5.3 Light-Emitting Diodes

Light-emitting diodes (LEDs) are two-lead devices that are similar to *pn*-junction diodes, except that they are designed to emit visible or infrared light. When a LED's anode lead is made more positive in voltage than its cathode lead (by at least 0.6 to 2.2 V), current flows through the device and light is emitted. However, if the polarities are reversed (anode is made more negative than the cathode), the LED will not conduct, and hence it will not emit light. The symbol for an LED is shown below.

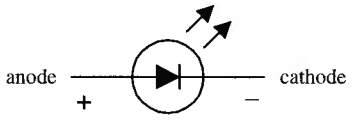


FIGURE 5.5

Unlike lamps, which give off a wide range of different colored photons—yielding white light—LEDs only give off a particular color photon. Typical emitted photon colors are red, yellow, green, and infrared. In terms of applications, LEDs are frequently used as indicator lights for displays or for low-level lighting applications (e.g., bicycle signal lights). Often, LEDs (especially infrared LEDs) are used as transmitting elements in remote-control circuits (e.g., TV remote control). The receiving element in this case may be a phototransistor that responds to changes in LED light output by altering the current flow within the receiving circuit.

5.3.1 How an LED Works

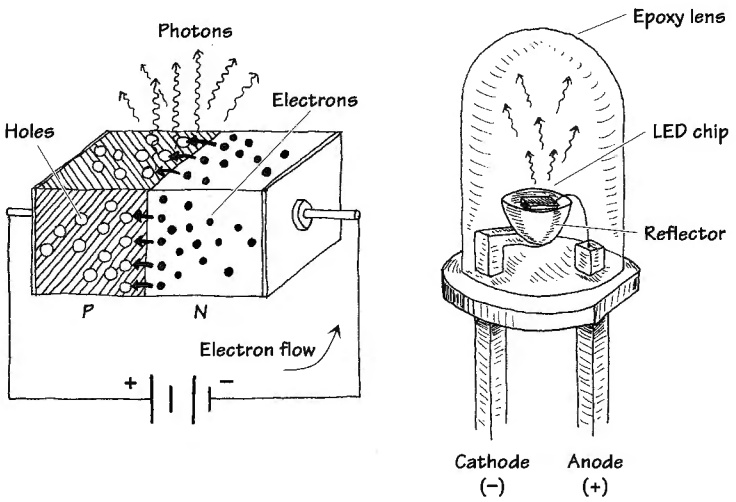


FIGURE 5.6

The light-emitting section of an LED is made by joining *n*-type and *p*-type semiconductors together to form a *pn* junction. When this *pn* junction is forward-biased, electrons in the *n* side are excited across the *pn* junction and into the *p* side, where they combine with holes. As the electrons combine with the holes, photons are emitted. Typically, the *pn*-junction section of an LED is encased in an epoxy shell that is doped with light-scattering particles to diffuse the light and make the LED appear brighter. Often a reflector placed beneath the semiconductor is used to direct light upward.

5.3.3 Technical Stuff about LEDs

Like conventional *pn*-junction diodes, LEDs are current-dependent devices. To control an LED's output intensity, the current that enters the diode (called the *forward current*, or I_F) is varied. The maximum amount of current an LED can handle under continuous drain is relatively small, perhaps as much as 100 mA. However, LEDs can handle a considerable amount of pulsed current, perhaps as large as 10 A.

To protect a diode from excessive current, a resistor is placed in series with the LED. The value of the series resistor R_S depends on the *forward voltage* V_F of the LED, the supply voltage V_+, and the desired forward current I_F. To find the value of R_S, apply Ohm's law, as shown in Fig. 5.8*a*.

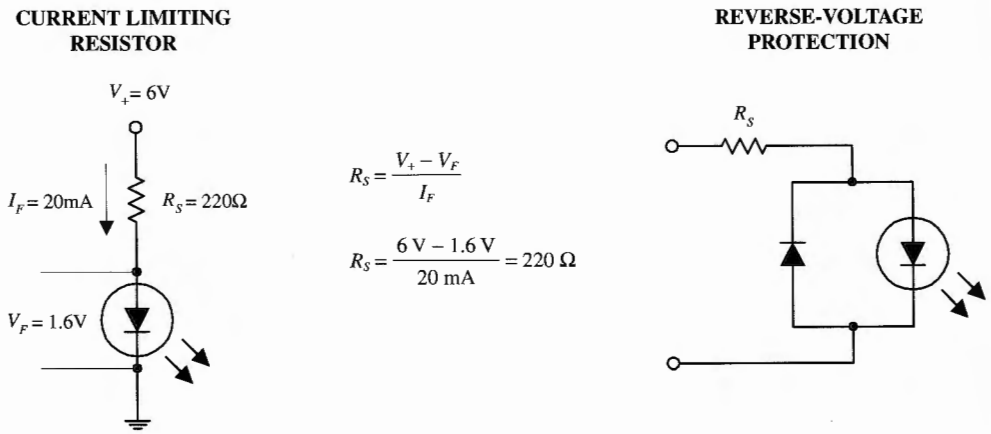


FIGURE 5.8

Like conventional *pn*-junction diodes, LEDs have a reversed breakdown voltage V_R that, if exceeded, may result in a zapped semiconductor. The reversed breakdown voltage for LEDs is relatively small, typically around 5 V. To provide reverse polarity protection to a LED, a diode can be placed in series and in the reverse direction with respect to the LED; the diode will conduct before the reversed voltage across the LED becomes dangerously large (see Fig. 5.8*b*). It is important to note that LEDs provide a small voltage drop when placed in circuits—a result of the *pn* junction. The magnitude of this voltage drop is equal to the forward voltage V_F and may vary from 0.6 V to around 2.2 V depending of the type of semiconductors used. Table 5.2 shows a typical specification listing for a few types of LEDs.

10

Making Movement

By now, you've learned a good bit about how to make the computer listen well to human expression, and you've also learned a few techniques for responding, through light and sound coming out of either the microcontroller or the multimedia computer. The next step is actually animating physical media. This can give you a more unexpected, compelling, even magical response than you get from run-of-the mill-multimedia. For example, even the simplest of animatronic characters in a haunted house (a skeleton arm tapping the visitor on the shoulder, perhaps) can evoke a louder scream from visitors than the most realistic, blood-curdling recorded screams or the largest projection of a ghost. This chapter introduces techniques for creating motion. We'll discuss various types of motors and discuss their characteristics in general, and then we'll talk about some special electrical needs of motors. Following that, we'll discuss the various motor types in detail and describe how to control them from a microcontroller. Finally, we'll discuss some basic mechanical principles, so that you can convert motors' movements into movements that work with your particular system.

Types of Motion, Types of Motors

There are two basic types of motion covered in this chapter: *linear motion*, or motion in a straight line, and *rotary motion,* or motion in a circle. Most of the devices we'll talk about create rotary motion, so the first challenge you'll often face once you get them working will be to convert that rotary motion into linear motion.

The most basic motor you'll use is the *DC motor.* DC motors have two electrical connections. They spin continually when given enough electrical current at the proper voltage. When the current is reversed, they spin in the opposite direction. If the voltage is lowered, they spin slower. If it's raised, they spin faster.

Most DC motors run at fairly high speeds but don't have a lot of pulling force, or *torque.* In order to give them more torque and lower their speed, a series of gears can be added. The next most basic motor, the *gearhead motor,* is just a DC motor with a gearbox on the top. Gearhead motors don't move very fast, but they are much stronger than regular DC motors without a gearhead. The gearbox reduces the speed of the motor, exchanging speed for increased turning strength, or *torque.*

You've already encountered one type of gearhead motor in this book, the *RC servo motor*. A servo motor is a special type of motor that gives feedback about its position (generally using a potentiometer). The RC servo motor is particularly beloved among physical computing enthusiasts because it combines the advantages of a gearhead with a self-contained feedback system that allows you to pulse it in order to move it to a particular position within a 180-degree range. RC servos can't move a full 360 degrees like other motors, but they can be positioned precisely within their range of movement.

The *stepper motor* combines both precise positioning and a full 360-degree range of motion. Stepper motors move in discrete steps around a circle. For example, a 200-step motor moves 360 degrees in 200 steps (1.8 degrees per step). They can be rotated continuously in either direction by continuing to step them forward or backward. They have reasonably good torque, as well. However, they are more complex to connect than other motors.

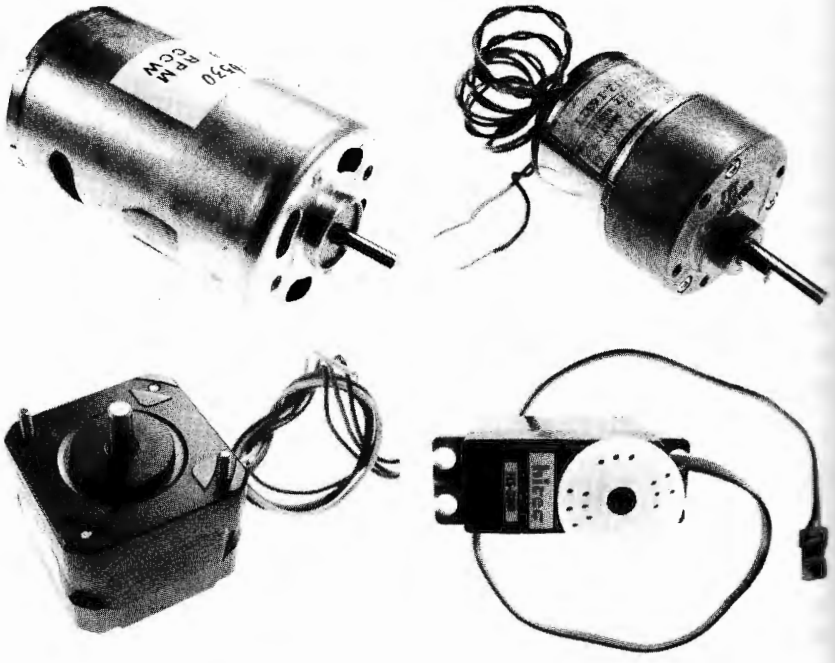


Figure 10.1
Motor types (clockwise from top left): DC motor, gearhead motor, RC servo, stepper motor.

All of the motors in Figure 10.1 create rotary motion and have to be attached to one of the mechanical devices described in the second part of this chapter to create other motions. In order to create linear motion directly, we'll introduce the *solenoid*, a device with a moving rod that pulls in or pushes out when it's given current. You will also come across other actuators that are not motors. In particular, nickel-titanium (*nitinol*) wire has some initial appeal because it contracts like a muscle (it's also known as *muscle wire*) when you apply power. For most applications it is slow to respond, offers minimal movement for the effort needed to get it working, and is finicky about power. You can find products using muscle wire and a related device called *air muscle* at SI Images (http://www/imagesco.com), but we think you will usually get better results using motor and solenoids.

Characteristics of Motors

There are a few characteristics common to all motors that you should keep in mind when looking for a motor for your project.

The *rated voltage* of a motor is the voltage at which it operates at peak efficiency. Most DC motors can be operated somewhat above or below their range, but it's best to plan to operate them at their rated voltage. Dropping below the motor's rated voltage reduces the motor's torque, and operating too high above the rated voltage can burn the motor out. If you're varying the motor's voltage in order to vary its speed, plan on the motor's top speed being at the rated voltage, and its slowest speed at no less than 50 percent of the rated voltage.

Motors draw *current* depending on the load they're pulling. Usually more load means more current. Every motor has a *stall current*, which is the current it draws when it's stopped by an opposing force (like the weight it's pulling). The stall current is much greater than the *running current*, or current that it draws when it has no load. Your power supply for a motor should be able to handle the stall current with extra amperage to spare. For example, if a motor's stall current is two amps, you should use power that can supply at least three amps, to be safe. Likewise, any components that regulate the motor's current should be able to handle three amps. Motors draw almost the stall current for a brief period of time when they start up, to overcome inertia.

Often you'll see a motor rated in ohms. This gives you the *resistance* of the motor. Using Ohm's Law (current = voltage/resistance), you can calculate the motor's running current if you know the rated voltage and the resistance.

Motor *speed* is given in revolutions per minute (RPM) or revolutions per second (RPS). Very occasionally, you'll see motor speeds in Hertz (Hz), which is the same as revolutions per second. Speed is more important when selecting DC and gearhead motors because they're used for continuous motion. For RC servos and stepper motors, speed is usually less important than accurate positioning. However, when quick positioning is a priority, the speed of a stepper may be given as well. For example, hard disk motors are rated by their speed and their positional accuracy because being able to read data off a disk quickly is important when you're reading data for real-time playback, like playing music files or movies.

Position resolution is only a factor for RC servos and stepper motors. It's usually given in degrees or in steps per revolution. Greater resolution means you can create a smoother motion from the steps or achieve greater accuracy when you're positioning whatever it is that the motor is moving.

Torque is the measure of a motor's pulling force. A motor's torque is a measure of how much force it can generate at a given distance from its center of rotation. For example, if a motor can lift a one-pound load that's suspended one foot from the center of its shaft, the motor's torque is one pound-foot. Motor manufacturers haven't standardized this measurement, so sometimes you will see it as lb.-ft., oz.-in., g-cm (gram-centimeter), and any other weight-to-length variation you can think of.

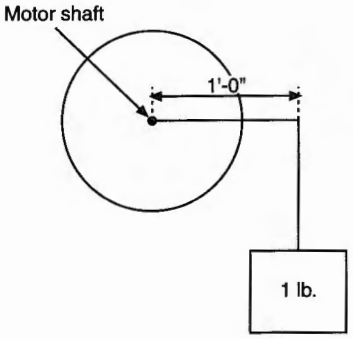


Figure 10.2
Torque of a motor. This motor can deliver 1 pound-foot of torque.

Special Electrical Needs of Motors

Motors work on the electrical principle of *inductance*. When you put electric current through a wire, it generates a magnetic field around the wire. By placing a charged coil of wire in an existing magnetic field (say, between two magnets), the coil will be attracted to one magnet and repelled by the other. Which magnet it's attracted to and which it's repelled by will depend on the direction of the current flow. A higher current will generate a greater magnetic field, and therefore a greater attraction or repulsion.

Inductive Loads and Back Voltage

At the heart of a DC motor is a coil with a current running through it mounted on a spinning shaft. The coil is positioned between two magnets. As the coil is alternately attracted to one magnet and repelled by the other, it spins, and you get circular motion. All *inductive loads* (like motors, electromagnets, and solenoids) work on this same principle: you induce a magnetic field by putting current through a wire, and then use that magnetic field to attract or repulse a magnetic body.

However, the principle works in reverse as well. When you move a coil of wire in an existing magnetic field, the field induces a current in the wire. So if you've got a motor spinning near a magnet, and you turn it off, the magnetic field will induce a current in the wire for a brief amount of time. This current moves in the reverse direction of the current flow you generated to run the motor. It's called *blowback*, or *back voltage*, and it can cause damage to your electronics. You can stop it by putting a diode in parallel with the electronics that you want to protect, facing in the opposite direction of the normal current flow. A diode used in this way is called a *snubber diode*. It should be able to carry the full stall current of the motor. You should protect the rest of your circuit with a snubber diode any time you're using an inductive load. Otherwise your microcontroller will continually reset itself, and you'll probably damage the microcontroller, the transistor that's controlling the motor, and other components in the circuit. The schematic in Figure 10.3 shows a snubber diode protecting a transistor that's controlling a motor. You can also put the snubber diode in parallel with the motor, facing in the opposite direction of the current flow. Whether or not you're using a transistor to control your motor, it's a good idea to use a snubber diode in this way.

Figure 10.3
A snubber diode use
to protect a transist
that's controlling a
motor.

SHOPPING FOR

Most motors requi
find a power suppl
expensive, but you
power supplies fro

There are a few ter
are *regulated*, mea
electronic circuits
as *switching* suppli
supplies, and gener

Computer power su
available inexpensiv
tell you their rating
voltages available ar
currents as well. Mo
+12 volts. To be sure

The ideal power su
allow you to vary th
need, you can then
do a lot of motor an
stock no. 7036 PS (h
using a bench top s

Please be careful wh
particular, be carefu
the power supply is

When possible, it's w

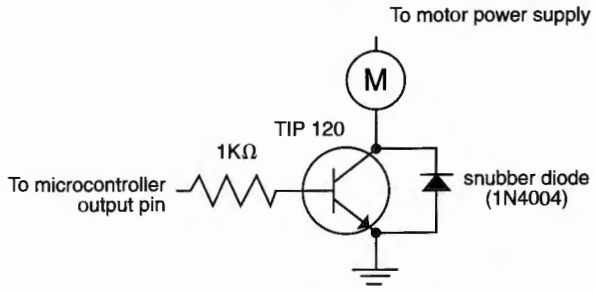


Figure 10.3
A snubber diode used to protect a transistor that's controlling a motor.

Shopping for a Power Supply

Most motors require more electrical power than the ones we've recommended so far, so you need to find a power supply that matches the electrical characteristics of your motor. Power supplies can be expensive, but you can use surplus supplies for many motor applications. You can often repurpose power supplies from other applications.

There are a few terms you should be familiar with before looking for a power supply. Many supplies are *regulated*, meaning that the voltage will not drop as the current increases. This is vital for electronic circuits and useful for motor applications as well. You'll also see power supplies listed as *switching* supplies or *linear* supplies. Switching supplies are generally more efficient than linear supplies, and generally more expensive. If you can afford a switching supply, get it.

Computer power supplies can be particularly useful, as they often have the amperage you need and are available inexpensively or free (if you've got access to a discarded machine). Most computer supplies will tell you their ratings on the side and the amperages they can supply at various voltages. The most common voltages available are 5 volts and 12 volts, though many will also supply –5 volts and –12 volts at low currents as well. Most of the supplies we've seen have black wires for ground, red for +5, and yellow for +12 volts. To be sure, measure the voltage with a meter if you don't have the data sheet on the supply.

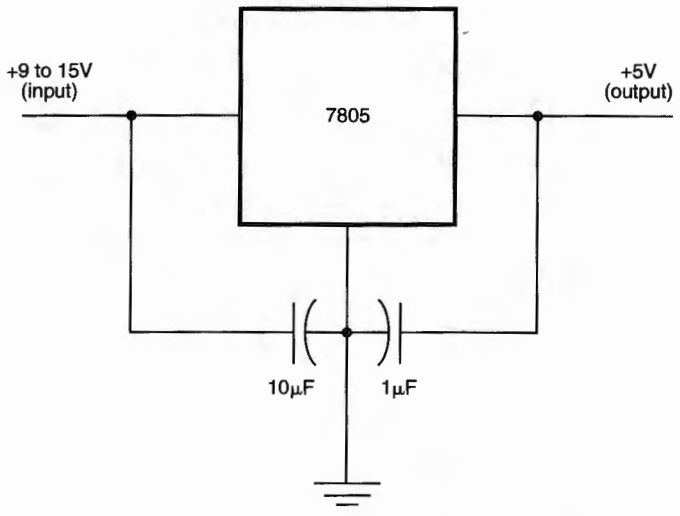
The ideal power supply for electronic and motor applications is the variable bench top supply. These allow you to vary the voltage and current while you test your application. After you see what you need, you can then buy a dedicated power supply. They're not cheap, but they're very handy when you do a lot of motor and electronics work. Marlin P. Jones & Associates carry a nice 0-20V, 0-10A model, stock no. 7036 PS (http://www.mpja.com). Once you've found the voltage and amperage needed by using a bench top supply, you can get a fixed power supply to replace it for your project.

Please be careful when using and handling these power supplies. They can be dangerous. In particular, be careful of the large capacitors in them, which can hold a significant charge even after the power supply is turned off.

When possible, it's wise to separate your motor's power supply from your microcontroller's power supply.

Smoothing Current Drops Using Decoupling Capacitors

Because motors draw so much current when they start up, they affect the current going to the circuit that controls them. Often, when a motor attached to a microcontroller starts up, it will draw so much current that the voltage in the circuit drops and the microcontroller resets itself. We recommend that you put a debug statement at the very start of your program, outside the main loop, to expose this problem when it happens. To prevent the problem, you should use capacitors to *decouple* your microcontroller's voltage regulator, and for the microcontroller itself (see the "Decoupling Capacitors: Stabilizing Your Voltage Regulator" sidebar in Chapter 6 for more on decoupling capacitors). These will smooth out the dips in the current. In fact, it's good practice to put a decoupling capacitor between the power and ground of every IC in your circuit whenever you're using an inductive load in your project (it doesn't hurt to do it whether you've got an inductive load or not). Figure 10.4 shows a properly decoupled voltage regulator. Even when your motor power supply is separated from your microcontroller's supply, they will have a common ground, and the motor will affect the microcontroller. So always decouple your microcontroller when using motors. It's also helpful to use separate voltage regulators for the motor and the microcontroller when possible.



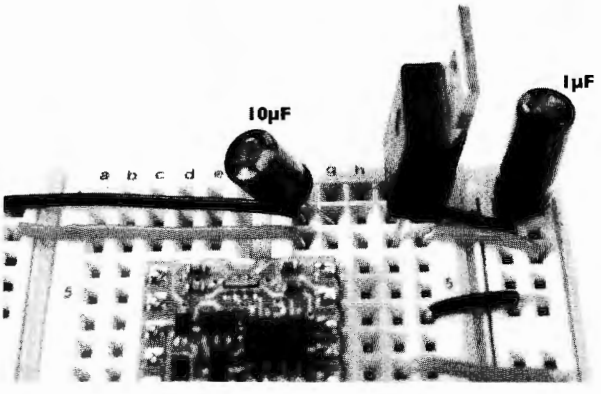


Figure 10.4
Use decoupling capacitors liberally when you're working with inductive loads.

Controlling Motors

Motors require more current than a microcontroller's output pins can supply, so they must be controlled using a combination of transistors or relays, depending on the application. The different types of motors we've mentioned require different control techniques.

The easiest way to control most motors is to get a motor control module that interfaces to your microcontroller. For example, there is a module called the MotorMind B from SolutionsCubed that controls DC motors based on serial commands sent to it from a microcontroller. Solarbotics makes a cheaper module called the L293D Secret Motor Driver that's controllable directly from the microcontroller, using two output pins to set the direction and one to set the speed via pulse width modulation. The Mini-SSC II from Scott Edwards Electronics controls up to eight servomotors with serial commands from a microcontroller. There is also the Little Step-U module from Parallax that can be used for controlling stepper motors in a similar fashion. With these modules you will be finished controlling your motor and you'll be ready to move on to other problems much faster than if you build your own circuitry as described later in this chapter. In some cases they will even provide extra functionality, like ramping up and ramping down the speed of the motor, which will also save you a lot of programming time. In keeping with the high-level approach of this book, we recommend them. The main disadvantage is cost. They are not particularly expensive, but if you are controlling many motors, the cost would add up. Also, if the motor you need to control needs more voltage or current than these modules can provide, you would have to build your own circuitry. We will not cover how to use these modules because they all come with good instructions. Instead, this section of the chapter is about controlling motors at a lower level than these devices. Even if you do use a high-level controller, reading what follows may help you to understand what's going on.

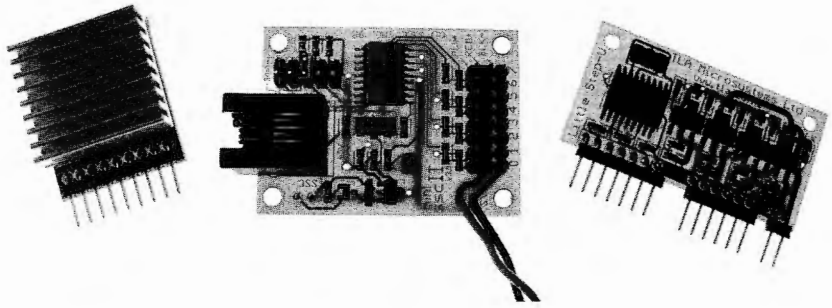


Figure 10.5
A variety of motor controllers: the MotorMind B from SolutionsCubed (a DC motor controller), the Mini-SSC from Scott Edwards Electronics (a servo motor controller), and the Little Step-U from TLA Microsystems (a stepper motor controller).

Controlling DC Motors and Gearhead Motors

DC motors and most gearhead motors are functionally the same in that they have two electrical connections. There are two easily controllable parameters of a DC motor, direction and speed. To control the direction, you reverse the current so that positive becomes negative, and vice versa. This is also called *reversing the polarity* of the motor. To control the speed, you vary the input voltage using pulse width modulation (PWM), as described in Chapter 6.

DC Motor Control Circuit

To control a DC motor from a microcontroller, you use a switching arrangement called an *H-bridge* (see Figure 10.6).

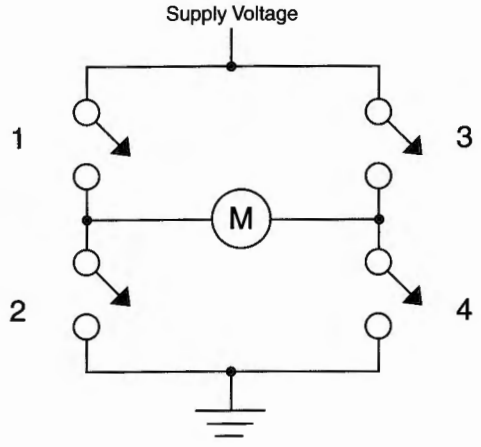


Figure 10.6
Switches arranged in an H-bridge.

When switches 1 and 4 are closed and 2 and 3 are open, voltage flows from the left lead of the motor to the right. When 2 and 3 are closed and 1 and 4 are open, polarity is reversed and voltage flows from the right lead of the motor to the left.

An H-bridge can be built from transistors so that a microcontroller can switch the motor. Although you can make your own H-bridges, it's usually easier to use an H-bridge controller manufactured specifically for the job. A pre-manufactured H-bridge chip will include diodes to protect the transistors from back voltage, sometimes a current sensing pin to sense the current the motor is drawing, and much more. There are many motor drivers available from various electronics suppliers. Look around to find one that suits your needs and price range.

Any H-bridge chip will have certain elements:

- Pins for logic input (control from the microcontroller)
- Pins for supply voltage (supply for the motor)
- Pins for logic voltage (for the transistors inside the H-bridge that read the signals from the microcontroller)
- Pins for supply output (to feed the motor)
- Pins for ground

The logic voltage pins usually take the same voltage and current as your microcontroller. The supply voltage takes whatever voltage and current you need to run your motor. The logic inputs connect to the pins on your microcontroller that you use to output control signals to the H-bridge, and the supply output pins go to your motor. The configuration of these pins might vary slightly depending on the manufacturer of the H-bridge. They might also use slightly different names, but the concepts are the same.

The following example uses an H-bridge integrated circuit, the SN754410, made by Texas Instruments and other manufacturers. These are fairly common chips; Acroname, Digi-key, and other retailers carry them. This particular chip has two H-bridges and can therefore control two motors. It can drive up to one amp of current at between 4.5 and 36 volts. The L293 is another motor driver with the same pin connections. It can be used interchangeably with the SN754410 if you can't find that chip. If your motor needs more voltage or amperage, look around for a different H-bridge. The principles will be similar to this example.

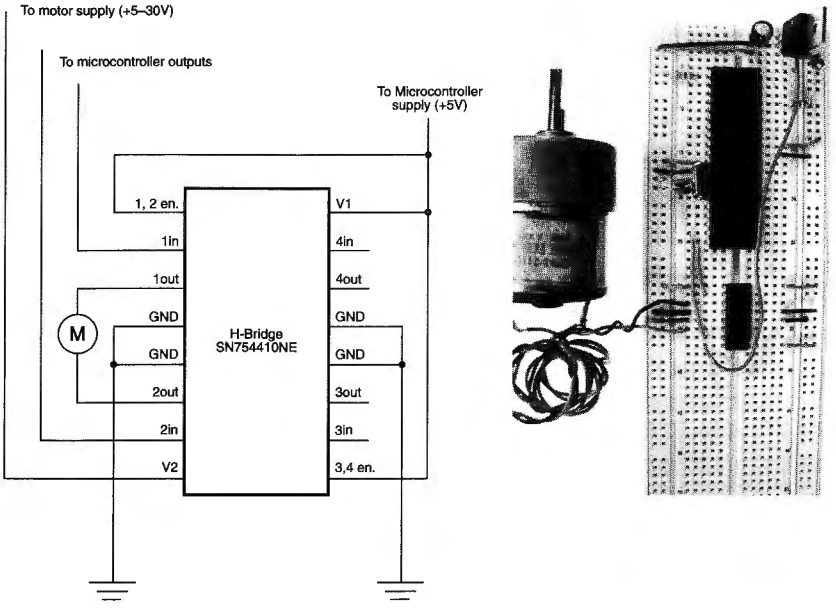


Figure 10.7 A microcontroller connected to a SN754410 dual H-bridge chip.

The pins of this H-bridge that you're using are as follows (taking a pin high means attaching it to 5 Volts):

- *1,2 enable.* Allows inputs 1 and 2 to control outputs 1 and 2, respectively, when this pin is at 5V.
- *3,4 enable.* Does the same as 1,2 enable for outputs 3 and 4.
- *V1.* Voltage supply for logic.
- *V2.* Voltage supply for motors.
- *1in.* Input from controller.
- *1out.* Current flows from V2 to this pin when 1in is high.
- *heat sink/GND.* All four of these pins are ground.
- *2in.* Input from controller.
- *2out.* Current flows from V2 to this pin when 2in is high.

Since you're not using the 3rd and 4th ins/outs, don't bother to take the 3,4 enable pin high.

DC Motor Control Programming

DC motor direction control using an H-bridge is very simple. The following pseudocode illustrates what needs to happen. In this example, if the switch is off, the motor turns one direction, and if the switch is on, the motor turns the other direction. Because this example is so basic, we'll leave it to you to translate into code for your particular microcontroller:

```
loop
    if the switch is off
        make pin 1 high
        make pin 2 low
    else
        make pin 2 high
        make pin 1 low
    end if
end loop
```

To control the speed of a motor using an H-bridge, you use pulse width modulation. There are two ways you can do this. You could pulsewidth modulate (PWM) whichever motor pin should be high instead of just giving it 5 volts. However, there is a simpler approach. You can pulse the enable pin on the H-bridge. This pin is basically an "on-off" switch for the H-bridge, so pulsing it will pulse whichever output pin is active. This has the added advantage of letting you control speed with a different output pin and a different routine than the one that controls direction. To do this, you need to add another connection between the H-bridge and the microcontroller, as shown in Figure 10.8.

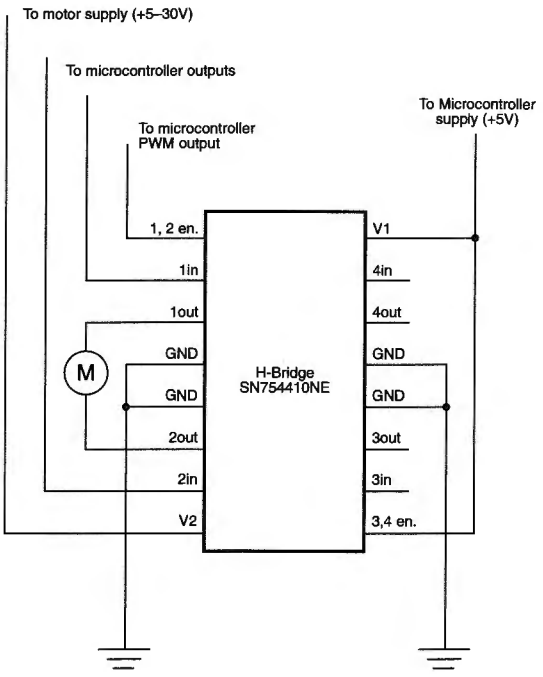


Figure 10.8
Connect the enable pin of the H-bridge to your microcontroller to control speed.

Controlling RC Servos

As we discussed in Chapter 6, RC servo motors are very easy to control. They can be precisely positioned along a 180-degree arc. For instance, you could place a webcam on an RC servo and pan to particular places of interest in your room. You control an RC servo by sending a pulse on the control line every 20 milliseconds. Depending on the pulsewidth (duration of the pulse), the motor moves to a position within the 180-degree range. A 1-millisecond pulse moves the motor to 0 degrees, a 2-millisecond pulse moves it to 180 degrees. Any pulse between 1 and 2 milliseconds moves the motor to a position proportionally between 0 and 180 degrees.

HARDWARE PWM

Keeping a constant PWM signal going while juggling other tasks is tricky because when you're not pulsing the pin, the motor's not moving. On all of the processors except the Basic Stamp, there is another solution to this problem. *Hardware pulse width modulation (HPWM)* is a method for sending a constant PWM signal on a given pin while other tasks are being executed. For more on hardware PWM, consult the manuals and examples for your particular processor and programming environment.

There are several servo motor controllers on the market that interface easily to a microcontroller. If you're planning a project that needs more than one servo, it's wise to invest in a controller so you don't have to spend programming time getting the timing right. The Mini-SSC2 from Scott Edwards Electronics (http://www.seetron.com) will control up to eight servos using serial commands. There are dozens of others on the market that control varying numbers of servos.

If you're controlling only one or two servos, they're very easy to control directly from your microcontroller. The two examples in Chapter 6 give you all the code you need to make that happen.

Controlling Stepper Motors

Stepper motors are different than regular DC motors in that they don't turn continuously but move in a series of very precise steps. They are different from the RC servo in that they can turn continuously in a full circle and they don't give you any feedback about their absolute position. They're useful any time you want to move a precise distance around a circle, or when you want to translate that distance to a precise linear distance. For example, you could attach one to the knobs of an Etch A Sketch toy to draw pictures with more detail than human patience and muscle control allows. Or you might put a sonar ranging device on a stepper motor at the center of a room so you can rotate the sensor around the room, taking distance measurements at regular intervals to learn the contours of the room.

Stepper motors have several coils of wire inside, not just one. The center shaft has a series of magnets mounted on it, and the coils surrounding the shaft are alternately given current or not, creating magnetic fields which repulse or attract the magnets on the shaft, causing the motor to rotate. Figure 10.9 shows the inside of a stepper motor.

Figure 10.9
The inside of a stepper motor. In this motor, many of the coils around the edge are wired together so that there are only four coil circuits.

This design allows for very precise control of the motor: by proper pulsing, it can be turned in accurate steps of set degree increments (for example, 1.8-degree increments, half-degree increments, and so on). They are used in printers, disk drives, and other devices where precise positioning of the motor is necessary. In fact, old printers and scanners are an excellent source for good steppers for physical computing projects. If you see a printer in the garbage, don't pass it up. There are at least two steppers in there that you can use.

Steppers usually move much slower than DC motors, since there is an upper limit to how fast you can step them. However, unlike DC motors, steppers often provide more torque at lower speeds. They can be very useful for moving a precise distance or a specific number of rotations. Furthermore, stepper motors have very high torque when stopped, since the magnetic field of the motor coils holds the motor in place like a brake.

Steppers can slip if they're moving a heavy load, so it's wise to devise a system to continually check their position against an absolute position, if needed. The methods for sensing rotation in Chapter 9 will come in handy here. The simplest method is to use the homemade tachometer described in that chapter. Place a magnet on the disc that the stepper is turning and a Hall-effect sensor or a reed switch on the fixed base on which the motor is mounted. Each time the motor passes the switch, you know one rotation has passed, and you can start counting steps from zero again.

To control a stepper, you need to energize the coils in the right sequence to make the motor move forward. To do this, you need to understand the how the wires are connected. There are two basic types of stepper motors, unipolar steppers and bipolar steppers. There are, of course, a number of stepper motor controller modules available, such as Parallax's Little Step-U module, and if you've got the cash, these will save you time. But if you don't, here's how to do it on your own.

Unipolar Stepper Motors

The *unipolar stepper* motor has five or six wires and four coils (actually two coils divided by center connections on each coil). The center connections of the coils are tied together and used as the power connection. They are called unipolar steppers because power always comes in on this one pole. Figure 10.10 shows the typical wiring for a unipolar stepper.

If you're lucky, your stepper will come with instructions as to which wire is which, and you can wire it up simply. They don't often come with clear wiring instructions (especially

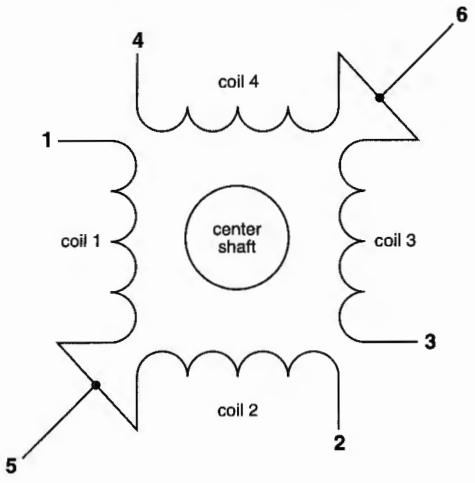


Figure 10.10
Wiring for the coils of a six-wire unipolar stepper (for a five-wire unipolar stepper, wires 5 and 6 are connected).

if you ripped them out of something you found in the garbage), so you might have to determine which wire goes to which coil by yourself. To do this, take an ohmmeter to the wires and measure the resistance from one wire to another. The outer wires for each coil will have a resistance that is double the resistance between the inner wire and either of the two outer wires. For example, if the resistance between wires 1 and 5 is x ohms, then that between 1 and 2 is $2x$ ohms. Remember, two wires that are not connected (for example, 1 and 3, 4, or 6) have infinite resistance, which should read as an error on your meter. When you put voltage across two wires of a coil (for example, 1 to 2, or 3 to 4), you should find that the motor is very difficult to turn (don't force it, that's bad for the motor).

Like other motors, the stepper requires more power than a microcontroller can give it, so you'll need a separate power supply for it. Ideally, you'll know the voltage from the manufacturer, but if not, get a variable DC power supply, apply the minimum voltage (hopefully 5 volts or so), apply voltage across two wires of a coil (for example, 1 to 2 or 3 to 4), and slowly raise the voltage until the motor is difficult to turn. It is possible to damage a motor this way, so don't go too far. Typical voltages for a stepper might be 5 volts, 9 volts, 12 volts, or 24 volts. Higher than 24 volts is less common for small steppers, and, frankly, above that level it"s best not to guess.

To control the stepper, apply voltage to each of the coils in a specific sequence. The sequence would go like those shown in Table 10.1.

Table 10.1
Unipolar Stepper Motor Stepping Sequence

Step	Coil 1	Coil 2	Coil 3	Coil 4
1	high	low	high	low
2	low	high	high	low
3	low	high	low	high
4	high	low	low	high

Note: wires 5 and 6 are wired to the supply voltage.

You drive the stepper by connecting the four coils to a transistor and the two common wires to the supply voltage, as shown in Figure 10.11.

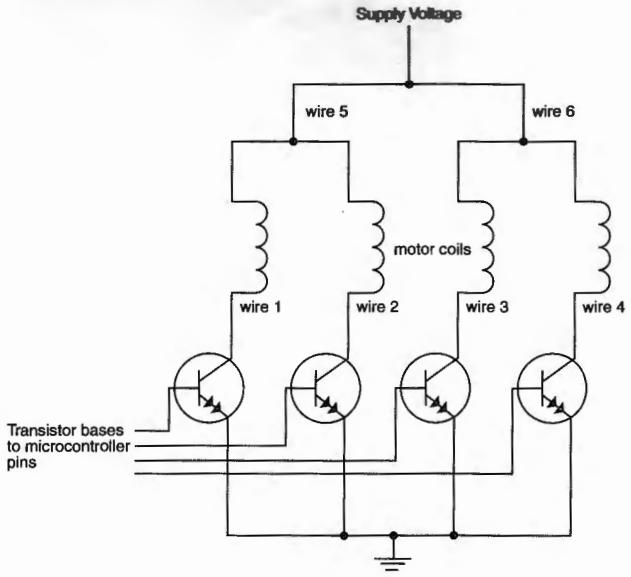


Figure 10.11
A transistor control for a four-wire unipolar stepper motor.

In Figure 10.11, the transistors are TIP120 Darlington transistors. A convenient way to do this is with a Darlington transistor array, such as the ULN2004. Most of the major online electronics retailers will carry Darlington arrays, and they're cheap. Figure 10.12 shows the schematic for wiring a unipolar stepper to a Darlington transistor array.

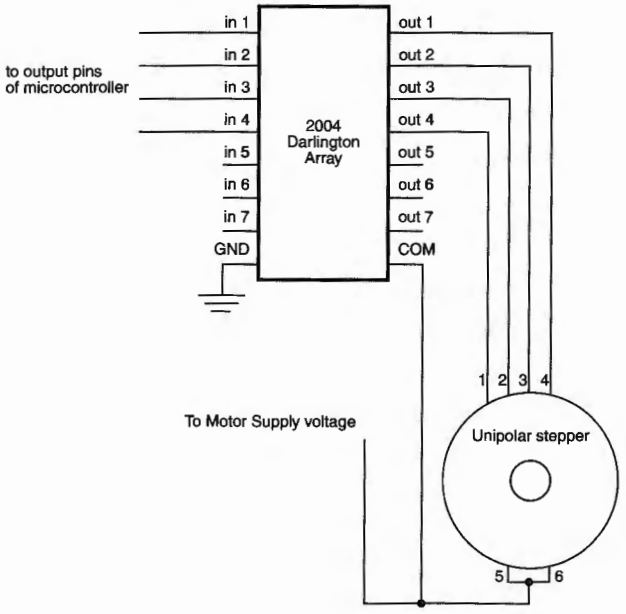


Figure 10.12
A unipolar stepper motor control with a Darlington transistor array. Many steppers use this color scheme for the four coils:

coil 1: orange

coil 2: yellow

coil 3: black

coil 4: brown

Power: red

This may or may not work for your stepper.

If you have trouble getting a unipolar stepper to work with the stepping sequence in Table 10.1, try stepping one coil at a time, like Table 10.2.

Table 10.2
Alternate Unipolar Stepper Motor Stepping Sequence

Step	Coil 1	Coil 2	Coil 3	Coil 4
1	high	low	low	low
2	low	high	low	low
3	low	low	high	low
4	low	low	low	high

Note: wires 5 and 6 are wired to the supply voltage.

Once you have the motor stepping in one direction, stepping in the other direction is simply a matter of doing the steps in reverse order. Knowing the position is a matter of knowing how many degrees per step, and counting the steps and multiplying by that many degrees. So for example, if you have a 1.8-degree stepper and it's turned 200 steps, then it's turned 1.8 × 200 degrees, or 360 degrees, or one full revolution.

Bipolar Stepper Motors

The *bipolar stepper motor* usually has four wires coming out of it. Unlike unipolar steppers, bipolar steppers have no common center connection. In practice, there's not a lot of difference between unipolar steppers and bipolar steppers, for your purposes. However, it's worthwhile to know about both because you never know what kind you'll end up using, particularly if you're salvaging them from trashed appliances. They have two independent sets of coils instead. You can distinguish them from unipolar steppers by measuring the resistance between the wires. You should find two pairs of wires with equal resistance. If you've got the leads of your meter connected to two wires that are not connected (that is, not attached to the same coil), you should see infinite resistance (or no continuity).

NOTE

A six-wire unipolar stepper motor is really a bipolar stepper motor with center connections on each coil. When we connect the two center taps together, we're turning it into a unipolar motor. With four-wire bipolar steppers, it's not possible to do this.

To control a bipolar stepper motor, you give the coils current using the same steps as for a unipolar stepper motor. However, instead of using four coils, you use both poles of the two coils and reverse the polarity of the current. This may seem confusing at first, but let's look at the stepper motor stepping sequence again, and see what it means when we apply it to a bipolar stepper. Figure 10.13 shows the way that current flows in a bipolar stepper motor's coils when we apply current using the steps below.

Table 10.3
Bipolar Stepper Motor Stepping Sequence

STEP	WIRE 1	WIRE 2	WIRE 3	WIRE 4
1	high	low	high	low
2	low	high	high	low
3	low	high	low	high
4	high	low	low	high

Note: wires 2 and 5 are wired to the supply voltage.

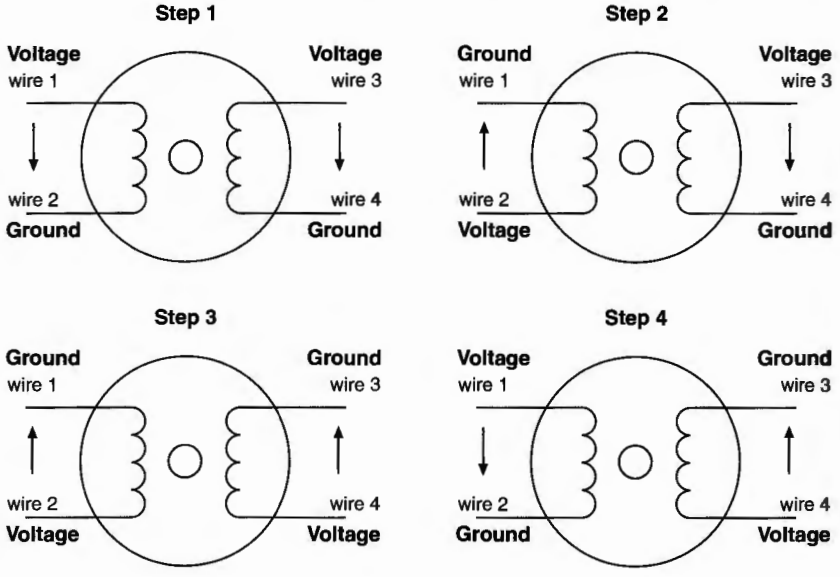


Figure 10.13
A bipolar stepper motor stepping sequence.

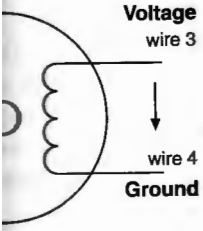
The easiest way to reverse the polarity in the coils is to use a pair of H-bridges. The SN754410 dual H-bridge that you used in the DC motor example above has two H-bridges in the chip, so it will work nicely for this purpose. Figure 10.14 shows the connections you need to make.

Stepper Motor Control Programming

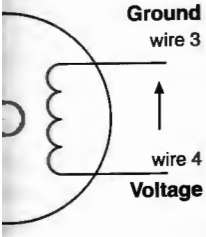
Because both unipolar and bipolar stepper motors are controlled by the same stepping sequence, we can use the same microcontroller code to control either one. In the code examples below, connect either the darlington transistor array (for unipolar steppers) or the dual H-bridge (for bipolar steppers) to the pins of your microcontroller as described in each example. There is a switch attached to the microcontroller as well. When the switch is high, the motor turns one direction. When it's low, it turns the other direction.

WIRE 4
low
low
high
high

p 2

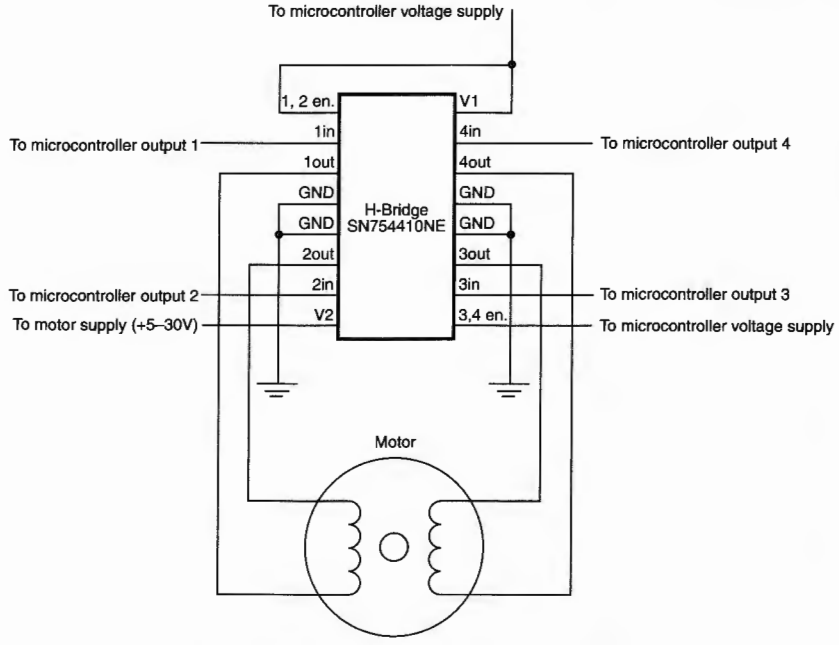


p 4



ges. The
two H-bridges
connections you

ne stepping
In the code
r steppers) or
as described in
hen the switch is
tion.



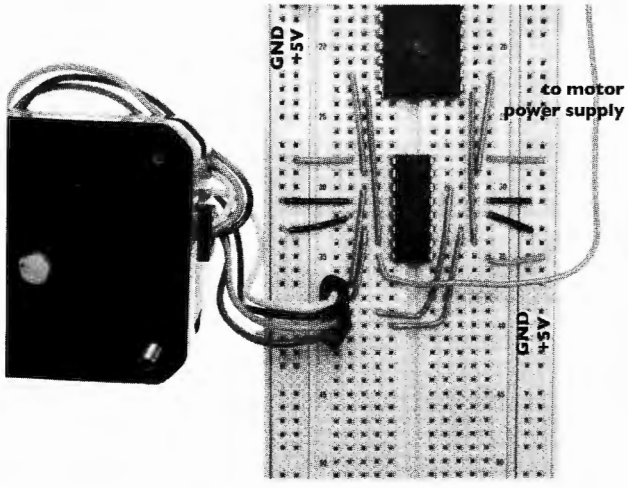


Figure 10.14
A bipolar stepper motor controlled by a SN754410 dual H-bridge.

Here's the pseudocode for this example:

```
If switch is high then
    Turn motor one direction:
        Set coils to step 1 of sequence
        Set coils to step 2 of sequence
        Set coils to step 3 of sequence
        Set coils to step 4 of sequence
Else
    Turn motor the other direction:
```

```
        Set coils to step 4 of sequence
        Set coils to step 3 of sequence
        Set coils to step 2 of sequence
        Set coils to step 1 of sequence
End if
```

PBASIC
MBasic
PicBasic Pro

```
' switch is on pin 8
' stepper motor connections are on pins 0 to 3, as follows:
' connection 1: pin 0
' connection 2: pin 1
' connection 3: pin 2
' connection 4: pin 3
' we will set all of pins 0 to 7 at once, rather than setting each pin individually.

' set variables:
steps       var  word         ' keeps track of which step we're on
stepArray   var  byte(4)  ' an array to hold the stepping sequence


' set the pins 0 to 3 to output:
DIRL = %00001111

' set all the pins 0 to 7 high:
OUTL= %11111111
' set in 8 (the switch) to input:
input 8
' set the values in the step array:
stepArray[0] = %00001010
stepArray[1] = %00000110
stepArray[2] = %00000101
stepArray[3] = %00001001

' pause half a second at startup:
Pause 500

main:
    ' if the switch is high, add one to steps; otherwise, subtract one:
    if IN8 = 1 then
        steps = steps + 1
    else
        steps = steps - 1
    endif
    ' the modulo operator (//) gives the remainder of the division of the
    ' two quantities in the operation. So x // 4 can never be more than 3,
    ' since anything divided by 4 has a maximum remainder of 3.
    ' below, we use modulo to convert the number in steps to a number
```

```
        if getPin(13) = 1 then
            thisStep = thisStep + 1
        else
            thisStep = thisStep - 1
        end if

        call stepMotor(thisStep)
    loop
End Sub


sub stepMotor(byref whatStep as integer)
    ' the modulo operator (//) gives the remainder of the division of the
    ' two quantities in the operation. So x // 4 can never be more than 3,
    ' since anything divided by 4 has a maximum remainder of 3.
    ' below, we use modulo to convert the number in steps to a number
    ' between 0 and 3. Then we set all the pins form 5 to 12 by setting
    ' register.portc equal to one of the 4 elements of the step array:

    ' sets the value of the eight pins of register.portc to whatStep:
    register.portc = motorStep(whatStep mod 4)

    call delay (0.01) ' vary this delay as needed to make your stepper step.
end sub
```

If you're having trouble getting your stepping sequence to work correctly, try setting the pause or delay between steps to a full second. Then watch the motor step by step as it moves in its sequence. It should move consistently in the same direction each time. If it doesn't, perhaps taking three steps forward and one back, you probably have the wires connected incorrectly. Try changing the arrangement of the connections between the H-bridge or transistor array and the motor until you get it right.

Controlling Solenoids

All the motors we've discussed so far create rotary motion, and therefore need some sort of mechanical system to convert their motion to linear motion. In contrast to the motors we've discussed, *solenoids* create linear motion. A solenoid is basically a coil of wire with an iron shaft in the center (see Figure 10.15). When the coil is given current, it creates a magnetic field, and the shaft is pulled or pushed as a result. When the current is turned off, the magnetic field disappears and the shaft moves back, either because gravity pulls it back down or because it's attached to a spring mechanism that pushes or pulls it back into place. Solenoids operate much like electromagnetic relays, which we described in Chapter 6. Solenoids cannot be positioned variably. They have only two positions, on or off. The shaft can be at full extension or compression, or not. There is no way to position the shaft halfway. In that sense, they are digital output devices.

Figure 10.15
A solenoid.

Since they are made up of charged coils that induce a field, solenoids share many of the same characteristics as motors. They have a characteristic operating voltage and amperage, the coil has a fixed resistance, and they can exert a certain fixed amount of force. They have a few additional characteristics that you need to know in order to use them as well.

Solenoids can be push-type or pull-type. In a *push-type solenoid*, the shaft is pushed out of the barrel when the coil is given current. In a *pull-type solenoid*, the rod is pulled into the barrel when current is applied. Both types require some mechanism for returning the shaft to its rest position. If the system to which the solenoid is attached doesn't pull or push it back into place, this is usually done using a spring.

A solenoid's *draw* refers to how far the shaft travels. Most solenoids do not have a very long draw. An inch is a long draw. Solenoids are useful for applications where you need a short linear movement, like automatic door locks, valve closing and opening, and so forth. In cases where a long linear motion is needed, motors are the better tools.

Solenoids have a given *duty cycle*, which is a measurement of the on time divided by the on time plus the off time. For example, if a solenoid is on for 1 second and off for 3 seconds, its duty cycle is 1/1+3, or 1/4, or 25 percent

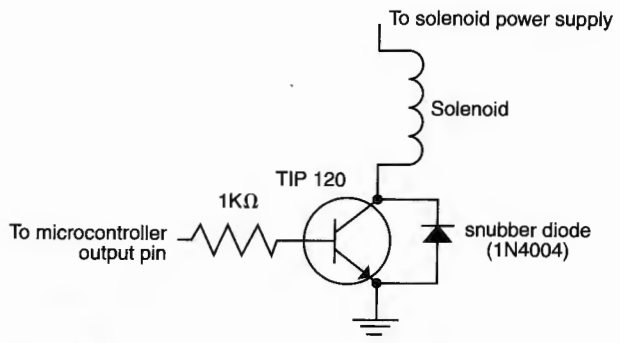
Solenoids also have a given *maximum on time*, after which they need to be turned off for the rest of the duty cycle. A solenoid operated consistently for more than its maximum on time, or for a greater duty cycle than it's rated for, will usually overheat and stop working. For example, if a solenoid has a 20 percent duty cycle and a maximum on time of a tenth of a second, then it can't be turned on for more than 0.1 seconds every half second. Solenoids with short maximum on times or duty cycles are usually referred to as *momentary solenoids*, and those that can stay on continually are called *continuous solenoids*.

Though there are some reversible solenoids, they are rare (reversible solenoids use a magnetic shaft). Most solenoids cannot be reversed, so changing the polarity on a push-type solenoid will not make it a pull-type solenoid. Make sure to get the one you need.

Most electronic and mechanical retailers and surplus suppliers carry a variety of solenoids, as they're very common in electronic appliances. All Electronics is our favorite source for solenoids, but many of the other retailers listed in this book will carry them, too.

Controlling a solenoid is very easy. All you need is a relay or transistor that can switch the voltage and current that the solenoid requires, and a diode to block the back voltage

generated by the solenoid (remember, it is an inductive load). You can use the circuit shown in Figure 10.16.



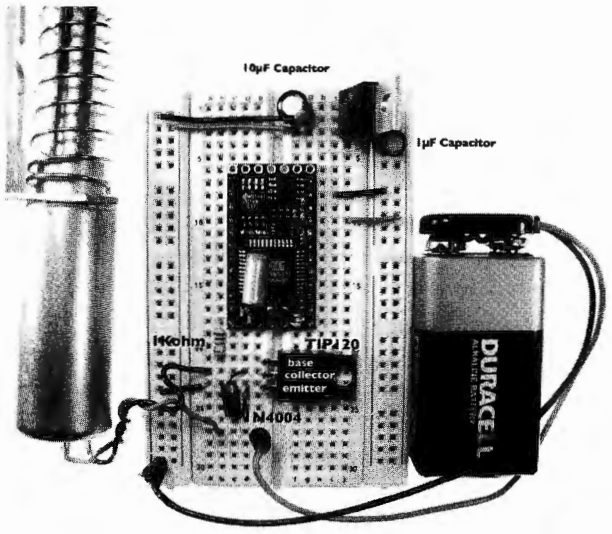


Figure 10.16
A solenoid control circuit.

Programming for a solenoid is the same as programming for a digital output. All you need to do is to take the output pin that the solenoid's connected to high to turn it on, and take it low to turn it off. Make sure that your program can't keep the solenoid on for longer than the maximum on time or duty cycle, however.

Basic Mechanics: Converting Motor Motion to Usable Motion

With any motorized system, the first problem you face after you get the motor working is how to convert the motor's motion into something you can use. There are a number of methods common to mechanical systems of all sorts that are useful to know about when you want to control motion of any sort. While this section is by no means comprehensive, it will help get you started on understanding a few basics of how to move things, as well as introduce some of the terminology you'll need when you start looking for parts.

The most important thing to keep in mind when planning any mechanical system is this: all machines are used to do work of some kind. This may seem obvious, but work, in this case, is defined as a force applied over a certain distance (work = force × distance). Machines are designed to allow you to vary the ratio of force to distance in order to get more work for less force. They are mechanical transducers: they convert one form of mechanical energy into another. Some systems are more efficient than others, but no machine converts 100 percent of the input force into output force. They all lose some energy along the way, to friction, heat, and so forth. Deciding what system will do the work best for the least amount of effort is your prime task in dealing with mechanical systems.

When you build or adapt a mechanical system to do your bidding, you always have to consider a few factors:

- *Can it be done with a servo motor?* You might consider working backward from the easiest motor to use with a microcontroller, the RC servo motor. Even if you have to add a small mechanical linkage to make your project work with one of these, it will probably save you work compared to powering, wiring, and controlling anything else.
- *Does it do the work you need it to do?* If the mechanical system you're designing won't convert the force that your motor can deliver into the force that you need, it's no good.
- *Will it work with the motor you have to work with?* Is there a simple way to connect the system to your motor? If not, don't waste tons of time trying to adapt one to the other. Figure out which one is easier to replace, and replace it.
- *Will it fit the space you've got to work with?* If you've got to fit everything in a one-foot cube, and you've got a gear system that takes twice that space, it's no good. Find a better way. Alternately, ask yourself if you can hide the mechanism somewhere behind the one-foot cube.
- *Is it more cost- and time-effective to modify an existing mechanism to do what you want rather than to make your own from scratch?* Like electronics, mechanics can be very seductive. Making your own gear train or system of linkages can be very satisfying. But if doing so takes all the time you've got for your project, such that you never get to consider the other issues brought up in this book, then you shouldn't build your own. Many toys, appliances, and other everyday devices have mechanisms built into them that can be cannibalized for physical computing systems. Take a look inside that old VCR, CD player, Baby Poops-A-Lot, or cymbal-playing monkey that you broke years ago. You're likely to find some useful mechanisms.

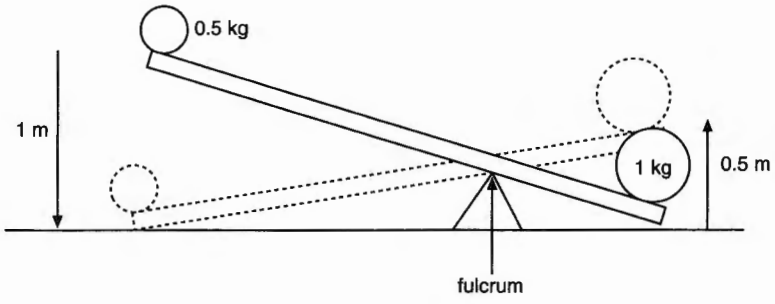
Simple Machines

There are a handful of simple machines that occur in almost every mechanical system. Combinations of these machines are the building blocks of devices that create motion. Understanding how these simple machines convert mechanical energy will make it easier for you to design ways of creating motion in your work.

The Lever

The *lever* is about the simplest of machines, one you're used to dealing with every day, from can openers to car jacks and millions of other everyday devices. A lever is used to change the direction and distance of a force. Every lever has a *fulcrum point* about which the lever pivots. The two arms of a lever are not equal. A lever lets you move a heavy weight with a small force, by taking advantage of this inequality. Imagine that you have a lever whose long arm is twice the length of the short arm. If you place the load you need to move on the shorter side of the lever, and apply half the force needed to move that load to the longer side of the lever, you move the load. You have to apply the force for twice the distance in order to do the same amount of work, however. You can see this in action in Figure 10.17. A one-kilogram weight is moved half a meter by putting a half-kilogram weight on the long arm of the lever and letting it push down one meter.

Figure 10.17
The lever.



The ratio of the long arm to the short arm gives you the *mechanical advantage* of the lever, which is the ratio of work put into the system to work coming out of the system. If the long arm is twice as long as the short arm, the ratio is 2:1, and the mechanical advantage is 2. So you could move the weight with half the force needed if you're willing to apply your force twice as far. This comes in handy when you've got a motor that can apply only a limited amount of torque, but you've got a lot of space to work with.

The Pulley

A *pulley* is a series of moving wheels and ropes, chains, or wires used to reverse the direction of an applied force or to gain mechanical advantage. A single pulley just allows you to change the direction of the force, which is very useful when you want to add a counterweight to a moving load, but multiple pulleys in a system add mechanical advantage. However, if the single pulley is moving, it doubles the force because it makes you pull twice as far. Look at the pulleys in Figure 10.18. In the system on the left, the pulley is fixed and there is no mechanical advantage. Only the direction of the force is changed. In the system on the right, the mechanical advantage is 2, because you have to pull the rope up two meters to move the weight one meter. What you lose in distance, you gain in force, because it takes only half the force that gravity applies to the weight to pull it up.

The more moving pulleys you add, the greater the mechanical advantage you get. At the same time, you increase the distance you have to pull in order to move the weight. Adding fixed pulleys, on the other hand, only allows you to change the direction of the force. Figure 10.19 illustrates this with a few pulley arrangements and their mechanical advantages.

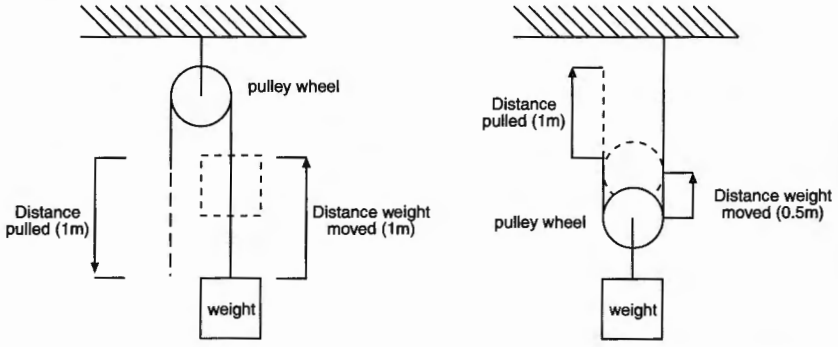


Figure 10.18
Single-pulley systems: fixed pulley (left) and moving pulley (right).

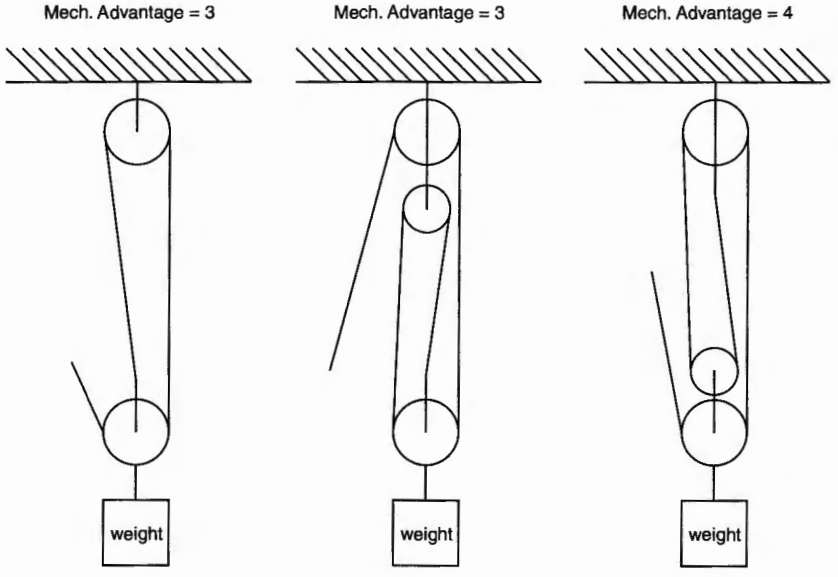


Figure 10.19
Multiple-pulley systems.

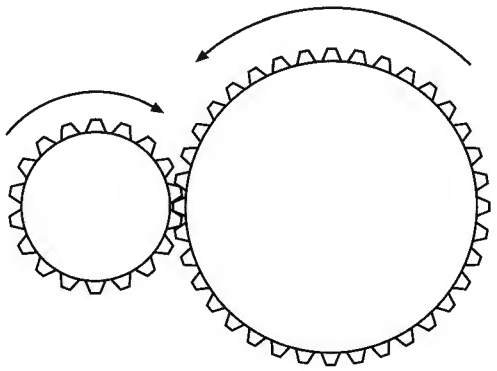
The Gear

Gears are used to convert rotational motion, both by changing direction and by trading speed for torque. The mechanical advantage of a gear train is called the *gear ratio*, which is determined by measuring the ratio of the radii of the gears. In the gear train in Figure 10.20, the smaller gear is half the size of the larger, so the gear ratio is 2:1. The larger gear will move half the speed of the smaller but will provide twice the torque.

The gears in Figure 10.21 move in opposite directions. So if the smaller gear were attached to a motor, the larger gear would move at half the speed of the motor and provide twice the torque in the opposite direction of rotation.

Certain gears can be used to change the axis of motion as well. *Bevel gears* have their teeth mounted at an angle to the axis of the gear so that the mating gear does not have to be mounted at the same angle. *Helical gears* have their teeth cut at an angle to the face of the gear. This increases the power of the gear train because more of the face of each gear tooth is exposed to the other gear.

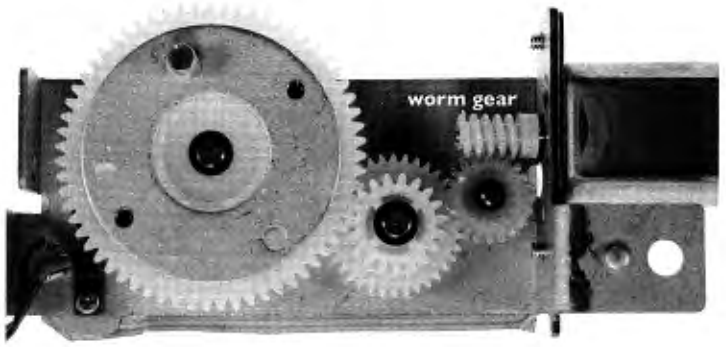
Figure 10.20
A simple gear train.



Screws can be used not only as fasteners, but also as gears. Screws convert rotational motion into linear motion. A screw mounted to a motor with a moving head fastened to it is at the core of most handheld CD players, for example, to move the laser beam from the edge of the disk to the outside. Screws are also used in vises, clamps, and many other common tools.

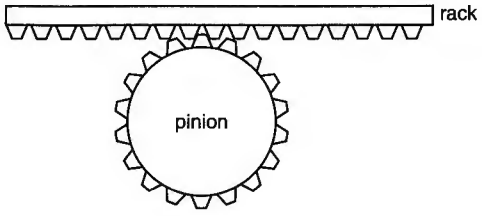
Worm gear mechanisms combine a regular, or helical, gear and a screw. They generally have very high gear ratios and shift the axis of motion by 90 degrees. Figure 10.21 shows a typical worm gear.

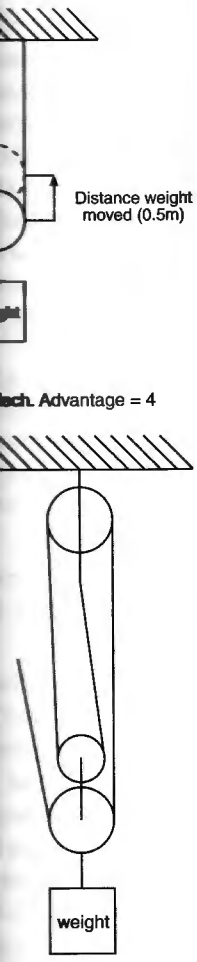
Figure 10.21
A worm gear.



Rack-and-pinion gears are also used to convert rotary motion to linear motion. In a rack-and-pinion, teeth are mounted along a linear track (rack) that moves by being run against a normal gear (pinion). Rack-and-pinion systems are common steering mechanisms in cars; the rack is attached to the axle mount, and the pinion is attached to the steering wheel. Figure 10.22 shows a rack-and-pinion.

Figure 10.22
A rack-and-pinion.





Although there are many sources for individual gears and gear trains, it's often easiest to look for gear trains in existing toy and hobby sets. Kits like K'NEX, LEGO, Erector, and others have very useful gears, gear trains, axles, shaft collars, and other parts for making customized motor systems. Even if you're not interested in the look of these parts, consider them for the infrastructure of your project, if they can be clothed in materials that match the look you want. Using existing parts like this will save you hours of construction time.

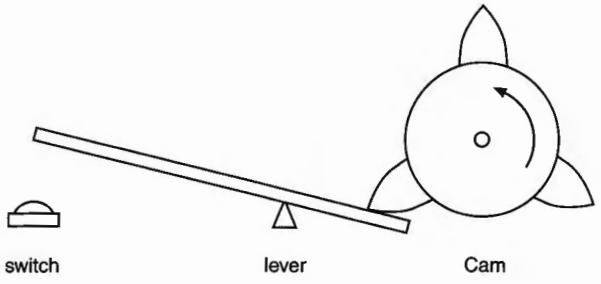
The Cam

The *cam* is a wheel mounted eccentrically (off-center) on a shaft. It seems very simple, but serves a number of purposes. The simplest use of a cam is as a vibrator. By spinning the motor, the weight of the cam causes the motor's axis to shift. If the motor is attached to some solid surface, say, a pager or cell phone body, the surface vibrates. In fact, pager motors are very common (available from most of the retailers mentioned), very small, and very useful for creating vibration in all kinds of projects. You can also create your own vibrating motors just by gluing a nut eccentrically on a small DC motor.

Cams can also be used to create oscillating or periodic motion from rotary motion. By placing a shaft against the edge of a cam, the shaft will move up and down as the cam rotates eccentrically on the motor. Cams can also produce periodic motion.

For example, the cam in Figure 10.23 would produce motion with three sudden jumps per revolution, moving the lever nearby to hit the switch on the other side of the lever.

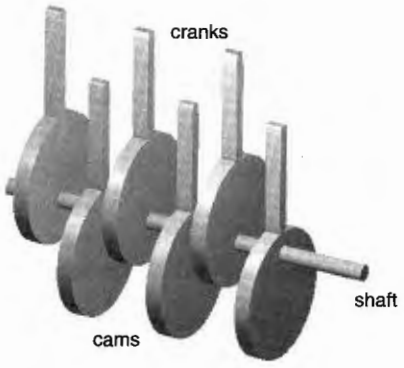
Figure 10.23
A cam used to create periodic action.



Another useful form of cam is the *camshaft*, in which a rotating shaft has bends in it to produce several cams all on the same shaft. Camshafts are used in car engines to move the valves in opposition to each other. A typical camshaft would look like the one in Figure 10.24.

Camshafts are also used in mechanical automata, where one crank may turn several dancing figurines at a time. Any time you want to move several objects from one motor in a periodic motion, a camshaft will do the job.

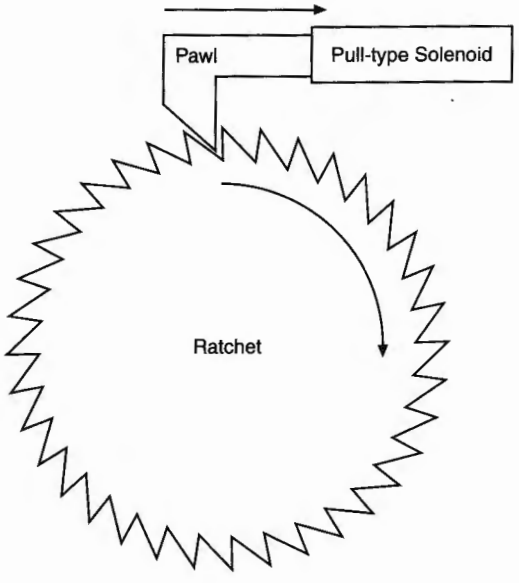
Figure 10.24
A camshaft.



The Ratchet

The *ratchet* is related to the gear in that it's a wheel with teeth, but it's used for a different purpose. Ratchets allow the wheel to move in one direction, but not the other. The teeth of a ratchet are cut with one vertical edge and one diagonal edge. An external lever hook called a *pawl* can slide up the diagonal side of each tooth, or catch against the vertical side of the tooth. In Figure 10.25, a pull-type solenoid pulls in when given power, moving the ratchet clockwise a few degrees. Ratchets can be useful in systems where you can apply force in one direction, but not the other, and need to continually apply force for longer than the distance of your motor. Ratchets are often used in combination with levers to create winches.

Figure 10.25
A ratchet.

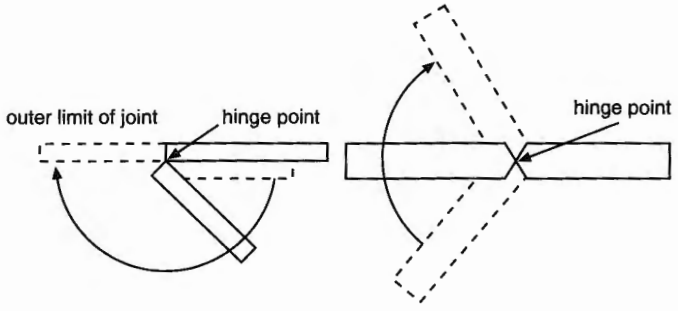


Joints

There are endless ways to join structural elements together to create movement, and each unique system of connections will give you a different type of motion. There are three types of joints that are most common, however: rotating joints, sliding joints, and bending joints. Each type of joint will have its own advantages and limitations, its own weak points, and points of greatest friction.

Bending joints, like a *hinge*, will give you limited motion in one axis. A hinge will be able to move at most 180 degrees, most likely less, depending on where your two structural pieces are hinged together. Figure 10.26 shows the effect of hinge placement on the range of motion between the two joined pieces.

Figure 10.26
Hinges.



Rotating joints are joints where circular structural elements rotate around a common axis. In a rotating joint, there is friction between the rotating shaft and the collar, or *bushing*, in which it's rotating. Figure 10.26 shows a typical shaft and bushing. This can be managed by using some form of oil or grease lubricant, a high-density plastic bushing that has very little friction, an oil-impregnated bushing, or by using a bearing of some sort. The job of a bushing is to keep the rotation of a joint in one axis, so you want to make sure the shaft and the bushing fit together snugly, with just enough room to turn. Too tight, and it won't turn. Too loose, and the shaft will shift off the ideal axis of rotation and bind up. *Bearings* consist of a metal chassis containing balls or cylinders that are free to roll within the chassis so that they provide friction relief by turning. Bearings are often used to lessen the friction in a rotating joint.

When dealing with any rotational motion, it's important to know where the system will bear the load, or weight. An *axial load* is a load parallel to the shaft, and a *radial load* is a load perpendicular to the shaft. Both stress the joint in different ways. An axial load will tend to grind the shaft against the end of the bushing, and a radial load will pull or push the rotation off-axis and place extra stress on one side. Every load has both axial and radial components. A well-balanced load is one that limits both as much as possible. Almost any application where you're converting the rotational motion to linear motion, for example using the motor as a winch, will place a radial load on the shaft. Wheels place a radial load on the shaft of an axle. Axial loads will place pressure on the socket where the shaft sits in the bushing and will cause the joint to wobble unless they're perfectly balanced. Turntables place an axial load on the shaft that turns them and a radial load when they're off-balance.

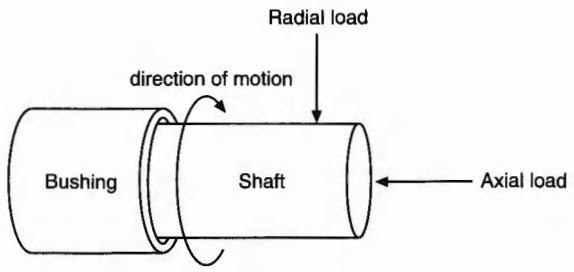


Figure 10.27
A rotating joint (shaft and bushing).

Rotating joints are perhaps the most common joints due to the flexibility they offer; for example, many bending joints, such as hinges, are made of a rotating joint with two radial loads. In addition, to get more than one axis of motion, a spherical rotating joint called a *ball-and-socket joint* (like your hip joint) is necessary. In a ball-and-socket joint, a rod with a spherical end (ball) is fitted into a cup (socket) at the end of a second rod. Lubricant is placed between the ball and socket to reduce friction.

Sliding joints are joints where one surface slides against another. Like rotating joints, you have to take care to reduce friction but also ensure a snug fit, as well as avoid binding because of a misfit. Ideally, the sliding section of a sliding joint should be at least twice as long as the width of the slot. This will minimize the chance of the piece binding up. Sliding joints can generally be lubricated with the same materials as rotating joints. Figure 10.28 shows a typical sliding joint.

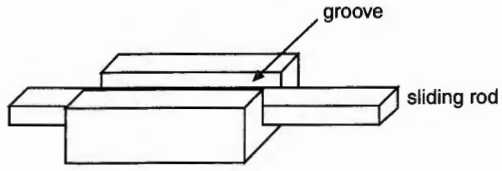


Figure 10.28
A sliding joint.

Linkages

Stiff and flexible support members can be linked using various joints to create a variety of movement and support systems. Linkages can also convert one type of motion to another, similar to cams.

The *four-bar linkage* is a common tool for creating complex motion from simple rotational motion. It consists of four stiff bars connected by joints about which they can rotate. One bar is kept stationary, and is called the *ground*. The bar being moved is the *coupler*, and it's opposite the ground. The two joining bars can be of varying lengths, depending on the application. If a joining bar can rotate a full 360 degrees, it's called a *crank*, and if it's constrained by the rest of the system from rotating all the way around, it's called a *rocker*. Systems with joining bars of unequal length change the speed of motion similar to gears, but they also change the path of motion. The system in Figure 10.29 creates a very complex pattern of movement from the end of the coupler as the crank turns all the way around, not unlike the shape of a windshield wiper blade (the pattern of movement is shown on the right). Furthermore, if you turn the whole mechanism on its side clockwise, use the crank as a ground instead, and attach the end of the former ground to the edge of a wheel, you get a back-and-forth motion that's not too far off from the movement of a walking leg.

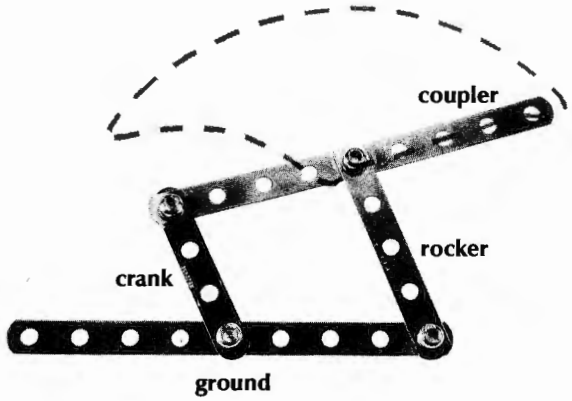


Figure 10.29
A four-bar linkage, showing the movement pattern of the tip of the coupler.

SHAFT COUPLERS

No matter what you need to attach your motor to, you'll need some extra hardware in order to attach it. The part you're looking for is usually called a *shaft coupler*, or a *collar*. It's basically a metal sleeve that fits over the shaft of the motor. It attaches securely to the motor shaft and the part you're attaching to the motor. You can find couplers with different-sized holes on either end to be able to join shafts of differing diameters. The most basic shaft couplers rely on friction to do the job. These can slip, however, so some shafts and couplers are designed with a notch taken out of them, so that the coupler can fit the shaft in only one way. (See the gearhead motor in Figure 10.1 for an example of a notched shaft.) *Set screws* are often used with couplers to assist in making a solid connection as well. A set screw is a screw mounted perpendicular to the shaft that screws through the coupler to bind against the shaft, securing the coupler. There are many varieties of couplers, but most will be a variation on what's described here. When you're looking for a motor to match a gear, pulley, or wheel, or vice versa, make sure you know the necessary shaft diameter and what kind of coupling is possible. Figure 10.30 shows a variety of shaft coupling methods.

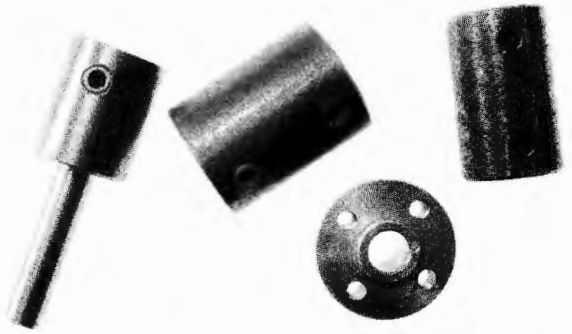


Figure 10.30
Collars and couplers.

Scissor linkages combine several stiff supports in an X fashion, with rotating joints at the corners. A scissor can retract and extend a great distance with reasonable support. Figure 10.31 shows a typical scissor linkage.

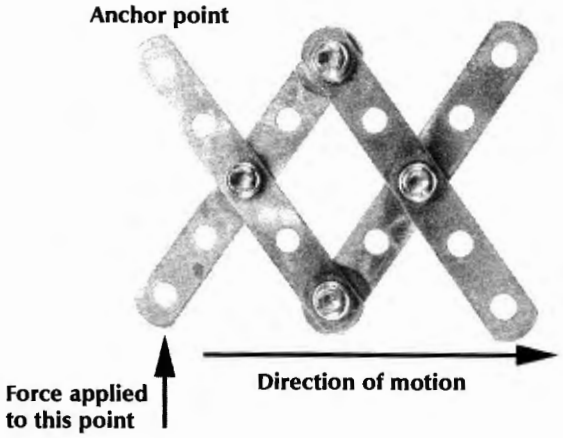


Figure 10.31
A scissor linkage.

The Piston (Rod and Crank)

A *piston* is a combination of a wheel and a series of linkages. It's a simple way to convert rotary motion to back-and-forth linear motion. A crank is attached to the side of the wheel by a rotating link, with its other end attached to a rod or block (piston head). The rod or block is constrained to move in a line, so when the wheel turns, the rod or block moves forward and backward (see Figure 10.32). Pistons are used in most engines, but can also be adapted for applications like animatronics, hydraulics, pneumatics, and more.

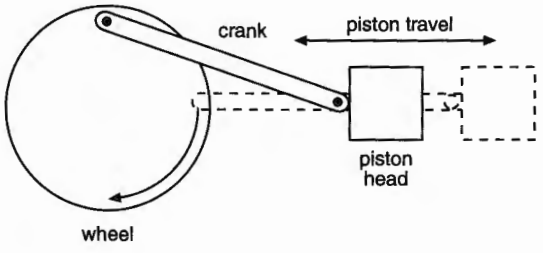


Figure 10.32
A piston.

Construction

Whenever you create motion, you need a solid structure on which to mount your moving parts. We haven't talked about construction materials in this book because we feel it's a topic that merits its own book. However, there are a number of materials that we use for quick and easy construction of prototypes that are worth mentioning here.

If we had to single out one supplier for construction and mechanical parts, it would be McMaster-Carr. They carry a very wide variety of hardware and construction materials, and their catalog and Website is very detailed and helpful. If you can't find what you're looking for locally, check out http://www.mcmaster.com.

Foamcore

Foamcore is a wonderful construction material for quick projects, because it's lightweight and very easy to cut and form. Though it tends to buckle when you put too much weight on it or apply too much force, it's very easy to strengthen with ribs and struts made from foamcore placed on end, or from wood. It holds together well with white glue or wood glue, and is very fast to build with. Many fabricators, designers and architects use it for initial models because it's so easy. Your local art store will carry it in varying colors and thicknesses.

Tupperware

Tupperware and other household plastic containers make useful housings for electronic parts. They're relatively easy to make holes in and can be sealed up with hot glue. While it's certainly not a professional-grade housing, we see it used in physical computing projects all the time.

Wood

If you've got the saws necessary to work with wood, it's great for large physical structures, of course. However, carpentry is its own discipline, and can take as much or more time as the electrical and programming work laid out in this book. It's electrically non-conductive, which is handy when working with circuits, but sawdust can carry static electricity, so make sure you dust off any wood structures before you introduce your electronics.

Plexiglas

Plexiglas (acrylic) makes good housings, and can also make good, lightweight structures. You can cut thin Plexiglas sheets by simply scoring it with a matte knife along a straight edge and snapping it apart. Many industrial plastics retailers will have laser cutters for cutting custom forms out of Plexiglas and will cut it for you for a fee. Some of them will simply have you bring in a graphics file from CorelDraw or Adobe Illustrator as a template. Drilling good holes in Plexiglas can be done with regular bits, but to make clean holes that don't have chipped edges requires a special drill bit. A special adhesive called Weld-On works well for joining Plexiglas, but it's very toxic. You can get it at most plastic retailers.

Adhesives

Hot glue is great way to adhere two things together. It may not last forever, but it will adhere two things long enough to see if they are worth attaching more permanently. It fills gaps and is also an excellent insulator. Self-tapping drywall screws allow you to work very quickly with wood. Gaffer's tape (not to be confused with duct tape) is a cloth tape that is easily hand-torn to different length and widths. You can write on it, it neatly covers seams, and it can be applied and taken off very easily. It's also got a strong adhesive, and holds well to many materials. Cable ties should already be in your kit for neatening and providing strain relief to your electronics. They also come in handy for strapping anything to anything with considerable force. Hose clamps are also very useful, especially when you have a lot of wires or a few heavy wires to contain.

Erector, Meccano, K'nex

Many toy kits make great structural material for projects. If you don't want your project to look like it was made from toys, build a skin or housing to cover it. The advantage these offer is the freedom to build stable structures to hold motors, gears, and other moving parts securely without having to fabricate your own struts and supports.

There are countless other construction materials, of course; these are just a few that we find convenient and quick to work with. If you've never done any construction of this sort, we recommend browsing the handyman section of your bookstore for more information or seeing if your local university, high school, or community organization offers any basic construction classes.

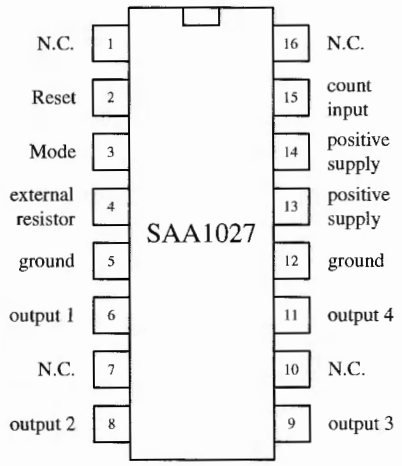
Black Cloth

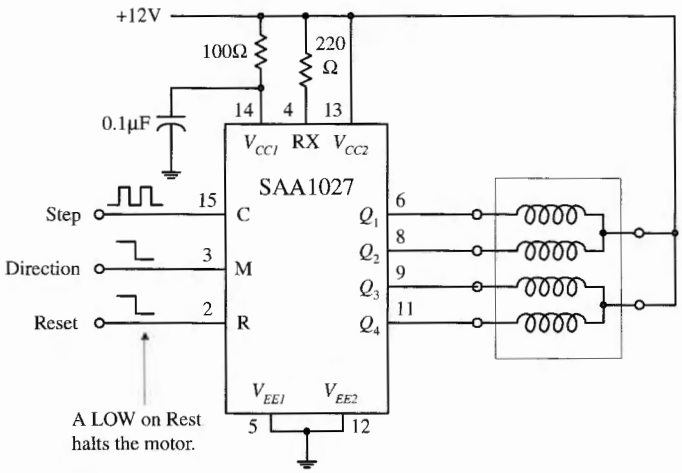
Black cloth (canvas, velour, and duvetyn curtains and the like) is not structural, but it hides a multitude of sins. It makes a great backing and discreetly hides your ungainly contraptions and electronics. It draws the eye to brighter and more lively things in front of it, and tells people to pay no attention to that computer behind the curtain. This frees you up to concentrate on getting the interaction right by getting your project in front of real people and worrying about neatening up the implementation later. We get a lot of our fabrics from Rose Brand (http://www.rosebrand.com), who will also custom-tailor stage curtains, but any local fabric store will do as well.

Conclusion

Motion can be one of the most exciting and expressive tools in the physical computing toolbox. By now, most people are desensitized to motion onscreen, whether it's a monitor or a movie screen. We respond to it, but only as filtered through our conscious mind. When you see an object that has its own physical mass move, however, it triggers a more visceral reaction. We've seen the simple movement of a motor electrify students hundreds of times. However, creating motion is one of the most time-intensive physical computing tasks and can create more confusion than sense if it's not carefully considered as part of the whole system. Just as a screen filled with animated icons for no reason can seem overly busy, a space filled with moving things can overwhelm anyone in it. Used sparingly and thoughtfully, it can get your users' attention like no other form of output. The first time you make something move with a microcontroller is the second most exciting moment after you light your first LED. When you've had some luck with this chapter, reward yourself with another celebratory hokeypokey.

chip has high-noise-immunity inputs, clockwise and counterclockwise capability, a reset control input, high output current, and output voltage protection. Its supply voltage runs from 9.5 to 18 V, and it accepts input voltages of 7.5 V minimum for high (1) and 4.5 V maximum for low (0). It has a maximum output current of 500 mA. Figure 13.13 will paint the rest of the picture.





M = 0				M = 1			
Q_1	Q_2	Q_3	Q_4	Q_1	Q_2	Q_3	Q_4
0	1	0	1	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	1	0	1	0
0	1	1	0	1	0	0	1
0	1	0	1	0	1	0	1

FIGURE 13.13

Count input C (pin 15)—A low-to-high transition at this pin causes the outputs to change states.

Mode input M (pin 3)—Controls the direction of the motor. See table to the left.

Reset input R (pin 2)—A low (0) at the *R* input resets the counter to zero. The outputs take on the levels shown in the upper and lower line of the table to the left.

External resistor RX (pin 4)—An external resistor connected to the RX terminal sets the base current of the transistor drivers. Its value is based on the required output current.

Outputs Q_1 through Q_4 (pins 6, 8, 9, 11)—Output terminals that are connected to the stepper motor.

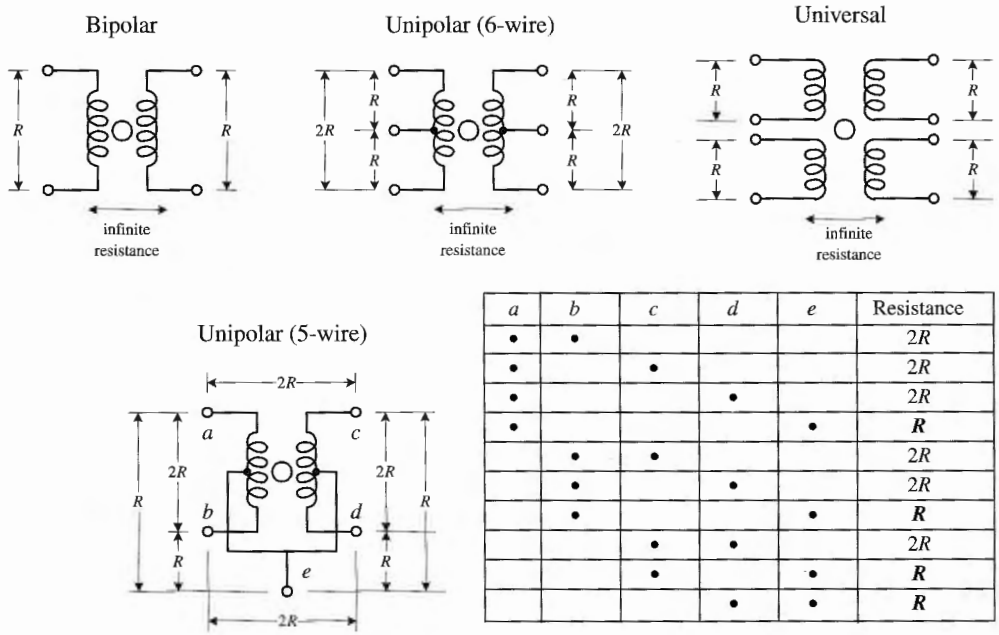
As mentioned, the SAA1027 is a classic chip (old chip). Newer and better stepper control ICs are available from a number of different manufacturers. If you are interested in learning more about these chips, try searching the Internet. You will find a number of useful Web sites that discuss stepper controller ICs in detail. Also, these Web sites often will provide links to manufacturers and distributors of stepper motors and controller ICs.

13.9 A Final Word on Identifying Stepper Motors

When it comes to identifying the characteristics of an unknown stepper, the following suggestions should help. The vast majority of the steppers on the market today are unipolar, bipolar, or universal types. Based on this, you can guess that if your stepper has four leads, it is most likely a bipolar stepper. If the stepper has five leads, then the motor is most likely a unipolar with common center taps. If the stepper has six leads, it is probably a unipolar with separate center taps. A motor with eight leads would most likely be a universal stepper. (If you think your motor might be a

variable-reluctance stepper, try spinning the shaft. If the shaft spins freely, the motor is most likely a variable-reluctance stepper. A coglike resistance indicates that the stepper is a permanent-magnet type.)

Once you have determined what kind of stepper you have, the next step is to determine which leads are which. A simple way to figure this out is to test the resistance between various leads with an ohmmeter.



a	b	c	d	e	Resistance
•	•				2R
•		•			2R
•			•		2R
•				•	**R**
	•	•			2R
	•		•		2R
	•			•	**R**
		•	•		2R
		•		•	**R**
			•	•	**R**

FIGURE 13.14

Decoding the leads of a bipolar stepper is easy. Simply use an ohmmeter to determine which wire pair yields a low resistance value. A low resistance indicates that the two wires are ends of the same winding. If the two wires are not part of the same winding, the resistance will be infinite. A universal stepper can be decoded using a similar approach. Decoding a six-wire unipolar stepper requires isolating two three-wire pairs. From there, you figure out which wire is the common by noticing which measured pair among the isolated three wires gives a unit R worth of resistance and which pair gives a unit of $2R$ worth of resistance (see Fig. 13.14). Now, decoding a five-wire unipolar (with common center tap) is a bit more tricky than the others because of the common, but hidden, center tap. To help decode this stepper, you can use the diagram and table shown in Fig. 13.14. (The dots within the table represent where the ohmmeter's two probes are placed within the diagram.) With the table you isolate e (common tap wire) by noting when the ohmmeter gives a resistance of R units. Next, you determine which of the two wires in your hand is actually e by testing one of the two with the rest of the wires. If you always get R, then you are holding e, but if you get $2R$, you are not holding e. Once the e wire is determined, any more ohmmeter deducing does not work—at least in theory—because you will always get $2R$. The best bet now is to connect the motor to the driver circuitry and see if the stepper steps. If it does not step, fiddle around with the wires until it does.