Chapter 12

Dissecting a Move Profile

Features

Introduction

During one of my recent tutorials, I spent considerable time in the analysis of the well known, but often times misunderstood, ***S*** curve. The purpose of this chapter is to show how trapezoidal and ***S*** profiles work and their relationship to motion control and product handling. Sample algorithms are provided for the ***S*** curve and general trapezoidal profiles in Chapter 10.

There are three things that occur when a motion profile is initiated . . .

1) **The calculation of the desired profile which is done internal to the CPU in the controller.** This is an ideal profile generated to guide the system along what would be considered the *perfect* or *ideal* trajectory, but it only exists inside the computer.

2) **The DAC output signal used to control the motor via the motor amplifier.** The DAC output, controlled by some form of error generating (gain) mechanism, physically attempts to maintain coordination between the *desired* trajectory and the *actual* move profiles.

3) **System Motion.** All system components (i.e., mechanics, load, friction, environment, . . .) must be maintained within the original design parameters to ensure proper operation. The system should be engineered for the least possible mechanical error between the required and actual response to deliver the smoothest possible motion.

The Desired Profile

To move anything controlled by a computer, there are two basic approaches:

1) **Model the move, and then develop a control signal to emulate the model.** In other words, allow the computer to do the move mathematically, and by use of some form of feedback monitoring, develop a difference signal between the desired and the actual profiles to physically accomplish the required move.

2) **Simply control the output signal directly via a model (as is done with stepper motor motion), and simply assume that the physical move has occurred.**

In either case, the computer must generate a mathematical model of the required move. I will refer to the trajectory acceleration (or deceleration) as a profile rather than a slope. The usage of the word *slope* is to infer a straight line path between velocity changes that does not necessarily conform to the DAC output signal. This is because DAC output changes are voltage steps, not smoothly transitioning analog signals. Also, the resultant DAC output steps may vary from desired trajectory

values due to changes in mechanics (i.e., load fluctuations, resonant points, . .) and the error correcting algorithm.

Two types of profiles to be discussed are:

- Trapezoidal Velocity profile
- **S** Acceleration profile

Please note that throughout this chapter, the synonymous terms: *update time, update period, update, sample time, sample period,* and *sample* will be used interchangeably, and will refer to the cyclic time period in which the controller will recalculate (update) the desired, actual, error, and DAC information.

The DAC Step and Acceleration Control

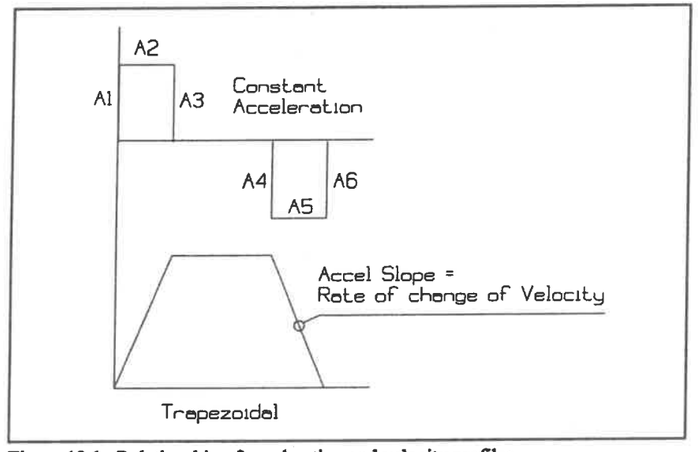

Figure 12.1 Relationship of acceleration and velocity profiles.

It is important to restate, that the slope of any calculated move profile (moves generated by computers) occurs in voltage steps, and because of this, ***all*** changes in the calculated acceleration/deceleration, velocity, position and DAC output are **step functions.**

Figure 12.1 shows the relationship between acceleration and velocity in the trapezoidal profile. The acceleration curve is represented by an instantaneous jump (**A1**) to the programmed acceleration rate (**A2**), which remains at a steady state value while accelerating to the desired velocity, and an instantaneous jump (**A3**) to zero when the required velocity has been attained. Steps (**A4**), (**A5**), and (**A6**) reflect deceleration in a subtraction mode from the given velocity. At the instant the acceleration transitions from zero to the required value, a surge (or impulse force) will be exerted on various system components as well as the load. This impulse force can be estimated using simple calculations as seen in the following example:

Imagine a five-pound steel cube sitting on a steel plate (see Figure 12.2). Assume the friction coefficient between the cube and the plate is 0.1, and that the system must attain a velocity of 100 ft/min.

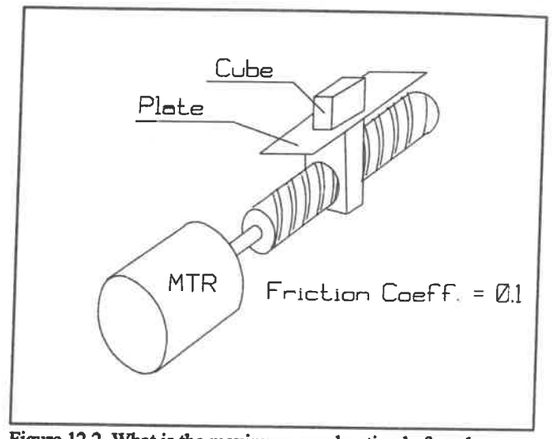

Figure 12.2 What is the maximum acceleration before the product will slip?

The questions to answer are: (see Figure 12.3)

1) What is the maximum acceleration (in ft/sec²) that can be applied to the system without allowing the cube to slip on the plate?

2) What is the fastest possible time (in seconds) that the system can achieve the 100 ft/min velocity without allowing the cube to slip on the plate?

Newton's Laws of Motion state . . .

Bodies not in motion are in equilibrium; and when at equilibrium, action and reaction are **equal** and **opposite.**

What this means, is that a slight disturbance to the equilibrium of the cube will cause it to move.

Based on the weight of the cube and the coefficient of friction between the cube and the plate, it will take an exerted force of just over half a pound (5 x 0.1) to cause the cube to slide on the plate.

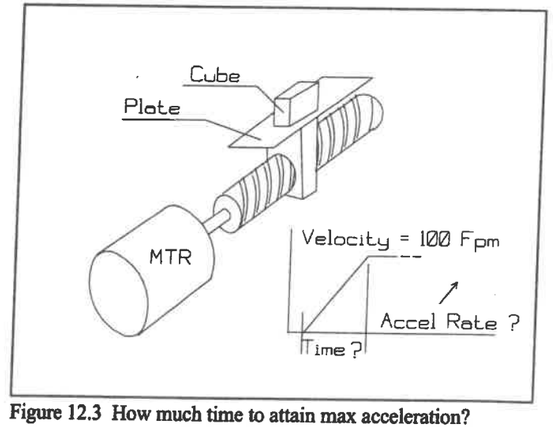

Figure 12.3 How much time to attain max acceleration?

For a horizontal body:

$$Friction\ Force\ (F_f) = (Mass)\ [Acceleration\ (A)\ ]$$

or:

$$F_f = \left(\frac{Weight}{32.2}\right)(A)$$

and:

$$(5)\ (0.1) = \left(\frac{5}{32.2}\right)(A)$$

Therefore, the system maximum acceleration rate is ...

$$A = \frac{(0.5)\ (32.2)}{5} = 3.22\ Ft/Sec^2$$

The fastest possible time to reach 100 ft/min will be ...

$$Velocity\ (Ft/Sec) = (A)\ [Time\ (T)\ ]$$

and:

$$\frac{100}{60} = (3.22)\ (T)$$

and:

$$T = \frac{\frac{100}{60}}{3.22} = 0.517598\ Sec.$$

Realize that there are two forces at work in this example (see Figure 12.4):

1) The frictional force (F_f) which tries to keep the cube in its original position on the plate.

2) The inertial force exerted by the cube (F_{jc}) which tries to maintain the cube's attitude in "space" (in other words, keep it stationary).

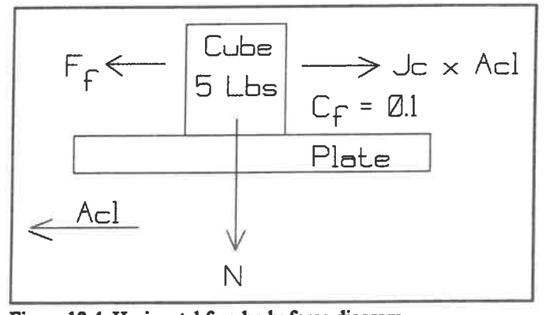

Figure 12.4 Horizontal free-body force diagram.

For the horizontal system in our example, the force due to friction is simply the weight of the cube multiplied by the coefficient of friction [(5) (0.1) = 0.5 Lbs.]. So, when a horizontal force

For the horizontal system in our example, the force due to friction is simply the weight of the cube multiplied by the coefficient of friction [(5) (0.1) = 0.5 Lbs.]. So, when a horizontal force slightly greater than 0.5 lbs is applied to the cube, the cube moves. The horizontal force that causes the cube to move is a result of the inertia of the cube multiplied by the acceleration rate of the plate (similar to pulling a table cloth out from under a set of dishes).

The problem is not the ability to calculate or respond to a profile requirement fast enough (i.e., DSP processors . . .), but rather the applied force exerted by the ***first*** DAC step (or steps, as required) in the motion trajectory.

In our example, the time required for the mechanics to go from zero velocity to the first DAC step velocity is calculated like this:

$$Maximum\ System\ Acceleration = 3.22\ feet\ per\ second^2$$

Assume a voltage drive (to easily ratio the velocity steps) and a 12 bit DAC ...

$$1\ DAC\ Step = \frac{100}{2048} = 0.048828125\ Ft/Min$$

For the ***system*** to stay within the 3.22 ft/sec^2 acceleration rate it must not experience a DAC step change in under ...

$$Time\ (T) = \frac{Velocity\ (Ft/Sec)}{Acceleration\ (Ft/Sec^2)}$$

$$T = \frac{(0.048828125\ /\ 60)}{3.22} = 0.000252733\ sec.$$

The one DAC step change must occur in under 253-microseconds calculated above in order to approximate the required maximum acceleration slope of the system. However, it is important to understand, that the DAC output must be held at this level for a longer period of time to prevent the motor from applying an impulse force greater than 0.5 lbs. to the system mechanics (see Figures 12.5, and 12.6). If the controller's error generator does not take into account the ***DAC step rate of change*** and the ***motor's electrical time constant***, then the system could exceed the acceleration requirement regardless of whether the trajectory generator is operating

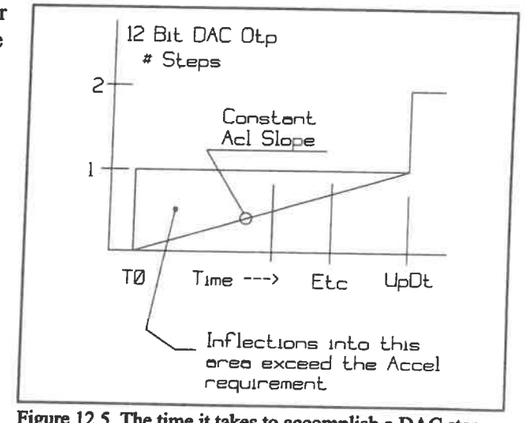

Figure 12.5 The time it takes to accomplish a DAC step.

in perfect harmony with desired time and position calculations or being calculated by trapezoidal or **S** curve algorithms.

The electrical time constant of the motor (in this example) must not be less than 2/3 of the time calculated above or 168 microseconds. This will ensure the proper buildup of *power-over-time* in the motor. The DAC output in this case must be maintained for a minimum of 840 microseconds (five times the motor electrical time constant (168 microseconds in this example)). The motor will respond to changes in the DAC output voltage in the same manner as an RL time constant responds to a pulse waveform, and it is calculated as follows:

$$\textit{Motor Electrical Time Constant (Sec)} = \frac{\textit{Motor Inductance (H)}}{\textit{Motor Resistance (Ohms)}}$$

Note also, that the acceleration calculated previously is not relevant to **any** motion control computer; the only relevant consideration is the size of a DAC change, or number of DAC steps generated as the **motion** is done in relation to the top velocity of the system being calculated.

The motor electrical time constant will allow the buildup of the motor's velocity in increments of 63% of the difference between the DAC output *velocity* and the actual motor velocity each update period. If the motor velocity is allowed to follow DAC step changes (as a step function), then the power transferred from the motor to the load will appear as an infinite impulse force to the load (sometimes referred to as *jerk*), since motor power delivered is a function of the time-rate-of-change of current in the motor.

The key to ensuring that the DAC influence on the acceleration stays within the system requirement is one of the following:

1) Use short update periods to effectively modulate the DAC interpolating a *straight line* voltage gradient of the proper slope.

2) Select a motor with the appropriate electrical time constant coordinated with **DAC output changes** and **DAC output hold time.**

3) Use a load-coupling or motor amplifier with acceleration/deceleration analog profiling control to reduce any impulse force that would be presented to the load.

4) Use a higher resolution DAC for improved voltage control and a reduction of each DAC step impulse versus update (see Figures 12.6, and 12.10).

5) Ensure that the gain structure takes into account the DAC output time-rate-of-change in order to prevent the DAC from over-torquing the system.

6) Use a smart drive with analog ramping control to disallow the motor's ability to exceed the required profile.

Figure 12.6 shows the relationship between DAC step changes and the motors electrical time constant. Segment (A) is a single DAC step change, and segment (B) is a two DAC step change. Segment (C) describes a curve produced by a system with an electrical time constant equal to 1/5 of the update time period. Note, that the maximum slope of curve (C) exceeds the maximum allowed slope (dashed line). Segments (D) through (G) show the response curve for a system with an electrical time constant equal to 2/3 of the update time period.

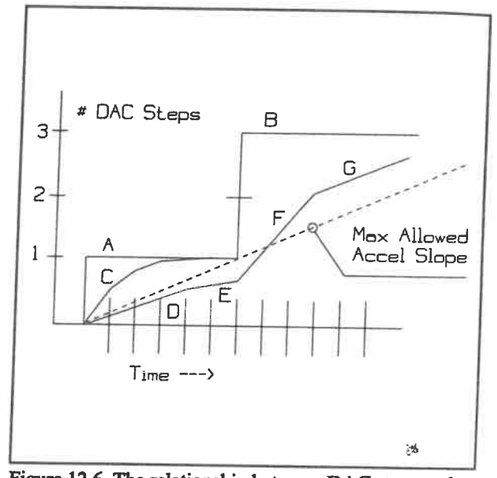

Figure 12.6 The relationship between DAC steps and electrical time constant.

The acceleration slope of curve (D) to (G) can ensure an acceleration value within what is required, **if** the DAC output is held at a given level for a long enough period of time—allowing the system to "catch up". Notice that the slope of segment (F) has exceeded the maximum allowed slope (dashed line) because of the two step DAC change. What this means is that a motion controller may at any time exceed the required acceleration rate due to *real-time* position error, gain structure, update time and/or fluctuating system mechanics.

The update time and DAC resolution can only reduce the impulse force presented to the load if the system time constants have been considered. DACs with finer bit resolutions (see Figure 12.10) and faster updates will reduce the impulse force by better approximation of the true analog acceleration slope. But these devices **must** maintain their fast update period, or slower operation will result. This is because the longer the controller is away from the action, the larger the positional error and the larger the DAC output will become to offset the error. Then, the resultant impulse force on the system will increase. Finally, longer motor electrical time constants will require longer DAC hold times to maintain the low impulse on *takeoff*; but again, they **must** be properly coordinated with all of the system components (requirements) to give the desired performance.

An algorithm for the DAC clamp and DAC hold time requirements solving the problem just described can be found at the end of this chapter.

Trapezoidal Profile Consideration

The trapezoidal velocity profile is generated within the processor by the following progression formulas:

Accel. = User Entered Value
New Vel. = Last Vel. + Accel.
New Pos. = Last Pos. + New Vel.

where:

Velocity is calculated in counts-per-sample period (calculated every sample period).

Accel. is the *desired* acceleration in counts-per-sample-per-sample.

Position is the *desired* position in encoder counts used to develop the DAC control signal.

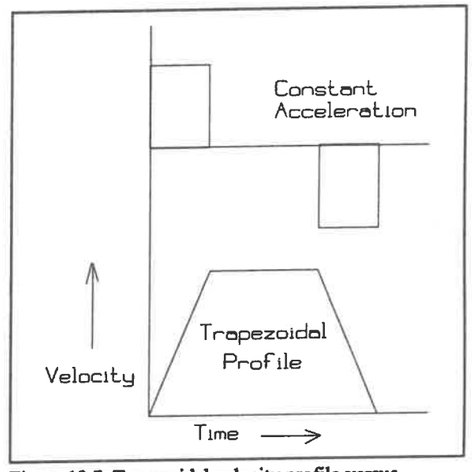

Figure 12.7 Trapezoidal velocity profile versus acceleration.

Since the math is calculated every update period, acceleration is automatically converted to velocity as the math is done. The actual trapezoidal profile is shown in Figure 12.7.

Gain Consideration

At the end of the trajectory calculation, the error generator then calculates the difference between the *desired* trajectory position, and the ***actual*** system position (encoder feedback). For an in-depth discussion of the PID, refer to Chapter 11.

The result of this calculation is applied to the DAC output as follows (assuming a 12 bit DAC) . . .

$$DAC = \frac{(K_P)\,(E) + \left(\frac{(K_I)\,(E_T)}{256}\right) + [(K_D)\,(E - E_L)]}{16}$$

where:

DAC = Actual number of DAC steps outputted
K_P = Proportional Gain
K_I = Integral Gain
K_D = Differential Gain

E = Desired Position - Actual Position
EL = Last E
E_T = Total E since power was turned "on", accumulated every update period.

Note, that depending upon the gain *values* implemented and the actual system position at any time, the DAC can produce wide output swings **instantly** within its step range. For infinitely smooth motion, a PWM DAC output is **not** recommended due to its *course* resolution.

S Curve Profile Consideration

The well known ***S*** acceleration curve can be developed by applying a sine-based formula to the velocity value in order to slowly increase the rate of change of the DAC velocity command over time, or by a linear increase of the acceleration over time as shown in Figure 12.8. The time period used to calculate the new acceleration rate can range from microseconds to seconds depending upon the requirement of the system.

An ***S*** curve algorithm is given in chapter 10. Notice that the ***S*** acceleration shifts can be calculated in different time intervals than the trajectory update.

The *trapezoidal* calculation uses the acceleration to calculate the velocity and then the position. The ***S*** *profile*, on the other hand, uses time to calculate a new acceleration rate, then the velocity, and then the position. It really doesn't matter how you get there as long as you do.

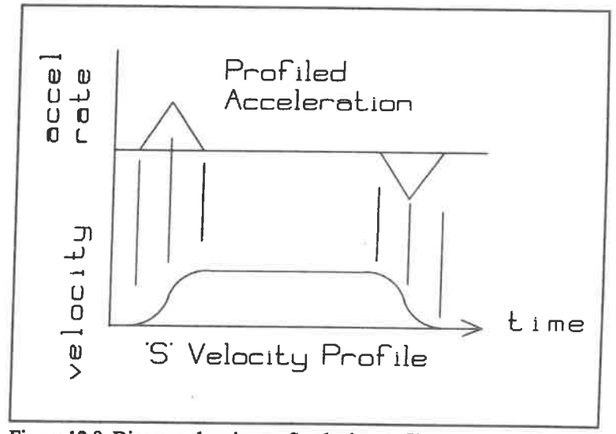

Figure 12.8 Diagram showing an S velocity profile versus acceleration over time.

In any case, the ***S*** profile must still deliver the result to a DAC (via the error generator), and in the end the same physical problems that plague any profile generator (not directly in control of the DAC) will exist here as well. The first DAC step impulse, the maximum allowable acceleration slope, and the end of the acceleration impulse will still have to be dealt with to insure smooth system operation.

Velocity Profiling
Using S or Cubic Math and the National LM628

S curve development can be done by velocity changes that are incorporated into the motion profile on *time* intervals not necessarily associated with the update period. This velocity approach can also be used with the National LM628/29 series of motion control chips. A guideline I use, is not to *talk* to the National chip more often than once every three to four update periods (750 microseconds, minimum).

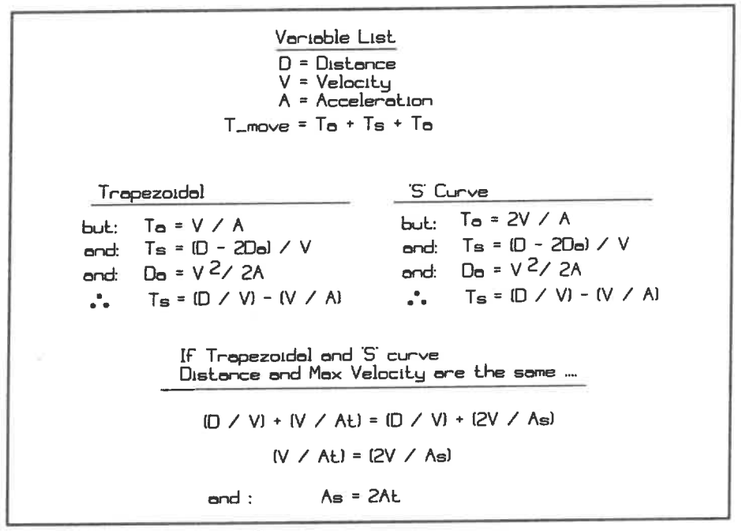

Figure 12.9 Trapezoidal and **S** profile calculations.

Discussion

Things to consider when developing a system are:

1) If acceleration is the limiting factor in the profile, a *trapezoidal* trajectory will achieve the required velocity twice as fast as the **S** curve (see the equations in Figure 12.9). In other words, to move the same distance in the same amount of time at the same maximum velocity, the trapezoidal profile will accelerate at **1/2** the **peak** rate of the **S** curve profile. This yields lower trapezoidal torques, which translates into smaller motors for the same given job (time versus throughput).

2) If an ***S curve*** is required, make sure a single DAC step value is **time coordinated** with the motor's electrical and mechanical time constants to prevent exceeding the maximum

designated acceleration. If this is not possible, make certain the gain structure takes into account the *DAC output time-rate-of-change limit* to prevent the DAC from over-torquing the system, or use a *smart* drive with ramp control.

3) If a 16-bit DAC is used (see Figures 12.6, and 12.10) with a short update time, make sure the update time does not accumulate as more axes are added. If it does, the advantage gained by the 16-bit DAC may be destroyed, since the longer the control is away from the action the higher the error; and so, the slower the system must run to achieve the originally designed result.

4) The electrical and mechanical time constants of the motor and system will have a definite bearing on the outcome of the profile. If a high degree of profile control is necessary and the system response is high, a 16-bit DAC with low update times or direct DAC control may be required.

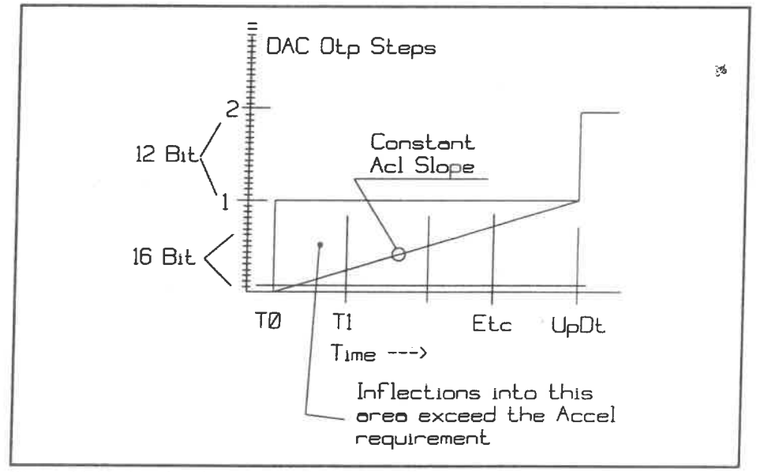

Figure 12.10 Acceleration slope with respect to DAC steps.

Figure 12.10 reflects better control of the acceleration slope with a 16-bit DAC than with a 12-bit DAC; but using correct system time constants along with properly developed profiles can usually do the same task for less cost.

Also, the initial impulse force (jerk) applied to the system mechanics can be reduced significantly with tight mechanics, low system backlash, low friction forces, and/or the use of *smart* drives with ramp control.

In any event, take the time to properly evaluate your *system needs* before jumping into a costly situation that may turn out not to give you the operation you were expecting. You will generally find that lost effort is more expensive than a qualified design.

Following, is an algorithm that controls both the desired and **actual** system trajectories.

Direct DAC Override Algorithm

Sometimes direct DAC control is required, especially if absolute system (i.e., mechanical) acceleration control is required. This is because without direct DAC control, surges in the mechanics due to position recovery by gain algorithms (even in *velocity* modes) can cause system G forces to exceed maximum required limits.

The following algorithm modifies the DAC based on both **time** and **step** increment limitations after the trajectory calculations have been made and gain algorithms have been applied. Although the software is shown in C code, the *method* rather than the language is the consideration.

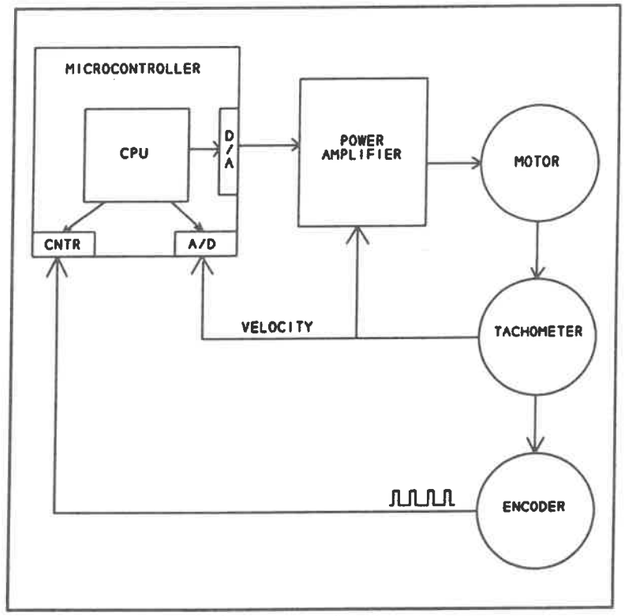

Figure 12.11 A typical microprocessor-based motion system.

DAC Control Algorithm Written in C Language

```
/*      If the Motor is slowing down or speeding up due to external forces (ie., forces not
        caused by the CPU's motion generator), or cannot keep up with the trajectory
        generator because of restrictions placed on it's accel/decel capability, the DAC
        output is adjusted to stay within the limits of the accel/decel parameters.  In
        other words, as the trajectory generator calculates or models the profile in
        successive update periods, the DAC output is modified as necessary to keep the
        system mechanics within the imposed accel/decel limitations. */


                /**********      Do a Trajectory Calculation     **********/
UpDtCnt++;                                              /* number of elapsed updates */
LstVel = Vel;                                           /* last update velocity */
LstPos = Pos;                                           /* last update position */
ActVEL = read_A_D();                                    /* get the actual velocity */

            /***** check to see if the deceleration point has occurred *****/
if (UpDtCnt < ReqCnt)
        {       if ((((Pos + Vel) > Distance / 2) && (Vel < MxVel))
                        || ((Pos + Vel) > DeclPos))     /* test if deceleration is required */
                          {       Vel = Vel - Accel;    /* lower the velocity */
                                  if (Vel < 0)  Vel = 0;/* never lower than zero */
                                  UpDtCnt = 0;          /* restart the time counter */
                                  Pos = Pos + Vel;      /* get new pos at the lowered vel */
                                  Decl_Flg = 1;         /* only once thru this routine */
                                  goto Calc_DAC;        /* Get the new DAC value */
                          }
                else    {       Pos = Pos + Vel;      /* get the new pos at the current vel
                                */
                                goto Traj_ext;        /* exit this routine */
                        }
        }
            /***** check to see if the deceleration point has occurred *****/
if (Decl_Flg == 0)
        {       if ((((Pos + Vel) > Distance / 2) && (Vel < MxVel))
                        || ((Pos + Vel) > DeclPos))     /* test if deceleration is required
                */
                        {  Decl_Flg = 1;  }             /* only once thru this routine */
        }
             /*****      Do a trapezoidal trajectory calculation     *****/
if (Decl_Flg == 0)
        {       Vel = Vel + Accel;                      /* Incre vel if accelerating */
                if (Vel > MxVel)        Vel = MxVel;    /* never greater than allowed max */
        }
else    {       Vel = Vel - Accel;                      /* Decre vel if decelerating */
                if (Vel < 0)            Vel = 0;        /* never lower than zero */
        }
Pos = Pos + Vel;                                        /* get the new position */
                    /*****      calculate the NewDac value      *****/
Calc_DAC:
Err = Pos - ActPos;                                     /* Get the current error */

  /***** Get the NEW Trajectory Generator DAC Step Output using a PID gain structure
*****/

/* 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;
else if (Integ_Scale < -Integ_Lim) Integ_Scale = -Integ_Lim;

/* Differential (Kd) using "weighted" differential-sample-time */

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;
          }
                    /*****     New DAC Result     *****/

NewDAC = (Prop_Val + Integ_Scale + Diff_Val) / 16;

     /** Verify that the DAC step change calculated is less than the Max Allowed **/
               /* (does not exceed the actual accel/decel requirement) */

if ((abs(ActVEL) - abs(NewDAC)) > MxDAC_Stps)
  {     if ((NewDAC >= 0) && (ActVEL >= 0) && (NewDAC > ActVEL))
               {  NewDAC = ActVEL + MxDAC_Stps;  }
else if ((NewDAC >= 0) && (ActVEL >= 0) && (NewDAC < ActVEL))
               {  NewDAC = ActVEL - MxDAC_Stps;  }
else if ((NewDAC < 0) && (ActVEL < 0) && (abs(NewDAC) > abs(ActVEL)))
               {  NewDAC = ActVEL - MxDAC_Stps;  }
else if ((NewDAC < 0) && (ActVEL < 0) && (abs(NewDAC) < abs(ActVEL)))
               {  NewDAC = ActVEL + MxDAC_Stps;  }
else if ((((NewDAC >= 0) && (ActVEL < 0)) && ((NewDAC - ActVEL) > MxDAC_Stps)))
               {  NewDAC = ActVEL + MxDAC_Stps;  }
else if ((((NewDAC < 0) && (ActVEL > 0)) && ((ActVEL - NewDAC) > MxDAC_Stps))
               {  NewDAC = ActVEL - MxDAC_Stps;  }

                    /*****     Implement the new DAC     *****/

DAC = NewDAC;                              /* set the DAC to the determined value */
UpDtCnt = 0;                               /* restart the time counter */

Traj_ext:                                  /* This ends the traj update intrpt */
```


Notes: