



**WIKIPEDIA**  
The Free Encyclopedia

- [Main page](#)
- [Contents](#)
- [Featured content](#)
- [Current events](#)
- [Random article](#)
- [Donate to Wikipedia](#)

Interaction

- [Help](#)
- [About Wikipedia](#)
- [Community portal](#)
- [Recent changes](#)
- [Contact Wikipedia](#)

Toolbox

Print/export

Languages

- [Български](#)
- [Català](#)
- [Česky](#)
- [Deutsch](#)
- [Español](#)
- [Français](#)
- [□□□](#)

- [Bahasa Indonesia](#)
- [Italiano](#)
- [Lietuvių](#)
- [Magyar](#)
- [Nederlands](#)
- [□□□](#)
- [Norsk \(bokmål\)](#)
- [Polski](#)
- [Português](#)
- [Русский](#)
- [Slovenčina](#)
- [Suomi](#)
- [Svenska](#)

- [Türkçe](#)
- [Tiếng Việt](#)
- [□□](#)

Article [Talk](#)

Read [Edit](#)



# PID controller

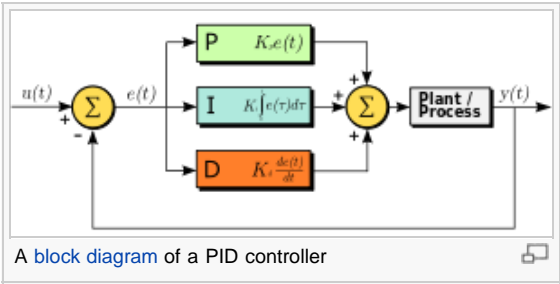
From Wikipedia, the free encyclopedia

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.

The PID controller calculation ([algorithm](#)) involves three separate constant parameters, and is accordingly sometimes called **three-term control**: the [proportional](#), the [integral](#) and [derivative](#) values, denoted *P*, *I*, and *D*. [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.<sup>[1]</sup> 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 supplied to a heating element.

In the absence of knowledge of the underlying process, a PID controller has historically been considered to be the best controller.<sup>[2]</sup> By tuning the three parameters 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 actions to provide the appropriate system control. This is achieved by setting the other parameters 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 term may prevent the system from reaching its target value due to the control action.



A [block diagram](#) of a PID controller

**Contents** [\[hide\]](#)

- [1 Control loop basics](#)
- [2 PID controller theory](#)
  - [2.1 Proportional term](#)
    - [2.1.1 Droop](#)
  - [2.2 Integral term](#)
  - [2.3 Derivative term](#)
- [3 Loop tuning](#)
  - [3.1 Stability](#)
  - [3.2 Optimum behavior](#)
  - [3.3 Overview of methods](#)
  - [3.4 Manual tuning](#)
  - [3.5 Ziegler–Nichols method](#)
  - [3.6 PID tuning software](#)
- [4 Modifications to the PID algorithm](#)
- [5 History](#)
- [6 Limitations of PID control](#)
  - [6.1 Linearity](#)
  - [6.2 Noise in derivative](#)
- [7 Improvements](#)

- [7.1 Feed-forward](#)
- [7.2 Other improvements](#)
- [8 Cascade control](#)
- [9 Physical implementation of PID control](#)
- [10 Alternative nomenclature and PID forms](#)
  - [10.1 Ideal versus standard PID form](#)
  - [10.2 Basing derivative action on PV](#)
  - [10.3 Basing proportional action on PV](#)
  - [10.4 Laplace form of the PID controller](#)
  - [10.5 PID Pole Zero Cancellation](#)
  - [10.6 Series/interacting form](#)
  - [10.7 Discrete implementation](#)
  - [10.8 Pseudocode](#)
- [11 PI controller](#)
- [12 Notes](#)
- [13 See also](#)
- [14 References](#)
- [15 External links](#)
  - [15.1 PID tutorials](#)
  - [15.2 Special topics and PID control applications](#)

## Control loop basics

[\[edit\]](#)

*Further information:* [Control system](#)

A familiar example of a control loop is the action taken when adjusting hot and cold faucets (valves) to maintain the water at a desired temperature. This typically involves the mixing of two process streams, the hot and cold water. The person touches the water to sense or measure its temperature. Based on this feedback they perform a control action to adjust the hot and cold water valves until the process temperature stabilizes at the desired value.

The sensed water temperature is the [process variable](#) or process value (PV). The desired temperature is called the setpoint (SP). The input to the process (the water valve position) is called the manipulated variable (MV). The difference between the temperature measurement and the setpoint is the error (e) and quantifies whether the water is too hot or too cold and by how much.

After measuring the temperature (PV), and then calculating the error, the controller decides when to change the tap position (MV) and by how much. When the controller first turns the valve on, it may turn the hot valve only slightly if warm water is desired, or it may open the valve all the way if very hot water is desired. This is an example of a simple **proportional** control. In the event that hot water does not arrive quickly, the controller may try to speed-up the process by opening up the hot water valve more-and-more as time goes by. This is an example of an **integral** control.

Making a change that is too large when the error is small is equivalent to a high gain controller and will lead to overshoot. If the controller were to repeatedly make changes that were too large and repeatedly overshoot the target, the output would [oscillate](#) around the setpoint in either a constant, growing, or decaying [sinusoid](#). If the oscillations increase with time then the system is unstable, whereas if they decrease the system is stable. If the oscillations remain at a constant magnitude the system is [marginally stable](#).

In the interest of achieving a gradual convergence at the desired temperature (SP), the controller may wish to [damp](#) the anticipated future oscillations. So in order to compensate for this effect, the controller may elect to temper its adjustments. This can be thought of as a **derivative** control method.

If a controller starts from a stable state at zero error (PV = SP), then further changes by the controller will be in response to changes in other measured or unmeasured inputs to the process that impact on the process, and hence on the PV. Variables that impact on the process other than the MV are known as disturbances. Generally controllers are used to reject disturbances and/or implement setpoint changes. Changes in feedwater temperature constitute a disturbance to the faucet temperature control process.

In theory, a controller can be used to control any process which has a measurable output (PV), a known ideal value for that output (SP) and an input to the process (MV) that will affect the relevant PV. Controllers are used in industry to regulate [temperature](#), [pressure](#), [flow rate](#), [chemical](#) composition, [speed](#) and practically every other variable for which a measurement exists.

PID controller theory

[edit]

*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). 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:

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

where

- $K_p$ : Proportional gain, a tuning parameter
- $K_i$ : Integral gain, a tuning parameter
- $K_d$ : Derivative gain, a tuning parameter
- $e$ : Error =  $SP - PV$
- $t$ : Time or instantaneous time (the present)

Proportional term

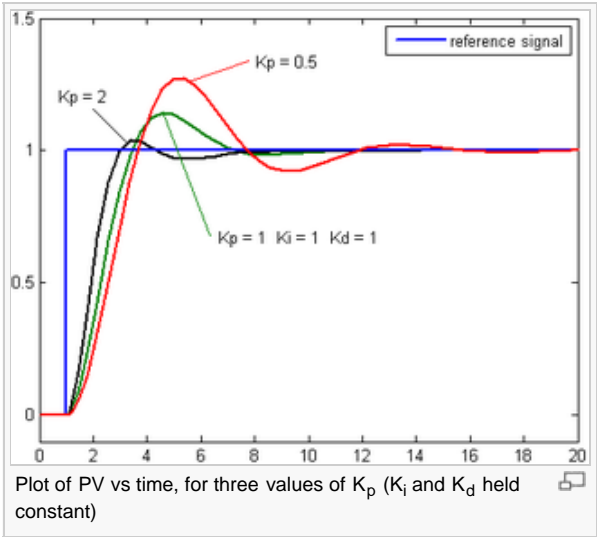
[edit]

The proportional term produces an output value 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:

$$P_{out} = K_p e(t)$$

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 less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. Tuning theory and industrial practice indicate that the proportional term should contribute the bulk of the output change.<sup>[[citation needed](#)]</sup>



Droop

[edit]

Because a non-zero error is required to drive the controller, a pure proportional controller generally operates with a steady-state error, referred to as *droop*.<sup>[[note 1](#)]</sup> Droop is proportional to the process gain and inversely proportional to proportional gain. Droop may be mitigated by adding a compensating [bias term](#) to the setpoint or output, or corrected by adding an integral term.

Integral term

[edit]

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The [integral](#) in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain ( $K_i$ ) and added to the controller output. The integral term is given by:

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

The integral term accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to **overshoot** the setpoint value (see [the section on loop tuning](#)).

Derivative term

The **derivative** of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain  $K_d$ . The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain,  $K_d$ .

The derivative term is given by:

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

The derivative term slows the rate of change of the controller output. Derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability. However, the derivative term slows the **transient response** of the controller. Also, 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**.

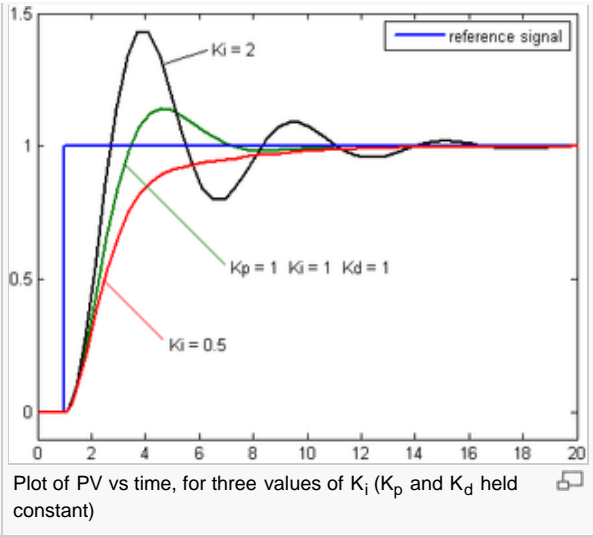
Loop tuning

*Tuning* a control loop is the adjustment of its control parameters (proportional band/gain, 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 requirements may conflict with one another.

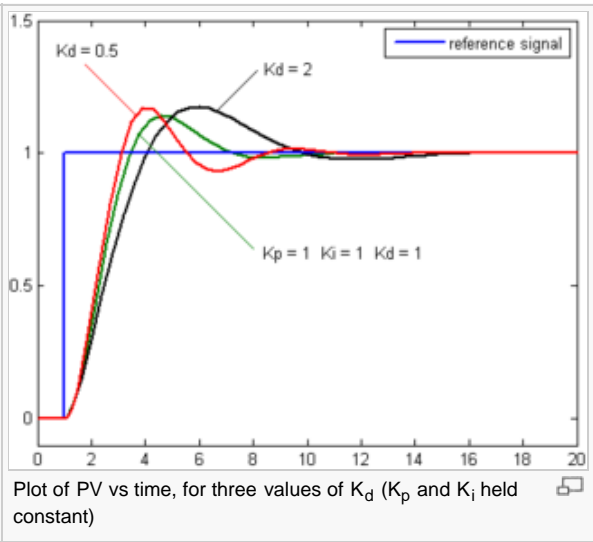
PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the **limitations of PID control**. There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject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

Designing and tuning a PID controller appears to be conceptually intuitive, but can be hard in practice, if multiple (and often conflicting) objectives such as short transient and high stability are to be achieved. Usually, initial designs need to be adjusted repeatedly through computer simulations until the closed-loop system performs or compromises as desired.

Some processes have a degree of **non-linearity** and so parameters that work well at full-load conditions don't work



[edit]



[edit]

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 using default tunings, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning.

**Stability** [[edit](#)]

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, stabilization of response 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.<sup>[*[citation needed](#)*]</sup>

**Optimum behavior** [[edit](#)]

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

Two basic requirements 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** [[edit](#)]

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. Can support Non-Steady State (NSS) Tuning. | Some cost and training involved.                                |
| Cohen-Coon               | Good process models.  | Some math. Offline method. Only good for first-order processes. |

**Manual tuning** [[edit](#)]

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 corrected 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 <sup>[3]</sup>              |
|----------------------|-------------------------|----------------|----------------|------------------------|---------------------------------------|
| <i>K<sub>p</sub></i> | Decrease                | Increase       | Small change   | Decrease               | Degrade                               |
| <i>K<sub>i</sub></i> | Decrease <sup>[4]</sup> | Increase       | Increase       | Decrease significantly | Degrade                               |
| <i>K<sub>d</sub></i> | Minor decrease          | Minor decrease | Minor decrease | No effect in theory    | Improve if <i>K<sub>d</sub></i> small |

Ziegler–Nichols method

[edit]

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<sub>i</sub>* and *K<sub>d</sub>* gains are first set to zero. The *P* gain is increased until it reaches the ultimate gain, *K<sub>u</sub>*, at which the output of the loop starts to oscillate. *K<sub>u</sub>* and the oscillation period *P<sub>u</sub>* are used to set the gains as shown:

Ziegler–Nichols method

| Control Type | <i>K<sub>p</sub></i>      | <i>K<sub>i</sub></i>                            | <i>K<sub>d</sub></i>                 |
|--------------|---------------------------|---|--------------------------------------|
| <i>P</i>     | 0.50 <i>K<sub>u</sub></i> | -   | -                                    |
| <i>PI</i>    | 0.45 <i>K<sub>u</sub></i> | 1.2 <i>K<sub>p</sub></i> / <i>P<sub>u</sub></i> | -                                    |
| <i>PID</i>   | 0.60 <i>K<sub>u</sub></i> | 2 <i>K<sub>p</sub></i> / <i>P<sub>u</sub></i>   | <i>K<sub>p</sub>P<sub>u</sub></i> /8 |

These gains apply to the ideal, parallel form of the PID controller. When applied to the standard PID form, the integral and derivative time parameters *T<sub>i</sub>* and *T<sub>d</sub>* are only dependent on the oscillation period *P<sub>u</sub>*. Please see the section "*Alternative nomenclature and PID forms*".

PID tuning software

[edit]

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 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.

Advances in automated PID Loop Tuning software also deliver algorithms for tuning PID Loops in a dynamic or Non-Steady State (NSS) scenario. The software will model the dynamics of a process, through a disturbance, and calculate PID control parameters in response.

Modifications to the PID algorithm

[edit]

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 term accumulates an error larger than the maximal value for the regulation variable (windup), thus the system overshoots and continues 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

**Overshooting from known disturbances**

For example, a PID loop is used to control the temperature of an electric resistance furnace, the system has stabilized. Now 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 overshoot 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 setpoint 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 not switch it to closed-loop operation. It may also be useful if the controller is inside a branch of a complex control system that 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 the Process Variable**

In this case the PID controller measures the derivative of the measured [process variable](#) (PV), rather than the derivative of the error. This quantity is always continuous (i.e., never has a step change as a result of changed setpoint). For this technique to be effective, the derivative of the PV must have the opposite sign of the derivative of the error, in the case of negative feedback control.

**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**

[\[edit\]](#)

 This section requires [expansion](#).

PID controllers date to 1890s governor design.<sup>[\[2\]](#)<sup>[\[5\]](#)</sup></sup> 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,<sup>[6]</sup> 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;<sup>[7]</sup> this was then made mathematical by Minorsky. 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  $\pm 2^\circ$ , while adding D yielded yaw of  $\pm 1/6^\circ$ , better than most helmsmen could achieve.<sup>[8]</sup>

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.

## Limitations of PID control

[\[edit\]](#)

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 *feedback* 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,<sup>[2]</sup> 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

[\[edit\]](#)

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

[\[edit\]](#)

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 is a much better choice. Alternatively, a nonlinear [median filter](#) may be used, which improves the filtering efficiency and practical performance.<sup>[9]</sup> 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

[\[edit\]](#)


PID theory developed by observing the action of [helmsmen](#).

[\[edit\]](#)



## 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 change the combined output 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

[\[edit\]](#)

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](#).<sup>[10]</sup><sup>[11]</sup> 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. Another new method for improvement of PID controller is to increase the degree of freedom by using fractional order. The order of the integrator and differentiator add increased flexibility to the controller.<sup>[*clarification needed*]</sup>

## Cascade control

[\[edit\]](#)

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 setpoint 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 setpoint, usually controlling a more rapid changing parameter, flowrate or acceleration. It can be mathematically proven<sup>[*citation needed*]</sup> that the working frequency of the controller is increased and the time constant of the object is reduced by using cascaded PID controller.<sup>[*vague*]</sup>.

## Physical implementation of PID control

[\[edit\]](#)

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.

[\[edit\]](#)

## Alternative nomenclature and PID forms

### Ideal versus standard PID form

[\[edit\]](#)

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:

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

where

$T_i$  is the *integral time*

$T_d$  is the *derivative time*

In this standard form, the parameters have a clear physical meaning. In particular, the inner summation produces a new single error value which is compensated for future and past errors. The addition of the proportional and derivative components effectively predicts the error value at  $T_d$  seconds (or samples) in the future, assuming that the loop control remains unchanged. The integral component adjusts the error value to compensate for the sum of all past errors, with the intention of completely eliminating them in  $T_i$  seconds (or samples). The resulting compensated single error value is scaled by the single gain  $K_p$ .

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

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

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.

### Basing derivative action on PV

[\[edit\]](#)

In most commercial control systems, derivative action is based on PV rather than error. This is because the digitised version of the algorithm produces a large unwanted spike when the SP is changed. If the SP is constant then changes in PV will be the same as changes in error. Therefore this modification makes no difference to the way the controller responds to process disturbances.

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

### Basing proportional action on PV

[\[edit\]](#)

Most commercial control systems offer the option of also basing the proportional action on PV. This means that only the integral action responds to changes in SP. While at first this might seem to adversely affect the time that the process will take to respond to the change, the controller may be retuned to give almost the same response - largely by increasing  $K_p$ . The modification to the algorithm does not affect the way the controller responds to process disturbances, but the change in tuning has a beneficial effect. Often the magnitude and duration of the disturbance will be more than halved. Since most controllers have to deal frequently with process disturbances and relatively rarely with SP changes, properly tuned the modified algorithm can dramatically improve process performance.

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

Tuning methods such as Ziegler-Nichols and Cohen-Coon will not be reliable when used with this algorithm. King<sup>[12]</sup> describes an effective chart-based method.

### Laplace form of the PID controller

[\[edit\]](#)

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.

**PID Pole Zero Cancellation**

[\[edit\]](#)

The PID equation can be written in this form:

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

When this form is used it is easy to determine the closed loop transfer function.

$$H(s) = \frac{1}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$$

If

$$\begin{aligned} \frac{K_i}{K_d} &= \omega_0^2 \\ \frac{K_p}{K_d} &= 2\zeta\omega_0 \end{aligned}$$

Then

$$G(s)H(s) = \frac{K_d}{s}$$

This can be very useful to remove unstable poles

**Series/interacting form**

[\[edit\]](#)

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

$$\begin{aligned} K_p &= K_c \cdot \alpha, T_i = \tau_i \cdot \alpha, \text{ and} \\ T_d &= \frac{\tau_d}{\alpha} \end{aligned}$$

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**

[\[edit\]](#)

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*.<sup>[13]</sup> Approximations for first-order derivatives are made by backward [finite differences](#). The integral term is discretised, with a sampling time  $\Delta t$ , 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:

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

**Pseudocode** [edit]

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

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

**PI controller** [edit]

A **PI Controller** (proportional-integral controller) is a special case of the PID controller in which the derivative (D) of the error is not used.

The controller output is given by

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

where  $\Delta$  is the error or deviation of actual measured value (**PV**) from the setpoint (**SP**).

$$\Delta = SP - PV.$$

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

$$G = K_P = \text{proportional gain}$$
$$G/\tau = K_I = \text{integral gain}$$

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

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.

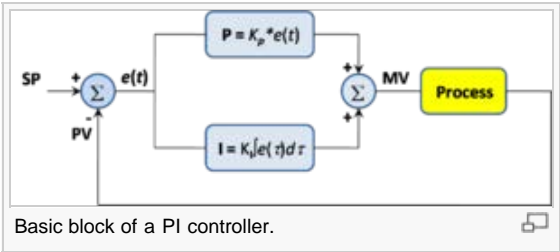
Without derivative action, a PI-controlled system is less responsive to real (non-noise) and relatively fast alterations in state and so the system will be slower to reach setpoint and slower to respond to perturbations than a well-tuned PID system may be.

**Notes** [edit]

- <sup>^</sup> The only exception is where the target value is the same as the value obtained when the proportional gain is equal to zero.


**See also** [edit]

- [Control theory](#)
- [Feedback](#)
- [Instability](#)
- [Oscillation](#)




References


















[edit]



This article **needs additional citations for verification**. Please help [improve this article](#) by adding citations to [reliable sources](#). Unsourced material may be [challenged](#) and [removed](#). *(June 2009)*



This article includes a [list of references](#), but **its sources remain unclear because it has insufficient inline citations**. Please help to [improve](#) this article by [introducing](#) more precise citations. *(January 2010)*











- <sup>↑</sup> Araki, M.. "PID Control" .
- <sup>↑</sup> <sup>**a**</sup> <sup>**b**</sup> <sup>**c**</sup> Bennett, Stuart (1993). *A history of control engineering, 1930-1955*. IET. p.p. 48 . ISBN 978-0-86341-299-8
- <sup>↑</sup> Ang, K.H., Chong, G.C.Y., and Li, Y. (2005). PID control system analysis, design, and technology, *IEEE Trans Control Systems Tech*, 13(4), pp.559-576. <http://eprints.gla.ac.uk/3817/> 
- <sup>↑</sup> Jinghua Zhong (Spring 2006). *PID Controller Tuning: A Short Tutorial* . Retrieved 2011-04-04.
- <sup>↑</sup> Bennett, Stuart (November 1984). "Nicholas Minorsky and the automatic steering of ships" . *IEEE Control Systems Magazine* **4** (4): 10–15. doi:10.1109/MCS.1984.1104827 . ISSN 0272-1708 
- <sup>↑</sup> "A Brief Building Automation History" . Retrieved 2011-04-04.
- <sup>↑</sup> (Bennett 1993, p. 67 )
- <sup>↑</sup> Bennett, Stuart (June 1986). *A history of control engineering, 1800-1930*. IET. pp.142–148 . ISBN 978-0-86341-047-5.
- <sup>↑</sup> Li, Y. and Ang, K.H. and Chong, G.C.Y. (2006) PID control system analysis and design - Problems, remedies, and future directions. *IEEE Control Systems Magazine*, 26 (1). pp. 32-41. ISSN 0272-1708 
- <sup>↑</sup> Yang, T. (June 2005). "Architectures of Computational Verb Controllers: Towards a New Paradigm of Intelligent Control". *International Journal of Computational Cognition* (Yang's Scientific Press) **3** (2): 74–101.
- <sup>↑</sup> Liang, Y.-L.(.) et al (2009). "Controlling fuel annealer using computational verb PID controllers". *Proceedings of the 3rd international conference on Anti-Counterfeiting, security, and identification in communication (IEEE)*: 417–420.
- <sup>↑</sup> King, Myke. *Process Control: A Practical Approach*. Wiley, 2010, p. 52-78
- <sup>↑</sup> "Discrete PI and PID Controller Design and Analysis for Digital Implementation" . Scribd.com. Retrieved 2011-04-04.
- Minorsky, Nicolas (1922). "Directional stability of automatically steered bodies". *J. Amer. Soc. Naval Eng.* **34** (2): 280–309
- Liptak, Bela (1995). *Instrument Engineers' Handbook: Process Control*. Radnor, Pennsylvania: Chilton Book Company. pp.20–29. ISBN 0-8019-8242-1.
- Tan, Kok Kiong; Wang Qing-Guo, Hang Chang Chieh (1999). *Advances in PID Control*. London, UK: Springer-Verlag. ISBN 1-85233-138-0.
- King, Myke (2010). *Process Control: A Practical Approach* . Chichester, UK: John Wiley & Sons Ltd.. ISBN 978-0-470-97587-9.
- Van, Doren, Vance J. (July 1, 2003). "Loop Tuning Fundamentals" . *Control Engineering* (Reed Business Information).
- Sellers, David. "An Overview of Proportional plus Integral plus Derivative Control and Suggestions for Its Successful Application and Implementation"  (PDF). Archived from the original  on March 7, 2007. Retrieved 2007-05-05.
- Graham, Ron (10/03/2005). "FAQ on PID controller tuning" . Retrieved 2009-01-05.

External links

[edit]

PID tutorials

[edit]

- PID Tutorial 
- P.I.D. Without a PhD : a beginner's guide to PID loop theory with sample programming code
- What's All This P-I-D Stuff, Anyhow?  Article in Electronic Design
- Shows how to build a PID controller with basic electronic components  <sup>[dead link]</sup> (pg. 22)
- Virtual PID Controller Laboratory 
  - PID Design & Tuning 
- Online PID Tuning applet from University of Texas Control Group  <sup>[dead link]</sup>
- example PID implementation in 16F887 microchip 
- PID Control with MATLAB and Simulink 
- Improving the Beginner's PID – Introduction 

Special topics and PID control applications

[edit]

- Proven Methods and Best Practices for PID Control 

- [PID Control Primer](#) <sup>[*dead link*]</sup> Article in Embedded Systems Programming

Rate this page  
**What's this?**

[View page ratings](#) 

|   |   |   |   |
|---|---|---|---|
| <input type="checkbox"/> Trustworthy  | <input type="checkbox"/> Objective  | <input type="checkbox"/> Complete   | <input type="checkbox"/> Well-written   |
|      |      |      |      |

I am highly knowledgeable about this topic (optional)

[Submit ratings](#)

Categories: [Control theory](#) | [Control engineering](#) | [Control devices](#) | [Classical control](#)

This page was last modified on 1 April 2012 at 04:23.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. See [Terms of use](#) for details. Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Contact us](#)

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Mobile view](#)

