

# Project Blog

Project updates and... um...

---

« [Improving the Beginner's PID – Introduction](#)  
[Improving the Beginner's PID – Derivative Kick](#) »

## 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

```
1  /*working variables*/
2  unsigned long lastTime;
3  double Input, Output, Setpoint;
4  double errSum, lastErr;
5  double kp, ki, kd;
6  int SampleTime = 1000; //1 sec
7  void Compute()
8  {
9      unsigned long now = millis();
10     int timeChange = (now - lastTime);
11     if(timeChange>=SampleTime)
12     {
13         /*Compute all the working error variables*/
14         double error = Setpoint - Input;
15         errSum += error;
16         double dErr = (error - lastErr);
17
18         /*Compute PID Output*/
19         Output = kp * error + ki * errSum + kd * dErr;
20
21         /*Remember some variables for next time*/
22         lastErr = error;
23         lastTime = now;
```

```

24     }
25 }
26
27 void SetTunings(double Kp, double Ki, double Kd)
28 {
29     double SampleTimeInSec = ((double)SampleTime)/1000;
30     kp = Kp;
31     ki = Ki * SampleTimeInSec;
32     kd = Kd / SampleTimeInSec;
33 }
34
35 void SetSampleTime(int NewSampleTime)
36 {
37     if (NewSampleTime > 0)
38     {
39         double ratio = (double)NewSampleTime
40                       / (double)SampleTime;
41         ki *= ratio;
42         kd /= ratio;
43         SampleTime = (unsigned long)NewSampleTime;
44     }
45 }

```

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 implementation, 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 to 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.


## 23 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. Then 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 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

22.  *DG* says:  
[October 9, 2012 at 1:03 pm](#)

Brett, how important is it that the algorithm be called on a regular schedule? I'm planning on using your algorithm in an environment (Android) that can't guarantee that it will be called with millisecond accuracy. If the SampleTime is 15 s but there is a 16 s difference between calls will this seriously affect the PID accuracy? Thanks.

23.  *Brett* says:  
[October 9, 2012 at 1:09 pm](#)

@DG the variation you describe wouldn't have a HUGE impact, and subsequent evaluations would correct mistakes made earlier on. it wouldn't be 100% repeatable however, which can sometimes be a big issue. if you think this might be a big deal, you could leave in the time divisions I remove in this step.

## Leave a Reply

 Name (required) Mail (will not be published) (required) Website

• Search for:

## • Links



o



o



o

## • This Site

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

## • Categories

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

## • Archives

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



- [July 2010](#)
- [March 2010](#)
- [November 2009](#)
- [October 2009](#)
- [September 2009](#)
- [August 2009](#)
- [July 2009](#)
- [June 2009](#)
- [May 2009](#)

---

Project Blog is proudly powered by [WordPress](#)  
[Entries \(RSS\)](#) and [Comments \(RSS\)](#).