Project Blog

Project updates and… um…

« Collapsible Knitting Needle
Knitting »

Arduino PID Autotune Library

At long last, I've released an Autotune Library to compliment the Arduino PID Library. When I released the current version of the PID Library, I did an insanely extensive series of posts to get people comfortable with what was going on inside.

While not nearly as in-depth, that's the goal of this post. I'll explain what the Autotune Library is trying to accomplish, and how it goes about its business.

Attribution

For A couple years I've wanted to have an Autotune Library, but due to an agreement with my employer, I wasn't able to write one. BUT! when I found the AutotunerPID Toolkit by William Spinelli I was good to go; My company had no problem with me porting and augmenting and existing open source project.

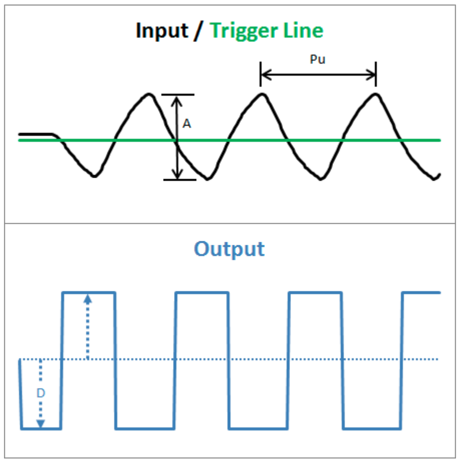
I converted the code from matlab, made some tweaks to the peak identification code, and switched it from the Standard form (Kc, Ti, Td) to the Ideal form (Kp, Ki, Kd.) Other than that, all credit goes to Mr. Spinelli.

The Theory

The best tuning parameters (Kp, Ki, Kd,) for a PID controller are going to depend on what that controller is driving. The best tunings for a toaster oven are going to be different than the best tunings for a sous-vide cooker.

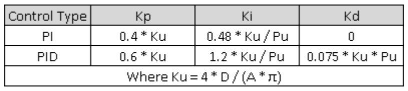
Autotuners attempt to figure out the nature of what the controller is driving, then back-calculate tuning parameters from that. There are various methods of doing this, but most involve changing the PID Output in some way then observing how the Input responds.

The method used in the library is known as the relay method. here's how it works:



Starting at steady state (both Input and Output are steady,) the Output is stepped in one direction by some distance D. When the Input crosses a trigger line, the output changes to the other direction by distance D.

By analyzing how far apart the peaks are, and how big they are in relation to the output changes, the Autotuner can tell the difference between one type of process and another. As a result, different systems will get custom tuning parameters:



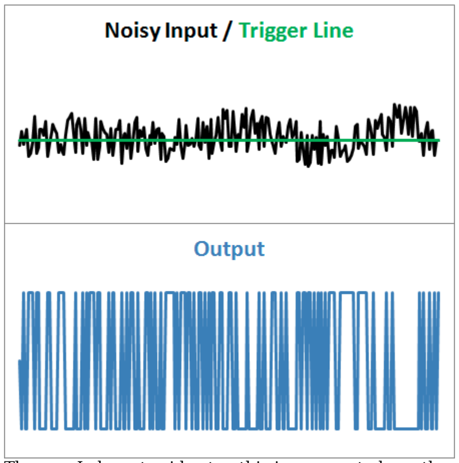
Control Type	Kp	Ki	Kd
PI	0.4 * Ku	0.48 * Ku / Pu	0
PID	0.6 * Ku	1.2 * Ku / Pu	0.075 * Ku * Pu
Where Ku = 4 * D / (A * π)			

The Implementation

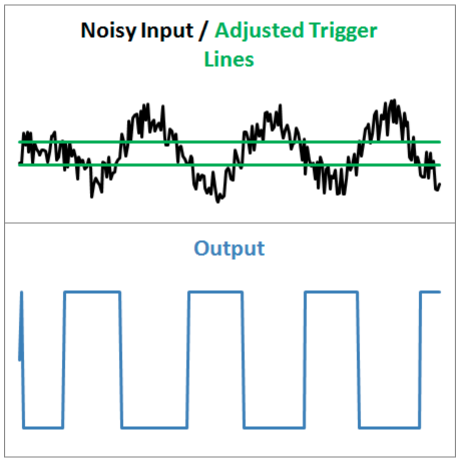
This works well in theory, but real-world data isn't very cooperative. The input signal is usually noisy, which causes two main problems.

Problem #1: When to step?

Since a noisy signal is choppy, it's likely that the trigger line will be crossed several times as the Input moves past it. This can cause mild chatter in the output, or if severe, can completely destroy things:

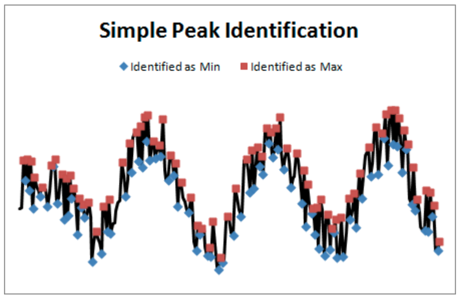


The way I chose to side-step this issue was to have the user specify a noise band. In effect, this creates two trigger lines. Since the distance between them is equal to the noise (if properly set) it's less likely that multiple crossings will occur due to signal chatter.



Problem #2: Peak Identification

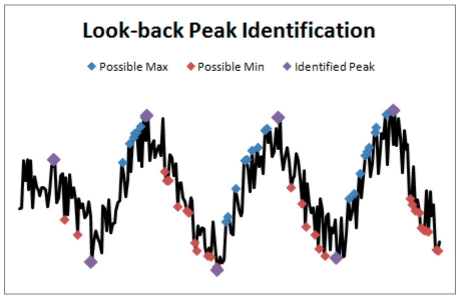
In a simulated world, identifying the peaks is easy: when the Input signal changes direction, that's a minimum or a maximum (depending an which change occured.) In a noisy world however, this method fails:



Every noise blip is a direction change. To deal with this issue I added a "look-back time" parameter. It's an awful name. If you can think of something better let me know.

At any rate, the user defines some window, say 10 seconds. The Library then compares the current point to the last ten seconds of data. If it is a min or a max, it gets flagged as a possible peak.

When the flagged point switches from being a max to a min, or vice versa, the previously flagged point is confirmed as a peak.



Another way of explaining the look-back time is that a point will be identified as a peak if it is the largest (or smallest) value within one look-back into the future or past. Like I said: awful name.

You should also know…

- The number of cycles performed will vary between 3 and 10. The algorithm waits until the last 3 maxima have been within 5% of each other. This is trying to ensure that we've reached a stable oscillation and there's no external strangeness happening. This leads me to…
- I'm not the biggest fan of Autotune. I've often said, and still believe, that a moderately trained person will beat an Autotuner every day of the week. There's just so much that can go wrong without the algorithm knowing about it. That being said, Autotune is a valuable tool to help the novice get into the ballpark.

Flattr this!

Tags: Autotune, osPID, PID

This entry was posted on Saturday, January 28th, 2012 at 2:41 pm and is filed under Coding, 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.

21 Responses to "Arduino PID Autotune Library"

1. *Kenneth Finnegan* says:
   January 28, 2012 at 8:47 pm

   I think I've learned more useful stuff from your half-a-dozen blog posts than I did last year in two quarters of MechE controls classes at the university. They just refuse to talk about inconveniences like actual real-world data.

   You wouldn't happen to know about some outstanding textbook on the subject? My Controls text by Nise is pretty useless when you try to actually do anything with it.

2. *Brett* says:
   January 28, 2012 at 9:18 pm

   Yeah I got lucky in college. My controls professor didn't talk about laplace transforms until near the end of the semester, preferring to teach us using real world simulations. Hands down the reason I'm a controls engineer today.

   You'll want to read his entire website: http://controlguru.com/ It's formatted like it's 2004, but it's solid gold. If you think you're learning something from me, that site will be like superfly TNT.

3. *Jean-François Théorêt* says:
   January 29, 2012 at 11:21 am

   Thanks again for a great post!

   As for: "It's an awful name. If you can think of something better let me know."

   How about hysteresis time?

4. *pid_hobby* says:
   February 6, 2012 at 8:47 am

   i learn from this blog very much!!!

   i am chinese engineer. if someone need chinese version of "improving the beginner's PID-introduction" , drop me a mail "lijian0222#163.com" #->@ ☺

   i have translated this veryyyyyyyyyyyyyyyyyyyyyy good blog!!

   Brett's blog is superfly TNT!!!! ☺ Sorry for my poor english~~

   BTW:

   KU = 4 * D / A*PI (why???) may be i can find the answer from matlab source!!!

   i forgot how to use matlab. in my collage, i use this magic math tools .
   but when i go to work. i forgot how to use matlab anymore.

5. *Max* says:
   March 11, 2012 at 5:53 am

   Thanks for a great post!
   Sorry for my bad English but I have few questions for you.
   Could you explain peak identification in more detail maner? I looked at your source code and many things not clear for me. For example why "peakCount++" only in isMin branch…

6. *Max* says:
   March 11, 2012 at 7:00 am

   I also suppose that nLookBack is a need to be replaced by initCount in this piece of code:

   ```
   if(nLookBack<9)
   { //we don't want to trust the maxes or mins until the inputs array has been filled
   initCount++;
   return 0;
   }
   ```

7. *leo* says:
   April 10, 2012 at 5:56 am

   hello brett,

   I've read your posts on the PID library (Improving the Beginner's PID) and found them really resourceful!

   I have a question regarding the implementation of the PID Autotune though. First off, I'm no expert at programming.

   I am trying to use the PID Autotune to control the hovering of my quadcopter. The thing is, I don't know what Kp, Ki and Kd values to use for the control. Long story short, when I use the Autotune library and its functions, is it necessary for me to initialise the Kp, Ki and Kd values? I don't know what the values should be though. And it cannot be any value, right?

8. *tom* says:
[April 25, 2012 at 11:25 pm]

   Hi Brett. I love your projects and blog. A couple of comments.

   Looks like the default value for the hysteresis in the library is beta = 0.5/30 = 0.016. Am I right in thinking that is pretty aggressive?

   Would it be a good idea to use some kind of low pass filter on the measurements? The better to estimate period and amplitude free of noise.

9. *Brett* says:
[April 26, 2012 at 7:00 am]

   @tom there are many, many improvements that could be done do this code, both in signal processing and in pulse generation. As I said, because I work for a pid tuning company, I felt squeamish just making the additions that I did. The beauty of open source, however, is that someone who knows what they're doing can make a branch on github and improve the heck out of this thing! (those improvements would be rolled into the osPID firmware, with attribution of course)

10. *tom* says:
[April 26, 2012 at 2:05 pm]

   OK, I might try porting a little bit more of the original MATLAB code.

11. *tom* says:
[May 2, 2012 at 11:03 am]

   I forked the code and added some stuff. I'm afraid I don't really know how to use Github so I hope I didn't mess up.

12. *Brett* says:
[May 2, 2012 at 11:18 am]

   @tom dither! neat. one thing: on the .h you have SetDither(double), but in the cpp it's an int. I'm surprised that compiled for you.

   I also see you put it some of the additional tuning correlations. also cool! can't wait to see what else you add!

13. *tom* says:
[May 2, 2012 at 2:48 pm]

   Thanks Brett, ahem it didn't compile so I just changed it. ☺

   Figured I need the dither because I am measuring using a DS18B20 accurate to about 0.1 oC and it tends to hunt around the set temperature. I've added some similar code to the PID library.

   I am looking at coding low pass filters for the input (to avoid aliasing) and the derivative (to limit gain).

14. *jazz* says:
[May 13, 2012 at 9:41 am]

   Thanks for all the great work in both libraries, I'll rewrite it in assembly for the msp430 together with hardware design . Nice project

   jazz,Saigon

15. *jazz* says:
[May 13, 2012 at 10:07 am]

   wow…i had not seen your ospid project, so you really materialized it ☺
   great

16. *gadg* says:
[June 12, 2012 at 3:00 pm]

   Brett thank you so much for this library it has helped me out sooo much

   saved me a lot of work and fantastic it comes from a pid professional

   i think you could write a book on pid !

   i have actually decided to stick with arduino as a hardware platform because this library is so good.

   can the code be ported to netduino or gadgeteer ?

17. *yoyomikeyc* says:
[June 24, 2012 at 8:19 pm]

   Hi, Thanks for this great code.

   Question: Wont the following code run off the end of the lastInputs Array?

   nLookBack could potentially be 100, in which case 'i' starts at 99, the end of the array, only for lastInputs[100] to eventually be updated. I would think the array should be declared of size 101 to fix this, yes?

```
for(int8_t i=nLookBack-1;i>=0;i–)
{
double val = lastInputs[i];
if(isMax) isMax = refVal>val;
if(isMin) isMin = refVal<val;
lastInputs[i+1] = lastInputs[i];
}
```

18. *Brett* says:
[June 25, 2012 at 8:02 am]

   good catch. added it as an issue to github: https://github.com/br3ttb/Arduino-PID-AutoTune-Library/issues/4

19. *yoyomikeyc* says:

July 7, 2012 at 10:13 pm

Oh cool. It was a bug– My programming isnt as rusty as I thought.

I had another question that admittedly stems from my poor controls background. ☺

When looking at the autotune algorithm, wouldn't it technically be more correct to compute an AbsMax & AbsMin that comes from the data of the last two peaks/temp cycles? That is, say the algorithm ran for a total of 9 peaks and eventually had an average separation that was sufficiently small, and thus computed Kp, Ki, and Kd. To do so, it would currently use an AbsMax and AbsMin that potentially came from the first or second cycle, which the algorithm decided wasnt close enough in amplitude to each other. Wont that technically lead to somewhat inaccurate results? (does that make sense?)

I ask this because in my system, for reasons that aren't necessary to explain here, the first cycle always has a lot of 'momentum' and thus the first peak is always extremely high, and then all subsequent peaks are roughly the same height. When the algorithm eventually finishes on the 3rd peak, it uses an AbsMax that came from the first peak/first cycle and was thus extremely high. It would seem that for me, my "AbsMax-AbsMin" in FinishUp() would be too large.

Would you say that it would be more correct to compute an AbsMax and AbsMin after the algo is done, based on the final 2 peaks and troughs?

(sorry, I hope that makes sense)
(thanks again for this great code)

20. *Nicholas Spencer* says:
September 6, 2012 at 2:08 pm

Hi Brett,
Do you think running a analysis of a series of step functions is more accurate than this method (especially with an asymmetric 'A' value)? I'm happy to add to the code library but just looking for your opinion on the matter.

Nick

21. *Brett* says:
September 13, 2012 at 7:13 pm

@Nigholas, Autotuning isn't really my area of expertise so I don't think I can speak with any authority on that. how would you analyze the data? if this is to be run on a microprocessor, doing regressive models or something might be a bit computationally intensive.

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

- **Archives**
  - November 2012
  - September 2012
  - July 2012
  - June 2012
  - April 2012
  - March 2012
  - January 2012
  - December 2011
  - October 2011
  - September 2011
  - August 2011
  - July 2011
  - June 2011
  - May 2011
  - April 2011
  - September 2010
  - August 2010
  - July 2010
  - March 2010
  - November 2009
  - October 2009
  - September 2009
  - August 2009
  - July 2009
  - June 2009
  - May 2009
