6

PID Control

6.1 Introduction

The PID controller is the most common form of feedback. It was an essential element of early governors and it became the standard tool when process control emerged in the 1940s. In process control today, more than 95% of the control loops are of PID type, most loops are actually PI control. PID controllers are today found in all areas where control is used. The controllers come in many different forms. There are stand-alone systems in boxes for one or a few loops, which are manufactured by the hundred thousands yearly. PID control is an important ingredient of a distributed control system. The controllers are also embedded in many special-purpose control systems. PID control is often combined with logic, sequential functions, selectors, and simple function blocks to build the complicated automation systems used for energy production, transportation, and manufacturing. Many sophisticated control strategies, such as model predictive control, are also organized hierarchically. PID control is used at the lowest level; the multivariable controller gives the setpoints to the controllers at the lower level. The PID controller can thus be said to be the "bread and butter" of control engineering. It is an important component in every control engineer's tool box.

PID controllers have survived many changes in technology, from mechanics and pneumatics to microprocessors via electronic tubes, transistors, integrated circuits. The microprocessor has had a dramatic influence on the PID controller. Practically all PID controllers made today are based on microprocessors. This has given opportunities to provide additional features like automatic tuning, gain scheduling, and continuous adaptation.

6.2 The Algorithm

We will start by summarizing the key features of the PID controller. The "textbook" version of the PID algorithm is described by:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \tag{6.1}$$

where y is the measured process variable, r the reference variable, u is the control signal and e is the control error ($e = y_{sp} - y$). The reference variable is often called the set point. The control signal is thus a sum of three terms: the P-term (which is proportional to the error), the I-term (which is proportional to the integral of the error), and the D-term (which is proportional to the derivative of the error). The controller parameters are proportional gain K, integral time T_i, and derivative time T_d. The integral, proportional and derivative part can be interpreted as control actions based on the past, the present and the future as is illustrated in Figure 2.2. The derivative part can also be interpreted as prediction by linear extrapolation as is illustrated in Figure 2.2. The action of the different terms can be illustrated by the following figures which show the response to step changes in the reference value in a typical case.

Effects of Proportional, Integral and Derivative Action

Proportional control is illustrated in Figure 6.1. The controller is given by (6.1) with $T_i = \infty$ and $T_d = 0$. The figure shows that there is always a steady state error in proportional control. The error will decrease with increasing gain, but the tendency towards oscillation will also increase.

Figure 6.2 illustrates the effects of adding integral. It follows from (6.1) that the strength of integral action increases with decreasing integral time T_i. The figure shows that the steady state error disappears when integral action is used. Compare with the discussion of the "magic of integral action" in Section Section 2.2. The tendency for oscillation also increases with decreasing T_i. The properties of derivative action are illustrated in Figure 6.3.

Figure 6.3 illustrates the effects of adding derivative action. The parameters K and T_i are chosen so that the closed-loop system is oscillatory. Damping increases with increasing derivative time, but decreases again when derivative time becomes too large. Recall that derivative action can be interpreted as providing prediction by linear extrapolation over the time T_d. Using this interpretation it is easy to understand that derivative action does not help if the prediction time T_d is too large. In Figure 6.3 the period of oscillation is about 6 s for the system without derivative

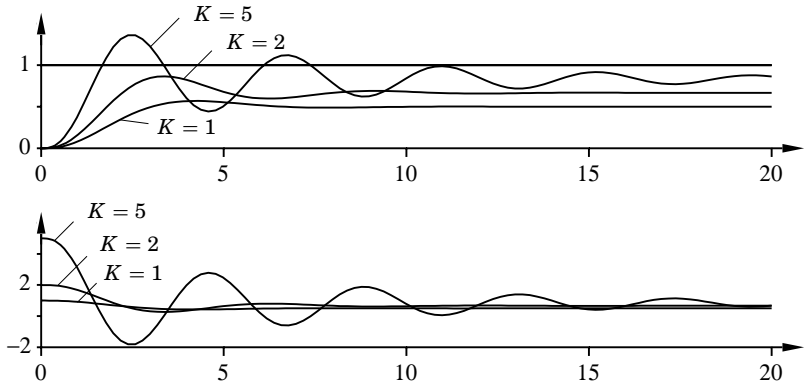


Figure 6.1 Simulation of a closed-loop system with proportional control. The process transfer function is $P(s) = 1/(s+1)^3$.

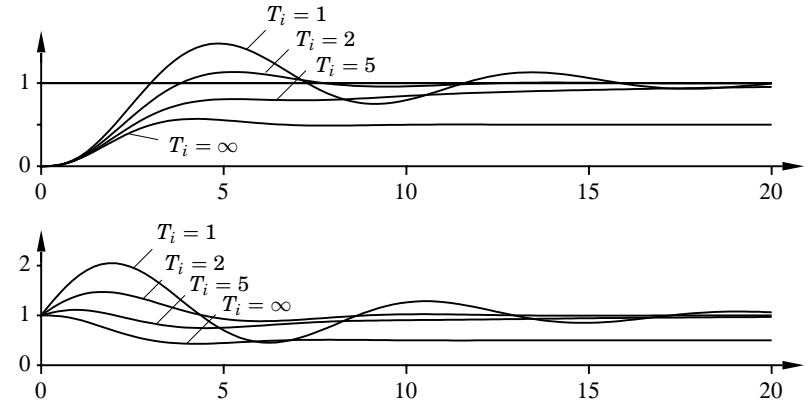


Figure 6.2 Simulation of a closed-loop system with proportional and integral control. The process transfer function is $P(s) = 1/(s+1)^3$, and the controller gain is $K = 1$.

action. Derivative actions ceases to be effective when T_d is larger than a 1 s (one sixth of the period). Also notice that the period of oscillation increases when derivative time is increased.

A Perspective

There is much more to PID than is revealed by (6.1). A faithful implementation of the equation will actually not result in a good controller. To obtain a good PID controller it is also necessary to consider

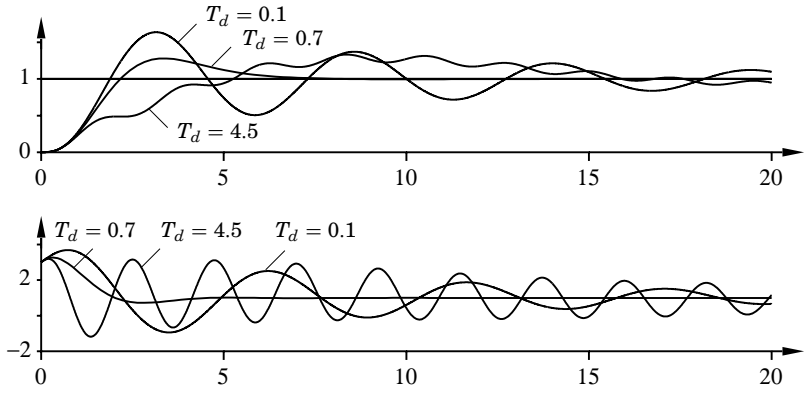


Figure 6.3 Simulation of a closed-loop system with proportional, integral and derivative control. The process transfer function is $P(s) = 1/(s+1)^3$, the controller gain is $K = 3$, and the integral time is $T_i = 2$.

- Noise filtering and high frequency roll-off
- Set point weighting and 2 DOF
- Windup
- Tuning
- Computer implementation

In the case of the PID controller these issues emerged organically as the technology developed but they are actually important in the implementation of all controllers. Many of these questions are closely related to fundamental properties of feedback, some of them have been discussed earlier in the book.

6.3 Filtering and Set Point Weighting

Filtering

Differentiation is always sensitive to noise. This is clearly seen from the transfer function $G(s) = s$ of a differentiator which goes to infinity for large s. The following example is also illuminating.

EXAMPLE 6.1—DIFFERENTIATION AMPLIFIES HIGH FREQUENCY NOISE
Consider the signal

$$y(t) = \sin t + n(t) = \sin t + a_n \sin \omega_n t$$

where the noise is sinusoidal noise with frequency ω. The derivative of the signal is

$$\frac{dy(t)}{dt} = \cos t + n(t) = \cos t + a_n \omega \cos \omega_n t$$

The signal to noise ratio for the original signal is $1/a_n$ but the signal to noise ratio of the differentiated signal is ω/a_n. This ratio can be arbitrarily high if ω is large. □

In a practical controller with derivative action it is therefor necessary to limit the high frequency gain of the derivative term. This can be done by implementing the derivative term as

$$D = -\frac{sKT_d}{1 + sT_d/N} Y \tag{6.2}$$

instead of $D = sT_dY$. The approximation given by (6.2) can be interpreted as the ideal derivative sT_d filtered by a first-order system with the time constant T_d/N. The approximation acts as a derivative for low-frequency signal components. The gain, however, is limited to KN. This means that high-frequency measurement noise is amplified at most by a factor KN. Typical values of N are 8 to 20.

Further limitation of the high-frequency gain

The transfer function from measurement y to controller output u of a PID controller with the approximate derivative is

$$C(s) = -K\left(1 + \frac{1}{sT_i} + \frac{sT_d}{1 + sT_d/N}\right)$$

This controller has constant gain

$$\lim_{s\to\infty} C(s) = -K(1 + N)$$

at high frequencies. It follows from the discussion on robustness against process variations in Section 5.5 that it is highly desirable to roll-off the

controller gain at high frequencies. This can be achieved by additional low pass filtering of the control signal by

$$F(s) = \frac{1}{(1 + sT_f)^n}$$

where T_f is the filter time constant and n is the order of the filter. The choice of T_f is a compromise between filtering capacity and performance. The value of T_f can be coupled to the controller time constants in the same way as for the derivative filter above. If the derivative time is used, $T_f = T_d/N$ is a suitable choice. If the controller is only PI, $T_f = T_i/N$ may be suitable.

The controller can also be implemented as

$$C(s) = -K\left(1 + \frac{1}{sT_i} + sT_d\right)\frac{1}{(1 + sT_d/N)^2} \tag{6.3}$$

This structure has the advantage that we can develop the design methods for an ideal PID controller and use an iterative design procedure. The controller is first designed for the process $P(s)$. The design gives the controller parameter T_d. An ideal controller for the process $P(s)/(1+sT_d/N)^2$ is then designed giving a new value of T_d etc. Such a procedure will also give a clear picture of the trade-off between performance and filtering.

Set Point Weighting

When using the control law given by (6.1) it follows that a step change in the reference signal will result in an impulse in the control signal. This is often highly undesirable therefor derivative action is frequently not applied to the reference signal. This problem can be avoided by filtering the reference value before feeding it to the controller. Another possibility is to let proportional action act only on part of the reference signal. This is called set point weighting. A PID controller given by (6.1) then becomes

$$u(t) = K\left(br(t) - y(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\left(c\frac{dr(t)}{dt} - \frac{dy(t)}{dt}\right)\right) \tag{6.4}$$

where b and c are additional parameter. The integral term must be based on error feedback to ensure the desired steady state. The controller given by (6.4) has a structure with two degrees of freedom because the signal path from y to u is different from that from r to u. The transfer function from r to u is

$$\frac{U(s)}{R(s)} = C_r(s) = K\left(b + \frac{1}{sT_i} + csT_d\right) \tag{6.5}$$

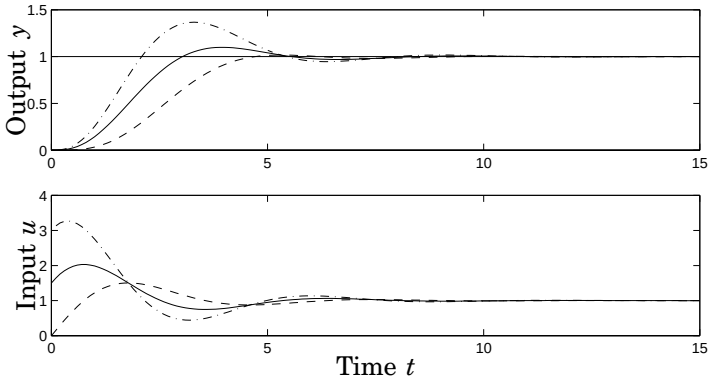


Figure 6.4 Response to a step in the reference for systems with different set point weights $b = 0$ dashed, $b = 0.5$ full and $b = 1.0$ dash dotted. The process has the transfer function $P(s) = 1/(s+1)^3$ and the controller parameters are $k = 3$, $k_i = 1.5$ and $k_d = 1.5$.

and the transfer function from y to u is

$$\frac{U(s)}{Y(s)} = C_y(s) = K\left(1 + \frac{1}{sT_i} + sT_d\right) \tag{6.6}$$

Set point weighting is thus a special case of controllers having two degrees of freedom.

The system obtained with the controller (6.4) respond to load disturbances and measurement noise in the same way as the controller (6.1) . The response to reference values can be modified by the parameters b and c. This is illustrated in Figure 6.4, which shows the response of a PID controller to setpoint changes, load disturbances, and measurement errors for different values of b. The figure shows clearly the effect of changing b. The overshoot for setpoint changes is smallest for $b = 0$, which is the case where the reference is only introduced in the integral term, and increases with increasing b.

The parameter c is normally zero to avoid large transients in the control signal due to sudden changes in the setpoint.

6.4 Different Parameterizations

The PID algorithm given by Equation (6.1) can be represented by the

transfer function

$$G(s) = K\left(1 + \frac{1}{sT_i} + sT_d\right) \tag{6.7}$$

A slightly different version is most common in commercial controllers. This controller is described by

$$G'(s) = K'\left(1 + \frac{1}{sT_i'}\right)(1 + sT_d') = K'\left(1 + \frac{T_d'}{T_i'} + \frac{1}{sT_i'} + sT_d'\right) \tag{6.8}$$

The controller given by Equation (6.7) is called non-interacting, and the one given by Equation (6.8) interacting. The interacting controller Equation (6.8) can always be represented as a non-interacting controller whose coefficients are given by

$$\begin{aligned} K &= K'\frac{T_i' + T_d'}{T_i'} \\ T_i &= T_i' + T_d' \\ T_d &= \frac{T_i'T_d'}{T_i' + T_d'} \end{aligned} \tag{6.9}$$

An interacting controller of the form Equation (6.8) that corresponds to a non-interacting controller can be found only if

$$T_i \geq 4T_d$$

The parameters are then given by

$$\begin{aligned} K' &= \frac{K}{2}\left(1 + \sqrt{1 - 4T_d/T_i}\right) \\ T_i' &= \frac{T_i}{2}\left(1 + \sqrt{1 - 4T_d/T_i}\right) \\ T_d' &= \frac{T_i}{2}\left(1 - \sqrt{1 - 4T_d/T_i}\right) \end{aligned} \tag{6.10}$$

The non-interacting controller given by Equation (6.7) is more general, and we will use that in the future. It is, however, sometimes claimed that the interacting controller is easier to tune manually.

It is important to keep in mind that different controllers may have different structures when working with PID controllers. If a controller is replaced by another type of controller, the controller parameters may have to be changed. The interacting and the non-interacting forms differ only

when both the I and the D parts of the controller are used. If we only use the controller as a P, PI, or PD controller, the two forms are equivalent. Yet another representation of the PID algorithm is given by

$$G''(s) = k + \frac{k_i}{s} + sk_d \tag{6.11}$$

The parameters are related to the parameters of standard form through

$$k = K \qquad k_i = \frac{K}{T_i} \qquad k_d = KT_d$$

The representation Equation (6.11) is equivalent to the standard form, but the parameter values are quite different. This may cause great difficulties for anyone who is not aware of the differences, particularly if parameter $1/k_i$ is called integral time and k_d derivative time. It is even more confusing if k_i is called integration time. The form given by Equation (6.11) is often useful in analytical calculations because the parameters appear linearly. The representation also has the advantage that it is possible to obtain pure proportional, integral, or derivative action by finite values of the parameters.

The PIPD Controller

The controller with set point weighting can also be represented by the block diagram in Figure 6.5. To see this we introduce the transfer functions of the blocks

$$G_{PI}(s) = k' + \frac{k_i'}{s}$$
$$G_{PD}(s) = 1 + k_d's$$

Notice that the proportional gain of the PD controller must be one in order to have zero steady state error. The input-output relation of the complete controller is

$$U(s) = k'R(s) + \frac{k_i'}{s}(R(s) - Y(s)) - (k' + k_d'h_i')Y(s) - k'k_d'sY(s)$$

Which shows that the controller is thus identical to the controller given

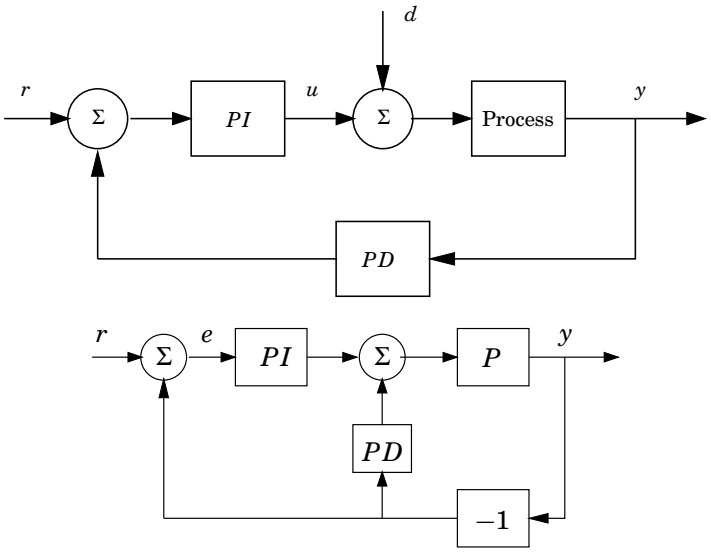


Figure 6.5 Block diagram of a PI-PD controller. This controller is equivalent to a conventional PID controller with set point weighting.

by (6.4). The parameters are related in the following way

$$
\begin{aligned}
k &= k' + k'_d k'_i \\
k_i &= k'_i \\
k_d &= k' k'_d \\
T_i &= \frac{k' + k'_d k'_i}{k'_i} = \frac{k'}{k'_i} \\
T_d &= \frac{k' k'_d}{k'_i} \\
b &= \frac{k'}{k' + k'_d h'_i} \\
c &= 0
\end{aligned}
$$

Following the same pattern the controller with $b = 0$ and $c = 0$ is sometimes called an I-PD controller, and the controller with $b = 1$ and $c = 0$ is called a PI-D controller.

Notice, however, that the representation given by (6.4) is much better suitable for tuning, because the parameters k, T_i and T_d can first be determined to deal with load disturbances, measurement noise and process

uncertainty. When this is done the response to set points can be adjusted by choosing the parameters b and c. The controller parameters appear in a much more complicated way in the PIPD controller.

6.5 Windup

Although many aspects of a control system can be understood based on linear theory, some nonlinear effects must be accounted for in practically all controllers. Windup is such a phenomena, which is caused by the interaction of integral action and saturations. All actuators have limitations: a motor has limited speed, a valve cannot be more than fully opened or fully closed, etc. For a control system with a wide range of operating conditions, it may happen that the control variable reaches the actuator limits. When this happens the feedback loop is broken and the system runs as an open loop because the actuator will remain at its limit independently of the process output. If a controller with integrating action is used, the error will continue to be integrated. This means that the integral term may become very large or, colloquially, it "winds up". It is then required that the error has opposite sign for a long period before things return to normal. The consequence is that any controller with integral action may give large transients when the actuator saturates. We will illustrate this by an example.

EXAMPLE 6.2—ILLUSTRATION OF INTEGRATOR WINDUP
The wind-up phenomenon is illustrated in Figure 6.6, which shows control of an integrating process with a PI controller. The initial setpoint change is so large that the actuator saturates at the high limit. The integral term increases initially because the error is positive; it reaches its largest value at time $t = 10$ when the error goes through zero. The output remains saturated at this point because of the large value of the integral term. It does not leave the saturation limit until the error has been negative for a sufficiently long time to let the integral part come down to a small level. Notice that the control signal bounces between its limits several times. The net effect is a large overshoot and a damped oscillation where the control signal flips from one extreme to the other as in relay oscillation. The output finally comes so close to the setpoint that the actuator does not saturate. The system then behaves linearly and settles. □

The example show integrator windup which is generated by a change in the reference value. Windup may also be caused by large disturbances or equipment malfunctions. It can also occur in many other situations.

The phenomenon of windup was well known to manufacturers of analog controllers who invented several tricks to avoid it. They were described

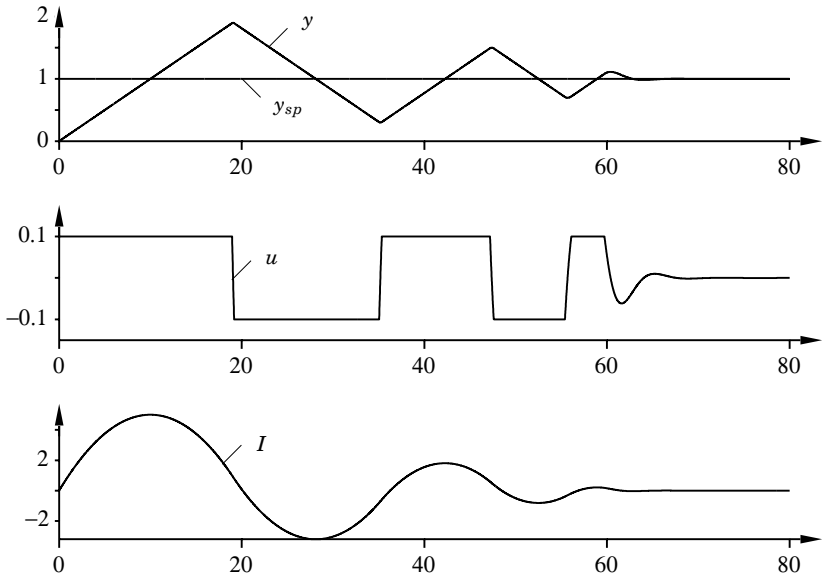


Figure 6.6 Illustration of integrator windup. The diagrams show process output y, setpoint y_{sp}, control signal u, and integral part I.

under labels like preloading, batch unit, etc. Although the problem was well understood, there were often restrictions caused by the analog technology. The ideas were often kept as trade secrets and not much spoken about. The problem of windup was rediscovered when controllers were implemented digitally and several methods to avoid windup were presented in the literature. In the following section we describe some of the methods used to avoid windup.

Setpoint Limitation

One attempt to avoid integrator windup is to introduce limiters on the setpoint variations so that the controller output never reaches the actuator limits. This frequently leads to conservative bounds and poor performance. Furthermore, it does not avoid windup caused by disturbances.

Incremental Algorithms

In the early phases of feedback control, integral action was integrated with the actuator by having a motor drive the valve directly. In this case windup is handled automatically because integration stops when the valve stops. When controllers were implemented by analog techniques,

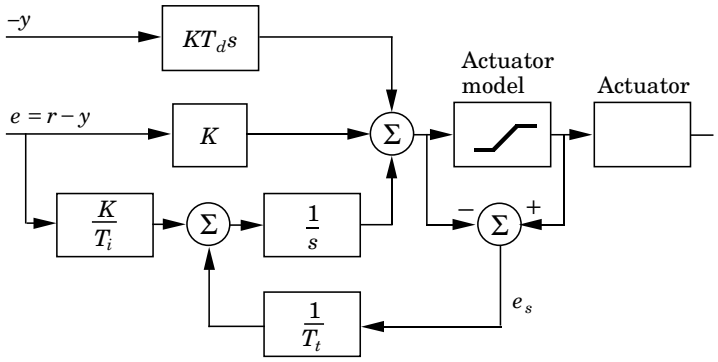


Figure 6.7 Controller with anti-windup where the actuator output is estimated from a mathematical model.

and later with computers, many manufacturers used a configuration that was an analog of the old mechanical design. This led to the so-called velocity algorithms. A velocity algorithm first computes the rate of change of the control signal which is then fed to an integrator. In some cases this integrator is a motor directly connected to the actuator. In other cases the integrator is implemented internally in the controller. With this approach it is easy to avoid windup by inhibiting integration whenever the output saturates. This method is equivalent to back-calculation, which is described below. If the actuator output is not measured, a model that computes the saturated output can be used. It is also easy to limit the rate of change of the control signal.

Back-Calculation and Tracking

Back-calculation works as follows: When the output saturates, the integral term in the controller is recomputed so that its new value gives an output at the saturation limit. It is advantageous not to reset the integrator instantaneously but dynamically with a time constant T_t.

Figure 6.7 shows a block diagram of a PID controller with anti-windup based on back-calculation. The system has an extra feedback path that is generated by measuring the actual actuator output and forming an error signal (e_s) as the difference between the output of the controller (v) and the actuator output (u). Signal e_s is fed to the input of the integrator through gain $1/T_t$. The signal is zero when there is no saturation. Thus, it will not have any effect on the normal operation when the actuator does not saturate. When the actuator saturates, the signal e_s is different from zero. The normal feedback path around the process is broken because the

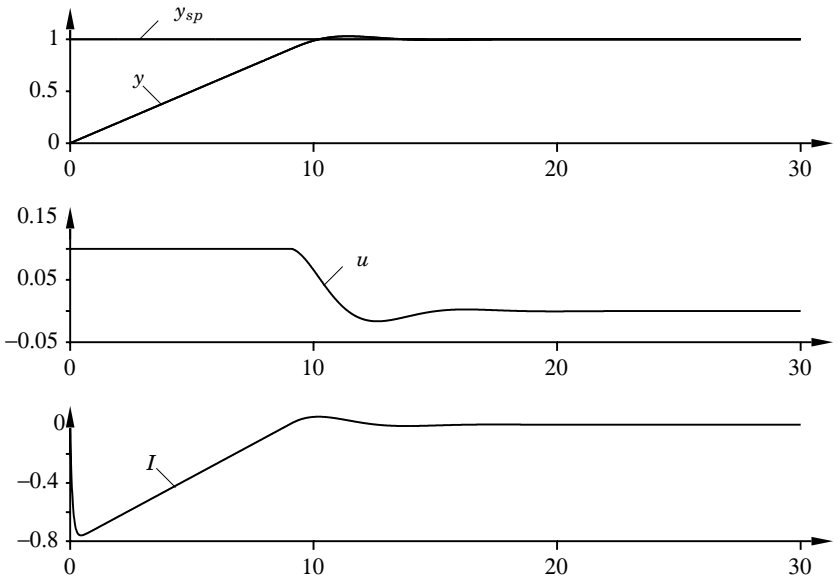


Figure 6.8 Controller with anti-windup applied to the system of Figure 6.6. The diagrams show process output y, setpoint y_{sp}, control signal u, and integral part I.

process input remains constant. There is, however, a feedback path around the integrator. Because of this, the integrator output is driven towards a value such that the integrator input becomes zero. The integrator input is

$$\frac{1}{T_t}\, e_s + \frac{K}{T_i}\, e$$

where e is the control error. Hence,

$$e_s = -\frac{K T_t}{T_i}\, e$$

in steady state. Since $e_s = u - v$, it follows that

$$v = u_{\lim} + \frac{K T_t}{T_i}\, e$$

where u_{lim} is the saturating value of the control variable. This means that the signal v settles on a value slightly out side the saturation limit and the control signal can react as soon as the error changes time. This prevents

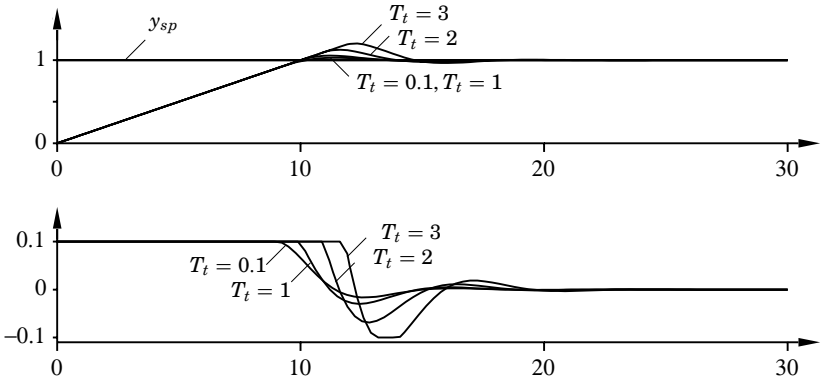


Figure 6.9 The step response of the system in Figure 6.6 for different values of the tracking time constant T_t. The upper curve shows process output y and setpoint y_{sp}, and the lower curve shows control signal u.

the integrator from winding up. The rate at which the controller output is reset is governed by the feedback gain, $1/T_t$, where T_t can be interpreted as the time constant, which determines how quickly the integral is reset. We call this the tracking time constant.

It frequently happens that the actuator output cannot be measured. The anti-windup scheme just described can be used by incorporating a mathematical model of the saturating actuator, as is illustrated in Figure 6.7.

Figure 6.8 shows what happens when a controller with anti-windup is applied to the system simulated in Figure 6.6. Notice that the output of the integrator is quickly reset to a value such that the controller output is at the saturation limit, and the integral has a negative value during the initial phase when the actuator is saturated. This behavior is drastically different from that in Figure 6.6, where the integral has a positive value during the initial transient. Also notice the drastic improvement in performance compared to the ordinary PI controller used in Figure 6.6.

The effect of changing the values of the tracking time constant is illustrated in Figure 6.9. From this figure, it may thus seem advantageous to always choose a very small value of the time constant because the integrator is then reset quickly. However, some care must be exercised when introducing anti-windup in systems with derivative action. If the time constant is chosen too small, spurious errors can cause saturation of the output, which accidentally resets the integrator. The tracking time constant T_t should be larger than T_d and smaller than T_i. A rule of thumb

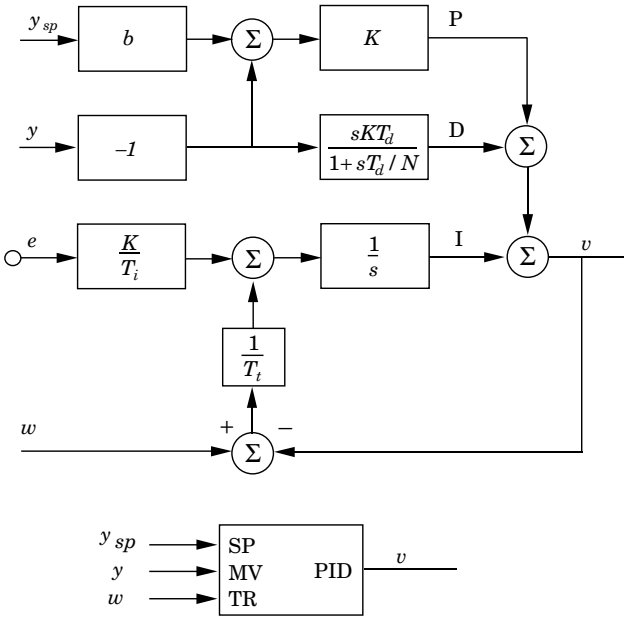


Figure 6.10 Block diagram and simplified representation of PID module with tracking signal.

that has been suggested is to choose $T_t = \sqrt{T_i T_d}$.

Controllers with a Tracking Mode

A controller with back-calculation can be interpreted as having two modes: the normal *control mode*, when it operates like an ordinary controller, and a *tracking mode*, when the controller is tracking so that it matches given inputs and outputs. Since a controller with tracking can operate in two modes, we may expect that it is necessary to have a logical signal for mode switching. However, this is not necessary, because tracking is automatically inhibited when the tracking signal w is equal to the controller output. This can be used with great advantage when building up complex systems with selectors and cascade control.

Figure 6.10 shows a PID module with a tracking signal. The module has three inputs: the setpoint, the measured output, and a tracking signal. The new input TR is called a tracking signal because the controller output will follow this signal. Notice that tracking is inhibited when $w = v$. Using the module the system shown in Figure 6.7 can be presented as shown in Figure 6.11.

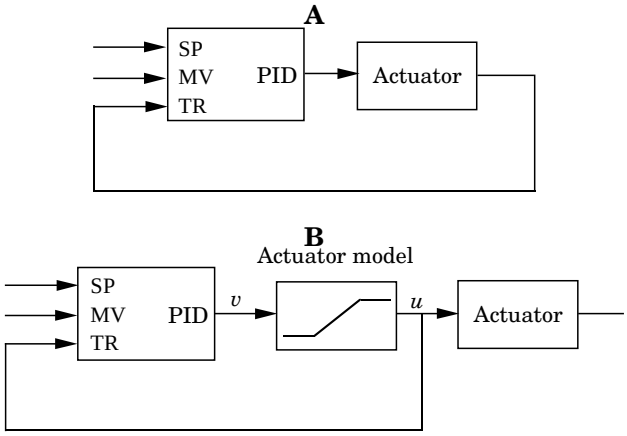


Figure 6.11 Representation of the controllers with anti-windup in Figure 6.7 using the basic control module with tracking shown in Figure 6.10.

6.6 Tuning

All general methods for control design can be applied to PID control. A number of special methods that are tailor-made for PID control have also been developed, these methods are often called tuning methods. Irrespective of the method used it is essential to always consider the key elements of control, load disturbances, sensor noise, process uncertainty and reference signals.

The most well known tuning methods are those developed by Ziegler and Nichols. They have had a major influence on the practice of PID control for more than half a century. The methods are based on characterization of process dynamics by a few parameters and simple equations for the controller parameters. It is surprising that the methods are so widely referenced because they give moderately good tuning only in restricted situations. Plausible explanations may be the simplicity of the methods and the fact that they can be used for simple student exercises in basic control courses.

The Step Response Method

One tuning method presented by Ziegler and Nichols is based on a process information in the form of the open-loop step response obtained from a bump test. This method can be viewed as a traditional method based on modeling and control where a very simple process model is used. The step response is characterized by only two parameters a and L, as shown in

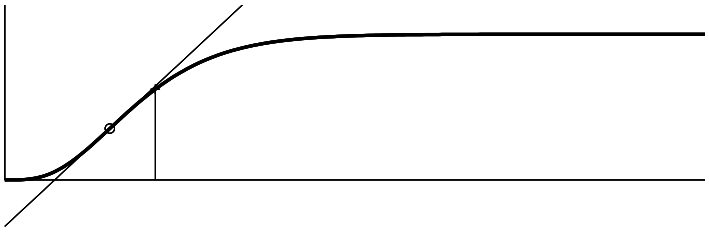


Figure 6.12 Characterization of a step response in the Ziegler-Nichols step response method.

Table 6.1 PID controller parameters obtained for the Ziegler-Nichols step response method.

Controller	K	T_i	T_d	T_p
P	1/a			4L
PI	0.9/a	3L		5.7L
PID	1.2/a	2L	L/2	3.4L

Figure 6.12.

The point where the slope of the step response has its maximum is first determined, and the tangent at this point is drawn. The intersections between the tangent and the coordinate axes give the parameters a and L. The controller parameters are then obtained from Table 6.1. An estimate of the period T_p of the closed-loop system is also given in the table.

The Frequency Response Method

A second method developed by Ziegler and Nichols is based on a simple characterization of the the frequency response of the process process dynamics. The design is based on knowledge of only one point on the Nyquist curve of the process transfer function $P(s)$, namely the point where the Nyquist curve intersects the negative real axis. This point can be characterized by two parameters the frequency ω_{180} and the gain at that frequency $k_{180} = |P(i\omega_{180})|$. For historical reasons the point has been called the ultimate point and characterized by the parameters $K_u = 1/K_{180}$ and $T_u = 2\pi/\omega_{180}$, which are called the *ultimate gain* and the *ultimate period*. These parameters can be determined in the following way. Connect a controller to the process, set the parameters so that control action is proportional, i.e., $T_i = \infty$ and $T_d = 0$. Increase the gain slowly until the

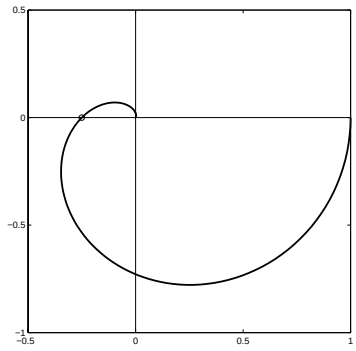


Figure 6.13 Characterization of a step response in the Ziegler-Nichols step response method.

Table 6.2 Controller parameters for the Ziegler-Nichols frequency response method.

Controller	K	T_i	T_d	T_p
P	$0.5K_u$			T_u
PI	$0.4K_u$	$0.8T_u$		$1.4T_u$
PID	$0.6K_u$	$0.5T_u$	$0.125T_u$	$0.85T_u$

process starts to oscillate. The gain when this occurs is K_u and the period of the oscillation is T_u. The parameters of the controller are then given by Table 6.2. An estimate of the period T_p of the dominant dynamics of the closed-loop system is also given in the table.

The frequency response method can be viewed as an empirical tuning procedure where the controller parameters are obtained by direct experiments on the process combined with some simple rules. For a proportional controller the rule is simply to increase the gain until the process oscillates and then reduce it by 50%.

Assessment of the Ziegler Nichols Methods

The Ziegler-Nichols tuning rules were developed to give closed loop systems with good attenuation of load disturbances. The methods were based on extensive simulations. The design criterion was quarter amplitude decay ratio, which means that the amplitude of an oscillation should be reduced by a factor of four over a whole period. This corresponds to closed loop poles with a relative damping of about $\zeta = 0.2$, which is too small.

Controllers designed by the Ziegler-Nichols rules thus inherently give closed loop systems with poor robustness. It also turns out that it is not sufficient to characterize process dynamics by two parameters only. The methods developed by Ziegler and Nichols have been been very popular in spite of these drawbacks. Practically all manufacturers of controller have used the rules with some modifications in recommendations for controller tuning. One reason for the popularity of the rules is that they are simple and easy to explain. The tuning rules give ball park figures. Final tuning is then done by trial and error. Another (bad) reason is that the rules lend themselves very well to simple exercises for control education.

With the insight into controller design that has developed over the years it is possible to develop improved tuning rules that are almost as simple as the Zigler-Nichols rules. These rules are developed by starting with a solid design method that gives robust controllers with effective disturbance attenuation. We illustrate with some rules where the process is characterized by three parameters.

An Improved Step Response Method

This method characterizes the unit step response by three parameters K, L and T for stable processes and $Kv = K/T$ and L for integrating processes. This parameterization matches the transfer functions

$$P_1(s) = \frac{k_p}{1+sT}e^{-sL}$$
$$P_2(s) = \frac{k_v}{s}e^{-sL}$$

The transfer function $P_1(s)$, which is called a first order system with time delay or a KLT model. Parameter L is determined from the intercept of the tangent with largest slope with the time axis as was described in Figure 6.12. Parameter T is also determined as shown in the figure as the difference between the time when the step response reaches 63% of its steady state value. Parameter k_p is the static gain of the system. The parameter k_v is the largest slope of the unit step response. Parameter L is called the apparent time delay and parameter T the apparent time constant or the apparent lag. The adverb apparent is added to indicate that parameters are based on approximations. The parameter

$$\tau = \frac{L}{L+T}$$

is called the relative time delay. This parameter is a good indicator of process dynamics.

To obtain improved tuning rules we use a design method that maximizes integral gain subject to the robustness constraint that the maximum sensitivity is less than $M_s = 1.4$. The procedure has been applied to a a large test batch representing many different processes. One tuning rule is

$$K = \begin{cases} 0.3\dfrac{T}{K_v L} & \text{for} + L < 2T \\ 0.15K_p & \text{for } 2T < L \end{cases}$$
$$T_i = \begin{cases} 8L & \text{for } L < 0.1T \\ 0.8T & \text{for } 0.1T < L < 2T \\ 0.4L & \text{for } 2T < L \end{cases} \tag{6.12}$$

The properties of the improved tuning rules are illustrated by applying them to systems with the transfer functions

$$P_1(s) = \frac{1}{(s+1)(0.2s+1)}$$
$$P_2(s) = \frac{1}{(s+1)^4}$$
$$P_3(s) = \frac{1}{0.05s+1)^2} e^{-1.2s}$$

The process $P_1(s)$ has lag dominated dynamics, process $P_3(s)$ has delay dominated dynamics and process $P_2(s)$ has balanced dynamics.

Figure 6.14 shows the response to a step change in the reference at time zero and a step change in a load disturbance at the process input for PI control of the process $P_1(s)$. The dashed lines show the responses obtained by the Ziegler-Nichols step response method and the full line shows the response obtained with the improved rule which restricted the maximum sensitivity to 1.4. The oscillatory responses to obtained by the Ziegler-Nichols method are clearly visible in the figure which reflects the design choice of quarter amplitude damping. The response to load disturbances obtained by the Ziegler-Nichols method comes at a price of poor sensitivity. There is also a very large overshoot in the response to reference values. Figure 6.15 shows the corresponding responses for the system $P_4(s)$. The oscillatory character obtained with Ziegler Nichols tuning is clearly visible. Figure 6.16 shows the response for a process that is delay dominated. The figure shows that Ziegler-Nichols tuning performs very poorly for this process. Overall we find that the improved tuning rules work for a wide range of processes and that they give robust systems with good responses.

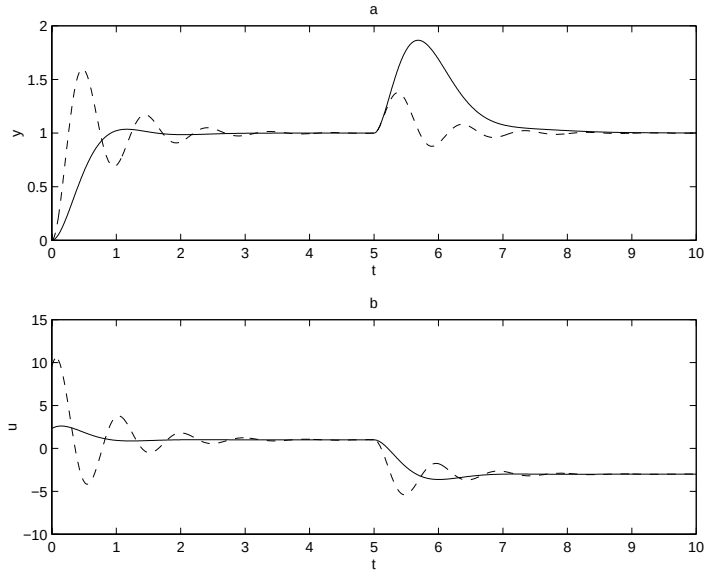


Figure 6.14 Behavior of closed loop systems with PI controllers designed by the Ziegler-Nichols rule (dashed) and the improved tuning rules (solid). The process has lag dominated dynamics with the transfer function $P(s) = \frac{1}{(s+1)(0.2s+1)}$.

6.7 Computer Implementation

Most controllers are nowadays implemented in computers. In this section we will discuss many practical issues related to computer implementation.

Sampling

When the controller is implemented in a computer, the analog inputs are read and the outputs are set with a certain sampling period. This is a drawback compared to the analog implementations, since the sampling introduces dead-time in the control loop.

When a digital computer is used to implement a control law, the ideal sequence of operation is the following.

1. Wait for clock interrupt
2. Read analog input
3. Compute control signal
4. Set analog output

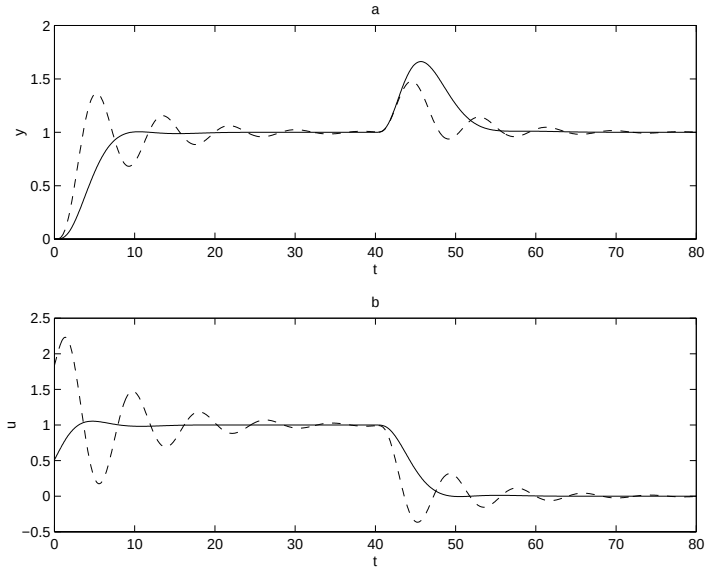


Figure 6.15 Behavior of closed loop systems with PI controllers designed by the Ziegler-Nichols rule (dashed) and the improved tuning rules (solid). The process has balanced dynamics with the transfer function $P(s) = \frac{1}{(s+1)^4}$.

5. Update controller variables
6. Go to 1

With this implementation, the delay is minimized. If the analog input is read with a sampling period h, the average delay of the measurement signal is $h/2$. The computation time is often short compared to the sampling period. This means that the total delay is about $h/2$. However, most controllers and instrument systems do not organize the calculation in this way. Therefore, the delays introduced because of the sampling is often several sampling periods.

Aliasing

The sampling mechanism introduces some unexpected phenomena, which must be taken into account in a good digital implementation of a PID controller. To explain these, consider the signals

$$s(t) = \cos(n\omega_s t \pm \omega t)$$

and

$$s_a(t) = \cos(\omega t)$$

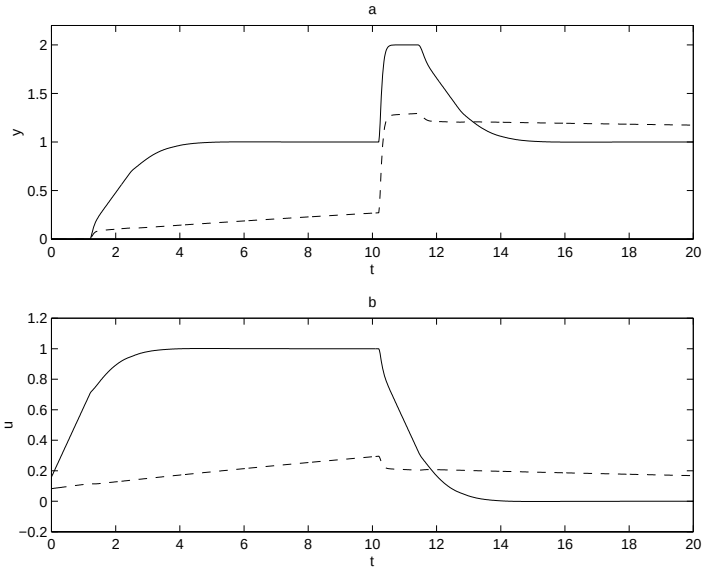


Figure 6.16 Behavior of closed loop systems with PI controllers designed by the Ziegler-Nichols rule (dashed) and the improved tuning rules (solid). The process has delay dominated dynamics with the transfer function $P(s) = \frac{1}{0.05s+1)^2} e^{-1.2s}$.

where $\omega_s = 2\pi/h$ [rad/s] is the sampling frequency. Well-known formulas for the cosine function imply that the values of the signals at the sampling instants $[kh, k = 0, 1, 2, ...]$ have the property

$$s(kh) = \cos(nkh\omega_s \pm \omega kh) = \cos(\omega kh) = s_a(\omega kh)$$

The signals s and s_a thus have the same values at the sampling instants. This means that there is no way to separate the signals if only their values at the sampling instants are known. Signal s_a is, therefore, called an *alias* of signal s. This is illustrated in Figure 6.17. A consequence of the aliasing effect is that a high-frequency disturbance after sampling may appear as a low-frequency signal. In Figure 6.17 the sampling period is 1 s and the sinusoidal disturbance has a period of 6/5 s. After sampling, the disturbance appear as a sinusoid with the frequency

$$f_a = 1 - \frac{5}{6} = 1/6 \text{ Hz}$$

This low-frequency signal with time period 6 s is seen in the figure.

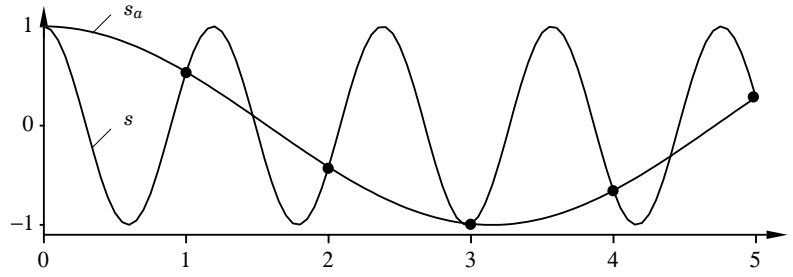


Figure 6.17 Illustration of the aliasing effect. The diagram shows signal s and its alias s_a.

Prefiltering

The aliasing effect can create significant difficulties if proper precautions are not taken. High frequencies, which in analog controllers normally are effectively eliminated by low-pass filtering, may, because of aliasing, appear as low-frequency signals in the bandwidth of the sampled control system. To avoid these difficulties, an analog prefilter (which effectively eliminates all signal components with frequencies above half the sampling frequency) should be introduced. Such a filter is called an anti-aliasing filter. A second-order Butterworth filter is a common anti-aliasing filter. Higher-order filters are also used in critical applications. The selection of the filter bandwidth is illustrated by the following example.

EXAMPLE 6.3—SELECTION OF PREFILTER BANDWIDTH

Assume it is desired that the prefilter attenuate signals by a factor of 16 at half the sampling frequency. If the filter bandwidth is ω_b and the sampling frequency is ω_s, we get

$$(\omega_s/2\omega_b)^2 = 16$$

Hence,

$$\omega_b = \frac{1}{8}\,\omega_s$$

□

Notice that the dynamics of the prefilter is often significant. It should be accounted for in the control design by combining it with the process dynamics.

Discretization

To implement a continuous-time control law, such as a PID controller in a digital computer, it is necessary to approximate the derivatives and the integral that appear in the control law. A few different ways to do this are presented below.

Proportional Action The proportional term is

$$P = K(by_{sp} - y)$$

This term is implemented simply by replacing the continuous variables with their sampled versions. Hence,

$$P(t_k) = K\left(by_{sp}(t_k) - y(t_k)\right) \tag{6.13}$$

where $\{t_k\}$ denotes the sampling instants, i.e., the times when the computer reads the analog input.

Integral Action The integral term is given by

$$I(t) = \frac{K}{T_i}\int_0^t e(s)ds$$

It follows that

$$\frac{dI}{dt} = \frac{K}{T_i}e \tag{6.14}$$

The derivative is approximated by a forward difference gives

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{K}{T_i}e(t_k)$$

This leads to the following recursive equation for the integral term

$$I(t_{k+1}) = I(t_k) + \frac{Kh}{T_i}e(t_k) \tag{6.15}$$

Derivative Action The derivative term is given by Equation (6.2), i.e.

$$\frac{T_d}{N}\frac{dD}{dt} + D = -KT_d\frac{dy}{dt} \tag{6.16}$$

This equation can be approximated in the same way as the integral term. In this case we approximate the derivatives by a backward difference.

$$\frac{T_d}{N}\frac{D(t_k) - D(t_{k-1})}{h} + D(t_k) = -KT_d\frac{y(t_k) - y(t_{k-1})}{h}$$

This can be rewritten as

$$D(t_k) = \frac{T_d}{T_d + Nh} D(t_{k-1}) - \frac{KT_dN}{T_d + Nh} (y(t_k) - y(t_{k-1})) \tag{6.17}$$

The advantage by using a backward difference is that the parameter $T_d/(T_d + Nh)$ is in the range of 0 to 1 for all values of the parameters. This guarantees that the difference equation is stable.

Summarizing we find that the PID controller can be approximated by

$$\begin{aligned} p(t_k) &= k * (br(t_k) - y(t_k)) \\ e(t_k) &= r(t_k) - y(t_k) \\ d(t_k) &= \frac{T_d}{T_d + Nh} \Big(d(t_{k-1}) - kN\big(y(t_k) - y(t_{k-1})\big)\Big) \\ u(t_k) &= p(t_k) + i(t_k) + d(t_k) \\ i(t_{k+1}) &= i(t_k) + \frac{kh}{T_i} e(t_k) \end{aligned}$$

Velocity Algorithms

The algorithms described so far are called positional algorithms because the output of the algorithms is the control variable. In certain cases the control system is arranged in such a way that the control signal is driven directly by an integrator, e.g., a motor. It is then natural to arrange the algorithm in such a way that it gives the velocity of the control variable. The control variable is then obtained by integrating its velocity. An algorithm of this type is called a velocity algorithm. A block diagram of a velocity algorithm for a PID controller is shown in Figure 6.18.

Velocity algorithms were commonly used in many early controllers that were built around motors. In several cases, the structure was retained by the manufacturers when technology was changed in order to maintain functional compatibility with older equipment. Another reason is that many practical issues, like wind-up protection and bumpless parameter changes, are easy to implement using the velocity algorithm. This is discussed further in Sections 6.5 and 6.7. In digital implementations velocity algorithms are also called incremental algorithms.

Incremental algorithm

The incremental form of the PID algorithm is obtained by computing the time differences of the controller output and adding the increments.

$$\Delta u(t_k) = u(t_k) - u(t_{k-1}) = \Delta P(t_k) + \Delta I(t_k) + \Delta D(t_k)$$

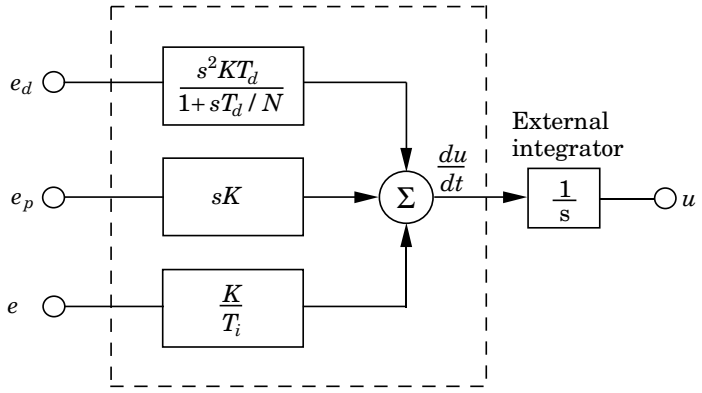


Figure 6.18 Block diagram of a PID algorithm in velocity form.

In some cases integration is performed externally. This is natural when a stepper motor is used. The output of the controller should then represent the increments of the control signal, and the motor implements the integrator. The increments of the proportional part, the integral part, and the derivative part are easily calculated from Equations 6.13, 6.15 and 6.17:

$$\Delta P(t_k) = P(t_k) - P(t_{k-1}) = K\,(by_{sp}(t_k) - y(t_k) - by_{sp}(t_{k-1}) + y(t_{k-1}))$$
$$\Delta I(t_k) = I(t_k) - I(t_{k-1}) = b_{i1}\,e(t_k) + b_{i2}\,e(t_{k-1})$$
$$\Delta D(t_k) = D(t_k) - D(t_{k-1}) = a_d \Delta D(t_{k-1}) - b_d\,(y(t_k) - 2y(t_{k-1}) + y(t_{k-2}))$$

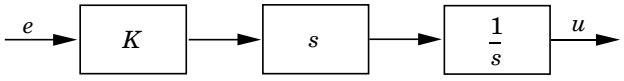
One advantage with the incremental algorithm is that most of the computations are done using increments only. Short word-length calculations can often be used. It is only in the final stage where the increments are added that precision is needed.

Velocity algorithms for controllers without integral action

A velocity algorithm cannot be used directly for a controller without integral action, because such a controller cannot keep the stationary value. This can be understood from the block diagram in Figure 6.19A, which shows a proportional controller in velocity form. Stationarity can be obtained for any value of the control error e, since the output from the derivation block is zero for any constant input. The problem can be avoided with the modification shown in Figure 6.19B. Here, stationarity is only obtained when $u = Ke + u_b$.

If a sampled PID controller is used, a simple version of the method illustrated in figure 6.19B is obtained by implementing the P controller

A



B

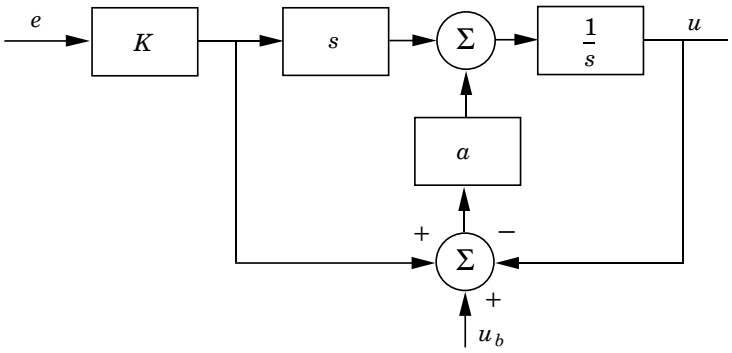


Figure 6.19 Illustrates the difficulty with a proportional controller in velocity form (A) and a way to avoid it (B).

as

$$\Delta u(t) = u(t) - u(t-h) = Ke(t) + u_b - u(t-h)$$

where h is the sampling period.

Feedforward control

In feedforward control, the control signal is composed of two terms,

$$u = u_{FB} + u_{FF}$$

Here u_{FB} is the feedback component and u_{FF} is the feedforward component, either from a measurable disturbance or from the setpoint.

To avoid integrator windup, it is important that the anti-windup mechanism acts on the final control signal u, and not only on the feedback component u_{FB}.

Unfortunately, many of the block-oriented instrument systems available today have the anti-windup mechanisms inside the feedback controller blocks, without any possibility to add feedforward signals to these blocks. Hence, the feedforward signals must be added after the controller blocks. This may lead to windup. Because of this, several tricks, like feeding the feedforward signal through high-pass filters, are used to reduces

the windup problem. These strategies do, however, lead to a less effective feedforward.

Incremental algorithms are efficient for feedforward implementation. By first adding *the increments* of the feedback and feedforward components,

$$\Delta u = \Delta u_{FB} + \Delta u_{FF}$$

and then forming the control signal as

$$u(t) = u(t-h) + \Delta u(t)$$

windup is avoided. This requires that the feedback control blocks have inputs for feedforward signals.

Operational Aspects

Practically all controllers can be run in two modes: manual or automatic. In manual mode the controller output is manipulated directly by the operator, typically by pushing buttons that increase or decrease the controller output. A controller may also operate in combination with other controllers, such as in a cascade or ratio connection, or with nonlinear elements, such as multipliers and selectors. This gives rise to more operational modes. The controllers also have parameters that can be adjusted in operation. When there are changes of modes and parameters, it is essential to avoid switching transients. The way the mode switchings and the parameter changes are made depends on the structure chosen for the controller.

Bumpless Transfer Between Manual and Automatic

Since the controller is a dynamic system, it is necessary to make sure that the state of the system is correct when switching the controller between manual and automatic mode. When the system is in manual mode, the control algorithm produces a control signal that may be different from the manually generated control signal. It is necessary to make sure that the two outputs coincide at the time of switching. This is called *bumpless transfer*.

Bumpless transfer is easy to obtain for a controller in incremental form. This is shown in Figure 6.20. The integrator is provided with a switch so that the signals are either chosen from the manual or the automatic increments. Since the switching only influences the increments there will not be any large transients.

A similar mechanism can be used in the series, or interacting, implementation of a PID controller shown in Figure 6.22. In this case there will be a switching transient if the output of the PD part is not zero at the switching instant.

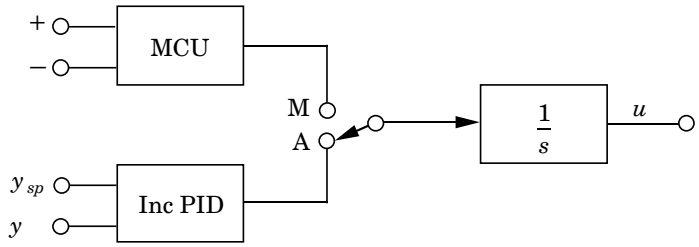


Figure 6.20 Bumpless transfer in a controller with incremental output. MCU stands for manual control unit.

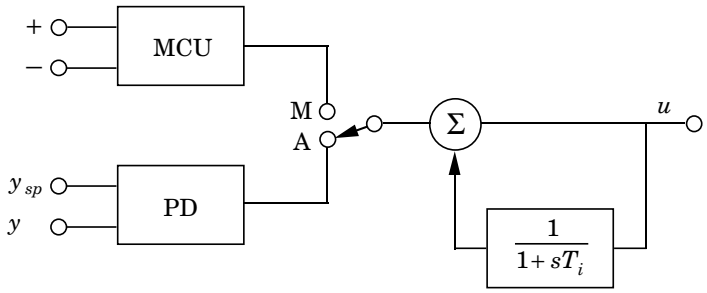


Figure 6.21 Bumpless transfer in a PID controller with a special series implementation.

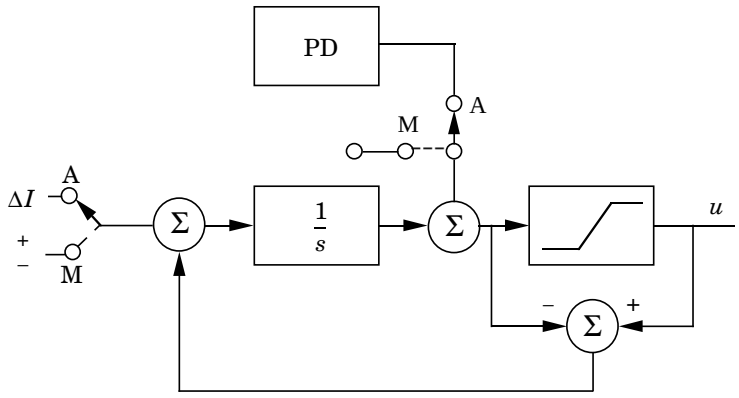


Figure 6.22 A PID controller where one integrator is used both to obtain integral action in automatic mode and to sum the incremental commands in manual mode.

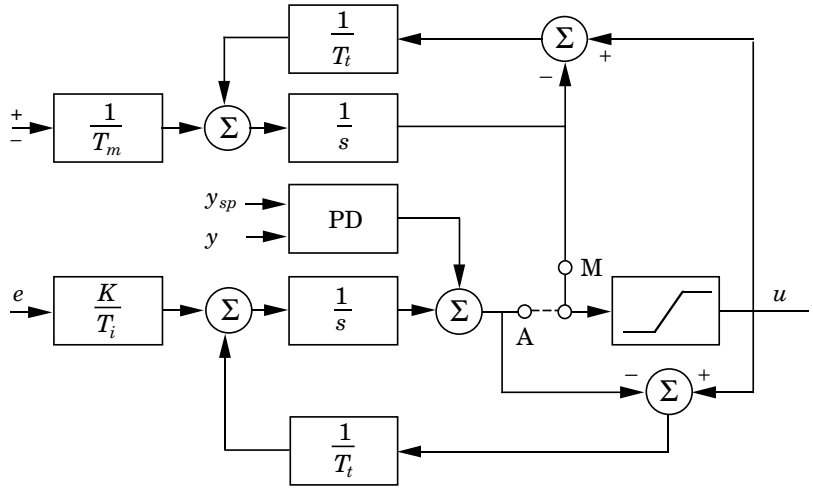


Figure 6.23 PID controller with parallel implementation that switches smoothly between manual and automatic control.

For controllers with parallel implementation, the integrator of the PID controller can be used to add up the changes in manual mode. The controller shown in Figure 6.22 is such a system. This system gives a smooth transition between manual and automatic mode provided that the switch is made when the output of the PD block is zero. If this is not the case, there will be a switching transient.

It is also possible to use a separate integrator to add the incremental changes from the manual control device. To avoid switching transients in such a system, it is necessary to make sure that the integrator in the PID controller is reset to a proper value when the controller is in manual mode. Similarly, the integrator associated with manual control must be reset to a proper value when the controller is in automatic mode. This can be realized with the circuit shown in Figure 6.23. With this system the switch between manual and automatic is smooth even if the control error or its derivative is different from zero at the switching instant. When the controller operates in manual mode, as is shown in Figure 6.23, the feedback from the output v of the PID controller tracks the output u. With efficient tracking the signal v will thus be close to u at all times. There is a similar tracking mechanism that ensures that the integrator in the manual control circuit tracks the controller output.

Bumpless Parameter Changes

A controller is a dynamical system. A change of the parameters of a dynamical system will naturally result in changes of its output. Changes in the output can be avoided, in some cases, by a simultaneous change of the state of the system. The changes in the output will also depend on the chosen realization. With a PID controller it is natural to require that there be no drastic changes in the output if the parameters are changed when the error is zero. This will hold for all incremental algorithms because the output of an incremental algorithm is zero when the input is zero, irrespective of the parameter values. For a position algorithm it depends, however, on the implementation.

Assume that the state is chosen as

$$x_I = \int^t e(\tau)d\tau$$

when implementing the algorithm. The integral term is then

$$I = \frac{K}{T_i}x_I$$

Any change of K or T_i will then result in a change of I. To avoid bumps when the parameters are changed, it is essential that the state be chosen as

$$x_I = \int^t \frac{K(\tau)}{T_i(\tau)}\, e(\tau)d\tau$$

when implementing the integral term.

With sensible precautions, it is easy to ensure bumpless parameter changes if parameters are changed when the error is zero. There is, however, one case where special precautions have to be taken, namely, if setpoint weighting is used. To have bumpless parameter changes in such a case it is necessary that the quantity $P + I$ is invariant to parameter changes. This means that when parameters are changed, the state I should be changed as follows

$$I_{\text{new}} = I_{\text{old}} + K_{\text{old}}(b_{\text{old}}\, y_{sp} - y) - K_{\text{new}}(b_{\text{new}}\, y_{sp} - y)$$

To build automation systems it is useful to have suitable modules. Figure 6.24 shows the block diagram for a manual control module. It has two inputs, a tracking input and an input for the manual control commands. The system has two parameters, the time constant T_m for the manual control input and the reset time constant T_t. In digital implementations

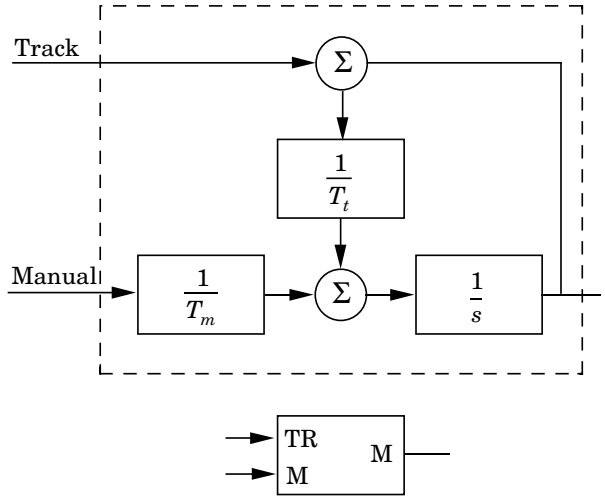


Figure 6.24 Manual control module.

it is convenient to add a feature so that the command signal accelerates as long as one of the increase-decrease buttons are pushed. Using the module for PID control and the manual control module in Figure 6.24, it is straightforward to construct a complete controller. Figure 6.25 shows a PID controller with internal or external setpoints via increase/decrease buttons and manual automatic mode. Notice that the system only has two switches.

Computer Code

As an illustration, the following is a computer code for a PID algorithm. The controller handles both anti-windup and bumpless transfer.

```
"Compute controller coefficients
bi=K*h/Ti                    "integral gain
ad=(2*Td-N*h)/(2*Td+N*h)
bd=2*K*N*Td/(2*Td+N*h)       "derivative gain
a0=h/Tt

"Bumpless parameter changes
I=I+Kold*(bold*ysp-y)-Knew*(bnew*ysp-y)

"Control algorithm
r=adin(ch1)                  "read setpoint from ch1
y=adin(ch2)                  "read process variable from ch2
```

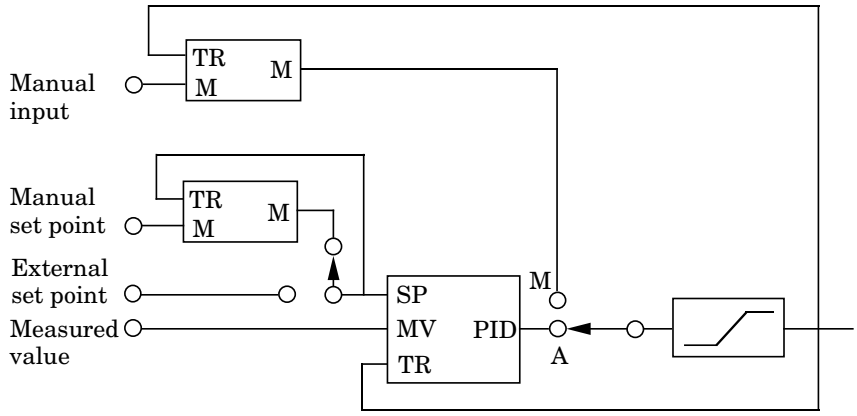


Figure 6.25 A reasonable complete PID controller with anti-windup, automatic-manual mode, and manual and external setpoint.

```
P=K*(b*ysp-y)                "compute proportional part
D=ad*D-bd*(y-yold)           "update derivative part
v=P+I+D                      "compute temporary output
u=sat(v,ulow,uhigh)          "simulate actuator saturation
daout(ch1)                   "set analog output ch1
I=I+bi*(ysp-y)+ao*(u-v)      "update integral
yold=y                       "update old process output
```

The computation of the coefficients should be done only when the controller parameters are changed. Precomputation of the coefficients ad, ao, bd, and bi saves computer time in the main loop. The main program must be called once every sampling period. The program has three states: yold, I, and D. One state variable can be eliminated at the cost of a less readable code. Notice that the code includes derivation of the process output only, proportional action on part of the error only ($b \neq 1$), and anti-windup.

6.8 Summary

In this Section we have given a detailed treatment of the PID controller, which is the most common way controller. A number of practical issues have been discussed. Simple controllers like the PI and PID controller are naturally not suitable for all processes. The PID controller is suitable for processes with almost monotone step responses provided that the

requirements are not too stringent. The quantity

$$m = \frac{\int_0^t g(t)dt}{\int_0^t |g(t)|dt}$$

where $g(t)$ is the impulse response can be used as a measure of monotonicity. PID control is not suitable for processes that are highly oscillatory or when the requirements are extreme.

The PI controller has no phase advance. This means that the PI controller will not work for systems which have phase lag of 180° or more. The double integrator is a typical example. Controller with derivative action can provide phase advance up to about 50°. Simple processes can be characterized by the relative time delay τ introduced in the Ziegler-Nichols tuning procedure. PI control is often satisfactory for processes that are lag dominated, i.e. when τ close to one. Derivative action is typically beneficial for processes with small relative delay τ.