

BridgeVIEW™ and LabVIEW™

PID Control Toolkit for G Reference Manual

Internet Support

E-mail: support@natinst.com

FTP Site: <ftp.natinst.com>

Web Address: <http://www.natinst.com>

Bulletin Board Support

BBS United States: 512 794 5422

BBS United Kingdom: 01635 551422

BBS France: 01 48 65 15 59

Fax-on-Demand Support

512 418 1111

Telephone Support (USA)

Tel: 512 795 8248

Fax: 512 794 5678

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

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 manual 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

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

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Organization of This Manual	ix
Conventions Used in This Manual.....	x
Related Documentation.....	x
Customer Communication	xi

Chapter 1

Overview of the PID Control Toolkit

Package Contents	1-1
Installation Procedure	1-1
Windows 95/NT	1-1
Windows 3.x	1-2
Macintosh and Power Macintosh	1-2
PID Control Toolkit Applications.....	1-2

Chapter 2

PID Algorithms

The PID Algorithm	2-1
Gain Scheduling	2-5
The Autotuning Algorithm	2-5
Tuning Formulas	2-6

Chapter 3

Using the PID Software

Designing a Control Strategy.....	3-1
Setting Timing	3-2
Manual Tuning Techniques.....	3-4
Closed-Loop (Ultimate Gain) Tuning Procedure.....	3-4
Open-Loop (Step Test) Tuning Procedure.....	3-5
Using the PID VIs.....	3-7
The PID VI	3-7
The PID (Gain Schedule) VI	3-8
The PID (Compatibility) VI	3-9
The PID with Autotuning VI and the Autotuning Wizard	3-10
Using PID with Data Acquisition Hardware (DAQ)	3-13
Software-Timed DAQ Control Loop.....	3-13
Software-Timed DAQ Control Loop Using Advanced DAQ Functions	3-15

Hardware-Timed DAQ Control Loop.....	3-16
Event-Driven DAQ Control Loop (BridgeVIEW).....	3-17
Using a VI Server for DAQ Control	3-18

Chapter 4

PID Software VIs

PID.....	4-1
PID (Gain Scheduling)	4-3
PID (Compatibility).....	4-6
PID with Autotuning	4-7
Lead-Lag.....	4-11
Ramp.....	4-12

Chapter 5

Process Control Examples

Simulation VIs.....	5-1
Tank Level	5-1
General PID.....	5-2
Plant Simulator.....	5-4
Cascade and Selector	5-5
Demonstration VIs.....	5-8
PID with MIO Board.....	5-8
Lead-Lag	5-10

Appendix A

References

Appendix B

Customer Communication

Glossary

Index

Figures

Figure 2-1.	Nonlinear Multiple for Integral Action ($SP_{rng} = 100$)	2-3
Figure 2-2.	Process under PID Control with Setpoint Relay	2-5
Figure 3-1.	Control Flowchart and Block Diagram	3-1
Figure 3-2.	Cascaded Control Functions.....	3-3
Figure 3-3.	Output and Process Variable Strip Chart	3-6
Figure 3-4.	Gain Scheduling Input Example.....	3-9
Figure 3-5.	Updating PID Parameters Using a Shift Register	3-11
Figure 3-6.	Updating PID Parameters Using a Local Variable.....	3-11
Figure 3-7.	Storing PID Parameters in a Datalog File	3-12
Figure 3-8.	Software-Timed DAQ Control Loop	3-13
Figure 3-9.	Software-Timed DAQ Control Loop with Advanced Features.....	3-15
Figure 3-10.	Hardware-Timed DAQ Control Loop	3-16
Figure 3-11.	Event-Driven DAQ Control Loop (BridgeVIEW)	3-17
Figure 3-12.	VI Server Example for DAQ Control.....	3-18
Figure 5-1.	Front Panel of the Tank Level VI.....	5-1
Figure 5-2.	Block Diagram of the Tank Level VI.....	5-2
Figure 5-3.	Front Panel of the General PID VI.....	5-3
Figure 5-4.	Block Diagram of a Pressure Control PID VI.....	5-3
Figure 5-5.	Front Panel of the Plant Simulator VI.....	5-4
Figure 5-6.	Block Diagram of the Plant Simulator VI.....	5-5
Figure 5-7.	Front Panel of the Cascade and Selector VI.....	5-6
Figure 5-8.	Block Diagram of the Cascade and Selector VI.....	5-7
Figure 5-9.	Front Panel of the PID VI with Controls Set for an MIO Data Acquisition Board.....	5-8
Figure 5-10.	Block Diagram of the PID VI with Controls Set for an MIO-16 Data Acquisition Board	5-9
Figure 5-11.	Front Panel of the Lead-Lag Example VI	5-10

Tables

Table 2-1.	Tuning Formula under P-only Control (fast)	2-6
Table 2-2.	Tuning Formula under P-only Control (normal).....	2-6
Table 2-3.	Tuning Formula under P-only Control (slow).....	2-7
Table 2-4.	Tuning Formula under PI Control (fast).....	2-7
Table 2-5.	Tuning Formula under PI Control (normal).....	2-7
Table 2-6.	Tuning Formula under PI Control (slow).....	2-8
Table 3-1.	Closed-Loop–Quarter-Decay Ratio Values.....	3-5
Table 3-2.	Open-Loop–Quarter-Decay Ratio Values	3-6
Table 3-3.	PID Parameter Ranges.....	3-9

About This Manual

The *PID Control Toolkit for G Reference Manual* describes the PID Control VIs. You should have a basic understanding of process control strategies and algorithms. If you are not familiar with process control terminology, methods, and standards, see Appendix A, [References](#), for other sources of information.


Organization of This Manual

The *PID Control Toolkit for G Reference Manual* is organized as follows:

- Chapter 1, [Overview of the PID Control Toolkit](#), lists the contents of the PID Control Toolkit and describes the installation procedure and PID Control applications.
- Chapter 2, [PID Algorithms](#), explains two algorithms: PID and Autotuning.
- Chapter 3, [Using the PID Software](#), contains the basic information you need to begin using the PID Control VIs.
- Chapter 4, [PID Software VIs](#), contains descriptions of the six PID VIs. Two VIs remain from the previous toolkit: Lead-Lag and Ramp. The four new VIs include PID, PID (gain schedule), PID (compatibility), and PID with autotuning.
- Chapter 5, [Process Control Examples](#), describes examples that use the PID Control VIs. You do not need any data acquisition plug-in boards to run the Simulation VIs, but you must have the appropriate hardware to run the Demonstration VIs.
- Appendix A, [References](#), lists sources of information you can consult if you are not familiar with process control terminology, methods, and standards.
- Appendix B, [Customer Communication](#), contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The [Glossary](#) contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

Conventions Used in This Manual

The following conventions are used in this manual:

bold	Bold text denotes the names of menus, menu items, parameters, dialog box buttons or options, icons, Windows 95 tabs, or LEDs.
<i>italic</i>	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept.
<i>bold italic</i>	Bold italic text denotes a note.
monospace	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, filenames, and extensions, and for statements and comments taken from program code.
	This icon to the left of bold italicized text denotes a note, which alerts you to important information.

Related Documentation

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

- *LabVIEW User Manual*
- *BridgeVIEW User Manual*
- *G Programming Reference Manual*
- *LabVIEW QuickStart Guide*
- *LabVIEW Function and VI Reference Manual*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix B, *Customer Communication*, at the end of this manual.

Overview of the PID Control Toolkit

This chapter lists the contents of the PID Control Toolkit and describes the installation procedure and PID Control applications.

Package Contents

The PID Control Toolkit contains the following materials:

- *PID Control Toolkit for G Reference Manual*
- Process control and example VIs distribution software containing the following VI libraries:
 - `prctrlex.llb`—A library containing examples that demonstrate the use of the PID Control Toolkit. The examples are divided into two types: Simulation VIs, which do not require any data acquisition hardware, and Demonstration VIs, which require a National Instruments plug-in data acquisition board.
 - `prctr1.llb`—A library containing VIs that make up the core of the PID Control Toolkit. The example programs, along with future PID VIs you might build, call these subVIs.
 - `autopid.llb`—A library containing VIs used for automatic tuning. These VIs use the `prctr1.llb` library functions.

Installation Procedure

The following sections contain instructions for installing the PID Control Toolkit on the Windows 95/NT, Windows 3.x, and Power Macintosh platforms.

Windows 95/NT

Complete the following steps to install the PID Control Toolkit.

1. Launch Windows 95 or NT.
2. Insert disk 1 of the PID Control Toolkit into the 3.5-inch disk drive.

3. From the **Start** menu, choose **Run** and type `A:\setup.exe`.
4. Follow the instructions on your screen.

Once you have completed the on-screen installation instructions, you are ready to run the PID Control Toolkit.

Windows 3.x

Complete the following steps to install the PID Control Toolkit.

1. Launch Windows.
2. Insert disk 1 of the PID Control Toolkit into the 3.5-inch disk drive.
3. From the File Manager, run `SETUP.EXE`.
4. Follow the instructions on your screen.

Once you have completed the on-screen installation instructions, you are ready to run the PID Control Toolkit.

Macintosh and Power Macintosh

Complete the following steps to install the PID Control Toolkit.

1. Insert disk 1 of the PID Control Toolkit into the 3.5-inch disk drive and double-click on the **PID Control Installer** icon.
2. Follow the instructions on your screen.

Once you have completed the on-screen installation instructions, you are ready to run the PID Control Toolkit.

PID Control Toolkit Applications

The PID Control Toolkit contains functions you can use to develop LabVIEW and BridgeVIEW control applications. Currently, PID (Proportional-Integral-Derivative) is the most common control algorithm used in industry. Often, PID is used to control processes such as heating and cooling systems, fluid level monitoring, flow control, and pressure control. The system parameter being controlled is referred to as the *process variable* (for example, temperature, pressure, or flow rate). The operator must specify a *setpoint* or desired value for the process variable that is to be controlled. A PID controller determines a *controller output* value (for example, heater power or valve position). The controller output value is applied to the system which in turn affects the process variable and drives it toward the setpoint value.

You can use the PID Control Toolkit VIs with National Instruments hardware to develop LabVIEW and BridgeVIEW control applications. Use I/O hardware, such as DAQ (data acquisition), FieldPoint, and GPIB, to interface your PC with the system you want to control. You can use the I/O VIs provided in BridgeVIEW and LabVIEW with the PID Control Toolkit to develop a control application or modify the examples provided with the toolkit.

Using the VIs described in this manual, you can develop control applications based on proportional-integral-derivative (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 ramp generation
- Multiloop cascade control
- Feedforward control
- Override (minimum/maximum selector) control
- Ratio/bias control

By combining these PID Control VIs with LabVIEW math and logic functions, you can assemble block diagrams that execute real control strategies. The PID Control VIs implement the algorithms using LabVIEW functions and library subVIs without any Code Interface Nodes (CINs). You can modify the VIs for your applications in LabVIEW without writing any conventional code.

Documentation for each VI is included in its **Get Info...** box. In addition, **Description** boxes document each control and indicator. Use the LabVIEW Help window to see control and indicator descriptions.

PID Algorithms

This chapter explains two algorithms: PID and Autotuning.

The PID Algorithm

In the PID (Proportional-Integral-Derivative) controller, the setpoint is compared to the process variable to obtain the error

$$e = SP - PV.$$

You can then calculate the controller action theoretically as

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

where K_c is controller gain. If the error and the controller output have the same range, that is -100% to 100% , controller gain is the reciprocal of *proportional band*. T_i is the integral time in minutes (also called *reset time*), and T_d is the derivative time in minutes (also called *rate*). The proportional action is

$$u_p(t) = K_c e,$$

the integral action is

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

and the derivative action is

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

The PID VIs implement the positional PID algorithm as described in the following sections. The subVIs used in these VIs are labelled such that you can modify any of these features as necessary.

PV filtering—Process variable filtering minimizes the effects of noise.

$$PV_f = 0.5PV + 0.25PV(k-1) + 0.175PV(k-2) + 0.075PV(k-3)$$

Error calculation—The current error used in calculating *integral action* and *derivative action* is

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

The error for calculating *proportional action* is

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

where SP_{rng} is the range of the setpoint, β (beta) is the setpoint factor (for the *Two Degree of Freedom PID* algorithm described under *Proportional Action*), 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 . This use of a nonlinear gain term is referred to as an Error-squared PID algorithm.

Proportional Action—In applications, setpoint changes are normally greater and more rapid than load disturbances, while load disturbances appear as a slow departure of the controlled variable from the setpoint. PID tuning for good load-disturbance responses often results in setpoint responses of unacceptable oscillation. On the other hand, tuning for good setpoint responses often yields sluggish load-disturbance responses. The factor β , when set to less than 1, reduces the setpoint-response overshoot without affecting the load-disturbance response. This is referred to as a *Two Degree of Freedom PID* algorithm. Intuitively, β is an index of the setpoint response importance, from 0 to 1. For example, if you consider load response the most important loop performance, set β to 0. Conversely, if you want the process variable to follow the setpoint change quickly, set β to 1.

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

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

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

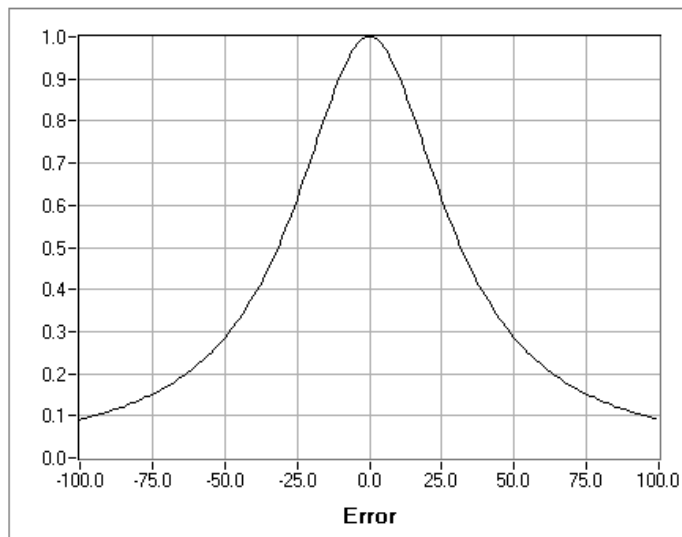


Figure 2-1. Nonlinear Multiple for Integral Action ($SP_{rng} = 100$)

Partial Derivative Action—Because of abrupt changes in setpoint (SP), apply only derivative action to a *filtered* PV (not the error e) to avoid *derivative kick*.

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

Controller Output—Controller output is the summation of the proportional, integral, and derivative action.

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

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 practical model of the PID controller is

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

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

The default ranges for the parameters **setpoint**, **process variable**, and **output** correspond to percentage values; however, you can use actual engineering units. Adjust corresponding ranges accordingly. **Reverse acting** (also called increase-decrease) is the *normal* controller mode in which the output decreases if the **process variable** is greater than the **setpoint**. The VIs measure T_i and T_d in minutes. Switching to hold mode or manual mode freezes the output at the current value. In the manual model, you can increase or decrease the output by changing the manual input. All transfers, from automatic to manual or automatic to hold, and from manual to automatic or hold to automatic, are bumpless.

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 describes a system where controller parameters are changed depending on measured operating conditions. For instance, the scheduling variable can be the setpoint, the process variable, a controller output, or an external signal. For historical reasons, the word *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.

In this toolkit, you can define unlimited sets of PID parameters for gain scheduling. For each schedule you can run autotuning to update the PID parameters.

The Autotuning Algorithm

Autotuning is used to improve performance. Often, many controllers are poorly tuned—some too aggressive, some too sluggish. When you are not sure about disturbance or process dynamic characteristics, tuning a PID controller is difficult; therefore, the need for autotuning arises.

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 this process is implemented directly with the PID autotuning VI. The existing controller remains in the loop.



Note

Although it might be poorly tuned, a stable controller must be established.

02Gfig03.bmp

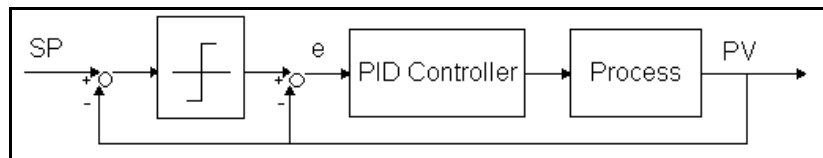


Figure 2-2. Process under PID Control with Setpoint Relay

For most systems, a limiting cycle generates because of the nonlinear relay characteristic. From this cycle, we identify the relevant information needed for PID tuning:

- If the existing controller is proportional only, ultimate gain K_u and ultimate period T_u .
- If the existing model is PI or PID, dead time τ and time constant T_p 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 autotuning VIs, select one of 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 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 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 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.

Using the PID Software

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

Designing a Control Strategy

When designing a control strategy, sketch a process flowchart showing control elements (for example, valves) and measurements. Add feedback and any required computations. Then translate the flowchart into a block diagram using the Control VIs in this toolkit and the math and logic functions in LabVIEW or BridgeVIEW. Figure 3-1 is an example of a control block diagram. The only elements missing from this simplified VI are the loop-tuning parameters and the automatic to manual switching.

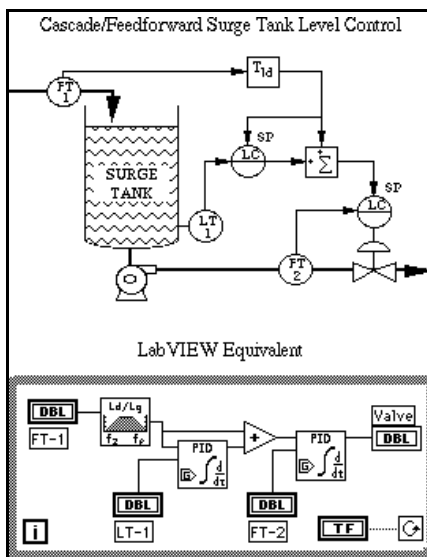


Figure 3-1. Control Flowchart and Block Diagram

You can handle the inputs and outputs through data acquisition (DAQ) boards, GPIB instruments, or serial I/O ports and adjust polling rates in real time. These polling rates are limited only by your hardware, the number of VIs, and the graphic complexity of your VIs.

Setting Timing

The PID and Lead-Lag VIs in this toolkit are time dependent. A VI can acquire timing information through a value you supply in the **cycle time** control or through a built-in time keeper. If **cycle time** is less than or equal to zero (the default), the VI calculates new timing information each time it is called. 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 subVIs, you can achieve fairly regular timing, and the internal time keeper compensates for variations. However, tick timer resolution is limited to 1 ms on Windows 95 and NT, Macintosh, and Power Macintosh and to 55 ms on Windows 3.1. Because of this limitation, you should not try to run the PID VIs faster than 5 or 10 Hz when **cycle time** is less than or equal to zero.



Note

You can improve resolution under Windows 3.1 to 1 ms by deselecting Use default timer in the Performance and Disk preferences. See the `LVRREADME.WRI` file for information about increasing your timer resolution.

If **cycle time** is a positive value (in seconds), the VI uses **cycle time** in the calculations, regardless of the elapsed time. This method should be used for fast loops, such as when controller input is timed with acquisition hardware. For an example of using hardware timing with the combined PID and DAQ VIs, see Demo-HW Timed PID VI in the example library `prctllex.llb`. In this example, the analog input is sampled at precisely timed intervals and the **actual scan rate** parameter from the AI Start VI is inverted and wired into the **cycle time** input.

According to control theory, a sampled control system must run about 10 times faster than the fastest time constant in the plant under control. For instance, a temperature control loop is probably quite slow—a time constant of 60 s is common 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 or BridgeVIEW application.

In some situations, you might want to run several Control VIs at different cycle times, yet share data between them, perhaps in a cascade. Because all VIs within a loop execute once per iteration (that is, at the same **cycle time**), you must separate the VIs into independently timed While Loops, as shown in Figure 3-2. A global variable passes the output of Loop A to the **process variable** (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 data that the two While Loops might share.

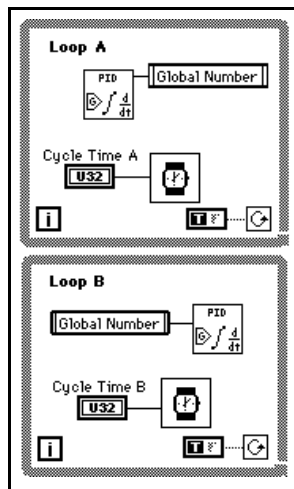


Figure 3-2. Cascaded Control Functions

If the front panel does not contain graphics that LabVIEW or BridgeVIEW 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.

Manual Tuning Techniques

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 on your process or with one of the process control Simulation VIs to compare them. For different processes, one method might be easier or more accurate than the other. Some techniques you can use with online controllers cannot stand the gross upsets described here. Consult the literature listed in Appendix A, *References*, for more information.

To perform these tests with LabVIEW or BridgeVIEW, set up your control strategy with the **process variable (PV)**, **setpoint**, and **output** displayed on a large strip chart with the axes showing the values versus time. Perturb the loop as described in the *Closed-Loop (Ultimate Gain) Tuning Procedure* and *Open-Loop (Step Test) Tuning Procedure* sections of this chapter and determine the response from the graph. Refer to Corripio (1990) as listed in Appendix A, *References*, for more information on 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 **process variable** on a strip chart. To perform the closed-loop tuning procedure, complete the following steps:

1. Set both the **rate** and **reset** on your PID controller to 0.
2. With the controller in automatic mode, carefully increase the **proportional gain** (K_c) in small steps. Disturb the loop after each step by making a small change in the **setpoint**. The process variable should start oscillating as you increase the K_c . Keep making changes until the oscillation is perfectly sustained, neither growing nor decaying over time.
3. Record the controller **proportional band** as PB_u as a percent, where $PB_u = 100 / K_c$.
4. Record the period of oscillation as 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. The table provides the proper values for a quarter-decay ratio.

If you want less overshoot, increase the PB, which has the same effect as reducing the gain.

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	$1.67PB_u$	$0.50T_u$	$0.125T_u$



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 less hazardous for many processes. Observe the output and the **process variable** (PV) on a strip chart that shows time on the X axis. To perform the open-loop tuning procedure, complete the following steps:

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.

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

The values are as follows:

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

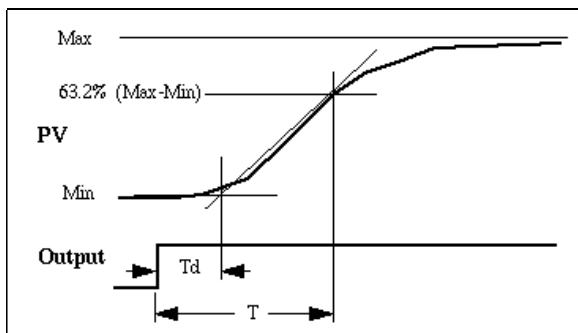


Figure 3-3. Output and Process Variable Strip Chart

- 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, increase the PB, which has the same effect as reducing 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, all use the same algorithm described in Chapter 2. The PID VI is the basic PID algorithm. Other variations provide additional functionality as described in the following sections. All of these VIs use consistent inputs and outputs where appropriate, so you can use them interchangeably.

The PID VI

The PID VI has inputs for **setpoint**, **process variable**, **manual control** and **PID parameters**. The **PID parameters** input is a cluster of three values for proportional gain (or proportional band), integral time, and derivative time. Use the cluster input **options** to specify additional input parameters. You can leave them unwired for most applications. One parameter in the **options** cluster is a boolean value that specifies whether the proportional input parameter is the **proportional gain** (K_c) or **proportional band** (PB). Gain is related to **proportional band** as $K_c = 100/PB$. By default, this option specifies proportional gain input in the **PID parameters** input.

You can specify several other options for the PID VI. You can set Boolean input values to hold the controller **output** value at its current state or control the output with the **manual control** input. Additionally, you can specify ranges for the **setpoint** and controller **output** (the **process variable** range corresponds to the **setpoint** range). These ranges default at 0 to 100 for **setpoint** and -100 to 100 for the controller **output**; thus, these ranges correspond to input and output values specified in terms of percentage of full scale. However, you can change these ranges to values appropriate for your control system so that the controller gain relates engineering units to engineering units instead of percentage to percentage. Additional parameters include the **linearity** and **beta** parameters described in Chapter 2 and the **reverse acting** Boolean value. With the **dt** option, you can specify the control loop cycle time, which defaults to -1, indicating that the PID VI uses the operating system clock for dynamic control characteristics (integral and derivative action). If the loop cycle time is deterministic, you can provide this input to the PID VI.

The **iteration** input is used for process variable filtering as described in *The PID Algorithm* section in Chapter 2. This input value prevents the PID algorithm from filtering values before the third iteration of the control loop. To enable filtering, wire the iteration terminal of your control loop to this input. This value defaults to 0, therefore disabling process variable filtering when left unwired.



Note *The iteration input must be wired for derivative and integral action.*

The PID (Gain Schedule) VI

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 may not produce a satisfactory response at another operating point.

The gain-scheduling PID VI input and output values are identical to the PID VI with two exceptions:

- The **gain schedule** input replaces **PID parameter** input.
- The gain-scheduling PID VI provides an additional **gain scheduling variable** input.

The **gain schedule** input is a cluster consisting of a **gain schedule** array, an array of **PID parameters**, and an input to specify which variable is the **gain scheduling variable**. The PID gains can be scheduled according to the **setpoint**, **process variable**, **controller output**, or a user-defined scheduling variable wired to the **gain scheduling variable** input.

The **gain schedule** array specifies the ranges of the scheduling variable corresponding to the sets of PID parameters in the **PID parameters** array. The first value of the **PID parameters** array corresponds to the scheduling variable range from the minimum value to the first value of the **gain schedule** array. The second value of the PID parameters array corresponds to the scheduling variable range from the first **gain schedule** array value to the second **gain schedule** array value. Subsequent sets follow the same pattern so that the last value of the **gain schedule** array should correspond to the maximum value of the scheduling variable.



Note *The gain schedule and PID parameters arrays should always have the same array length.*

In Figure 3-4 the **setpoint** value is used as the **gain scheduling variable** with a default range of 0 to 100. Table 3-3 summarizes the **PID parameters**.

Table 3-3. PID Parameter Ranges

Range	Parameters
$0 \leq SP \leq 30$	Kc = 10, Ti = 0.02, Td = 0.02
$30 < SP \leq 70$	Kc = 12, Ti = 0.02, Td = 0.01
$70 < SP \leq 100$	Kc = 15, Ti = 0.02, Td = 0.005

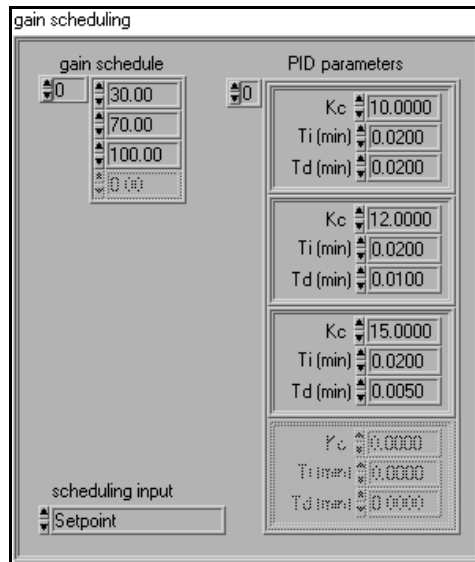


Figure 3-4. Gain Scheduling Input Example

The PID (Compatibility) VI

This toolkit includes the PID (compatibility) VI for users of previous versions of the PID Control software. This VI implements the PID algorithm described in Chapter 2 using input and output consistent with those of the PID VI of previous software versions. For this VI, the proportional parameter is specified as the **proportional band PB**, not the **proportional gain K_c**. The **setpoint** and **process variable** default range is from 0 to 100 and the controller **output** range is -100 to 100 . Express the input and output values as percentages. This algorithm is the same as the

PID VI, with the **beta** and **linearity** parameters set to 1. You can set other optional parameters as input.

To use the compatibility VI in an existing control application, pop-up on the PID VI and choose **Replace**. Then you can choose the PID (compatibility) VI from the functions palette. All input values are identical except the additional **iteration** input on the compatibility VI. Wire the control loop iteration terminal to this input to enable process variable filtering and derivative and integral action.

The PID with Autotuning VI and the Autotuning Wizard

To use the Autotuning Wizard to improve your controller performance, you must first develop 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, PID (gain schedule) VI, or the PID with autotuning VI. Because the autotuning PID VI has input and output consistent with the other PID VIs, you can pop-up on any PID VI and replace it with the PID with autotuning VI.

The autotuning VI has several additional input and output values to specify the autotuning procedure. The two additional input values are **autotuning parameters** and **autotune?**. The **autotuning parameters** input is a cluster of parameters used for the autotuning process. Because the Autotuning Wizard allows you to specify all of these parameters manually, you may leave the **autotuning parameters** input unwired. The **autotune?** input is a Boolean value that specifies when to begin autotuning. Wire a Boolean control on the front panel of your application to this input. Set the Boolean control mechanical action to **Latch when released** so that the Autotuning Wizard is not invoked repeatedly when the user presses the Boolean control. The Autotuning Wizard steps you through the autotuning process as described in [The Autotuning Algorithm](#) section in Chapter 2 and provides online help if you need it.

The autotuning VI also has two additional output values: **tuning completed?** and **updated parameters**. The **tuning completed?** output is a Boolean value indicating that the autotuning process is complete.

It is usually FALSE and set to TRUE only on the iteration during which the autotuning is completed. The **updated parameters** output provides the updated PID parameters (**gain scheduling** cluster) from the autotuning procedure. Normally, the **updated parameters** output value passes through the **gain scheduling** cluster input values and outputs the **updated parameters** only when the autotuning procedure completes. You have several ways to use these outputs in your applications.

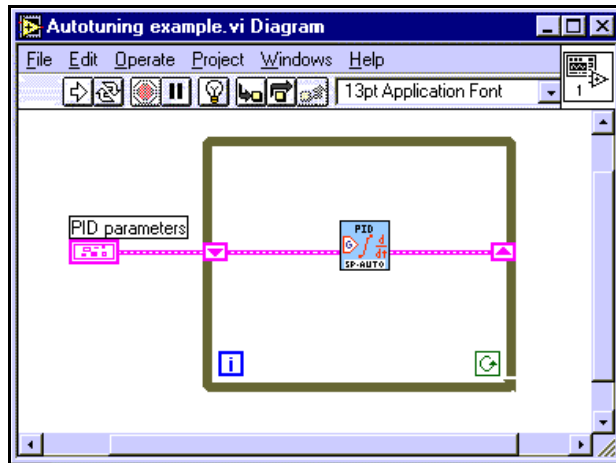


Figure 3-5. Updating PID Parameters Using a Shift Register

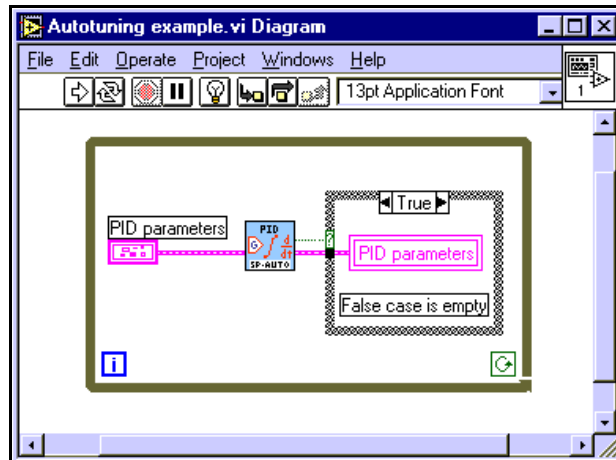


Figure 3-6. Updating PID Parameters Using a Local Variable

Figure 3-5 shows one possible implementation of the PID with autotuning VI. Notice the use of a shift register to store the initial value of the PID parameters. The **updated parameters** output value is then passed to the right shift register terminal when each control loop iteration completes. Although this method is simple, it suffers from one limitation: the **PID parameters** cannot be changed manually while the control loop is running. A second method in Figure 3-6 shows a local variable used to store the updated **PID parameters**. In this example, the **PID parameters** control is read 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 parameters** while the control loop is executing.

In both examples, **PID parameters** must be saved manually such that the **updated parameters** from the autotuning process are retained for the next control-application run. Ensure that the **PID parameters** control shows the current **updated parameters**. Choose **Make Current Values Default** from the **Operate** menu, and save the VI manually.

To avoid manually saving the VI each time it is run, you can use a datalog file to save the **PID parameters**. An example of this process is shown in Figure 3-7.

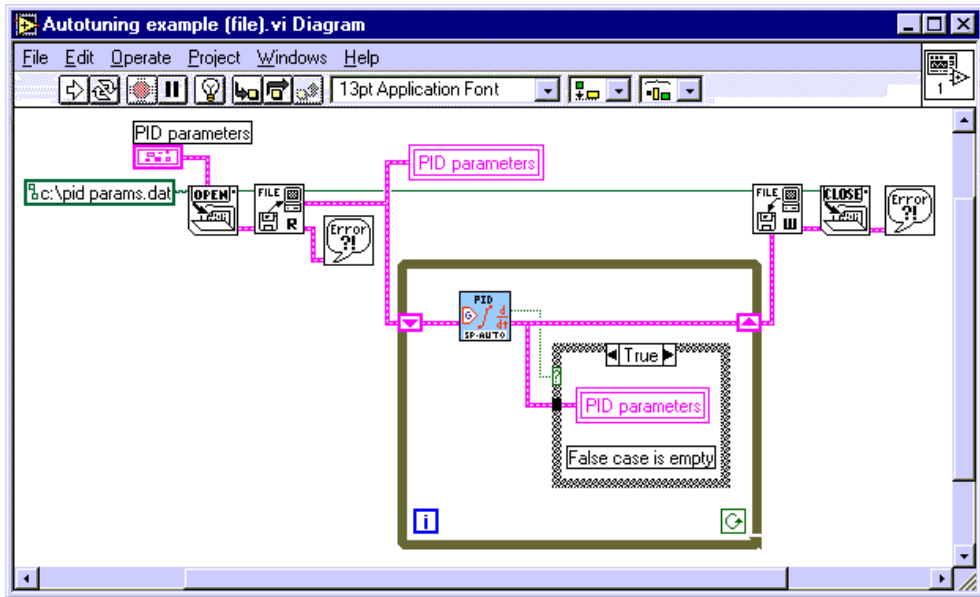


Figure 3-7. Storing PID Parameters in a Datalog File

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

Using PID with Data Acquisition Hardware (DAQ)

The remaining sections in this chapter address several important issues if you are implementing PID control of an actual process using the Data Acquisition (DAQ) VIs. The following examples illustrate the use of easy-level DAQ functions compared with low-level DAQ functions and hardware timing compared to software timing. In addition, some BridgeVIEW DAQ applications are explained.

Software-Timed DAQ Control Loop

Figure 3-8 illustrates the basic elements of software control. The model assumes you have a plant (your real process) to control. A simple analog input VI reads process variables from sensors monitoring the process. In actual applications, you might need to scale values to engineering units instead of voltages.

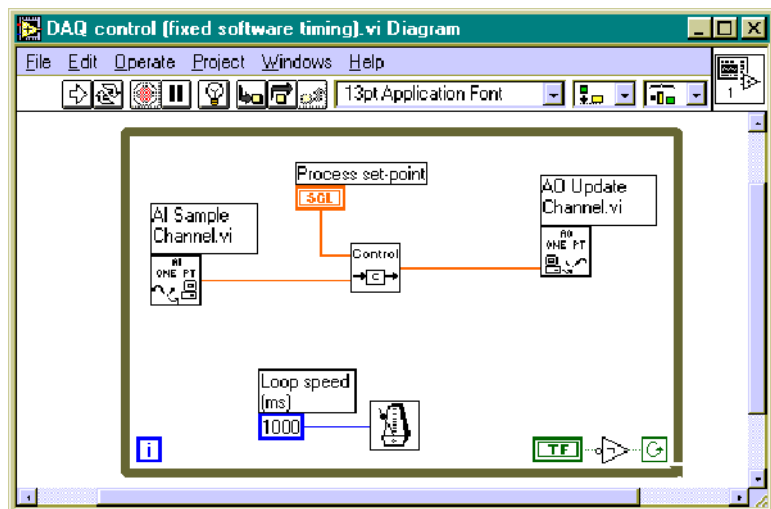


Figure 3-8. Software-Timed DAQ Control Loop

The Control VI represents the algorithm used to implement software control. It may be a subVI you have written in G, a PID controller from the PID toolkit, or the Fuzzy Controller VI from the Fuzzy Logic toolkit. An analog output VI updates the analog voltages that serve as your controller outputs to the process.

This example also uses a Wait Until Next ms Multiple function to control the loop timing. Notice that this function specifies only a minimum time for the loop to execute. Other operations in LabVIEW and BridgeVIEW can cause execution time of the loop functions to exceed the specified time. The time for the first loop iteration is not deterministic (see the online reference for details).

Software-Timed DAQ Control Loop Using Advanced DAQ Functions

For faster I/O and loop speeds, use the advanced-level DAQ functions for analog input and output. The easy-level functions used in Figure 3-8 actually use the lower-level DAQ functions shown in this example; however, the easy-level VIs configure the analog input and output with each loop iteration, creating unnecessary overhead that can slow your control loops.

By using VIs from the advanced subpalettes shown in Figure 3-9, you can 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 VIs AI SingleScan and AO Single Update call the code interface nodes directly instead of through other subVI calls, minimizing overhead for DAQ functions.

Notice that this example does not use a timing function to specify the loop speed; thus, the control loop runs as fast as possible and the control loop rates are maximized. Yet, any other operation in LabVIEW or BridgeVIEW causes the loop to slow down, and the speed from iteration to iteration might vary. Because Windows NT/95 is a pre-emptive multitasking operating system, other applications that you open and run have an effect on the loop speed.

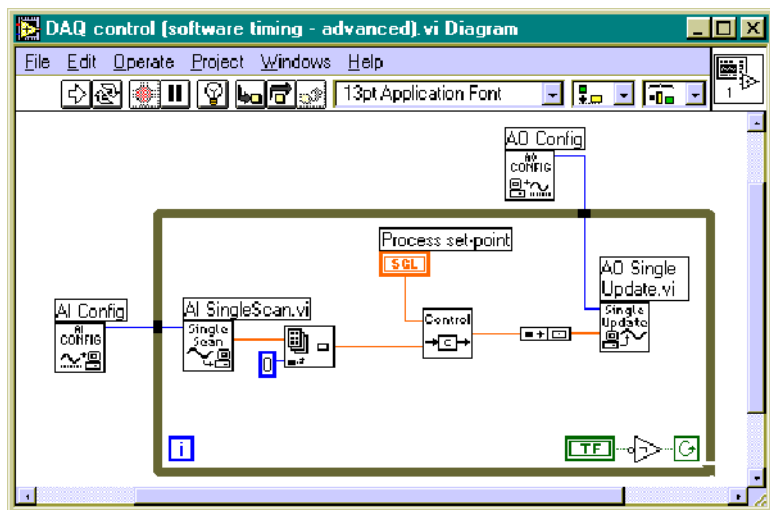


Figure 3-9. Software-Timed DAQ Control Loop with Advanced Features

Hardware-Timed DAQ Control Loop

Figure 3-10 demonstrates hardware timing rather than software timing. Here, a continuous analog input operation controls the loop speed. Notice that the intermediate- and advanced-level DAQ functions specify the acquisition rate for the analog input scanning operation. Analog output functions are identical to those in the previous example.

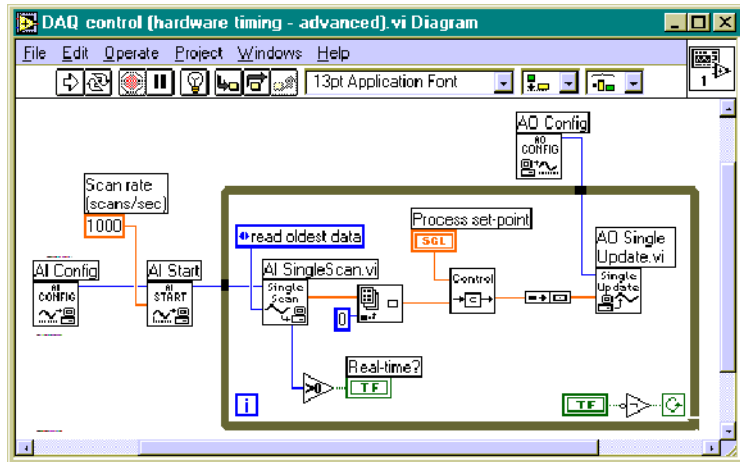


Figure 3-10. Hardware-Timed DAQ Control Loop

With each loop iteration, the AI SingleScan VI returns one scan of data, when available. The control VI processes data, and the analog output channels are updated 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 processing of the analog input, control algorithm, and analog output takes longer than the specified scan interval (1 ms in this example), the software falls behind the hardware acquisition rate and does not maintain real time. By monitoring the scan backlog when calling AI SingleScan, you can monitor this condition to determine if scans have been missed. If this value remains zero, the control is real time.

Event-Driven DAQ Control Loop (BridgeVIEW)

Figure 3-11 shows the use of event-driven loop timing using a VI written in BridgeVIEW. With the DAQ device server, tags are configured for the analog input and output values. By using the Read Tag and Write Tag VIs, the control loop executes on a change-only basis; the rate of change of the process variable dictates the rate of the control loop. This is another example of a control loop that may have variable timing.

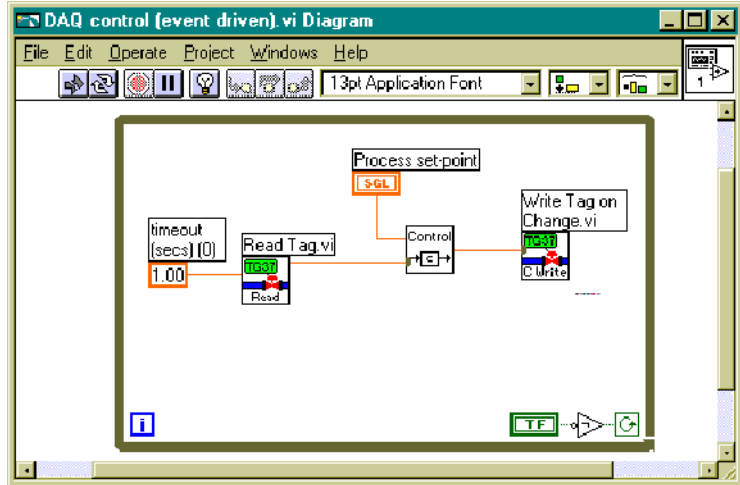


Figure 3-11. Event-Driven DAQ Control Loop (BridgeVIEW)

You can use the **timeout** value for the Read Tag VI to specify a maximum loop iteration time for slowly changing process variables, ensuring a minimum loop rate for the control loop. You may want to implement this limit for cases where the control algorithm is time dependent (for example, the integral action of a PID controller).

You can implement all previous examples of control loops using DAQ exactly the same way in both BridgeVIEW and LabVIEW, whereas the method shown in Figure 3-11 applies only to BridgeVIEW.

Using a VI Server for DAQ Control

Another possible implementation in BridgeVIEW involves using your VI as a server registered with the BridgeVIEW engine. In Figure 3-12, the simple control loop used in the first example also uses the SRVR Write Input Queue VI to update the input and output values to the BridgeVIEW engine. By writing the input and output values to tags configured in the BridgeVIEW engine, you can monitor these values from your BridgeVIEW user application. The setpoint value is read using the SRVR Read Output Queue VI.

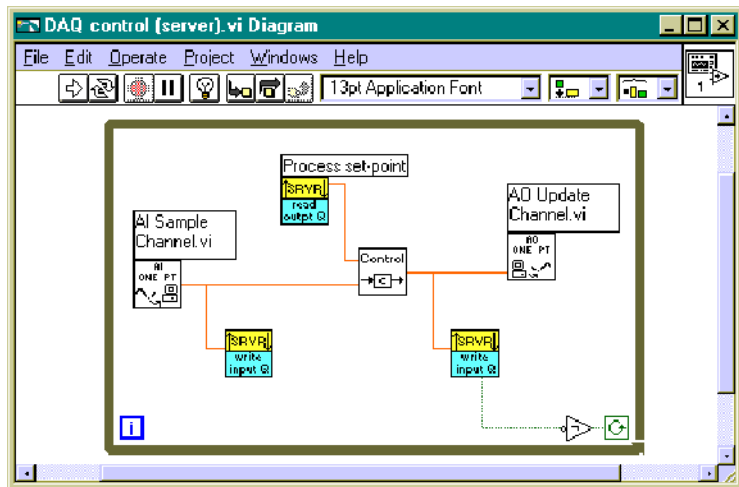


Figure 3-12. VI Server Example for DAQ Control

Notice the primary advantage to using this method: the control loop runs as a separate process from your BridgeVIEW HMI VIs. Thus, operations (for example, file I/O) that may temporarily monopolize the BridgeVIEW execution thread do not completely halt the control application. Rather, both operations run as separate tasks on the CPU, and a pre-emptive multitasking operating system, such as Windows NT/95, share processor time between the two tasks. Operations in your BridgeVIEW user application still affect the control loop speed, but the effect is minimized.

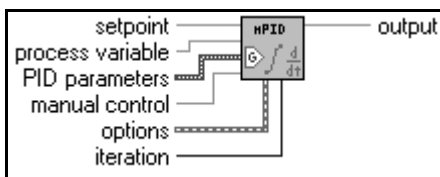
For a similar approach in LabVIEW, create an executable program with the Application Builder package. The executable has its own thread but shares execution time with your LabVIEW user application. However, you cannot directly monitor the input and output values of the executable from your LabVIEW applications.

PID Software VIs

This chapter contains descriptions of the six PID VIs. Two VIs remain from the previous toolkit: Lead-Lag and Ramp. The four new VIs include PID, PID (gain schedule), PID (compatibility), and PID with autotuning.

PID

Calculates an analog value based on the current value of the **process variable** and **setpoint** using the PID algorithm.



setpoint is the desired value for the process variable.



process variable is the value of the feedback control loop.



PID parameters is a cluster of proportional gain, integral time, and derivative time parameters.



Kc is the proportional gain.



Ti is the integral time in minutes. A value of 0 disables integral action.



Td is the derivative time in minutes. A value of 0 disables derivative action.



manual control is a relative controller output value.



options is a cluster of 11 elements specifying optional parameters for the PID algorithm.



sp low is the minimum value for **process variable** and **setpoint**. The default is 0.



sp high is the maximum value for **process variable** and **setpoint**. The default is 100.



out low is the minimum value for controller **output**. The default is -100.



out high is the maximum value for controller **output**. The default is 100.



hold (F), when TRUE, puts the controller in hold mode. Reset action stops and **output** freezes. Bumpless transfer is used from run to hold.



auto (T), if TRUE, selects automatic control and puts the controller in manual mode when FALSE. Bumpless transfer is used from automatic to manual.



pro. band (F) selects whether the proportional value of **PID parameters** input is proportional gain or proportional band. The default value is FALSE, specifying proportional gain. Proportional gain (K_c) is $100/PB$ where PB is proportional band.



reverse acting (T), if TRUE, selects reverse (increase-decrease) action, the usual mode for controllers (the **output** decreases if the input is greater than the **setpoint**).



beta is the relative emphasis of disturbance rejection to **setpoint** tracking. The default value of 1 is appropriate for most applications. A smaller value between 0 and 1 can be used to specify emphasis on disturbance rejection (such as process load changes).



linearity sets the linearity of error response, ranging from zero to one. 1.0 gives a plain linear response, while 0.1 gives an approximately parabolic (square law) response.



dt (s) is the interval (in seconds) at which this VI is called. If the interval is less than or equal to zero, an internal timer uses a 1 ms resolution.



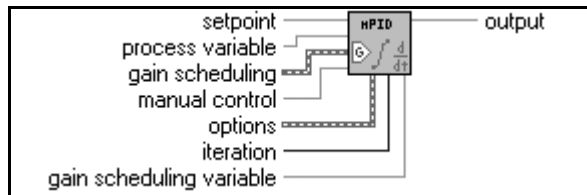
iteration is the control loop iteration value. **iteration** must be wired from the application control loop iteration terminal for derivative or integral action. The default value is zero.



output is the output of the control algorithm.

PID (Gain Scheduling)

Use this VI to implement a gain schedule for the proportional gain, integral time, and derivative time.



setpoint is the desired value for the process variable.



process variable is the value of the feedback control loop.



gain scheduling defines the gain schedule for proportional, integral, and derivative parameters.



gain schedule defines the ranges of the **gain scheduling variable** for the respective **PID parameters**. Each value specifies the maximum value of the range of the **gain scheduling variable**.



PID parameters is an array of proportional gain, integral time, and derivative time parameters. Each set of **PID parameters** corresponds to a range of the **gain schedule** input.



PID Param is a cluster of proportional gain, integral time, and derivative time parameters.



Kc is the proportional gain.



Ti is the integral time in minutes. A value of 0 disables integral action.



Td is the derivative time in minutes. A value of 0 disables derivative action.



scheduling input defines which value is used as the **gain scheduling variable**. Selecting **gain scheduling input** indicates that the value wired to the **gain scheduling variable** input is to be used for gain scheduling.



manual control is a relative controller output value.



options is a cluster of 11 elements specifying optional parameters for the PID algorithm.



sp low is the minimum value for **process variable** and **setpoint**. The default is 0.



sp high is the maximum value for **process variable** and **setpoint**. The default is 100.



out low is the minimum value for controller **output**. The default is -100.



out high is the maximum value for controller **output**. The default is 100.



hold (F), if TRUE, puts the controller in hold mode. Reset action stops and the **output** freezes. Bumpless transfer is used from run to hold.



auto (T), if TRUE, selects automatic control and puts the controller in manual when FALSE. Bumpless transfer is used from automatic to manual.



pro. band (F) selects if the proportional value of **PID parameters** input is proportional gain or proportional band. The default value is FALSE, which specifies proportional gain. Proportional gain (K_c) is $100/PB$ where PB is proportional band.



reverse acting (T), if TRUE, selects **reverse acting** (increase-decrease), the usual mode for controllers where **output** decreases if the input is greater than **setpoint**.



beta specifies the relative emphasis of disturbance rejection to setpoint tracking. The default value of 1 is appropriate for most applications. A smaller value between zero and one can be used to specify emphasis on disturbance rejection (such as process load changes).



linearity sets the linearity of error response, ranging from zero to one. 1.0 gives a plain linear response, while 0.1 gives an approximately parabolic (square law) response.



dt (s) is the interval (in seconds) at which this VI is called. If the interval is less than or equal to zero, an internal timer uses a 1 ms resolution.



iteration is the control loop iteration value. **iteration** must be wired from the application control loop iteration terminal for derivative or integral action. The default value is zero.



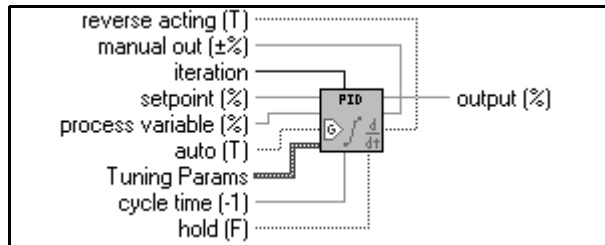
gain scheduling variable is the value used to determine **PID parameters** from **gain schedule**. This input is used only when the scheduling input is set to **gain scheduling input**. The **setpoint**, **process variable**, controller **output**, or a user-provided value can serve as the **scheduling variable**.



output is the output of the control algorithm.

PID (Compatibility)

Use this VI with any PID VIs from the previous PID toolkit to achieve compatibility. The inputs and outputs are identical to the previous PID VI; however, this VI uses the revised PID algorithm. To use this VI in an existing application, simply pop-up on the PID VI and choose **Replace** to select this VI. There is one additional input. The **iteration** input should be wired from the iteration terminal of your control loop to allow for initialization of the PID algorithm when the control application is run.



reverse acting (T), if TRUE, selects **reverse acting** (increase-decrease), the usual mode for controllers in which **output** decreases if the input is greater than **setpoint**.



manual out (±%) is the manual output setting, expressed as a relative percentage control.



iteration is the control loop iteration value. This input must be wired from the application control loop iteration terminal for derivative or integral action. The default value is zero.



setpoint (%) is the process **setpoint**, expressed as a percent ranging from 0 to 100%.



process variable (%) is the object you are trying to control, expressed as a percent ranging from 0 to 100%.



auto (T), if TRUE, selects automatic control and puts the controller in manual when FALSE. Bumpless transfer is used from automatic to manual.



Tuning Params is a cluster containing the following parameters:



Kc is the proportional band in percent. Controller gain is $100/PB$.



Ti is the integral i minutes per reset. Enter 0 to disable reset.



Td is the rate in minutes per repeat. Enter 0 to disable derivative action.



cycle time (-1) is the interval (in seconds) at which this VI is called. If **cycle time** is less than or equal to zero, an internal timer with 1 ms resolution is used.



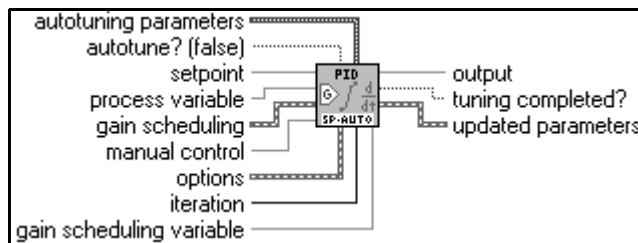
hold (F), if TRUE, puts the controller in hold mode. Reset action stops and the **output** freezes. Bumpless transfer is used from run to hold.



output (%) is the output of control algorithm expressed as a percentage ranging from 0 to 100%.

PID with Autotuning

Use this VI as a basic PID VI with gain scheduling. This VI uses inputs to specify and initiate a setpoint relay autotuning procedure. Passing a TRUE value to the **autotune?** input initiates the Autotuning Wizard.



autotuning parameters are various parameters used for the autotuning process. You can select these values manually in the Autotuning Wizard.



Controller Type specifies whether the output of the tuning process should be proportional/integral/derivative parameters, proportional/integral parameters, or proportional only.



of cycles to avg specifies the number of **setpoint** relay cycles used to determine the ultimate gain and period. More cycles result in better parameter estimation; however, slower systems may require too much time for many cycles.



Relay amplitude determines amplitude for setpoint relay action. Setpoint relay is between (**setpoint – Relay amplitude**) and (**setpoint + Relay amplitude**).



Control design specifies the desired response performance of **PID parameters** determined by the autotuning process. Faster response generally results in a smaller rise time, but slower response results in less overshoot.



Noise level is an estimation of the noise level of the measured **process variable**. This value is used as a hysteresis for the setpoint relay action.



autotune? (false), if TRUE, indicates the start of the autotuning procedure and invokes the Autotuning Wizard. This input should be wired from a Boolean control with latched mechanical action and a default value of FALSE.



setpoint is the desired value for the process variable.



process variable is the value of the feedback control loop.



gain scheduling defines **gain schedule** for proportional, integral, and derivative parameters.



gain schedule defines the ranges of the **gain scheduling variable** for the respective **PID parameters**. Each value specifies the maximum value of the range of the **gain scheduling variable**.



PID parameters is an array of proportional gain, integral time, and derivative time parameters. Each set of **PID parameters** corresponds to a range of the **gain schedule input**.



PID Param is a cluster of proportional gain, integral time, and derivative time parameters.



Kc is the proportional gain.



Ti is the integral time in minutes. A value of 0 disables integral action.



Td is the derivative time in minutes. A value of 0 disables derivative action.



scheduling input defines which value is used as the **gain scheduling variable**. Selecting **gain scheduling input** indicates that the value wired to the **gain scheduling variable** input is to be used for gain scheduling.



manual control is a relative controller output value.



options are various optional parameters for the PID algorithm.



sp low is the minimum value for **process variable** and **setpoint**. The default is 0.



sp high is the maximum value for **process variable** and **setpoint**. The default is 100.



out low is the minimum value for controller **output**. The default is -100.



out high is the maximum value for controller **output**. The default is 100.



hold (F), if TRUE, puts the controller in hold mode. Reset action stops and the output freezes. Bumpless transfer is used from run to hold.



auto (T), if TRUE, selects automatic control and puts the controller in manual when FALSE. Bumpless transfer is used from automatic to manual.



pro. band (F) selects whether the proportional value of **PID parameters** input is proportional gain or proportional band. The default value is false, specifying proportional gain. Proportional gain (K_c) is $100/PB$ where PB is proportional band.



reverse acting (T), if TRUE, selects **reverse acting** (increase-decrease), the usual mode for controllers where **output** decreases if the input is greater than **setpoint**.



beta specifies the relative emphasis of disturbance rejection to **setpoint** tracking. The default value of 1 is appropriate for most applications. A smaller value between 0 and 1 can be used to specify emphasis on disturbance rejection (such as process load changes).



linearity sets the linearity of error response, ranging from zero to one. 1.0 gives a plain linear response, while 0.1 gives an approximately parabolic (square law) response.



dt (s) is the interval (in seconds) at which this VI is called. If the interval is less than or equal to zero, an internal timer uses a 1 ms resolution.



iteration is the control loop iteration value. **iteration** must be wired from the application control loop iteration terminal for derivative or integral action. The default value is zero.



gain scheduling variable determines **PID parameters** from **gain schedule** only when the scheduling input is set to **gain scheduling input**.



output is the output of control algorithm.



tuning completed? indicates the completion of the autotuning process. You can use it to determine when to update PID gain values as well.



updated parameters are the **gain scheduling** parameters with updated PID parameters after the autotuning process completes. Normal output is an echo of the **gain scheduling** input cluster.



gain schedule defines the ranges of the **gain scheduling variable** for the respective **PID parameters**. Each value specifies the maximum value of the range of the **gain scheduling variable**.



PID parameters is an array of proportional gain, integral time, and derivative time parameters. Each set of **PID parameters** corresponds to a range of the **gain schedule input**.



PID Param is a cluster of proportional gain, integral time, and derivative time parameters.



Kc is the proportional gain.



Ti (min) is the integral time in minutes. A value of 0 disables integral action.



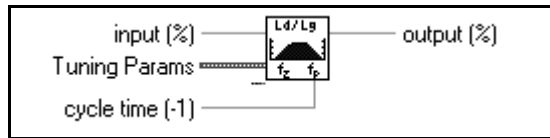
Td (min) is the derivative time in minutes. A value of 0 disables derivative action.



scheduling input defines which value is used as the **gain scheduling variable**. Selecting **gain scheduling input** indicates that the value wired to the **gain scheduling variable** input is to be used for gain scheduling.

Lead-Lag

Calculates the dynamic compensator in feedforward control schemes.



The general transfer function for this block is as follows:

$$output = \left(\frac{T_{lead}s + 1}{T_{lag}s + 1} \right) input ,$$

where T_{lead} is the **lead time**, T_{lag} is the **lag time**, and s is the Laplacian frequency operator.

Like the PID VIs, you should call this VI from inside a While Loop with a fixed **cycle time**. For an example of this VI, see the [Lead-Lag](#) section in Chapter 5, *Process Control Examples*.



input is expressed in percent. The range is 0 to 100%. **input** defaults to zero.



Tuning Params is a cluster containing the following parameters:



gain is expressed in percent. The range is $-\infty$ to ∞ . **gain** defaults to 1.0.



lag time is expressed in minutes. The range is 0 to ∞ . **lag time** defaults to 0.01. Zero turns **lag time** off.



lead time is expressed in minutes. The range is 0 to ∞ . **lead time** defaults to zero. Zero turns **lead time** off.



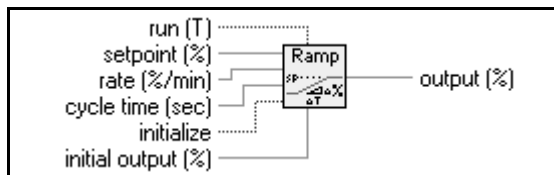
cycle time is often a value you supply, expressed in seconds. Values less than or equal to zero force the VI to use an internal timer. See the [Setting Timing](#) section in Chapter 3, *Using the PID Software*, for detailed information on timer resolution. **cycle time** defaults to -1 .



output is expressed in percent. The range is -100 to 200% .

Ramp

Generates a setpoint ramp.



As with PID VIs, you should call this VI from inside a While Loop with a fixed **cycle time**. You must supply a **cycle time**. Control calibration depends on this value.



run determines whether to execute the ramp. If **run** is TRUE, the VI executes the ramp. If **run** is FALSE, the VI suspends execution and does not return a new value. The default is TRUE.



setpoint is the desired value for the process variable, expressed in percent. The range is 0 to 100%. **setpoint** defaults to zero.



rate is the ramp slope in percentage per minute. **rate** defaults to zero. Positive values make the ramp increase with time.



cycle time is the time interval between calls to this VI, expressed in seconds. **cycle time** defaults to 1.0.



initialize determines whether to force **output** to **initial output**. If **initialize** is TRUE, the VI forces **output** to **initial output**. If **initialize** is FALSE, the VI performs normal ramp action. The **initialize** value defaults to FALSE and overrides **run**.



initial output is the value the VI copies to **output** when **initialize** is TRUE. **initial output** is expressed in percent and defaults to zero. The range is 0 to 100%.



output is expressed in percent. The range is 0 to 100%. **output** ramps linearly toward the **setpoint** at **rate** percent per minute. The sign (\pm) of **rate** determines the polarity of the ramp. **output** does not overshoot the **setpoint**.

Process Control Examples

This chapter describes examples that use the PID Control VIs. You do not need any data acquisition plug-in boards to run the Simulation VIs, but you must have the appropriate hardware to run the Demonstration VIs.

Simulation VIs

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 can also switch from auto to manual.

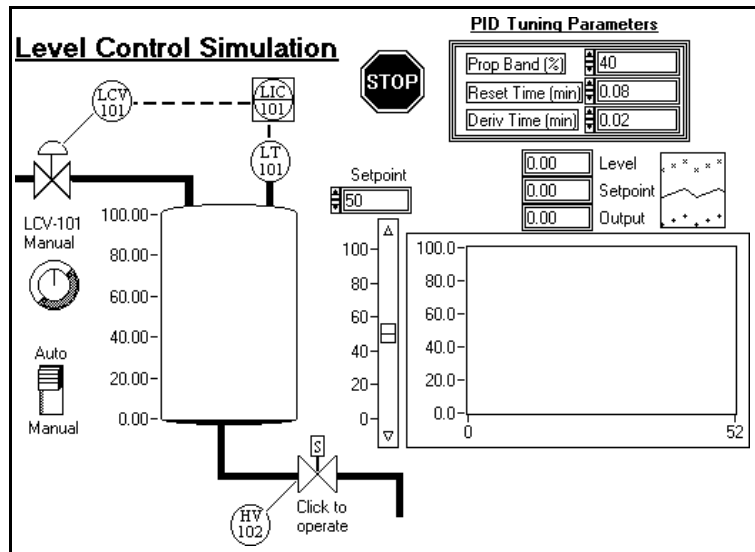


Figure 5-1. Front Panel of the Tank Level VI

The Tank Level VI uses an integrating process with added noise, valve deadband, lag, and deadtime. It 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.

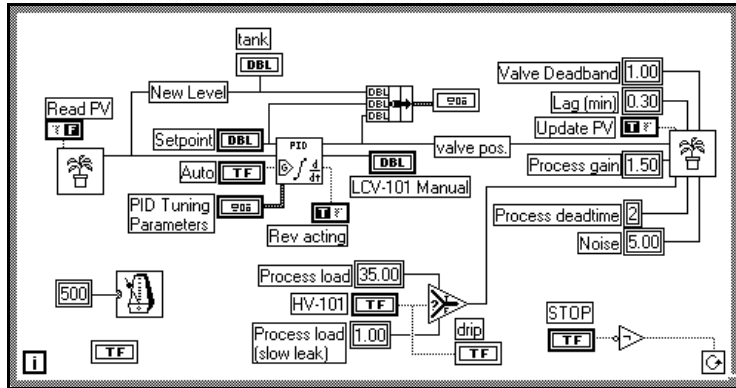


Figure 5-2. Block Diagram of the Tank Level VI

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 HV-101, the drain valve. When you open the valve, the tank level drops.

General PID

The General PID VI is similar to the Tank Level VI, except that all process adjustments appear on the front panel of the General PID VI (see Figure 5-3). 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, this VI does not correct itself for the loop cycle time. Unless you want to modify the process, you should set **Cycle Time** to 1 s.

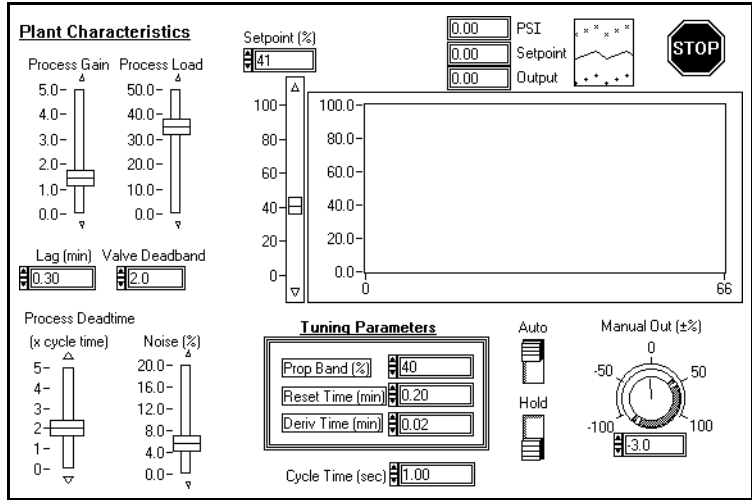


Figure 5-3. Front Panel of the General PID VI

This VI also demonstrates switching from automatic mode to manual mode and from run mode to hold mode.

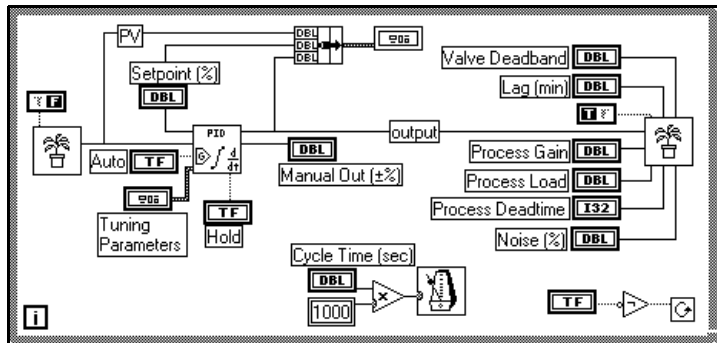


Figure 5-4. Block Diagram of a Pressure Control PID VI

Like the Tank Level VI, this pressure controller simulation uses the Plant Simulator subVI. 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. Therefore, if you change **Cycle Time**, you must adjust **Process Deadtime** to keep the process response constant. Because the lag time is time aware, you do not need to adjust **Lag**.

Plant Simulator

The Plant Simulator VI simulates a physical plant response.

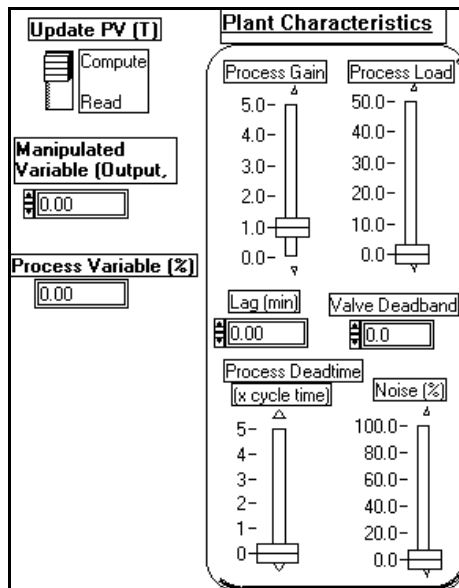


Figure 5-5. Front Panel of the Plant Simulator VI

This VI measures the **process variable (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. Finally, call this VI with the **Update PV** switch set to TRUE. For the VI to calculate the new PV value, supply all the plant characteristics and connect the PID output to the **Manipulated Variable**, as shown in Figure 5-6.

When the **Update PV** control is FALSE, the value of the last **Process Variable** passes to the output. When the **Update PV** control is TRUE, the VI reads and delays the **Process Variable**, 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 (instead of using the Addition function), adding a reasonable time constant to the apparent response of the tank. The output of this filter represents the new process variable.

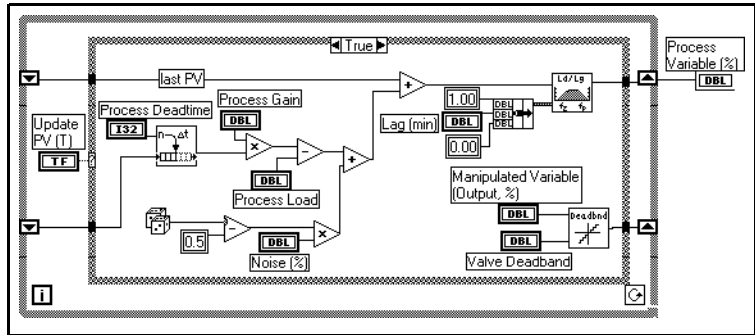


Figure 5-6. Block Diagram of the Plant Simulator VI

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 requiring 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 setpoint, 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 setpoint for the compressor speed.

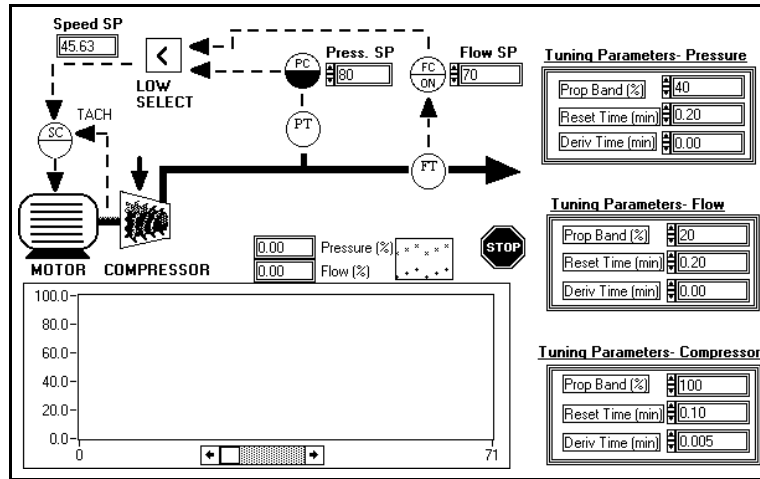


Figure 5-7. Front Panel of the Cascade and Selector VI

When you run the VI, the **PC** (pressure controller) and **FC** (flow controller) symbols read ON when you select them. Normally, you should leave the pressure setpoint constant. Try making a large upward step in flow and notice that the **PC** takes over control as pressure overshoots for a while. 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 passing 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. You should use a factor of 10 to prevent oscillation. In this simulation, the compressor lag is smaller (faster) than that 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 and passes 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.

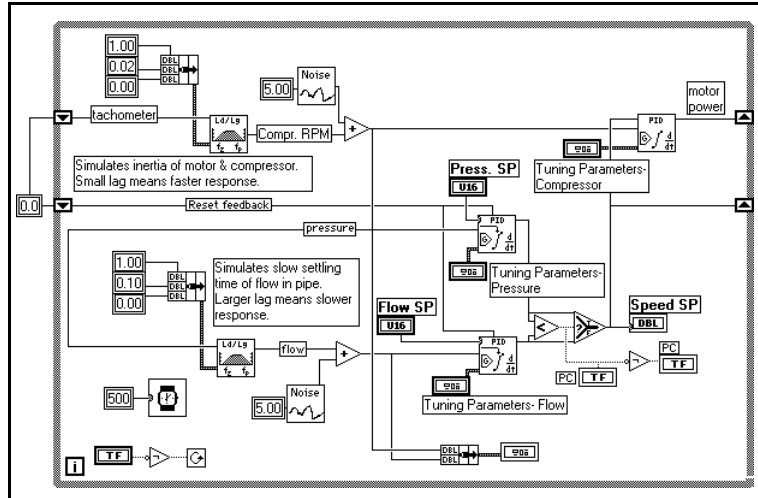


Figure 5-8. Block Diagram of the Cascade and Selector VI

The right half of the diagram contains two PID VIs—one for pressure control and one for flow control. The While Loop that produces the lowest output value is selected as the active controller, and its output is routed to the compressor speed control PID setpoint.

Feedback for the two controllers is derived 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

PID with MIO Board

The PID with MIO Board VI turns your computer into a real single-loop PID controller through the use of a National Instruments data acquisition board. Connect the analog output to the analog input through the resistor-capacitor network shown on the front panel in Figure 5-9.

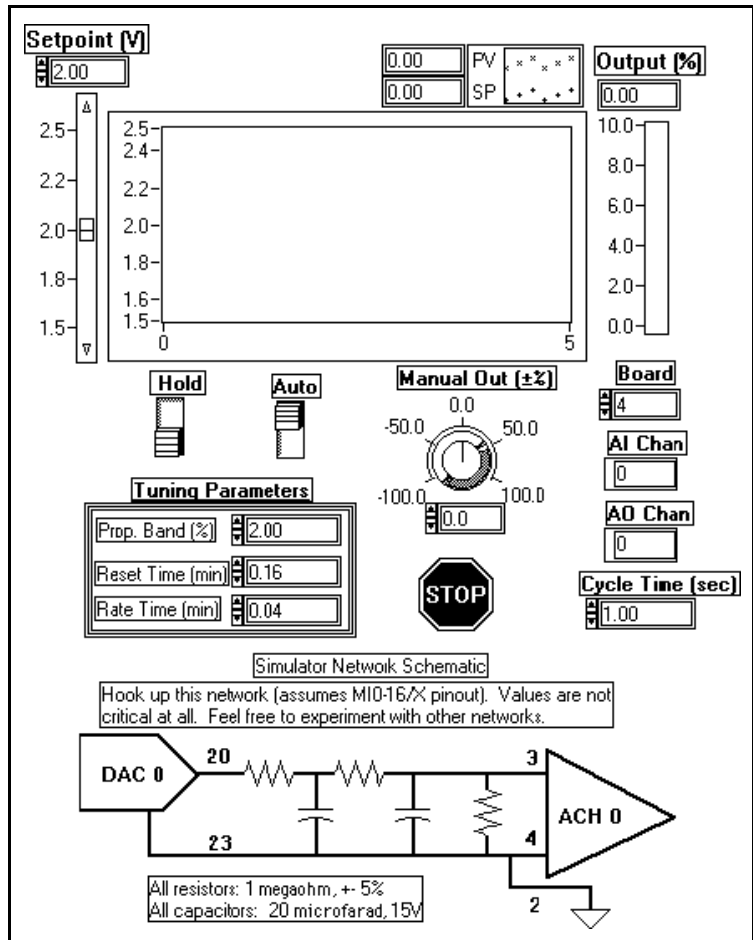


Figure 5-9. Front Panel of the PID VI with Controls Set for an MIO Data Acquisition Board

This VI adjusts the analog output so that the input (process variable or PV) equals the setpoint (SP). The VI displays SP and PV on a strip chart. You can experiment with different controller tuning methods and try for the fastest settling time with the least overshoot. The default tuning parameters are optimum for the network shown in Figure 5-10.

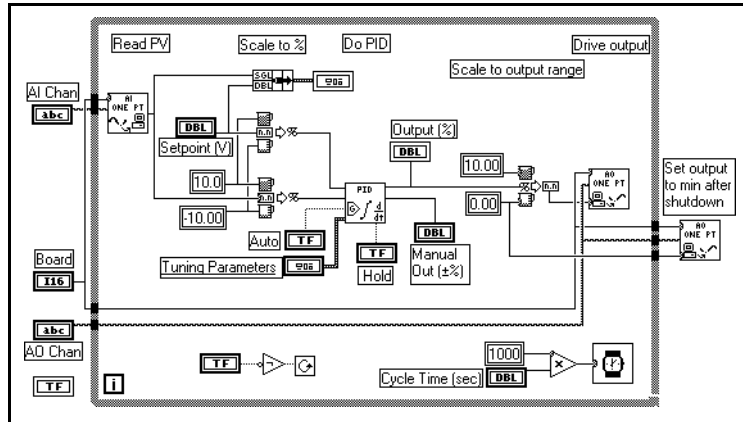


Figure 5-10. Block Diagram of the PID VI with Controls Set for an MIO-16 Data Acquisition Board

The input span is -10 to 10 V, or 20 V equals 100% . The output span is 0 to 10 V, or 10 V equals 100% . PV and SP are expressed in volts. Set your I/O board for differential input to ± 10 V range, and the output to bipolar 10 V range (these are all factory defaults). If you use other settings, change the block diagram constants for the data acquisition board configuration.

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, remote setpoint (by using one of the analog inputs), and remote automatic to manual switching (by using one of the digital inputs).



Note

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

Lead-Lag

Excitation for the Lead-Lag Example VI, shown in Figure 5-11, is either a sine wave or a square wave. The waveform is synchronized to the **Cycle Time** you choose. By varying 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, almost making it disappear.

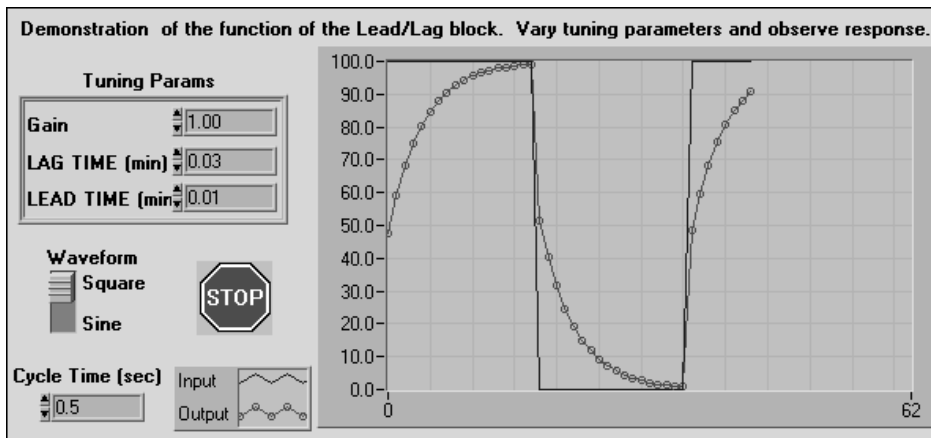


Figure 5-11. Front Panel of the Lead-Lag Example VI

References

This appendix lists sources of information you can consult if you are not familiar with process control terminology, methods, and standards.

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 is very helpful—*Single and Multiloop Control Strategies*, course number T510. 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.

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

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

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422

Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59

Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as anonymous and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

support@natinst.com

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

Fax (____) _____ Phone (____) _____

Computer brand _____ Model _____ Processor _____

Operating system (include version number) _____

Clock speed _____ MHz RAM _____ MB Display adapter _____

Mouse ___yes ___no Other adapters installed _____

Hard disk capacity _____ MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

List any error messages: _____

The following steps reproduce the problem: _____

PID Control Toolkit

Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

Hardware revision _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

National Instruments software _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *PID Control Toolkit for G Reference Manual*

Edition Date: January 1998

Part Number: 320563B-01

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

E-Mail Address _____

Phone (____) _____ Fax (____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meaning	Value
k-	kilo-	10^3
m-	milli-	10^{-3}
μ -	micro-	10^{-6}
n-	nano-	10^{-9}

Numbers/Symbols

° Degrees.

∞ Infinity.

Ω Ohms.

% Percent.

A

A Amperes.

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.

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

- C Celsius.
- cascade control Control in which the output of one controller is the setpoint for another controller.
- 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*.
- Code Interface Node (CIN) Special block diagram node through which you can link conventional, text-based code to a VI.
- controller output *See* manipulated variable.

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. Also called *rate action*.
- deviation Any departure from a desired value or expected value or pattern.
- downstream loop In a cascade, the controller whose setpoint is provided by another controller.

E

EGU Engineering units.

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.

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*.

General Purpose
Interface Bus (GPIB) The common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1987.

global variable A built-in LabVIEW object you use to easily access a given set of values throughout your LabVIEW application. A global variable is a special kind of VI with front panel controls that define the data type of the global variable.

H

Hz Hertz.

I

Instrument Society of America (ISA)	The organization that sets standards for process control instrumentation in the United States.
integral action rate	<i>See</i> 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.
I/O	Input/output.

K

K	Process gain.
kHz	Kilohertz.

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench.
lag	A lowpass filter or integrating response with respect to time.
loop cycle time	Time interval between calls to a control algorithm.

M

magnitude ratio	<i>See</i> 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 <i>controller output</i> .
MB	Megabytes of memory.
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.

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

proportional kick The response of a proportional controller to a step change in the setpoint or process variable.

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.

S

s	Seconds.
scope chart	Chart indicator modeled on the operation of an oscilloscope.
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 variable.
span	The algebraic difference between the upper and lower range values.
strip chart	A plotting indicator modeled after a paper strip chart recorder, which scrolls as it plots data.

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	<i>See</i> overshoot.

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.
virtual instrument (VI)	LabVIEW program; so called because it models the appearance and function of a physical instrument.

W

While Loop	Post-iterative-test loop structure that repeats a section of code until a condition is met. Comparable to a Do loop or a Repeat-Until loop in conventional programming languages.
------------	---

Index

A

Anti-windup, 2-4
Astom, K.J., A-1
autopid.llb, 1-1
Autotuning Algorithm, 2-5 to 2-8
 tuning formulas, 2-6
 PI control (fast), 2-7
 PI control (normal), 2-7
 PI control (slow), 2-8
 P-only control (fast), 2-6
 P-only control (normal), 2-6
 P-only control (slow), 2-7
Autotuning Wizard, 3-10 to 3-13

B

bibliography, A-1
bulletin board support, B-1
bumpless transfer, 2-4

C

calculating controller action, 2-1 to 2-4
Compatibility VI, 3-9 to 3-18
control applications, 1-2
control strategies, 3-1
controller output, 2-3
Corripio, A.B., A-1
customer communication, xi, B-1 to B-7

D

DAQ hardware with PID, 3-13 to 3-18
 event-driven DAQ control loop, 3-17
 hardware-timed DAQ control loop, 3-16
 server-timing for DAQ control, 3-18

 software-timed DAQ control loop,
 3-13 to 3-14
 software-timed DAQ control loop with
 advanced DAQ functions, 3-15
Demonstration VIs, 5-8 to 5-10
 Lead-Lag, 5-10
 PID with MIO Board, 5-8 to 5-9
designing control strategies, 3-1 to 3-6
 manual tracking techniques, 3-4
 closed-loop tuning, 3-4
 open-loop tuning, 3-5
 step test, 3-5
 ultimate gain, 3-4
 setting timing, 3-2
documentation
 conventions used in this manual, x
 organization of this manual, ix
 related documentation, x

E

electronic support services, B-1 to B-2
e-mail support, B-2
error calculation, 2-2

F

fax and telephone support, B-2
FTP support, B-1

G

gain scheduling, 2-5, 3-8 to 3-9

H

Hagglund, T., A-1
Hang C.C., A-1
Ho, W.K., A-1

I

installation
 Macintosh and Power Macintosh, 1-2
 Windows 3.x, 1-2
 Windows 95 and Windows NT, 1-1

L

Lead-Lag VI
 description, 4-11
 example, 5-10

M

Macintosh and Power Macintosh
 installation, 1-2
manual
 See documentation.

N

Nichols, N.B., A-1
nonlinear adjustment of integral action, 2-3

O

output limiting, 2-4

P

package contents, 1-1
partial derivative action, 2-3

PID (compatibility) VI
 description, 4-6 to 4-7
 using, 3-9

PID (gain schedule) VI
 description, 4-3 to 4-5
 using, 3-8

PID Algorithm, 2-1 to 2-5
 calculating controller action, 2-1 to 2-4
 controller output, 2-3
 error calculation, 2-2
 nonlinear adjustment of integral
 action, 2-3
 output limiting, 2-4
 partial derivative action, 2-3
 proportional action, 2-2
 PV filtering, 2-2
 trapezoidal integration, 2-3
 gain scheduling, 2-5

PID algorithms, 2-1 to 2-8
 Autotuning Algorithm, 2-5 to 2-8
 PID Algorithm, 2-1 to 2-5

PID software, 3-1 to 3-18
 Autotuning Wizard, 3-10 to 3-13
 Compatibility VI, 3-9 to 3-10
 gain scheduling, 3-8 to 3-9
 using PID VIs, 3-7 to 3-8
 with DAQ, 3-13 to 3-18

PID toolkit, 1-1
 applications, 1-2
 contents, 1-1
 using with DAQ, 3-13 to 3-18
 event-driven DAQ control loop, 3-17
 hardware-timed DAQ control
 loop, 3-16
 server-timing for DAQ control, 3-18
 software-timed DAQ control
 loop, 3-13 to 3-14
 software-timed DAQ control loop
 with advanced DAQ
 functions, 3-15

VIs, 4-1 to 4-12
 Lead-Lag, 4-11
 PID, 4-1 to 4-3
 PID (compatibility), 4-6 to 4-7
 PID (gain schedule), 4-3 to 4-5
 PID with autotuning, 4-7 to 4-10
 Ramp, 4-12
 PID VI
 description, 4-1 to 4-3
 using, 3-7
 PID with autotuning VI
 description, 4-7 to 4-10
 using, 3-10
 prctrl.llb, 1-1
 prctrllex.llb, 1-1
 process control examples, 5-1 to 5-10
 Demonstration VIs, 5-8 to 5-10
 Lead-Lag, 5-10
 PID with MIO Board, 5-8 to 5-9
 Simulation VIs, 5-1 to 5-7
 Cascade and Selector, 5-5 to 5-7
 General PID, 5-2 to 5-4
 Plant Simulator, 5-4 to 5-5
 Tank Level, 5-1 to 5-2
 proportional action, 2-2
 PV filtering, 2-2

R
 Ramp VI, 4-12
 references, A-1

S

server, VI
 for DAQ control, 3-18
 setpoint relay experiment, 2-5
 Shinskey, F.G., A-1

Simulation VIs, 5-1 to 5-7
 Cascade and Selector, 5-5 to 5-7
 General PID, 5-2 to 5-4
 Plant Simulator, 5-4 to 5-5
 Tank Level, 5-1 to 5-2

T

technical support, B-1 to B-2
 telephone and fax support, B-2
 timing, 3-2
 trapezoidal integration, 2-3
 tuning, 3-4
 Two Degree of Freedom PID algorithm, 2-2

U

using PID VIs, 3-7 to 3-8

V

VI server
 for DAQ control, 3-18
 VIs
 Lead-Lag VI, 4-11, 5-10
 PID (compatibility) VI, 3-9, 4-6 to 4-7
 PID (gain schedule) VI, 3-8, 4-3 to 4-5
 PID VI, 3-7, 4-1 to 4-3
 PID with autotuning VI, 3-10, 4-7 to 4-10
 Ramp VI, 4-12

W

Windows 3.x installation, 1-2
 Windows 95 and Windows NT installation, 1-1
 works cited, A-1

Z

Ziegler, J.G., A-1