

Project Blog

Project updates and... um...

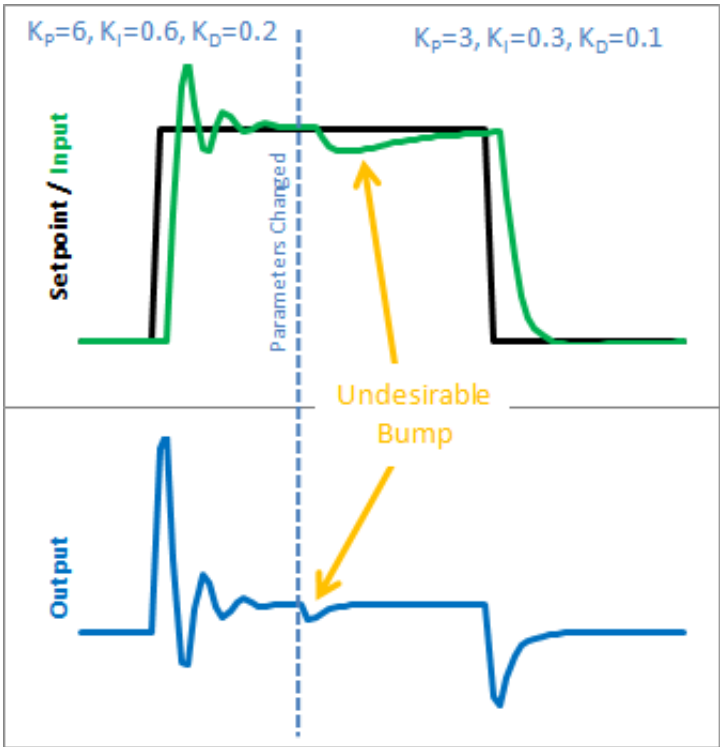
« [Improving the Beginner’s PID – Derivative Kick](#)
[Improving the Beginner’s PID: Reset Windup](#) »

Improving the Beginner’s PID: Tuning Changes

(This is Modification #3 in a [larger series](#) 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:

Output is halved

	Output	=	kp	*	error	+	ki	*	errSum	-	kd	*	dInput
Just Before	0.98		6		-0.01		0.6		1.73		-0.2		0.02
Just After	0.49		3		-0.01		0.3		1.72		-0.1		-0.01

Because the Integral Term suddenly gets halved

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:

$$K_I \int e dt \approx K_I [e_n + e_{n-1} + \dots]$$

This interpretation works fine until the K_i is changed. Then, all of a sudden, you multiply this new K_i 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`. K_i 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?)

$$K_i \int e \, dt = \int K_i e \, dt$$
$$\int K_i e \, dt \approx K_{i_n} e_n + K_{i_{n-1}} e_{n-1} + \dots$$

Instead of having the K_i 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 K_i is at that time. We then store the sum of THAT. When the K_i changes, there's no bump because all the old K_i '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.

The Code

```
1  /*working variables*/
2  unsigned long lastTime;
3  double Input, Output, Setpoint;
4  double ITerm, lastInput;
5  double kp, ki, kd;
6  int SampleTime = 1000; //1 sec
7  void Compute()
8  {
9      unsigned long now = millis();
10     int timeChange = (now - lastTime);
11     if(timeChange>=SampleTime)
12     {
13         /*Compute all the working error variables*/
14         double error = Setpoint - Input;
15         ITerm += (ki * error);
16         double dInput = (Input - lastInput);
17
18         /*Compute PID Output*/
19         Output = kp * error + ITerm - kd * dInput;
20
21         /*Remember some variables for next time*/
22         lastInput = Input;
23         lastTime = now;
24     }
25 }
26
27 void SetTunings(double Kp, double Ki, double Kd)
28 {
29     double SampleTimeInSec = ((double)SampleTime)/1000;
30     kp = Kp;
31     ki = Ki * SampleTimeInSec;
```

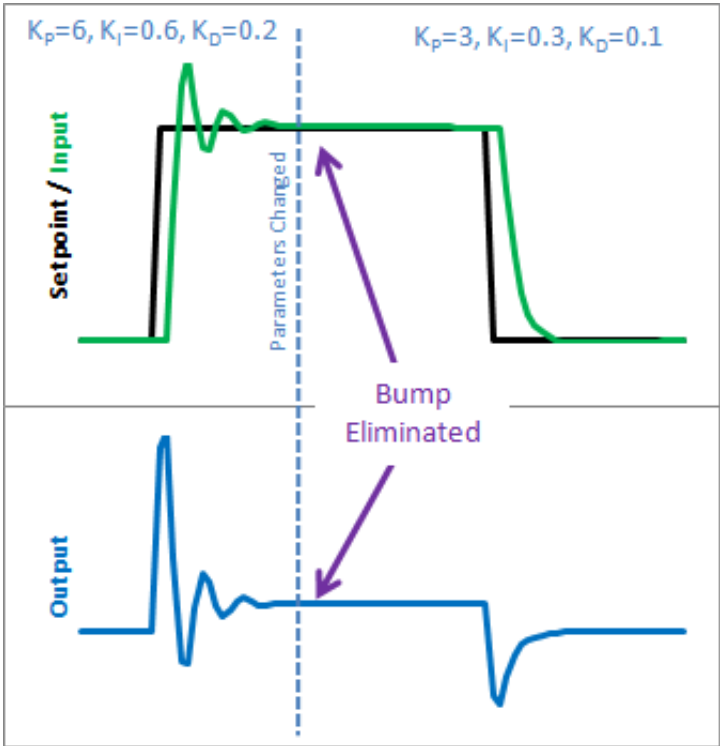
```

32     kd = Kd / SampleTimeInSec;
33 }
34
35 void SetSampleTime(int NewSampleTime)
36 {
37     if (NewSampleTime > 0)
38     {
39         double ratio = (double)NewSampleTime
40                        / (double)SampleTime;
41         ki *= ratio;
42         kd /= ratio;
43         SampleTime = (unsigned long)NewSampleTime;
44     }
45 }

```

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



No Output Bump

	Output	kp	error	ITerm	kd	dInput
Just Before	0.98	6	-0.01	1.04	-0.2	0.02
Just After	1.01	3	-0.01	1.04	-0.1	-0.01

Because now Ki only affects us moving forward


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 >>](#)



Tags: [Arduino](#), [Beginner's PID](#), [PID](#)

6 Responses to “Improving the Beginner’s PID: Tuning Changes”

1.  *Vasileios Kostelidis* says:
[July 3, 2012 at 5:54 am](#)

I don’t get it. I am trying for about an hour but I cannot get it.
It looks the same arithmetically to me.


I don’t know what am I missing.

2.  *Brett* says:
[July 4, 2012 at 7:36 am](#)

when you look at the calculus, it is identical. when it comes to implementation, it is different. let’s say that at t=10, Ki changes from 5 to 10, with Kp and Kd staying the same. now, let’s look at the integral term values at t=20:

old way: $10 * (\text{Integral of error from } t=0 \text{ to } 20)$
new way: $(\text{Integral of } (5 * \text{error}) \text{ from } t=0 \text{ to } 10) + (\text{Integral of } (10 * \text{error}) \text{ from } t=10 \text{ to } 20)$

hopefully this helps reveal the difference. again. this is something you only notice if the value of Ki changes. if it stays the same, both methods are identical.

3.  *Vasileios Kostelidis* says:
[July 4, 2012 at 8:06 am](#)

So, since:

old way: $10 * (\text{Integral of error from } t=0 \text{ to } 20)$
new way: $(\text{Integral of } (5 * \text{error}) \text{ from } t=0 \text{ to } 10) + (\text{Integral of } (10 * \text{error}) \text{ from } t=10 \text{ to } 20)$

Then:

old way: $10 * (\text{Integral of error from } t=0 \text{ to } 20)$
new way: $5 * (\text{Integral of error from } t=0 \text{ to } 10) + 10 * (\text{Integral of error from } t=10 \text{ to } 20)$

You are basically changing the algorithm. I kind of understand it now, but, is it OK to change the algorithm like that?

I know that this may very well be a silly question, but I don’t remember much from my control theory :).

Vasilis.

4.  *Brett* says:
[July 4, 2012 at 8:11 am](#)

This entire series is about changing the algorithm. the idea is that the PID algorithm behaves in a predictable, reliable way, EXCEPT for in certain situations. The goal of these changes (all of them) is to make the algorithm work as you would expect, EVEN WHEN these special conditions occur.

5.  *Vasileios Kostelidis* says:

[July 5, 2012 at 7:59 am](#)

OK, sounds nice. Thanks for the quick and accurate help!
Will let you know when my PID controller works.



6. *Marcelo* says:

[November 19, 2012 at 5:49 pm](#)

The algebra is actually not correct. If both terms were equal, the result should be the same. Your k_i is time dependent, which is why it cannot get out of the integrand. Still, the integral output term really is $o_i(t) = \int k_i(t) * e(t) dt$.

Anyway, this is a nice fix and a very useful library.

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

• Search for:

• Links



o



o



o

• This Site

- o [About](#)
- o [Project Index](#)

• Categories

- o [PID](#) (23)
 - [Coding](#) (11)
 - [Front End](#) (2)
 - [Showcase](#) (4)
- o [Projects](#) (40)
 - [Craft](#) (6)
 - [Electronic](#) (12)
 - [Mechanical](#) (26)
- o [Uncategorized](#) (5)

• Archives

- o [November 2012](#)
 - o [September 2012](#)
 - o [July 2012](#)
 - o [June 2012](#)
 - o [April 2012](#)
 - o [March 2012](#)
 - o [January 2012](#)
 - o [December 2011](#)
 - o [October 2011](#)
 - o [September 2011](#)
 - o [August 2011](#)
 - o [July 2011](#)
 - o [June 2011](#)
 - o [May 2011](#)
 - o [April 2011](#)
 - o [September 2010](#)
 - o [August 2010](#)
 - o [July 2010](#)
 - o [March 2010](#)
 - o [November 2009](#)
 - o [October 2009](#)
 - o [September 2009](#)
 - o [August 2009](#)
 - o [July 2009](#)
 - o [June 2009](#)
 - o [May 2009](#)
-

