Real Time Stepper Motor Linear Ramping Just by Addition and Multiplication

Aryeh Eiderman <leib@eiderman.com>

1. Kinematic basics

The linear acceleration (ramping) formulas are:

$$S = v_0 \cdot t + a \cdot t^2 / 2 \qquad [1],$$

$$v = v_0 + a \cdot t \qquad [2]$$

where

- **S** - acceleration distance, in stepper motor case - **number of steps**,
- v_0 - initial velocity, **base speed** (steps per second),
- **v** - target velocity, **slew speed** (steps per second),
- **a** - **acceleration** (steps per second per second),
- **t** - acceleration time, **ramping period** (seconds).

By rearranging [2]

$$t = (v - v_0) / a \qquad [3]$$

and putting it in [1] we have

$$S = (v^2 - v_0^2) / (2 \cdot a) \qquad [4]$$

and

$$v = (v_0^2 + 2 \cdot a \cdot S)^{1/2} \qquad [5]$$

that can be represented as a **recursive form** of speed calculation for **one step**:

$$v_i = (v_{i-1}^2 + 2 \cdot a)^{1/2} \qquad [6]$$

where

- **i** - step number $(1 \le i \le S)$.

2. Control basics

To produce the speed profile for stepper motor we need to provide the real time delays between step pulses:

$$p_i = F / v_i \qquad [7]$$

where

- p_i - delay period for the **i**-th step (timer ticks),
- **F** - **timer frequency** (count of timer ticks per second),

so according to [6] the exact delay value will be:

$$p_i = F / ((F / p_{i-1})^2 + 2 \cdot a)^{1/2} \qquad [8]$$

or

$$p_i = p_{i-1} / (1 + p_{i-1}^2 \cdot 2 \cdot a / F^2)^{1/2} \qquad [9].$$

3. Approximation

Using Taylor series

$$1 / (1 + n)^{1/2} \simeq 1 - n / 2 \qquad [10]$$

when $-1 < n \le 1$ we can approximate [9] to

$$p_i = p_{i-1} \cdot (1 - p_{i-1}^2 \cdot a / F^2) \qquad [11].$$

Let's check the $-1 < n \le 1$ condition. Our **n** was

$$n = p_{i-1}^2 \cdot 2 \cdot a / F^2 \qquad [12]$$

or, by velocity,

$$n = 2 \cdot a / v_{i-1}^2 \qquad [13].$$

The maximum **n** value will be at minimum speed, on the first calculated step, where $i = 2$

$$n_{max} = 2 \cdot a / v_1^2 \qquad [14].$$

Because the minimal v_0 is 0, from [6] we have

$$v_{1min} = (2 \cdot a)^{1/2} \qquad [15].$$

So **n** will be **always** less than or equal to 1. Because our calculations are forward-only we have no limitation in case of deceleration (negative acceleration) too.

4. Implementation

The given parameters are:

- v_0 - base speed,
- **v** - slew speed,
- **a** - acceleration,
- **F** - timer frequency

and the calculated parameters are:

- **S** - acceleration/deceleration distance

$$S = (v^2 - v_0^2) / (2 \cdot a) \qquad [4, 16],$$

- p_1 - delay period for the **initial** step

$$p_1 = F / (v_0^2 + 2 \cdot a)^{1/2} \qquad [17],$$

- p_S - delay period for the **slew speed** steps

$$p_S = F / v \qquad [18],$$

- **R** - constant multiplier

$$R = a / F^2 \qquad [19].$$

The variable delay period **p** (initially $p = p_1$) that will be recalculated for each next step is:

$$p = p \cdot (1 + m \cdot p \cdot p) \qquad [20].$$

where

- **m** - variable multiplier that depends on the movement phase:
  - $m = -R$ during acceleration phase,
  - $m = 0$ between acceleration and deceleration phases,
  - $m = R$ during deceleration phase.

For accuracy purpose let's set

- $p = p_S$ if $p < p_S$ or between acceleration and deceleration phases,
- $p = p_1$ if $p > p_1$.

5. Optional enhancement

Using the higher order approximation of Taylor series

$$1 / (1 + n)^{1/2} \simeq 1 - n / 2 + 3 \cdot n^2 / 8 \qquad [21]$$

we can get more accurate results replacing [20] with

$$p = p \cdot (1 + q + 1.5 \cdot q \cdot q) \qquad [22]$$

where

$$q = m \cdot p \cdot p .$$

By [22] we have *excellent* precision but with two extra multiplications and one extra addition vs [20]'s *good* precision way. Finally let's construct a *very good* compromise with just one extra multiplication and one extra addition:

$$p = p \cdot (1 + q + q \cdot q) \qquad [23].$$

However I think that the *good* way ([20]) is not *merely good* but even *good enough* for most of stepper motor applications.
The enhancement is important for servo drivers with the step/direction control that ramping up from and down to zero speed.

6. Programming note

This algorithm was designed for **floating point** mathematics and in this form it works faster than in the **integer** form that requires division.

** This algorithm was developed by the author in 1994 for L.I.D. Ltd as a part of POEM Stepper Organizer software to control up to 4 axes through IBM PC's parallel port (LPT) and was ported to microcontroller platform in 2004.*
The main field of usage since 1994 - laser diamond cutting.

*** Special thanks to David Austin <dave@slotech.fsnet.co.uk> who helped me with the enhancement part.*