

## Re: modified PIC control loop

**Source:** <http://sci.tech-archive.net/Archive/sci.electronics.design/2004-08/0482.html>

---

**From:** terry (*the\_domes\_at\_xtra.co.nz*)

**Date:** 08/03/04

Date: Tue, 3 Aug 2004 14:42:53 +1200

"Terry Given" <the\_domes@xtra.co.nz> wrote in message  
news:j2gMc.4065\$N77.263011@news.xtra.co.nz...  
> "Tim Wescott" <tim@wescottnospamdesign.com> wrote in message  
> news:10g2hsjrf0ovl3e@corp.supernews.com...  
> > Rene Tschaggelar wrote:  
> >  
> > > *The PID being the simplest control loop in case an accurate model is  
> > > somehow out of reach knows many enhancements to speed it up or  
> > > otherwise  
> > > improve it. We face the problem that the control output is limited in  
> > > reality. When the limit is reached, the integrator keeps integrating  
> > > and the delay until it comes down again is felt as dead time. A  
> > > customer  
> > > suggested to freeze the integrator when the output hits the rail.*  
While  
> > > *I investigated the subject ages ago, but apparently lost some  
knowledge  
> > > due to non-use.  
> > > What is the drawback when I freeze the integrator ?  
> > >  
> > > Rene  
> >  
> > *The problem is called "windup". I've never heard of freezing the  
> > integrator when the output hits the rail -- that's an interesting one.*  
> >  
> > *The usual anti-windup scheme is to limit the integrator state. A good  
> > first choice is to full-scale, but you may want to limit it to less than  
> > full scale if you know that's all you'll need, i.e.:*  
> >  
> > *p\_part = p\_gain \* in\_error;  
> > i\_state += i\_gain \* in\_error;  
> > if (i\_state > i\_lim) i\_state = i\_lim;  
> > else if (i\_state < -i\_lim) i\_state = -i\_lim;  
> > return p\_part + i\_state;*  
> >  
> > *The anti-windup that I prefer is to hold the integrator state to that  
> > necessary to just achieve full output (I can't think of a clearer way of**

```
> > putting that). So you do something like:
> >
> > p_part = p_gain * in_error;
> > i_state += i_gain * in_error;
> >
> > if (i_state + p_part > max_out) i_state = max_out - p_part;
> > else if (i_state + p_part < -max_out) i_state = -max_out - p_part;
> >
> > return p_part + i_state;
> >
> > The nice thing about this method is that the integrator is _never_ wound
> > up beyond the point where changes in the integrator state are not
> > immediately effective. In a well tuned loop it eliminates not only the
> > instability from integrator windup but also the "windup overshoot" that
> > you often see as well.
> >
> > Tim Wescott
>
> nice. And a hell of a lot simpler than most anti-windup schemes I have
> seen/played with which usually involve some sort of controller that varies
> Ki (or +/- Ilim) as a function of the commanded-versus-actual output
signal
> error. Then you have to tune the dynamics of the anti-windup loop....
>
> If you use saturating arithmetic (eg a DSP), then you could could
calculate
> an offset to add to the integrator output to force saturation, then
subtract
> it again.
>
> Likewise it can be a good idea to limit the magnitude of in_error.
>
> Another good approach is to low-pass filter the reference signal – with
the
> same Fcross as the closed control loop. That way when you command a step
> change (which NO controller can EVER follow) you avoid exciting the loop
> unnecessarily. ramp-rate limiting of the reference is another way of
> achieving the same goal – prevent asking the controller for things it cant
> deliver.
>
> I like Internal Model Control.
>
>
> cheers
> Terry
```

Another problem that frequently arises is overshoot due to large-signal behaviour – a large (step) change in reference invariably causes overshoot due to the integrator. A very simple technique can greatly improve the behaviour of a PID loop to such large signals, viz.:

conventional PID controller:

$$\text{controller output } U = (\text{setpoint} - \text{plant output}) * [K_p + K_i/s + K_d*s]$$

can be re-written as:

$$U(s) = A(s)*\text{setpoint}(s) - A(s)*\text{plant\_output}(s)$$

$$A(s) = K_p + K_i/s + K_d*s$$

weighted PID controller:

$$U = (b*\text{setpoint} - \text{plant output})*K_p + (\text{setpoint} - \text{plant output})K_i/s + (c*\text{setpoint} - \text{plant output})*K_d*s$$

can be rewritten as

$$U(s) = B(s)*\text{setpoint}(s) - A(s)*\text{plant\_output}(s)$$

$$A(s) = K_p + K_i/s + K_d*s$$

$$B(s) = b*K_p + K_i/s + c*K_d*s$$

simply setting  $b = c = 0$  can make a dramatic improvement to the large-signal response. other weights can of course be used. Astrom covers this in detail...

cheers

Terry