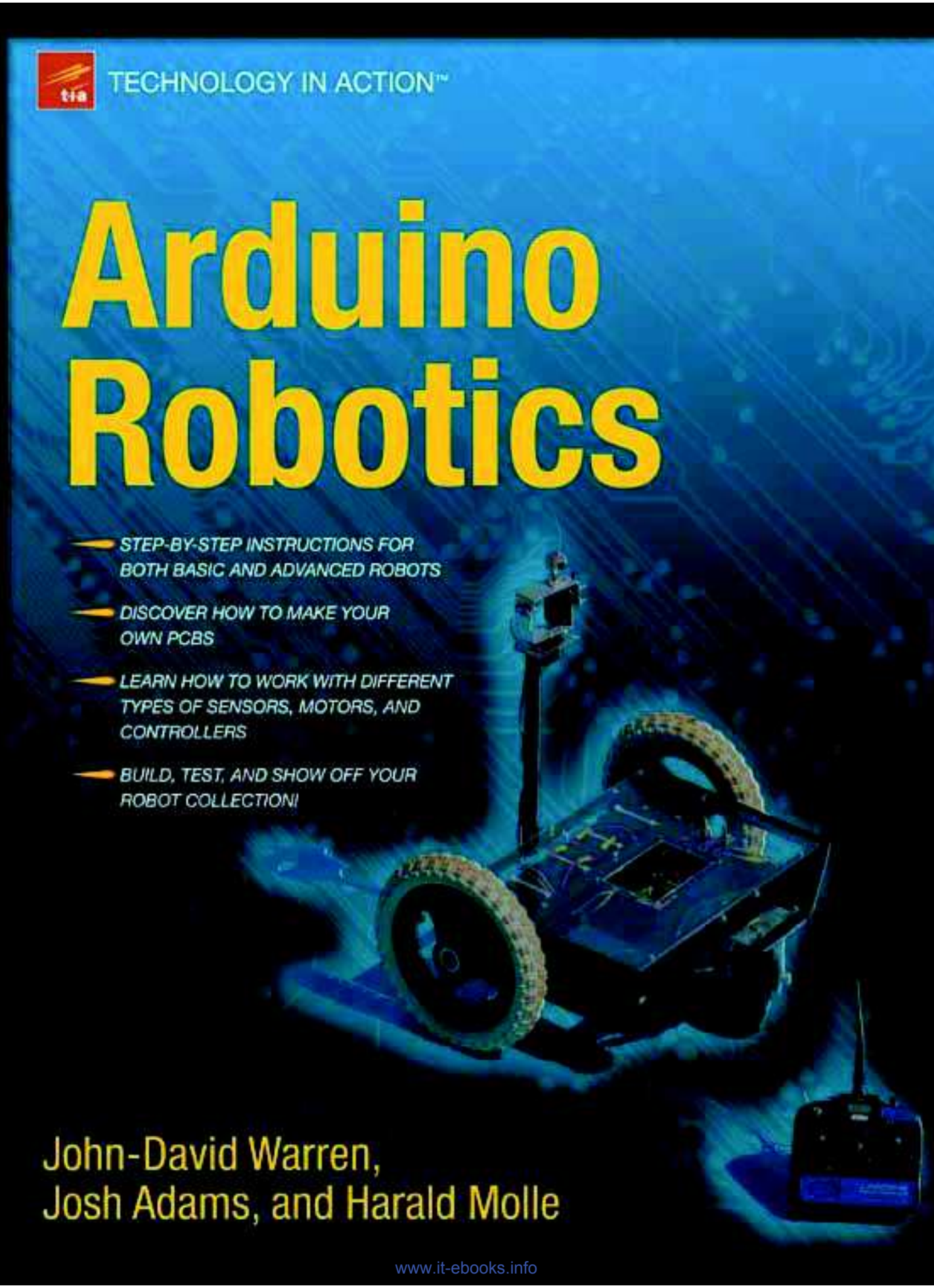

CHAPTER 2

Arduino for Robotics

With some of the basics of electricity, Arduino, and general robot building out of the way, we jump right in to some of the specific interfacing tasks that are needed to complete the projects in this book. In Chapter 1, the code examples use low-power components that can be connected directly to the Arduino (LEDs, potentiometers, R/C receivers, button switches, and so on). This chapter focuses on how to interface your Arduino to mechanical, electronic, and optical switches, as well as some different input control methods, and finally some talk about sensors.

First we discuss the basics of interfacing relays, transistors, and motor-controllers to the Arduino. We then discuss the various methods of controlling your Arduino—focusing on the popular methods of wireless control. Lastly, I give you my two cents about the many different types of sensors available for robotic use.

There are no code examples in this chapter, but the information presented is useful to understand the interfacing methods, control types, and sensors used throughout this book. Let's start by introducing some switching components that can enable the Arduino to control high-powered devices.

Interfacing Arduino

Because the Arduino can supply only around 40ma of current through any one of its Output pins, it is severely limited to what it can power by itself. A typical 5mm red LED requires about 30ma of current, so the Arduino has no problem lighting it up to 100%—but anything more, and it will struggle. To use the Arduino to control a high-powered device requires the use of an "amplifier." Also called a "signal-buffer," an amplifier simply reproduces a low-power input signal, with a much higher output power to drive a load.

A basic amplifier has an input and an output—the input is a low-power signal (like the Arduino) and is used to drive the larger output signal that will power the load. A perfect amplifier is able to switch the high-power signal as quickly and efficiently as the Arduino switches the low-power signal. In reality, amplifiers are not perfectly efficient and some heat is dissipated in the switching process, which often requires the use of a heat sink on the switching device and possibly a fan to remove heat (like the CPU in your computer).

There are different types of amplifying circuits that can be interfaced with the Arduino depending on the type of signal output used. For slow-switching signals using the digitalWrite() command, you can interface the Arduino to a high-power relay. For fast-switching PWM signals using the analogWrite() command, you must use a solid-state switch, which allows for full 0-100% digital output control. You can also purchase a preassembled electronic speed controller and use the Arduino to provide the input control signals.

First let's talk about an electrically activated switch called a relay, which can conduct very large amounts of current and can be controlled using the Arduino.

J.-D. Warren et al., *Arduino Robotics*

Relays

A *relay* is an electrical switch that uses an electro-magnetic solenoid to control the position of a mechanical power contactor. A solenoid is similar to a motor because it uses a magnetic field to produce physical movement of the solenoid cylinder—but instead of spinning like a motor output shaft, the solenoid cylinder moves back and forth in a linear motion. Most relays are encased in a plastic or metal housing to keep the moving parts free from outside interference and dust (see Figure 2-1).

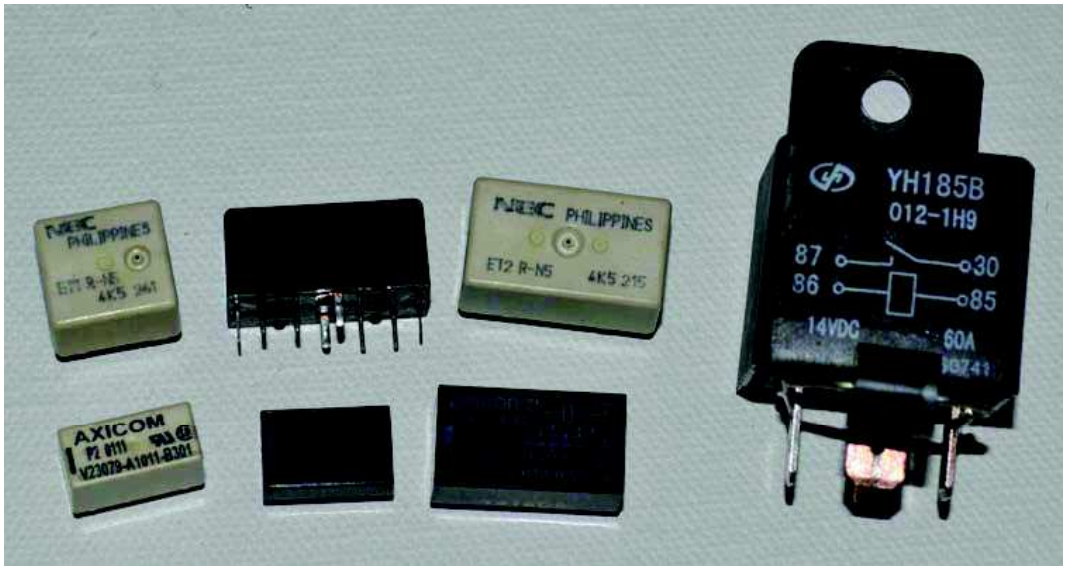

Figure 2-1. *Here you can see a variety of relays in small to large sizes. The three smaller relays on the bottom row are called "signal" relays, meaning their contacts are rated for less than 2 amps of current. The three relays on the top row are called "power" relays, ranging from 5 amp to 25 amp contact ratings. Lastly, the mammoth relay on the far right is an automotive power relay, which is rated at 60 amps.*

There are two parts to a relay: the solenoid and the contactor, and each is electrically isolated from the other. These two parts can essentially be treated as separate (but related) parts of a circuit, because each has its own ratings. The solenoid inside a relay has an electrical coil with a magnetic plunger that provides the movement needed to flip the contactor switch on and off. The relay coil should have the coil resistance listed as well as the operating voltage so that you can calculate how much current it will consume when in use. The contactor in a relay is where the high-power signal is switched. The contactor switch also has a voltage and current rating that tells you how much power you can expect the relay to conduct before the contacts fail.

Types of Relays

Relays are available with several different operation types depending on your application, so it is useful to understand how each type operates to make sure you get the right relay for the job.

- Normally-Open (NO): This simply means that the two power contacts of the relay are connected when the relay coil is turned on and disconnected when the relay coil is turned off.
- Normally-Closed (NC): This is the opposite of Normally-Open; the power contacts are connected when the relay is off and disconnected when the relay is on.
- Latching: This means that the contactor switch in the relay is not spring-loaded, and it stays in whatever position it is placed into until the polarity is reversed to the coil, which returns the contactor switch to its original position. This is comparable to a standard home light switch—it stays on until you turn it off.
- Non-latching: This is the "normal" type of relay that we use for failsafe switches. The relay contactor switch is spring-loaded and returns to the preset position unless power is applied to the coil. This is comparable to a momentary button switch—it stays on only while you press the button; otherwise, it springs back to the off position.

Relay Configurations

In addition to having different operating types, relays can have their contacts arranged in various configurations depending on the use. There are four common types of relays that we briefly discuss—each of these relays has only solenoid coil, but a varying number of power contacts. Any of these relay configurations can be Normally-Open or Normally-Closed as well as latching or Non-latching as described.

- Single Pole, Single Throw (SPST): This type of relay uses one coil to control one switch with two contacts—there are four total contacts on this relay (see Figure 2-2).

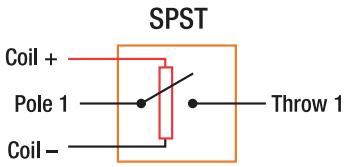

Figure 2-2. *This SPST relay has one pole, with one contact (a simple switch).*

- Single Pole, Double Throw (SPDT): This type of relay uses one coil to operate one switch with three contacts (see Figure 2-3). The middle contact is for the load, the upper contact is for Voltage1, and the lower contact is for Voltage2 (or GND). This relay has five total contacts and is useful for switching one contact (Pole 1) between two different sources (Throw 1-1 and 1-2) —also called a three-way switch.

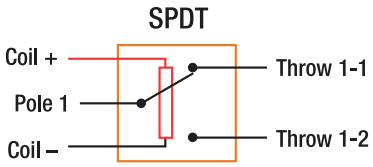

Figure 2-3. *This SPDT relay has one pole, with two contacts (a three-way switch).*

- Double Pole, Single Throw (DPST): This type of relay uses one coil to operate two independent SPST switches at the same time (see Figure 2-4). This relay has six total contacts and is useful for switching two loads at the same time—the two loads being switched can be associated (like a set of motor wires) or separate (like a dual-voltage power switch).

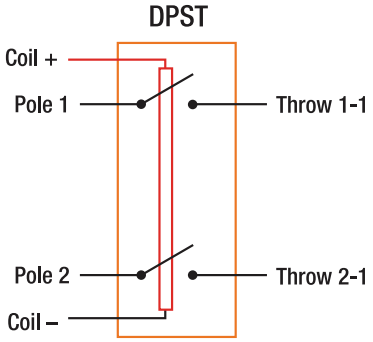

Figure 2-4. *This DPST relay has two poles, and each pole has one contact (a double switch).*

- Double Pole, Double Throw (DPDT): This type of relay uses one coil to operate two independent DPDT switches at the same time (see Figure 2-5). This relay has eight total contacts and can be configured as an H-bridge circuit, which is discussed in Chapter 3 (for controlling the direction of a load).

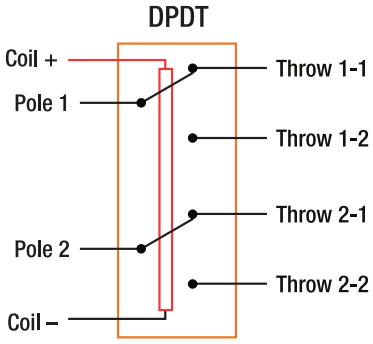

Figure 2-5. *This DPDT relay has two poles, and each pole has two contacts (a double three-way switch).*

Uses

Relays have the advantage of using thick copper contacts, so they can easily be used to switch high currents with a relatively small amount of input current. Because the solenoid takes some time to move the contactor, PWM does not work with a relay. The PWM signal appears to the relay as an Analog voltage, which is either high enough to turn the relay coil on or it just stays off—but it is not generally a good idea to use a PWM signal on a relay.

You can, however, use a relay to switch high-power loads using the Arduino—including AC and DC lighting, motors, heaters, appliances, and almost anything else that uses electricity. The relay is extremely useful in robotics, because it can both switch a high-power load and be controlled electronically (and thus remotely), which opens many possibilities for its use. You can use a power relay as an emergency power disconnect on a large remote-controlled robot or a remote power switch for an electric motor or lights.

Using two SPDT (three-way) relay switches, we can control the direction of a DC motor. In Figure 2-6, you can see that if both relay coils (control 1 and control 2) are activated, the upper motor terminal will be connected to the positive voltage supply and the lower terminal will be connected to the negative voltage supply, causing the motor to spin in a clockwise direction. If power is removed from both relay coils, the upper motor terminal will be connected to the negative voltage supply and the lower terminal to the positive voltage supply, causing the motor to spin in a counter-clockwise direction.

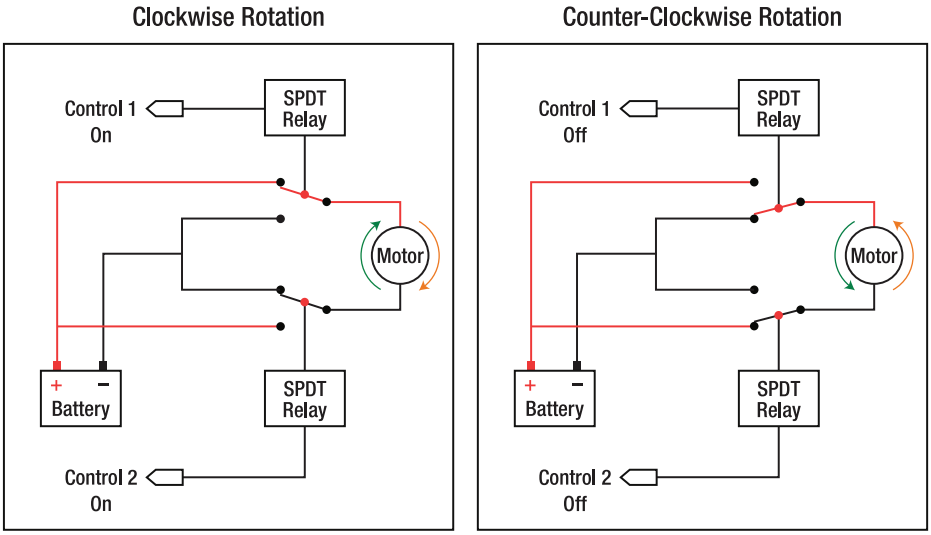

Figure 2-6. *These figures show how a DC motor can be controlled using two SPDT relay switches (or one DPDT relay switch).*

Before we can use the relay, we need to calculate how much power is needed to drive the relay coil. If the relay coil draws more current than 40mA that the Arduino can supply, an interface switch will be needed to turn on the relay coil using the Arduino.

Calculating Current Draw

To determine the amount of current that a relay draws, you must first determine the coil resistance by checking the relay datasheet. If this information is not available, you can measure the resistance with a

multi-meter. Using the coil resistance and voltage rating of the relay, use Ohm's law to calculate the current draw from the coil.

In Figure 2-7, you can see a sample of the datasheet from the Omron G5-CA series relays. As you can see, the relay is available with three different coil voltages (5v, 12v, or 24v). The coil resistance for each model is listed below along with the rated current. The 5v version of this relay coil has a rated current of 40mA, which is low enough to be powered by the Arduino without using an interface circuit.

Specifications

■ Coil Ratings

Item	Standard, high-capacity, or quick-connect terminals		
	5 VDC	12 VDC	24 VDC
Rated current	40 mA	16.7 mA	8.3 mA
Coil resistance	125 Ω	720 Ω	2,880 Ω
Must-operate voltage	75% of rated voltage (max.)		
Must-release voltage	10% of rated voltage (min.)		
Max. voltage	150% (standard)/130% (high-capacity, quick-connect terminals) of rated voltage (at 23°C)		
Power consumption	Approx. 200 mW		

Note: 1. The rated current and coil resistance are measured at a coil temperature of 23°
2. The operating characteristics are measured at a coil temperature of 23°C.
3. The "maximum voltage" is the maximum voltage that can be applied to the rela

■ Contact Ratings

Item	Standard		
	Resistive load	Inductive load (cosϕ = 0.4, L/R = 7 ms)	
Contact form	Single		
Contact material	Silver alloy		
Rated load	10 A at 250 VAC; 10 A at 30 VDC	3 A at 250 VAC; 3 A at 30 VDC	
Rated carry current	10 A		
Max. switching voltage	250 VAC, 125 VDC		
Max. switching current	10 A		
Max. switching power (reference value)	2,500 VA, 300 W	750 VA, 90 W	

Figure 2-7. *This is a sample portion of a relay datasheet; you can see both the coil and contact ratings.*

Even though this particular datasheet displays the rated current of the relay coil, some relays have only the operating voltage listed. In this case, you must manually measure the resistance of the relay coil using your multi-meter and then use Ohm's law to calculate the current draw.

From the datasheet in Figure 2-7, we use Ohm's law to verify the current draw for a 5v relay with a coil resistance of 125 ohms.

V = I * R

I = V / R

I = 5v / 125 ohms

I = 0.040 amps (40mA) —The datasheet is correct!

Back-EMF Considerations

Remember from Chapter 1 that a relay coil (solenoid) is an inductive type load and produces a jolt of Backwards Electro-Motive Force, anytime the solenoid is turned off. This Back-EMF can severely damage electronic switching components that are not protected with a standard current rectifying diode, like the 1n4004 diode used in Figure 2-8. The diode is placed across the terminals of the load (in this case, the relay coil) to prevent the Back-EMF from damaging the Arduino output pin.

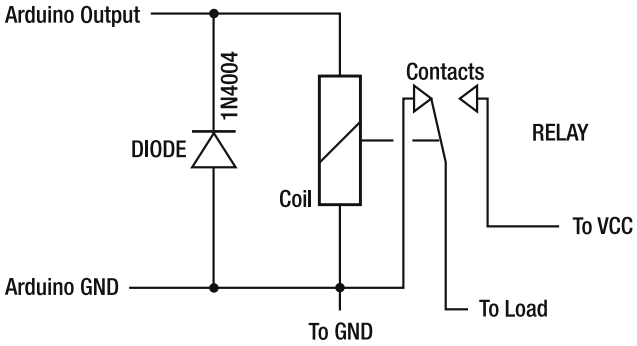

Figure 2-8. *This schematic shows the use of a diode around the relay coil to protect the Arduino output pin or other switching device from Back-EMF produced by the relay coil.*

Although the relay in Figure 2-7 *can* be driven directly by the current available from the Arduino, most power-relays require a bit more than 40mA to turn on. In this case, we need a signal interface switch to provide power to the relay coil using the Arduino. To do this, we first need to discuss solid-state (electronic) switches.

Solid-State switches

A *solid-state switch i*s one that switches an electrical load using doped silicon chips that have no moving parts. Transistors, Mosfets, photo-transistors, and solid-state relays are all examples of solid-state switches. Because solid-state electronics have no moving parts, they can be switched much faster than mechanical ones. You should check the manufacturer's datasheet for the part you are using, but PWM signals can typically be applied to these switches to provide a variable output to the load device.

There are two places that we can put a switch in the circuit to control power to the load. If the switch is between the load and the positive voltage supply, it is called a *high-side switch*. If the switch is between the load and the negative voltage supply, it is called a *low-side switch,* as shown in Figure 2-9.

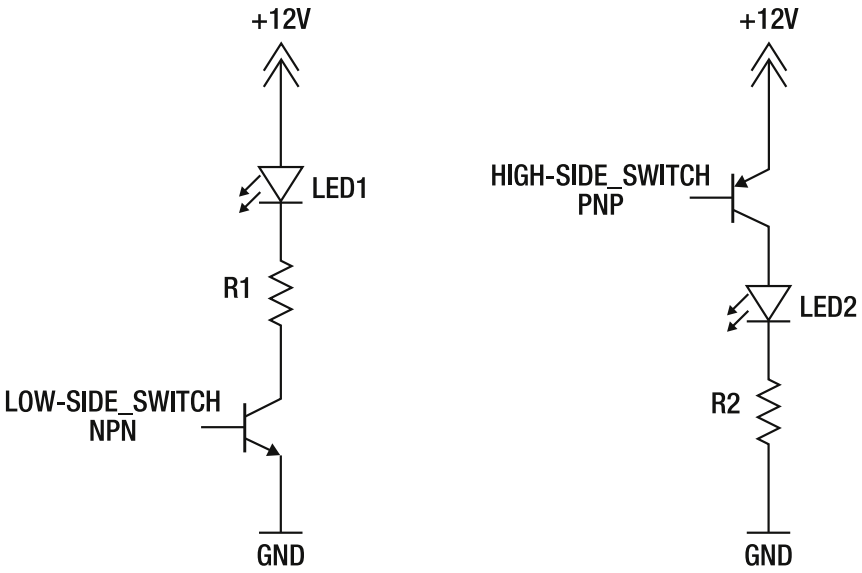

Figure 2-9. *Here you can see the difference between a high-side and low-side switch.*

Transistors

A *transistor i*s an electronic switch that uses a small input signal to switch a large output signal, using a common voltage reference. Transistor switches differ from normal switches (like relays) because they cannot be placed just anywhere in the circuit. A low-side switch must use a negatively doped transistor, whereas a high-side switch must use a positively doped transistor.

There are three common types of transistors that we use: the Bipolar Junction Transistor (BJT), the Metal-oxide Semi-Conductor Field Effect Transistor (MOSFET), and the photo-transistor. All of these devices are transistor (electronic) switches and operate as such, but each is activated using a different means. The BJT is activated by supplying a specific amount of electrical current to its base pin. The MOSFET acts like a BJT, but instead of current, you must supply a specific voltage level to the MOSFET gate pin (usually 5v or 12v). The photo-transistor is the most different of the three, because this transistor is not activated by an electrical signal, but by light. We can interface all three of these types of transistors directly to the Arduino.

All types of transistors have a voltage and current (amperage) rating in their datasheet—the voltage rating should be strictly adhered to, because going over this limit will likely destroy the transistor. The current rating should be used as a guide to determine at what point the switch becomes unusably hot. As mentioned, you can install a heat sink and cooling fan to remove heat from the transistor, which increases its current rating.

Bipolar Junction Transistor (BJT)

The most common type of transistor, the BJT, is a current-driven amplifier/switch whose output current is related to its input current, called “gain.” It is usually necessary to use a current-limiting resistor between the Arduino and BJT transistor to keep it from receiving too much current and overheating. Transistors also have no diode protection in case of Back-EMF from an inductive load, so if driving a motor or relay solenoid, you should use a protection diode as shown in Figure 2-10. If no Back-EMF

protection diode is used, the Arduino output pin can potentially be damaged if the transistor switch is harmed.

A basic BJT has three pins: the **Base** (input), **Collector** (output), and **Emitter** (common). The emitter is always connected to either the positive or negative voltage supply (the polarity depends on the type of transistor) and the collector is always connected to the load. The base pin is used to activate the switch, which connects the emitter and collector pins together. There are two types of BJT transistors that are labeled by the arrangement of the three doped silicon layers on the semi-conductor chip.

- Positive Negative Positive (PNP): Intended to be used as a high-side switch, the emitter of a PNP transistor connects to the positive voltage supply, the collector connects to the load, and the base is used to activate the switch. To turn this transistor off, its base pin must be equal to its emitter pin (positive voltage supply, or simply remove power to the base pin). Turning this transistor on is counter-intuitive because you have to apply a negative current, or a 0v (GND) signal to the base pin.
- Negative Positive Negative (NPN): Intended to be used as a low-side switch, the emitter of an NPN transistor connects to the negative voltage supply (GND), the collector connects to the load, and the base is used to activate the switch. To turn this transistor off, its base pin must be equal to its emitter pin (negative voltage supply). This transistor is turned on by applying a positive current to the base pin (see datasheet for specific transistor rating).

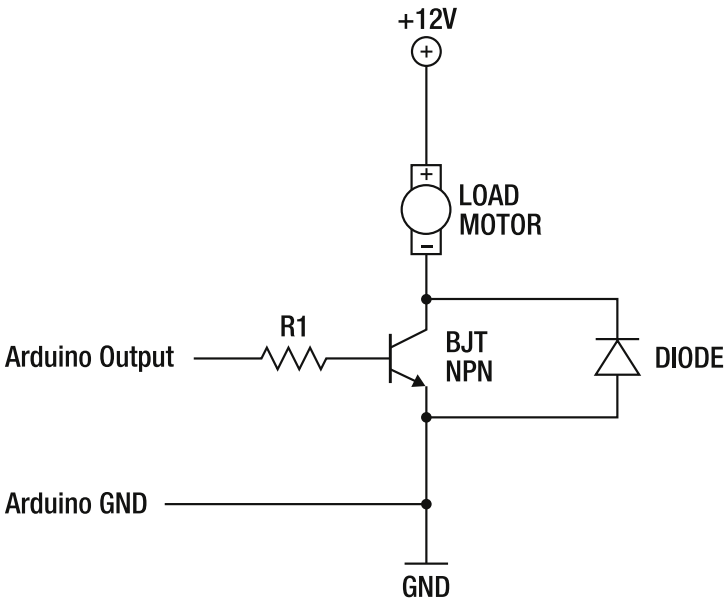

Figure 2-10. *This schematic shows a BJT used as a low-side switch to drive an inductive load (motor) with a Back-EMF protection diode around the switch. Notice that the transistor is driven through a current-limiting resistor (R1).*

Most BJTs require logic-level signals (+5v), to be applied to the base pin in order to activate the switch. Because a BJT is current-driven; when power is removed from its base pin the transistor quickly turns off. The current needed to switch on different transistors varies, but we will only use transistors

that can be driven at levels provided by the Arduino. The common 2n2222a NPN transistor can be fully switched on with only a few milliamps of current and it can switch nearly 1 ampere, so it can be used as a simple low-side amplifier switch. The 2n2907a is the PNP counterpart to the 2n2222a that is commonly used as a simple high-side switch. Both of these parts are available at Radio Shack, Sparkfun.com, and Digikey.com and are inexpensive (less than $1 each).

Mosfets

A MOSFET is a type of transistor that is voltage-driven instead of current-driven like the BJT. This type of switch is also capable of extremely high PWM speeds and typically has very low internal resistance, making them ideal for use in motor-controllers. Mosfets usually include an internal protection diode (as shown in Figure 2-11) to isolate the output voltages from the input signal and protect from Back-EMF produced by the load, so it is generally acceptable to interface the Arduino directly to a MOSFET switch; this is one less part that must be added into the circuit.

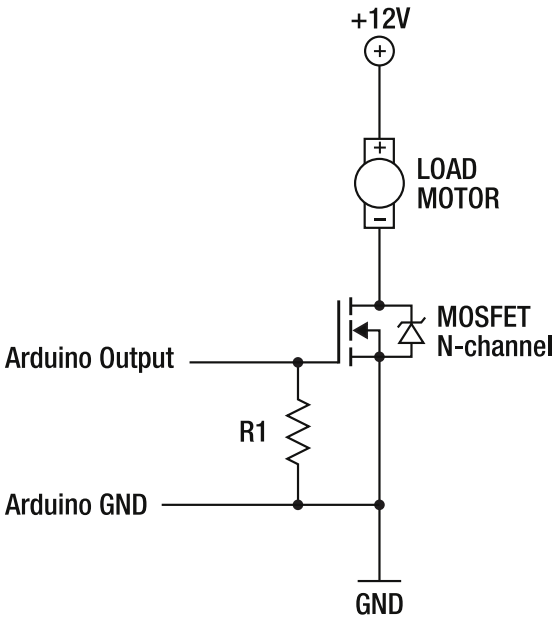

Figure 2-11. *This schematic shows a MOSFET switch (with built-in diode) used as a low-side switch to drive an inductive load (motor). Notice that there is no current-limiting resistor needed, but instead a pull-down resistor (R1) is used to keep the MOSFET switch turned off when not used.*

A MOSFET transistor is similar to a BJT transistor because they have corresponding pins and types. The MOSFET pins are labeled **Gate** (input), **Drain** (output), and **Source** (common), which correspond to the BJT transistors Base, Collector, and Emitter, respectively (see Figure 2-12). Also, a MOSFET is not labeled as NPN or PNP, but rather **N-channel** or **P-channel** to denote its mode of operation. For practical purposes, these terms are interchangeable. Because MOSFET switches are voltage-driven and consume very little current, it is not necessary to use a current-limiting resistor in series with the gate pin of a MOSFET (as with a BJT), but it is good practice to use a resistor from the gate to source pin (see R1 in Figure 2-11) to fully turn the switch off when not in use.

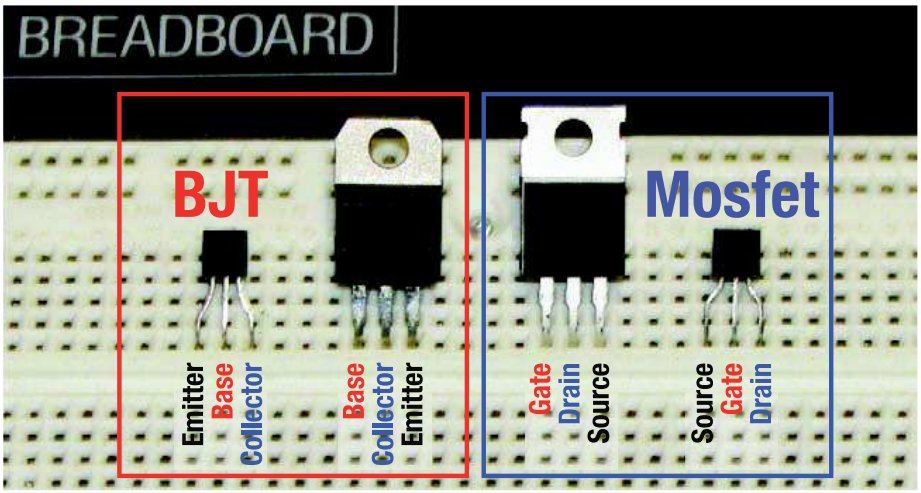

Figure 2-12. *Although they might physically look the same, the BJT (transistors) on the left are current driven, and the MOSFETs (transistors) on the right are voltage driven. Below each transistor, the pins are labeled—notice that the similar transistor packages have corresponding pins.*

Logic-Level vs. Standard

A normal MOSFET requires around 10v applied to the Base pin to fully turn on. Because driving anything above 5v with an Arduino requires using a level-shifter or amplifier, we use what is called a logic-level MOSFET for direct integration. A logic-level MOSFET can be turned on with a 5v "logic level" signal, which can be easily interfaced to the Arduino. Remember that a MOSFET requires a specific voltage level to be activated, but little current.

Mosfets are also sensitive to excessive gate-to-source voltages. If the limit is exceeded for even a second, it can destroy the MOSFET, so care should be taken to work within the voltage limits of the MOSFET. The maximum voltage that can be applied to the gate pin is listed in the datasheet as the "Gate to Source Voltage" or "Vgs" —this number is usually between 18vdc and 25vdc.

To drive a standard gate MOSFET, there are many different MOSFET-driver ICs that use logic-level input signals and a secondary power source (usually 12v) to send the amplified output signal to the MOSFET gate pin. Many MOSFET drivers are intended to provide the MOSFET gate pin with large amounts of current very quickly to allow for high frequency PWM switching speeds. Because of the higher PWM frequencies available with a high-current driver, we use MOSFET driver ICs in several of the projects in this book.

Mosfet Capacitance

Mosfets have tiny capacitors attached to their gate pins to maintain the voltage present at the gate. The capacitor charge enables the MOSFET to stay activated, even after the power is removed from the gate pin. Each time the MOSFET is switched, the gate capacitor must fully charge and discharge its current. For this reason, it is a good idea to ensure that the gate is forced to its off state by using a "pull-down" resistor to drain the capacitor when not actively powered by the Arduino (see R1 in Figure 2.11). Using a 10kOhm pull-down resistor from the gate pin to the source pin (gate to GND on n-channel, gate to VCC on p-channel) will be sufficient to keep the mosfet turned off when not in use.

As the PWM frequency that is applied to the MOSFET switch increases, the time allowed for the gate capacitor to charge and discharge decreases. As this happens, the gate-capacitor will require more current from the driver to fully charge and discharge in the shorter amount of time. If the current available from the driver is not sufficient to fully charge and discharge between switching cycles, the gate will be left in a partially conducting state, which can result in excess heating.

Saying that a MOSFET needs a lot of available current to switch quickly might seem confusing, because MOSFETs require a specific voltage to turn on and typically very little current. Although the 40mA that the Arduino PWM output pin can supply is plenty of current to fully switch a MOSFET on or off slowly, it is not enough to fully charge and discharge the MOSFETs gate-capacitor at high PWM frequencies where the MOSFET capacitor needs to be fully charged and drained 10,000 to 32,000 times per second!

Using a MOSFET driver IC (specialized signal-buffer) is the best way to drive a MOSFET switch because it can provide much more current during each switching cycle than the Arduino is capable of. A MOSFET driver can deliver enough current to the MOSFET to completely charge and drain the gate capacitor even at high PWM frequencies, which is important to reduce heat that is generated in the switch when it is not driven efficiently. You can also omit the pull-up or pull-down resistors from the gate pin when using a MOSFET driver to control a MOSFET—instead you should use a pull-down resistor at each input pin on the MOSFET driver IC, being driven from an Arduino PWM output pin.

On-State Resistance—Rds(On)

One of the most important properties of a MOSFET is the internal resistance between its Drain and Source pins (Rds) when the switch is on. This is important because the resistance of the switch determines the amount of heat that it will create with a given power level. We can determine the maximum Rds(On) value by checking the manufacturer's datasheet. The maximum power that is dissipated is determined using the Rds(On) resistance and the continuous current (in amps) that will pass through the switch.

Calculating heat using Rds(On) and amperage of DC motor:

How much total power will be passed through a MOSFET with an Rds(On) = 0.022 ohms (22 milliohms) and a continuous current draw of 10 Amperes? Use the Ohm's law pie chart from Figure 1-3 in Chapter 1—we want to know the heat produced in Watts, and we know the resistance of the MOSFET and the continuous current level passing through the circuit. So we need to use the formula: Watts = Current² x Resistance.

$$W = I^2 * R$$

$$W = 10 \text{ amps}^2 * 0.022 \text{ ohms}$$

$$W = 100 \text{ amps} * 0.022 \text{ ohms}$$

$$W = 2.2 \text{ watts}$$

This means that a single MOSFET with an Rds(On) = 0.022 ohms dissipates 2.2 watts if you try to pass 10 amperes through the switch. In my experience, dissipating more than 2 watts from a MOSFET in the TO-220 package results in excessive heating of the MOSFET. Any time more heat dissipation is needed, it is a good idea to add a heatsink or cooling fan to reduce the operating temperature and get rid of more heat. A good heat sink and fan can greatly increase the amount of power (or heat) allowed to

safely pass through the MOSFET. If cooling methods do not suffice, you can arrange multiple identical MOSFETs in a parallel circuit to multiply the amount of current the switch device can handle. If you place multiple MOSFETs in parallel, it still operates only as one switch because they are opened and closed simultaneously, and their common pins are connected.

Parallel Mosfets

One of the most useful features of a MOSFET is the capability to arrange multiple switches in parallel for increased current capacity and decreased resistance. This is done by simply connecting the Drain terminals together and the Source terminals together (see Figure 2-13). The Gate terminals should be driven by the same control signal, but each MOSFET should have its own gate resistor to divide the total available current equally to each MOSFET used in parallel—these resistors can be a very low value from 10 ohms to 330 ohms.

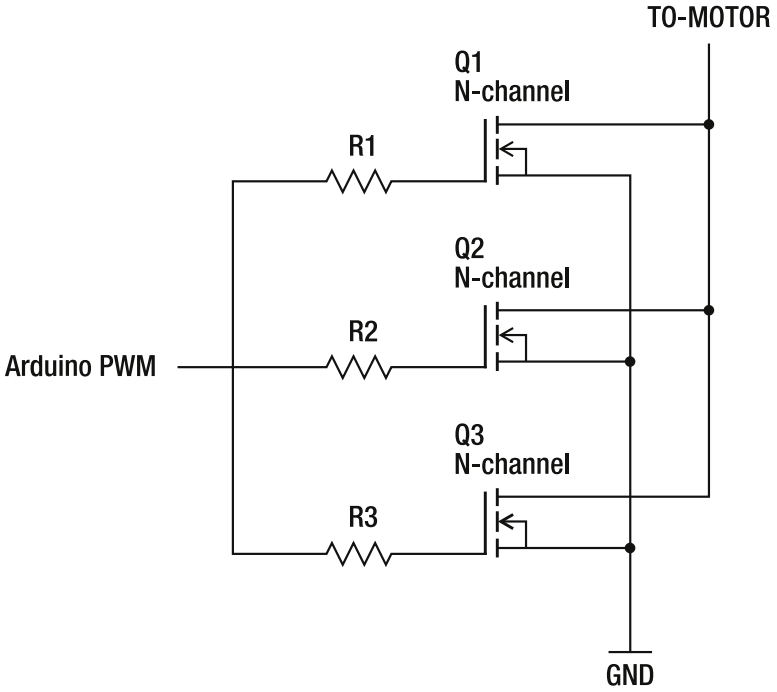

Figure 2-13. *Three MOSFETs (Q1, Q2, and Q3) are arranged in a parallel circuit (all like pins tied together) to allow three times the current flow and a third of the resistance as using only one MOSFET. The resistors (R1, R2, and R3) are in place only to evenly distribute the available current from the Arduino, but are not required.*

■ **Note** The voltage limits of the MOSFETs do not change even when using the parallel method. If the voltage limit is exceeded, you will likely blow up every MOSFET that is connected!

The total current that can be transferred through a parallel set of MOSFETs is equal to the amount of current that can be passed by a single MOSFET, times the number of MOSFETs used in parallel. In addition, the total resistance of the parallel set of MOSFETs is equal to the Rds(On) rating divided by the number of MOSFETs in the parallel circuit. This means that by using two MOSFETs in parallel, you decrease the resistance by half—and when the resistance is decreased, so is the heat dissipation.

Photo-Transistors

A photo-transistor operates like a standard NPN transistor, except that it is activated using infrared light from an LED instead of electrical current. These transistors are commonly used for line-following robots to detect reflective light differences on colored surfaces. If the infrared emitter and detector are enclosed in an IC package, the device is an optical-isolator, because the low power device (infrared emitter) is electrically isolated from the high power switch (photo-transistor), enabling the input and output circuits to be separated (they have different power sources).

This type of switch is like a transistor/relay hybrid; it has electrical isolation like a relay, but the switch is interfaced as a transistor; the unique feature that you get with a photo-transistor is an electrically isolated switch with PWM switching capabilities. In Figure 2-14, the base is driven using light from an infrared LED connected to the Arduino (using current-limiting resistor R1), the collector (pin 4) is connected to the negative load terminal (as a low-side switch), and the emitter (pin 3) is connected to the GND supply.

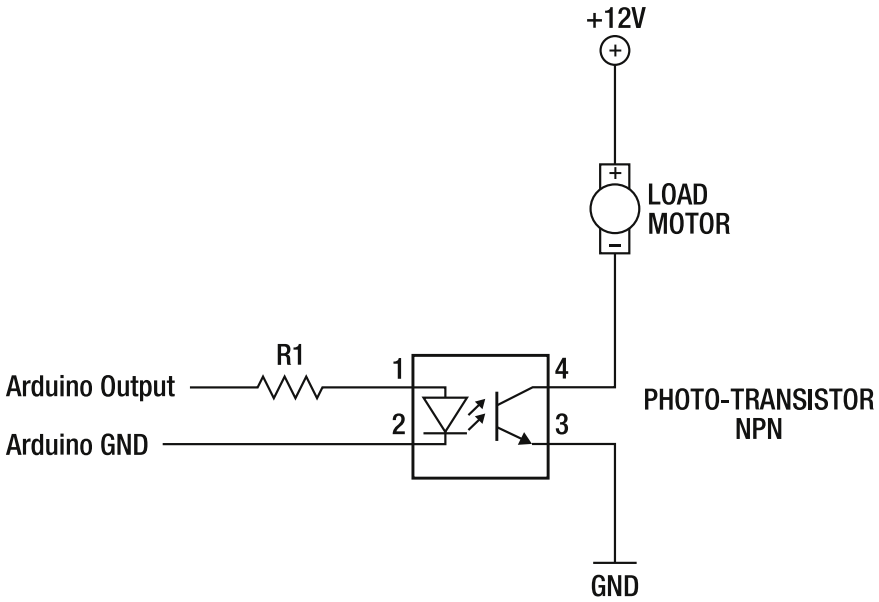

Figure 2-14. *This optical-isolator (IRED and photo-transistor pair) is used as a low-side switch to drive a motor. Because the only thing connecting the Arduino to the Load is a beam of infrared light (no common GND signal), it is not required that you use a protection diode on the switch (though it is recommended).*

$25 each. An adapter board is needed because the pin spacing of each Xbee radio is 0.05 inch, which is not compatible with a breadboard or perforated prototyping boards, which use 0.1 inch spacing. In Figure 2-20, you can see two standard Xbee radios (Sparkfun.com part# WRL-08665), a Sparkfun Xbee Explorer Regulated breakout board (Sparkfun.com part# WRL-09132) to connect to the Arduino, and a Sparkfun Xbee Explorer USB breakout board (Sparkfun.com part# WRL-08687) for connecting to your computer. The makers of the Xbee radios have also created a software program called X-CTU, which is used to change settings on the Xbee radios while connected to your computer—the X-CTU software is free to use, but currently works only in Windows.

Sensor Navigation

Although creating a control link that converts user input into robotic output can be extremely useful, there are some tasks that require the robot to make decisions of its own without consulting a human. The first three robotic projects in this book (chapters 4, 5, and 7) use some type of external awareness to direct the robot to its destination without using any user guidance.

Believe it or not, you are actually an autonomous (self-controlled) being that uses several different "sensors" to help determine the environment around you. Your eyes, ears, nose, hands, and mouth each have their own sensation that your brain can interpret into some form of intelligence. From these sensors, your brain is able to make informed decisions about how your body should proceed and keep you safe from harm. Along the same lines, a robotic sensor is a device that is attached to a robot to gather information about its surrounding area. Without sensors, the robot would not have any way of knowing what is around it or how to proceed. This is the easiest way to add intelligence to your bot.

There are many types of sensors and each reads the environment differently, so it is common to add several types of sensors to one robot to effectively navigate around obstacles and gather important information. A sensor can measure light, distance, proximity, heat, gas, acceleration, angle, humidity and moisture, contact, and rotational position (among others). We focus on the sensors that are readily available and offer the most versatility for the price.

Contact Sensing

The most simple type of sensor that can be implemented is the contact switch, which simply tells the robot whether or not it is touching something. This type of sensor is commonly called a "bump-switch" and is used on the iRobot Roomba robotic vacuum cleaner to determine when it has bumped into a wall or other object. The main caveat of this sensor type is that it requires the robot to make physical contact with an object before it is detected.

Bump Switch

The bump switch is a simple form of sensor because it consists of as few as two electrical contacts (see Figure 2-21). If the contacts are touching, the switch is closed; otherwise, it is open. We use this form of switch as a method of telling the robot when it has run into something. If we place them at several spots on the bot, we not only know when the bot has bumped into something, we can also determine the best direction to travel to keep from hitting the same objects again.

This type of sensor is also useful as an over-travel switch. These are commonly installed on garage-door openers. When the door opens to a certain point, it touches the over-travel switch and the main board receives a command to stop the lifting motor. This is how it knows where to turn it off.

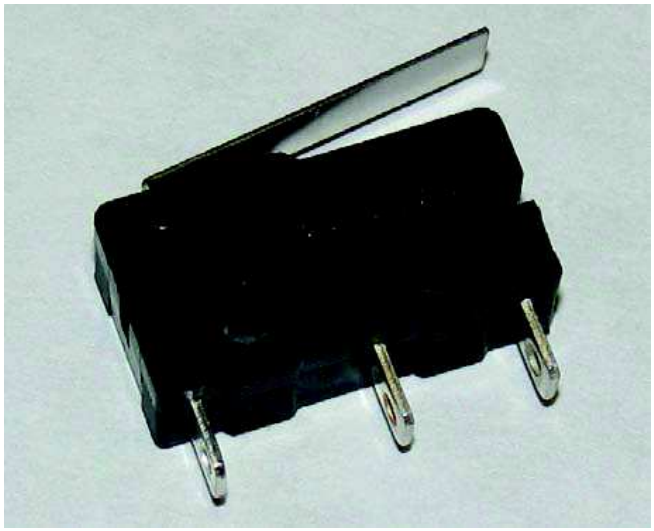

Figure 2-21. *A typical tactile bump-switch with lever*

This type of sensor (or any switch) is read as a digital input using the digitalRead() command (see Listing 1-1 in Chapter 1).

Distance and Reflection Sensing

Range detection is useful when trying to determine whether an object is near without the robot having to touch it. A good range detector can calculate the distance from an object with accuracy down to the nearest inch. Range detection sensors use reflected sound or light waves to measure the distance between the sensor and any obstacle within range. Different range detection methods result in different effective ranges, accuracy, and prices. Range detection sensors can have an effective sensing range from 1 centimeter to 25 feet and cost anywhere from just a few dollars for an IR range finder to several thousand dollars for a Laser range finder. We use infrared detection for Linus in Chapter 4 and ultrasonic range finders on Wally in Chapter 7.

IR Sensor

IR detectors use an infrared emitter to send out IR light "packets" and a detector to check for reflections of that light that have bounced off any nearby objects. By measuring the amount of time that the light takes to return to the detector, the sensor can determine the distance from that object. IR finders can detect objects at distances up to 5 feet away—use an ultrasonic range finder for farther distances. The Sharp GP2 series of infrared proximity sensors are available at Sparkfun.com (part# SEN-08958) for under $15 and can be used for short-range object detection up to 5 feet (see Figure 2-22).

Figure 2-22. *This is the Sharp GP2 IR finder (Sparkfun.com part# SEN-08958).*

A simple infrared emitter and detector pair can be used at close ranges (under 3 inches) to determine the approximate reflectivity of a surface to infrared light. These simple IR emitter and detector pairs are the basis for the line-following robot in Chapter 4. The emitter and detector are mounted side by side and facing the same direction. The emitter is constantly sending a stream of infrared light toward the ground, whereas the detector is constantly reading the reflections of the light that is bouncing off the ground.

We use a piece of reflective tape as a guidance track for the bot to follow. As the bot moves away from the reflective tape, the IR sensors on each side begin to receive less IR light and therefore adjust the motor outputs to keep the bot centered on the tape. Using this simple guidance scheme, we can easily modify the path that the bot will follow by altering the path of the reflective tape. There are several different types of infrared emitter and photo-transistor packages that work for a line-following robot (see Figure 2-23).

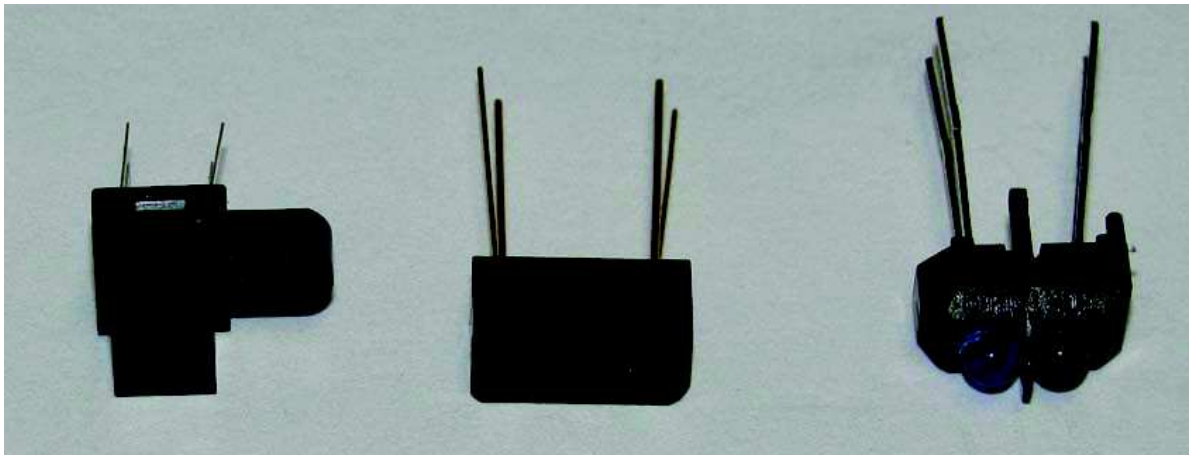

Figure 2-23. *These are three different types of infrared emitter and detector pairs. The IR pair on the far right is the type used in Linus, the line-following robot from Chapter 4. These sensors range in price from $1 to $3 each from Digikey.com.*

■ **Note** Many household appliances that use a remote control contain a useful IR receiver like the one you can find at Radio Shack or Digikey. If you happen to have a broken VCR, DVD player, TV, or stereo that you don't mind dismantling, you can de-solder the IR sensor from the PCB and save a few bucks. They typically only have three pins: +5v, GND, and Signal.

Ultrasonic Range Finder

The ultrasonic range finder uses high frequency sound waves that are reflected off nearby objects to calculate their distance. Some ultrasonic sensors require a microprocessor to both send and receive a signal from the sensor, whereas other sensors calculate the distance within the sensor and have a proportional output signal that is easily read by the Arduino.

Ultrasonic range finders are available in a variety of beam angles that determines the width of the detectable area. A narrow beam angle is better suited to detect objects farther away, whereas a broad beam angle better detects objects at short distances. These sensors are typically between $30–50 and can be easily read by the Arduino.

The Maxbotix brand of ultrasonic range finders have built-in processing that enables it to output independent serial, analog, and PWM signals at the same time to increase interfacing flexibility (see Figure 2-24). These range finders accurately measure distances from about 6 inches to 25 feet, and are well suited for obstacle avoidance and detection on robots. I prefer to use this brand of ultrasonic range finder because it is reliable and easy to interface to the Arduino using any of the three built-in output signals.

Figure 2-24. *The MaxBotix LV-EZ0 ultrasonic range finder with an effective range of 6 inches to 25 feet (Sparkfun.com part# SEN-08502)*

Laser Range Finder

This type of range finder uses a laser to scan the objects around it, much like a laser scanner at the grocery store checkout. A laser range finder can have a viewing angle of up to 240 degrees, giving it a

much wider view of its surroundings than other sensors. Each time the laser makes a rotation, it takes distance readings at set intervals. When the rotation is complete, the signal is compiled to create a snapshot of the surrounding area. Although this sensor has advanced features, the maximum detection range is around 15 feet and they are expensive (usually costing around $1,000). Until the price comes down a bit, we won't test any of these units.

Orientation (Positioning)

There are several different sensors that can determine one or more aspects of a robot's position or orientation. A GPS sensor can tell you where the sensor is on a map using latitude and longitude coordinates, whereas an accelerometer and gyroscope can tell you the angular position (tilt) or rotational speed of your robot. Using these sensors, we can create an auto-leveling platform for a Segway type robot or upload a set of GPS coordinates for a robot to navigate to.

Accelerometer

An accelerometer measures gravitational force or acceleration. By tilting an accelerometer along its measured axis, we can read the gravitational force relative to the amount of tilt. Accelerometers are available with up to three axis of sensing and can be directly interfaced to the Arduino with an Analog output signal. Many devices currently use accelerometers for input control, shock detection, stabilization platforms, and auto-leveling or tilt interfaces—you can find these devices inside of digital cameras, cell phones, laptop computers, and Nintendo Wii controllers to name a few.

Most accelerometers available today are small surface mount components, but there are many different breakout boards that have the sensor and all necessary filtering components (resistors and capacitors) soldered into place, so you can easily interface them to an Arduino. Sparkfun.com has a large selection of these "Arduino-ready" sensor boards in different configurations ranging from $20 to $50.

There are three axes that can be measured by an accelerometer and they are labeled X, Y, and Z, which correspond to the roll, pitch, and yaw respectively (see Figure 2-25). A single axis accelerometer measures either the X or Y axis, a dual axis accelerometer measures both X and Y axes, and a triple axis accelerometer measures all three axes. Each measured axis represents a separate Degree of Freedom (DOF) from the sensor—thus a triple axis accelerometer might be labeled as 3 DOF.

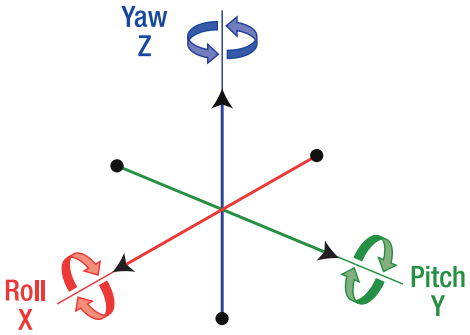

Figure 2-25. *This figure shows the three axes of rotation: roll, pitch, and yaw with their corresponding accelerometer symbols X, Y, and Z.*

Accelerometers are used for measuring gravitational changes, but they are also extremely sensitive to vibrations and sudden movements or shocks, which can cause the output signal to become distorted.

Current Sensor

A current sensor is used to measure the amperage passing a given point at a given time. If either motor draws an excess amount of current, we can program the Arduino (which controls the motor-controller) to stop driving that motor for a specified amount of time (1–2 seconds) to keep from overheating. By protecting the motor-controller, it is far less likely to fail.

There are several types of current sensors, but I prefer to use a current sensor IC, like the ACS-712 (5 amp) or ACS-714 (30 amp) from Allegro MicroSystems (Figure 2-29). Although only available as an 8-SOIC (Small Outline IC) surface mount package, it is easily interfaced to both the Arduino and a motor. The IC only needs +5v and GND signals to begin outputting an analog voltage at the VOUT pin, which is easily read using the Arduino on any analog input pin—you can even use the Arduino regulated +5v supply to power the current sensor IC.

Figure 2-29. *The ACS-712 bi-directional current sensor can be used in series with one of the load terminals to measure the amperage level. It can easily be read using an Analog input on the Arduino.*

In Figure 2-30, you can see a simple schematic depicting how you might connect the ACS714 current sensor to your Arduino. The current sensor IC needs a 5v power supply and a bypass capacitor connected from the filter pin to GND. You must then route at least one of the motor supply wires through the current sense pins of the sensor (as shown in Figure 2-30).

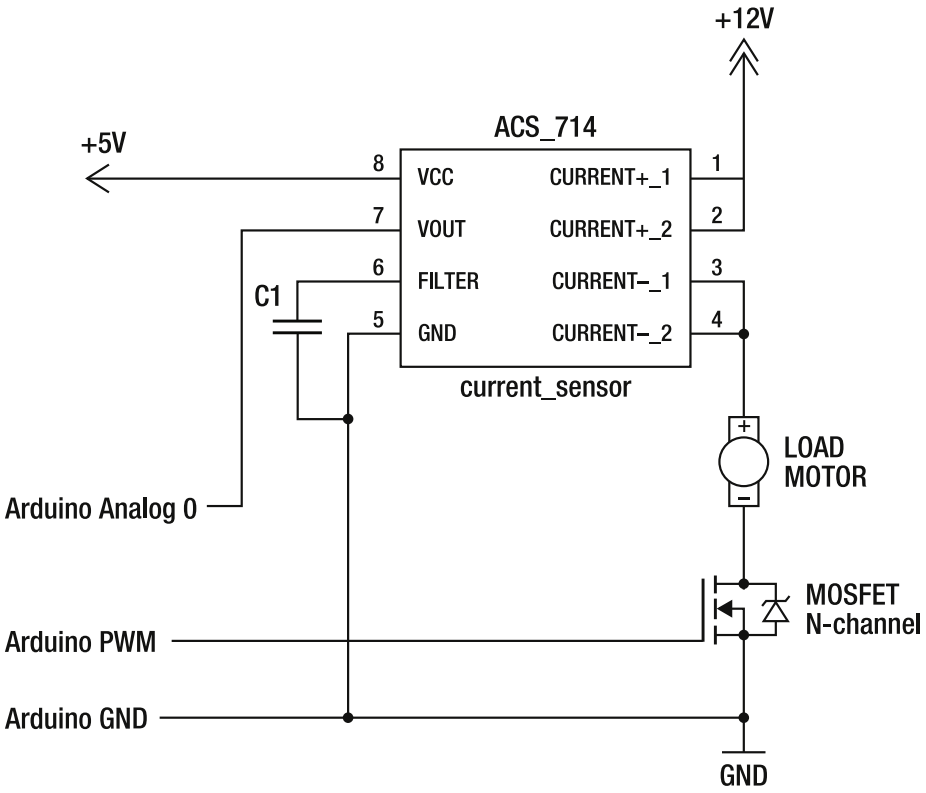

Figure 2-30. *A simple schematic for the ACS-712/714 current sensor IC in a circuit*

In Chapter 8: The Explorer-Bot, we use the ACS-714 bi-directional 30-amp current sensor (big brother to the ACS-712 shown in Figure 2-29) to sense the current passing through either drive motor. Using the Arduino to monitor the analog output voltage of the current sensor, we can tell the Arduino to stop sending motor commands if the reading from the current sensor is above a certain level—this is called "over-current" protection and can keep a motor-controller from destroying itself.

Summary

This chapter briefly discussed the various interfacing and control methods that are used in this book, as well as some of the different sensors.

Relays are reliable and easy to interface, but produce Back-EMF and must use a protection diode to drive directly from the Arduino. Transistors are available in several forms and can be driven directly by the Arduino, but must be specifically placed as either a high-side or low-side switch. ESCs are available for those who do not want to build their own circuits, but typically cost more to buy than building a similar motor-controller using transistors or relays.

We then talked about some various control methods using both wired and wireless links, including infrared controllers and radio control. Radio control is implemented using a variety of different methods, the most common of which is 2.4GHz. We use 2.4GHz hobby radio control systems and Xbee wireless serial links to both control and monitor robotic functions. Wireless radio control is used on the Explorer-bot, Lawnbot, and Battle-bot projects in this book.

The stroke of a linear actuator refers to the maximum distance it can extend. The speed of the actuator tells you how fast it will travel, usually rated in inches/second. The power of the actuator is determined by the power rating of the motor that is driving it and is usually rated by the maximum load capacity in pounds that the actuator can lift. It is typically fine to use a relay to control a linear actuator for simple On/Off control, unless you need extremely precise control in which case you should use a motor-controller.

Calculating Power

Because the amperage of a motor varies depending on the load, most DC motors list the voltage level at which it can safely operate. Although DC motors are usually forgiving and can be slightly over-powered without causing problems, an excess voltage level can burn up the motor coil.

As discussed in Chapter 1, the amperage that is consumed by a motor is dependent on the voltage level and the internal resistance of the motors coil. After the operating voltage is decided, you can measure the motor's coil to determine its resistance, and lastly use the voltage and resistance to calculate the amperage of the motor. With the amperage and voltage known, you can select a properly sized motor-controller for the motor.

Driving

The DC motor is the simplest motor to power; apply a positive signal to one wire and a negative signal to the other and your motor should move, as shown in Figure 3-8. If you swap the polarity of the wires, the motor will spin in the opposite direction.

The speed of the motor is dependent on the positive supply voltage level—the higher the voltage, the faster the motor shaft spins. The power of the motor is its capability to maintain its speed, even under a load and that is determined by the amperage available from the power source—as the workload of the motor increases, more amperage is drawn from the batteries.

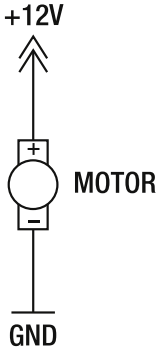

Figure 3-8. *To power a DC motor, simply connect one wire to the Positive supply and the other wire to the Negative supply.*

Some of our robots have powerful motors that can operate at up to 24vdc—if the entire 24 volts is applied to the motor all at once, it is likely to spin the tires or pop a wheelie! We don't want to break any of our equipment or hit anyone nearby because our bot launches when we turn the motors on, so we will use a motor-controller to vary the voltage to the motor from 0v to the power supply voltage (in most

cases, 6v, 12v, or 24v). This enables the bot to start slowly and work its way up to full speed, which causes less strain on the batteries during start-up and provides more precise control.

We can vary the voltage level to the motors by using a pulse-width modulation PWM signal to determine the output duty-cycle, or percentage of On time. Because using PWM means that the output is either fully on or fully off, the motors receive as much amperage as the power supply allows for the given duty-cycle, and the power can be varied from 0% to 100% for full speed control.

Finding the Right Motor

DC motors are in virtually any device that has moving parts—you can harvest useful DC motors from old cassette tape players, VCRs, toys, and cordless tools. Salvaging a DC motor is usually easy because they are rarely ever soldered to a printed circuit board (PCB), so you simply unplug the wires and remove any fasteners that are holding the motor in place. If the wires are soldered into place, just cut them leaving as much wire connected to the motor as possible (unless you plan on soldering your own wires to the motor terminals). Once removed, you can test the motor by powering it with a 6v or 12v battery (depending on its size).

As previously mentioned, gear-motors reduce the speed of a motor shaft to a usable RPM for driving a robot. When salvaging parts, you might come across a motor assembly that has plastic or metal reducing gears attached to the motor; you can re-use these gears and create your own makeshift gear-motor. Gear-motors and gear assemblies can also be found at surplus and commercial websites.

Surplus:

- `www.allelectronics.com`
- `www.goldmine-elec.com`
- `www.alltronics.com`

Commercial:

- `www.Sparkfun.com`
- `www.trossenrobotics.com`
- `www.pololu.com`
- `www.superdroidrobots.com`
- `www.robotmarketplace.com`

You can find 12v automotive windshield-wiper motors at your local junkyard that can be used as drive motors for a medium-sized bot. You can also find powerful motors at your local thrift-store by looking for cordless drills that have bad battery packs or cosmetic blemishes, but working motors and gear boxes.

The H-Bridge

When driving a DC motor in only one direction, we do not need any special circuitry to switch the motor on or off; a simple switch in series with one motor terminal will do. But to reverse the polarity of the voltage of the motor terminal, we need a *half-bridge* circuit or push-pull driver. This circuit uses two switches (S1 and S3 as shown in Figure 3-9) to provide a path from one motor terminal to either the Positive voltage supply or the Negative voltage supply (Ground). By using only one of these switches at a

time, a short-circuit is avoided—the other motor terminal is permanently connected to either VIN or GND

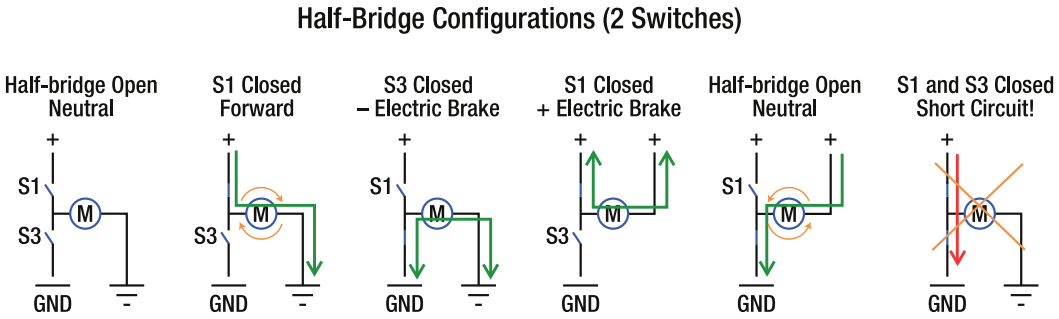

Figure 3-9. *Various half-bridge states*

The bridge is used to route the correct polarity to the motor terminals at the appropriate time. To avoid a short-circuit, you should *never* close both switches on the same side of the bridge (both the positive and negative) at the same time (see Figure 3-12). To control the polarity to both motor terminals, we need two identical half-bridges arranged in an H-bridge (see Figure 3-10).

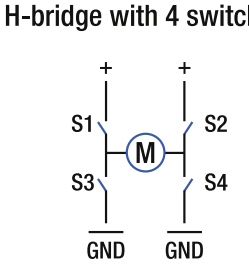

Figure 3-10. *Notice how the circuit looks like the letter "H," which is why they call it an H-bridge.*

For the motor to spin, the battery current must flow from the Positive supply, through the Motor, and to the Ground supply to complete the circuit. To make this happen we must open one switch from each side of the bridge, one Low-side and an opposite High-side—that means we can either turn on S1 and S4 to go Forward, or we can turn on S2 and S3 to go in Reverse. The direction of the current flow through the motor terminals determines the direction that the motor spins. We can manipulate the flow of the current by closing the two corresponding switches together to give us directional control of the motor. If all four switches are open (disconnected), the motor is *coasting*, meaning there is no path for the current to travel.

Generating a Brake

There is also an acceptable condition called *electric-braking*, which refers to connecting both motor terminals to the same voltage supply—as opposed to leaving them disconnected. Because most DC motors act as a generator if you spin the motor shaft, by connecting both terminals to either the Positive supply or the Ground supply, we are essentially trying to force the generated electricity back into the

same supply (see Figure 3-11). This results in the motor resisting to spin—that is, it will keep the motor shaft from moving by forcing opposing voltages into the same supply. We can tell the Arduino to keep both Low-side switches closed to form an electric brake when the bot is in Neutral to make sure it does not roll down a hill or move without being commanded. Alternatively, if all switches are left open in Neutral (coasting), there will be no resistance to the motor generating electricity—so if it is on a hill, it will roll.

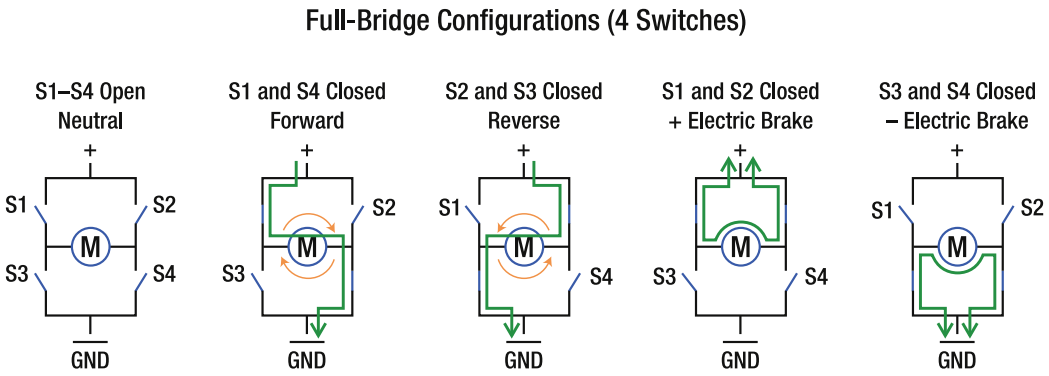

Figure 3-11. *Acceptable H-bridge states*

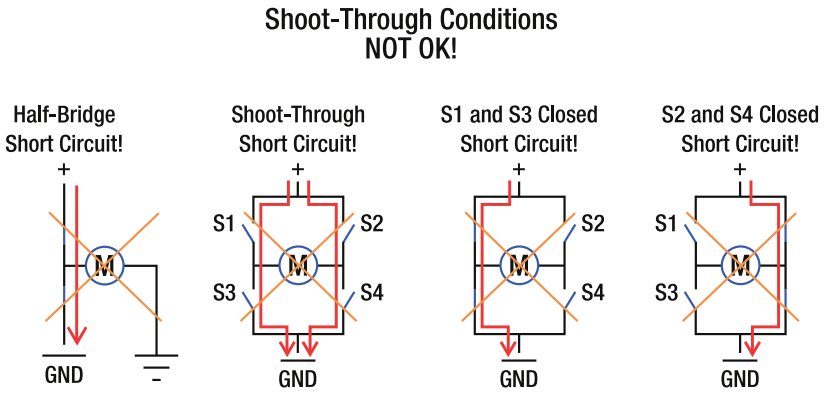

Figure 3-12. *Shoot-through H-bridge states—bad!*

Implementation

To create an H-bridge circuit, we simply need four switches—two of the switches must control the path of the current from the positive supply to each motor terminal, and the other two must control the path of the current from the negative supply to each motor terminal. These switches are labeled as S1, S2, S3, and S4 in the illustrations. We can use any type of switch that we want in the H-bridge, depending on our application. Relay switches work fine for single speed (On/Off) operation, whereas bipolar transistors or mosfets are more appropriate for full-speed control using PWM.

If you are making a smaller H-bridge with BJT transistors, you should include protection diodes from the drain to source of each transistor to protect them from Back EMF. Mosfets have built-in "body diodes" that are capable of handling the same voltages and amperage as the mosfet itself, so these are usually safe to interface directly to the Arduino.

There are four different homemade approaches to building an H-bridge circuit that we discuss, each with their own benefits and drawbacks. We start with the most simple implementation and progress to the most complex.

Method 1: Simple Switches

We can make a full H-bridge using (2) three-way (SPDT) switches from the hardware store, a DC motor, and a 9v battery. This simple bridge has built-in short-circuit protection so it cannot be commanded into a shoot-through state. It can, however, be placed into any acceptable H-bridge state: forward, reverse, electric brake (positive), electric brake (negative), or neutral. Each switch in the circuit has three positions, On/Off/On, and switches the center contact between the two outer contacts (or in this case, the positive and negative battery wires).

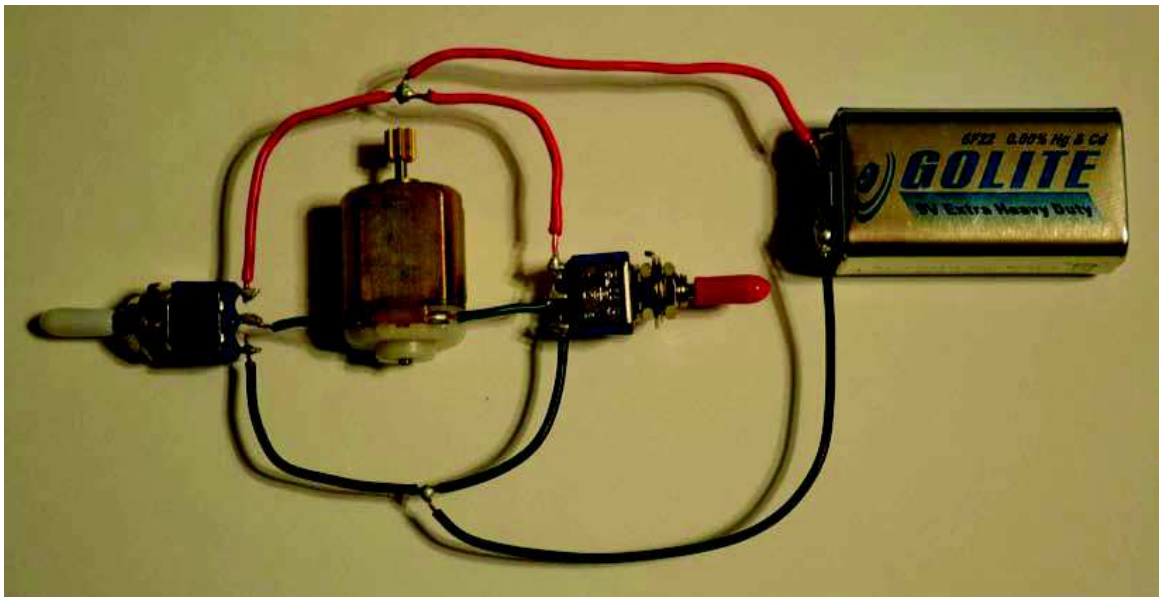

Figure 3-13. *Here you can see a basic H-bridge circuit using two SPDT switches, a DC motor, and a 9v battery. Notice how the top terminals of each switch share a common positive supply wire, while the bottom terminals share a common negative supply wire. The center terminals of each switch are used to route the power signals to the motor terminals.*

This method shows the simplicity of a basic H-bridge circuit, but does not provide speed control (it is either on or off). Although this might be a rugged circuit, its use is limited, so it is usually only good for testing and educational purposes.

Method 2: DPDT Relay with Simple

This method is wired the same as Method 1, but we combine the two SPDT switches and use one DPDT Relay, so it can be controlled by the Arduino. Also we can use the Arduino to provide a simple PWM signal for speed control of the motor (see Figure 3-14). The simplest way to do this is to add a Logic-level N-channel mosfet (or several in parallel) to control the entire circuit's path to ground. By using a PWM signal on the Ground supply to the H-bridge (Relay), we can control the speed of the motor from 0–100%, whereas the relay switches the motor's direction. The relay acts as both the High-side and Low-side switches in the bridge, so there are actually two low-side switches in this configuration—the relay used to route the power terminals and the N-channel mosfet used to provide the PWM speed control.

This provides complete 0–100% speed control and requires as few as four parts other than the relay: (2) logic level N-channel mosfets, (1) diode (for relay coil), and (1) small prototyping PCB (or you can make your own). Depending on the mosfet, you can expect to carry about 10 amperes at 24vdc with no heatsink or fan; an n-channel logic-level mosfet can be found at Digikey.com for between $0.50–$5.00 each and with an amperage rating from 100mA–200amps. I usually select power mosfets with the highest amperage rating in my price range (anything above 75 amps), a higher voltage rating than I plan to use in my project (usually 30v–55v is good), and the lowest possible on-state resistance (check the datasheet for Rds(On)).

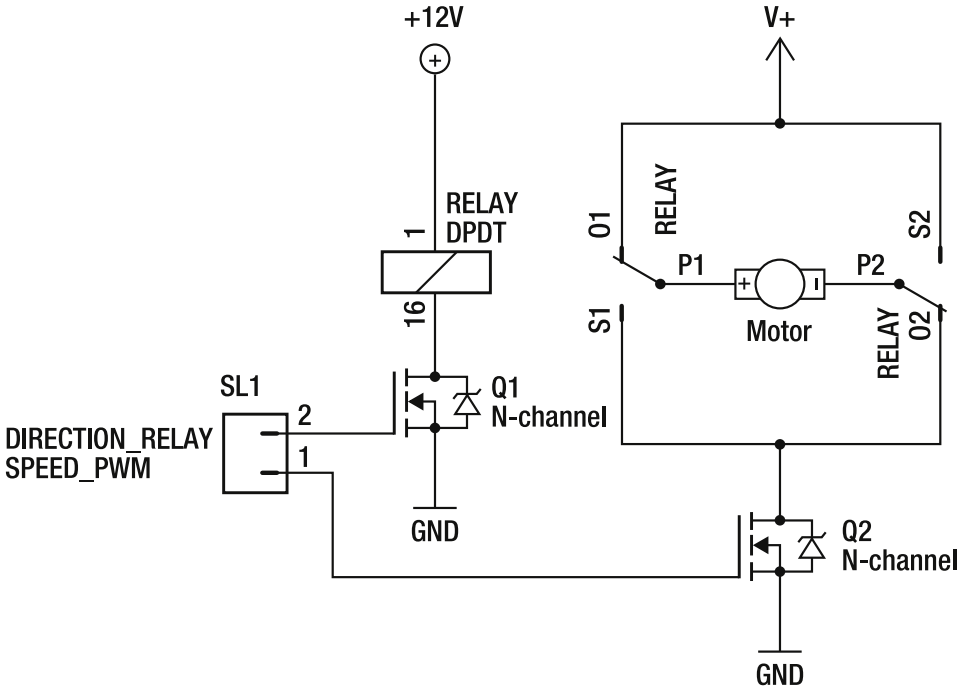

Figure 3-14. *A relay-based PWM speed control circuit*

We can build this circuit with (2) FQP50N06L N-channel mosfets from Digikey. One mosfet is needed to provide PWM speed control, and the other mosfet is needed to interface the relay coil to the Arduino for direction control.

The relay mosfet can be controlled by any Arduino digital output pin, whereas the speed control mosfet should be controlled by an Arduino PWM output. Next we connect the mosfet Drain pin to the Relay as shown in Figure 3-14, and the mosfet Source pin to the main Ground supply. The prototyping PCB makes this easier to put together and you can add screw-terminals for easy wiring. The voltage and current limits of this circuit are dependent on the mosfet and relay ratings, giving this circuit potential despite using a mechanical relay switch.

Method 3: P-Channel and N-Channel Mosfets

Moving up, we have a basic solid-state H-bridge that uses P-channel mosfets for the high-side switches and n-channel mosfets for the low-side switches. This H-bridge has no internal protection against short-circuit, so you must be careful not to open both switches on the same side of the bridge because this will result in a shoot-through condition. This design can easily be implemented on a prototyping PCB as well as adding multiple mosfets in parallel to increase the current capacity. This H-bridge can be built using only two p-channel power mosfets, two n-channel power mosfets, two n-channel signal mosfets, and a few resistors (see Figure 3-15).

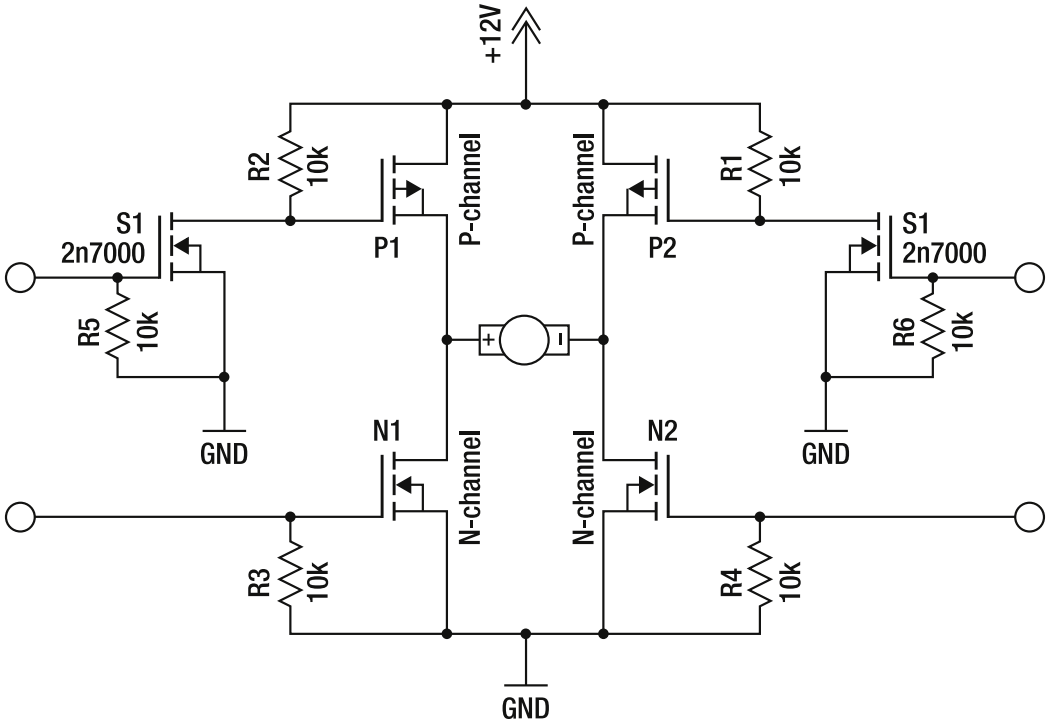

Figure 3-15. *Notice the 10k pull-up resistors on the P-channel mosfets and the 10k pull-down resistors on the N-channel mosfets. This keeps the mosfets in the off state when not in use.*

This method enables for a full solid-state circuit without using any mechanical switches or relays. If this circuit is operated within the voltage and current limits of the mosfets, it will easily outlast either of the previous methods. Even though this bridge is more complex than the previous two, it still has limitations; this design is not optimized for high PWM frequencies or high voltages, but costs little and is easy to build.

Method 4: N-Channel H-Bridge

Most p-channel mosfets have higher Rds(On) values, lower amperage ratings, and higher prices than their n-channel counterparts, making it difficult to design a symmetrical H-bridge.

As you might recall, to turn on an n-channel mosfet (logic-level), the Gate pin must be 5v higher than the Source pin (usually Ground). By connecting an n-channel mosfet backward, we can get it to conduct as a high-side switch. To do this, we connect the mosfet Drain pin to the Positive voltage supply and the Source pin to the motor or load. The only catch is that we must now get the Gate pin to be at least 5v above the Positive supply voltage through boot-strapping.

So how do we make a voltage that is higher than the Positive supply voltage of the batteries? A charge pump is used to collect voltage through a diode and into a capacitor each time the PWM signal is cycled. This is called a *bootstrap circuit* and is effectively a simple voltage doubler used to provide the mosfet Gates with an elevated voltage level. There are several H-bridge driver ICs that include all of the circuitry needed for this operation and require only an external capacitor and diode. We use this type of driver chip to enable the Arduino to control each switch in the H-bridge individually. This type of H-bridge allows for high current capacity and fast PWM switching speeds, which are useful features for a robot motor-controller.

For more information about n-channel H-bridges and circuit diagrams, check out the Open Source Motor Controller (OSMC) project. You can download complete circuits and PCB files, ask questions, or design your own variation and submit your progress to share with the group.

```
http://www.robotpower.com/products/osmc_info.html
```

H-Bridge ICs

To build your own H-bridge, but leave the designing to a professional, you might be interested in an H-bridge IC. An H-bridge IC is a complete H-bridge circuit that is contained on a tiny integrated circuit chip. These are usually fitted into a circuit with very few extra components, typically only a few resistors and a regulated power supply for the logic controls. When using an H-bridge IC, you can usually expect shoot-through protection, thermal overload protection, and high frequency capabilities. Although these H-bridge chips are far less likely to be destroyed by user error than a completely homemade design, they also have much lower power ratings than a homemade H-bridge, typically under 3amps of continuous current.

There are several H-bridge IC chips that include all four switches and a method of controlling them safely. The L293D is a Dual H-bridge IC that can handle up to 36 volts and 600 milliamp per motor. The L298N is a larger version of the L293D that can handle up to 2amps (see Figure 3-16). There are a few ICs that can control up to 25amps, but they are expensive and hard to find. There are several H-bridge ICs that work for some of the smaller projects in this book, but the larger bots require a higher powered H-bridge capable of conducting 10amps or more.

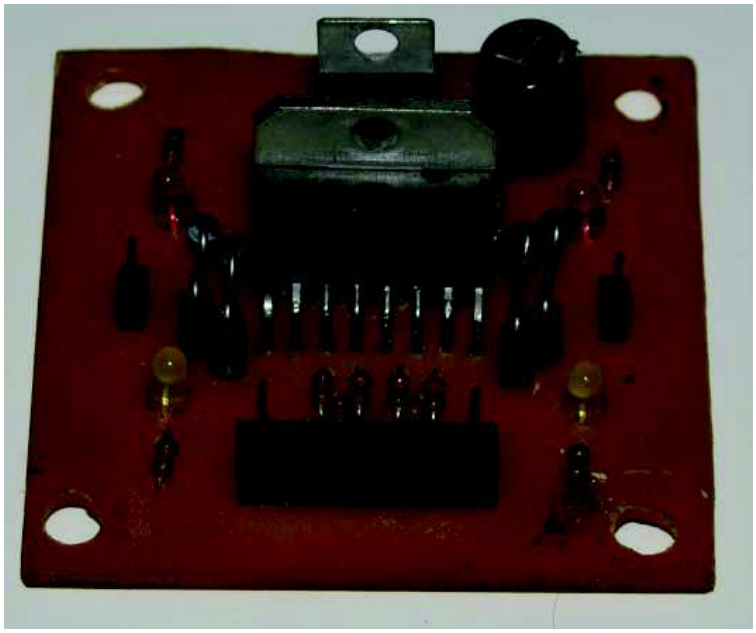

Figure 3-16. *Here is the popular L298N dual 2amp H-bridge motor-controller on a homemade PCB. The other components include a 7805 5v regulator, a few EMF protection diodes, a capacitor, and some direction LEDs. This board can be used to control the speed and direction of two independent DC motors.*

Changing PWM Frequencies

We talked about how higher PWM frequencies eventually lead to switching losses and cross-conduction, so what is a good PWM frequency to use? If you leave your Arduino alone and don't change anything, the PWM outputs will run at 1kHz (pins 5 and 6) and 500Hz (pins 11, 3, 9, and 10). This is considered a relatively low PWM frequency for motor-controllers because at this frequency, there is an audible "whine" that you can hear from the motor coils being switched.

Because most motor-controllers can easily handle a 1kHz PWM signal, you might want to leave the Arduino at its default values. If however, you want your motors to be silent during operation, you must use a PWM frequency that is above the audible human-hearing range, typically around 24,000Hz (24kHz). A problem arises because some motor-controllers are not capable of switching at such high frequency—switching losses increase as the PWM frequency increases. Because it is a difficult design task, motor-controllers that can operate at silent switching speeds (24kHz or higher) are usually more expensive and well built.

The frequency of each PWM output pin on the Arduino is controlled by one of three system timers that are built into the Arduino. Think of each system timer in the Arduino as a digital metronome, that determines how many beats will be in each second. The value of each timer can be changed using one line of code and a specific setting selected from Table 3-1.

To change the frequency of a PWM pin, select an available frequency from Table 3-1 and replace the <setting> in the following code with the appropriate setting from the chart. Then add the following line of code into the setup() function of your sketch, depending on the timer you want to change:

```
TCCR0B = TCCR0B & 0b11111000 | <setting>; //Timer 0 (PWM pins 5 & 6)
TCCR1B = TCCR1B & 0b11111000 | <setting>; //Timer 1 (PWM pins 9 & 10)
TCCR2B = TCCR2B & 0b11111000 | <setting>; //Timer 2 (PWM pins 3 & 11)
```

***Table 3-1**. Available PWM Frequency Settings for Each Arduino System Timer*

Arduino Timer	<setting>	Divisor	Frequency (Hertz)
0 (pins 5 and 6)	0x01	1	62500
0 (pins 5 and 6)	0x02	8	7812.5
0 (pins 5 and 6)	0x03	64	976.56
0 (pins 5 and 6)	0x04	256	244.14
0 (pins 5 and 6)	0x05	1024	61.04
1 (pins 9 and 10)	0x01	1	31250
1 (pins 9 and 10)	0x02	8	3906.25
1 (pins 9 and 10)	0x03	64	488.28
1 (pins 9 and 10)	0x04	256	122.07
1 (pins 9 and 10)	0x05	1024	30.52
2 (pins 3 and 11)	0x01	1	31250
2 (pins 3 and 11)	0x02	8	3906.25
2 (pins 3 and 11)	0x03	32	976.56
2 (pins 3 and 11)	0x04	64	488.28
2 (pins 3 and 11)	0x05	128	244.14
2 (pins 3 and 11)	0x06	256	122.07
2 (pins 3 and 11)	0x07	1024	30.52

This tableTable 3-1 shows the available frequencies with their corresponding settings—you might notice that some frequencies are available only on certain timers, making each PWM pin unique. For example, to change the frequency on PWM pins 9 and 10 from the default 500Hz to an ultra-sonic

switching speed of 32kHz, change the setting for system timer 1 in the setup() function, as shown in the following:

```
void setup(){
    TCCR1B = TCCR1B & 0b11111000 | 0x01;
}
```

By changing timer 1 to a setting of "0x01", PWM pins 9 and 10 will now operate at 32kHz frequency anytime the analogWrite() command is used on either pin. Alternatively, you can set these same PWM pins to operate at their lowest available frequency (30Hz) by changing the <setting> to "0x05".

If you operate the PWM output at a too low of a frequency (below 100Hz), it will significantly decrease the resolution of the control—that is, a small change in the input will cause a drastic change in output and changes will appear choppy and not smooth as they do at higher frequencies. If in doubt, simply stay with the Arduino default PWM frequencies because they are sufficient for most robotics projects, even if you can hear your motors.

■ **Note** Changing the Arduino system timer 0 affects the output of certain Arduino timing functions that rely on timer 0, such as the delay(), millis(), and micros() functions.

Back EMF

Back Electro-Motive Force (Back EMF) is the term used to describe the energy that must be disposed of when the electro-magnetic field of an inductor collapses. This collapse happens each time the motor is stopped or changes directions. If the voltage cannot escape through a rectifying diode, it can damage an unprotected transistor and possibly damage the Arduino pin that is driving it. A simple rectifier diode (1n4001) works for most relay coils and small BJT transistor-based H-bridges up to 1amp.

A protection diode should be placed between the motor terminal and the power supply. If using an H-bridge, a diode must be placed between each motor terminal and both the positive and negative power supply for a total of four diodes (see Figure 3-17). If you have an H-bridge that has no protection diodes, you can add the diodes directly onto the motor terminals as shown in Figure 3-18.

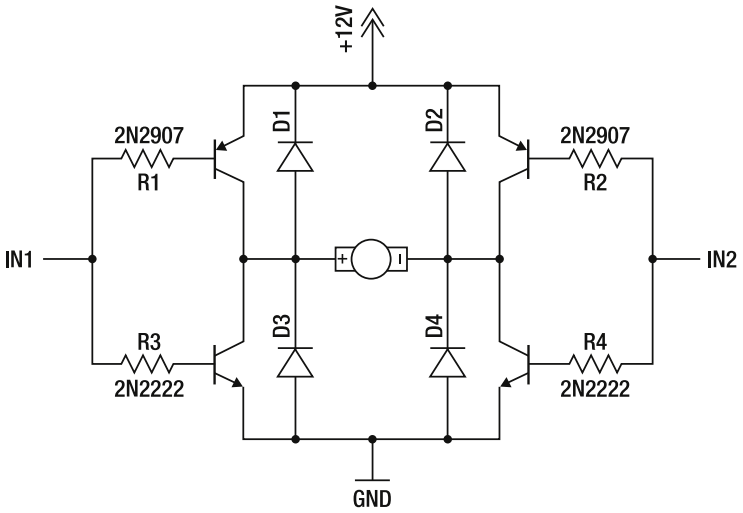

Figure 3-17. *Protection diodes should be placed around the switches to protect them from motor back EMF, D1–D4 in the image.*

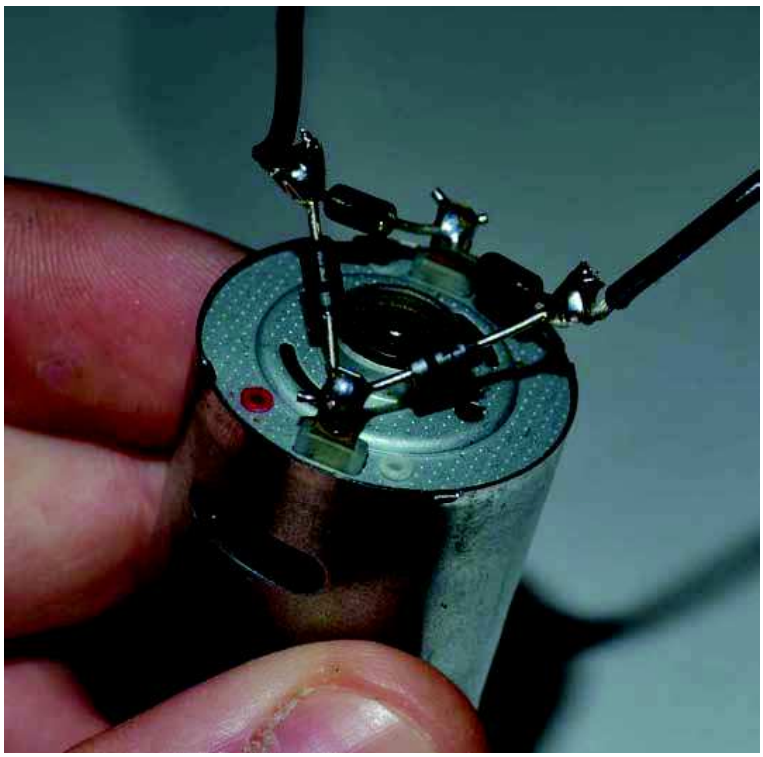

Figure 3-18. *Notice this implementation of Back-EMF protection diodes, soldered directly onto the motor terminals—this alleviates the need for diodes built in to the motor-controller.*

Current Sensing

Sometimes, the best way to protect a homemade H-bridge is to install a current-sensing device to monitor the level of amperage that is passing through the H-bridge. By reading the output of a current-sensor with the Arduino, we can send a stop command to each motor if the current level exceeds a given point. The over-current protection feature uses current sensing to disable the driver if the power reaches an unsafe level to protect it from overheating. Using this feature nearly eliminates user errors that can result in a destroyed motor-controller.

The simplest way to measure the amperage level in an H-bridge is to measure the voltage drop across a power resistor. This resistor must be placed in series with motor and the positive voltage supply, and the motor must be powered and running while you are measuring the voltage across the resistor (see Figure 3-19). Knowing the exact value of the resistor in ohms and the measured voltage across the resistor, we can use Ohm's law to calculate the amperage that is passing through the resistor, and therefore the circuit.

The only problem with this method is that the resistor creates heat in the process (wasting electricity). For this reason, it is ideal to pick the lowest value resistor possible (0.01–1 ohm) and it must have a power rating that is sufficient for the amount of current you will pass through it.

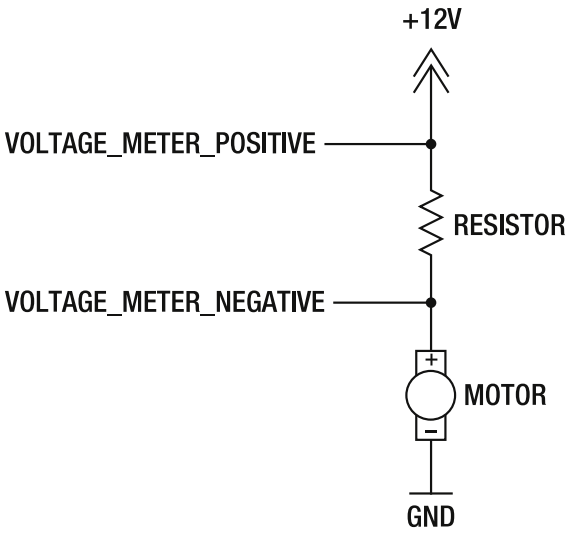

Figure 3-19. *By measuring the voltage across a current-sensing resistor, we can calculate the amount of current that a motor is using.*

For example, if the voltage drop across a current sensing resistor is 0.5 volts, and the resistor value is 0.05 ohms, how much current is passed through the resistor, and what will the power rating need to be for the resistor?

First we measure the current through the resistor:

V = I * R

0.5v = I * 0.05 ohms

I = 0.5v / 0.05 ohms

I = 10 amps

If you measure 0.5 volts across a 0.05-ohm current sensing resistor, the amount of current that is passing through the resistor is 10amps.

Now to calculate the power dissipation of the resistor:

$$W = I^2 * R$$

$$W = (10amps * 10amps) * 0.05 ohms$$

$$W = 5 watts$$

As you can see, the resistor must be rated for 5 watts to be able to handle 10 amperes flowing through it without failing.

There are better options available for current sensing in an H-bridge, like the hall-effect based ACS-714 current sensor that can accurately measure up to 30amps in either direction, and outputs a proportional analog output voltage that can be read using the Arduino. This sensor is mentioned in Chapter 2 as a non-autonomous sensor. It is available on a breakout board for use with an existing motor-controller or as an IC that can be soldered directly into a motor-controller design (like the ones used on the Explorer bot in Chapter 8). With this motor feedback mechanism, we can use the Arduino to monitor the motor output current and create a custom over-current protection method to keep the motor-controller from overheating.

Commercial H-Bridges (Motor-Controllers)

If you do not plan to build your own motor-controller, you will still need to decide which one to buy. It is important to select a motor-controller with a voltage limit that is at least a few volts above your desired operating voltage, because a fully charged battery is usually a few volts higher than its rating. This is important because if the maximum voltage limit is exceeded even for a few seconds, it can destroy the mosfets, which will result in a broken H-bridge. The amperage rating is a bit more forgiving, in that if it is exceeded the H-bridge will simply heat up. Remember that using a heat sink or cooling fan can increase the maximum amperage limit by removing the dangerous heat, so many commercial units have heatsinks or fans built-in to aid in heat dissipation.

A commercial H-bridge can range in price from $10–$500+, but we will assume that you are not made of money, and focus mainly on the budget motor controllers. Most units accept PWM, Serial, or R/C pulse signals and some have the capability to read several different signal types using on-board jumpers to select between modes. You can find units that handle anywhere from 1amp to 150amps of continuous current and have voltage ratings from 6VDC to 80VDC.

Small (Up to 3amps)

This size H-bridge powers small hobby motors, typically not larger than a prescription medicine bottle. There are many different H-bridges available from online retailers that can handle several amps and range in price from $10–$30 (see Table 3-2). The L293D and L298N are two common H-bridge ICs that many small commercial motor-controllers are based on. The Sparkfun Ardumoto is an Arduino compatible dual motor-controller shield that is based on a surface mount version of the L298N H-bridge IC shown in Figure 3-16, and is capable of handling up to 2amps per channel (see Figure 3-20).