Chapter 9

CONTROL

Edition 3

Contents (continued)

Contents (continued)

Introduction

This chapter describes the CONTROL class of function blocks, which provides a range of PID control Function Blocks, which when used in conjunction with INPUTS Function Blocks and OUTPUTS Function Blocks, enable control loops to be configured. Controllers which provide PID with or without self and adaptive tuning algorithms and counterparts which enable control of motorised valves are included.

More complex control strategies such as ratio and cascade control or gain scheduling may be implemented by interconnection of multiple control Function Blocks.

It is recommended that the PC3000 Control Overview in this chapter is read as an introduction to the control function provided by the PC3000. This includes information on related subjects such as cascade and ratio control and setpoint programming.

CONTROL OVERVIEW

PREFACE

Scope

This document covers the following areas of functionality:

Basic description of On/Off and PID control

Relation of the above functionality and Eurotherm's suite of Control function blocks

Basic loop configuration/ software wiring

Standard methods of manual loop tuning

Eurotherm's auto and adaptive tune mechanisms

Setpoint programming

Gain scheduling

Cascade Control

Ratio Control

Feedforward Control

Alternative Strategies

Purpose

The purpose of this document is to describe some of the standard methods and techniques used for control of processes using the PC3000 built-in control function blocks. It can be used as an initial introduction to control principles. Some basic control systems knowledge however will be necessary to understand some of the more technical sections of this document.

Introduction

Control of a process or a machine encompasses a large number of coordinated activities. One such activity which is traditionally allocated to discrete instruments, is feedback control of some of the measurable quantities of the process. These are typically variables such as temperature, pressure, flow, moisture, humidity, level, composition, etc. The PC3000 function blocks are designed so that suitable continuous control strategies for process with different response characteristics can be constructed.

In addition to the basic regulatory control there are a large number of activities within a control strategy which are concerned with the control of machinery and instrumentation. These include startups, shutdowns, activities during sensor break conditions, failsafe downgrading of the performance, checks on operation cycles to ensure that the operator is warned about routine maintenance, generation of batch reports, smooth switching between recipes, etc. In addition, the control system handles interlocks, sequence logic, synchronisation with other equipment etc. The majority of these activities are based on "if then else" conditions and may be programmed using a combination of soft wiring and sequential function charts.

A large majority of processes are basically nonlinear. In order therefore to run them optimally it is sometimes necessary to combine the style of "if then else" logic with regulatory control functions such as PID. In fact in many applications it is generally necessary to do this. Notable examples of these are startup and shutdown procedures which must inevitably exist for any control scheme. Such simple rule based control schemes can be written in the form of steps, macros and wirings with PC3000 user programs. As these activities are essentially application specific they are not addressed here. The issues associated with sequential control of processes are discussed in the PC3000 User Guide Book 2 Languages.

There are essentially two methods of continuous control: "Feedback" and "Feedforward" control. By feedback control, the variable is measured (usually referred to as process value) and compared with its desired value, the setpoint. The control decision (i.e. setting a value for output) is made as a result of this comparison.

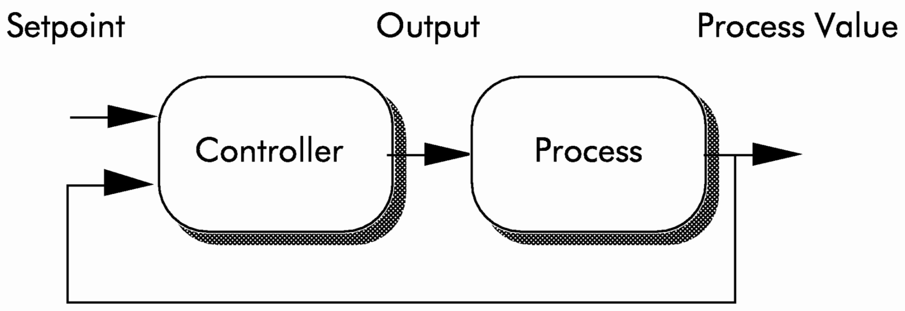


Figure 9-1 Feedback control

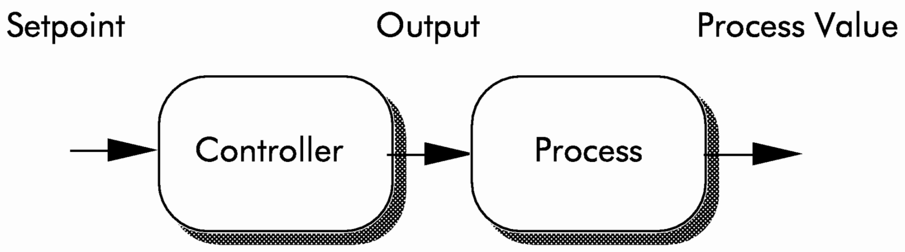


Figure 9-2 Feedforward control

Figure 9-1 shows a simple feedback controller. Consider an electric furnace. The "Process_Value" is a measurement of the temperature, and "Setpoint" the desired temperature of the furnace. "Output" could be a time-proportioned signal to a logic output or a triac. The controller is typically a Proportional, Integral, Derivative (PID) alogorithm. In feedforward control the control decision is made without measuring the Process_Value. Figure 9-2 depicts feedforward control in its most basic form. Switching an oven to full heat for a specific period of time without measuring its temperature, for preheat purposes, is a crude example of feedforward control. For tight control of process variables, frequently both strategies have to be used in tandem. Typically, feedback control requires far less process knowledge than feedforward control. The primary aim of feedback control can be summarised as:

Reducing process uncertainty (e.g. reducing temperature variability due to environmental conditions such as opening and closing doors and changes of loads in an oven)

Stabilising open-loop unstable systems (e.g. exothermic reactors).

Feedforward control on the other hand is typically used to overcome time-delays or to compensate for the effect of external influences such as control signals from other loops in the process.

TYPES OF CONTROL

On / Off control

On / Off or Bang-Bang control is the simplest form of feedback control. In its simplest form it could consist of:

```
DOP.Process_Val  :=  Setpoint.Val > ANIN.Process_Val;
```

where DOP and ANIN are instances of the respective digital output and analogue input function blocks. The control is fully "On" when the process value (measured via ANIN) is below setpoint and set to "Off" when it is equal or above setpoint. Such a control scheme will continually oscillate at a rate dependent on the gain and time scale of the underlying process. See figure 9-3 for an example of a single channel control scheme (in this case heat-only).

In order to overcome the problem of asymmetric oscillation as shown in figure 9-3, as well as possible relay chatter, a hysteresis (or dead-band) can be added. The power is switched off when the process value reaches the setpoint but is only turned on when the process value falls below the setpoint by an amount more than a user-chosen hysteresis (or dead-band). This can be realised by the following simple wiring and the use of SEL_BOOL function:

```
DOP.Process_Val := SEL_BOOL( G := DOP.Process_Val,
     IN0 := ANIN.Process_Val < Setpoint.Val - Hyst.Val,
     IN1 := ANIN.Process_Val < Setpoint.Val);
```

The result of the above algorithm is shown in figure 9-4. If there are both heat and cool channels then the algorithm can be extended to implement a setup such as that of figure 9-5.

Another class of On/ Off controllers (e.g. sometimes associated with valve position control) is the use of dead-zones. Here we no longer have any hysteresis. In a dual channel case output 1 is on when the error is large and negative, output 2 is on when the error is large and positive and finally both are off when the error is within a user-chosen zone. Figure 9-6 shows the characteristics of such an On/ Off controller.

```
DOP1.Process_Val := (Setpoint.Val - ANIN.Process_Val) > DB1.Val;
DOP2.Process_Val := (ANIN.Process_Val - Setpoint.Val) > DB2.Val;
```

The wiring for the On/ Off controller with dead-zone is:

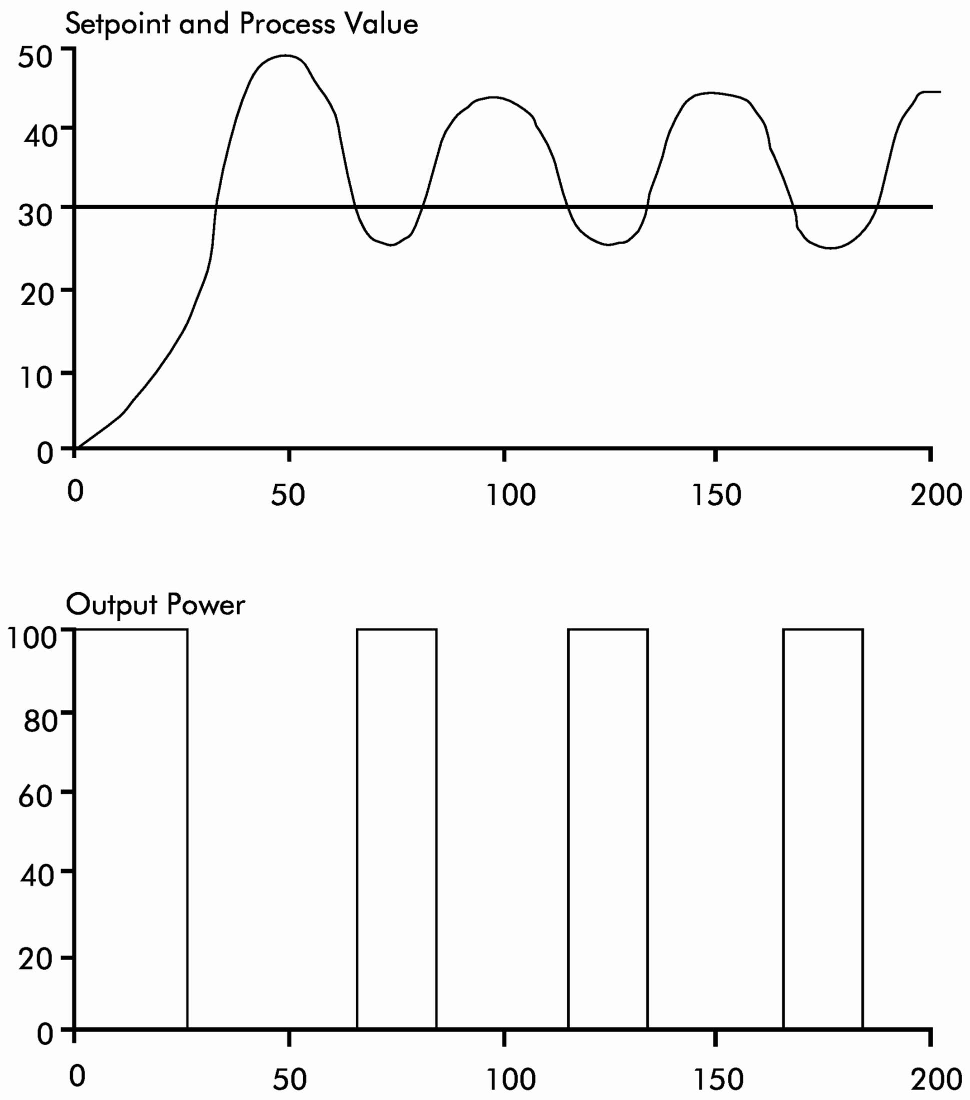


Figure 9-3 Basic on/off control

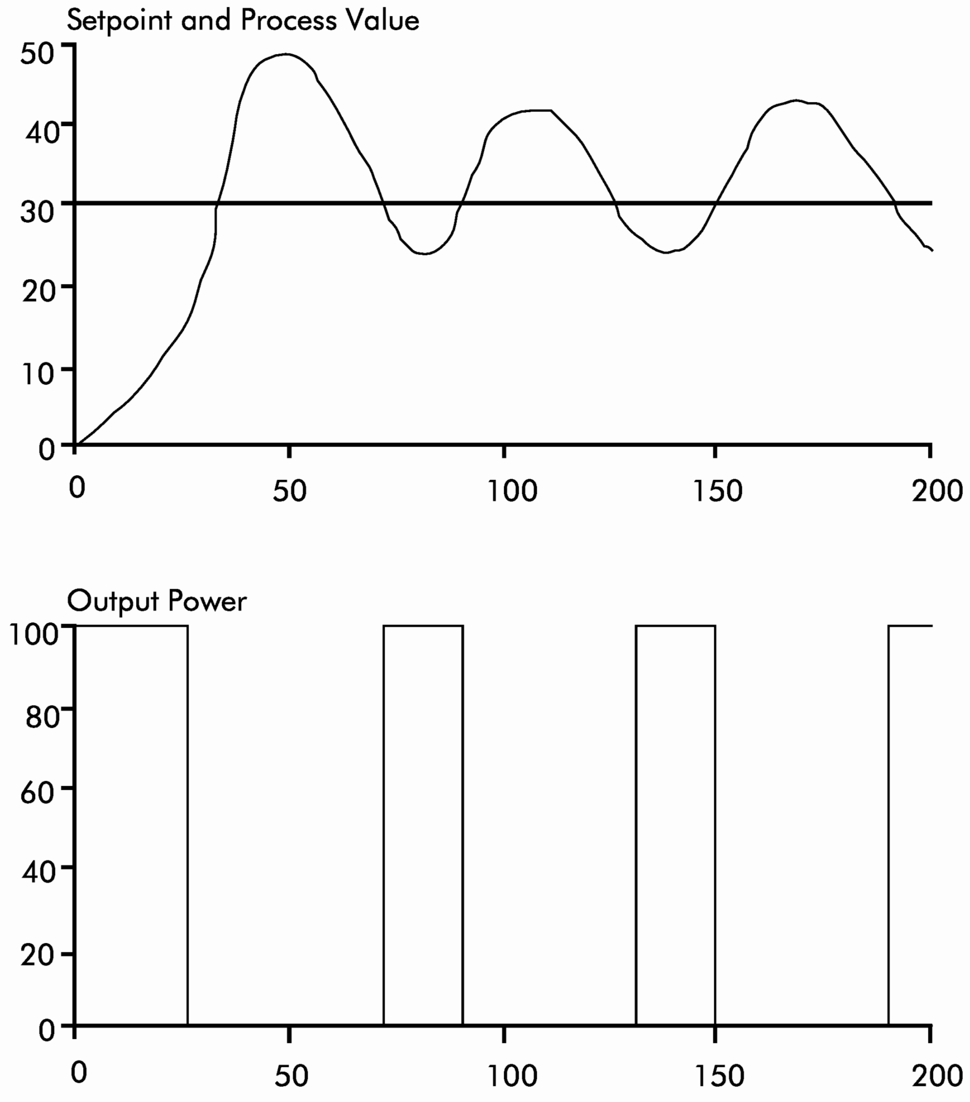


Figure 9-4 On/off control with hysteresis

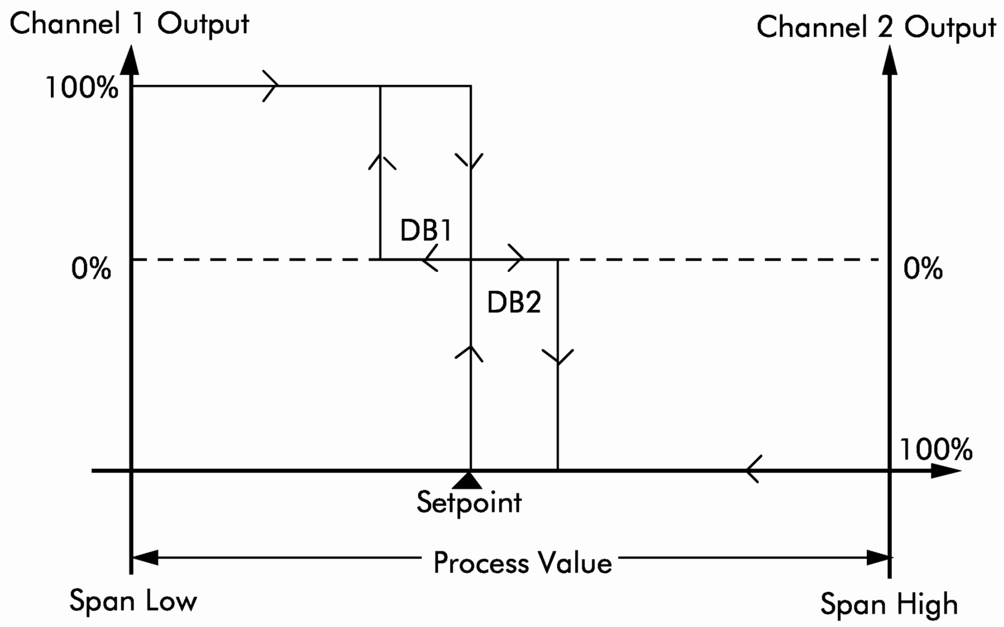


Figure 9-5 Dual channel on/off control with hysteresis

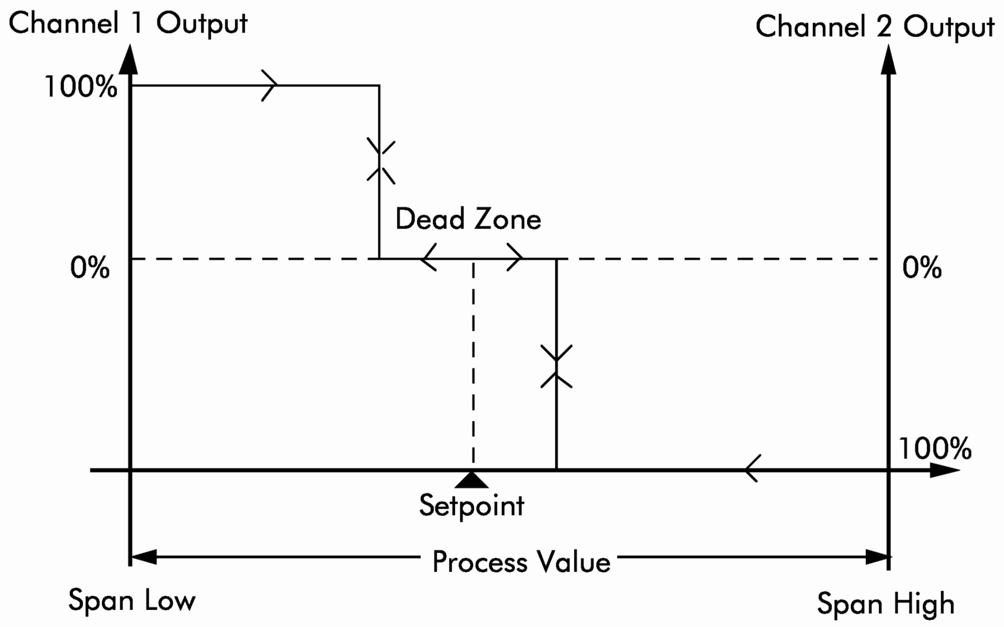


Figure 9-6 Dual channel on/off control with dead-zone

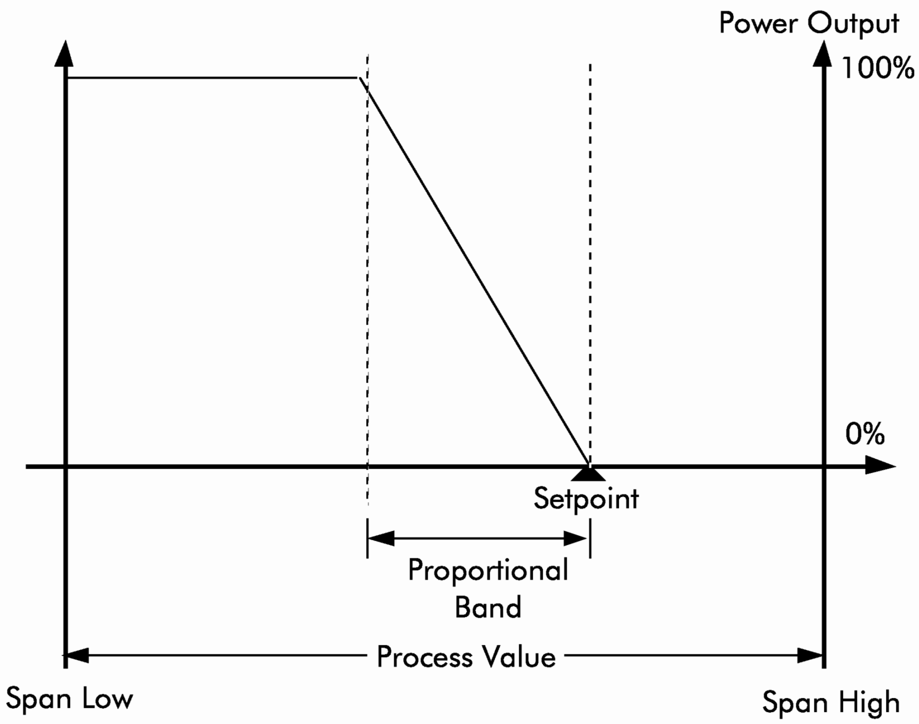


Figure 9-7 Proportional control

PID Control

Proportional control

Clearly, the main problem of On/ Off control is that the process variable never settles at the setpoint. More accurate control can be achieved if instead of applying a hysteresis or a dead-zone we apply an output proportional to the difference between the setpoint and process value. This is shown in figure 9-7. 100% output is applied for process value more than one proportional band setting below the setpoint. It is then linearly decreased from 100% to 0% as the process variable approaches the setpoint. The output is set to zero for process values larger than setpoint. If more than one output channel is used the proportional band can be extended accordingly. Usually limits are applied for the process value and the setpoint to lie between Span_Low and Span_High. There is some confusion regarding the use of the term "Proportional Band". It refers to two distinct but related quantities:

The region or band where the control is a linear function of error

The setting (i.e. width) of the linear region for control

In this document we refer to the former as the proportional band and the latter as the proportional band setting or width. In the PC3000 suite of function blocks the width is set through the parameter "Prop_Band". The proportional band width in PC3000 is set as a percentage of the span (the difference between Span_High and

Span_Low).

Consider an oven. At steady state the total heat input balances the losses. This implies that for any "lossy" oven with a proportional controller there will be an error between the setpoint and process value at steady state. The magnitude of the error is clearly a function of the size of the proportional band: the narrower the proportional band the smaller the steady-state error. The narrower the proportional band, however, the closer the controller will be to an On/Off controller and as such more oscillatory. This implies that there is a limit beyond which decreasing the proportional band setting will be detrimental to the performance of the loop.

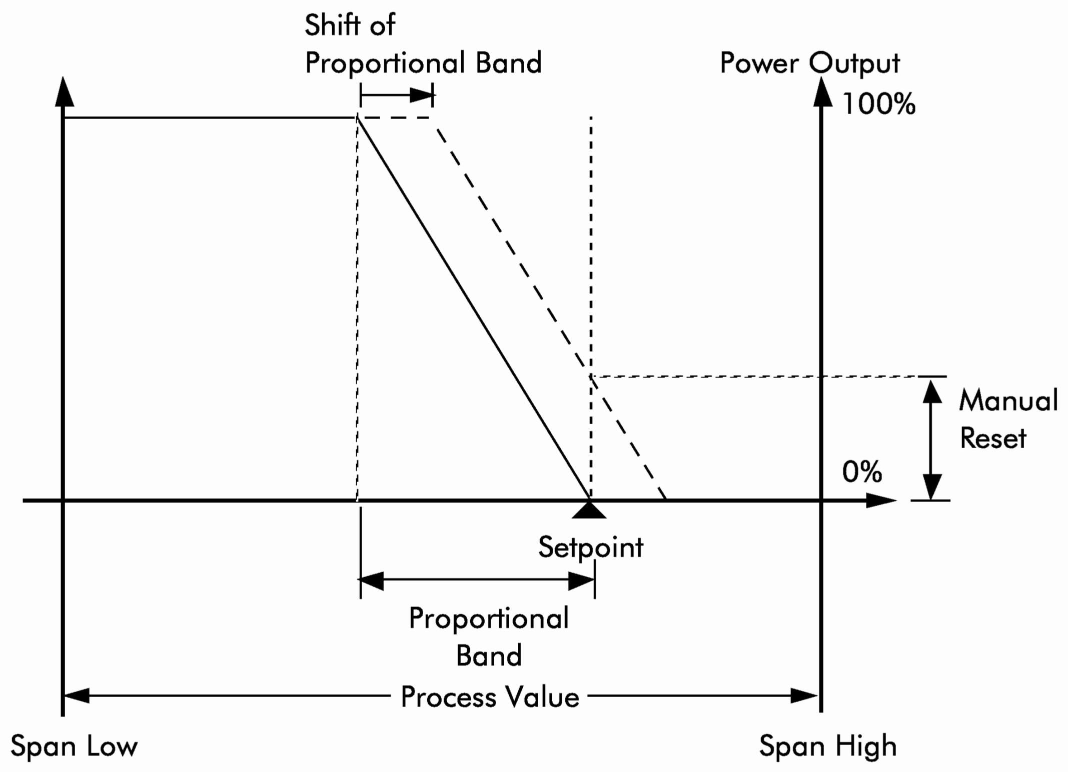


Figure 9-8 The effect of manual reset

Proportional plus integral control

In order to correct for steady state errors one of two strategies can be employed:

The function of "manual reset" can be used. Manual reset, as the name implies can be used to reset the error to zero manually. The value of manual reset is chosen as a percentage of the output span and is typically set to 50%. Manual Reset is added to the contribution of the proportional term. Setting the manual reset value can be considered graphically as shifting the proportional band sideways by MR x PB x Span / 10000 where MR is the Manual_Reset value, PB is the proportional band width in percentage of the span. Figure 9-8 shows the relationship of manual reset and the proportional band.

'Automatic Reset' or integral control can be used. With integral control the output of the controller is proportional to the integral of the error over time.

$$\text{Integral\_Out} = \frac{10000}{\text{Span x PB x } T_i} \int e(t)dt$$

$$e(t) = \text{Setpoint} - \text{Process\_Value}$$

Ti is a user-chosen value for the integral time and is traditionally defined as the time taken for the proportional band to shift by one unit for a constant unity error. The proportional band will continue to shift until such time as the error is driven to zero at steady-state. The integral term could be viewed as setting the average value of control while the proportional term reacts to short term disturbances. The value of the integral time should therefore be set commensurate with the open-loop process time constants and desired closed-loop responses. See PID Tuning for details on tuning control loops.

For proportional plus integral control the contribution of the integral and proportional control are added together. Note that manual and automatic reset are mutually exclusive. Proportional plus Integral control is the most common method for feedback control.

Proportional plus integral plus derivative control

PI control can solve most control problems. The control action in PI control however, is always retrospective. Overshoot and slow recovery from disturbances are very common for PI controllers.

Consider a process with a large thermal inertia. On a setpoint increase for a PI controller we will have the following sequence of events. The proportional part of the control will cause an initial kick in the output. The process value will begin to move very slowly towards the new setpoint. Meanwhile, the integral term will continue to integrate the error between the setpoint and process value. This will cause a subsequent gradual increase of the controller output. The most likely event is that the controller output as well as the contribution of the integral term will grow significantly larger than the required new steady-state value. The only way the control could move back towards the new steady-state value is for the error to change sign: the process value must overshoot its new target. For processes with large thermal inertia this overshoot may become unacceptable.

Derivative control provides a means for dealing with this kind of processes. Consider the error term e(t). In order to overcome the overshoot problem the contribution of the proportional term must change sign before the instantaneous error e(t) does. A simple and effective way is to introduce an anticipatory term so instead of looking at e(t) the controller must see an estimate of e(t + Td) (the prediction of the error Td seconds ahead).

To a first approximation this term is given by:

$$e(t+T_d) \approx e(t) + T_d \frac{de(t)}{dt}$$

which in effect is the linear extrapolation of the error on the basis of its current rate of change. If the proportional control operates on this predicted error the problem of overshoot will to a great extent be eliminated. This implies that the control will now become:

$$\text{Output} = \frac{10000}{\text{Span} \ \ x \ \ \text{Prop\_Band}} \left[e(t) + \frac{1}{T_i}\int e(t) + T_d \frac{de(t)}{dt}\ \right]$$

This is the standard textbook PID control law. In many applications it is customary to set a gain value K instead of a proportional band setting. The gain K is then equivalent to:

$$K = \frac{10000}{\text{Span} \ \ x \ \ \text{Prop\_Band}}$$

PID control has another beneficial effect. Consider a furnace with only a PI controller. The drop in the temperature due to opening a door can be quite rapid. If a wide proportional band setting is used however, the response of the controller can be quite slow. What is required is adjustment of the proportional band according to this rate of change. If the process value is moving rapidly away from the target the effective proportional band should be narrowed down. This will improve the recovery time of the controlled process. Derivative term in a PID provides such a mechanism for narrowing the effective proportional band.

Figure 9-9 shows the difference between PI and PID control for setpoint changes and disturbance rejection. The top two graphs are the setpoint, process value and the output power of the PI controller respectively. Note the overshoot in the initial setpoint transient. The bottom two graphs are the PID controller response with the same process. Note the much improved recovery time for the PID controller from a disturbance as well as elimination of the initial overshoot. The price paid, however, is the increased activity of the output signal.

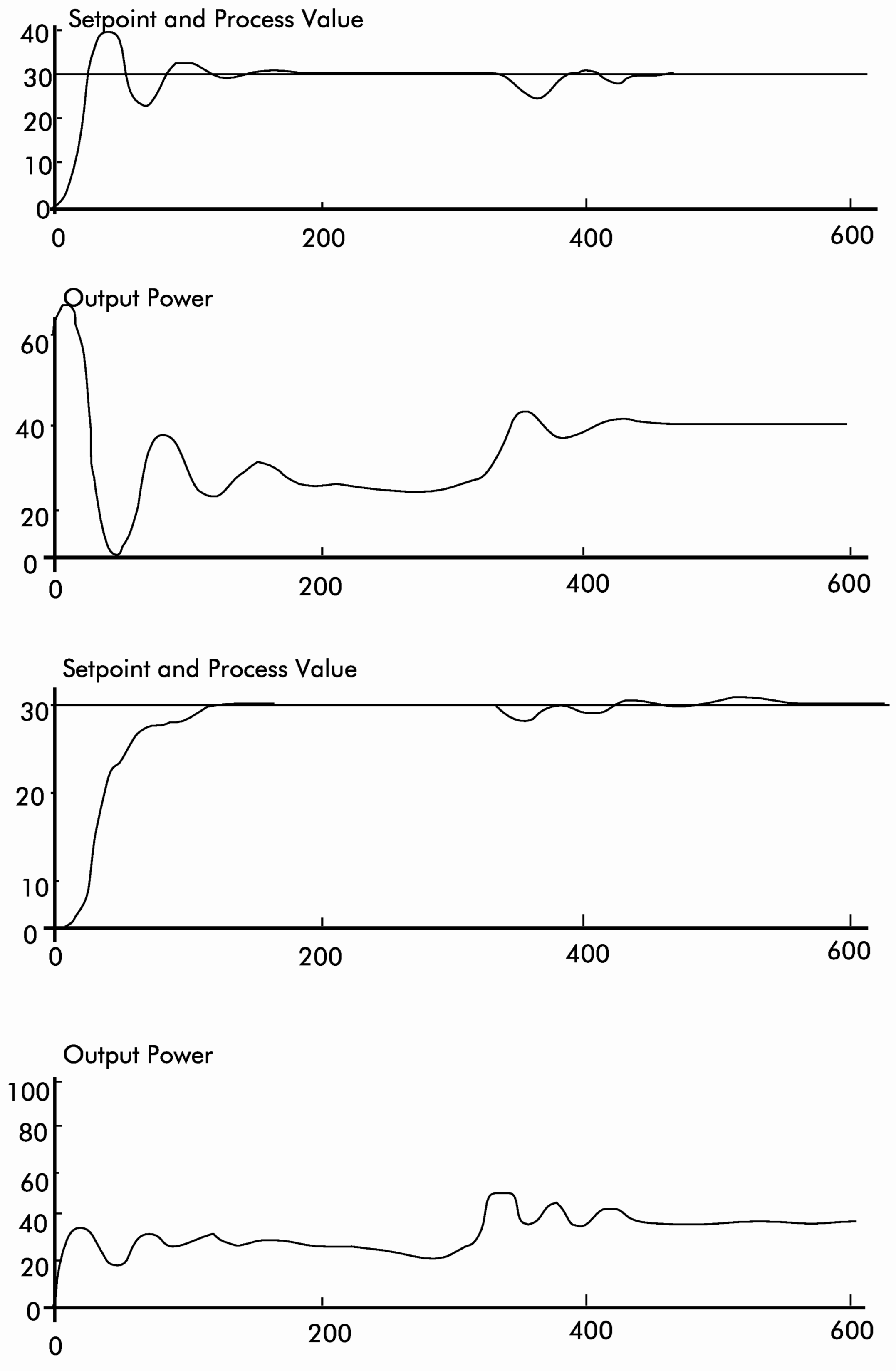


Figure 9-9   PI and PID control

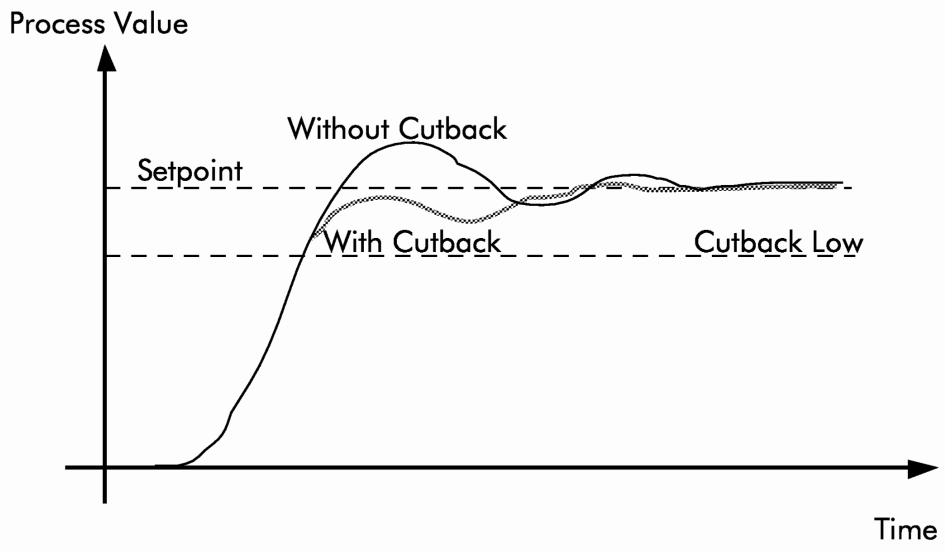


Figure 9-10 Cutback in PID control

In practice the derivative term is filtered by a first order low pass filter to overcome the detrimental effects of amplifying noise at high frequencies. In addition, in process control applications the derivative term is taken from the process value as opposed to the error signal.

The Eurotherm algorithm is a non-interacting PID. If the PID values are known for an interacting PID loop the conversion to the non-interacting version is given by

$$PB = T'_i \times PB' / (T'_i + T'_d)$$

$$Ti = T'_i + T'_d$$

$$Td = T'_i \, T'_d / (T'_i + T'_d)$$

The primes indicate the settings of the equivalent interacting PID. Note that PB and PB' either should be in engineering units or in percentage of the span. PC3000 function blocks require the proportional band as a percentage of the span.

Cutback

For most processes basic PID control or subsets thereof (P, PI or PD) are adequate provided the changes in the operating conditions are not too large. For large errors more drastic and immediate action is required. Cutback provides one such mechanism. If the process value is below (Setpoint - Cutback_Low) value maximum output power is applied and if the process value is above (Setpoint+Cutback_Low) minimum output power is applied. Consider figure 9-10. If the overshoot is due to control saturation then cutback may be able to compensate for this overshoot and give a response as shown in figure 9-10. As the process value crosses this so-called cutback band the control reverts to standard PID bumplessly. Judicious choice of cutback values can improve large setpoint response and disturbance recovery of the controlled system.

The cutback values can be set manually or by Autotune in the case PID_Auto and VP_Auto function blocks. A reasonable tuning strategy is to

> Set high and low cutback values to zero (off);
>
> When you are satisfied with the response to small disturbances (those which do not require maximum/ minimum output levels) set cutback parameters to one proportional band;
>
> Measure the response to large changes of setpoint and decide whether overshoot performance is satisfactory;
>
> Change the cutback parameter setting to correct the overshoot performance, and check the result. The setpoint change used to measure overshoot should be considerably greater than the cutback setting. The change in the cutback setting should be approximately equal to the increase or decrease of overshoot required. Typically, decreasing the cutback parameter will increase overshoot (and decrease warm-up time) and vice versa. If cutback is set too wide it will be totally ineffective. A value of zero also means off.

Cutback is sometimes considered as the point where the controller begins to cut the output power back from the maximum value. This however will only happen if the rate of change of process value is within a specified range. Consider the case of approaching the cutback region from below. While the process value is smaller than the setpoint minus the cutback low value, the output power sits at maximum. This is sometimes expressed as the proportional band being clamped at the low cutback. As the process value crosses the low cutback point the control is reverted to standard PID. The controller will only begin to reduce the power if the rate of increase of the control signal due to the integral action is balanced by the rate of decrease due to proportional and derivative terms. Broadly speaking this means that the controller will begin to cut the power back at the cutback point only if the rate of increase of process value is greater than the cutback value divided by integral time. This in turn means that the rate of rise of measured value must be faster than the movement of the proportional band.

Consider a process which overshoots on large setpoint changes. If for small setpoint changes the process response is acceptable but not for large ones it is probably because the controller does not cut the power back fast enough. Here cutback can be quite useful. Set cutback to the proportional band setting in engineering units. Measure the overshoot, if any, of the process on a similar size setpoint change. Also take note of the maximum rate of change of process value. Set the new value of cutback to the old one plus the overshoot. If this value divided by the integral time is still smaller than the maximum rate of change, cutback will eliminate overshoot. If not, set the cutback value to integral time times the maximum rate of change and try to improve the response by using derivative action if possible.

In many application areas the designer has to perform a compromise between the startup speed and overshoot. Here also cutback can be used effectively to improve the response. The designer can perform a simple compromise between the degree of overshoot and startup speed.

Cutback therefore is a primary tool for control with large signal deviations whereas the basic PID settings are used for the response of the controller to small signal variations.

Dual channel PID

Consider barrel zones of an extruder. Heating is performed via electric heaters and cooling via cooling fans. The signals for heating and cooling appear on two separate channels. Note that typically the heating and cooling gains are different. This can be compensated for by using the function of relative channel 2 gain. It may also be desirable to either; have a dead-zone between turning channel 1 off and turning channel 2 on; or turn channel 2 on before turning channel 1 off completely.

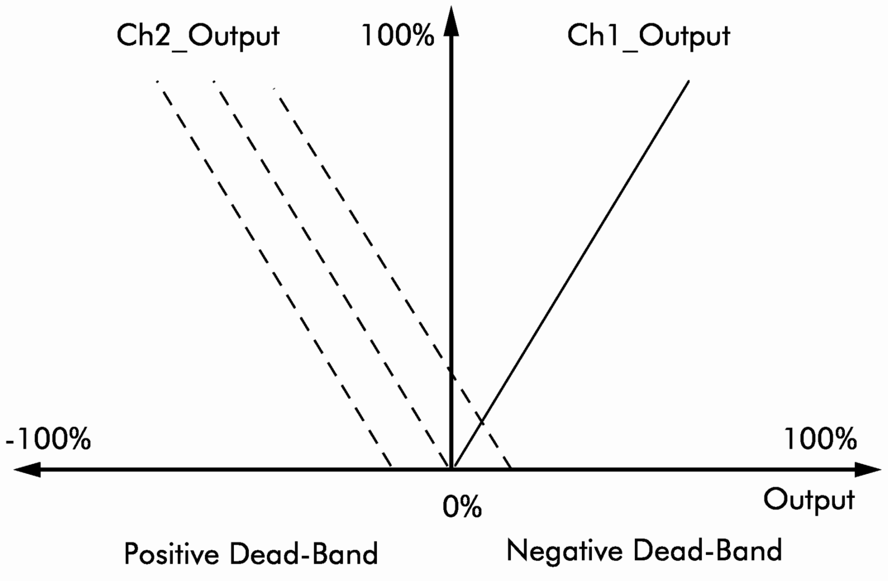


Figure 9-11 Graphical relationship of channel 2 gains and dead-zones

Both of these functions can be catered for via the channel-1-channel-2 dead-band facility. Conceptually then the outputs of the dual channel PID are:

$$\text{Ch1\_Output} = \text{Output}$$
$$\text{Ch2\_Output} = \text{Rel\_Ch2\_Gain} \times (\text{Output} + \text{Ch1\_Ch2\_D\_B})$$

Of course only positive values are valid for any of the channel outputs. Figure 9-

11 shows the graphical relationship of the output of the basic PID law and the output of the separate channels in a PID. See the PID function block description for more details. The dual channel PID is also useful when using split level valves with chemical batch reactors. The dual channel PID is selected by setting Output_low to some negative value (typically -100%).

Extensions to the basic PID

There are a number of occasions when the standard PID as described in the previous sections is not suitable and minor modifications are necessary to improve its response. Some of these are described below.

Modes of control

In order to set integral or derivative terms off the Integral and Derivative times should be set to zero. So P-only can be achieved by setting Ti and Td to zero, PD by setting Ti to zero, and PI by setting Td to zero.

Rate limits

It is quite common to put maximum rate limits on the output of the function block. This is typically used to give smoother variation of the control signals, or to take into account the fact that some of the elements within the control loop (e.g. valves) are slew rate limited.

Proportional kick

On stepwise change of the setpoint the standard PID will give a kick in the control signal proportional to the size of the instantaneous error, and inversely proportional to the magnitude of the proportional band setting. In many situations this type of response may not be suitable. If "Debump" is set at the same time as the setpoint change then the PID will respond smoothly to a setpoint change. This may be actioned within a step of the SFC.

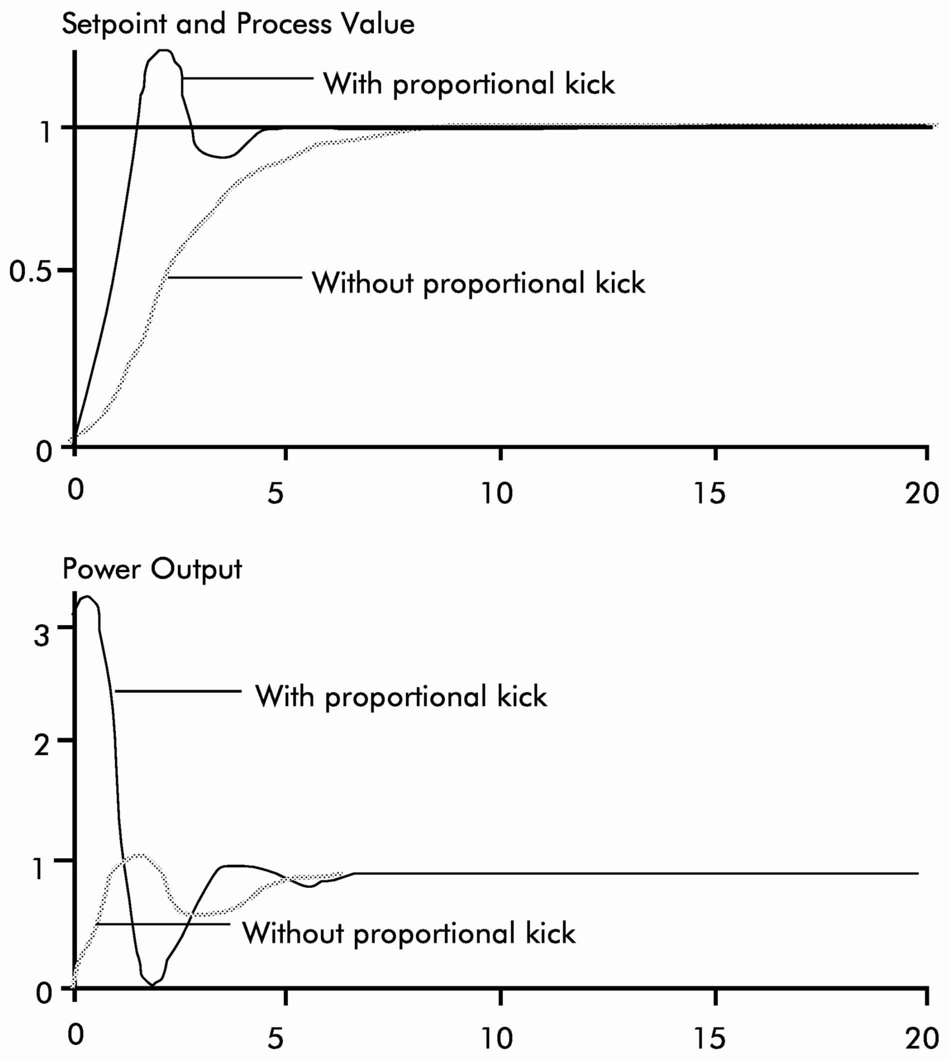


Figure 9-12 PID with and without proportional kick

```
Pid.Debump : = 1 (* Yes *);
Pid.Setpoint : = NewSP.Val;
```

This ensures that the proportional action is on the process value alone. This method only works if the instantaneous error is no more than two proportional band widths in engineering units. For larger setpoint changes the function of output rate limit could be employed whereby the maximum rate of change of output can be specified and output rate limit is enabled before the setpoint change is performed. Figure 9-12 shows a PI controller with and without the proportional kick. The PI with proportional kick responds faster to a setpoint change, but note that the output power `jumps' instantaneously. When the proportional kick is eliminated the power ramps gently towards its new value. The settling time in the latter case is much longer. It is also possible to use the function of feedforward to get various forms of PID.

Derivative kick

In addition to the proportional kick described above the derivative acting on the error will cause a kick due to derivative action on setpoint changes. To eliminate this the derivative can be set to operate on process value instead of the error. This is done by setting the Deriv_On_PV input of the PID function block. Figure 9-13 shows the case of PID control with and without the derivative acting on the error. In the case where the derivative is on the error the power output is set to a large value (outside the scale of the graph) for a very brief period. This results in faster startup and also less overshoot. The power output in the case where the derivative acts on the process value, only exhibits the initial proportional kick and is smooth there after.

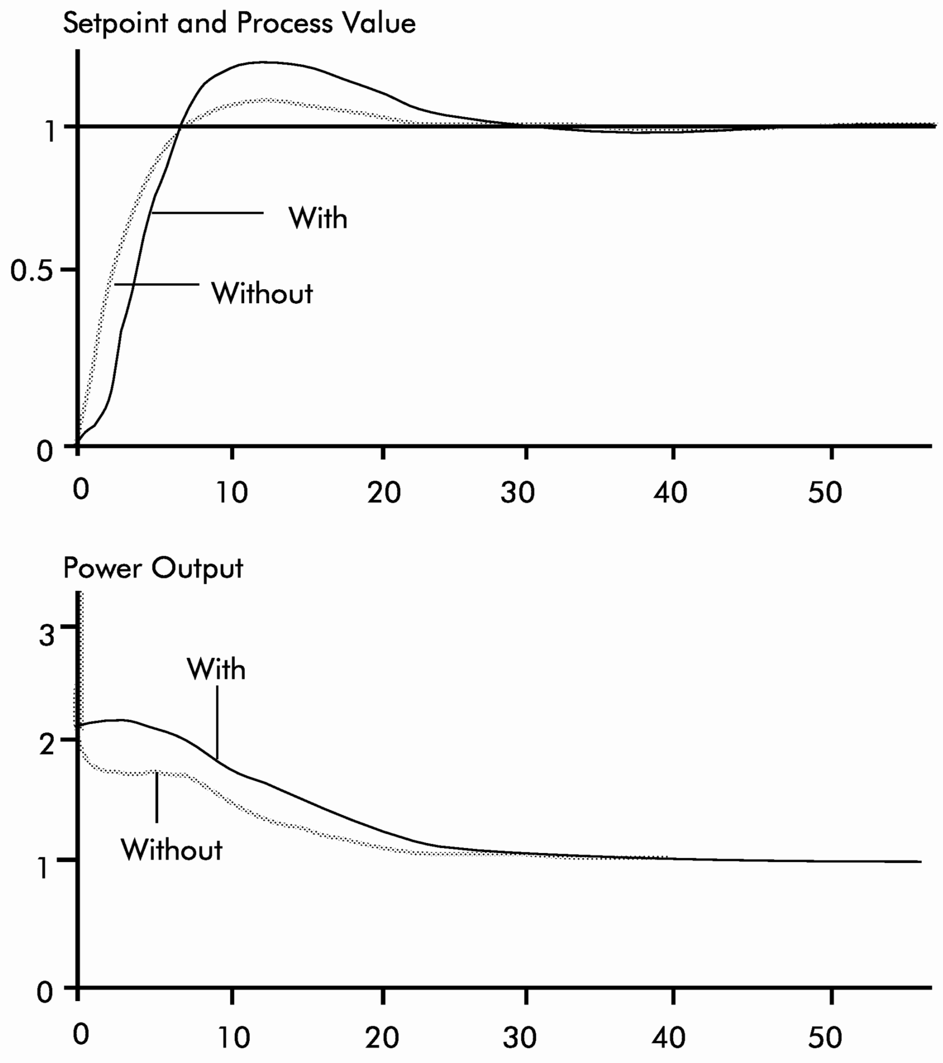


Figure 9-13 PID with and without derivative kick

Setpoint tracking

It is common, especially in some cascade loops to have the setpoint track process value in manual. This can of course be done via an external selection block. The selection is given by:

```
Pid.Setpoint :=       SEL_REAL(G:= Pid.Manual, IN0:= SP.Val,
                                 IN1:= Pid.Process_Val);
```

SP.Val is set to Pid.Setpoint in manual.

This is important when changing between AUTO and MANUAL control modes. The change-over is always bumpless as is normally required for cascade loops.

Direct and reverse acting control

Feedback is used to restore the process value to the setpoint in the event of external disturbances or changes of setpoint. In the case of a gas furnace a drop in the temperature due to a disturbance is counteracted by an increase in the gas (i.e. reverse acting control). Controlling the moisture in a rotary drier using the gas as the manipulated variable, or controlling the temperature in cryogenic applications by manipulating the flow of liquid nitrogen or helium on the other hand requires a direct acting controller whereby a reduction in the process value is counteracted by a decrease in the output signal. In PC3000 function blocks the boolean input Direct allows a switchover from reverse acting to direct acting control.

Sensor breaks

With a large class of low level (millivolt measurements) inputs, the hardware is capable of indicating sensor break conditions. In addition it is possible to postulate soft sensor break conditions by checking process values against limits. In either case, the PID has to be disabled. The function block has a boolean input Sensor_Break and the default output of the PID (i.e. safe value of output for such conditions) is set by Break_Output. Clearly, more sofisticated strategies can be applied in the user program depending on the severity of the sensor failure.

VALVE POSITIONERS

Frequently, a constant-speed reversible motor is used to drive a valve or a lever. The valve may deliver gas / combustion air to some gas burners in a furnace. There are essentially three possible states for the controller: sending a "Raise"pulse, a "Lower" pulse or no pulse at all. The main control requirement in the case of a gas furnace for example is accurate temperature control. There are several ways of dealing with this depending on the availability and reliability of the valve position signals from potentiometers.

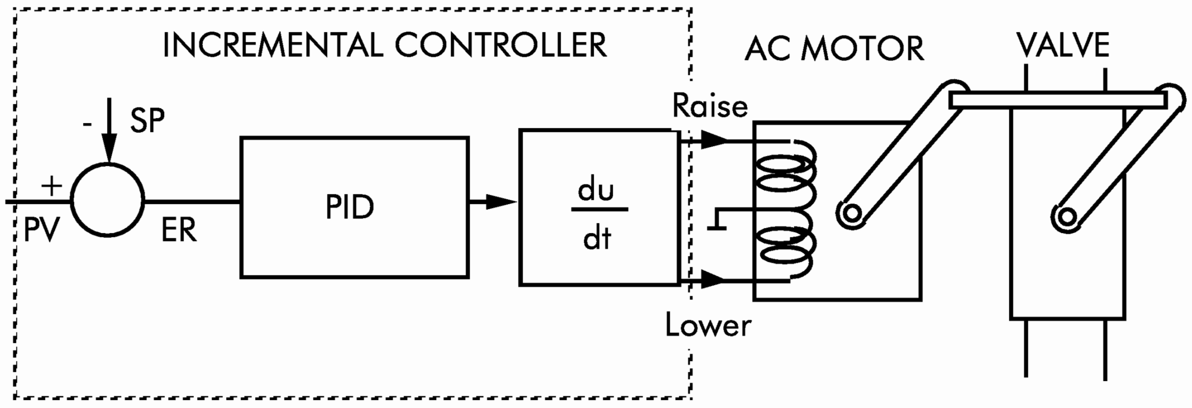


Figure 9-14 Notional view of a valve positioning algorithm

Valve positioning without potentiometer feedback

In cases where no potentiometer signal is available the PC3000 VP or its auto tune version can be used. Figure 9-14 shows the basic operation of a valve positioner.

In addition to the usual PID settings the following characteristics of the valve must also be incorporated.

Valve travel time

This is the time taken for the valve to travel from one end stop to the other. In many instances the time may be different for the travel times from fully closed to fully open as opposed to fully open to fully closed. In these circumstances the average value is adopted.

Minimum on time

This is the minimum time the controller will stay in the same state of opening the valve, closing the valve or keeping it stationary. This is because many valves have backlash and minimum on time is the length of time a pulse has to persist for the valve to react to the incoming signal from the controller.

Update time

This is the interval between updates of the update filter of the valve positioner as shown in figure 9-15. See PID Tuning which describes the method of tuning these values for the particular application.

Valve positioning with unreliable potentiometer feedback

There are facilities within the VP for use of potentiometer feedback signals. Typically the pot feedback signals are not terribly reliable. Firstly, they tend to be rather noisy and secondly it is not uncommon to lose the signals altogether. For these reasons the VP algorithm uses these as indicators of the position and they are used for setting hard endstop limits (Pot_Limit_Hi and Pot_Limit_Lo) for the valve as opposed to true position control.

Valve positioning with reliable potentiometer feedback

If the valve position signal is known to be available and reliable then it could be used for proper feedback control. A simple cascade structure that uses a basic On/Off control with dead-zone and hysteresis can be used for the position feedback stage and a standard PID can provide the necessary setpoint for this position loop.

Figure 9-16 shows one such setup. Such a structure converts the integration properties of the valve to something which behaves more like a lag, thus improving the performance of the control loop.

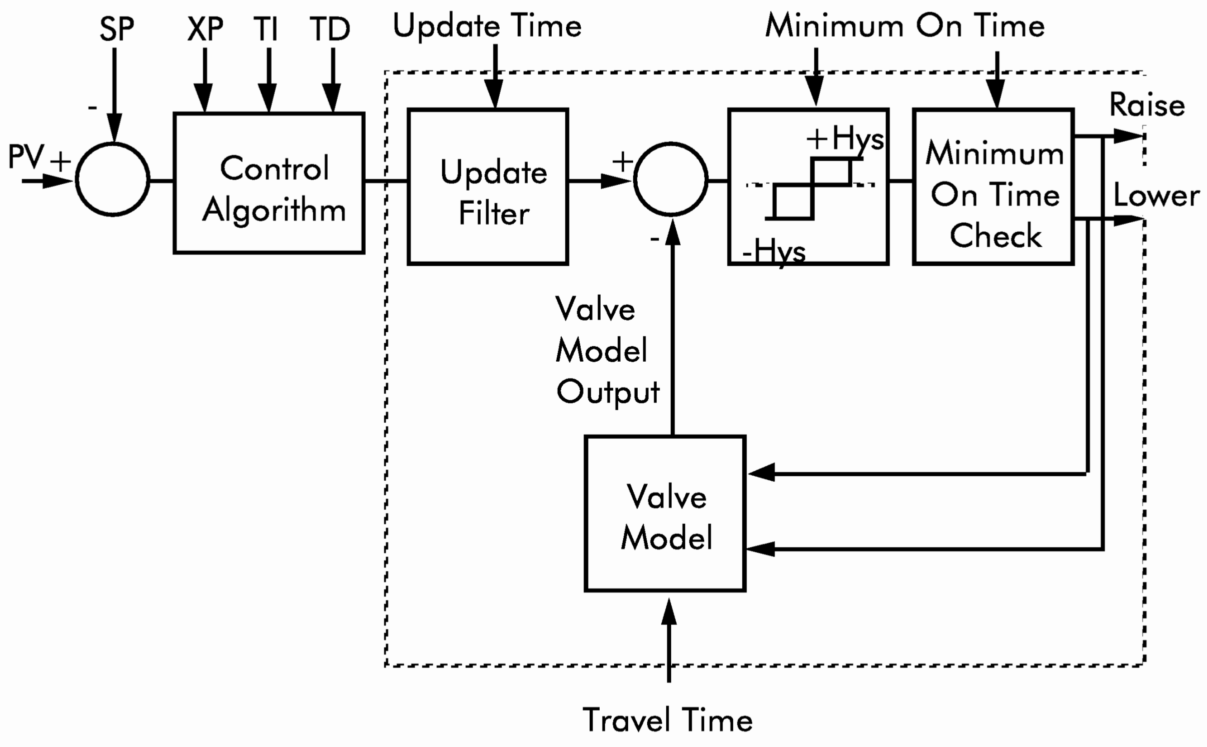


Figure 9-15 Basic incremental valve positioner block diagram

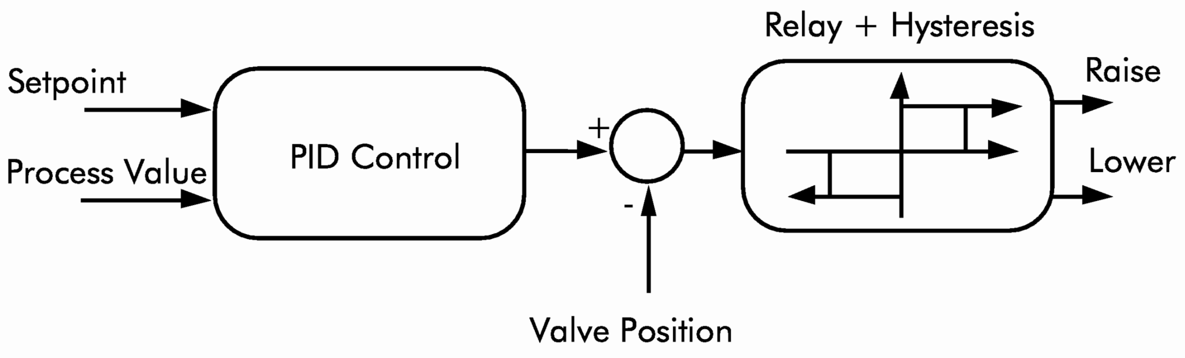


Figure 9-16 Cascade valve positioner

INPUTS, OUTPUTS AND CONTROL LOOPS

Input signal conditioning

The analogue signals measured by PC3000 for control are processed in the following way:

With AIM2/AIM4 there is an analogue first order low pass filter with a cut-off frequency at 1.6 Hz (100ms time constant), as well as a 4 sample rolling average filter. The four sample rolling average is tied to the analogue task. There is also a discrete-time first order low pass filter which the user can select its time constant. The choice of the low pass filter time constant is discussed in the section on tuning. The necessary calculations for cold junction compensation and linearizations are performed within the module for a variety of thermocouples and/or resistance thermometer inputs.

With the AIO8 the low pass filter is a second order butterworth filter with cut-off frequencies selectable to be at 160Hz, 80Hz, 32Hz and 16Hz. There is no rolling average filtering. There is no user-selectable filter as part of the function block either.

For signals measured via communications links, and those computed inferentially where there is no filtering. Also there is no spike filtering of any kind on the incoming signals. These could be done via user programs. Spike filtering could prove to be quite useful. There are a variety of methods but a simple scheme could use the Rate_Limit function block.

Input Linearisation

Most of the linearisation is performed in the module itself. There is the facility to scale and offset measurements before and after the linearisation curves for the thermocouples. There is a square root extraction also available as a part of the linearisation. This allows measurements such as pressure from a differential pressure cell to be converted and calibrated for flow control. Other conversions such as inferring relative humidity measurements from wet and dry temperatures according to BS 4883, must be performed via the user program.

Results of polynomial fits can be easily used in a simple wiring statement. For example, if

$$\text{Inferred\_Var} = a\text{x}PV^2 + b\text{x}PV + c$$

then the simple wiring statement in PC3000 can be performed whereby

Inferred.Val := (a.Val * ANIN.Process_Val + b.Val) * ANIN.Process_Val + c.Val;

Clearly if other functions such as exponentials, square roots or logarithms are also used then these can be incorporated in the linearisation equations.

Another method of linearisation is via linear interpolation between several given points. Consider the three segments as shown in figure 9-17

The linearisation can be done via the wiring shown below

```
Y.Val := SEL_REAL(G  := (X.Val >= X1.Val) AND (X.Val < X2.Val),
IN0:= SEL_REAL(G  := (X.Val >= X2.Val) AND (X.Val < X3.Val),
IN0:= SEL_REAL(G  := (X.Val >= X3.Val) AND (X.Val < X4.Val),
IN0:= Y4.Val,
IN1:= Y3.Val +(X.Val-X3.Val)*(Y4.Val-Y3.Val)/(X4.Val-X3.Val))
IN1:= Y2.Val+(X.Val-X2.Val)*(Y3.Val-Y2.Val)/(X3.Val-X2.Val)),
     IN1:= Y1.Val +(X.Val-X1.Val)*(Y2.Val-Y1.Val)/(X2.Val-X1.Val));
```

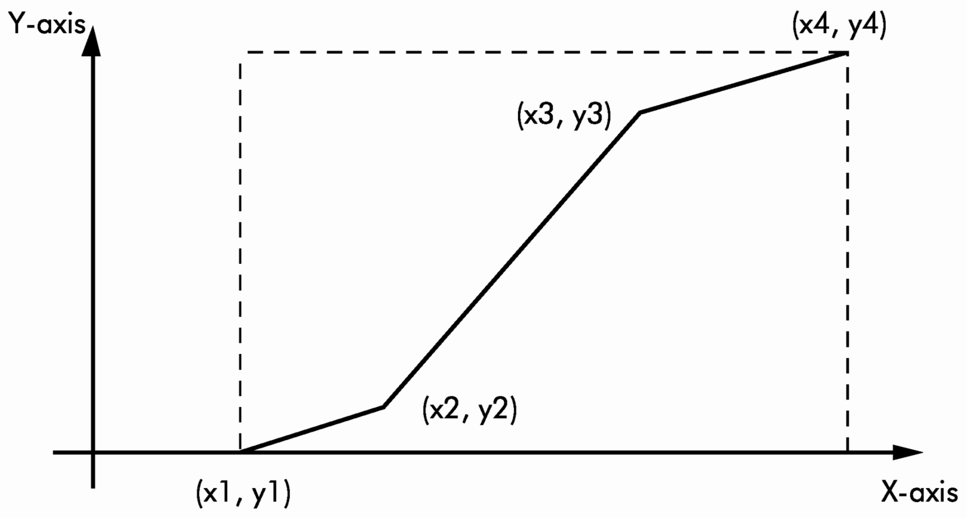


Figure 9-17   Input characteristics

Outputs

Typically, the output of the PID function block is wired to

Analogue Output Module

Time Proportioning Output

Remote/Slave variable

Another internal function block

The analogue output module can deliver a variety output ranges as well as the choice between voltages and currents.

With the majority of temperature control applications time proportioning outputs are used. Conceptually, this is as shown in figure 9-18. The choice of cycle times is described as part of tuning the PID.

Remote/Slave variables are typically used when outstations such as 900 series of discrete instruments are used in either a local control mode or as retransmitting stations. This mode provides both integrity and flexibility through the use of redundancy in the control system design.

Other internal function blocks are typically used when configuring cascade or multi-loop type control strategies.

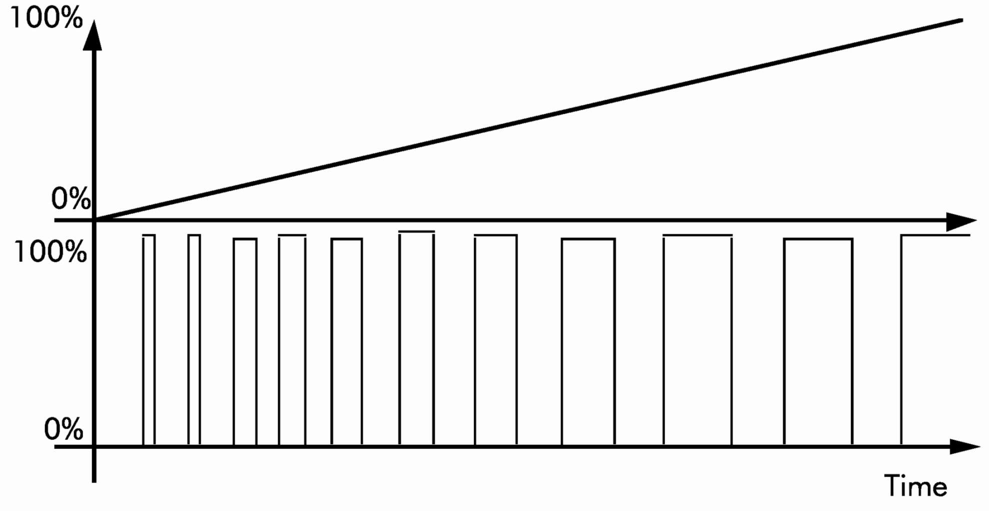


Figure 9-18 Time proportioning outputs

Control loops

The simplest method of building a PID control loop is through having an instance of

PID function block

Analogue Input function block

Analogue Output function block.

and the following software wiring

```
Pid.Process_Val   := ANIN.Process_Val;
ANOUT.Process_Val := Pid.Ch1_Output;
```

for a single channel PID. If dual channel is necessary then clearly the second output is wired to another analogue output. The ranges of the output are selected within the output function block.

For time proportioning outputs cycle-times will have to be set as well. In case of PID_Auto (auto tuning version of the PID) the cycle-times can be wired to the values set by the PID_Auto block if desired. Time-proportioning outputs used in conjunction with nonlinear cooling (water-cooling) of plastic extruders are selected via the Delin_Type input of the function block.

WARNING!

In many applications it is important to include an independent protection mechanism. The best form of protection is a completely independent 'Policeman'. This is a separate overtemperature alarm with its own thermocouple or sensor, and, on alarm will pull out the main contactor or shut off the valve to ensure plant safety. The normal function of the 'Policeman' is to act as an overtemperature alarm forming part of the overall process protection strategy. As such it is essential that all elements of the alarm system be regularly checked to ensure that they are in full working order.

Startup and shutdown

The PID function block performs a debump on its first execution. This implies that the initial output power will be set at zero (assuming a cold start) even if the error between the cold start setpoint and process value is large. The power will then gradually increase at a rate dependent on the settings of the controller and the dynamics of the process itself. This could take an inordinate length of time especially if integral time is set to a large value.

If the initial conditions of the process are known then it is best to start the PID in manual mode. The user program is then at liberty to set the initial output power. The controller could then be set to automatic. The transfer will be bumpless and PID will start operating from a known state. If the initial state is not known the function of cutback can be used. Another possibility is to startup under proportional only control and switchover to PI soon after the startup.

A similar situation arises when for example the machine under control is switched off, or the actuators are disconnected from the controller but the PID is left in automatic mode. As the controller behaves as if it is in control this may result in applying full power to keep the process value at setpoint. However, since the actuator is no longer connected, the process value will coast towards its 'ambient' value. As the machine is switched back on or the actuator reconnected the controller will immediately apply full power. This could have disasterous consequences. Again cutback could prove helpful here. The best solution, however, is to ensure that the controller is put into manual or track mode as soon as the actuator is disconnected. After reconnection as with the initial startup condition the power level should be set to some safe value and only then the controller set to automatic again.

In most cases no special precaution for startups and shutdowns needs to be taken. Most temperature loops will compensate for there situations. However, handling startup and shutdown conditions is a function of the application and as such should be handled in user programs. Relying on PID function block to deal with the initial conditions on its own may have unexpected consequences.

Valve positioners

The input signals for a valve positioning control scheme are

The primary process value (typically a temperature);

The potentiometer position feedback. This feedback signal is derived from a potentiometer. A standard analogue input channel can be used for the potentiometer position measurement. The power for this could either be supplied externally or from an analogue output (if one is available). It is also relatively easy to set up a simple circuit with a pull up resistor so that sensor break conditions can be detected in software.

The output of the controller is via any digital output (relay or logic).

PID TUNING

The default PID values of the PC3000 function blocks will probably work well for most cases involving extruders, or medium size furnaces. In the more general case however optimum machine performance is obtained only if the control loops are well tuned. The reader is referred to various control textbooks mentioned at the end of this document for loop tuning details. Here we concentrate on some general guidelines which are applicable to most cases.

Manual tuning

The most common method of tuning PID controllers is trial and error. Through the use of a few simple rules of thumb, however, it is possible to adjust the PID parameters to quite reasonable accuracy for most processes. It must be noted that the accuracy of control is to a large extent a function of transducer quality, actuation power, and open-loop dynamics of the process as it is a function of the settings of the PID controller. This is why it is important to pay a reasonable attention to the characteristics of the process under control when choosing tuning parameters for the controller.

The reaction curve method

A large majority of processes are open-loop stable. This means that if left alone the process value will stabilise at some value instead of steadily drifting to one of the limits. There are exceptions to this. An exothermic batch reactor is one such example: it requires tight closed-loop control as soon as the reaction gets going. This is because as the reaction gets going, it begins to generate excess heat which has to be taken away by active cooling. If left alone, this reaction heat will continue to heat up the reactants thus speeding the reaction up and increasing the temperatures further. Such a process is open-loop unstable. The tuning method described in this section is only applicable to open-loop stable processes.

The reaction curve method is as follows. If the process is under automatic control set the controller to manual and wait until the process value stabilises. Apply a step change of a few percent (usually about 10%) to the controller output, and examine the reaction curve of the process. The size of the output step has to be large enough so that its effect is distinguishable from normal noise and not too large so that the nonlinear effects are not encountered. When the process value has settled at its new value change the output to its original value. Examine the reaction of the process to this reverse step. Typically one of the following types of responses could be expected.

Single Lag plus Dead-Time

Multiple Lags plus Dead-Time

Nonminimum Phase Behaviour

Oscillatory Behaviour

Dead-time is the period of time where there is no significant change in the process value after changing the output of the controller. The four examples are shown in figure 9-19.

The first two occur frequently in practice with industrial processes. Nonminimum phase behaviour is one where the process initially reacts in the opposite direction, the shrink and swell effect in boilers is a typical example of nonminimum phase behaviour. Some rotary driers also exhibit this behaviour during moisture control. The fourth, oscillatory behaviour is usually seen with either mechanical systems or with closed-loop behaviour of the slave loops in cascade structures. If the cycle time of the mechanical oscillation is less than 100 times the PID function block task cycle time, damping of the oscillations through PID tuning is not possible. For example, for a PID function block running in a 100ms task, oscillations with a cycle time of less than 10 seconds cannot be controlled.

Oscillatory behaviour can be due to the damping of the inner loop of a cascade loop being too low in which case the inner loop should be retuned first to stabilise the behaviour.

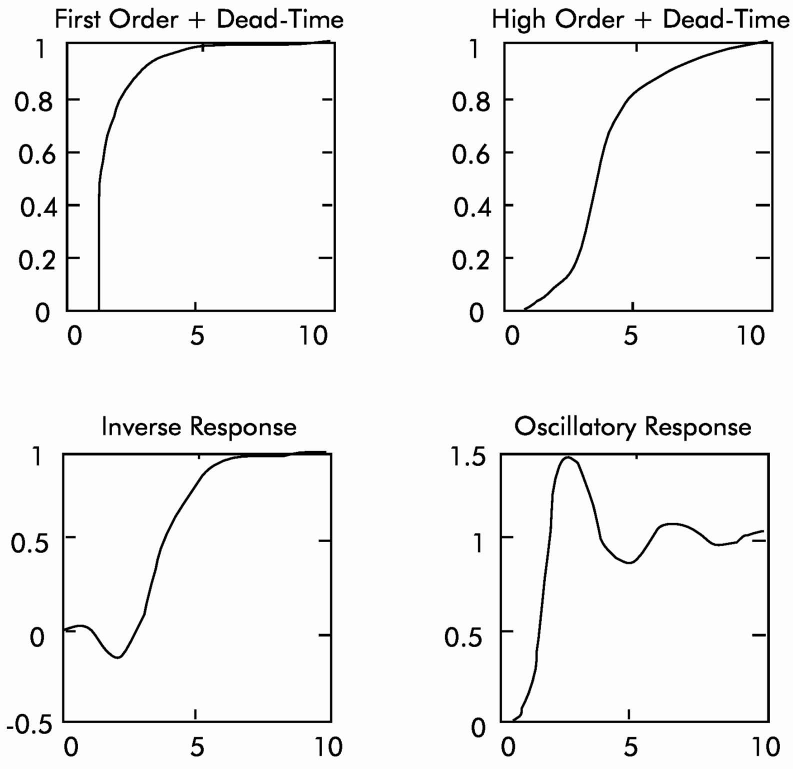


Figure 9-19 Some typical reaction curves

Three quantities are computed from the reaction curve, see figure 9-20.

Dead-Time (Dp): is the estimated dead time.

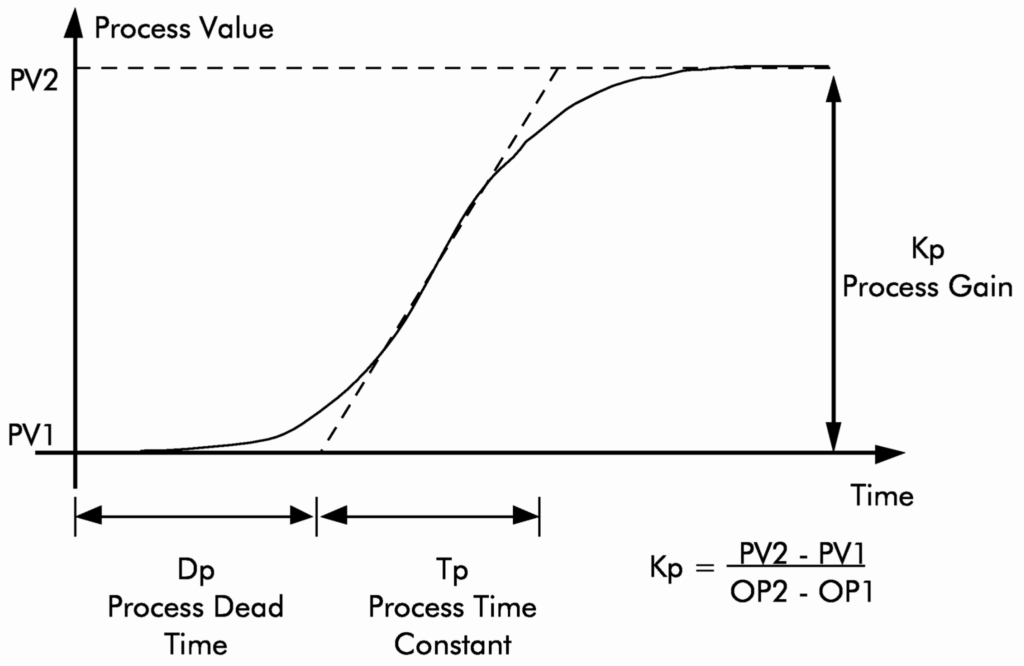


Figure 9-20 Gain, Time Constant and Dead Time Computation from Reaction Curve

Controller type	Prop_Band	Integral	Derivative
P	100KpNd		
PI	110KpNd	3Dp	
PID	80KpNd	2Dp	Dp/2

Table 9-1 PID settings from a reaction curve

Controller type	Prop_Band	Integral	Derivative
P	2Pu		
PI	2.2Pu	0.8Tu	
PID	1.67Pu	0.5Tu	0.12Tu

Table 9-2 Ziegler-Nichols rules for tuning

Time Constant (Tp): This is the estimated time constant or reaction time of the process. See figure 9-20. This is the time from the intercept of the asymptote with the time axis and its intercept with the new output level.

Process Gain (Kp): This is the ratio of the change in the process value steady-states to the change in controller output.

The normalised dead-time is then given by

$$N_d = \frac{Dp}{Tp}$$

There are a variety of design rules give the open-loop reaction curve of the process, the simplest forms are given in table 9-1. Note that the proportional band settings in table 9- 1 are in engineering units.

There are corrections to the table for the PID values given by Cohen and Coon which deal with the case of delay dominant systems a lot better. Shinskey also gives PID values for various ratios of time constants to dead-times. His calculations give a controller with minimum mean square error. These are included in the appendix.

The ultimate sensitivity method

The ultimate sensitivity method is a classical method inspired by Ziegler and Nichols. The idea is to control the process under proportional action only and to decrease the proportional band setting until the loop oscillates. In the case of heat/cool control the relative channel 2 gain may need to be adjusted too, until the oscillations are reasonably symmetrical. This value of the proportional band Pu and the period of oscillation Tu are noted. A possible sequence of operation is as shown in figure 9-21. The proportional band setting is decreased progressively until sustained oscilation is achieved. The PID values can then computed from table 9-2.

These settings are acceptable for a large variety of systems and were originally designed to give quarter decay ratio for disturbances. This usually gives poor setpoint responses and is not very good for processes with long dead-time. Refinements of these settings are given by Hang, Astrom and Ho [5]. Their refinements uses a combination of reaction curve and the ultimate sensitivity tests. See also the section on more advanced strategies in this document for the rules and guidelines laid down by these refined methods.

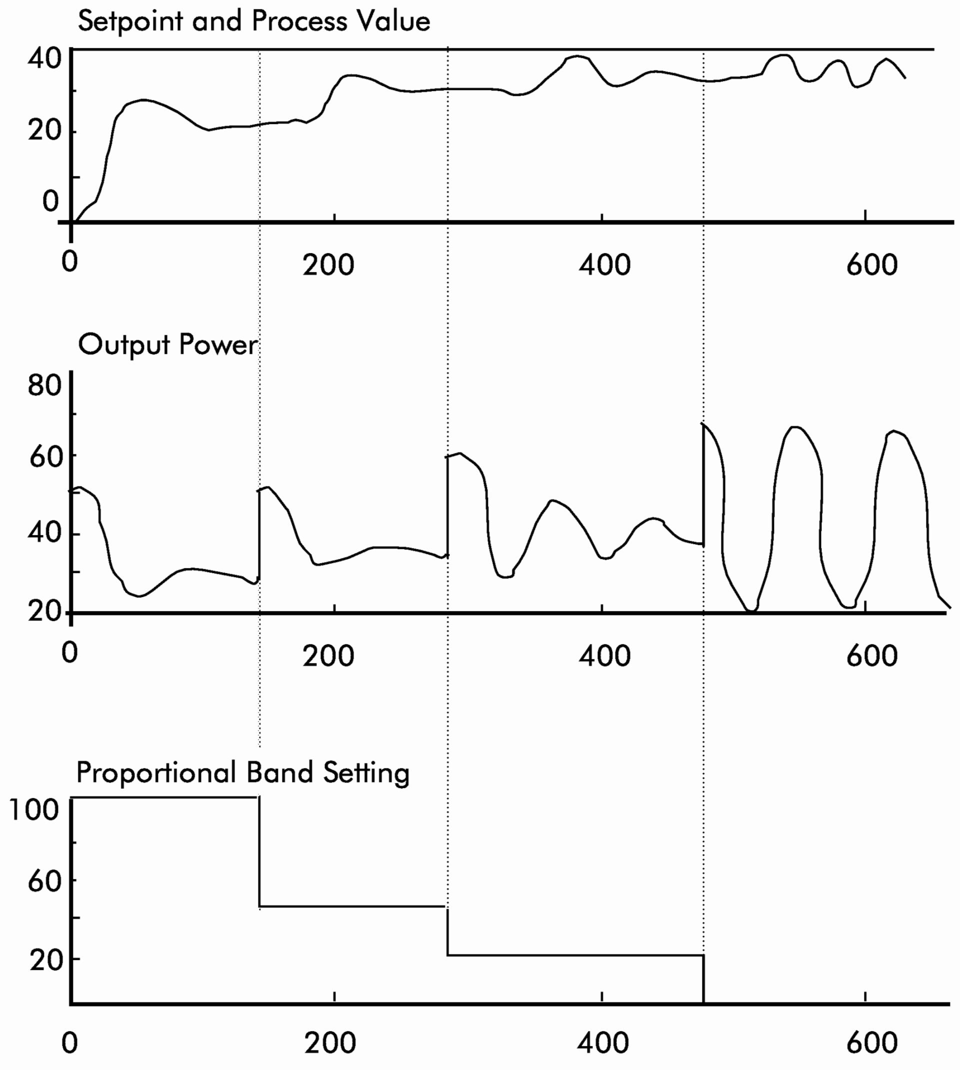


Figure 9-21 Example of an ultimate sensitivity test

Subsequent fine tuning of the controllers can then be done manually.

Trial and error method

In many instances it is not possible to employ either of the methods described above for getting a starting value for PID settings. Certain basic rules help tuning the control loop.

1. Tuning is typically performed in the order P, I, D, relative channel 2 gain and cycle-time, cutback (if applicable). Set cutbacks to off before starting manual tuning with trial and error. Set cycle-times to minimum and relative channel 2 gain to unity unless it is clear that they are set reasonably. Set Span_High and Span_Low to the absolute high and low limits of the process value. Note that the function blocks will enter a sensor break condition if the process value is more than 10% outside the span.

2. Ensure that the correct form of control (i.e. reverse or direct acting) is used.
3. Do not use integral or derivative if they are not really necessary. With most single capacities a narrow proportional band settings is adequate. Level control in a surge tank is a good example. Inner loops in cascade control is another example. P only or PD control (in special cases) usually work quite well.
4. PI control is quite acceptable in most regulation problems. Derivative control is of little or no use with ;

   Processes with small lags or inertia: Derivative action has nothing to contribute here.

   Processes with large dead-times: Derivative action does not provide the correct form of process value prediction here.

   Processes with high noise levels: Derivative action amplifies the noise levels significantly thus resulting in poor performance.
5. PID works extremely well for processes with large thermal lags and little or no dead-times.
6. Increasing the Prop_Band reduces the recovery speed from disturbances and improves stability for open-loop stable processes. For unstable systems like batch reactors the Prop_Band must be narrowed down up to a point to improve stability.
7. Increasing Integral time slows down oscillations in open-loop stable processes.
8. Increasing Derivative time improves recovery speed and stability up to a point. The ratio of Integral to Derivative of less than 3 to 1 is not recommended.
9. In a dual channel PID increase relative channel 2 gain if the controller uses too much power in the second channel thus causing oscillations.
10. When using time proportioning outputs, if logic outputs are used set the cycle time to some small value such as 1 second or less. If relays are used reasonable values are about 1/10th to 1/20th of integral time. Too fast a cycle time will wear the relays out and is usually unnecessary. Too large a cycle time will increase the tendency of the loop to limit cycle (hunt).
11. If PID values appear to be adequate for small changes and not for large ones use the function of cutback. Set initial values of cutback to the proportional band setting in engineering units. If using derivative control, then increasing the cutback values slightly outside the proportional band setting may improve overshoots to setpoint changes. Do not deviate too far outside or inside the proportional band setting of the PID controller when adjusting the cutback values.
12. The aim of the filters on the incoming analogue signals is to filter out the unwanted noise. Too long a filter time constant may lead to loop instability problems as then the lag on the process value may become too

large. The rule of thumb for the filter selection is that the filter time constant should be at least an order of magnitude smaller than the process time scale. If the noise is still prevalent at this scale then there is very little any controller can do. A better sensing device may be necessary.

Tuning of valve positioners and additional parameters

Valve positioners require tuning of three more parameters. These are the valve travel time, the minimum on time and the update time. The valve travel time is as discussed in the section, is the time taken for the continuous travel of the valve from one end stop to the other.

Typically, an experiment in measuring this travel time consists of opening the valve fully and subsequently closing it. Travel time is then the average of the two times. The minimum on time is the minimum length of time an on pulse must last before the valve begins to react. In case of availability of a valve position signal the minimum on time can be set much more accurately by examining this signal.

The position signal should probably be filtered by a filter time constant of one second or so to remove the effect of noise to a reasonable extent. The valve update like the cycle time of time proportioning outputs regulates the amount of activity of the raise and lower pulses. The valve update time, as the name suggests, is the interval at which the "setpoint" of the valve position part of the algorithm is updated. This means that every "valve update time" the PID part of the controller requests a "new" position from the positioner part of the algorithm. If the valve update time is set to its minimum value (i.e. the task interval of the function block) then the valve activity will be at its highest. The controller will almost always require to raise or lower the valve slightly at every sample.

As we increase the update time there will be an interval where the valve receives a few raise and lower pulses in the initial part of the update time and there is a subsequent period of total inactivity. If the update time is set too long (like cycle time in time proportioning) there is a likelihood of limit cycling effects (hunting). A reasonable value for the valve update time is the larger of Ti/24 or Td/4. In some exceptional applications, especially when dealing with faster processes the update time may be needed to be set to values larger than the recommended figures suggested here to reduce the valve activity.

For PID tuning using the tests described the situation is somewhat different. Clearly, if the valve position signals are not available it would be impossible to know exactly what output signal is being applied, so the size of the "bump" is selected by setting the duration of an on or an off pulse. If the valve travel time is known to reasonable accuracy then the size of the incremental output applied is known.

Provided the response time of the process is reasonably longer than that of the valve the PID settings using the tables is probably quite adequate. Typically, the proportional band may have to be increased somewhat. The ultimate sensitivity method can be applied and the tuning parameters should be reasonably adequate. In either case it is important to have a reasonable idea about the travel time of the valve.

Automatic one shot tuning

PC3000 in common with 800 and 900 series of instruments offers single shot self-tune facility. A brief description of the method used is given here. The readers are also referred to [3] which gives a detailed discussion of the performance of the algorithm on four zones of a wire coating plastic extruder.

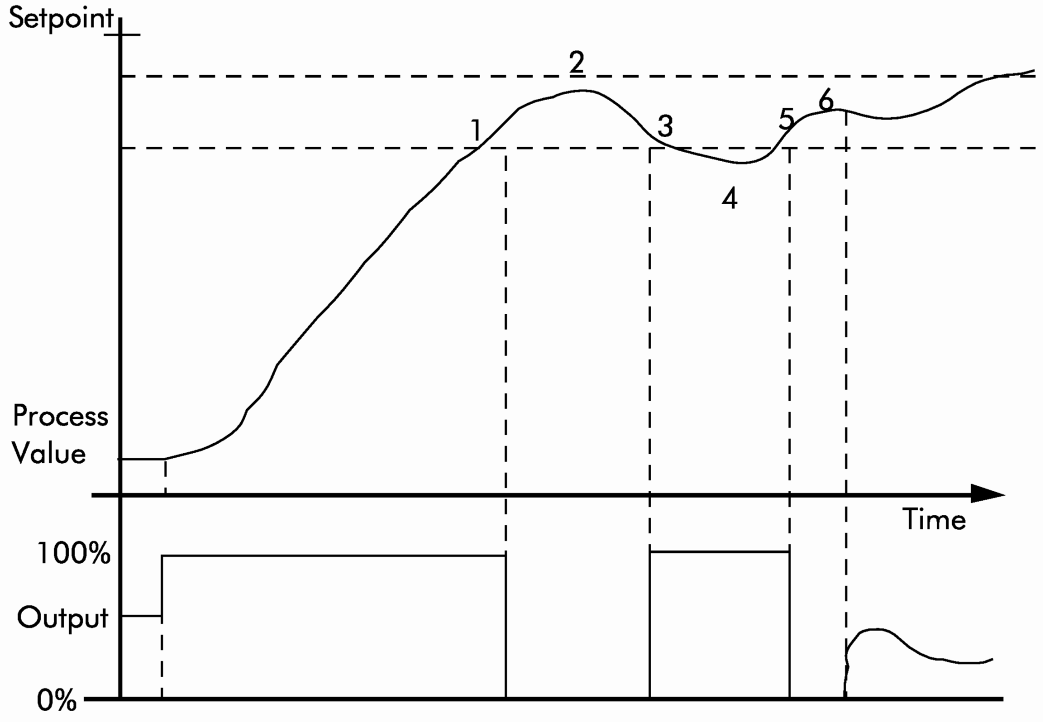


Figure 9-22 Basic PID_Auto sequence

The method is based on closed-loop cycling method (CLCM). The rules applied are to give a critically damped closed-loop response. The tuner can operate in any of the variety of P, PI, PD and PID control configuration. The method requires minimal prior knowledge -- the only requirement is the magnitude of maximum/minimum output values allowable on the process. It is undesirable to allow closed-loop cycling on some processes: auto tune can not be used for these. By default the maximum and minimum values of the controller output during an auto tune are set to the values of Output_High and Output_Low. In more recent versions of firmware AT_Out_High and AT_Out_Low can be used. Before starting an Autotune the user can limit these values to reduce the magnitude of process upset during the tuning phase.

For the implementation of Autotune the controller output is either

frozen if the initial error is less than 1% of the span, or

set to zero if the initial error is larger than 1% of the span.

The controller output is maintained at this level for a prescribed period of time depending on the task the PID_Auto function block is associated with. This is one minute for task intervals upto 5 seconds and it is appropriately adjusted for longer intervals but never more than five minutes. For most applications it is unlikely that task intervals as long as 5 seconds will be required for the PID block. During the wait period the block examines the overall trend of the process variable and computes a triggering level for the subsequent adaptive tuning if required.

During the initial wait period the setpoint can be set to the required operating condition. For setpoints which differ from the process value by more than 1% of the span, the algorithm applies maximum or minimum controller output to drive the process value to the setpoint as quickly as possible. The process value is continuously monitored, and an estimate of the delay time in the process is made from these readings. This estimate is continuously updated until the next stage of tuning is entered. This estimate is used to provide timeouts in a supervisory algorithm, should the process not complete in any of the subsequent phases as expected.

To avoid overshoots the tuner performs its test at an artificial setpoint below or above the actual setpoint. Some processes with long dead-times will overshoot this setpoint despite the precaution of the artificial setpoint. It is recommended to use limited output values for processes with long dead-times. When the process value reaches this artificial setpoint, the algorithm proceeds into a finite state machine implementation of on-off control at this process value. The magnitude of this peak and the time to achieve this are stored and the algorithm proceeds to the next state. The algorithm continues through states 1 to 6 (7 if dual channel output is chosen) as shown in figure 9-22 The tuner checks all timings for reasonableness and when the final stage is reached the PID values according to a modified Ziegler-Nichols rule are computed.

The tuner will perform a tune at setpoint if the process value at the end of the initial wait period is no more than 1% away from the setpoint. In this case, the dead-time of the process is not estimated and the PID parameters are computed on the basis of the oscillation amplitude and periods alone.

Figure 9-23 shows an example of autotune sequences upto a new setpoint, at setpoint and down to a new setpoint. As can be seen the disturbance caused by the autotune is not very significant.

In cases where nonlinear water cooling is used the Ch2_Linear should be set to No(1) and the maximum cooling output used during the Autotune will be set to 20%. Note that the Delin_type parameter of the time proportioning output for this channel has to be set to either D_800 or D_EM1: whichever appropriate for the application. See the PC3000 function block manual for more details.

The tuner will compute the following parameters:

1. P, I and D: If either integral or derivative are set to zero the tuner computes the parameters for P-only (if both are zero), PI (if D is set to zero), PD (if I set to zero) and PID if none are set to zero. If the tuner detects a delay dominant system it automatically turns the derivative term off.
2. Relative Channel 2 Gain: This is set if dual channel PID is used

(i.e.Output_Low is negative). This parameter is set to unity if the tuner fails to complete successfully..

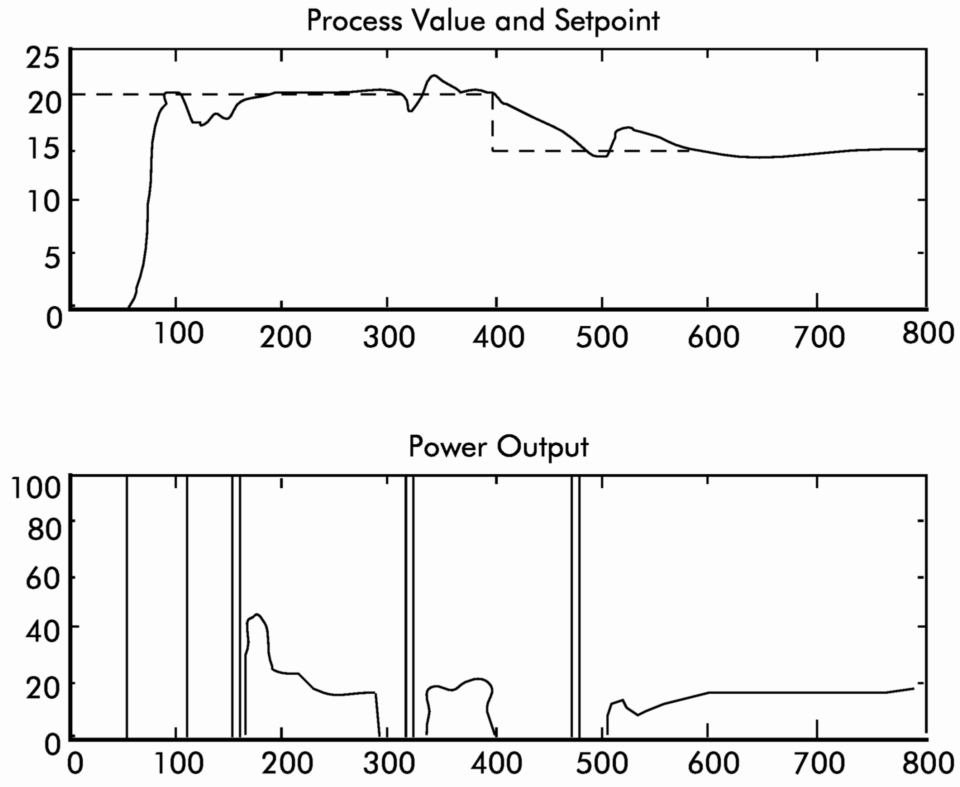


Figure 9-23   PID_Auto autotune sequence

3. Cutback: Cutback_Low is set (if deemed necessary by the tuner) on a tune upto a setpoint. Similarly, Cutback_High is set (if deemed necessary) on a tune down to a setpoint. No cutback is set during tunes at setpoint unless the cutback value is lower than the newly computed proportional band.
4. Cycle Times: The cycle times are computed if their initial values are set larger than 5 seconds. The tuner does not change these values if they are set below five seconds.
5. Adaptive Tune Parameters: The Autotune also sets the three adaptive tune parameters: Trigger_Val, MTC, Q parameters for DRA (Disturbance Response Analysis) and LSAT (Least Squares Adaptive Tuner) respectively. See the section on adaptive tune for more details.

The tuner will not change the dead-zone (dead-band) parameter. Also if manual mode, sensor break or track conditions are entered at any point during the Autotune the whole sequence is aborted and no change is made to any of the parameters.

Note: The user must ensure the correct form of direct or reverse acting control is used: the tuner does not make this decision.

The diagnostic parameters AT_State, and Zn_Stage indicate the appropriate stage of the Autotune.

How to get the best results from Autotune

This section provides some tips on how to use self-tune at its best advantage and how to avoid / minimise some problems:

AT_Atate	Description
0	Reset
1	Initialisation
2	Monitor Quiescent Noise
3	End of Monitor Noise
4	Start with New Setpoint
5	End of Startup with New Setpoint
6	Startup with PV at Setpoint
7	End of Startup with PV at Setpoint
8	Ziegler-Nichols Sequence
9	Calculate New Parameters
10	Write Update Status
11	Autotune Aborted
12	Autotune Completed

Table 9-3 The Autotune variable AT_State

Zn_Stage	**Description**
0	Initialisation
1	Find peak PV & Reverse Output
2	Find PV crossing PV1
3	Find peak PV & Reverse Output
4	Find PV crossing PV1 & Test for dominant delay
5	Find peak PV & reverse Output or PV change
6	Find PV crossing PV1 & Adjust Trend and Output
7	Find Peak PV & Reverse Output
8	Find PV crossing PV1 & Calculate Parameters

Table 9-4 The Autotune diagnostic variable Zn_Stage

1. Always attempt to select self-tune when the instrument is providing a stable level of output and the process variable is fairly steady for the best results (i.e. select selftune with the initial setpoint at process value and then switch the setpoint to the new desired value). Only change the setpoint (if needed) during the first minute wait stage.
2. If the start-up conditions of the process are always very similar and there are no significant changes to the load characteristics then it is necessary to perform self-tune only once and not every time the process is turned on.
3. Tunes up to a setpoint in temperature control of extruders and furnaces are usually more effective than tunes down to or at setpoints.
4. Autotunes with processes with time constants below 20 seconds or so are not usually very successful, unless PID_Auto is moved to a faster task interval. The tuning operation takes about 5 milliseconds out of the task time. The task interval can not be usefully reduced by a factor more than 2 or 3 compared to the 100ms task interval. When increasing the task interval for the PID_Auto function block make sure that the task interval is about two orders of magnitude smaller than the process time constant. E.g. If process time constant is 100 seconds, the task internal should be shorter than 1 second.
5. Do not use the Autotune with processes which have cyclic disturbances. This may confuse the tuner in setting the PID values.
6. The tuner is disabled if IntegralHold is set.
7. Ensure that the right values of output limits are used. Do not change the limits during the autotune phase.
8. Ensure that the process value is within the range measurable by the sensor. Some sensors (e.g. pyrometers) are only useful over a limited

range and below that they indicate their minimum value although the actual process value is well below that value. When using multiple sensors for different ranges make sure that the Autotune is performed in the appropriate range.

9. Note that the tuning rules are optimised for extruder/ furnace temperature control and tend to give quite tight and active control. If such a response type is not suited to particular systems the parameters must then be manually modified.
10. There are various timeout mechanisms built into the Autotune sequence. If the process fails to respond in the prescribed manner the tuner may go through one of these timeouts before skipping to the sections of the sequence it is capable of completing and gives its best estimate of the PID settings.

Tuning with valve positioners

The one shot tuning facility is also available for the valve positioner. The tuner however does not set the valve update time,minimum on time and valve travel time. Note that the valve travel time should be set to the actual value of the travel time of the valve from one end stop to the other. Failure to do this causes the tuner to set an incorrect proportional band setting for the controller.

Adaptive tuning

PC3000 function block also has two more tuners which unlike the one-shot scheme operate continuously. These are DRA (Disturbance Response Analysis) which in essence is a simple expert system examining the response curve of the process variable to setpoint changes and external disturbances and LSAT (Least Squares Adaptive Tuner) which is basically a variant of model reference strategy. DRA performs the coarse tuning and LSAT the fine tuning. There is also an overseer which coordinates the activity of these two tuners as well as providing safety supervision for these tuners such as

setting limits on the PID parameters

ensuring that the parameter set is consistent

checking for loss of restoring action in the controller

Disturbance response analysis

This technique is based on continuous analysis of process-value waveforms as a function of time, during operational closed-loop control. The actual responses monitored are the error signals generated, when the control loop experiences a disturbance. As control errors can be generated by process noise, external process disturbances and setpoint changes, it is helpful to be able to recognise these specific sources, while evaluating the response to the disturbance. A fundamental problem of the waveform analysis technique is that it may not have enough information to be aware of changes of the initial disturbance.

The problem of noise is soluble by providing a means of measuring the noise environment, during the Autotune test prior to initialising the adaptive tuner, and then parameterising the noise filters. As mentioned earlier Autotune sets the Trigger_Val parameter which should be roughly set to the peak to peak magnitude of the noise level. This value can also be adjusted by the user. Setting the value too low, results in the adaptive tuner being confused by the process noise and as a result it will widen the proportional band setting and increase integral times to compensate for this. If at all possible it is best to do an Autotune before an adaptive tune. Otherwise, ensure that the Trigger_Val is at least set to the peak to peak value of process noise.

The tuner monitors setpoint changes and it takes appropriate action when there are changes of the setpoint. If the changes are too frequent the overseer disables changes of PID parameters as it is then impossible to distinguish whether the process value is oscillating as a result of frequent setpoint changes or the loop tuning is poor.

DRA analyses the response of the process over a long enough period of time to enable consistency checks to be performed on the measurement. This typically amounts to a period of time commensurate with five or six integral times before DRA is confident that the tuning needs proper adjustment.

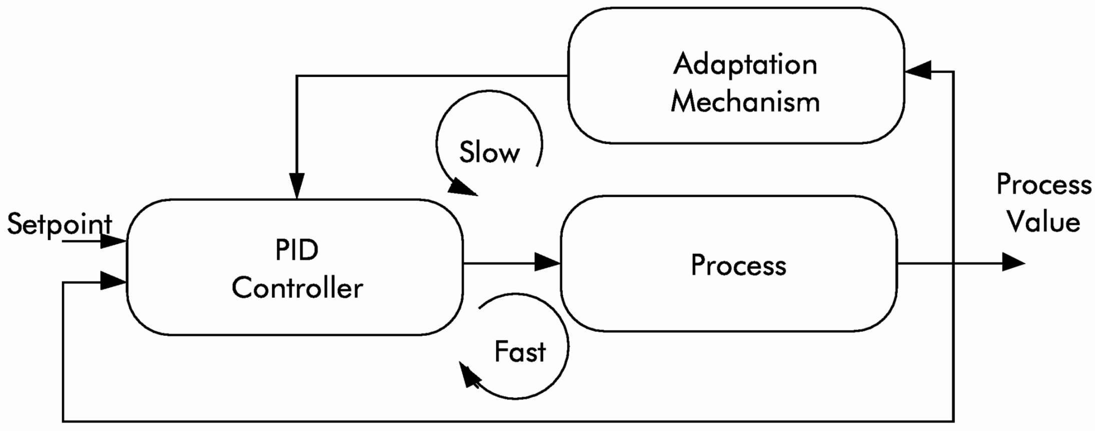
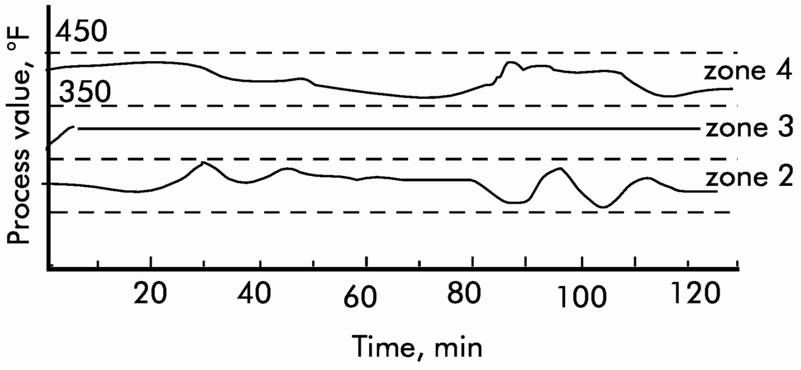


Figure 9-24 A basic adaptive control scheme



Zone 2	**Auto tune**	**Manual setting**	**No change**	**Manual setting**	**Adapt**
XP(%)	1.9	1.9		1.9	1.9
TI(sec)	930	240		30	960
TD(sec)	152	60		10	160

Zone 4	**Auto tune**	**Manual setting**	**Adapt**	**Manual**	**Adapt**
XP(%)	1.7	0.8	1.4	1.4	1.4
TI(sec)	520	520	520	1800	613
TD(sec)	88	88	88	300	102

Figure 9-25 DRA adaptive tuning of a multizone extruder

The distinguishing feature of 'adaptive systems' is that there are two basic loops present as shown in figure 9-24. The adaptation loop is at least an order of magnitude slower than the PID feedback loop. It would therefore be unreasonable to expect the adaptation loop to respond quickly to rapid process variations. For example, pH processes go through rapid and large gain variations close to the neutralisation region. A general purpose adaptive control scheme can not compensate for these changes -- the gain of the pH process varies almost as rapidly as the effluent flow variables and should therfore be treated as a bona fide state of the process. Gain scheduling provides a far better solution here. Adaptive schemes do however compensate for initial poor tuning over a sufficiently long period of time (e.g. a few integral times) provided the underlying process is not varying too rapidly.

Once the DRA tuner has been triggered by an error exceeding the Trigger_Val, subsequent error peaks are measured in similar way to the Autotune (CLCM) described in the previous section. Poorly tuned loops may exhibit a damped oscillatory recovery from a disturbance such as the ones shown in figure 9-25. DRA can then adjust these parameters using the rules described in the PC3000 function block manual.

One disadvantage with DRA and any other waveform analysis technique is that identification of a poor control response should occur before the control response can be corrected. The method is extremely robust provided the trigger value is set appropriately as previously described.

DRA may decide to include integral control even when it is not originally selected but it will not tune the derivative, if either it is turned off by the user or by Autotune.

DRA has two diagnostic parameters which are described in the tables 9-5 and 9-6. DRA_State indicates the current state of DRA and DRA_Last the last change DRA made to the control settings.

DRA_State	**Description**
0	Allow Settle
1	Wait for Trigger
2	Find Peak 1
3	Find Zero 1
4	Find Peak 2
5	Find Zero 2
6	Find Peak 3
7	Find Zero 3
8	Find Peak 4
9	Find Zero 4
10	Find Peak 5
11	End on Zero 4 Abort
12	End on Peak 4 Found
13	End on Peak 5 Abort
14	End on Peak 5 Found
15	Prepare Update

Table 9-5 The states of DRA

DRA_Last	Description
0	Not adapted
1	Reduced Damping
2	Increased Gain
3	Decreased Times
4	Increased Times
5	Decreased Gain

Table 9-6 The adaption of DRA

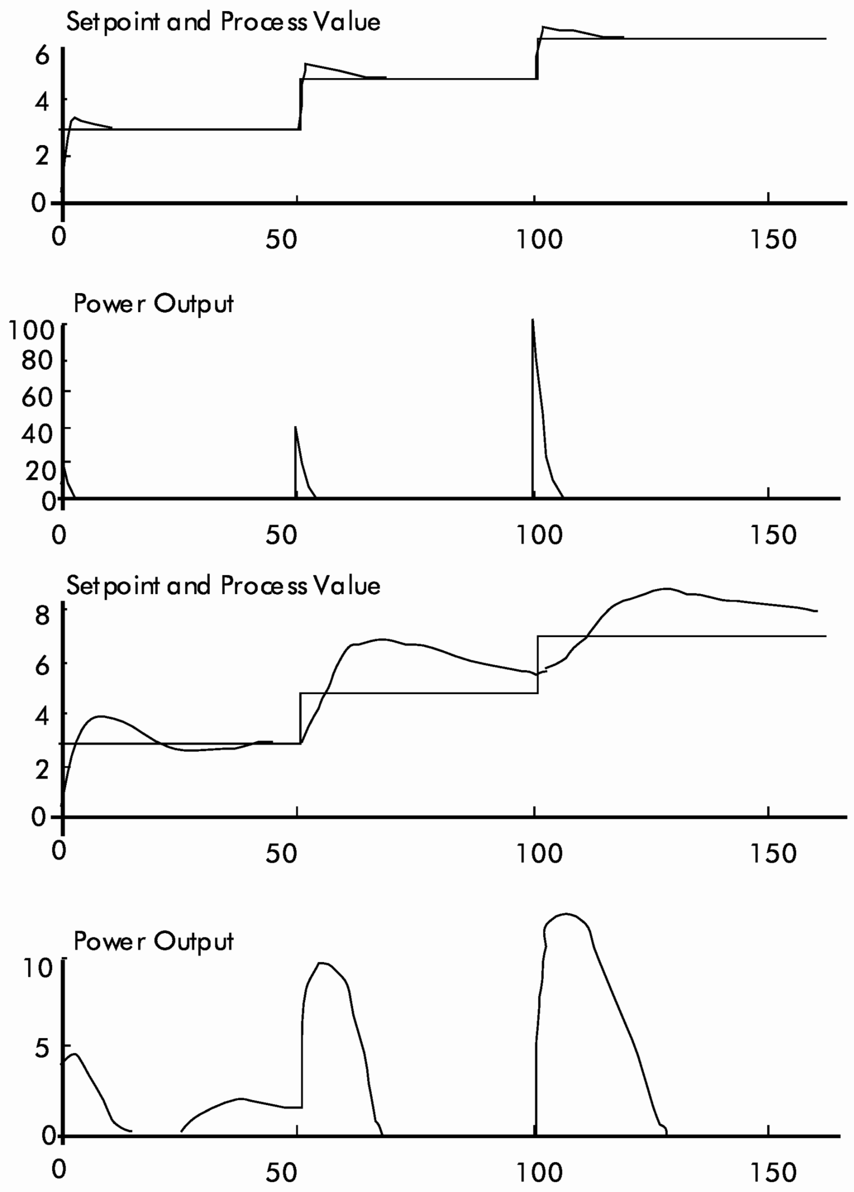


Figure 9-26 Schematic diagram of the least squares adaptive tuner

The least squares adaptive tuner

LSAT is a model based tuning scheme which has a variant of the standard recursive least squares estimator and it estimates the parameters of a predictor model which are then used to set the PID parameters. LSAT operates as a fine tuning mechanism in tandem with the DRA described in the preceeding section.

There are two parameters MTC (Model Time Constant) and Q (Control detuning factor) which are used with LSAT. Autotune and DRA set these parameters automatically. When the controller is in manual these values are set to one tenth of the integral time and eight percent of the proportional band setting (in engineering units). They can also be adjusted by the user though there is no need to do so in practice unless the user has sufficient amount of experience and expertise of LSAT. Briefly, increasing MTC has the effect of increasing the required closed loop model time constant and increasing Q has a similar effect but through reducing the loop gains. LSAT is deactivated if MTC is less than the task interval in milliseconds divided by 100. For example for a PID_Auto instance on 100ms task MTC must at least be set to 1 which in turn implies that integral time must at least be 10 seconds.

Schematically, the control scheme is as shown in figure 9-34.

LSAT is used as a fine tuning mechanism and should therefore not be expected to change the parameter set by Autotune or DRA by an enormous amount. Unlike DRA, that operates only if errors are sufficiently large, it is operating continuously and as such can make changes at any time. The overseer ensures that paramaters are within specified ranges.

The resulting three tuners form the "Eurotherm's Composite Tuner" which is also available with the 900EPC series of instruments.

LSAT like DRA may activate integral control when needed but will not activate derivative if deactivated by the user or Autotune.

How to get the best results from adaptive tune

There are certain simple precautions which are necessary when using adaptive control.

1. Wherever possible perform an Autotune before an adaptive tune. This ensures that the starting PID parameters are far closer to their optimum values than they would normally be. The disruption caused by an Autotune sequence is usually no worse than a poorly tuned PID!
2. If an Autotune can not be performed make sure that trigger value is set to some sensible value (typically the peak to peak value of the noise).
3. Do not use adaptive tune on processes with cyclic disturbances. If it is needed to do so then ensure that the trigger value is set to the peak to peak value of this disturbance.
4. Do not leave the controller in adaptive tune while changing the wiring, disconnecting thermocouples or servicing the equipment. There is a

loss or restoring action detector inside the algorithm which is designed to detect such situations but the best policy is avoidance.

5. Like Autotune there is no point in attempting to use adaptive tune on processes which have time constants less than two orders of magnitude larger than the task interval of PID_Auto.

6. Note that the adaptive tuner does not adjust the dead zone in the dual channel PID and the minimum on time, travel time and update time of the valve positioner. The user has to choose these.

It is frequently only necessary to use the Autotune to reduce commissioning time. Feedback control overcomes 90% of the problems -adaptation is really only necessary for a small class of processes. For example, using a adaptive PI controller on a process with long dead-time can not improve response times significantly: proper dead-time compensation or system redesign is necessary here not adaptive control. It is difficult to pin point under what conditions adaptive control gives better results and under what circumstances worse. Suffice it to say that continuous adaptive tuners are not a panacea for problem loops: a one shot tuner will be more reliable on a noisy loop, and continuous retuning is often not necessary.

Continuous adaptation to changing loop conditions is a different problem. Here the user must understand that the adaptive tuner, when enabled, has absolute control over the process and so he/she should exercise caution until there is a certain degree of confidence that the process is benefitting from this. Adaptive tuners can mask process problems. For example, fouling of pipes in a heat exchanger will cause a gradual change in the heat exchange rates which will be compensated by the controller to obtain optimum performance for the given conditions. If this goes totally undetected the problem will only surface when it is too late thus making proper regular maintenance harder!

GAIN SCHEDULING

Gain scheduling is a simple though effective method of control. Most industrial processes are nonlinear and as such a fixed set of PID parameters may not be adequate for "optimum" performance.

It is sometimes possible to find auxilliary variables which correlate closely with changes in the process dynamics. One can use these to adjust the controller parameters to values which are more suited to the new conditions. The changeover of the parameters can be either done continuously where the PID parameters (e.g. proportional bands, integral and derivative times) area continuous function of the auxiliary variable or in discrete steps where the region of operation is split into several segments and PID parameters are adjusted as the auxilliary variable crosses over from one segment to the next.

One drawback of gain scheduling is that it is an open-loop compensation of feedback parameters. There is no feedback to compensate an incorrect schedule. Another drawback is that the design may be time consuming. The regulator parameters have to be determined for many operating conditions. The function of Autotune mechanism described in the previous section may help reduce the length of time during the design cycle.

The main advantage of gain scheduling on the other hand is rapid response to process changes. There is no estimator and tuning transient which typically is at least an order of magnitude slower than the response times of the loops itself. Thus very rapid changes in the process can be catered for, very precisely.

Continuous scheduling

Consider a level control problem with a conical tank. Assume for the purposes of this example that we are required to use a PI controller for reasons of steady-state errors. This is very rarely the case with level control in practice but we assume it is for the duration of this example. It is then easy to see that

The cross section area of the conical tank increases as square of the level of the fluid;

The outflow of the tank increases as square root of the level of the fluid.

Figure 9-27 shows the response of the PI level controller to a series of equal changes of the setpoint from 1 to 3, from 3 to 5 and subsequently from 5 to 7 units. With a fixed PI (the bottom figure) control the response tends to become slower as the level increases. By analysing the level control problem it can be seen that a way to compensate for this is to narrow the proportional band with the increase in the cross section area. Recall that the "gain" of the tank varies as inverse of the level.

```
Pid.Prop_Band := 50 / (PV.Val * PV.Val);
Pid.Debump_Dis := 1;
```

This ensures that we get consistent responses at different levels. See the top half of figure 9-27.

Note that we have had to disable debumping of the PID during the gain schedule. This is because every time a change to the proportional band setting is made the controller goes through an automatic debumping procedure. The debumping operation in this case disables the control altogether and the controller will never get going properly!

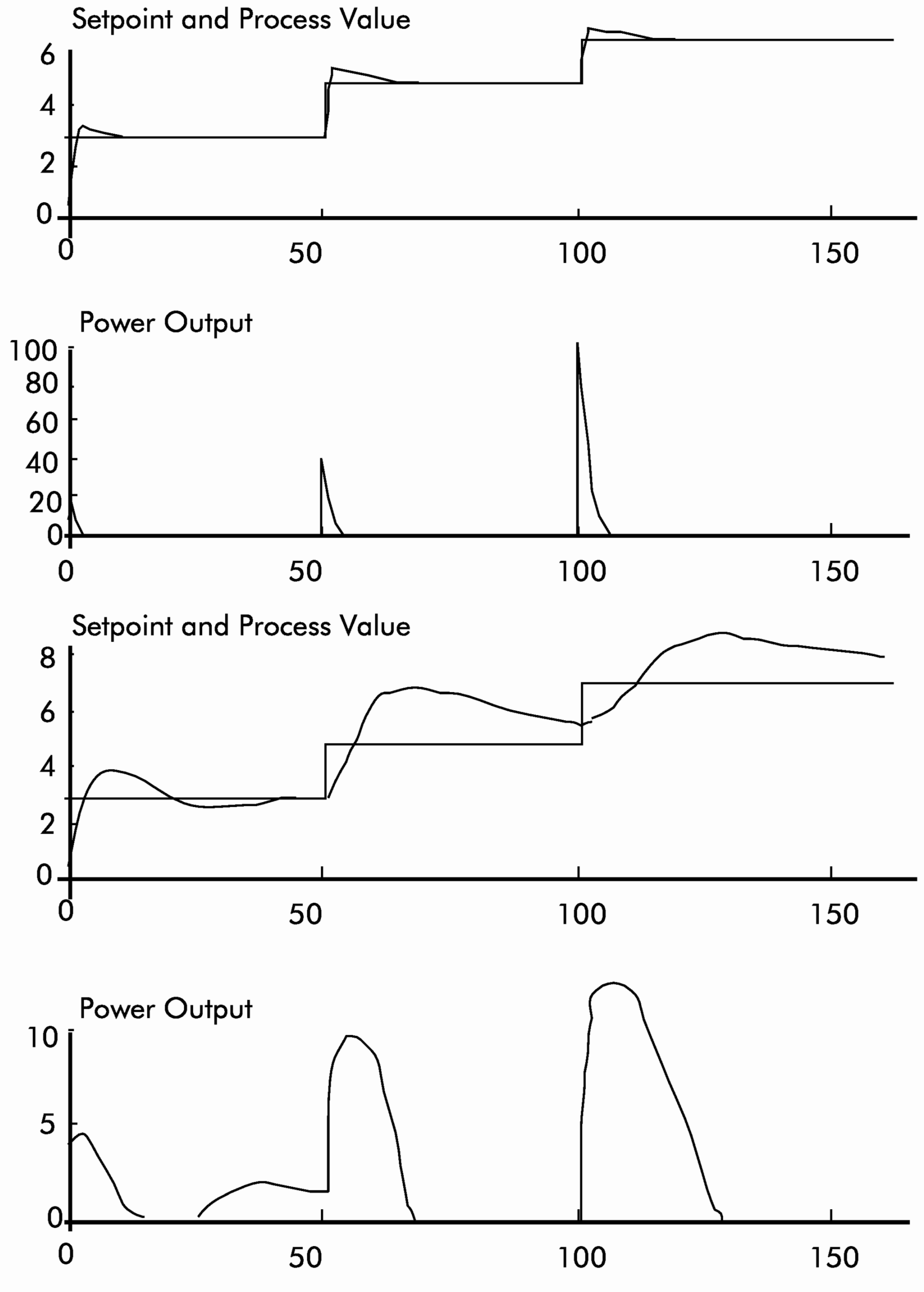


Figure 9-27 Continuous gain scheduling

Gain scheduling

Consider a heat exchanger. The temperature on the secondary side of the heat exchanger is controlled by a valve position on the primary side. The gain and process time constant of the heat exchange are functions of the flow in the secondary.

The variations of the process can be compensated by setting up a gain schedule against the flow of the secondary side of the heat exchanger. As the flow in the secondary tubing increases the gain of the heat exchange decreases and the time constant increases. A simple way of setting up the gain schedule for proportional band setting and the integral time is via Autotunes at different operating conditions and storing these values in a select function block.

One can setup three different segments of low, medium and high flow rates. One can store the PI parameters in a Select_REAL, Select_TIME block, for example. The wiring for such a setup is therefore

```
Pid.Prop_Band := Pid_XP.Output;
Pid.Integral  := Pid_TI.Output;
Pid_XP.Index  := REAL_TO_DINT(IN := Flow.Process_Val/33.34) + 1;
Pid_TI.Index  := Pid_XP.Index;
```

If all of the function blocks are on the same task then the order of execution is guaranteed and therefore PID is ensured to use a consistent set of parameters. The select function blocks will execute before the PID does.

Note that unlike the example in the preceding section debumping is not disabled. This is to ensure that the switchover from one set of parameters to the other does not create a bump in the output. Of course what is expected is that the gain scheduling variable (i.e. flow) is not oscillating rapidly either side of one of the boundaries. One may employ a low pass filter to ensure that the controller only reacts to true changes in the flow.

Note also that scheduling need not be limited to P, I and D. Frequently, some other parameter has to be scheduled, for example the relative cool gain. Consider an environmental chamber where temperature is controlled from a few hundred degrees celsius to minus 190 degrees celsius. Liquid nitrogen will provide very efficient cooling when the process value is well above the liquid nitrogen temperature but obviously is far less effective when the process value reaches -180 degrees, for example. In order to get commensurate type responses along the entire temperature range the relative cool gain of the controller has to be scheduled against either the setpoint or process value.

It may also be necessary to switch the structure of the controller. For example, when ramping is active it is frequently beneficial to have the derivative term acting on the error and during regulation one switches back to derivative on process value. Clearly, such changeovers are trivial in the case of PC3000.

Another way of setting up schedules is via different recipes. If the same equipment is used to produce different brands of products then the recipe system can be used for switching PID values as well as setpoints.

SETPOINT PROGRAMMING

One of the requirements for a control system is good setpoint following. In many temperature control applications it is required to follow a sequence of ramps and dwells of various lengths. In PC3000 this can be achieved very easily through the function of the ramp function block. The setpoint of the PID is wired to the output of a ramp function block.

```
Pid.Setpoint := ramp.Output;
```

The reference profile is segmented to a series of ramps and dwells. Each of the segments can then form a step of a macro in the sequence program whose function is to generate the setpoint profile.

Assume that the output of the ramp is set to

SP1.Val and we require to ramp to SP2.Val in T.Val seconds. Then the ramp rate is set in the following simple way

```
ramp.Rate     := (SP2.Val - SP1.Val) / TIME_TO_REAL(IN:= T.Val);
ramp.Setpoint := SP2.Val;
```

The Recipe function block suite provides an excellent mechanism for storing a variety of setpoint programs for different grades of products. It is a relatively straightforward task to write a SFC macro which picks the correct inputs and outputs from a series of recipe slave variables.

Overshoot inhibition

One of the problems associated with PI or PID control is that process value may overshoot the reference at the end of a ramp. This is primarily due to the integral action effects. There are several solutions available and depending on the application area one or more of these are applicable to the problem at hand.

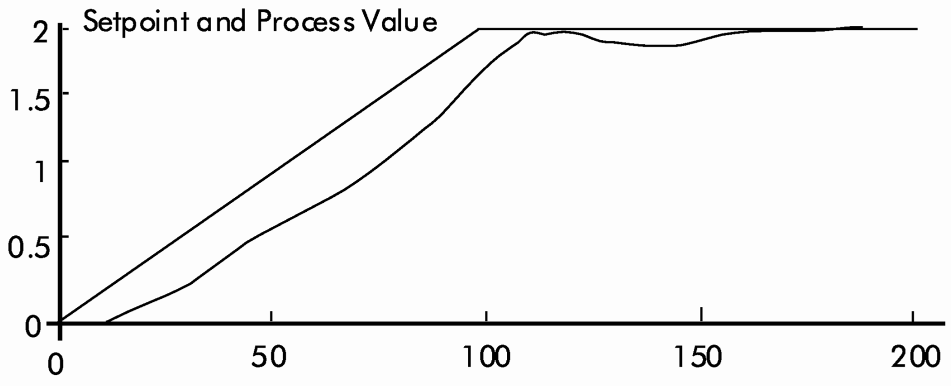
Use of PID_Auto overshoot inhibition scheme

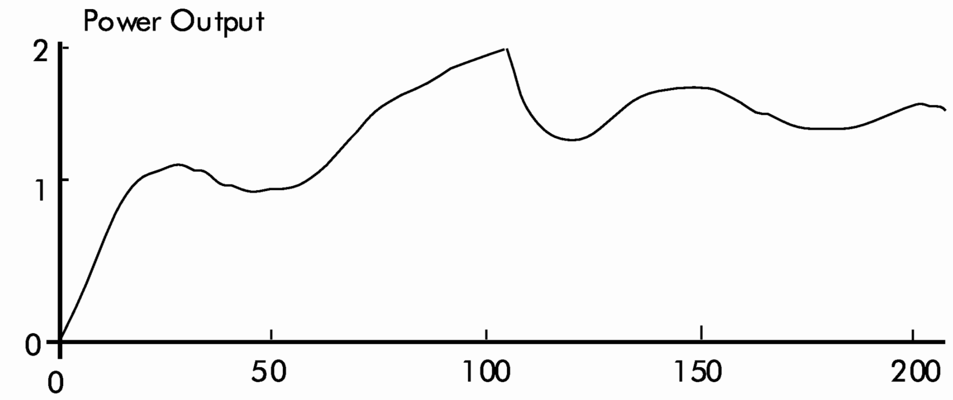
The 'in-built' overshoot inhibition scheme of PID_Auto function block is described in the PC3000 function block manual. The following wiring activates the overshoot inhibition.

```
Pid_Auto.Setpoint  := ramp.Output;

Pid_Auto.Ramp_Rate := ramp.Rate;

Pid_Auto.Ramp_Units:= ramp.Rate_Units;

Pid_Auto.Target_SP := ramp.Setpoint;
```

The Inhibiter parameter can then be adjusted to give the appropriate response. It should typically be around 0.5.

The operation of the internal inhibition scheme is as follows. As in most cases the overshoot is due to the accumulation effect of the integral term, the inhibition scheme discharges the excess integral effect before the process value reaches the target setpoint. The place where this discharge takes place is a function of the Inhibiter value. This inhibiter value is set to a number between 0 and 1. Zero implies inhibition is off. Figure 9-28 shows the response of PID to a ramp with (bottom half) and without (top half) the overshoot inhibition activated. As can be seen the response with the overshoot inhibition active is identical to the one without until the process value gets close to the target.





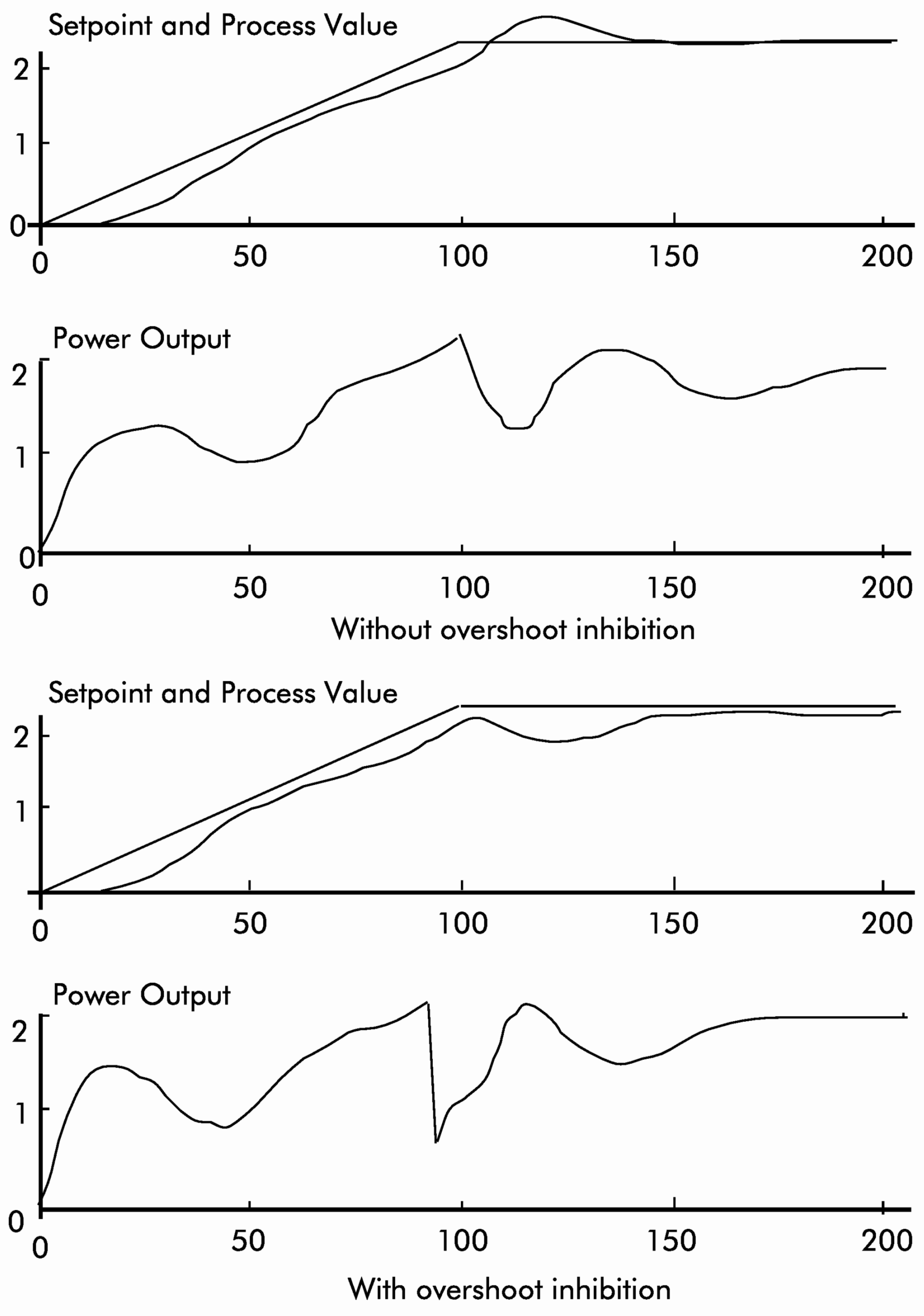


Figure 9-28 Overshoot inhibition in PID_Auto

Use with PID function block

The actions taken by the internal overshoot inhibition scheme of the PID_Auto function block may be too drastic (e.g. we require the control to be rate limited) at the moment of the discharge of the integral term. If the proportional action only works on the process value as opposed to the error the ramp response is unlikely to overshoot but it will also be slowed down considerably. Typically, derivative should also operate on error as opposed to process value to reduce the magnitude of overshoot. Figure 9-29 shows the effect of the use of setpoint weighting described later. It is possible to obtain the appropriate response for a process by adjusting the relevant scaling factors.

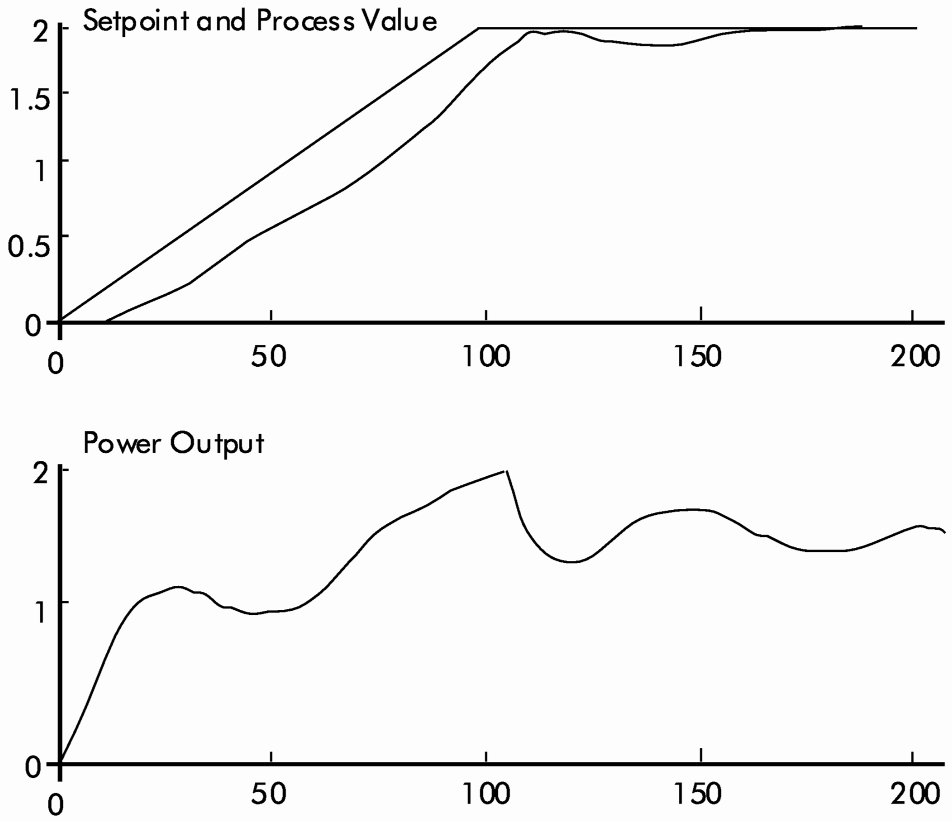


Figure 9-29 Overshoot and PID structure

PROCESS CONTROL

Cascade control

One of the most powerful tools in control system design is the use of cascade control. It is quite common to have to deal with cases where it is possible to measure some of the states of a process like temperatures, pressures and so forth along the line. In extrusion the most common use of the cascade structure is the use of deep and shallow thermocouples. In this way the effect of disturbances such as the work heat, effect of the granularity of the plastic pellets or their chemical composition, can be seen earlier from the thermocouple and can therefore be compensated for quickly. There are a variety of other areas such as drying, environmental chambers, control of processes with valves (e.g. cascade of flow and composition) where cascade control is particularly useful. In furnaces it is also quite common to have a cascade of load and air temperatures.

The task of appropriate wiring for cascade control in PC3000 is left to the application developers as the extra flexibility allows more lattitude in the control system designs unlike the restricted form of the 900 series of instruments. This document describes some typical forms of wiring for cascade control of PIDs. It should therefore be seen as a guideline rather than the only solution for developing cascade control applications.

The basic setup

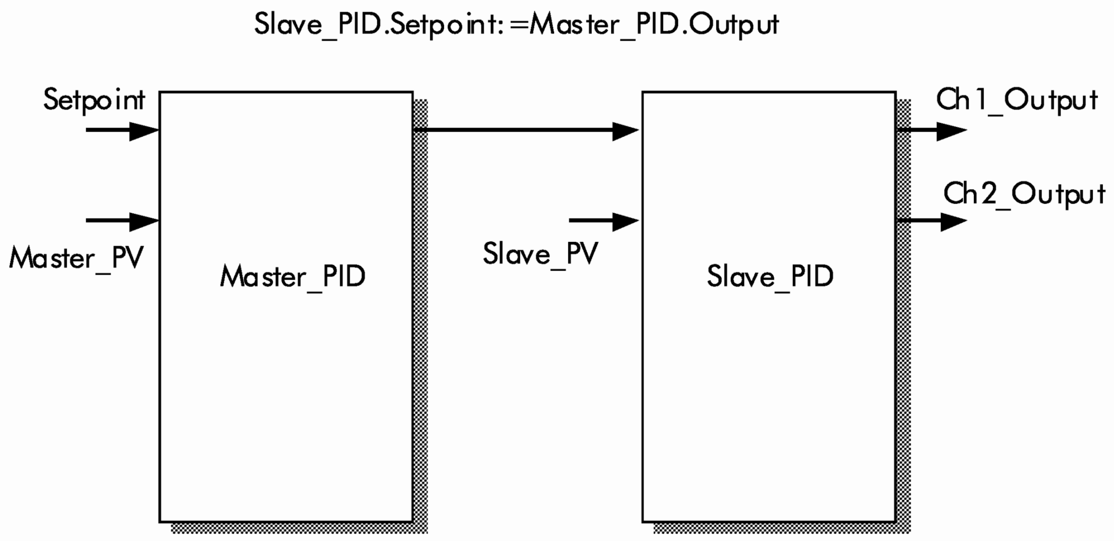


Figure 9-30 The most basic cascade structure

In its most basic form cascade of PIDs consists of a wiring of the form shown in figure 9-30. The only wiring here is

```
Slave_PID.Setpoint := Master_PID.Ch1_Output
```

This of course has several drawbacks, namely

There is no mechanism to turn the cascade operation off.

The slave process value has to be in percentages because the Output of the Master _PID function block is in percentages.

There is no limit on the slave setpoint.

When the slave is in manual, the master output is inconsequential and there is a distinct posibility of integral windup and severe 'bumping' when the control is returned to automatic again.

Figure 9-31 shows a more reasonable structure for cascade control. The following modifications have been done on the basic structure:

1. The Output of the master PID is scaled to the slave loop units as follows:

```
Slave_Span.Val  :=  Slave_PID.Span_High -Slave_PID.Span_Low;
Scaled_OP.Val   :=  Master_PID.Ch1_Output * Slave_Span.Val/100 +
                  Slave_PID.Span_Low;
```

2. If the cascade control is on then this setpoint is limited by user defined high and low limits. This can be achieved through the following wiring, for example

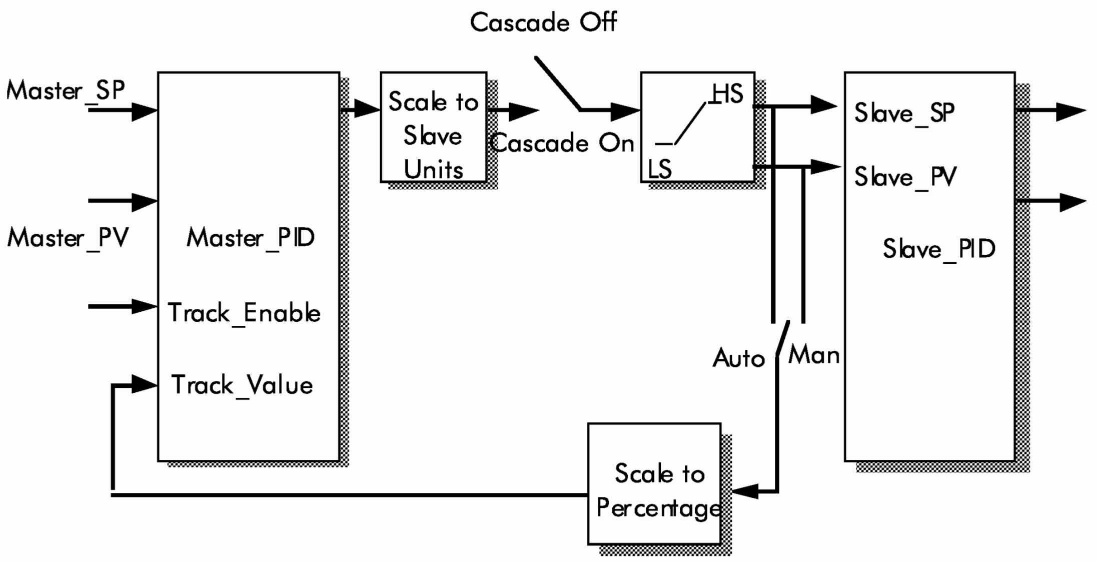


Figure 9-31   A practical cascade structure

```
Slave_PID.Setpoint  :=  SEL_REAL( G:= Cascade.Val,
IN0:=External_SP.Val,
IN1:= SEL_REAL( G := Scaled_OP.Val > HS.Val,                      IN0 :=
MAX_REAL(IN1:= Scaled_OP.Val,                                  IN2 := LS.Val),
           IN1 := MN_REAL(IN1:= Scaled_OP.Val
                IN2:= HS.Val)));
```

External_SP is a user defined variable which can be used as a local setpoint.

3. In order to ensure bumpless transfer the master PID is set to track the process value or setpoint of the slave loop i.e.

```
        Master_PID.Track_Value  :=  SEL_REAL(G := Slave_PID.Manual,
        IN0(Slave_PID.SetpointSlave_PID.Span_Low)*100/Slave_Span.Val,
                IN1 := (Slave_PID.Process_Val-
        Slave_PID.Span_Low)*100/
                Slave_Span.Val);
     Master_PID.Track_Enable := Slave_PID.Manual OR NOT Cascade.Val
                                  OR SP_Limited.Val;
```

The master is set to track if

The slave is in manual.

Cascade is off.

The slave setpoint has hit the limits.

The check for setpoint being limited can be performed as

```
SP_Limited.Val := Scaled_OP.Val>(HS.Val+EPS.Val)
                          OR Scaled_OP.Val<(LS.Val-EPS.Val);
```

EPS.Val is a user variable holding a small value such as 0.0001. This is necessary to ensure that rounding off errors do not lock the set up in track mode ad infinitum.

Under many circumstances (e.g. cascade temperature control of a barrel zone) the inner (slave) loop does all of the control activity and the outer loop is required to work primarily in feedforward with a feedback trim to ensure that the steady states are correct. Also what is required is that the temperature difference between the primary and secondary variable does not become too large. For this reason it is possible to employ the facility for setpoint or process value feedforward.

1. Setpoint feedforward is performed by setting the SP_FF_Enable flag in the master loop and setting the required value of the trim in percentage. The rest of the calculations are performed by the function block. Note that setpoint feedforward is only available with heat only instuments. This ensures that the output of the function block varies by the maximum value of the SP_FF_Trim about the mean value set by

$$\frac{\text{Master\_PID.Setpoint} - \text{Master\_PID.Span\_Low}}{\text{Master\_PID.Span\_High} - \text{Master\_PID.Span\_Low}} \text{x100}$$

This facility limits the difference between the master and slave setpoints to a user defined maximum value. Consider a temperature on temperature cascade loop. The span of both the primary (master) and the secondary (slave) loop are set to 600 degrees, say. A setpoint of 450 degrees on the master is translated to 450/600 or 75% output level for the master loop. If a trim level of 5% is chosen then the output of the primary can vary between 70% and 80%. This scaled for the slave (secondary) loop setpoint implies a maximum variation of 420 degrees to 480 degrees for the slave setpoint. Changing the setpoint of the master to 500 degrees is followed by an immediate shift in the slave setpoint by the appropriate amount.

2. In order to perform process value feedforward set PV_FF_Enable flag and set PV_FF_Trim. The calculations are similar to that of setpoint feedforward. PV feedforward is useful in delta temperature control.

   Consider a batch reactor. It is typical to require a maximum temperature difference between the jacket and batch temperature. Process value feedforward can be used in this case where in effect the slave controller, controls the difference between process values and the master sets the base level of the main process value. Care must be taken in tuning these controllers as there are positive feedback paths which act to destabilise the loop in this configuration.

Tuners

In order to perform automatic tuning of the parameters of the PIDs in each of the loops an additional set of wiring is necessary to ensure that the outer (master) loop is aware of the conditions of the slave loop. Such a setup is performed in the 900EPC series of controllers and the block diagram is as shown in figure 9-32

Note that the variations on the setup of figure 9-31.

Track is also enabled when the inner (slave) loop is in Autotune

If the outer (master) loop is at steady-state then the monitor noise sequence in the slave loop should not cause any changes in the output level during the first minutes to avoid disrupting the state. For this reason if the absolute value of the error is less than 1 percent of the span then the initial setpoint at the begining of the slave loops monitor noise is set to its current process value. There after during the first minute the setpoint can be changed to a desired value for the Autotune.

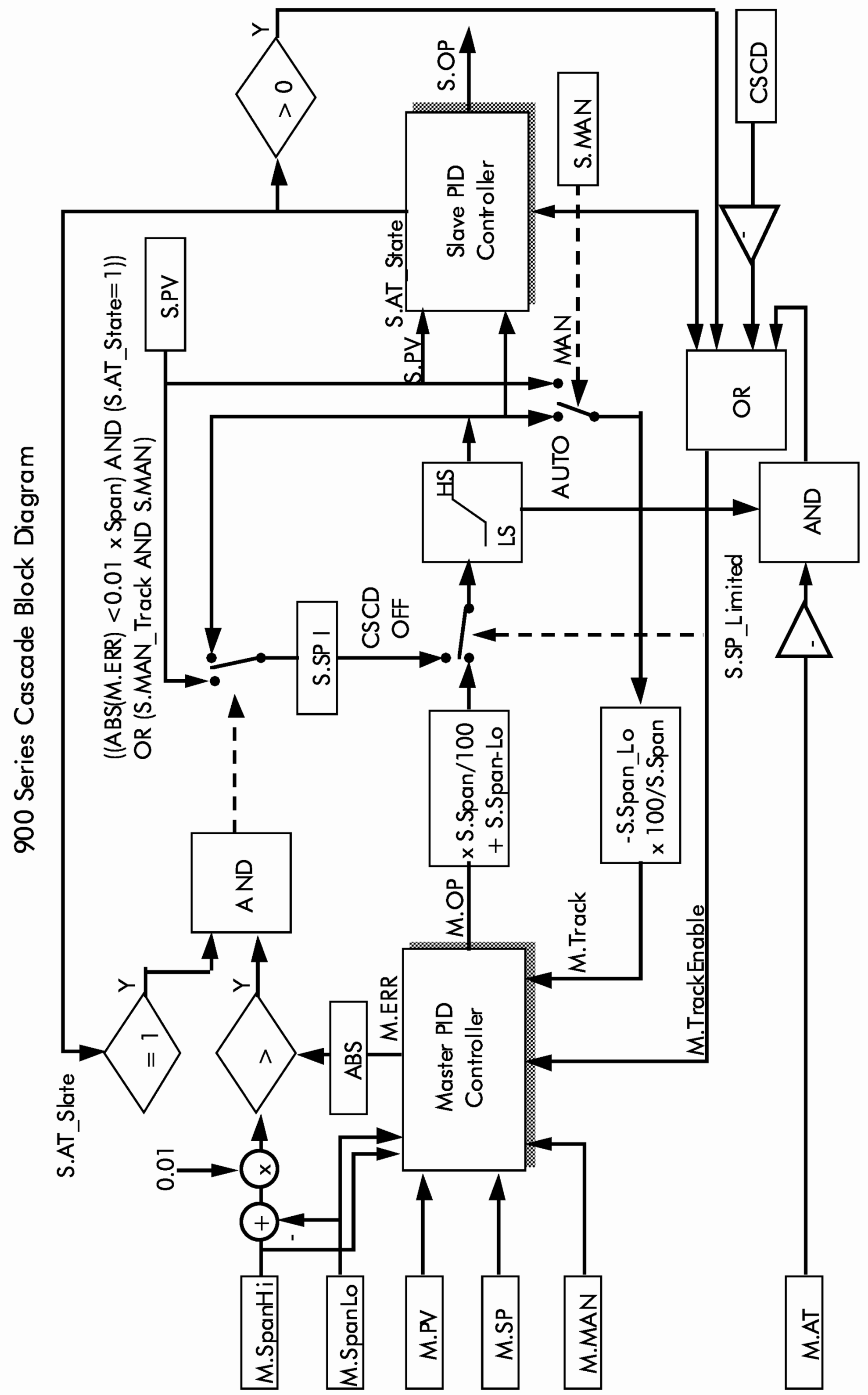


Figure 9-32 Cascade of PID_Auto function blocks

In some versions of the firmware Autotune sequence is fixed such that it generates outputs between Output_High and Output_Low irrespective of any other settings. For master loops with limited trims this may not be acceptable. In order to overcome this problem during Autotunes the power output limts Output_High and Output_Low must be set to limited values explicitly.

Note that:

1. Autotune should always be performed from the inner loops outwards.
2. It is possible to set both inner and outer loops to adaptive tune state (Tune_Type = 4). However, for secure tuning the adaptive tuner disables parameter updates if the setpoint is continuously changing in a random manner. The adaptive tuner of the slave loop may therefore not change its parameters if the master loop is very active.
3. Care should be exercised in the application of adaptive tune to the master loop, as the adaptive tuner on this loop will attempt to compensate for instabilities of the slave loop by adjusting the master loop parameters.

Controller structure

The actual structure of the cascade control is very much an application specific issue. There are however certain simple issues which have to be considered.

1. Integral Windup: So long as the ouput of the secondary controller has not hit any of the saturation limits there are no problems.

   However, if the output of the secondary controller is at one of the saturation limits for an appreciably long time then the integral of the primary loop is very likely to windup. A simple solution to overcome this problem is to inhibit integration in the primary loop when the control in the secondary loop hits saturation using the integral inhibit function. Another possible solution is to put the primary controller in track when the secondary is at limit.

   It is necessary to back-calculate the value of secondary setpoint which would have just caused the control to saturate. This should be reconverted to a percentage of the primary output and used as the track value. This is more complicated than stopping integration which has a similar effect.
2. PI and PD action: Ordinarily, it is unnecessary to apply full PID control to both primary and secondary control loops. Usually, the main aim is to have a sufficiently tight inner loop. This can be achieved with proportional only or PD control. The derivative however must operate on the secondary process value as opposed to the secondary error. If the derivative acts on the secondary error the control will be far too active because of continuous changes of the secondary setpoint (i.e. primary control). The primary control should by the same token be PI control to reduce the total activity of the control due to a possible derivative action.

3. Primary and Secondary Sensor Break: The sensor break strategy for a multi-loop system may have to be quite different to single loop strategy where the control output is set to some safe value. When the inner loop sensor breaks it may be possible to continue with the outer loop acting as a low gain single loop controller and vice versa. The user programs can cater for a variety of safety mechanisms.

The basic operation

Typically both master and slave loops should be configured for setpoints tracking process values when in manual. This can be performed by the SEL_REAL function with Master_PID.Manual or Slave_PID.Manual as the gate. Note that this is an extra requirement over and above those described in the previous sections.

Changing the slave to Auto causes the controller to initially retain its last output value and the slave setpoint is set to the slave process value at that instant, thus passing bumplessly from manual to automatic. A subsequent change to automatic control in the master loop, causes the master output to be set to a value equivalent to the existing slave setpoint, while the master setpoint is set at master process value. Once the changeover to fully automatic cascade control has been achieved, all changes to the process are made by changing the master setpoint.

Should manual control of the process be necessary, then putting the slave in manual causes the slave output to be frozen at the last value until the slave output is changed by the operator. While the slave is in manual, the master output is set to track the scaled slave process value so that when the slave returned to auto both master output and the slave setpoint are equal to the slave process value. The master setpoint is however not affected.

If setpoints do not track process values in manual process 'bumps' may occur on changeovers.

For safety reasons, manual operation must override any other control activity (e.g. Autotune).

Ratio control

It is very rare that the loops of a control system are totally independent of each other. Mixing and blending are the most obvious examples for this. Most basic ratio applications control the ratio of one flow to another -- it should be appreciated that many flow measurements are either noisy or heavily quantised, or both. As a result input filters should normally be configured for all ratio applications. Here, we basically have a single loop PID whose setpoint is derived as

```
Pid.Setpoint := PV_Lead * Ratio_Gain + Bias;
```

Clearly, PV_Lead is the leading process value, Ratio_Gain itself can be either fixed or adjustable and the whole thing could act as basic adjustment on a fixed bias term.

Dosing applications are a natural extension of the basic ratio application, with optional control of the lead flow variable, requiring a second PID loop. The lead flow is normally controlled to a fixed setpoint, although setpoint programming may also be required depending on the application area. The dosing flow setpoint is typically adjusted in proportion to the actual lead process value with an external manual trim depending on some offline measurements.

Air/Fuel ratio control in burners is another obvious example. In this example, it is common to have the air flow loop proportional to the fuel flow loop. Typically, the air loop is tuned tightly to react rapidly and the fuel loop is tuned such that it responds slowly to changes of setpoints, etc. This implies that the setpoint of the air flow loop is always set to a fixed ratio from the process value of the fuel flow loop. The simple setup of figure 9-33 shows a basic air fuel ratio loop.

Combustion is often required to be air rich in order to minimise the chance of smoke/soot production in the flue, so the ratio trim can be used to adjust the ratio over and above the stoicheometric value. To maintain an air rich atmosphere however, the controller is required to switch between air lead when a rise in temperature is demanded, and gas lead when reducing temperature. In this case, both loops are tuned with similar reaction times. Typically, the fuel setpoint comes from a master controller controlling the temperature of the furnace and an oxygen analyser is used to trim the actual ratio so that the appropriate amount of excess oxygen is present in the flue.

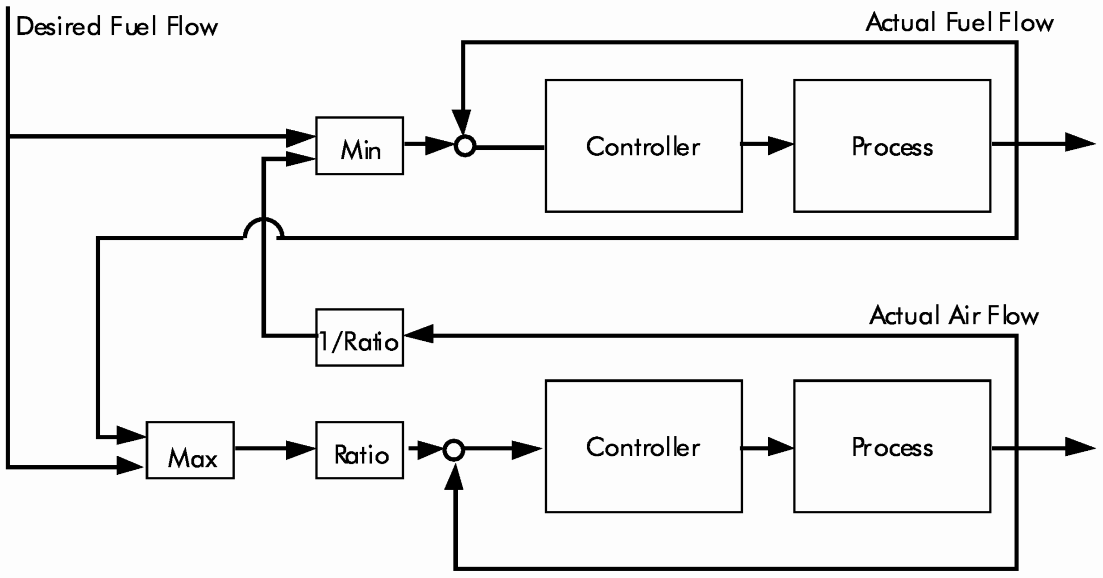


Figure 9-33 A simple ratio control structure

Boiler manufacturers have designed their burners to be flexible regarding whether oil or gas is being burned and provide simple switching between these fuels. This problem can be overcome by having ratio and gain switching using select functions in the simple case and using recipes in the more complicated cases.

Caution

Care must be taken with rapid changes to fuel and air flow since ratios which deviate substantially from the setpoint are potentially explosive. The burners, in general have built-in safety mechanisms but a certain level of security can also be obtained through the user programs in software.

Tuning

Much like the cascade situation the user software should provide facilities for engaging and disengaging ratio. Auto tune should typically be performed when ratio is not engaged. In most cirumstances this means that auto-tune precedes normal operation. Typically, this may have to be done with tighter limits on the outputs of the function block during the Autotune.

Feedforward control

Feedforward like feedback is a very powerful tool in control of processes. Feedback is in essence a retrospective type of control: the effect of disturbances are known before the control system can react to compensate for the imbalance. There are many occasions where it is possible to predict the effect of external disturbances and compensate for these in advance. Consider control of heating, ventilation and air conditioning systems in large buildings. There are many external influences which are known or are measurable. The normal period of occupancy of the building is relatively well known, the effect of solar gains are to a large extent measurable, temperature and humidity of the air intake from outdoors can be easily measured. Instead of putting the burden on feedback alone it is possible to compensate for the effect of these using feedforward whereby the control signals are a function of these external influences as well as the normal feedback signals of room temperatures and humidity. Briefly, if the source of external disturbance is known and can be measured accurately then a combination of feedforward and feedback gives best results.

Unlike feedback there is no danger of instability with feedforward. There is a price to be paid, however. Firstly, the complexity of the control scheme increases as the number of feedforward signals increase: each one requires a transducer, transmitter, probably some signal conditioning -- each of these add to the complexity. Secondly, it is possible to have worse performance with feedforward than without. This is because the feedforward action is an open loop action. Consider the example of the HVAC (Heating Ventilation and Air Conditioning) system. If the air intake is cold then the chillers should be set to work less actively. If the feedforward compensation is to make the chillers work harder, then the humidifiers and the heaters have to work that much harder to compensate for this incorrect feedforward action! This implies that for reasonable feedforward control good process knowledge and process models are required. Another important point is that given the fact that no model is perfect and that feedforward has no self-correcting mechanism we must always combine feedforward with feedback.

Typically, there are two possible scenarios

Feedback with feedforward trim, or

Feedforward with a feedback trim.

The type of model one has to construct for either of the above cases is different. Consider a simple temperature control in a rotary drier. The moisture content of the incoming feed can for example be very simply measured by the load currents in the belt drives which deliver the feedstock to the drier: the wetter the feedstock, the heavier it is and the heavier the load on the belt, the higher the motor torques and hence higher drive currents are necessary. This signal can then be used for feedforward control. This can be potentially very useful as usually there is a significant delay between application of gas and an increase in the temperature in the drier.

If feedback is used alone, the incoming wet feedstock will have to first cause an appreciable drop in the drier temperature and only then the control can begin to react. But the influence of the corrective action of the control can not be seen until the delay time of the drier has passed by and so forth. With feedforward corrective action can be taken as the wet feedstock enters the drier. The short term corrections are crucial to be reasonable as the feedback controller has no way of correcting these and there is a potential for setting up oscillations if the short term feedforward corrections cause overcompensation.

It is important to define the correct short term model. The medium and long term situation is somewhat different. The temperature controller is designed to compensate for medium/long term variations in a drier -that is why the controller is there in the first instance. Therefore, it is unimportant if the long/medium term models for feedforward are incorrect in this case, feedback will take care of that. By long and medium we refer to time constants of the order of one closed loop time constant to 5 closed loop time constants (note the relationship between the integral time and closed loop time constant is about 3 to 1).

When feedforward is the main controller and feedback is the trim then it is important to get the feedforward compensation right for medium terms effects as well. The steady-state level can be left to the feedback trimmer. Typically in these cases the PI trimmer is given limited authority too (i.e. the control output signal has tight bounds).

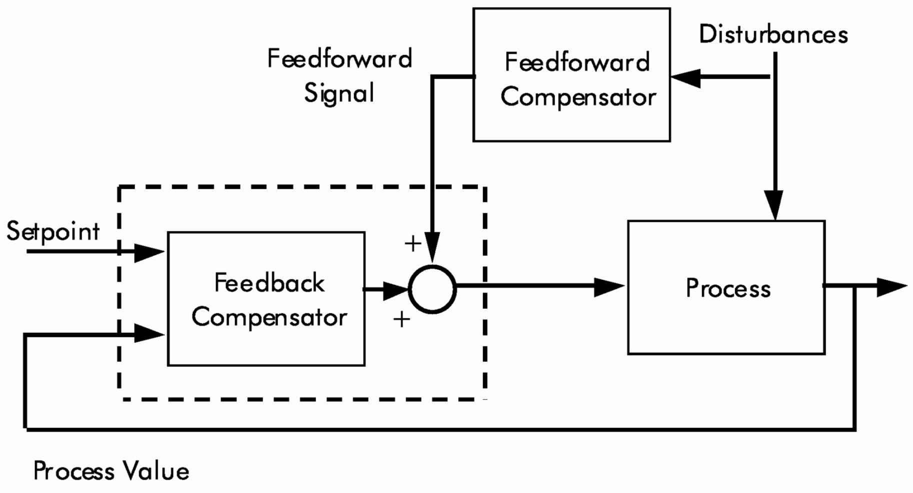


Figure 9-34 Combined feedforward and feedback control

Often it is possible to get a long way using fairly simple feedforward compensators -- the marginal benefits of subsequent improvements become so low that it is no longer economical to pursue that line.

Basic feedforward scheme

Figure 9-34 shows a typical block diagram of feedforward control. Note that the feedback and feedforward control signals are added together and subsequently processed to ensure such effects as manual/auto bumpless transfer, integral desaturation, etc. Depending on whether internal trim limit is required or not the following wiring can be used for feedforward control and a PID block.

No PID Trim Limit Required

```
Pid.Feedforward := FeedFW.Val;


PID Trim Limit is Required
Pid.Feedforward := FeedFW.Val;
Pid.Output_High := MIN_REAL(IN1:= FeedFW.Val + Trim.Val,
                       IN2:=High_Limit.Val);
Pid.Output_Low  := MAX_REAL(IN1:= FeedFW.Val - Trim.Val,
                       IN2:=Low_Limit.Val);
```

Note that the limiting described above does not take the effect of relative channel 2 gain into account.

FeedFW is the compensation signal calculated for feedforward. There are several issues which are important in the design and tuning of feedforward compenstors.

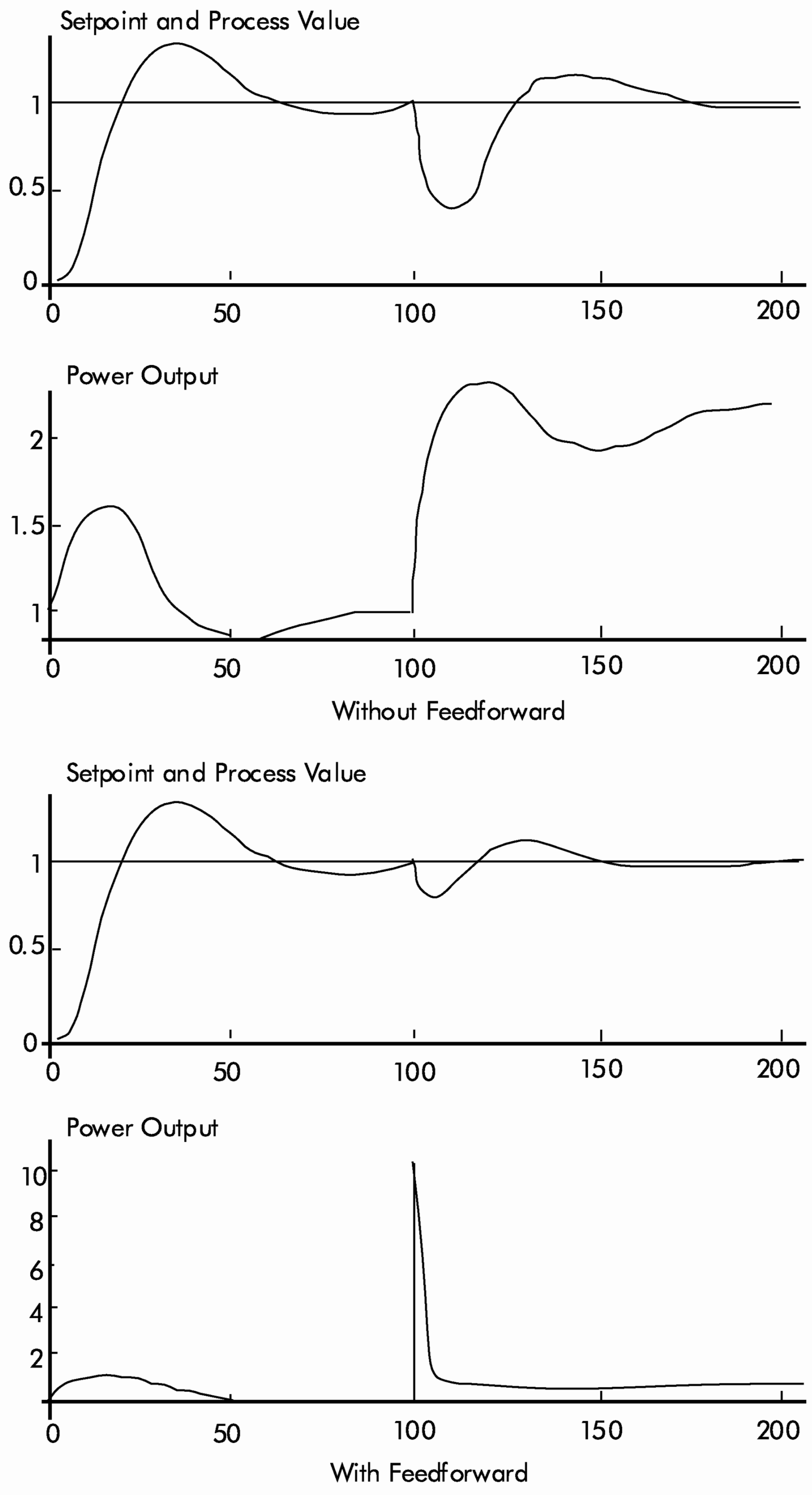


Figure 9-35 Effect of disturbance and feedforward

Consider a straight forward temperature control in an environmental chamber. The effect of a temperature drop in the air intake on the final temperature is probably not dissimilar from the example in figure 9-32. The top half of the figure shows the response of the feedback controller to the step demand and the disturbance. The bottom half shows the same feedback controller assisted by a lead compensator as a feedforward controller. Note the marked improvement in the disturbance rejection properties.

The response is typical of a large class of processes. There is a dead time during which the effect of the change has not reached the sensor. Then there is a transient period which can be approximated by some kind of a lag where the temperature reaches its new steady state value. Each of the responses can be approximated by three numbers: a gain (K), a dead-time (D) and a lag time constant (T). The method of calculating these from the reaction curve is identical to that described earlier. The feedforward compensator should therefore perform a back calculation for computations of its control signal. There are three basic cases: dead-time from the disturbance to the process value (Df) is larger than the dead-time from the controller output to the process value (Dp), Df is equal Dp and finally Df is smaller than Dp. Each case is dealt with in turn.

When delays are matched exactly there is no requirement other than attempting to compensate for different reaction rates (i.e. time constants). This means that the compensator should be of the form

$$u_{ff}(t) = -\frac{Kf \ \mathrm{x} \ (1+sTp)}{Kp \ \mathrm{x} \ (1+sTf)} v(t)$$

which is a lead or a lag style compensator. See the next section for the realisation of this transfer function in PC3000. Tp is the time constant of the control signal to process value and Tf is the time constant of the feedforward to the process value. Ks are the appropriate gains. Note that gains can be negative as well as positive.

When Dp is smaller than Df then the best policy is to delay the compensation for the duration of the difference between the two dead-times and then proceed as in the previous case. The compensator transfer function is therefore

$$u_{ff}(t) = -\frac{Kf \ \mathrm{x} \ (1+sTp)}{Kp \ \mathrm{x} \ (1+sTf)} e^{-s(Df-Dp)} v(t)$$

The final case is where the effect of the feedforward is more rapid than the feedback signal. There is nothing one can do regarding the initial transient: the control signal can not reach the sensing point in time. The rest of the transient can however be improved somewhat. Broadly speaking the numerator time constant should be the control to process value reaction times and the denominator the feedforward to process value reaction time. Feedback will also be strongly effective in this case.

In many cases especially where time constants are commensurate a straight forward gain is sufficient. As was discussed in the introduction it is usually better to err on the side of caution. Overcorrecting is usually worse than not correcting enough. Once the appropriate feedforward gain is measured from process data one can reduce the feedforward gain a little: very good results can be obtained in this way.

As with all model based strategies the better the model the better the overall performance but most of the time simple models can give very good results.

A note must be added regarding the methods of obtaining these models.

Feedforward signal measurable and manipulatable: This is a classical case in most of multi-loop designs in control of processes.Feedforward is used in these cases to decouple the loops as much as possible. Feedforward is taken from either setpoints or controller outputs in these cases. It is possible to obtain quite reasonable models (noise levels and process conditions permitting).

Feedforward signal not manipulatable: In these cases it is possible to run the system with a feedback controller only for a sufficiently long length of time so as to capture enough data which contains shifts of levels in the feedforward signal. It is then possible to low pass filter the acquired data and take a reasonable guess at the gains involved. The gain and time constant for the control signal to process value can be obtained via open loop tests. This leaves the denominator (lag) time constant. This will have to be done by trial an error or by physical considerations.

Obtaining step response data as mentioned here is prone to large errors. By far the most reliable approach is proper statistical time series analysis and application of appropriate test signals. This however requires very sophisticated tools and an extensive study period for it to be of any value.

More advanced control strategies

Alternative PID structures

It is very difficult to design a PID controller which has good disturbance rejection properties (e.g. quarter decay ratio damping) as well as good reference following. Such a design typically requires two degrees of freedom but the standard PID only allows one. With PC3000 it is relatively easy to recover the extra degree of freedom.

Typically, the controller is tuned to give good regulation performance (e.g. via Ziegler-Nichols rules). PIDs tuned via Ziegler and Nichols are known to give poor reference following -- frequently with unacceptable overshoot. Three solutions are possible

1. Detune the controller parameters to give a compromise response for both situtations;
2. Use ramps or pre-filters on the setpoint to reduce the magnitude of the overshoot;
3. Use setpoint weighting. Setpoint weighting is discussed by Hang, Astrom and Ho [5] and the control law is given by

$$\text{Output} = \frac{1000}{\text{Span} \times \text{PB}}\left((a \times \text{SP} - \text{PV}) + \frac{1}{\text{Ti}}\int e(t)dt + \text{Td}\frac{de}{dt}\right)$$

The derivative may be chosen to act on process value instead of the error as is more usual in the process industry. Note that the difference between the normal PID structure and this so-called two degrees of freedom version is the choice in the weighting term a. This structure can be realised with the straight forward wiring described below

```
Pid.Feedforward := b.Val * Pid.Setpoint;
b.Val           := (a.Val-1.0)*10000/(Pid.Prop_Band*(Pid.Span_Low-
                                         Pid.Span_High));
```

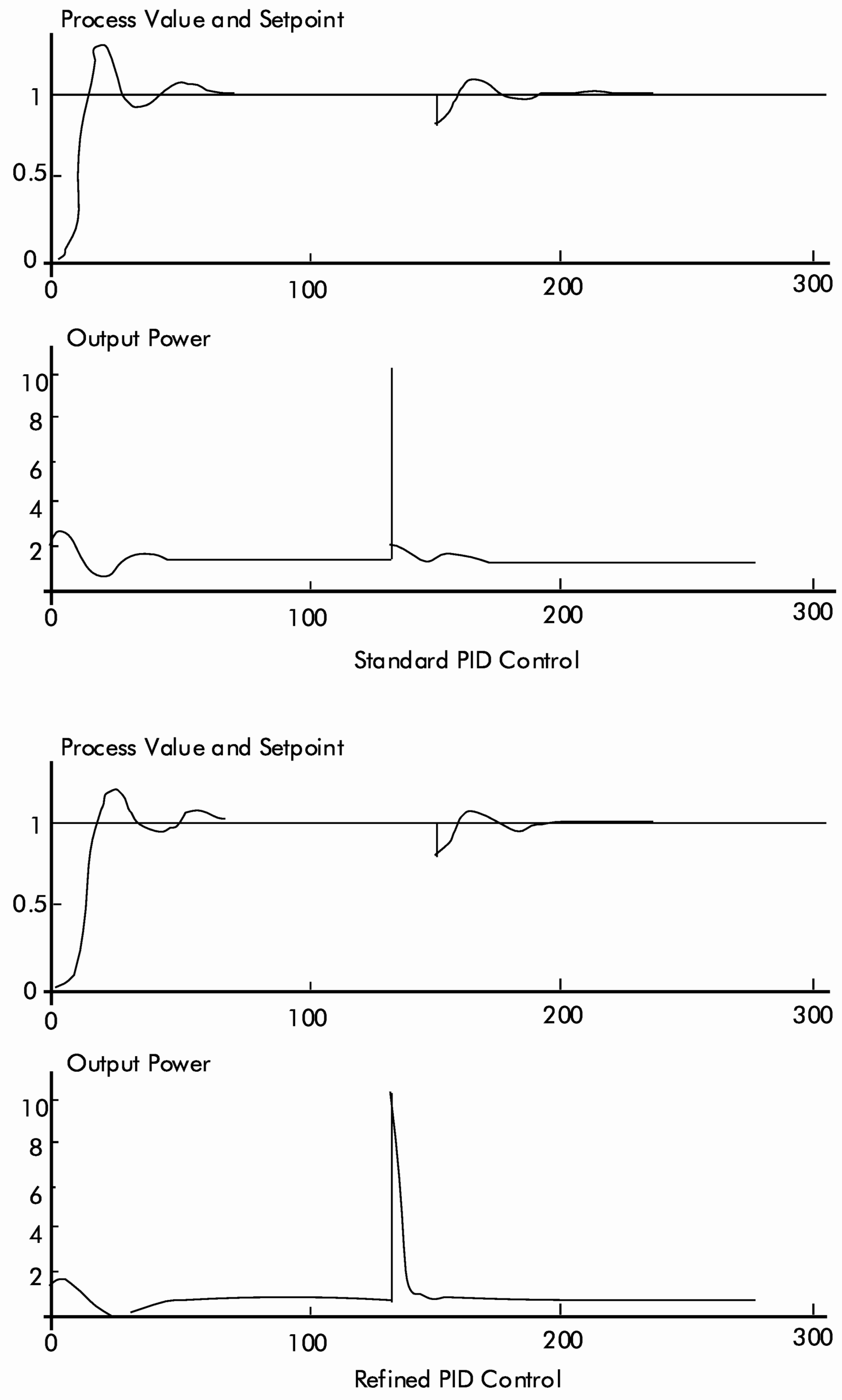


Figure 9-36 Standard and refined PID control

The weighting b need only be evaluated anytime either a or the proportional band setting or the span values are changed. Setting a to zero is equivalent to the proportional term operating on the process value only.

Figure 9-36 shows an example of PID control tuned via the Ziegler-Nichols rules and the refined method. Note that the setpoint response with the refined method is far better but the good disturbance rejection is unaffected. Clearly, it is possible to apply the same technique to the derivative term (i.e. part of the derivative could operate on error and part on process value instead). Note, this is not as useful in the case of the derivative term.

It is common to use the rate of change of setpoint as a feedforward term during a ramp following. This can be done very simply by wiring the rate parameter of the ramp and the feedforward with an appropriate adjustable gain.

Process value switching and tracking

It is quite common to have to control a particular variable (e.g. pressure) during a particular regime of the machine and subsequently to switch over to another variable (e.g. tension) during the next regime. This is also quite common with sensors which have a limited range. The process value has to be switched from one source to another depending on the conditions. Depending on the situation, a variety of solutions are available.

Process Value Hard Switch Over: If one is simply switching between banks of sensors and they all measure the same characteristics of the process then the simplest solution is to perform the switch over with a SEL_REAL function. In order to avoid a bump at the switch over it is important to set the Debump parameter simultaneously.

Process Value Soft Switch Over: In many cases the switchover can not be performed as a hard changeover operation described earlier. In these cases it is possible to define a region where the two process values are fed to the controller in proportion.

```
Pid.Process_Val := Weight.Val*ANIN1.Process_Val + (1.0-Weight.Val)*
ANIN2.Process_Val;
```

In this way the switchover can be performed smoothly. The user variable Weight can be adjusted linearly with changes in one of the process values.

PID Tracking Facility: Where the controller setting, setpoint, as well as controller process values are switched it is usually easier to have two controllers with one tracking the other while not active. This is achieved by wiring the PID controllers back to back with a switch connected to the track enable signal. Here we assume there are two single channel controllers.

```
Pid_1.Track_Enable := NOT Pid_1_Enable;
Pid_1.Track_Value  := Pid_2.Ch1_Output (* Feedback *);
Pid_2.Track_Enable := Pid_1_Enable; Pid_2.Track_Value
      := Pid_1.Ch1_Output;
```

In many cases the selection is performed automatically. Consider a case where two variables say the jacket temperature and batch temperature (PV1) of a batch reactor need to be controlled. The first PID attempts to control the batch temperature at the desired setpoint and the second PID attempts to control the jacket temperature at PV1 plus a fixed difference. The control actuator (heating element) is driven from the PID which is providing the minimum output.

A similar situation arises with a valve used to control flow as well as pressure upstream in a drum boiler. A simple way to achieve this is via the following wiring.

```
Pid_1.Track_Enable := SEL_BOOL(G:= Pid_1.Track_Enable,
                               IN1:= 0,
                               IN0:= Pid_1.Ch1_Output > MINOP.Val
                               (*Feedback *) + EPS.Val OR
                      Pid_1.Ch1_Output < MINOP.Val       (* Feedback *) -
EPS.Val);
Pid_1.Track_Value := MINOP.Val (* Feedback *);
Pid_2.Track_Enable := SEL_BOOL(G:= Pid_2.Track_Enable,
                               IN1:=0,
                               IN0:= Pid_2.Ch1_Output > MINOP.Val(*
Feedback *) + EPS.Val OR       Pid_2.Ch1_Output < MINOP.Val(* Feedback *)
-                              EPS.Val);
Pid_2.Track_Value := MINOP.Val (* Feedback *);
MINOP.Val := MIN_REAL(IN1 := Pid_1.Ch1_Output, IN2 :=Pid_2.Ch1_Output);
```

EPS is a small positive number, and is used to avoid round-off problems with floating point arithmetic. The sequence of operation will therefore be as follows. If the output of the PID was not selected at the last sample the function block is put in track state and is set to track the last minimum value. If the output of the controller was selected on the other hand, the PID is set to choose its output accordingly. If the controller was set to track at this sample, it will be pulled out of track at the next. Such a setup mimics an incremental version of the PID controller with each tracking the other.

It may have been easier to perform some of these decisions in a SFC step instead of using function block wiring.

Lead, lag and lead-lag compensation

In many application areas, especially feedforward control as discussed in section 8-9 there is a need to have functions which perform lead or lag or lead-lag compensators.

A lead or a lag block can be represented in the form

$$OP = \frac{1+sT_1}{1+sT_2} \times PV$$

This can be performed with the following wiring using the Lag1 function block.

```
lag1.Input    := PV.Val;
Gain.Val      := TIME_TO_REAL(IN:= T1.Val)/TIME_TO_REAL(IN:=T2.Val);
OP.Val        := Gain.Val * (PV.Val - lag1.Output) + lag1.Output;
```

The "Gain" parameter need not be computed in wiring as it can be computed in the SFC every time the time constants are adjusted. The Lag1 function block instance should typically be associated with a task with intervals less than T2/10. Setting the initial condition for this is very simple. Lag1 function block output tracks its input in Track state, so engaging lead/lag compensator is equivalent to changing the Track state to Limit state.

A lead-lag or a lag-lead block is simply cascade of a lag followed by a lead or vice versa.

Model based control and delay compensation

PI and PID control are not suited to control of delay dominant systems. If tight control is required frequently, a model-based strategy may have to be used. Internal Model Control (IMC) of Morari and Zafiriou [7] provides a useful framework for design of model-based control startegies within PC3000.

A simple smith prediction scheme

The Smith predictor scheme translated into the internal model control framework is represented by the block diagram shown in figure 9-37.

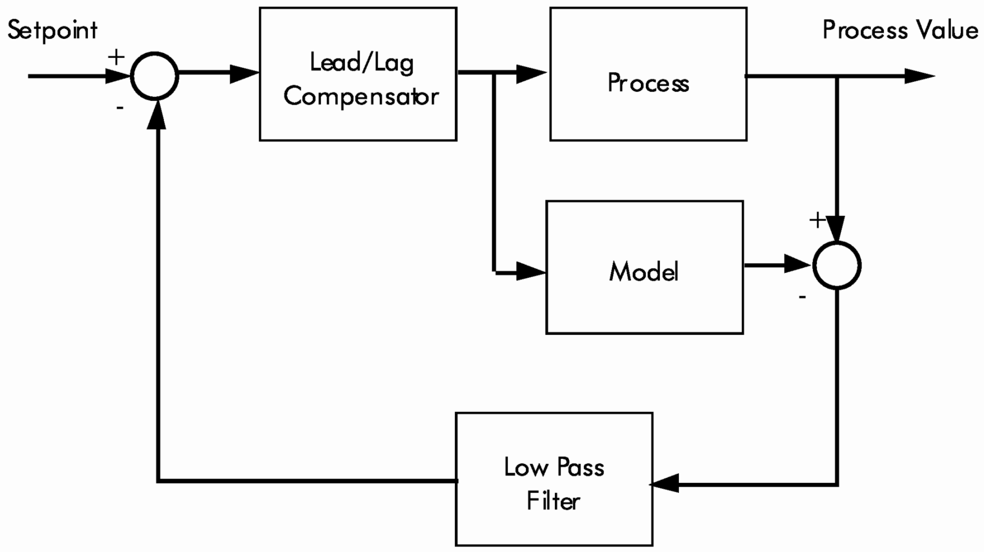


Figure 9-37 A Smith predictor

Note that the controller for the IMC design consists of three distinct parts

An Internal Model: The internal model is intended to replicate the process value as closely as possible. Two causes of mismatch between the model process value and the actual process value are plant/model mismatch and the effect of load disturbances. Both of these are reflected in the error signal which is fed back via the low pass filter. If there are no modelling errors or disturbances then the feedback signal will be zero. This implies that the reference following is then directly adjustable through the Lead/Lag compensator.

Note that provided there is no modelling errors the closed loop stability is guaranteed if the process, its internal model and the compensator are all stable. Moreover, the resulting control scheme is guaranteed to have integral action if the steady state gain of the Lead/Lag compensator is set to be the inverse of the steady state gain of the model. The model is typically first or second order with dead-time. Although, in principle there is no reason why the model has to be of low order or even linear. In fact, quite sofisticated models can be built with the simple tools available.

A Low Pass Filter: This low pass filter is intended to filter out the effect of noise on the process variable as well as providing a certain degree of robustness to unmodelled dynamics of the process such as the effect of nonlinearities and higher order dynamics. The time constant of the filter must be large enough to reduce the noise level and small enough so as not to degrade the disturbance rejection properties excessively. The steady-state gain of the filter must be set to unity.

A Lead/Lag compensator: The lead/lag compensator is used to set the "closed-loop" response time. The compensator typically takes the form of a straightforward lead or lag compensator. Lead is used to speed the closed loop response compared to the open loop and lag to slow the closed loop compared to the open loop response. Typically, a lead with a factor of 2 between time constants is used where the numerator has the same time constant as the open loop process and the numerator has half the time constant.

A simple method of obtaining a reasonable model is by the reaction curve method described eariler. Consider the step response in figure 9-38. The internal model of the system is set to

$$M(s) = \frac{K_m e^{-sDm}}{(1 + sT_m / 2)^2}$$

The implementation of the delay can be provided using the Shift_Real function block. In this example, the estimated time delay is divided by 10 and this sets the clocking rate of the output of the lead network in to the Shift_Real function block. The wiring for each of the separate section of the IMC design is as follows.

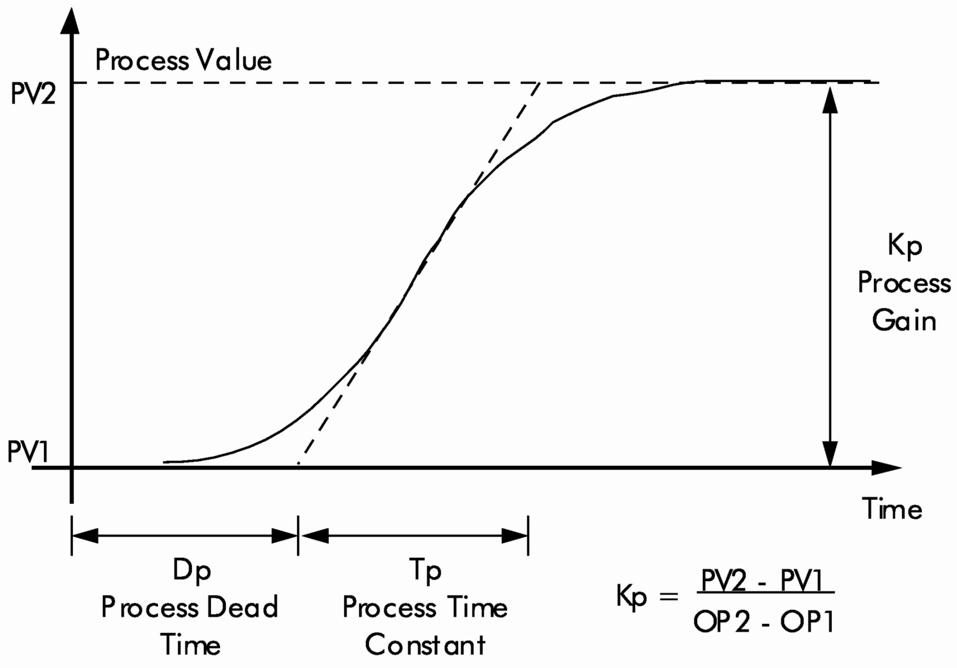


Figure 9-38 A typical step response

Internal Model

```
OnDelay.Input      := OffDelay.Q_Output;
OffDelay.Input     := NOT OnDelay.Q_Output (* Feedback *);
Delay.Clock        := OnDelay.Q_Output;
Delay.Process_Val  := Lead.Val;
Model_1.Input      := Delay.Output_11;
Model_2.Input      := Model_1.Output;
```

The program time on each of the on and off delay timers is set to estimated delay divided by 10. The time constant of each of the model lags is set to estimated time constant divided by 2. Lead.Val is the limited value of the lead compensator given below.

Lead Compensator

```
lag1.Input    := Setpoint.Val - FilterErr.Output;
Gain.Val      := TIME_TO_REAL(IN:= T1.Val)/TIME_TO_REAL(IN:=T2.Val);
OP.Val        := (Gain.Val * (PV.Val - lag1.Output) + lag1.Output)/
                    Mod_Gain.Val;
Lead.Val      := MIN_REAL(IN1:= MAXOP.VAL,
              IN2:= MAX_REAL(IN1:=MINOP.VAL,IN2:=OP.Val));
```

Typically, T1 is set to Tm and T2 to Tm/2

FilterErr is a lag1 function block instance with a time constant set typically to

(Tm + Dm)/2

Low Pass Filter

```
FilterErr.Input := ActualPV.Val - Mod_Gain.Val * Model_2.Output;
```

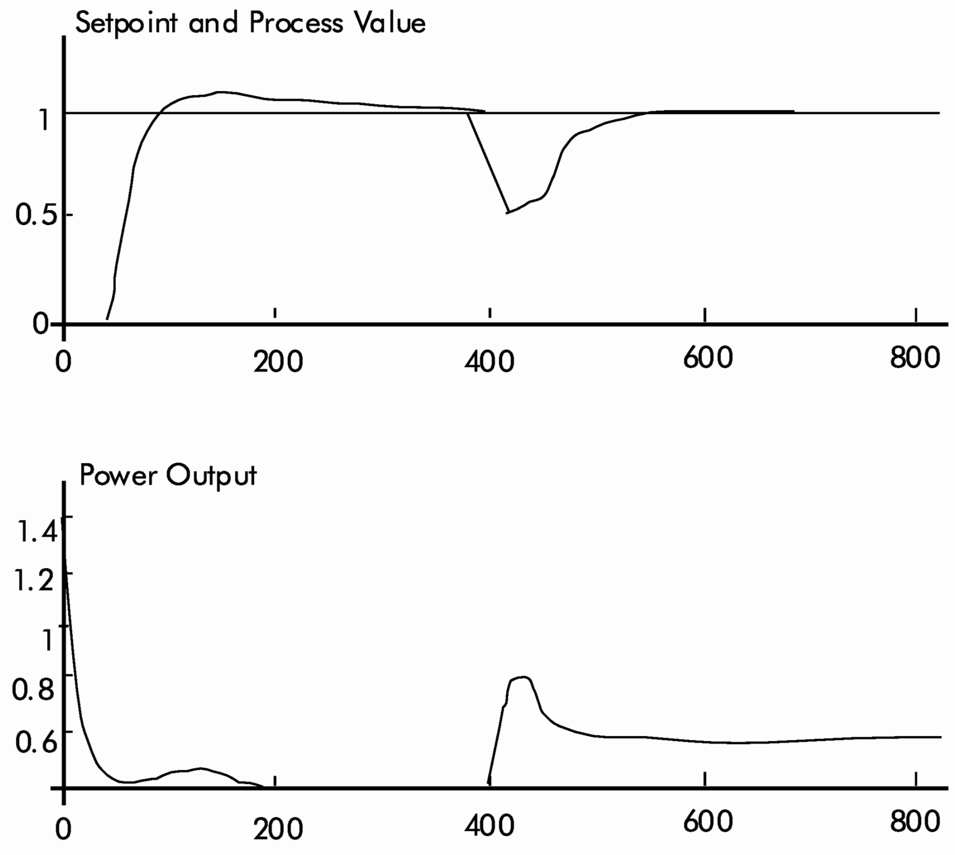


Figure 9-39 Internal model control of a time delay process

Figure 9-39 shows the response of system using the simple IMC strategy. In this example the process model does not match the actual process but the response of the controller is very good. The response can be compared with a well tuned PID for the same process as shown in figure 9-40. Clearly, the step response shows a great improvement for the IMC control but the disturbance rejection properties of the two schemes are similar.

Inferential control

Frequently, it is impossible to measure the control variables directly. The most prominent example for this is the growth of "bacteria" in fermentation processes for manufacturing products, such as penicillin. Measurements of the colour and glaze in ceramics is another example. In many cases it is possible to develop a dynamical relationship between the variable which cannot be directly measured on-line and some other environmental variables such as temperature, pressure, humidity, oxygen level, etc. In such cases it is possible to use PC3000 for the inferential control of this secondary variable.

Inference of the unmeasurable variable can be on the basis of static or dynamic relationships. Static relations are like the required computations to infer mass flow rates from volumetric flow rate computed from differential pressure measurements across the orifice plate and temperature/ density relationship. Another example is that of the relative humidity and dry/wet bulb temperature measurements. Consider the case where there is a dynamic relationship of temperature and composition in a product. If the dynamical model is computed empirically or from some time series study then the Lag1 and Shift_Real function block can be used for realising the emulation/ prediction equations.

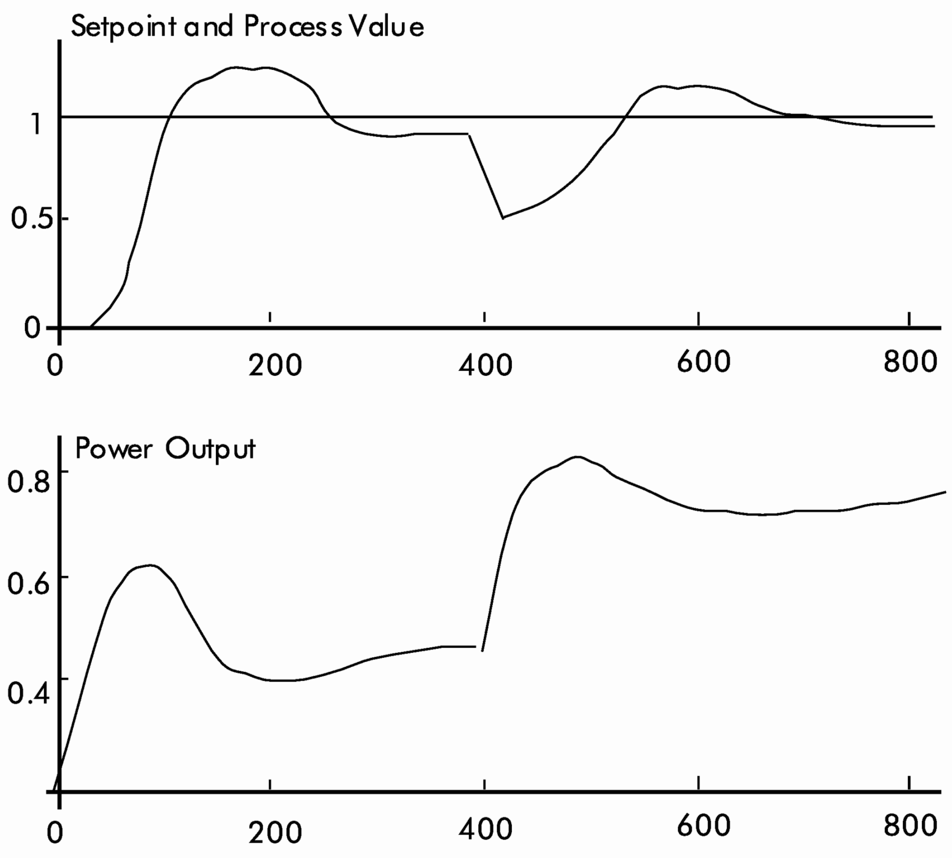


Figure 9-40 PI Control of a time delay process

Alternative control techniques

Discrete time control strategies

True discrete time or sampled data control design has received a lot of attention with process control computers. Most feedback control problems can be solved with the well tried and tested PID designs with adaptation, gain scheduling, multiple loops, etc. When these fail model based strategies can tackle some of the remaining problems. There are some class of problems, however, which render themselves to sampled data strategies. Many measuring devices such as chemical analysers, gas chromatographs, some thickness or profile measuring instruments produce measurements at fixed periodic intervals. Many other measurements are communicated via communication links and are therefore by their very nature sampled data. Others still, like peak pressure and temperature measurements in an injection moulding machine are sampled data because of the characteristics of the process. Certain computer algorithms such as optimisation, statistical analysis or even some failure detection schemes are discrete time by their very nature.

With PC3000 it is a relatively straight forward task to develop discrete time control related algorithms.

Consider a first order Infinite Input Response (IIR) discrete time low pass filter. This can be very simply done by the wiring below.

```
PVf.Val := PVf.Val + 0.1 * (Input.Val - PVf.Val);
```

This is a low pass filter with a time constant of 1/0.1 or 10 samples. The actual sampling interval of the filter is set by associating PVf with an appropriate task interval.

A large number of discrete time control designs result in a difference equation of the form

$$R(q^{-1})u(t) = T(q^{-1})w(t) - S(q^{-1})y(t)$$

where R, T, and S are polynomials in the backward shift operator q-1 and y(t), w(t) and u(t) are the process value, setpoint and the control signal respectively. There are many ways of implementing such schemes. A simple way of doing this is by having three instances Shift_Real function block, one for each data stream of process value, setpoint and output.

```
OnDelay.Input   := OffDelay.Q_Output;
OffDelay.Input         := NOT OnDelay.Q_Output (* Feedback *);
Y.Clock         := OnDelay.Q_Output;
Y.Process_Val   := PV.Val;
W.Clock         := OnDelay.Q_Output;
W.Process_Val   := Setpoint.Val;
U.Clock         := OnDelay.Q_Output;
U.Process_Val   := OP.Val (* Feedback *);
OP.Val          := (T1.Val * W.Output_1 + T2.Val * W.Output_2 +
                    R2.Val * U.Output_1 + R3.Val * U.Output_2-
                    S2.Val * Y.Output_1 + R3.Val * Y.Output_2)/R1.Val;
```

The sample interval in this case is set by the Prog_Time variable in the on delay and off delay timers. Dahlin style controllers can be realised in this way.

One area where sampled data control is particularly useful is with systems with variable production rates. Clearly, as the production rates increase or decrease the time scales and sometimes gains of the process under control change too. Variable dead-times frequently cause serious problems for control system design. With such discrete time strategies it is possible to sample with respect to production rate as opposed to time. This means that with careful design it may be possible to convert a problem which in essence is time-varying to a position invariant one, for example. Concentration control in a pipe is an example where sampling interval can be set to be inversely proportional to the flow. As the flow increases the sampling interval decreases and vice versa. The basic discrete time controller need not change and therefore a potentially difficult problem can be solved in a simple way.

Simple adaptive mechanisms

Currently, PC3000 does not provide general purpose function blocks for building adaptive mechanisms in controllers. It is, however, possible to program simple application specific gradient algorithms for some parameter adaptation.

The general equation of discrete parameter adaptation is

$$\theta(t) = \theta(t-1) + k\varnothing(t) \in (t)$$

where q is the parameter set, k the adaptation gain, Δ is the data vector, Œ is the error between the actual and predicted value. It is quite straight forward to set up such a simple update equation.

Consider the following predictor equations:

$$y(t+1) = g_1u(t) + g_2u(t-1) + f_1y(t) + f_2y(t-1)$$

The calculations can typically be done in several consecutive steps of a macro so that the load of computations is more evenly distributed. The ST generatared by the programming station for the above example is given below.

```
(* MACRO : MAIN *)
INITIAL_STEP  MAIN : MAIN__ACTION(N) ; END_STEP
(*
S  Start
     T1
     |============i============i============i
m Monitor     m Control     m  Alarm      m UsrInfce
     !============!============!=========== T1


End
*)

(* MACRO : Control *)
STEP  Control : Control__ACTION(N) ; END_STEP
ACTION  Control__ACTION :
(*
C GetData

     T
     |
ResetTim
     T1
     |
NewPars
     T1
     |
CompPred
     T1
     |
> GetData
*)
(* CONTINUOUS *)
INITIAL_STEP  GetData : GetData__ACTION(N) ; END_STEP
ACTION  GetData__ACTION :

     Y_Val    := load.Main_PV ;
END_ACTION

TRANSITION
FROM GetData
TO   ResetTim
:=
Watch.Elapsed_Time >= SampTime.Val ;
END_TRANSITION
```

```
(* SINGLE SHOT *)
STEP  ResetTim : ResetTim__ACTION(P) ; END_STEP ACTION  ResetTim__ACTION :
     Watch_Reset       := 1(*On*) ;
     (* Prediction Error Equation *)
     Error_Val         := Y.Val - Yhat.Val ;
     (* Denominator of the Update Equation *)
     Denom_Val         := U1.Val * U1.Val + U2.Val * U2.Val + Y1.Val *
                                 Y1.Val + Y2.Val * Y2.Val + 1.0 ;
     (* Compute Update Gains *)
     K_Val             := SEL_REAL( G :=  ABS_REAL( IN :=  Error.Val ) <
     Error.Max_Val AND ABS_REAL( IN :=  Error.Val )              >
Error.Min_Val , IN0 :=  0.0 , IN1 :=                     Error.Val *
Gain.Val / Denom.Val ) ;
END_ACTION

TRANSITION
     FROM ResetTim
     TO   NewPars
:= 1; (* NULL transition - default TRUE *)
END_TRANSITION

(* SINGLE SHOT *)
STEP  NewPars : NewPars__ACTION(P) ; END_STEP
ACTION  NewPars__ACTION :

Watch_Reset   := 0(*Off*) ;
(* Update the Parameters *)
G1_Val        := G1.Val + K.Val * U1.Val ;
G2_Val        := G2.Val + K.Val * U2.Val ;
F1_Val        := F1.Val + K.Val * Y1.Val ;
F2_Val        := F2.Val + K.Val * Y2.Val ;
(* Limit the Parameters *)
G1_Val        := MAX_REAL( IN1 :=  G1.Min_Val , IN2 :=  MIN_REAL(
              IN1 :=  G1.Max_Val , IN2 :=  G1.Val ) ) ; G2_Val
     := MAX_REAL( IN1 :=  G2.Min_Val , IN2 :=  MIN_REAL(
              IN1 :=  G2.Max_Val , IN2 :=  G2.Val ) ) ; F1_Val
     := MAX_REAL( IN1 :=  F1.Min_Val , IN2 :=  MIN_REAL(
              IN1 :=  F1.Max_Val , IN2 :=  F1.Val ) ) ; F2_Val
     := MAX_REAL( IN1 :=  F2.Min_Val , IN2 :=  MIN_REAL(
              IN1 :=  F2.Max_Val , IN2 :=  F2.Val ) ) ; END_ACTION

TRANSITION
     FROM NewPars
     TO   CompPred
:= 1; (* NULL transition - default TRUE *)
```

```
END_TRANSITION



(* SINGLE SHOT *)
STEP  CompPred : CompPred__ACTION(P) ; END_STEP
ACTION  CompPred__ACTION :

  (* Compute the Control *)
     Yhat_Val          := G2.Val * U1.Val + F1.Val * Y.Val + F2.Val *
Y1.Val
                                ;
     U_Val             := ( W.Val - Yhat.Val ) / G1.Val ;
     U_Val             := MIN_REAL( IN1 :=  MAX_REAL( IN1 :=  U.Val
                                , IN2 :=  U.Min_Val ) , IN2 :=  U.Max_Val )
; (* Store the Data *)
     Yhat_Val          := Yhat.Val + G1.Val * U.Val ;
     U2_Val            := U1.Val ;
     U1_Val            := U.Val ;
     Y2_Val            := Y1.Val ;
     Y1_Val            := Y.Val ;
END_ACTION

TRANSITION
     FROM CompPred
     TO   GetData
:= 1; (* NULL transition - default TRUE *)
END_TRANSITION

END_ACTION (* Control__ACTION *)
```

Multivariable control

It is a relatively straight forward task to build a multi-variable control system using PC3000 function blocks. The PC3000 function blocks can be interconnected to construct complex control systems.

The multivariable controller need not be limited to a series of single loop controllers. Although, in practice a large number of problems can be solved that way. A mixture of model based and classical techniques can be applied here to get the best results. Feedforward can also be used for problem loops. Multiple rate designs are also catered for via the use of the PC3000 tasking system. The communications capability of PC3000 enables discrete instruments to be integrated into the overall design. This also increases the total integrity of the system.

Irrespective of the ultimate design strategy adopted it may be beneficial to examine the level of interaction in the process before attempting to design a complicated control system. This is usually done by computing the "Bristol's relative gain array". This is fully explained in reference [6,7,8]. Computation of the static and dynamic relative gains can be performed in a variety of ways. The Bristol's relative gain is defined approximately as

Incremental Gain of the channel assumming all other loops are open
Incremental Gain of the channel assumming all other loops are closed

For a more precise definition see [8]. Clearly, if there is no interaction between channels then the relative gain will be 1: there is no difference whether other process values are kept constant or are allowed to vary -- they do not influence this channel anyway. Mathematically this means that

$$PV_j = \sum_{i=1}^{N} K_{ij} \times OP_i \text{ (In Open Loop)}$$

$$OP_j = \sum_{i=1}^{N} H_{ij} \times PV_i \text{ (In Closed Loop)}$$

$$\lambda_{ij} = \frac{K_{ij}}{H_{ji}}$$

Without proper multi-variable compensation care is necessary regarding the effect of interactions. Briefly,

Loop pairing has to be done with elements corresponding to relative gains close to unity;

If the appropriate relative gain is less than 1 then integral time and proportional band settings may have to be increased;

If the appropriate relative gain is greater than 1 then only proportional bands may need to be increased;

If the apppropriate relative gain is negative then there is a strong possibility of inverse response behaviour and the action of the controller may have to be reversed. Extra care has to be taken here. Loop integrity may be lost.

Channels which have very large (in absolute value sense) relative gains are problem areas. The implication is that the process values can not be set independent of each other. There is a significant directionality present in the process.

Relative gain arrays give a clear and immediate indication of whether;

The control problem is likely to be troublesome;

The loop pairing is sensible.

For problem areas more advanced techniques such as feedforward compensation, observer based designs, or any of the extensive variety of control design methodologies may need to be applied. Realisation of feedforward and model based designs is discussed in the previous section.

Here we outline the case of a simple observer based controller for a process with 2 inputs, 2 outputs and 3 states is shown as an example.

Assume that the controller equations are

$$x(t) = Ax(t) + Bu(t) + K_f y(t)$$

$$u(t) = U_c(t) + K_c x(t)$$

x is the observed state, y is the array of process values and u the control outputs.

Kc are typically designed off-line using some CACSD (Computer Aided Control System Design) package and are transferred for realisation in PC3000 as are the elements of the state matrices. Kc and Kf should typically be designed to give some desirable properties such as stability, robustness and reduced high frequency coupling and Uc is then the output of PI controllers on each of the channels to get good steady state accuracy. If in addition trapezoidal integration is used we have

```
SX1.Val:= a11.Val * X1.Val + a12.Val * X2.Val + a13.Val * X3.Val
              + Dummy.Val * Old_SX1.Val + b11.Val * OP1.Val(* Feedback *)
              +b12.V11 * OP2.Val(* Feedback *) + KF11.Val * PV1.Val
              + KF12.Val * PV2.Val;
Old_SX1.Val:= SX1.Val(* Feedback *);
X1.Val := X1.Val + h.Val * (SX1.Val + Old_SX1.Val) / 2.0;
OP1.Val:= OP1_Bar.Val +  KC11.Val * X1.Val + KC12.Val * X2.Val +
                       KC13.Val * X3.Val;
```

The variable Dummy is used to create an algebraic loop in the calculation so that it is possible to force an order in the calculations done via wiring (in most instances these calculations are done in a step and as such the order is fixed). Dummy variable is set to zero. Variable h is the sample interval of the task. Clearly, if the designs are done in discrete time then tasking can be used to fix the sample intervals. The additional error and safety constraints usually available for control are missing but can be added by the user when needed.

PC3000 provides facilities for the implementation of control system methodologies in a very simple and safe way. It is possible for example, to devise control systems which use an observer based design and have a standard PID backup with say the appropriate PIDs in track mode when the Multiple Input Multiple Output (MIMO) controller is active. In the situation where the "new" strategy is not as successful, the operators can switch to the "conventional" scheme which is known to work.

CONTROL
PID FUNCTION BLOCK

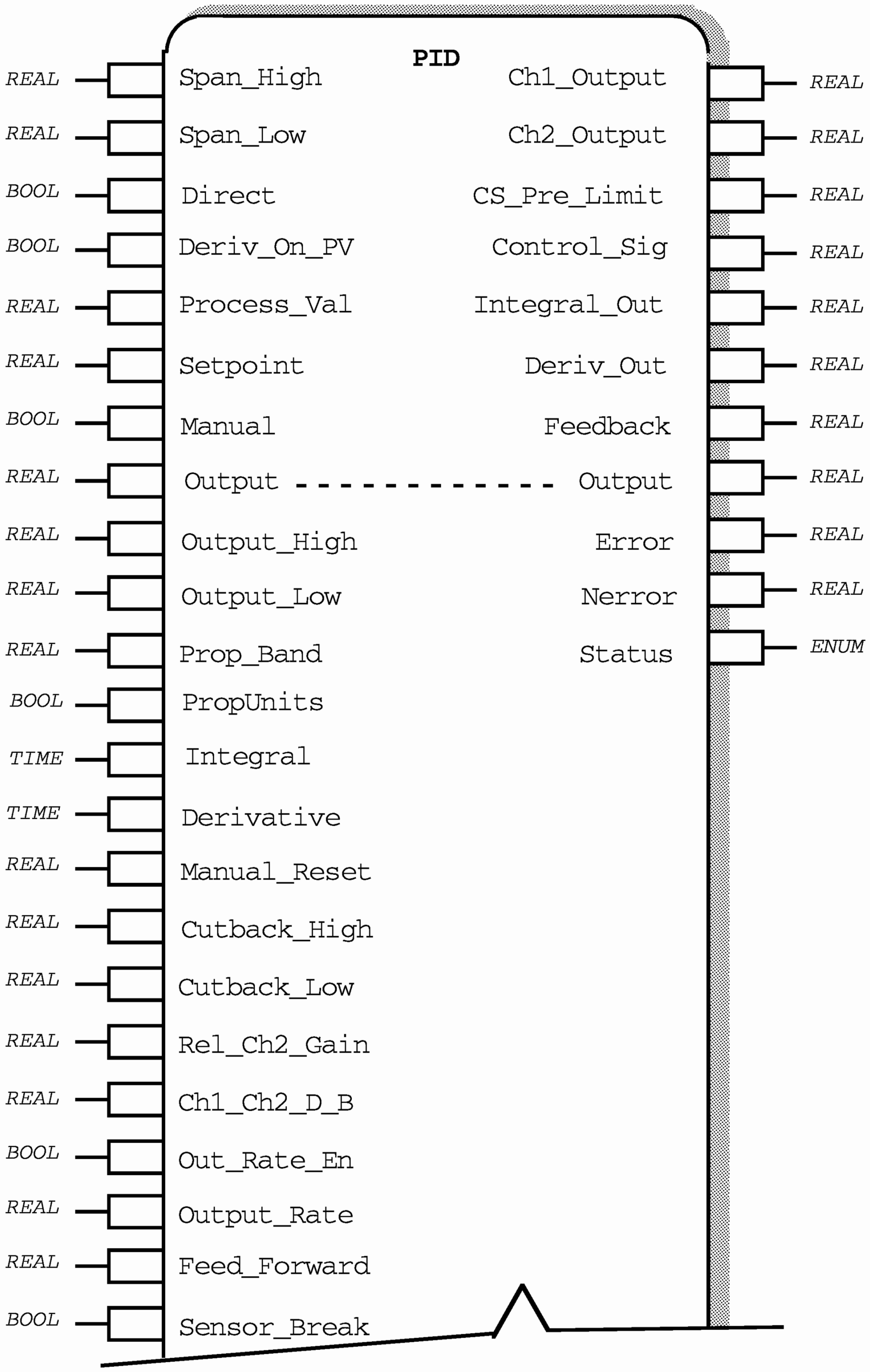


Figure 9-41 PID Function Block Diagram

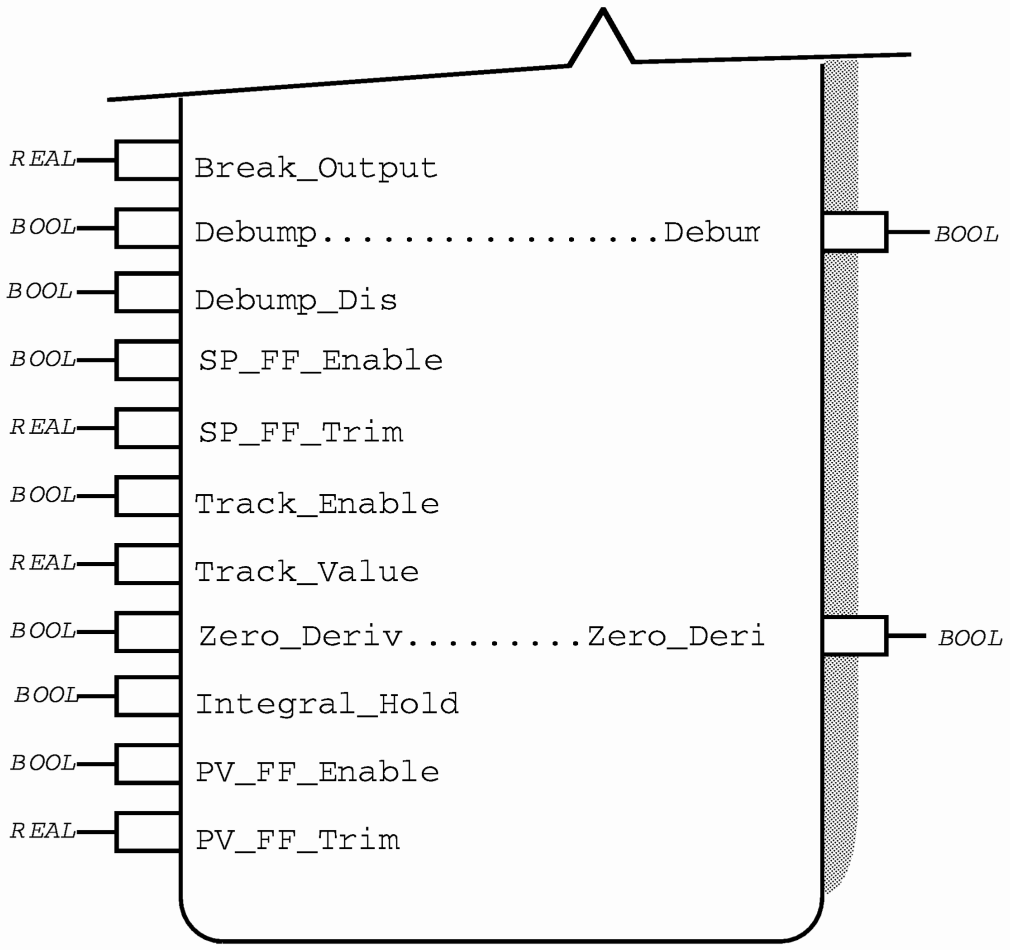


Figure 9-41 PID Function Block Diagram (continued)

Functional Description

The PID function block implements the proportional plus integral plus derivative control algorithm which is also used in Eurotherm's 900 series control instruments. The basic PID algorithm can be represented by the equation:

$$\text{Output} = \left(\frac{10000}{\text{Span} * \text{Prop\_Band}}\right)\left(E(t) + \frac{1}{\text{Integral}}\int E(t).dt + \text{Derivative}\,.\,\frac{d.E\,(t)}{dt}\right)$$

where E (t) is given by **Setpoint - Process_Val** and **Span** is given by S**pan_High - Span_Low.** In the PC3000, this basic algorithm is supported by additional functionality to improve the control performance and to enable the function block to be configured to control a wide range of systems. In the description of the function block's operation the parameters have been grouped together under the common functional classifications of Configuration, Dynamic Input, Control, Output Related and Diagnostic.

Configuration parameters are those which affect the structure of the controller and need only to be set during the initial design.

Function Block Attributes

Type:	20 38
Class:	CONTROL
Default Task:	Task_2
Short List:	Setpoint, Process_Val, Manual, Output
Memory Requirement:	268 Bytes
Execution Time:	1.17 ms for single output operation
	1.40 ms for dual output operation.

Dynamic Inputs are those which may typically change frequently during the run time of the function block. Control related parameters affect the turing of the loop. 'Output' Related parameters are those concerned directly with the Output stage of the block. Finally, Diagnostic parameters give information regarding various stages of calculation within the block.

Parameter Descriptions

Break_Output:	Configuration	Manual:	Control
Ch1	Output	Manual_Reset:	Control
Ch2	Output	Nerror:	Diagnostic
Control_Sig:	Diagnostic	Output:	Output
CS_Pre_Limit:	Diagnostic	Process_Val:	Dynamic Input
Cutback_High:	Control	Prop_Band:	Control
Cutback_Low:	Control	PV_FF_Enable	Dynamic Input
Debump:	Dynamic Input	PV_FF_Trim	Dynamic Input
Debump_Dis:	Dynamic Input	Sensor_Break:	Dynamic Input
Deriv_On_PV:	Configuration	Setpoint:	Dynamic Input
Deriv_Out:	Diagnostic	Span_High:	Configuration
Derivative:	Control	Span_Low:	Configuration
Direct:	Configuration	SP_FF_Enable:	Dynamic Input
Error:	Diagnostic	SP_FF_Trim:	Dynamic Input
Feed_Forward:	Dynamic Input	Status:	Diagnostic
Feedback:	Diagnostic	Zero_Deriv:	Control
Integral:	Control		
Integral_Hold:	Control		
Integral_Out	Diagnostic		

Table 9-7 Parameter Classification

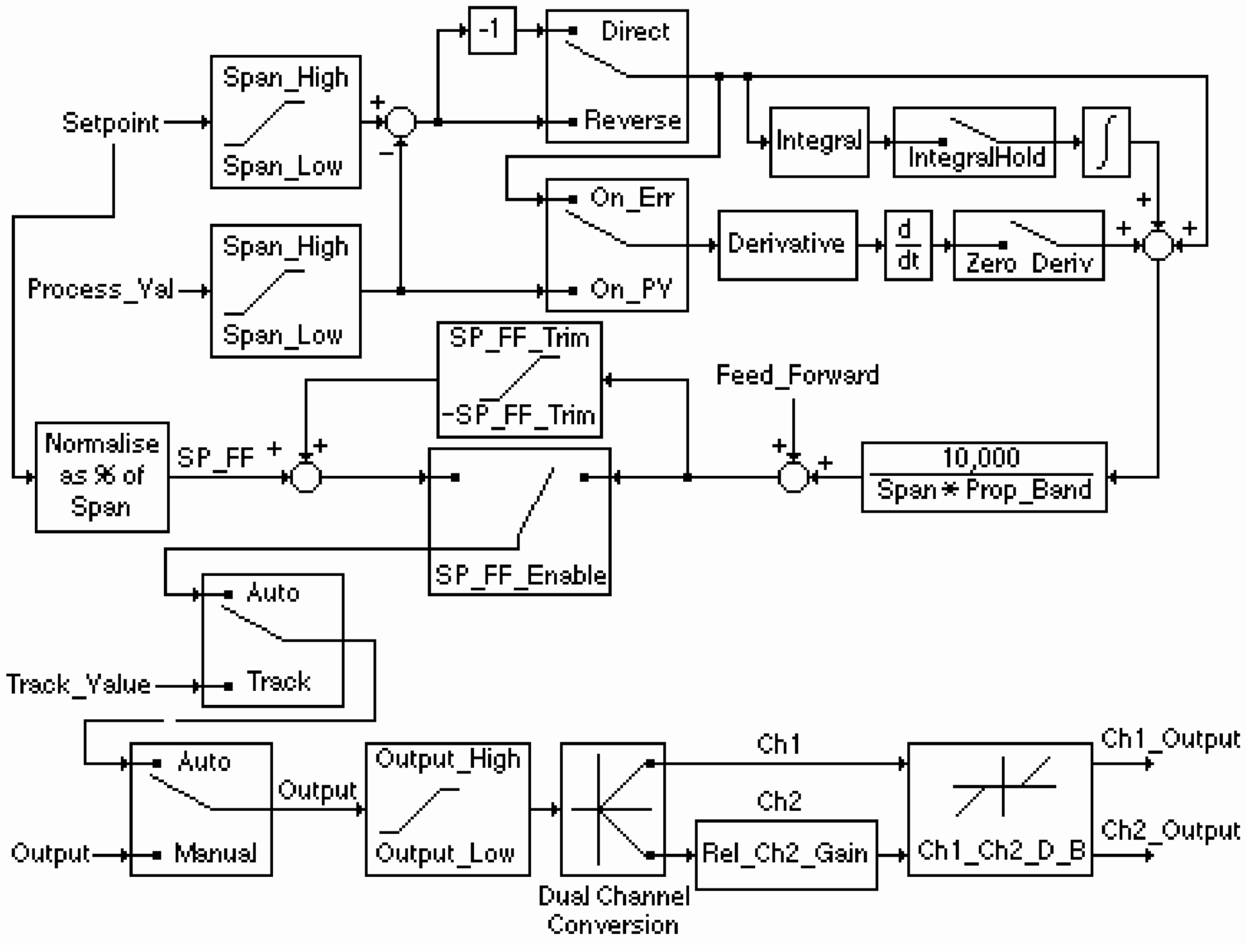


Figure 9-42 PID Functional Block Diagram

Configuration Parameters

For optimum operation, the PID function block must be configured to the type of control problem that it is to be applied to. This configuration is performed by appropriate setting of the input parameters.

Span_High and Span_Low.

Span_High and **Span_Low** define the maximum and minimum limits of the working range of the function block. Generally these are set to values which represent physical boundaries in the operation of the process, such as the calibrated range of a transducer, or the safe limits of a pressure vessel. The proportional band of the PID algorithm is defined as a percentage of the span of the function block, which is found by subtracting **Span_Low** from **Span_High.** PV subject to limits of **Span High** +10% and **Span Low** -10%. Setpoint subject to limits of **Span High** and **Span Low** -. If the **Process_Val or** setpoint move outside the span the function block will enter a sensor break condition. Output will take the value set by the **Break_Output** parameter.

Direct

Direct defines whether the function block is direct acting or reverse acting. If the function block is direct acting, the **Output** will tend to increase if the **Process_Val** is greater than the **Setpoint.** If the function block is reverse acting, the **Output** will tend to increase if the **Process_Val** is less than the **Setpoint.**

Deriv_On_PV

Deriv_On_PV defines whether the derivative action is responds to changes to **Process_Val** only (On_PV (1)) or to changes to the difference between the **Setpoint** and the **Process_Val** (On_Err (0)).

Break_Output

Break_Output defines the output level to which the function block will default when a sensor break condition has been detected. This is either triggered by Sensor_Break being set to Break (1), or by the Process_Val moving outside the span of the function block ± 10% or setpoint moving outside span

Dynamic Input Parameters

Dynamic inputs to the function block include those whose values can be expected to be continuously changing, such as the Process_Value and those for which a change in value can be expected to occur at any time by either the sequence program or changes in process conditions, such as a Sensor_Break.

Setpoint and Process_Val

The Process_Val is the controlled variable of the function block and the Setpoint is the target value against which the Process_Val is controlled. The PID algorithm acts to reduce the difference between the Setpoint and the Process_Val to zero.

Sensor_Break

The sensor break input provides a trigger which can be used to set the function block into a sensor break condition. When the function block is in sensor break, the Output will be set to Break_Output and the PID algorithm will be disabled. On leaving the sensor break condition, the algorithm will hold the output at Break_Output for sixteen samples, to ensure that the sensor has been properly re-acquired.

Debump and Debump_Dis

Debump functionality is employed by the function block to ensure that changes to operating conditions or control parameters do not result in sharp deviations of the Output signal. In the PC3000, debumping is automatically carried out whenever changes are made to Prop_Band, Derivative, Span_High, Span_Low, Ch1_Ch2_D_B, Rel_Ch2_Gain, or when the mode of the function block is switched between Manual and Auto. The parameter Debump can be used to trigger debumping for other changes, such as changes to the Setpoint. Debump_Dis can be used to disable the debump functionality, which will allow the Output to "kick" in response to changes in the parameters listed above.

SP_FF_Enable and SP_FF_Trim

SP_FF_Enable is used to enable Setpoint feedforward functionality, which is generally employed when the function block is being used as part of a cascade control application. SP_FF_Trim is used to limit the peak level of the setpoint feedforward trim term. The units of SP_FF_Trim are in percent of the span of the Output.

PV_FF_Enable and SP_FF_Trim

Are used when PV feedforward is required in cascade. The functionality is just as setpoint feedforward with process value replacing the setpoint.

Feed_Forward

This is added directly to the output of the PID algorithm, before the output limiting and dual output conversions are performed.

Control Parameters

The function block has several parameters which are not intended to be changed dynamically, but instead control the operation of the algorithm. These include the tuning parameters, such as Prop_Band, as well as booleans which are used to change the mode of operation.

Manual

Used to select whether the function block operates in Manual or Auto mode. In Manual mode, the PID algorithm is disabled and the value of Output is taken directly from its input. In Auto mode the full controller functionality is active.

PropUnits (Version 3.00 onwards)

PropUnits is used to define whether the **Prop_Band** is defined as a percentage of the controller span (PctSpan (0)) or in engineering units (EngUnts (1)).

Prop_Band

Prop_Band is the proportional band of the PID control algorithm.

Prior to Version 3.00, the proportional band could only be defined as a percentage of span. The engineering units mode was **not** supported.

If **PropUnits** is set to a percentage of span (PctSpan (0)), the proportional band is defined as the percent of the total span of the controller for which a control error will produce an output signal equivalent to the maximum output of the instrument. For example, if **Span_High** is 1000, **Span_Low** is 0, **Setpoint** is 500, **Process_Val is** 490 and **Prop_Band** is 1%, a proportional only reverse acting controller will produce an output of +100%, because the control error will be 10 units, which is one proportional band.

> **Note:-** When defined as a percentage of span, the the proportional gain is given by:

$$\text{Gain} = \frac{10{,}000}{(\text{Span\_High} - \text{Span\_Low}) * \text{Prop\_Band}}$$

If **PropUnits** is set to engineering units (EngUnts (1)), the proportional band is defined as the magnitude of the control error which will produce an output signal equivalent to the maximum output of the instrument. For example, if **Setpoint** is 500°C, **Process_Val** is 495°C and **Prop_Band** is 5°C, a proportional only reverse acting controller will produce an output of +100%, because the control error is 5°C which is equal to one proportional band.

When defined in Engineering units, the proportional gain is given by:

$$\text{Gain} = \frac{100}{\text{Prop\_Band}}$$

Integral

Integral is the integral time constant of the PID control algorithm. Integral time is defined as the time period in which the part of the output signal due to integral action increases by an amount equal to the part of the output signal due to proportional action, for a constant error state.

Derivative

Derivative is the derivative time constant of the PID control algorithm. **Derivative** time is defined as the time interval in which the part of the output signal due to derivative action increases by an amount equal to the part of the output signal due to proportional action, when the control error is changing at a constant rate.

Manual_Reset

Manual_Reset is only active when **Integral** is set to zero. It provides an offset to the **Output,** which can be used to reduce the control error to zero when integral action is not employed.

Cutback_High and Cutback_Low

Cutback can be employed to reduce the amount of time it takes the **Process_Val** to respond to large changes in **Setpoint** and to limit the overshoot that can occur during the transient period. **Cutback_High** operates when the **Process_Val** is initially greater than the target **Setpoint** and **Cutback_Low** operates when the

Process_Val is initially less than the target **Setpoint.** The units of **Cutback_High** and **Cutback_Low** are engineering units. Cutback operates by forcing the **Output** to its appropriate maximum or minimum limit in response to a **Setpoint** change which is greater than the cutback band. As the **Process_Val** approaches the **Setpoint**, the control error (Setpoint - Process_Val) reduces to less than the cutback value and normal control is resumed.

Track_Enable and Track_Value

Track_Enable allows the PID algorithm to be disabled and the **Output** signal to be read directly from **Track_Value.** These parameters are generally used when the function block forms part of a cascade control application.

Zero_Deriv

Zero_Deriv can be used to force the derivative output to zero. It is automatically cleared by the function block after one sample.

Integral_Hold

Integral_Hold enables the integral output to be frozen at its current value. It will remain constant throughout the period that **Integral_Hold** is enabled.

Output Parameters

Output, Ch1_Output and Ch2_Output

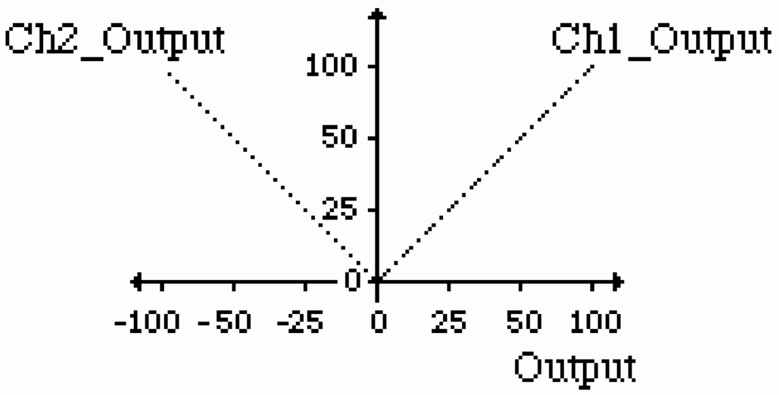


Figure 9-43 Dual Output Relationship, with Unity Rel_Ch2_Gain and zero Ch1_Ch2_D_B

The PID function block channel can be configured for either single or dual channel operation. In single channel operation, the control is on-Output, which is limited within the range + 100 % to 0 % by the parameters Output_High and Output_Low. Dual channel operation is designed for systems such as heat-cool applications, in which negative values of Output must be output to a refrigeration unit as absolute values to increase the refrigeration rate. For dual channel operation Output_Low is set to - 100%.

Output_High and Output_Low

Output_High and Output_Low define the upper and lower limits of Output They must both be set within the range -100% to +100%, with Output_High being greater than or equal to Output_Low. When the function block is being used for dual output operation, Ch1_Output and Ch2_Output are related to Output as shown in Figure 9.2, with Output being bounded by Output_High and Output_Low.

Rel_Ch2_Gain and Ch1_Ch2_D_B

When the function block is being used for dual channel control, the full relationship between Ch2_Output and Output is given by:

Ch2_Output = (Output + Ch1_Ch2_D_B) * Rel_Ch2_Gain

Rel_Ch2_Gain is intended for use in non linear dual output control situations, such as heat / cool systems, to compensate for the differing gains of the equipment being driven by the two output channels. Ch1_Ch2_D_B introduces a deadband between the two output channels, which can either be set to a positive value to provide a region in which neither channel is active, or to a negative value to provide a region of overlap in which both outputs are active.

Output_Rate and Out_Rate_En

Output_Rate can be used to limit the maximum rate of change of Output per second. It is enabled by setting Out_Rate_En to Yes (1). Note that Output_Rate affects the rate of change of Output directly, so that the rate of change of Ch2_Output will be modified by the Rel_Ch2_Gain.

Diagnostic Parameters

{1} CS_Pre_Limit

This is the output of the PID controller after the Feed_Forward has been added, but before Output limiting has been performed.

Control_Sig

Control_Sig is the sum of the proportional, integral, derivative and feed forward components after being high, low and rate of change limited, but before the dual output relative gain and deadband have been added.

Integral_Out and Deriv_Out

Integral_Out and Deriv_Out are the outputs of the integral and derivative components respectively.

Feedback

Feedback is the value of the signal which is fed back into the PID to indicate the actual signal that is output by the controller.

Error

Error is the difference between the Process_Val and the Setpoint,

(Process_Val - Setpoint).

Nerror

Nerror is the output of the proportional component of the function block.

Status

Status provides an indication of the function of the PID function block. it can have eight possible states:

Ok (0):	The function block is operating normally
SnsrBrk (1):	An external sensor break has been detected
PV_High (2):	The Process_Val is greater than Span_High +10%
PV_Low (3):	The Process_Val is less than Span_Low -10%
SP_High (4):	The Setpoint is greater than Span_High

SP_Low (5):	The Setpoint is less than Span_Low
GainNeg (6):	The Prop_Band has been set with a negative value, or Span_High has been set less than Span_Low
GainHi (7):	The Prop_Band has been set with too small a value or the span of the function block has been set too small, resulting in a very large gain.

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Break_Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Ch1_Ch2_D_B	**REAL**	0	Oper	Oper	High Limit Low Limit	10 -10
Ch1_Output	**REAL**	0	Oper		High Limit Low Limit	100 The higher of 0 or Output_Low
Ch2_Output	**REAL**	0	Oper		High Limit Low Limit	The lower of 0 or Output_High -100
Control_Sig	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
CS_Pre_Limit	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Cutback_High	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Cutback_Low	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Debump	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Debump_Dis	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Deriv_On_PV	**BOOL**	On_Err (0)	Config	Config	Senses	On_Err (0) On_PV (1)
Deriv_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000

Table 9-8 PID Parameter Attributes

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
Derivative	**TIME**	50s	Oper	Oper	High Limit Low Limit	01d_03h 0
Direct	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Error	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Feedback	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Integral	**TIME**	5m	Oper	Oper	High Limit Low Limit	01d_03h 0
Integral_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Manual	**BOOL**	Manual (1)	Oper	Oper	Senses	Auto (0) Manual (1)
Manual_Reset	**REAL**	0	Oper	Oper	High Limit Low Limit	100 -100
Nerror	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Out_Rate_En	**BOOL**	No (0)	Oper	Config	Senses	No (0) Yes (1)
Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Output_High	**REAL**	100	Oper	Oper	High Limit Low Limit	100 Output_Low
Output_Low	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High -100
Output_Rate	**REAL**	10	Oper	Oper	High Limit Low Limit	10,000 0.1

Table 9-8 PID Parameter Attributes (continued)

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
Process_Val	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Prop_Band	**REAL**	5 %	Oper	Oper	High Limit Low Limit	10,000 0.1
PropUnits	**BOOL**	PctSpan (0)	Oper	Oper	Senses	PctSpan (0) EngUnts (1)
Rel_Ch2_Gain	**REAL**	1	Oper	Oper	High Limit Low Limit	10 0.1
Sensor_Break	**BOOL**	Ok (0)	Oper	Super	Senses	Ok (0) Break (1)
Setpoint	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
SP_FF_Enable	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
SP_FF_Trim	**REAL**	0	Config	Config	High Limit Low Limit	100,000 -100,000
PV_FF_Enable	**BOOL**	No(0)	Super	Config	Sences	No (0) Yes (1)
PV_FF_Trim	**REAL**	0	Config	Config	High Limit Low Limit	100,000 -100,000
Span_High	**REAL**	100	Config	Config	High Limit Low Limit	100,000 Span_Low
Span_Low	**REAL**	0	Config	Config	High Limit Low Limit	Span_High -100,000
Status	**ENUM**	Ok (0)	Oper		Senses	Ok (0) SnsrBrk (1) PV_High (2) PV_Low (3) SP_High (4) SP_Low (5) GainNeg (6) GainHi (7)

Table 9-8 PID Parameter Attributes (continued)

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
Track_Enable	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Track_Value	**REAL**	0	Super	Config	High Limit Low Limit	Output_High Output_Low
Zero_Deriv	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)

Table 9-8 PID Parameter Attributes (continued)

VP FUNCTION BLOCK

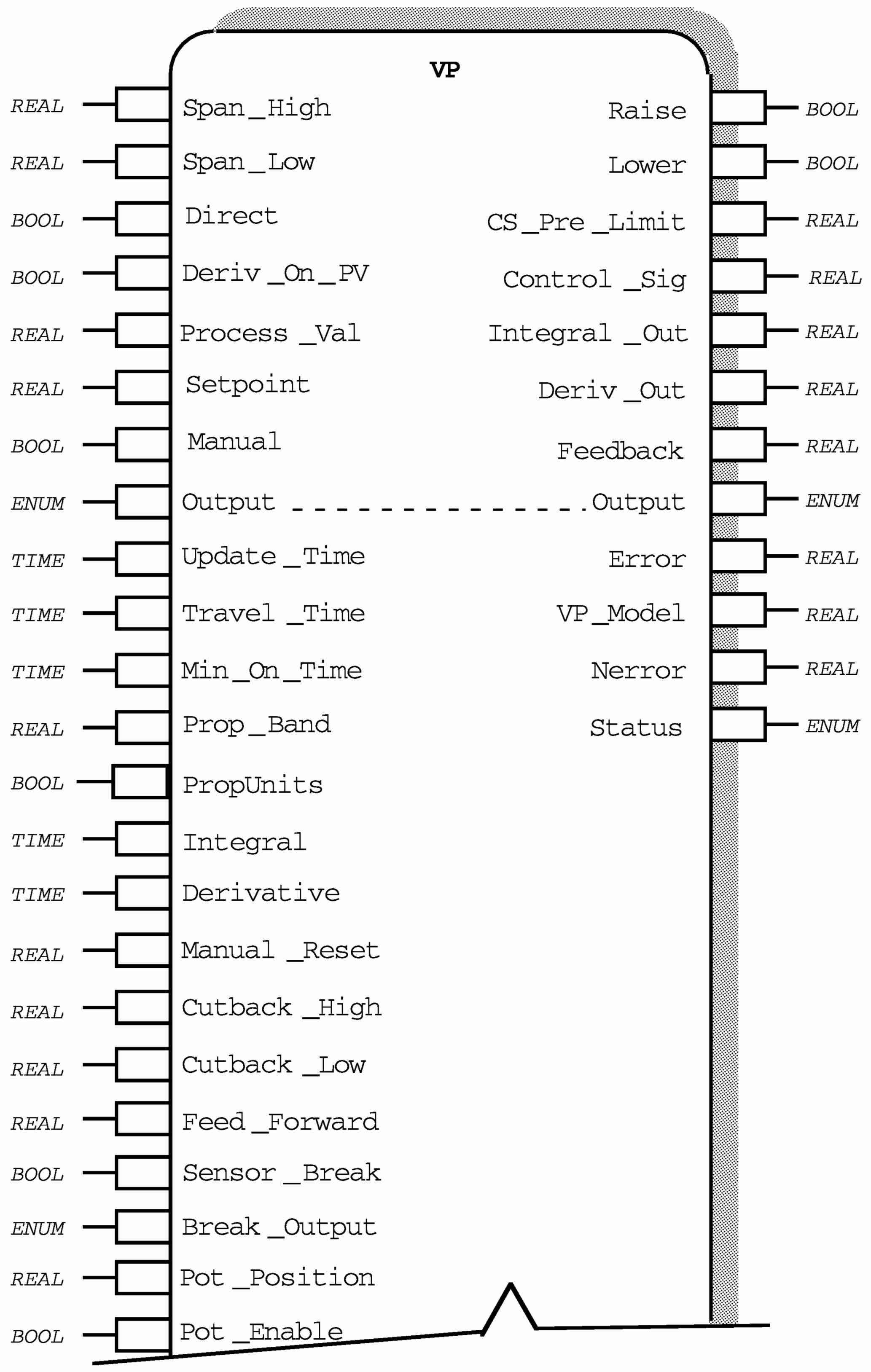


Figure 9-44 VP Function Block Diagram

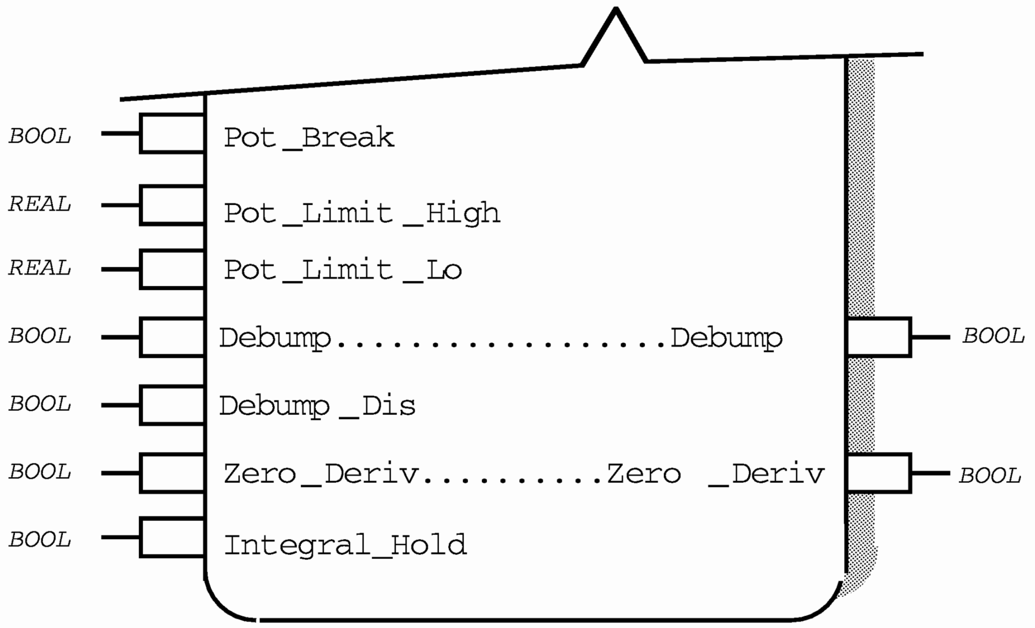


Figure 9-44 VP Function Block Diagram (continued)

Functional Description

The VP function block is an enhanced version of the PID function block, which implements the proportional plus integral plus derivative control algorithm which is also used in Eurotherm's 900 series control instruments, along with additional output stages to enable it to control loops having motorised valves as actuators. The basic PID algorithm can be represented by the equation:

$$\text{Output} = \left(\frac{10000}{\text{Span} * \text{Prop\_Band}}\right)\left(E(t) + \frac{1}{\text{Integral}}\int E(t).dt + \text{Derivative} \cdot \frac{d.E\ (t)}{dt}\right)$$

where E (t) is given by (Setpoint - Process_Val) and Span is given by (Span_High - Span_Low). In the PC3000, this basic algorithm is supported by additional functionality to improve the control performance and to enable the function block to be configured to control a wide range of systems.. In the description of the function block's operation and use given below, the parameters have been grouped together under the common functional classifications of Configuration, Dynamic Input, Control, Output Related and Diagnostic. Configuration parameters are those which affect the structure of the controller and need only to be set during the initial design.

Function Block Attributes

Type:	20 40
Class:	CONTROL
Default Task:	Task_2
Short List:	Setpoint, Process_Val, Manual, Output
Memory Requirements:	264 Bytes
Execution Time:	1.9 ms

Dynamic Inputs are those which may typically change frequently during the run time of the function block. Control related parameters affect the turing of the loop. 'Output' Related parameters are those concerned directly with the Output stage of the block. Finally, Diagnostic parameters give information regarding various stages of calculation within the block.

Valve Positioner Output Stage Description

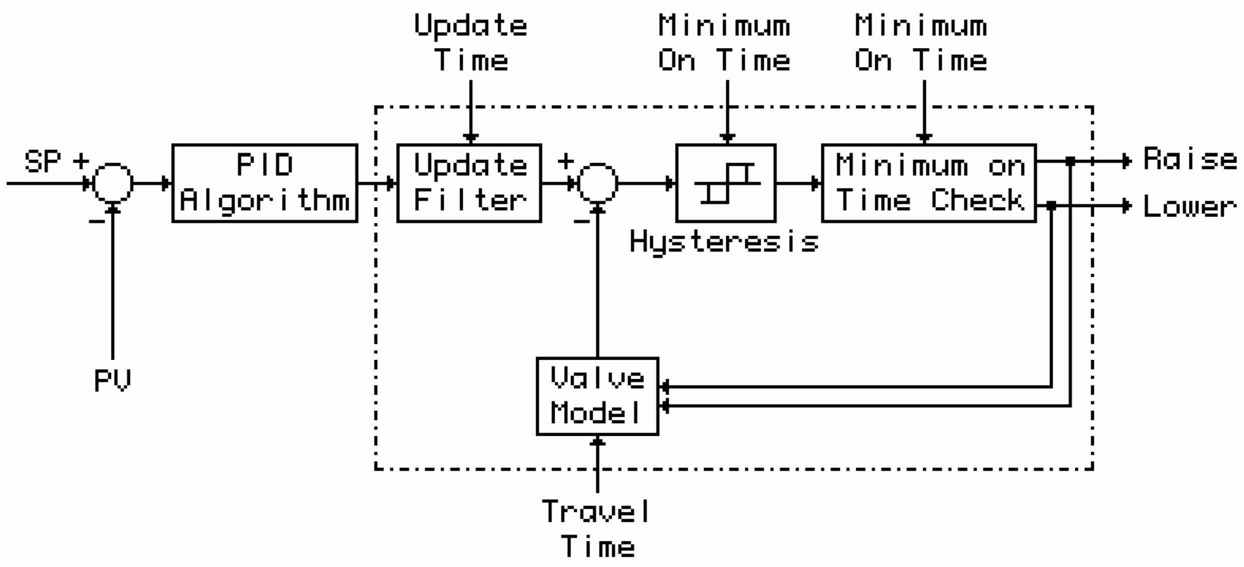


Figure 9-45 Valve Positioner Block Diagram.

The function block's valve positioner stages use a boundless algorithm, which has been incorporated because it does not require position feedback from the valve, since this can often be unreliable. A model of the valve is included for valve position output control. Potentiometer position feedback can be used in conjunction with the valve model. In this mode of operation, the valve model position is used for control. The actual position is used for limitting purposes only. This enables the controller to continue operating if the feedback potentiometer should fail.

Parameter Descriptions

Break_Output:	Configuration	Min_On_Time	Configuration
Control_Sig:	Diagnostic	Nerror:	Diagnostic
CS_Pre_Limit:	Diagnostic	Output:	Output
Cutback_High:	Control	Pot_Break:	Pot Feedback
Cutback_Low:	Control	Pot_Enable:	Pot Feedback
Debump:	Dynamic Input	Pot_Limit_Hi:	Pot Feedback
Debump_Dis:	Dynamic Input	Pot_Limit_Lo:	Pot Feedback
Deriv_On_PV:	Configuration	Pot_Position:	Pot Feedback
Deriv_Out:	Diagnostic	Process_Val:	Dynamic Input
Derivative:	Control	Prop_Band:	Control
Direct:	Configuration	Raise:	Output
Error:	Diagnostic	Sensor_Break:	Dynamic Input
Feed_Forward:	Dynamic Input	Setpoint:	Dynamic Input
Feedback:	Diagnostic	Span_High:	Configuration
Integral:	Control	Span_Low:	Configuration
Integral_Hold:	Control	Status:	Diagnostic
Integral_Out	Diagnostic	Travel_Time:	Configuration
Lower:	Output	Update_Time:	Configuration
Manual:	Control	VP_Model	Diagnostic
Manual_Reset:	Control	Zero_Deriv:	Control

Table 9-9 Parameter Classification

Function Block Configuration Parameters

For optimum operation, the VP function block must be configured to the type of control problem that it is to be applied to. This configuration is performed by appropriate setting of the input parameters. The use of these configuration parameters is described below.

Span_High and Span_Low.

Span_High and Span_Low define the maximum and minimum limits of the working range of the function block. Generally these are set to values which represent physical boundaries in the operation of the process, such as the calibrated range of a transducer, or the safe limits of a pressure vessel. The proportional band of the PID algorithm is defined as a percentage of the span of the function block, which is found by subtracting Span_Low from Span_High. If the Process_Val moves outside the span the function block will enter a sensor break condition.

Direct

Direct defines whether the function block is direct acting or reverse acting. If the function block is direct acting, the Output will tend to increase if the Process_Val is greater than the Setpoint. If the function block is reverse acting, the Output will tend to increase if the Process_Val is less than the Setpoint.

Deriv_On_PV

Deriv_On_PV defines whether the derivative action is responds to changes to Process_Val only (On_PV (1)) or to changes to the difference between the Setpoint and the Process_Val (On_Err (0)).

Update_Time

Update_Time defines the sample time of the valve positioner output stages. Generally, this should be set to one tenth of the Travel_Time. Increasing the Update_Time will reduce the amount of activity of the valve, but can also reduce the accuracy of the control. Similarly, reducing the Update_Time can improve the control performance, but will also increase the amount of activity of the valve.

Travel_Time

Travel_Time is the amount of time it takes the valve to travel from the lower operating position to the upper operating position.

Min_On_Time

Min_On_Time defines the minimum time that the valve must remain in a state of opening, closing or remaining stationary.

Break_Output

Break_Output defines the output state to which the function block will default when a sensor break condition has been detected. This is either triggered by Sensor_Break being set to Break (1), or by the Process_Val or setpoint moving outside the span of the function block.

Potentiometer Feedback Parameters

Pot_Position

Pot_Position is the input to which the valve position feedback potentiometer is connected.

Pot_Enable

When Pot_Enable is set to Yes (1), the potentiometer feedback algorithm is activated. Setting Pot_Enable to No (0) causes the function block to operate without potentiometer feedback.

Pot_Break

Pot_Break provides an external trigger to indicate that the potentiometer feedback has become disconnected, or is faulty. Setting Pot_Break to Yes (1) indicates a fault condition has occured. Setting Pot_Break to No (0) indicates the sensor is functioning normally.

Pot_Limit_Hi

Pot_Limit_Hi defines the upper operating position of the valve.

Pot_Limit_Lo

Pot_Limit_Lo defines the lower operating position of the valve.

Function Block Dynamic Input Parameters

Setpoint and Process_Val

The Process_Val is the controlled variable of the function block and the Setpoint is the target value against which the Process_Val is controlled. The PID algorithm acts to reduce the difference between the Setpoint and the Process_Val to zero.

Sensor_Break

The sensor break input provides a trigger which can be used to set the function block into a sensor break condition. When the function block is in sensor break, the Output will be set to Break_Output and the PID algorithm will be disabled. On leaving the sensor break condition, the algorithm will hold the output at Break_Output for sixteen samples, to ensure that the sensor has been properly re-acquired.

Debump and Debump_Dis

Debump functionality is employed by the function block to ensure that changes to operating conditions or control parameters do not result in sharp deviations of the Output signal. In the PC3000, debumping is automatically carried out whenever changes are made to Prop_Band, Derivative, Span_High, Span_Low, Ch1_Ch2_D_B, Rel_Ch2_Gain, or when the mode of the function block is switched between Manual and Auto. The parameter Debump can be used to trigger debumping for other changes, such as changes to the Setpoint. Debump_Dis can be used to disable the debump functionality, which will allow the Output to "kick" in response to changes in the parameters listed above.

SP_FF_Enable and SP_FF_Trim

SP_FF_Enable is used to enable Setpoint feedforward functionality, which is generally employed when the function block is being used as part of a cascade control application. SP_FF_Trim is used to limit the peak level of the setpoint feedforward trim term. The units of SP_FF_Trim are in percent of the span of the Output signal of the PID port.

Function Block Control Parameters

The function block has several parameter which are not intended to be changed dynamically, but instead control the operation of the algorithm. These include the tuning parameters, such as Prop_Band, as well as booleans which are used to change the mode of operation. These parameters are described below.

Manual

Manual is used to select whether the function block operates in Manual or Auto mode. In Manual mode, the PID algorithm is disabled and the value of Output is taken directly from its input. In Auto mode the full controller functionality is active.

PropUnits (Version 3.00 onwards)

PropUnits is to define whether the Prop_Band is defined as a percentage of the controller span (PctSpan (0)) or in engineering units (EngUnts (1)).

Prop_Band

Prop_Band is the proportional band of the PID control algorithm.

Prior to Version 3.00, the proportional band could only be defined as a percentage of span. The engineering units mode was not supported.

If PropUnits is set to a percentage of span (PctSpan (0)), the proportional band is defined as the percent of the total span of the controller for which a control error will produce an output signal equivalent to the maximum output of the instrument. For example, if Span_High is 1000, Span_Low is 0, Setpoint is 500, Process_Val is 490 and Prop_Band is 1%, a proportional only reverse acting controller will produce an output of +100%, because the control error will be 10 units, which is one proportional band.

Note:- When defined as a percentage of span, the the proportional gain is given by:

$$\text{Gain} = \frac{10{,}000}{(\text{Span\_High} - \text{Span\_Low})\ {*}\text{Prop\_Band}}$$

If PropUnits is set to engineering units (EngUnts (1)), the proportional band is defined as the magnitude of the control error which will produce an output signal equivalent to the maximum output of the instrument. For example, if Setpoint is 500°C, Process_Val is 495°C and Prop_Band is 5°C, a proportional only reverse acting controller will produce an output of +100%, because the control error is 5°C which is equal to one proportional band.

When defined in Engineering units, the proportional gain is given by:

$$\text{Gain} = \frac{100}{\text{Prop\_Band}}$$

Integral

Integral is the integral time constant of the PID control algorithm. Integral time is defined as the time period in which the part of the output signal due to integral action increases by an amount equal to the part of the output signal due to proportional action, for a constant error state.

Derivative

Derivative is the derivative time constant of the PID control algorithm. Derivative time is defined as the time interval in which the part of the output signal due to derivative action increases by an amount equal to the part of the output signal due to proportional action, when the control error is changing at a constant rate.

Feed_Forward

Feed_Forward is added directly to the output of the PID algorithm, before the output limiting and dual output conversions are performed.

Manual_Reset

Manual_Reset is only active when Integral is set to zero. It provides an offset to the Output, which can be used to reduce the control error to zero when integral action is not employed.

Cutback_High and Cutback_Low

Cutback can be employed to reduce the amount of time it takes the Process_Val to respond to large changes in Setpoint and to limit the overshoot that can occur during the transient period. Cutback_High operates when the Process_Val is initially greater than the target Setpoint and Cutback_Low operates when the Process_Val is initially less than the target Setpoint. The units of Cutback_High and Cutback_Low are engineering units. Cutback operates by forcing the Output to its appropriate maximum or minimum limit in response to a Setpoint change which is greater than the cutback band. As the Process_Val approaches the Setpoint, the control error (Setpoint - Process_Val) reduces to less than the cutback value and normal control is resumed.

Zero_Deriv

Zero_Deriv can be used to force the derivative output to zero. It is automatically cleared by the function block after one sample.

Integral_Hold

Integral_Hold enables the integral output to be frozen at its current value. It will remain constant throughout the period that Integral_Hold is enabled.

Function Block Output Parameters

Output, Raise and Lower

When the function block is operating in Manual mode, Output serves as an input which can be used to directly control the Raise and Lower outputs to the valve.

Function Block Diagnostic Parameters

CS_Pre_Limit

This is the output of the PID controller after the Feed_Forward has been added, but before Output limiting has been performed.

Control_Sig

Control_Sig is the sum of the proportional, integral, derivative and feed forward components after being high, low and rate of change limited, but before the dual output relative gain and deadband have been added.

Integral_Out and Deriv_Out, Nerror

Nerror, Integral_Out and Deriv_Out are the outputs of the proportional integral and derivative components respectively.

Feedback

Feedback is the value of the signal which is fed back into the PID to indicate the actual signal that is output by the controller.

Error

Error is the difference between the Process_Val and the Setpoint.

VP_Model

VP_Model is the output of the internal model of the valve.

Status

Status provides an indication of the function of the function block. it can have eight possible states:

Ok (0):	The function block is operating normally
SnsrBrk (1):	An external sensor break has been detected
PV_High (2):	The Process_Val is greater than Span_High +10%
PV_Low (3):	The Process_Val is less than Span_Low -10%
SP_High (4):	The Setpoint is greater than Span_High
SP_Low (5):	The Setpoint is less than Span_Low
GainNeg (6):	The Prop_Band has been set with a negative value, or Span_High has been set less than Span_Low
GainHi (7):	The Prop_Band has been set with too small a value or the span of the function block has been set too small, resulting in a very large gain.

Parameter Attributes

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
Break_Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Control_Sig	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
CS_Pre_Limit	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Cutback_High	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Cutback_Low	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Debump	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Debump_Dis	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Deriv_On_PV	**BOOL**	On_Err (0)	Config	Config	Senses	On_Err (0) On_PV (1)
Deriv_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Derivative	**TIME**	50s	Oper	Oper	High Limit Low Limit	01d_03h 0
Direct	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Error	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Feed_Forward	**REAL**	0	Super	Super	High Limit Low Limit	100 -100
Feedback	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000

Table 9-10 VP Parameter Attributes

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
Integral	**TIME**	5m	Oper	Oper	High Limit Low Limit	01d_03h 0
Integral_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
IntegralHold	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Lower	**BOOL**	Off (0)	Oper		Senses	Off (0) Lower (1)
Manual	**BOOL**	Manual (1)	Oper	Oper	Senses	Auto (0) Manual (1)
Manual_Reset	**REAL**	0	Oper	Oper	High Limit Low Limit	100 -100
Min_On_Time	**TIME**	100ms	Oper	Oper	High Limit Low Limit	5s 100ms
Nerror	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Pot_Break	**BOOL**	No (0)	Oper	Super	Senses	No (0) Yes (1)
Pot_Enable	**BOOL**	No (0)	Oper	Super	Senses	No (0) Yes (1)
Pot_Limit_Hi	**REAL**	100	Oper	Super	High Limit Low Limit	100 0
Pot_Limit_Lo	**REAL**	0	Oper	Super	High Limit Low Limit	100 0
Pot_Position	**REAL**	0	Oper	Super	High Limit Low Limit	100 0
Process_Val	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low

Table 9-10 VP Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Prop_Band	**REAL**	5 %	Oper	Oper	High Limit Low Limit	10,000 0.1
PropUnits	**BOOL**	PctSpan (0)	Oper	Oper	Senses	PctSpan (0) EngUnts (1)
Raise	**BOOL**	Off (0)	Oper		Senses	Off (0) Raise (1)
Sensor_Break	**BOOL**	Ok (0)	Oper	Super	Senses	Ok (0) Break (1)
Setpoint	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Span_High	**REAL**	100	Config	Config	High Limit Low Limit	100,000 Span_Low
Span_Low	**REAL**	0	Config	Config	High Limit Low Limit	Span_High -100,000
Status	**ENUM**	Ok (0)	Oper		Senses	Ok (0) SnsrBrk (1) PV_High (2) PV_Low (3) SP_High (4) SP_Low (5) GainNeg (6) GainHi (7)
Travel_Time	**TIME**	20s	Oper	Oper	High Limit Low Limit	16m_40s 5s
Update_Time	**TIME**	1s	Oper	Oper	High Limit Low Limit	20s 100ms
VP_Model	**REAL**	50	Config		High Limit Low Limit	100 0
Zero_Deriv	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)

Table 9-10 VP Parameter Attributes (continued)

PID_AUTO FUNCTION BLOCK

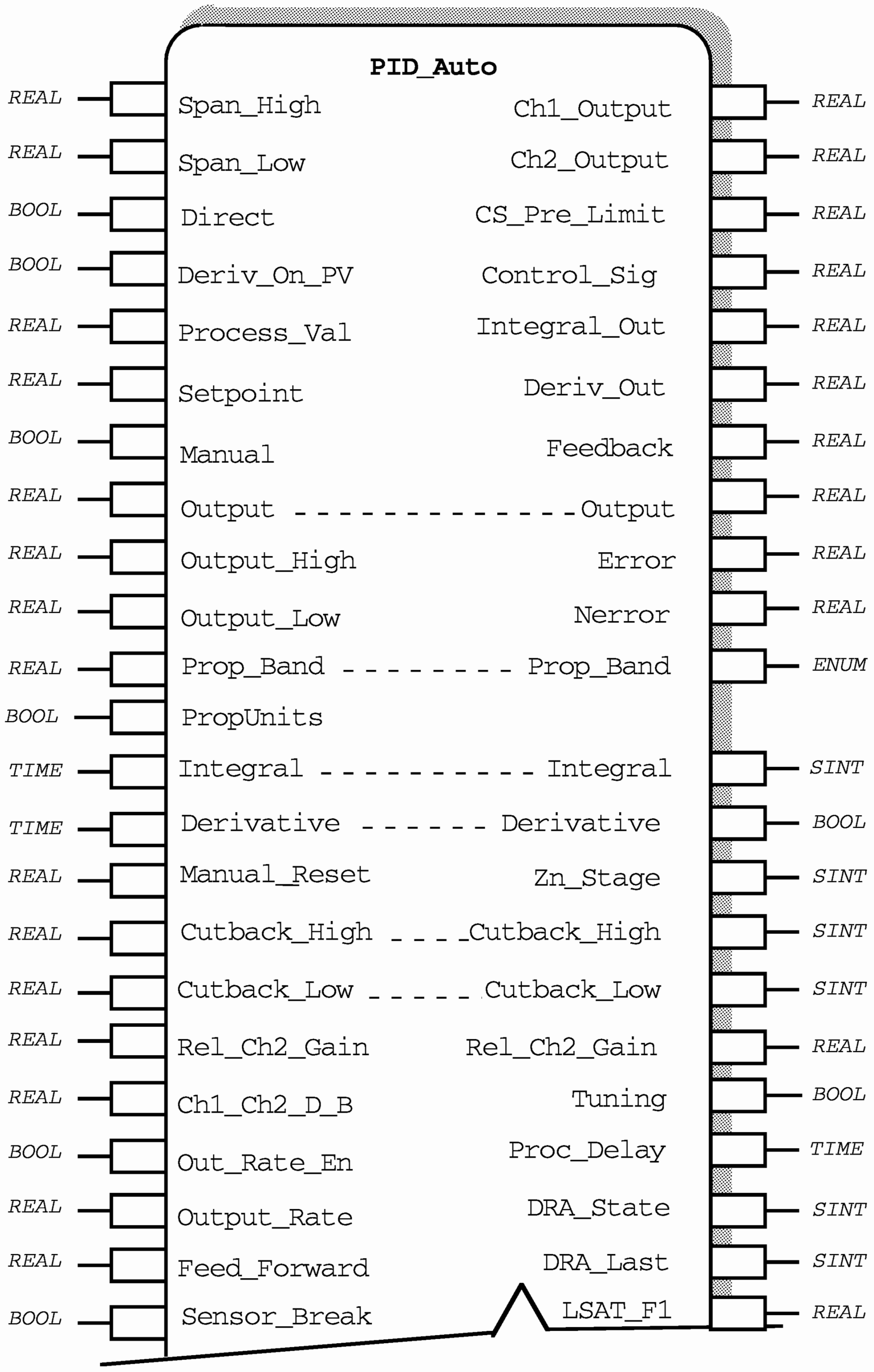


Figure 9-46 PID_Auto Function Block Diagram

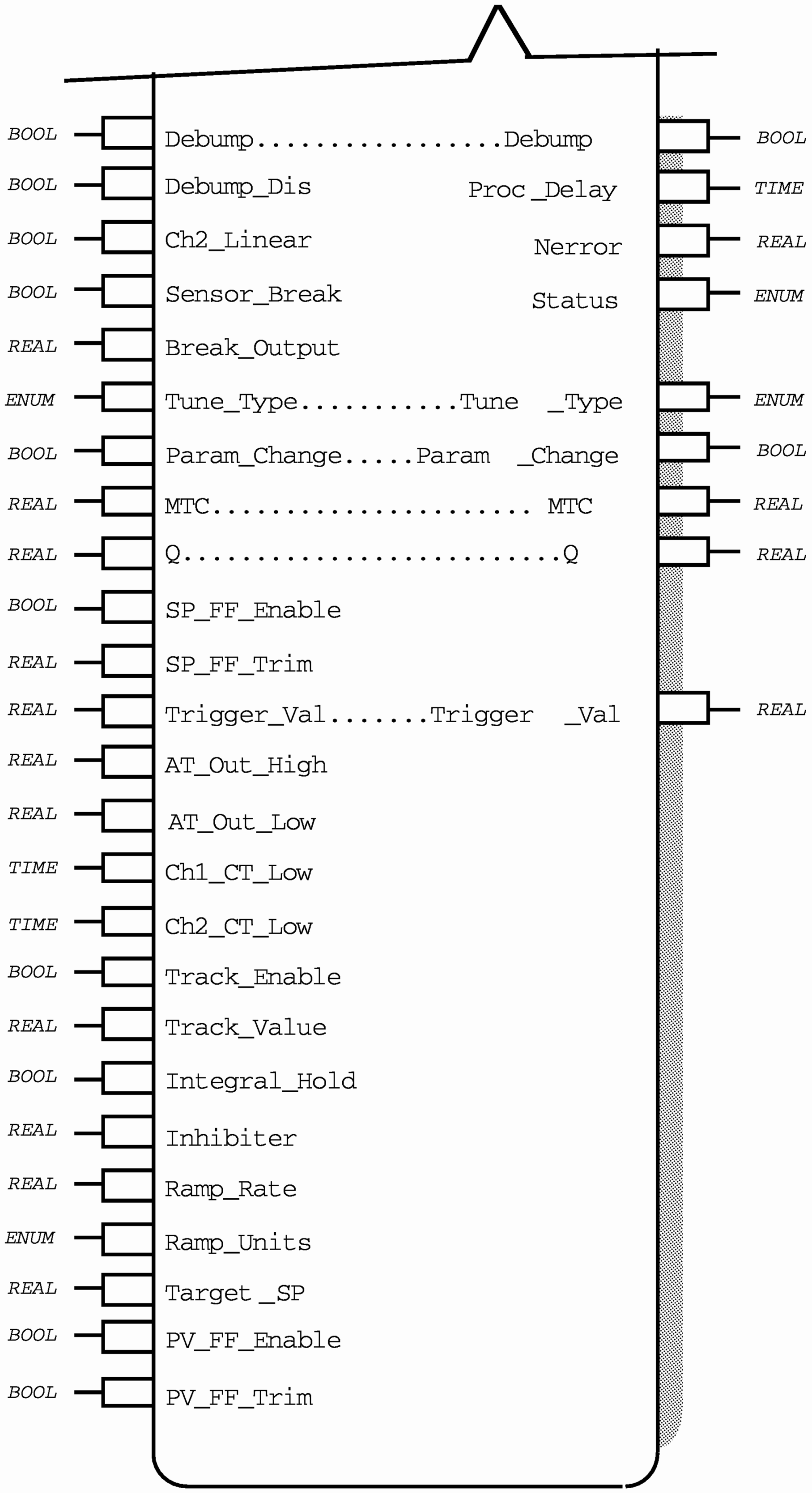

BOOL Debump.................Debump BOOL
BOOL Debump_Dis Proc_Delay TIME
BOOL Ch2_Linear Nerror REAL
BOOL Sensor_Break Status ENUM
REAL Break_Output
ENUM Tune_Type..........Tune _Type ENUM
BOOL Param_Change.....Param _Change BOOL
REAL MTC..................... MTC REAL
REAL Q........................Q REAL
BOOL SP_FF_Enable
REAL SP_FF_Trim
REAL Trigger_Val.......Trigger _Val REAL
REAL AT_Out_High
REAL AT_Out_Low
TIME Ch1_CT_Low
TIME Ch2_CT_Low
BOOL Track_Enable
REAL Track_Value
BOOL Integral_Hold
REAL Inhibiter
REAL Ramp_Rate
ENUM Ramp_Units
REAL Target _SP
BOOL PV_FF_Enable
BOOL PV_FF_Trim

Figure 9-46 PID_Auto Function Block Diagram (continued)

Functional Description

The PID_Auto function block implements the proportional plus integral plus derivative control algorithm and the Auto and Adaptive tuning algorithms that are also used in the Eurotherm 900 series control instruments. It is a complex function block which includes three tuning algorithms which can be used to set the parameters of the PID controller, plus an overshoot inhibition algorithm for use when the function block is following ramping Setpoint profiles.

The control algorithm used in PID_Auto is identical to that employed in the PID function block. The basic PID algorithm can be represented by the equation:

$$\text{Output} = \left(\frac{10000}{\text{Span} * \text{Prop\_Band}}\right)\left(E(t) + \frac{1}{\text{Integral}}\int E(t).dt + \text{Derivative} . \frac{d.E\ (t)}{dt}\right)$$

where E (t) is given by (Setpoint - Process_Val) and Span is given by (Span_High - Span_Low). In the PC3000, this basic algorithm is supported by additional functionality to improve the control performance and to enable the function block to be configured to control a wide range of systems.

Achieving the optimum performance from the block requires that the controller and tuners are correctly configured and activated. A description of the separate elements of the function block is given below.

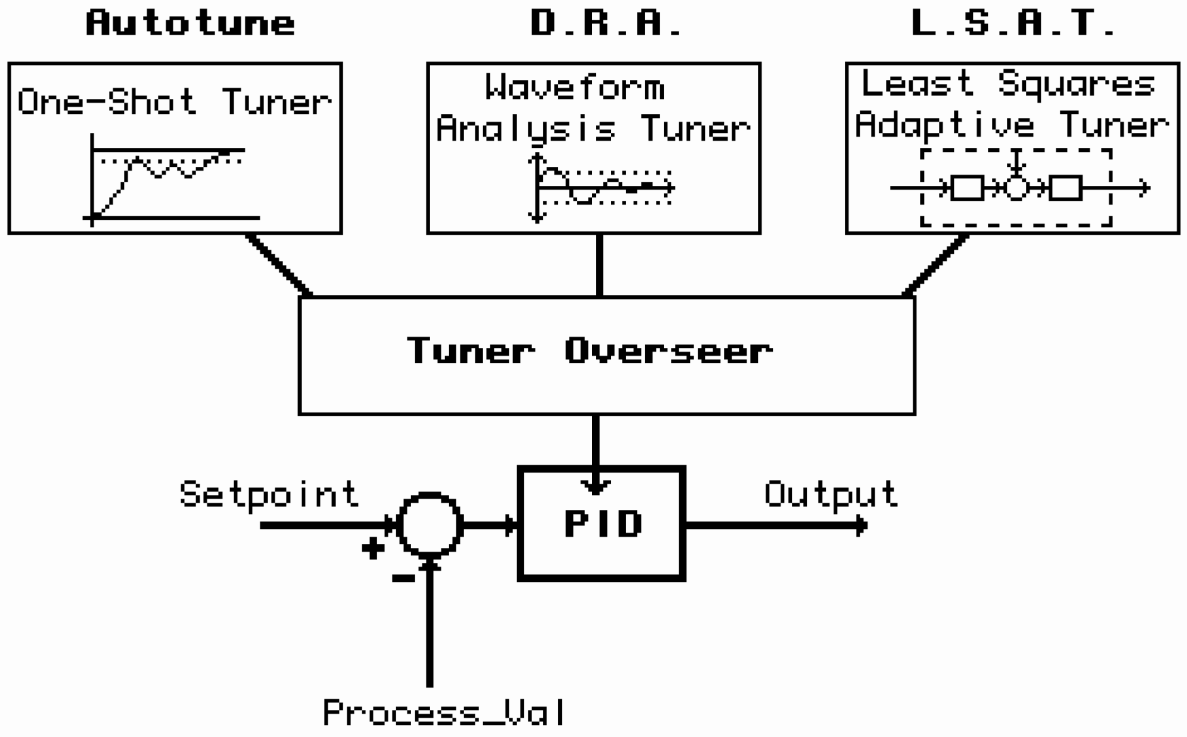


Figure 9-47 Block Diagram of the Principal PID_Auto Functionality.

Function Block Attributes

Type:	20 50
Class:	CONTROL
Default Task:	Task_2
Short List:	Setpoint, Process_Val, Manual, Output
Memory Requirements:	2030 bytes
Execution Time:	1.63ms when not tuning
	3.61ms maximum when tuning

Parameter Descriptions

AT_Out_High:	Tuning	Manual_Reset:	Control
AT_Out_Low:	Tuning	MTC:	Tuning
Break_Output:	Configuration	Nerror:	Diagnostic
Ch1_Ch2_D_B:	Output	Out_Rate_En:	Output
Ch1_CT_Low:	Tuning	Output:	Output
Ch1_Cycle_T:	Tuning	Output_High:	Output
Ch1_Output:	Output	Output_Low:	Output
Ch2_CT_Low:	Tuning	Output_Rate:	Output
Ch2_Cycle_T:	Tuning	Param_Change:	Tuning
Ch2_Linear:	Tuning	Process_Val:	Dynamic Input
Ch2_Output:	Output	Prop_Band	Control
Control_Sig:	Diagnostic	PV_FF_Enable	Dynamic Input
CS_Pre_Limit:	Diagnostic	PV_FF_Trim	Dynamic Input
Cutback_High:	Control	Q:	Tuning
Cutback_Low:	Control	Ramp_Rate:	Control
Debump:	Dynamic Input	Rel_Ch2_Gain:	Output
Debump_Dis:	Dynamic Input	Sensor_Break:	Dynamic Input
Deriv_On_PV:	Configuration	Setpoint:	Dynamic Input
Derivative:	Control	SP_FF_Enable:	Dynamic Input
Direct:	Configuration	SP_FF_Trim:	Dynamic Input
Error:	Diagnostic	Span_High:	Configuration
Feed_Forward:	Dynamic Input	Span_Low:	Configuration
Feedback:	Diagnostic	Status:	Diagnostic
Inhibiter:	Control	Track_Enable:	Control
Integral:	Control	Track_Value:	Control
Integral_Hold:	Control	Trigger_Val:	Tuning
Manual:	Control	Tune_Type:	Tuning

Table 9-11 Parameter Classification

Function Block Configuration Parameters

For optimum operation, the function block must be configured to the type of control problem that it is to be applied to. This configuration is performed by appropriate setting of the input parameters. The use of these configuration parameters is described below.

Span_High and Span_Low.

Span_High and Span_Low define the maximum and minimum limits of the working range of the function block. Generally these are set to values which represent physical boundaries in the operation of the process, such as the calibrated range of a transducer, or the safe limits of a pressure vessel. The proportional band of the PID algorithm is defined as a percentage of the span of the function block, which is found by subtracting Span_Low from Span_High. If the Process_Val or Setpoint move outside the span the function block will enter a sensor break condition.

Direct

Direct defines whether the function block is direct acting or reverse acting. If the function block is direct acting, the Output will tend to increase if the Process_Val is greater than the Setpoint. If the function block is reverse acting, the Output will tend to increase if the Process_Val is less than the Setpoint.

Deriv_On_PV

Deriv_On_PV defines whether the derivative action is responds to changes to Process_Val only (On_PV (1)) or to changes to the difference between the Setpoint and the Process_Val (On_Err (0)).

Break_Output

Break_Output defines the output level to which the function block will default when a sensor break condition has been detected. This is either triggered by Sensor_Break being set to Break (1), or by the Process_Val moving outside the span of the function block ±10% or Setpoint moving outside span.

Control Parameters

The function block has several parameter which are not intended to be changed dynamically, but instead control the operation of the algorithm. These include the tuning parameters, such as **Prop_Band,** as well as booleans which are used to change the mode of operation. These parameters are described below.

Manual

Manual is used to select whether the function block operates in **Manual** or **Auto** mode. In **Manual** mode, the PID algorithm is disabled and the value of **Output** is taken directly from its input. In **Auto** mode the full controller functionality is active.

PropUnits (Version 3.00 onwards)

PropUnits is to define whether the **Prop_Band** is defined as a percentage of the controller span (PctSpan (0)) or in engineering units (EngUnts (1)).

Prop_Band

Prop_Band is the proportional band of the PID control algorithm.

Prior to Version 3.00, the proportional band could only be defined as a percentage of span. The engineering units mode was not supported.

If **PropUnits** is set to a percentage of span (PctSpan (0)), the proportional band is defined as the percent of the total span of the controller for which a control error will produce an output signal equivalent to the maximum output of the instrument. For example, if **Span_High** is 1000, **Span_Low** is 0, Setpoint is 500, **Process_Val** is 490 and **Prop_Band** is 1%, a proportional only reverse acting controller will produce an output of +100%, because the control error will be 10 units, which is one proportional band.

Note:- When defined as a percentage of span, the the proportional gain is given by:

$$\text{Gain} = \frac{10{,}000}{(\text{Span\_High} - \text{Span\_Low})\ {*}\text{Prop\_Band}}$$

If **PropUnits** is set to engineering units (EngUnts (1)), the proportional band is defined as the magnitude of the control error which will produce an output signal equivalent to the maximum output of the instrument. For example, if **Setpoint** is 500°C, **Process_Val** is 495°C and **Prop_Band** is 5°C, a proportional only reverse acting controller will produce an output of +100%, because the control error is 5°C which is equal to one proportional band.

When defined in Engineering units, the proportional gain is given by:

$$\text{Gain} = \frac{100}{\text{Prop\_Band}}$$

Integral

Integral is the integral time constant of the PID control algorithm. **Integral** time is defined as the time period in which the part of the output signal due to integral action increases by an amount equal to the part of the output signal due to proportional action, for a constant error state.

Derivative

Derivative is the derivative time constant of the PID control algorithm. **Derivative** time is defined as the time interval in which the part of the output signal due to derivative action increases by an amount equal to the part of the output signal due to proportional action, when the control error is changing at a constant rate.

Feed_Forward

Feed_Forward is added directly to the output of the PID algorithm, before the output limiting and dual output conversions are performed.

Manual_Reset

Manual_Reset is only active when Integral is set to zero. It provides an offset to the Output, which can be used to reduce the control error to zero when integral action is not employed.

Cutback_High and Cutback_Low

Cutback can be employed to reduce the amount of time it takes the Process_Val to respond to large changes in Setpoint and to limit the overshoot that can occur during the transient period. Cutback_High operates when the Process_Val is initially greater than the target Setpoint and Cutback_Low operates when the Process_Val is initially less than the target Setpoint. The units of Cutback_High and Cutback_Low are engineering units. Cutback operates by forcing the Output to its appropriate maximum or minimum limit in response to a Setpoint change which is greater than the cutback band. As the Process_Val approaches the Setpoint, the control error (Setpoint - Process_Val) reduces to less than the cutback value and normal control is resumed.

Track_Enable and Track_Value

Track_Enable allows the PID algorithm to be disabled and the Output signal to be read directly from Track_Value. These parameters are generally used when the function block forms part of a cascade control application.

Integral_Hold

Integral_Hold enables the integral output to be frozen at its current value. It will remain constant throughout the period that Integral_Hold is enabled.

Dynamic Input and Output Parameters

Setpoint and Process_Val

The Process_Val is the controlled variable of the function block and the Setpoint is the target value against which the Process_Val is controlled. The PID algorithm acts to reduce the difference between the Setpoint and the Process_Val to zero.

Zero_Deriv

Zero_Deriv can be used to force the derivative output to zero. It is automatically cleared by the function block after one sample.

Integral_Hold

Integral_Hold enables the integral output to be frozen at its current value. It will remain constant throughout the period that Integral_Hold is enabled.

Output Parameters

Output, Ch1_Output and Ch2_Output

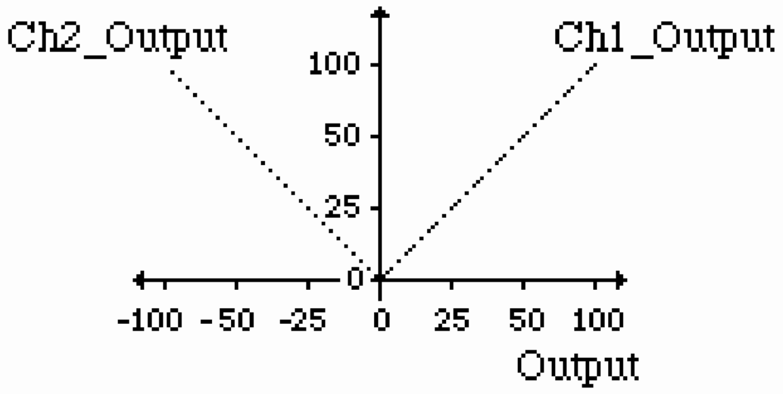


Figure 9-48 Dual Output Relationship, with Unity Rel_Ch2_Gain and zero Ch1_Ch2_D_B

The PID function block channel can be configured for either single or dual channel operation. In single channel operation, the control is on-Output, which is limited within the range + 100 % to 0 % by the parameters Output_High and Output_Low. Dual channel operation is designed for systems such as heat-cool applications, in which negative values of Output must be output to a refrigeration unit as absolute values to increase the refrigeration rate. For dual channel operation Output_Low is set to - 100%.

Output_High and Output_Low

Output_High and Output_Low define the upper and lower limits of Output They must both be set within the range -100% to +100%, with Output_High being greater than or equal to Output_Low. When the function block is being used for dual output operation, Ch1_Output and Ch2_Output are related to Output as shown in Figure 9-7, with Output being bounded by Output_High and Output_Low.

Rel_Ch2_Gain and Ch1_Ch2_D_B

When the function block is being used for dual channel control, the full relationship between Ch2_Output and Output is given by:

Ch2_Output = (Output + Ch1_Ch2_D_B) * Rel_Ch2_Gain

Rel_Ch2_Gain is intended for use in non linear dual output control situations, such as heat / cool systems, to compensate for the differing gains of the equipment being driven by the two output channels. Ch1_Ch2_D_B introduces a deadband between the two output channels, which can either be set to a positive value to provide a region in which neither channel is active, or to a negative value to provide a region of overlap in which both outputs are active.

Output_Rate and Out_Rate_En

Output_Rate can be used to limit the maximum rate of change of Output per second. It is enabled by setting Out_Rate_En to Yes (1). Note that Output_Rate affects the rate of change of Output directly, so that the rate of change of Ch2_Output will be modified by the Rel_Ch2_Gain.

Output_Rate can be used to limit the maximum rate of change of Output per second. It is enabled by setting Out_Rate_En to Yes (1). Note that Output_Rate affects the rate of change of Output directly, so that the rate of change of Ch2_Output will be modified by the Rel_Ch2_Gain.

Ch1_Cycle_T and Ch2_Cycle_T

Ch1_Cycle_T and Ch2_Cycle_T are the cycle times of channels 1 and 2, which can be used when the function block is connected to time proportioned outputs. The values of the two cycle times will be automatically set by the Autotuner on completion of the tuning sequence, only if their initial value is greater than 5s.

Ch2_Linear

Ch2_Linear is a boolean, which provides an indication to the tuning and control algorithms that the output to channel 2 is linear. If Ch2_Linear is Yes (1), the tuning sequences will operate with equal peak output levels on both output channels. If Ch2_Linear is No (0), the output to channel 2 will be limited to 20 % and the tuner will tune the control parameters to suit the water cooling outputs relationship.

Ch1_CT_Low and Ch2_CT_Low

Ch1_CT_Low and Ch2_CT_Low define the lower limits of the cycle times of output channels 1 and 2 which can be set by the autotuner when the function block is being used to drive time proportioned outputs.

Sensor_Break

The sensor break input provides a trigger which can be used to set the function block into a sensor break condition. When the function block is in sensor break, the Output will be set to Break_Output and the PID algorithm will be disabled. On leaving the sensor break condition, the algorithm will hold the output at Break_Output for sixteen samples, to ensure that the sensor has been properly re-acquired.

Debump and Debump_Dis

Debump functionality is employed by the function block to ensure that changes to operating conditions or control parameters do not result in sharp deviations of the Output signal. In the PC3000, debumping is automatically carried out whenever changes are made to Prop_Band, Derivative, Span_High, Span_Low, Ch1_Ch2_D_B, Rel_Ch2_Gain, or when the mode of the function block is switched between Manual and Auto. The parameter Debump can be used to trigger debumping for other changes, such as changes to the Setpoint. Debump_Dis can be used to disable the debump functionality, which will allow the Output to "kick" in response to changes in the parameters listed above.

SP_FF_Enable and SP_FF_Trim

SP_FF_Enable is used to enable Setpoint feedforward functionality, which is generally employed when the function block is being used as part of a cascade control application. SP_FF_Trim is used to limit the peak level of the setpoint feedforward trim term. The units of SP_FF_Trim are in percent of the span of the Output signal.

PV_FF_Enable and PV_FF_Trim

These are used when PV feedforward is required in cascade. The functionality is just as setpoint feedforward with process value replacing the setpoint.

The PID_Auto tuning algorithms

The Autotune Algorithm

The autotune algorithm is a one shot tuner which was previously implemented in the Eurotherm 818 and 815 instruments. The principles of operation of this tuner are described in detail in the PC3000 Control Overview. The description given here is intended as an outline to enable the user to understand its role within PID_Auto.

When the autotune algorithm is selected the PID control algorithm is disconnected from the process, with the autotuner driving the controller's outputs directly using ON/OFF control. On selection of autotune, if the current process value is within 1% of setpoint, the outputs are frozen at their existing values, otherwise the outputs are set to zero. This output value is held for one minute, during which the autotuner monitors the process to asses the noise level and the action of the process value. During this first minute the operator can select the setpoint at which the tuning is to be performed. This can be the current setpoint, if required, or a setpoint greater than or less than the current value.

When the one minute monitoring cycle has elapsed, the autotuner cycle will move into its active phase. If the current setpoint is more than 1% of span greater than the current process value, a tune up to setpoint will be performed. If the setpoint is more than 1% of span less than the current process value, a tune down to setpoint will be performed. If the current setpoint is within 1% of span of the process value, a tune at setpoint will be performed. These are described in detail in the references quoted above. The process value and output curves for the three types of autotune cycles, for a heat only instrument, are shown in the figure 9-49.

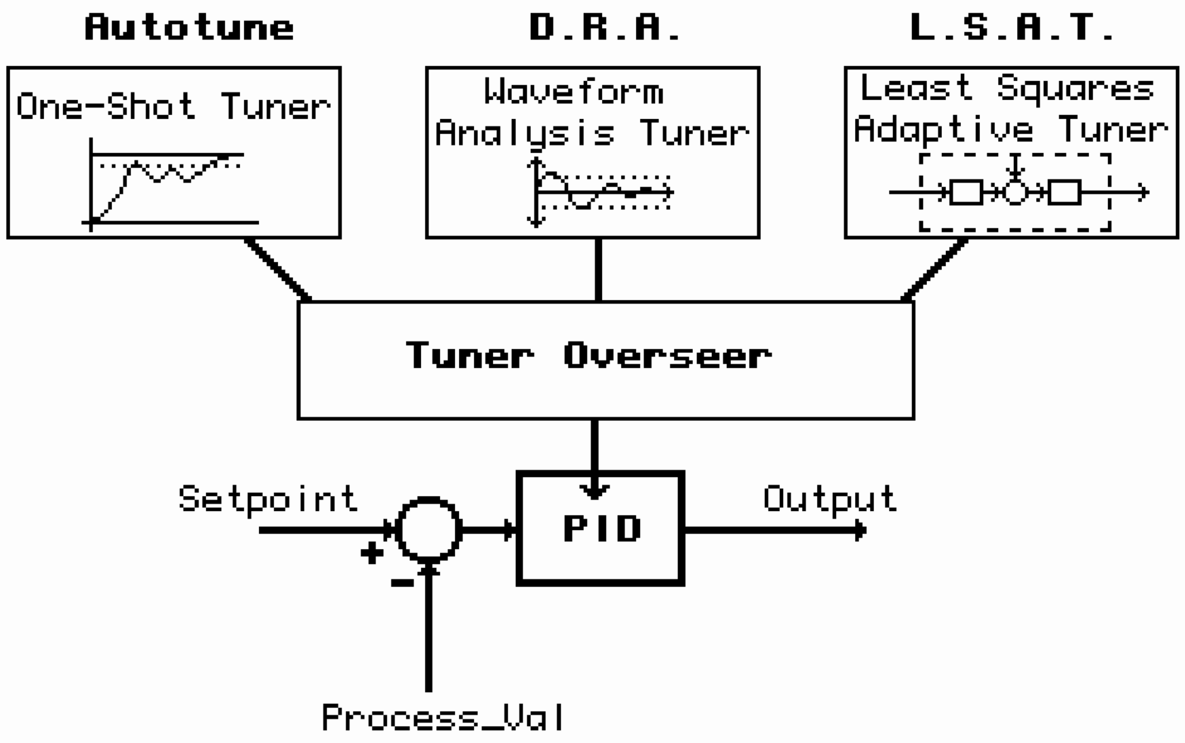


Figure 9-49 Heat only autotune cycles.

At the end of the autotune cycle, the following parameters are calculated and loaded into the controller:

Proportional Band	
Integral time	
Derivative time	
Cutback High	(tune down to setpoint only)
Cutback Low	(tune up to setpoint only)
Ch1 Cycle Time	(if time proportioned output)
Ch2 Cycle Time	(if time proportioned output and cool channel selected)
Rel Ch2 Gain	(if cool channel selected)
Trigger Val	(used by DRA and LSAT)
MTC	(used by LSAT)
Q	(used by LSAT)

The Disturbance Response Analysis Algorithm (DRA)

The DRA is an adaptive tuner which was previously implemented in the Eurotherm 818 instrument. It is an elementary expert system, which acts by identifying patterns in the error (setpoint–process value) response and adjusting the controller parameters to compensate for responses which are slow or oscillatory. Unlike the autotuner, DRA does not drive the controller outputs directly.

DRA is triggered into monitoring the control response when the absolute error exeeds the Trigger Value, which can be set manually but is automatically set by the autotuner. Once triggered by a disturbance, such as a setpoint change or a load disturbance, DRA monitors up to two cycles of oscillation of the error before deciding what, if any, modification of the control terms is required.

The ratio of peaks of the error (the damping ratio) is the main criteria which is used to decide whether retuning is required. If the amplitude of the peaks is small, the tuner disables itself and will not retune the controller parameters.

In the 818 implementation of DRA, if the algorithm determined that retuning was necessary, the Porportional Band, Integral time and Derivative time were adjusted. In the PID_Auto implementation, DRA also adjusts the LSAT referencing terms MTC and Q. MTC is adjusted in series with the Integral and Derivative times. Q is adjusted with the Proportional Band.

If the response is analysed to be oscillatory, the oscillation time is used to determine whether the gain, the integral and derivative times, or both the gain and the times require adjustment.

DRA adaptive tune strategy selection.

The Least Squares Adaptive Tuner algorithm (LSAT).

The LSAT algorithm is an adaptive tuner which, like the DRA algorithm, does not drive the controller outputs directly, but instead tunes the PID control coefficients. The LSAT should be considered to consist of two main parts:

{1} A process model identifier

{2} A controller designer.

The algorithm looks at the controller output and the process value. It continually feeds the controller output into an internal model of the process and compares the process value predicted by this model with the actual process value.

The recursive least squares algorithm is used to adjust the parameters of the model so that the predicted process value matches the actual process value. In this way the model is continuously being fine tuned so that it matches the actual process being controlled. The model is then used by the controller designer to define and set the optimum PID parameters for the process being controlled. This employs a model–reference approach, in which the closed–loop performance of the system is optimised against that of the internal reference closed–loop model. A block diagram of the LSAT algorithm is shown below.

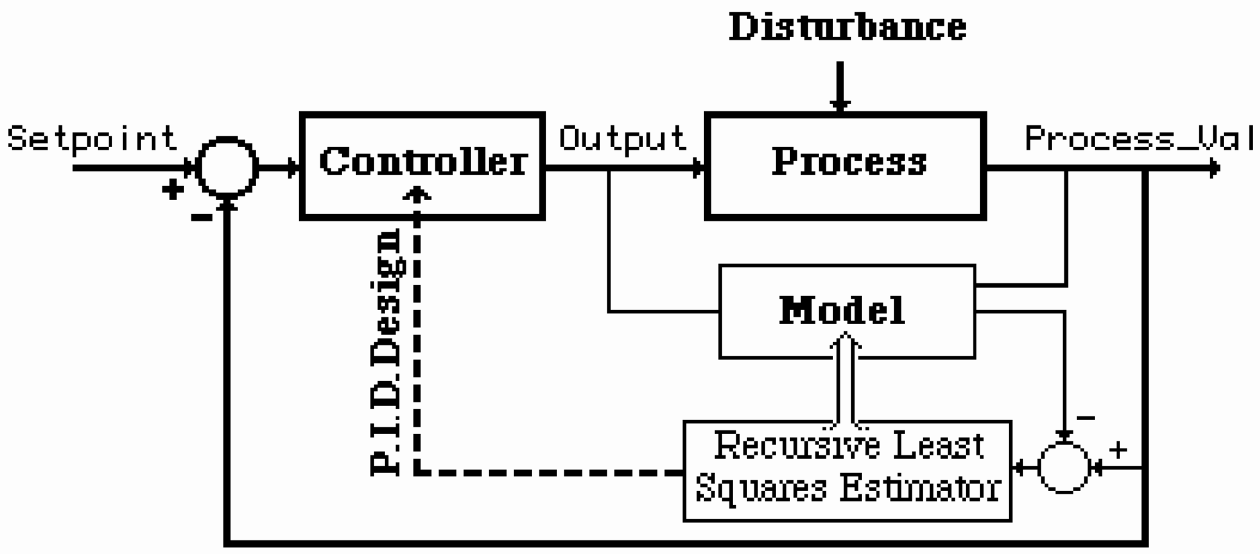


Figure 9-50 Block diagram of the Least Squares Adaptive Tuner.

An advantage of the LSAT is that, unlike the DRA, it does not require large process disturbances or step setpoint changes in order to act. It can fine tune the controller during regulatory control, since it is able to filter elements of the process noise signal and utilise these in the model identification.

The fine tuning action of the LSAT is an important feature when implemented in the PID_Auto adaptive tuner. It enables it both to complement the coarser tuning action of the DRA and to fine tune the parameters set by the Autotuner.

The procedure of model identification through prediction and design through model reference enables the LSAT to continuously tune the controller parameters during on–line control operations. However, the LSAT algorithm is not able to determine the structure of the process model, it fits parameters to a model whose structure is pre–defined. This restriction could limit its suitability for

implementation as a stand–alone general purpose adaptive tuner, so the algorithm has been developed to adapt its identification and parameter design in response to external referencing. This takes the form of two parameters – "MTC", a process time constant scaler from which the LSAT determines its sample frequency and "Q", which gives an indication of the unmodelled (high frequency) process coefficients. Although this information can be input manually, it requires a high level of LSAT expertise and process knowledge, so the Autotune and DRA algorithms have been developed to identify MTC and Q and to prime the LSAT with these values. The resulting three tuner interaction forms the "Composite" tuner.

The Operation of the PID_Auto Tuners.

The Composite tuner combines the functionality of the three tuners described above, Autotune, DRA and LSAT. It has been developed to enable the strengths of the three algorithms to be utilised to provide the optimum tuning performance under the widest range of control conditions. The functions performed by the three tuning algorithms of the composite tuner may be summarised as:

{i} Autotune: initial one–shot tuning of PID parameters, coarse identification of process model, identification of process noise levels.

{ii} DRA: coarse adaption to changes in process operating conditions, coarse adjustment of LSAT reference model, adjustment of controller when necessary in response to large process disturbances.

{iii} LSAT: fine tuning of PID parameters after Autotune and during adaptive tune sequence, fine tuning during setpoint ramps and in response to small process changes or small disturbances.

Selecting the Autotuner will result in the one–shot tuner sequence being activated. Selecting Adaptive tune will activate both the DRA and the LSAT adaptive tuning algorithms together. It should be noted that **the optimum tuning performance will be obtained when AUTO/TUNE precedes ADAPT/TUNE.** The Autotune algorithm has been enhanced to provide the initialisation that LSAT requires. If Autotune does not precede Adaptive, then DRA is required to initialise LSAT, which can require several DRA adaption sequences to take place and can result in a very slow tuning of the controller.

When Autotune followed by Adaptive tune is selected, an Autotune sequence is performed followed by an automatic switch to Adaptive tune. The Adaptive Tune sequence then begins by disabling the LSAT until the estimator has converged on the process model. Once the LSAT has converged, the Adaptive Tune sequence then continues with both the DRA and the LSAT algorithms tuning the controller, under the guidance of the tuner overseer software. The sequence of AUTO/TUNE followed by ADAPT/TUNE is represented below for a tune up to setpoint Autotune.

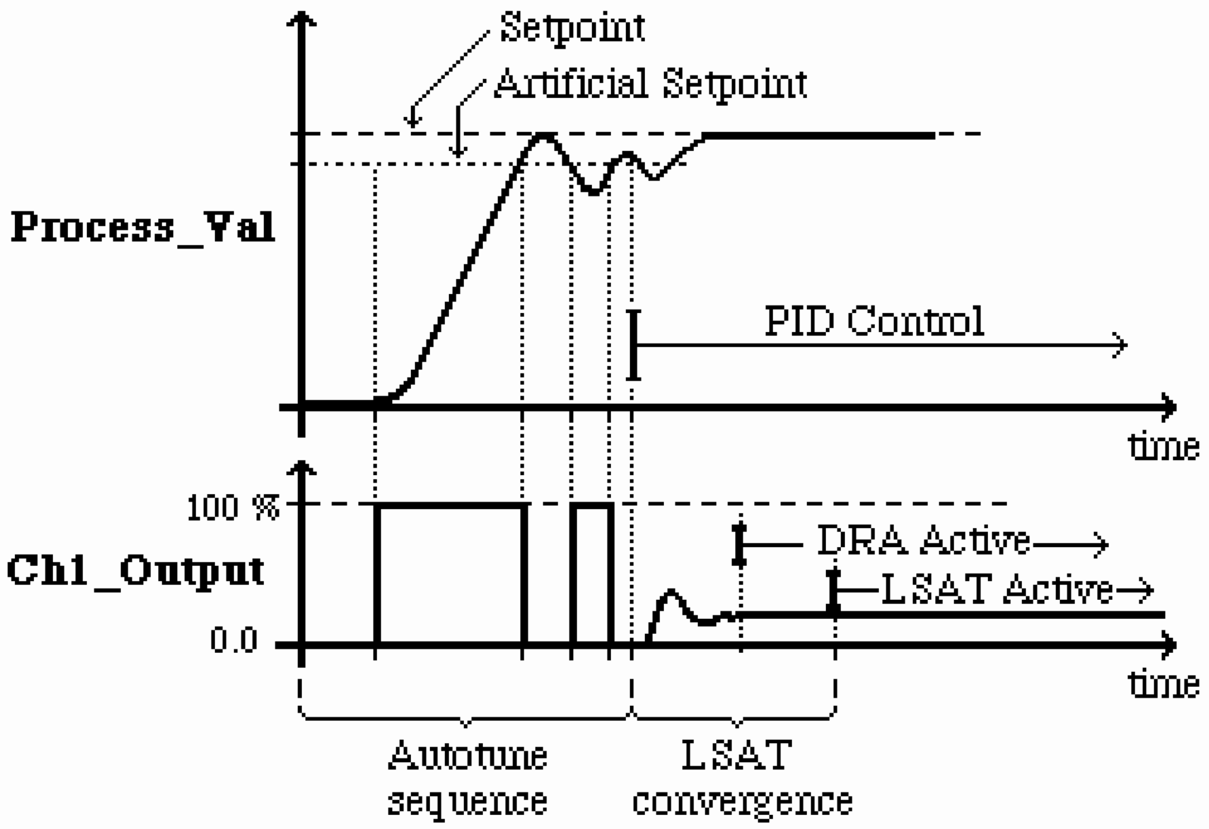


Figure 9-51 Autotune–Adaptive tune convergence sequence during a start up tune.

Function Block Tuning Parameters

{1} Tune_Type

Tune_Type defines which tuner, if any, is active. It can be set to one of eight values:

None (0):	No tuners active.
AT (1):	Autotuner active.
DRA (2):	Adaptive tune (DRA) active.
LSAT (3):	Adaptive tune (DRA and LSAT)) active.
DRA_LS (4):	Adaptive tune (DRA and LSAT) active.
AT_DRA (5):	Autotuner followed by Adaptive tuner (DRA) active.
AT_LSAT (6):	Autotuner followed by Adaptive tuner (DRA and LSAT) active.
AT_D_L (7):	Autotuner followed by Adaptive tuner (DRA and LSAT) active.

Param_Change

Param_Change is set by the tuners to indicate that the value of one of the parameters has been changed. Once it has been changed by a tuner to Yes (1), it must be reset externally to No (0).

Trigger_Val

Trigger_Val provides an indication to the Adaptive tuner of the amount of process noise which is present on the Process_Val. It is automatically set by the Autotuner during the tuning sequence, but can be set manually by the user.

AT_Out_High and AT_Out_Low

AT_Out_High and AT_Out_Low limit the output swing during the autotune (on firmware versions after 2-27). Prior to this version, OutputHigh and Output_Low may be used to limit output swing during auto tune.

Tuning

Tuning provides an indication that the tuners are functioning. If Auto or Adaptive tuners are active, Tuning will be Active (1), else Tuning will be Off (0)

Function Block Overshoot Inhibition Function

The PID_Auto function block includes the ability to inhibit the Process_Val overshoot that can occur at the end of Setpoint ramps. To use the inhibition function, it is necessary to provide the function block with information about the target setpoint and rate of the ramp and to tune the level of overshoot inhibition required. This is performed using the four inhibition inputs.

Inhibiter

Inhibiter defines the amount of overshoot inhibition required. It can be set to a value between 0 and 1. A value of 0 indicates that the inhibition is off.

Ramp_Rate and Ramp_Units

Ramp_Rate defines the rate of change of the Setpoint during the ramp. Its units are set by Ramp_Units.

Target_SP

Target_SP is the final target value to which the Setpoint is ramping.

In a correctly tuned PID controller, setpoint overshoot generally occurs at the end of ramps because the integrator has wound up during the ramp. The overshoot inhibition function operates by reducing the integrator wind up towards the end of the ramp, which removes the "momentum" of the process value and enables it to approach the target setpoint without overshoot.

The inhibition function is initialised at the start of the setpoint ramp, when the value of Target_SP changes. The point at which the integrator wind up is discharged and the amount of discharge is then determined according to the gain and time constant of the controller, the Ramp_Rate and the value of Inhibiter, which can be set within the range 0 to 1. Examples of overshoot performance for no inhibition, ideal Inhibiter setting and Inhibiter settings too large and too small are illustrated in the figure below.

The algorithm has been designed to give overdamped performance with Inhibiter set to 0.5. When the setpoint ramp reaches the appropriate point near to the Target_SP, the integrator is discharged, which inhibits the overshoot. This is illustrated in figure {2}. If Inhibiter is set too large, the integrator will discharge too early then begin to wind up again, which will result in a delayed overshoot as shown in figure {3}. If Inhibiter is set too small, the integrator will discharge too late and overshoot will occur due to the stored momentum of the process value, which is illustrated in figure {4}.

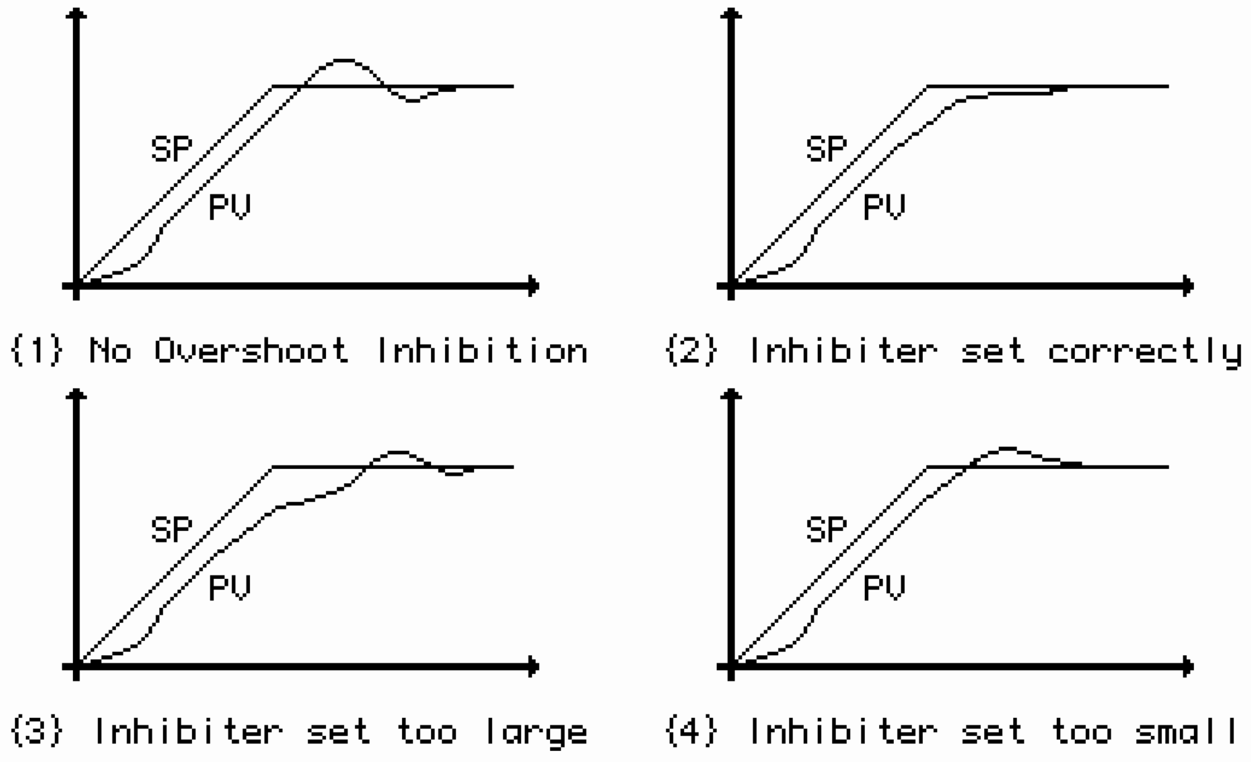


Figure 9-52 Setpoint Ramp Responses Illustrating the Effect of Inhibition.

Function Block Diagnostic Parameters

MTC

This is the LSAT reference model time constant.

Q

Q is the LSAT reference detuning factor.

CS_Pre_Limit

This is the output of the PID controller after the Feed_Forward has been added, but before Output limiting has been performed.

Control_Sig

Control_Sig is the sum of the proportional, integral, derivative and feed forward components after being high, low and rate of change limited, but before the dual output relative gain and deadband have been added.

Nerrer, Integral_Out and Deriv_Out,

These are the outputs of the proportional, integral and derivative components respectively.

Feedback

It is the value of the signal which is fed back into the PID to indicate the actual signal that is output by the controller.

Error

The difference between the Process_Val and the Setpoint.

AT_State

AT_State indicates the state of the Autotuner.

0	Reset
1	Initiatisation
2	Monitor quiescent noise
3	End of monitor noise
4	Startup with a new setpoint
5	End of startup with new setpoint
6	Startup with PV at setpoint
7	End of startup with PV at setpoint
8	Zeigler-Nichols sequence
9	Calulate new parameters
10	Write update status
11	Autotune aborted
12	Autotune completed

Table 9-12 AT_State Values

ZN_Stage

ZN_Stage indicates the state of the Autotuner's Ziegler-Nichols tuning stages.

3	Find peak PV & reverse output
4	Find PV crossing PV1 & test again for dominant delay
5	Find peak PV & either reverse PV change or reverse output
6	Find PV crossing PV1 & adjust trend and output
7	Find peak PV & reverse output
8	Find PV crossing PV1 & calculate new parameters

Table 9-13 Zn_State Values

DRA_State

DRA_State indicates the state of the DRA adaptive tuner.

0	Allow settling
1	Wait for trigger
2	Find peak 1
3	Find zero 1
4	Find peak 2
5	Find zero 2
6	Find peak 3
7	Find zero 3
8	Find peak 4
9	Find zero 4
10	Find peak 5
11	End on zero 4 abort
12	End on peak 4 found
13	End on peak 5 abort
14	End on peak 5 found
15	Prepare update

Table 9-14 DRA_State Values

DRA_Last

DRA_Last records the last tuning strategy performed by the DRA adaptive tuner.

0	No change
1	Reduce Damping
2	Increase gain
3	Decreased Times
4	Increased Times
5	Decreased Gain

Table 9-15 DRA_Last Values

LSAT_F1

LSAT_F1 indicates the form of the process model which has been identified by the LSAT.

Proc_Delay

Proc_Delay is the estimated delay time of the open-loop system which has been estimated by the Autotuner.

Status

Status provides an indication of the function of the PID_Auto function block. it can have eight possible states:

OK (0):	The function block is operating normally
SnsrBrk (1):	An external sensor break has been detected
PV_High (2):	The Process_Val is greater than Span_High +10%
PV_Low (3):	The Process_Val is less than Span_Low -10%
SP_High (4):	The Setpoint is greater than Span_High
SP_Low (5):	The Setpoint is less than Span_Low
GainNeg (6):	The Prop_Band has been set with a negative value, or Span_High has been set less than Span_Low
GainHi (7):	The Prop_Band has been set with too small a value or the span of the function block has been set too small, resulting in a very large gain.

Parameter Attributes

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
AT_Out_High	**REAL**	0	Config	Config	High Limit Low Limit	Output_High AT_Out_Low
AT_Out_Low	**REAL**	0	Config	Config	High Limit Low Limit	AT_Out_High AT_Out_Low
AT_State	**SINT**	0	Config		High Limit Low Limit	255 0
Break_Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Ch1_Ch2_D_B	**REAL**	0	Oper	Oper	High Limit Low Limit	10 -10
Ch1_CT_Low	**TIME**	0	Super	Super	High Limit Low Limit	1d 0
Ch1_Cycle_T	**TIME**	5s	Super	Super	High Limit Low Limit	1d 0
Ch1_Output	**REAL**	0	Oper		High Limit Low Limit	100 The higher of 0 or Output_Low
Ch2_CT_Low	**TIME**	0	Super	Super	High Limit Low Limit	1d 0
Ch2_Cycle_T	**TIME**	5s	Super	Super	High Limit Low Limit	1d 0
Ch2_Linear	**BOOL**	Yes (1)	Super	Config	Senses	No (0) Yes (1)
Ch2_Output	**REAL**	0	Oper		High Limit Low Limit	The lower of 0 or Output_High -100

Table 9-16 PID_Auto Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Control_Sig	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
CS_Pre_Limit	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Cutback_High	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Cutback_Low	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Debump	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Debump_Dis	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Deriv_On_PV	**BOOL**	On_Err (0)	Config	Config	Senses	On_Err (0) On_PV (1)
Deriv_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Derivative	**TIME**	50s	Oper	Oper	High Limit Low Limit	01d_03h 0
Direct	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
DRA_Last	**SINT**	0	Config		High Limit Low Limit	255 0
DRA_State	**SINT**	0	Config		High Limit Low Limit	255 0
Error	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Feed_Forward	**REAL**	0	Super	Super	High Limit Low Limit	100 -100
Feedback	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000

Table 9-16 PID_Auto Parameter Attributes (continued)

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Inhibiter	**REAL**	0	Oper	Oper	High Limit Low Limit	1 0
Integral	**TIME**	5m	Oper	Oper	High Limit Low Limit	01d_03h 0
Integral_Hold	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Integral_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
LSAT_F1	**REAL**	0	Config		High Limit Low Limit	100,000 0
Manual	**BOOL**	Manual (1)	Oper	Oper	Senses	Auto (0) Manual (1)
Manual_Reset	**REAL**	0	Oper	Oper	High Limit Low Limit	100 -100
MTC	**REAL**	30	Config	Config	High Limit Low Limit	100,000 0.1
Nerror	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Out_Rate_En	**BOOL**	No (0)	Oper	Config	Senses	No (0) Yes (1)
Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Output_High	**REAL**	100	Oper	Oper	High Limit Low Limit	100 Output_Low
Output_Low	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High -100
Output_Rate	**REAL**	10	Oper	Oper	High Limit Low Limit	10,000 0.1

Table 9-16 PID_Auto Parameter Attributes (continued)

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Param_Change	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Proc_Delay	**TIME**	0	Config		High Limit Low Limit	1d 0
Process_Val	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Prop_Band	**REAL**	5 %	Oper	Oper	High Limit Low Limit	10,000 0.1
PropUnits	**BOOL**	PctSpan (0)	Oper	Oper	Senses	PctSpan (0) EngUnts (1)
Q	**REAL**	0.4	Config	Config	High Limit Low Limit	100,000 0
Ramp_Rate	**REAL**	0	Oper	Oper	High Limit Low Limit	10,000 0
Ramp_Units	**ENUM**	/ Second (0)	Oper	Super	Senses	/ Second (0) / Minute (1) / Hour (2) / Day (3)
Rel_Ch2_Gain	**REAL**	1	Oper	Oper	High Limit Low Limit	10 0.1
Sensor_Break	**BOOL**	Ok (0)	Oper	Super	Senses	Ok (0) Break (1)
Setpoint	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
SP_FF_Enable	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
SP_FF_Trim	**REAL**	0	Config	Config	High Limit Low Limit	100,000 -100,000
PV_FF_Enable	**BOOL**	No(0)	Super	Config	Sences	No (0) Yes (1)
PV_FF_Trim	**REAL**	0	Config	Config	High Limit Low Limit	100,000 -100,000
Span_High	**REAL**	100	Config	Config	High Limit Low Limit	100,000 Span_Low

Table 9-16 PID_Auto Parameter Attributes (continued)

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Span_Low	**REAL**	0	Config	Config	High Limit Low Limit	Span_High - 100,000
Status	**ENUM**	Ok (0)	Oper		Senses	Ok (0) SnsrBrk (1) PV_High (2) PV_Low (3) SP_High (4) SP_Low (5) GainNeg (6) GainHi (7)
Target_SP	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Track_Enable	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Track_Value	**REAL**	0	Super	Config	High Limit Low Limit	Output_High Output_Low
Trigger_Val	**REAL**	0.1	Config	Config	High Limit Low Limit	100,000 0
Tune_Type	**ENUM**	None (0)	Config	Config	Senses	None (0) AT (1) DRA (2) LSAT (3) DRA_LS (4) AT_DRA (5) AT_LSAT (6) AT_D_L (7)
Tuning	**BOOL**	Off (0)	Super		Senses	Off (0) Active (1)
Zn_Stage	**SINT**	0	Config		High Limit Low Limit	255 0

Table 9-16 PID_Auto Parameter Attributes (continued)

VP_AUTO FUNCTION BLOCK

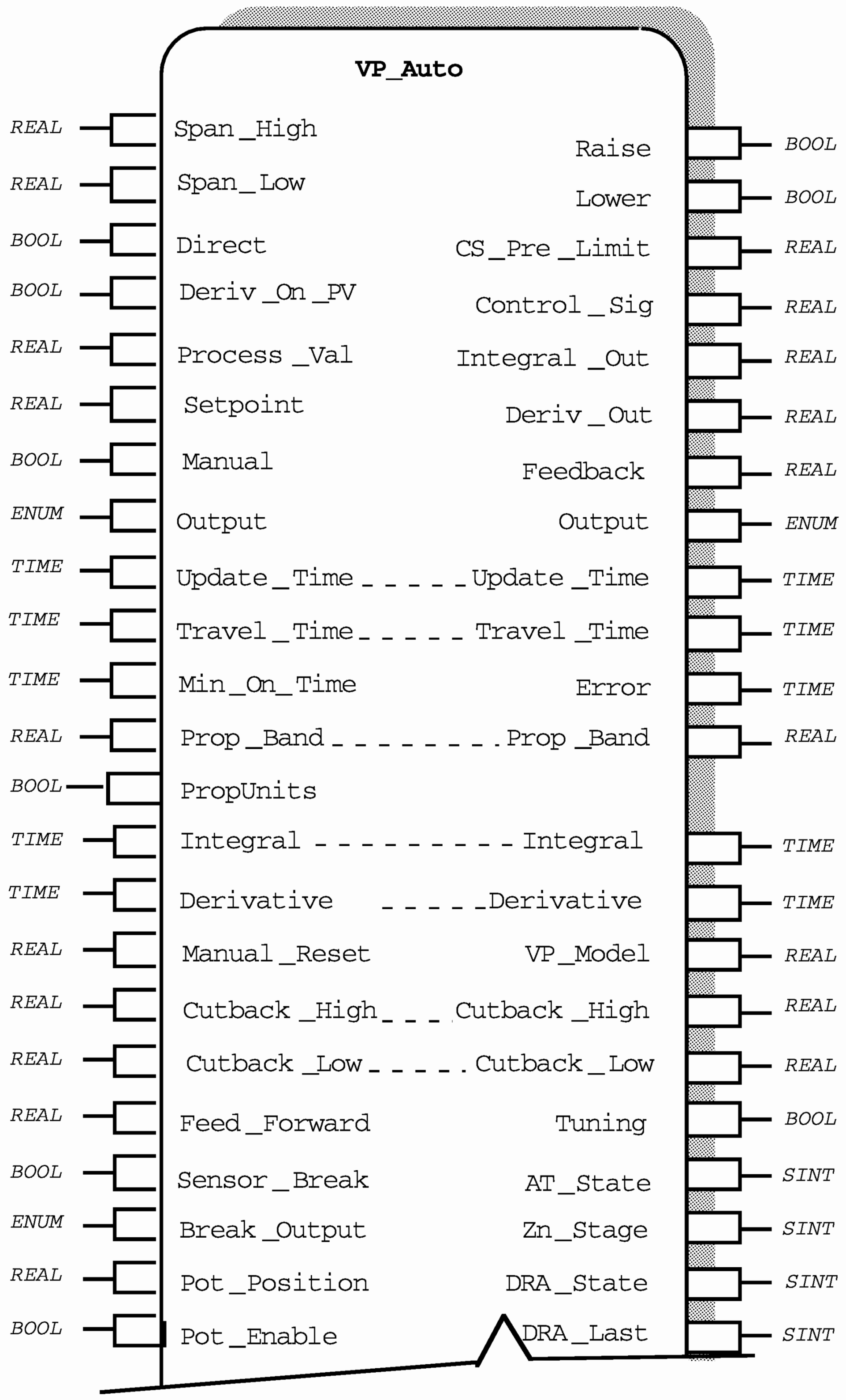


Figure 9-53 VP_Auto Function Block Diagram

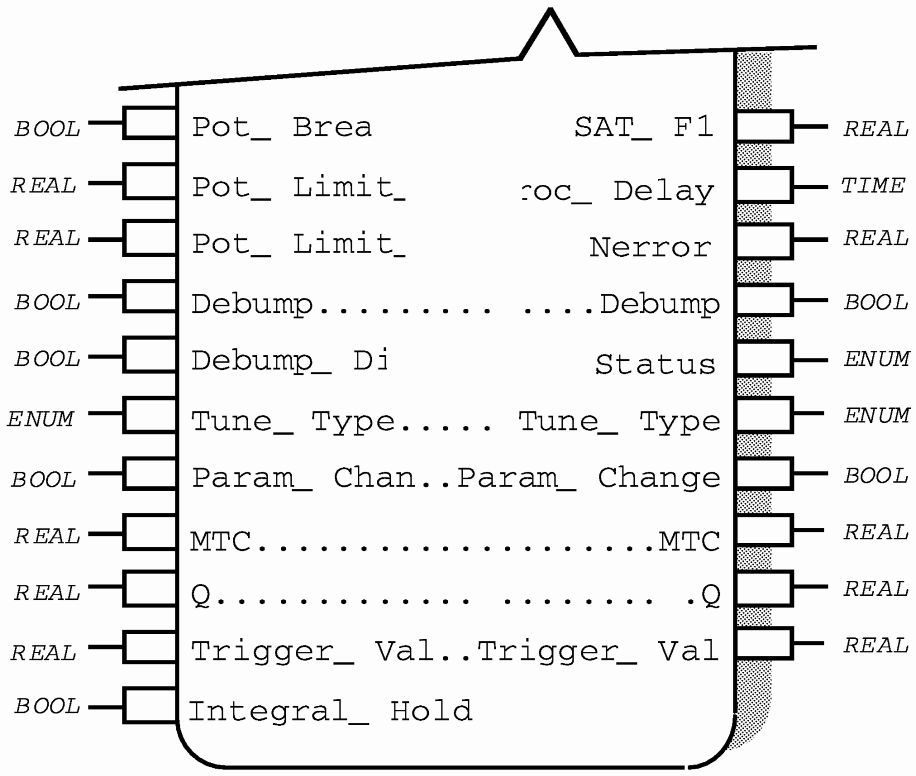


Figure 9-53 VP_Auto Function Block Diagram (continued)

Functional Description

The VP_Auto function block implements the proportional plus integral plus derivative control algorithm and the Auto and Adaptive tuning algorithms that are also used in the Eurotherm 900 series control instruments. The control algorithm is an enhanced version of that employed in the PID function block. The basic PID algorithm can be represented by the equation:

$$\text{Output} = \left(\frac{10000}{\text{Span *Prop\_Band}}\right)\left(E(t) + \frac{1}{\text{Integral}}\int E(t).dt + \text{Derivative} \,.\frac{d.E\,(t)}{dt}\right)$$

where E (t) is given by Setpoint - Process_Val and Span is given by Span_High - Span_Low. In the PC3000, this basic algorithm is supported by additional functionality to improve the control performance and to enable the function block to be configured to control a wide range of systems.

The function block's valve positioner stages use a boundless, or algorithm, which has been incorporated because it does not require position feedback from the valve, since this can often be unreliable. A model of the valve is included for valve position output control. Potentiometer position feedback can be used in conjunction with the valve model. In this mode of operation, the valve model position is used for control, with the actual valve position being used to limit the movement of the motor. This enables the controller to continue operating if the feedback potentiometer should fail.

VP_Auto is a complex function block. Achieving the optimum performance from the block requires that the controller, output stage and tuners are correctly configured and activated. A description of the separate elements of the function block is given below.

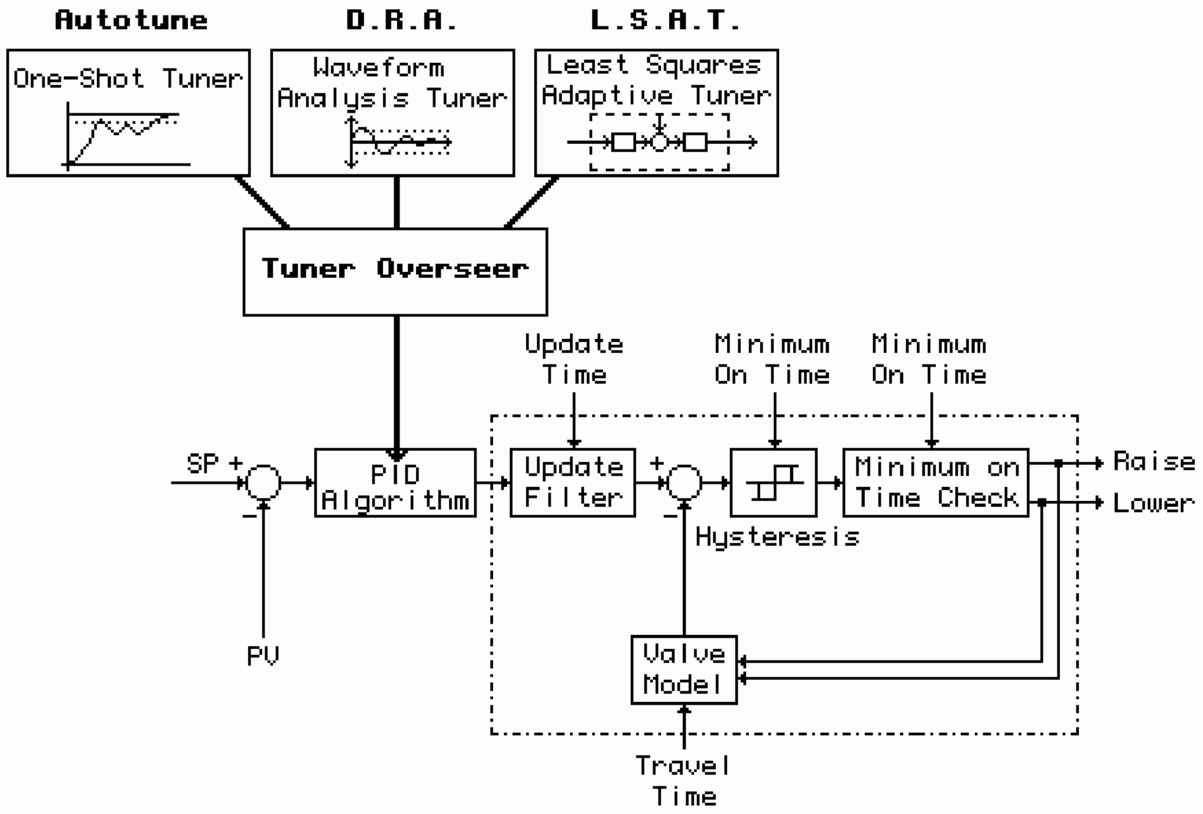


Figure 9-54 Block Diagram of the Principal VP_Auto Functionality.

Function Block Attributes.

Type:	20 60
Class:	CONTROL
Default Task:	Task_2
Short List:	Setpoint, Process_Val, Manual, Output
Memory Requirements:	1930 Bytes
Execution Time:	1.8 ms when not tuning
	3.73 ms maximum when tuning

Parameter Descriptions

AT_State:	Diagnostic	Nerror:	Diagnostic
Break_Output:	Configuration	Output:	Output
Control_Sig:	Diagnostic	Param_Change:	Tuning
CS_Pre_Limit	Diagnostic	Pot_Break:	Pot Feedback
Cutback_High:	Control	Pot_Enable:	Pot Feedback
Cutback_Low:	Control	Pot_Limit_Hi:	Pot Feedback
Debump:	Dynamic Input	Pot_Limit_Lo:	Pot Feedback
Debump_Dis:	Dynamic Input	Pot_Position:	Pot Feedback
Deriv_On_PV:	Configuration	Process_Val:	Dynamic Input
Deriv_Out:	Diagnostic	Proc_Delay:	Diagnostic
Derivative:	Control	Prop_Band:	Control
Direct:	Configuration	Q:	Tuning
DRA_Last:	Diagnostic	Raise:	Output
DRA_State:	Diagnostic	Sensor_Break:	Dynamic Input
Error:	Diagnostic	Setpoint:	Dynamic Input
Feed_Forward:	Dynamic Input	Span_High:	Configuration
Feedback:	Diagnostic	Span_Low:	Configuration
Integral:	Control	Status:	Diagnostic
Integral_Hold:	Control	Travel_Time:	Configuration
Integral_Out:	Diagnostic	Trigger_Val:	Tuning
Lower:	Output	Tune_Type:	Tuning
LSAT_F1:	Diagnostic	Tuning:	Tuning
Manual:	Control	Update_Time:	Configuration
Manual_Reset:	Control	VP_Model	Diagnostic
MTC:	Tuning	Zero_Deriv:	Control
Min_On_Time:	Configuration	ZN_Stage:	Diagnostic

Table 9-17 Parameter Classification

Controller Configuration Parameters

For optimum operation, the controller must be configured to the type of control problem that it is to be applied to. This configuration is performed by appropriate setting of the input parameters. The use of these configuration parameters is described below.

Span_High and Span_Low.

Span_High and Span_Low define the maximum and minimum limits of the working range of the function block. Generally these are set to values which represent physical boundaries in the operation of the process, such as the calibrated range of a transducer, or the safe limits of a pressure vessel. The proportional band of the PID algorithm is defined as a percentage of the span of the function block, which is found by subtracting Span_Low from Span_High. If the Process_Val moves outside the span the function block will enter a sensor break condition.

Direct

Direct defines whether the function block is direct acting or reverse acting. If the function block is direct acting, the Output will tend to increase if the Process_Val is greater than the Setpoint. If the function block is reverse acting, the Output will tend to increase if the Process_Val is less than the Setpoint.

Deriv_On_PV

Deriv_On_PV defines whether the derivative action is responds to changes to Process_Val only (On_PV (1)) or to changes to the difference between the Setpoint and the Process_Val (On_Err (0)).

Update_Time

Update_Time defines the sample time of the valve positioner output stages. Generally, this should be set to one tenth of the Travel_Time. Increasing the Update_Time will reduce the amount of activity of the valve, but can also reduce the accuracy of the control. Similarly, reducing the Update_Time can improve the control performance, but will also increase the amount of activity of the valve.

Travel_Time

Travel_Time is the amount of time it takes the valve to travel from the lower operating position to the upper operating position.

Min_On_Time

Min_On_Time defines the minimum time that the valve must remain in a state of opening, closing or remaining stationary.

Break_Output

Break_Output defines the output level to which the function block will default when a sensor break condition has been detected. This is either triggered by Sensor_Break being set to Break (1), or by the Process_Val moving outside the span of the function block.

Potentiometer Feedback Parameters

Pot_Position

Pot_Position is the input to which the valve position feedback potentiometer is connected.

Pot_Enable

When Pot_Enable is set to Yes (1), the potentiometer feedback algorithm is activated. Setting Pot_Enable to No (0) causes the function block to operate without potentiometer feedback.

Pot_Break

Pot_Break provides an external trigger to indicate that the potentiometer feedback has become disconnected, or is faulty. Setting Pot_Break to Yes (1) indicates a fault condition has occured. Setting Pot_Break to No (0) indicates the sensor is functioning normally.

Pot_Limit_Hi

Pot_Limit_Hi defines the upper operating position of the valve.

Pot_Limit_Lo

Pot_Limit_Lo defines the lower operating position of the valve.

Dynamic Input and Output Parameters

Setpoint and Process_Val

The Process_Val is the controlled variable of the function block and the Setpoint is the target value against which the Process_Val is controlled. The PID algorithm acts to reduce the difference between the Setpoint and the Process_Val to zero.

Output, Raise and Lower

When the function block is operating in Manual mode, Output serves as an input which can be used to directly control the Raise and Lower outputs to the valve. In Auto mode, Output is automatically set to Off (0).

Sensor_Break

The sensor break input provides a trigger which can be used to set the function block into a sensor break condition. When the function block is in sensor break, the Output will be set to Break_Output and the PID algorithm will be disabled. On leaving the sensor break condition, the algorithm will hold the output at Break_Output for sixteen samples, to ensure that the sensor has been properly re-acquired.

Debump and Debump_Dis

Debump functionality is employed by the function block to ensure that changes to operating conditions or control parameters do not result in sharp deviations of the Output signal. In the PC3000, debumping is automatically carried out whenever changes are made to Prop_Band, Derivative, Span_High, Span_Low, Ch1_Ch2_D_B, Rel_Ch2_Gain, or when the mode of the function block is switched between Manual and Auto. The parameter Debump can be used to trigger debumping for other changes, such as changes to the Setpoint. Debump_Dis can be used to disable the debump functionality, which will allow the Output to "kick" in response to changes in the parameters listed above.

Control Parameters

The function block has several parameter which are not intended to be changed dynamically, but instead control the operation of the algorithm. These include the tuning parameters, such as Prop_Band, as well as booleans which are used to change the mode of operation. These parameters are described below.

Manual

Manual is used to select whether the function block operates in Manual or Auto mode. In Manual mode, the PID algorithm is disabled and the value of Output is taken directly from its input. In Auto mode the full controller functionality is active.

PropUnits (Version 3.00 onwards)

PropUnits is to define whether the Prop_Band is defined as a percentage of the controller span (PctSpan (0)) or in engineering units (EngUnts (1)).

Prop_Band

Prop_Band is the proportional band of the PID control algorithm.

Prior to Version 3.00, the proportional band could only be defined as a percentage of span. The engineering units mode was not supported.

If **PropUnits** is set to a percentage of span (PctSpan (0)), the proportional band is defined as the percent of the total span of the controller for which a control error will produce an output signal equivalent to the maximum output of the instrument. For example, if **Span_High** is 1000, **Span_Low** is 0, **Setpoint** is 500, **Process_Val** is 490 and **Prop_Band** is 1%, a proportional only reverse acting controller will produce an output of +100%, because the control error will be 10 units, which is one proportional band.

Note:- When defined as a percentage of span, the the proportional gain is given by:

$$\text{Gain} = \frac{10{,}000}{(\text{Span\_High} - \text{Span\_Low})\ {*}\text{Prop\_Band}}$$

If **PropUnits** is set to engineering units (EngUnts (1)), the proportional band is defined as the magnitude of the control error which will produce an output signal equivalent to the maximum output of the instrument. For example, if **Setpoint** is 500°C, **Process_Val** is 495°C and **Prop_Band** is 5°C, a proportional only reverse acting controller will produce an output of +100%, because the control error is 5°C which is equal to one proportional band.

When defined in Engineering units, the proportional gain is given by:

$$\text{Gain} = \frac{100}{\text{Prop\_Band}}$$

Integral

Integral is the integral time constant of the PID control algorithm. **Integral** time is defined as the time period in which the part of the output signal due to integral action increases by an amount equal to the part of the output signal due to proportional action, for a constant error state.

Derivative

Derivative is the derivative time constant of the PID control algorithm. Derivative time is defined as the time interval in which the part of the output signal due to derivative action increases by an amount equal to the part of the output signal due to proportional action, when the control error is changing at a constant rate.

Feed_Forward

Feed_Forward is added directly to the output of the PID algorithm, before the output limiting and dual output conversions are performed.

Manual_Reset

Manual_Reset is only active when Integral is set to zero. It provides an offset to the **Output,** which can be used to reduce the control error to zero when integral action is not employed.

Cutback_High and Cutback_Low

Cutback can be employed to reduce the amount of time it takes the **Process_Val** to respond to large changes in Setpoint and to limit the overshoot that can occur during the transient period. **Cutback_High** operates when the **Process_Val** is initially greater than the target Setpoint and **Cutback_Low** operates when the **Process_Val** is initially less than the target Setpoint. The units of Cutback_High and Cutback_Low are engineering units. Cutback operates by forcing the Output to its appropriate maximum or minimum limit in response to a Setpoint change which is greater than the cutback band. As the **Process_Val** approaches the Setpoint, the control error (**Setpoint - Process_Val)** reduces to less than the cutback value and normal control is resumed.

Integral_Hold

Integral_Hold enables the integral output to be frozen at its current value. It will remain constant throughout the period that Integral_Hold is enabled.

The VP_Auto tuning algorithms

The Autotune Algorithm

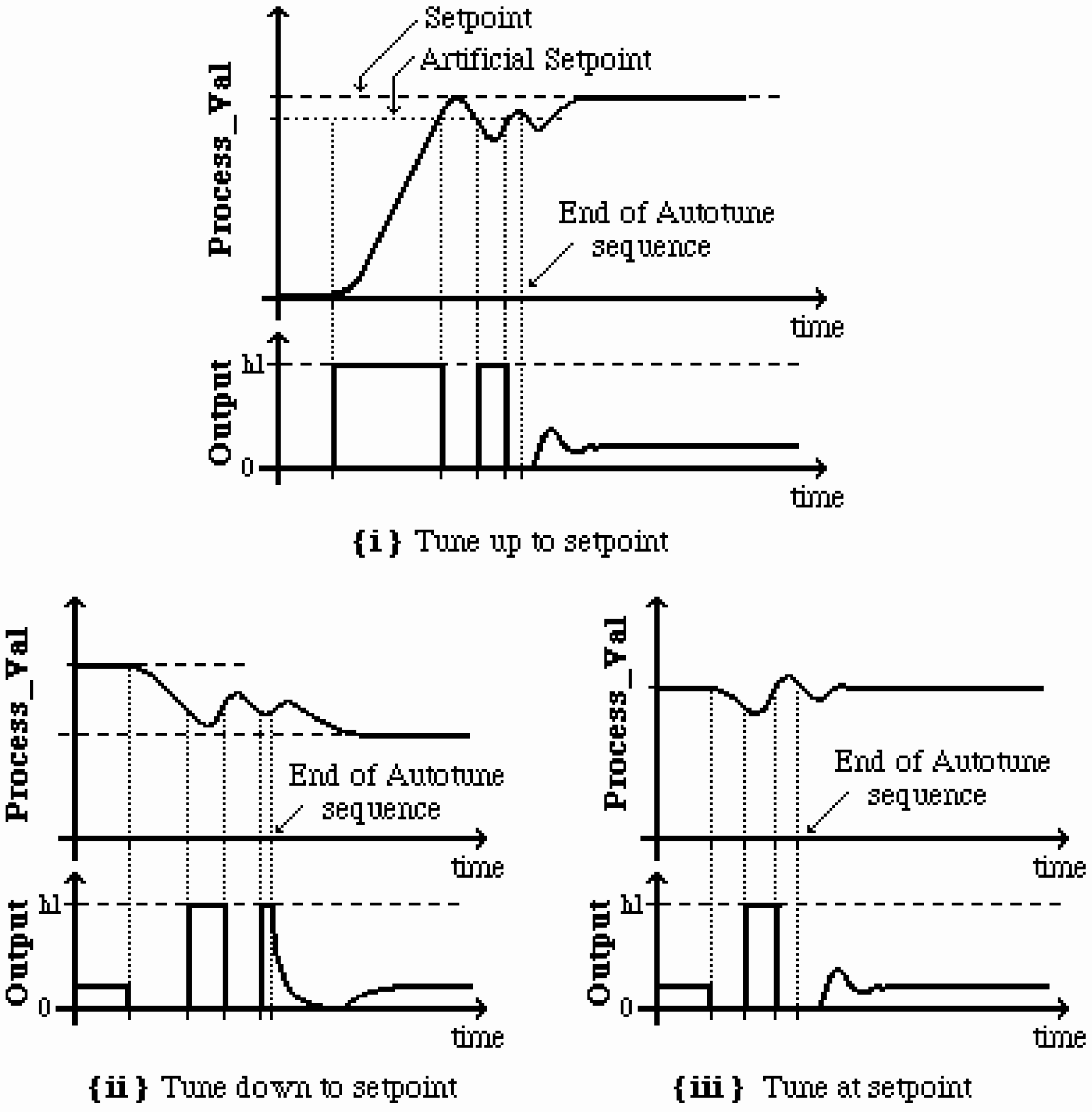


Figure 9-55 Heat only autotune cycles.

The autotune algorithm is a one shot tuner which was previously implemented in the Eurotherm 818 and 815 instruments. The principles of operation of this tuner are described in detail in The PC3000 Control Overview earlier in this chapter.

When the autotune algorithm is selected the PID control algorithm is disconnected from the process, with the autotuner driving the controller's outputs directly using ON/OFF control. On selection of autotune, if the current process value is within 1% of setpoint, the outputs are frozen at their existing values, otherwise the outputs are set to zero. This output value is held for one minute, during which the autotuner monitors the process to asses the noise level and the action of the process value. During this first minute the operator can select the setpoint at which the tuning is to be performed. This can be the current setpoint, if required, or a setpoint greater than or less than the current value.

When the one minute monitoring cycle has elapsed, the autotuner cycle will move into its active phase. If the current setpoint is more than 1% of span greater than the current process value, a tune up to setpoint will be performed. If the setpoint is more than 1% of span less than the current process value, a tune down to setpoint will be performed. If the current setpoint is within 1% of span of the process value, a tune at setpoint will be performed. These are described in detail in the references quoted above. The process value and output curves for the three types of autotune cycles, for a heat only instrument, are shown in the figure below.

At the end of the autotune cycle, the following parameters are calculated and loaded into the controller:

Proportional Band	
Integral time	
Derivative time	
Cutback High	(tune down to setpoint only)
Cutback Low	(tune up to setpoint only)
Trigger Val	(used by DRA and LSAT)
MTC	(used by LSAT)
Q	(used by LSAT)

The Disturbance Response Analysis Algorithm (DRA)

The DRA is an adaptive tuner which was previously implemented in the Eurotherm 818 instrument. It is an elementary expert system, which acts by identifying patterns in the error (setpoint–process value) response and adjusting the controller parameters to compensate for responses which are slow or oscillatory. Unlike the autotuner, DRA does not drive the controller outputs directly.

DRA is triggered into monitoring the control response when the absolute error exceeds the Trigger Value, which can be set manually but is automatically set by the autotuner. Once triggered by a disturbance, such as a setpoint change or a load

disturbance, DRA monitors up to two cycles of oscillation of the error before deciding what, if any, modification of the control terms is required. The ratio of peaks of the error (the damping ratio) is the main criterion which is used to decide whether retuning is required. If the amplitude of the peaks is small, the tuner disables itself and will not retune the controller parameters. In the 818 implementation of DRA, if the algorithm determined that retuning was necessary, the Proportional Band, Integral time and Derivative time were adjusted. In the PID_Auto implementation, DRA also adjusts the LSAT referencing terms MTC and Q. MTC is adjusted in series with the Integral and Derivative times. Q is adjusted with the Proportional Band.

If the response is analysed to be oscillatory, the oscillation time is used to determine whether the gain, the integral and derivative times, or both the gain and the times require adjustment. The figure below shows how the tuner determines which adaption strategy to adopt, which is calculated according to the values of the existing PID parameters.

DRA adaptive tune strategy selection.

The Least Squares Adaptive Tuner algorithm (LSAT).

The LSAT algorithm is an adaptive tuner which, like the DRA algorithm, does not drive the controller outputs directly, but instead tunes the PID control coefficients. The LSAT should be considered to consist of two main parts:

{1} A process model identifier

{2} A controller designer.

The algorithm looks at the controller output and the process value. It continually feeds the controller output into an internal model of the process and compares the process value predicted by this model with the actual process value. The recursive least squares algorithm is used to adjust the parameters of the model so that the predicted process value matches the actual process value. In this way the model is continuously being fine tuned so that it matches the actual process being controlled. The model is then used by the controller designer to define and set the optimum PID parameters for the process being controlled. This employs a model–reference approach, in which the closed–loop performance of the system is optimised against that of the internal reference closed–loop model. A block diagram of the LSAT algorithm is shown in the figure below.

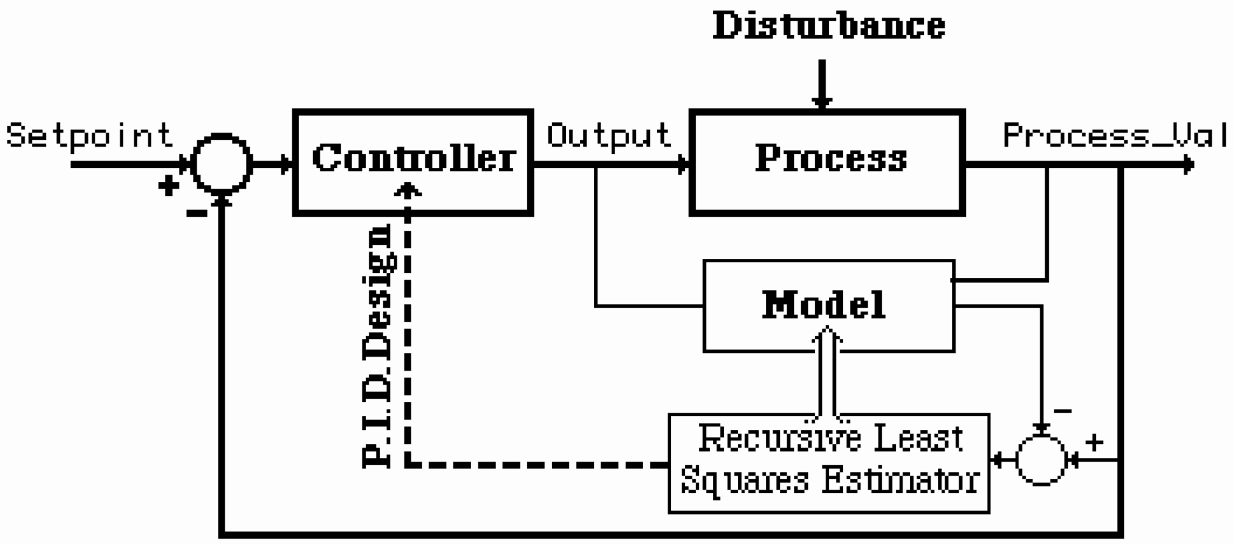


Figure 9-56 Block diagram of the Least Squares Adaptive Tuner.

An advantage of the LSAT is that, unlike the DRA, it does not require large process disturbances or step setpoint changes in order to act. It can fine tune the controller during regulatory control and during setpoint ramps, since it is able to filter elements of the process noise signal and utilise these in the model identification. The fine tuning action of the LSAT is an important feature when implemented in the PID_Auto adaptive tuner. It enables it both to complement the coarser tuning action of the DRA and to fine tune the parameters set by the Autotuner.

The procedure of model identification through prediction and design through model reference enables the LSAT to continuously tune the controller parameters during on–line control operations. However, the LSAT algorithm is not able to determine the structure of the process model, it fits parameters to a model whose structure is pre–defined. This restriction could limit its suitability for implementation as a stand–alone general purpose adaptive tuner, so the algorithm has been developed to adapt its identification and parameter design in response to external referencing. This takes the form of two parameters – "MTC", a process time constant scaler from which the LSAT determines its sample frequency and "Q", which gives an indication of the unmodelled (high frequency) process coefficients. Although this information can be input manually, it requires a high level of LSAT expertise and process knowledge, so the Autotune and DRA algorithms have been developed to identify MTC and Q and to prime the LSAT with these values. The resulting three tuner interaction forms the "Composite" tuner.

The Operation of the VP_Auto Tuners.

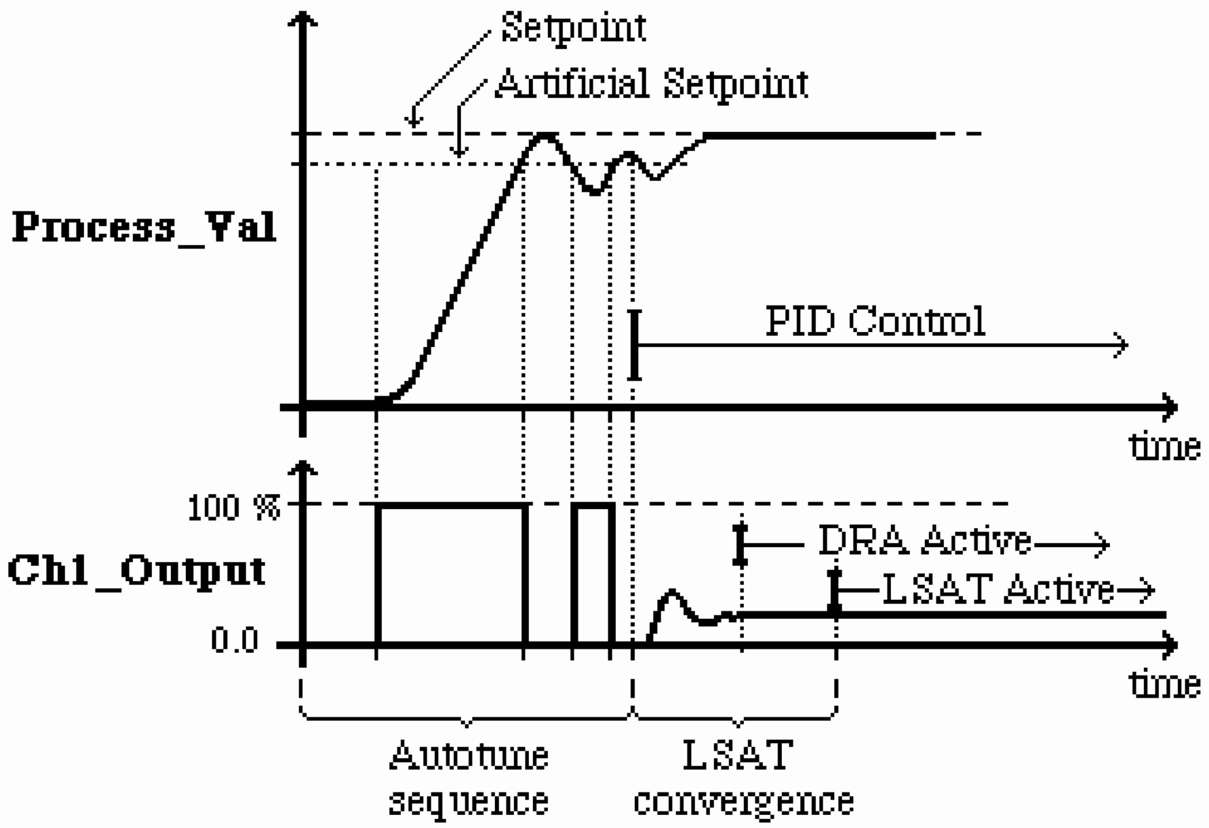


Figure 9-57 Autotune–Adaptive tune convergence sequence during a start up tune.

The Composite tuner combines the functionality of the three tuners described above, Autotune, DRA and LSAT. It has been developed to enable the strengths of the three algorithms to be utilised to provide the optimum tuning performance under the widest range of control conditions. The functions performed by the three tuning algorithms of the composite tuner may be summarised as:

{i} Autotune: initial one–shot tuning of PID parameters, coarse identification of process model, identification of process noise levels.

{ii} DRA: coarse adaption to changes in process operating conditions, coarse adjustment of LSAT reference model, adjustment of controller when necessary in response to large process disturbances.

{iii} LSAT: fine tuning of PID parameters after Autotune and during adaptive tune sequence, fine tuning during setpoint ramps and in response to small process changes or small disturbances.

Selecting the Autotuner will result in the one–shot tuner sequence being activated. Selecting Adaptive tune will activate both the DRA and the LSAT adaptive tuning algorithms together. It should be noted that **the optimum tuning performance will be obtained when AUTO/TUNE precedes ADAPT/TUNE.** The Autotune algorithm has been enhanced to provide the initialisation that LSAT requires. If Autotune does not precede Adaptive, then DRA is required to initialise LSAT, which can require several DRA adaption sequences to take place and can result in a very slow tuning of the controller.

When Autotune followed by Adaptive tune is selected, an Autotune sequence is performed followed by an automatic switch to Adaptive tune. The Adaptive Tune sequence then begins by disabling the LSAT until the estimator has converged on the process model. Once the LSAT has converged, the Adaptive Tune sequence then continues with both the DRA and the LSAT algorithms tuning the controller, under the guidance of the tuner overseer software.

Function Block Tuning Parameters

Tune_Type

Tune_Type defines which tuner, if any, is active. It can be set to one of eight values:

None (0):	No tuners active.
AT (1):	Autotuner active.
DRA (2):	Adaptive tune (DRA and) active.
LSAT (3):	Adaptive tune (DRA and LSAT) active.
DRA_LS (4):	Adaptive tune (DRA and LSAT) active.
AT_DRA (5):	Autotuner followed by Adaptive tuner (DRA) active.
AT_LSAT (6):	Autotuner followed by Adaptive tuner (DRA and LSAT) active.
AT_D_L (7):	Autotuner followed by Adaptive tuner (DRA and LSAT) active.

Param_Change

Param_Change is set by the tuners to indicate that the value of one of the parameters has been changed. Once it has been changed by a tuner to Yes (1), it must be reset externally to No (0).

Trigger_Val

Provides an indication to the Adaptive tuner of the amount of process noise which is present on the Process_Val. It is automatically set by the Autotuner during the tuning sequence, but can be set manually by the user.

Tuning

This provides an indication that the tuners are functioning. If Auto or Adaptive tuners are active, Tuning will be Active (1), else Tuning will be Off (0)

Function Block Diagnostic Parameters

MTC

Is the LSAT reference model time constant.

Q

This is the LSAT reference detuning factor.

CS_Pre_Limit

This is the output of the PID controller after the Feed_Forward has been added, but before Output limiting has been performed.

Control_Sig

Control_Sig is the sum of the proportional, integral, derivative and feed forward components after being high, low and rate of change limited, but before the dual output relative gain and deadband have been added.

Nerror, Integral_Out and Deriv_Out

These are the outputs of the proportional integral and derivative components respectively.

Feedback

Feedback is the value of the signal which is fed back into the PID to indicate the actual signal that is output by the controller.

Error

The difference between the Process_Val and the Setpoint.

VP_Model

VP_Model is the output of the internal model of the valve

AT_State

Indicates the state of the Autotuner.

0	Reset
1	Initiatisation
2	Monitor quiescent noise
3	End of monitor noise
4	Startup with a new setpoint
5	End of startup with new setpoint
6	Startup with PV at setpoint
7	End of startup with PV at setpoint
8	Zeigler-Nichols sequence
9	Calulate new parameters
10	Write update status
11	Autotune aborted
12	Autotune completed

Table 9-18 AT_State Values

ZN_Stage

ZN_Stage indicates the state of the Autotuner's Ziegler-Nichols tuning stages.

3	Find peak PV & reverse output
4	Find PV crossing PV1 & test again for dominant delay
5	Find peak PV & either reverse PV change or reverse output
6	Find PV crossing PV1 & adjust trend and output
7	Find peak PV & reverse output
8	Find PV crossing PV1 & calculate new parameters

Table 9-19 Zn_State Values

DRA_State

DRA_State indicates the state of the DRA adaptive tuner.

0	Allow settling
1	Wait for trigger
2	Find peak 1
3	Find zero 1
4	Find peak 2
5	Find zero 2
6	Find peak 3
7	Find zero 3
8	Find peak 4
9	Find zero 4
10	Find peak 5
11	End on zero 4 abort
12	End on peak 4 found
13	End on peak 5 abort
14	End on peak 5 found
15	Prepare update

Table 9-20 DRA_State Value

DRA_Last

Records the last tuning strategy performed by the DRA adaptive tuner.

0	No change
1	Reduce Damping
2	Increase gain
3	Decreased Times
4	Increased Times
5	Decreased Gain

Table 9-21 DRA_Last Value

LSAT_F1

Indicates the form of the process model which has been identified by the LSAT.

Proc_Delay

This is the delay time of the open-loop system which has been estimated by the Autotuner.

Nerror

Nerror is the output of the proportional component of the function block.

Status

Provides an indication of the function of the PID_Auto function block. it can have eight possible states:

OK (0):	The function block is operating normally
SnsrBrk (1):	An external sensor break has been detected
PV_High (2):	The Process_Val is greater than Span_High +10%
PV_Low (3):	The Process_Val is less than Span_Low -10%

SP_High (4):	The Setpoint is greater than Span_High
SP_Low (5):	The Setpoint is less than Span_Low
GainNeg (6):	The Prop_Band has been set with a negative value, or Span_High has been set less than Span_Low
GainHi (7):	The Prop_Band has been set with too small a value or the span of the function block has been set too small, resulting in a very large gain.

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
AT_State	**SINT**	0	Config		High Limit Low Limit	255 0
Break_Output	**ENUM**	Lower (1)	Oper	Oper	Senses	Off (0) Lower (1) Raise (2)
Control_Sig	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
CS_Pre_Limit	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Cutback_High	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Cutback_Low	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Debump	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)
Debump_Dis	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Deriv_On_PV	**BOOL**	On_Err (0)	Config	Config	Senses	On_Err (0) On_PV (1)
Deriv_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Derivative	**TIME**	50s	Oper	Oper	High Limit Low Limit	01d_03h 0
Direct	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
DRA_Last	**SINT**	0	Config		High Limit Low Limit	255 0
DRA_State	**SINT**	0	Config		High Limit Low Limit	255 0

Table 9-22 VP_Auto Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Error	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Feed_Forward	**REAL**	0	Super	Super	High Limit Low Limit	100 -100
Feedback	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Integral	**TIME**	5m	Oper	Oper	High Limit Low Limit	01d_03h 0
Integral_Hold	**BOOL**	No (0)	Config	Config	Senses	No (0) Yes (1)
Integral_Out	**REAL**	0	Config		High Limit Low Limit	100,000 -100,000
Lower	**BOOL**	Off (0)	Oper		Senses	Off (0) Lower (1)
LSAT_F1	**REAL**	0	Config		High Limit Low Limit	100,000 0
Manual	**BOOL**	Manual (1)	Oper	Oper	Senses	Auto (0) Manual (1)
Manual_Reset	**REAL**	0	Oper	Oper	High Limit Low Limit	100 -100
Min_On_Time	**TIME**	100ms	Oper	Oper	High Limit Low Limit	5s 100ms
MTC	**REAL**	30	Config	Config	High Limit Low Limit	100,000 0.1
Nerror	**REAL**	0	Super		High Limit Low Limit	100,000 -100,000
Output	**ENUM**	Off (0)	Oper	Oper	Senses	Off (0) Lower (1) Raise (2)
Param_Change	**BOOL**	No (0)	Super	Config	Senses	No (0) Yes (1)

Table 9-22 VP_Auto Parameter Attributes (continued)

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Pot_Break	**BOOL**	No (0)	Oper	Super	Senses	No (0) Yes (1)
Pot_Enable	**BOOL**	No (0)	Oper	Super	Senses	No (0) Yes (1)
Pot_Limit_Hi	**REAL**	100	Oper	Super	High Limit Low Limit	100 0
Pot_Limit_Lo	**REAL**	0	Oper	Super	High Limit Low Limit	100 0
Pot_Position	**REAL**	0	Oper	Super	High Limit Low Limit	100 0
Proc_Delay	**TIME**	0	Config		High Limit Low Limit	1d 0
Process_Val	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Prop_Band	**REAL**	5 %	Oper	Oper	High Limit Low Limit	10,000 0.1
PropUnits	**BOOL**	PctSpan (0)	Oper	Oper	Senses	PctSpan (0) EngUnts (1)
Q	**REAL**	0.4	Config	Config	High Limit Low Limit	100,000 0
Raise	**BOOL**	Off (0)	Oper		Senses	Off (0) Raise (1)
Sensor_Break	**BOOL**	Ok (0)	Oper	Super	Senses	Ok (0) Break (1)
Setpoint	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Span_High	**REAL**	100	Config	Config	High Limit Low Limit	100,000 Span_Low
Span_Low	**REAL**	0	Config	Config	High Limit Low Limit	Span_High -100,000

Table 9-22 VP_Auto Parameter Attributes (continued)

Name	**Type**	**Cold Start**	**Read Access**	**Write Access**	**Type Specific Information**	
Status	**ENUM**	Ok (0)	Oper		Senses	Ok (0) SnsrBrk (1) PV_High (2) PV_Low (3) SP_High (4) SP_Low (5) GainNeg (6) GainHi (7)
Travel_Time	**TIME**	20s	Oper	Oper	High Limit Low Limit	16m_40s 5s
Trigger_Val	**REAL**	0.1	Config	Config	High Limit Low Limit	100,000 0
Tune_Type	**ENUM**	None (0)	Config	Config	Senses	None (0) AT (1) DRA (2) LSAT (3) DRA_LS (4) AT_DRA (5) AT_LSAT (6) AT_D_L (7)
Tuning	**BOOL**	Off (0)	Super		Senses	Off (0) Active (1)
Update_Time	**TIME**	1s	Oper	Oper	High Limit Low Limit	20s 100ms
VP_Model	**REAL**	50	Config		High Limit Low Limit	100 0
Zn_Stage	**SINT**	0	Config		High Limit Low Limit	255 0

Table 9-22 VP_Auto Parameter Attributes (continued)

PIDHEATCOOL FUNCTION BLOCK

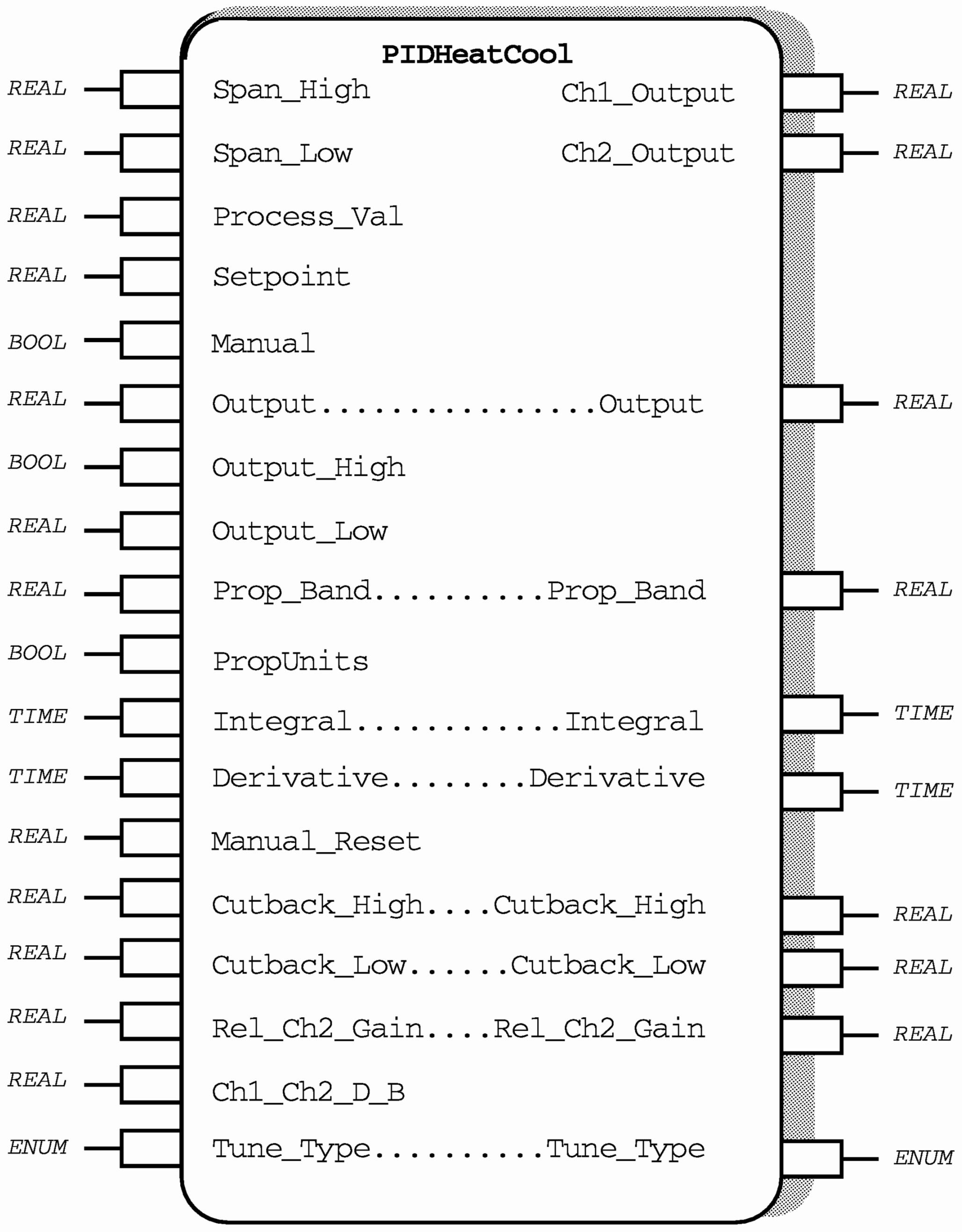


Figure 9-58 PIDHeatCool Function Block Diagram

Functional Description

The PIDHeatCool function block implements the proportional plus integral plus derivative control algorithm and the Auto and Adaptive tuning algorithms that are also used in the Eurotherm 900 series control instruments.and in the other PC3000 Control function blocks (PID, VP, PID_Auto and VP_Auto).

The PIDHeatCool function block is a cut down version of the PID_Auto function block. The interface parameters which are not commonly used have been removed to provide a simpler function block suitable for use in a wide range of applications. Those interface parameters which have been removed are set internally to their default values. For a full description of these parameters, reference should be made to the PID_Auto function block description.

Function Block Attributes.

Type:...................................	20 51
Class:	CONTROL
Default Task:	Task_2
Short List:	Setpoint, Process_Val, Manual, Output
Memory Requirements:	2056

Parameter Descriptions

Span_High and Span_Low.

Span_High and **Span_Low** define the maximum and minimum limits of the working range of the function block. Generally these are set to values which represent physical boundaries in the operation of the process, such as the calibrated range of a transducer, or the safe limits of a pressure vessel. The proportional band of the PID algorithm can be defined as a percentage of the span of the function block, which is found by subtracting **Span_Low** from **Span_High.**

Setpoint and Process_Val

The **Process_Val** is the controlled variable of the function block and the **Setpoint** is the target value against which the **Process_Val** is controlled. The PID algorithm acts to reduce the difference between the **Setpoint** and the **Process_Val** to zero.

Manual

Manual is used to select whether the function block operates in Manual or Auto mode. In Manual mode, the PID algorithm is disabled and the value of **Output** is taken directly from its input. In Auto mode the full controller functionality is active.

Output, Ch1_Output and Ch2_Output

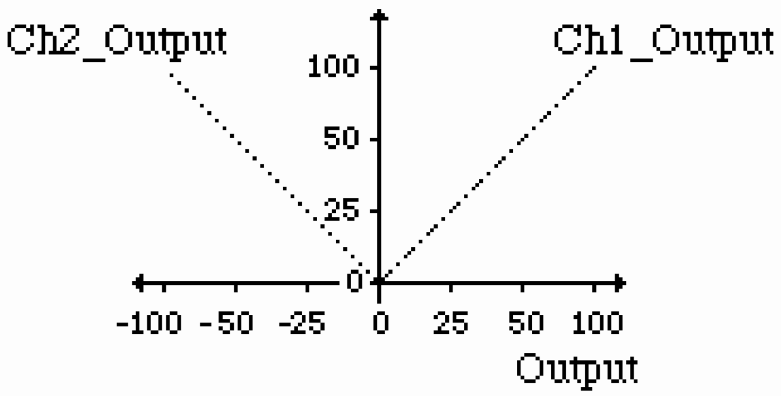


Figure 9-59 Dual Output Relationship, with Unity Rel_Ch2_Gain and zero Ch1_Ch2_D_B

The PID function block channel can be configured for either single or dual channel operation. In single channel operation, the control is on-**Output**, which is limited within the range + 100 % to 0 % by the parameters **Output_High** and **Output_Low**. Dual channel operation is designed for systems such as heat-cool applications, in which negative values of **Output** must be output to a refrigeration unit as absolute values to increase the refrigeration rate. For dual channel operation **Output_Low** is set to - 100%.

Output_High and Output_Low

Output_High and **Output_Low** define the upper and lower limits of **Output** They must both be set within the range -100% to +100%, with **Output_High** being greater than or equal to **Output_Low.** When the function block is being used for dual output operation, **Ch1_Output** and **Ch2_Output** are related to **Output** as shown in Figure 9-59, with **Output** being bounded by **Output_High** and **Output_Low**.

Rel_Ch2_Gain and Ch1_Ch2_D_B

When the function block is being used for dual channel control, the full relationship between **Ch2_Output** and **Output** is given by:

$$\text{Ch2\_Output} = (\text{Output} + \text{Ch1\_Ch2\_D\_B}) * \text{Rel\_Ch2\_Gain}$$

Rel_Ch2_Gain is intended for use in non linear dual output control situations, such as heat / cool systems, to compensate for the differing gains of the equipment being driven by the two output channels. **Ch1_Ch2_D_B** introduces a deadband between the two output channels, which can either be set to a positive value to provide a region in which neither channel is active, or to a negative value to provide a region of overlap in which both outputs are active.

Manual_Reset

Manual_Reset is only active when **Integral** is set to zero. It provides an offset to the **Output,** which can be used to reduce the control error to zero when integral action is not employed.

Cutback_High and Cutback_Low

Cutback can be employed to reduce the amount of time it takes the **Process_Val** to respond to large changes in **Setpoint** and to limit the overshoot that can occur during the transient period. **Cutback_High** operates when the **Process_Val** is initially greater than the target **Setpoint** and **Cutback_Low** operates when the **Process_Val** is initially less than the target **Setpoint.** The units of **Cutback_High** and **Cutback_Low** are engineering units. **Cutback** operates by forcing the **Output** to its appropriate maximum or minimum limit in response to a **Setpoint** change which is greater than the cutback band. As the **Process_Val** approaches the **Setpoint**, the control error (**Setpoint - Process_Val)** reduces to less than the cutback value and normal control is resumed.

PropUnits

PropUnits is to define whether the **Prop_Band** is defined as a percentage of the controller span (PctSpan (0)) or in engineering units (EngUnts (1)).

Prop_Band

Prop_Band is the proportional band of the PID control algorithm. If **PropUnits** is set to a percentage of span (PctSpan (0)), the proportional band is defined as the percent of the total span of the controller for which a control error will produce an output signal equivalent to the maximum output of the instrument. For example, if **Span_High** is 1000, **Span_Low** is 0, Setpoint is 500, **Process_Val i**s 490 and

Prop_Band is 1%, a proportional only reverse acting controller will produce an output of +100%, because the control error will be 10 units, which is one proportional band.

Note:- When defined as a percentage of span, the the proportional gain is given by:

$$\text{Gain} = \frac{10{,}000}{(\text{Span\_High} - \text{Span\_Low}) * \text{Prop\_Band}}$$

If **PropUnits** is set to engineering units (EngUnts (1)), the proportional band is defined as the magnitude of the control error which will produce an output signal equivalent to the maximum output of the instrument. For example, if **Setpoint** is 500°C, **Process_Val** is 495°C and **Prop_Band** is 5°C, a proportional only reverse acting controller will produce an output of +100%, because the control error is 5°C which is equal to one proportional band.

When defined in Engineering units, the proportional gain is given by:

$$\text{Gain} = \frac{100}{\text{Prop\_Band}}$$

Integral

Integral is the integral time constant of the PID control algorithm. **Integral** time is defined as the time period in which the part of the output signal due to integral action increases by an amount equal to the part of the output signal due to proportional action, for a constant error state.

Derivative

Derivative is the derivative time constant of the PID control algorithm. **Derivative** time is defined as the time interval in which the part of the output signal due to derivative action increases by an amount equal to the part of the output signal due to proportional action, when the control error is changing at a constant rate.

Tune_Type

Tune_Type is used to control the various auto and adaptive tuning algorithms available within the function block.

For a full description of these algorithms, reference should be made to the **PID_Auto** function block.

Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Ch1_Ch2_D_B	**REAL**	0	Oper	Oper	High Limit Low Limit	10 -10
Ch1_Output	**REAL**	0	Oper		High Limit Low Limit	100 The higher of 0 or Output_Low
Ch2_Output	**REAL**	0	Oper		High Limit Low Limit	The lower of 0 or Output_High -100
Cutback_High	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Cutback_Low	**REAL**	0	Super	Super	High Limit Low Limit	Span_High 0
Derivative	**TIME**	50s	Oper	Oper	High Limit Low Limit	01d_03h 0
Integral	**TIME**	5m	Oper	Oper	High Limit Low Limit	01d_03h 0
Manual	**BOOL**	Manual (1)	Oper	Oper	Senses	Auto (0) Manual (1)
Manual_Reset	**REAL**	0	Oper	Oper	High Limit Low Limit	100 -100
Output	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High Output_Low
Output_High	**REAL**	100	Oper	Oper	High Limit Low Limit	100 Output_Low
Output_Low	**REAL**	0	Oper	Oper	High Limit Low Limit	Output_High -100
Process_Val	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Prop_Band	**REAL**	5 %	Oper	Oper	High Limit Low Limit	10,000 0.1
PropUnits	**BOOL**	PctSpan(0)	Oper	Oper	Senses	PctSpan (0) EngUnits (1)

Table 9-23 PIDHeatCool Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Rel_Ch2_Gain	**REAL**	1	Oper	Oper	High Limit Low Limit	10 0.1
Setpoint	**REAL**	0	Oper	Oper	High Limit Low Limit	Span_High Span_Low
Span_High	**REAL**	100	Config	Config	High Limit Low Limit	100,000 Span_Low
Span_Low	**REAL**	0	Config	Config	High Limit Low Limit	Span_High - 100,000
Tune_Type	**ENUM**	None (0)	Config	Config	Senses	None (0) AT (1) DRA (2) LSAT (3) DRA_LS (4) AT_DRA (5) AT_LSAT (6) AT_D_L (7)

Table 9-23 PIDHeatCool Parameter Attributes (Continued)

APPENDIX A - PID SETTINGS

Tables for refinements of Ziegler and Nichols

This appendix deals with the various proposed refinements available for tuning PIDs using the open loop reaction curve test and the closed loop ultimate sensitivity method. Cohen and Coon refinements for PID settings from the reaction curve method are given in table 1.

Controller Type	Prop_Band	Integral	Derivative
P	100KpNd/(1 + 0.35Nd)		
PI	100KpNd/(0.9 + 0.083Nd)	(3.3+0.31Nd)Dp/(1+2 .2Nd	
PID	100KpNd/(0.25Nd+1.35)	(2.5+0.46Nd)Dp/(1+0 .61Nd)	0.37Dp (1+0.19Nd)
PD	100KpNd/(0.16Nd+1.24)		(0.27-0.088Nd)Dp/(1+0.13Nd)

Table 1 Cohen and Coon PID settings from a reaction curve

Shinskey [8] and Hang, Astrom and Ho [5] point out that for reasonably secure manual tuning both the reaction curve method and the ultimate sensitivity method have to be performed to obtain the best result. Briefly, there appears to be an empirical relationship between the ratio of the open loop dead-time to time constant Nd = Dp/Tp and the ratio of proportional band setting during ultimate sensitivity test to the open loop process gain K = 100Kp/Pu. It is given by

$$K = \frac{1(11N_d + 13)}{(37N_d \_ 4)}$$

If K is larger than 1.5 the results of PI and PID control as per Ziegler and Nichols setting can be poor. Three situations arise

If K < 1.5 retain the PID values;

If 1.5 < K < 2.25, adjust integral term Ti=2KTu/9 with a setpoint weight of 8(4K/9 + 1)/17;

If 2.25 < K, retain PID values and set the setpoint weight to 36/(27+5K).

For PI control if 1.2 < K < 15, PB to 6Pu(14K+15)/5(12+K) and Ti to Tu(4K+15)/75. Setpoint weighting as that of the PID can be used if needed.

Shinskey [8] has a very similar view. He looks at the ratio of the ultimate period Tu to the estimated dead-time Tu/Dp. Four situations arise:

Tu/Dp :	The process is pure dead-time;
2 < Tu/Dp < 4 :	The process is dead-time dominant;

Tu/Dp=4 :	There is single dominant capacity (first order system plus small dead-time;
Tu/Dp > 4 :	More than one capacity (lag) present.

Table 2 gives the refinements of Ziegler and Nichols rules for some of these cases.

If the process is open-loop unstable then tuning can only be realistically be performed manually. Table 3 gives recommendations of Shinskey for an open loop unstable system with a gain Kp, dead-time Dp and time constant Tp.

PI Control			**PID Control**		
	Tu/Dp	**Ti/Tu**	**PB/Pu**	**Ti/Tu**	**Td/Tu**
PB/Pu					
	2	0.25	2.35		No PID
	2.74	0.37	2.17	0.34	0.12
1.66					
	3.09	0.47	2.10	0.38	0.12
1.51					
	3.41	0.66	1.85	0.42	0.12
1.43					
	3.71	0.81	1.71	0.48	0.11
1.30					
	4.00	1.00	1.65	0.48	0.12
1.70					

Table 2 PI/PID settings from open and closed loop tests: Shinskey's refinements

Dp/Tp	**Ti/Dp**	**Td/Dp**	**PB**
-0.10	1.70	0.6	11Kp
-0.20	1.90	0.60	20Kp
-0.50	2.00	0.80	56Kp
-0.67	2.25	0.90	77Kp
-0.80	2.40	1.00	96Kp

Table 3 PID settings for first order UNSTABLE systems with delay

References

[1] K.J. Astrom and B. Wittenmark, 1984, Computer Controlled Systems: Theory and Design, Prentice-Hall International.

[2] K.J. Astrom and B. Wittenmark, 1989, Adaptive Control, Addison Wesley Publishing Company.

[3] P.J. Gawthrop, P.E. Nomikos and L.S.P.S. Smith, 1990, Adaptive Temperature

[4] T. Hugglund, 1991, Process Control in Practice, Chartwell Bratt Ltd.

[5] C.C. Hang, K.J. Astrom and W.K. Ho, 1991, Refinements of Ziegler-Nichols tuning formula, IEE Proceedings Part D, Vol 138, No. 2.

[6] J.M. Maceijowski, 1989, Multivariable Feedback Design, Adison-Wesley Publication Company.

[7] M.Morari and E. Zafiriou, 1989, Robust Process Control, Prentice Hall Internaional.

[8] F.G. Shinskey, 1988, Process Control Systems, McGraw-Hill Book Co.

[9] J.G. Ziegler and N.B. Nichols, 1942, Optimum settings for automatic controllers, Transctions ASME, Vol 65, pp. 433-444.