CONTROL ENGINEERING ONLINE
Covering control, instrumentation, & automation systems worldwide

Understanding PID Control

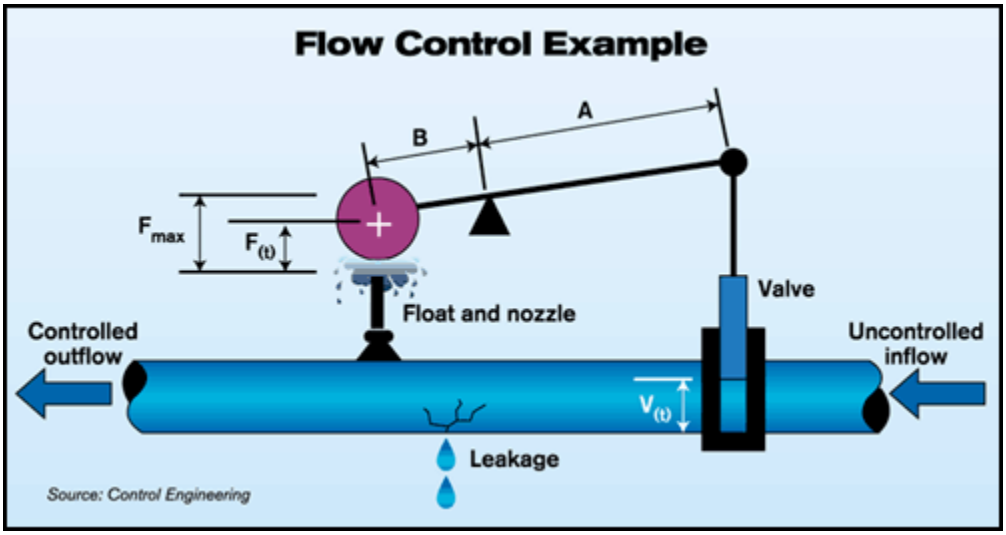
Familiar examples show how and why proportional-integral-derivative controllers behave the way they do.

Keywords:
Process control | Control theory | Controllers | Loop controllers | PID

Vance J. VanDoren, CONTROL ENGINEERING

A feedback controller is designed to generate an output that causes some corrective effort to be applied to a process so as to drive a measurable process variable towards a desired value known as the setpoint. The controller uses an actuator to affect the process and a sensor to measure the results.

Virtually all feedback controllers determine their output by observing the error between the setpoint and a measurement of the process variable. Errors occur when an operator changes the setpoint intentionally or when a disturbance or a load on the process changes the process variable accidentally. The controller's mission is to eliminate the error automatically.



A mechanical flow controller manipulates the valve to maintain the downstream flow rate in spite of the leakage. The size of the valve opening at time **t** is **V(t)**. The flowrate is measured by the vertical position of the float **F(t)**. The gain of the controller is **A/B**. This arrangement would be entirely impractical for a modern flow control application, but a similar principle was actually used in James Watt's original fly-ball governor.

Watt used a float to measure the speed of his steam engine (through a mechanical linkage) and a lever arm to adjust the steam flow to keep the speed constant.

An example

Consider for example, the mechanical flow controller depicted above. A portion of the water flowing through the tube is bled off through the nozzle on the left, driving the spherical float upwards in proportion to the flow rate. If the flowrate slows because of a disturbance such as leakage, the float falls and the valve opens until the desired flow rate is restored.

In this example, the water flowing through the tube is the process, and its flowrate is the process variable that is to be measured and controlled. The lever arm serves as the controller, taking the process variable measured by the float's position and generating an output that moves the valve's piston. Adjusting the length of the piston rod sets the desired flowrate; a longer rod corresponds to a lower setpoint and vice versa.

Suppose that at time **t** the valve opening is **V(t)** inches and the resulting flowrate is sufficient to push the float to a height of **F(t)** inches. This process is said to have a *gain* of $\mathbf{G_p = F(t)/V(t)}$. The gain of a process shows how much the process variable changes when the controller output changes. In this case,

$$\mathbf{F(t) = G_p\, V(t)} \quad [1].$$

Equation [1] is an example of a *process model* that quantifies the relationships between the controller's efforts and its effects on the process variable.

The controller also has a gain $\mathbf{G_c}$, which determines the controller's output at time t according to

$$\mathbf{V(t) = G_c\, (F_{max} - F(t))} \quad [2]$$

The constant $\mathbf{F_{max}}$ is the highest possible float position, achieved when the valve's piston is completely depressed. The geometry of the lever arm shows that $\mathbf{G_c = A/B}$, since the valve's piston will move **A** inches for every **B** inches that the float moves. In other words, the quantity $\mathbf{(F_{max} - F(t))}$ that enters the controller as an input "gains" strength by a factor of **A/B** before it is output to the process as a control effort **V(t)**.

Note that controller equation [2] can also be expressed as

$$\mathbf{V(t) = G_c\, (F_{set} - F(t)) + V_B} \quad [3]$$

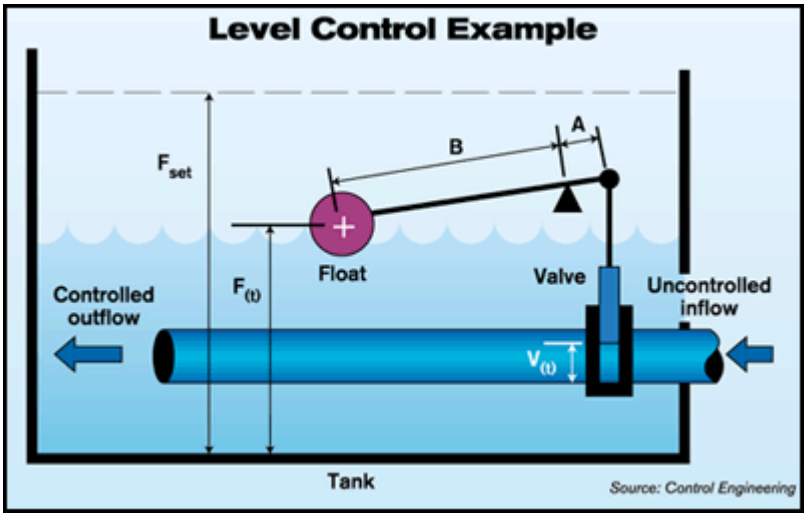
where $\mathbf{F_{set}}$ is the desired float position (achieved when the flow rate equals the setpoint) and $\mathbf{V_B = G_c\, (F_{max} - F_{set})}$ is a constant known as the *bias*. A controller's bias represents the control effort required to maintain the process variable at its setpoint in the absence of a load.

Proportional control

Equation [3] shows how this simple mechanical controller computes its output as a result of the error between the process variable and the setpoint. It is a *proportional* controller because its output changes in proportion to a change in the measured error. The greater the error, the greater the control effort; and as long as the error remains, the controller will continue to try to generate a corrective effort.

So why would a feedback controller have to be any more sophisticated than that? The problem is a proportional controller tends to settle on the *wrong* corrective effort. As a result, it will generally leave a *steady state error* (offset) between the setpoint and the process variable after it has finished responding to a setpoint change or a load.

This phenomenon puzzled early control engineers, but it can be seen in the flow control example above. Suppose the process gain $\mathbf{G_p}$ is 1 so that any valve position **V(t)** will cause an identical float position **F(t)**. Suppose also the controller gain $\mathbf{G_c}$ is 1 and the controller's bias $\mathbf{V_B}$ is 1. If the flow-rate's setpoint requires $\mathbf{F_{set}}$ to be 3 inches and the actual float position is only 2 inches, there will be an error of $\mathbf{(F_{set} - F(t)) =}$ 1 inch. The controller will amplify that 1 inch error to a 2 inch valve opening according to equation [3]. However, since that 2 inch valve opening will in turn cause the float position to remain at 2 inches, the controller will make no further change to its output and the error will remain at 1 inch.



The same mechanical controller now manipulates the valve to shut off the flow once the tank has filled to the desired level $\mathbf{F_{set}}$. The controller' gain of **A / B** has been set much lower, since the float position now spans a much greater range.

Integral control

Even bias-free proportional controllers can cause steady-state errors (try the previous exercise again with $\mathbf{G_p}$ = 1, $\mathbf{G_c}$ = 2, and $\mathbf{V_B}$ = 0). One of the first solutions to overcome this problem was the introduction of *integral* control. An integral controller generates a corrective effort proportional not to the present error, but to the sum of all previous errors.

The level controller depicted above illustrates this point. It is essentially the same float-and-lever mechanism from the flow control example except that it is now surrounded by a tank, and the float no longer hovers over a nozzle but rests on the surface of the water. This arrangement should look familiar to anyone who has inspected the workings of a common household toilet.

As in the first example, the controller uses the valve to control the flowrate of the water. However, its new objective is to refill the tank to a specified level whenever a load (i.e., a flush) empties the tank. The float position **F(t)** still serves as the process variable, but it represents the level of the water in the tank, rather than the water's flowrate. The setpoint $\mathbf{F_{set}}$ is the level at which the tank is full.

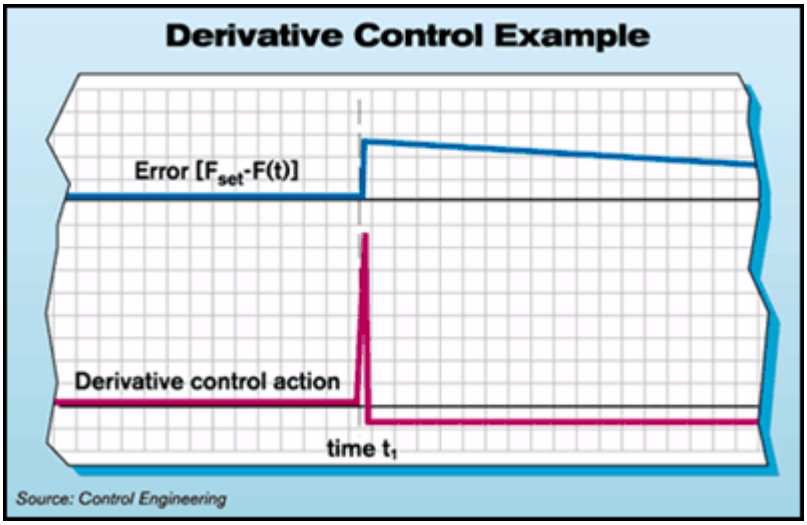
The process model is no longer a simple gain equation like [1], since the water level is proportional to the accumulated volume of water that has passed through the valve. That is

$$F(t) = \int G_p \, V(t) \, dt \quad [4]$$

Equation [4] shows that tank level **F(t)** depends not only on the size of the valve opening **V(t)** but also on how long the valve has been open.

The controller itself is the same, but the addition of the integral action in the process makes the controller more effective. Specifically, a controller that contains its own integral action or acts on a process with inherent integral action will generally *not* permit a steady-state error.

That phenomenon becomes apparent in this example. The water level in the tank will continue to rise until the tank is full and the valve shuts off. On the other hand, if *both* the controller and the process happened to be pure integrators as in equation [4], the tank would overflow because back-to-back integrators in a closed loop cause the steady-state error to grow without bound!



The blue trace on this strip chart shows the error between the process variable **F(t)** and its desired value $\mathbf{F_{set}}$. The derivative control action in red is the time derivative of this difference. Derivative control action is zero when the error is constant and spikes dramatically when the error changes abruptly.

Derivative control

Proportional (P) and integral (I) controllers still weren't good enough for early control engineers. Combining the two operations into a single "PI" controller helped, but in many cases a PI controller still takes too long to compensate for a load or a setpoint change. Improved performance was the impetus behind the development of the *derivative* controller (D) that generates a control action proportional to the time derivative of the error signal.

The basic idea of derivative control is to generate one large corrective effort immediately after a load change in order to begin eliminating the error as quickly as possible. The strip chart in the derivative control example shows how a derivative controller achieves this. At time t1, the error, shown in blue, has increased abruptly because a load on the process has dramatically changed the process variable (such as when the toilet is flushed in the level control example).

The derivative of the error signal is shown in red. Note the spike at time $\mathbf{t_1}$. This happens because the derivative of a rapidly increasing step-like function is itself an even more rapidly increasing impulse function. However, since the error signal is much more level after time $\mathbf{t_1}$, the derivative of the error returns to roughly zero thereafter.

In many cases, adding this "kick" to the controller's output solves the performance problem nicely. The derivative action doesn't produce a particularly precise corrective effort, but it generally gets the process moving in the right direction much faster than a PI controller would.

Combined PID control

Fortunately, the proportional and integral actions of a full "PID" controller tend to make up for the derivative action's lack of finesse. After the initial kick has passed, derivative action generally dies out while the integral and proportional actions take over to eliminate the remaining error with more precise corrective efforts. As it happens, derivative-only controllers are very difficult to implement anyway.

On the other hand, the addition of integral and derivative action to a proportional-only controller has several potential drawbacks. The most serious of these is the possibility of *closed-loop instability* (see "Controllers must balance performance with closed-loop stability," *Control Engineering*, May 2000). If the integral action is too aggressive, the controller may over-correct for an error and create a new one of even greater magnitude in the opposite direction. When that happens, the controller will eventually start driving its output back and forth between fully on and fully off, often described as *hunting*. Proportional-only controllers are much less likely to cause hunting, even with relatively high gains.

Another problem with the PID controller is its complexity. Although the basic operations of its three actions are simple enough when taken individually, predicting just exactly how well they will work

together for a particular application can be difficult. The stability issue is a prime example. Whereas adding integral action to a proportional-only controller can *cause* closed-loop instability, adding proportional action to an integral-only controller can *prevent* it.

PID in action

Revisiting the Flow control example, suppose an electronic PID controller capable of generating integral and derivative action as well as proportional control has replaced the simple lever arm controller. Suppose too a viscous slurry has replaced the water so the flow rate changes gradually when the valve is opened or closed.

Since this viscous process tends to respond slowly to the controller's efforts—when the process variable suddenly differs from the setpoint because of a load or setpoint change—the controller's immediate reaction will be determined primarily by the derivative action, as shown on the Derivative control example. This causes the controller to initiate a burst of corrective efforts the instant the error moves away from zero. The change in the process variable will also initiate the proportional action that keeps the controller's output going until the error is eliminated.

After a while, the integral action will begin to contribute to the controller's output as the error accumulates over time. In fact, the integral action will eventually dominate the controller's output, since the error decreases so slowly in a sluggish process. Even after the error has been eliminated, the controller will continue to generate an output based on the accumulation of errors remaining in the controller's integrator. The process variable may then *overshoot* the setpoint, causing an error in the opposite direction, or perhaps closed-loop instability.

If the integral action is not too aggressive, this subsequent error will be smaller than the original, and the integral action will begin to diminish as negative errors are added to the history of positive ones. This whole operation may then repeat several times until both the error and the accumulated error are eliminated. Meanwhile, the derivative term will continue to add its share to the controller output based on the derivative of the oscillating error signal. The proportional action also will come and go as the error waxes and wanes.

Now replace the viscous slurry with water, causing the process to respond quickly to the controller's output changes. The integral action will not play as dominant a role in the controller's output, since the errors will be short lived. On the other hand, the derivative action will tend to be larger because the error changes rapidly when the process is highly responsive.

Clearly the possible effects of a PID controller are as varied as the processes to which they are applied. A PID controller can fulfill its mission to eliminate errors, but only if properly configured for each application.

For more information on control loop analysis and tuning, visit www.controleng.com.

Consulting Editor, Vance J. VanDoren, Ph.D., P.E.,
is president of VanDoren Industries, West Lafayette, Ind.

Comments? E-mail controleng@msn.com

Control Engineering June 2000

CONTROL ENGINEERING ONLINE
Covering control, instrumentation, & automation systems worldwide

Examining the Fundamentals of PID Control

Algorithms take the simple feedback controller one step further

Vance J. VanDoren, CONTROL ENGINEERING

This tutorial presents an overview of how and why PID controllers work. It is the first in a four part series on the fundamental concepts of modern control theory.

PROCESS CONTROL TUTORIAL

- Process control and instrumentation
- PID (proportional/integral/derivative)
- Process controllers
- Open architecture
- Programmable logic controllers (PLCs)

A feedback controller is designed to generate an "output" that causes some corrective effort to be applied to a "process" so as to drive a measurable "process variable" towards a desired value known as the "setpoint." Figure 1 shows a typical feedback control loop, with blocks representing the dynamic elements of the system and arrows representing the flow of information, generally in the form of electrical signals.

Virtually all feedback controllers determine their output by observing the *"error"* between the setpoint and the actual process variable measurement. A home thermostat, for example, uses the air conditioning system to correct the temperature in a process comprised of a room and the air inside. It sends an electrical signal (an output) to turn on the air conditioner when the error between the actual temperature (the process variable) and the desired temperature (the setpoint) is too high.

A look at PID control

A proportional-integral-derivative or PID controller performs much the same function as the thermostat, but with a more elaborate algorithm for determining its output. It looks at the current value of the error, the integral of the error over a recent time interval, and the current derivative of the error signal to determine not only how much of a correction to apply, but for how long. Those three quantities are each multiplied by a "tuning constant" and added together to produce the current controller output CO(t) , thusly:

$$CO(t) = P \cdot e(t) + I \cdot \left(\int_0^t e(\tau)\, d\tau \right) + D \cdot \left(\frac{d}{dt} e(t) \right) \quad \text{[eq. 1]}$$

In equation [1], P is the "proportional" tuning constant, I is the "integral" tuning constant, D is the "derivative" tuning constant, and e(t) is the error between the setpoint SP(t) and the process variable PV(t) at time t.

$$e(t) = SP(t) - PV(t) \quad \text{[eq. 2]}$$

If the current error is large, has been sustained for some time, or is changing rapidly, the controller will attempt to make a large correction by generating a large output. Conversely, if the process variable has matched the setpoint for some time, the controller will leave well enough alone.

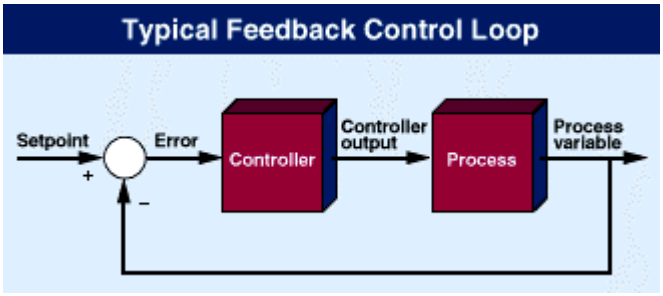


Fig. 1: Feedback controllers determine their output by observing the error between setpoint and the actual process variable measurement.

Tuning the controller

Conceptually, that's all there is to a PID controller. The tricky part is "tuning" it; i.e., setting the P, I, and D tuning constants so that the weighted sum of the proportional, integral, and derivative terms produces a controller output that steadily drives the process variable in the direction required to eliminate the error.

The brute force solution to this problem would be to generate the largest possible output by using the largest possible tuning constants. A controller thus tuned would amplify every error and initiate extremely aggressive efforts to eliminate even the slightest discrepancy between the setpoint and the process variable. However, an overly aggressive controller can actually make matters worse by driving the process variable past the setpoint as it attempts to correct a recent error. In the worst case, the process variable will end up even further away from the setpoint than before.

On the other hand, a PID controller that is tuned too conservatively may not be able to eliminate one error before the next one appears. A well-tuned controller performs at a level somewhere between those two extremes. It works aggressively to eliminate an error quickly, but without overdoing it.

How to best tune a PID controller depends upon how the process responds to the controller's corrective efforts. Processes that react instantaneously and predictably don't really require feedback at all. A car's headlights, for example, come on as soon as the driver hits the switch. No subsequent corrections are required to achieve the desired illumination.

On the other hand, the car's cruise controller cannot accelerate the car to the desired cruising speed as quickly. Because of friction and the car's inertia, there is always a delay between the time that the cruise controller activates the accelerator and the time that the car's speed reaches the setpoint (see Fig. 2). A PID controller must be tuned to account for such "lags."

PID in action

Consider a sluggish process with a relatively long lag—accelerating an overloaded car with an undersized engine, for example. Such a process tends to respond slowly to the controller's efforts. If such errors are introduced abruptly (as when the setpoint is changed), the controller's initial reaction will be determined primarily by the actions of the derivative term in equation 1. This will cause the controller to initiate a burst of corrective effort the instant the error changes from zero. The proportional term will then come into play to keep the controller's output going until the error is eliminated.

After a while, the integral term will also begin to contribute to the controller's output as the error accumulates over time. In fact, the integral term will eventually come to dominate the output signal because the error decreases so slowly in a sluggish process. Even after the error has been eliminated, the controller will continue to generate an output based on the history of errors that have been accumulating in the controller's integrator. The process variable may then "overshoot" the setpoint, causing an error in the opposite direction.

If the integral tuning constant is not too large, this subsequent error will be smaller than the original, and the integral term will begin to diminish as negative errors are added to the history of positive ones. This whole operation may then repeat several times until both the error and the accumulated error are eliminated. Meanwhile, the derivative term will continue to add its share to

the controller output based on the derivative of the oscillating error signal. The proportional term will also come and go as the error waxes and wanes.

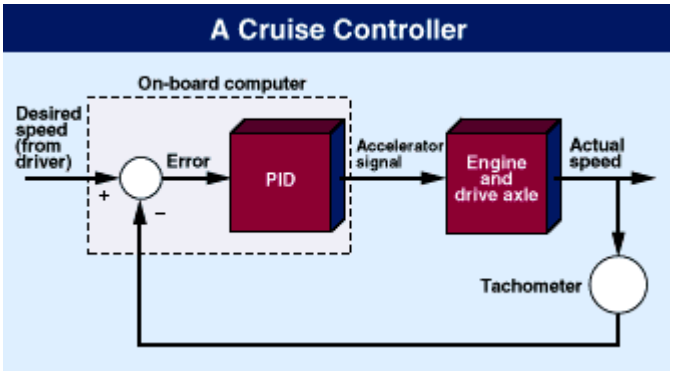


Fig. 2: A familiar real-world example of feedback control can be found in the "cruise control" feature common in many automobiles.

Short lag time

Now suppose the process has very little lag so that it responds quickly to the controller's efforts. The integral term in equation 1 will not play as dominant a role in the controller's output since the errors will be so short-lived. On the other hand, the derivative term will tend to be larger, since the error changes rapidly in the absence of long lags.

Clearly, the relative importance of each term in the controller's output depends on the behavior of the controlled process. Determining the best mix suitable for a particular application is the essence of controller tuning. For the sluggish process, a large value for the derivative tuning constant D might be advisable in order to accelerate the controller's reaction to a setpoint change. For the fast-acting process, however, an equally large value for D might cause the controller's output to fluctuate wildly, as every change in the error is amplified by the controller's derivative action.

Tuning techniques

There are three schools of thought on how to select the values of P, I, and D required to achieve an acceptable level of performance for the controller. The first method is simple trial and error—tweak the tuning parameters and watch the controller handle the next error. If it can eliminate the error in a timely fashion, quit. If it proves to be too conservative or too aggressive, increase or decrease one or more of the tuning constants. Experienced control engineers seem to know just how much proportional, integral, and derivative action to add or subtract in order to correct the performance of a poorly tuned controller.

Unfortunately, intuitive tuning procedures can be difficult to develop because a change in one tuning constant tends to affect the performance of all three terms in the controller's output. For example, turning down the integral action reduces overshoot. This in turn slows the rate of change of the error and thus reduces the derivative action as well.

Using math models

The analytical approach to the tuning problem, which is the second method, is more rigorous. It involves a mathematical "model" of the process that relates the value of the process variable at time t to the current rate of change of the process variable and a history of the controller's output. For example,

$$PV(t) = K \cdot CO(t-d) - T \cdot \left(\frac{d}{dt} PV(t)\right)$$ [eq. 3]

This particular model describes a process with a "gain" of K, a "time constant" of T, and a "deadtime" of d. The process gain represents the magnitude of the controller's effect on the process variable. A large value of K corresponds to a process that amplifies small control efforts into large changes in the process variable.

The time constant in equation 3 represents the severity of the process lag. A large value of T corresponds to a long lag in a sluggish process. The deadtime d represents another kind of delay

present in many processes, where the "sensor" used to measure the process variable is located some distance from the "actuator" used to implement the controller's corrective efforts. The time required for the actuator's effects to reach the sensor is the deadtime. During that interval, the process variable does not respond at all to the actuator's activity. Only after the deadtime has elapsed does the lag time begin (see Fig. 3).

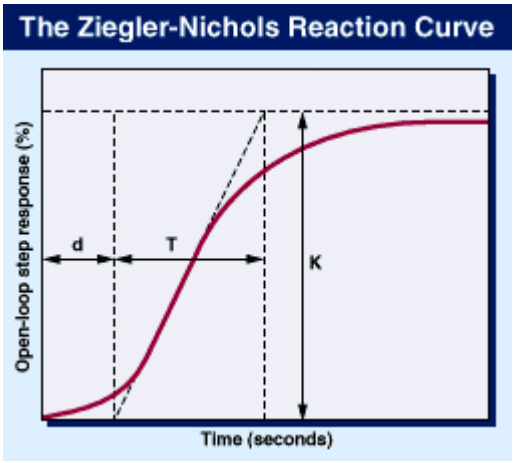


Fig. 3: Examination of the Ziegler-Nichols reaction curve reveals the response of a simple lag process to a unit step change in the controller output.

In the thermostat example above, the air conditioner is the actuator and the thermostat's onboard thermocouple is the sensor. If there is any ductwork between the air conditioner and the thermostat, there will be a deadtime while each slug of cool air travels down the duct. The room temperature will not begin to drop until the first slug of cool air emerges from the duct.

There are other characteristics of process behavior that can be factored into a process model, but equation 3 is one of the simplest and most widely used. It applies to any process with a process variable that changes in proportion to its current value. For example, a car of mass m accelerates when its cruise control calls for the engine to apply a force F_e to the drive axle. However, that acceleration a(t) is opposed by frictional forces F_f that are proportional to the car's current velocity v(t) by a factor of K_f. If the force applied by the engine is proportional to the controller's output by a factor of K_e, then applying Newton's second law to the process gives:

$$F_e - F_f = m \cdot a(t) \quad \text{[eq. 4]}$$

$$K_e \cdot CO(t) - K_f \cdot v(t) = m \cdot \left(\frac{d}{dt} v(t)\right) \quad \text{[eq. 5]}$$

$$v(t) = K \cdot CO(t) - T \cdot \frac{d}{dt} v(t) \quad \text{[eq. 6]}$$

"The process variable is v(t) , the process gain is $K = K_e/K_f$ and the process time constant is $T = m/K_f$. In this example no deadtime exists, since the speed of the car begins to change as soon as the cruise controller activates the accelerator.

If a model like equation 3 can be defined for a process, its behavior can be quantified by analyzing the model's parameters. In equation 6, for example, the values of K and T (computed from K_e, K_f, and m) determine how the velocity of the car will change in response to any control effort. A model's parameters in turn dictate the tuning constants required to modify the behavior of the process with a feedback controller.

Literally hundreds of analytical techniques can translate model parameters into tuning constants. Each approach uses a different model, different controller objectives, and different mathematical tools. Several examples of analytical tuning will be explored in future installments of this series.

The Ziegler-Nichols approach

The third approach to the tuning problem is something of a compromise between purely self-teaching trial-and-error techniques and the more rigorous analytical techniques. It was originally proposed in 1942 by John G. Ziegler and Nathaniel B. Nichols, and remains popular today because of its simplicity and its applicability to any process governed by a model in the form of equation 3. Through trial-and-error experiments, Ziegler and Nichols created a set of "tuning rules" that translate the parameters of equation 3 into values for P, I, and D, giving generally acceptable controller performance. In particular,

$$P = \frac{1.2 \cdot T}{K \cdot d}$$

$$I = \frac{0.6 \cdot T}{K \cdot d^2} \quad \text{[eqs. 7]}$$

$$D = \frac{0.6 \cdot T}{K}$$

Ziegler and Nichols also came up with a practical method for estimating the values of K, T, and d experimentally. With the controller in manual mode (no feedback), they induced a step change in the controller's output, then analyzed the process reaction graphically (see Fig. 3). They concluded that the process gain K can be approximated by dividing the net change of the process variable by the size of the step change generated by the controller. They estimated the deadtime d from the interval between the controller's step change and the beginning of a line drawn tangent to the reaction curve at its steepest point. They also used the inverse slope of that line to estimate the time constant T.

Other tuning rules have since been developed for more complex models and for other controller performance objectives. Several of these, as well as a reprint of Ziegler and Nichols' 1942 paper, can be found in "Reference Guide to PID Tuning—A collection of reprinted articles of PID tuning techniques" published by *Control Engineering* in 1991).

Control Engineering – February 1996