LabWindows™/CVI™

PID Control Toolkit User Manual

March 2006
371685B-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

`ni.com`

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 41190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 0 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 1800 226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 02 2377 2222, Thailand 662 278 6777, United Kingdom 44 0 1635 523545

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

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on `ni.com/legal` for more information about National Instruments trademarks.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

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

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

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

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

Contents

About This Manual

Chapter 1
Overview of the PID Control Toolkit

Chapter 2
PID Algorithms

Chapter 3
Using the PID Control Toolkit

About This Manual

The *LabWindows/CVI PID Control Toolkit User Manual* describes the PID Control Toolkit for LabWindows™/CVI™. The manual describes the features, functions, and operation of the toolkit. To use this manual, you need a basic understanding of process control strategies and algorithms.

Conventions

The following conventions appear in this manual:

» — The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

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

bold — Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic — Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

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

Related Documentation

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

- *LabWindows/CVI PID Control Toolkit Help*
- *LabWindows/CVI Help*
- *NI-DAQmx Help*
- *Traditional NI-DAQ (Legacy) C Function Reference Help*

1

Overview of the PID Control Toolkit

This chapter lists the contents of the PID Control Toolkit for LabWindows/CVI, describes how to install the toolkit, and describes Proportional-Integral-Derivative (PID) control applications.

Package Contents

The PID Control Toolkit contains the following materials:

- Software that includes control functions and example programs
- *LabWindows/CVI PID Control Toolkit User Manual*

System Requirements

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

- LabWindows/CVI 7.*x* or later
- Windows 2000/XP

Installation Instructions

If you install the PID Control Toolkit on a computer that has multiple versions of LabWindows/CVI installed, the toolkit is installed with the latest version. If you already have the LabWindows/CVI PID Toolkit 1.*x* installed on your computer, you must uninstall it before installing this version of the PID Control Toolkit.

Complete the following steps to install the PID Control Toolkit:

1. Insert the PID Control Toolkit CD into the CD-ROM drive.
2. Click **Install Toolkit**.
3. Follow the instructions on the screen. When the PID Control Toolkit has been successfully installed, click **Finish**.

PID Control Toolkit Applications

The PID Control Toolkit contains functions you can use to develop LabWindows/CVI control applications. For more information about the types of applications you can develop, refer to the example programs that are installed with the toolkit.

PID Control

Currently, the PID algorithm is the most common control algorithm used in industry. Often, PID is used to control processes that include heating and cooling systems, fluid level monitoring, flow control, and pressure control. When using 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. 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. When applied to the system, the controller output value drives the process variable toward the setpoint value.

You can use the PID Control Toolkit functions with National Instruments hardware to develop LabWindows/CVI control applications. Use I/O hardware, such as DAQ devices, FieldPoint I/O modules, or GPIB boards, to connect your PC to the system you want to control. You can use the LabWindows/CVI I/O functions with the PID Control Toolkit to develop a control application or modify the examples provided with the toolkit.

Using the PID Control Toolkit functions, 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
- Precise PID
- Lead-lag compensation
- Setpoint profile generation
- Multiloop cascade control
- Feedforward control
- Override (minimum/maximum selector) control
- Ratio/bias control

Refer to the *LabWindows/CVI PID Control Toolkit Help*, which you can access by selecting **Start»Programs»National Instruments»PID Control Toolkit for CVI»LabWindows CVI PID Help**, for more information about the functions.

2

PID Algorithms

This chapter explains the fast PID, precise PID, and autotuning algorithms.

The PID Algorithm

The PID controller compares the setpoint (SP) to the process variable (PV) to obtain the error (e), as follows:

$$e = SP - PV$$

Then the PID controller calculates the controller action, $u(t)$, as follows. In this equation, K_c is the controller gain.

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

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.

$$u_I(t) = \frac{K_c}{T_i}\int_0^t edt$$

The following formula represents the derivative action.

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

Implementing the PID Algorithm with the PID Functions

This section describes how the PID Control Toolkit functions implement the fast (positional) PID algorithm. The fast PID algorithm is the default algorithm used in the PID Control Toolkit.

Error Calculation

The following formula represents the current error used in calculating proportional, integral, and derivative action, where PV_f is the filtered process variable.

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

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 the PV or SP. Use nonlinear adjustment of the integral action to counteract overshoot. The following formula represents the trapezoidal integration action.

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

Partial Derivative Action

Because of abrupt changes in the SP, apply derivative action to only 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, as follows:

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

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

All the PID control functions are reentrant. Multiple calls from high-level functions use separate and distinct data.

Note As a general rule, manually drive the PV until it meets or comes close to the SP before you perform the manual-to-automatic transfer.

Gain Scheduling

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

The Precise PID Algorithm

This section describes how the PID Control Toolkit functions implement the precise PID algorithm.

Error Calculation

The current error used in calculating integral action for the precise PID algorithm is shown in the following formula:

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

where SP_{range} is the range of the SP 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 a precise PID algorithm.

The error for calculating proportional action for the precise PID algorithm is shown in the following formula:

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

where β is the setpoint factor for the Two Degree of Freedom PID algorithm described in the *Proportional Action* section. The formula used to calculate derivative action for the precise PID algorithm is the same formula used to calculate derivative action for the fast 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. 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. 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 PV 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 the PV or SP. The following formula represents the trapezoidal integration action for the precise PID algorithm. Use nonlinear adjustment of integral action to counteract the overshoot. The larger the error, the smaller the integral action, as shown in the following formula and in Figure 2-1.

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

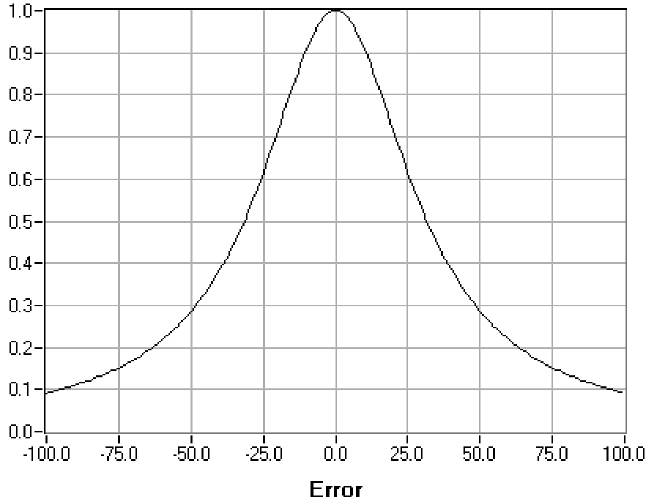

Figure 2-1. Nonlinear Multiple for Integral Action (SP_{rng} = 100)

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 SP. Notice that the PID Library autotuning functions directly implement this process. The existing controller remains in the loop.

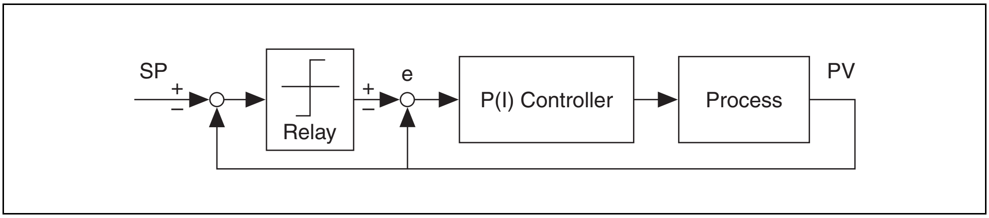

Figure 2-2. Process under PID Control with Setpoint Relay

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 K_u and ultimate period T_u. If the existing model is PI or PID, the autotuning algorithm identifies the dead time τ and time constant T_p, which are two parameters in the integral-plus-deadtime model, as follows:

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

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 types of loop performance: fast (1/4 damping ratio), normal (some overshoot), or 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. It is not necessary 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 Control Toolkit

This chapter contains the basic information you need to design a control strategy using the PID Control Toolkit functions.

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 PID Control Toolkit functions to translate the flowchart into an application.

You can handle the inputs and outputs using DAQ devices, FieldPoint I/O modules, GPIB boards, or serial I/O ports. You can adjust polling rates in real time. Potential polling rates are limited only by your hardware.

Setting Timing

According to control theory, a control system must sample a physical process at a rate that is approximately 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 6 s is sufficient. Faster cycling offers no improvement in performance (Corripio 1990).

The PID control feature, lead-lag feature, and setpoint profile feature in the PID Control Toolkit are time-dependent. A component can acquire the timing information either from a value you supply to the `pidAttrDeltaT` attribute or from the built-in internal timer. By default, the `pidAttrUseInternalTimer` attribute is set to 1, so the component uses the internal timer. Call `PidSetAttribute` and `PidGetAttribute` to set and get PID controller attributes.

The internal timer calculates new timing information each time `PidNextOutput` is called. When the function is called, the timer determines the time since the last call to `PidNextOutput` and uses that time difference in its calculations.

Note The internal timer uses the Windows SDK `GetTickCount` function. Therefore, do not try to run the PID loops faster than 5 or 10 Hz when you use the internal timer.

You can set the component to use the value you have supplied to the `pidAttrDeltaT` attribute by setting `pidAttrUseInternalTimer` to 0. Use the `pidAttrDeltaT` attribute for fast loops, including instances in which you use acquisition hardware to time the controller input.

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 the process control simulation examples 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, 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 for more information about disturbing the loop and determining the response from the graph. Refer to *Tuning of Industrial Control Systems* 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 the SP to disturb the loop after each increment. As you increase K_c, the value of the 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).

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

Table 3-1. Factors for Determining Tuning Parameter Values (Closed Loop)

Controller	PB (Percent)	Reset (Minutes)	Rate (Minutes)
P	2.00 PB_u	—	—
PI	2.22 PB_u	0.83 T_u	—
PID	1.67 PB_u	0.50 T_u	0.125 T_u

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 the 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 3-1. The variables represent the following values:
   - *Td*—Deadtime, in minutes
   - *T*—Time constant, in minutes
   - *K*—Process gain = change in PV/change in output

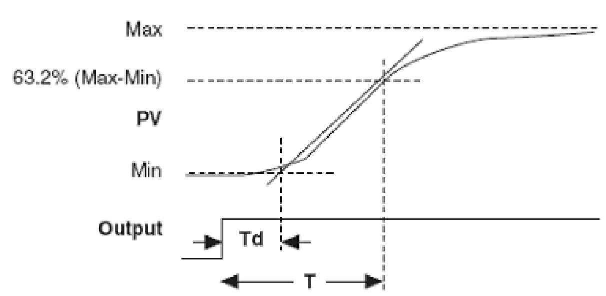

Figure 3-1. Output and Process Variable Strip Chart

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

Table 3-2. Factors for Determining Tuning Parameter Values (Open Loop)

Controller	PB (Percent)	Reset (Minutes)	Rate (Minutes)
P	100 (KT_d/T)	—	—
PI	110 (KT_d/T)	3.33 T_d	—
PID	80 (KT_d/T)	2.00 T_d	0.50 T_d

Using the PID Library

The following sections describe how to use the PID Library to implement a control strategy.

PID Controller

The PID controller requires several inputs, including SP, PID gains, timer interval (in case the internal timer is not used), PV, and output range. PID gains include proportional gain, integral time, and derivative time. The following steps provide an overview of typical PID controller use.

1. Provide the PID gains to `PidCreate` to create a PID controller. `PidCreate` returns a handle that you can use to identify the PID controller in subsequent function calls.
2. Use `PidSetAttribute` to set the PID controller attributes such as SP, time interval, minimum and maximum controller output values, and so on.
3. Provide the PV to the controller in a loop and use `PidNextOutput` to obtain the controller output, which is again applied on the system.
4. Once the control loop ends, call `PidDiscard` to discard the PID controller and free its resources.

You can call `PidSetAttribute` with the `pidAttrOutputMin` and `pidAttrOutputMax` attributes to specify the range of the controller output. 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 so that the controller gain relates engineering units to engineering units instead of percentage to percentage. The PID controller coerces the controller output to the specified range. In addition, the PID controller 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.

PID Algorithms

The PID controller can use the following types of PID algorithms to determine the controller output.

- Fast PID algorithm (`pidFastPidAlgorithm`)
- Precise PID algorithm (`pidPrecisePidAlgorithm`)

Use the `pidAttrAlgorithm` attribute, which you can set using `PidSetAttribute`, to specify the algorithm to use. `pidFastPidAlgorithm` is the default value.

The fast PID algorithm is faster and simpler than the precise PID algorithm. Use the fast algorithm in fast control loops. The precise PID algorithm uses the Two Degree of Freedom algorithm to control the PV, which gives better results than the fast PID algorithm. The precise PID algorithm also uses extra parameters such as Beta, Linearity, and Setpoint Range, which you can specify using `PidSetAttribute`. The precise PID algorithm implements a bumpless manual-to-automatic transfer, which ensures a smooth controller output during the transition from manual to automatic control mode.

Control Input Filter

You can use the filtered PV to filter high-frequency noise from the measured values in a control application. For example, you can use a filtered PV if you are measuring process variable values using a DAQ device. To use a filtered PV, set `pidAttrUseFilteredPV` to 1. By default, this attribute is set to 0. You can use `PidSetProcessVariableFilter` and `PidGetProcessVariableFilter` to set or get custom filters.

As discussed in the *Setting Timing* section, 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 that are 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 filtered PV uses a low-pass fifth-order Finite Impulse Response (FIR) filter to filter out unwanted noise from input signals. The cutoff frequency of the low-pass filter is one-tenth of the sampling frequency, regardless of the actual sampling frequency value.

Output Rate Limiting

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

You can use output rate limiting to avoid the problem of sudden changes in controller output. To enable output rate limiting, set `pidAttrLimitOutputRate` to 1, set `pidAttrOutputRate` and `pidAttrInitialOutput` to limit the rate of change of the controller output, and specify the controller output value on the first iteration of the control loop, respectively. Call `PidSetAttribute` and `PidGetAttribute` to set and get these attributes.

Using PID with Autotuning

You can use autotuning to improve controller performance. There are two ways in which you can autotune a controller.

- **Wizard-Based Autotuning**—You can use the PID Autotuning Wizard to tune the parameters.
- **Classic Autotuning**—You can use the functions in the Autotuning class to develop a custom autotuning user interface.

Complete the following steps to autotune a controller. These steps explain both wizard-based and classic autotuning.

1. Call `PidCreateWithAutotune` to create the controller and obtain the PID handle that identifies that controller in subsequent function calls. The PID Library invokes the callback function provided to `PidCreateWithAutotune` when the following PID events occur:
   - **pidNoiseEstimateEvent**—Noise estimation is complete
   - **pidRelayCycleEvent**—A setpoint relay cycle is complete
   - **pidAutotuneEvent**—Autotuning is complete

   When you use wizard-based autotuning, the library invokes the callback function only when the autotuning is complete.
2. Provide the PV to the controller in a loop and obtain the controller output, which is again applied on the system.
3. While the PID control loop is being run, call `PidAutotuneShowDialog` if you want to use wizard-based autotuning. This function launches the Autotuning Wizard. To use classic autotuning, call the functions in the Autotuning class.
4. Once the control loop ends, call `PidDiscard` to discard the PID controller and release its resources.

Distributing Applications That Use Wizard-Based Autotuning

Use the LabWindows/CVI application distribution feature to deploy applications you create using the PID Control Toolkit. The PID Control Toolkit installs `CVIPIDRuntime.msm` in the `C:\Program Files\Common Files\Merge Modules` directory. This file installs

CVIPIDAtUI.dll in the system directory. If you deploy applications that use wizard-based autotuning, you must add this merge module to the distribution.

The version of LabWindows/CVI you are using determines how you add the merge module to the distribution. If you are using LabWindows/CVI 7.*x*, create a file group in the distribution kit named _MSMS_. Include CVIPIDRuntime.msm in that file group. Build the distribution kit, and CVIPIDRuntime.msm will be seamlessly merged in as an MSI merge module, instead of just being included as a file.

If you are using LabWindows/CVI 8.0 and later, click **Add Additional Module** in the **Drivers & Components** tab of the Edit Installer dialog box. In the Select Merge Module dialog box, browse to and select CVIPIDRuntime.msm. For additional information about distributing LabWindows/CVI applications, refer to the *LabWindows/CVI Help*.

Using PID with Gain Scheduling

Most processes are non-linear. Therefore, PID parameters that produce a desired response at one operating point might not produce a satisfactory response at another operating point. Using the gain scheduling feature, you can apply different sets of PID parameters for different regions of controller operation.

The gain scheduler selects and outputs one set of PID gains from a gain schedule based on the current gain scheduling value. The gain scheduling value input can be anything and is based on the gain scheduling criteria that you set.

Use the pidGSAttrGainScheduleCriteria attribute to set the gain scheduling criteria. Call PidSetGainScheduleAttribute and PidGetGainScheduleAttribute to set and get gain scheduling attributes. The pidGSAttrGainScheduleCriteria attribute can take the following values:

- Setpoint
- Process variable
- Controller output
- Gain schedule variable provided by the user through the pidGSAttrUserGainScheduleVariable attribute

The gain schedule is a list of gain sets. A gain set consists of the following features:

- Proportional gain (K_c)
- Integral time (T_i)
- Derivative time (T_d)
- Gain control value

The PID Library uses the gain set that has the smallest control value that is greater than the value of the signal specified by the gain schedule criteria. For example, if three gain sets have control values equal to 10, 20, and 30 and the value of the signal specified by the gain schedule criteria is 15, then the second gain set is used.

If the value of the signal specified by the gain schedule criteria is greater than the gain schedule's largest control value, then the gain set with the largest control value is used. You also can set `pidGSAttrSelectionMode` to `pidManual` to allow you to set the gain sets manually. By default, the mode is automatic.

Using PID with Lead-Lag

The lead-lag compensator uses a positional algorithm that approximates a true exponential lead-lag. Feed forward control schemes often use this kind of algorithm as a dynamic compensator. Using lead-lag, you can simulate inertia of motors, slow settling times in pipes, and so on. Lead compensation stabilizes a closed loop by reacting to how fast something is changing rather than its current state. This process speeds up the reaction. Lag compensation stabilizes a closed loop by slowing down the reaction to the present value so that the correction is made more slowly and does not overshoot. This process slows down the reaction.

The typical usage of the lead-lag follows:

1. Pass gains to `PidLeadLagCreate` to create a lead-lag compensator. `PidLeadLagCreate` returns a handle you can use to identify the lead-lag compensator in subsequent function calls.
2. Use `PidSetLeadLagAttribute` and `PidGetLeadLagAttribute` to set and get the lead-lag compensator attributes such as time intervals and minimum/maximum output values. Provide the input to the compensator in a loop and use `PidLeadLagNextOutput` to obtain the output. This output is either applied to the system or to the controller, based on whether the lead-lag is being used as an input or an output filter.
3. Once the control loop ends, call `PidLeadLagDiscard` to discard the lead-lag compensator and release its resources. Also call `PidDiscard` to discard the PID controller.

The lead-lag compensator can be used either as an input or an output filter to the PID controller. The default output range is –100 to 100, which corresponds to values specified in terms of percentage of full scale. However, you can change this range so that the controller gain relates engineering units to engineering units instead of percentage to percentage. The lead-lag compensator coerces the controller output to the specified range.

Using PID with Setpoint Profiling

Using the setpoint profiling feature, you 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. Use this feature to implement any arbitrary combination of ramp, hold, and step functions. Provide (setpoint, time) pairs to the setpoint profile. The setpoint profile maintains the setpoint specified in each pair for the corresponding times specified in the time array.

You can use the setpoint profiler as follows:

1. Call `PidSetpointProfileCreate` to create a setpoint profile. Use a pair of time and setpoint value arrays to specify the setpoint profile with the time values in ascending order.
2. Use `PidSetSetpointProfileAttribute` to set the setpoint profile attributes.
3. Use `PidSetpointProfileNextSetpoint` to obtain the setpoint from the profile in a loop and provide this setpoint to the controller.
4. Once the control loop ends, call `PidSetpointProfileDiscard` to discard the setpoint profile and release its resources. Also call `PidDiscard` to discard the PID controller.

At any time in the control loop, you can use `pidSetpointProfileAttrElapsedTime` to get the elapsed time. You also can check if the profile is complete using the `pidSetpointProfileAttrProfileComplete` attribute.

Using Ramp Generators

A ramp generator is a simple component that you can use to generate a ramp output. Typically, you use a ramp generator as follows:

1. Call `PidRampCreate` to create a ramp generator. Specify the SP, initial output, and the rate at which the output of the ramp changes.
2. Call `PidSetRampAttribute` to set the ramp generator attributes.
3. Use `PidRampNextOutput` to obtain the output of the ramp in a loop.
4. Call `PidRampDiscard` to discard the ramp generator and release its resources.

Converting between Percentage of Full Scale and Engineering Units

As described in the previous sections, the default SP, PV, and output ranges for the PID Library functions correspond to a percentage of the full scale. 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 such as volts for analog output. You can use `PidConvertEGUToPercentage` to convert any input from real engineering units to percentage of full scale and `PidConvertPercentageToEGU` to convert the controller output from percentage to real engineering units. `PidConvertPercentageToEGU` has an additional input parameter, **bCoerce**. The default value of **bCoerce** is TRUE, which indicates that the output is coerced to the range.

Note The PID Library functions 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 the SP and PV of 10, regardless of the output range and setpoint range.

Using PID on Real-Time (RT) Targets

Some PID applications are deterministic and, therefore, cannot be run on desktop operating systems. Because the PID Library is supported on real-time (RT) targets, you can use it to develop deterministic applications. For more information about developing and running RT applications, refer to the *LabWindows/CVI Help*.

Because RT systems do not support user interfaces, you cannot use wizard-based autotuning in PID applications that are targeted for RT platforms. Any applications that are targeted for RT must not include the following functions:

- `PidAutotuneShowDialog`
- `PidAutotuneCloseDialog`

However, you can use other functions in the Autotuning class to tune the PID controller in applications targeted for RT platforms.

Using PID with DAQ Devices

This section addresses several important issues you might encounter when you use the DAQ APIs to control actual processes.

Complete the following steps to use the PID Library with a DAQ device.

1. Configure the DAQ device and channels for both input and output. Also configure the sample clocks, if necessary.
2. Call `PidCreate` to create the PID controller. Then call `PidSetAttribute` to configure the controller attributes.
3. Within the control loop, complete the following steps:
   a. Read the input from the DAQ device.
   b. Modify/manipulate the input so that it can be provided to the controller. This step is optional and required only in cases in which the controller input is derived from the acquired DAQ input.
   c. Supply this input to the controller and obtain the controller output.
   d. Modify/manipulate the controller output so that it can be used as the DAQ device output. This step is optional and required only in cases in which the output on the DAQ device is derived from the controller output.
   e. Write the output to the DAQ device.

4. Discard the controller to release its resources. Also clear all the DAQ tasks. In some cases, just before clearing the DAQ tasks, the DAQ device outputs a 0.

The control loops can be timed in the following ways:

- **Software-Timed**—In software-timed control loops, the timing is controlled by the software loop rate. You can implement a software loop with a timer construct, such as a timer control or asynchronous timer, or a while loop with a delay or sleep operation at the end.
- **Hardware-Timed**—In hardware-timed control loops, the timing is controlled by the DAQ device. The DAQ device is configured with the appropriate sample rate, and the sample mode is set to hardware-timed.

For more information about DAQ, refer to the *NI-DAQmx Help* or *Traditional NI-DAQ (Legacy) C Function Reference Help*, depending on the DAQ API you are using. Also refer to the DAQ example programs.

A

References

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.

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. This book provides detailed tuning procedures.

The following material is referenced in this manual:

Corripio, A. B. 1990. *Tuning of Industrial Control Systems*. Raleigh, North Carolina: ISA.

Ziegler, J. G. and N. B. Nichols. 1942. "Optimum Settings for Automatic Controllers." *Trans. ASME* 64:759–68.

B

Technical Support and Professional Services

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

- **Support**—Online technical support resources at `ni.com/support` include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at `ni.com/exchange`. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit `ni.com/services` or contact your local office at `ni.com/contact`.
- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

A

algorithm	A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.
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 Autotuning functions. The Autotuning Wizard gathers some information about the desired control from the user and then steps through the PID autotuning process. You must specify to use wizard-based autotuning in `PidCreateWithAutotune` to use this feature.

B

bias	The offset added to a controller's output.
bumpless transfer	A process in which the next output always increments from the current output, regardless of the current controller output value. Therefore, transfer from automatic to manual control is always bumpless.

C

cascade control	Control in which the output of one controller is the setpoint for another controller.
closed loop	A signal path that includes a forward path, a feedback path, and a summing point and that forms a closed circuit. Also called a *feedback loop*.
controller	Hardware and/or software used to maintain parameters of a physical process at desired values.
controller output	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.
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.

derivative (control) action — Control response to the time rate of change of a variable.

E

EGU — Engineering units.

F

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.

G

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

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 (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.

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 0 indicates a squared nonlinear operation.
load disturbance	The ability of a controller to compensate for changes in physical parameters of a controlled process while the setpoint value remains constant.

M

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 traveling beyond a desired maximum range.
overshoot	The maximum excursion beyond the final steady-state value of output as the result of an input change.

P

P	Proportional.
PD	Proportional, derivative.
PI	Proportional, integral.
PID	Proportional, integral, derivative.
PID control	A common control strategy in which a process variable is measured and compared to a desired setpoint 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 = $100/K_c$.
PSI	Pounds per square inch.

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 input stimulus.

reentrant execution — Mode in which calls to multiple instances of a function can execute in parallel with distinct and separate data storage.

S

s — Seconds.

setpoint (SP) — An input variable that sets the desired value of the controlled process variable.

span — The algebraic difference between the upper and lower range values.

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.

trapezoidal integration — A numerical 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.

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

K

L

N

O

P

R

S

T

U

W

Z