Chapter 11

Inside the PID Loop

Features

Working Inside the PID Loop
PID Structure
PID Element Definitions
A PID Analogy
What the PID Elements Do
A PID Tuning Procedure

Working Inside the PID Loop

In the world of servo motion, the PID loop has become a very popular method of gain control. However, tuning the PID loop still proves to be a formidable task for nearly everyone. Some common questions that always arise are:

- What do the PID elements do?
- Is there an adjustment order?
  - If there is, which element do you adjust first?
- How interactive are the elements?
- Is it necessary to change the gain (Kx) values for varying loads?
- How can I insure optimum response?

These questions are the most often asked because arriving at concise definitions of PID loop operation and tuning is often difficult. If a procedure cannot be explained clearly, then in most cases, the end item that it affects usually suffers. System tuning has possibly suffered from this inadequacy. It has always been something of a *seat-of-the-pants* process requiring a high degree of experience. This has led to complex and ambiguous tuning procedures, and as a result, from a system standpoint, solutions have been just good enough at best.

Therefore, the objective of this chapter is to develop a clearer understanding of how the PID loop works. With an understanding of PID element action and interaction, you should be able to better comprehend how the PID will affect your *real-time* system. In the process, you will enjoy a dramatic improvement in your tuning ability.

The PID Structure

Before we go further, it is important to understand certain terms that will be used from here throughout this book.

K – A generally accepted variable used to denote gain, an arbitrary multiplier, or a constant. Here, it will be used as a multiplier. The specific gain term will be shown by a lower case companion such as **Kp** for Proportional gain.

Update – An arbitrary but evenly spaced period of time, which controls when the motion registers, will be updated with position and/or velocity information. The DAC error corrections are also calculated in the update along with other monitoring requirements. Update time periods are chosen by either the hardware designer or the software writers (in the case of processor-based motion control devices). Some factors governing update intervals are: DAC conversion time, speed, and resolution of the system and processor.

Trajectory – The control effort applied to the motor in order to cause it to move according to a certain profile. This can be either a *position move* or a *velocity move.*

- **Velocity move** – No final stopping position will be given to the motion controller. Thus, when a START command is issued, the motor will rotate (indefinitely) until it is commanded to stop. When STOP is commanded, the motor will follow a calculated deceleration slope until it comes to a halt.

- **Position move** – A move position will be loaded into the motion registers. Calculations will determine when to begin the deceleration slope in order to ensure the stopping point is at the required position.

Error – Calculated trajectory position minus actual position. When the servo is commanded to move, the trajectory generator will follow the guidelines in Chapter 10.

It is important to understand that the current position register is used to keep track of the actual system position, and to provide the math registers with a means of calculating the error during PID loop updates.

DAC – A DAC[1] is the interface between the computer's digital output and the actual motor movement. DACs are available in three popular sizes: 8, 12, and 16-bit. This means that the DAC can partition the total peak-to-peak output signal swing into 256, 4096, or 65536 steps, respectively. Simply stated, the higher the DAC bit count, the finer the DAC resolution.

The motor will require a certain friction breakaway current to do a move. Assume a system that exhibits a five inch-pound friction torque-loading. A motor that can generate 20 inch-pounds per Amp fed from a motor drive amplifier with an input requirement of ±10 VDC and a gain of 200 Amps/Volt, will draw:

$$(5.0\ in\text{-}lb) \div \left(20\ \frac{in\text{-}lb}{amp}\right) = 0.25\ amp$$

to initiate the move and require:

$$(10.0)\left((0.25\ amp) \div \left(200\ \frac{amp}{volt}\right)\right) = 0.0125\ VDC$$

into the motor drive amplifier to deliver the 0.25A.

An 8-bit, ±10 VDC DAC will produce 0.079 volts-per-step, and a 12-bit DAC, 0.00488 volts-per-step. The 8-bit DAC will produce more than enough motor current to break friction (in the above example) with a single DAC step change, while the 12-bit DAC will require just under 3 DAC steps, and

[1] Digital to Analog Converter

the 16-bit DAC, 40.96 steps. If the system requires fine velocity resolution, the 16-bit DAC will undergo less severe modulation to generate the motion, yielding more stable performance.

Element definition

The classical PID structure consists of four independent elements:

Output = SetPoint + Proportional + Integral + Derivative

- **Output**– This is the count result actually applied to the DAC input. The count will be converted into either a voltage or current to be used by the device drive amplifier (generally ±10VDC).

- **SetPoint** – This is a fixed count value that is usually used with analog style control such as valves. For servo motor control, it could be used to offset the effects of gravity in a vertical type system.

SetPoint = Constant value

If used, its count value is directly applied to the servo controller DAC input.

- **Proportional** – This value is derived by directly multiplying the Kp term by the position error.

Proportional = (Kp) (Position Error)

The proportional count result is applied to the servo controller DAC input, either directly or scaled.[2]

- **Integral** – This value is derived by first adding the computed error to an error total and then multiplying it by the Ki value.

Integral = (Ki) (Error Total ± Current Error)

The integral count result is applied to the servo controller DAC input, either directly or scaled.

- **Derivative** – This value is derived by multiplying the Kd term by the difference between the position error found in this sample time minus the position error found in the previous Kd sample period.

Derivative = (Kd) (Error – Previous Error)

The Derivative element can be updated in a different time interval than the other two elements. This will increase its dynamic range, allowing the system more time to react (high inertia systems). The derivative count result is applied to the servo controller DAC input, either directly or scaled.[2]

[2] In the *National* LM628 the Kp and Kd terms are divided by 16, and the Ki by 256, to provide the proper scaling.

The next step is to understand how each element is determined, and how they interact with each other.

Determining PID Elements

Have you ever read about or tried to think of an analogy for a PID gain structure that you can quickly relate to? The generally accepted analogy relating shock absorbers to Kd and springs to Kp has not allowed sound discussions to prevail, and an analogy for Ki to this point has yet to be *solidly defined.* So, let's begin understanding PID gain structures by relating them to something we are all intimately familiar with . . . our bodies.

Activating your Kp

- Stand up straight with your arms at your sides.
- Have an assistant read the following directions to you while you close your eyes.

Remember . . . SAFETY FIRST!!

- Carefully lift one foot off the floor, and place it behind the other at ankle level.
- Immediately, you will feel your supporting ankle reacting to changes in your body's balance. This is your Kp error correction system reacting to the instantaneous changes in balance your mind is sensing. The update period is the time it takes for you to mentally record and correct for these changes.

Your mind is using learned Kp gain values to compensate for offsets in your balance. As you feel your balance change, your mind multiplies your Kp value by your balance offset, or error. It then applies the adjustment value calculated to your ankle. If you consciously put your learned Kp number to zero, your system stops reacting, and you will fall. If on the other hand you raise your Kp value too high, your ankle will constantly over-correct (i.e., oscillate out of control). This action is exactly how the Kp will respond in your motor system.

Activating your Kd

- Slowly raise your arms until they are extending outwardly to either side of your body, parallel to the ground. Try to maintain them in this position for the remainder of the analogy. Note this position as zero.
- After a few moments, you may notice that your arms have shifted (possibly abruptly) to a new attitude in order to help maintain your balance.
- Notice further that your arms may maintain the new position for a period of time longer than the normal update period created by your ankle Kp. Your arms may continue to shift to new positions where a return to your zero point is nearly impossible.

This adjust-and-hold-over-time method of reacting to body balance change is the Kd reacting to error differences at the end of given time intervals (your Kd update). The update time for your arm (Kd) reaction may be different from the time required to update your ankle (Kp) reaction. Relating this back to the real world, your actual mechanical system may also require different update time periods for Kd and Kp. The Kd has an update time period that can be calibrated to the mechanical requirement. A starting point for the Kd update time period is 5 percent of the system mechanical time constant and should never need to be greater than 10 percent.

Activating your Ki

- At this point, try to maintain your body at the zero attitude position, which is standing straight on one foot, eyes closed, arms out horizontally from the body.
- Notice that your torso may shift from the vertical position at a slower rate than the Kp or Kd, then recover slowly. You are mentally monitoring your balance on your way back to zero.
- Further notice that as you recover closer to zero, your ankle (Kp), arms (Kd), and torso (Ki) never stops correcting, but will make smaller corrections over time.

The long term Torso loop readjusts itself continually with increasing or decreasing amounts to help pull the system into alignment and maintain zero error. Your torso correcting over the long term represents Ki. As your body moves to catch up to where your mind (the controller) says it should be, the newly computed Ki values may still be large (your body is still leaning or bent over), but in time the new Ki value percentage in relationship to the total Ki error is smaller, thus reducing its effect. Once your body stabilizes, the Ki value will change as required to restore your initial balanced position (following error). Notice that your Kp and Kd body elements meanwhile are working toward *holding you up* while the Ki element is working toward straightening you up.

It should be obvious that if the total Ki is allowed to grow to a value that is too large for your body (your system) to handle, you will lose control and fall. To prevent an excessive torso reaction, the mind invokes a limiting (Il) value. Your body will not lean beyond this limit. However, notice that because of this your ability to recover is slightly degraded (following error). Also notice that if your body is not limited in the way it can react (Integral limit (Il) set too high), you will overshoot your target position and sway around the position point.

In a PID control system, at the beginning of any move (zero to some velocity) the newly computed total Ki error value is large in comparison to the *initial* value when the system was not moving (horizontal system). As the move continues, the total Ki error value will grow. Newly generated Ki error values will, therefore, be a progressively smaller percentage of the overall Ki value until the Ki limiting value is reached.

Hopefully, this analogy with PID gain structure is helps you to understand PID gain structure in motion control. Refer back and forth freely as you see fit.

What the PID Elements Do

Setpoint (SP):

The SP is simply a fixed bias value or reference used to secure a fixed output voltage or current from the DAC. This value will sum together with the error portions of the PID structure. To understand the function of the SP, let's assume Kp, Ki, and Kd are zero and SP is some arbitrary value. The total DAC output voltage then will be derived from the SP value. Since there is no error correction applied, the servo motor will try to rotate at the SP speed and in the SP direction.

So what could the SP be used for? Perhaps an application could be a vertical system without a counterbalance. Since gravity will exert a downward force in addition to load weight, a small to moderate SP count that is dependent on the weight of the system could be applied to the DAC input to counteract this effect. However, the servo motor drive amplifier balance or offset control accomplishes the same task (provided the drive amplifier has one). It is possible for the PID gain functions to do this operation.

In general, the SP value should be used discriminately and in small increments.

Proportional:

The proportional element was described as:

$$Proportional = (Kp)(error)$$

Note that the proportional element is a proportioned error value, and its effect is immediate as the result is applied to the DAC at the end of each servo loop update period. Since the proportional element deals with an immediate proportioned error value, its function is to compensate for immediate changes in servo error position. Changes in position generally occur during acceleration, deceleration, and in moves where velocity changes occur because of on-the-fly changes in the system dynamics.

Anytime the system dynamics exceed the ability of the motor to keep up, the proportional element will immediately try to counteract the change in the newly generated following-error. The proportional element is the prime element used to determine the stiffness desired to hold the system position at rest.

A method for setting the proportional gain is given at the end of this chapter.

When adjusted correctly, the proportion loop element will control the motor "stiffness." The slightest deviation of the system from the commanded position will cause an immediate Kp correction to be generated.

The Kp value is not really designed to be the sole mechanism in maintaining the stability of the motor. Using the following example, it will become clear that all of the PID elements are necessary to round out the control loop. All of the examples discussed will use the mechanical concept shown in Figure 11.1.

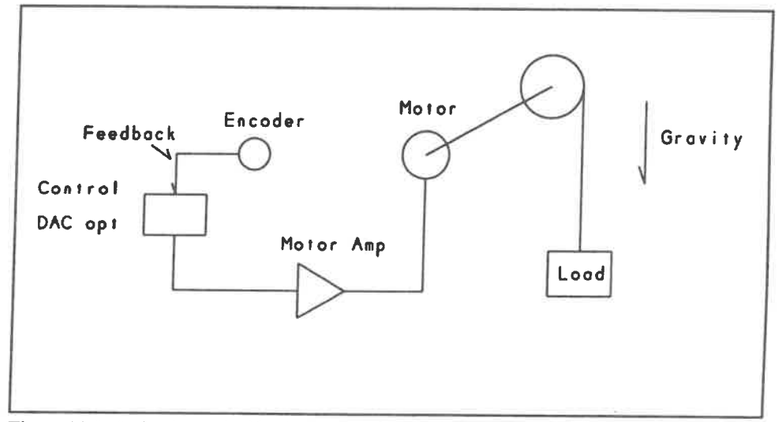

Figure 11.1 A simple vertical load servo motion model. The load will accelerate at a rate of 20 counts per update² due to gravity.

Example #1: Vertical system at rest.

Load weight must be capable of moving the system 20 encoder counts-per-update-period in the down direction, due to gravity.

12 bit DAC Kp = ? Ki = 0 Kd = 0

100 DAC input counts will attempt to overcome the effects of gravity by moving the system up five actual counts. Can Kp solely stabilize the system to zero? Set the Kp to 10.

Update #	Error	DAC Input in Counts
1	-20	(10) (-20) = -200
2	-30	(10) (-30) = -300
3	-35	(10) (-35) = -350
4	-37.5	(10) (-37.5) = -375

The system appears to be stabilizing at 40 counts below zero.

Now, set Kp to 30.

Update #	Error	DAC Input in Counts
1	-20	(30) (-20) = -600
2	-10	(30) (-10) = -300
3	-15	(30) (-15) = -450
4	-12.5	(30) (-12.5) = -375
5	-13.75	(30) (-13.75) = -412

The system appears to be stabilizing at 12 counts below zero.

Set Kp to 50.

Update #	Error	DAC Input in Counts
1	-20	(50) (-20) = -1000
2	10	(50) (10) = 500
3	-35	(50) (-35) = -1750
4	32.5	(50) (32.5) = 1625

The system is now oscillating out of control.

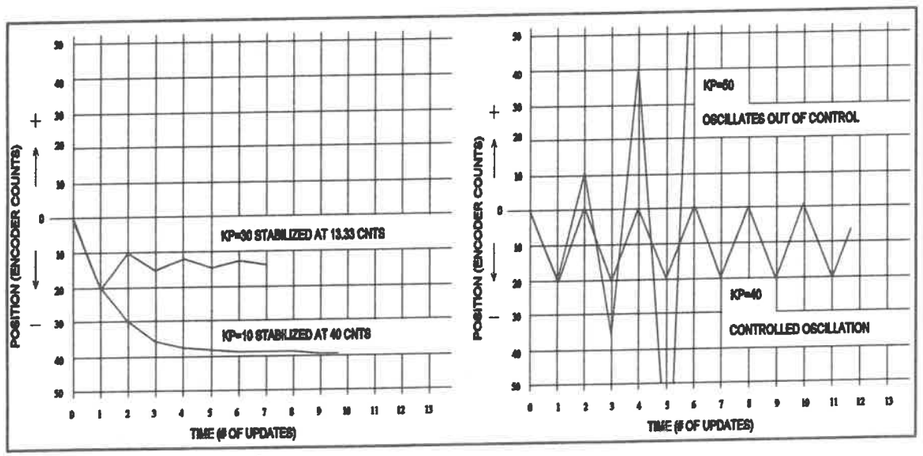

Figure 11.2 Kp response graph.

The system will oscillate continuously once the Kp is large enough to allow the system actual position to either recover to the zero position point, or to crossover the zero position point. This is because the error will force the DAC voltage to zero upon reaching the zero position (thus, recycling the controlled oscillation), or it will produce a reversed DAC output when it exceeds the zero position point that will actually help gravity and drive the load in the downward direction.

If you try other Kp values, you will find that there is really no value of Kp that will simply stop the oscillations, **and** hold the system at the requested zero position point (see Figure 11.2).

Integral:

The integral element was described as . . .

Integral = (Ki) (Last Error Total ± New Error)

The integral value is shown as a long term accumulation of error values over time. The time element of the result appears from the fact that the new error is summed to previous error totals in successive update periods. When an update period has elapsed, a new integral value is calculated. Since the integral is accumulating error count value over time, the integral will either grow or decay as succeeding calculations are made.

Note, that the integral element has more instant effect at the beginning of a move profile through instant correction, but its real purpose is to compensate for motor loading and close up following-error as the move progresses. Eventually, the new error is a small percentage of the total error and begins to simulate more of a setpoint condition.

Example #2: Vertical system at rest.

Load weight: capable of moving the system 20 encoder counts per update period in the down direction due to gravity.

12 bit DAC Kp = 0 Ki = ? Kd = 0

100 DAC input counts will attempt to overcome the effects of gravity and move the system up five actual counts. Can Ki itself stabilize the system to zero? Set Ki = 10. (See Fig. 11.3)

Update #	Error	DAC Input in Counts
1	-20	0 + (10) (-20) = -200
2	-30	-200 + (10) (-30) = -500
3	-25	-500 + (10) (-35) = -750
4	- 7.5	-750 + (10) (-37.5) = -825
5	13.75	-825 + (10) (13.75) = -687
6	28.12	-687 + (10) (28.12) = -406
7	28.43	-406 + (10) (28.43) = -121
8	14.53	-121 + (10) (14.53) = 23

Ki is too large . . . Set Ki = 3 (See Fig. 11.3)

Update #	Error	DAC Input in Counts
1	-20	0 + (3) (-20) = -60
2	-37	- 60 + (3) (-37) = -171
3	-48.45	-171 + (3) (-48.45) = -316
4	-52.63	-316.3 + (3) (-52.63) = -474
5	-48.91	-474.2 + (3) (-48.91) = -621
6	-37.86	-621 + (3) (-37.86) = -734
7	-21.13	-734.5 + (3) (-21.13) = -798
8	- 1.13	-798.2 + (3) (-1.13) = -
9	18.95	-801.63 + (3) (18.95) = -745
10	36.19	-744.77 + (3) (36.19) = -636

It appears as though the integral element itself cannot do the job. Let us combine both the proportional and integral elements to stabilize the system position.

Set . . . Kp = 30 Ki = 3 DAC = proportional + integral

Update #	Error
1	-20
2	-7
3	-12.45
4	-7.85
5	-8.97
6	-7.07

Update #	Error
7	-6.96
8	-5.97
9	-5.57
10	-4.93
11	-4.51
12	-4.04

This combination can now stabilize the system. Notice that the system is not fluctuating as wildly as in the two previous examples. Also note that the ideal stabilization value is 400 counts or five counts of actual motion for 100 counts of DAC input. However, also note that stabilization took about eight update periods. How can we speed this up without degrading stability? Figure 11.3 shows a response with Kp = 20 and Ki = 10.

To sharpen the system response to error changes, increase the Ki, reduce the *windup* of Ki over time, and set the Integral summation limit (Il). The total value of the Ki result can be constrained by setting the Integral gain limit (Il) to any value that stops the accumulation of the summation total. This will allow larger Ki elements to respond rapidly to the calculated errors without allowing the accumulated Ki total grow to unstable proportions. As the Ki increases, the Kp may have to be decreased, since the initial reaction of the Ki is similar to that of the Kp term.

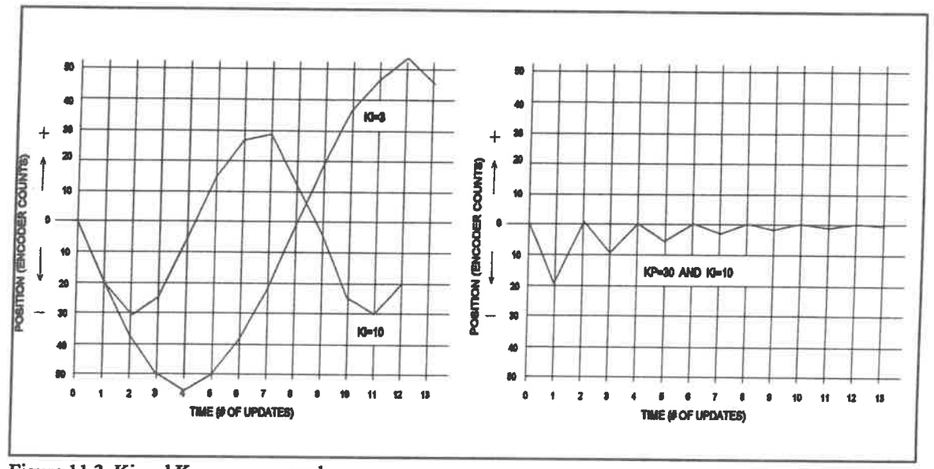

Figure 11.3 Ki and Kp response graph

Let us repeat the last workup with ideal Kp, Ki, and Il settings . . .

Kp = 20 Ki = 30 Il = 400 Kd = 0

Update #	Error
1	-20
2	0
3	0
4	0
5	0
6	0

Update #	Error
7	0
8	0
9	0
10	0
11	0
12	0

Vary the Il.

Set . . . Kp = 20 Ki = 30 Il = 300 Kd = 0

Update #	Error
1	-20
2	5
4	5
5	5
6	5

Update #	Error
7	5
8	5
10	5
11	5
12	5

Set . . . Kp = 20 Ki = 30 Il = 500 Kd = 0

Update #	Error
1	- 20
2	5
4	- 2.5
5	1.25
6	- 0.625

Update #	Error
7	0.313
8	- 0.156
10	0.078
11	- 0.039
12	0.020

Note that:

- The error originally transitioned to zero within one update period.
- The system will be stable only with the right selection of PID element values.
- When using Ki, always set the (Il) limiting term.

When tuning servo systems, each element must be considered unique. Learning a procedure well can make your tuning effort easier. Understanding the PID elements, their interaction within the servo loop and learning how to read your system response is essential.

A method for setting the integral gain for a high inertia/friction linear system is given at the end of this chapter.

Derivative:

The Derivative element was described as . . .

Derivative = (Kd) (Error - Previous Error)

The derivative element is shown as a term that reacts to a **change** in error over the derivative sampling time. When we are in a derivative update sample period, the derivative value is calculated by multiplying the Kd value by the *current* error **minus** the error calculated in the *previous derivative* update sample-time. Since the derivative is only active if the error value has **changed** from one update calculation to the next, it will not affect a stable system and can be used to **simulate** a feed-forward gain element. Note that the Kd term has more effect at the start and end of a move profile when velocity values change or when there is a change in the system loading.

To show the differential effect, let us continue with another example of the vertical system analogy.

Example #3: Vertical system at rest.

Load weight: capable of moving the system 20 encoder counts per update period in a downward direction due to the effect of gravity.

12 bit DAC Kp = 30 Ki = 3

100 DAC input counts will try to overcome the effects of gravity and move the system up five actual counts.

Set Kd = 7

Update #	Error
1	-20
2	- 3
3	-10.45
4	- 1.95
5	- 5.51
6	- 1.80
7	- 4.7
8	-1.6
9	- 2.5
10	- 0.48

Update #	Error
11	- 1.37
12	- 0.44
13	- 1.03
14	- 0.38
15	- 0.57
16	- 0.11
17	- 0.30
18	- 0.10
19	- 0.23
20	- 0.08

Reviewing the results, the system appears to have stabilized at the sixth update whereas the previous example needed over ten updates without the derivative element. Although the system is still closing in on the zero position, it is evident that by choosing correct PID gain values this can occur much sooner. Also, if you use higher gain values and allow the error to cross over the desired position point, the system will oscillate. The PID element values you select should be set to yield fast system stabilization.

It should be apparent that because the differential element reacts immediately, similarly to the proportional element, it will also help to stabilize the move similar to Kp. The differential term will retain its applied correction value for the number of update periods you prescribed, up to 65 milliseconds. This feature allows the Kd to act as a feedforward gain control. Instead of reacting to every error present in every update period, the Kd update time allows the system to respond. A starting point for the Kd update time is 1/10 of the system mechanical time constant (see Chapter 10, lines 3060 through 3070).

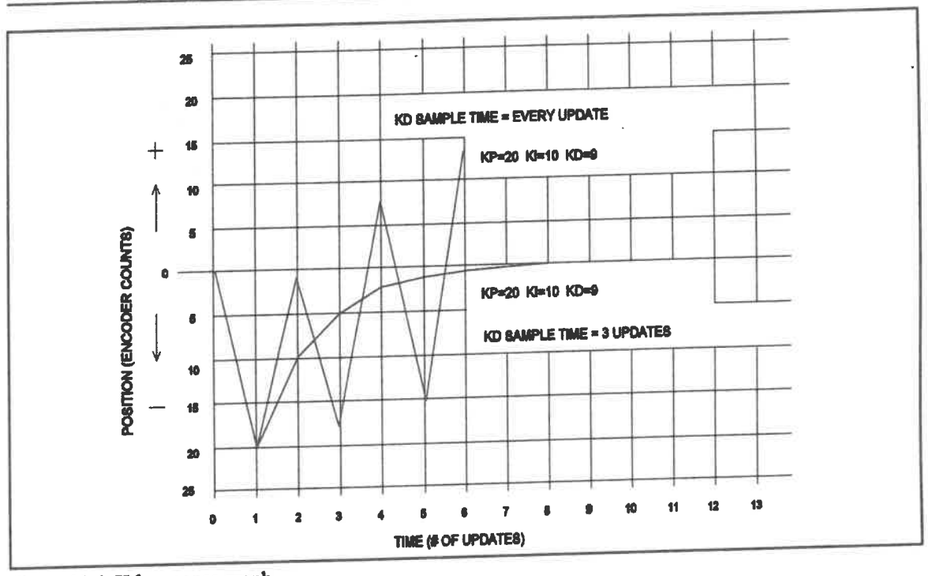

Figure 11.4 Kd response graph.

The generally accepted Kd analogy is to envision the derivative term as an automobile shock absorber. The shock is designed to react to changes in the car's attitude over time. The reactive, shock absorber applied force is held for a period of time to prevent violent wheel oscillations.

The elements are interactive; and depending on what you are trying to accomplish in tuning your system, you must know how the elements react.

A method you can use for setting the derivative gain term for a high inertia/friction linear system is given at the end of this chapter.

Discussion

The normal PID filter operation, previously discussed could possibly produce acceleration takeoffs too severe for the load. This is due to the linear calculations of the velocity loop and the reactive nature of the PID elements and the trapezoidal profile generation when a move begins. With some systems, it would be more desirable to produce an *S* curve acceleration/deceleration profile to soften the inertial loading of the motor system.

If this situation exists in your system, there are two possible cures:

1) Try using gain break accel/decel profiles. Use the PID values necessary to maintain good loop following response while running at the 60 percent (or greater) velocity point. Before achieving the 60 percent velocity point, use reduced (or different) PID loop values. This will allow the system inertia to develop some the following error on takeoff to soften the loading. Gain breaks are generally placed at the 30 and 60 percent velocity points.

2) Use an *S* curve acceleration/deceleration profile. The *S* curve is presented in Chapter 10. A graph comparing *S* curve versus trapezoidal acceleration is illustrated in Figure 10.1.

The objective of either of these methods is to gradually change the commanded acceleration/deceleration value until it is safe to apply full capacity to the system without damaging, shifting, losing control of the load, or prematurely wearing out the mechanics.

A PID Tuning Approach

The prime purpose behind any gain algorithm is to produce smooth, stable motion. Of all of the gain algorithms currently in use, the PID seems to be the most popular. When I am asked why, the only reply I can come up with is that the PID approach seems to be the most ***intuitive*** of all of the gain structures I have come in contact with. That is to say, its variables allow me to set up smooth, stable motion (tune the system) with the least possible effort, and in the shortest possible time.

As I do more and more tutorial and training sessions with motion control engineers, I realize that this statement may not be as true as I would like to believe. It has become apparent that a more defined procedure, or method, to tune the gain loop is necessary. There has to be more than "luck" involved when a user is required to tune his or her application. The use of current or voltage motor amplifiers, a vertical or horizontally mounted axis, and high or low friction existing in the traverse, are only a few of the many variables that exist in any given system. Since any combination of these variables may be lurking in any system, it points to a need for an established tuning procedure instead of widely diverse methods.

I have used the procedure outlined below extensively for the *National* LM628 PID structure. I have also used a slightly modified version of this tuning method with variations of the *National* PID, such as in long interval process control loops, the PMD DSP motion control chip, PLC's, and others. A variation of the *National* LM628 PID would be one set up as a PD with Feed-Forward, for example.

The key to this method, or any other tuning approach for that matter, is to completely understand the gain term variables in order to make the tuning approach ***intuitive***. You must know not only what gain element to adjust, but in what order to adjust it. It is extremely important to learn how to "read" what the motion is telling you, either visually, by sound, feel, or with the help of an oscilloscope connected to strategic signal points, in order to bring the tuning exercise to a successful completion.

A PID Tuning Procedure

Term definitions for this procedure: *Step* refers to the major steps of the procedure. Lines (line) refer to the sub-steps within the major steps.

Step 1) Set the Kp:

1) Set the **Kp** to 2 (or some arbitrary low value)
2) Make a short move to simply **bump** the motor
3) Is the motor oscillating when stopped?
   If *Yes* - Goto line 1.6
4) Increase the **Kp** (example: Kp = Kp + 5)
5) Goto line 2
6) Reduce the **Kp** until the motor stops oscillating
7) Set the **Kp** to (a value of the Kp found in Step 6) (.9)
8) Is the **Kp** > 75 ?
   If *Yes* - Goto *Step #5*

Step 2) Setting the IL:

The purpose of the Integral Limit is to reduce position error at standstill. The value of IL should not be set to a value higher than what it will take to do this.

1) Set the **IL** to five times the Kp value
2) Set the **Ki** to 5
3) Make a short move (10000 encoder counts for example)
4) Wait for 10 seconds after the move has completed
5) Read the actual position
6) Is the actual position within ±5 counts (within allowable tolerance) of the desired position?
   If *Yes* - Goto *Step #3*

Note: The KI value may be raised slightly at this point to allow the axis to respond a bit faster, but **remember** that the objective here is to tune the IL, which takes time, and not allow other gain elements to cause motion to overshoot. The results of this action would invalidate the setup of the IL term.

7) If the move is short of the desired position, raise the **IL** value (small increments . . . 10-30 points for example)
8) If the move is beyond the desired position, lower the **IL** value (small increments . . . 10-30 points for example)
9) Goto line 3 (above)

Step 3) Setting the Ki:

The **Ki** term is used to control the response of the Integral Limit (**IL**) action of reducing position error.

1) Increase the **Ki** (example: Ki = Ki + 5)
2) Does the motor oscillate at the end of the move ?
   If *No* - Goto line 1
3) Reduce the **Ki** until the motor does not oscillate

Step 4) Setting the Kd and SI:

If the system is going to have large load or inertia variations in excess of 6:1 during the motion, as would be encountered on a drilling machine when the drill makes contact with the workpiece, the action of the **Kd** gain would be needed.

The **Kd** reacts to and controls variations in velocity by noting changes in the actual encoder position (counts) over time (the sample time). It is related to velocity feedforward in that it can respond faster than the system mechanics, and over longer time intervals (SI) than the other PID terms. This allows the **Kd** to anticipate and head off wide mechanical velocity swings before they occur.

1) Set **SI** to equal to its minimum sample interval. (256 microseconds for the LM628)
2) Set the **Kd** value to zero
3) View the actual axis motion with an oscilloscope by monitoring the velocity output signal of the motor amplifier, an analog tach., the DAC output of the motion controller card, or any other suitable signal.
   a) Make a move and allow the load change to occur.
   b) Record the actual velocity excursion as noted on the oscilloscope.
4) Raise the **Kd** value to force a lowering of the axis velocity excursion.
5) When the **Kd** achieves a value where no further increase in the **Kd** term reduces the velocity excursion, record the velocity excursion waveform as viewed on the oscilloscope.
6) Set the **Kd** value to (Kd) (50%)
7) Raise the **SI** value one increment (256 microseconds).
8) Raise or lower the **Kd** value as required, to force a reduction in the axis velocity excursion.
9) When the **Kd** achieves a value where no further increase in the Kd term will improve the axis stability for the given velocity excursion, record that velocity excursion waveform as viewed on the oscilloscope.
10) Is this better or worse than that achieved in line 5 ?
11) If *better*—Repeat steps in lines 6 through 11 as required until the optimum **Kd** and **SI** values are achieved.
12) If *worse*—Go to the previous setting for **Kd** and **SI**.

STOP HERE

This tuning procedure is now complete. Go to the *Discussion*.

Step 5) Reset the Kp:

At this point, the system is showing signs of:

- having either high response or low friction,
- being in the velocity mode operation of the motor's amplifier,
- having a high inertial flywheel effect (a rotary system for example), or
- some combination of these or other situations.

Thus, the tuning approach used, will use the Kd term as the primary tuning element to achieve a finer degree of system control.

1) Set the **Kp** to 2 (or some arbitrary low value)

Step 6) Setting the Kd and SI:

A good starting point for the SI term is a value equal to 1/10 the mechanical time constant of the axis being tuned, if you have the means of calculating it.

The Kd and SI terms will be used in this tuning method as the prime tuning device. The Kd term will react to velocity variations while the SI term will make sure that a flat line slew (motion) exists.

1) Set **SI** to equal to its minimum sample interval—256 microseconds
2) Set the **Kd** value to 10
3) While viewing the actual motion with an oscilloscope by monitoring the velocity output signal of the motor amplifier, an analog tach., the DAC output of the motion controller card, or some other suitable signal, make a profile move. Record the actual velocity profile as noted on the oscilloscope.
4) Raise the **Kd** value until the actual acceleration ramp peak achieves the desired velocity value.
5) Is the move slew period following the acceleration ramp a relatively flat line?
   If *Yes* – Goto *Step #7*
6) Does the point of transition from slew to deceleration appear to have an overshoot?
   If *No* – Goto line 7
   If *Yes* – Reinstall the last **Kd** and **SI** values
7) Set the **Kd** value to **(Kd)** (50%)
8) Raise the **SI** value one increment . . . SI = SI + 1
9) Goto line 4

Step 7) Readjust the Kp:

1) Make a profile move
2) Is the motor oscillating at the end of the move?
   If *Yes* – Goto line 5
3) Increase the **Kp** (example: Kp = Kp + 5)
4) Goto line 1
5) Reduce the **Kp** until the motor stops oscillating

Step 8) Setting the IL:

The Integral Limit is set up to reduce position error. The value of IL should not be set to a value higher than what it will take to do this job.

1) Set the **IL** to the current KP value
2) Set the **Ki** to 2
3) Make a short move (10000 encoder counts for example)
4) Wait for 10 seconds after the move has completed
5) Read the actual position
6) Is the actual position within ±5 counts (or some allowable tolerance) of the desired position?
   If *Yes* – Goto *Step #9*.

> **Note:** The Ki may be raised sightly at this point to allow the axis to respond a bit faster, but the object here is not to allow overshoot of the motion, otherwise the results of this setup will be invalid.

7) If the move is short of the desired position, raise the **IL** value (small increments. . . 10-30 points, for example)
8) If the move is beyond the desired position, lower the **IL** value (small increments. . . 10-30 points, for example)
9) Goto line 3.

Step 9) Setting the Ki:

The Ki term is used to control the response of the Integral Limit (IL) action of reducing position error.

1) Increase the **Ki** (example: Ki = Ki + 5)
2) Does the motor oscillate at the end of the move?
   If *No* – Goto line 1
3) Reduce the **Ki** until the motor does not oscillate

This tuning procedure is now complete.

Discussion

There is actually only one tuning approach used here which will follow one of two different paths depending upon what the results of the initial Kp value becomes (section 1). These two paths will allow a first time tuning operation to happen in a relatively short amount of time. The important part of this whole exercise is to gain an understanding of the gain terms, and what they're capable of controlling. It cannot be left to "luck". It was once indicated to me that tuning was more of an "art" than a "science". I hope this procedure, as simple as it may appear, can put tuning back into the realm of an engineering exercise.

Conclusion

When tuning any system, first determine the system type. Optimizing response is nothing more than insuring that the system can accomplish the necessary profile: in the allotted time, without compromising system performance and without damaging the product. Why operate the unit at 100 feet-per-minute when it can do the required specification at 50 feet-per-minute? You might gain product throughput, but you could possibly experience side-effects such as premature mechanical wear and tear. If you need to run at 100 feet per minute make it a design specification. Do not just arbitrarily run it faster.

If the unit will be performing multiple axis interpolation, the resolution and loading of the various axes must be taken into account. The following error generated by each axis, for identical profiles, should be identical. When optimizing the profile, use an oscilloscope. With a scope probe connected to the DAC output, first set the Kp value as described above. You will see the DAC oscillate before feeling or hearing it. Set the Il value and the Ki and run a low speed profile. As the system is moving back and forth, adjust the Il and Ki values for the best response as previously described in this chapter. While under load, continue to raise the velocity and acceleration in small increments, readjusting the Kp, Ki and Il as necessary. When you have completed this operation, try varying the load. If noticeable oscillations occur, start applying some Kd. A starting point for the Kd time base should be approximately 1/10 the system mechanical time constant.

Note that high response and rotary systems will generally require the Kd to be set first, then the Ki and then the Kp that is just the reverse.

Notes: