system using the following equation.

$$1 + KH(s) = D(s) + KN(s) = 0$$

This equation restates the fact that the open-loop system poles, which correspond to $K = 0$, are the roots of the transfer function denominator, $D(s)$. As K becomes large, the roots of the previous characteristic equation approach the roots of $N(s)$, the zeros of the open-loop system, or infinity. For a closed-loop system with a nonzero, finite gain K, the solutions to the preceding equation are given by the values of s that satisfy both of the following conditions.

$$|KH(s)| = 1 \qquad \angle H(s) = \pm(2k+1)\pi \qquad (k = 0,\ 1,\ ..)$$

The root locus is a plot in the real-imaginary axis showing the values of s that correspond to pole locations for all gains, starting at the open-loop poles, $K = 0$ and ending at $K = \infty$.

Root locus plots provide an important indication of what gain ranges you can use while keeping the closed-loop system stable. Continuous systems are stable as long as their poles have a negative real part and are in the left half of the s-plane. Discrete systems are stable as long as their poles remain within the z-plane unit circle.

The CD Root Locus VI computes and draws root locus plots for continuous and discrete SISO state-space or transfer function systems. Refer to the *Root-Locus Design* section of Chapter 9, *Classical Control Design*, for information about how to use the CD Root Locus VI in synthesizing controller.

Additional Characteristics

The LabVIEW Control Design Toolkit provides Dynamic Characteristic VIs that you can use to compute characteristics such as the DC gain, damping ratio, and natural frequency. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about the VIs you can use to compute these characteristics.

9

Classical Control Design

Classical control design involves techniques for designing controllers in the frequency domain and the *s*-plane. Classical control design techniques involve modifying the frequency response of a system to achieve a desired response. These techniques are primarily applicable for linear time-invariant systems, though you can extend some of these techniques to nonlinear systems.

The process of designing a controller begins with specifying the control objective, such as the minimization of a cost function or the desired locations of the poles or zeros. You then select the architecture of the controller and synthesize a controller by selecting an appropriate set of parameters to satisfy the stated objectives.

This chapter describes various classical design techniques for designing controllers. You can use these techniques for SISO systems such as PID controllers and frequency-domain design techniques, such as the root-locus method. Refer to Chapter 12, *Modern Control Design Synthesis*, for information about modern control design techniques.

Root-Locus Design

You can use the root locus in the design of SISO systems by analyzing the variation of closed-loop pole positions for all possible changes in a controller variable. When the controller is specified in terms of more than one variable, the process of root locus must be done one variable at a time, fixing all other variables while one is being modified. The closed-loop zeros of a system, between any two points in the control loop, are a subset of the open-loop zeros and poles of the feedback element. This is what enables you to analyze the closed-loop behavior in terms of the value of a variable in the feedback transfer function.

For example, consider a system with the following open-loop transfer function.

$$H(s) = \frac{1}{(s+1)(s+2)(s+3)}$$

If a simple proportional feedback controller controls this system, the following equation describes the characteristic equation.

$$1 + H(s)K = 1 + \frac{K}{s^3 + 6s^2 + 11s + 6} = 0$$

The objective of root-locus analysis is to understand how the roots of this equation vary with respect to the gain *K* and to determine the optimal value of *K* for a given performance specification.

Figure 9-1 illustrates the root locus of this system.

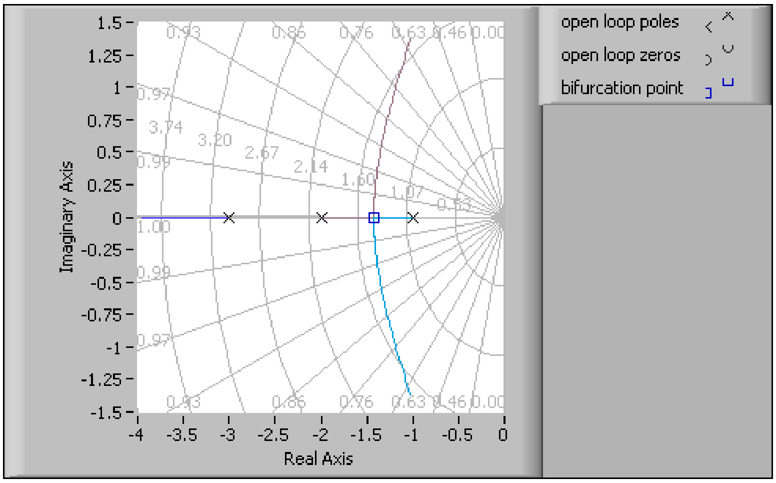


Figure 9-1. Root Locus

This graph gives you an idea of closed-loop pole locations due to the variations in the gain. The control design process involves selecting a specific gain value to achieve a certain performance requirement. You derive a desired pole position from the performance requirement and then base the gain value on that pole position.

Synthesizing Controllers Using Root-Locus Design

You can use root-locus design to synthesize a variety of different controller configurations, such as lead compensators, lag compensators, notch compensators, and PID controllers. The difference in these controller configurations is the transfer function equations you use to synthesize the controller. Different transfer function models result in different ways that you can use the controller.

Lead compensators lower the rise time and decrease the transient overshoot. Lag compensators improve the steady-state accuracy of the system. You use notch compensators to achieve stability for systems with lightly damped flexible modes. You can achieve this stability by adding a zero near the resonance point of the flexible mode.

Refer to *Feedback Control of Dynamic Systems*[1] and *Modern Control Engineering*[2] for more information about the structure and design of these configurations. Refer to the *Proportional-Integral-Derivative Controller* section for information about the PID controller.

For example, consider a controller transfer function model defined by the following equation.

$$D(s) = K\frac{s+z}{s+p}$$

This transfer function model results in a lead compensator if $z < p$ and a lag compensator if $z > p$. You typically place this lead compensator in series with the plant $H(s)$ in the feed-forward path. The characteristic equation of such a system is defined by the following equation.

$$1 + D(s) + H(s) = 0$$

You can rewrite the equation using the following equation.

$$1 + KL(s) = 0$$

To perform root-locus design, you want to find the best value of K based on the closed-loop pole locations of this system. Refer to the CDEx Interactive Root Locus VI in the `labview\examples\Control Design\Getting Started.llb` for an example VI that demonstrates this process of determining K by moving the closed-loop poles along the root-locus plot.

[1] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[2] Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2001.

In addition to root-locus analysis, you can use other frequency-domain tools, such as Bode, Nyquist, and Nichols plots, to analyze a system. These tools enable you to determine the specific locations and shape of key points on these plots and to iteratively modify the controller parameters to achieve these specifications. The number and nature of the controller parameters depends on the topology of the controller. Refer to *Feedback Control of Dynamic Systems*[1] and *Modern Control Engineering*[2] for information about design processes that use these frequency domain plots.

Proportional-Integral-Derivative Controller

The proportional-integral-derivative (PID) controller, also known as the three-term controller, is the most popular and widely used architecture in most industrial settings. The PID architecture combines proportional (*P*), integral (*I*), and derivative (*D*) compensation. This controller compares the measured value against the setpoint and initiates the appropriate corrective action. For a setpoint *R*(*t*) and process variable *B*(*t*), the following equation defines the error.

$$e(t) = R(t) - B(t)$$

The control action is a function of the error. The following equation defines control action for a general PID controller.

$$\iota(t) = K_c\left[e(t) + \frac{1}{\tau_I}\int_0^t e(t^*)dt^* + \tau_d\frac{de(t)}{dt}\right]$$

The following equation defines the corresponding transfer function.

$$\frac{U(s)}{E(s)} = K_c\left(1 + \frac{1}{\tau_I s} + \tau_d s\right)$$

[1] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[2] Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2001.

This transfer function is difficult to realize physically because it is improper. To make the transfer function realizable, you can modify the transfer function to contain an additional pole placed at $-1/\alpha\tau_d$, where α is a small number, typically between 0.05 and 0.2, that has negligible effect on the system dynamics.

$$\frac{U(s)}{E(s)} = K_c\left(\frac{\tau_I s + 1}{\tau_I s}\right)\left(\frac{\tau_D s + 1}{\alpha\tau_D s + 1}\right)$$

The preceding equation describes the PID Series form of the PID controller. The LabVIEW Control Design Toolkit supports three forms of the PID controller—the PID Academic, PID Parallel, and PID Series. Use the CD Construct Special Model VI to create these forms of the PID controller. You then can combine these models with various plant models and analysis functions to perform design operations. Table 9-1 describes the three forms you can use to construct a PID controller.

Table 9-1. PID Controllers in the Control Design Toolkit

PID Controller Form	Equation
PID Academic	$\frac{U(s)}{E(s)} = K_c\left(1 + \frac{1}{T_i s} + \frac{T_d s}{\alpha T_d s + 1}\right)$
PID Parallel	$\frac{U(s)}{E(s)} = K_c + \frac{K_i}{s} + \frac{\tau K_d s}{\alpha K_d s + 1}$
PID Series	$\frac{U(s)}{E(s)} = K_c\left(1 + \frac{1}{T_i s}\right)\left(\frac{T_d s + 1}{(HF_1 s + 1)(HF_2 s + 1)}\right)$

Note In some applications, you specify the gain in the PID Academic transfer function in terms of a proportional band (PB).

$$PB = \frac{1}{K_c} \times 100\%$$

A proportional band, defined by the following equation, is the percentage of the input range of the controller that causes a change equal to the maximum range of the output.

Each topology has advantages and disadvantages. For example, the PID Parallel topology allows you to adjust each term independently, unlike the other topologies where some terms interact with the others. The PID Academic and PID Serial forms allow you to treat the time constant terms differently from the gain term. The PID form you use depends on the design decisions you make. Factors, such as the design specifications or data results of a tuning process, can influence the PID form you choose to use.

You can use the root locus and Bode design methods to determine appropriate gain values for the PID controller. Refer to *PID Controllers: Theory, Design, and Tuning*[1] for more information about the use of these techniques for PID design. Refer to the *LabVIEW PID Control Toolset User Manual* for more information about experimentally determining controller gain parameters.

[1] Astrom, K. and T. Hagglund. *PID Controllers: Theory, Design, and Tuning*, 2nd ed. ISA, 1995.

10 State-Space Analysis

This chapter describes the state-space analysis properties—controllability, observability, Grammians, and balancing—that enable you to determine the stability of a system so you then can determine how to design a controller.

You can build a state-space representation of the system based on the dynamics of the system. The following first-order differential equations, as defined in Chapter 2, *Creating Dynamic System Models*, represent a state-space model of a continuous system.

$$\begin{aligned} \dot{x} &= \boldsymbol{A}x + \boldsymbol{B}u \\ y &= \boldsymbol{C}x + \boldsymbol{D}u \end{aligned}$$

Similarly, for discrete systems, the following first-order difference equations represent a state-space model of a discrete system.

$$\begin{aligned} x(k+1) &= \boldsymbol{A}x(k) + \boldsymbol{B}(k) \\ y(k) &= \boldsymbol{C}x(k) + \boldsymbol{D}u(k) \end{aligned}$$

A system can have many inputs and many outputs. By using state variables, you can represent all inputs and outputs of a system using these equations. For example, consider a motor that has power as its input and speed and acceleration as outputs. In a state-space model, the state variables are speed and acceleration. State variables often have physical meaning and represent some internal state of the system under analysis.

For a given linear time-invariant MIMO system, the state-space representation is not unique. You must determine which representation is best for the analysis and design of a state-space controller.

System Stability

In SISO systems, the unstable poles lead to unstable behavior in the system. Therefore, in a state-space representation, the time evolution of the states determines the stability of the system. In fact, if you have initial conditions and you eliminate all inputs to the system, the matrix $\boldsymbol{A}$ alone governs the behavior. Then control theory enables you to find the counterparts of poles, which you can use in transfer function and pole-zero analysis. The counterparts of poles are the eigenvalues, or modes, of the matrix $\boldsymbol{A}$. The location of these eigenvalues determines the stability of the system.

Controllability

A system is controllable if all the states that describe the system respond to the inputs of the system. You can influence the outputs of the system by adjusting the inputs. If a system is controllable, there is an input that forces the system states to go from any initial condition at $t = 0$ to zero at a time $t > 0$. So if a system becomes unstable, you can adjust the input to have a specific effect on the behavior of the states.

If a system contains states that the inputs can never affect, the system is not controllable.

For continuous systems, you can verify the controllability of a system by verifying that the controllability matrix $\boldsymbol{Q}$, shown in the following equation, has full row rank or is nonsingular for a SISO system.

$$\boldsymbol{Q} = [\boldsymbol{B} \;\; \boldsymbol{AB} \;\ldots \boldsymbol{A}^{n-1}\boldsymbol{B}]$$

The system matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ determine the controllability properties of a state-space model and are used in calculating the controllability matrix. A system is controllable if its controllability matrix $\boldsymbol{Q}$ is full row rank. For example, if $\boldsymbol{B}$ is an n-dimensional column vector that is colinear to an eigenvector of null eigenvalues of $\boldsymbol{A}$, you obtain the following matrix.

$$\boldsymbol{Q} = [\boldsymbol{B} \;\; 0 \;\; 0 \;..0]$$

This matrix is row rank deficient for $n > 1$. The null eigenvalue represents an uncontrollable mode of the system. If the colinearity of $\boldsymbol{B}$ with eigenvectors of null eigenvalues of $\boldsymbol{A}$ is coincidental, the systems are usually controllable.

From the definition of a controllable system you can conclude that to place the system states at zero at any desired time indicates that you can place all system poles anywhere necessary to make the closed-loop response reach zero at time t as quickly as possible.

When you can adjust all system poles locations to a desired location, you can calculate a full state-feedback controller gain K to arbitrarily place the eigenvalues of the closed-loop system, $\boldsymbol{A}' = \boldsymbol{A} - \boldsymbol{B}K$. Conversely, the eigenvalues associated with modes that are not controllable cannot be adjusted, regardless of the value you choose for K.

Use the CD Controllability Matrix VI to calculate the controllability matrix of the model and determines whether the system is controllable. Use the CD Controllability Staircase VI to transform a state-space model into a model that you can use to identify controllable states in the system. The CD Controllability Staircase VI calculates the controllability matrix of the transformed model and inspects the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices of the transformed model to determine the controllable states.

Observability

Observability is a property that indicates whether you can determine each state of the system only by looking at the output of the system. A system is observable if you can determine the state at time t_0 by observing the output from time t_0 to t_1.

System observability is an important property to have when designing full state feedback controllers. If a system is not observable, you never can determine certain state values, and you can use only partial state feedback controllers.

Observability depends on the matrix $\boldsymbol{C}$ and the matrix $\boldsymbol{A}$ of the system. You can check observability by verifying if the matrix $\boldsymbol{O}$, defined in the following equation, is full column rank or is nonsingular for a SISO system.

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \vdots \\ \boldsymbol{CA}^{n-1} \end{bmatrix}$$

If the system is observable and $\boldsymbol{C}$ is not the identity matrix, you can use a state estimator, or observer, to estimate the states from the output. Refer to the *State Estimator* section of Chapter 12, *Modern Control Design Synthesis*, for information about state estimators and observers.

Use the CD Observability Matrix VI to calculate the observability matrix of the model and determine whether the system is observable. Use the CD Observability Staircase VI to transform a state-space model into a model that you can use to identify observable states in the system. The CD Observability Staircase VI calculates the observability matrix of the transformed model and inspects the ***A*** and ***C*** matrices of the transformed model to determine the observable states.

Controllability and Observability Grammians

An alternative and numerically more stable approach to assessing controllability and observability is to compute the Grammians of the state-space matrices. The controllability Grammian is an $n \times n$ matrix that results from analyzing the behavior of the system under small perturbations in the state trajectory. If the controllability Grammian is positive-definite, meaning all eigenvalues are real and greater than zero, the chosen state-space representation is controllable. Moreover, states corresponding to small eigenvalues of the controllability Grammian matrix affect the output and consequently affect the internal states the least.

Similarly, the observability Grammian is an $n \times n$ matrix that results from analyzing how many of the initial conditions you can estimate by looking at the outputs of the system. If the observability Grammian is positive-definite, the chosen state-space representation is observable. States corresponding to small eigenvalues of the observability Grammian contribute less when estimating the initial conditions of the system.

Use the CD Grammians VI to determine the controllability and observability Grammians of a state-space model for a stable system.

Balanced Systems

A balanced state-space model is one that has identical controllability and observability diagonal Grammians. A diagonal matrix is positive-definite if all diagonal elements are greater than zero. A balanced model simplifies the analysis and use of Grammians in model order reduction.

In model order reduction, balancing highlights the relative importance of the state to the input/output performance of the system. Balancing consists of finding a similarity transformation from the original model to generate a state-space representation that has some desired property. Use the CD Balance State-Space Model (Diagonal) VI and the CD Balance State-Space Model (Grammians) VI to balance a state-space system.

If you use the CD Balance State-Space Model (Grammians) VI, the balanced state-space model has equal and diagonal controllability and observability Grammians.

If you use the CD Balance State-Space Model (Diagonal) VI, the balanced system has an even eigenvalue spread for $\boldsymbol{A}$ or the composite matrix, which contains the natural composition of $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.

11

Model Reduction

This chapter describes the minimal realization and model order reduction techniques you can use to simplify a model.

A dynamic system model describes the internal behavior of a system and its response to stimuli. In most cases, different models can represent the same internal behavior. For example, you can have two state-space models with different numbers of states that represent the same system at varying degrees of fidelity. Often you can simplify these models to obtain a less complicated representation of the system.

How you simplify, or reduce, a model depends on the representation you use to describe the system. The order of the model refers to the number of poles or the number of states that a model has. Reducing the number of states in a state-space model reduces the order of the model. Similarly, reducing the number of poles in a transfer function or zero-pole-gain model reduces the order of those models. To reduce the number of poles, you can cancel matching poles and zeros. Use the Model Reduction VIs to simplify or reduce the order of a model.

Minimal Realization

Consider the following transfer function model $H(s)$ of a system.

$$H(s) = \frac{s^2 + 6s + 8}{s^3 - 8s^2 - 21s + 108} = \frac{(s+2)(s+4)}{(s+4)(s-3)(s-9)} = \left.\frac{(s+2)}{(s-3)(s-9)}\right\} \textit{Minimal Realization}$$

This model has a pole and zero in the same location, so you can choose to cancel this zero-pole pair at –4 by using the CD Minimal Realization VI. When you remove all zero-pole pairs in a transfer function or zero-pole-gain model, the resulting model is the minimal representation of a system.

Minimal realizations are minimal because the only modes represented in the model are those modes that you can infer by observing the inputs and outputs of the system. The modes that you eliminate to obtain a minimal transfer function or zero-pole-gain model still exist in the system, but you cannot infer their existence by simply observing the input and outputs of the model.

The following equation shows the corresponding zero-pole-gain model $H(s)$.

$$H(s) = \frac{s^2 - 2s - 8}{s^3 - 16s^2 + 75s - 108} = \frac{(s+2)(s-4)}{(s-4)(s-3)(s-9)} = \left.\frac{(s+2)}{(s-3)(s-9)}\right\} \textit{Minimal Realization}$$

In this zero-pole-gain model, there is an unstable zero-pole pair at 4. Canceling this zero-pole pair gives you a minimal realization of the model. While mathematically correct, this minimal realization is undesirable because the zero-pole pair is unstable. If a pole is unstable, canceling the zero-pole pair results in a model that appears stable even though the physical system still contains unstable internal dynamics.

Transfer function and zero-pole-gain models describe the dynamics of a system using only inputs and outputs. In addition to the inputs and outputs, you can use states to represent the internal dynamics of a system. When a transfer function or zero-pole-gain model contains unstable internal dynamics and corresponding zero-pole cancellations, consider converting the model to a state-space representation to better capture the true dynamics of the system. You can design a controller using a state-space model, even if it is unstable, as long as the model is controllable and observable.

A minimal realization for a state-space model is a state-space representation in which you remove all states that are not observable or controllable. You can use the CD Minimal State Realization VI to determine the minimal realization for a state-space model. Refer to Chapter 10, *State-Space Analysis*, for more information about controllability and observability.

Model Order Reduction

In certain situations, you might want to work with a lower order model of the system. Use the CD Model Order Reduction VI to simplify high-order models.

You can reduce the order of the model by decreasing the order of the stable modes. Reducing stable modes of the model does not affect the unstable modes of the model.

Model order reduction applies only to a state-space model of a system. The goal of model order reduction is to remove the states that correspond to stable modes of the plant and have the smallest impact on the behavior of the system.

You have several options for reducing the order of a model. You can match the DC gain between the reduced order model and the original model. You also can delete the states directly. You might be able to observe a better frequency response approximation when you directly delete a state.

With model order reduction, you must know the effect of each state on the behavior of the model. To determine the effect of each state, compute the observability and controllability Grammians of the state-space model. Then compute the eigenvalues of the Grammians. Each eigenvalue directly corresponds to a state and to a mode of the system. Eliminating stable states with small eigenvalues in magnitude ensures more accurate approximations.

Balancing the original state-space model for the system can make the model order reduction process easier. When you balance the state-space model, the Grammian matrices are diagonal and you avoid computing the eigenvalues.

Given a state-space model, complete the following steps to reduce the model order.

1. Balance the state-space model.
2. Compute the Grammians.
3. Remove stable states corresponding to small eigenvalues, in proportion to the other eigenvalues, of the Grammian matrix.
4. Repeat steps 1 through 3 until the model is of the desired order.

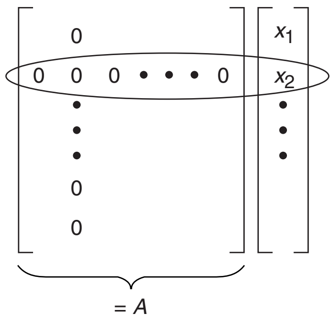
Refer to the *Controllability and Observability Grammians* section and the *Balanced Systems* section of Chapter 10, *State-Space Analysis*, for more information about computing controllability and observability Grammians and balancing a model.

Selecting and Removing an Input, Output, or State

You can use the CD Select IO from Model VI and CD Remove IO from Model VI to reduce a model by directly removing inputs, outputs, or states.

In a transfer function or zero-pole-gain model, manipulating the system representation corresponds to ignoring certain inputs and outputs, such as those connected by a unit gain. In a state-space model, manipulating the system representation corresponds to removing the undesired states from the description.

Manipulating a model is useful for building new models from old ones and for quickly removing zero-states from a large state-space model representation. Zero-states are states for which the matrix ***A*** has zeros in the corresponding row and corresponding column, as shown in the following matrix ***A***.



If the matrix has no zero rows or columns, you must use another method to reduce the model order.

Note When you work with transfer function and zero-pole-gain models, you generally do not select and remove specific inputs and outputs to reduce the model order. You mainly use this method with state-space models.

12 Modern Control Design Synthesis

Classical control design involves creating controllers based only on the relationship between the inputs and outputs of the system. Unlike classical control design, modern control design involves creating controllers based on the relationship between inputs and states of the system. Modern control design techniques use state-space models to synthesize and analyze controllers. By using a state-space model of the system you want to control, you have access to the state dynamics of a system. Knowledge of the state dynamics enables you to design a more optimal controller for the system by improving the performance of the controller.

Similar to classical control design, the process of designing a controller begins with specifying the control objective. However, in the case of modern control design, the objective is to minimize a cost function or to place the poles and zeros of a system in specific locations. You then select the architecture of the controller, such as whether the feedback is based only on outputs or on all the states of the system. With this information, you synthesize a controller by selecting an appropriate set of parameters to satisfy the stated objectives.

This chapter describes modern control design techniques for determining estimator and controller gain values that you can use to design a controller. These techniques include pole placement design, linear quadratic regulator, Kalman gain, or linear quadratic gaussian. This chapter also describes how to design state-space controllers and state-space estimators. State-space controllers and state-space estimators have different configurations that either include or exclude the system, the estimator, and the system noise. This chapter provides information about how the LabVIEW Control Design Toolkit implements these controllers using the different configurations.

Refer to Chapter 9, *Classical Control Design*, for information about classical control design.

State Feedback Synthesis

The advantage of using state-space models is that they provide a representation that fully captures aspects of the system that an input-output based model might not capture. State-space techniques are based on the following continuous and discrete state-space model equations, as defined in the *State-Space Model* section of Chapter 2, *Creating Dynamic System Models*.

Continuous State-Space Model

$$\begin{aligned}\dot{x} &= \boldsymbol{A}x + \boldsymbol{B}u \\ y &= \boldsymbol{C}x + \boldsymbol{D}u\end{aligned}$$

Discrete State-Space Model

$$\begin{aligned}x(k+1) &= \boldsymbol{A}x(k) + \boldsymbol{B}u(k) \\ y(k) &= \boldsymbol{C}x(k) + \boldsymbol{D}u(k)\end{aligned}$$

Before you can synthesize and implement a controller or estimator, you need to calculate the controller and estimator gain values. State feedback synthesis involves finding these gain values. The following sections describe various design techniques—pole placement, linear quadratic regulator, Kalman gain, and linear quadratic Gaussian—that enable you to perform a state feedback synthesis.

Pole Placement Design

Consider a state-space system in which you can measure all state variables and use them for feedback. If this system is completely controllable, you can place the poles of the closed-loop system at any desired location using state feedback with an appropriate gain matrix. Pole placement is a design procedure that computes the desired gain matrix from specifications of the desired closed-loop pole locations.

Consider the following SISO state-space system with $u = -Kx$ as the control action.

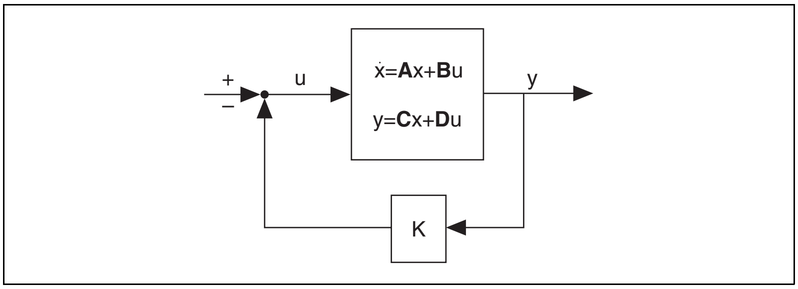


Figure 12-1. Full-State Feedback Used to Relocate the Eigenvalues of a Controllable System Based on the Value of the Gain *K*

Given a specification of the desired closed-loop pole locations, $\lambda_1, \lambda_2, \ldots \lambda_n$, you can compute the gain K that achieves this goal. The system under control must be closed-loop controllable, so you must convert the system equations to the controllable canonical form.

The closed-loop continuous system has the following form.

$$\dot{x} = \tilde{\boldsymbol{A}}x$$
$$\tilde{\boldsymbol{A}} = \boldsymbol{A} - \boldsymbol{B}K$$

Using the matrix $\tilde{\boldsymbol{A}}$ to satisfy the characteristic equation of this closed-loop system, you get the following equation.

$$\phi(\tilde{\boldsymbol{A}}) = \tilde{\boldsymbol{A}}^n + \alpha_1\tilde{\boldsymbol{A}}^{n-1} + \ldots + \alpha_{n-1}\tilde{\boldsymbol{A}} + \alpha_n\boldsymbol{I} = 0$$

Using this equation, you can calculate Ackermann's formula, which provides an efficient way to compute the gains for the SISO case.

$$K = \begin{bmatrix}0 & 0 & \ldots 1\end{bmatrix}\begin{bmatrix}\boldsymbol{B} & \boldsymbol{AB} & \ldots\boldsymbol{A}^{n-1}\boldsymbol{B}\end{bmatrix}^{-1}\phi(\tilde{\boldsymbol{A}})$$

Ackermann's formula is easy to compute. However, Ackermann's formula only applies to SISO systems. The computation of the gain for MIMO systems is more complex and based on a Sylvester matrix equation. Refer to Chapter 14, *Advanced Equation Solvers*, for information about the Sylvester matrix equation.

Use the CD Ackermann VI to compute the gain K for a SISO system. The CD Pole Placement VI computes the gain matrix K for a SISO or MIMO system.

Linear feedback control design methods based on Bode plots and Root Locus plots are more common in designing of SISO systems. Historically, these methods are related closely to input-output system descriptions involving transfer function and zero-pole-gain models. Pole placement is a state feedback design technique that takes into account the desired pole locations that you can derive from performance specifications such as damping ratio or settling time.

Another set of design techniques is based on the use of optimal control theory. These design techniques synthesize controllers by trying to minimize a cost function, also known as a performance index. The following section describes one of these techniques.

Linear Quadratic Regulator

One of the more popular techniques for optimal state feedback control design is the Linear Quadratic Regulator (LQR). For a continuous state-space model, this controller minimizes a performance index of the following form.

$$(x_0,\ u(t),\ t_f) = \frac{1}{2}x^T(t_f)\boldsymbol{N}x(t_f) + \frac{1}{2}\int_0^{t_f} [x^T(t)\boldsymbol{Q}x(t) + u^T(t)\boldsymbol{R}u(t)]d$$

Q is a symmetric positive semi-definite matrix that penalizes the state vector *x* in the cost function. ***R*** is a positive definite matrix (usually symmetric) that penalizes the input vector *u* in the cost function. ***N*** is a symmetric positive semi-definite matrix that penalizes the cross product between input and state vectors.

The Control Design Toolkit computes the solution to the steady state version of this problem as $t_f \to \infty$. All unstable modes of the system must be controllable and observable.

The design process for LQR requires that you specify matrices ***Q*** and ***R***, which specify the weights on the state and control action, respectively. The algorithm then computes the full state feedback gain matrix required to control the system. Typically, the selection of these gain matrices is an iterative process.

The LQR design has many benefits. The LQR controller is guaranteed to provide the best linear controller that minimizes the performance index specified. Also, the closed-loop system has good robustness.

Use the CD Linear Quadratic Regulator VI to compute the gain matrix K for a given state-space model. You can use the matrix K, computed by this VI, with the CD State-Space Controller VI to implement the optimal controller.

Kalman Gain

One disadvantage of state feedback is that not all states are measurable. You can overcome this problem by using an estimator that uses the measurements of the output variables to infer the values of unmeasured states.

The following equation defines a continuous state-space system where w is the process noise, v is the measurement noise, and u is the known input.

$$\begin{aligned} \dot{x} &= \boldsymbol{A}x + \boldsymbol{B}u + \boldsymbol{G}w \\ y &= \boldsymbol{C}x + \boldsymbol{D}u + \boldsymbol{H}w + v \end{aligned}$$

The covariance of these signals are defined by the following matrices.

$$\begin{aligned} E(ww^T) &= \boldsymbol{Q} \\ E(vv^T) &= \boldsymbol{R} \\ E(wv^T) &= \boldsymbol{N} \end{aligned}$$

If you denote the estimated values of the state by $\hat{x}$, then the estimator updates the state equation as follows.

$$\dot{\hat{x}} = \boldsymbol{A}\hat{x} + \boldsymbol{B}u + L(y - \hat{y})$$

The steady state Kalman Filter is optimal because it minimizes the covariance of the estimation error $(x - \hat{x})$.

Use the CD Kalman Gain VI to compute the gain matrix $\boldsymbol{L}$ for a given state-space model. You can use the matrix $\boldsymbol{L}$, computed by this function, with the CD State Estimator VI to perform optimal estimation of unmeasured states. Refer to the *State Estimator* section of this chapter for more information about the CD State Estimator VI.

Linear Quadratic Gaussian

The use of an estimator with a design such as LQR might not result in the most optimal design of the controller. If the estimator starts with the same initial condition as the unmeasured states, $\hat{x}(0) \equiv x(0)$, and if a number of controllability and observability conditions are satisfied, the closed-loop system with the observer based controller will have the same responses as the LQR design. This form of state feedback controller is called the Linear Quadratic Gaussian (LQG) controller. Certainty equivalence is the property that allows this combined usage of optimal estimator and controller. Certainty equivalence is important because you can synthesize a controller gain K and estimator gain $\boldsymbol{L}$ independently. In practice, you can build a controller assuming all states are measurable and then estimate unmeasured states using an optimal estimator. The resulting design is optimal for the specified problem. However, the robustness properties of LQG are not the same as that of LQR. You have no guarantee that robustness properties can be established for an estimated state feedback controller.

Use the CD Linear Quadratic Regulator VI and CD Kalman Gain VI with the CD State-Space Controller VI to create an LQG controller.

Estimator and Controller Configurations

Calculations of the controller and estimator gains are based on closed-loop performance specifications and model noise approximations. Therefore, you need to analyze and simulate these designs before applying them to the actual process.

However, a problem with using state-space models is that not all states are physically measurable. You also can overcome this problem by using observers that estimate the unmeasured states based on observations of the outputs. A majority of state-space design procedures are equally applicable to both observer and controller design problems. If you cannot measure all states, the Control Design Toolkit still can calculate the gain values.

The following section describes how the Control Design Toolkit accounts for a situation when some of the inputs and outputs of a system are unavailable. The following sections also describe how to incorporate these gains into a dynamic model to implement a state estimator and state controller. You can use these estimator and controller models for performance analysis, simulation, and real-time deployment.

Measured Outputs, Known Inputs, and Adjustable Inputs

The estimator gain $\boldsymbol{L}$ considers all inputs u and outputs y, which are known and measured. Also, some inputs and outputs might be unavailable. So you decompose the system into adjustable inputs, measured outputs, unknown inputs, and unmeasured outputs. The decomposition is based on diagonal matrices, such as Λ_u and Λ_y.

Diagonal matrices incorporate the effect of known, unknown, measured, and unmeasured inputs and outputs into the equation. A diagonal element in these matrices equals unity for the known and measured inputs and outputs, and zero for the unknown and unmeasured inputs and outputs or states. The following equation describes how the diagonal elements for the inputs and outputs are incorporated in the controller model.

$$\begin{aligned}\dot{\hat{x}} &= \boldsymbol{A}\hat{x} + \boldsymbol{B}^* u + L^*(y - \hat{y}) \\ \hat{y} &= \boldsymbol{C}\hat{x} + \boldsymbol{D}^* u\end{aligned}$$

In this equation, $\boldsymbol{B}^* = \boldsymbol{B}\Lambda_u$, $\boldsymbol{D}^* = \boldsymbol{D}\Lambda_u$, and $\boldsymbol{L}^* = \boldsymbol{L}\Lambda_y$. These substitutions apply to both estimators and controllers. Controllers have an additional substitution when inputs are not adjustable. For a controller, the controller gain $\boldsymbol{K}^*$ is given by $\boldsymbol{K}^* = \boldsymbol{K}\Lambda_z$, where Λ_z is a diagonal matrix with the same characteristics as Λ_u and Λ_y. Therefore, a diagonal element in Λ_z equals unity for the adjustable input, and zero for the nonadjustable or system disturbances. By default, matrices Λ_u, Λ_y, and Λ_z are identity matrices whose size equals the number of inputs and outputs, respectively.

The following sections describe how to implement a state estimator and a state controller using the Control Design Toolkit. The estimators and controllers in these sections assume that all inputs are known and all outputs are measurable.

State Estimator

State controllers use state information to calculate the control action. In a physical system, not all states are measurable. You therefore must estimate the states. To implement a state estimator, you need a model of the system you want to control and an estimator gain. You can calculate the estimator gain using the CD Pole Placement VI, CD Ackermann VI, or CD Kalman Gain VI, described in the *Pole Placement Design* section of this chapter.

The CD State Estimator VI integrates the estimator gain $\boldsymbol{L}$ into a dynamic system for analyzing and simulating the estimator performance. Consider the following equations that represent a continuous state-space system.

$$\begin{aligned} \dot{x} &= \boldsymbol{A}x + \boldsymbol{B}u \\ y &= \boldsymbol{C}x + \boldsymbol{D}u + r_y \end{aligned}$$

Assume that the estimator gain $\boldsymbol{L}$ is based on this system, some estimator performance specifications, and the output noise r_y covariance. You can calculate the estimated states $\hat{x}$ using the following equations for a dynamic model.

$$\begin{aligned} \dot{\hat{x}} &= \boldsymbol{A}\hat{x} + \boldsymbol{B}u + L(y - \hat{y}) \\ \hat{y} &= \boldsymbol{C}\hat{x} + \boldsymbol{D}u \end{aligned}$$

Note To simplify the equations in the rest of this chapter, assume that all inputs are known and all outputs are measurable. This assumption means $\boldsymbol{B}^* = \boldsymbol{B}$, $\boldsymbol{L}^* = \boldsymbol{L}$, and $\boldsymbol{D}^* = \boldsymbol{D}$.

The state-space system and dynamic model equations share the same system matrices and input u. The states x and $\hat{x}$ are different because the initial conditions of the system might differ from the model and because of the noise input r_y. However, in the absence of a noise input, the model states track the system states, making the difference $x - \hat{x}$ converge asymptotically to zero. The following equation shows how the estimator gain $\boldsymbol{L}$ is expected to enhance the convergence of the error $\dot{e}$ to zero.

$$\dot{e}_x \equiv \dot{\hat{x}} - \dot{x} = \boldsymbol{A}(\hat{x} - x) + L(y - \hat{y}) = (\boldsymbol{A} - L\boldsymbol{C})e_x + Lr_y$$

In the absence of the noise input, the error convergence is defined by the following equation.

$$\dot{e}_x = (\boldsymbol{A} - L\boldsymbol{C})e_x$$

The estimator gain $\boldsymbol{L}$ is designed to place the poles of the matrix $\boldsymbol{A} - \boldsymbol{LC}$ in the desired complex-plane location.

Configuration Types

There are different configurations to synthesize the estimator. These configurations are system included, system included with noise, and standalone. The system included configuration appends the actual states of the system to the estimated states. The system included configuration with noise incorporates noise r_y into the system included configuration. The standalone configuration is used for implementing the estimator on a real-time target or when you have a system-model mismatch.

A general system configuration is one where the original model states (x) are appended to the estimation model states ($\hat{x}$) to represent the estimator, as shown in the following equations.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & 0 \\ 0 & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} & L \\ \boldsymbol{B} & 0 \end{bmatrix} \begin{bmatrix} u \\ y - \hat{y} \end{bmatrix}$$

$$\begin{bmatrix} \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} & 0 \\ 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{D} & 0 \\ \boldsymbol{D} & 0 \end{bmatrix} \begin{bmatrix} u \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} 0 \\ r_y \end{bmatrix}$$

Given this general system configuration, the following sections show how to derive the system included, system included with noise, and standalone configurations.

System Included Configuration

In the system included configuration, the output estimator error is defined by the following equation.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x})$$

By substituting the output estimator error in the general system equations and removing the sensor noise r_y from the system, you obtain the following equations that describe the system included configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - L\boldsymbol{C} & L\boldsymbol{C} \\ 0 & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B} \end{bmatrix} u$$

$$\begin{bmatrix} \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} & 0 \\ 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{D} \\ \boldsymbol{D} \end{bmatrix} u$$

Figure 12-2 represents the system described by these equations.

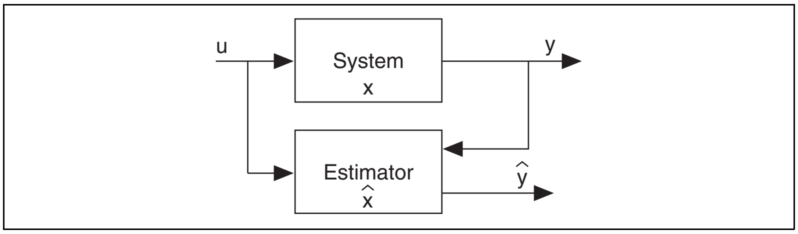


Figure 12-2. Estimator Included

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ x \end{bmatrix}$, u, and $\begin{bmatrix} \hat{y} \\ y \end{bmatrix}$, respectively.

The system included configuration is useful because you can analyze and simulate the estimated states and the original states at the same time.

System Included with Noise Configuration

In the system included with noise configuration, there is a noise input r_y in the system. The output estimator error is defined by the following equation.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x}) + r_y$$

By substituting the output estimator error in the general system equations, you obtain the following equations that describe the system included with noise configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - L\boldsymbol{C} & L\boldsymbol{C} \\ 0 & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} & L \\ \boldsymbol{B} & 0 \end{bmatrix} \begin{bmatrix} u \\ r_y \end{bmatrix}$$

$$\begin{bmatrix} \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} & 0 \\ 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{D} & 0 \\ \boldsymbol{D} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} u \\ r_y \end{bmatrix}$$

Figure 12-3 represents the system described by these equations.

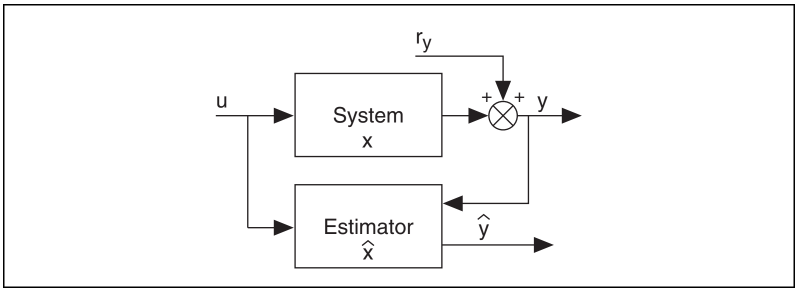


Figure 12-3. State Estimator with Noise

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ x \end{bmatrix}$, $\begin{bmatrix} u \\ r_y \end{bmatrix}$, and $\begin{bmatrix} \hat{y} \\ y \end{bmatrix}$, respectively.

The system included with noise configuration is useful because you can analyze and simulate the estimated states and the original states at the same time. Furthermore, you can analyze and simulate the effects of the sensor noise in the system.

Standalone Configuration

In the standalone configuration, the system is detached from the estimator model and the system outputs *y* become inputs. Unlike the system included and system included with noise configurations, there is no output estimation error.

The following equations describe the standalone configuration.

$$\dot{\hat{x}} = (\boldsymbol{A} - L\boldsymbol{C})\hat{x} + \begin{bmatrix} \boldsymbol{B} - L\boldsymbol{D} & L \end{bmatrix} \begin{bmatrix} u \\ y \end{bmatrix}$$

$$\hat{y} = \boldsymbol{C}\hat{x} + \begin{bmatrix} \boldsymbol{D} & 0 \end{bmatrix} \begin{bmatrix} u \\ y \end{bmatrix}$$

The original system is not appended, meaning the system output is not generated internally but considered as another input to the estimator, as shown in Figure 12-4.

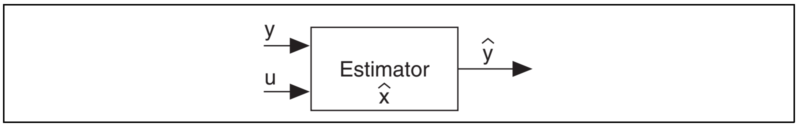


Figure 12-4. Standalone Estimator

The states, inputs, and outputs of the estimator are $\hat{x}$, $\begin{bmatrix} u \\ y \end{bmatrix}$, and $\hat{y}$, respectively.

The main purpose of the standalone configuration is to implement the estimator on a real-time target. A secondary purpose of the standalone configuration is to perform offline simulation and analysis of the estimator. Offline simulation and analysis is useful for testing the estimator with mismatched models and systems. Mismatched models and systems have a calculated estimator gain that applies to a model with uncertainties.

State Controller

State controllers use state information to calculate the control action. The controller gain K is based on closed-loop performance specifications. To implement a state controller, you need a controller gain. You can calculate the controller gain using the CD Pole Placement VI, CD Ackermann VI, CD Kalman Gain VI, or CD Linear Quadratic Regulator VI, as described in *State Feedback Synthesis* section of this chapter. Some state controllers require a state estimator, which requires a model of the system you want to control.

The Control Design Toolkit provides three types of state controllers—a compensator, a regulator, and a regulator with integral action. The difference in these controllers is in how you calculate the control action u.

For a state compensator, the control action is defined by $u = K(r_x - x)$, where r_x is a state reference. When the states are estimated, the state compensator control action is defined by $u = K(r_x - \hat{x})$.

For a state regulator, the control action is defined by $u = r_u - Kx$, where r_u is an input reference. When the states are estimated, the state compensator control action is defined by $u = r_u - K\hat{x}$.

For a state regulator with integral action, the control action is defined by the following equation.

$$\iota = -\begin{bmatrix} K & K_I \end{bmatrix} \begin{bmatrix} x \\ \int (y_{ref} - y) \end{bmatrix}$$

y_{ref} is the output reference, or the setpoint.

Configuration Types of a State Controller

There are different configurations to synthesize a state controller. These configurations are system included, system included with noise, standalone with estimator, and standalone without estimator. Both the system included and system included with noise configurations automatically include an estimator in the configuration.

The system included configuration appends the actual states of the system to the estimated states. The system included configuration is useful because you can analyze and simulate the estimated states and the original states at the same time.

The system included with noise configuration incorporates sensor noise r_y into the system included configuration. The system included with noise configuration is useful because you can analyze and simulate the estimated states and the original states at the same time. Furthermore, you can analyze and simulate the effects of the sensor noise in the system.

The standalone without estimator configuration bases the control action u on the actual states x because it does not require an estimator. The standalone without estimator configuration helps you perform a closed-loop analysis of the system.

The standalone with estimator configuration is used when the actual system matrices are unknown or do not match the model matrices. The standalone with estimator configuration implements the estimator on a real-time target. The standalone configuration also is useful for performing offline simulations and analyses of the controller. You can use offline simulations and analyses to test the controller with mismatched models and systems. Mismatched models and systems have a calculated estimator and controller gain that applies to the mismatched model, or the model with uncertainties.

State Compensator

A general system configuration is one where the original model states (x) are appended to the estimation model states ($\hat{x}$) to represent the compensator with an estimator, as shown in the following equations.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K & 0 \\ -\boldsymbol{B}K & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}K & L \\ \boldsymbol{B}K & 0 \end{bmatrix} \begin{bmatrix} r_x \\ y - \hat{y} \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K & 0 \\ \boldsymbol{C} - \boldsymbol{D}K & 0 \\ -\boldsymbol{D}K^* & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} K & 0 \\ \boldsymbol{D}K & 0 \\ \boldsymbol{D}K & 0 \end{bmatrix} \begin{bmatrix} r_x \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ r_y \end{bmatrix}$$

Given this general system configuration, the following sections show how to derive the system included, system included with noise, standalone with estimator, and standalone without estimator configurations.

System Included Configuration

In the system included configuration, the output error is defined by the following equation.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x})$$

By substituting the output error in the general system equations and removing the sensor noise r_y from the system, you obtain the following equations that describe the system included configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K - L\boldsymbol{C} & L\boldsymbol{C} \\ -\boldsymbol{B}K & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}K \\ \boldsymbol{B}K \end{bmatrix} r_x$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K & 0 \\ \boldsymbol{C} - \boldsymbol{D}K & 0 \\ -\boldsymbol{D}K & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} K \\ \boldsymbol{D}K \\ \boldsymbol{D}K \end{bmatrix} r_x$$

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ x \end{bmatrix}$, r_x, and $\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix}$, respectively.

Figure 12-5 represents the system described by these equations.

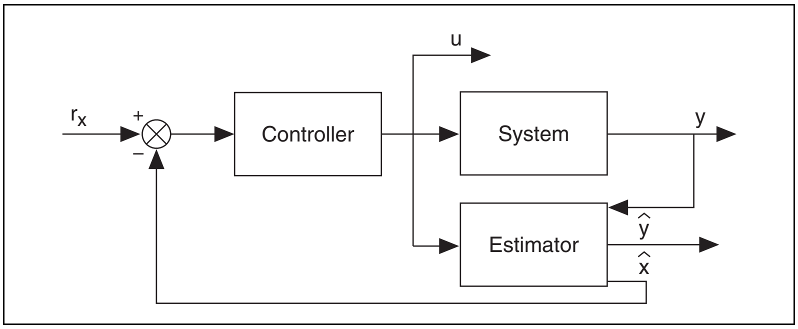


Figure 12-5. Compensator Included

The reference vector r_x has as many elements as the number of states. Also the control action u is internally calculated and given as an output of the state compensator.

System Included with Noise Configuration

In the system included with noise configuration, there is a sensor noise input r_y in the system. The output error is defined by the following equation.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x}) + r_y$$

By substituting the output error in the general system equations, you obtain the following equations that describe the system included with noise configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K - L\boldsymbol{C} & L\boldsymbol{C} \\ -\boldsymbol{B}K & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}K & L \\ \boldsymbol{B}K & 0 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K & 0 \\ \boldsymbol{C} - \boldsymbol{D}K & 0 \\ -\boldsymbol{D}K & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} K & 0 \\ \boldsymbol{D}K & 0 \\ \boldsymbol{D}K & I \end{bmatrix} \begin{bmatrix} r_x \\ r_y \end{bmatrix}$$

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ x \end{bmatrix}$, $\begin{bmatrix} r_x \\ r_y \end{bmatrix}$, and $\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix}$, respectively.

Figure 12-6 represents the system described by these equations.

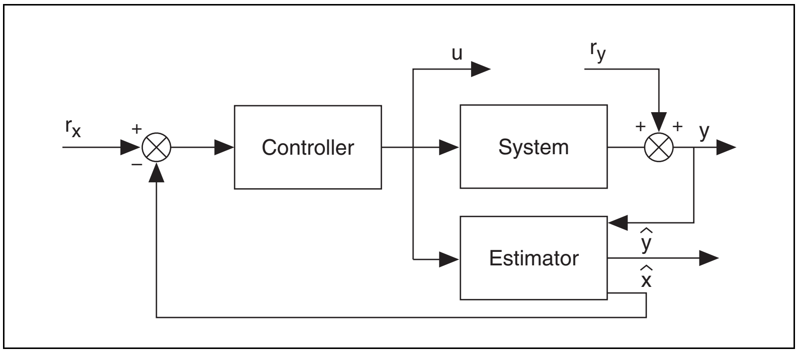


Figure 12-6. Compensator with Noise

The reference vector r_x has as many elements as the number of states. Also, the control action *u* is internally calculated and given as an output of the state compensator.

Standalone Configuration with Estimator

In the standalone configuration, the system model is detached from the controller and the system outputs *y* become inputs. Unlike the system included and system included with noise configurations, there is no output error. In the CD State-Space Controller VI, you must wire a value to the **Estimator Gain (L)** input to include the estimator in the standalone compensator.

The following equations describe the standalone configuration.

$$\dot{x} = [\boldsymbol{A} - \boldsymbol{B}K - L(\boldsymbol{C} - \boldsymbol{D}K)]\hat{x} + \begin{bmatrix}\boldsymbol{B}K - L\boldsymbol{D}K & L\end{bmatrix}\begin{bmatrix} r_x \\ y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \end{bmatrix} = \begin{bmatrix} -K \\ \boldsymbol{C} - \boldsymbol{D}K \end{bmatrix}\hat{x} + \begin{bmatrix} K & 0 \\ \boldsymbol{D}K & 0 \end{bmatrix}\begin{bmatrix} r_x \\ y \end{bmatrix}$$

The original system is not appended, meaning the system output is not generated internally but is considered as another input to the estimator, as shown in Figure 12-7.

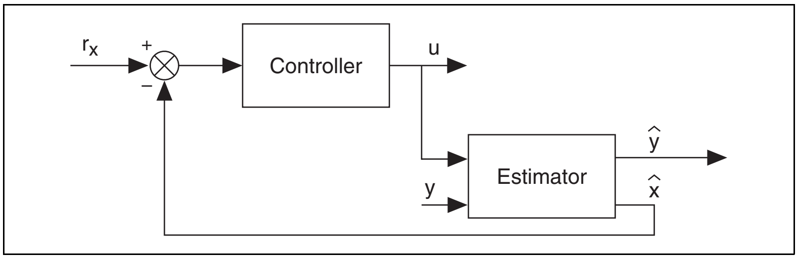


Figure 12-7. Standalone Compensator with Estimator

The states, inputs, and outputs of the estimator are $\hat{\boldsymbol{x}}$, $\begin{bmatrix} r_x \\ y \end{bmatrix}$, and $\begin{bmatrix} u \\ \hat{y} \end{bmatrix}$, respectively.

Standalone Configuration without Estimator

In the standalone without estimator configuration, all system states are used in the calculation of the control action *u*. Therefore, you do not need an estimator. In the CD State-Space Controller VI, do not wire a value to the **Estimator Gain (L)** input to exclude the estimator in the standalone compensator.

The following equations describe the standalone configuration.

$$\begin{aligned} \dot{x} &= (\boldsymbol{A} - \boldsymbol{B}K)x + \boldsymbol{B}Kr_x \\ y &= (\boldsymbol{C} - \boldsymbol{D}K)x + \boldsymbol{D}Kr_x \end{aligned}$$

The states and outputs of the standalone compensator without estimator correspond to the states and outputs of the actual system, as shown in Figure 12-8.

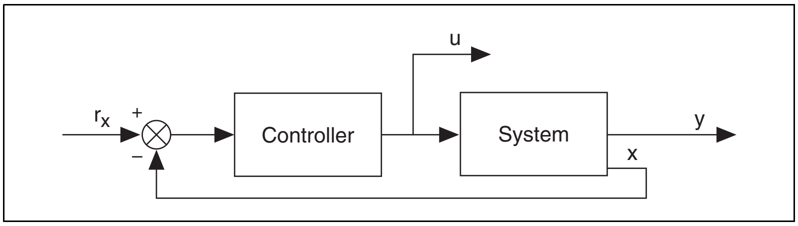


Figure 12-8. Standalone Compensator without Estimator

The states, inputs, and outputs of the estimator are x, r_x, and $\begin{bmatrix} u \\ y \end{bmatrix}$, respectively.

Table 12-1 summarizes the different state compensator configurations and their corresponding states, inputs, and outputs.

Table 12-1. State Compensator Configurations

Configuration Type	States	Inputs	Outputs
System Included	$[\hat{\boldsymbol{x}}, \boldsymbol{x}]$	$\boldsymbol{r}_x$	$[\boldsymbol{u}, \hat{\boldsymbol{y}}, \boldsymbol{y}]$
System Included with Noise	$[\hat{\boldsymbol{x}}, \boldsymbol{x}]$	$[\boldsymbol{r}_x, \boldsymbol{r}_y]$	$[\boldsymbol{u}, \hat{\boldsymbol{y}}, \boldsymbol{y}]$
Standalone with Estimator	$\hat{\boldsymbol{x}}$	$[\boldsymbol{r}_x, \boldsymbol{y}]$	$[\boldsymbol{u}, \hat{\boldsymbol{y}}]$
Standalone without Estimator	$\boldsymbol{x}$	$[\boldsymbol{r}_x]$	$[\boldsymbol{u}, \boldsymbol{y}]$

State Regulator

A general system configuration is one where the original model states (x) are appended to the estimation model states ($\hat{x}$) to represent the state regulator with an estimator, as shown in the following equations.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K & 0 \\ -\boldsymbol{B}K & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} & L \\ \boldsymbol{B} & 0 \end{bmatrix} \begin{bmatrix} r_u \\ y - \hat{y} \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K & 0 \\ \boldsymbol{C} - \boldsymbol{D}K & 0 \\ -\boldsymbol{D}K & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{I} & 0 \\ \boldsymbol{D} & 0 \\ \boldsymbol{D} & 0 \end{bmatrix} \begin{bmatrix} r_u \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ r_y \end{bmatrix}$$

Given this general system configuration, the following sections show how to derive the system included, system included with noise, standalone with estimator, and standalone without estimator configurations.

System Included Configuration

In the system included configuration, the output error is defined by the following equation.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x})$$

By substituting the output error in the general system equations and removing the sensor noise r_y from the system, you obtain the following equations that describe the system included configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K - L\boldsymbol{C} & L\boldsymbol{C} \\ -\boldsymbol{B}K & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B} \end{bmatrix} r_u$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K & 0 \\ \boldsymbol{C} - DK & 0 \\ -\boldsymbol{D}K & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{I} \\ D \\ \boldsymbol{D} \end{bmatrix} r_u$$

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ x \end{bmatrix}$, r_u, and $\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix}$, respectively.

Figure 12-9 represents the system described by these equations.

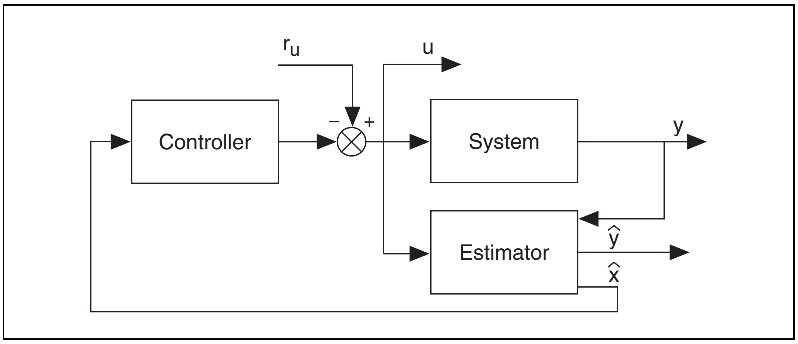


Figure 12-9. Regulator Included

The reference vector, or actuator noise, r_u has as many elements as the number of inputs. Also, the control action u is internally calculated and given as an output of the state regulator.

System Included with Noise Configuration

In the system included with noise configuration, there is a sensor noise input r_y in the system. The output error is defined by the following equation.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x}) + r_y$$

By substituting the output error in the general system equations, you obtain the following equations that describe the system included with noise configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K - L\boldsymbol{C} & L\boldsymbol{C} \\ -\boldsymbol{B}K & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} & L \\ \boldsymbol{B} & 0 \end{bmatrix} \begin{bmatrix} r_u \\ r_y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K & 0 \\ \boldsymbol{C} - \boldsymbol{D}K & 0 \\ -\boldsymbol{D}K & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{I} & 0 \\ \boldsymbol{D} & 0 \\ \boldsymbol{D} & I \end{bmatrix} \begin{bmatrix} r_u \\ r_y \end{bmatrix}$$

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ x \end{bmatrix}$, $\begin{bmatrix} r_u \\ r_y \end{bmatrix}$, and $\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix}$, respectively.

Figure 12-10 represents the system described by these equations.

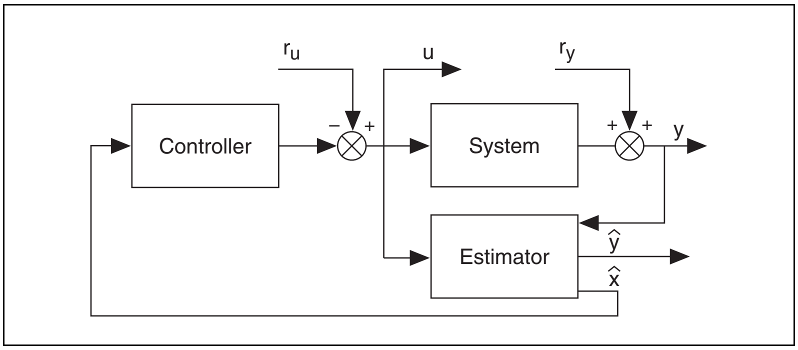


Figure 12-10. Regulator with Noise

The reference vector, or actuator noise, r_u has as many elements as the number of inputs. Also the control action u is internally calculated and given as an output of the state regulator.

Standalone Configuration with Estimator

In the standalone configuration, the system is detached from the controller and the system outputs y become inputs. Unlike the system included and system included with noise configurations, there is no output error. In the CD State-Space Controller VI, you must wire a value to the **Estimator Gain (L)** input to include the estimator in the standalone regulator.

The following equations describe the standalone configuration.

$$\dot{\hat{x}} = [\boldsymbol{A} - \boldsymbol{B}K - \hat{L}(\boldsymbol{C} - \boldsymbol{D}K)]\hat{x} + \begin{bmatrix} \boldsymbol{B} - L\boldsymbol{D} & L \end{bmatrix} \begin{bmatrix} r_u \\ y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \end{bmatrix} = \begin{bmatrix} -K \\ \boldsymbol{C} - \boldsymbol{D}K \end{bmatrix} \hat{x} + \begin{bmatrix} \boldsymbol{I} & 0 \\ \boldsymbol{D} & 0 \end{bmatrix} \begin{bmatrix} r_u \\ y \end{bmatrix}$$

The original system is not appended, meaning the system output is not generated internally but is considered as another input to the estimator, as shown in Figure 12-11.

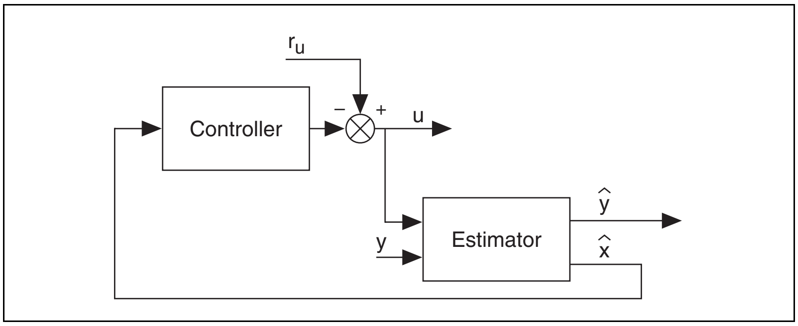


Figure 12-11. Standalone Regulator with Estimator

The states, inputs, and outputs of the estimator are $\hat{x}$, $\begin{bmatrix} r_u \\ y \end{bmatrix}$, and $\begin{bmatrix} u \\ \hat{y} \end{bmatrix}$, respectively.

Standalone Configuration without Estimator

In the standalone without estimator configuration, all system states are used in the calculation of the control action u. Therefore, you do not need an estimator. In the CD State-Space Controller VI, do not wire a value to the **Estimator Gain (L)** input to exclude the estimator in the standalone regulator.

The following equations describe the standalone configuration.

$$\dot{\hat{x}} = (\boldsymbol{A} - \boldsymbol{B}K)x + \boldsymbol{B}r_u$$

$$y = (\boldsymbol{C} - \boldsymbol{D}K)x + \boldsymbol{D}r_u$$

The states and outputs of the standalone regulator without estimator corresponds to the states and outputs of the actual system, as shown in Figure 12-12.

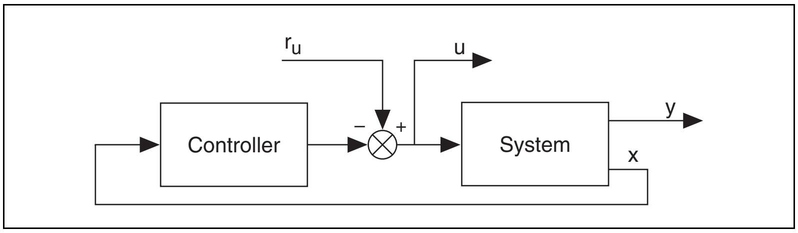


Figure 12-12. Standalone Regulator without Estimator

The states, inputs, and outputs of the estimator are x, r_u, and $\begin{bmatrix} u \\ y \end{bmatrix}$, respectively.

Table 12-2 summarizes the different state regulator configurations and their corresponding states, inputs, and outputs.

Table 12-2. Regulator Configuration Types

Configuration Type	States	Inputs	Outputs
System Included	$[\hat{\boldsymbol{x}},\ \boldsymbol{x}]$	$\boldsymbol{r}_u$	$[\boldsymbol{u},\ \hat{\boldsymbol{y}},\ \boldsymbol{y}]$
System Included with Noise	$[\hat{\boldsymbol{x}},\ \boldsymbol{x}]$	$[\boldsymbol{r}_u,\ \boldsymbol{r}_y]$	$[\boldsymbol{u},\ \hat{\boldsymbol{y}},\ \boldsymbol{y}]$
Standalone with Estimator	$\hat{\boldsymbol{x}}$	$[\boldsymbol{r}_u,\ \boldsymbol{y}]$	$[\boldsymbol{u},\ \hat{\boldsymbol{y}}]$
Standalone without Estimator	$\boldsymbol{x}$	$[\boldsymbol{r}_u]$	$[\boldsymbol{u},\ \boldsymbol{y}]$

Regulator with Integral Action

A general system configuration is one where the output error integrator (z) is appended to the estimation model states ($\hat{x}$). In addition, resulting vector ($\hat{x}$, z) is augmented with the original model states (x) to represent the state regulator with integral action and an estimator.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & 0 & 0 \\ 0 & \Gamma & 0 \\ 0 & 0 & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} & 0 & L \\ 0 & \boldsymbol{I} & 0 \\ \boldsymbol{B} & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ y_{ref} - y \\ y - \hat{y} \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K_x & -K_i & 0 \\ \boldsymbol{C} & 0 & 0 \\ 0 & 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \boldsymbol{D} & 0 & 0 \\ \boldsymbol{D} & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ y_{ref} - y \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ r_y \end{bmatrix}$$

Note Γ in these equations varies depending on whether the model describes a continuous or discrete system. $\Gamma = 0$ if the system is continuous and $\Gamma = \boldsymbol{I}$ if the system is discrete.

When you define the control action for a state regulator with integral action using the output error integrator z, you obtain the following control action equation.

$$u = -\begin{bmatrix} K & K_I \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}$$

Substituting the control action into state dynamics of the general system configuration defined in the previous equation, you obtain the following equation that also defines the general system configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K_x & 0 - \boldsymbol{B}K_i & 0 \\ 0 & 0 & 0 \\ -\boldsymbol{B}K_x & -\boldsymbol{B}K_i & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 & \boldsymbol{L} \\ \boldsymbol{I} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{ref} - y \\ y - \hat{y} \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K_x & -K_i & 0 \\ \boldsymbol{C} - \boldsymbol{D}K_x & -\boldsymbol{D}K_i & 0 \\ -\boldsymbol{D}K_x & -\boldsymbol{D}K_i & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{ref} - y \\ y - \hat{y} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ r_y \end{bmatrix}$$

Given this general system configuration, the following sections show how to derive the system included, system included with noise, standalone with estimator, and standalone without estimator configurations.

System Included Configuration

In the system included configuration, the output error and system output are defined by the following equations.

$$y - \hat{y} = \boldsymbol{C}(x - \hat{x})$$
$$y = -\boldsymbol{D}(K_x\hat{x} + K_i z) + \boldsymbol{C}x$$

By substituting the output error and system output in the general system equations and removing the sensor noise r_y from the system, you obtain the following equations that describe the system included configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K_x - L\boldsymbol{C} & -\boldsymbol{B}K_i & L\boldsymbol{C} \\ \boldsymbol{D}K_x & \Gamma + \boldsymbol{D}K_i & -\boldsymbol{C} \\ -\boldsymbol{B}K_x & -\boldsymbol{B}K_i & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{I} \\ 0 \end{bmatrix} y_{ref}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K_x & -K_i & 0 \\ \boldsymbol{C} - \boldsymbol{D}K_x & -\boldsymbol{D}K_i & 0 \\ -\boldsymbol{D}K_x & -\boldsymbol{D}K_i & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} y_{ref}$$

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix}$, y_{ref}, and $\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix}$, respectively.

Figure 12-13 represents the system described by these equations.

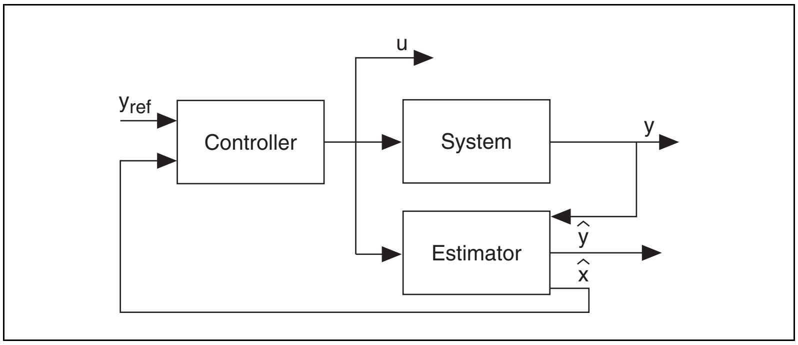


Figure 12-13. Integral Included

The reference vector y_{ref} has as many elements as the number of outputs. Also the control action u is internally calculated and given as an output of the state regulator with integral action.

System Included with Noise Configuration

In the system included with noise configuration, there is a sensor noise input r_y in the system. The output error and system output are defined by the following equations.

$$y - \hat{y} = -\boldsymbol{C}\hat{x} + \boldsymbol{C}x + r_y$$
$$y = -\boldsymbol{D}(K_x\hat{x} + K_i z) + \boldsymbol{C}z + r_y$$

By substituting the output error and system output in the general system equations, you obtain the following equations that describe the system included with noise configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K_x - L\boldsymbol{C} & -\boldsymbol{B}K_i & L\boldsymbol{C} \\ \boldsymbol{D}K_x & \Gamma + \boldsymbol{D}K_i & -\boldsymbol{C} \\ -\boldsymbol{B}K_x & -\boldsymbol{B}K_i & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 & L \\ \boldsymbol{I} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{ref} \\ r_y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} -K_x & -K_i & 0 \\ \boldsymbol{C} - \boldsymbol{D}K_x & -\boldsymbol{D}K_i & 0 \\ -\boldsymbol{D}K_x & -\boldsymbol{D}K_i & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} y_{ref} \\ r_y \end{bmatrix}$$

The states, inputs, and outputs of the estimator are $\begin{bmatrix} \hat{x} \\ z \\ x \end{bmatrix}$, $\begin{bmatrix} y_{ref} \\ r_y \end{bmatrix}$, and $\begin{bmatrix} u \\ \hat{y} \\ y \end{bmatrix}$, respectively.

Figure 12-14 represents the system described by these equations.

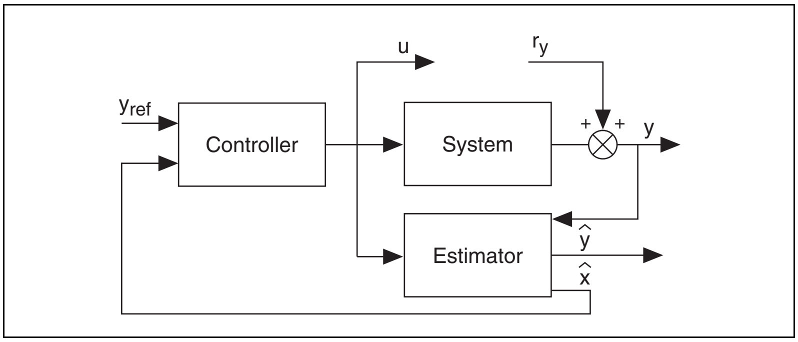


Figure 12-14. Integral with Noise

The reference vector y_{ref} has as many elements as the number of outputs. Also, the control action u is internally calculated and given as an output of the state regulator with integral action.

Standalone with Estimator Configuration

In the standalone configuration, the system is detached from the controller and the system outputs *y* become inputs. Unlike the system included and system included with noise configurations, there is no output error. In the CD State-Space Controller VI, you must wire a value to the **Estimator Gain (L)** input to include the estimator in the standalone regulator with integral action.

The following equations describe the standalone configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - (\boldsymbol{B} - L\boldsymbol{D})K_x - L\boldsymbol{C} & (\boldsymbol{D} - \boldsymbol{B})K_i \\ 0 & \Gamma \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} + \begin{bmatrix} 0 & L \\ \boldsymbol{I} & 0 \end{bmatrix} \begin{bmatrix} y_{ref} - y \\ y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \end{bmatrix} = \begin{bmatrix} -K_x & -K_i \\ \boldsymbol{C} - \boldsymbol{D}K_x & -\boldsymbol{D}K_i \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{ref} - y \\ y \end{bmatrix}$$

To make the inputs independent, use the following substitution.

$$\begin{bmatrix} 0 & L \\ \boldsymbol{I} & 0 \end{bmatrix} \begin{bmatrix} y_{ref} - y \\ y \end{bmatrix} = \begin{bmatrix} Ly \\ y_{ref} - y \end{bmatrix} = \begin{bmatrix} 0 & L \\ \boldsymbol{I} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} y_{ref} \\ y \end{bmatrix}$$

The equations describing the standalone configuration then become defined as follows.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - (\boldsymbol{B} - L\boldsymbol{D}) & K_x - L\boldsymbol{C}(\boldsymbol{D} - \boldsymbol{B})K_i \\ 0 & \Gamma \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} + \begin{bmatrix} 0 & L \\ \boldsymbol{I} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} y_{ref} \\ y \end{bmatrix}$$

$$\begin{bmatrix} u \\ \hat{y} \end{bmatrix} = \begin{bmatrix} -K_x & -K_i \\ \boldsymbol{C} - \boldsymbol{D}K_x & -\boldsymbol{D}K_i \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{ref} \\ y \end{bmatrix}$$

The original system is not appended, meaning the system output is not generated internally but is considered as another input to the estimator, as shown in Figure 12-15.

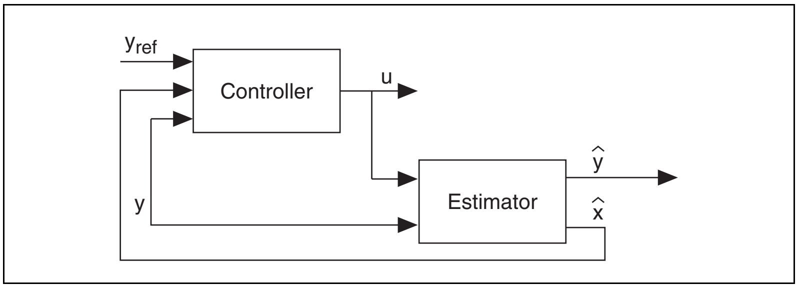


Figure 12-15. Standalone Integral with Estimator

The states, inputs, and outputs of the estimator are $\hat{x}$, $\begin{bmatrix} y_{ref} \\ y \end{bmatrix}$, and $\begin{bmatrix} u \\ \hat{y} \end{bmatrix}$, respectively.

Standalone without Estimator Configuration

In the standalone without estimator configuration, all system states are used in the calculation of the control action *u*. Therefore, you do not need an estimator. In the CD State-Space Controller VI, do not wire a value to the **Estimator Gain (L)** input to exclude the estimator in the standalone regulator with integral action.

The following equations describe the standalone configuration.

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}K_x & -\boldsymbol{B}K_i \\ 0 & \Gamma \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix} (y_{ref} - y)$$

$$\begin{bmatrix} u \\ \hat{y} \end{bmatrix} = \begin{bmatrix} -K_x & -K_i \\ \boldsymbol{C} - \boldsymbol{D}K_x & -\boldsymbol{D}K_i \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} (y_{ref} - y)$$

To make the inputs independent, use the following substitution.

$$\begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix} (y_{ref} - y) = \begin{bmatrix} 0 \\ y_{ref} - y \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \boldsymbol{I} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} y_{ref} \\ y \end{bmatrix}$$

The equations describing the standalone configuration then become defined as follows.

$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} = \begin{bmatrix} \boldsymbol{A}-\boldsymbol{B}K_x & -\boldsymbol{B}K_i \\ 0 & \Gamma\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}0 & 0\\ \boldsymbol{I} & -\boldsymbol{I}\end{bmatrix}\begin{bmatrix}y_{ref}\\ y\end{bmatrix}$$

$$\begin{bmatrix}u\\ y\end{bmatrix} = \begin{bmatrix} -K_x & -K_i \\ \boldsymbol{C}-\boldsymbol{D}K_x & -\boldsymbol{D}K_i\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}y_{ref}\\ y\end{bmatrix}$$

The states and outputs of the standalone regulator without estimator corresponds to the states and outputs of the actual system, as shown in Figure 12-16.

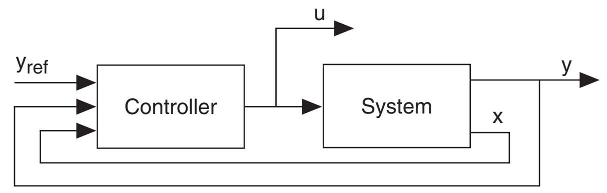


Figure 12-16. Standalone Integral without Estimator

The states, inputs, and outputs of the estimator are $\begin{bmatrix}x\\ z\end{bmatrix}$, $\begin{bmatrix}y_{ref}\\ y\end{bmatrix}$, and $\begin{bmatrix}u\\ y\end{bmatrix}$, respectively.

Table 12-3 summarizes the different state regulator with integral action configurations and their corresponding states, inputs, and outputs.

Table 12-3. Regulator with Integral Action Configuration Types

Configuration Type	States	Inputs	Outputs
System Included	$[\hat{\boldsymbol{x}},\ \boldsymbol{x}]$	$\boldsymbol{y}_{ref}$	$[\boldsymbol{u},\ \hat{\boldsymbol{y}},\ \boldsymbol{y}]$
System Included with Noise	$[\hat{\boldsymbol{x}},\ z,\ \boldsymbol{x}]$	$[\boldsymbol{y}_{ref},\ \boldsymbol{r}_y]$	$[\boldsymbol{u},\ \hat{\boldsymbol{y}},\ \boldsymbol{y}]$
Standalone with Estimator	$\hat{\boldsymbol{x}}$	$[\boldsymbol{y}_{ref}-\boldsymbol{y},\ \boldsymbol{y}]$	$[\boldsymbol{u},\ \hat{\boldsymbol{y}}]$
Standalone without Estimator	$\boldsymbol{x}$	$\boldsymbol{y}_{ref}$	$[\boldsymbol{u},\ \boldsymbol{y}]$

13 Synthesizing a Controller

The LabVIEW Control Design Toolkit enables you to implement a controller, as described in Chapter 12, *Modern Control Design Synthesis*, and then deploy that controller to an RT target. This chapter provides two examples that show how to implement a state estimator and a state compensator using the Control Design VIs.

Implementing a State Estimator

Consider the following continuous second-order SISO state-space model with poles at –0.2 and –0.1.

$$\dot{x} = \begin{bmatrix} -0.2 & 0.5 \\ 0 & -0.1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \end{bmatrix} u$$

This system is observable so you can define a full state estimator for this system. To implement a state estimator, you need to calculate the estimator gain L for this state-space model. The CD Ackermann VI enables you to calculate the estimator gain L by placing the poles of the matrix $\boldsymbol{A} - \boldsymbol{LC}$ at [–1, –1]. This location is to the left of the original pole location in the complex plane. You can use this estimator gain L to study the performance of the estimator using the CD State Estimator VI.

Note You can use the CD Observability Matrix VI to verify that this system is observable.

The following sections describe the three available configurations—system included, system included with noise, and standalone—for implementing a state estimator using the Control Design Toolkit. The state estimators implemented in the following sections are based on the previously defined second-order SISO state-space model.

Refer to the *State Estimator* section of Chapter 12, *Modern Control Design Synthesis*, for more information about the different configuration methods.

Creating a System Included State Estimator

The example in Figure 13-1 uses the CD Ackermann VI to determine the estimator gain *L* of the second-order SISO **State-Space Model**. The CD State Estimator VI uses the gain calculated by the CD Ackermann VI to create the state estimator, represented by the **Estimator Model**, for the system. The CD State Estimator VI uses the system included configuration in implementing the state estimator. The CD Initial Response VI enables you to study the performance of the state estimator.

Note The examples in this chapter use the CD Ackermann VI to calculate the estimator and controller gain values. However, the CD Ackermann VI is only one VI you can use to calculate these values. You also can use other VIs, such as the CD Pole Placement VI, the CD Kalman Gain VI, or the CD Linear Quadratic Regulator VI, to calculate estimator and controller gain values.

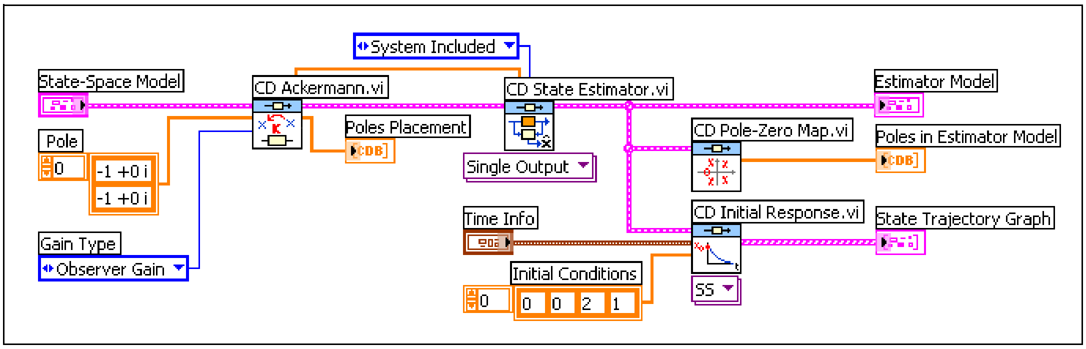


Figure 13-1. System Included Estimator

The system included configuration creates an **Estimator Model** that represents the original, or actual, states of the system and the estimated states in the same model. The **Estimator Model** consists of four states because the original second-order SISO state-space model is appended to the state estimator, as shown in the following expression.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{LC} & \boldsymbol{LC} \\ 0 & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B} \end{bmatrix} u$$

$$\begin{bmatrix} \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} & 0 \\ 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix}$$

Note The system matrix **D** is not part of the expression because it is null in this example.

The system included configuration monitors the response of the actual states of the system to a set of initial conditions. The CD Initial Response VI uses [0, 0, 2, 1] as the initial conditions. These initial conditions mean that the initial conditions of the actual states are [2, 1], while initial conditions of the estimated states are [0, 0]. Therefore, the initial condition vector of the **Estimator Model** is [0, 0, 2, 1], as shown in Figure 13-1.

The **State Trajectory Graph**, as shown in Figure 13-2, displays the response of the system and state estimator to the initial conditions [0, 0, 2, 1].

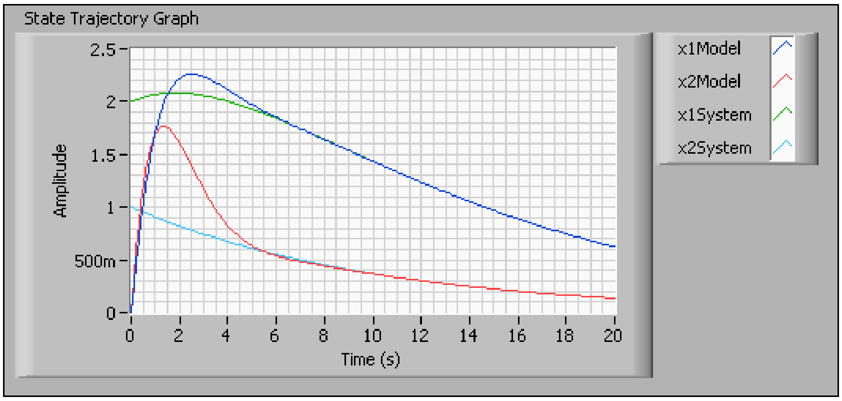


Figure 13-2. State Trajectory System Included

The initial conditions of the actual states are [2, 1], therefore the response of the actual states starts at 2 and 1. The initial conditions of the estimated states are [0, 0], therefore the response of the estimated states start at the origin. The estimated states promptly begin to track the actual states as the response of the actual system settles to steady state. This state estimator takes approximately six seconds to track the response of the system.

Creating a System Included State Estimator with Noise

In theory, you can place the poles of the state estimator as far left of the complex plane as necessary. This placement leads to very aggressive state estimators. However, noise and system uncertainties prevent you from configuring such aggressive estimators. Implementing a state estimator using the system included with noise configuration enables you to test a state estimator that has output noise. Consider the following system included with noise configuration.

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{LC} & \boldsymbol{LC} \\ 0 & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B} \end{bmatrix} u$$

$$\begin{bmatrix} \hat{y} \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{C} & 0 \\ 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix} r_y$$

The configuration of this system is the same as the system in the *Creating a System Included State Estimator* section with the addition of the output noise r_y. Assume that the output noise in this example is a Gaussian noise in the system. The output noise influences the estimated model dynamics through the estimator gain L. The larger the estimator gain, the larger the effect of the noise when calculating the estimated states. Figure 13-3 shows how to account for a Gaussian noise of 0.1 standard deviation in the **Estimator Model**.

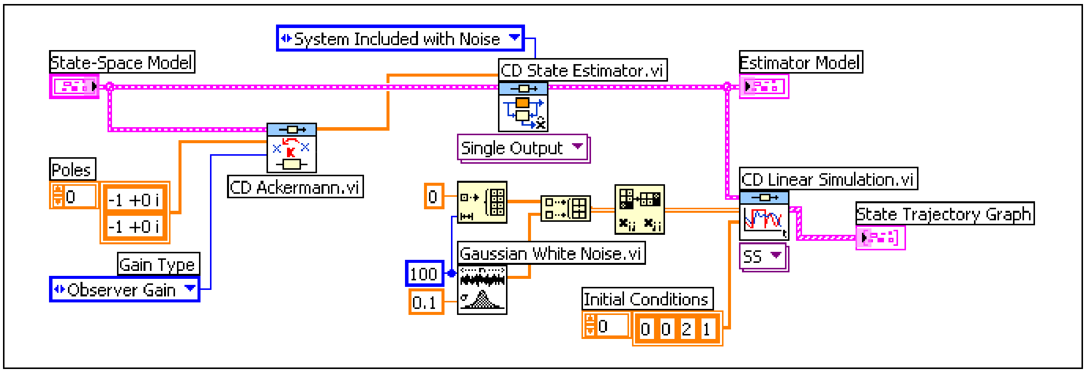


Figure 13-3. System Included with Noise Estimator

The example in Figure 13-3 uses the state-space model and the CD Ackermann VI to determine the estimator gain L. The CD State Estimator VI then uses the system included with noise configuration to implement the state estimator, represented by the **Estimator Model**. The Gaussian White Noise VI enables you to view the effects of Gaussian noise on the system and the state estimator.

Note The CD Linear Simulation VI provides the response to a Gaussian noise with the same initial conditions as in Figure 13-1.

The **State Trajectory Graph**, as shown in Figure 13-4, displays the response of the system and state estimator to the same initial conditions [0, 0, 2, 1] used in the *Creating a System Included State Estimator* section.

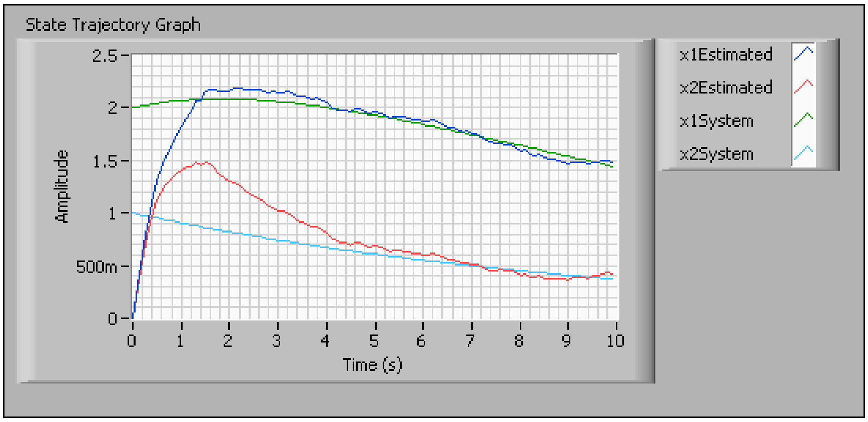


Figure 13-4. State Trajectory Noise

Similar to the graph in the *Creating a System Included State Estimator* section, this state-trajectory graph shows the response of the actual states starting at 2 and 1. The graph also shows the response of the estimated states starting at the origin. Notice the effect of the output noise r_y on the state estimation. Without noise, the state estimator took approximately 6 seconds to begin tracking the actual system. With noise, the state estimator takes much longer to track the actual system and the state estimator cannot track the actual system perfectly.

You can place the estimator poles closer to the origin to reduce the effect of the noise. However, when you move the estimator poles closer to the origin on the left side of the complex plane, you diminish the performance of the estimator in tracking the actual states.

One solution is to use the Kalman filter to obtain an estimator gain that effectively tracks the system states with an acceptable level of noise rejection. Refer to the *Kalman Gain* section of Chapter 12, *Modern Control Design Synthesis*, for information about using the CD Kalman Gain VI to find an optimal solution to this state estimator problem.

Creating a Standalone Estimator

Most systems are complex and have many parameters and uncertainties. You often do not know all the parameters of a system when you create a model of that system, or you cannot create a model that encompasses all the uncertainties of the system. Thus the actual system and the model of the system do not match.

When you build a state estimator based on a model that does not match the actual system, the result is a system-model mismatch. Therefore, you need to use the standalone configuration. The standalone configuration detaches the model used in the state estimator from the actual system.

Consider the following state-space model. This model represents the same system that the model in the *Implementing a State Estimator* section represented. However, for this example, assume that the actual system contains uncertainties that cause this state-space model to be an inaccurate representation of the system.

$$\dot{x} = \begin{bmatrix} -0.1 & 0.5 \\ 0 & -0.1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \end{bmatrix} u$$

Therefore, because of system uncertainties, this state-space model does not match the model in the *Implementing a State Estimator* section. The difference is in the first entry of the system matrix **A**.

The block diagram in Figure 13-5 shows how the mismatched model, **State-Space Model**, is used to created the standalone state estimator using the CD State Estimator VI. Then the actual system, **System**, and the mismatched model, **State-Space Model**, are connected in series so the actual system can provide the output *y* to the standalone state estimator.

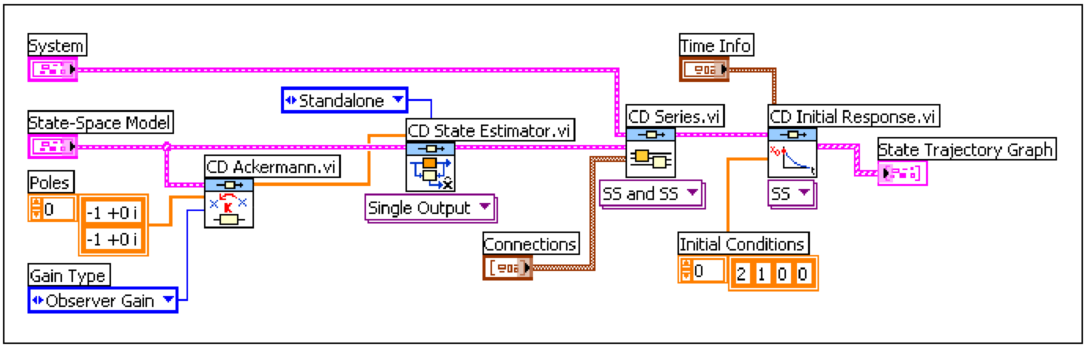


Figure 13-5. Standalone Estimator

The example uses the CD Initial Response VI to evaluate the effectiveness of the state estimator. The **State Trajectory Graph** in Figure 13-6 shows the response of the actual and estimated states to the same set of initial conditions as in the *Creating a System Included State Estimator* section.

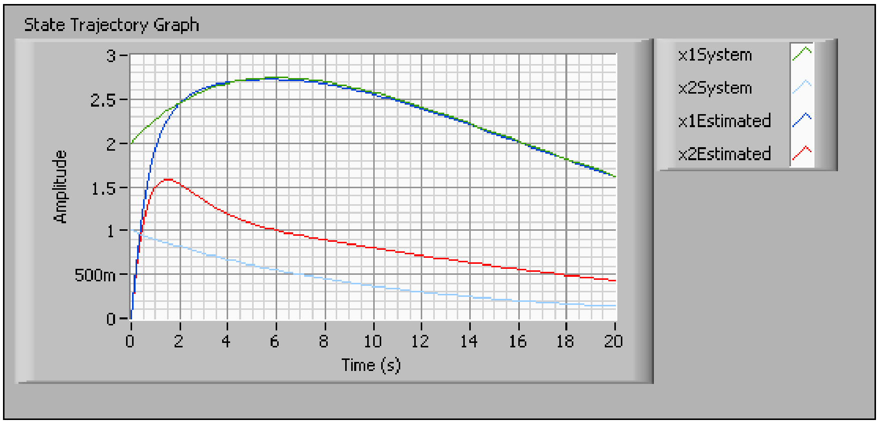


Figure 13-6. State Trajectory Standalone

Notice that a mismatch in the actual system and the model of the system greatly impacts the estimation of the second state. After 20 seconds, the state estimator is still unable to track the actual state. Therefore, you must study the system and model mismatch to determine the effect of the mismatch on the state estimation.

Implementing a State Compensator

Consider the following state-space model with poles at –0.2 and –0.1.

$$\dot{x} = \begin{bmatrix} -0.2 & 0.5 \\ 0.1 & -0.1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \end{bmatrix} u$$

Figure 13-7 shows the response of this system to initial conditions of [2, 1]. This system is unstable because the response increases exponentially and does not settle at a steady state value.

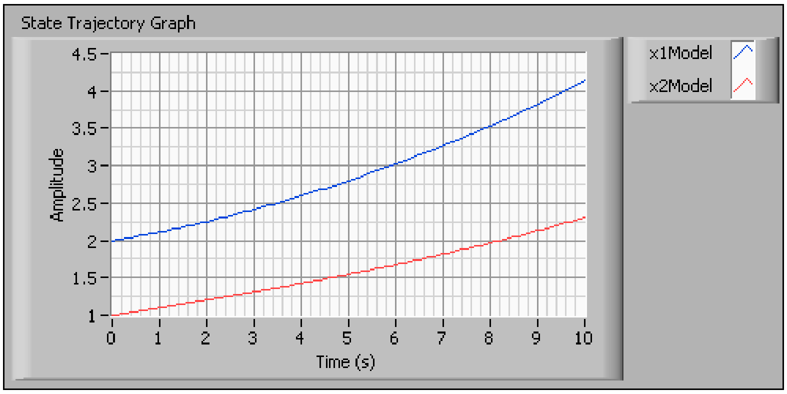


Figure 13-7. Unstable Open-Loop System

However, this system is controllable so you can use a state compensator to place the closed-loop poles in the left-hand side of the complex plane to make the response stable. You can calculate the controller gain K by using the CD Ackermann VI to place the poles of the matrix $\boldsymbol{A} - \boldsymbol{B}K$ at [–1, –1]. You can use this gain to study the performance of the compensator using the CD State Compensator VI.

The following sections describe the four available configurations—system included, system included with noise, standalone, and standalone without estimator—for implementing a state compensator using the Control Design Toolkit.

Refer to the *State Compensator* section of Chapter 12, *Modern Control Design Synthesis*, for more information about the different configuration methods.

Creating a Standalone Compensator without Estimator

This state compensator uses the standalone without estimator configuration, which indicates that you do not need a state estimator because the states are directly available for control. The compensator model is described by the following equations.

$$\dot{x} = (\boldsymbol{A} - \boldsymbol{B}K)\boldsymbol{x} + \boldsymbol{B}Kr_x$$
$$y = \boldsymbol{C}x$$

Note The system matrix ***D*** is not part of the expression because it is null in this example.

The poles, or the eigenvalues of ***A*** – ***B****K*, of the closed-loop system are on the left side of the complex plane. If you set the output noise r_x to zero, the controller gain *K* immediately drives the states to zero.

Figure 13-8 shows how you use the CD Ackermann VI to calculate the controller gain *K*, which you then use to study the performance of the state compensator.

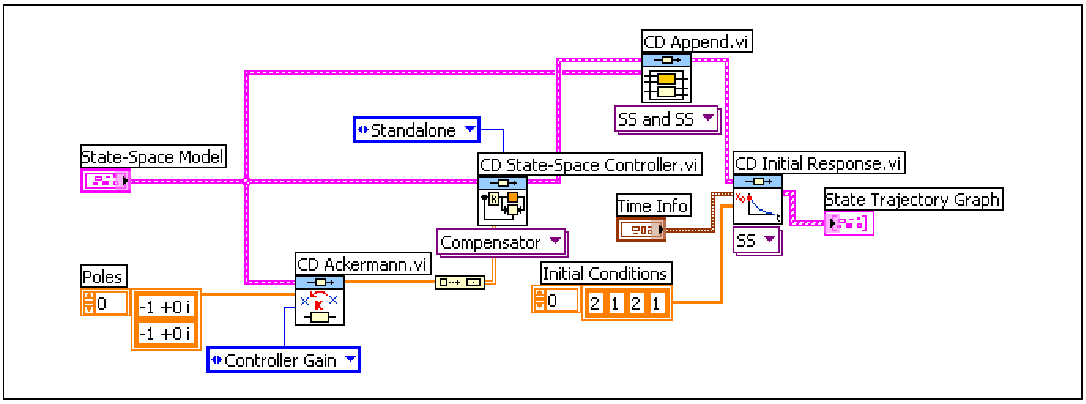


Figure 13-8. Standalone Compensator

Note To view both the original response of the actual system and the response of the system controlled by the state compensator, you must append the model of the actual system, **State-Space Model**, to the model of the state compensator. Therefore, in the **State Trajectory Graph**, shown in Figure 13-9, you can see the difference in the system response due to the effect of the compensator gain *K*.

By adding a state compensator to the actual system, you create a closed-loop model of the resulting system. The actual system, without a state compensator, is an open-loop system. Figure 13-9 shows the response of the open-loop and closed-loop systems to initial conditions of [2, 1].

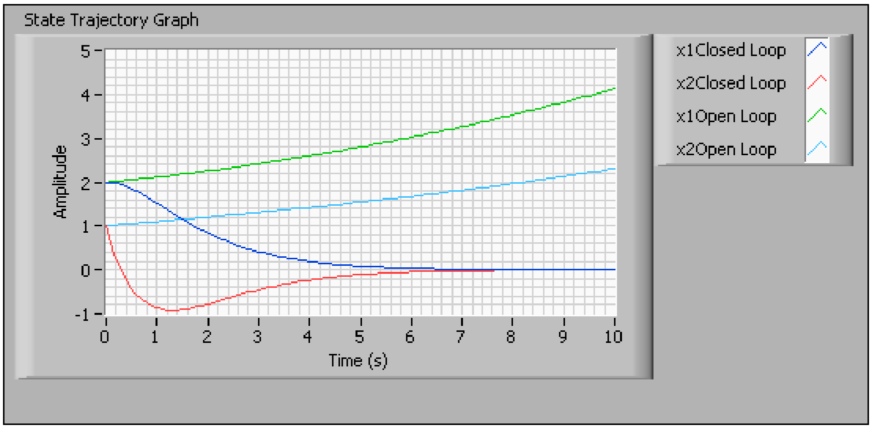


Figure 13-9. State Trajectory for Standalone Compensator

Notice that despite the instability of the actual system, the state compensator is able to drive the closed-loop states toward zero. Thus the addition of a state compensator to the actual system stabilizes the resulting system.

Because the standalone state compensator stabilizes the actual system, you must use a state compensator with this system. Therefore, the following examples describe the three remaining ways the Control Design Toolkit enables you to configure a state compensator.

Creating a System Included Compensator

In theory, you cannot always directly measure the system states for control purposes. Therefore, you must synthesize a controller using the system outputs. To calculate the control action based on the estimated states, the estimator needs to approach the actual states faster than the controller. Therefore, you can calculate an estimator gain such that $\boldsymbol{A} - \boldsymbol{LC}$ has eigenvalues at [–5, –5], which is farther to the left of the origin than the poles of the controller located at [–1, –1].

The system included configuration takes both the estimator gain L and the controller gain K and uses them to synthesize a state compensator. Figure 13-10 shows the implementation of a state compensator using the system included configuration.

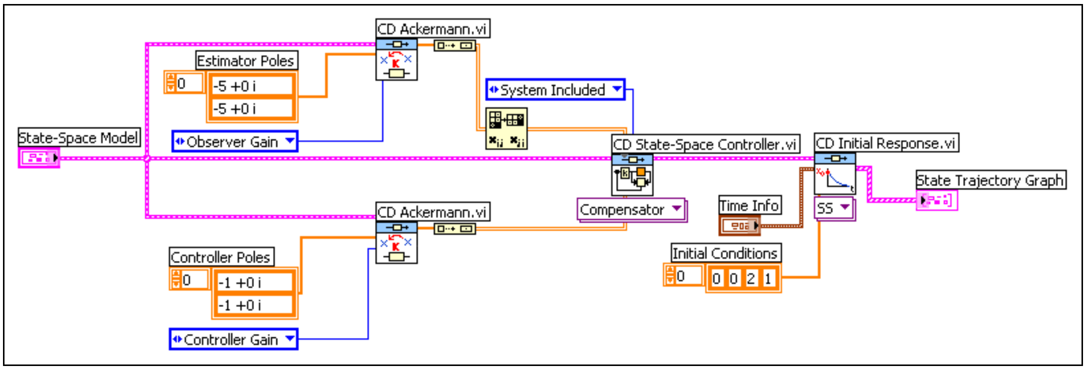


Figure 13-10. Compensator System Included

The CD Initial Response VI uses [0, 0, 2, 1] as the initial conditions. As in the *Creating a System Included State Estimator* section, these initial conditions mean that the initial conditions of the actual states are [2, 1], while initial conditions of the estimated states are [0, 0]. Figure 13-11 shows the response of the system to those initial conditions.

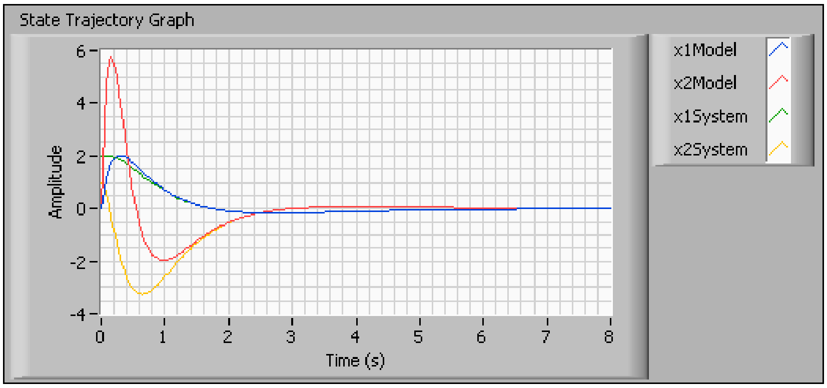


Figure 13-11. State Trajectory of Compensator with System Included

Notice that the time it takes for the estimator to track the actual states is much shorter than the time it takes for the actual states to reach a steady state. The estimator takes between 1 and 1.5 seconds to track the actual states, while the actual states take approximately 6 seconds to reach a steady state. The estimator tracks the actual states faster than the controller stabilizes the system because the estimator poles are at [–5, –5] and the controller poles are at [–1, –1]. Placing the poles of the estimator farther to the left than the controller poles makes the performance of the estimator better than the controller.

Creating a System Included Compensator with Noise

In general, the compensator accepts two inputs, r_x and r_y. The input r_x represents state set points. The input r_y represents output noise and is available only in the system included with noise configuration. Figure 13-12 shows the use of both types of inputs for the compensator.

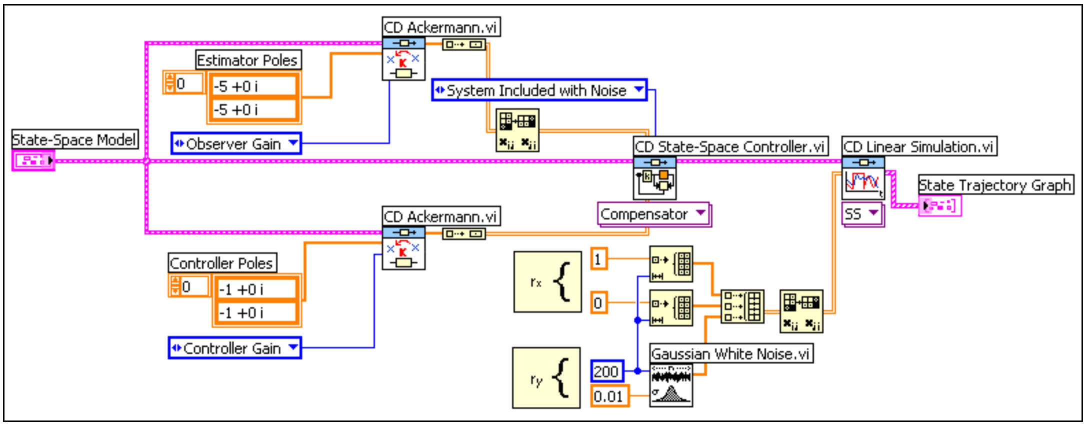


Figure 13-12. System Included with Noise Compensator

The system included with noise configuration analyzes the effect of output noise on the system. There are a total of three inputs in this example. The first two inputs are set points to the controller, given by $r_x = [1, 0]$. The last input represents the output noise r_y, which has a standard deviation of 0.01. Figure 13-13 shows the response to these inputs.

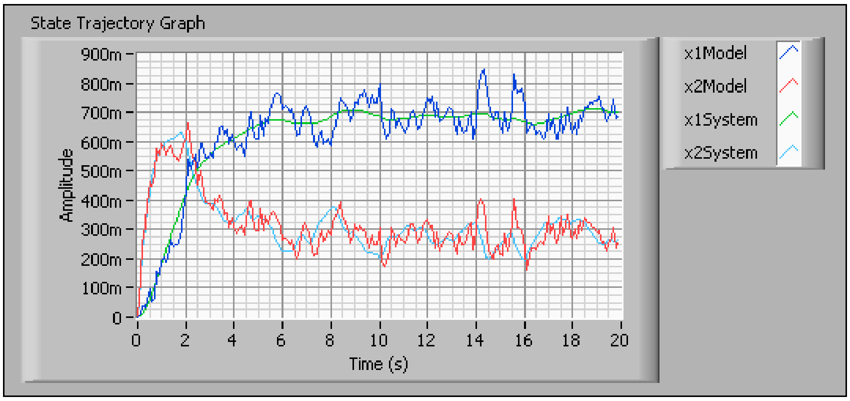


Figure 13-13. State Trajectory System Included with Noise Compensator

Notice that the state compensator lacks integral action, which originates the offsets on the state responses with respect to their respective set points. Therefore, the states do not reach the specified set points $r_x = [1, 0]$.

Creating a Standalone Compensator with Estimator

As described in the *Creating a Standalone Estimator* section, most systems are complex and have many uncertainties that might result in a model not describing a system accurately. In this situation, there is a system-model mismatch. When you build a state compensator based on a model that does not match the actual system, you need to use the standalone with estimator configuration. This configuration detaches the system from the model so you can determine the effect of the system-model mismatch. The standalone compensator with state estimation studies the effect of model uncertainties in a model-based controller driven by a model-based estimator.

Consider the following state-space model. This model represents the same system that the model in the *Implementing a State Compensator* section represented. However, for this example, assume that the actual system contains uncertainties that cause this state-space model to be an inaccurate representation of the system.

$$\dot{x} = \begin{bmatrix} -0.1 & 0.5 \\ 0 & -0.5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \end{bmatrix} u$$

Therefore, because of the system uncertainties, this state-space model does not match the model in the *Implementing a State Compensator* section. The difference is in the last entry of the system matrix **A**.

Figure 13-14 shows how the mismatched model, **State-Space Model**, is used to created the standalone state compensator with an estimator using the Compensator instance of the CD State-Space Controller VI. Then the actual system, **System**, and the mismatched model, **State-Space Model**, are connected in series so the actual system can provide the output y to the standalone compensator with an estimator.

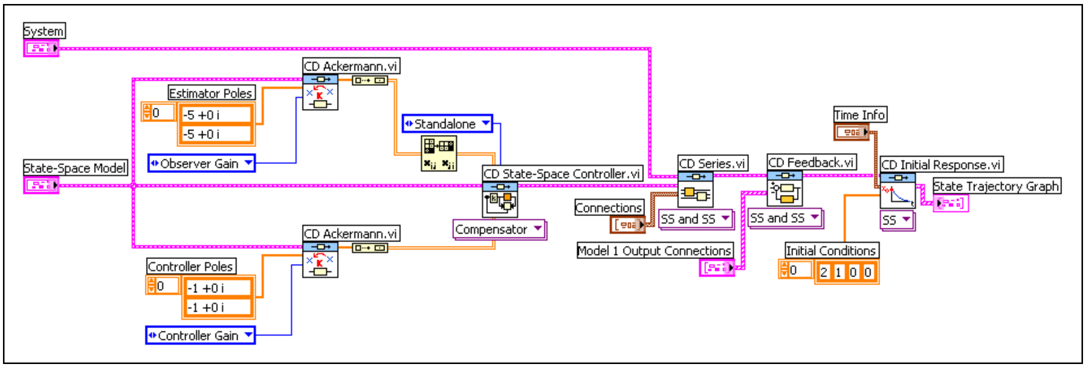


Figure 13-14. Standalone with Estimator Compensator

In this example, the system output *y* is supplied to the state compensator model through a series connection, while the input *u*, which the compensator calculates, is sent to the actual system using the feedback connection. The CD Initial Response VI uses the same initial conditions to test the effectiveness of the controller and estimator. Figure 13-15 shows the effect of a using a model that does not match the actual system.

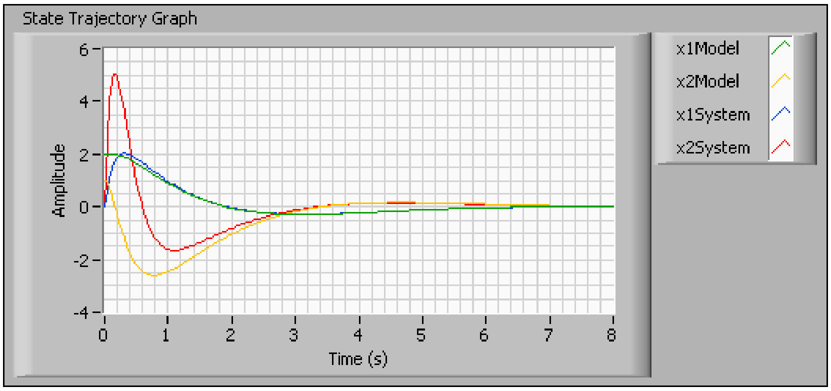


Figure 13-15. State Trajectory Standalone with Estimator Compensator

Even though Figure 13-13 and Figure 13-15 use the same system, the response to the same initial conditions is different because this example uses an estimator and controller based on an inaccurate model of the actual system. The estimator takes approximately 4 seconds to finally track the actual states. Notice that the time the estimator takes to track the actual states is much longer in this example than in the example in the *Creating a System Included Compensator* section. When using an accurate model, the estimator only took 1 to 1.5 seconds to track the actual states.

This example is similar to real-world applications where you do not know what the actual system is. Therefore, these tests are important to determine how sensitive the controller is to the system-model mismatches before the controller is deployed to an RT target. Using a design method called robust control design, you can create model-based controllers that take into account possible modeling errors. Refer to *Essentials of Robust Control*[1] for information about robust control design.

[1] Zhou, Kemin and John C. Doyle. *Essentials of Robust Control*, 1st ed. Upper Saddle River, NJ: Prentice Hall, 1998.

14 Advanced Equation Solvers

Many of the LabVIEW Control Design Toolkit design and analysis procedures rely on the solution of certain specialized forms of matrix equations. This chapter describes these equations and the techniques used in solving them.

Lyapunov and Sylvester Equations

The Lyapunov equation and the Sylvester equation are related and similar in form. Table 14-1 defines both the continuous and discrete versions of each equation.

Table 14-1. Lyapunov and Sylvester Equations

Equation	Continuous	Discrete
Lyapunov	$\boldsymbol{AX} + \boldsymbol{XA}^T + \boldsymbol{Q} = 0$	$\boldsymbol{AXA}^T - \boldsymbol{X} + \boldsymbol{Q} = 0$
Sylvester	$\boldsymbol{AX} + \boldsymbol{XB} + \boldsymbol{C} = 0$	$\boldsymbol{AXB} - \boldsymbol{X} + \boldsymbol{C} = 0$

The differences in these equations are in the underlying assumptions and meaning attributed to the coefficient matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$. These differences ensure certain properties of the solution $\boldsymbol{X}$.

This set of equations is used in analyzing the stability of systems and related applications such as the computation of Grammians, norms, and so on. Refer to *Numerical Linear Algebra Techniques for Systems and Control*[1] for more information about these computations. Refer to *Robust Pole Assignment via Sylvester Equation Based State Feedback Parametrization*[2] for more information about the Sylvester equation.

[1] Laub, Alan J., Paul Van Dooren, and R.V. Patel (eds). *Numerical Linear Algebra Techniques for Systems and Control*. Piscataway, NJ: IEEE Press, 1994.

[2] Varga, A. *Robust Pole Assignment via Sylvester Equation Based State Feedback Parametrization*. IEEE International Symposium on Computer Aided Control System Design, CACSD'2000. Anchorage, Alaska, 2000.

The Sylvester equation is the basis for solving both forms of these equations. The solution is based on the Bartels-Stewart algorithm, which reduces both $\boldsymbol{A}$ and $\boldsymbol{F}$ to the real Schur form and then performs a backward substitution to complete the solution. The underlying implementation of this algorithm uses the Mathematics VIs, which are based on BLAS/LAPACK algorithms.

The following steps summarize the procedure required to solve the continuous version of the Lyapunov equation.

1. Calculate the Schur decompositions of $\boldsymbol{A}$ and $\boldsymbol{B}$.

$$\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U} = \boldsymbol{R} \text{ and } \boldsymbol{V}^T\boldsymbol{B}\boldsymbol{T}\boldsymbol{V} = \boldsymbol{S}$$

2. Construct $\boldsymbol{F} = \boldsymbol{U}^T\boldsymbol{C}\boldsymbol{V}$.
3. Solve $\boldsymbol{R}\boldsymbol{Y} + \boldsymbol{Y}\boldsymbol{S}^T = \boldsymbol{F}$ using the backward-substitution procedure.
4. Solve $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{Y}\boldsymbol{V}^T$.

You can apply the same methodology to the discrete version of the Lyapunov equation if you complete the following steps.

1. Consider the continuous Lyapunov equation, $\boldsymbol{A}_1^T\boldsymbol{X} + \boldsymbol{X}\boldsymbol{A}_1 + \boldsymbol{C}_1 = 0$.
2. If $(\boldsymbol{I} - \boldsymbol{A}_1)^{-1}$ exists and if $\boldsymbol{A} = (\boldsymbol{I} - \boldsymbol{A}_1)^{-1}(\boldsymbol{I} + \boldsymbol{A}_1)$ and $\boldsymbol{A}_1 = (\boldsymbol{A} - \boldsymbol{I})(\boldsymbol{A} + \boldsymbol{I})^{-1}$, the Lyapunov equation is as follows:

$$(\boldsymbol{A} + \boldsymbol{I})^{-T}(\boldsymbol{A}^T - \boldsymbol{I})\boldsymbol{X} + \boldsymbol{X}(\boldsymbol{A} - \boldsymbol{I})(\boldsymbol{A} + \boldsymbol{I})^{-1} + \boldsymbol{C}_1 = 0$$

3. If you define the matrix $\boldsymbol{C}$ with the following equation:

$$\boldsymbol{C} = \frac{1}{2}(\boldsymbol{A} + \boldsymbol{I})^T\boldsymbol{C}_1(\boldsymbol{A} + \boldsymbol{I})$$

4. If you rearrange terms in the equation $\boldsymbol{C}$, you get the following discrete equation.

$$\boldsymbol{A}^T\boldsymbol{X}\boldsymbol{A} - \boldsymbol{X} + \boldsymbol{C} = 0$$

Riccati Equations

The symmetric $n \times n$ algebraic Riccati equation for the continuous time case is $\boldsymbol{A}^T\boldsymbol{X} + \boldsymbol{XA} - \boldsymbol{XMX} + \boldsymbol{Q} = 0$.

The solution of this equation is based on the reduction of a Hamiltonian matrix, defined by the following equation.

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A}^T & -\boldsymbol{M} \\ -\boldsymbol{Q} & -\boldsymbol{A} \end{bmatrix}$$

If $\boldsymbol{V}$ is the modal matrix of this Hamiltonian matrix, for stable eigenvalues, you can express $\boldsymbol{V}$ with the following equation.

$$\boldsymbol{V} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}$$

You then can express the steady state solution of the Riccati equation with the following equation.

$$\boldsymbol{X} = V_{22}V_{12}^{-1}$$

You can reduce a continuous algebraic Riccati equation, $\boldsymbol{A}^T\boldsymbol{X} + \boldsymbol{XA} - (\boldsymbol{XB} + \boldsymbol{S})\boldsymbol{R}^{-1}(\boldsymbol{B}^T\boldsymbol{X} + \boldsymbol{S}^T) + \boldsymbol{Q} = 0$, to the simpler form considered earlier by applying the following transformations.

$$\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{BR}^{-1}\boldsymbol{S}^T$$

$$\boldsymbol{M} = \boldsymbol{BR}^{-1}\boldsymbol{B}^T$$

$$\boldsymbol{Q} = \boldsymbol{Q} - \boldsymbol{SR}^{-1}\boldsymbol{S}^T$$

The discrete version of the Riccati equation is defined by the following equation.

$$\boldsymbol{A}^T\boldsymbol{XA} - \boldsymbol{X} - \boldsymbol{A}^T\boldsymbol{X}B_1(\boldsymbol{B}_2 + \boldsymbol{B}_1^T\boldsymbol{XB}_1)^{-1}\boldsymbol{B}_1^T\boldsymbol{XA} + C = 0$$

If $\boldsymbol{B} = \boldsymbol{B}_1\boldsymbol{B}_2^{-1}\boldsymbol{B}_1^T$, the Hamiltonian matrix for this equation is defined by the following equation.

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} + \boldsymbol{B}A^{-T}\boldsymbol{C} & -\boldsymbol{B}A^{-T} \\ -A^{-T}\boldsymbol{C} & A^{-T} \end{bmatrix}$$

You can use this Hamiltonian matrix in a reduction procedure to compute the solution of the discrete algebraic Ricatti equation.

If the discrete Ricatti equation is defined as follows, you can reduce the equation using the Hamiltonian matrix.

$$(\boldsymbol{A} - \boldsymbol{BR}^{-1}\boldsymbol{S}^T)\boldsymbol{X}(\boldsymbol{A} - \boldsymbol{BR}^{-1}\boldsymbol{S}^T) - (\boldsymbol{A} - \boldsymbol{BR}^{-1}\boldsymbol{S}^T)^T\boldsymbol{XB}(\boldsymbol{B}^T\boldsymbol{XB} + \boldsymbol{R})^{-1}\boldsymbol{B}^T\boldsymbol{X}(\boldsymbol{A} - \boldsymbol{BR}^{-1}\boldsymbol{S}^T) + \boldsymbol{Q} - \boldsymbol{SR}^{-1}\boldsymbol{S}^T = \boldsymbol{X}$$

The following transformations reduce the discrete Ricatti equation to the simpler form based on the determined Hamiltonian matrix.

$$\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{BR}^{-1}\boldsymbol{S}^T$$

$$\boldsymbol{B}_1 = \boldsymbol{B};\ \boldsymbol{B}_2 = \boldsymbol{R}$$

$$\boldsymbol{C} = \boldsymbol{Q}\ \boldsymbol{SR}^{-1}\boldsymbol{S}^T$$

A Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Online technical support resources at `ni.com/support` include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at `ni.com/exchange`. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, go to `ni.com/services` or contact your local branch at `ni.com/contact`.
- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.