LabVIEW™

PID and Fuzzy Logic Toolkit User Manual

June 2009
372192D-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

This manual describes the LabVIEW PID and Fuzzy Logic Toolkit. The PID and Fuzzy Logic Toolkit includes VIs for Proportional-Integral-Derivative (PID) and fuzzy logic control. You can use these VIs with input/output (I/O) functions such as data acquisition (DAQ) to implement control of physical processes.

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, controls and buttons on the front panel, dialog boxes, sections of dialog boxes, menu names, and palette names.
italic	Italic text denotes variables, linguistic terms, emphasis, a cross-reference, or an introduction to a key concept. Italic text 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, filenames, and extensions.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

Related Documentation

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

- *LabVIEW Help*, availably by launching LabVIEW and selecting **Help»Search the LabVIEW Help**.
- LabVIEW FPGA Module documentation.
- LabVIEW Real-Time Module documentation.
- LabVIEW Control Design and Simulation Module documentation.

Note The following resources offer useful background information on the general concepts discussed in this documentation. These resources are provided for general informational purposes only and are not affiliated, sponsored, or endorsed by National Instruments. The content of these resources is not a representation of, may not correspond to, and does not imply current or future functionality in the PID and Fuzzy Logic Toolkit or any other National Instruments product.

Aström, K. J. and T. Hagglund. 1984. Automatic tuning of simple regulators. In *Proceedings of IFAC 9th World Congress*, Budapest: 1867–72.

Aström, K. J., T. Hagglund, C. C. Hang, and W. K. Ho. 1993. Automatic tuning and adaptation for PID controllers: a survey. *Control Engineering Practice* 1, no. 4:699–714.

Corripio, A. B. 2000. *Tuning of Industrial Control Systems*. 2d ed. Raleigh, North Carolina: ISA.

Shinskey, F. G. 1988. *Process Control Systems: Application, Design, and Tuning*. 3d ed. Texas: McGraw-Hill.

Yen, J., R. Langari, and L. A. Zadeh, eds. 1995. *Industrial Applications of Fuzzy Logic and Intelligent Systems*. Piscataway, NJ: IEEE Press.

Ziegler, J. G., and N. B. Nichols. 1942. Optimum settings for automatic controllers. *Trans. ASME* 64:759–68.

Zimmerman, H.-J. 2001. *Fuzzy Set Theory – and Its Applications*. 4th ed. Norwell, MA: Kluwer Academic Publishers.

—. 1987. *Fuzzy Sets, Decision Making, and Expert Systems*. Boston: Kluwer Academic Publishers.

1

Overview of the PID and Fuzzy Logic Toolkit

You can use the LabVIEW PID and Fuzzy Logic Toolkit to perform both PID control and fuzzy logic control.

PID Control

Currently, the Proportional-Integral-Derivative (PID) algorithm is the most common control algorithm used in industry. Often, people use PID to control processes that include heating and cooling systems, fluid level monitoring, flow control, and pressure control. In PID control, you must specify a process variable and a setpoint. The process variable is the system parameter you want to control, such as temperature, pressure, or flow rate, and the setpoint is the desired value for the parameter you are controlling. A PID controller determines a controller output value, such as the heater power or valve position. The controller applies the controller output value to the system, which in turn drives the process variable toward the setpoint value.

You can use the PID VIs with National Instruments hardware to develop LabVIEW control applications. Use I/O hardware, like a DAQ device, FieldPoint I/O modules, or a GPIB board, to connect your PC to the system you want to control. You can use the I/O VIs provided in LabVIEW with the PID and Fuzzy Logic Toolkit to develop a control application or modify the examples provided with the toolkit.

Use the PID VIs to develop the following control applications based on PID controllers:

- Proportional (P); proportional-integral (PI); proportional-derivative (PD); and proportional-integral-derivative (PID) algorithms
- Gain-scheduled PID
- PID autotuning
- Error-squared PID
- Lead-Lag compensation

- Setpoint profile generation
- Multi-loop cascade control
- Feedforward control
- Override (minimum/maximum selector) control
- Ratio/bias control

You can combine these PID VIs with LabVIEW math and logic functions to create block diagrams for real control strategies.

Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about the VIs.

Fuzzy Logic

Fuzzy logic is a method of rule-based decision making used for expert systems and process control. Fuzzy logic differs from traditional Boolean logic in that fuzzy logic allows for partial membership in a set. You can use fuzzy logic to control processes represented by subjective, linguistic descriptions. Refer to Chapter 4, *Overview of Fuzzy Logic*, for more information about fuzzy logic.

Use the Fuzzy System Designer, available by selecting **Tools»Control Design and Simulation»Fuzzy System Designer**, to design a fuzzy system interactively. Refer to Chapter 9, *Designing a Fuzzy System with the Fuzzy System Designer*, for more information about the Fuzzy System Designer.

Use the Fuzzy Logic VIs to design and control fuzzy systems programmatically. Refer to Chapter 10, *Modifying a Fuzzy System with the Fuzzy Logic VIs*, for more information about the Fuzzy Logic VIs.

To implement real-time decision making or control of a physical system, you can wire acquired data to a fuzzy controller. You also can use outputs of the fuzzy controller with DAQ analog output hardware to implement real-time process control.

2

PID Algorithms

This chapter explains the PID, advanced PID, and autotuning algorithms.

The PID Algorithm

The PID controller compares the setpoint (SP) to the process variable (PV) to obtain the error (e).

$$e = SP - PV$$

Then the PID controller calculates the controller action, $u(t)$, where K_c is controller gain.

$$u(t) = K_c\left(e + \frac{1}{T_i}\int_0^t e\,dt + T_d\frac{de}{dt}\right)$$

If the error and the controller output have the same range, –100% to 100%, controller gain is the reciprocal of proportional band. T_i is the integral time in minutes, also called the reset time, and T_d is the derivative time in minutes, also called the rate time. The following formula represents the proportional action.

$$u_p(t) = K_c e$$

The following formula represents the integral action.

$$u_I(t) = \frac{K_c}{T_i}\int_0^t e\,dt$$

The following formula represents the derivative action.

$$u_D(t) = K_c T_d\frac{de}{dt}$$

Implementing the PID Algorithm with the PID VIs

This section describes how the PID VIs implement the positional PID algorithm. The subVIs used in these VIs are labelled so you can modify any of these features as necessary.

Error Calculation

The following formula represents the current error used in calculating proportional, integral, and derivative action.

$$e(k) = (SP - PV_f)$$

Proportional Action

Proportional action is the controller gain times the error, as shown in the following formula.

$$u_P(k)=(K_c * e(k))$$

Trapezoidal Integration

Trapezoidal integration is used to avoid sharp changes in integral action when there is a sudden change in *PV* or *SP*. Use nonlinear adjustment of integral action to counteract overshoot. The larger the error, the smaller the integral action, as shown in the following formula.

$$u_I(k)=\frac{K_c}{T_i}\sum_{i=1}^{k}\left[\frac{e(i)+e(i-1)}{2}\right]\Delta t$$

Partial Derivative Action

Because of abrupt changes in *SP*, only apply derivative action to the PV, not to the error *e*, to avoid derivative kick. The following formula represents the partial derivative action.

$$u_D(k) = -K_c \frac{T_d}{\Delta_t}(PV_f(k) - PV_f(k-1))$$

Controller Output

Controller output is the summation of the proportional, integral, and derivative action, as shown in the following formula.

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$

Output Limiting

The actual controller output is limited to the range specified for control output.

$$\text{If } u(k) \geq u_{max} \text{ then } u(k) = u_{max}$$

and

$$\text{if } u(k) \leq u_{min} \text{ then } u(k) = u_{min}$$

The following formula shows the practical model of the PID controller.

$$u(t) = K_c\left[(SP - PV) + \frac{1}{T_i}\int_0^t (SP - PV)dt - T_d\frac{dPV_f}{dt}\right]$$

The PID VIs use an integral sum correction algorithm that facilitates anti-windup and bumpless manual to automatic transfers. Windup occurs at the upper limit of the controller output, for example, 100%. When the error *e* decreases, the controller output decreases, moving out of the windup area. The integral sum correction algorithm prevents abrupt controller output changes when you switch from manual to automatic mode or change any other parameters.

The default ranges for the parameters **SP**, **PV**, and **output** correspond to percentage values; however, you can use actual engineering units. Adjust corresponding ranges accordingly. The parameters T_i and T_d are specified in minutes. In the manual mode, you can change the manual input to increase or decrease the output.

You can call these PID VIs from inside a While Loop with a fixed **cycle time**. All the PID Control VIs are reentrant. Multiple calls from high-level VIs use separate and distinct data.

Note As a general rule, manually drive the process variable until it meets or comes close to the setpoint before you perform the manual to automatic transfer.

Gain Scheduling

Gain scheduling refers to a system where you change controller parameters based on measured operating conditions. For example, the scheduling variable can be the setpoint, the process variable, a controller output, or an external signal. For historical reasons, the term *gain scheduling* is used even if other parameters such as **derivative time** or **integral time** change. Gain scheduling effectively controls a system whose dynamics change with the operating conditions.

With the PID Control VIs, you can define unlimited sets of PID parameters for gain scheduling. For each schedule, you can run autotuning to update the PID parameters.

The Advanced PID Algorithm

Error Calculation

The following formula represents the current error used in calculating proportional, integral, and derivative action.

$$e(k) = (SP - PV_f)(L+(1-L)*\frac{|SP - PV_f|}{SP_{range}})$$

The error for calculating proportional action is shown in the following formula.

$$eb(k) = (\beta*SP - PV_f)(L+(1-L)*\frac{|\beta SP - PV_f|}{SP_{range}})$$

where SP_{range} is the range of the setpoint, β is the setpoint factor for the two degree-of-freedom PID algorithm described in the *Proportional Action* section of this chapter, and L is the linearity factor that produces a nonlinear gain term in which the controller gain increases with the magnitude of the error. If L is 1, the controller is linear. A value of 0.1 makes the minimum gain of the controller 10% K_c. Use of a nonlinear gain term is referred to as an error-squared PID algorithm.

Proportional Action

In applications, SP changes are usually larger and faster than load disturbances, while load disturbances appear as a slow departure of the controlled variable from the SP. PID tuning for good load-disturbance responses often results in SP responses with unacceptable oscillation. However, tuning for good SP responses often yields sluggish load-disturbance responses. The factor β, when set to less than one, reduces the SP-response overshoot without affecting the load-disturbance response, indicating the use of a two degree-of-freedom PID algorithm. Intuitively, β is an index of the SP response importance, from zero to one. For example, if you consider load response the most important loop performance, set β to 0.0. Conversely, if you want the process variable to quickly follow the SP change, set β to 1.0.

$$u_P(k)=(K_c*eb(k))$$

Trapezoidal Integration

Trapezoidal integration is used to avoid sharp changes in integral action when there is a sudden change in *PV* or *SP*. Use nonlinear adjustment of integral action to counteract overshoot. The larger the error, the smaller the integral action, as shown in the following formula and in Figure 2-1.

$$u_I(k) = \frac{K_c}{T_i} \sum_{i=1}^{k} \left[\frac{e(i) + e(i-1)}{2} \right] \Delta t \left[\frac{1}{1 + \frac{10 * e(i)^2}{SP_{rng}^{\ 2}}} \right]$$

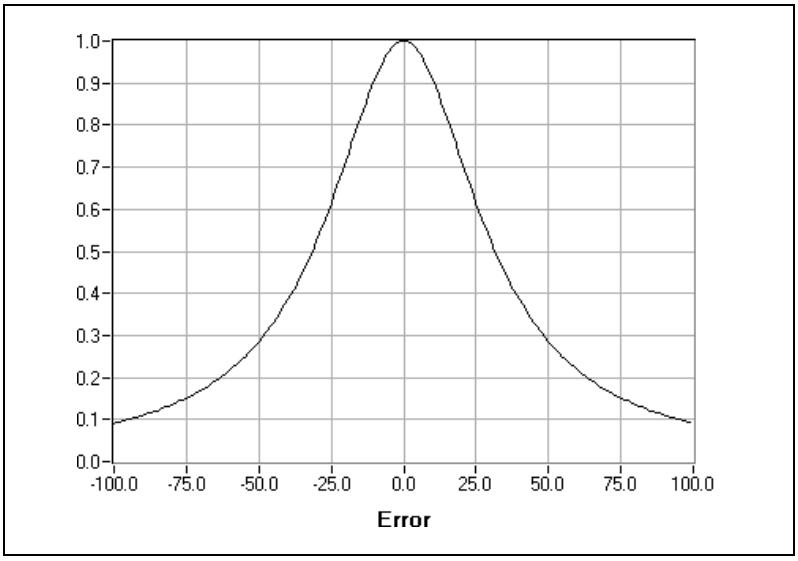

Figure 2-1. Nonlinear Multiple for Integral Action (SP_{rng} = 100)

The Autotuning Algorithm

Use autotuning to improve performance. Often, many controllers are poorly tuned. As a result, some controllers are too aggressive and some controllers are too sluggish. PID controllers are difficult to tune when you do not know the process dynamics or disturbances. In this case, use autotuning. Before you begin autotuning, you must establish a stable controller, even if you cannot properly tune the controller on your own.

Figure 2-2 illustrates the autotuning procedure excited by the setpoint relay experiment, which connects a relay and an extra feedback signal with the setpoint. Notice that the PID Autotuning VI directly implements this process. The existing controller remains in the loop.

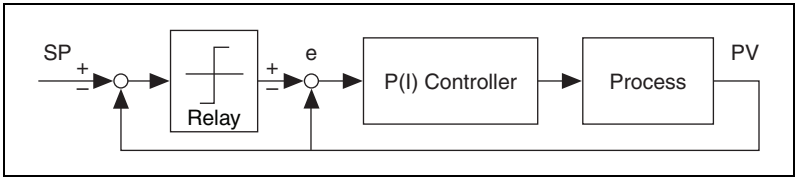

Figure 2-2. Process under PID Control with Setpoint Relay

For most systems, the nonlinear relay characteristic generates a limiting cycle, from which the autotuning algorithm identifies the relevant information needed for PID tuning. If the existing controller is proportional only, the autotuning algorithm identifies the ultimate gain ***Ku*** and ultimate period ***Tu***. If the existing model is PI or PID, the autotuning algorithm identifies the dead time τ and time constant ***Tp***, which are two parameters in the integral-plus-deadtime model.

$$G_P(s) = \frac{e^{-\tau s}}{T_p s}$$

Tuning Formulas

The LabVIEW PID and Fuzzy Logic Toolkit uses Ziegler and Nichols' heuristic methods for determining the parameters of a PID controller. When you autotune, select one of the following three types of loop performance: fast (1/4 damping ratio), normal (some overshoot), and slow (little overshoot). Refer to the following tuning formula tables for each type of loop performance.

Table 2-1. Tuning Formula under P-Only Control (Fast)

Controller	K_c	T_i	T_d
P	$0.5K_u$	—	—
PI	$0.4K_u$	$0.8T_u$	—
PID	$0.6K_u$	$0.5T_u$	$0.12T_u$

Table 2-2. Tuning Formula under P-Only Control (Normal)

Controller	K_c	T_i	T_d
P	$0.2K_u$	—	—
PI	$0.18K_u$	$0.8T_u$	—
PID	$0.25K_u$	$0.5T_u$	$0.12T_u$

Table 2-3. Tuning Formula under P-Only Control (Slow)

Controller	K_c	T_i	T_d
P	$0.13K_u$	—	—
PI	$0.13K_u$	$0.8T_u$	—
PID	$0.15K_u$	$0.5T_u$	$0.12T_u$

Table 2-4. Tuning Formula under PI or PID Control (Fast)

Controller	K_c	T_i	T_d
P	T_p/τ	—	—
PI	$0.9T_p/\tau$	3.33τ	—
PID	$1.1T_p/\tau$	2.0τ	0.5τ

Table 2-5. Tuning Formula under PI or PID Control (Normal)

Controller	K_c	T_i	T_d
P	$0.44T_p/\tau$	—	—
PI	$0.4T_p/\tau$	5.33τ	—
PID	$0.53T_p/\tau$	4.0τ	0.8τ

Table 2-6. Tuning Formula under PI or PID Control (Slow)

Controller	K_c	T_i	T_d
P	$0.26T_p/\tau$	—	—
PI	$0.24T_p/\tau$	5.33τ	—
PID	$0.32T_p/\tau$	4.0τ	0.8τ

Note During tuning, the process remains under closed-loop PID control. You do not need to switch off the existing controller and perform the experiment under open-loop conditions. In the setpoint relay experiment, the SP signal mirrors the SP for the PID controller.

3

Using the PID Software

This chapter contains the basic information you need to begin using the PID Control VIs.

Designing a Control Strategy

When you design a control strategy, sketch a flowchart that includes the physical process and control elements such as valves and measurements. Add feedback from the process and any required computations. Then use the VIs in this toolkit, combined with the math and logic VIs and functions in LabVIEW, to translate the flowchart into a block diagram. Figure 3-1 is an example of a control flowchart, and Figure 3-2 is the equivalent LabVIEW block diagram. The only elements missing from this simplified VI are the loop-tuning parameters and the automatic-to-manual switching.

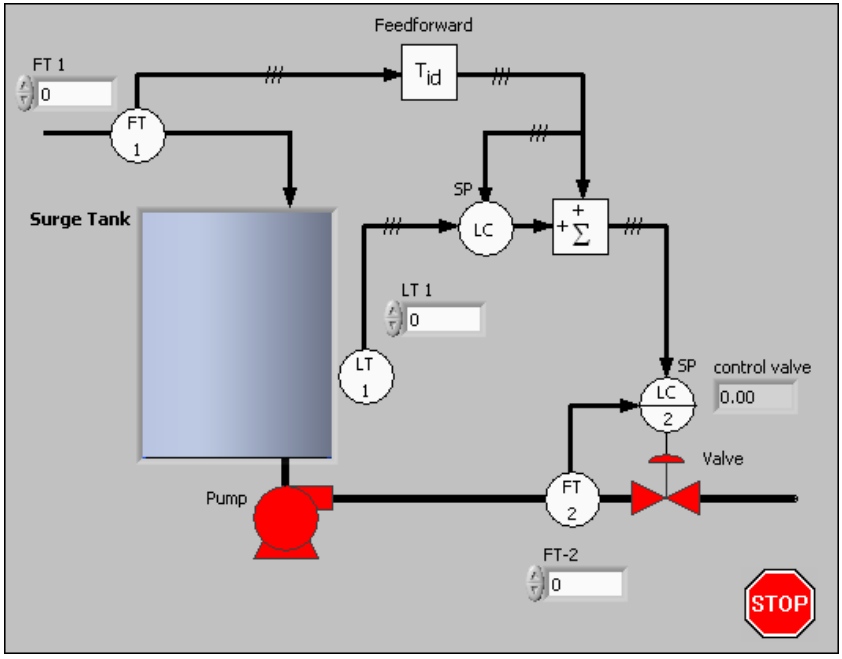

Figure 3-1. Block Diagram Representation of the PID Control Flowchart. FT = Flow Transmitter. LT = Level Transmitter. LC = Level Control. SP = Setpoint.

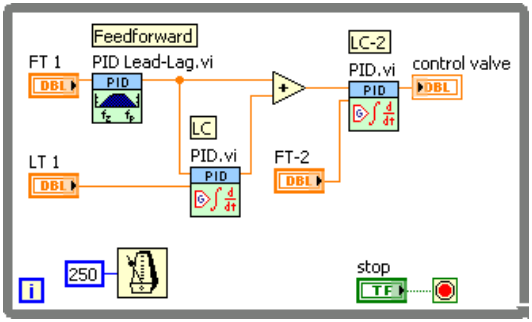

Figure 3-2. LabVIEW Block Diagram

You can handle the inputs and outputs through DAQ devices, FieldPoint I/O modules, GPIB instruments, or serial I/O ports. You can adjust polling rates in real time. Potential polling rates are limited only by your hardware and by the number and graphical complexity of your VIs.

Setting Timing

The PID VI and the PID Lead-Lag VI in the LabVIEW PID and Fuzzy Logic Toolkit are time dependent. A VI can acquire timing information either from a value you supply to the cycle time control, **dt**, or from a time keeper such as those built into the PID VIs. If **dt** is less than or equal to zero, the VI calculates new timing information each time LabVIEW calls it. At each call, the VI measures the time since the last call and uses that difference in its calculations. If you call a VI from a While Loop that uses one of the LabVIEW timing VIs, located on the **Time & Dialog** palette, you can achieve fairly regular timing, and the internal time keeper compensates for variations. However, the resolution of the Tick Count (ms) function is limited to 1 ms.

If **dt** is a positive value in seconds, the VI uses that value in the calculations, regardless of the elapsed time. National Instruments recommends you use this method for fast loops, such as when you use acquisition hardware to time the controller input or real-time applications. Refer to the example VIs located in the `labview\examples\control\pid\prctrlex.llb` directory for examples of using timing with the PID VIs. If you installed NI-DAQmx, you also can view relevant examples in the `labview\examples\DAQmx\Control\Control.llb` directory.

According to control theory, a control system must sample a physical process at a rate about 10 times faster than the fastest time constant in the physical process. For example, a time constant of 60 s is typical for a temperature control loop in a small system. In this case, a cycle time of

about 6 s is sufficient. Faster cycling offers no improvement in performance (Corripio 2000). In fact, running all your control VIs too fast degrades the response time of your LabVIEW application.

All VIs within a loop execute once per iteration at the same cycle time. To run several control VIs at different cycle times and still share data between them, as for example in a cascade, you must separate the VIs into independently timed While Loops. Figure 3-3 shows an example of a cascade with two independently timed While Loops.

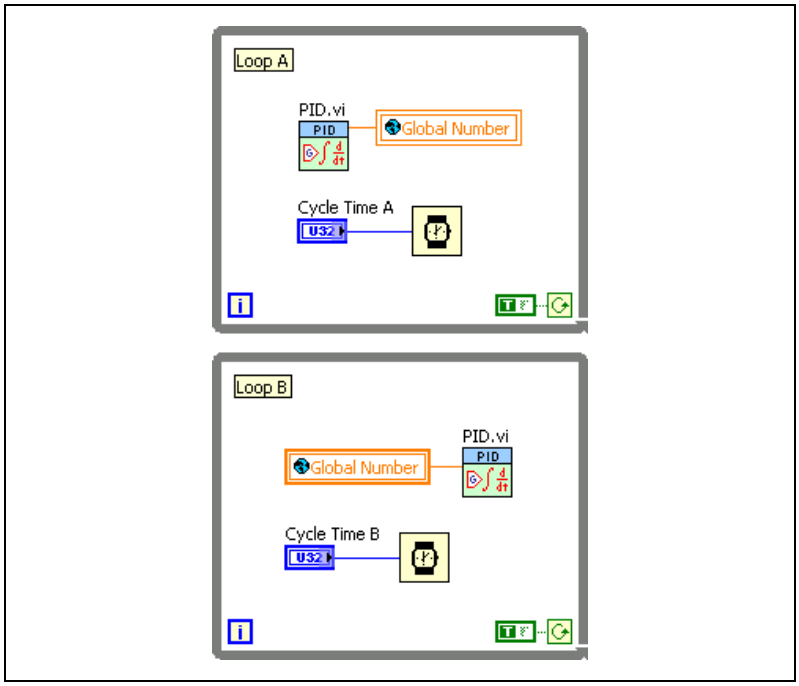

Figure 3-3. Cascaded Control Functions

A global variable passes the output of Loop A to the **PV** input of Loop B. You can place both While Loops on the same diagram. In this case, they are in separate VIs. Use additional global or local variables to pass any other necessary data between the two While Loops.

If the front panel does not contain graphics that LabVIEW must update frequently, the PID Control VIs can execute at kilohertz (kHz) rates. Remember that actions such as mouse activity and window scrolling interfere with these rates.

Tuning Controllers Manually

The following controller tuning procedures are based on the work of Ziegler and Nichols, the developers of the quarter-decay ratio tuning techniques derived from a combination of theory and empirical observations (Corripio 2000). Experiment with these techniques and with one of the process control simulation VIs to compare them. For different processes, one method might be easier or more accurate than another. For example, some techniques that work best when used with online controllers cannot stand the large upsets described here.

To perform these tests with LabVIEW, set up your control strategy with the **PV**, **SP**, and **output** displayed on a large strip chart with the axes showing the values versus time. Refer to the *Closed-Loop (Ultimate Gain) Tuning Procedure* and *Open-Loop (Step Test) Tuning Procedure* sections of this chapter for more information about disturbing the loop and determining the response from the graph. Refer to Corripio (2000), as listed in the *Related Documentation* section of this manual, for more information about these procedures.

Closed-Loop (Ultimate Gain) Tuning Procedure

Although the closed-loop (ultimate gain) tuning procedure is very accurate, you must put your process in steady-state oscillation and observe the PV on a strip chart. Complete the following steps to perform the closed-loop tuning procedure.

1. Set both the **derivative time** and the **integral time** on your PID controller to `0`.
2. With the controller in automatic mode, carefully increase the **proportional gain** (K_c) in small increments. Make a small change in **SP** to disturb the loop after each increment. As you increase K_c, the value of **PV** should begin to oscillate. Keep making changes until the oscillation is sustained, neither growing nor decaying over time.
3. Record the controller **proportional band** (PB_u) as a percent, where $PB_u = 100/K_c$.
4. Record the period of oscillation (T_u) in minutes.

5. Multiply the measured values by the factors shown in Table 3-1 and enter the new tuning parameters into your controller. Table 3-1 provides the proper values for a quarter-decay ratio.

   If you want less overshoot, increase the gain K_c.

Table 3-1. Closed-Loop–Quarter-Decay Ratio Values

Controller	PB (percent)	Reset (minutes)	Rate (minutes)
P	$2.00PB_u$	—	—
PI	$2.22PB_u$	$0.83T_u$	—
PID	$1.67PB_u$	$0.50TT_u$	$0.125T_u$

Note Proportional gain (K_c) is related to proportional band (PB) as $K_c = 100/PB$.

Open-Loop (Step Test) Tuning Procedure

The open-loop (step test) tuning procedure assumes that you can model any process as a first-order lag and a pure deadtime. This method requires more analysis than the closed-loop tuning procedure, but your process does not need to reach sustained oscillation. Therefore, the open-loop tuning procedure might be quicker and more reliable for many processes. Observe the output and the **PV** on a strip chart that shows time on the x-axis. Complete the following steps to perform the open-loop tuning procedure.

1. Put the controller in manual mode, set the output to a nominal operating value, and allow the **PV** to settle completely. Record the **PV** and output values.
2. Make a step change in the output. Record the new output value.

3. Wait for the **PV** to settle. From the chart, determine the values as derived from the sample displayed in Figure 3-4.

   The variables represent the following values:

   - T_d—Deadtime in minutes
   - T—Time constant in minutes
   - K—Process gain = (change in PV) / (change in output)

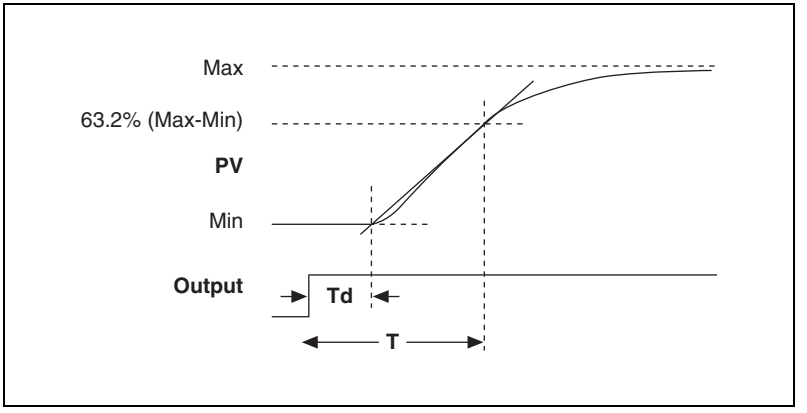

Figure 3-4. Output and Process Variable Strip Chart

4. Multiply the measured values by the factors shown in Table 3-2 and enter the new tuning parameters into your controller. The table provides the proper values for a quarter-decay ratio. If you want less overshoot, reduce the gain, K_c.

Table 3-2. Open-Loop–Quarter-Decay Ratio Values

Controller	PB (percent)	Reset (minutes)	Rate (minutes)
P	$100\frac{KT_d}{T}$	—	—
PI	$110\frac{KT_d}{T}$	$3.33T_d$	—
PID	$80\frac{KT_d}{T}$	$2.00T_d$	$0.50T_d$

Using the PID VIs

Although there are several variations of the PID VI, they all use the algorithms described in Chapter 2, *PID Algorithms*. The PID VI implements the basic PID algorithm. Other variations provide additional functionality as described in the following sections. You can use these VIs interchangeably because they all use consistent inputs and outputs where possible.

The PID VI

The PID VI has inputs for **setpoint**, **process variable**, **PID gains**, **dt**, **output range**, and **reinitialize?**. The **PID gains** input is a cluster of three values—proportional gain, integral time, and derivative time.

You can use **output range** to specify the range of the controller output. The default range of the controller output is –100 to 100, which corresponds to values specified in terms of percentage of full scale. However, you can change this range to one that is appropriate for your control system, so that the controller gain relates engineering units to engineering units instead of percentage to percentage. The PID VI coerces the controller output to the specified range. In addition, the PID VI implements integrator anti-windup when the controller output is saturated at the specified minimum or maximum values. Refer to Chapter 2, *PID Algorithms*, for more information about anti-windup.

You can use **dt** to specify the control-loop cycle time. The default value is –1, so by default the PID VI uses the operating system clock for calculations involving the loop cycle time. If the loop cycle time is deterministic, you can provide this input to the PID VI. Note that the operating system clock has a resolution of 1 ms, so specify a **dt** value explicitly if the loop cycle time is less than 1 ms.

The PID VI will initialize all internal states on the first call to the VI. All subsequent calls to the VI will make use of state information from previous calls. However, you can reinitialize the PID VI to its initial state at any time by passing a value of `TRUE` to the **reinitialize?** input. Use this function if your application must stop and restart the control loop without restarting the entire application.

Fault Protection

If the PID VI receives an invalid input, such as NaN (Not a Number), the VI outputs NaN until you pass a value of `TRUE` to the **reinitialize?** input. You can use the Not A Number/Path/Refnum? VI to check for invalid inputs and respond in a way that is appropriate to the application. For example, the following block diagram uses the Not A Number/Path/Refnum? VI with a Select function and a Feedback Node to reuse the last valid input when the VI receives an invalid input.

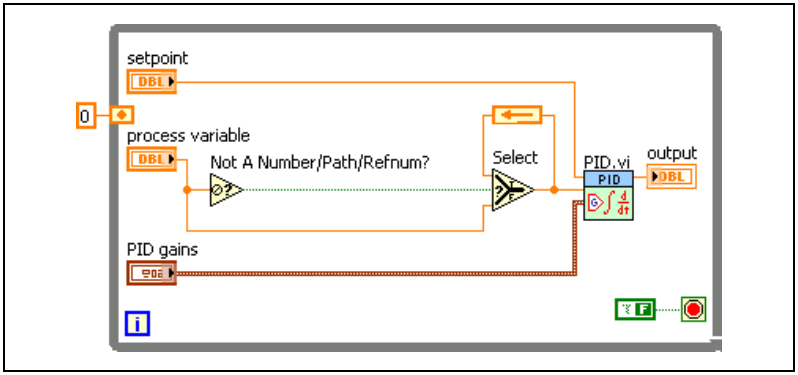

Figure 3-5. Fault Protection

Refer to the General PID with Fault Protection VI in the `labview\examples\control\pid\prctrlex.llb` directory for an example of a single-channel PID implementation that includes fault protection.

The PID Advanced VI

The PID Advanced VI has the same inputs as the PID VI, with the addition of inputs for **setpoint range**, **beta**, **linearity**, **auto?**, and **manual control**. You can specify the range of the setpoint using the **setpoint range** input, which also establishes the range for the process variable. The default setpoint range is 0 to 100, which corresponds to values specified in terms of percentage of full scale. However, you can change this range to one that is appropriate for your control system, so that the controller gain relates engineering units to engineering units instead of percentage to percentage. The PID Advanced VI uses the setpoint range in the nonlinear integral action calculation and, with the linearity input, in the nonlinear error calculation. The VI uses the **beta** input in the two degree-of-freedom algorithm, and the **linearity** input in the nonlinear gain factor calculation. Refer to Chapter 2, *PID Algorithms*, for more information about these calculations.

You can use the **auto?** and **manual control** inputs to switch between manual and automatic control modes. The default value of **auto?** is TRUE, which means the VI uses the PID algorithm to calculate the controller output. You can implement manual control by changing the value of **auto?** to FALSE so that the VI passes the value of **manual control** through to the output.

Bumpless Automatic-to-Manual Transfer

The PID Advanced VI cannot implement bumpless automatic-to-manual transfer. In order to ensure a smooth transition from automatic to manual control mode, you must design your application so that the manual output value matches the control output value at the time that the control mode is switched from automatic to manual. You can do this by using a local variable for the **manual control** control, as shown in Figure 3-6.

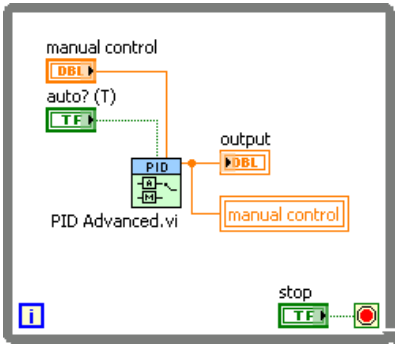

Figure 3-6. Bumpless Automatic-to-Manual Transfer

Although this VI does not support automatic-to-manual transfer, it does support bumpless manual-to-automatic transfer, which ensures a smooth controller output during the transition from manual to automatic control mode.

Multi-Loop PID Control

Most of the PID VIs are polymorphic VIs for use in multiple control-loop applications. For example, you can design a multi-loop PID control application using the PID VI and DAQ functions for input and output. A DAQ analog input function returns an array of data when you configure it for multiple channels. You can wire this array directly into the **process variable** input of the PID VI. The polymorphic type of the PID VI automatically switches from DBL to DBL Array, which calculates and

returns an array of **output** values corresponding to the number of values in the **process variable** array. Note that you also can switch the type of the polymorphic VI manually by right-clicking the VI icon and selecting **Select Type** from the shortcut menu.

When the polymorphic type is set to DBL Array, other inputs change automatically to array inputs as well. For example, the PID VI inputs **setpoint**, **PID gains**, and **output range** all become array inputs. Each of these inputs can have an array length ranging from 1 to the array length of the **process variable** input. If the array length of any of these inputs is less than the array length of the **process variable** input, the PID VI reuses the last value in the array for other calculations. For example, if you specify only one set of PID gains in the **PID gains** array, the PID VI uses these gains to calculate each **output** value corresponding to each **process variable** input value. Other polymorphic VIs included with the PID and Fuzzy Logic Toolkit operate in the same manner.

Setpoint Ramp Generation

The PID Setpoint Profile VI can generate a profile of setpoint values over time for a "ramp and soak" type PID application. For example, you might want to ramp the setpoint temperature of an oven control system over time, and then hold, or soak, the setpoint at a certain temperature for another period of time. You can use the PID Setpoint Profile VI to implement any arbitrary combination of ramp, hold, and step functions.

Specify the setpoint profile as an array of pairs of time and setpoint values with the time values in ascending order. For example, a ramp setpoint profile can be specified with two setpoint profile array values, as shown in Figure 3-7.

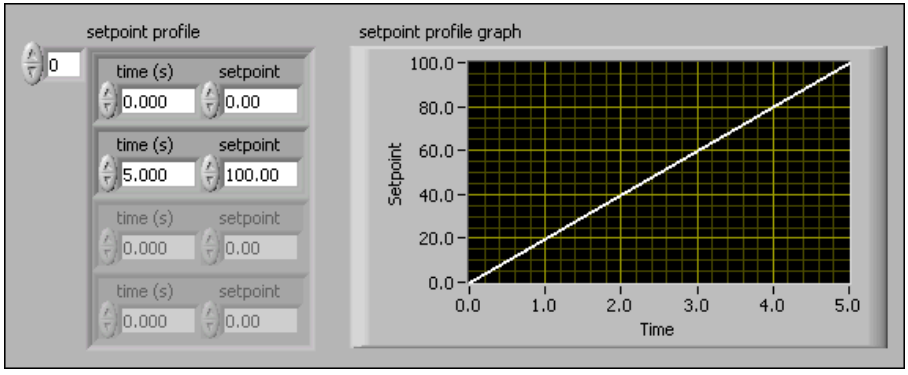

Figure 3-7. Ramp Setpoint Profile

A ramp and hold setpoint profile also can have two successive array values with the same setpoint value, as shown in Figure 3-8.

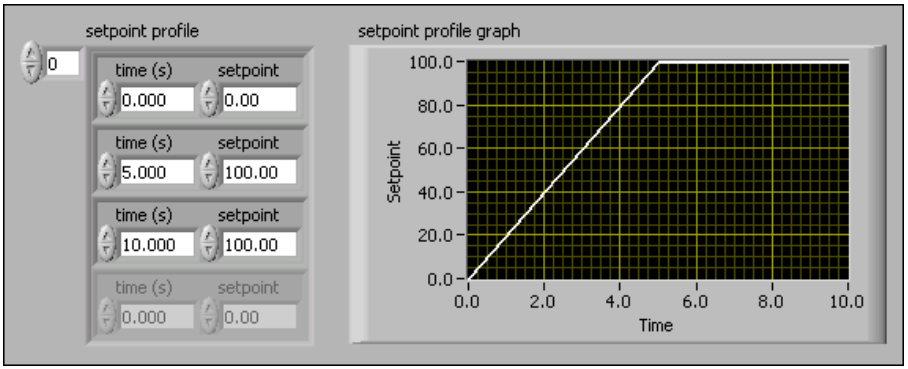

Figure 3-8. Ramp and Hold Setpoint Profile

Alternatively, a step setpoint profile can have two successive array values with the same time value but different setpoint values, as shown in Figure 3-9.

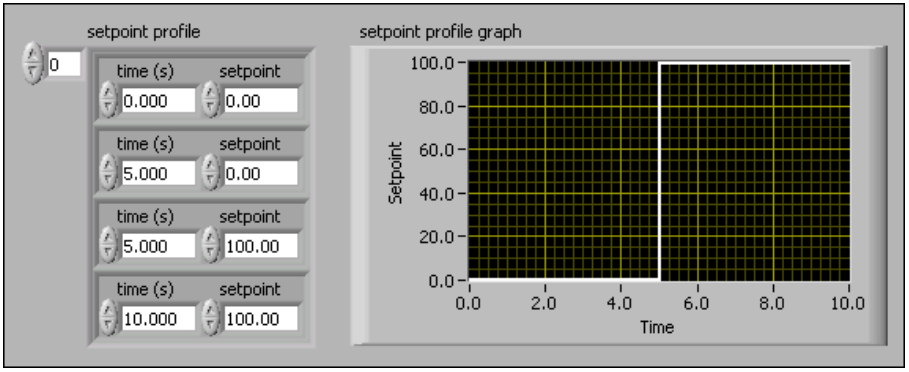

Figure 3-9. Step Setpoint Profile

The PID Setpoint Profile VI outputs a single **setpoint** value determined from the current elapsed time. Therefore, you should use this VI inside the control loop. The first call to the VI initializes the current time in the setpoint profile to 0. On subsequent calls, the VI, determines the current time from the previous time and the **dt** input value. If you reinitialize the current time to 0 by passing a value of `TRUE` to the **reinitialize?** input, you can repeat the specified setpoint profile.

If the loop cycle time is deterministic, you can use the input **dt** to specify its value. The default value of dt is –1, so by default the VI uses the operating system clock for calculations involving the loop cycle time. The operating system clock has a resolution of 1 ms, so specify a **dt** value explicitly if the loop cycle time is less than 1 ms.

Filtering Control Inputs

You can use the PID Control Input Filter VI to filter high-frequency noise from measured values in a control application, for example, if you are measuring **process variable** values using a DAQ device.

As discussed in the *Setting Timing* section of this chapter, the sampling rate of the control system should be at least 10 times faster than the fastest time constant of the physical system. Therefore, if correctly sampled, any frequency components of the measured signal greater than one-tenth of the sampling frequency are a result of noise in the measured signal. Gains in the PID controller can amplify this noise and produce unnecessary wear on actuators and other system components.

The PID Control Input Filter VI filters out unwanted noise from input signals. The algorithm it uses is a lowpass fifth-order finite impulse response (FIR) filter. The cutoff frequency of the lowpass filter is one-tenth of the sampling frequency, regardless of the actual sampling frequency value. You can use the PID Control Input Filter VI to filter noise from input values in the control loop before the values pass to control functions such as the PID VI.

Use the Filters PtByPt VIs or the LabVIEW Digital Filter Design Toolkit to perform additional filtering tasks. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `dfdt` for more information about the Digital Filter Design Toolkit.

Gain Scheduling

With the PID Gain Schedule VI, you can apply different sets of PID parameters for different regions of operation of your controller. Because most processes are nonlinear, PID parameters that produce a desired response at one operating point might not produce a satisfactory response at another operating point. The PID Gain Schedule VI selects and outputs one set of PID gains from a gain schedule based on the current value of the **gain scheduling value** input. For example, to implement a gain schedule based on the value of the process variable, wire the process variable value to the **gain scheduling value** input and wire the **PID gains out** output to the **PID gains** input of the PID VI.

The **PID gain schedule** input is an array of clusters of **PID gains** and corresponding **max values**. Each set of **PID gains** corresponds to the range of input values from the **max value** of the previous element of the array to the **max value** of the same element of the array. The input range of the **PID gains** of the first element of the **PID gain schedule** is all values less than or equal to the corresponding **max value**.

In Figure 3-10, the Gain Scheduling Input example uses the **setpoint** value as the **gain scheduling variable** with a default range of 0 to 100. Table 3-3 summarizes **PID parameters**.

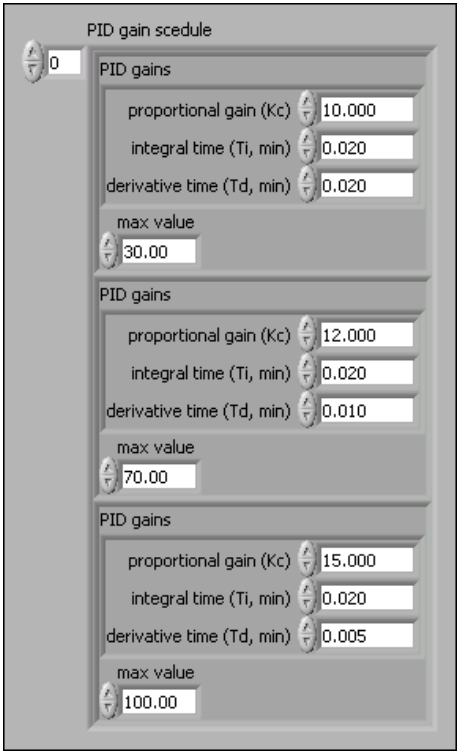

Figure 3-10. Gain Scheduling Input Example

Table 3-3. PID Parameter Ranges

Range	Parameters
$0 \leq SP \leq 30$	**Kc** = 10, **Ti** = 0.02, **Td** = 0.02

Table 3-3. PID Parameter Ranges (Continued)

Range	Parameters
$30 < SP \leq 70$	**Kc** = 12, **Ti** = 0.02, **Td** = 0.01
$70 < SP \leq 100$	**Kc** = 15, **Ti** = 0.02, **Td** = 0.005

Control Output Rate Limiting

Sudden changes in control output are often undesirable or even dangerous for many control applications. For example, a sudden large change in setpoint can cause a very large change in controller output. Although in theory this large change in controller output results in fast response of the system, it may also cause unnecessary wear on actuators or sudden large power demands. In addition, the PID controller can amplify noise in the system and result in a constantly changing controller output.

You can use the PID Output Rate Limiter VI to avoid the problem of sudden changes in controller output. Wire the **output** value from the PID VI to the **input (controller output)** input of the PID Output Rate Limiter VI. This limits the slew, or rate of change, of the output to the value of the **output rate (EGU/min)**.

Assign a value to **initial output** to specify the **output** value on the first call to the VI. You can reinitialize the output to the initial value by passing a value of `TRUE` to the **reinitialize?** input.

You can use **dt** to specify the control-loop cycle time. The default value is –1, so that by default the VI uses the operating system clock for calculations involving the loop cycle time. If the loop cycle time is deterministic, you can provide this input to the PID Output Rate Limiter VI. Note that the operating system clock has a resolution of 1 ms; therefore, you should specify a **dt** value explicitly if the loop cycle time is less than 1 ms.

The PID Lead-Lag VI

The PID Lead-Lag VI uses a positional algorithm that approximates a true exponential lead/lag. Feedforward control schemes often use this kind of algorithm as a dynamic compensator.

You can specify the range of the output using the **output range** input. The default range is –100 to 100, which corresponds to values specified in terms of percentage of full scale. However, you can change this range to one that is appropriate for your control system, so that the controller gain relates

engineering units to engineering units instead of percentage to percentage. The PID Lead-Lag VI coerces the controller output to the specified range.

The **output** value on the first call to the VI is the same as the **input** value. You can reinitialize the output to the current **input** value by passing a value of TRUE to the **reinitialize?** input.

You can use **dt** to specify the control-loop cycle time. The default value is –1, so that by default the VI uses the operating system clock for calculations involving the loop cycle time. If the loop cycle time is deterministic, you can provide this input to the PID Lead-Lag VI. Note that the operating system clock has a resolution of 1 ms; therefore you should specify **dt** explicitly if the loop cycle time is less than 1 ms.

Converting Between Percentage of Full Scale and Engineering Units

As described above, the default setpoint, process variable, and output ranges for the PID VIs correspond to percentage of full scale. In other words, **proportional gain** (K_c) relates percentage of full scale output to percentage of full scale input. This is the default behavior of many PID controllers used for process control applications. To implement PID in this way, you must scale all inputs to percentage of full scale and all controller outputs to actual engineering units, for example, volts for analog output.

You can use the PID EGU to Percentage VI to convert any input from real engineering units to percentage of full scale, and you can use the PID Percentage to EGU function to convert the controller output from percentage to real engineering units. The PID Percentage to EGU VI has an additional input, **coerce output to range?**. The default value of the **coerce output to range?** input is TRUE.

Note The PID VIs do not use the setpoint range and output range information to convert values to percentages in the PID algorithm. The controller gain relates the output in engineering units to the input in engineering units. For example, a gain value of 1 produces an output of 10 for a difference between setpoint and process variable of 10, regardless of the output range and setpoint range.

Using the PID Autotuning VI and the Autotuning Wizard

To use the Autotuning Wizard to improve your controller performance, you must first create your control application and determine PID parameters that produce stable control of the system. You can develop the control application using either the PID VI, the PID Gain Schedule VI, or the PID Autotuning VI. Because the PID Autotuning VI has input and output consistent with the other PID VIs, you can replace any PID VI with it.

The PID Autotuning VI has several additional input and output values to specify the autotuning procedure. The two additional input values are **autotuning parameters** and **autotune?**. The **autotuning parameters** input is a cluster of parameters that the VI uses for the autotuning process. Because the Autotuning Wizard allows you to specify all of these parameters manually, you can leave the **autotuning parameters** input unwired. The **autotune?** input takes a Boolean value supplied by a user control. Wire a Boolean control on the front panel of your application to this input. When the user presses the Boolean control, the Autotuning Wizard opens automatically. Set the Boolean control mechanical action to **Latch When Released** so that the Autotuning Wizard does not open repeatedly when the user presses the control. The Autotuning Wizard steps the user through the autotuning process. Refer to Chapter 2, *PID Algorithms*, for more information about the autotuning algorithm. The PID Autotuning VI also has two additional output values—**tuning completed?** and **PID gains out**. The **tuning completed?** output is a Boolean value. It is usually `FALSE` and becomes `TRUE` only on the iteration during which the autotuning finishes. The autotuning procedure updates the PID parameters in **PID gains out**. Normally, **PID gains out** passes through **PID gains** and outputs **PID gains out** only when the autotuning procedure completes. You have several ways to use these outputs in your applications.

Figure 3-11 shows one possible implementation of the PID Autotuning VI. The shift register on the left stores the initial value of the PID gains. **PID gains out** then passes to the right-hand shift register terminal when each control loop iteration completes. Although this method is simple, it suffers from one limitation. The user cannot change **PID gains** manually while the control loop is running.

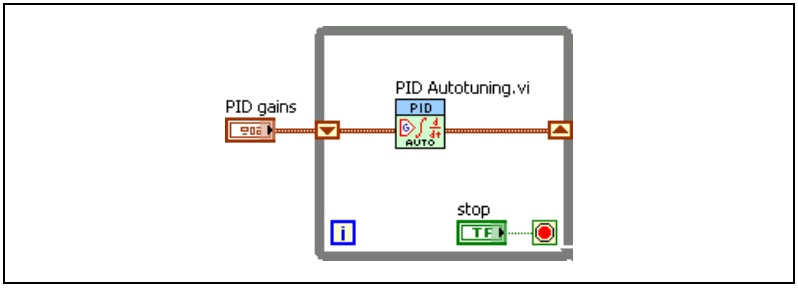

Figure 3-11. Updating PID Parameters Using a Shift Register

Figure 3-12 shows a second method, which uses a local variable to store the updated **PID gains**. In this example, the VI reads the **PID gains** control on each iteration, and a local variable updates the control only when **tuning complete?** is TRUE. This method allows for manual control of the **PID gains** while the control loop executes. In both examples, you must save **PID gains** so that you can use the **PID gains out** values for the next control application run. To do this, ensure that the **PID gains** control shows the current updated parameters, then choose **Make Current Values Default** from the **Operate** menu, and then save the VI.

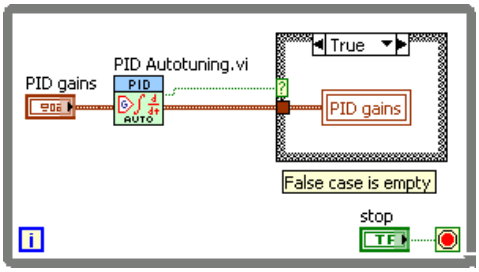

Figure 3-12. Updating PID Parameters Using a Local Variable

To avoid having to manually save the VI each time it runs, you can use a datalog file to save the **PID gains**, as shown in Figure 3-13.

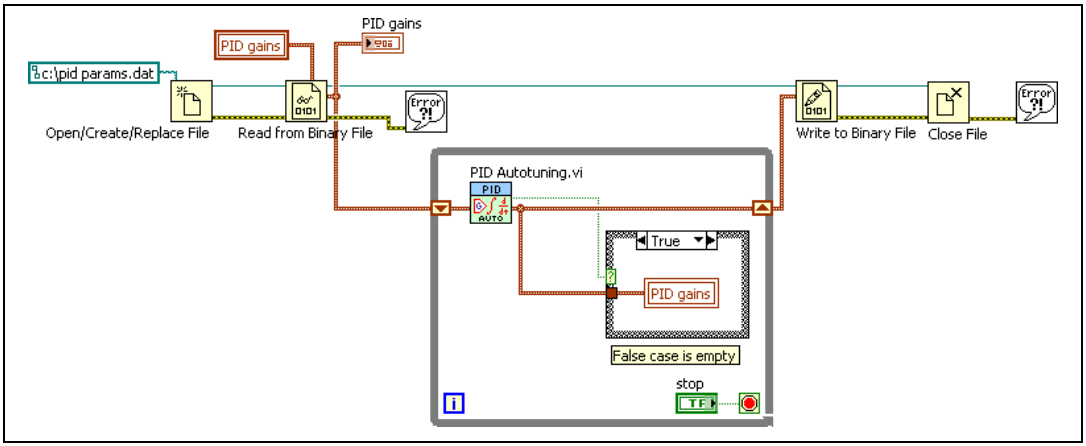

Figure 3-13. Storing PID Parameters in a Datalog File

Before the control loop begins, the File I/O VIs read a datalog file to obtain the **PID gains** parameters. When the autotuning procedure runs, a local variable updates the **PID gains** control. After the control loop is complete, the VI writes the current **PID gains** cluster to the datalog file and saves it. Each time it runs, the VI uses updated parameters.

Using PID on FPGA Targets

Use the PID (FPGA) Express VI to implement single-channel or multi-channel PID on a LabVIEW FPGA target.

Note The PID (FPGA) Express VI is available only if you install both the PID and Fuzzy Logic Toolkit and the LabVIEW FPGA Module.

The PID (FPGA) Express VI implements a fixed-point PID algorithm on FPGA targets. The PID algorithm features control output range and uses an integrator anti-windup calculation to limit the effect of the integral action during transients. The PID algorithm also features bumpless controller output for PID gain changes.

The PID (FPGA) Express VI represents the PID proportional, integral, and derivative gains as signed fixed-point numbers with word length 16 and integer word length 8. Given the proportional gain (K_c), integral time (T_i [minutes]), and derivative time (T_d [minutes]), respectively, this Express VI normalizes the gains according to the following formulas:

$$K_p = K_c$$

$$K_i = \frac{K_c \times T_s}{T_i \times 60}$$

$$K_d = \frac{K_c \times T_d \times 60}{T_s}$$

where T_s is the sampling period, in seconds, at which the PID loop runs.

The PID controller compares the setpoint *SP* to the process variable *PV* at a given sample instant *k* and produces the error *e*, which is defined as the following:

$$e(k) = SP(k) - PV(k)$$

The PID controller operates on this error and tries to drive it to zero. The PID controller produces an output $u(k)$, also known as a control action, according to the following formula:

$$u(k) = u_p(k) + u_i(k) + u_d(k)$$

where $u_p(k)$ is the control output due to the proportional term, $u_i(k)$ is the control output due to the integral term, and $u_d(k)$ is the control output due to the derivative term. The PID controller calculates these control outputs according to the following formulas:

$$u_p(k) = K_p e(k)$$

$$u_i(k) = K_p K_i \sum_{i=1}^{k} \left[\frac{e(i) + e(i-1)}{2} \right]$$

$$u_d(k) = -K_p K_d [PV(k) - PV(k-1)]$$

Finally, the PID controller limits the output within an output range that you specify.

$$\begin{cases} \text{If } u(k) \geq u_{max}, u(k) = u_{max} \\ \text{If } u(k) \leq u_{min}, u(k) = u_{min} \end{cases}$$

Implementing a Single-Channel PID on FPGA Targets

To configure the PID (FPGA) Express VI for a single-channel implementation, enter `1` in the **Number of channels** control on the configuration dialog box. For single-channel PID implementations, any changes you make to the PID parameters on the host VI take effect immediately on the FPGA target.

Refer to the `Using Discrete PID - cRIO.lvproj` in the `labview\examples\control\pid\fpga.llb\CompactRIO` directory for an example of a single-channel PID implementation on an FPGA target. You also can refer to the `Using Discrete PID - R Series.lvproj` in the `labview\examples\control\pid\fpga.llb\R Series` directory.

Implementing a Multi-Channel PID on FPGA Targets

To configure the PID (FPGA) Express VI for a multi-channel implementation, enter a number greater than `1` in the **Number of channels** control on the configuration dialog box. To handle the values for each channel, the host VI can include an initialization loop and processing loop, which you can use to modify parameters or reset channels. The following sections provide information about these loops.

Initialization Loop on the Host VI

The initial parameter values you enter on the configuration dialog box for the PID (FPGA) Express VI set the initial values for all channels. To specify unique values for individual channels, you must create an initialization loop on the host VI.

In the initialization loop, you might update the parameter values of some channels while other channels retain the initial values you specified in the configuration dialog box. In Figure 3-14, the initialization loop initializes the parameters of all channels.

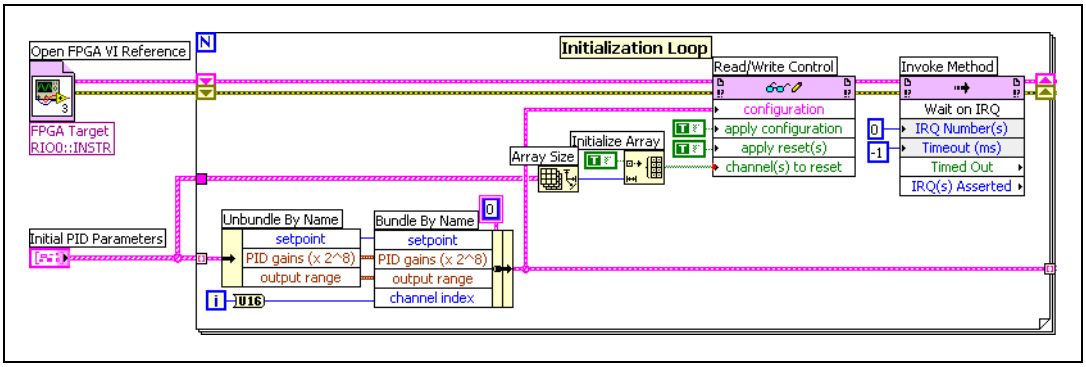

Figure 3-14. Initializing PID Parameters on the Host VI

Processing Loop on the Host VI

After initializing the PID parameters and while running the VI on the FPGA target, you might want to modify parameters or reset channels on a per-channel basis, without affecting the execution of other channels. Create a processing loop on the host VI to modify parameters or reset channels asynchronously during execution of the FPGA VI.

Resetting a PID channel resets the internal channel states to 0 and restarts the PID algorithm while using either the parameter values stored in memory or the parameter values from the host VI. To modify different parameters, you must specify the channel index, which tells the PID algorithm the channel number to which to apply the new parameter values. Figure 3-15 shows an example of a processing loop.

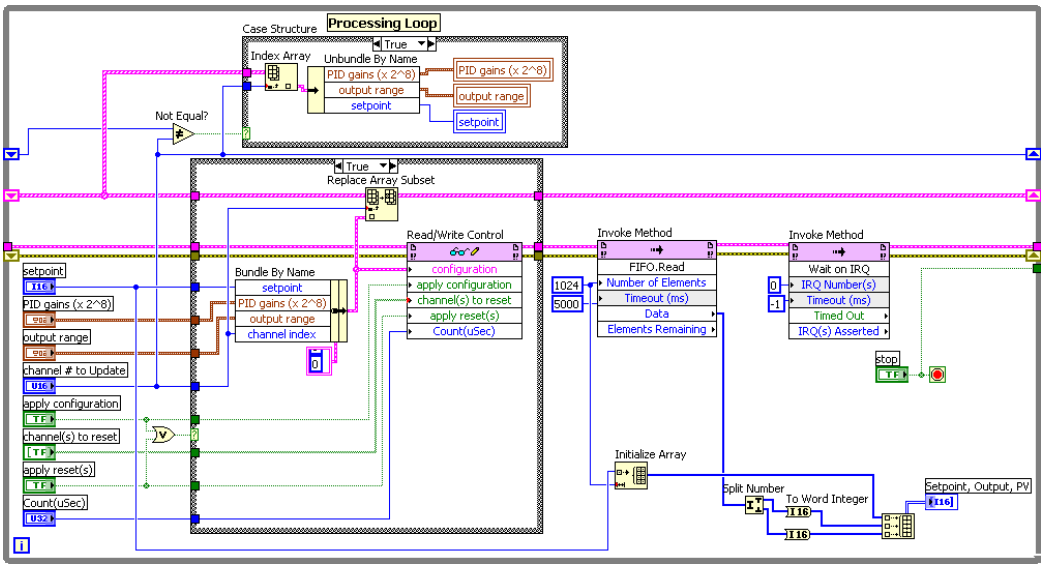

Figure 3-15. Processing PID Parameters on the Host VI

Refer to the `Template Multichannel PID.lvproj` in the `labview\examples\control\pid\fpga.llb\Template\Multichannel` directory for an example of a multi-channel PID implementation on an FPGA target.

4

Overview of Fuzzy Logic

Fuzzy logic is a method of rule-based decision making used for expert systems and process control. Fuzzy logic differs from traditional Boolean logic in that fuzzy logic allows for partial membership in a set.

Traditional Boolean logic is two-valued in the sense that a member either belongs to a set or does not. Values of one and zero represent the membership of a member to the set with one representing absolute membership and zero representing no membership. Fuzzy logic allows for partial membership, or a degree of membership, which might be any value along the continuum of zero to one.

Fuzzy Systems

A *fuzzy system* is a system of variables that are associated using fuzzy logic. A *fuzzy controller* uses defined rules to control a fuzzy system based on the current values of input variables.

You can use the Fuzzy System Designer and the Fuzzy Logic VIs to design and control fuzzy systems. Refer to Chapter 9, *Designing a Fuzzy System with the Fuzzy System Designer*, for information about the Fuzzy System Designer. Refer to Chapter 10, *Modifying a Fuzzy System with the Fuzzy Logic VIs*, for information about the Fuzzy Logic VIs.

Fuzzy systems consist of three main parts: linguistic variables, membership functions, and rules.

Linguistic Variables

Linguistic variables represent, in words, the input variables and output variables of the system you want to control. For a heater, you might have two input linguistic variables, *current temperature* and *desired temperature*, and one output linguistic variable, *heater setting*. Each linguistic variable has a range of expected values. For example, the range of *current temperature* might be 0 to 100 degrees. The range of *desired temperature* might be 50 to 80 degrees.

A fuzzy controller requires at least one input linguistic variable and one output linguistic variable.

Refer to the *Creating Linguistic Variables* section of Chapter 5, *Designing a Fuzzy System*, for more information about linguistic variables.

Linguistic Terms and Membership Functions

Linguistic terms represent, in words, categories for the values of a linguistic variable. The linguistic variables *current temperature* and *desired temperature* each might include the linguistic terms *cold*, *moderate*, and *hot*. The linguistic variable *heater setting* might include the linguistic terms *off*, *low*, and *high*.

Membership functions are numerical functions corresponding to linguistic terms. A membership function represents the degree of membership of linguistic variables within their linguistic terms. The degree of membership is continuous between 0 and 1, where 0 is equal to 0% membership and 1 is equal to 100% membership. For example, the linguistic variable *current temperature* might have full membership (1) within the linguistic term *hot* at 100 degrees, no membership (0) within that term at 70 degrees or less, and partial membership at all temperatures between 70 and 100 degrees.

Refer to the *Creating Membership Functions* section of Chapter 5, *Designing a Fuzzy System*, for more information about membership functions.

Rules

Rules describe, in words, the relationships between input and output linguistic variables based on their linguistic terms. For example, you might define the following rule:

IF *current temperature* is *cold* AND *desired temperature* is *moderate*, THEN *heater setting* is *low.*

The clauses "*current temperature* is *cold*" and "*desired temperature* is *moderate*" are the antecedents of this rule. The AND connective specifies how the fuzzy logic controller relates the two antecedents to determine the truth value for the aggregated rule antecedent. The clause "*heater setting* is *low*" is the consequent of this rule.

A *rule base* is the set of rules for a fuzzy system. The rule base is equivalent to the control strategy of the controller.

Refer to the *Creating a Rule Base* section of Chapter 5, *Designing a Fuzzy System*, for more information about rules.

Fuzzy Controllers

You can use fuzzy controllers to control fuzzy systems. Most traditional control algorithms require a mathematical model of the system you want to control. However, many physical systems are difficult or impossible to model mathematically. In addition, many processes are either nonlinear or too complex for you to control with traditional strategies. However, if you can describe a control strategy qualitatively, you can use fuzzy logic to create a fuzzy controller that emulates a heuristic rule-of-thumb strategy.

Figure 4-1 illustrates the process of a fuzzy controller.

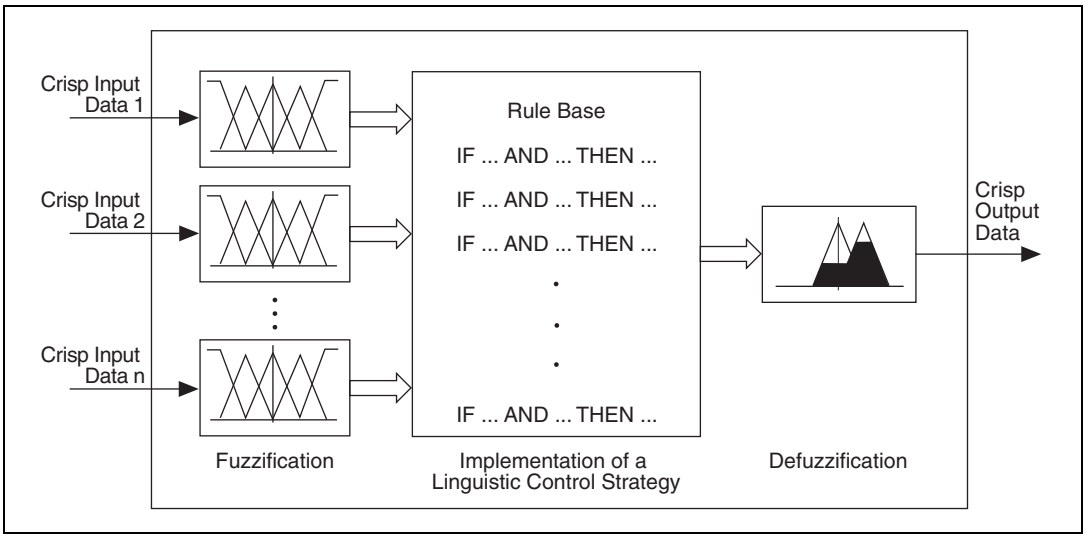

Figure 4-1. Process of a Fuzzy Controller

Fuzzification

Fuzzification is the process of associating *crisp*, or numerical, input values with the linguistic terms of the corresponding input linguistic variables.

For example, a fuzzy controller might associate the temperature reading from a thermometer with the linguistic terms *cold*, *moderate*, and *hot* for the *current temperature* linguistic variable. Depending on the membership functions for the linguistic terms, the temperature value might correspond to one or more of the linguistic terms.

Implementing a Linguistic Control Strategy

After a fuzzy controller fuzzifies the input values of a fuzzy system, the fuzzy controller uses the corresponding input linguistic terms and the rule base to determine the resulting linguistic terms of the output linguistic variables.

For example, suppose the *current temperature* of a room is 50 degrees, which corresponds to a linguistic term of *cold* with a degree of membership of 0.4. Also suppose the desired temperature is 70, which corresponds to a linguistic term of *moderate* with a degree of membership of 0.8. The fuzzy controller invokes the following rule of the fuzzy system: IF *current temperature* is *cold* AND *desired temperature* is *moderate*, THEN *heater setting* is *low.*

Notice that this rule consists of two antecedents, "*current temperature* is *cold*" and "*desired temperature* is *moderate*". The truth value of each antecedent is equal to the degree of membership of the linguistic variable within the corresponding linguistic term. The fuzzy logic controller uses an *antecedent connective* to determine how to calculate the truth value of the aggregated rule antecedent. Suppose the invoked rule in this example uses the AND (Minimum) antecedent connective, which specifies to use the smallest degree of membership of the antecedents as the truth value of the aggregated rule antecedent. Therefore, the truth value of the aggregated rule antecedent is 0.4.

You can specify a degree of support for each rule of a fuzzy system. The weight of a rule is equal to the degree of support multiplied by the truth value of the aggregated rule antecedent. The fuzzy controller uses an *implication method* to scale the membership functions of an output linguistic variable based on the rule weight before performing *defuzzification*.

Refer to the *Specifying a Consequent Implication* section of Chapter 5, *Designing a Fuzzy System*, for more information about implication methods.

Defuzzification

Defuzzification is the process of converting the degrees of membership of output linguistic variables within their linguistic terms into crisp numerical values. A fuzzy controller can use one of several mathematical methods to perform defuzzification. The most accurate defuzzification method for a fuzzy controller varies based on the control application.

Refer to Chapter 6, *Defuzzification Methods*, for more information about defuzzification methods.

5

Designing a Fuzzy System

A fuzzy system consists of three main parts: linguistic variables, membership functions, and rules. This chapter describes the general process of designing a fuzzy system. Refer to Chapter 9, *Designing a Fuzzy System with the Fuzzy System Designer*, and Chapter 10, *Modifying a Fuzzy System with the Fuzzy Logic VIs*, for information about designing a fuzzy system in the LabVIEW PID and Fuzzy Logic Toolkit.

Creating Linguistic Variables

Linguistic variables represent, in words, the input variables and output variables of the system you want to control.

When you create a linguistic variable to represent an input or output variable, decide how many linguistic terms, or categories of values of the linguistic variable, you want to create. Linguistic variables usually have an odd number of linguistic terms, with a middle linguistic term and symmetric linguistic terms at each extreme. In most applications, three to seven linguistic terms are sufficient for categorizing the values of a linguistic variable.

Consider an example in which you want to automate a vehicle to park itself from an arbitrary starting position. A driver can control the vehicle by constantly evaluating the current status of the vehicle, such as the distance from the target position and the orientation of the vehicle, to derive the correct steering angle. Figure 5-1 represents this example.

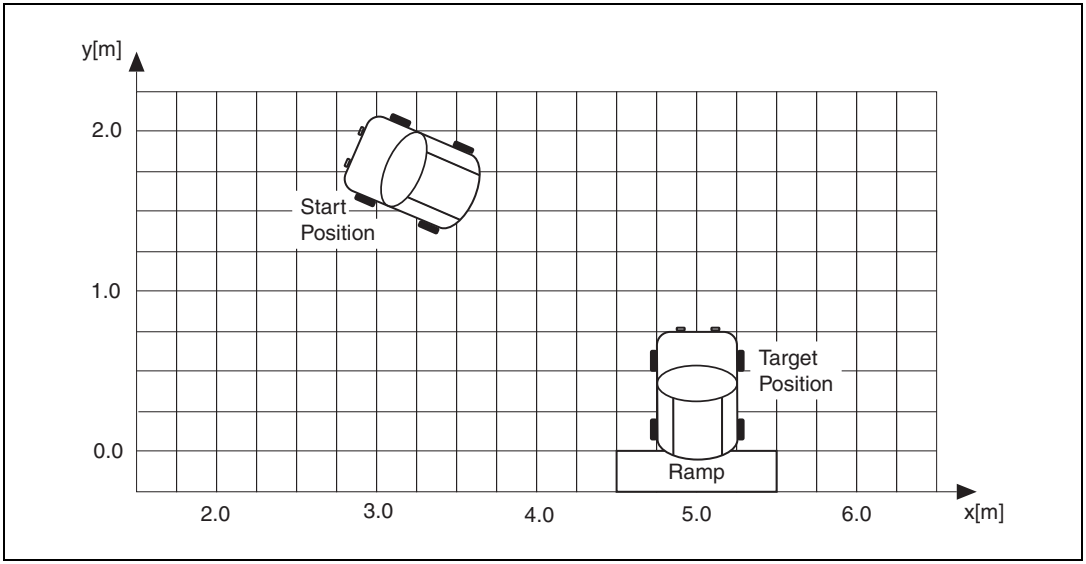

Figure 5-1. Automating Vehicle Parking

You can define two input linguistic variables for this example. *Vehicle Position x* represents the vehicle position in relation to the destination. *Vehicle Orientation* β represents the orientation of the vehicle. You also can define an output linguistic variable, *Steering Angle* φ, to represent the steering angle of the vehicle that you want to control.

You can define linguistic terms of *Left*, *Left Center*, *Center*, *Right Center*, and *Right* for the *Vehicle Position x* input linguistic variable to describe the possible positions of the vehicle in relation to the destination. You can define linguistic terms of *Left Down*, *Left*, *Left Up*, *Up*, *Right Up*, *Right*, and *Right Down* for the *Vehicle Orientation* β input linguistic variable to describe the possible orientations of the vehicle. The linguistic terms of the *Steering Angle* φ output linguistic variable must represent both the direction and magnitude that the steering angle changes. Therefore, you can use the linguistic terms *Negative Large*, *Negative Medium*, *Negative Small*, *Zero*, *Positive Small*, *Positive Medium*, and *Positive Large* for this output linguistic variable.

Creating Membership Functions

Membership functions are numerical functions corresponding to linguistic terms. A membership function represents the degree of membership of linguistic variables within their linguistic terms.

You can apply the normalized standard membership functions illustrated in Figure 5-2 to most technical processes. These standard functions include Λ-type (triangular shape), Π-type (trapezoidal shape), singleton-type (vertical line shape), Sigmoid-type (wave shape), and Gaussian-type (bell shape) membership functions.

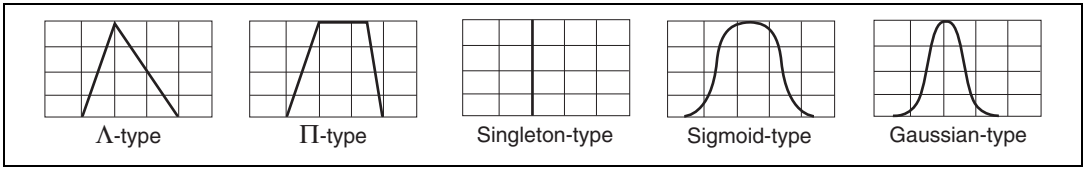

Figure 5-2. Shapes of Standard Membership Functions

For example, the linguistic variable *Vehicle Position x* might have full membership (1) within the linguistic term *Center* at 5 meters, no membership (0) within that term at 4 meters or less and 6 meters or greater, and partial membership at all distances between 4 and 6 meters. If you plot the degree of membership against the value of *Vehicle Position x*, you can see that the resulting membership function is a triangle function.

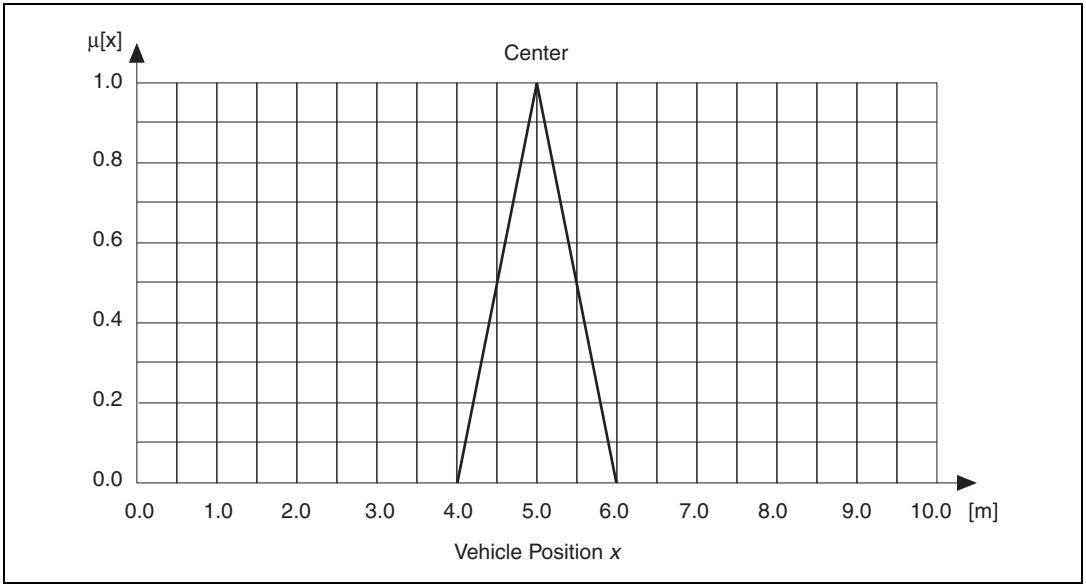

Figure 5-3. Triangular Membership Function for the Linguistic Term *Center*

Sometimes a linguistic variable has full membership within a linguistic term at a range of values rather than at a point value. If, for example, the linguistic variable *Vehicle Position x* has full membership within the linguistic term *Center* at values $x = 5 \pm 0.25$ m, a trapezoidal membership function applies, as shown in Figure 5-4.

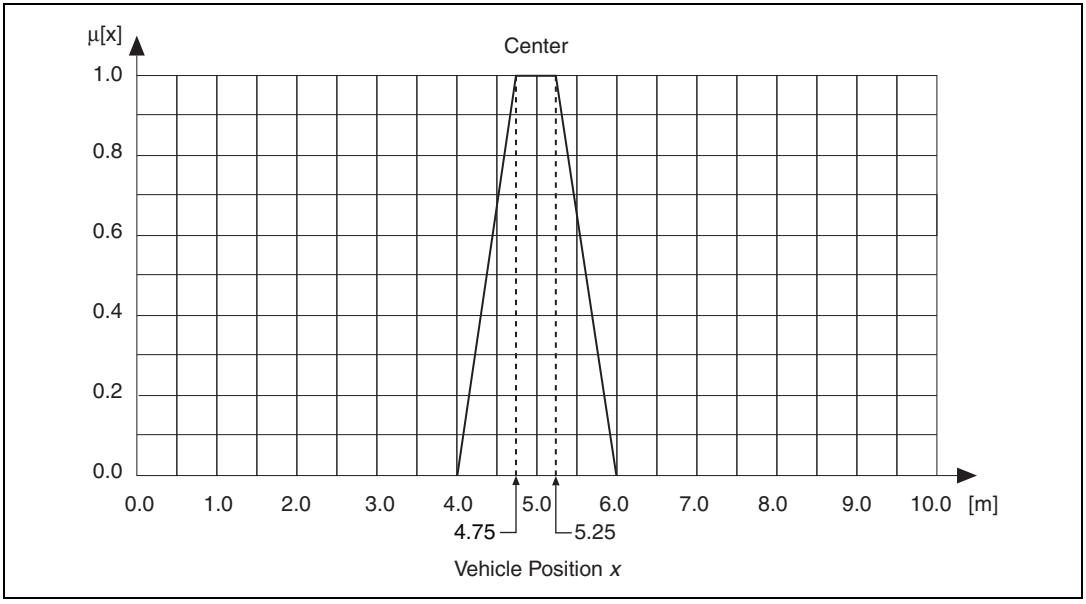

Figure 5-4. Trapezoidal Membership Function for the Linguistic Term *Center*

Figures 5-5, 5-6, and 5-7 show all membership functions for the input and output linguistic variables of the vehicle maneuvering fuzzy system.

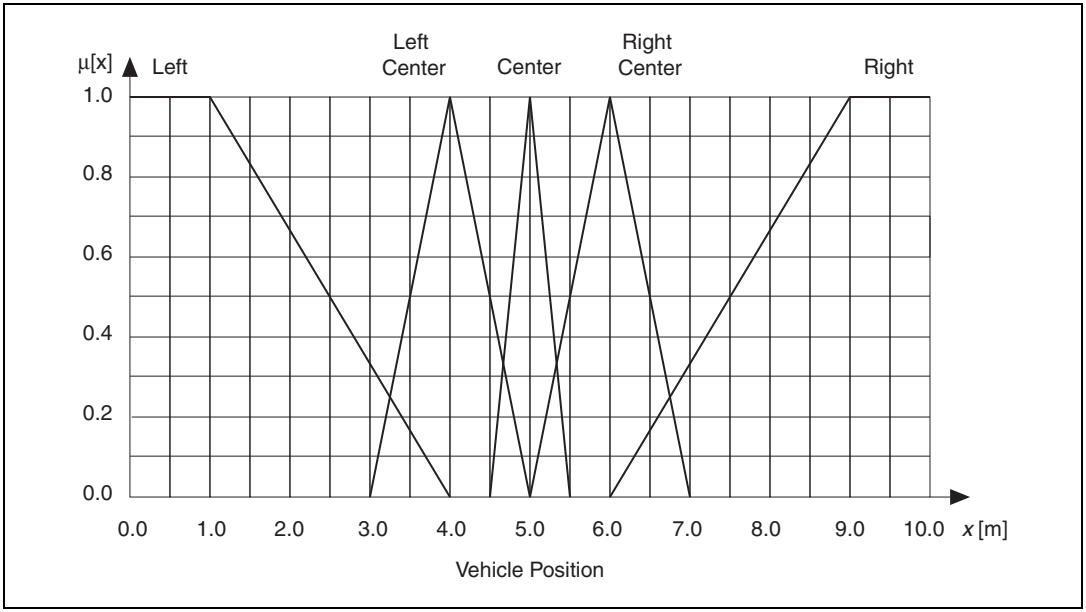

Figure 5-5. Membership Functions for *Vehicle Position x*

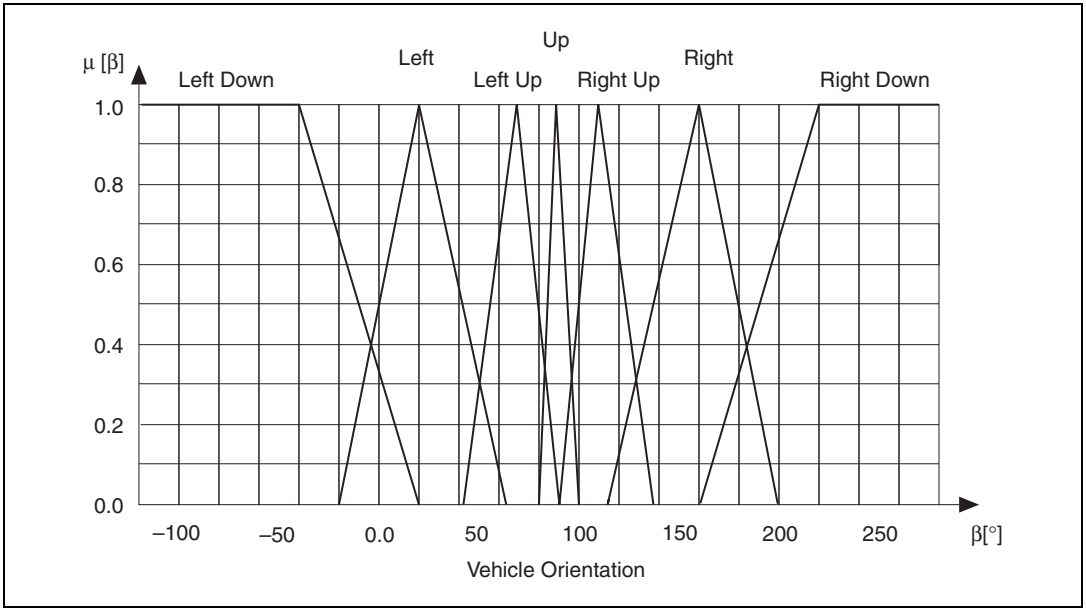

Figure 5-6. Membership Functions for *Vehicle Orientation* β

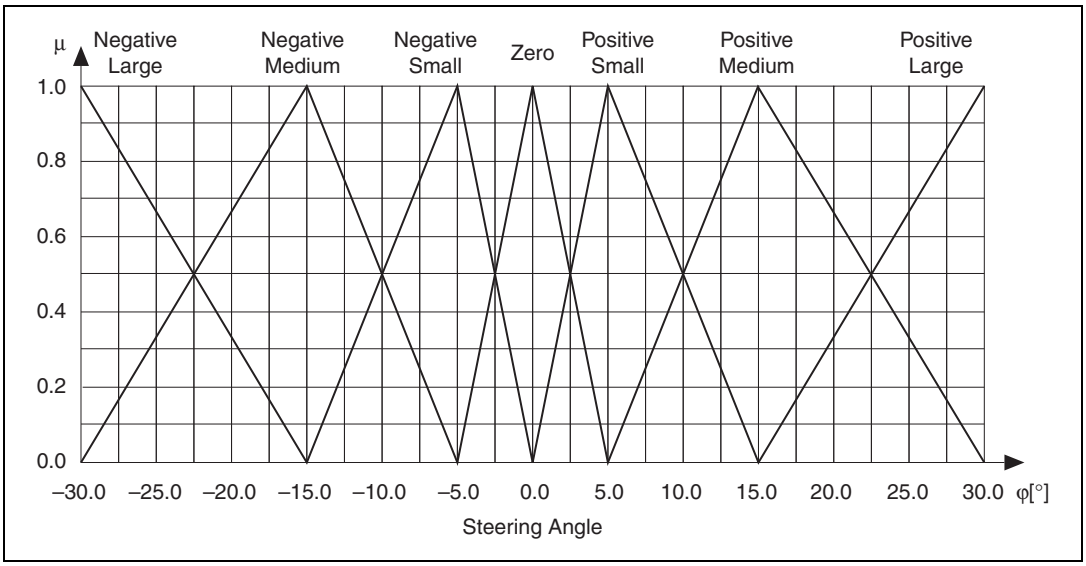

Figure 5-7. Membership Functions for *Steering Angle* φ

Creating a Rule Base

Rules describe, in words, the relationships between input and output linguistic variables based on their linguistic terms. A rule base is the set of rules for a fuzzy system.

To create a rule, you must specify the antecedents, or IF portions, and consequents, or THEN portions, of the rule. For example, consider the following rule: IF *Vehicle Position x* is *Left Center* AND *Vehicle Orientation* β is *Left Up*, THEN *Steering Angle* φ is *Positive Small*. The clauses "*Vehicle Position x* is *Left Center*" and "*Vehicle Orientation* β is *Left Up*" are the antecedents of this rule. The clause "*Steering Angle* φ is *Positive Small*" is the consequent of this rule.

Associate an input linguistic variable with a corresponding linguistic term to form an antecedent. Associate an output linguistic variable with a corresponding linguistic term to form a consequent. The consequent of a rule represents the action you want the fuzzy controller to take if the linguistic terms of the input linguistic variables in the rule are met.

When constructing a rule base, avoid contradictory rules, or rules with the same IF portion but different THEN portions. A consistent rule base is a rule base that has no contradictory rules.

The total number *N* of possible rules for a fuzzy system is defined by the following equation:

$$N = p_1 \times p_2 \times \ldots \times p_n$$

where p_n is the number of linguistic terms for the input linguistic variable *n*.

If each input linguistic variable has the same number of linguistic terms, the total number *N* of possible rules is defined by the following equation:

$$N = p^m$$

where *p* is the number of linguistic terms for each input linguistic variable and *m* is the number of input linguistic variables. For example, for three input linguistic variables with five linguistic terms each, the total number of possible rules is $N = 5^3 = 125$.

A rule base with at least one active rule for each possible combination of input linguistic variables and linguistic terms is a complete rule base. If you define an incomplete rule base, you must specify a default linguistic term for each output linguistic variable so the fuzzy controller can handle situations in which no rules are active.

The *Vehicle Position x* input linguistic variable has five linguistic terms, and the *Vehicle Orientation* β linguistic variable has seven linguistic terms. Therefore, the rule base of the vehicle maneuvering example consists of $N = 5 \times 7 = 35$ rules. You can document the complete rule base in matrix form, as shown in Figure 5-8.

AND		Vehicle Position *x* [m]				
		Left	Left Center	Center	Right Center	Right
Vehicle Orientation β[°]	Left Down	Negative Small	Negative Medium	Negative Medium	Negative Large	Negative Large
	Left	Positive Small	Negative Small	Negative Medium	Negative Large	Negative Large
	Left Up	Positive Medium	Positive Small	Negative Small	Negative Medium	Negative Large
	Up	Positive Medium	Positive Medium	Zero	Negative Medium	Negative Medium
	Right Up	Positive Large	Positive Medium	Positive Small	Negative Small	Negative Medium
	Right	Positive Large	Positive Large	Positive Medium	Positive Small	Negative Small
	Right Down	Positive Large	Positive Large	Positive Medium	Positive Medium	Negative Small

Figure 5-8. Complete Rule Base for the Vehicle Maneuvering Example

Each column or row represents an antecedent of a rule. The term at the intersection of a column and a row is the consequent of the rule corresponding to the aggregated rule antecedent. For example, the following rule is highlighted in Figure 5-8.

IF *Vehicle Position x* is *Left Center* AND *Vehicle Orientation* β is *Left*, THEN *Steering Angle* φ is *Negative Small*.

Plotting a rule base in matrix form, as in Figure 5-8, is helpful for detecting inconsistencies, such as contradictory rules. However, plotting a rule base in matrix form is efficient only for small rule bases. Detecting inconsistencies in large rule bases is difficult. For fuzzy systems with numerous controller inputs, you can use cascading fuzzy systems to avoid large rule bases. In cascading fuzzy systems, the outputs of the first fuzzy system serve as the inputs to the next fuzzy system, and so on.

Specifying an Antecedent Connective

If a rule has more than one antecedent, you must specify an antecedent connective to determine how to calculate the truth value of the aggregated rule antecedent.

Because linguistic variables can have partial degrees of membership within linguistic terms, you cannot use Boolean operators from conventional dual logic as antecedent connectives. The PID and Fuzzy Logic Toolkit uses the following antecedent connectives instead.

AND (Minimum):
$\mu A \bullet B = min(\mu A, \mu B)$

AND (Product):
$\mu A \bullet B = (\mu A, \mu B)$

OR (Maximum):
$\mu A + B = max(\mu A, \mu B)$

OR (Probabilistic):
$A + B = ((A + B) - (AB))$

Notice that these definitions agree with the logical operators used in Boolean logic. A truth table uses conventional operators to yield equivalent results.

The AND (Minimum) antecedent connective specifies to use the smallest degree of membership of the antecedents as the truth value of the aggregated rule antecedent, while the AND (Product) specifies to use the product of the degrees of membership of the antecedents. The OR (Maximum) antecedent connective specifies to use the largest degree of membership of the antecedents. The OR (Probabilistic) antecedent connective specifies to use the probabilistic sum of the degrees of membership of the antecedents.

Assume the following rules are invoked for a particular set of input values.

(1) IF *Vehicle Position x* is *Center* (degree of membership = 0.8) AND (Minimum) *Vehicle Orientation* β is *Left Up* (degree of membership = 1.0) = 0.8

THEN *Steering Angle* φ is *Negative Small*

(2) IF *Vehicle Position x* is *Right Center* (degree of membership = 0.1) AND (Minimum) *Vehicle Orientation* β is *Left Up* (degree of membership = 1.0) = 0.1

THEN *Steering Angle* φ is *Negative Medium*

Notice that each rule uses the AND (Minimum) antecedent connective. In Rule 1, *Vehicle Position x* has a degree of membership of 0.8 within the linguistic term *Center* and *Vehicle Orientation* β has a degree of membership of 1.0 within the linguistic term *Left Up*. Because the antecedent connective is AND (Minimum), the fuzzy controller for this fuzzy system uses the smallest degree of membership of the antecedents, or 0.8, as the truth value of the aggregated rule antecedent. Similarly, the smallest degree of membership of the antecedents in Rule 2 is 0.1. Therefore, the fuzzy controller uses 0.1 as the truth value of the aggregated rule antecedent. The truth value of the aggregated rule antecedent is equivalent to the degree of truth of the rule.

If these two rules are the only rules invoked for a given set of input values, the other linguistic terms for the *Steering Angle* φ output linguistic variable have a truth value of 0. The following table describes the final truth values for each of the linguistic terms.

Negative Large	to a degree of	0.0
Negative Medium	to a degree of	0.1
Negative Small	to a degree of	0.8
Zero	to a degree of	0.0
Positive Small	to a degree of	0.0
Positive Medium	to a degree of	0.0
Positive Large	to a degree of	0.0

If a rule has more than one consequent, the fuzzy logic controller must evaluate the aggregated rule consequent. When you implement a fuzzy controller in the PID and Fuzzy Logic Toolkit, the fuzzy controller always considers only the consequent that has the largest degree of membership.

Specifying a Degree of Support

You can specify a degree of support, between 0 and 1, for each rule of a fuzzy system. The degree of support represents the relative significance of each rule and allows for fine-tuning of the rule base. In most cases, the degree of support is 1. The final weight of a rule is equal to the degree of support multiplied by the truth value of the aggregated rule antecedent.

Specifying a Consequent Implication

A fuzzy controller uses a consequent implication method to scale the membership functions of each output linguistic variable based on the corresponding rule weight before performing defuzzification. Refer to Chapter 6, *Defuzzification Methods*, for more information about defuzzification.

In the PID and Fuzzy Logic Toolkit, you can use either the Minimum or Product consequent implication method.

In the Minimum implication method, the fuzzy logic controller truncates the output membership functions at the value of the corresponding rule weights. For example, if an output linguistic variable has three membership functions with rule weights 0.5, 0.8, and 0.3, respectively, the scaled membership functions might appear similar to Figure 5-9.

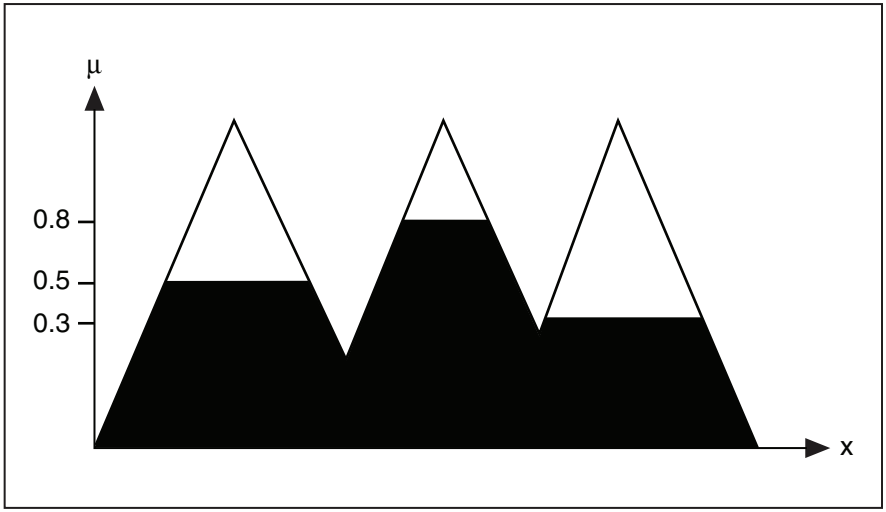

Figure 5-9. Minimum Implication Method

In the Product implication method, the fuzzy logic controller scales the output membership functions at the value of the corresponding rule weights. For example, if an output linguistic variable has three membership functions with rule weights 0.5, 0.8, and 0.3, respectively, the scaled membership functions might appear similar to Figure 5-10.

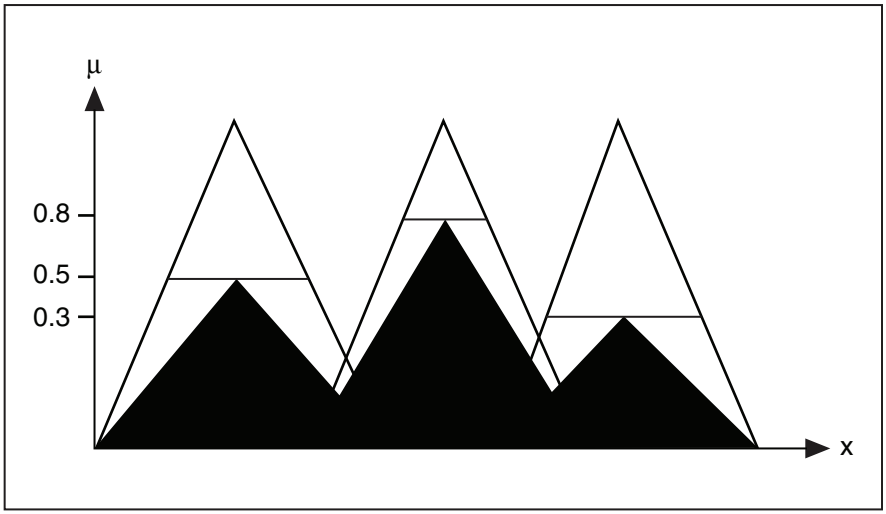

Figure 5-10. Product Implication Method

6

Defuzzification Methods

In Chapter 5, *Designing a Fuzzy System*, you learn how to design fuzzy systems. This chapter describes how a fuzzy controller performs defuzzification on a fuzzy system.

Defuzzification is the process of converting the degrees of membership of output linguistic variables within their linguistic terms into crisp numerical values. For example, recall the following rules described in the *Specifying an Antecedent Connective* section of Chapter 5, *Designing a Fuzzy System*.

(1)	IF *Vehicle Position x* is *Center* (degree of membership = 0.8)	AND (Minimum)	*Vehicle Orientation* β is *Left Up* (degree of membership = 1.0) = 0.8

THEN *Steering Angle* φ is *Negative Small*

(2)	IF *Vehicle Position x* is *Right Center* (degree of membership = 0.1)	AND (Minimum)	*Vehicle Orientation* β is *Left Up* (degree of membership = 1.0) = 0.1

THEN *Steering Angle* φ is *Negative Medium*

These two rules specify two non-zero values for the *Steering Angle* φ output linguistic variable:

Negative Medium	to a degree of	0.1
Negative Small	to a degree of	0.8

A fuzzy controller performs defuzzification to evaluate these two linguistic values and convert them into a single numerical output value.

Note Fuzzy controllers use an implication method to scale the membership functions of output linguistic variables before performing defuzzification. Refer to the *Specifying a Consequent Implication* section of Chapter 5, *Designing a Fuzzy System*, for more information about implication methods.

A fuzzy controller can use one of several mathematical methods to perform defuzzification: Center of Area (CoA), modified Center of Area (CoA), Center of Sums (CoS), Center of Maximum (CoM), or Mean of Maximum (MoM).

Center of Area

In the Center of Area (CoA) defuzzification method, also called the Center of Gravity (CoG) method, the fuzzy controller first calculates the area under the scaled membership functions and within the range of the output variable. The fuzzy logic controller then uses the following equation to calculate the geometric center of this area.

$$CoA = \frac{\int_{x_{min}}^{x_{max}} f(x) \cdot x \, dx}{\int_{x_{min}}^{x_{max}} f(x) dx}$$

where CoA is the center of area, x is the value of the linguistic variable, and x_{min} and x_{max} represent the range of the linguistic variable.

The Center of Area defuzzification method effectively calculates the best compromise between multiple output linguistic terms.

Figure 6-1 illustrates the Center of Area defuzzification method for the *Steering Angle* φ output linguistic variable, assuming the Minimum implication method. The shaded portion of the graph represents the area under the scaled membership functions.

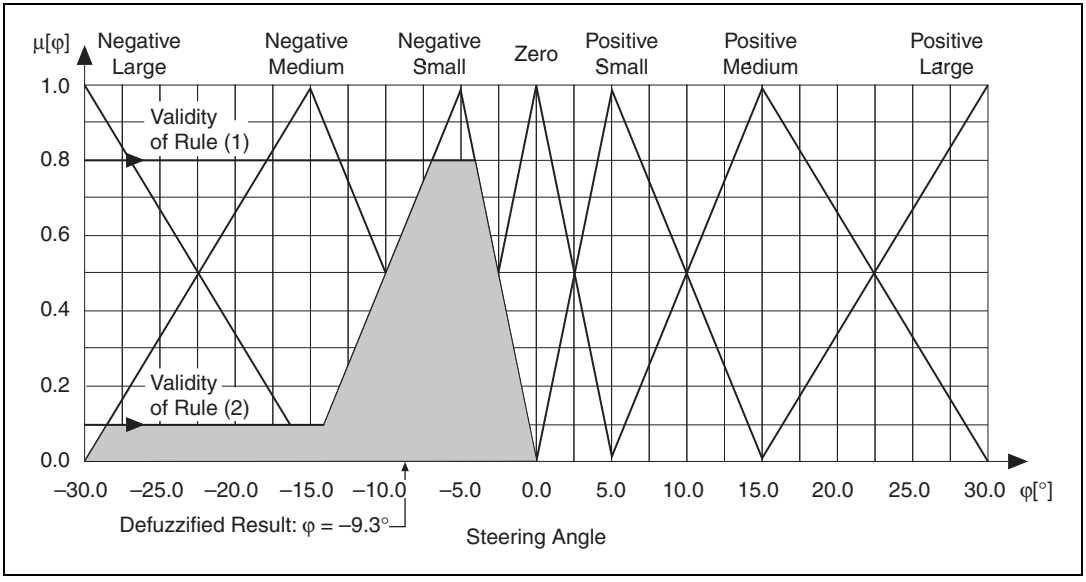

Figure 6-1. Center of Area (CoA) Defuzzification Method

Figure 6-2 summarizes the process of a fuzzy controller for the vehicle maneuvering example described in Chapter 5, *Designing a Fuzzy System*, using the CoA method of defuzzification.

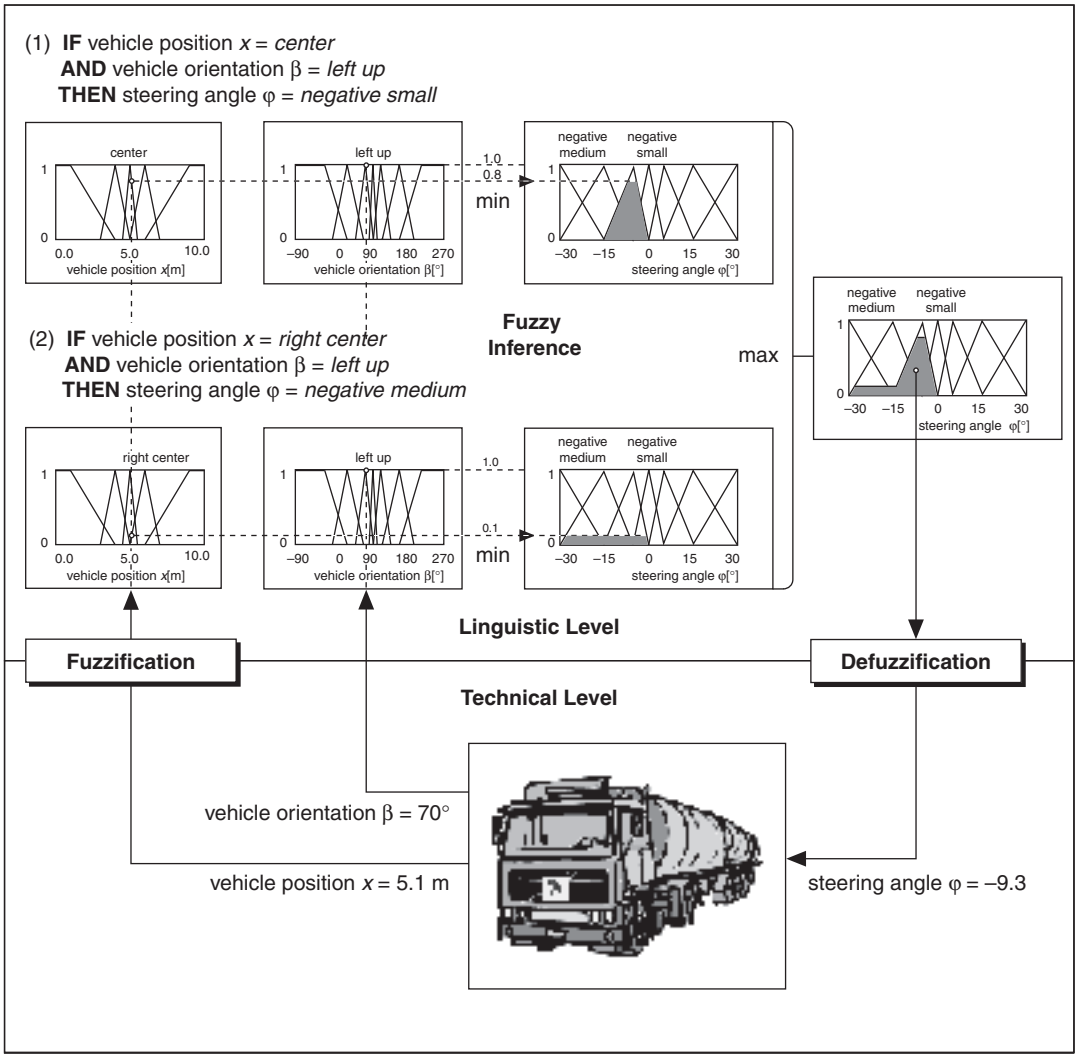

Figure 6-2. Process of a Fuzzy Controller Using CoA Defuzzification

Modified Center of Area

Because the CoA defuzzification method evaluates the area under the scaled membership functions only within the range of the output linguistic variable, the resulting crisp output values cannot span the full range. To solve this problem, use the modified Center of Area defuzzification method.

The modified Center of Area defuzzification method is similar to the Center of Area defuzzification method. However, the fuzzy logic controller considers the full area under the scaled membership functions, even if this area extends beyond the range of the output variable. The fuzzy logic controller uses the following equation to calculate the geometric center of the full area under the scaled membership functions.

$$mCoA = \frac{\int f(x) \cdot x \, dx}{\int f(x) dx}$$

where mCoA is the modified center of area.

The interval of integration is between the minimum membership function value and the maximum membership function value. Note that this interval might extend beyond the range of the output variable.

Figure 6-3 illustrates the difference between the CoA and modified CoA defuzzification methods.

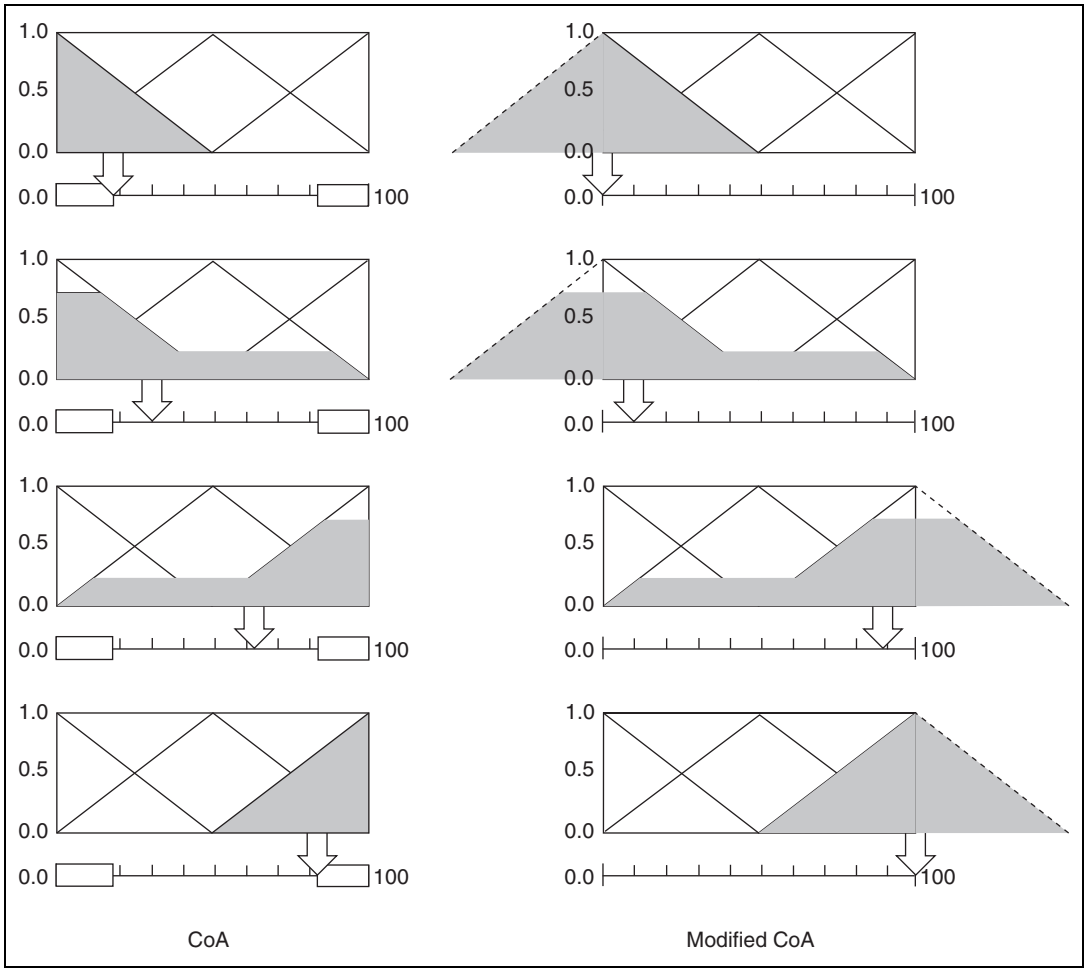

Figure 6-3. Comparison of CoA and Modified CoA Defuzzification Methods

Center of Sums

In the Center of Sums (CoS) defuzzification method, the fuzzy logic controller first calculates the geometric center of area for each membership function, as the Figure 6-4 illustrates:

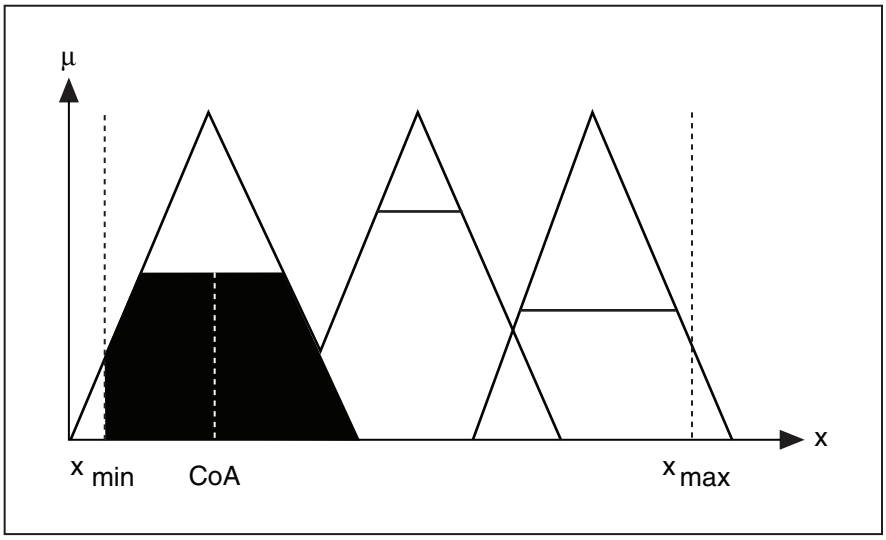

Figure 6-4. Calculating the Geometric Center of Area for a Membership Function

The fuzzy controller then uses the following equation to calculate a weighted average of the geometric center of area for all membership functions.

$$x_{final} = \frac{(CoA_1 area_1 + CoA_2 area_2 + \ldots + CoA_n area_n)}{(area_1 + area_2 + \ldots + area_n)}$$

where CoA_n is the geometric center of area of the scaled membership function n, and $area_n$ is the area of the scaled membership function n.

Center of Maximum

In the Center of Maximum (CoM) defuzzification method, the fuzzy logic controller first determines the typical numerical value for each scaled membership function. The typical numerical value is the mean of the numerical values corresponding to the degree of membership at which the membership function was scaled.

The fuzzy logic controller then uses the following equation to calculate a weighted average of the typical values.

$$x_{final} = \frac{(x_1\mu_1 + x_2\mu_2 + \ldots + x_n\mu_n)}{(\mu_1 + \mu_2 + \ldots + \mu_n)}$$

where x_n is the typical numerical value for the scaled membership function n, and μ_n is the degree of membership at which membership function n was scaled.

Figure 6-5 illustrates how to use the CoM defuzzification method with the vehicle maneuvering example described in Chapter 5, *Designing a Fuzzy System*.

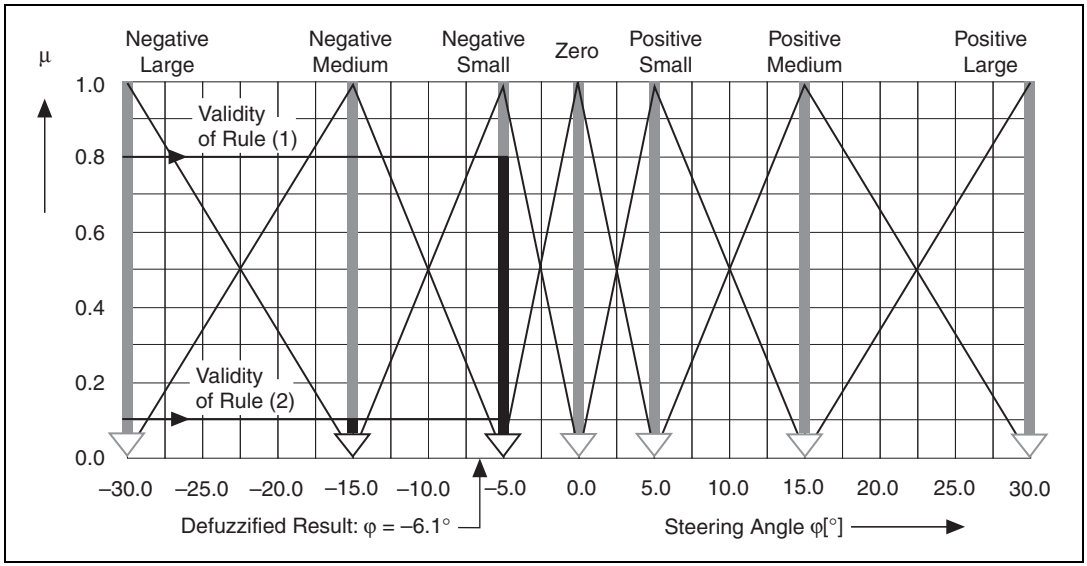

Figure 6-5. Center of Maximum (CoM) Defuzzification Method

The values –15° and –5° are the typical values of the linguistic terms *Negative Medium* and *Negative Small*. The degrees of truth for these linguistic terms are 0.1 and 0.8, respectively. Therefore, the defuzzified crisp output value φ_{final} is calculated by the following equation:

$$\varphi_{final} = \frac{((-15°)(0.1) + (-5°)(0.8))}{(0.1 + 0.8)} = -6.1°$$

The defuzzification method CoM is identical to using the CoA method with singleton membership functions.

The CoM and CoA defuzzification methods usually apply to closed-loop control applications of fuzzy logic. These methods usually result in continuous output signals because a small change in input values does not change the best compromise value for the output.

Mean of Maximum

Use the Mean of Maximum (MoM) defuzzification method for pattern recognition applications. This defuzzification method calculates the most plausible result. Rather than averaging the degrees of membership of the output linguistic terms, the MoM defuzzification method selects the typical value of the most valid output linguistic term.

Figure 6-6 illustrates the MoM defuzzification method.

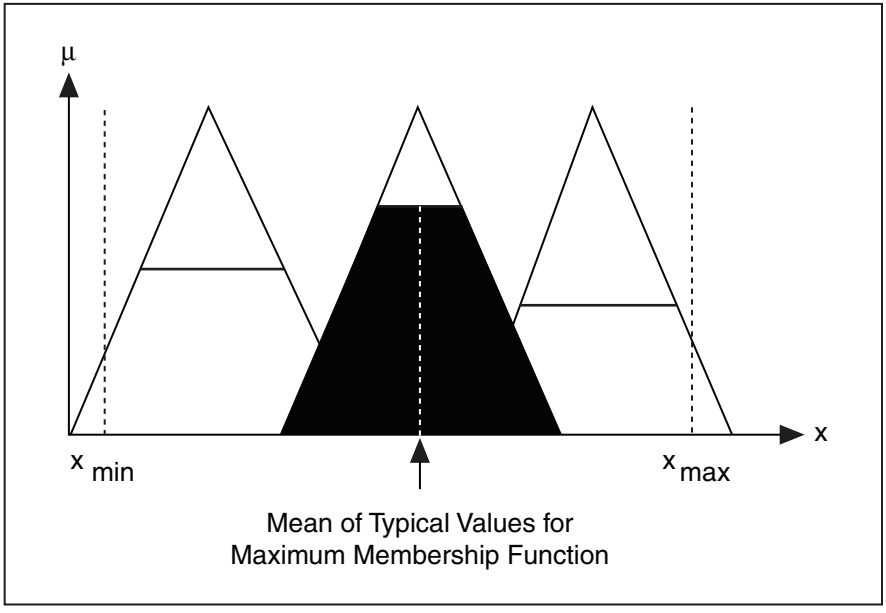

Figure 6-6. Mean of Maximum (MoM) Defuzzification Method

Selecting a Defuzzification Method

In decision support systems, the choice of the defuzzification method depends on the context of the decision you want to calculate with the fuzzy controller. For quantitative decisions like project prioritization, apply the CoM method. For qualitative decisions, such as an evaluation of credit worthiness, MoM is the correct method.

An important aspect of a defuzzification method is the continuity of the output signal. Consider a fuzzy system with a complete rule base and overlapping membership functions. A defuzzification method is continuous if an arbitrary small change of an input value never causes an abrupt change in the output signal.

In this respect, the defuzzification methods CoM and CoA are continuous because, assuming overlapping output membership functions, the best compromise does not jump to a different value with a small change to the inputs. The defuzzification method MoM, however, is discontinuous because an arbitrary small change in the input values of the fuzzy system can cause the output value to switch to another, more plausible result.

Table 6-1 compares the different defuzzification methods based on various assessment criteria.

Table 6-1. Comparison of Different Defuzzification Methods

	Method			
Assessment Criteria	**Center of Area (CoA) and Modified Center of Area (mCoA)**	**Center of Sums (CoS)**	**Center of Maximum (CoM)**	**Mean of Maximum (MoM)**
Linguistic Characteristic	Best Compromise	Best Compromise	Best Compromise	Most Plausible Result
Fit with Intuition	Implausible with varying membership function shapes and strong overlapping membership functions	Implausible with varying membership function shapes and strong overlapping membership functions	Good	Good

Table 6-1. Comparison of Different Defuzzification Methods (Continued)

Assessment Criteria	Method			
	Center of Area (CoA) and Modified Center of Area (mCoA)	**Center of Sums (CoS)**	**Center of Maximum (CoM)**	**Mean of Maximum (MoM)**
Continuity	Yes	Yes	Yes	No
Computational Effort	Very High	Medium	Low	Very Low
Application Field	Closed-Loop Control, Decision Support, Data Analysis	Closed-Loop Control, Decision Support, Data Analysis	Closed-Loop Control, Decision Support, Data Analysis	Pattern Recognition, Decision Support, Data Analysis

7

I/O Characteristics of Fuzzy Controllers

You can consider a fuzzy controller to be a nonlinear characteristic field controller. The rule base and membership functions of the fuzzy system determine the behavior of the controller. Because the controller has no internal dynamic aspects, the I/O characteristics can describe the transient response of the controller completely.

To illustrate how the I/O characteristics of a fuzzy controller depend on design parameters such as the rule base and membership function specifications of the fuzzy system, first consider a single-input fuzzy controller. Many of the characteristics of a single-input fuzzy controller apply to fuzzy controllers with two or more inputs.

Figure 7-1 shows the I/O characteristic of a fuzzy controller for a fuzzy system that has only three linguistic terms for the input variable *x* and the output variable *y*. The rule base consists of three rules, which indicate that the increasing input values cause the output value to increase.

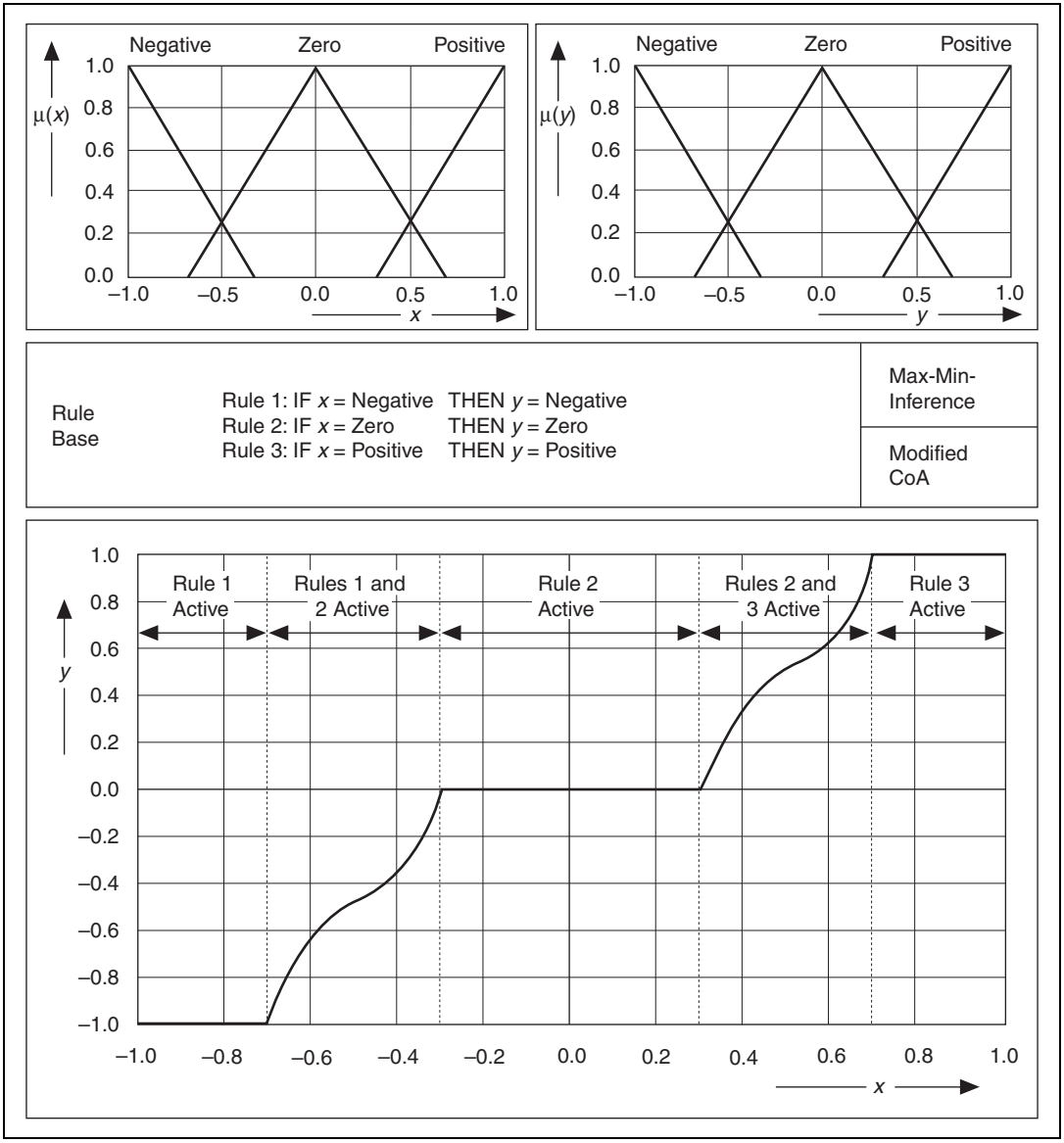

Figure 7-1. I/O Characteristic of a Fuzzy Controller (Partially Overlapping Input Terms)

The resulting controller characteristic shows nonlinear behavior. You obtain different intervals within the controller characteristic because the input linguistic terms partially overlap. The rule base has only one valid rule outside of the overlapping regions. The output therefore has a constant value determined by the output linguistic term of the output linguistic variable, which is independent of the degree of truth for that rule.

The overlapping sections of the antecedent terms lead to the rising intervals of the controller characteristic. Within these parts, two rules are simultaneously active. The different consequent terms, weighted by the degrees of truth of the different active rules, determine the output value. Notice that the overlapping triangular consequent terms cause the rising edges of the controller characteristic to be nonlinear.

Figure 7-2 shows the resulting controller characteristic for antecedent terms that overlap entirely. The consequent term distribution and the rule base remain unchanged for this case.

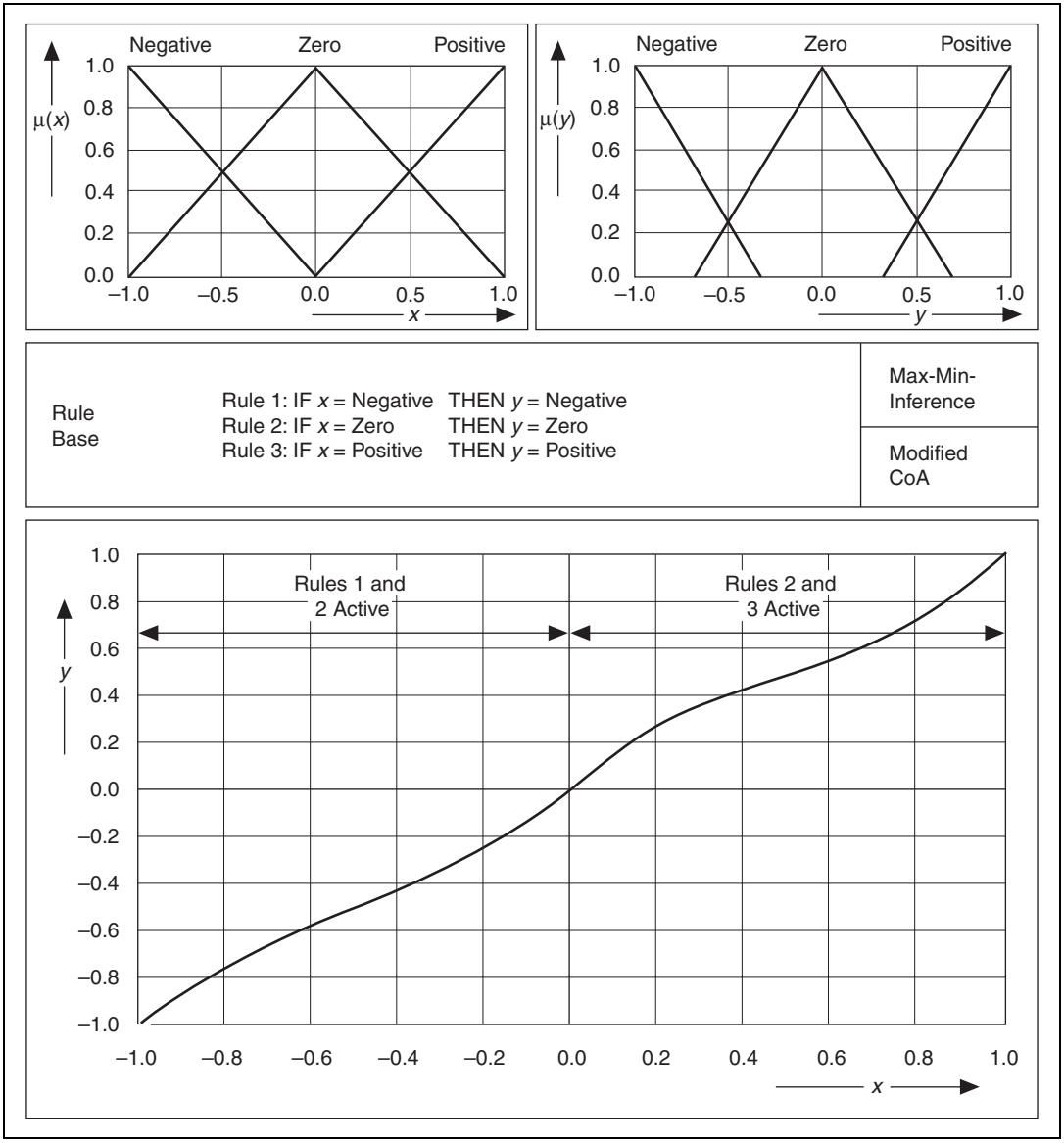

Figure 7-2. I/O Characteristic of a Fuzzy Controller (Entirely Overlapping Input Terms)

Because the antecedent terms completely overlap, the rule base always has two active rules. The different consequent terms that lead to the nonlinear pass of the controller characteristic and that are weighted by the degree of truth for the different active rules determine the output value.

Figure 7-3 shows the controller characteristic that results when nonoverlapping antecedent terms describe the input variable.

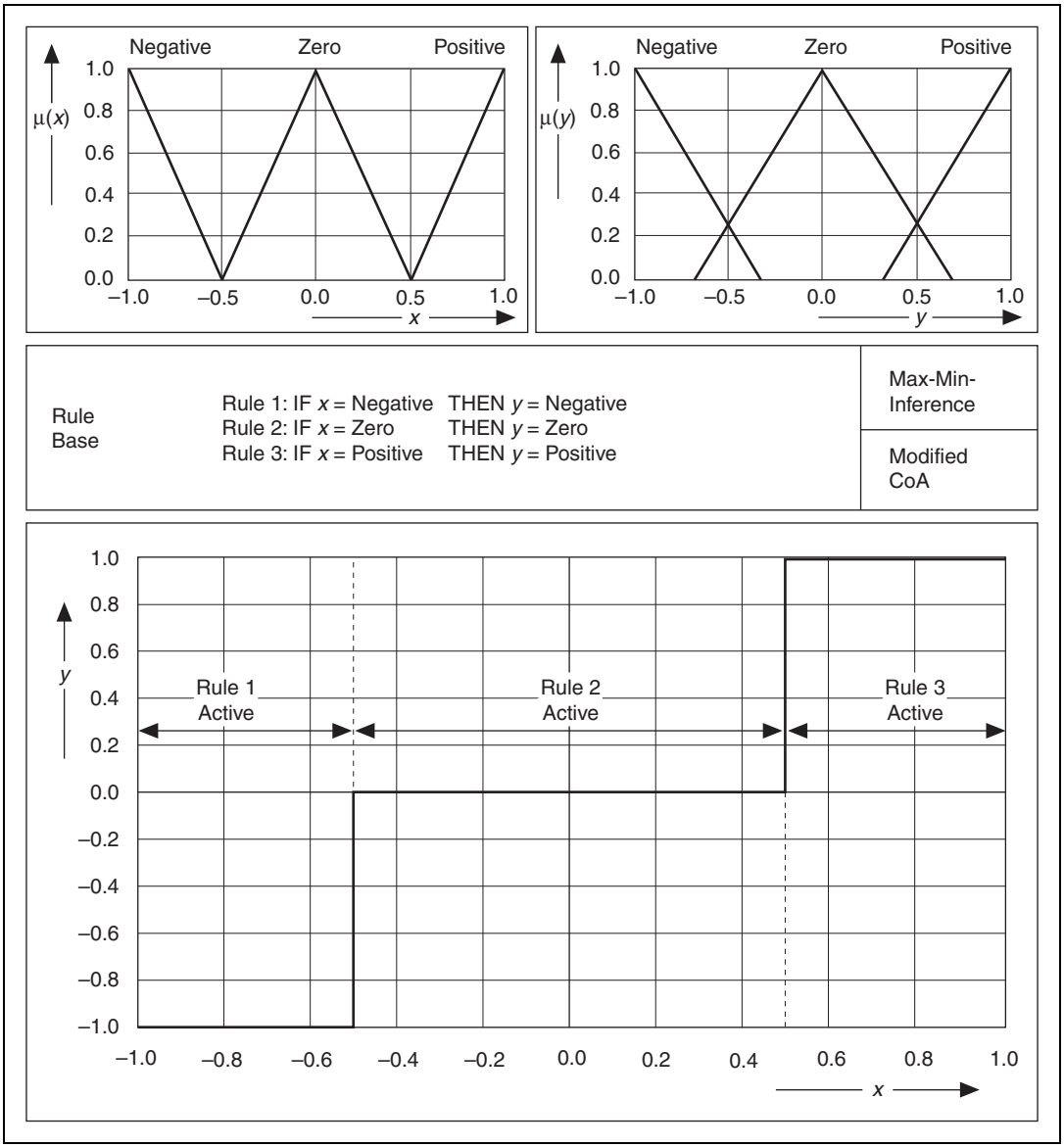

Figure 7-3. I/O Characteristic of a Fuzzy Controller (Nonoverlapping Input Terms)

In this case, only one rule is active for each input situation that leads to the stepped controller characteristic shown in Figure 7-3.

If the rule base has undefined intervals within input and output linguistic terms, or if the rule base is incomplete, you must specify the output of the fuzzy controller. If no rule is available for a certain situation, the output value remains undefined. One way to avoid this problem is to leave the current output value unchanged until the controller encounters a situation that is covered by the rules. Figure 7-4 shows the resulting effect on the controller characteristic.

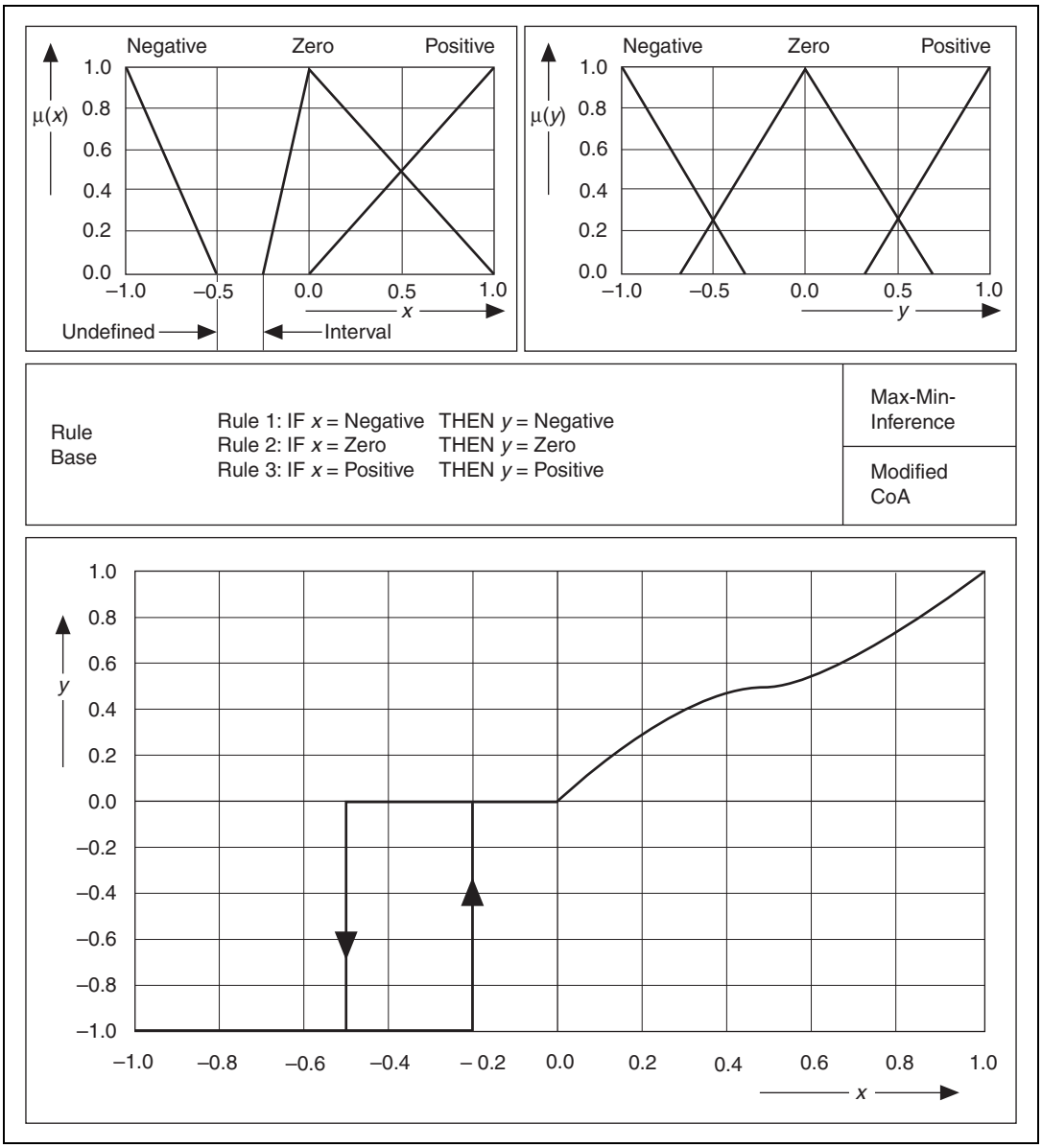

Figure 7-4. I/O Characteristic of a Fuzzy Controller (Undefined Input Term Interval)

If you use an old output value as a default value, undefined intervals or incomplete rule bases can lead to hysteretic effects on the controller characteristic.

You can use nonoverlapping, rectangular-shaped consequent terms to obtain an exact linear controller characteristic for a single-input controller. In this case, both the area and momentum vary linearly with the degree of truth, and overlapping regions of the output linguistic terms do not cause any distortion.

The simplest way to obtain a linear controller characteristic is to use singletons as consequent terms with entirely overlapping input terms. Figure 7-5 shows an example of such a controller.

Singletons are normalized rectangular membership functions with an infinitely small width. Using singleton membership functions for consequent terms makes the Center of Area (CoA) defuzzification method identical to the Center of Maximum (CoM) method. Figure 7-5 shows the controller for the CoA method using singleton membership functions.

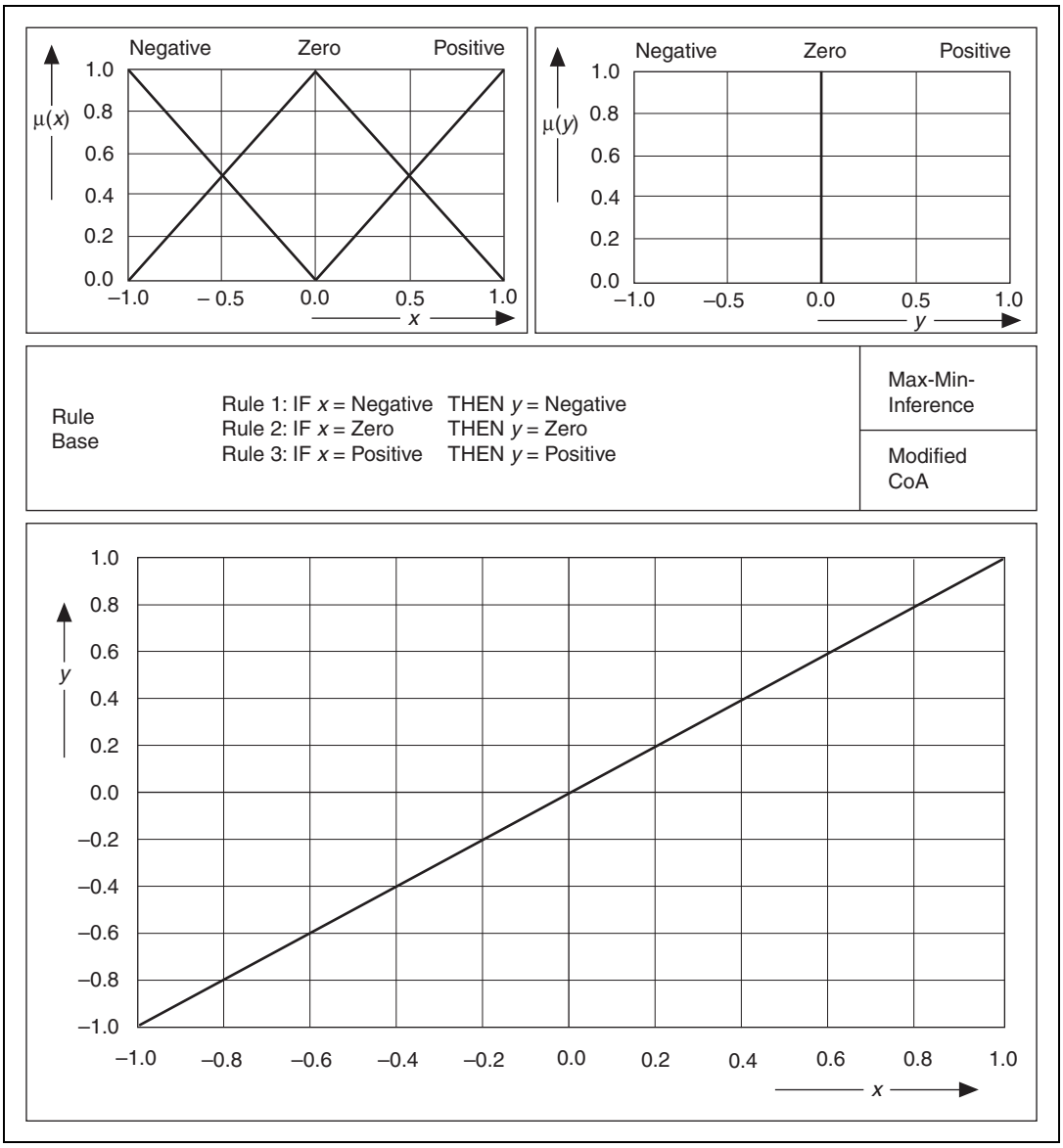

Figure 7-5. I/O Characteristic of a Fuzzy Controller (Singletons as Output Terms, Entirely Overlapping Input Terms)

The controller characteristic remains relatively unchanged when you leave the input terms entirely overlapped to vary the overlapping degree of the membership functions for the consequent terms, especially if all the consequent terms are equal in width. Then only the typical values of the consequent terms are significant.

Therefore, in most closed-loop control applications, you can use singleton membership functions to model the output terms sufficiently rather than using triangular or other membership function types.

Figure 7-6 shows that if all the consequent terms are equal in width, the overlapping degree of the membership functions for the consequent terms has no significant influence on the controller characteristic.

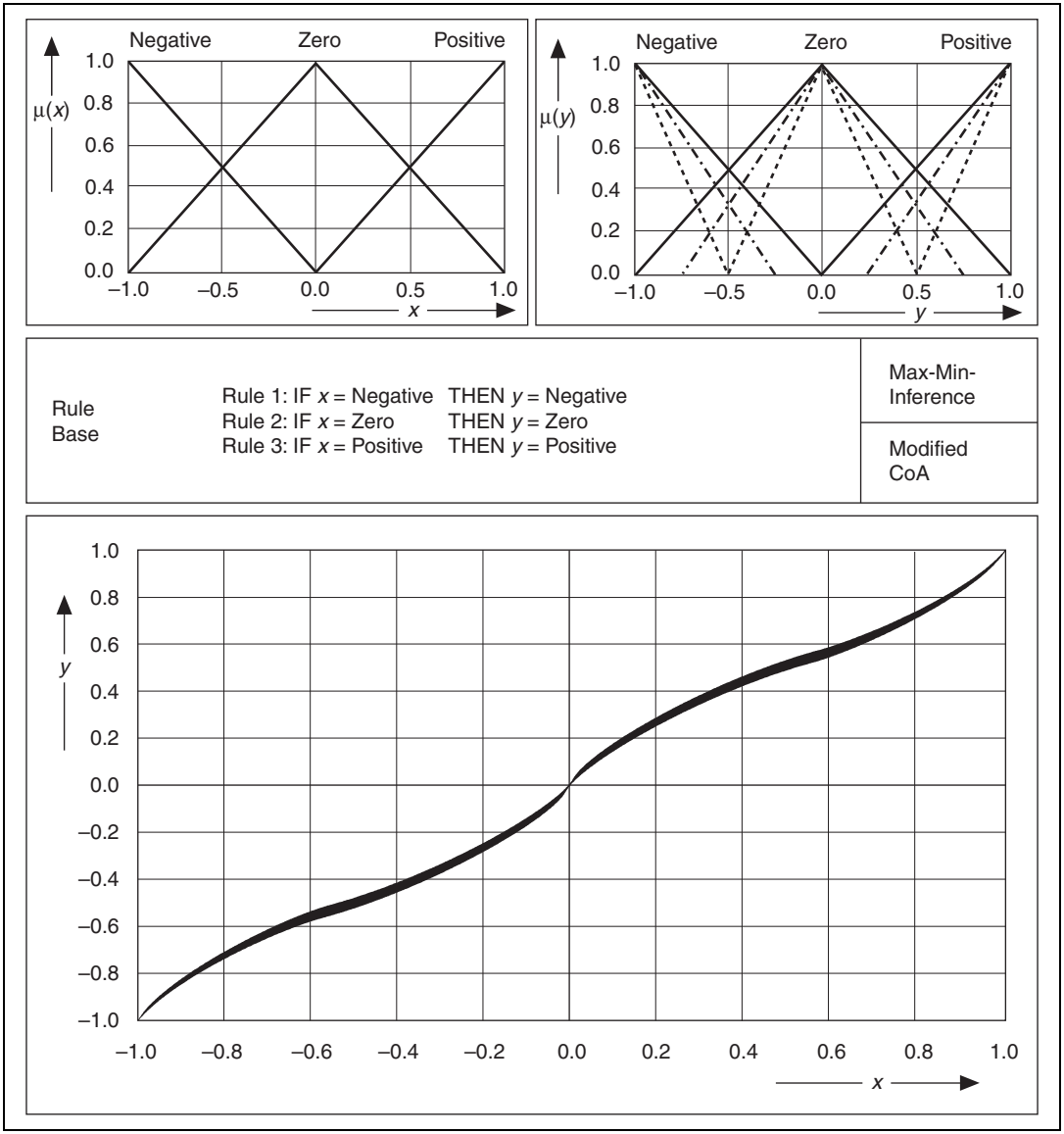

Figure 7-6. I/O Characteristics of a Fuzzy Controller (Different Overlapping Degrees of Membership Functions for the Output Terms)

If you want to significantly influence the controller characteristic, use output terms that membership functions model with equally distributed typical values but different scopes of influence instead. The different terms have different areas and thus different weights with respect to the defuzzification process. A wide output term has more influence on the inference result than a small neighboring output term. Figure 7-7 demonstrates this effect.

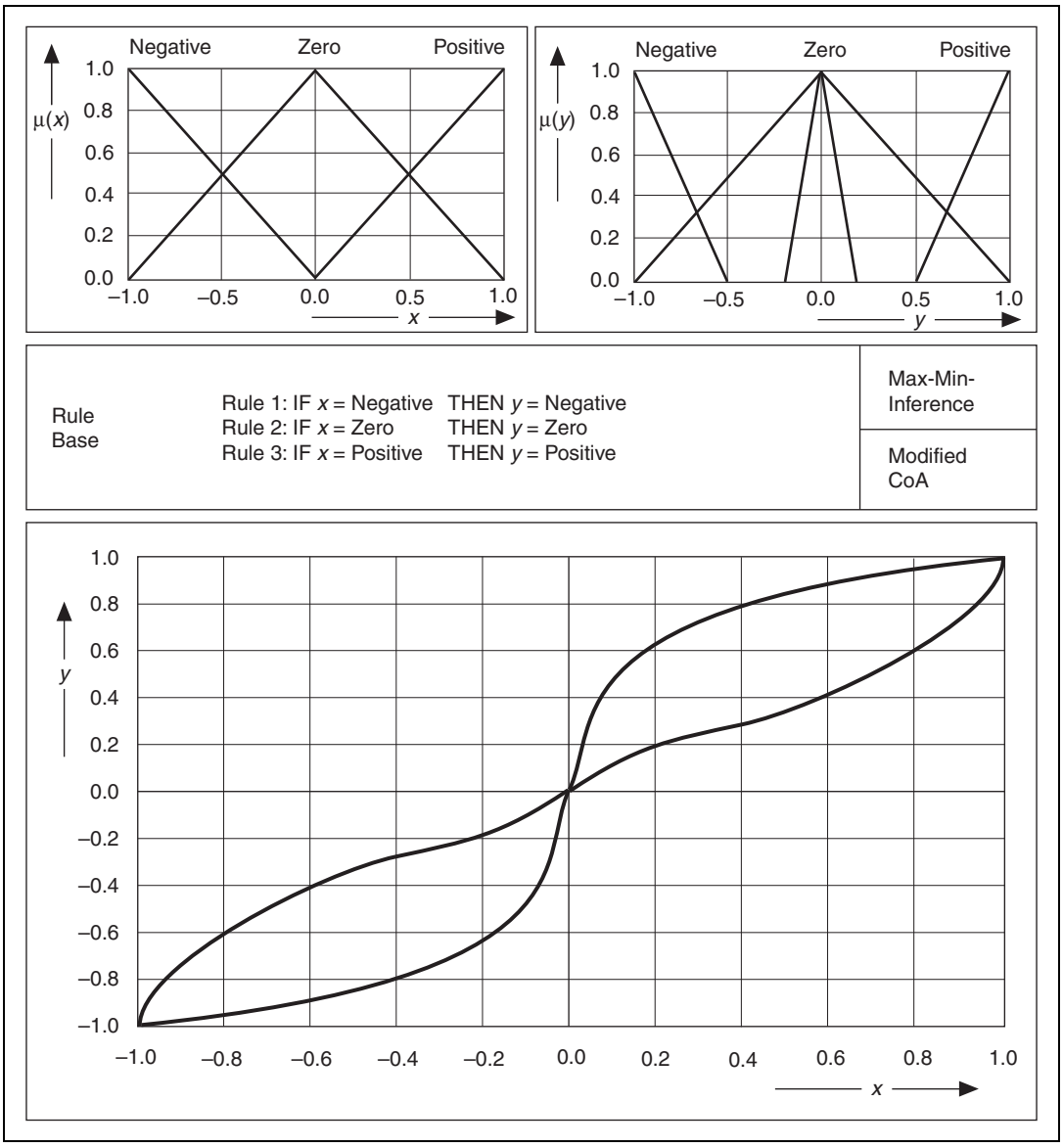

Figure 7-7. I/O Characteristics of a Fuzzy Controller (Wide and Small Membership Functions for the Output Terms)

Using CoA or CoM as the defuzzification method results in continuous controller characteristic functions, especially within those intervals of the input values in which two or more rules are active simultaneously. This behavior results from the averaging character of the defuzzification methods described in Chapter 6, *Defuzzification Methods*.

When you use the Mean of Maximum (MoM) defuzzification method, you calculate the most plausible result. In other words, the fuzzy controller uses the typical value of the consequent term of the most valid rule as the crisp output value. This behavior results in stepped output characteristics, as shown in Figure 7-8.

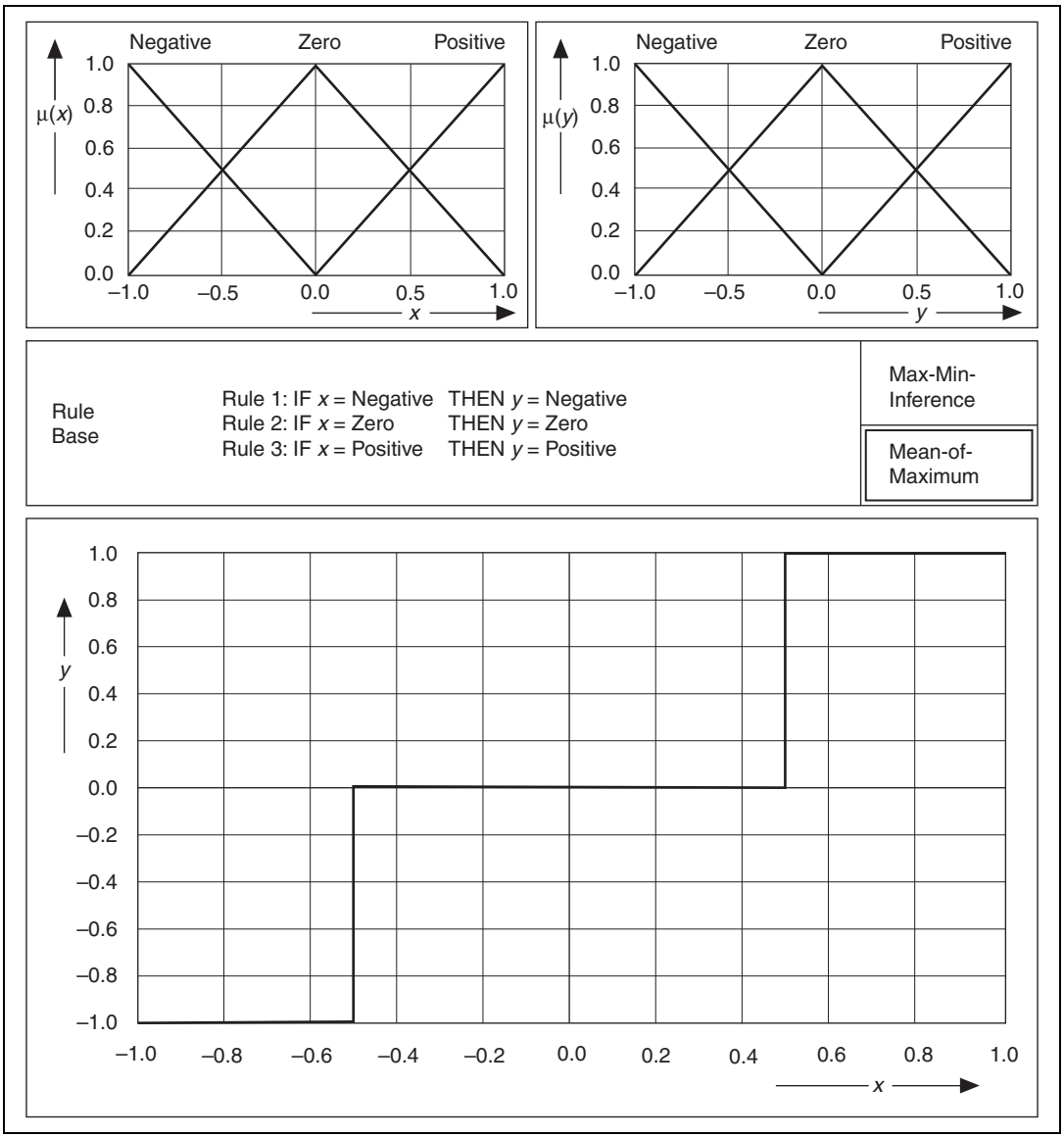

Figure 7-8. I/O Characteristic of a Fuzzy Controller with Mean-of-Maximum (Entirely Overlapping Membership Functions for Input and Output Terms)

The rule base itself has the biggest influence on the controller characteristic. The rule base determines the principal functionality of the controller.

Figure 7-9 illustrates how the controller characteristic changes if you change the rule base of the previous example to include the following rules:

Rule 1: IF x = negative THEN y = negative
Rule 2: IF x = zero THEN y = positive
Rule 3: IF x = positive THEN y = negative

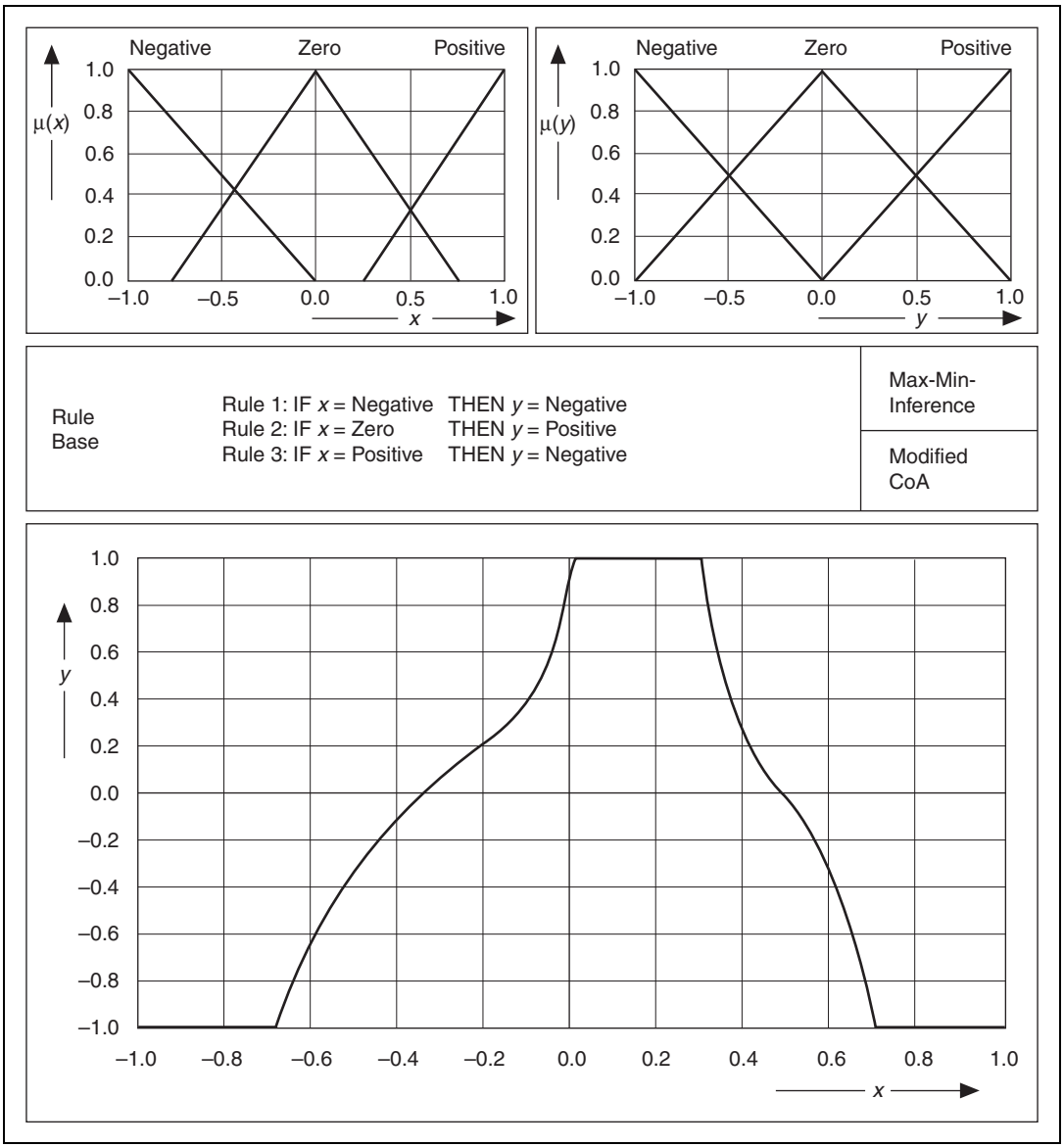

Figure 7-9. I/O Characteristic of a Fuzzy Controller with a Changed Rule Base

These examples show that you can use a fuzzy controller to perform arbitrary I/O operations. The number of input and output linguistic terms depends on the desired characteristic type and the precision to which you approximate the given I/O characteristic.

Consider, for example, the stepped linear characteristic curve shown in Figure 7-10. You can describe the four linear sections with the five circled base points (xi, yi).

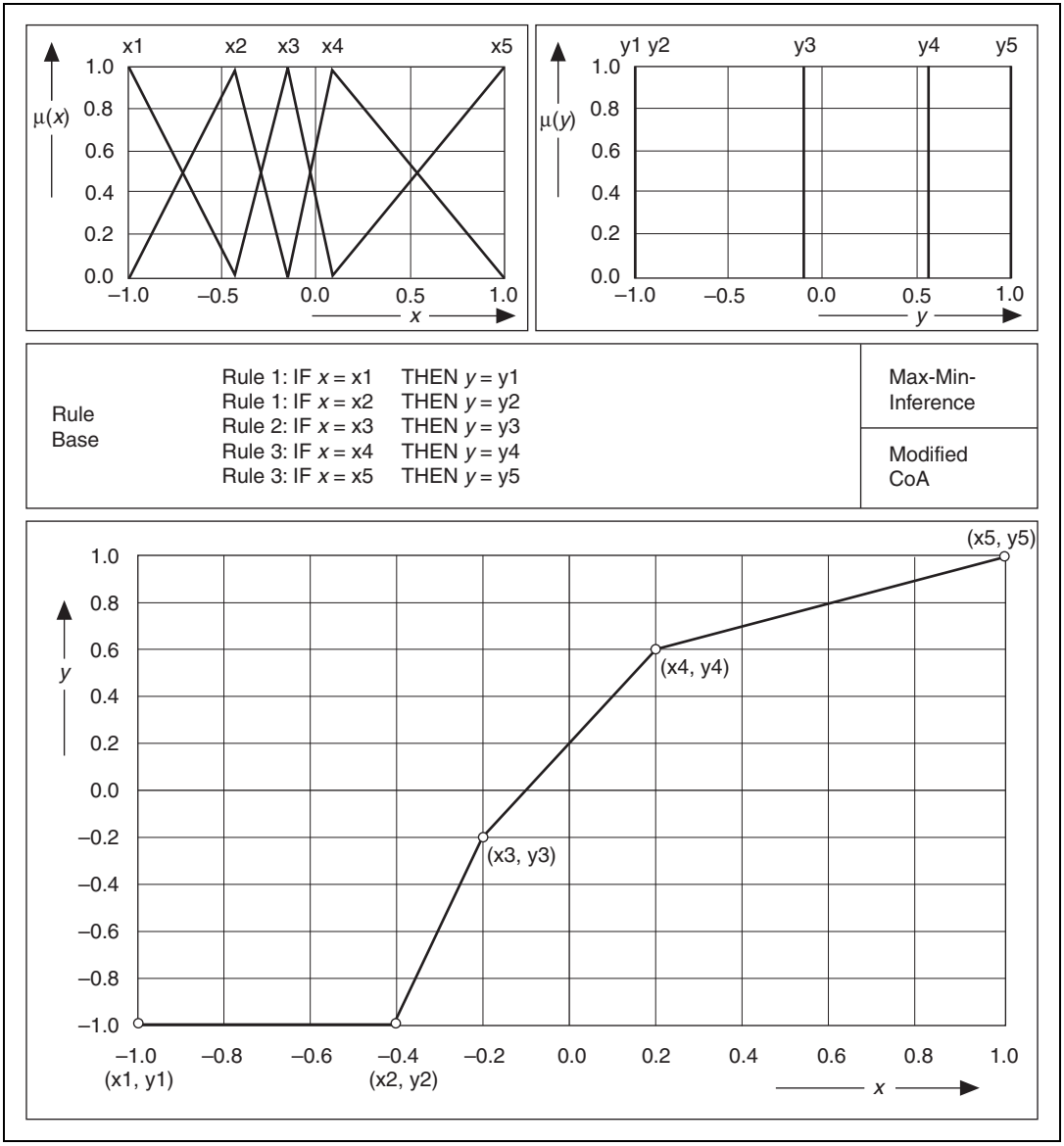

Figure 7-10. Fuzzy Controller for a Given I/O Characteristic

To use a single-input fuzzy controller to reproduce the given characteristic, use five linguistic terms each for the input and output quantities, naming them x1, x2, …, x5 and y1, y2, …, y5, respectively. To obtain the stepped linear sections between the base points, you always must have exactly two available active rules. To implement this situation, entirely overlap the triangular membership functions for the input variable, giving each a typical value that corresponds to a certain base point component, xi.

To obtain characteristic sections that are exactly linear, you must model the output variable with singleton membership functions, each of which has a typical value that corresponds to a certain base point component, yi. The rule base is then a linguistic enumeration of the five base points.

In principle, these conclusions about I/O characteristics are valid for fuzzy controllers with two or more inputs as well. However, using the AND (Minimum) antecedent connective to combine the different input conditions raises an additional nonlinear effect. With the AND (Minimum) antecedent connective, the fuzzy controller considers the antecedent of the rule with the lowest degree of truth. Refer to Figure 7-10 for an example. Figure 7-11 shows the I/O characteristic field for a dual-input fuzzy controller.

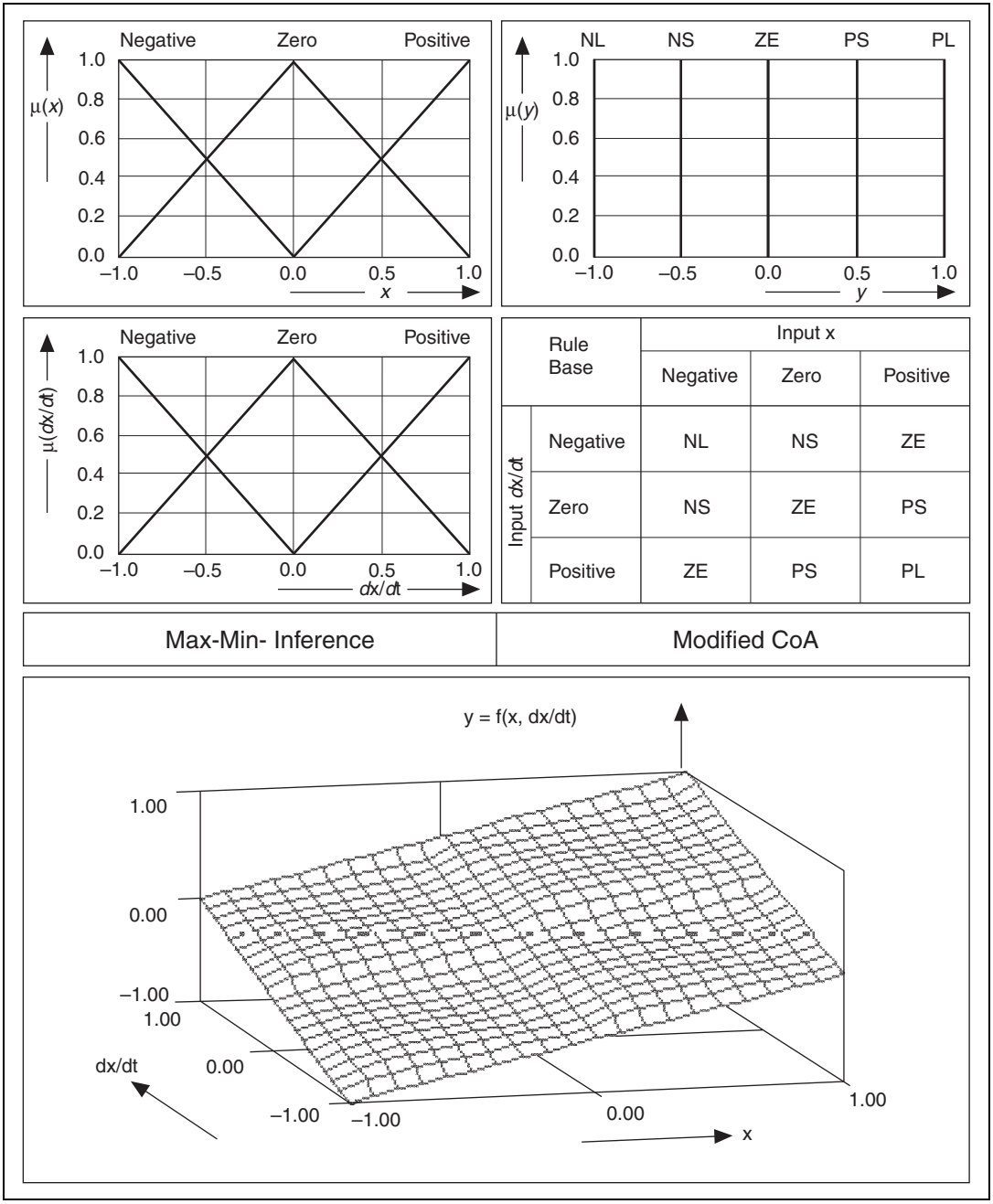

Figure 7-11. I/O Characteristic Field of a Dual-Input Fuzzy Controller

Because the AND (Minimum) antecedent connective is nonlinear, the characteristic field is not exactly linear despite the membership functions that overlap entirely for both input variables. Nonoverlapping membership functions yield a stepped characteristic field with constant planes, as shown in Figure 7-12.

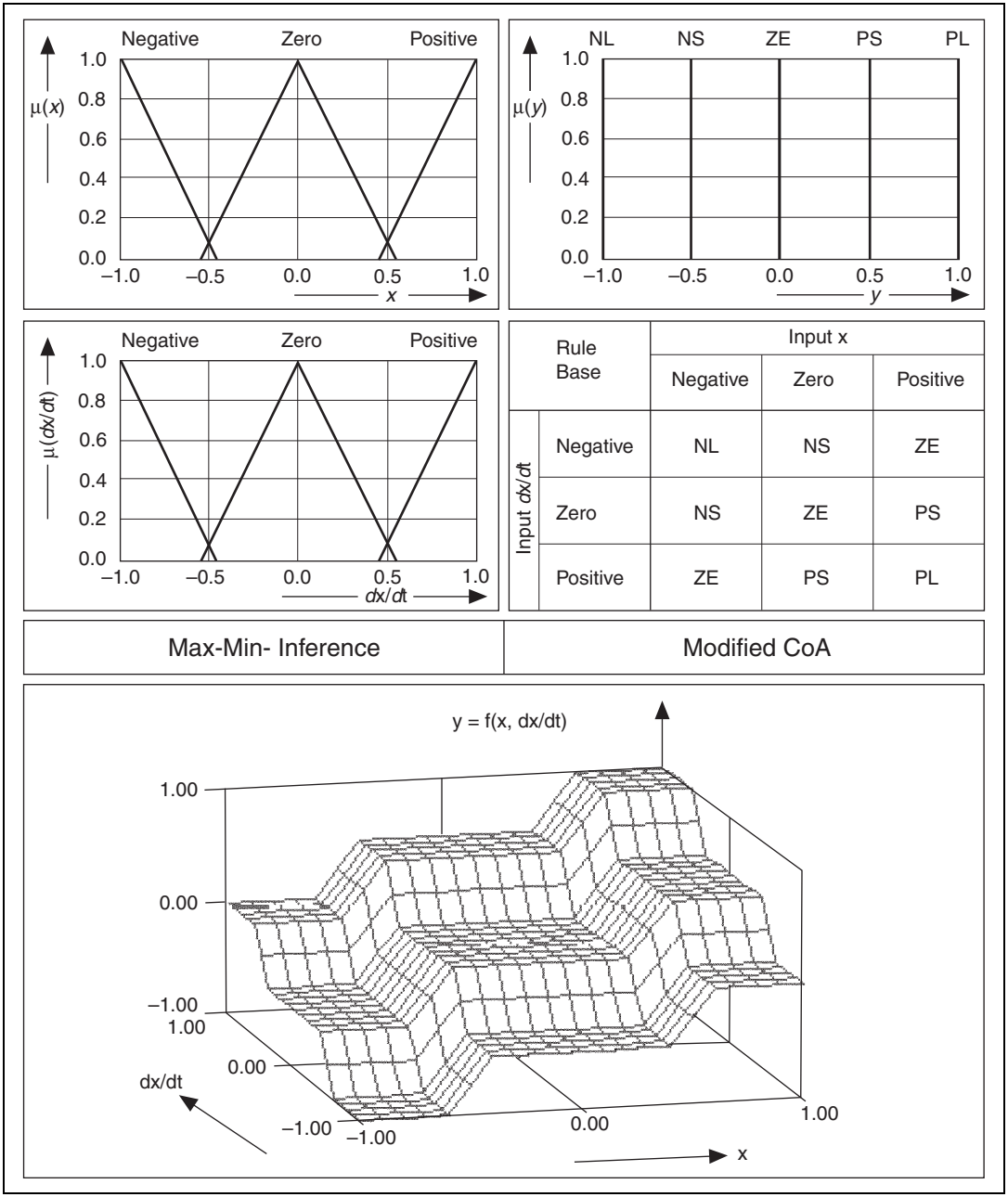

Figure 7-12. I/O Characteristic Field of a Dual-Input Fuzzy Controller (Slightly Overlapping Input Terms)

8

Closed-Loop Control Structures with Fuzzy Controllers

The most common use case for fuzzy controllers is in closed-loop control structures. The most basic structure of closed-loop control applications uses sensor signals as input signals for the system and the outputs as command values to drive the actuators of the process. A corresponding control loop structure is shown in Figure 8-1.

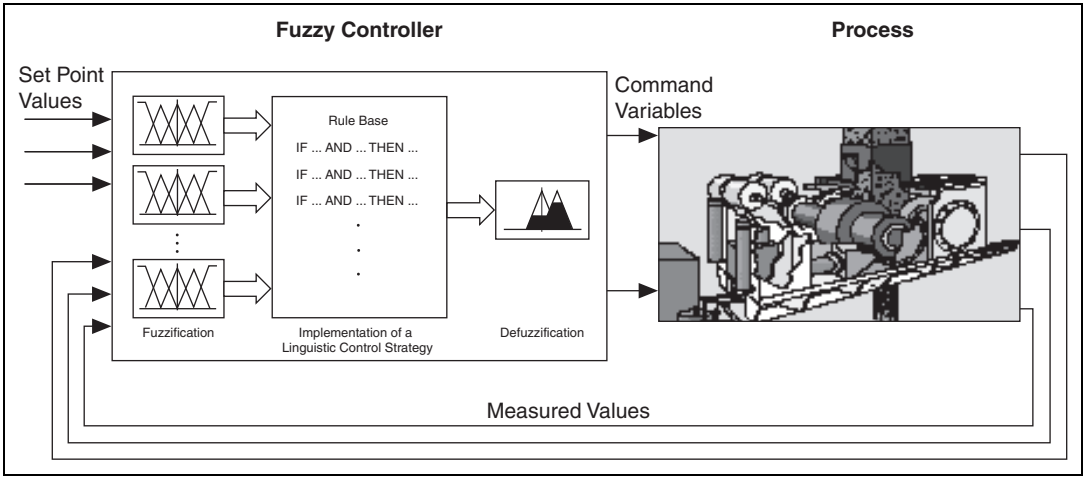

Figure 8-1. Simple Closed-Loop Control Structure with Fuzzy Controller

Pure fuzzy control applications are more the exception than the rule. In most cases the fuzzy controller outputs serve as reference parameters, such as gains, that you provide to a conventional controller instead of directly to driving actuators.

Because you can regard a fuzzy controller as a nonlinear characteristic field controller, it has no internal dynamic aspects. Thus, you must implement any dynamic property by an appropriate preprocessing of the measured input data.

A PI controller is a controller that produces proportional plus integral control action. The PI controller has only one input and one output. The output value increases when the input value increases. A fuzzy-PI controller is a generalization of the conventional PI controller that uses an error signal and its derivative as input signals. Fuzzy-PI controllers have two inputs and one output. Multiple inputs allow for greater control diversity for a fuzzy-PI controller over a conventional PI controller.

The fuzzy-PI controller, shown in Figure 8-2, uses the error signal $e(t)$ and its derivative $de(t)/dt$ from the measured data preprocessing step as inputs. If the output signal describes the necessary difference toward the current output value, you need a subsequent integrator device to build up the command variable value.

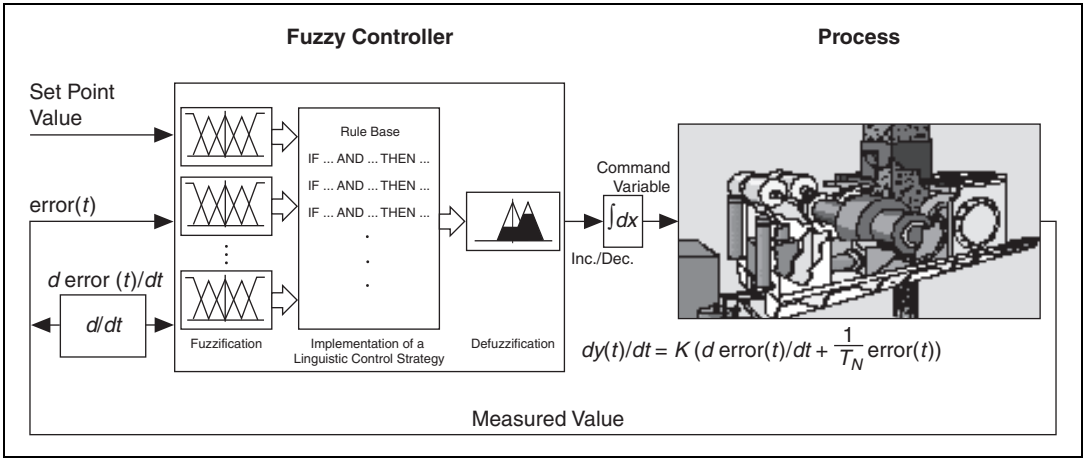

Figure 8-2. Closed-Loop Control Structure with Fuzzy-PI Controller

The benefit of the fuzzy-PI controller is that it does not have a special operating point. The rules evaluate the difference between the measured value and the set value, which is the error signal. The rules also evaluate the tendency of the error signal to determine whether to increase or decrease the control variable. The absolute value of the command variable has no influence.

Another advantage of a fuzzy-PI controller over a conventional PI controller is that it can implement nonlinear control strategies and that it uses linguistic rules. With fuzzy-PI controllers, you can consider the error tendency by itself when the error becomes small.

The chemical industries and process technologies often use fuzzy controllers with underlying PID control loops. The fuzzy controllers these industries use are PID fuzzy controllers that control single-process parameters. Usually, people supervise the operating point of the entire process.

Figure 8-3 shows the controller structure of the fuzzy controller with underlying PID control loops.

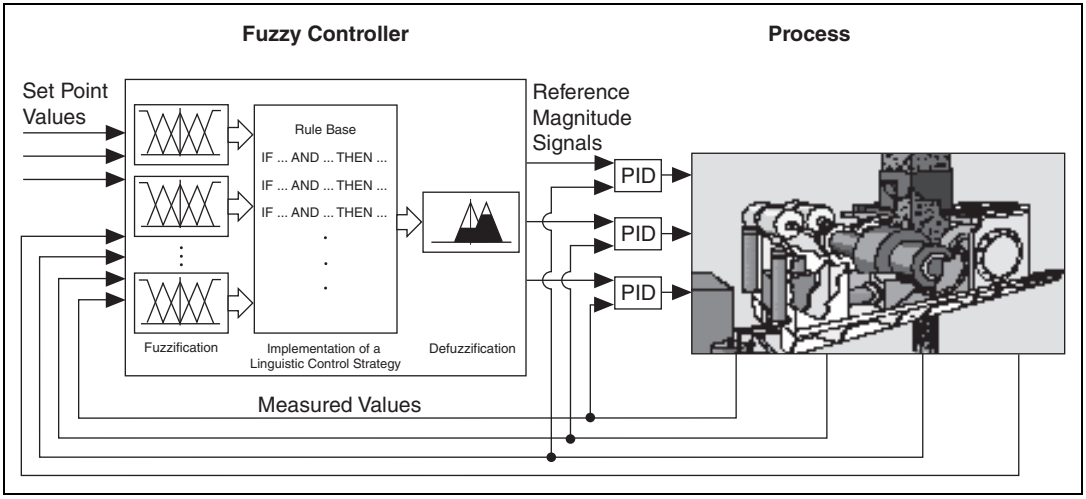

Figure 8-3. Fuzzy Controller with Underlying PID Control Loops

For automatic operation of such multivariable control problems, you must build a model-based controller. However, for most applications, either the process is too complex to model adequately, or the mathematical modeling task requires too much time.

The next example structure shows how to use a fuzzy controller to tune the parameters of a conventional PID controller automatically. This fuzzy controller constantly interprets the process reaction and calculates the optimal P, I, and D gains. You can apply this control structure to processes that change their characteristics over time. Figure 8-4 shows this control structure.

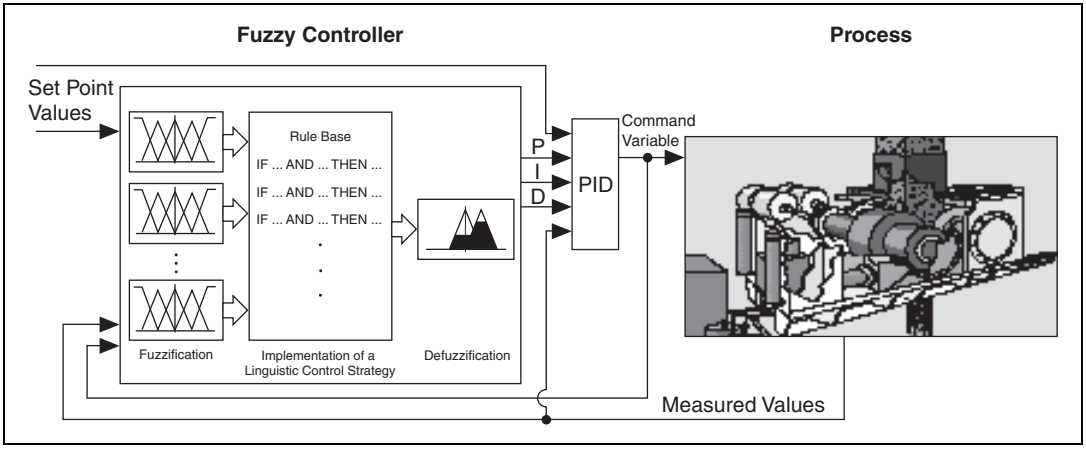

Figure 8-4. Fuzzy Controller for Parameter Adaptation of a PID Controller

In Figure 8-5, both the fuzzy controller and the PID controller work in parallel. The process adds the output signals from both controllers, but the output signal from the fuzzy controller is zero under normal operating conditions. The PID controller output leads the process. The fuzzy controller intervenes only when it detects abnormal operating conditions, such as strong disturbances.

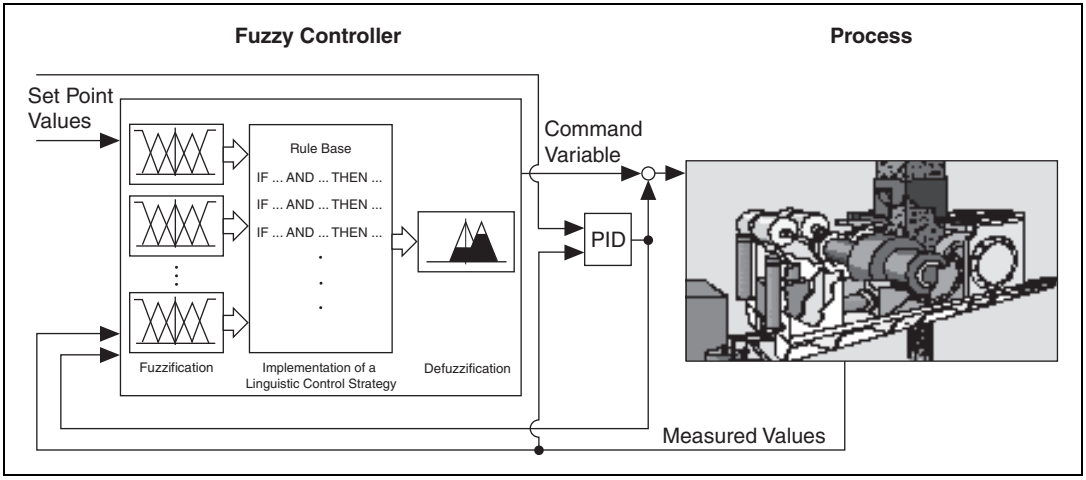

Figure 8-5. Fuzzy Controller for Correction of a PID Controller Output

You can use the LabVIEW PID and Fuzzy Logic Toolkit to design fuzzy systems and to implement the control structures described in this chapter for those fuzzy systems. Use either the Fuzzy System Designer or the Fuzzy Logic VIs to design a fuzzy system. Then use the FL Fuzzy Controller VI to implement a fuzzy controller for the fuzzy system. Finally, you can integrate the fuzzy controller into a control structure you create using the PID VIs.

Refer to Chapter 3, *Using the PID Software*, for more information about creating control structures using the PID VIs. Refer to Chapter 9, *Designing a Fuzzy System with the Fuzzy System Designer*, or Chapter 10, *Modifying a Fuzzy System with the Fuzzy Logic VIs*, for more information about designing fuzzy systems and implementing fuzzy controllers.

9

Designing a Fuzzy System with the Fuzzy System Designer

You can use the LabVIEW PID and Fuzzy Logic Toolkit to design a fuzzy system in one of two ways: using the Fuzzy System Designer or using the Fuzzy Logic VIs. This chapter describes how to use the Fuzzy System Designer to design a fuzzy system. Select **Tools»Control Design and Simulation»Fuzzy System Designer** to launch the Fuzzy System Designer. Refer to Chapter 10, *Modifying a Fuzzy System with the Fuzzy Logic VIs*, for information about designing a fuzzy system using the Fuzzy Logic VIs.

In this chapter, you use the Fuzzy System Designer to design a fuzzy system similar to the vehicle maneuvering example from Chapter 5, *Designing a Fuzzy System*. The `Truck - backward.fs` fuzzy system, located in the `labview\examples\control\fuzzy\Car Parking` directory, provides the complete fuzzy system that you design in this chapter.

Creating Linguistic Variables

The first step in designing a fuzzy system with the Fuzzy System Designer is to create the input and output linguistic variables for the system. Refer to the *Creating Linguistic Variables* section of Chapter 5, *Designing a Fuzzy System*, for more information about linguistic variables.

In the vehicle maneuvering example in Chapter 5, *Designing a Fuzzy System*, the fuzzy system has two linguistic input variables, *Vehicle Position x* and *Vehicle Orientation* β, and one linguistic output variable, *Steering Angle* φ. Figure 5-5, *Membership Functions for Vehicle Position x*, Figure 5-6, *Membership Functions for Vehicle Orientation β*, and Figure 5-7, *Membership Functions for Steering Angle φ*, illustrate the range and membership functions for each of these linguistic variables.

Complete the following steps to create an input linguistic variable corresponding to the *Vehicle Position x* input linguistic variable of the vehicle maneuvering example.

1. Select **Tools»Control Design and Simulation»Fuzzy System Designer** to launch the Fuzzy System Designer. The Fuzzy System Designer displays the **Variables** page by default.
2. Click the **Add Input Variable** button to the right of the **Input variables** list to launch the **Edit Variable** dialog box.
3. Enter `vehicle-position` in the **Name** text box.
4. Enter a **minimum** value of `0` and a **maximum** value of `10`. This range specifies that the position of the vehicle relative to the destination ranges from 0.0 to 10.0 meters.
5. Click the **Add Membership Function** button to create a new membership function for the *vehicle-position* linguistic variable.
6. Enter `left` in the **Name** text box.
7. Select **Trapezoid** from the **Shape** pull-down menu to specify that a trapezoid function determines the degrees of membership for the linguistic variable.
8. Select the color you want to use for the membership function in the **Membership functions graph** from the color picker.
9. Enter `0`, `0`, `1`, and `4` in the fields of the **Points** array. These points specify the values of the linguistic variable corresponding to the base and top points, in order from left to right and base to top, of the membership function.
10. Repeat steps 5 through 9 for each of the membership functions in Figure 5-5, *Membership Functions for Vehicle Position x*.

Note In the **Points** array for each membership function, ensure that each point is greater than or equal to all previous points.

Figure 9-1 displays the **Edit Variable** dialog box with all membership functions for the *vehicle-position* input variable.

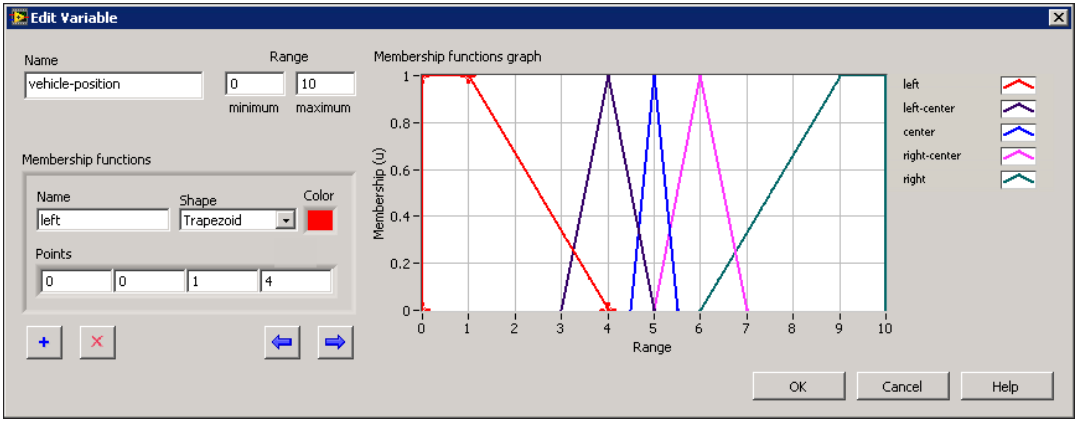

Figure 9-1. Membership Functions for the vehicle-position Input Variable

11. Click the **OK** button to save the changes. Notice that the *vehicle-position* input variable appears in the **Input variables** list and the corresponding membership functions appear in the **Input variable membership functions** graph.

You can repeat the previous procedure to create an input linguistic variable corresponding to the *Vehicle Orientation* β input linguistic variable of the vehicle maneuvering example. Figure 5-6, *Membership Functions for Vehicle Orientation β*, illustrates the range and membership functions for the *Vehicle Orientation β* input linguistic variable. For the purposes of this example, you can use approximate values for the points of the membership functions when creating this input linguistic variable. Figure 9-2 displays the **Edit Variable** dialog box with all membership functions for the *vehicle-orientation* input variable.

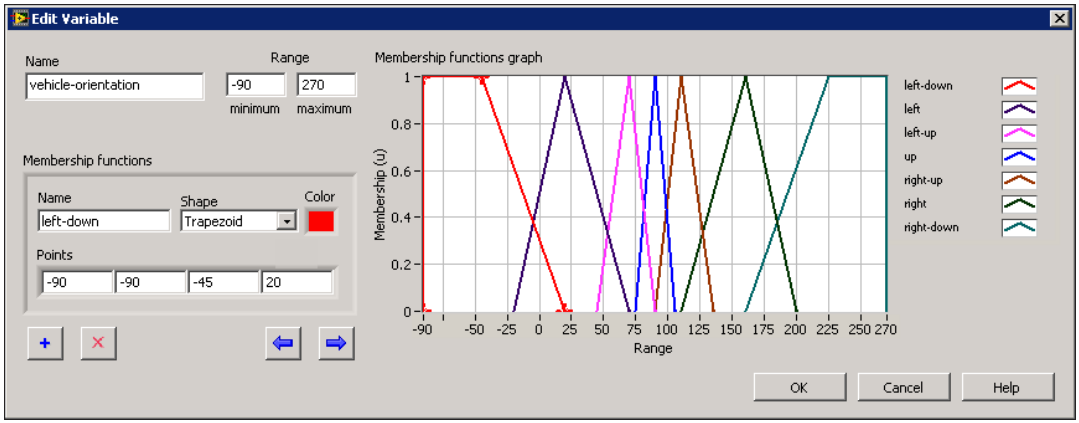

Figure 9-2. Membership Functions for the vehicle-orientation Input Variable

You create output linguistic variables similarly to how you create input linguistic variables in the Fuzzy System Designer. Complete the following steps to create an output linguistic variable corresponding to the *Steering Angle* φ output linguistic variable of the vehicle maneuvering example.

1. Click the **Add Output Variable** button to the right of the **Output variables** list to launch the **Edit Variable** dialog box.
2. Enter `steering-angle` in the **Name** text box.
3. Enter a **minimum** value of `-30` and a **maximum** value of `30`. This range specifies that the amount that the steering wheel can turn ranges from –30.0 to 30.0 degrees.
4. Click the **Add Membership Function** button to create a new membership function for the *steering-angle* output variable.
5. Enter `NegBig` in the **Name** text box.
6. Select **Triangle** from the **Shape** pull-down menu to specify that a triangle function determines the degrees of membership for the linguistic variable.
7. Select the color you want to use for the membership function in the **Membership functions graph** from the color picker.
8. Enter `-30`, `-30`, and `15` in the fields of the **Points** array. These points specify the values of the linguistic variable corresponding to the base and top points, in order from left to right and base to top, of the membership function.
9. Repeat steps 4 through 8 to create each of the membership functions in Figure 5-7, *Membership Functions for Steering Angle φ*.

Figure 9-3 displays the **Edit Variable** dialog box with all membership functions for the *steering-angle* output variable.

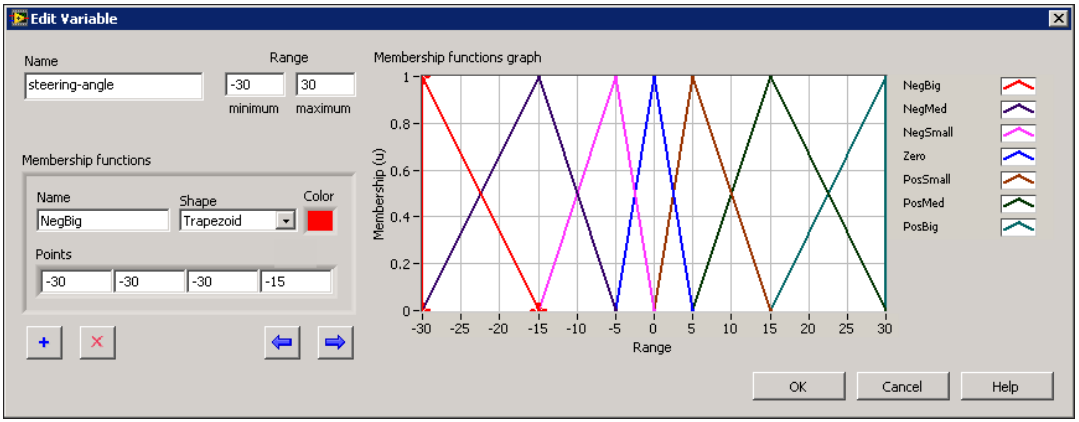

Figure 9-3. Membership Functions for the steering-angle Output Variable

10. Click the **OK** button to save the changes.
11. Select **File»Save As** to save the fuzzy system as `Vehicle Maneuvering.fs` in an easily accessible location.

After you define the input and output linguistic variables for the fuzzy system, you use the linguistic variables to create a rule base for the fuzzy system.

Creating a Rule Base

Rules describe, in words, the relationships between input and output linguistic variables based on their linguistic terms. The rule base of a fuzzy system determines the output values of the fuzzy system based on the input values. Refer to the *Creating a Rule Base* section of Chapter 5, *Designing a Fuzzy System*, for more information about rule bases.

Recall that the *vehicle-position* input linguistic variable you created in the *Creating Linguistic Variables* section of this chapter has five membership functions, and the *vehicle-orientation* input linguistic variable has seven membership functions. Therefore, you can construct rules to associate 35 possible combinations of the linguistic terms of these input linguistic variables with the linguistic terms of the *steering-angle* output linguistic variable. Refer to Figure 5-8, *Complete Rule Base for the Vehicle Maneuvering Example*, to view this complete rule base.

Complete the following steps to create the complete rule base for the vehicle maneuvering example using the Fuzzy System Designer. You must have created both input linguistic variables and the output linguistic variable in order to create this rule base.

1. Click the **Rules** tab of the Fuzzy System Designer to display the **Rules** page.
2. Select **Operate»Pre-Generate Rules** to display the **Pre-Generate Rules** dialog box.
3. Select **AND (Minimum)** from the **Antecedent connective** pull-down menu. This option specifies that the smallest degree of membership of the antecedents determines the truth value of the aggregated rule antecedent for each rule.
4. Enter a **Degree of support** of `1`. This option specifies the weight that you want to apply to each rule. The final rule weight for each rule is equal to the **Degree of support** multiplied by the truth value of the aggregated rule antecedent.
5. Select **Minimum** from the **Consequent implication** pull-down menu. This option specifies that the fuzzy logic controller truncates the output membership functions at the value of the corresponding rule weights before performing defuzzification.
6. Click the **OK** button to populate the **Rules** list with all 35 combinations of linguistic terms of the input variables. Each rule uses the same antecedent connective, degree of support, and implication method that you specified in the **Pre-Generate Rules** dialog box.

Notice that the consequent of each rule in the **Rules** list is "*THEN 'steering-angle' IS 'NegBig'*." However, as Figure 5-8, *Complete Rule Base for the Vehicle Maneuvering Example*, illustrates, the linguistic term of the *steering-angle* output variable depends on the combination of linguistic terms of the input variables.

Complete the following steps to modify the pre-generated rules to use the correct consequents.

1. Select **Rule 1** in the **Rules** list.
2. In the **THEN** column for **Rule 1**, select **NegSmall** as the linguistic term for the *steering-angle* output linguistic variable. Notice the consequent of Rule 1 changes to "*THEN 'steering-angle' IS 'NegSmall'*."

3. Repeat steps 1 and 2 for each rule using the consequents in Figure 5-8, *Complete Rule Base for the Vehicle Maneuvering Example*.
4. Select **File»Save** to save the fuzzy system.

The fuzzy system now has a complete linguistic rule base with which to analyze input data and produce output data. However, the rule base returns the output data as linguistic terms. The fuzzy controller must defuzzify the output data that the rule base specifies before the output can apply to the control structure the fuzzy system automates. You therefore must specify a defuzzification method for the fuzzy controller to use.

Specifying a Defuzzification Method

After you create the rule base for a fuzzy system, you must specify how a fuzzy controller performs defuzzification for the system. Defuzzification is the process of converting the degrees of membership of output linguistic variables into numerical values.

In the vehicle maneuvering example, you must supply a continuous output signal to control the steering angle of the vehicle. Therefore, you must use a defuzzification method that calculates the best compromise between any rules that apply at a given time. According to the guidelines in Table 6-1, *Comparison of Different Defuzzification Methods*, you can choose either the Center of Maximum, Center of Area, or Center of Sums defuzzification method.

On the **Rules** page of the Fuzzy System Designer, select either of these defuzzification methods from the **Defuzzification method** pull-down menu. Then select **File»Save** to save the fuzzy system.

The fuzzy system is complete. Before you integrate the fuzzy system into the control structure you want to automate, however, you might want to test the functionality of the system.

Testing the Fuzzy System

You can test the relationship between the input and output values of a fuzzy system to validate the rule base of the fuzzy system. Use the **Test System** page of the Fuzzy System Designer to test the fuzzy system you created in the previous sections of this chapter.

Complete the following steps to test the vehicle maneuvering fuzzy system.

1. Click the **Test System** tab of the Fuzzy System Designer to display the **Test System** page.

2. Enter an **Input value** of `5` for the *vehicle-position* input linguistic variable. Recall from the *Creating Linguistic Variables* section of this chapter that a value of 5 for the *vehicle-position* input linguistic variable corresponds to the *center* linguistic term.
3. Enter an **Input value** of `–30` for the *vehicle-orientation* input linguistic variable. Recall from the *Creating Linguistic Variables* section of this chapter that a value of –30 for the *vehicle-orientation* input linguistic variable corresponds to the *left-down* linguistic term.

   In the **Invoked Rules** table, notice that the fuzzy system invokes Rule 15: *IF 'vehicle-position' IS 'center' AND 'vehicle-orientation' IS 'left-down' THEN 'steering-angle' IS 'NegMed'*

   Recall from the *Creating Linguistic Variables* section of this chapter that the *NegMed* linguistic term for the *steering-angle* output linguistic variable corresponds to the range between –30 and –5. Notice that the **Output value** of the *steering-angle* output variable is **`-16.7334`**, which is within this range.

Note If you used approximate values for the points of the membership functions of the *steering-angle* output variable, you might see a slightly different **Output value**.

The fuzzy controller calculates the **Output value** based on the weight of the rule, the consequent implication method, and the defuzzification method.

You also can use the **Input value 1** and **Input value 2** slide controls to sweep a range of values for the input linguistic variables of the fuzzy system. You then can observe the corresponding change in the **Input/Output relationship** graph. You can use the **Input/Output relationship** graph to verify that the rule base is reasonable and complete. For example, if the **Input/Output relationship** graph is 0 at some points, the rule base might be incomplete.

Controlling the Fuzzy System

After you create a fuzzy system in the Fuzzy System Designer, you can use the FL Fuzzy Controller VI to implement a controller for the fuzzy system. The FuzzyEx Car Backward Parking example VI, located in the `labview\examples\control\fuzzy\Car Parking` directory, illustrates how to implement a controller for a fuzzy system.

The block diagram of the FuzzyEx Car Backward Parking example VI appears similar to the following figure.

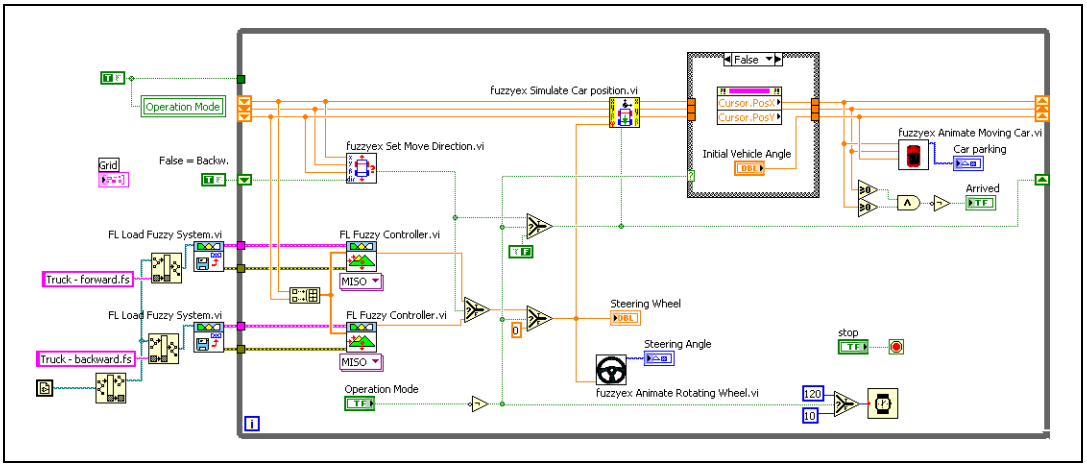

Figure 9-4. Block Diagram of the FuzzyEx Car Backward Parking Example VI

This example VI uses the FL Load Fuzzy System VI to load both the `Truck - forward.fs` and `Truck - backward.fs files`. These files represent the fuzzy systems for moving the vehicle forward and backwards, respectively.

Each FL Load Fuzzy System VI passes a `.fs` file to a FL Fuzzy Controller VI, which implements a fuzzy controller for the fuzzy system.

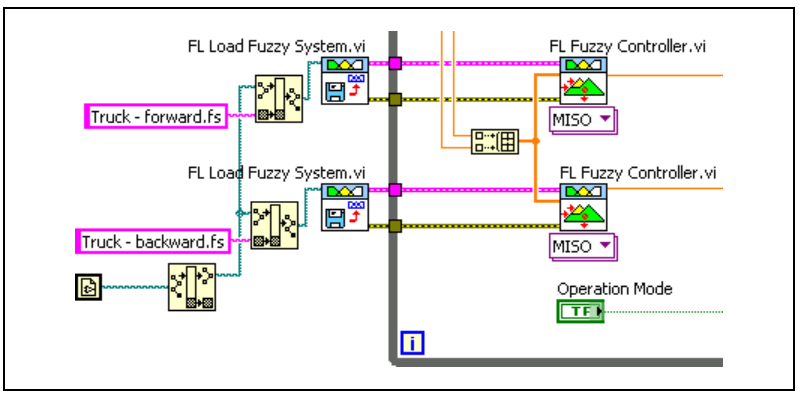

Figure 9-5. Detail of the FuzzyEx Car Backward Parking Example VI

The fuzzy system you created with the Fuzzy System Designer in the previous sections of this chapter is similar to the `Truck - backward.fs` file. You can implement a fuzzy controller for the fuzzy system you created in the same fashion that the FuzzyEx Car Backward Parking example VI implements a controller for the `Truck - backward.fs` file.

10

Modifying a Fuzzy System with the Fuzzy Logic VIs

In Chapter 9, *Designing a Fuzzy System with the Fuzzy System Designer*, you design a fuzzy system interactively using the Fuzzy System Designer and then implement a fuzzy controller for that system using the FL Fuzzy Controller VI. You also can design a fuzzy system programmatically using the Fuzzy Logic VIs. However, in most cases, use the Fuzzy System Designer to design the fuzzy system first, and then use the Fuzzy Logic VIs to make programmatic modifications to the fuzzy system. In this chapter, you modify a fuzzy system programmatically using the Fuzzy Logic VIs.

This chapter uses the example of a greenhouse control system to illustrate how to modify a fuzzy system using the Fuzzy Logic VIs. The `greenhouse.fs` file, located in the `labview\examples\control\fuzzy\Dynamic greenhouse controller` directory, represents the fuzzy system that you modify in this chapter.

Observing the Fuzzy System

Before modifying the greenhouse fuzzy system, first open and observe the fuzzy system in the Fuzzy System Designer.

Complete the following steps to open and observe the greenhouse fuzzy system in the Fuzzy System Designer.

1. Select **Tools»Control Design and Simulation»Fuzzy System Designer** to launch the Fuzzy System Designer.
2. Select **File»Open**.
3. In the file dialog box that appears, navigate to the `labview\examples\control\fuzzy\Dynamic greenhouse controller` directory and select the `greenhouse.fs` file.
4. Click the **OK** button. The greenhouse fuzzy system appears in the Fuzzy System Designer.

   On the **Variables** page, notice that the **Input variables** and **Output variables** lists display the input and output linguistic variables,

respectively, for the fuzzy system. The greenhouse fuzzy system has two input linguistic variables, *Temperature* and *Humidity*, and two output linguistic variables, *Electric Roof* and *Water Spills*. The *Temperature* input linguistic variable represents the temperature, in degrees Celsius, inside the greenhouse.

By default, the **Input variable membership functions** graph and the **Output variable membership functions** graph display information about the *Temperature* input linguistic variable and the *Electric Roof* output linguistic variable, respectively. Notice that the *Temperature* input linguistic variable has a range from 0 to 50 and three membership functions: *Cold*, *Normal*, and *Warm*.

Notice that a trapezoid function represents the shape of the *Cold* membership function. The left base, left top, right top, and right base points of this membership function are 0, 0, 13, and 21, respectively.

5. Click the **Rules** tab of the Fuzzy System Designer to display the **Rules** page. Notice that the **Rules** list displays nine rules for the fuzzy system. This rule base corresponds to all possible combinations of the membership functions of the two input linguistic variables.

   Also notice that the **Defuzzification method** for the fuzzy system is **Center of Area**. Refer to Chapter 6, *Defuzzification Methods*, for information about defuzzification methods.

6. Click the first rule in the **Rules** list: *IF 'Temperature' IS 'Cold' AND 'Humidity' IS 'Dry' THEN 'Electric Roof' IS 'Closed' ALSO 'Water Spills' IS 'Moderate'*. Notice that the **Antecedent connective**, **Degree of support**, and **Consequent implication** for this rule are **AND (Minimum)**, **1**, and **Product**, respectively. Refer to the *Creating a Rule Base* section of Chapter 5, *Designing a Fuzzy System*, for information about these rule characteristics.

7. Click the **Close** button to close the Fuzzy System Designer.

The following sections describe how to modify the greenhouse fuzzy system using the Fuzzy Logic VIs.

Loading the Fuzzy System

Before modifying the greenhouse fuzzy system, you must load the `.fs` file. Complete the following steps to load the `greenhouse.fs` file.

1. Select **File»New VI** to create a new VI.
2. Press the <Ctrl-E> keys to display the block diagram.
3. Add an FL Load Fuzzy System VI to the block diagram.

4. Right-click the **file path** input of the FL Load Fuzzy System VI and select **Create»Constant** from the shortcut menu.
5. Enter the absolute path to the `labview\examples\control\fuzzy\Dynamic greenhouse controller\greenhouse.fs` file in the path constant.

Note You also can use the File I/O VIs and functions to create a relative path to wire to the **file path** input of the FL Load Fuzzy System VI.

6. Select **File»Save** and save the VI as `Modified Greenhouse Fuzzy System.vi` in an easily accessible location.

Modifying a Linguistic Variable

You can use the Variables VIs to modify the linguistic variables in a fuzzy system. In this section, you modify the range of the *Temperature* input linguistic variable of the greenhouse fuzzy system to use degrees Fahrenheit instead of degrees Celsius.

Complete the following steps to modify the range of the *Temperature* input linguistic variable.

1. Add an FL Set Variable VI to the block diagram. This VI modifies the name, range, or membership functions of a linguistic variable in a fuzzy system.
2. Wire the **fuzzy system out** output of the FL Load Fuzzy System VI to the **fuzzy system in** input of the FL Set Variable VI.
3. Right-click the **input/output** input of the FL Set Variable VI and select **Create»Constant** from the shortcut menu. This input specifies whether you want to modify an input or output linguistic variable.
4. Select **Input** from the **input/output** constant.
5. Right-click the **variable index** input of the FL Set Variable VI and select **Create»Constant** from the shortcut menu. This input specifies the index of the linguistic variable whose information you want to modify. The index is zero-based and corresponds to the order in which the linguistic variable was created.
6. Enter `0` in the **variable index** constant to specify the first linguistic variable. Recall that the *Temperature* variable is the first input linguistic variable in the fuzzy system.

7. Create constants for the **minimum** and **maximum** inputs of the FL Set Variable VI and set them to `32` and `122`, respectively. This new range represents the possible temperature, in degrees Fahrenheit, inside the greenhouse.
8. Save the Modified Greenhouse Fuzzy System VI. The block diagram should appear similar to Figure 10-1.

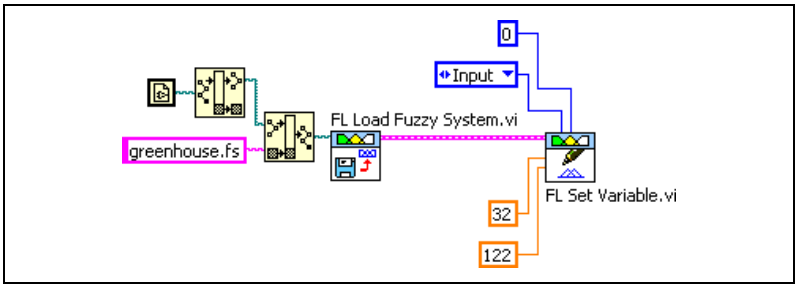

Figure 10-1. Modifying the Range of a Linguistic Variable

Modifying Membership Functions

You can use the Membership VIs to modify the membership functions for linguistic variables in a fuzzy system. In this section, you modify the membership functions of the *Temperature* input linguistic variable to use degrees Fahrenheit instead of degrees Celsius.

Complete the following steps to modify the membership functions of the *Temperature* input linguistic variable.

1. Add an FL Set Membership Function VI to the block diagram.
2. Wire the **fuzzy system out** output of the FL Set Variable VI to the **fuzzy system in** input of the FL Set Membership Function VI.
3. Create constants for the **membership function index**, **variable index**, and **input/output** inputs of the FL Set Membership Function VI and set them to `0`, `0`, and **Input**, respectively. These inputs specify that you want to modify the first membership function, *Cold*, of the *Temperature* input linguistic variable.
4. Create a constant for the **shape** input of the FL Set Membership Function VI. This input specifies the shape of the function that determines the degrees of membership for the linguistic variable.
5. Select **Trapezoid** from the **shape** constant.
6. Create a constant for the **points** input of the FL Set Membership Function VI. This input specifies the values of the linguistic variable

corresponding to the base and top points, in order from left to right and base to top, of the membership function.

7. In the **points** array constant, enter the values `32`, `32`, `55`, and `70`.
8. Repeat steps 1 through 5 for the *Normal* and *Warm* membership functions of the *Temperature* input linguistic variable. Wire the **fuzzy system out** output of each FL Set Membership Function VI to the **fuzzy system in** input of the next FL Set Membership Function VI. Use a Gaussian shape with points 59, 68, 77, and 86 for the *Normal* membership function. Use a Trapezoid shape with points 75, 90, 122, and 122 for the *Warm* membership function.
9. Save the Modified Greenhouse Fuzzy System VI. The block diagram should appear similar to Figure 10-2.

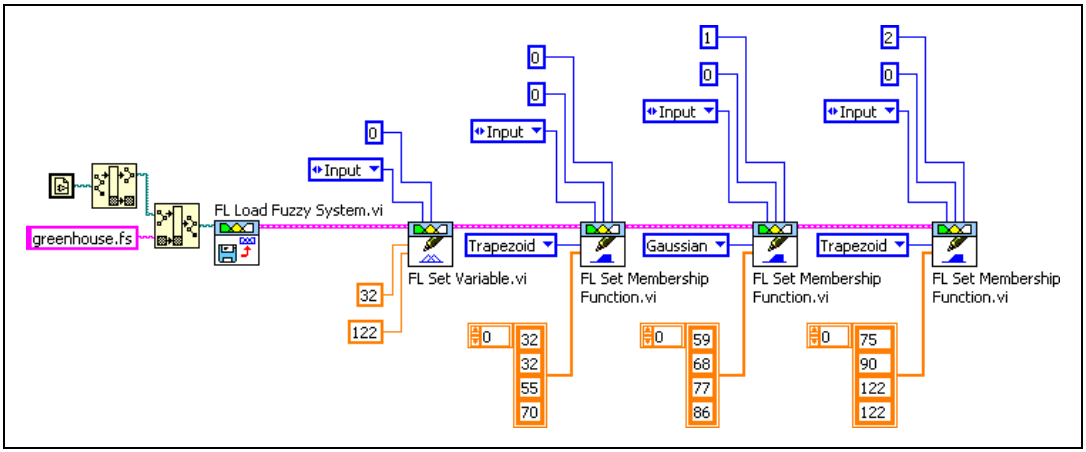

Figure 10-2. Modifying the Membership Functions of a Linguistic Variable

Modifying a Rule

You can use the Rules VIs to modify the rules for a fuzzy system. This section demonstrates how to recreate the first rule of the greenhouse fuzzy system, *IF 'Temperature' IS 'Cold' AND 'Humidity' IS 'Dry' THEN 'Electric Roof' IS 'Closed' ALSO 'Water Spills' IS 'Moderate'*, and replace the existing first rule of the fuzzy system with the rule you create.

When you use the Rules VIs to create rules, you must create the rules in parts. First, create the antecedents and consequents separately. Then combine the antecedents and consequents for a specific rule.

Creating the Antecedents

The first step in creating a rule is to create the antecedents of the rule. Complete the following steps to create the *Temperature' IS 'Cold'* and *'Humidity' IS 'Dry'* antecedents.

1. Add an FL Create Antecedent VI to the block diagram. This VI creates an antecedent, or IF portion, of a rule for a fuzzy system.
2. Create constants for the **variable index** and **membership function index** inputs of the FL Create Antecedent VI and set them both to `0`. These inputs specify that you want to associate the *Temperature* input linguistic variable with the *Cold* membership function.
3. Create a constant for the **condition** input of the FL Create Antecedent VI. This input specifies, when =, to calculate μ, the degree of membership of the linguistic variable within the membership function. Otherwise, this input specifies to calculate the degree of non-membership, or 1 – μ.
4. Select = from the **condition** constant. The FL Create Antecedent VI now creates the *'Temperature' IS 'Cold'* antecedent.
5. Repeat steps 1 through 4 to create the *'Humidity' IS 'Dry'* antecedent. In the greenhouse fuzzy system, the *Dry* membership function is the first membership function for the *Humidity* input linguistic variable, so use `0` as the **membership function index**.
6. Add a Build Array function to the block diagram.
7. Resize the Build Array function to display two input elements.
8. Wire the **antecedent** output of the first FL Create Antecedent VI to the first element of the Build Array function.
9. Wire the **antecedent** output of the second FL Create Antecedent VI to the second element of the Build Array function.
10. Save the Modified Greenhouse Fuzzy System VI. The relevant portion of the block diagram should appear similar to Figure 10-3.

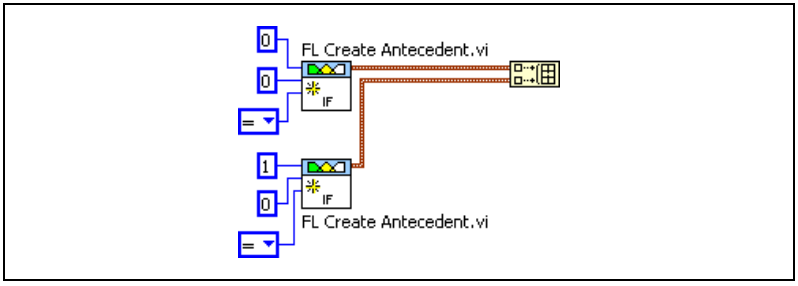

Figure 10-3. Creating Rule Antecedents

Creating Consequents

You create rule consequents similarly to how you create rule antecedents. Complete the following steps to create the *'Electric Roof' IS 'Closed'* and *'Water Spills' IS 'Moderate'* consequents.

1. Add an FL Create Consequent VI to the block diagram. This VI creates a consequent, or THEN portion, of a rule for a fuzzy system.
2. Create constants for the **variable index** and **membership function index** inputs of the FL Create Consequent VI and set them both to `0`. These inputs specify that you want to associate the *Electric Roof* output linguistic variable with the *Closed* membership function.

Note Unlike the FL Create Antecedent VI, the FL Create Consequent VI does not have a **condition** input. The FL Create Consequent VI always returns the consequent corresponding to the degree of membership of the output variable within the membership function.

3. Repeat steps 1 and 2 to create the *'Water Spills' IS 'Moderate'* consequent. In the greenhouse fuzzy system, the *Moderate* membership function is the second membership function for the *Water Spills* output linguistic variable, so use `1` as the **membership function index**.
4. Add a Build Array function to the block diagram.
5. Resize the Build Array function to display two input elements.
6. Wire the **consequent** output of the first FL Create Consequent VI to the first element of the Build Array function.
7. Wire the **consequent** output of the second FL Create Consequent VI to the second element of the Build Array function.
8. Save the Modified Greenhouse Fuzzy System VI. The relevant portion of the block diagram should appear similar to Figure 10-4.

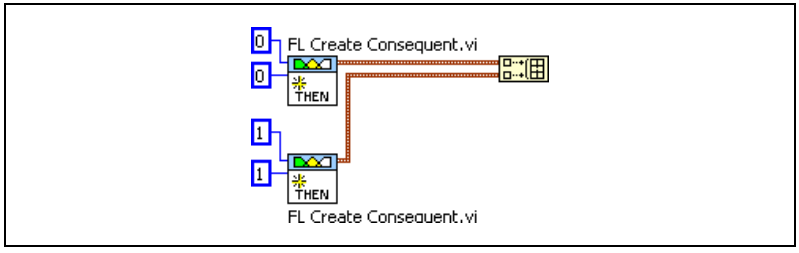

Figure 10-4. Creating Rule Consequents

Combining the Antecedents and Consequents for a Rule

After you create the antecedents and consequents of a rule, you must combine them to form a specific rule. Complete the following steps to combine the antecedents and consequents you created in the previous sections.

1. Add an FL Set Rule VI to the block diagram. This VI modifies the antecedents, consequents, or relationships of a rule in a fuzzy system.
2. Wire the **fuzzy system out** output of the last FL Set Membership Function VI to the **fuzzy system in** input of the FL Set Rule VI.
3. Create a constant for the **rule index** input of the FL Set Rule VI and set it to `0`. This input specifies that you want to modify the first rule in the fuzzy system.
4. Wire the **appended array** output of the first Build Array function to the **antecedents** input of the FL Set Rule VI.
5. Wire the **appended array** output of the second Build Array function to the **consequents** input of the FL Set Rule VI.
6. Create a constant for the **antecedent connective** input of the FL Set Rule VI. This input specifies how the VI calculates the truth value of the aggregated rule antecedent.
7. Select **AND (Minimum)** from the **antecedent connective** constant. The aggregated rule antecedent now is *IF 'Temperature' IS 'Cold' AND 'Humidity' IS 'Dry'*, where this VI uses the smallest degree of membership of the individual antecedents to calculate the truth value of the aggregated rule antecedent.

Note You cannot specify how this VI calculates the truth value of an aggregated rule consequent. This VI always uses the greatest degree of membership of the individual consequents to calculate the truth value of the aggregated rule consequent.

8. Create a constant for the **consequent implication** input of the FL Set Rule VI. This input specifies the implication method this VI uses to scale the membership functions of the output linguistic variables based on the rule weight.
9. Select **Product** from the **consequent implication** constant to specify that this VI uses the Product implication method.
10. Create a constant for the **degree of support** input of the FL Set Rule VI and set it to `1`. This input specifies the weight that you want to apply to the rule. The final rule weight is equal to the **degree of support** multiplied by the truth value of the aggregated rule antecedent.

11. Save the Modified Greenhouse Fuzzy System VI. The relevant portion of the block diagram should appear similar to Figure 10-5.

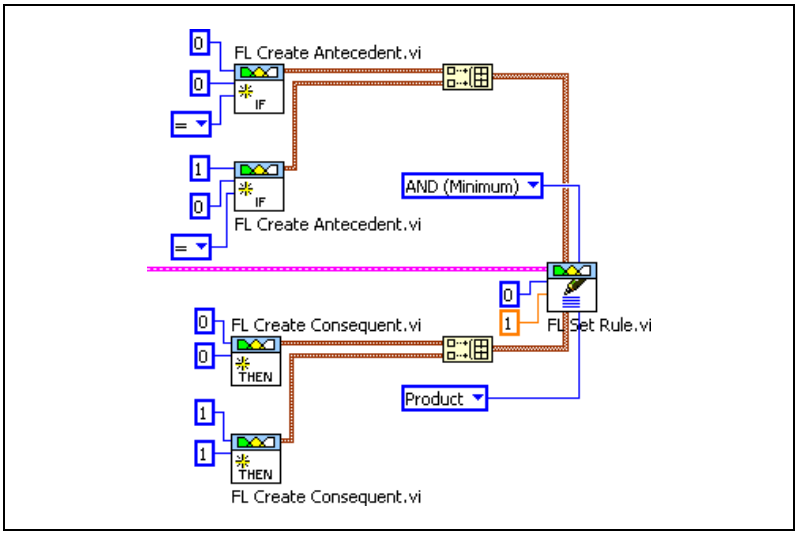

Figure 10-5. Modifying a Rule of a Fuzzy System

Saving the Fuzzy System

In previous sections of this chapter, you modified the linguistic variables, membership functions, and rules of the greenhouse fuzzy system. You now can save the fuzzy system to a `.fs` file.

Complete the following steps to save the modified fuzzy system to a `.fs` file.

1. Add an FL Save Fuzzy System VI to the block diagram.
2. Wire the **fuzzy system out** output of the FL Set Rule VI to the **fuzzy system in** input of the FL Save Fuzzy System VI.
3. Right-click the **file path** input of the FL Save Fuzzy System VI and select **Create»Constant** from the shortcut menu.
4. Enter the absolute path to a `modified greenhouse.fs` file in the path constant.

Note You also can use the File I/O VIs and functions to create a relative path to wire to the **file path** input of the FL Save Fuzzy System VI.

5. Save the Modified Greenhouse Fuzzy System VI. The block diagram should appear similar to Figure 10-6.

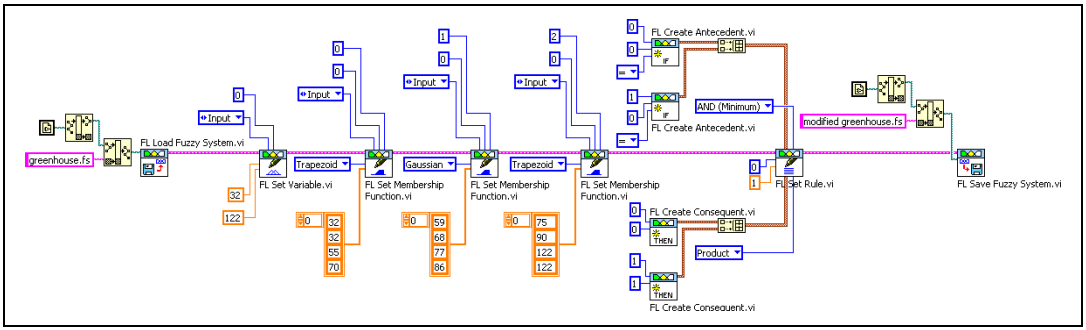

Figure 10-6. Block Diagram of the Modified Greenhouse Fuzzy System VI

6. Run the Modified Greenhouse Fuzzy System VI. This VI saves the `modified greenhouse.fs` file in the location you specified.

You now can use the Fuzzy System Designer to open the `modified greenhouse.fs` file and observe the modifications you made to the greenhouse fuzzy system.

You also can use the FL Fuzzy Controller VI to implement a fuzzy logic controller for the modified greenhouse fuzzy system. The FuzzyEx Dynamic Fuzzy Controller for a greenhouse example VI in the `labview\examples\control\fuzzy\Dynamic greenhouse controller` directory implements a controller for the greenhouse fuzzy system. This example VI uses the FL Load Fuzzy Controller VI to load the `greenhouse.fs` file. You can load the `modified greenhouse.fs` file instead and then run the FuzzyEx Dynamic Fuzzy Controller for a greenhouse example VI to implement a controller for the modified greenhouse fuzzy system.

A

Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Technical support at ni.com/support includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit ni.com/support 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. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support as well as exclusive access to on demand training modules via the Services Resource Center. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

    For information about other technical support options in your area, visit ni.com/services, or contact your local office 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.

Glossary

A

algorithm	A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.
antecedent	The IF portion of a rule in a fuzzy system.
antecedent connective	An operator that specifies how to calculate the truth value of an aggregated rule antecedent.
anti-reset windup	A method that prevents the integral term of the PID algorithm from moving too far beyond saturation when an error persists.
autotuning	Automatically testing a process under control to determine the controller gains that will provide the best controller performance.
Autotuning Wizard	An automated graphical user interface provided in the PID Autotuning VI. The Autotuning Wizard gathers some information about the desired control from the user and then steps through the PID autotuning process.

B

bias	The offset added to a controller output.
Boolean set theory	Traditional set theory based on strict membership or nonmembership of elements to a set. Examples are TRUE or FALSE, ON or OFF, 1 or 0, and so on.
bumpless transfer	A process in which the next output always increments from the current output, regardless of the current controller output value; therefore, transfer from automatic to manual control is always bumpless.

C

cascade control	Control in which the output of one controller is the setpoint for another controller.
Center of Area (CoA)	Method of defuzzification in which the crisp output is determined by the geometrical center of area of the composite output membership functions within the range of the output linguistic variable. Also known as Center of Gravity (CoG).
Center of Maximum (CoM)	Method of defuzzification in which the crisp output is determined by a weighted average of the typical values of each output membership function. This method is equivalent to the Center of Area method using singleton sets.
Center of Sums (CoS)	Method of defuzzification in which the crisp output is determined by a weighted average of the center of area of each output membership function.
closed loop	A signal path which includes a forward path, a feedback path, and a summing point and which forms a closed circuit. Also called a *feedback loop*.
consequent	The THEN portion of a rule in a fuzzy system.
controller	Hardware and/or software used to maintain parameters of a physical process at desired values.
controller output	*See* manipulated variable.
crisp value	A finite single value such as a measured physical quantity, for example, $x = 5.3$ m.
cycle time	The time between samples in a discrete digital control system.

D

damping	The progressive reduction or suppression of oscillation in a device or system.
deadtime (T_d)	The interval of time, expressed in minutes, between initiation of an input change or stimulus and the start of the resulting observable response.

defuzzification	The process of converting the degrees of membership of output linguistic variables within their linguistic terms into crisp numerical values.
degree of membership	A value that represents the degree of partial membership of a linguistic variable within a linguistic term. This value can range from 0 to 1.
degree of support	A weighting value, ranging from 0 to 1, that is applied to each rule in the rule base of a fuzzy controller. This weighting value represents the relative significance of each rule and allows for fine-tuning of the rule base.
derivative (control) action	Control response to the time rate of change of a variable. Also called *rate action*.
derivative kick	A sudden change in PID controller output resulting from derivative action applied to the error signal after a change in setpoint value. Derivative kick is normally avoided in PID control by applying derivative action only to the process variable and not to the error signal.
deviation	Any departure from a desired value or expected value or pattern.
downstream loop	In a cascade, the controller whose setpoint is provided by another controller.

E

EGU	Engineering units.
expert	A human operator of a system or process that has acquired knowledge related to controlling the process through experience.

F

FC	Flow controller.
feedback control	Control in which a measured variable is compared to its desired value to produce an actuating error signal that is acted upon in such a way as to reduce the magnitude of the error.
feedback loop	*See* closed loop.
fuzzification	The process of associating crisp input values with the linguistic terms of corresponding input linguistic variables.

fuzzy controller — A controller that uses defined rules to control a fuzzy system based on the current values of input linguistic variables.

fuzzy logic — An extension of traditional Boolean set theory that allows for partial membership in a set.

fuzzy system — A system of input and output variables associated using fuzzy logic.

G

gain — For a linear system or element, the ratio of the magnitude, or amplitude, of a steady-state sinusoidal output relative to the causal input; the length of a phasor from the origin to a point of the transfer locus in a complex plane. Also called the *magnitude ratio*.

gain scheduling — The process of applying different controller gains for different regions of operation of a controller. Gain scheduling is most often used in controlling nonlinear physical processes.

I

implication method — A mathematical method for scaling the membership functions of an output linguistic variable based on the rule weight before performing defuzzification.

integral (control) action — Control action in which the output is proportional to the time integral of the input. That is, the rate of change of output is proportional to the input.

ISA — Instrument Society of America—The organization that sets standards for process control instrumentation in the United States.

K

K — Process gain.

K_c — Controller gain.

L

lag	A lowpass filter or integrating response with respect to time.
linearity factor	A value ranging from 0 to 1, used to specify the linearity of a calculation. A value of 1 indicates a linear operation. A value of 1 indicates a squared nonlinear operation
linguistic term	A word or set of words that represents categories for the values of a linguistic variable. A linguistic term is defined quantitatively by the corresponding membership function.
linguistic variable	A word or set of words that represents an input variable or output variable of a fuzzy system.
load disturbance	The ability of a controller to compensate for changes in physical parameters of a controlled process while the setpoint value remains constant.
loop cycle time	Time interval between calls to a control algorithm.

M

magnitude ratio	*See* gain.
manipulated variable	A quantity or condition that is varied as a function of the actuating error signal so as to change the value of the directly controlled variable. Also called *controller output*.
Mean of Maximum (MoM)	Method of defuzzification in which the crisp output is determined by selecting a value corresponding to the maximum degree of membership of the composite output membership function. If there are multiple maximums, the mean of the corresponding values is selected.
membership function	A numerical function that quantitatively defines the degree of membership of a linguistic variable within a linguistic term.
modified Center of Area (CoA)	Method of defuzzification in which the crisp output is determined by the geometrical center of area of the composite output membership functions. Unlike the Center of Area method, this method allows the crisp output to realize the full range of the output variable.

N

noise	In process instrumentation, an unwanted component of a signal or variable. Noise may be expressed in units of the output or in percent of output span.

O

output limiting	Preventing a controller output from travelling beyond a desired maximum range.
overshoot	The maximum excursion beyond the final steady-state value of output as the result of an input change. Also called *transient overshoot*.

P

P	Proportional.
P controller	A controller which produces proportional control action only; that is, a controller that has only a simple gain response.
partial membership	In fuzzy logic, a condition in which the value of a member partially fulfills the requirements of the membership function of a set.
PC	Pressure controller.
PD	Proportional, derivative.
PD controller	A controller that produces proportional plus derivative (rate) control action.
PI	Proportional, integral.
PI controller	A controller that produces proportional plus integral (reset) control action.
PID	Proportional, integral, derivative.
PID control	A common control strategy in which a process variable is measured and compared to a desired set point to determine an error signal. A proportional gain (P) is applied to the error signal, an integral gain (I) is applied to the integral of the error signal, and a derivative gain (D) is applied to the derivative of the error signal. The controller output is a linear combination of the three resulting values.

PID controller — A controller that produces proportional plus integral (reset) plus derivative (rate) control action.

process gain (K) — For a linear process, the ratio of the magnitudes of the measured process response to that of the manipulated variable.

process variable (PV) — The measured variable (such as pressure or temperature) in a process to be controlled.

proportional action — Control response in which the output is proportional to the input.

proportional band (PB) — The change in input required to produce a full range change in output due to proportional control action. PB = 100 /K_c.

Q

quarter-decay ratio — A response in which the amplitude of each oscillation is one-quarter that of the previous oscillation.

R

ramp — The total (transient plus steady-state) time response resulting from a sudden increase in the rate of change from zero to some finite value of the input stimulus. Also called *ramp response*.

rate action — Control response to the time rate of change of a variable. Also called *derivative control action*.

reentrant — Execution mode in which calls to multiple instances of a subVI can execute in parallel with distinct and separate data storage.

reset rate — Of proportional plus integral or proportional plus integral plus derivative control action devices: for a step input, the ratio of the initial rate of change of output due to integral control action to the change in steady-state output due to proportional control action.

Of integral control action devices: for a step input, the ratio of the initial rate of change of output to the input change. Also called *integral action rate*.

rule — A linguistic representation of the relationships between input and output linguistic variables based on their linguistic terms.

rule base — The set of rules for a fuzzy system.

S

selector control	The use of multiple controllers and/or multiple process variables in which the connections may change dynamically depending on process conditions.
singleton	A normalized membership function with an infinitely small width. A singleton is used to model a crisp value with a fuzzy set.
SP	Setpoint—An input variable which sets the desired value of the controlled process variable.
span	The algebraic difference between the upper and lower range values.
stochastic uncertainty	The degree of uncertainty of the occurrence of a given future nondeterministic event.

T

time constant (T)	In process instrumentation, the value T (in minutes) in an exponential response term, A exp (–t/T), or in one of the transform factors, such as 1 + sT.
transient overshoot	*See* overshoot.
trapezoidal integration	A numerical of integration in which the current value and the previous value are used to calculate the addition of the current value to the integral value.

W

windup area	The time during which the controller output is saturated at the maximum or minimum value. The integral action of a simple PID controller continues to increase (wind up) while the controller is in the windup area.