# Teaching Your PI Controller to Behave (Part V)

Dave (Wisconsin) Wilson Mar 23, 2013

*Dave Wilson, Motion Products Evangelist, Texas Instruments*

So far in this series, we have discussed how to distill the design of a cascaded velocity controller from a bunch of seemingly uncorrelated PI coefficients down to a single system parameter; the *damping factor* ($\delta$). The damping factor represents the tradeoff between system stability and system bandwidth in a single parameter. But how do you choose an appropriate damping factor value for your design, and what does it mean in terms of your system's transient response?

To help answer these questions, let's take a look at figure 1, which illustrates the *open-loop* magnitude and phase response for a speed control system. Note that all frequencies are scaled to the value of the velocity filter pole. Assuming that the current controller's pole is very high with respect to this pole, the *shape* of the curves won't change, regardless of the velocity pole value. In Figure 1, the damping factor is swept from 70 to 1.5 in 8 discrete steps to show how it affects system response. A $\delta$ value of 1.0 corresponds to the condition where the open-loop magnitude response has unity gain at the same frequency as the velocity filter's pole. This scenario results in the velocity filter pole exactly cancelling out the zero which is creating all that nice phase lead we need to stabilize our system. So we see that a $\delta$ value of 1.0 yields a system with zero phase margin, which of course is unstable.

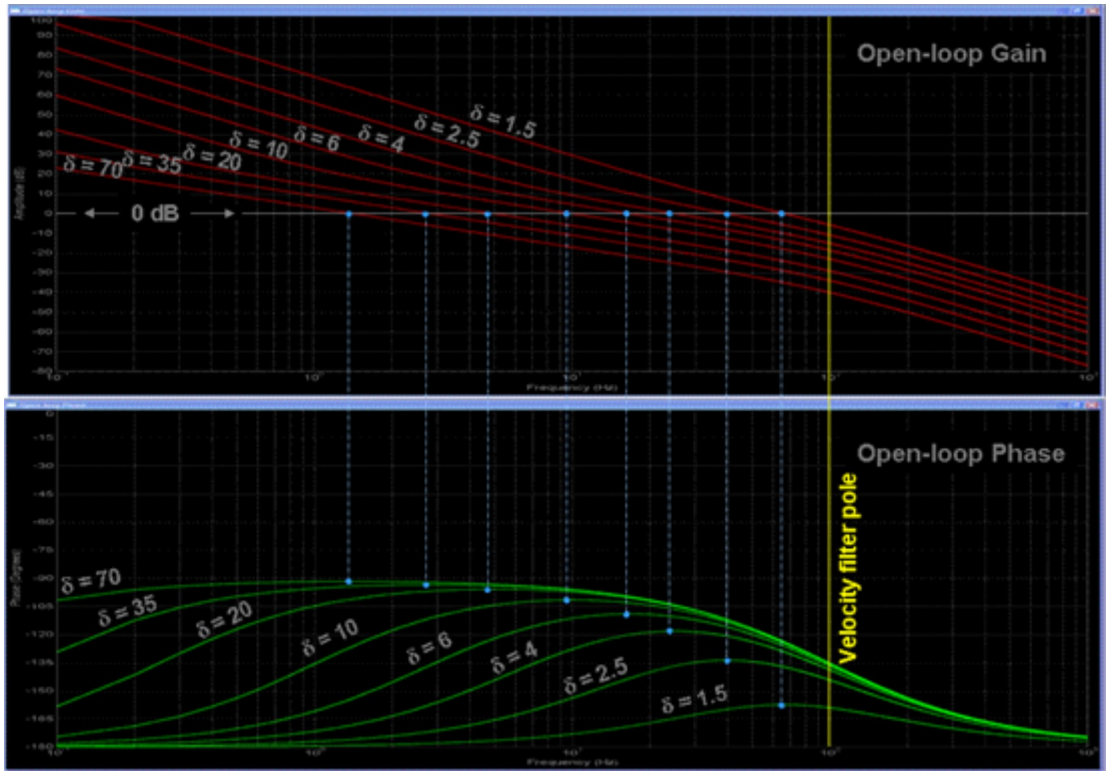


Figure 1. Velocity Controller Open Loop Magnitude and Phase Response as a Function of $\delta$.

One of the goals of the damping factor is to achieve maximum stability for a given bandwidth. This is confirmed in the plots above, which show that the phase margin always peaks to its maximum value at the unity gain frequency. As the damping factor is increased, you eventually reach a point of diminishing returns for phase margin improvement as the signal phase shift approaches -90 degrees. However, the *gain* margin continues to improve as the damping factor is increased.

Figure 2 illustrates the *closed-loop* magnitude response of the velocity loop with respect to the velocity filter pole as $\delta$ is swept from 70 to 1.5.

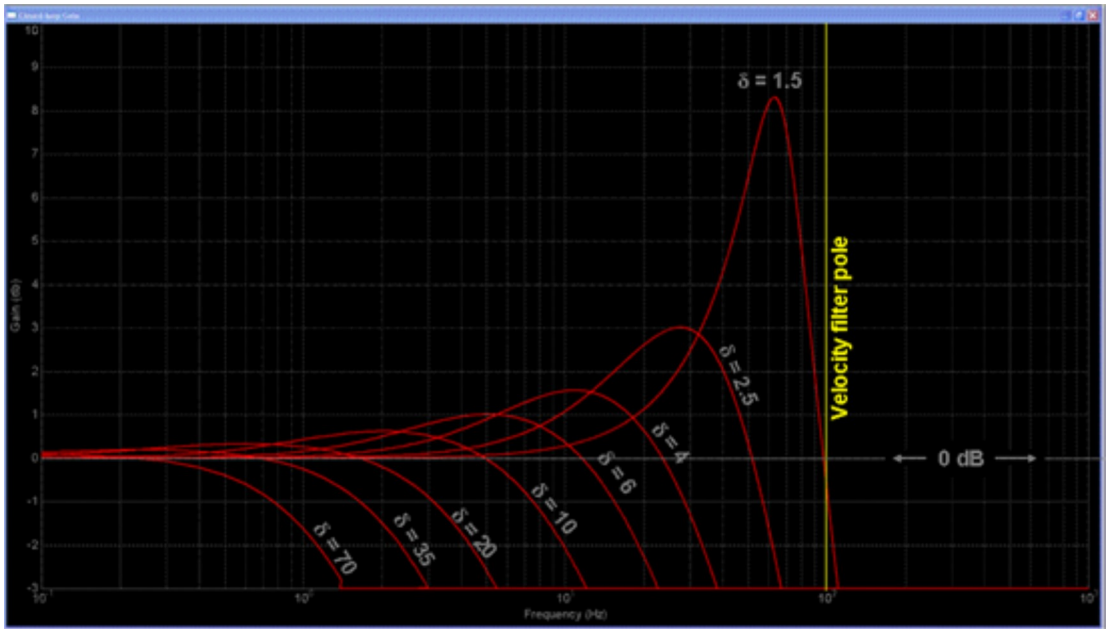


Figure 2. Velocity Controller Closed Loop Bandwidth as a Function of $\delta$.

From figure 2 you can see how decreasing the damping factor increases the bandwidth of the velocity loop. In fact, decreasing the damping factor from 70 to 1.5 increases the bandwidth by almost two orders of magnitude. But as the damping factor approaches unity, the complex poles in the velocity loop approach the $j\omega$ axis, resulting in pronounced peaking of the gain response. This peaking effect is usually undesirable, as it results in excessive underdamped ringing of the step response. You can better see this in figure 3, which shows the normalized step response of the system for various values of damping factor. Values below 2 are usually unacceptable due to the large amount of overshoot they produce. At the other end of the scale, values much above 35 produce extremely long rise and settling times. Your design target window will usually be somewhere in-between these two values.

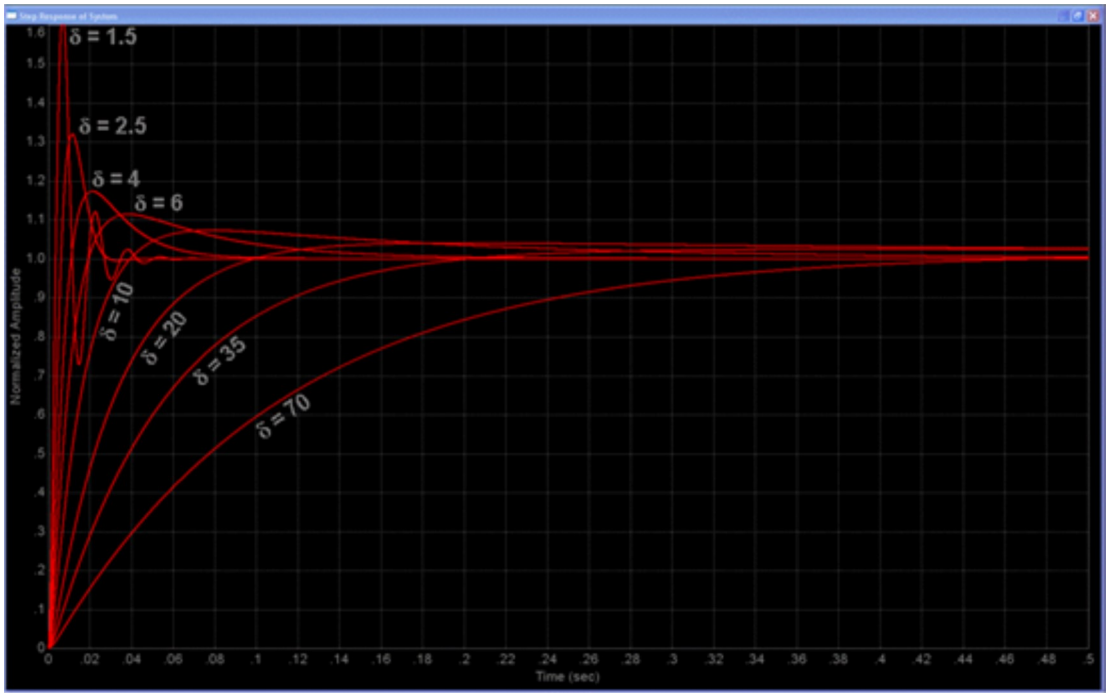


Figure 3. Step Response of Velocity Controller as a Function of $\delta$.

What happens if you pick a damping factor based on a desired *shape* for your step response, but the response *time* is too long? Well, assuming you have the design flexibility to do so, your best recourse might be to increase the velocity filter pole. In figures 1 and 2, all frequencies are scaled to this pole value, which means that if you change it, all other frequencies will change proportionally. So if you double the value of the velocity filter pole in figures 1 and 2, you will half the time axis in figure 3.

But what if you *don't* have this flexibility? Unfortunately, you will be forced to choose between response time and overshoot. In other words, you can solve one problem or the other, but not both at the same time. It's kind of like a water balloon. You can squeeze it down in one spot, but it will pop up somewhere else. I know…it's not fair. But that's the way life is sometimes.

So far we have discussed how to set three of the four PI coefficients in a velocity control loop. But there is still one left…Ka. Recall that Ka sets the bandwidth of the current controller. In many respects, Ka is the hardest one to set since its effect on system performance is more subjective than the other three. In my next blog, let's take a closer look at Ka, its impact on system performance, and hopefully gain some insight into how to set it correctly. In the meantime…

Keep Those Motors Spinning,

www.ti.com/motorblog

2 comments 0 members are here

**High Hopes** *over 11 years ago*

this is really nicely done for a subject that is not covered as succinctly as it should be in the texts .. thanks for demystifying! a couple of questions / comments ..

1. do you think it is acceptable for a controller to rail to the output (anti-windup limit) during motor acceleration in a high performance drive? doesn't that mean you are not in control during that period?

2. anti-windup is a necessary function but i wonder about its common implementation to just turn OFF the integrator. this is not consistent with continuous control theory

3. is there any reason why one would choose a parallel PI controller even though it is harder to tune? maybe it is easier implement in a digital controller? just wondering why anyone would chose it when it is so hard to use in practise. the parallel PI controller i have never seen a tutorial that was understandable and practical. hope you tackle that subject next!

thanks for sharing your knowledge.

**Dave (Wisconsin) Wilson** *over 11 years ago*

Thanks High Hopes for your excellent questions. Several of them deal with topics which I plan to address in greater detail in later parts of this series. But let me try to provide some insight into your specific questions.

1. Yes, I think it is acceptable for a controller to saturate, as long as it is done in a controlled fashion. For example, in a cascaded controller where the velocity controller output feeds the current controller, there are many examples where we reach the current limit during acceleration. This is usually implemented by providing rail limits on the output of the velocity controller, which should also affect the integrator limits inside the velocity controller. So even though the velocity loop hits the rail, the current loop is still functioning correctly by limiting the current.

2. Turning the integrator on and off is a common practice which I have actually done myself in the past. There are 2 ways I know of to do this: A. the switch is controlled by a timer which just waits a specific amount of time after a move has been initiated. B. Wait until the actual trajectory closes to within a certain limit w.r.t. the commanded trajectory, and then switch on the integrator to drive it the rest of the way home. However, integrator switching can result in a "pop" in the output waveform under high torque loading when you have to turn the integrator off at the start of the next trajectory.

3. I talked about the comparison between the parallel and series forms of the PI controller in part 1 of this series. From an implementation point of view, they require the same MIPS and pretty much the same coding complexity. In fact, the two forms yield identical results by simply scaling Kb to equal Ki/Kp. I would guess that most people who use the parallel form just don't know about some of the analysis benefits of the series form. Either that, or they are content with a more emperical approach to tuning.

-Dave