Project Blog

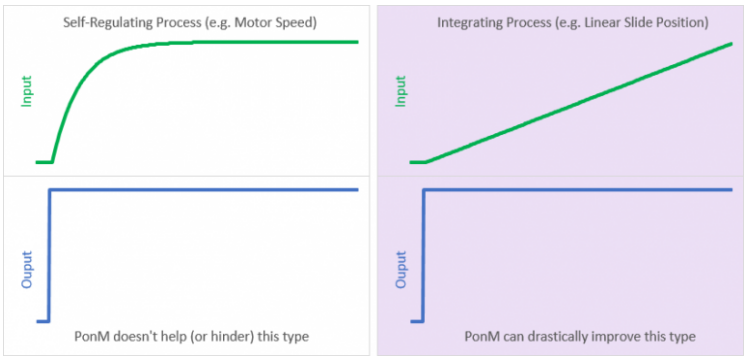
Project updates and... um...

« Priorites
Proportional on Measurement – The Code »

Introducing Proportional On Measurement

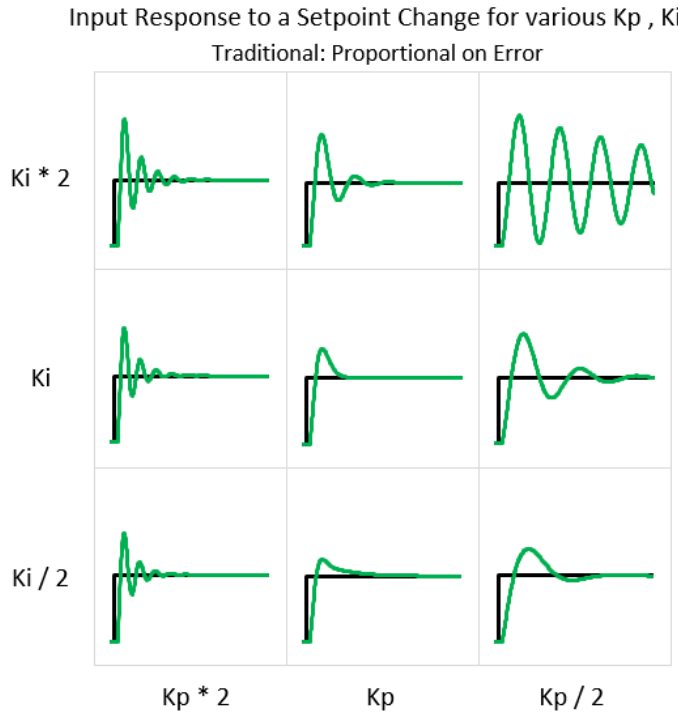
It's been quite a while, but I've finally updated the Arduino PID Library. What I've added is a nearly-unknown feature, but one which I think will be a boon to the hobby community. It's called "Proportional on Measurement" (PonM for short).

Why you should care

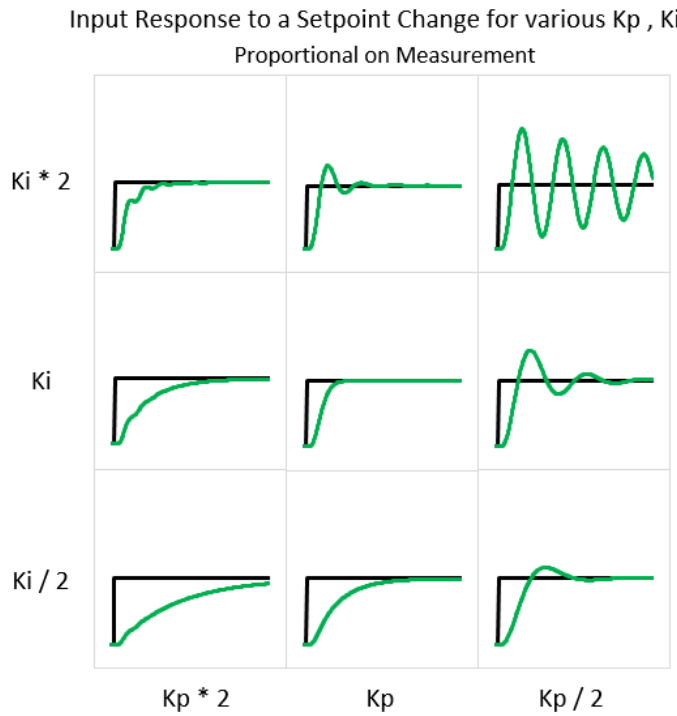


There are processes out there that are known as "Integrating Processes." These are processes for which the output from the pid controls the rate of change of the input. In industry these comprise a small percentage of all processes, but in the hobby world these guys are **everywhere**: Sous-vide, linear slide, and 3D printer extruder temperature are all examples of this type of process.

The frustrating thing about these processes is that with traditional PI or PID control they overshoot setpoint. Not sometimes, but always:



This can be maddening if you don't know about it. You can adjust the tuning parameters forever and the overshoot will still be there; the underlying math makes it so. Proportional on Measurement changes the underlying math. As a result, it's possible to find sets of tuning parameters where overshoot doesn't occur:



Overshoot can still happen to be sure, but it's not unavoidable. With PonM and the right tuning parameters, that sous vide or linear slide can coast right in to setpoint without going over.

So What is Proportional on Measurement?

Similar to Derivative on Measurement, PonM changes what the proportional term is looking at. Instead of error, the P-Term is fed the current value of the PID input.

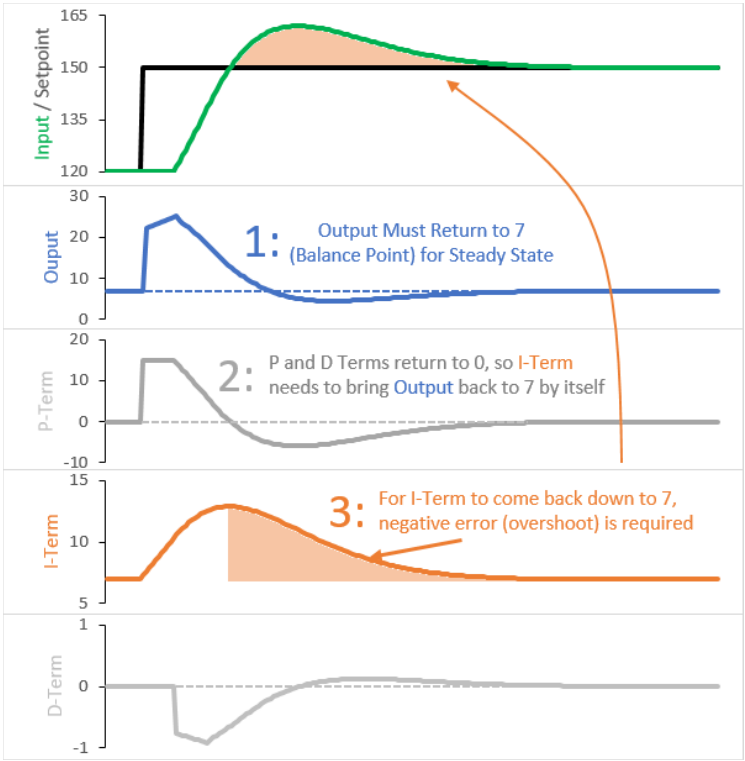
Proportional on Error: $$\text{Output} = K_p e(t) + K_I \int e(t)dt - K_d \frac{d\text{Input}}{dt}$$

Proportional on Measurement: $$\text{Output} = -K_p[\text{Input}(t) - \text{Input}_{init}] + K_I \int e(t)dt - K_d \frac{d\text{Input}}{dt}$$

Unlike Derivative on Measurement, the impact on performance is drastic. With DonM, the derivative term still has the same job: resist steep changes and thus dampen oscillations driven by P and I. Proportional on Measurement, on the other hand, fundamentally changes what the proportional term does. Instead of being a driving force like I, it becomes a resistive force like D. This means that with PonM, a bigger Kp will make your controller more conservative.

Great. But how does this eliminate overshoot?

To undestand the problem, and fix, it helps to look at the different terms and what they contribute to the overall PID output. Here's a response to a setpoint change for an integrating process (a sous-vide) using a traditional PID:



The two big things to notice are:

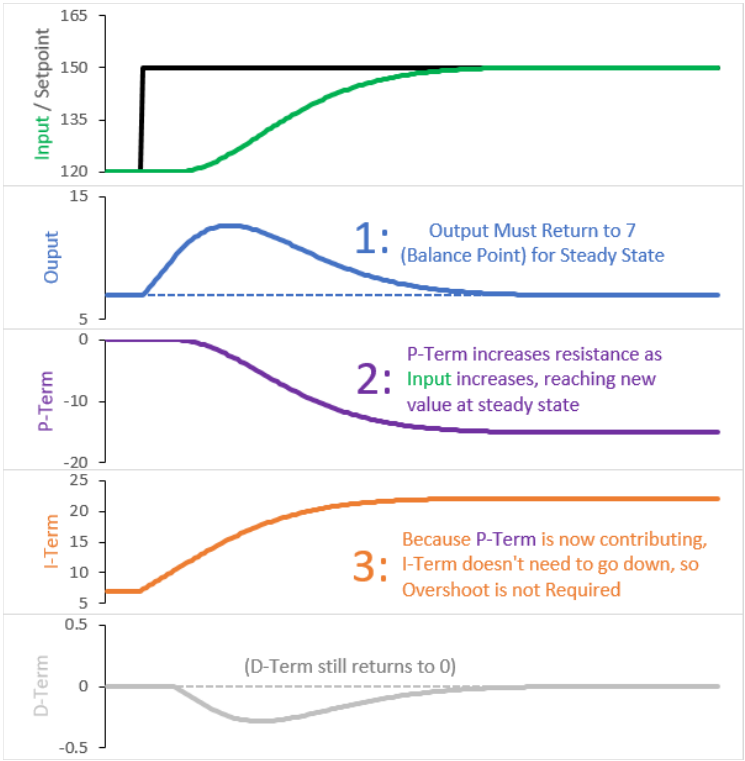
- When we're at setpoint, the I-Term is the only contributor to the overall output.
- Even though the setpoint is different at the start and end, the output returns to the same value. This value is generally known as the "Balance Point": the output which results in 0 Input slope. For a sous-vide, this would correspond to just enough heat to compensate for heat-losses to the surroundings.

Here we can see why overshoot happens, and will always happen. When the setpoint first changes, the error present causes the I-Term to grow. To keep the process stable at the new setpoint, the output will need to return to the balance point. The only way for that to happen is for the I-Term to shrink. The only way for THAT to happen is to have negative error, which only happens if you're above setpoint.

PonM changes the game

Here is the same sous-vide controlled using Proportional on Measurement (and the same tuning

parameters):



Here you should notice that:

- The P-Term is now providing a resistive force. The higher the Input goes, the more negative it becomes.
- Where before the P-Term became zero at the new setpoint, it now continues to have a value.

The fact the the P-Term doesn't return to 0 is the key. This means that the I-Term doesn't have to return to the balance point on its own. P and I together can return the output to the balance point without the I-Term needing to shrink. Because it doesn't need to shrink, overshoot isn't required.

How to use it in the new PID Library

If you're ready to try Proportional on Measurement, and you've installed the latest version of the PID library, setting it up is pretty easy. The primary way to use PonM is to specify it in the overloaded constructor:

```
PID myPID(&input, &output, &setpoint, kp, ki, kd, P_ON_M, DIRECT);
```

If you'd like to switch between PonM and PonE at runtime, the SetTunings function is also overloaded:

```
myPID.SetTunings(kp, ki, kd, P_ON_M);
```

You only need to call the overloaded method when you want to switch. Otherwise you can use the regular SetTunings function and it will remember your choice.

Flattr this!

This entry was posted on Tuesday, June 20th, 2017 at 8:19 am and is filed under PID. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site.

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

- Search for: Search

- **Links**
  - 
  - 
  - 

- **This Site**
  - About
  - Project Index

- **Categories**
  - PID (25)
    - Coding (12)
    - Front End (2)
    - Showcase (4)
  - Projects (40)
    - Craft (6)
    - Electronic (12)

Project Blog is proudly powered by WordPress
Entries (RSS) and Comments (RSS).