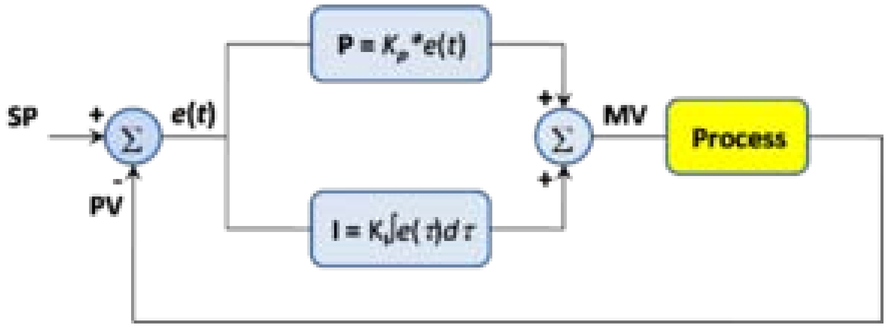
PI Controller

In control engineering, a **PI Controller** (proportional-integral controller) is a feedback controller which drives the plant to be controlled with a weighted sum of the error (difference between the output and desired set-point) and the integral of that value. It is a special case of the common PID controller in which the derivative (D) of the error is not used.



PI Controller Model

The controller output is given by

$$K_P\Delta + K_I \int \Delta \, dt$$

where Δ is the error or deviation of actual measured value (***PV***) from the set-point (***SP***).

Δ = **SP - PV**.

A PI controller can be modelled easily in software such as Simulink using a "flow chart" box involving Laplace operators:

$$C = \frac{G(1 + \tau s)}{\tau s}$$

where

$G = K_P$ = proportional gain
$G / \tau = K_I$ = integral gain

Setting a value for G is often a trade off between decreasing overshoot and increasing settling time.

Finding a value for τ

Finding a proper value for τ is an iterative process.

1) Set a value for G from the optimal range.

2) View the Nichols plot for the open-loop response of the system. Observe where the response curve crosses the 0dB line. This frequency is known as the cross-over frequency f_c.

3) The value of τ can be calculated as:

$$\tau = 1 / f_c$$

4) Decreasing τ decreases the phase margin, however it eliminates the steady-state errors quicker.

Advantages and disadvantages

- The integral term in a PI controller causes the steady-state error to reduce to zero, which is not the case for proportional-only control in general.
- The lack of derivative action may make the system more steady in the steady state in the case of noisy data. This is because derivative action is more sensitive to higher-frequency terms in the inputs.
- Without derivative action, a PI-controlled system is less responsive to real (non-noise) and relatively fast alterations in state and so the system will be slower to reach setpoint and slower to respond to perturbations than a well-tuned PID system may be.

PID controller

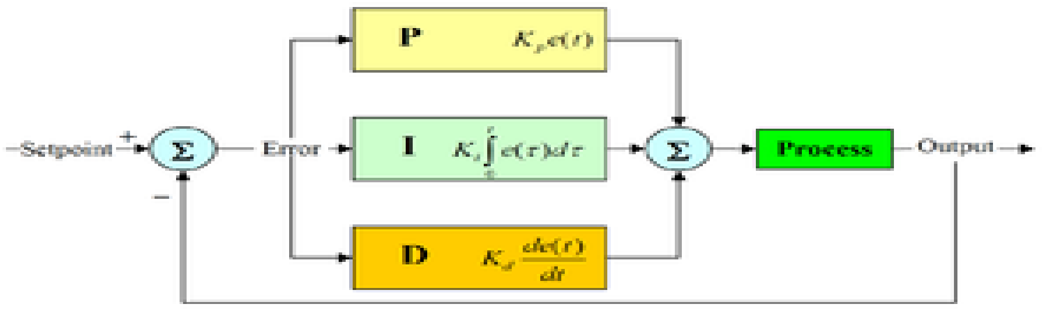
A **proportional–integral–derivative controller** (**PID controller**) is a generic control loop feedback mechanism (controller) widely used in industrial control systems – a PID is the most commonly used feedback controller. A PID controller calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs. In the absence of knowledge of the underlying process, PID controllers are the best controllers.[1] However, for best performance, the PID parameters used in the calculation must be tuned according to the nature of the system – while the design is generic, the parameters depend on the specific system.

The PID controller calculation (algorithm) involves three separate parameters, and is accordingly sometimes called **three-term control**: the proportional, the integral and derivative values, denoted *P*, *I*, and *D*. The *proportional* value determines the reaction to the current error, the *integral* value determines the reaction based on the sum of recent errors, and the *derivative* value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element. Heuristically, these values can be interpreted in terms of time: *P* depends on the *present* error, *I* on the accumulation of *past* errors, and *D* is a prediction of *future* errors, based on current rate of change.[2]

By tuning the three constants in the PID controller algorithm, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the setpoint and the degree of system oscillation. Note that the use of the PID algorithm for control does not guarantee optimal control of the system or system stability.

Some applications may require using only one or two modes to provide the appropriate system control. This is achieved by setting the gain of undesired control outputs to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions. PI controllers are fairly common, since derivative action is sensitive to measurement noise, whereas the absence of an integral value may prevent the system from reaching its target value due to the control action.

Note: Due to the diversity of the field of control theory and application, many naming conventions for the relevant variables are in common use.



PID controller theory

This section describes the parallel or non-interacting form of the PID controller. For other forms please see the section "Alternative nomenclature and PID forms".

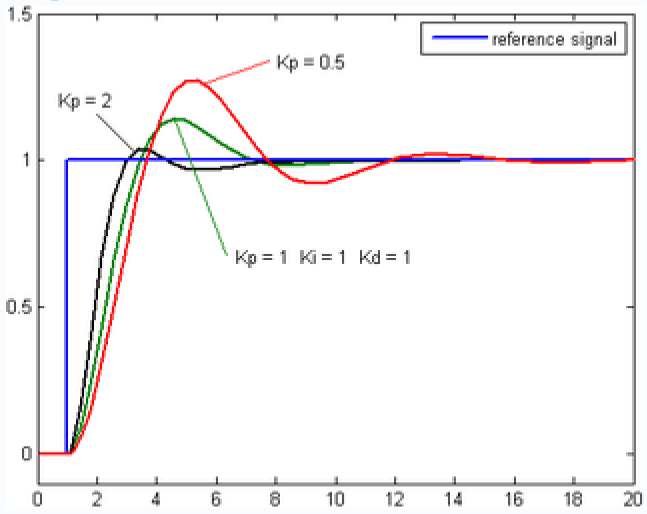
The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). Hence:

$$\mathrm{MV(t)} = P_{\mathrm{out}} + I_{\mathrm{out}} + D_{\mathrm{out}}$$

where

> P_{out}, I_{out}, and D_{out} are the contributions to the output from the PID controller from each of the three terms, as defined below.

Proportional term



Plot of PV vs time, for three values of K_p (K_i and K_d held constant)

The proportional term (sometimes called *gain*) makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain.

The proportional term is given by:

$$P_{\mathrm{out}} = K_p\, e(t)$$

where

P_{out}: Proportional term of output

K_p: Proportional gain, a tuning parameter

e: Error = $SP - PV$

t: Time or instantaneous time (the present)

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable (see the section on loop tuning). In contrast, a small gain results in a small output response to a large input error, and a less responsive (or sensitive) controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

Droop

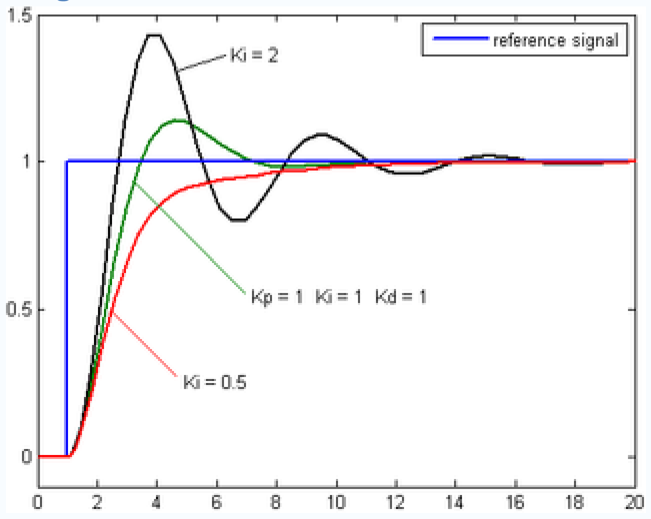
In the absence of disturbances, pure proportional control will not settle at its target value, but will retain a steady state error (**droop**) that is a function of the proportional gain and the process gain. Specifically, if the process gain – the long-term drift in the absence of control, such as cooling of a furnace towards room temperature – is denoted by G and assumed to be approximately constant in the error, then the droop is when this constant gain equals the proportional term of the output, P_{out}, which is *linear* in the error, $G = K_p e$, so $e = G / K_p$. This is when the proportional term, which is pushing the parameter towards the set point, is exactly offset by the process gain, which is pulling the parameter away from the set point. If the process gain is down, as in cooling, then the steady state will be *below* the set point, hence the term "droop".

Only the drift component (long-term average, zero-frequency component) of process gain matters for the droop – regular or random fluctuations above or below the drift cancel out. The process gain may change over time or in the presence of external changes, for example if room temperature changes, cooling may be faster or slower.

Droop is proportional to process gain and inversely proportional to proportional gain, and is an inevitable defect of purely proportional control. Droop can be mitigated by adding a *bias* term (setting the setpoint above the true desired value), or corrected by adding an integration term (in a PI or PID controller), which effectively computes a bias adaptively.

Despite the droop, both tuning theory and industrial practice indicate that it is the proportional term that should contribute the bulk of the output change.

Integral term



Plot of PV vs time, for three values of K_i (K_p and K_d held constant)

The contribution from the integral term (sometimes called *reset*) is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output. The magnitude of the contribution of the integral term to the overall control action is determined by the integral gain, K_i.

The integral term is given by:

$$I_{\mathrm{out}} = K_i \int_0^t e(\tau)\, d\tau$$

where

I_{out}: Integral term of output

K_i: Integral gain, a tuning parameter

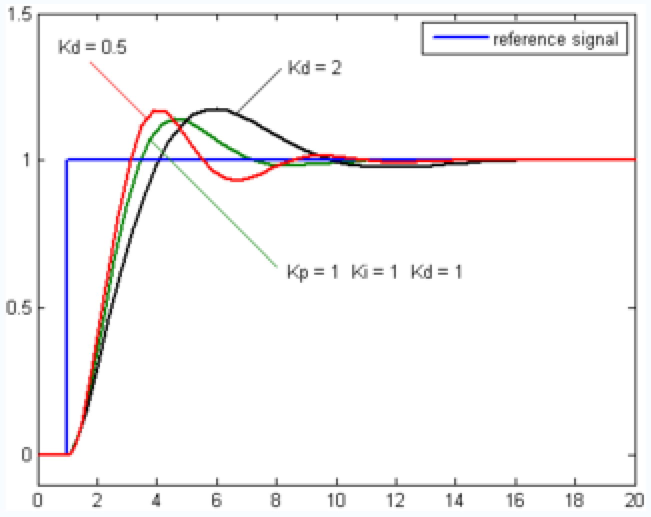
e: Error = $SP - PV$

t: Time or instantaneous time (the present)

τ: a dummy integration variable

The integral term (when added to the proportional term) accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the setpoint value (cross over the setpoint and then create a deviation in the other direction). For further notes regarding integral gain tuning and controller stability, see the section on loop tuning.

Derivative term



Plot of PV vs time, for three values of K_d (K_p and K_i held constant)

The rate of change of the process error is calculated by determining the slope of the error over time (i.e., its first derivative with respect to time) and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term (sometimes called *rate*) to the overall control action is termed the derivative gain, K_d.

The derivative term is given by:

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

where

D_{out}: Derivative term of output

K_d: Derivative gain, a tuning parameter

e: Error = $SP - PV$

t: Time or instantaneous time (the present)

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller setpoint. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability. However, differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term, and can cause a process to become unstable if the noise and the derivative gain are sufficiently large. Hence an approximation to a differentiator with a limited bandwidth is more commonly used. Such a circuit is known as a Phase-Lead compensator.

Summary

The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller output, the final form of the PID algorithm is:

$$\mathrm{u(t)} = \mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t)$$

where the tuning parameters are:

Proportional gain, K_p

Larger values typically mean faster response since the larger the error, the larger the proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

Integral gain, K_i

Larger values imply steady state errors are eliminated more quickly. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before reaching steady state.

Derivative gain, K_d

Larger values decrease overshoot, but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

Loop tuning

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. Stability (bounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and some desiderata conflict. Further, some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load; this can be corrected by gain scheduling (using different parameters in different operating regions). PID controllers often provide acceptable control even in the absence of tuning, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning.

PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the limitations of PID control. There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject of patents; this section describes some traditional manual methods for loop tuning.

Stability

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by *excess* gain, particularly in the presence of significant lag.

Generally, stability of response (the reverse of instability) is required and the process must not oscillate for any combination of process conditions and setpoints, though sometimes marginal stability (bounded oscillation) is acceptable or desired.

Optimum behavior

The optimum behavior on a process change or setpoint change varies depending on the application.

Two basic desiderata are *regulation* (disturbance rejection – staying at a given setpoint) and *command tracking* (implementing setpoint changes) – these refer to how well the controlled variable tracks the desired value. Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the setpoint if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new setpoint.

Overview of methods

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing P, I, and D based on the dynamic model parameters. Manual tuning methods can be relatively inefficient, particularly if the loops have response times on the order of minutes or longer.

The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and the response time of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

Choosing a Tuning Method

Method	Advantages	Disadvantages
Manual Tuning	No math required. Online method.	Requires experienced personnel.
Ziegler–Nichols	Proven Method. Online method.	Process upset, some trial-and-error, very aggressive tuning.
Software Tools	Consistent tuning. Online or offline method. May include valve and sensor analysis. Allow simulation before downloading.	Some cost and training involved.
Cohen-Coon	Good process models.	Some math. Offline method. Only good for first-order processes.

Manual tuning

If the system must remain online, one tuning method is to first set K_i and K_d values to zero. Increase the K_p until the output of the loop oscillates, then the K_p should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase K_i until any offset is correct in sufficient time for the process. However, too much K_i will cause instability. Finally, increase K_d, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much K_d will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot, in which case an *over-damped* closed-loop system is required, which will require a K_p setting significantly less than half that of the K_p setting causing oscillation.

Effects of *increasing* a parameter independently

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability[3]
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease[4]	Increase	Increase	Decrease significantly	Degrade
K_d	Minor decrease	Minor decrease	Minor decrease	No effect in theory	Improve if K_d small

Ziegler–Nichols method

For more details on this topic, see Ziegler–Nichols method.

Another heuristic tuning method is formally known as the Ziegler–Nichols method, introduced by John G. Ziegler and Nathaniel B. Nichols in the 1940s. As in the method above, the K_i and K_d gains are first set to zero. The P gain is increased until it reaches the ultimate gain, K_u, at which the output of the loop starts to oscillate. K_u and the oscillation period P_u are used to set the gains as shown:

Ziegler–Nichols method

Control Type	K_p	K_i	K_d
P	$0.50K_u$	-	-
PI	$0.45K_u$	$1.2K_p / P_u$	-
PID	$0.60K_u$	$2K_p / P_u$	$K_pP_u / 8$

PID tuning software

Most modern industrial facilities no longer tune loops using the manual calculation methods shown above. Instead, PID tuning and loop optimization software are used to ensure consistent results. These software packages will gather the data, develop process models, and suggest optimal tuning. Some software packages can even develop tuning by gathering data from reference changes.

Mathematical PID loop tuning induces an impulse in the system, and then uses the controlled system's frequency response to design the PID loop values. In loops with response times of several minutes, mathematical loop tuning is recommended, because trial and error can literally take days just to find a stable set of loop values. Optimal values are harder to find. Some digital loop controllers offer a self-tuning feature in which very small setpoint changes are sent to the process, allowing the controller itself to calculate optimal tuning values.

Other formulas are available to tune the loop according to different performance criteria. Many patented formulas are now embedded within PID tuning software and hardware modules.

Modifications to the PID algorithm

The basic PID algorithm presents some challenges in control applications that have been addressed by minor modifications to the PID form.

Integral windup

For more details on this topic, see Integral windup.

One common problem resulting from the ideal PID implementations is integral windup, where a large change in setpoint occurs (say a positive change) and the integral terms accumulates a significant error during the rise (windup), thus overshooting and continuing to increase as this accumulated error is unwound. This problem can be addressed by:

- Initializing the controller integral to a desired value
- Increasing the setpoint in a suitable ramp
- Disabling the integral function until the PV has entered the controllable region
- Limiting the time period over which the integral error is calculated
- Preventing the integral term from accumulating above or below pre-determined bounds

Freezing the integral function in case of disturbances

If a PID loop is used to control the temperature of an electric resistance furnace, the system has stabilized and then the door is opened and something cold is put into the furnace the temperature drops below the setpoint. The integral function of the controller tends to compensate this error by introducing another error in the positive direction. This can be avoided by freezing of the integral function after the opening of the door for the time the control loop typically needs to reheat the furnace.

Replacing the integral function by a model based part

Often the time-response of the system is approximately known. Then it is an advantage to simulate this time-response with a model and to calculate some unknown parameter from the actual response of the system. If for instance the system is an electrical furnace the response of the difference between furnace temperature and ambient temperature to changes of the electrical power will be similar to that of a simple RC low-pass filter multiplied by an unknown proportional coefficient. The actual electrical power supplied to the furnace is delayed by a low-pass filter to simulate the response of the temperature of the furnace and then the actual temperature minus the ambient temperature is divided by this low-pass filtered electrical power. Then, the result is stabilized by another low-pass filter leading to an estimation of the proportional coefficient. With this estimation it is possible to calculate the required electrical power by dividing the set-point of the temperature minus the ambient temperature by this coefficient. The result can then be used instead of the integral function. This also achieves a control error of zero in the steady-state but avoids integral windup and can give a significantly improved control action compared to an optimized PID controller. This type of controller does work properly in an open loop situation which causes integral windup with an integral function. This is an advantage if for example the heating of a furnace has to be reduced for some time because of the failure of a heating element or if the controller is used as an advisory system to a human operator who may or may not switch it to closed-loop operation or if the controller is used inside of a branch of a complex control system where this branch may be temporarily inactive.

Many PID loops control a mechanical device (for example, a valve). Mechanical maintenance can be a major cost and wear leads to control degradation in the form of either stiction or a deadband in the mechanical response to an input signal. The rate of mechanical wear is mainly a function of how often a device is activated to make a change. Where wear is a significant concern, the PID loop may have an output deadband to reduce the frequency of activation of the output (valve). This is accomplished by modifying the controller to hold its output steady if the change would be small (within the defined deadband range). The calculated output must leave the deadband before the actual output will change.

The proportional and derivative terms can produce excessive movement in the output when a system is subjected to an instantaneous step increase in the error, such as a large setpoint change. In the case of the derivative term, this is due to taking the derivative of the error, which is very large in the case of an instantaneous step change. As a result, some PID algorithms incorporate the following modifications:

Derivative of output

In this case the PID controller measures the derivative of the output quantity, rather than the derivative of the error. The output is always continuous (i.e., never has a step change). For this to be effective, the derivative of the output must have the same sign as the derivative of the error.

Setpoint ramping

In this modification, the setpoint is gradually moved from its old value to a newly specified value using a linear or first order differential ramp function. This avoids the discontinuity present in a simple step change.

Setpoint weighting

Setpoint weighting uses different multipliers for the error depending on which element of the controller it is used in. The error in the integral term must be the true control error to avoid steady-state control errors. This affects the controller's setpoint response. These parameters do not affect the response to load disturbances and measurement noise.

History

PID theory developed by observing the action of helmsmen.

PID controllers date to 1890s governor design.[1][5] PID controllers were subsequently developed in automatic ship steering. One of the earliest examples of a PID-type controller was developed by Elmer Sperry in 1911,[6] while the first published theoretical analysis of a PID controller was by Russian American engineer Nicolas Minorsky, in (Minorsky 1922). Minorsky was designing automatic steering systems for the US Navy, and based his analysis on observations of a helmsman, observing that the helmsman controlled the ship not only based on the current error, but also on past error and current rate of change;[7] this was then made mathematical by Minorsky. The Navy ultimately did not adopt the system, due to resistance by personnel. Similar work was carried out and published by several others in the 1930s.

Initially controllers were pneumatic, hydraulic, or mechanical, with electrical systems later developing, with wholly electrical systems developed following World War II.

Minorsky's work

Minorsky developed the PID on the USS *New Mexico*.

In detail, Minorsky's work proceeded as follows.[8] His goal was stability, not general control, which significantly simplified the problem. While proportional control provides stability against small disturbances, it was insufficient for dealing with a steady disturbance, notably a stiff gale (due to droop), which required adding the integral term. Finally, the derivative term was added to improve control. Trials were carried out on the USS *New Mexico*, with the controller controlling the *angular velocity* (not angle) of the rudder. PI control yielded sustained yaw (angular error) of ±2°, while adding D yielded yaw of ±1/6°, better than most helmsmen could achieve.

Limitations of PID control

While PID controllers are applicable to many control problems, and often perform satisfactorily without any improvements or even tuning, they can perform poorly in some applications, and do not in general provide *optimal* control. The fundamental difficulty with PID control is that it is a feed*back* system, with *constant* parameters, and no direct knowledge of the process, and thus overall performance is reactive and a compromise – while PID control is the best controller with no model of the process,[1] better performance can be obtained by incorporating a model of the process.

The most significant improvement is to incorporate feed-forward control with knowledge about the system, and using the PID only to control error. Alternatively, PIDs can be modified in more minor ways, such as by changing the parameters (either gain scheduling in different use cases or adaptively modifying them based on performance), improving measurement (higher sampling rate, precision, and accuracy, and low-pass filtering if necessary), or cascading multiple PID controllers.

PID controllers, when used alone, can give poor performance when the PID loop gains must be reduced so that the control system does not overshoot, oscillate or *hunt* about the control setpoint value. They also have difficulties in the presence of non-linearities, may trade off regulation versus response time, do not react to changing process behavior (say, the process changes after it has warmed up), and have lag in responding to large disturbances.

Linearity

Another problem faced with PID controllers is that they are linear, and in particular symmetric. Thus, performance of PID controllers in non-linear systems (such as HVAC systems) is variable. For example, in temperature control, a common use case is active heating (via a heating element) but passive cooling (heating off, but no cooling), so overshoot can only be corrected slowly – it cannot be forced downward. In this case the PID should be tuned to be overdamped, to prevent or reduce overshoot, though this reduces performance (it increases settling time).

Noise in derivative

A problem with the Derivative term is that small amounts of measurement or process noise can cause large amounts of change in the output. It is often helpful to filter the measurements with a low-pass filter in order to remove higher-frequency noise components. However, low-pass filtering and derivative control can cancel each other out, so reducing noise by instrumentation means is a much better choice. Alternatively, a nonlinear median filter may be used, which improves the filtering efficiency and practical performance [9]. In some case, the differential band can be turned off in many systems with little loss of control. This is equivalent to using the PID controller as a *PI* controller.

Improvements

Feed-forward

The control system performance can be improved by combining the feedback (or closed-loop) control of a PID controller with feed-forward (or open-loop) control. Knowledge about the system (such as the desired acceleration and inertia) can be fed forward and combined with the PID output to improve the overall system performance. The feed-forward value alone can often provide the major portion of the controller output. The PID controller can be used primarily to respond to whatever difference or *error* remains between the setpoint (SP) and the actual value of the process variable (PV). Since the feed-forward output is not affected by the process feedback, it can never cause the control system to oscillate, thus improving the system response and stability.

For example, in most motion control systems, in order to accelerate a mechanical load under control, more force or torque is required from the prime mover, motor, or actuator. If a velocity loop PID controller is being used to control the speed of the load and command the force or torque being applied by the prime mover, then it is beneficial to take the instantaneous acceleration desired for the load, scale that value appropriately and add it to the output of the PID velocity loop controller. This means that whenever the load is being accelerated or decelerated, a proportional amount of force is commanded from the prime mover regardless of the feedback value. The PID loop in this situation uses the feedback information to effect any increase or decrease of the combined output in order to reduce the remaining difference between the process setpoint and the feedback value. Working together, the combined open-loop feed-forward controller and closed-loop PID controller can provide a more responsive, stable and reliable control system.

Other improvements

In addition to feed-forward, PID controllers are often enhanced through methods such as PID gain scheduling (changing parameters in different operating conditions), fuzzy logic or computational verb logic [10] [11] . Further practical application issues can arise from instrumentation connected to the controller. A high enough sampling rate, measurement precision, and measurement accuracy are required to achieve adequate control performance.

Cascade control

One distinctive advantage of PID controllers is that two PID controllers can be used together to yield better dynamic performance. This is called cascaded PID control. In cascade control there are two PIDs arranged with one PID controlling the set point of another. A PID controller acts as outer loop controller, which controls the primary physical parameter, such as fluid level or velocity. The other controller acts as inner loop controller, which reads the output of outer loop controller as set point, usually controlling a more rapid changing parameter, flowrate or acceleration. It can be mathematically proven[citation needed] that the working frequency of the controller is increased and the time constant of the object is reduced by using cascaded PID controller.[vague].

Physical implementation of PID control

In the early history of automatic process control the PID controller was implemented as a mechanical device. These mechanical controllers used a lever, spring and a mass and were often energized by compressed air. These pneumatic controllers were once the industry standard.

Electronic analog controllers can be made from a solid-state or tube amplifier, a capacitor and a resistance. Electronic analog PID control loops were often found within more complex electronic systems, for example, the head positioning of a disk drive, the power conditioning of a power supply, or even the movement-detection circuit of a modern seismometer. Nowadays, electronic controllers have largely been replaced by digital controllers implemented with microcontrollers or FPGAs.

Most modern PID controllers in industry are implemented in programmable logic controllers (PLCs) or as a panel-mounted digital controller. Software implementations have the advantages that they are relatively cheap and are flexible with respect to the implementation of the PID algorithm.

Variable voltages may be applied by the time proportioning form of Pulse-width modulation (PWM) – a cycle time is fixed, and variation is achieved by varying the proportion of the time during this cycle that the controller outputs +1 (or −1) instead of 0. On a digital system the possible proportions are discrete – e.g., increments of .1 second within a 2 second cycle time yields 20 possible steps: percentage increments of 5% – so there is a discretization error, but for high enough time resolution this yields satisfactory performance.

Alternative nomenclature and PID forms

Ideal versus standard PID form

The form of the PID controller most often encountered in industry, and the one most relevant to tuning algorithms is the *standard form*. In this form the K_p gain is applied to the I_{out}, and D_{out} terms, yielding:

$$\mathrm{MV(t)} = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau + T_d \frac{d}{dt} e(t) \right)$$

where

T_i is the *integral time*

T_d is the *derivative time*

In the ideal parallel form, shown in the controller theory section

$$\mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

the gain parameters are related to the parameters of the standard form through $K_i = \frac{K_p}{T_i}$ and $K_d = K_p T_d$. This parallel form, where the parameters are treated as simple gains, is the most general and flexible form. However, it is also the form where the parameters have the least physical interpretation and is generally reserved for theoretical treatment of the PID controller. The standard form, despite being slightly more complex mathematically, is more common in industry.

Laplace form of the PID controller

Sometimes it is useful to write the PID regulator in Laplace transform form:

$$G(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

Having the PID controller written in Laplace form and having the transfer function of the controlled system makes it easy to determine the closed-loop transfer function of the system.

Series/interacting form

Another representation of the PID controller is the series, or *interacting* form

$$G(s) = K_c \frac{(\tau_i s + 1)}{\tau_i s} (\tau_d s + 1)$$

where the parameters are related to the parameters of the standard form through

$K_p = K_c \cdot \alpha$, $T_i = \tau_i \cdot \alpha$, and

$$T_d = \frac{\tau_d}{\alpha}$$

with

$$\alpha = 1 + \frac{\tau_d}{\tau_i}.$$

This form essentially consists of a PD and PI controller in series, and it made early (analog) controllers easier to build. When the controllers later became digital, many kept using the interacting form.

Discrete implementation

The analysis for designing a digital implementation of a PID controller in a Microcontroller (MCU) or FPGA device requires the standard form of the PID controller to be *discretised.* Approximations for first-order derivatives are made by backward finite differences. The integral term is discretised, with a sampling time Δt,as follows,

$$\int_0^{t_k} e(\tau)\,d\tau = \sum_{i=1}^{k} e(t_i)\Delta t$$

The derivative term is approximated as,

$$\frac{de(t_k)}{dt} = \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$

Thus, a *velocity algorithm* for implementation of the discretised PID controller in a MCU is obtained by differentiating $u(t)$, using the numerical definitions of the first and second derivative and solving for $u(t_k)$ and finally obtaining:

$$u(t_k) = u(t_{k-1}) + K_p \bigg[\left(1 + \frac{\Delta t}{T_i} + \frac{T_d}{\Delta t}\right) e(t_k) + \left(-1 - \frac{2T_d}{\Delta t}\right) e(t_{k-1}) + \frac{T_d}{\Delta t} e(t_{k-2})$$

Pseudocode

Here is a simple software loop that implements the PID algorithm in its 'ideal, parallel' form:

```
previous error = 0
integral = 0
start:
  error = setpoint - actual position
  integral = integral + (error*dt)
  derivative = (error - previous_error)/dt
  output = (Kp*error) + (Ki*integral) + (Kd*derivative)
  previous error = error
  wait(dt)
  goto start
```