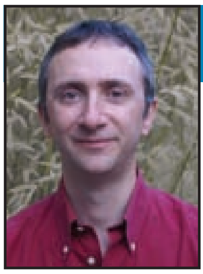


by Robert Lacoste

PID Control Without Math

If you need to design an optimized control loop for a hardware control problem, consider trying a proportional integral derivative (PID) controller. In this article, Robert explains that PID regulations are simple to code, wire, and tune to find the results you're looking for.

Welcome back to the Darker Side. The vast majority of real-world systems are based around feedback loops, which are used to manipulate the inputs to a system to obtain a desired effect on its outputs in a controlled way. Your DVD player uses a feedback loop to drive its spinning motor for a precise rotation speed. Your mobile phone has a feedback loop to adjust its transmit power to the required level. And, of course, your car has plenty of feedback loops (not only in the cruise control module). Systems incorporating feedback work even if the relationship between the desired value (e.g., the spinning speed of a motor) and the controlled one (in that case, the current applied to the motor) is not straightforward. Delay effects, inertia, and nonlinearities make life more interesting, and so do external conditions. For example, a cruise control device needs to apply more torque on the motor if you are climbing a hill. It should keep overshoots as small as possible to avoid getting a fine after you get to the top of a hill.

On the theoretical side, these problems have been well-known for years and are covered by the Control System theory. The theory explains how to design an optimized control loop for a given problem, at least if the problem is well formalized. The theory—which started with the works of James Clerk Maxwell in 1868, followed by a lot of inspired mathematicians, such as Alexander Lyapunov and Harry Nyquist, to name a few—is not simple. The first pages of control system books usually start with mathematical notions like pole placement, Z-transforms, and sampling theorems, which nonspecialists may find difficult to deal with, even if good books have adopted a more engineer-oriented approach (e.g., Tim Wescott's *Applied Control Theory for Embedded Systems*, Newnes, 2006). Should you give up? No. Fortunately, some classical control algorithms are applicable to a lot of problems. More importantly, for design guys like us, they are easy to implement in firmware or hardware. The most usual is the ubiquitous proportional integral derivative (PID) control, which has one interesting characteristic: it is

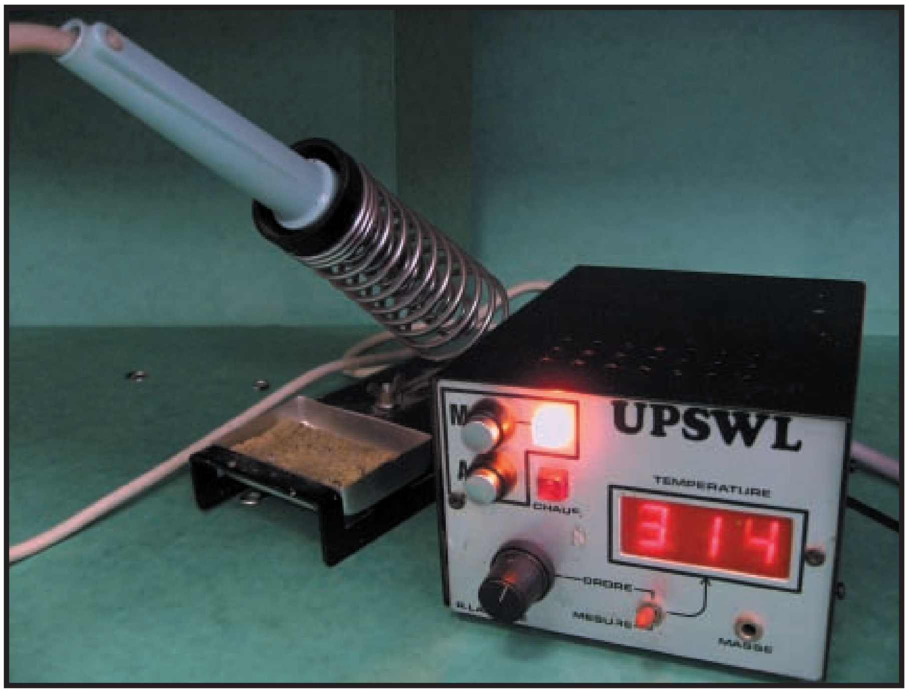


Photo 1—This is the power supply and controller I built in 1987 and its Weller VR20 soldering iron. Its first basic "threshold comparator" design was working, but it had a ±3°C oscillation around the preset temperature. A simple control improvement reduced the oscillation to ±1°C.

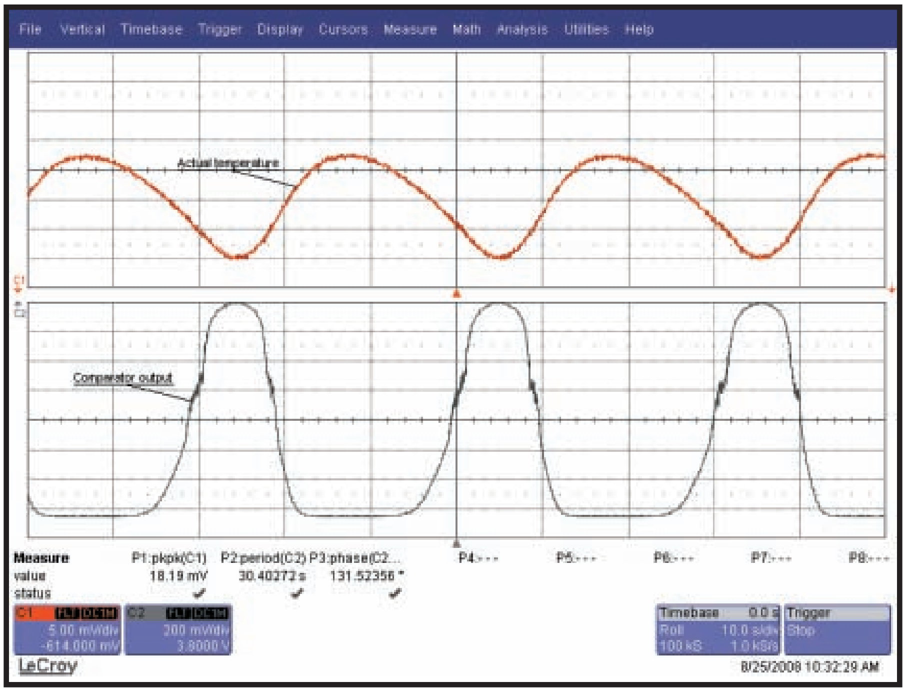

File Vertical Timebase Trigger Display Cursors Measure Math Analysis Utilities Help
Actual temperature
Comparator output
Measure P1:pkpk(C1) P2:period(C2) P3:phase(C2... P4:--- P5:--- P6:--- P7:--- P8:---
value 18.19 mV 30.40272 s 131.52356 °
status
C1 5.00 mV/div -614.000 mV
C2 200 mV/div 3.8000 V
Timebase 0.0 s Roll 10.0 s/div 100 kS 1.0 kS/s
Trigger Stop
LeCroy
8/25/2008 10:32:29 AM

Photo 2—This plot was taken on my new LeCroy WaveRunner 6050A oscilloscope connected to the iron controller in its initial version. The top channel is the sensor measurement. The bottom is the command sent to the output TRIAC. In this first design, I used a simple comparator. Both curves are exactly in phase; but as a consequence, the oscillation is high due to thermal inertia: 19 mV_{PP} corresponding to 6°C. Heating cycles are roughly 30 s each.

tuned by only three or four parameters. As you will see, the parameters can be determined by empirical methods, without even knowing the exact behavior of the controlled process. Don't get me wrong. I am not saying that a PID will solve any control problem, especially if the system has multiple inputs and outputs. But you may want to give it a try before digging into more complex solutions. In any case, it's a must-have for every engineer. So, let's go with PID!

A BASIC CASE

The most basic example of a feedback loop is a temperature controller used to drive a heater or cooler to get a precise temperature (or temperature profile) on a temperature sensor on a device. Regular readers may remember my article about a home-made reflow oven controller based on a firmware PID loop ("Easy Reflow: Build an SMT Reflow Oven Controller," *Circuit Cellar* 168, 2004). This month, I will take an even simpler example. Twenty years ago, I bought a Weller VR20 soldering iron, which had a built-in resistive temperature sensor. At that time, I couldn't afford the corresponding power supply, so of course I built my own (see Photo 1). It was easy. I used a 24-V transformer, a potentiometer to tune the temperature, a digital voltmeter

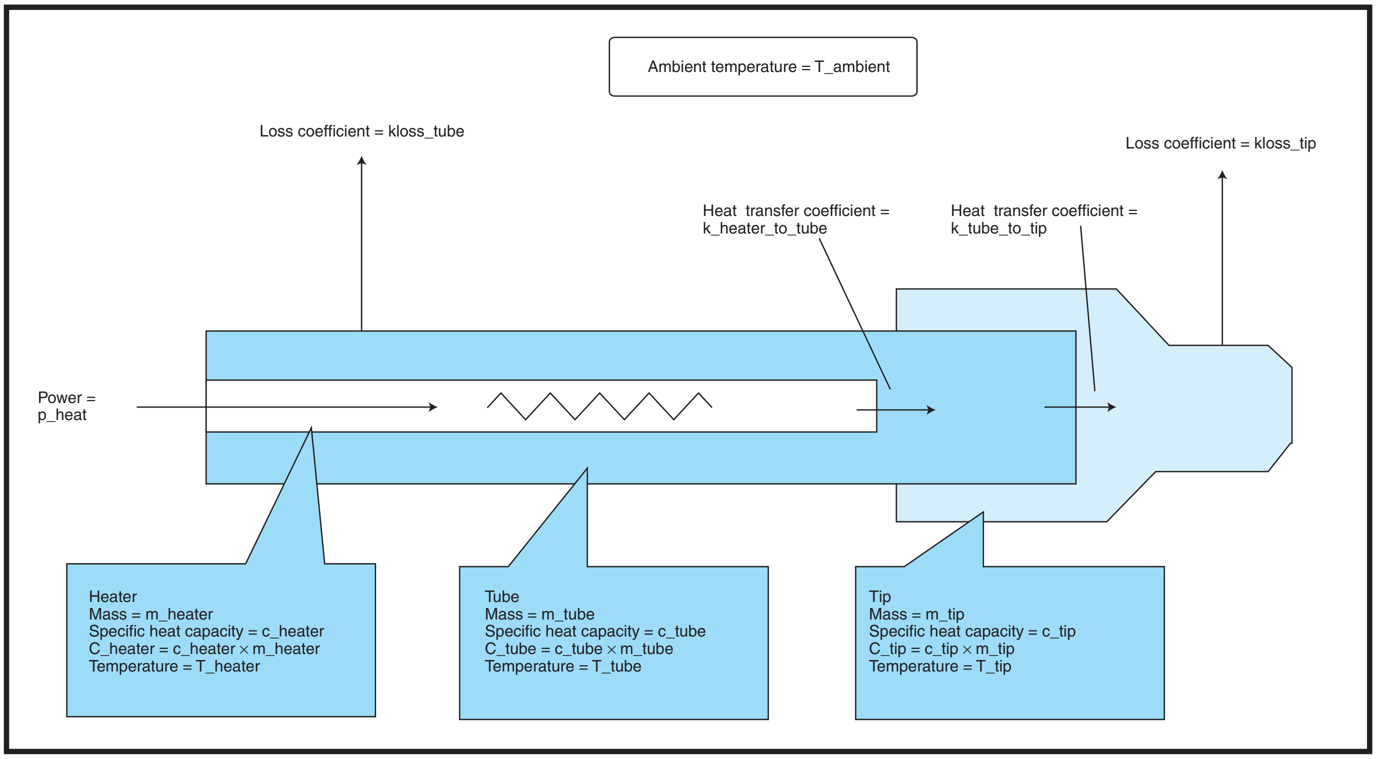

Ambient temperature = T_ambient
Loss coefficient = kloss_tube
Loss coefficient = kloss_tip
Heat transfer coefficient = k_heater_to_tube
Heat transfer coefficient = k_tube_to_tip
Power = p_heat
Heater
Mass = m_heater
Specific heat capacity = c_heater
C_heater = c_heater × m_heater
Temperature = T_heater
Tube
Mass = m_tube
Specific heat capacity = c_tube
C_tube = c_tube × m_tube
Temperature = T_tube
Tip
Mass = m_tip
Specific heat capacity = c_tip
C_tip = c_tip × m_tip
Temperature = T_tip

Figure 1—For this article, I developed Scilab code based on this thermal latency model. The heater is heating a tube, which heats the iron tip. I assumed the respective temperatures of the heater, tube, and tip to be homogeneous. The heat power transferred from heater to tube is supposed to be a constant factor multiplied by the temperature difference between both elements. The same goes for transfer from tube to tip or to ambient air. This simplistic model enabled me to get reasonably realistic simulation results.

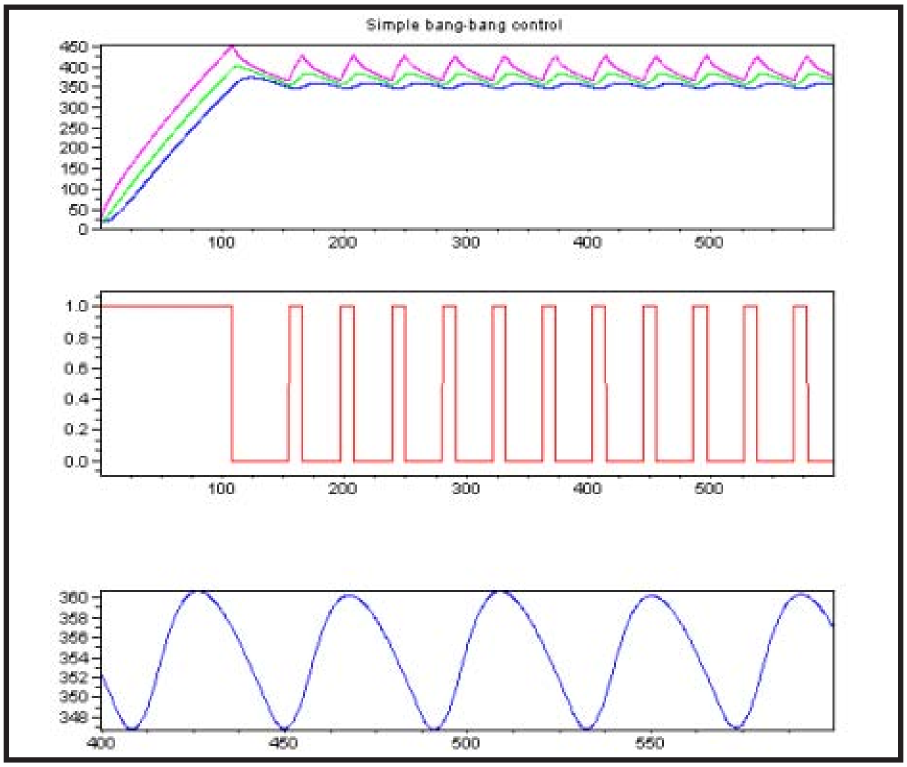


Figure 2—This is the simulation result for a simple "bang bang comparator" system like the one used in the initial version of my iron controller. The top curves are respective heater, tube, and iron temperatures. The middle plot is the power applied on the heater. The bottom curve is a zoom on the tip temperature, showing a 12°C swing around the 350°C target with the parameters I have used.

(Remember the old CA3161/CA3162 three-digit voltmeter chipset?), and a crude "control system." I used an LM311 comparator to drive an output TRIAC on and off, whether the measured temperature was above or below the threshold. For control system specialists, this is the most basic form of a "bang-bang control" algorithm—and it is so for obvious reasons. (Just imagine yourself driving your car with only two settings: throttle fully open or brakes fully engaged.) It was working, but the temperature regulation was oscillating around 3°C above and below the preset value. That corresponded to an 18-mV oscillation on the sensor measurement (see Photo 2).

Why? Just because the heater is not in direct contact with the sensor and because the assembly does not weigh 0 g. The heat takes some time to go from the heater to the tip, so if you wait for the sensor to reach the target temperature before switching the power off, it is already too late. The heat will continue to flow from the heater to the tip and you will get a significant overshoot, which ultimately gives oscillation around the threshold. It is easy to model this behavior (see Figure 1). I have even coded a Scilab simulation of this system (see Figure 2). Close to the behavior of my old iron controller, isn't it?

Just as a reminder: Scilab is an open-source Matlab-like tool with great simulation toolboxes. The Scilab simulation sources I coded for this article are posted on the *Circuit Cellar* FTP site. Don't hesitate to read them because I've commented them heavily.

As I will show you at the end of this article, I improved this iron controller a couple of years after its assembly and got a drastically improved regulation. On the hardware side, I do not recommend that you duplicate this design because it is based on obsolete technology. But it is a perfect example to introduce PID controls.

PROPORTIONAL?

How do you improve the thermal regulation? Using a full on/off drive on the heater is simple, but it is not the best solution. Why should you use 80 W of heating power if you are close to the target? It should be reduced more and more as you approach. The simplest

a) P control :

```
Error = actual - target
Command = Kp . Error
Command = Limit(Command,Commandmin,Commandmax)
Pheater = Pmax . Command
```

b) PD control (theoretical) :

```
Error = actual - target
Command = Kp . Error + Kd . d(Error)/dt
```

c) PD control (implementation) :

```
Previous error = Error
Error = actual - target
Command = Kp . (actual - target) + Kd/timestep . (Error - Previous error)
```

d) PID control (theoretical) :

```
Error = actual - target
Command = Kp . Error + Kd . d(Error)/dt + Ki . Integral (Error.dt)
```

e) PID control (implementation) :

```
Previous error = Error
Error = actual - target
Integral = Integral + error
Command = Kp . (actual - target) + Kd/timestep . (Error - Previous error)
        + Ki . Timestep . Integral
```

Figure 3—A P control simply calculates the error between the target and measured values, and uses it to drive the output. This is better than a simple threshold comparison but may induce overshoots. A PD control damps the system with the addition of a derivative term, which is usually calculated as the difference between successive errors. Lastly, the full PID control also adds an integral term, which is the sum of the error over time, in order to avoid any systematic error.

way to do so would be to calculate the heating power as proportional to the distance to the target. This is where the "P" of PID comes from. Rather than just comparing the measured value to a threshold and switching the output on and off, a proportional controller manages it more smoothly. The algorithm calculates the error, which is the target value minus the measurement, multiplies it by a given gain (usually denoted Kp), and uses the result of this calculation to drive the output after limiting it to reasonable values. Refer to the pseudocode algorithm in Figure 3a.

You may criticize this approach as a linear power supply stage would be needed to implement such a proportional control, with its power inefficiency and added complexity, and you would be right. But for a thermal controller, nothing forbids you from using the "Pheater" value to directly drive a high-speed PWM output rather than a DC power generator. The heating power will be proportional to the mean PWM value, which is the desired command. What improvement could you get with such a proportional control system? I did the simulation in Figure 2. The results show that the temperature oscillation is reduced from 12° down to 7°C, at least under the hypothesis of the simulation (see Figure 4).

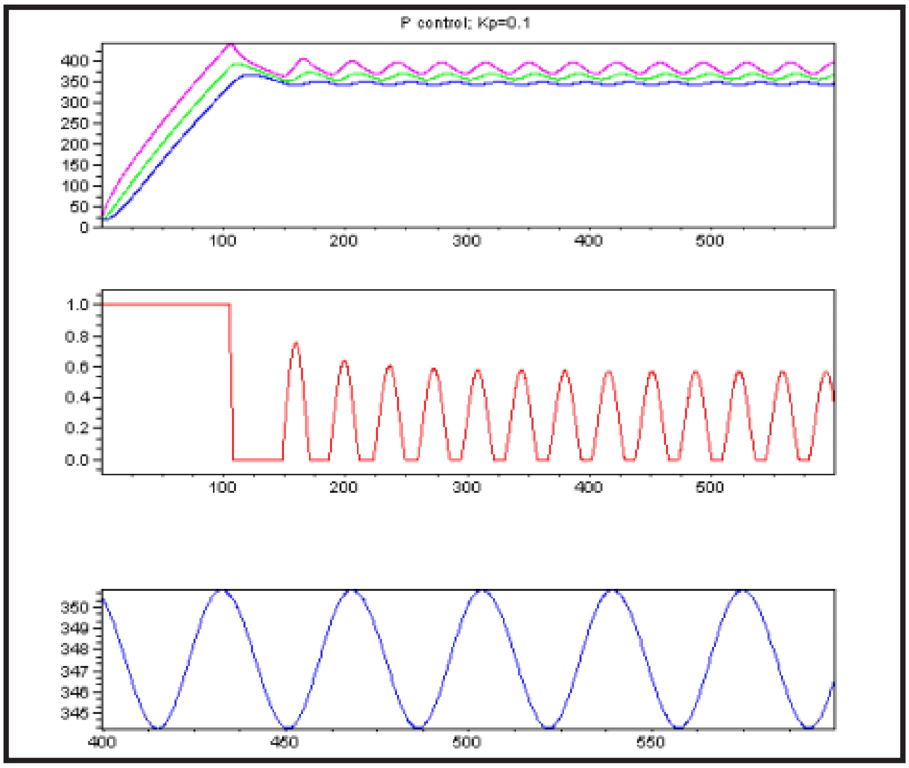


Figure 4—A proportional control enables you to generate an analog output value. It's low if the current measurement is close to the preset target but higher if the target is far (medium curve). The overall temperature oscillation is reduced from 12° down to 7°C, as compared to the "bang-bang" control in Figure 2.

The optimal value of Kp must be determined for each application because it is dependent on the system parameters and your preferences. Figure 5 illustrates the system's behavior with different Kp values. My experience tells me that it is to start with low Kp values and increase it up to a point where oscillations and ringing starts to be a little too high.

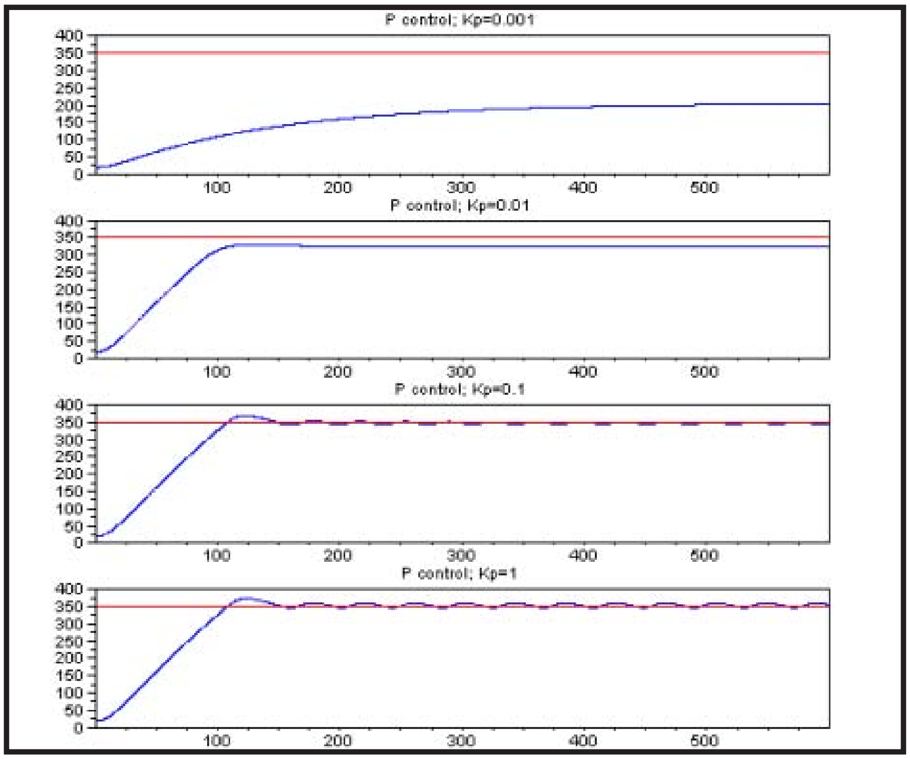


Figure 5—This simulation shows you the behavior of the simulated system with a proportional control and different values of the Kp gain. With small Kp values (top plot), the actual temperature takes a long time to reach the target, and may not reach it. The loop is "soft." If you increase the Kp gain, the regulation becomes quicker and quicker up to a point where overshoot starts to appear. The optimum value is often just after the appearance of oscillations (Kp = 0.1 in this instance).

DERIVATIVE HELPS!

A proportional controller uses only the current measurement to determine the output value. It doesn't have memory or forecasting to improve the regulation. When you press the brake as you park your car in your garage, you don't apply only a pressure proportional to the distance between your car and the back wall. You also use other information, such as the speed at which your car is approaching the wall, and this makes sense. As you may remember from your youth, speed is the derivative of

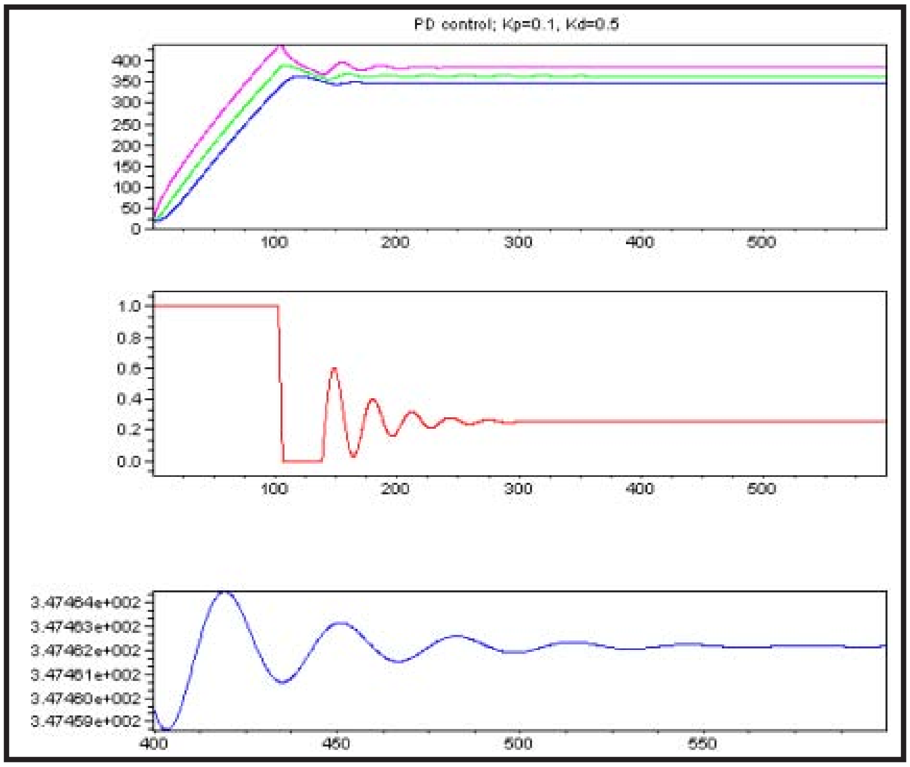


Figure 6—The addition of a derivative parameter gives a PD control loop. Thanks to the derivative, the oscillations are damped out and the system reaches a steady state, at least in this simulation. However, note that the stabilized value is not exactly the preset 350°C. It's a little lower (347.4°C) due to heat dissipation in the ambient air.

the distance over time. A proportional-derivative control (PD) adds the derivative of the error to the calculation with another gain noted as Kd (see Figure 3b).

For discrete time systems (e.g., microcontroller-based), the derivative of the error can be approximated as the current error minus the previous one, divided by the duration of the calculation time step. The result is the algorithm in Figure 3c.

How does it work? If the error is increasing quickly, then the d(Error)/dt term is high. So, for the same absolute error, the command applied on the output is higher. This enables you to return to the target quicker. On the contrary, if the error is quickly reducing, then the d(Error)/dt term is negative. This reduces the power applied on the output, which reduces risks of overshoots. For simple systems, like the one I simulated, a PD algorithm provides an impressive improvement, even if the situation is not so easy in real life (see Figure 6).

How do you tune the Kd coefficient? As illustrated in Figure 7, the system is more and more damped when Kd is higher. If Kd is null, then we are back to a simple proportional control. An increased Kd reduces oscillations and overshoots but also increases the time needed to reach a new setpoint. If you increase Kd even more, the system starts to be soft and often too soft. This is why I previously said that the Kp proportional parameter should usually be set a little higher than the value which gave the first oscillations because Kd will help reduce them. So, in a nutshell, with a PD regulation, you can first set Kd and Kp to 0, increase Kp until small oscillations appear on the output, and then increase Kd until there is enough damping but not more. The resulting parameter set is often close to optimal for common systems.

INTEGRAL, FOR FULL PID

Take another look at the PD control simulation in Figure 6 and you will see that the system reaches a steady state, which is not exactly the target value. On this simulation, the long-term sensor value is 347°C, 3° lower than the preset 350°C target. This is because the system is dissipative, meaning that some energy flows

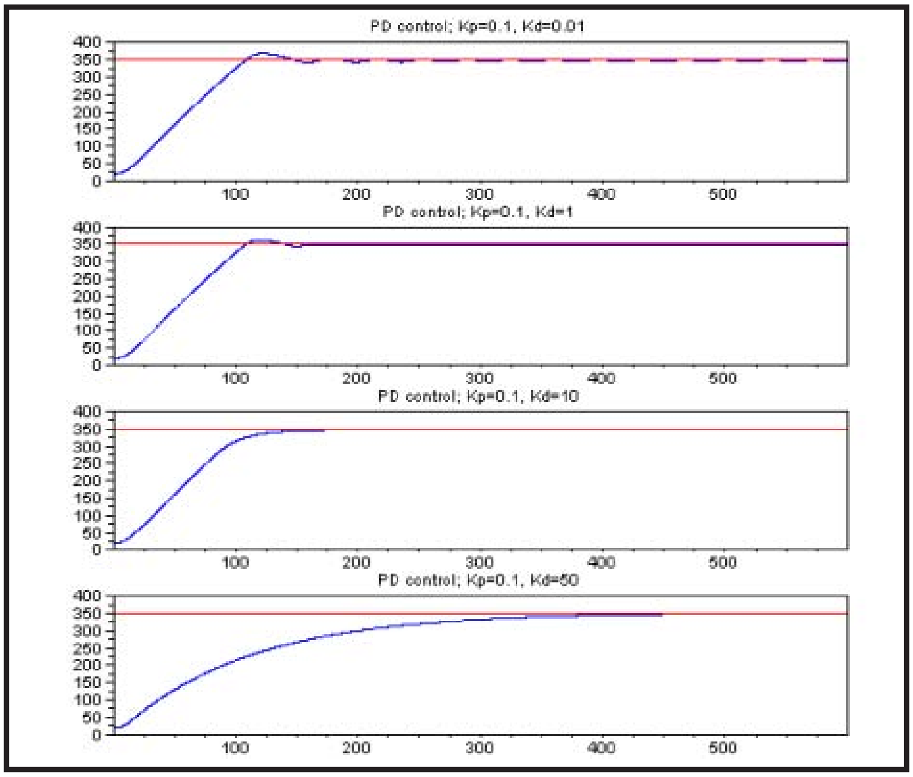


Figure 7—This is what usually happens when the derivative gain Kd is increased. The oscillations are damped more and more. At a given point, the system is no longer oscillating but starts to be long to reach its target. The optimal setting depends on the system constraints.

from the iron to the ambient air. At equilibrium, the heat loss to the ambient air is exactly equal to the 3°C error multiplied by Kd. The system is stable but will never reach its target setpoint: the error stays constant so its derivative is null and Kd is useless.

"PID" appears in this article's title. I already covered "PD" feedback loops, so we need to add the "I" in order to avoid such long-term errors. Not only do you need to take into account the error and its derivative over time, but also its integral over time. If there is a constant error, this integral will be higher and higher over time. If you add it to the command through another Ki gain, then the equilibrium state will be forced to be exactly at the setpoint value (see Figure 3d). The result, for time-sampled systems, is the algorithm in Figure 3e.

This works and it is the final form of the PID control algorithm (see Figure 8). However, please take care. The integral term must be manipulated with caution. Contrary to the Kd and Kp gains (at least with reasonable values), an improper Ki gain can make the system unstable. Moreover, the effect of Ki is usually opposed to Kd: a higher Ki gives higher oscillations and longer stabilization time. Practically speaking, it is best to always start

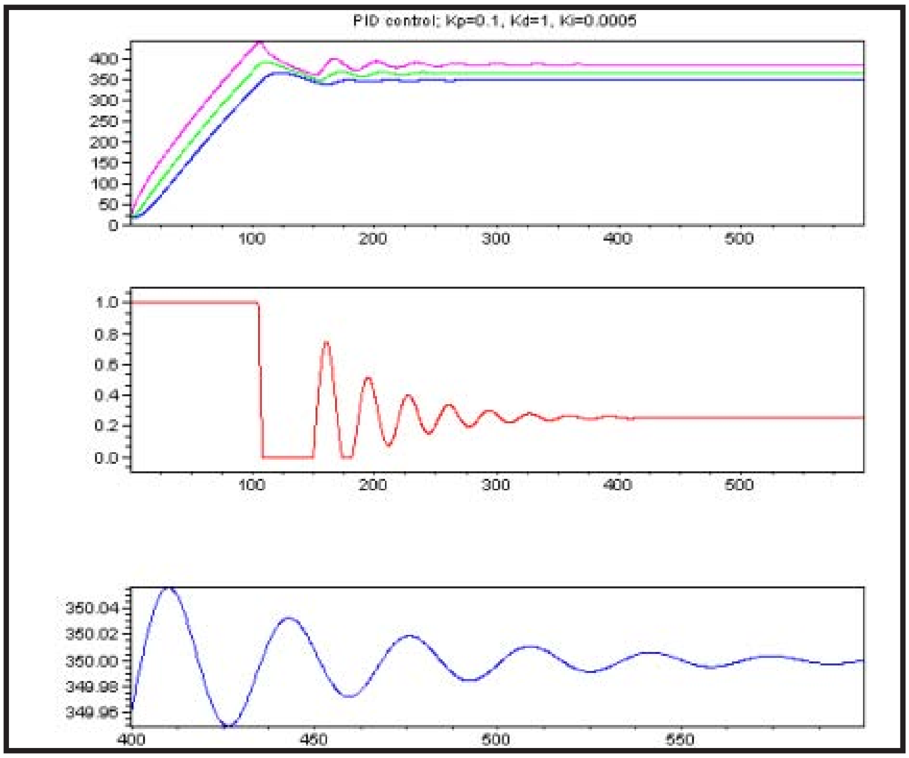


Figure 8—The addition of an integral term enables you to ensure that the steady point is equal to the preset 350°C target. However, note that the oscillations and stabilization time are a little higher than with a simpler PD control loop as on Figure 6.

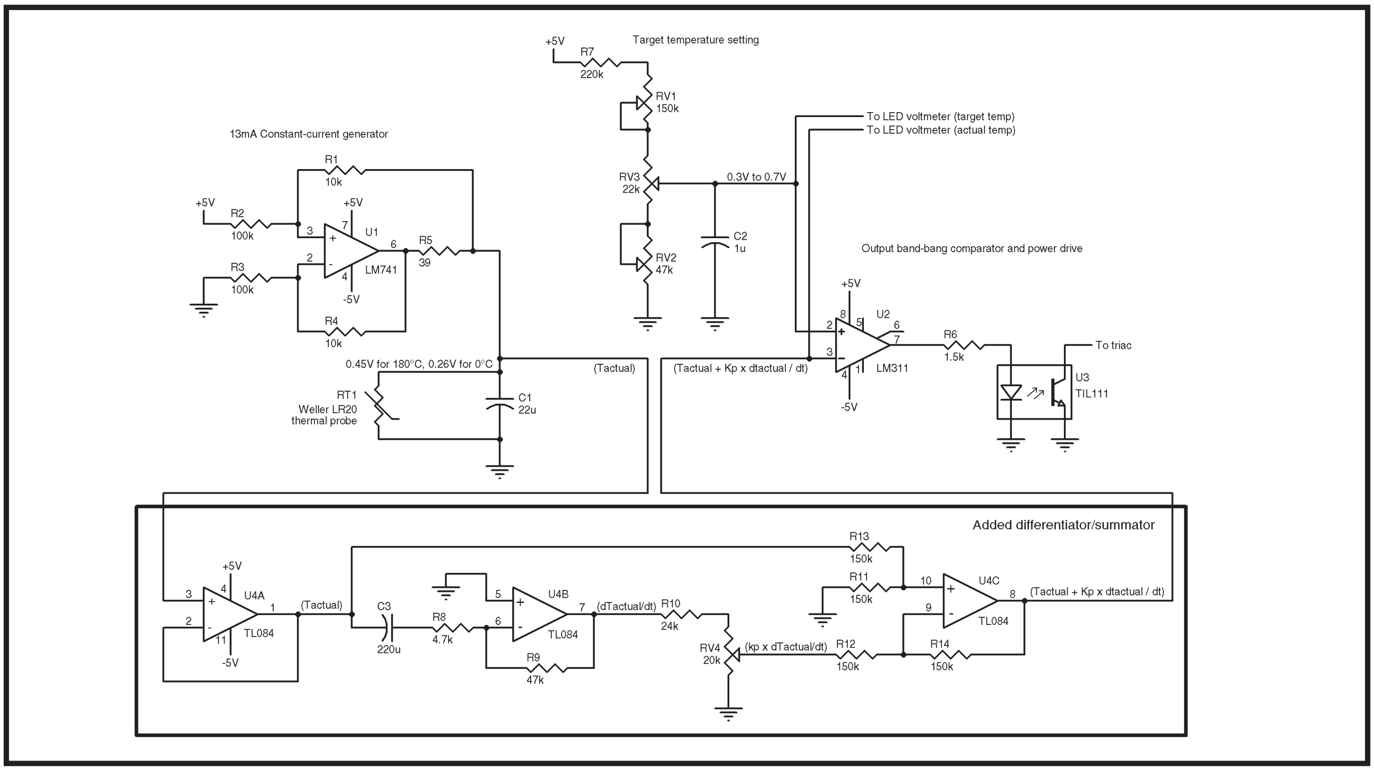
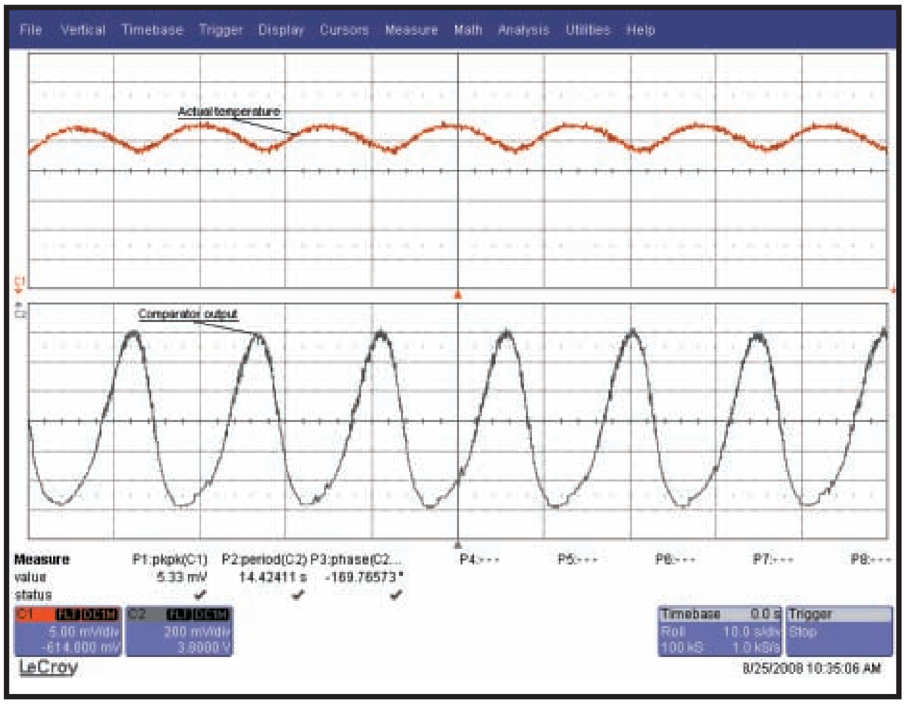

Target temperature setting
13mA Constant-current generator
0.45V for 180°C, 0.26V for 0°C
RT1 Weller LR20 thermal probe
0.3V to 0.7V
To LED voltmeter (target temp)
To LED voltmeter (actual temp)
Output band-bang comparator and power drive
To triac
(Tactual)
(Tactual + Kp x dtactual / dt)
Added differentiator/summator
(dTactual/dt)
(kp x dTactual/dt)
(Tactual + Kp x dtactual / dt)
LM741
LM311
TIL111
TL084

Figure 9—At the top is the original schematic of the temperature control section of my iron controller. A 13-mA constant current generator was driving the resistor temperature sensor, providing a voltage roughly proportional to the temperature. The voltage was simply compared to a preset threshold through an LM311 comparator, then driving a TRIAC through an optoisolator. The modification involved the addition of a quad TL084 op-amp to buffer (U4.A), derive over time (U4B), and sum (U4.C) the signal with its derivative. The potentiometer RV4 sets the Kd gain.



```
Measure   P1:pkpk(C1)  P2:period(C2) P3:phase(C2...   P4:---   P5:---   P6:---   P7:---   P8:---
value         5.33 mV     14.42411 s   -169.76573°
status           ✓             ✓              ✓
C1 FLT DC1M   C2 FLT DC1M                            Timebase   0.0 s   Trigger
  5.00 mV/div   200 mV/div                           Roll   10.0 s/div  Stop
 -614.000 mV    3.8000 V                             100 kS  1.0 kS/s
                                                     8/25/2008 10:35:06 AM
```

Photo 3—This is how the actual iron temperature regulation behaves with the addition of the derivative term. The oscillation of the sensor output is reduced to a little more than 5 mV_{PP}, four times less than with the simple comparator, providing a stability of 1°C.

with Ki = 0. If it is mandatory, increase Ki a little after determining the optimal Kp and Kd parameters, just to a value providing a good, long-term convergence. Then, you will probably need to retune Kd and Kp to readapt the short-term behavior, and then Ki again, and more. Another good way to reduce the risk of instability is to limit the maximum and minimum value of the integral term to a given range with a new parameter MaxIntegral. This is another value to determine by experimentation; but globally, you end up with only four numbers to optimize, which is far easier than going through the full control system theory.

HARDWARE IMPROVEMENTS

It is time to go back to my dear 1987 soldering iron regulator. What did I do in 1989 to reduce the temperature oscillations? I simply added a differential term. I didn't even change the output stage, which is still a TRIAC driven by a comparator in full on/off mode, but I no longer compared the measured value to the preset value. I compared the measured value plus Kd times d(measured value)/dt to the preset value. Think about it twice. This is exactly the same as the PD algorithm as long as the preset value is constant. A PD control loop can be 100% analog (see Figure 9). What were the actual improvements? Compare the oscillogram in Photo 3 with the initial one (see Photo 2). The addition of a Kd parameter reduced the temperature oscillation from 6° down to 1.3°C. That's not bad with just a couple of op-amps more. Photo 3 shows the derivative term in action. The output is no longer fully in phase with the sensor. It starts to increase as soon as the sensor temperature starts to reach its maximum, even if the actual temperature is still above target. This is anticipation.

WRAPPING UP

To be honest, I no longer use the iron controller on a daily basis because I have a newer one. However, I still use it from time to time, even if it is not lead-free compatible. Anyway, I hope I have demonstrated that PID regulations are simple to code, or even to wire with a couple of op-amps. Moreover, they are easy to tune at least for simple systems. Using only the proportional term may already give good results. The derivative term could be added to provide damping (or shaping) of the response. The addition of an integral term ensures that there will be no systematic errors. If the average error is not zero, the integral term will increase over time, reducing the error. However, the integral term is a little more difficult to manage than the Kp and Kd terms because it can make the system unstable. Handle it with care, or add limits on the integral term allowed values. Don't forget that these coefficients can be negative too.

Lastly, note that playing with the Kp, Ki, and Kd coefficients is easy, and looking at their effects on the real-life controlled system is fun. Consider experimenting with a small controlled system rather than exercise yourself directly on your nearby nuclear plant. Anyway, I hope that PID is no longer on the darker side for you! ■

Robert Lacoste lives near Paris, France. He has 18 years of experience working on embedded systems, analog designs, and wireless telecommunications. He has won prizes in more than 15 international design contests. In 2003, Robert started a consulting company, ALCIOM, to share his passion for innovative mixed-signal designs. You can reach him at rlacoste@alciom.com. Don't forget to write "Darker Side" in the subject line to bypass his spam filters.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/221.

RESOURCES

T. Wescott, *Applied Control Theory for Embedded Systems*, Newnes, Burlington, MA, 2006.

Wikipedia, "Control Theory," http://en.wikipedia.org/wiki/Control_theory.

SOURCE

WaveRunner 6050A Oscilloscope
LeCroy Corp.
www.lecroy.com