Control of Stepping Motors

A Tutorial

by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science
This material expands on material originally posted to the rec.railroad newsgroup in 1990. Significant parts of this material have been republished as sections 5.2.10, 10.8, 10.9 and 10.10 of the *Handbook of Small Electric Motors* edited by W. H. Yeadon and A. W. Yeadon, McGraw-Hill, 2001.
Copyright © 1995, Douglas W. Jones; major revision 1998. This work may be transmitted or stored in electronic form on any computer attached to the Internet or World Wide Web so long as this notice is included in the copy. Individuals may make single copies for their own use. All other rights are reserved.

Index

Abstract

This tutorial covers the basic principles of stepping motors and stepping motor control systems, including both the physics of steppers, the electronics of the basic control systems, and software architectures appropriate for motor control.

Introduction

Stepping motors can be viewed as electric motors without commutators. Typically, all windings in the motor are part of the stator, and the rotor is either a permanent magnet or, in the case of variable reluctance motors, a toothed block of some magnetically soft material. All of the commutation must be handled externally by the motor controller, and typically, the motors and controllers are designed so that the motor may be held in any fixed position as well as being rotated one way or the other. Most steppers, as they are also known, can be stepped at audio frequencies, allowing them to spin

quite quickly, and with an appropriate controller, they may be started and stopped "on a dime" at controlled orientations.

For some applications, there is a choice between using servomotors and stepping motors. Both types of motors offer similar opportunities for precise positioning, but they differ in a number of ways. Servomotors require analog feedback control systems of some type. Typically, this involves a potentiometer to provide feedback about the rotor position, and some mix of circuitry to drive a current through the motor inversely proportional to the difference between the desired position and the current position.

In making a choice between steppers and servos, a number of issues must be considered; which of these will matter depends on the application. For example, the repeatability of positioning done with a stepping motor depends on the geometry of the motor rotor, while the repeatability of positioning done with a servomotor generally depends on the stability of the potentiometer and other analog components in the feedback circuit.

Stepping motors can be used in simple open-loop control systems; these are generally adequate for systems that operate at low accelerations with static loads, but closed loop control may be essential for high accelerations, particularly if they involve variable loads. If a stepper in an open-loop control system is overtorqued, all knowledge of rotor position is lost and the system must be reinitialized; servomotors are not subject to this problem.

Stepping motors are known in German as *Schrittmotoren*, in French as *moteurs pas à pas*, and in Spanish as *motor paso paso*.

1. Stepping Motor Types

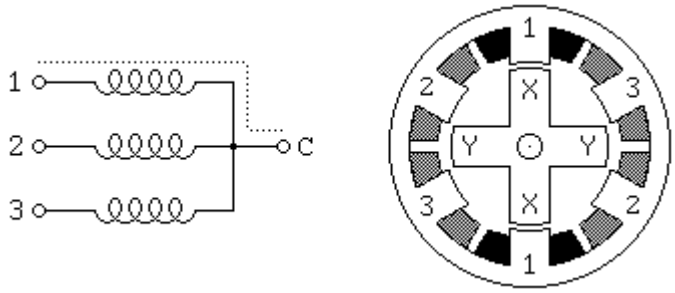
Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

Introduction

Stepping motors come in two varieties, *permanent magnet* and *variable reluctance* (there are also *hybrid* motors, which are indistinguishable from permanent magnet motors from the controller's point of view). Lacking a label on the motor, you can generally tell the two apart by feel when no power is applied. Permanent magnet motors tend to "cog" as you twist the rotor with your fingers, while variable reluctance motors almost spin freely (although they may cog slightly because of residual magnetization in the rotor). You can also distinguish between the two varieties with an ohmmeter. Variable reluctance motors usually have three (sometimes four) windings, with a common return, while permanent magnet motors usually have two independent windings, with or without center taps. Center-tapped windings are used in unipolar permanent magnet motors.
Stepping motors come in a wide range of angular resolution. The coarsest motors typically turn 90 degrees per step, while high resolution permanent magnet motors are commonly able to handle 1.8 or even 0.72 degrees per step. With an appropriate controller, most permanent magnet and hybrid motors can be run in half-steps, and some controllers can handle smaller fractional steps or microsteps.
For both permanent magnet and variable reluctance stepping motors, if just one winding of the motor is energised, the rotor (under no load) will snap to a fixed angle and then hold that angle until the torque exceeds the holding torque of the motor, at which point, the rotor will turn, trying to hold at each successive equilibrium point.

Variable Reluctance Motors

Figure 1.1



If your motor has three windings, typically connected as shown in the schematic diagram in Figure 1.1, with one terminal common to all windings, it is most likely a variable reluctance stepping motor.

In use, the common wire typically goes to the positive supply and the windings are energized in sequence.
The cross section shown in Figure 1.1 is of 30 degree per step variable reluctance motor. The rotor in this motor has 4 teeth and the stator has 6 poles, with each winding wrapped around two opposite poles. With winding number 1 energised, the rotor teeth marked X are attracted to this winding's poles. If the current through winding 1 is turned off and winding 2 is turned on, the rotor will rotate 30 degrees clockwise so that the poles marked Y line up with the poles marked 2. An animated GIF of figure 1.1 is available.
To rotate this motor continuously, we just apply power to the 3 windings in sequence. Assuming positive logic, where a 1 means turning on the current through a motor winding, the following control sequence will spin the motor illustrated in Figure 1.1 clockwise 24 steps or 2 revolutions:

```
 Winding 1 10010010010010010010010 01
  Winding 2 01001001001001001001001 00
  Winding 3 00100100100100100100100 10
              time --->
```

The section of this tutorial on Mid-Level Control provides details on methods for generating such sequences of control signals, while the section on Control Circuits discusses the power switching circuitry needed to drive the motor windings from such control sequences.
There are also variable reluctance stepping motors with 4 and 5 windings, requiring 5 or 6 wires. The principle for driving these motors is the same as that for the three winding variety, but it becomes important to work out the correct order to energise the windings to make the motor step nicely.
The motor geometry illustrated in Figure 1.1, giving 30 degrees per step, uses the fewest number of rotor teeth and stator poles that performs satisfactorily. Using more motor poles and more rotor teeth allows construction of motors with smaller step angle. Toothed faces on each pole and a correspondingly finely toothed rotor allows for step angles as small as a few degrees.

Unipolar Motors

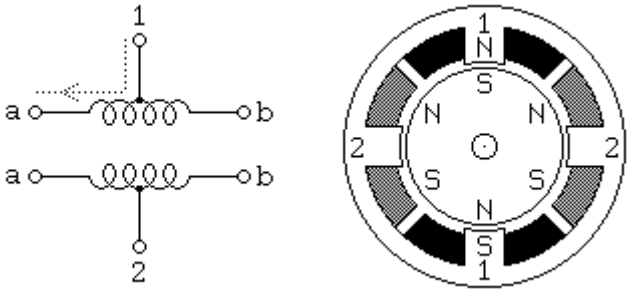


Figure 1.2

Unipolar stepping motors, both Permanent magnet and hybrid stepping motors with 5 or 6 wires are usually wired as shown in the schematic in Figure 1.2, with a center tap on each of two windings. In use, the center taps of the windings are typically wired to the positive supply, and the two ends of each winding are alternately grounded to reverse the direction of the field provided by that winding. An animated GIF of figure 1.2 is available.
The motor cross section shown in Figure 1.2 is of a 30 degree per step permanent magnet or hybrid motor -- the difference between these two motor types is not relevant at this level of abstraction. Motor winding number 1 is distributed between the top and bottom stator pole, while motor winding number 2 is distributed between the left and right motor poles. The rotor is a permanent magnet with 6 poles, 3 south and 3 north, arranged around its circumfrence.
For higher angular resolutions, the rotor must have proportionally more poles. The 30 degree per step motor in the figure is one of the most common permanent magnet motor designs, although 15 and 7.5 degree per step motors are widely available. Permanent magnet motors with resolutions as good as 1.8 degrees per step are made, and hybrid motors are routinely built with 3.6 and 1.8 degrees per step, with resolutions as fine as 0.72 degrees per step available.

As shown in the figure, the current flowing from the center tap of winding 1 to terminal a causes the top stator pole to be a north pole while the bottom stator pole is a south pole. This attracts the rotor into the position shown. If the power to winding 1 is removed and winding 2 is energised, the rotor will turn 30 degrees, or one step.
To rotate the motor continuously, we just apply power to the two windings in sequence. Assuming positive logic, where a 1 means turning on the current through a motor winding, the following two control sequences will spin the motor illustrated in Figure 1.2 clockwise 24 steps or 4 revolutions:

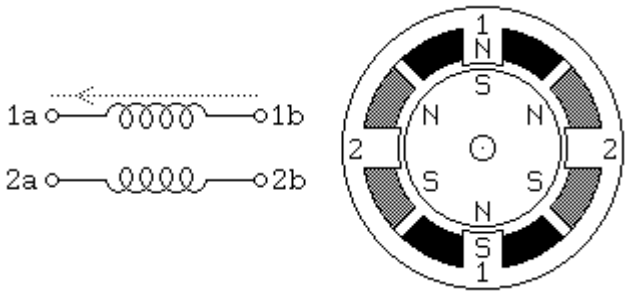
```
Winding 1a 100010001000100010001000
Winding 1b 001000100010001000100010
Winding 2a 010001000100010001000100
Winding 2b 000100010001000100010001
            time --->

Winding 1a 110011001100110011001100
Winding 1b 001100110011001100110011
Winding 2a 011001100110011001100110
Winding 2b 100110011001100110011001
            time --->
```

Note that the two halves of each winding are never energized at the same time. Both sequences shown above will rotate a permanent magnet one step at a time. The top sequence only powers one winding at a time, as illustrated in the figure above; thus, it uses less power. The bottom sequence involves powering two windings at a time and generally produces a torque about 1.4 times greater than the top sequence while using twice as much power.
The section of this tutorial on Mid-Level Control provides details on methods for generating such sequences of control signals, while the section on Control Circuits discusses the power switching circuitry needed to drive the motor windings from such control sequences.
The step positions produced by the two sequences above are not the same; as a result, combining the two sequences allows half stepping, with the motor stopping alternately at the positions indicated by one or the other sequence. The combined sequence is as follows:

```
Winding 1a 110000011100000111000001111
Winding 1b 000111000001110000011100000
Winding 2a 011100000111000001110000001
Winding 2b 000001110000011100000111000
            time --->
```

Bipolar Motors

Figure 1.3



Bipolar permanent magnet and hybrid motors are constructed with exactly the same mechanism as is used on unipolar motors, but the two windings are wired more simply, with no center taps. Thus, the motor itself is simpler but the drive circuitry needed to reverse the polarity of each pair of motor poles is more complex. The schematic in Figure 1.3 shows how such a motor is wired, while the motor cross section shown here is exactly the same as the cross section shown in Figure 1.2.

The drive circuitry for such a motor requires an *H-bridge* control circuit for each winding; these are discussed in more detail in the section on Control Circuits. Briefly, an H-bridge allows the polarity of the power applied to each end of each winding to be controlled independently. The control sequences for single stepping such a motor are shown below, using + and - symbols to indicate the polarity of the power applied to each motor terminal:

```
Terminal 1a +---+---+---+---  ++--++--++--++--
Terminal 1b --+---+---+---+-  --++--++--++--++
Terminal 2a -+---+---+---+--  -++--++--++--++-
Terminal 2b ---+---+---+---+  +--++--++--++--+
            time --->
```

Note that these sequences are identical to those for a unipolar permanent magnet motor, at an abstract level, and that above the level of the H-bridge power switching electronics, the control systems for the two types of motor can be identical.
Note that many full H-bridge driver chips have one control input to enable the output and another to control the direction. Given two such bridge chips, one per winding, the following control sequences will spin the motor identically to the control sequences given above:

```
Enable    1 1010101010101010  1111111111111111
Direction 1 1x0x1x0x1x0x1x0x  1100110011001100
Enable    2 0101010101010101  1111111111111111
Direction 2 x1x0x1x0x1x0x1x0  0110011001100110
            time --->
```

To distinguish a bipolar permanent magnet motor from other 4 wire motors, measure the resistances between the different terminals. It is worth noting that some permanent magnet stepping motors have 4 independent windings, organized as two sets of two. Within each set, if the two windings are wired in series, the result can be used as a high voltage bipolar motor. If they are wired in parallel, the result can be used as a low voltage bipolar motor. If they are wired in series with a center tap, the result can be used as a low voltage unipolar motor.

Bifilar Motors

Bifilar windings on a stepping motor are applied to the same rotor and stator geometry as a bipolar motor, but instead of winding each coil in the stator with a single wire, two wires are wound in parallel with each other. As a result, the motor has 8 wires, not four.
In practice, motors with bifilar windings are always powered as either unipolar or bipolar motors. Figure 1.4 shows the alternative connections to the windings of such a motor.

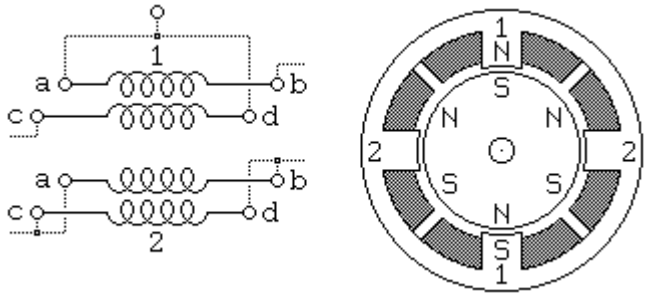


Figure 1.4

To use a bifilar motor as a unipolar motor, the two wires of each winding are connected in series and the point of connection is used as a center-tap. Winding 1 in Figure 1.4 is shown connected this way.
To use a bifilar motor as a bipolar motor, the two wires of each winding are connected either in parallel or in series. Winding 2 in Figure 1.4 is shown with a parallel connection; this allows low voltage high-current operation. Winding 1 in Figure 1.4 is shown with a series connection; if the

center tap is ignored, this allows operation at twice the voltage and half the current as would be used with the windings in parallel.
It should be noted that essentially all 6-wire motors sold for bipolar use are actually wound using bifilar windings, so that the external connection that serves as a center tap is actually connected as shown for winding 1 in Figure 1.4. Naturally, therefore, any unipolar motor may be used as a bipolar motor at twice the rated voltage and half the rated current as is given on the nameplate.
For those who salvage old motors, finding an 8-wire motor poses a challenge! Which of the 8 wires is which? It is not hard to figure this out using an ohm meter, an AC volt meter, and a low voltage AC source. First, use the ohm meter to identify the motor leads that are connected to each other through the motor windings. Then, connect a low-voltage AC source to one of these windings. The AC voltage should be below the advertised operating voltage of the motor; voltages under 1 volt are recommended. The geometry of the magnetic circuits of the motor guarantees that the two wires of a bifilar winding will be strongly coupled for AC signals, while there should be almost no coupling to the other two wires. Therefore, probing with an AC volt meter should disclose which of the other three windings is paired to the winding under power.

Multiphase Motors

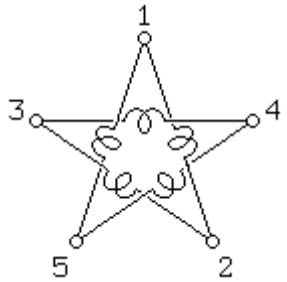


Figure 1.5

A less common class of permanent magnet stepping motor is wired with all windings of the motor in a cyclic series, with one tap between each pair of windings in the cycle. The most common designs in this category use 3-phase and 5-phase wiring. The control requires 1/2 of an H-bridge for each motor terminal, but these motors can provide more torque from a given package size because all or all but one of the motor windings are energised at every point in the drive cycle. Some 5-phase motors have high resolutions on the order of 0.72 degrees per step (500 steps per revolution).
With a 5-phase motor, there are 10 steps per repeat in the stepping cycle, as shown below:

```
Terminal 1  ++++-----++++------++
Terminal 2  --+++++-----++++++---
Terminal 3  +-----++++------++++
Terminal 4  +++++-----++++------
Terminal 5  ----+++++-----++++++-
            time --->
```

Here, as in the bipolar case, each terminal is shown as being either connected to the positive or negative bus of the motor power system. Note that, at each step, only one terminal changes polarity. This change removes the power from one winding attached to that terminal (because both terminals of the winding in question are of the same polarity) and applies power to one winding that was previously idle. Given the motor geometry suggested by Figure 1.5, this control sequence will drive the motor through two revolutions.
To distinguish a 5-phase motor from other motors with 5 leads, note that, if the resistance between two consecutive terminals of the 5-phase motor is R, the resistance between non-consecutive terminals will be 1.5R.
Note that some 5-phase motors have 5 separate motor windings, with a total of 10 leads. These can be connected in the star configuration shown above, using 5 half-bridge driver circuits, or each winding can be driven by its own full-bridge. While the theoretical component count of half-bridge drivers is lower, the availability of integrated full-bridge chips may make the latter approach preferable.

2. Stepping Motor Physics

Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

Introduction

In any presentation covering the quantitative physics of a class of systems, it is important to beware of the units of measurement used! In this presentation of stepping motor physics, we will assume standard physical units:

	English	**CGS**	**MKS**
MASS	slug	gram	kilogram
FORCE	pound	dyne	newton
DISTANCE	foot	centimeter	meter
TIME	second	second	second
ANGLE	radian	radian	radian

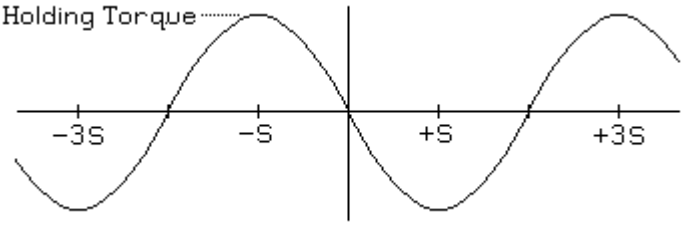
A force of one pound will accelerate a mass of one slug at one foot per second squared. The same relationship holds between the force, mass, time and distance units of the other measurement systems. Most people prefer to measure angles in degrees, and the common engineering practice of specifying mass in pounds or force in kilograms will not yield correct results in the formulas given here! Care must be taken to convert such irregular units to one of the standard systems outlined above before applying the formulas given here!

Statics

For a motor that turns S radians per step, the plot of torque versus angular position for the rotor relative to some initial equilibrium position will generally approximate a sinusoid. The actual shape of the curve depends on the pole geometry of both rotor and stator, and neither this curve nor the geometry information is given in the motor data sheets I've seen! For permanent magnet and hybrid motors, the actual curve usually looks sinusoidal, but looks can be misleading. For variable reluctance motors, the curve rarely even looks sinusoidal; trapezoidal and even assymetrical sawtooth curves are not uncommon.

For a three-winding variable reluctance or permanent magnet motors with S radians per step, the period of the torque versus position curve will be 3S; for a 5-phase permanent magnet motor, the period will be 5S. For a two-winding permanent magnet or hybrid motor, the most common type, the period will be 4S, as illustrated in Figure 2.1:

Figure 2.1

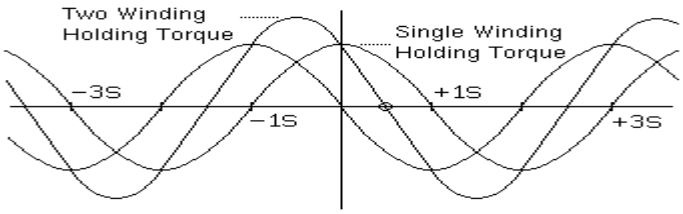


Again, for an ideal 2 winding permanent magnet motor, this can be mathematically expressed as: $T = -h \sin(((\pi/2) / S) \theta)$ Where: T -- torque
h -- holding torque
S -- step angle, in radians
θ= shaft angle, in radians But remember, subtle departures from the ideal sinusoid described here are very common.
The *single-winding holding torque* of a stepping motor is the peak value of the torque versus position curve when the maximum allowed current is flowing through one motor winding. If you attempt to apply a torque greater than this to the motor rotor while maintaining power to one winding, it will rotate freely.
It is sometimes useful to distinguish between the *electrical shaft angle* and the *mechanical shaft angle*. In the mechanical frame of reference, 2π radians is defined as one full revolution. In the electrical frame of reference, a revolution is defined as one period of the torque versus shaft angle curve. Throughout this tutorial, θ refers to the mechanical shaft angle, and $((\pi/2)/S)\theta$ gives the electrical angle for a motor with 4 steps per cycle of the torque curve.
Assuming that the torque versus angular position curve is a good approximation of a sinusoid, as long as the torque remains below the holding torque of the motor, the rotor will remain within 1/4 period of the equilibrium position. For a two-winding permanent magnet or hybrid motor, this means the rotor will remain within one step of the equilibrium position.
With no power to any of the motor windings, the torque does not always fall to zero! In variable reluctance stepping motors, residual magnetization in the magnetic circuits of the motor may lead to a small residual torque, and in permanent magnet and hybrid stepping motors, the combination of pole geometry and the permanently magnetized rotor may lead to significant torque with no applied power.
The residual torque in a permanent magnet or hybrid stepping motor is frequently referred to as the *cogging torque* or *detent torque* of the motor because a naive observer will frequently guess that there is a detent mechanism of some kind inside the motor. The most common motor designs yield a detent torque that varies sinusoidally with rotor angle, with an equilibrium position at every step and an amplitude of roughly 10% of the rated holding torque of the motor, but a quick survey of motors from one manufacturer (Phytron) shows values as high as 23% for one very small motor to a low of 2.6% for one mid-sized motor.

Half-Stepping and Microstepping

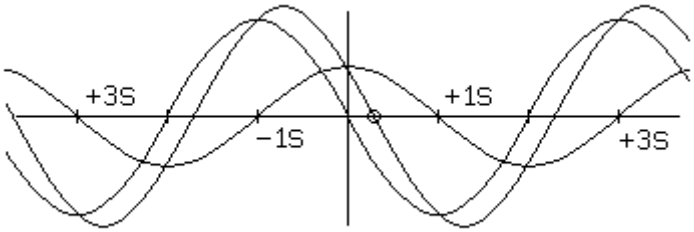
So long as no part of the magnetic circuit saturates, powering two motor windings simultaneously will produce a torque versus position curve that is the sum of the torque versus position curves for the two motor windings taken in isolation. For a two-winding permanent magnet or hybrid motor, the two curves will be S radians out of phase, and if the currents in the two windings are equal, the peaks and valleys of the sum will be displaced S/2 radians from the peaks of the original curves, as shown in Figure 2.2:

Figure 2.2



This is the basis of *half-stepping*. The *two-winding holding torque* is the peak of the composite torque curve when two windings are carrying their maximum rated current. For common two-winding permanent magnet or hybrid stepping motors, the two-winding holding torque will be: $h_2 = 2^{0.5} h_1$ where: h_1 -- single-winding holding torque
h_2 -- two-winding holding torque This assumes that no part of the magnetic circuit is saturated and that the torque versus position curve for each winding is an ideal sinusoid.
Most permanent-magnet and variable-reluctance stepping motor data sheets quote the two-winding holding torque and not the single-winding figure; in part, this is because it is larger, and in part, it is because the most common full-step controllers always apply power to two windings at once.
If any part of the motor's magnetic circuits is saturated, the two torque curves will not add linearly. As a result, the composite torque will be less than the sum of the component torques and the equilibrium position of the composite may not be exactly S/2 radians from the equilibria of the original.
Microstepping allows even smaller steps by using different currents through the two motor windings, as shown in Figure 2.3:

Figure 2.3



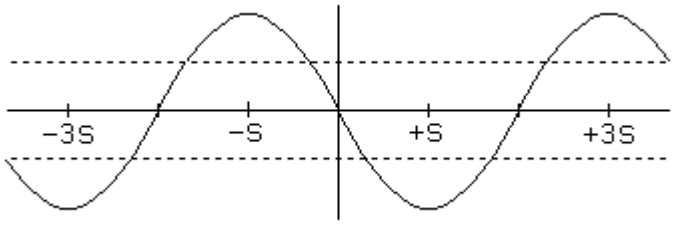
For a two-winding variable reluctance or permanent magnet motor, assuming nonsaturating magnetic circuits, and assuming perfectly sinusoidal torque versus position curves for each motor winding, the following formula gives the key characteristics of the composite torque curve: $h = (a^2 + b^2)^{0.5}$
$x = (S / (\pi/2)) \arctan(b / a)$ Where: a -- torque applied by winding with equilibrium at 0 radians.

b -- torque applied by winding with equilibrium at S radians.
h -- holding torque of composite.
x -- equilibrium position, in radians.
S -- step angle, in radians. In the absence of saturation, the torques a and b are directly proportional to the currents through the corresponding windings. It is quite common to work with normalized currents and torques, so that the single-winding holding torque or the maximum current allowed in one motor winding is 1.0.

Friction and the Dead Zone

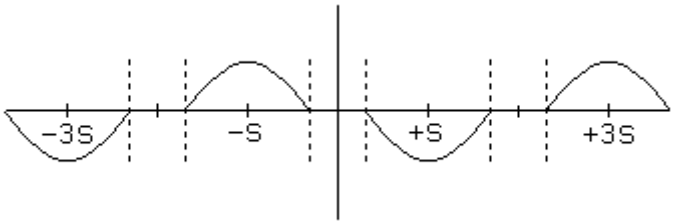
The torque versus position curve shown in Figure 2.1 does not take into account the torque the motor must exert to overcome friction! Note that frictional forces may be divided into two large categories, static or sliding friction, which requires a constant torque to overcome, regardless of velocity, and dynamic friction or viscous drag, which offers a resistance that varies with velocity. Here, we are concerned with the impact of static friction. Suppose the torque needed to overcome the static friction on the driven system is 1/2 the peak torque of the motor, as shown in Figure 2.4.

Figure 2.4



The dotted lines in Figure 2.4 show the torque needed to overcome friction; only that part of the torque curve outside the dotted lines is available to move the rotor. The curve showing the available torque as a function of shaft angle is the difference between these curves, as shown in Figure 2.5:

Figure 2.5

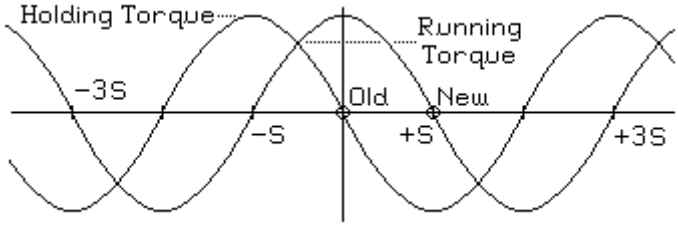


Note that the consequences of static friction are twofold. First, the total torque available to move the load is reduced, and second, there is a *dead zone* about each of the equilibria of the ideal motor. If the motor rotor is positioned anywhere within the dead zone for the current equilibrium position, the frictional torque will exceed the torque applied by the motor windings, and the rotor will not move. Assuming an ideal sinusoidal torque versus position curve in the absence of friction, the angular width of these dead zones will be: $d = 2 (S / (\pi/2)) \arcsin(f / h) = (S / (\pi/4)) \arcsin(f / h)$ where: *d* -- width of dead zone, in radians
S -- step angle, in radians
f -- torque needed to overcome static friction
h -- holding torque
The important thing to note about the dead zone is that it limits the ultimate positioning accuracy! For the example, where the static friction is 1/2 the peak torque, a 90° per step motor will have dead-zones 60° wide! That means that successive steps may be as large as 150° and as small as 30°, depending on where in the dead zone the rotor stops after each step!
The presence of a dead zone has a significant impact on the utility of microstepping! If the dead zone is x° wide, then microstepping with a step size smaller than x° may not move the rotor at all. Thus, for systems intended to use high resolution microstepping, it is very important to minimize static friction.

Dynamics

Each time you step the motor, you electronically move the equilibrium position S radians. This moves the entire curve illustrated in Figure 2.1 a distance of S radians, as shown in Figure 2.6:

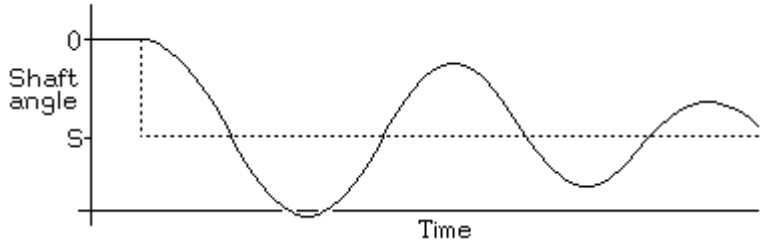
Figure 2.6



The first thing to note about the process of taking one step is that the maximum available torque is at a minimum when the rotor is halfway from one step to the next. This minimum determines the *running torque*, the maximum torque the motor can drive as it steps slowly forward. For common two-winding permanent magnet motors with ideal sinusoidal torque versus position curves and holding torque h, this will be $h/(2^{0.5})$. If the motor is stepped by powering two windings at a time, the

running torque of an ideal two-winding permanent magnet motor will be the same as the single-winding holding torque.
It shoud be noted that at higher stepping speeds, the running torque is sometimes defined as the *pull-out torque*. That is, it is the maximum frictional torque the motor can overcome on a rotating load before the load is pulled out of step by the friction. Some motor data sheets define a second torque figure, the *pull-in torque*. This is the maximum frictional torque that the motor can overcome to accelerate a stopped load to synchronous speed. The pull-in torques documented on stepping motor data sheets are of questionable value because the pull-in torque depends on the moment of inertia of the load used when they were measured, and few motor data sheets document this!
In practice, there is always some friction, so after the equilibrium position moves one step, the rotor is likely to oscillate briefly about the new equilibrium position. The resulting trajectory may resemble the one shown in Figure 2.7:

Figure 2.7



Here, the trajectory of the equilibrium position is shown as a dotted line, while the solid curve shows the trajectory of the motor rotor.

Resonance

The resonant frequency of the motor rotor depends on the amplitude of the oscillation; but as the amplitude decreases, the resonant frequency rises to a well-defined small-amplitude frequency. This frequency depends on the step angle and on the ratio of the holding torque to the moment of inertia of the rotor. Either a higher torque or a lower moment will increase the frequency!
Formally, the small-amplitude resonance can be computed as follows: First, recall Newton's law for angular acceleration: $T = \mu A$ Where: T -- torque applied to rotor
μ -- moment of inertia of rotor and load
A -- angular acceleration, in radians per second per second We assume that, for small amplitudes, the torque on the rotor can be approximated as a linear function of the displacement from the equilibrium position. Therefore, Hooke's law applies: $T = -k\Theta$ where: *k* -- the "spring constant" of the system, in torque units per radian
Θ-- angular position of rotor, in radians We can equate the two formulas for the torque to get: $\mu A = -k\Theta$ Note that acceleration is the second derivitive of position with respect to time: $A = d^2\Theta/dt^2$ so we can rewrite this the above in differential equation form: $d^2\Theta/dt^2 = -(k/\mu)\Theta$ To solve this, recall that, for: $f(t) = a \sin bt$ The derivitives are: $df(t)/dt = ab \cos bt$
$d^2f(t)/dt^2 = -ab^2 \sin bt = -b^2 f(t)$ Note that, throughout this discussion, we assumed that the rotor is resonating. Therefore, it has an equation of motion something like: $\Theta = a \sin(2\pi f t)$
a = angular amplitude of resonance
f = resonant frequency This is an admissable solution to the above differential equation if we agree that: $b = 2\pi f$
$b^2 = k/\mu$ Solving for the resonant frequency *f* as a function of *k* and μ, we get: $f = (k/\mu)^{0.5} / 2\pi$ It is crucial to note that it is the moment of inertia of the rotor plus any coupled load that matters. The moment of the rotor, in isolation, is irrelevant! Some motor data sheets include information on resonance, but if any load is coupled to the rotor, the resonant frequency will change!
In practice, this oscillation can cause significant problems when the stepping rate is anywhere near a resonant frequency of the system; the result frequently appears as random and uncontrollable motion.

Resonance and the Ideal Motor

Up to this point, we have dealt only with the small-angle spring constant *k* for the system. This can be measured experimentally, but if the motor's torque versus position curve is sinusoidal, it is also a simple function of the motor's holding torque. Recall that: $T = -h \sin(((\pi/2)/S) \theta)$ The small angle spring constant *k* is the negative derivitive of T at the origin. $k = -dT / d\theta = - (- h ((\pi/2)/S) \cos(0)) = (\pi/2)(h / S)$ Substituting this into the formula for frequency, we get: $f = ((\pi/2)(h / S) / \mu)^{0.5} / 2\pi = (h / (8\pi \mu S))^{0.5}$ Given that the holding torque and resonant frequency of the system are easily measured, the easiest way to determine the moment of inertia of the moving parts in a system driven by a stepping motor is indirectly from the above relationship! $\mu = h / (8\pi f^2 S)$ For practical purposes, it is usually not the torque or the moment of inertia that matters, but rather, the maximum sustainable acceleration that matters! Conveniently, this is a simple function of the resonant frequency! Starting with the Newton's law for angular acceleration: $A = T / \mu$ We can substitute the above formula for the moment of inertia as a function of resonant frequency, and then substitute the maximum sustainable running torque as a function of the holding torque to get: $A = (h / (2^{0.5})) / (h / (8\pi f^2 S)) = 8\pi S f^2 / (2^{0.5})$ Measuring acceleration in steps per second squared instead of in radians per second squared, this simplifies to: $A_{steps} = A / S = 8\pi f^2 / (2^{0.5})$ Thus, for an ideal motor with a sinusoidal torque versus rotor position function, the maximum acceleration in steps per second squared is a trivial function of the resonant frequency of the motor and rigidly coupled load!

For a two-winding permanent-magnet or variable-reluctance motor, with an ideal sinusoidal torque-versus-position characteristic, the two-winding holding torque is a simple function of the single-winding holding torque: $h_2 = 2^{0.5} h_1$ Where: h_1 -- single-winding holding torque h_2 -- two-winding holding torque Substituting this into the formula for resonant frequency, we can find the ratios of the resonant frequencies in these two operating modes: $f_1 = (h_1 / ...)^{0.5}$ $f_2 = (h_2 / ...)^{0.5} = (2^{0.5} h_1 / ...)^{0.5} = 2^{0.25} (h_1 / ...)^{0.5} = 2^{0.25} f_1 = 1.189... f_1$ This relationship only holds if the torque provided by the motor does not vary appreciably as the stepping rate varies between these two frequencies.

In general, as will be discussed later, the available torque will tend to remain relatively constant up until some cutoff stepping rate, and then it will fall. Therefore, this relationship only holds if the resonant frequencies are below this cutoff stepping rate. At stepping rates above the cutoff rate, the two frequencies will be closer to each other!

Living with Resonance

If a rigidly mounted stepping motor is rigidly coupled to a frictionless load and then stepped at a frequency near the resonant frequency, energy will be pumped into the resonant system, and the result of this is that the motor will literally lose control. There are three basic ways to deal with this problem:

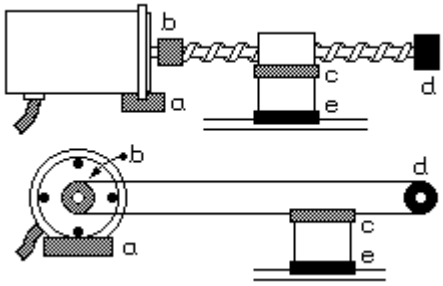
Controlling resonance in the mechanism

Use of elastomeric motor mounts or elastomeric couplings between motor and load can drain energy out of the resonant system, preventing energy from accumulating to the extent that it allows the motor rotor to escape from control.

Or, viscous damping can be used. Here, the damping will not only draw energy out of the resonant modes of the system, but it will also subtract from the total torque available at higher speeds. Magnetic eddy current damping is equivalent to viscous damping for our purposes.

Figure 2.8 illustrates the use of elastomeric couplings and viscous damping in two typical stepping motor applications, one using a lead screw to drive a load, and the other using a tendon drive:

Figure 2.8



In Figure 2.8, elastomeric moter mounts are shown at a and elastomeric couplings between the motor and load are shown at b and c. The end bearing for the lead screw or tendon, at d, offers an opportunity for viscous damping, as do the ways on which the load slides, at e. Even the friction found in sealed ballbearings or teflon on steel ways can provide enough damping to prevent resonance problems.

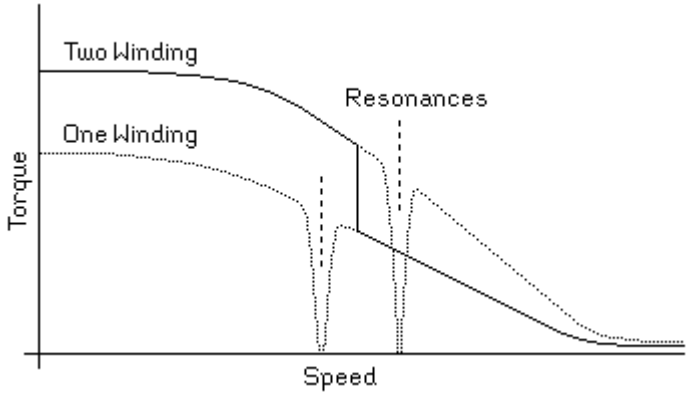
Controlling resonance in the low-level drive circuitry

A resonating motor rotor will induce an alternating current voltage in the motor windings. If some motor winding is not currently being driven, shorting this winding will impose a drag on the motor rotor that is exactly equivalent to using a magnetic eddy current damper.
If some motor winding is currently being driven, the AC voltage induced by the resonance will tend to modulate the current through the winding. Clamping the motor current with an external inductor will counteract the resonance. Schemes based on this idea are incorporated into some of the drive circuits illustrated in later sections of this tutorial.

Controlling resonance in the high-level control system

The high level control system can avoid driving the motor at known resonant frequencies, accelerating and decelerating through these frequencies and never attempting sustained rotation at these speeds.
Recall that the resonant frequency of a motor in half-stepped mode will vary by up to 20% from one half-step to the next. As a result, half-stepping pumps energy into the resonant system less efficiently than full stepping. Furthermore, when operating near these resonant frequencies, the motor control system may preferentially use only the two-winding half steps when operating near the single-winding resonant frequency, and only the single-winding half steps when operating near the two-winding resonant frequency. Figure 2.9 illustrates this:

Figure 2.9

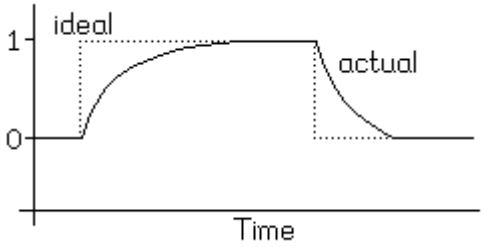


The darkened curve in Figure 2.9 shows the operating torque achieved by a simple control scheme that delivers useful torque over a wide range of speeds despite the fact that the available torque drops to zero at each resonance in the system. This solution is particularly effective if the resonant frequencies are sharply defined and well separated. This will be the case in minimally damped systems operating well below the cutoff speed defined in the next section.

Torque versus Speed

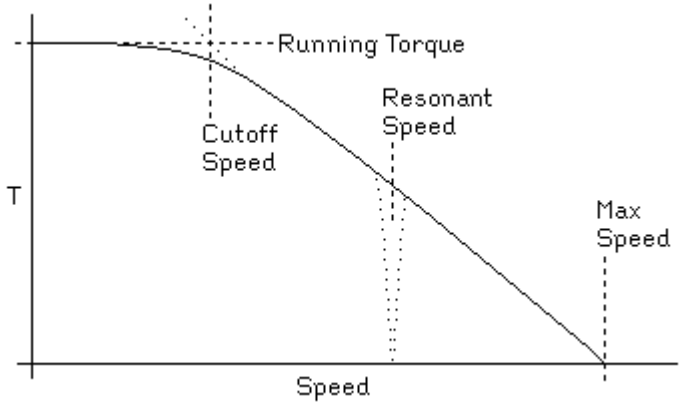
An important consideration in designing high-speed stepping motor controllers is the effect of the inductance of the motor windings. As with the torque versus angular position information, this is frequently poorly documented in motor data sheets, and indeed, for variable reluctance stepping motors, it is not a constant! The inductance of the motor winding determines the rise and fall time of the current through the windings. While we might hope for a square-wave plot of current versus time, the inductance forces an exponential, as illustrated in Figure 2.10:

Figure 2.10



The details of the current-versus-time function through each winding depend as much on the drive circuitry as they do on the motor itself! It is quite common for the time constants of these exponentials to differ. The rise time is determined by the drive voltage and drive circuitry, while the fall time depends on the circuitry used to dissipate the stored energy in the motor winding.
At low stepping rates, the rise and fall times of the current through the motor windings has little effect on the motor's performance, but at higher speeds, the effect of the inductance of the motor windings is to reduce the available torque, as shown in Figure 2.11:

Figure 2.11



The motor's *maximum speed* is defined as the speed at which the available torque falls to zero. Measuring maximum speed can be difficult when there are resonance problems, because these cause the torque to drop to zero prematurely. The *cutoff speed* is the speed above which the torque begins to fall. When the motor is operating below its cutoff speed, the rise and fall times of the current through the motor windings occupy an insignificant fraction of each step, while at the cutoff speed, the step duration is comparable to the sum of the rise and fall times. Note that a sharp cutoff is rare, and therefore, statements of a motor's cutoff speed are, of necessity, approximate.
The details of the torque versus speed relationship depend on the details of the rise and fall times in the motor windings, and these depend on the motor control system as well as the motor. Therefore, the cutoff speed and maximum speed for any particular motor depend, in part, on the control system! The torque versus speed curves published in motor data sheets occasionally come with documentation of the motor controller used to obtain that curve, but this is far from universal practice!
Similarly, the resonant speed depends on the moment of inertia of the entire rotating system, not just the motor rotor, and the extent to which the torque drops at resonance depends on the presence of mechanical damping and on the nature of the control system. Some published torque versus speed curves show very clear resonances without documenting the moment of inertia of the hardware that may have been attached to the motor shaft in order to make torque measurements.

The torque versus speed curve shown in Figure 2.11 is typical of the simplest of control systems. More complex control systems sometimes introduce electronic resonances that act to increase the available torque above the motor's low-speed torque. A common result of this is a peak in the available torque near the cutoff speed.

Electromagnetic Issues

In a permanent magnet or hybrid stepping motor, the magnetic field of the motor rotor changes with changes in shaft angle. The result of this is that turning the motor rotor induces an AC voltage in each motor winding. This is referred to as the *counter EMF* because the voltage induced in each motor winding is always in phase with and counter to the ideal waveform required to turn the motor in the same direction. Both the frequency and amplitude of the counter EMF increase with rotor speed, and therefore, counter EMF contributes to the decline in torque with increased stepping rate.
Variable reluctance stepping motors also induce counter EMF! This is because, as the stator winding pulls a tooth of the rotor towards its equilibrium position, the reluctance of the magnetic circuit declines. This decline increases the inductance of the stator winding, and this change in inductance demands a decrease in the current through the winding in order to conserve energy. This decrease is evidenced as a counter EMF.
The reactance (inductance and resistance) of the motor windings limits the current flowing through them. Thus, by ohms law, increasing the voltage will increase the current, and therefore increase the available torque. The increased voltage also serves to overcome the counter EMF induced in the motor windings, but the voltage cannot be increased arbitrarily! Thermal, magnetic and electronic considerations all serve to limit the useful torque that a motor can produce.
The heat given off by the motor windings is due to both simple resistive losses, eddy current losses, and hysteresis losses. If this heat is not conducted away from the motor adequately, the motor windings will overheat. The simplest failure this can cause is insulation breakdown, but it can also heat a permanent magnet rotor to above its curie temperature, the temperature at which permanent magnets lose their magnetization. This is a particular risk with many modern high strength magnetic alloys.
Even if the motor is attached to an adequate heat sink, increased drive voltage will not necessarily lead to increased torque. Most motors are designed so that, with the rated current flowing through the windings, the magnetic circuits of the motor are near saturation. Increased current will not lead to an appreciably increased magnetic field in such a motor!
Given a drive system that limits the current through each motor winding to the rated maximum for that winding, but uses high voltages to achieve a higher cutoff torque and higher torques above cutoff, there are other limits that come into play. At high speeds, the motor windings must, of necessity, carry high frequency AC signals. This leads to eddy current losses in the magnetic circuits of the motor, and it leads to skin effect losses in the motor windings.
Motors designed for very high speed running should, therefore, have magnetic structures using very thin laminations or even nonconductive ferrite materials, and they should have small gauge wire in their windings to minimize skin effect losses. Common high torque motors have large-gauge motor windings and coarse core laminations, and at high speeds, such motors can easily overheat and should therefore be derated accordingly for high speed running!
It is also worth noting that the best way to demagnetize something is to expose it to a high frequency-high amplitude magnetic field. Running the control system to spin the rotor at high speed when the rotor is actually stalled, or spinning the rotor at high speed against a control system trying to hold the rotor in a fixed position will both expose the rotor to a high amplitude high-frequency field. If such operating conditions are common, particularly if the motor is run near the curie temperature of the permanent magnets, demagnetization is a serious risk and the field strengths (and expected torques) should be reduced accordingly!

3. Basic Stepping Motor Control Circuits

Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

Introduction

This section of the stepper tutorial deals with the basic final stage drive circuitry for stepping motors. This circuitry is centered on a single issue, switching the current in each motor winding on and off, and controlling its direction. The circuitry discussed in this section is connected directly to the motor windings and the motor power supply, and this circuitry is controlled by a digital system that determines when the switches are turned on or off.

This section covers all types of motors, from the elementary circuitry needed to control a variable reluctance motor, to the H-bridge circuitry needed to control a bipolar permanent magnet motor. Each class of drive circuit is illustrated with practical examples, but these examples are not intended as an exhaustive catalog of the commercially available control circuits, nor is the information given here intended to substitute for the information found on the manufacturer's component data sheets for the parts mentioned.

This section only covers the most elementary control circuitry for each class of motor. All of these circuits assume that the motor power supply provides a drive voltage no greater than the motor's rated voltage, and this significantly limits motor performance. The next section, on current limited drive circuitry, covers practical high-performance drive circuits.

Variable Reluctance Motors

Typical controllers for variable reluctance stepping motors are variations on the outline shown in Figure 3.1:

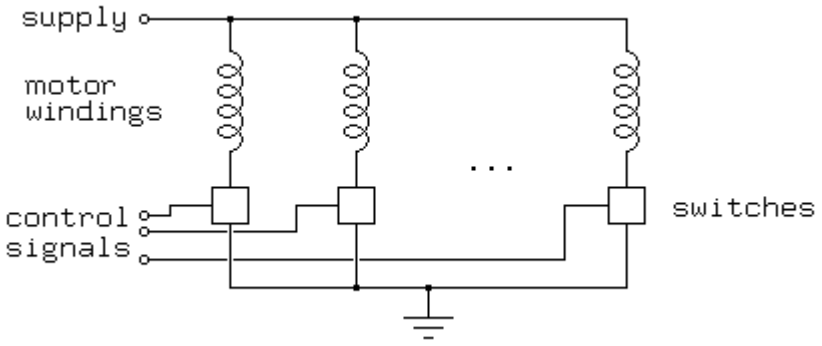
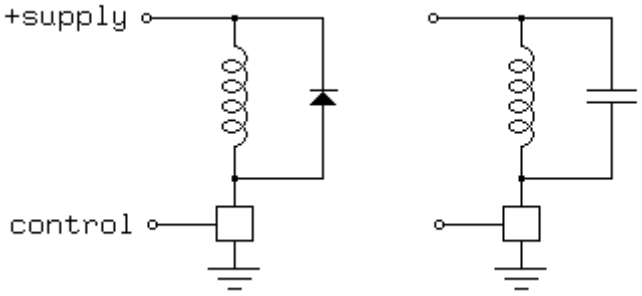


Figure 3.1

In Figure 3.1, boxes are used to represent switches; a control unit, not shown, is responsible for providing the control signals to open and close the switches at the appropriate times in order to spin

the motors. In many cases, the control unit will be a computer or programmable interface controller, with software directly generating the outputs needed to control the switches, but in other cases, additional control circuitry is introduced, sometimes gratuitously!
Motor windings, solenoids and similar devices are all inductive loads. As such, the current through the motor winding cannot be turned on or off instantaneously without involving infinite voltages! When the switch controlling a motor winding is closed, allowing current to flow, the result of this is a slow rise in current. When the switch controlling a motor winding is opened, the result of this is a voltage spike that can seriously damage the switch unless care is taken to deal with it appropriately. There are two basic ways of dealing with this voltage spike. One is to bridge the motor winding with a diode, and the other is to bridge the motor winding with a capacitor. Figure 3.2 illustrates both approaches:

Figure 3.2

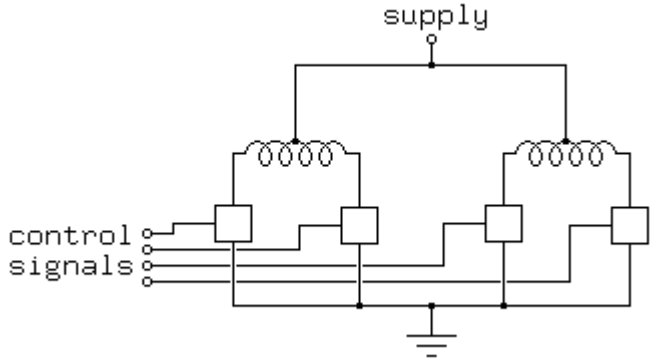


The diode shown in Figure 3.2 must be able to conduct the full current through the motor winding, but it will only conduct briefly each time the switch is turned off, as the current through the winding decays. If relatively slow diodes such as the common 1N400X family are used together with a fast switch, it may be necessary to add a small capacitor in parallel with the diode.
The capacitor shown in Figure 3.2 poses more complex design problems! When the switch is closed, the capacitor will discharge through the switch to ground, and the switch must be able to handle this brief spike of discharge current. A resistor in series with the capacitor or in series with the power supply will limit this current. When the switch is opened, the stored energy in the motor winding will charge the capacitor up to a voltage significantly above the supply voltage, and the switch must be able to tolerate this voltage. To solve for the size of the capacitor, we equate the two formulas for the stored energy in a resonant circuit: $P = C V^2 / 2$
$P = L I^2 / 2$ Where: P -- stored energy, in watt seconds or coulomb volts
C -- capacity, in farads
V -- voltage across capacitor
L -- inductance of motor winding, in henrys
I -- current through motor winding Solving for the minimum size of capacitor required to prevent overvoltage on the switch is fairly easy: $C > L I^2 / (V_b - V_s)^2$ Where: V_b -- the breakdown voltage of the switch
V_s -- the supply voltage Variable reluctance motors have variable inductance that depends on the shaft angle. Therefore, worst-case design must be used to select the capacitor. Furthermore, motor inductances are frequently poorly documented, if at all.
The capacitor and motor winding, in combination, form a resonant circuit. If the control system drives the motor at frequencies near the resonant frequency of this circuit, the motor current through the motor windings, and therefore, the torque exerted by the motor, will be quite different from the steady state torque at the nominal operating voltage! The resonant frequency is: $f = 1 / (2\pi (L C)^{0.5})$ Again, the electrical resonant frequency for a variable reluctance motor will depend on shaft angle! When a variable reluctance motors is operated with the exciting pulses near resonance, the oscillating current in the motor winding will lead to a magnetic field that goes to zero at twice the resonant frequency, and this can severely reduce the available torque!

Unipolar Permanent Magnet and Hybrid Motors

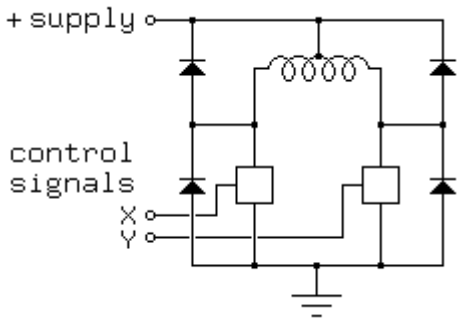
Typical controllers for unipolar stepping motors are variations on the outline shown in Figure 3.3:

Figure 3.3



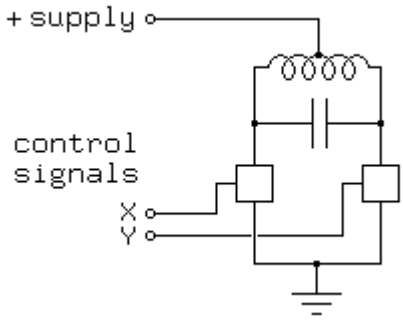
In Figure 3.3, as in Figure 3.1, boxes are used to represent switches; a control unit, not shown, is responsible for providing the control signals to open and close the switches at the appropriate times in order to spin the motors. The control unit is commonly a computer or programmable interface controller, with software directly generating the outputs needed to control the switches.
As with drive circuitry for variable reluctance motors, we must deal with the inductive kick produced when each of these switches is turned off. Again, we may shunt the inductive kick using diodes, but now, 4 diodes are required, as shown in Figure 3.4:

Figure 3.4



The extra diodes are required because the motor winding is not two independent inductors, it is a single center-tapped inductor with the center tap at a fixed voltage. This acts as an autotransformer! When one end of the motor winding is pulled down, the other end will fly up, and visa versa. When a switch opens, the inductive kickback will drive that end of the motor winding to the positive supply, where it is clamped by the diode. The opposite end will fly downward, and if it was not floating at the supply voltage at the time, it will fall below ground, reversing the voltage across the switch at that end. Some switches are immune to such reversals, but others can be seriously damaged.
A capacitor may also be used to limit the kickback voltage, as shown in Figure 3.5:

Figure 3.5



The rules for sizing the capacitor shown in Figure 3.5 are the same as the rules for sizing the capacitor shown in Figure 3.2, but the effect of resonance is quite different! With a permanent magnet motor, if the capacitor is driven at or near the resonant frequency, the torque will increase to as much as twice the low-speed torque! The resulting torque versus speed curve may be quite complex, as illustrated in Figure 3.6:

Figure 3.6

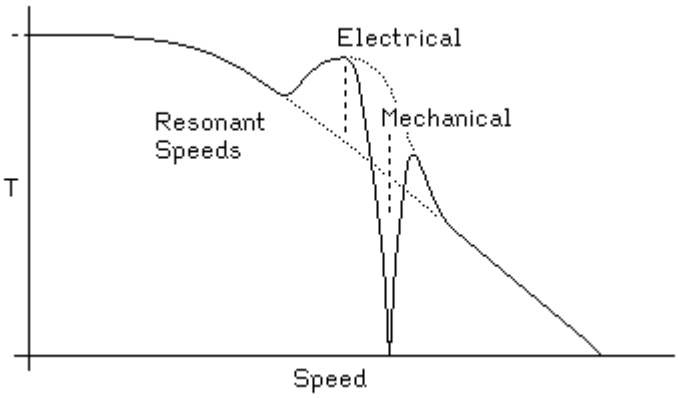
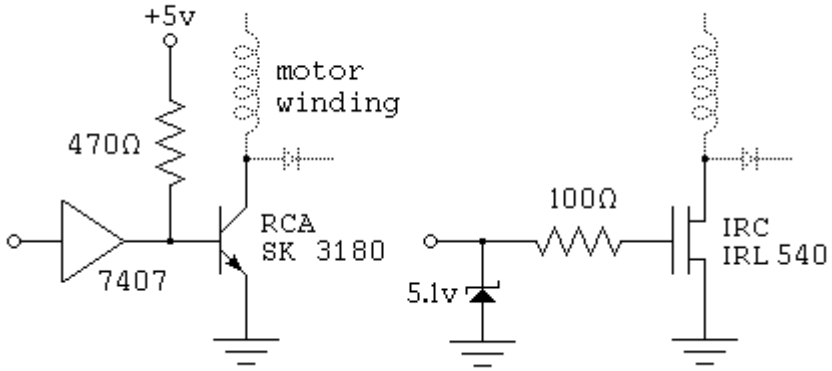


Figure 3.6 shows a peak in the available torque at the electrical resonant frequency, and a valley at the mechanical resonant frequency. If the electrical resonant frequency is placed appropriately above what would have been the cutoff speed for the motor using a diode-based driver, the effect can be a considerable increase in the effective cutoff speed.
The mechanical resonant frequency depends on the torque, so if the mechanical resonant frequency is anywhere near the electrical resonance, it will be shifted by the electrical resonance! Furthermore, the width of the mechanical resonance depends on the local slope of the torque versus speed curve; if the torque drops with speed, the mechanical resonance will be sharper, while if the torque climbs with speed, it will be broader or even split into multiple resonant frequencies.

Practical Unipolar and Variable Reluctance Drivers

In the above circuits, the details of the necessary switches have been deliberately ignored. Any switching technology, from toggle switches to power MOSFETS will work! Figure 3.7 contains some suggestions for implementing each switch, with a motor winding and protection diode included for orientation purposes:

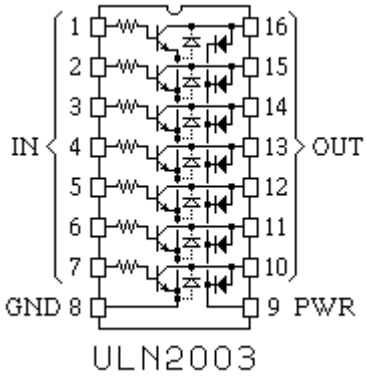
Figure 3.7



Each of the switches shown in Figure 3.7 is compatible with a TTL input. The 5 volt supply used for the logic, including the 7407 open-collector driver used in the figure, should be well regulated. The motor power, typically between 5 and 24 volts, needs only minimal regulation. It is worth noting that these power switching circuits are appropriate for driving solenoids, DC motors and other inductive loads as well as for driving stepping motors.
The SK3180 transistor shown in Figure 3.7 is a power darlington with a current gain over 1000; thus, the 10 milliamps flowing through the 470 ohm bias resistor is more than enough to allow the transistor to switch a few amps current through the motor winding. The 7407 buffer used to drive the darlington may be replaced with any high-voltage open collector chip that can sink at least 10 milliamps. In the event that the transistor fails, the high-voltage open collector driver serves to protects the rest of the logic circuitry from the motor power supply.
The IRC IRL540 shown in Figure 3.7 is a power field effect transistor. This can handle currents of up to about 20 amps, and it breaks down nondestructively at 100 volts; as a result, this chip can

absorb inductive spikes without protection diodes if it is attached to a large enough heat sink. This transistor has a very fast switching time, so the protection diodes must be comparably fast or bypassed by small capacitors. This is particularly essential with the diodes used to protect the transistor against reverse bias! In the event that the transistor fails, the zener diode and 100 ohm resistor protect the TTL circuitry. The 100 ohm resistor also acts to somewhat slow the switching times on the transistor.
For applications where each motor winding draws under 500 milliamps, the ULN200x family of darlington arrays from Allegro Microsystems, also available as the DS200x from National Semiconductor and as the Motorola MC1413 darlington array will drive multiple motor windings or other inductive loads directly from logic inputs. Figure 3.8 shows the pinout of the widely available ULN2003 chip, an array of 7 darlington transistors with TTL compatible inputs:

Figure 3.8

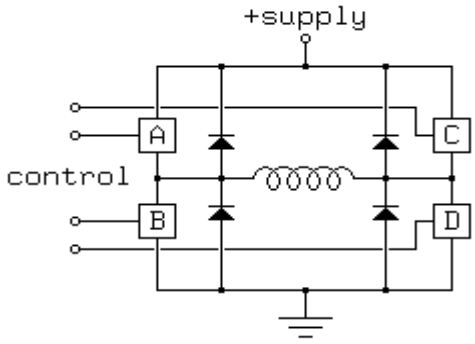


The base resistor on each darlington transistor is matched to standard bipolar TTL outputs. Each NPN darlington is wired with its emitter connected to pin 8, intended as a ground pin, Each transistor in this package is protected by two diodes, one shorting the emitter to the collector, protecting against reverse voltages across the transistor, and one connecting the collector to pin 9; if pin 9 is wired to the positive motor supply, this diode will protect the transistor against inductive spikes.
The ULN2803 chip is essentially the same as the ULN2003 chip described above, except that it is in an 18-pin package, and contains 8 darlingtons, allowing one chip to be used to drive a pair of common unipolar permanent-magnet or variable-reluctance motors.
For motors drawing under 600 milliamps per winding, the UDN2547B quad power driver made by Allegro Microsystems will handle all 4 windings of common unipolar stepping motors. For motors drawing under 300 milliamps per winding, Texas Instruments SN7541, 7542 and 7543 dual power drivers are a good choice; both of these alternatives include some logic with the power drivers.

Bipolar Motors and H-Bridges

Things are more complex for bipolar permanent magnet stepping motors because these have no center taps on their windings. Therefore, to reverse the direction of the field produced by a motor winding, we need to reverse the current through the winding. We could use a double-pole double throw switch to do this electromechanically; the electronic equivalent of such a switch is called an H-bridge and is outlined in Figure 3.9:

Figure 3.9



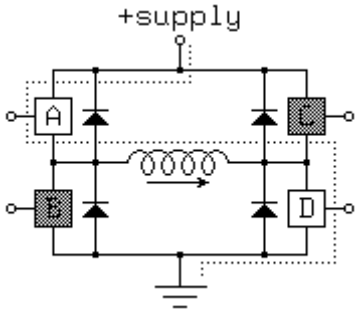
As with the unipolar drive circuits discussed previously, the switches used in the H-bridge must be protected from the voltage spikes caused by turning the power off in a motor winding. This is usually done with diodes, as shown in Figure 3.9.
It is worth noting that H-bridges are applicable not only to the control of bipolar stepping motors, but also to the control of DC motors, push-pull solenoids (those with permanent magnet plungers) and many other applications.
With 4 switches, the basic H-bridge offers 16 possible operating modes, 7 of which short out the power supply! The following operating modes are of interest:

Forward mode, switches A and D closed.

Reverse mode, switches B and C closed.

These are the usual operating modes, allowing current to flow from the supply, through the motor winding and onward to ground. Figure 3.10 illustrates forward mode:

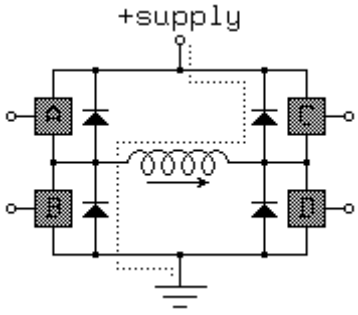
Figure 3.10



Fast decay mode or *coasting mode*, all switches open.

Any current flowing through the motor winding will be working against the full supply voltage, plus two diode drops, so current will decay quickly. This mode provides little or no dynamic braking effect on the motor rotor, so the rotor will coast freely if all motor windings are powered in this mode. Figure 3.11 illustrates the current flow immediately after switching from forward running mode to fast decay mode.

Figure 3.11

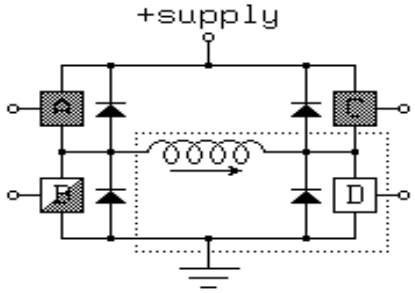


Slow decay modes or *dynamic braking modes.*

In these modes, current may recirculate through the motor winding with minimum resistance. As a result, if current is flowing in a motor winding when one of these modes is entered, the current will decay slowly, and if the motor rotor is turning, it will induce a current that will act

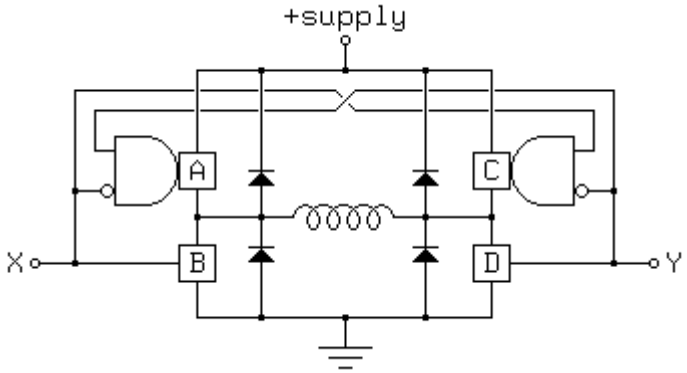
as a brake on the rotor. Figure 3.12 illustrates one of the many useful slow-decay modes, with switch D closed; if the motor winding has recently been in forward running mode, the state of switch B may be either open or closed:

Figure 3.12



Most H-bridges are designed so that the logic necessary to prevent a short circuit is included at a very low level in the design. Figure 3.13 illustrates what is probably the best arrangement:

Figure 3.13



Here, the following operating modes are available:

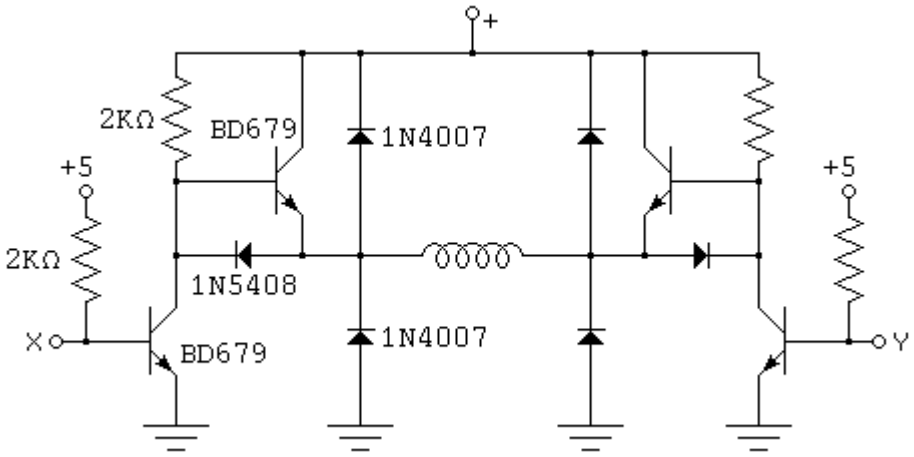
XY		ABCD	Mode
00		0000	fast decay
01		1001	**forward**
10		0110	**reverse**
11		0101	slow decay

The advantage of this arrangement is that all of the useful operating modes are preserved, and they are encoded with a minimum number of bits; the latter is important when using a microcontroller or computer system to drive the H-bridge because many such systems have only limited numbers of bits available for parallel output. Sadly, few of the integrated H-bridge chips on the market have such a simple control scheme.

Practical Bipolar Drive Circuits

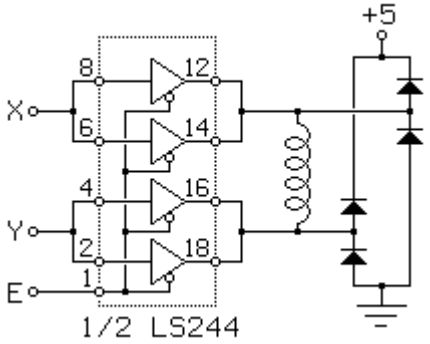
There are a number of integrated H-bridge drivers on the market, but it is still useful to look at discrete component implementations for an understanding of how an H-bridge works. Antonio Raposo (ajr@cybill.inesc.pt) suggested the H-bridge circuit shown in Figure 3.14.
The X and Y inputs to this circuit can be driven by open collector TTL outputs as in the darlington-based unipolar drive circuit in Figure 3.7. The motor winding will be energised if exactly one of the X and Y inputs is high and exactly one of them is low. If both are low, both pull-down transistors will be off. If both are high, both pull-up transistors will be off. As a result, this simple circuit puts the motor in dynamic braking mode in both the 11 and 00 states, and does not offer a coasting mode.

Figure 3.14



The circuit in Figure 3.14 consists of two identical halves, each of which may be properly described as a push-pull driver. The term half H-bridge is sometimes applied to these circuits! It is also worth noting that a half H-bridge has a circuit quite similar to the output drive circuit used in TTL logic. In fact, TTL tri-state line drivers such as the 74LS125A and the 74LS244 can be used as half H-bridges for small loads, as illustrated in Figure 3.15:

Figure 3.15



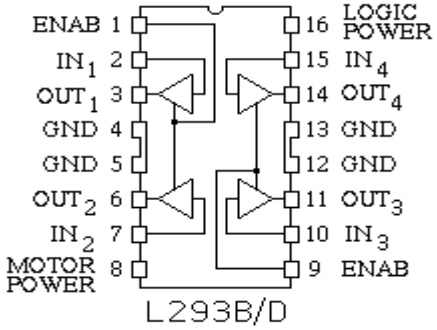
This circuit is effective for driving motors with up to about 50 ohms per winding at voltages up to about 4.5 volts using a 5 volt supply. Each tri-state buffer in the LS244 can sink about twice the current it can source, and the internal resistance of the buffers is sufficient, when sourcing current, to evenly divide the current between the drivers that are run in parallel. This motor drive allows for all of the useful states achieved by the driver in Figure 3.13, but these states are not encoded as efficiently:

XYE		**Mode**
--1		fast decay
000		slower decay
010		**forward**
100		**reverse**
110		slow decay

The second dynamic braking mode, XYE=110, provides a slightly weaker braking effect than the first because of the fact that the LS244 drivers can sink more current than they can source.
The Microchip (formerly Telcom Semiconductor) TC4467 Quad CMOS driver is another example of a general purpose driver that can be used as 4 independent half H-bridges. Unlike earlier drivers, the data sheet for this driver even suggests using it for motor control applicatons, with supply voltages up to 18 volts and up to 250 milliamps per motor winding.
One of the problems with commercially available stepping motor control chips is that many of them have relatively short market lifetimes. For example, the Seagate IPxMxx series of dual H-bridge

chips (IP1M10 through IP3M12) were very well thought out, but unfortunately, it appears that Seagate only made these when they used stepping motors for head positioning in Seagate disk drives. The Toshiba TA7279 dual H-bridge driver would be another another excellent choice for motors under 1 amp, but again, it appears to have been made for internal use only.
The SGS-Thompson (and others) L293 dual H-bridge is a close competitor for the above chips, but unlike them, it does not include protection diodes. The L293D chip, introduced later, is pin compatible and includes these diodes. If the earlier L293 is used, each motor winding must be set across a bridge rectifier (1N4001 equivalent). The use of external diodes allows a series resistor to be put in the current recirculation path to speed the decay of the current in a motor winding when it is turned off; this may be desirable in some applications. The L293 family offers excellent choices for driving small bipolar steppers drawing up to one amp per motor winding at up to 36 volts. Figure 3.16 shows the pinout common to the L293B and L293D chips:

Figure 3.16



This chip may be viewed as 4 independent half H-bridges, enabled in pairs, or as two full H-bridges. This is a power DIP package, with pins 4, 5, 12 and 13 designed to conduct heat to the PC board or to an external heat sink.
The SGS-Thompson (and others) L298 dual H-bridge is quite similar to the above, but is able to handle up to 2-amps per channel and is packaged as a power component; as with the LS244, it is safe to wire the two H-bridges in the L298 package into one 4-amp H-bridge (the data sheet for this chip provides specific advice on how to do this). One warning is appropriate concerning the L298; this chip very fast switches, fast enough that commonplace protection diodes (1N400X equivalent) don't work. Instead, use a diode such as the BYV27. The National Semiconductor LMD18200 H-bridge is another good example; this handles up to 3 amps and has integral protection diodes.
While integrated H-bridges are not available for very high currents or very high voltages, there are well designed components on the market to simplify the construction of H-bridges from discrete switches. For example, International Rectifier sells a line of half H-bridge drivers; two of these chips plus 4 MOSFET switching transistors suffice to build an H-bridge. The IR2101, IR2102 and IR2103 are basic half H-bridge drivers. Each of these chips has 2 logic inputs to directly control the two switching transistors on one leg of an H-bridge. The IR2104 and IR2111 have similar output-side logic for controlling the switches of an H-bridge, but they also include input-side logic that, in some applications, may reduce the need for external logic. In particular, the 2104 includes an enable input, so that 4 2104 chips plus 8 switching transistors can replace an L293 without additional logic.
The data sheet for the Microchip (formerly Telcom Semiconductor) TC4467 family of quad CMOS drivers includes information on how to use drivers in this family to drive the power MOSFETs of H-bridges running at up to 15 volts.
A number of manufacturers make complex H-bridge chips that include current limiting circuitry; these are the subject of the next section. It is also worth noting that there are a number of 3-phase bridge drivers on the market, appropriate for driving Y or delta configured 3-phase permanent magnet steppers. Few such motors are available, and these chips were not developed with steppers in mind. Nonetheless, the Toshiba TA7288P, the GL7438, the TA8400 and TA8405 are clean designs, and 2 such chips, with one of the 6 half-bridges ignored, will cleanly control a 5-winding 10 step per revolution motor.

4. Current Limiting for Stepping Motors

Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

Introduction

Small stepping motors, such as those used for head positioning on floppy disk drives, are usually driven at a low DC voltage, and the current through the motor windings is usually limited by the internal resistance of the winding. High torque motors, on the other hand, are frequently built with very low resistance windings; when driven by any reasonable supply voltage, these motors typically require external current limiting circuitry.

There is good reason to run a stepping motor at a supply voltage above that needed to push the maximum rated current through the motor windings. Running a motor at higher voltages leads to a faster rise in the current through the windings when they are turned on, and this, in turn, leads to a higher cutoff speed for the motor and higher torques at speeds above the cutoff.

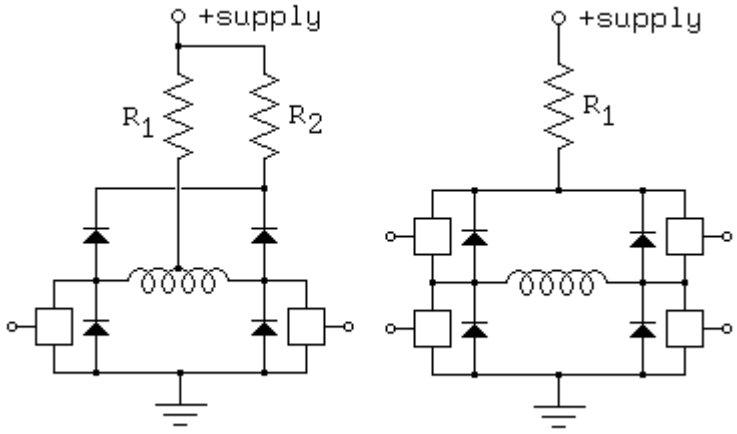
Microstepping, where the control system positions the motor rotor between half steps, also requires external current limiting circuitry. For example, to position the rotor 1/4 of the way from one step to another, it might be necessary to run one motor winding at full current while the other is run at approximately 1/3 of that current.

The remainder of this section discusses various circuits for limiting the current through the windings of a stepping motor, starting with simple resistive limiters and moving up to choppers and other switching regulators. Most of these current limiters are appropriate for many other applications, including limiting the current through conventional DC motors and other inductive loads.

Resistive Current Limiters

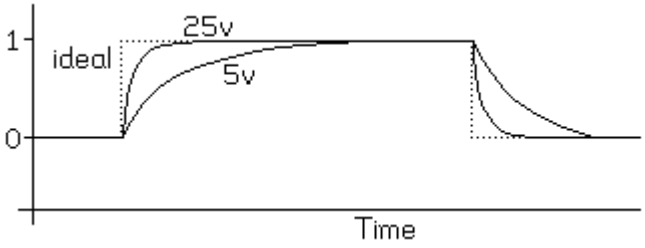
The easiest to understand current limiter is a series resistor. Most motor manufacturers recommended this approach in their literature up until the early 1980's, and most motor data sheets still give performance curves for motors driven by such circuits. The typical circuits used to control the current through one winding of a permanent magnet or hybrid motor are shown in Figure 4.1.

Figure 4.1



R_1 in this figure limits the current through the motor winding. Given a rated current of I and a motor winding with a resistance R_w, Ohm's law sets the maximum supply voltage as $I(R_w+R_1)$. Given that the inductance of the motor motor winding is L_w, the time constant for the motor winding will be $L_w/(R_w+R_1)$. Figure 4.2 illustrates the effect of increasing the resistance and the operating voltage on the rise and fall times of the current through one winding of a stepping motor.

Figure 4.2



R_2 is shown only in the unipolar example in Figure 4.1 because it is particularly useful there. For a bipolar H-bridge drive, when all switches are turned off, current flows from ground to the motor supply through R_1, so the current through the motor winding will decay quite quickly. In the unipolar case, R_2 is necessary to equal this performance.

Note: When the switches in the H-bridge circuit shown in Figure 4.1 are opened, the direction of current flow through R_1 will reverse almost instantaneously! If R_1 has any inductance, for example, if it is wire-wound, it must either be bypassed with a capacitor to handle the voltage kick caused by this current reversal, or R_2 must be added to the H-bridge.

Given the rated maximum current through each winding and the supply voltage, the resistance and wattage of R_1 is easy to compute. R_2 if it is included, poses more interesting problems. The resistance of R_2 depends on the maximum voltage the switches can handle. For example, if the supply voltage is 24 volts, and the switches are rated at 75 volts, the drop across R_2 can be as much as 51 volts without harming the transistors. Given an operating current of 1.5 amps, R_2 can be a 34 ohm resistor. Note that an interesting alternative is to use a zener diode in place of R_2.

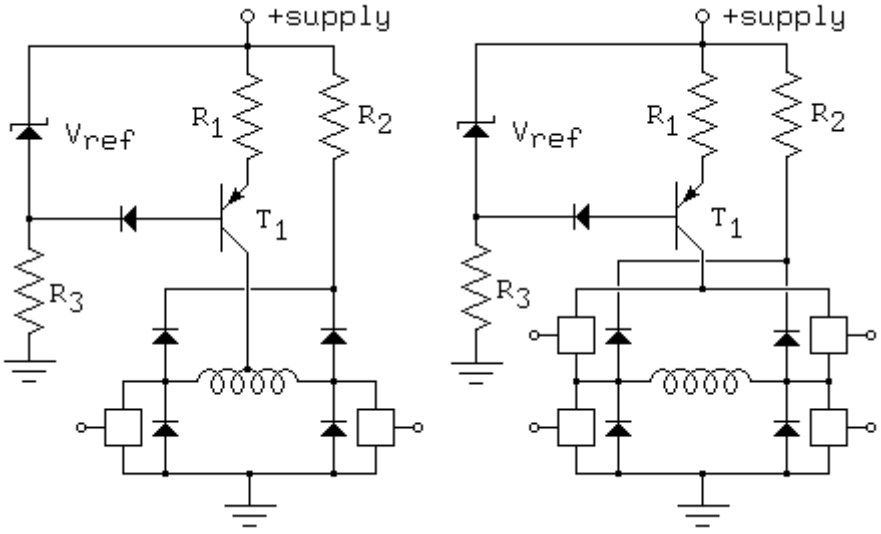
Figuring the peak average power R_2 must dissipate is a wonderful exercise in dynamics; the inductance of the motor windings is frequently undocumented and may vary with the rotor position. The power dissapated in R_2 also depends on the control system. The worst case occurs when the control system chops the power to one winding at a high enough frequency that the current through the motor winding is effectively constant; the maximum power is then a function of the duty cycle of the chopper and the ratios of the resistances in the circuit during the on and off phases of the chopper. Under normal operating conditions, the peak power dissipation will be significantly lower.

Linear Current Limiters

A pair of high wattage power resistors can cost more than a pair of power transistors plus a heat sink, particularly if forced air cooling is available. Furthermore, a transistorized constant current source, as shown in Figure 4.3, will give faster rise times through the motor windings than the

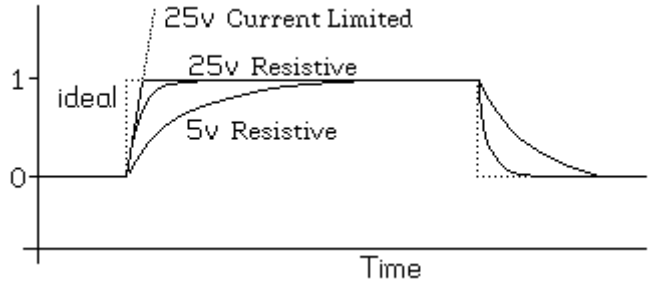
current limiting resistor shown in Figure 4.1. This is because a current source will deliver the full supply voltage across the motor winding until the current reaches the rated current; only then will the current source drop the voltage.

Figure 4.3



In Figure 4.3, a transistorized current source (T_1 plus R_1) has been substituted for the current limiting resistor R_1 used in the examples in Figure 4.1. The regulated voltage supplied to the base of T_1 serves to regulate the voltage across the sense resistor R_1, and this, in turn, maintains a constant current through R_1 so long as any current is allowed to flow through the motor winding.
Typically, R_1 will have as low a resistance as possible, in order to avoid the high cost of a power resistor. For example, if the forward voltage drops across the diode in series with the base T_1 and V_{BE} for T_1 are both 0.65 volts, and if a 3.3 volt zener diode is used for a reference, the voltage across R_1 will be maintained at about 2.0 volts, so if R_1 is 2 ohms, this circuit will limit the current to 1 amp, and R_1 must be able to handle 2 watts.
R_3 in Figure 4.3 must be sized in terms of the current gain of T_1 so that sufficient current flows through R_1 and R_3 to allow T_1 to conduct the full rated motor current.
The transistor T_1 used as a current regulator in Figure 4.3 is run in linear mode, and therfore, it must dissapate quite a bit of power. For example, if the motor windings have a resistance of 5 ohms and a rated current of 1 amp, and a 25 volt power supply is used, T_1 plus R_1 will dissapate, between them, 20 watts! The circuits discussed in the following sections avoid this waste of power while retaining the performance advantages of the circuit given here.
When an H-bridge bipolar drive is used with a resistive current limiter, as shown in Figure 4.1, the resistor R_2 was not needed because current could flow backwards through R_1. When a transistorized current limiter is used, current cannot flow backwards through T_1, so a separate current path back to the positive supply must be provided to handle the decaying current through the motor windings when the switches are opened. R_2 serves this purpose here, but a zener diode may be substituted to provide even faster turn-off.
The performance of a motor run with a current limited power supply is noticably better than the performance of the same motor run with a resistively limited supply, as illustrated in Figure 4.4:

Figure 4.4



With either a current limited supply or a resistive current limiter, the initial rate of increase of the current through the inductive motor winding when the power is turned on depends only on the inductance of the winding and the supply voltage. As the current increases, the voltage drop across a resistive current limiter will increase, dropping the voltage applied to the motor winding, and therefore, dropping the rate of increase of the current through the winding. As a result, the current will only approach the rated current of the motor winding asymptotically
In contrast, with a pure current limiter, the current through the motor winding will increase almost linearly until the current limiter cuts in, allowing the current to reach the limit value quite quickly. In fact, the current rise is not linear; rather, the current rises asymptotically towards a limit established by the resistance of the motor winding and the resistance of the sense resistor in the current limiter. This maximum is usually well above the rated current for the motor winding.

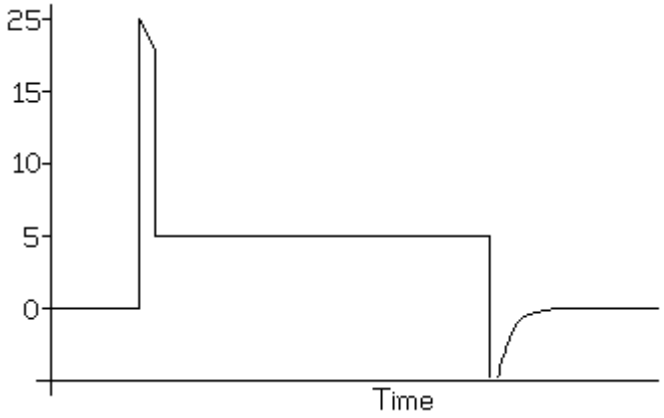
Open Loop Current Limiters

Both the resistive and the linear transistorized current limiters discussed above automatically limit the current through the motor winding, but at a considerable cost, in terms of wasted heat. There are two schemes that eliminate this expense, although at some risk because of the lack of feeback about the current through the motor.

Use of a Voltage Boost

If you plot the voltage across the motor winding as a function of time, assuming the use of a transistorized current limiter such as is illustrated in Figure 4.3, and assuming a 1 amp 5 ohm motor winding, the result will be something like that illustrated in Figure 4.5:

Figure 4.5



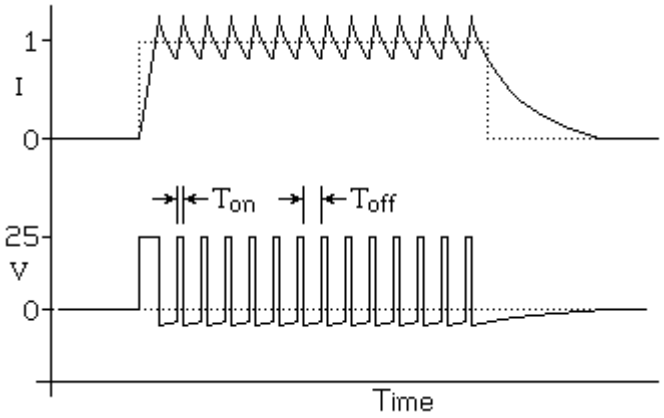
As long as the current is below the current limiter's set point, almost the full supply voltage is applied across the motor winding. Once the current reaches the set point, the voltage across the motor winding falls to that needed to sustain the current at the set point, and when the switches open, the voltage reverses briefly as current flows through the diode network and R_2.
An alternative way to get this voltage profile is to use a dual-voltage power supply, turning on the high voltage for as long as it takes to bring the current in the motor winding up to the rated current, and then turning off the high voltage and turning on the sustaining voltage. Some motor controllers do this directly, without monitoring the current through the motor windings. This provides excellent performance and minimizes power losses in the regulator, but it offers a dangerous temptation.
If the motor does not deliver enough torque, it is tempting to simply lengthen the high-voltage pulse at the time the motor winding is turned on. This will usually provide more torque, although saturation of the magnetic circuits frequently leads to less torque than might be expected, but the cost is high! The risk of burning out the motor is quite real, as is the risk of demagnitizing the motor rotor if it is turned against the imposed field while running hot. Therefore, if a dual-voltage supply is used, the temptation to raise the torque in this way should be avoided!

The problems with dual voltage supplies are particularly serious when the time intervals are under software control, because in this case, it is common for the software to be written by a programmer who is insufficiently aware of the physical and electrical characteristics of the control system.

Use of Pulse Width Modulation

Another alternative approach to controlling the current through the motor winding is to use a simple power supply controlled by *pulse width modulaton* (PWM) or by a *chopper*. During the time the current through the motor winding is increasing, the control system leaves the supply attached with a 100% duty cycle. Once the current is up to the full rated current, the control system changes the duty cycle to that required to maintain the current. Figure 4.6 illustrates this scheme:

Figure 4.6



For any chopper or pulse width modulator, we can define the duty-cycle D as the fraction of each cycle that the switch is closed: $D = T_{on} / (T_{on} + T_{off})$ Where T_{on} -- time the switch is closed during each cycle
T_{off} -- time the switch is open during each cycle The voltage curve shown above indicates the full supply voltage being applied to the motor winding during the on-phase of every chopper cycle, while when the chopper is off, a negative voltage is shown. This is the result of the forward voltage drop in the diodes that are used to shunt the current when the switches turn off, plus the external resistance used to speed the decay of the current through the motor winding.
For large values of T_{on} or T_{off}, the exponential nature of the rise and fall of the current through the motor winding is significant, but for sufficiently small values, we can approximate these as linear. Assuming that the chopper is working to maintain a current of I and that the amplitude is small, we will approximate the rates of rise and fall in the current in terms of the voltage across the motor winding when the switch is closed and when it is open: $V_{on} = V_{supply} - I(R_{winding} + R_{on})$
$V_{off} = V_{diode} + I(R_{winding} + R_{off})$ Here, we lump together all resistances in series with the winding and power supply in the on state as R_{on}, and we lump together all resistances in the current recirculation path when the switch(es) are open as R_{off}. The forward voltage drops of any diodes in the current recirculation path have been lumped as V_{diode}; if the off-state recirculation path runs from ground to the power supply (H-bridge fast decay mode), the supply voltage must also be included in V_{diode}. Forward voltage drops of any switches in the on-state and off-state paths should also be incorporated into these voltages.
To solve for the duty cycle, we first note that: $dI/dt = V/L$ Where I -- current through the motor winding
V -- voltage across the winding
L -- inductance of the winding
We then substitute the specific voltages for each phase of operation:
$I_{ripple} / T_{off} = V_{off} / L$
$I_{ripple} / T_{on} = V_{on} / L$ Where I_{ripple} -- the peak to peak ripple in the current Solving for T_{off} and T_{on} and then substituting these into the definition of the duty cycle of the chopper, we get: $D = T_{on} / (T_{on} + T_{off}) = V_{off} / (V_{on} + V_{off})$ If the forward voltage drops in diodes and switches are negligable, and if

the only significant resistance is that of the motor winding itself, this simplifies to: $D = I R_{winding} / V_{supply} = V_{running} / V_{supply}$ This special case is particularly desirable because it delivers all of the power to the motor winding, with no losses in the regulation system, without regard for the difference between the supply voltage and the running voltage.
The AC ripple I_{ripple} superimposed on the running current by a chopper can be a source of minor problems; at high frequencies, it can be a source of RF emissions, and at audio frequencies, it can be a source of annoying noise. For example, with audio frequency chopping, most stepper controlled systems will "squeel", sometimes loudly, when the rotor is displaced from the equilibrium position. To find the ripple amplitude, first recall that: $I_{ripple} / T_{off} = V_{off} / L$ Then solve for I_{ripple}: $I_{ripple} = T_{off} V_{off} / L$ Thus, to reduce the ripple amplitude at any particular duty cycle, it is necessary to increase the chopper frequency. This cannot be done without limit because switching losses increase with frequency. Note that this change has no significant effect on AC losses; the decrease in such losses due to decreased amplitude in the ripple is generally offset by the effect of increasing frequency.
The primary problem with use of a simple chopping or pulse-width modulation control scheme is that it is completely open loop. Design of good chopper based control systems requires knowledge of motor characteristics such as inductance that are frequently poorly documented, and as with dual-voltage supplies, when motor performance is marginal, it is very tempting to increase the duty-cycle without attention to the long-term effects of this on the motor. In the designs that follow, this weakness will be addressed by introducing feedback loops into the low level drive system to directly monitor the current and determine the duty cycle.

One-Shot Feedback Current Limiting

The most common approach to automatically adjusting the duty cycle of the switches in the stepper driver involves monitoring the current to the motor windings; when it rises too high, the winding is turned off for a fixed interval. This requires a current sensing system and a one-shot, as illustrated in Figure 4.7:

Figure 4.7

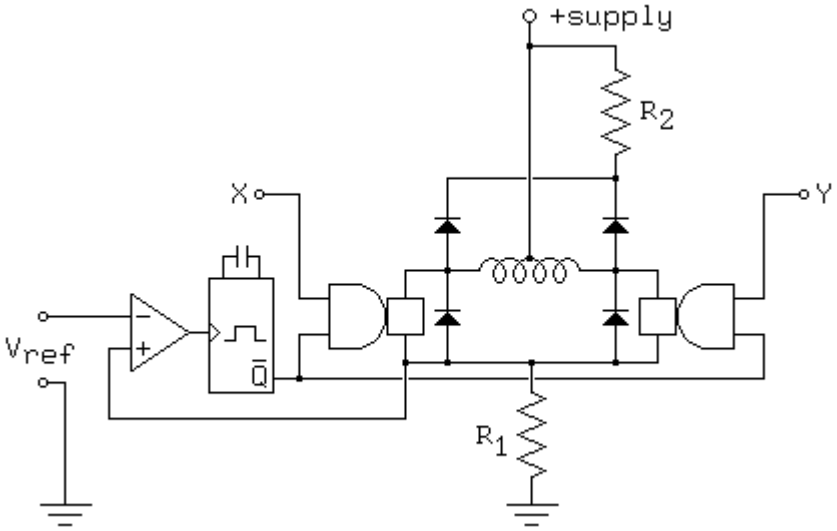


Figure 4.7 illustrates a unipolar drive system. As with the circuit given in Figure 4.3, R_1 should be as small as possible, limited only by the requirement that the sense voltage provided to the comparator must be high enough to be within its operating range. Note that when the one-shot output (¬Q) is low, the voltage across R_1 no-longer reflects the current through the motor winding. Therefore, the one-shot must be insensitive to the output of the comparator between the time it fires and the time it resets. Practical circuit designs using this approach involve some complexity to meet this constraint!
Selecting the value of R_2 for the circuit shown in Figure 4.7 poses problems. If R_2 is large, the current through the motor windings will decay quickly when the higher level control system turns off this motor winding, but when the winding is turned on, the current ripple will be large and the power lost in R_2 will be significant. If R_2 is small, this circuit will be very energy efficient but the current

through the motor winding will decay only slowly when this winding is turned off, and this will reduce the cutoff speed for the motor.

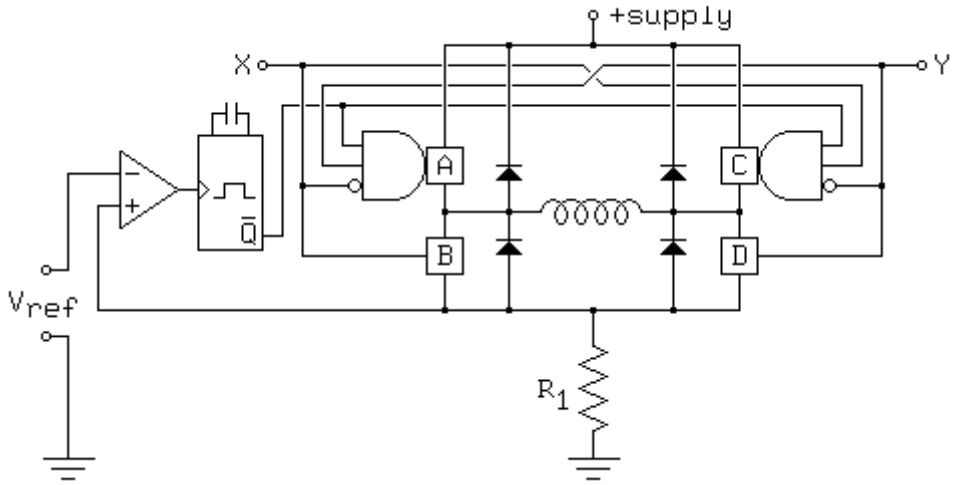
The peak power dissapated in R_2 will be I^2R_2 during T_{off} and zero during T_{on}; thus, the average power dissapated in R_2 when the motor winding is on will be: $P_2 = I^2R\ T_{off} / (T_{on} + T_{off})$ Recall that the duty cycle D is defined as $T_{on}/(T_{on}+T_{off})$ and may be approximated as $V_{running}/V_{supply}$. As a result, we can approximate the power dissapation as: $P_2 = I^2R_2\ (1 - V_{running}/V_{supply})$. Given the usual safety margins used in selecting power resistor wattages, a better approximation is not necessary.

When designing a control system based on pulse width modulation, note that the cutoff time for the one-shot determines T_{off}, and that this is fixed, determined by the timing network attached to the one-shot. Ideally, this should be set as follows: $T_{off} = L\ I_{ripple} / V_{off}$ This presumes that the inductance L of the motor winding is known, that the acceptable magnitude of I_{ripple} is known, and that V_{off}, the total reverse voltage in the current recirculation path, is known and fixed.

Note that this scheme leads to a variable chopping rate. As with the linear current limiters shown in Figure 4.3, the full supply voltage will be applied during the turn-on phase, and the chopping action only begins when the motor winding reaches the current limit set by V_{ref}. This circuit will vary the chopping rate to compensate for changes in the back EMF of the motor winding, for example, those caused by rotor motion; in this regard, it offers the same quality of regulation as the linear current limiter.

The one-shot current regulator shown in Figure 4.7 can also be applied to an H-bridge regulator. The encoded H-bridge shown in Figure 3.13 is an excellent candidate for this application, as shown in Figure 4.8:

Figure 4.8



Unlike the circuit in Figure 4.7, this circuit does not provide design tradeoffs in the selection of the resistance in the current decay path; instead, it offers the same selection of decay paths as was available in the original circuit from Figure 3.13. If the X and Y control inputs are held in a running mode (01 or 10), the current limiter will alternate between that running and slow decay modes, maximizing energy efficiency. When the time comes to turn off the current through the motor winding, the X and Y inputs may be set to 00, using fast decay mode to maximize the cutoff speed, while if the damping effect of dynamic braking is needed to control resonance, X and Y may be set to 11.

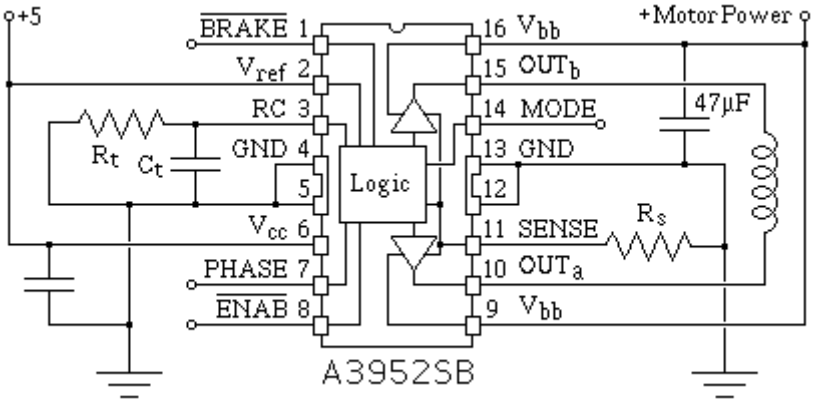
Note that the current recirculation path during dynamic braking does not pass through R_1, and as a result, if the motor generates a large amount of power, burnt out components in the motor or controller are likely. This is unlikely to cause problems with stepping motors, but when dynamic braking is used with DC motors, the current limiter should be arranged to remain engaged while in braking mode!

Practical Examples

SGS-Thompson (and others) L293 (1A) and L298 (2A) dual H-bridges are designed for easy use with partial feedback current limiters. These chips have enable inputs for each H-bridge that can be directly connected to the output of the one-shot, and they have ground connections for motor-power that are isolated from their logic ground connections; this allows sense resistors to be easily incorporated into the circuit.

The 3952 H-bridge from Allegro Microsystems can handle up to 2-amps at 50 volts and incorporates all of the logic necessary for current control, including comparators and one-shot. This chip is available in many package styles; Figure 4.9 illustrates the DIP configuration wired for a constant current limit:

Figure 4.9



If R_t is 20 Kohm, and C_t is 1000pF, T_{off} for the pulse-width modulation will be fixed at 20 (±2) microseconds. The 3952 chip incorporates a 10 to 1 voltage divider on the V_{ref} input, so attaching V_{ref} to the 5 volt logic supply sets the actual reference voltage to 0.5 V. Thus, if the sense resistor R_s is 0.5 ohms, this arrangement will attempt to maintain a regulated current through the load of 1 A.

Note that all power switching chips are potentially serious sources of electromagnetic interfence! The 47µF capacitor shown between the motor power and ground should be as close to the chip as possible, and the path from the SENSE pin through R_s to ground and back to a ground pin of the chip should be very short and with a very low resistance.

On the 5 volt side, because V_{ref} is taken from V_{cc}, a small decoupling capacitor should be placed very close to the chip. It may even be appropriate to isolate the V_{ref} input from V_{cc} with a small series resistor and a separate decoupling capacitor. If this is done, note that the resistance from the V_{ref} pin to ground through the chip's internal voltage divider is around 50 Kohms.

One of the more dismaying features of the 3952 chip, as well as many of its competitors, is the large number of control inputs. These are summarized in the following table:

BRAKE	***ENABLE***	**PHASE**	**MODE**		**OUT_a**	**OUT_b**	**Notes**
0	-	-	0		0	0	Brake
0	-	-	1		0	0	Limited Brake
1	1	-	0		-	-	Standby
1	1	-	1		-	-	Sleep
1	0	0	0		0	1	Reverse, Slow
1	0	0	1		0	1	Reverse, Fast
1	0	1	0		1	0	Forward, Slow
1	0	1	1		1	0	Forward, Fast

In the forward and reverse running modes, the mode input determines whether fast or slow decay modes are used during T_{off}. In the dynamic braking modes, the mode input determines whether the current limiter is enabled. This is of limited value with stepping motors, but use of dynamic braking without a current limiter can be dangerous with DC motors.
In sleep mode, the power consumption of the chip is minimized. From the perspective of the load, sleep and standby modes put the load into fast decay mode (all switches off) but in sleep mode, the chip draws considerably less power, both from the logic supply and the motor supply.

Hysteresis Feedback Current Limiting

In many cases, motor control systems are expected to operate acceptably with a number of different stepping motors. The one-shot based current regulators illustrated in Figures 4.7 to 4.9 have an accuracy that depends on the inductance of the motor windings. Therefore, if fixed accuracy is required, any motor substition must be balanced by changes to the RC network that determines the off-time of the one-shot.
This section deals with alternative designs that eliminate the need for this tuning. These alternative designs offer fixed precision current regulation over a wide range of load inductances. The key to this approach is arrange the recirculation paths so that the current-sense resistor R_1 is always in the circuit, and then turn the switches on or off depending only on the current.
The usualy way to build this type of controller is to use a comparator with a degree of hysteresis, for example, by feeding the output of the comparator back into one of its inputs through a resistor network, as illustrated in Figure 4.10:

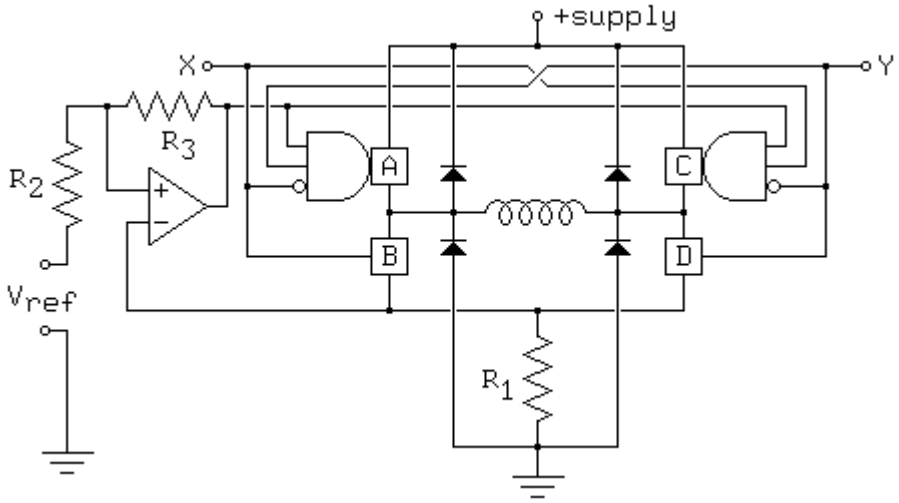


Figure 4.10

To compute the desired values of R_2 and R_3, we note that: $V_{ripple} > V_{hysteresis}$ Where: $V_{ripple} = I_{ripple} R_1$ I_{ripple} -- the maximum ripple allowed in the current and: $V_{hysteresis} = V_{swing} R_2 / (R_2 + R_3)$ V_{swing} -- the voltage swing at the output of the comparator We can solve this for the ratio of the resistances: $R_2 / (R_2 + R_3) < I_{ripple} R_1 / V_{swing}$ For example, if R_1 is 0.5 ohms and we wish to regulate the current to within 10 milliamps, using a comparator with TTL compatable outputs and a voltage swing of 4 volts, the ratio must be no greater than .00125.
Note that the sum $R_2 + R_3$ determines the loading on V_{ref}, assuming that the input resistance of the comparator is effectively infinite. Typically, therefore, this sum is made quite large.
One problem with the circuit given in Figure 4.10 is that it does not limit the current through the motor in dynamic braking or slow decay modes. Even if the current through the sense resistor vastly exceeds the desired current, switches B and D will remain closed in dynamic braking mode, and if the reference voltage is variable, rapid drops in the reference voltage will not be enforced by this control system.
The designers of the Allegro 3952 chip faced this problem, and passed the solution back to the user, providing a MODE input to determine whether the chopper alternated between running and fast decay mode or running and slow decay mode. Note that this chip uses a fixed off-time set by a

one-shot, and therefore, switching between the two decay modes will change the precision of the current regulator. Given that such a change in precision is acceptable, we can modify the circuit from Figure 4.10 to automatically thrown the system into fast-decay mode if the running or dynamic braking current exceeds the set-point of the comparator by too great a margin. Figure 4.11 illustrates how this can be done using a second comparator:

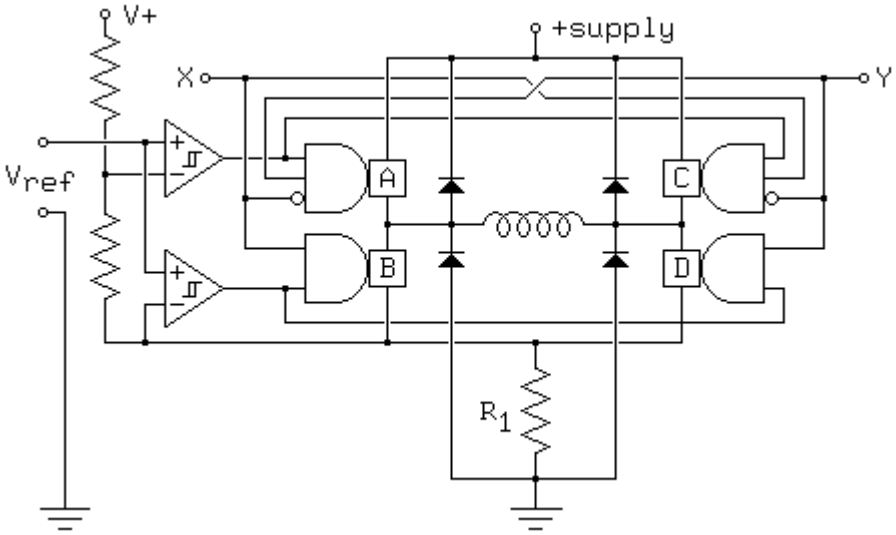


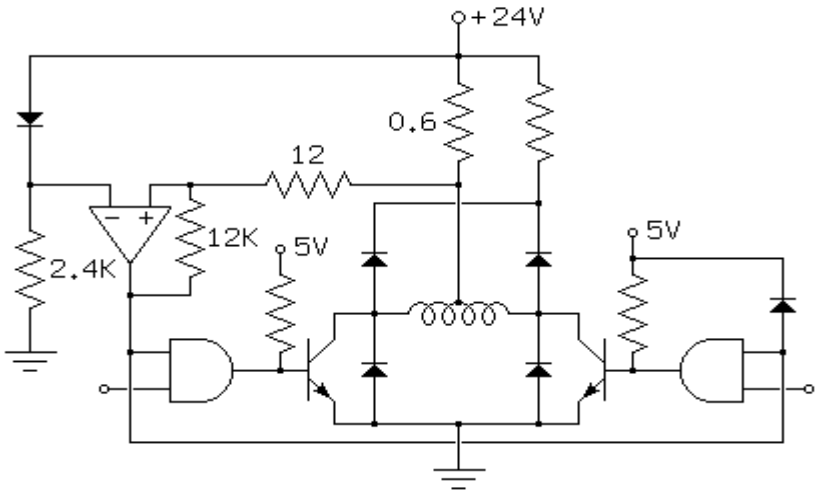
Figure 4.11

As shown in Figure 4.11, the lower comparator directly senses the voltage across R1, while the upper comparator senses a higher voltage, determined by a resistor network. This network should hold the negative inputs of the two comparators just far enough apart to guarantee that, as the voltage across R1 rises, the top comparator will always open the top switches before the bottom comparator opens the bottom switches, and as the voltage across R1 falls, the bottom comparator will always close the bottom switches before the top comparator closes the top switches.
As a result, this system has two basic steady-state running modes. If the motor winding is drawing power, one of the bottom switches will remain closed while the opposite switch on the top is used to chop the power to the motor winding, alternating the state of the system between running and slow-decay mode.
If the motor winding is generating power, the top switches will remain open and the bottom switches will do the chopping, alternating between fast-decay and slow-decay modes as needed to keep the current within limits.
If the two comparators have accuracies on the order of a millivolt with hysteresis on the order of 5 millivolts, it is reasonable to use a 5 millivolt difference between the top and bottom comparator. If we use the 5 volt logic supply as the pull-up supply for the resistor network, and we assume a nominal operating threshold of around 0.5 volts, the resistor network should have a ratio of 1:900; for example, a 90k resistor from +5 and a 100 ohm resistor between the two comparator inputs.

Practical Examples

The basic idea described in this section is also applicable to unipolar stepping motor controllers, although in this context, it is somewhat easier to apply if the reference voltage is measured with respect to the unregulated motor power supply. Figure 4.12 illustrates a practical example, using the forward voltage drop across an ordinary silicon diode as the reference voltage.

Figure 4.12



The circuit shown in Figure 4.12 uses a 2.4K resistor to provide a bias current of 10ma to the reference diode. A small capacitor should be added across the reference diode if the motor power supply is minimally regulated.

The 0.6 ohm value used for the current sensing resistor sets the regulator to 1 amp, assuming that the reference voltage is 0.6 volts. The 1000 to 1 ratio on the feedback network around the comparator sets the allowed ripple in the regulated current to around 8 ma.

The comparator shown in Figure 4.12 can be powered from the minimally regulated motor power supply, but only if it is able to operate with the inputs very close to its positive supply voltage. Although I have not tried it, the Mitsubishi M5249L comparator appears to be ideally suited to this job; it can work from a positive supply of up to 40 volts, and the input voltages are allowed to slightly exceed the positive supply voltage! The output of this comparator is open collector, so the hysteresis network shown in the figure also acts as a pull-up network, providing a pull-up current of a few milliamps. The diode to +5 shown in the figure clamps the comparator output to the logic supply voltage, protecting the and gate inputs from overvoltage.

Other Current Sensing Technologies

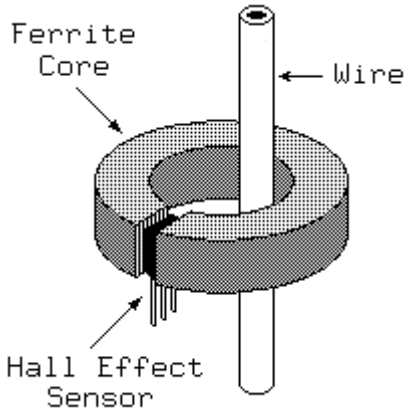
The feedback loops of all of the current limiters given above use the voltage drop across a small resistor to measure the current. This is an excellent choice for small motors, but it poses difficulties for large high-current motors! There are other current sensing technologies appropriate for such settings, most notably those that deliver only a fraction of the motor current to the sensing resistor, and those that measure the current by sensing the magnetic field around the conductor.

National Semiconductor had incorporated a very clever current sensor into a number of their H-bridges. This delivers a current to the sense resistor that is proportional to the current through the motor winding, but far lower. For example, on the LMD18200 H-bridge, the sense resistor receives exactly 377 microamps per ampere flowing through the motor winding.

The key to the current sensing technology used in the National Semiconductor line of H-bridges is found in the internal structure of the DMOS power switching transistors they use. These transistors are composed of thousands of small MOSFET transistor cells wired in parallel. A small but representative fraction of these cells, typically 1 in 4000, is used to extract the sense current while the remainder of the cells control the motor current. The data sheet for the National LMD18245 H-bridge contains an excellent writeup on how this is done.

When very high currents are involved, precluding use of an integrated H-bridge, an appealing and well established current sensing technology involves the use of a split ferrite core and a hall effect sensor, as illustrated in Figure 4.13:

Figure 4.13



Simple linear Hall effect sensors require a small regulated bias current between two of their terminals, and they generate a DC voltage proportional to the magnetic field on a third terminal. The magnetic field across the gap sawed in the ferrite core is proportional to the current through the wire, and therefore, the voltage reported by the Hall effect sensor will be proportional to the current.

Allegro Microsystems and others make a full lines of Hall effect sensors, but pre-calibrated hall effect current sensors are available; these include the split core, the hall effect sensor, and auxiliary components, all mounted on a small PC board or potted as a unit. Newark Electronics lists a few sources of these, including Honeywell, F. W. Bell and LEM Instruments.
An intriguing new current sensor is just becoming available, as of 1998, based on a thin-film magnetoresistive sensor; the sensitivity of this technology eliminates the need for the ferrite core and the result is a very compact current sensor. The NT series sensors made by F. W. Bell use this technology.

5. Microstepping of Stepping Motors

Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

Under Development

Introduction

Microstepping serves two purposes. First, it allows a stepping motor to stop and hold a position between the full or half-step positions, second, it largely eliminates the jerky character of low speed stepping motor operation and the noise at intermediate speeds, and third, it reduces problems with resonance.

Although some microstepping controllers offer hundreds of intermediate positions between steps, it is worth noting that microstepping does not generally offer great precision, both because of linearity problems and because of the effects of static friction.

Sine Cosine Microstepping

Recall, from the discussion in Part 2 of this tutorial, on Stepping Motor Physics, that for an ideal two-winding variable reluctance or permanent magnet motor the torque versus shaft angle curve is determined by the following formulas: $h = (a^2 + b^2)^{0.5}$

$x = (S / (\pi/2)) \arctan(b / a)$ Where: a -- torque applied by winding with equilibrium at angle 0.

b -- torque applied by winding with equilibrium at angle S.

h -- holding torque of composite.

x -- equilibrium position.

S -- step angle. This formula is quite general, but it offers little in the way of guidance for how to select appropriate values of the current through the two windings of the motor. A common solution is to arrange the torques applied by the two windings so that their sum h has a constant magnitude equal to the single-winding holding torque. This is referred to as sine-cosine microstepping: $a = h_1 \sin(((\pi/2)/S)\Theta)$

$b = h_1 \cos(((\pi/2)/S)\Theta)$ Where: h_1 -- single-winding holding torque

$((\pi/2)/S)\Theta$ -- the electrical shaft angle Given that none of the magnetic circuits are saturated, the torque and the current are linearly related. As a result, to hold the motor rotor to angle Θ), we set the currents through the two windings as: $I_a = I_{max} \sin(((\pi/2)/S)\Theta)$

$I_b = I_{max} \cos(((\pi/2)/S)\Theta)$ Where: I_a -- current through winding with equilibrium at angle 0.

I_b -- current through winding with equilibrium at angle S.

I_{max} -- maximum allowed current through any motor winding. Keep in mind that these formulas apply to two-winding permanent magnet or hybrid stepping motors. Three pole or five pole motors have more complex behavior, and the magnetic fields in variable reluctance motors don't add following the simple rules that apply to the other motor types.

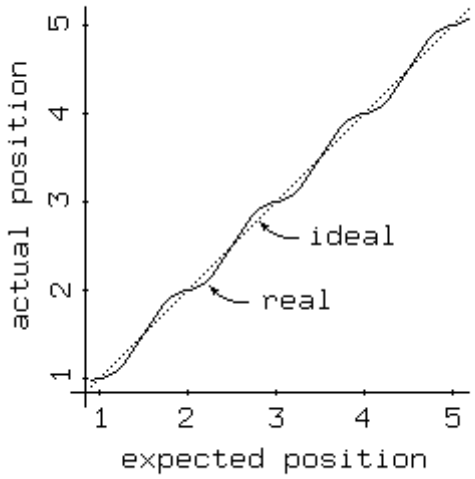
Limits of Microstepping

The utility of microstepping is limited by at least three consideraitons. First, if there is any static friction in the system, the angular precision achievable with microstepping will be limited. This effect was discussed in more detail in the discussion in Part 2 of this tutorial, on Stepping Motor Physics, in the discussion of friction and the dead zone.

Detent Effects

The second problem involves the non-sinusoidal character of the torque versus shaft-angle curves on real motors. Sometimes, this is attributed to the detent torque on permanent magnet and hybrid motors, but in fact, both detent torque and the shape of the torque versus angle curves are products of poorly understood aspects of motor geometry, specifically, the shapes of the teeth on the rotor and stator. These teeth are almost always rectangular, and I am aware of no detailed study of the impact of different tooth profiles on the shapes of these curves.
Most commercially available microstepping controllers provide a fair approximation of the sine-cosine drive current that would drive an ideal stepping motor to uniformly spaced steps. Ideal motors are rare, and when such a controller is used with a real motor, a plot of the actual motor position as a function of the expected position will generally look something like the plot shown in Figure 5.1.

Figure 5.1



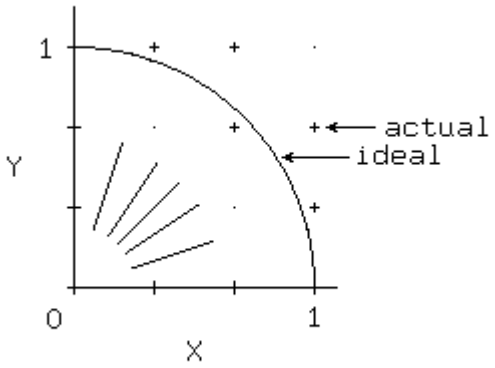
Note that the motor is at its expected position at every full step and at every half step, but that there is significant positioning error in the intermediate positions. The curve shown is the curve that would result from a perfect sin-cosine microstepping controller used with a motor that had a torque versus position curve that included a significant 4th harmonic component, usually attributed to the detent torque.

Quantization

The third problem arises because most applications of microstepping involve digital control systems, and thus, the current through each motor winding is quantized, controlled by a digital to analog converter. Furthermore, if typical PWM current limiting circuitry is used, the current through each motor winding is not held perfectly constant, but rather, oscillates around the current control circuit's set point. As a result, the best a typical microstepping controller can do is approximate the desired currents through each motor winding.
The effect of this quantization is easily seen if the available current through one motor winding is plotted on the X axis and the available current through the other motor winding is plotted on the Y

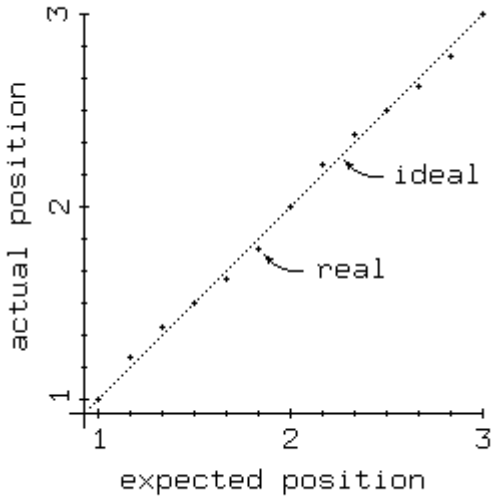
axis. Figure 5.2 shows such a plot for a motor controller offering only 4 uniformly spaced current settings for each motor winding:

Figure 5.2



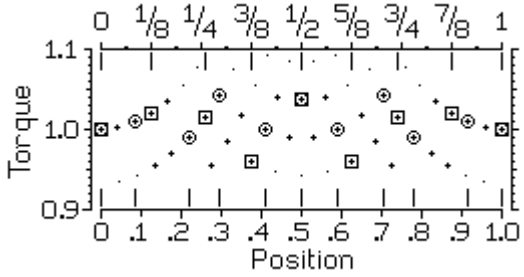
Of the 16 available combinations of currents through the motor windings, 6 combinations lead to roughly equally spaced microsteps. There is a clear tradeoff between minimizing the variation in torque and minimizing the error in motor position, and the best available motor positions are hardly uniformly spaced! Use of higher precision digital to analog conversion in the current control system reduces the severity of this problem, but it cannot eliminate it!
Plotting the actual rotor position of a motor using the microstep plan outlined in Figure 5.2 versus the expected position gives the curve shown in Figure 5.3:

Figure 5.3



It is very common for the initial microsteps taken away from any full step position to be larger than the intended microstep size, and this tends to give the curve a staircase shape, with the downward steps aligned with the full step positions where only one motor winding carries current. The sign of the error at intermediate positions tends to fluctuate, but generally, the position errors are smallest between the full step positions, when both motor windings carry significant current.
Another way of looking at the available microsteps is to plot the equilibrium position on the horizontal axis, in fractions of a full-step, while plotting the torque at each available equilibrium position on the vertical axis. If we assume a 4-bit digital-to-analog converter, giving 16 current levels for each each motor winding, there are 256 equilibrium positions. Of these, 52 offer holding torques within 10% of the desired value, and only 33 are within 5%; these 33 points are shown in bold in Figure 5.4:

Figure 5.4



If torque variations are to be held within 10%, it is fairly easy to select 8 almost-uniformly spaced microsteps from among those shown in Figure 5.4; these are boxed in the figure. The maximum errors occur at the 1/4 step points; the maximum error is .008 full step or .06 microsteps. This error will be irrelevant if the dead-zone is wider than this.

If 10 microsteps are desired, the situation is worse. The best choices, still holding the maximum torque variation to 10%, gives a maximum position error of .026 full steps or .26 microsteps. Doubling the allowable variation in torque approximately halves the positioning error for the 10 microstep example, but does nothing to improve the 8 microstep example.

One option which some motor control system designers have explored involves the use of nonlinear digital to analog converters. This is an excellent solution for small numbers of microsteps, but building converters with essentially sinusoidal transfer functions is difficult if high precision is desired.

Typical Control Circuits

As typically used, a microstepping controller for one motor winding involves a current limited H-bridge or unipolar drive circuit, where the current is set by a reference voltage. The reference voltage is then determined by an analog-to-digital converter, as shown in Figure 5.5:

Figure 5.5

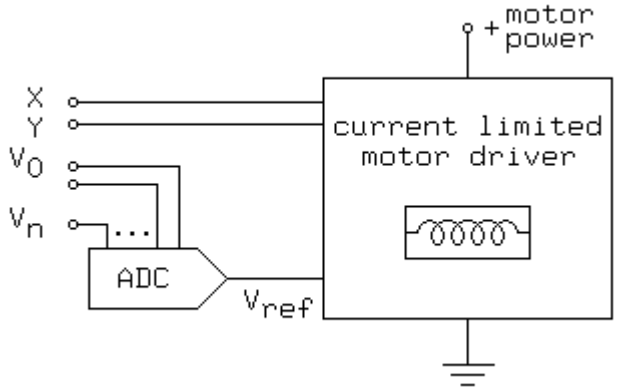


Figure 5.5 assumes a current limited motor controller such as is shown in Figures 4.7, 4.8, 4.10 or 4.11. For all of these drivers, the state of the X and Y inputs determines the whether the motor winding is on or off and if on, the direction of the current through the winding. The V_0 through V_n inputs determine the reference voltage and this the current through the motor winding.

Practical Examples

There are a fair number of nicely designed integrated circuits combining a current limited H-bridge with a small DAC to allow microstepping control of motors drawing under 2 amps per winding. The UDN2916B from Allegro Microsystems is a dual 750mA H-bridge, with a 2-bit DAC to control the current through each. bridge. Another excellent example is the UC3770 from Unitrode. Unitrode. This chip integrate a 2-bit DAC with a PWM controlled H-bridge, packaged in either 16 pin power-dip format or in surface mountable form. The 3717 a slightly cleaner design, good for 1.2 A, while the 3770 is good for up to between 1.8 A or 2 A, depending on how the chip is cooled.

The 3955 from Allegro Microsystems incorporates a 3-bit non-linear DAC and handles up to 1.5 A; this is available in 16-pin power DIP or SOIC formats. The nonlinear DAC in this chip is specifically designed to minimize step-angle errors and torque variations using 8 microsteps per full-step.
The LMD18245 from National Semiconductor is a good choice for microstepped control of motors drawing up to 3 amps. This chip incorporates a 4-bit linear DAC, and an external DAC can be used if higher precision is required. As indicated by the data shown in Figure 5.4, a 4-bit linear DAC can produce 8 reasonably uniformly spaced microsteps, so this chip is a good choice for applications that exceed the power levels supported by the Allegro 3955.

6. Midlevel Control of Stepping Motors

Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

WARNING: This material is new but fairly stable

Introduction

All of the low-level motor control interfaces described in the previous sections are quite similar, at an abstract level. Each interface has some number of logic inputs. Some of these inputs may be used to directly control which switches are open or closed, others may be encoded, while others may control subsystems such as the analog to digital converter in a microstepping interface.
The states of each of these logic inputs is referred to as the *control vector* for the motor, and a sequence of states used to rotate the motor is referred to as a *control trajectory* for the motor.
For example, the control vector for controlling a permanent magnet or hybrid stepping motor using any of the control circuits shown in Figures 3.4, 3.5, 3.13, 3.14, 3.15, 4.7, 4.8, 4.10, 4.11 or 4.12 will contain 4 bits, 2 bits to control each motor winding. In each case, the control vector can be expressed as $<X_1Y_1X_2Y_2>$, where X_1 and Y_1 control the current through motor winding 1 and X_2 and Y_2 control the current through motor winding 2. For any interface with this control vector, the following trajectory will step the motor through one full electrical cycle, using full stepping:

1010
1001
0101
0110
1010

Similarly, the following trajectory will half-step motor through the same electrical cycle:

1010
1000
1001
0001
0101
0100
0110
0010
1010

Other controllers have different control vectors. For example, the control vector for a permanent magnet or hybrid motor controlled by a pair of Allegro 3952 chips (see Figure 4.9) will be $<B_1E_1P_1M_1B_2E_2P_2M_2>$, where B, E, P and M are the ¬BRAKE, ¬ENABLE, PHASE and MODE control inputs for each chip. In this case, the following control trajectory will full-step the motor through 1 electrical cycle:

10001000
10001010
10101010
10101000
10001000

The following control trajectory will half-step the same motor through one electrical cycle, using dynamic braking to control resonance whenever a motor winding is turned off:

10001000
10000000
10001010
00001010
10101010
10100000
10101000
00001000
10001000

It is worth noting that, while dynamic braking on disconnected motor windings is an excellent way to control resonance during low speed operation, this will reduce the available torque at higher motor speeds.

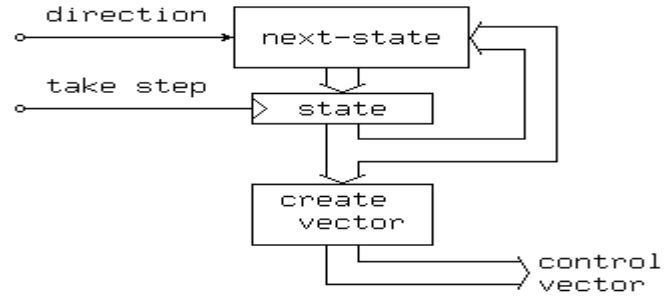
The control vectors required for microstepped motors are more complex, but the basic idea remains the same. The problem we face here is to develop higher level control systems that will generate appropriate control trajectories, moving the motor one step, half-step or microstep each time the higher level control system requires a move.

Hardware Solutions

Early solutions to the problem of generating appropriate control trajectories for stepping motors were almost always based on direct synthesis of the control trajectory in hardware. Such solutions are still appropriate for some applications, but these days, when programmable interface controller chips are commonly used to replace random low-speed logic and when most stepping motor applications are ultimately controlled by computer systems of one kind or another, it is common to use software to generate the control trajectory.

All hardware solutions to generating the control trajectory are subsumed by the general model illustrated in Figure 6.1:

Figure 6.1



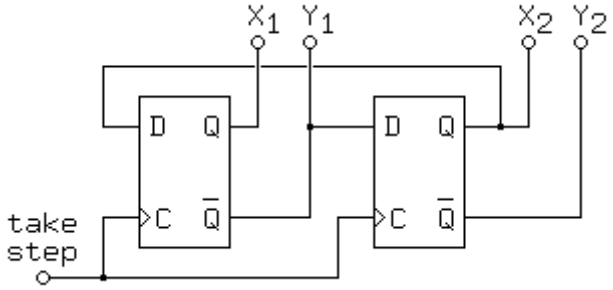
A general model for mid-level control

The *next-state* and *create-vector* blocks in Figure 6.1 are combinational logic functions, while the *state* block is a register. Depending on how the state is encoded, the *create-vector* function may be trivial; for certain applications, the *next-state* function is also trivial, but in the general case, *next-state* becomes an up-down counter while *create-vector* becomes a ROM holding the sequence of state vectors needed to form the control trajectories.
It is worth noting that some stepping motor control systems include addtional inputs to the mid-level control system. Common additions include *half-step* and *brake* inputs. Braking control is meaningless in full-step mode but it is of some use in half-step mode. In the latter mode, shorting unused motor windings at low speed is an excellent way to limit resonance while at higher speeds, unused motor windings should be left open for efficiency. Finally, some motor control systems include a *disengage* input that removes power from all windings; in this case, if *brake* is asserted, it typically shorts all motor windings.

Practical Examples

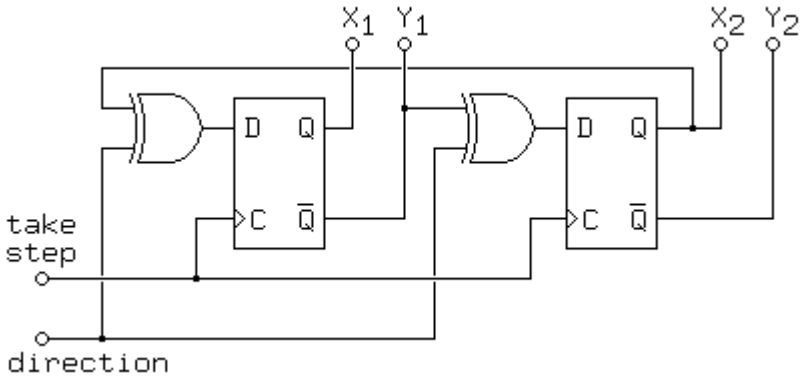
Amature astronomers frequently need very slow motors to turn their telescopes, and in recent years, stepping motors have taken the place of synchronous motors and gear trains for many applications, particularly for barn-door or Scotch mounts, a class of extremely simple camera mounts used in astrophotography. For this application, circuits as simple as that shown in Figure 6.2 are common:

Figure 6.2



The circuit shown in Figure 6.2 only operates in one direction, generating the signals needed to turn a permanent magnet or hybrid motor one full step for each pulse on the take-step input. In terms of the general model in Figure 6.1, the next-state and create-vector functions are trivial and require no logic to generate so long as each flipflop in the state register has outputs available in both straight and complemented form.
Modifying this circuit for bidirectional operation is straightforward, as is illustrated in Figure 6.3:

Figure 6.3



The circuit shown in Figure 6.3 uses two exclusive or gates to compute two different versions of the next-state function, depending on the value of the direction input. Each flip-flop presents both inverted and non-inverted outputs to the world; this allows an equivalent circuit to be made by substituting a double-pole double-throw switch for the exclusive-or gates, and another equivalent circuit can be derived from this by substituting a pair of 2-input 1-output multiplexors for the switch.

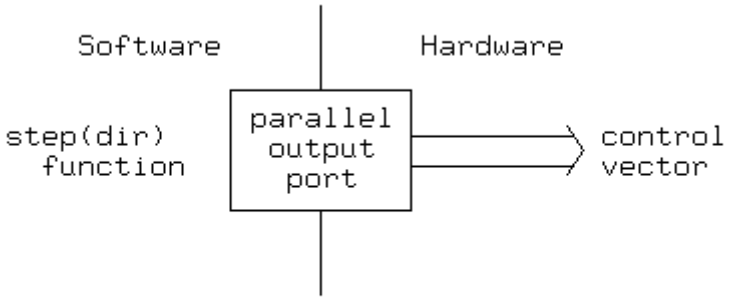
A number of manufacturers of stepping motor control circuitry make integrated circuits that include slightly more complex logic allowing both full and half stepping modes. For example, the Motorola

(and others) MC3479 chip includes a 16 volt 350mA H-bridge and control logic with step, direction, and mode inputs to control half-stepping.
The SGS-Thompson (and others) L297 includes just the mid-level control logic for full or half-step control of a pair of H-bridges used to control a permanent magnet or variable reluctance motor. This chip is specifically designed to control the L298 dual 2 Amp H-bridge, and in addition to the mid-level control logic, it includes the one-shots and comparators required to use this H-bridge as a current limited chopper.

Software Solutions

If a microprocessor, programmable interface controller or other computer system is used to control a stepping motor, the computer can directly generate the control vector in software and present it to the motor interface through a parallel output port, as shown in Figure 6.4:

Figure 6.4



A general model for mid-level control

In this software-centered model, the basic problem of mid-level control is reduced to how the *step* function is implemented. Each call to *step*(*d*) causes the control vector to advance along the control trajectory in the direction specified by dir. The call *step*(+1) causes the motor to step forward, while the call *step*(-1) causes it to take one step back.
The control trajectory for any motor can be described as a circular array of control vectors. The simple full-stepped trajectory given in the introduction can be represented as:

t, a 4 element array where

t[0] = 10
t[1] = 9
t[2] = 6
t[3] = 5

Similarly, the corresponding half-stepped trajectory for the same motor interface would be:

t, an 8 element array where

t[0] = 10
t[1] = 8
t[2] = 9
t[3] = 1
t[4] = 5
t[5] = 4
t[6] = 6
t[7] = 2

Similar definitions, differing only in the size of the trajectory array and the values of each entry, will suffice to describe one complete cycle of the control trajectory for any stepping motor interface!
The step routine itself does not depend in any way on the control vector. For all of the above examples, the step routine is:

p, a statically allocated integer variable, initially zero.

step(*d*)
-- a function of one argument *d*
-- where *d* must be either +1 or -1
-- no value is returned

p = *p* + *d* (mod *size*(*t*))
output(*t*[*p*])

Here, we assume that the *output* function sends one control vector to the motor. The details of this function will depend on the computer, the interface used, and the operating system, if any. The above pseudocode also assumes a *size* function that returns the size of the array holding a cycle of the the trajectory; how this is done will definitely vary from one programming language to another. Note that, when translating the above code into real programming languages, the simple use of the mod operator above can rarely be preserved. Mathematicians generally agree that $y > x \pmod{y} > 0$ but the designers of programming languages frequently depart from this definition when *x* is negative. As a result, in languages such as C, C++, Java and Pascal, care must be taken to avoid negative values on the righthand side of the mod (or %) operator!

Simple Practical Examples

The following C code will control a single 3-phase variable reluctance motor taking its control vector from the three least significant bits of the parallel port of a Unix or Linux system, assuming that the parallel port has been opened in the correct mode using the file descriptor `pp`:

```
#define STEPS 3
    int t[STEPS] = { 1, 2, 4 };

    int p = 0;

    void step( int d )
    {
        char c;
        p = (p + STEPS + d) % STEPS;

     /* output t[p] using low-level Unix I/O */
        c = t[p];
        write( pp, &c, 1 );
    }
```

Rewriting the first few lines of the above code allows us to convert it for use with a permanent magnet or variable reluctance motor operated in half-step mode from the 4 least significant bits of the parallel port:

```
#define STEPS 8
    int t[STEPS] = { 10, 8, 9, 1, 5, 4, 6, 2 };
```

An Object Oriented Design

Things are more complex if multiple motors are in use! Although ad-hoc solutions can are common, a systematic approach is appropriate, and even in languages with no support for object oriented methodology, the easiest way to describe such solutions is in object oriented terms.
After initialization, which may depend on many low level details, the high level software isn't interested in how the motor works. Therefore, for each motor object, the visible interface needs only the *step* function. The class of motor objects is polymorphic because the details of how *step* operates for one motor may differ considerably from the details of how it works for another.

The following code is given in Java; translation to C++, Simula 67 or other object oriented languages should be straightforward:

```
abstract class StepMotor
    {
        public abstract void step(int d);
            // step the motor in direction d (+1 or -1)
    }
```

This is an abstract class; that is, objects of this class cannot be instantiated because we have yet to specify any of the details of how the motor or interface works. Each useful stepping motor interface must be supported by an extension or subclass of this abstract class! The class below illustrates this, incorporating the code discussed in the sections above:

```
class UniversalStepMotor
extends StepMotor
    {
        private int s;          // the size of the trajectory
        private int[] t;        // the trajectory for this motor
        private int p;          // motor's position in trajectory
        private OutputStream o;// the output port to use

        public void step(int d)
            // step the motor in direction d (+1 or -1)
        {
            p = (p + s + d) % s;
            o.write( t[p] );
        }

        public UniversalStepMotor( int[] table, OutputStream out )
         // initializer
        {
         s = table.length;
            t = table;
            p = 0;
            o = out;

            o.write( t[p] );
        }
    }
```

The above code assumes that the interfaces to motors are accessible through output streams of type `java.io.OutputStream`, and the initializer not only builds the data structure but sends an initial control vector to the motor.

Given that `MotorPort` is an object of type `java.io.OutputStream`, the following Java code will create a `UniversalStepMotor` object `m` for a 3-phase variable reluctance motor:

```
StepMotor m = UniversalStepMotor(
        new int[] {4,2,1}, // step table for the motor
        MotorPort          // OutputStream for the motor
    );
```

With this declaration and initialization in place, the call `m.step(1)` will turn the motor one step.

For many applications, the class `StepMotor` defined above will need to be extended with a `reset` procedure, used to reset both the motor object and the motor interface. This procedure should probably be incorporated directly into the definition of the StepMotor class, as opposed to adding it by class extension. This procedure would typically be called as the first step in recovering from an error detected at higher levels in the control hierarchy.

When Objects Won't Do

For programmers who can't use object oriented design, for any reason, the following example illustrates an appropriate mid-level design that avoids the use of such features. Using Pascal, the following types can be used to describe each motor:

```
type
        direction = -1 .. +1;

        trajectory = array [0 .. MaxTrajectorySize] of int;

        StepMotor = record
            s: integer;    { size of this motor's trajectory }
            t: trajectory;
            p: integer;    { this motor's position in trajectory }
            o: port;       { the output port to use for this motor
}
        end { record };
```

The `StepMotor` record type given here corresponds exactly to an object of the class `UniversalStepMotor` defined in Java above. The array type trajectory is explicitly named so that a trajectory may be passed, as a formal parameter, to an initializer procedure for records of this type. The subrange type `direction` allows Pascal's subrange restriction mechanisms to check the correctness of parameters to the step function, a desirable feature that is missing in languages descended from C.

Given the above definitions, we can now construct a general purpose step procedure:

```
procedure step(var m: StepMotor, d: direction);
        { step motor m in direction d }
    begin
        with m do begin
            p = (p + s + d) mod s;
            output( o, t[p] );
        }
    end { step };
```

In all of the above, we have assumed that variables of the type `port` are used to describe output ports to which motors may be attached, and that the procedure `output` outputs one vector to the indicated port.

In effect, in abandoning programming languages with direct support for object orientation, we have had to chang our notation from: `motor.step( dir );`

to: `step( motor, dir );`

This is not a major sacrifice in a system where there is only one type of motor interface, but when there are multiple motor types, support for the object oriented model will be useful. If this must be done in a language like Pascal, the declaration for the type `StepMotor` must be modified to allow for all types of motor interfaces, for example, by declaring it to be a variant record, and the `step` procedure must be modified to check the motor type and act appropriately.

7. High Level Control of Stepping Motors

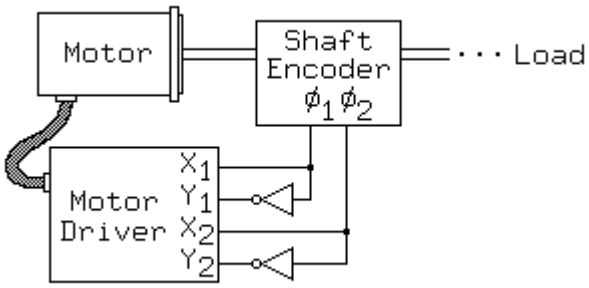
Part of Stepping Motors
by Douglas W. Jones
THE UNIVERSITY OF IOWA Department of Computer Science

Under Development

Introduction

The key question to be answered by the high-level control system for a stepping motor is, when should the next step be taken? While this almost always depends on the application, the similarities between different applications are sufficient to justify the development of fairly complex general purpose stepping motor controllers.
Stepping motor control may be based on *open loop* or *closed loop* models. We are primarily interested in open loop models, because this is where stepping motors excel, but we will treat closed loop models briefly because they are somewhat simpler. Figure 7.1 illustrates an extreme example:

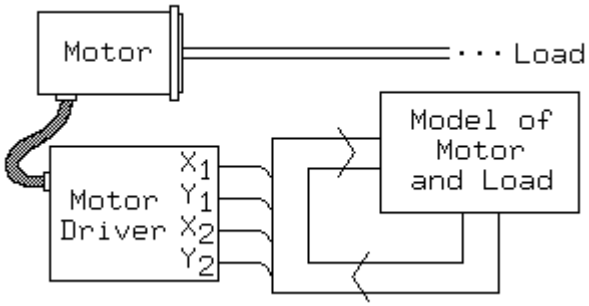
Figure 7.1



In Figure 7.1, a quadrature shaft encoder is attached to the drive shaft of a permanent magnet or hybrid stepping motor, and the two phase output of this encoder is used to directly generate the control vector for the motor driver. Rotary shaft encoders are typically rated in output pulses per channel per revolution; for this example to be useful, for a motor with n steps per revolution, the shaft encoder output must gives n/2 pulses per channel per revolution. If this is the case, the behavior of this system will depend on how the shaft encoder is rotated around the motor shaft relative to the motor.
If the shaft encoder is rotated into a position where the output of the shaft encoder translates to a control vector that holds the motor shaft in its initial position, the motor shaft will not rotate of itself, and if the motor shaft is rotated by force, it will stay wherever it is left. We will refer to this position of the shaft encoder relative to the motor as the neutral position.
If the shaft encoder is rotated one step clockwise (or counterclockwise) from the neutral position, the control vector output by the shaft encoder will pull the rotor clockwise (or counterclockwise). As the rotor turns, the shaft encoder will change the control vector so that the rotor is always trying to maintain a position one step clockwise (or

counterclockwise) from where it is at the moment. The torque produced by this method will fall off with rotor speed, but this control system will always produce the maximum torque the motor is able to deliver at any speed.
In effect, with this one-step displacement, we have constructed a brushless DC motor from a stepping motor and a collection of off-the-shelf parts. In practice, this is rarely done, but there are numerous applications of stepping motors in closed-loop control systems that are based on this model, usually with a microprocessor included in the feedback loop between the shaft encoder and the motor controller.
In an open-loop control system, this feedback loop is broken, but at a high level, the basic principle remains quite similar, as illustrated in Figure 7.2:

Figure 7.2



In Figure 7.2, we replace the shaft encoder from Figure 7.1 with a simulation model of the response of the motor and load to the control vector. At any instant, the actual position of the rotor is unknown! Nonetheless, we can use the simulation model to predict, based on an assumed rotor position and velocity, how the motor will respond to the control vector, and we can construct this model so that its output is the control vector generated by a simulated shaft encoder.
So long as the model is sufficiently accurate, the behavior of the motor controlled by this model will be the same as the behavior of the motor controlled by a closed loop system!

Model Variables

In the example given in Figure 7.1, the only control variable offered is the angle of the shaft encoder relative to the motor. In effect, this controls the extent to which the equilibrium point of the motor's torque versus shaft angle curve leads or follows the current rotor position. In theory, any desired motor behavior can be elicited by adjusting this angle, but it is far more convenient to speak in terms of other variables:
Θ -- The predicted shaft position (radians)
Θ_{target} -- The target shaft position determined by the application
$V = d\Theta/dt$ -- The predicted velocity (radians per second)
V_{target} -- The target velocity determined by the application
$A = dV/dt$ -- The predicted acceleration (radians per second squared)
A_{target} -- The target acceleration, may be determined by the application As in the section on Stepping Motor Physics, we will define the basic motor characteristics: S -- step or microstep angle, in radians
μ -- moment of inertia of rotor and load
h -- the holding torque of the motor Note that here, the step angle S is not the physical step angle of the motor, but rather, the step angle offered by the mid-level motor interface; this may be a full step, a half step, or a microstep of some size!

Models

The simplest model that will do the job is almost always the best. For some applications, this means the model is so simple that it is hard to identify it as a model! For example, consider the case where the application demands a constant motor velocity: $A_{target} = 0$
V_{target} = Constant In this case, $t_{step} = S / V_{target}$ where t_{step} -- time per step This barely looks like a model; in part, this is because we have omitted the statement that, every t_{step} seconds, we advance the control vector one step:

repeat the following cycle forever:

wait t_{step} seconds and then
$\Theta = \Theta + S$
step(1)

A more interesting model is required if we want to maintain a constant acceleration. Obviously, we can't do this forever, but we'll use this model as a component in more complex models that require changes of velocity or position. In this case, A_{target} = Constant where $A_{target} < h / \mu$ In developing a model, we begin with the observation that, for constant acceleration A and assuming a standing start at time 0, $\Theta = 1/2\ A\ t^2$ More generally, if the motor starts at position Θ and velocity V, after time t the new position Θ' and velocity V' will be: $\Theta' = 1/2\ A\ t^2 + V\ t\ \Theta$ $V' = A\ t + V$ Setting Θ=0 and Θ'=S, we solve first for t, the time taken to move one step, as a function of V and A:
$1/2\ A\ t^2 + V\ t - S = 0$
$t = (-V \pm (V^2 + 2\ A\ S)^{0.5}) / A$ Here, we have applied the quadratic formula, and for our situation, this gives two real roots! The additive root is the root we are concerned with; for this, we can use the resulting time to compute the velocity at the end of one step: $t_{step} = (-V + (V^2 + 2\ A\ S)^{0.5}) / A$
$V' = (V^2 + 2\ A\ S)^{0.5}$
We can combine this model for acceleration with the model for constant speed running to make a motor controller that will seek V_{target}, assuming that an outside agent may change V_{target} at any time:

repeat the following cycle forever:

if $V = V_{target}$ do the following:
wait S/V_{target} seconds and then
$\Theta = \Theta + S$
step(1)

otherwise, if $V < V_{target}$, accelerate as follows:
wait $(-V + (V^2 + 2\ A_{accel}S)^{0.5}) / A_{accel}$ seconds and then
$\Theta = \Theta + S$
$V = (V^2 + 2\ A_{accel}S)^{0.5}$
step(1)

otherwise, $V > V_{target}$, decelerate as follows:
wait $(-V + (V^2 + 2\ A_{decel}S)^{0.5}) / A_{decel}$ seconds and then
$\Theta = \Theta + S$
$V = (V^2 + 2\ A_{decel}S)^{0.5}$
step(1)

This control system is not fully satisfactory for a number of reasons! First, it only allows the motor to operate in one direction and it fails utterly when V reaches zero; at that point, if a divide by zero operation is allowed to produce an infinite result, the program will wait infinitely and never again respond to change in the control input.
The second shortcoming of this program is simpler to correct: As written, there is an infinitesimal probability of the motor speed reaching the desired speed and staying there with V_{target} equal to V. Far more likely, what will happen is that V will oscillate around V_{target}, taking alternate accelerating and decelerating steps and never settling down at the desired running speed.
A quick and dirty solution to this latter problem is to add code to recognize when V passes V_{target} during acceleration or deceleration; when this occurs, V can be set to V_{target}. Formally, this is incorrect, but if the acceleration and deceleration rates are not too high and if there is sufficient damping in the system, this will work quite well.
In a frictionless system using sine-cosine microstepping at speeds below the cutoff speed for the motor, the available torque is effectively constant and we can use the full torque to accelerate or decelerate the motor, so the above control algorithm will work with $A_{accel} = A_{decel} = h / \mu$ If there is significant static friction, we can take this into account as follows: $A_{accel} = (h - f) / \mu$
$A_{decel} = (h + f) / \mu$ where f -- frictional torque If the motor is run using the maximum available acceleration and decleration, any unexpected increase in the load will cause the motor rotor to fall behind its predicted position, and the result will be a failure of the control system. As a result, open-loop stepping motor control systems are never run at the accelerations give above! In the case of full or half-stepping, where there is no sine-cosine torque compensation, the available torque varies over a range of a factor of $2^{0.5}$, so we typically adjust the accelerations

given above by this amount: $A_{accel} = ((h / 1.414) - f) / \mu$
$A_{decel} = ((h / 1.414) + f) / \mu$ If we operate consistently near the edge of the performance envelope, and if we never request a velocity V_{target} near the resonant speed of the motor, we can safely accelerate through resonances without relying on damping. If, on the other hand, we select acceleration values that are significantly below the maximum that is possible, electrical or mechanical damping may be needed to avoid problems with resonance.
Note that it is not difficult to extend the above control model to account, at least approximately, for viscous friction and for the dropoff of torque as a function of speed. To do this, we merely modify the above formulas for A_{accel} and A_{decel} so that *h* and *f* are functions of V. Thus, instead of treating these as constants of the control algorithm, we must recompute the available acceleration at each step.
If our goal is to turn the motor smoothly from one set postion to another, we must first accelerate it, then perhaps coast at fixed speed for a while, then decelerate. The decision governing when to begin decelerating rests on a knowledge of the stopping distance from any particular velocity. Assuming that the available acceleration is constant over the relevant range of speeds, we can compute this from: $V = A_{decel} t$
$\Theta = 1/2 A_{decel} t^2$ First we solve for the stopping time, $t = V / A_{decel}$ and then we solve for the stopping angle $\Theta = 1/2 A_{decel} (V / A_{decel})^2 = V^2 / (2 A_{decel})$ Given this, we can outline a procedure for moving the motor from its current estimated position to a step just beyond some target position:

moveto(Θ_{target})

-- a function of one argument Θ
-- no value is returned

while $V < V_{target}$

and while $\Theta < \Theta_{target} - V^2 / (2 A_{decel})$ repeat the following to accelerate

step(1)
wait $(-V + (V^2 + 2 A_{accel} S)^{0.5}) / A_{accel}$ seconds and then
$\Theta = \Theta + S$
$V = (V^2 + 2 A_{accel} S)^{0.5}$

$V = V_{target}$

while $\Theta < \Theta_{target} - V^2 / (2 A_{decel})$ repeat the following to coast

step(1)
wait S/V_{target} seconds and then
$\Theta = \Theta + S$

while $\Theta < \Theta_{target}$ repeat the following to decelerate

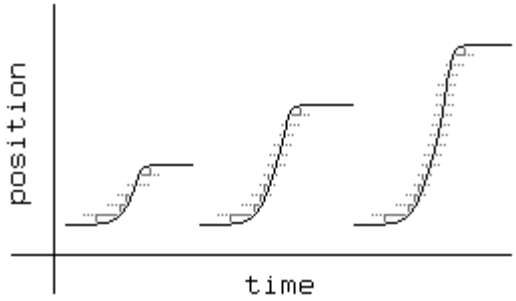
step(1)
wait $(-V + (V^2 + 2 A_{decel} S)^{0.5}) / A_{accel}$ seconds and then
$\Theta = \Theta + S$
$V = (V^2 + 2 A_{accel} S)^{0.5}$

$V = 0$

done, Θ and Θ_{target} are within a step of each other!

The control model only moves the motor one direction, it fails to plan in terms of the quantization of available stopping positions, and it doesn't account for the cyclic nature of Θ. Nonetheless, it is a useful illustration. Note that we have used V_{target} as a limiting velocity in this code, but that this will only be relevant during long moves; for short moves, the motor will never reach this speed.
With the above, code, so long as the acceleration and deceleration rates are high enough to avoid dwelling for too long at resonant speeds, and so long as V_{target} is not too close to a resonant speed, a plot of rotor position versus time will show fairly clean moves, as illustrated in Figure 7.3:

Figure 7.3

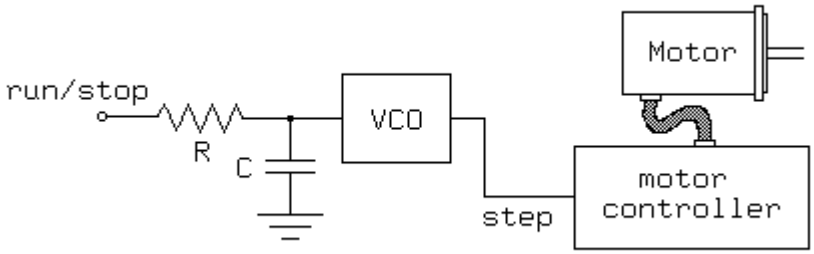


If the motor is to be accelerated at the maximum possible rate, the control model used above is not sufficient. In that case, during acceleration, the equilibrium position must be maintained between 0.5 and 1.5 steps ahead of the rotor position as the rotor moves, and during deceleration, the equilibrium position must be maintained the same distance behind the rotor position. This requires careful logic at the turnaround point, when the change is made from accelerating to decelerating modes. The above control model omits any such considerations, but it is adequate at accelerations sufficiently below the maximum available!

Hardware Solutions

Today, it is rare to find high-level stepping motor control done purely in hardware, and when it is done, it is usually only in the very simplest of applications. For example, consider the problem of starting and stopping a stepping motor under load. Direct generation of the quadratic functions necessary to achieve smooth acceleration is quite difficult in hardware, but it is easy to generate exponentials that are adequate approximations of these. The circuit outlined in Figure 7.4 illustrates how this can be done:

Figure 7.4



Here, the resistor R and capacitor C form a low pass filter on the control input of the voltage controlled oscillator VCO. When the input level is at run, the VCO output oscillates at its maximum rate. When the input level is at stop, the VCO output ceases to oscillate. The RC time constant of the low pass filter determines the rate of acceleration applied to the motor.
With such a design, the time constant RC is usually determined empirically by setting up the system and then adjusting R and C until the system operates properly.

Practical Examples

The NE555 timer can be used as a voltage controlled oscillator, but I first saw this done with discrete components on a controller for a paper-tape reader designed around 1970.

Software Solutions

The basic control models outlined at the start of this section can be directly incorporated into the software for controlling a stepping motor, and this must be done if, for example, the motor is driving a load with a variable moment of inertia or driving a load against variable frictional loadings. Most open-loop stepping motor applications are not that complicated, however! So long as the inertia and frictional loadings are constant, the control software can be greatly simplified, replacing complex model computations with a table of precomputed delays.
Consider the problem of accelerating the motor from a standing start. No matter where the motor starts, so long as the torque, moment of inertia and frictional loadings remain the same, the time sequence of steps will be the same. Therefore, we need only pre-compute this time sequence of steps and save it in an array. We can use this array as

follows to accelerate the motor: array AV, the acceleration vector, holds time intervals
i is the index into AV

i = 0

repeat the following cycle to accelerate forever:

wait AV[i] seconds and then
step(1)
i = i + 1

We may use i, the counter in the above code, as a stand-in for the motor velocity, since stepping the motor every AV[i] seconds will move the motor at a speed of S/AV[i].
It is a straightforward exercise in elementary physics to compute the entries in the array A. If the motor is accelerating at A_{target}, $\Theta_i = 1/2\ A_{target}\ t_i^2$ where Θ_i -- the shaft angle at each successive step Solving for time as a function of position, we get: $t_i = (2\Theta_i / A_{target})^{0.5}$ If we define $\Theta_0 = 0$ so that $\Theta_i = Si$ and $t_0 = 0$ we can conclude that $t_i = k\ i^{0.5}$ where $k = (2S/A_{target})^{0.5}$ The acceleration vector entries are then: $A[0] = (2S/A_{target})^{0.5}$ and $A[i] = (i^{0.5} - (i - 1)^{0.5})A[0]$ The following table gives the ratios of the first 20 entries in A[i] to A[0]:

0	1.000	10	0.154
1	0.414	11	0.147
2	0.318	12	0.141
3	0.268	13	0.136
4	0.236	14	0.131
5	0.213	15	0.127
6	0.196	16	0.123
7	0.183	17	0.120
8	0.172	18	0.116
9	0.162	19	0.113

In general, we aren't interested in indefinite acceleration, but rather, we are interested in accelerating until some speed or position restriction is satisfied, and then the control system should change, for example, from acceleration to deceleration or constant speed operation. So long as friction can be ignored, so the same rates can be used for acceleration and deceleration, we can make a clean move to a target position as follows: array AV is the acceleration vector
i is the index into AV

i = 0

$D = \Theta_{target} - \Theta$

while D nonzero do the following

if D > 0 -- spin one way
step(1)
D = D - 1
else -- spin the other way
step(-1)
D = D + 1
endif
wait AV[i] seconds and then
if (i < |D|) and ($S/AV[i] < V_{target}$) -- accelerate
i = i + 1
else if i > |D| -- decelerate
i = i - 1
endif

endloop

Given an appropriate acceleration vector, the above code will cleanly accelerate a motor up to a speed near the target velocity, hold that speed, and then decelerate cleanly to a stop at the target position.
The above code does not take advantage of the higher rates of deceleration allowed when there is friction. In general, this should not cause any problems, but if the fastest possible moves are desired, a separate deceleration table should be maintained. Here is one idea: array AV holds acceleration intervals
array C holds coasting intervals
array T holds transition information
array D holds deceleration intervals

```
i = 0

repeat the following until the desired speed is reached
    wait A[i] seconds and then
    step( 1 )
    i = i + 1

repeat the following to maintain the speed
    wait C[i] seconds and then
    step( 1 )

repeat the following to maintain the speed

i = i - T[i]

repeat the following until i = 0
i = i - 1
step( 1 )
```

In the above, the arrays A and D are constructed identically, except that one has intervals used for acceleration, at a rate limited by friction, while the other has intervals used for deceleration, at a rate assisted by friction. Note that, after accelerating for i steps from a standing start, the motor will reach a velocity from which it can decelerate to a halt in i-T[i] steps. This relationship determines the values pre-computed in the array T.

Stepping Motor Control Software

Old Part 5 of Stepping Motors *by* Douglas W. Jones

Here's the code to make your motor run as if you had one of those fancy stepper controllers. I've used Pascal for no particular reason. This code assumes only one motor, and it assumes it's attached to the least significant bits of one parallel output port. In practice, it's nice to have one parallel output port per motor, although with a bit of care, you can use the high bits of a port for another motor or other applications, and you can multiplex one port to handle multiple motors. (The July 1993 issue of Model Railroader has plans for a parallel port multiplexer circuit for IBM PC systems in it).

Assume these declarations and values for a three winding variable reluctance motor: `const maxstep = 2;`

```
        steps = 3;
var     steptab: array [0..maxstep] of integer;
        step: integer;
        motor: file of integer; { this is the I/O port for the motor }
begin
        step := 0;
        steptab[0] = 1; { binary 001 }
        steptab[1] = 2; { binary 010 }
        steptab[2] = 4; { binary 100 }
        write( motor, steptab[step] );
```

Assume these declarations and values for a permanent magnet motor, either unipolar, with center tapped windings, or bipolar, with H-bridge drive circuits: `const maxstep = 3;`

```
        steps = 4;
var     steptab: array [0..maxstep] of integer;
        step: integer;
        motor: file of integer; { this is the I/O port for the motor }
begin
        step := 0;
        steptab[0] = 1; { binary 0001 }
        steptab[1] = 4; { binary 0100 }
        steptab[2] = 2; { binary 0010 }
        steptab[3] = 8; { binary 1000 }
        write( motor, steptab[step] );
```

Assume these declarations and values for half-step control of a permanent magnet motor: `const maxstep = 7;`

```
        steps = 8;
var     steptab: array [0..maxstep] of integer;
        step: integer;
        motor: file of integer; { this is the I/O port for the motor }
begin
        step := 0;
        steptab[0] = 1;  { binary 0001 }
        steptab[1] = 5;  { binary 0101 }
        steptab[2] = 4;  { binary 0100 }
        steptab[3] = 6;  { binary 0110 }
        steptab[4] = 2;  { binary 0010 }
        steptab[5] = 10; { binary 1010 }
        steptab[6] = 8;  { binary 1000 }
        steptab[7] = 9;  { binary 1001 }
        write( motor, steptab[step] );
```

Assume these declarations and values for control of a 5-phase motor, with an H-bridge on each of the 5 leads to the motor: `const maxstep = 9;`

```
        steps = 10;
```

```
var   steptab: array [0..maxstep] of integer;
      step: integer;
      motor: file of integer; { this is the I/O port for the motor }
begin
      step := 0;
      steptab[0] = 13;  { binary 01101 }
      steptab[1] = 9;   { binary 01001 }
      steptab[2] = 11;  { binary 01011 }
      steptab[3] = 10;  { binary 01010 }
      steptab[4] = 26;  { binary 11010 }
      steptab[5] = 18;  { binary 10010 }
      steptab[6] = 22;  { binary 10110 }
      steptab[7] = 20;  { binary 10100 }
      steptab[8] = 21;  { binary 10101 }
      steptab[9] = 5;   { binary 00101 }
      write( motor, steptab[step] );
```

The remainder of the code is the same and doesn't depend on the motor. The following procedure will advance the motor one step in either direction, where the direction parameter must be either +1 or -1 to indicate the direction.

```
procedure onestep( direction: integer );
begin
    step := step + direction;
    if step > maxstep then step := 0
    else if step < 0 then step := maxstep;
    write( motor, steptab[step] );
end;
```

Software control of a stepping motor is a real-time task, and you need at least a bit of feedback. One bit is enough; typically, this will be a bit indicating that a cam on the turntable (or whatever the motor is driving) is interrupting a light beam or closing a microswitch. To avoid hysteresis problems in reading the position from this cam, you should only read zero to one transitions as indicating the home position when the motor is spinning in one direction. Especially with switches or where gear trains are involved between the motor and the turntable, the one to zero transition in the other direction won't usually occur at exactly the same position.

Given that you can read the sense bit and that you have a programmable interval timer interrupt on your system, it is easy to make the timer interrupt service routine operate the motor as follows:

```
const maxpos = 11111;
{ maxpos + 1 is calls to onestep per rev }
var position: integer; { current position of motor }
    destination: integer; { desired position of motor }
    direction: integer; { direction motor should rotate }
    last: integer; { previous value from position sensor }
    sensor: file of integer; { parallel input port }
begin
    read( sensor, last );
    position := 1;
    setdest( 0, 1 ); { force turntable to spin on power-up until
                       it finds it's home position }

procedure timer; { interval timer interrupt service routine }
var sense: integer;
begin
    read( sensor, sense );
    if (direction = 1) and (last = 0) and (sense = 1)
      then position = 0;
    last := sense;

    if position <> destination then begin
        onestep( direction );
        position := position + direction;
```

```
            if position > maxpos then position := 0
            else if position < 0 then position := maxpos;
        end;

        if position <> destination
          then settimer( interval_until_next_step );
    end;
```

The following procedure is the only procedure that user code should call. This procedure sets the destination position of the turntable and sets the direction of rotation, then sets the interval timer to force an immediate interrupt and lets the timer routine finish rotating the turntable while the applications program does whatever else it wants.

```
procedure setdest( dst,dir: integer );
    begin
        destination := dst;
        direction := dir;
        if position <> destination
          then settimer( min_interval ); { force a timer interrupt }
    end;
```

If you want to control multiple stepping motors, it is easiest if you have one interval timers and one parallel port per motor. If you hardware has only one timer, then you can use it to simulate multiple interval timers, but this is most of the way to the job of writing a real-time executive.

A final note: If you try to step a motor too fast, it will slip and your software will lose track of the motor position. Motors typically come with a rating that indicates a maximum number of steps per second, but you may not be able to accelerate the motor to that number of steps per second from a dead start without running it at a lower speed first. This is especially true if the inertia of the load is fairly large, and in fact, with appropriate acceleration sequences, you can usually excede the maximum rated speed.

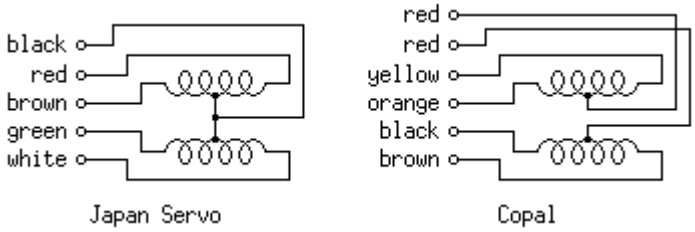
In the above code, interval_until_next_step is shown as a constant. If you are dealing with high-inertia loads or very short intervals, you'll have to make this a variable, using longer intervals during starting and stopping to take care of accelerating and decelerating the motor.

A Worked Stepping Motor Example

Old Part 6 of Stepping Motors *by* Douglas W. Jones

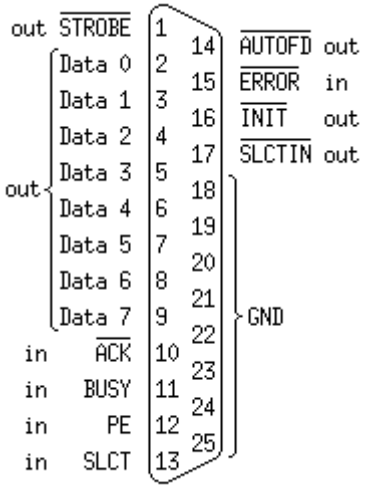
Perhaps some of the most commonly available stepping motors, for the experimenter, are the head positioning motors from old diskette drives. These can be found at electronics swap meets, in computer surplus outlets, and even in trash dumpsters. In addition to a stepper, a typical full-height 5.25 inch disk drive includes a 12 volt DC motor with tachometer and motor control circuit board, two microswitches, and a matched LED-photosensor pair.
A common stepper to find in a full-height IBM or Tandon 5.25 inch diskette drive is the type KP4M4 made by either Japan Servo Motors or Tandon; this is a permanent magnet motor with 3.6 degrees per step and 150 ohms per winding, with the center-taps of each winding tied to a common lead. Many half-height 5.25 inch diskette drives use very similar motors, although there is much more variety in newer drives, including some that use bipolar steppers.
Another stepper sometimes found in half-height drives is the `pancake format' motor from a 1/2 height 5.25 inch diskette drive; for example, a permanent magnet motor made by Copal Electronics, with 1.8 degrees per step and 96 ohms per winding, with center taps brought out to separate leads. The leads on these motors are brought out to multipin in-line connectors, laid out as follows:

Figure 6.1



When the center-taps of these motors are connected to +12 and one end of either winding is grounded, the winding will draw from 170 mA to 250 mA, depending on the motor, so any of a number of motor drive circuits can be used. The original IBM full-height diskette drives used a pair of UDN3612N or UDN5713 chips; these are equivalent to chips in the SN7547X series (X in 1,2,3). The ULN2003 darlington arrays from Allegro Microsystems is probably the most widely available of the applicable chips, so it will be used in this example.
Consider the problem of controlling multiple steppers comparable to those described above from an IBM compatible DB25-based parallel output port. The pinout of this connector is given in Figure 6.2, as seen looking at the face of the female connector on the back of an IBM PC (or equivalently, as seen from the back of the male connector that mates with this):

Figure 6.2



The IEEE 1284 standard gives the best available definition of the parallel port, but as an after-the-fact standard, nonconformance is common. Some documentation of this standard is available in the net. There is an extensive set of tutorial material available on the web discussing the IBM PC Parallel port. Another index of parallel port information is available from Ian Harries. There is some confusion in the documentation of this connector about the labels on the

SLCT and SLCTIN lines (pins 13 and 17); this is because these names date back to a Centronics printer design of the early 1970's, and the name SLCTIN refers to an input to the printer, which is to say, an output from the computer. The names of some of these lines are relics of the original intended purpose of this I/O port, as a printer port. Depending on the level at which you wish to use the printer, some may be ignored. If the BIOS printer support routines of the IBM PC or the parallel port driver under various versions of UNIX sare to be used, however, it is important to pay attention to some of these signals:
The BIOS handles reinitializing the printer by producing a negative pulse on INIT (pin 16). We can use this as a reset pulse, but otherwise, it should be ignored! In the reset state, all motor windings should be off.
When no output activity is going on, the BIOS holds the output signal lines as follows:

- `STROBE` (pin 1) high, data not valid.
- `AUTOFD` (pin 14) high, do not feed paper.
- `INIT` (pin 16) high, do not initialize.
- `SELCTIN` (pin 17) low, printer selected.

To print a character, the BIOS waits for BUSY (pin 11) to go low, if it is not already low, and then outputs the new data (pins 2 through 9). Following this (with a delay of at least 0.5 microsecond), STROBE (pin 1) is pulsed low for at least 0.5 microsecond. The BIOS returns the inputs ACK, BUSY, PE and SLCT (pins 10 to 13) to the user program after printing each character.
The computer is entitled to wait for ACK (pin 10) to go low for at least 5 microseconds to acknowledge the output strobe, but apparently, IBM's BIOS does not do so; instead, it relies on device to set BUSY to prevent additional outputs, and it leaves the output data unmodified until the next print request. While neither MS/DOS nor the BIOS use the interrupt capability of the parallel port, OS/2 and various versions of UNIX use it. A rising edge on ACK (pin 10) will cause an interrupt request if interrupts are enabled. This interrupt design is only useful if, in normal operation, the trailing edge of the ACK pulse happens when BUSY falls, so that finding BUSY high upon interrupt signals an error condition.
Note that all input output to the parallel port pins is done by writing to various I/O registers; so as long as interrupts are disabled and the I/O registers are directly manipulated by application software, all 5 input pins and all 12 output pins may be used for any purpose. To maintain compatibility with existing drivers, however, we will limit our misuse of these pins.
If we only wanted to support a single motor, it turns out that the logic on the standard 5.25 inch diskette drive can, with an appropriate cable, be driven directly from the parallel port. Documentation on this approach to recycling motors from old diskette drives has been put on the web by Tomy Engdahl.
Since we are interested in supporting multiple motors, we will use DATA lines 4 to 7 to select the motor to control, while using the least significant bits to actually control the motor. In addition, because almost all stepping motor applications require limit switches, we will use the PE (12) bit to provide feedback from limit switches. The IEEE 1284 standard defines the PE, SLCT and ERR signals as user defined when in either Enhanced Parallel Port or Compatibility mode, so this is not a bad choice. Unfortunately, the BIOS occasionally checks this bit even when it is aware of no printer activity (for example, when the keyboard type-ahead buffer fills); thus, it is a good idea to disable the BIOS when the parallel port is used for motor control!
Note that fanout is not a problem on the IBM PC parallel port. The IEEE 1284 standard defines the parallel port outputs as being able to source and sink 14 milliamps, and older IBM PC parallel ports could sink about 24 milliamps. Given that a standard LS/TTL load sources only 0.4 milliamps and some devices (comparitors, for example) source 1.2 milliamps, an IEEE 1284 port should be able to handle up to 10 of motor interfaces in parallel.

A Minimal Design

As mentioned above, we will use the ULN2003 darlington array to drive the stepping motor. Since we want to drive multiple motors, the state of each motor must be stored in a register; while many chips on the market can store 4 bits, careful chip selection significantly reduces the parts count and allows for a single-sided printed circuit card!
With appropriate connections, both the 74LS194 and the 74LS298 can use a positive enable signal to gate a negative clock pulse; we will use the 74LS194 because it is less expensive and somewhat simpler to connect. Finally, we will

use the 74LS85 chip to compare the 4 bit address field of the output with the address assigned to the motor being driven.
Figure 6.3 summarizes the resulting design:

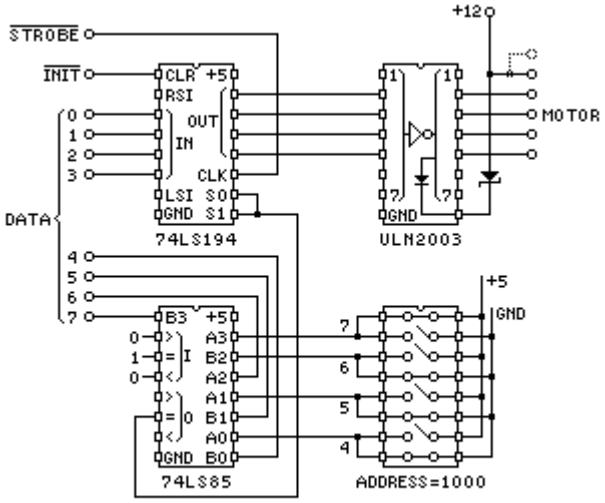


Figure 6.3

The 74LS194 was designed as a parallel-in, parallel-out shift-register with inputs to select one of 4 modes (do nothing, shift left, shift right, and load). By wiring the two mode inputs in parallel, we eliminate the shift modes, converting the mode input to an enable line. The unused right and left shift input pins on this chip can remain disconnected or can be grounded, tied to +5, or connected to any signal within the loading constraints.
Here, we show the 74LS194 being loaded only when bits 4 to 7 of the output data match a 4-bit address coded on a set of address switches. The comparison is done by a 74LS85, and the address switches are shown in Figure 6.3 as an 8-position DIP-switch. The cost of a DIP switch may be avoided in production applications by substituting jumpers.
One interesting aspect of this design is that the LS-TTL outputs driving the ULN2003 chip are used as current sources -- they pull up on the inputs to the darlington pairs. This is a borderline design, but careful reading of the LS-TTL spec sheets suggests that there is no reason it should not work, and the ULN2003 is obviously designed to be driven this way, with more than enough forward current gain to compensate for the tiny pull-up capacity of an LS-TTL output!
The Zener diode connected between pin 9 of the ULN2003 and the +12 supply increases the reverse voltage on the motor windings when they are turned off. Given the 50 volt maximum rating of the ULN2003, this may drop as much as 50-12 or 38 volts, but note that power dissipation may actually be an issue! At high frequency with unconventional control software, the power transfer to this diode can be quite efficient! With the stepping motors from old diskette drives, it may be possible to push a 12 volt zener to on the order of 1 watt. I used a 15 volt 1 watt zener, 1N3024.
If this motor is to be driven by software that directly accesses the low-level parallel port interface registers, directly loading data and then directly raising and lowering the strobe line, no additional hardware is needed. If it is to be driven through the BIOS or higher level system software, additional logic will be needed to manipulate ACK and BUSY.
Although the 74LS85 and the 74LS194 are no longer stocked by some mass-market chip dealers, they are still in production by Motorola, TI, Thompson SK and NTE. If over-the-counter availability of chips is your primary concern, adding a chipload of inverters or 2-input nand gates will allow just about any 4-bit latch to be used for the register, and address decoding can be done by a quad XOR chip and a 4-input nand gate.
If over-the-counter chips are of no concern, you can reduce the chip count even further! The Micrel MIC5800 4-bit parallel latch/driver can easily handle the loading of small unipolar steppers and combines the function of the ULN2003 and the 74LS194 used here! The resulting design is very clean.

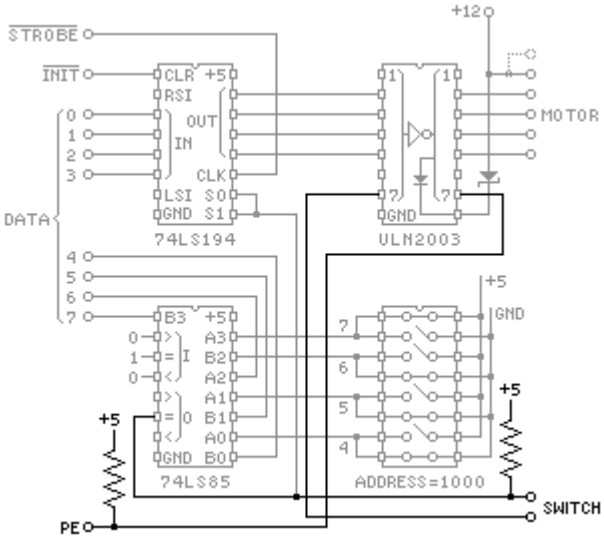
Adding One Bit of Feedback

Surprisingly, no additional active components are needed to add one bit of feedback to this design! There are 3 spare darlington pairs on the ULN2003 driver chip, and, with a pull-up resistor for each, these can serve as open-collector inverters to translate one or two switch settings into inputs appropriate for the PC!

The ULN2003 includes pull-down resistors on each input, guaranteeing that the corresponding output will be turned off if left disconnected. Thus, connecting a ULN2003 input to a positive enable signal (pin 5 of the 74LS85 chip for example) will turn the output on only if it is both enabled and the switch is closed. It may be necessary to add a 1K pull-up to the LS-TTL output because, in addition to driving the ULN2003, it is also driving two normal LS-TTL inputs. Adding this pull-up will do no harm if it isn't needed (an LS-TTL output should be able to handle even a 300 ohm pull-up).

Since the ULN2003 is an open-collector inverter, the output needs a pull-up. We could rely on the PE input of the IBM PC parallel port to pull this line up (an open TTL input usually rises of its own accord), but it is better to provide a pull-up resistor somewhere. Here, we provide a 10K pull up on each stepping motor drive card; these pull-ups will be in parallel for all motors attached to a single parallel port, so if fewer than 10 motors are in use, proportionally smaller resistors may be substituted or the pull-ups may be omitted from some of the controller cards. Figure 6.4 summarizes these additions to the design:

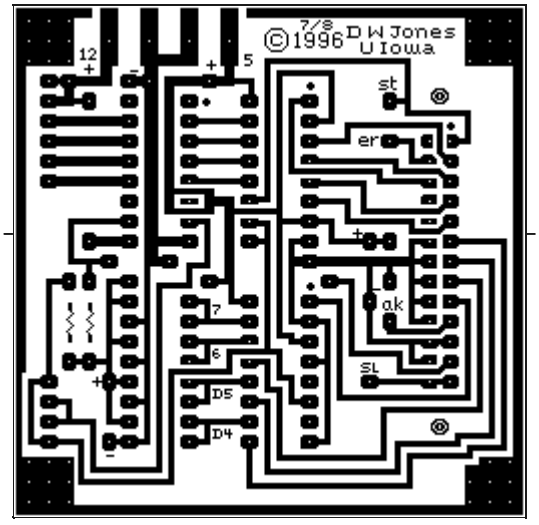
Figure 6.4



Something You Can Build

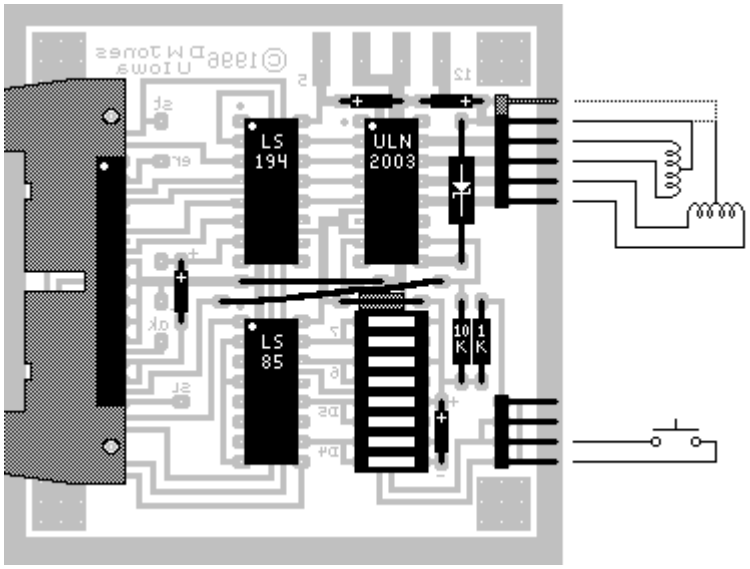
Figure 6.5 shows a single-sided PC board layout for a 2.5 inch square board that incorporates all of the ideas given above. I have etched and tested the 7/8/1996 version of this artwork.

Figure 6.5



(What's that about copyright notices? Well, put simply, if you're going to sell my design, please get in touch with me about it. You're free, however, to make a handful of boards from this design to control your own motors.)
This version of the board includes a jumper to ground the BUSY signal on the parallel port (pin 11) and it brings out the STROBE, ERROR, ACK and SELECT signals to allow for possible jumpering. These changes make it slightly more likely that this board can be hacked to work with native operating system drivers for the parallel port. As is, with no special jumpering other than the grounding of BUSY, it works under Linux.
To use the artwork in Figure 6.5 for etching a board, reproduce it at 100 pixels per inch (a slightly finer version, using 150 pixels per inch, is also available). Both versions are positive images of the foil side of the board; depending on how you transfer the image to the resist for etching, you may need to flip it left-to-right and/or invert the black and white. Most GIF viewers allow for such transformations.
Figure 6.6 shows the component side of this board, with jumpers and parts in place.

Figure 6.6



Note that this layout does not show mounting holes for the board, but that 4 corner pads are provided in appropriate locations. The layout also doesn't show a power connector, but the standard 4-pin Molex connectors used with 5.25" diskettes will fit the pads provided. The 26 pin header is wired so that it can be directly wired to a 25 pin DB-25 plug using ribbon cable and insulation displacement connectors. If multiple motors are to be used, a single ribbon cable can be made with multiple 26 pin connectors on it, one per motor.
Figure 6.6 shows 4 capacitors, 3 between +5 and ground, and 1 between +12 and ground. The two capacitors farthest from the power input connector can be .01 or .1 microfarad capacitors; it may be better to use larger capacitors at the power input pins, 1 microfarad or more.
The plug from the stepping motor goes on the the top header shown in Figure 6.6, with the center-tap lead(s) topmost. The board is arranged so that either a 5 or 6 pin header may be used here, compatable with the plugs on the motors shown in Figure 6.1. The limit switch goes on the bottom header. The latter is wired so that it can be used with either microswitch salvaged from a full-height Tandon 5.25 inch diskette drive.
The address may be hard-wired using 4 jumpers, or it may be set using a DIP-switch, as shown in Figure 6.6. Each bit of the address is set by two switches or jumper positions, the topmost pair of these sets the most significant of the 4 bits. To set a bit to 1, close the top switch and open the bottom switch in the corresponding pair (or put the jumper in the top position of the pair); to set a bit to 0, close the bottom switch and open the top one (or put the jumper in the bottom position of the pair). If it is at all likely that someone will change the address switches while power is applied to the board, use a 47 ohm resistor in place of the jumper directly above the switches! This will protect your power supply from the accidental short circuits that can result from improper switch settings.

Testing the Board

Under Linux

The standard Linux line printer driver attaches the device `/dev/lp0` to the standard parallel port found on most PCs, and `/dev/lp1` and `/dev/lp2` to the optional additional parallel ports. The line printer driver has many operating modes that may be configured and tested with the `tunelp` command. The default mode works, and the following `tunelp` command will restore these defaults: `tunelp /dev/lp0 -i 0 -w 0 -a off -o off -c off`
This turns off interrupts with `-i 0` so that the board need not deal with the acknowledge signal, and it uses a very brief strobe pulse with `-w 0`, sets the parallel port to ignore the error signal with `-a off`, ignores the status when the port is opened with `-o off`, and does not check the status with each byte output with `-C off`. The settings of the tunelp options `-t` and `-c` should not matter because this interface is always ready and thus polling loop iteration is never required.

Given a correctly configured printer port, the C routines allow output of motor control bytes to the port: `/****`

```
 *  ppaccess.c                                          *
 *  package for Linux access to the parallel port *
                                ****/
#include <fcntl.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <linux/lp.h>
#include <stdio.h>

#define paper_empty 0x20

static int pp; /* parallel port device descriptor */

void ppsetup()
/* setup access to the parallel port */
{
    pp = open("/dev/lp0",O_WRONLY, 0);
    if (pp < 0) {
     printf( "can't open, %s\n",
          _sys_errlist[errno] );
           exit( 0 );
       }
}

int ppput(int b)
/* put b to the parallel port, return the status */
{
       char cbuf = (char) b;
       int ibuf;
    /* begin critical section */
       if(write( pp, &cbuf, 1 ) != 1){
           printf( "write error, %s\n",
          _sys_errlist[errno] );
     exit( 0 );
       }
    if(ioctl( pp, LPGETSTATUS, (char*)&ibuf ) < 0){
           printf( "/dev/lp0 ioctl error, %s\n",
          _sys_errlist[errno] );
```

```
        exit( 0 );
          }
        /* end critical section */
        return ibuf & paper_empty;
    }
```

The comments above indicate a critical section! This is only a concern if the parallel port is shared by multiple processes, perhaps with one process in charge of each of a number of motors. In that case, a semaphore must be used to guard the critical section (see the UNIX semctl() kernel call), or a file lock using F_LOCK and F-ULOCK fcntl() commands, preferably through the lockf() function with a very large size. The latter approach (file locking using lockf()) is probably preferable.

Given these routines, the following main program should spin the motor at a few steps per second in a direction that depends on the sense switch setting:

```
/****
 *  spin.c
 *  Linux program to spin a permanent magnet motor *
    the sense switch gives forward/reverse control *
                                ****/
#include <unistd.h>

#define motor_number 15
#define microsec     1
#define millisec     (1000*microsec)

int steptab[] = { 9, 10, 6, 5 };

main()
{
    int step = 0;
    ppsetup();
    for (;;) {
     if (ppput(steptab[step]|motor_number<<4)) {
         step = (step + 1)&3;
     } else {
         step = (step - 1)&3;
     }
     usleep( 100 * millisec );
    }
}
```

Under Qbasic

To test your board, use the following little basic program. This was developed under Microsoft's Qbasic (C:DOS/QBASIC on a typical off-the-shelf PC) under bare MS/DOS (no version of windows running). This code has been tested with the prototype hardware for the design given above.

The first subroutine in this program outputs the data D to motor M attached to printer port P, and reads the status into S.

```
100 OUT P, D + (16 * M)
OUT P + 2, &HD
OUT P + 2, &HC
S = INP(p + 1)
RETURN
```

The second subroutine updates D, the motor control output, to rotate the motor one step, then uses the first subroutine to output D.

```
200 IF D = 9 THEN
    D = 10
ELSEIF D = 5 THEN
    D = 9
ELSEIF D = 6 THEN
    D = 5
```

```
        ELSEIF D = 10 THEN
            D = 6
        END IF
        GOSUB 100
        RETURN
```

The main program connects the second subroutine to the real-time clock and uses it to step the motor once per second, then repeatedly prints the status reported by the motor.

```
        P = &H378
        M = 15
        D = 10
        ON TIMER(1) GOSUB 200
        TIMER ON
    305 LOCATE 5, 5
        PRINT "STATUS: "; S; "  "
        GOTO 305
```

Unfortunately, QBASIC doesn't give you access to the high resolution of the hardware real-time clock, so this prototype code is only good for testing the hardware. While this code is running, the status of the microswitch (if connected) will be displayed on the screen, embedded in the rest of the I/O port status word.

Other Sources of Information

Web Sites

Other Motor Control Web Pages

- Advanced Micro Systems Stepper Motor Basics
  an excellent tutorial from a maker of motors and controllers.
- motioncontrol.com
  a commercially operated gateway to motion control resources on the web
- Ian Harries on Stepping Motors
  with a nice set of information on reverse engineering salvaged motors and a number of example applications.
- Euclid Research MotionScope demo
  excellent illustrations of physical behavior of some real motors.

Motor Manufacturers

- Advanced Micro Systems (1.8 degree per step, large permanent magnet motors)
- Astrosyn. (UK)
- Donovan Micro-Tek Inc. (very small motors)
- Eastern Air Devices Inc. (midsize motors and linear actuators)
- Gunda Electronic GmbH (German) (Google's English translation)
- Haydon Switch and Instrument, Inc.
- IntelLiDrives (high-resolution linear and 2-d planar stepping motors)
- Lin Engineering (100 to 800 steps per revolution)
- MicroMo Electronics (very small motors)
- Mitsumi (Japan)
- Phytron, Inc. (motors and controllers)
- Portescap Inc.
- Shinano Kenshi Corp. (SKC)
- Micro Precision Systems (remarkably small motors and controllers)

Controllers

- Advanced Micro Systems
- Astrosyn. (UK)
- Advanced Micro Systems Inc.
- Alzanti Limited (UK)
- Arrick Robotics
- Control Technology Corporation
- E-Lab Digital Engineering, Inc.

- GreenSpring Computers
- Magna Associates Inc.
- Netmotion
- StepperControl.com

Distributors

- ACP&D Limited (UK) (UK version) (distributor for COLIBRI integrated motor/controllers and maker of COBRA linear and planar stepping motors)
- Alzanti Limited (UK)
- Electro Sales Inc. (northeast USA)
- Flexible Technologies, Inc. (Southwest USA)
- MESA Systems Co. (USA) (distributor for COLIBRI integrated motor/controllers)
- Motionex (southeast USA)
- Smart Motion Control Inc. dba ABC Motion Control (northeast USA)
- Technovation Systems Ltd. (UK)

Surplus and Hobbyist Suppliers

- ALL Electronics (new and surplus)
- DIY Electronics (kits, Hong Kong)
- EIO's Stepper Motor Page (surplus)
- PC Gadgets (the Gadgetmaster interface)
- Hi-Tech Surplus
- Surplus Center (mostly heavy industrial surplus, Nebraska)
- Vorlac (Surplus, australia)
- Wirz Electronics (Hobbyist oriented, controllers)

Motor Design, Selection and Prototype Fabricaton Services

- Yeadon Engineering Services, yes@up.net (Michigan)
  YES is the contact for the Small Motor Manufacturer's Association.

Other Web Pages

- The Art of Motion Control;
  Bruce Shapiro's stepper-controlled machine-shop and artist's studio.
- EIO's Stepper Motor Page;
  A surplus dealer, but listed here because of their extensive index of information about stepping motors.
- Schmitz Engineering Liaison
  A rotary shaft position encoder distributor offering consulting services on encoder use. Roger Schmitz wrote Encoder Output Choices for System Designers for MOTION Magazine.

Books

Handbook of Small Electric Motors
William H. Yeadon and Alan W, Yeadon, eds.
McGraw-Hill, c2001.
LC number: TK2537 .H34 2001

Stepping motors: a guide to modern theory and practice
Acarnley, P. P.
P. Peregrinus on behalf of the IEE, 1984, c1982.
LC number: TK2537 .A28 1984
A third edition has recently been released.

Stepping motors and their microprocessor controls
Kenjo, Takashi
Oxford University Press, c1984.
LC number: TK2785 .K4 1984