Chapter 16

Discussing Applications: Questions, Answers, and Issues

Features

System Response and Small DAC Changes
Hard-to-Tune System
Pneumatic Motion Control
Homing and How to Accomplish It
Flying Cutoff Control
Following Error
Telescope Application
4-Axis System, 3-Axis Control
Selecting the Optimum Encoder
Servo Motion and the Motor Voltage Mode
Update Time & Master/Slave Applications

System Response and Small DAC Changes

Someone once raised the question as to why the system described in Figure 16.1 did not respond to single DAC step changes with the following system specifications:

- Required rotary load base speed: 75 ±0.25 rpm.
- DAC resolution: 16 bit.
- Motor top speed: 2000 rpm.
- System required to make 45 degree move increments in 20 milliseconds
- Motor package operating in current mode.

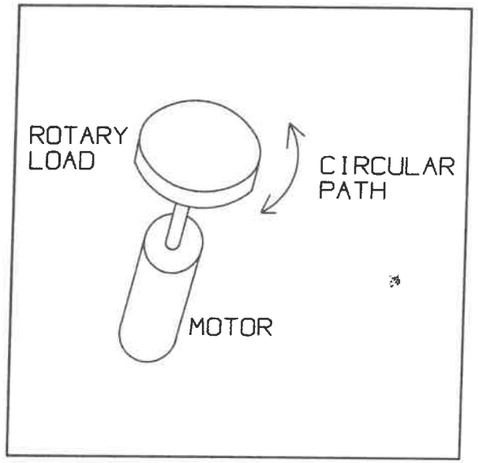

Figure 16.1 Rotary system.

The motor had enough power to do the required operations, but there were subtleties about the motion control package that the user failed to take into account.

First, there was the issue of motor and amplifier package **speed regulation** as stand alone items. The ability of the amplifier to maintain motor rotation without excessively oscillating about the requested speed setpoint was critical to this operation (see Figure 16.2). Motor package operation in *current mode* was not going to yield the best stability for high speed indexing (i.e., system friction); therefore, I recommended that the user operate in *voltage mode*.

If we remove the motor drive signal and apply a fixed DC voltage to the motor amplifier and then monitor motor speed with a digital **Strobotac** with 1 rpm resolution, any variation in speed would be a direct result of the combined motor/amplifier regulation. By knowing the speed regulation of the driving package, we can figure out whether the system will fit the specified stability criteria. In this case, the user operated the system in *voltage mode* with no tachometer. This meant that system stability exceeded 1 percent (±10 rpm). Paying attention to the motor and amplifier speed regulation will determine how reactive the gain structure of the controller must be in order to maintain smooth motion.

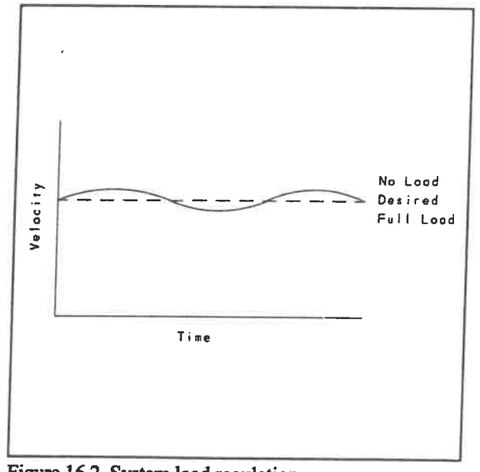

Figure 16.2 System load regulation.

Since the motor had to operate at 3.75 percent of maximum speed, I suggested using one of the following:

- A gearbox to increase motor rpm,
- A lower speed motor winding to reduce motor top speed and therefore, the instability,
- A motor designed specifically for low rpm use,
- A velocity voltage feedback loop (such as a tachometer) to increase the motor stability at low rpm.

The second issue was the use of a 16-bit DAC. There is a tendency to feel that the higher the DAC resolution, the more stable or controllable the system becomes. However, control is not a function solely of the DAC resolution, but also of the motor amplifier **sensitivity** (gain) to the voltage applied at its signal input pins. There are 32,767 steps on each side of the 16-bit DAC range (±10VDC). Since this system is using a voltage drive, each DAC step results in a desire to change the motor speed by 0.0610 rpm (2000rpm @ 10VDC). With the sensitivity (gain) of the motor amplifier at 1 millivolt (mV), the 16-bit DAC controller will cause a change in the actual motor speed every 3.2768 DAC steps (see Figure 16.3).

Discussion

Speed-regulation, and **sensitivity** of the motor amplifier package dictates how the motion control computer will handle the system. For continuous motor velocities that operate in the bottom 10 percent range of rated rpm, I heartily recommend using a motor designed for low-speed operation. If, for whatever reason, this motor design cannot be an option, then use some form of gearing to boost the motor rpm into a more suitable operating region. You should then consider the operating mode of the motor package—voltage or current. Since this system required high speed indexing, *voltage mode* using a tachometer velocity loop is preferred.

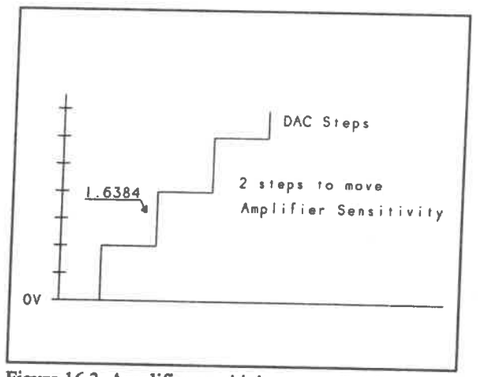

Figure 16.3 Amplifier sensitivity versus DAC step height.

The third issue facing a designer using high-speed small-move indexing is that the PID gain structure requires an **error**[1] to generate the DAC voltage. In order to make the required move of 45 degrees, the trajectory generator would be required to move 250 counts (500 line encoder x 4 = 2000 counts per revolution ÷ 8). The National LM628 processor will generate a DAC output based on the update formulas shown in *Example 1* below. It should be evident that a relationship exists between the smallest move size and system **resolution**.

[1] Error is defined as the difference between where the computer calculates it should be, and where the encoder counter indicates it actually is. (See Chapter 11.)

Knowing how the gain structure works, we'll use the Kd parameter as the prime factor in handling the move. Observing the CPU control signal with an oscilloscope, we'll apply a 10VDC step to the motor amplifier. Then:

- Record the time required to go from rest to rated velocity.
- Set the derivative sample time between 5 and 10 percent of the response time noted in the above step.
- Next, adjust the Kd PID term until the system indexes with stability.
- Use maximum velocity and acceleration limits in order to ensure that the move performs within the required profile time of 20 milliseconds.

The key is to not exceed the required profile time by more than 5 percent, since inertial loading goes up by square functions of the allotted time. In other words, small changes in time can cause large changes in torque-loading which can cause unstable reactions and require larger motor torques. Also, the Kd term will be used in a gain-lead fashion allowing the motion CPU to *kick* the indexing system into place. Ki will only be used to *tweek* the system into final position (very small doses), and Kp will also be used in small doses to maintain axis stiffness (see Chapter 11: PID Tuning).

The fourth and final issue is to ensure that the system **update time** is capable of handling the system speed. If the update period were 1 millisecond, there would only be 20 update calculations accomplished in the move. In this system the update period is 256 microseconds. This will result in 78 update periods from the start to the end of the move (20 / 0.256). The number of updates and the system resolution must be properly coordinated with a gain structure that ensures move stability. The more updates performed in a move profile (providing the resolution has been properly figured out for the move size, the acceleration rate, and the top velocity requirement), the more stable the moves can be performed.

Example 1: A Simple Trapezoidal Trajectory Control Algorithm

Update:

```
if (Traj_Position < Decl_Position)
   {   Velocity = Velocity + Traj_Accel;
         if (Velocity >= Max_Velocity)  Velocity = Max_Velocity;
   }
else  {   Velocity = Velocity - Traj_Accel;
            if (Velocity < 0)    Velocity = 0;
         }

Traj_Position = Traj_Position + Velocity;
Error_Cnt = Traj_Position - Real_Position;

/* Proportional Kp */
Prop_Val = Kp * Error_Cnt;

/* Integral Ki */
Integ_Err_Sum = Integ_Err_Sum + Error_Cnt;
Integ_Scale = Ki * Integ_Err_Sum / 16;
if (Integ_Scale > Integ_Lim) Integ_Scale  = Integ_Lim;
if (Integ_Scale < -Integ_Lim) Integ_Scale = -Integ_Lim;

/* Differential Kd and DST */
Differ_Act_Tme++;
if (Differ_Act_Tme >= Differ_Smple_Tme)
   {   if ((Error_Cnt >= 0 ) && (Last_Eror >= 0))
            { Eror_Diff = Error_Cnt - Last_Eror; }
         else if ((Error_Cnt < 0) && (Last_Eror < 0))
              { if (abs(Error_Cnt) > abs(Last_Eror))
                   { Eror_Diff = abs(Last_Eror)-abs(Error_Cnt); }
                else
                   { Eror_Diff = abs(Error_Cnt)-abs(Last_Eror); }
              }
         else { if ((Error_Cnt >=0) && (Last_Eror < 0))
                 { Eror_Diff = Error_Cnt + abs(Last_Eror); }
                 else if ((Error_Cnt < 0) && (Last_Eror >= 0))
                      { Eror_Diff = - (abs(Error_Cnt)+abs(Last_Eror)); }
              }
         Diff_Val = Kd * Eror_Diff;
         Differ_Act_Tme = 0;
         Last_Eror = Error_Cnt;
   }

DAC_Steps = Prop_Val + Integ_Scale + Diff_Val;
if (DAC_Steps > Max_DAC) DAC_Steps = Max_DAC;
else if (DAC_Steps < -Max_DAC) DAC_Steps = -Max_DAC;

UpdateExt:     Return from the Trajectory handler routine
```


System Hard to Tune, Unable to Make Rate, Mis-positioning

This problem bothered me. Computer control mis-positioning an axis can be caused by literally anything: couplings, noise, bad encoder, bad CPU, and the list goes on. I led the user through a series of tests and checks to determine the problem, and I wound up uncovering four separate problems, each producing a different effect on the operation at various speeds and accelerations.

Figure 16.4 shows the motor system operating in tangential mode controlling a set of nip rollers pulling the product along (Figure 16.5). To make rate for any profile, the control had to accelerate from zero to 2000 rpm in under 50 milliseconds. This was just over the system step response capability of 43 milliseconds.

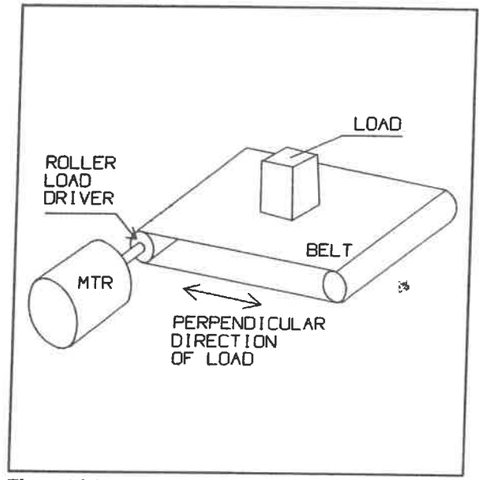

Figure 16.4 A tangential system.

The system could resolve 1000 counts per inch and could operate in voltage mode with a low inertia *pancake* style motor well suited for this application. The encoder was a 500-line unit (2000 count/revolution).

I noted three problems:

1) Accumulated position error at low speed ranged in the vicinity of 1/32 inch per index.
2) Accumulated error at maximum speed ranged in the vicinity of 1 inch per index.
3) The system did not respond properly to the PID terms as my experience indicated it should have.

In general, to troubleshoot software problems you approach them one at a time, since each problem you encounter usually alters operational flow. However, when dealing with a machine, all visible symptoms can be related in one form or another to *bugs*, which require a slightly different troubleshooting tactic.

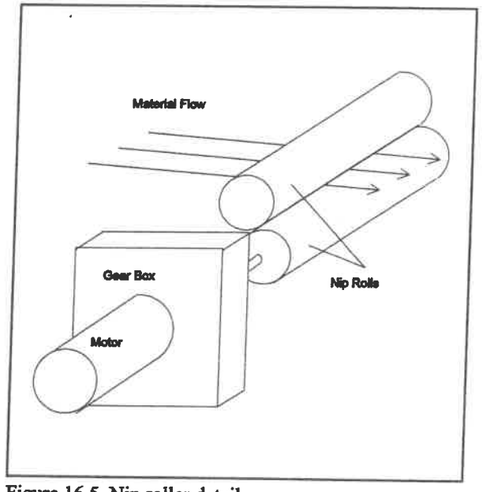

Figure 16.5 Nip roller detail.

In this case, we eliminated the possibility of noise induced mis-positioning by first observing the encoder signals with a storage oscilloscope—they were clean. Because of this, I decided to tune the system and make the index rate before finding out why the system mis-positioned. Therefore, if the reason(s) for mis-positioning became evident, tuning would correct them immediately.

I had the user lower the maximum velocity to 500 rpm and the acceleration (and deceleration) to one second. We gradually raised the value of the Kp PID term until the motor began to oscillate. The Kp equaled 50 at the point of oscillation, yet the maximum allowable Kp value was 32000. I suspected that the system leaned more toward a rotary type, and decided to begin tuning with the Kd gain function. Setting the Kd sample time to 5 percent of 43 milliseconds (approximately 2 milliseconds), we began incrementing the axis and raising the Kd value. At the point of axis instability, we lowered Kd until the axis again stabilized.

We then increased the acceleration and deceleration ramps until we created a flat top slew rate in the velocity profile (see Figure 16.6). We continued this process, retuning Kd if required and raising the maximum velocity until we met the worst case profile requirement. Upon completing this profiling operation we met all requirements for stability, velocity, and product rate.

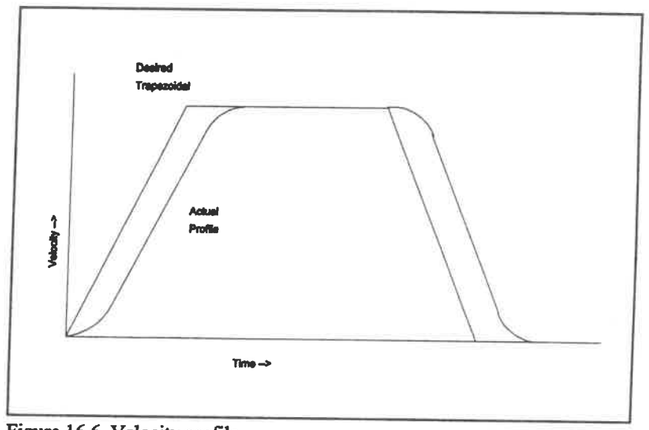

Figure 16.6 Velocity profile.

Next, we had to find the cause of axis mis-positioning. We marked all shaft coupling points and slowed the system down to a crawl. After operating the unit for a few minutes, the system deviated out of position. I immediately checked the program index routine.

Indexing a motion axis the same incremental amount time after time should be done in an *incremental* mode. The user in this case made an absolute move from zero (0), zeroed the counters, and then repeated the same absolute move profile. The problem here is that the counters zeroed at the **actual** (stopping) position, not at the **desired** (stopping) position. Because of this, the actual stopping position became the new zero position. Since it is rare that an axis will stop repeatedly at the precise (no-error) position called for, the axis will *creep* from its original home (0) position. It can do that in either the forward or reverse direction. We changed the software to do index moves in *incremental mode* rather than *absolute mode*, and the problem disappeared.

Now that the unit ran properly, we increased the system speed to once again make rate, and there it was again ... mis-positioning! We waited a moment to ensure a reasonably large deviation from the desired position and then stopped the operation. Upon inspecting the marks we had previously made at each shaft and coupling point, nothing had moved. Once again, we checked the encoder signals, and they were clean.

We decided to change the encoder—not to another brand, but to a unit with a higher acceleration rating. Problem solved. We found that the disk inside the encoder was slipping, causing mis-positioning at higher acceleration rates.

Interestingly enough, the customer had done all of his homework as far as calculating all possible inertias, ratios, requirements, etc., but had not checked the encoder data sheet to determine whether motor acceleration exceeded encoder acceleration—it did.

At the end of this session we found the following:

- Axes were tuned in the wrong order.
- An absolute move profile determined the index zero at the end of a move, rather than an incremental move profile.
- System acceleration exceeded that what axis encoder was capable of handling.
- The customer exceeded the system profile time requirements without realizing it because of multiple problems.

Pneumatic System Requiring High Speed Profile

Interesting. This caller wanted our controller to take charge of pneumatic motion. Since the gain structure of our controller is a PID loop with a very wide latitude in parameter settings, my primary concern centered not around the controller, but around the system mechanics.

Figure 16.7 shows the general idea of the user's system. Figure 16.8 shows the desired system move profile. In order to get the absolute system requirements, I asked the user several questions. I also verified that the answers I returned were complete and correct.

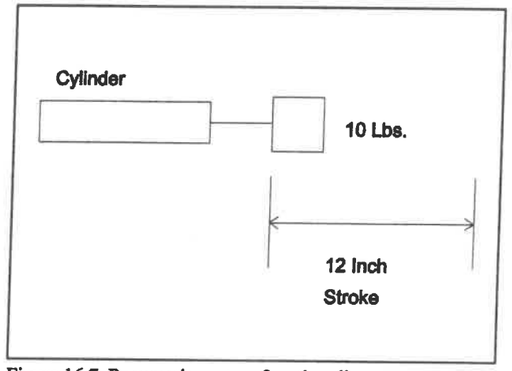

Figure 16.7 Pneumatic system function diagram.

The following is a discussion of my approach to the problem:

Taking the profile at face value, I asked the user for the cylinder inside diameter and the available air pressure. The cylinder cross section measured one square inch, and the pressure could be whatever was required.

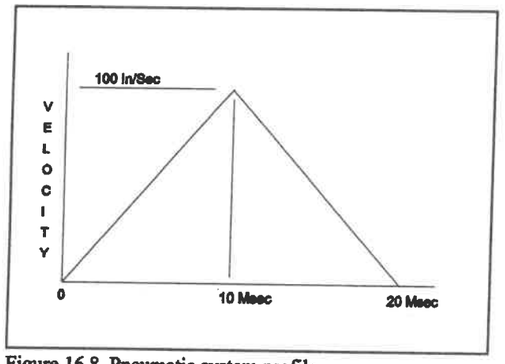

Figure 16.8 Pneumatic system profile

The volume of the cylinder is:

$$\pi r^2 L$$

$$\therefore \quad Vol = \left(\frac{\pi (0.56238)^2 (12)}{1728} \right)$$

$$= .0069 \ ft^3$$

Case 1:

What flow rate is required to meet the profile *timing?*

$$a = \frac{V}{t} = \frac{100 \frac{in}{sec}}{0.01 \ sec} = 10000 \frac{in}{sec^2} = 833.34 \frac{ft}{sec^2}$$

$$F = \left(\frac{W}{g} \right) a = \left(\frac{10}{32.2} \right) (833.334) = 258.8 \ Lbs.$$

and:

$$\left(\frac{1}{.02 \ sec} \right) (0.0069 \ CF) \left(\frac{60 \ sec}{1 \ min} \right) = 20.7 \ SCFM$$

Case 2:

Using 100 lbs of *air pressure* as a restriction, what will be the new profile timing and SCFM requirement?

$$F = \left(\frac{W}{32.2} \right) a$$

$$\therefore \quad a = \frac{(100)(32.2)}{10} = 322 \frac{ft}{sec^2} = 3864 \frac{in}{sec^2}$$

$$Vel = at$$

$$\therefore \quad t = \frac{v}{a} = \frac{100}{3864} = 0.02587 \ sec.$$

and:

$$Flow = \left(\frac{1}{0.05176 \ sec} \right) (0.0069 \ CF) \left(\frac{60 \ sec}{1 \ min} \right) = 7.998 \ SCFM$$

Case 3:

Using 2 SCFM as a restriction, what will be the new profile timing, and SCFM requirement?

$$Flow = \left(\frac{1}{x \text{ sec}}\right)(0.0069\ CF)\left(\frac{60 \text{ sec}}{1 \text{ min}}\right)$$

$$\therefore \quad x = \frac{(60)(0.0069)}{2} = 0.207 \text{ sec}$$

$$t = AclTme + DecTme = (2)\,AclTme$$

$$\therefore \quad a = \frac{t}{2} = 0.1035 \text{ sec}$$

and:

$$F = \left(\frac{W}{32.2}\right)a$$

$$\therefore \quad F = \left(\frac{10}{32.2}\right)(80.5) = 25 \text{ lbs}.$$

The problems with system control in this case become evident once you do some preliminary calculations. To ensure the desired profile, you need 258 psi at 21 SCFM. This is over twice the average air pressure found in most plants. If you drop the pressure to 100 psi, the profile time stretches out by two and a half times (50 milliseconds versus 20 milliseconds). Lowering the pressure or the CFM rating only slows the operation.

Therefore, the real question is why consider pneumatics at all when a servo system can easily solve the problem? Not only can the problem be resolved electrically, but the user will gain benefits by eliminating the low efficiency of air movers, pumps, surge tanks (accumulators), valves, and other miscellaneous air handling hardware.

Furthermore, a possible overlooked problem is that even if air pressure and flow rate **are** available at the required levels calculated in ***Case 1,*** the operating latency of forward and reverse valves for cylinder motion will generally be in the 10 to 50 millisecond range. This does not include the air compression factor, which also tends to add time to the actual move. We have already exceeded the profile specifications, and we haven't even started to move yet!

As you might have guessed, we turned down this particular job, not because we did not want to sell the user a controller. Rather, we chose not to accept the job because we could not guarantee that our controller would meet the profile specifications provided.

Most designers these days believe that high speed DSP computers or motion algorithms can solve most all mechanical problems. Don't be fooled! How well a system performs is dependent on the mechanics, not the electronics. You can enjoy real cost-effective solutions to your problem . . . **if** . . . you allow yourself a degree of latitude when ***defining your problem solution.***

What is "HOMING," and How Is It Done?

This is an interesting question. The term *homing* means different things to different people, and it can be accomplished in a multitude of ways. Basically, *homing* a system means simply providing a point of reference to which all future axis motion or programmed moves can be measured.

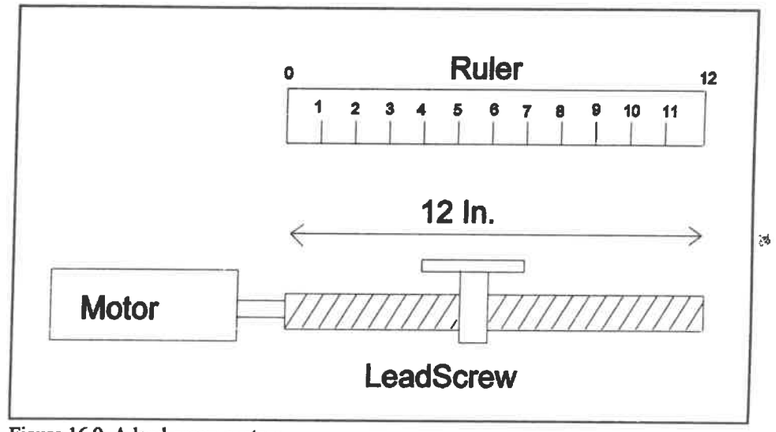

Figure 16.9 A lead-screw system.

Sound complex? Let's try a simple homing procedure. Assume a single axis motion control system like that shown in Figure 16.9. The ruler indicates that there is twelve inches of available travel. The motor can be a servo, or a stepper. (But for reasons we'll address later, different rules apply.)

What is the simplest home routine that you can do? The answer is simply to **Zero the Counters** at whatever location the slide (axis) is at.

Method 1: Simple Zero Counter

Sometime in your experience, you may have seen a machinist operating a manual machine such as a mill or a drill-press with encoders mounted on the moveable axes slides, and readout counters set up to show the slides current **actual** positions. The machinist homes a machine axis simply by manually moving it to a reference position and zeros a position counter. All subsequent axis movement is now known, and it is measured relative to this preset zero position. You might consider this a crude method, but it's really quite effective.

Though effective, accuracy leaves much to be desired. The ability of an operator to place the slide at *precisely* the same position *every time* the axis needs to be re-referenced is almost impossible to guarantee. Precision becomes especially important when the jig or clamp holding the part is physically mounted to the motion slide. An inability to consistently zero the position counters at the identical physical slide position can destroy very expensive parts.

To meet this challenge, some machines use a method of *pinning* the slide. An operator moves the axis slowly to a position where a steel pin or dowel can be inserted into a hole-set. This aligns the slide with a reference point (see Figure 16.10). Once the pin is placed into the hole-set, the counters can be zeroed.

In the case of a motor driven axis, the same method applies. *Jog* (manually move) the axis as close as possible to the desired home position and zero the counter. If the pinning method is used, jog the axis as close as possible to the pin position, power down the motor, and hand position the axis until the pin drops into place. Zero the counter(s), remove the pin, and reapply power to the motor.

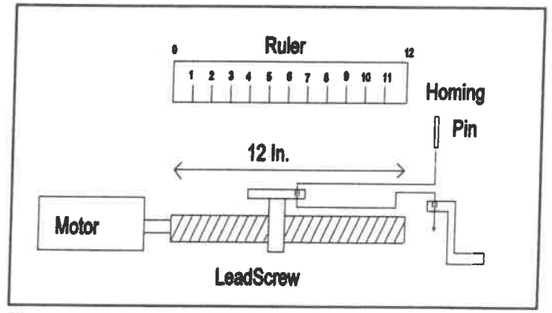

Figure 16.10 Zeroing a slide by pinning.

It really doesn't matter how the zero reference is obtained as long as the reference position allows the production of parts within the customer specification.

Method 2: Referencing to a Switch Edge

An improvement to Method 1 is shown in Figure 16.11. A switch installed on the axis signals a control or counter when it is at home position.

Note that in Figure 16.11 the home switch is actuated by the movable slide. It's not really important where the switch is located along the axis, as long as it is within reach of a moveable part of the axis. The physical switch trigger point with respect to true home position can be accounted for using a *Home Offset* software function. The Home Offset value will automatically reposition the computer internal counters to the precise home position you desire.

For example, using the method shown in Figure 16.10, the switch will trigger at about the 11.5 inch ruler position. Since we want the true home position to correlate with the ruler markings, we simply tell the control that the home offset is +11.5 inches. By doing this, the next *Return-to-Zero* command issued by the control will send the axis slide to the leftmost or home position. Since the switch does not physically move, the home offset position will always reflect the true home position, provided that the switch is *repeatable*[2]. Since most switches activate and deactivate at different

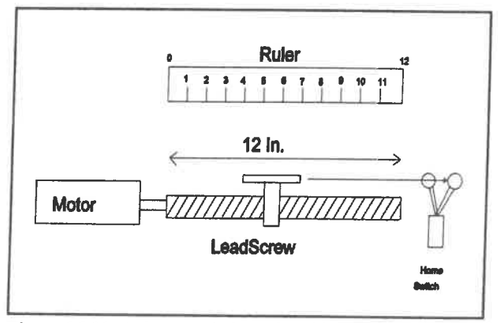

Figure 16.11 Using a switch to signal home.

[2] See Chapter 1 for a discussion on repeatability.

physical positions, a phenomenon known as switch *hysterisis*; and since most switches are not 100 percent repeatable at a given trigger position, the homing method may require conditioning to ensure an acceptable level of accuracy and repeatability.

Method 3: Referencing to a Switch Center Point

As previously stated, if a switch provides the only homing criterion, and if the axis position repeatability requirement is tighter than the switch can provide, then the following homing method might be necessary.

Figure 16.12 illustrates a switch used in an axis homing routine. Switch hysterisis in this case is 0.030 inches, typically with a ±0.005 inch tolerance. Therefore, switch repeatability is ±0.005 inches. Therefore, the objective is to home the axis within ±0.005 inches.

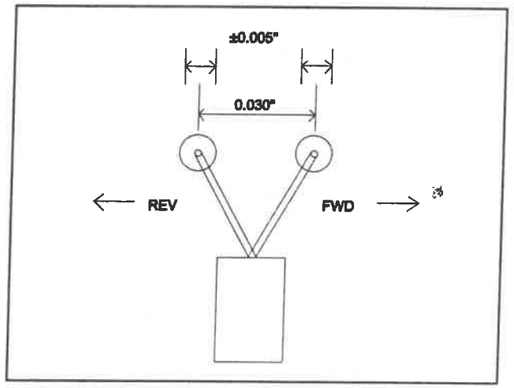

Figure 16.12 Referencing to a switch center point.

A method for doing so is as follows:

1) Move the axis forward until the switch activates. Record the position.

2) Reverse the axis until the switch deactivates. Again, record the position.

3) Repeat steps 1 and 2 above, recording the positions until the central point of the switch can be mathematically determined and still be within the required operating specifications.

As you can probably tell, this method can significantly increase the time it will take to reference the axis. If the machine were reasonably complex with multiple axes, it could take several minutes to complete the homing sequence. In addition, the amount of time necessary to home using this method will be directly proportional to the quality of the switch used.

Method 4: Referencing to an Encoder Index

Referencing to an encoder signal is the most accurate and repeatable method for obtaining an axis zero point.

A typical encoder index signal with quadrature signals A and B is shown in Figure 16.13. The index signal is shown to be an active high pulse, lasting for a period of 90 electrical degrees. The index typically occurs only **once** in a complete encoder rotation (360 physical degrees). Under no circumstances should an index signal remain active for a period exceeding 360 electrical degrees.

The reason for the restriction on the index period (360 electrical degrees), is because homing can generally be accomplished from either direction. Also, the current position with reference to the index might need to be recorded for count referencing. If the index were, say, 720 electrical degrees it would be recorded at two different physical positions four quadrature counts apart. If your system were in a

home sequence, depending upon encoder direction, a different physical zero reference position would be selected. It could ruin your product . . . and your day!

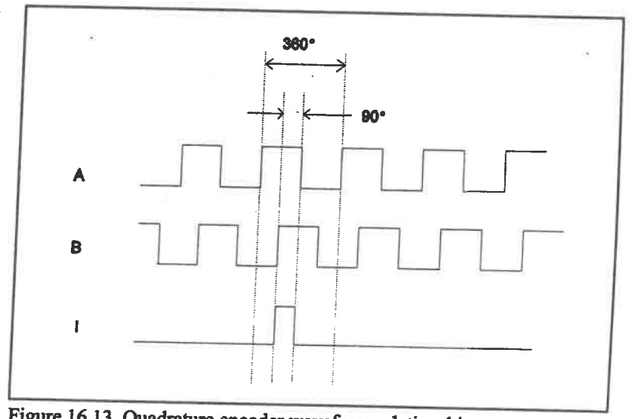

Figure 16.13 Quadrature encoder waveform relationships.

The main drawback to this method is that when using a rotary encoder, the index pulse is repeated every encoder revolution. Therefore, if a complete encoder revolution translated to 0.1 inch of linear travel, and you have a 12-inch system, then 120 index pulses occur as the axis moves toward home position. It is then necessary to somehow jog the axis close to the desired index position and then command the system to zero reference. This can be bothersome, inaccurate, and time consuming with low-encoder count/high-resolution systems.

Method 5: Referencing to a switch, then to the encoder index

This is the preferred method of referencing. By implementing a combination of methods 2 and 4, you gain the best compromise. You can high-speed move to a course zero position via a switch and then slowly move in the proper direction to the actual home position via the index signal.

With any of these methods, once you have zeroed the axis the appropriate home offset value can be inserted. The reason for the home offset value is to allow accurate axis homing whether or not the switch and/or index pulse can be positioned precisely at absolute true zero (refer to **Method 2**, Figure 16.11).

Servo and Stepper Homing Considerations

Knowing the difference between stepper and servo motors can greatly reduce the costs involved when dealing with stepper system homing (see Chapters 6 and 7). A stepper motor is a torque device that maintains a constant phase current requirement whether in motion or otherwise. Therefore, if a stepper motor is driven into a "hard-stop" such as a machine part at the axis end-of-travel, the stepper motor will still draw the same current as during the move. Servo current, on the other hand, will increase as the computer attempts to correct for the following error[3] created.

Because of this difference, it becomes apparent that the servo motor must be controlled at all times via switches, feedback devices, and other components. This is to prevent the motor from burning out or damaging the machine as the motor is moved from one position to another. Steppers, on the other hand, can legitimately though carefully be forced into a hardstop, and the system counters zeroed (or preset to some home offset value).

One caution, however, is that when you drive any motor into a hardstop situation, you must ensure that your system can withstand the resulting shock.

Since the basis for homing is to reference the system counting mechanisms (i.e., CPU, counter(s), etc.) to some predefined axis position, it can be misconstrued that **absolute** position encoders do not require a homing sequence. This will only be the case **if** one revolution of the absolute encoder reflects a **complete system cycle** (such as one revolution of a rotary axis represents one revolution of the absolute encoder). However, if the absolute encoder is required to rotate more than once to produce the total feedback information for a complete axis cycle, then you may be required to do an axis home sequence. Remember that position feedback of an absolute encoder device is only guaranteed to give true position information within **one** physical revolution of that device.

[3] This is the difference between where the computer wants to be, and where the axis actually is.

Following Error . . . Should It Be Eliminated?

Following error can be a very mysterious and elusive motion control factor. Depending on who you talk to, you can be *stuck* with it. It can be good. It can be bad. In this discussion, I'm going to attempt to describe following error and how it can affect your system (if at all). With proper information, only you will determine whether it *should* be eliminated.

This discussion is divided into the following sections, which represent the five regions of a motion profile:

- Profile Start
- Acceleration
- Constant velocity (Slew)
- Deceleration
- Profile Complete

These are the trajectory profile regions that your system attempts to follow. This section discusses how following error affects products being manufactured using coordinated (multiple axis) moves.

Profile START From Zero Velocity

Refer to Figure 16.14. At the beginning of a move (T0) when motion is at zero velocity, and assuming the axis *actual* position is at the controller *desired* position (i.e., zero following error), two phenomena occur. First, a position shift takes place between the start of the computer generated profile and the actual start of the physical move. This is due to the propagation delay of *all* of the motion components of the system. Second, there is an immediate and unstoppable following error generated the instant the computer calculates the very first DAC step output (update algorithm).

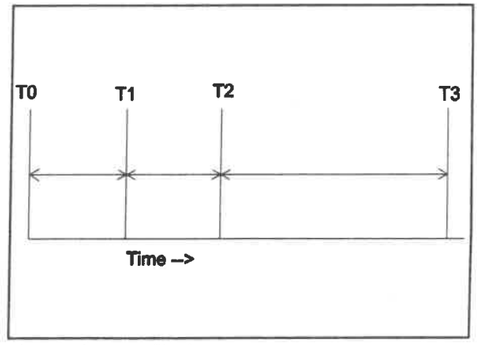

Figure 16.14 Profile time.

Position Shift from Zero

In any controlled motion application, the motion device whether it is electrical, hydraulic, or pneumatic, must lag behind the controller START signal. This is inherent since it is impossible for any motion device to begin a move *prior* to the controller issuing a move signal. The propagation delay for a motion device to actually begin moving once the *Start* command is generated within the computer will yield a mismatch between the *commanded start* ***position*** and the *actual start* ***position***. This is called ***position, or following error.***

Referring to Figure 16.14, assume that an operator presses the START button at time T0. The computer may sense the issuing of the start signal at time T0, or it may take as long as time T1 depending on what point the computer processing is at in the program loop.

Assume that the computer sees the start signal at time T1. Due to program latency, the actual start command sent to the device may not appear until as late as time T2. Also, if the servo update time were, say 500 microseconds, and the start command were issued just after an update occurred, the start may be delayed until the next update period—an additional 500 microseconds.

Once the actual start command is issued to the motor drive amplifier (let's assume at time T2), it can take as long as time T3 for a physical move to actually begin. A delay in starting the actual mechanical motion can be due to motor power building in the motor over time (high L/R time constant), motor cogging, friction, etc. The system mechanical time constant in conjunction with the motor electrical time constant, help retard the motion with respect to the desired generating following error.

Note, that the time from T0 to T1 is variable, but the time from T1 to T2 can be variable or fixed depending on what the program requirement is prior to starting the move. The time from T2 to T3 is fixed and depends entirely on the motor, the amplifier, and the system mechanics.

Beginning at T0, the elapsed time from generation of the commanded start signal within the computing unit, to the point that the actual start command is sent to the amplifier at T2 must be kept to a minimum. This ensures that the computer or controlling device does not become a problem to the required motion timing.

Will the existing command propagation delay (T0 to T2) degrade the performance of the system beyond the required specifications?

Will the actual motion propagation delay (T2 to T3) degrade the performance of the system to a point where the required system specifications cannot be met?

For example, very high speed systems may require a *Slave* axis to accelerate to the speed of a *Master* axis in continuous motion (i.e., conveyor, extrusion . . .). The overall propagation delay from time T0 to T3 will cause the actual starting position of the *Slave* axis to shift several thousandths, or even as much as several tenths of an inch from what is desired.

This shift must be accounted for, when determining system accuracy or repeatability. Position shift will be small at low velocities, large at higher velocities. Consider starting point instability as a minimum tolerance window within which the controller cannot guarantee position. This is true regardless of system operation type.

It is first necessary to determine how much starting delay you can tolerate. Let's assume an extrusion system will operate at 250 fpm. Further assume that the system

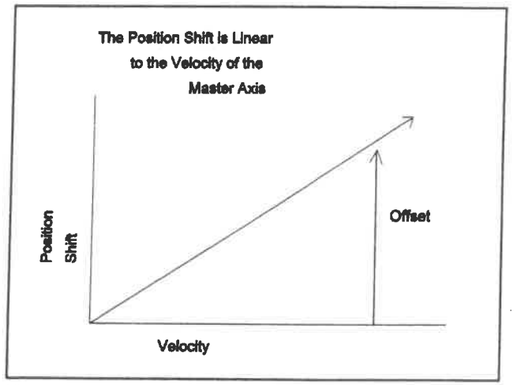

Figure 16.15 Position shift versus velocity.

task is to cut the extruding material at pre-specified marked positions. Obviously shifts in cutter position with respect to the marks on the material beyond a certain range cannot be tolerated. If you simply want to cut the extruded material to a given length, you may disregard the delay from time T0 to T3, provided that the T0 to T3 time window does not fluctuate enough to allow position shifting error outside the specified length tolerance. Therefore, for this operation type you must know the **± Length Tolerance** specification. This specification sets the maximum allowable variance from time T0 to T3.

For critical delays (as in the first example), you may offset the controller position tracking system by the amount of position shift caused by delay at any velocity (Figure 16.15).

Following Error from Zero

To initiate a move, a controller DAC output voltage must be generated. But to generate this, an error between the computer ***desired*** position and the system ***actual*** position **must** exist. It is this error that the computer gain calculations will manipulate to produce the very first DAC step voltage output to start the actual move.

Is it possible to eliminate this starting error?

No!

Many designers believe that they can eliminate starting error by using a form of feedforward gain. But the prime purpose of a feedforward structure is to **reduce** following error *during* a move; it cannot eliminate the error at the start of the move.

On the other hand, *pre-starting* will offset this delay and eliminate the error by putting out a DAC voltage for a prescribed period of time at the exact moment the start is issued (at time T2 in Figure 16.14). But, motor power is built up over **time**—not instantly—and the system reaction is a function of the mechanical time constant of the system. These two facts stop any physical motion at the exact instant the pulsed computer signal is issued.

Therefore, the key might be to issue this pulse **prior** to the true start command in order to offset the mechanical delay. But this too can create problems. Just imagine what could happen if some external operation intervened at the last instant and commanded the unit **not to move** prior to the start and after the prestart. You now have no suitable means by which to stop. Apply a reverse pulse? Now our controlled system is an out-of-control hodgepodge consisting of forward and reverse jerking. This will do *wonders* for position shifting. Furthermore, if the external signal were a **no-go**—it's too late! The unit was told to move too soon, and now the damage is done.

It is also important bear in mind that the computer will (and should) **Lead** the system mechanics from the beginning of the profile. What is key, though, is whether this is detrimental to system operation. In my experience, as long as you plan ahead, 99.999 percent of all systems won't have following error problems.

Following Error during Acceleration

Once the move has started, how you coordinate the *actual* move **acceleration** profile with the *desired* move **acceleration** profile is paramount. A myriad of variables can alter the actual acceleration profile with respect to the desired. Variables may include:.

- lubricants
- friction
- temperature
- weight changes
- gain structures
- DAC linearity
- tool and material interaction
- product variances (i.e., hardness, shapes . . .)
- motor and amplifier types (gains, adjustments, and linearity)
- motor to motor differences of the same type.

These are but a few of the system variables working together to create a following error situation.

Your greatest concern at this point, is to find out whether the motor system requires a linear response. Refer to Figure 16.16. Boxes A and B show areas in the acceleration profile where significant non-linearities may occur. Any non-linearities in the motor or amplifier package will guarantee a deviation from the desired following error.

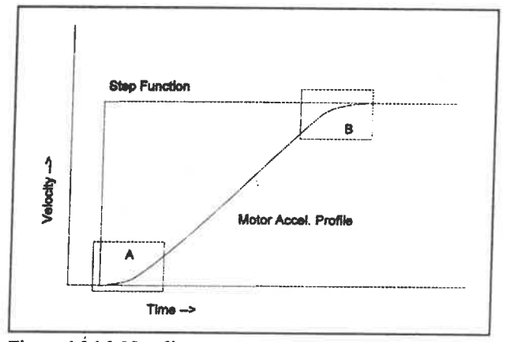

Figure 16.16 Non-linear step response.

Some questions to ask about following error are:

What following error is being generated—*instantaneously* due to disturbances, or *constantly* due to non-linear regions in the profile?

Is following error relevant to system operation in terms of accuracy, repeatability, or stability?

Can following error be eliminated?

Can following error be reduced?

It should be clear that non-linear areas of the motor acceleration curve shown in Figure 16.16 can prevent a digital system from achieving a perfect *real-time* position match. The best that you can hope for, within reasonable constraints, is to start the motor prior to the motion profile command or simply allow the linear portion of the acceleration slope to lie as close as possible to the desired slope (refer to

Figure 16.17). Of course, this assumes that we allow the motor to accelerate under its own control and only allow the CPU to track what is actually happening rather than control what the motor does. In reality, this is not very practical.

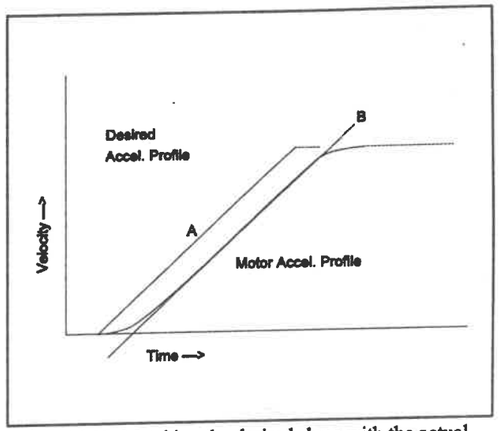

Figure 16.17 Matching the desired slope with the actual slope.

If we cannot eliminate following error, the least we can do is reduce it. The first problem to resolve is whether the control can **react** to sudden changes, and whether the control has the ability to cope with continuously changing positions due to non-linearities in the system itself (motor, motor amplifier, springs, pressures, . . .). Your main concern is to keep the following error as small as possible without allowing the mechanics to *lead* the control.

Your next concern is whether following error will be detrimental to the overall system operation, and how much would be considered acceptable, if so.

Differential gain (Kd) in the PID structure generally reacts to position changes as a function of the time rate of change. The general formula for this is:

DAC Change - (*Kd*) (*NewFolErr* - *LastFolErr*)

The following error (position error) calculated in each successive update period is compared to the following error calculated previously, and the controlling DAC voltage is adjusted based on the difference between the two. This, however, is only one method of maintaining a reasonably instant reaction to an instant position change. A second method is to apply a form of **closed-loop** feedforward gain control as show in Figure 16.18.

The PID gain structure handles the errors generated from *actual* **and** *desired* position mismatches. In addition, velocity feedforward, acceleration feedforward, and a setpoint can also be used in the operation.

The velocity part of the trajectory algorithm (Tvel—see Figure 16.18) is used to override the PID loop improving motion stability at very low velocities and during high velocity short move indexes. The acceleration part of the trajectory algorithm (Tacl—see Figure 16.18) can override the PID loop to help maintain closer correlation of *actual* to *desired* acceleration (provided the motor can supply it). You can think of the setpoint as the *pulse* that was described in Section 1 of this discussion.

As with any error handler, if the velocity and/or acceleration of the **actual** motion is used in the gain calculations, the stability of the profile will be improved. If the gain algorithm does not use feedback (i.e., open loop), then there is no guarantee that it will either eliminate or reduce following error. It is further possible that if your algorithm operates with no feedback, master/slave systems such as

flying cutoffs or high response systems may wind up with worse position variations than had feedforward terms not been used at all!

The important part of this discussion is to place the emphasis on the motor package. In a system required to accelerate at 1000 rev/sec^2, you would not think of using a motor that could only accelerate at a rate of 500 rev/sec^2. If you did, the motor would **definitely** lag behind the desired position, and it would generate significant following error.

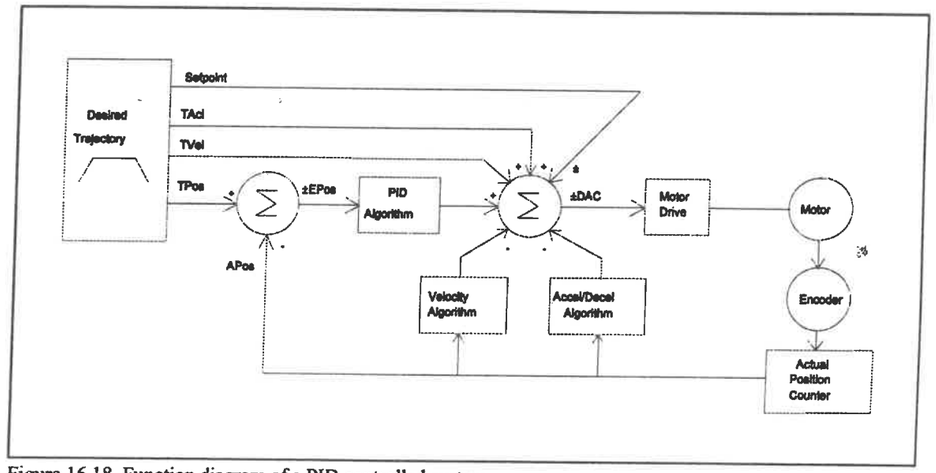

Figure 16.18 Function diagram of a PID controlled system.

Would you then want to overspeed in order to catch up to the desired position point?

Not necessarily!

As long as your system is designed to account for mechanical requirements along with the motor and motor amplifier needs, all of the *real-time* operating considerations and problem areas the system will face will be maintained within the required customer specifications.

Regardless of the system type, as long as following error is stable and repeatable, the job can be done with precision.

Following Error during Slew (Constant Velocity)

Once the desired acceleration has reached running velocity, if the gain algorithm is properly set, will permit the motion device to either maintain or catch up to desired position. It is at this point that many users feel they *must* have **zero** following error in order to achieve proper system performance.

Let's look at some examples to find out if this is true:

Example #1: Match Speed and Print Application

System specifications:

- Max speed: 4000 inches per minute
- 4-inch registration marks
- 1 print head on a 2.25 inch diameter wheel
- Print 2 inches from any mark
- ±.010 inch position tolerance
- Registration marks at random positions but not less than 5 inches apart

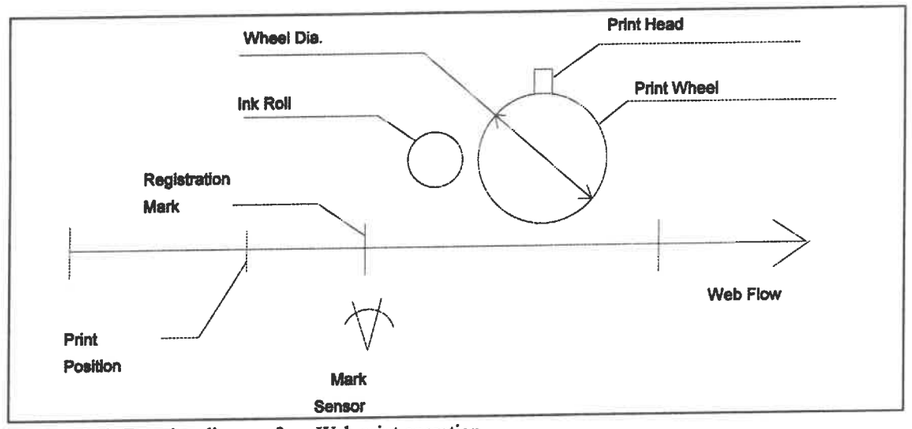

Figure 16.19 Function diagram for a Web print operation.

Our task here is to print information on a continuous roll of material called a *Web*. We want to print at predefined positions on the Web indicated by a series of marks called *registration* marks. We can adjust the printing position with respect to the position of the marks. This operation is shown in Figure 16.19.

Will the system perform better **with** or **without** following error?

Figure 16.20 shows the desired profile timing diagram. When the mark is sensed, the CPU must calculate not only the start moment for the printing operation but a Return-to-Home profile to ensure that the print head is at home position before the start of the next cycle. Note that two acceleration profiles are shown in the figure. If the move is to be initiated when the mark is sensed, then the acceleration is set to a slope that will allow the print wheel to achieve a mainline velocity at the instant the print head touches the web. However, if the acceleration is to be **fixed** at the best case for the print wheel motor, then the controller must calculate a start position **offset** to properly place the printing on the web.

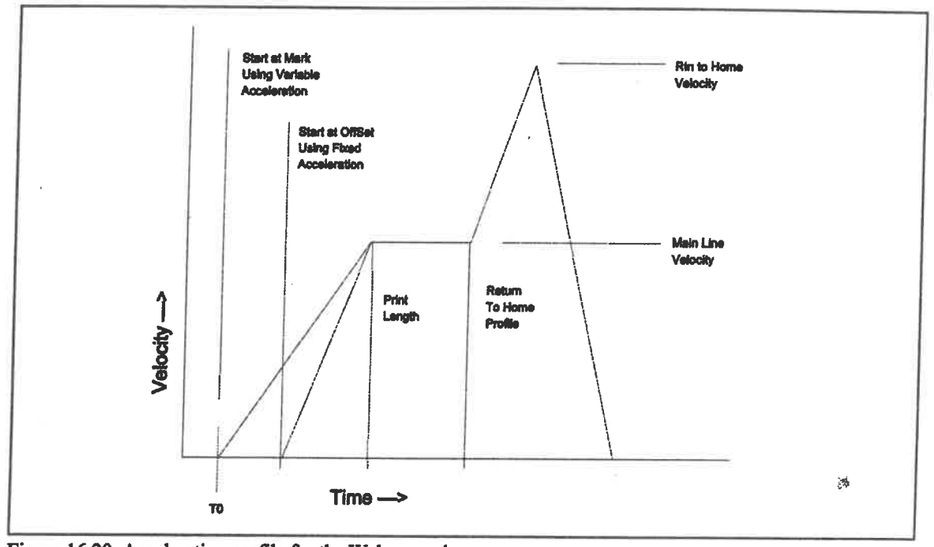

Figure 16.20 Acceleration profile for the Web operation.

If we overlay the profile shown in Figure 16.20 on the actual move profiles shown in Figure 16.21, we notice that the actual profiles are not trapezoidal at all but rather an ***S*** curve. This is so, even though the entire sequence is created by a *trapezoidal* profile generator. We can determine this by simply running the motor and viewing the profiles on an oscilloscope. Also, two different acceleration rates are used (on the same system) to find out how they affect the specification.

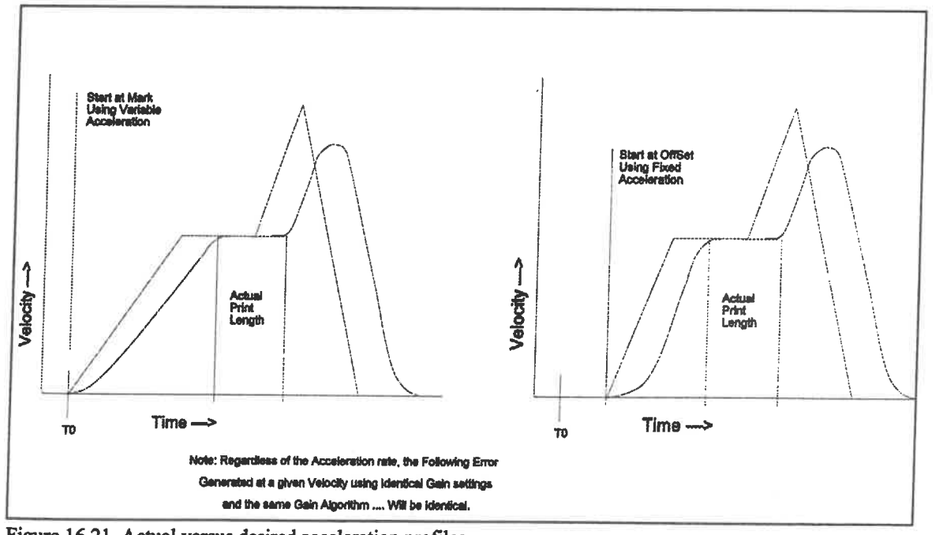

Figure 16.21 Actual versus desired acceleration profiles.

The result is as expected; the acceleration torque required to produce the move increases, but the position shift at any given velocity remains the same—provided that you do not alter the gain settings.

Testing the print position at different speeds using 10-inch marks and a simple *trapezoidal* profile, the offset created by the following error causes the actual print position to shift linearly.

This means two things:

1) Since following error is reasonably linear, multiplying the required print position on the web by a following error (shift) factor allows the print position to remain stable even though we print at different velocities.

2) At 16.667 prints per second[4], the time for one print cycle is 0.06 seconds or 0.015 seconds per inch. Assuming you want to *pull out* the following error, and it takes 0.005 seconds to do this, the web will have traveled 0.333 inches. Therefore, the print position must be offset by 0.333 inches at maximum velocity.

As you can see, calculating an offset at different velocities to accommodate following error must be done, regardless of method. So, what is the advantage of zero following error in this application?

None!

Example #2: Linear and Circular Interpolation

System specifications:

- Max speed to 400 inches per minute
- Two axes

This example is a little more interesting. To interpolate between two axes, it is necessary to understand what is happening as motion takes place and to understand the motion involved.

Linear Interpolation

With linear interpolation the objective is to produce a straight line **vector** motion. This means coordinating actual positions at any time in the move. It would be impractical to allow the gain structures of two different axes to override what the control is attempting to do, and still maintain straight vectors. As the axes accelerate and subsequently maintain slew velocity, any variation in velocity of one axis must be accompanied by an adjustment of the velocity in the other axis.

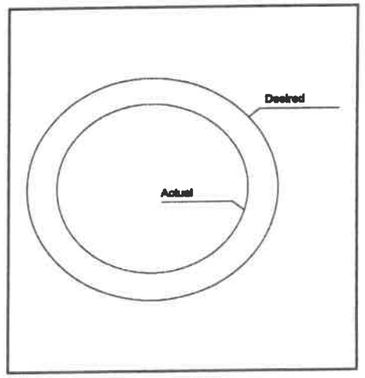

Figure 16.22 Effect of following error on circular interpolation.

[4] (4000 inches per minute)/(4 inches)/(60 seconds) = 16.667

Circular Interpolation

In circular interpolation, most (if not all) of the motion is done during the acceleration and deceleration profile regions. Slew velocity is never achieved. When handled properly, allowing following error to exist in this type of interpolation simply results in a smaller than desired circle (see Figure 16.22).

All circles can be described as polygons constructed with very small straight-line segments. In reality, though, circles are created by a finite number of straight line moves, the type of which are illustrated in Figure 16.23. Diameter tolerance and chordal error determine the velocity that can be achieved during a coordinated motion.

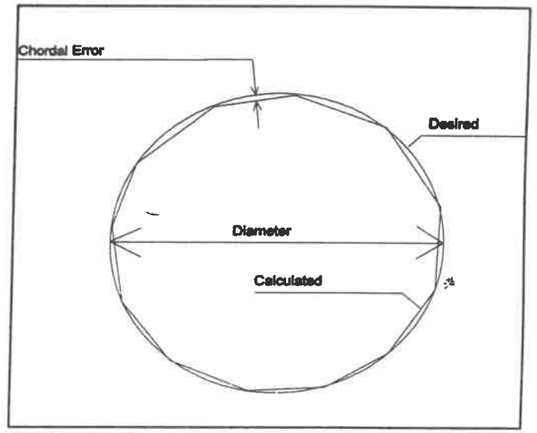

Figure 16.23 Constructing a circle using circular interpolation.

For either type of interpolated motion you won't have to worry about following error. If handled properly, it either (1) does not exist, (2) is too small to be of any concern, or (3) is self-canceling between axes.

Although you can disregard following error, this is not to say that the circle will be of proper size. As following error grows, the constructed circle will shrink, but if following error is handled properly, the circular motion will produce **round** circles. If error is handled incorrectly or if error differs between axes, you will produce shapes more skewed in nature.

Example #3: Cut to Length Application

System specifications:

- Max speed to 250 feet per minute
- ±.005 inch length tolerance
- 20-inch carrier travel

As in the Web example, whether printing to a specified position or cutting to a required length, you need not consider following error a factor in machine operation. Depending on product tolerances and allowable product loss, the difference in a cut-to-length operation is that **no** following error adjustment has to be made for changes in velocity.

For example, in the case of an extrusion, the tolerance might be ±1/16 inch. If the mainline velocity changes are made in 5 to 10 percent increments over several parts, there could be no out-of-tolerance lengths cut. If the mainline velocity in the length of a single part is changed by 50 percent, that part could possibly be cut out of specification, but the remaining lengths would be within specification. The reason for this is that once the new mainline velocity is achieved, all future *Slave* axis (cutter carriage) following errors will be identical; therefore, all future cut lengths will be to dimension.

The objective at slew velocity is to achieve stability as fast as possible, not zero following error.

Following Error during Deceleration

If following error (1) inherently **lags** at the *start* of a profile, (2) tends to **lag** the profile during *acceleration*, and (3) is allowed to maintain a constant **lag** when it achieves *slew* velocity (in order to achieve the fastest slew stability), it only makes sense that following error will lag at the start of deceleration for the same reasons as for acceleration following error.

Figure 16.24 shows a triangular profile and two variations of possible gain settings for a PID algorithm (see Figure 16.18).

Note that following error has shifted polarity on the center graph but has remained negative throughout the entire move on the lower graph. The reason for this is the actual profile is tracking the desired at a higher or *tighter* gain setting. When the desired profile begins decelerating, the actual system position is forced to *overshoot* and maintain a position **ahead** of the desired. In order to maintain a **lagging** following error, the desired position must always **lead** the system actual position. This is shown in the lower graph.

In other words, if you require a steep deceleration profile and the following error is zero or close to zero, there is a greater the chance for system oscillation or overshoot during velocity changes. Lower gain settings will eliminate this tendency and stabilize overall system operation.

For high-speed indexing applications, you must decide whether overshoot is acceptable and by how much. The greater the system lag, the lower the chance of overshoot and the slower the response. However, this does not imply that the response will be **significantly** slower, just that it *is* slower. The overshoot when the trajectory generator stops depends on the gain, the gain algorithm, the motor system power, and the system stability.

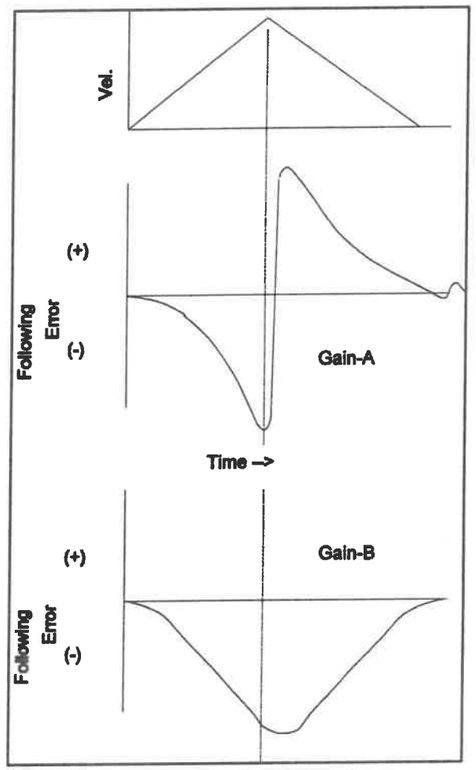

Figure 16.24 Triangular acceleration profile.

Following Error at the Stopping position

Internally, the CPU has now completed the move . . .

Is the actual move done?

If not, how long before the move is done?

Has the system stopped in an acceptable manner?

You must remember that **all** built-up following error **will**, in fact, be removed prior to physically stopping. Excessive following error can be observed by simply *jogging* an axis. When you press the jog button, you expect the motor to begin moving. Similarly, when you release the jog button, you expect the motor to stop immediately, or you expect it to stop within the deceleration guidelines. Very long deceleration times will cause the motor to wind down, but excessive following error will *delay* the start of the deceleration profile. In some instances, following error is so severe that it causes the system to appear as though no deceleration is going to take place at all!

Figure 16.24 shows how different amounts of following error can react differently in the system. Following error cases are commonly referred to as **Large FE** (*softly* tuned) , **Moderate FE** (*properly* tuned), and **Small or Zero FE** (*tightly* tuned). This is reasonable since following error is a direct result of gain algorithm parameter settings. Therefore, *size* and *tuning* are synonymous. No **one** FE case implies that another case is **wrong**; they merely differentiate between each other. It is strictly up to you to apply the proper one for your system.

Some parameters to consider are:

1) the amount of acceptable oscillation while the axis is stopping,
2) the maximum time within which the axis **must** be absolutely stationary (as in perhaps a vision system), and
3) the amount of actual error existing before an axis is considered stopped.

As with acceleration, the system-loading, motor package capability, profile to be operated, etcetera, will all determine the best system deceleration rate. Exceeding this at the starting point will cause a shift in the actual position with respect to the desired. Exceed this during deceleration, and the system will be unstable at the stopping point.

Discussion

When developing a system, it is necessary to understand, to some degree, the impact that following error will have on system performance. Understanding what following error is and how it will work with your system is made easier when the motion profile is divided into its five constituent regions. With the veil of mystery removed, the proper controlling technique will emerge.