A LOW-COST SOLUTION TO THE HIGH SPEED OPERATION OF STEPPER MOTORS.

EurIng Julian Cox BSc(hons) Msc MSAIEE CEng
Senior lecturer, CITE, University of East London.
f.j.cox@uel.ac.uk

Abstract: Stepper motors are small electric motors whose shaft rotation can be accurately controlled by electronic pulses (a "step" pulse). They are able to provide a simple solution in applications where positional or accurate speed control is required without the need for position sensing – i.e. open loop control. In order to maintain reliable control careful consideration must be given to the timing of the step pulses. This paper describes a novel approach to the design of the software for a low cost microcontroller that will generate the step pulses to accelerate and decelerate the motor, thereby maintain positional integrity.

1. Introduction

A stepper motor is a synchronous AC motor with the number of rotor and stator poles increased, but with no common denominator. Soft magnetic material with many teeth on the rotor and stator multiplies the number of poles (reluctance motor). Modern steppers are of hybrid design, having both permanent magnets and soft iron cores. In figure 1 the armature has four poles and the stator has 25 poles so it will require 100 steps to complete one revolution. In most commercial stepper motors there are 200 steps/rev resulting in 1.8^{o}/step.
To achieve full rated torque, the coils in a stepper motor must reach their full rated current during each step. Winding inductance and reverse EMF generated by a moving rotor tend to resist changes in drive current, so that as the motor speeds up, less and less time is spent at full current, thus reducing motor torque. As speeds further increase, the current will not reach the rated value, and will therefore produce less torque. Eventually the motor will produce insufficient torque to cause a step rotation. The resulting missed step will therefore compromise the positional integrity.

1.1 Pull-in torque

This is the measure of the torque produced by a stepper motor when it is operated without an acceleration state or when the initial step frequency is too high. At low speeds the stepper motor can synchronize itself with an applied step frequency, and this pull-in torque must overcome static friction and inertia. The start-up step frequency must therefore be less than the value for which the motor fails to rotate. Once the motor has started to rotate, its inertia will allow the step frequency to be increased. Even then a sudden large change in step frequency can cause the motor to lose synchronisation.

1.2 Pull-out torque

The stepper motor pull-out torque is measured for a given torque loading by accelerating the motor until the motor stalls or "pulls out of synchronism".

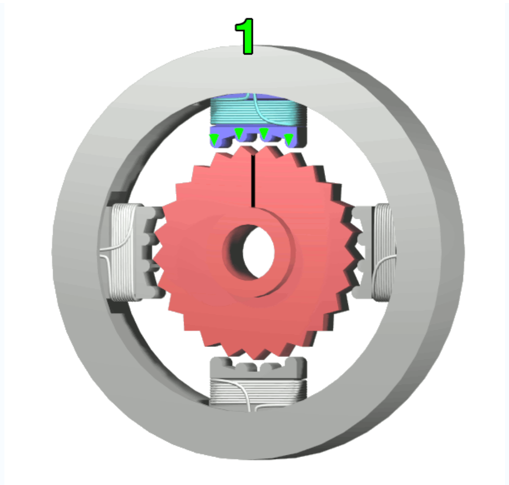


Figure 1. Schematic view of a stepper motor

i

Thiss is the maximum value of step frequency for a particular motor/load combination. The motor can also stall if the acceleration is too great and will rotate extra steps if the deceleration is too great.
In both of the above cases the motor will oscillate at the step frequency. Therefore the maximum value for start-up frequency, acceleration and running frequency must be used in determining the operating characteristics.

2. Dynamics of the stepper motor.

For constant acceleration, k*:* -

$a = k \therefore v = kt + c \therefore s = kt^2 + dt + e$ ...(1)

$where\ a = acceleration,$

$v = velocity,$

$s = angular\ displacement$

$c, d\ and\ e\ are\ constants\ of\ integration$

Since the motor initial condition is "at rest" *c*, *d* and *e* are all zero. Therefore: -

$a = k,\ \ v = kt,\ \ s = kt^2$ ...(2)

If the motor is accelerated to some value v_{max}, run at constant velocity and decelerated to zero velocity the profile is shown in figure 2. During the acceleration and deceleration phases step frequency increases linearly and the displacement follows a quadratic curve.

2.1 Timing of the step pulses.

One simple method for generating the step pulses is to use a single pulse generator and vary the time between pulses using a high frequency oscillator driving a presettable counter thereby controlling the time between pulses as determined by the counter preset value and the oscillator. By continuously varying the preset value between pulses acceleration and deceleration can be controlled. A microcontroller is ideally suited to this task as it typically contains the necessary counter. All that is required is to set the count value between pulses.
For a body in motion, the distance travelled is defined by the equation: -

$s = ut + 1/2at^2$

$where\ u\ is\ the\ initial\ velocity$ ...(3)

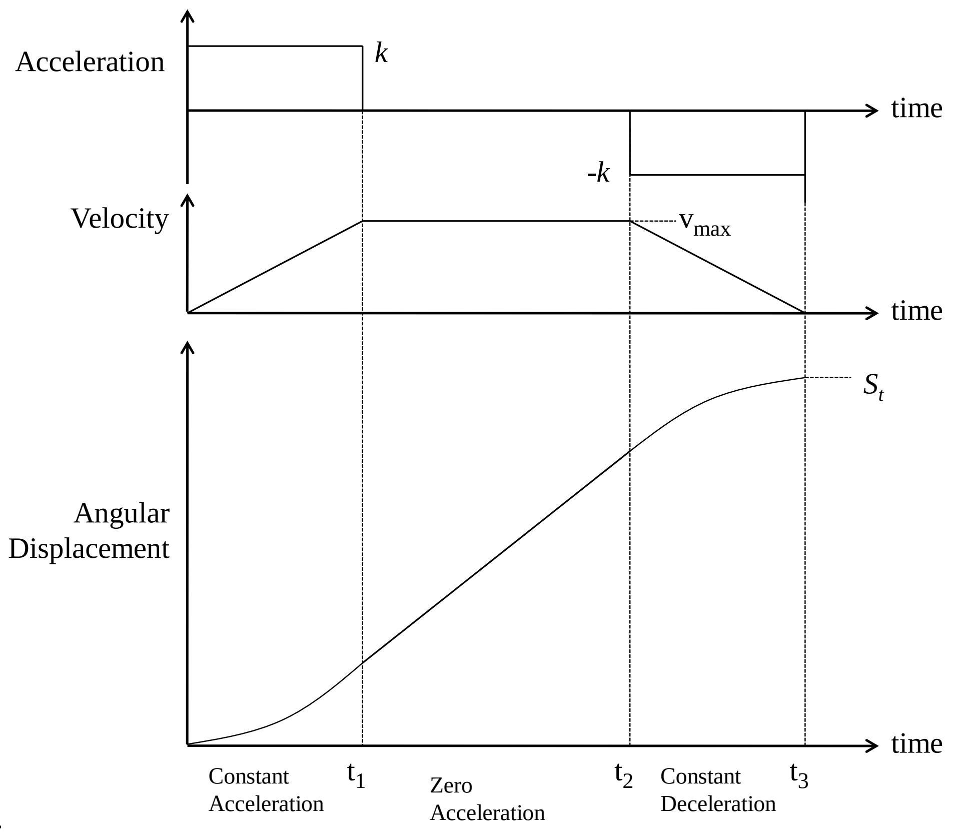
Since the velocity is required to increase linearly it can be assigned an arbitrary value equal to the step number, starting from zero. Equation (3) can be simplified to: -

$Assuming\ a = 1, s = kt^2\ or\ \ T = \frac{1}{\sqrt{s}}$ ...(4)

Since each step is a discrete event, T can be determined for each step using equation (4). Evaluating equation (4) for each step interval presents two mathematical problems:

- The square root of the total number of steps must be calculated.
- The reciprocal of the result must be calculated.

In both cases, for a small microprocessor, this may present an unacceptable computational overhead.



.

Figure 2. Motor displacement characteristics

A practical implementation.

An 8-bit microprocessor typically contains a number of 16-bit presettable counters, driven by the system clock, which can generate an interrupt when the counter overflows Thus the largest possible count value is 65536 (2^{16}) giving a value for T of $65535/f_{clock}$ and this value is used for T at time t_0.
A spreadsheet (Excel) can be used to calculate all values for T. The table of values produced are normalises with all but the first value less than one. The data table must therefore be scaled and rounded to integer such that all values are greater than 1. (the smallest possible value for T). A look-up table for T has now been generated.
In use it is only necessary to define a pointer into the table and after each overflow of counter the pointer is incremented and the new value for T is loaded as a preset value into the counter. Figure 3 shows the pulse timing scaled by a factor of 50000.

2.2 Data Table configuration.

Experimentation has shown that for a *1MHz* clock, an initial value of *65535*, gives a first value for T of approximately 65ms, which is more than sufficient to overcome the pull-in torque of all but the largest motors. Also a value of *1*, which provides a value for T of *1µs* will cause pull-out on all but the smallest unloaded motors.
Referring to equation (3), as T decreases the velocity increases. Thus, to set a particular value of acceleration, *k*, all that is necessary is to integer-divide every value in the look-up table by the constant *k*.

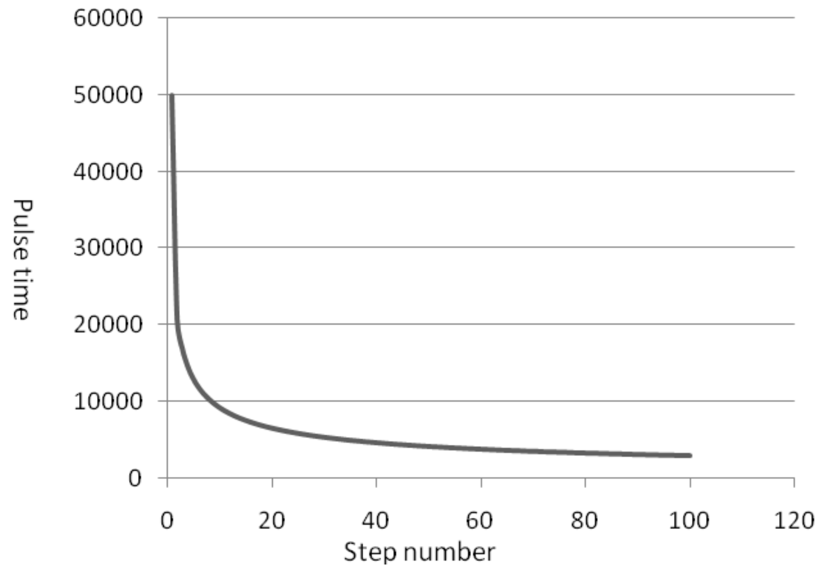


Figure 4. Graph of Pulse time vs Step No. For 100 pulses

To set the maximum velocity for a given acceleration the table is simply truncated. It is not necessary to be able to set the maximum acceleration for a given maximum velocity since it is acceleration that is the pre-dominant factor. Since the truncation process determines the number of steps in the acceleration phase (N_1) and therefore the deceleration phase (N_3) and given the total number of steps (N) it is a simple exercise to calculate the number of steps at constant velocity (N_2).

$N_2 = N - N_1 - N_3 = N - 2N_1$ ...(5)

Because the scaling and sizing of the table occurs before the motor is activated the only operations between steps are to increment the pointer, read the *T*-value and copy it into the counter. The values for k, v_{max} and N can be user inputs or program constants.

If they are user inputs then their values can only be changed at the end of a movement cycle. A data table of 100 integer values will result in a piecewise approximation to equation 3 with a worst case error of less than 2% and therefore the scaling process entails 100 integer divisions. Setting the maximum speed is achieved by stepping through the scaled table until a value less than that equal to the desired maximum speed is reached. That table value is determined by the equation: -

$n = \frac{1}{200T}\ rev/s$ ....(6)

A microcontroller with 8051 architecture and a 12MHz clock will generate a counter clock of 1MHz which means that the time between step pulses will be

$T = x_i * 10^{-6}$ ....(7)

$where\ x_i\ is\ the\ i^{th}\ value\ in\ the\ data\ table$

Therefore $n = \frac{1}{200 * x_i * 10^{-6}} = \frac{5000}{x_i}\ rev/s$

$or\ alternatively\quad x_i = \frac{5000}{n}$ ....(8)

Again this is an integer division.

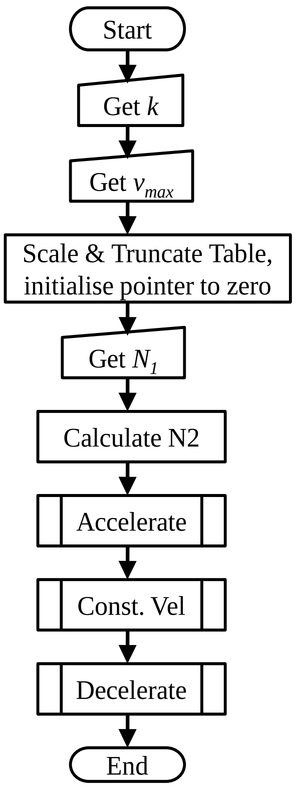


Figure 3. Top level flow chart

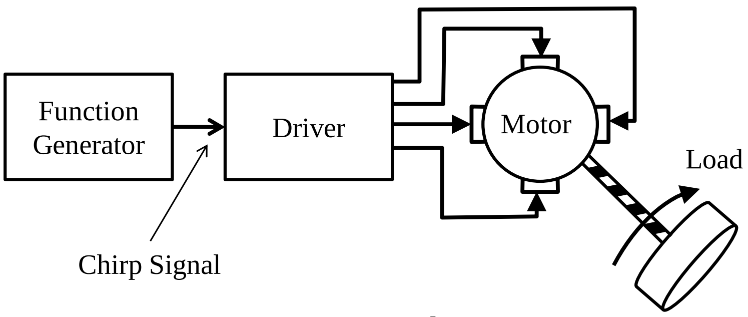


Figure 4. Motor parameter determination

2.3 Determination of limiting values.

In order to calibrate the data table it is necessary to determine the minimum step frequency, the maximum step frequency and the maximum acceleration. For a given application this can be done in a single experiment using the arrangement shown in figure 4. The function generator is set to produce a frequency modulated signal that is modulated by a ramp signal (chirp). By varying the start frequency, the stop frequency and the period of the ramp signal all three parameters can be determined. The start value is increased until the motor fails to start (pull-in) and the end value is increased and the ramp duration is decreased until the motor stalls (pull-out). These values (with a 10% safety margin) are then used to scale the data table. The data table can then be used in the control software without further modification.

3. Experimental results

In practical tests a NXP 89LPC932 microcontroller was used. This is a 12-cycle processor with two 16-bit presettable counters, sufficient on-chip code and data memory, 26 I/O pins and an instruction set that includes bit-level operations. An Intersil controller/driver integrated circuit operating with a supply voltage of 35v was connected to a medium size stepper motor current limited to 2A/phase and driven in bipolar (4-phase) mode. With this configuration the maximum speed was measured to be 1500 rev/min. At this speed the step period is

$$T = \frac{60}{1500 * 200} = 200\mu s \qquad ....(9)$$

To be able to operate the motor at this speed it is necessary that the control algorithm be executed in less than 200µs.

```
T = Table[0];                // Get first
table entry
I=1;                         // Initialise
table index
while (i <= i_max)           // Total
number of steps
{
KB0 = 1;             // Set step pulse high
TH0 = T[i]>>8;       // Load counter
TL0 = T[i];
TR0 = 1;             // Start counter
KB0 = 0;             // Set pulse low
T = Table[i];        // Get next table value
i++;                 // Increment index
while (!TF0);  //Wait for time out }
```

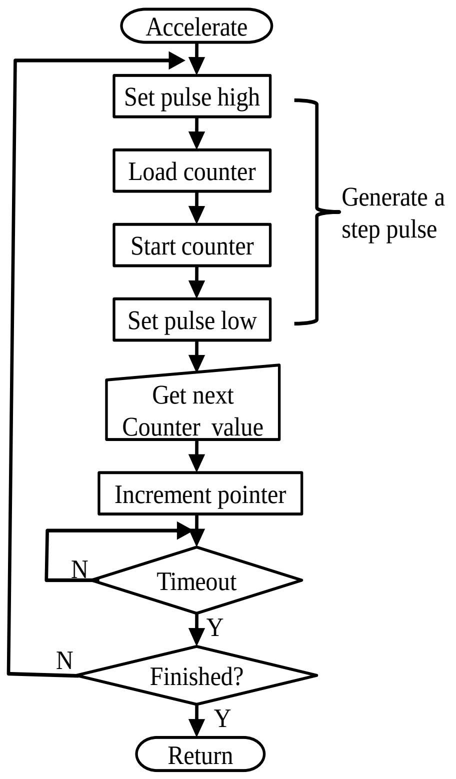


Figure 5. Acceleration code and flow chart

The algorithm with the longest execution time is the Accelerate function (the

Decelerate function takes exactly the same time). The flow chart and C code is shown in figure 5.
Using a NXP LPC900 series microcontroller with a 12MHz clock this code is executed in 7µs. This means that using this device, which has two 16-bit counters, it is possible to control two motors using a single microcontroller.

4. Conclusions

This method of stepper motor control was implemented and tested on a number of different size motors, a 100 entry data table and a 10% safety margin applied to the limit values of acceleration and maximum speed. In all test no missed steps were detected.
With a correctly configured data table, this method of controlling a stepper motor provides the following advantages: -

- A stepper motor can be operated at high speed without losing synchronism; thereby maintaining positional integrity without the need for positional feedback.
- A stepper motor can be operated synchronously with the speed accurately maintained with the precision determined by the system crystal controlled clock without the need for velocity feedback.
- The maximum speed and acceleration can be controlled by the user without the need to recalculate the data table.
- Since the step pulse period values for acceleration and deceleration are stored in a data table all calculations are integer multiplications and divisions and therefore floating point calculations and the accompanying C run-time library is not required. Furthermore, no calculations are carried out within the step algorithms. This minimises code execution time and code size.
- The use of an optimised speed profile means that the motor can be operated at, or close to, its maximum speed. This could be higher than the application requires so an option would be to use an integral reduction gear. This, “mechanical advantage” reduces the load on the motor which in turn provides the option to use a smaller size motor.
- Once a data table has been created it can be used with any motor/load combination. All that is necessary is to determine the acceleration and pull-out frequency per section 2.3 and use these values to scale and truncate the table.

Bibliography

Takashi K, “Stepping motors and their microprocessor controls”,
Oxford University Press, c1984.

UM10108 P89LPC924/925 User manual Rev. 02 — 2 March 2005, www.nxp.com

Data sheets, IC L297 Stepper motor controller, IC L298 Full bridge driver
ST Microelectronics Ltd.