Adapting LeibRamp to Integer Calculations

Gabriel Venberg

2025-11-23

Introduction

A paper by Aryeh Eiderman [1] submits an efficient algorithm for real time stepper motor ramping where all expensive computations are precomputed, leaving only multiplication and addition in the real time loop. However, Eiderman's paper is explicitly designed for floating point arithmetic, and does not work for integer arithmetic. Eiderman notes that was originally designed for an IBM PC, which may have had a floating point coprocessor. However, today most stepper motors are controlled by microcontrollers and not full x86 machines. Many microcontrollers do not have a floating point unit and so here we investigate a modification to allow Eiderman's algorithm to work with integer arithmetic.

This modification was developed for `iter-step-gen`, a Rust-based asynchronous stepper motor step planner and driver written out of spite to control the author's window blinds with an esp32c3.

Original formula

With the following inputs,

$$\begin{aligned} d &= \text{move distance} \\ v_0 &= \text{Inital speed } \left(\frac{\text{steps}}{\text{sec}}\right) \\ v &= \text{Max speed } \left(\frac{\text{steps}}{\text{sec}}\right) \\ F &= \text{tick frequency (Hz)} \\ a &= \text{target acceleration } \left(\frac{\text{steps}}{\text{sec}^2}\right) \end{aligned} \tag{1}$$

Eidermans update formula is as follows:

$$p = p(1 + mp^2) \tag{2}$$

Where:

$$\begin{aligned} p &= \text{the delay between steps} \\ m &= \begin{cases} -\frac{a}{F^2} \text{ if accelerating} \\ 0 \text{ if cruising} \\ \frac{a}{F^2} \text{ if decelerating} \end{cases} \end{aligned} \tag{3}$$

Avoiding small numbers

LeibRamp works fine for floating point values, and indeed, the paper calls out that this algorithm is designed for them. However, for integer math, natively transcribing the above algorithm into code results in several issues:

- m is almost always 0, as F^2 is very large (for some microcontrollers, it is in-fact dangerously close to 2^{64})
- $(1 + mp^2)$ is intended to always be between 0 and 2, usually around 1. In integer math, this means it is always 0 or 1, resulting in no motion or no acceleration.

However, we can do a few transformations to avoid small numbers in intermediate calculations, making the fractional part much less significant.

Firstly, instead of storing the (most likely precomputed) $m = \frac{a}{F^2}$, we can store its inverse, $m^{-1} = \frac{F^2}{a}$. This will be a very large number rather than a very small number, avoiding truncation to zero. Due to this transformation, we now divide by m^{-1} in Eq. 2. The update formula becomes:

$$p = p\left(1 + \frac{p^2}{m^{-1}}\right) \tag{4}$$

Secondly, we can change the grouping of the final calculation. Where $(1 + mp^2)$ is $\cong 1$, both m and p^2 are relatively large. We can use this to distribute p in Eq. 4, causing the intermediate calculations to avoid small numbers, like so:

$$p = p + \frac{p^3}{m^{-1}} \tag{5}$$

Finally, if we are also using unsigned integers, during acceleration we can, instead of negating m^{-1} in Eq. 5, we can subtract p from $\frac{p^3}{m^{-1}}$, making the update function:

$$p = p \pm \frac{p^3}{m^{-1}} \tag{6}$$

Remainder carrying

Unfortunately, the flooring after every division inherent in integer arithmetic reduces precision significantly, and causes the acceleration curve to be asymmetrical with respect to the deceleration curve. This can be fixed, however, by storing the remainder of each division and adding that remainder to the next iteration. Eq. 6 the following pair of equations:

$$\begin{aligned} p &= p \pm \frac{p^3 + r}{m^{-1}} \\ r &= (p^3 + r) \bmod m^{-1} \end{aligned} \tag{7}$$

Modifying the optional enhancement

Eiderman posits an optional precision enhancement using a couple extra computations to increase the accuracy of the algorithm:

$$p = p(1 + q + q^2) \tag{8}$$

where $q = mp^2$.

We can apply similar transformations to this. As we have already calculated m^{-1}, we can redefine q as:

$$q = \frac{p^2}{m^{-1}} \tag{9}$$

and distribute p in Eq. 8:

$$p = p \pm pq + pq^2 \tag{10}$$

Unfortunately, q is also very close to 0, so we instead calculate the inverse, $q^{-1} = \frac{m^{-1}}{p^2}$.

and divide rather than multiply in Eq. 10:

$$p = p \pm \frac{p}{q} + \frac{p}{q^2} \tag{11}$$

Adding remainder storage is straightforward with this enhancement, though it requires 3 separate remainder variables to be stored:

$$\begin{aligned} q^{-1} &= \frac{m^{-1} + r_1}{p^2} \\ p &= p \pm \frac{p + r_2}{q} + \frac{p + r_3}{q^2} \\ r_1 &= (m^{-1} + r_1) \bmod p^2 \\ r_2 &= (p + r_2) \bmod q \\ r_3 &= (p + r_3) \bmod q^2 \end{aligned} \tag{12}$$

Unlike Eidermans method, where this enhancement requires only one extra addition and one extra multiplication, in the integer form it requires 2 extra divisions and an addition. Due to the extra 2 divisions, and the extra space needed for the 2 extra remainders, this was deemed not worth the extra precision in the authors usecase.

Implementation considerations

For convenience of the reader, the following are the remaining variables needed to implement a linear ramping step planner.

$$\begin{aligned} p_1 &= \frac{F}{\sqrt{v_0^2 + 2a}} && \text{delay period for inital step} \\ p_c &= \frac{F}{v} && \text{delay period for cruise period steps} \\ S &= \frac{v^2 - v_0^2}{2a} && \text{distance needed for acceleration to } v \\ S_a &= \begin{cases} S \text{ if } d > 2S \\ \lceil \frac{d}{2} \text{ if } d \leq 2S \end{cases} && \text{actual distance needed for acceleration/decceleration} \end{aligned} \tag{13}$$

A move can be split into 3 parts, the acceleration phase, the cruise phase, and the deceleration phase. During the acceleration phase, which lasts until $p \leq p_c$, the $\pm$ is a subtraction. During the cruise phase, which lasts until the remaining steps in the move $\leq S_a$, p should be held constant at p_c. During the deceleration phase, which lasts until the target position is reached, the $\pm$ is an addition.

Finally, the *ideal* formula, useful in unit tests and verification, is:

$$p = \frac{F}{\sqrt{\left(\frac{F}{p}\right)^2 + 2a}} \tag{14}$$

Bibliography

[1] A. Eiderman, “Real Time Stepper Motor Linear Ramping Just by Addition and Multiplication,” Accessed: Nov. 23, 2025. [Online]. Available: http://hwml.com/LeibRamp.pdf