**STM32H750으로 스텝 모터 제어하기**

1. **개요**

스텝 모터(Step Motor), 스테퍼 모터(Stepper Motor), 스테핑 모터(Stepping Motor)등으로 불리고 있는 모터 얘기를 할까 합니다. 모두 같은 것을 말하는 것으로 여기서는 스텝 모터라고 부르겠습니다.

스텝 모터는 종류도 많고 구동 방법 또한 다양합니다. 이것에 대해 설명하려면 그냥 훑고 지나가기만 해도 지루해 질 정도로 양이 많으므로, 여기서는 모터와 구동 드라이버가 갖추어져 있어서 “한 펄스를 주면 일정한 각도만큼 회전하는 모터”에서부터 시작을 하려고 합니다. 보통 여기까지는 하드웨어 적인 것이고 구매해서 사용하면 되기 때문입니다. 현재는 한 때 고급 기술이었던 마이크로 스텝(Micro Step) 기능까지 겸비한 드라이버들을 쉽게 구할 수 있습니다.

스텝 모터는 센서 피드백이 없이(Open Loop) 정확한 각도나 위치를 간단히 제어하기 편리한 모터입니다. 비교적 저렴 하기도 합니다. 이런 이점을 살려 주변에서 흔히 볼 수 있는 3D 프린터에 거의 대부분이라고 해도 좋을 만큼 무척 많이 사용됩니다. 움직임을 보고 있으면 환상적이죠.

그러나 구동하기 좀 까다로운 면도 있습니다. 가속과 감속이 그것입니다. 빠른 속도로 돌리려면 천천히 속도를 높여 주어야 합니다. 갑자기 빠른 펄스를 인가하면 탈조라는 현상을 일으키며 삐~ 소리만 내고 돌아가지 않습니다. 갑자기 멈추는 것도 조심해야 합니다. 전체 무게(관성)과 최대 속도, 모터의 크기(구동 능력)등을 검토하여 가속도를 얼마로 해야 할 지 결정해야 합니다. 그리하여 물체를 옮기는 데 있어서 어디까지 가속하고, 어디까지는 정속으로 가고, 어디부터 감속을 할 것인지 결정한 후 시간에 맞게 펄스열을 만들어 스텝 모터 드라이버에 보내 주어야 합니다. 골치 아픈 일이 아닐 수 없습니다.

1. **Leib Ramp**

가장 자연스럽고 최대 성능을 끌어 내려면 고등학교 물리 시간에 배운 가속도 운동에 관한 공식을 풀어서 시간에 따른 펄스 열을 배열해야 합니다. 이것을 해 놓은 사람이 있습니다. 그것도 사양이 낮은 마이크로 컨트롤러를 위해서 덧셈과 곱셈만으로 계산할 수 있게 풀어놨습니다.

Leib Ramp 알고리즘이라는 것인데, <http://www.hwml.com/LeibRamp.htm> 에 가면 볼 수 있습니다. 길지도 않습니다. 좀 긴 한 페이지 분량입니다. 그리 길지도 않으니 여기에 옮겨와 보겠습니다.

**Real Time Stepper Motor Linear Ramping  
Just by Addition and Multiplication**

**Aryeh Eiderman**[**<leib@eiderman.com>**](mailto:leib@eiderman.com)

|  |
| --- |
| **1. Kinematic basics**  The linear acceleration (ramping) formulas are:  **S** **=** **v0 . t + a . t2 / 2**        [1],  **v** **=** **v0 + a . t**        [2]  where          **S**  -  acceleration distance, in stepper motor case - **number of steps**,         **v0**  -  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 - v0) / a**        [3]  and putting it in [1] we have  **S** **=** **(v2 - v02) / (2 . a)**        [4]  and  **v** **=** **(v02 + 2 . a . S)1/2**        [5]  that can be represented as a **recursive form** of speed calculation for **one step**:  **vi** **=** **(vi-12 + 2 . a)1/2**        [6]  where          **i**  -  step number  (1 < **i** < **S**).  **2. Control basics**  To produce the speed profile for stepper motor we need to provide the real time delays between step pulses:  **pi** **=** **F / vi**        [7]  where          **pi**  -  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:  **pi** **=** **F / ((F / pi-1)2 + 2 . a)1/2**        [8]  or  **pi** **=** **pi-1 / (1 + pi-12 . 2 . a / F2)1/2**        [9].  **3. Approximation**  Using Taylor series  **1 / (1 + n)1/2** **~** **1 - n / 2**        [10]  when  -1 < **n** < 1  we can approximate [9] to  **pi** **=** **pi-1 . (1 - pi-12 . a / F2)**        [11].  Let's check the  -1 < **n** < 1  condition. Our  **n**  was  **n** **=** **pi-12 . 2 . a / F2**        [12]  or, by velocity,  **n** **=** **2 . a / vi-12**        [13].  The maximum  **n**  value will be at minimum speed, on the first calculated step, where  **i** = 2  **nmax** **=** **2 . a / v12**        [14].  Because the minimal  **v0**  is 0, from [6] we have  **v1min** **=** **(2 . a)1/2**        [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:          **v0**  -  base speed,         **v**  -  slew speed,         **a**  -  acceleration,         **F**  -  timer frequency  and the calculated parameters are:          **S**  -  acceleration/deceleration distance  **S** **=** **(v2 - v02) / (2 . a)**        [4, 16],          **p1**  -  delay period for the **initial** step  **p1** **=** **F / (v02 + 2 . a)1/2**        [17],          **pS**  -  delay period for the **slew speed** steps  **pS** **=** **F / v**        [18],          **R**  -  constant multiplier  **R** **=** **a / F2**        [19].  The variable delay period  **p**  (initially  **p** = **p1**)  that will be recalculated for each next step is:  **p** **=** **p . (1 + m . p . p)**        [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** = **pS**  if  **p** < **pS**  or between acceleration and deceleration phases,         **p** = **p1**  if  **p** > **p1** .  **5. Optional enhancement**  Using the higher order approximation of Taylor series  **1 / (1 + n)1/2** **~** **1 - n / 2 + 3 . n2 / 8**        [21]  we can get more accurate results replacing [20] with  **p** **=** **p . (1 + q + 1.5 . q . q)**        [22]  where          **q** **=** **m . p . 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 . (1 + q + q . q)**        [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>*](mailto:dave@slotech.fsnet.co.uk)*who helped me with the enhancement part.* |

1. **프로그래밍(코딩)**

그래도 만만해 보이지는 않습니다. 실제 3D 프린터의 오픈소스 프로그램인 MarLin (<https://marlinfw.org>)과 역시 간단한 CNC 같은 것의 오픈소스인 Grbl (<https://github.com/grbl/grbl>) 에서도 사용하는 알고리즘입니다. 앞 둘 중에 하나의 소스코드를 내려 받아 분석해 보면 실제 어떻게 구현 되었는지 알 수 있을 뿐더러, 여러 개의 모터를 동시에 움직일 때의 코딩까지 얻을 수 있습니다. 하지만 너무 방대하고 어렵습니다. 그래서 인터넷을 뒤져서 허접하게 구현된 몇몇 자료들을 참고로 유래택(<https://uraetech.co.kr/>)의 STM32H750VBT6 보드에서 돌아갈 수 있게 코딩하였습니다.

펄스를 만드는 데 2개의 타이머가 쓰이는 데, 하나는 펄스 간격을 조절하는 타이머입니다. 이 타이머의 주기를 설정하는 데 Leib Ramp 알고리즘이 사용됩니다. 그리고 다른 하나의 타이머는 펄스의 폭을 결정합니다.

실제 사용을 위해서는 좀 더 보완 코드가 필요하지만 개념을 이해하는 데 중점을 두었습니다. 다른 마이컴으로도 쉽게 포팅이 가능할 것입니다.

소스 크기도 얼마 안되고, 주석으로 자세히 설명을 해 놓았으므로 참고하면 금방 이해할 수 있습니다. 프로그램은 크게 2부분으로 구성이 되는 데, StepperMotor.c 에 모터의 움직임에 관한 코드가 들어있고 main.c 에 PC와 통신 인터페이스에 관한 코드가 들어있습니다.

PC에서는 HTem(<http://der-hammer.info/pages/terminal.html>)과 같은 시리얼 터미널 프로그램에서 텍스트로 명령을 내리고 결과를 봅니다. UART와 USB CDC를 사용할 수 있게 했습니다. 주의할 것은 한 명령 라인의 끝을 LF(Line Feed)로 하므로 라인 끝에 CR/LF 또는 LF 가 붙게 설정해 주어야 합니다. 대소문자는 가리지 않고 공백은 무시됩니다.

예를 들어 속도를 100mm/sec로 설정하고 싶다면

S 100(LF)

를 보내면 됩니다. 원점으로부터 12.3mm 이동하라는 명령어는

P 12.3(LF)

이고, 현 위치에서 -5mm 가라는 것은

D -5(LF)

입니다.

그 외에 현재 A, L, G, E, I 명령이 있는 데 기능은 다음과 같습니다.

**A : 가속도 설정**

출발과 정지할 때 속도를 어느 정도로 높이거나 줄일 지 결정합니다.

**S : 속도 설정**

움직일 때 최대 속도입니다. 가속이 끝나고 감속이 시작하기 전의 속도입니다. 이동 거리가 짧아서 가속과 감속 구간의 합보다 작으면 이 속도에 도달하지 못하기도 합니다.

**L : 현 위치 재설정**

현재 위치를 임의의 값으로 설정합니다. L 10.5(LF) 라고 보내면 지금부터 현위치는 원점으로부터 10.5mm 떨어진 곳입니다.

**G : 현 위치 표시**

파라메터 없이 G(LF) 를 보내면 현재 위치를 리턴해 줍니다.

**D : 상대 위치 이동**

현재 위치를 기준으로 얼마만큼 이동하고 싶을 때 사용합니다. D -1.5(LF)는 현위치에서 역방향으로 1.5mm 이동하라는 뜻입니다.

**P : 절대 위치 이동**

지정한 위치로 이동합니다. 본 프로그램은 내부적으로 위치를 관리하고 있습니다.

**E : 즉시 정지**

이동 중에 이 명령을 보내면 감속 과정 없이 바로 정지합니다.

**I : 현재 설정 상태 표시**

현재의 가속도, 최대 속도, 위치 등을 리턴합니다.

간단하므로 소스에 조금 코딩을 더하면 새로운 기능을 추가할 수도 있습니다.

실제 사용을 위해서는 각종 예외 상황에 대한 대처 등 신뢰성을 높이는 코딩이 추가되어야 하고, 경로의 처음과 끝을 처리할 수 있는 리미트에 대한 하드웨어와 소프트웨어가 상황에 맞게 추가 되어야 할 것입니다.

\*\*\* 본격적인 모터 제어를 위해서 Grbl를 포팅해 놓은 소스 코드도 있으니 참고하시기 바랍니다. \*\*\*