5

THE GARAGE SENTRY PARKING ASSISTANT

This project is a reliable electronic device to gauge the distance you need to pull your car into your garage. If you park in a garage, you're probably familiar with the problem: how far do you pull your car into the garage to make sure there's room in front for whatever is there and enough space behind so the garage door will close? Some people suspend a tennis ball on a string from the ceiling and stop at the point when the ball meets the windshield. That works fine, but the ball is a pain to set up and adjust, and it often gets in the way if you want to use the garage for something other than parking the car.

Arduino offers a better solution. This Garage Sentry project is the electronic version of the classic tennis-ball-on-a-string device, only better. The Garage Sentry accurately detects when your car reaches exactly the right position in the garage and sets off an alarm that blinks so you know when to hit the brakes.

In addition, at the end of the chapter, I'll show you how to modify the basic Garage Sentry into a deluxe version that alerts you when you're getting close to the perfect stopping point.

INSPIRATION BEHIND THE GARAGE SENTRY

This project evolved out of playing with an *ultrasonic transceiver module*, a device that emits sound waves and then detects them after they travel to an object, reflect off that object, and travel back to the module. The output of the module allows a microcontroller to measure the time it takes to travel to and from the object and, knowing the speed of sound, determine the distance. To test the ultrasonic transceiver's sensitivity and limits, I used the battery-operated breadboard version in my garage, which had enough space to move objects around for different distances. It turns out cars are great reflectors for ultrasonic energy. From this experimentation, I was inspired to turn my test apparatus into a Garage Sentry.

Required Tools

This project doesn't require many tools or materials, but you will need the following tools for both the standard and deluxe versions:

- A drill with a 3/8-inch or 1/2-inch chuck (powered by battery or with 110/220V from the wall)
- Drill bits for potentiometer (9/32 inches), power input (1/4 inches), and LED (3/8 inches)
- Soldering iron and solder
- Tapered reamer
- Screwdrivers (See the Introduction for screwdrivers you should have on hand.)
- Pliers (I recommend needle nose.)
- 28- or 30-gauge hookup wire
- (Optional) Wire-wrap tool and wire
- (Optional) 1/4-inch tap

Parts List

You'll need the following parts to build the basic Garage Sentry:

- One Arduino Nano
- One HC-SR04 ultrasonic sensor (Try eBay, Adafruit, Sparkfun, and so on.)

- Two high-intensity LEDs (>12,000 MCD; available from eBay and other online stores)
- Two 1/4 W (or more), 270-ohm resistors (to limit current to the LEDs)
- One 1/8 W, 20-ohm potentiometer
- Two NPN-signal transistors rated for a collector current of at least 1.5 A (I used ZTX-649 transistors, which you can find at Mouser, Digikey, and Newark.)
- One enclosure (I recommend a blue Hammond 1591 ATBU, clear 1591 ATCL, or something similar.)
- (Optional) One 0.80-inch aluminum strip for mounting bracket
- (Optional) Two 1/4-inch × 20-inch × 3/4-inch bolts with nuts
- One section (approximately 1 inch × 1 inch) perforated board (can include copper-foil rings on one side)
- One 3.5 mm jack for power
- Two 2-56 × 3/8-inch screws and nuts
- Two additional 2-56 nuts to use as spacer
- One 9V, 100 mA plug-in wall adapter power supply (Anything from 7.5V to 12V DC at 100 mA or upward should work well.)
- One length double-sided foam tape
- One LM78L05 (TO-92 package) regulator (for the breadboard build only)

Because the basic version doesn't require a lot of additional components, I suggest building the circuit on a standard perforated circuit board instead of a shield. To power your circuit, you can use a 9V, 100 mA wall adapter plugged into a 3.5 mm jack (see Figure 5-1). You shouldn't need an on/off switch.

Figure 5-1: I used a Magnavox AC adapter, but any similar power supply with a DC output from 7.5V to 12V should work. These are readily available online and cost from under $1.00 to about $3.00.

Be sure to use two bright LEDs that are clearly visible, even when a car's headlights are on. Bright LEDs range from 10,000 MCD (millicandela) to more than 200,000 MCD. The brighter, the better; just remember that brighter LEDs require more power, so the current-limiting resistor will need a higher power rating for the brighter lamps. The 270 W current-limiting resistors result in a current drain of about 30–40 mA each with the 12,000 MCD LEDs I used at 5V. (Remember that power equals volts times amps, or $P = VI$, so at 40 mA and 5V, you'd have 0.20 W.) It's best to use a 1/2 W or greater resistor even though you can easily get by with a smaller value—as I did with 1/4 W—because the LEDs are on only intermittently.

Optional Parts

In addition to the components for the basic Garage Sentry, you'll need the following extra components if you want to build the deluxe version:

- Two high-intensity green LEDs
- Two high-intensity amber LEDs
- Two additional 270-ohm resistors (1/4 W)
- Two additional transistors (ZTX-649)
- One enclosure Hammond 1591 BTCL (to replace the 1591 ATCL)
- One PCB (shield)

Downloads

- Sketches: *GarageSentry.ino* and *GarageSentryDel.ino*
- Drilling template: *Transducer.pdf*
- Drawing: *Handle.pdf*
- Shield file for Deluxe Garage Sentry: *GarageSentreDel.pcb*

Basics of Calculating Distance

This project measures the time it takes for a sound to originate, bounce off an object, and be received back at the point of origin, and it uses that time to calculate the distance between the object and the sensor.

The basic distance calculation is not much different from determining the distance of a storm by counting the seconds between a lightning flash and a thunderclap. Each second represents a distance of 1,125 feet, or about 0.2 miles. Given that sound travels at 1,125 feet per second in air at sea level, if there's a five-second delay between a lightning flash and the thunderclap, you can determine that the storm is roughly a mile away. In the case of the Garage Sentry, once you know how long it takes for the sound to make a round trip and know the speed of sound, you can calculate the distance according to the time-speed-distance formula:

$$\textit{Distance} = \textit{Speed} \times \textit{Time}$$

How the Garage Sentry Works

This project takes advantage of *ultrasonic sound*, which, unlike thunder, is above the hearing range of most individuals. If your hearing is good, you can detect sound ranging from about 30 Hz to close to 20 kHz, although hearing attenuates quickly above 10 kHz or 15 kHz.

NOTE *For reference, middle C on the piano is 261.6 Hz. Young children (and most dogs) can often hear high frequencies, but hearing, especially in the upper registers, deteriorates quickly with age.*

The ultrasonic transceiver module used in this project sends out pulses at a frequency of about 25 kHz and listens for an echo with a microphone. If there is something for the signal to bounce off, the system receives the return echo and tells the microcontroller a signal has been received and to calculate the distance. For the Garage Sentry, the unit is placed in the front of the garage, and the signal is sent out to bounce off the front—or rear if you are backing in—of your vehicle. To calculate your car's distance from the ultrasonic transceiver, the Arduino measures the time it takes for the signal's round trip from the transceiver to the target and back. For example, if the Arduino measures a time of 10 milliseconds (0.010 seconds), you might calculate the distance as:

$$\textit{Distance} = 1125\frac{\text{ft}}{\text{s}} \times 0.010\ \text{s} = 11.25\ \text{ft}$$

Ah, but not so fast. Remember the signal is traveling to the car and then back to the microphone. To get the correct distance to the vehicle, we will have to divide by two. If the controller measures 10 ms, then the distance to your car would be:

$$\textit{Distance} = \frac{1{,}125\frac{\text{ft}}{\text{s}} \times 0.010\,\text{s}}{2} = 5.625\ \textit{ft}$$

The HC-SR04 ultrasonic module sends out a signal at the instruction of the Arduino (see Figure 5-2). Then, the sketch instructs the transmitter to shut down, and the microphone listens for an echo.

If there is an object for the signal to bounce off, the microphone picks up the reflected signal. The Arduino marks the exact time the signal is sent out and the time it is received and then calculates the delay.

The HC-SR04 module is more than a speaker and microphone, though. The module includes transducers—a loudspeaker and mic—and a lot of electronics, including at least three integrated circuits, a crystal, and several passive components. These components simplify its interface to the Arduino: the 25 kHz tone is actually generated by the module and turned on and off with the microcontroller. Some of the components also enhance the receiver's, or the microphone's, sensitivity, which gives it a better range.

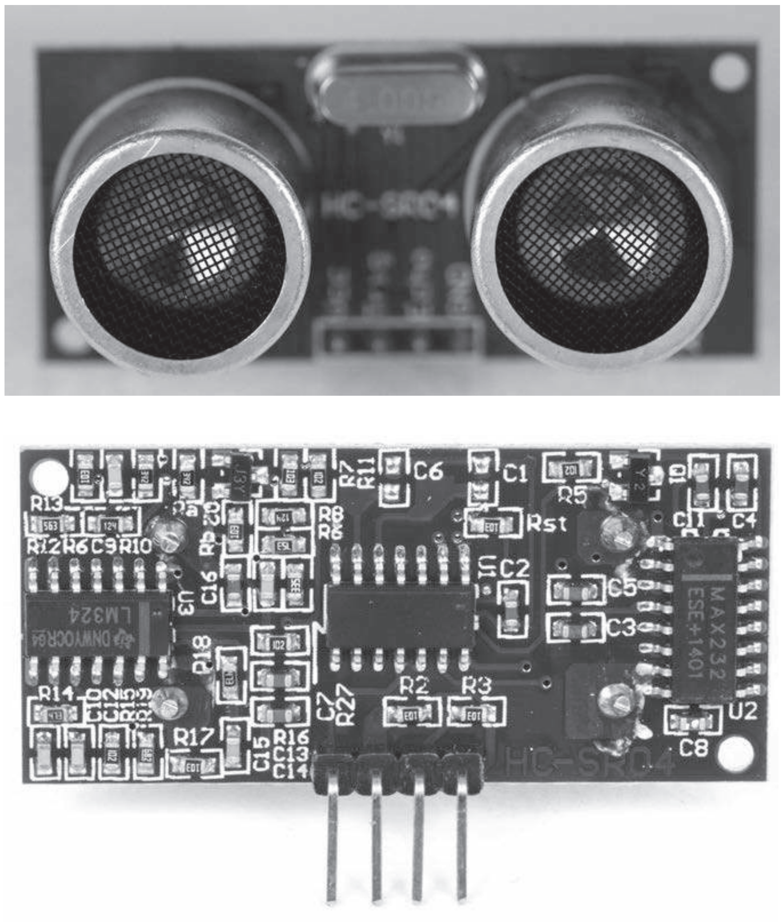

Figure 5-2: The ultrasonic sensor module. The back of the module (bottom) has connection terminals at the bottom.

The range of the HC-SR04 ultrasonic transducer is approximately 10 to 12 feet. The returning signal is always a lot weaker than the transmitted signal because some of the sound wave's energy dissipates in the air (see the dotted lines in Figure 5-3).

The arithmetic to calculate the distance between the sender and the object is not difficult. You take the number of microseconds it takes for the signal to return, divide by the 73.746 microseconds it takes sound to travel an inch, and then divide by two because the signal is going out and coming back. The full arithmetic for this appears later in "Determining Distance" on page 143.

The sketch provides a response in inches or centimeters depending on your preference. We'll use inches for setting up the distance for the alarm, but converting to centimeters simply requires a remapping of the analog input and setting the numbers a bit differently. The sketch also does the basic arithmetic for determining the centimeter measurement for you.

With the high-level overview out of the way, let's dig in to how you'll wire the Garage Sentry.

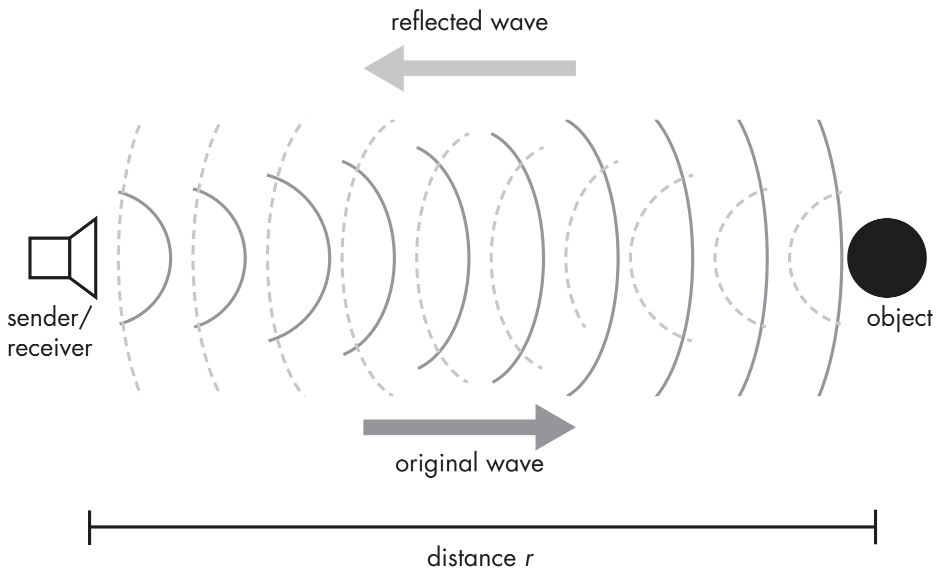

Figure 5-3: In this project, sound is transmitted from a sender, bounces off an object, and is received.

The Schematic

Figure 5-4 shows the schematic for the Garage Sentry.

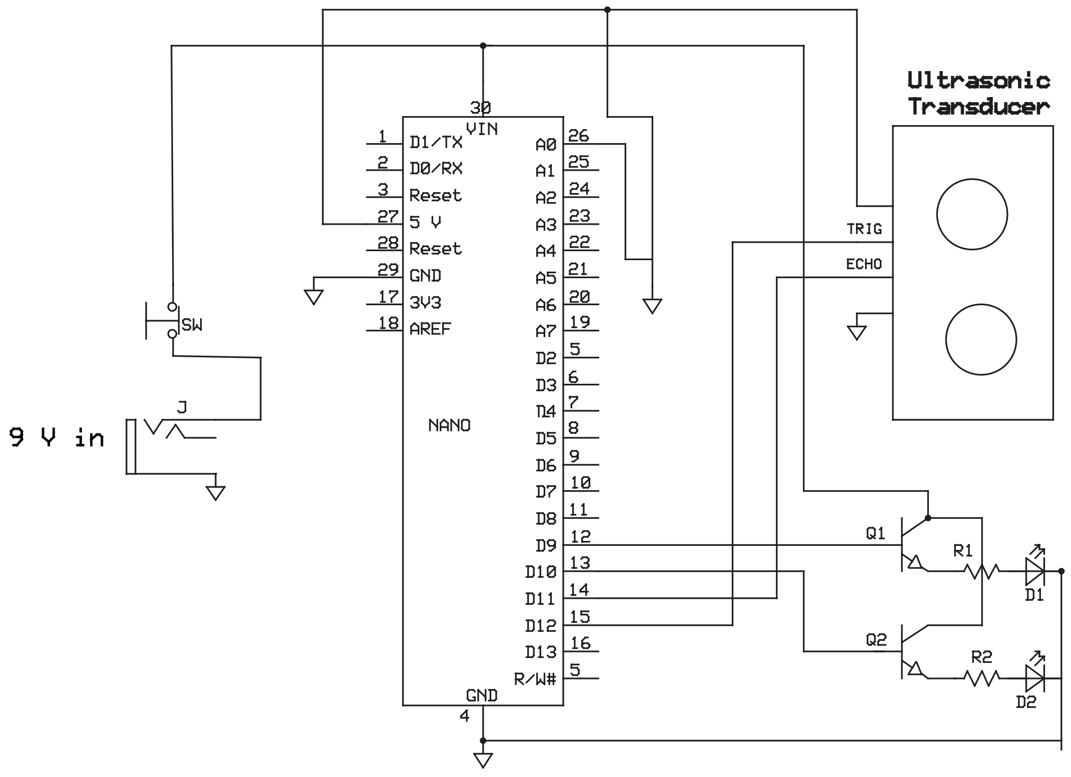

Figure 5-4: Schematic diagram of the Garage Sentry

R1 and R2 are the 270 W resistors for the LEDs and should be 1/4 W or larger. If a higher wattage resistor is not available, you could place several resistors in parallel to gain the required wattage. First, find the right resistor value with the formula:

$$\frac{1}{R_{\text{total}}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \ldots + \frac{1}{R_n}$$

You can also use an automatic calculator, such as the one at *http://www.1728.org/resistrs.htm*, which is a lot easier than doing the math yourself.

To avoid extra calculations, select resistors of the same value. This way, the same amount of current flows through each one. For example, two 1/8 W resistors in parallel will give you a 1/4 W value.

If you do use resistors of different values, you will have to calculate the current flowing through each and the total dissipation.

This schematic also leaves you with room to customize your alarm. While this version of the project uses LEDs to create a visual alarm, with a slight modification, you can easily create an audible alarm as well. Simply replace either the red or blue LED with an audible device, such as a Sonotone Sonalert, and the alarm will sound. To replace an LED, you would need to connect the Sonalert across that LED's connections; just make sure to get the polarity correct. Alternatively, you could keep both LEDs and add an audible device for a third warning.

NOTE *In this project, the Nano takes advantage of its on-board voltage regulator, which is why there's no external regulator in the schematic.*

The Breadboard

The entire Garage Sentry fits on a small breadboard, so you can set it up, program it, power it with a battery, and walk around to test it out. As you play with it, I'm sure other applications of ultrasonic technology will come to mind. The breadboard I assembled appears in Figure 5-5.

In Figure 5-5, the breadboard is powered by a 9V battery. Usually, you could wire the battery directly to the VIN of the Nano and use the Nano's built-in voltage regulator. But you'll power the Nano with a USB cable when you program and test it for the first time, so on the breadboard, you'll set up the positive and negative rails for 5V for both the Nano and the ultrasonic module. To avoid risking damage to the Nano or the module and avoid overcomplicating the build, I included a single-chip external voltage regulator (LM78L05) so the entire breadboard runs on 5V. Take a look at Figure 5-6 to see how it's wired up.

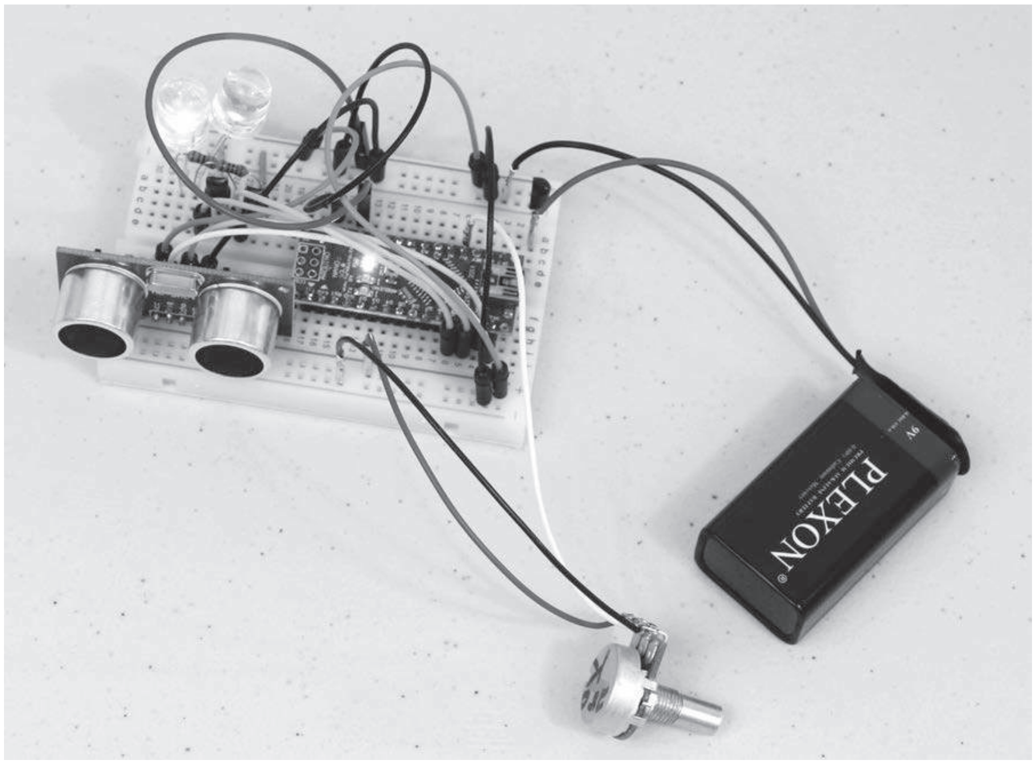
Figure 5-5: Here's the breadboard wired up. I used a 9V battery so I could experiment in different environments. Both LEDs look illuminated because of the length of the exposure of the camera.

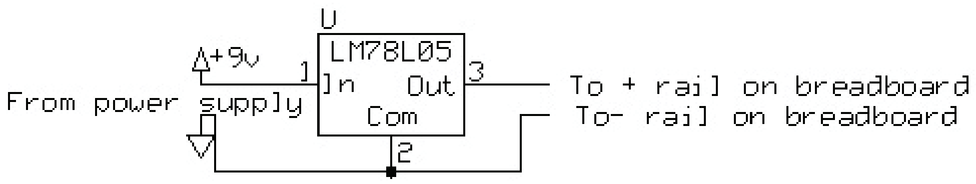

Figure 5-6: This is how the LM78L05 TO-92 regulator is wired up on the breadboard. Bypass/filter capacitors are not required.

Here's a blow-by-blow list of the steps to wire the breadboard:

1. First, put the ultrasonic module at the lower end of the breadboard facing out, and plug the Nano in to the breadboard, leaving four rows of connections above it.
2. Make sure the positive and negative (red and blue) strips on the left and right are connected properly—red to red, blue to blue. If you connect red to blue, it will cause a major problem.
3. Connect the red strip to the 5V power supply (pin 27 of the Nano, labeled *5V*). This is necessary if you are operating from the USB connector.

4. Connect pin 4 of the Nano (labeled *GND*) to the breadboard's negative rail (blue strip).
5. Connect VCC of the HC-SR04 transducer to the positive rail.
6. Connect GND of the HC-SR04 transducer to the negative rail.
7. Connect TRIG of the HC-SR04 transducer to pin 15 (D12) of the Nano.
8. Connect ECHO of the HC-SR04 transducer to pin 14 (D11) of the Nano.
9. Insert two ZTX-649 transistors into the breadboard. Select an area where all three pins of each transistor can have their own row.
10. Connect pin 12 (D9) of the Nano to the base of transistor Q1.
11. Connect pin 13 (D10) of the Nano to the base of transistor Q2.
12. Connect the collectors of both transistors to the positive rail (red strip).
13. Connect the emitter of transistor Q1 to one end of a 270-ohm resistor.
14. Connect the other end of the 270-ohm resistor connected to the emitter of transistor Q1 to a blank row on the breadboard.
15. Connect the emitter of transistor Q2 to one end of another 270-ohm resistor. Connect the other end of the 270-ohm resistor connected to the emitter of transistor Q2 to another blank row on the breadboard.
16. Connect the + (long end) of LED (D1) to the 270-ohm resistor and the other end to ground (blue strip).
17. Connect the + (long end) of LED (D2) to the second 270-ohm resistor and the other end to ground (blue strip).
18. Connect one end of the 20 kW potentiometer to the positive rail (red stripe).
19. Connect the opposite end of the potentiometer to the negative rail (blue stripe).
20. Connect the wiper (center) of the potentiometer to analog pin A0 (26) of the Nano.

You should be good to go! If you use the AC connection, simply connect it to the VCC connection of the Nano.

To add a battery connection, include the 78L05 with its center pin to ground (negative rail), the input to the positive side of the battery, and the output to the positive rail (see Figure 5-7). Connect the negative terminal of the battery to the negative rail.

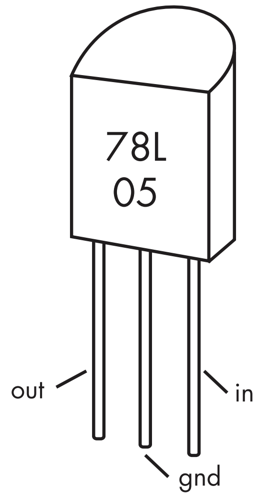

Figure 5-7: Pinout of the 78L05 voltage regulator

The Sketch

Once the breadboard is complete, the sketch can be loaded onto the Nano. Download the *GarageSentry.ino* file from *http://www.nostarch.com/arduinoplayground/*. To load the file onto the Nano, follow the instructions outlined in Chapter 0 on page XX. Remember to select the correct board type. Once it's loaded, the unit is ready for experimentation.

The sketch for the Garage Sentry serves several functions. It tells the ultrasonic sensor to generate a wave and detects how long it takes the echo to return. It then calculates the distance based on that time and, if necessary, alerts you to stop by turning on the LEDs. Here's the sketch in full; I'll walk you through it next.

```
/* Garage Sentry 3b
*/

int ledPin = 10;
int ledPin1 = 9;
int count;
int analogPin = A0;
int val;
int y;

void setup() {

  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(analogPin, INPUT);
}
void loop() {
  val = analogRead (analogPin);
  long duration, inches, cm;
  //Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
❶ pinMode(12, OUTPUT);  //Attach pin 12 to Trig
  digitalWrite(12, LOW);
  delayMicroseconds(2);
  digitalWrite(12, HIGH);
  delayMicroseconds(5);
  digitalWrite(12, LOW);

  pinMode(11, INPUT);  //Pin 11 to receive Echo
  duration = pulseIn(11, HIGH);

  //Convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  val = map (val, 0, 1023, 0, 100);
  if(inches == 0)
    digitalWrite(ledPin, LOW);

  if(count == 0 && inches > 0 && inches < val) {
❷   for(y = 0; y < 200; y++)
    {
      digitalWrite(ledPin, HIGH);
      digitalWrite(ledPin1, LOW);
      delay(100);
      digitalWrite(ledPin, LOW);
      digitalWrite(ledPin1, HIGH);
      delay(100);
    }
    count = count + 1;
  }

  digitalWrite(ledPin1, LOW);
  if(inches > 10) {
    //delay(1000);
    count = 0;
  }
  Serial.print(inches);
  Serial.print("   inches ");
  Serial.print(count);
  Serial.print(" count   ");
  Serial.println();
  Serial.print(" Val       ");
  Serial.println (val);
  delay(100);
}
long microsecondsToInches(long microseconds)
{
  return microseconds / 74 / 2;
}
long microsecondsToCentimeters(long microseconds)
{
  return microseconds / 29 / 2;
```

First, we define several variables, establish parameters, and load libraries (if any). In this case, define `LED1pin` and `LED2pin`, which will serve as the alarm. Other definitions (`int`) include `cm` and `count` (a variable that will be used internally), `analogPin` (as A0), `val` (to hold the limit information), and `y` (used in the loop).

Inside the setup() Function

Next is the `setup()` function. Here, you set up Arduino features that you might want to use; this sketch includes the serial monitor, which you probably will not need in the final product but is often useful in debugging code, particularly if you want to change the code. This sketch sets the rate of the monitor at 9600 baud, which is standard in many applications. It also defines the mode of the pins you'll use as either input or output. You could set the `pinMode` values at almost any point in the code, including before or inside the setup; they're also often defined within the main loop, particularly if the definitions are expected to change.

Inside the loop() Function

The `loop()` function is where everything really happens. The loop continually executes unless it's delayed or halted by a command. So even when it appears that nothing is happening, the controller is continually cycling through the code. In this application, one of the first tasks the controller performs in the loop is to set the variable `val` to store the input from the potentiometer connected to the analog pin (`analogPin`).

In order to initiate the ultrasonic module's transmit/receive function, the sketch first calls for a low signal to be sent to the transmitter (`Trig`) to purge the module to assure that the following high signal will be clean. You can see this in the lines starting at ❶.

Next, there's a delay to let things settle before the sketch writes a high to the transmit pin, which orders the transmitter to transmit an ultrasonic signal. This is followed by another delay, and then the sketch drives digital pin 12 low to turn off the transmitter and activates the receiver by calling the `pulseIn()` method.

Determining Distance

If there's no echo—that is, if `inches == 0` or inches approaches infinity—the controller continues to run the code until it reaches the end and then starts again at the beginning. If it detects an echo, the number of microseconds between turning the transmitter on and receiving signal (`duration`) is then converted to both inches and centimeters. This gives us a measurement of how far the transceiver is from the object. Note that throughout this explanation, I will refer to inches, but you could follow along in centimeters, too.

The `microsecondsToInches` and `microsecondsToCentimeters` commands convert the time measurement to inches and centimeters, respectively, according to the arithmetic discussed in "How the Garage Sentry Works" on page 135. The data type `long` is used, as opposed to `int`, because it provides 4 bytes of data storage instead of just 2, and the number of microseconds could exceed the 2-byte limit of 32,767 bits. So far, so good.

In a regular formula, the distance arithmetic looks like this:

$$\text{inches} = \frac{\text{time}}{2} \div \frac{74\,\mu\text{s}}{\text{inch}}$$

$$\text{centimeters} = \frac{\text{time}}{2} \div \frac{29\,\mu\text{s}}{\text{centimeters}}$$

In either case, we first divide by 2 because the signal travels from the transducer to the target and back, as previously discussed. In the inches function, we then divide the halved number of microseconds by 74, and in the centimeters function, we divide by 29. (It takes 74 µs for the signal to travel 1 inch, and 29 µs for it to travel 1 cm; I arrived at those numbers by following the arithmetic in "Time-to-Distance Conversion Factors" on page 144.)

TIME-TO-DISTANCE CONVERSION FACTORS

You could simply trust my math and copy the time-to-distance conversion code, but you can apply this arithmetic to any project using a similar ultrasonic module or other sensor, so I encourage you to work through the math yourself.

As I describe in "How the Garage Sentry Works" on page 135, the speed of sound is roughly 1,125 feet per second. Multiply that by 12 inches per foot to get 13,500 inches per second.

To get the number of seconds per inch, you simply divide this value by 13,500 inches:

$$\frac{1 \text{ second}}{13,500 \text{ inches}} = 0.000074 \frac{\text{seconds}}{\text{inch}}$$

It takes about 74 microseconds, or 0.000074 seconds, for sound to travel an inch. To determine the distance in centimeters, go through the same exercise, but use 343 meters per second for the speed of sound, multiply it by 100 centimeters per meter, and take the reciprocal.

Triggering the Alarm

The sketch is not done yet. Now we have to look at the number of inches (or centimeters) measured and compare it to the predetermined value—`val`, in this case—to see whether the alarm should be activated. To establish the variable `val` as a numeric value, take a potentiometer (R2) straddling the power supply on either end and tie the wiper to pin A0 (see Figure 5-4). Because A0 is the input to a 10-bit analog-to-digital converter, it converts that voltage (between 0V and 5V) to a numeric digital value between 0 and 1,023. Reading that value with an `analogRead` command results in a value between 0 and 1,023 depending on the position of the potentiometer.

That value is then used to establish the trigger point for the alarm. But allowing all 1,024 values would essentially allow the distance to be set from 0 to 1,023 inches. Because the control rotates only 270 degrees, to adjust between, say, 40 and 42 inches would represent a very minuscule rotation—beyond the granularity of most potentiometers.

To scale this for the potentiometer, the sketch maps the value so the entire rotation of the potentiometer represents a distance of only about 100 inches with the following line of code:

```
val = map (val, 0, 1023, 0, 100);
```

Mapping the potentiometer value changes the maximum distance from 1,023 inches down to 100 inches while leaving the minimum distance of 0 inches unchanged. You can map any set of values so the Garage Sentry's target distance can be from *X* to *Y*, with full rotation of the potentiometer, so when you set up your Garage Sentry, you may want to test it and this range until it's right for your garage.

A conditional control structure sets the limit for the alarm. This structure makes sure that the LED is turned off when the measured distance is 0, regardless of whether the sketch is using inches or centimeters. First, the value `inches` is compared to `val` in the following expression:

```
count == 0 && inches > 0 && inches < val
```

If this statement is true, the alarm is set off and the `for` loop at ❷ is activated (see page 141), which alternately blinks the LEDs 200 times before timing out and turning the LEDs off.

The `for` loop just counts from 0 to 200, but that can be easily changed. After each count, it turns on an LED, delays briefly and turns off the same LED, delays slightly and turns on a second LED, delays slightly and turns off the LED, and then goes to the next count. At the end of the 200 count, the system turns off the LEDs and the program continues to the next line where it is reset. That is, the program starts again at the beginning.

Construction

The trickiest part of the Garage Sentry is mounting the ultrasonic module on the box. Because the module can send out sound waves only in a straight line, you need to be able to adjust its direction so that the ultrasonic sensor can hit its target and receive the echo. But the module includes only two mounting holes, diagonally opposed from each other, so there's no easy way to fasten it to a flexible mounting. We'll tackle that first.

Drilling Holes for the Electronics

To solve this problem, I mounted the transceiver directly to the box and just aimed the box as required. To mount the module, drill 5/8-inch holes in the mounting box and use standoffs to hold the board securely. See the template in Figure 5-8 for drilling measurements. A PDF of the template is available in this book's online resources at *http://www.nostarch.com/arduinoplayground/*, in case you want to print it and lay it over your box as a guide. The box I recommend is made of polycarbonate plastic and is less likely to crack than styrene or acrylic; however, it tends to catch the drill, so be careful.

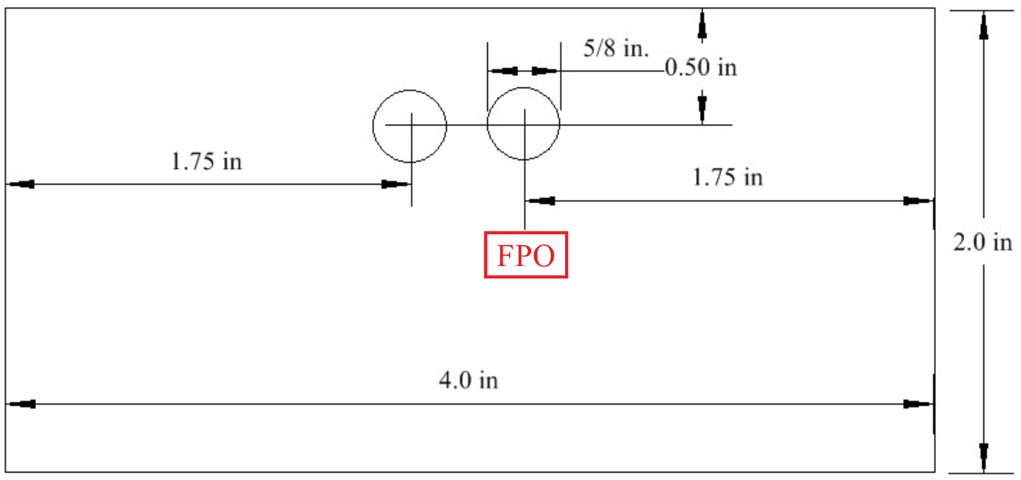

Figure 5-8: Template for drilling transducer holes

There are several ways to drill the 5/8-inch holes. If you are good at drilling, you could simply use a 5/8-inch drill bit and bore the holes directly. But I discovered that the holes can be bored safely and easily by first drilling a hole about 1/4 to 3/8 inches in diameter and then enlarging it with a tapered reamer, available from Amazon for under $15. The larger reamer in the Amazon set will ream a hole up to 7/8 inches in diameter, and it is handy to have around for other projects. Use a 1/8-inch drill to drill the holes for the standoffs, as shown in the drawing, which you can use as a template.

If you ream out the hole, make sure to ream from both sides. Enlarge the hole to a size that holds the transducer elements tightly—but not too tightly. While this is not the most precise way to bore a hole and would probably be frowned upon by professional machinists, it works well enough here.

WARNING *Regardless of the size of the hole, do not hold your work piece with your hand when drilling. Always clamp it securely. If the drill binds, the work will want to spin or climb up the drill. Drill at a slow speed and go gently.*

Next, drill the holes for the potentiometer, power jack, and two LEDs. Select a drill size based on the particular power jack and potentiometer you have. I used a 9/32-inch drill for the potentiometer, a 1/4-inch drill for the 3.5 mm jack, and a bit of approximately 25/64 inches for the LEDs. The size of the 10 mm LEDs tends to vary a bit from manufacturer to manufacturer, so I would recommend that you select a smaller drill bit, say 3/8 inches, and ream until the LED fits tightly. Because the LED is tapered, ream from the rear of the box so that the LED will fit better.

The location of both the potentiometer and power jack is not important, but make sure that neither crowds the transducer or Nano. You want them to be on the bottom of the enclosure so that they are accessible after the box is mounted (see Figure 5-12).

Mounting Options

Before you stuff the Arduino, ultrasonic sensor, and perforated board circuit into your enclosure, figure out how you want to mount the Garage Sentry. There are several ways to mount the box onto whatever surface you need.

Velcro Strips

If you have a good flat surface to mount the assembly to, you could simply affix the box with adhesive Velcro (see Figure 5-9). Two sentries have been in place in my garage that way for several months, with no sign of slippage or deterioration.

Figure 5-9: Adhesive Velcro mounting strips used to mount the Garage Sentry enclosure

A U-Bracket That Can Be Aimed

If you don't have a good surface and need to aim the module at an angle, mounting it on a U bracket that lets the sensors swing up and down or left and right will work. In this section, I'll describe how to build the U bracket mount shown in Figure 5-10.

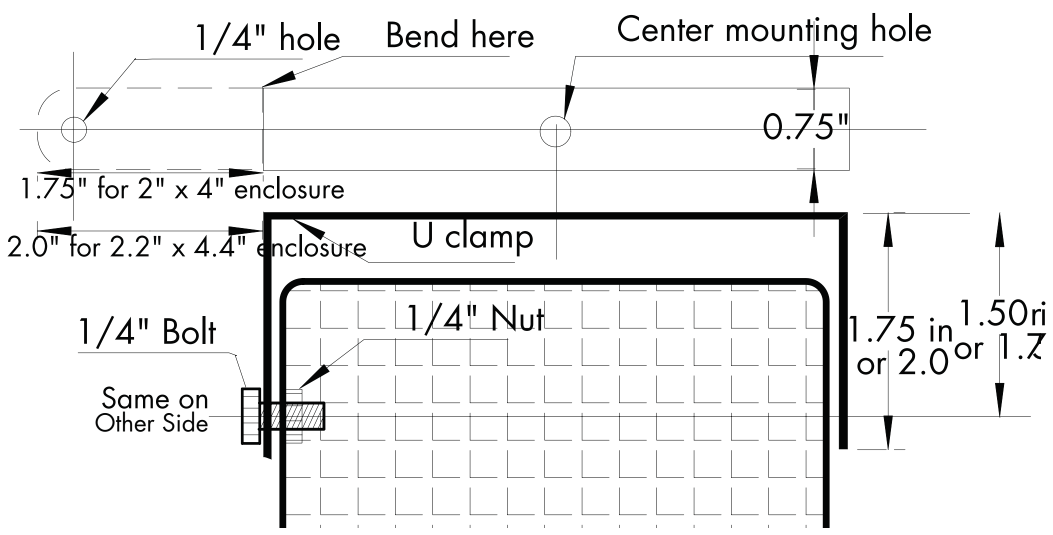

Figure 5-10: This drawing illustrates the size and shape of the optional U bracket handle and how it connects to the enclosure. Where you see two measurements for a single dimension, the smaller applies to the Standard Garage Sentry, while the larger applies to the deluxe version.

To make the U bracket for the 1591 ATCL 2 × 4-inch box, take a strip of 3/4-inch × 0.080-inch ×5 1/2-inch long aluminum (available at Ace Hardware, Home Depot, or Lowe's), and drill 1/4-inch holes 5/16 inches from the ends of the aluminum strip. Drill corresponding holes with a No. 7 or 15/64 drill in the side of the enclosure centered on the ends, and thread the holes with a 1/4-inch-20 tap. Bend the aluminum strip 1.5 inches from each end for the standard version and 2 inches for the deluxe version (see Figure 5-10). Using a vise is the easiest way to bend the metal, but if that's not convenient, you can sandwich it between a bench and piece of metal, clamp it down, and bend it by hand (see Figure 5-10).

For the U bracket for the 1591 BTCL 2.2 × 4.4-inch box, use a 6 3/8-inch long strip of the same material, and drill the 1/4-inch holes 1/2 inches from the ends. Then, bend the aluminum at right angles at 3/4 inches from either end for the standard version and 1 inch from each end for the deluxe version.

To fasten the U bracket to either enclosure, you can start by drilling a hole in the center of each end of the enclosure. It's simplest to drill a No. 7—15/64 is close enough—hole at either end of the box. The easiest way to center the holes is to draw a line along each diagonal on both ends. Where the lines intersect is the center. Thread the holes with a 1/4-inch-20 tap, and you'll be able to fasten the box to the U bracket directly. The threads in the thin ABS plastic will not be very strong, so be careful not to overtighten the bolts.

When you're finished attaching the bracket to your enclosure, it should look like Figure 5-11.

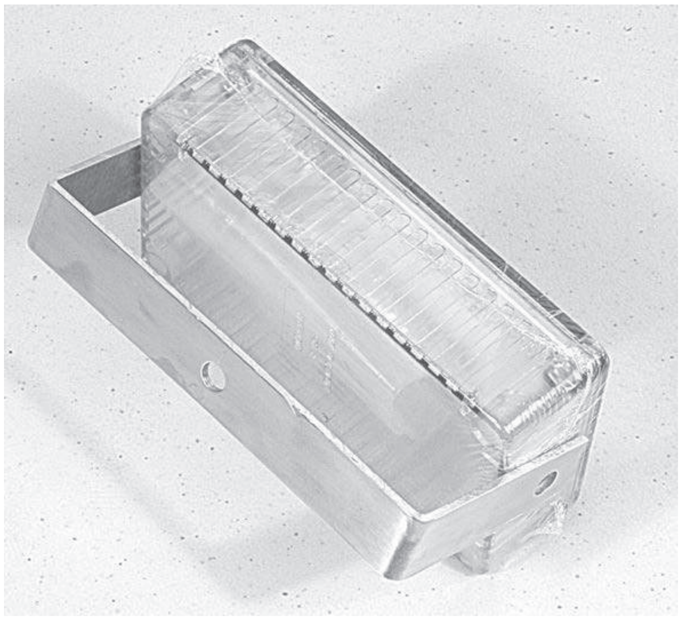

Figure 5-11: The enclosure for the Garage Sentry can be mounted with the bracket so it can be tilted or rotated to point the transducers in the correct direction.

Soldering the Transistors and Current-Limiting Resistors

After testing your circuit on a breadboard and deciding how to mount the Garage Sentry, solder the driver transistors and current-limiting resistors to a small section of perforated phenolic or FR-4 predrilled board. Use the schematic in Figure 5-4 or the instructions in "The Breadboard" on page 138 as a guide to wiring and soldering the components in the perforated board.

Make the connections in the schematic, but otherwise, there is no right or wrong way to assemble the perf board. I do recommend using perforated board with copper pads for each hole to simplify soldering. Solder all the hookup wires for the power, potentiometer, Nano, ultrasonic module, and LEDs before attempting to mount the board on the inside of the box.

When you're done soldering, mount the perforated board anywhere in the box where you can find room. I used double-sided foam adhesive, and it worked well. Mount the Nano, LEDs, and ultrasonic module next.

Wiring the Pieces Together

Finally, use 30-gauge hookup wire to connect the Nano, ultrasonic sensor, perforated board circuit, and LEDs according to the schematic in Figure 5-4. Optionally, you can use wire-wrap wire and a wire-wrap tool to wire up the sections, but it is not necessary and can be expensive if you don't already have the tool and wire.

Wiring the components and fitting them in the box may be a little messy, but it saves building a shield. Figure 5-12 shows the box as it was being assembled.

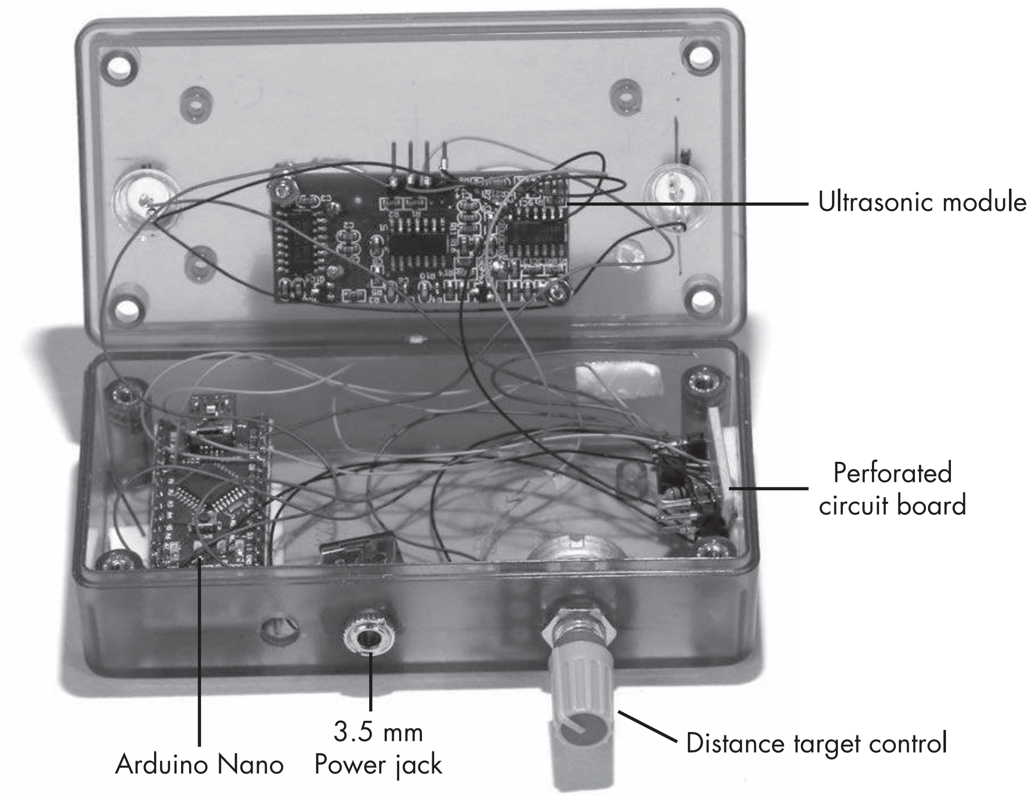

Figure 5-12: The Standard Garage Sentry uses wire wrap for the final connections.

The Deluxe Garage Sentry

That's it! Or is it?

I have been using the standard model in my garage for several months; it does what it's supposed to do and does it well. But it seems like something's missing. The alarm goes off when you reach the desired spot in the garage, but why not have it give you a little warning before you get there so you can slow down as you approach the stopping point?

The idea is to have the system warn you at some pre-established distance from the stopping point so you don't have to stop suddenly. It isn't much extra effort to add two more LEDs to go off at different distances. Figure 5-13 shows the Deluxe Garage Sentry.

Now, let's discuss how to assemble the Deluxe Garage Sentry.

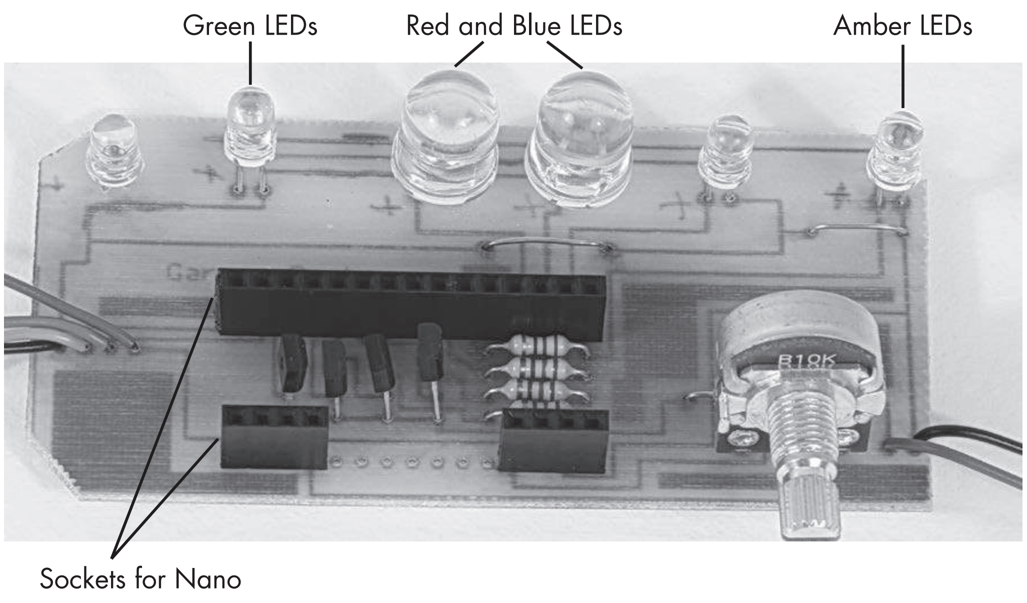

Figure 5-13: The Deluxe Garage Sentry sets off three stages of alarms.

The Deluxe Schematic

Hand-wiring everything in the standard version is tedious. So for the deluxe version, I developed a shield (PCB) that holds the LEDs, potentiometer, Nano, transistors, and current-limiting resistors (see Figure 5-13). Adding the LEDs and extra transistors required some changes in the circuitry. Figure 5-14 shows the revised schematic.

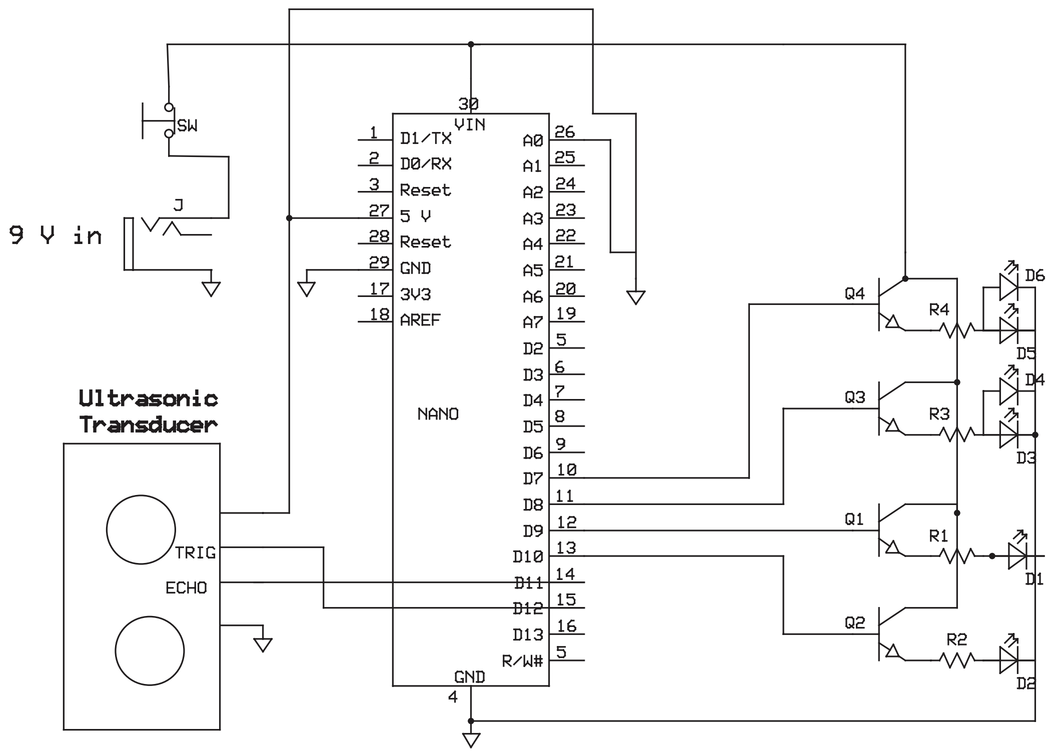

Figure 5-14: The deluxe schematic has additional LEDs, driver transistors, and current-limiting resistors on the right-hand side.

Note that this circuit drives the transistors as emitter followers. As such, the base shows a high resistance, and therefore no resistor is required between the Arduino and the resistors Q1 through Q4. If you were driving using a common emitter, however, you would need a resistor, as current would flow through the base-emitter junction, short out the driver, and burn out the transistor.

NOTE *To improve the Garage Sentry further, you could also conceivably double or otherwise increase the range using special transducers and electronics, but in this application, the 10-foot operating range is more than enough.*

A Bigger Box

Both the green and amber LEDs operate as pairs, so only a single driver transistor is required for each pair. But with all this new circuitry, the deluxe board does not easily fit in the same enclosure as the standard version.

You'll need to find a larger box for the deluxe version, which provides you with some other benefits. With a larger, clear polycarbonate enclosure, like Hammond 1591 BTCL, the LEDs can stay inside the box and still be visible so you won't have to drill holes to mate with the LEDs in the PC board. You'll have to drill only four holes: the two large holes for the ultrasonic sensor, a hole for the power jack, and one for the potentiometer. These holes make it possible to mount the ultrasonic sensor on top of the Nano board with double-sided foam tape, which is in turn mounted to the shield (see Figure 5-15). In other words, you create a sandwich with the Nano in the middle, the shield on the bottom, and the ultrasonic module on the top. This design eliminates the need for the mounting screws used in the first version.

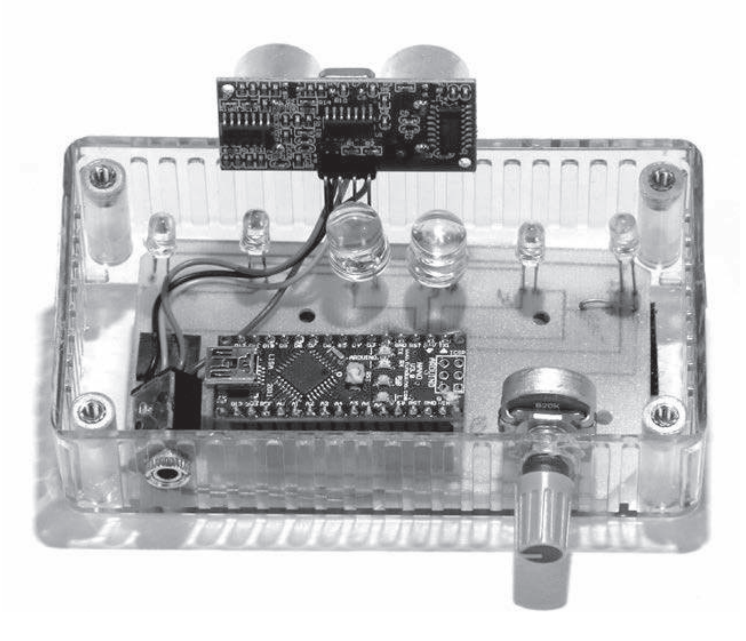

Figure 5-15: Compared to the Standard Garage Sentry, there is virtually no hand wiring in the deluxe version. The driver transistors and current-limiting resistors for the LEDs are located under the Nano board.

Simply use the same template you used in the standard version, and drill (and/or ream) the two 5/8-inch holes for the two ultrasonic elements. The shield itself can be fastened to the bottom of the box with small flat-head screws and nuts or with more double-sided foam tape. When you affix the potentiometer to the box, it should hold the board in place, as its leads are soldered to the shield. Figure 5-15 shows the deluxe version; note how much neater it is than the standard version from Figure 5-12.

Before drilling the hole for the potentiometer, carefully measure the height of the potentiometer hole and drill the hole slightly oversized so that the shaft and screw can be inserted at an angle into the box. You'll also note in Figure 5-15 that the corners of the printed circuit board have been clipped off so as not to get in the way of the studs used for the top screws of the enclosure.

I suggest mounting the power connection on the same surface of the box as the potentiometer—that is, the bottom. Here, the power connection and potentiometer will be accessible after mounting, leaving the top free to fit snugly against a shelf. The unit can just as easily be mounted upside down with the adjustment and power jack on the top.

The Shield

Figure 5-16 shows the shield for the Deluxe Garage Sentry. If you want to build this shield, download this book's resource files, look for the file *GarageSentryDel.pcb*, follow the etching instructions in "Making Your Own PCBs" on page XX, and solder your components to the board. You can also take the file and send it out to one of the service bureaus to have the board made for you.

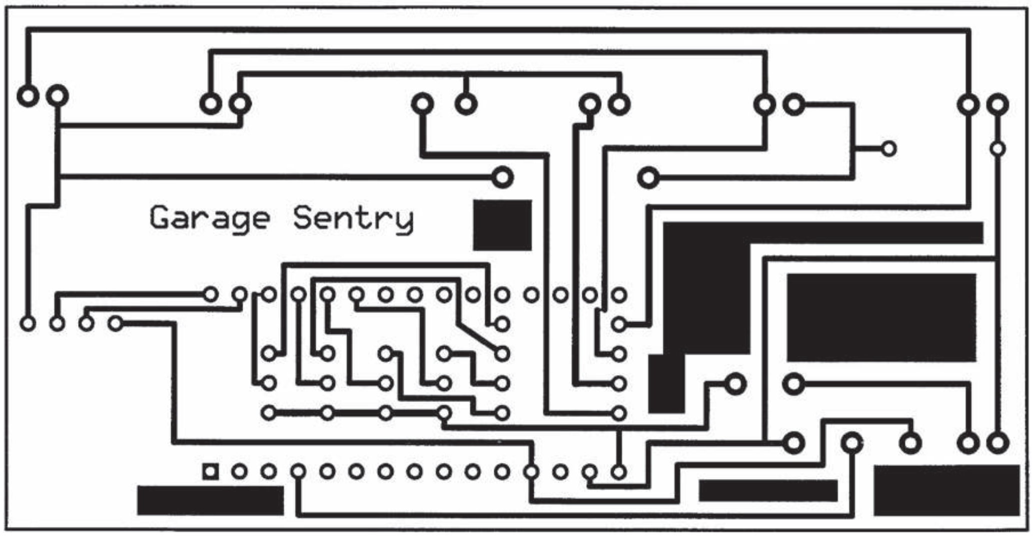

Figure 5-16: This is the shield for the Deluxe Garage Sentry, which simplifies the individual wires that had to be soldered to complete the earlier version.

The potentiometer is soldered directly to the shield, though you'll still have to solder wires to the power jack and to the ultrasonic module. As you can see in Figure 5-16, the connections for the ultrasonic sensor are located on the left-hand side.

The green LEDs are on the outermost edges, the amber LEDs are next, and the flashing red and blue LEDs are in the middle. The connections for the potentiometer are on the lower right-hand side, next to the first two connections, which are ground and VIN (from left to right). The potentiometer helps hold the shield in place in the enclosure.

This version uses high-power LEDs that draw a fair amount of current. Because the sentry uses the 5V voltage regulator on the Nano, the transistors driving the LEDs are wired directly to the 9V input voltage, allowing the unit to function without a separate voltage regulator. The LEDs are configured with the driver transistors as emitter followers, so the voltage to the LEDs will "follow" the voltage on the base of the transistor—that is, 5V—and not present LEDs with 9V.

There are several jumpers required on this shield, including the jumper for the power, which connects to the collectors of the transistors and connects the raw input to the (VIN) Nano. There are also jumpers to connect the ground to the LEDs.

I mounted the transistors and current-limiting resistors under the Nano to save some space. Also, note that the connections for the ultrasonic module in the standard version call for a right-angle female header, but doing that in the deluxe version means the length of the male part gets in the way of the LEDs, so I simply soldered the connections to the PC board and to the ultrasonic module to keep the wires out of the way.

The Sketch for the Deluxe Garage Sentry

Before you build the Deluxe Garage Sentry, download *GarageSentryDel.ino* from this book's resource files at *http://www.nostarch.com/arduinoplayground/*, and upload it onto your Arduino Nano according to the instructions provided in Chapter 0 on page XX. The sketch is basically the same as the Standard Garage Sentry sketch, but it's updated to include the new LEDs.

```
/* Deluxe Garage Sentry: goes with the shield PCB
*/

int ledPin = 8;
int ledPin1 = 7;
int ledPin2 = 10;
int ledPin3 = 9;
int count;
int analogPin = A0;
int val;
int y;
```

```
  void setup() {
    // initialize serial communication:
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT);
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2,OUTPUT);
    pinMode(ledPin3,OUTPUT);
    pinMode(analogPin, INPUT);
  }

  void loop() {
    val = analogRead(analogPin);

    long duration, inches, cm;

    pinMode(12, OUTPUT);
    digitalWrite(12, LOW);
    delayMicroseconds(2);
    digitalWrite(12, HIGH);
    delayMicroseconds(5);
    digitalWrite(12, LOW);

    pinMode(11, INPUT);   //attached to Echo
    duration = pulseIn(11, HIGH);

    // convert the time into a distance
    inches = microsecondsToInches(duration);
    cm = microsecondsToCentimeters(duration);

    val = map(val, 0, 1023, 0, 100);
    //map the value of the potentiometer to 0 to 100

    if(inches == 0)
      digitalWrite(ledPin, LOW);

❶   if(count == 0 && inches > 0 && inches < val + 15)
      digitalWrite(ledPin2, HIGH);
    else digitalWrite(ledPin2, LOW);

❷   if(count == 0 && inches > 0 && inches < val + 7.5)
      digitalWrite(ledPin3, HIGH);
    else digitalWrite(ledPin3, LOW);

    if(count == 0 && inches > 0 && inches < val) {
❸     for(y = 0; y < 200; y++) //repeating blink sequence {
        digitalWrite(ledPin, HIGH);
        digitalWrite(ledPin1, LOW);
        delay(100);
        digitalWrite(ledPin, LOW);
        digitalWrite(ledPin1, HIGH);
        delay(100);
      }
    count = count + 1; //turn off instruction
  }
  digitalWrite(ledPin1, LOW);

  if(inches > 10) { //reset if inches > 10
    delay(1000);
    count = 0;
  }

  Serial.print(inches);
  Serial.print("   inches ");
  Serial.print(count);
  Serial.print(" count   ");
  Serial.println();
  Serial.print(" val       ");
  Serial.println(val);
  delay(100);
}

long microsecondsToInches(long microseconds) {
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
  return microseconds / 29 / 2;
}
```

The most notable difference between the deluxe sketch and the standard sketch is that in the two `if-else` statements at ❶ and ❷, the green and amber LEDs are activated at different distances, based on the stopping point. For example, if the stopping point is set to 36 inches, or `Val` in the sketch, then the green LED turns on at `VAL + 15`, or 52 inches, and the amber LED turns on at `VAL + 7.5`, or 43.5 inches. This way the green LEDs will turn on when the car is 15 inches from the final stopping point, and the amber LEDs will turn on when the car is 7.5 inches from the stopping point. These numbers were selected arbitrarily, and you can change them.

The red and blue LEDs start flashing when the car has reached the stopping point. You can see how the LEDs are flashed at ❸.

Figure 5-17 shows the completed Deluxe Garage Sentry mounted on my garage workbench with the car in place.

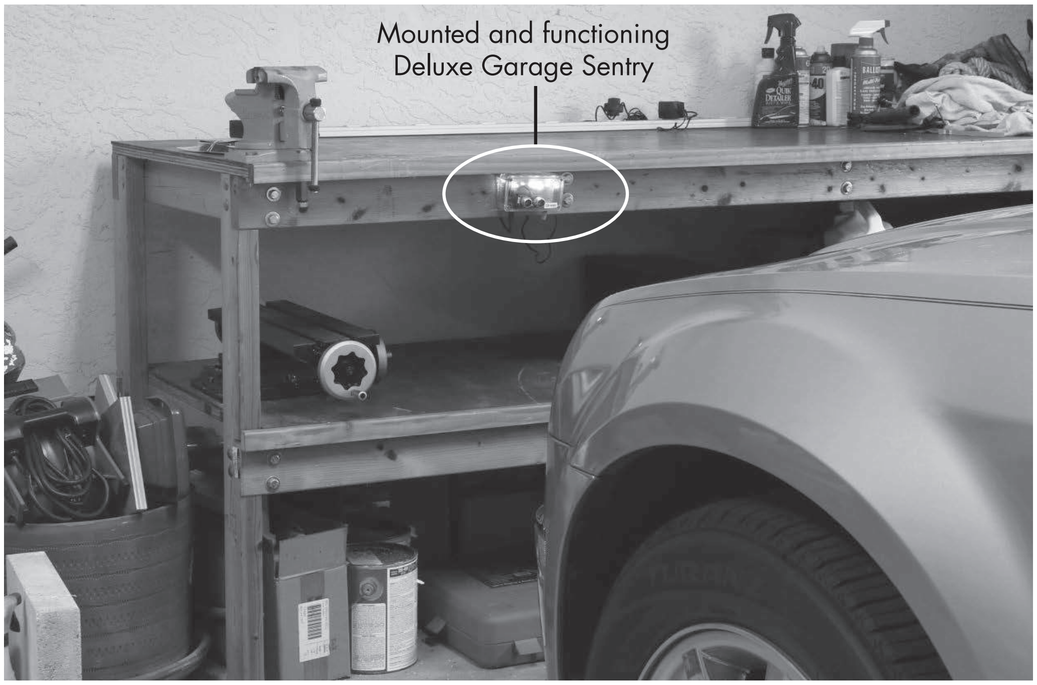

Figure 5-17: Completed Deluxe Garage Sentry mounted on my garage workbench with the car in place

The completed sentry unit works flawlessly. Depending on your particular garage and where you place the unit, you might want to adjust the sketch to make the green and amber lights turn on at different distances. The unit pictured has been working perfectly for almost six months now, and I don't know how I'd be able to pull my car in the garage without it.