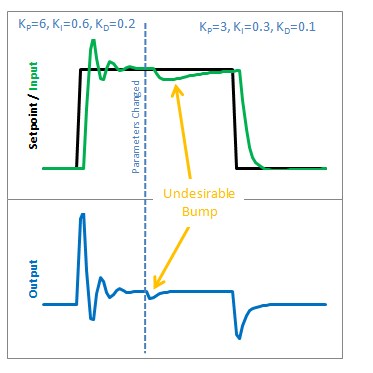
**Improving the Beginner’s PID: Tuning Changes**

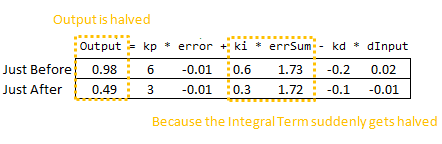
(This is Modification #3 in a [larger series](http://brettbeauregard.com/blog/2011/04/improving-the-beginner%e2%80%99s-pid-tuning-changes/improving-the-beginners-pid-introduction) on writing a solid PID algorithm)

**The Problem**

The ability to change tuning parameters while the system is running is a must for any respectable PID algorithm.



The Beginner’s PID acts a little crazy if you try to change the tunings while it’s running. Let’s see why. Here is the state of the beginner’s PID before and after the parameter change above:

[](http://brettbeauregard.com/blog/wp-content/uploads/2011/03/BadIntegralCode.png)

So we can immediately blame this bump on the Integral Term (or “I Term”). It’s the only thing that changes drastically when the parameters change. Why did this happen? It has to do with the beginner’s interpretation of the Integral:

[http://brettbeauregard.com/blog/wp-content/uploads/2011/03/BadIntegralEqn1.png](http://brettbeauregard.com/blog/wp-content/uploads/2011/03/BadIntegralEqn1.png)

This interpretation works fine until the Ki is changed. Then, all of a sudden, you multiply this new Ki times the entire error sum that you have accumulated. That’s not what we wanted! We only wanted to affect things moving forward!

### The Solution

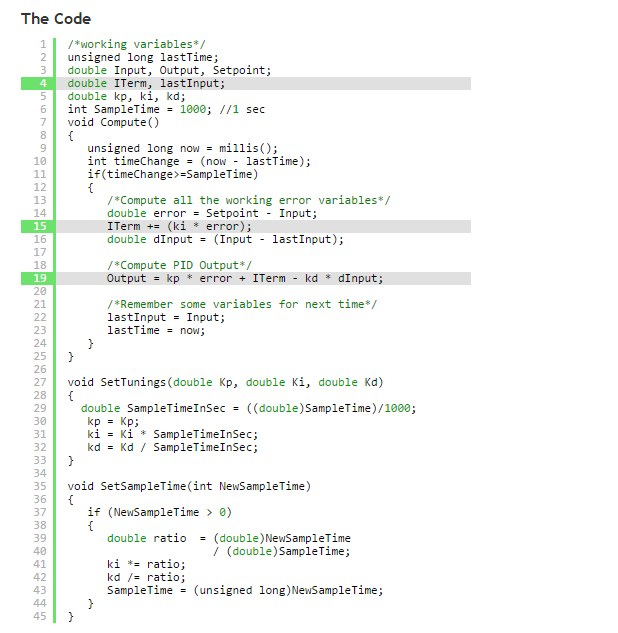
There are a couple ways I know of to deal with this problem. The method I used in the last library was to rescale errSum. Ki doubled? Cut errSum in Half. That keeps the I Term from bumping, and it works. It’s kind of clunky though, and I’ve come up with something more elegant. (There’s no way I’m the first to have thought of this, but I did think of it on my own. That counts damnit!)

The solution requires a little basic algebra (or is it calculus?)

[http://brettbeauregard.com/blog/wp-content/uploads/2011/03/GoodIntegralEqn.png](http://brettbeauregard.com/blog/wp-content/uploads/2011/03/GoodIntegralEqn.png)

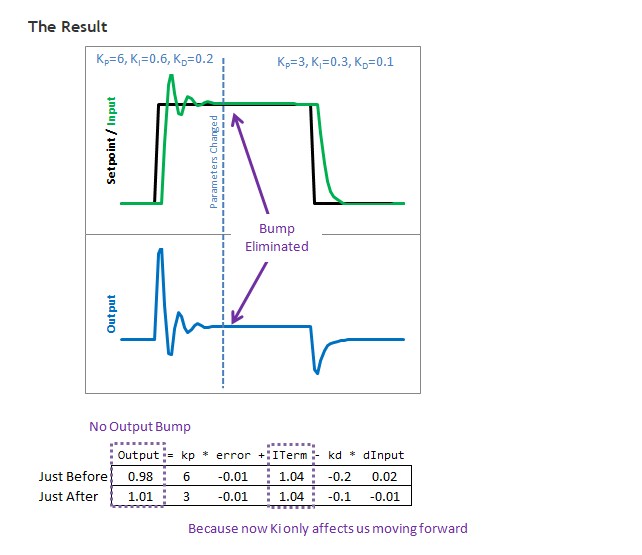
Instead of having the Ki live outside the integral, we bring it inside. It looks like we haven’t done anything, but we’ll see that in practice this makes a big difference.

Now, we take the error and multiply it by whatever the Ki is at that time. We then store the sum of THAT. When the Ki changes, there’s no bump because all the old Ki’s are already “in the bank” so to speak. We get a smooth transfer with no additional math operations. It may make me a geek but I think that’s pretty sexy.



So we replaced the errSum variable with a composite ITerm variable [Line 4]. It sums Ki\*error, rather than just error [Line 15]. Also, because Ki is now buried in ITerm, it’s removed from the main PID calculation [Line 19].

### The Result



So how does this fix things. Before when ki was changed, it rescaled the entire sum of the error; every error value we had seen. With this code, the previous error remains untouched, and the new ki only affects things moving forward, which is exactly what we want.  
[Next >>](http://brettbeauregard.com/blog/2011/04/improving-the-beginner%e2%80%99s-pid-tuning-changes/improving-the-beginner%E2%80%99s-pid-reset-windup)