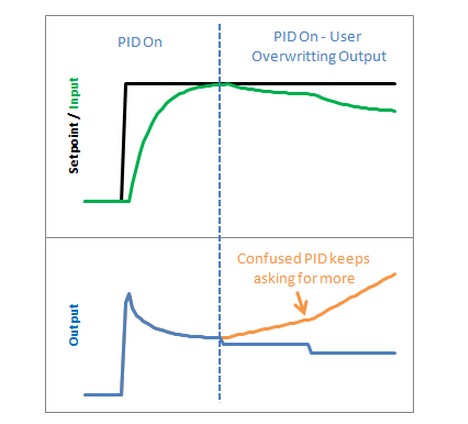
**Improving the Beginner’s PID: On/Off**

(This is Modification #5 in a [larger series](http://brettbeauregard.com/blog/2011/04/improving-the-beginner%e2%80%99s-pid-onoff/improving-the-beginners-pid-introduction) on writing a solid PID algorithm)

**The Problem**

As nice as it is to have a PID controller, sometimes you don’t care what it has to say.



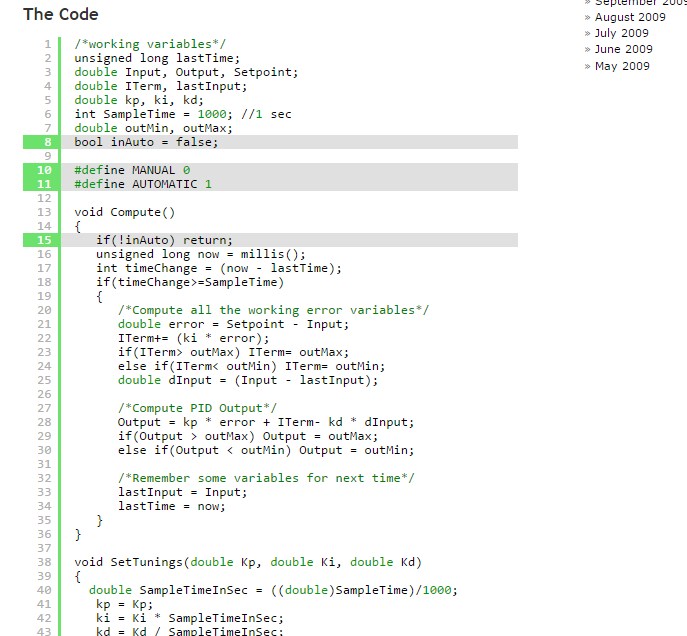
Let’s say at some point in your program you want to force the output to a certain value (0 for example) you could certainly do this in the calling routine:

void loop()  
{  
Compute();  
Output=0;  
}

This way, no matter what the PID says, you just overwrite its value. This is a terrible idea in practice however. The PID will become very confused: “I keep moving the output, and nothing’s happening! What gives?! Let me move it some more.” As a result, when you stop over-writing the output and switch back to the PID, you will likely get a huge and immediate change in the output value.

### The Solution

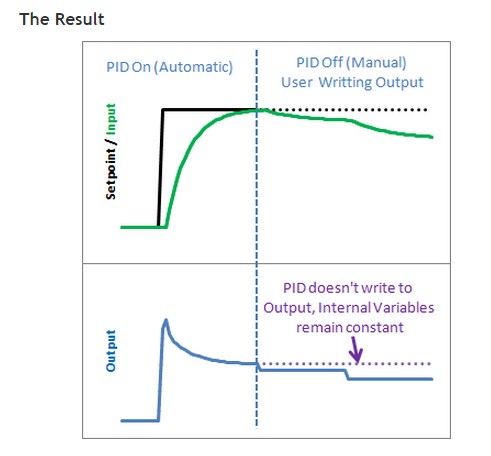
The solution to this problem is to have a means to turn the PID off and on. The common terms for these states are “Manual” (I will adjust the value by hand) and “Automatic” (the PID will automatically adjust the output). Let’s see how this is done in code:





A fairly simple solution. If you’re not in automatic mode, immediately leave the Compute function without adjusting the Output or any internal variables.

### The Result



It’s true that you could achieve a similar effect by just not calling Compute from the calling routine, but this solution keeps the workings of the PID contained, which is kind of what we need. By keeping things internal we can keep track of which mode were in, and more importantly it let’s us know when we change modes. That leads us to the next issue…  
[Next >>](http://brettbeauregard.com/blog/2011/04/improving-the-beginner%e2%80%99s-pid-onoff/improving-the-beginner%E2%80%99s-pid-initialization)