NATIONAL INSTRUMENTS™
LabVIEW™

PID Control Toolset
User Manual

NATIONAL INSTRUMENTS™

November 2001 Edition
Part Number 322192A-01

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 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, China (Shanghai) 021 6555 7838,
China (ShenZhen) 0755 3904939, Czech Republic 02 2423 5774, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186,
India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Malaysia 603 9596711, Mexico 001 800 010 0793, Netherlands 0348 433466, New Zealand 09 914 0488,
Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011, Russia 095 2387139,
Singapore 2265886, Slovenia 386 3 425 4200, South Africa 11 805 8197, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com.

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

FieldPoint™, LabVIEW™, National Instruments™, NI™, ni.com™, and NI-DAQ™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications. Refer to `ni.com/legal/patents` for the most current list of patents covering this product.

The LabVIEW PID Control Toolset is covered by one or more of the following Patents: U.S. Patent No(s).: 6,081,751

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

Chapter 3
Using the PID Software

Chapter 4
Process Control Examples

PART II
Fuzzy Logic Control

Chapter 5
Overview of Fuzzy Logic

Chapter 6
Fuzzy Controllers

Chapter 7
Design Methodology

Chapter 8
Using the Fuzzy Logic Controller Design VI

About This Manual

The PID Control Toolset User Manual describes the new PID Control Toolset for LabVIEW. This toolset includes PID Control, Fuzzy Logic Control, and Advanced Control VIs.

Organization of This Manual

The PID Control Toolset User Manual is organized as follows:

Part I, *PID Control*—This section of the manual describes the features, functions, and operation of PID Control portion of the PID Control Toolset. To use this section, you need a basic understanding of process control strategies and algorithms. Refer to Appendix A, *References*, for other sources of information on process control, terminology, methods, and standards.

Part II, *Fuzzy Logic Control*—This section of the manual describes the features, functions, and operation of the Fuzzy Logic Control portion of the PID Control Toolset. You can use the Fuzzy Logic Controls to design and implement rule-based fuzzy logic systems for process control or expert decision making. To use this section effectively, you need to be familiar with basic control theory. Knowledge of rule-based systems and fuzzy logic helps as well.

Part III, *Advanced Control*—This section of the manual describes the functions of the the Advanced Control portion of the PID Control Toolset. You can use the Advanced Controls to develop advanced control algorithms and simulate physical systems for Hardware-In-the-Loop (HIL) simulation applications.

Conventions Used in This Manual

The following conventions appear in this manual:

[ ] Square brackets enclose optional items—for example, [`response`].

» 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 tip, which alerts you to advisory information.

This icon denotes a note, which alerts you to important information.

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

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. This font also denotes text that is a placeholder for a word or value that you must supply.

`monospace` — Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, filenames and extensions, and code excerpts.

`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.

`monospace italic` — Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

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

- *LabVIEW PID Control Toolset Help*
- *LabVIEW User Manual*
- *LabVIEW Measurements Manual*
- *Getting Started with LabVIEW*

1

Overview of the PID Control Toolset

This chapter lists the contents of the PID Control Toolset, describes how to install the toolset, and describes the PID Control applications.

Package Contents

The PID Control Toolset contains the following materials:

- *PID Control Toolset User Manual*
- Software that includes control and example VIs

System Requirements

Your computer must meet the following minimum system requirements to run the PID Control Toolset:

- LabVIEW 6.0 or later
- Windows 2000/NT/9*x*

Installation Procedure

Complete the following steps to install the PID Control Toolset on Windows 2000/NT/9*x*.

1. Launch Windows.
2. Insert the PID Control Toolset CD.
3. Follow the instructions on your screen.

After you complete the on-screen installation instructions, you are ready to run the PID Control Toolset.

PID Control Toolset Applications

The PID Control Toolset contains functions you can use to develop LabVIEW control applications.

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 Control Toolset 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 Control Toolset to develop a control application or modify the examples provided with the Toolset.

Using the VIs described in the *PID Control* section of the manual, you can 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
- Multiloop cascade control
- Feedforward control
- Override (minimum/maximum selector) control
- Ratio/bias control

You can combine these PID Control VIs with LabVIEW math and logic functions to create block diagrams for real control strategies. The PID Control VIs use LabVIEW functions and library subVIs, without any Code Interface Nodes (CINs), to implement the algorithms. You can modify the VIs for your applications in LabVIEW, without writing any text-based code.

Refer to the *LabVIEW PID Control Toolset Help*, available by selecting **Help»PID Control Toolset 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 that emulates the rule-of-thumb thought process that human beings use.

You can use fuzzy logic to control processes that a person manually controls, based on expertise gained from experience. A human operator who is an expert in a specific process often uses a set of linguistic control rules, based on experience, that he can describe generally and intuitively. Fuzzy logic provides a way to translate these linguistic descriptions to the rule base of a fuzzy logic controller. Refer to Chapter 5, *Overview of Fuzzy Logic*, for more information.

How Do the Fuzzy Logic VIs Work?

With the Fuzzy Logic VIs, you can design a fuzzy logic controller, an expert system for decision making, and implement the controller in your LabVIEW applications. The Fuzzy Logic Controller Design VI, available by selecting **Tools»Fuzzy Logic Controller Design**, defines the fuzzy membership functions and controller rule base. The Controller Design VI is a stand-alone VI with a user interface you can use to completely define all controller and expert system components and save all of the parameters of the defined controller to one controller data file.

You use two additional VIs to implement the fuzzy controller in your LabVIEW application. The Load Fuzzy Controller VI loads all the parameters of the fuzzy controller previously saved by the Controller Design VI. The Fuzzy Controller VI implements the fuzzy logic inference engine and returns the controller outputs. To implement real-time decision making or control of your physical system, you can wire the data acquired by your data acquisition device to the fuzzy controller. You also can use outputs of the fuzzy controller with your DAQ analog output hardware to implement real-time process control.

Advanced Control

The Advanced Control VIs include VIs for complex control applications as well as Hardware-In-the-Loop (HIL) applications. Use VIs such as the Discrete State-Space VI for control applications. Use VIs such as the Linear Transfer Function VI and Friction VI for HIL simulations.

The Advanced Control VIs are divided into the following groups:

- Continuous Linear Control
- Discrete Linear Control
- Nonlinear Control

Part I

PID Control

This section of the manual describes the PID Control portion of the PID Control Toolset.

- Chapter 2, *PID Algorithms*, introduces the algorithms used by the PID Control VIs.
- Chapter 3, *Using the PID Software*, explains how to use the PID Control VIs.
- Chapter 4, *Process Control Examples*, provides examples of different applications that use PID Control VIs.

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 Controls, 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.

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

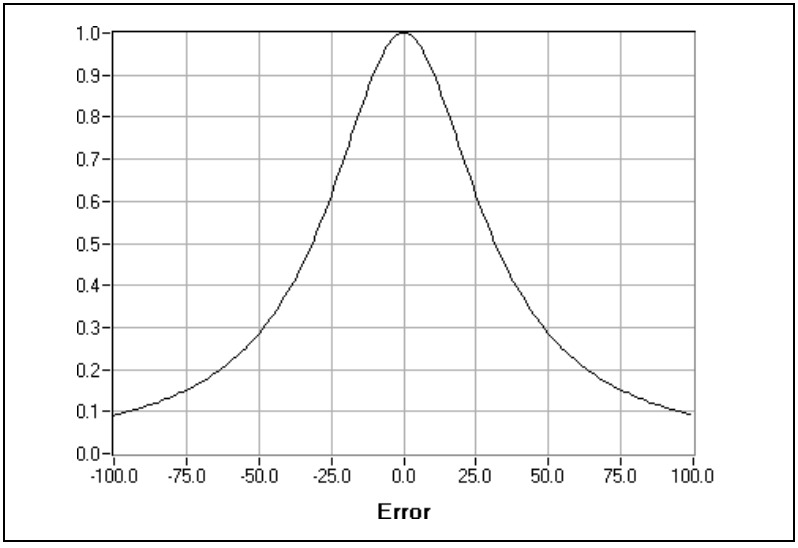


Figure 2-1. Nonlinear Multiple for Integral Action (SP_{rng} = 100)

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.

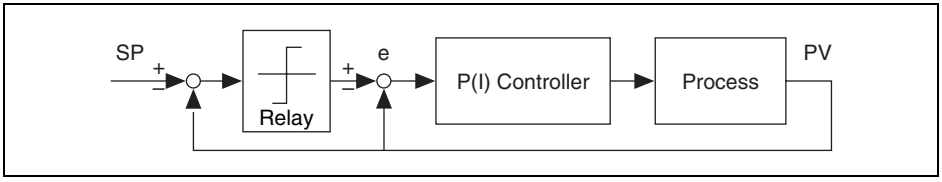


Figure 2-2. Process under PID Control with Setpoint Relay

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

This package 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 Control VIs in this toolset, 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 the equivalent LabVIEW block diagram. The only elements missing from this simplified VI are the loop-tuning parameters and the automatic-to-manual switching.

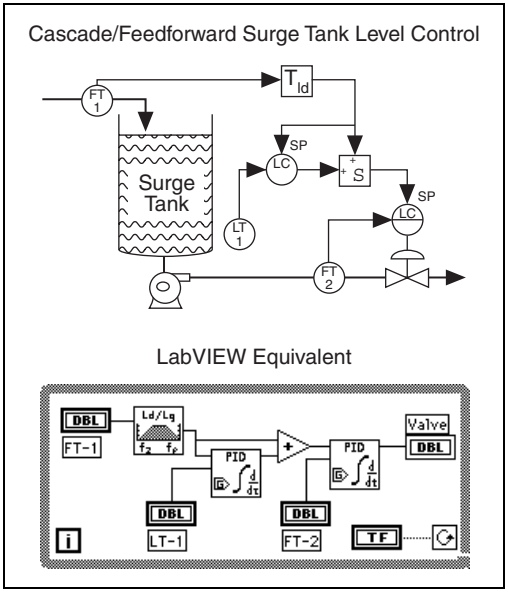


Figure 3-1. Control Flowchart and Block Diagram

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 and Lead-Lag VIs in this toolset 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 in Windows 2000. Because of this limitation, do not try to run the PID VIs faster than 5 or 10 Hz when **dt** is less than or equal to zero. Refer to the `LVREADME.WRI` file for more information about increasing your timer resolution.

If **dt** is a positive value in seconds, the VI uses that value in the calculations, regardless of the elapsed time. Use this method for fast loops, such as when you use acquisition hardware to time the controller input. Refer to the Demo-HW Timed PID VI located in the example library `prctlex.llb` for an example of using hardware timing with the combined PID and DAQ VIs. In this example, LabVIEW samples the analog input at precisely timed intervals, inverts the **actual scan rate** parameter from the AI Start VI, and wires the **actual scan rate** into the **dt** input.

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 1990). 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 3-2 shows an example of a cascade with two independently timed While Loops.

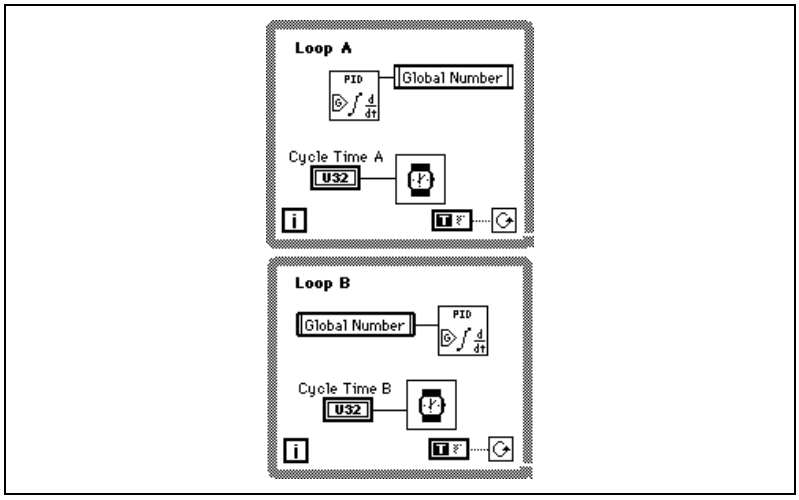


Figure 3-2. Cascaded Control Functions

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 1990). 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 (1990) as listed in Appendix A, *References*, 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 3-3.

   The variables represent the following values:

   - T_d—Deadtime in minutes
   - T—Time constant in minutes
   - K—Process gain = $\frac{\text{change in output}}{\text{change in } PV}$

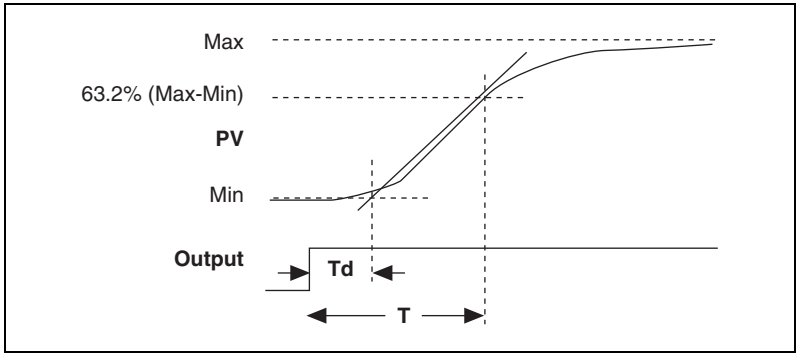


Figure 3-3. Output and Process Variable Strip Chart

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.

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 Advanced PID VI implements bumpless manual-to-automatic transfer, ensuring a smooth controller output during the transition from manual to automatic control mode. However, the Advanced PID 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 3-4.

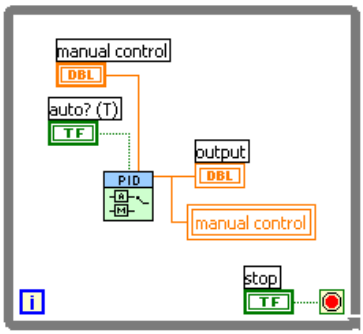


Figure 3-4. Bumpless Automatic-to-Manual Transfer

Multi-Loop PID Control

Most of the PID control 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 Control Toolset operate in the same manner.

Setpoint Ramp Generation

The PID Setpoint Profile VI located on the PID palette 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 3-5.

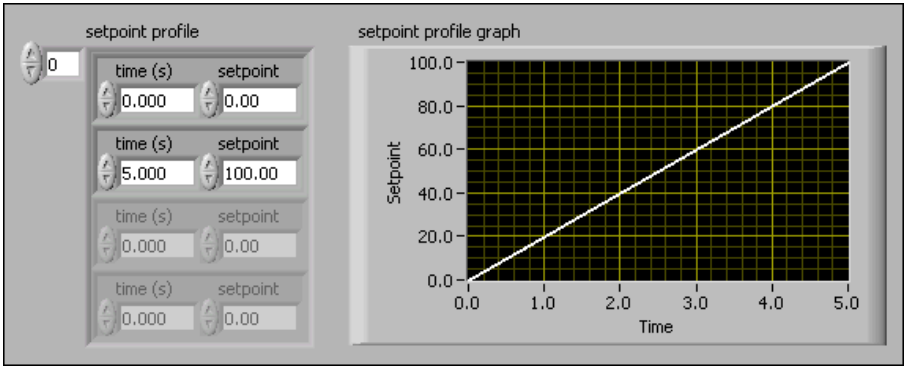


Figure 3-5. Ramp Setpoint Profile

A ramp and hold setpoint profile also can have two successive array values with the same setpoint value, as shown in Figure 3-6.

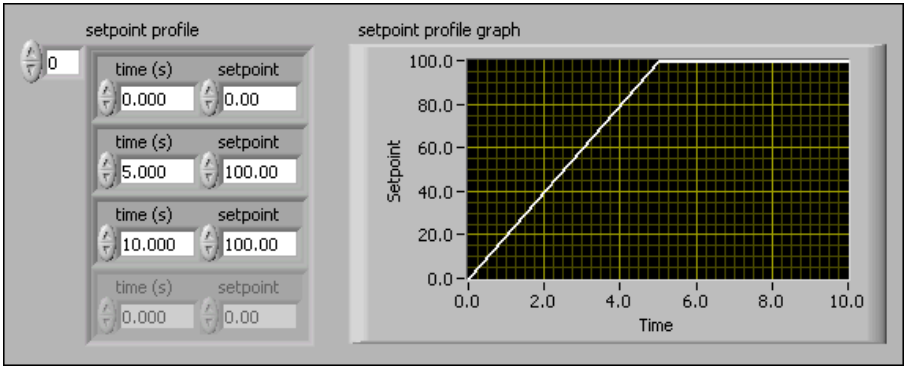


Figure 3-6. Ramp and Hold Setpoint Profile

Alternatively, a step setpoint profile can have two successive array values with the same time value but different setpoint values, as shown in Figure 3-7.

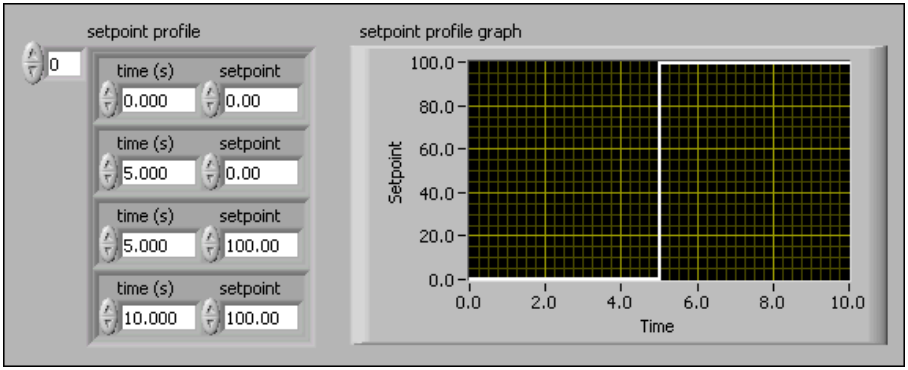


Figure 3-7. Step Setpoint Profile

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 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 low-pass fifth-order Finite Impulse Response (FIR) filter. The cutoff frequency of the low-pass 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.

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 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 3-8, the Gain Schedule Example uses the **setpoint** value as the **gain scheduling variable** with a default range of 0 to 100. Table 3-3 summarizes **PID parameters**.

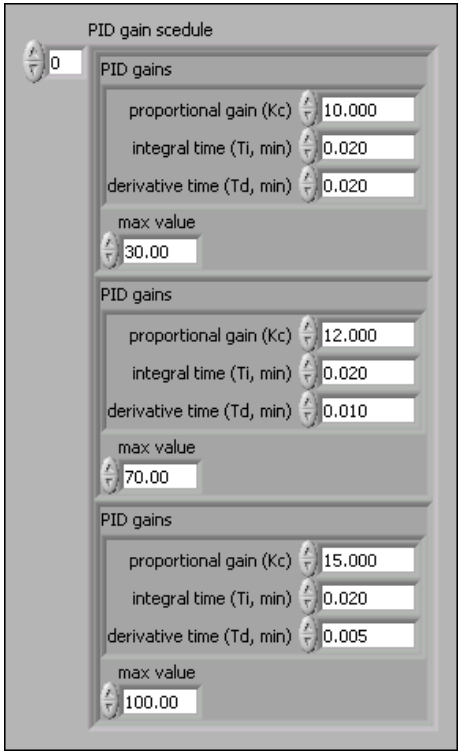


Figure 3-8. Gain Scheduling Input Example

Table 3-3. PID Parameter Ranges

Range	Parameters
$0 \le SP \le 30$	**Kc** = 10, **Ti** = 0.02, **Td** = 0.02
$30 < SP \le 70$	**Kc** = 12, **Ti** = 0.02, **Td** = 0.01
$70 < SP \le 100$	**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** and this will be 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 % VI to convert any input from real engineering units to percentage of full scale, and you can use the PID % to EGU function to convert the controller output from percentage to real engineering units. The PID % 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 with 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 with Autotuning VI. Because the PID with Autotuning VI has input and output consistent with the other PID VIs, you can replace any PID VI with it. The PID with Autotuning VI has several additional input and output values to specify the autotuning procedure. The two additional input values are **autotuning parameters** and **autotune?**. **autotuning parameters** 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 with 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 3-9 shows one possible implementation of the PID with 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.

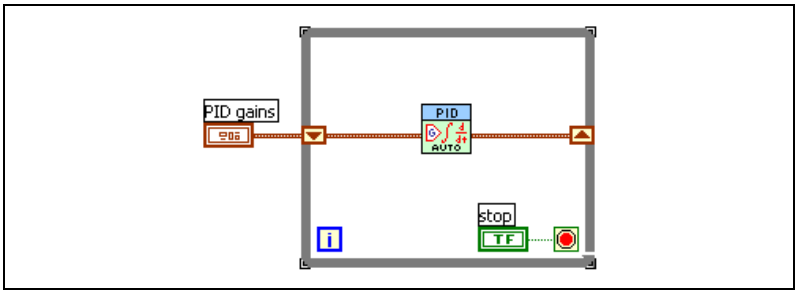


Figure 3-9. Updating PID Parameters Using a Shift Register

Figure 3-10 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.

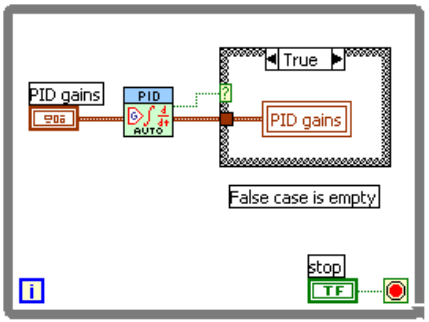


Figure 3-10. Updating PID Parameters Using a Local Variable

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 3-11.

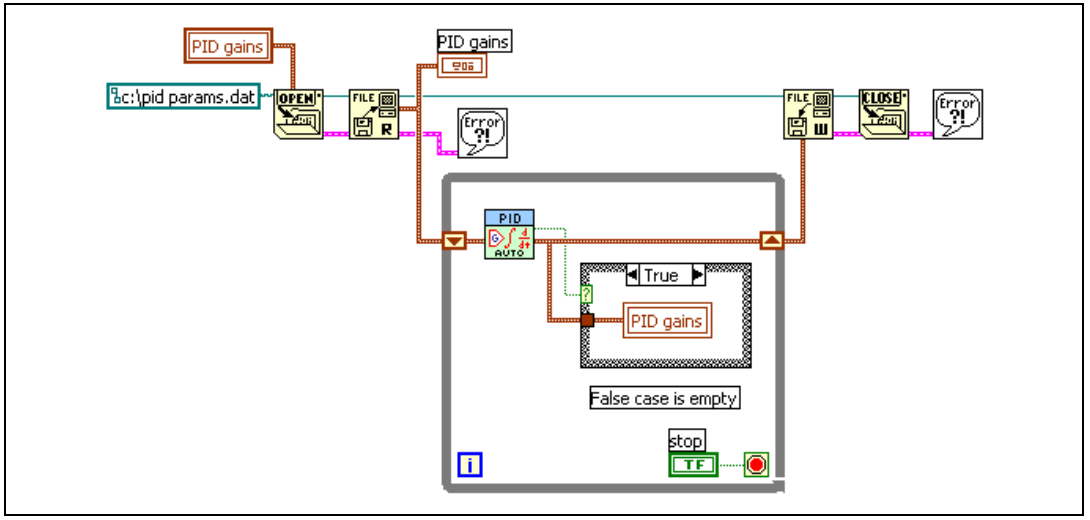


Figure 3-11. Storing PID Parameters in a Datalog File

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 control VI uses updated parameters.

Using PID with DAQ Devices

The remaining sections in this chapter address several important issues you might encounter when you use the DAQ VIs to implement control of an actual process. The following examples illustrate the differences between using easy-level DAQ VIs and using advanced DAQ VIs, as well as the differences between hardware timing and software timing.

Note Refer to `[LabVIEW]\examples\daq\solution\control.llb` for additional examples of control with DAQ VIs.

Software-Timed DAQ Control Loop

Figure 3-12 illustrates the basic elements of software control. The model assumes you have a plant, a real process, to control. A basic analog input VI reads process variables from sensors that monitor the process. In actual applications, you might need to scale values to engineering units instead of voltages.

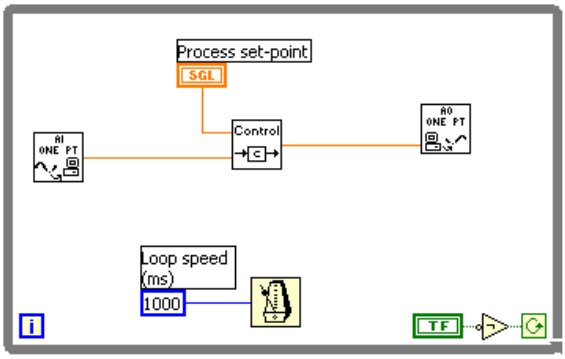


Figure 3-12. Software-Timed DAQ Control Loop

The Control VI represents the algorithm that implements software control. The Control VI can be a subVI you write in LabVIEW, a PID controller, or the Fuzzy Controller VI. An analog output VI updates the analog voltages that serve as your controller outputs to the process.

The Wait Until Next ms Multiple function that controls the loop timing in this example specifies only a minimum time for the loop to execute. Other operations in LabVIEW can increase the execution time of the loop functions. The time for the first loop iteration is not deterministic. Refer to *LabVIEW Help* for more information about timing control loops.

Implementing Advanced DAQ VIs in Software-Timed DAQ Control Loops

For faster I/O and loop speeds, use the advanced-level DAQ VIs for analog input and output. The easy-level VIs shown in Figure 3-12 actually use the advanced-level DAQ VIs shown in this example. However, the easy-level VIs configure the analog input and output with each loop iteration, which creates unnecessary overhead that can slow your control loops.

You can use the advanced-level DAQ VIs to configure the analog input and output only once instead of on each loop iteration. Be sure to place the configuration functions outside the loop and pass the task ID to the I/O functions inside the loop. The AI SingleScan and AO Single Update VIs call the DAQ driver directly instead of through other subVI calls, minimizing overhead for DAQ functions.

This example does not use a timing function to specify the loop speed. Thus, the control loop runs as fast as possible, and LabVIEW maximizes the control loop rates. However, any other operation in LabVIEW can slow down the loop and vary the speed from iteration to iteration. Because Windows NT/2000 is a preemptive multitasking operating system, other running applications can affect the loop speed.

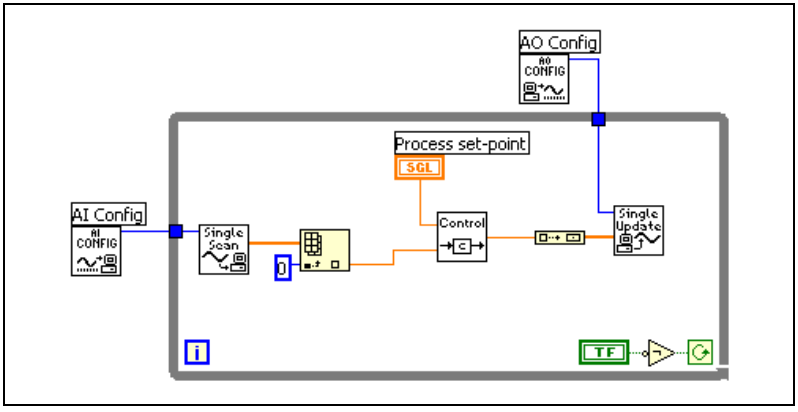


Figure 3-13. Software-Timed DAQ Control Loop with Advanced Features

Hardware-Timed DAQ Control Loop

Figure 3-14 demonstrates hardware timing. In this example, a continuous analog input operation controls the loop speed. Notice that the intermediate- and advanced-level DAQ VIs specify the acquisition rate for the analog input scanning operation. The analog output VIs are identical to those in the previous example.

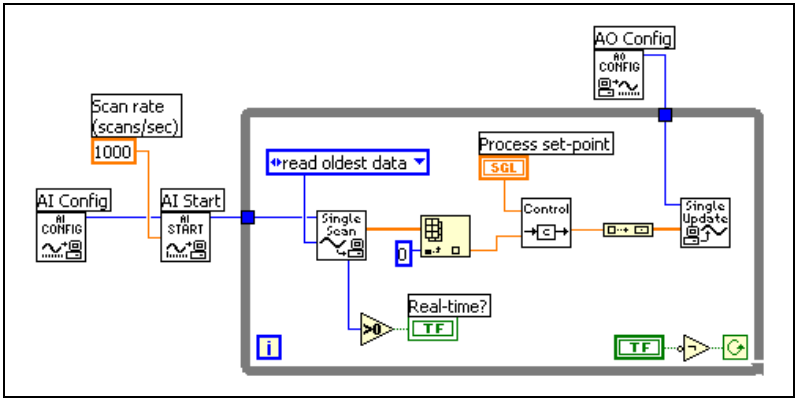


Figure 3-14. Hardware-Timed DAQ Control Loop

With each loop iteration, the AI SingleScan VI returns one scan of data. The Control VI processes data, and LabVIEW updates the analog output channels as quickly as the VI can execute.

If the processing time of the loop subdiagram remains less than the scan interval, the **scan rate** dictates the control rate. If the processing of the analog input, control algorithm, and analog output takes longer than the specified scan interval, which is 1 ms in this example, the software falls behind the hardware acquisition rate and does not maintain real time. If you monitor **data remaining** when you call AI SingleScan, you can determine whether the VI has missed any scans. If **data remaining** remains zero, the control is real time.

4

Process Control Examples

This chapter describes examples that use the PID Control VIs. You do not need any DAQ devices to run the Simulation VIs, but you must have the appropriate hardware to run the Demonstration VIs.

Simulation VIs

The simulation examples demonstrate control of a process that is simulated entirely in software. You can use these examples to learn about the operation of a PID controller without connecting the control application to a real physical process.

Tank Level

The Tank Level VI is a simple process simulation for tank level. A level controller adjusts the flow into a tank. To represent a change in process loading, click the on/off value that serves as a drain. With this VI, you also can switch from automatic mode to manual mode. Figure 4-1 shows the front panel of the Tank Level VI.

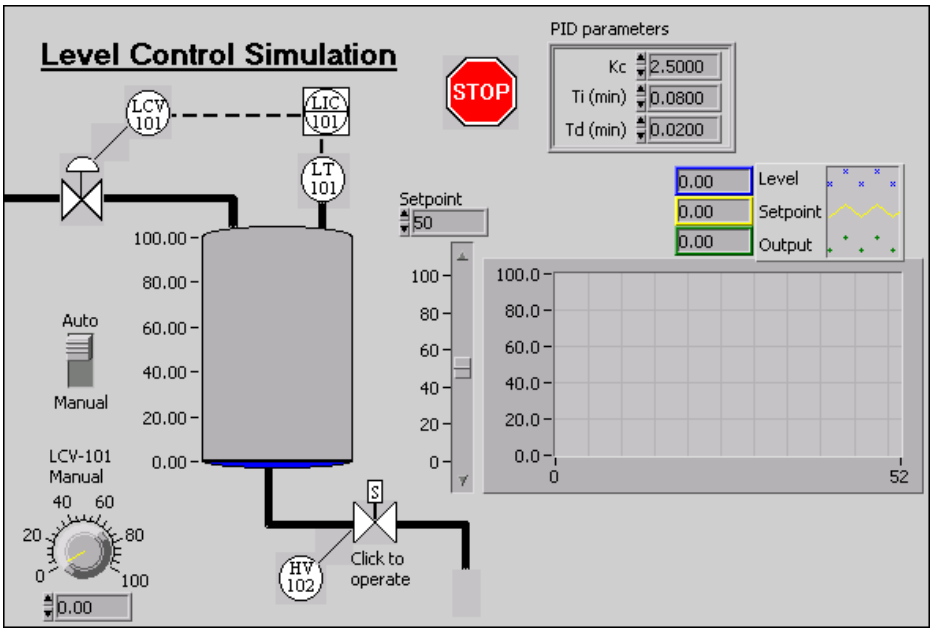


Figure 4-1. Front Panel of the Tank Level VI

The Tank Level VI uses an integrating process with added noise, valve deadband, lag, and deadtime. This VI is not time aware and, unlike the PID block, this VI does not correct itself for the loop cycle time. The cycle time is fixed at 0.5 s. Figure 4-2 shows the block diagram of the Tank Level VI.

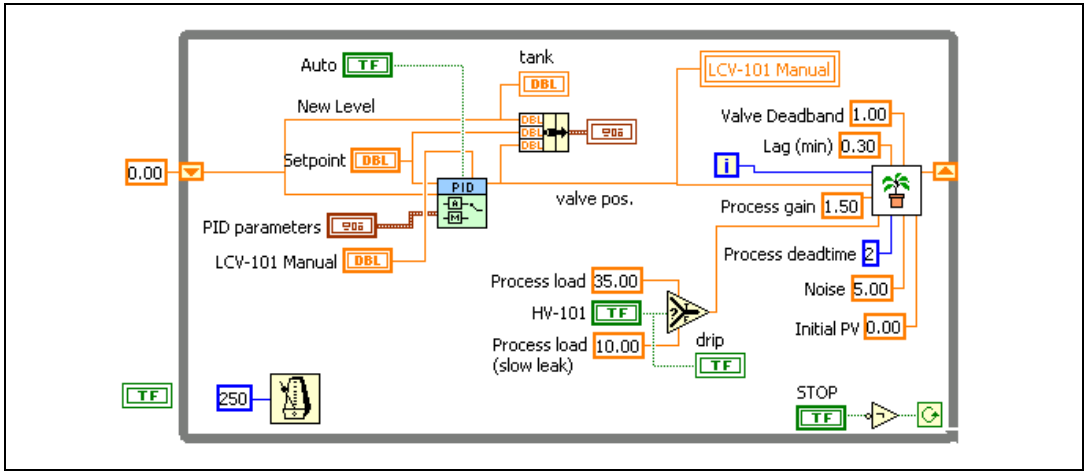


Figure 4-2. Block Diagram of the Tank Level VI

The Plant Simulator subVI, which simulates this process, reads and delays the previous valve position and scales it according to the process gain. The gain represents how fast the tank fills versus the position of the valve. The process load value depends on the state of the drain valve, HV-101. The tank level drops when you open the valve.

General PID Simulator

The General PID Simulator VI resembles the Tank Level VI, except that all process adjustments appear on the front panel of the General PID Simulator VI. This VI uses a simple integrating process, such as a level control loop, with added noise, valve deadband, lag, deadtime, and variable loading, all of which you can adjust. This VI is not time aware. Unlike the PID VI, the General PID Simulator VI does not correct itself for the loop cycle time. Set **Cycle Time** to 1 s, unless you want to modify the process. Figure 4-3 shows the font panel of the General PID Simulator VI.

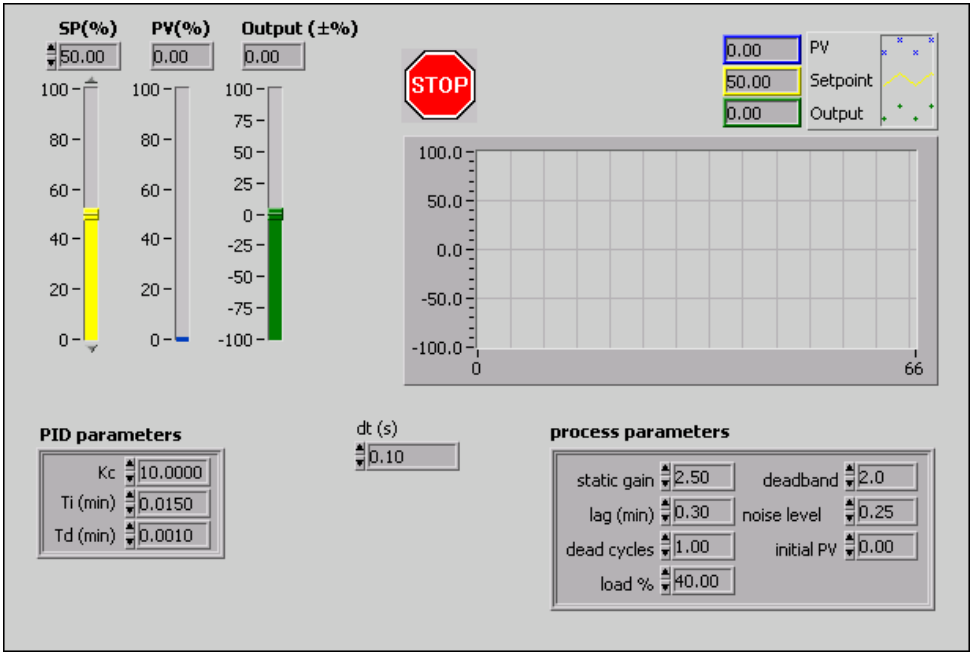


Figure 4-3. Front Panel of the General PID Simulator VI

The General PID Simulator VI also demonstrates switching between automatic and manual modes and run and hold modes. Figure 4-4 shows the block diagram of the General PID Simulator VI.

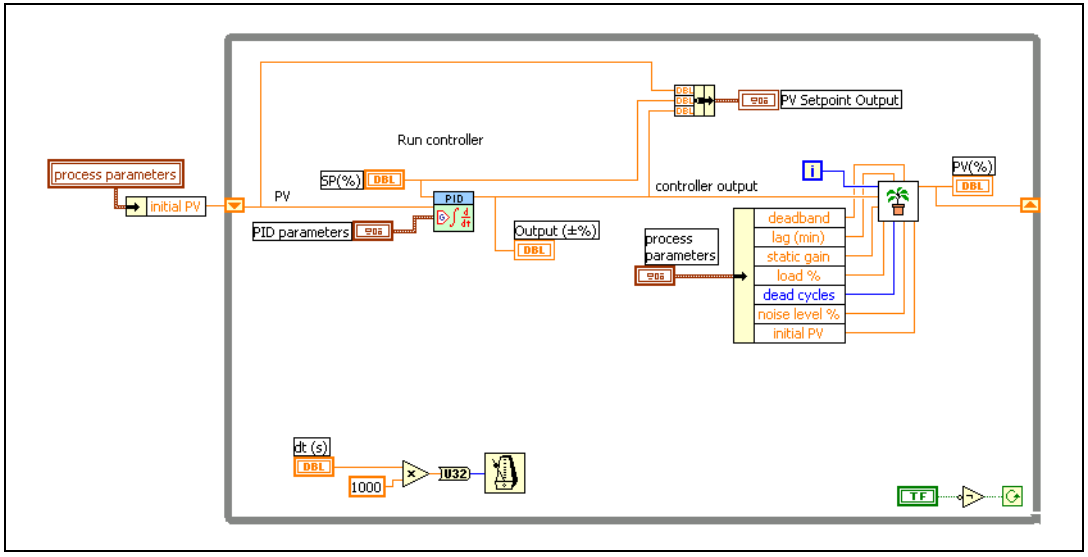


Figure 4-4. Block Diagram of the General PID Simulator VI

Like the Tank Level VI, this controller simulation uses the Plant Simulator VI. This general PID simulation can be thought of as a pressure-control application. The next execution of the While Loop reads and delays the previous valve position, then scales it according to the process gain. The gain represents the process response versus the valve position. The gain might have units of PSI per valve percent.

Process Deadtime is a multiple of **Cycle Time** rather than an absolute, fixed delay. If you change **Cycle Time**, you must adjust **Process Deadtime** to keep the process response constant. You do not need to adjust **Lag** because the lag time is time aware.

Plant Simulator

The Plant Simulator VI, shown in Figure 4-5, simulates a physical plant response.

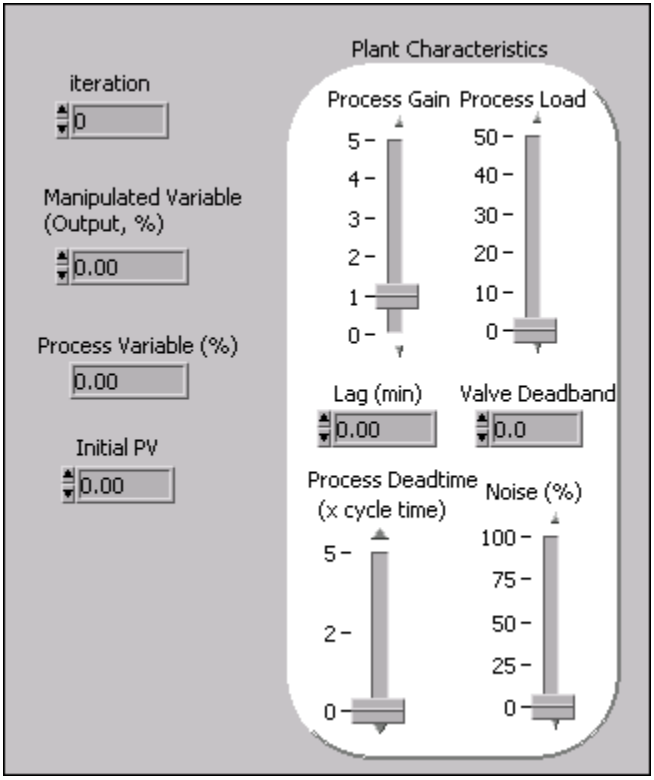


Figure 4-5. Front Panel of the Plant Simulator VI

The Plant Simulator VI measures the **PV** in percent. To use this VI with feedback control loops, call the VI with the **Update PV** switch set to FALSE and implement your PID algorithm. Then, call this VI with the **Update PV** switch set to TRUE. When you supply all the plant characteristics and connect the PID output to the Manipulated Variable, the VI calculates the new PV value.

When you set the **Update PV** control to FALSE, the value of the last **PV** passes to the output. When you set the **Update PV** control to TRUE, the VI reads and delays the **PV**, and then scales it according to the **Process Gain**. The VI adds noise and the resulting process response to the old-level value through a first-order lag filter, rather than using the Addition function, which adds a reasonable time constant to the apparent response

of the tank. The output of this filter is the new **PV**. Figure 4-6 shows the block diagram of the Plant Simulator VI.

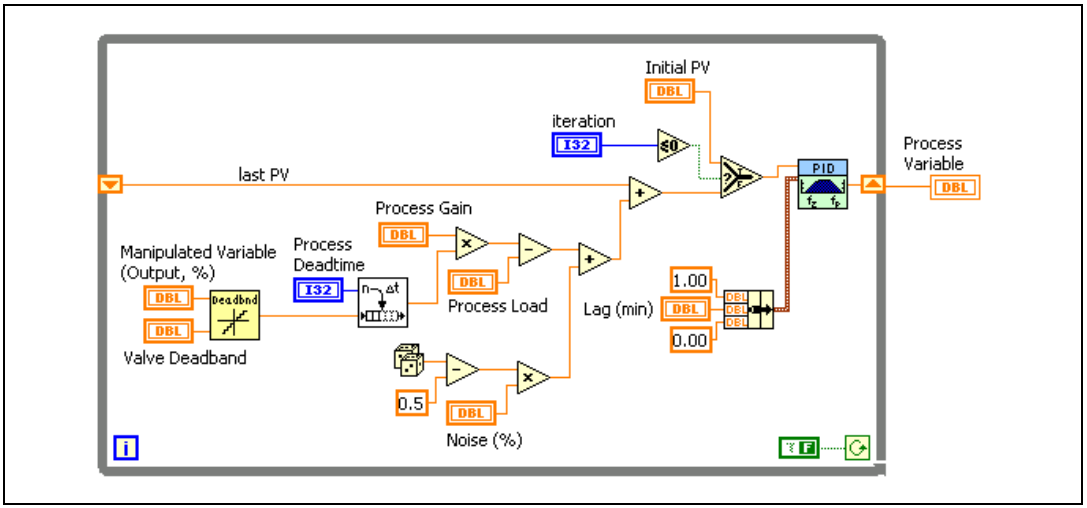


Figure 4-6. Block Diagram of the Plant Simulator VI

To simulate more than one plant simultaneously, save multiple copies of this VI under different names.

Cascade and Selector

The Cascade and Selector VI demonstrates a cascade and selector control, also known as a limit or high-low control. This VI simulates a compressor driven by a motor with a tachometer, which requires a PID loop to control the speed, the downstream loop. The flow and pressure from the compressor pass to individual PID controllers. You want to control the flow, but if the pressure exceeds a specified SP, the pressure becomes the controlled variable. This calls for a low select function to combine the two upstream controller outputs. The lower of the two outputs becomes the SP for the compressor speed. Figure 4-7 shows the front panel of the Cascade and Selector VI.

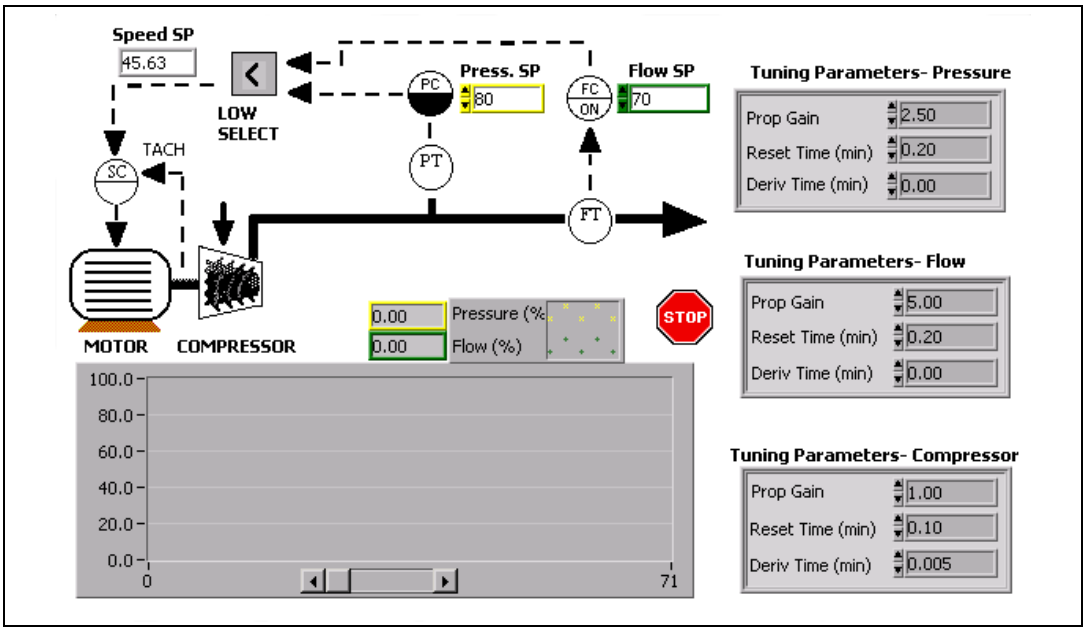


Figure 4-7. Front Panel of the Cascade and Selector VI

When you run the VI, the pressure controller (**PC**) and flow controller (**FC**) symbols read ON when you select them. Normally, you should leave the pressure **SP** constant. Increase the flow significantly and notice that the **PC** takes over control as pressure overshoots the setpoint for a short period of time. In the real world, a plugged pipe also causes the pressure loop to take over.

Note All variables in this simulation are percentages. In a real application, you might want to normalize all the input and setpoint values to percentages before you pass them to the PID controllers.

The downstream loop, also known as the inner loop, which is the compressor speed control, must be faster than the outer loops. Use a factor of 10 to prevent oscillation. In this simulation, the compressor lag is smaller and faster than the lag of the outer loops.

The block diagram for this VI contains cascaded loops. The inner loop at the top of the diagram consists of the compressor PID speed controller, a lag, and added noise. The output of the PID represents power supplied to the motor. LabVIEW passes the output to a shift register. The next iteration of the While Loop applies a lag to simulate the inertia of the

motor/compressor system. Added noise makes the simulation more realistic. Figure 4-8 is the block diagram of the Cascade and Selector VI.

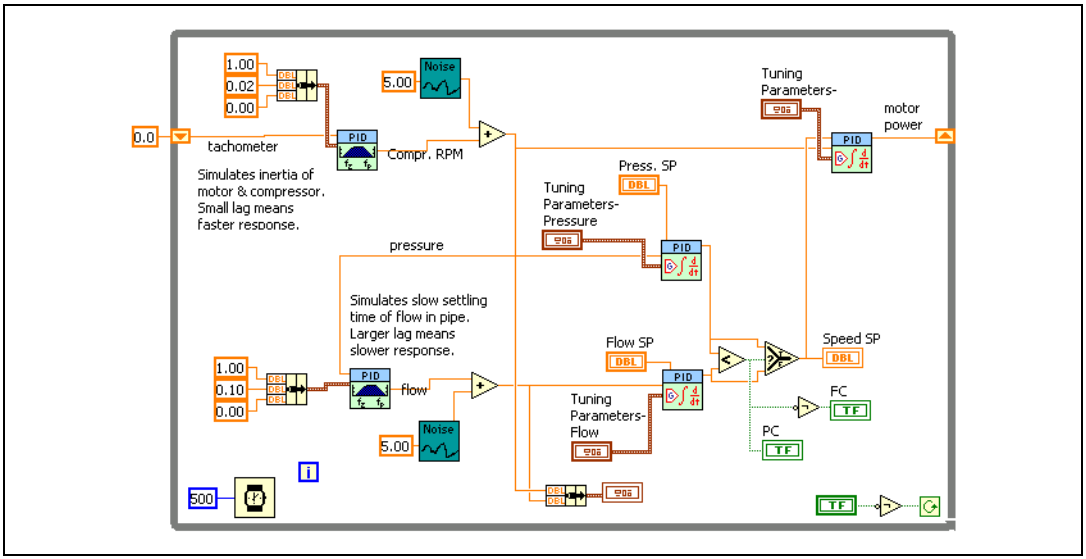


Figure 4-8. Block Diagram of the Cascade and Selector VI

The right half of the diagram contains two PID VIs, one for pressure control and one for flow control. The VI selects the While Loop that produces the lowest output value as the active controller, and routes the output of that While Loop to the compressor speed control PID SP.

The VI derives feedback for the two controllers from the process response of the compressor. Although you can add lag and/or noise, assume the **pressure** is the same as the compressor RPM. This VI adds both lag and noise to the **flow** to better simulate the real-world response of flowing fluids.

Demonstration VIs

The demonstration VIs show how you can use the PID Control Toolset functions to control real physical processes.

PID with MIO Board

The PID with MIO Board VI turns your computer into a single-loop PID controller when you use a National Instruments (NI) DAQ device. Connect the analog output to the analog input through the resistor-capacitor network shown on the front panel in Figure 4-9.

Note You must have the appropriate hardware to run this example. The pin numbers shown in Figure 4-9 correspond to the standard MIO pinouts, such as a standard 50-pin cable from an E-Series DAQ device.

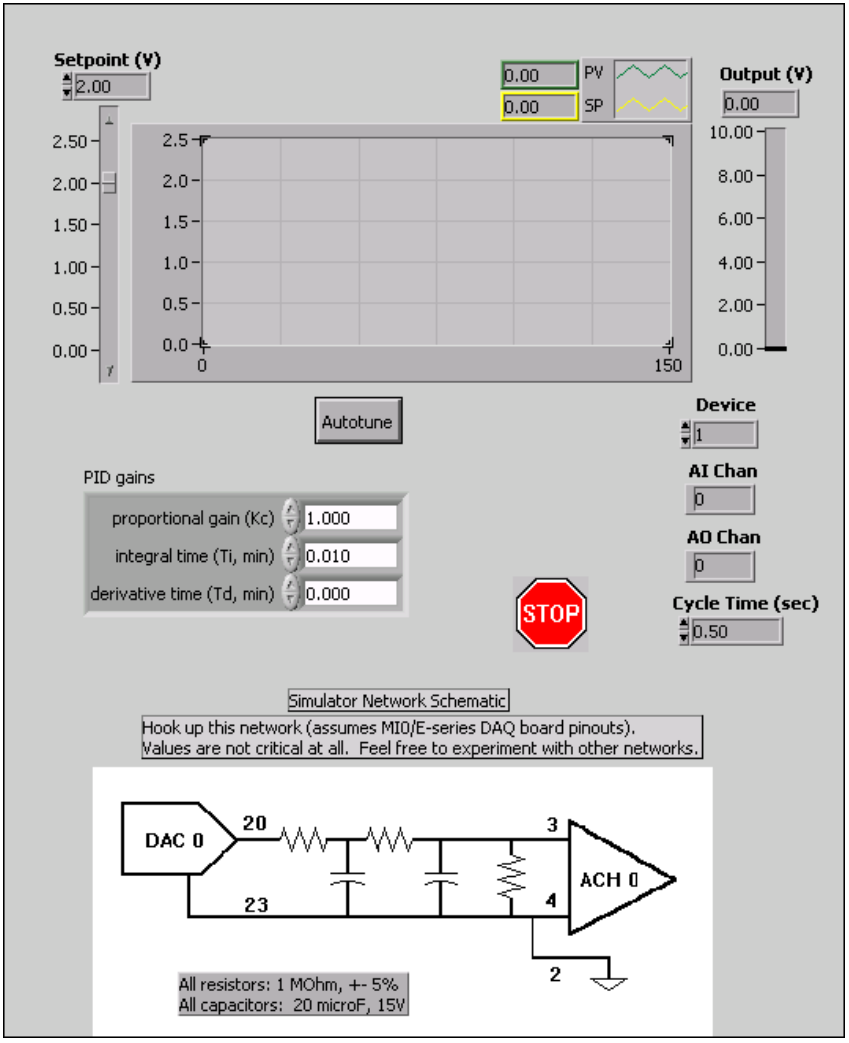


Figure 4-9. Front Panel of the PID VI with Controls Set for an MIO Data Acquisition Board

This VI adjusts the analog output so that the input, PV, equals the SP. The VI displays SP and PV on a strip chart. You can experiment with different controller tuning methods to find the fastest settling time with the least

amount of overshoot. The default tuning parameters are optimal for the network shown in Figure 4-10.

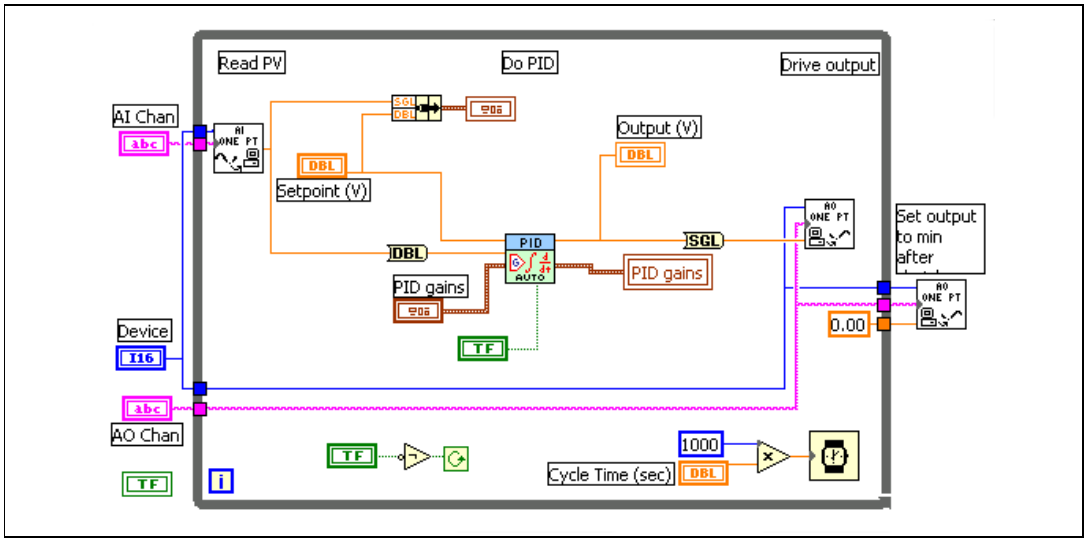


Figure 4-10. Block Diagram of the PID VI with Controls Set for an E Series DAQ Device

The input span is –10 to 10 V and the output span is 0 to 10 V. Both PV and SP are expressed in volts. Set the analog input of the DAQ device to differential input mode in the ±10 V range, and set the output to bipolar in the 10 V range; these are all factory defaults. If you use other settings, change the block diagram constants for the DAQ device configuration to correspond with the new settings.

The recommended network has a DC gain of 0.33, an effective deadtime of about 5 s, and an effective time constant of about 30 s.

To customize this demonstration VI, add alarm limits that set the digital output lines on the I/O board, use one of the analog inputs to set the remote SP, and use one of the digital inputs to set remote automatic to manual switching.

Note Try replacing the PID VI with the PID with autotuning VI to see the effect of autotuning the controller.

Lead-Lag

The Lead-Lag Example VI, shown in Figure 4-11, uses either a sine wave or a square wave for excitation. The waveform is synchronized to the **Cycle Time** you choose. When you vary the tuning parameters, you can see the time-domain response of the Lead-Lag Example VI. A large **Lead** setting causes a wild ringing on the output, while a large **Lag** setting heavily filters the signal, making it almost disappear.

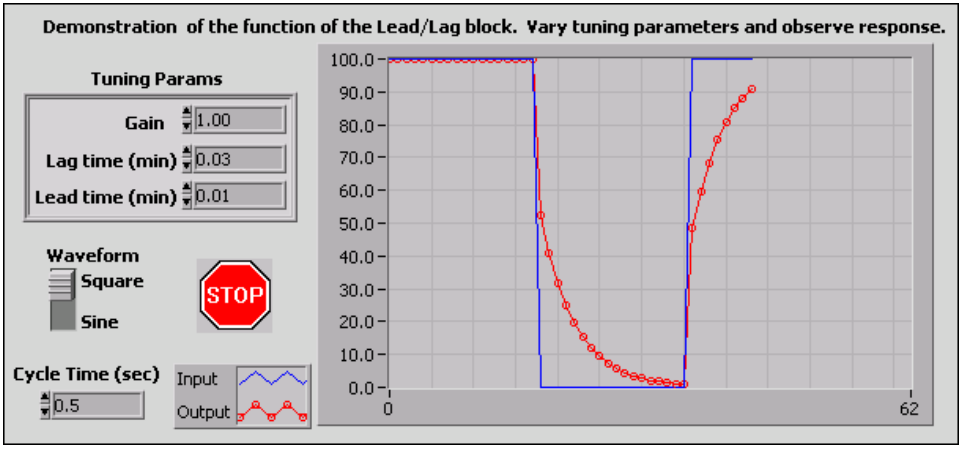


Figure 4-11. Front Panel of the Lead-Lag Example VI

Part II

Fuzzy Logic Control

This section of the manual describes the Fuzzy Logic portion of the PID Control Toolset.

- Chapter 5, *Overview of Fuzzy Logic*, introduces fuzzy set theory and fuzzy logic control.
- Chapter 6, *Fuzzy Controllers*, describes different implementations of fuzzy controllers and the I/O characteristics of fuzzy controllers.
- Chapter 7, *Design Methodology*, provides an overview of the design methodology of a fuzzy controller.
- Chapter 8, *Using the Fuzzy Logic Controller Design VI*, describes how to use Fuzzy Logic VIs to design a fuzzy controller.
- Chapter 9, *Implementing a Fuzzy Controller*, describes how to use Fuzzy Logic VIs to implement the custom controller in your applications.

5

Overview of Fuzzy Logic

This chapter introduces fuzzy set theory and provides an overview of fuzzy logic control.

What is Fuzzy Logic?

Fuzzy logic is a method of rule-based decision making used for expert systems and process control that emulates the rule-of-thumb thought process human beings use. Lotfi Zadeh developed fuzzy set theory, the basis of fuzzy logic, in the 1960s. Fuzzy set theory differs from traditional Boolean set theory in that fuzzy set theory allows for partial membership in a set.

Traditional Boolean set theory is two-valued in the sense that a member either belongs to a set or does not, which is represented by a one or zero, respectively. Fuzzy set theory allows for partial membership, or a degree of membership, which might be any value along the continuum of zero to one.

You can use a a type of fuzzy set called a *membership function* to quantitatively define a linguistic term. A membership function specifically defines degrees of membership based on a property such as temperature or pressure. With membership functions defined for controller or expert system inputs and outputs, you can formulate a rule base of IF-THEN type conditional rules. Then, with fuzzy logic inference, you can use the rule base and corresponding membership functions to analyze controller inputs and determine controller outputs.

After you define a fuzzy controller, you can quickly and easily implement process control. Most traditional control algorithms require a mathematical model to work on, but 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 an expert can qualitatively describe a control strategy, you can use fuzzy logic to define a controller that emulates the heuristic rule-of-thumb strategies of the expert. Therefore, you can use fuzzy logic to control a process that a human manually controls with knowledge he gains from experience. You can directly translate from the linguistic control rules developed by a human expert to a rule base for a fuzzy logic controller.

Types of Uncertainty

Real world situations are often too uncertain or vague for you to describe them precisely. Thoroughly describing a complex situation requires more detailed data than a human being can recognize, process, and understand.

When you apply fuzzy logic concepts, there are the following different types of uncertainty: stochastic, informal, and linguistic.

Stochastic uncertainty is the degree of uncertainty that a certain event will occur. The event itself is well-defined, and the stochastic uncertainty is not related to when the event occurs. This type of uncertainty is used to describe only large-numbered phenomena.

Informal uncertainty results from a lack of information and knowledge about a situation.

Linguistic uncertainty results from the imprecision of language. *Much greater*, *too high*, and *high fever* describe subjective categories with meanings that depend on the context in which you use them.

Modeling Linguistic Uncertainty with Fuzzy Sets

One of the basic concepts in fuzzy logic is the use of fuzzy sets to mathematically describe linguistic uncertainty. People often must make decisions based on imprecise, subjective information. Even when the information does not contain precise quantitative elements, people can use fuzzy sets to successfully manage complex situations.

You do not need to have well-defined rules to make decisions. Most often, you can use rules that cover only a few distinct cases to approximate similar rules that apply them to a given situation. The flexibility of the rules makes this approximation possible.

For example, if the family doctor agrees to make a house call if a sick child has a high fever of 102 °F, you would definitely summon the doctor when the thermometer reads 101.5 °F.

However, you cannot use conventional dual logic to satisfactorily model this situation because the patient with a body temperature of 101.5 °F does not fulfill the criterion for suffering from a high fever, and thus conventional dual logic tells you not to call the doctor. Figure 5-1 shows a graphical representation of the set.

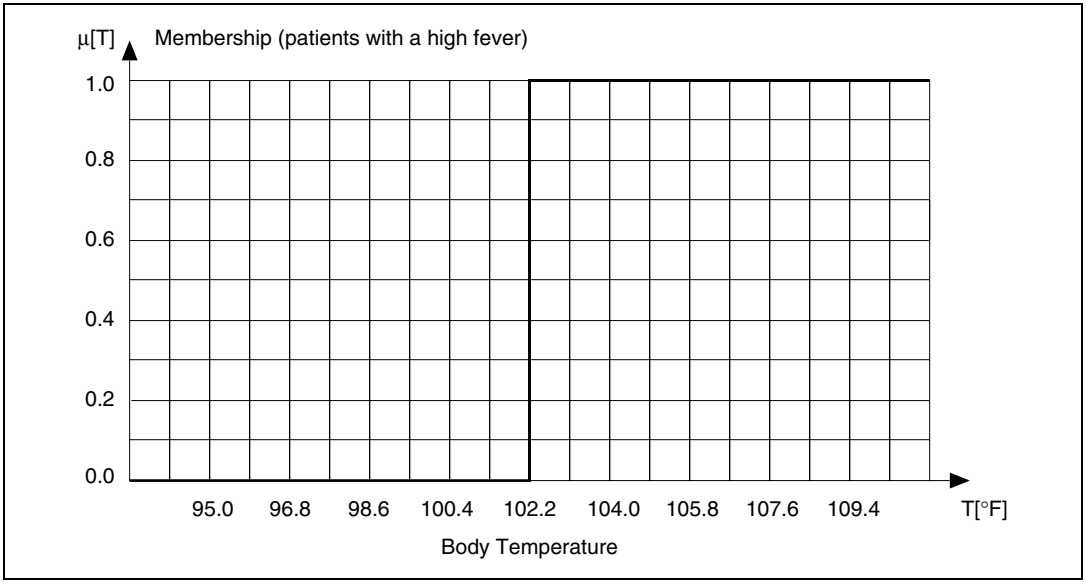


Figure 5-1. Modeling Uncertainty by Conventional Set Membership

Even if you measured the body temperature with an accuracy of up to five decimal places, the situation remains the same. The higher precision does not change the fact that patients with a body temperature below 102 °F do not fit into the category of patients with a high fever, while all patients with a body temperature of 102 °F and higher fully belong to that category.

Modeling uncertain facts, such as *high fever*, sets aside the strict distinction between the two membership values one, TRUE, and zero, FALSE, and instead allows arbitrary intermediate membership degrees. With respect to conventional set theory, you can generalize the set notion by allowing elements to be more-or-less members of a certain set. This type of set is known as a fuzzy set. Figure 5-2 shows a graphical representation of the set.

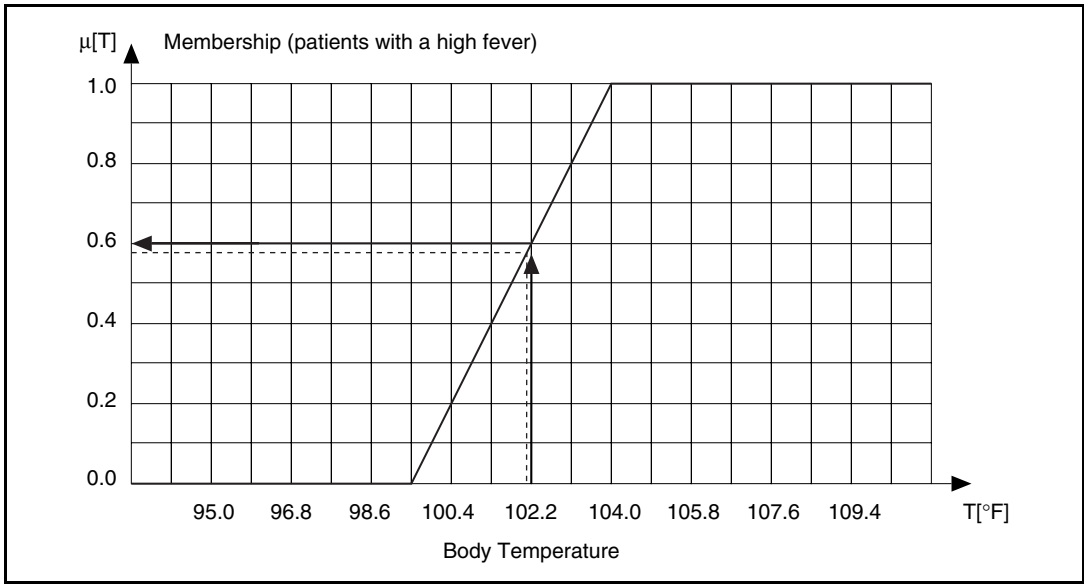


Figure 5-2. Modeling Uncertainty by Fuzzy Set Membership

In Figure 5-2, the graph associates each body temperature with a certain degree of membership (μ(T)) to the *high fever* set. The function μ(T) is called the degree of membership of the element (T ∈ BT) to the fuzzy set *high fever*. The body temperature is called the characteristic quantity or base variable T of the universe BT. Notice that μ ranges from zero to one, the values representing absolutely no membership to the set and complete membership, respectively.

You also can interpret the degree of membership to the fuzzy set *high fever* as the degree of truth given to the statement that the patient suffers from high fever. Thus, using fuzzy sets defined by membership functions within logical expressions leads to the notion of Fuzzy Logic.

As shown in Figure 5-2, a continuous function μ(T), often called a fuzzy set, represents the degree of membership. Refer to the *Defining Linguistic Variables* section of Chapter 7, *Design Methodology* for more information about how to define membership functions for certain applications.

Notice that a body temperature of 102 °F is considered only slightly different from a body temperature of 101.5 °F, and not considered a threshold.

Linguistic Variables and Terms

The primary building block of fuzzy logic systems is the linguistic variable. A linguistic variable is used to combine multiple subjective categories that describe the same context. In the previous example, there is *high fever* and *raised* temperature as well as *normal* and *low* temperature in order to specify the uncertain and subjective category body temperature. These terms are called linguistic terms and represent the possible values of a linguistic variable. A fuzzy set defined by a membership function represents each linguistic term.

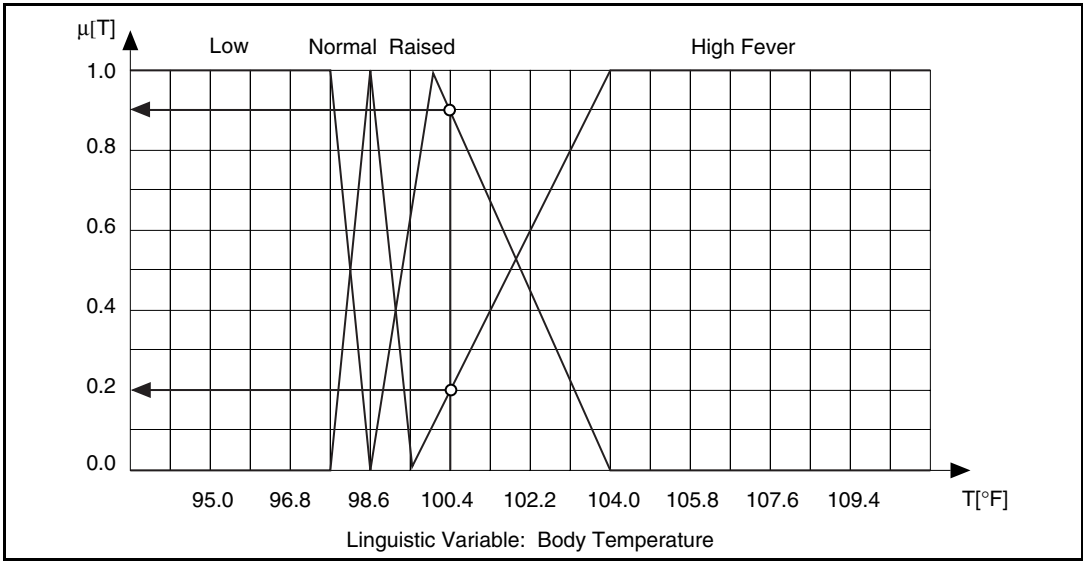


Figure 5-3. A Linguistic Variable Translates Real Values into Linguistic Values

The linguistic variable shown in Figure 5-3 allows for the translation of a crisp measured body temperature, given in degrees Fahrenheit, into its linguistic description. A doctor might evaluate a body temperature of 100.5 °F, for example, as a *raised* temperature, or a slightly *high fever*. The overlapping regions of neighboring linguistic terms are important when you use linguistic variables to model engineering systems.

Rule-Based Systems

Another basic fuzzy logic concept involves rule-based decision-making processes. You do not always need a detailed and precise mathematical description to optimize operation of an engineering process. In other words, human operators are often capable of managing complex plant situations without knowing anything about differential equations. Their engineering knowledge is perhaps available in a linguistic form such as “if the liquid temperature is correct, and the pH value is too high, adjust the water feed to a higher level.”

Because of fully-developed nonlinearities, distributed parameters, and time constants that are difficult to determine, it is often impossible for a control engineer to develop a mathematical system model. Fuzzy logic uses linguistic representation of engineering knowledge to implement a control strategy.

Suppose you must automate the maneuvering process that leads a truck from an arbitrary starting point to a loading ramp. The truck should run at a constant low speed and stop immediately when it docks at the loading ramp. A human driver is capable of controlling the truck by constantly evaluating the current drive situation, mainly defined by the distance from the target position and the orientation of the truck, to derive the correct steering angle. This is shown in Figure 5-4.

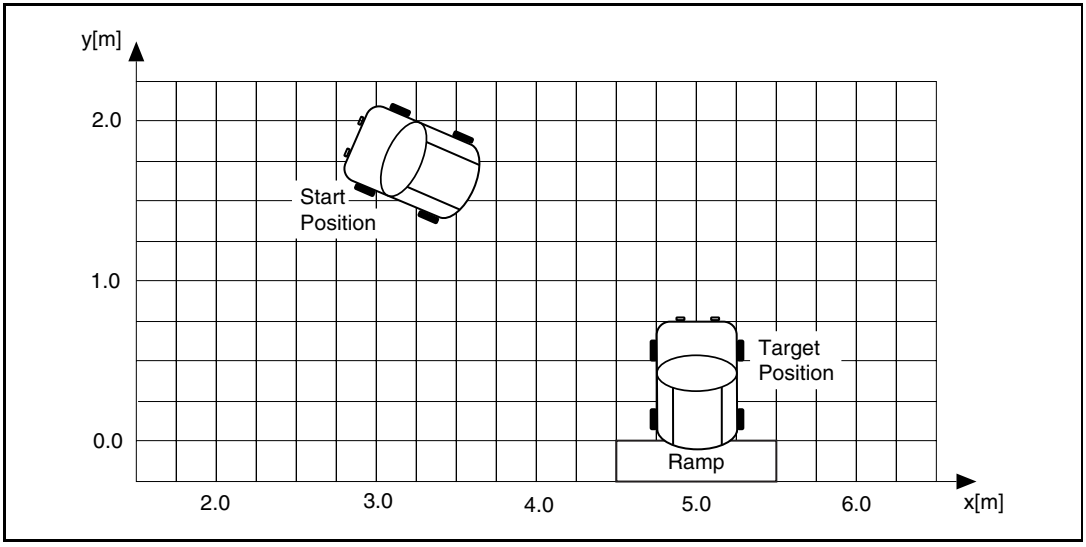


Figure 5-4. Automation of a Maneuvering Process Example

Implementing a Linguistic Control Strategy

To automate the truck control, an ultrasonic distance sensor monitors the truck position in *x*-direction, and an electronic compass monitors the truck orientation. Each drive situation is identified by at least two conditions. The first condition describes the vehicle position *x* from the loading ramp, and the second condition describes the vehicle orientation β. The conditions are combined with the word AND, which represents the fact that both conditions must be valid for the respective situation.

Figure 5-5 shows a description of a vehicle position left from the target center with a left-hand orientation β, and a large negative steering angle φ with the steering wheel turned all the way to the left.

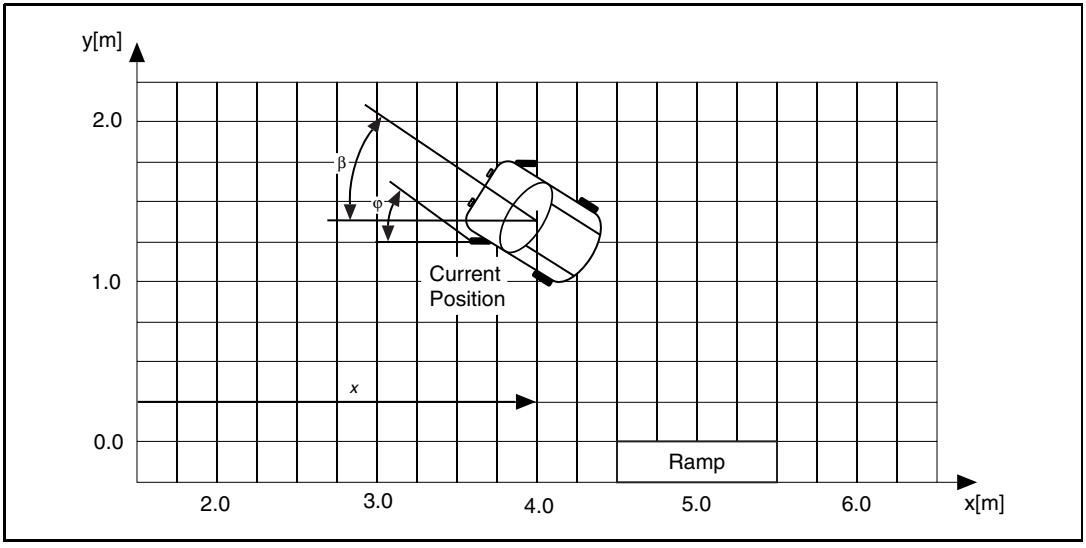


Figure 5-5. Condition: Vehicle Position *x* and Orientation β, Action: Steering Angle φ

You can then use IF-THEN rules, such as

IF <situation> THEN <action>

to define a control strategy.

The above rule format describes the necessary reaction, or conclusion, to a certain situation, or condition.

An expert driver could tell you the rules of thumb he uses to maneuver the vehicle to the target position. Then you can describe those rules with IF-THEN rules.

IF *vehicle position x* is *left center* AND *vehicle orientation* β is *left up* THEN adjust *steering angle* φ to *positive small*,

or

IF *vehicle position x* is *center* AND *vehicle orientation* β is *left up* THEN adjust *steering angle* φ to *negative small*,

or

IF *vehicle position x* is *left center* AND *vehicle orientation* β is *up* THEN adjust *steering angle* φ to *positive medium*,

or

IF *vehicle position x* is *center* AND *vehicle orientation* β is *up* THEN adjust *steering angle* φ to *zero*.

Note Uncertain linguistic terms like *left center*, *left up*, and so on, compose the conditions of each rule. Even the conclusion of each rule contains vague and imprecise facts such as *negative small*. Because there are no precise definitions of the words used in the rules above, there is no way to use a text-based programming language to directly implement the rules with IF-THEN statements.

You can use fuzzy logic to implement a linguistic control strategy that is capable of using fuzzy sets to model uncertain linguistic facts like *left center* or *high fever.*

First, you must define a linguistic variable for each characteristic quantity of the maneuvering process. For example, *vehicle position x* and *vehicle orientation* β are process or input variables, and *steering angle* φ is an output variable.

A linguistic variable consists of a number of linguistic terms that describe the different linguistic interpretations of the characteristic quantity you are modeling. The appropriate membership function defines each linguistic term.

Figures 5-6, 5-7, and 5-8 show membership functions for the inputs and output of the truck controller.

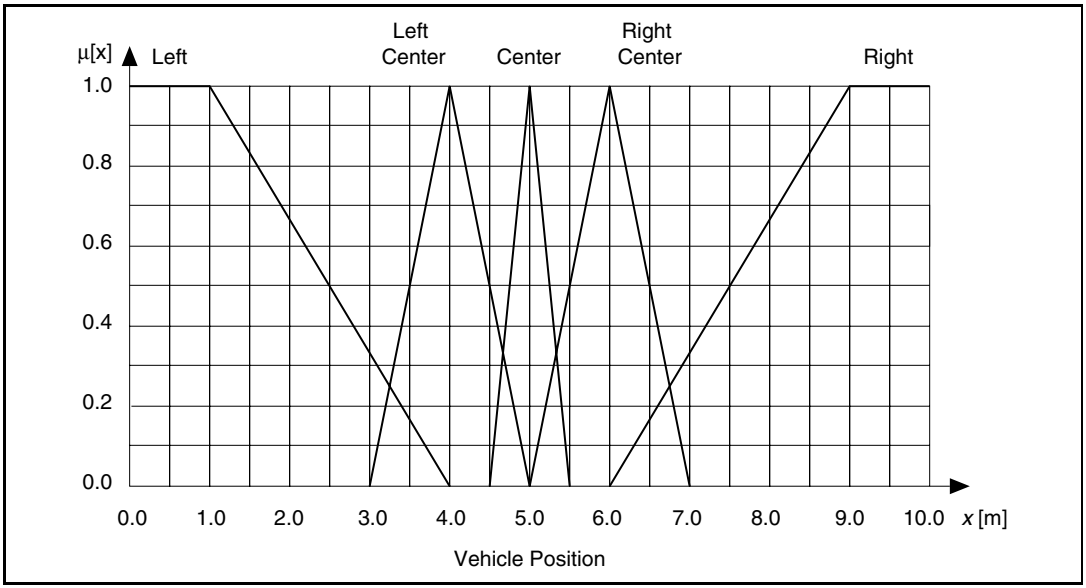


Figure 5-6. Linguistic Variable Vehicle Position x and Its Linguistic Terms

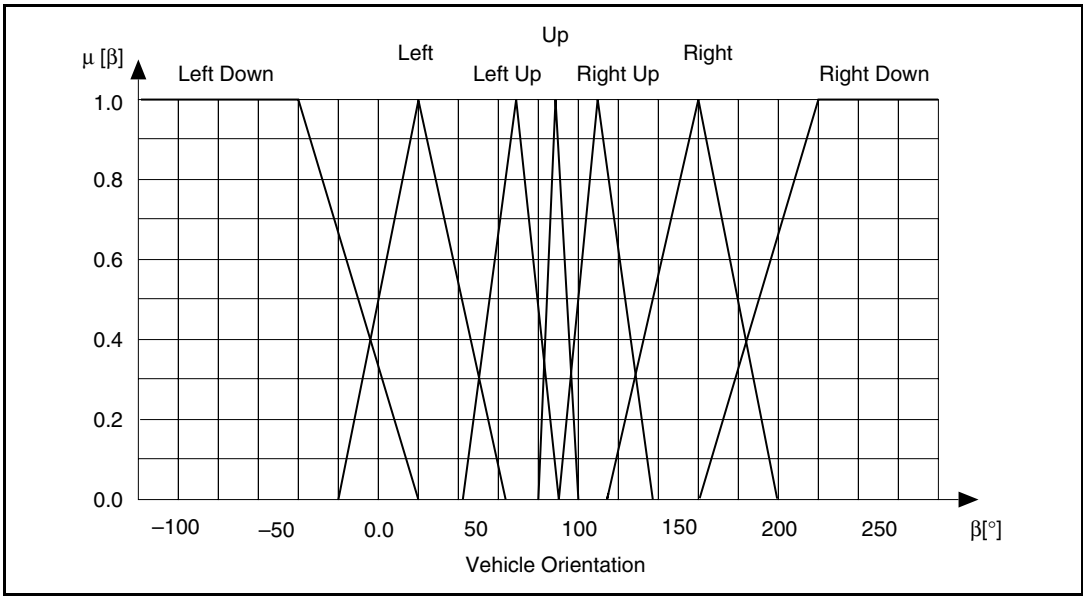


Figure 5-7. Linguistic Variable Vehicle Orientation β and Its Linguistic Terms

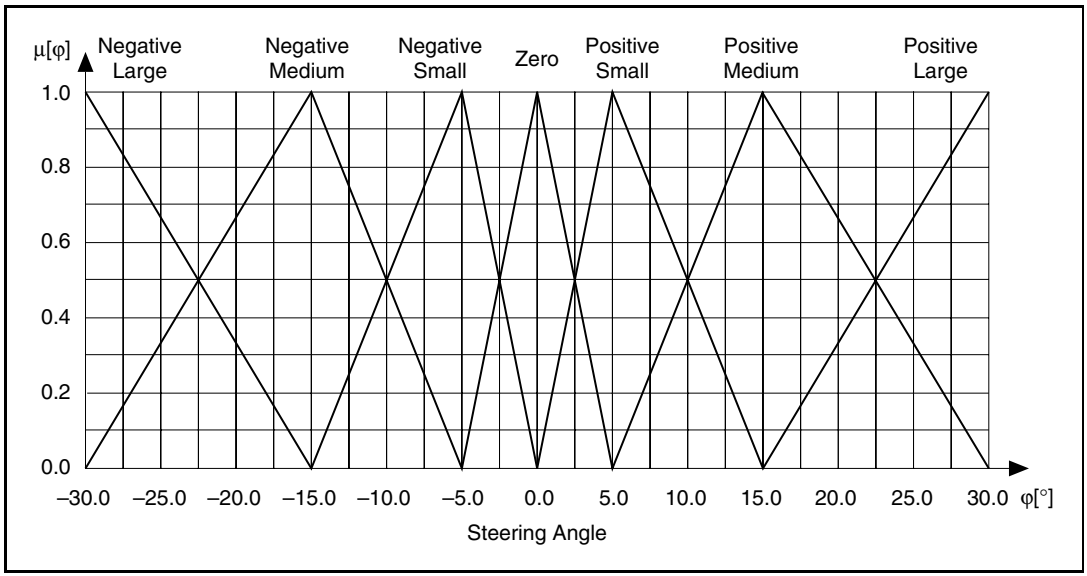


Figure 5-8. Linguistic Variable Steering Angle φ and Its Linguistic Terms

IF *vehicle position x* is *center* AND *vehicle orientation* β is *up* THEN adjust *steering angle* φ to *zero*,

In the above rule of the linguistic control strategy, the condition is composed of the linguistic term *center* from the linguistic variable *vehicle position x*, and the linguistic term *up* from the linguistic variable *vehicle orientation* β, combined by the AND operator.

Because there are five terms for *vehicle position x* and seven terms for *vehicle orientation* β, there are at most $N = 35$ different rules available to form a consistent rule base. Because there are only two input variables in this case, you can document the complete rule base in matrix form, as shown in Figure 5-9.

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 5-9. Complete Linguistic Rule Base

Each combination of a column and a row describes a specific maneuvering situation, the condition of a certain rule. The term at the intersection of the column and row is the conclusion.

As an example, the following rule is highlighted in Figure 5-9.

IF *vehicle position x* is *left center* AND *vehicle orientation* β is *left* THEN adjust *steering angle* φ to *negative small*.

Structure of the Fuzzy Logic Vehicle Controller

The complete structure of a fuzzy logic controller is shown in Figure 5-10.

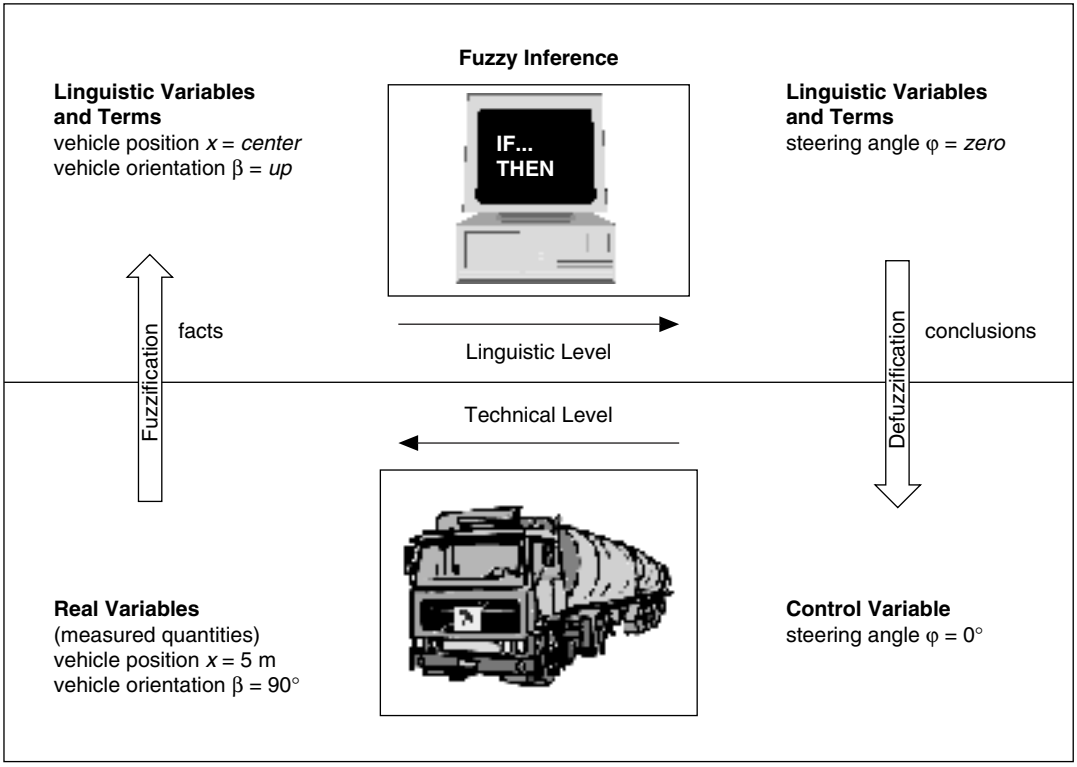


Figure 5-10. Complete Structure of a Fuzzy Controller

In the first step, you must translate all sensor signals into linguistic variables. For example, you must translate a measured *vehicle position x* of 4.8 m to the linguistic value *almost center, just slightly left center.* This step is called *fuzzification* because it uses fuzzy sets to translate real variables into linguistic variables.

Once you translate all input variable values into their corresponding linguistic variable values, use the fuzzy inference step to derive a conclusion from the rule base that represents the control strategy. The step results in a linguistic value for the output variable. For example, the linguistic result for steering angle adjustment might be *steering angle φ a little less than zero*.

The defuzzification step translates the linguistic result back into a real value that represents the current value of the control variable.

Fuzzification Using Linguistic Variables

For a more detailed look at the fuzzification process, consider a maneuvering situation in which the vehicle position *x* is 5.1 m and the vehicle orientation β is 70°.

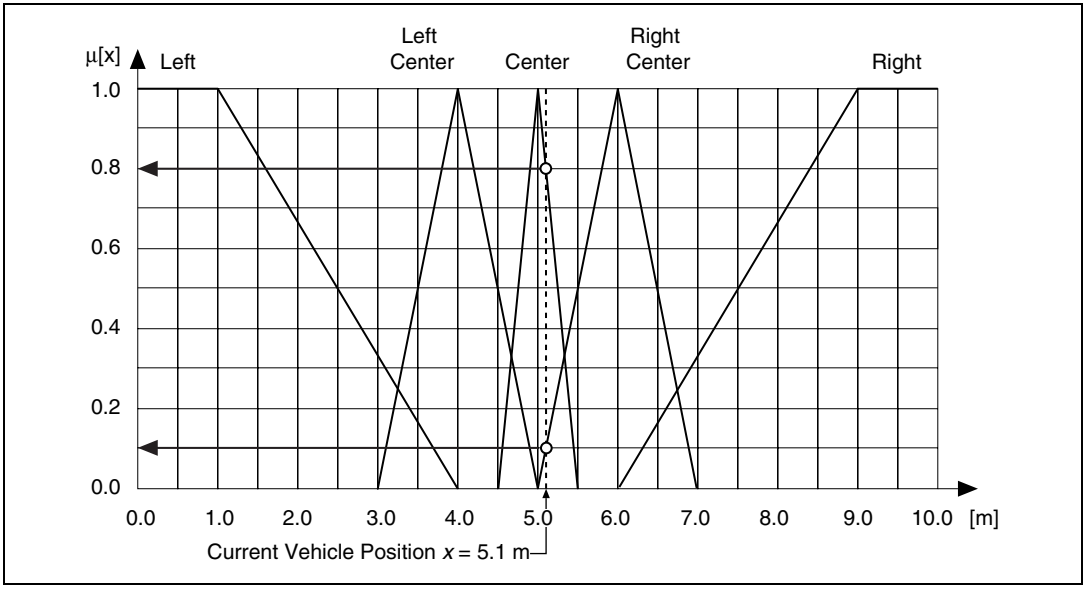


Figure 5-11. Fuzzification of the Vehicle Position *x* = 5.1 m

The current vehicle position *x* = 5.1 m belongs to the following linguistic terms, which are defined by fuzzy sets:

left	with a degree of	0.0
left center	with a degree of	0.0
center	with a degree of	0.8
right center	with a degree of	0.1
right	with a degree of	0.0

The current vehicle position of 5.1 m is translated into the linguistic value {0.0, 0.0, 0.8, 0.1, 0.0}, which you can interpret as *still center, just slightly right center.*

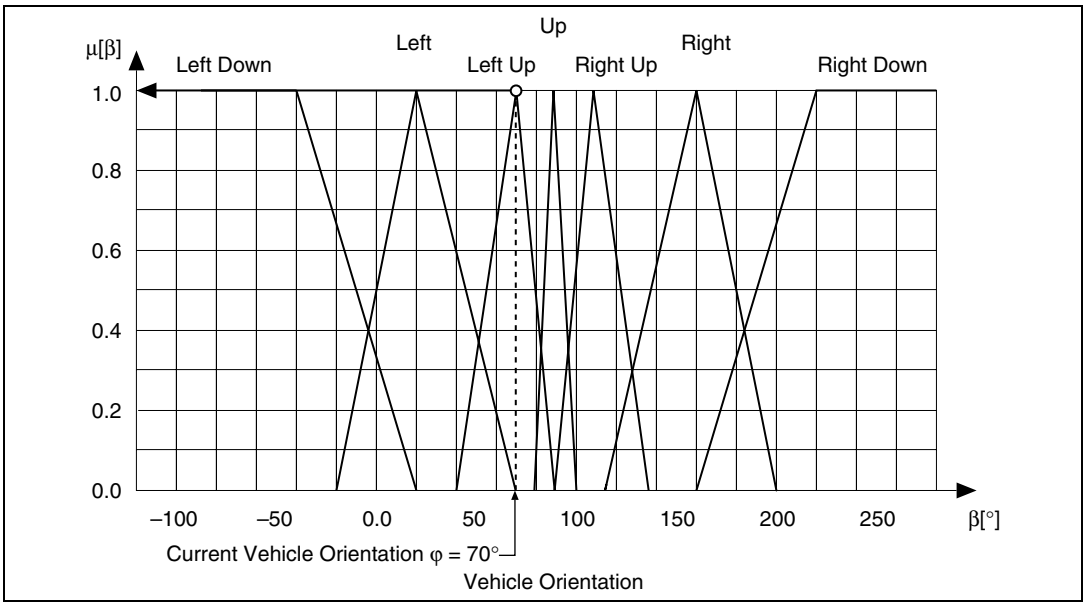


Figure 5-12. Fuzzification of the Vehicle Orientation φ = 70°

The current vehicle orientation φ = 70° belongs to the following linguistic terms (fuzzy sets):

left down	with a degree of	0.0
left	with a degree of	0.0
left up	with a degree of	1.0
up	with a degree of	0.0
right up	with a degree of	0.0
right	with a degree of	0.0
right down	with a degree of	0.0

The current vehicle orientation of 70° is translated into the linguistic value {0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0}, which you can interpret as *left up*.

Refer to Chapter 7, *Design Methodology*, for more information about defining linguistic terms and memberships.

Using IF-THEN Rules in Fuzzy Inference

After you convert all physical input values into linguistic values, identify all rules from the rule base that apply to the current maneuvering situation. Identify these rules so you can calculate the values of the linguistic output variable. The fuzzy inference step consists of two components.

Aggregation involves the evaluation of the IF part, condition, of each rule. Composition involves the valuation of the THEN part, conclusion, of each rule.

In the following example, notice that the IF part of each rule logically combines two linguistic terms from different linguistic variables with the conjunction AND. Because the linguistic terms represent conditions that are partially true, the Boolean AND from conventional dual logic is not an appropriate choice to model the conjunction AND. You must define new operators that represent logical connections such as AND, OR, and NOT.

The three operators used in the majority of fuzzy logic applications are defined as follows:

AND:
$\mu A \bullet B = \mathbf{min}(\mu A, \mu B)$

OR:
$\mu A + B = \mathbf{max}(\mu A, \mu B)$

NOT:
$\mu \neg A = 1 - \mu A$

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

The minimum operator represents the word AND. Apply AND in the aggregation step to calculate a degree of truth for the IF condition of each rule in the rule base that indicates how adequately each rule describes the current situation.

In the example situation, only the following two rules are valid descriptions of the current situation. These rules are usually called the active rules. All the other rules are called inactive.

(1) IF *vehicle position x* is *center* (degree of truth = 0.8) AND minimum *vehicle orientation b* is *left up* (degree of truth = 1.0) = 0.8

THEN adjust steering angle φ to *negative small*

(2) IF vehicle position *x* is *right center* (degree of truth = 0.1) AND minimum *vehicle orientation* β is *left up* (degree of truth = 1.0) = 0.1

THEN adjust *steering angle* φ to *negative small*

Each rule defines an action to take in the THEN condition. The applicability of the rule to the current situation determines the degree to which the action is valid. The aggregation step calculates this adequacy as the degree of truth of the IF condition.

In this case, the first rule results in the action "adjust *steering angle* φ to *negative small*" with a degree of 0.8. The second rule results in the action "adjust steering angle φ to *negative medium*" with a degree of 0.1.

The composition step ensures that the resulting action is composed of the differently weighted THEN conclusions of the active rules.

The rules of this rule base are defined alternatively, which means that they are logically linked by the word OR. Because the resulting conclusions of the rules are partially true, you cannot use the OR operator from conventional dual logic to calculate the resulting conclusion. In fuzzy logic, you must use the maximum operator instead.

For example, assume that two rules assert different degrees of truth for the linguistic term *positive medium*. One rule asserts *positive medium* with a degree of truth of 0.2, while another asserts *positive medium* with a degree of truth of 0.7. Because the OR operator relates two rules to each other, the output of the fuzzy inference for the linguistic term is the maximum value of 0.7. Because the truck example has only one rule asserting a nonzero degree of truth for both *negative medium* and *negative small*, those values become the maximum values you use.

The final result of the fuzzy inference for the linguistic variable *steering angle* φ includes the following linguistic terms and their corresponding values:

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

This type of fuzzy inference is called Max-Min inference. Because of certain optimization procedures of fuzzy systems, sometimes it is necessary to associate individual weights with each rule.

Using Linguistic Variables in Defuzzification

The fuzzy inference process results in a linguistic value for the output variable. In this case, you can interpret the linguistic value {0.0, 0.1, 0.8, 0.0, 0.0, 0.0,0.0} as *still negative small* or *just slightly negative medium*. To use this linguistic value to adjust the steering wheel, you must translate it into a real, physical value. This step is called defuzzification. Refer to Figure 5-10 for a diagram of the different steps.

The membership functions that describe the terms of the linguistic output variable always define the relationship between the linguistic values and the corresponding real values. Refer to Figure 5-8 for more information about membership functions. In the example, you obtain a fuzzy inference result that is both fuzzy and ambiguous because you acquire the nonzero truth degree of two different actions at the same time. You must combine two conflicting actions, defined as fuzzy sets, to form a crisp real value. A solution to this problem is to find the best compromise between the two different goals. This compromise represents the best final conclusion received from the fuzzy inference process.

One of the two most common methods for calculating the best compromise is the Center-of-Area (CoA) method, also called the Center-of-Gravity (CoG) method.

Following this defuzzification method, truncate all membership functions that represent the conclusion terms at the degree of validity of the rule to which the conclusion term belongs. The areas under the resulting function of all truncated terms make up the grey area of Figure 5-13. Find the geometric center of this area to determine the crisp compromise value.

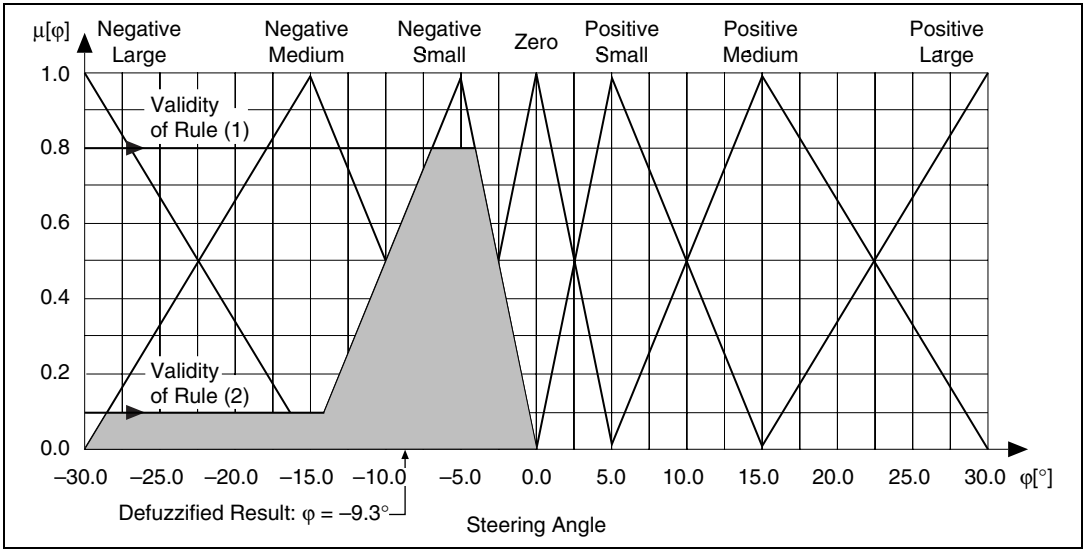


Figure 5-13. Defuzzification According to Center-of-Area (CoA)

The numerical integration necessary to calculate the center-of-area in this defuzzification method requires a lot of computation.

The second defuzzification method is called the Center-of-Maximum (CoM) method. In the first step of this method, determine the typical value of each term in the linguistic output variable. In the second step, calculate the best compromise with a weighted average of typical values of the terms.

The most common approach to determining the typical value of each term is to find the maximum of the corresponding membership function. In the case of trapezoidal membership functions, choose the median of the maximizing interval.

Weight each typical value by the degree to which the action term, conclusion, is true. Then, calculate the crisp output value with a weighted average as shown in Figure 5-14.

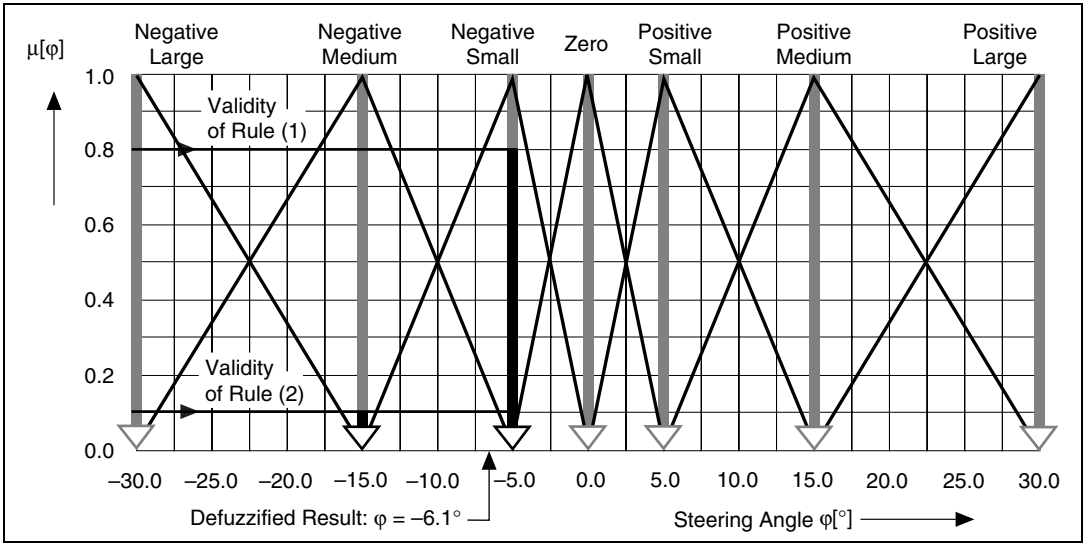


Figure 5-14. Defuzzification According to Center-of-Maximum (CoM)

With φ (*negative medium*) = –15° and φ (*negative small*) = –5° as typical values of the linguistic terms *negative medium* and *negative small*, and with the validity values V (rule 1) = 0.8 and V (rule 2) = 0.1 for the active rules, the possible defuzzification results are:

$$\varphi(\text{out}) = \frac{\varphi(\textit{negative medium}) \bullet \text{V(rule 2)} + \varphi(\textit{negative small}) \bullet \text{V(rule 1)}}{\text{V(rule 2)} + \text{V(rule 1)}}$$

$$\varphi(\text{out}) = -6.1°$$

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

Figure 5-15 summarizes the fuzzy inference process for the maneuvering situation described above using the CoA method of defuzzification.

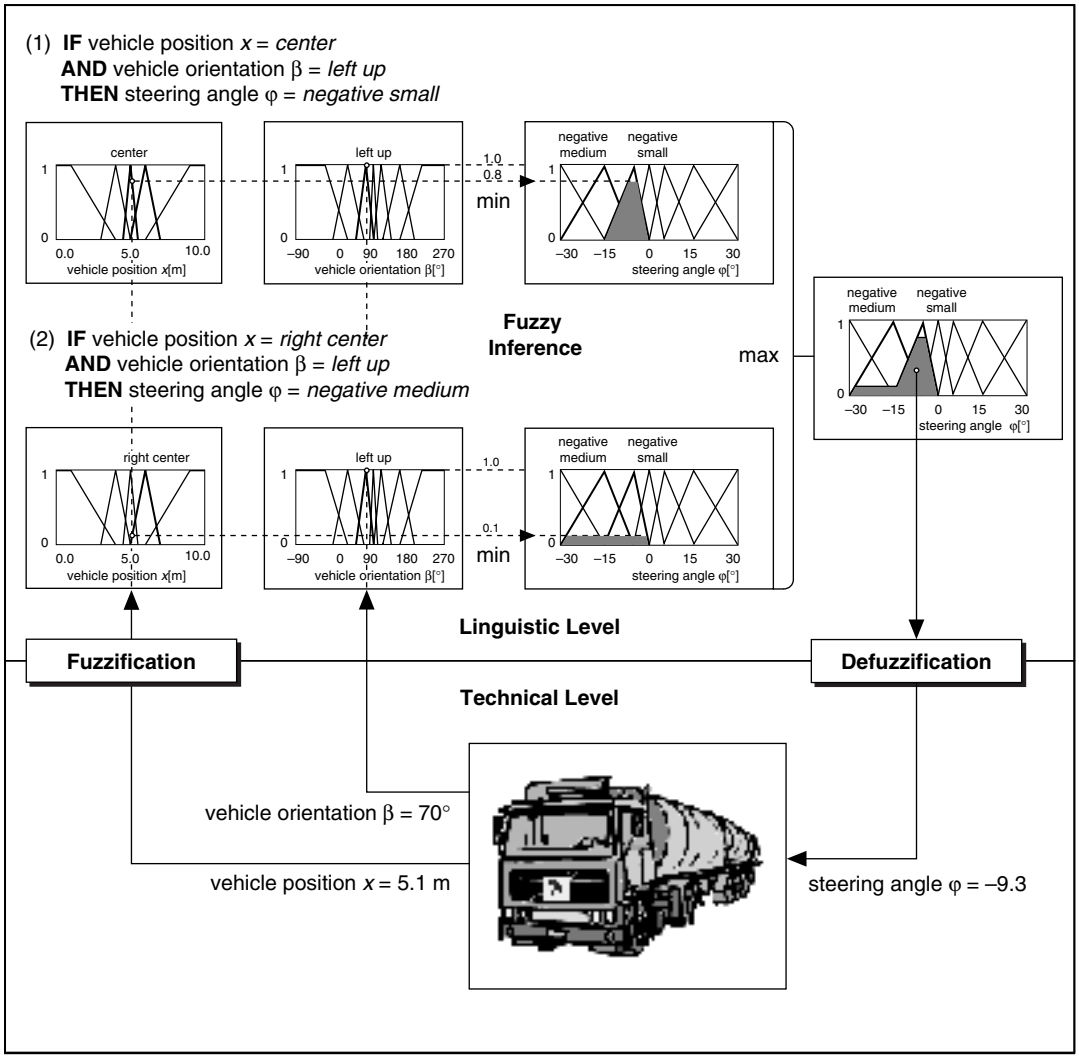


Figure 5-15. Fuzzification, Fuzzy Inference and Defuzzification for a Specific Maneuvering Situation

Without modification, the CoA defuzzification method limits the range of the output value compared to the possible range. To solve this problem, add a fictitious extension of the left and right side border terms when you compute the center of area. With this extension, the output variable can

realize the complete value range, shown in Figure 5-16. In this case the defuzzification method is called modified CoA.

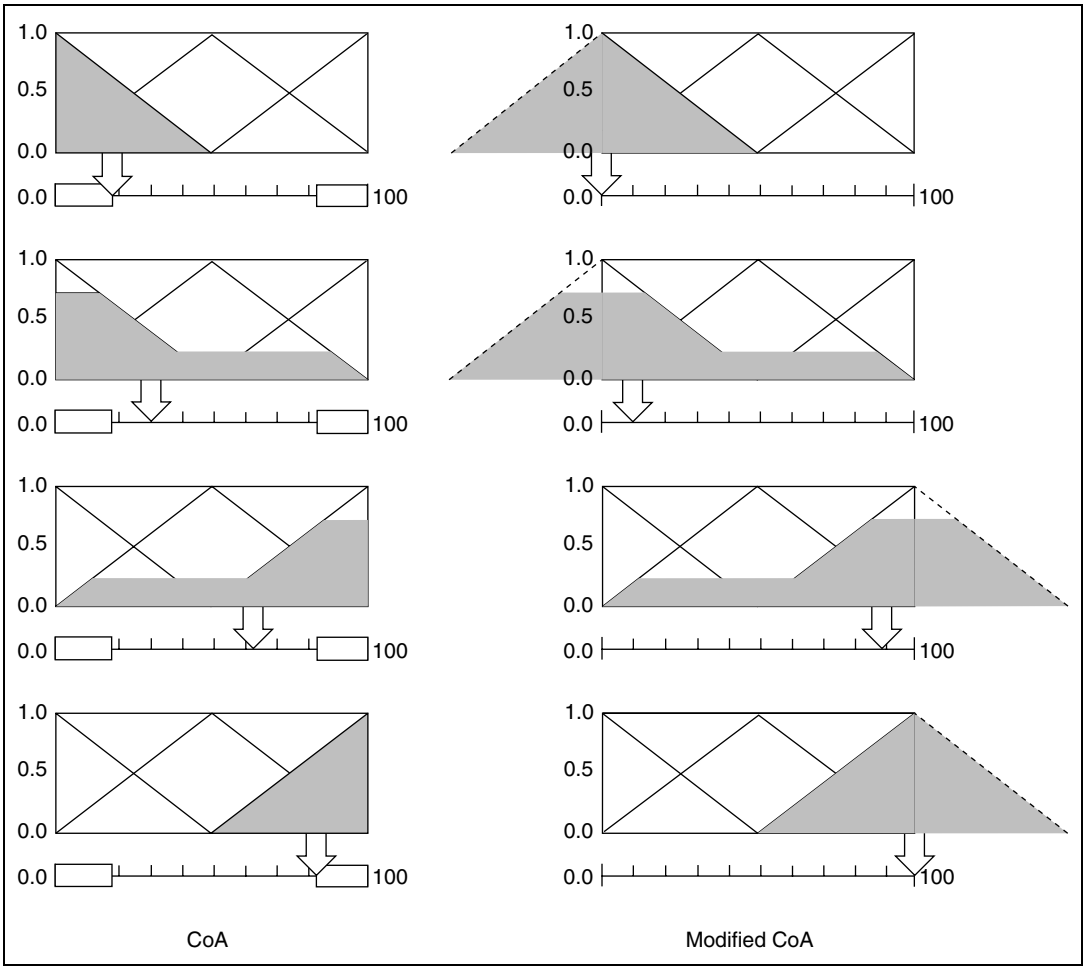


Figure 5-16. Modified CoA for Complete Output Value Range

The CoM and CoA defuzzification methods are usually applied to closed-loop control applications of fuzzy logic. These methods usually lead to continuous output signals because the best compromise can never jump to a different value with a small change to the inputs.

For pattern recognition applications, you must apply the Mean-of-Maximum (MoM) defuzzification method. This defuzzification method calculates the most plausible result. Rather than averaging the

different inference results, MoM selects the typical value of the most valid output term.

In the example situation, the output term *negative small* is the most valid term. Refer to Figures 5-13 and 5-14 for more information. The typical value of the term is φ (*negative small*) = 5°, which is the immediate defuzzification result. If you want to classify a sensor signal to identify objects, for example, you are interested in the most plausible result.

In decision support systems, the choice of the defuzzification method depends on the context of the decision you want to calculate with the fuzzy system. 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.

6

Fuzzy Controllers

This chapter describes various implementations and I/O characteristics of fuzzy controllers.

Structure of a Fuzzy Controller

A fuzzy controller is composed of the following three calculation steps: fuzzification, fuzzy inference, and defuzzification. Linguistic rules integrated into the rule base of the controller implement the control strategy that you base on engineering experience with respect to a closed-loop control application.

A fuzzy controller has a static and deterministic structure, as shown in Figure 6-1, which you can describe with an I/O characteristic curve.

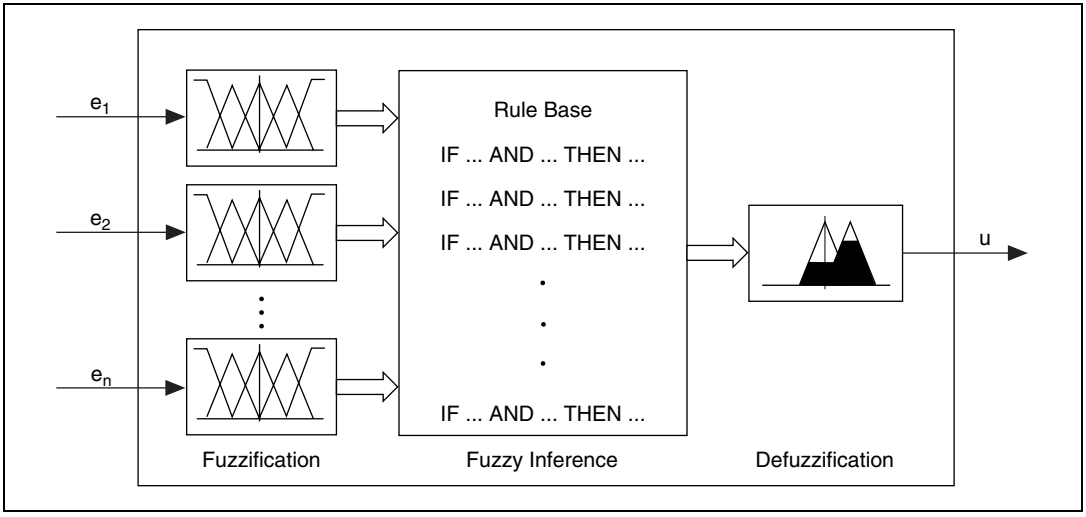


Figure 6-1. Internal Structure of a Fuzzy Controller

In principle, there are two different implementation forms. With the first type of implementation, the Offline Fuzzy Controller, you transform the three-step calculation scheme into a reference table from which you can derive the command values. You can use interpolation to calculate

intermediate command values. In the second type of implementation, the Online Fuzzy Controller, you evaluate the three-step calculation scheme online. This is the standard implementation form of the Fuzzy Logic Controls.

Closed-Loop Control Structures with Fuzzy Controllers

There are many different ways to use fuzzy controllers in closed-loop control applications. The most basic structure uses the sensor signals from the process as input signals for the fuzzy controller and the outputs as command values to drive the actuators of the process. A corresponding control loop structure is shown in Figure 6-2.

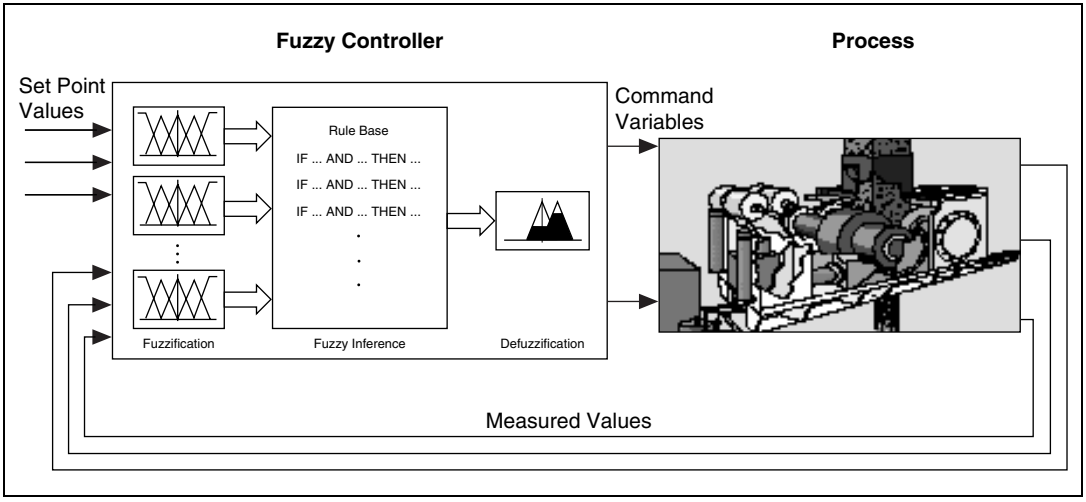


Figure 6-2. Simple Closed-Loop Control Structure with Fuzzy Controller

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

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

The Fuzzy-PI Controller, shown in Figure 6-3, 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.

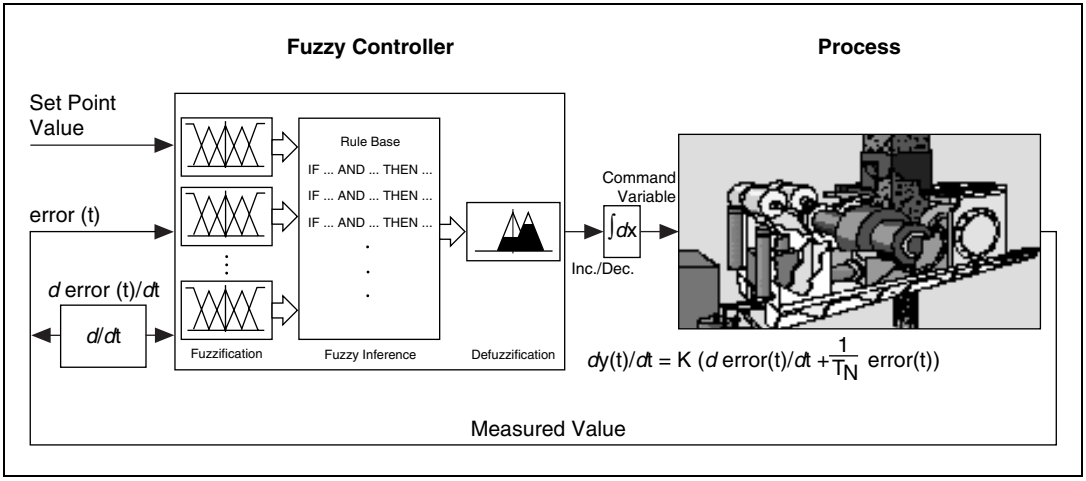


Figure 6-3. Closed-Loop Control Structure with Fuzzy-PI Controller

A Fuzzy-PI Controller is a fuzzy controller with two inputs and one output. The output value increases when the input values increase. If you use an error signal and its derivative as input signals, the Fuzzy-PI Controller is essentially a generalization of the conventional 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.

The 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. It is possible to consider only the error tendency when the error becomes small.

Chemical industry and process technology often use the Fuzzy Controller with Underlying PID Control Loops. This application uses PID controllers to control single process parameters. Usually, human operators supervise the operating point of the entire process.

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

With fuzzy controllers, you can often use the experience and the knowledge gained by the supervising operators to form a linguistic rule base with much less effort. Figure 6-4 shows the controller structure of the Fuzzy Controller with Underlying PID Control Loops.

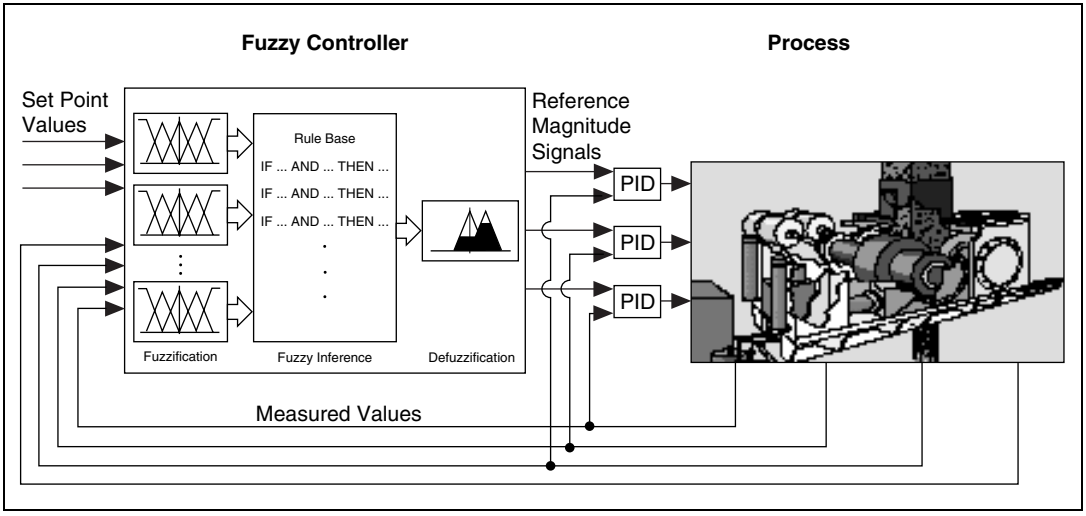


Figure 6-4. Fuzzy Controller with Underlying PID Control Loops

The next example structure shows how to use a fuzzy controller to automatically tune the parameters of a conventional PID controller. For this, the 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. Figures 6-5 and 6-6 show this control structure.

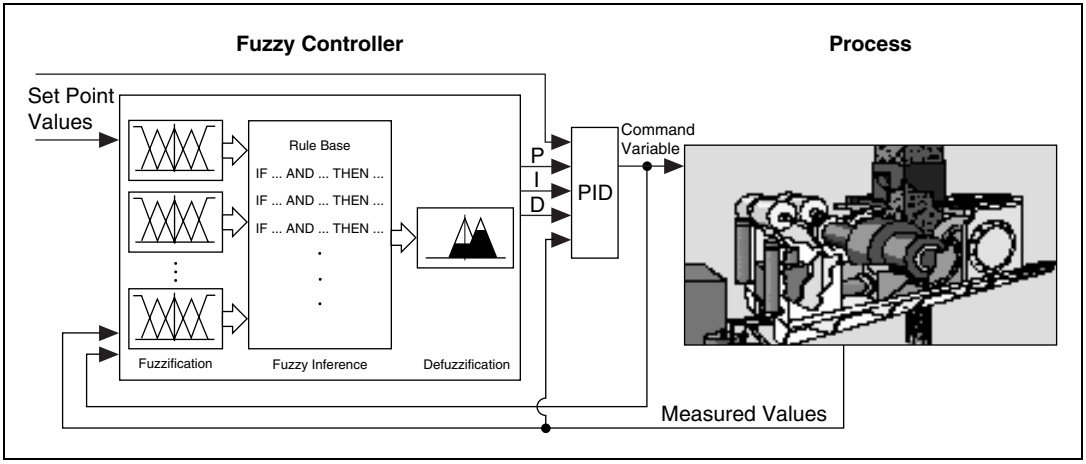


Figure 6-5. Fuzzy Controller for Parameter Adaptation of a PID Controller

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.

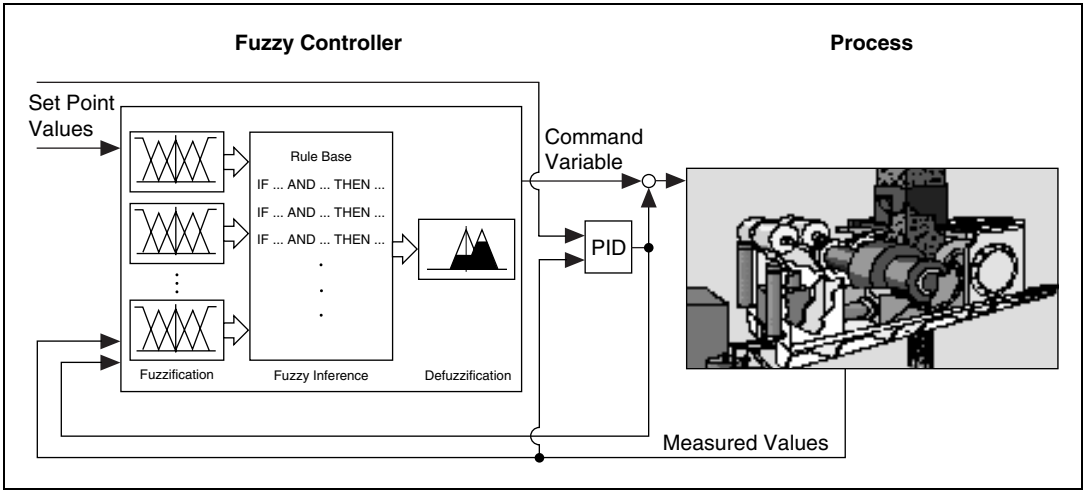


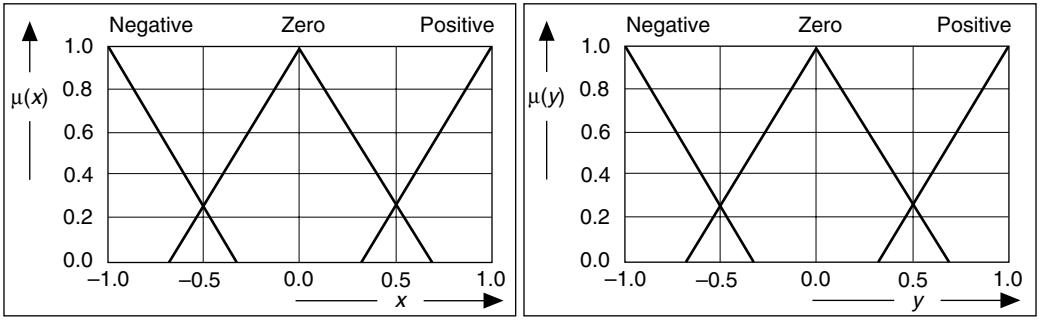
Figure 6-6. Fuzzy Controller for Correction of a PID Controller Output

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 that model the terms of the linguistic input and output variables for the controller determine the behavior of the controller. Because the controller has no internal dynamic aspects, the I/O characteristics can entirely describe the transient response of the controller.

To illustrate how the I/O characteristics of a fuzzy controller depend on design parameters such as rule base and membership function specification, you must first restrict yourself to a single-input fuzzy controller. Most of these ideas apply directly to fuzzy controllers with two or more inputs.

Figure 6-7 shows the I/O characteristic of a fuzzy controller 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 to increase.



Rule Base		
	Rule 1: IF *x* = Negative THEN *y* = Negative Rule 2: IF *x* = Zero THEN *y* = Zero Rule 3: IF *x* = Positive THEN *y* = Positive	Max-Min-Inference Modified CoA

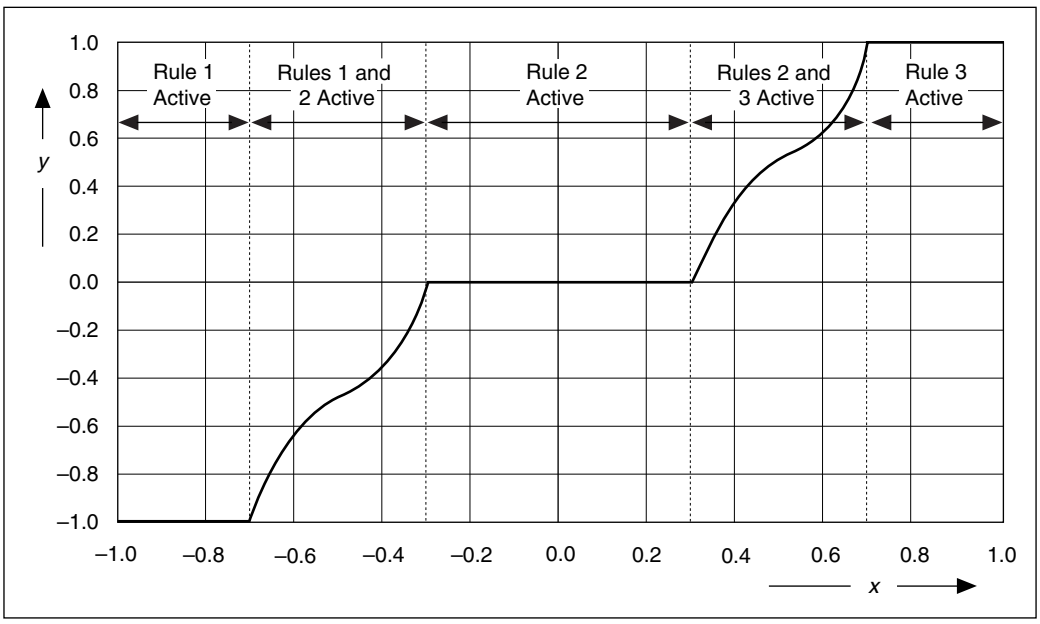


Figure 6-7. I/O Characteristic of a Fuzzy Controller (Partially Overlapping Input Terms)

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

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

Figure 6-8 shows the resulting controller characteristic for antecedence terms that overlap entirely. The conclusion term distribution and the rule base remain unchanged for this case.

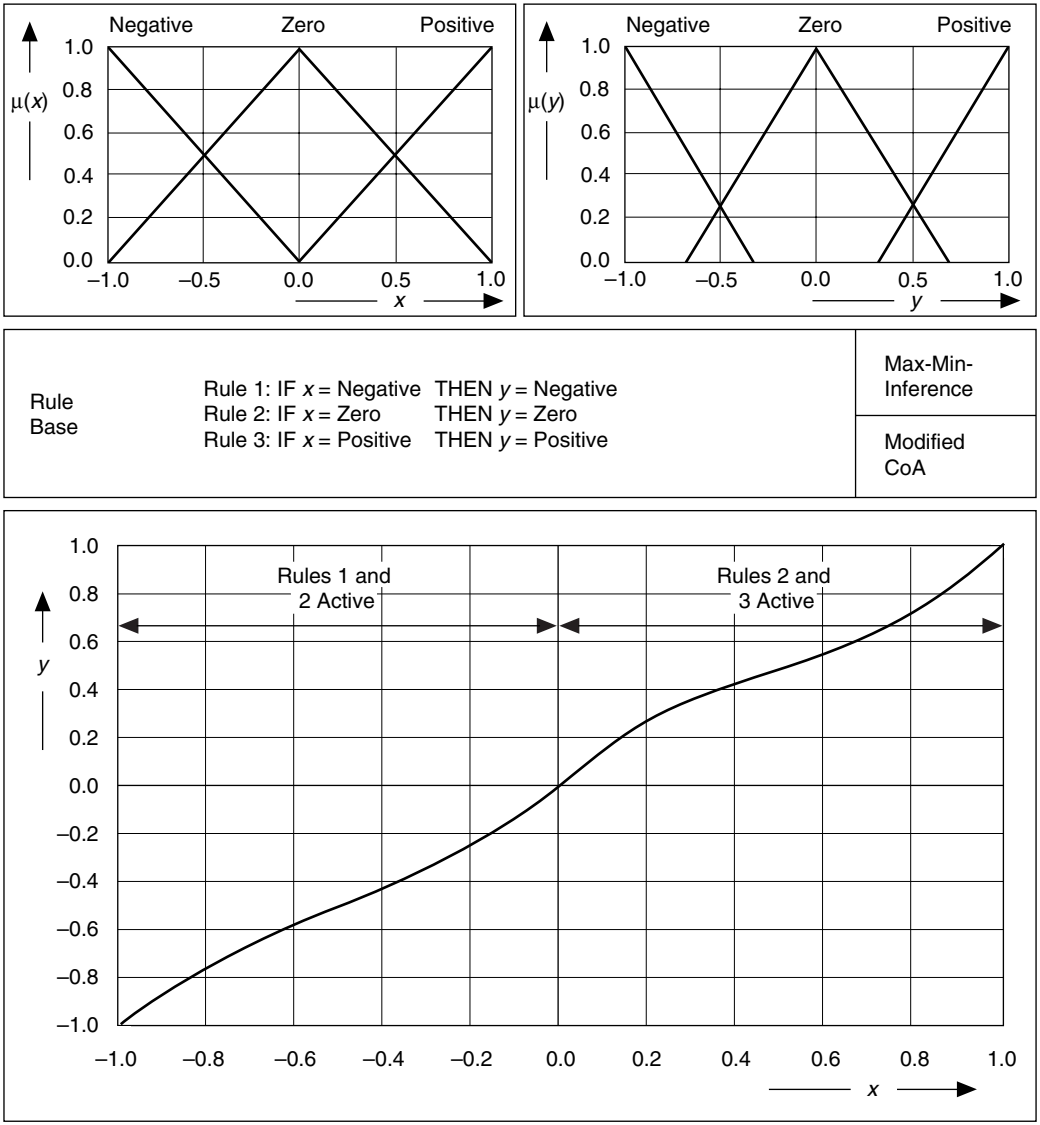


Figure 6-8. I/O Characteristic of a Fuzzy Controller (Entirely Overlapping Input Terms)

Because the antecedence terms completely overlap, there are always two active rules. The different conclusion terms, weighted by the degree of truth

for the different active rules, that lead to the nonlinear pass of the controller characteristic, determine the output value.

Figure 6-9 shows the controller characteristic that results when nonoverlapping antecedence terms describe the input variable.

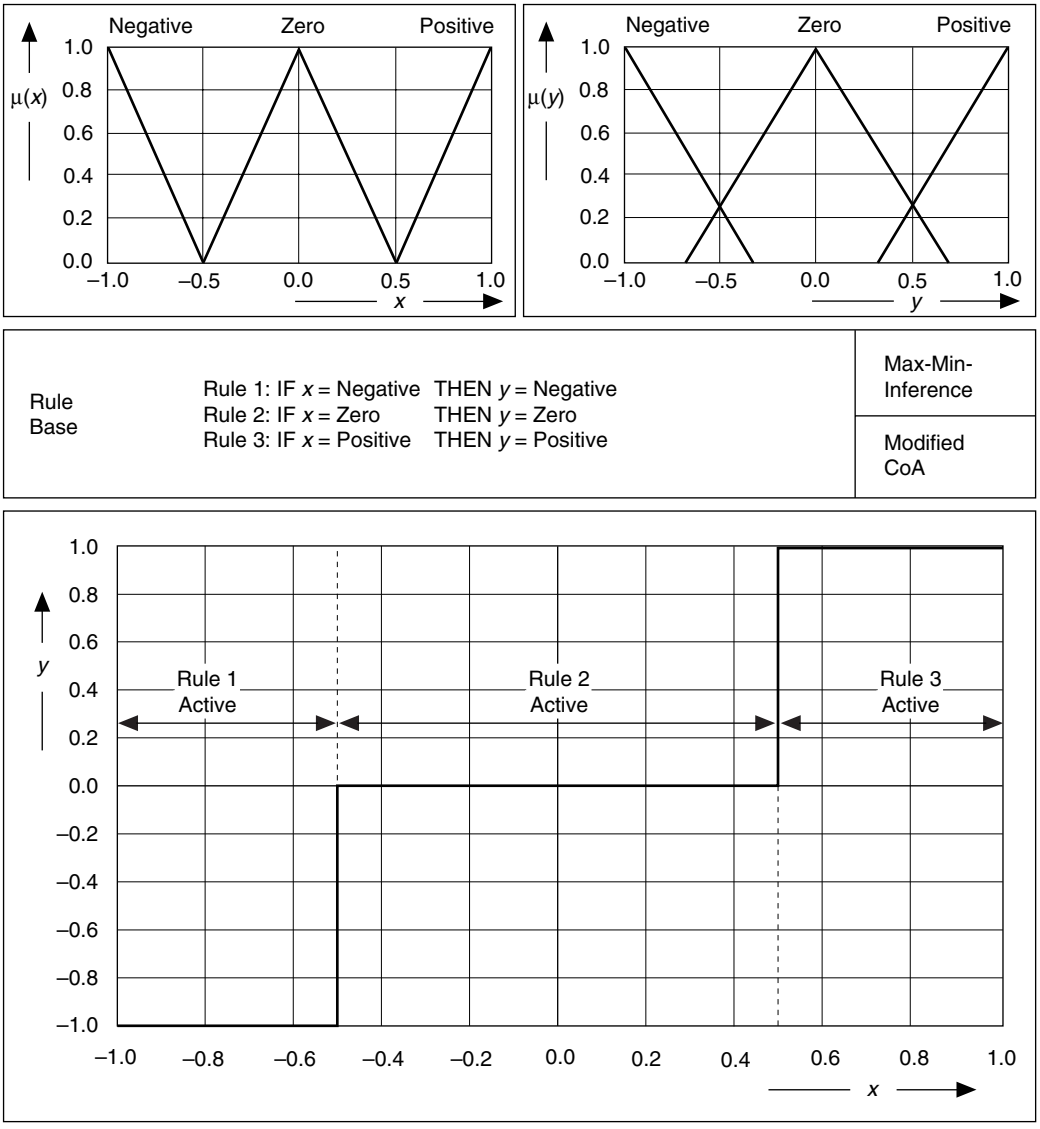


Figure 6-9. I/O Characteristic of a Fuzzy Controller (Nonoverlapping Input Terms)

In this case, only one rule is active for each input situation that leads to the stepped controller characteristic shown in Figure 6-9.

If there are undefined intervals within input and output terms, or the rule base is incomplete, you must tell the fuzzy controller what to do. If there is no rule 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 6-10 shows the resulting effect on the controller characteristic.

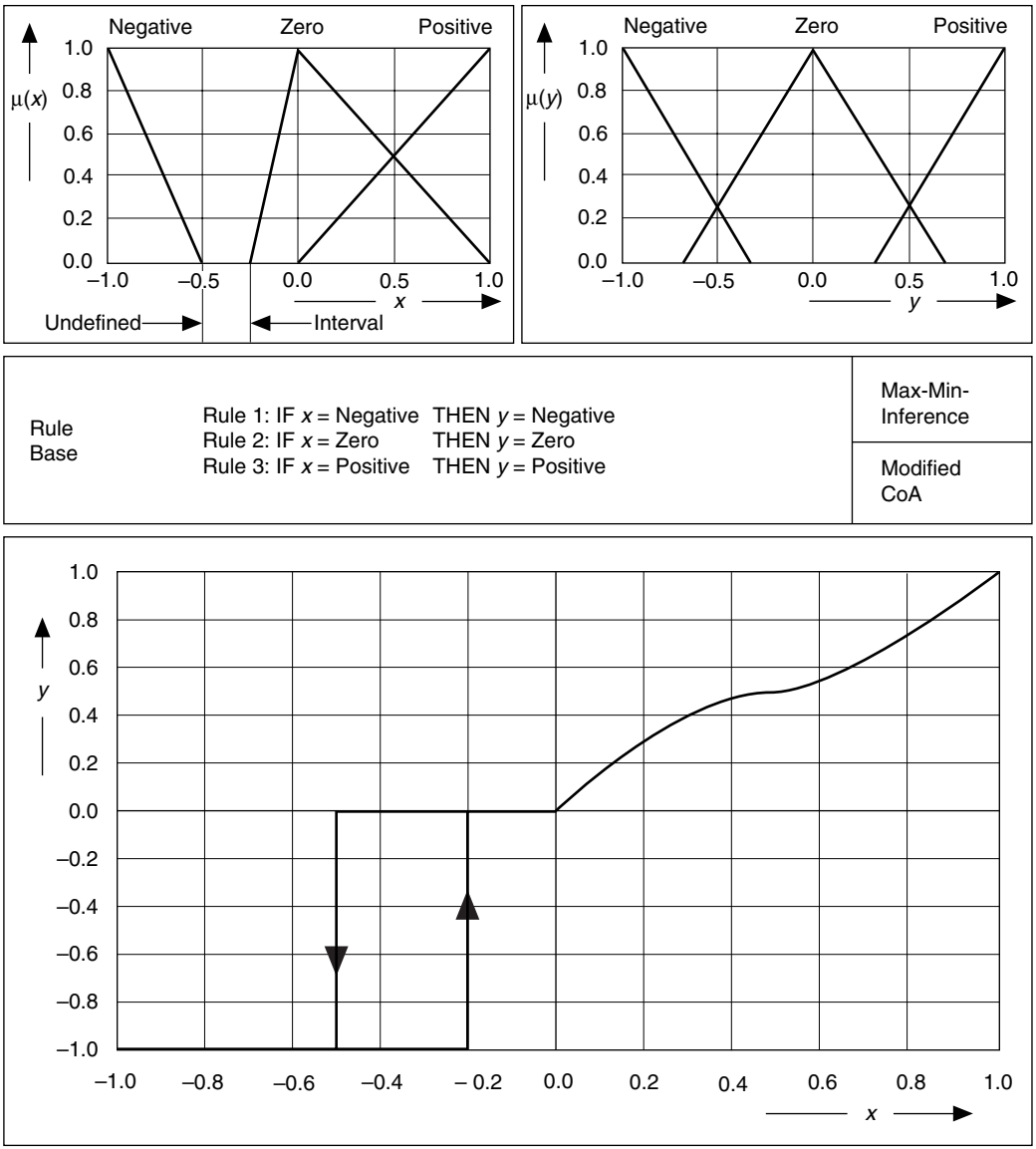


Figure 6-10. I/O Characteristic of a Fuzzy Controller (Undefined Input Term Interval)

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

The simplest way to obtain a linear controller characteristic is to use singletons as conclusion terms with entirely overlapping input terms. Refer to Figure 6-11 for an example of such a controller. Singletons are normalized rectangular membership functions with an infinitely small width.

Using singleton membership functions for the conclusion terms makes the CoG defuzzification method identical to the CoM method. Figure 6-11 shows the controller for the CoG method using singleton membership functions.

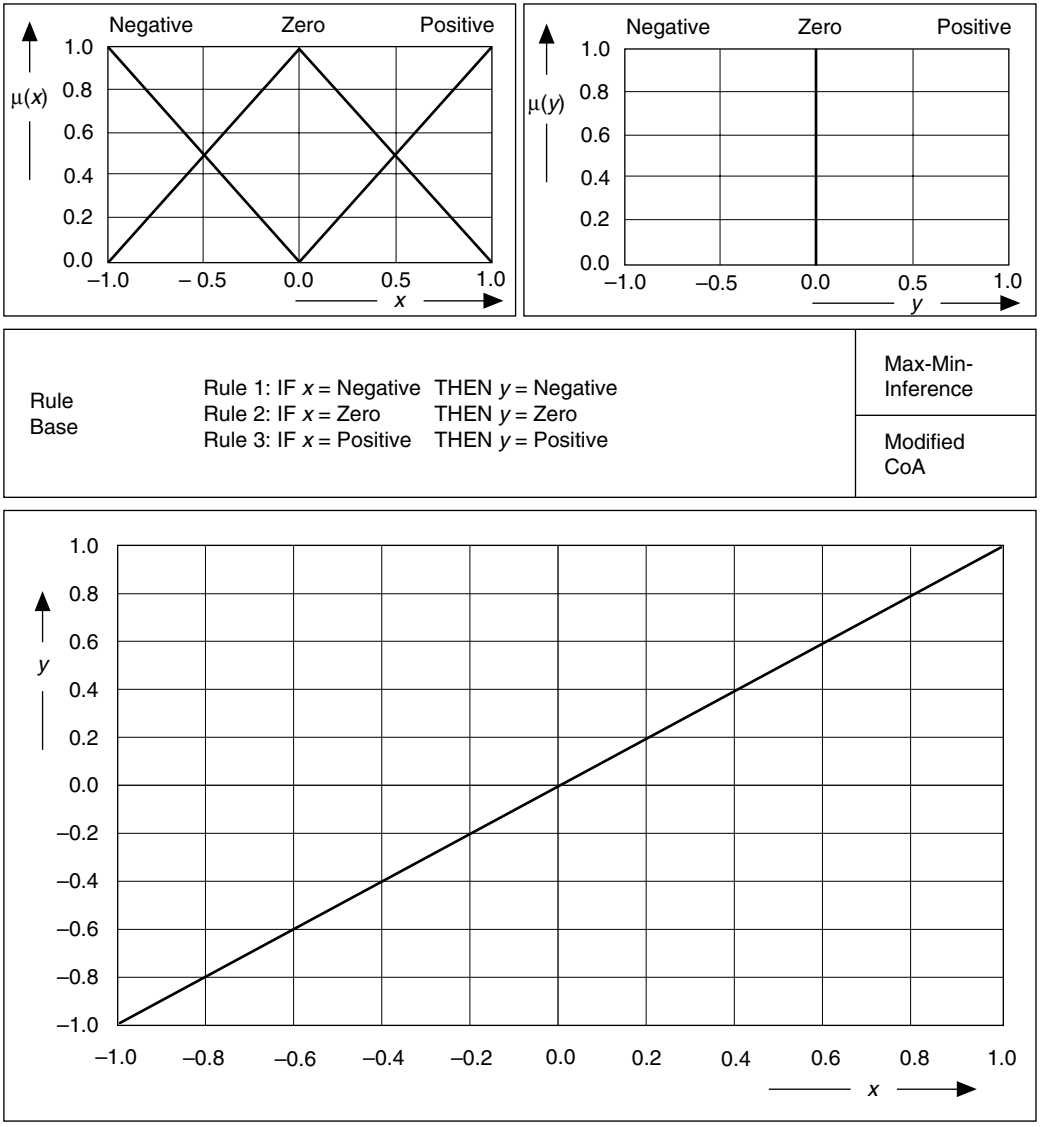


Figure 6-11. I/O Characteristic of a Fuzzy Controller (Singletons as Output Terms, Entirely Overlapping Input Terms)

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 conclusion terms, especially if all the conclusion terms are equal in width. Then only the typical values of the conclusion terms are significant.

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

Figure 6-12 shows that if all the conclusion terms are equal in width, the overlapping degree of the membership functions for the conclusion terms has no significant influence on the controller characteristic.

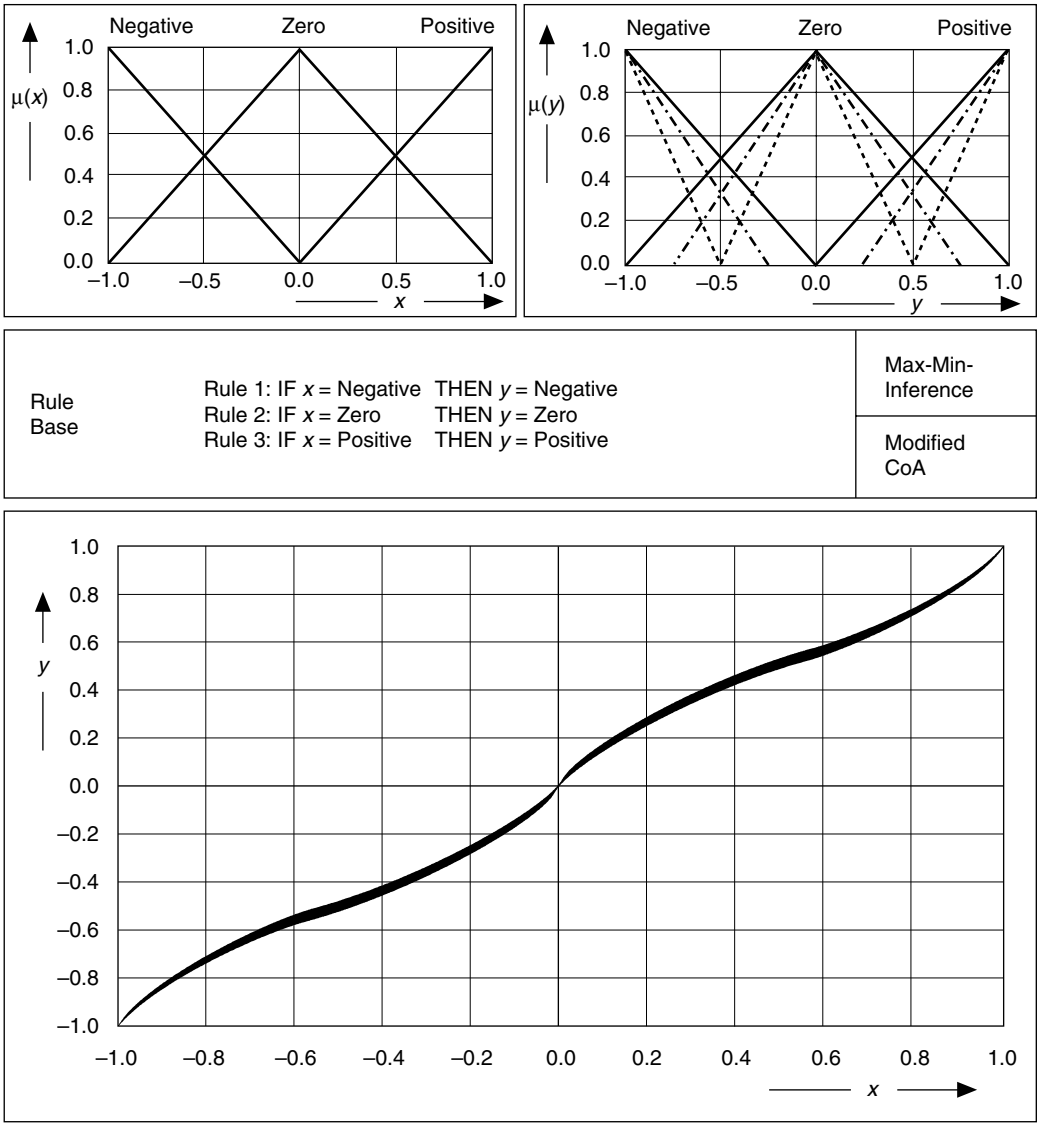


Figure 6-12. I/O Characteristics of a Fuzzy Controller (Different Overlapping Degrees of Membership Functions for the Output Terms)

Instead, use output terms that membership functions model with equally distributed typical values but different scopes of influence, to significantly influence the controller characteristic. 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. This effect is demonstrated in Figure 6-13.

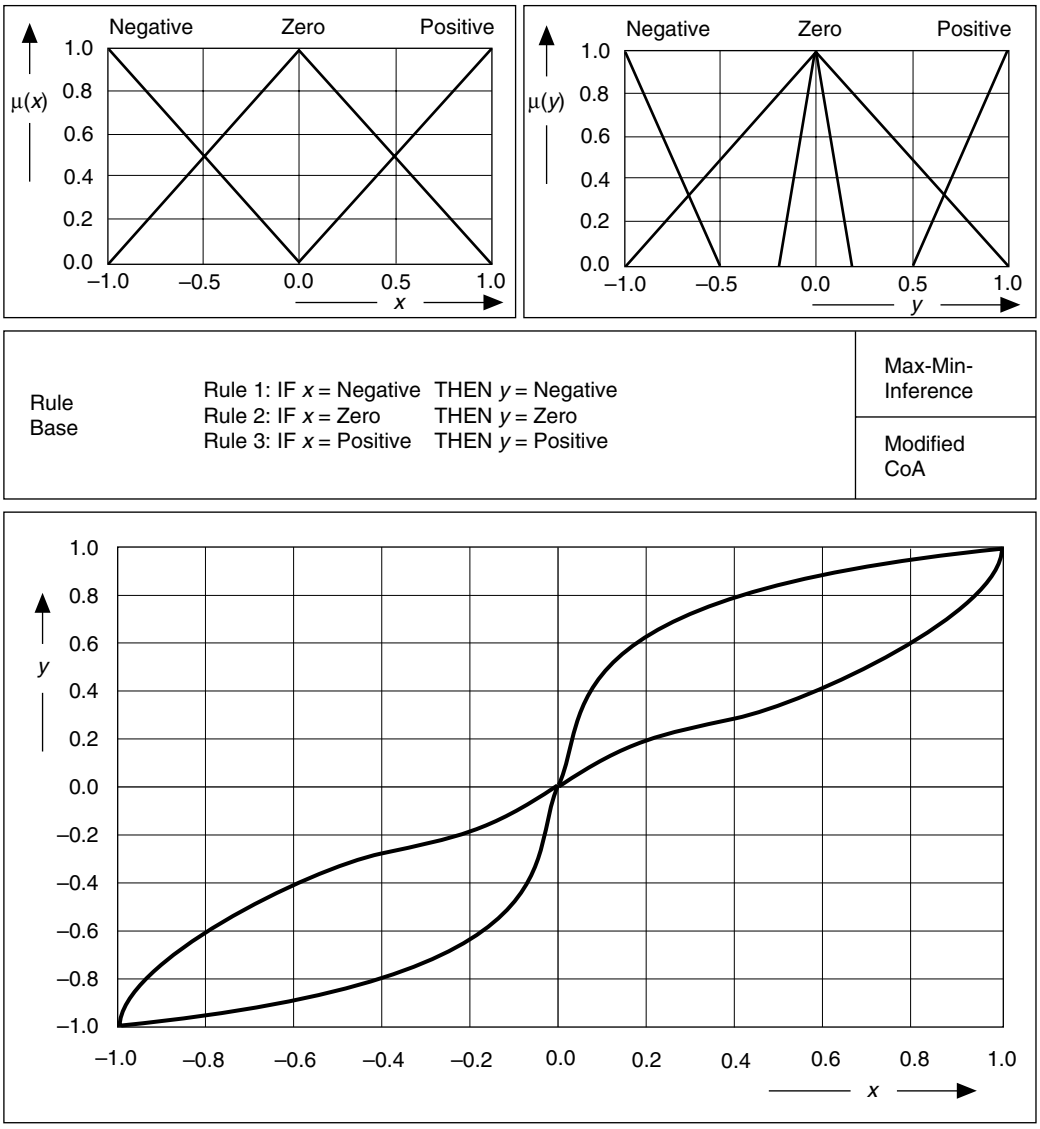


Figure 6-13. I/O Characteristics of a Fuzzy Controller (Wide and Small Membership Functions for the Output Terms)

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 is because of the averaging character of the defuzzification methods described in Chapter 5, *Overview of Fuzzy Logic*.

When you use the MoM defuzzification method, you calculate the most plausible result. In other words, the typical value of the conclusion term of the most valid rule is taken as a crisp output value, which results in stepped output characteristics as shown in Figure 6-14.

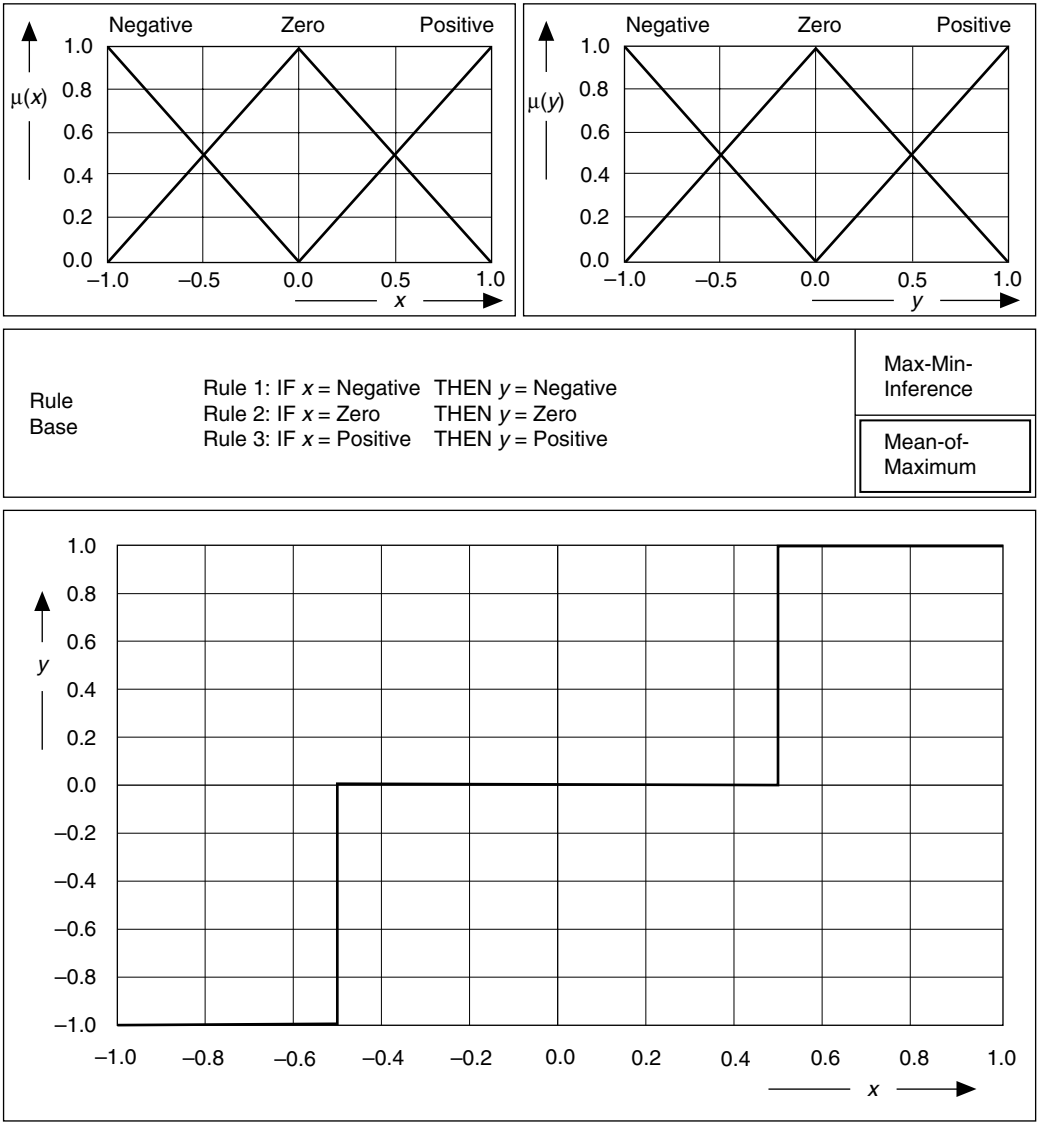


Figure 6-14. I/O Characteristic of a Fuzzy Controller with Mean-of-Maximum (Entirely Overlapping Membership Functions for Input and Output Terms)

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

Figure 6-15 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

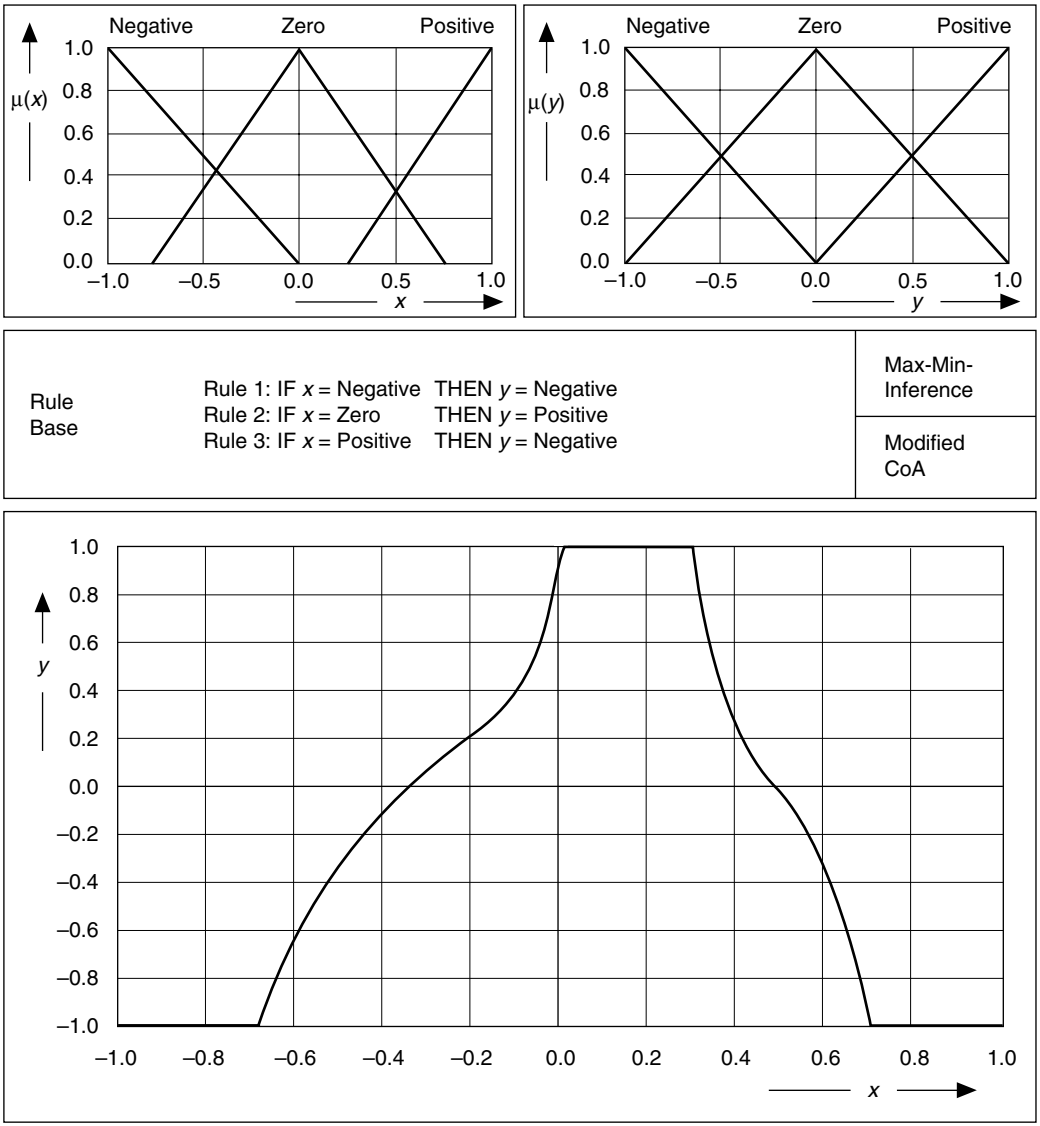


Figure 6-15. I/O Characteristic of a Fuzzy Controller with a Changed Rule Base

The examples show that you can use a fuzzy controller to perform arbitrary I/O operations. The number of linguistic input and output 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 6-16. There are four linear sections that you can describe with the five circled base points (xi, yi).

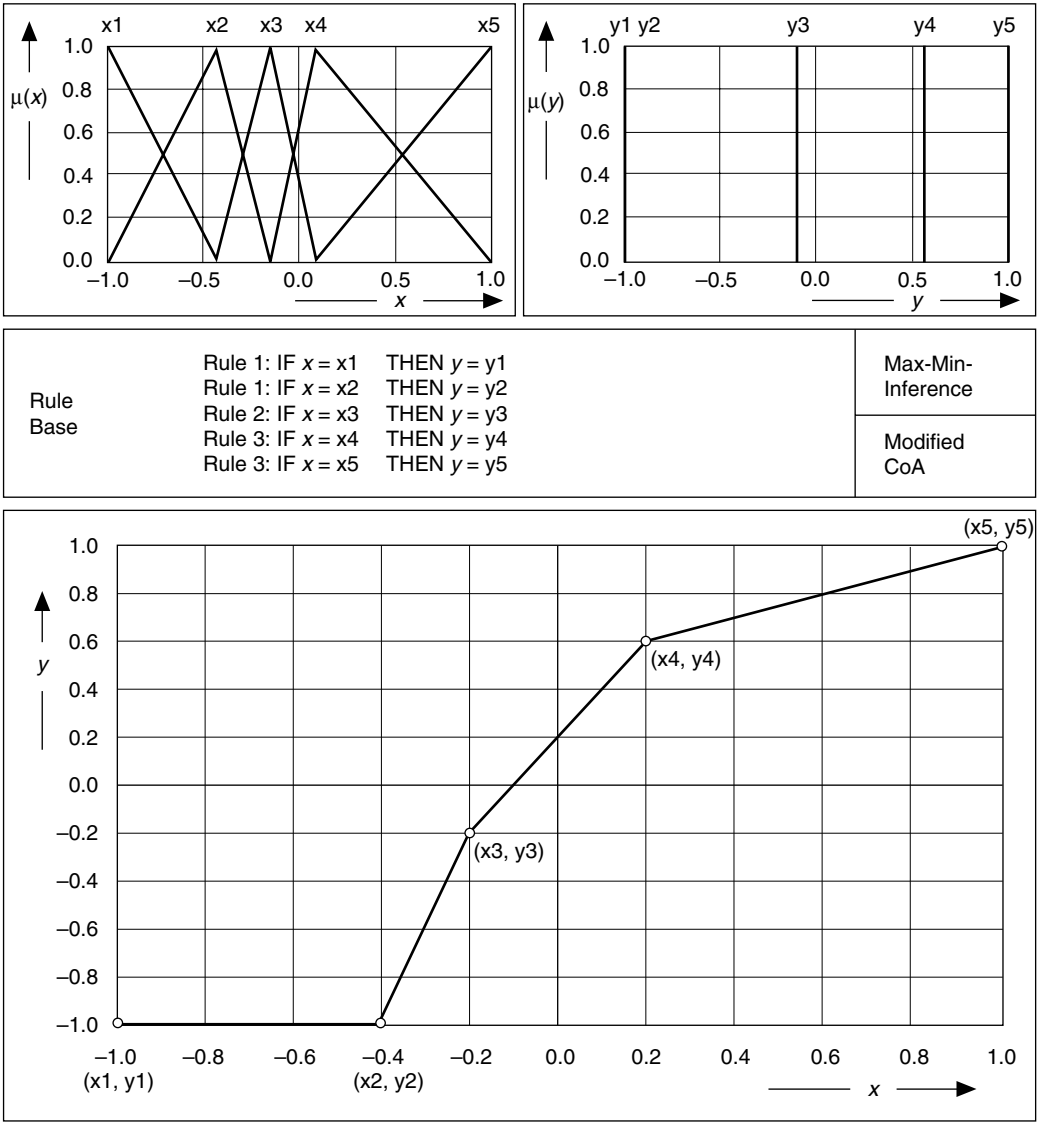


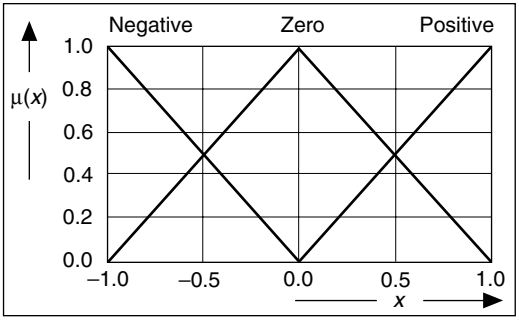
Figure 6-16. Fuzzy Controller for a Given I/O Characteristic

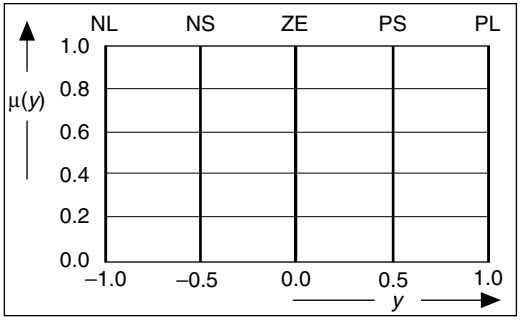
To use a single input fuzzy controller to reproduce the given characteristic, use five linguistic terms each for the input and output quantities, naming

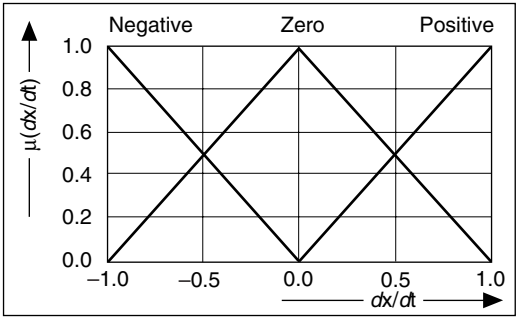
them x1, x2, …, x5 and y1, y2, …, y5, respectively. To obtain the stepped linear sections between the base points, you must always have exactly two available active rules. To implement this, entirely overlap the triangular membership functions for the input variable, giving each a typical value that 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 operation to combine the different input conditions raises an additional nonlinear effect. Usually the minimum operator models the AND-operation that always prefers as a result the antecedence term of the rule with the lowest degree of truth. Refer to Figure 6-16 for an example. Figure 6-17 shows the I/O characteristic field for a dual input fuzzy controller.







Rule Base		Input x		
		Negative	Zero	Positive
Input dx/dt	Negative	NL	NS	ZE
	Zero	NS	ZE	PS
	Positive	ZE	PS	PL

Max-Min- Inference	Modified CoA

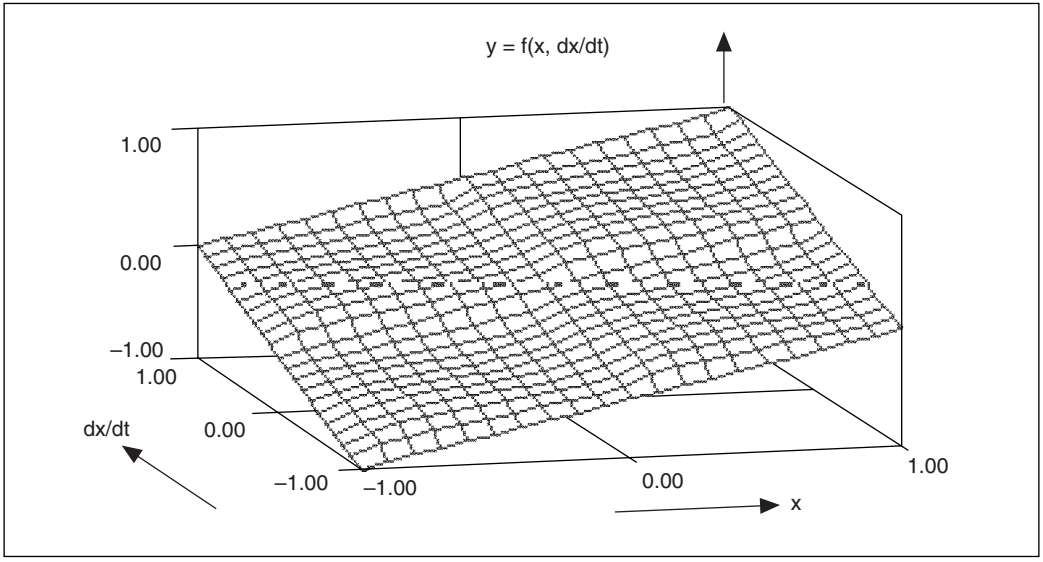
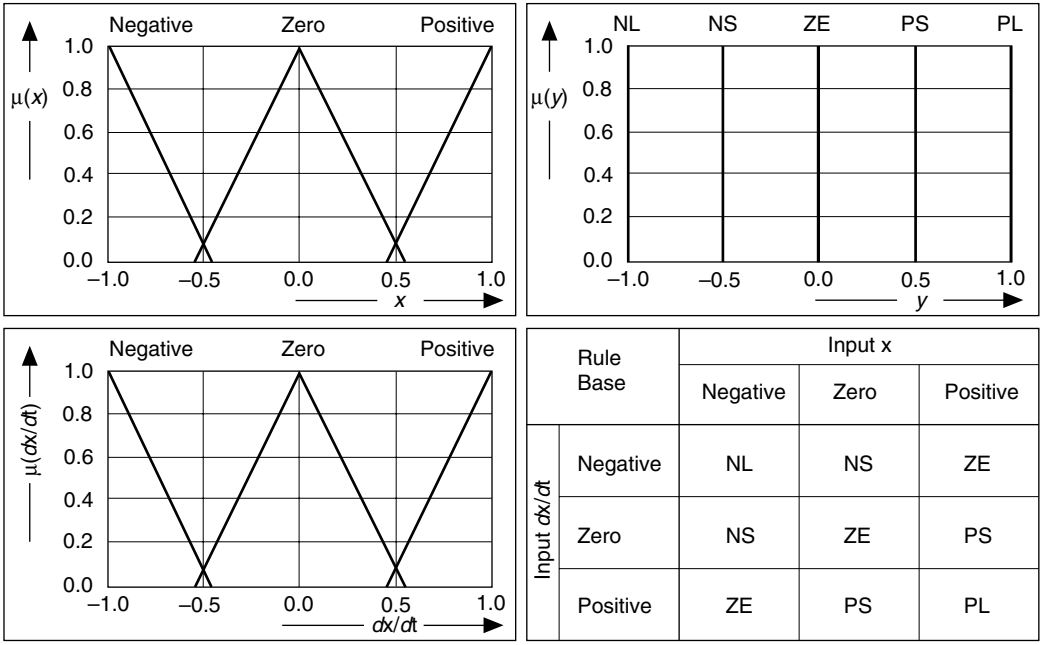


Figure 6-17. I/O Characteristic Field of a Dual Input Fuzzy Controller

Because the minimum operator used in the aggregation step is nonlinear, the characteristic field is not exactly linear despite the entirely overlapping membership functions that overlap entirely for both input variables. Nonoverlapping membership functions yield a stepped characteristic field with constant planes, as shown in Figure 6-18.



Rule Base		Input x		
		Negative	Zero	Positive
Input dx/dt	Negative	NL	NS	ZE
	Zero	NS	ZE	PS
	Positive	ZE	PS	PL

Max-Min- Inference	Modified CoA

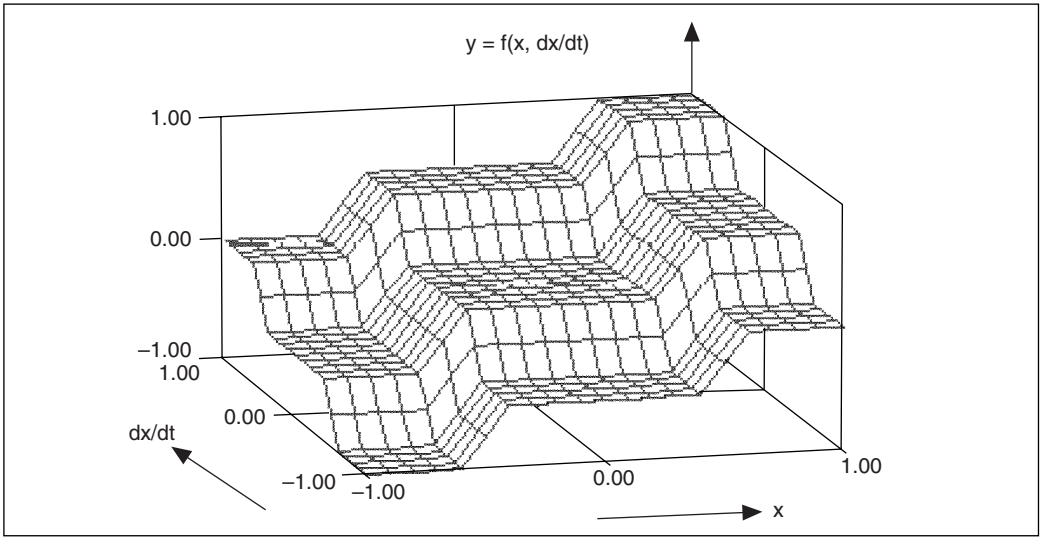


Figure 6-18. I/O Characteristic Field of a Dual Input Fuzzy Controller (Slightly Overlapping Input Terms)

7

Design Methodology

This chapter provides an overview of the design methodology of a fuzzy controller.

Design and Implementation Process Overview

Acquiring Knowledge

The knowledge base of a fuzzy controller determines its I/O characteristics and thus the dynamic behavior of the complete closed-loop control circuit. The knowledge base consists of the following parts:

- Linguistic terms, defined by membership functions, that describe the input and output quantities of the controller
- Rule base that contains engineering knowledge
- Operators for both the AND and OR operations
- Fuzzy inference method and defuzzification method

Within the first system design step, you must establish all of the linguistic variables and terms for the given application as the vocabulary of the rule-based system. Use the rule base to formulate the control strategy, then select an appropriate defuzzification method.

Optimizing Offline

Within this design step, you should test the prototype controller and simulate it with either real process data previously recorded from the process or simulation data obtained from a mathematical process model. You can perform transfer characteristics analysis and time response analysis to observe the system behavior and optimize the controller. LabVIEW supports both types of analysis. In this step, you also can use Neuro Fuzzy techniques, as well as Genetic or Evolutionary Algorithms, to optimize your system.

Optimizing Online

With the data acquisition capabilities of LabVIEW, you can run the fuzzy controller in conjunction with a process. Then, you can use online optimization techniques to make modifications to the running system.

Implementing

Although you can use the fuzzy controller directly with LabVIEW, real-time performance constraints might make it necessary to download the fuzzy controller to a fast microcontroller board.

Defining Linguistic Variables

The sensors and actuators of the system to be automated determine the input and output quantities of a fuzzy controller. Each additional quantity you measure provides more information about the current process state. However, although additional sensors can improve accuracy, they also can increase cost.

Fuzzy systems do not require high-precision measurement equipment. In fact, using inexpensive, lower-precision sensors to obtain many values is better than using expensive, higher-precision sensors to acquire less data. If measuring exact process quantities is too difficult, secondary quantities that reveal less specific process information might be sufficient.

Number of Linguistic Terms

The possible values of a linguistic variable are the linguistic terms which are linguistic interpretations of technical quantities. For example, the quantity vehicle position *x*, which is usually called the base variable and is measured in meters, can have the linguistic interpretations *left*, *left center*, *center*, *right center*, and *right*.

When you create a linguistic variable, first determine how many terms define the linguistic variable. In most applications, between three and seven terms make up a linguistic variable. It makes no sense to use less than three terms, because most linguistic concepts have at least two extreme terms with a middle term between them. On the other hand, linguistic systems that use more than seven terms are difficult to understand because humans use their short-term memory to interpret technical quantities, and the human short-term memory can only compute up to seven symbols simultaneously.

Linguistic variables usually have an odd number of terms because they are defined symmetrically and they include a middle term between the extremes.

As a starting point, set up the input variables with at least three or five terms and the output variables with five or seven terms.

Standard Membership Functions

The degree of truth to which a measurement value of a technical quantity satisfies the linguistic concept of a certain term of a linguistic variable is called degree of membership. You can use a mathematical function to model the degree of membership of a continuous variable.

You can apply the normalized standard membership functions illustrated in Figure 7-1 to most technical processes. These standard functions include **Z**-type, **Λ**-type (triangular shape), **Π**-type (trapezoidal shape), and **S**-type membership function shapes.

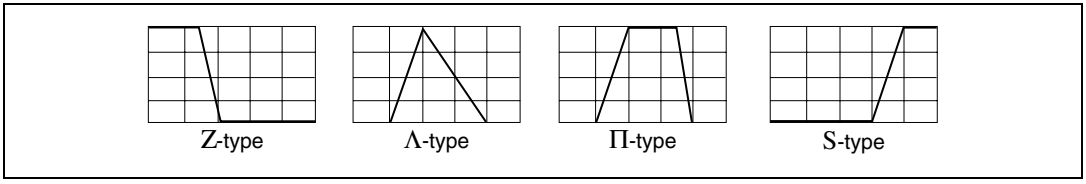


Figure 7-1. Shapes of Standard Membership Functions

To establish standard membership functions, complete the following steps, illustrated in Figure 7-2.

1. Define the typical value for each term. This is the value that best fits the linguistic meaning of the term and yields the membership degree $\mu = 1$.
2. For each term, set the membership degree to $\mu = 0$ at the typical values of neighboring terms.
3. Connect the point $\mu = 1$ with the points $\mu = 0$ by straight lines, creating triangular membership function shapes for all inner terms.
4. Because there are no terms beyond the rightmost term and below the leftmost term, all values that fall into this region belong to the respective border term with the membership degree $\mu = 1$.

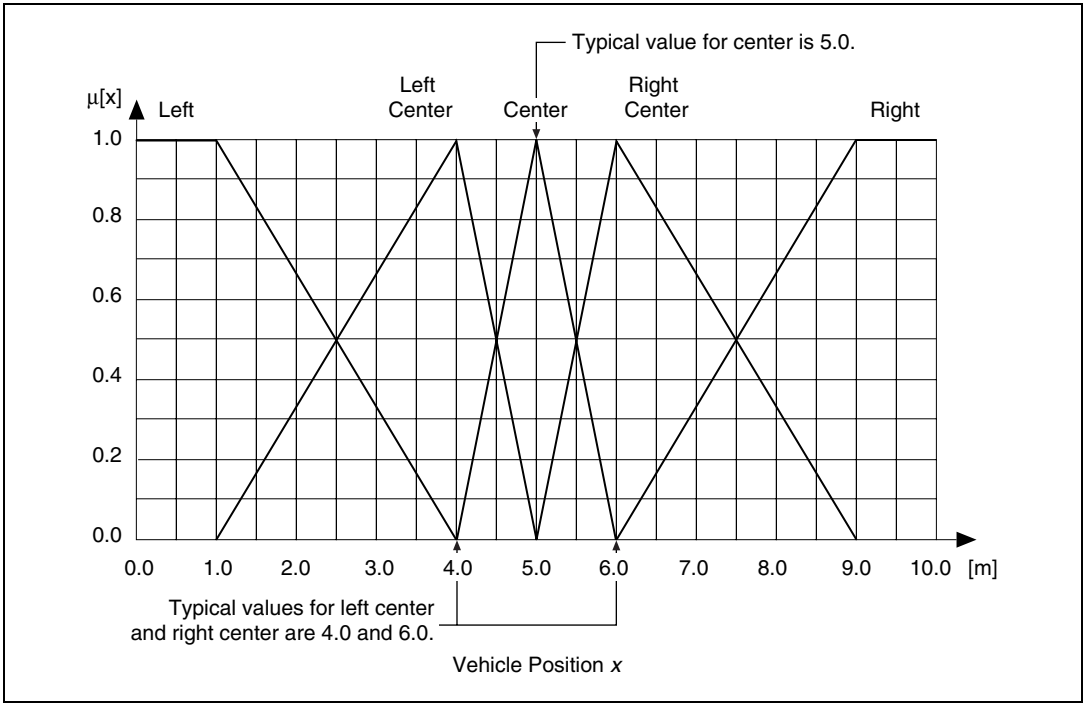


Figure 7-2. Definition of a Triangular Membership Function for the Linguistic Term *Center*

Sometimes the typical value of a term is an interval rather than a crisp value. If, for example, the position *center* is characterized by the statement $x = 5 \pm 0.25$ m, a trapezoidal membership function applies, as shown in Figure 7-3.

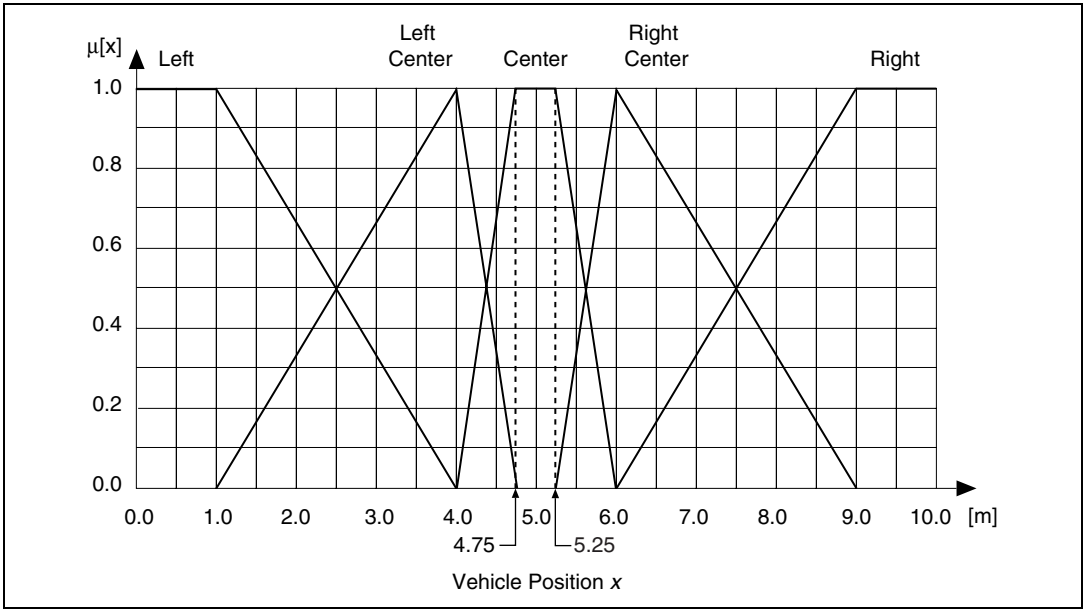


Figure 7-3. Definition of a Trapezoidal Membership Function for the Linguistic Term *Center*

If there is no *a priori* information available, begin with terms equally spaced within the range of the associated variable, with each term entirely overlapping the neighboring terms. Cover the desired stable region of the system with a larger number of linguistic terms that have a small influence interval rather than trying to cover the border regions with a smaller number of linguistic terms that have a large influence interval. A term distribution like this makes the controller more sensitive within the stable state region of the system.

You must take into account disturbance effects, such as noise, on input values such as noise. Do not set up membership functions with an interval of influence that is smaller than the amplitude of the noise signal.

Defining a Fuzzy Logic Rule Base

The fuzzy logic rule base is the main part of a fuzzy system and contains all the engineering knowledge necessary to control a system. The rule base supplies all the actions the fuzzy controller should perform in certain situations. In a sense, the rule base represents the intelligence of the controller.

Changes to a single rule only have a local influence on the controller characteristic. Thus you can selectively change the behavior of the fuzzy controller for a certain input situation by modifying a particular rule. Because the modification of a rule is usually carried out in discrete steps through changes to the consequence term, modifications to a rule have a much greater influence on the controller characteristic than modifications to the membership functions. Implement weight factors, which are Degrees of Support, for the rules to enhance or reduce the influence of a rule on the controller characteristic.

To build up a rule base, define one rule for each combination of antecedent terms, of the input variables used in the IF-part of the rule. Then select the most plausible conclusion from the output variable to specify the THEN-part of each rule.

Assume that you are building a fuzzy controller with m input variables, each of which has p terms each. The total number N of possible rules is

$$N = p^m$$

p = number of terms for each input variable
m = number of input variables

For example, for three input variables with five terms each, the total number of possible rules is 125. The complete rule base for five input variables with seven terms each totals 16,807 rules.

Notice that for systems with numerous controller inputs, you can use cascading fuzzy controllers to avoid large rule bases. Outputs from fuzzy controllers serve as the inputs to the next layer of fuzzy controllers.

In the case of a fuzzy controller with m input variables, each with an individual number of terms p_i (with $1 \le i \le m$), there are a total of N possible rules according to

$$N = \prod_{i=1}^{m} p_i$$

p_i = number of terms for input variable i
m = number of input variables

This great degree of freedom allows a lot of design flexibility. However, it is very difficult to implement the complete rule base in large and complex systems. In such cases, you usually only implement the rules that cover the normal system operation.

Note A fuzzy controller with an incomplete rule base must have a default action value, which is usually the last command value, for input situations with no active rule.

A rule base with at least one active rule for each possible combination of crisp input values is called a complete rule base. Because there are overlapping regions of the membership functions, an undefined output in a rule base does not necessarily mean that there is no rule active for a certain input situation.

The completeness of a rule base is not the only aspect to consider when you deal with large rule bases. Avoid contradicting rules, rules with the same IF-part but different THEN-parts because they are illogical. Contradicting rules have only a marginal effect on the controller characteristic because of the averaging process that occurs during the defuzzification step. A consistent rule base is a rule base that has no contradicting rules.

If the rule base is small enough to contain all possible rules, it is not difficult to detect inconsistencies. This is guaranteed for rule bases that can be built in the form of a matrix. Refer to Figure 5-9, *Complete Linguistic Rule Base* for more information about rule bases in matrix form. However, many rule bases are larger and more complex. To build these rule bases, begin with just a few rules to operate input quantities and gradually add more rules. It is difficult to detect inconsistencies in larger rule bases.

For fuzzy controllers with only two or three input quantities, it is possible to estimate the qualitative controller characteristic just by looking at the rule base. Neighboring terms within a rule matrix with strongly differing meanings like *positive large* and *negative small* indicate steeply sloped edges in the control surface, which usually are not desired. This is referred to as the continuity of a rule base. If neighboring rules have the same or similar conclusions, the rule base is said to be continuous.

Within large rule bases it is possible to have multiple definitions of the same rule. This is called redundancy. It has no influence on the inference result at all if the Max-Min inference method is implemented. But there are other inference methods, which are not discussed in this manual, such as the Sum-Product method, in which multiple rules can effect the inference result.

Operators, Inference Mechanism, and the Defuzzification Method

In closed-loop control applications that use fuzzy logic, the standard common operators for the AND- and the OR-operation are the Min- and Max-operators discussed in the *Using IF-THEN Rules in Fuzzy Inference* section of Chapter 5, *Overview of Fuzzy Logic*. Within certain control applications in the field of process technology, however, it might be necessary to use a compensatory AND-operator rather than the pure AND. The most important compensatory AND-operator is the γ-operator, which is not discussed in detail here. The γ-operator allows a continuous tuning between AND, no compensation, and OR, full compensation. In real situations the word AND is sometimes used to combine two antecedences meaning as well as, indicating that you can compensate when you have a little less of one quantity. This is exactly what the γ-operator, also called the compensatory AND, can model. Refer to Appendix A, *References*, for a list of documents with more information about this topic.

The standard inference mechanism is the Max-Min method. Other inference methods have only a marginal influence on the controller characteristic.

The defuzzification method derives a crisp output value that best represents the linguistic result obtained from the fuzzy inference process. As explained in Chapter 5, *Overview of Fuzzy Logic*, there are generally two different linguistic meanings of the defuzzification process: calculating the best compromise, CoM or CoA, and calculating the most plausible result, MoM.

An important aspect of the defuzzification method is the continuity of the output signal. Consider a fuzzy logic system with a complete rule base and overlapping membership functions. A defuzzification method is continuous if an arbitrary small change of an input value can never cause 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 can never jump to a different value with a small change to the inputs. To the contrary, the defuzzification method MoM is discontinuous because there is always a point at which an arbitrary small change in the input situation of the system will cause a switch to another more plausible result. Refer to Table 7-1 for a comparison of different fuzzification methods.

Table 7-1. Comparison of Different Defuzzification Methods

Assessment Criteria	Method		
	Center-of-Gravity (CoG) **Center-of-Area (CoA)**	**Center-of-Maximum (CoM)**	**Mean-of-Maximum (MoM)**
Linguistic Characteristic	Best Compromise	Best Compromise	Most Plausible Result
Fit with Intuition	Implausible with varying membership function shapes and strong overlapping membership functions	Good	Good
Continuity	Yes	Yes	No
Computational Effort	Very High	Low	Very Low
Application Field	Closed-loop Control, Decision Support, Data Analysis	Closed-loop Control, Decision Support, Data Analysis	Pattern Recognition, Decision Support, Data Analysis

8

Using the Fuzzy Logic Controller Design VI

This chapter describes how to use the Fuzzy Logic Controller Design VI to design a fuzzy controller.

Overview

The Fuzzy Logic Controller Design VI consists of the following parts:

- Project Manager—Maintains a fuzzy logic project
- Fuzzy Set Editor—Defines and modifies linguistic variables including their linguistic terms
- Rule Base Editor—Defines and modifies the rule base of a fuzzy system to be designed
- Testing and project maintenance utilities

Refer to the *PID Control Toolset Help*, available by selecting **Help»PID Control Toolset Help,** for more information about fuzzy logic control.

The following restrictions are valid:

- The maximum number of linguistic variables is four.
- The maximum number of linguistic terms for each linguistic variable is nine.
- The types of membership functions are normalized triangular and trapezoidal membership functions (Z-, Λ-, Π- and S-Type) and singletons.

Project Manager

Select **Tools»Fuzzy Logic Controller Design**. The Fuzzy Logic Controller Design VI runs immediately when you open it. This VI is a stand-alone application with a graphical user interface for designing and editing a fuzzy controller. Although this VI has no inputs or outputs, you can use it as a subVI. Place the icon on your application diagram to allow your user to programmatically edit the fuzzy controller.

The Project Manager front panel has a Project Description Field, indicated by the keyword description, where you can enter important project information. This description contains development ideas and other *a priori* development information for the fuzzy controller.

In addition to this, there is a Project Identification Field, an input box marked with the keyword developer, where the developer can enter his name. The Fuzzy Logic Controls process the other entries, controller, date, and time. When you close or save the project for the first time, LabVIEW prompts you to enter a project name, which then appears in the controller indication line of the Project Identification Field when you next open the project.

LabVIEW automatically calls the Fuzzy-Set-Editor when you create a new fuzzy logic project.

Figure 8-1 displays the Fuzzy Logic Controller Design VI front panel.

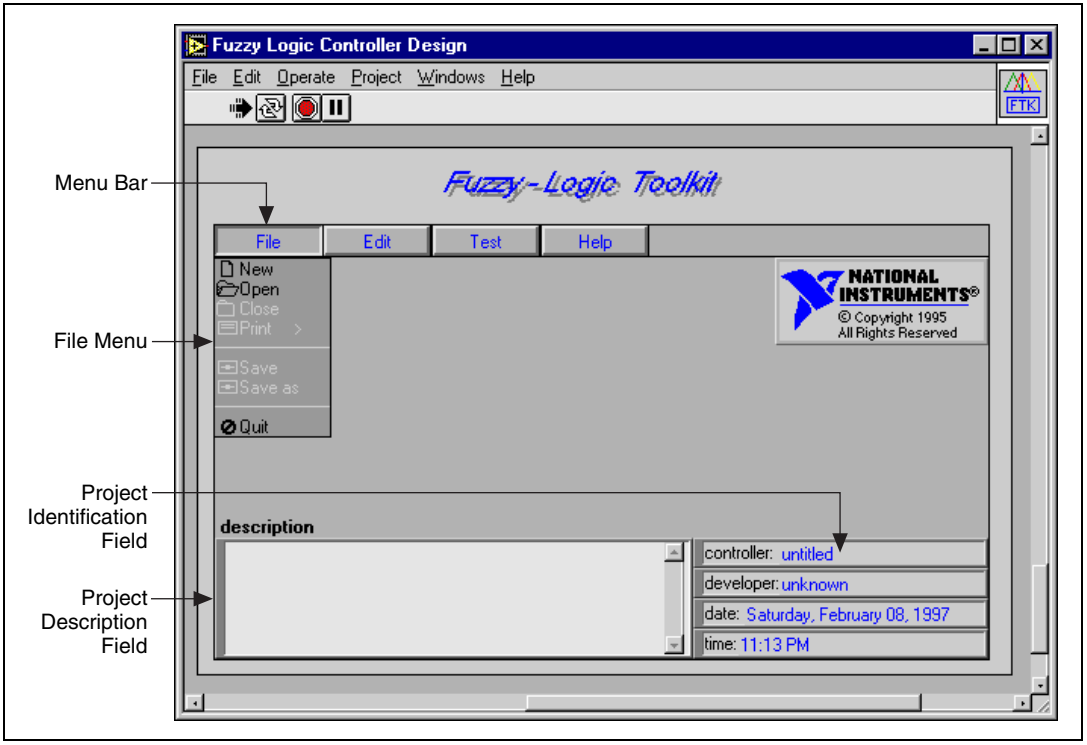


Figure 8-1. Project Manager Front Panel

Many of the commands in the Fuzzy Logic Control portion of the PID Control Toolset work similarly to those in LabVIEW. Select **File»Save** or **File»Save as** to store the project data to a file with an `.fc` extension. Refer to the *PID Control Toolset Help*, available by selecting **Help»PID Control Toolset Help,** for more information about the Fuzzy Logic controls.

Fuzzy-Set-Editor

Now, consider designing a fuzzy controller for the truck maneuvering example described in the *Rule-Based Systems* section of Chapter 5, *Overview of Fuzzy Logic*. When you begin a new project, it is best to enter at least a short project description and the name of the developer into the Project Identification Field.

Select **File»New** to start the Fuzzy-Set-Editor. If there is an existing project already loaded, select **Edit»Set-Editor** to open the Fuzzy-Set-Editor. The Fuzzy-Set-Editor front panel is shown in Figure 8-2.

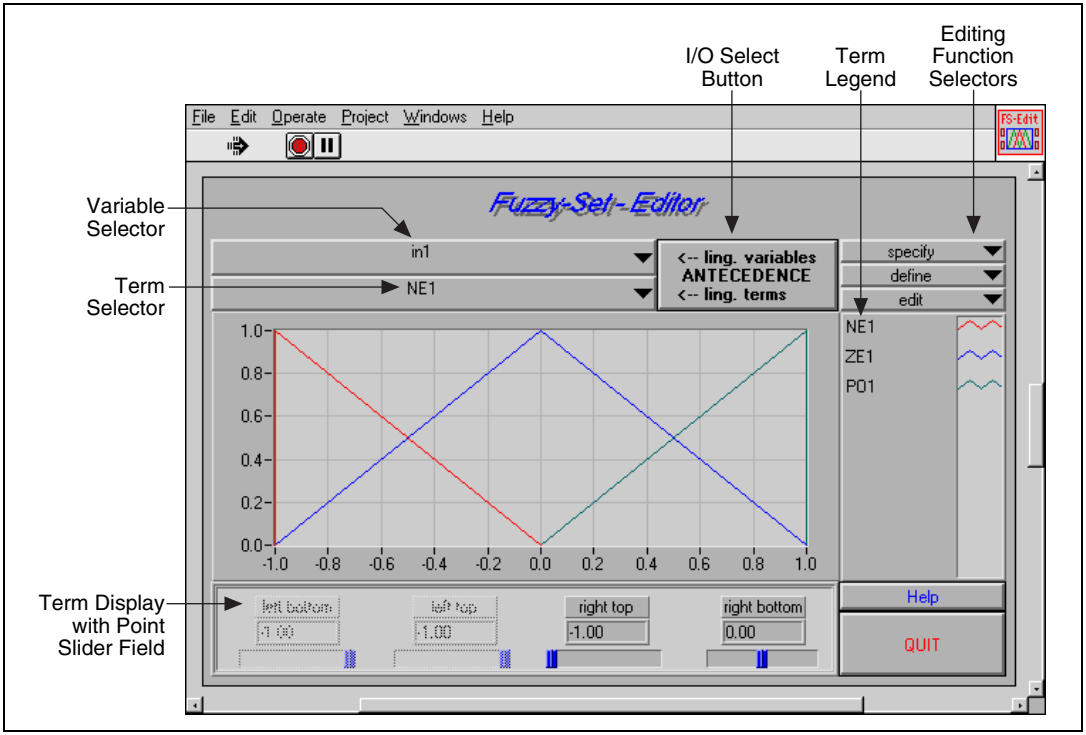


Figure 8-2. Default Fuzzy Controller Settings

A new project has certain default settings. Among these are two normalized linguistic input variables with the default description identifiers in1 and in2. Each input variable ranges from –1.0 to 1.0 by default. Each linguistic input variable is composed of three entirely overlapping linguistic terms. For in1, the linguistic terms NE1 (negative), ZE1 (zero), and PO1 (positive) are predefined. For in2, the linguistic terms NE2 (negative), ZE2 (zero), and PO2 (positive) are predefined. There is one normalized linguistic output variable comprising the three entirely overlapping linguistic terms NEo (negative), ZEo (zero), and POo (positive). The default range of the output variable is –1.0 to 1.0.

Term Display shows the linguistic terms of the linguistic variable that the **Variable Selector** activates, while the **Term Legend** displays the term description identifiers.

You can adjust the sliders or input controls in the Point Slider Field to interactively modify the linguistic term activated by the Term Selector.

The Fuzzy-Set-Editor controls modifications to terms with respect to plausibility restrictions. To prevent the user from making implausible term arrangements, LabVIEW dims all input sliders of term points that cannot be modified because of plausibility restrictions.

When you move a particular point slider to modify a term shape, the Fuzzy-Set-Editor controls and updates all input sliders according to plausibility restrictions, too. Thus, the right top value of the term NE1 might not override the left top value of the term ZE1. When you move the right top slider, the Fuzzy-Set-Editor constantly updates this slider according to the plausibility restriction mentioned above so that this point, right top of NE1, cannot exceed the left top of ZE1. As the example in Figure 8-3 illustrates, you cannot move the left-bottom point or left-top point of the term NE1 below the left-hand range limit of the input variable.

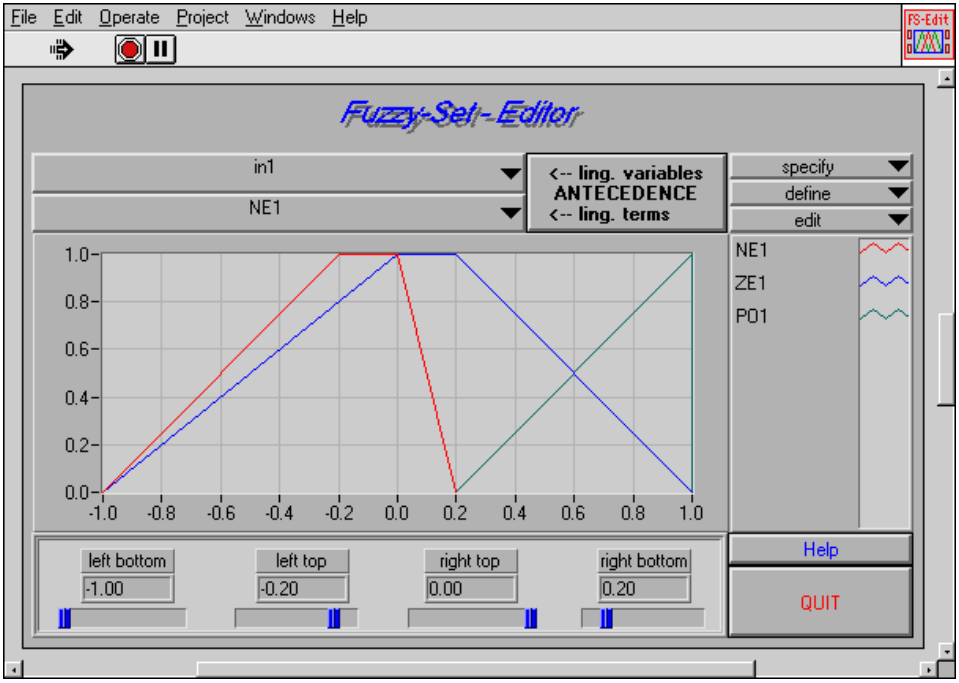


Figure 8-3. Plausibility Checking and Point Slider Movement

In the truck maneuvering example in the *Rule-Based Systems* section of Chapter 5, *Overview of Fuzzy Logic*, there are two linguistic input

variables, *vehicle position x* and *vehicle orientation* β, and one linguistic output variable, *steering angle* φ. It is a good idea to use descriptive variable names instead of the default identifiers offered by the Fuzzy-Set-Editor.

Select **Specify»Rename Variable** to display the **Rename Variable** dialog box. Now you can enter the new description identifier `vehicle-position` into the text input box above the **OK** button to change the selected variable identifier **in1**. Figure 8-4 shows the dialog box. Click the **OK** button or press <Enter> to save the new variable identifier.

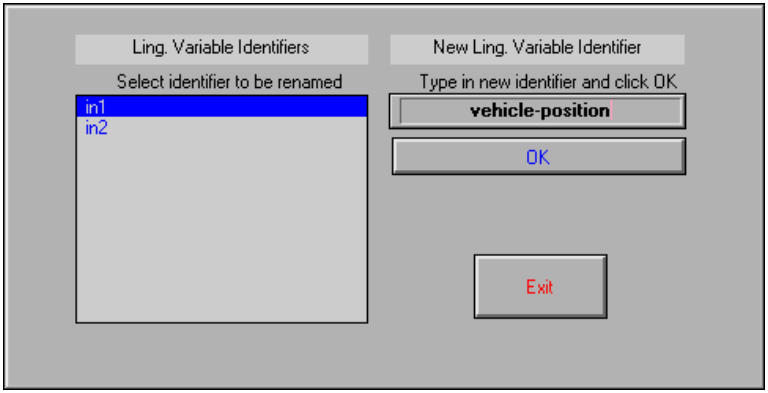


Figure 8-4. Rename Variable Dialog Box

After this, select the variable identifier **in2** and enter the description identifier `vehicle-orientation` into the text input box. Again, click **OK** or press <Enter> to save the new variable identifier. Click **Exit** to close the **Rename Variable** dialog box.

Select **ANTECEDENCE/CONSEQUENCE** on the **I/O Select** button to access the output variable. Follow the steps listed above to rename the variable. Return the button to the **ANTECEDENCE** position to be able to use the Variable Selector to access the input variables.

The Fuzzy-Set-Editor starts a new project with two input variables, each of which has the default data range interval [–1.0, +1.0]. The variable data ranges must be changed for the truck application example. The vehicle-position ranges from 0.0 to 10.0 meters and the vehicle-orientation from –90.0 to +270.0 degrees.

Select **specify»edit range** to display the **Edit Range** dialog box, from which change the data range of the input variable vehicle position.

Open the **Edit Range** dialog box to enter the range boundaries as shown in Figure 8-5.

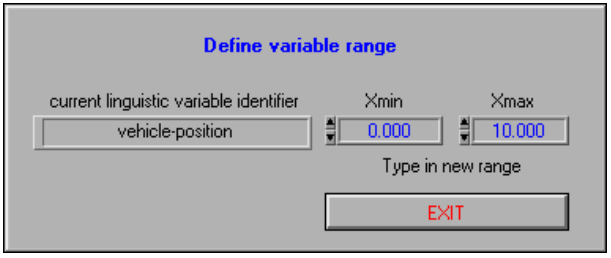


Figure 8-5. Edit Range Dialog Box

Close the dialog box. Notice that all linguistic terms of the linguistic variable are adapted to the new data range proportionally, as shown in Figure 8-6.

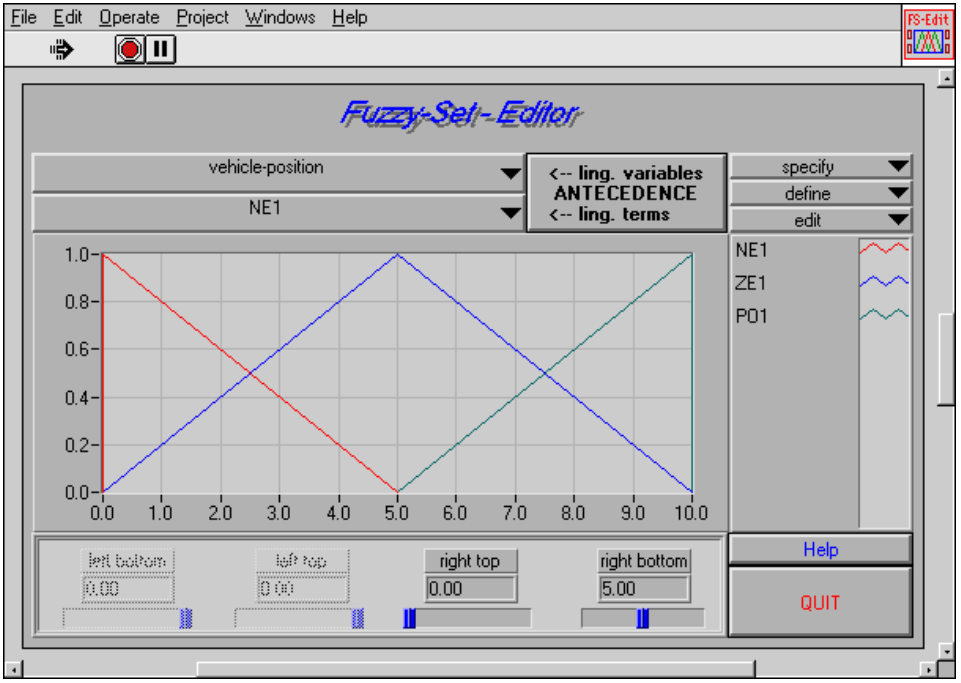


Figure 8-6. Current Input Variable Data Range Changed

For the application example, repeat the steps discussed above to set up the correct data range for the second input variable *vehicle-orientation* and for the output variable *steering-angle*, which ranges from –30.0 to +30.0 degree.

For the next step, you must have access to the input variable *vehicle-position*. To access the variable, switch **I/O Select** to the **ANTECEDENCE** position and select the desired input variable from the Variable Selector.

Any modifications made during the Fuzzy-Set-Editor session can have a significant influence on the rule base. It is always a good idea to open the Rulebase-Editor immediately after you close the Fuzzy-Set-Editor. Because you started your Fuzzy-Set-Editor session with a new project, the Fuzzy Logic Control VIs automatically call the Rulebase-Editor to create a rule base.

Because you still have to do additional work on the knowledge base, you should add and set up all linguistic terms according to the application example. You do not need to work with the Rulebase-Editor at this point in the project, so click **Quit** to exit the Rulebase-Editor.

LabVIEW does not automatically call the Rulebase-Editor when you are working on an existing project and you close the Fuzzy-Set-Editor. Regardless, closing the Fuzzy-Set-Editor as well as closing the Rulebase-Editor activates the Project Manager.

Use the **File»Save** or **File»Save As** command to save your project. When LabVIEW prompts you to enter a file name, type in `FuzzyTruck` as the project name. Notice that fuzzy controller project files always have the extension `.fc`.

Use **File»Open** to load an existing project that has not yet been loaded as shown in Figure 8-7.

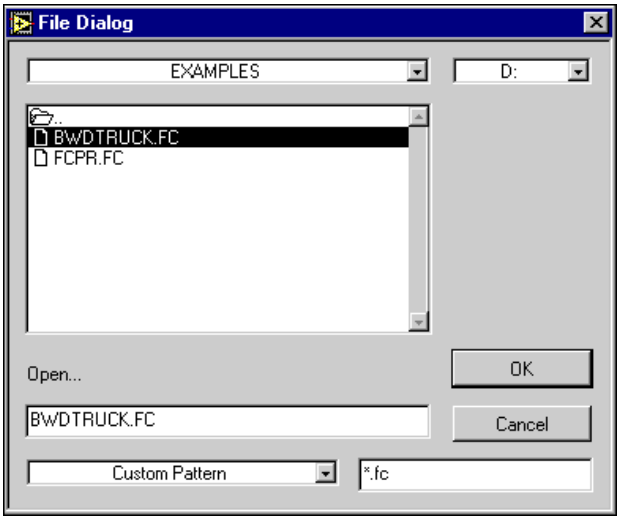


Figure 8-7. Open Command and File Dialog Box

Immediately after the Project Manager loads a project, select **Edit»Set-Editor** to call the Fuzzy-Set-Editor. Now the input and output variables have the correct names and data ranges.

The three entirely overlapping default terms NE1, ZE1, and PO1 still set up the input variable, *vehicle-position*. Because *vehicle-position* must be composed of the five linguistic terms shown in Figure 5-6, Linguistic Variable Vehicle Position x and Its Linguistic Terms, you must add two new linguistic terms. Refer to the *Rule-Based Systems* section in Chapter 5, *Overview of Fuzzy Logic*, for more information about linguistic variables and linguistic terms. All linguistic terms must have the same names and shapes so that the complete term arrangement corresponds to that in Figure 5-6.

Select **define»add term after** to add a new linguistic term between the terms NE1 and ZE1, as shown in Figure 8-8.

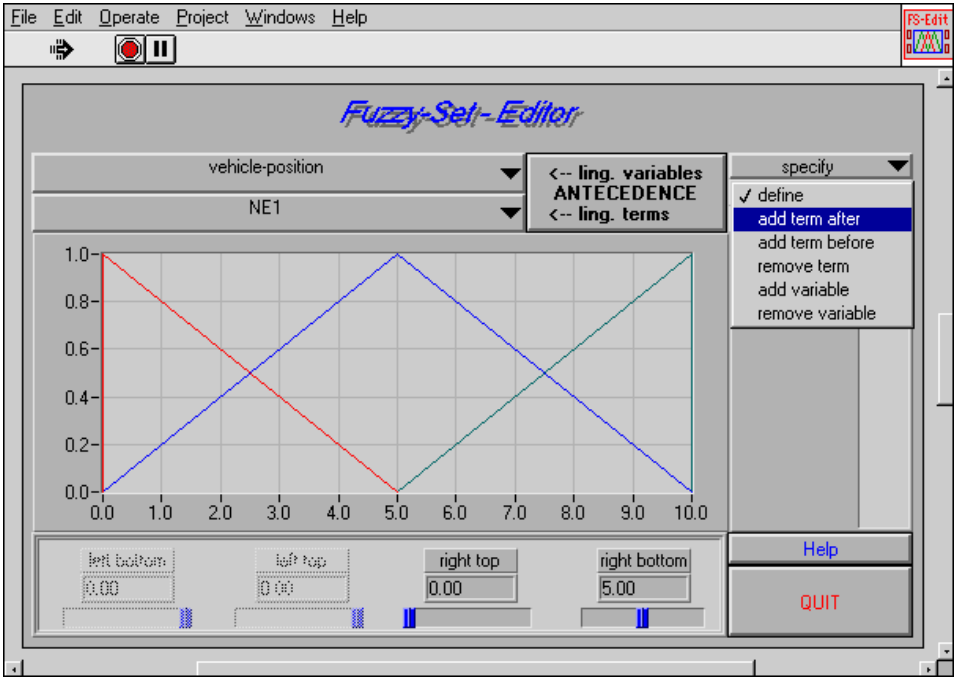


Figure 8-8. Selecting the Add Term After Command

The new linguistic term is located below the active term, as shown in Figure 8-9. The term identifier of the referred term with a + symbol added to its right side composes the new term identifier.

NE1 is the term identifier of the active term, and the new term is NE1+. Notice that the new term becomes the active term and you can modify it immediately.

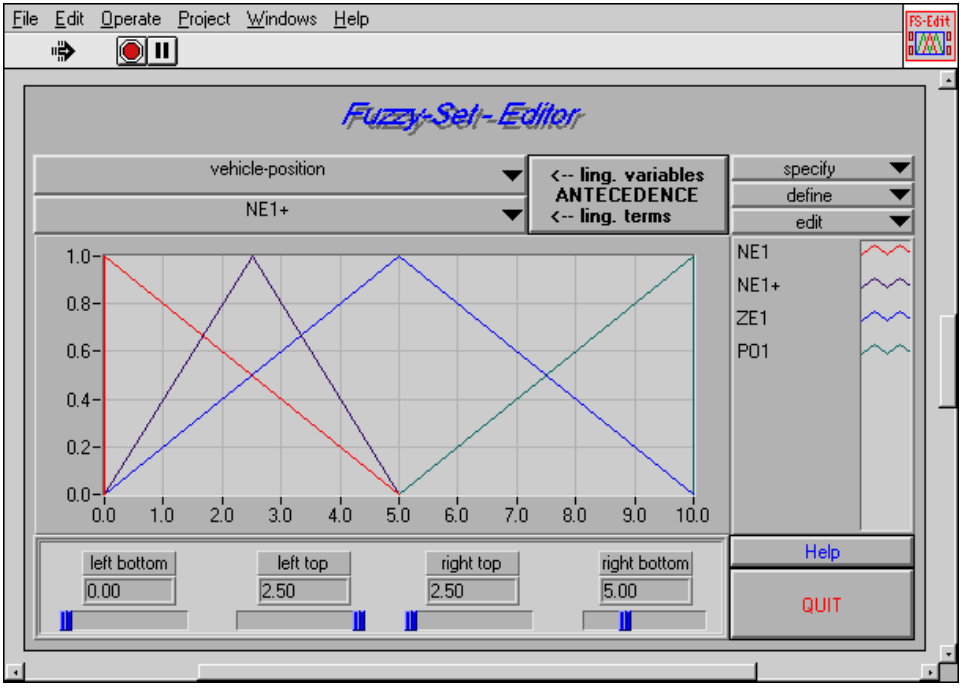


Figure 8-9. New Term Added to the Vehicle-Position Variable

Note Adding a new term to an input variable, especially one that is part of an existing project, causes significant changes to the rule base. Additional rules automatically extend the rule base. Each rule has a conclusion that is predefined as none. Adding a new consequence term only extends the possibility to select conclusion terms within the Rulebase-Editor. Remember that each input and output variable can have a maximum of nine linguistic terms.

To add the second new term between ZE1 and PO1, first select ZE1 from the Term Selector. With ZE1 as the active term, you can select **define»add term after** to add the new term. LabVIEW adds the new term, ZE1+, to the Term Display, as shown in Figure 8-10.

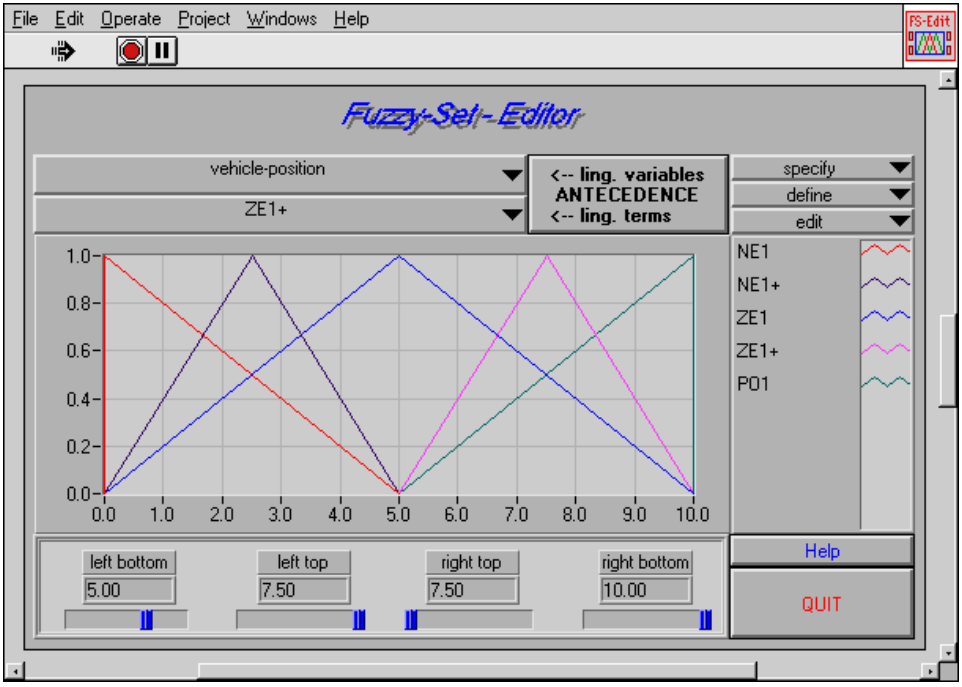


Figure 8-10. Another New Term Added to the Vehicle-Position Variable

Before rearranging the linguistic terms according to the desired pattern, select **specify»rename term** to assign the correct term identifiers. Refer to Figure 5-6, Linguistic Variable Vehicle Position x and Its Linguistic Terms, for more information about the desired pattern. Figure 8-11 shows an intermediate state and Figure 8-12 shows the final result of this renaming process.

Note You also can use the **specify** menu to add or remove linguistic variables.

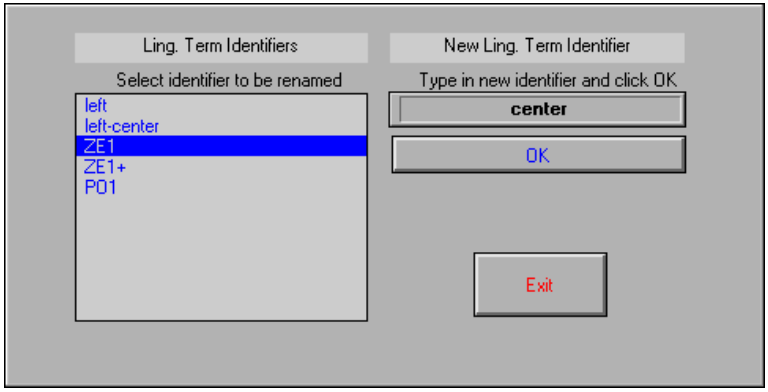


Figure 8-11. *Rename Term* Dialog Box

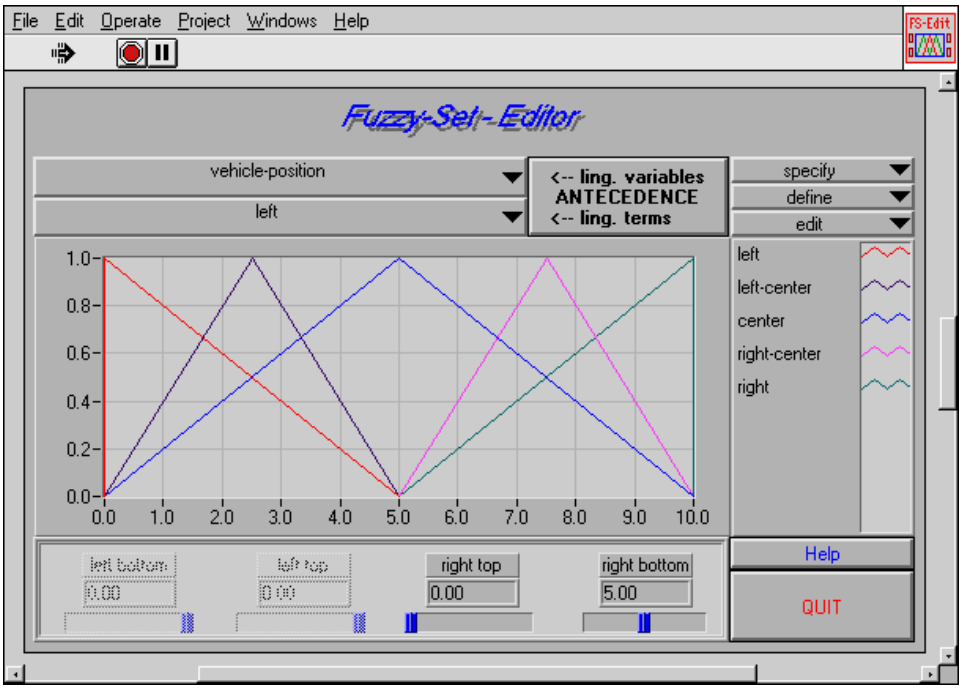


Figure 8-12. All Vehicle-Position Terms Named Correctly

The Fuzzy-Set-Editor offers many functions that you can use to modify single terms or the whole term arrangement of the active variable. It is a good idea to experiment with this function at this point in your project because you must modify the whole term arrangement according to the desired term arrangement shown in Figure 5-6, Linguistic Variable Vehicle Position x and Its Linguistic Terms. Figure 8-13 shows the term arrangement you obtain when you select **edit»full term-overlap all**, which results in a term arrangement with all terms of the active linguistic variable completely overlapping each other.

The **edit** menu also has several other functions for automatically editing membership functions. You can change individual membership functions, or all of the membership functions, to singleton fuzzy sets, which are typically used only for controller output. The tolerance function changes a trapezoidal membership to a triangular function. In addition, you can set the overlap between functions and make all functions symmetric. This command does not affect the left side of the left-most term and the right side of the right-most term.

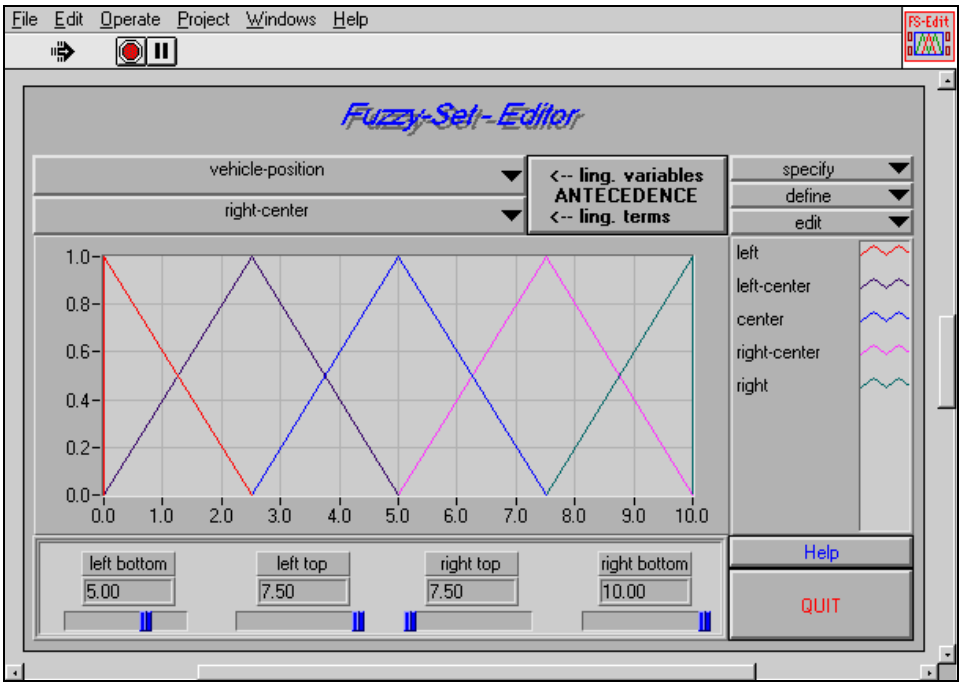


Figure 8-13. A Term Arrangement of Completely Overlapping Terms

With the Fuzzy-Set-Editor functions described in this section, you can edit all linguistic variables, including the desired term arrangements for the `FuzzyTruck` example project. Figure 8-14 shows the result of the complete editing session.

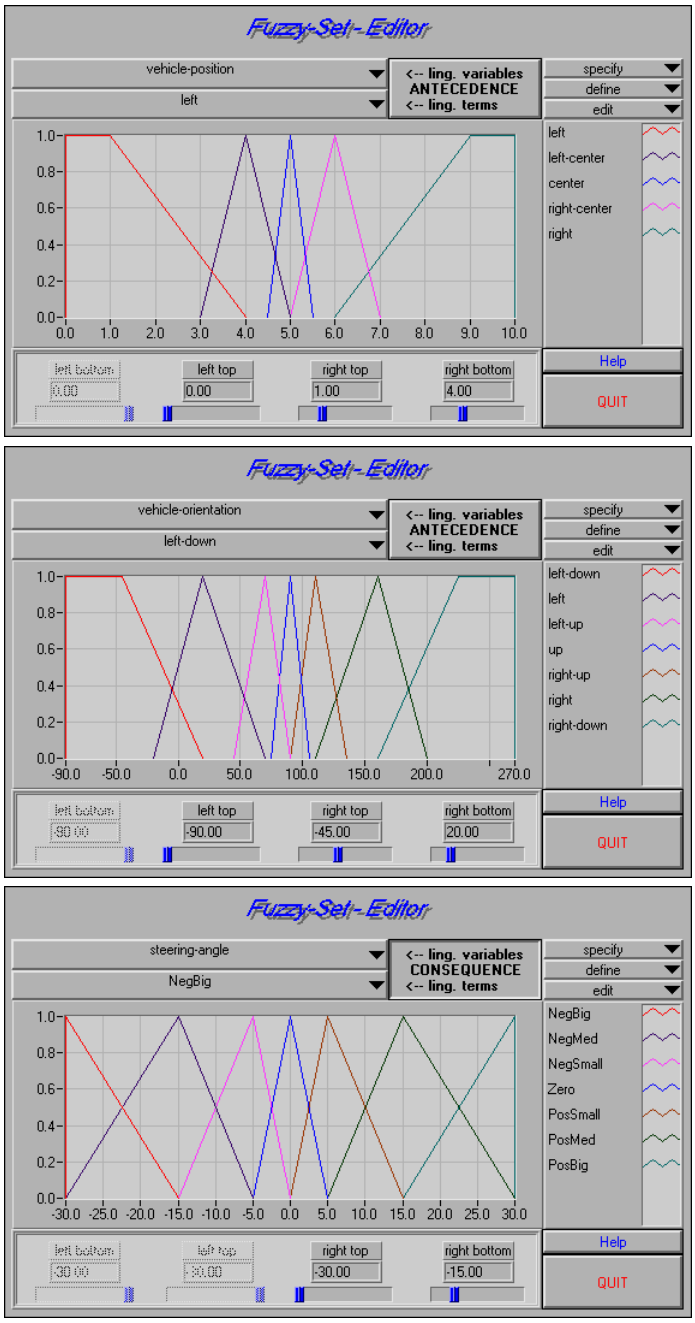


Figure 8-14. Results of the Complete Editing Session Example

Rulebase-Editor

After you enter all the linguistic information of the application example into your `FuzzyTruck` project, you can begin editing. The rule base represents expert knowledge about the vehicle maneuvering process.

If it is not already active, select **File»Open** to load the example project, `FuzzyTruck`. Select **Edit»Rulebase** to open the Rulebase-Editor.

Because you have not explicitly entered or modified a rule at this point in the example project, the Rulebase-Editor begins with a project-specific, complete default rule base.

LabVIEW assigns each possible combination of linguistic terms, for each of the input variables to a single rule with its consequence part set to **none**.

The Rulebase-Editor offers a rule base that contains 35 rules because there are five terms for the first input variable, *vehicle-position*, and seven terms for the second input variable, *vehicle-orientation*.

If there are more than 15 rules available, LabVIEW activates a scrollbar, to access the rules not currently displayed on the Rulebase-Editor front panel.

Each rule is associated with a weight factor to enhance or reduce the influence of a rule on the controller characteristic. The DoS ranges from 0.0 to 1.0. In a default rule base, all DoS values are automatically set to 1.0. Use the **Utils** menu to set weights for all rules.

Use weight factors in combination with other techniques, such as genetic algorithms, to optimize controller performance.

Figure 8-15 shows the project-specific complete default rule base.

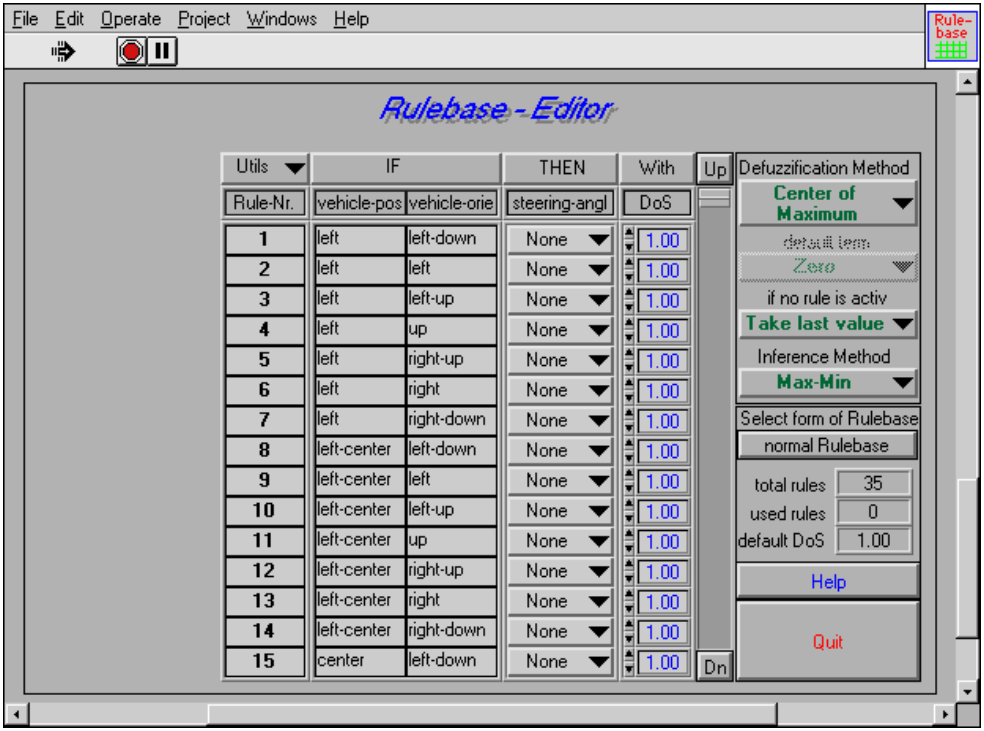


Figure 8-15. Project-Specific Complete Default Rule Base

The Rulebase-Editor front panel also contains menu buttons for interactively selecting the defuzzification method and inference method. If the default setting does not fit the application needs, you can change the default controller output for situations with no active rules.

Figure 8-16 shows the scrollbar that LabVIEW activates when there are more than 15 rules available.

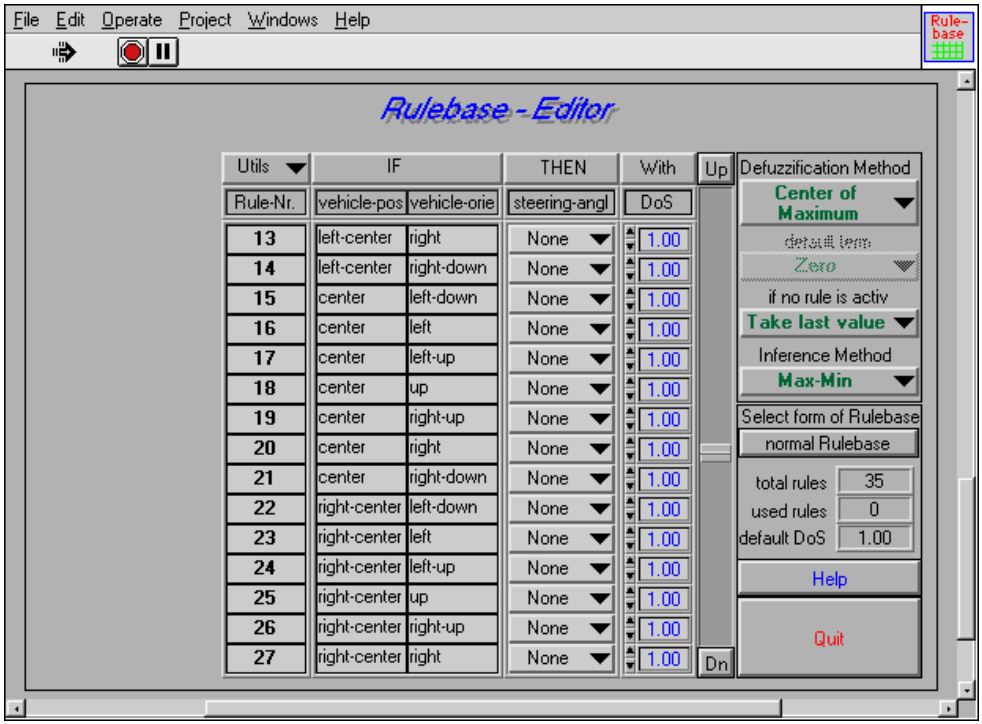


Figure 8-16. Using the Rulebase-Editor Scrollbar

Enter the desired consequence of each rule to begin editing the rule base. The consequence part of each rule is implemented as a term selection box containing all possible consequence terms. You can select a consequence term from the term selection box to specify the consequence of a particular rule.

According to the rule base specified in Figure 5-9, *Complete Linguistic Rule Base*, if the vehicle position is *left* and the vehicle orientation is *left down*, the consequence term is *negative small*. When you select **NegSmall** from the term selection box of the consequence part, the THEN part the first rule of the rule base is

IF vehicle-position is *left* AND vehicle-orientation is *left down*, THEN set steering-angle to *negative small*.

You can enter the complete rule base this way. The IF part of the Rulebase-Editor panel automatically accommodates the number of input variables used in the fuzzy controller.

Next, select an appropriate defuzzification method. Because there must be a continuous output signal for the steering angle control, you must select a defuzzification method that calculates the best compromise. Follow the guidelines in Table 7-1, *Comparison of Different Defuzzification Methods*, to choose either the CoM method or the CoA method.

Select the defuzzification method from the appropriate selector on the Rulebase-Editor panel, as shown in Figure 8-17.

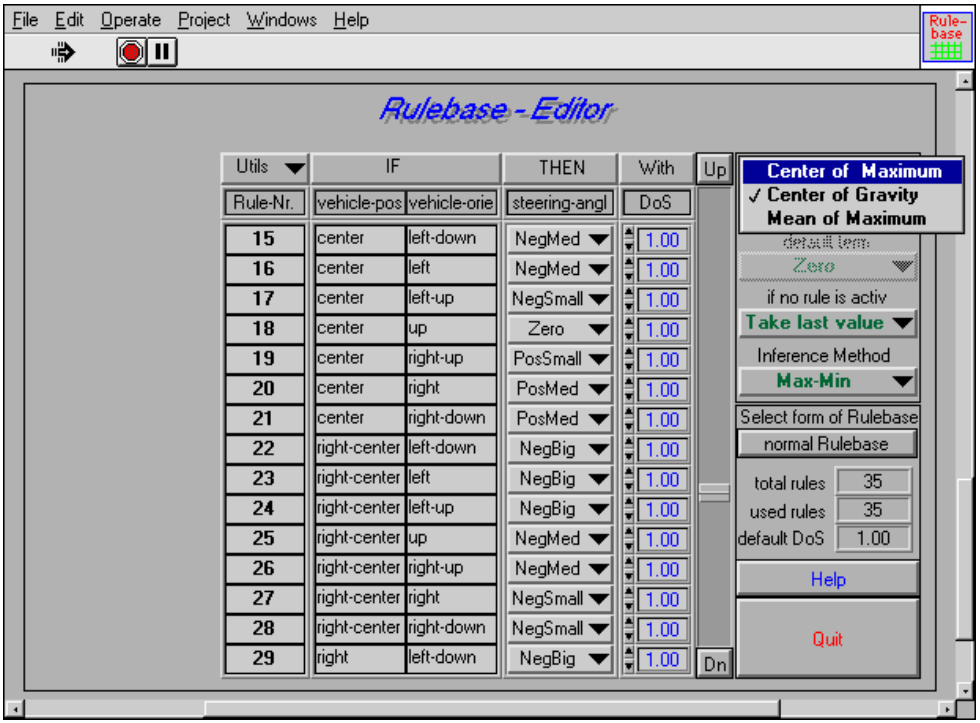


Figure 8-17. Selecting a Defuzzification Method

You can use the default setting shown in Figure 8-18 as the default controller output. The default setting does not affect the application example because the fuzzy controller has a complete rule base and overlapping term arrangements. In the example, no input variables have definition gaps or undefined intervals. Refer to Figure 6-10,

I/O Characteristic of a Fuzzy Controller (Undefined Input Term Interval), for more information about input variables.

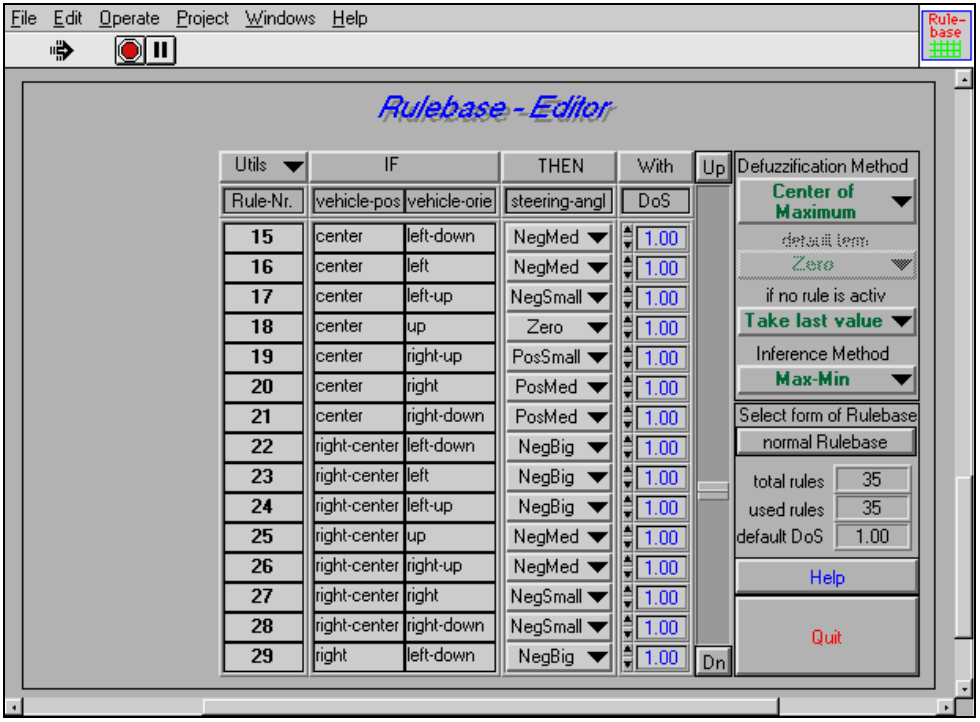


Figure 8-18. Default Settings for Default Controller Output and Inference Method

The design work for the example project is complete. It is time to save the project and see what documentation features are available for the Fuzzy Logic Controls.

Documenting Fuzzy Control Projects

The **File»Print** submenu offers documentation facilities for printing information about the active project. Select **Print»Complete Documentation** to print the complete controller documentation for the example project.

Test Facilities

Before you run a fuzzy controller within a designated system environment, study the I/O characteristics of the controller within the toolset. You can use these characteristics to optimize the fuzzy controller and make any necessary modifications. The Fuzzy Logic Controls provide an appropriate test environment.

Select **Test»I/O-Characteristics** to call the test facility to perform the I/O characteristic studies of a fuzzy controller.

For the application example, `FuzzyTruck`, previously loaded, the **I/O-Characteristic** test facility starts with a front panel similar to the one shown in Figure 8-19.

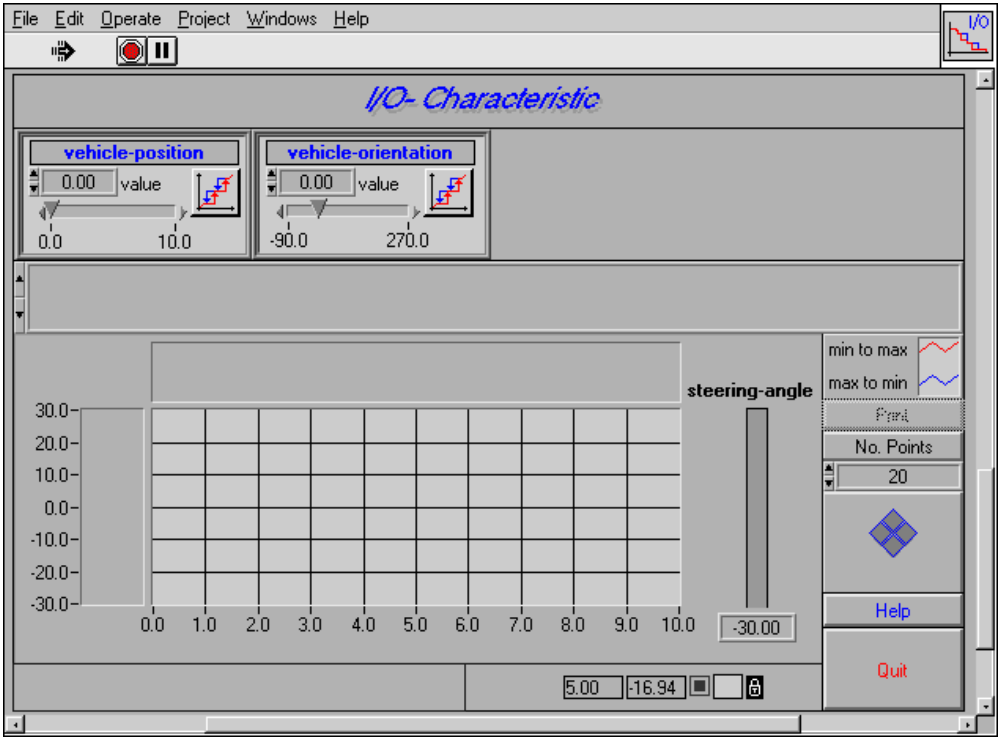


Figure 8-19. I/O-Characteristic Project-Specific Front Panel

There is a different parameter control block in the Input Parameter Field of the I/O-Characteristic front panel for each input variable of the fuzzy controller. The toolset uses the blocks to set up the desired test conditions for the different controller inputs.

Suppose you want to vary the vehicle-position within the input data range and keep the vehicle-orientation constant at 0° to observe how the behavior of the controller output variable, *steering-angle*, changes with the *vehicle-position* and the *vehicle-orientation*.

To set up these test conditions, first enter the desired test value into the parameter control block for vehicle-orientation, as shown in Figure 8-20.

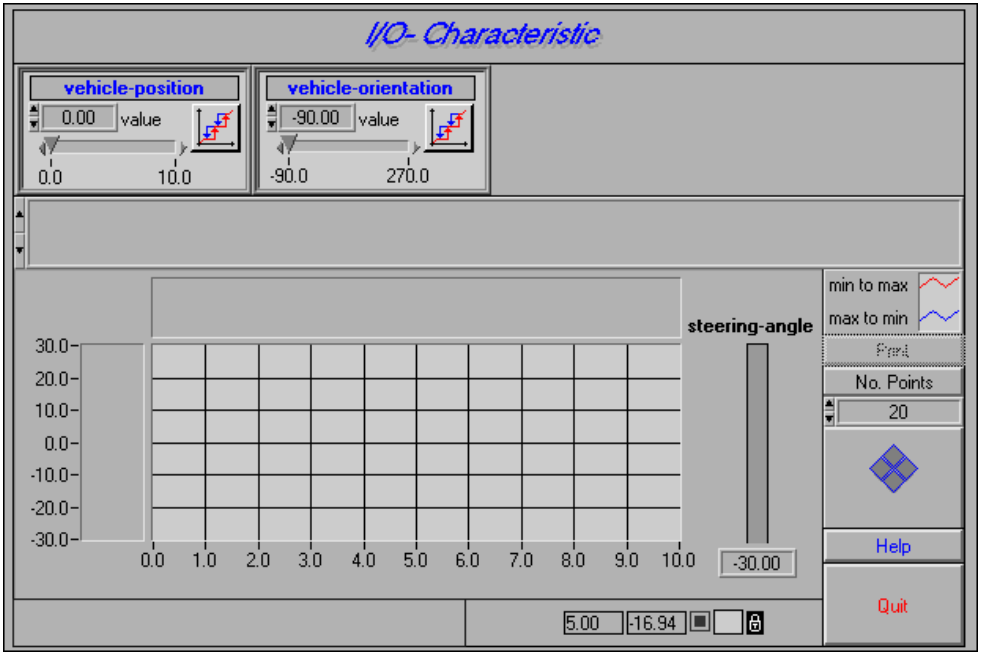


Figure 8-20. Entering a Test Condition into a Parameter Control Block of the I/O-Characteristic Front Panel

Then, click the **Run** button to begin calculating the I/O characteristic within the parameter control block for vehicle-position.

LabVIEW executes the I/O characteristics calculation according to the number of points specified in the **No. Points** control box. To animate the calculation process, move the slider of the varying input variable.

Note The controller characteristic is calculated twice, varying the activated input variable, which is vehicle-position in this example, from the minimum value up to the maximum value, and vice versa. This happens because of possible hysteresis effects that occur with incomplete rule bases. Definition gaps in the term arrangement of the input variable, which cause the controller to use the default output value or the last originally-computed value, can also cause LabVIEW to calculate the controller characteristic twice.

As soon as the characteristic calculation completes, LabVIEW displays the characteristic curve in the I/O-Characteristic display, as shown in Figure 8-21.

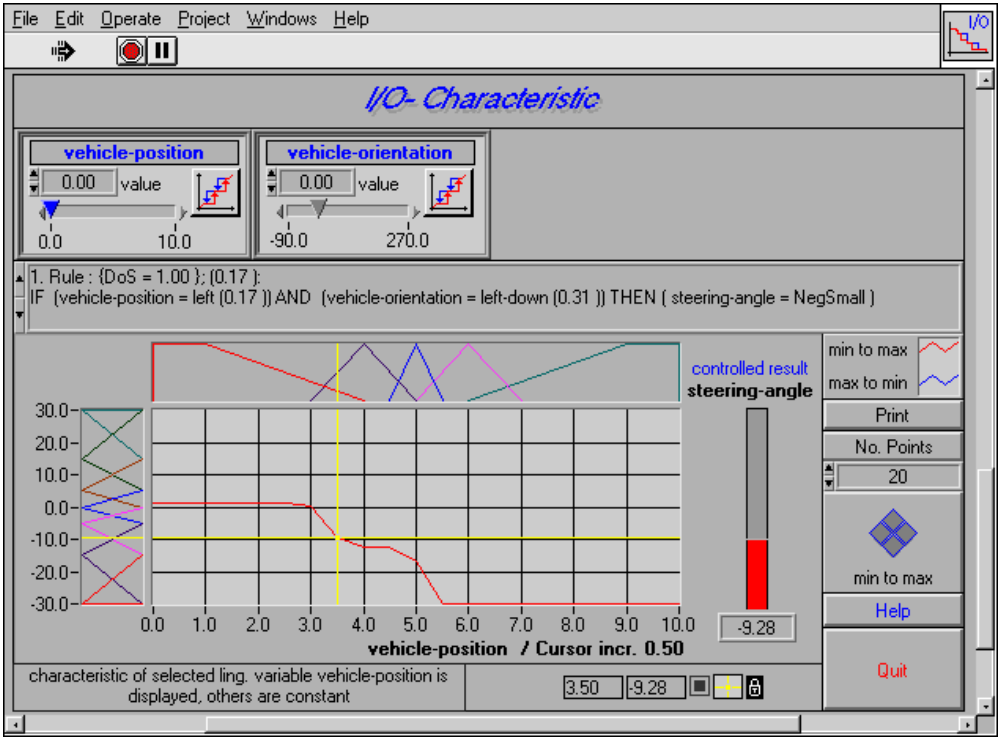


Figure 8-21. Controller Characteristic Displayed

The I/O-Characteristic display contains a cursor that you can control with the Cursor Navigation block. The cursor can travel along the characteristic curve and identify the active rules for the input situation at each cursor position. The I/O Characteristic function panel displays the current input values and controller output value.

The Active Rules display shows all active rules within the input situation determined by the cursor position, including the degree of truth for each antecedence term. You can select each active rule as shown in Figure 8-22.

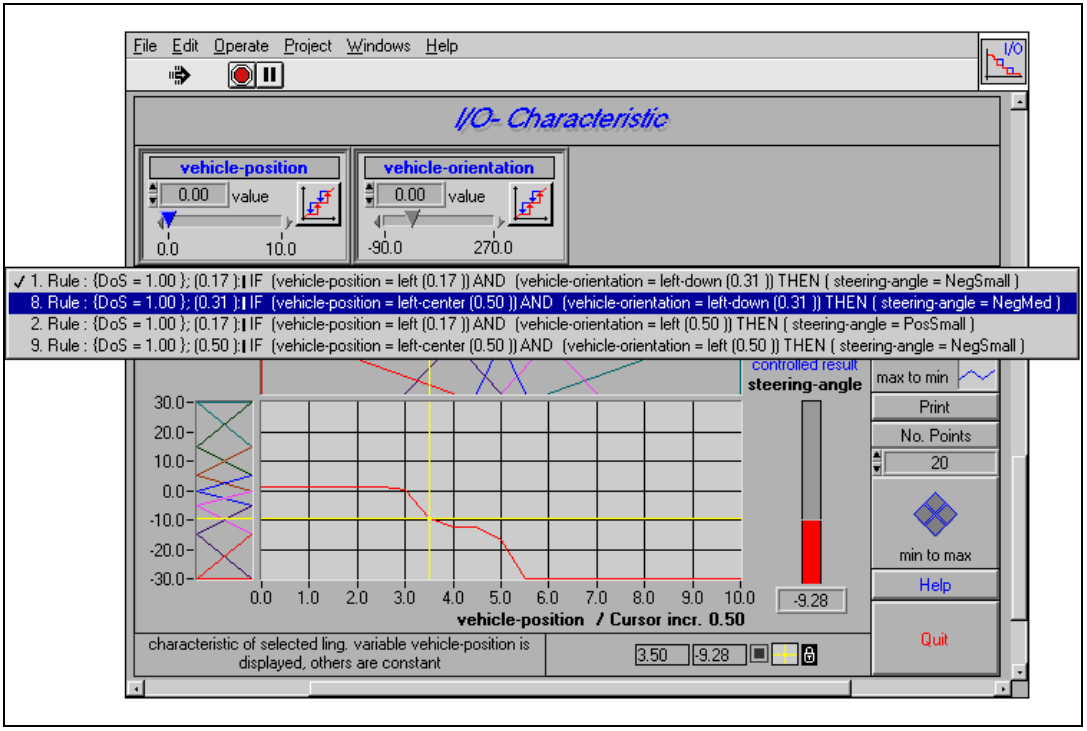


Figure 8-22. Selecting One of the Active Rules from the Active Rules Display

Click the **Print** button to print out the current situation for documentation purposes.

9

Implementing a Fuzzy Controller

This chapter describes how to implement a fuzzy controller and includes a pattern recognition application example. There are several different ways to use the Fuzzy Logic VIs to implement a fuzzy controller. The easiest implementation uses the Fuzzy Controller VI.

Pattern Recognition Application Example

Suppose you need to develop and implement a fuzzy controller that identifies the shape of different-sized triangular, hexagonal, and rectangular plastic parts moving on a conveyor belt through a simple reflex light barrier, as shown in Figure 9-1.

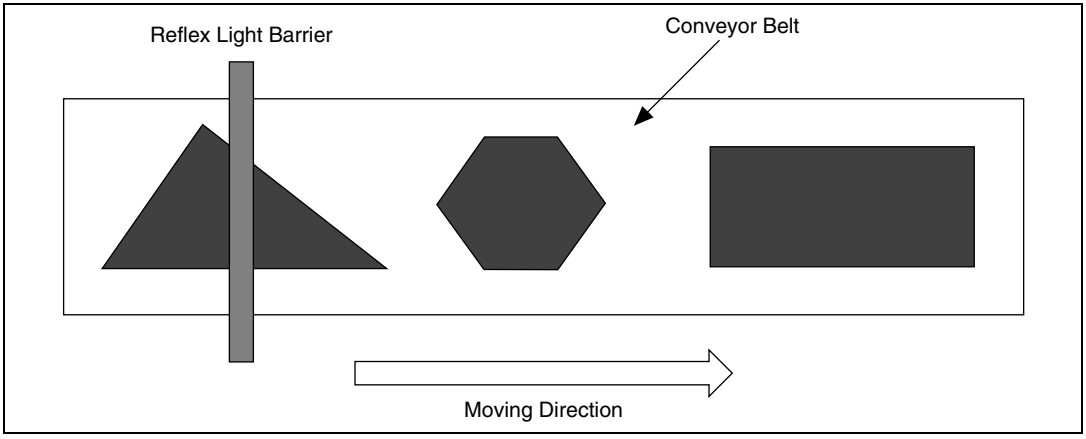


Figure 9-1. Sensor Facility

The plastic parts can be symmetric or asymmetric. The reflex light barrier reads a characteristic voltage signal for each plastic part. The signal depends on the resistances set up on the light barrier. Measuring these signals with a real sensor shows that even the signals of identical plastic parts vary to a certain extent. Different environmental conditions such as

scattered light can affect the signal. Figure 9-2 shows some typical voltage drop curves derived from an asymmetric triangle, a lefthand-shaped triangle.

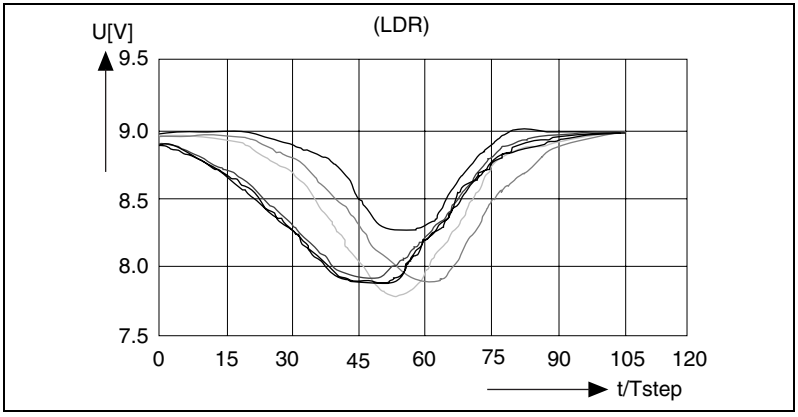


Figure 9-2. Typical Voltage Drop Curves Obtained from a Lefthand-Shaped Triangle

To obtain a simple but efficient controller, abstract the curves shown in Figure 9-2 into the idealized curve outline that is shown in Figure 9-3.

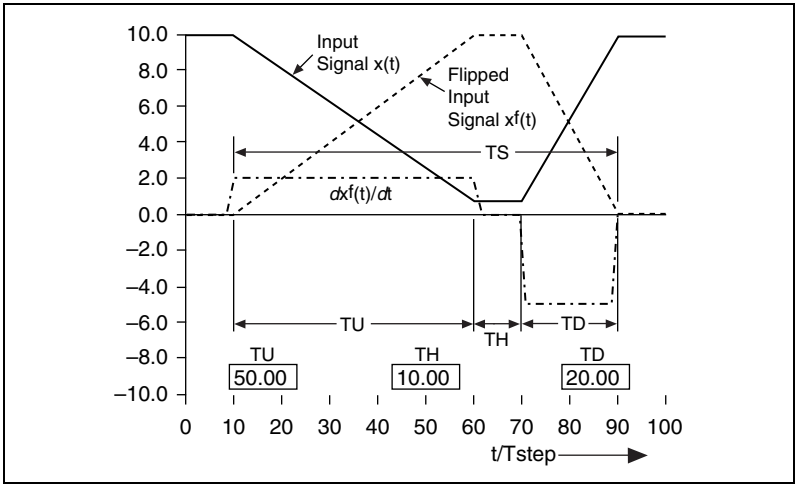


Figure 9-3. Abstract Voltage Drop Curve for Feature Extraction

There are three distinguishable parts of the flipped input signal represented by the dashed curve $x^f(t)$ in Figure 9-3. There is a rising curve part, a constant part, and a falling curve part. Differentiation of the flipped input signal yields the dash-dotted curve, $dx^f(t)/dt$, from which you can derive the

time intervals TU (up), TH (hold) and TD (down). When TS (signal) represents complete operation time, you can extract the following features for the desired pattern recognition:

TH / TS ≈ 0 ==> Triangle	(TU–TD) / TS > 0 ==> lefthand-shaped
0 < TH / TS > 1 ==> Hexagon	(TU–TD) / TS ≈ 0 ==> symmetrical
TH / TS ≈ 1 ==> Rectangle	(TU–TD) / TS < 0 ==> righthand-shaped

Note You can use existing functions or functions you can write in LabVIEW to execute all the signal processing steps described above.

Because the real sensor signal is not an idealized signal as shown above, the characteristic features derived from it are not precise. You can model them directly by the appropriate linguistic terms for the two linguistic input variables TH/TS and (TU–TD)/TS. Using the Fuzzy Logic Control as described in Chapter 8, *Using the Fuzzy Logic Controller Design VI*, the term arrangements shown in Figures 9-4 and 9-5 exist for the input variables TH/TS and (TU–TD)/TS.

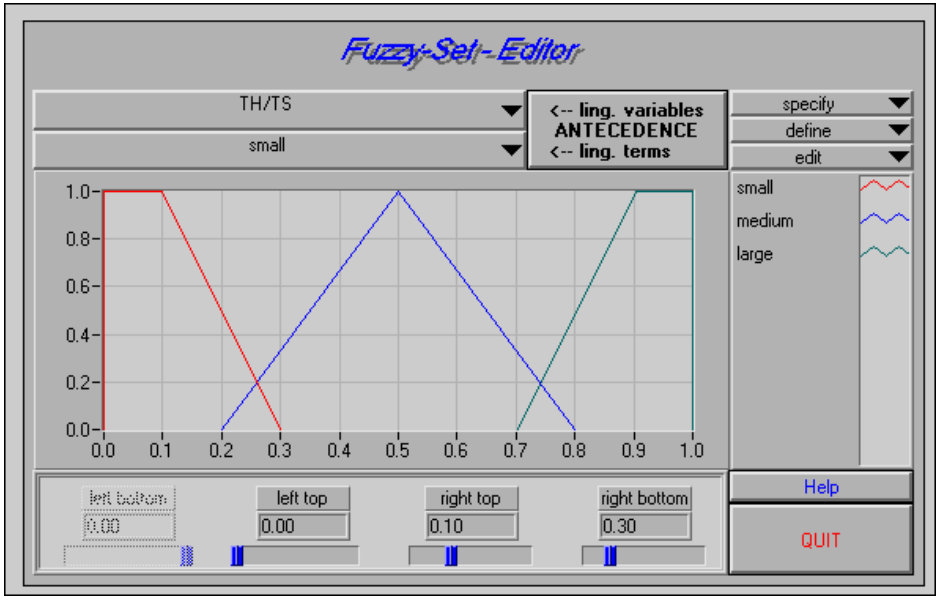


Figure 9-4. Linguistic Term Arrangement of Input Variable TH/TS

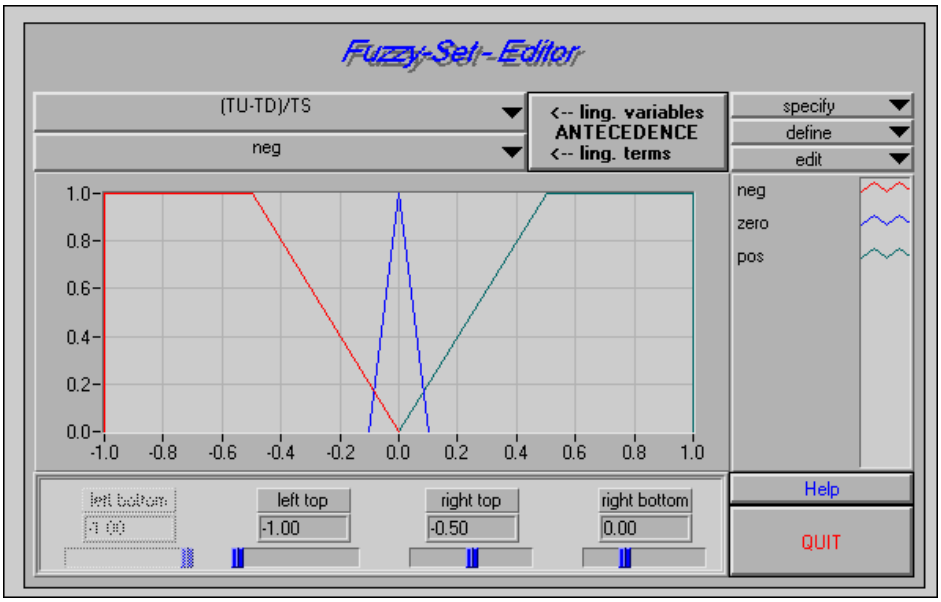


Figure 9-5. Linguistic Term Arrangement of Input Variable (TU–TD)/TS

The linguistic output variable *object* can be composed of singletons, each of which represents a specific shape. Figure 9-6 shows the term arrangement and Figure 9-7 shows the rule base.

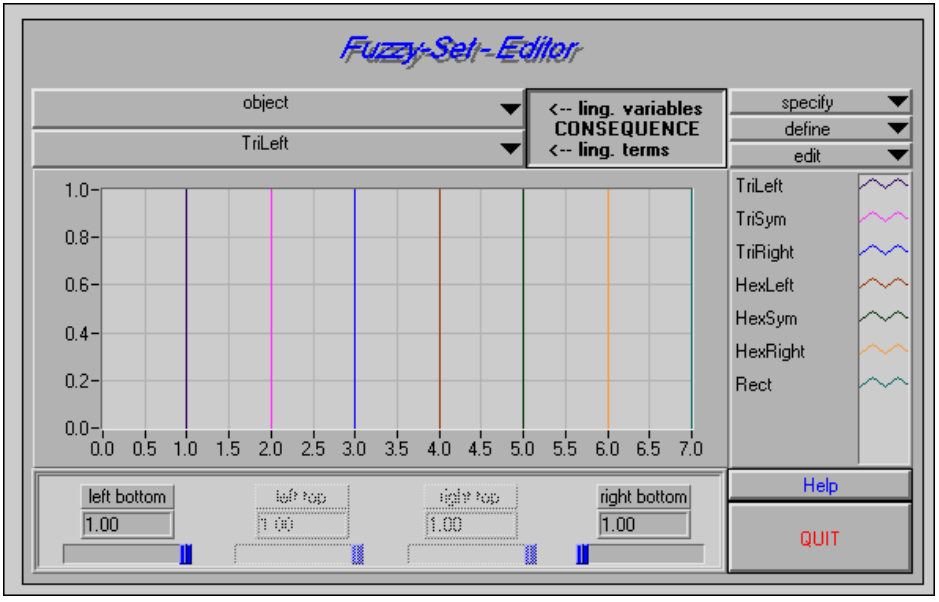


Figure 9-6. Linguistic Term Arrangement of the Output Variable, Object

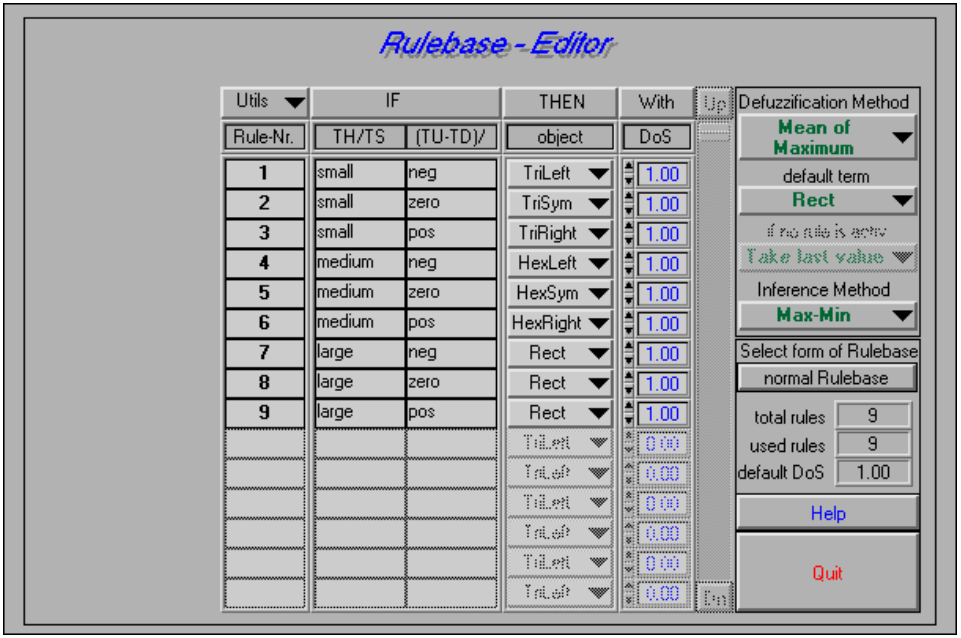


Figure 9-7. Complete Rule Base Describing the Pattern Recognition Process

The principal program structure of the pattern recognition facility is a loop structure, which repeatedly takes the input signal from a data acquisition board using the easy I/O VIs, for example, and processes the signal Consider the following simulation environment to experiment with the fuzzy controller independent of specific data acquisition equipment.

The SignalGen VI on the left side of the block diagram shown in Figure 9-8 corresponds to the input side of a process controller. You can regard the NumtoString VI on the right side of the diagram as the output side of a process controller. The VI supplies all necessary output signals, including the signals used for process animation.

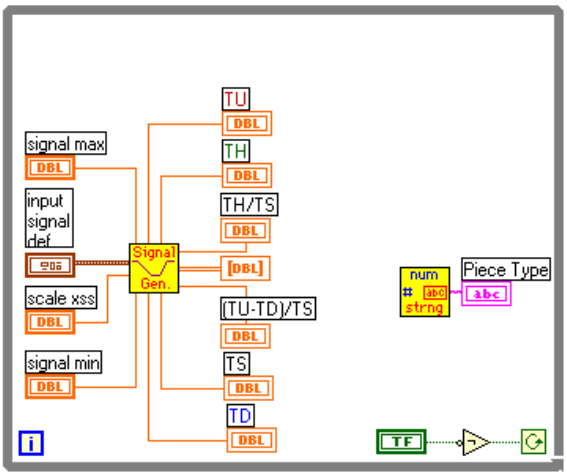


Figure 9-8. Block Diagram of the Pattern Recognition Application Prepared for Entering the Pre-Defined Fuzzy Controller VI

The SignalGenVI replaces the data acquisition part, including all the data pre-processing activities, which directly supplies the necessary input signals, TH/TS and (TU–TD)/TS, for the example application. All other input and output signals used in the block diagram are part of the user interface that includes all the controls and indicators you can use to adjust the pattern recognition application example. Figure 9-9 shows the front panel of the example.

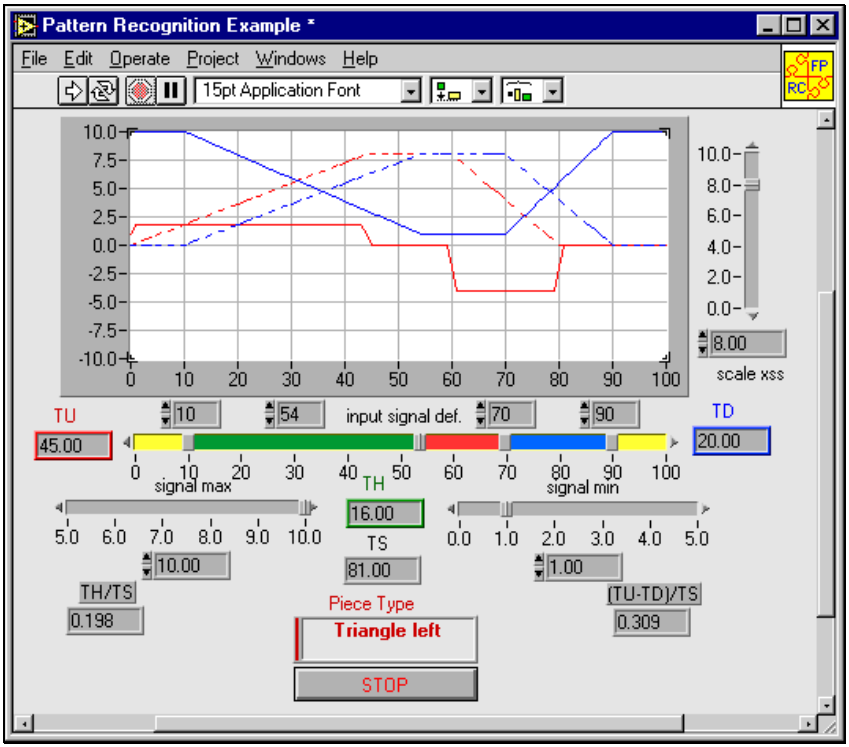


Figure 9-9. Front Panel of the Pattern Recognition Application

You can use the **input signal def** sliders to simulate the signal from the reflex light barrier of the real system. You also can modify the **signal max** and **signal min** sliders to use them to test how the fuzzy controller works despite having a signal with a very small amplitude The **scale xss** slider models a gain factor towards the signal that the data pre-processing step performs. You also can use the slider to study how different signal conditions can affect the result of the pattern recognition process.

Fuzzy Controller Implementation

Now incorporate the fuzzy controller into the application block diagram. You do not need to program the fuzzy controller, just use the pre-defined Fuzzy Controller VI available with the Fuzzy Logic Controls, shown in Figure 9-10.

The pre-defined Fuzzy Controller VI can be connected with as many as four input signals from a process and one output signal used as a control value. Although the Fuzzy Controller VI has many different inputs and outputs, at this time you only need those inputs and outputs shown in bold in Figure 9-10.

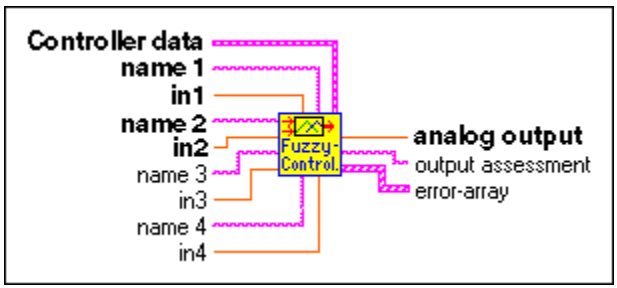


Figure 9-10. Fuzzy Controller VI

You can connect the input signals TH/TS and (TU–TD)/TS to the Fuzzy Controller VI inputs **input1** and **input2**. You also can connect the output signal of the Fuzzy Controller VI, called **analog output**, to the input side of the NumtoString VI. Leave the rest of the inputs unconnected at this time.

Loading Fuzzy Controller Data

You can compare the Fuzzy Controller VI to a microprocessor that does not have an executable program loaded. To obtain the specific data for the fuzzy controller, you must use the Load Fuzzy Controller VI to load the required data into the Fuzzy Controller VI. This VI also is included in the Fuzzy Logic Controls.

Because the controller data must be loaded into the Fuzzy Controller VI when the pattern recognition application is started, place it outside the While-Loop, as shown in Figure 9-11.

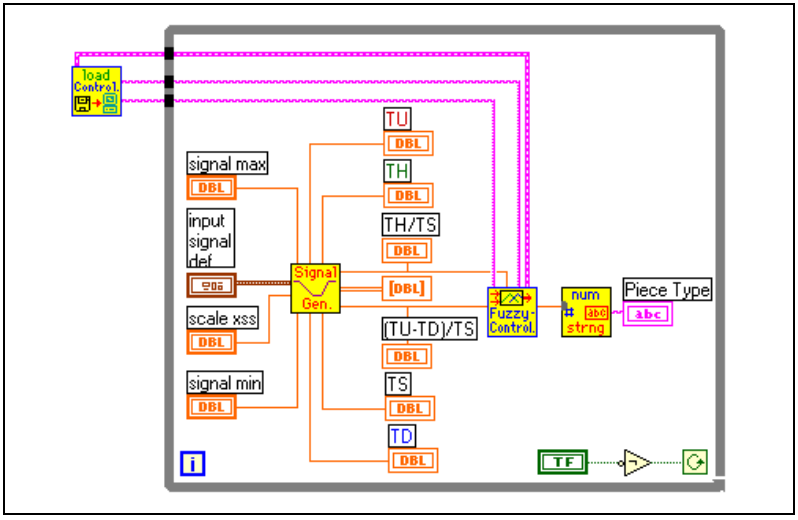


Figure 9-11. Block Diagram of the Pattern Recognition Application

The application example is complete. You can switch back to the front panel from the fuzzy controller and run the VI to start the pattern recognition application.

Immediately after the application begins, a file dialog box prompts you to enter the name of a file that contains the appropriate controller data. Open the project file `FCPR.fc`, which represents the fuzzy controller you designed earlier.

When you load the Fuzzy Controller, drag the sliders to try different settings for the pattern recognition process. You can see how the pattern recognition process changes with different input signal conditions. Refer to Figure 9-12.

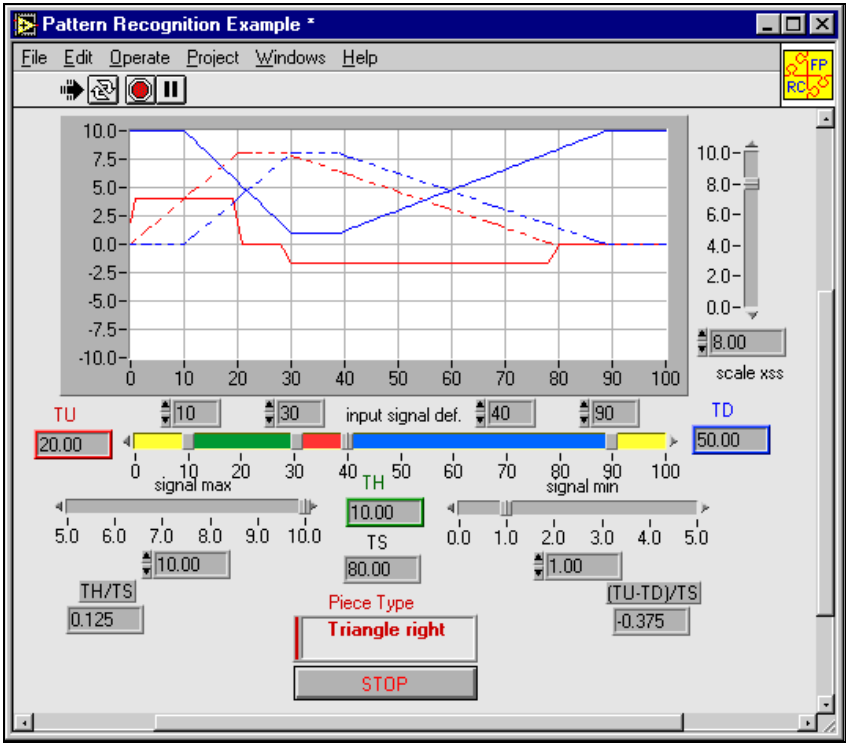


Figure 9-12. Running the Pattern Recognition Application

Selecting the **Cancel** button rather than selecting the fuzzy controller data file, `FCPR.fc`, executes the default fuzzy controller repeatedly. Without having actual data loaded to the controller, it will use the default data. See the block diagram of the complete pattern recognition application shown in Figure 9-11.

Because of security aspects that can occur when running a controller within a real application environment, if someone selects the Cancel button, the controller should not start. To improve your controller design, place the While Loop into a Case Structure and connect the **selection terminal** to the **cancel** output of the Load Fuzzy Controller VI. Figure 9-13 shows the result. The **TRUE** case is empty, and the application quits if you click **Cancel**.

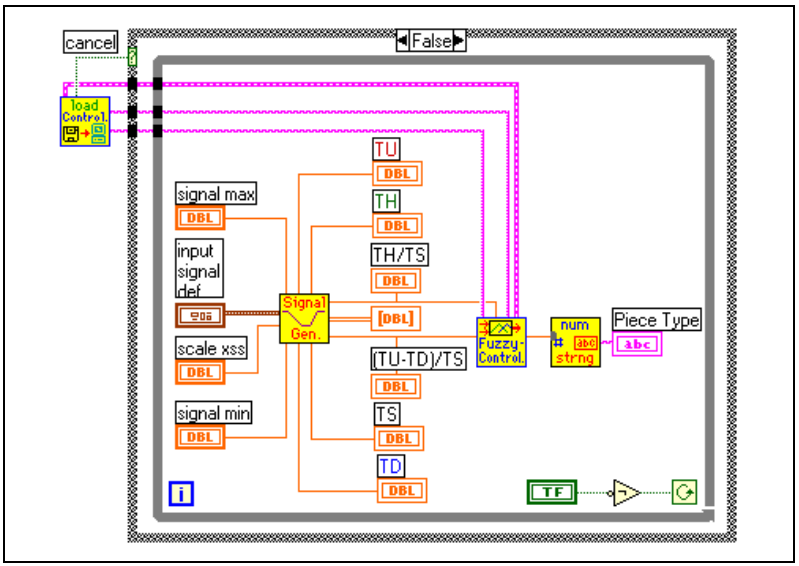


Figure 9-13. Improved Controller Application Block Diagram

The complete pattern recognition application example also is available within the Fuzzy Logic Controls.

Saving Controller Data with the Fuzzy Controller

You might want to use a fuzzy controller like a predefined VI that you do not have to load to run. You might wonder how the currently valid controller data file can be the default for the controller so you can use it as a stand-alone controller.

Complete these steps to build a stand-alone Fuzzy Controller VI for the pattern recognition application example.

1. Bring the application block diagram to the front and double-click the icon of the Fuzzy Controller VI to open the VI.
2. Bring the application front panel to the front.

3. Start the application to open the input file dialog box that requests a fuzzy controller data file.
4. Select the desired fuzzy controller data file.
5. Stop the application.
6. Bring the front panel of the Fuzzy Controller VI to the front.
7. Select **Operate»make current values default** to make the currently valid controller data the default.
8. Choose one of two options. Save a copy of the Fuzzy Controller VI if you want it to be available under a unique name. Select **No** when asked to save the original Fuzzy Controller VI. Or, you can save the original Fuzzy Controller VI, which now has the current controller data as default values. Only the default values of the original Fuzzy Controller VI have been changed. You can still use the VI as a general-purpose Fuzzy Controller VI because the VI only uses the default values when you apply the controller without loading specific data into the VI.
9. Close the application.

Now you can use either the new VI or the modified one as a stand-alone fuzzy controller as shown in Figure 9-14.

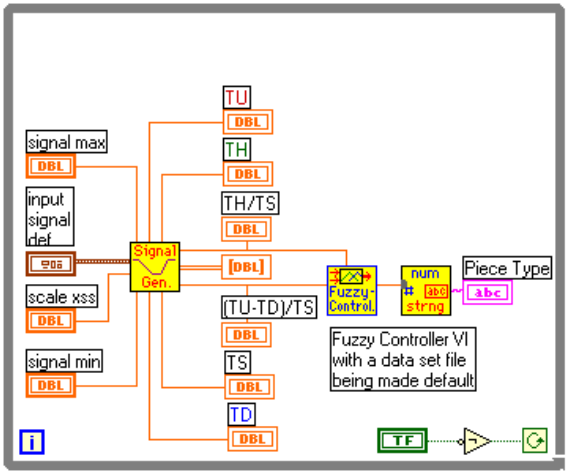


Figure 9-14. Application Block Diagram with Stand-alone Fuzzy Controller VI

Testing the Fuzzy Controller

There is another predefined VI available with the Fuzzy Logic Controls that you can use to build or test fuzzy control applications. The Test Fuzzy Control VI supplies a fuzzy control test and application environment for as many as four different controller inputs. Input assignment is set automatically according to the data being loaded into the controller. This VI was created to show the proper use of all input and output signals supplied by the Load Fuzzy Controller VI and the Fuzzy Controller VI. Figure 9-15 shows the Test Fuzzy Control VI front panel.

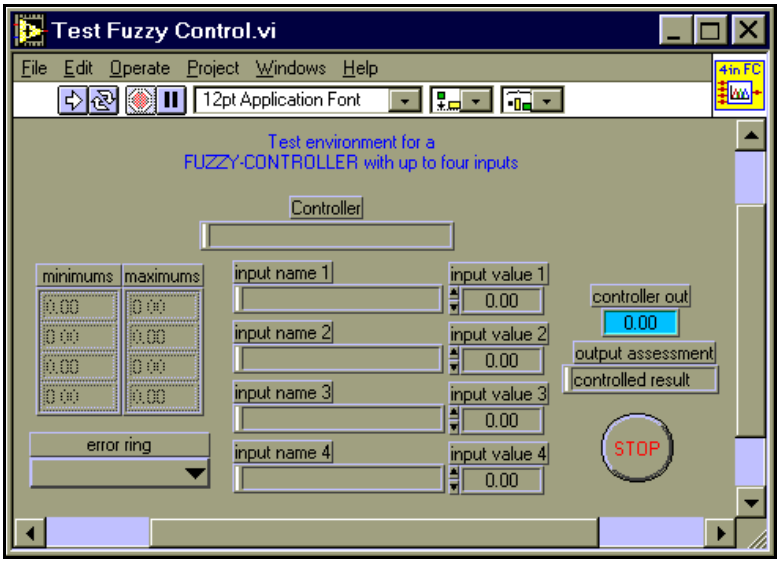


Figure 9-15. Test Fuzzy Control VI Front Panel

The controller displays the fuzzy controller project identifier as soon as you load the fuzzy controller data file. The input name displays the identifiers of all used inputs. The minimum and maximum display the appropriate currently valid data range for each used input variable. You can use **input value** to enter input values to stimulate the controller. The controller out displays the output value. The lower data range values automatically initialize the corresponding input value.

Figure 9-16 shows the application front panel immediately after loading the fuzzy controller data file for the pattern recognition example.

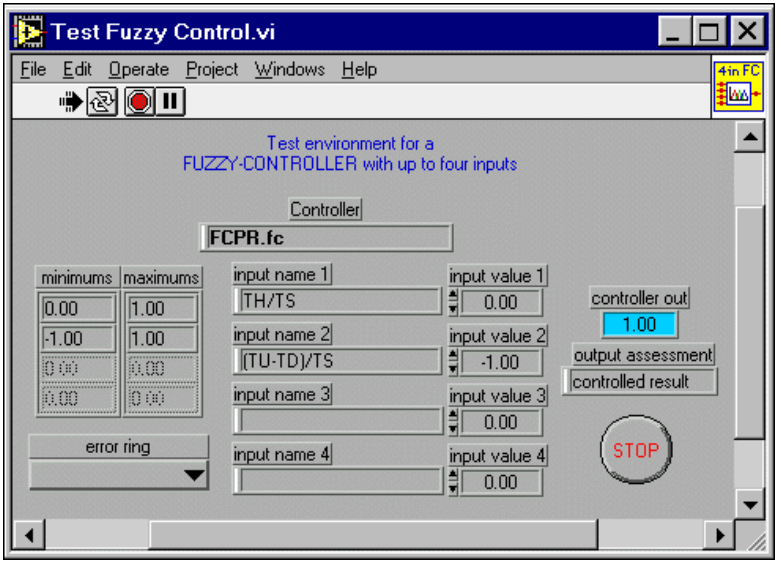


Figure 9-16. Test Fuzzy Control VI Front Panel with Controller Data Loaded

Remember that if there is an input situation not covered by active rules, a fuzzy controller uses default values. The output assessment displays a message to indicate such a situation.

If input values exceed the data range assigned to the related input variable, the error ring displays an error message and the output value is set to the default output value, as shown in Figure 9-17.

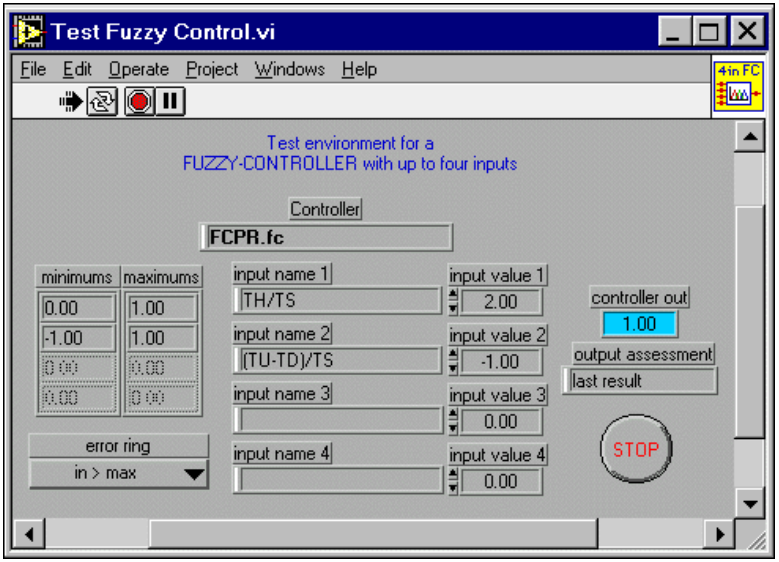


Figure 9-17. Test Fuzzy Control VI Front Panel with Incorrect Input Value for Input 1

Figure 9-18 shows the proper use of all input and output signals supplied by the Load Fuzzy Controller VI and the Fuzzy Controller VI. You can use this program structure as a basis for building your own fuzzy logic applications.

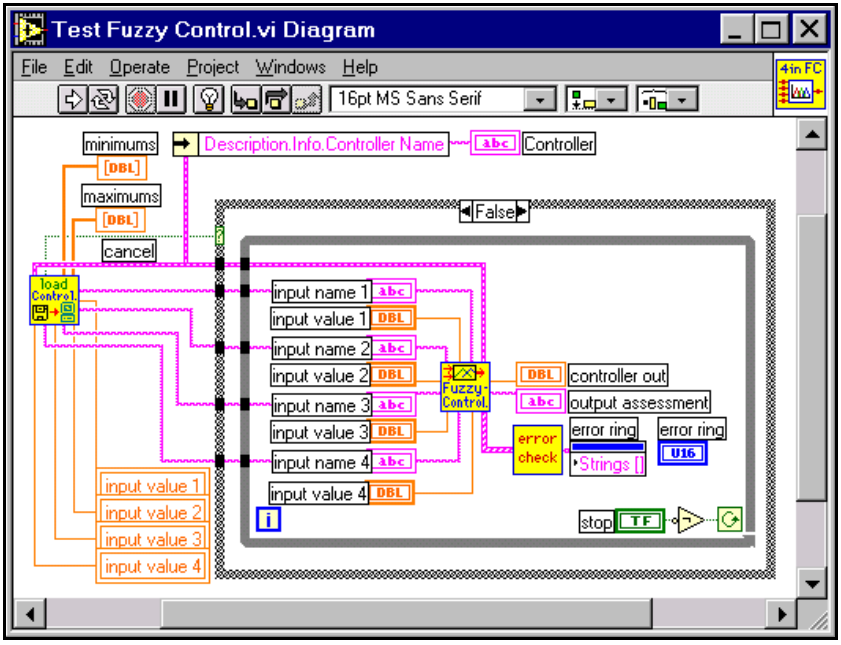


Figure 9-18. Test Fuzzy Control VI Block Diagram Example

Note You can connect the inputs and the controller output directly to the outputs and inputs of the DAQ VIs available in LabVIEW in order to use real process data from sensors instead of the values from the panel controls as shown in Figures 9-16 and 9-17.

Part III

Advanced Control

This section of the manual describes the Advanced Controls in the PID Control Toolset.

- Chapter 10, *Advanced Control*, describes the Advanced Control Functions in the PID Control Toolset and provides examples of control systems you can create with the Advanced Control Functions.

10

Advanced Control

The Advanced Control VIs are divided among the following three subpalettes: **Continuous Linear**, **Discrete Linear**, and **Nonlinear**. Each palette provides basic control function blocks that you can use together to develop advanced control algorithms. In addition, you can use these VIs to simulate physical systems for Hardware-In-the-Loop (HIL) simulation applications.

Continuous Linear VIs

The Continuous Linear VIs include the Integrator, Derivative, and Transfer Function VIs. You can use these VIs to simulate real-world, continuous signals. For example, the voltage in a simple analog electronic circuit is a continuously varying signal.

Discrete Linear VIs

The Discrete Linear VIs include the Discrete Integrator and Discrete Filter VIs. You can use these VIs to implement discrete digital controllers with your computer. You can use modern control design methods to create control algorithms and use the Discrete Linear VIs to implement the algorithms in LabVIEW or LabVIEW Real-Time.

Nonlinear VIs

The Nonlinear VIs include the Saturation, Dead Zone, and Rate Limiter VIs. You can use these VIs to simulate the nonlinear effects present in real physical systems. For example, you can use the Friction VI to simulate the effects of friction in a mechanical system. Use these VIs with the Continuous Linear VIs to develop accurate HIL simulations. In addition, you can use VIs such as the Rate Limiter VI in control applications to improve controller performance.

HIL Simulation Applications

One example of an HIL simulation is the dynamics of an automobile suspension system. An automobile suspension system is a continuous mechanical system with second-order dynamics. During the development of an automobile design, if the engineer wants to simulate these dynamics in real time without actually building the physical system, she can use LabVIEW Real-Time and the Advanced Control VIs in this toolset to simulate the dynamics.

One example of a simulated automobile suspension system uses the Integrator VIs from the set of Continuous Linear VIs. In this example, you simulate one-quarter of the vehicle using the spring and shock absorber at one wheel with one-quarter of the vehicle mass. There are two feedback loops in the block diagram, which represent the height of the vehicle and the vertical velocity. This example uses shift registers to implement the feedback loops. The Square Wave PtbyPt function generates a square waveform pattern to simulate the profile of the road. Then the block diagram determines the dynamic response of the vehicle from the suspension system. Figure 10-1 shows the block diagram of this example.

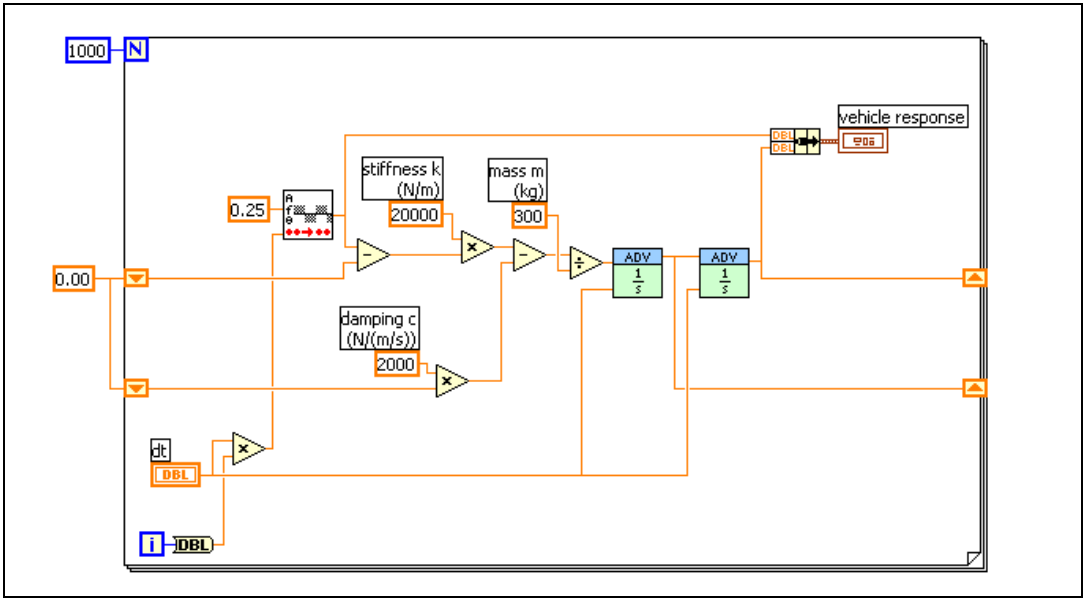


Figure 10-1. Auto Suspension Simulation with Shift Register Feedback Loops

You can also use front panel controls with local variables on the block diagram to represent the feedback loops. Figure 10-2 shows another

representation of the automobile suspension system simulation with local variables. This representation more closely resembles the structure of feedback loops used in control block diagrams.

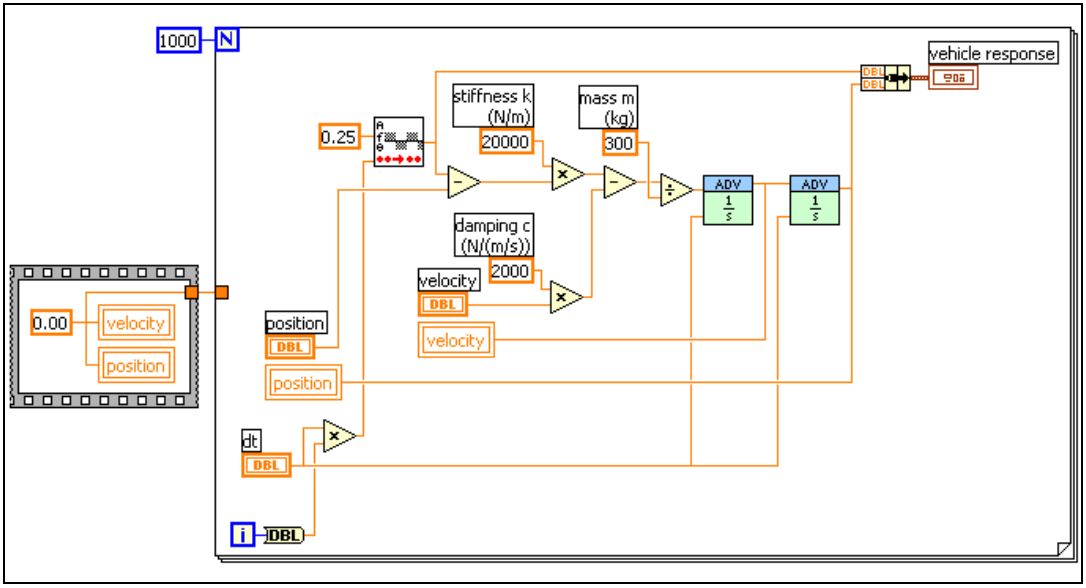


Figure 10-2. Auto Suspension Simulation with Local Variable Feedback Loops

You can also use a transfer function to equivalently represent the dynamics of the automobile suspension system. The example in Figure 10-3 uses the Transfer Function VI to simulate the same automobile suspension system displayed in Figures 10-1 and 10-2. Note that the dynamic system responds exactly the same in this VI as in the two previous examples.

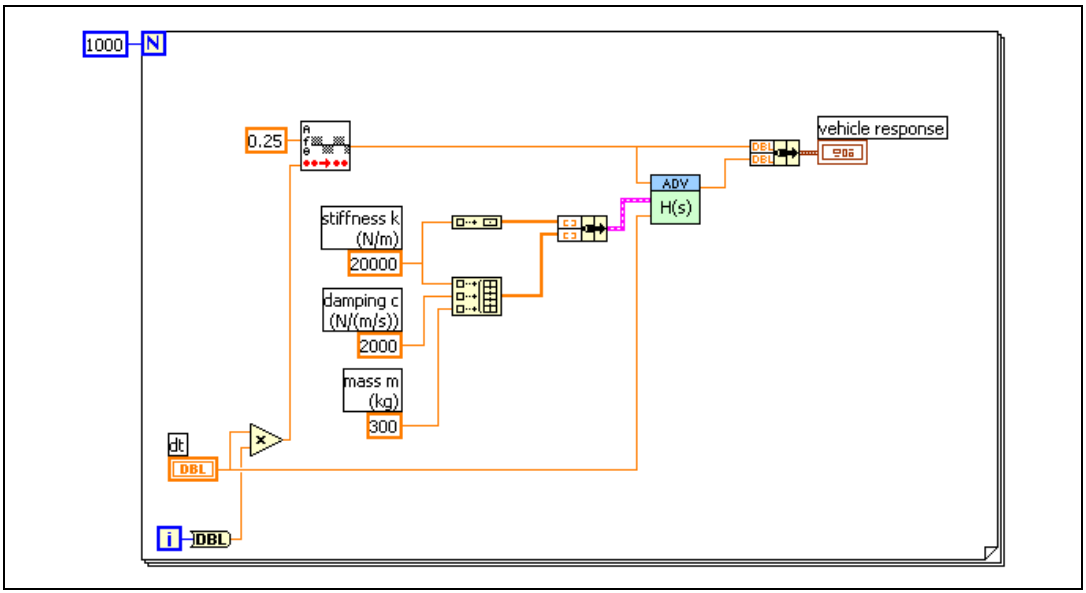


Figure 10-3. Auto Suspension Simulation with Transfer Function

For actual HIL simulation, the model of the simulated system must run in real time and you must physically connect the signals to other systems. To run the simulation in real-time, you must run the VI in LabVIEW Real-Time to provide deterministic real-time response. You can use the input and output VIs in LabVIEW, including DAQ, Controller Area Network (CAN), and so on, to connect the signals to other systems.

In the next example, the automobile suspension simulation is connected to another system. In this case, an actual physical signal provides the input for the road profile of the vehicle. The example uses a DAQ analog input function to read the signals. Then the VI uses a DAQ analog output function to output the response of the vehicle due to the suspension dynamics. Note that for best real-time performance, use other DAQ VIs with hardware-clocked and -synchronized inputs and outputs. Refer to the Advanced Control examples in `[LabVIEW]\examples\control\advanced` for more information about this technique. Figure 10-4 shows the block diagram of the example of HIL simulation.

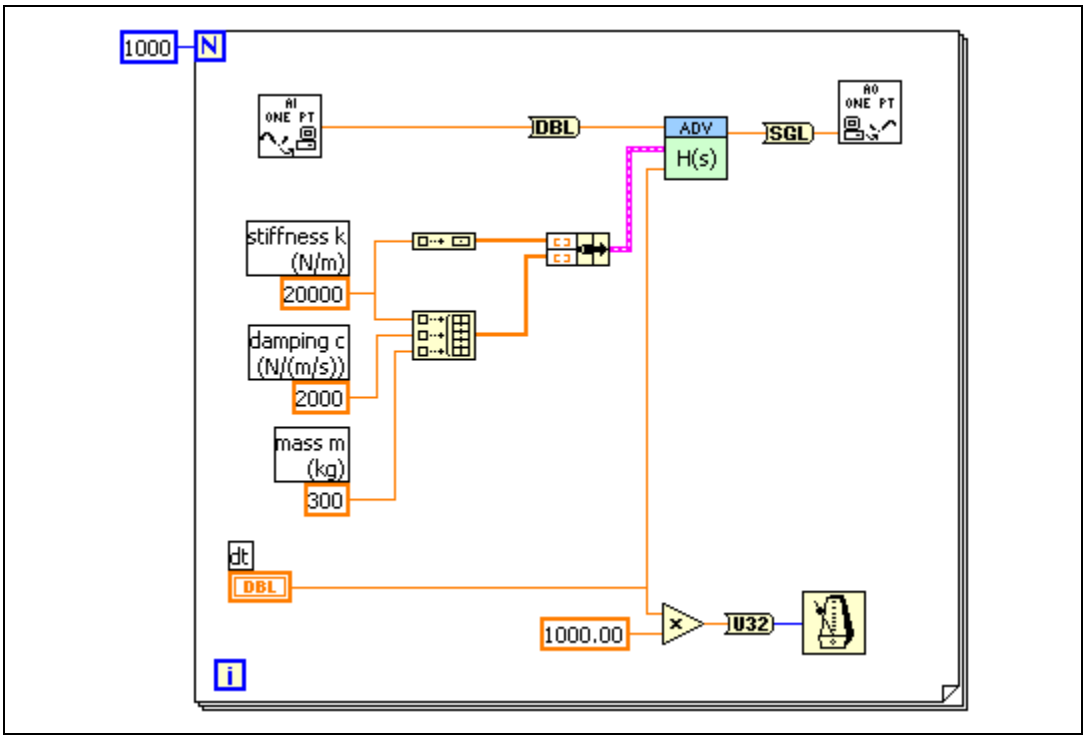


Figure 10-4. Auto Suspension HIL Simulation

Control Applications

In addition to HIL simulation, you can use the Advanced Control functions to simulate analog control systems. Controls systems differ from simulations because in control systems, the controller connects to an external physical system and controls the system parameter(s).

The next example function simulates the behavior of a simple PID controller. This VI uses the Continuos Linear functions to simulate the behavior of a controller implemented with an analog circuit. Note that the Integrator and Derivative functions represent the integral and derivative portions of the controller, respectively. The Multiply function represents the proportional part of the controller. The DAQ analog input and output functions input the process variable from the controlled system and output the controller output signal. Figure 10-5 shows the block diagram of analog PID application.

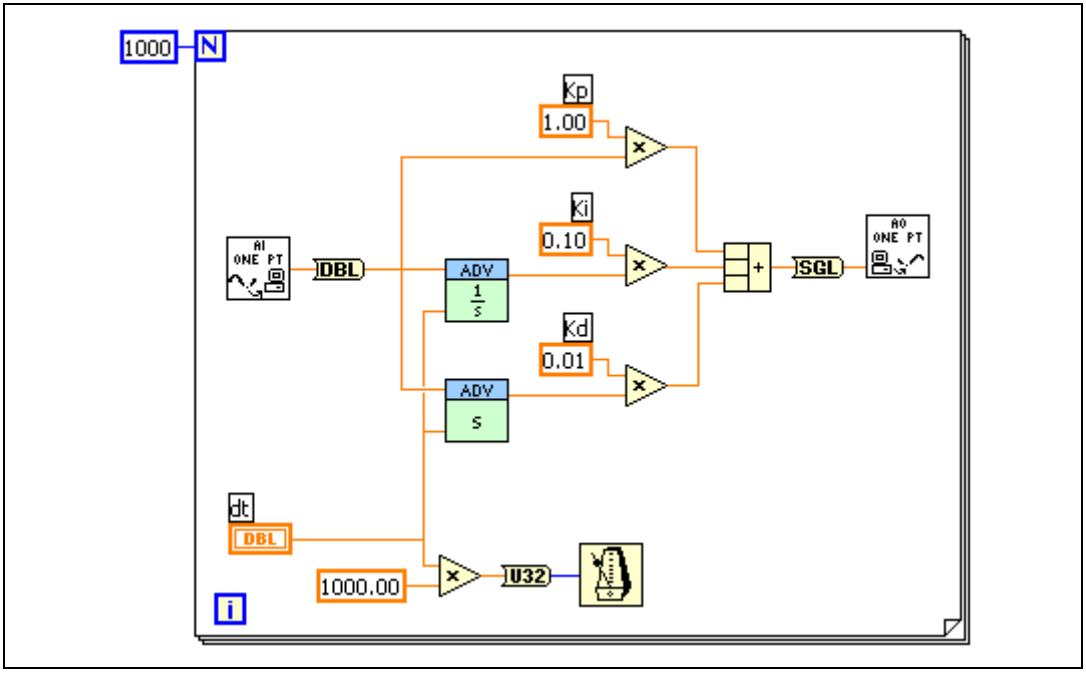


Figure 10-5. Simple Analog PID Application with Continuous Linear Functions

You can use the Advanced Control functions to implement control algorithms that are more complex than a simple PID controller. A PID controller is an example of a single-input-single-output (SISO) system. Systems that multiple parameters to control simultaneously are multiple-input-multiple-output (MIMO) systems. Often the dynamics of the controlled system cause interaction between the various controlled parameters. In this case, you need control methods beyond the capability of PID VIs.

The State Space Function, one of the Advanced Control functions, can control MIMO systems. The example VI in Figure 10-6 shows how to use the Discrete State Space functions with DAQ functions to implement a discrete MIMO controller.

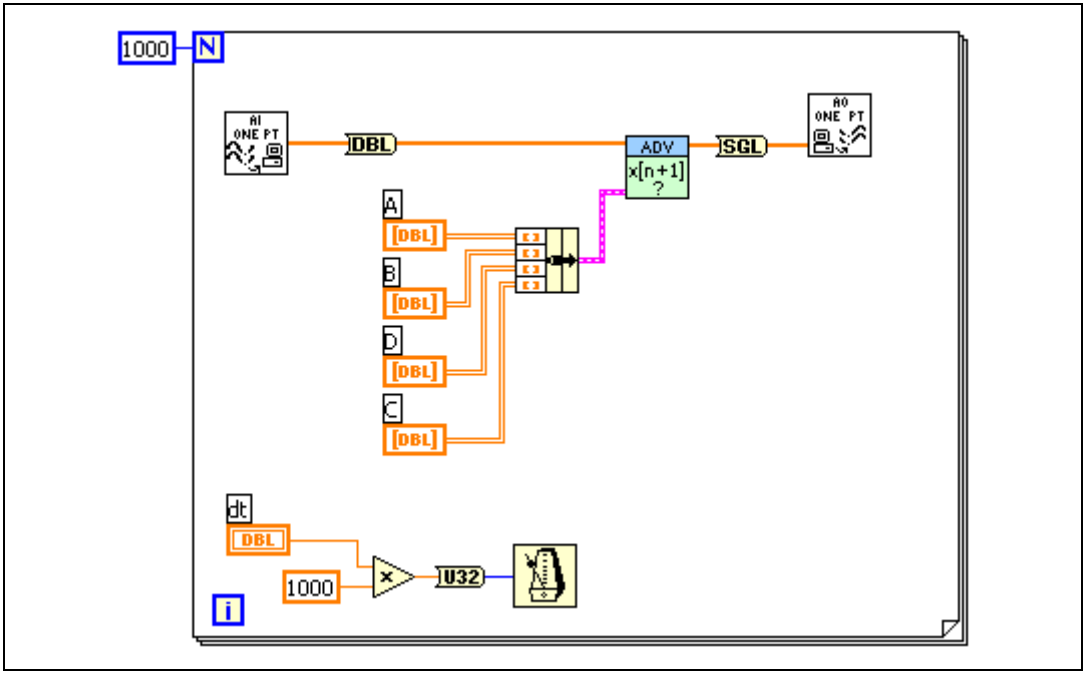


Figure 10-6. Discrete State-Space MIMO Controller

A

References

This appendix lists the reference material used to produce the VIs in this manual. These references contain more information on the theory and algorithms implemented in the fuzzy logic VIs.

The Instrument Society of America (ISA), the organization that sets standards for process control instrumentation in the United States, offers a catalog of books, journals, and training materials to teach you the basics of process control programming. One particular course, *Single and Multiloop Control Strategies*, course number T510, is very helpful. Contact the ISA at its Raleigh, N.C., headquarters at (919) 549-8411.

Corripio (1990) is an ISA Independent Learning Module book. It is organized as a self-study program covering measurement and control techniques, selection of controllers, and advanced control techniques. Tuning procedures are detailed and yet easily understandable. Shinskey (1988) is an outstanding general text covering the application, design, and tuning of all common control strategies. It contains all of the basic algorithms used in the PID control VIs.

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

Astom, K. J., T. Hagglund, C. C. Hang, and W. K. Ho. 1993. Automatic tuning and adaption for PID controllers: a survey. *Control Engineering Practice* 1:669–714.

Corripio, A. B. 1990. *Tuning of industrial control systems*. Raleigh, North Carolina: ISA.

Kahlert, J. and Frank, H. *Fuzzy Logik und Fuzzy Control*. Braunschweig, Wiesbaden: Vieweg, 1993.

Kahlert, J. and Frank, H. *Fuzzy Control fuer Ingeniere*. Braunschweig, Wiesbaden: Vieweg, 1995.

Shinskey, F. G. 1988. *Process control systems*. New York: McGraw-Hill.

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

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

Zimmerman, H.-J. *Fuzzy Set Theory and Its Applications*, Second Revised Edition. Boston, MA: Kluwer Academic Publishers, 1991.

Zimmerman, H.-J. *Fuzzy Sets, Decision Making, and Expert Systems*. Boston, Dordrecht, London: Kluwer Academic Publishers, 1987.

B

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`.

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`.

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

A

aggregation	An operation in fuzzy logic in which several fuzzy sets are combined to produce a single fuzzy set.
algorithm	A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.
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 with 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's 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

C	Celsius.
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 the composite output membership function. 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 maximum values of each output membership function. This method is equivalent to the Center of Area method using singleton sets.
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*.
composition	The process by which a fuzzy controller combines all of the fuzzy subsets assigned to each output variable to form a single fuzzy subset for each output variable.
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.
DC	Direct current.
dead time (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 linguistic output of the rulebase evaluation to a crisp controller output value.
degree of membership	A value that represents the degree of partial membership of an element to a fuzzy set. This value may 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*.
deviation	Any departure from a desired value or expected value or pattern.
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.
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 evaluating crisp controller input values, process parameters, using the defined membership functions to determine linguistic input variables for the rulebase evaluation.
fuzzy inference	The process by which the rules of the rulebase are evaluated to determine output linguistic variables for defuzzification.

fuzzy set	A set that allows for partial membership of elements. Fuzzy sets usually represent linguistic terms and are defined quantitatively by a membership function.
fuzzy set theory	An extension of traditional Boolean set theory, fuzzy set theory is based on the idea that fuzzy sets may be defined such that elements can have partial membership to the set.

G

gain	For a linear system or element, the ratio of the magnitude, 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.

H

Hz	Hertz. Cycles per second.

I

Instrument Society of America (ISA)	The organization that sets standards for process control instrumentation in the United States.
integral action rate	*See* reset rate.
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.

K

K	Process gain.
K_c	Controller gain.
kHz	Kilohertz.

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 to describe a quality of a process variable (for example, hot, very low, small positive, and so on). The term is defined quantitatively by the corresponding membership function.
linguistic variable	Defines the state of a process variable by the degree of membership of the parameter to each linguistic term defined (for example, vehicle position {left 0.0; left center 0.0; center 0.8; right center 0.1; right 0.0}).
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*.
Max-Min inference	Fuzzy inference method using the maximum function for the OR operator and the minimum function for the AND operator. Another common inference method is the Max-Prod, method which uses the product function for the AND operator.
MB	Megabytes of memory. 1 MB is equal to 1,024 KB.
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 function that defines degree of membership to the fuzzy set over a defined universe of discourse of the variable parameter.
ms	Milliseconds.

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's 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.
partial membership	In fuzzy set theory, a condition in which the value of a member partially fulfills the requirements of the membership function of a set.
P controller	A controller which produces proportional control action only; that is, a controller that has only a simple gain response.
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.
proportional kick	The response of a proportional controller to a step change in the setpoint or process variable.
PSI	Pounds per square inch.

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*.
reverse acting (increase-decrease) controller	A controller in which the value of the output signal decreases as the value of the input (measured variable) increases.
RPM	Revolutions per minute.
rule	A linguistic definition of a specific control action of the form IF {condition} AND {condition}... THEN {action}. For example, IF *vehicle position* is right center AND *vehicle orientation* is left up THEN *steering angle* is negative medium.
rule base	A complete set of rules defined for control of a given system. Used during fuzzy inference to determine the linguistic controller output.

S

s	Seconds.
selector control	The use of multiple controllers and/or multiple process variables in which the connections may change dynamically depending on process conditions.
setpoint (SP)	An input variable which sets the desired value of the controlled process variable.
singleton	A normalized membership function with an infinitely small width. A singleton is used to model a crisp value with a fuzzy set.
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.

V

V	Volts.
valve dead band	In process instrumentation, the range through which an input signal may be varied, upon reversal of direction, without initiating an observable change in output signal.

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.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W