

Adding feedback can reduce the errors normally found in an open-loop control system.

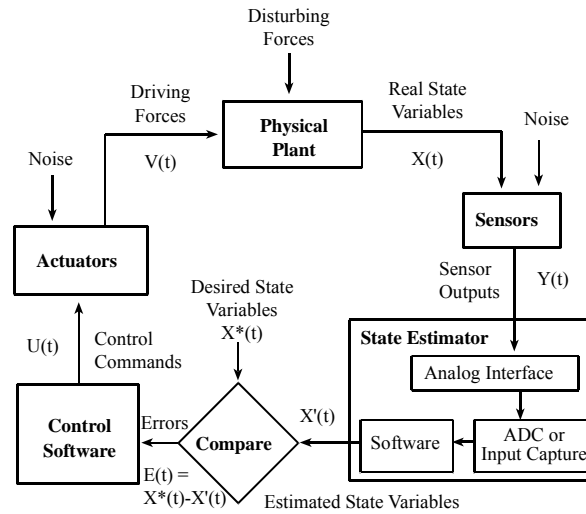


Figure 13.1. Block diagram of a microcomputer-based closed-loop control system.

13.4. PID Controllers

13.4.1. General Approach to a PID Controller

There are three components of a proportional integral derivative PID controller.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}$$

Let $\mathbf{X}(s)$ be the Laplace transform of the state variable $\mathbf{x}(t)$.

Let $\mathbf{X}^*(s)$ be the Laplace transform of the desired state variable $\mathbf{x}^*(t)$.

Let $\mathbf{E}(s)$ be the Laplace transform of the error

$$\mathbf{E}(s) = \mathbf{X}^*(s) - \mathbf{X}(s)$$

Let $\mathbf{G}(s)$ be the transfer equation of the PID linear controller.

$$\mathbf{G}(s) = c(\mathbf{k}_p + \mathbf{k}_d s + \mathbf{k}_i/s)$$

Let $\mathbf{H}(s)$ be the transfer equation of the physical plant.

\mathbf{m} is the DC gain and τ is its time constant.

The transfer function of the motor is

$$\mathbf{H}(s) = \frac{\mathbf{m}}{1 + \tau \cdot s}$$

The overall gain of the control system is

$$\frac{\mathbf{X}(s)}{\mathbf{X}^*(s)} = \frac{\mathbf{G}(s)\mathbf{H}(s)}{1+\mathbf{G}(s)\mathbf{H}(s)}$$

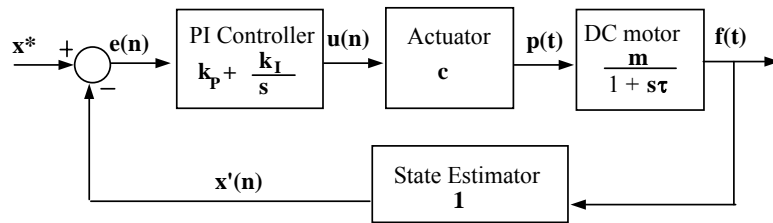


Figure 13.12. Block diagram of a linear control system in the frequency domain.

Theoretically we can choose controller constants, k_p , k_i and k_d , to create the desired controller response. Unfortunately it can be difficult to estimate c , m and τ . If a load is applied to the motor, then m and τ will change.

I.e., let

$$u(t) = p(t) + i(t) + d(t)$$

Using a proportional term creates a control system that applies more energy to the plant when the error is large.

$$p(t) = K_p e(t)$$

we simply convert the above equation into discrete time.

$$p(n) = K_p e(n)$$

where the index “n” refers to the discrete time input of $e(n)$ and output of $p(n)$.

Observation: In order to develop digital signal processing equations, it is imperative that the control system be executed on a regular and periodic rate.

Common error: If the sampling rate varies, then controller errors will occur.

The integral term makes the actuator output related to the integral of the error. Using an integral term often will improve the steady state error of the control system. If a small error accumulates for a long time, this term can get large. Some control systems put upper and lower bounds on this term, called anti-reset-windup, to prevent it from dominating the other terms.

$$i(t) = K_i \int_0^t e(\tau) d\tau$$

The implementation of the integral term requires the use of a discrete integral or sum.

$$i(n) = K_i \sum_1^n e(n) \Delta t = i(n-1) + K_i e(n) \Delta t$$

where Δt is the sampling rate of $E(n)$.

The derivative term makes the actuator output related to the derivative of the error. This term is usually combined with either the proportional and/or integral term to improve the

transient of the control system. The proper value of K_D will provide for a quick response to changes in either the set point or loads on the physical plant. An incorrect value may create an overdamped (very slow response) or an underdamped (unstable oscillations) response.

$$d(t) = K_d \frac{de}{dt}$$

There are a couple of ways to implement the discrete time derivative. The simple approach is

$$d(n) = K_d \frac{e(n) - e(n-1)}{\Delta t}$$

In practice, this first order equation is quite susceptible to noise. The following figure shows a sequence of $E(n)$ with some added noise. Notice that huge errors occur when the above equation is used to calculate derivative.

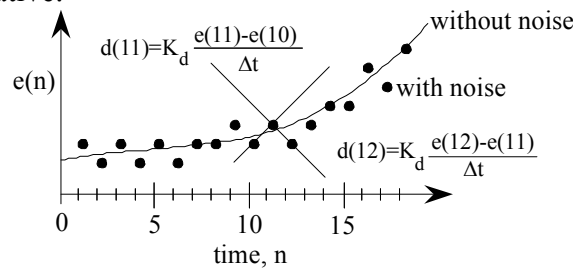


Figure 13.14. Illustration of the effect noise plays on the calculation of discrete derivative.

In most practical control systems, the derivative is calculated using the average of two derivatives calculated across different time spans. For example

$$d(n) = K_d \left\{ \frac{1}{2} \frac{e(n) - e(n-3)}{3\Delta t} + \frac{1}{2} \frac{e(n-1) - e(n-2)}{\Delta t} \right\}$$

that simplifies to

$$d(n) = K_d \frac{e(n) + 3e(n-1) - 3e(n-2) - e(n-3)}{6\Delta t}$$

Motor Speed PID Controller

The objective of this example is to develop the fixed point equations that implement the a PID velocity controller. The error $e(t)$ is $x^* - x(t)$

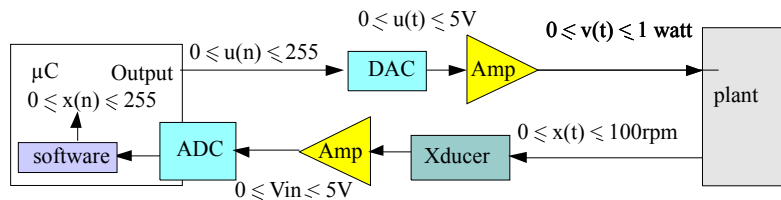
$$v(t) = K_p(x^* - x(t)) + K_i \int_0^t (x^* - x(\tau)) d\tau + K_d \frac{d(x^* - x(t))}{dt}$$

where $v(t)$ is in watts, $x(t)$ is in rpm, x^* is in rpm, t is in seconds.

$$K_p = 0.06123 \text{ watt/rpm.}$$

$$K_i = 234.5 \text{ watt/rpm-sec.}$$

$$K_d = -0.0000005 \text{ watt-sec/rpm.}$$



Simulated Motor Interface of a PID velocity controller.

The first step is to convert the inputs and outputs into voltage. Let V^* be the desired speed in volts. In particular

$$x(t) = 20 \cdot V_{in}$$

$$v(t) = u(t)/5$$

$$\frac{1}{5}u(t) = 20 K_p \left(V^* - V_{in}(t) \right) + 20 K_i \int_0^t \left(V^* - V_{in}(\tau) \right) d\tau + 20 K_d \frac{d \left(V^* - V_{in}(t) \right)}{dt}$$

where $u(t)$ V_{in} V^* are all in volts, t is in seconds. Next convert the equation to digital form

$$p(n) = 100 K_p e(n)$$

$$i(n) = i(n-1) + 100 K_i e(n) \Delta t$$

$$d(n) = 100 K_d \frac{e(n) - e(n-1)}{\Delta t}$$

Assume Δt is $100 \mu s$.

$$p(n) = 6.123 e(n)$$

$$i(n) = i(n-1) + 2.345 e(n)$$

$$d(n) = -0.5(e(n) - e(n-1))$$

```
short Xstar; // desired speed in 0 to 255 units
short Xprime; // estimated speed in 0 to 255 units
short Xlast; // estimated speed last time
short E; // error, E=X*-X'
short U; // actuator control, 0 to 255
short UP; // proportional term
short UI; // integral term
short UD; // derivative term
/* -----called every 100us----- */
interrupt 14 void TC6hndlr(void){
    TC6 = TC6+800; // next interrupt in 800 cycles
    TFLG1 = C6; // acknowledge C6F
    ATDCTL5 = 0x82; // 4 conversions on channel 2
    while((ATDSTAT&0x0001)==0); // wait for CCF0
    Xprime = ATDR0H; // estimated speed
    E = Xstar - Xprime; // E=X*-X'
    // UP = (Kp*E)/1000;
asm ldd E
asm ldy #6123 // Kp=6.123
asm emuls
```

```

asm  ldx  #1000
asm  edivs
asm  sty  UP
//  UI = UI+(Ki*E)/1000;
asm  ldd  E
asm  ldy  #2345    // Ki=2.345
asm  emuls
asm  ldx  #1000
asm  edivs
asm  tfr  y,d
asm  addd UI
asm  std  UI
    if(UI>100) UI = 100;          // anti-reset windup
    else if(UI<-100) UI = -100;
    UD = (-5*(Xprime-Xlast))/10; //Kd=-0.5
    Xlast = Xprime;
    U = UP+UI+UD;
    if(U>255) U = 255;          // overflow
    else if(U<0) U = 0;        // underflow
    PTT = U;                    // set D/A actuator
}

```

Observation: PID control will work extremely well (fast, accurate and stable) if the physical plant can be described with a set of linear differential equations.

Empirical method to determine PID controller parameters

Start with just a proportional term (K_p).

proportional controller will generate a smooth motor speed

choose the sign of the term K_p so the system is stable.

try different K_p constants until the response times are fast enough

The next step is to add some integral term (K_i)

a little at a time

to improve the steady state controller accuracy

without adversely affecting the response time.

choose the sign of the term K_i so the system is stable.

Don't change both K_p and K_i at once.

The last step is the derivative term (K_d)

a little at a time

reduce the overshoots/undershoots in the step response

choose the sign of the term K_d so the overshoots/undershoots are reduced.

An open-loop method, the **process reaction curve approach**,

first proposed by Ziegler/Nichols and Cohen/Coon in 1953

1) Single experimental test is made with the controller in open loop mode

small step change, ΔU , in the controller output is introduced
measured process response is recorded,

2) Obtain parameters of the process,

a tangent is drawn at its point of maximum slope (at the inflection point).

This slope is R , which is called the **process reaction rate**.

intersection of tangent line with original base line gives L , the process lag.

L is really a measure of equivalent dead time for the process.

Figure 13.14. A process reaction curve used to determine controller settings.

3) Design controller (Ziegler and Nichol)

ΔT is the time step for the digital controller.

run P and PI controllers with $\Delta T = 0.1L$,

run a PID controller $\Delta T = 0.05L$.

Proportional Controller

$$K_P = \Delta U / (L * R)$$

Proportional-Integral Controller

$$K_P = 0.9 \Delta U / (L * R)$$

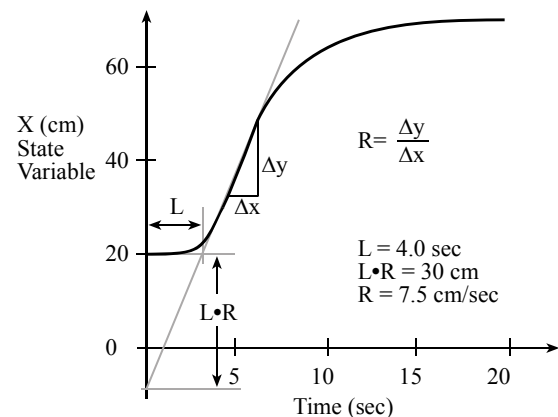
$$K_I = K_P / (3.33L)$$

Proportional-Integral-Derivative Controller

$$K_P = 1.2 \Delta U / (L * R)$$

$$K_I = 0.5 K_P / L$$

$$K_D = 0.5 K_P L$$



Controller performance

The **response time** is the delay after X^* is changed for the system to reach a new constant state.

Steady state controller accuracy is defined as the average difference between X^* and X .

Overshoot is defined as the maximum positive error that occurs when X^* is increased.

Similarly, **undershoot** is defined as the maximum negative error that occurs when X^* is decreased.

Example PI Controller Design

Tachometer

speed is measured using 16-bit input capture

speed measurement resolution of 0.01 RPM.

repeatedly updates a global variable, called **Speed**.

units of 0.01 RPM

range of 0 to 16000 (meaning 0.00 to 160.00).

Desired, which has the same units as **Speed**

Actuator

8-bit pulse-width modulation to control power to the motor.

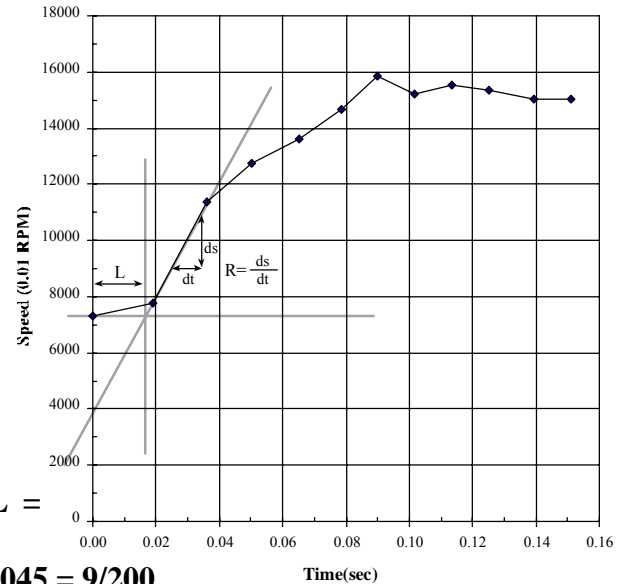
Duty, which ranges from 0 (0%) to 250 (100%).

Process reaction curve

Duty is changed at time=0 from 100 to 250

Actual data from a DC geared motor (no load)

Figure 10.1. Example process reaction curve.



From the problem description

$$\Delta U = 150 \text{ (duty-units)}$$

From the graph (rough approximation)

$$L = 0.02 \text{ second}$$

$$R = 150000 \text{ (0.01 RPM/sec)}$$

It is recommended to run PI controllers with $\Delta t = 0.1L = 0.002\text{sec}$.

$$K_P = 0.9 \Delta U / (L * R) = 0.9 * 150 / (0.02 * 150000) = 0.045 = 9/200$$

$$K_I = K_P / (3.33L) = 0.045 / (3.33 * 0.02) = 0.6757,$$

$$K_I * \Delta t = 0.6757 * 0.002 = 0.001351 = 1/740$$

$$E(n) = \text{Desired} - \text{Period}$$

$$\begin{aligned} P(n) &= K_P * E(n) \\ &= 0.045 * E(n) \\ &= (9 * E(n)) / 200 \end{aligned}$$

$$\begin{aligned} I(n) &= I(n-1) + K_I * E(n) * \Delta t \\ &= I(n-1) + 0.001351 * E(n) \\ &= I(n-1) + E(n) / 740 \end{aligned}$$

$$\text{Duty} = U(n) = P(n) + I(n)$$

Putting together, and simplifying, this code is executed every 0.002 sec. All variables are 16-bit signed integers.

```

E = Desired - Speed;
P = (9 * E) / 200;
I = I + E / 740;
if (I > 250) I = 250;           // antireset windup
if (I < 0) I = 0;
Duty = P + I;
if (Duty > 250) Duty = 250;    // range is 0 to 250
if (Duty < 0) Duty = 0;
    
```

Notice that if the motor is spinning too slowly, the error will be positive, making **P** positive and increasing the magnitude of **Duty**. Conversely, if the motor is spinning too quickly, the error will be negative, making **P** negative and decreasing the magnitude of **Duty**.