Improving the Beginner's PID – Introduction

In conjunction with the release of the new Arduino PID Library I've decided to release this series of posts. The last library, while solid, didn't really come with any code explanation. This time around the plan is to explain in great detail why the code is the way it is. I'm hoping this will be of use to two groups of people:

- People directly interested in what's going on inside the Arduino PID library will get a detailed explanation.
- Anyone writing their own PID algorithm can take a look at how I did things and borrow whatever they like.

It's going to be a tough slog, but I think I found a not-too-painful way to explain my code. I'm going to start with what I call "The Beginner's PID." I'll then improve it step-by-step until we're left with an efficient, robust pid algorithm.

The Beginner's PID

Here's the PID equation as everyone first learns it:

$$\text{Output} = K_P e(t) + K_I \int e(t)\,dt + K_D \frac{d}{dt} e(t)$$

$$\text{Where}: e = \text{Setpoint} - \text{Input}$$

This leads pretty much everyone to write the following PID controller:

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
void Compute()
{
   /*How long since we last calculated*/
   unsigned long now = millis();
   double timeChange = (double)(now - lastTime);

   /*Compute all the working error variables*/
   double error = Setpoint - Input;
   errSum += (error * timeChange);
   double dErr = (error - lastErr) / timeChange;

   /*Compute PID Output*/
   Output = kp * error + ki * errSum + kd * dErr;

   /*Remember some variables for next time*/
   lastErr = error;
   lastTime = now;
}

void SetTunings(double Kp, double Ki, double Kd)
{
   kp = Kp;
   ki = Ki;
   kd = Kd;
}
```

Compute() is called either regularly or irregularly, and it works pretty well. This series isn't about "works pretty well" though. If we're going to turn this code into something on par with industrial PID controllers, we'll have to address a few things:

1. **Sample Time -** The PID algorithm functions best if it is evaluated at a regular interval. If the algorithm is aware of this interval, we can also simplify some of the internal math.
2. **Derivative Kick -** Not the biggest deal, but easy to get rid of, so we're going to do just that.
3. **On-The-Fly Tuning Changes -** A good PID algorithm is one where tuning parameters can be changed without jolting the internal workings.
4. **Reset Windup Mitigation -**We'll go into what Reset Windup is, and implement a solution with side benefits
5. **On/Off (Auto/Manual) -** In most applications, there is a desire to sometimes turn off the PID controller and adjust the output by hand, without the controller interfering
6. **Initialization -** When the controller first turns on, we want a "bumpless transfer." That is, we don't want the output to suddenly jerk to some new value
7. **Controller Direction -** This last one isn't a change in the name of robustness per se. it's designed to ensure that the user enters tuning parameters with the correct sign.

Once we've addressed all these issues, we'll have a solid PID algorithm. We'll also, not coincidentally, have the code that's being used in the lastest version of the Arduino PID Library. So whether you're trying to write your own algorithm, or trying to understand what's going on inside the PID library, I hope this helps you out. Let's get started.
Next >>

UPDATE: In all the code examples I'm using doubles. On the Arduino, a double is the same as a float (single precision.) True double precision is WAY overkill for PID. If the language you're using does true double precision, I'd recommend changing all doubles to floats.

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:00 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.

55 Responses to "Improving the Beginner's PID – Introduction"

Newer Comments »

1. *Anil* says:
   April 18, 2011 at 2:52 pm

   Hello Brett,
   I have been searching a lot for PID basics and how to code it. This is the best material I have discovered.I am doing a project on speed control of DC motor.
   Now I want to following your instructions here on the page and want to see myself why I need the improvements that you have suggested.I will post the feedback on my way.thanks

2. *Anil* says:
   April 19, 2011 at 12:10 am

   Aha!! my PID seems working now..Lot of things start making sense.
   One quick thought here and I would expect feedback from the author that the- errSum should be "static double" not just "static"
   otherwise everytime u call the function the variable errSum would have destroyed its previous value.
   thanks
   Anil

3. *Brett* says:
   April 19, 2011 at 6:54 am

   Anil, What programming language are you using? generally declaring errSum at the class level (the way it's written above) makes the value persist in the way you are describing.

4. *Mini Ultra 8 MHz – Current Consumption (Part 2) |* says:
   April 25, 2011 at 1:27 pm

   [...] the weekend, works on the PID control of the reflow oven controller has progressed extremely well (thank you Brett for helping out) and now we get to complete the testing on Mini Ultra 8 [...]

5. *Greg* says:
   May 1, 2011 at 10:40 am

   Brett,
   Fantastic description, 3 weeks ago PID just meant process identifier to me. Now, you've opened up a whole new world. Thanks for the clear and insightful definition.

   Oh, _raising hand_ .. I have a system which has an very high lag on the input. How would you incorporate high lag?

   Regards,
   Greg.

6. *Brett* says:
   May 1, 2011 at 12:37 pm

   Thanks Greg. as far as lag… I guess it depends what you mean by lag. if the process is just slow, then conservative tuning parameters will be all you need. If, on the other hand, you have a process where you change the output, then wait for awhile, then all of a sudden the Input takes off… pid alone may not be enough. You might be able to get away with just using conservative tunings, but this is where things like feed forward control come into play. feed forward is outside the scope of this series.

   if you'd like to talk specifically about your project, you can shoot me an email. Even better would be to post on the newly created diy-pid-control google group.

7. *Curtis Brooks* says:
   May 1, 2011 at 8:10 pm

This is a great article. I've been working on a reflow controller and intended to use time proportional control. I've been reviewing your PID RelayOutput example and was wondering if you could explain your example a bit. More specifically, the section where the PID output is used to determine when the relay is turned on or off. I'm not understanding how that portion of the routine is determining on time vs. off time.

Thanks,
Curtis

8. *Greg* says:
May 19, 2011 at 1:19 am

Thanks Brett, I've posted on the site you suggested.

Oh, I just noticed a little bug above – probably just a typo
But, in SetTunings the Kd input parameter is getting assigned to itself…
My assumption is you wanted to have

kd = Kd not Kd=Kd

Regards,
Greg

9. *Brett* says:
May 19, 2011 at 7:06 am

Thanks Greg! It's been fixed.

10. *QXS5264* says:
July 12, 2011 at 10:55 pm

very good!

11. *Demystifying PID Control with a look at the new Arduino PID library - Hack a Day* says:
July 21, 2011 at 5:02 pm

[...] [Brett] sent us a note about his work on the new Arduino PID library. He is the author of the original library and recently decided it was time for a ground-up rewrite. But along the way he took the time to explain PID control and the choices he made during development. [...]

12. *bogdanm* says:
July 22, 2011 at 2:35 am

Thanks for an excellent article. By far the best PID material I've ever read.

13. *pat* says:
July 22, 2011 at 6:23 am

Thanks very much for the work on the library, and even more for the explanation & documentation.

I am wanting to build a kiln controller, where I will need to ramp the temperature up from 150C to 700C over 5 hours, then keep it at 700 for 2 hours (plus a few more steps, but you get the picture).

Thanks to your work I think I understand how to implement the keep stable part of the process, but I am not sure about the ramping up temperature. Is it as simple as calculating that at T+10mins I need to be at say 160C, T+20 170C etc and then plugging the new temperature values in to the PID function? This feels wrong, but I am not sure why.

I would appreciate any assistance you are able to give me (including what I am trying to do is actually called).

Pat

14. *Brett* says:
July 22, 2011 at 7:08 am

pat, it feels so wrong but it’s oh so right.

What you’re referring to is called Setpoint Ramping, and it’s done all the time in industry. Just compute a new setpoint every minute, or every 10 seconds, whatever. The PID will follow, trying to make the temperature equal to the setpoint you gave it. over at rocketscream they used this aproach to get a sexy reflow curve

as a side note. feel free to join the diy pid control google group. it’s a little more conducive to Q&A.

15. *jojo* says:
July 22, 2011 at 1:50 pm

Very nice lib !
But if you do he math from the laplace form to the iteration one, you will see that the Integral term should be calculated with the PREVIOUS errorSum.
You should add the error to the errorSum AFTER you compute the output.

16. *Brett* says:
July 22, 2011 at 2:47 pm

@jojo interesting. I’m a weird controls engineer in that I don’t really do much work in the laplace domain (perish the thought.) it might be interesting to pit the two methods against each other to see the difference

17. *pat* says:
July 23, 2011 at 4:52 am

Thanks, I will check out the google group.

18. *Andrew* says:
July 23, 2011 at 5:46 pm

You, sir, are a gentleman and a complete badass.

I wrote a PID algorithm last year for a hotplate I built, and ran into every single one of these problems. I solved a few of them, but you covered solutions to all of them here in incredible detail.

Thank you!!

19. *Brett* says:
July 23, 2011 at 5:52 pm

@Andrew YOU sir, win the make me laugh comment award. I only wish I had done this a year ago so I could have saved you some trouble.

20. *Brett* says:
July 24, 2011 at 9:23 am

@jojo I was intrigued by your suggestion and did a few tests. I've posted the results here

21. *Jason* says:
July 30, 2011 at 6:11 am

Hi Brett, excellent article!
However, can you provide an example of what setpoint/input and output could mean. I am having quite a hard time following the graphs without some actual units/numerical values.
Take temperature for example, would the variable Input be the ADC result from sampling the sensor? Do we then compute in terms of voltages or temperature?
What would the Output be then? Some values to control the PWM or DAC??
What units would that be in?

22. *Brett* says:
July 30, 2011 at 6:49 am

@jason I toyed around with putting units on the graphs but ultimately decided against it. One of the beauties of PID is that it's so general. The setpoint, input, and output can be any type analog signal (provided the setpoint and input are on the same scale) Using a thermocouple input as an example. You could feed the raw chip output into the PID Input, OR you could calculate the actual temperature value and feed THAT into the PID. both would work. You'd need different tuning parameters because the magnitudes would be different, but you could get the same performance from either.

On the output side you are correct. The pid sends out an analog value which in most cases you must convert to the largely digital language of microprocessors. This can be a PWM, DAC, or if PWM is too fast for you (if you're using a relay or SSR,) you can compute something like "Relay On Time" for a given window.

If you have a specific application in mind, I recommend posting to the diy-pid-control google group. We can flesh things out there.

23. *Kasun* says:
August 9, 2011 at 3:31 am

Would love to see someone doing a similar on a AVR mcu using C (gcc).

24. *Luqasek* says:
[August 19, 2011 at 11:00 am]

Hello Brett!
Excellent job! In my project PID library works almost fine! But i have one problem: when my Setpoint = 0 and Input > 0 then Output is > 0, but when Input < 0 then Output is always = 0. I need it to work in both directions. Please help me!

25. *Brett* says:
[August 19, 2011 at 11:35 am]

@Luqasek
You need to use the [SetOutputLimits] function. By default the library Output is limited to 0-255 (Arduino PWM limits.) Using SetOutputLimits you can change the limits to whatever you like

26. *Luqasek* says:
[August 22, 2011 at 5:58 am]

Thank you very much! It was so simple! I do not know why I had not thought about that before:) Now my project works almost fine! I have to try miscellaneous settings of Kp Ki and Kd. Studies taught us about the selection of the optimal PID controller settings, but it's not that easy and I do not know how I would do it. I remember a little method called Ziegler–Nichols. Do you have any tips for me? Thanks again!

27. *Darryl* says:
[October 8, 2011 at 9:49 am]

I've been looking for this and can't thank you enough. Your code will be used in my Quadcopter for stabilization. Thank you so much for publishing this.

28. *ginny* says:
[October 13, 2011 at 2:57 am]

sir/mam,
i am working on project temperature controller using PID algorithm. sir in my project i am clear with the hardware, but m getting struck in the programming part.
i am supposed to give the output of this pid algorithm to the DAC which will pass it's signals to the fan or heater depending upon the present temperature .
i am confused as in how will this output Output = kp * error + ki * errSum + kd * dErr; will go to DAC .
pls help
regards n thanks in advance
ginny 😃

29. *Brett* says:
[October 13, 2011 at 10:06 am]

The Output value will need to be written directly to the DAC. How to do that is outside my experience, and outside the scope of this post. I would suggest searching for: "how to write a float (or double) to DAC" then use that information to send the value of Output.

30. *Engineer* says:
November 14, 2011 at 12:27 pm

Hey Brett,

Thank you for your nice explanation and documentation of your code. It really helped clear out a lot of things.

I am only wondering why at the end of your code, you defined the PID parameters as: kp=KP, ki=Ki, kd=Kd? What is the use behind it although we have kp, ki, kd as the parameters in the output equation.

Thanks

Engineer.

31. *Mr.P* says:
December 11, 2011 at 11:06 am

Hi Brett,

Thank you so much for your nice explanation, I have cleared in my understand PID controller implement.

Thanks again.
Mr.P

32. *Bill* says:
December 28, 2011 at 12:35 pm

Hello.

It is a wonderful tutorial. However, I have a question.

I am working to build (as many others) a sous vide machine.

The heating element of the water is an immersion heater, you just plug it to the wall power jack. In this case, I will connect it with a SSR (Solid State Relay). The power in Greece, is 220 Volts, 50 Hz, so, a full period is 20 mseconds.

When I want to reduce the temperature of the water, I will not allow the power to be transfered to my heater for the full period (I will control the SSR with a microcontroller). When I want to raise the temperature, I will allow the power to be transfered to my heater for a bigger amount of time than before.

The problem is that there is not a standard way to determine the output of the temperature when the immersion heaters are ON for the full period. It largely depents on the volume of the liquid and on many other factors.

So, how will I translate the output of the PID to a time between 0 and 20 mseconds?

I am afraid that I didn't make my question clear. Please ask what you did not understand.

Thanks,
Bill.

33. *Brett* says:
December 28, 2011 at 1:17 pm

Hi Bill, take a look at the Relay output example included with the arduino pid library. that should give you the code you need to convert the analog output into a pulse duration.

34. *Bill* says:
December 28, 2011 at 4:13 pm

Thanks for the quick answer.

I think I understand now but I am still a bit confused. I will go over your tutorial once more and if I have questions I will ask again.
But I think that I won't have problems.

Thanks again,
Bill.

35. *Bill* says:
December 29, 2011 at 7:31 am

Hello again. I am back with more questions!

Two things I don't understand from the previous example.

1. The choice of the 5000 is random. How is this selected? I mean, I have a couple of ideas, but they all include experimentation.

2. The system seems to be working relative with the current PWM. How will I set the first value when it starts? Again, experiments to determine which is the best value?

Also, my PWM will be from 0 to 20 mseconds, so it is quite fast. My immersion heater is not scaled and I don't know of a proper way to scale it myself. Again, I can play with experiments. But, what if I want to produce many of them? I cannot play with every one of my machines. I need a standard way.

For example, if my heater is powerful and I set the mid value… in the middle, let's say from the 256 steps total, I will have it in 128, it may be too strong for my purposes, how will the PID compensate for that? Is't this going to be way to complicated? Like working in another control system, far more difficult than PID?

I hope my question is clear now.

Thanks again,
Bill.

36. *Doc Pedant* says:
January 6, 2012 at 8:36 pm

Notation nit-picking:

“Where e = Setpoint – Input” should instead be “Where: Error(t) = Setpoint – Input(t)”

The notation “e(t)” hasn’t been pre-defined, and is unnecessary in any case.

It should be made clear that the input depends on time, but the Setpoint doesn’t. Well, Setpoint can depend on time, but it is pre-programmed, unlike the input.

- Doc Pedant

37. *Stefan Weber* says:
January 17, 2012 at 5:46 am

Hey Brett or anybody !

Have you guys been doing some PID-Tn controllers ? My idea would be to discretize the ODE with Euler but I am not sure if that would be the best idea.

Maybe somebody can recommend some good literature ?

Greets !

38. *Brett* says:
January 18, 2012 at 9:03 am

@Stefan, I’ve only ever done this once, and when I did I used the method you describe.

39. *Stefan Weber* says:
January 23, 2012 at 4:46 am

Ok thanks ! I’ll probably do it with c2d() in matlab – seems to be working fine enough – altough its everything else but elegant ;)

40. *Muthu* says:
February 13, 2012 at 1:23 am

Great, Hope Continue such sort of work dude.

41. *bennyhuang* says:
February 24, 2012 at 1:55 am

Thanks,but i guess that your program is not suitable for use in the computer programing.In your program,it calculate the time between now and last time we run the program;but we usually call the program periodically instead of calculating the time .

42. *Brett* says:
[February 24, 2012 at 9:28 am]

@bennyhuang I'm confused. did you look at that's the fist issue I address. This page is "the beginner's pid" I make many improvements over the course of the series.

43. *Ryan* says:
[February 24, 2012 at 2:25 pm]

Hey Brett,

This is a very good article and covering exactly what I was looking for. I am building a PID temperature control system with Arduino, but am interested in learning how the sketch works. The input for this system is temperature (obviously) and the output is fan speed, but I was wondering what this compute() function actually outputs to the pins on Arduino. I assume it is just a voltage, but I am wondering how the system is able to convert a temperature reading to a voltage. Also, I am using PWM to control the fans which take 12V, if that matters.

44. *Brett* says:
[February 24, 2012 at 2:46 pm]

@Ryan, strictly speaking, both the input to and output from the pid are numbers (the output is limit to 0-255 by default.) how these numbers get to/from the pid are unknown to it. the most likely setup for your system would be:
Input = analogRead(inputPin); //this converts 0-5V on the pin into an int between 0-1023
pid.Compute();
analogWrite(outputPin, Output); //this takes the output from the pid (0-255) and converts it to a 5V pwm signal.

this is the simplest setup. if you wanted to convert the input value to the actual temperature (or something else) before feeding it into the pid that's fine, just be advised that the optimal values of kp, ki, and kd will be different then the non-converted system.

45. *dav* says:
[April 3, 2012 at 1:34 am]

hi, thanks for the article!!

the first time i understand how to use the [t] in such formulas.

1.can you explain more the way you convert the output to on/off relay?

2.there is a way to look inside the function like "SetOutputLimits()"?

I'm using c and c++ so i have to convert the code.

thanks alot,

David

46. *dav* says:
    April 3, 2012 at 1:36 am

    correction ; I'm using c and NOT C++

47. *dav* says:
    April 13, 2012 at 6:37 am

    hi ,

    thanks for your article.

    i have 2 questions , can you explain if you call the compute() every x time or it's ruining all time?

    and about the time var. , this var become zero and start all over again, how do you prevent "now – last_time" become negative?

    thanks

    david

48. *Ven* says:
    April 23, 2012 at 4:15 pm

    I'm told by my professor that if I put my finger on the spinning DC motor, and the value of rpm changes then I should have the microcontroller (Arduino Uno) to apply PID to readjust the rpm back to its normal regular speed…

    For example…
    Like let's say the rpm of a motor goes to 12000 rpm and then I add my finger to the dc motor and it reads 8000 rpm and so my professor told me to have PID applied so that even if I put my finger into the dc motor, the PID will take care and have the dc motor go back to original 12000 rpm.

    how would I make this to be?

    I wrote a pseudocode below to show what I mean…

    if interruption on spinning dc motor occurs
    then calculate PID on rpm

    how would I make this happen?

    thanks.

49. *Brett* says:
    April 23, 2012 at 8:46 pm

    @ven actually, that pseudo-code should happen when you're designing the system. if the motor isn't strong enough to handle the load of your finger, then you should set up your system to continuously use PID. most of the time it won't be doing much ("everything's fine, keep the output constant") but when someone applies a finger it will automatically respond and increase power to the motor.

50. *akshay kumar vyas* says:
[April 25, 2012 at 2:32 pm]

i do understand a bit, but i have a few questions.

lets take temperature control for example where a situation arises to keep the temperature constant. a heating coil is used which is controlled by a TRIAC. if the set point is 200 and input is 198. then error is 2. then the output is calculated by the formula Output = kp * error + ki * errSum + kd * dErr. but how is the output given to the TRIAC. i mean in what form. how is this done.

1. *Brett* says:
   April 25, 2012 at 4:05 pm

   @akshay this output can be passed to PWM logic. on the arduino however, the PWM is too fast for a triac (assuming 60hz mains,) so you need to write your own code that implements a slower version of PWM: http://arduino.cc/playground/Code/PIDLibraryRelayOutputExample

2. *Ven* says:
   April 28, 2012 at 12:27 am

   hi, is the code above in this website (under beginner's PID) a code for arduino I can use for PID controller for DC motor. Basically I'm trying to get PID values of a spinning DC motor and I'm using IR sensor to get the rpm of the DC motor… whenever I put my finger on the DC motor, the rpm values will change thus allowing to get outputs of PID values to control the DC motor. so is this the code that allows me to do that?

3. *Brett* says:
   April 28, 2012 at 8:26 am

   @ven you CAN use this code, but as I explain it has some deficiencies. if you use the Arduino PID Library you will get a more robust controller.

4. *Ven* says:
   April 29, 2012 at 1:00 am

   it seems like my comment did not go through this website. I wanted to ask few questions… well more like I have a problem here. I tried to use your code to print out the PID when I apply physical interruption on DC motor (severe change of rpm) but it seems like I"m not getting any PID values. How can I be able to solve this? I went to arduino library website and used some of basic output examples with still no results. Is there a way for me to send you my current code so that you can take a look at it to see what I'm doing wrong?

5. *akshay* says:
   July 26, 2012 at 10:00 am

   sir,
   you said that the PID output can be passed to the PWM logic. but can u please tell me what happens in the PWM logic . is it so that the PID value is multiplied with the duty cycle of the PWM. please explain . i just want to know how the PID output reduces or increases the PWM pulse width.

Improving the Beginner's PID – Sample Time

(This is Modification #1 in a larger series on writing a solid PID algorithm)

The Problem

The Beginner's PID is designed to be called irregularly. This causes 2 issues:

- You don't get consistent behavior from the PID, since sometimes it's called frequently and sometimes it's not.
- You need to do extra math computing the derivative and integral, since they're both dependent on the change in time.

The Solution

Ensure that the PID is called at a regular interval. The way I've decided to do this is to specify that the compute function get called every cycle. based on a pre-determined Sample Time, the PID decides if it should compute or return immediately.

Once we know that the PID is being evaluated at a constant interval, the derivative and integral calculations can also be simplified. Bonus!

The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      errSum += error;
      double dErr = (error - lastErr);

      /*Compute PID Output*/
      Output = kp * error + ki * errSum + kd * dErr;

      /*Remember some variables for next time*/
      lastErr = error;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   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    }
```

On lines 10&11, the algorithm now decides for itself if it’s time to calculate. Also, because we now KNOW that it’s going to be the same time between samples, we don’t need to constantly multiply by time change. We can merely adjust the Ki and Kd appropriately (lines 31 & 32) and result is mathematically equivalent, but more efficient.

one little wrinkle with doing it this way though though. if the user decides to change the sample time during operation, the Ki and Kd will need to be re-tweaked to reflect this new change. that’s what lines 39-42 are all about.

Also Note that I convert the sample time to Seconds on line 29. Strictly speaking this isn’t necessary, but allows the user to enter Ki and Kd in units of 1/sec and s, rather than 1/mS and mS.

The Results

the changes above do 3 things for us

1. Regardless of how frequently Compute() is called, the PID algorithm will be evaluated at a regular interval [Line 11]
2. Because of the time subtraction [Line 10] there will be no issues when millis() wraps back to 0. That only happens every 55 days, but we’re going for bulletproof remember?
3. We don’t need to multiply and divide by the timechange anymore. Since it’s a constant we’re able to move it from the compute code [lines 15+16] and lump it in with the tuning constants [lines 31+32]. Mathematically it works out the same, but it saves a multiplication and a division every time the PID is evaluated

Side note about interrupts

If this PID is going into a microcontroller, a very good argument can be made for using an interrupt. SetSampleTime sets the interrupt frequency, then Compute gets called when it’s time. There would be no need, in that case, for lines 9-12, 23, and 24. If you plan on doing this with your PID implentation, go for it! Keep reading this series though. You’ll hopefully still get some benefit from the modifications that follow. There are three reasons I didn’t use interrupts

1. As far as this series is concerned, not everyone will be able to use interrupts.
2. Things would get tricky if you wanted it implement many PID controllers at the same time.
3. If I’m honest, it didn’t occur to me. Jimmie Rodgers suggested it while proof-reading the series for me. I may decide to use interrupts in future versions of the PID library.

Next >>

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:01 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 "Improving the Beginner's PID – Sample Time"

1. *Coding Badly* says:
   April 16, 2011 at 3:27 am

   I think moving the time calculations out of Compute is a good idea but it does create a potential pitfall for new users. The calculations will not be accurate if more than SampleTime has elapsed between calls. This could be difficult (or impossible) to debug. I suggest adding a check for "(grossly) exceeded SampleTime". If the SampleTime has been exceeded, set a flag. This gives the user a convenient way to check for an overrun.

2. *Brett* says:
   April 16, 2011 at 7:48 am

   That's definitely a issue that can arise. I hate flags, but that may be the best solution. I'll have to think about it some more. Until then we'll just have to keep our eye out for people reporting suspicious performance degradation. "…then I set my sample time to 1mS" AHA!

3. *prashant* says:
   July 16, 2011 at 1:33 pm

   how much is ur blog applicable for a line follower robot (with 5 digital sensors) as it is?
   i mean wat changes we may hv to do ? apart from our own algorithm language n al?

   to be specific i hv 3 major questions
   1) is SetTunings method required for us?i mean til now we thot kp,ki,kd are constants and we ll give them values?n they shouldnt change their values in between right?

   2)how exactly to call compute method after certain time (say 1ms) from main ?
   u hvnt specified it on how to call it from main? n is millis() method used for it?

   n lastly

   3)how we relate output/correction with speed of motors (2)? we know that we ll hv to use PWM. we know how to do it. but we want ur word on this.

   we ll be grateful for ur help. ty

4. *somun* says:
   July 21, 2011 at 4:07 pm

Thanks for the write up Brett!

One point that I have trouble understanding is the 2nd result you mention. Looks to me that your argument regarding being safe for timer wrapping to 0, does not hold. Say the lastTime is just 1 second below wrapping to zero and now() returns, say 2. Than timeChange would be a negative number which would evaluate to false in the if statement. What am I missing?

5. *Brett* says:
[July 21, 2011 at 4:30 pm]

Somun, the trick is that now is an UNSIGNED long. so rather than having negative numbers it uses that space to allow for larger positive numbers. when you subtract 5 from 2 say, the result isn’t -3, it’s 4,294,967,292.

6. *somun* says:
[July 22, 2011 at 12:57 am]

I think that’s not the case unless timeChange is defined as unsigned, too (it’s an int in the code). Just tried on Visual Studio.

7. *Brett* says:
[July 22, 2011 at 6:59 am]

ah. with a different programming language your mileage may vary. thanks for the rigorous testing!

8. *David* says:
[July 22, 2011 at 11:27 am]

Jimmie Rogers is absolutely right about using interrupts. Calling from the main loop will only give you consistent timing if you have plenty of processing margin. For robustness you really want to make sure that any time critical code (such as a control loop) is called from a timer interrupt. None time sensitive code can then be left in the main loop and it doesn’t matter if it overruns.

You are correct though; it is much, much harder to have multiple PIDs without hard-coding them in.

9. *Brett* says:
[July 22, 2011 at 11:40 am]

Yeah I was thinking the best solution might be to have a common interrupt set to the lowest common denominator of the sample time of the various pid loops, then using that to decide which pid needed to be computed when. But I just don’t have that kind of AVR skill. I felt I needed to include that though, because somebody is reading this right now and saying “well that’s easy! let me whip that up”

10. *somun* says:
[July 22, 2011 at 12:52 pm]

@Brett. Well last time I checked Visual Studio was an IDE 😃

Anyway, it is just C/C++ behavior (Arduino compiler is a branch of gcc). When that non-negative big number is put into an int, it becomes negative due to the way 2′s complement works. Please go ahead and verify if you don't believe me. 'timeChange' needs to be unsigned.

Sorry for insisting on a small issue in a great write-up but I guess I am just trying to contribute to the bullet-proofness 😃

11. *Eric Anderson* says:
July 22, 2011 at 1:06 pm

Regarding result 2:

If timeChange can become negative at wrap time due to conversion from unsigned to signed as stated above, the code will not just miss computing a pid, the code would permanently stop adjusting the output value.

If timeChange is just made large when time wraps back to 0, then you may have an iteration where the output changes on a short cycle. Exactly how short would depend on the relationship between sampleTime and the the size of long. (ie. what is MAXLONG % sampleTime).

I agree that an interrupt based solution is better, to remove or simplify the time keeping code in the PID.

12. *Brett* says:
July 22, 2011 at 1:15 pm

interesting. I tested this feature by making now = millis()+ 4294960000 and letting it run, and there were no issues.

wait. I think I know what happened. the timeChange variable was added for clarification during this explanation. initially I just had the subtraction inside the if statement. that would do it. good call guys. will need to change that to an unsigned long in the next iteration of the library.

13. *Eric Anderson* says:
July 22, 2011 at 2:35 pm

How did you check? If the value goes negative, what you would see is that the output would stick on a value. Otherwise, you will get one output generated that is out of sync with your time interval (and the subsequent updates would be at the correct interval, but not in line with the previous updates).

It all depends on how closely you need to track the time based part of the PID.

With the code as it is (ignoring the wraparound issue), it assumes but does not force regularity. The logic says update the output if at least time interval has passed, but it computes the PID as if exactly time interval has passed.

This may or may not be an issue depending on what it is you are controlling and how close the actual time elapsed is to the time interval.

B.T.W. – I really do appreciate this PID overview. I am always on the look out for such things to help with training I do for FIRST robotics teams.

14. *Krille* says:
September 19, 2011 at 6:17 am

Nice…
But I think if you add an "Window value" the library will be perfect
If you use an servo the regulating of the servo can damiched the feedback potentiometer if you regulate to often.. So if I can have a which I wold love to have a window value… Else I will add it by my self.. 😃

Best regards
Krille

15. *Jason C* says:
January 2, 2012 at 10:59 am

While on the topic of sampling time, I am wondering what are the effects different sampling time have on the performance of the PID control algorithm. e.g. Do you like reach stabilization faster with a shorter sampling time?

16. *Brett* says:
January 2, 2012 at 11:13 am

@Jason, if your sampling time is too slow it will certainly impact loop performance. as it gets faster, you will notice improvement up to a point. after this the performance will be virtually identical regardless of sample time. the value at which this occurs will be different for each system.

17. *Thomas* says:
January 3, 2012 at 3:14 pm

Hello Brett,

Thank you very much for writing such a great library and tutorial! I suppose in line 10 the expression will become negative when the overflow of millis() occurs:
int timeChange = (now – lastTime) 0:
if(timeChange>=SampleTime)
Maybe you had an unsigned long for timeChange in mind, then it should work out:
unsigned long timeChange = (now – lastTime) >0
Cheers, Thomas

18. *Brett* says:
January 3, 2012 at 3:18 pm

@Thomas Yup. there's a discussion to that effect a little further up the comment chain. The PID Library code has been updated

19. *Thomas* says:
January 3, 2012 at 3:18 pm

Sorry, there was missing something in my last comment. There should be written:
I suppose in line 10 the expression will become negative when the overflow of millis() occurs:

int timeChange = (now – lastTime) <0
In this case the execution of the controller in line 11 will never happen, because 0=SampleTime)
Maybe you had an unsigned long for timeChange in mind, then it should work out:
unsigned long timeChange = (now – lastTime) >0
Cheers, Thomas

20. *pid_hobby* says:
February 5, 2012 at 5:19 am

We can merely adjust the Ki and Kd appropriately (lines 30&31) and result is mathematically equivalent, but more efficient.

error lines 30& 31

to :
lines 31 & 32

21. *Brett* says:
February 5, 2012 at 8:37 am

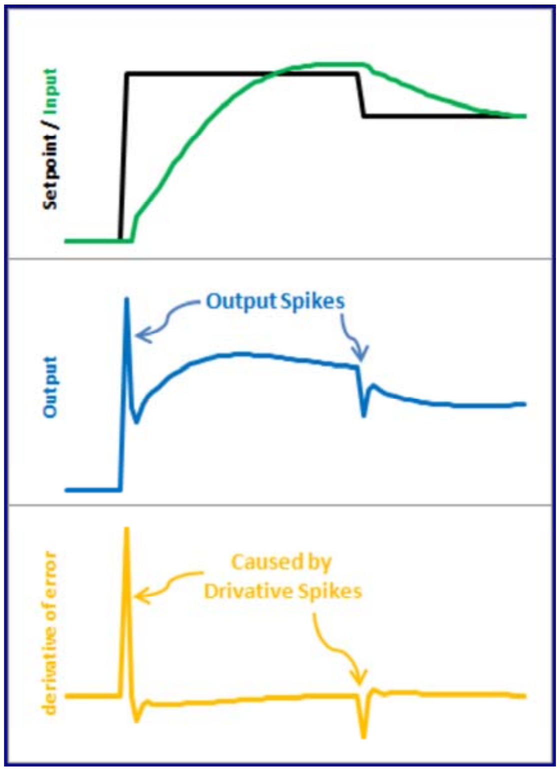
@pid_hobby fixed thanks

Improving the Beginner's PID – Derivative Kick

(This is Modification #2 in a larger series on writing a solid PID algorithm)

The Problem

This modification is going to tweak the derivative term a bit. The goal is to eliminate a phenomenon known as "Derivative Kick".



The image above illustrates the problem. Since error=Setpoint-Input, any change in Setpoint causes an instantaneous change in error. The derivative of this change is infinity (in practice, since dt isn't 0 it just winds up being a really big number.) This number gets fed into the pid equation, which results in an undesirable spike in the output. Luckily there is an easy way to get rid of this.

The Solution

$$\frac{d\text{Error}}{dt} = \frac{d\text{Setpoint}}{dt} - \frac{d\text{Input}}{dt}$$

When Setpoint is constant :

$$\frac{d\text{Error}}{dt} = -\frac{d\text{Input}}{dt}$$

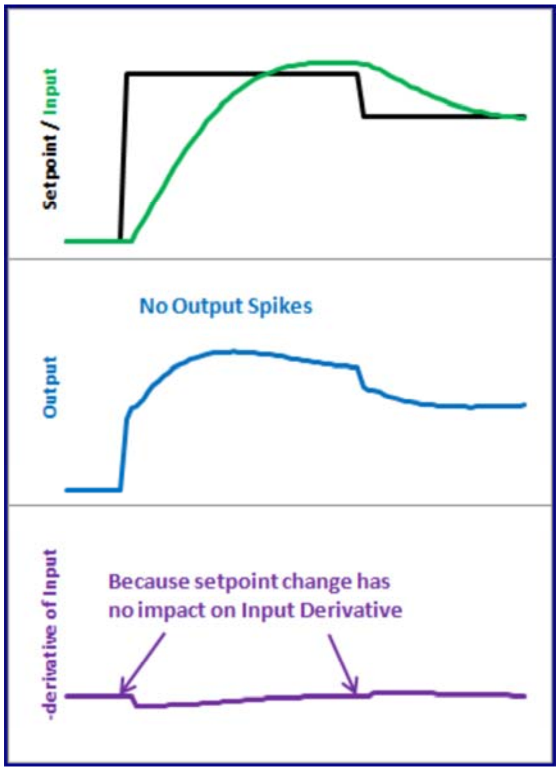
It turns out that the derivative of the Error is equal to negative derivative of Input, EXCEPT when the Setpoint is changing. This winds up being a perfect solution. Instead of adding (Kd * derivative of Error), we subtract (Kd * derivative of Input). This is known as using "Derivative on Measurement"

The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      errSum += error;
      double dInput = (Input - lastInput);

      /*Compute PID Output*/
      Output = kp * error + ki * errSum - kd * dInput;

      /*Remember some variables for next time*/
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}
```

The modifications here are pretty easy. We're replacing +dError with -dInput. Instead of remembering the lastError, we now remember the lastInput

The Result



Here's what those modifications get us. Notice that the input still looks about the same. So we get the same performance, but we don't send out a huge Output spike every time the Setpoint changes.

This may or may not be a big deal. It all depends on how sensitive your application is to output spikes. The way I see it though, it doesn't take any more work to do it without kicking so why not do things right?
Next >>

Flattr this!

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:02 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.

9 Responses to "Improving the Beginner's PID – Derivative Kick"

1. *Coding Badly* says:
   April 16, 2011 at 3:35 am

   > This number gets fed into the pid equation, which results in an undesirable spike in the output.

   Obviously, this is true for some (most?) applications but it is not universally true. I worked on a controller for an electric heater that required a rather "aggressive" derivative. Dramatic changes in the output were not only acceptable, they were required.

I've seen two methods for getting roughly the same effect: 1. Output rate of change clamping; 2. Output filtering. The advantage is the user can eliminate all spikes, allow some "spikiness", or allow all spikes.

2. *Brett* says:
   April 16, 2011 at 7:35 am

   That's a fair point. It explains why all manufacturers allow you to choose "derivative on error" even if they default to "derivative on measurement." I overstated when I said that looking at measurement is the right way to do things.

   I've seen those output massagers as well. They're a little out of the scope of this series, but highly effective in the right hands. They've been added to the "things I'll try to talk about later" list.

3. *Mandar* says:
   July 5, 2011 at 2:29 am

   Im working on a line follower……..
   Can u help me with dat……….How to define a setpoint if im using Digital Sensors(Five).Is it that the setpoint will always be that the middle sensor should be on the line.
   And then how do i calculate the error and use it for my motors….

4. *Brett* says:
   July 5, 2011 at 8:27 am

   You would need to convert the 5 digital signals into 1 analog signal (-2 < -> 2) I'm not sure if this would give you enough resolution to use a pid effectively, but that is how you would do it.

5. *prashant* says:
   July 6, 2011 at 8:30 am

   hey me and madar are working together

   how much is ur blog applicable for a line follower robot (with 5 digital sensors) as it is?
   i mean wat changes we may hv to do ? apart from our own algorithm language n al?

   to be specific i hv 3 major questions
   1) is SetTunings method required for us?i mean til now we thot kp,ki,kd are constants and we ll give them values?n they shouldnt change their values in between right?

   2)how exactly to call compute method after certain time (say 1ms) from main ?
   u hvnt specified it on how to call it from main? n is millis() method used for it?

   n lastly

   3)how we relate output/correction with speed of motors (2)? we know that we ll hv to use PWM. we know how to do it. but we want ur word on this.

   we ll be grateful for ur help. ty

6. *bjbsquared* says:
   July 22, 2011 at 4:22 pm

   Another thing to consider is that the reference can be changed at a controlled rate. There is no advatantage of instantly changing the set point if the controller can't keep up. It will tend to go "open loop". You can gradually change the setpoint within the PID loop.

   For a positive change:
   Linear approach : if (setPntnow<setpoint) setPntnow + aLittleMore;

   asymptotic : if (setPntnow<setpoint) setPntnow= setPntnow+(setpoint – setPntnow)/someDivisor;

   'aLittleMore' and 'someDivisor' could be any number depending on the wanted rate of change. Of course 2, 4, 8, ect would be a simple shift for 'someDivisor.

7. *Brett* says:
   July 22, 2011 at 4:34 pm

   @bjbsquared while the setpoint ramping you describe makes things more forgiving for difficult processes or less-than-perfect tuning parameters, I think it's overstating a bit to say that there is "no advantage" to instantly changing the setpoint. For many processes and with a properly tuned controller, you can achieve a new setpoint far more quickly by just stepping the setpoint to the new value.

   that being said, setpoint ramping is a valuable technique used often and industry that can save you a lot of tuning work if you have a slow process

8. *Jason Short* says:
   July 25, 2011 at 11:53 pm

   I've found that if the PID is running at a high rate (200 hz) and a sensor that measures the rate of change is running at a lower rate (say 10hz) you'll end up with spikes if your D term is high. We have to run a moving average filter over the derivative to keep that under control or else you'll have a quad copter engine that sounds terrible or other more negative effects.

9. *Brett* says:
   July 26, 2011 at 7:04 am

   @Jason First of all thanks for even being here. I'm a huge fan of the diydrones project. now on to pid. In my experience (which is almost entirely large, industrial PID,) you don't gain anything by running the PID more quickly than the input signal changes. you're asking it to compute a new output without any new information. and yes, you get weird derivative spikes. since the PID is evaluated 20 times for every input change, and the derivative is looking at this point compared to the last… you get 19 0′s then a blip.

   I rarely use filter in my professional life. generally I find that I'm better off removing D than trying to massage the signal to make it work. This requires things to be tuned fairly well however, and it takes some time to get a knack for that.

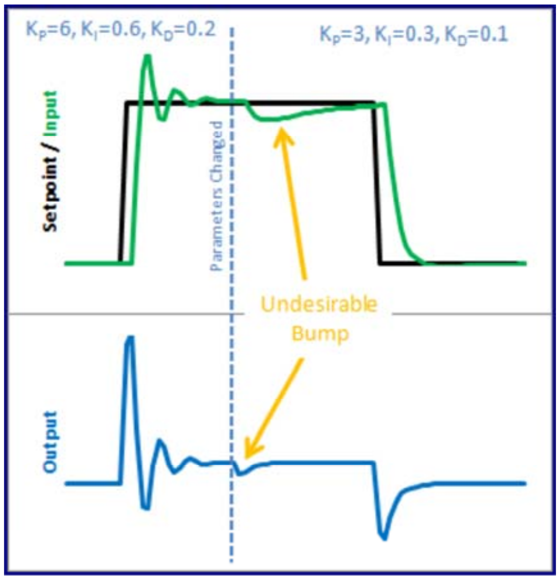
   but as I said, I have very little robotics / motor control pid experience, so there may be advantages I'm unaware of. have you tried running your PID more slowly?

Improving the Beginner's PID: Tuning Changes

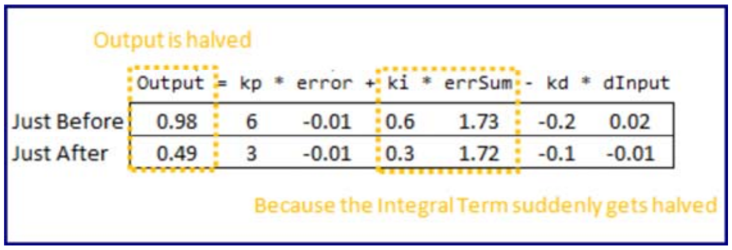
(This is Modification #3 in a larger series on writing a solid PID algorithm)

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:

$$\boxed{\mathrm{K_I}\int e\,dt \approx \mathrm{K_I}_n\left[e_n + e_{n-1} + \ldots\right]}$$

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?)

$$\mathrm{K_I}\int e\,dt = \int \mathrm{K_I}\, e\,dt$$
$$\int \mathrm{K_I}\, e\,dt \approx \mathrm{K_I}_n e_n + \mathrm{K_I}_{n-1} e_{n-1} + \ldots$$

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.

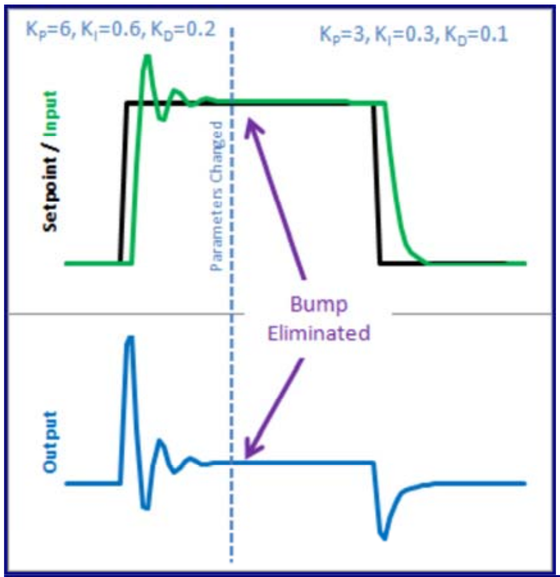
Now, we take the error and multiply it by whatever the Ki is at that time. We then store the sum of THAT. When the Ki changes, there’s no bump because all the old Ki’s are already “in the bank” so to speak. We get a smooth transfer with no additional math operations. It may make me a geek but I think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

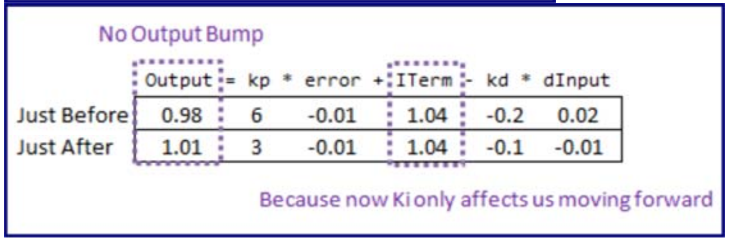
The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      ITerm += (ki * error);
      double dInput = (Input - lastInput);

      /*Compute PID Output*/
      Output = kp * error + ITerm - kd * dInput;

      /*Remember some variables for next time*/
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   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

This entry was posted on Friday, April 15th, 2011 at 3:03 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.

5 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*(Integral of error from t=0 to 20)
   new way: (Integral of (5*error) from t=0 to 10) + (Integral of (10*error) from t=10 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*(Integral of error from t=0 to 20)
   new way: (Integral of (5*error) from t=0 to 10) + (Integral of (10*error) from t=10 to 20)

   Then:

   old way: 10*(Integral of error from t=0 to 20)
   new way: 5*(Integral of error from t=0 to 10) + 10(Integral of error from t=10 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.

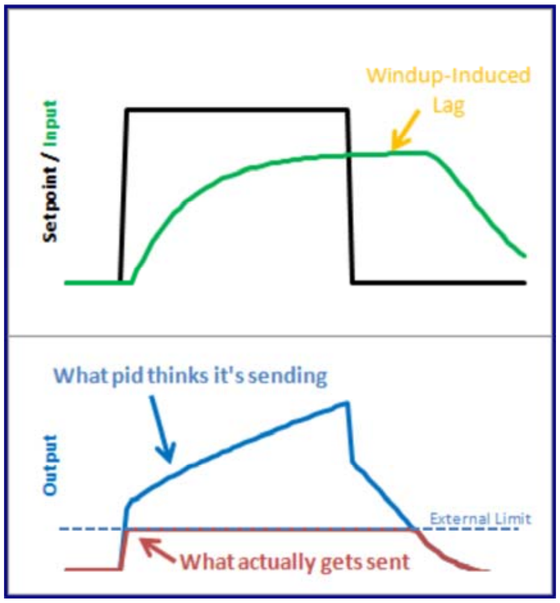
5. *Vasileios Kostelidis* says:
   July 5, 2012 at 7:59 am

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

Improving the Beginner's PID: Reset Windup

(This is Modification #4 in a larger series on writing a solid PID algorithm)

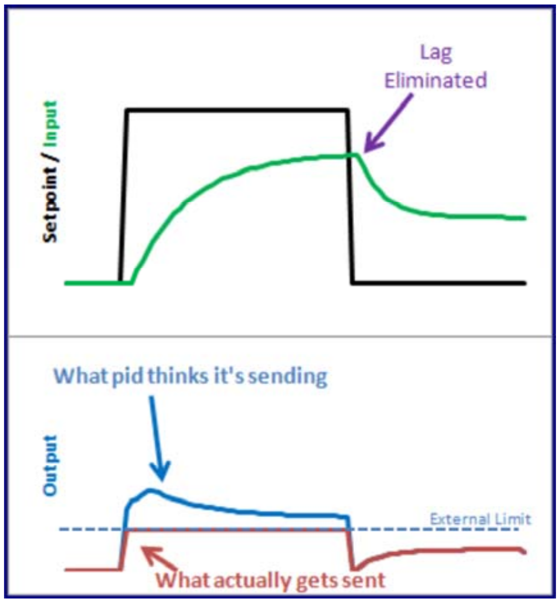
The Problem



Reset windup is a trap that probably claims more beginners than any other. It occurs when the PID thinks it can do something that it can't. For example, the PWM output on an Arduino accepts values from 0-255. By default the PID doesn't know this. If it thinks that 300-400-500 will work, it's going to try those values expecting to get what it needs. Since in reality the value is clamped at 255 it's just going to keep trying higher and higher numbers without getting anywhere.

The problem reveals itself in the form of weird lags. Above we can see that the output gets "wound up" WAY above the external limit. When the setpoint is dropped the output has to wind down before getting below that 255-line.

The Solution – Step 1



There are several ways that windup can be mitigated, but the one that I chose was as follows: tell the PID what the output limits are. In the code below you’ll see there’s now a SetOuputLimits function. Once either limit is reached, the pid stops summing (integrating.) It knows there’s nothing to be done; Since the output doesn’t wind-up, we get an immediate response when the setpoint drops into a range where we can do something.

The Solution – Step 2

Notice in the graph above though, that while we got rid that windup lag, we’re not all the way there. There’s still a difference between what the pid thinks it’s sending, and what’s being sent. Why? the Proportional Term and (to a lesser extent) the Derivative Term.

Even though the Integral Term has been safely clamped, P and D are still adding their two cents, yielding a result higher than the output limit. To my mind this is unacceptable. If the user calls a function called “SetOutputLimits” they’ve got to assume that that means “the output will stay within these values.” So for Step 2, we make that a valid assumption. In addition to clamping the I-Term, we clamp the Output value so that it stays where we’d expect it.

(Note: You might ask why we need to clamp both. If we’re going to do the output anyway, why clamp the Integral separately? If all we did was clamp the output, the Integral term would go back to growing and growing. Though the output would look nice during the step up, we’d see that telltale lag on the step down.)

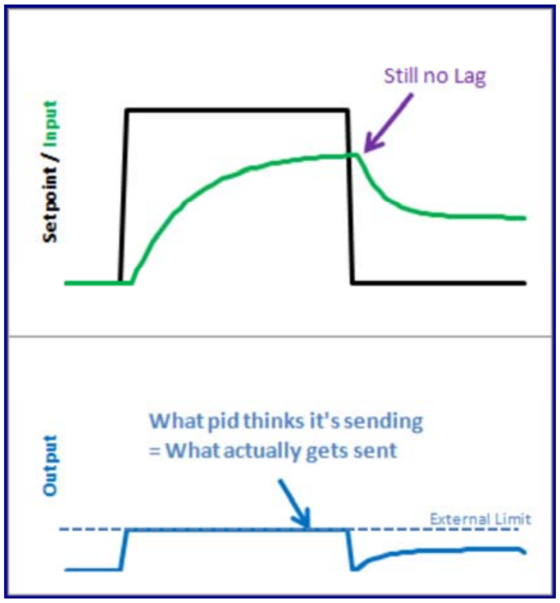
The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
void Compute()
{
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
```

```
14       /*Compute all the working error variables*/
15       double error = Setpoint - Input;
16       ITerm+= (ki * error);
17       if(ITerm> outMax) ITerm= outMax;
18       else if(ITerm< outMin) ITerm= outMin;
19       double dInput = (Input - lastInput);
20 
21       /*Compute PID Output*/
22       Output = kp * error + ITerm- kd * dInput;
23       if(Output > outMax) Output = outMax;
24       else if(Output < outMin) Output = outMin;
25 
26       /*Remember some variables for next time*/
27       lastInput = Input;
28       lastTime = now;
29    }
30 }
31 
32 void SetTunings(double Kp, double Ki, double Kd)
33 {
34   double SampleTimeInSec = ((double)SampleTime)/1000;
35    kp = Kp;
36    ki = Ki * SampleTimeInSec;
37    kd = Kd / SampleTimeInSec;
38 }
39 
40 void SetSampleTime(int NewSampleTime)
41 {
42    if (NewSampleTime > 0)
43    {
44       double ratio  = (double)NewSampleTime
45                       / (double)SampleTime;
46       ki *= ratio;
47       kd /= ratio;
48       SampleTime = (unsigned long)NewSampleTime;
49    }
50 }
51 
52 void SetOutputLimits(double Min, double Max)
53 {
54    if(Min > Max) return;
55    outMin = Min;
56    outMax = Max;
57 
58    if(Output > outMax) Output = outMax;
59    else if(Output < outMin) Output = outMin;
60 
61    if(ITerm> outMax) ITerm= outMax;
62    else if(ITerm< outMin) ITerm= outMin;
63 }
```

A new function was added to allow the user to specify the output limits [lines 52-63]. And these limits are used to clamp both the I-Term [17-18] and the Output [23-24]

The Result



As we can see, windup is eliminated. in addition, the output stays where we want it to. this means there’s no need for external clamping of the output. if you want it to range from 23 to 167, you can set those as the Output Limits.

Next >>

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:04 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.

11 Responses to “Improving the Beginner’s PID: Reset Windup”

1. *Jan Bert* says:
   July 22, 2011 at 6:36 am

   Thank you for your great explanation! What do you think of my suggestion?

   Instead of separately clamping the Iterm, you could limit the Iterm a bit sooner. When the output is bigger than max, there is no point in increasing the Iterm.

   ```
   if(Output > outMax)
   {
   // Anti wind-up: undo integration if output>max
   ITerm -= ki * error;
   // Clamp output
   Output = outMax;
   }
   ```

   You could use a temporary variable for ‘ki * error’ to re-use the previous multiplication.

2. *Brett* says:
   July 22, 2011 at 7:02 am

   That's certainly something you could do. I chose not to go that route because if the output is slammed around by the P and D terms it's possible that it would incorrectly affect how the I term sums.

3. *Will* says:
   July 22, 2011 at 9:15 am

   Another similar trick is to clamp the integral AFTER the output has been computed but before it is clamped. so the math looks something like:

   ```
   …
   iTerm += ki * error;
   …
   output = pTerm + iTerm + dTerm;
   if (output > maxLimit)
   iTerm -= output – maxLimit;
   output = maxLimit;
   else if (output < minLimit)
   iTerm += minLimit – output;
   output = minLimit;
   …
   ```

   The difference is subtle but reduces the windup to zero instead of the really small error that still exists with clamping the iTerm to the limit instead of clamping the output to the limit.

4. *Will* says:
   July 22, 2011 at 9:17 am

   doh! formatting got removed. should look like (underscore -> tab/space):

   ```
   if (output > maxLimit) {
   ____iTerm -= output – maxLimit;
   ____output = maxLimit;
   } else if (output < minLimit) {
   ____iTerm += minLimit – output;
   ____output = minLimit;
   }
   ```

5. *Brett* says:
   July 22, 2011 at 9:20 am

   Will, excellent trick! I think the next update to the library will be using this method

6. *Evan* says:
   July 23, 2011 at 2:47 am

   Really great post. I've been interested in PIDs since I learned about them in an introductory control systems course. Our final project was making a PID very similar to your "beginner's PID" on a PIC. Now, I'm thinking about using PID for an arduino project and I'm glad I've read this, since your code solves three problems I might not have known I had. I'll use this info if I ever get around to my project idea.

7. *Brett* says:
   July 23, 2011 at 6:56 am

   @Evan remember that there's actually a pid library for the arduino, so most of the hard work is already done there. it'd be a shame for you to do it all from scratch, unless it's something you intentionally want to do

8. *Ali* says:
   August 15, 2011 at 2:49 pm

   Hi guys, I am using arduino to design a PI controller to control the flow rate in the system by controlling the pump's output. Currently, I designed PI controller but it has got a delay in responding and the output is not stable. What do you suggest? I have taken the coded here for the pi controller but it did not work at all. does anybody have an Idea?

9. *Brett* says:
   August 15, 2011 at 2:52 pm

   @Ali I recommend posting this question to the diy-pid-control google group. that venue has a better setup for Q&A

10. *Dustyn* says:
    February 6, 2012 at 12:51 pm

    Wouldn't the SetOutputLimits function get much smaller if you used the built in constrain(# to constrain, min, max) function within Arduino? Great code btw! Thanks

11. *Brett* says:
    February 6, 2012 at 1:47 pm

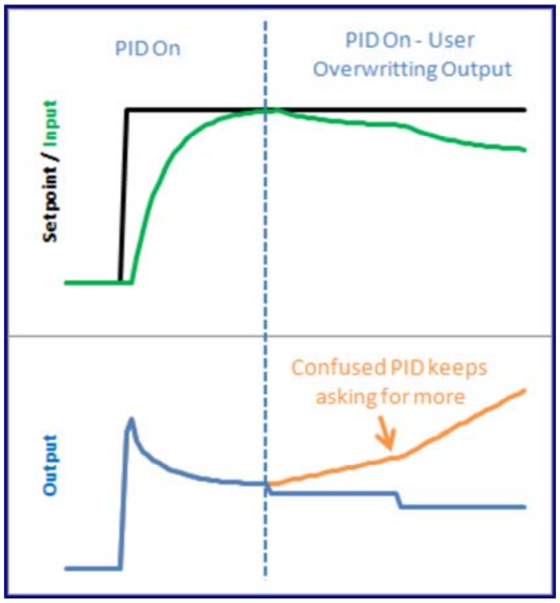
    @dustyn interesting! I was unaware of the constrain function. it may be faster, but I'm guessing that the code inside is probably similar to what I have here. you're also calling a function which I think will slightly increase ram usage. still, something to try!

Improving the Beginner’s PID: On/Off

(This is Modification #5 in a larger series 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:

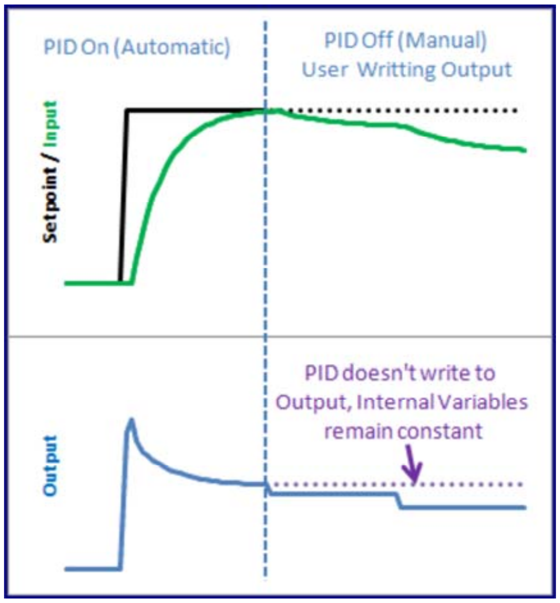
The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1

void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      /*Compute PID Output*/
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;

      /*Remember some variables for next time*/
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
```

```
     else if(Output < outMin) Output = outMin;

     if(ITerm> outMax) ITerm= outMax;
     else if(ITerm< outMin) ITerm= outMin;
  }

  void SetMode(int Mode)
  {
    inAuto = (Mode == AUTOMATIC);
  }
```

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 >>

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:05 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.

8 Responses to "Improving the Beginner's PID: On/Off"

1. *Autopilot* says:
   July 26, 2011 at 6:15 pm

Hello,

Thanks for your tutorial but isn't it a good idea to reset integral sum before resuming automatic control after manual mode? Let's imagine controller had accumulated some output in integral term, then it's switched to manual mode for some time, then again back to automatic. In that case controller would try to use old accumulated value (based on history) under perhaps different conditions. What's your view?

2. *Brett* says:
   July 27, 2011 at 6:34 am

   Definitely agree on that. That's covered in the next section: Initialization.

3. *Adrian* says:
   August 19, 2011 at 8:35 am

   I believe there is an error in the definition of the timeChange variable, on line 17. I have found that it is possible that timeChange can become negative when switching between MANUAL and AUTOMATIC. This was stopping the subsequent if statement from return to correct value , resulting in the PID not updating.

   Changing the definition from int to unsigned long appears to have resolved this issue for me.

   Oh, and thanks a heap for both the Arduino PID library, and your excellent documentation.

   Best regards,
   Adrian

4. *Brett* says:
   August 19, 2011 at 8:50 am

   @Adreian
   yes that is indeed an error. there's a conversation on the sample time page that offers a solution.

5. *Adrian* says:
   August 19, 2011 at 9:20 am

   Thanks for the fast response confirming my hack solution might actually work. :)

   Cheers, Adrian

6. *Kyle* says:
   April 9, 2012 at 1:25 pm

   One 'complaint', line 15, if(!inAuto) return;, is bad programming. A more correct way would be while (inAuto) { … }. Also, I was trying to think of a way to fix your last change that was written the same way, if(Min > Max) return;. You could rewrite it the same way. Both changes would require you to make the test false at the end of the while loop, so it would drop out properly. Yes, it is more code, but, it is written more correctly.

7. *Kyle* says:
   April 9, 2012 at 11:32 pm

   Stupid me. I should have suggested you just change your logic instead of changing the language completely. So, rather than if(!inAuto) return; you should do if (inAuto) { … }. The same goes for the if (Min > Max) return;. It should be if (Min < Max) { … }. The preceding follows good programming practices.

8. *Anthony* says:
   April 30, 2012 at 1:32 pm

   Great article!

   I just wanted to respond to Kyle’s comments about the “bad” programming style of having multiple returns in your function. There’s nothing wrong with that. In fact, if multiple conditions decide whether to exit a function, it’s sure as hell a whole lot nicer to have a return there instead of a large amount of nested ifs. In this case, the return at line 15 makes it very clear what will happen in manual mode, so I prefer it to the if clause around the whole rest of the function (as Kyle suggests). Just my 2 cents.

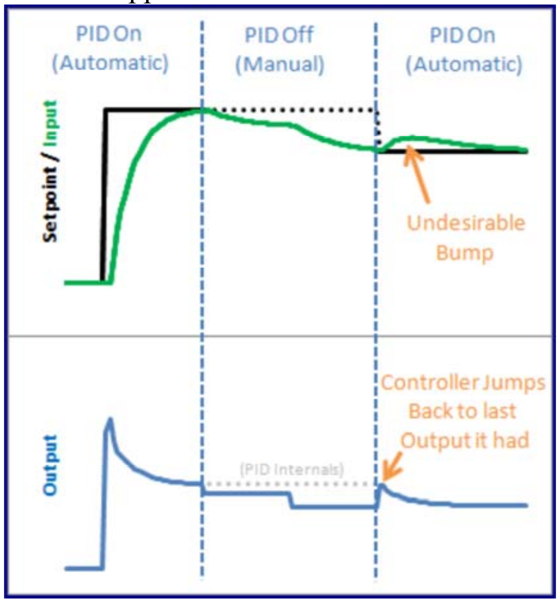
   Cheers, Anthony

Improving the Beginner's PID: Initialization

(This is Modification #6 in a larger series on writing a solid PID algorithm)

The Problem

In the last section we implemented the ability to turn the PID off and on. We turned it off, but now let's look at what happens when we turn it back on:



Yikes! The PID jumps back to the last Output value it sent, then starts adjusting from there. This results in an Input bump that we'd rather not have.

The Solution

This one is pretty easy to fix. Since we now know when we're turning on (going from Manual to Automatic,) we just have to initialize things for a smooth transition. That means massaging the 2 stored working variables (ITerm & lastInput) to keep the output from jumping.

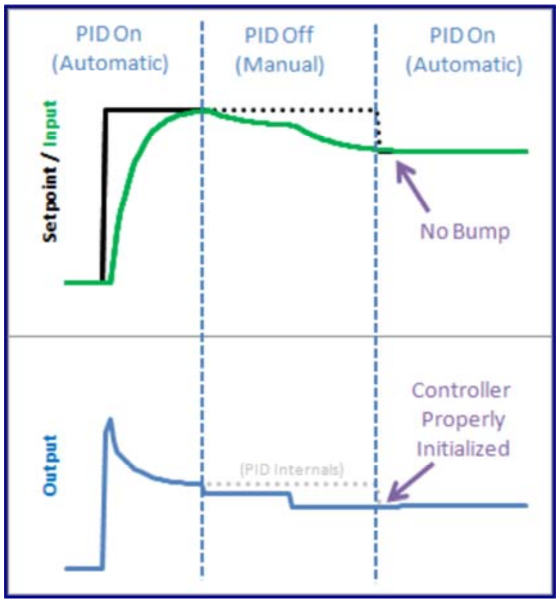
The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1

void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      /*Compute PID Output*/
      Output = kp * error + ITerm- kd * dInput;
      if(Output> outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;

      /*Remember some variables for next time*/
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}

void SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto && !inAuto)
    {  /*we just went from manual to auto*/
```

```
            Initialize();
        }
        inAuto = newAuto;
    }

    void Initialize()
    {
       lastInput = Input;
       ITerm = Output;
       if(ITerm> outMax) ITerm= outMax;
       else if(ITerm< outMin) ITerm= outMin;
    }
```

We modified SetMode(…) to detect the transition from manual to automatic, and we added our initialization function. It sets ITerm=Output to take care of the integral term, and lastInput = Input to keep the derivative from spiking. The proportional term doesn't rely on any information from the past, so it doesn't need any initialization.

The Result



We see from the above graph that proper initialization results in a bumpless transfer from manual to automatic: exactly what we were after.
Next >>

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:06 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.

3 Responses to “Improving the Beginner’s PID: Initialization”

1. *Paul* says:
   March 8, 2012 at 3:21 pm

   Why do you set the Iterm=output?

2. *wezzo* says:
   May 4, 2012 at 9:29 pm

   This is a really fantastic blog that gives a really good balance of practical problem->solution stuff and enough background to give it context.

   Really very well done!

   I’m still geeking out, and my inner nerd has ‘horny rimmed glasses’ over this. I jest ye not!

   Hoping to adapt this to Arduino PID temp controller and maybe notsowobbly wobblebot.

3. *Eugeniu* says:
   June 3, 2012 at 5:20 am

   Thanks. This series of posts helped me a lot in developing pid controllers for freescale car competition.

Improving the Beginner's PID: Direction

(This is the last modification in a larger series on writing a solid PID algorithm)

The Problem

The processes the PID will be connected to fall into two groups: direct acting and reverse acting. All the examples I've shown so far have been direct acting. That is, an increase in the output causes an increase in the input. For reverse acting processes the opposite is true. In a refrigerator for example, an increase in cooling causes the temperature to go down. To make the beginner PID work with a reverse process, the signs of kp, ki, and kd all must be negative.

This isn't a problem per se, but the user must choose the correct sign, and make sure that all the parameters have the same sign.

The Solution

To make the process a little simpler, I require that kp, ki, and kd all be >=0. If the user is connected to a reverse process, they specify that separately using the SetControllerDirection function. this ensures that the parameters all have the same sign, and hopefully makes things more intuitive.

The Code

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000; //1 sec
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1

#define DIRECT 0
#define REVERSE 1
int controllerDirection = DIRECT;

void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      /*Compute all the working error variables*/
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm > outMax) ITerm= outMax;
      else if(ITerm < outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      /*Compute PID Output*/
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;

      /*Remember some variables for next time*/
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
   if (Kp<0 || Ki<0|| Kd<0) return;

  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;

  if(controllerDirection ==REVERSE)
   {
      kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime
                      / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm > outMax) ITerm= outMax;
   else if(ITerm < outMin) ITerm= outMin;
}

void SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto == !inAuto)
    {  /*we just went from manual to auto*/
        Initialize();
    }
    inAuto = newAuto;
```

```
}

void Initialize()
{
   lastInput = Input;
   ITerm = Output;
   if(ITerm > outMax) ITerm= outMax;
   else if(ITerm < outMin) ITerm= outMin;
}

void SetControllerDirection(int Direction)
{
   controllerDirection = Direction;
}
```

PID COMPLETE

And that about wraps it up. We’ve turned “The Beginner’s PID” into the most robust controller I know how to make at this time. For those readers that were looking for a detailed explanation of the PID Library, I hope you got what you came for. For those of you writing your own PID, I hope you were able to glean a few ideas that save you some cycles down the road.

Two Final Notes:

1. If something in this series looks wrong please let me know. I may have missed something, or might just need to be clearer in my explanation. Either way I’d like to know.
2. This is just a basic PID. There are many other issues that I intentionally left out in the name of simplicity. Off the top of my head: feed forward, reset tiebacks, integer math, different pid forms, using velocity instead of position. If there’s interest in having me explore these topics please let me know.

Flattr this!

Tags: Arduino, Beginner's PID, PID

This entry was posted on Friday, April 15th, 2011 at 3:07 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.

77 Responses to “Improving the Beginner’s PID: Direction”

Newer Comments »

1. *Tom L* says:
   April 15, 2011 at 5:35 pm

   These are by far the clearest presentation of entry-level PID issues I’ve encountered: very nice job, and very helpful for newbies trying to get our own code up and making sense.

   Yes, more articles would be useful! All the topics you mentioned plus cascaded controllers, and skipping intermediate loops….

   Thanks very much for your work!

2. *Brett* says:
   April 16, 2011 at 7:56 am

   Glad it was useful to you. I'll start making a list of topics to address.

3. *Mike* says:
   April 18, 2011 at 11:07 am

   really in-depth set of program notes
   thanks for taking the time
   and explaining so clearly!

4. *Khalid Khattak* says:
   April 18, 2011 at 1:20 pm

   Thanks for such a nice writeup..i really enjoyed every word..you are a good teacher..thanks for sharing…

5. *rom* says:
   April 27, 2011 at 1:00 am

   muy buen trabajo y muy util! gracias por compartirlo con todos

6. *Brett* says:
   April 27, 2011 at 6:35 am

   ¡De nada! Que felicidad tengo sabiendo que esto he sido útil para el mundo español. Puedo hablar español, pero no lo puedo escribir a el nivel requerido por esta serie.

7. *Ricardo* says:
   May 3, 2011 at 12:09 am

   Primero, enhorabuena y muchas gracias por la biblioteca y el blog!! y segundo, me surgen un par de dudas al probar la libreria , en el ejemplo PId_basic por ejemplo , los valores que se utilizan para kp,ki,kd son 2,5,1, no?

   PID myPID(&Input, &Output, &Setpoint,2,5,1, DIRECT);

   entonces cual es el maximo y que se podria utilizar en cada uno? lo poco que vi de pid siempre utilizan hasta 1.(0,1,0,3…)

   con la biblioteca seria igual pero hasta 10? o 100?
   y otra pregunta, como input se podria utilizar una variable obtenida desde un sensor digital,habria problemas?

   como puedes ver no se mucho de esto, solo intento aprender

   muchas gracias !! y un saludo

8. *Brett* says:
   May 3, 2011 at 8:03 am

   Gracias Ricardo. Los valores de Kp, Ki, Kd no son porcentajes. Se puede usar cualquier valor de tipo "double". Normalmente encuentro que valores entre 0.01 y 10 me dan los mejores resultados, pero a veces eh usado 15 o 20. recomiendo empezando con valores pequeños.

   para usar un sensor digital tendrás que primero convertir el señal a un valor analógico. Aparate de eso, lo demás sera igual.

9. *Ricardo* says:
   May 5, 2011 at 4:38 am

   ok entendido lo de los valores, la duda de el sensor digital , es por que pretendia experimentar con el pid para un control de temperatura, la temperatura la obtendría en principio con un ds1820, entonces, me recomiendas un convertidor digital/analogico intentarlo con alguno analogico tipo lm35?

   muchas gracias por la resupuesta, un saludo

10. *Brett* says:
    May 5, 2011 at 6:59 am

    Nada tan complicado. Solo quise decir que no puedes usar 01001011011… Tendrás que usar la temperatura propia (130, 131, …)

11. *Murl* says:
    May 7, 2011 at 2:34 pm

    Thanks for all of the time and effort you've put into this library, not to mention documenting it so well. And – thanks to Control Station for allowing you to do it – it's another indication of what a stand-up company you work for.

    I don't know if it's just my browser, but the listing above seems to be corrupted in several places. For example, I think line 22 is supposed to be:
    if(timeChange>=SampleTime)
    but on this page, it shows up as:
    if(timeChange&gt;=SampleTime)

12. *Brett* says:
    May 8, 2011 at 12:16 pm

    Thanks for the heads up Murl. the plug-in I'm using to display the code can be a little finicky. I've fixed the > thing.

    As far as Control Station goes, yeah they've been great about this. I do my best, however, to limit references to my day job. I need to make sure I can swear if I want to without it reflecting poorly on my employer.

13. *Mark Barnes* says:

May 9, 2011 at 5:46 pm

Brilliant set of tutorials, I learnt so much. Thanks.

14. *Pablo* says:
June 3, 2011 at 3:59 pm

Hola,
cambiando de Manual a Auto tarda como 10 segundos en empezar a cambiar la salida. Despues funciona perfecto.
Alguna idea?
Gracias por la ayuda,
Pablo

15. *Brett* says:
June 4, 2011 at 7:40 am

Pablo, no me ocurre nada obvio. capas hay algo en el programa que se enciende a la misma vez?

16. *matthew* says:
June 30, 2011 at 5:17 pm

Thanks so much for this! Just what I needed!

17. *raul* says:
June 30, 2011 at 8:09 pm

genius!! I read the whole write up and had a great time, I am now tuning the constants.. 😀 Thanks!!!

18. *kps* says:
July 6, 2011 at 9:12 am

Excellent series of articles.
Most lucid explanation.

Long Live Brett.

19. *paul* says:
July 21, 2011 at 6:14 pm

Excellent series. Great explanation, so easy to follow with the code a graph examples. Very good idea with the code highlighting for the relevant section!

20. *Jorge* says:
July 21, 2011 at 7:05 pm

Excellent Job!!

Thank you for the clear concise explanations.

Best Regards,
Jorge

21. *noonv* says:
July 22, 2011 at 1:34 am

Great work! Thanks!

22. *Sean* says:
July 22, 2011 at 11:37 am

Just wanted to chime in with another big Thank You!

Really appreciate the time you put into sharing this…

23. *Oleksiy* says:
July 22, 2011 at 11:40 am

Thank you for the write up. I'm a big fan of control theory, and this is very useful! One other thing that comes up often is choosing how often to call the function, i.e. sampling time vs. the speed of change of the underlying system.

Delightful reading!

24. *Kees Reuzelaar* says:
July 22, 2011 at 3:25 pm

Excellent write-up! I love the way you take a 'too simple' version step by step to an actual useful version. Very easy to follow.

Oh, and YES, I'd love to see you implement (and document!) more advanced concepts such as feed-forward and whatever else you can think of.

25. *Prickle* says:
July 23, 2011 at 4:38 am

Brilliant introduction to the subject. I am also planning to implement a firmware PID and you have helped me past many pitfalls.

I think describing techniques for tuning the PID controller would complement this series. How does one find good parameters?

26. *Brett* says:
July 23, 2011 at 7:01 am

@Prickle I would suggest taking a look at the diy-pid-control google group. I personally can't help, as I have an employment agreement to contend with. (I work for a company that, among other things, does pid tuning)

27. *Lance* says:
July 23, 2011 at 1:48 pm

I just wanted to drop you a note and say this was a great tutorial!

28. *Bert* says:
July 24, 2011 at 5:03 pm

Very, very good story !
Thanks for taking the trouble to put this in the net.

Do you have some ideas or code for self-tuning of the PID ?
I was investigating this for some time now but did not find any good source for it. Please drop me a line if you do !

29. *Eduardo Lavratti* says:
July 28, 2011 at 3:02 pm

Very good Xplanation about pid i ever seen .
Thank you.

30. *Jacob* says:
July 30, 2011 at 12:39 am

I'm interested in similar short articles about feed forward and using velocity instead of position. I've heard of these concepts in other contexts, but your explanations & graphs are very clear. Thanks!

31. *James Brown* says:
August 5, 2011 at 10:19 am

The most eloquent and understandable explanations of PID I've found. Please continue to expand the articles – they are a great resource – thanks!

32. *gyro_john* says:
August 8, 2011 at 11:42 pm

Re: autotuning. I use PID temperature controllers. Their input is a thermocouple and the output is On/Off control of current to an electric heater. The control algorithm adjusts the on/off time of the output in order to hit and hold temperature.

In my experience, all modern temperature controllers have an Auto Tune function, and it works very well. I haven't had to manually tune the parameters since ever.

The auto tune algorithm works like this:
- starting from cold, or from below temperature at least,
- the output is turned on full, and stays on until the temperature reaches the set point.
- at that point the output is turned off, and there will be a big temperature overshoot.
- the output will remain off until the temperature drops to the set point.
- then the output will be turned on full again. It will catch the falling temperature (there will be an

undershoot) and stay on until the temperature climbs to the set point again.
- at that point the output will turn off again and there will be another temperature overshoot.
- This time, when the temperature drops through the set point, the Auto Tune routine terminates (after having completed two overshoot / undershoot cycles), it puts appropriate values in the P, I and D variables, and PID mode takes over.

Apparently this performance allows the algorithm to put reasonable numbers to the system's performance, and calculate appropriate values for the parameters.

The auto-tune function only needs to be run once on any given setup. The parameters calculated will continue to work fine until that controller is repurposed for some other system.

Question: Is such an auto-tune procedure easy to implement here, thus getting around the need for trial-and-error parameter adjustment?

Thanks very much for the very informative tutorial.

33. *Brett* says:
[August 9, 2011 at 7:04 am]

@John you're describing the "relay" autotune method. if you had asked me this 2 weeks ago I would have given my much repeated line that I have an employment agreement that precludes me from writing this code. I have recently, however, come across some open-source code, and have received permission from my company to port it to arduino. so.. stay tuned 😃

34. *Eric Morrison* says:
[September 6, 2011 at 1:45 pm]

You can remove the 'if' block (lines 51-56) by using DIRECT = 1 and REVERSE = -1. Now for every pass through "SetTunings" you use:

kp = Kp * controllerDirection
ki = (Ki * SampleTimeInSec) * controllerDirection
kd = (Kd * SampleTimeInSec) * controllerDirection

(lines 47-49)
Not a big code savings but every cycle counts sometimes.

35. *Brett* says:
[September 6, 2011 at 1:53 pm]

I could be wrong, but it's my understanding that multiplication is more computationally intensive than subtraction (or an if statement). as written the program is slightly bigger, but I think faster than using 3 multiplications, provided my understanding is correct.

36. *Steve Houtchen* says:
[October 3, 2011 at 11:01 pm]

Brett,

I have an PID application using an 8 bit embedded
controller that can use any speed improvement in
the algorithm you can think of..

In particular I am interested in your thoughts about using
gains that are powers of 2 for fast multiplication
, and using integer math…

SteveH

37. *Michel* says:
October 19, 2011 at 6:29 am

    @Brett: I’d be VERY interested in the “relay” autotuning port you’re working on. Can you give us an update of your progres?

    And, if possible, do you have a link to that open-source project that you found? I wouldn’t mind having a look at that code too, even if it’s not for the Arduino platform (or in a totally different programming language).

    Thanks!

38. *Brett* says:
October 19, 2011 at 11:00 am

    @Michel: I have a rough version of the code working, but it’s not polished. I used this code as the starting point, although I modified the peak identification method to better handle the noise of a real process.

39. *Michel* says:
October 19, 2011 at 2:19 pm

    Thanks Brett, very much appreciated!

40. *Cameron* says:
October 29, 2011 at 1:02 am

    This is awesome, thanks so much! Really well written article, excellent explanations.

41. *Elwood Downey* says:
November 4, 2011 at 8:54 pm

    Thank you very much: clear, excellent discussion. My vote for next topic is Kvff and Kaff. Thanks again.

42. *Jeramy* says:
November 5, 2011 at 9:51 pm

    Thanks Brett, this is really nice work. With regard to advanced topics, I would be interested in both feed forward as well as integer math implementations of PID.

43. *Elwood Downey* says:
November 7, 2011 at 8:10 pm

I think I have Kvff and Kaff working. Basically I just add them as two more terms to output as in:

output = kp * error + ITerm- kd * dInput + Kvff*Vcmd and Kaff*Acmd;

where Vcmd and Acmd are two new user parameters for the commanded velocity and acceleration.

But now my question is: suppose I want to enforce Vmax and Amax? How does one constrain the PID output to never generate output that moves beyond these limits? Thanks.

44. *Andrew G* says:
November 14, 2011 at 4:25 am

This was the best explanation of PID control I've ever read. I wrote a PID for a mechatronics project last year, and ran into most of the same problems, and solved them in similar ways (but generally not as elegantly). Definitely would have saved loads of time if I'd had this as a reference then, will be saving this and recommending it to anyone who might need it.

Looking forward to news on the auto-tuning code port. I actually used that matlab toolkit briefly working with my project last year, but ended up hand-tuning, because my code had some confusing integer scaling.

have you considered a version that uses pure integer math to decrease processor load and save space?

45. *Andrew G* says:
November 14, 2011 at 4:26 am

I forgot to say what I intended my previous post to be–thank you very much for the time and effort you put into the code and especially this fantastic explanation!

46. *Cortade* says:
November 22, 2011 at 7:35 am

Very good series! I would like to ask you: could you recommend us any books or sources you have used lately?
Thanks a lot!

47. *Brett* says:
November 22, 2011 at 8:03 am

@Andrew I'm glad this was useful to you. I have considered an integer version, and it's on my list. the main stumbling block is that I still haven't decided how I'm going to specify fraction tuning parameters. there are several ways it can be done, and I need to settle on the way think is the most intuitive to the hobbyist.

@Cortade this is largely an original work, based on several years of writing pid algorithms. I don't know of any books that cover this subject (although there has to be a least SOME, I just have never seen one.) A lot of my understanding over the years has come from reading countless manufacturer

specifications. here's one from rockwell automation:
http://samplecode.rockwellautomation.com/idc/groups/literature/documents/wp/logix-wp008_-en-p.pdf
for general PID understanding I would recommend controlguru.com, created by the guy who taught me about pid.

48. *Robin* says:
November 30, 2011 at 2:37 am

That was a brilliant series! I found your series from the Arduino PIDLibrary page and I'm glad I stumbled across it! I really liked the graphs of the behavior before and after each modification to illustrate the impact. Thanks!

49. *Cortade* says:
December 1, 2011 at 6:37 am

Thanks a lot. I didn't know this website. Strongly recommendable.

50. *Converge Fitness and Nutrition* says:
January 4, 2012 at 1:00 am

totally kick ass. I can only imagine how much effort this must have taken to simply derive all that let alone write it up for us folks to learn too. My sincerest thank yous. I eagerly await your implementation of an auto-tune loop.

1. *Stefan Weber* says:
   January 11, 2012 at 11:02 am

   Nice job ! Keep the good work up ! Have you thought of doing different controll approaches like IMC or predictive control ?

2. *Brett* says:
   January 11, 2012 at 11:31 am

   I have written an IMC controller (for fun) in the past, but I would consider that a "beginner" IMC; not something I would want to give people as an example.

3. *Aaron Newsome* says:
   January 21, 2012 at 3:07 pm

   This is really incredible. I wish I would have found it a long time ago. I'm grateful to have finally stumbled upon some code and an explanation I can wrap my head around! Great job!

4. *wwwoholic* says:
   January 29, 2012 at 8:34 pm

   Wouldn't it be MUCH simpler to invert output in one place instead of fiddling with constants all over the program?

   And on related note, every time SetSampleTime gets called you introduce an error (however small) to constant values. Why not keep values set by user and calculate "running" constants as you do in SetTunings.

   Also, it would be nice to see time overflow problem fixed on these pages. For those who do not read responses 😃

5. *Brett* says:
   January 29, 2012 at 9:50 pm

   @wwwoholic thanks for the comments.

   I'm not sure what you mean by inverting the output. Make it negative? This would work if I was using the velocity form, but just making the output -25 instead of 25 would not produce the desired effect. At the very least this would require some modifications to the initialization code. Another common way of doing the direction thing is to define error as PV-SP instead of SP-PV. Maybe that's what you meant? Also, I'm not sure what you mean by "all over the program." I make them negative in one place. other than that all the code is the same.

   Your SampleTime point is well taken. I used placeholders for the tuning parameters because I foresaw a situation (adaptive tuning) where that function would be called repeatedly. With the SampleTime, I just couldn't see a time were that would be called over and over, so I decided the error risk was minor, and not worth the extra (albeit small) overhead addition.

   It's on my list to go through the series and update both the overflow and reset windup code. With a day job and the ospid I haven't been able to find the time.

6. *wwwoholic* says:
   February 5, 2012 at 10:44 pm

   It has been long time since I’ve learned how PID works, so I am more than fuzzy on velocity or ideal or any other forms. However discrete form that you implementing quite easy to comprehend.
   Mathematically speaking negating all k constants is exactly the same as negating output
   Output = – (kp * error + ki * errSum – kd * dInput)
   Unless, of course, you meant something else when you wrote kp = (0 – kp);

   However since you did not change the output limiting (lines 33, 34) and it will not work if you negate output then I guess you assumed Input, Output and Setpoint always be positive or zero.

   In this case changing direction is usually done by:
   Output = outMax – Output
   and not by changing sign of constants.

7. *Brett* says:
   February 5, 2012 at 11:56 pm

   @wwwoholic. I get what you’re saying now. That would certainly be mathematically equivalent. I guess my style was to leave the Compute function as uncluttered as possible, and sort of compartmentalize the direction code.
   You could cerainly throw an if structure into the compute function to decide if you need to multiply by -1. Whatever you’re comfortable with.
   The ultimate goal of this series was to get people familiar with the intenals of the library to the point where they could say “well I’m going to do it THIS way.” It looks like in your case I succeeded! 😉

8. *A* says:
   April 15, 2012 at 11:21 pm

   This is a terrific explanation. Thanks a lot for putting so much time and care into this! I learned a lot.

9. *Dave* says:
   April 16, 2012 at 5:53 pm

   Brett,
   I’ve been reading your (Great job by the way!!!!) blog and I’m trying to understand how the PID with a relay works. I want to fire a relay with a digital output at a set interval until the analog input reaches the setpoint at which time it goes into standby mode waiting for the analog signal to drop at which times this process repeats itself.

   Do you have any ideas if your PID would work for this application? And if so do you have a moment to write a short code example?

   Thank you
   Dave
   dpina@jgmaclellan.com

10. *Kyle* says:
    April 23, 2012 at 2:20 am

This is a really great resource. I don’t actually program micro controllers but work with PLC and DCS systems, and this really helped give me a rough idea of what is going on inside the processor.

I found the examples and calculations done in college where useless, and we actually had vendors in when an instructor was giving lectures on PID algorithms using all calculus. The vendor was quick to inform the instructor that the methods being taught were not how controllers implement PID anymore.

Anyway this is definitely an awesome series, much appreciated.

11. *Peter Jordan* says:
April 30, 2012 at 8:17 am

Brett,

Great explanation and code. I want to use PID for pH control in a hydroponics setup. However, there are two reagents involved, both of which have their own solenoid to dispense. I’m not sure how to relate the output to these two devices. As well, they will liekly hae a different effect on the process variable for the same on time interval. Very difficult to normalize this. Your assistance would be very much appreciated in terms of understanding how to apply the code.

12. *Brett* says:
April 30, 2012 at 8:19 am

@peter this a very involved (and highly interesting) topic. I recommend re-posting on the diy-pid-control google group where it can be discussed as its own thread, rather than interspersed in these comments

13. *Alex* says:
May 4, 2012 at 9:08 am

Awesomeness redefined !!! the best tutorial that I have seen for a long time .. Lots of respect..

14. *jazz* says:
May 13, 2012 at 9:07 am

@peter, the problem with ph control is that it is exponential ,you can’t use a solenoid valve for that. You must get real control valves with the right curve of the plug and seat,In industries we use a split range for the 2 control valves, the valve for acid uses 4..12 mA (0..50%) and the valve for alkaline is using 12..20mA(50..100%) this means by pH7 both valves are closed.
google for micropak control valves
Hopefully this gives you some ideas

15. *Brett* says:
May 13, 2012 at 11:18 am

@jazz I’m not sure I agree with you on that. 4-20mA control valves are definitely more precise, but you can approximate an analog behavior by controlling how long the solenoid pulses over a given widow (similar to PWM, but slower) we've been having a great conversation on the diy-pid-control group. I'd love to hear your thoughts.

16. *Tom* says:
May 17, 2012 at 12:16 pm

Just to say, yes the advanced topics are of interest!

17. *Brain* says:
May 18, 2012 at 6:52 pm

Hey mate, first, as many others said, great work, and thanks for sharing it.
Second, i have Arduino 1 installed, and although the PID library is working just fine, i have not been able to compile this code you've posted here, as i get: undefined reference to `setup', and, undefined reference to `loop', quite annoying it's been actually, and i just don't see where the problem is, don't think is the code.

So anyway, if you happen to know the solution to my problem i'd very much appreciate it! cheers!

18. *Brett* says:
May 18, 2012 at 9:48 pm

@Brian all arduino programs must have "void loop()" and "void setup()" declared.

19. *Brain* says:
May 19, 2012 at 7:01 pm

yes! and say on the basic PID code you wrote void Compute() and void SetTunings(double Kp, double Ki, double Kd), which should be enough right?
Yet it shows me that error. I really don't know whats going on here…

20. *Mason* says:
May 29, 2012 at 3:38 pm

This is excellent! Thanks for writing this. I work in state based control now and only had ever created PID to the level you begin with. Your tutorial helped me remember the basics and taught me some new tricks as well. Much appreciated!

21. *Thomas* says:
June 13, 2012 at 9:49 am

Woohh, I'm deeply impressed!
In 1975 I programmed a true direct digital control system on a real time computer as my master thesis.
I could do individual PI PD P PID….
Some of your solutions are quite familiar to me.
This broght back a lot of memories. The Computer was a million time as big as the Ardurino bord today 😃

22. *Thomas* says:
June 13, 2012 at 11:11 am

…..and I had about 4500 lines of assembler code on punch cards. I guess not many people know how a punch card looks like today. The free memory of the computer was about 24kB so a bit smaller than the one on the Ardrino. The minimum cycle was 100ms. My challange was the time to access the 5MB disk drive during the cycle to read and store my calculated PID data. I could run up to 24 different PIDs including master slave configurations. The communication device was a Teletype…. At the end a got a "A+" rating 😃))

23. *Steven* says:
July 10, 2012 at 6:34 pm

WOW THANK YOU! This is great! Do your output limits still work if the input and output are negative?

24. *Brett* says:
July 11, 2012 at 9:30 am

Yup. Should work fine

25. *ShangTao Yan* says:
July 20, 2012 at 7:00 am

Thanks very Much!!! A nice job i have nerver saw before.

But i still don't how to use output to control a runing motor with arduino.

Could you give more detai Code ? Thanks again

26. *Seba* says:
July 23, 2012 at 3:13 pm

Thanks for the guide! Things like this makes life easier for begginers!

27. *Scott* says:
July 27, 2012 at 8:04 pm

Thanks Brett. You did a real nice job with this (PID). I took an electrical engineering class in control theory and I assure you it was far less helpful (laplace transforms and partial fraction integration until I couldn't stand it).

You have provided a relatively simple and understandable primer for PID and the code. This is very empowering. Nice work Brett.