Demystifying PID Control

The proportional-integral-derivative (PID) control algorithm is a powerful resource for embedded systems programmers. PID control is useful in systems that sample an output, compare the sample to a desired result, and take corrective action to force the controlled element closer to the desired result. In a well-designed PID control loop, the controlled element reaches the desired value rapidly and can be made to exhibit a great deal of stability.

The words proportional, integral, and derivative relate to the correction calculated by the algorithm based on the error, the integrated error over time, and the error's rate of change. Coeffi-dents applied to each of these terms tune the system's response.

The paradigmatic form of the PID equation is:

Output =
Kp * error + Ki * sum(error) + Kd *
d(error)/dt

The K coefficients are constant or variable values determining the weight assigned by the programmer to each term.

A DRIVE THROUGH THE LOOP

The different parts of this mathematical model can be correlated to actions taken while operating a car. First, let's look at the proportional term. Let's assume that you are driving along a circular road. If you observe an error--in this case your automobile is deviating from its lane of travel--you will correct your course by placing the steering wheel in a new (constant) position and holding it there.

Next, we'll look at the integral term. Think of driving down a straight highway. As you attempt to stay in the center of your lane, perturbations (wind, potholes, road crown, and so on) affect your relative position. Rather than changing your steering wheel's position, you allow the drift (error) to accumulate (integrate) until it is noticeable, then factor a small correction into the steering-wheel position. As the car begins to drift in the opposite direction, the same process is repeated.

Finally, we'll look at the derivative term. Pretend you're 19 again. As you slide your 1965 Mustang out of a corner, you know from past experience that if you wait until the end of the curve to center the steering wheel, it will be too late. So, as the rate of turn begins to decrease, you turn the steering wheel away from the direction of turn before the turn actually ends to anticipate the car's path as it leaves the turn.

Effective PID control reaches the desired result as quickly as possible without oscillation. In our Mustang metaphor, oscillation is the same as fishtailing.

FROM THEORY TO CODE

While the model of the PID algorithm is simply stated, the implementation typically requires a great deal of insight and hand tuning. We will examine each term in detail to better understand its physical contributions and characteristics.

The proportional term produces an output directly proportional to the error. The corollary is that to have any output from this term, you must have an error. The proportional term has no time dependence and will produce an output immediately. This term should be emphasized in systems that require fast response.

The integral term is really the heart of the PID control loop. As the error decreases to zero, the product of the integral term and its coefficient produce the result needed to regulate the output to a desired value. The integral term also contains the summation of the errors.

For example, we have a proportional heater that will be used in a hot tub. If we set the heater to 100, we want the water to be promptly warmed to 100 ***. The direct approach would have us output a command of 100 to the heating element. The system will be designed so that this command will result in the heating element exerting 50% of its maximum wattage, enough to maintain the water temperature at 100 *** if the water is already at that temperature.

With this type of control, the water temperature will rise to 100*** --but it may take days. Obviously, the heating element will have to be controlled in a more sophisticated manner, allowing the wattage to rise so the desired temperature will be reached within a reasonable amount of time.

REFINING THE MODEL

Our next attempt, somewhat less naive, is to calculate a varying output to the heating element based on a history of samples and their cumulative error. This output is equivalent to a PID equation lacking a derivative term. Let's illustrate this approach by calculating our output value:

```
Output =
  .5 * error + .5 * sum(error) + 0 *
  d(error)/dt
```

No derivative term contribution exists.

Now we turn on the power. **Table 1** shows the temperature readings for our hot tub with respect to time. By the time we reach Sample 6, we have reached our desired value and begun to overshoot due to the large value contained in the integral term. However, as soon as negative errors begin to decrease the integral term, our loop gets closer to the desired value. Some time after Sample 13, the integral term will be 200 and the error contribution will be zero. The integral will sit forever at 200 as long as nothing perturbs the loop.

Small deviations will produce an error that will be corrected by the proportional term. If anything permanent happens, such as the heater losing heat-transfer ability (so that 100 does not tend to yield 100^{-} in operation), the integral term will effectively servo to whatever value it takes to produce 100^{-} of water temperature and the system will tend to stabilize at that new output value. The integral term stabilizes the output so that no errors exist. If low final error is the goal, this term should be emphasized.

The derivative term is our ace-in-the-hole for fast servo action. As the weighting of a proportional term is increased, instability results. We can cancel some of the susceptibility to instability by dampening our system with the derivative term. This term reduces the driving force if the error is decreasing. As long as the controlled element is making progress toward the desired value, the derivative term restrains the output. If the error remains constant, the derivative term has no effect.

The problem in the previous example is that anyone in the hot tub would be parboiled by the time the water temperature stabilized at 100^{-}, due to the detour through 130^{-} of water temperature. Let's reexamine our example but this time calculate the d(error)/dt term at each step to see what damping effect the derivative term would have had if it had been present.

In **Table 2**, we use the same values but do not calculate the output because the modified output would modify the next sample. We wish only to illustrate what the derivative term would have been at each step, given the same sample set as in the previous example.

As you can see by looking at the new readings, the d(error)/dt term works against the error and sum(error) terms, as evidenced by the opposite sign. The error is decreasing as the system progresses towards the desired value (Samples 1 through 6). When the temperature passes through the target point value, the proportional term and the derivative term act together to bring the value towards zero error (Samples 7 and 8). However, at Sample 9, the error is decreasing again, and the derivative term is holding back the rate of closure.

GETTING DOWN TO CODE

Properly implemented PID algorithm allows programmers to tailor the response of their systems from underdamped to over-damped by modifying the three critical coefficients. **Listing 1** is a partial PID implementation. The complete source code can be found on the

Real-Time Control & Forth Board, at (303) 2780364, the Embedded Systems Programming library on CompuServe (library 12 of CLMFORUM), or the Embedded Systems Programming bulletin board system at (415) 267-7674.

This code was written as an illustration of the PID principal, not as optimal control code. Consequently, it is not particularly Forthy--only a few arguments are passed on the stack, and variables are used to hold intermediate values. The system-specific words were written to run on a Vesta Technology single-board computer based on the Intel 80188 microprocesser, employing its onboard Forth-83 + development environment.

The output of our model control system is an 8-bit D/A converter. The controlled element consists of an 8-bit A/D converter connected through two phase-lag networks to the output of the D/A converter, as shown in **Figure 1**. The phase-lag networks simulate a real-world mechanical system that cannot respond instantly to changes in its input. In our hot tub example, if you turn the heater on, it is some time before the output is affected. A voltmeter, oscilloscope, or chart recorder connected to the input of the A/D will display the effect of the PID control algorithm on the control element as the software attempts to stabilize the servo-feedback loop.

The code example begins by defining the variables and initializing the variables to reasonable values. Kp, Ki, and Kd are the three PID coefficients. The variable COMMAND is the desired value that the loop should attain. The variable ACTUAL is the result of reading the A/D converter. ERROR is the difference between what the output is and what it should be.

The variable INTEGRAL contains a sum of the errors. Positive errors will increase INTEGRAL, negative errors will reduce it. A common problem with this term is integrator windup. If large errors persist while the loop is stabilizing, the integral term will wrap up to a high value, which will necessitate an equally large number of corresponding negative errors to wind down the integrator. This problem can be avoided by limiting the integral's attainable value. The maximum absolute value that the integral term can attain is stored in INTEGRAL_LIMIT.

LOOP_TIME controls the frequency the PID code will execute. The system has a clock incrementing at 1/18 of a second. ATOD, DTOA, and CLOCK are system-specific words that do just what you would expect.

The proportional term is calculated by the word CALC_PROPORTIONAL_TERM. After multiplying the value of Ki by the error, the result is divided by 100 to allow more resolution before the effects of integer math are felt.

The integral term is calculated by CALC_INTEGRAL_TERM. Before calculating, INTEGRATE adds the contents of ERROR to INTEGRAL and calls upon LIMITER to check that the product of Ki times INTEGRAL will not be more than INTEGRAL_LIMIT.

CALC_DERIVATIVE_TERM calculates the derivative term. (Who says Forth is hard to read?) The error is compared to the error from the last time the loop was executed. The result, DELTA_ERROR, is the change in the error. This result is divided by the time since the last error to normalize the error's rate of change.

The three PID terms are summed by SUMMATION, which outputs the result to the D/A converter. A REPORT is generated to the screen showing the key variables as they change. The final word, PID, loops with each loop sending a correction to the D/A converter attempting to cause the input of the A/D to track to the desired value.

PID is written as a Forth background task so that new values of COMMAND can be written by the user as the task is executing. This interactive-development approach has obvious benefits, but PID controls can be implemented in practically any language--even Ada. (Personal experience suggests that Forth is unparalleled as a rapid development tool.)

Whatever your implementation language, you'll find the range of PID applications is virtually limitless. The PID algorithm will work in all facets of process control--from remote-operated vehicle steering to temperature, pressure, flow, and position control.

BY STEVEN E. SARNS AND JACK WOEHR

Jack Woehr is a programmer at Vesta Technology Inc. in Wheat Ridge, Colo. He can be reached on Usenet at jax@well.uucp or as VESTA on GEnie. Steven Sarns is the president of Vesta Technology. His is a member of Mensa, Intertel, and the Michigan Society of Professional Engineers. Sarns is also a founding member of the Denver Forth Interest Group.

Listing 1

PID.F by Steven E. Sarns and Jack Woehr.

```
( Complete source code is available from the Real-Time bulletin board system, at (303)
278-0364, the Embedded )

( Systems Programming library on CompuServe (library 12 of CLMFORUM), or the
Embedded Systems Programming )

( bulletin board system at (415) 267-7674. Variables include the KP, KI, KD coefficients
plus a number of )

( intermediate calculation values and miscellaneous machine-specific addresses. )

...
```

```
( CLOCK is where the program spends all of its time. ATOD, which fetches a reading and
normalizes, is executed )

( for lack of anything better to do during this time. )

: CLOCK ( --- )

  LOOP_TIME @ 0 do ticks @

          begin ATOD pause dup ticks @ <> until
        drop loop;

...

( In calculations, the intermediate results are divided by 100 to allow "fractional" values
LIMITER prevents )

( integrator "windup" during open loop periods )

: LIMITER ( --- )

  INTEGRAL_LIMIT @ 100 KI @ */round

  INTEGRAL @ min INTEGRAL !

  INTEGRAL_LIMIT @ 100 KI @ */round negate

  INTEGRAL @ max

  INTEGRAL ! ;

: INTEGRATE ( --- )

  INTEGRAL @ ERROR @ + INTEGRAL ! ;

: CALC_INTEGRAL_TERM ( --- )

  INTEGRATE LIMITER ( Check the case if)

  (INTEGRAL is negative)

  INTEGRAL @ KI @ 100 */round INTEGRAL_TERM ! ;

( INTEGRAL_TERM is Ki * sum(error) ( since time began)
```

```
( Derivative Term: DELTA_ERROR is the change since the last reading )

: CALC_DELTA_ERROR ( --- )

  ERROR @ LAST_ERROR @ - DELTA_ERROR ! ( this error - last error)

  ERROR @ LAST_ERROR ! ;

( DERIVATIVE_TERM is Kd * d[error]/d[time])

: CALC_DERIVATIVE_TERM ( --- )

  CALC_DELTA_ERROR DELTA_ERROR @ LOOP_TIME @ 18 */round

  KD @ 100 */round DERIVATIVE_TERM ! ;

: TERMS ( --- ) cr KP @ . KI @ . KD @ . ;

  TERMS! ( n1 n2 n3 --- ) KD ! KI ! KP ! ;

: SUMMATION ( --- ) CALC_ERROR

  CALC_PROPORTIONAL_TERM CALC_INTEGRAL_TERM
CALC_DERIVATIVE_TERM PROPORTIONAL_TERM @

  INTEGRAL_TERM @ DERIVATIVE_TERM @ + + 128 + 255 min 0 max dup DTOA
OUTPUT ! ;

: REPORT ( --- ) cr

  COMMAND @ 6 .r                              ACTUAL @ 6 .r

  PROPORTIONAL_TERM @ 6 . r                        INTEGRAL_TERM @ 6 .r

  DERIVATIVE_TERM @ 6 .r                          OUTPUT @ 128 - 6 .r ;

BACKGROUND: PID ( --- )

  BEGIN CLOCK SUMMATION REPORT AGAIN ;

: GO ( --- ) 0 CMD PID WAKE MULTI ;

( As with any Forth program, the highest level words are at the bottom of the listing. In
this case, we're )
```

(executing a background task (PID) and then using the foreground task as an interactive console for)

(tweaking the coefficients. For example, you might define new coefficients (with a phrase such as 20 80 0)

(TERMS! or even define test words such as:)

(CRIT_DAMP (---) 40 30 50 TERMS! ;)