Generate stepper-motor speed profiles in real time

From <https://www.embedded.com/> (December 30, 2004)

**A new algorithm for stepper-motor acceleration allows speed profiles to be parameterized and calculated in real time. This algorithm can run on a low-end microcontroller using only simple fixed-point arithmetic operations and no data tables. It develops an accurate approximation for the timing of a linear ramp with constant acceleration and deceleration.**

It's commonly thought that the timing of a linear speed ramp for a stepper motor is too complex to be calculated in real time. The exact formula for the step delay is in Equation 8. The solution has been to store the ramp data in precompiled arrays, but this method is inflexible and wastes memory. The alternative has been to use a more powerful and expensive processor than otherwise needed or a high-level stepper-control IC. This article develops an accurate approximation that has been implemented in C using 24.8 fixed-point arithmetic on a mid-range PIC microcontroller.

Motor step signals can be generated to be timed to the resolution of one timer period. Each step advances the motor by a constant increment, typically 1.8 degrees on a typical stepper motor.

The timer frequency should be as high as possible while still allowing long delays as the motor is accelerated from stop. A timer frequency of 100 kHz has been used. A maximum motor speed of 300rpm is then equivalent to a delay count of 100. It's necessary to have high timer resolution to give smooth acceleration at high speed.

**Notation and basic formulas**  
Delay (sec) programmed by timer count *c*:

 Equation 1:

= timer frequency (Hz).  
Motor speed ω (rad/sec) at fixed timer count c:

 Equation 2:

α = motor step angle (radian).  
1rad = 180/π = 57.3 deg. 1rad/sec = 30/π = 9.55 rpm.

Acceleration  (rad/sec2) from adjacent timer counts  and :

 Equation 3:

Equation 3 assumes fixed-count speed (Equation 2) at the midpoint of each step interval (Equation 1), as on a linear ramp, Figure 1. Note that  resolution is inversely proportional to the cube of the speed.

A picture containing line, text, plot, diagram

Description automatically generated

Figure 1: Ramp geometry: move of m =12 steps

**Linear speed ramp—exact**  
On a linear ramp, acceleration ω' is constant, and speed ω(*t*) = ω'*t*. Integration gives the motor shaft angle θ(*t*):

 Equation 4:

*n*≥ 0 step number (real). When the shaft is at θ = *n*α, (integer *n*) it's time for the *n-*th step pulse:

 Equation 5:

The exact timer count to program the delay between the *n-*th and (*n*+1)-th pulses (*n*≥ 0) is:

 Equation 6:

The initial count *c*0 factorizes out to give Equations 7 and 8:

 Equation 7:

 Equation 8:

Note that *c*0 sets the acceleration, proportional to (1/*c*0)2 .

In real-time, Equation 8 would require calculation of a square root for each step, with the added problem of loss of precision by subtraction.

**Approximating linear ramp**  
Ratio of successive exact timer counts from Equation 8:

 Equation 9:

Taylor series:

 Equation 10:

Equation 11 is the second-order approximation to Equation 9 using Equation 10:

 Equation 11:

Equation 11 can be rearranged for faster calculation:

 Equation 12:

Finally, we can disconnect the physical step number, *i*, from the step number *n*on a ramp from zero, to give the general-purpose ramp algorithm shown in Equation 13. Here *n*determines the acceleration and increments with *i*for constant acceleration. To ramp up from stop, *ni*= *i*, *i*= 1,2, … :

 Equation 13:

Negative *n*-values give deceleration. Equation 14, with *ni*= *i*– *m*, can be used to ramp any speed down to stop in the final steps of a move of *m*steps:

 Equation 14:

**Table 1: Accuracy of the step-delay approximation**

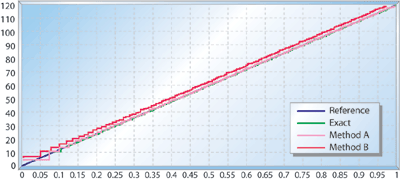
|  |  |  |  |
| --- | --- | --- | --- |
| **Step n** | **Exact (9)** | **Approx (11)** | **Relative error** |
| 1 | 0.4142 | 0.6000 | 0.4485 |
| 2 | 0.7673 | 0.7778 | 0.0136 |
| 3 | 0.8430 | 0.8462 | 0.00370 |
| 4 | 0.8810 | 0.8824 | 0.00152 |
| 5 | 0.9041 | 0.9048 | 7.66E-4 |
| 6 | 0.9196 | 0.9200 | 4.41E-4 |
| 10 | 0.9511 | 0.9512 | 9.42E-5 |
| 100 | 0.9950 | 0.9950 | 9.38E-8 |
| 1,000 | 0.9995 | 0.9995 | 9.37E-11 |

**Accuracy of approximation**  
Table 1 shows that the approximation is accurate even at low step number *n*and relative error decreases with *n3*. However, *n*=1 has a significant inaccuracy. The inaccuracy at *n*=1 can be handled in two ways:

* Treat *n*=1 as a special case. Using *c*1 0.4056 *c*0 compensates for the inaccuracies at the start of the ramp and allows Equation 7 to be used to calculate *c*0.
* Ignore the inaccuracy. In place of Equation 7 use Equation 15:

 Equation 15:

The first alternative gives an almost perfect linear ramp. The second alternative starts with a fast step. This is to the good, as it helps keep the motor moving between step pulses 0 and 1-and establishes the angle error needed to generate torque. It also allows a wider range of accelerations to be generated with a 16-bit timer and has the advantage of simplicity. It's therefore recommended to ignore the inaccuracy at *n*=1.

  
Figure 2: Stepper-motor speed ramp

A picture containing text, screenshot, line, plot

Description automatically generated  
Figure 3: Start of ramp detail

A picture containing text, screenshot, line, plot

Description automatically generated  
Figure 4: End of ramp detail

Figures 2 through 4 compare the options for a target ramp from 0 to 120 rpm in 1sec. For clarity, step changes in speed are shown, calculated from Equation 2. The true profile should be close to a straight line.

2*c*/(4*n*+1) in Equation 12 could be approximated by *c*/2*n*. Some effects would be:

* The algorithm would still produce a linear ramp.
* *c0*would be closer to the “exact” value shown in Equation 7: 88.6% instead of 67.6% for the same ramp acceleration.
* A single equation like Equation 13 could no longer be used for both acceleration and deceleration.

**Changes of acceleration**  
From Equations 4 and 5 we can obtain an expression for the step number *n*as a function of speed and acceleration:

Equation 16:

Thus, the number of steps needed to reach a given speed is inversely proportional to the acceleration:

Equation 17:

This makes it possible to change the acceleration at a point on the ramp by changing the step number *n*in the ramp algorithm Equation 13. Moreover, using signed ω' values results in signed *n*-values that behave correctly in the algorithm. Only ω' = 0 needs special handling.

The *n*-value given by Equation 17 is correct for *tn*. However, *cn*represents an average for the interval *tn*… *tn+1*. Equation 17 is usually adequate, but it's more accurate to add a half-step to *n*-values for use in the ramp algorithm:

 Equation 18:

The numerical example shown in Table 2 changes acceleration from 10 to 5 and to -20rad/sec2 from step 200. Complex speed profiles can be built up piecewise in this way.

**Table 2: Acceleration changes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Step i** | **ni** | **ci (13)** | **ω' (3)** | **notes** |
| 198 | 198 | 2813.067 |  |  |
| 199 | 199 398.5 -100.25 | 2806.008 | 10 | 10(199+0.5) = 5(398.5+0.5) = -20(-100.25+0.5) |
| 200 | 399.5 | 2803.498 | 5 | etc. |
| 201 | 400.5 | 2799.001 | 5 |
| 200 | -99.25 | 2820.180 | -20 | etc. |
| 201 | -98.25 | 2834.568 | -20 |

**Deceleration ramp**  
For a short move of *m*steps, where the up-ramp at ω'1 meets the down-ramp at ω'2 before max speed is reached, the step number *m*at which to start decelerating is, from Equation 17:

 Equation 19:

ω'1 = acceleration, ω'2 = deceleration (positive). Round *n*to integer and calculate *cn*… *cm-1*using Equation 14.

In other cases, Equations 17 or 18 can be used to calculate the number of steps *n*2 needed to stop at deceleration ω'2, given that the present speed was reached at step *n*1 with acceleration ω'1. Round *n*2 to integer and calculate *cm-n2…cm-1*using Equation 14.