Project Book

EXPLORING AIR QUALITY WITH DIY COLOR- AND GAS-SENSING APPARATUS

Atmospheric Monitoring with Arduino

Building Simple Devices to Collect Data About the Environment

Patrick Di Justo & Emily Gertz

O'REILLY®

Maker PRESS™

Project Book

Atmospheric Monitoring with Arduino

Makers around the globe are building low-cost devices to monitor the environment, and with this hands-on guide, so can you. Through succinct tutorials, illustrations, and clear step-by-step instructions, you'll learn how to create gadgets for examining the quality of our atmosphere, using Arduino and several inexpensive sensors.

Detect harmful gases, dust particles such as smoke and smog, and upper atmospheric haze—substances and conditions that are often invisible to your senses. You'll also discover how to use the scientific method to help you learn even more from your atmospheric tests.

- **Get up to speed on Arduino with a quick electronics primer**
- **Build a tropospheric gas sensor to detect carbon monoxide, LPG, butane, methane, benzene, and many other gases**
- **Create an LED photometer to measure how much of the sun's blue, green, and red light penetrate the atmosphere**
- **Build an LED sensitivity detector—and discover which light wavelengths each LED in your photometer is receptive to**
- **Measure light wavelengths to determine the amount of water vapor, ozone, and other substances in the atmosphere**
- **Upload your data to Cosm and share it with others via the Internet**

"The future will rely on citizen scientists collecting and analyzing their own data. The easy and fun gadgets in this book show everyone from Arduino beginners to experienced Makers how best to do that."

—Chris Anderson, Editor in Chief of *Wired* magazine, author of *Makers: The New Industrial Revolution* (Crown Business)

US $5.99 CAN $6.99
ISBN: 978-1-449-33814-5

O'REILLY®
Twitter: @oreillymedia
facebook.com/oreilly
oreilly.com

Maker PRESS™
makerpress.com

Atmospheric Monitoring with Arduino

Patrick Di Justo and Emily Gertz

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Atmospheric Monitoring with Arduino

by Patrick Di Justo and Emily Gertz

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Shawn Wallace and Brian Jepson
Production Editor: Kara Ebrahim
Proofreader: Kara Ebrahim
Cover Designer: Mark Paglietti
Interior Designer: David Futato
Illustrator: Rebecca Demarest

November 2012: First Edition

Revision History for the First Edition:

2012-11-19 First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449338145* for release details.

ISBN: 978-1-449-33814-5

[LSI]

We dedicate this book to our sisters and brothers:
Andy, Lucy, Mathius, and Melissa

Contents

Preface

There's a story (it's either an old vaudeville joke or a Zen koan) in which a fisherman asks a fish, "What's the water like down there?" and the fish replies "What is water?" If the story is just a joke, the point is to make us laugh; but if it's a koan, the point is that the most obvious and ubiquitous parts of our immediate environment are, paradoxically, often the easiest to overlook.

We as a species are probably a little bit smarter than fish: at least we know that we spend our lives "swimming" at the bottom of an ocean of air. About 4/5th of that ocean is the relatively harmless gas nitrogen. Around another 1/5 of it is the highly reactive and slightly toxic gas oxygen. The Earth's atmosphere also contains trace amounts of other harmless or slightly toxic gases like argon, carbon dioxide, and methane. And depending on where you live, it may contain even smaller, but much more toxic, amounts of pollutants like soot, carbon monoxide, and ozone.

Yet how many of us, like the fish in the koan, overlook the atmosphere? Who in your life can tell you the general composition of the air around them? How many people know what's inside every breath they take? Do you? Reading this book and building these gadgets will take you on the first steps of a journey toward understanding our ocean of air.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if this book includes code examples, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Atmospheric Monitoring with Arduino* by Patrick Di Justo and Emily Gertz (O'Reilly). Copyright 2013 Patrick Di Justo and Emily Gertz, 978-1-4493-3814-5."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

You can write to us at:

Maker Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

Maker Media is a division of O'Reilly Media devoted entirely to the growing community of resourceful people who believe that if you can imagine it, you can make it. Consisting of Make magazine, Craft magazine, Maker Faire, as well as the Hacks, Make:Projects, and DIY Science book series, Maker Media encourages the Do-It-Yourself mentality by providing creative inspiration and instruction.

For more information about Maker Media, visit us online:

MAKE: *www.makezine.com*
CRAFT: *www.craftzine.com*
Maker Faire: *www.makerfaire.com*
Hacks: *www.hackszine.com*

We have a web page for this book, where we list examples, errata, examples, and plans for future editions. You can find this page at *http://oreil.ly/atmospheric-arduino*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

1/The World's Shortest Electronics Primer

If you're a DIY electronics or Arduino novice, the information in this chapter will help you get the most out of building and programming the gadgets in this book.

If you're already building your own electronics, consider this chapter a refresher to dip into as needed.

What Is Arduino?

Arduino is best described as a single-board computer that is deliberately designed to be used by people who are not experts in electronics, engineering, or programming. It is inexpensive, cross-platform (the Arduino software runs on Windows, Mac OS X, and Linux), and easy to program. Both Arduino hardware and software are open source and extensible.

Arduino is also powerful: despite its compact size, it has about as much computing muscle as one of the original navigation computers from the Apollo program, at about 1/35,000 the price.

Programmers, designers, do-it-yourselfers, and artists around the world take advantage of Arduino's power and simplicity to create all sorts of innovative devices, including interactive sensors, artwork, and toys.

We built each of the products in this book using the Arduino Uno (Figure 1-1 and Figure 1-2), which, at the time of writing, is the latest model. By the time you're reading this, there may be something newer.

You don't have to know Arduino Uno's technical specifications to build and program the gadgets in this book, but if you're interested, you can find them at the official Arduino website.

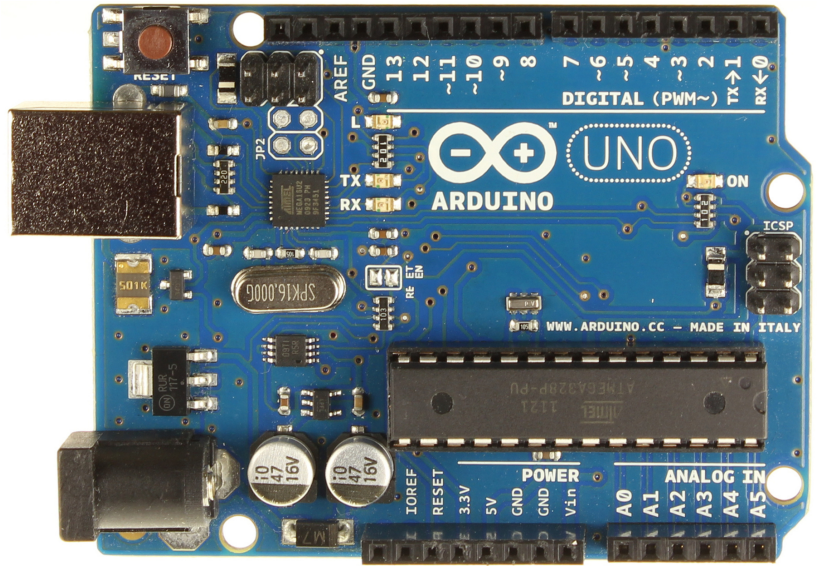

Figure 1-1. *Front of the Arduino Uno (Rev. 3).*

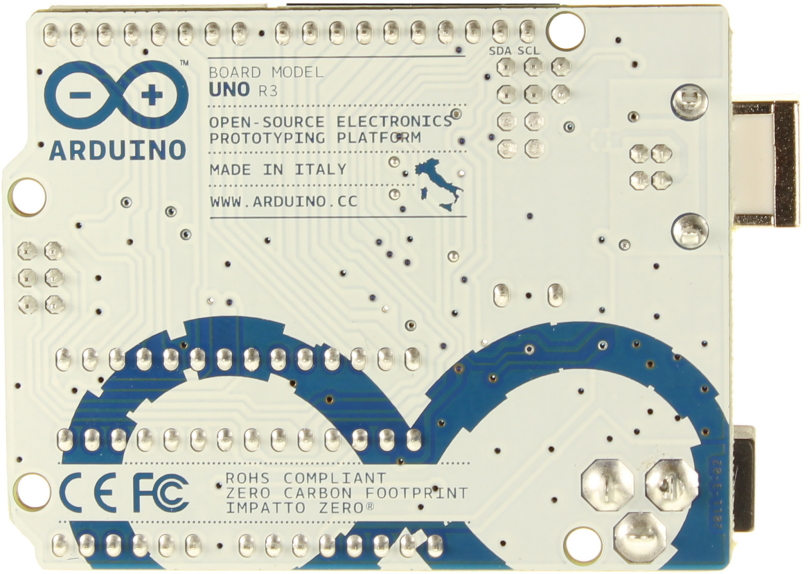

Figure 1-2. *Back of the Arduino Uno.*

Electronic Circuits and Components

An electronic circuit is, as the term implies, electricity moving in a path very much like a circle. Each circuit has a beginning, a middle, and an end (which is usually very close to where it began). Somewhere in the middle, the circuit often runs through various electronic components that modify the electrical current in some way.

Each device in this book is a circuit that combines Arduino with different electronic components. Some of these manage the power and path of the electricity, others sense certain conditions in the environment, and still others display output about those conditions.

Let's take a look at some of the components we will be using in our circuits:

Light emitting diodes (LEDs)

An LED is a lamp made of various rare-earth metals, which give off a large amount of light when a tiny current is run through them. The composition of the substances within the LED determine the particular wavelength of light emitted: you can buy green, blue, yellow, red, even ultraviolet and infrared LEDs.

Technically, the LEDs used in our gadgets are "miniature LEDs," tiny lamps with two wire leads: one long (called the anode) and the other a bit shorter (called the cathode). These come in various useful forms (including single lamps from 2 mm to 8 mm in diameter, display bars, and alphanumeric readouts) and can serve as indicators, illuminators, or even data transmitters.

You'll learn how to use these different types of LEDs while building the different environmental sensors in this book.

Resistors

Resistors are the workhorses of the electronics world. What do resistors do? They simply resist letting electricity flow through by being made of materials that naturally conduct electricity poorly. In this way, resistors serve as small dumb regulators to cut down the intensity of electric current.

Resistance is valuable because some electronic components are very delicate: they burn out easily if they're powered with too much current. Putting a resistor in the circuit ensures that only the proper amount of electricity reaches the component. It's hard to imagine any circuit working without a resistor, and with LEDs, resistors are almost mandatory.

While building the projects in this book, you'll learn various creative ways to regulate current with resistors.

Soldering

Soldering involves heating up conductive metal, called *solder*, and then using it to fuse other pieces of metal together. In small-scale electronics, we use an electrical tool called a soldering iron, which has a small tip, to heat up thin wires of solder and drip the solder onto the components we wish to join into the circuit.

Soldering creates a very stable circuit, and that stability can be a drawback. Fusing together components can make it difficult to reuse or reconfigure circuits. You also must be very careful to not short-circuit components while soldering. Knowing how to solder can be a very useful skill in DIY electronics. If you're interested in learning how, this online resource is a good place to start.

The alternative to soldering is to use a solderless breadboard.

Solderless breadboards

Solderless breadboards are small plastic boards studded with pins that can hold wires (more about these next). These wires can then be connected to other electronic components, including Arduino.

Solderless breadboards make it much easier to design circuits, because they allow you to quickly try out various assemblies and components without having to solder the pieces together. While solderless breadboards typically are intended for use only in the design phase, many hobbyists keep a breadboard in the final version of a device because they're so fast and easy to use.

If you don't feel like soldering circuit boards, solderless breadboards are the way to go. Each gadget in this book uses a solderless breadboard.

Wire

Wire is the most basic electronic component, creating the path along which electrons move through a circuit. The projects in this book use 1 mm "jumper wires," which have solid metal tips perfectly sized to fit into Arduino and breadboard pins, and come sheathed in various colors of insulation.

Get as much jumper wire as you can afford, in several colors. When building circuits with Arduino, you can't have too many jumper wires.

We order most of our electronics components from these online retailers:

- Adafruit Industries
- Eemartee
- Electronic Goldmine
- SparkFun

Maker Shed, from MAKE and O'Reilly Media, sells books, kits, and tools, as well as many of the components needed to build the projects in this book including Arduino, breadboards, sensors, and basic electronic components. Maker Shed also supplies convenient bundles for many of the projects in this book (you can find more information about these bundles in the individual project chapters).

Don't count out your friendly local RadioShack, though. While writing this book, more than once we ran out to RadioShack for a last-minute component.

For years RadioShack cut back on its electronic components inventory, apparently seeing a better future for the business by featuring cell phones and other consumer electronics. But the company has recently begun to embrace the maker movement; at the time of writing, most of their stores around the country are even carrying Arduinos. We're hopeful RadioShack is on the return path to being the hacker heaven it was years ago.

Programming Arduino

A computer program is a coded series of instructions that tells the computer what to do. The programs that run on Arduino are called *sketches*.

The sketches used in this book mostly tell Arduino to read data from one of the pins, such as the one connected to a sensor, and to write information to a different pin, such as the pin connected to an LED or display unit.

Sometimes the sketches also instruct Arduino to process that information in a certain way: to combine data streams, or compare the input with some reference, or even place the data into a readable format.

An Arduino program has two parts: `setup()` and `loop()`.

setup()
: The `setup()` part tells Arduino what it needs to know in order to do what we want it to do. For example, `setup()` tells Arduino which pins it needs to configure as input, which pins to configure as output, and which pins won't be doing much of anything. If we're going to use a special type of output to show our results, such as an LCD display, `setup()` is where we

tell Arduino how that output works. If we need to communicate with the outside world through a serial port or an ethernet connection, `setup()` is where we put all the instructions necessary to make that connection work.

loop()
: `loop()` tells Arduino what to do with the input or output. Arduino runs the instructions in `loop()`, then goes back to the top of `loop()` and runs them again. And again. And again. `loop()` continues to loop as long as the Arduino has power.

First Sketch: Make an LED Blink

By long tradition (going back to 2006), the first Arduino sketch you will write is to make an LED blink.

Arduino pins can be used for input and output, as long as you tell the computer which is which. So in this sketch, we tell the Arduino to set pin 13 to be the LED OUTPUT pin, and then we alternately send electricity to pin 13 (setting the pin HIGH) and cut off the electricity to pin 13 (setting the pin LOW). With each alternation, the LED turns on and off.

We'll write all the sketches in this book using the Arduino *integrated development environment* (IDE), which, simply put, is special software for writing and uploading code to Arduino.

Parts

1. Arduino Uno
2. Breadboard
3. LED

Install the IDE

Download the Arduino IDE, and follow the provided instructions to install it on your computer.

Once you've installed the software, open the IDE. You should see a screen that looks something like Figure 1-3.

Breadboard the Circuit

The circuit portion of this project is very simple: take an LED and place the long lead into pin 13 on Arduino, as you can see in the Figure 1-4 breadboard view.

Figure 1-3. *The Arduino IDE on a Mac.*

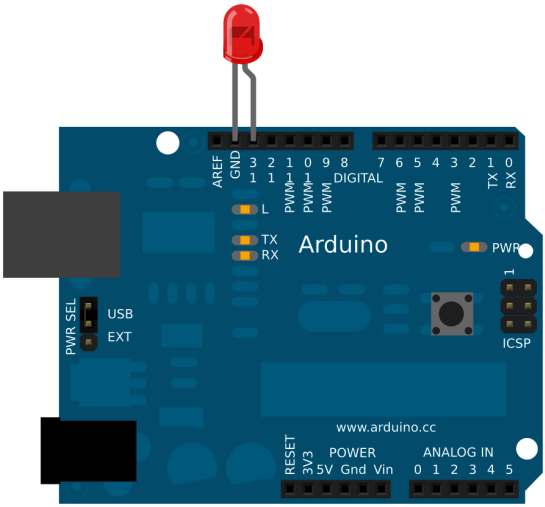

Figure 1-4. *LED long lead inserted into pin 13 on the Arduino (image made with Fritzing).*

Write the Code

You can find this code in the Arduino IDE under File → Examples or on the EMWA GitHub Repository → chapter-1 → blink.

```
/*
  Blink
  Turns on an LED for one second,
  then off for one second, repeatedly.
  This example code is based on example code
  that is in the public domain.
*/

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Normally, you'd need to put a resistor in between the power source and the LED, so as not to burn out the LED. Arduino Unos (and later models) have a resistor built into pin 13, so that's taken care of.

In this sketch, the code in `loop()` simply tells Arduino to set pin 13 HIGH—taking it up to 5 volts—for 1,000 milliseconds (one second), followed by setting it LOW—taking it down to 0 volts—for another 1,000 milliseconds.

Do you notice the /* … */ sections and the // lines in the example above? Those are ways to put comments into your code to explain to others (and to yourself) what the code does:

- /* and */ tell the computer that everything between those marks should be ignored while running the program.

- // tells the computer that everything afterward on that line is a comment.

Why Comment Code?

Commenting code simply means adding explanations in plain English to your sketch that describe how the code works. Adding comments to code is a very good idea. Here's why:

Suppose, after hours trying to get your Arduino to do something, the solution suddenly comes to you. Eureka! You hook up your Arduino, bang out your code, load it up, and voilà: it works.

Fast forward: months later, working on another project, you want your Arduino to do something similar to your earlier project. "No sweat, I'll just reuse my earlier code," you think. But you open up the sketch and ... none of it makes sense!

You wrote that earlier code in a highly creative state of mind, when your brain chemicals were flowing like a river and your ideas were flashing like summer lightning. In all the excitement, you didn't comment your code. So now, months later, when you're in a completely different state of mind, you can't remember what the code does, and you have to start all over. Is that any way to live?

If you had commented your code from the beginning, you'd know exactly what each variable was used for, what each function did, and what each pin controlled. Your life would be so much more enjoyable.

In short, always take a few minutes to comment your code.

Things to Try

Modify this sketch to make the LED do something different:

1. Blink twice as quickly.
2. Blink twice as slowly.
3. Light up for half a second with a 2-second pause between blinks.

Congratulations, you're an Arduino programmer! Now let's have some real fun.

2/Gadget: Tropospheric Gas Detector

We can easily go several hours without drinking water. We can comfortably go the better part of a day without eating food. But try and go more than a few minutes without breathing. (No, don't really try.) Understanding the composition of the lower atmosphere—the troposphere—is among the most important environmental measurements we can take.

Everything floating around the troposphere—nitrogen, oxygen, carbon dioxide, water vapor, and all sorts of pollution—winds up in our lungs, on our plants, in our food, and in our water (see Figure 2-1). It dusts our windows, our automobiles, and our buildings. For this reason, the authors (as well as organizations like the American Lung Association) believe that it's vitally important to know what's inside every breath we take.

In the old days, when people wanted to know what was in the atmosphere, they used chemically-treated filter paper, and hung it in a breeze. The chemicals reacted with whatever was in the air and would respond by changing color. Or they bubbled the atmosphere through water and measured the different compounds that resulted as gas dissolved in water. This kind of work could only be performed in a dedicated chemistry lab.

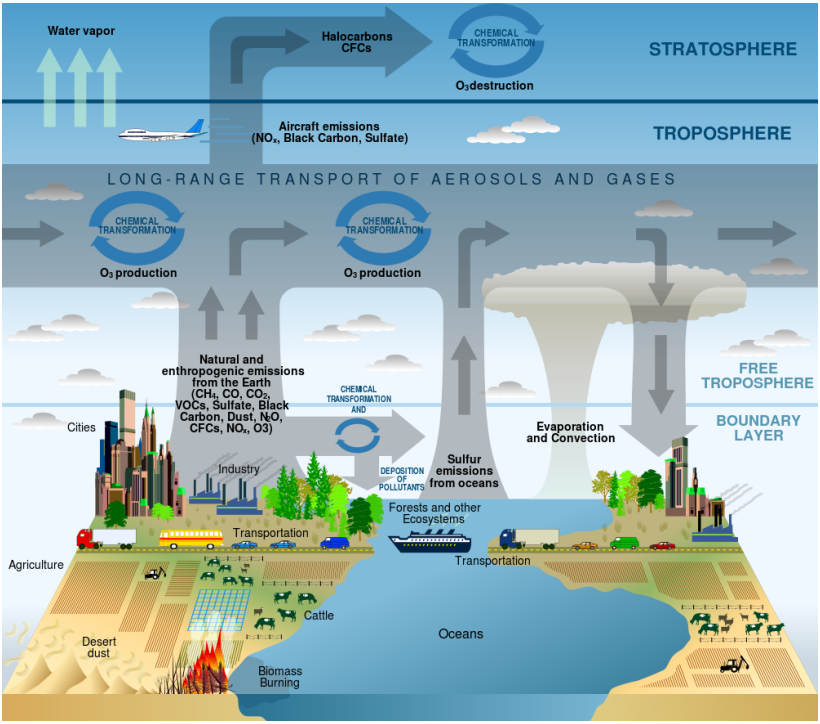

Figure 2-1. *This illustration shows how different chemicals and other substances move into and through the troposphere. Credit: U.S. Climate Change Science Program, 2003.*

Fortunately for us, we can now purchase a small, complete atmospheric laboratory for less than $10, in the form of an electronic gas sensor (Figure 2-2). These sensors detect different substances in the atmosphere by measuring the changing resistance of a film made of tin dioxide.

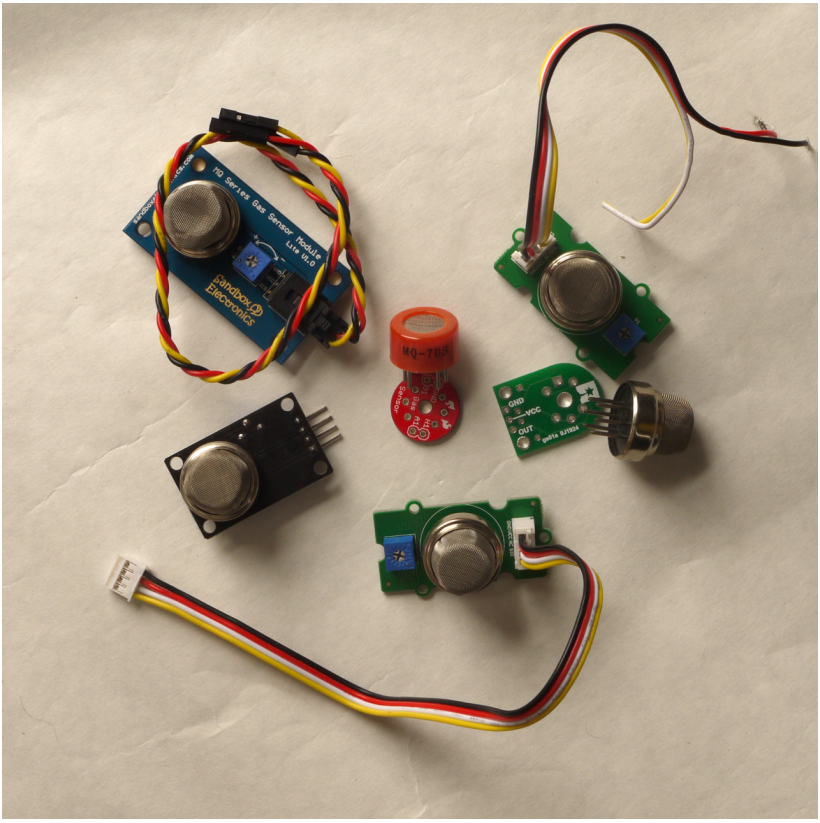

Figure 2-2. *There are lots of inexpensive sensors on the market that can be used for DIY monitoring.*

How Gas Sensors Work

Oxygen in the atmosphere removes electrons from the tin dioxide film, which decreases its conductivity (and increases its resistance). When other types of gases, particularly those that are chemically reducing, touch the tin dioxide film, electrons are injected into the material. This *increases* the conductivity (and *lowers* the resistance) of the tin dioxide layer. You can use your Arduino to measure that change in resistance.

It's important to keep in mind that tin dioxide sensors tend to be broadly selective. While certain sensors may be marketed as being "alcohol" sensors or "carbon monoxide" sensors, they actually respond to more than just alcohol or carbon monoxide, respectively; they respond to a wide family of

similar gases. Manufacturers can make the tin dioxide-based gas sensors more selective by adding various catalysts into the sensor head, or by using external filters. The datasheet provided with each sensor explains more completely how to adjust the sensitivity of each sensor for various gases.

Which Gases Can We Monitor?

There are electronic sensors for a wide range of gases. As we write this book in the summer of 2012, there are easy-to-use inexpensive sensors on the market to detect carbon monoxide, carbon dioxide, liquid petroleum gas, butane, propane, methane (natural gas), hydrogen, ethyl alcohol, benzene, volatile organic compounds, ammonia, ozone, hydrogen sulfide, and more. It's not unreasonable to expect that it won't be long before cheap sensors hit the market that can detect nitrogen oxides and other contaminants. All of these gases count as pollutants; in varying concentrations, all of them can be harmful.

How This Gadget Works

We're going to use the MQ-2 and MQ-6 sensors from Hanwei, Inc. in this gadget. Both detect combustible gases: the MQ-6 detects butane and liquefied petroleum gas (LPG), also called propane (both hydrocarbons), while the MQ-2 is sensitive to LPG, methane (the primary component of natural gas, and a potent greenhouse gas), and smoke. We feel that both sensors together are a great way to start measuring ground-level air pollution.

What Is Smoke?

Smoke, for our purposes, is defined as a byproduct of incompletely burned carbon-based fuel. It includes solid and liquid particulate matter (otherwise known as soot), as well as some gaseous remnants of the original fuel mixed with air. Hanwei's datasheet for the MQ-2 does not specify what "smoke" means for this sensor. But since we know that the MQ-2 detects certain hydrocarbon gases, we're assuming that the smoke it detects is also hydrocarbon-based: a component of automobile or truck exhaust, or the burning of natural gas.

A heating element in the electronic circuit heats the metal, making it more reactive with atmospheric gases. As the various gases react with the metal, the resistance changes in proportion to the amount of that gas present in the air exposed to the sensor. This change in resistance is measured by the Arduino analog port. That's basically it.

If we plug the heater directly into Arduino, we find ourselves with a problem. The heater consumes 800 mW, which works out to equal 200 mA (.8 W / 5

V = .2 A). A standard Arduino pin can only reliably source 20 mA (in other words, only about 10% of the power the heater needs). We have correspondence from the manufacturers indicating that the heater *can* be powered by connecting it to the +5 V Arduino pin, but frankly, we're skeptical. We've got to come up with a way to use Arduino to *control* the amount of power that goes to the heating units, so that the heating unit is not on constantly, without actually having Arduino *provide* that power.

Both these problems—providing power to the heater and controlling that power—have a single solution, probably the greatest invention of the 20th century: the transistor.

Transistorized!

Transistors are used to amplify electronic signals cheaply and efficiently, with very little noise, while giving off very little heat (Figure 2-3). Transistors also act as tiny, efficient digital switches. Since all computer activity breaks down into a series of binary "on and off" states represented by 1s and 0s, transistors by the millions, embedded into a silicon chip, control those on and off signals.

We really can't overstate the importance of the transistor. We don't have room in this book to discuss the details of how transistors work; suffice it to say that the lightweight, cheap electronic gadgets in our lives—handheld cell phones, computers, digital cameras, flat screen TVs, microwave ovens, cable television, touchtone phones, simple portable AM/FM radios, essentially anything more complicated than a flashlight—would be impossible without the transistor.

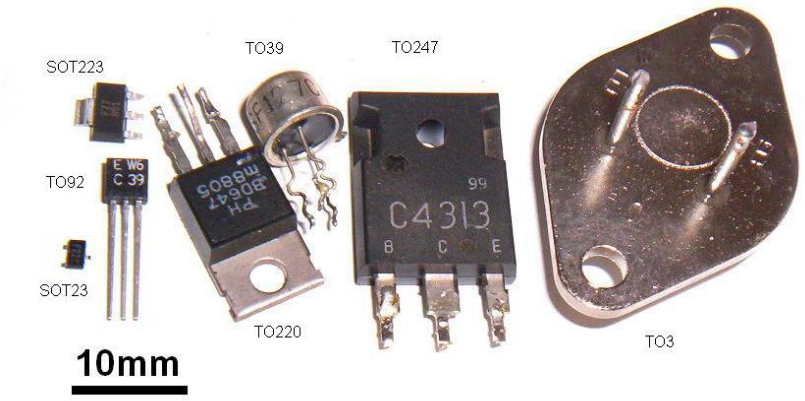

Figure 2-3. *Various transistors. Source: Ulf Seifert.*

The first thing you notice when you look at a transistor is that unlike almost every other electronic device we've seen so far, a transistor has three terminals. We can control the voltage between two of the terminals by applying a specific electric current or voltage to the third terminal. The three terminals are the base, the collector, and the emitter. The base is the controller; voltage applied here determines whether or not electricity flows from the collector to the emitter. The collector is the "source" of the electrical current, and the emitter is the output.

If we were to send varying levels of current from the base, we can regulate the amount of current flowing from the collector to the emitter. This is how a transistor acts as an amplifier: a very low signal coming into the base is repeated at a much larger voltage provided by the collector.

When we use a transistor as a switch, the circuitry is even simpler. A transistor switch is either fully on or fully off. A small data signal to the base determines whether the transistor is switched on or off. When it is switched on, current flows between the ground and the collector. This simple setup lets us use Arduino to turn on components that have a separate power supply.

Build the Gadget

Amount	Part Type	Properties/(Assembly Code)
2	1 k Ω resistor	Package THT; tolerance 5%; bands 4; resistance 1 k Ω; pin spacing 400 mil (R1 & R2)
1	Voltage regulator, 5 V	Package TO220 [THT]; voltage 5 V (U1)
2	NPN-transistor	Package TO92 [THT]; type NPN (Q1 & Q2)
1	Arduino UNO R3	(Arduino1)
1	LCD screen	Character type, 16 pins (LCD1)
1	Battery block 9 V	(VCC1)

1. Connect a wire from the GND pin of Arduino to the GND rail of the breadboard. Connect the GND rail of the breadboard to the EMITTER pin of the transistor (Figure 2-4).
2. Connect the BASE pin of the transistor to a 1 K resistor, and connect the resistor to an Arduino digital pin (Figure 2-5).

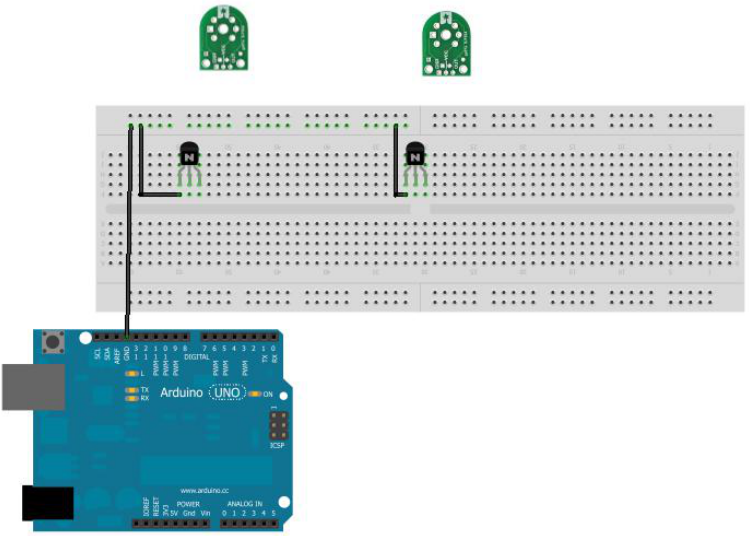

Figure 2-4. *Step one. Source: these images were created with Fritzing.*

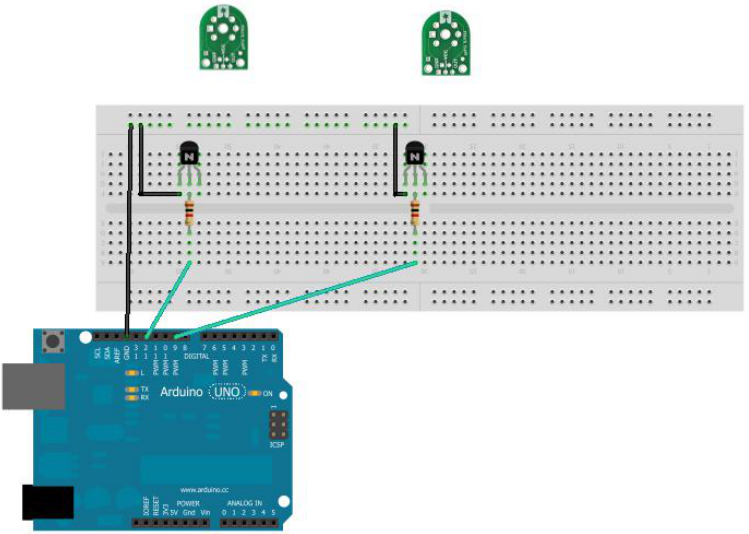

Figure 2-5. *Step two.*

3. Connect the COLLECTOR pin of the transistor to the GND pin of the sensor (Figure 2-6).

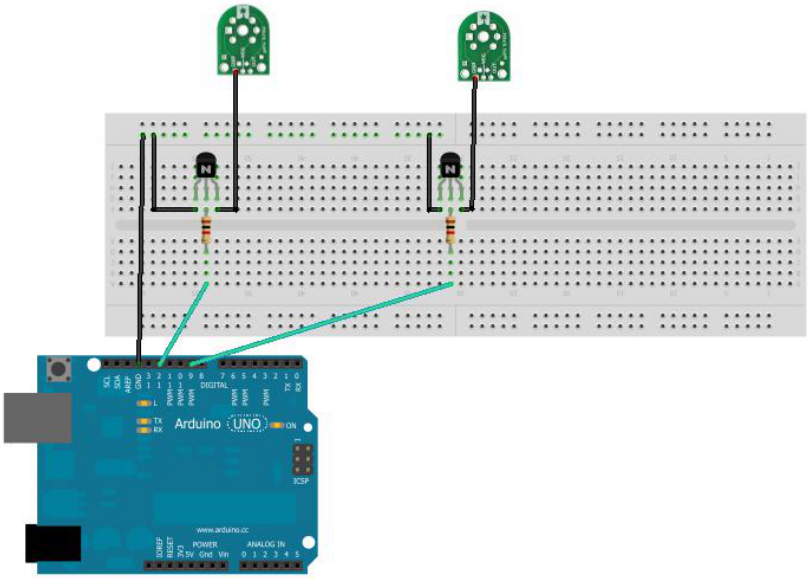

Figure 2-6. *Step three.*

4. Connect the +5 (VCC) sensor pins to the breadboard's power rail (Figure 2-7). Don't worry; we're going to add a power supply later.
5. Connect the data lines from the sensors to Arduino analog ports 4 and 5 (Figure 2-8).

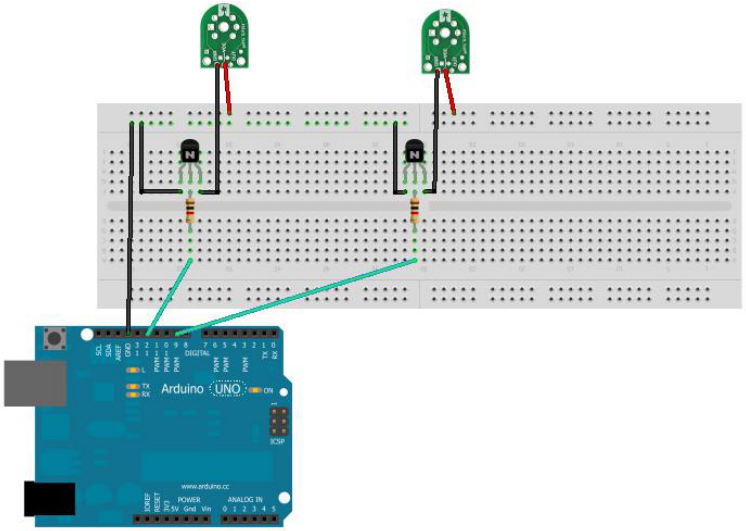

Figure 2-7. *Step four.*

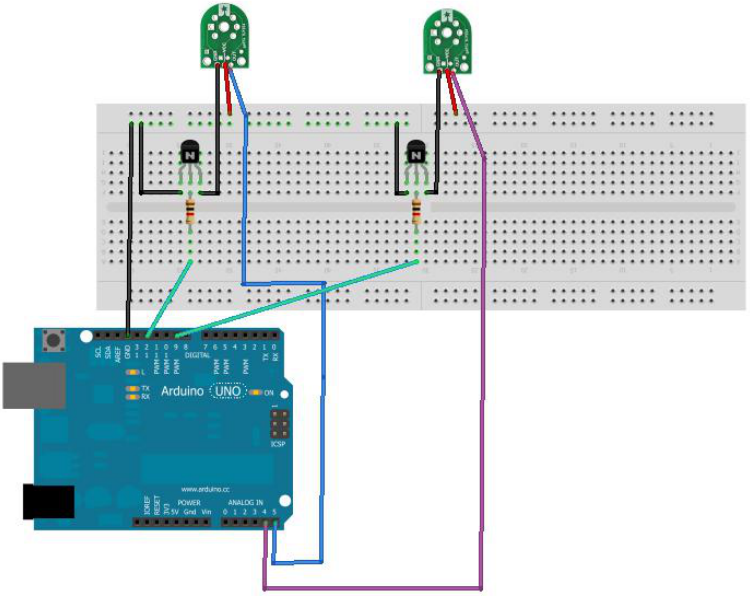

Figure 2-8. *Step five.*

6. Connect the 7805 +5 VDC voltage regulator. This regulates the voltage coming from your independent power source for the sensor heaters. Like the transistor, the voltage regulator also has three terminals: a center pin goes to GND, the pin on the left is input, and the pin on the right is output (Figure 2-9).

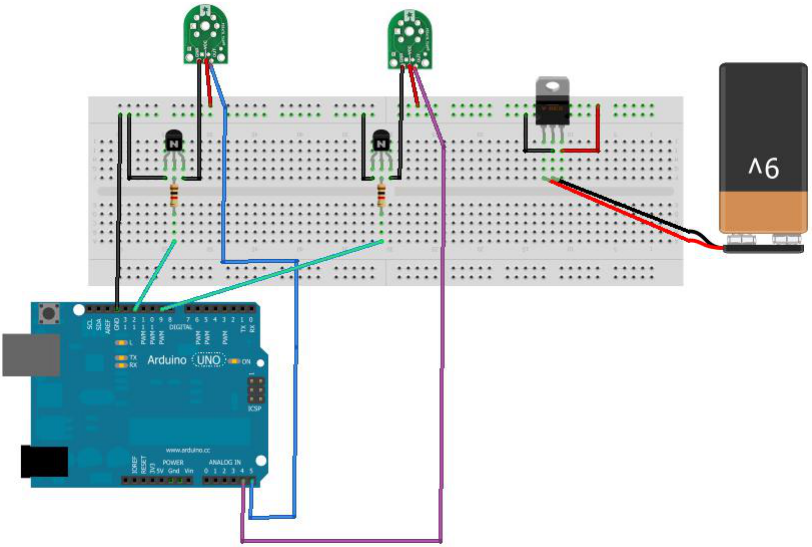

Figure 2-9. *Step six.*

7. Connect the GND pin to the GND rail of the breadboard AND to the black (or –) wire on your power supply. Connect the input pin to the red (or +) wire on your power supply. Connect the output pin to the power rail on the breadboard (Figure 2-10).

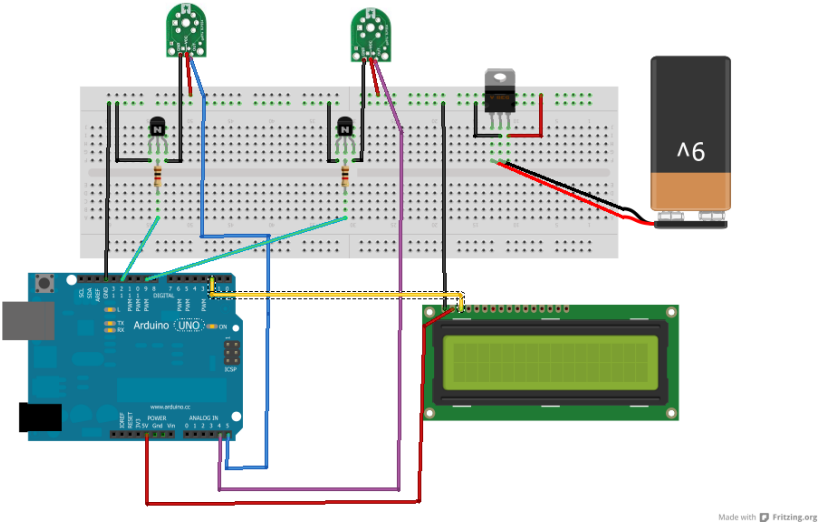

Figure 2-10. *Step seven.*

This device takes an input current (up to 35 volts), and changes it to a stable, fixed +5 VDC. In our example, we're using a standard 9 volt battery at the current source, but you can use just about anything: a 6 volt lantern battery, two 3.7 volt lithium polymer batteries connected in series, even a bunch of AA batteries. The total capacity of your power source should be your main determining factor: smaller batteries generally provide fewer amp-hours, meaning that the lifespan of your gadget can be cut short if you run out of power.

Don't connect the battery until you actually need it, or add an ON-OFF switch to power it up when you're ready to start taking readings.

Optionally, connect the LCD. The data line is Arduino pin 2, ground goes to GND, and the power supply is Arduino's 3.3 V pin.

Load the Sketch

You can find this sketch in the AMWA GitHub repository.

```
#include <SoftwareSerial.h>
#include <SD.h>
#include <EEPROM.h>
#include<stdlib.h>

// Liquid Crystal Display
```

```
// Define the LCD pins: We'll be using a serial-based LCD display
// which only required +3.3Volts, GND, and a single data line.
// databuff and displaybuff hold the data to be displayed
#define LCDIn 2
#define LCDOut 5
SoftwareSerial mySerialPort(LCDIn, LCDOut);


// Data Buffers for the LCD
char databuff1[16];
char databuff2[16];
char dispbuff[16];

// GAS SENSORS
// Analog input pin that reads the first gas sensor
const int gasPin1 = A5;

// Analog input pin that reads the gas sensor
const int gasPin2 = A4;

// The digital pin that controls the heater of gas sensor 1
const int heaterPin1 = 7;

// The digital pin that controls the heater of gas sensor 2
const int heaterPin2 = 9;

// LED connected to digital pin 13
const int ledPin = 13;

// value read from the sensor A5
int gasVal1 = 0;

// value read from the sensor A4
int gasVal2 = 0;

long warmup = 180000;             // enter time for heaters to warmup, in
milliseconds.
// 180,000 milliseconds = 3 minutes

long downtime = 360000;           // enter delay between readings, in
milliseconds.
// 360,000 milli seconds = 6 minutes

//EEPROM records require two bytes to store a 1024 bit value.
//Each gas sensor returns a value from 0-1024, taking 2 bytes.
//To store gas sensor data would require a record index,
//plus two bytes for the first gas sensor, two bytes for the second gas
sensor
//For a total of five bytes per record.

// current EEPROM address
int addr =0;

//EEPROM record number
int record = 0;
```

```
//EEPROM record length
int reclen = 5;

//switch to tell if an SD card is present
int SDPresent = 1;

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);

  pinMode(heaterPin1, OUTPUT);  // sets the digital pins as output
  pinMode(heaterPin2, OUTPUT);
  pinMode(LCDOut, OUTPUT);

  //reset the LCD
  mySerialPort.begin(9600);
  mySerialPort.write(0xFE);
  mySerialPort.write(0x01);
  sprintf(databuff1,"Wakeup Test");
  sprintf(dispbuff,"%-16s",databuff1);
  mySerialPort.print(dispbuff);

  // Set up SD card, let us know if SD card is absent
  pinMode(10, OUTPUT);
  if (!SD.begin(4))
  {
    SDPresent =0;
    sprintf(databuff2,"NO SD CARD!!!");
    sprintf(dispbuff,"%-16s",databuff2);
    mySerialPort.print(dispbuff);
    Serial.println("NO SD CARD!!!");
    delay(6000);
  }
  delay(3333);
}

void loop()
{
  long scratch=0;   // scratch variable

  // set the timer
  unsigned long counter = millis();

  //turn first heater on
  digitalWrite(heaterPin1, HIGH);

  // wait 3 minutes for heater to heat up
  while(millis() < (counter + warmup))
  {
    sprintf(databuff1,"Unit1 Activated");
    sprintf(dispbuff,"%-16s",databuff1);
    mySerialPort.print(dispbuff);
```

```
  scratch = (int)((counter+warmup - millis())/1000);
  sprintf(databuff2,"Countdown: %3d", scratch);
  sprintf(dispbuff,"%-16s",databuff2);
  mySerialPort.print(dispbuff);

  Serial.println(scratch);
}

// read the analog in value:
gasVal1 = analogRead(gasPin1);
sprintf(databuff1,"read unit 1");
sprintf(dispbuff,"%-16s",databuff1);
mySerialPort.print(dispbuff);

// shut off the first heater
digitalWrite(heaterPin1, LOW);

//turn second heater on
digitalWrite(heaterPin2, HIGH);
sprintf(databuff2,"turning on unit2");
sprintf(dispbuff,"%-16s",databuff2);
mySerialPort.print(dispbuff);

// wait 3 minutes for heater to heat up
while(millis() < (counter + warmup + warmup))
{
  sprintf(databuff1,"Unit2 Activated");
  sprintf(dispbuff,"%-16s",databuff1);
  mySerialPort.print(dispbuff);

  scratch = (int)((counter+warmup+warmup - millis())/1000);
  sprintf(databuff2,"Countdown: %3d", scratch);
  sprintf(dispbuff,"%-16s",databuff2);
  mySerialPort.print(dispbuff);

  Serial.println(scratch);
}

// read the analog in value:
gasVal2 = analogRead(gasPin2);
sprintf(databuff2,"reading unit2");
sprintf(dispbuff,"%-16s",databuff2);
mySerialPort.print(dispbuff);

// shut off the second heater
digitalWrite(heaterPin2, LOW);

//Display on LCD
sprintf(databuff1,"Gas1:%4d",gasVal1);
sprintf(dispbuff,"%-16s",databuff1);
mySerialPort.print(dispbuff);
sprintf(databuff2,"Gas2:%4d",gasVal2);
sprintf(dispbuff,"%-16s",databuff2);
mySerialPort.print(dispbuff);
```

```
//write to SD card
  if(SDPresent = 1)
  {
    writeDataToSD(databuff1, databuff2);
  }

  //Wait downtime and start again
  //to make more frequent measurements, change value of downtime
  while(millis() < (counter +downtime))
  {
  }
}

void writeDataToSD(String dataString1, String dataString2)
{
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File dataFile = SD.open("datalog.txt", FILE_WRITE);

  // if the file is available, write to it:
  if (dataFile)
  {
    Serial.println("Hooray, we have a file!");
    dataFile.print(millis());
    dataFile.print(",");
    dataFile.print(dataString1);
    dataFile.print(",");
    dataFile.println(dataString2);

    dataFile.close();

    // print to the serial port too:
    Serial.print(millis());
    Serial.print(",");
    Serial.print(dataString1);
    Serial.print(",");
    Serial.println(dataString2);

    //Print to LCD
    mySerialPort.print("Datafile written");
  }
}
```

Displaying and Storing Your Data

You can connect Arduino to components that display data, as well as those that store data for later use.

Liquid Crystal Displays

Liquid crystal displays (LCDs) are cheap and easy ways to display data, status, warnings, and other messages from Arduino. They come in many different colors: you can buy LCDs with amber characters on a black back-

ground, black characters on a green background, yellow characters on a blue background, and other color combinations. Some LCDs have two rows of 16 characters, others four rows of 20 characters, and other display combinations are available as well. But for our uses, the biggest differences in LCDs involve the way they handle data.

The most basic (and least expensive) LCDs make you do all the data handling. They can take up as many as 10 digital data pins (most Arduinos only have 13), and might even require you to design your own characters. Some makers love doing stuff like that, but others just want to plug in a device and have it work.

For our uses, we've decided to go with a serial-controlled LCD, one in which a small microprocessor attached to the LCD takes care of all the data and character management. It's more expensive, but also much easier to use. All we need to do is ground the device, give it some power, and feed it data.

Step seven of the build explains how to connect the LCD to the tropospheric gas detector.

Reading Data Off EEPROM

You might have noticed in the code that Arduino writes the data it recieves to something called EEPROM. This stands for "Electrically Erasable Programmable Read-Only Memory." This is a type of computer memory that is nonvolatile; it remains in place after Arduino is powered down, or after a new program is loaded. EEPROM is perfect for storing data that has to last a long time (a long time by Arduino standards, that is), such as months or years. Our gadget uses 5 bytes of EEPROM to store a single observation record: 1 byte for the record number, and 2 bytes apiece for each sensor's data.

We've included a small program to extract tropospheric gas data from the Arduino's EEPROM. Simply connect your Arduino to your computer via the USB cable, upload the following sketch, and then view the serial monitor.

```
#include <EEPROM.h>

// start reading from the first byte (address 0) of the EEPROM

int address = 1;
int record = 0;
unsigned int Sensor1 = 0;
unsigned int Sensor2 = 0;

int q;
int m;

void setup()
{
  Serial.begin(9600);
  Serial.println("Record#, Sensor1, Sensor2");
```

```
  for(int i =0; i<=95; i++)
  {
    readData();
  }
}

void loop()
{
  // There's no need for this program to loop.  We know what we want to do,
  // and we just did it in setup().
}

void readData()
{
  record = EEPROM.read(address++);
  Serial.print(record);
  Serial.print(",");
  q = EEPROM.read(address++);
  delay(20);
  m = EEPROM.read(address++);
  delay(20);
  Sensor1 = (q*256)+m;

  Serial.print(Sensor1);
  Serial.print(",");
  q = EEPROM.read(address++);
  delay(20);
  m = EEPROM.read(address++);
  delay(20);
  Sensor2 = (q*256)+m;

  Serial.print(Sensor2);
  Serial.print(",");
  q = EEPROM.read(address++);
  delay(20);
  m = EEPROM.read(address++);
  delay(20);

 }
```

The result is a CSV display of your data, ready to be copied and pasted into your favorite spreadsheet program.

The Limits of EEPROM

EEPROM is not limitless. The Arduino Duemilanove and Uno each have a single kilobyte of EEPROM available, which (at 5 bytes per observation record) will hold about 200 observations worth of data. At 5 observations per hour, that's enough for more than a day and a half of solid observation. Arduino Megas have 4 kilobytes of EEPROM, about enough to hold a week's worth of atmospheric data. The old Arduino NGs, Nanos, and Diecimilas have a paltry 512 bytes of EEPROM, about 20 hours' worth.

What do you do when your EEPROM is full? The Arduino IDE has a sketch called EEPROM_Clear. Simply run that, and it will wipe your Arduino's non-volatile memory and make it ready for the next season's worth of data. Of course, you will have already backed up your data to a spreadsheet, a hard drive, an SD card, or Cosm before you wipe your EEPROM, right?

Reading Data from an SD Card

If you are storing your gas detector data on the SD card, you'll find a file called *DATALOG.TXT*. That file contains your measurements in CSV (comma separated value) format. To save space on Arduino, we did not massage the data in any way. You can open this file in your favorite spreadsheet program and work with it as desired.

Things to Try

Of course, there's not just one way to build a gas detector. There are many different configurations you can try (such as more or different sensors) that will provide more detailed data, but what changes (if any) will you have to make to the circuit to be sure it works properly? Will you need to beef up the batteries, or try a totally new power source? Here are some other ideas to make your gas detector more versatile.

Other Sensors

Does hot humid air hold more pollution than cold dry air? If you add a temperature/humidity sensor to the gadget, can you detect any correlations between temperature/humidity and toxic gas concentration?

As we noted earlier, Hanwei and other manufacturers offer many different gas sensors, often (but not always) built around the same basic circuitry, making it relatively easy to add more sensors to a gadget, or swap new sensors for old ones. It might be interesting to add an MQ-131 ozone sensor to the MQ-2 and MQ-6 sensors—is there a correlation between automobile pollution and ozone? between temperature/humidity and ozone?

Solar Powered

One drawback to the current gadget is the batteries last only a few days. Is it possible to have the gadget work forever by powering it with the sun?

Yes, with some caveats. The heating elements of the gas sensors are real power hogs, and running this device totally on solar power might or might not be practical depending on the size of the solar panel, the amount of

sunlight at your location, and the number of gas sensors you have. If you want to give it a try, we'd recommend starting with something like the Adafruit USB/SOLAR LiPoly charger. This device will handle all the power management for you.

The key question is the size of the solar panel: if you calculate the amount of power your Arduino and sensors use, and compare it to the net power you can expect from the solar panel/charger (taking into account your location and time of year), you can get a good idea if solar will work for your gadget.

GSM

Wouldn't it be great if, instead of having to fetch data from your gadget, you could make your gadget send you its data? You can, with a GSM module. This device, available from all the usual suppliers like Sparkfun and Adafruit, is essentially the guts of a cell phone without the keypad, display, speaker, microphone, or ringtones. By connecting it in place of (or along with) the LCD display, you can have your gadget send you text messages (or even post on Twitter!) its latest findings. You'll need to get a separate SIM card with a data plan, but these usually cost in the range of $5 to $10 per month.

Do Not Deploy Your Gadget in Public Without Official Permission

Now we come to the strangest part of this book: the warning not to leave your gas detector unattended, anywhere, ever. Here's why:

In May 2012, Takeshi Miyakawa, a visual artist and furniture designer in New York City, placed a portable battery-powered light inside a translucent "I Love NY" plastic shopping bag, and hung it from a metal rod attached to a tree. When the lamp was on, the bag glowed from within. A friend of the artist said he did this "to lift people's spirits. He was simply trying to say that he loves the city and spread that attitude around."

Unfortunately for Miyakawa, someone in the neighborhood saw a contraption with wires, batteries, and plastic boxes and called the New York Police Department's bomb squad, which evacuated the area and took two hours to determine that a battery and a light was just a battery and a light. When the police tracked the device to Miyakawa, they arrested him and charged him with two counts of first-degree reckless endangerment, two counts of placing a false bomb or hazardous substance in the first degree, two counts of placing a false bomb or hazardous substance in the second degree, two counts of second-degree reckless endangerment, and two counts of second-degree criminal nuisance. A judge ordered him sent to Bellevue hospital for psychiatric evaluation, as well.

All for hanging a light in a tree.

Miyakawa's experience is not unique, unfortunately. In another case, researchers from Carnegie-Mellon University built similar detectors to ours, and placed them at various points around the Pittsburgh area. While the sensors were enclosed in professionally made plastic cases, and were clearly labeled as an approved research project, once again the police bomb squad was called. (The students didn't report being arrested. However, they did say it took a great deal of negotiation with the authorities to get their equipment back.)

The US Department of Homeland Security (DHS) urges people to "be vigilant, take notice of your surroundings, and report suspicious items or activities to local authorities immediately." As a result, homemade electronic gadgets that once upon a time (at least, in the lifetimes of the authors) might have been regarded as interesting curiosities are now treated as threats. Many people don't understand hobby electronics, and are frightened by what they do not understand. Police believe they must treat every suspicious package as if it may be harmful.

The point of telling you all this is not to scare you from building monitoring devices, but to warn you that a home-built, Arduino-based gas sensor might, to the unknowing eye, look like a box of batteries and wires and a scary thing with blinking lights. Leaving something like this unattended, without permission, on property you don't own, can get you into a world of trouble.

The following subsections are a couple ideas for solving this problem.

Get Official Permission

Look into official channels through which you can deploy experimental packages in public. At A.M.W.A. World Headquarters in New York City, we looked to the Parks Department, which allows researchers to leave experimental devices in some of the city's parks. The entire application process is on the city's website. It's fairly extensive, requiring an explanation of the experiment and a description of the device, but can be filled out in a single sitting.

Unfortunately, processing these permits seems to take a long time. We applied for one in early May 2012, and it still hadn't arrived by the time this book went to press in November.

Get Your Community Involved

- If you're a student, get your teachers on your side when it comes to your projects. Arduino is so new that there's a good chance you'll be teaching the teachers for a change, and once your teachers understand what you're doing, they'll be among your staunchest supporters.

- If you're a member of a hackerspace, have an open house. Invite members of the community to meet you in person and learn about what you're doing. This may work wonders toward alleviating some of the public's fear of DIY electronics.
- If you're not a member of a hackerspace, consider joining one, or even starting one. Good online resources for finding and learning more about hackerspaces are the Hackerspace Wiki and the UK Hackspace Foundation.

3/A Brief Introduction to LEDs

Before we start working on the LED photometer, we're going to take a brief look at why and how the gadget's crucial component—the light emitting diode, or LED—can be used to monitor certain properties of the atmosphere.

Even if you've worked with LEDs before, chances are your projects used them simply as lights or indicators of some sort, with the color choices more about aesthetics than science. By comparison, in this book we use LEDs in ways that take advantage of the physics of light.

We think you'll get more out of building and using the LED photometer if you know more about that. But if you're feeling impatient, skip to the project in Chapter 4, the LED Sensitivity Tester, and proceed from there.

It is essential to test your LEDs before you build the LED photometer. Testing takes two forms: (1) testing to see if the LEDs light up and (2) testing the LEDs for wavelength sensitivity. The first test is easy: you can connect the short wire of an LED to Arduino GND and the long end to pin 13, and run the BLINK sketch that comes with the Arduino IDE software; if the LED blinks, it works. (An even easier way is to connect an LED to a 3 v coin cell—with the long wire on the positive side of the battery. If the LED lights, it works.)

Testing the LED for wavelength sensitivity is much more complicated; we deal with that in Chapter 4.

What Is a Diode?

A diode is a two-terminal electronic device that lets electricity flow easily in one direction, and prevents (or resists) it from flowing in the other direction.

Many electrical components can be run "backwards": for example, a loudspeaker can be used as a microphone, and vice versa. A motor, which turns electricity into movement, can often be used as a generator that turns movement into electricity.

This reversibility of electronic components is partly thanks to the fact that electricity usually has no problem flowing both ways through a circuit; it makes no difference to an electron if it flows from a battery to a motor or from a motor to a battery.

But the direction of electron flow matters if we're designing a generator circuit: we don't want that circuit to suddenly work in reverse and start spinning like a motor! This is where diodes are useful. They work like valves that prevent the "backflow" of electricity, ensuring that electrons will move only in the direction we want them to.

Discovery of the Diode

The 1870s were an exciting time to be working with electricity. In America, Thomas Edison was inventing some of the devices that would define the 20th century: the light bulb, the phonograph, and an improved telephone. Meanwhile in Europe, scientists were taking a more theoretical approach, seeking to understand how electricity flowed.

In 1874, a German scientist named Karl Ferdinand Braun published an account of what he called "the unilateral conduction of current by natural metal sulfides." Up to this point, scientists knew that electricity could travel along a wire in either direction. Braun discovered that a crystal of tetrahedrite (a compound of copper, antimony, and sulfur) let electricity flow through it in only one direction. This demonstration of one-way electrical flow was the first known semiconductor diode—in fact, it was the first semiconductor in history.

What Is a Light Emitting Diode?

A light emitting diode is a diode that emits light.

Okay, snark aside, in 1907 a scientist named Henry Joseph Round discovered that crystals of carborundum gave off a faint yellow light when they were energized by electrons. These crystals were also diodes, and didn't emit light when electricity tried to flow the other way. Scientists later discovered that in an LED, electrons combine with electron "holes" in the crystal, and release excess energy as photons in a very narrow range of wavelengths known as the emission band. The narrowness of this emission band is rivaled only by laser light, and explains the jewel-like purity of LED light.

Figure 3-1 shows some LEDs in sizes 8 mm, 5 mm, and 3 mm. Two important things to know for DIY purposes are that the longer lead is called the anode

(commonly thought of as the positive terminal), and the shorter lead is called the cathode (commonly thought of as the negative terminal). An easy way to remember this is that the negative terminal has something "subtracted" from it. Because an LED is a diode, electricity will only easily flow through it in one direction. So when you plug an LED into a circuit, be sure to connect the anode to the positive side of the circuit and the cathode to the negative side or ground, if you expect to make it glow.

Figure 3-1. *LEDs in 8 mm, 5 mm, and 3 mm sizes.*

How Are We Using LEDs in the LED Photometer?

LEDs not only emit light; they absorb it, too. An amateur scientist discovered this capability of LEDs in the 1970s. Forrest M. Mims III, a former US Air Force officer, had been doing experiments with photocells and light since he was a boy. Understanding that many electronic devices can be run "backwards" [see "What Is a Diode?" (page 33)], Mims reasoned that since LEDs take in electricity and emit light (along that narrow emission band, remember), they should also take in light and give off electricity. A series of experiments proved him right, and by 1973 he had formalized what is now known as the Mims Effect: LEDs will absorb light along a relatively narrow band of color, and emit a small amount of electricity. These light-absorbing qualities of LEDs are what we put to work in our LED photometer.

An LED absorbs light occurring in wavelengths near its own emission band, or slightly shorter. A green LED absorbs light in the greenish-blue part of the spectrum, a red LED absorbs light in the reddish-orange part of the spectrum, and so on. We can use this difference in the way LEDs absorb light to tell us a great deal about the atmosphere.

What Is Color?

Color is a property of matter that we perceive depending upon which light waves matter absorbs, and which it reflects or emits. Our eyes see the reflected or emitted light waves, and our brains process that information as color. The shorter wavelengths occur in what we percieve as the blue-indigo-violet range of the spectrum; the wavelengths we think of as green sit in the middle; and the longer wavelengths occur in what we see as the red-orange-yellow range. Ultraviolet (a very short wavelength of light) and infrared (a very long one) are ends of the spectrum not visible to human eyes—although we can see the re-emission of ultraviolet light, as with the psychedelic "black-light" posters of the 1960s and 1970s, which were printed with inks containing materials that excelled at absorbing those short light waves and re-emitting visible light. And we can feel the long heat waves created by infrared. When something appears white in sunlight, it is made of matter that reflects all wavelengths of light; if it looks black, it's made up of stuff that absorbs nearly all wavelengths.

4/Gadget: LED Sensitivity Tester

Before building and using the LED photometer, we first need to figure out exactly which wavelengths of light each of our LEDs is sensitive to. Otherwise, we won't be able to get good data with the photometer. Unfortunately, no LED datasheet that we've ever seen troubles itself with listing LED input wavelength, because LED manufacturers generally don't think of their products as input devices. Fortunately, we can determine input wavelengths with a gadget we can build ourselves: the LED sensitivity detector.

Mission: Inputtable

Your mission, should you choose to accept it, is to determine the peak input wavelength for a series of LEDs. You will do this by shining every wavelength of visible light, from 350–700 nm, into each candidate LED, and measuring that LED's responsive voltage. The LED that's most sensitive to the wavelength is the one that produces the highest response voltage. Good luck.

How are we going to generate every visible wavelength of light to test our LEDs, from ultraviolet to infrared? If your answer is "use a full-spectrum LED," you're correct. This special type of LED is called an RGB LED and it can output red light, green light, blue light, and any combination of those colors: the full ROY G. BIV. Since it's beyond the scope of this book to go too deeply into the science of combining light to make different colors, suffice it to say that by mixing varying intensities of red, green, and blue light, just about any color can be reproduced. (Look very carefully at the pixels of your computer screen to see this in action.)

Amount	Part Type	Properties/(Assembly Code)
1	Loudspeaker	(SPKR1)
1	TEST LED, 5 mm	5 mm [THT]; (LED2)
1	RGB LED (com. anode, rbg)	5 mm [THT]; pin order rgb; polarity common anode; rgb RGB (LED1)
2	47 Ω resistor	Package THT; tolerance ffl5%; bands 4; resistance 47 Ω; pin spacing 400 mil (R1 & R2)
1	62 Ω resistor	Package THT; tolerance ffl5%; bands 4; resistance 62 Ω; pin spacing 400 mil (R3)
1	Arduino UNO R3	(Arduino1)
1	LCD screen	Character type 16 pins (LCD1)
1	Permanent marker in a dark color	

Build the Gadget

1. **Connect the RGB LED to Arduino.** Most RGB LEDs are common anode, so follow the datasheet that came with your LED to connect the anode to the +5 pin on Arduino (Figure 4-1). The R, G, and B pins should be connected to Arduino pins 8, 6, and 4, respectively. For our RGB LED, the red cathode needed a 62 ohm resistor, while the green and blue cathodes needed a 47 ohm resistor. (Your values may vary.)

When we say "connect an LED to Arduino" in these instructions, we intend for you to do this via a breadboard and jumper wires, as illustrated in the figures for each step. Given the amount of gadget handling involved in testing a bunch of LEDs, using a breadboard will save a lot of wear and tear on Arduino.

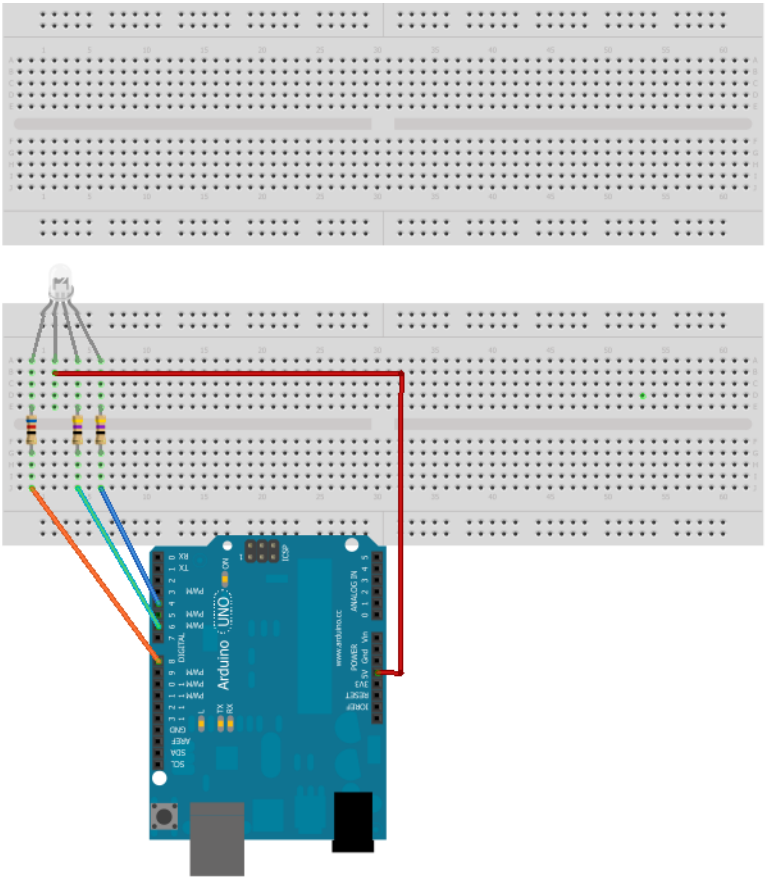

Figure 4-1. *Step one.*

2. **Connect the LCD.** Connect the common ground lead on the LCD (screen display) to the GND pin on Arduino. Connect the VCC lead on the LCD to the 3.3 power pin on Arduino. Connect the DATA lead on the LCD to Arduino pin 2 (Figure 4-2).

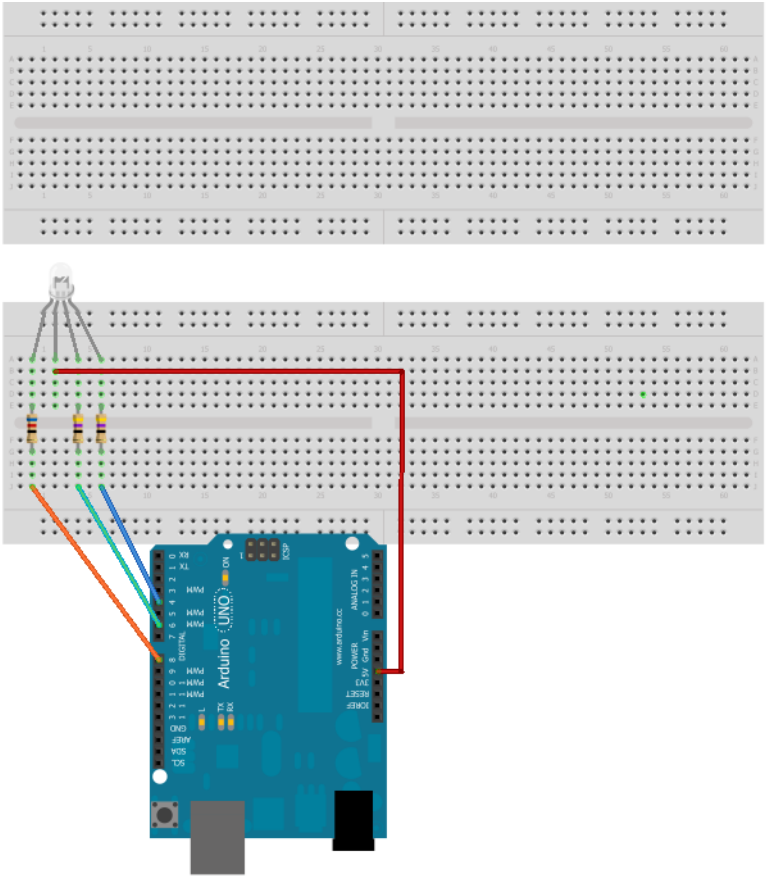

Figure 4-2. *Step two.*

3. **Add an audio speaker for additional fun.** Connect the speaker's black wire to GND on Arduino, and the speaker's red wire to Arduino pin 8. The schematic looks something like Figure 4-3.

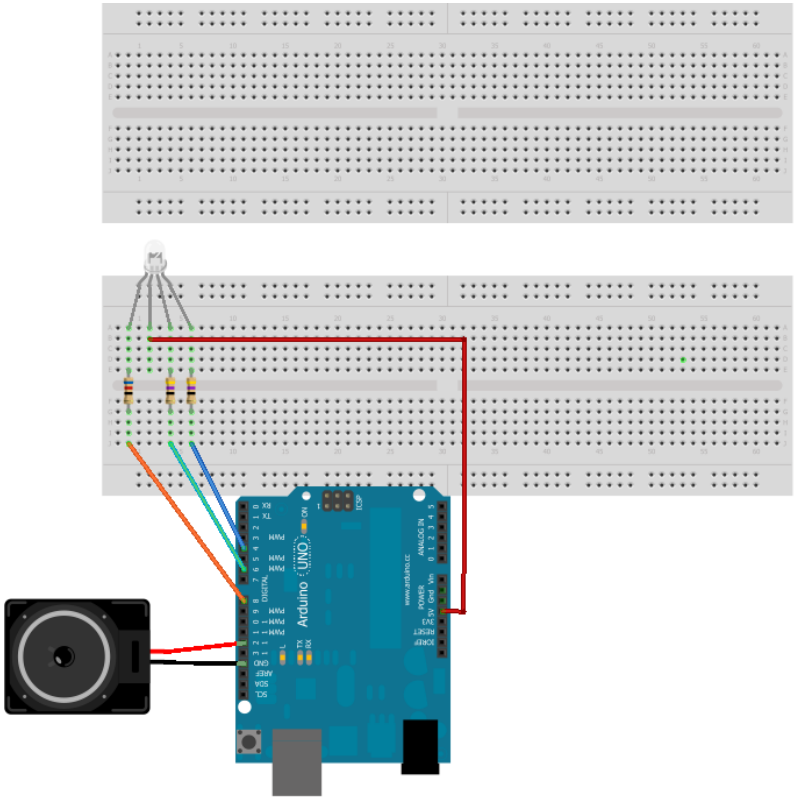

Figure 4-3. *Step three.*

4. **Load the sketch.** Now we need a way to translate a wavelength of light into its red, green, and blue values. Fortunately, a scientist named Dan Brunton has developed a formula that will do just that. We programmed Brunton's algorithm into the following Arduino sketch. After setting up all the components, running this code will cause the RGB LED to emit light from 350–700 nm, the range of visible light.

You can find this sketch in the AMWA GitHub repository.

```
/**
* Determining RGB color from a wavelength
* The code is based on an algorithm from Dan Bruton's Color Science Page.
* http://www.midnightkite.com/color.html
* by Patrick Di Justo
* 2012 08 28
*
**/
```

```
#include <EEPROM.h>
#include <SoftwareSerial.h>

//Set up the Liquid Crystal Display
#define LCDIn 3
#define LCDOut 2
SoftwareSerial mySerialPort(LCDIn, LCDOut);

//LCD Display buffers
char databuff[16];
char dispbuff[16];

// Variables needed for RGB calculations
float Gamma = 1.00;
int MaxIntensity = 255;
float fBlue;
float fGreen;
float fRed;
float Factor;

int iR;
int iG;
int iB;

//Our eyes can generally see light wavelengths between 350 and 700
nanometers.
//Here, we start the RGB Led with 350
int i = 350;

//RGB is plugged into these arduino digital pins
const int redOutPin = 8;
const int greenOutPin = 6;
const int blueOutPin = 4;

// LED to be tested is plugged into A0
int testPin = A0;

// variables to store the value coming from the sensor
int sensorValueTest =0;
int oldTest =0;
int peaknm =0;

//EEPROM start data
int addr=0;

//Music
int notelen = 90;
int dlx = 130;

void setup()
{
  pinMode(LCDOut, OUTPUT);
  pinMode(LCDIn, INPUT);
```

```
//Set the RGB LED pins to output

  pinMode(redOutPin, OUTPUT);
  pinMode(greenOutPin, OUTPUT);
  pinMode(blueOutPin, OUTPUT);


// Initialize the LCD display
  mySerialPort.begin(9600);
  mySerialPort.write(0xFE);
  mySerialPort.write(0x01);

  // test to see if the RGB LED works
  makeColor(i);
  analogWrite(redOutPin,255-iR);
  analogWrite(greenOutPin, 255-iG);
  analogWrite(blueOutPin, 255-iB);
  delay(5000);
}

void loop()
{

// set the RGB LED to a specific color
  makeColor(i);
  analogWrite(redOutPin, 255-iR);
  analogWrite(greenOutPin, 255-iG);
  analogWrite(blueOutPin, 255-iB);
  delay(500);

// read the sensitivity of the Test LED
  sensorValueTest= analogRead(testPin);

  if (sensorValueTest > oldTest)
  {
    oldTest = sensorValueTest;
    peaknm = i;
  }

// Display the values on the LCD
  sprintf(databuff,"CV:%3d Cnm:%3d",sensorValueTest,i);
  sprintf(dispbuff,"%-16s",databuff);
  mySerialPort.print(dispbuff);

  sprintf(databuff,"XV:%3d Xnm:%3d",oldTest, peaknm);
  sprintf(dispbuff,"%-16s",databuff);
  mySerialPort.print(dispbuff);



  writeData();
  i++;

  // If we've reached the upper limit of 700 nm, play a little melody
  if (i>700)
```

```
  {
    for (int f = 0; f<=100; f++)
    {
      tone(7,196,notelen);
      delay(dlx);

      tone(7,131,notelen);
      delay(dlx);

      tone(7,261,notelen);
      delay(dlx);

      tone(7,330,notelen);
      delay(dlx);

      tone(7,294,notelen);
    }
    delay(10000);
  }
}



void writeData()
{
  int quotient = i/256;
  int mod = i % 256;

  EEPROM.write(addr++,quotient);
  EEPROM.write(addr++,mod);

  quotient = sensorValueTest/256;
  mod = sensorValueTest % 256;
  EEPROM.write(addr++,quotient);
  EEPROM.write(addr++,mod);
}



void makeColor(int lambda)
{
  if (lambda >= 350 && lambda <= 439)
  {
    fRed    = -(lambda - (float)440.0) / ((float)440.0 - (float)350.0);
    fGreen = (float)0.0;
    fBlue       = (float)1.0;
  }
  else if (lambda >= (float)440.0 && lambda <= (float)489.0)
  {
    fRed        = 0.0;
    fGreen = (lambda - (float)440.0) / ((float)490.0 - (float)440.0);
    fBlue       = 1.0;
  }
  else if (lambda >= (float)490.0 && lambda <= (float)509.0)
  {
```

```
    fRed = 0.0;
    fGreen = 1.0;
    fBlue = -(lambda - (float)510.0) / ((float)510.0 - (float)490.0);

  }
  else if (lambda >= (float)510.0 && lambda <= (float)579.0)
  {
    fRed = (lambda - (float)510.0) / ((float)580.0 - (float)510.0);
    fGreen = 1.0;
    fBlue = 0.0;
  }
  else if (lambda >= (float)580.0 && lambda <= (float)644.0)
  {
    fRed = 1.0;
    fGreen = -(lambda - (float)645.0) / ((float)645.0 - (float)580.0);
    fBlue = 0.0;
  }
  else if (lambda >= 645.0 && lambda <= 780.0)
  {
    fRed = 1.0;
    fGreen = 0.0;
    fBlue = 0.0;
  }
  else
  {
    fRed = 0.0;
    fGreen = 0.0;
    fBlue = 0.0;
  }

  if (lambda >= 350 && lambda <= 419)
  {
      Factor = 0.3 + 0.7*(lambda - (float)350.0) / ((float)420.0 -
(float)350.0);
  }
  else if (lambda >= 420 && lambda <= 700)
  {
    Factor = 1.0;
  }
  else if (lambda >= 701 && lambda <= 780)
  {
      Factor = 0.3 + 0.7*((float)780.0 - lambda) / ((float)780.0 -
(float)700.0);
  }
  else
  {
    Factor = 0.0;
  }
  iR = factorAdjust(fRed, Factor, MaxIntensity, Gamma);
  iG = factorAdjust(fGreen, Factor, MaxIntensity, Gamma);
  iB = factorAdjust(fBlue, Factor, MaxIntensity, Gamma);
}

int factorAdjust(float C, float Factor, int MaxIntensity, float Gamma)
{
```

```
  if(C == 0.0)
  {
    return 0;
  }
  else
  {
    return (int) round(MaxIntensity * pow(C * Factor, Gamma));
  }
}
```

5. **Select an LED to test.** If you've purchased a large number of LEDs to test, they're likely to arrive in individual plastic pouches. Neatly pull or cut open the plastic pouch containing the LED, take the LED out, and set the pouch aside; you'll need it again soon (Figure 4-4).

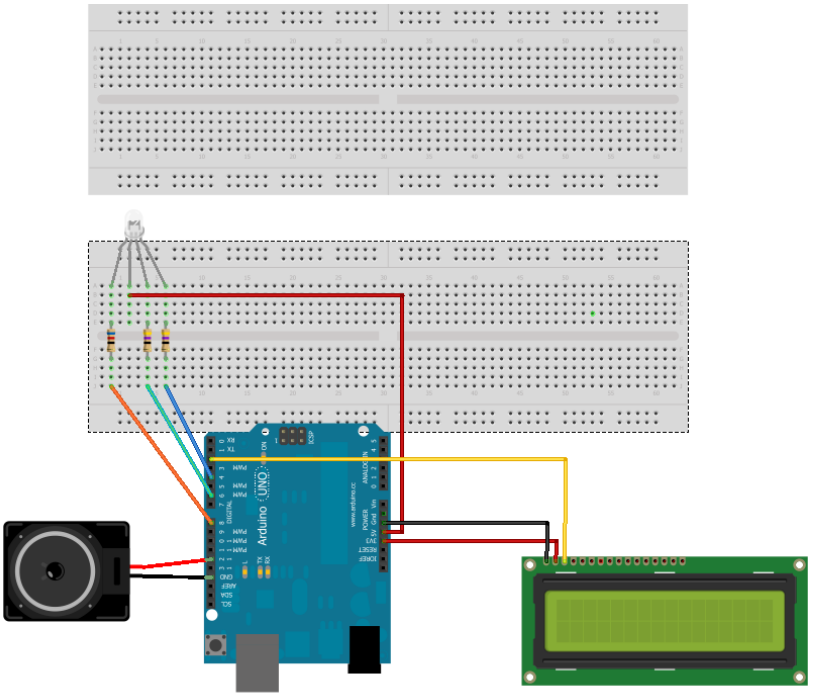

Figure 4-4. *Step four.*

6. **Connect the LED.** Hook up the anode to Arduino analog pin 0, and the cathode to GND (Figure 4-5).

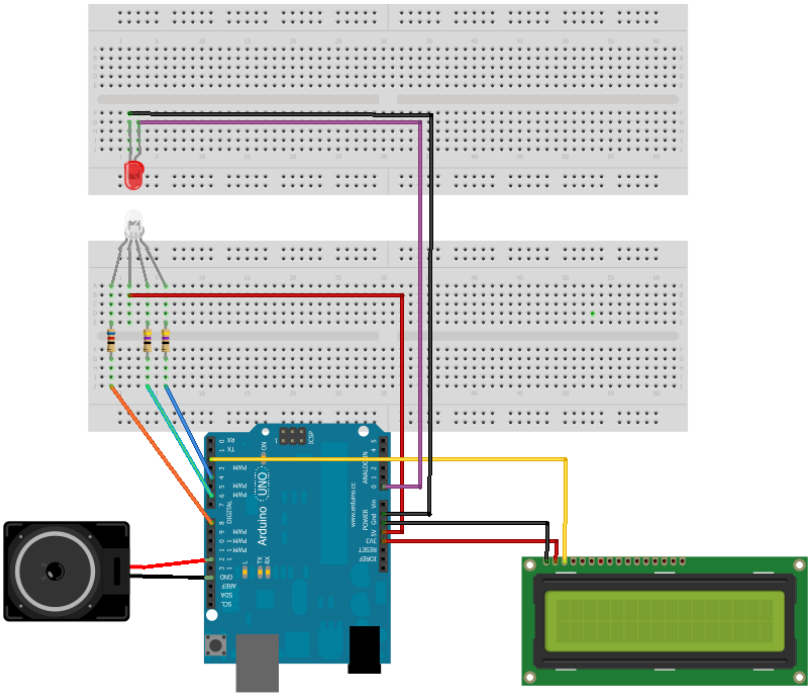

Figure 4-5. *Step five.*

As you test each LED, be careful to plug it into the breadboard with the proper polarity: anode to Arduino's analog pin 0 and cathode to Ardunio's GND pin.

7. **Test the LED.** For best results, enclose the test LED and the RGB LED in a single small, opaque tube, such as a piece of shrinkwrap tubing, a blackened soda straw, or even the opaque, sawed-off body of a ball point pen. Ideally, the two LEDs should be extremly close without actually touching. When everything is ready, turn on or reset Arduino.

   Several things will happen. The RGB LED will start to glow, and the top line of the LCD will inform you that the RGB LED is currently displaying a particular wavelength of light ranging from 350–700 nm. At each wavelength, the test LED absorbs the light being put out by the RGB LED, and converts that to a voltage.

- Low voltage means that the test LED is not absorbing much light at that wavelength; in other words, the test LED is not very sensitive to this color. A higher voltage indicates that the test LED is more sensitive to this particular wavelength and is therefore creating more electricity from the light it absorbs.
- As Arduino works its way up the spectrum, it keeps track of the peak voltage the test LED puts out, and the particular wavelength that caused the peak voltage.
- You'll know the test is complete when the counter reaches 700 nm, and Arduino plays a little melody.

8. **Note your results.** Using the permanent marker, write your data—the peak voltage and the light wavelength that caused it—on the plastic pouch that you set aside in step four. Carefully remove the test LED from the gadget and slide it back into the pouch.
9. Return to step five and hit the reset button on Arduino to repeat the sensitivity test with another LED.

In order to build an LED photometer that works properly, it's very important to test each LED before you use it in the photometer. You need to know the particular wavelengths of light that your LEDs are detecting, because this impacts the types of gases you'll be able to detect in the atmosphere, and how well you'll be able to detect them.

nm? What is this nm?

Wavelengths of light are so small they are measured not in millimeters (1/1,000ths of a meter) or micrometers (1/1,000,000ths of a meter) but in nanometers (1/1,000,000,000ths of a meter). Because wavelengths of light are so small, a great many things interfere with them. Imagine that you're inside a walk-in closet or windowless bathroom: by simply sealing the door, it would be easy to stop all light waves from entering the room. But if you had a portable radio with you (if you know what that is), or your cell phone (which is very much like a portable radio), you'd have no problem getting a radio signal even though all external light into your room was cut off—unless your closet was deep inside an underground, reinforced concrete security bunker!

Radio waves and light waves are the same physical phenomena: electromagnetic radiation, differing only in wavelength size. So we can say that clearly, light waves are more easily stopped by physical objects than other types of electromagnetic waves. In fact, that's the whole purpose of this LED photometer. Molecules of gas and particles of dust that by themselves are invisible to the naked eye all are large enough to interfere with certain wavelengths of

light. By measuring which light waves are affected more strongly and which light waves are affected less strongly by the given particles in the atmosphere, we can get a very good estimate of the concentration of those gases or particles in the atmosphere.

Table 4-1 shows some values we got with a sample of LEDs from different manufacturers. Notice that while the LED peak input value is usually lower than the output value, that's not always the case. LEDs at the blue end of the spectrum have input wavelengths almost exactly the same as their output wavelengths. As we move down the spectrum toward the red end, the distance between input and oputput wavelength grows, but not uniformly. For example, look at the two yellow LEDs giving off light with a wavelength of 592 nm. Their input values are exactly 20 nm different!

This explains why we said earlier that it is important to test your LEDs. You need to know exactly which peak wavelength your LEDs are absorbing.

Table 4-1. *Frequencies of light for various LEDs by part number.*

Manufacturer	Mfgr Part Number	Color	Output nm	Input nm
Jameco Valuepro	LVB3330	Blue	430	429
Avago Technologies	QLMP-LB98	Blue	470	471
Avago Technologies	QLMP-LM98	Green	525	498
Agilent	HLMP-CM39-UVCDD	Green	535	505
Siemans Corporation	LG5469FH	Green	565	507
Valuepro	LUY3833H	Yellow	590	537
Valuepro	BVT-5E1TT4E	Yellow	592	552
Jameco Valuepro	BVT-529TT8E	Yellow	592	532
Valuepro	LVY3333/A	Yellow	595	532
Valuepro	RL50-PY543	Yellow	595	517
Valuepro	LUE3333	Orange	620	625
Valuepro	UT9C13-86-UDC2-R	Red	630	580
Jameco Valuepro	BVT-5E1QT4ER	Red	634	555
Valuepro	RL50-PR543	Red	635	534

5/Gadget: LED Photometer

A photometer is a device that measures one or more qualities of light. Most photometers consist of an interference filter and a photo detector. The interference filter is a colored piece of plastic or glass that filters out nonessential colors, letting through only the precise wavelengths of light we're interested in studying. A photo detector behind that filter monitors the intensity of the light that makes it through the filter.

In this gadget, LEDs replace both the interference filter and the photo detector. As Forrest Mims showed, LEDs are photodiodes: diodes that generate electrical current proportional to the light that falls on them, and they act as their own color filter. As you learned from your own experiments with the LED sensitivity tester, a blue LED is most sensitive to blue light and much less sensitive to red. By recording the amount of current from differently colored LEDs, we can learn how much of each color of the sun's light (blue, green, and red) is getting through the atmosphere to the gadget.

The main advantage of an LED photometer is that it is relatively inexpensive. LEDs are cheap, rugged, and stable. They come in a variety of colors and a variety of spectral responses. The electronics that go with them can be purchased online or at any good electronics store.

A couple things to keep in mind:

- Although LEDs can work as detectors, they were not built for this purpose. Whereas an ideal detector for our needs would detect only a very narrow range of wavelengths, and be perfectly centered around the

wavelengths we wish to study, LEDs can detect light over a fairly wide range of wavelengths. That's why we've tested the LEDs: to know their detection peaks, so that we know which wavelengths of light we're measuring.

- Also, the current generated by an LED can change with the temperature of the LED; for this reason, we advise you to keep the gadget at room temperature, and make all outside measurements relatively quickly, before the gadget heats up (or cools down) too much.

Even allowing for these shortcomings, however, it is perfectly possible to get scientifically valid data by using the LED detectors to measure the upper atmosphere.

Build the Gadget

Amount	Part Type	Properties
1	Piezo Speaker	
1	Blue LED, 5 mm	Package 5 mm [THT]; leg yes; color Blue
1	Green LED, 5 mm	Package 5 mm [THT]; leg yes; color Green
1	Red LED, 5 mm	Package 5 mm [THT]; leg yes; color Red
1	Arduino	Processor ATmega; variant Arduino
1	LCD screen	Type character; pins 16

There are two sequences of steps that are equally essential to creating the LED photometer: building it, and calibrating it.

1. Connect the LEDs to Arduino (Figure 5-1). We've found it's easiest to do this as follows:
   a. Connect the negative (i.e., shorter wire) of each LED to the GND rail of the breadboard.
   b. Connect the positive (i.e., longer wire) of each LED to the breadboard.
   c. Using jumpers, connect each LED to the appropriate Analog pin on Arduino. Our Arduino sketch expects the red LED to be plugged into A1, the green LED into A2, and the blue LED into A3.
2. Connect the speaker (Figure 5-2). Plug the positive lead into Arduino digital pin 7 and the negative pin into Arduino GND. Many speakers don't list the polarity of their connection, so don't worry too much about this; just plug it in!

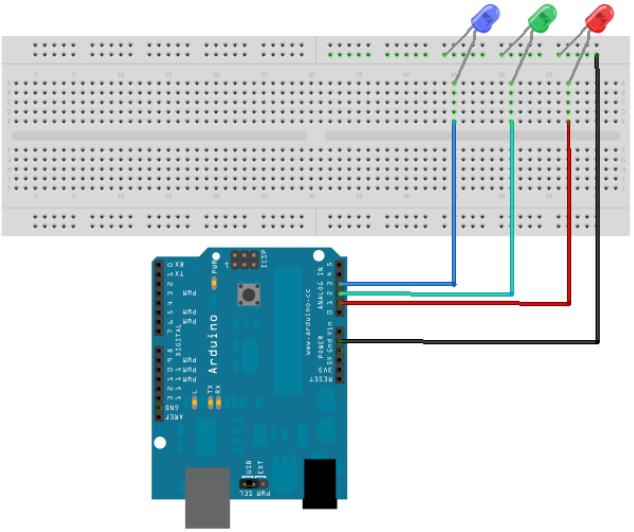

Figure 5-1. *Step one.*

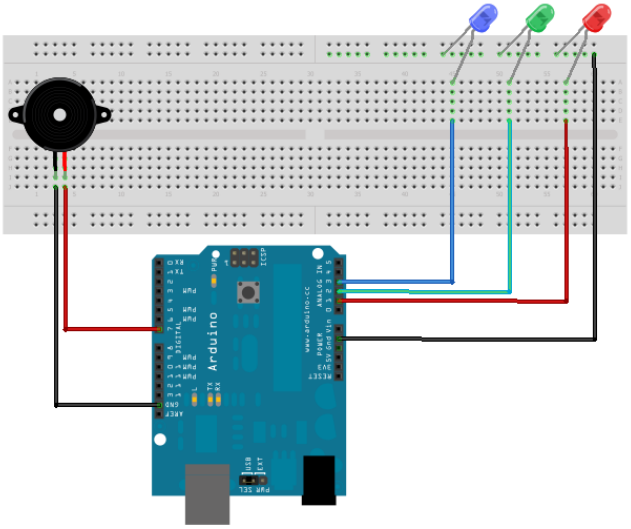

Figure 5-2. *Step two.*

3. Connect the LCD unit to Arduino (Figure 5-3). The data line connects to Arduino digital pin 5, the power line connects to Arduino 3.3 v pin, and GND line connects to GND.

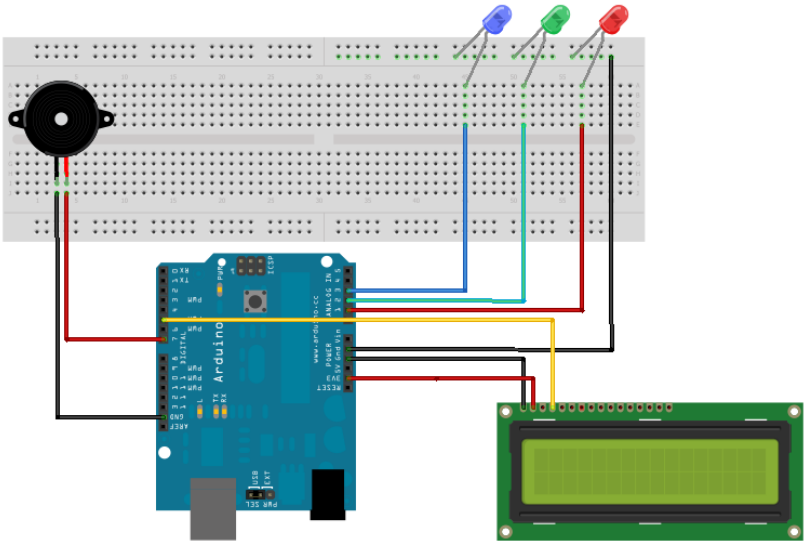

Figure 5-3. *Step three.*

Load the Sketch

You can find this sketch in the AMWA GitHub repository.

```
/*
 LED Photometer, based on Analog Input
 by Patrick Di Justo
 2012 08 30
*/

#include <EEPROM.h>
#include <SD.h>
#include <SoftwareSerial.h>

// Liquid Crystal Display
// Define the LCD pins: We'll be using a serial-based LCD display
// which only required +3.3Volts, GND, and a single data line.
// databuff and displaybuff hold the data to be displayed

#define LCDIn  3
#define LCDOut 5
```

```
SoftwareSerial mySerialPort(LCDIn, LCDOut);

// Data Buffers for the LCD
char databuff1[16];
char databuff2[16];
char dispbuff[16];

// LED variables section
// Which Arduino pin goes to which LED
int redPin = A1;
int greenPin =A2;
int bluePin=A3;


// A place to store the values coming from the analog port
int sensorValueRed =0;
int sensorValueGreen =0;
int sensorValueBlue = 0;

//A place to store the maximum value for each LED
int maxRed = 0;
int maxGreen =0;
int maxBlue = 0;


//EEPROM variables
// The record length is 7: 1 byte for the record number, 2 bytes
// each for the 3 LEDs.  For each additional LED you add, increase
// the record length by 2.
int record=0;
int reclen = 7;
int addr =0;

//  the  following  variable  is  long  because  the  time,  measured  in
miliseconds,
// will quickly become a bigger number than can be stored in an int.
long timeSinceLastSensorHigh = 0;
int dataWritten = 0;

// music section
int notelen =40;
int dlx = notelen *1.33;

//switch to tell if an SD card is present
int SDPresent = 1;

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);

  // Set the Analog Pins
  // Why are we setting input pins to output?
  // We're doing this to prevent "leakage" from the pins.
  // Setting the pins to output activates a pullup resistor,
```

```
  // which makes it difficult for voltage to come into the Arduino,
  // until we're ready for it.

  Serial.println("Setting up the Analog Pins");

  pinMode(redPin, OUTPUT);
  digitalWrite(redPin, LOW);
  pinMode(greenPin, OUTPUT);
  digitalWrite(greenPin, LOW);
  pinMode(bluePin, OUTPUT);
  digitalWrite(bluePin, LOW);

  // Set up SD card, let us know if SD card is absent
  pinMode(10, OUTPUT);
  if (!SD.begin(4)) delay(10);
    SDPresent =0;

  //Set up LCD
  pinMode(LCDOut, OUTPUT);

  mySerialPort.begin(9600);
  mySerialPort.write(0xFE);
  mySerialPort.write(0x01);
  sprintf(databuff1,"Wakeup Test");
  sprintf(dispbuff,"%-16s",databuff1);
  mySerialPort.print(dispbuff);


//set up EEPROM
  record = EEPROM.read(0);
  addr = (record * reclen) +1;
  Serial.println("BEEP");

// Play music to let user know the gadget is ready
  tone(7,294,notelen);
  delay(dlx);
  tone(7,330,notelen);
  delay(dlx);
  tone(7,261,notelen);
  delay(dlx);
  tone(7,131,notelen);
  delay(dlx);
  tone(7,196,notelen);
  delay(dlx);
  delay(3000);
}


void loop()
{
  // read the value from the sensor:
  /*
    Back in setup(), we enabled a pullup resistor on the analog pins, which
    made it difficult for electricity to come into the analog pins.
      Here, we disable the pullup resistor, wait 10ms for the pin to
```

```
stabilize,
      read the voltage coming into the pin, then reenable the pullup
resistor.
  */

  pinMode(redPin, INPUT);
  delay(10);
  Serial.print("Reading red:  ");
  sensorValueRed= analogRead(redPin);
  pinMode(redPin, OUTPUT);
  Serial.println(sensorValueRed);
  delay(10);

  pinMode(greenPin, INPUT);
  delay(10);
  sensorValueGreen = analogRead(greenPin);
  pinMode(greenPin, OUTPUT);
  delay(10);

  pinMode(bluePin, INPUT);
  delay(10);
  sensorValueBlue = analogRead(bluePin);
  pinMode(bluePin, OUTPUT);
  delay(10);

  Serial.println("Comparing sensor values...");

  // Here we compare each sensor to its maximum value.
  // If any of the sensors has reached a new peak, sound a tone
  if(    (sensorValueRed>maxRed)
      || (sensorValueGreen>maxGreen)
      || (sensorValueBlue>maxBlue))
    {
    tone(7,maxRed+maxGreen+maxBlue,500);
    timeSinceLastSensorHigh = millis();
    }

  // Here we reset the old maximum value with a new one, if necessary
  if(sensorValueRed>maxRed) maxRed = sensorValueRed;
  if(sensorValueGreen>maxGreen) maxGreen = sensorValueGreen;
  if(sensorValueBlue>maxBlue) maxBlue = sensorValueBlue;

  // Display the sensor values on the LCD screen
  sprintf(databuff1,"R%3d G%3d B%3d",maxRed,maxGreen,maxBlue);
  sprintf(dispbuff,"%-16s",databuff1);
  mySerialPort.print(dispbuff);
  Serial.print(dispbuff);

// If 10 seconds has gone by without any new measurements, write
// data to storage.

  if(millis() > (timeSinceLastSensorHigh + 10000))
    {
    if(dataWritten ==0)
      {
```

```
        writeData();
        if(SDPresent = 1)
          {
          writeDataToSD(databuff1, databuff2);
          }
       }
     }
}



void writeData()
{
  Serial.print("I'm writing data!!!!!!!!!!!!!!!");
  record++;
  EEPROM.write(0, record);
  EEPROM.write(addr++, record);

  /*
  The problems of data storage:
   The analog pins read a value from 0 to 1023. This is 1024 different
values,
  and 1024 = 2 ^ 10. It would take 10 bits of data space to safely
  store the value from the analog pins.  Unfortunately, a standard byte
  of data is only 8 bits.  How can you fit 10 bits into 8 bits?
  You can't.

  What we're doing here is splitting the 10 bits of data into two sections,
  which can be stored in two bytes of memory space.
  */


  int quotient = sensorValueRed/256;
  int mod = sensorValueRed % 256;
  EEPROM.write(addr++,quotient);
  EEPROM.write(addr++,mod);

  quotient = sensorValueGreen/256;
  mod = sensorValueGreen % 256;
  EEPROM.write(addr++,quotient);
  EEPROM.write(addr++,mod);

  quotient = sensorValueBlue/256;
  mod = sensorValueBlue % 256;
  EEPROM.write(addr++,quotient);
  EEPROM.write(addr++,mod);
  dataWritten = 1;

  sprintf(databuff1,"EEPROM written");
  sprintf(dispbuff,"%-16s",databuff1);
  mySerialPort.print(dispbuff);
  Serial.println("FINAL BEEP");

  tone(7,196,notelen);
  delay(dlx);
```

```
  tone(7,131,notelen);
  delay(dlx);

  tone(7,261,notelen);
  delay(dlx);

  tone(7,330,notelen);
  delay(dlx);

  tone(7,294,notelen);
}


void writeDataToSD(String dataString1, String dataString2)
{
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File dataFile = SD.open("LEDdata.txt", FILE_WRITE);

  // if the file is available, write to it:
  if (dataFile)
  {
    dataFile.print(millis());
    dataFile.print(",");
    dataFile.println(dataString1);
    dataFile.close();

    sprintf(databuff1,"SDCard written");
    sprintf(dispbuff,"%-16s",databuff1);
    mySerialPort.print(dispbuff);
  }
}
```

Calibrate the Gadget: Air Mass, Atmospheric Optical Thickness, and Extraterrestrial Constant

Remember when we said that the sunlight travels to the ground through different thickness of atmosphere, depending on how high the sun is above the horizon? This can have an effect on the readings you get with your LED photometer, as well (see Figure 5-4).

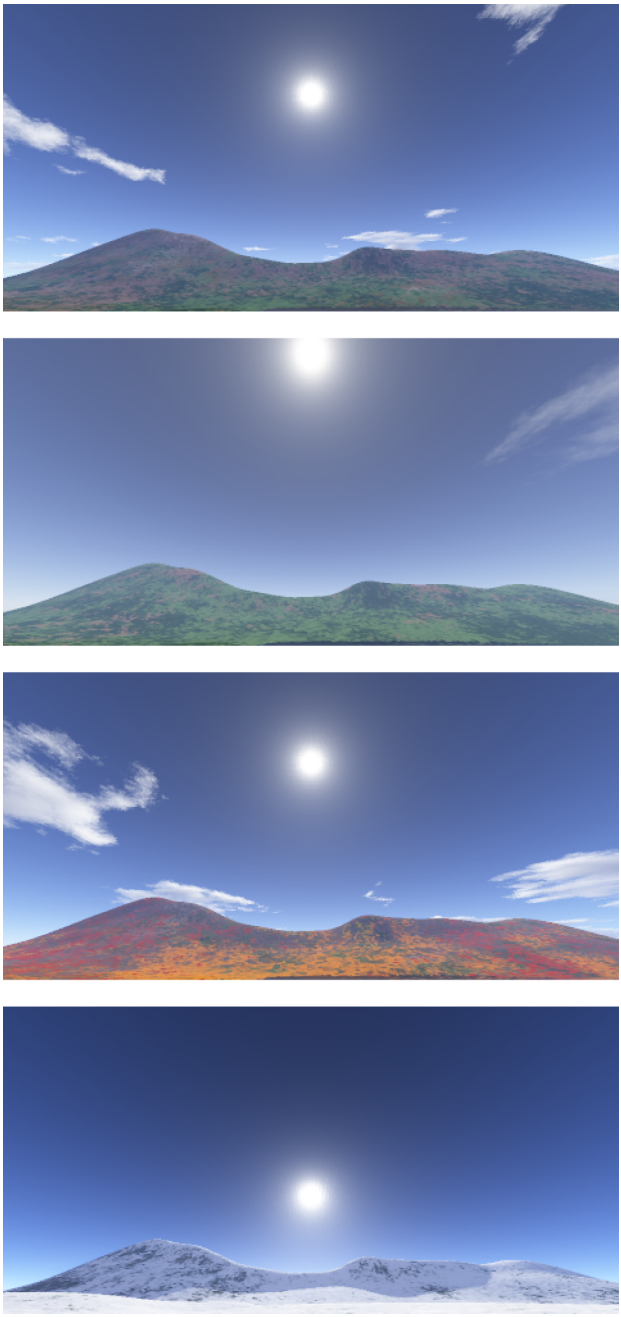

Figure 5-4. *Sun angles in spring, summer, fall, and winter.*

Generally, this device is meant to measure the atmosphere at around noon each day, when the sun is at its highest in the sky and its path through the atmosphere is shortest. At the very least, if you can't make regular noon measurements, you should try to take measurements around the same time each day.

As long as you keep the components as they are (e.g., don't swap out various LEDs, or use a different Arduino), your photometer doesn't need to be calibrated ever again. There are examples of LED photometers that have been working for 20 years without further calibration.

But that raises an interesting problem: as the Earth revolves around the sun, the position of the sun at noon changes enormously over the course of the year. On the first day of summer, the noonday sun is as close to being directly overhead as it will ever get, while on the first day of winter, the noonday sun hugs the horizon 47° lower than the summer sun. This can have a profound effect on the measurements you get with your LED photometer, since the sunlight is moving through significantly differing distances of air through the course of a year.

This is why we have to take into account a value called *air mass*. The formula for air mass is very simple:

m = 1 / sin(theta)

where theta is the angle of the sun above the horizon.

Let's take a look at how the air mass at noon will differ over the course of a year from A.M.W.A. World Headquarters (see Table 5-1).

Table 5-1. *Sun angles and air mass from New York City.*

Date	Noon Sun Angle	Air Mass
March 20, 2013	49.4	1.3170
June 21, 2013	72.7	1.0474
September 22, 2013	49.3	1.3190
December 21, 2013	25.9	2.2894

At noon on the first day of spring, March 20, the noon sun is 49.4° above the horizon. Plugging that into the preceding formula results in an air mass of about 1.3170. This means that the sun's light had to travel through 1.3170 times as much air as it would if the sun were directly overhead.

At midday on the first day of summer, June 21, the sun was at 72.7° above the horizon. This gives us an air mass of 1.0474, meaning that the sun's light had to travel through 1.0474 times as much air as it would if the sun were directly overhead; that's a pretty negligible difference on its own terms, but about 25% lower than the air mass in March.

On September 22, the first day of autumn, the sun returned to 49.3° above the horizon, almost exactly where it was in March.

On December 21, the first day of winter, the sun at noon was 25.9° above the horizon. Plugging that into the preceding formula, we see that the air mass is a whopping 2.2895! The sun's light is traveling through more than twice as much air as it did on the first day of summer. This is an enormous difference, and, if uncorrected, would have an effect on the data we collect.

To fix this, we can use the data we collect to calculate a value called the *atmospheric optical thickness* (AOT). AOT measures the clarity of the air for any given air mass by comparing the data that you measure on the surface of the earth with the data your LED photometer would get if it were above the atmosphere, out in space.

Are we going to send our LED photometer into space? If only! Although that sounds like a great project, it's a little beyond the scope of this book—and unnecessary. What we're going to do is to take several measurements that will tell us what our LED photometer would measure *if it were out in space*.

Calculating Atmospheric Optical Thickness

Once you've built your LED photometer, wait for an exceptionally clear day. You'll want a day where the humidity remains constant, the cloud cover is almost nonexistent, and a place where there are no obvious nearby forest fires, or anything else that would contribute haze to the atmosphere. (Conversely, you could also wait for an overcast day, as long as the cloud cover is uniform. However, the data you get might not be as good as one from an exceptionally clear day.)

On this clear day, you're going to take a series of measurements with your LED photometer every half hour for half of the daylight hours (Table 5-2). You can take these measurements from dawn until noon, or from noon until sunset. Carefully keep track of the sun's angle above the horizon at each measurement point.

Table 5-2. *Solar data, noon to near sunset.*

Time	Sun Angle	Air Mass	Averaged LED Value
12:00	58	1.179	578.33
12:30	57.2	1.189	539
13:00	55	1.220	553

13:30	51.8	1.272	552
14:00	47.9	1.347	563.33
14:30	43.3	1.458	552.66
15:00	38.3	1.613	575
15:30	33	1.836	520.33
16:00	27.5	2.165	539
16:30	21.9	2.681	502
17:00	16.3	3.562	490
17:10	14.4	4.021	474.66
17:20	12.5	4.620	472.66
17:30	10.6	5.436	463.33
17:40	8.7	6.611	443.33

To find the angle of the sun at any given time of day, visit the US Naval Observatory's Sun Calculator.

Once you've collected this data, take it back to your workstation and, using graph paper or a spreadsheet program, create a scatter chart in which the Y axis is the logarithm of the data from any one of the LEDs (or all of the LEDs) and the X axis is the air mass at which these readings were taken.

Draw a best-fit line through the data points you get, and extend that line back to the Y intercept where the air mass is zero. The Y value at that point is known as the *extraterrestrial constant*, or EC. (If you're using a spreadsheet, look for a function called linear regression to accomplish the same thing.) Once you've calculated the EC, your LED photometer is calibrated.

Now that we've derived EC, we can finally use that to calculate atmospheric optical thickness, using the following formula:

AOT = (log(EC) / log(LED photometer reading)) / m

When your data is run through this formula, it will be scientifically "fit." The natural variance in the height of the sun above the horizon at various times of year is now accounted for. The AOT is the true measure of atmospheric optical thickness and should apply to all data you collect with your LED photometer.

Every LED Photometer Has a Unique Extraterrestrial Constant

The extraterrestrial constant varies for each LED photometer. Our LED photometer will have an EC that's very slightly different then the LED photometer you build. The variability may be as simple as different thickness of wire used to connect the LEDs to Arduino, or different models of Arduino, or different models of LEDs. The atmosphere's opacity may change from day to day as weather and environmental conditions shift, but the extraterrestrial constant is a property of the instrument. Once you determine the extraterrestrial constant for your own LED photometer, it shouldn't need to be recalibrated ever again.

Things to Try

Now that you've built the apparatus, let's look at some things that you can do with it.

Detecting "Ozone Holes": Measuring the Ozone Layer

The ozone layer is a narrow band in the Earth's atmosphere, approximately 25 kilometers above the surface, where the concentration of the O_3 ozone molecule is relatively high, reaching about 10 parts per million (compared to the usual 0.6 ppm in other parts of the atmosphere).

The ozone layer is essential to life on Earth, because it absorbs around 97% of the ultraviolet radiation from the sun, in the range of about 200–315 nm (what scientists and sunscreen manufacturers call the UV-B band). UV-B rays have enough energy to damage many biological molecules, including DNA. Without an ozone layer, the Earth might never have developed life.

Save the Ozone Layer!

Widespread use of molecules called chlorofluorocarbons (CFCs) as propellants in aerosol cans did a significant amount of damage to the ozone layer in the 20th century; CFCs can break apart ozone molecules, leaving them unable to absorb UV light. A 1987 international agreement, called the Montreal Protocol on Substances that Deplete the Ozone Layer (or just "Montreal Protocol" for short), banned most CFCs from use; nearly 200 nations have

ratified it. Still, CFC residues remain in the stratosphere; some studies report a 4% decrease in ozone concentrations per decade since the late 20th century. However, since the turn of the century, scientists have begun to detect evidence of ozone layer recovery, as well.

Measuring the ozone layer with LEDs is conceptually not very difficult: all you'd need to do is to find an LED that aborbs ultraviolet light in the UV-A range (about 200–315 nm), and compare its output with an LED that absorbs light in the UV-B range. If the ozone layer didn't exist, the surface measurement of UV-A and UV-B would be equal, so any difference in the reading of the two LEDs would most likely be related to the presence of the ozone layer.

All this seems easy enough to allow you to build an upper atmosphere ozone monitor; but as a practical matter, UV LEDS become more and more expensive as you go down the wavelengths. It's also considerably more difficult (and requires more and different equiment) to test a UV LED to see its absorption range.

However, there might be another way to make this work. Everyone knows that ozone absorbs ultraviolet light, but not many people know that ozone also has an absorption band centered around 602 nm, in the orange part of the spectrum. This region, called the Chappuis band, is nowhere near as deep as the UV-B absorption bands; still, scientists have gotten good data from measuring how ozone absorbs orange light. Perhaps you can build an LED photometer that measures the ozone absorption in the orange part of the spectrum, if you can find an LED sensitive to 602 nm. Good luck!

Add an Accelerometer

Adding an accelerometer to the LED photometer will automatically tell you the angle at which you're pointing the device at the sun.

As we noted earlier, the sun's angle at noon can vary by 47° over the course of a year. While it's not going to kill you to expend the effort to look up or calculate the sun's declination when you take a measurement, it would be infinitely easier if Arduino itself told you the angle. You can make this happen with a multiple axis accelerometer attached to your Arduino. This can make the calculation of AOT so simple you might be able to include it in the Arduino code, and store the resulting value with the rest of the data.

But there's a drawback: most accelerometers use up to three analog inut pins to get data to Arduino. You might have to cut back on the number of LEDs you're using, if you use an accelerometer. If you can think of another way to make this work, we'd love to hear from you.

6/Using the LED Photometer

One of the oldest bits of weather lore is "Red sky at night, sailor's delight. Red sky in morning, sailors take warning." The sky is often red, or reddish orange, at sunrise and sunset. Reds, pinks, and oranges dominate at these times of day because the distance from your eye to the sun is greater at these times than when the sun is directly overhead. When sunlight travels through the atmosphere—and especially when it travels that extra distance from the horizon—an effect known as *Rayleigh scattering* takes place.

The British physicist John William Strutt (the third Baron Rayleigh) explained in 1871 that air molecules are so small they interfere with and scatter photons of light. The various molecules that make up the Earth's natural atmosphere—mostly nitrogen and oxygen, with trace amounts of other gases—scatter blue light (shorter wavelength light) more efficiently than red, orange, or yellow light (longer wavelengths). So when you look up, you see more scattered blue light than red or orange light. This is why our sky appears blue during most of the daytime (Figure 6-1).

When the sun is on the horizon at dawn or dusk, however, its light must travel through more of the atmosphere before we see it than when it is overhead (Figure 6-2). This leads to more and more of the blue light being scattered away, leaving behind the red-orange-yellow light. As well, dust and aerosols in the atmosphere more ably scatter long wavelengths of light—leading to lurid red sunsets.

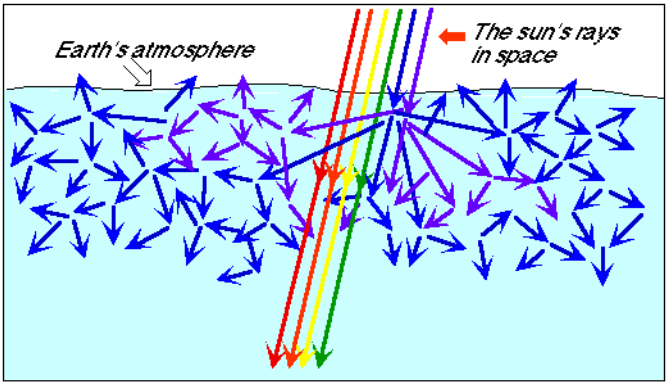

Figure 6-1. *As sunlight hits the Earth's atmosphere, air molecules scatter the shorter, blue wavelengths of light, but let the longer wavelengths through. Credit: NOAA ERSL.*

As atmospheric particles get larger, they tend to scatter all wavelengths of light equally and indiscriminately. This explains why clouds, which are made of relatively large water droplets, are white.

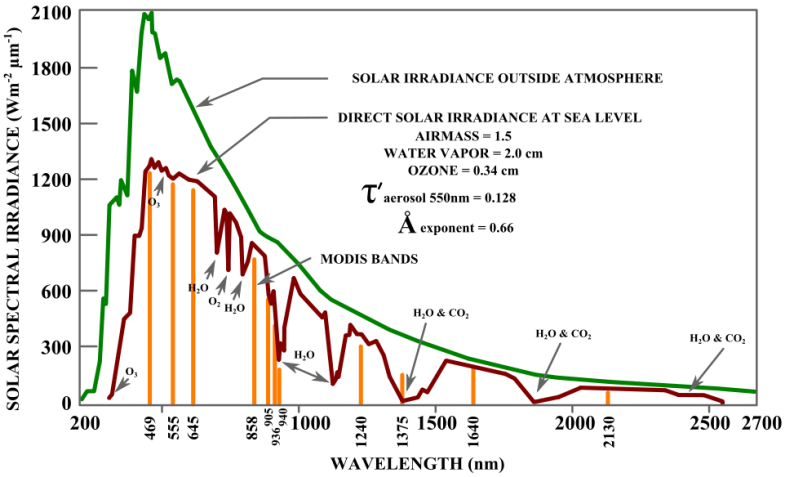

Figure 6-2. *This chart describes the absorption of sunlight by various atmospheric molecules. Look carefully in the 700 nm range of wavelengths: the dips in the "Direct Solar Irradiance at Sea Level" line indicate where molecules like water and oxygen are blocking out certain wavelengths of sunlight.*

Of course, molecules don't just scatter wavelengths of light. Certain molecules, such as water vapor, carbon dioxide, and ozone, also absorb specific wavelengths of light. To a measuring instrument on the ground, there is less light at those specific wavelengths than there is at other nearby wavelengths, because some of that light has been absorbed. By measuring these "holes" where wavelengths of light are absent, you can get a very good estimate of how much water vapor, ozone, and some other substances are in the atmosphere above your head.

Atmospheric Aerosols

Atmospheric aerosols are tiny particles of matter suspended in the atmosphere. These minute particles scatter and absorb sunlight, reducing the clarity and color of what we see. This is the effect we're referring to when we talk about "haze" in the sky.

Sources of Haze

Atmospheric haze comes from diverse sources. Most inhabited places on Earth always have some water vapor in the atmosphere, and enough water vapor can cause haze. Clouds and fog add more particles to the atmosphere, making visibility even hazier. Forest fires, volcanic eruptions, and dust storms can all increase haze in the atmosphere, sometimes hundreds of miles away from where they originate.

Human activites contribute to haze as well: smoke given off by cooking or heating fires, for example, or by industrial facilities such as coal-fired power plants. When sunlight hits the exhaust from automotive vehicles, the nitrogen oxides and hydrocarbons in the exhaust react to create photochemical smog and ozone; both create haze. Even contrails from high flying airplanes can make the sky hazy.

According to the US Environmental Protection Agency, since 1988 the atmosphere in national parks and wilderness areas in the United States has gotten so hazy that average visibility has dropped from 90 miles to 15–25 miles in the east, and from 140 miles to 35–90 miles in the west.

The LED photometer measures the difference between the voltage produced by a red LED, which absorbs light around the 580 nm range, and the voltage produced by a green LED, which absorbs light around the 500 nm range. The readings appear on the LCD, and are then stored on the Arduino's EEPROM (and/or SD card, if you choose to use one).

You can chart this data by hand, extract the data, and graph it in a spreadsheet program, or upload it to Cosm (formerly Pachube) to share it with the world. More on this shortly.

On a clear day with very little haze, the voltages produced by the green LED and red LED will be somewhat similar. As atmospheric aerosols increase, however—whether over the course of a day, several days, or from season to season—the difference between the red LED's voltage and the green LED's voltage will increase. The green LED will produce less current when there is more atmospheric haze, because less green light than usual is reaching it.

The voltages will almost never be exactly the same; Rayleigh scattering guarantees that more green light will be scattered by our atmosphere than red light. Also, white light reflected by clouds can sometimes cause the green LED to spike. The best results come from days without clouds (or with a uniform cloud cover).

Photosynthetically Active Radiation (PAR)

Photosynthesis, the process by which green plants turn sunlight into carbohydrates, does not use green light.

Plants appear green precisely because they *reflect* green light; they have no use for those wavelengths, and therefore can afford to throw them away. Green leaves absorb a great deal of the light at red and blue wavelengths. You can use LEDs to measure how much of those wavelengths—how much photosynthetically active radiation—is reaching your detector.

You do this by summing the outputs from two LEDs: the same red LED as in the aerosol detector, and a blue LED. As with the aerosol detector, Arduino reads the voltages produced by the red and blue LEDs, displays them, and stores them where they can be graphed, extracted into a spreadsheet, or uploaded to Cosm.

Forrest M. Mims, the man who discovered the Mims effect, has shown that using red and blue LEDs to measure photosynthetically active radiation agrees very well with more "professional" PAR sensors.

Water Vapor (WV)

Atmospheric water vapor absorbs EM wavelengths in the range of infrared light, at around 940 nm. By using an infrared LED specifically designed to absorb light at 940 nm, you can get a very accurate representation of the amount of water vapor in the atmosphere. Simply compare the outputs of the 940 nm LED with a similar infrared LED sensitive to around 880 nm.

One thing to keep in mind, however, is that this type of measurement is incredibly dependent upon local weather conditions. The rapid passage of a storm front over your observational area can play havoc with your readings: obviously there's going to be more water vapor in the air when it is raining, snowing, or about to do either. Thus, this gadget is better suited to measuring upper atmosphere water vapor—water vapor high above what we might think of as "the weather." Because of this, the best water vapor measurements are those made on clear days.

Extracting Data from the LED Photometer

Okay, so you've collected a bunch of data with your LED photometer. What do you do now?

Graphing Data in a Spreadsheet

Once you've processed the data in a spreadsheet, you can graph it. Your results might end up looking like these charts by Forrest M. Mims, who has been collecting atmospheric data with an LED photometer in Texas since the early 1990s (Figure 6-3).

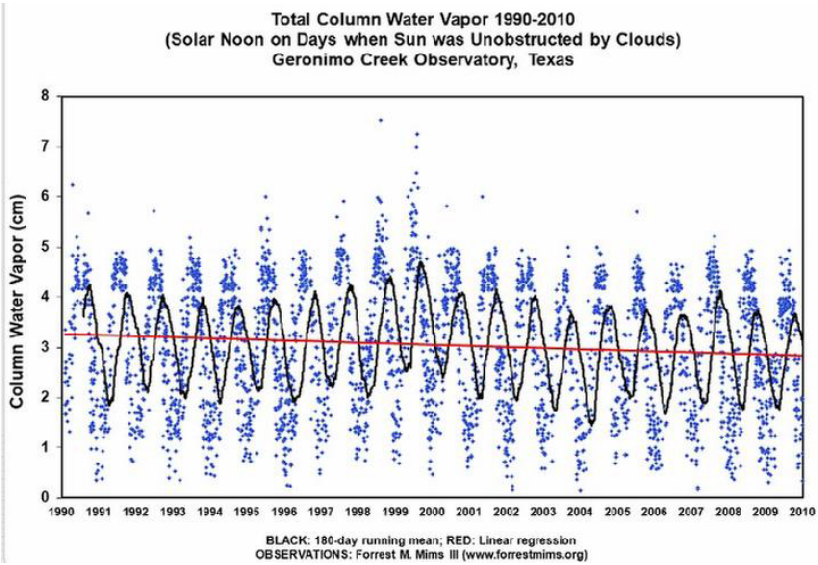

Figure 6-3. *Forrest Mims III has gathered this data on total water vapor in the atmosphere using an LED-based photometer. Image used with permission.*

Notice the patterns in Mims's decades of data: nearly every reading shows some kind of seasonal variation. It might take a year or more before you can see a full cycle in your data.

Sending Data to COSM

You can upload your data for all the Internet to see thanks to a service called Cosm. Our previous book, *Environmental Monitoring With Arduino* (O'Reilly), contains a whole tutorial on getting started with Cosm; for now, let's just say that Cosm makes it very easy to upload a CSV data file. Just cut and paste the serial output of the EEPROM reader program, or use the file *DATALOG.TXT* from the SD card, and place it on your website. Then go to your Cosm account and tell it where to find that file. Before you know it, your LED photometer data will be on the internet, neatly graphed, for all the world to see.

7/Doing Science: How to Learn More from Your Atmospheric Data

Science can be defined as the practice of observing the natural world, and trying to make objective sense of it by uncovering facts or cause-and-effect relationships.

The gadgets in this book detect substances and conditions in the atmosphere that otherwise would be invisible to your senses. (Essentially, the gadgets are technological extensions of your senses.) Building them will help hone your skills with DIY electronics and Arduino programming. These are fun, interesting, and practical things to do—but doing them by themselves is not doing science.

Suppose you'd like to learn more from what you uncover. Maybe you'd like to measure atmospheric conditions over days or weeks, and then interpret those readings; or monitor the atmosphere in different parts of your neighborhood, county, or state, and compare that data usefully; or perhaps even organize people around the world to build gas sensors or photometers, and compare findings from these different places in meaningful ways. To do these things, you'll need to apply some intellectual elbow grease to how you use your gadget. You'll have to do some science.

The Scientific Method

The scientific method is the foundation of how most of the serious science in the world gets done. It's a systematic process of investigation that tests ideas about how cause and effect operate in the natural world, helps to reduce or eliminate bias, and allows the meaningful comparison of information from different sources.

The scientific method is appealingly linear in the abstract: an observation leads to a question, which leads to a hypothesis, which leads to an experiment, which leads to a result, which (if you're lucky) can lead to another question, and so the process begins again.

Testing hypotheses by gathering evidence is a core concern of science. What most scientists will tell you, though, is that their work tends not to progress as tidily as the scientific method looks on paper. More often they move back and forth between these steps, because science is an *iterative process*: a repeating process in which the end result is used as a starting point for the next run. Researchers often repeat the same steps over and over in order to test new ideas and tools, to deepen their questions about what they're studying, and to figure out how to do their research more effectively and accurately.

Researchers also test each other's hypotheses, because modern science demands that a result be *replicable*: that different people conducting identical experiments can come up with very similar, even identical results. If an experiment's results cannot be duplicated independently of the original researcher or team, then those results are cast into doubt.

Still, the scientific method is featured in the early chapters of many a Science 101 textbook because it's a good jumping-off point for learning how to set up an experiment and collect data.

Steps in the Scientific Method

At their most basic, the steps in the scientific method go like this:

- Observe something in the world.
- Ask a question about it.
- Formulate a potential answer (a hypothesis) for it.
- Conduct an experiment that tests the hypothesis.
- Compare the predicted result to the actual result:
  - — Result supports the hypothesis.
  - — Result doesn't support the hypothesis.
  - — Result partially supports the hypothesis.
- Consider the result.
- Ask another question and begin again.

Let's look more closely at each step.

Observe Something in the World

Observation and exploration of what's going on in the environment is essential to figuring out what questions to ask. *Asking a question* really means "asking an answerable question," one that you can then test with an experiment. Testable questions begin with how, what, when, who, or which ("why" is impossible to answer).

Ask an Answerable Question

Devising a good experiment question can itself involve several steps. Is your question uninteresting or interesting? Can it be narrowed down to look at a single thing, to collect data on one variable only (an observational question), or to change one variable and learn what results (a manipulative question)? It's important to test only one variable—a factor that exists at different levels or amounts—at a time in your experiment, in order to be reasonably sure your test and the conclusions you draw from it are valid. If you test more than one variable at a time, then cause and effect relationships are much less clear.

An example of a testable question that would work with one of the gadgets in this book is, "When are atmospheric hydrocarbon levels outside my window at their highest concentration?"

Formulate a Hypothesis

Formulating a hypothesis involves using what you already know to come up with a potential answer to your question: an explanation for what you've observed. It's not merely an educated guess; it is your formal statement of what you're going to test (the variable) and a prediction of what the results will be.

For example, working off the previous question, you could form your hypothesis statement as, "If heavier car traffic increases atmospheric hydrocarbons outside my window, and I measure those levels from 4 to 6 pm as well as from 4 to 6 am, then I should detect higher levels from 4 to 6 pm, which is afternoon rush hour." The IF statement is your hypothesis; the AND statement is the design of your experiment; the THEN statement is your prediction of what you'll learn from the experiment. Since what you're measuring will be atmospheric hydrocarbons, the variable in this experiment is time.

Compare the Predicted to Actual Results, Considering the Results

On the face of it, this experiment sounds like a no-brainer: rush hour traffic means more car exhaust means higher levels of hydrocarbons, right? If your results support your hypothesis, then this experiment may be over.

But what if levels are low during both time periods? Those results fail to support your hypothesis. It wouldn't hurt to check your build and programming, to be sure the gadget is working correctly. Assuming it is, you may need to ask a new question, or broaden the scope of your experiment—such as taking measurements more often during the day, or on different days of the week, or during different types of weather.

And what if you get high hydrocarbon levels from 4 to 6 pm, and also from 4 to 6 am? That's a partial confirmation of your hypothesis that may lead you to...

Ask Another Question

Maybe there's something going on in the world around you that you didn't know about, like a delivery truck idling on the street early in the morning (we get that a lot on our street!). Do deliveries happen often enough to affect hydrocarbon levels most of the time, or just some of the time? To learn more, you'll need to figure out the next interesting, testable question, reformulate your hypothesis, and restructure your experiment.

By testing enough variables (time of day, day of week, month, weather conditions, etc.) you should be able to build up a very accurate profile of the hydrocarbon pollution outside your window. In fact, you should be able to predict future events: for example, if tomorrow is a delivery day, you could confidently predict there will be more pollution than usual. Or if tomorrow is going to be rainy, there will be less pollution, as the raindrops wash the pollution out of the atmosphere.

When you feel you've gotten the pollution profile for your neighborhood down pat, your work has just begun—now it's time to measure a different neighborhood! Science never stops!

About the Authors

Patrick Di Justo is a contributor to *Wired* magazine—where he writes the magazine's monthly "What's Inside" column—and the author of *The Science of Battlestar Galactica* (Wiley, October 2010). His work has appeared in *Dwell*, *Scientific American*, *Popular Science*, *The New York Times*, and more. He has also worked as a robot programmer for the Federal Reserve. He bought his first Arduino in 2007.

Emily Gertz has been covering DIY environmental monitoring since 2004, when she interviewed engineer-artist Natalie Jeremijenko for Worldchanging.com. Her work has also appeared in *Popular Science*, *Popular Mechanics*, *Scientific American*, *Grist*, *Dwell*, *OnEarth*, and more. She has been hands-on with Internet technologies since 1994 as a web producer, community host, and content and social media strategist.

Built with Atlas. O'Reilly Media, Inc., 2013.