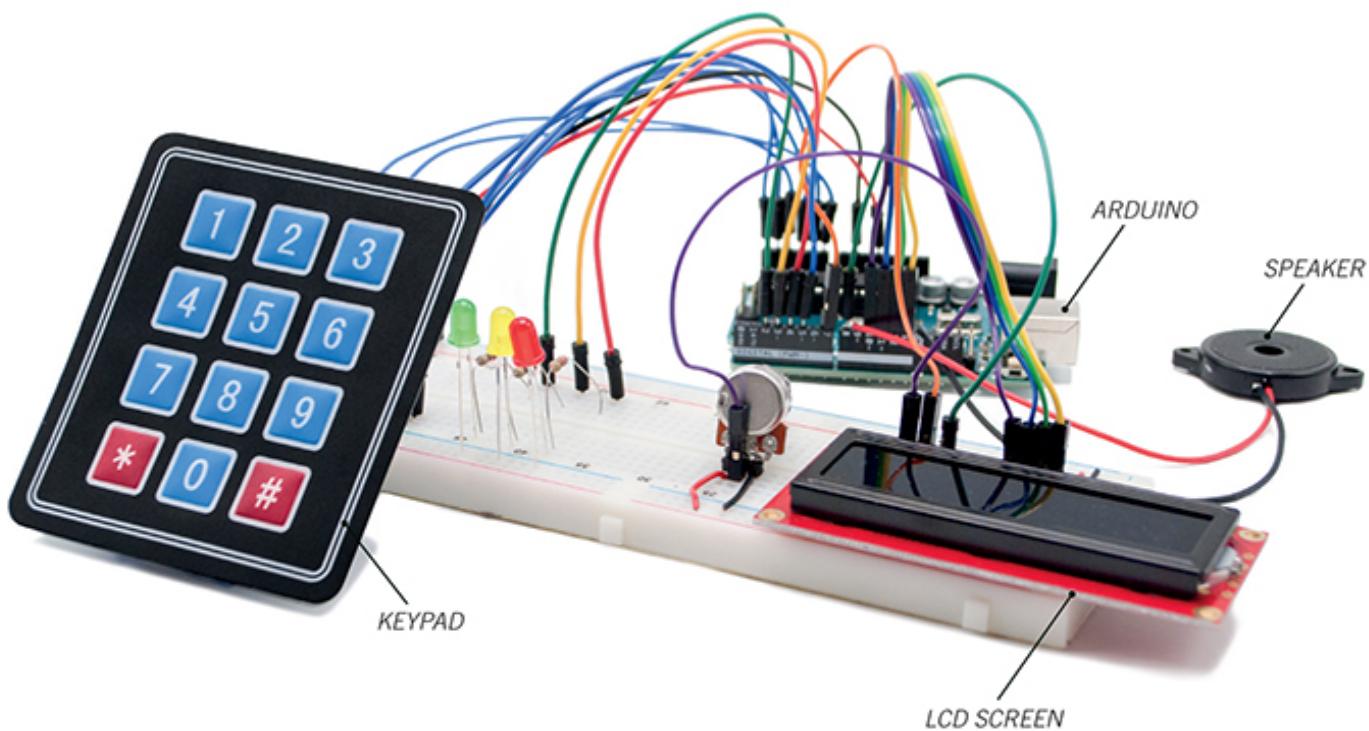


# ARDUINO PROJECT HANDBOOK

25 SIMPLE ELECTRONICS PROJECTS FOR BEGINNERS

VOLUME 2



MARK GEDDES



# **ARDUINO PROJECT HANDBOOK**

## **VOLUME 2:**

## **25 SIMPLE ELECTRONICS PROJECTS FOR BEGINNERS**

**MARK GEDDES**



**no starch  
press**

**SAN FRANCISCO**

## **ARDUINO PROJECT HANDBOOK, VOLUME 2.** Copyright © 2017 by Mark Geddes.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-10: 1-59327-818-7

ISBN-13: 978-1-59327-818-2

Publisher: William Pollock

Production Editor: Serena Yang

Cover and Interior Design: Beth Middleworth

Cover Photo: Max Burger

Developmental Editor: Liz Chadwick

Technical Reviewer: Sam Stratter

Copyeditor: Rachel Monaghan

Compositor: Serena Yang

Proofreader: James Fraleigh

Circuit diagrams made using Fritzing (<http://fritzing.org/>).

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

phone: 1.415.863.9900; [info@nostarch.com](mailto:info@nostarch.com)

[www.nostarch.com](http://www.nostarch.com)

*The Library of Congress has catalogued the first volume as follows:*

**Names:** Geddes, Mark.

**Title:** Arduino project handbook : 25 practical projects to get you started /  
by Mark Geddes.

**Description:** San Francisco : No Starch Press, [2016] | Includes index.

**Identifiers:** LCCN 2015033781 | ISBN 9781593276904 | ISBN 1593276907

**Subjects:** LCSH: Programmable controllers. | Microcontrollers--Programming. |  
Science projects--Design and construction. | Arduino (Programmable  
controller)

**Classification:** LCC TJ223.P76 G433 2016 | DDC 629.8/9551--dc23

LC record available at <http://lccn.loc.gov/2015033781>

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

**CAMERON AND JEMMA, YOU ARE THE CREATORS AND  
MAKERS OF THE FUTURE.  
THIS BOOK IS FOR YOU!**

# CONTENTS

## Introduction

## Primer: Getting Started

### LEDs

#### Project 1: LED Light Bar

#### Project 2: Light-Activated Night-Light

#### Project 3: Seven-Segment LED Count Down Timer

#### Project 4: LED Scrolling Marquee

#### Project 5: Mood Light

#### Project 6: Rainbow Strip Light

#### Project 7: NeoPixel Compass

### Sound

#### Project 8: Arduino Piano

#### Project 9: Audio LED Visualizer

### Motors

#### Project 10: Old-School Analog Dial

#### Project 11: Stepper Motor

#### Project 12: Temperature-Controlled Fan

### LCDs

#### Project 13: Ultrasonic Range Finder

#### Project 14: Digital Thermometer

#### Project 15: Bomb Decoder Game

#### Project 16: Serial LCD Screen

#### Project 17: Ultrasonic People Counter

#### Project 18: Nokia 5110 LCD Screen Pong Game

**Project 19: OLED Breathalyzer**

**Security**

**Project 20: Ultrasonic Soaker**

**Project 21: Fingerprint Scanner**

**Smart Machines**

**Project 22: Ultrasonic Robot**

**Project 23: Internet-Controlled LED**

**Project 24: Voice-Controlled LED**

**Project 25: GPS Speedometer**

**Troubleshooting Tips for Common Errors**

**Components**

**Arduino Pin Reference**

# Acknowledgments

Once again, many thanks to Bill Pollock and the fantastic team at No Starch Press for their dedicated support and guidance in the creation of this book, particularly Liz Chadwick and Serena Yang for being so patient through the process. Thanks also to Sam Stratton for his technical reviews and suggestions.

This book wouldn't exist if it wasn't for the inspirational Arduino founders; Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. Thank you again for introducing me and the world to the wonder that is Arduino.

Special thanks to Warwick Smith, James Newbould, Joey Meyer, Chase Cooley, Onur Avun, Nick Koumaris, Chris Campbell, Mouad Er Rafay, Pololu, and Brainy-Bits.com for their amazing support and kind permission to reproduce their projects. The creativity of the ever-growing Arduino community never ceases to amaze me.

Thanks to everyone who read *Arduino Project Handbook, Volume 1* for the kind words and messages of encouragement—it's made writing this volume that little bit easier.

Finally, I have to thank my wonderful wife, Emily, for being so supportive and patient over the last year—I promise that my “man cave” will not expand any further!

# Introduction

Welcome to *Arduino Project Handbook, Volume 2*. If you haven't read the first volume, don't worry—each project in this book is completely independent and designed to gently introduce you to the world of building with Arduino. We'll cover some of the important aspects of getting started with Arduino here and in the next chapter, so if you've read Volume 1 you can either skim through as a refresher or skip ahead to dive straight into the new projects.

This book uses the Arduino Uno, a small, inexpensive computer that can be programmed to control endless devices and creations. You'll soon use the Arduino to control a whole host of projects, like a musical keyboard, temperature-controlled fan, digital thermometer, fingerprint entry system, and many others.

The Arduino board is composed of two main elements: the hardware, or microcontroller, which is the brain of the board; and the software that you'll use to send your program to the microcontroller. The software, called the Arduino integrated development environment (IDE), is available free for download, and I'll show you how to use it to set up a simple project in the primer.

## ABOUT THIS BOOK

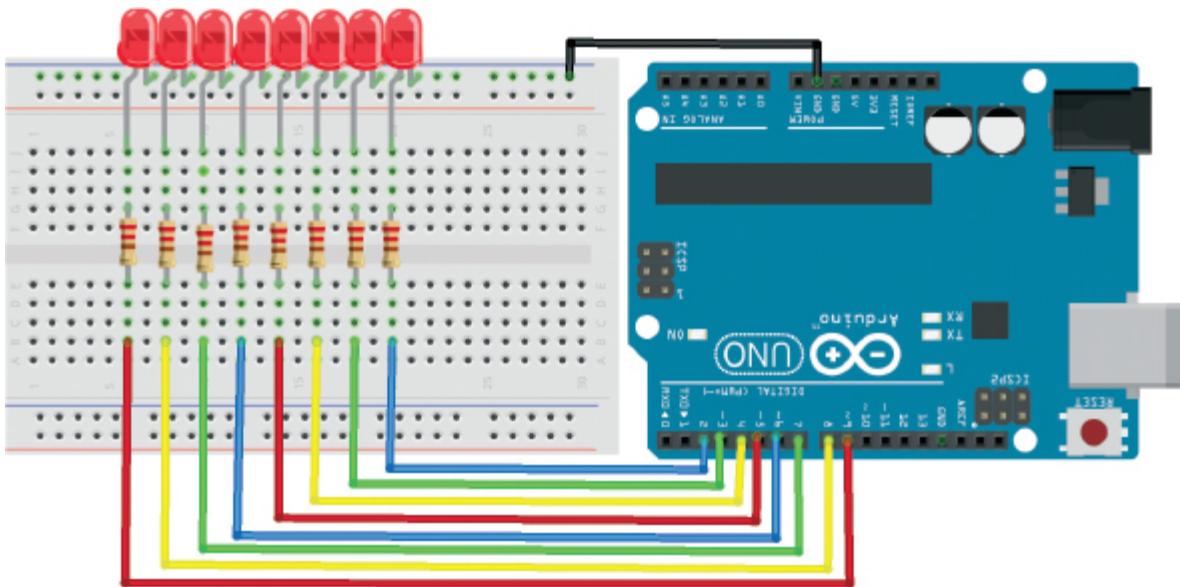
What inspired me to write this book? The internet is bursting with tutorials, videos, and articles covering the Arduino and potential projects, but many lack detailed visuals or the code required to build these projects. This book is intended to help you build simple projects that will inspire you to create your own inventions as you apply the skills and techniques that you'll learn.

### NOTE

*In this book you'll create your projects on a breadboard. This is the best way to learn about how circuits work, because the connections are not permanent; if you make a mistake, you can just unplug the wire or component and try again.*

Each project includes a description of what it will do, the items you'll need, pictures of the setup, simple step-by-step instructions with tables for quick connection references, a circuit diagram (see Figure 1), and the necessary code, so you don't have to worry about learning to program before you begin. The early projects provide simple explanations of what's happening in the code, to help you understand the process of programming enough to make your own modifications if you want to. If you don't want to type that much code out, the sketches are available to download at <https://www.nostarch.com/arduinohandbook2/>.

**FIGURE 1:** The circuit diagrams in this book were created with Fritzing (<http://www.fritzing.org/>), a free, open source program.



At the beginning of each project, I include an indication of the cost of the components required in addition to the Arduino Uno (see Table 1) and an estimated time for the build. At the end, I provide a troubleshooting section specific to that project.

**TABLE 1:** The cost indication used in this book

INDICATOR	COST
\$	\$1–\$9
\$\$	\$10–\$19
\$\$\$	\$20–\$29
\$\$\$\$	\$30+

I've written this book to teach you how to create your own gadgets. By giving you the technical know-how, I allow you to focus on the creative design element. The idea is that learning the function of circuits can open up your imagination to ways of using those circuits practically. Although I don't delve deeply into electronics theory or programming, the projects in this book progress steadily in complexity and will give you a good starting point.

This book gives you practical information so you can, for example, reference the pin connections and replicate them when needed in a different project. You can also combine projects to make more complicated and interesting gadgets. A lot of Arduino books focus on the programming element, and that's great for a certain kind of learning, but I think there's also a place for plug-and-play electronics. By following the steps in the projects, you'll learn as you go.

I've written the book that I was looking for but couldn't find when I started out with the Arduino. I hope you'll enjoy reading and working through this book as much as I enjoyed writing it.

## ORGANIZATION OF THIS BOOK

I recommend you try out some of the earlier projects first, as you'll find information there that's useful for the more complicated builds, but if

you see a project you like and feel confident enough to take it on, you can skip to it. The parts of the book are organized as follows:

**Primer: Getting Started** Learn all about the Arduino Uno and how to use a breadboard, and then test your board with a simple program and get a crash course in soldering.

**Part I: LEDs** Here you'll start out by learning how to control simple light-emitting diodes (LEDs) with variable resistors, and then combine components to build a light-activated LED, a scrolling text display, a flashing multicolored compass, and more.

**Part II: Sound** In this part, you'll use a *piezo*, a device that emits sound, to make tunes with a musical keyboard and create a simple audio visualizer that makes LEDs dance to your music.

**Part III: Motors** These projects use various types of motors to bring your creations to life. You'll build an analog dial that gauges light levels, learn how a stepper motor works, and build a temperature-controlled fan to keep you cool.

**Part IV: LCDs** The LCD screen is useful in lots of projects for displaying messages and results. In these projects, you'll learn how to set up a serial LCD screen and then build a defusable bomb game, an ultrasonic range finder, a mobile *Pong* game, and even an alcohol breathalyzer.

**Part V: Security** Protect your space with a motion sensor that triggers an ultrasonic soaker water pistol and a security system that uses a fingerprint scanner to keep unauthorized persons out.

**Part VI: Smart Machines** In this final part you'll combine the Arduino with motors and sensors to create an intelligent robot, control lights using Bluetooth technology, and even build a GPS speedometer to track your movements.

At the end of the book, I provide some helpful reference information, including a review of some of the more common program errors and how to fix them, information on the components used in this

book and where to buy them, and a reference table for the pins on the Arduino Uno.



ARDUINO

BOARD MODEL  
**UNO** R3  
ELECTRONICS  
OPEN-SOURCE PLATFORM  
PROTOTYPING  
MADE IN ITALY  
[www.arduino.cc](http://www.arduino.cc)



COMPLIANT FOOTPRINT  
ROHS  
ZERO  
TMPATTO

# Primer: Getting Started

Before you start building with the Arduino, there are a few things you need to know and do. First, let's take a look at the hardware and software you'll need for this book. Then, you'll test out the Arduino with a simple LED project and get started with a few techniques that will come in handy, like soldering and downloading useful code libraries.

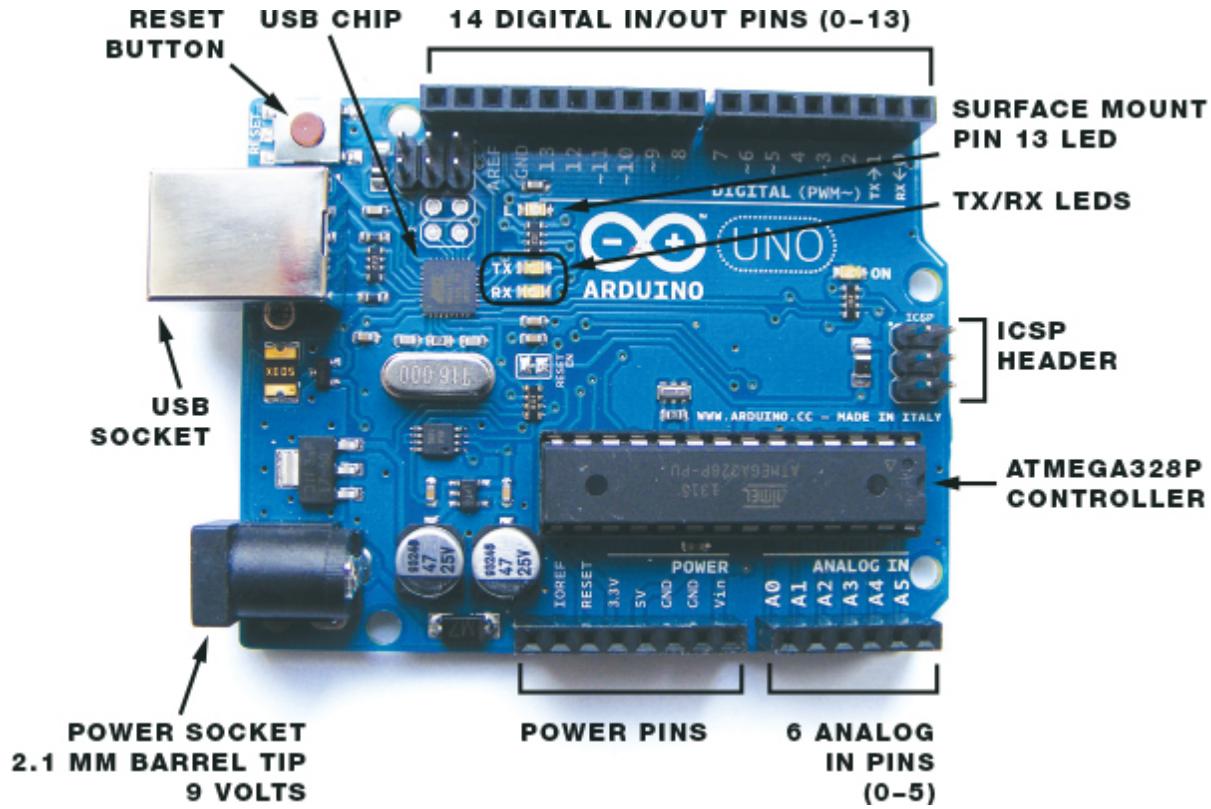
## HARDWARE

First let's look at the Arduino Uno board and a few pieces of hardware that you'll use in almost every project.

### The Arduino Uno

There are numerous types of Arduino boards available, but this book uses only the most popular one, the Arduino Uno shown in Figure 0-1. The Arduino Uno is open source (meaning its designs may be freely copied), so as well as the official board, which costs about \$25, you will find numerous compatible clone boards for around \$15.

FIGURE 0-1: The Arduino Uno board

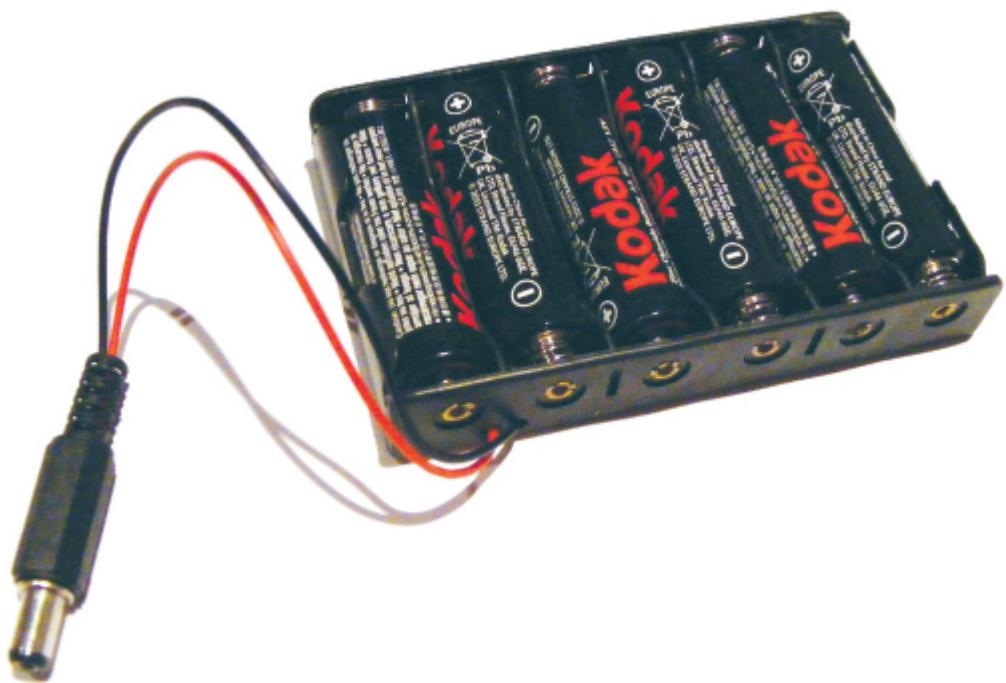


The Arduino controls components you attach to it, like motors or LEDs, by sending information to them as *output* (information sent *out* from the Arduino). Data that the Arduino reads from a sensor is *input* (information going *in* to the Arduino). There are 14 digital input/output pins (pins 0–13) on the Arduino. Each can be set to either input or output (see “Arduino Pin Reference” on page 253 for a full pin reference table).

## Power

When you connect the Arduino Uno board to your PC to upload a program, it is powered from your computer’s USB port. When the Arduino is not linked to your PC, you can have it run independently by connecting it to a 9-volt AC adapter or 9-volt battery pack with a 2.1 mm jack, with the center pin connected to positive power as shown in Figure 0-2. Simply insert the jack into the power socket of the Arduino.

**FIGURE 0-2:** A 9-volt battery pack, which you can plug into the Arduino to give it power

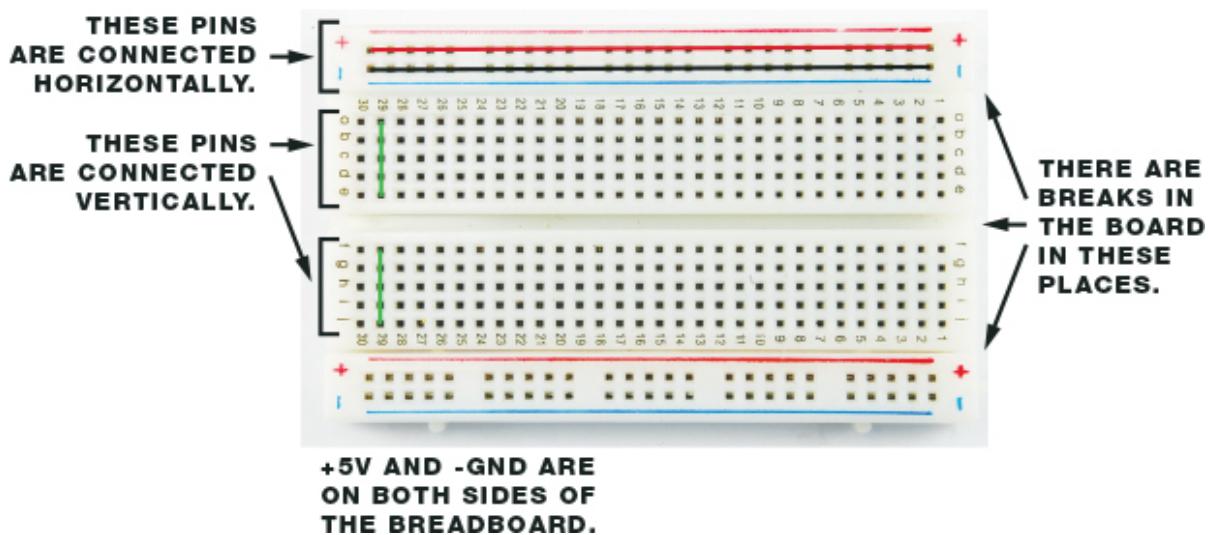


## Breadboards

A breadboard acts as a construction base for electronics prototyping. You'll use a breadboard for all of the projects in this book instead of soldering parts together.

The name *breadboard* dates back to when electronics projects were created on wooden boards. Hobbyists hammered nails into the wood and wrapped wires around them to connect components without having to solder them permanently. Today's breadboards are made of plastic with predrilled holes (called *tie points*) into which you insert components or wires, which are held in place by clips underneath. The tie points are connected by lengths of conductive material that run beneath the board, as shown in Figure 0-3.

**FIGURE 0-3:** Breadboard connections



Breadboards come in various sizes. To build the projects in this book, you'll ideally need three breadboards: one full-size, typically with 830 holes; one half-size, with about 420 holes; and one mini board with 170 holes. The full-size breadboard is ideal for projects that use an LCD screen or a lot of components, and the half-size and mini boards are best for smaller projects. For the projects in this book, I recommend that you buy breadboards that look like the one shown in Figure 0-3, with red and blue lines and a center break between the holes.

### TIP

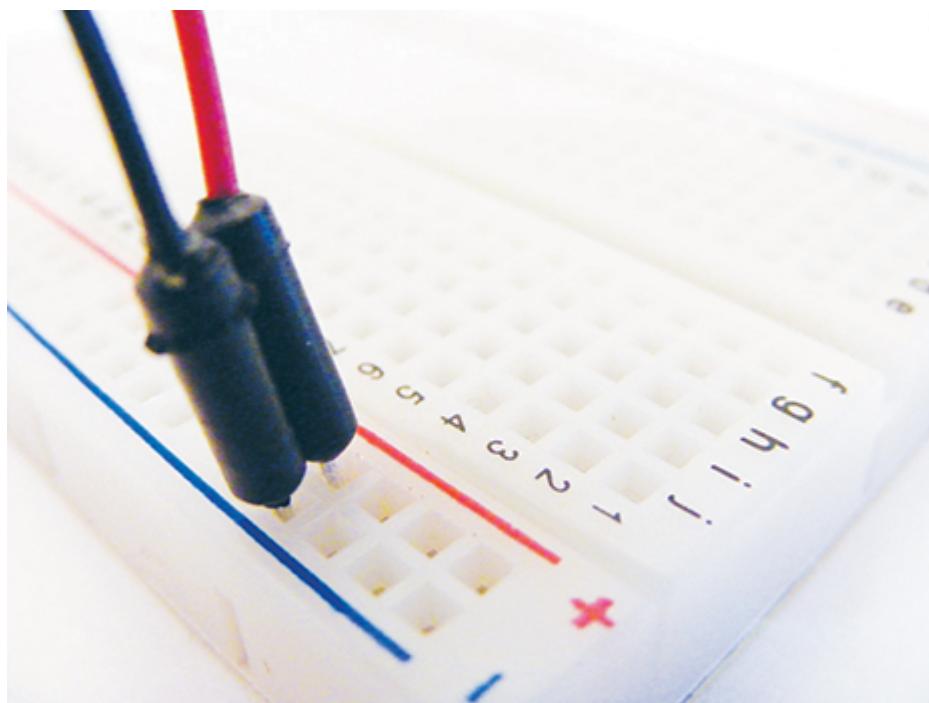
*It's useful to use red wires for connections to 5V and black wires for connections to ground (GND). The rest of the wires can be your choice of color.*

The main board area has 30 columns of tie points that are connected vertically, as shown in Figure 0-3. You'll often have to position components so they straddle the breadboard's center break to complete your circuit. This break helps to prevent components from short-circuiting, which can derail your project and even damage your components. You'll learn more about this as you start to build.

The blue and red lines at the top and bottom are power rails that you use to power the components inserted in the main breadboard area

(see Figure 0-4). The power rails connect all the holes in the rail horizontally; the red lines are for positive power and the blue lines for negative power (or *ground*, as you'll often see it called).

**FIGURE 0-4:** Positive and negative breadboard rails

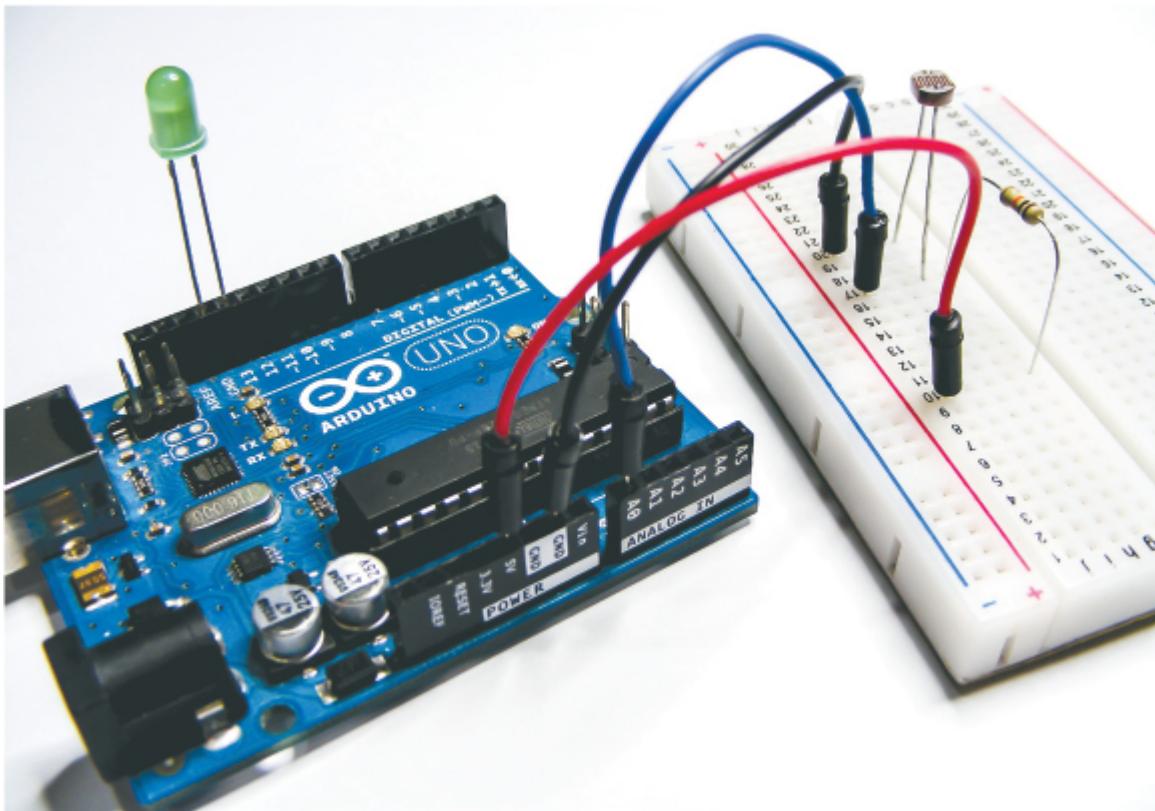


## Jumper Wires

You'll use *jumper wires* to make connections on the breadboard. Jumper wires are solid-core wire with a molded plastic holder on each end that makes it easier to insert and remove the wires. (You could use your own wire if you have it, but make sure to use solid-core wire—stranded wire is not strong enough to push into the hole clips.)

When you insert a jumper wire into a breadboard hole, it's held in place from beneath the board by a small spring clip, making an electrical connection in that row. You can then place a component in an adjoining hole to help create a circuit, as shown in Figure 0-5.

**FIGURE 0-5:** An example breadboard circuit



## NOTE

*Because the IDE versions can change fairly quickly, I won't take you through installing them, but installation should be straightforward and the instructions on the Arduino site are clear. All versions of the IDE and full details of how to install for your operating system are available at <http://www.arduino.cc/>.*

## PROGRAMMING THE ARDUINO

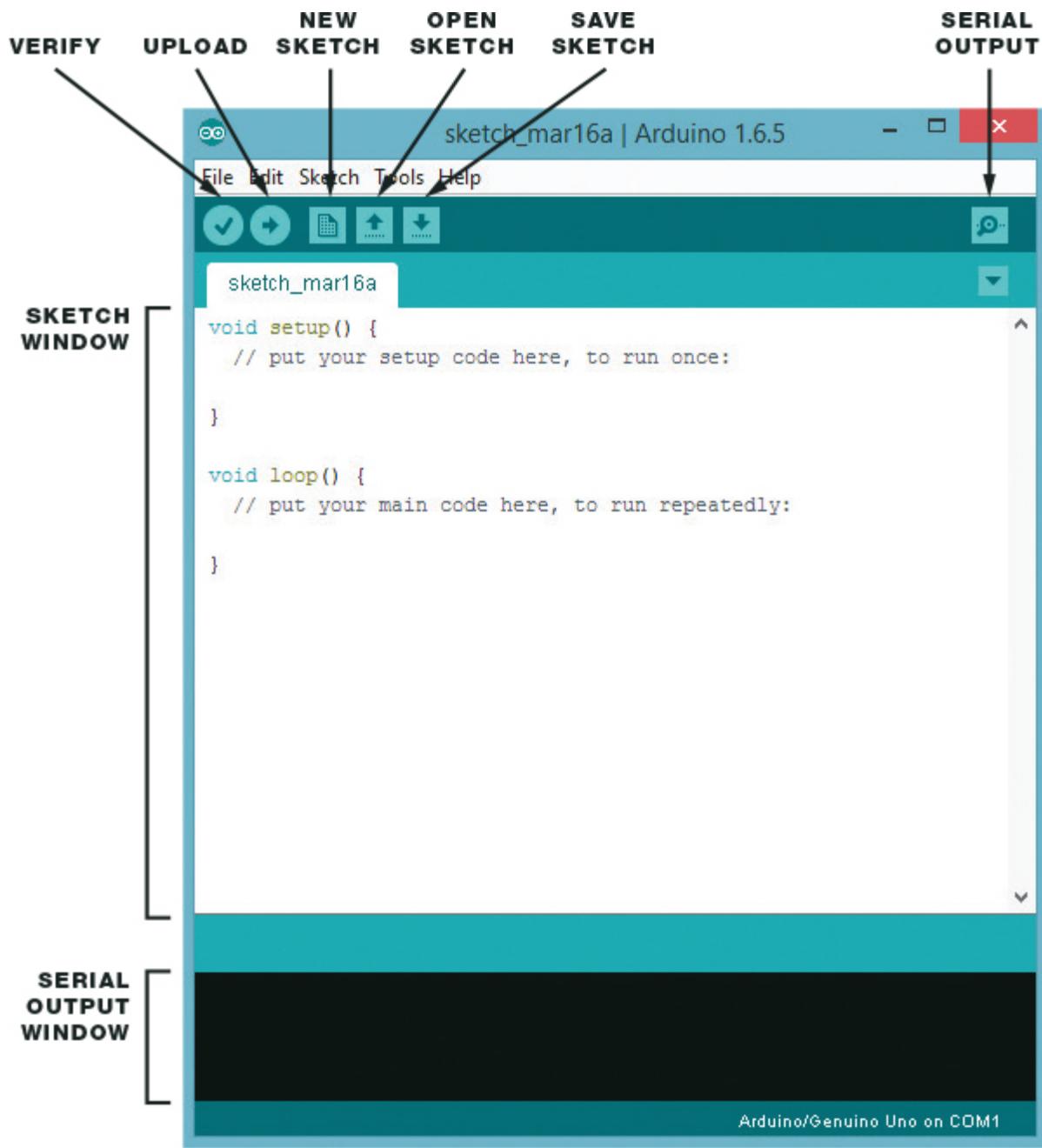
To make our projects do what we want, we need to write programs that give the Arduino instructions. We do so using the Arduino *integrated development environment (IDE)*. The Arduino IDE is available to download free from <http://www.arduino.cc/>, and will run on Microsoft Windows, OS X, and Linux. It enables you to write computer programs (a set of step-by-step instructions, known as *sketches* in the Arduino world) that you then upload to the Arduino using a USB cable. Your

Arduino will carry out the instructions based on its interaction with the outside world.

## The IDE Interface

When you open the Arduino IDE, it should look similar to Figure 0-6. The IDE screen is divided into a toolbar at the top with buttons for the most commonly used functions; the sketch window in the center, where you'll write or view your programs; and the Serial Output window at the bottom. The Serial Output window displays communication messages between your PC and the Arduino, and also lists any errors if your sketch doesn't compile properly.

**FIGURE 0-6:** The Arduino IDE



## Arduino Sketches

I'll give you the sketch for each project within the relevant project itself, and talk through it there. All of the sketches are available to download from <http://www.nostarch.com/arduinohandbook2/>.

Like any program, sketches are a very strict set of instructions and very sensitive to errors. It's best to download the sketch and open the

file in the IDE, rather than try to copy it from the book. To make sure it works correctly, click the green check mark at the top of the screen. This is the Verify button, and it checks for mistakes and tells you in the Serial Output window whether the sketch has compiled correctly.

## Libraries

In the Arduino world a *library* is a piece of code that carries out a specific function. Rather than enter this same code repeatedly in your sketches wherever you need, you can simply add a command that borrows that code from the library. This shortcut saves time and makes it easy for you to connect to items such as a sensor, display, or module.

The Arduino IDE includes a number of built-in libraries—such as the LiquidCrystal library, which makes it easy to talk to LCD displays—and there are many more available online. To create the projects in the book, you'll need to import the following libraries: PololuLedStrip, FastLED, HMC5883L, Keypad, Tone, Adafruit\_GFX, Adafruit\_SDD1306, NewPing, Adafruit Fingerprint Sensor, and Adafruit Motor Shield. You'll find all of the libraries you need in the resources at <http://www.nostarch.com/arduinohandbook2/>.

## Installing Libraries

Once you've downloaded the libraries, you'll need to install them. To install a library in Arduino version 1.0.5 and higher, follow these steps:

1. Choose **Sketch ▶ Include Library ▶ Add .ZIP Library**.
2. Browse to the ZIP file you downloaded and select it. In older versions of Arduino, unzip the library file and put the whole folder and its contents into the *sketchbook/libraries* folder on Linux, *My Documents\Arduino\Libraries* on Windows, or *Documents/Arduino/libraries* on OS X.

To install a library manually, go to the ZIP file containing the library and uncompress it. For example, to install a library called *keypad*

in a compressed file called *keypad.zip*, you would uncompress *keypad.zip*, which expands into a folder called *keypad*, which in turn contains files like *keypad.cpp* and *keypad.h*. Once the ZIP file is expanded, you would drag the *keypad* folder into the *libraries* folder on your operating system: *sketchbook/libraries* in Linux, *My Documents\Arduino\Libraries* on Windows, and *Documents/Arduino/libraries* on OS X. Then you'd restart the Arduino application.

Libraries are listed at the start of a sketch and are easily identified because they begin with the command `#include`. Library names are surrounded by `< >` and end with `.h`, as in this code to call the Servo library:

```
-----  
#include <Servo.h>  
-----
```

Go ahead and install the libraries you'll need for the projects now to save yourself a bit of time later.

## TESTING YOUR ARDUINO: BLINKING AN LED

Let's begin our tour with the classic first Arduino project: blinking an LED (short for *light-emitting diode*, which is like a little light bulb). Not only is this the simplest way to make sure that your Arduino is working correctly, but it will also introduce you to a simple sketch. The Arduino can hold only one program at a time, so once you upload your sketch to your Arduino, that sketch will run every time the Arduino is switched on until you change it.

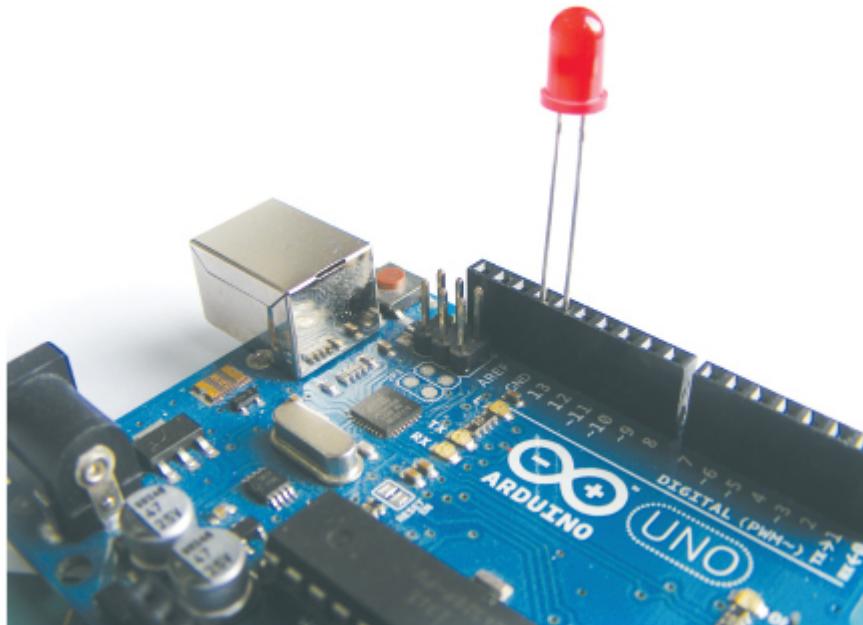
### The Build

For this project we'll use the *Blink* example sketch that comes with the IDE. The Blink program turns an LED on for 1 second and then off, repeatedly. The LED works only with current flowing in one direction, so its longer wire must connect to a positive power connection. LEDs require a *current-limiting resistor* or else the bulb may burn out. There is a built-in resistor in pin 13 of the Arduino that we'll use.

Follow these steps to set up your test:

1. Insert the longer, positive leg of the LED to pin number 13 on the Arduino, as shown in Figure 0-7. Connect the shorter, negative wire to the GND pin next to pin 13.

**FIGURE 0-7:** The *Blink* project setup



2. Connect the Arduino to your computer with the USB cable.
3. Open the Arduino IDE on your computer, then choose **File ▶ Examples ▶ Blinking LED** from the drop-down menu. The sketch will appear in the main program area of the IDE.

---

```
①// Blinking LED Project - This example code is in the public domain

② int led = 13;
③ void setup() {
④   pinMode(led, OUTPUT);
⑤ }
⑥ void loop() {
⑦   digitalWrite(led, HIGH);
⑧   delay(1000);
⑨   digitalWrite(led, LOW);
```

```
❾   delay(1000);  
❿ }
```

---

4. In the IDE, click the **Verify** button to check that the sketch is working correctly.
5. Click the **Upload** button to send the sketch to your Arduino. Running this code should make your LED flash on and off.

## Understanding the Sketch

Here's what's happening on each line of the sketch:

- ❶ This is a comment. Any line in your program starting with // is meant to be read by the user only and is ignored by the Arduino, so use this technique to enter notes and describe your code (called *commenting* your code). If a comment extends beyond one line, start the first line with /\* and end the comment with \*/. Everything in between will be ignored by the Arduino.
- ❷ This gives pin 13 the name led. Every mention of led in the sketch will refer to pin 13.
- ❸ The code between the curly brackets, {}, will run once when the program starts. The open curly bracket, {, begins the setup code.
- ❹ This tells the Arduino that pin 13 is an output pin, indicating that we want to send power to the LED from the Arduino. The closing curly bracket, }, ends the setup code.
- ❺ This creates a loop. Everything between the curly brackets, {}, after the loop() statement will run once the Arduino is powered on and then repeat until it is powered off.
- ❻ This tells the Arduino to set led (pin 13) to HIGH, which sends power to that pin. Think of it as switching the pin on. In this sketch, this turns on the LED.
- ❼ This tells the Arduino to wait for 1 second. Time on the Arduino is measured in milliseconds, so 1 second = 1,000 milliseconds.

- ❸ This tells the Arduino to set `led` (pin 13) to `LOW`, which removes power and switches off the pin. This turns off the LED.
- ❹ Again the Arduino is told to wait for 1 second.
- ❺ This closing curly bracket ends the loop. All code after the initial `setup` must be enclosed within curly brackets. A missing bracket can easily be overlooked and is a common cause of errors that will prevent your sketch from compiling correctly. After this curly bracket, the code goes back to the start of the loop at ❸.

Now that you've tested your Arduino and understand how a sketch works and how to upload it, we'll take a look at the components you'll need to carry out all of the projects in this book. "Components" on page 238 has more details about each component, what it looks like, and what it does.

## PROJECT COMPONENT LIST

This is a complete list of the items you'll need in order to complete the projects in this book. The most important part, of course, is the Arduino board itself, and all projects use the Arduino Uno R3 version. Only the official boards are named Arduino, but you'll find compatible clone boards from companies like SlicMicro, Sainsmart, and Adafruit. (You'll find a list of official suppliers at <http://arduino.cc/en/Main/Buy/>.)

You can buy each item individually, but I suggest buying an electronics hobby starter kit or Arduino kit, which will provide you with several of the items here. See the "Retailer List" on page 249 for a list of suggested suppliers. Alternatively, each project begins with a list of the required parts, so you can flip to a project that interests you and obtain just those components if you'd like.

- 1 Arduino Uno R3 (or compatible)
- 1 9V battery pack with 2.1 mm jack for 6 AA batteries
- 1 9V battery snap and battery
- 3 breadboards: 1 full-size, 1 half-size, 1 mini

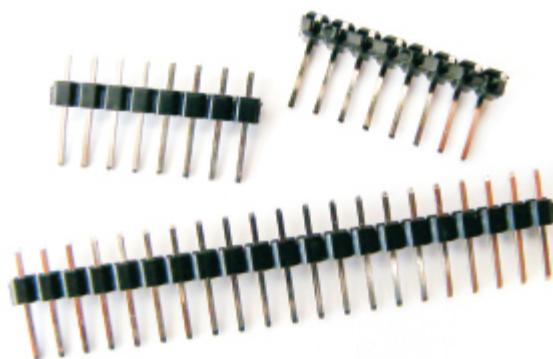
50 male-to-male jumper wires  
10 female-to-male jumper wires  
Solid-core wire  
9 220-ohm resistors  
4 10k-ohm resistors  
8 1k-ohm resistors  
40 5 mm LEDs in red, green, yellow, blue (10 of each)  
1 RGB common-cathode LED  
1 RGB LED strip (WS2812B 5V 32-LED strip)  
1 Adafruit NeoPixel ring with 16 RGB LEDs  
1 HMC5883L three-axis sensor  
2 50k-ohm potentiometers  
1 10k-ohm potentiometer  
8 momentary tactile pushbuttons  
1 seven-segment, single-digit common-cathode LED  
1 piezo sounder  
1 3.5 mm female headphone jack  
1 Tower Pro SG90 9g servomotor  
1 photoresistor (*light-dependent resistor*, or *LDR*)  
1 28BYJ-48 stepper motor with ULN2003 driver module  
1 HC-SR04 ultrasonic sensor  
1 3×4 membrane keypad  
1 LM35 temperature sensor  
1 12V mini computer cooling fan  
1 5V single-channel relay module  
1 HD44780 16×2 LCD screen  
1 Nokia 5110 LCD screen  
1 serial LCD screen module

- 1 OLED monochrome screen (128x64)
- 1 8x8 LED Maxim 7219 matrix module
- 1 Keyes MQ3 alcohol sensor module
- 1 optical fingerprint sensor (ZFM-20 series)
- 1 L293d motor shield
- 1 robot chassis kit, including two DC motors and wheels, center wheel, base, and fittings
- 1 Ethernet shield W5100 LAN expansion board
- 1 Ethernet cable
- 1 WLToys V959-18 Water Jet Pistol
- 1 HC-06 Bluetooth module
- 1 Ublox NEO-6M GPS module aircraft flight controller and antenna

## QUICK SOLDERING GUIDE

The majority of the projects in this book do not requiring soldering, but there are a few components that may come with their header pins (Figure 0-8) unattached for ease of transport. Header pins come in strips that can be easily snapped to the size needed.

**FIGURE 0-8:** Header pins



For example, the GPS module used in Project 25 doesn't come with the pins attached, so I'll explain how to solder those in place. A general-

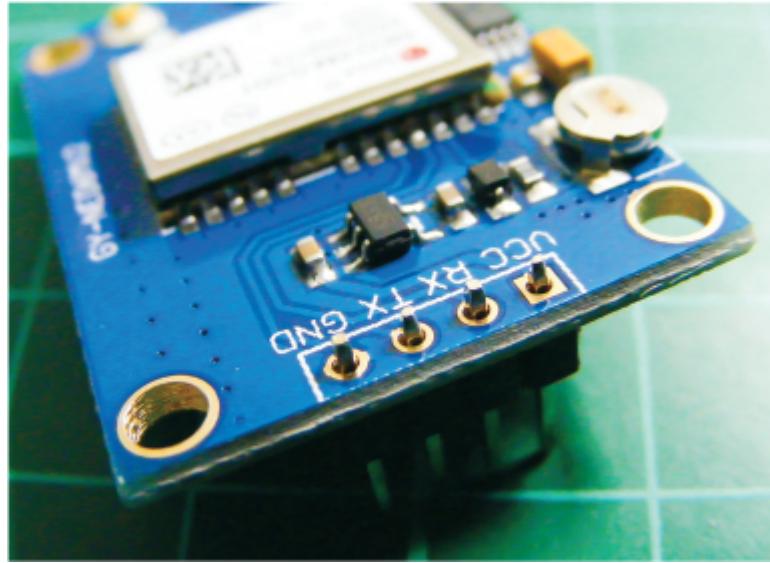
purpose, 30-watt soldering iron with a fine tip should meet your needs. It is worthwhile to buy a kit that includes a soldering iron, stand, and solder (Figure 0-9).

**FIGURE 0-9:** Soldering iron



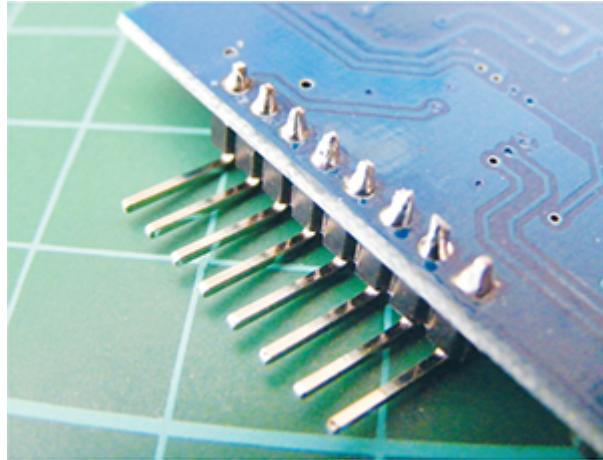
1. Plug in your soldering iron and wait at least 5 minutes for it to reach operating temperature.
2. To solder, break off a strip of header pins with the number you need. Insert them into the module as shown in Figure 0-10.

**FIGURE 0-10:** Insert the header pins into the module.



3. Now solder the pins in place, starting with the leftmost pin. Hold the heated tip of the soldering iron to both the pin and module contact at the same time. You only need to hold it there for about 2 seconds. While holding the iron in place, add solder to the area; the solder should melt and flow and create a *join*. Note that you do not apply solder directly to the iron, only to the joint you are soldering. Quickly remove both the iron and solder—more than a couple of seconds of contact could damage your components.
4. A good solder joint should look like a shiny cone (Figure 0-11). With a little bit of practice, you will be able to solder cleanly in no time at all.

**FIGURE 0-11:** Solder joins should look like this.



## Safety First

Soldering irons get very, very hot and should be used with extreme care under adult supervision. Here are a few safety tips:

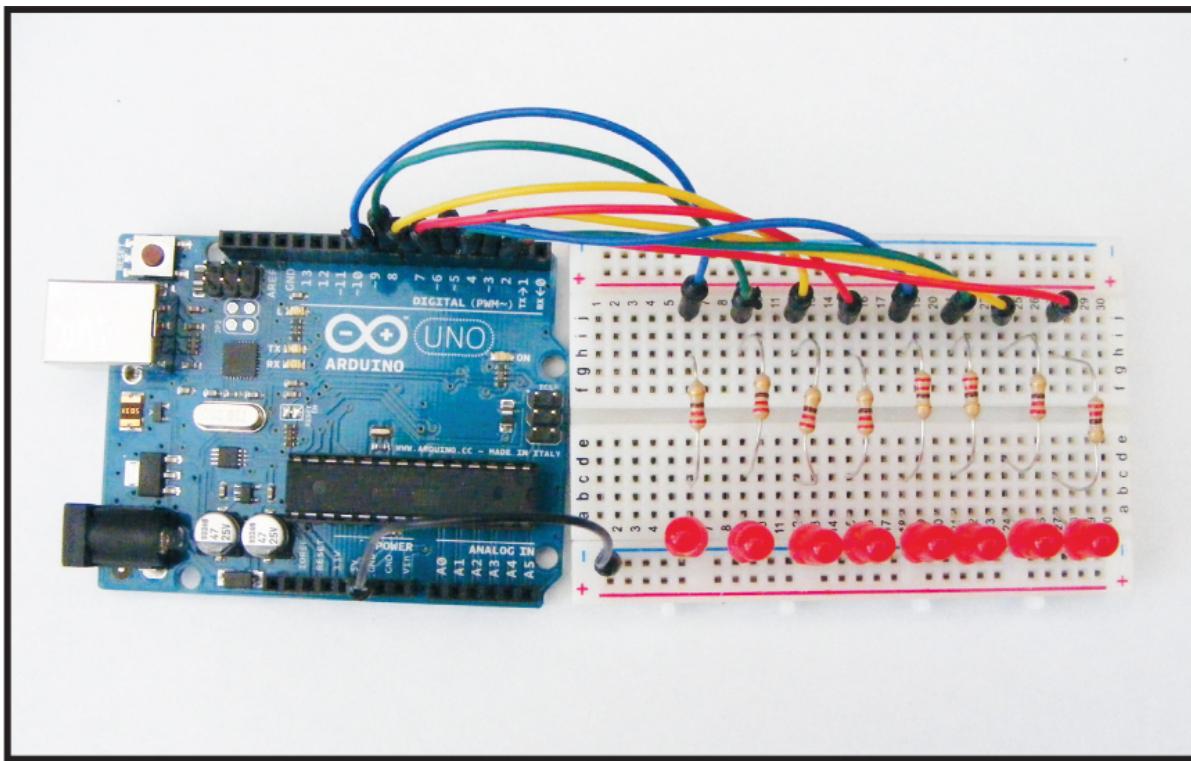
- Be sure to use a stand and never lay a hot soldering iron down on a table.
- Solder in a well-ventilated room. The fumes released from melting solder can be harmful.
- Keep flammable materials away from your work area.
- Keep equipment out of reach of children.
- Wear eye protection.
- Wait for a soldering iron to cool down completely before storing it.

# LEDs

1

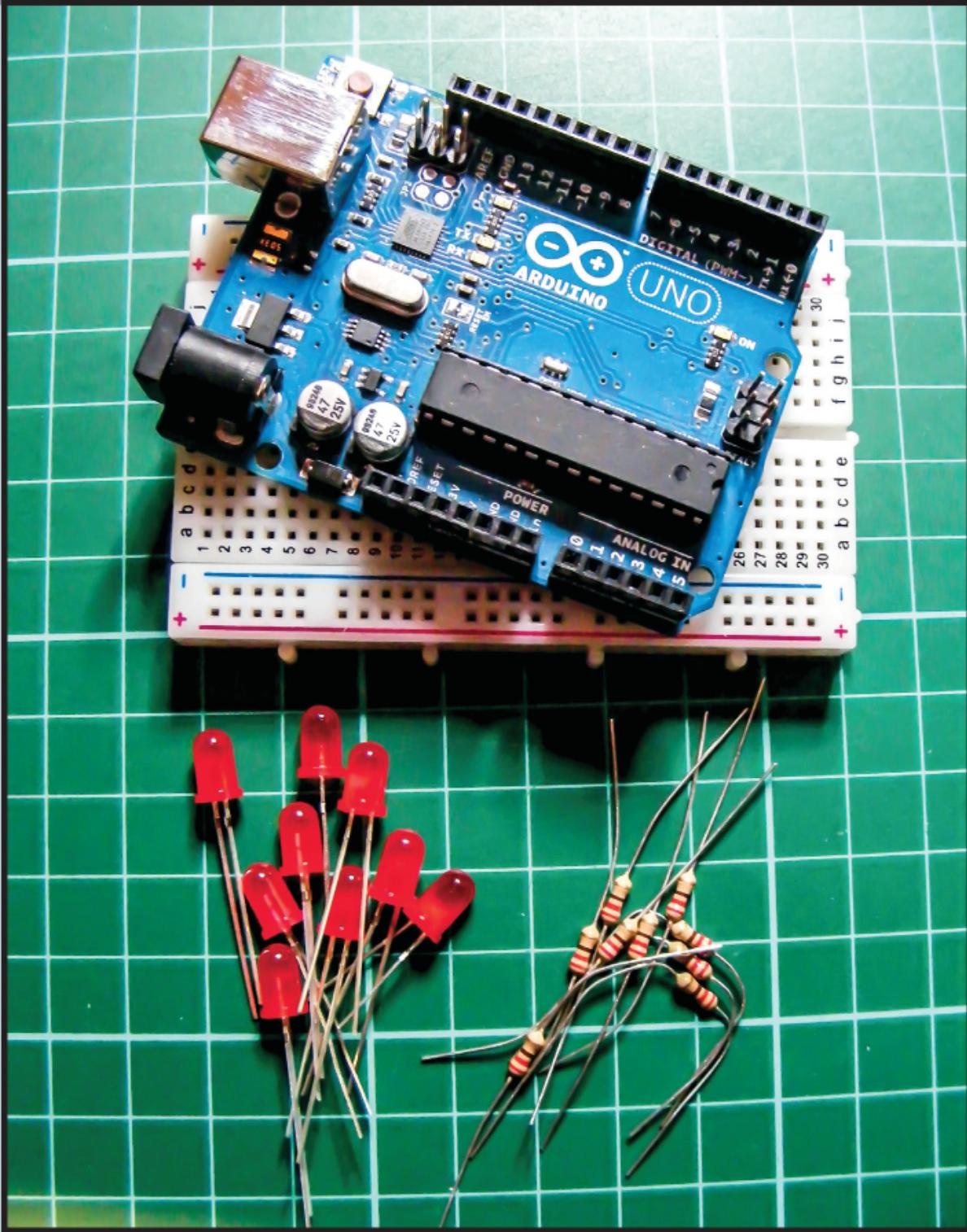
# LED Light Bar

In this project we'll flash a row of LEDs back and forth in sequence, sort of like KITT from the 1980s TV series *Knight Rider*.



COST: \$

TIME: 15 MINUTES



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**8 LEDs**

**8 220-ohm resistors**

## HOW IT WORKS

An LED emits light when a small current is passed through it. LEDs are *polarized*, which means one side is positive and one side is negative. This is because the LED will work only with current flowing in one direction, from positive to negative. The longer leg of the LED is positive and must connect to a positive power connection. The Arduino sketch controls the sequence of flashes.

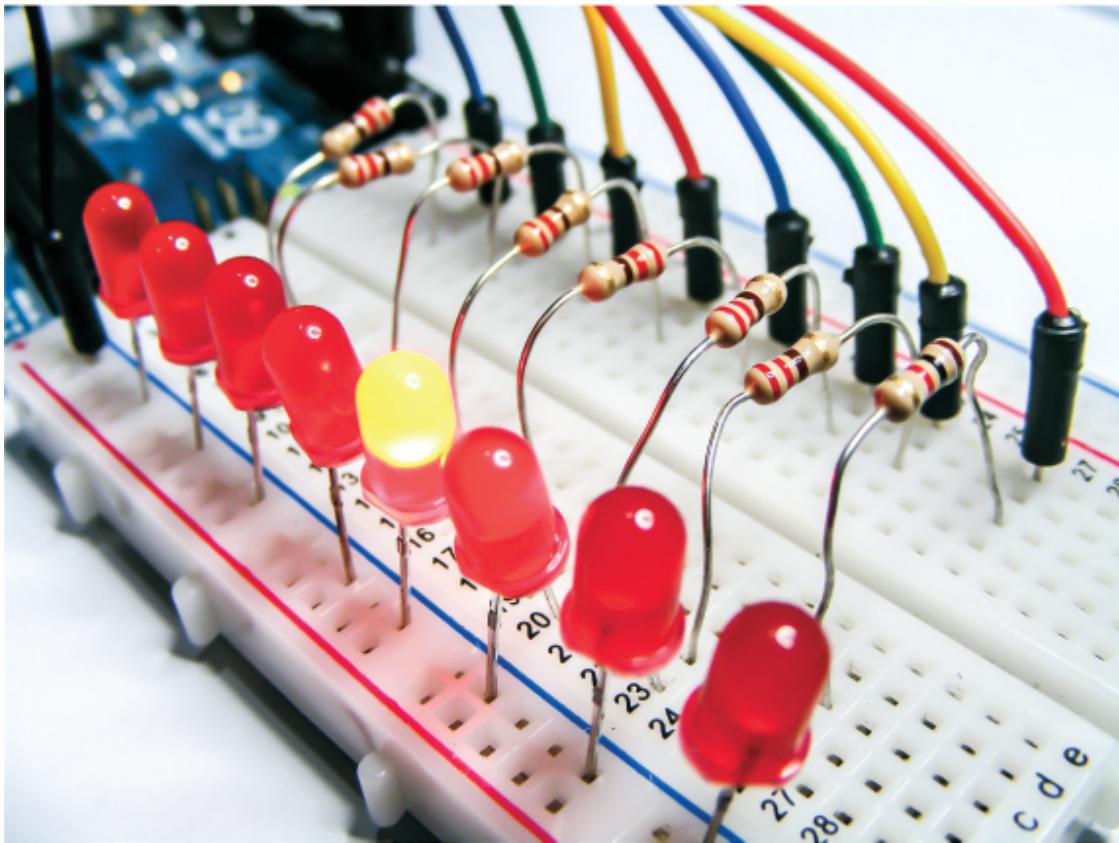
LEDs are delicate parts, requiring only a small amount of voltage to light up—smaller than the voltage the Arduino provides. To prevent the LEDs from being overloaded with voltage and burning out, we use *resistors*, which limit the amount of voltage passing through them to the LED on the other end.

You can change the color of your LEDs and use this light bar to decorate a car, scooter, bike, picture frame, subwoofer, or almost anything else you choose. You can add up to 10 LEDs on the Uno before you run out of pins.

## THE BUILD

1. Insert the LEDs into the breadboard with their shorter, negative legs in the GND rail at the top of your breadboard. Then connect this rail to GND on the Arduino, as shown in Figure 1-1.

**FIGURE 1-1:** The LEDs flash back and forth in sequence. The short leg of the LED is in the GND rail of the breadboard, and the long leg is connected to the Arduino via a resistor.

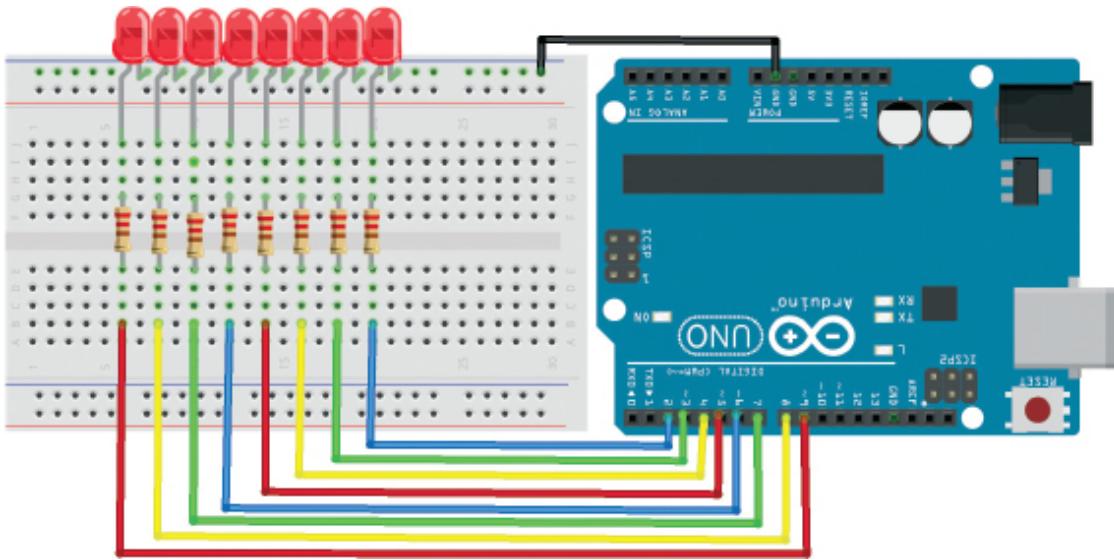


2. Connect the LEDs in sequence to Arduino digital pins 2–9, as shown in the following circuit diagram. Place a 220-ohm resistor between each LED and digital pin, ensuring that the resistors bridge the center divide in the breadboard.

LEDS	ARDUINO
Positive legs	Pins 2–9 via resistor
Negative legs	GND

3. Check your setup against Figure 1-2, and then upload the code in “The Sketch” below.

**FIGURE 1-2:** The circuit diagram for the LED light bar



## THE SKETCH

The sketch sets the pins connected to the LEDs as outputs, and then defines a function to turn all the LEDs off at the same time. This function is called in the loop cycle to turn the LEDs off, and then the LEDs are turned on one at a time—with a 200-millisecond delay between each one—to create a sweeping effect. Another loop sends the sequence back the other way.

---

```
// Used with kind permission from
// Warwick A Smith, startingelectronics.com
// Knight Rider display on eight LEDs

void setup() {
  for (int i = 2; i < 10; i++) { // Choose pins 2-9
    pinMode(i, OUTPUT); // Set the pins as outputs
  }
}

// Define function to turn off all LEDs at the same time
void allLEDsOff(void) {
  for (int i = 2; i < 10; i++) {
    digitalWrite(i, LOW);
  }
}

// Switch on LEDs in sequence from left to right
void loop() {
  for (int i = 2; i < 9; i++) { // Run loop once for each LED
    allLEDsOff(); // Turn off all LEDs
    // Turn on LED at position i
  }
}
```

```
    digitalWrite(i, HIGH); // Turn on current LED
    delay(200); // Delay of 200 ms,
                 // then repeat loop to move on to next LED
}
for (int i = 9; i > 2; i--) { // Light LEDs from right to left
    allLEDsOff();
    digitalWrite(i, HIGH);
    delay(200);
}
}
```

---

## TROUBLESHOOTING

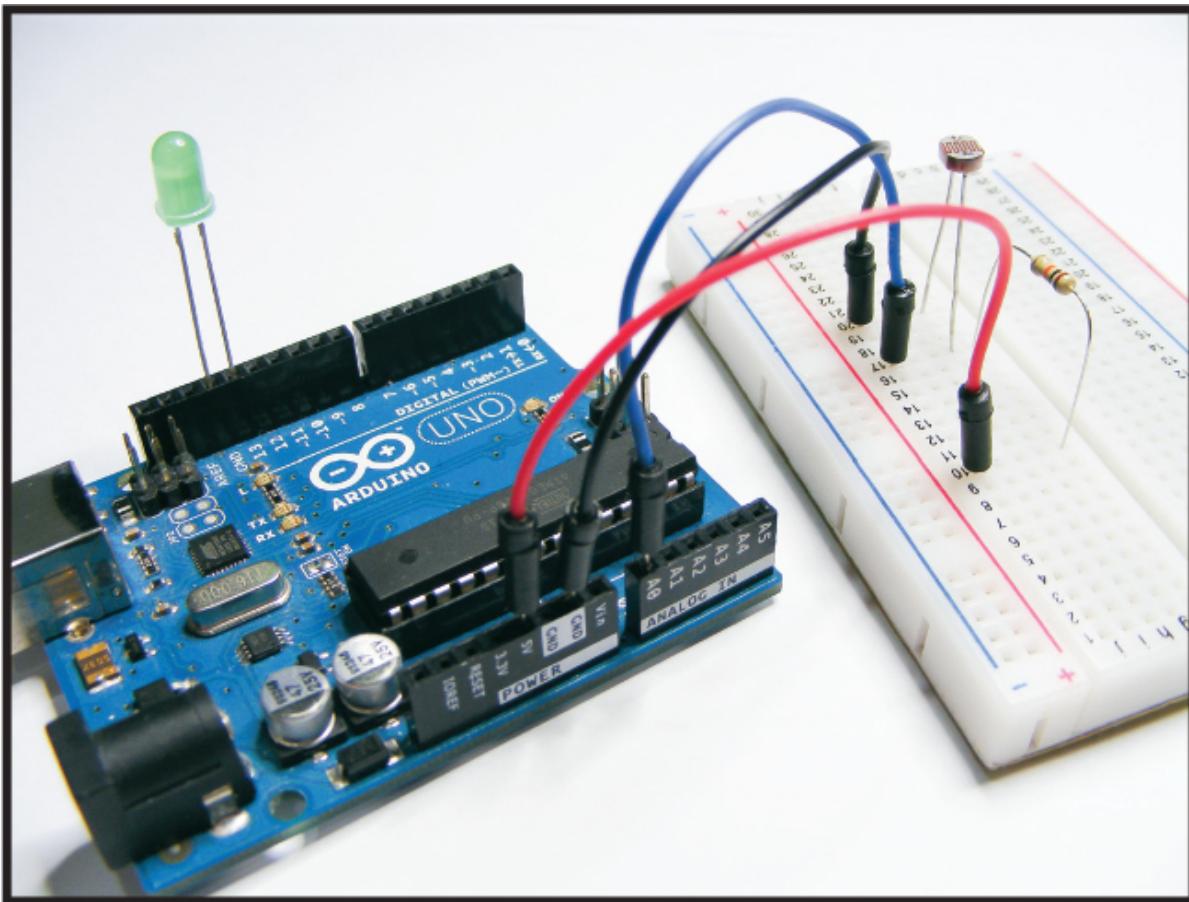
**Q.** *The code compiles, but some or all of the LEDs do not light up as expected.*

- If none of the LEDs light, make sure you've connected the GND wire from the Arduino to the correct breadboard power rail and that the Arduino has power connected.
- If only some LEDs light, check that the LEDs are inserted the correct way, with the longer wire to positive power and the shorter wire to GND. Because LEDs are polarized, they must be connected the correct way. Check that the resistors are inserted fully and lined up in the same row as the corresponding LED leg.
- Make sure the LEDs are connected to the Arduino pins defined in “The Sketch” on page 19. The first part of the sketch defines pins 2–9 as outputs, so these are the pins you should use.
- If an LED still fails to light, it may have burnt out or be faulty. An easy way to check is to swap the LED with another in the sequence and see if that resolves the issue. If you find that the LED works in another position, it means the resistor is either faulty or not inserted fully. Depending on the outcome, replace the LED or resistor with a functioning component.

2

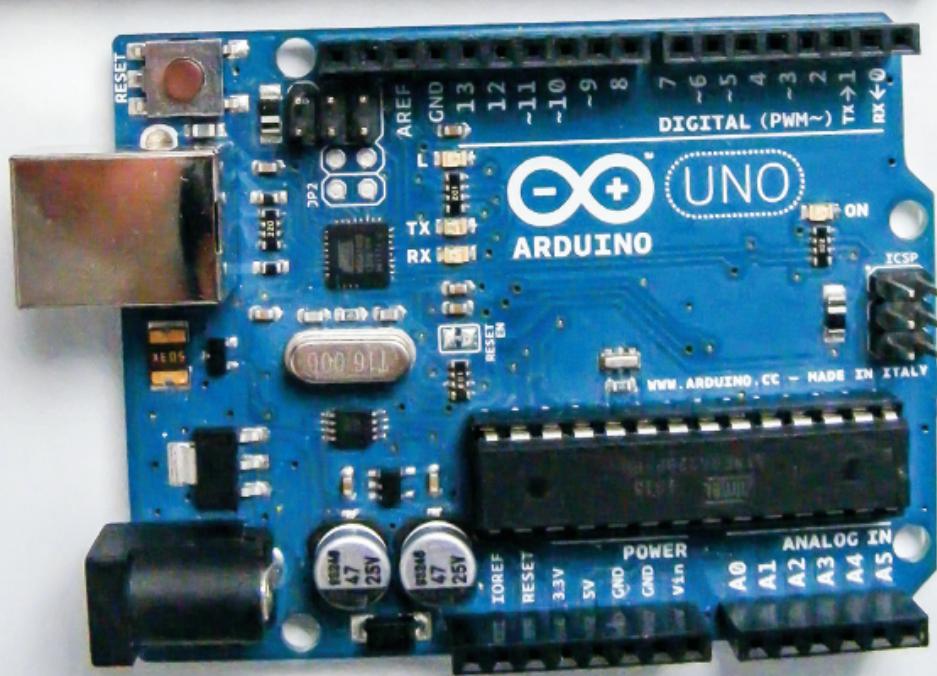
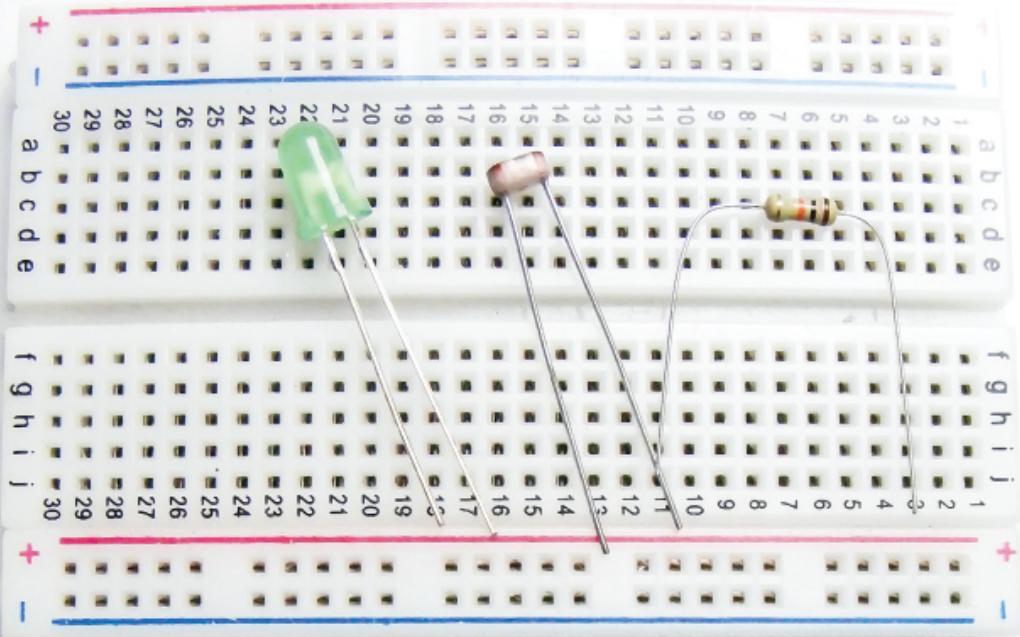
## Light-Activated Night-Light

This project is a simple test of a photoresistor's functionality: we'll create a night light that gets brighter depending on the amount of light detected.



**COST: \$**

**TIME: 10 MINUTES**



**PARTS REQUIRED**

**Arduino board**

**Breadboard**

**Jumper wires**

**Photoresistor**

**LED**

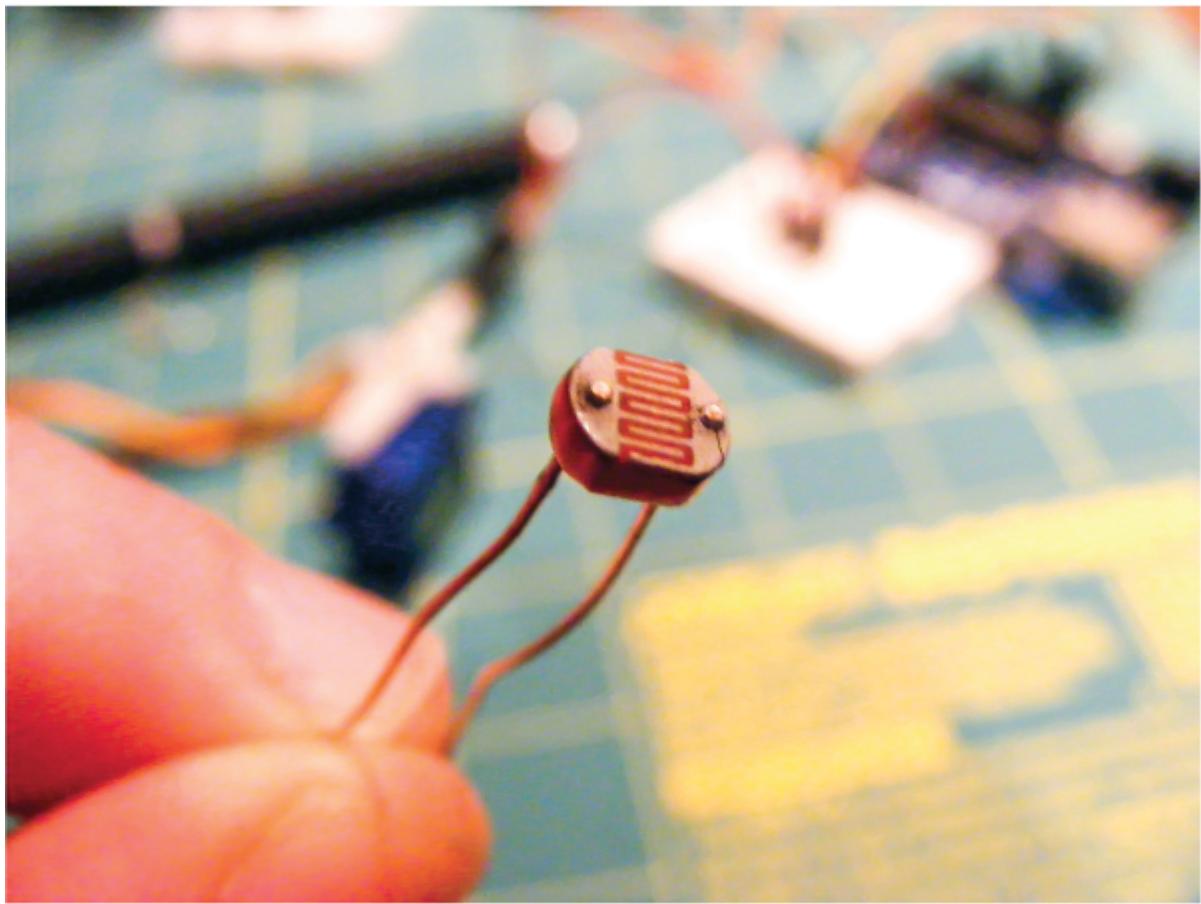
**10k-ohm resistor**

## HOW IT WORKS

A *photoresistor* is a variable resistor that reacts to light; the less light that shines on it, the higher the resistance it provides. This resistance value varies the voltage that's sent to the input pin of the Arduino, which in turn sends that voltage value to the output pin as the power level of the LED, so in low light the LED will be bright. There are different styles of photoresistors, but they usually have a small, clear, oval head with wavy lines (see Figure 2-1). Photoresistors do not have polarity, so it doesn't matter which way you connect the legs.

The principles at work here are similar to those of a child's night-light. You can use a photoresistor to control more than just LEDs, as we'll see in upcoming chapters. Since we only have two power and GND connections, we won't be using the breadboard power rails here.

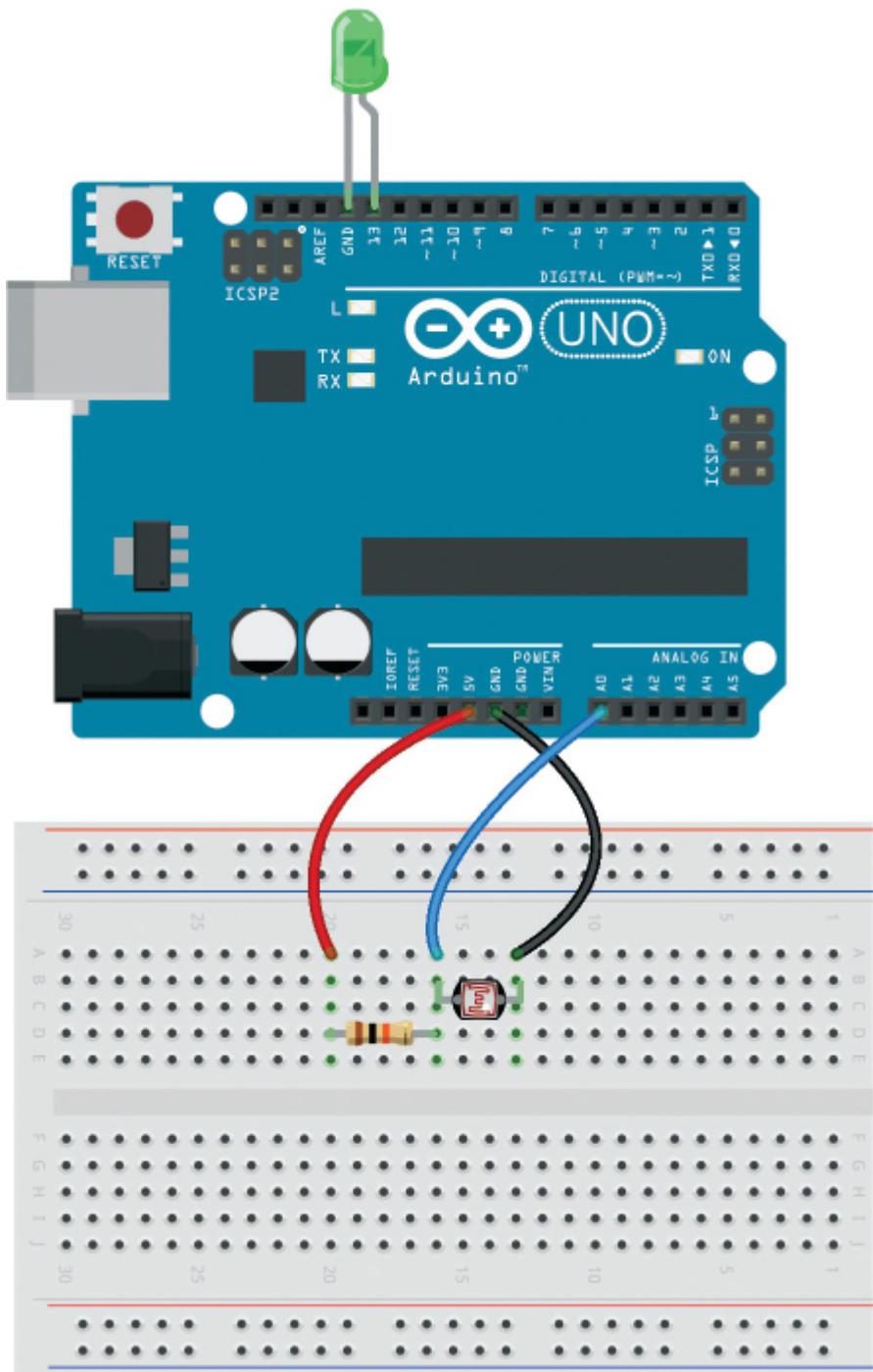
**FIGURE 2-1:** A photoresistor



## THE BUILD

1. Place your photoresistor in the breadboard, connecting one leg to GND directly on the Arduino and the other leg to Arduino A0.
2. Connect one leg of the 10k-ohm resistor to +5V, and connect the other leg to the A0 photoresistor leg, as shown in the circuit diagram in Figure 2-2.

**FIGURE 2-2:** The circuit diagram for the light-activated LED



3. Insert the longer, positive leg of the LED directly into pin 13 on the Arduino and the shorter, negative leg directly into Arduino GND. We would normally use a resistor to limit the current to an LED, but we don't need one here because pin 13 on the Arduino has one built in.

4. Upload the code in “The Sketch” below.

## THE SKETCH

The sketch first connects the photoresistor to Arduino pin A0 as our INPUT and the LED to pin 13 as our OUTPUT. We run the serial communication with `Serial.begin(9600)`, which (when your Arduino is connected to your PC) will send information to the Arduino’s Serial Monitor. This means the resistance value of the photoresistor will be displayed in the Serial Monitor on your computer, as shown in Figure 2-3.

**FIGURE 2-3:** The Serial Monitor will display the resistance of the photoresistor.

The screenshot shows the Arduino IDE interface. On the left is the code editor window titled "Project\_2\_Light\_activated\_LED | Arduino 1.6.5". The code is as follows:

```
Project_2_Light_activated_LED.ino
int lightPin = A0; //pin connected to the photoresistor
int ledPin=13;// pin connected to the LED
void setup()
{
  Serial.begin(9600); // Begin serial communication
  pinMode( ledPin, OUTPUT ); // Setting the LED pin as an output
}
void loop() // This loop reads the analog pin value and sends that
// LED as an output
{
  Serial.println(analogRead(lightPin)); //Read the value of the phot
  analogWrite(ledPin, analogRead(lightPin)/4); //send the value to

  delay(10); //short delay before the sequence loops again
}
```

Below the code editor, a status bar indicates "Done uploading." and memory usage: "Global variables use 204 bytes (9%) of dynamic memory, leaving 1,044 bytes for local variables. Maximum is 2,048 bytes."

On the right is the Serial Monitor window titled "COM4 (Arduino/Genuino Uno)". It shows a continuous stream of numerical values representing the analog read from pin A0, ranging from 426 to 488. The monitor settings are "No line ending" and "9600 baud".

The loop reads the photoresistor’s analog value and sends it to the LED as a voltage value. The A0 pin can read 1,024 values, which means there are 1,024 possible brightness levels for the LED. Minuscule

changes between this many levels aren't very visible, so we divide that number by 4 to scale down to only 256 values, making it easier to detect when there is a change in voltage to the LED.

---

```
int lightPin = A0; // Pin connected to the photoresistor
int ledPin = 13; // Pin connected to the LED
void setup() {
    Serial.begin(9600); // Begin serial communication
    pinMode(ledPin, OUTPUT); // Setting the LED pin as an output
}

// This loop reads the analog pin value and
// sends that to the LED as an output
void loop() {
    // Read the value of the photoresistor
    Serial.println(analogRead(lightPin));
    // Write the value to the Serial Monitor
    // Send the value to the ledPin and divide by 4
    analogWrite(ledPin, analogRead(lightPin) / 4);
    delay(10); // Short delay before the sequence loops again
}
```

---

## TROUBLESHOOTING

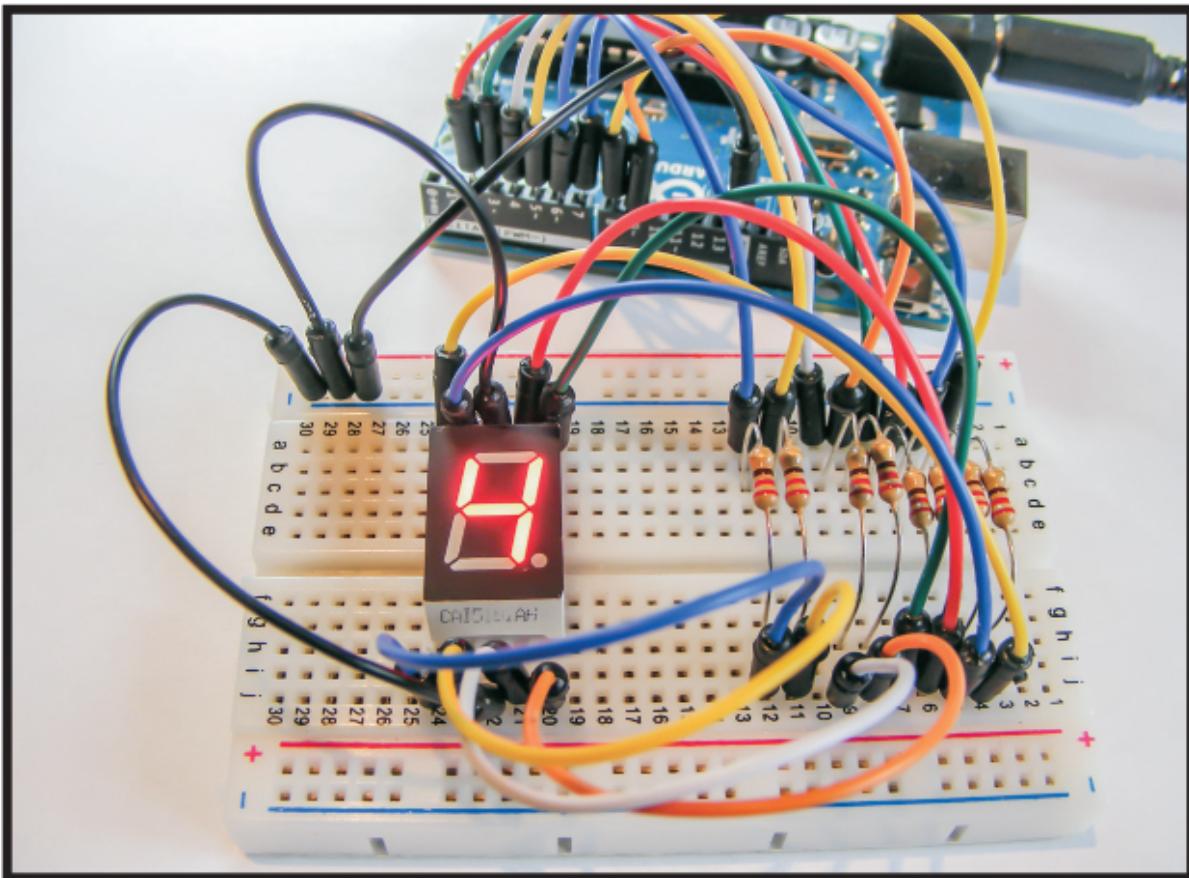
**Q.** *The code compiles, but the LED does not light when it's dark.*

- Make sure that the LED is inserted with the long, positive leg in pin 13 and the short, negative leg in GND next to it.
- Make sure the photoresistor is connected to Arduino A0 as shown in the circuit diagram in Figure 2-2. Open the Serial Monitor to see if there's a reading. If you're getting a reading but the LED doesn't light, the LED may be faulty, so try replacing it with another one.

3

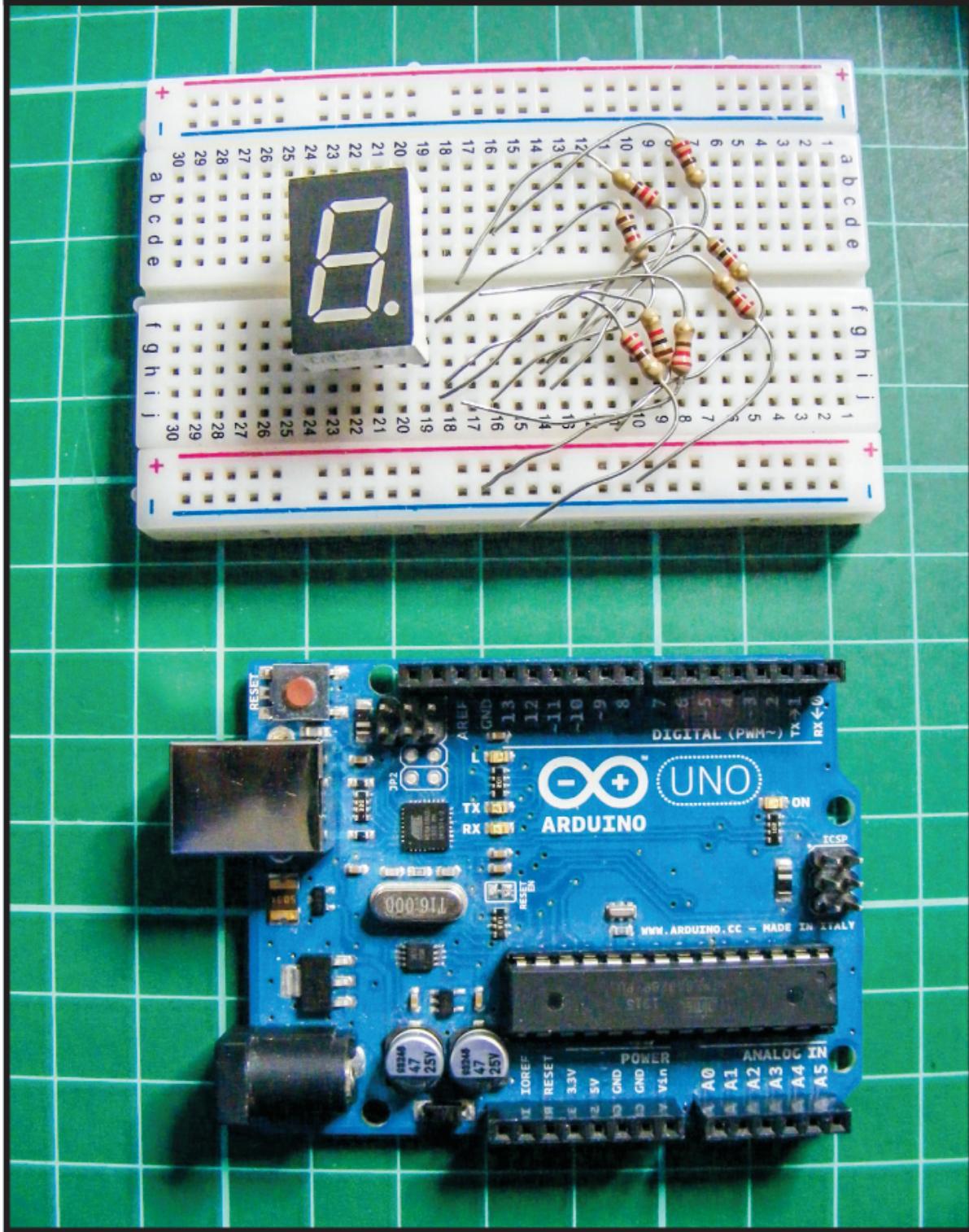
# Seven-Segment LED Count Down Timer

In this project we'll create a simple timer that counts down from 9 to 0. This can be used in any number of useful projects!



**COST: \$**

TIME: 30 MINUTES



# PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**Seven-segment, single-digit common-cathode LED**

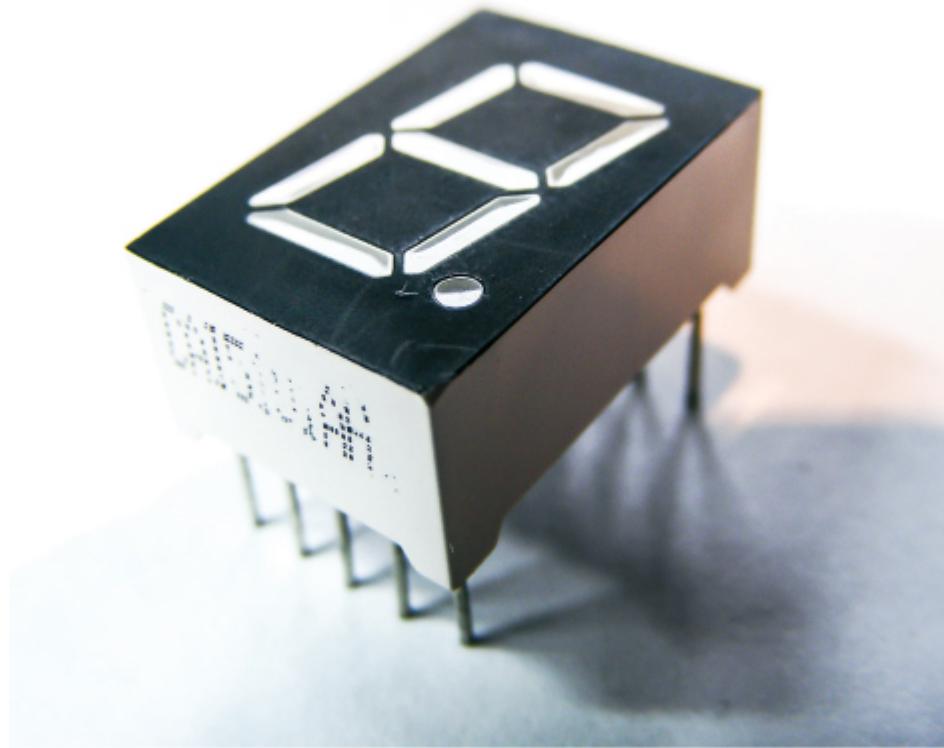
**8 220-ohm resistors**

## HOW IT WORKS

A seven-segment LED display shows a single digit or character using LED segments. Each segment is an individual LED, and by controlling which segments are lit at any time, we can display numeric values.

We're using a single-digit display in this project, shown in Figure 3-1, but there are also two-, three-, four-, and eight-digit variations available.

**FIGURE 3-1:** A seven-segment LED



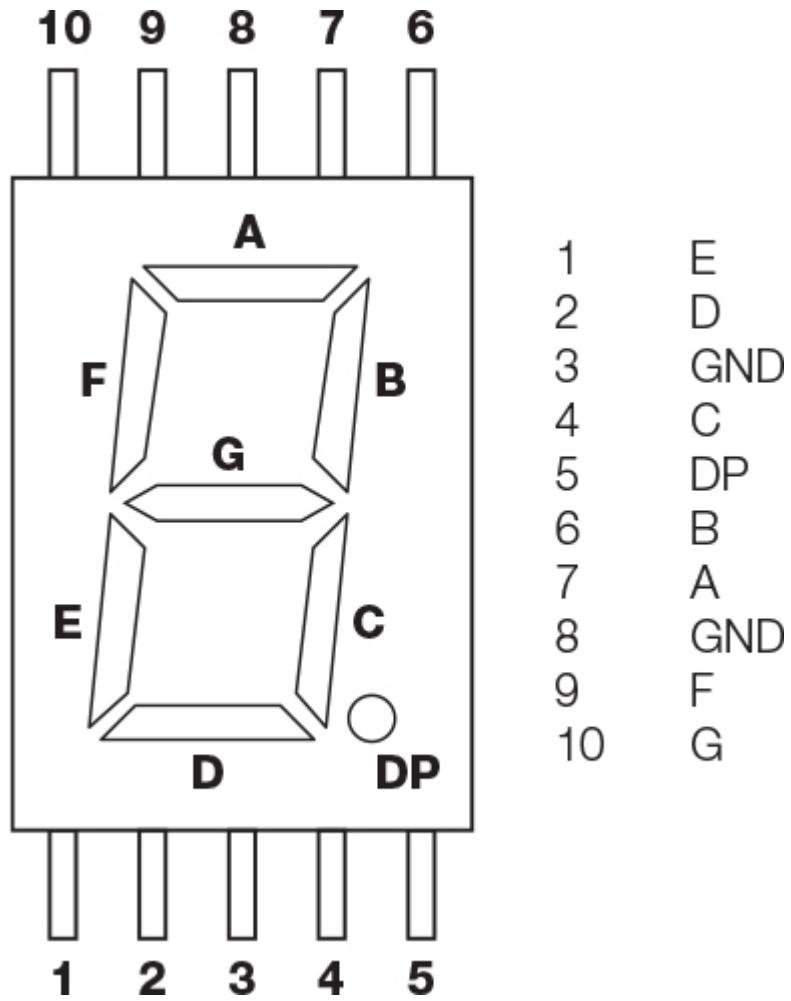
**NOTE**

*The cathode of a device is the negative connection, usually indicated with a minus sign (-) and sometimes referred to as ground (abbreviated GND). It is connected to negative power. The anode of a device is the positive connection, usually indicated with a plus sign (+) and connected to positive power.*

This project will create a simple timer to count down from 9 to 0. The seven-segment LED has 10 pins. Seven pins control the seven LEDs that light up to form each digit, and the eighth pin controls the decimal point. The other two pins are the common-cathode (-) or common-anode (+) pins, which add power to the project. Our seven-segment LED is common cathode, meaning one side of each LED needs to connect to ground. It's important to note that the code will work only with a common-cathode LED. If you have a common-anode LED you want to use, check the troubleshooting section at the end of this chapter before uploading the sketch. Each LED segment requires a resistor to limit the current; otherwise, it will burn out.

The pins are labeled with a letter, as shown in Figure 3-2. The numbered pins control the segments as shown on the right. The Arduino creates the number by turning the LEDs off or on in different combinations.

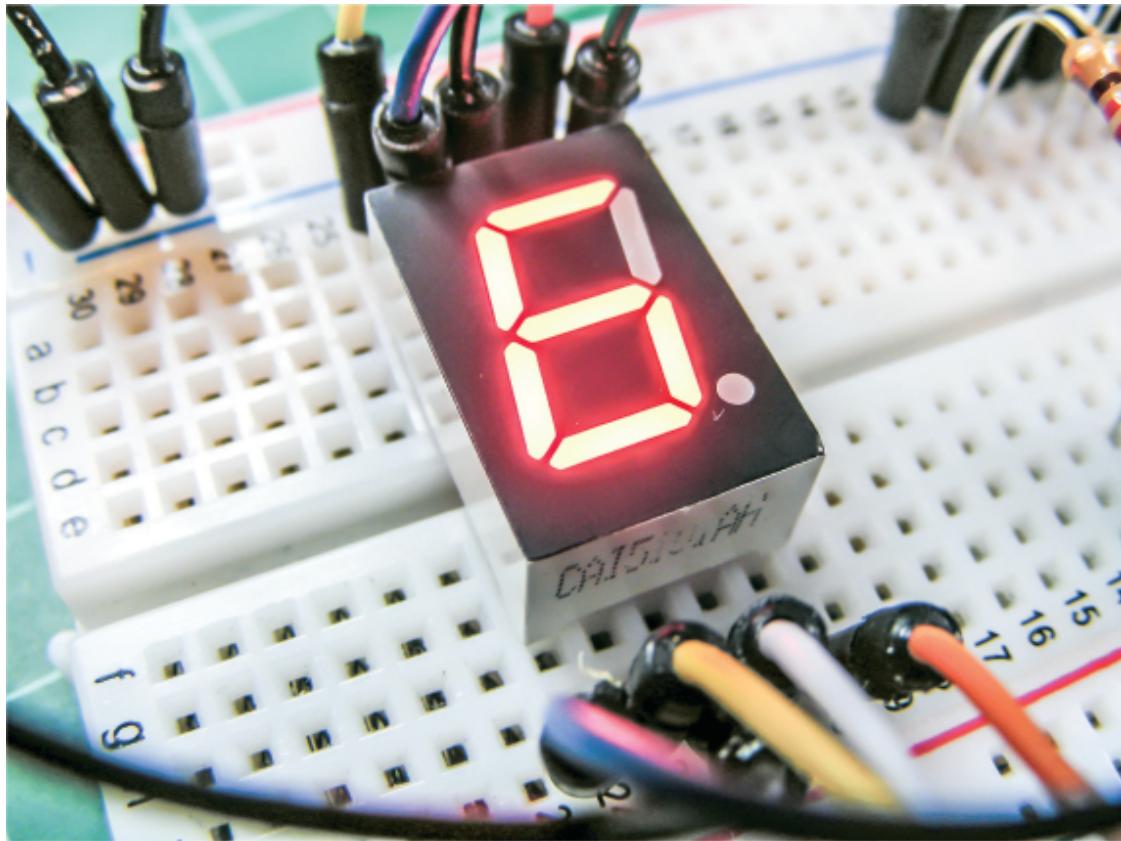
**FIGURE 3-2:** A typical pin layout for a seven-segment LED



## THE BUILD

1. Place the seven-segment display in a breadboard as shown in Figure 3-3, making sure the pins straddle either side of the center break. Connect LED pins 3 and 8 to the GND rail.

**FIGURE 3-3:** The seven-segment LED pins should straddle the center break of the breadboard.

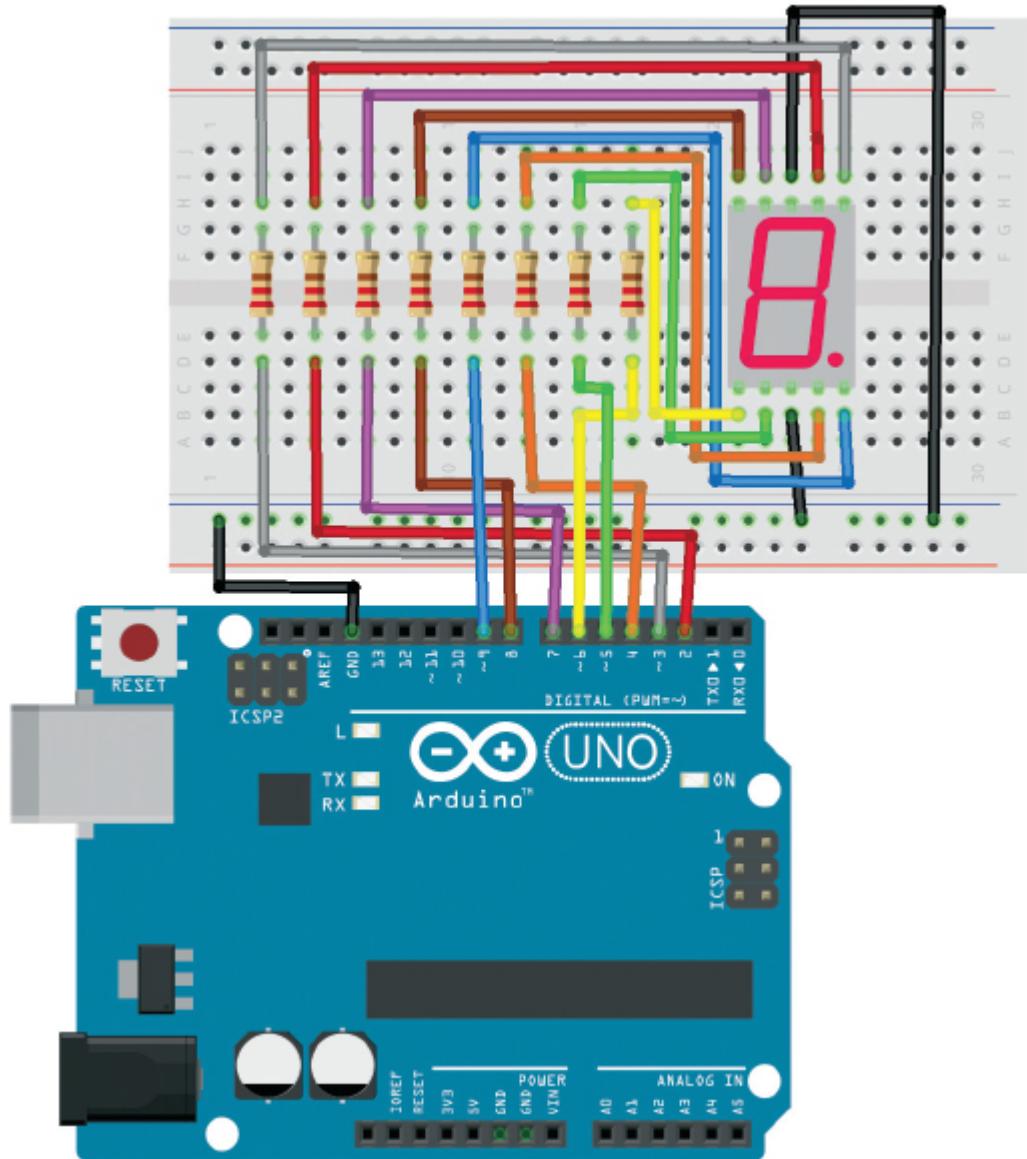


2. Connect LED pins 1, 2, 4, 5, 6, 7, and 9 as shown in the following table, remembering to insert a 220-ohm resistor between the LED and the Arduino connection. It's important that the resistors straddle the center break on the breadboard, as shown in the circuit diagram in Figure 3-4.

ARDUINO	SEVEN-SEGMENT LED SECTION	SEVEN-SEGMENT LED DISPLAY
Pin 2	A	Pin 7
Pin 3	B	Pin 6
Pin 4	C	Pin 4
Pin 5	D	Pin 2

ARDUINO	SEVEN-SEGMENT LED SECTION	SEVEN-SEGMENT LED DISPLAY
Pin 6	E	Pin 1
Pin 7	F	Pin 9
Pin 8	G	Pin 10
Pin 9	DP	Pin 5

**FIGURE 3-4:** The circuit diagram for the seven-segment LED countdown timer



3. Upload the code in “The Sketch” on page 32.

## THE SKETCH

The sketch starts by defining the digits 0 to 9 as combinations of off (0) and on (1) LEDs. The pins controlling the LEDs are set as output, so they can set their corresponding LEDs to either HIGH or LOW. The combination of 1 and 0 values lights up to form the digit.

Note that these patterns are for common-cathode displays. For common-anode displays, change each 1 to 0 and each 0 to 1. In the code,

a value of 1 means the LED is on, and 0 means the LED is off.

---

```
// Arduino seven-segment display example software
// http://hacktronics.com/Tutorials/arduino-and-7-segment-led.html
// License: http://www.opensource.org/licenses/mit-license.php

// Define the LEDs to be lit to create a number
byte seven_seg_digits[10][7] = { { 1, 1, 1, 1, 1, 1, 0 }, // = 0
{ 0, 1, 1, 0, 0, 0, 0 }, // = 1
{ 1, 1, 0, 1, 1, 0, 1 }, // = 2
{ 1, 1, 1, 1, 0, 0, 1 }, // = 3
{ 0, 1, 1, 0, 0, 1, 1 }, // = 4
{ 1, 0, 1, 1, 0, 1, 1 }, // = 5
{ 1, 0, 1, 1, 1, 1, 1 }, // = 6
{ 1, 1, 1, 0, 0, 0, 0 }, // = 7
{ 1, 1, 1, 1, 1, 1, 1 }, // = 8
{ 1, 1, 1, 0, 0, 1, 1 } // = 9
};

// Set the seven-segment LED pins as output
void setup() {
    pinMode(2, OUTPUT);
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
    pinMode(7, OUTPUT);
    pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
    writeDot(0); // Start with the decimal point off
}

void writeDot(byte dot) {
    digitalWrite(9, dot);
}

void sevenSegWrite(byte digit) {
    byte pin = 2;
    for (byte segCount = 0; segCount < 7; ++segCount) {
        digitalWrite(pin, seven_seg_digits[digit][segCount]);
        ++pin;
    }
}
void loop() {
    for (byte count = 10; count > 0; --count) { // Start the countdown
        delay(1000); // 1 second between each digit
        sevenSegWrite(count - 1); // Counting down by 1
    }
    delay(4000);
}
```

---

## TROUBLESHOOTING

**Q.** *Some LED segments do not light up.*

Check that the LEDs' wires are inserted securely and line up with the resistors on the breadboard.

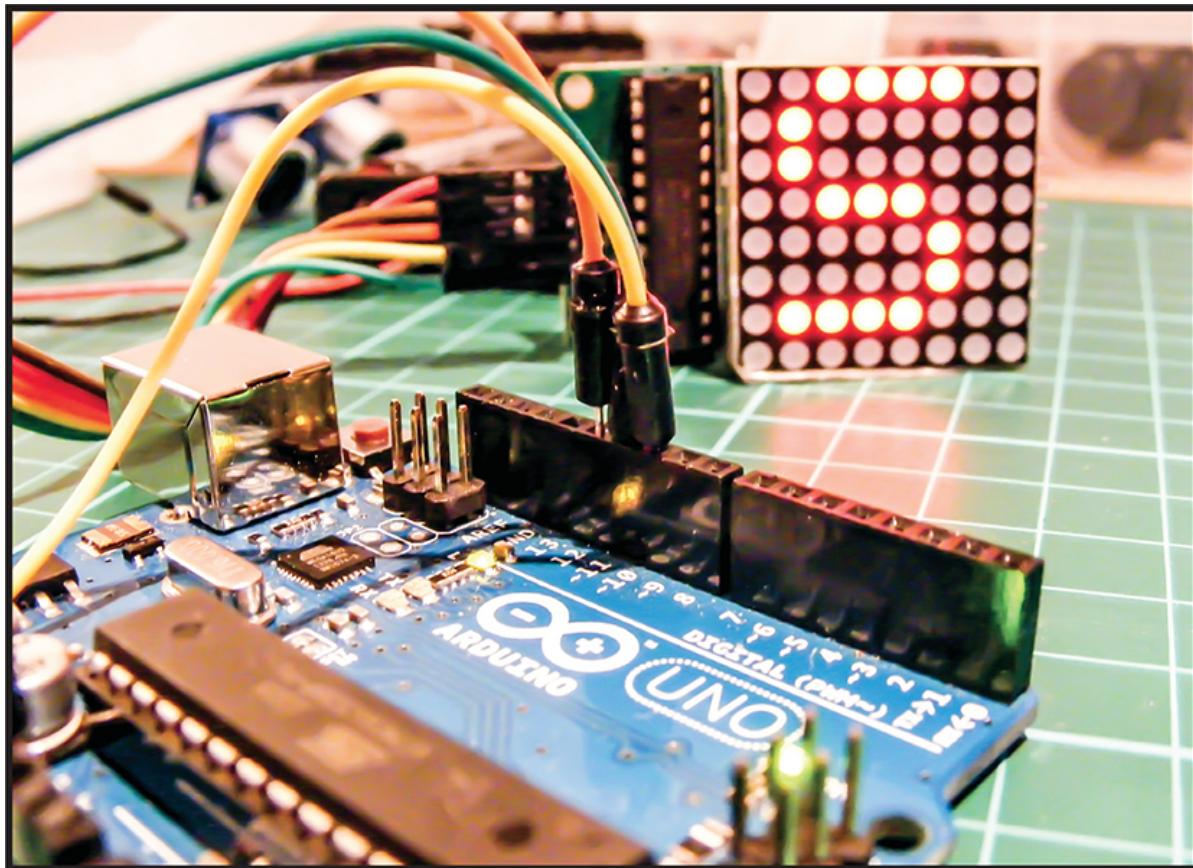
**Q.** *The display is not showing numbers correctly and looks erratic.*

- Recheck that your wiring matches the diagrams as shown, as it's easy to insert some wires in the wrong place.
- If all wiring is in the correct place and the timer's still not working, the configuration of your seven-segment LED may be different from the one used here. Check the data sheet for your part and use that to direct your circuit along with the seven-segment pin table. You can also check which pin corresponds to each LED by connecting it up: attach the GND pin of the seven-segment LED to the negative end of a battery; connect a jumper wire to the positive end of the battery, via a 220-ohm resistor; and touch each pin in turn to light the segments individually. Note which segment each pin lights up.
- Remember, this wiring is for a seven-segment, common-cathode LED; for common-anode displays, change each 1 to 0 and each 0 to 1 in the sketch.

# 4

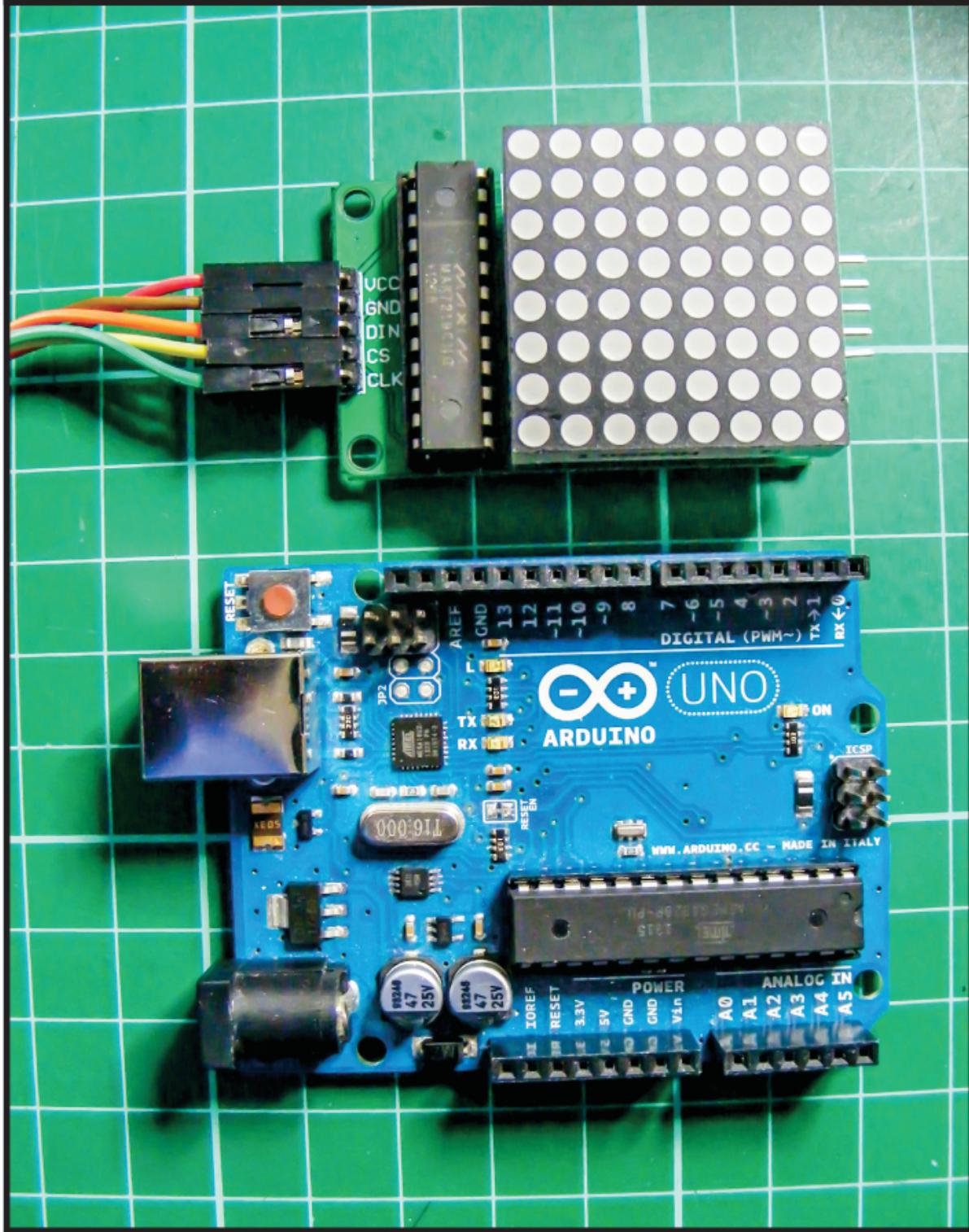
## LED Scrolling Marquee

In this project we'll use a built-in driver module to create a scrolling message on an 8×8 matrix.



**COST: \$**

**TIME: 15 MINUTES**



## PARTS REQUIRED

**Arduino board**  
**Female-to-male jumper wires**  
**8x8 LED Maxim 7219 matrix module**

## LIBRARY REQUIRED

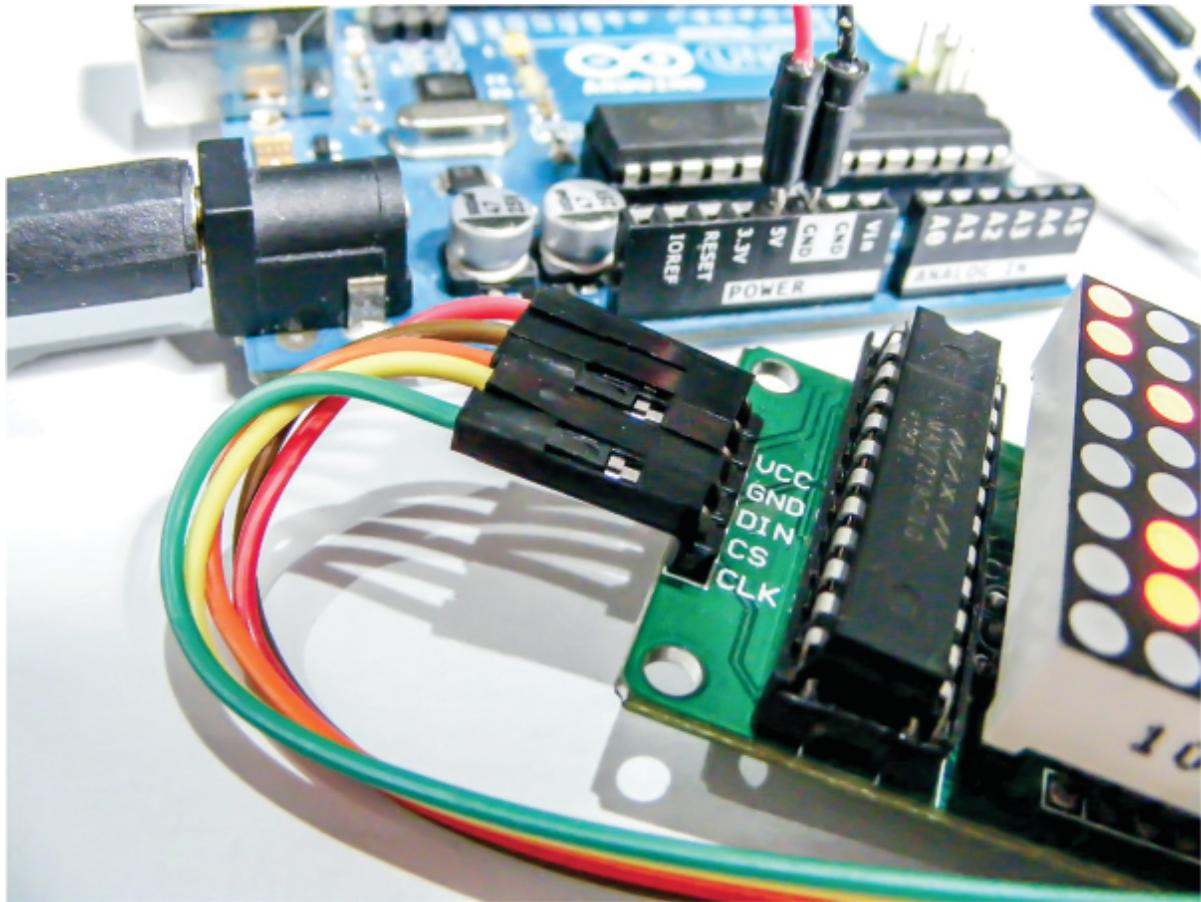
**MaxMatrix**

## HOW IT WORKS

An LED matrix is an array of LEDs that you can control individually to make patterns, text, images, or whatever you can program. The 8x8 LED matrix we'll use in this project comes prebuilt with a *driver module* —a board, driven by a Maxim 7219 chip, that lets you control the entire matrix with only five pins connected to your Arduino. These modules are inexpensive and can be chained together so you have multiple matrices running from one sketch.

The matrix module has three pins: DIN, CS, and CLK, shown in Figure 4-1. *DIN* stands for Data IN, *CS* for Chip Select, and *CLK* for CLocK. The remaining two pins connected to your Arduino power the matrix. The CLK pin senses pulses and controls the speed at which the Arduino and matrix communicate with each other in sync. The matrix uses a *serial peripheral interface (SPI)* communication protocol to speak with the Arduino, and the CS pin detects which SPI device is in use. DIN reads the data—in this case, the project's sketch—from the Arduino.

**FIGURE 4-1:** The Maxim 7219 chip controls the LED matrix.



Each module has extra connections so you can add another module. By chaining together modules and changing the number of matrices in the code, you could scroll a message over a larger area.

## THE BUILD

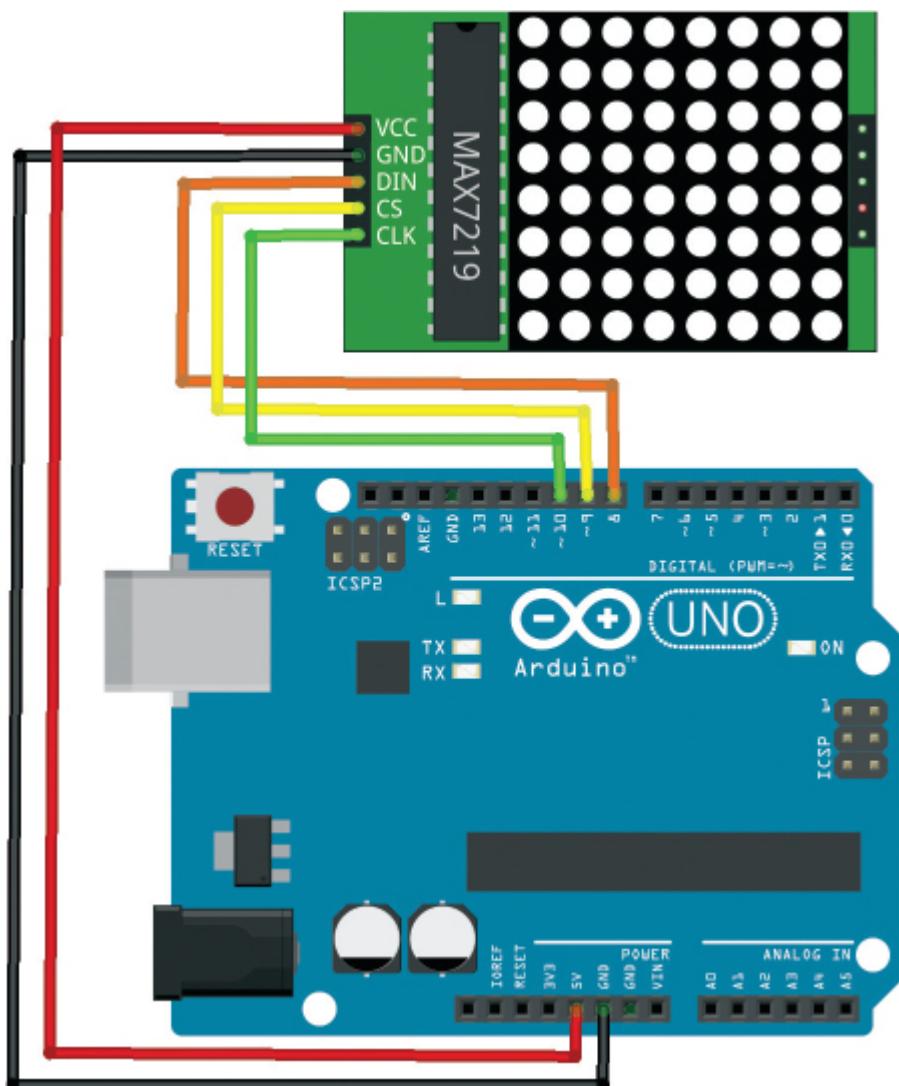
1. Connect the module directly to the Arduino using the female-to-male jumper wires, connecting the female end to the module. As shown in the following table, connect VCC on the LED matrix module to +5V on the Arduino, GND to GND, DIN to Arduino pin 8, CS to Arduino pin 9, and CLK to Arduino pin 10.

LED MATRIX MODULE	ARDUINO
VCC	+5V

LED MATRIX MODULE	ARDUINO
GND	GND
DIN	Pin 8
CS	Pin 9
CLK	Pin 10

2. Confirm that your setup matches the circuit diagram in Figure 4-2, and upload the code in “The Sketch” on page 38.

**FIGURE 4-2:** The circuit diagram for the scrolling LED marquee



## THE SKETCH

This sketch works by calling on the MaxMatrix library to control the matrix module. We then define the characters to display, and set the Arduino pins that control the matrix. Your message will be displayed in a continuous loop on the LEDs.

```
#include <MaxMatrix.h> // Call on the MaxMatrix library

PROGMEM const unsigned char CH[] = {
 3, 8, B00000000, B00000000, B00000000, B00000000, B00000000, // space
 1, 8, B01011111, B00000000, B00000000, B00000000, B00000000, // !
 3, 8, B00000011, B00000000, B00000011, B00000000, B00000000, // "
 5, 8, B00010100, B00111110, B00010100, B00111110, B00010100, // #
}
```

4, 8, B00100100, B01101010, B00101011, B00010010, B00000000, // \$  
5, 8, B01100011, B00010011, B00001000, B01100100, B01100011, // %  
5, 8, B00110110, B01001001, B01010110, B00100000, B01010000, // &  
1, 8, B00000011, B00000000, B00000000, B00000000, B00000000, // '  
3, 8, B00011100, B00100010, B01000001, B00000000, B00000000, // ('  
3, 8, B01000001, B00100010, B00011100, B00000000, B00000000, // ')  
5, 8, B00101000, B00011000, B00001110, B00011000, B00101000, // \*  
5, 8, B00001000, B00001000, B00111110, B00001000, B00001000, // +  
2, 8, B10110000, B01110000, B00000000, B00000000, B00000000, // ,  
4, 8, B00001000, B00001000, B00001000, B00001000, B00000000, // -  
2, 8, B01100000, B01100000, B00000000, B00000000, B00000000, // .  
4, 8, B01100000, B00011000, B00000010, B00000001, B00000000, // /  
4, 8, B00111110, B01000001, B01000001, B00111110, B00000000, // 0  
3, 8, B01000010, B01111111, B01000000, B00000000, B00000000, // 1  
4, 8, B01100010, B01010001, B01001001, B01000110, B00000000, // 2  
4, 8, B00100010, B01000001, B01001001, B00110110, B00000000, // 3  
4, 8, B00011000, B00010100, B00010010, B01111111, B00000000, // 4  
4, 8, B00100111, B01000101, B01000101, B00111001, B00000000, // 5  
4, 8, B00111110, B01001001, B01001001, B00110000, B00000000, // 6  
4, 8, B01100001, B00010001, B00001001, B00000111, B00000000, // 7  
4, 8, B00110110, B01001001, B01001001, B00110110, B00000000, // 8  
4, 8, B00000110, B01001001, B01001001, B00111110, B00000000, // 9  
2, 8, B01010000, B00000000, B00000000, B00000000, B00000000, // :  
2, 8, B10000000, B01010000, B00000000, B00000000, B00000000, // ;  
3, 8, B00010000, B00101000, B01000100, B00000000, B00000000, // <  
3, 8, B00010100, B00010100, B00010100, B00000000, B00000000, // =  
3, 8, B01000100, B00101000, B00010000, B00000000, B00000000, // >  
4, 8, B00000010, B01011001, B00001001, B00000110, B00000000, // ?  
5, 8, B00111110, B01001001, B01010101, B01011101, B00001110, // @  
4, 8, B01111110, B00010001, B00010001, B01111110, B00000000, // A  
4, 8, B01111111, B01001001, B01001001, B00110110, B00000000, // B  
4, 8, B00111110, B01000001, B01000001, B00100010, B00000000, // C  
4, 8, B01111111, B01000001, B01000001, B00111110, B00000000, // D  
4, 8, B01111111, B01001001, B01001001, B01000001, B00000000, // E  
4, 8, B01111111, B00001001, B00001001, B00000001, B00000000, // F  
4, 8, B00111110, B01000001, B01001001, B01111010, B00000000, // G  
4, 8, B01111111, B00001000, B00001000, B01111111, B00000000, // H  
3, 8, B01000001, B01111111, B01000001, B00000000, B00000000, // I  
4, 8, B00110000, B01000000, B01000001, B00111111, B00000000, // J  
4, 8, B01111111, B00001000, B00010100, B01100011, B00000000, // K  
4, 8, B01111111, B01000000, B01000000, B01000000, B00000000, // L  
5, 8, B01111111, B00000010, B00001100, B00000010, B01111111, // M  
5, 8, B01111111, B00000100, B00001000, B00010000, B01111111, // N  
4, 8, B00111110, B01000001, B01000001, B00111110, B00000000, // O  
4, 8, B01111111, B00001001, B00001001, B00000110, B00000000, // P  
4, 8, B00111110, B01000001, B01000001, B01111110, B00000000, // Q  
4, 8, B01111111, B00001001, B00001001, B01110110, B00000000, // R  
4, 8, B01000110, B01001001, B01001001, B00110010, B00000000, // S  
5, 8, B00000001, B00000001, B01111111, B00000001, B00000001, // T  
4, 8, B00111111, B01000000, B01000000, B00111111, B00000000, // U

```

5, 8, B00001111, B00110000, B01000000, B00110000, B00001111, // v
5, 8, B00111111, B01000000, B00111000, B01000000, B00111111, // w
5, 8, B01100011, B00010100, B00001000, B00010100, B01100011, // x
5, 8, B00000111, B00001000, B01110000, B00001000, B00000111, // y
4, 8, B01100001, B01010001, B01001001, B01000111, B00000000, // z
2, 8, B01111111, B01000001, B00000000, B00000000, B00000000, // [
4, 8, B00000001, B00000110, B00011000, B01100000, B00000000, // \
2, 8, B01000001, B01111111, B00000000, B00000000, B00000000, // ]
3, 8, B00000010, B00000001, B00000010, B00000000, B00000000, // hat
4, 8, B01000000, B01000000, B01000000, B01000000, B00000000, // -
2, 8, B00000001, B00000010, B00000000, B00000000, B00000000, // ^
4, 8, B00100000, B01010100, B01010100, B01111000, B00000000, // a
4, 8, B01111111, B01000100, B01000100, B00111000, B00000000, // b
4, 8, B00111000, B01000100, B01000100, B00101000, B00000000, // c
4, 8, B00111000, B01000100, B01000100, B01111111, B00000000, // d
4, 8, B00111000, B01010100, B01010100, B00011000, B00000000, // e
3, 8, B00000100, B01111110, B00000010, B00000000, B00000000, // f
4, 8, B10011000, B10100100, B10100100, B01111000, B00000000, // g
4, 8, B01111111, B000000100, B000000100, B01111000, B00000000, // h
3, 8, B01000100, B01111101, B01000000, B00000000, B00000000, // i
4, 8, B01000000, B10000000, B100000100, B01111101, B00000000, // j
4, 8, B01111111, B00010000, B00101000, B01000100, B00000000, // k
3, 8, B01000001, B01111111, B01000000, B00000000, B00000000, // l
5, 8, B01111100, B000000100, B01111100, B000000100, B01111000, // m
4, 8, B01111100, B000000100, B000000100, B01111000, B00000000, // n
4, 8, B00111000, B01000100, B01000100, B00111000, B00000000, // o
4, 8, B11111100, B00100100, B00100100, B00011000, B00000000, // p
4, 8, B00011000, B00100100, B00100100, B11111100, B00000000, // q
4, 8, B01111100, B000001000, B000000100, B000000100, B00000000, // r
4, 8, B01001000, B01010100, B01010100, B00100100, B00000000, // s
3, 8, B00000100, B00111111, B01000100, B00000000, B00000000, // t
4, 8, B00111100, B01000000, B01000000, B01111100, B00000000, // u
5, 8, B00011100, B00100000, B01000000, B00100000, B00011100, // v
5, 8, B00111100, B01000000, B00111100, B01000000, B00111100, // w
5, 8, B01000100, B00101000, B00010000, B00101000, B01000100, // x
4, 8, B10011100, B10100000, B10100000, B01111100, B00000000, // y
3, 8, B01100100, B01010100, B01001100, B00000000, B00000000, // z
3, 8, B00001000, B00110110, B01000001, B00000000, B00000000, // {
1, 8, B01111111, B00000000, B00000000, B00000000, B00000000, // |
3, 8, B01000001, B00110110, B00001000, B00000000, B00000000, // }
4, 8, B00001000, B00000100, B00001000, B00000100, B00000000, // ~
};

int data = 8; // Pin connected to DIN pin of MAXIM7219 module
int load = 9; // Pin connected to CS pin of MAXIM7219 module
int clock = 10; // Pin connected to CLK pin of MAXIM7219 module

① int maxInUse = 1; // Set the number of matrices you are using
MaxMatrix m(data, load, clock, maxInUse); // Define the module
byte buffer[10];

```

```

// Set message to scroll on the screen
② char string1[] = " Arduino Project Handbook . . . ";
void setup() {
    m.init(); // Start module
    m.setIntensity(0);
    Serial.begin(9600); // Start serial communication
}

void loop() {
    byte c;
    while (Serial.available() > 0) {
        byte c = Serial.read();
        Serial.println(c, DEC);
        printCharWithShift(c, 100);
    }
    delay(100);
    m.shiftLeft(false, true);
    printStringWithShift(string1, 100);
}

// The remainder of this sketch moves the scrolling characters
// depending on the number of matrices that are attached
void printCharWithShift(char c, int shift_speed) {
    if (c < 32) return;
    c -= 32;
    memcpy_P(buffer, CH + 7 * c, 7);
    m.writeSprite(maxInUse * 8, 0, buffer);
    m.setColumn(maxInUse * 8 + buffer[0], 0);
    for (int i = 0; i < buffer[0] + 1; i++) {
        delay(shift_speed);
        m.shiftLeft(false, false);
    }
}

void printStringWithShift(char* s, int shift_speed) {
    while (*s != 0) {
        printCharWithShift(*s, shift_speed);
        s++;
    }
}

void printString(char* s) {
    int col = 0;
    while (*s != 0) {
        if (*s < 32) continue;
        char c = *s - 32;
        memcpy_P(buffer, CH + 7 * c, 7);
        m.writeSprite(col, 0, buffer);
        m.setColumn(col + buffer[0], 0);
    }
}

```

```
    col += buffer[0] + 1;  
    s++;  
}  
}
```

---

You can change the message on the LED matrix by altering the text inside the quotation marks at ❷. If you want to chain your matrices together, change the number at ❶ to the number you have (the maximum number of matrices you can chain together is seven).

## TROUBLESHOOTING

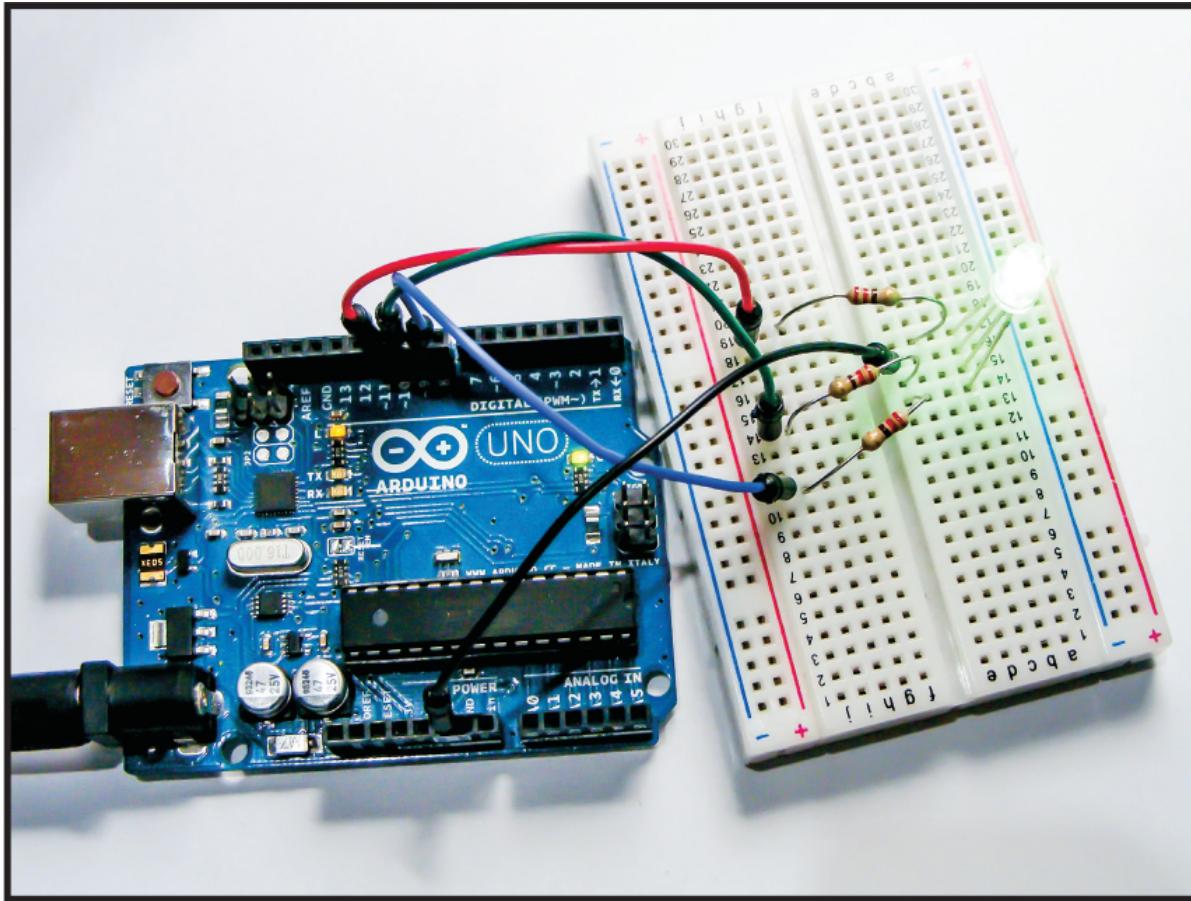
**Q.** *The matrix does not light up or the LED shows erratic symbols.*

- If none of the LEDs light, make sure you have connected the matrix as shown in the circuit diagram in Figure 4-2; the pins must match exactly.
- Make sure that your Arduino is powered and the TX light is flashing. If not, recheck your batteries or power supply.
- Make sure the Maxim 7219 chip is securely inserted in the module.

# 5

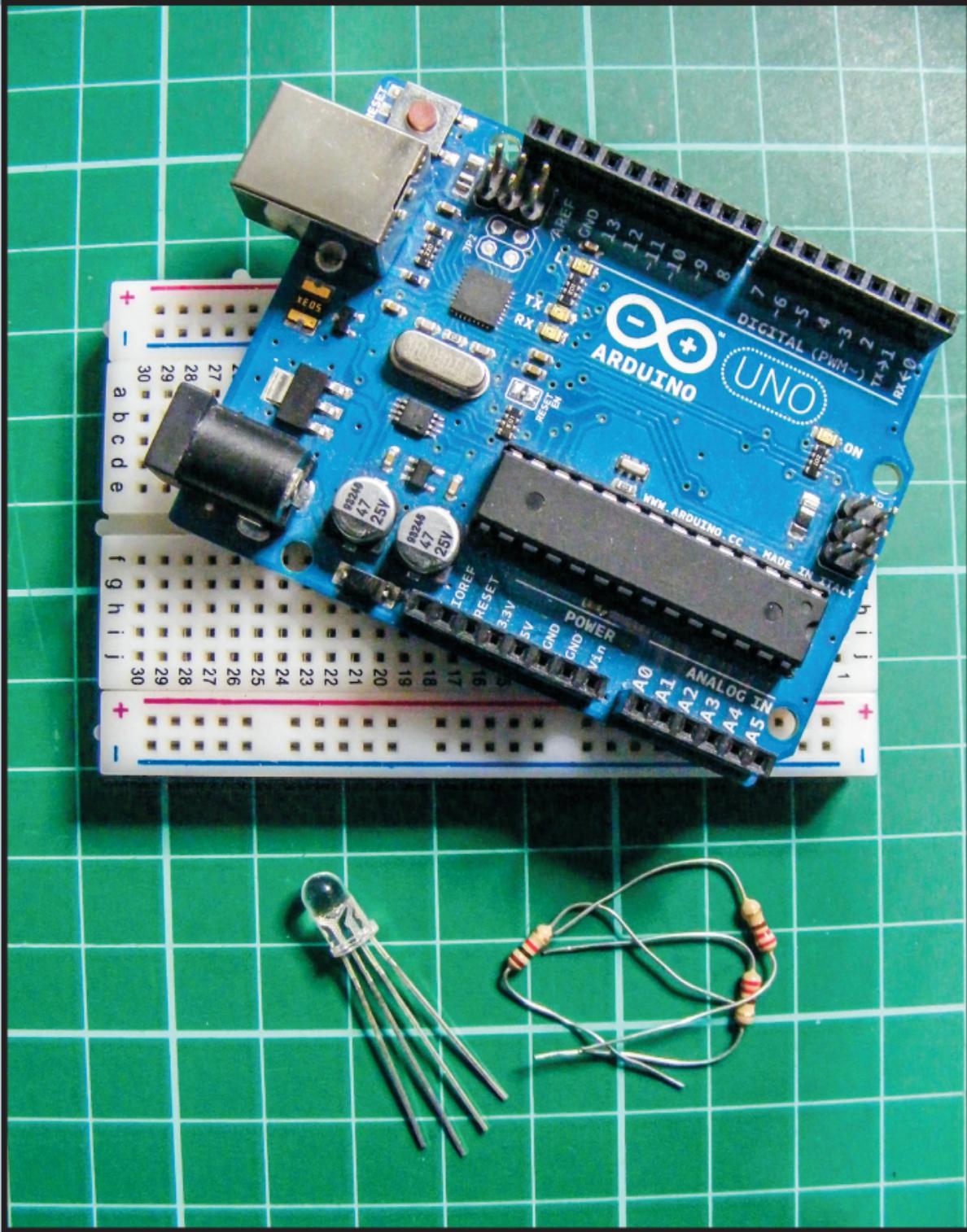
## Mood Light

In this project we'll create a soothing mood light using a single multicolored LED.



**COST: \$**

**TIME: 10 MINUTES**



# PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

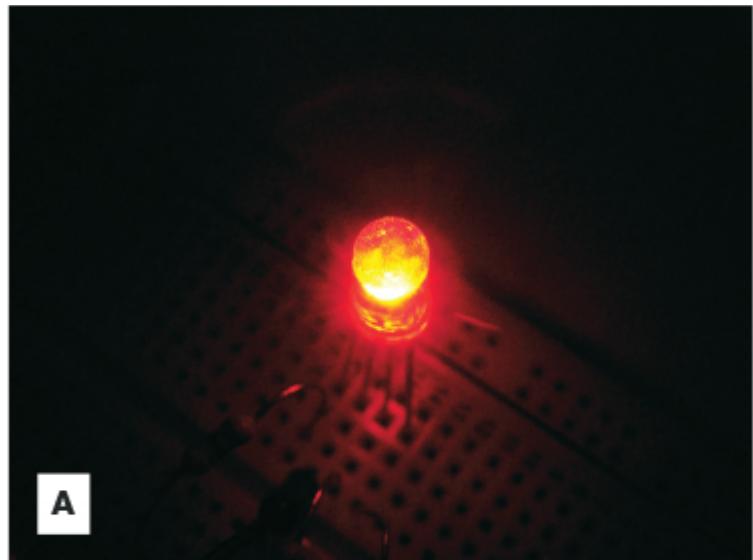
**RGB common-cathode LED**

**3 220-ohm resistors**

## **HOW IT WORKS**

LEDs come in many different colors and forms, but one of the most useful is the RGB LED. As its name implies, an RGB LED is actually three LEDs in one: red, green, and blue (see Figure 5-1).

**FIGURE 5-1:** The primary colors of the RGB LED



**A**



**B**

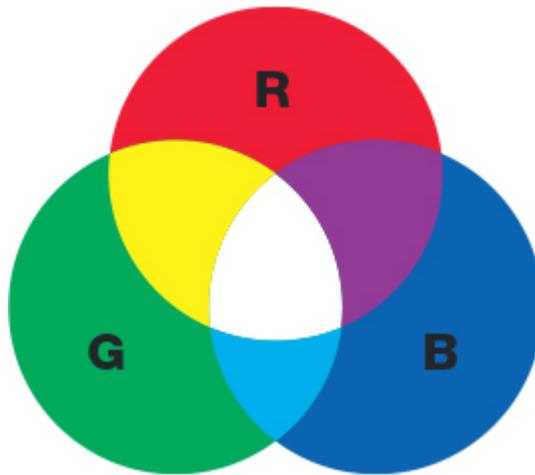


**C**

---

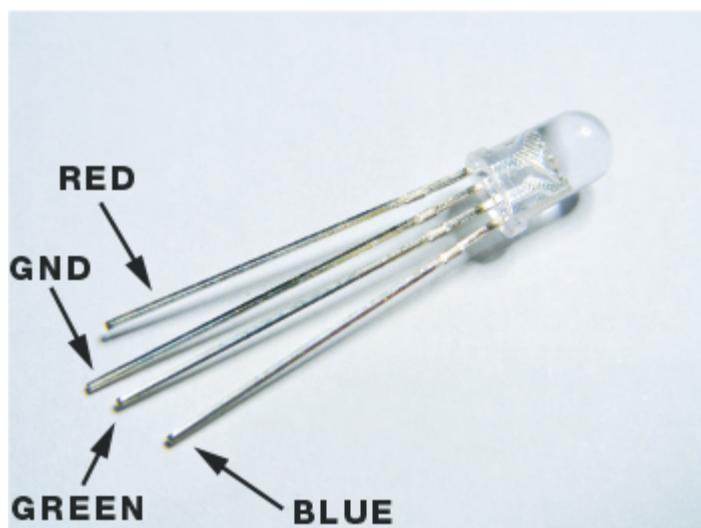
RGB is an *additive* color model, which means that by combining the light of two or more colors we can create other colors. Red, green, and blue are the additive primary colors used as the base for other colors, as shown in Figure 5-2.

**FIGURE 5-2:** RGB is an additive color model.



Let's look at an RGB LED in a bit more detail in Figure 5-3.

**FIGURE 5-3:** An RGB LED



You'll see that the RGB LED has four legs instead of the usual two: one each for red, green, and blue, and the fourth one is either the cathode or anode. We'll be using a *common-cathode* RGB LED like the

one in the figure, where the longest leg is the cathode and connects to ground.

We can use our RGB LED to create a random-color output that cycles through the colors of the rainbow, fading each one in and out. This lighting effect is used quite often in clubs or bars to create a relaxing mood. You could also place the LED in an opaque vase or box for a soothing night-light.

## THE BUILD

1. Begin by placing the common-cathode RGB LED into your breadboard with the red leg in the hole to the left of the long GND (or cathode) leg. Connect a 220-ohm resistor to each of the three color legs.

### NOTE

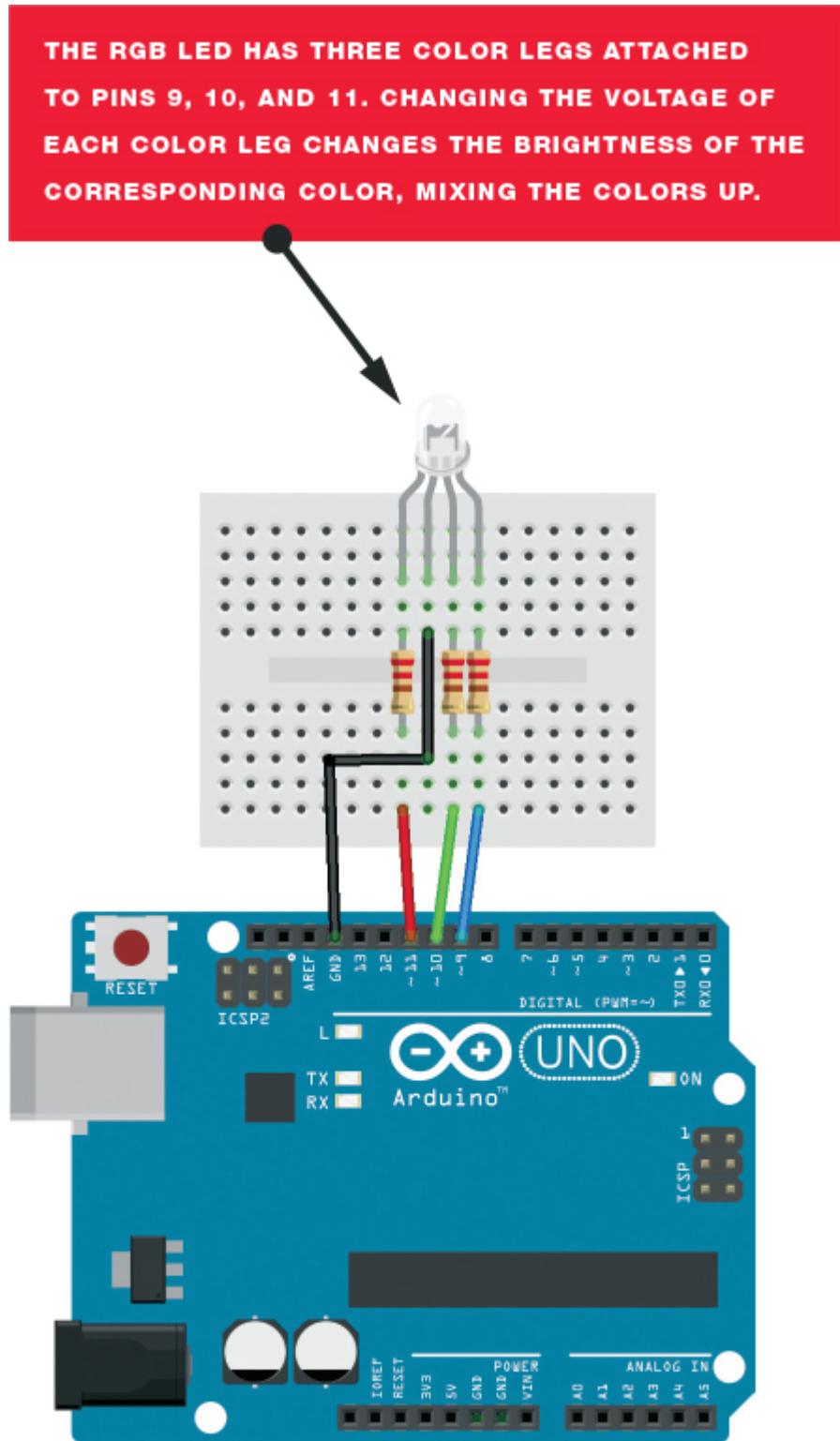
*On some RGB LEDs the green and blue legs are the other way around.*

2. Connect the red leg to Arduino pin 11, GND to Arduino GND, green to Arduino pin 10, and blue to Arduino pin 9.

COMMON-CATHODE RGB LED	ARDUINO
Red	Pin 11
GND	GND
Green	Pin 10
Blue	Pin 9

3. Confirm that your setup matches the circuit diagram in Figure 5-4, and upload the code in “The Sketch” on page 47.

**FIGURE 5-4:** The circuit diagram for the mood light



## THE SKETCH

The sketch first sets Arduino pins 9, 10, and 11 as outputs. This sketch varies the brightness (power) value of each light on the RGB LED in turn by switching them on and off incredibly quickly—the longer an LED is lit for, the brighter it appears. To do this the Arduino uses a technique called *pulse width modulation (PWM)*. The Arduino creates a pulse by switching the power on and off very quickly. The duration that the power is on or off (known as the *pulse width*) in the cycle determines the average output, and by varying this pulse width the Arduino can simulate voltages between full on (5 volts) and off (0 volts). If the signal from the Arduino is on for half the time and off for half, the average output will be 2.5 volts, halfway between 0 and 5. If the signal is on for 80 percent and off for 20 percent, the voltage is 4 volts, and so on.

We define an RGB value between 0 and 255, with an increment of 5 volts, to create a fade effect. In simple terms, each color of the LED brightens from 0 to 5 volts in sequence, and then fades out when it reaches its maximum value of 255. The Arduino can handle values between 0 and 1023 (1,024 values in total), but because this is such a high number we divide it by 4 and use 255 as the maximum LED value so the color change is more noticeable.

```
-----  
int redPin = 11; // Pin connected to red leg of the RGB LED  
int greenPin = 10; // Pin connected to green leg of the RGB LED  
int bluePin = 9; // Pin connected to blue leg of the RGB LED  
  
void setup() {  
    setRgb(0, 0, 0); // Set all colors at 0  
}  
  
void loop() {  
    int Rgb[3]; // 3 RGB pins  
  
    Rgb[0] = 0; // A value for each  
    Rgb[1] = 0;  
    Rgb[2] = 0;  
  
    // Colors increase and decrease in value  
    for (int decrease = 0; decrease < 3; decrease += 1) {  
        int increase = decrease == 2 ? 0 : decrease + 1;  
    }  
}
```

```
for (int i = 0; i < 255; i += 1) { // Fade the colors
    Rgb[decrease] -= 1;
    Rgb[increase] += 1;
    setRgb(Rgb[0], Rgb[1], Rgb[2]);
    delay(20);
}
}

void setRgb (int red, int green, int blue) {
    analogWrite(redPin, red);
    analogWrite(greenPin, green);
    analogWrite(bluePin, blue);
}
```

---

## TROUBLESHOOTING

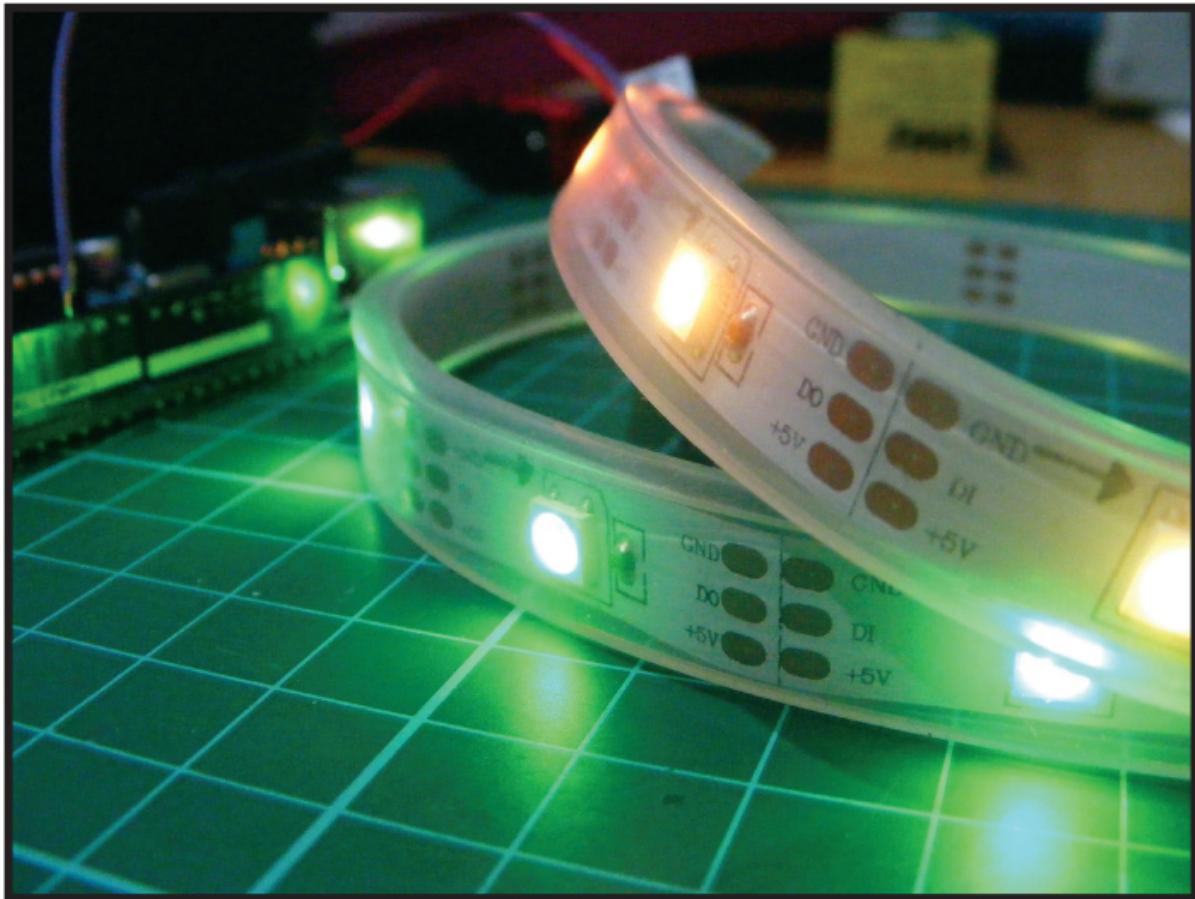
**Q.** *The code compiles, but the RGB LED does not light up as expected.*

- If the RGB LED does not light at all, make sure you've connected the GND wire from the Arduino to the correct leg on the RGB LED—the long cathode leg—and that the Arduino has power connected.
- If you have a common-anode RGB LED, then you should connect the long leg to +5V on the Arduino. Check the data sheet for your part to find out which kind of RGB LED you have.
- If the colors don't appear as expected, your RGB LED may have a different pin configuration; check your data sheet or try swapping the connections to the green and blue legs around.

# 6

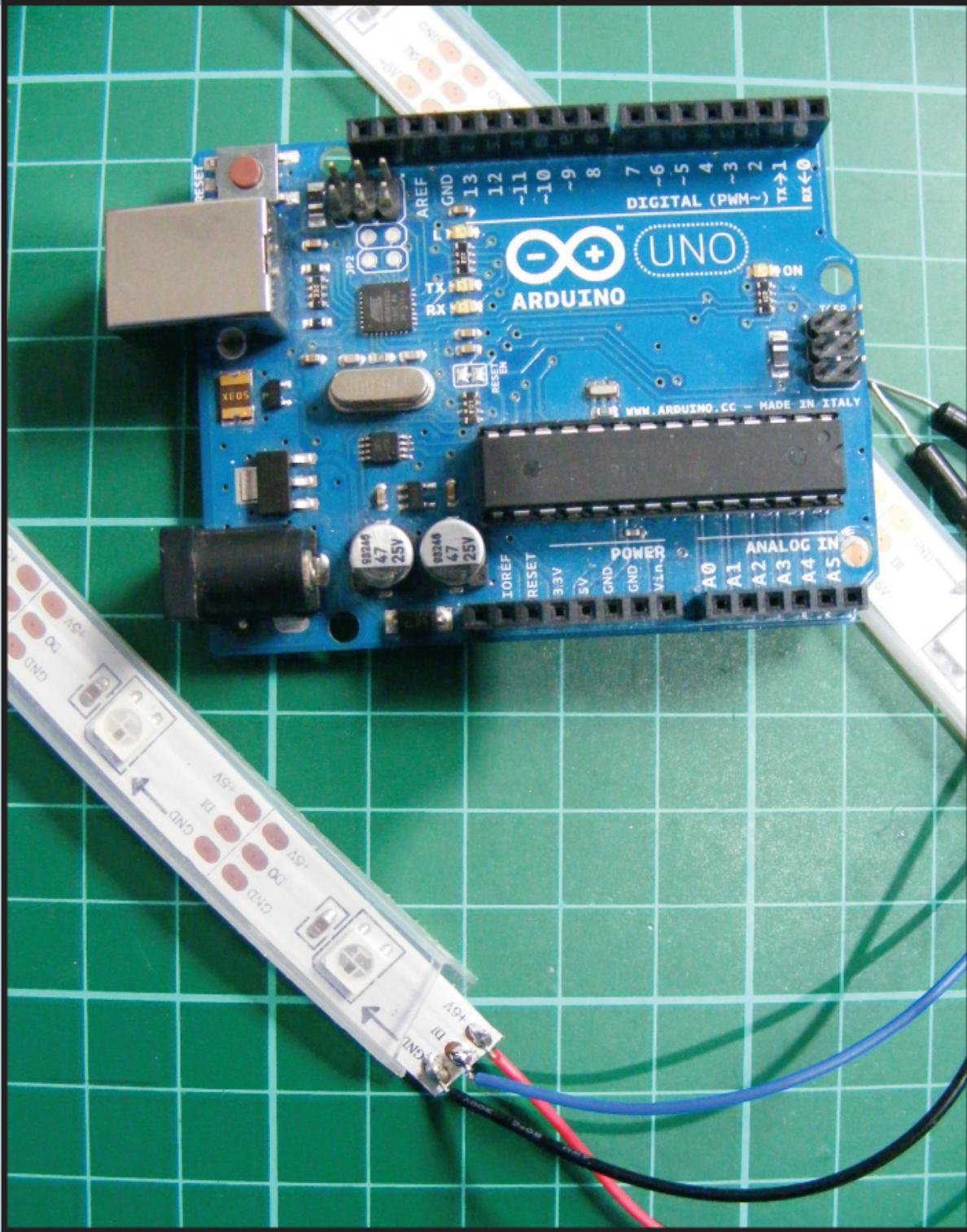
## Rainbow Strip Light

In this chapter we'll use an RGB LED strip light to create a decorative ambient strip of rainbow colors.



COST: \$\$

TIME: 15 MINUTES



## PARTS REQUIRED

**Arduino board**  
**Solid-core wires**  
**RGB LED strip (WS2812B 5V 32-LED strip)**

## LIBRARY REQUIRED

**PololuLedStrip**

## HOW IT WORKS

LED strip lights are often used to create ambiance as a decorative feature, such as backlighting for a TV or lighting beneath kitchen cabinets. They are low-powered, typically between 5 and 12 volts, so they're easy to install anywhere with their own power supply—and they look good too!

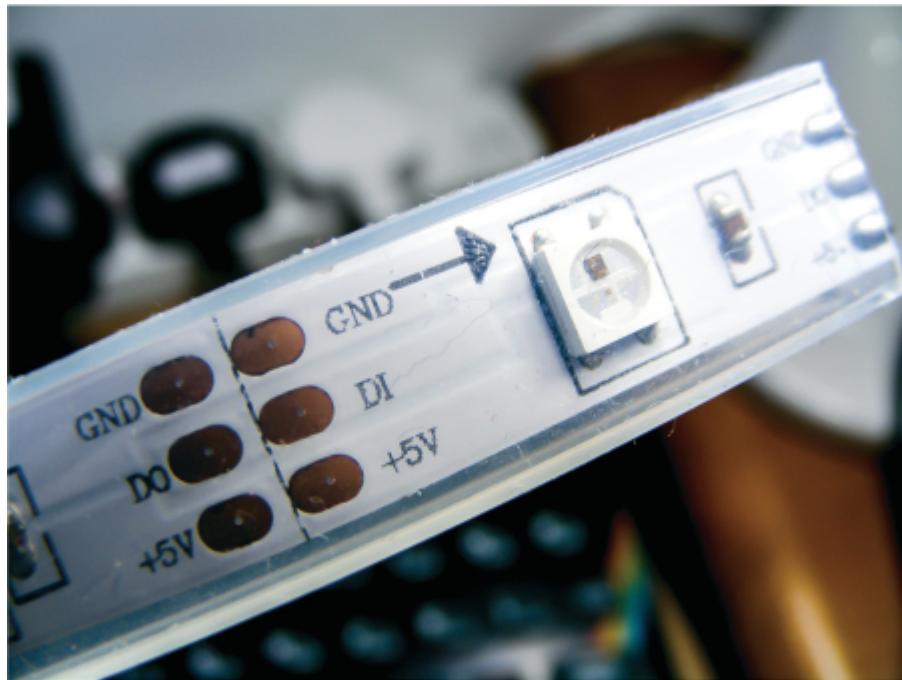
Strip lights generally come in two varieties. Single-color or multicolor *nonaddressable* strips can only light all the LEDs in one color at a time. RGB multicolored strips are generally *addressable*, which means that each LED has its own chip and can be individually controlled, allowing multiple colors on different LEDs to light at a time.

We'll be using a strip light of addressable RGB LEDs. Unlike the RGB LED from Project 5, the LEDs on a strip light are *surface mounted*. This means that the components are placed directly onto the surface of a printed circuit board—in this case, a flexible strip—rather than being individually inserted into a circuit.

There are two main kinds of addressable RGB strip lights. Three-pin RGB LED strips have GND, Data, and +5V connections to control the LEDs. The Data pin connects to the Arduino and uses the same *pulse width modulation (PWM)* function explained in Project 5 to create the colors and sequence on the strip. Four-pin RGB LED strips have GND, Clock, Data In, and +5V connections and use *Serial Peripheral Interface (SPI)* to control their LEDs. SPI is a communication method that allows the two-way transfer of data between devices.

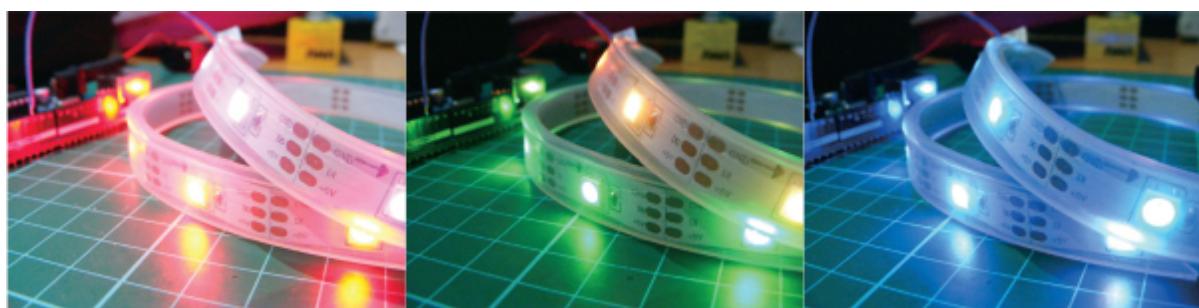
Our addressable RGB LED strip, shown in Figure 6-1, is the three-pin type using PWM. It calls on the PololuLedStrip library, created by Pololu Robotics and Electronics (<https://www.pololu.com/>), to control the LEDs.

**FIGURE 6-1:** A three-pin addressable RGB LED strip light



We'll use our RGB LED strip to create a color output that cycles through the colors of the rainbow, fading each color in and out, as shown in Figure 6-2.

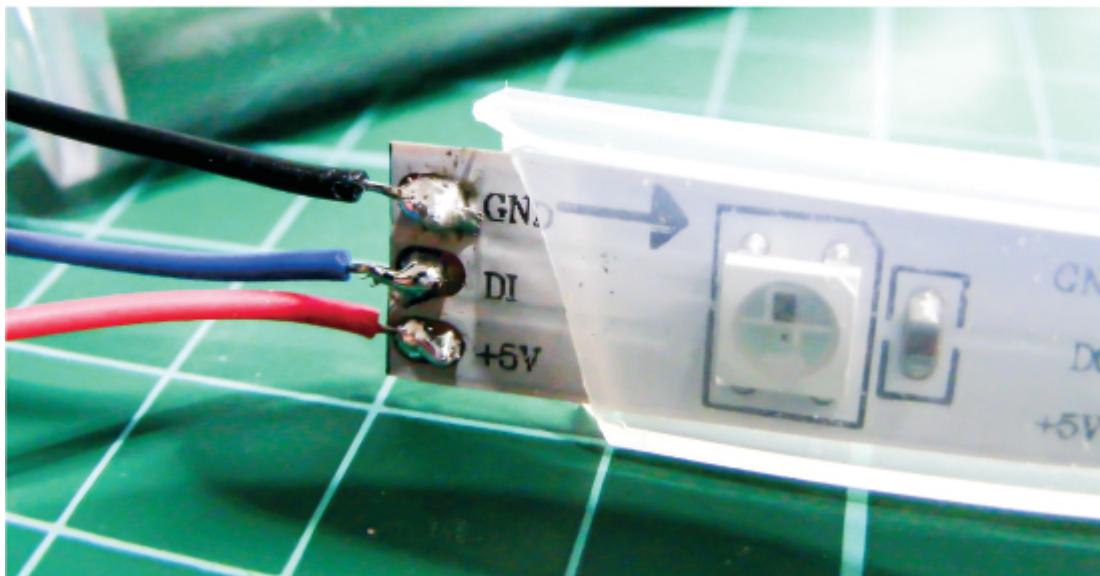
**FIGURE 6-2:** RGB LED strip cycling through the colors of the rainbow



## THE BUILD

1. Download and add the PololuLedStrip library to your Arduino IDE (check the primer for guidance on saving libraries).
2. The setup for this project is very simple and doesn't take long to complete. Most three-pin addressable RGB LED strips come without wires attached to the strip connections, so you'll have to connect them. With the LEDs facing upward, begin by soldering solid-core wire to the three connections at the left end of the strip, as shown in Figure 6-3.

**FIGURE 6-3:** Soldering wires to the left-side connections

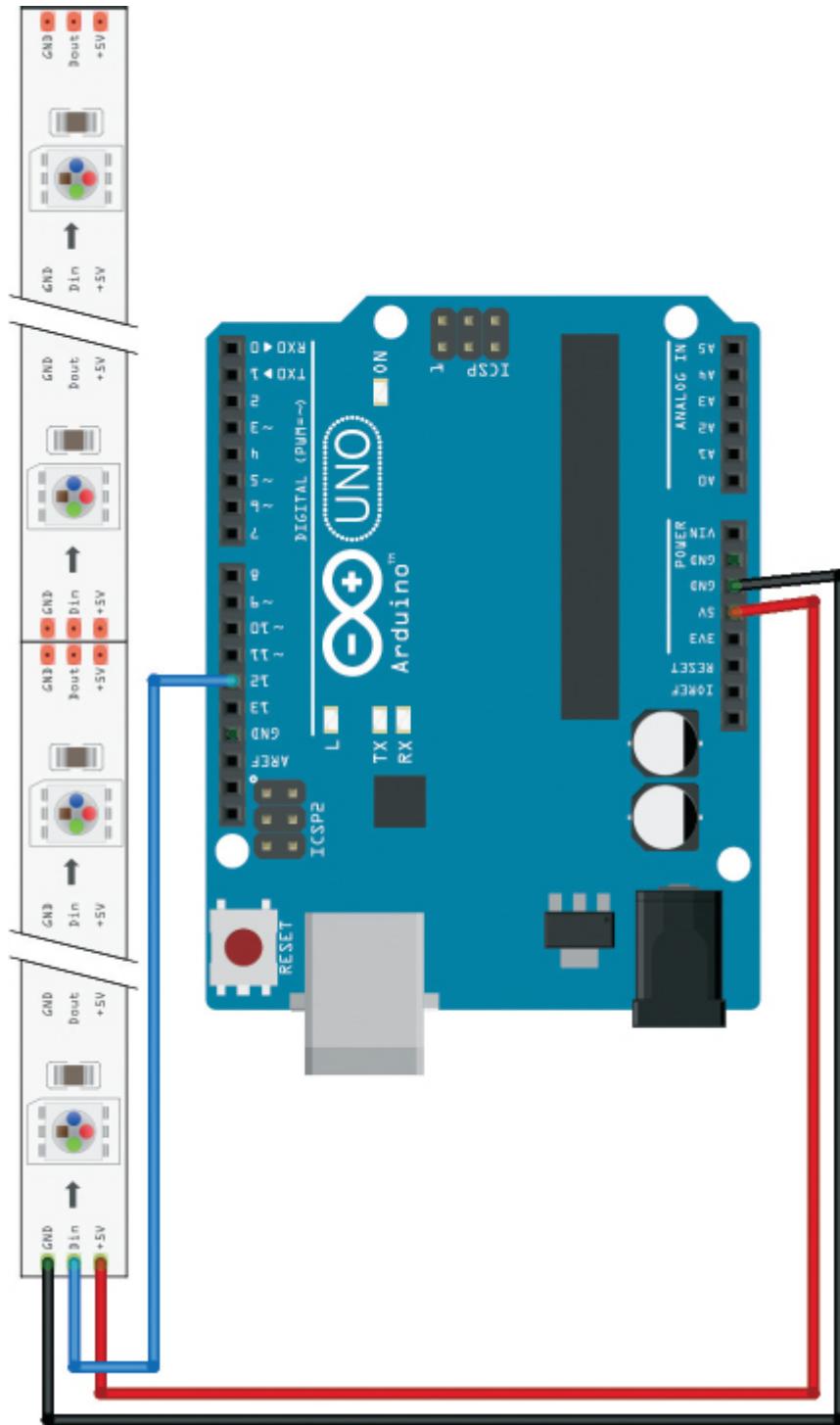


3. Connect the LED's GND pin to Arduino GND, DI to Arduino pin 12, and +5V to Arduino +5V, as shown in the following table.

RGB LED STRIP	ARDUINO
GND	GND
DI (data in)	Pin 12
+5V	+5V

4. Check your setup against the circuit diagram in Figure 6-4, and then upload the code in “The Sketch” below and power the Arduino using your battery pack.

**FIGURE 6-4:** The circuit diagram for the rainbow strip light



## THE SKETCH

The sketch first calls on the PololuLedStrip library, which we use to control the individual LEDs. Next, it defines the pin to control the data going from the Arduino to the LED strip as 12 and sets the number of LEDs on the strip to 32—you would change this if your strip had a different number of LEDs.

Next is a calculation to control the hue, saturation, and value (HSV) of our LEDs to generate the RGB colors. You can change these using an HSV chart if you want; just do a quick internet search to find a chart for reference.

The WS2812B data sheet states that the color of each LED is encoded as three LED brightness values, which must be sent in GRB (green-red-blue) order. The first color transmitted applies to the LED that is closest to the data input connector, the second color transmitted applies to the next LED in the strip, and so on.

---

```
/* PololuLedStrip Library Copyright (c) 2012 Pololu Corporation.  
For more information, see http://www.pololu.com/;  
http://forum.pololu.com/
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

LedStripRainbow: Example Arduino sketch that shows how to make a moving rainbow pattern on an Addressable RGB LED Strip from Pololu. To use this, you will need to plug an Addressable RGB LED strip from

Pololu into pin 12. After uploading the sketch, you should see a moving rainbow. \*/

```
#include <PololuLedStrip.h>

// Create an ledStrip object and specify the pin it will use.
PololuLedStrip<12> ledStrip;

// Create a buffer for holding the colors (3 bytes per color).
#define LED_COUNT 32
rgb_color colors[LED_COUNT];

void setup() {
}

// Converts a color from HSV to RGB.
// h is hue, as a number between 0 and 360.
// s is saturation, as a number between 0 and 255.
// v is value, as a number between 0 and 255.

rgb_color hsvToRgb(uint16_t h, uint8_t s, uint8_t v) {
    uint8_t f = (h % 60) * 255 / 60;
    uint8_t p = (255 - s) * (uint16_t)v / 255;
    uint8_t q = (255 - f * (uint16_t)s / 255) * (uint16_t)v / 255;
    uint8_t t = (255 - (255 - f) * (uint16_t)s / 255) * (uint16_t)v / 255;
    uint8_t r = 0, g = 0, b = 0;
    switch((h / 60) % 6) {
        case 0: r = v; g = t; b = p; break;
        case 1: r = q; g = v; b = p; break;
        case 2: r = p; g = v; b = t; break;
        case 3: r = p; g = q; b = v; break;
        case 4: r = t; g = p; b = v; break;
        case 5: r = v; g = p; b = q; break;
    }
    return (rgb_color) {
        r, g, b
    };
}

void loop() {
    // Update the colors.
    uint16_t time = millis() >> 2;
    for (uint16_t i = 0; i < LED_COUNT; i++) {
        byte x = (time >> 2) - (i << 3);
        colors[i] = hsvToRgb((uint32_t)x * 359 / 256, 255, 255);
    }

    // Write the colors to the LED strip.
    ledStrip.write(colors, LED_COUNT);
```

```
    delay(10);  
}
```

---

## TROUBLESHOOTING

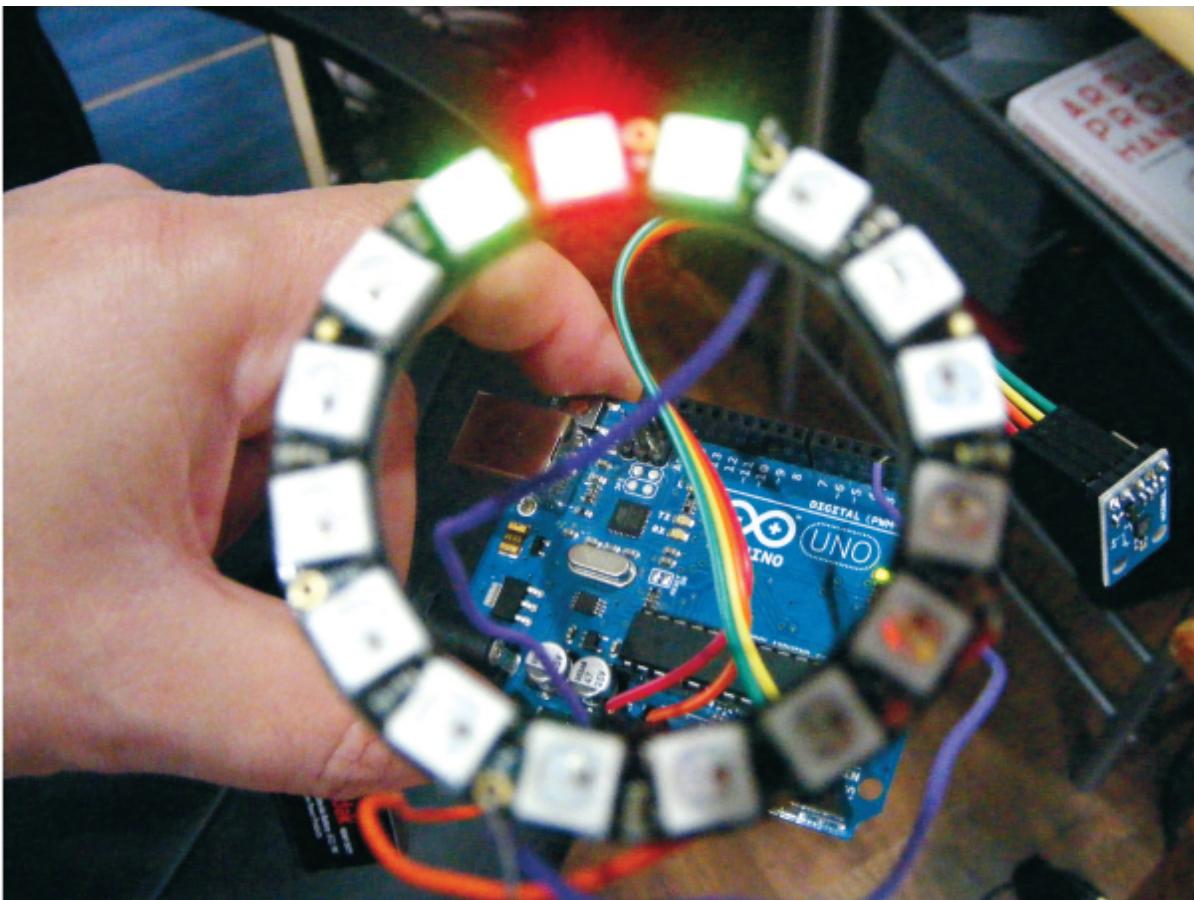
**Q.** *The code compiles, but the RGB LED does not light up as expected.*

- If the RGB LED strip does not light, make sure that your wires are connected as shown in Figure 6-4, and that your LED strip is the WS2812B type specified.
- If you aren't doing so already, use an external power source for the RGB LED strip.

# 7

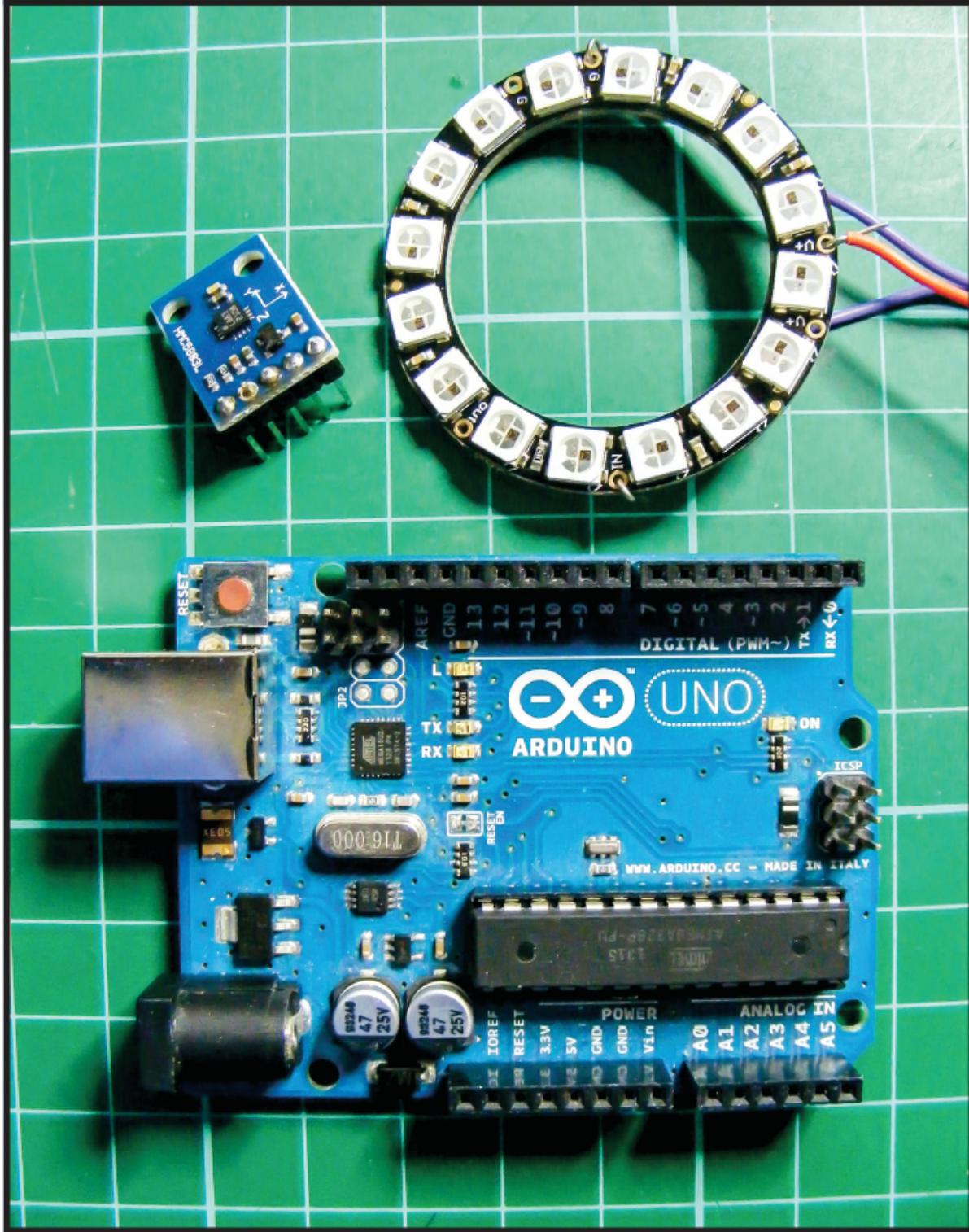
## NeoPixel Compass

In this chapter we'll use a three-axis sensor and an RGB LED ring to create a compass that indicates north by lighting the LEDs in that direction.



COST: \$\$

TIME: 25 MINUTES



## PARTS REQUIRED

**Arduino board**

**Jumper wires**

**HMC5883L three-axis sensor**

**Adafruit NeoPixel ring with 16 RGB LEDs**

**9V battery pack with 6 AA batteries**

## **LIBRARIES REQUIRED**

**Wire**

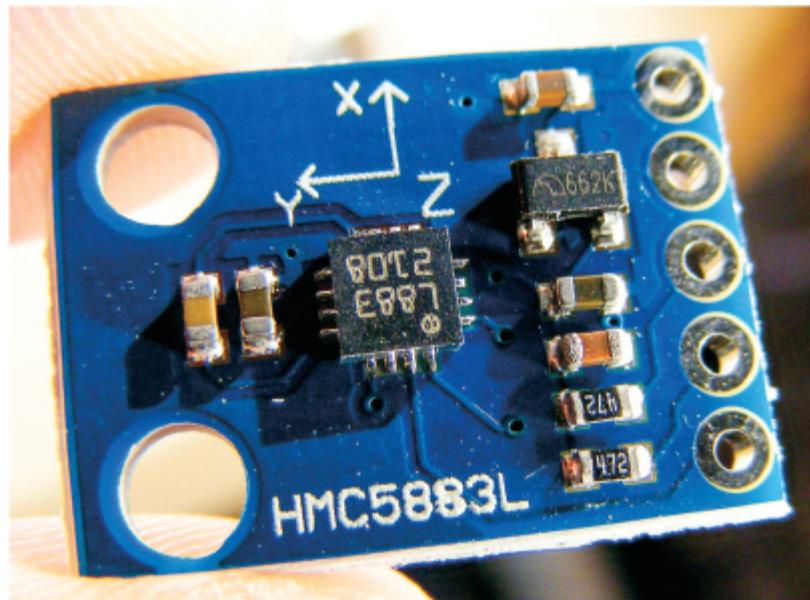
**FastLED**

**HMC5883L**

## **HOW IT WORKS**

The HMC5883L three-axis sensor (Figure 7-1) is a multichip module that senses magnetic force. The module measures both the direction and the magnitude of Earth's magnetic fields. We will use the HMC5883L library to turn our project into an electronic compass.

**FIGURE 7-1:** The HMC5883L three-axis module runs on 3.3V rather than 5V.

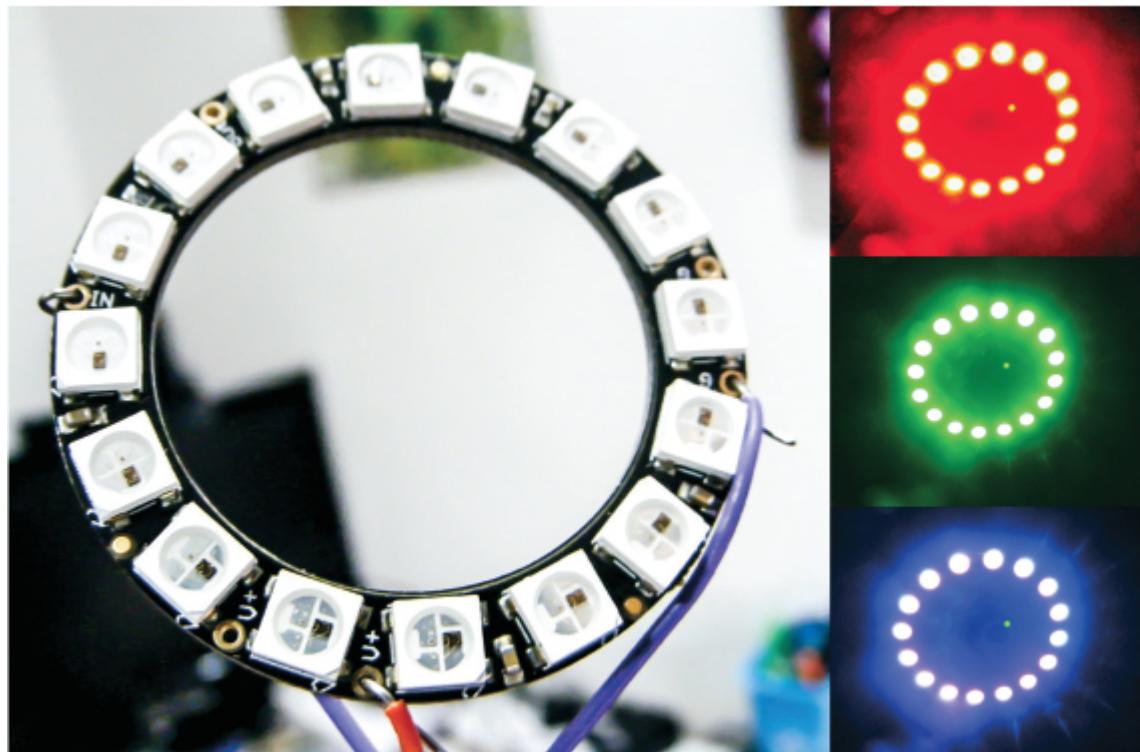


Earth's magnetic field is believed to be generated by electric currents in the conductive material of its core that are created by heat

escaping. Since Earth is effectively a magnet, the north end of a compass magnet is drawn to align with its magnetic field.

To visualize our compass direction we will use the Adafruit NeoPixel ring, shown in Figure 7-2. The NeoPixel ring is made up of 16 RGB LEDs, each of which has its own driver chip and so can be controlled individually. A single data line controls the LEDs, and we'll use the FastLED library to control the colors.

**FIGURE 7-2:** The Adafruit 16 RGB NeoPixel ring



When the project is powered up, the HMC5883L module will detect magnetic north and display it on the NeoPixel ring by lighting the LEDs in that direction. If you turn around while holding the powered NeoPixel compass, the LED lights will move to always point north.

## THE BUILD

### NOTE

*The pin labeled DRDY on the compass module is not used in this project.*

Your HMC5883L module may arrive with the header pins loose, so the first step is to solder the header pins into the module. You will need the strip of five header pins that should come with the module. Insert the header pins into the five available holes on the module and solder each pin for a couple of seconds (check the “Quick Soldering Guide” on page 12 if you need help). The module communicates with the Arduino using I2C and the Wire library.

1. In order to use the compass properly you need to calibrate the HMC5883L module. Connect the module to the Arduino as shown in the following table.

HMC5883L MODULE	ARDUINO
VCC	+3.3V
GND	GND
SCL	Pin A5 (SLC)
SDA	Pin A4 (SDA)

2. Download the HMC5883L library and add it to the Arduino library folder on your PC. Check the library section in the primer if you need a reminder of how to do this. Once you have the library saved, restart your Arduino IDE. When it opens again, it should have the library saved in *Examples*. Select **File** ▶ **Examples** ▶ **Arduino-HMC5883L-Master** ▶ **HMC5883L\_calibrate**. If you can't see the sketch, make sure you've saved the library in your Arduino library folder. The following sketch will be shown in the IDE main window:

```
/*
  Calibrate HMC5883L. Output for HMC5883L_calibrate_processing.pde
```

Read more: <http://www.jarzebski.pl/arduino/czujniki-i-sensory/3-osiowy-magnetometr-hmc5883l.html>  
GIT: <https://github.com/jarzebski/Arduino-HMC5883L>  
Web: <http://www.jarzebski.pl>  
(c) 2014 by Korneliusz Jarzebski  
\*/

```
#include <Wire.h>
#include <HMC5883L.h>

HMC5883L compass;

int minX = 0;
int maxX = 0;
int minY = 0;
int maxY = 0;
int offX = 0;
int offY = 0;

void setup() {
    Serial.begin(9600);

    // Initialize Initialize HMC5883L
    while (!compass.begin()) {
        delay(500);
    }
    // Set measurement range
    compass.setRange(HMC5883L_RANGE_1_3GA);
    // Set measurement mode
    compass.setMeasurementMode(HMC5883L_CONTINUOUS);
    // Set data rate
    compass.setDataRate(HMC5883L_DATARATE_30HZ);
    // Set number of samples averaged
    compass.setSamples(HMC5883L_SAMPLES_8);
}

void loop() {
    Vector mag = compass.readRaw();
    // Determine Min / Max values
    if (mag.XAxis < minX) minX = mag.XAxis;
    if (mag.XAxis > maxX) maxX = mag.XAxis;
    if (mag.YAxis < minY) minY = mag.YAxis;
    if (mag.YAxis > maxY) maxY = mag.YAxis;

    // Calculate offsets
    offX = (maxX + minX)/2;
    offY = (maxY + minY)/2;

    /*Serial.print(mag.XAxis);
    Serial.print(":");
    Serial.print(mag.YAxis);
    Serial.print("\n");*/
}
```

```

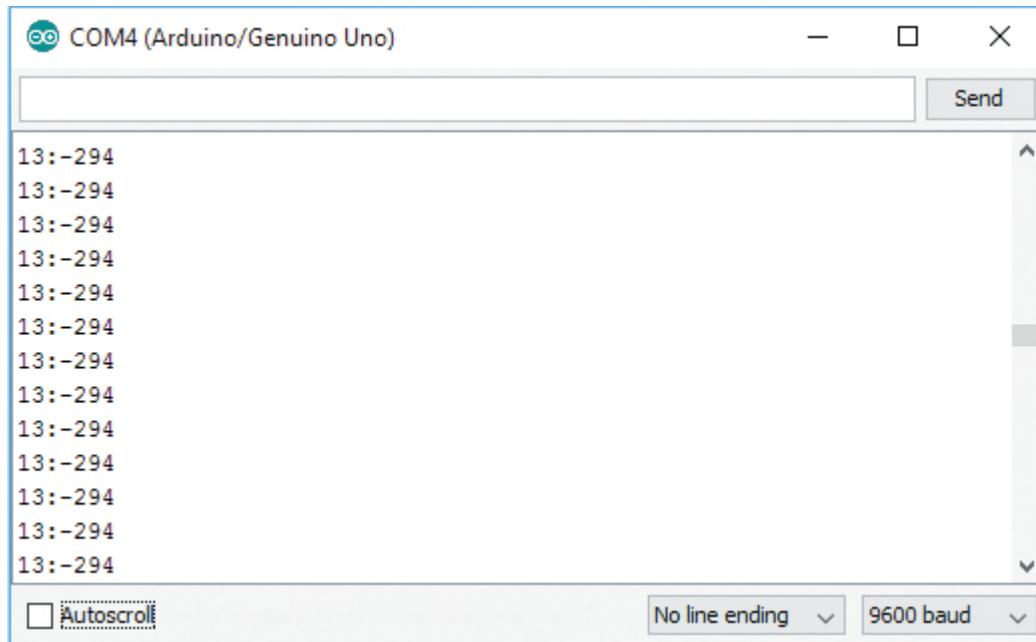
Serial.print(mag.YAxis);
Serial.print(":");
Serial.print(minX);
Serial.print(":");
Serial.print(maxX);
Serial.print(":");
Serial.print(minY);
Serial.print(":");
Serial.print(maxY);
Serial.print(":"); */
Serial.print(offX);
Serial.print(":");
Serial.print(offY);
Serial.print("\n");
}

```

---

3. We only need the X and Y `Serial.print` lines in this last bunch of `Serial.print` commands, so comment out the `Serial.print` lines of the sketch shown in bold. Upload the sketch to the Arduino and open the Serial Monitor. A series of numbers will display, as shown in Figure 7-3.

**FIGURE 7-3:** The calibration numbers will be shown in the IDE Serial Monitor window.

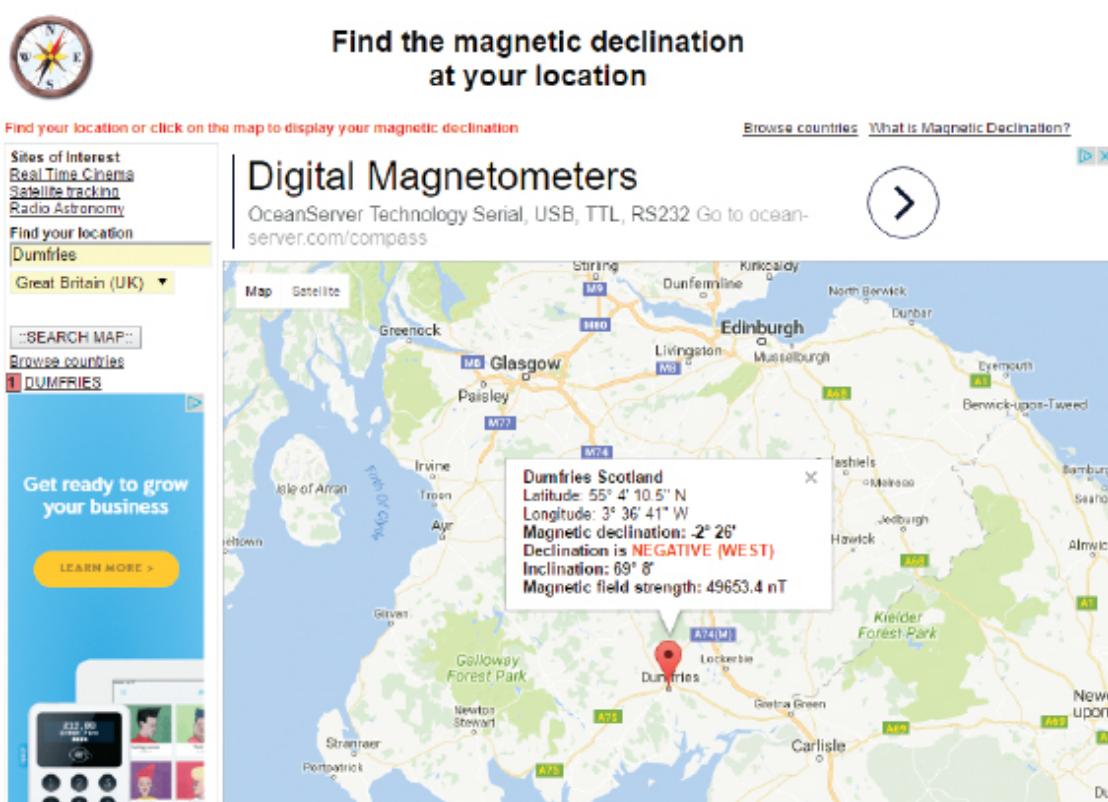


4. Rotate the sensor 360 degrees while it's connected to the Arduino IDE Serial Monitor, and you should see two digits displayed; in

Figure 7-3, they're 13 and -294. You'll need these calibration numbers in the sketch later, so make a note of them.

5. You can improve the accuracy of your compass by finding the *magnetic declination* for your location. The magnetic declination, or variation, is the angle on the horizontal plane between magnetic north (where a compass points) and true north (the direction toward the geographic North Pole). You can find your magnetic declination by visiting <http://www.magnetic-declination.com/> and entering your location in the search bar at the top left. Your result will appear as shown in Figure 7-4.

**FIGURE 7-4:** The magnetic declination for your location can be found at <http://www.magnetic-declination.com/>.



6. The values you need are the magnetic declination and the inclination; in Figure 7-4, they're  $-2^\circ 26'$  and NEGATIVE (WEST), respectively, but yours will be different. Record these values too, as we'll use them in the sketch at the end of the project

—with one minor change. For example, my values were  $-2$  and  $26$ . We don't put the negative (minus) sign before the first value but instead put it after, like so:

```
float declinationAngle = (2 - (26.0 / 60.0)) / (180 / M_PI);
```

If your location's declination were POSITIVE (WEST), then you would add the positive (plus) sign instead:

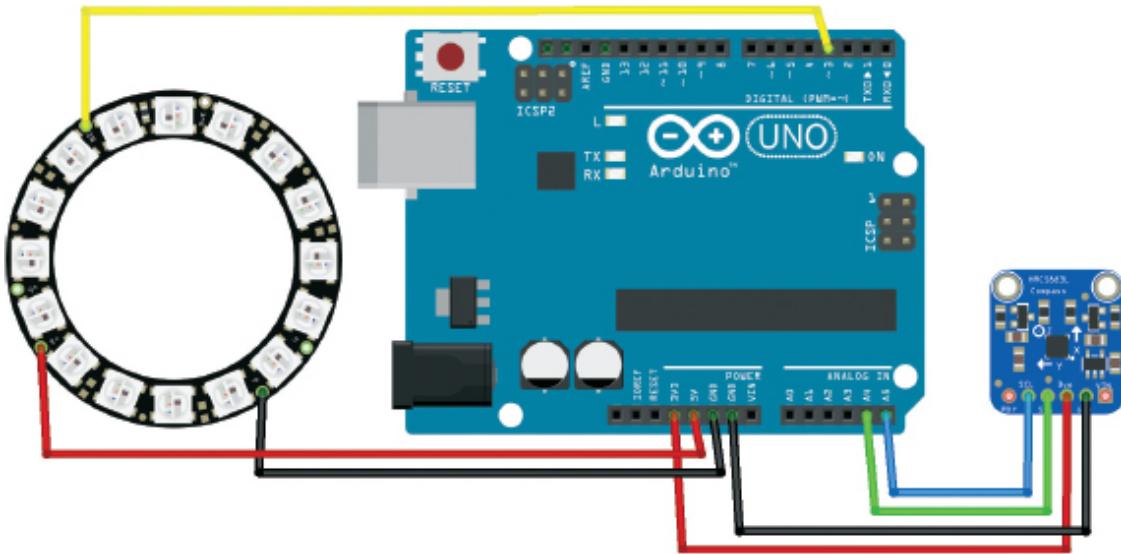
```
float declinationAngle = (2 + (26.0 / 60.0)) / (180 / M_PI);
```

Next, add the NeoPixel ring to the Arduino by connecting V on the NeoPixel to  $+5V$  on the Arduino, GND to GND, and In on the NeoPixel to pin 3 on the Arduino.

NEOPIXEL	ARDUINO
V	+5V
GND	GND
In	Pin 3

7. Check your setup against the circuit diagram in Figure 7-5, and then upload the code in “The Sketch” below.

**FIGURE 7-5:** The circuit diagram for the NeoPixel compass



## THE SKETCH

First we call on the Wire, FastLED, and HMC5883L libraries. The Wire library is installed with the Arduino IDE, but you need to add the others. Download them in the book’s resources at <http://www.nostarch.com/arduinohandbook2/>, and follow the guide in the primer for more information on adding libraries.

Next we declare the number of LEDs on the NeoPixel ring (16) and assign pin 3 on the Arduino to control it. We then call on a number of settings in the HMC5883L library to control the compass module. At ① we add the compass offset values for  $x$  and  $y$ , which should match your calibration from Step 4 earlier; mine were 13, -294, respectively. At ② we add the magnetic declination from Step 6. Again, remember to change it to the one for your location.

The next set of calculations allows the sensor to map to a 360-degree rotation. Then we set the LEDs on the NeoPixel to move depending on the readings of the sensor to point north. Three LEDs are lit: one red LED that points north and a green LED on either side of it. The compass is best used outdoors with the module, away from any strong electrical or magnetic sources, and should be powered from a battery pack rather than a USB connection.

```
-----  
// Code by brainy-bits.com and used with kind permission  
// https://brainy-bits.com/tutorials/find-your-way-using-the-hmc5883l/  
  
#include <Wire.h>  
#include "FastLED.h"  
#include <HMC5883L.h>  
  
#define NUM_LEDS 16 // Number of LEDs on Ring  
#define DATA_PIN_RING 3 // Pin 3 connected to RGB Ring  
  
CRGB leds_RING[NUM_LEDS];  
  
HMC5883L compass;  
int fixedHeadingDegrees; // Used to store Heading value  
  
void setup() {  
    Serial.begin(9600);  
    Wire.begin(); //Setup I2C  
    // Set up the FastLED library with the neopixel ring data  
    FastLED.addLeds<NEOPIXEL,DATA_PIN_RING>(leds_RING, NUM_LEDS);  
  
    // Set measurement range  
    compass.setRange(HMC5883L_RANGE_1_3GA);  
  
    // Set measurement mode  
    compass.setMeasurementMode(HMC5883L_CONTINOUS);  
  
    // Set data rate  
    compass.setDataRate(HMC5883L_DATARATE_30HZ);  
  
    // Set number of samples averaged  
    compass.setSamples(HMC5883L_SAMPLES_8);  
  
    // Set calibration offset. See HMC5883L_calibration.ino  
❶ compass.setOffset(13, -224);  
}  
  
void loop() {  
    Vector norm = compass.readNormalize();  
  
    // Calculate heading  
    float heading = atan2(norm.YAxis, norm.XAxis);  
  
    // Set declination angle on your location and fix heading  
    // Find your declination on http://magnetic-declination.com/  
    // (+) Positive or (-) for negative  
    // For Dumfries, Scotland declination angle is -2 '26W (negative)  
    // Formula: (deg + (min / 60.0)) / (180 / M_PI);  
    float declinationAngle = (2.0 - (26.0 / 60.0)) / (180 / M_PI);
```

```

② heading -= declinationAngle;

// Correct for heading < 0deg and heading > 360deg
if (heading < 0) {
    heading += 2 * PI;
}

if (heading > 2 * PI) {
    heading -= 2 * PI;
}

// Convert to degrees
float headingDegrees = heading * 180 / M_PI;

// To fix rotation speed of HMC5883L compass module
if (headingDegrees >= 1 && headingDegrees < 240) {
    fixedHeadingDegrees = map(headingDegrees * 100, 0, 239 * 100, 0, 179 * 100) /
100.00;
}
else {
    if (headingDegrees >= 240) {
        fixedHeadingDegrees = map(headingDegrees*100, 240*100, 360*100, 180*100,
360*100) / 100.00;
    }
}

int headvalue = fixedHeadingDegrees / 18;
int ledtoheading = map(headvalue, 0, 15, 15, 0);

// Clear the ring
FastLED.clear();

// New heading
if (ledtoheading == 0) {
    leds_RING[15] = CRGB::Red;
    leds_RING[0] = CRGB::Green;
    leds_RING[14] = CRGB::Green;
}
else {
    if (ledtoheading == 15) {
        leds_RING[0] = CRGB::Red;
        leds_RING[15] = CRGB::Green;
        leds_RING[1] = CRGB::Green;
    }
    else {
        leds_RING[ledtoheading] = CRGB::Red;
        leds_RING[ledtoheading+1] = CRGB::Green;
        leds_RING[ledtoheading-1] = CRGB::Green;
    }
}
}

```

```
FastLED.setBrightness(50);
FastLED.show();
delay(100);
}
```

---

## TROUBLESHOOTING

**Q.** *The code compiles, but the RGB LEDs do not light up as expected.*

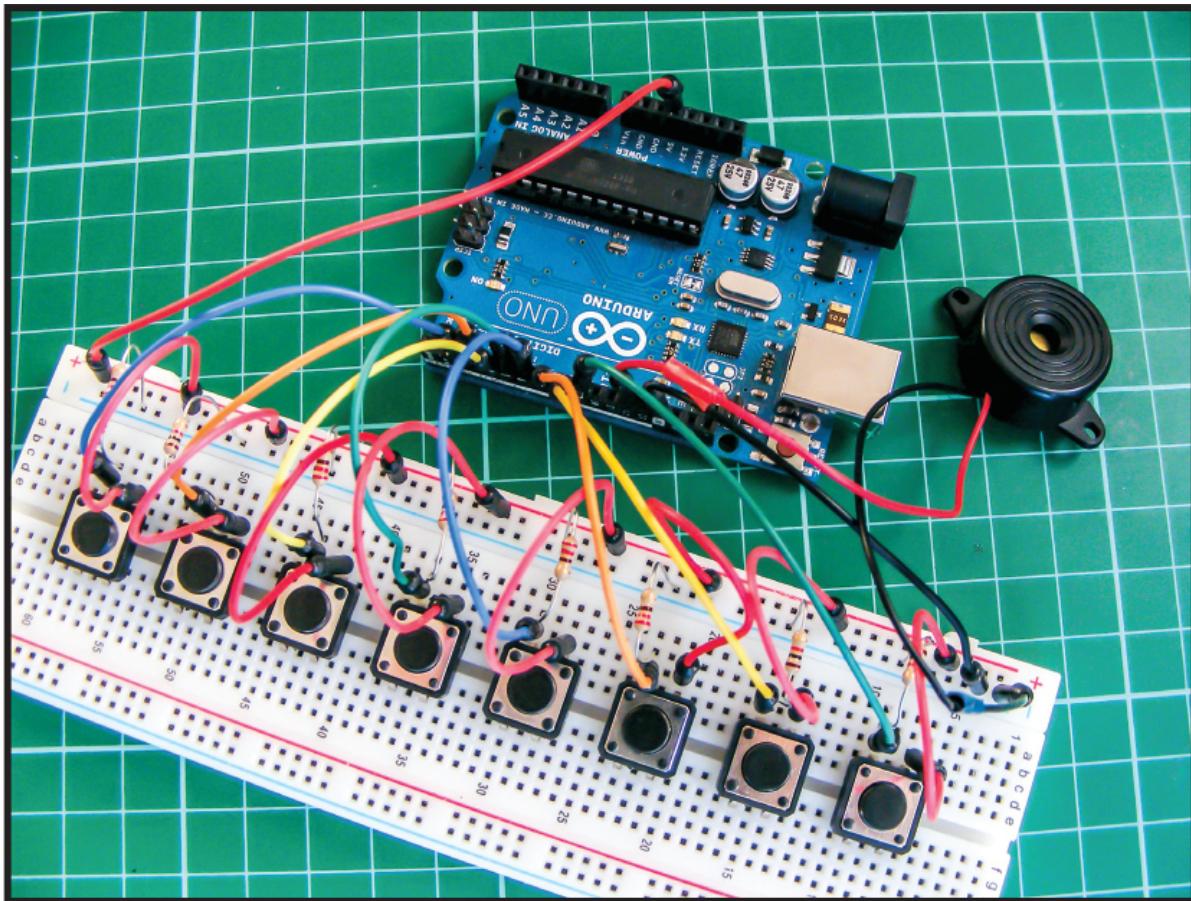
- If no LEDs are lit, double-check your wiring, particularly that the data pin of the NeoPixel is connected to pin 3 on the Arduino.
- Check that your power for the NeoPixel is connected to GND and +5V. The compass module should be connected to GND and +3.3V. The Arduino should be powered by your battery pack, not the USB cable from your PC.
- Make sure you have calibrated the module and entered the values using the steps shown earlier. The compass module should be held horizontally and in line with the RGB ring. The ring and the module should always be moved together.
- The module is best used outdoors, as it is very sensitive to metal and electrical interference.
- Try to keep the power for your Arduino and the sensor as far apart as possible to avoid interference.

# Sound

# 8

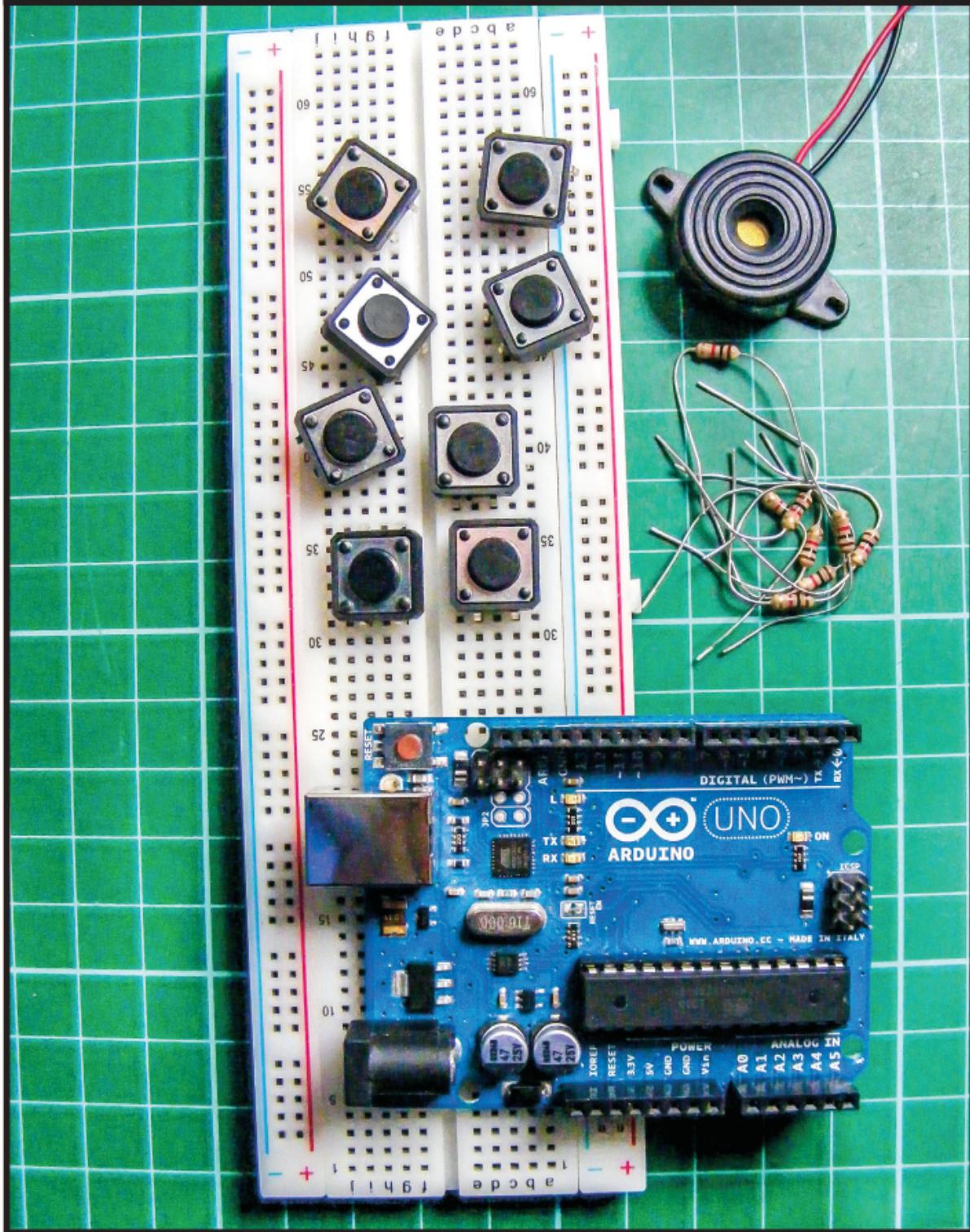
## Arduino Piano

In this project we'll use some momentary pushbuttons and a piezo sounder to create a simple piano.



**COST: \$**

**TIME: 30 MINUTES**



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**Piezo sounder**

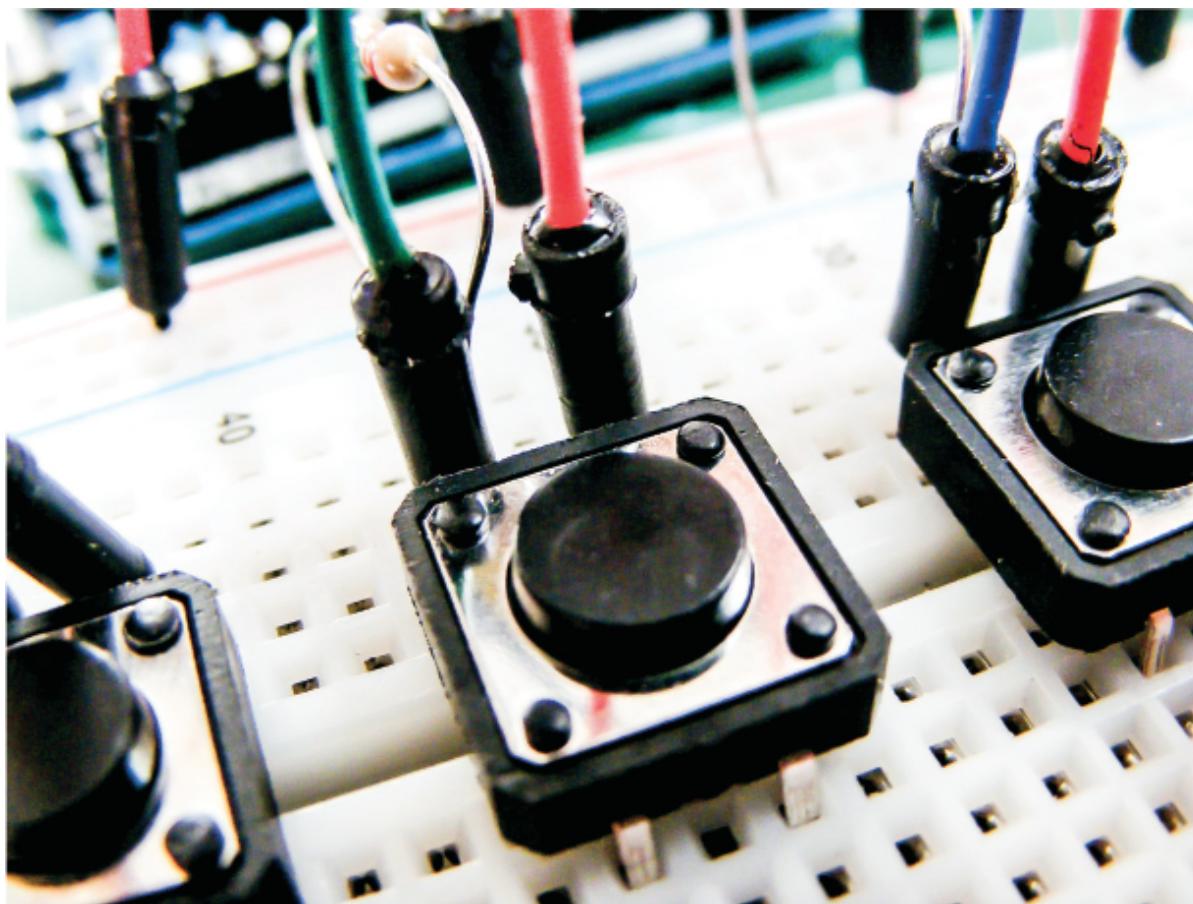
**8 momentary tactile pushbuttons**

**8 1k-ohm resistors**

## HOW IT WORKS

Each pushbutton in our project (see Figure 8-1) is connected to an Arduino pin, and when the pushbutton is pressed, the piezo sounder will emit one of eight notes.

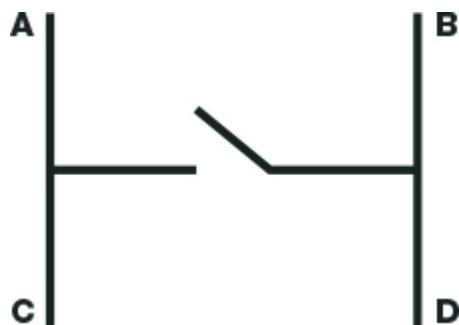
**FIGURE 8-1:** A momentary pushbutton and its circuit



When pressed, a pushbutton completes a circuit, turning it on. As soon as the button is released, the connection will spring back and break that circuit, turning it off. The pushbutton switch is also known as a *momentary* or *normally open switch*, and is used in, for example, computer keyboards. This is in contrast to a *toggle switch*, which stays either on or off until you toggle it to the other position, like a light switch.

This type of pushbutton has four pins, but you generally use only two at a time for connection. We're using the top pins in this project so it's easier to reach the button and play a tune, although the two unused pins at the bottom would do the same job. As Figure 8-2 shows, the pins work in a circuit. Pins A and C are always connected, as are pins B and D. When the button is pressed, the circuit is complete.

**FIGURE 8-2:** A pushbutton's incomplete circuit



The Arduino piano uses a piezo sounder, shown in Figure 8-3, to create frequencies that resemble recognizable notes. Piezo sounders, or just piezos for short, are inexpensive buzzers often used in small toys. A piezo element without its plastic housing looks like a gold metallic disc with connected positive (typically red) and negative (typically black) wires. A piezo is capable only of making a clicking sound, which we create by applying voltage.

**FIGURE 8-3:** A piezo sounder



We can make recognizable notes by getting the piezo to click hundreds of times a second at a particular frequency, so first we need to know the frequency of the different tones we want. Table 8-1 shows the notes and their corresponding frequencies. *Period* is the duration of the cycle, in microseconds, at which the frequency is created. For example, to get a C note (261 Hz), we need the piezo to cycle at a period of 3,830 microseconds. We halve the period to get the `timeHigh` value, which is used in the code to create the note. (The tone is caused by the piezo being turned on and off very quickly, so the time that the piezo is on, or `HIGH`, is half the period.)

**TABLE 8-1:** The Musical Notes and Frequencies Used in the Code

NOTE	FREQUENCY	PERIOD	TIMEHIGH
c	261 Hz	3,830	1915
d	294 Hz	3,400	1700

NOTE	FREQUENCY	PERIOD	TIMEHIGH
e	329 Hz	3,038	1519
f	349 Hz	2,864	1432
g	392 Hz	2,550	1275
a	440 Hz	2,272	1136
b	493 Hz	2,028	1014
c	523 Hz	1,912	956

## THE BUILD

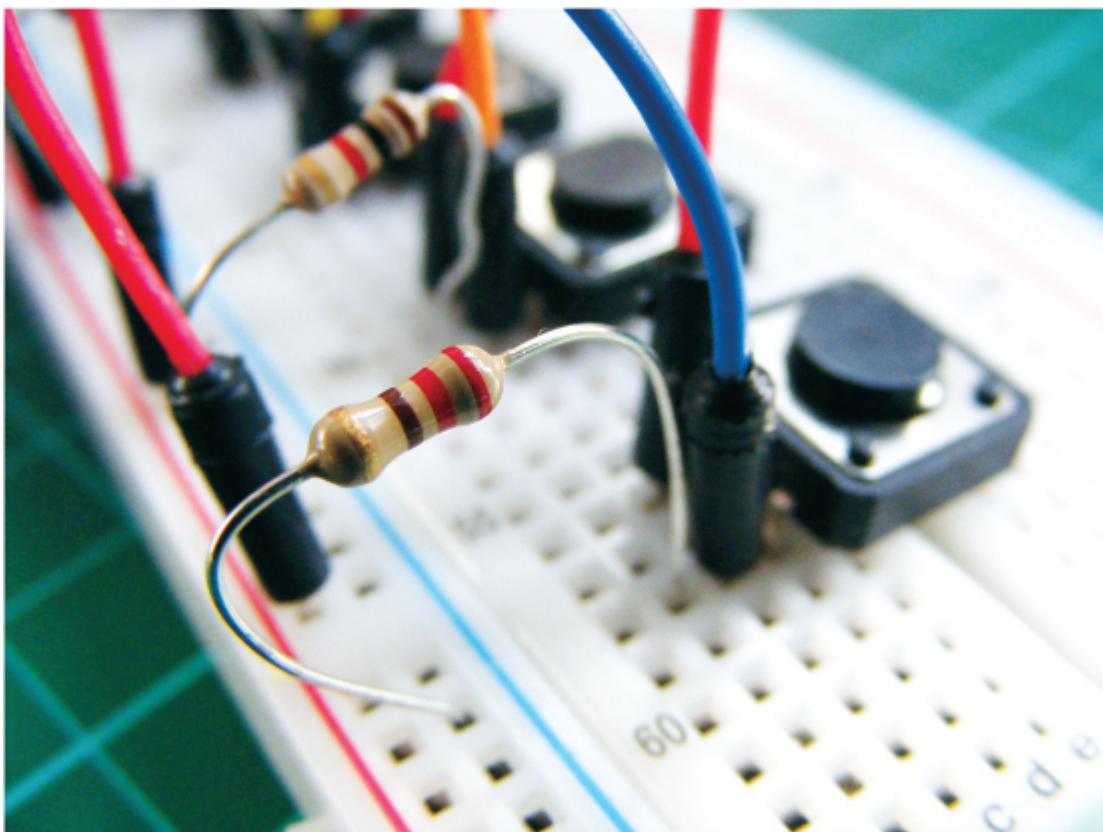
1. Insert the momentary pushbuttons into the breadboard with the pins straddling the center break of the breadboard.
2. Looking at the breadboard face on, number the pushbuttons 1–8 from left to right. Connect the top-left pin (A) of pushbutton 1 to Arduino pin 2 using a jumper wire. Connect the top-left pins of the other pushbuttons to the Arduino as shown here.

PUSHBUTTON	NOTE	ARDUINO
1	c	2
2	d	3
3	e	4
4	f	5
5	g	6
6	a	7

PUSHBUTTON	NOTE	ARDUINO
7	b	8
8	c	9

3. Insert a 1k-ohm resistor into the breadboard in line with the first pushbutton's top-left pin, as shown in Figure 8-4, and connect the other side of the resistor to the GND rail of the breadboard. Repeat this for the other pushbuttons. The resistor pulls the switch to GND when the button is not pressed to tell the Arduino that it's not in a positive state; when the button is pressed, the positive power sounds the corresponding note.

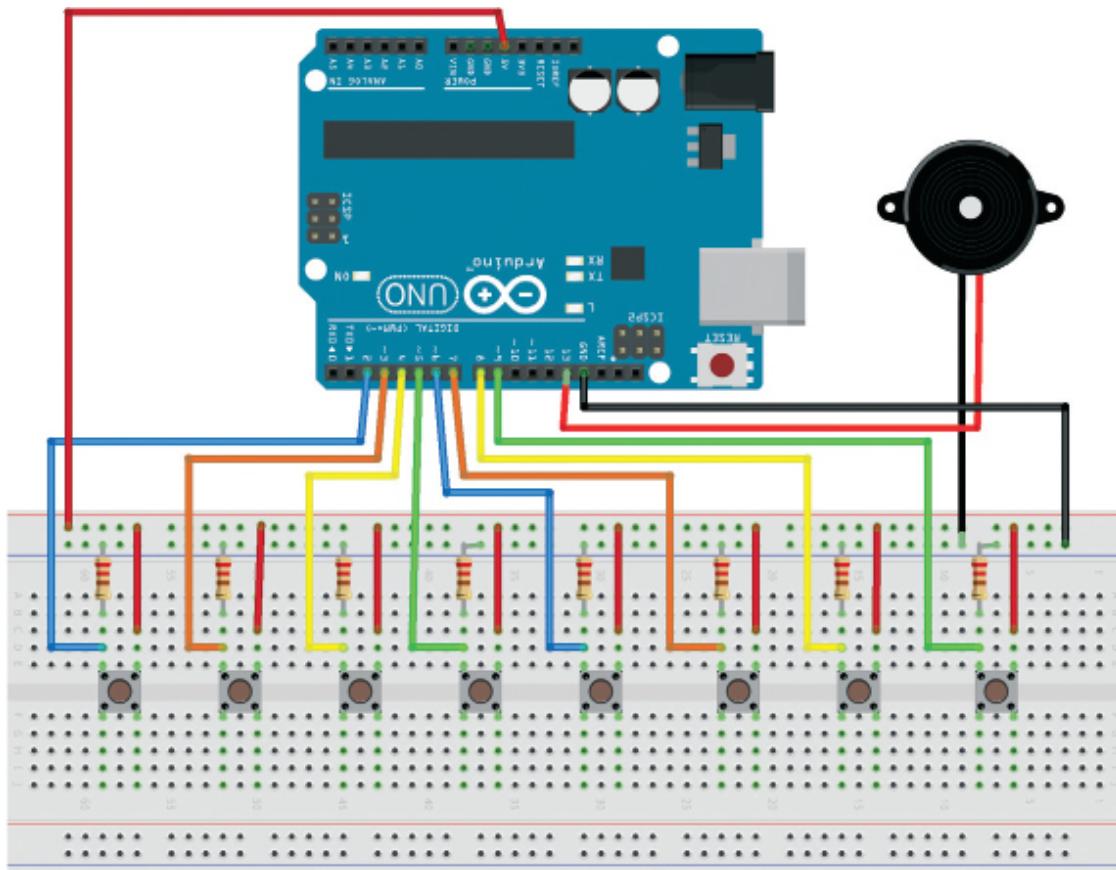
**FIGURE 8-4:** A 1k-ohm resistor connects the pushbutton pin to GND.



4. Connect the top-right pin (B) of each of the pushbuttons to the positive breadboard power rail using jumper wire.

5. Connect the piezo's red wire to Arduino pin 13 and its black wire to the GND rail of the breadboard, then connect the power rails to GND and +5V on the Arduino.
6. Make sure that your setup matches the circuit diagram in Figure 8-5, and then upload the code in "The Sketch" on page 74.

**FIGURE 8-5:** The circuit diagram for the Arduino piano



## THE SKETCH

The sketch first defines the pin that the piezo sounder is connected to and the pins for the pushbuttons. A value is defined for each pushbutton, and a tone is assigned to correspond with that value. The pushbuttons are set as inputs and the piezo sounder as an output. The loop cycle checks each button, playing the corresponding tone for as long as the button is held down. Only one note can be played at a time

because each note requires the loop to begin again, so when the button is released, the piezo sounder stops playing the tone and the loop starts over.

---

```
int speakerPin = 13; // Piezo defined as pin 13
int key_c = 2; // Define Arduino pins for the keys
int key_d = 3;
int key_e = 4;
int key_f = 5;
int key_g = 6;
int key_a = 7;
int key_b = 8;
int key_C = 9;

// Value for each key press
int keypress_c = 0; int keypress_d = 0; int keypress_e = 0;
int keypress_f = 0; int keypress_g = 0; int keypress_a = 0;
int keypress_b = 0; int keypress_C = 0;

// Define the frequency of each note
int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };
//           'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'
int keytone = 0; // Give a value for keytone

void setup() {
    pinMode(key_c, INPUT); // Set up key pins as inputs
    pinMode(key_d, INPUT);
    pinMode(key_e, INPUT);
    pinMode(key_f, INPUT);
    pinMode(key_g, INPUT);
    pinMode(key_a, INPUT);
    pinMode(key_b, INPUT);
    pinMode(key_C, INPUT);
    pinMode(speakerPin, OUTPUT); // Set up piezo pin as an output
}

// Start a loop to read the press of each key
void loop() {
    keypress_c = digitalRead(key_c); keypress_d = digitalRead(key_d);
    keypress_e = digitalRead(key_e); keypress_f = digitalRead(key_f);
    keypress_g = digitalRead(key_g); keypress_a = digitalRead(key_a);
    keypress_b = digitalRead(key_b); keypress_C = digitalRead(key_C);

    // And if the key press is HIGH, play the corresponding tone
    if ((keypress_c == HIGH) || (keypress_e == HIGH) ||
        (keypress_g == HIGH) || (keypress_d == HIGH) ||
        (keypress_f == HIGH) || (keypress_a == HIGH) ||
        (keypress_b == HIGH) || (keypress_C == HIGH))
    {
```

```

if (keypress_c == HIGH) {
    keytone = tones[0];
}
if (keypress_d == HIGH) {
    keytone = tones[1];
}
if (keypress_e == HIGH) {
    keytone = tones[2];
}
if (keypress_f == HIGH) {
    keytone = tones[3];
}
if (keypress_g == HIGH) {
    keytone = tones[4];
}
if (keypress_a == HIGH) {
    keytone = tones[5];
}
if (keypress_b == HIGH) {
    keytone = tones[6];
}
if (keypress_c == HIGH) {
    keytone = tones[7];
}
digitalWrite(speakerPin, HIGH); // Turn on piezo to play tone
delayMicroseconds(keytone);
digitalWrite(speakerPin, LOW); // Turn off after a short delay
delayMicroseconds(keytone);
}
else { // If no key is pressed, piezo remains silent
    digitalWrite(speakerPin, LOW);
}
}

```

---

## TROUBLESHOOTING

**Q.** *The code compiles, but some or all of the buttons do not produce a tone.*

- If the piezo sounder makes no noise at all, check that the piezo's red wire is connected to pin 13 and its black wire to GND on the breadboard. Make sure you have connected GND on the Arduino to the correct breadboard power rail and that the Arduino has power connected.
- If only some buttons make a sound, recheck the wiring for the pushbuttons that are silent. It's easy to misalign the jumper wires in

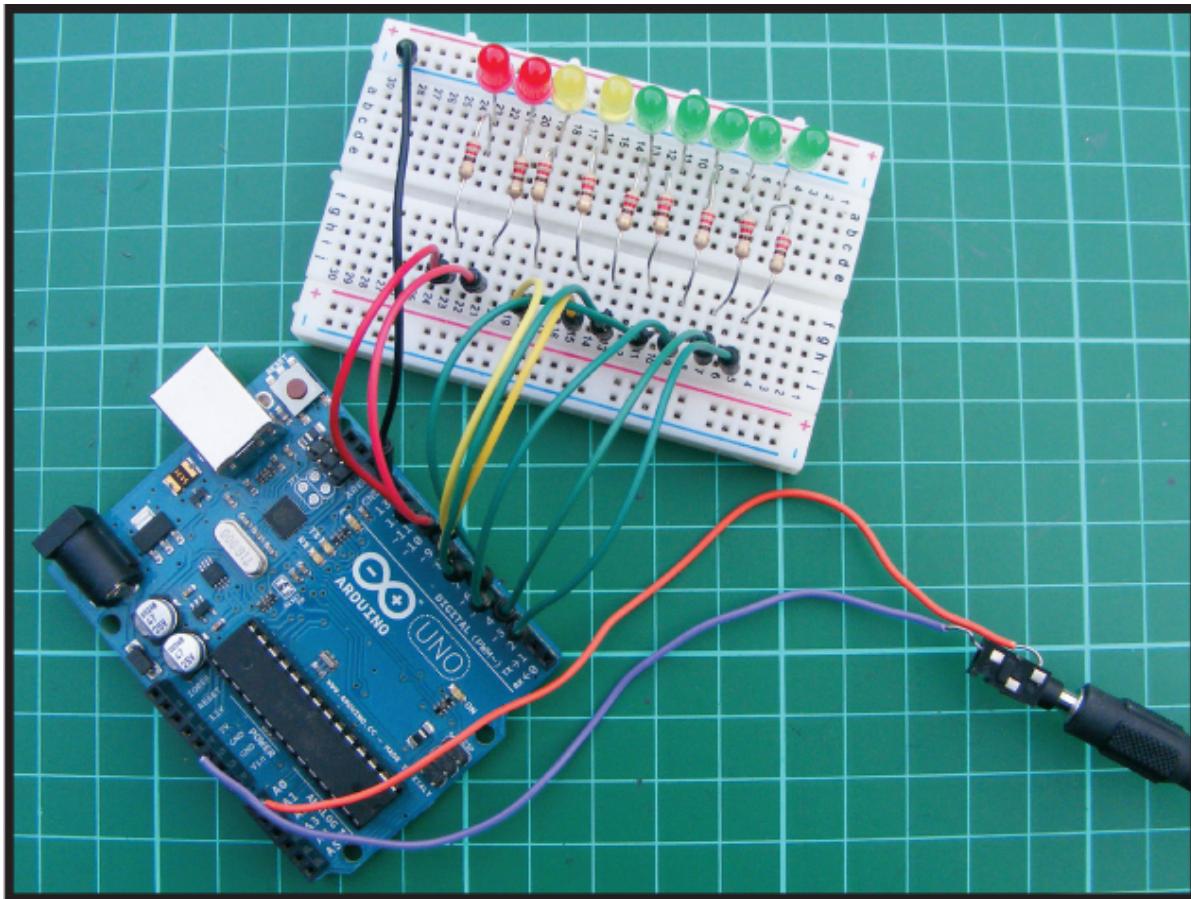
the breadboard so they don't actually line up in the row with the pushbutton pins.

- If you still have an issue, try swapping the offending pushbutton for one you know works; if this resolves your problem, then your original pushbutton may have been faulty.

# 9

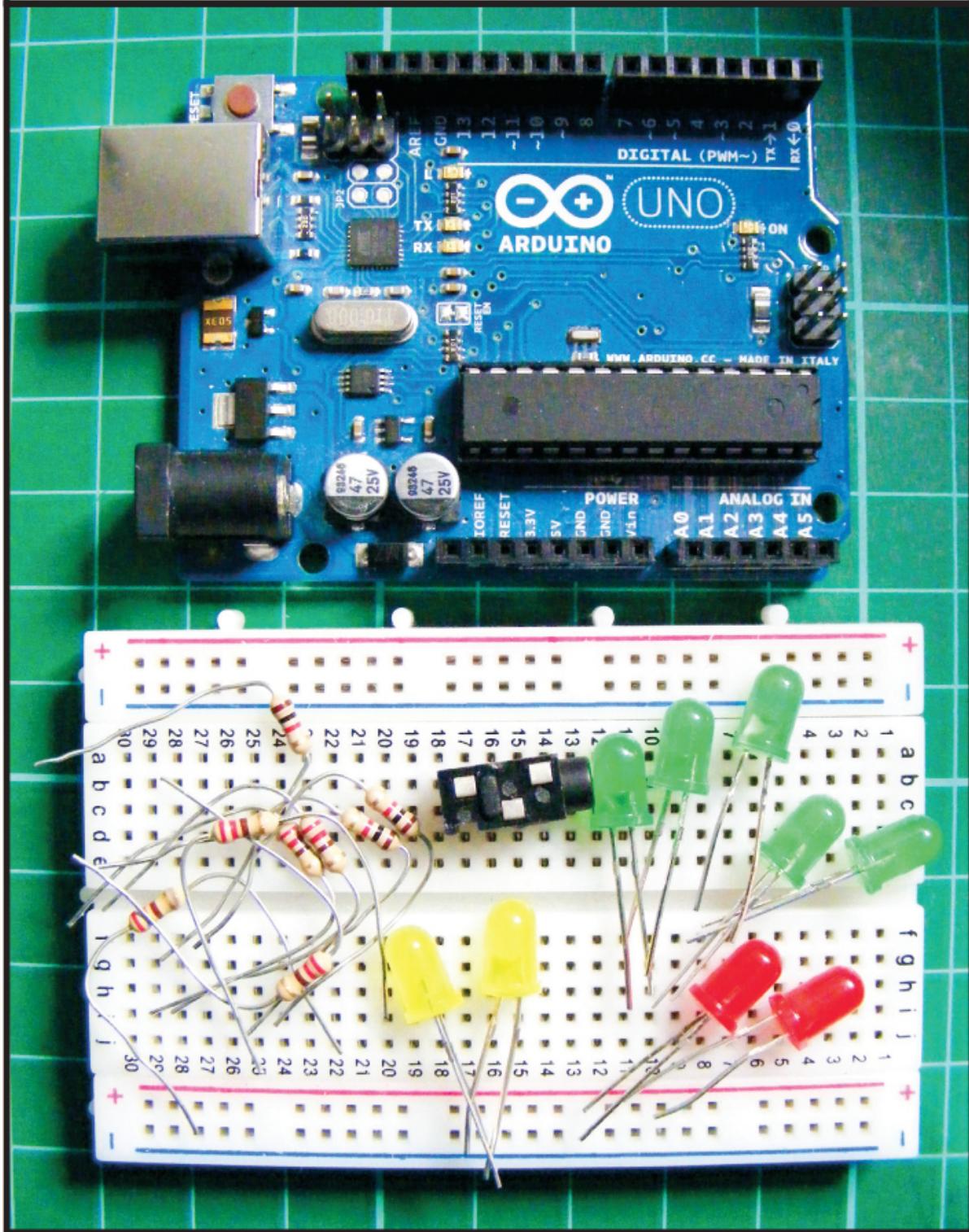
## Audio LED Visualizer

In this project we'll use a sound sensor that will light a series of LEDs depending on the beat and volume of the sound it detects.



COST: \$

TIME: 30 MINUTES



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Solid-core wires with ends stripped**

**Jumper wires**

**2 red LEDs**

**2 yellow LEDs**

**5 green LEDs**

**9 220-ohm resistors**

**3.5 mm female headphone jack**

## HOW IT WORKS

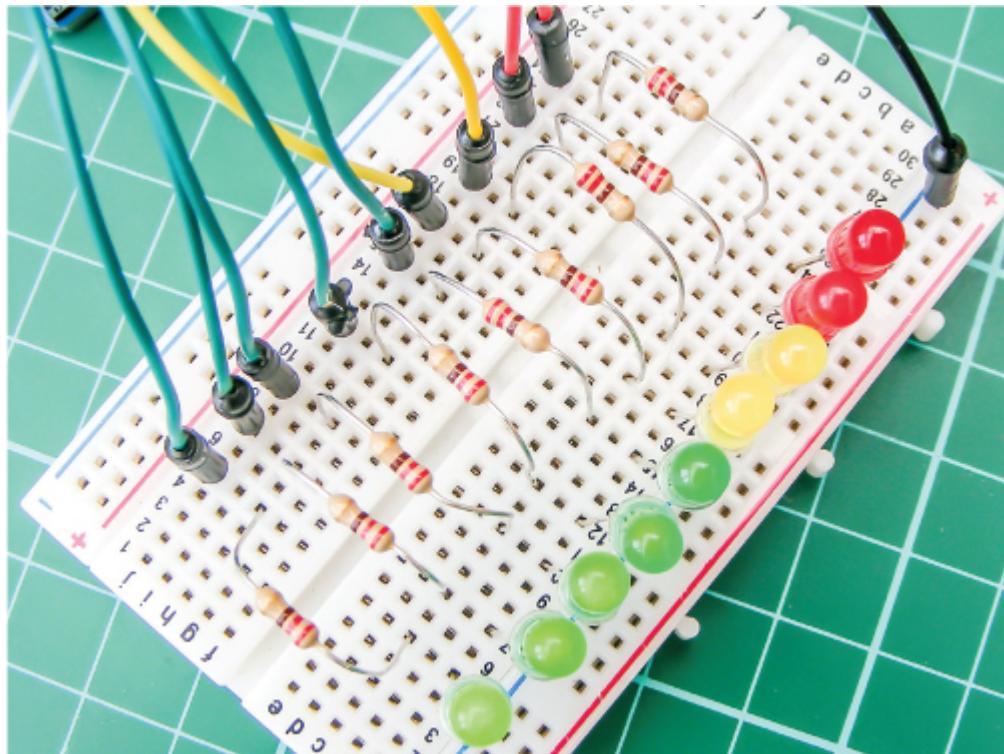
In Project 2 we created an LED night-light that was controlled by a light sensor. This project is similar, but the LEDs will be controlled by sound. We'll connect a headphone jack to the Arduino, hook the jack up to an MP3 player, and watch the lights "dance" to the music. The signal from the MP3 player is picked up by the headphone jack and received as pulses by the Arduino A0 pin. The pattern of the pulses depends on the beat and volume of the music. The Arduino then sends power to the LEDs in direct response to the pattern of the music. As an alternative to using the MP3 player, you could add a microphone and have your own voice visualized in colored lights.

## THE BUILD

1. Place the LEDs into the breadboard with the short, negative legs in the GND rail. Connect the GND rail on the breadboard to Arduino GND.
2. Insert a 220-ohm resistor for each LED, making sure the resistors straddle the center break, and connect one leg to each positive LED leg (see Figure 9-1). Connect the other leg of each resistor to Arduino digital pins 2 through 10 with jumper wires, as shown in the following table.

LED	ARDUINO
Positive leg	Digital pins 2–10 (via resistor)
Negative leg	GND

**FIGURE 9-1:** A resistor is required between the LEDs and power.



### NOTE

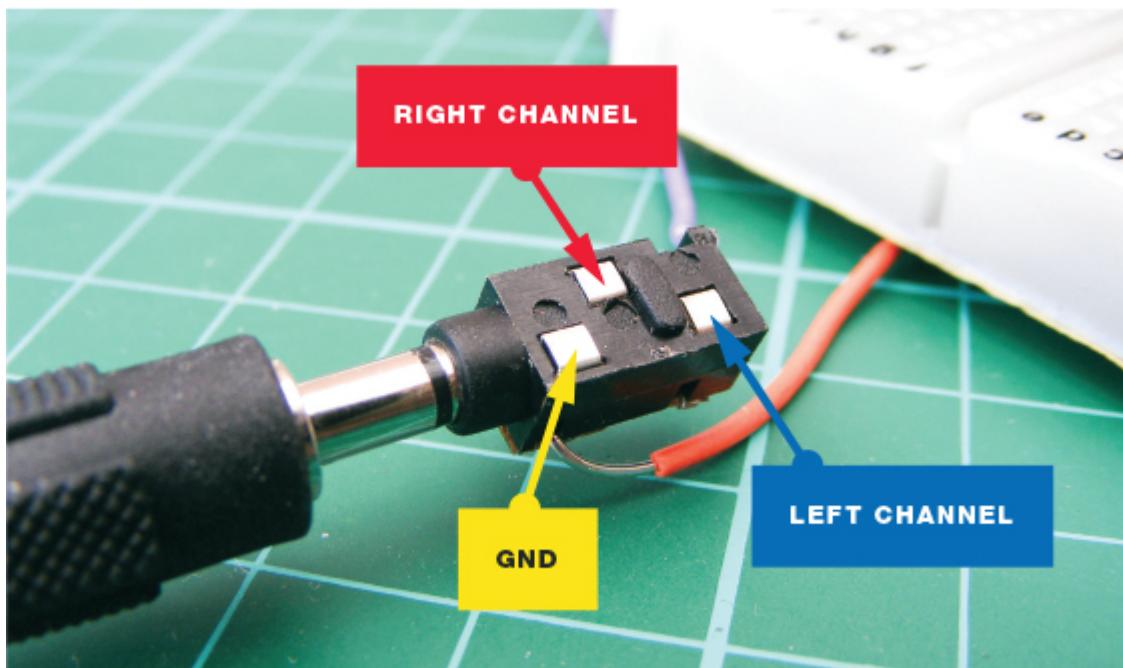
*This headphone jack was reclaimed from a radio bought in a dollar store, but if you can find one to purchase, that will work too. On the headphone jack, the pins are GND, right channel, and left channel.*

3. Connect the ground pin of the headphone jack directly to GND, and the left channel of the jack to Arduino pin A0, as outlined in the following table. You could use jumper wire for this, but I've used solid-core wire and stripped the ends for connections.

Stranded wire is too thin and won't connect easily to the Arduino pins. (See Figure 9-2 for the positions of the jack pins.)

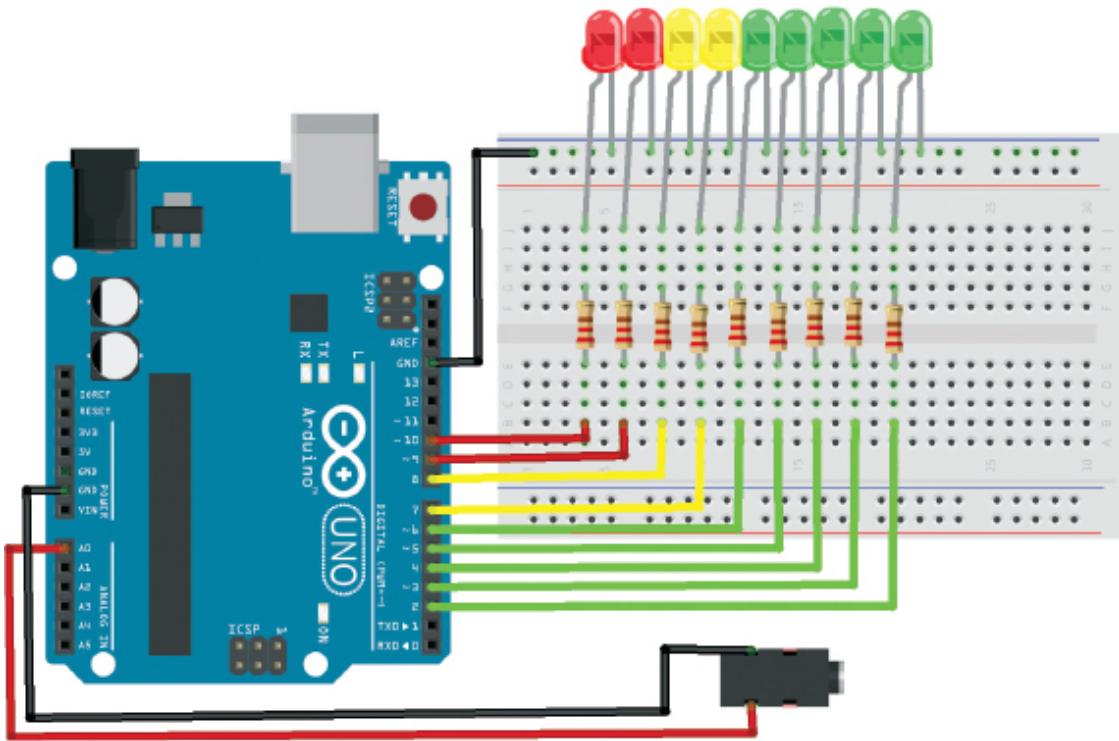
HEADPHONE JACK	ARDUINO
Ground	GND
Left channel	A0

**FIGURE 9-2:** 3.5 mm headphone jack with MP3 player jack plugged in



4. Check your setup against the circuit diagram in Figure 9-3, and then upload the code in “The Sketch” on page 81.

**FIGURE 9-3:** The circuit diagram for the audio LED visualizer



Plug your MP3 player into the headphone jack for audio input. The LEDs will dance to the beat and volume of your music!

## THE SKETCH

The sketch first sets the Arduino pins connected to the LEDs, pins 2–10, as outputs. The input in this sketch is the signal from the MP3 player, received through the headphone jack, which is read by analog pin A0. The music sent by the player is picked up as a series of pulses by A0, and the volume and beat of the music determine how the LEDs light up. The louder the music, the more LEDs will light; and the faster the music's beat, the faster the LEDs will flash.

---

```
// Used with kind permission from James Newbould
int led[9] = {2, 3, 4, 5, 6, 7, 8, 9, 10}; // Pins connected to LEDs
int leftChannel = A0; // Pin connected to headphone jack
int left, i; // Create a variable for left and i
void setup() {
  for (i = 0; i < 9; i++)
    pinMode(led[i], OUTPUT); // Set LEDs as output
}
```

```

void loop() { // Light LEDs from left to right and back again
    // depending on the value from A0
    left = analogRead(leftChannel); // Read left value
    left = left / 10; // Set level of sensitivity between 1 and 50
    if (left == 0) {
        for (i = 0; i < 9; i++) { // If value is low, turn off LED
            digitalWrite(led[i], LOW);
        }
    }
    else { // Or else turn on LEDs in sequence
        for (i = 0; i < left; i++) {
            digitalWrite(led[i], HIGH);
        }
        for (i = i; i < 9; i++) {
            digitalWrite(led[i], LOW);
        }
    }
}

```

---

## TROUBLESHOOTING

**Q.** *The code compiles, but some or all of the LEDs do not light up as expected.*

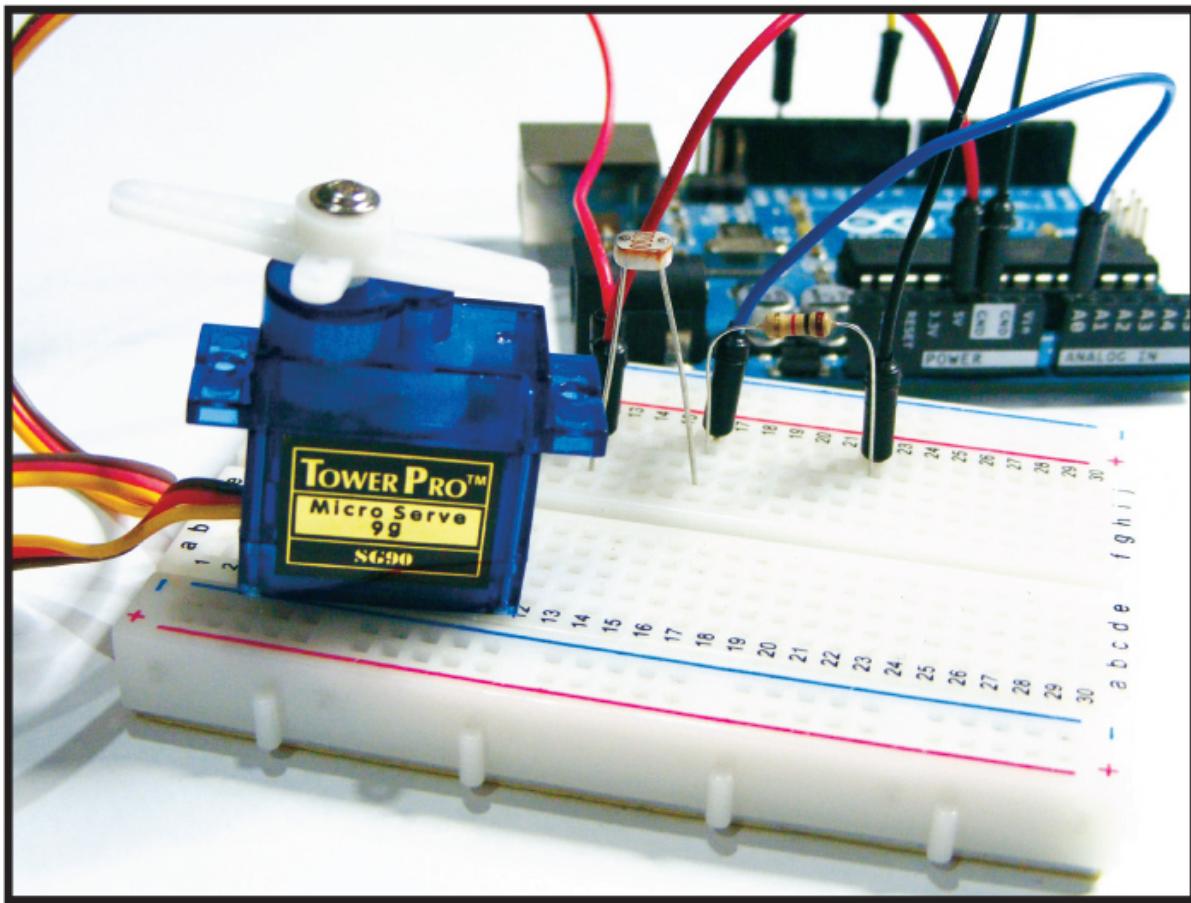
- If none of the LEDs light, make sure you've connected the GND wire from the Arduino to the correct breadboard power rail and that the Arduino has power connected.
- If only some LEDs light, check that the LEDs are inserted the correct way, with the longer leg connected to the positive power and the short leg to GND. LEDs have polarity, so they must be connected correctly. Check that each resistor is inserted fully and lines up in the same row as the corresponding LED leg.
- Make sure the LEDs are connected to the Arduino pins defined in the sketch and match the circuit diagram in Figure 9-3; the first part of the sketch defines pins 2–10 as outputs, so these should be used.
- If an LED still fails to light, it may be burned out or faulty. An easy way to check is to swap the LED with another in the sequence and see if that solves the issue. If you find that the LED works in another position, it means the resistor is either faulty or not inserted fully. Depending on the outcome, replace the LED or resistor with a functioning component.

# Motors

# 10

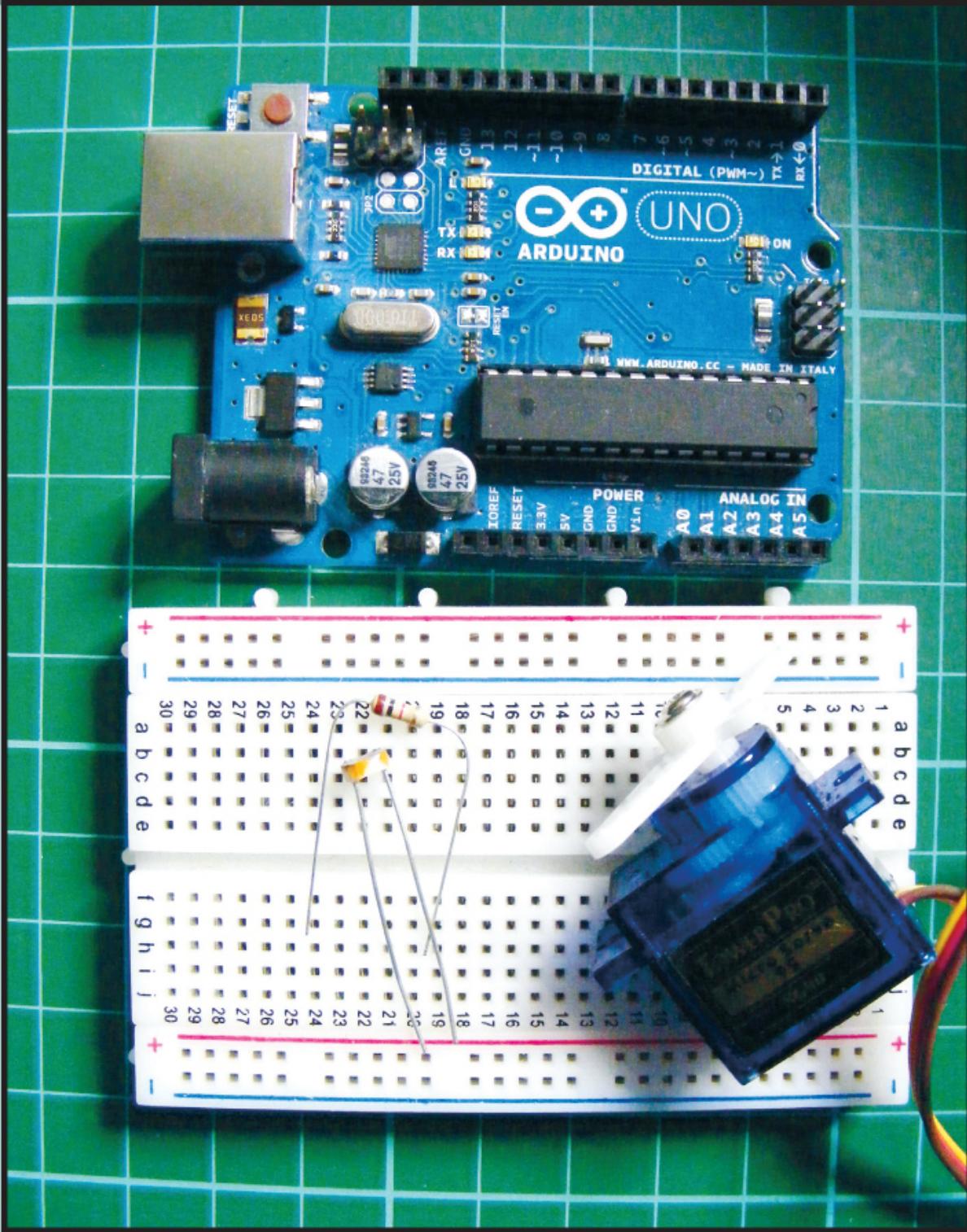
## Old-School Analog Dial

Old-fashioned analog displays have a certain charm. In this project I'll demonstrate how to make your own.



COST: \$\$

TIME: 20 MINUTES



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**Tower Pro SG90 9g servomotor**

**Photoresistor**

**10k-ohm resistor**

## LIBRARY REQUIRED

**Servo**

## HOW IT WORKS

Today, visual representations of measurements are usually displayed digitally on an LCD screen or with LED digits, but not that long ago analog dials were always used to show pressure, speed, and even time! The Arduino can detect a voltage input from a sensor, and we'll use that capability here to create a dial that the Arduino moves in response to the input received. We can use this dial in lots of ways to show measurements for different projects.

In this project, we'll use a photoresistor to measure light input, but you could easily swap in a water sensor to make a rain detector, or a gas sensor for a warning meter. A *photoresistor*, also referred to as a *light-dependent resistor*, produces a variable resistance depending on the amount of light the sensor detects, as discussed in Project 2.

The principles for adding an analog sensor are the same for whichever sensor you choose. Most sensors have three connections: ground, +5V, and a signal connection that connects to the analog A0 pin on the Arduino—this makes it easy to swap in a different sensor. The photoresistor is slightly different because it has only two connections, so one will go to power and one to A0.

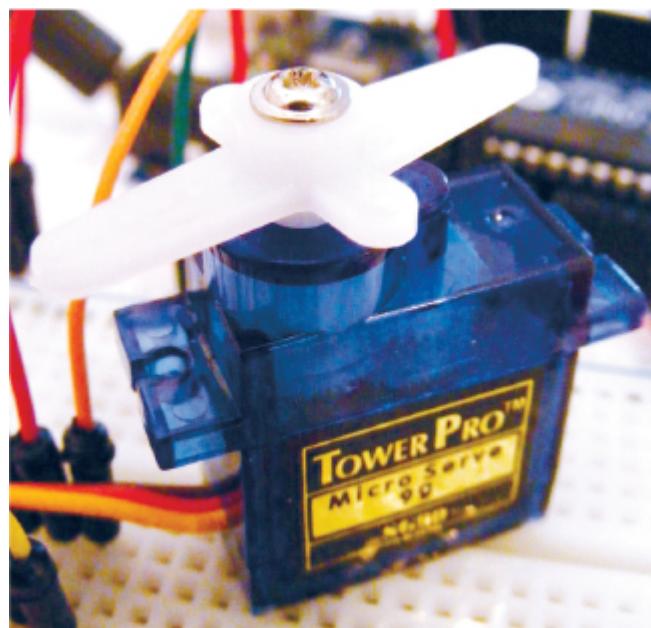
We'll use the sensor to measure light levels, and the Arduino will use that measurement to move the arm of a small servomotor (or “servo”

for short) to the corresponding angle. The angle of the motor arm indicates the strength of the light input.

A *servo*, shown in Figure 10-1, is a small, cheap, mass-produced motor used for small robotics and a variety of electronics tasks. The servo is controlled by three wires: ground (black or brown), power (red), and signal or control (typically orange, yellow, or white). Pulses are sent from the Arduino over the control wire via pulse width modulation (PWM; discussed in Project 5), and the input received by the photoresistor determines the angle of the servo's actuator arm. The servo expects a pulse every 20 milliseconds in order to retrieve the correct information about the angle.

The pulse width dictates the range of the servo's angular motion. Typically, a servo pulse width of 1.5 milliseconds sets the servo to its "neutral" position of 45 degrees, a pulse width of 1.25 milliseconds sets the angle to 0 degrees, and a pulse width of 1.75 milliseconds sets the angle to 90 degrees.

**FIGURE 10-1:** A servomotor



The physical limits of the arm angle and the timing of the servo hardware vary across brands and models, but in general a servo's angular

motion travels in the range of 90 to 180 degrees and the neutral position is almost always at 1.5 milliseconds.

## THE BUILD

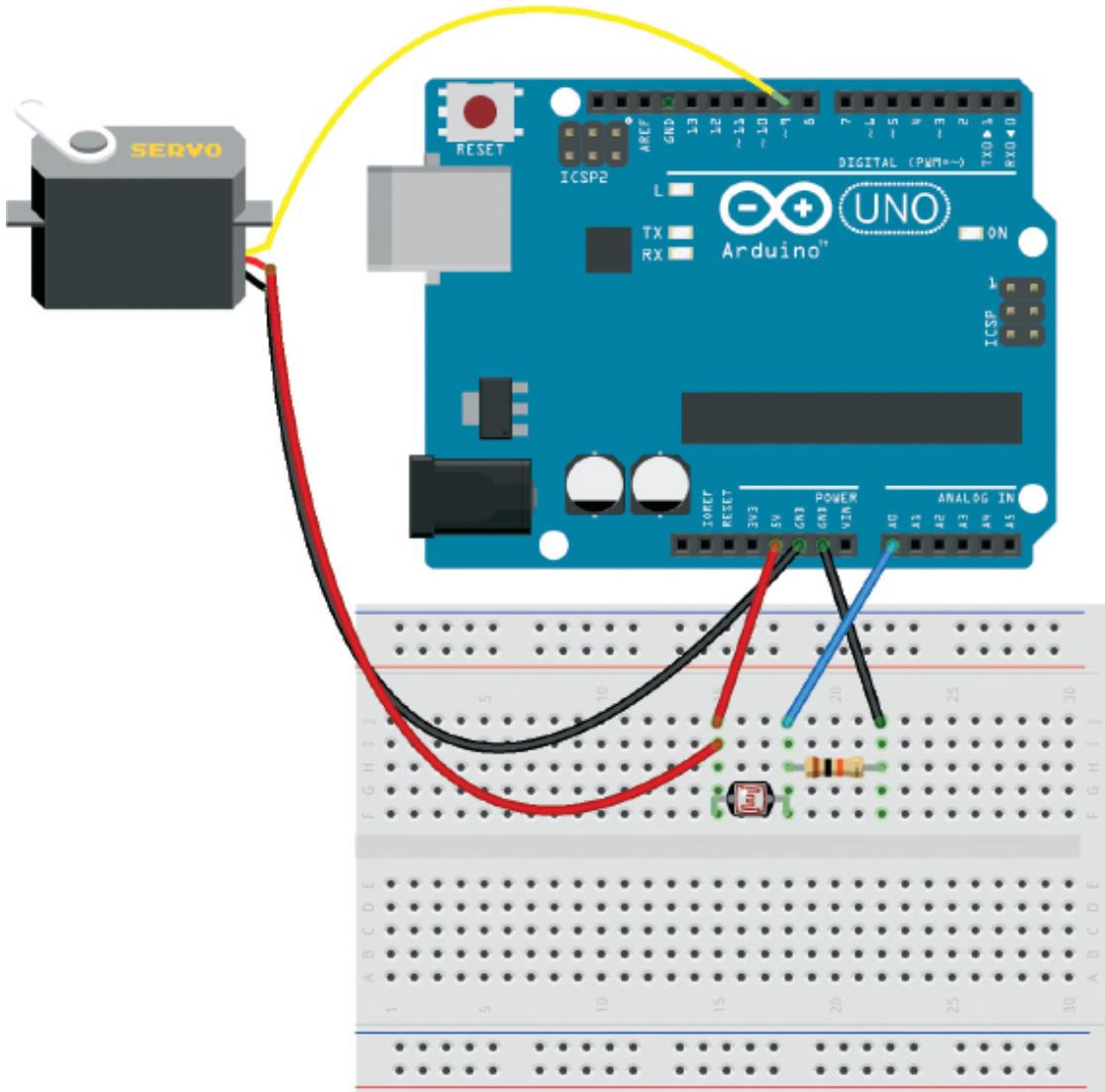
1. Connect the servo's red (power) wire directly to +5V on the Arduino, the brown (ground) wire to Arduino GND, and the yellow (signal) wire to Arduino pin 9, as shown in the following table.

SERVO	ARDUINO
Red (power) wire	+5V
Brown (ground) wire	GND
Yellow signal (control) wire	Pin 9

2. Place the photoresistor in the breadboard and connect one leg to +5V on the Arduino. Connect the photoresistor's other leg to a 10k-ohm resistor, as shown in the circuit diagram in Figure 10-2, and use a jumper wire to connect this resistor leg to Arduino pin A0 (see the following table). Connect the other leg of the 10k-ohm resistor to GND.

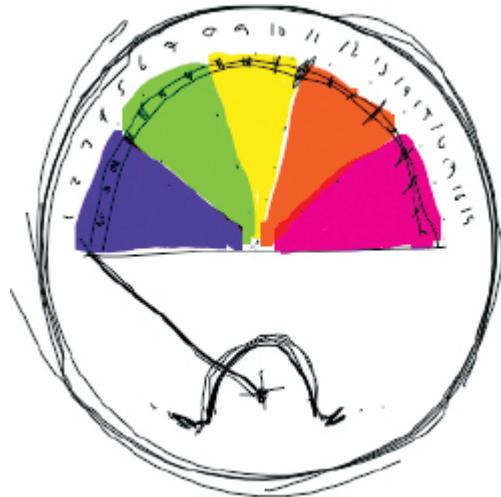
PHOTORESISTOR	ARDUINO
Leg 1	+5V
Leg 2	Pin A0 via 10k-ohm resistor

**FIGURE 10-2:** The photoresistor is connected to Arduino pin A0 and measures the amount of light. The servo is connected to pin 9 and moves according to the amount of light.



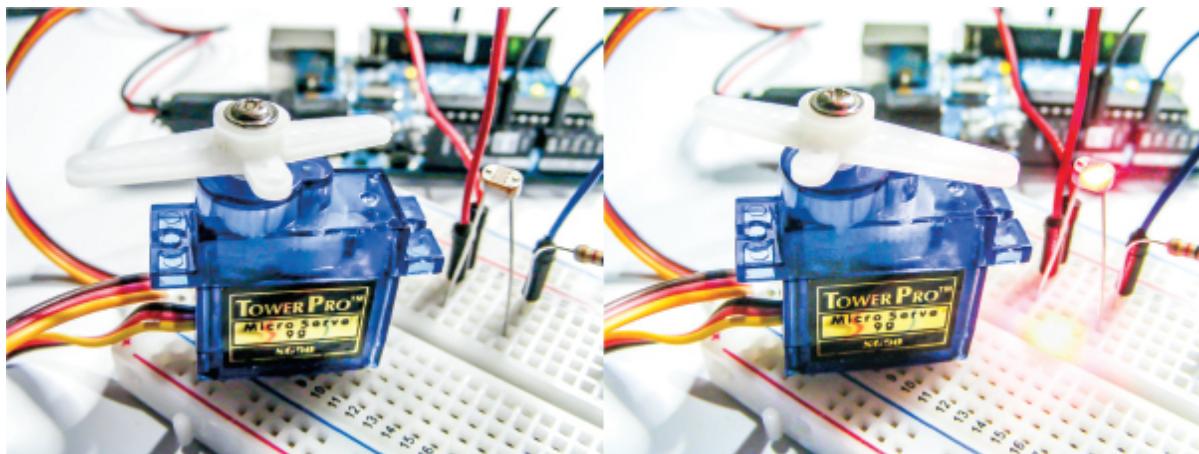
3. Upload the code in “The Sketch” on page 89.
4. Make a faceplate for your dial, like the one in Figure 10-3, and attach it to the servo. Be sure the servo arm can move over the measurements of the dial like a pointer. Cover the photoresistor completely when you add power to the Arduino, and then mark this position as 0 on the faceplate. Shine a bright flashlight at the light resistor to get the maximum value, and then mark that position on the faceplate as well. Add equally spaced marks between 0 and the max value to give you a scale.

**FIGURE 10-3:** An example faceplate



The servo's actuator arm will move up the scale as it detects light, depending on the brightness. For example, on the left of Figure 10-4, the servo arm is shown at position 0. On the right, the servo arm displays the brightness measurement when light—in this case, a laser—is applied to the photoresistor.

**FIGURE 10-4:** When light shines on the photoresistor, the servo arm moves.



## THE SKETCH

The sketch first calls the Servo library, which is already built into the Arduino IDE (so there's no need to download and install this library). We give the servo position a starting value of 0, and set the photoresistor pin as A0. We assign Arduino pin 9 to control the servo

and then read the value from the analog pin. Pin A0 is capable of reading an analog value from the photoresistor and converting it to a digital value in the range 0–1,023, so we scale this down to 0–179 (180 possible values) to fit the servo arm’s 180-degree range of movement. If no light is applied to the photoresistor, the value will be 0 and the servo position will be 0. As you add light, the servo arm will move, up to a maximum of 180 degrees. The angle depends on the brightness.

---

```
/* Created by David Cuartielles modified 30 Aug 2011 by Tom Igoe
This example code is in the public domain http://arduino.cc/en/Tutorial/AnalogInput */
#include <Servo.h> // Call the Servo library (built into the IDE)
Servo myservo;
int pos = 0; // Give the position a value
int lightPin = A0; // Pin connected to the photoresistor

void setup() {
  myservo.attach(9); // Pin connected to the servo
}

void loop() {
  // Read voltage from photoresistor, can read 1024 possible values
  int lightLevel = analogRead(lightPin);
  // Scale 1024 values to 180
  lightLevel = map(lightLevel, 0, 1023, 0, 179);
  // Scale of 0-179 (180 values)
  pos = constrain(lightLevel, 0, 179);

  myservo.write(pos); // Set the servo angle
  delay(100);
}
```

---

## TROUBLESHOOTING

**Q.** *The code compiles, but the servo does not move when light is applied to the photoresistor:*

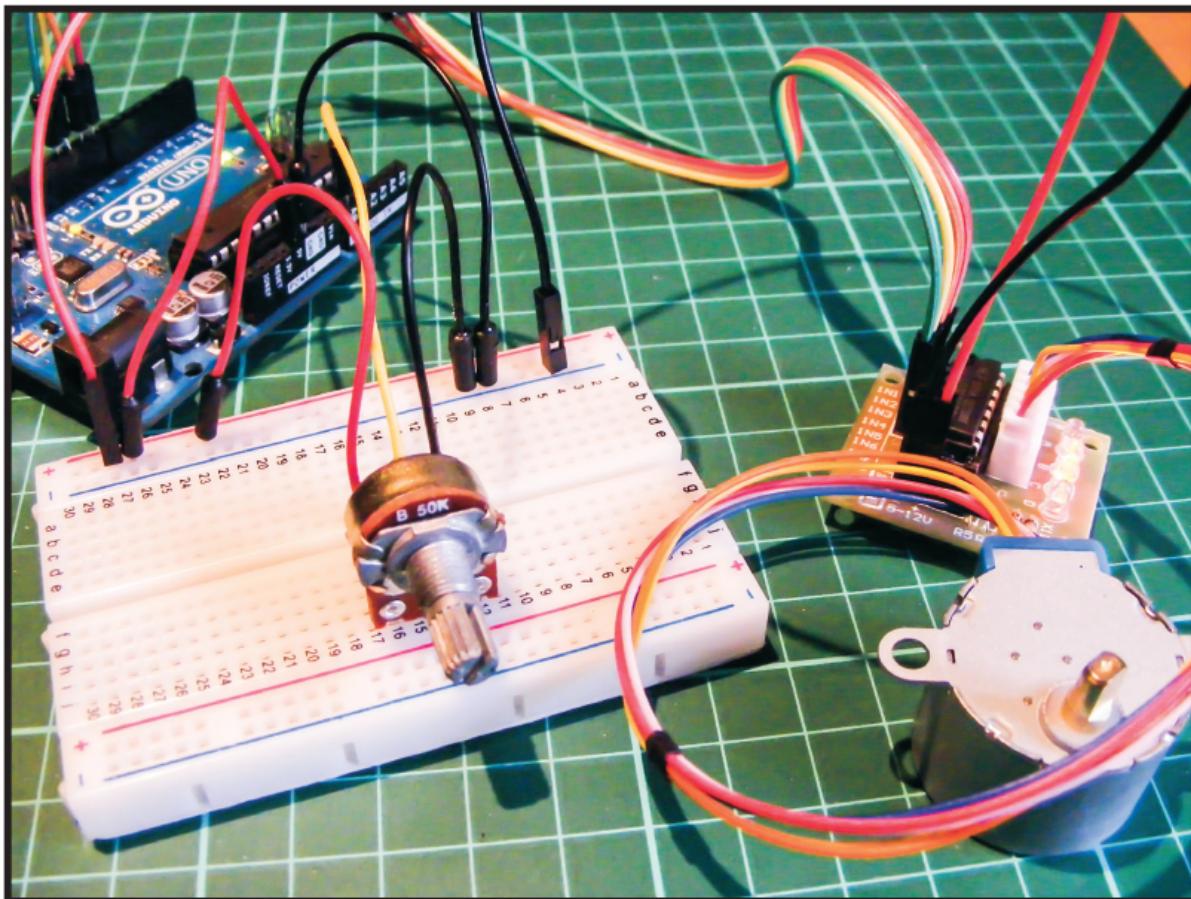
- If the servo does not move at all, make sure that your wiring matches the diagram in Figure 10-2 and that there’s power going to the Arduino.
- Connect the Arduino to your PC and open the Serial Monitor to check that there’s a reading from the photoresistor. If no reading is registered, check that the photoresistor is securely inserted into the

breadboard. If you still get no reading, your photoresistor may be faulty, so replace it with another one.

# 11

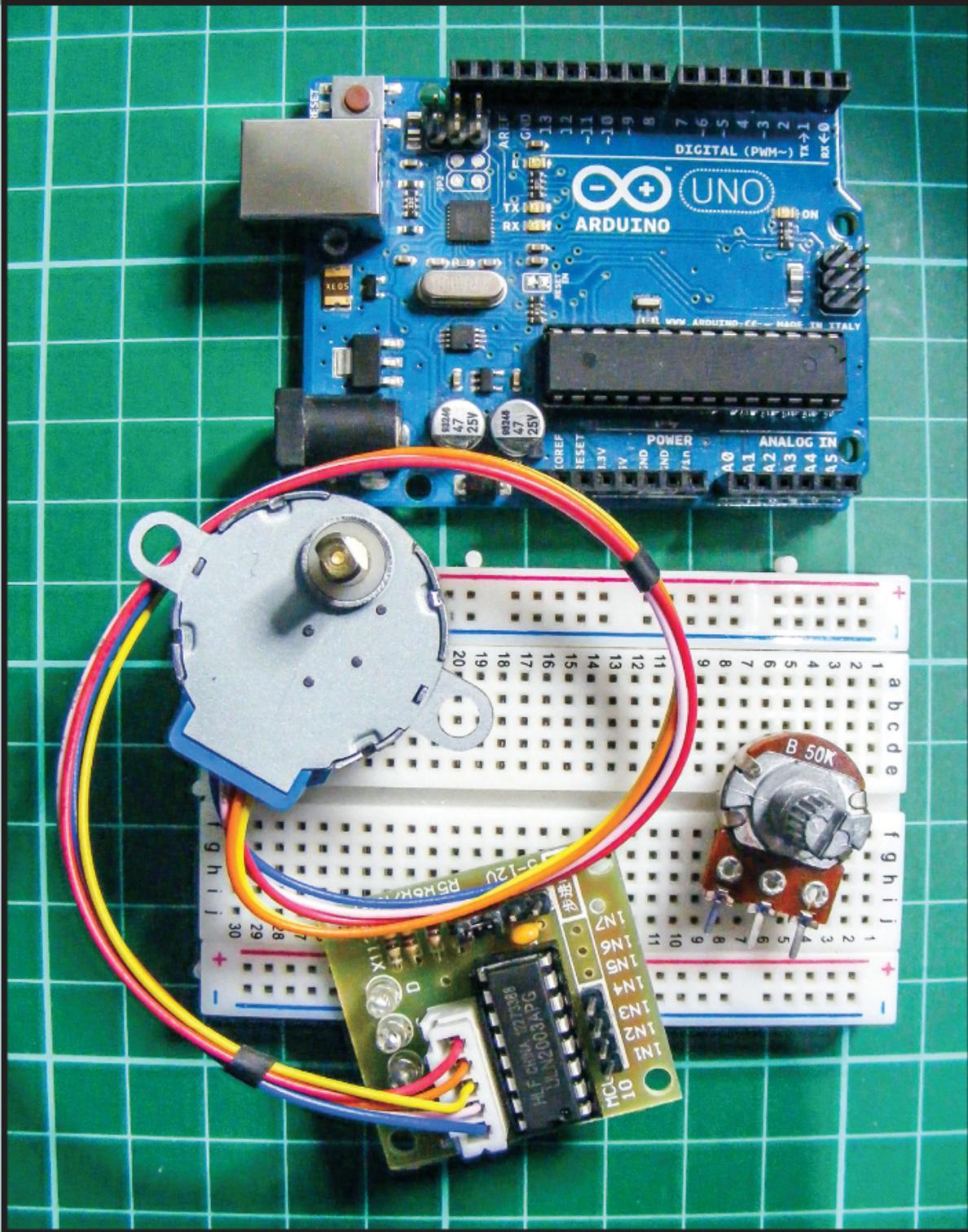
## Stepper Motor

In this project I'll introduce you to a stepper motor (or step motor), set it up, and discuss how it works.



COST: \$

TIME: 10 MINUTES



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**28BYJ-48 stepper motor with ULN2003 driver module**

**50k-ohm potentiometer**

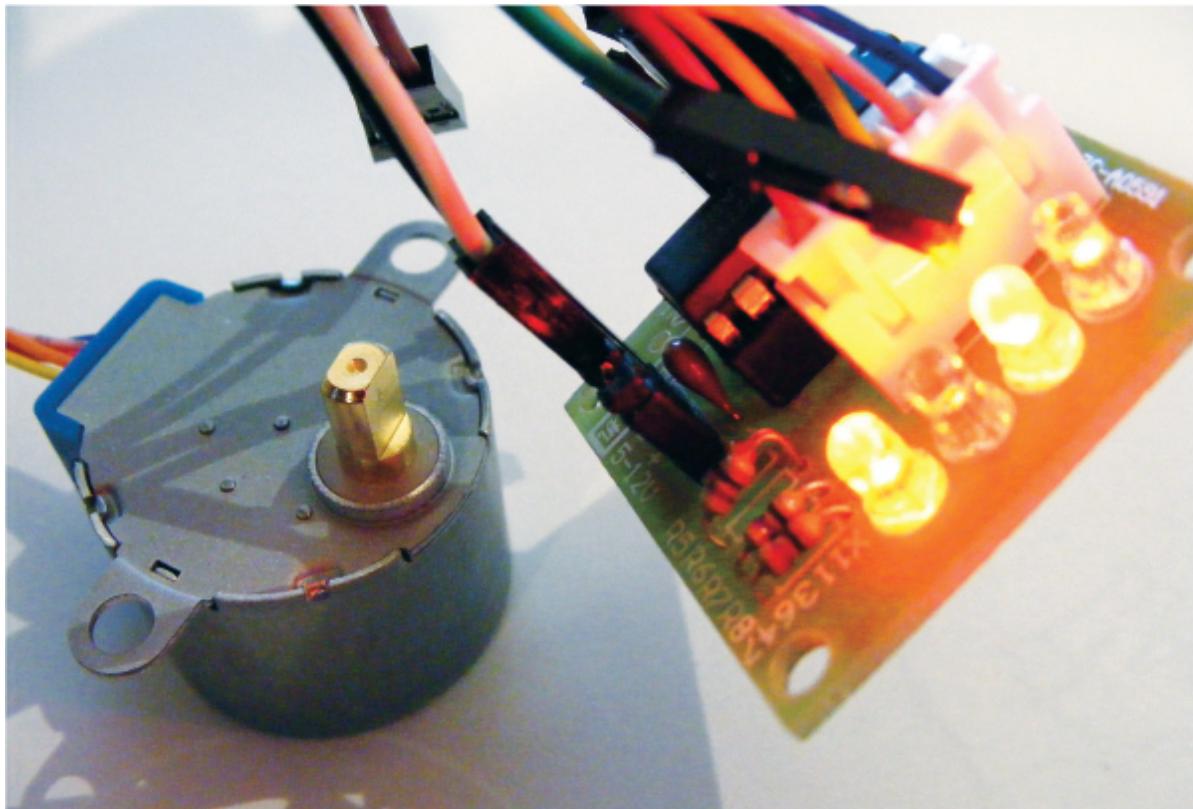
## **LIBRARY REQUIRED**

**Stepper**

## **HOW IT WORKS**

A *stepper motor*, like the one shown in Figure 11-1, is a direct current (DC) electric motor that divides a full rotation of the arm into a number of equal steps. Unlike the servomotor used in Project 10, this stepper motor turns 360 degrees and has the advantage of being able to position itself with great accuracy or rotate continuously.

**FIGURE 11-1:** A 28BYJ-48 stepper motor



The stepper motor's data sheet will state the number of steps it performs per revolution; a *step* is just one movement within one revolution. A motor with 200 steps per revolution will turn through 360 degrees in 200 steps, or 1.8 degrees per step. Within a stepper motor there are two interlocked discs, similar to gears, with teeth of opposing magnetism that alternate and connect to the center shaft or rotor. The motor moves in steps when power is sent to its *windings*—a series of wire coils that become electromagnets when voltage is applied. When powered, these electromagnets attract or oppose the gear-shaped discs and rotate the shaft.

You can control the motor's position and speed by commanding it to move to and hold at one of these steps. Since we know the angle each step represents, we can get accurate and precise turning angles and distance measurements. Stepper motors are commonly used in CD and DVD players and in 3D printers, where movements need to be very accurate.

When you’re looking to buy a stepper motor, there are a few things to consider. The first is whether or not it has a gearbox. A gearbox will improve the *torque* (moving power) but reduce the *revolutions per minute* (RPM, or speed).

The next consideration is whether the stepper motor is bipolar or unipolar. *Bipolar* motors switch polarity of the coils. Polarity is the direction the current flows; so if, for example, we reversed the 5V and GND connections, the motor would turn in the opposite direction. Bipolar motors have simpler windings but require more complicated drivers as they reverse the polarity for us. *Unipolar* motors essentially have a winding per polarity, but they can use simpler drivers. You can check whether your motor is bipolar or unipolar by looking at the connections: a bipolar motor has four connections, and a unipolar motor has five to eight connections. In this project we’re using a unipolar motor, the 28BYJ-48 stepper motor with the ULN2003 driver test module—a board that makes it easy to control the motor with the Arduino, like the module board for the LED matrix in Project 4. Some driver boards will have a slightly different setup, so I’d recommend getting the model of motor listed here for the project so you can follow the instructions closely.

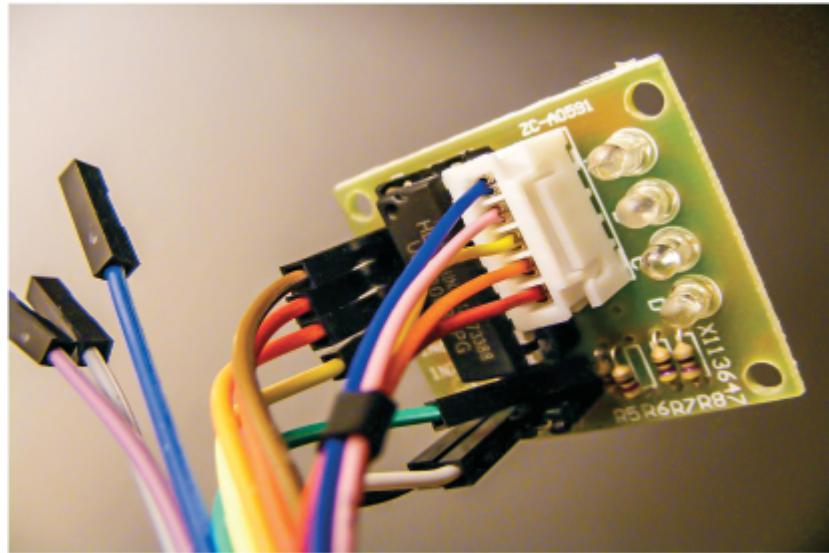
Turning the potentiometer alters the angle of the stepper motor arm, so as you move the potentiometer to the left or right, the stepper motor arm will follow your input. (A *potentiometer* is a variable resistor with a knob.) The resistance of the potentiometer changes as you turn the knob. They are commonly used in electrical devices such as volume controls on audio equipment.

## THE BUILD

1. Connect the stepper motor to the driver board, as shown in Figure 11-2. From the outermost pin to the innermost pin in the middle of the board, connect the wires from the motor in the following

order: blue, pink, yellow, orange, red. The connector can only be inserted in this way.

**FIGURE 11-2:** Connecting the stepper to the driver board



2. Connect the driver board pins 1, 2, 3, and 4 at the other end of the board directly to Arduino pins 8, 9, 10, and 11, respectively.

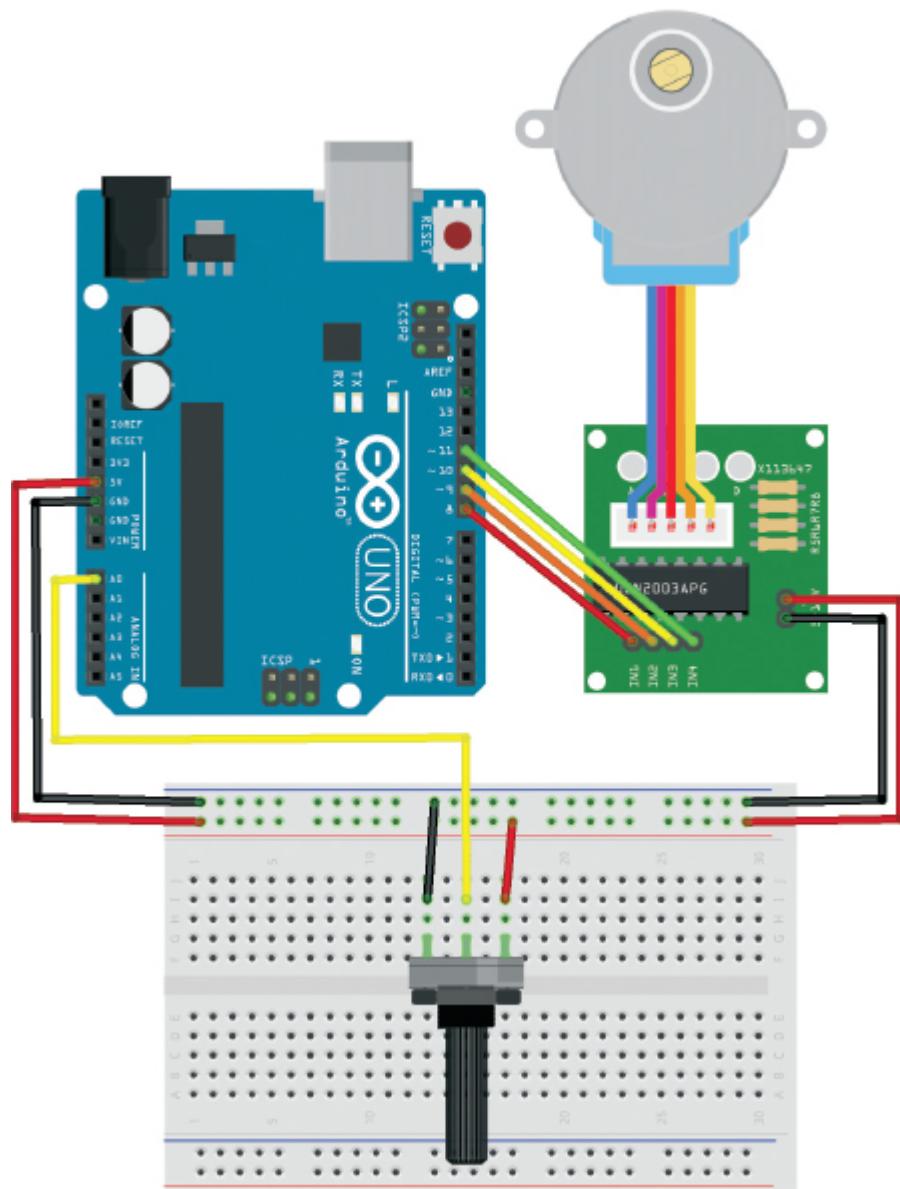
STEPPER DRIVER BOARD	ARDUINO
IN1	Pin 8
IN2	Pin 9
IN3	Pin 10
IN4	Pin 11
GND	GND
+5V	+5V

3. Insert a potentiometer into the breadboard, connecting its center pin to Arduino A0 and its outer two pins to Arduino +5V and GND in any order.

POTENTIOMETER	ARDUINO
Left pin	GND
Center pin	A0
Right pin	+5V

4. Connect the driver board GND and +5V to the breadboard GND and +5V, and connect the breadboard rails to the Arduino. Don't forget to attach the power rails of the breadboard to GND and +5V too.
5. Make sure that your setup matches the final configuration shown in Figure 11-3, and upload the code in "The Sketch" below.

**FIGURE 11-3:** The circuit diagram for the stepper motor



# THE SKETCH

This code comes with the Arduino IDE and can be found at File ▶ Examples ▶ Stepper ▶ MotorKnob. I've reproduced it here as you'll see it in the IDE. It uses the built-in stepper library, `<Stepper.h>`. The potentiometer is connected to the Arduino A0 pin and gives a variable voltage depending on the turn of the potentiometer, which then controls the position of the stepper motor.

```
-----  
/* MotorKnob  
 * http://www.arduino.cc/en/Reference/Stepper  
 * This example code is in the public domain.  
 */  
#include <Stepper.h>  
// Change this to the number of steps on your motor  
#define STEPS 100  
  
// Create an instance of the stepper class, specifying the number of  
// steps of the motor and the pins it's attached to  
Stepper stepper(STEPS, 8, 10, 9, 11);  
// The previous reading from the analog input  
int previous = 0;  
  
void setup() {  
    // Set the speed of the motor to 700 RPM  
    stepper.setSpeed(30);  
}  
  
void loop() {  
    // Get the sensor value  
    int val = analogRead(0);  
  
    // Move a number of steps equal to change in the sensor reading  
    stepper.step(val - previous);  
  
    // Remember the previous value of the sensor  
    previous = val;  
}
```

-----

## TROUBLESHOOTING

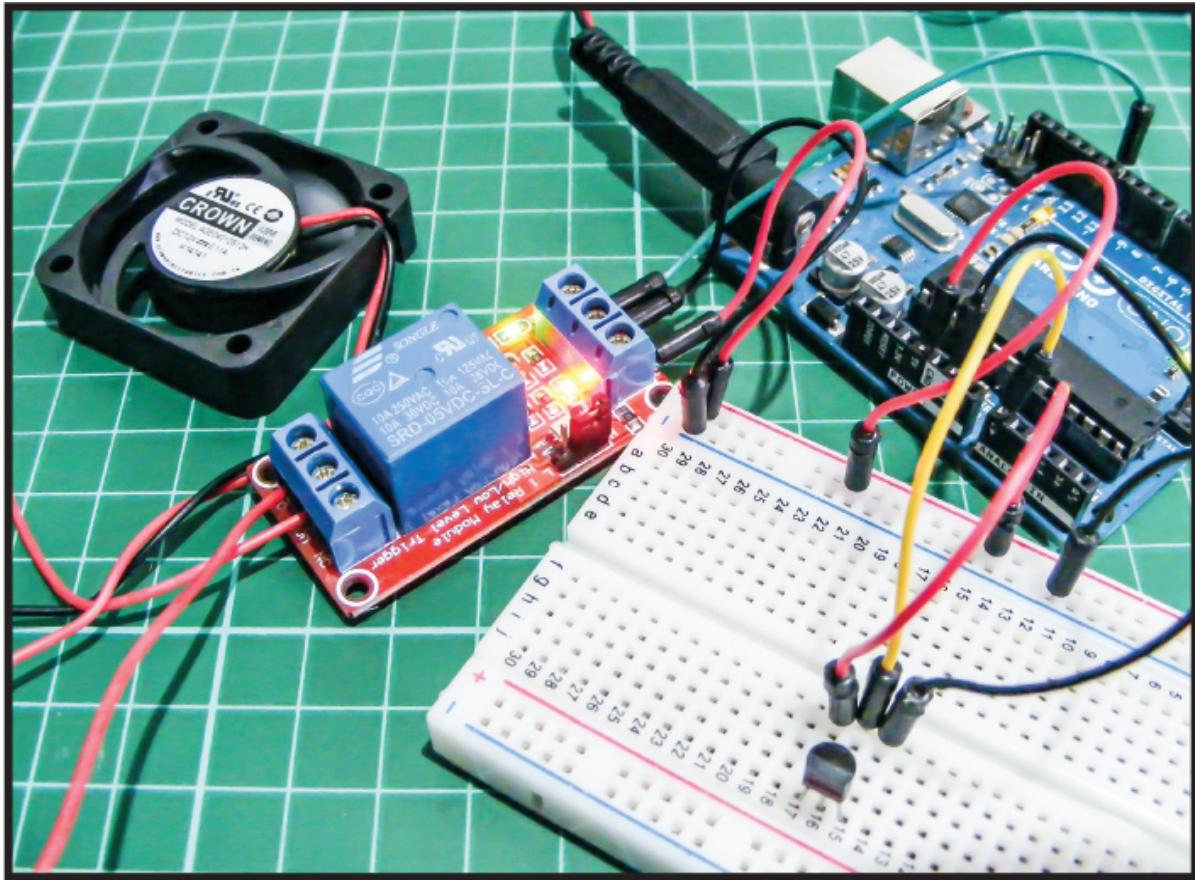
**Q.** *The code compiles, but the stepper motor does not move.*

- When you power the motor, lights should blink on the driver motor board. If they don't, there's an issue with power, so check that your setup matches the circuit diagram in Figure 11-3. Make sure the stepper motor connection is firmly inserted into the driver motor board—it can only go in one way.
- If the driver board lights but the motor does not move, check that the connections to the potentiometer are secure and match the tables shown earlier.

# 12

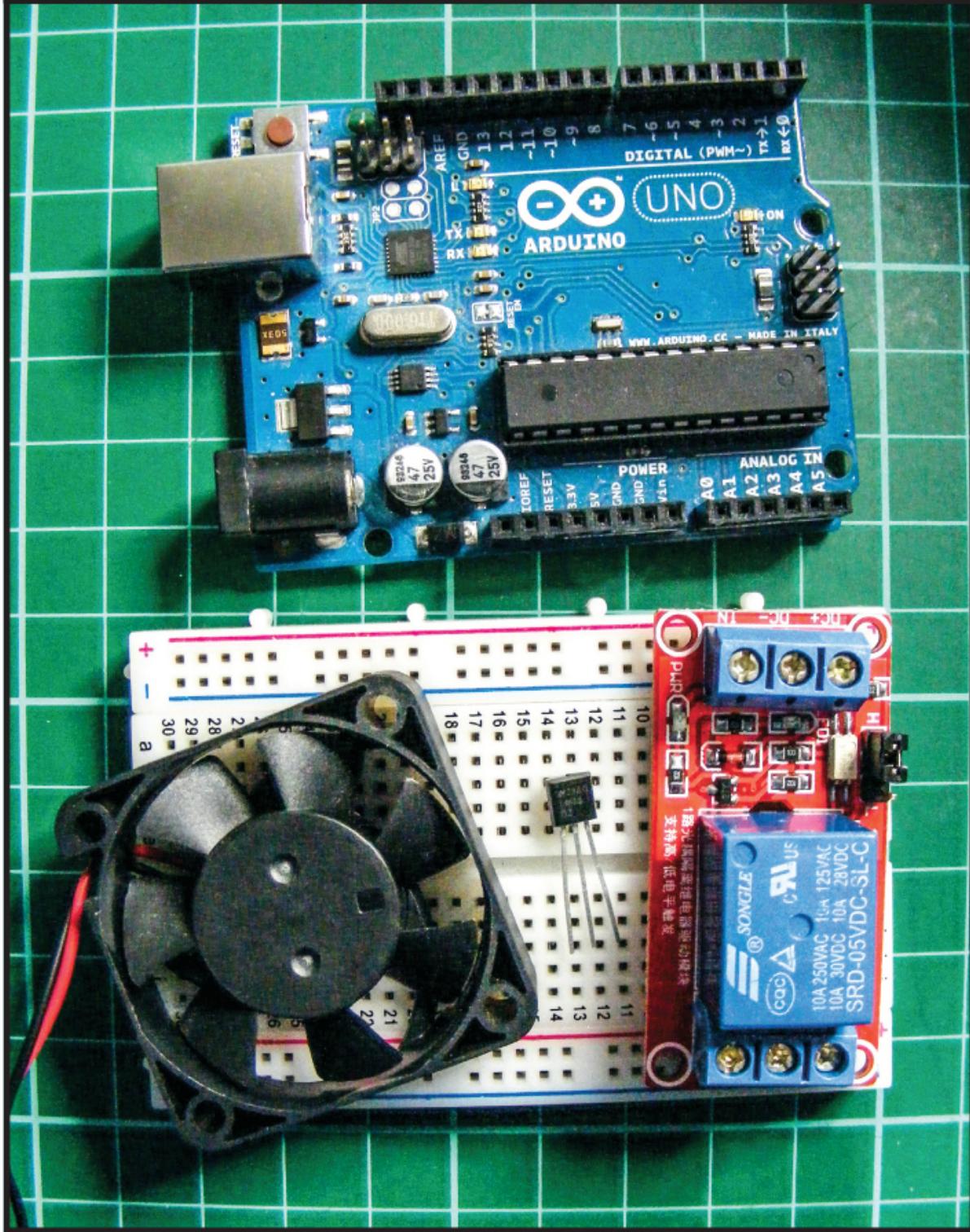
## Temperature-Controlled Fan

In this project, we'll use an LM35 temperature sensor to turn a fan on automatically when the temperature is too high.



COST: \$\$

TIME: 30 MINUTES



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**LM35 temperature sensor**

**5V single-channel relay module**

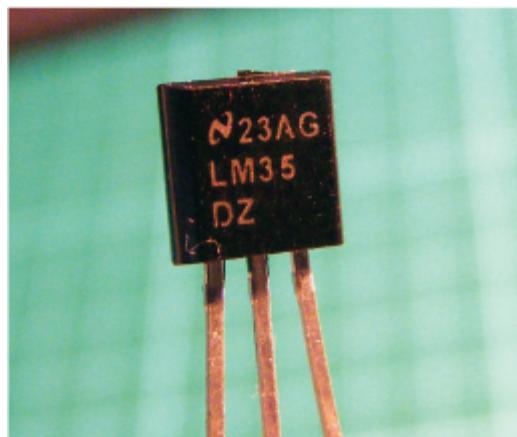
**12V mini computer cooling fan**

**9V battery snap and battery**

## HOW IT WORKS

The LM35 temperature sensor (shown in Figure 12-1) senses the temperature and sends that measurement to the Arduino in voltage. The Arduino converts this voltage value to temperature in degrees Celsius and then converts this value to degrees Fahrenheit. When the temperature reading is above 71 degrees Fahrenheit, the Arduino sends power to the relay, which turns on the computer fan.

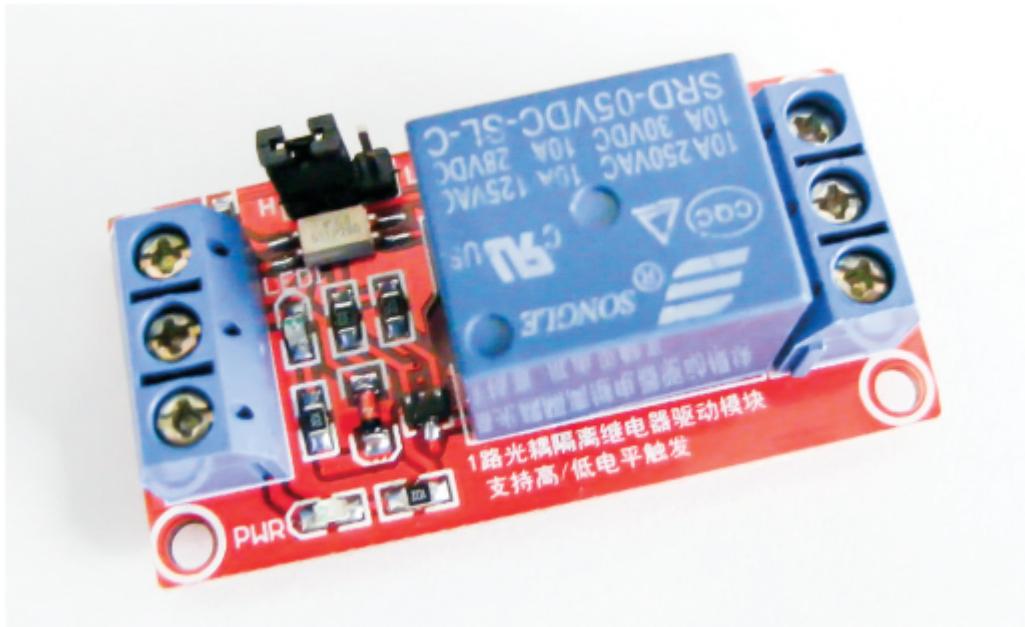
**FIGURE 12-1:** The LM35 temperature sensor: the left pin is +5V, center is data out, and right is GND.



The computer fan requires more power than the Arduino can provide, so we need to give it its own power supply: a 9V battery. This circuit is controlled by an *electronic relay*—an electronically operated switch that in this case uses an electromagnet to mechanically open or close the circuit (shown in Figure 12-2). A relay is generally used when a low-power device is required to switch on or off a much higher-voltage

device. Our relay is powered by 5 volts to operate the mechanical switch. In this project the circuit is only 9 volts, but the relay could control a circuit up to 240 volts. Adding higher-voltage circuits can be very dangerous, however, so do this only if you are comfortable working with electricity or can seek professional advice.

**FIGURE 12-2:** A 5V single-channel relay



## THE BUILD

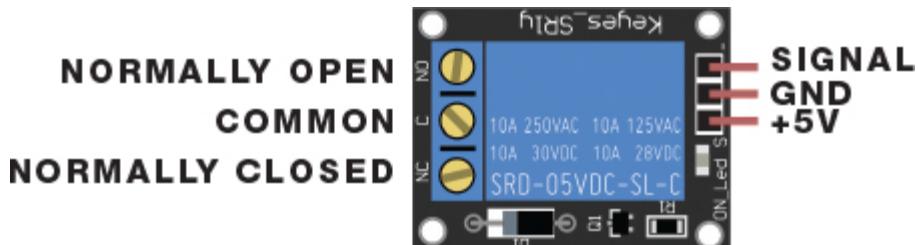
1. Insert the LM35 sensor into the breadboard with the front of the sensor (the flat surface with text on it) facing you. Connect the left pin to the +5V rail on the breadboard, the center pin to Arduino A0, and the right pin to the GND rail, as shown in the following table.

LM35 SENSOR	ARDUINO
Left pin	+5V
Center pin	A0

LM35 SENSOR	ARDUINO
Right pin	GND

2. There are a number of connections on the relay, as shown in Figure 12-3. If your relay module has a different layout, adapt the wiring accordingly (using the data sheet or the pin markings on the module). Our relay has an LED marked PWR to indicate when it's receiving power, and another LED to show when the electromagnetic switch is on (you can usually hear this, too, as it makes a satisfying clicking noise). The relay can be set to be HIGH or LOW when triggered, as indicated by a small jumper switch or pins. For our project, make sure the jumper is set to HIGH so the relay will send power when it is triggered.

**FIGURE 12-3:** Relay connections (your relay pins may differ, so follow the data sheet provided)



3. As Figure 12-3 shows, the pins on the right side of the relay module are Signal, GND, and +5V. Attach the relay's Signal pin to Arduino pin 5, GND to Arduino GND, and +5V to the Arduino power via the breadboard rails.

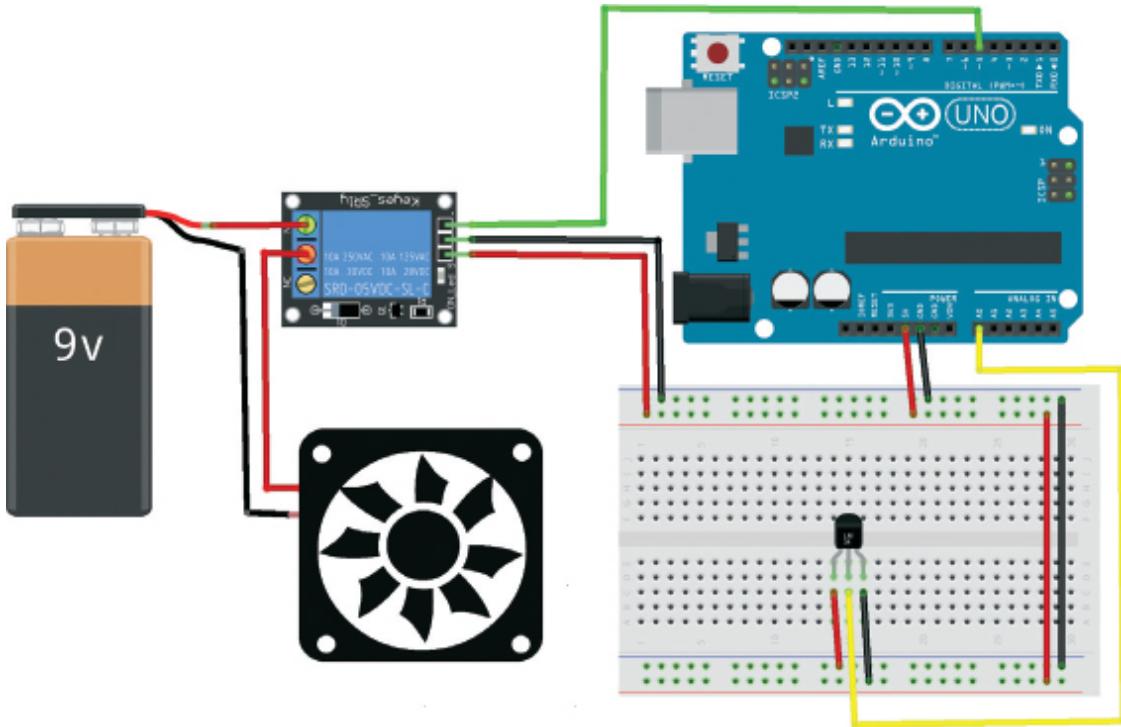
5V RELAY	ARDUINO
Signal	Pin 5
GND	GND
+5V	+5V

- On the left side of the relay module are the connections for the electromagnetic switch (Figure 12-3). The center pin is the common connection; the left pin is marked NO for *normally open*, meaning the circuit is broken and the default state is off; and the right pin is marked NC for *normally closed*, meaning the default state is on. If the relay is not switched, the common pin is connected to the NC pin. If the relay is switched, the common pin is connected to the NO pin. Because we want the circuit to be off until we use the switch, we will use the NO pin.
- Next, connect the black GND wire of the fan to the GND wire of the 9V battery. Then, as shown in the following table, attach the red positive wire of the fan to the common pin on the relay, and connect the positive wire of the 9V battery to NO on the relay.

<b>5V RELAY</b>	<b>FAN/9V BATTERY</b>
NO (normally open)	9V battery's positive wire
Common	Fan's positive wire
NC (normally closed)	Not connected

- Connect the breadboard power rails to each other and to the Arduino GND and +5V pins.
- Make sure your setup matches the circuit diagram in Figure 12-4, and then upload the code in “The Sketch” on page 103.

**FIGURE 12-4:** The circuit diagram for the temperature-controlled fan



## THE SKETCH

In this sketch we first set the sensor pin for the LM35 as A0 on the Arduino, define the fan as pin 5, and create a variable to read the value from the LM35. We then create a variable to store the temperature and set the fan pin as an output. A small calculation turns the voltage reading from the sensor into a temperature value in degrees Fahrenheit. We then start the Serial Monitor so you can see the LM35 reading value when the Arduino is connected to your PC, which is handy for making sure the sensor is working correctly. A loop reads the sensor every second, and if the temperature reaches 71 degrees Fahrenheit, power is sent to the fan pin, which triggers the relay and switches on the fan. If the temperature falls below 71, the relay switches the fan off.

```

#define SENS_PIN A0 // Defines A0 pin as "sensor"
#define FAN_PIN 5
int Vin; // Reads value from Arduino pin
float Temperature; // Receives converted voltage value to temp
float TF; // Receives converted value in °F

void setup() {

```

```

pinMode(FAN_PIN, OUTPUT); // Fan pin as an output
Serial.begin(9600); // Start Serial Monitor
}

void loop() {
    // Tells Arduino to read pin and stores value in Vin
    Vin = analogRead(SENS_PIN);

    // Converts voltage value into temperature and
    // stores value in Temperature (as °F)
    Temperature = (500 * Vin) / 1023 * (1.8) + 32;

    TF = Temperature;
    Serial.print("Temperature: "); // Sends text to display screen
    Serial.print(TF); // Shows value of temperature in Serial Monitor
    Serial.println(" F"); // Writes F to indicate it is in Fahrenheit
    if (TF > 71) { // If temperature is more than 71
        digitalWrite(FAN_PIN, HIGH); // Turn fan on
    }
    else if (TF < 71) {
        digitalWrite(FAN_PIN, LOW); // Or keep fan off
    }
    delay(1000); // Waits for a second to read the pin again
}

```

---

## TROUBLESHOOTING

**Q.** *The fan does not turn on when expected.*

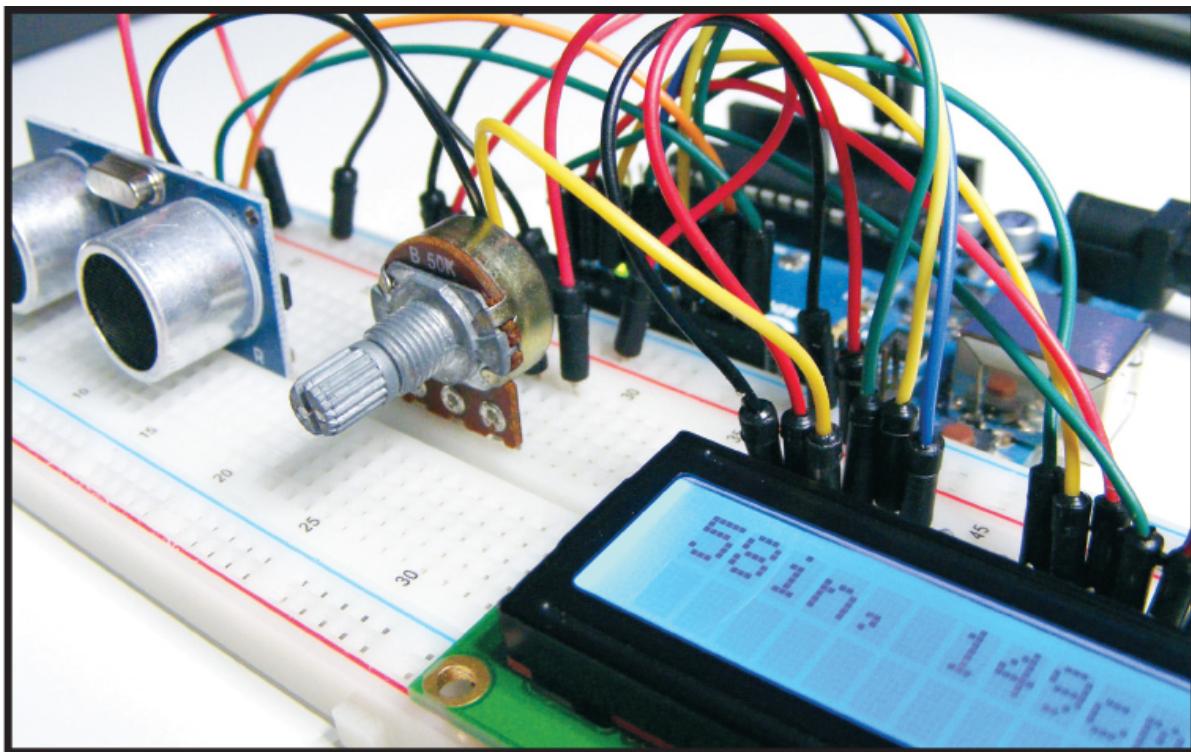
- Make sure the connections to the LM35 match the tables in this chapter and the circuit diagram in Figure 12-4. Connect the Arduino to your computer and open the IDE Serial Monitor to check whether the Arduino is reading the sensor correctly. If the reading is incorrect, recheck your wiring or change the sensor to another.
- Remember, your relay may not match the one used here, so the connections may be in a slightly different order; alter the wiring according to your relay and data sheet.
- The fan used here takes between 9 and 12 volts, so a 9V battery has enough power to run it. If you used a fan that requires more voltage, you will need to match its voltage input accordingly with a more powerful battery.

# LCDs

# 13

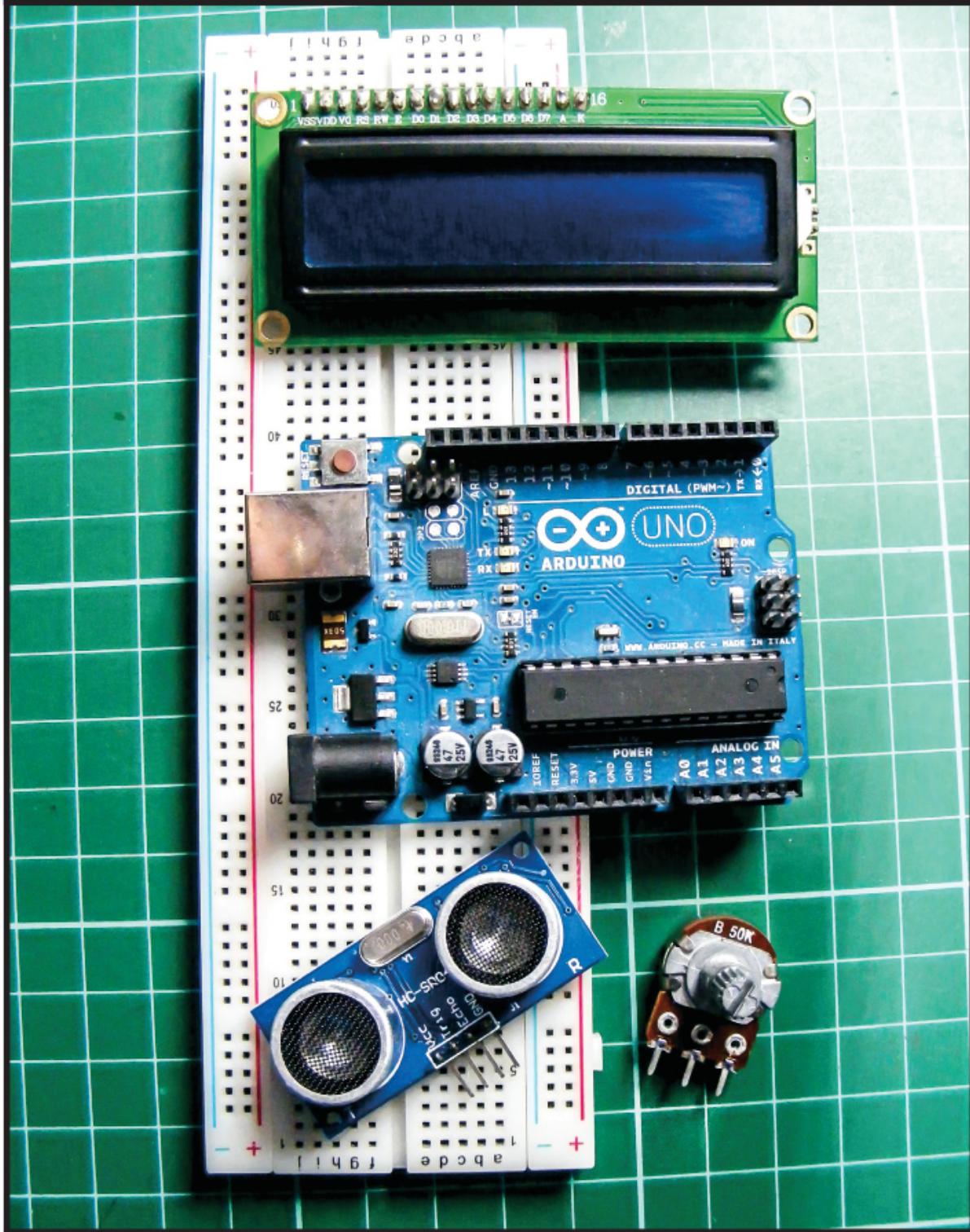
## Ultrasonic Range Finder

In this project we'll create a simple ultrasonic range finder with a screen that displays the distance of an object up to 5 meters from the sensor.



**COST: \$\$**

**TIME: 30 MINUTES**



## **PARTS REQUIRED**

**Arduino board**

**Breadboard**

**Jumper wires**

**HD44780 16x2 LCD screen**

**HC-SR04 ultrasonic sensor**

**50k-ohm potentiometer**

## **LIBRARY REQUIRED**

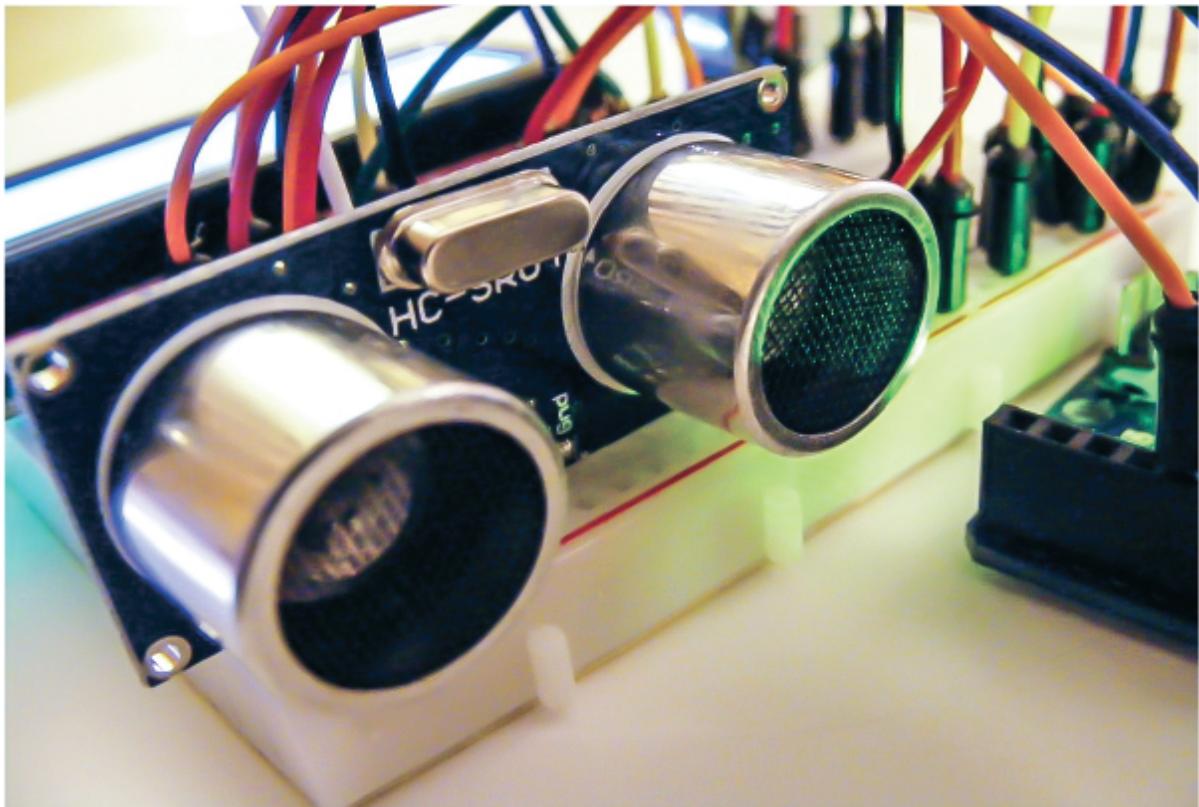
**LiquidCrystal**

## **HOW IT WORKS**

The ultrasonic range finder sends out a burst of ultrasound and listens for the echo that bounces off an object. The Arduino sends out a short pulse on the trigger pin to send the ultrasonic burst, then listens for a pulse on the echo pin using the `pulseIn` function.

This duration between sending and receiving the pulse is equal to the time taken by the ultrasound to travel to the object and back to the sensor. The Arduino converts this time to distance and displays it on the LCD screen. You can find an HC-SR04 unit (Figure 13-1) from one of the sources listed in the “Retailer List” on page 249, or you can search online for *HC-SR04 ultrasonic module*.

**FIGURE 13-1:** The HC-SR04 ultrasonic sensor



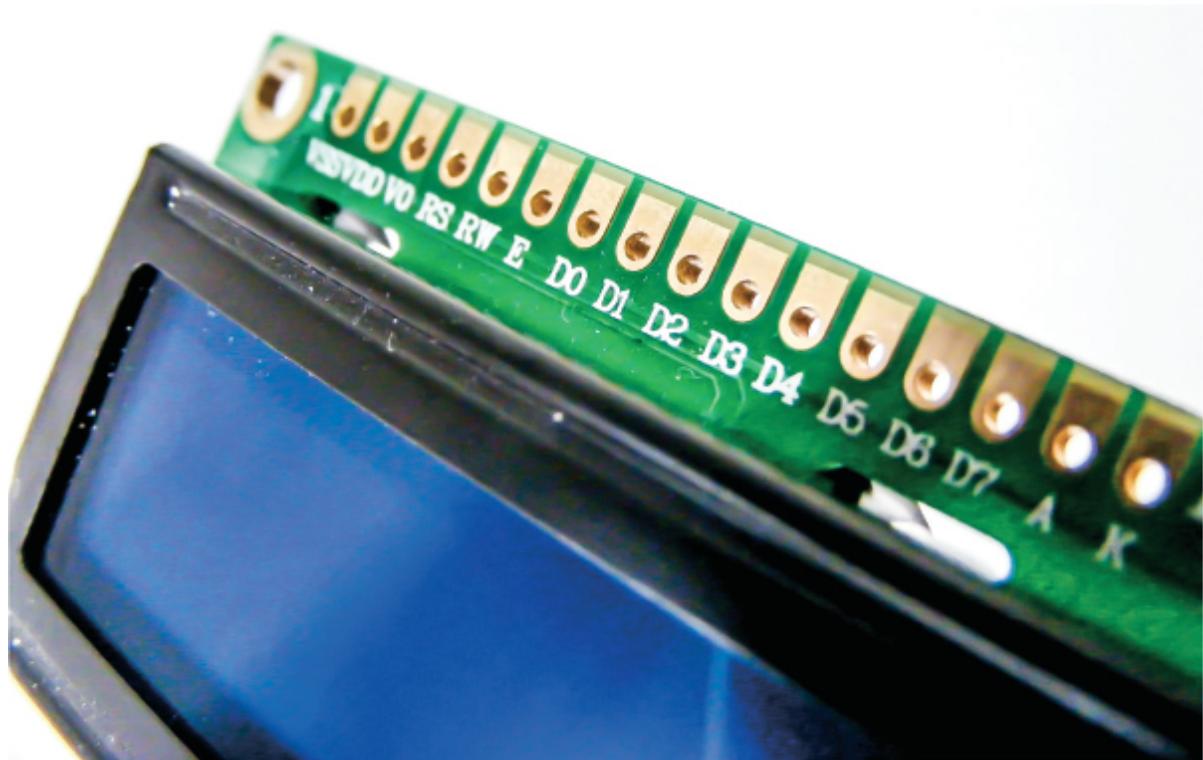
An LCD (liquid crystal display) screen is made of two sheets of polarizing material with a liquid crystal solution between them. Current passing through the solution makes the screen opaque, so by controlling which areas of the screen current passes through, the Arduino creates an image or, in this case, characters. You'll need an LCD screen that's compatible with the Hitachi HD44780 driver for it to work with the Arduino; there are lots of them out there and you can usually identify them by their 16-pin interface. We'll use the LiquidCrystal library to send characters to the LCD screen (refer to the primer if you need a refresher on libraries). The LiquidCrystal library maps the characters and uses the `print` commands to send messages to the screen.

## PREPARING THE LCD SCREEN

The LCD screen will probably require a bit of assembly. Your screen should have come with 16 holes, as shown in Figure 13-2, and a separate strip of header pins. Break off a row of 16 pins from the strip. Insert the

shorter side of the pins into the 16 LCD holes. You'll need to solder these in place; the primer has a quick soldering guide if you need pointers. Solder the far-right and far-left pins first to hold the strip in place and wait a moment for them to set. Then solder each pin in turn. Holding the iron to the pins for too long will damage them, so solder them only for a couple of seconds.

**FIGURE 13-2:** A 16×2 LCD screen



## THE BUILD

1. Place your LCD screen in the breadboard, inserting the header pins into the breadboard holes. Also place the potentiometer in the breadboard, and use jumper wires to connect your LCD screen, Arduino, and potentiometer as shown in the following table. The pins of the LCD screen should be labeled or numbered, either on the back or the front. If not, they usually start at 1 from the left when the pins are along the top. There are a number of

connections from the LCD screen to Arduino GND, so use the breadboard ground rail to make multiple connections to the Arduino GND pin.

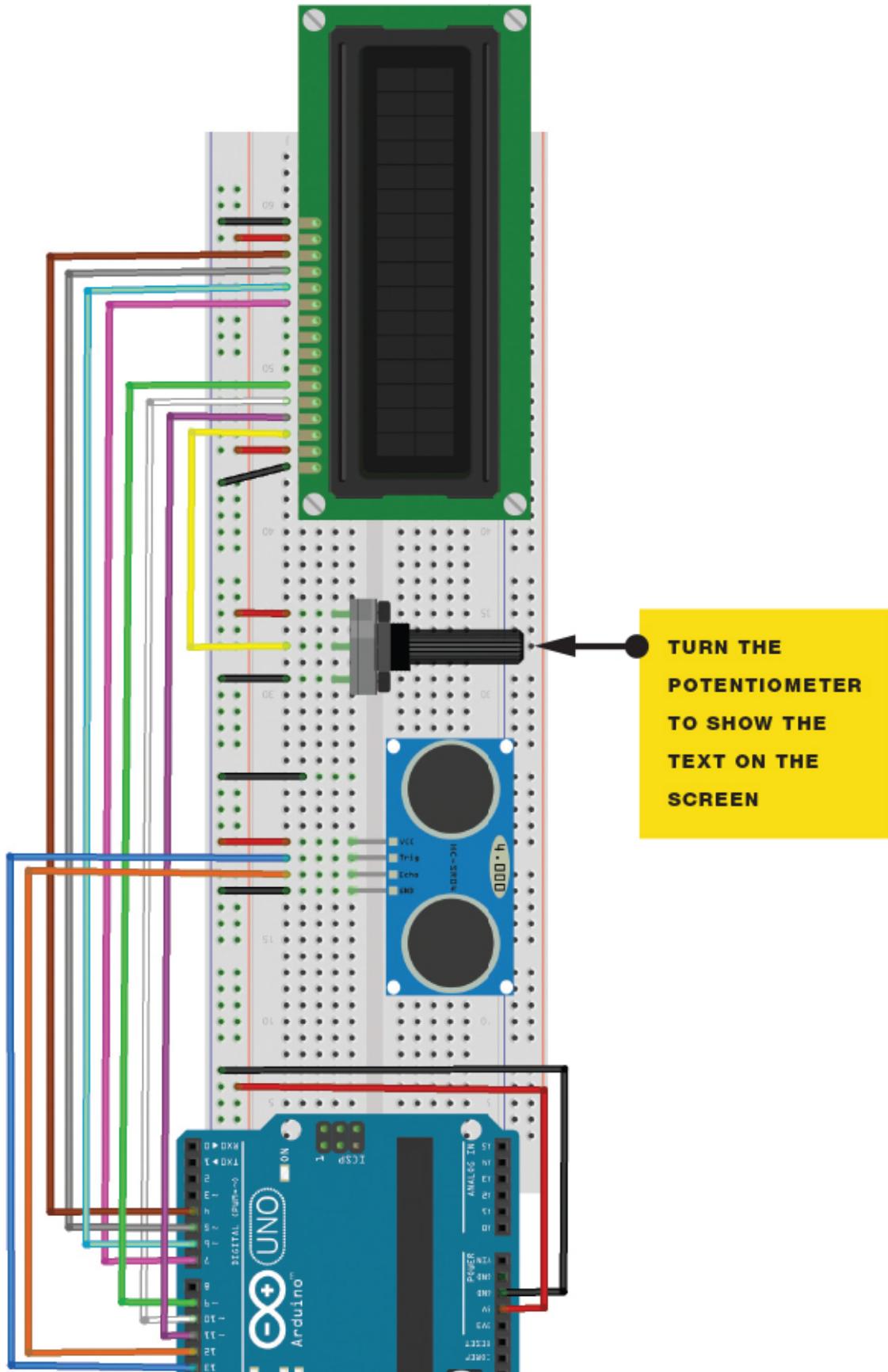
LCD SCREEN	ARDUINO
1 VSS	GND
2 VDD	+5V
3 VO contrast	Potentiometer center pin
4 RS	Pin 11
5 R/W	Pin 10
6 Enable	Pin 9
7 D0	No connection
8 D1	No connection
9 D2	No connection
10 D3	No connection
11 D4	Pin 7
12 D5	Pin 6
13 D6	Pin 5
14 D7	Pin 4
15 A BcL+	+5V
16 K BcL-	GND

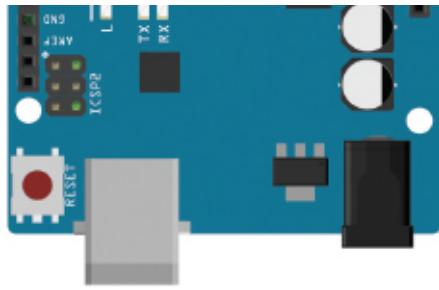
2. You should have already connected the center pin of the 50kohm potentiometer to LCD pin 3 (VO). Now connect one of the outer potentiometer pins to GND and the other to +5V. Twist the potentiometer to control the contrast of your LCD screen.
3. Backlit LCD screens will have resistors built in, but if you have a nonbacklit LCD screen, insert a 220-ohm resistor between LCD 15 and +5V. Check the data sheet for your screen if you're unsure.
4. Add the ultrasonic sensor module to your breadboard and connect VCC to +5V, Trig to Arduino pin 13, Echo to Arduino pin 12, and GND to GND, as shown in the following table.

ULTRASONIC SENSOR	ARDUINO
VCC	+5V
Trig	Pin 13
Echo	Pin 12
GND	GND

5. Connect your breadboard rails to Arduino +5V and GND for power.
6. Check that your setup matches the circuit diagram in Figure 13-3, and upload the code in “The Sketch” on page 112.

**FIGURE 13-3:** The circuit diagram for the ultrasonic range finder





## THE SKETCH

The sketch first calls on the LiquidCrystal library and defines the LCD pins connected to the Arduino. Pin 13 on the Arduino, connected to the trigger pin of the sensor, sends an ultrasonic signal out, and Arduino pin 12, connected to the echo pin of the sensor, receives the returning signal. The Arduino converts the time between sending and receiving the signal into distance and displays the result on the LCD screen, in both inches and centimeters. This sketch can be found on the Arduino site, so I've copied it here exactly as it appears there.

```
/*
 * Created 3 Nov 2008 by David A. Mellis;
 * Modified 30 Aug 2011 by Tom Igoe
 * This example code is in the public domain.
 */
#include <LiquidCrystal.h>

LiquidCrystal lcd(11, 10, 9, 7, 6, 5, 4);
int pingPin = 13;
int inPin = 12;

void setup() {
  lcd.begin(16, 2);
  lcd.print("testing...");
}

void loop() {
  // Establish variables for duration of the ping,
  // and the distance result in inches and centimeters:
  // long duration, inches, cm;

  // The PING)) is triggered by a HIGH pulse of 2 ms or more
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
```

```

delayMicroseconds(2);
digitalWrite(pingPin, HIGH);
delayMicroseconds(10);
digitalWrite(pingPin, LOW);

// The same pin is used to read the signal from the PING)):
// a HIGH pulse whose duration is the time (in microseconds)
// from the sending of the ping to the reception of its echo off
// of an object.
pinMode(inPin, INPUT);
duration = pulseIn(inPin, HIGH);

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

lcd.clear();
lcd.setCursor(0, 0);
lcd.print(inches);
lcd.print("in, ");
lcd.print(cm);
lcd.print("cm");

delay(100);
}

long microsecondsToInches(long microseconds) {
    // According to Parallax's datasheet for the PING)),
    // there are 73.746 ms/in (i.e. sound travels at 1130 fps).
    // This gives the distance traveled by the ping, outbound,
    // and return, so divide by 2 to get the distance of the obstacle.
    return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
    // The speed of sound is 340 m/s or 29 ms/cm.
    // The ping travels out and back, so to find the distance
    // of the object, take half of the distance traveled.
    return microseconds / 29 / 2;
}
-----
```

## TROUBLESHOOTING

**Q.** *Nothing is displayed on the LCD screen.*

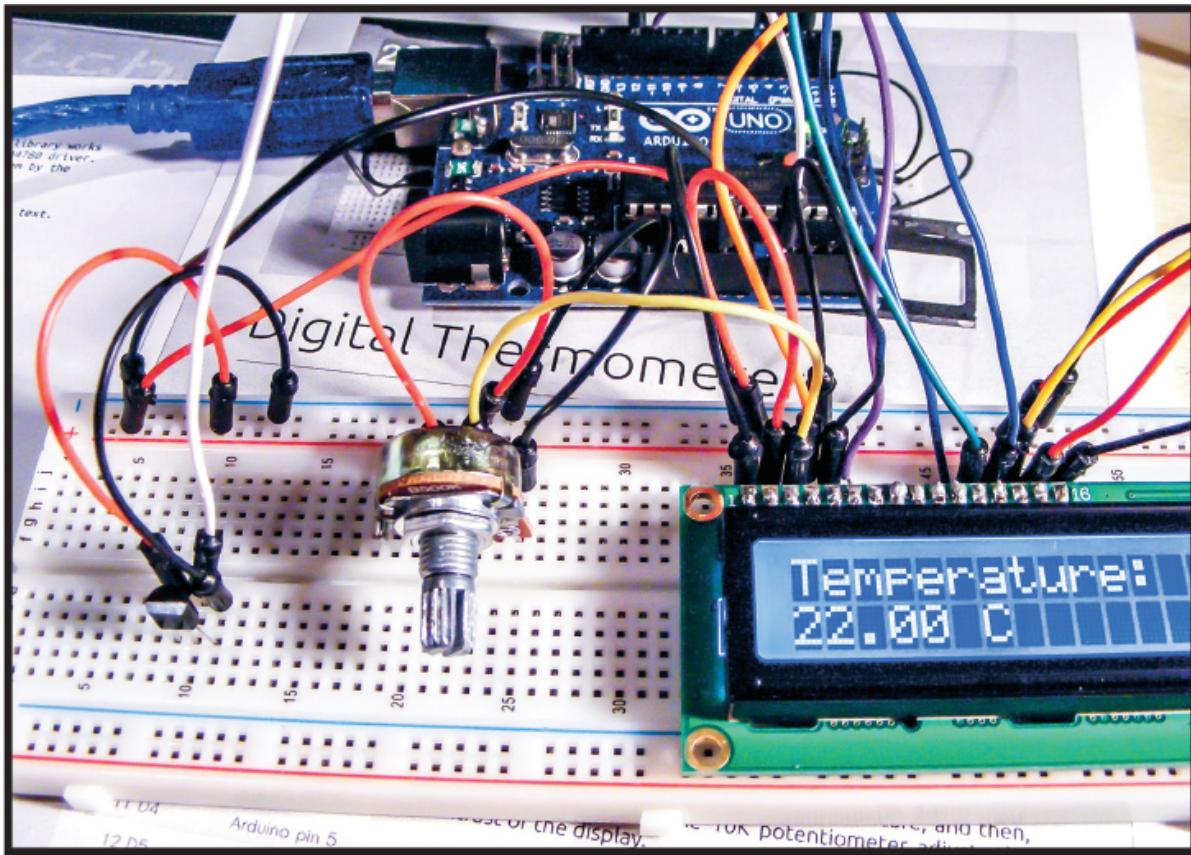
- Make sure you've connected power to the breadboard rails and the connections match the tables given earlier.

- Turn the potentiometer to change the contrast of the screen until you see text.
- If the screen has garbled messages on it, you have not wired it up correctly; recheck your wiring against Figure 13-3.

# 14

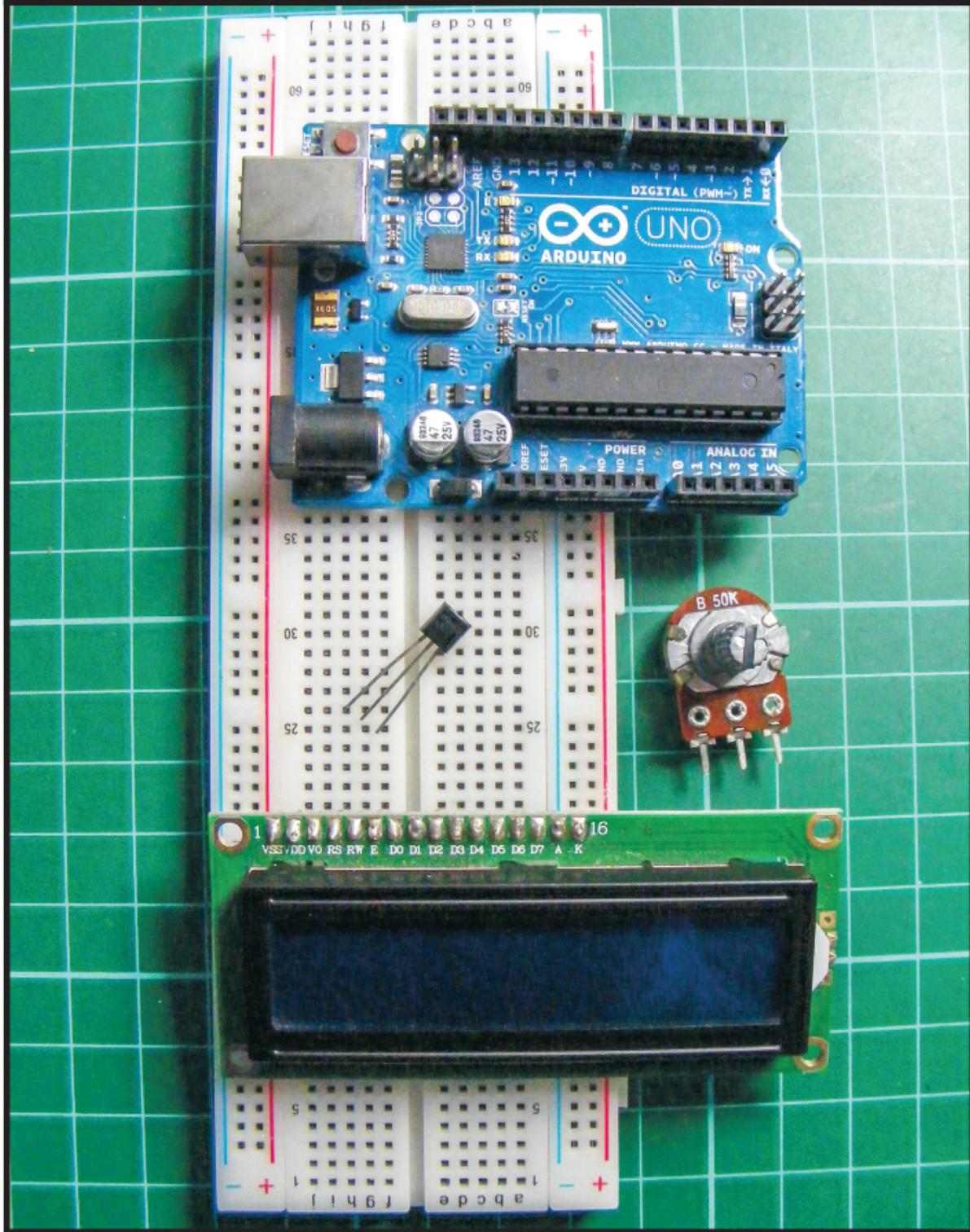
## Digital Thermometer

This project will add an LM35 temperature sensor to an LCD screen and Arduino to give you a digital thermometer.



**COST: \$\$**

**TIME: 20 MINUTES**



## **PARTS REQUIRED**

**Arduino board**

**Breadboard**

**Jumper wires**

**HD44780 16x2 LCD screen**

**LM35 temperature sensor**

**50k-ohm potentiometer**

## **LIBRARY REQUIRED**

**LiquidCrystal**

## **HOW IT WORKS**

The Arduino takes the voltage reading from the same LM35 temperature sensor we used in Project 12 and converts that value to temperature in degrees Celsius. The sketch then changes this value to Fahrenheit by multiplying the value by 9, dividing the result by 5, and adding 32. The LiquidCrystal library does all the hard work in displaying the temperature on the LCD screen using the `lcd.print` command. This project can easily be adapted with more sensors for an all-around weather center.

## **THE BUILD**

First, prepare the LCD screen according to “Preparing the LCD Screen” on page 109. Then follow these steps:

1. Insert your LCD screen and potentiometer into the breadboard; then use your breadboard and jumper wires to make the connections for the LCD screen as shown in the following table.

<b>LCD SCREEN</b>	<b>ARDUINO</b>
1 VSS	GND
2 VDD	+5V

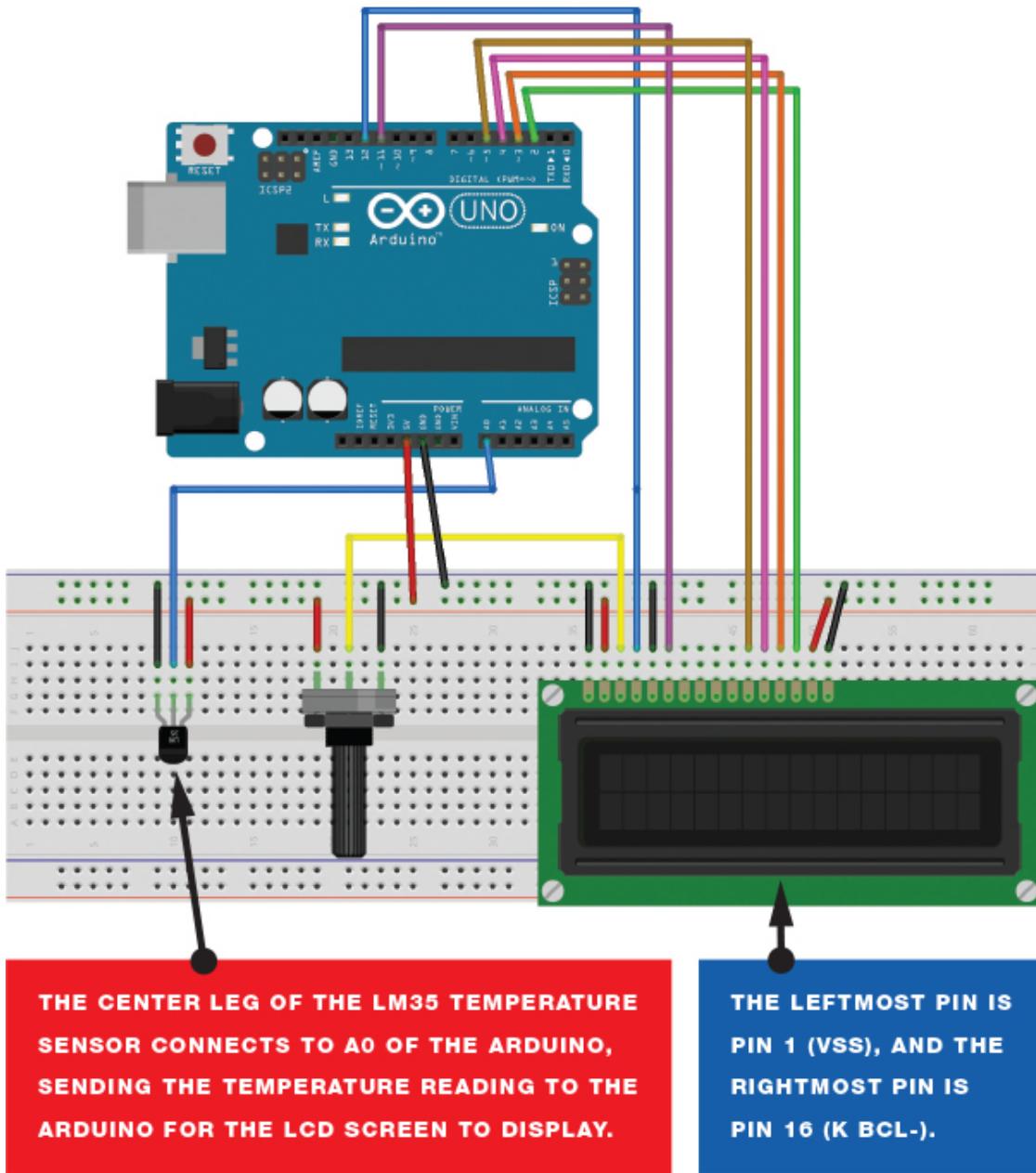
LCD SCREEN	ARDUINO
3 VO contrast	Potentiometer center pin
4 RS	Pin 12
5 R/W	GND
6 Enable	Pin 11
7 D0	No connection
8 D1	No connection
9 D2	No connection
10 D3	No connection
11 D4	Pin 5
12 D5	Pin 4
13 D6	Pin 3
14 D7	Pin 2
15 A BcL+	+5V
16 K BcL-	GND

2. Connect the GND and +5V rails to Arduino GND and +5V.
  3. You should have already connected the center pin of the 50k-ohm potentiometer to LCD pin 3 (VO). Now connect one of the outer pins to GND and the other to +5V.
  4. Connect the center pin of the LM35 temperature sensor to Arduino A0, the left pin to the +5V rail, and the right pin to the GND rail, as shown in the following table.
-

LM35 SENSOR	ARDUINO
Left	+5V
Center	A0
Right	GND

5. Make sure your setup matches the circuit diagram shown in Figure 14-2, and upload the code in “The Sketch” on page 118.

**FIGURE 14-1:** The circuit diagram for the digital thermometer



## THE SKETCH

The sketch uses the LiquidCrystal library to display a value on the screen according to what the LM35 sensor detects. The LM35 sensor sends a reading to Arduino pin A0, which is read as voltage. The sketch converts the voltage reading to a temperature value in Celsius, and then it uses a couple of calculations to show the final reading in Fahrenheit. The sketch updates and displays the reading every second.

```
-----  
#include <LiquidCrystal.h> // Call the LCD library  
#define sensor A0 // Pin connected to LM35 sensor (A0)  
int Vin; // Reads the value from the Arduino pin  
float Temperature; // Receives the voltage value converted to temp  
float TF; // Receives the converted value in °F  
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Pins connected the LCD  
  
void setup() {  
    lcd.begin(16, 2); // The display is 16x2  
    lcd.print("Temperature: "); // Sends text to the LCD  
}  
  
void loop() {  
    // Reads the A0 pin and stores the value in Vin  
    Vin = analogRead (sensor);  
    // Converts voltage value to temperature and  
    // stores value in Temperature (in °C)  
    Temperature = (500 * Vin) / 1023;  
    TF = ((9 * Temperature) / 5) + 32; // Changes °C to °F  
    lcd.setCursor(0, 1); // Move cursor of LCD to next line  
    lcd.print(TF); // Display the temperature on the LCD screen  
    lcd.print(" F"); // Write F for the Fahrenheit scale  
    delay(1000); // Wait for a second before reading the pin again  
}  
-----
```

## TROUBLESHOOTING

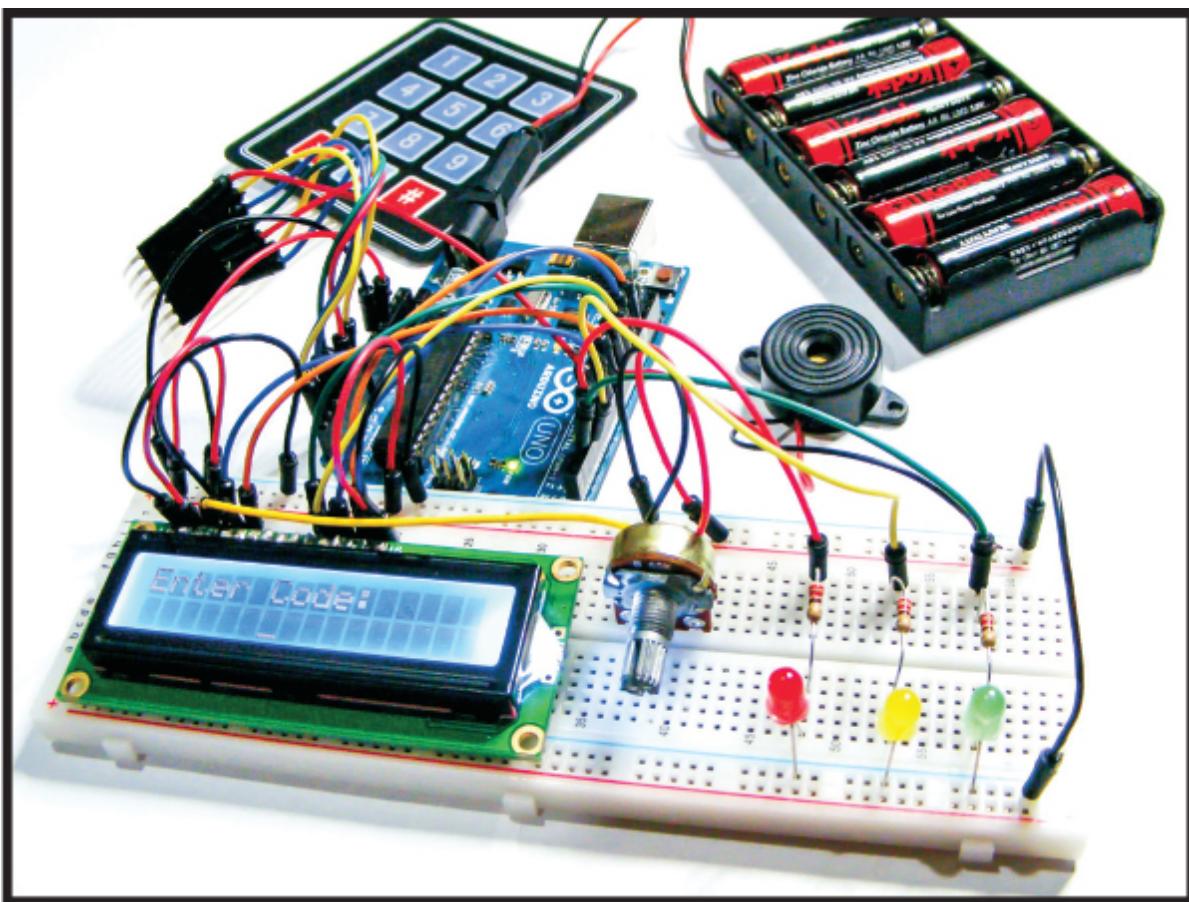
**Q.** *Nothing is displayed on the LCD screen.*

- Make sure you've connected power to the breadboard rails and that the connections match the tables given earlier.
- Turn the potentiometer to change the contrast of the screen until you see text.
- If the screen has garbled messages on it, you probably haven't wired it up correctly; recheck your wiring against Figure 14-2.
- If the value shown seems too high, make sure the LM35 sensor is firmly inserted in the breadboard and allow a moment for the reading to stabilize.

# 15

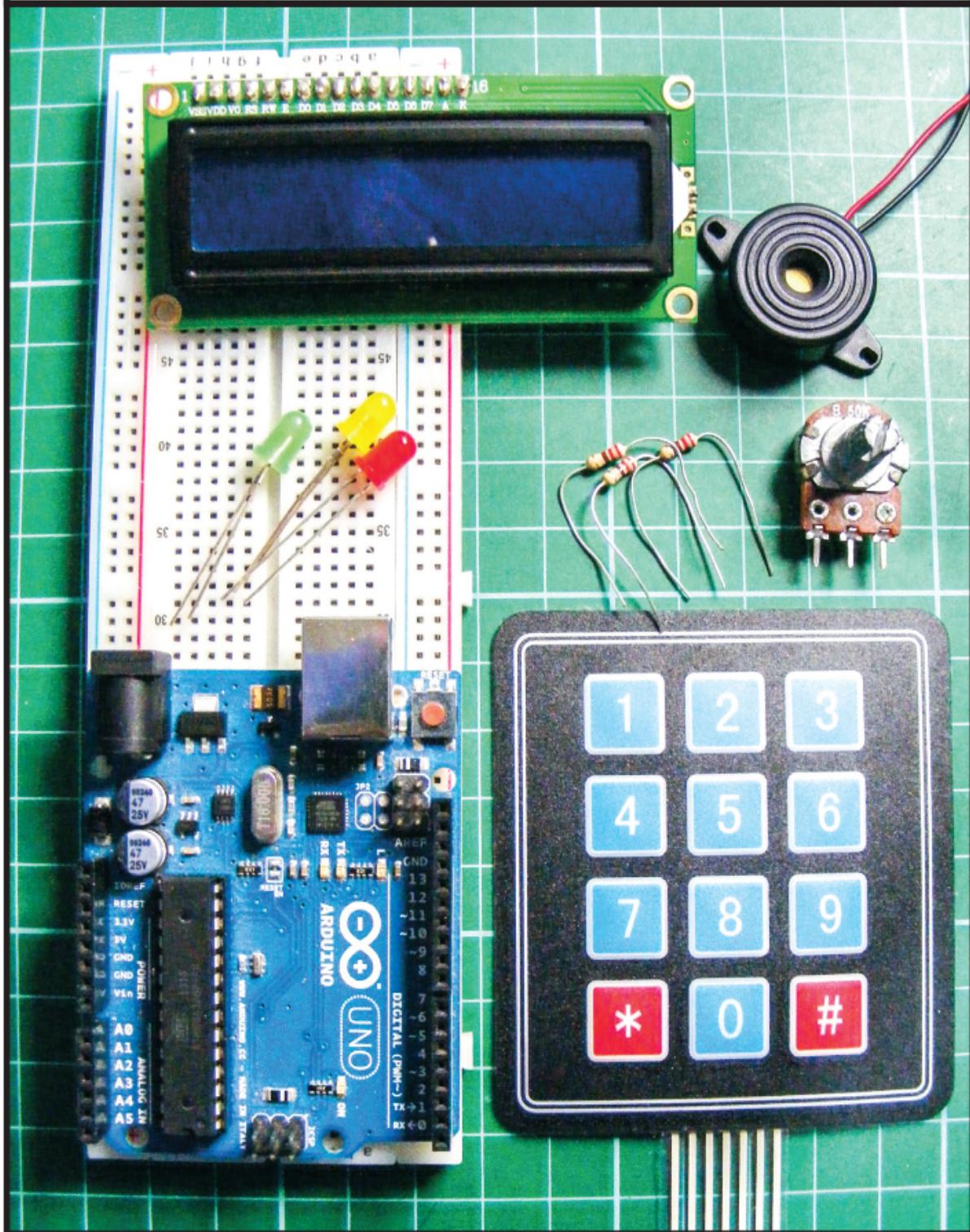
## Bomb Decoder Game

In this project we'll build a code-breaking bomb-decoding game. We'll use an LCD screen and a keypad to give the players instructions and take input.



**COST: \$\$\$**

**TIME: 45 MINUTES**



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**HD44780 16x2 LCD screen**

**10k-ohm potentiometer**

**Piezo sounder**

**3x4 membrane keypad**

**3 220-ohm resistors**

**Red LED**

**Yellow LED**

**Green LED**

## **LIBRARIES REQUIRED**

**LiquidCrystal**

**Keypad**

**Tone**

## **HOW IT WORKS**

When you power up the Arduino, one player enters a four-digit code to start the bomb timer. They give the timer to another player, who presses the \* button to begin decoding the bomb—this player (the “defuser”) must crack the code entered by the first player to defuse the bomb in time. If the defuser presses a wrong key, they can press # to delete their input and start again. If they enter the wrong code or the timer reaches zero, the bomb detonates and they lose.

During the game, the yellow LED flashes and the piezo sounder beeps in time to the countdown. The LCD screen displays the countdown and code input. When the bomb detonates, all the LEDs flash and the piezo sounds an explosion.

A good way to take this game further would be to ask the defuser four questions, each giving the defuser one digit of the bomb code. The

defuser has a set time to answer the questions and input the four-digit code. Answer incorrectly or too late, and the bomb explodes!

## THE BUILD

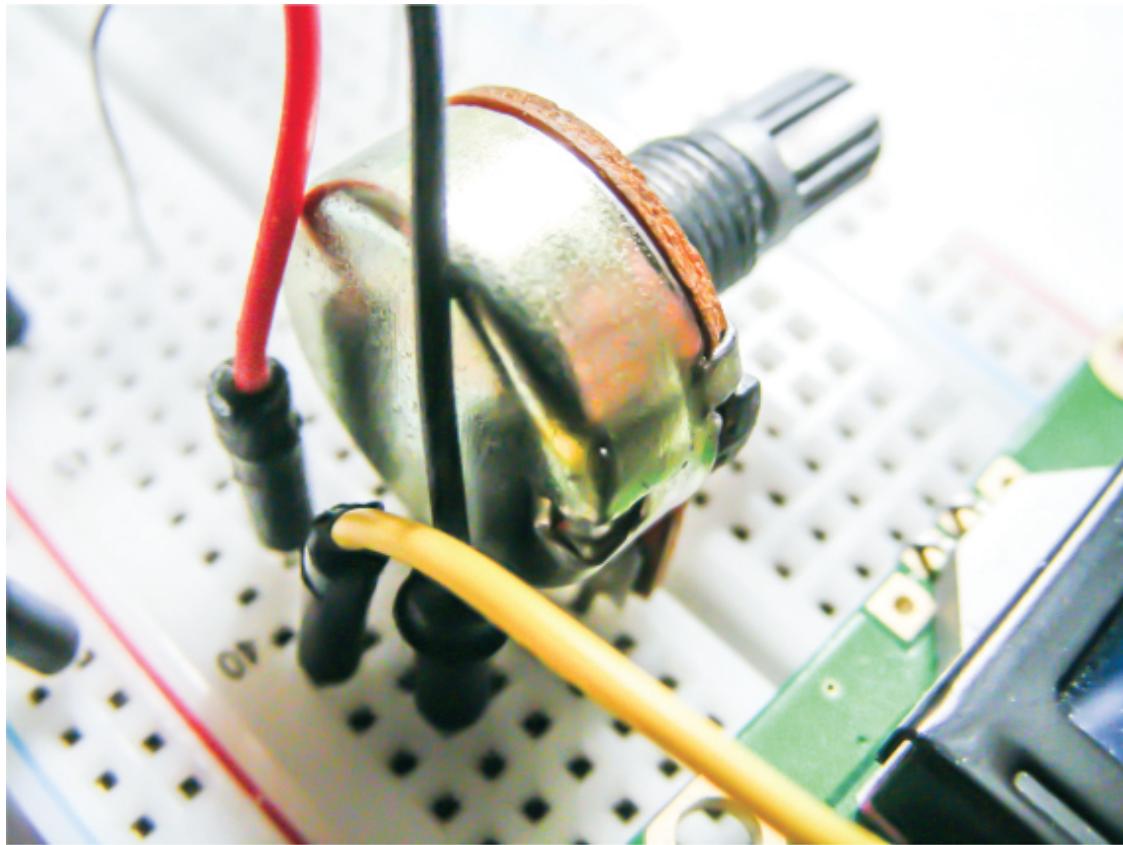
1. If required, prepare the LCD screen by soldering the header pins as described in “Preparing the LCD Screen” on page 109.
2. Place your LCD screen in the breadboard, inserting the header pins into the breadboard holes. Also place the potentiometer in the breadboard, and use the breadboard and jumper wires to connect your LCD screen, Arduino, and potentiometer as shown in the following table. There are multiple GND connections, so use the breadboard rail to make those connections to the Arduino GND pin.

LCD SCREEN	ARDUINO
1 VSS	GND
2 VDD	+5V
3 VO contrast	Potentiometer center pin
4 RS	Pin 7
5 R/W	GND
6 Enable	Pin 8
7 D0	No connection
8 D1	No connection
9 D2	No connection
10 D3	No connection

LCD SCREEN	ARDUINO
11 D4	Pin 10
12 D5	Pin 11
13 D6	Pin 12
14 D7	Pin 13
15 A BcL+	+5V
16 K BcL-	GND

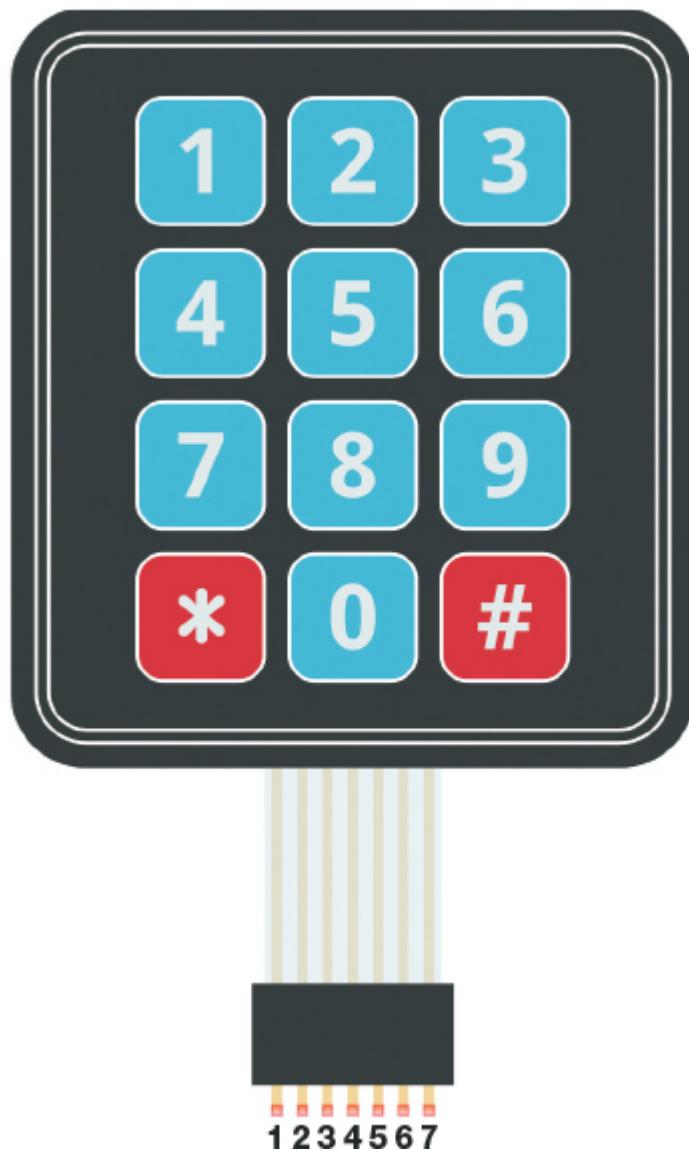
3. You should have already connected the center pin of the 10k-ohm potentiometer to LCD pin 3 (VO). Now connect one of the outer pins to GND and the other to +5V, as shown in Figure 15-1. This controls the contrast of your LCD screen.

**FIGURE 15-1:** The potentiometer controls the contrast of your LCD screen.



4. Looking at the keypad head-on, as in Figure 15-2, the pins are numbered 1–7 from left to right. Connect the keypad pins as shown in the following table.

**FIGURE 15-2:** The 3×4 numeric keypad with seven pin connections



KEYPAD	ARDUINO
Pin 1	Pin 5
Pin 2	Pin A5
Pin 3	Pin A4
Pin 4	Pin A2
Pin 5	Pin A1

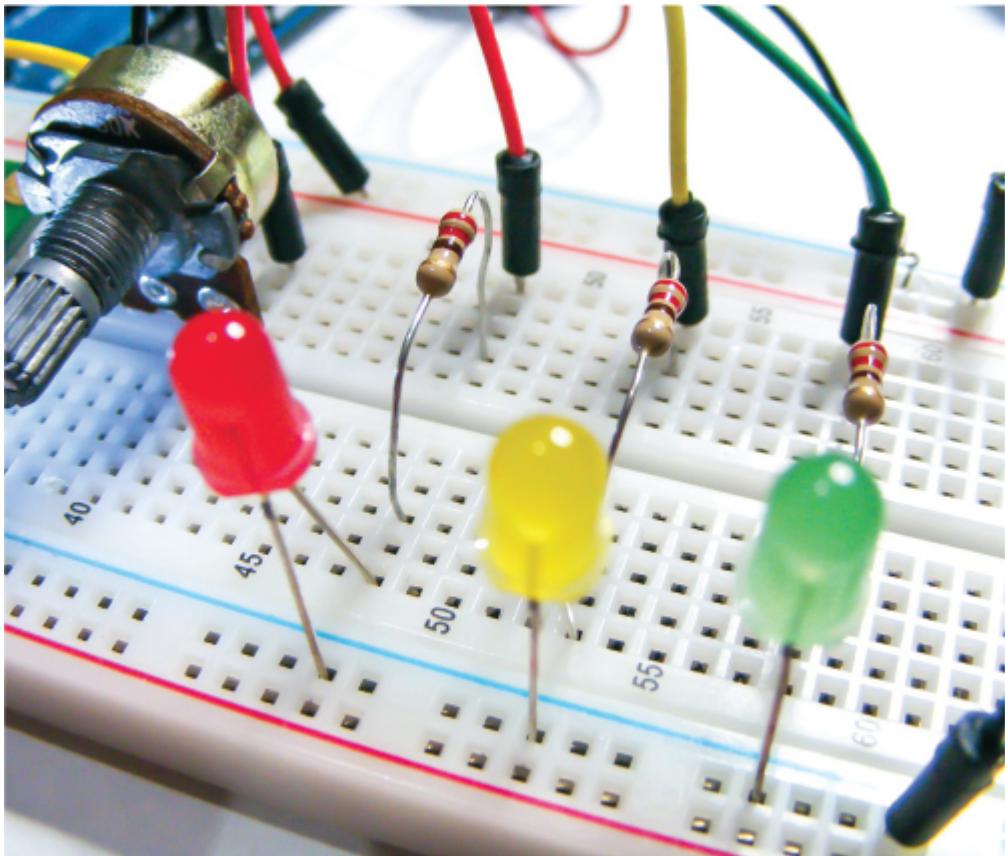
KEYPAD	ARDUINO
Pin 6	Pin A0
Pin 7	Pin A3

5. Connect the piezo sounder's red wire directly to Arduino pin 9 and its black wire to Arduino GND.

PIEZO SOUNDER	ARDUINO
Red wire	Pin 9
Black wire	GND

6. Place the green LED in the breadboard, connecting the short, negative leg to the negative breadboard rail via a 220-ohm resistor. Connect the green LED's long, positive leg to pin 2. Do the same with the yellow LED to pin 3 and the red LED to pin 4, as shown in Figure 15-3 and the table that follows.

**FIGURE 15-3:** Connect the LEDs to the Arduino via a 220-ohm resistor.

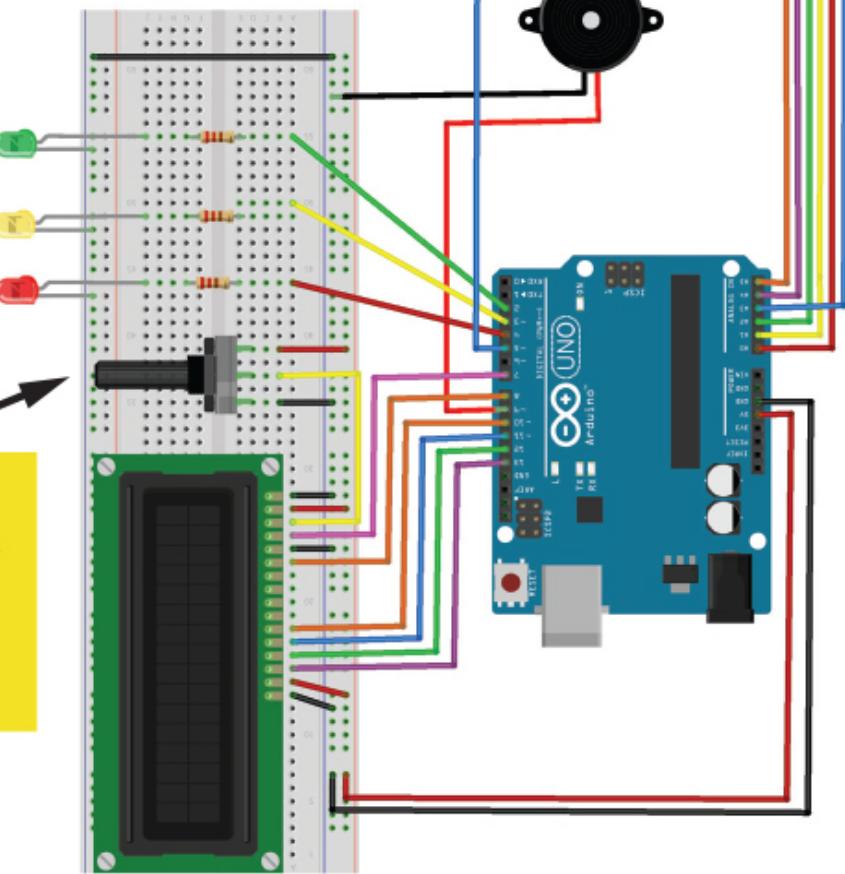


LEDS	ARDUINO
Negative legs	GND
Green positive leg	Pin 2 via 220-ohm resistor
Yellow positive leg	Pin 3 via 220-ohm resistor
Red positive leg	Pin 4 via 220-ohm resistor

7. Connect the positive and negative breadboard power rails to +5V and GND, respectively.
8. Make sure your completed project circuit matches Figure 15-4, remember to add the required libraries to your *Libraries* folder, and then upload the code in “The Sketch” on page 127.

**FIGURE 15-4:** The circuit diagram for the bomb decoder game

ENTER YOUR PASSWORD HERE UPON  
STARTUP. PRESS \* TO BEGIN DEFUSING;  
PRESS # TO DELETE YOUR INPUT.



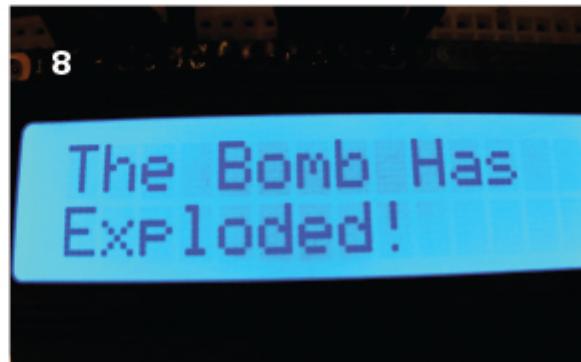
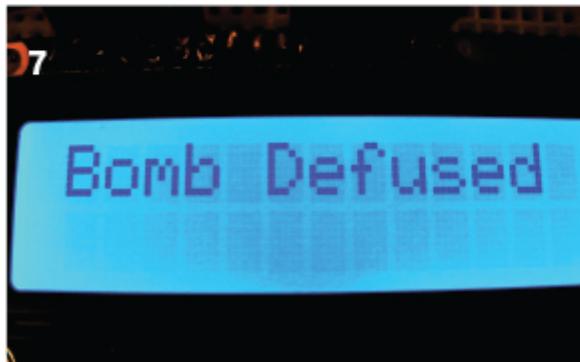
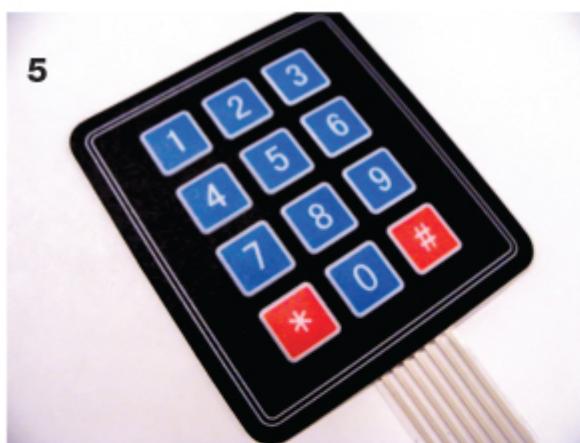
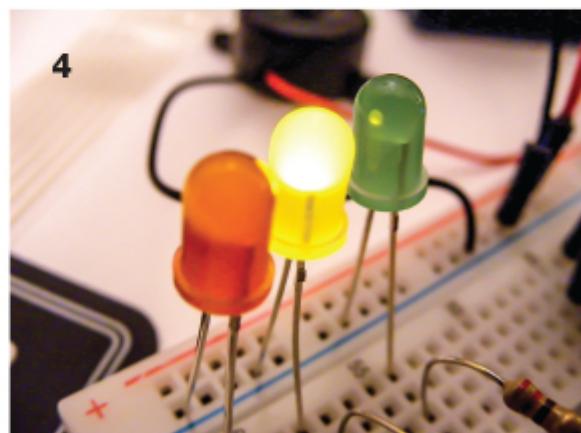
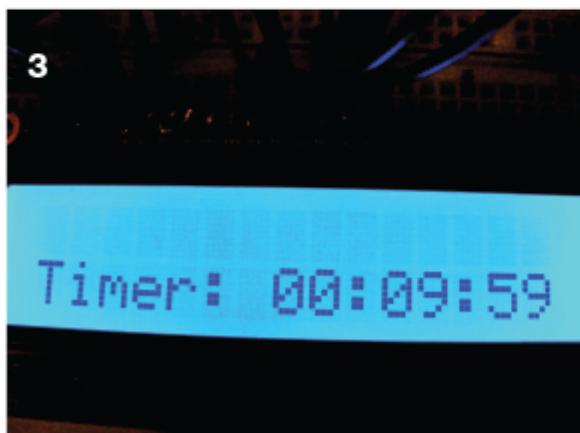
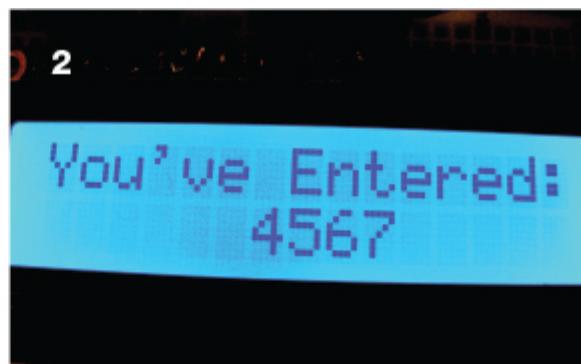
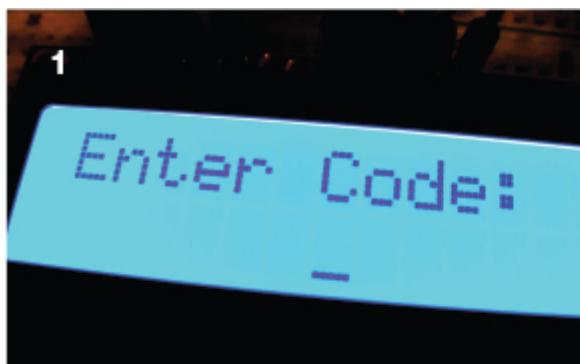
THE LCD SCREEN WILL SHOW THE

COUNTDOWN UNTIL THE EXPLOSION.

## PLAYING THE GAME

Figure 15-5 shows the different stages of playing the game.

**FIGURE 15-5:** Playing the game



1. Enter the code to set up the bomb.
2. The bomb confirms the code entered.
3. The timer starts the countdown sequence.
4. The yellow LED flashes in time to the countdown.
5. Pass the keypad to another player (the defuser). They press the \* button on the keypad, then enter the defuse code.
6. The screen does not show the numbers entered to defuse the bomb.
7. If the correct code is entered, the bomb is defused . . .
8. . . . but if not . . . Boom!

### NOTE

*All libraries and code can be downloaded from  
<https://www.nostarch.com/arduinohandbook2/>.*

## THE SKETCH

The sketch calls on the Keypad, LiquidCrystal, and Tone libraries. LiquidCrystal is included in your IDE, but you'll have to download Keypad and Tone from the book's resources at <https://www.nostarch.com/arduinohandbook2/> and save them in your Libraries folder for the Arduino (see the primer for details on how to do that if you're unsure).

First the sketch defines the timer duration, password length, LED pins, and keypad. It requests a code input from the first player by displaying “Enter Code:” and then stores that value as the bomb defusal code. When the second player (the defuser) presses \*, the timer starts and waits for a code to be entered, and the yellow LED flashes in time to the countdown. If the code the defuser enters does not match the defusal code, the text “The Bomb Has Exploded!” displays on the

screen and the LEDs and piezo indicate an explosion. If the defuser's input is correct, the timer stops, the green LED lights, and the message "Bomb Defused" displays on the screen. The bomb will also explode if the timer reaches zero with no input. When the game ends, the code resets, ready for another game.

---

```
// Original code by Joey Meyer and Chase Cooley
// and used with kind permission

#include <Keypad.h>
#include <LiquidCrystal.h>
#include <Tone.h>

Tone tone1;

int Scount = 10; // Change this to the number of seconds to start from
int Mcount = 5; // Change this to the number of minutes to start from
int Hcount = 0; // Count hours
int DefuseTimer = 0; // Set timer to 0

long secMillis = 0; // Store last time for second add
long interval = 1000; // Interval for seconds

char password[4]; // Number of characters in password
int currentLength = 0; // Defines number currently writing
int i = 0;
char entered[4];

int ledPin = 4; // Red LED
int ledPin2 = 3; // Yellow LED
int ledPin3 = 2; // Green LED
// The pins we use on the LCD
LiquidCrystal lcd(7, 8, 10, 11, 12, 13);

const byte ROWS = 4; // Four rows
const byte COLS = 3; // Three columns
char keys[ROWS][COLS] = {
    {'1', '2', '3'},
    {'4', '5', '6'},
    {'7', '8', '9'},
    {'*', '0', '#'}
};
byte rowPins[ROWS] = {5, A5, A4, A2}; // Connect to the row pinouts
// of the keypad
byte colPins[COLS] = {A1, A0, A3}; // Connect to the column pinouts
// of the keypad

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
```

```

void setup() {
    pinMode(ledPin, OUTPUT); // Sets the digital pin as output
    pinMode(ledPin2, OUTPUT); // Sets the digital pin as output
    pinMode(ledPin3, OUTPUT); // Sets the digital pin as output
    tone1.begin(9);
    lcd.begin(16, 2);
    Serial.begin(9600);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Enter Code: ");
    while (currentLength < 4) {
        lcd.setCursor(currentLength + 6, 1);
        lcd.cursor();
        char key = keypad.getKey();
        key == NO_KEY;
        if (key != NO_KEY) {
            if ((key != '*')&&(key != '#')) {
                lcd.print(key);
                password[currentLength] = key;
                currentLength++;
                tone1.play(NOTE_C6, 200);
            }
        }
    }

    if (currentLength == 4) {
        delay(500);
        lcd.noCursor();
        lcd.clear();
        lcd.home();
        lcd.print("You've Entered: ");
        lcd.setCursor(6, 1);
        lcd.print(password[0]);
        lcd.print(password[1]);
        lcd.print(password[2]);
        lcd.print(password[3]);
        tone1.play(NOTE_E6, 200);
        delay(3000);
        lcd.clear();
        currentLength = 0;
    }
}

void loop() {
    timer();
    char key2 = keypad.getKey(); // Get the key
    if (key2 == '*') {
        lcd.clear();
        lcd.setCursor(0, 0);
    }
}

```

```

lcd.print("Code: ");
while (currentLength < 4) {
    timer();
    char key2 = keypad.getKey();
    if (key2 == '#') {
        currentLength = 0;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Code: ");
    }
    else if (key2 != NO_KEY) {
        lcd.setCursor(currentLength + 7, 0);
        lcd.cursor();
        lcd.print(key2);
        entered[currentLength] = key2;
        currentLength++;
        tone1.play(NOTE_C6, 200);
        delay(100);
        lcd.noCursor();
        lcd.setCursor(currentLength + 6, 0);
        lcd.print("*");
        lcd.setCursor(currentLength + 7, 0);
        lcd.cursor();
    }
}
if (currentLength == 4) {
    if (entered[0] == password[0] && entered[1] == password[1] && entered[2] ==
password[2] && entered[3] == password[3]) {
        lcd.noCursor();
        lcd.clear();
        lcd.home();
        lcd.print("Bomb Defused");
        currentLength = 0;
        digitalWrite(ledPin3, HIGH);
        delay(2500);
        lcd.setCursor(0, 1);
        lcd.print("Reset the Bomb");
        delay(1000000);
    }
}

else {
    lcd.noCursor();
    lcd.clear();
    lcd.home();
    lcd.print("Wrong Password!");
    if (Hcount > 0) {
        Hcount = Hcount - 1;
    }
    if (Mcount > 0) {
        Mcount = Mcount - 59;
}

```

```

        }
        if (Scount > 0) {
            Scount = Scount - 59;
        }
        delay(1500);
        currentLength = 0;
    }
}
}

void timer() {
    Serial.print(Scount);
    Serial.println();
    if (Hcount <= 0) { // If timer reaches 0, LCD displays explosion
        if (Mcount < 0) {
            lcd.noCursor();
            lcd.clear();
            lcd.home();
            lcd.print("The Bomb Has ");
            lcd.setCursor(0, 1);
            lcd.print("Exploded!");
            while (Mcount < 0) {
                digitalWrite(ledPin, HIGH); // Sets the LED on
                tone1.play(NOTE_A2, 90);
                delay(100);
                digitalWrite(ledPin, LOW); // Sets the LED off
                tone1.play(NOTE_A2, 90);
                delay(100);
                digitalWrite(ledPin2, HIGH); // Sets the LED on
                tone1.play(NOTE_A2, 90);
                delay(100);
                digitalWrite(ledPin2, LOW); // Sets the LED off
                tone1.play(NOTE_A2, 90);
                delay(100);
                digitalWrite(ledPin3, HIGH); // Sets the LED on
                tone1.play(NOTE_A2, 90);
                delay(100);
                digitalWrite(ledPin3, LOW); // Sets the LED off
                tone1.play(NOTE_A2, 90);
                delay(100);
            }
        }
    }
}

lcd.setCursor(0, 1); // Sets cursor to 2nd line
lcd.print("Timer:");

if (Hcount >= 10) {
    lcd.setCursor(7, 1);
}

```

```

    lcd.print(Hcount);
}
if (Hcount < 10) {
    lcd.setCursor(7, 1);
    lcd.write("0");
    lcd.setCursor(8, 1);
    lcd.print(Hcount);
}

lcd.print(":");

if (Mcount >= 10) {
    lcd.setCursor(10, 1);
    lcd.print(Mcount);
}
if (Mcount < 10) {
    lcd.setCursor(10, 1);
    lcd.write("0");
    lcd.setCursor(11, 1);
    lcd.print(Mcount);
}

lcd.print (":");

if (Scount >= 10) {
    lcd.setCursor(13, 1);
    lcd.print(Scount);
}
if (Scount < 10) {
    lcd.setCursor(13, 1);
    lcd.write("0");
    lcd.setCursor(14, 1);
    lcd.print(Scount);
}

if (Hcount < 0) {
    Hcount = 0;
}

if (Mcount < 0) {
    Hcount--;
    Mcount = 59;
}

if (Scount < 1) { // If 60 do this operation
    Mcount--; // Add 1 to Mcount
    Scount = 59; // Reset Scount
}

if (Scount > 0) { // Do this operation 59 times

```

```
unsigned long currentMillis = millis();
if (currentMillis - secMillis > interval) {
    tone1.play(NOTE_G5, 200);
    secMillis = currentMillis;
    Scount --; // Add 1 to Scount
    digitalWrite(ledPin2, HIGH); // Sets the LED on
    delay(10); // Waits for a second
    digitalWrite(ledPin2, LOW); // Sets the LED off
    delay(10); // Waits for a second
}
}
```

---

## TROUBLESHOOTING

**Q.** *Nothing is displayed on the LCD screen.*

- Make sure you've connected power to the breadboard rails and the connections match the tables in this chapter.
- Turn the potentiometer to change the contrast of the screen until you see text.
- If the screen has garbled messages on it, you haven't wired it up correctly; recheck your wiring against the circuit diagram in Figure 15-4.

**Q.** *The LEDs do not light when expected.*

- Check your wiring against the circuit diagram in Figure 15-4 and ensure that the short leg of the LED is connected to the ground rail of the breadboard.
- It's easy to forget to add power to the breadboard rails, so make sure you connect the ground and power rails on either side of the breadboard to the Arduino with a jumper wire.
- Check that your LEDs and resistors are firmly inserted into the breadboard and they line up with one another.
- If the wrong LED lights up, you've probably connected to the wrong pin numbers by mistake, so just change them around.

**Q.** *The piezo sounder does not make a noise.*

- The positive red wire of the sounder should be connected to pin 9 and the black ground wire to GND. If the sounder does still not make a noise, try replacing it with another one.

**Q.** *When the keypad is pressed, the numbers are incorrect or do not register:*

- Make sure the connections of the keypad to the Arduino match the circuit diagram in Figure 15-4 exactly.
- The configuration is set up specifically for this project's 3x4 numeric keypad, so if your keypad is different, check the data sheet to find out which pins you need to connect.

# 16

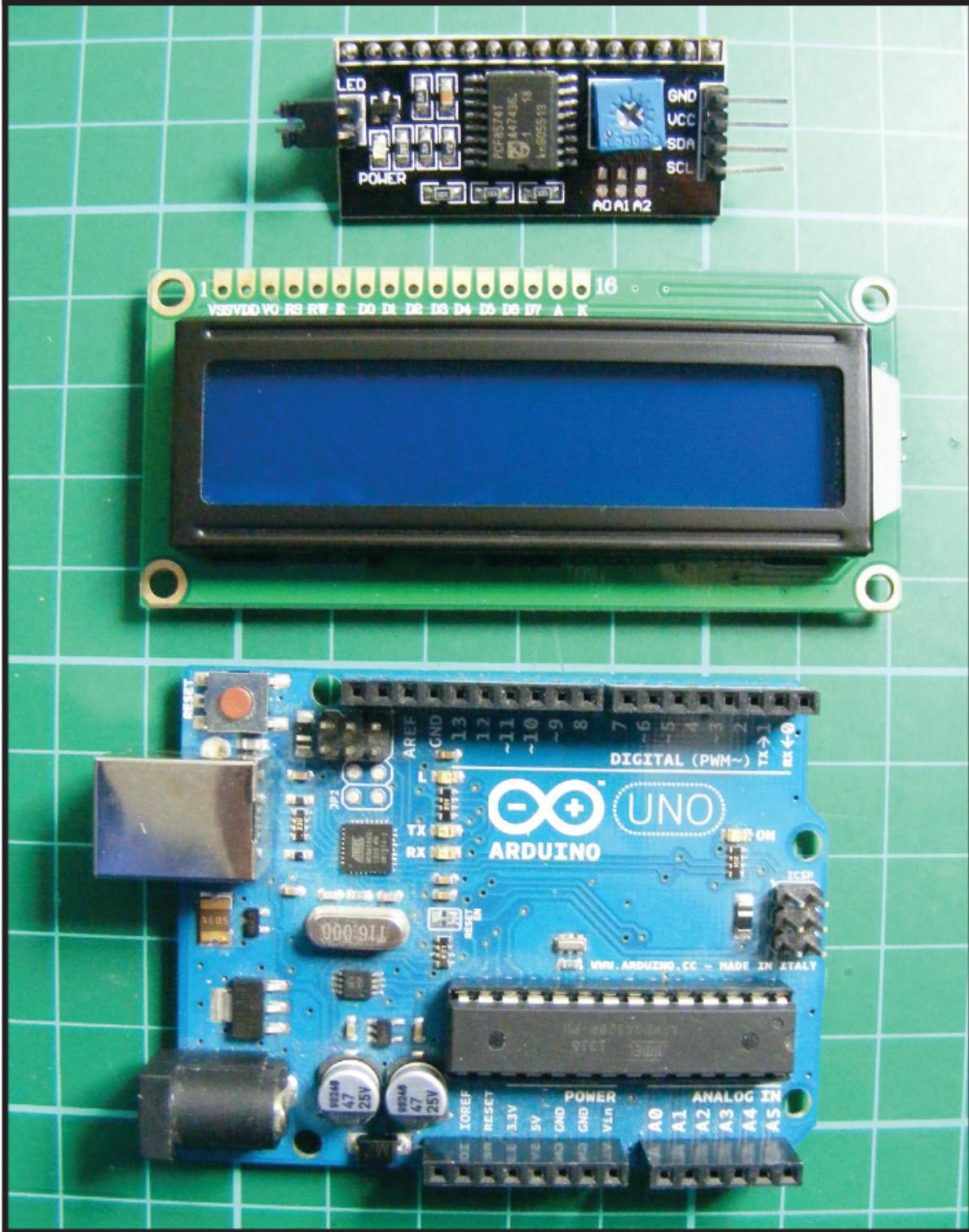
## Serial LCD Screen

In this project we'll take a 16×2 character LCD screen and a serial module to create a serial LCD that's controlled by only two wires.



COST: \$

TIME: 30 MINUTES



## PARTS REQUIRED

**Arduino board**  
**Female-to-male jumper wires**  
**HD44780 16×2 LCD screen**  
**Serial LCD screen module**

## **LIBRARIES REQUIRED**

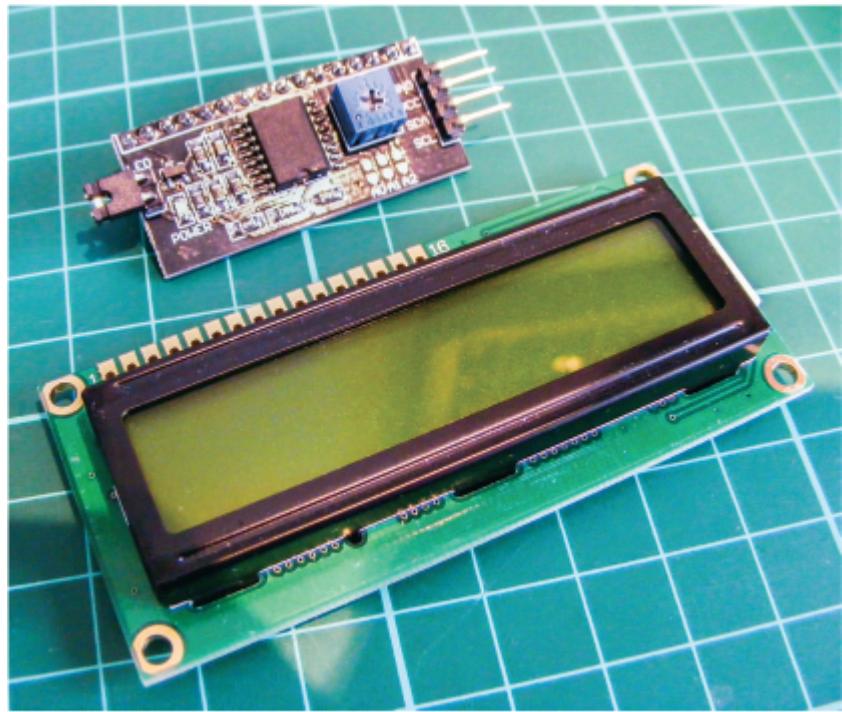
**Wire**  
**LiquidCrystal\_I2C**

## **HOW IT WORKS**

LCD screens are very useful in projects, but they use up a lot of pins on the Arduino. This means that if you're incorporating them into a more complex project, you might run out of pins. Thankfully there is a solution: use a *serial* LCD screen. Serial LCDs use the communication protocol *I2C*, which stands for *Inter-Integrated Circuit*, and differ from normal 16×2 LCD screens in that they can be controlled by your Arduino with only power and two pins.

Serial LCD screens usually come in kit form and require you to solder header pins, which I'll cover later in the chapter. You'll usually receive the 16×2 LCD screen and the serial module separately, as shown in Figure 16-1.

**FIGURE 16-1:** 16×2 LCD screen and serial module



## PREPARING THE SERIAL LCD SCREEN

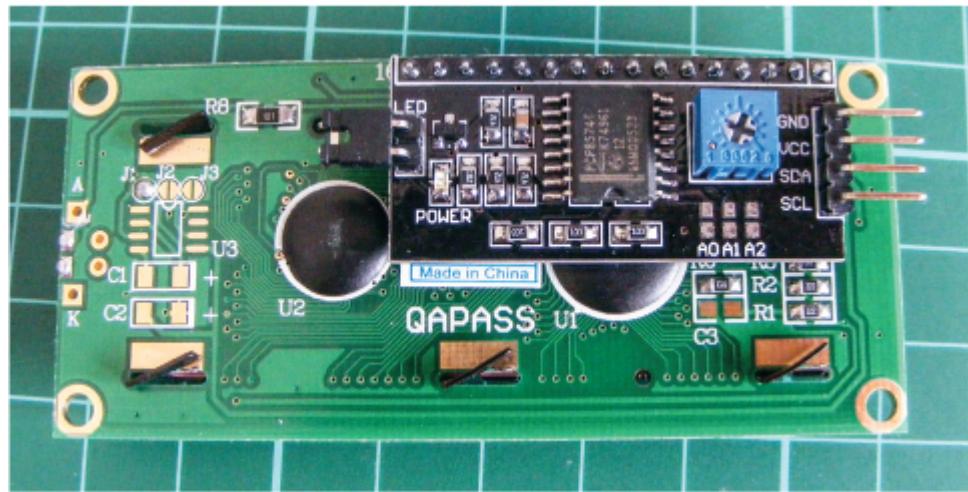
1. The serial module has a strip of 16 header pins already attached to one side. Turn the LCD screen over and you'll see 16 corresponding holes, as shown in Figure 16-2.

**FIGURE 16-2:** The reverse side of the LCD screen



2. Place the serial controller header pins into those holes, as shown in Figure 16-3.

**FIGURE 16-3:** Insert the serial module into the LCD screen holes.



3. Carefully add a small amount of solder to each of the pins to make a connection and hold the serial monitor to the screen. Turn to the primer for a quick soldering guide.

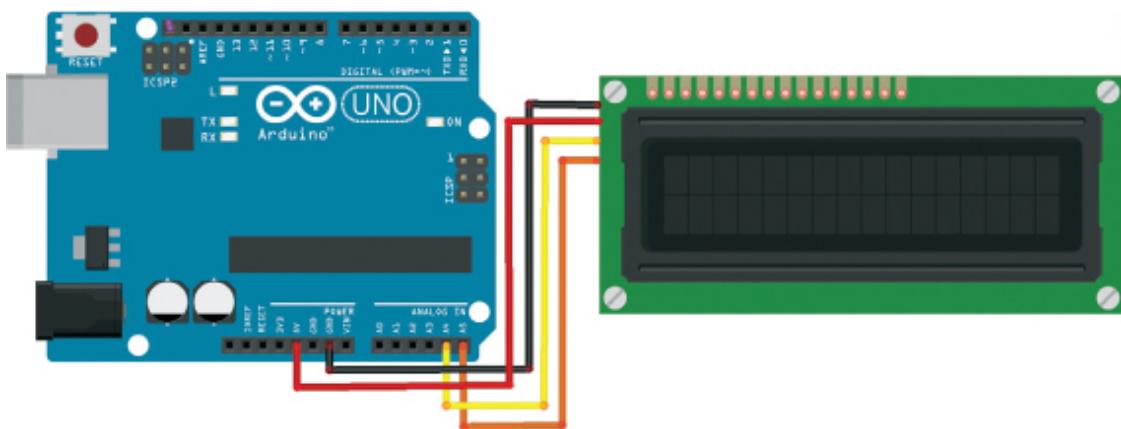
## THE BUILD

Your serial LCD screen has an assigned address that your Arduino needs in order to communicate with it. The addresses differ depending on the make, so you need to check the address of your specific screen, as you'll need it for the sketch later. To check the address, connect the LCD screen to your Arduino and run a quick sketch to scan the module—or you could also refer to the data sheet for your screen.

1. Connect your female-to-male jumper wires to the four pins on the controller for the LCD screen.
2. Wire up the serial LCD screen to the Arduino with GND to GND, VCC to +5V, SDA to Arduino pin A4, and SCL to Arduino pin A5, as shown in the following table and the circuit diagram in Figure 16-4.

SERIAL LCD SCREEN	ARDUINO
GND	GND
VCC	+5V
SDA	Pin A4 (SDA)
SCL	Pin A5 (SCL)

**FIGURE 16-4:** The circuit diagram for the serial LCD screen



3. Upload the following sketch to the Arduino. We'll get the address in *hexadecimal*, a number system that uses letters and numbers in an abbreviated form to represent a much larger number.

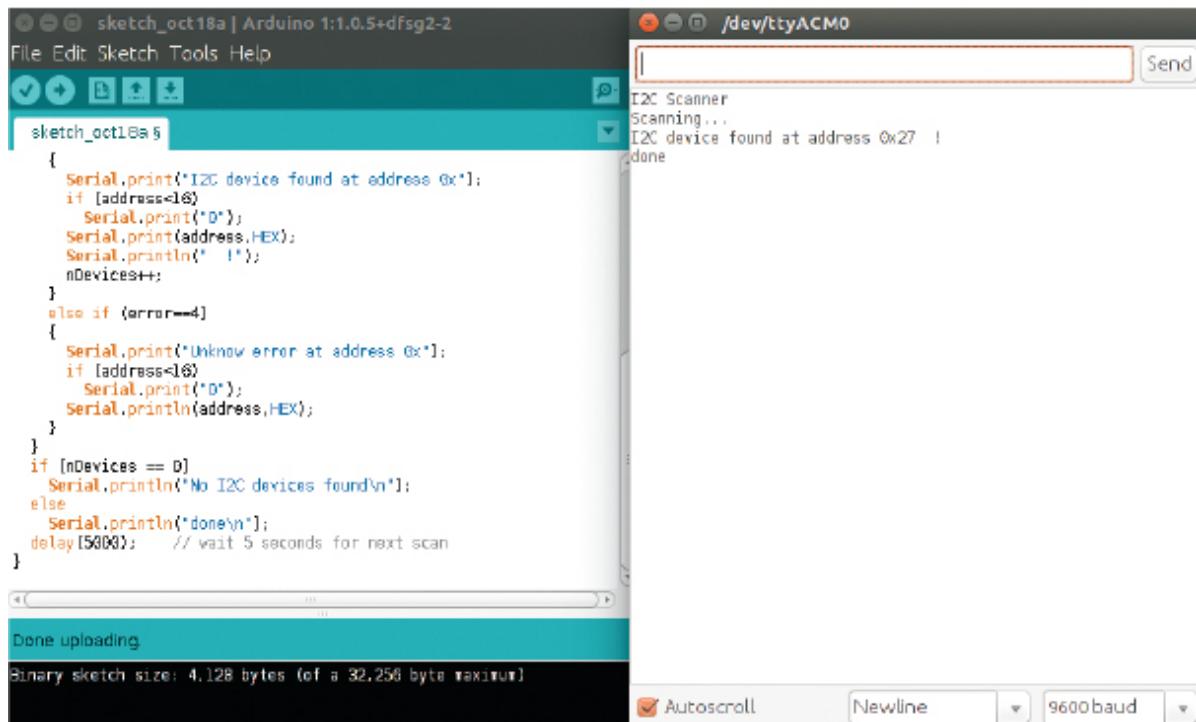
```
#include <Wire.h>
void setup() {
    Wire.begin();
    Serial.begin(9600);
    Serial.println("I2C Scanner");
}
void loop() {
    byte error, address;
    int nDevices;
    Serial.println("Scanning...");
    nDevices = 0;
    for (address = 1; address < 127; address++) {
        Wire.beginTransmission(address);
        error = Wire.endTransmission();
        if (error == 0) {
            Serial.print("I2C device found at address 0x");
            Serial.println(String(address, HEX));
            nDevices++;
        }
    }
    if (nDevices == 0)
        Serial.println("No I2C devices found");
}
```

```

    if (address < 16)
        Serial.print("0");
    Serial.print(address, HEX);
    Serial.println(" !");
    nDevices++;
}
else if (error == 4) {
    Serial.print("Unknown error at address 0x");
    if (address < 16)
        Serial.print("0");
    Serial.println(address, HEX);
}
if (nDevices == 0)
    Serial.println("No I2C devices found\n");
else
    Serial.println("done\n");
delay(5000); // Wait 5 seconds for next scan
}
-----
```

The sketch scans for all addresses on the Arduino's I2C bus and displays the output in the Serial Monitor, as shown in Figure 16-5.

**FIGURE 16-5:** The hexadecimal number of your module will be shown in the IDE Serial Monitor.



The address is the number that comes after the 0x. In my case that is 27, so I need to make a note of 0x27. You'll use this address in the final sketch.

## THE SKETCH

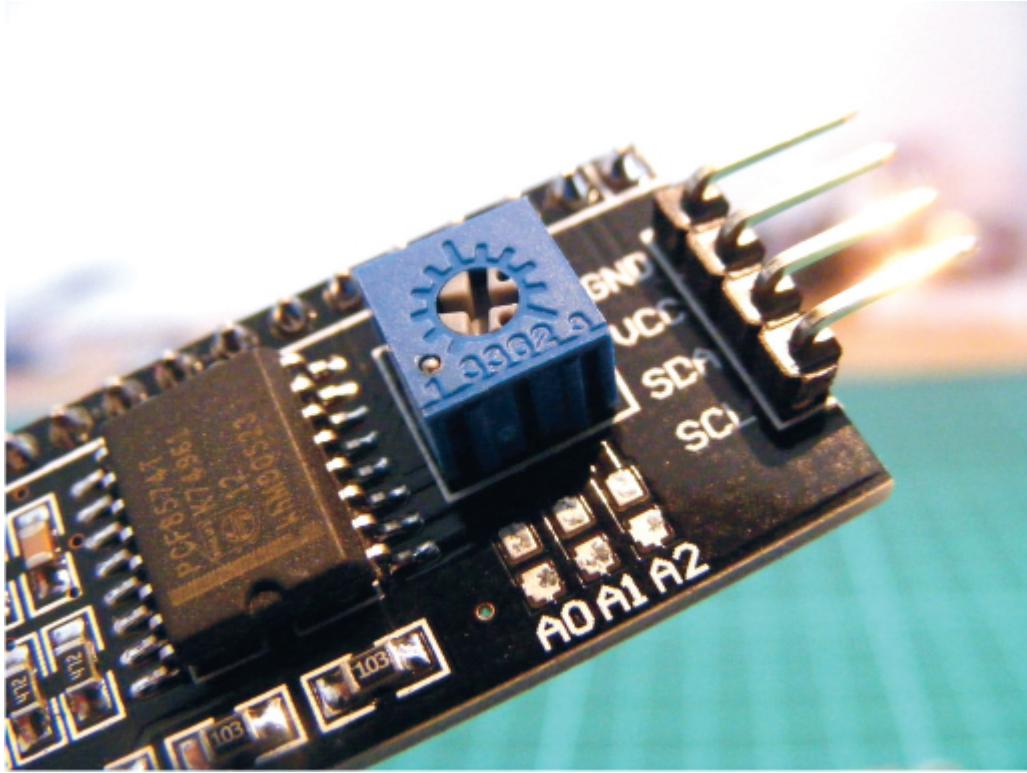
This sketch calls on the Wire and LiquidCrystal\_I2C libraries. The Wire library is included in the Arduino IDE, but you will need to install the LiquidCrystal\_I2C library by downloading it from <https://www.nostarch.com/arduinohandbook2/>. The libraries allow the Arduino to control the module using serial communication via just the SDA and SCL pins.

Change the code at ❶ so that the 0x27 is replaced with the address you just noted from your scan in the test sketch.

```
-----  
#include <Wire.h> // Call the wire library  
#include <LiquidCrystal_I2C.h> // Call the I2C library  
LiquidCrystal_I2C lcd(0x27❶,16,2); // Set LCD address to 0x27 for a  
// 16-character and 2-line display  
  
void setup() {  
    lcd.begin(); // Initialize the lcd  
    lcd.backlight();  
    lcd.print("Arduino Handbook"); // Print a message to the LCD  
}  
  
void loop() { // Loop around again  
}  
-----
```

There is a potentiometer built into the module to control the contrast of the LCD screen, shown in Figure 16-6. Turn this carefully with a small screwdriver until the contrast on the screen looks right.

**FIGURE 16-6:** The small blue box on the back of the module is a potentiometer to control the contrast.



## TROUBLESHOOTING

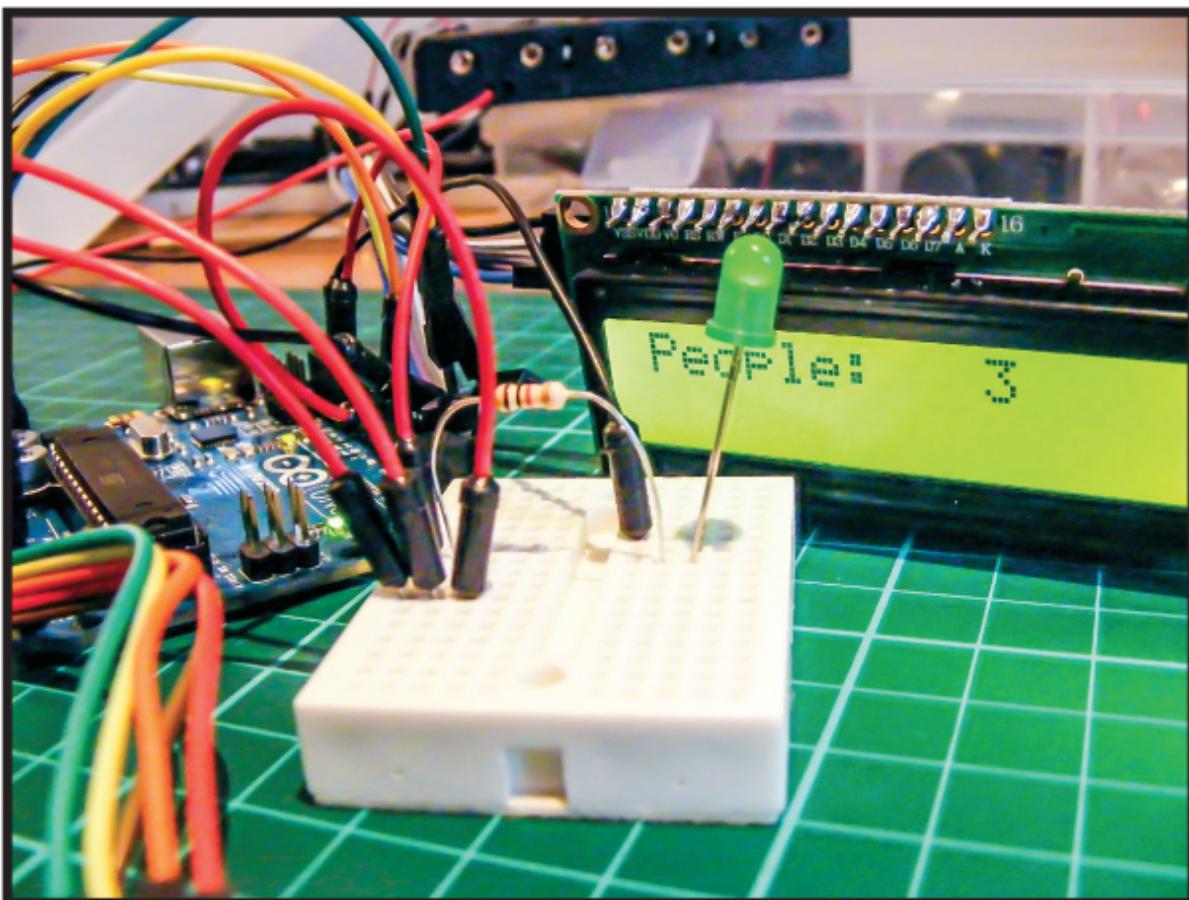
**Q.** *The code compiles, but nothing shows on the screen.*

- Double-check that the SDA and SCL pins are connected to the correct Arduino pins. If the LCD screen is lit but shows no characters, carefully turn the small potentiometer at the back of the module until the letters appear.
- If the screen still shows nothing and all the connections are correct, it may be that the solder on the header pins is not making a clean connection or you have soldered more than one pin together. Heat the area again with your soldering iron to melt the solder, and then use a solder sucker to remove any excess and resolder the header pins.

17

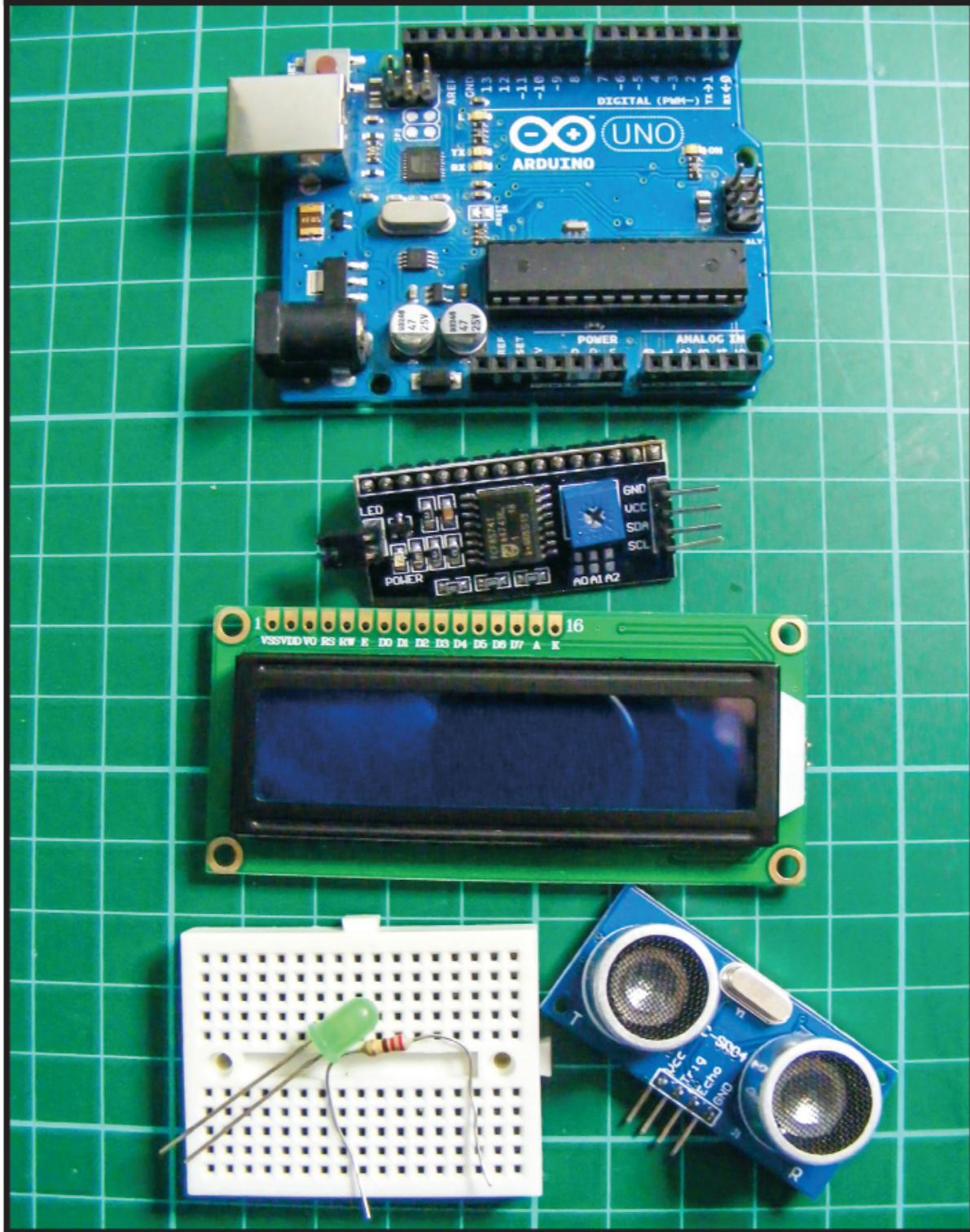
## Ultrasonic People Counter

This project teaches you how to use the HC-SR04 ultrasonic sensor to sense when people pass and then show that count on a serial LCD screen.



COST: \$\$

TIME: 20 MINUTES



## PARTS REQUIRED

**Arduino board**  
**Mini-breadboard**  
**Jumper wires, male-to-male and female-to-male**  
**LED**  
**Serial LCD screen module**  
**220-ohm resistor**  
**HC-SR04 ultrasonic sensor**

## **LIBRARIES REQUIRED**

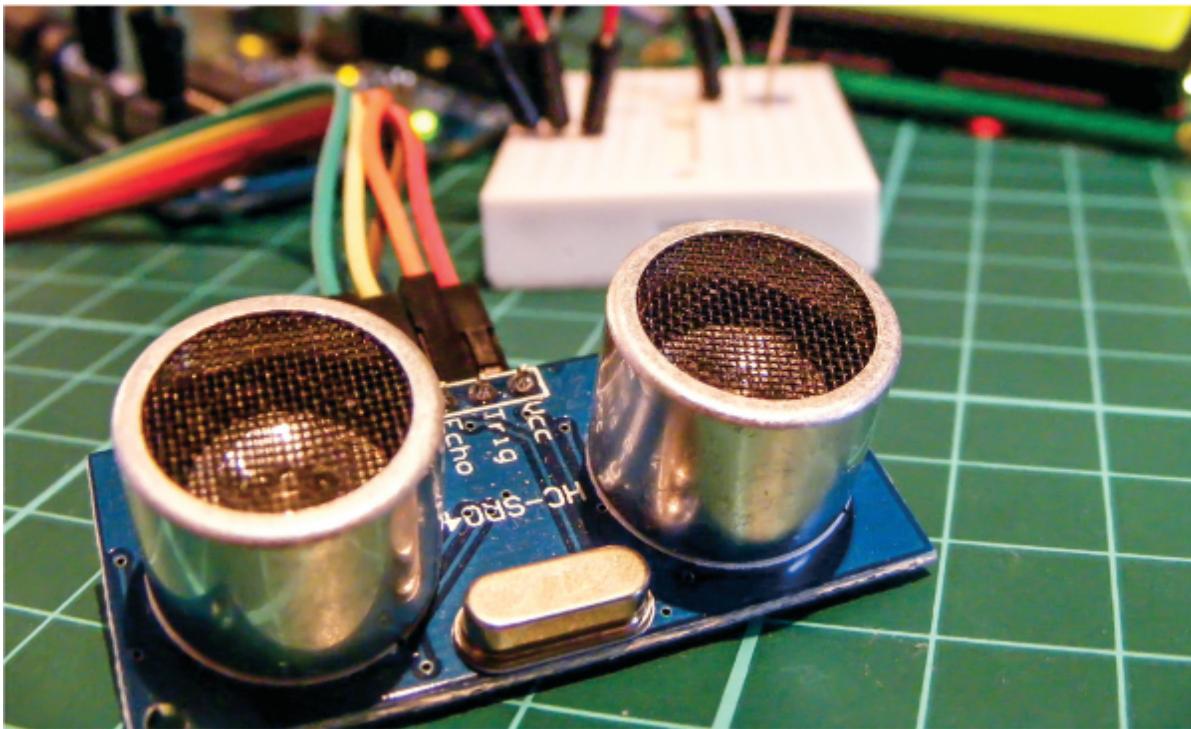
**NewPing**  
**Wire**  
**LiquidCrystal\_I2C**

## **HOW IT WORKS**

People counters are often used in shops or tourist attractions to count the number of visitors, but you could also use one to record the volume of traffic on a highway or in a parking lot, or to count how many times someone entered your room while you were out!

The ultrasonic sensor we'll use is the HC-SR04, shown in Figure 17-1, which you first saw in Project 13. It uses an ultrasonic signal, or *ping*, to calculate the distance between the sensor and an object. In this project we'll use this function to count every time someone or something passes in front of the sensor. An LED will flash when a count is registered, and the serial LCD screen will show the total number counted.

**FIGURE 17-1:** The HC-SR04 ultrasonic sensor uses a ping to calculate distances.

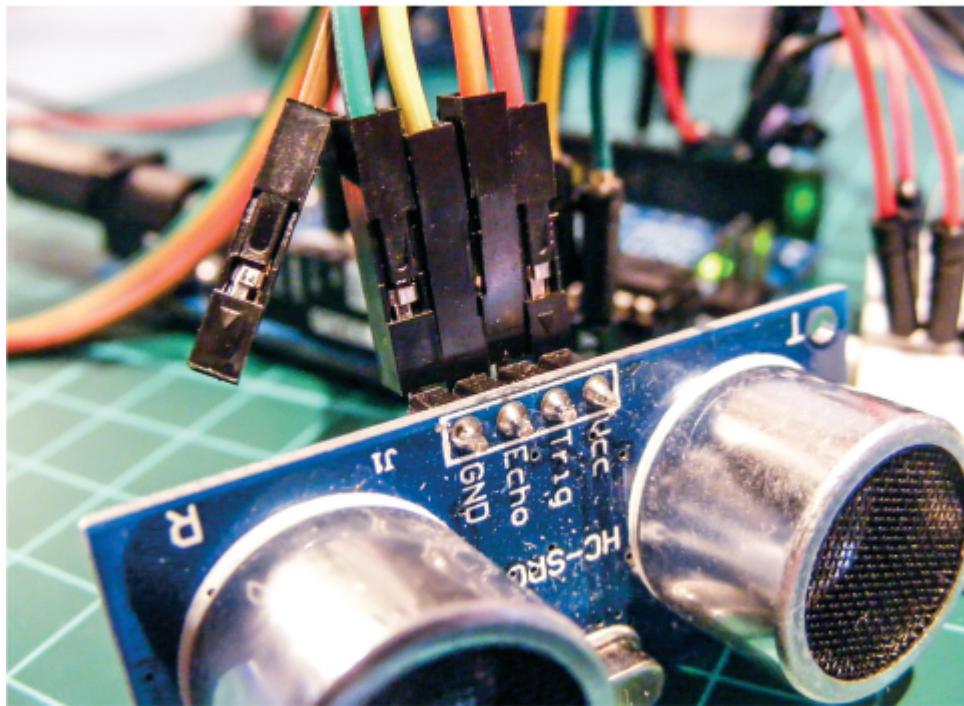


## THE BUILD

1. Use the female-to-male jumper wires to connect the HC-SR04 ultrasonic sensor to the Arduino with the VCC pin to Arduino +5V, GND to GND, and Trig and Echo to pins 7 and 8 on the Arduino, respectively, as shown in the following table and in Figure 17-2. Use the mini-breadboard for multiple connections.

ULTRASONIC SENSOR	ARDUINO
VCC	+5V
Trig	Pin 7
Echo	Pin 8
GND	GND

**FIGURE 17-2:** The connections from the ultrasonic sensor



2. Make sure to download the LiquidCrystal I2C and NewPing libraries and add them to the relevant folder on your computer (see the primer for guidance). The Wire library comes with the Arduino IDE, so you do not need to add it.
3. Connect the serial LCD screen to the Arduino as follows, using the mini-breadboard to connect to +5V.

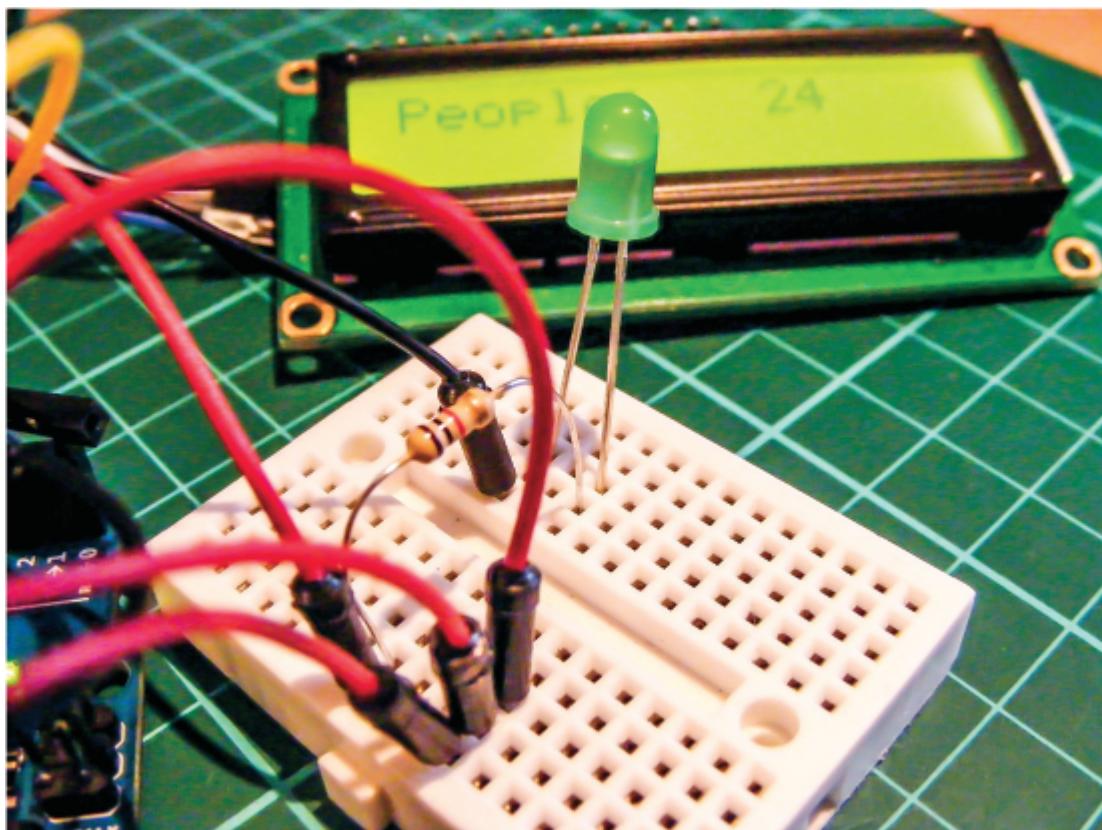
SERIAL LCD SCREEN	ARDUINO
GND	GND
VCC	+5V
SDA	Pin A4 (SDA)
SCL	Pin A5 (SCL)

4. Insert the LED into the mini-breadboard so that the shorter, negative (GND) leg is to the left and the longer, positive (+5V) leg is to the right, as shown in the following table and in Figure 17-3.

Connect the 220-ohm resistor to the positive leg of the LED, making sure the other leg of the resistor straddles the break in the breadboard. Connect this other resistor leg to pin 13 on the Arduino. Connect the shorter leg of the LED to GND on the Arduino.

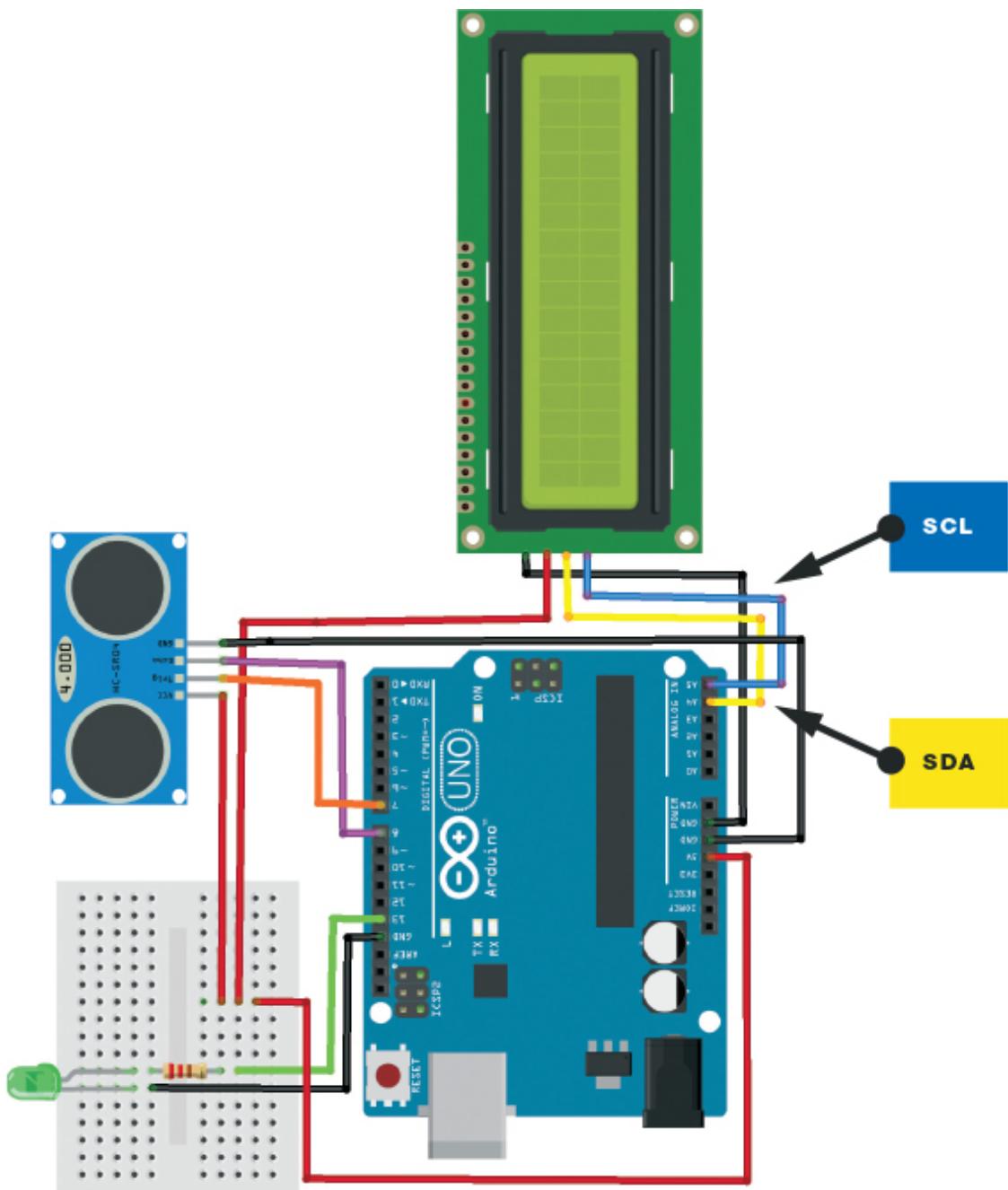
LED	ARDUINO
GND	GND
+5V	Pin 13 via 220-ohm resistor

**FIGURE 17-3:** We use the mini-breadboard to hold the LED and for multiple connections to the Arduino +5V.



5. Make sure your final circuit looks like Figure 17-4, and then upload the code in “The Sketch” on page 146 to the Arduino.

**FIGURE 17-4:** The circuit diagram for the ultrasonic people counter



## THE SKETCH

The sketch begins by calling on the `LiquidCrystal I2C`, `NewPing`, and `Wire` libraries to control the serial LCD screen and ultrasonic sensor. Next it defines the ultrasonic sensor Trig and Echo pins as Arduino pins 7 and 8, respectively. We set the maximum distance for the sensor to read to 200 centimeters (any reading beyond 200 centimeters is

ignored). Then we define pin 13 on the Arduino as the LED, which will be our counting indicator, and create variables to hold the distance and number of people. We create a `count` state so the Arduino can determine a valid record, and then we define the type of LCD screen. We initiate the LCD screen so that `People:` is printed to the screen, and set the LED pin as an output.

The loop section sends a ping from the sensor and if the ping that's returned is from a distance of more than 100 centimeters, the space in front of the sensor is considered empty and nothing is registered. If the distance recorded is less than 100 centimeters, it means something is within range in front of the sensor. In order for the `people:` variable to increment, someone has to move in front of the sensor, then out of the way. The sensor will keep counting every time a valid register is received, and the latest total is shown on the LCD screen.

The sensor could be placed to one side of an entrance, facing across the threshold, so as someone enters the sensor picks it up and registers a count. If the sensor is pointing toward a wall that's less than 100 centimeters away, you'll need to change the following line of code to a distance less than the distance to the wall; otherwise, the sensor will record a count every time the wall is detected.

```
-----  
if (distance < 100 && distance != 0 && !count)  
-----
```

Here is the full code:

```
-----  
#include <LiquidCrystal_I2C.h> // Call on the libraries  
#include <NewPing.h>  
#include <Wire.h>  
  
#define TRIGGER_PIN 7 // Ultrasonic sensor trig to Arduino pin 7  
#define ECHO_PIN 8 // Ultrasonic sensor echo to Arduino pin 8  
#define MAX_DISTANCE 200  
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);  
  
int LEDPin = 13; // Set LED to pin 13  
int distance; // Variable for distance  
int people = 0; // Variable for number of people  
boolean count = false; // State for counting  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
-----
```

```

void setup() { // Run once to set up the LCD screen and LED
  lcd.begin();
  lcd.backlight();
  pinMode(LEDPin, OUTPUT); // Set the LED as an output
  lcd.print("People:");
}

void loop() { // This loops forever to check for number of people
  delay(50);
  distance = sonar.ping_cm(); // Ping every 50 milliseconds
  // If more than 100 cm away, don't count
  if (distance > 100 && count) {
    count = false;
    digitalWrite(LEDPin, LOW);
  }
  // If less than 100 cm away, count 1
  if (distance < 100 && distance != 0 && !count) {
    count = true;
    people++;
    digitalWrite(LEDPin, HIGH);
    lcd.setCursor(10, 0);
    lcd.print(people); // Print number of people to LCD screen
  }
}

```

---

## TROUBLESHOOTING

**Q.** *The code compiles, but nothing shows on the screen.*

- Double-check that the SDA and SCL pins are connected to the correct Arduino pins.
- If the LCD screen is lit up but nothing shows, carefully turn the small potentiometer at the back of the module to change the contrast until the letters appear.

**Q.** *The sensor does not register a count or the LED does not light when expected.*

- Make sure that the ultrasonic sensor trigger pin is connected to Arduino pin 7 and the Echo pin to Arduino pin 8, and that power is connected to GND and +5V.
- If a count is registered and the LED does not light, recheck that the short leg of the LED is connected to GND and the long leg to +5V.

The resistor should straddle the break in the breadboard and be connected to the LED's long leg on one side and Arduino pin 13 on the other.

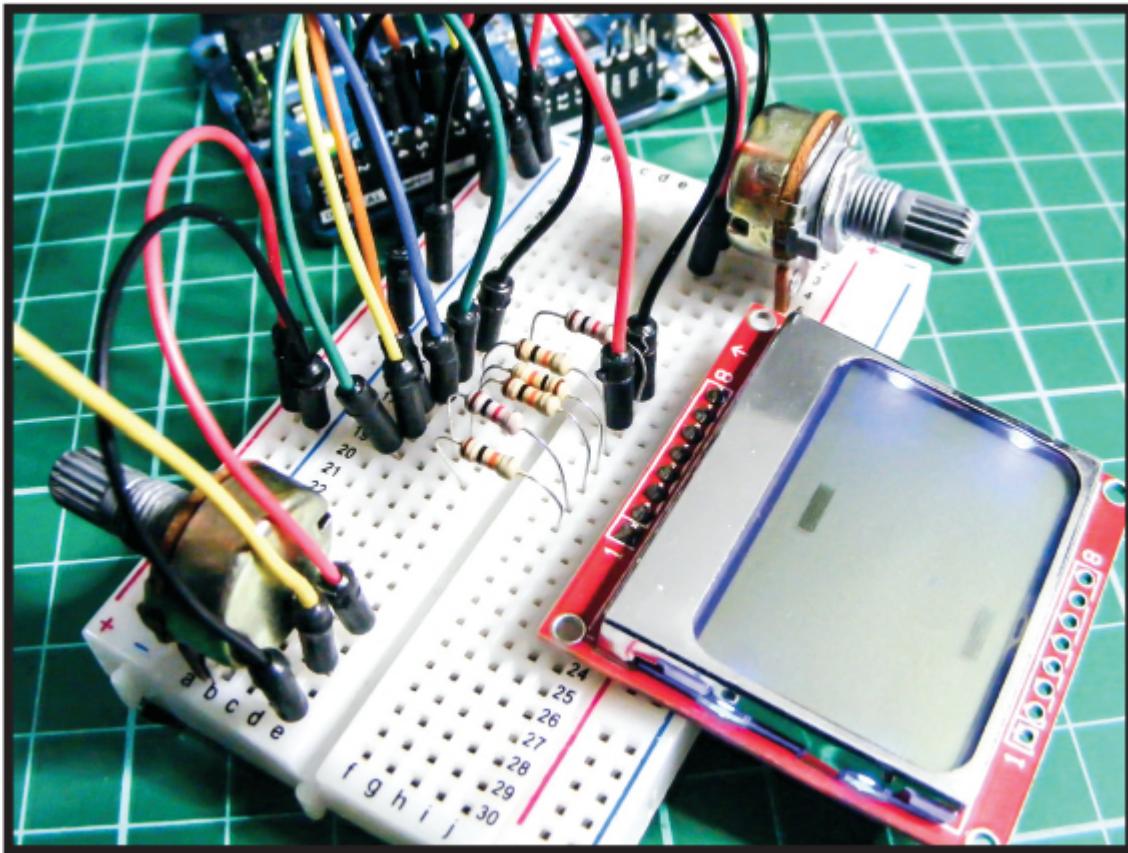
- Remember that the positioning of the sensor is important. If the distance to a fixed object (like a wall) is less than the distance in the sketch, the count will be incorrect.
- Your device may have a different address than the one we've used here. To check the address of your device, use the I2C scanner sketch available on the Arduino website (<http://playground.arduino.cc/Main/I2cScanner>). Run the sketch with your device attached to the Arduino and open the IDE Serial Monitor, and you should see the address of your device. Update the following line in this sketch with the address shown:

```
-----  
LiquidCrystal_I2C lcd(0x27,16,2);  
-----
```

18

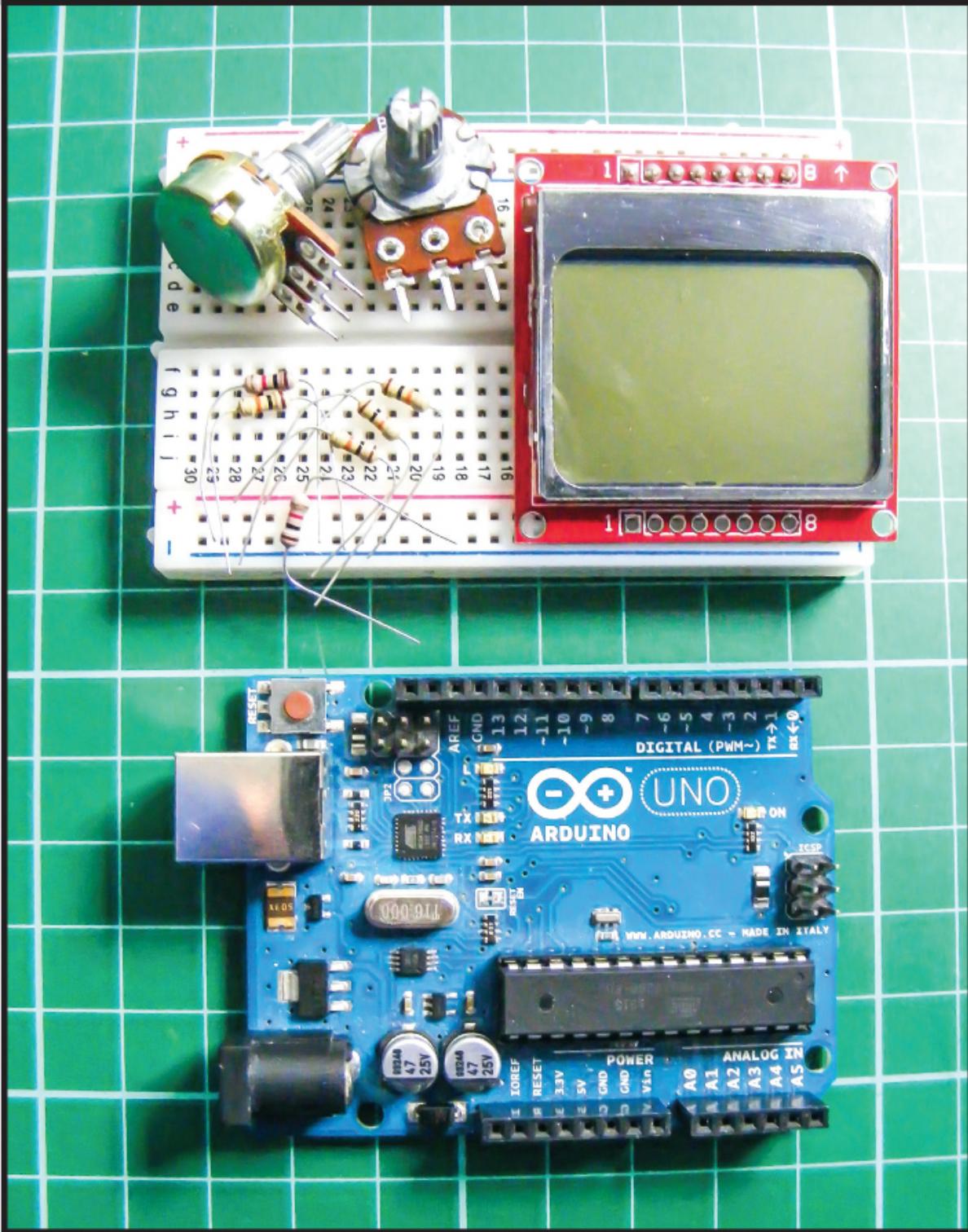
## Nokia 5110 LCD Screen Pong Game

This project shows you how to connect a Nokia 5110 LCD screen to your Arduino to recreate a *Pong*-style arcade game.



COST: \$\$

TIME: 30 MINUTES



## PARTS REQUIRED

**Arduino board**

**Breadboard**

**Jumper wires**

**Nokia 5110 LCD screen**

**4 10k-ohm resistors**

**2 1k-ohm resistors**

**2 50k-ohm potentiometers**

## HOW IT WORKS

Nokia 5110 LCD screens were used for all Nokia phones a few years back, so you should find plenty available online. We'll wire one up to the Arduino and create a simple *Pong*-style game by adding some potentiometers as controllers.

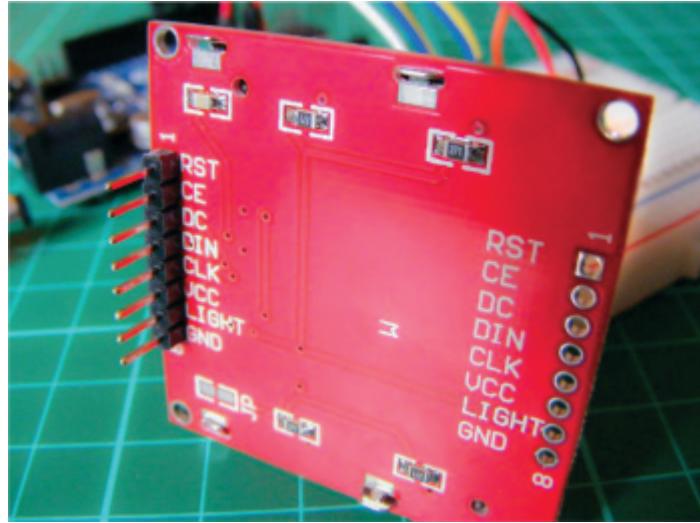
### NOTE

*See Project 13 for instructions on soldering header pins, and see the primer for general soldering instructions if you haven't soldered before.*

The screen is 84×48 pixels, which, with spaces between the characters so they aren't touching, gives us a 12×6-character screen. The screen works in the same way as the LCD screen in Project 13: by sending current through the liquid crystal from the Arduino to make certain pixels opaque and form letters or images.

Most screens come with the header pins separate for ease of transport, so you may need to solder them in place if you want to plug the screen into a breadboard. You'll need to solder a strip of eight header pins into the row of holes on one side of the screen, as you can see in Figure 18-1.

**FIGURE 18-1:** The reverse of the Nokia 5110 LCD screen showing the pin connections

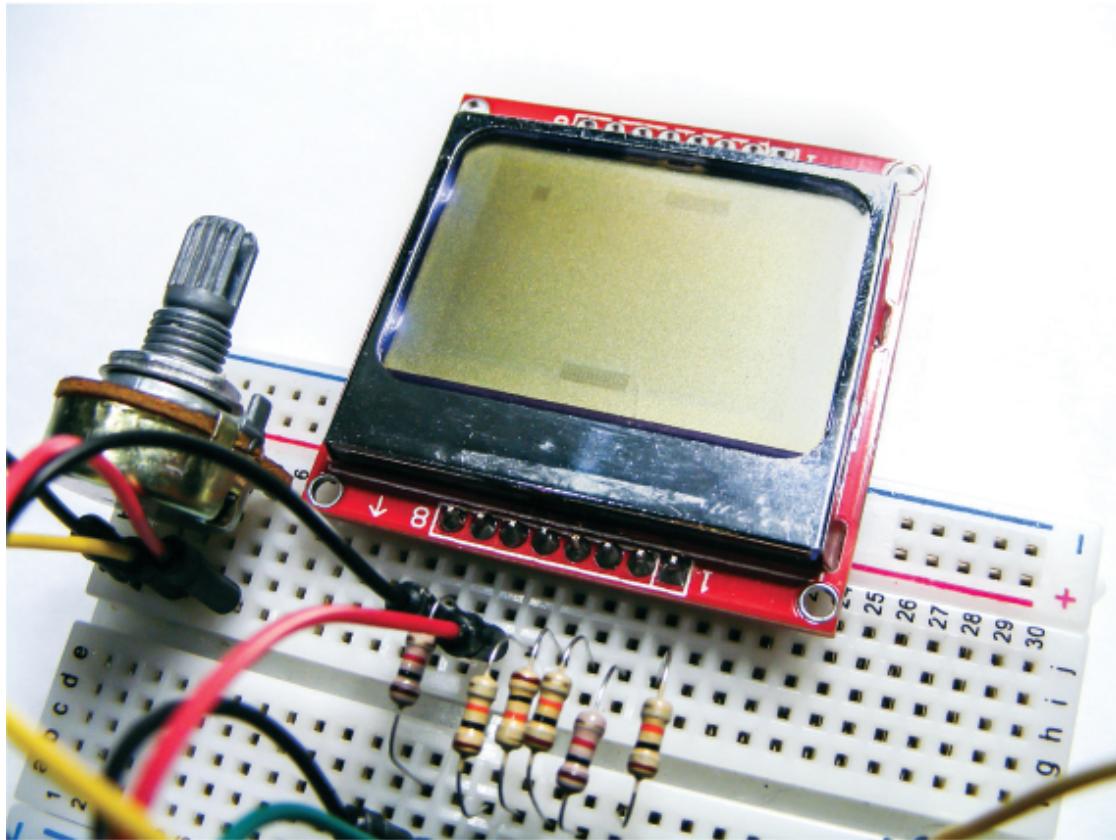


This project connects to +3.3V on the Arduino, rather than +5V.

## THE BUILD

1. Insert the Nokia 5110 screen into the breadboard.
2. The Nokia screen has eight pins. Insert a 10k-ohm resistor for Nokia pins 1, 3, 4, and 5, making sure they straddle the center break. Insert a 1k-ohm resistor for Nokia pins 2 and 7, as shown in Figure 18-2.

**FIGURE 18-2:** Insert the resistors for the Nokia LCD screen as shown here.



### **WARNING**

*It's really important to use the +3.3V power from the Arduino for the Nokia 5110 screen and not +5V for this project; otherwise, you will damage the screen.*

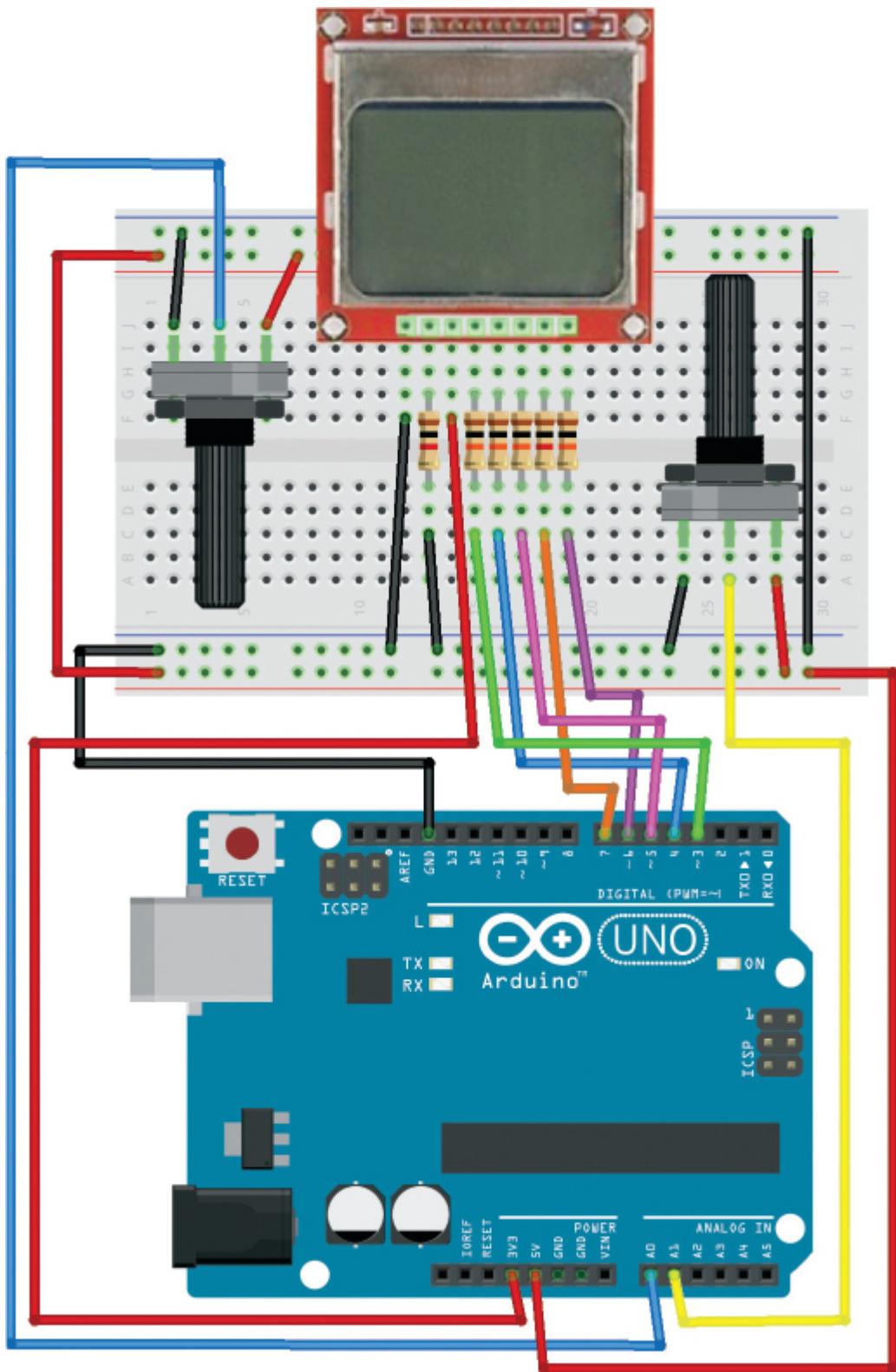
3. Use jumper wires to make the connections from the Nokia screen to Arduino pins 3–7 and to the breadboard power rails. Make sure to add the right value resistor to the correct pin, as shown in the following table. Some breakout boards may have the pins in different locations, so match the pin names on the Nokia screen with the Arduino pin.

NOKIA 5110 SCREEN	RESISTOR	ARDUINO
1 RESET	10k-ohm	Pin 6

NOKIA 5110 SCREEN	RESISTOR	ARDUINO
2 CE	1k-ohm	Pin 7
3 DC	10k-ohm	Pin 5
4 DIN	10k-ohm	Pin 4
5 CLK	10k-ohm	Pin 3
6 VCC	None	+3.3V
7 Light	1k-ohm	GND
8 GND	None	GND

4. Insert the potentiometers into the breadboard as shown in Figure 18-3. Connect the center pin of one potentiometer to Arduino A0 and the center pin of the other potentiometer to Arduino A1. Connect an outer pin of each potentiometer to the +5V rail of the breadboard and the other outer pins to the GND rail.
5. Connect the power rails of the breadboard to +5V and GND on the Arduino (this is for the potentiometers only).
6. Confirm that your setup matches Figure 18-3, and upload the code in “The Sketch” below.

**FIGURE 18-3:** The circuit diagram for the Nokia 5110 LCD Screen *Pong* Game



## THE SKETCH

The game starts with two bars on opposite sides of the screen and a ball bouncing between them. The object of the game is to use the potentiometers to move the bars like paddles, hitting the ball back and forth to stop it from going out of the play (beyond the screen perimeter). The ball bounces off the bars and gradually gets faster and faster. The game is over when the ball goes beyond the screen limit, at which point the display will invert and the game will start over again. Note that the ball can appear quite faint the faster it moves, due to the limitation of the screen graphics.

The first part of the sketch defines the pins connected to the Nokia 5110 LCD screen. It then defines the size of the screen, which is the area of our game that counts as in-play, and the size and starting positions of the bars and ball. The potentiometers read the analog signal from Arduino pins A0 and A1 and move their corresponding bars onscreen depending on how they're twisted. The calculations that follow determine whether the ball and the bar have met at certain coordinates. If they have, the ball bounces back; if they haven't, it means the bar has missed the ball, so the screen inverts and flashes to indicate the game is over.

---

// Arduino Pong by Onur Avun and reproduced with kind permission

```
#define PIN_SCE    7
#define PIN_RESET  6
#define PIN_DC     5
#define PIN_SDIN   4
#define PIN_SCLK   3
#define LCD_C      LOW
#define LCD_D      HIGH
#define LCD_X     84
#define LCD_Y     6

int barWidth = 16;
int barHeight = 4;
int ballPerimeter = 4;

unsigned int bar1X = 0;
unsigned int bar1Y = 0;
unsigned int bar2X = 0;
unsigned int bar2Y = LCD_Y * 8 - barHeight;
```

```
int ballX = 0;
int ballY = 0;

boolean isBallUp = false;
boolean isBallRight = true;
byte pixels[LCD_X][LCD_Y];
unsigned long lastRefreshTime;
const int refreshInterval = 150;

byte gameState = 1;
byte ballSpeed = 2;
byte player1WinCount = 0;
byte player2WinCount = 0;
byte hitCount = 0;
void setup() {
    LcdInitialise();
    restartGame();
}

void loop() {
    unsigned long now = millis();
    if (now - lastRefreshTime > refreshInterval) {
        update();
        refreshScreen();
        lastRefreshTime = now;
    }
}

void restartGame() {
    ballSpeed = 1;
    gameState = 1;
    ballX = random(0, 60);
    ballY = 20;
    isBallUp = false;
    isBallRight = true;
    hitCount = 0;
}

void refreshScreen() {
    if (gameState == 1) {
        for (int y = 0; y < LCD_Y; y++) {
            for (int x = 0; x < LCD_X; x++) {
                byte pixel = 0x00;
                int realY = y * 8;
                // Draw ball if in frame
                if (x >= ballX && x <= ballX + ballPerimeter - 1 && ballY + ballPerimeter > realY && ballY < realY + 8 ) {
                    byte ballMask = 0x00;
                    for (int i = 0; i < realY + 8 - ballY; i++) {
                        ballMask = ballMask >> 1;
                    }
                    pixels[x][y] = ballMask;
                } else {
                    pixels[x][y] = 0x00;
                }
            }
        }
    }
}
```

```

        if (i < ballPerimeter)
            ballMask = 0x80 | ballMask;
    }
    pixel = pixel | ballMask;
}

// Draw bars if in frame
if (x >= bar1X && x <= bar1X + barWidth -1 && bar1Y +
    barHeight > realY && bar1Y < realY + 8 ) {
    byte barMask = 0x00;
    for (int i = 0; i < realY + 8 - bar1Y; i++) {
        barMask = barMask >> 1;
        if (i < barHeight)
            barMask = 0x80 | barMask;
    }
    pixel = pixel | barMask;
}

if (x >= bar2X && x <= bar2X + barWidth -1 && bar2Y +
    barHeight > realY && bar2Y < realY + 8 ) {
    byte barMask = 0x00;
    for (int i = 0; i < realY + 8 - bar2Y; i++) {
        barMask = barMask >> 1;
        if (i < barHeight)
            barMask = 0x80 | barMask;
    }
    pixel = pixel | barMask;
}
    LcdWrite(LCD_D, pixel);
}
}
} else if (gameState == 2) {
}
}

void update() {
if (gameState == 1) {
    int barMargin = LCD_X - barWidth;
    int pot1 = analogRead(A0); // Read pots and set bar positions
    int pot2 = analogRead(A1);
    bar1X = pot1 / 2 * LCD_X / 512;
    bar2X = pot2 / 2 * LCD_X / 512;

    if (bar1X > barMargin) bar1X = barMargin;
    if (bar2X > barMargin) bar2X = barMargin;

    // Move the ball now
    if (isBallUp)
        ballY -= ballSpeed;
    else

```

```

    ballY += ballSpeed;
    if (isBallRight)
        ballX += ballSpeed;
    else
        ballX -= ballSpeed;
    // Check collisions
    if (ballX < 1) {
        isBallRight = true;
        ballX = 0;
    }
    else if (ballX > LCD_X - ballPerimeter - 1) {
        isBallRight = false;
        ballX = LCD_X - ballPerimeter;
    }
    if (ballY < barHeight) {
        if (ballX + ballPerimeter >= bar1X && ballX <= bar1X + barWidth) {
            // Ball bounces from bar1
            isBallUp = false;
            if (ballX + ballPerimeter / 2 < bar1X + barWidth / 2)
                isBallRight = false;
            else
                isBallRight = true;
            ballY = barHeight;
            if (++hitCount % 10 == 0 && ballSpeed < 5)
                ballSpeed++;
        } else { // Player 2 wins
            gameState = 2;
            player2WinCount++;
        }
    }
    if (ballY + ballPerimeter > LCD_Y * 8 - barHeight) {
        if (ballX + ballPerimeter >= bar2X && ballX <= bar2X + barWidth) {
            // Ball bounces from bar2
            isBallUp = true;
            if (ballX + ballPerimeter / 2 < bar2X + barWidth / 2)
                isBallRight = false;
            else
                isBallRight = true;
            ballY = LCD_Y * 8 - barHeight - ballPerimeter;
            if (++hitCount % 10 == 0 && ballSpeed < 5)
                ballSpeed++;
        } else { // Player 1 wins
            gameState = 2;
            player1WinCount++;
        }
    }
} else if (gameState == 2) {
    for (int i = 0; i < 4; i++) {
        LcdWrite(LCD_C, 0x0D); // LCD in inverse mode.
        delay(300);
}

```

```

        LcdWrite(LCD_C, 0x0C); // LCD in inverse mode.
        delay(300);
    }
    restartGame();
}
}

void LcdInitialise(void) {
    pinMode(PIN_SCE, OUTPUT);
    pinMode(PIN_RESET, OUTPUT);
    pinMode(PIN_DC, OUTPUT);
    pinMode(PIN_SDIN, OUTPUT);
    pinMode(PIN_SCLK, OUTPUT);
    delay(200);
    digitalWrite(PIN_RESET, LOW);
    delay(500);
    digitalWrite(PIN_RESET, HIGH);
    LcdWrite(LCD_C, 0x21 ); // LCD Extended Commands
    LcdWrite(LCD_C, 0xB1 ); // Set LCD Vop (Contrast)
    LcdWrite(LCD_C, 0x04 ); // Set Temp coefficient. //0x04
    LcdWrite(LCD_C, 0x14 ); // LCD bias mode 1:48. //0x13
    LcdWrite(LCD_C, 0x0C ); // LCD in normal mode.
    LcdWrite(LCD_C, 0x20 );
    LcdWrite(LCD_C, 0x80 ); // Select X Address 0 of the LCD ram
    LcdWrite(LCD_C, 0x40 ); // Select Y Address 0 of the LCD ram
    LcdWrite(LCD_C, 0x0C );
}

void LcdWrite(byte dc, byte data) {
    digitalWrite(PIN_DC, dc);
    digitalWrite(PIN_SCE, LOW);
    shiftOut(PIN_SDIN, PIN_SCLK, MSBFIRST, data);
    digitalWrite(PIN_SCE, HIGH);
}
-----
```

## TROUBLESHOOTING

**Q.** *Nothing is displayed on the LCD screen.*

- Make sure you've connected power to the LCD screen direct to the Arduino +3.3V power pin and the connections match the tables in this chapter.
- Make sure your resistors line up with the correct LCD pins, as well as the wires to the Arduino pins.

- If the backlight of the LCD screen is lit but there is no image, you may have some wires mixed up; they need to match the circuit in Figure 18-3 exactly.

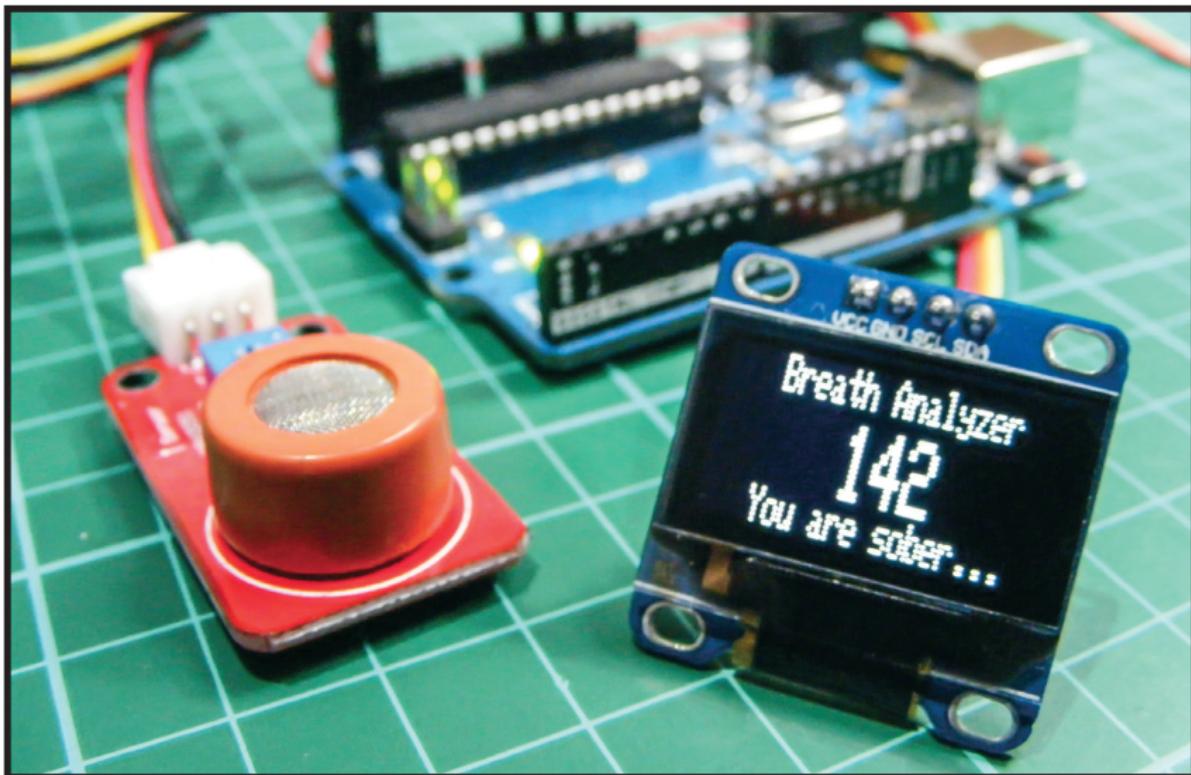
**Q.** *When the player turns the potentiometer, one or both of the bars do not move.*

- Make sure the potentiometers are connected firmly in the breadboard and that the wires connecting to the power rails and Arduino line up with the potentiometer pins.
- Remember that the potentiometers require +5V power and GND from the Arduino. These pins should be hooked up to the breadboard power rails via jumper wires.
- Make sure you also use jumper wires to connect the corresponding power rails on either side of the breadboard to each other.

19

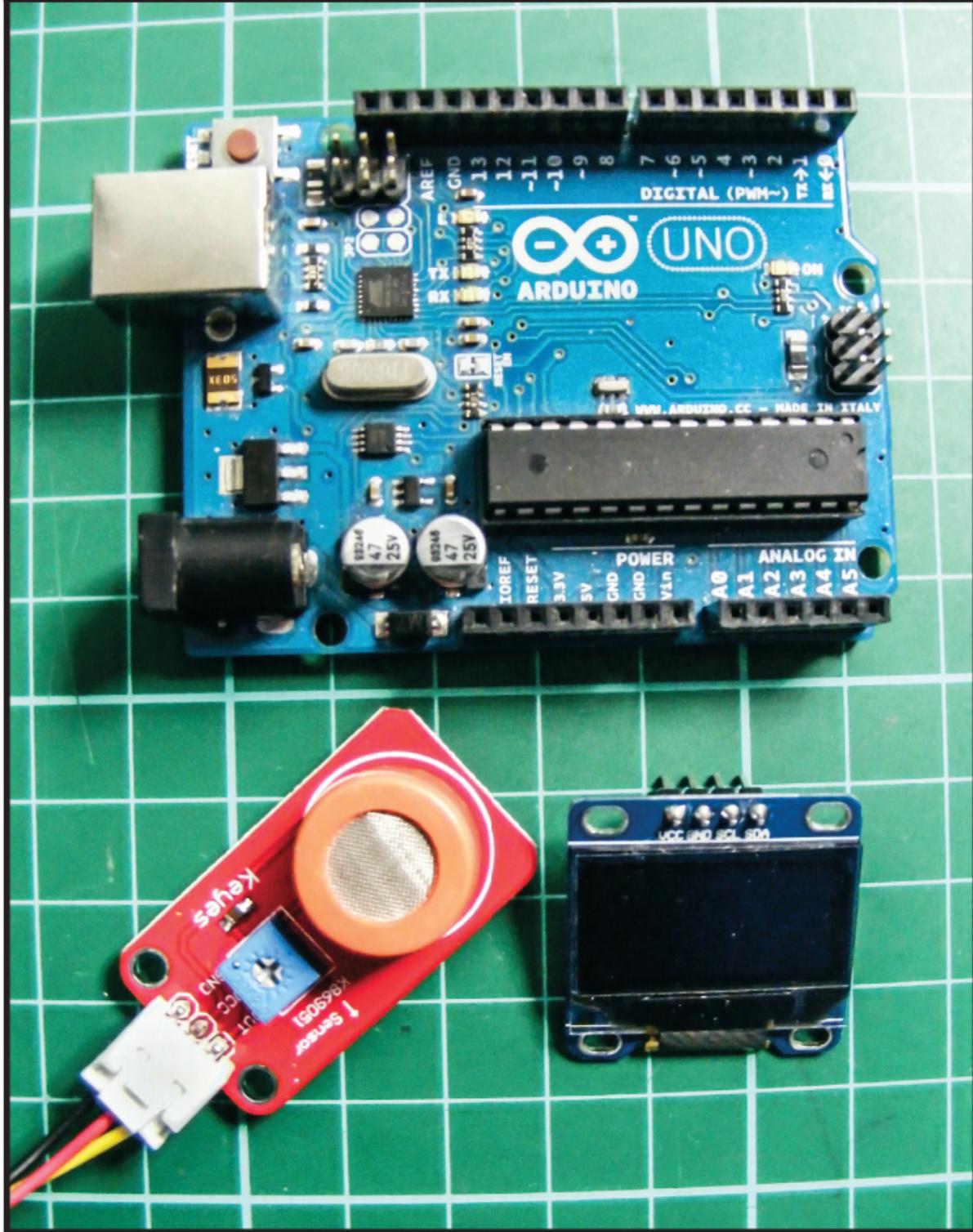
## OLED Breathalyzer

In this project we'll use the MQ3 alcohol sensor and an OLED LCD screen to make a mini-breathalyzer.



COST: \$\$\$

TIME: 30 MINUTES



## PARTS REQUIRED

**Arduino board**

**Female-to-male jumper wires**

**Keyes MQ3 alcohol sensor module**

**OLED monochrome screen (128x64)**

## **LIBRARIES REQUIRED**

**SPI**

**Wire**

**Adafruit\_GFX**

**Adafruit\_SSD1306**

## **HOW IT WORKS**

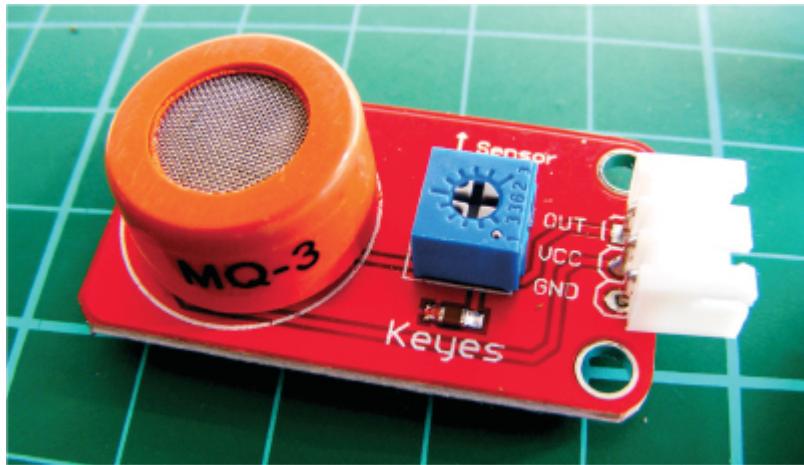
The MQ3 is part of a family of gas sensors that also includes the MQ2, sensitive to methane, butane, and smoke; the MQ4, sensitive to compressed natural gas; the MQ6, sensitive to butane and LPG gas; and the MQ7, sensitive to carbon monoxide. The MQ3 is sensitive to alcohol and ethanol, so it's the one we'll use in our breathalyzer.

### **DISCLAIMER**

*This project is for amusement only and should not be used to accurately determine anyone's alcohol intake.*

The Keyes MQ3 module (Figure 19-1) has the wiring we need for this project, including a built-in potentiometer and resistor. The three pin connections on the module are OUT, VCC, and GND.

**FIGURE 19-1:** The Keyes MQ3 alcohol sensor module. As with most MQ sensors, the module has a small heater inside with an electrochemical sensor used to measure the gas level. The value of the reading is sent to the OUT pin, which is then read by an analog pin on our Arduino.



To display the sensor readings, we'll use an OLED screen (Figure 19-2). OLED, which stands for *organic light-emitting diode*, is a light-emitting technology composed of a thin, multilayered organic film placed between an anode and cathode. When voltage is applied, an image is created through *electroluminescence*, which means the screen does not require a backlight. Our OLED screen is an I2C 128×64 monochrome version, meaning we can control it using only two pins to the Arduino and it measures 128 pixels by 64 pixels. This screen uses the same communication protocol as our serial LCD in Project 16 and is explained further there.

**FIGURE 19-2:** 128×64 OLED monochrome screen. When the MQ3 reads the value, the Arduino sends a message to the OLED screen indicating whether or not alcohol has been detected.



## WARNING

*As mentioned earlier, the MQ3 uses an internal heater as part of the sensor process. This heater can reach 120–140 degrees when powered, so take care when handling it when it's in use.*

## THE BUILD

1. Before you use the sensor for the first time, you need to “burn it in.” This process, which simply involves powering it up for a few hours to heat the mechanism inside, improves the sensor’s accuracy. To do this, connect the VCC and GND pins of the sensor to +5V and GND on your Arduino, respectively, using female-to-male jumper wires. When you power the Arduino, it will send the correct voltage to the MQ3. Leave it powered for two to three hours—you may notice a burning smell and the sensor will get hot, but this is normal.
2. Once the sensor is burned in, disconnect the power to the Arduino and connect the sensor to the Arduino using the female-to-male jumper wires, with the MQ3’s OUT pin connected to Arduino pin

A0, and the power and GND still connected as before (see the following table).

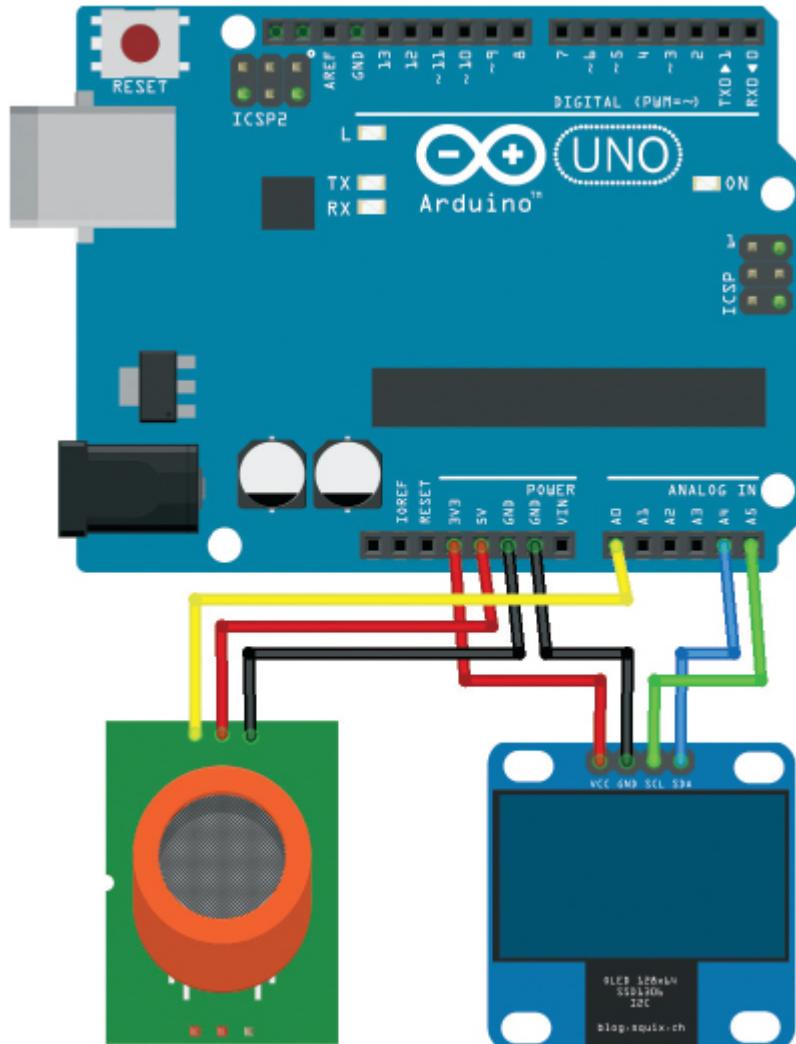
MQ3 ALCOHOL SENSOR	ARDUINO
OUT	Pin A0
VCC	+5V
GND	GND

3. Next, connect the OLED screen to the Arduino as shown in the following table, with SCL connected to pin A5, SDA to pin A4, VCC to +3.3V, and GND to GND.

OLED SCREEN	ARDUINO
SCL	Pin A5
SDA	Pin A4
VCC	+3.3V
GND	GND

4. This project requires a number of libraries to work correctly; the SPI and Wire libraries are built into the Arduino IDE, but we also need the Adafruit\_GFX and Adafruit\_SSD1306 libraries to control the OLED screen. Both are available from <https://www.nostarch.com/arduinohandbook2/>. Refer to the primer if you need a reminder on how to add libraries to the IDE.
5. Check that your setup matches the circuit diagram in Figure 19-3, and upload the code in “The Sketch” below.

**FIGURE 19-3:** The circuit diagram for the OLED breathalyzer



6. The heater inside the MQ3 sensor needs to heat up for about 4 minutes before it can operate accurately. The sketch has a timer so that when you power it up for the first time, the values won't appear onscreen until the required time has passed. The "Warming up" text will display with a small countdown bar until the sensor is ready.

## THE SKETCH

The sketch starts by calling on the SPI, Wire, Adafruit\_GFX, and Adafruit\_SSD1306 libraries to control communication and the OLED screen. We assign a time for the warm-up session (4 minutes) and set the analog pin as Arduino A0.

Next we set up the OLED screen. The Arduino sends different messages to the screen depending on the value read from the analog pin. For instance, if the sensor reading is above 200, the Arduino will ask you if you've had a beer. If the reading is below this value, the Arduino will say you're sober. The minimum level of alcohol the MQ3 will read is about 180. For anything over 450, the breathalyzer will let you know you're drunk!

The sketch loops every second to read the analog sensor. To use the breathalyzer, wait for the sensor to heat up for 4 minutes, then gently breathe onto the sensor. Try not to get the sensor wet or expose it to a smoky environment, as this will affect the reading.

---

```
// Re-created with kind permission from Nick Koumaris educ8s.tv

// Call the SPI, Wire, Adafruit_GFX, and Adafruit_SSD1306 libraries
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4 // Define the OLED screen
int TIME_UNTIL_WARMUP = 4; // Time for the warm-up delay in minutes
unsigned long time;
int analogPin = 0; // Set analog pin as A0
int val = 0; // Set a value to read from the analog pin
Adafruit_SSD1306 display(OLED_RESET);

void setup() { // Set up the OLED screen
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.clearDisplay();
}

void loop() { // Take the reading and show it onscreen
    delay(100);
    val = readAlcohol();
    printTitle();
    printWarming();
    time = millis() / 1000;
    time /= 60;
    if (time <= TIME_UNTIL_WARMUP) { // If warm-up is less than 4 mins
        time = map(time, 0, TIME_UNTIL_WARMUP, 0, 100); // Show countdown
        display.drawRect(10, 50, 110, 10, WHITE); // Empty bar
        display.fillRect(10, 50, time, 10, WHITE);
    } else { // When warm-up time has passed
        // the value and message are printed on the screen
        printTitle();
    }
}
```

```

        printAlcohol(val);
        printAlcoholLevel(val);
    }
    display.display();
}

void printTitle() { // Position and text of title on the screen
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(22, 0);
    display.println("Breath Analyzer");
}
void printWarming() { // Warm-up message
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(30, 24);
    display.println("Warming up");
}
void printAlcohol(int value) { // Print alcohol value to screen
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(50, 10);
    display.println(val);
}

void printAlcoholLevel(int value) { // Print message to screen
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(20, 25);
    if (value < 200) { // If value read is less than 200, you are sober
        display.println("You are sober...");
    }
    if (value >= 200 && value < 280) {
        display.println("You had a beer?");
    }
    if (value >= 280 && value < 350) {
        display.println("Two or more beers.");
    }
    if (value >= 350 && value < 450) {
        display.println("I smell VODKA!");
    }
    if (value > 450) {
        display.println("You are drunk!");
    }
}

// Finds average by summing three readings and
// dividing by 3 for better accuracy

```

```
int readAlcohol() {  
    int val = 0;  
    int val1;  
    int val2;  
    int val3;  
    display.clearDisplay();  
    val1 = analogRead(analogPin);  
    delay(10);  
    val2 = analogRead(analogPin);  
    delay(10);  
    val3 = analogRead(analogPin);  
    val = (val1 + val2 + val3) / 3;  
    return val;  
}
```

---

## TROUBLESHOOTING

**Q.** *The display is not showing readings correctly.*

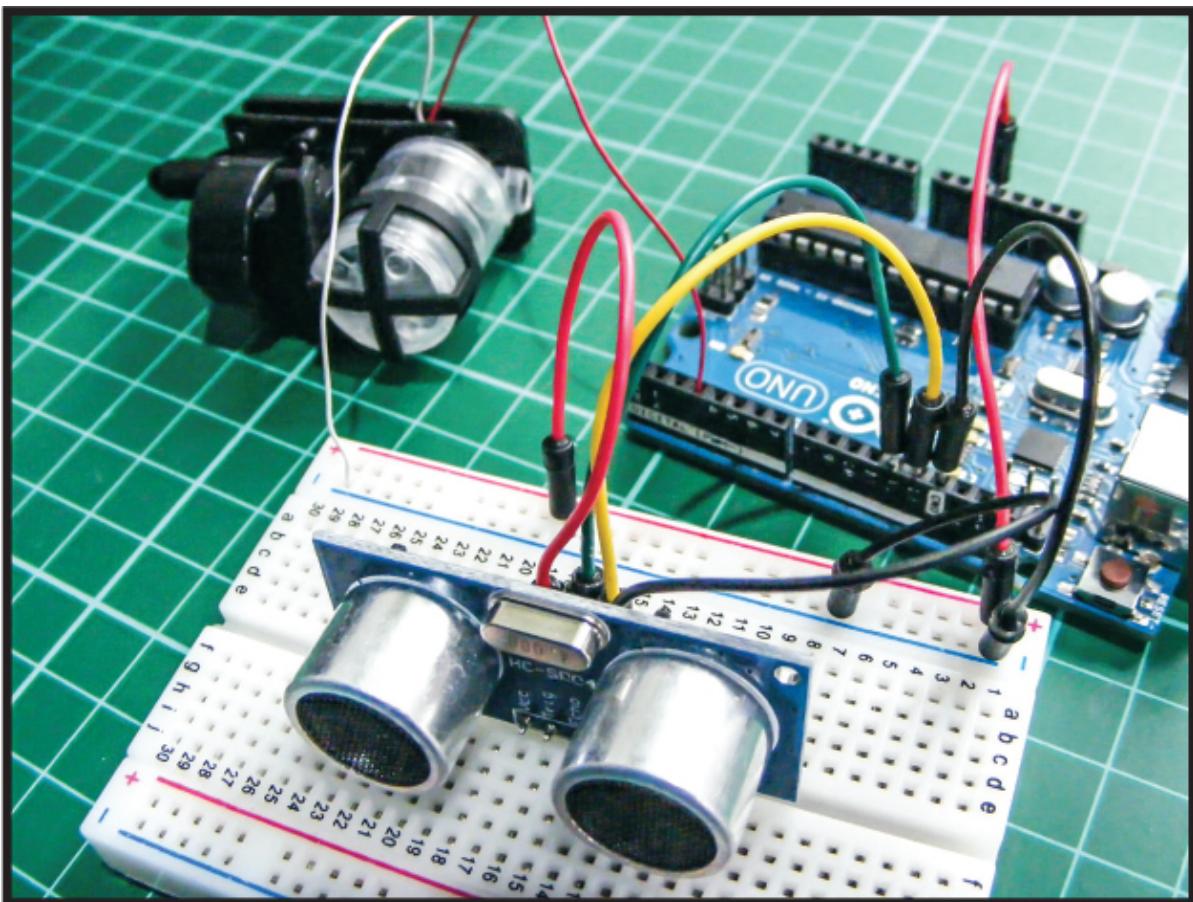
- Recheck that your wiring matches the diagram in Figure 19-3.
- If all your wiring is in the correct place, make sure you've carried out the earlier step to burn the sensor in by leaving it powered for a few hours.
- To check whether your components are faulty, temporarily swap a potentiometer in for the sensor. Connect the center pin of the potentiometer to A0 and add power to either side. If the potentiometer is working okay, it means your sensor is probably faulty, so replace your sensor—they are very inexpensive.

# Security

20

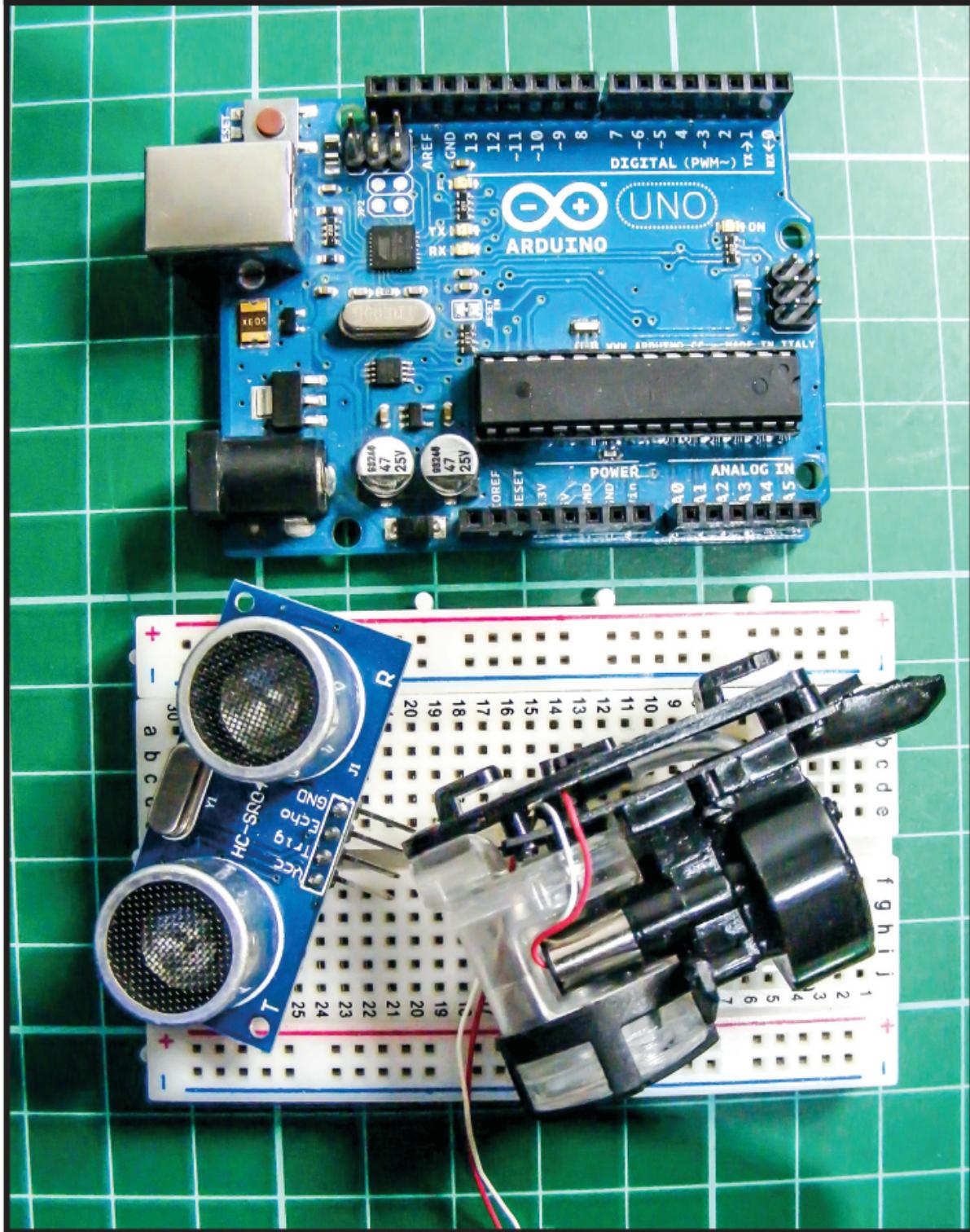
## Ultrasonic Soaker

In this project we'll use an ultrasonic sensor to trigger a toy water pistol. You could set this up to soak unsuspecting victims when they venture into forbidden territory!



**COST: \$\$**

**TIME: 30 MINUTES**



## **PARTS REQUIRED**

**Arduino board**

**Breadboard**

**Jumper wires**

**HC-SR04 ultrasonic sensor**

**WLtoys V959-18 Water Jet Pistol**

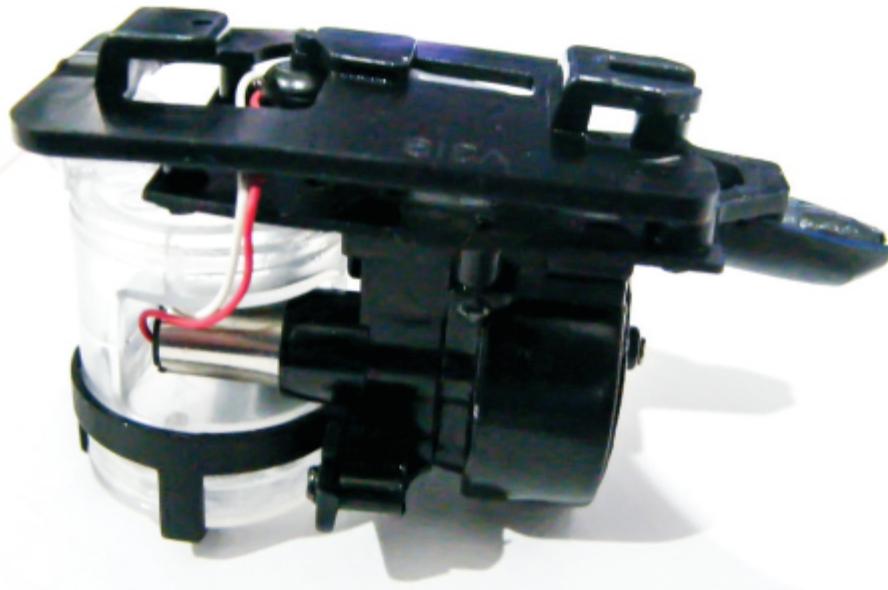
## **LIBRARY REQUIRED**

**NewPing**

## **HOW IT WORKS**

For our soaker, we'll use the WLtoys V959-18 Water Jet Pistol (Figure 20-1) attachment for RC helicopters, which is inexpensive and widely available online. The pistol has a small reservoir to hold water and a mini-pump that shoots the water through a nozzle at the front. The pistol has only two wires: red is positive power and white is ground. It requires only a little current, which lets us trigger the pump using the current supplied by the Arduino.

**FIGURE 20-1:** The WLtoys V959-18 Water Jet Pistol



**NOTE**

*Remember that water and electricity do not mix well, so try to keep your Arduino away from the water jet to minimize the chance of water short-circuiting your Arduino board.*

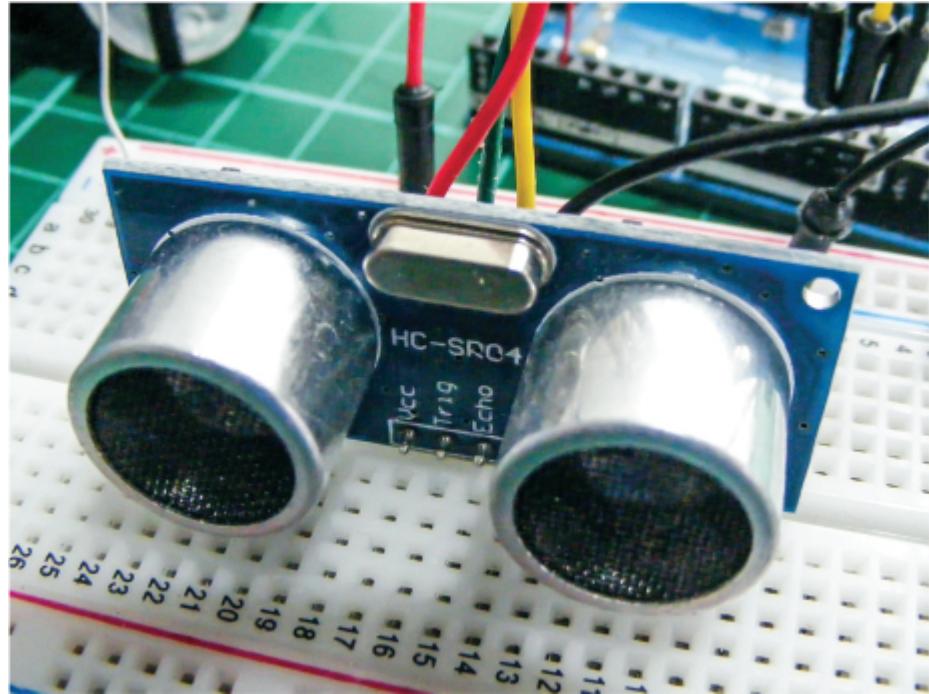
As we discussed in Project 13, the ultrasonic sensor sends out a burst of ultrasound and listens for the echo that bounces off an object to determine its distance. Here, the ultrasonic sensor looks for a bounceback that indicates an object is less than 15 centimeters away, in which case the Arduino sends power to the soaker to squirt water on our victims.

## THE BUILD

1. Add the ultrasonic sensor module (Figure 20-2) to your breadboard and connect VCC to +5V, Trig to Arduino pin 12, Echo to Arduino pin 13, and GND to GND, as shown in the following table.

ULTRASONIC SENSOR	ARDUINO
VCC	+5V
Trig	Pin 12
Echo	Pin 13
GND	GND

**FIGURE 20-2:** The ultrasonic sensor



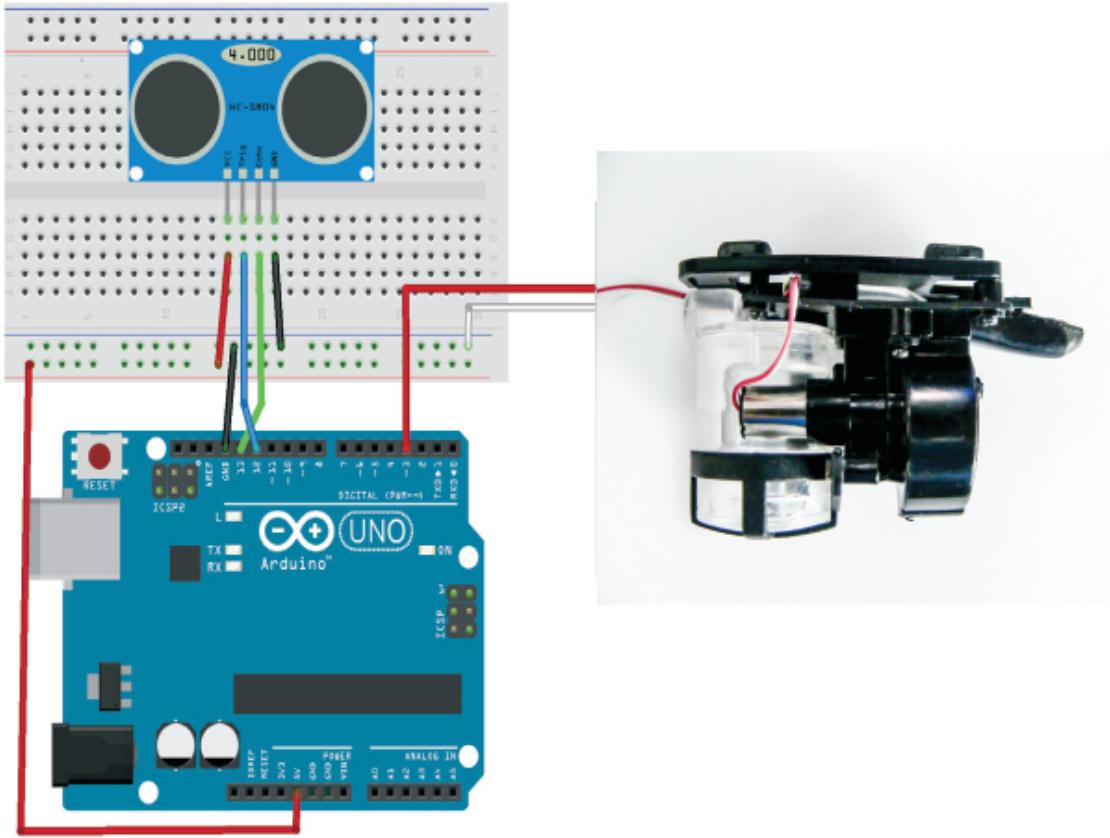
2. Connect the pistol's red power wire to Arduino pin 3 and its white wire to Arduino GND via the breadboard power rail. Connect the power rails of the breadboard to Arduino power. The pistol comes with a small pipette to help you fill the reservoir. Figure 20-3 shows where to fill the reservoir.

**FIGURE 20-3:** Use the pipette provided to fill the reservoir shown with water through the opening at the top.



3. Once you've confirmed that your setup matches the circuit diagram in Figure 20-4, upload the code in "The Sketch" on page 172 to your Arduino, making sure to add the NewPing library to the Arduino IDE.

**FIGURE 20-4:** The circuit diagram for the ultrasonic soaker



## THE SKETCH

Before entering the sketch, download the NewPing library from <http://www.nostarch.com/arduinohandbook2/>. The sketch calls on the NewPing library and defines the Arduino pin connections. Arduino pin 12 is connected to the sensor's trigger pin and sends out an ultrasonic signal, and Arduino pin 13, connected to the sensor's Echo pin, receives the returning signal. The Arduino converts the time between sending and receiving the signal into distance. The soaker is attached to Arduino pin 3, and a loop checks the distance to the detected object. If the distance is less than 15 centimeters, power is sent to pin 3 and the soaker shoots water at your unsuspecting friends!

---

```
#include <NewPing.h> // This calls the NewPing library
#define trigPin 12 // Trig pin attached to Arduino 12
#define echoPin 13 // Echo pin attached to Arduino 13
#define soakerPin 3
#define MAX_DISTANCE 500
```

```

NewPing sonar(trigPin, echoPin, MAX_DISTANCE);

void setup() {
  Serial.begin(9600);
  pinMode(soakerPin, OUTPUT);
}
void loop() {
  int distance;
  distance = sonar.ping_cm();
  Serial.print(distance);
  Serial.println(" cm");

  if (distance <= 15) { // If distance is less than 15
    digitalWrite(soakerPin, HIGH); // Soaker shoots water
    delay(250);
    digitalWrite(soakerPin, LOW); // Short pulse of water
  }
  else {
    digitalWrite(soakerPin, LOW); // Soaker will remain off
  }
  delay(500);
}

```

---

## TROUBLESHOOTING

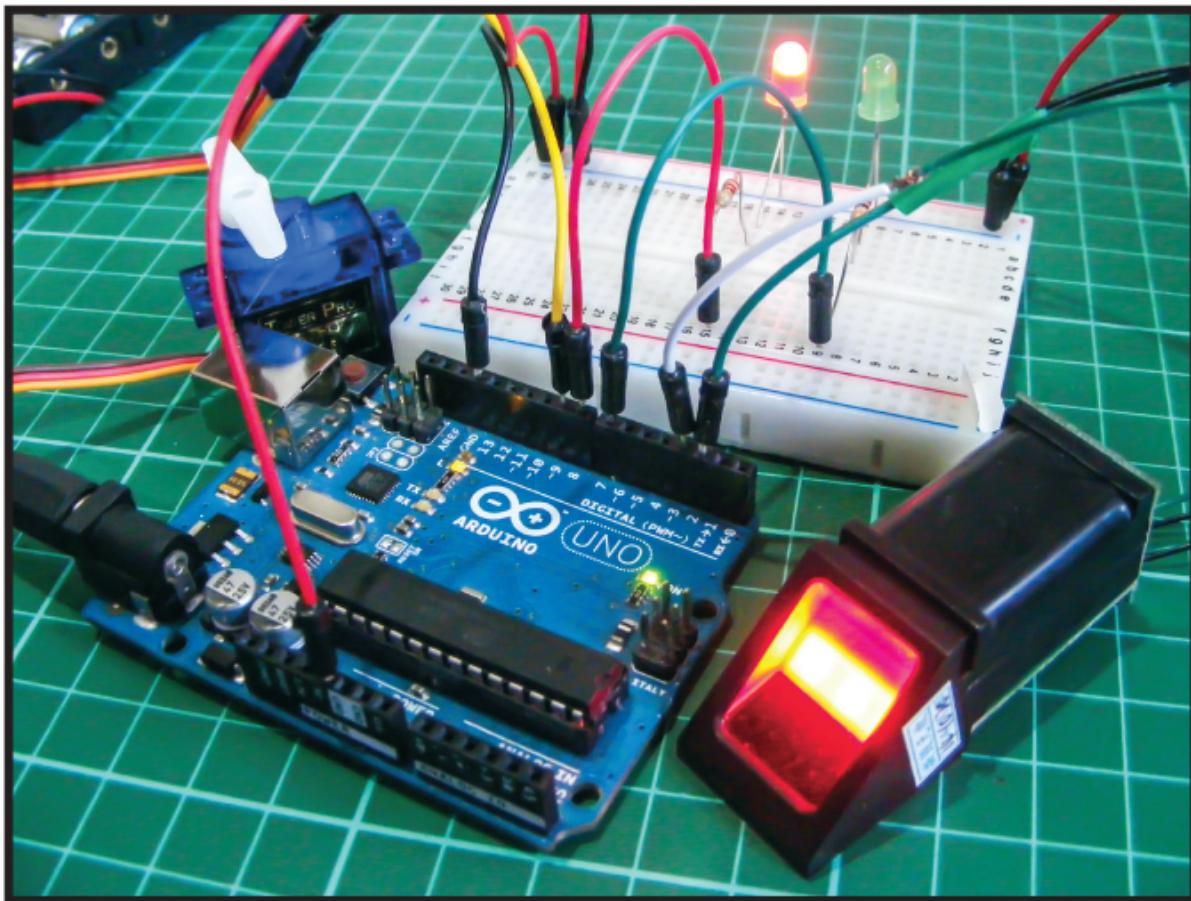
**Q.** *The ultrasonic soaker does not shoot.*

- Make sure the connections match the setup for the ultrasonic sensor by rechecking this chapter's tables and the circuit diagram in Figure 20-4.
- Remember that the water will shoot only when the sensor detects someone or something in front of it.
- Make sure you have added power to the breadboard power rails.
- Check that the water jet is working correctly by disconnecting it from the circuit and then connecting the wires to +5V and GND on the Arduino directly. You should hear the buzz of the pump motor; if you don't, your component may be faulty.

21

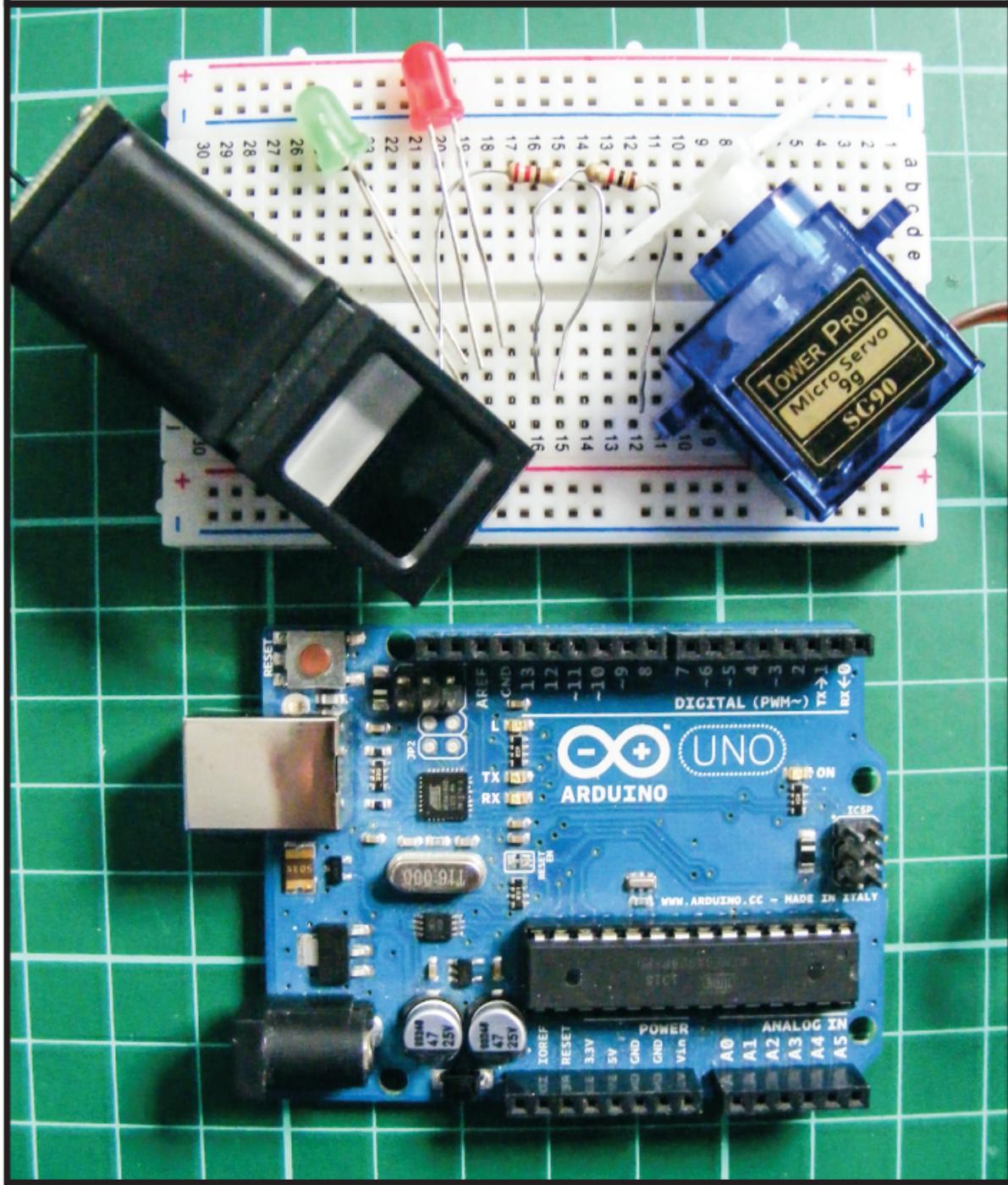
## Fingerprint Scanner

In this project we'll use a fingerprint sensor, a servomotor, and some LEDs to create a cool biometric entry system.



**COST: \$\$\$\$\$**

**TIME: 1 HOUR**



## PARTS REQUIRED

Arduino board

Breadboard

**Jumper wires**  
**Red LED**  
**Green LED**  
**2 220-ohm resistors**  
**Tower Pro SG90 9g servomotor**  
**Optical fingerprint sensor (ZFM-20 Series)**

## LIBRARIES REQUIRED

**Adafruit\_Fingerprint**  
**Servo**  
**SoftwareSerial**

### NOTE

*The software we're using in this project operates only on Windows.*

## HOW IT WORKS

Biometric identification is used to identify a person from a specific biological characteristic that remains the same even over a long period of time, such as a fingerprint or iris pattern. Since fingerprints are unique to each person, they're often used to help identify individuals for purposes like criminal investigations and security authentication. In this project, we'll use a fingerprint sensor to read a fingerprint and, if it matches a print on record with the right security clearance, allow access by moving a servomotor.

The sensor we'll use is the ZFM-20 Series Fingerprint Identification Module (see Figure 21-1) but will generally be referred to as an *optical fingerprint sensor module*. The sensor takes a photograph of a fingerprint, adds it to the module's database, and then checks the scanned fingerprint for a match. It can hold up to 162 fingerprints. The sensor is available online and from retailers such as Adafruit, which has also

created a specific Arduino library for the module that we'll use in the sketch.

**FIGURE 21-1:** The ZFM-20 Series Fingerprint Identification Module is an optical fingerprint sensor.

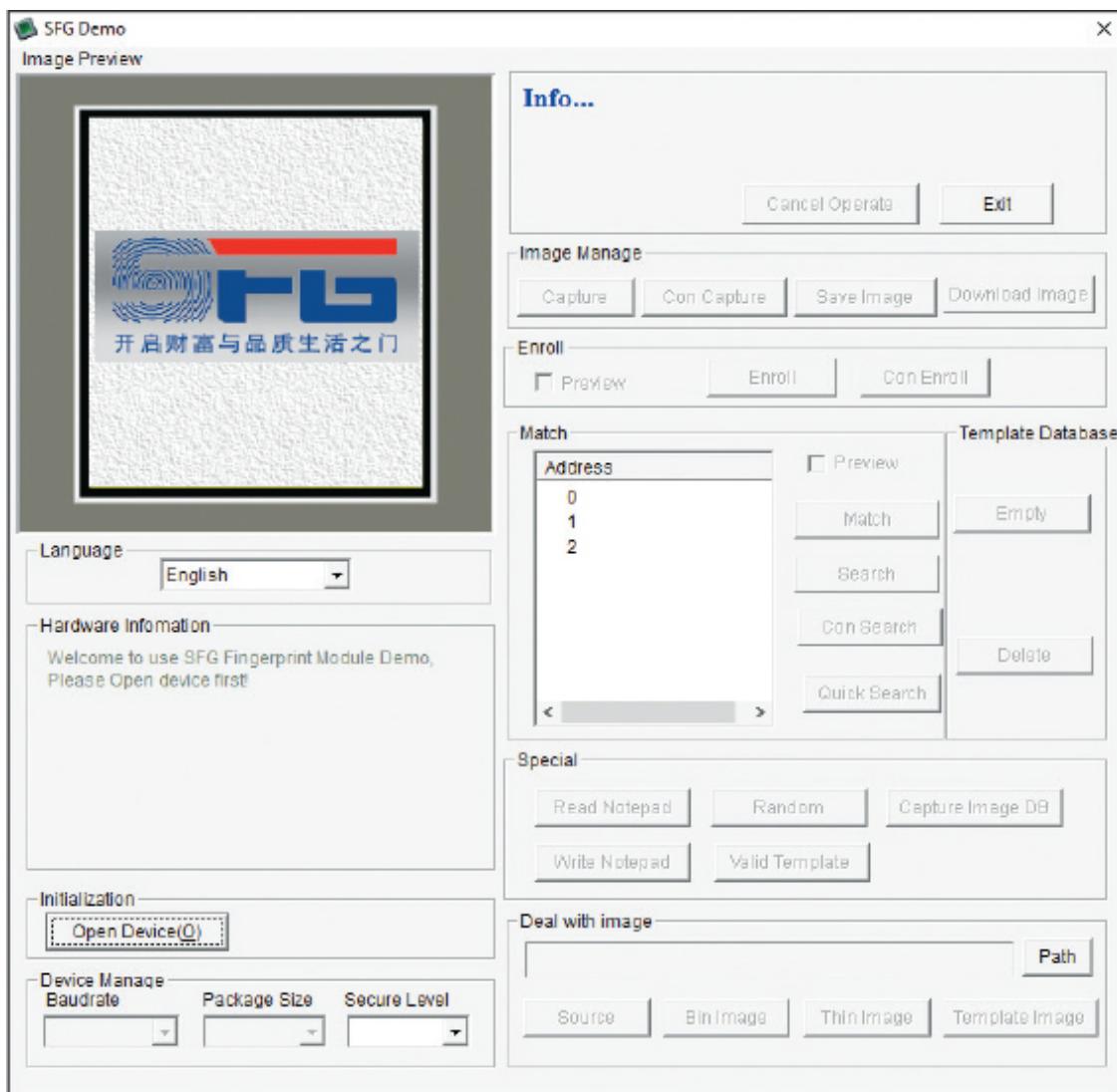


## PREPARING THE FINGERPRINT SENSOR

To use the sensor, we must first get the SFG Demo software, available to download from <http://www.adafruit.com/datasheets/SFGDemoV2.0.rar>. The SFG Demo software is a simple, free program that connects your PC to the Fingerprint ID module via an Arduino so you can control it, add or delete fingerprints, and assign an ID for each one.

1. Download the *SFGDemoV2.0.rar* file and unzip to a destination of your choice.
2. Once you have unzipped the *.rar* file, double-click the *SFGDemo.exe* file to run the program, and you'll see the screen shown in Figure 21-2.

**FIGURE 21-2:** The SFGDemo control screen



3. Now you need to connect the fingerprint sensor module to your PC via the Arduino. The connections for the module to Arduino are shown in the following table.

FINGERPRINT SENSOR	ARDUINO
GND (black wire)	+5V
RX (white wire)	Pin 0 (RX)
TX (green wire)	Pin 1 (TX)

FINGERPRINT SENSOR	ARDUINO
+5V (red wire)	+5V

4. You'll be using the Arduino as a bypass to connect the fingerprint scanner to your PC via the USB cable, so you need to load a blank sketch to get the Arduino to connect to the PC without carrying out a function. The easiest way to do this is to open the latest version of the Arduino IDE and upload the default sketch, shown next.

---

```

void setup() {
    // put your setup code here, to run once:

}

void loop() {
    // put your main code here, to run repeatedly:

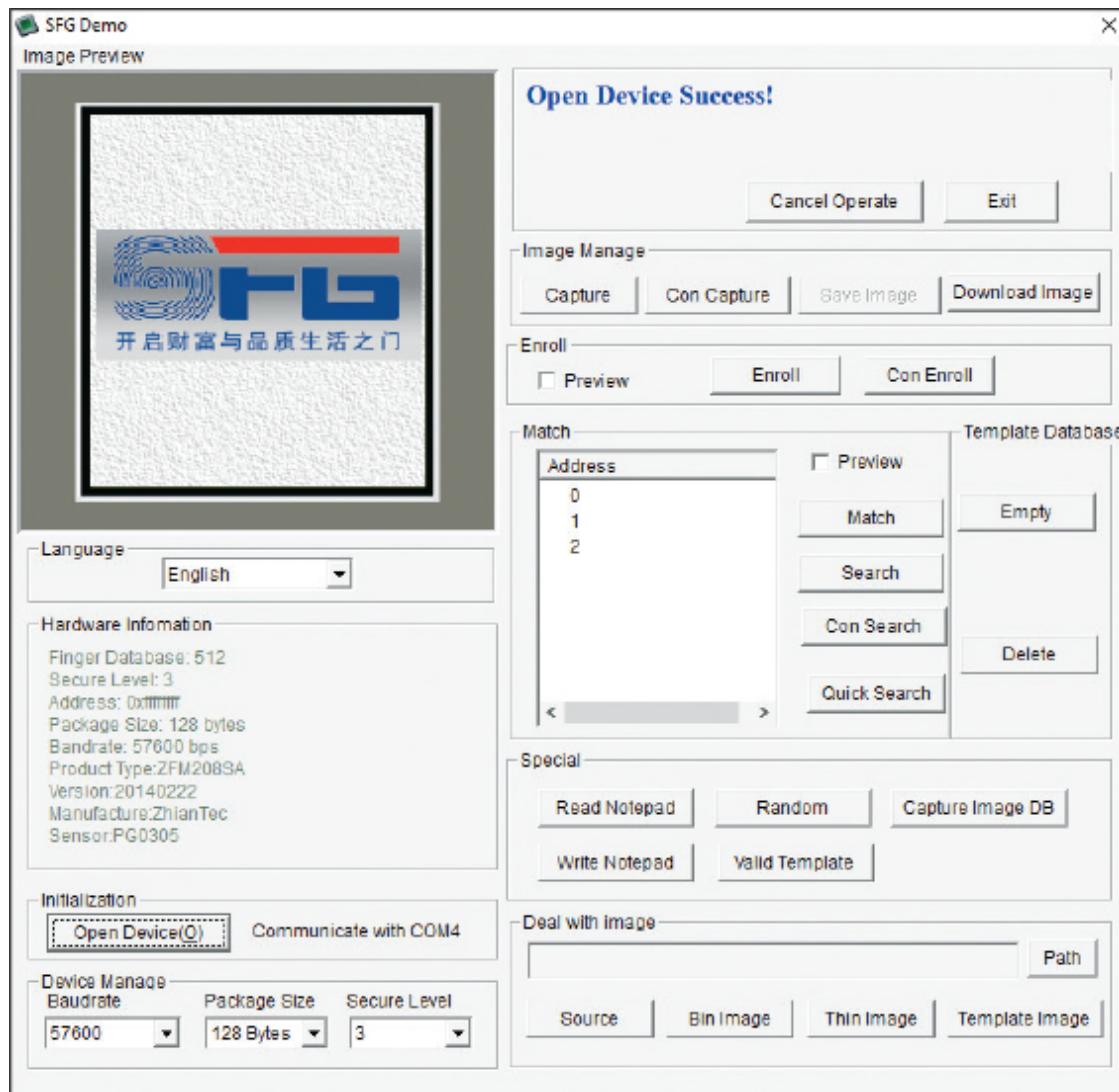
}

```

---

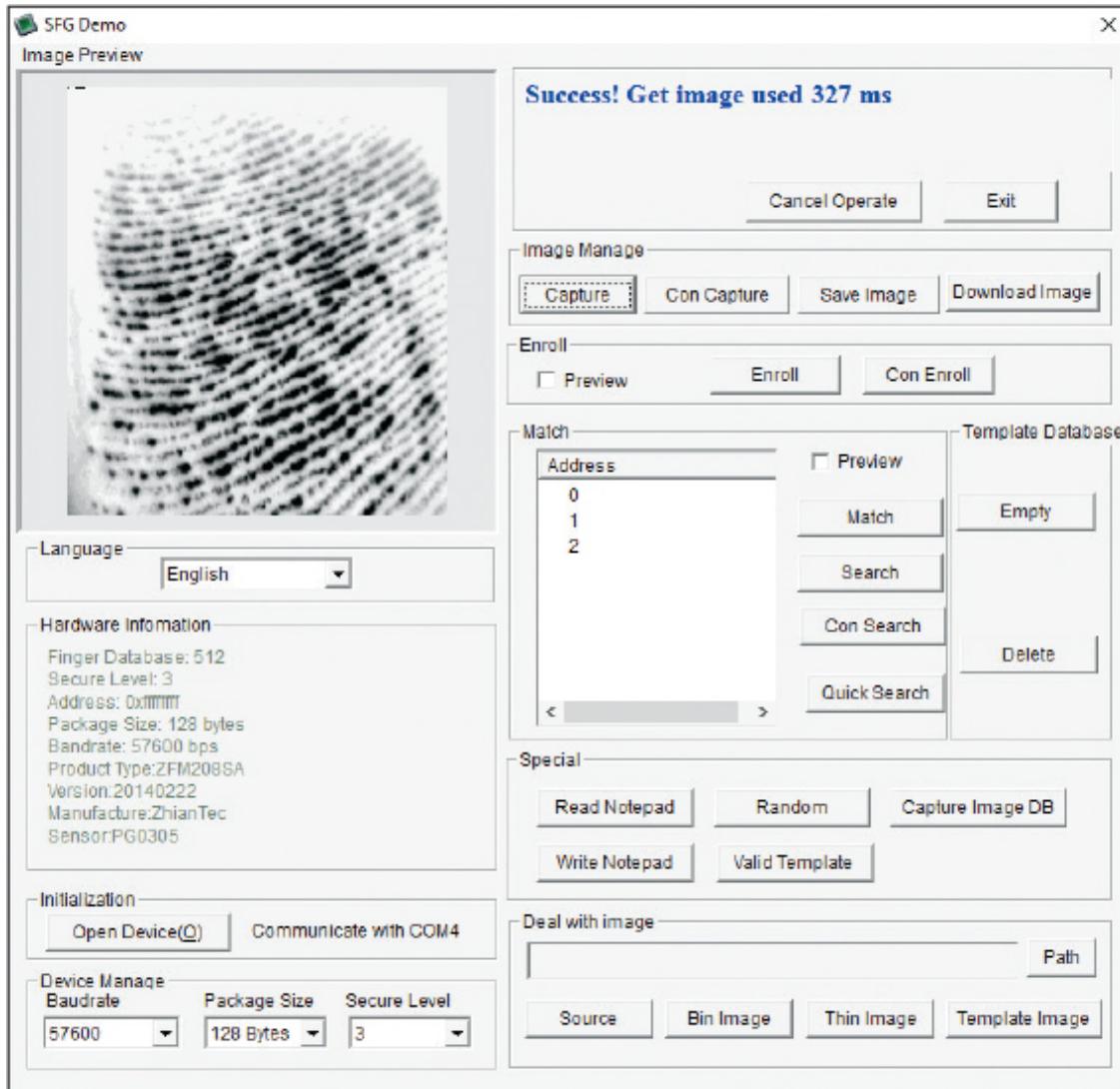
5. Next, connect the Arduino to your PC and in the SFGDemo program, select the **Open Device** button. From the **Com Port** drop-down menu that opens, choose the port your Arduino is connected to and click **OK**. You'll see a message indicating that your module is connected and recognized, as shown in Figure 21-3. Here my module is connected to the Arduino through com port 4, but you might need to use a different port.

**FIGURE 21-3:** When the device is connected correctly, the program shows the message “Open Device Success!”



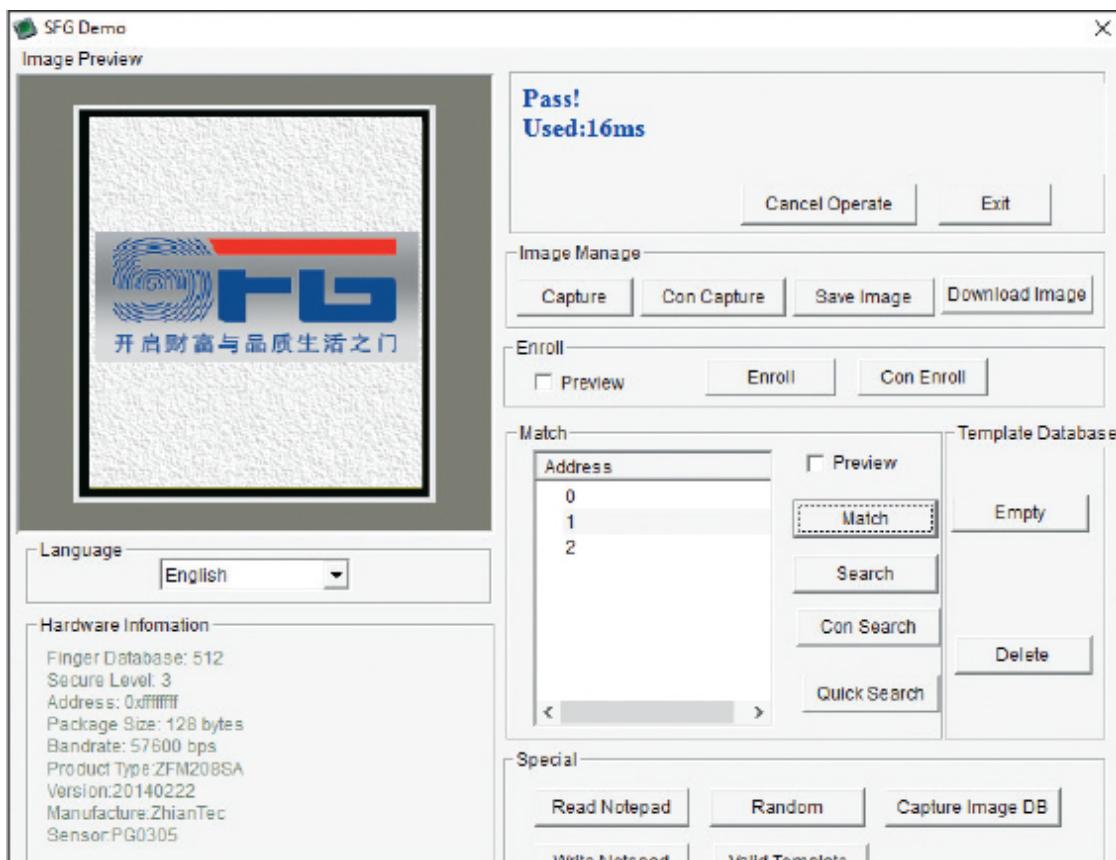
6. Next, you'll add a fingerprint to the database. Click **Enroll** on the SFGDemo control screen. When you see the message "Waiting for fingerprint," press a finger firmly against the fingerprint sensor module window and wait a few seconds. When the print is registered, you'll see the message "Success!" (as shown in Figure 21-4).

**FIGURE 21-4:** The module has successfully captured a fingerprint and shows a preview of the print in the top-left window.



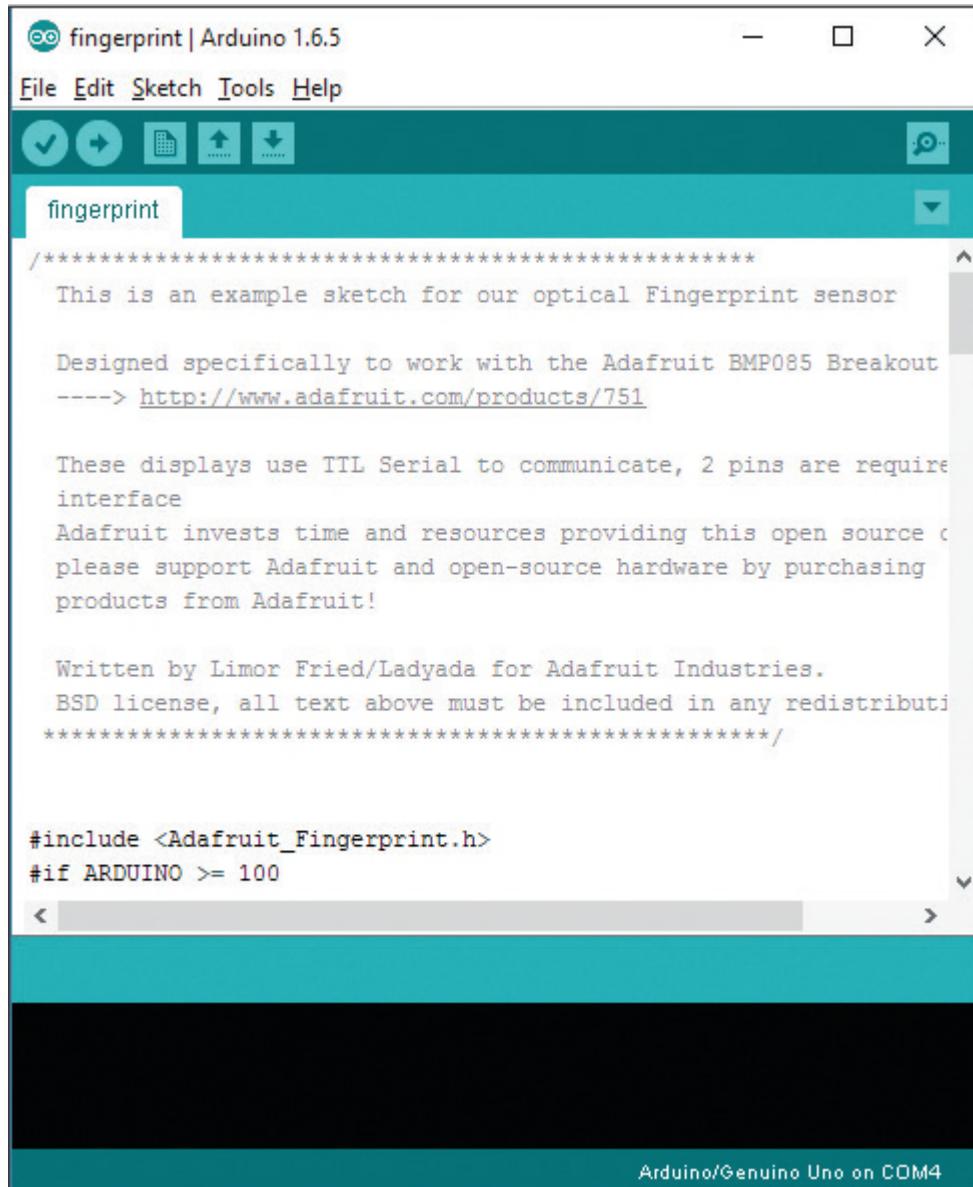
7. Now you'll test whether the module recognizes the fingerprint you just recorded. Click the **Match** button on the SFGDemo control screen. When prompted, press your finger against the window firmly for a few seconds. If the demo finds a match, you'll see the "Pass!" message shown in Figure 21-5.

**FIGURE 21-5:** The fingerprint matches and a "Pass!" message displays in the information panel of the SFGDemo control panel.



8. Now you need to check that the module recognizes your fingerprint when it's attached to the Arduino and not the PC. Close the SFGDemo program and, from the resources you downloaded from <https://www.nostarch.com/arduinohandbook2/>, add the Adafruit Fingerprint Sensor library to your IDE. If you need a refresher on adding libraries, check out the library section at the start of this book.
9. Once you've added the Adafruit Fingerprint Sensor library, open the IDE and select **Files** ▶ **Examples** ▶ **Adafruit-fingerprint-sensor-master** ▶ **Fingerprint** to choose the library fingerprint sketch shown in Figure 21-6. Upload this sketch to your Arduino.

**FIGURE 21-6:** The fingerprint demo sketch from the Adafruit Fingerprint Sensor library



The screenshot shows the Arduino IDE interface with the title bar "fingerprint | Arduino 1.6.5". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for save, undo, redo, cut, copy, paste, and upload. The main window displays the "fingerprint" sketch code. The code is a comment block that describes the sketch as an example for an optical Fingerprint sensor, designed for Adafruit BMP085 Breakout, and provides a link to Adafruit's product page. It also notes that the displays use TTL Serial communication and encourages supporting Adafruit. The code ends with a BSD license notice. At the bottom of the code editor, there is a status bar with the text "Arduino/Genuino Uno on COM4".

```
/*
This is an example sketch for our optical Fingerprint sensor
Designed specifically to work with the Adafruit BMP085 Breakout
----> http://www.adafruit.com/products/751

These displays use TTL Serial to communicate, 2 pins are required
interface
Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.
BSD license, all text above must be included in any redistribution
****/
```

```
#include <Adafruit_Fingerprint.h>
#if ARDUINO >= 100
```

## NOTE

*Your sensor may come with six wires, two of which aren't necessary for the demo.*

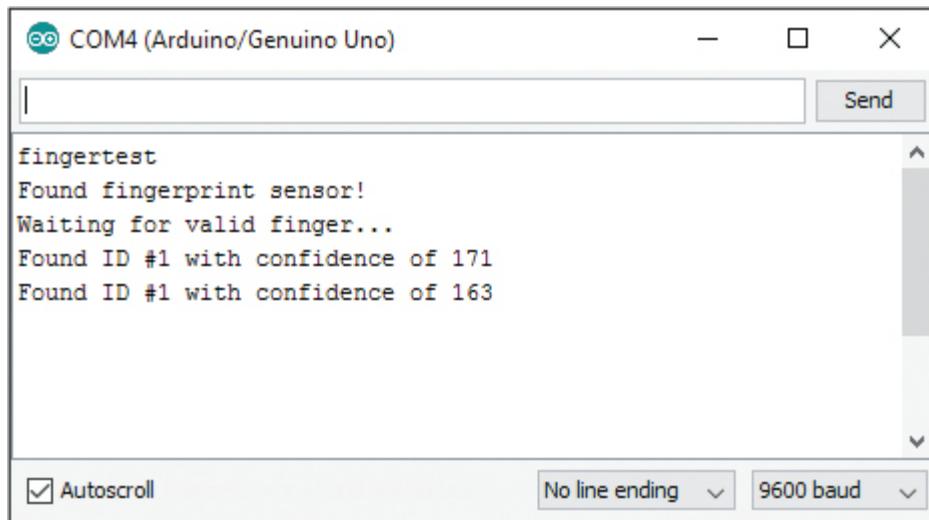
- Once you've uploaded the fingerprint sketch to your Arduino, disconnect from the PC. You now need to change your module/Arduino pin setup. Instead of connecting the module to the TX and RX pins, change these connections to pins 2 and 3 on the

Arduino, respectively, as shown in the following table. This keeps the TX and RX serial communication free to use the Arduino IDE Serial Monitor in the next step.

FINGERPRINT SENSOR	ARDUINO
GND (black wire)	+5V
TX (green wire)	Pin 2
RX (white wire)	Pin 3
+5V (red wire)	+5V

11. Now, reconnect your Arduino to the PC and open the Arduino IDE. Open the Serial Monitor of the IDE. When you press your finger to the module window, you should see something like Figure 21-7.

**FIGURE 21-7:** The module processes are displayed in the Arduino IDE serial screen.



## THE BUILD

Now that you know the module is working as expected, you'll use what you've learned to create the fingerprint entry system.

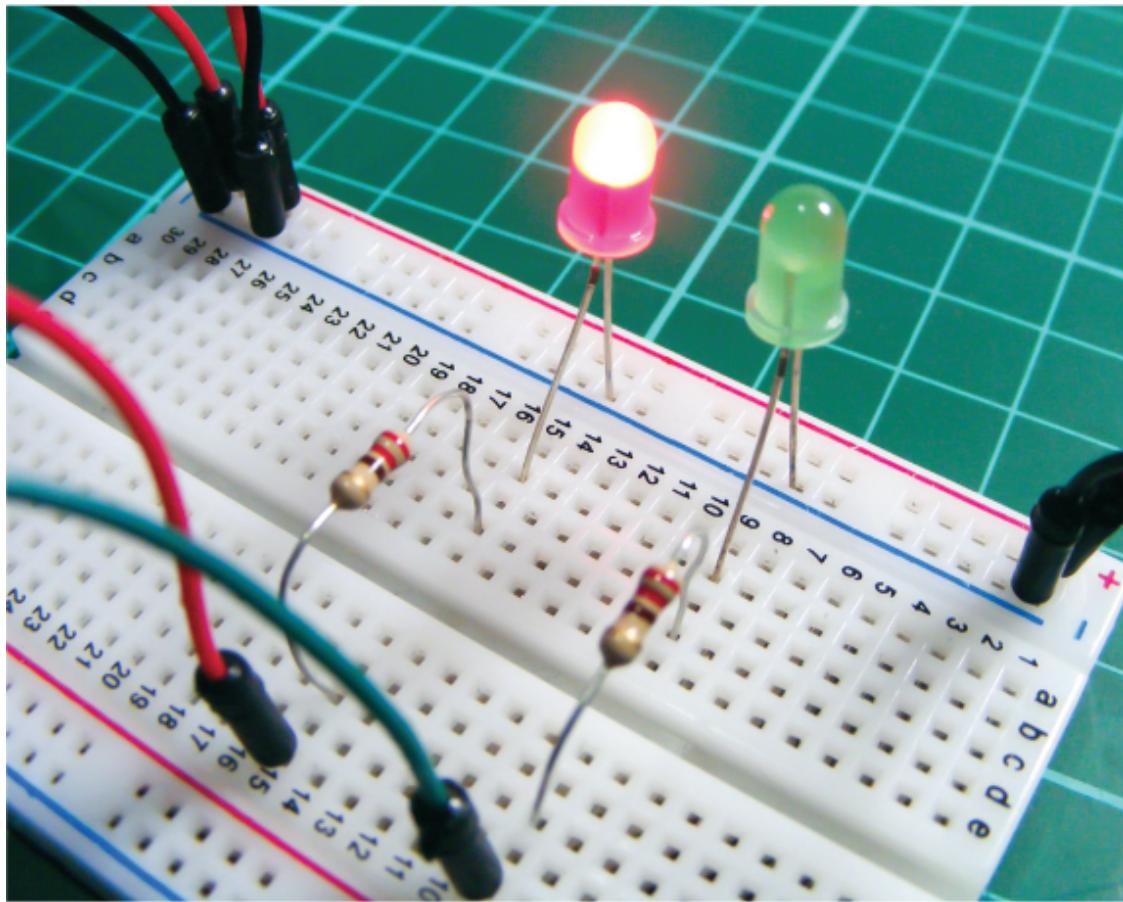
1. The fingerprint module should now be connected to the Arduino, but if you're starting at this point, follow the connections given in step 10 before proceeding.
2. Connect the servomotor to the GND and +5V power rails on the breadboard and connect the signal pin to Arduino pin 9, as shown in the following table.

SERVO	ARDUINO
Signal (yellow wire)	Pin 9
Positive power (red wire)	Breadboard +5V rail
Negative power (black wire)	Breadboard GND rail

3. Insert the LEDs into the breadboard so that the shorter, negative leg is connected to the GND power rail of the breadboard and the positive, longer leg is connected to Arduino pins 7 and 8 via a 220-ohm resistor, as shown in the following table. The resistors should straddle the center of the breadboard, as shown in Figure 21-8.

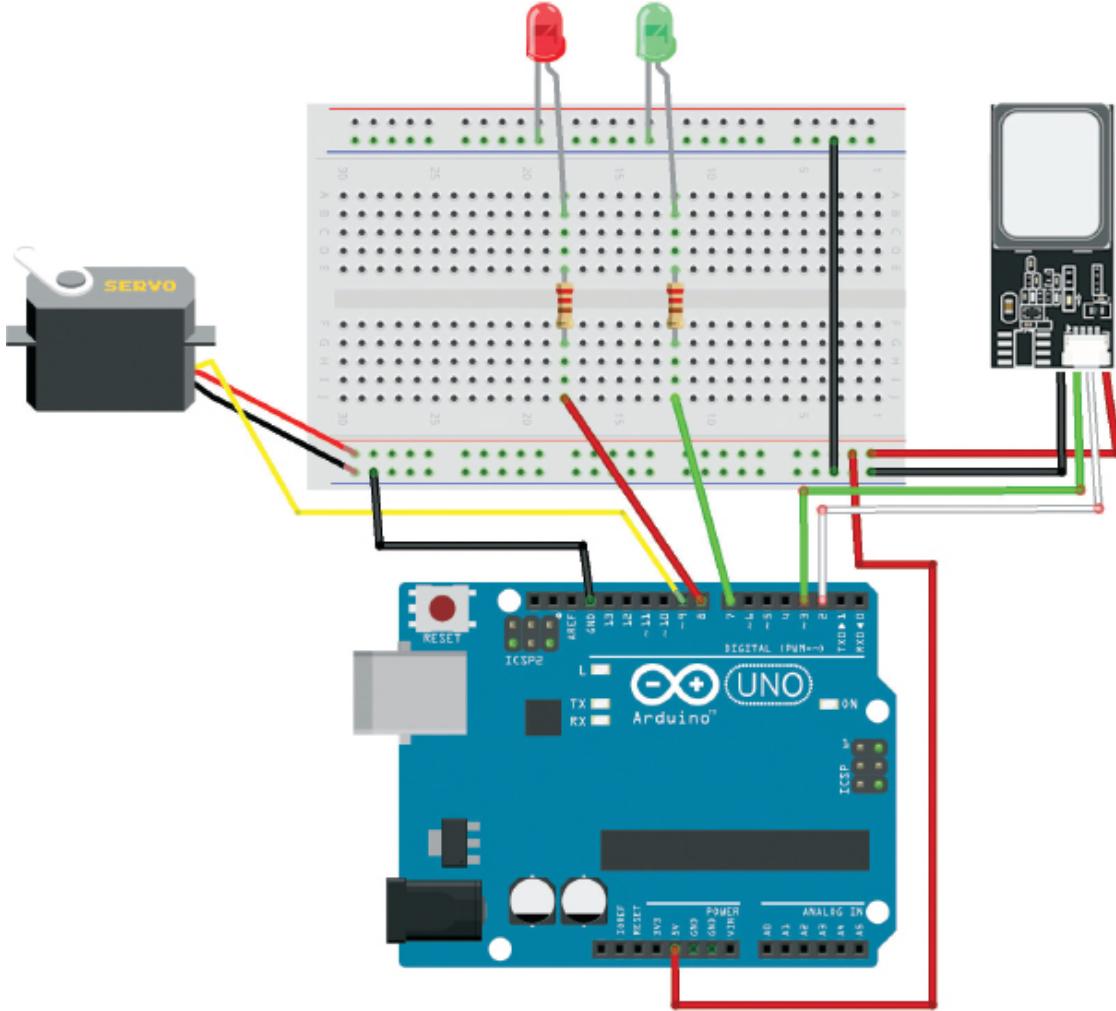
LEDS	ARDUINO
Green LED (positive, longer leg)	Pin 7 via 220-ohm resistor
Red LED (positive, longer leg)	Pin 8 via 220-ohm resistor
Negative power of both LEDs (shorter leg)	Breadboard GND rail

**FIGURE 21-8:** The LEDs are connected to the Arduino pins via 220-ohm resistors.



4. Connect the power rails of the breadboard to +5V and GND on the Arduino, and then check that your circuit matches Figure 21-9.
5. Upload the code in “The Sketch” on page 183.

**FIGURE 21-9:** The circuit diagram for the fingerprint scanner



# THE SKETCH

The sketch first calls on the Servo, SoftwareSerial, and Adafruit\_Fingerprint libraries. The LED and servo pins are defined as 7, 8, and 9, respectively, and pins 2 and 3 are defined for serial connection to the fingerprint sensor module. The fingerprint library handles the functionality of the module, and the sketch has a series of steps to read and store a fingerprint.

The sensor automatically scans every 5 seconds and reads the fingerprint when it is pressed to the window. If the fingerprint matches one in the module memory (which we stored earlier in the project), the red LED will turn off, the green LED will light, and the servomotor

will turn 180 degrees. This state will continue for 5 seconds, and the setup will reset and wait for another valid entry.

---

```
// Fingerprint Sensor Library reproduced with kind permission
// from Adafruit Industries
/*****************/
This is an example sketch for our optical Fingerprint sensor
```

Designed specifically to work with the Adafruit BMP085 Breakout  
----> <http://www.adafruit.com/products/751>

These displays use TTL Serial to communicate, 2 pins are required to interface

Adafruit invests time and resources providing this open source code,  
please support Adafruit and open-source hardware by purchasing  
products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.  
BSD license, all text above must be included in any redistribution
\*\*\*\*\*\*/

```
#include <Servo.h>
#include <Adafruit_Fingerprint.h>
#if ARDUINO >= 100
#include <SoftwareSerial.h>
#else
#include <NewSoftSerial.h>
#endif

int getFingerprintIDez();
int ledaccess = 7; // Green LED pin
int leddeny = 8; // Red LED pin
int servoPin = 9; // Servo pin

Servo doorLock;

// Pin #2 is IN from sensor (GREEN wire)
// Pin #3 is OUT from arduino (WHITE wire)
#if ARDUINO >= 100
SoftwareSerial mySerial(2, 3); // Pins for the fingerprint sensor
#else
NewSoftSerial mySerial(2, 3);
#endif

Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);

void setup() {
  doorLock.attach(servoPin); // We define the servo pin
  pinMode(ledaccess, OUTPUT); // Green LED pin set as an output
```

```

pinMode(leddeny, OUTPUT); // Red LED pin set as an output
pinMode(servopin, OUTPUT); // Servo pin set as an output
Serial.begin(9600); // Start sending messages to the Serial Monitor
Serial.println("fingertest");
finger.begin(57600); // Set data rate for the sensor serial port

// Start the module and checking for fingerprint
if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
} else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1);
}
Serial.println("Waiting for valid finger...");
}

void loop() {
    int ID = getFingerprintID(); // Get the fingerprint ID#
    // Reset the device to the test state
    digitalWrite(ledaccess, HIGH);
    digitalWrite(leddeny, HIGH);
    doorLock.write(0);
    if (ID >= 0) { // Valid ID. Unlocked state
        // Enable the access LED, turn off the deny LED
        digitalWrite(ledaccess, HIGH);
        digitalWrite(leddeny, LOW);
        // Unlock the servo
        doorLock.write(180);
    }
    else if (ID == -3) { // ID doesn't match any registered print
        // Locked state
        // Enable the deny LED, turn off the access LED
        digitalWrite(ledaccess, LOW);
        digitalWrite(leddeny, HIGH);
    }
    delay(5000);
}

uint8_t getFingerprintID() {
    uint8_t p = finger.getImage();
    switch (p) {
        case FINGERPRINT_OK: // Sensor takes a photo when a finger is
            // placed on the module window
            Serial.println("Image taken");
            break;
        case FINGERPRINT_NOFINGER:
            Serial.println("No finger detected");
            return p;
        case FINGERPRINT_PACKETRECEIVEERR:
            Serial.println("Communication error");
            return p;
    }
}

```

```

case FINGERPRINT_IMAGEFAIL:
    Serial.println("Imaging error");
    return p;
default:
    Serial.println("Unknown error");
    return p;
}

p = finger.image2Tz(); // OK success! We have a fingerprint and
                      // now check that it can be read
switch (p) {
    case FINGERPRINT_OK:
        Serial.println("Image converted");
        break;
    case FINGERPRINT_IMAGEMESS:
        Serial.println("Image too messy");
        return p;

    case FINGERPRINT_PACKETRECIEVEERR:
        Serial.println("Communication error");
        return p;
    case FINGERPRINT_FEATUREFAIL:
        Serial.println("Could not find fingerprint features");
        return p;
    case FINGERPRINT_INVALIDIMAGE:
        Serial.println("Could not find fingerprint features");
        return p;
    default:
        Serial.println("Unknown error");
        return p;
}

p = finger.fingerFastSearch(); // OK converted! It's valid, so
                             // check it against module memory
if (p == FINGERPRINT_OK) {
    Serial.println("Found a print match!");
} else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
} else if (p == FINGERPRINT_NOTFOUND) {
    Serial.println("Did not find a match"); // No match found,
                                         // back to the start
    return p;
} else {
    Serial.println("Unknown error");
    return p;
}
// We found a match! So the following will run:
Serial.print("Found ID #"); Serial.print(finger.fingerID);
Serial.print(" with confidence of "); Serial.println(finger.confidence);

```

```

    return finger.fingerID;
}
// Returns -1 if failed, otherwise returns ID #
int getFingerprintIDez() {
    int p = finger.getImage();
    if (p != FINGERPRINT_OK) return -1;

    p = finger.image2Tz();
    if (p != FINGERPRINT_OK) return -2;

    p = finger.fingerFastSearch();
    if (p != FINGERPRINT_OK) {
        Serial.println("No match found");
        return -3;
    }

    // Found a match!
    Serial.print("Found ID #"); Serial.print(finger.fingerID);
    Serial.print(" with confidence of "); Serial.println(finger.confidence);
    return finger.fingerID;
}
-----
```

## TROUBLESHOOTING

**Q.** *The code compiles, but the fingerprint sensor does not light up or function.*

- Make sure that your wiring matches the tables on page 181 and page 182. This code will work only with the fingerprint sensor I've used in this project.
- If your sensor has six wires instead of the expected four and the wire colors don't match as described, it is the first four pins you need: GND, TX, RX, and +5V. The other two connections are not used in this project, so you can remove these wires.
- If your module still does not light up, check the data sheet for the actual pin configuration and reconnect the wires according to that.
- Remember you need to set up the module first and test it as described in “Preparing the Fingerprint Sensor” on page 176.

**Q.** *The LEDs do not light up as expected.*

- Ensure the LEDs are firmly inserted into the breadboard and the resistors line up with the connections to the Arduino.

- Remember to connect power to the breadboard rails.

**Q.** *The servomotor does not move as expected.*

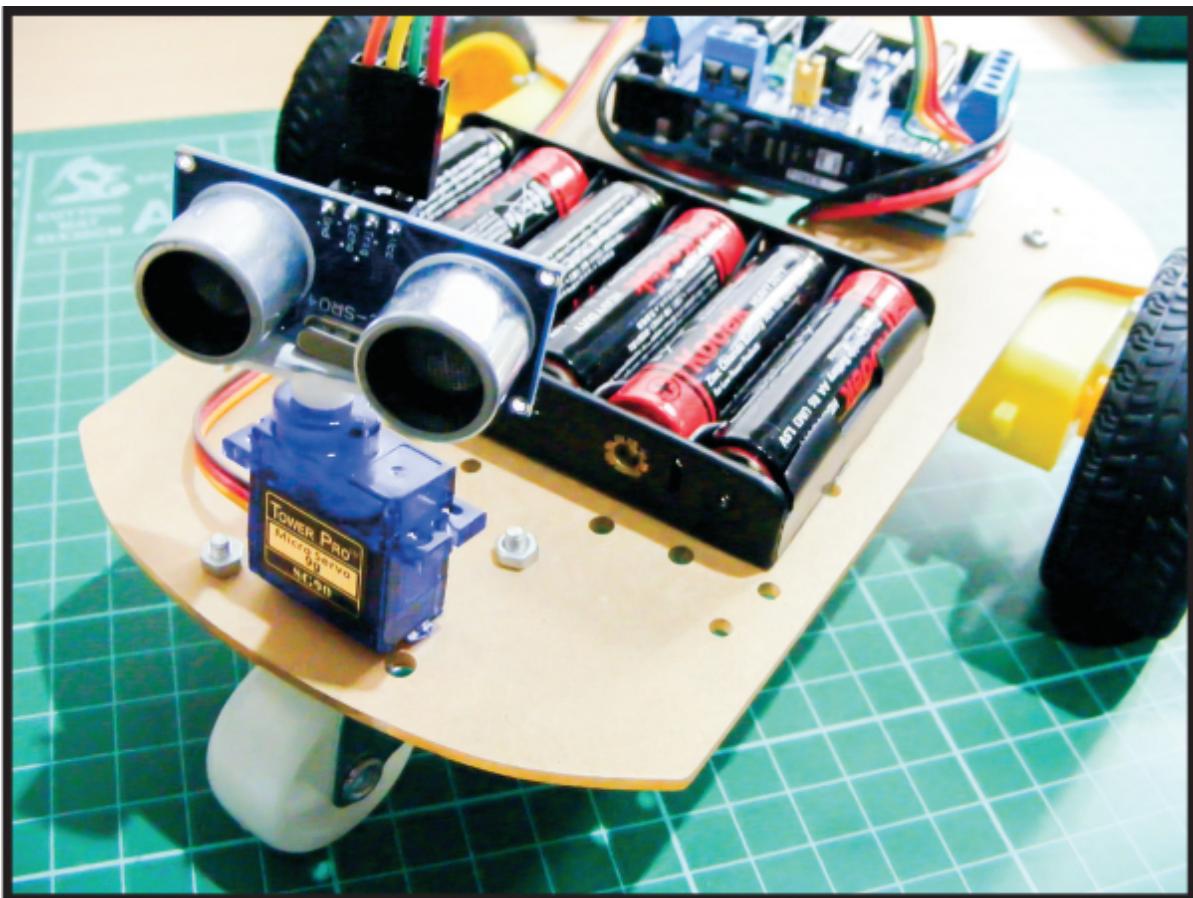
- Double-check that the wiring matches the servo connections shown in Figure 21-9.
- The module, servo, and LEDs combined draw a fair amount of power from your battery pack, and while the Arduino can still function at a lower voltage, the servomotor cannot. Change to fresh batteries.

# Smart Machines

**22**

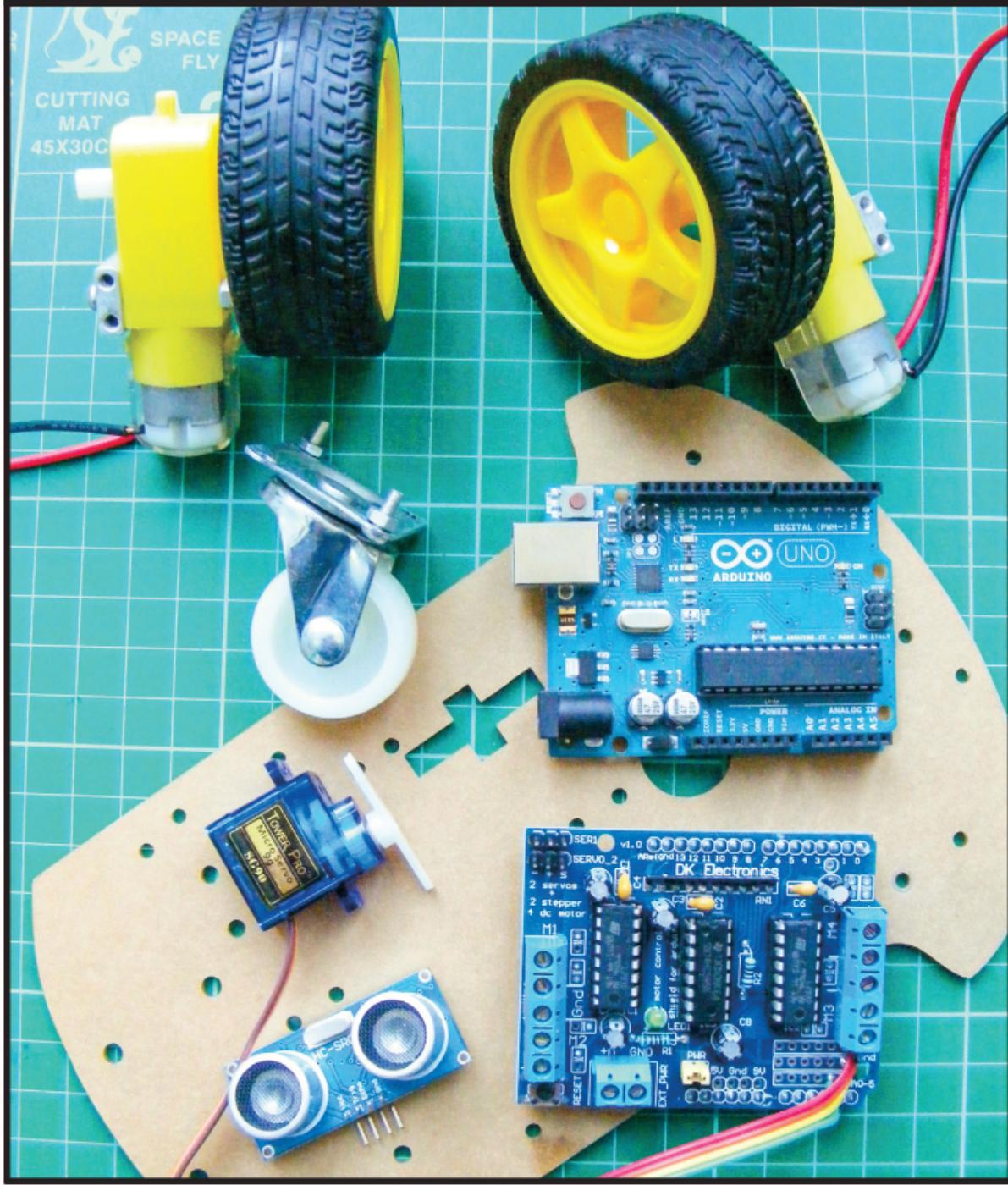
## **Ultrasonic Robot**

In this project we'll combine an ultrasonic sensor with two DC motors and a servomotor to create a simple object-avoiding robot.



**COST: \$\$\$\$\$**

**TIME: 2 HOURS**



## PARTS REQUIRED

Arduino board

Jumper wires

**L293d motor shield**  
**2 DC motors and wheels\***  
**HC-SR04 ultrasonic sensor**  
**9V AA battery pack**  
**Robot base with fittings\***  
**Center wheel\***  
**Tower Pro SG90 9g servomotor**

## **LIBRARIES REQUIRED**

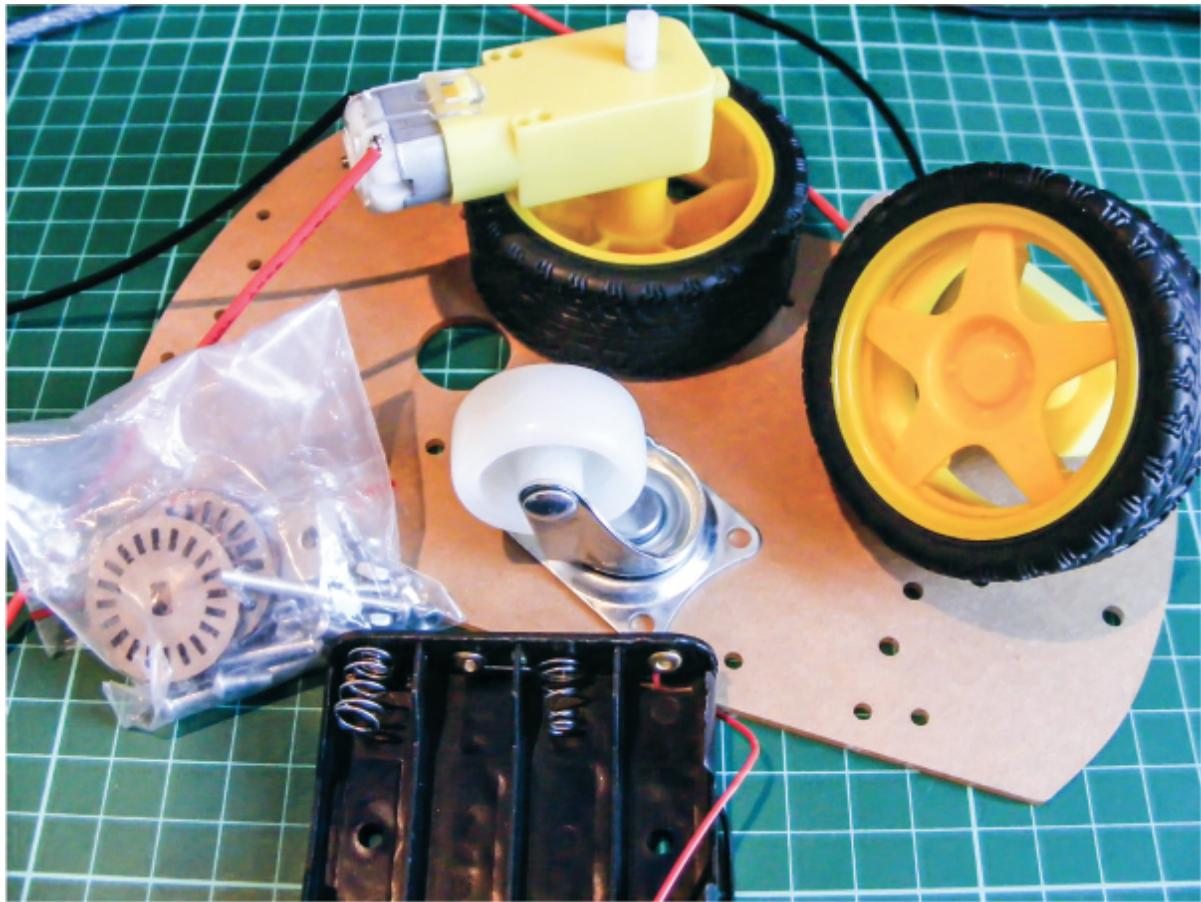
**Servo**  
**NewPing**  
**Adafruit Motor Shield V1**

\* These items can be purchased as part of a kit

## **HOW IT WORKS**

The key parts of the ultrasonic robot are the HC-SR04 ultrasonic sensor, L293d motor shield, and the motors. The motors I used were purchased as part of a kit; if you search online for “Arduino robot kit,” you too should be able to find a kit that contains the motors and wheels, base, battery pack, center wheel, and fittings needed. The one I bought is called the “2WD Smart Motor Robot Car Chassis Kit for Arduino 1:48,” so try a few of those keywords until you find something similar to the kit in Figure 22-1. Also try the suppliers listed in the “Retailer List” on page 249.

**FIGURE 22-1:** Robot motor kit



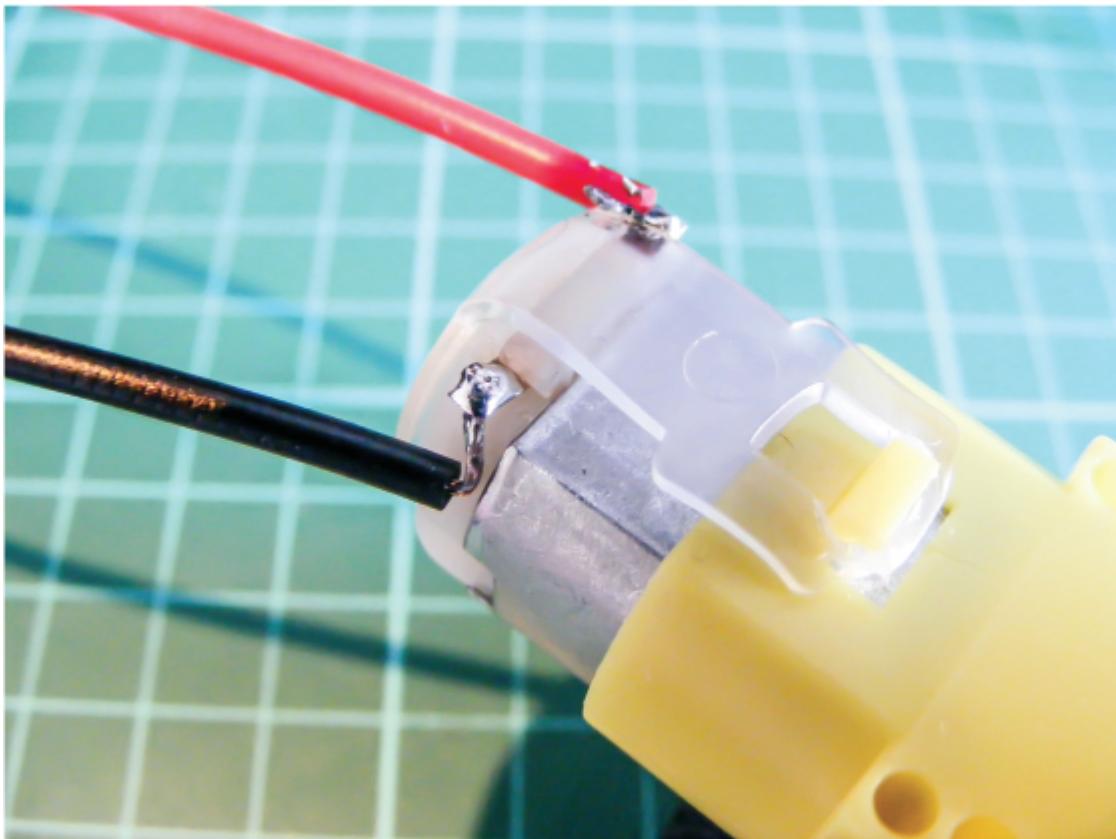
The ultrasonic sensor sends and receives a signal to determine the distance of an object. If there is an object less than 15 centimeters away, the robot will stop, look around, turn toward a direction in which it doesn't sense anything, and move in that direction. The ultrasonic sensor is mounted on a servomotor so that the robot can move and search for a clear route. For more on how the HC-SR04 ultrasonic sensor works, see Project 13. The L293d motor shield fits on top of the Arduino and controls the DC motors using the Adafruit Motor Shield library.

## THE BUILD

1. You will need to solder wires to the DC motors as shown in Figure 22-2. See the “Quick Soldering Guide” on page 12 if you need a refresher on how to do this. Solder the red, positive power wire to

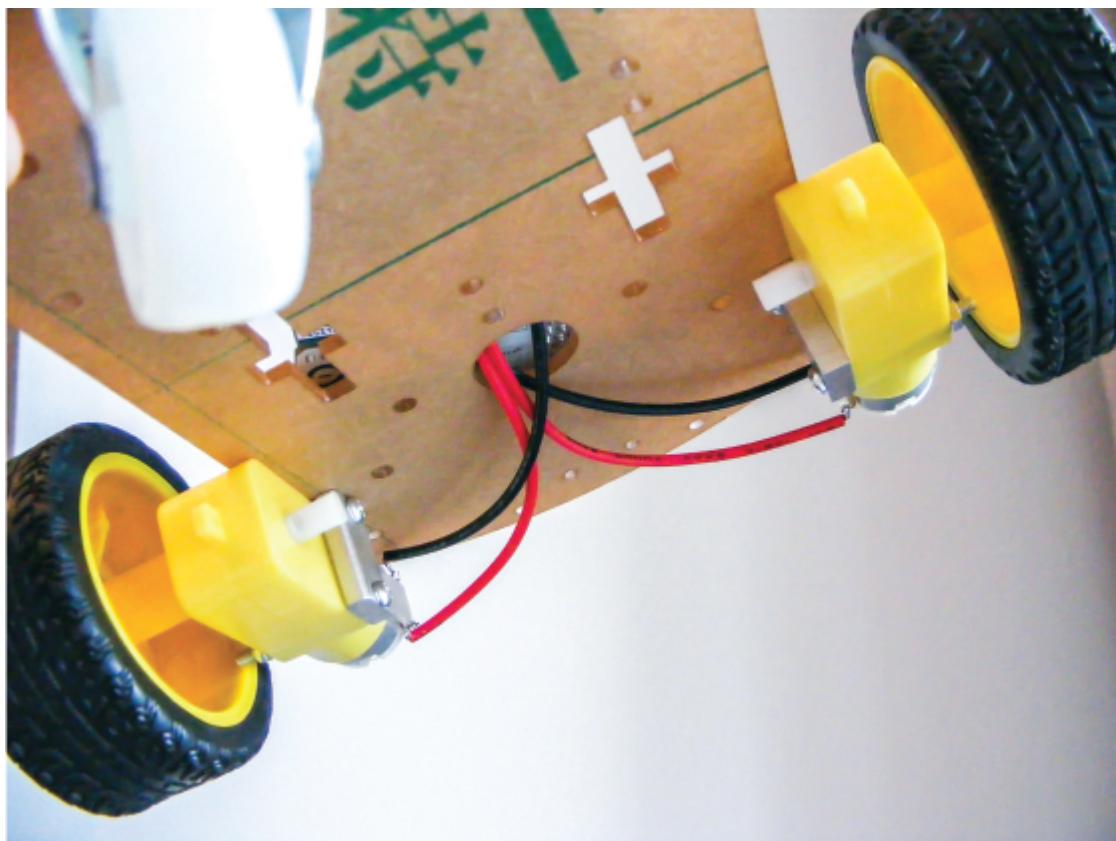
the left pin of one DC motor and the black ground wire to the right pin; reverse this order for the other motor. DC motors do not have polarity, so it doesn't matter which way you hold the motors to determine which is left and right, but power and GND need to be in opposite positions on the motors so the direction of the revolution will be the same.

**FIGURE 22-2:** Solder the red, positive power wire to the left pin of one DC motor, and the black ground wire to the right pin. Reverse this order for the other motor.



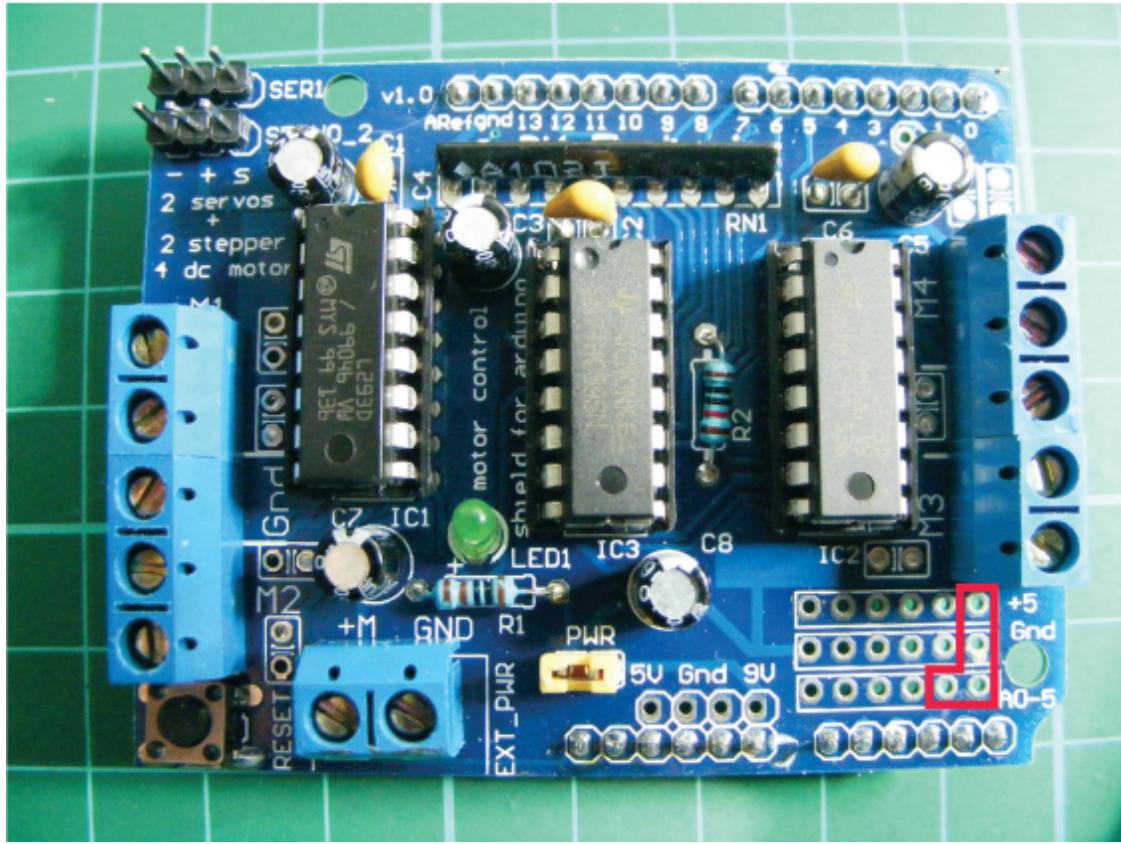
2. Attach the single wheel to the front of the robot base and the two rear wheels to the back using the screws and fittings provided. The underside of the robot should resemble Figure 22-3.

**FIGURE 22-3:** Assemble the base of the Arduino robot.



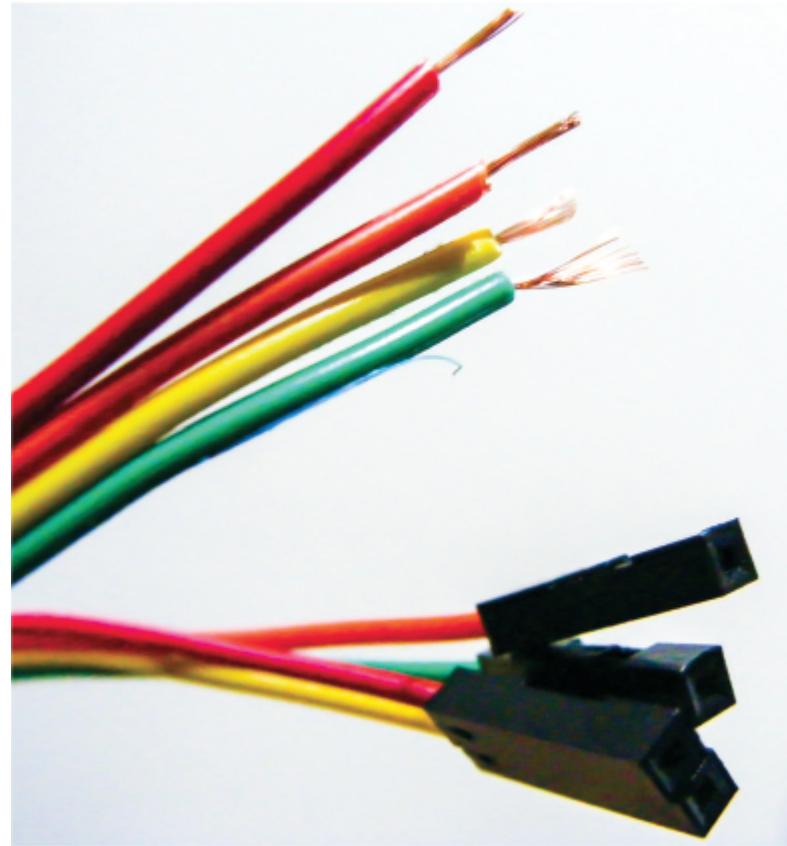
3. Now you need the L293d motor shield (Figure 22-4); we'll solder some wires to it to control the ultrasonic sensor.

**FIGURE 22-4:** The L293d motor shield. We'll solder four wires to the pins highlighted in the image.



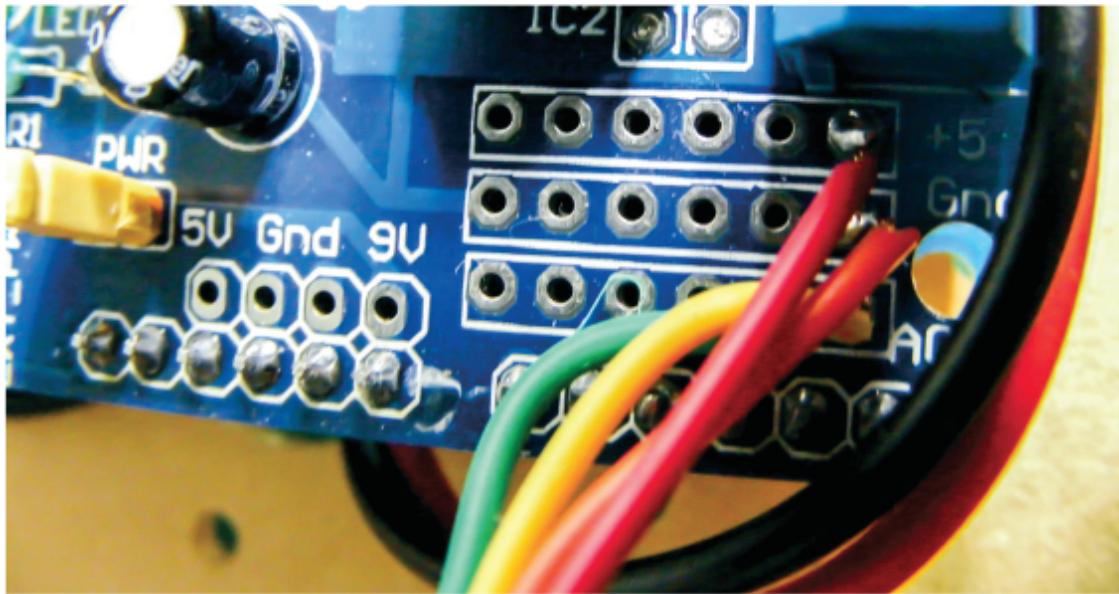
4. Take four female jumper wires and strip about 5 millimeters from one end of each, as shown in Figure 22-5.

**FIGURE 22-5:** Strip the ends of four female jumper wires to solder onto the motor shield.



5. Solder the stripped ends to the highlighted pins on the motor shield, as shown in Figure 22-6. This can be tricky, so take your time to create the best connection you can.

**FIGURE 22-6:** Solder the jumper wires to the motor shield (shown in Figure 22-4). The two pins below the power connections should connect to analog A4 and A5 to control the sensor.



6. Once you've soldered the wires to the motor shield, place the shield on top of the Arduino so that the pins of the shield line up with the holders in the Arduino below. The shield should fit exactly, but take care to align the pins to the holes and gently lower it in place.
7. Next, connect the ultrasonic sensor to the female ends of the jumper wires you soldered to the motor shield. Connect VCC on the sensor to +5V on the motor shield, Trig to A4, Echo to A5, and GND to GND (see the following table).

ULTRASONIC SENSOR	MOTOR SHIELD
VCC	+5V
Trig	Pin A4
Echo	Pin A5
GND	GND

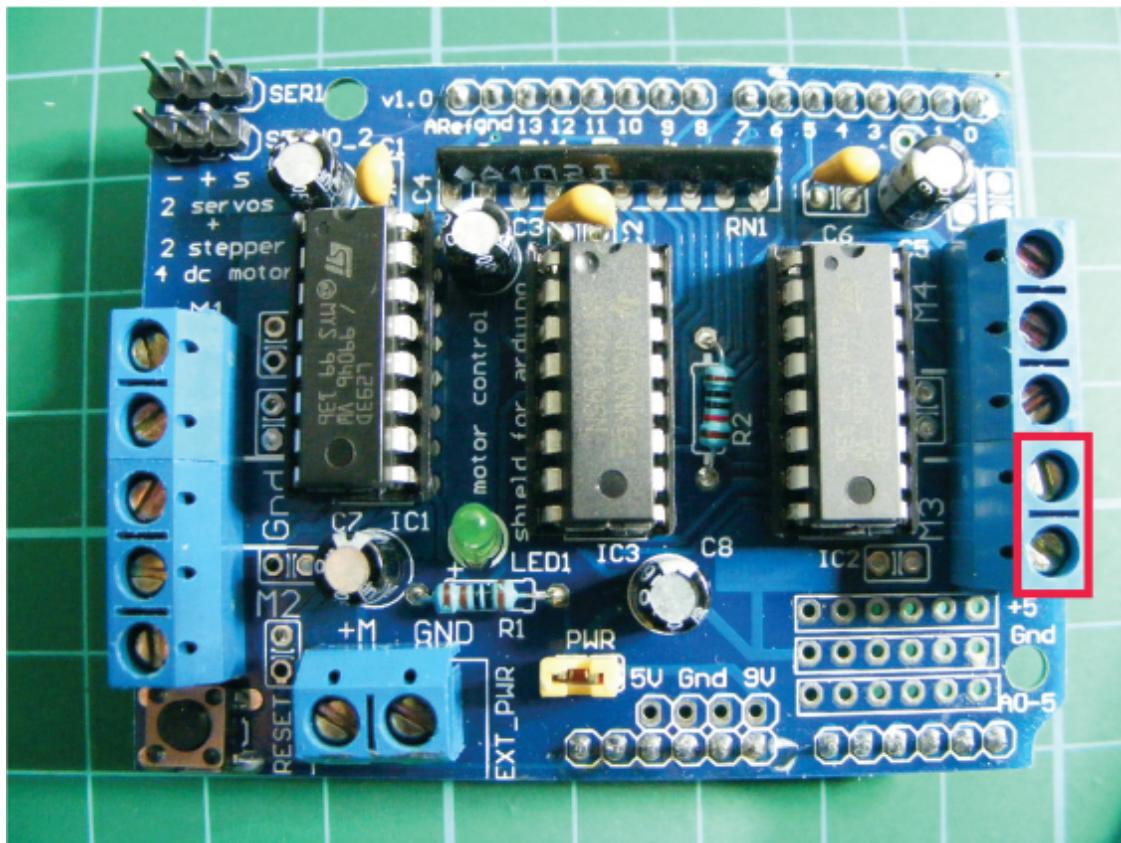
8. Connect the wires from the DC motors to the motor shield as shown in the following tables and Figure 22-7. You connect the

wires by feeding them through the pin and using the screws to grip the wires in place.

LEFT MOTOR	MOTOR SHIELD	ARDUINO
Red wire	M1	+5V
Black wire	M1	GND

RIGHT MOTOR	MOTOR SHIELD	ARDUINO
Red wire	M3	+5V
Black wire	M3	GND

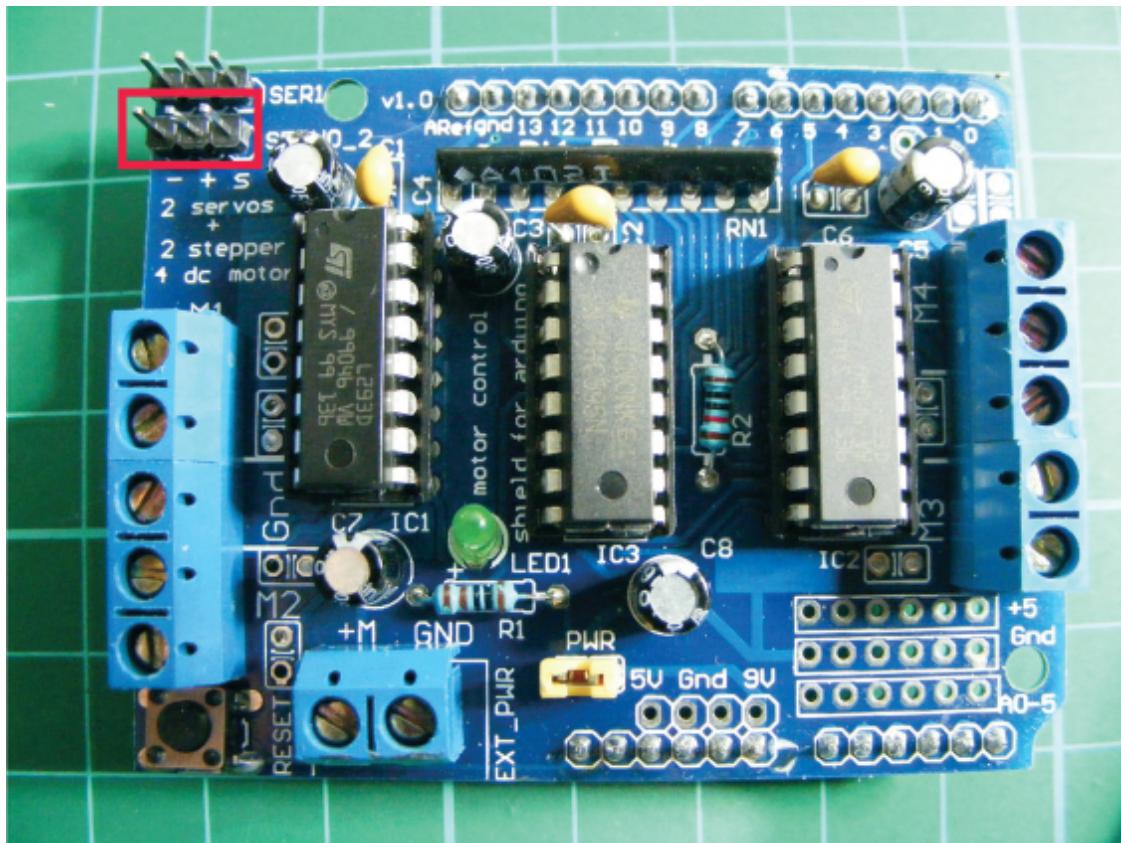
**FIGURE 22-7:** Connect the power wires of the DC motors as shown.



9. Next attach the servomotor to the shield, as shown in the following table and Figure 22-8.

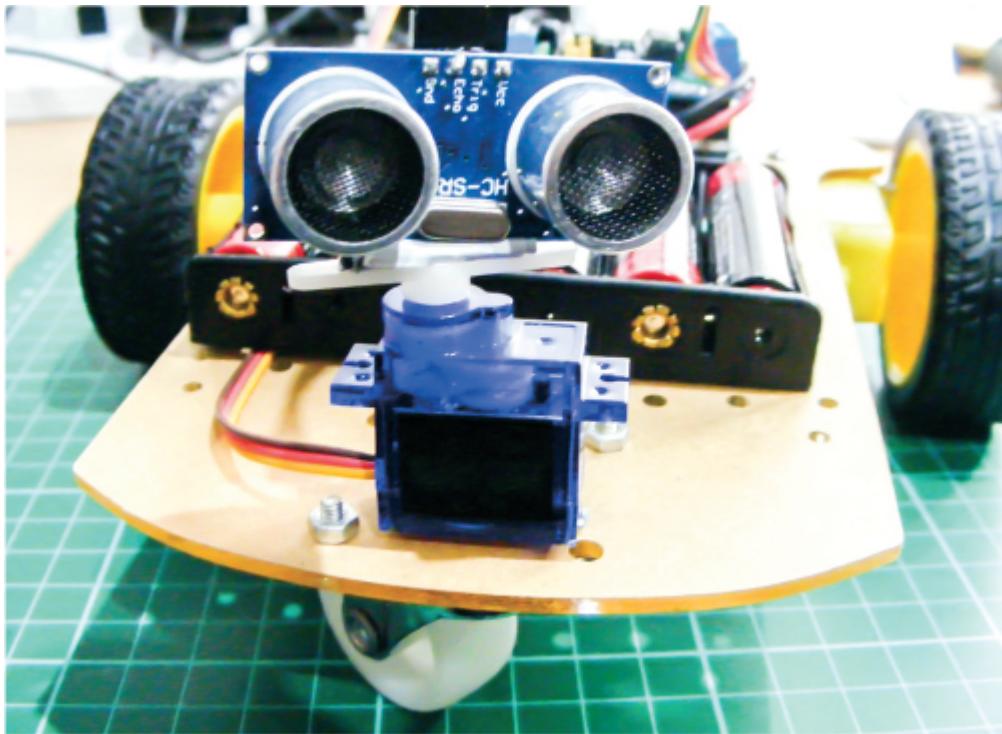
SERVOMOTOR	MOTOR SHIELD	ARDUINO
Brown wire	Servo_2 -	GND
Red wire	Servo_2 +	+5V
Yellow wire	Servo_2 s	Signal

**FIGURE 22-8:** Connect the servomotor to the shield as shown.



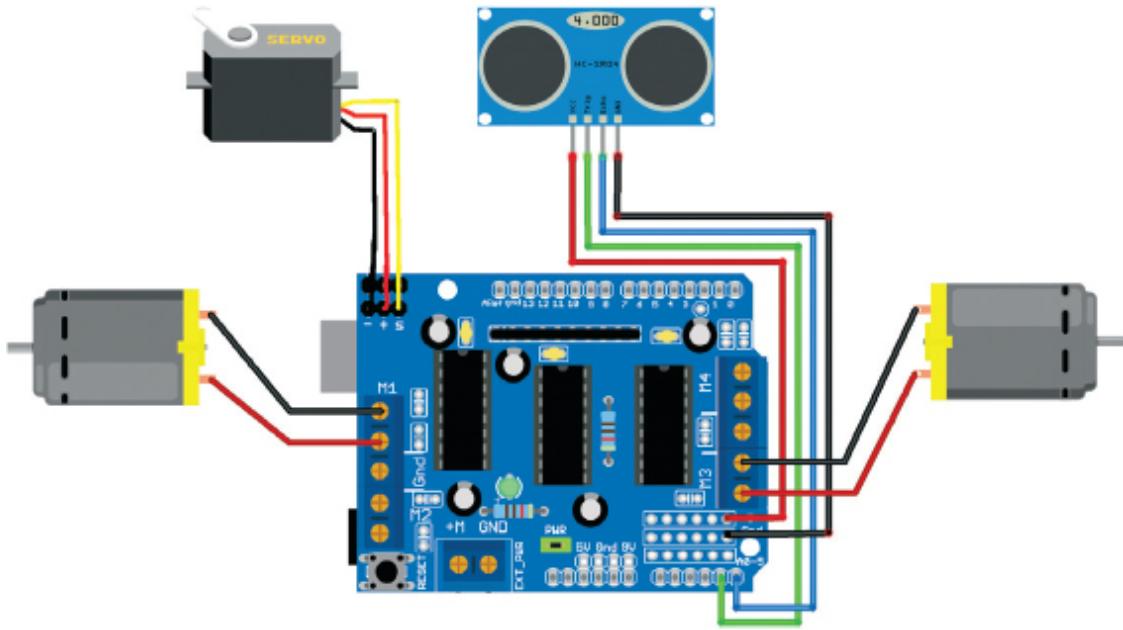
10. Attach the servomotor to the front of the robot using glue or tape. Then attach the ultrasonic sensor to the horn of the servomotor so it moves with the servo arm and your robot can look around. At this stage the robot should look something like Figure 22-9.

**FIGURE 22-9:** The completed robot with ultrasonic sensor attached to the servomotor



11. Make sure you've downloaded the NewPing and Adafruit Motor Shield libraries and added them to your IDE. The Servo library is already included in the IDE, so you don't need to install it.
12. Once you've confirmed that your setup matches the circuit diagram in Figure 22-10, upload the code in "The Sketch" on page 198 and connect the 9V battery pack to your Arduino to see your robot in action!

**FIGURE 22-10:** The circuit diagram for the ultrasonic robot



## THE SKETCH

The sketch starts by calling on the Adafruit Motor Shield, NewPing, and Servo libraries. The Trig pin of the ultrasonic sensor is defined as Arduino A4 and the Echo pin as Arduino A5. The maximum distance of the ultrasonic sensor is set at 200 centimeters and the speed of the DC motors is set at a medium speed of 190 (out of 255). The DC motors are defined to use connections M1 and M3 of the motor shield.

The servo is given a name and attached to pin 9 on the Arduino (via the connection on the motor shield). The loops after that take a reading from the ultrasonic sensor and, if it detects that an object is less than 15 centimeters away, the motors stop and reverse slightly, the servo moves left and right once to look around, and the robot turns to the left and continues to move forward until it discovers another object.

---

```
// Reproduced with kind permission from Nick Koumaris
// http://www.educ8s.tv
#include <AFMotor.h>
#include <NewPing.h>
#include <Servo.h>
#define TRIG_PIN A4
#define ECHO_PIN A5
#define MAX_DISTANCE 200
#define MAX_SPEED 190 // Sets speed of DC motors
```

```

#define MAX_SPEED_OFFSET 20

NewPing sonar(TRIG_PIN, ECHO_PIN, MAX_DISTANCE);
AF_DCMotor motor1(1, MOTOR12_1KHZ); // First motor to connection 1
AF_DCMotor motor2(3, MOTOR12_1KHZ); // Second motor to connection 3
Servo myservo; // Give the servo a name
boolean goesForward = false;
int distance = 100; // Define an int for distance and speed
int speedSet = 0;

void setup() {
  myservo.attach(9); // Servo attached to pin 9
  myservo.write(115); // Set servo at 115 degrees
  delay(2000);
  distance = readPing(); // Read the distance from the sensor
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
  distance = readPing();
  delay(100);
}
void loop() {
  int distanceR = 0;
  int distanceL = 0;
  delay(40);
  // If distance is less than 15 cm, carry out this function
  if (distance <= 15) {
    moveStop();
    delay(100);
    moveBackward();
    delay(300);
    moveStop();
    delay(200);
    distanceR = lookRight();
    delay(200);
    distanceL = lookLeft();
    delay(200);
    if (distanceR >= distanceL) {
      turnRight();
      moveStop();
    } else { // Or else carry on
      turnLeft();
      moveStop();
    }
  } else {
    moveForward();
  }
  distance = readPing();
}

```

```

}

int lookRight() { // The servo looks to the right
    myservo.write(50);
    delay(500);
    int distance = readPing();
    delay(100);
    myservo.write(115);
    return distance;
}

int lookLeft() { // The servo looks to the left
    myservo.write(170);
    delay(500);
    int distance = readPing();
    delay(100);
    myservo.write(115);
    return distance;
    delay(100);
}

int readPing() {
    delay(70);
    int cm = sonar.ping_cm();
    if (cm == 0) {
        cm = 250;
    }
    return cm;
}

void moveStop() {
    motor1.run(RELEASE);
    motor2.run(RELEASE);
}

void moveForward() {
    if (!goesForward) { // If area is clear, motors move forward
        goesForward = true;
        motor1.run(FORWARD);
        motor2.run(FORWARD);
        // Slowly bring up speed to avoid loading down
        // batteries too quickly
        for (speedSet = 0; speedSet < MAX_SPEED; speedSet += 2) {
            motor1.setSpeed(speedSet);
            motor2.setSpeed(speedSet + MAX_SPEED_OFFSET);
            delay(5);
        }
    }
}

```

```

void moveBackward() {
    goesForward = false;
    motor1.run(BACKWARD);
    motor2.run(BACKWARD);
    // Slowly bring up speed to avoid loading down
    // batteries too quickly
    for (speedSet = 0; speedSet < MAX_SPEED; speedSet += 2) {
        motor1.setSpeed(speedSet);
        motor2.setSpeed(speedSet + MAX_SPEED_OFFSET);
        delay(5);
    }
}

void turnRight() { // Movement for turning right
    motor1.run(FORWARD);
    motor2.run(BACKWARD);
    delay(300);
    motor1.run(FORWARD);
    motor2.run(FORWARD);
}

void turnLeft() { // Movement for turning left
    motor1.run(BACKWARD);
    motor2.run(FORWARD);
    delay(300);
    motor1.run(FORWARD);
    motor2.run(FORWARD);
}

```

---

## TROUBLESHOOTING

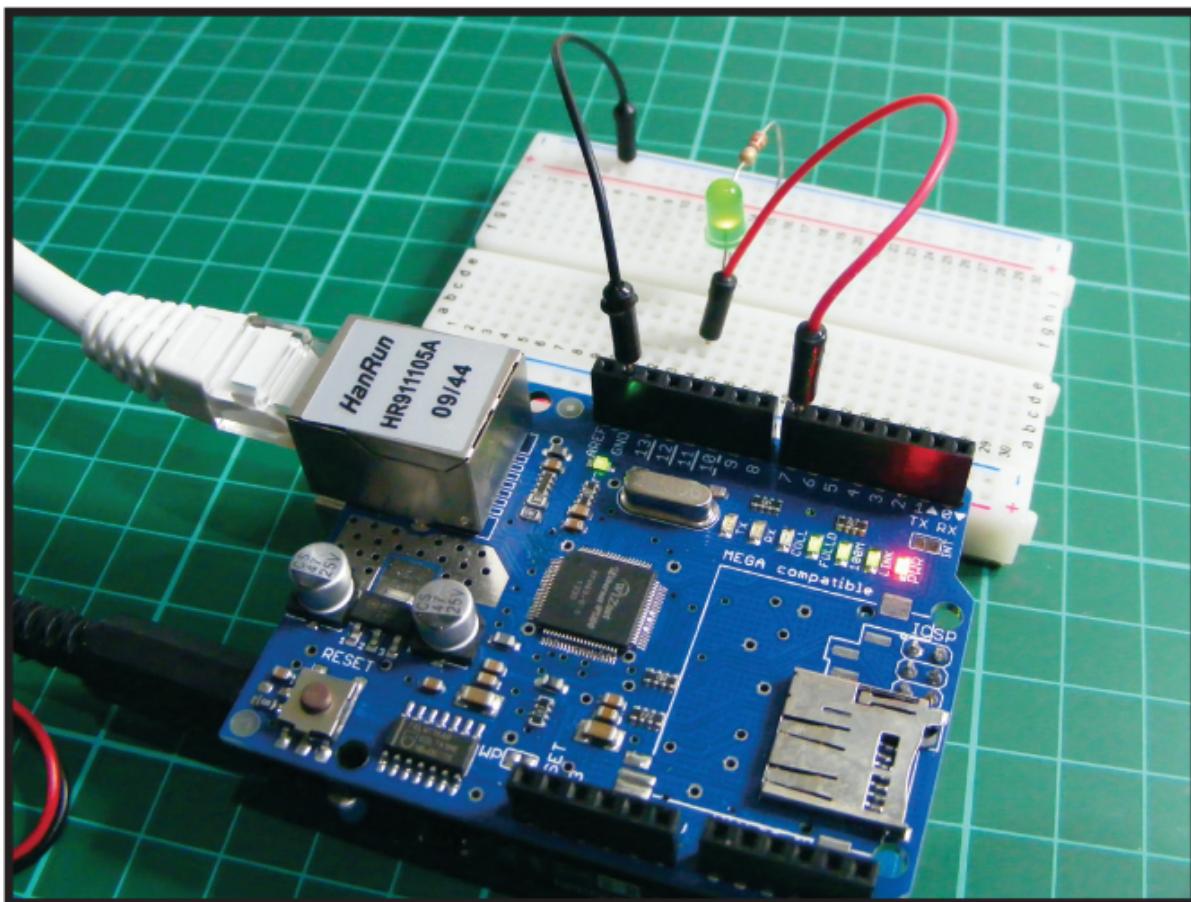
**Q.** *The code compiles, but the Arduino robot does not function as expected.*

- Make sure that your wiring matches the tables in steps 7, 8, and 9 and the circuit diagram in Figure 22-10.
- If your robot spins around rather than moving forward, reverse the wiring on one of the DC motors—as mentioned earlier, they don’t have polarity but changing the power connections will reverse the motor’s rotation.
- Power the robot with a pack of 1.5V AA batteries in series rather than a 9V battery, which has less amperage and will drain quicker.

**23**

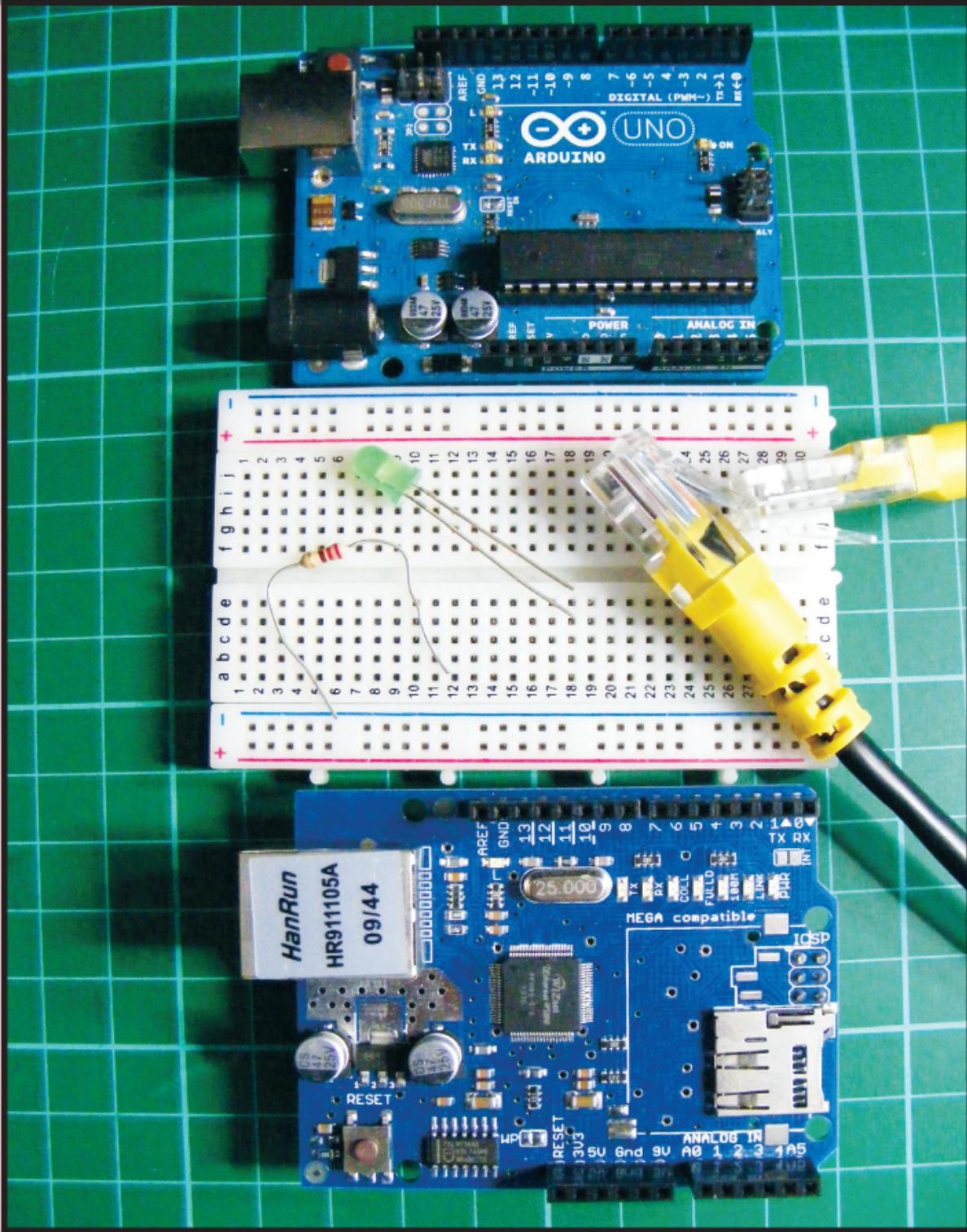
## Internet-Controlled LED

In this project we'll use an ethernet shield to connect our Arduino to the internet and control an LED from a web browser.



**COST: \$\$\$**

**TIME: 30 MINUTES**



## **PARTS REQUIRED**

**Arduino board**  
**Breadboard**  
**Jumper wires**  
**Ethernet shield W5100 LAN expansion board**  
**Ethernet cable**  
**LED**  
**220-ohm resistor**

## **LIBRARIES REQUIRED**

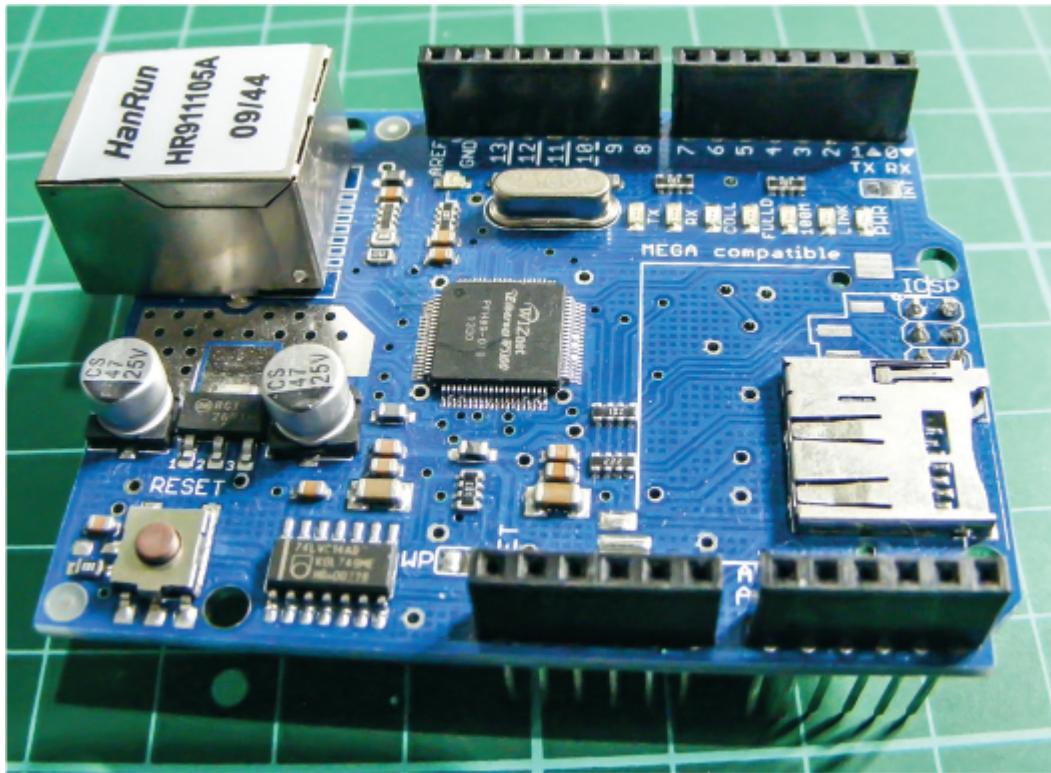
**SPI**  
**Ethernet**

The *Internet of Things (IoT)* is revolutionizing our use of everyday items. The term refers to objects or smart devices connected through a network, usually involving the internet. This allows us to control devices remotely, from inside or outside the house! Amazon Echo and Google Home are taking things further by allowing a multitude of devices to be connected and controlled via a central hub, even if you aren't at home. We'll create our own IoT project in its most basic form to demonstrate the principles involved.

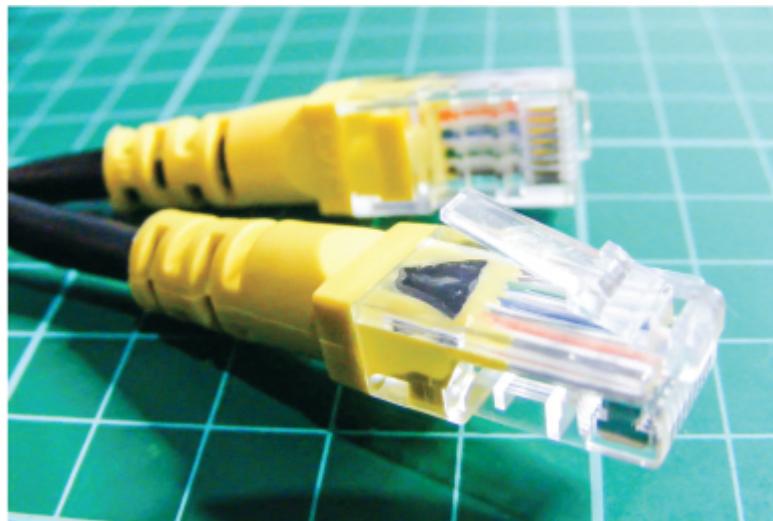
## **HOW IT WORKS**

The Ethernet shield W5100 LAN expansion board, shown in Figure 23-1, fits directly on top of the Arduino to provide additional functionality to the board. We'll use the Ethernet library built into the Arduino IDE to connect our board to the internet via an Ethernet cable, as shown in Figure 23-2.

**FIGURE 23-1:** Ethernet shield



**FIGURE 23-2:** Ethernet cable



The library allows the Arduino to act as a server to accept incoming commands, a client to send them out, or both. The shield communicates with the Arduino using the *Serial Peripheral Interface (SPI)* connections. On the Arduino Uno the SPI connections are on digital pins 10, 11, 12, and 13. In our project the Arduino will use both

functions to send information to the internet in the form of a simple web page and accept commands from this page to control an LED. Buttons on the web page will allow us to switch the LED on or off as long as the Arduino is powered and connected to the internet.

## SETTING UP YOUR ETHERNET CONNECTION

You need to know the MAC address of your shield for this project to work. A *MAC address* is a unique number assigned to devices for communication and is used as a network address for Ethernet and Wi-Fi. If you have a newer shield, the MAC address will be printed on a product sticker. If you have an older generic Ethernet shield like the one we are using, you can use the MAC address 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED for this project.

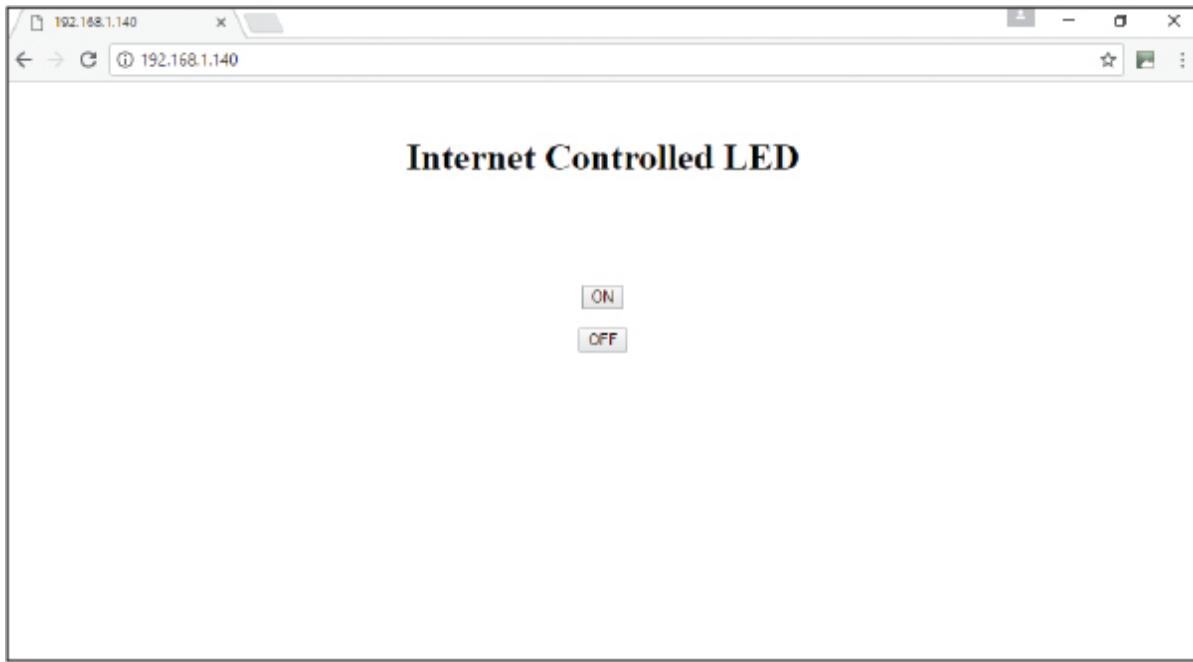
For communication, we'll use port 80, the default for HTTP. Short for HyperText Transfer Protocol, HTTP is the set of rules for transferring data over the internet. In this instance port 80 handles the transfer of data to a web page.

Our sketch includes some HTML (HyperText Markup Language) code, which tells a web browser how to display an internet page. If you right-click on any web page and select Inspect, you can see some of the HTML code behind that page.

The section of our sketch that includes HTML code is as follows and produces the web page displayed in Figure 23-3.

```
-----  
client.println("HTTP/1.1 200 OK");  
client.println("Content-Type: text/html");  
client.println();  
client.print("<center><br><h1>Internet Controlled LED</h1><br><br><br><FORM>");  
client.print("<P> <INPUT type=\"submit\" name=\"status\" value=\"ON\">");  
client.print("<P> <INPUT type=\"submit\" name=\"status\" value=\"OFF\">");  
client.print("</FORM></center>");  
-----
```

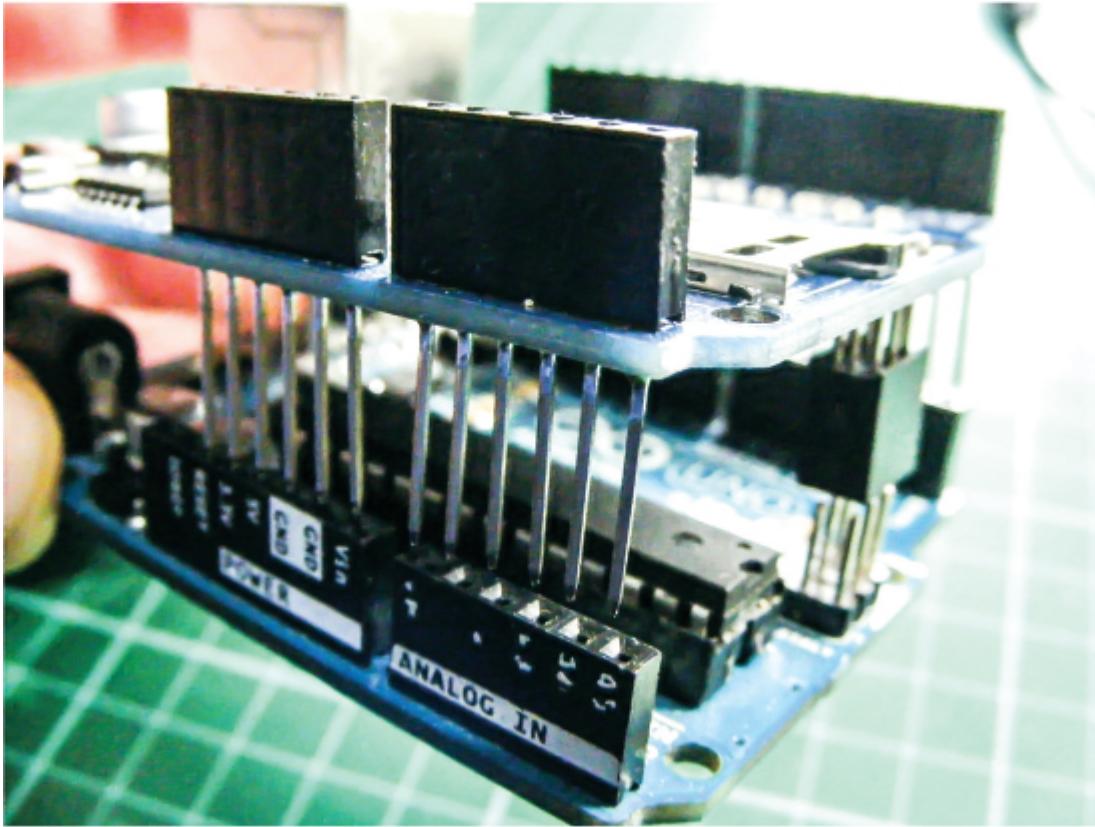
**FIGURE 23-3:** Our simple web page to control the LED



## THE BUILD

1. Attach the Ethernet shield on top of the Arduino board as shown in Figure 23-4. The board fits directly on top of the Arduino, so gently press the legs of the shield in place with the holes of the Arduino beneath.

**FIGURE 23-4:** Attach the Ethernet shield on top of the Arduino board.



2. Insert the LED into the breadboard with the legs straddling the center break of the board. Then, as shown in the following table, connect the shorter, negative leg of the LED to the GND rail of the breadboard via a 220-ohm resistor, and connect the longer, positive leg of the LED to pin 7 on the Arduino/Ethernet shield. Connect the GND rail of the breadboard to Arduino GND.

LED	ARDUINO
Negative leg	GND via 220-ohm resistor
Positive leg	Pin 7

3. With the Ethernet shield attached on top of the Arduino, connect the shield to your router with the Ethernet cable.

**NOTE**

*Take note of your IP address; it will be different from mine shown in Figure 23-5.*

4. Attach the Arduino to your PC and upload the code at the end of the project using the IDE. Once the code is uploaded, open the IDE Serial Monitor in order to ascertain the *IP address*—a unique string of numbers to identify a device attached to the internet—of the Arduino, which is acting as our server. You should see something similar to Figure 23-5.

**FIGURE 23-5:** The IP address of the Arduino shield will be shown in the Serial Monitor.



5. Open any web browser and enter your IP address. You should see a web page with an On and an Off button, as shown earlier in Figure 23-3. Press the On button to light the LED and press the Off button to switch it off.
6. This project will also work when you are not connected to your local network, as long as you have port 80 open on your internet

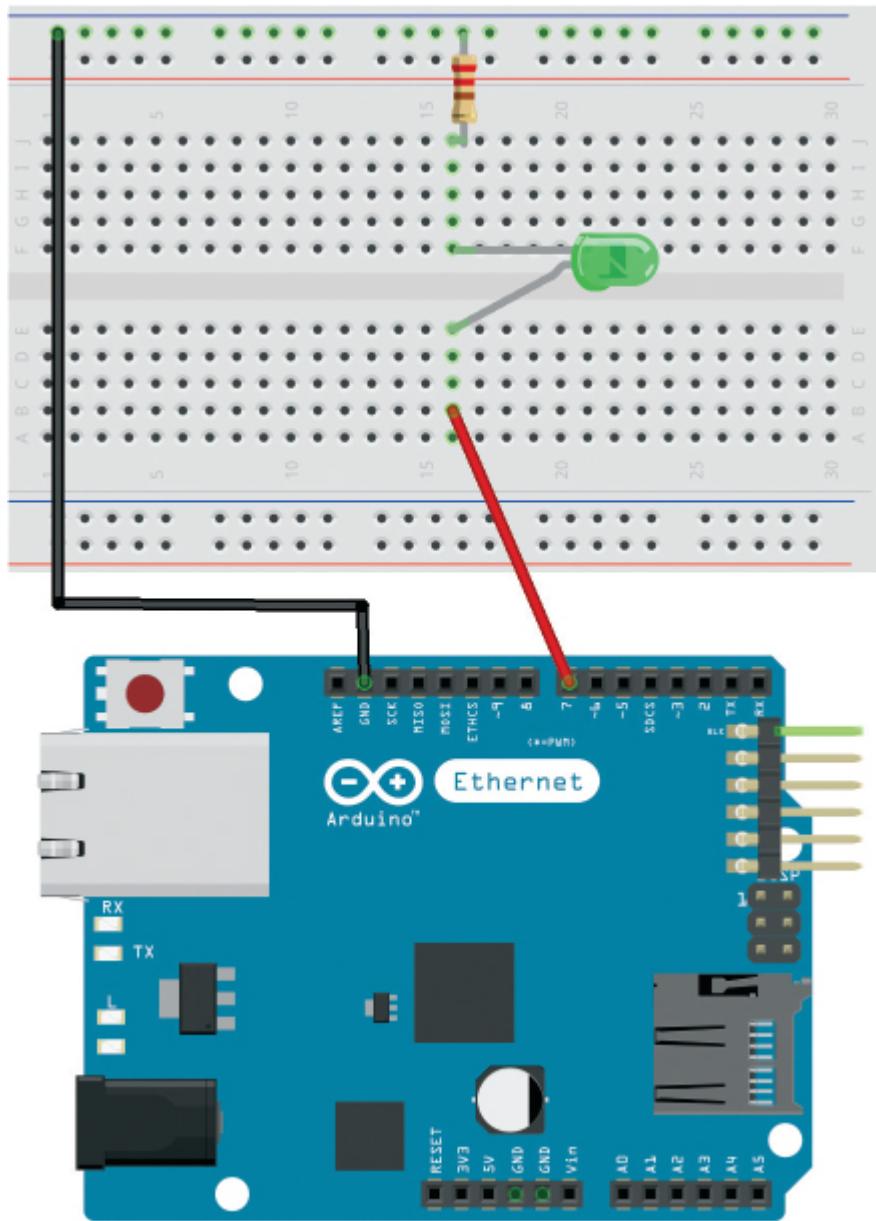
router. Many internet service providers (ISPs) have this port blocked for security reasons, so follow the instructions from your ISP to change this if required.

**WARNING**

*Carry out this function only if you are aware of the security risks and how to minimize them.*

7. Confirm that your setup matches the circuit diagram in Figure 23-6, and then upload the code in “The Sketch” on page 208.

**FIGURE 23-6:** The circuit diagram for the internet-controlled LED



## THE SKETCH

The sketch calls on the SPI and Ethernet libraries to control communication with the internet. We define the MAC address for the shield. This is the line you need to change if your shield came with its own MAC address; if not, the address given earlier in this project and shown in the code should work for you. We then set the server to use port 80 and define pin 7 on the Arduino as the LED pin.

The setup defines the LED pin as an output, begins the Ethernet shield, and starts serial communication so we can see the IP address of our server. The loop sets up our web page to the browser once it is called and waits for an input on the browser task bar. When the On button is pressed, the server tells the Arduino to set the LED pin as HIGH and the LED will light. When the Off button is pressed, the power to the LED is LOW and the LED will turn off.

You could easily change the LED for a relay switch such as the one used in Project 12 to control a larger-voltage device.

---

```
#include <SPI.h>
#include <Ethernet.h>

// MAC address for shield
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
EthernetServer server(80); // Using port 80
int led = 7; // LED attached to pin 7

void setup() {
    pinMode(led, OUTPUT); // LED set as an output
    Ethernet.begin(mac); // Start the Ethernet shield
    server.begin();
    Serial.begin(9600); // Start serial communication
    Serial.println("Server address:"); // Print server address
                                         // (Arduino shield)
    Serial.println(Ethernet.localIP());
}

void loop() {
    EthernetClient client = server.available();
    if (client) {
        boolean currentLineIsBlank = true;
        String buffer = "";
        while (client.connected()) {
            if (client.available()) {
                char c = client.read(); // Read from the Ethernet shield
                buffer += c; // Add character to string buffer
                // Client sent request, now waiting for response
                if (c == '\n' && currentLineIsBlank) {
                    client.println("HTTP/1.1 200 OK"); // HTTP response
                    client.println("Content-Type: text/html");
                    client.println(); // HTML code
                    client.print("<center><br><h1>Internet Controlled LED</h1><br><br><FORM>");
                    client.print("<P> <INPUT type=\"submit\" name=\"status\" value=\"ON\">");
                    client.print("<P> <INPUT type=\"submit\" name=\"status\" value=\"OFF\">");
                }
            }
        }
    }
}
```

```

        client.print("</FORM></center>");
        break;
    }
    if (c == '\n') {
        currentLineIsBlank = true;
        buffer = "";
    }
    else if (c == '\r') { // Command from webpage
        // Did the on button get pressed
        if (buffer.indexOf("GET /?status=ON") >= 0)
            digitalWrite(led, HIGH);
        // Did the off button get pressed
        if (buffer.indexOf("GET /?status=OFF") >= 0)
            digitalWrite(led, LOW);
    }
    else {
        currentLineIsBlank = false;
    }
}
client.stop(); // End server
}
-----
```

## TROUBLESHOOTING

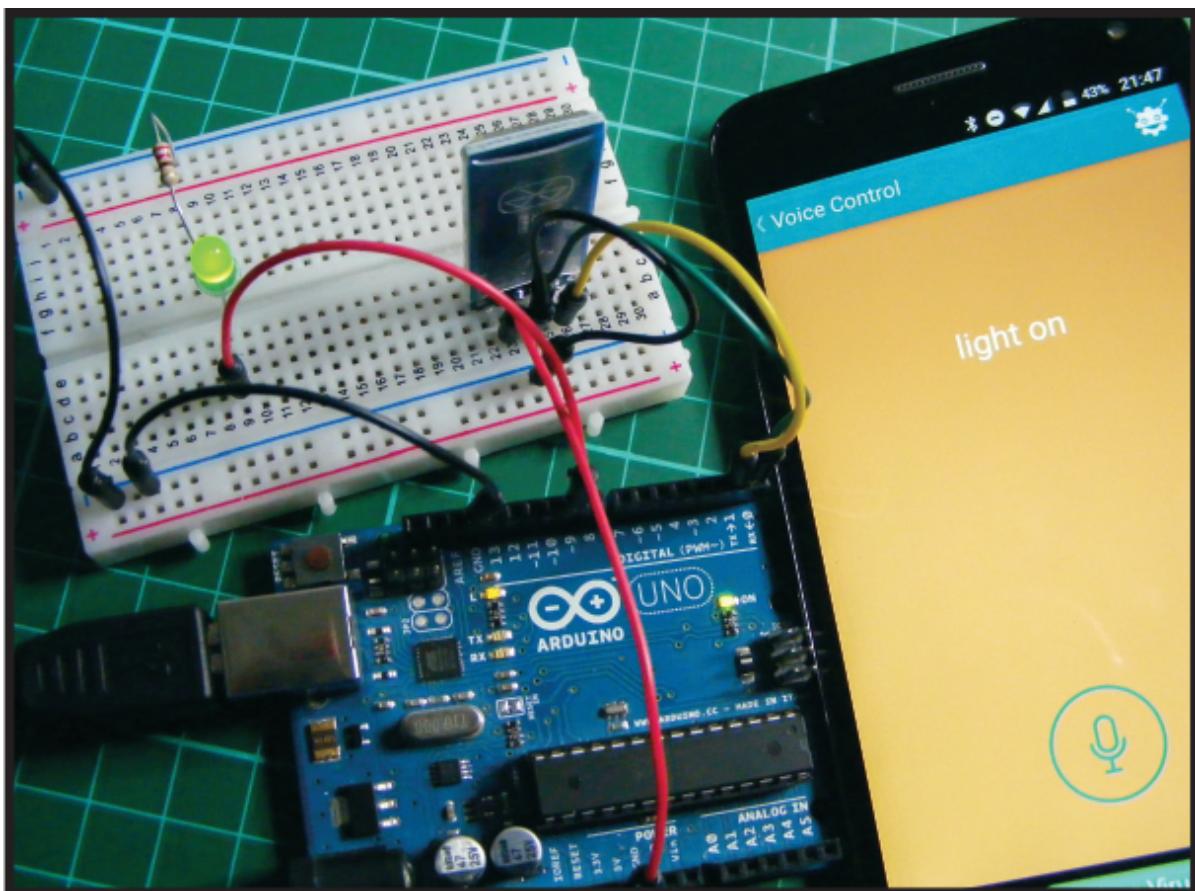
- Q.** *The code compiles, but the LED does not light as expected.*
- First, make sure you've connected the GND wire from the Arduino to the correct breadboard power rail and that the Arduino has power connected.
  - Check that the resistor is inserted fully and lines up with the corresponding LED leg.
  - Try checking that the project is working by connecting to your local area network and using that PC to connect to the Arduino.
- Q.** *You receive an error when calling the web page.*
- Make sure you have entered the IP address of the server exactly as you read it in the steps given earlier.
  - It is best to check the project is working by connecting to your local area network and using that PC to connect to the Arduino.

- If the project worked when you connected it to your local area network, but you receive an HTTP 403 error when connecting to the internet externally, then your ISP is blocking incoming traffic. You could add *port forwarding* to your router for port 80. This will differ for every device, so check with your ISP for detailed instructions. Do a quick internet search with your ISP and “port forwarding” as terms and follow the instructions, but be aware: this can compromise the security of your PC and should be done only if you understand the risks and are able to protect your network.

24

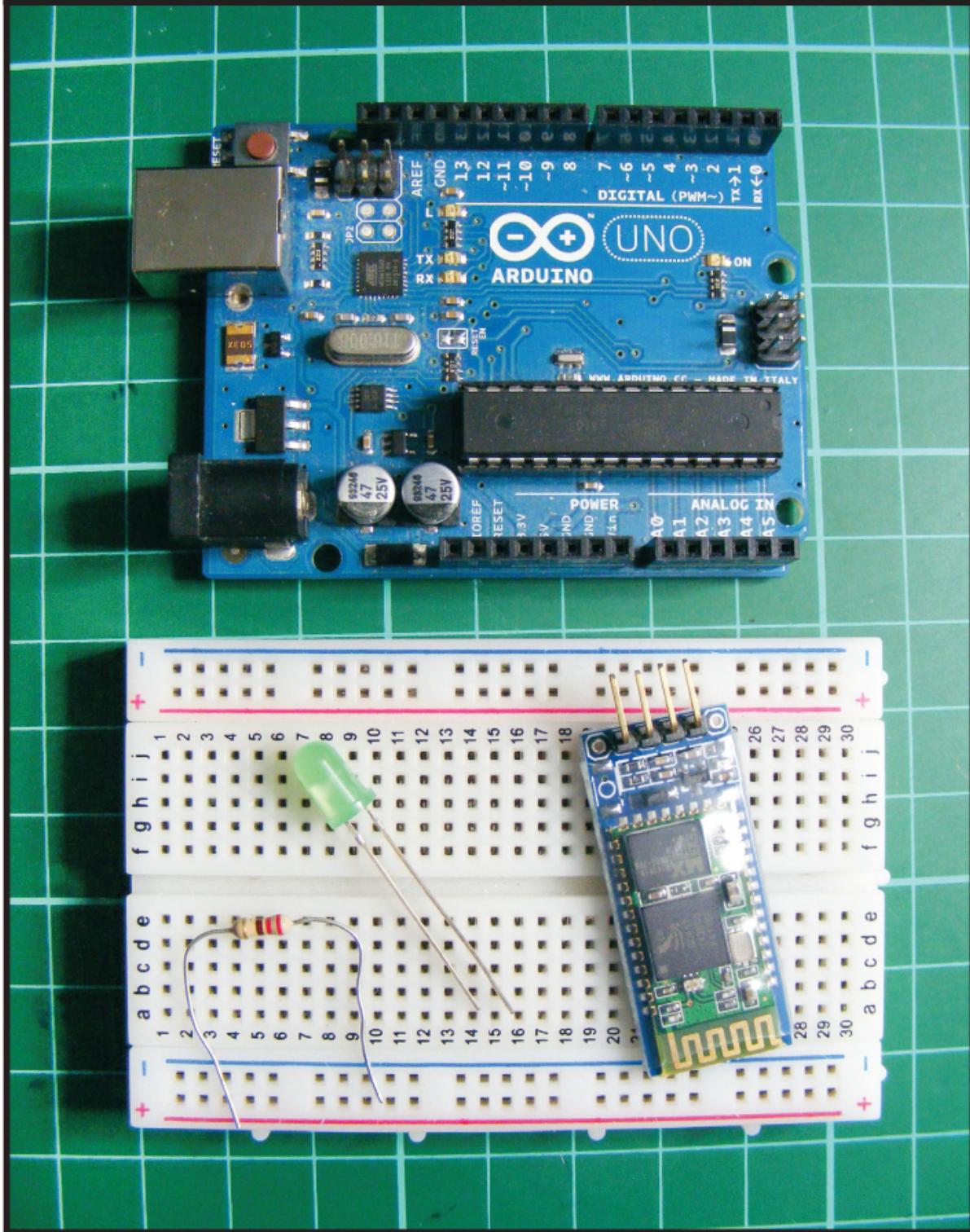
## Voice-Controlled LED

In this project we'll use a bluetooth module, a smartphone, and a voice recognition app to control an LED with vocal commands.



**COST: \$\$**

**TIME: 30 MINUTES**



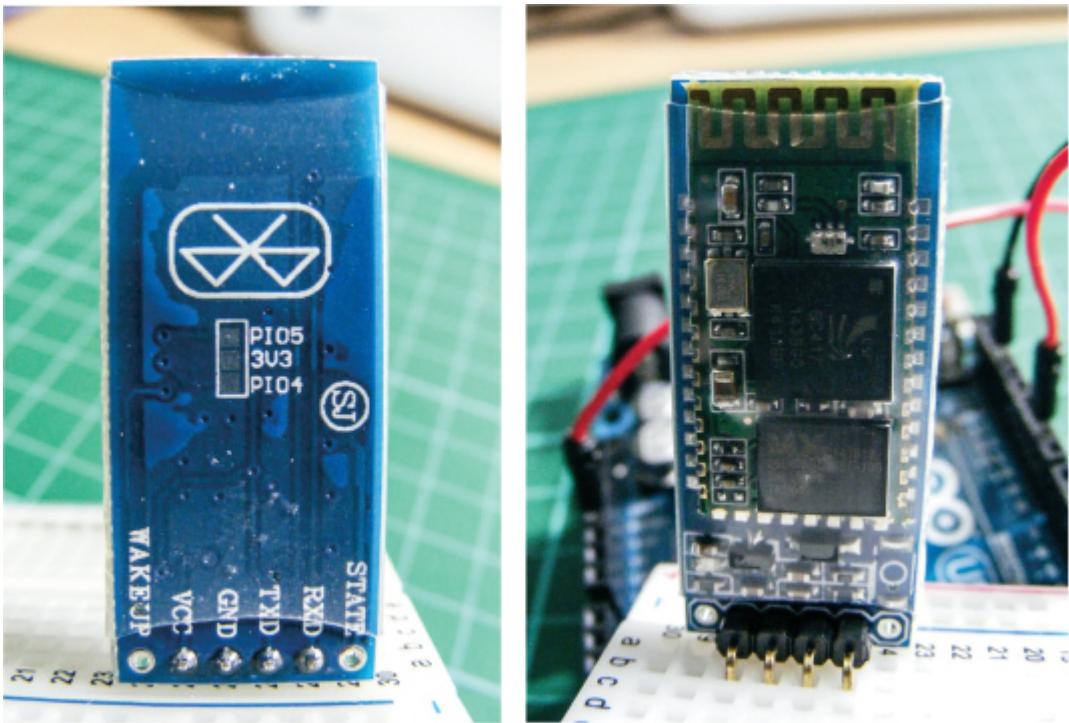
## **PARTS REQUIRED**

**Arduino board**  
**Breadboard**  
**Jumper wires**  
**HC-06 Bluetooth module**  
**LED**  
**220-ohm resistor**  
**Android smartphone**

## HOW IT WORKS

Bluetooth wireless technology uses radio waves to transmit and exchange data over short distances. Smartphones, laptops, and multimedia devices such as speakers use Bluetooth as a common standard. We'll use the inexpensive HC-06 Bluetooth module (Figure 24-1) to pair (connect) our Arduino to a smartphone so we can turn an LED on and off remotely using a voice recognition app. This module has six pins, but we'll just use the middle four. The pins should be labeled on the front.

**FIGURE 24-1:** The HC-06 Bluetooth module



The app we'll use is Arduino Bluetooth Control from BroxCode, available to download for free on the Google Play store for Android devices. There are many other similar free apps available for both Android and Apple devices and the principles of use should be the same for each, but the BroxCode app has some additional features that we're using in this project, such as voice recognition through Google Assistant.

## THE BUILD

Before you build the Bluetooth controller, you need to upload the code to the Arduino. This is because the serial communication from your PC to the Arduino uses the same pins that we'll be connecting to the Bluetooth module.

1. Upload the code in “The Sketch” on page 220, and then insert the Bluetooth module into the breadboard and connect VCC to the positive power rail of the breadboard, GND to the GND power rail, TXD to Arduino pin 0 (RX), and RXD to Arduino pin 1 (TX), as shown in the following table.

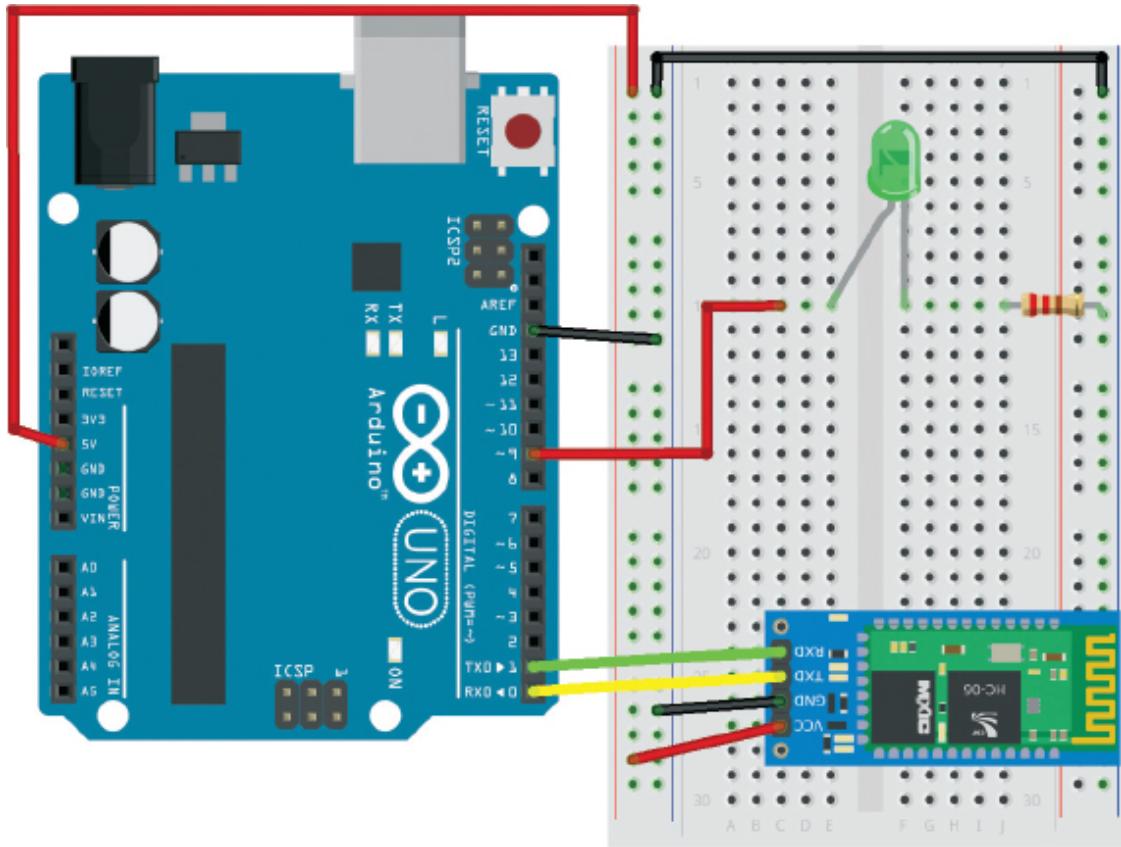
HC-06 BLUETOOTH MODULE	ARDUINO
VCC	+5V
GND	GND
TXD	Pin 0 (RX)
RXD	Pin 1 (TX)

2. Insert the LED into the breadboard with the legs straddling the center break. Use a 220-ohm resistor to connect the shorter, negative leg of the LED to the GND rail of the breadboard. Connect the longer, positive leg of the LED to pin 9 of the Arduino using a jumper wire, as outlined in the following table.

LED	ARDUINO
Positive leg	Pin 9
Negative leg	GND

3. Connect the GND rail of the breadboard to Arduino GND and the positive rail to Arduino +5V.  
 4. Check that your build matches the diagram in Figure 24-2.

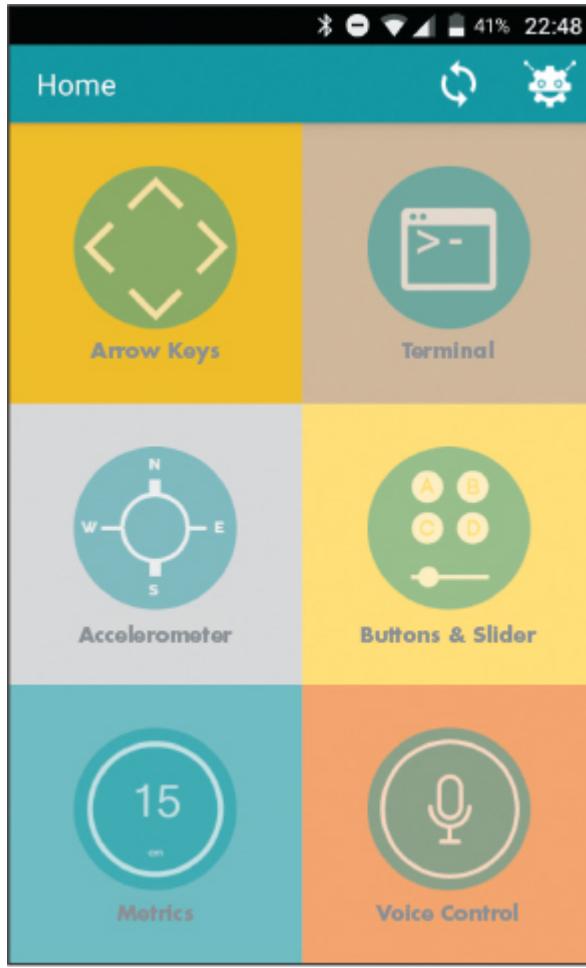
**FIGURE 24-2:** The circuit diagram for the Bluetooth voice-controlled LED



## ARDUINO BLUETOOTH CONTROL

The Arduino Bluetooth Control app offers six control options, all of which send data to the Arduino via different methods (Figure 24-3). You can customize each to your own preferences.

**FIGURE 24-3:** The menu screen on the Arduino Bluetooth Control app



- **Arrow Keys:** Here you'll find customizable arrow buttons.
- **Terminal:** This is a classic terminal for sending and receiving data, displayed with a timestamp corresponding to each action.
- **Accelerometer:** This tool reads movement using the gesture sensor of your phone.
- **Buttons and Slider:** Here you'll find six fully customizable buttons and a slider view that shows up when you rotate your device. You can set the range of the data for this slider.
- **Metrics:** This tool is optimized to receive data via the `println()` function of the Arduino, which allows your paired phone to receive notifications by SMS from another phone. You only need to specify

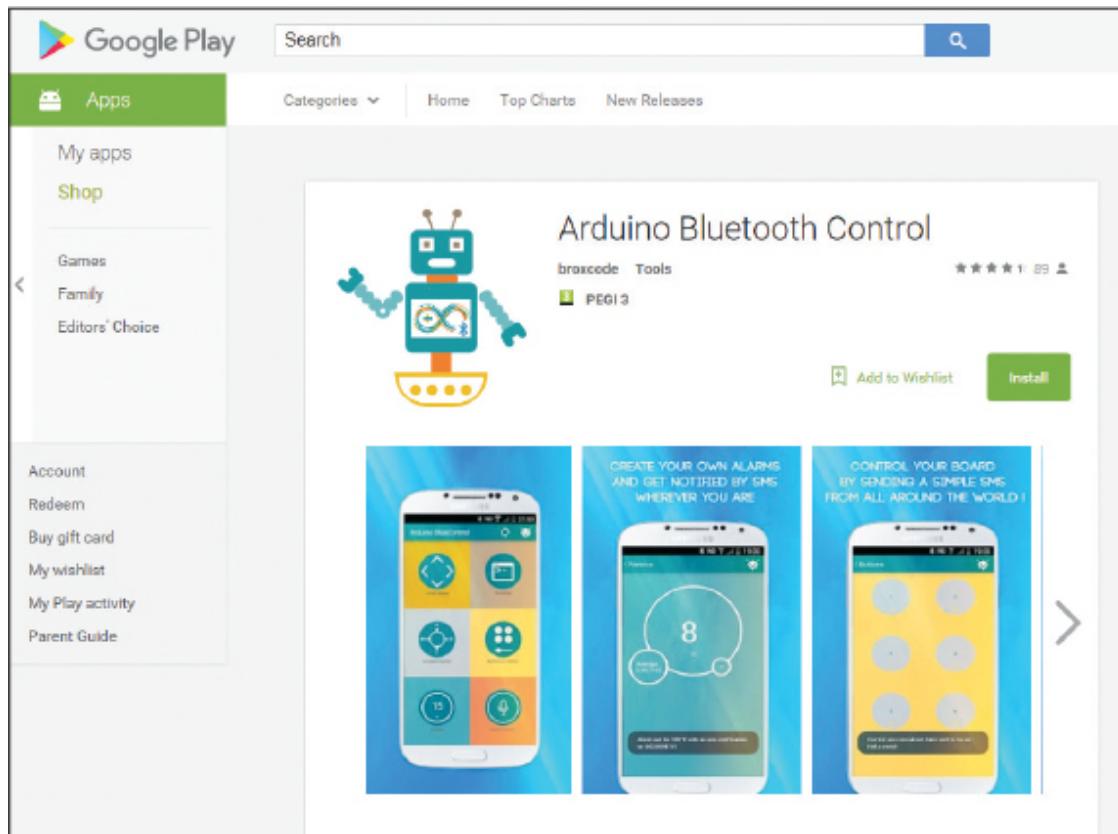
the number in the Settings section. This function is explained further shortly.

- **Voice Control:** This great tool uses the Google voice command on your Android device to let you customize your own vocal commands and use them to control the Arduino.

Now you need to download the Arduino Bluetooth Control app from the Google Play app store and set it up.

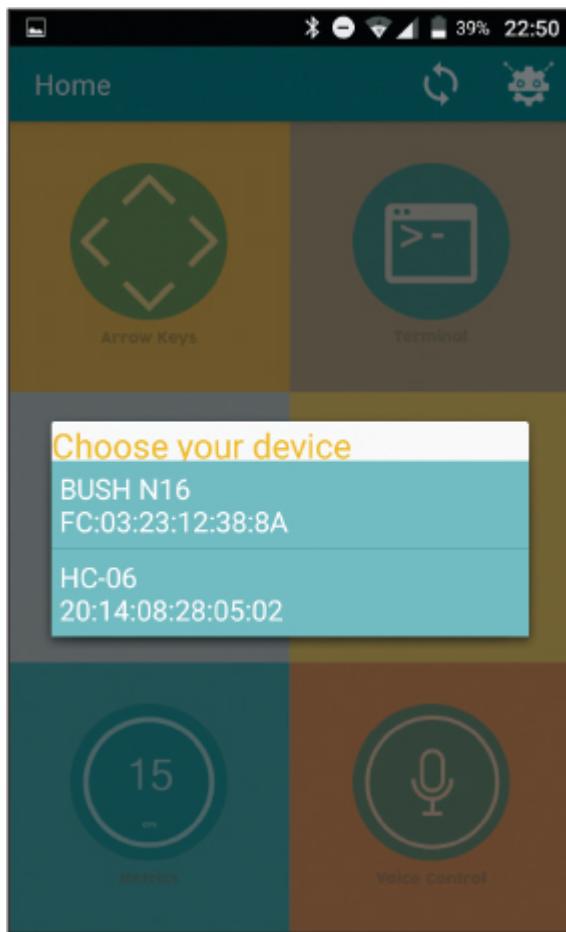
1. Go to <https://play.google.com/store/> and search for “Arduino Bluetooth Control.” You’ll probably get several apps in your results, but the one you want is precisely named “Arduino Bluetooth Control,” as shown in Figure 24-4. Click **Install** to download it to your device. The app is free but does include some ads.

**FIGURE 24-4:** Arduino Bluetooth Control from BroxCode on Google Play



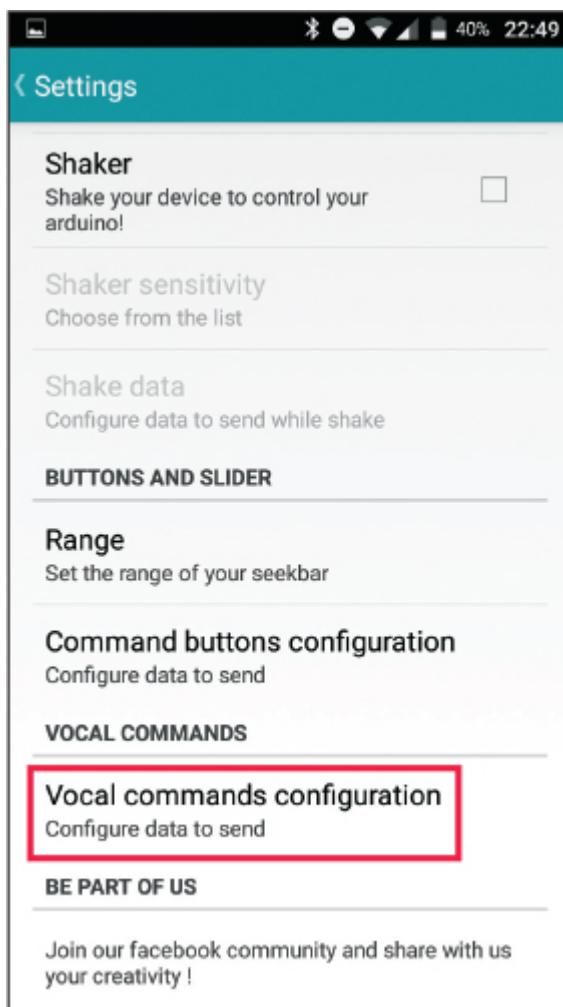
2. Once you've downloaded the app, power your Arduino to start the Bluetooth module. Go to your Bluetooth settings on your smartphone, turn on Bluetooth, and select **MORE SETTINGS** to view visible devices. You should see the HC-06 module as an available device. Select it to pair with your phone. You'll be asked for a password to connect: the default for the module is 1234 or in some instances 0000, so try both if the first doesn't work.
3. When your device is paired, open the Arduino Bluetooth Control app. From the window that appears showing all available devices, select the HC-06 module, as shown in Figure 24-5. You won't need to choose the device every time you power up—the app will remember it.

**FIGURE 24-5:** Pairing your device



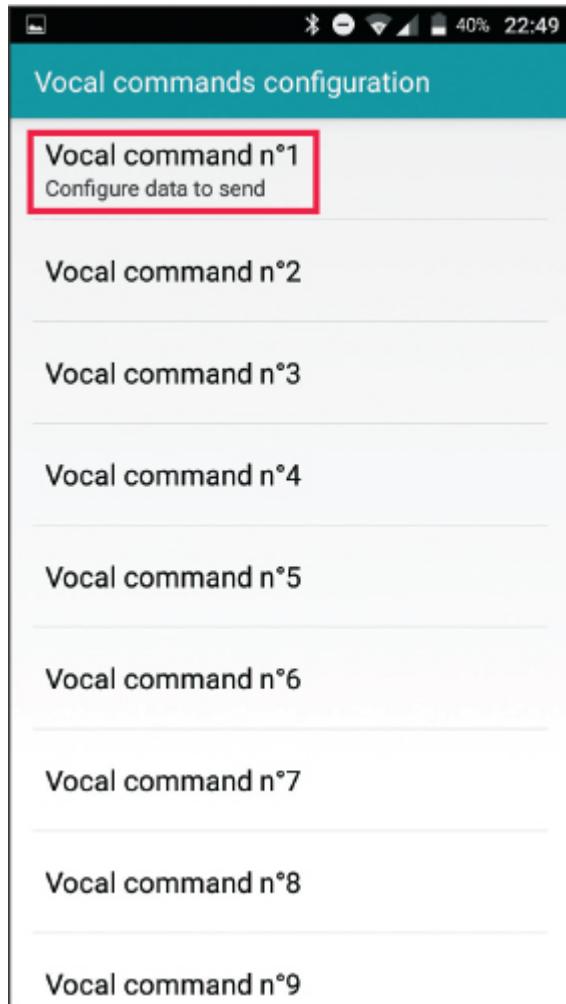
4. You're going to use the Voice Control function to turn the LED off and on when you speak certain commands into the smartphone. Select the Voice Control function, and you'll be taken to the Settings menu, shown in Figure 24-6. Choose **Vocal commands configuration**. We'll use this to define our input and output functions.

**FIGURE 24-6:** Selecting the Vocal commands configuration setting



5. Select **Vocal command n°1**, as shown in Figure 24-7.

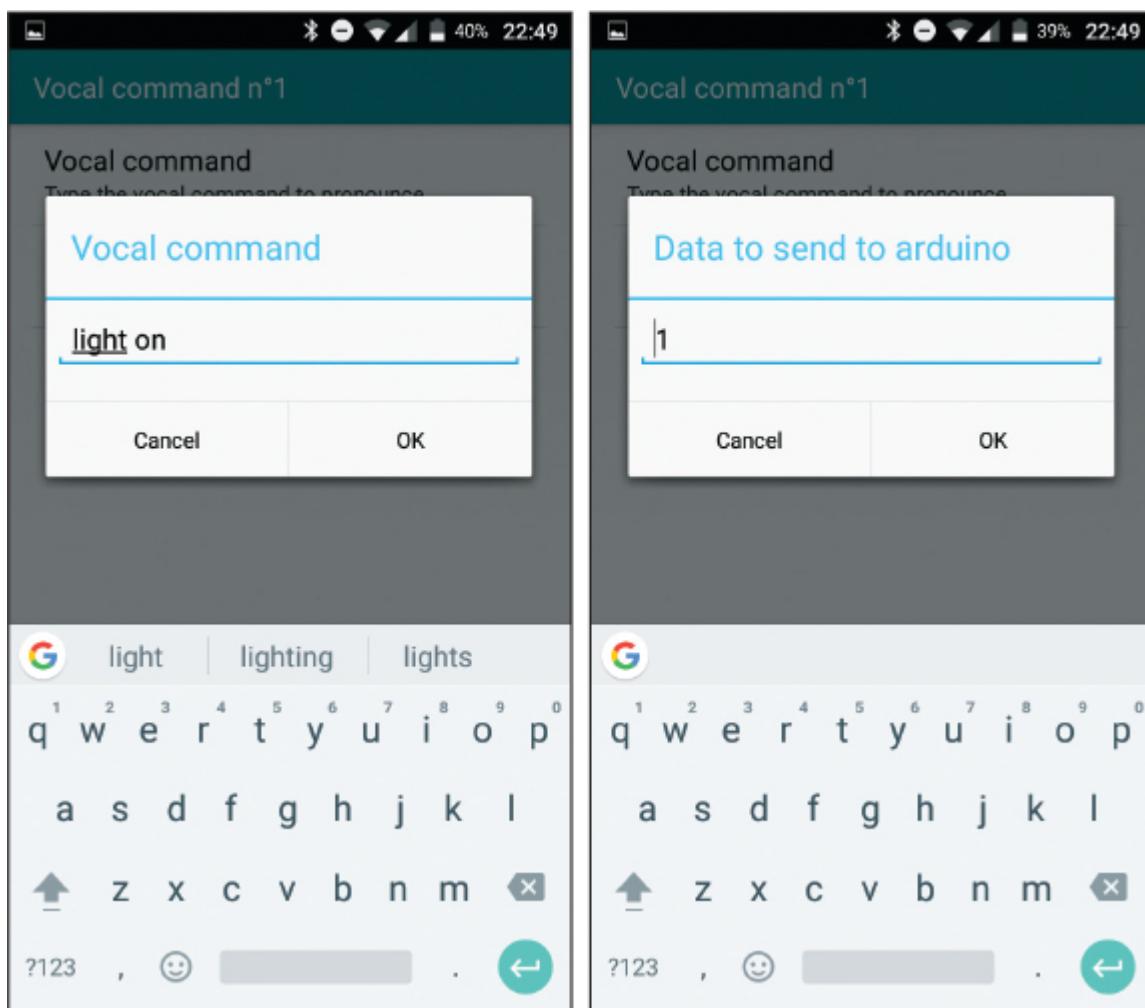
**FIGURE 24-7:** Setting your first voice command



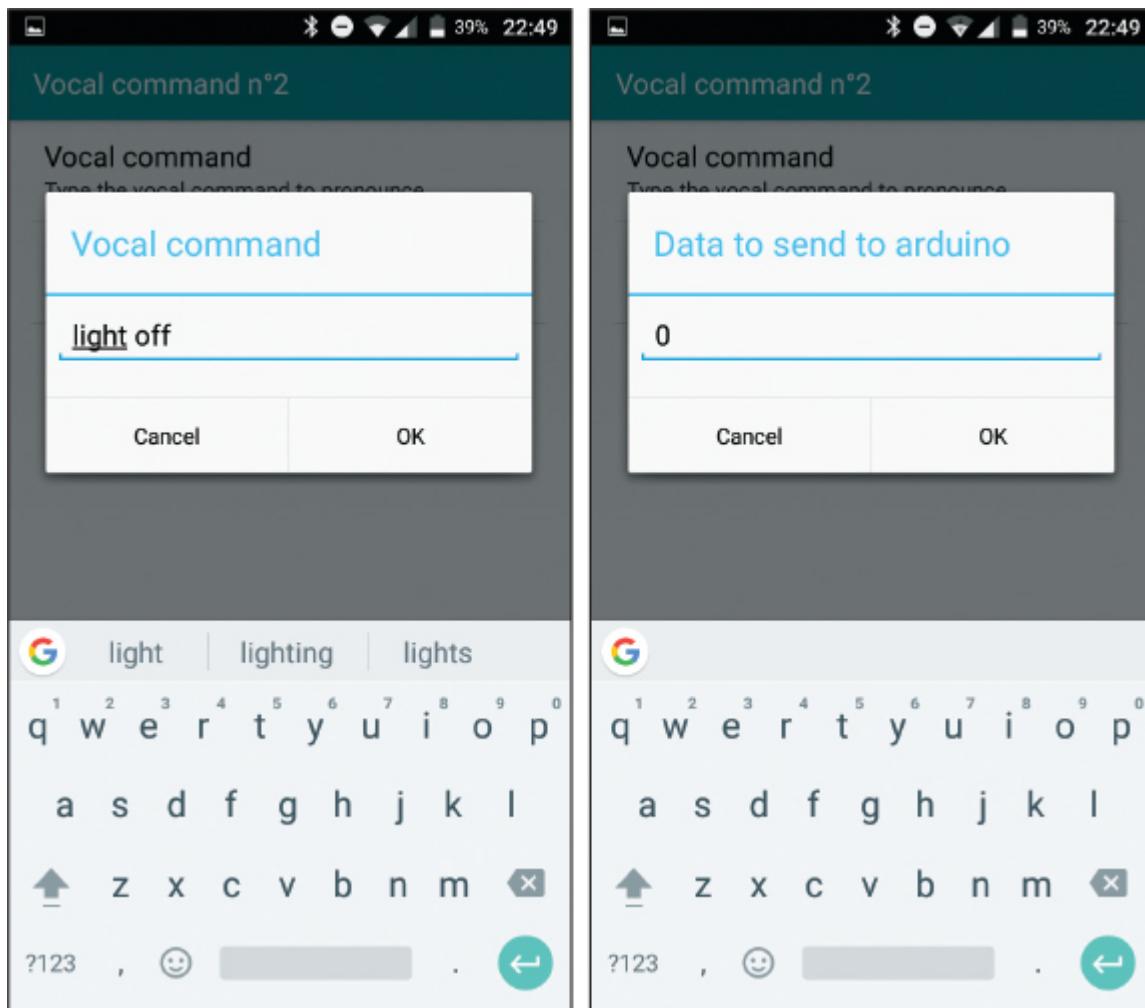
6. Here you give the input that will trigger the first function. Enter **light on** as text, as shown in the screen on the left in Figure 24-8. The app will then ask for the output data to send to the Arduino when you give the input command. On this screen, enter **1** for on or **HIGH**, as we've seen in previous LED projects (shown in the screen on the right in Figure 24-8). When the app hears the vocal command "light on" through the phone, the number **1** will be sent to the Arduino as an input, and power will be sent to the LED to light it up.
7. Carry out the same steps to define Vocal command n°2 with the input **light off** and the output data **0**, as shown in Figure 24-9. This command will switch the LED off.

Now you've configured your commands so that when you press the voice command function and tap the microphone button on the screen, the app will listen for your command and, depending on the input, switch the LED on or off.

**FIGURE 24-8:** Configuring our LED to turn on with the voice command "light on"



**FIGURE 24-9:** Configuring the "light off" function



The app also has a function to let you control the Arduino using SMS. Once the app is launched and connected to the Arduino, you can send data to the Arduino by sending an SMS text to the phone paired with the Bluetooth module, as long as the paired phone is in range of the module. Simply text **Arduino 1** to the phone connected to the Arduino, and that phone will send 1 to the module to light your LED. Text **Arduino 0**, and a 0 will be sent to switch your LED off. This way you can have control through Bluetooth from anywhere in the world!

## THE SKETCH

The sketch for this project is quite simple. It starts by creating a variable to hold the data from the Bluetooth module. It sets the data rate for serial communication to 9600 and sets pin 9 as an output for our

LED. In the loop, it checks for data to be sent to the Arduino from the Bluetooth module. The loop reads the data, and also sends it to the Serial Monitor so we can check that it's working correctly. If the Arduino receives a 1 from the app, pin 9 will be set to HIGH, which will turn on the LED. If the Arduino receives a 0, pin 9 is set as LOW and the LED is turned off.

Using these principles, you could add numerous relays in place of the LED and begin to automate your home from anywhere. You could set it up to turn on your living room lights before you enter your house, set the thermostat when you're on your way home, or have your favorite music already playing as you walk in the door.

---

```
char data = 0; // Create a variable for data
void setup() {
    Serial.begin(9600); // Data rate for serial communication
    pinMode(9, OUTPUT); // Set pin 9 as an output
}
void loop() {
    if (Serial.available() > 0) { // Send data
        data = Serial.read(); // Read incoming data and
                               // store it into variable data
        Serial.print(data); // Print data value to the Serial Monitor
        Serial.print("\n"); // Start a new line
        if (data == '1') // If value is 1, turn on LED
            digitalWrite(9, HIGH);
        else if (data == '0') // If value is 0, turn off LED
            digitalWrite(9, LOW);
    }
}
```

---

## TROUBLESHOOTING

**Q.** *The code compiles, but the LED does not light.*

- Make sure you've connected the GND and power pins from the Arduino to the correct breadboard power rails and that the Arduino has power connected.
- Check that the LED is inserted the correct way, with the longer leg connected to the positive power and the shorter leg to GND. Check

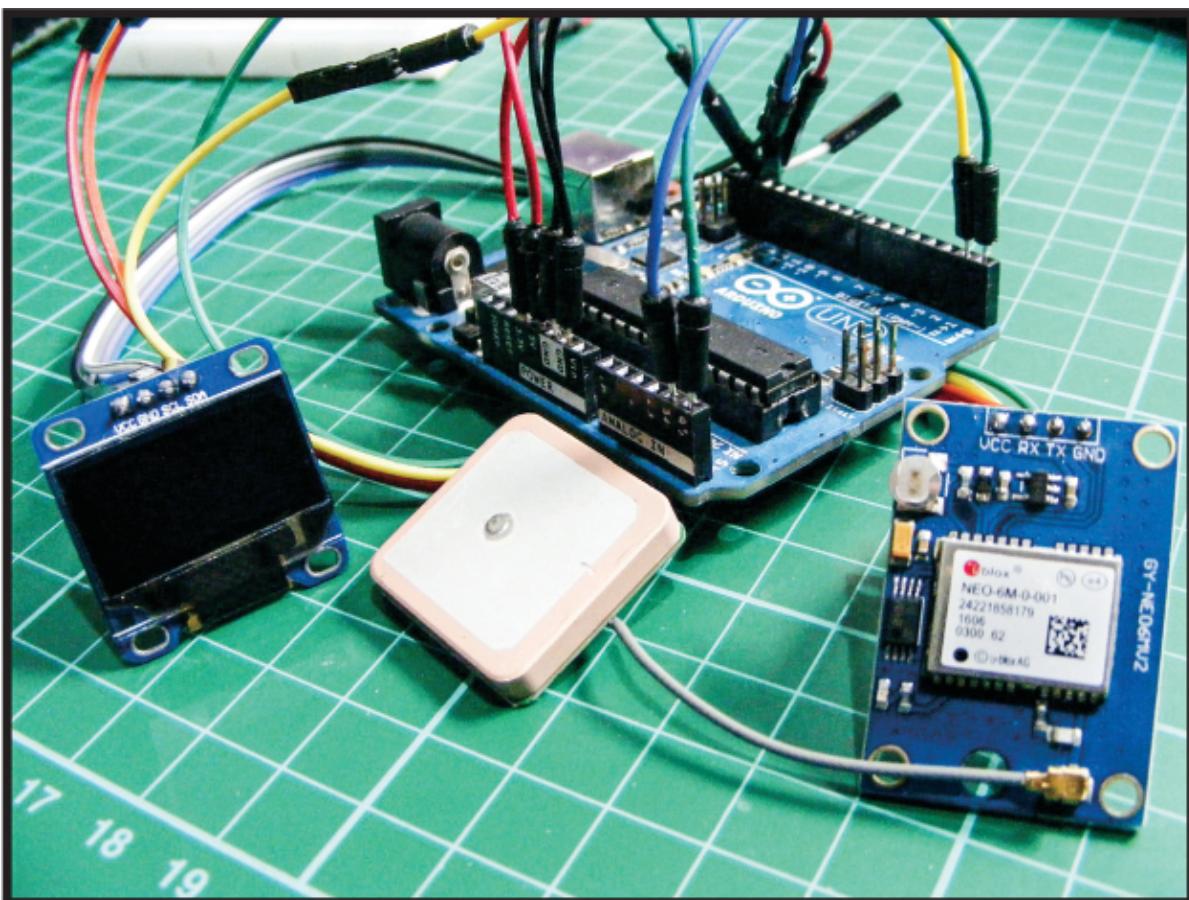
that the resistors are inserted fully and line up with the corresponding LED leg.

- With the project powered and connected to your PC, open the Arduino IDE Serial Monitor to see if the Arduino is receiving data from the app. If you don't see data streaming in the Serial Monitor, double-check that the TXD of the module is connected to RX of the Arduino and the RXD of the module to Arduino TX.
- If the app does not work when opened on your smartphone, check the compatibility of your phone with the app on the developer's site. You may need to use an alternative app.
- The data set in your app must match the data expected in the sketch, so make sure you've used 1 for on and 0 for off.

# 25

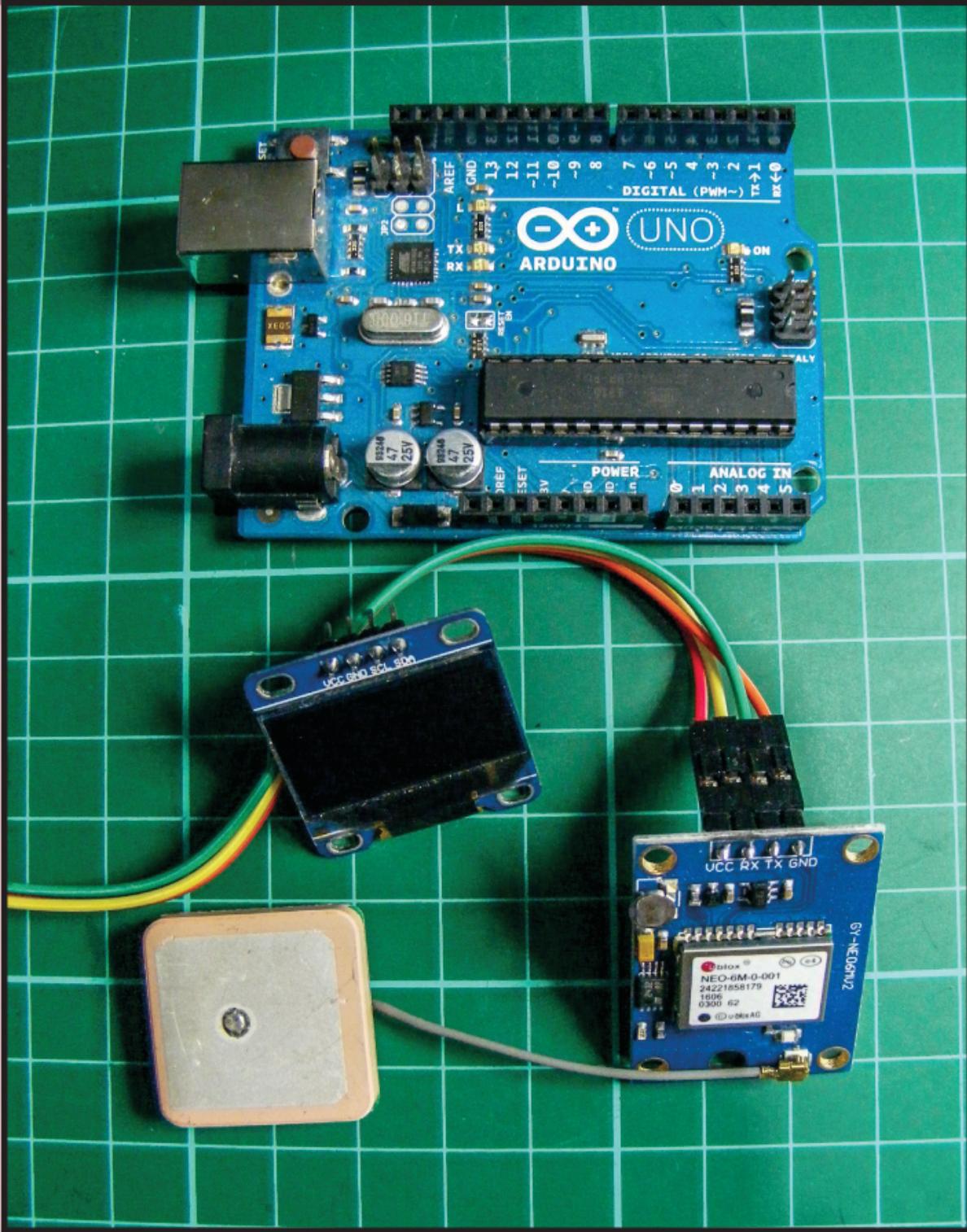
## GPS Speedometer

In this project we'll connect an OLED screen and GPS module to our Arduino to create a simple GPS speedometer that can track your speed from satellites.



**COST: \$\$\$**

**TIME: 30 MINUTES**



## PARTS REQUIRED

**Arduino board**  
**Female-to-male jumper wires**  
**OLED monochrome screen (128×64)**  
**Ublox NEO-6M GPS module aircraft flight controller and antenna**

## **LIBRARY REQUIRED**

**U8glib**

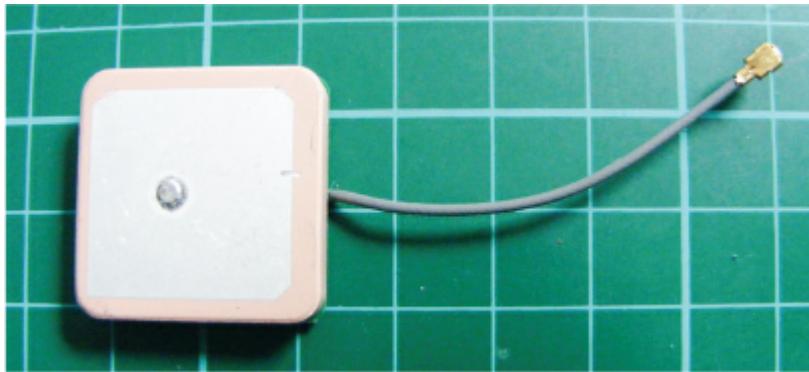
## **HOW IT WORKS**

The Ublox NEO-6M GPS module (Figure 25-1) we're using in this project is an inexpensive device generally used to track the position of model aircraft or drones. The module is widely available from the suppliers listed in the "Retailer List" on page 249, or you can search online for "Ublox NEO-6M GPS module." Make sure to buy a module that also comes with a GPS antenna, as shown in Figure 25-2.

**FIGURE 25-1:** The Ublox NEO-6M GPS module



**FIGURE 25-2:** The GPS antenna



The module uses *GPS* (*Global Positioning System*) technology to determine the exact location of the Arduino and display its speed in kilometers per hour on the OLED screen (see Project 19 for more on OLED screens). GPS consists of 32 satellites orbiting the earth, and it's used across the globe in everyday technology such as car satellite navigation systems, smartphones, and trackers.

The Navstar Global Positioning System was created in the 1970s by the United States government initially for military purposes, but it's now freely accessible for anyone with GPS receiver equipment, which probably includes you if you have a smartphone. To pinpoint the location of a receiver, the system uses the satellites, control stations on the ground, and your equipment to calculate distance, speed, and time for signals to be sent and received—with these, it can determine your location.

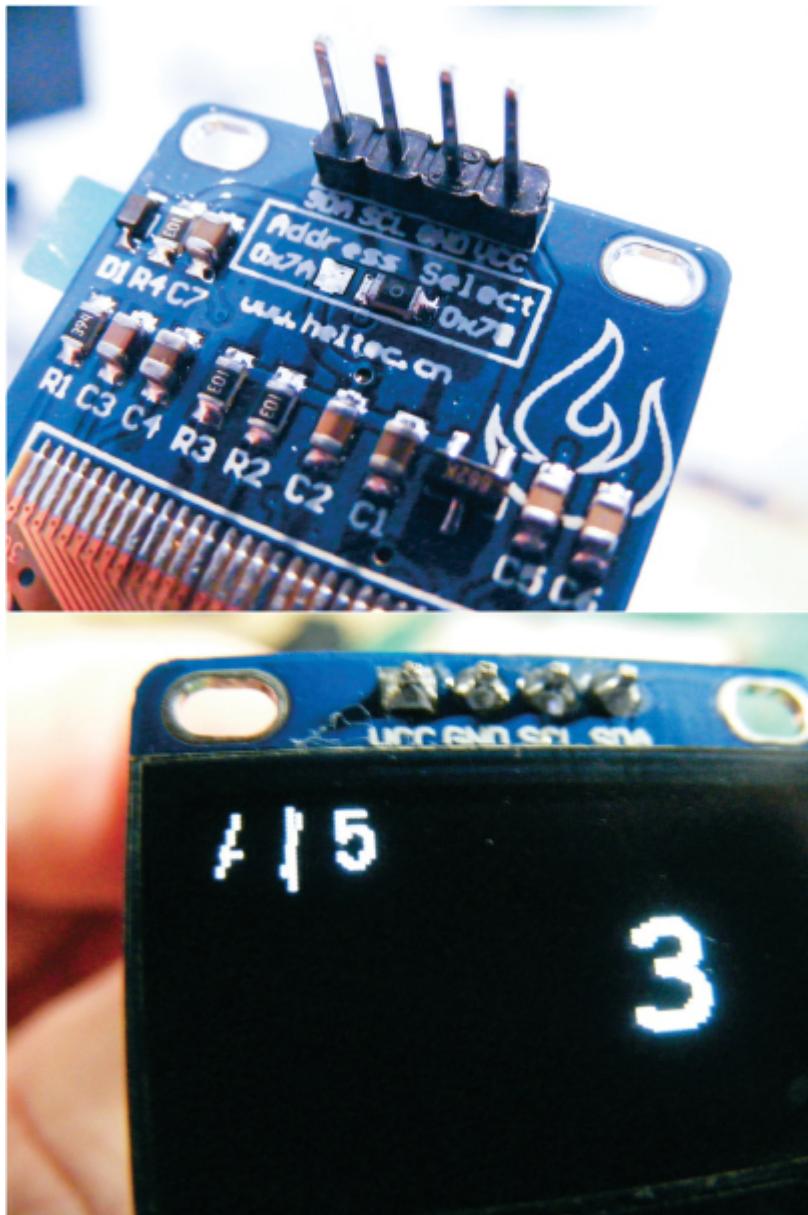
The Ublox NEO-6M GPS module receives satellite signals continuously and sends them to the Arduino to pinpoint your location. As soon as you move, your speed is sent to the OLED screen in kilometers per hour, serving as our speedometer.

While the functionality of this project is quite complex, the build is very simple. The board comes with the header pins separate, so you need to solder these in place before beginning. See the “Quick Soldering Guide” on page 12 if you need soldering guidance. The board has all the GPS circuitry built in, but you'll need to clip the GPS antenna in place; I'll show you how in a moment.

## THE BUILD

1. Take the OLED monochrome screen shown in Figure 25-3 and, using female-to-male jumper wires, make the connections in the following table. The OLED screen uses 3.3V, so make sure you connect it to Arduino 3.3V, not 5V, or you could damage the screen.

**FIGURE 25-3:** The OLED monochrome screen displays the speed of movement in kilometers per hour (digit on the right).



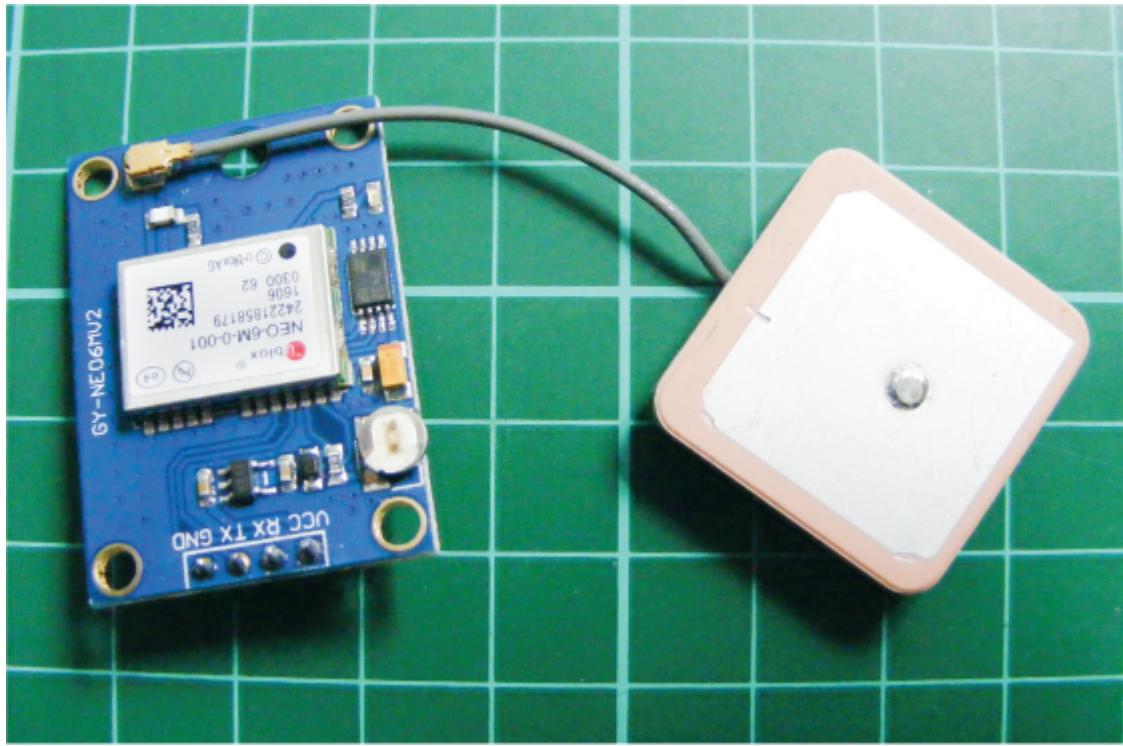
OLED SCREEN	ARDUINO
VCC	+3.3V
GND	GND
SCL	Pin A5
SDA	Pin A4

2. The GPS module uses the RX and TX pins of the Arduino for communication, but you also need these pins when uploading a sketch from your PC. Upload the code in “The Sketch” on page 227 now so those pins will be free. Connect the Arduino to your PC. Remember to first download the U8glib library and add it to the relevant folder in the Arduino IDE.
3. With the sketch uploaded, disconnect the Arduino from your PC and attach the GPS VCC to Arduino +5V, GND to GND, GPS TX to Arduino pin 0 (RX), and GPS RX to Arduino pin 1 (TX), as indicated in the following table.

GPS MODULE	ARDUINO
VCC	+5V
GND	GND
TX	Pin 0 (RX)
RX	Pin 1 (TX)

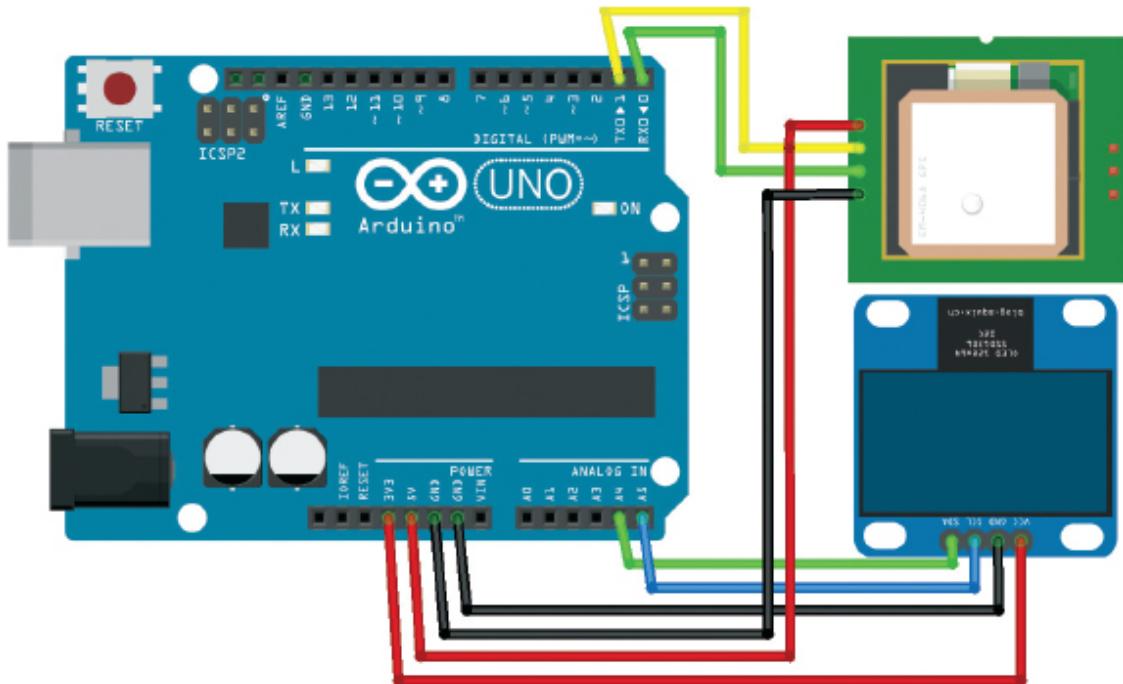
4. Clip the end of the antenna onto the module, as shown in Figure 25-4.

**FIGURE 25-4:** Clip the end of the antenna to the socket on the GPS module.



5. Confirm your setup matches the circuit diagram in Figure 25-5.

**FIGURE 25-5:** The circuit diagram for the GPS speedometer



6. Connect power to your Arduino, and the GPS speedometer is ready to use. The antenna needs to be facing upward to work, as shown in Figure 25-4, and works best outdoors because the GPS module requires line of sight with the orbiting satellites in order to function properly (though I've also had success when close to a window indoors, so experiment to see what works for you).
7. The GPS module will take about 30 seconds or so to connect to the satellites. When the connection is successful, the module LED will blink and the symbol at the top left of the OLED screen will spin.

## THE SKETCH

The sketch first calls on the U8glib library and then defines the OLED so we can control our screen. We define the GPS module as a serial connection, and tell it what information we want to receive from the satellites.

### NOTE

*Remember to disconnect the Arduino 0 (RX) pin of your build before uploading the sketch and then reconnect when running.*

The next section of code contains a long list of data. This section is quite complex, and the data sheet for the Ublox NEO-6M details all the information that can be received by the module if you're interested. For the purposes of our project, the code at ❶ contains the relevant data: the NAV-PVT data that includes the number of satellites the module is connecting to and the ground speed at which your GPS speedometer is moving. The remaining information requests are not used and are set as off.

The section that follows defines the NAV-PVT settings with a number of calculations to check that the data being received from the satellites is valid.

The loop at the end of the sketch checks to see if data is being received, and if so, animates the symbols at the top left of the OLED. The first symbol shows that the screen is refreshing correctly, and the second shows that the GPS data packet is being received from the satellites. The screen also displays the number of satellites it's connected to at the top left.

If all the data is being received as expected, the ground speed will be shown at the top right of the screen in kilometers per hour.

---

```
// Sketch reproduced with kind permission from Chris Campbell
/*
Connections:
GPS TX -> Arduino 0 // Disconnect Arduino 0 to upload this sketch
GPS RX -> Arduino 1
Screen SDA -> Arduino A4
Screen SCL -> Arduino A5
*/
#include "U8glib.h" // Call U8glib library to control OLED screen

U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_DEV_0|U8G_I2C_OPT_NO_ACK|U8G_I2C_OPT_FAST); // 
Fast I2C/TWI

#define GPS Serial // Define the serial connection as the GPS module

const unsigned char UBLOX_INIT[] PROGMEM = {
    // These lines of code request data from the satellites. Most are disabled and
    turned off.
    // Disable NMEA
    0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x24, // 
GxGGA off
    0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x01,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x2B, // 
GxGLL off
    0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x02,0x00,0x00,0x00,0x00,0x00,0x01,0x02,0x32, // 
GxGSA off
    0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x03,0x00,0x00,0x00,0x00,0x01,0x03,0x39, // 
GxGSV off
    0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x04,0x00,0x00,0x00,0x00,0x01,0x04,0x40, // 
GxRMC off
    0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x05,0x00,0x00,0x00,0x00,0x01,0x05,0x47, // 
GxVTG off

    // Disable UBX
    0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x07,0x00,0x00,0x00,0x00,0x00,0x17,0xDC, // 
NAV-PVT off
    0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x02,0x00,0x00,0x00,0x00,0x00,0x12,0xB9, //
```

```

NAV-POSLH off
0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x13,0xC0, //
NAV-STATUS off

// Enable UBX--this is the key information we require
❶ 0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x07,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x18,0xE1,
//NAV-PVT on
//0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x02,0x00,0x01,0x00,0x00,0x00,0x00,0x13,0xBE,
//NAV-POSLH on
//0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0x00,0x14,0xC5,
//NAV-STATUS on

// Rate
0xB5,0x62,0x06,0x08,0x06,0x00,0x64,0x00,0x01,0x00,0x01,0x00,0x7A,0x12,      // (10Hz)
// 0xB5,0x62,0x06,0x08,0x06,0x00,0xC8,0x00,0x01,0x00,0x01,0x00,0x00,0xDE,0x6A, // (5Hz)
// 0xB5,0x62,0x06,0x08,0x06,0x00,0xE8,0x03,0x01,0x00,0x01,0x00,0x01,0x39 // (1Hz)
};

const unsigned char UBX_HEADER[] = { 0xB5, 0x62 };

struct NAV_PVT { // This sets the GPS navigation data
    unsigned char cls;
    unsigned char id;
    unsigned short len;
    unsigned long iTOW; // GPS time of week of the navigation epoch (ms)

    unsigned short year;      // Year (UTC)
    unsigned char month;     // Month, range 1..12 (UTC)
    unsigned char day;        // Day of month, range 1..31 (UTC)
    unsigned char hour;       // Hour of day, range 0..23 (UTC)
    unsigned char minute;    // Minute of hour, range 0..59 (UTC)
    unsigned char second;    // Seconds of minute, range 0..60 (UTC)
    char valid;              // Validity Flags (see graphic below)
    unsigned long tAcc;      // Time accuracy estimate (UTC) (ns)
    long nano;                // Fraction of second, range -1e9 .. 1e9 (UTC) (ns)
    unsigned char fixType;   // GNSSfix Type, range 0..5
    char flags;               // Fix Status Flags
    unsigned char reserved1; // Reserved
    unsigned char numSV;     // Number of satellites used in Nav Solution

    long lon;                 // Longitude (deg)
    long lat;                 // Latitude (deg)
    long height;              // Height above Ellipsoid (mm)
    long hMSL;                // Height above mean sea level (mm)
    unsigned long hAcc; // Horizontal Accuracy Estimate (mm)
    unsigned long vAcc; // Vertical Accuracy Estimate (mm)

    long velN;                // NED north velocity (mm/s)
    long velE;                // NED east velocity (mm/s)
    long velD;                // NED down velocity (mm/s)
};

```

```

long gSpeed;           // Ground Speed (2-D) (mm/s)
long heading;          // Heading of motion 2-D (deg)
unsigned long sAcc;    // Speed accuracy estimate
unsigned long headingAcc; // Heading accuracy estimate
unsigned short pDOP;   // Position dilution of precision
short reserved2;       // Reserved
unsigned long reserved3; // Reserved
};

NAV_PVT pvt;

void calcChecksum(unsigned char* CK) {
    memset(CK, 0, 2);
    for (int i = 0; i < (int)sizeof(NAV_PVT); i++) {
        CK[0] += ((unsigned char*)(&pvt))[i];
        CK[1] += CK[0];
    }
}

long numGPSMessagesReceived = 0;

bool processGPS() {
    static int fpos = 0;
    static unsigned char checksum[2];
    const int payloadSize = sizeof(NAV_PVT);

    while ( GPS.available() ) {
        byte c = GPS.read();
        if ( fpos < 2 ) {
            if ( c == UBX_HEADER[fpos] )
                fpos++;
            else
                fpos = 0;
        }
        else {
            if ( (fpos-2) < payloadSize )
                ((unsigned char*)(&pvt))[fpos-2] = c;

            fpos++;

            if ( fpos == (payloadSize+2) ) {
                calcChecksum(checksum);
            }
            else if ( fpos == (payloadSize+3) ) {
                if ( c != checksum[0] )
                    fpos = 0;
            }
            else if ( fpos == (payloadSize+4) ) {
                fpos = 0;
                if ( c == checksum[1] ) {

```

```

        return true;
    }
}
else if ( fpos > (payloadSize+4) ) {
    fpos = 0;
}
}
}
return false;
}

void setup() {
    GPS.begin(9600);

    u8g.setColorIndex(1);

    // Send configuration data in UBX protocol
    for (unsigned int i = 0; i < sizeof(UBLOX_INIT); i++) {
        GPS.write( pgm_read_byte(UBLOX_INIT+i) );
        delay(5); // Simulate a 38400baud pace (or less),
                   // or otherwise commands are not accepted by the device
    }
}

long gSpeed = 0;
int numSV = 0;
unsigned long lastScreenUpdate = 0;
char speedBuf[16];
char satsBuf[16];

char* spinner = "/-\\"; // Symbol for the spinner on screen to
                       // show communication
byte screenRefreshSpinnerPos = 0;
byte gpsUpdateSpinnerPos = 0;

void loop() {
    if (processGPS()) {
        numSV = pvt.numSV;
        gSpeed = pvt.gSpeed;
        gpsUpdateSpinnerPos = (gpsUpdateSpinnerPos + 1) % 4;
    }

    unsigned long now = millis();
    if (now - lastScreenUpdate > 100) {
        updateScreen();
        lastScreenUpdate = now;
        screenRefreshSpinnerPos = (screenRefreshSpinnerPos + 1) % 4;
    }
}

```

```

void draw() {
    u8g.setFont(u8g_font_courB24);
    u8g.drawStr( 36, 45, speedBuf);
    u8g.setFont(u8g_font_fur11);
    u8g.drawStr( 2, 12, satsBuf);
}

void updateScreen() {

    int kmh = gSpeed * 0.0036;
    sprintf(speedBuf, "%3d", kmh);
    sprintf(satsBuf, "%c %c %d", spinner[screenRefreshSpinnerPos],
            spinner[gpsUpdateSpinnerPos], numSV);

    u8g.firstPage();
    do {
        draw();
    } while( u8g.nextPage() );
}

```

---

## TROUBLESHOOTING

- Q.** *The code compiles, but the expected information is not shown onscreen.*
- If nothing shows on the OLED screen, recheck that your wiring matches Figure 25-5; it's quite easy to reverse the TX and RX wires accidentally.
  - The symbols at the top left of the screen will rotate to show the screen is working correctly and that the GPS module is receiving data. If the far-left symbol spins but not the GPS symbol, you have your TX and RX wires crossed; recheck the wiring for the module.
  - The GPS module works best outdoors and should have line of sight to the satellites orbiting the earth, so try repositioning the module until you get a reading. It can take 30–60 seconds to get a stable reading.
  - Remember that the OLED screen should be connected to 3.3V and not 5V.

# Troubleshooting Tips for Common Errors

All the sketches for the projects in this book can be downloaded from <https://www.nostarch.com/arduinohandbook2/> and have been verified to work correctly. However, when you compile a sketch in the Arduino IDE, there are a number of problems that you may encounter.

This section will go through three of the most common types of errors, explaining why they occur and how to fix them. When an error occurs, the monitor box at the bottom of the IDE will helpfully highlight the line of code that caused the error, as shown in Figure A-1. This information will be invaluable to you in fixing your code.

**FIGURE A-1:** The IDE will highlight the line where the error has occurred.

The screenshot shows the Arduino IDE interface with the title bar "Blink\_LED | Arduino 1.8.1". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has icons for back, forward, save, and upload. The code editor contains the "Blink\_LED" sketch. The code is as follows:

```
// Blinking LED Project - This example code is in the public domain

int led = 13;
void setup()
{
pinMode(led, OUTPUT);
}
void loop()
{
digitalWrite(led, HIGH);
} delay(1000);
digitalWrite(led, LOW);
} delay(1000);
```

An orange status bar at the bottom left says "expected '}' at end of input". The status bar at the bottom right says "Arduino/Genuine Uno on COM1".

## UPLOAD ERROR

When you upload your code, you get a message like the one in Figure A-2, which says:

```
-----  
avrdude: ser_open(): can't open device "COM1": No such file or  
directory  
-----
```

**FIGURE A-2:** The error message “Problem uploading to board”

The screenshot shows the Arduino IDE interface with the title bar "Blink\_LED | Arduino 1.8.1". The status bar at the bottom right says "Arduino/Genuine Uno on COM1". The message window displays the following text:

```
Problem uploading to board. See http://www.arduino.cc/en/Guide/Troubleshooting#upload for suggestions. Copy error messages

Sketch uses 940 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
avrdude: ser_open(): can't open device 'COM1': No such file or directory
Problem uploading to board. See http://www.arduino.cc/en/Guide/Troubleshooting#upload for suggestions.
```

## Solutions

This error generally means that the IDE cannot find your Arduino board. Try one of these solutions

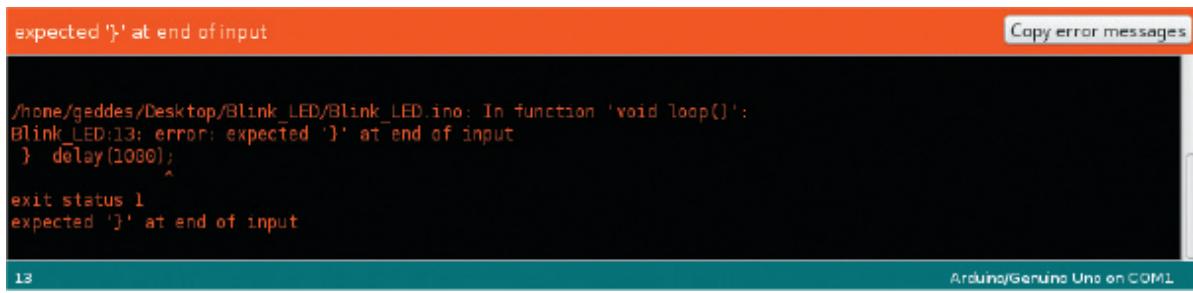
- Check that your USB connection is securely inserted into your PC's USB port.
- In the IDE, open the **Tools** tab and select **Port**. From the drop-down menu, you should see that one of the COM ports is highlighted. If this is not the port your Arduino is connected to, select the correct one.
- If the correct port is already highlighted, verify that the right board type is selected: open the **Tools** tab, select **Board**, and from the drop-down menu make sure the type of Arduino board you have attached is highlighted. This is set to Arduino Uno by default.
- You can also check the Arduino documentation for more possible solutions: <http://www.arduino.cc/en/Guide/Troubleshooting#upload>.

## CODE VERIFICATION ERROR #1

When you verify your code, you receive an error like the one in Figure A-3, which says:

-----  
expected '}' at end of input  
-----

**FIGURE A-3:** The error message “expected '}' at end of input”



The screenshot shows the Arduino IDE interface. At the top, there is an orange header bar with the text "expected '}' at end of input" and a "Copy error messages" button. Below the header is a black terminal window displaying the following error message:  
/home/geddes/Desktop/Blink\_LED/Blink\_LED.ino: In function 'void loop()':  
Blink\_LED:13: error: expected '}' at end of input  
} delay(1000);  
^  
exit status 1  
expected '}' at end of input

## Solution

Check that each opening curly bracket ({) has a closing curly bracket (}) and, if not, add the closing bracket. Curly brackets define the start and

end of a block of code, and every open bracket needs a closing bracket to complete a function or loop. In this instance, you would add a closed bracket at the end of your code.

## CODE VERIFICATION ERROR #2

When verifying your code, you receive the error shown in Figure A-4, which says:

```
-----  
expected ';' before '}' token  
-----
```

**FIGURE A-4:** The error message “expected ';' before '}' token”



The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with various icons. The main window has a title bar "Blink\_LED" and a status bar indicating "Arduino/Genuino Uno on COM1". The code editor contains the following sketch:

```
// Blinking LED Project - This example code is in the public domain

int led = 13;
void setup()
{
pinMode(led, OUTPUT);
}
void loop()
{
digitalWrite(led, HIGH);
{ delay(1000);
digitalWrite(led, LOW);
} delay(1000)
}
```

The line "}" on the 14th line is highlighted in red, indicating an error. The terminal window below shows the error message:

```
expected ';' before '}' token
/home/geddes/Desktop/Blink_LED/Blink_LED.ino: In function 'void loop()':
Blink_LED:14: error: expected ';' before '}' token
^
exit status 1
expected ';' before '}' token
```

The line number 14 is also highlighted in red in the terminal output.

## Solution

This error, one of the most common you'll encounter, indicates that you missed a semicolon (;) at the end of a line. Add a semicolon to the line above the one highlighted in the IDE.

# MISSING LIBRARY ERROR

When verifying your code, you receive an error like this:

```
-----  
fatal error: #NewPing.h no such file or directory  
-----
```

The example shown in Figure A-5 is from Project 20, which uses the NewPing library.

**FIGURE A-5:** The error message “Error compiling for board Arduino/Genuino Uno”

The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The title bar says "20\_super\_speaker". The code editor contains C++ code for a NewPing ultrasonic sensor. The code includes includes for NewPing.h, defines for trigPin (12), echoPin (13), and speaker (3), and setup/loop functions. A red box highlights the error message at the bottom of the code editor: "Error compiling for board Arduino/Genuino Uno." Below the code editor is the terminal window, which displays the command line output of the compilation process, including the error message "fatal error: NewPing.h: No such file or directory". The status bar at the bottom right shows "Arduino/Genuino Uno (COM1)".

```
Ble Edit Sketch Tools Help  
20_super_speaker  
-----  
#include <NewPing.h> // This calls the NewPing library  
#define trigPin 12 // Trig pin attached to Arduino 12  
#define echoPin 13 // Echo pin attached to Arduino 13  
#define MAX_DISTANCE 500  
NewPing sonar(trigPin, echoPin, MAX_DISTANCE);  
int speaker = 3; // speaker connected to pin 3  
  
void setup() {  
  Serial.begin (115200);  
  pinMode(trigPin, OUTPUT);  
  pinMode(echoPin, INPUT);  
  pinMode(speaker, OUTPUT);  
}  
  
void loop() {  
  int duration, distance; pos=0.1;  
  
  digitalWrite(trigPin, LOW);  
  delayMicroseconds(2);  
  digitalWrite(trigPin, HIGH); // Trig sends a ping  
  delayMicroseconds(10);  
  digitalWrite(trigPin, LOW);  
  duration = pulseIn(echoPin, HIGH); // Echo receives the ping  
  distance = (duration/2) / 29.1;  
  Serial.print(distance);  
  Serial.println(" cm");  
  
  if(distance<=15) // If distance is less than 15  
    -----  
    Error compiling for board Arduino/Genuino Uno.  
    -----  
  
/home/geddes/Dropbox/APH-Q2O/Projects/Project 20 - Ultrasonic speaker/_20_super_speaker/_20_super_speaker.ino:1:56: fatal error: NewPing.h: No such file or directory  
#include <NewPing.h> // This calls the NewPing library  
^  
compilation terminated.  
exit status 1  
Error compiling for board Arduino/Genuino Uno.  
-----  
Arduino/Genuino Uno (COM1)
```

## Solution

This error is also quite common and it means the IDE cannot find the expected library in the library folder. Follow the instructions in “Installing Libraries” on page 8 to make sure you’ve installed any libraries required by your code that are not included by default in the IDE. Remember that it is not enough to just download these libraries—you have to install them too.

Each project in this book lists the required libraries at the start of the chapter. You can download those not included in the IDE from <https://www.nostarch.com/arduinohandbook2/>.

# Components

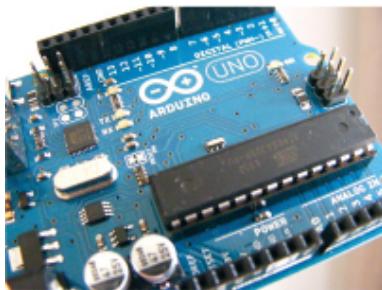
This section gives you some more information on the components used in this book. Each component is accompanied by a photo and a few details for quick reference and identification. At the end, I've also included a handy list of retailers to buy the parts from and a quick lesson on reading resistor values.

## COMPONENTS GUIDE

The components are listed in the order in which they appear in the book. Many of the items can be found with a simple search on websites like eBay and Amazon, but a list of specialist suppliers is also provided in the “Retailer List” on page 249.

### Arduino Uno R3

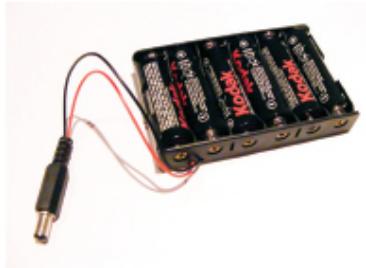
The Arduino Uno R3 microcontroller board is the main component for the book and the brain for all your projects.



- Quantity: 1
- Connections: 14
- Projects: All

### 9V Battery Pack

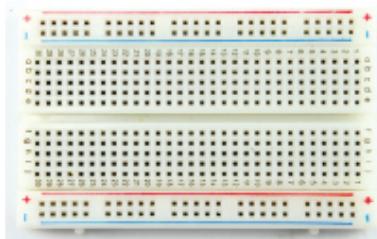
The 9V battery pack with a 2.1 mm jack for 6 AA batteries plugs into the power port on the Arduino and can be used to power your projects. Note that the Arduino can also be powered through the USB cable.



- Quantity: 1
- Connections: 1
- Projects: Optional for all

## Breadboard

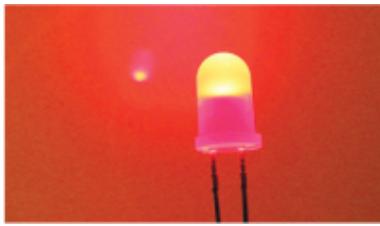
The breadboard is a prototyping board used to connect components together to create your projects. See the primer for more details.



- Quantity: 1 full-size board, 1 half-size board, 1 mini board
- Connections: 940 on a full board, 420 on a half board, 170 on a mini board
- Projects: All except Projects 4, 6, 7, 16, 19, 22, and 25

## LED

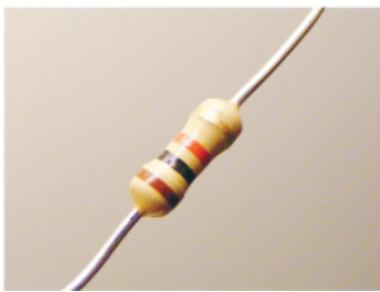
An LED, or *light-emitting diode*, is a small bulb that emits light when a low current is passed through it. It has two legs, the longer of which is the positive connection. LEDs generally require a resistor or they may burn out. LEDs are polarized, meaning current flows only in one direction.



- Quantity: 40 (10 red, 10 blue, 10 yellow, 10 green)
- Connections: 2
- Projects: 1, 2, 9, 15, 17, 21, 23, 24

## Resistor

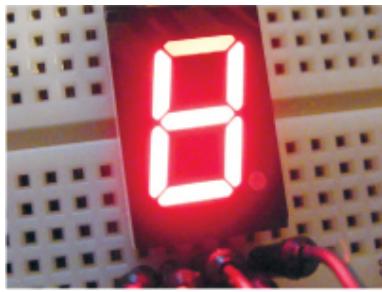
Resistors restrict the amount of current that can flow through a circuit to prevent components from overloading. A resistor looks like a cylinder with colored bands and a wire extending from each end. The resistance value is indicated by a color code—see “Decoding Resistor Values” on page 250 for more details. Check the value carefully, as it can be easy to choose the wrong one. Resistors come in four-, five-, and six-band varieties, so be aware that, for example, a four-band 220-ohm resistor can look slightly different from a five-band resistor of the same value.



- Quantity: 9 220-ohm, 4 10k-ohm, 8 1k-ohm
- Connections: 2
- Projects: 1–3, 5, 8–10, 15, 17, 18, 21, 23, 24

## Seven-Segment LED Display

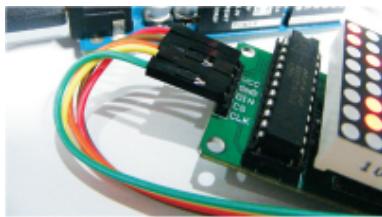
A seven-segment LED display forms a digit or character using LED segments, and is often used to display numbers for counters, clocks, or timers. You can get single-digit to eight-digit displays, and four-digit displays are commonly used for digital clocks.



- Quantity: 1
- Connections: 10
- Project: 3

## 8x8 LED Maxim 7219 Matrix Module

This prebuilt 8x8 LED matrix module needs only five pins connected to your Arduino to work.



- Quantity: 1
- Connections: 5
- Project: 4

## RGB LED

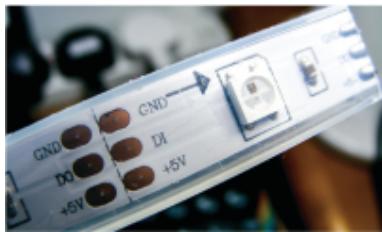
An RGB LED combines three colors—red, green, and blue—to make any color of the rainbow. It is a clear LED with four legs, each of which needs a resistor to limit the current and prevent the LED from burning out. The longest leg is either the common cathode or anode.



- Quantity: 1
- Connections: 4
- Project: 5

## RGB LED Strip (WS2812B 5V 32-LED Strip)

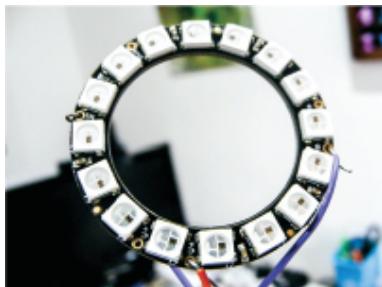
LED strips come in single-color or multicolored varieties, and can differ in how the individual LEDs are controlled. Single-color, or *multicolor nonaddressable*, strips can light only one color at a time. RGB multicolored strips are generally *addressable*, which means each LED has its own chip and can be individually controlled, allowing multiple colors to light simultaneously.



- Quantity: 1
- Connections: 3
- Project: 6

## Adafruit NeoPixel Ring with 16 RGB LEDs

The Adafruit NeoPixel ring has 16 RGB surface-mounted LEDs, each of which is addressable, allowing you to control each LED separately.

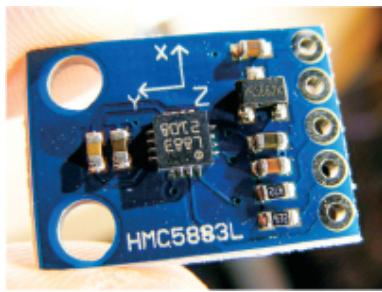


- Quantity: 1
- Connections: 3

- Project: 7

## HMC5883L Three-Axis Sensor

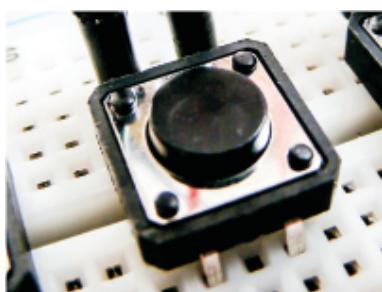
The HMC5883L three-axis sensor is a multichip module used for sensing magnetic fields—we use it to detect magnetic north to act as a compass. The module may require you to solder header pins.



- Quantity: 1
- Connections: 4
- Project: 7

## Pushbutton

A pushbutton is a simple switch that makes a connection when pushed. Also known as a momentary switch, a pushbutton connects a circuit when pushed in, and spring backs to break the connection when released. Pushbuttons vary in size, but most have four pins.



- Quantity: 8
- Connections: 4
- Project: 8

## Piezo Sounder

The piezo sounder is a very basic speaker often used in inexpensive toys. A pulse of current causes it to click extremely quickly, and a stream of pulses emits a tone. The piezo sounder often looks like a small black box with two wires. Taken out of the case, it looks like a small, gold disc.



- Quantity: 1
- Connections: 2
- Projects: 8, 15

## 3.5 mm Female Headphone Jack

The 3.5 mm female headphone jack is a simple jack that allows you to connect audio devices to your Arduino. It can be purchased on its own or reclaimed from a dollar-store radio.



- Quantity: 1
- Connections: 3
- Project: 9

## Servomotor

A servomotor is a motor with an arm attachment that you can position to specific angles by sending the servo a coded signal. The motor is in a small box with three wires and an output shaft to which you can attach the arm, known as a *horn*.

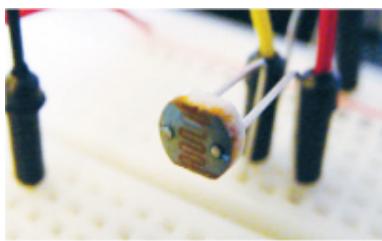
This book uses the Tower Pro SG90 9g servomotor, which turns 180 degrees; others are continuous and turn the full 360 degrees.



- Quantity: 1
- Connections: 3
- Projects: 10, 21, 22

## Photoresistor

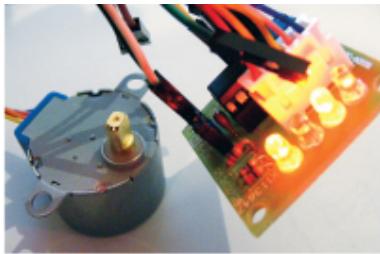
A photoresistor, also referred to as a *light-dependent resistor* or a *diode*, detects light levels by producing a variable resistance depending on the amount of light falling on it. There are different styles, but it usually looks like a small, clear oval with wavy lines and two legs. You will need to calibrate your photoresistor to determine light levels before using it in a program.



- Quantity: 1
- Connections: 2
- Project: 10

## 28BYJ-48 Stepper Motor with ULN2003 Driver Module

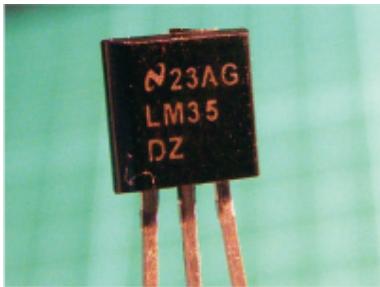
A stepper motor is a DC electric motor that divides a full 360-degree rotation of the arm into a number of equal steps for heightened control. We're using the 28BYJ-48 stepper motor, which comes with a ULN2003 driver module to control it.



- Quantity: 1
- Connections: 5
- Project: 11

## LM35 Temperature Sensor

The LM35 temperature sensor detects the temperature and sends the reading as a voltage value to the Arduino so we can measure heat.



- Quantity: 1
- Connections: 3
- Projects: 12, 14

## 12V Mini Computer Cooling Fan

A 12V mini computer cooling fan is the cooling fan used internally in a computer. We are using a 4 cm × 4 cm fan, but you could use a larger one if

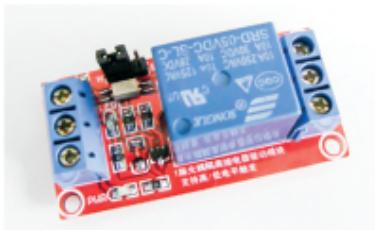
required. You could also reclaim one from an old PC, as long as it is no longer used.



- Quantity: 1
- Connections: 2
- Project: 12

## 5V Single-Channel Relay Module

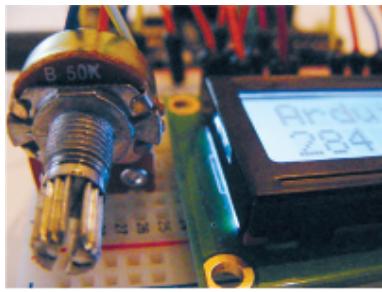
A relay is an electronically operated switch that, in this case, uses an electromagnet to mechanically open or close a circuit.



- Quantity: 1
- Connections: 6
- Project: 12

## Potentiometer

A potentiometer is a resistor whose value you can vary to manipulate the voltage flowing through it, allowing you to control how much power goes to a component. It has a knob that you can turn and three pins at the bottom. The center pin is the control pin, with power to either side. It's commonly used to control an output such as the volume on a radio. You connect power to pins 1 and 3, and it doesn't matter which way they are connected.

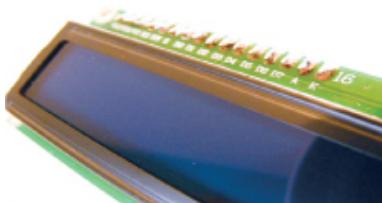


- Quantity: 2 50k-ohm, 1 10k-ohm
- Connections: 3
- Projects: 11, 13–15, 18

## LCD Screen

An LCD (liquid crystal display) screen is a display screen for outputting characters or images. It is composed of two sheets of polarizing material with a liquid crystal solution between them. Passing current through the liquid crystal makes it opaque, creating an image against a backlight.

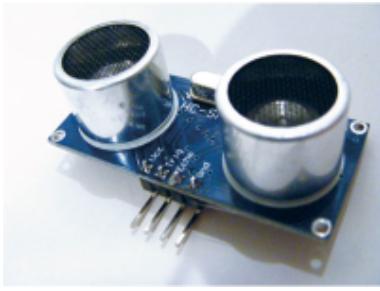
Screens come in various dimensions. The one shown here is an HD44780 16×2 (16 characters × 2 lines) and has 16 connections.



- Quantity: 1
- Connections: 16
- Projects: 13–16

## Ultrasonic Sensor

An ultrasonic sensor sends out a signal (often referred to as a *ping*), which bounces off an object and is returned to the sensor. Distance is calculated from the time the signal takes to return once it has been sent. The sensor used in this book is the HC-SR04 ultrasonic sensor, a module board with two round sensors and four pins.



- Quantity: 1
- Connections: 4
- Projects: 13, 17, 20, 22

## Keypad

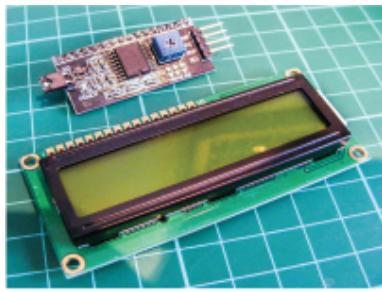
A 3×4 membrane keypad is basically a series of switches. The example shown here has 12 pushbuttons connected in series, but a 16-button version is also available. Of the seven connections, four control the rows and three control the columns. The Arduino will replicate the number of the button pressed.



- Quantity: 1
- Connections: 7
- Project: 15

## Serial LCD Screen Module

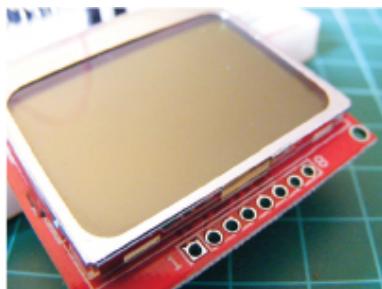
This 16×2 LCD screen has a serial module attached and thus requires only power and two pins connected to the Arduino.



- Quantity: 1
- Connections: 4
- Project: 16

## Nokia 5110 LCD Screen

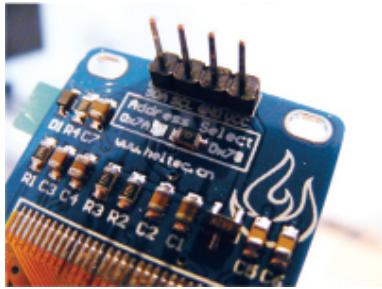
This is a Nokia 84×48-pixel screen that, accounting for spaces between the characters, gives us a 12×6-character screen. It works similarly to the LCD screen in Project 13, by sending current through the liquid crystal from the Arduino at certain pixels to form letters or images.



- Quantity: 1
- Connections: 8
- Project: 18

## OLED Monochrome Screen (128×64)

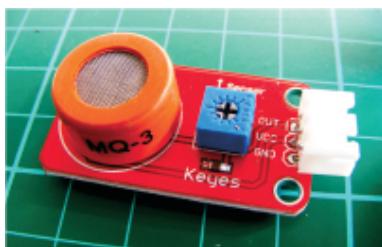
The OLED (organic light-emitting diode) screen is a light-emitting technology composed of a thin, multilayered organic film placed between an anode and cathode. The one we use in this book has a 128×64 screen size.



- Quantity: 1
- Connections: 4
- Projects: 19, 25

## Keyes MQ3 Alcohol Sensor Module

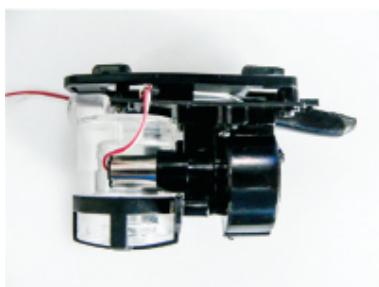
The MQ3 is a gas sensor sensitive to alcohol and ethanol. We use it in the breathalyzer in Project 19.



- Quantity: 1
- Connections: 3
- Project: 19

## WLToys V959-18 Water Jet Pistol

The V959-18 water jet pistol comprises a small reservoir to hold water and a mini pump that pushes water through a nozzle.



- Quantity: 1
- Connections: 2
- Project: 20

## Optical Fingerprint Sensor (ZFM-20 Series)

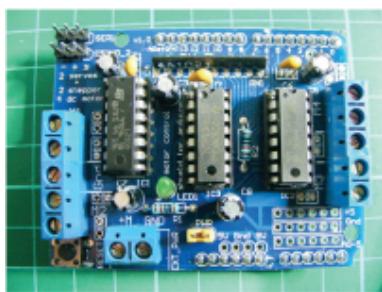
The ZFM-20 fingerprint sensor is a fingerprint comparison module that takes a photograph of a fingerprint and adds it to its database, allowing you to check if a new fingerprint matches one stored there. The sensor can hold up to 162 fingerprints.



- Quantity: 1
- Connections: 4
- Project: 21

## L293d Motor Shield

The L293d motor shield is a module for controlling motors that we use for our robot in Project 22.



- Quantity: 1
- Connections: fits on top of the Arduino
- Project: 22

## Robot Chassis Kit

If you search online for “Arduino robot kit,” you should be able to find a kit that contains two DC motors and wheels, a base plate, a battery pack, a center wheel, and the fittings needed to build an Arduino robot. The kit I bought was specifically named the “2WD Smart Motor Robot Car Chassis Kit for Arduino 1:48.”



- Quantity: 1
- Connections: 4 (2 for each motor)
- Project: 22

## Ethernet Shield W5100 LAN Expansion Board

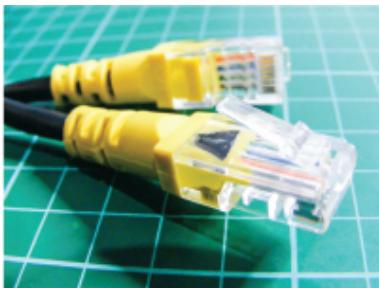
The Ethernet shield W5100 LAN expansion board fits directly on top of the Arduino to provide additional functionality, such as a web server or client that allows the Arduino to connect to a network.



- Quantity: 1
- Connections: multiple
- Project: 23

## Ethernet Cable

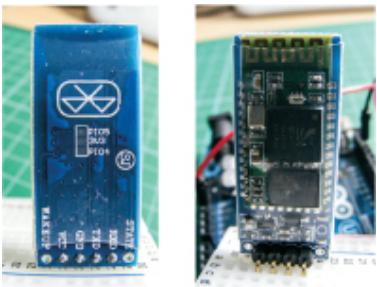
An Ethernet cable transmits data between an internet connection or network and a device.



- Quantity: 1
- Connections: 1
- Project: 23

## HC-06 Bluetooth Module

The HC-06 module provides Bluetooth wireless capabilities so the Arduino can transmit radio waves to exchange data over short distances. Smartphones, laptops, and multimedia devices such as speakers use Bluetooth technology as a common standard.

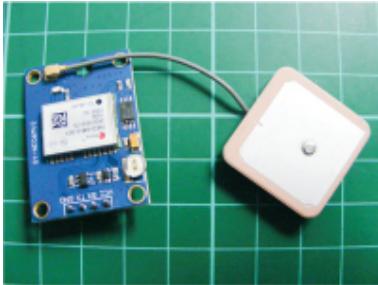


- Quantity: 1
- Connections: 4
- Project: 24

## Ublox NEO-6M GPS Module Aircraft Flight Controller and Antenna

The Ublox NEO-6M GPS module is a tracking device that connects top GPS satellites, generally used to track the position of model aircraft or

drones. The module is widely available from the sources listed here, or you can simply search for “Ublox NEO-6M GPS module” online. Make sure to get a module that also comes with a GPS antenna.



- Quantity: 1
- Connections: 5, including antenna
- Project: 25

## RETAILER LIST

As mentioned earlier, most electronic components can be found on generic sites like Amazon or eBay, but if you have trouble finding anything, the retailers listed here should be able to help.

### US Retailers

**Adafruit** <https://www.adafruit.com/>

**DigiKey** <http://www.digikey.com/>

**Jameco Electronics** <http://www.jameco.com/>

**MCM** <http://www.mcmelectronics.com/>

**Newark** <http://www.newark.com/>

**RS Components** <http://www.rs-components.com/>

**Seeed Studio** <https://www.seeedstudio.com/>

**SparkFun** <https://www.sparkfun.com/>

### Australian Retailers

**Core Electronics** <https://core-electronics.com.au/arduino.html>

**Little Bird Electronics** <http://www.littlebirdelectronics.com.au/>

## European Retailers

**Electronic Sweet Pea's** <http://www.sweetpeas.se/>

**Element 14** <http://www.element14.com/>

**Farnell** <http://www.farnell.com/>

## UK Retailers

**4tronix** <http://www.4tronix.co.uk/store/>

**Cool Components** <http://www.coolcomponents.co.uk/>

**CPC** <http://cpc.farnell.com/>

**Hobby Components** <https://www.hobbycomponents.com/>

**Mallinson Electrical** <http://www.mallinson-electrical.com/shop/>

**Maplin** <http://www.maplin.co.uk/>

**Oomlout** <http://oomlout.co.uk/>

**The Pi Hut** <http://thepihut.com/>

**Proto-pic** <http://proto-pic.co.uk/>

**Rapid Electronics** <http://www.rapidonline.com/>

**RS** <http://uk.rs-online.com/web/>

**Spiratronics** <http://spiratronics.com/>

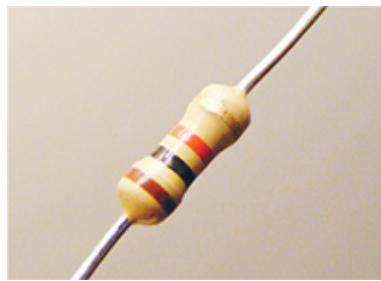
## DECODING RESISTOR VALUES

In most of the projects in this book we've used *resistors*. Resistors are electrical components that limit the amount of current allowed through a circuit (measured in ohms). They are used to protect components, like LEDs, from overloading and burning out. The value of a resistor is

identified by colored bands on the body. Resistors can have four, five, or six colored bands.

It's important to be able to determine the value of a resistor so that you know you're using the correct one in your project. Let's try to determine the value of the four-band resistor shown in Figure B-1.

**FIGURE B-1:** A four-band resistor



Viewing the resistor with the silver or gold band on the right, note the order of the colors from left to right. If the resistor has no silver or gold band, make sure the side with the three colored bands is on the left.

Use Table B-1 to determine the value of the resistor.

**TABLE B-1:** Calculating Resistor Values

COLOR	FIRST BAND	SECOND BAND	THIRD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	+/-1%
Red	2	2	2	100Ω	+/-2%
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	+/-0.5%
Blue	6	6	6	1MΩ	+/-0.25%
Violet	7	7	7	10MΩ	+/-0.10%

COLOR	FIRST BAND	SECOND BAND	THIRD BAND	MULTIPLIER	TOLERANCE
Gray	8	8	8		+/-0.05%
White	9	9	9		
Gold				0.1Ω	+/-5%
Silver				0.01Ω	+/-10%

The first and second bands give you the numerical value, the third band tells you how many zeros to add to that number, and the fourth band tells you the *tolerance*—that is, how much the actual value can vary from the intended value.

### NOTE

*While the band that denotes the tolerance is most commonly silver or gold, it can be any of the other colors that has a percentage listed in the tolerance column. If you have a resistor with a tolerance band that isn't silver or gold, there should be a small gap between the value bands and the tolerance band so you can tell which it is.*

So, for the resistor in Figure B-1:

- First band is brown (1) = 1.
- Second band is black (0) = 0.
- Third band is red (2) = 00 (2 is the number of zeros).
- Fourth band is gold, so the tolerance (accuracy) is +/– 5 percent.

So this resistor is 1,000 ohms or 1 kilohm, with a tolerance of 5 percent, meaning that the actual value can be up to 5 percent more or less than 1 kilohm. We can do the same calculation for a five- or six-band resistor.

If you're ever unsure of a resistor's value, you can look it up by doing a quick online search of the colored bands on the resistor's body. Just make

sure to list the colors in the correct order, reading them from left to right, with the tolerance band on the right.

# Arduino Pin Reference

Without going into too much detail, this section gives you a reference to the pins on the Arduino Uno, their technical names, and their functions. The pins are explained in more detail in the projects in which they're used, so the information here will probably make more sense once you've built a few projects.

ARDUINO PIN	FUNCTION AND LABEL	ADDITIONAL FUNCTION
0	RX—Used to receive TTL serial data	
1	TX—Used to transmit TTL serial data	
2	External interrupt	
3	External interrupt	Pulse width modulation
4	XCK/TO—External Clock Input/Output (Timer/Counter 0)	
5	T1 (Timer/Counter 1)	Pulse width modulation
6	AIN0—Analog comparator positive input	Pulse width modulation
7	AIN1—Analog comparator negative input	
8	ICP1—Input capture	

ARDUINO PIN	FUNCTION AND LABEL	ADDITIONAL FUNCTION
9	OC1A—Timer register	Pulse width modulation
10	SS—Slave Select (serial data) used in SPI communication	Pulse width modulation
11	MOSI—Master Out Slave In (data in) used in SPI communication	Pulse width modulation
12	MISO—Master In Slave Out (data out) used in SPI communication	
13	SCK—Serial Clock (output from master) used in SPI communication	
AREF	Reference voltage for analog inputs	
A0	Analog input can give 1,024 different values.	
A1	Analog input can give 1,024 different values.	
A2	Analog input can give 1,024 different values.	
A3	Analog input can give 1,024 different values.	

ARDUINO PIN	FUNCTION AND LABEL	ADDITIONAL FUNCTION
A4	Analog input can give 1,024 different values.	SDA (serial data line) pin supports TWI (two-wire interface) using the Wire library for I2C components.
A5	Analog input can give 1,024 different values.	SCL (serial clock line) pin supports TWI using the Wire library for I2C components.
RESET	Can be used to reset the microcontroller	
3.3V	3.3 volt output used for low voltage components. This is the only 3.3V source. The digital and analog pins operate at 5V.	
5V	Standard +5V output	
GND	Ground/negative power	
Vin	9V power can be input here or accessed if using power jack.	

**Serial: 0 (RX) and 1 (TX)** These pins are used to receive (RX) and transmit (TX) transistor-transistor logic (TTL) serial data. We use the TX and RX pins in Projects 21, 24, and 25.

**External interrupts: 2 and 3** These pins can be configured to trigger an interrupt on a low value, a *rising* or *falling edge* (a signal going from low to high or high to low, respectively), or a change in value. An *interrupt* is a signal that tells the Arduino to stop and carry out another

function when the pins have detected an external event, such as a pushbutton being pressed.

**PWM: 3, 5, 6, 9, 10, and 11** These pins can be used with pulse width modulation through the `analogWrite()` function. There's more information on this in Project 5.

**SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)** These pins support SPI communication using the SPI library and are used in Project 4.

**LED: 13** There is a built-in LED connected to digital pin 13. When the pin is `HIGH`, the LED is on; when the pin is `LOW`, it's off. The built-in LED on pin 13 is used to show when the onboard ATmega328p bootloader is running, usually when the Arduino is starting up.

**AREF** This is the reference voltage for the analog inputs; it's used with `analogReference()`. We can input from 0 to 5V, so if your sensor requires a lower voltage than 5V, you can use this pin to increase the resolution for a more accurate reading.

**Analog inputs: A0–A5** The Uno has six analog inputs, each of which provides 1,024 different values.

**TWI: A4 and A5** These pins support *TWI (two-wire interface)* communication using the Wire library. This is used to control and communicate with an I2C device, such as a serial LCD screen, using only two wires.

**RESET** Set this to `LOW` to reset the microcontroller. This is typically used to add a reset button.

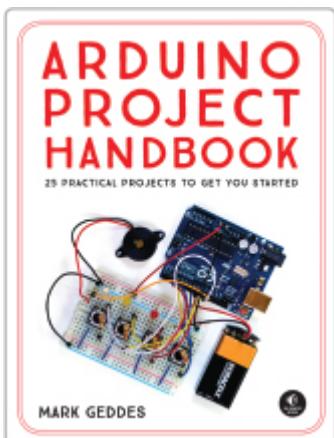
Don't worry if this information doesn't mean much to you right now. You might find it useful in your future Arduino endeavors, and you can reference it as you progress through the projects in the book.

*Arduino Project Handbook, Volume 2* is set in Helvetica Neue, Montserrat, True North, and TheSansMono Condensed.

## UPDATES

Visit <https://www.nostarch.com/arduinohandbook2/> for updates, errata, and other information.

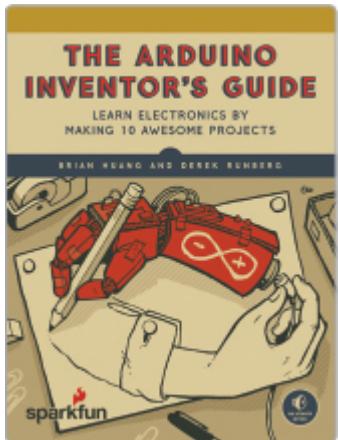
More no-nonsense books from  NO STARCH PRESS



## ARDUINO PROJECT HANDBOOK, VOL. 1

25 Practical Projects to Get You Started

by MARK GEDDES JUNE 2016, 272 PP., \$24.95 ISBN 978-1-59327-690-4 *full color*

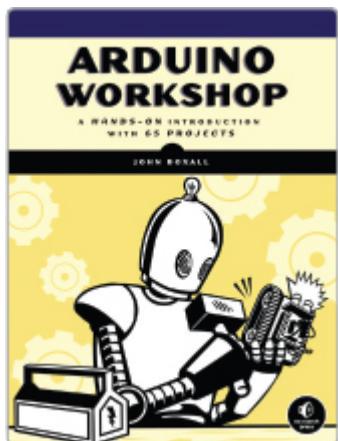


## THE ARDUINO INVENTOR'S GUIDE

Learn Electronics by Making 10 Awesome Projects

by BRIAN HUANG and DEREK RUNBERG JUNE 2017, 336 pp., \$29.95

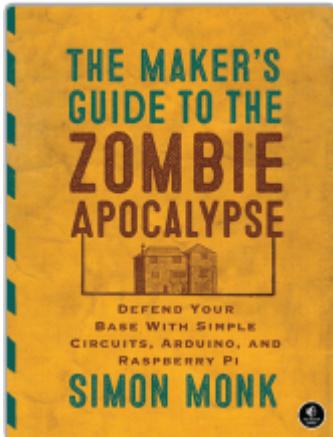
ISBN 978-1-59327-652-2 full color



## ARDUINO WORKSHOP

A Hands-On Introduction with 65 Projects

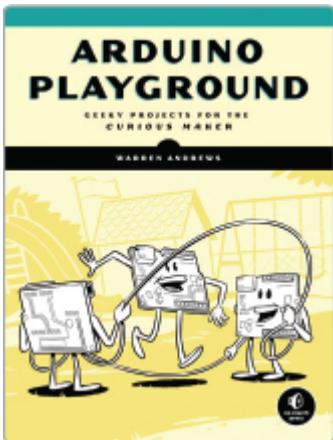
by JOHN BOXALL MAY 2013, 392 pp., \$29.95 ISBN 978-1-59327-448-1



## THE MAKER'S GUIDE TO THE ZOMBIE APOCALYPSE

Defend your Base with Simple Circuits, Arduino, and Raspberry Pi

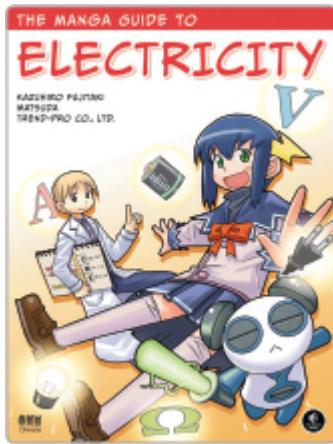
*by SIMON MONK OCTOBER 2015, 296 PP., \$24.95 ISBN 978-1-59327-667-6*



## ARDUINO PLAYGROUND

Geeky Projects for the Experienced Maker

*by WARREN ANDREWS MARCH 2017, 344 PP., \$29.95 ISBN 978-1-59327-744-4*



## THE MANGA GUIDE TO ELECTRICITY

by KAZUHIRO FUJITAKI, MATSUDA, and TREND-PRO CO., LTD. MARCH 2009, 224 PP., \$19.95 ISBN 978-1-59327-197-8

### PHONE:

1.800.420.7240 OR  
1.415.863.9900

### EMAIL:

SALES@NOSTARCH.COM

### WEB:

WWW.NOSTARCH.COM