Adith Jagdish Boloor, Samarth Shah, Utsav Shah, Marco Schwartz

Arduino: Building exciting LED based projects and espionage devices

Learning Path

Transform your tiny device into a secret agent gadget by designing and building fantastic devices and creative LED-based projects using the Arduino platform

Packt

Arduino: Building exciting LED based projects and espionage devices

Transform your tiny device into a secret agent gadget by designing and building fantastic devices and creative LED-based projects using the Arduino platform

A course in three modules

Packt

BIRMINGHAM - MUMBAI

Arduino: Building exciting LED based projects and espionage devices

Published on: Month 2011

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78712-107-2

www.packtpub.com

Credits

Authors
Adith Jagdish Boloor
Samarth Shah
Utsav Shah
Marco Schwartz

Reviewers
Tim Gorbunov
Francis Perea
Roberto Gallea

Content Development Editor
Abhishek Jadhav

Graphics
Kirk D' Phena

Production Coordinator
Shantanu N. Zhagade

Preface

Arduino an open source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. The open source Arduino software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open source software. With the growing interest in home-made, weekend projects among students and hobbyists alike, Arduino offers an innovative and feasible platform to create projects that promote creativity and technological tinkering

What this learning path covers

Module 1, Arduino by Example, introduces the reader to the Arduino platform. We begin with acquiring the necessary components and installing the software to write your first program and see the magic begin. This module brings in commonly used Arduino-friendly components such as an ultrasound sensor and a small programmable LCD panel, and put them together to create a digital ruler, which is capable of measuring distances using the sensor and displaying them in real time on the LCD screen. In this module, we use basic algorithms that allow the Arduino to translate finger gestures into corresponding characters that are then displayed graphically using software called Processing. We introduce the reader to using PIR sensors or motion sensors, implementing a remote camera with Arduino, and linking the Arduino to a smart phone. Additionally, the reader will learn about Python and how it interfaces with Arduino. This module combines the elements learned in the preceding project with a device that uses a sensor to detect motion at an entry point, which triggers a security camera to take the intruder's photo via Bluetooth and sends that image to your smart phone. Some chapters explain connecting the Arduino to the Wi-Fi network using a relay to control an electric appliance and communicating to it using Telnet. We use the Arduino to create a simple home automation system operating within the bounds of the Wi-Fi that would allow the user to control home appliances using a computer, smart phone, and their voice. A few chapters revolve around building a robot dog from scratch. You learn about the Arduino MEGA board, servos, and stand-alone power requirements for the board. This involves using household items to build the chassis of the dog and then completing the circuit using the Arduino MEGA board and a lot of servos. The reader will finally finish building the robot and will learn to calibrate and teach (program) the robot to stand, walk, and play. Also, finally, speech recognition will be implemented so that the dog can actually listen to the user.

Module 2, Arduino BLINK Blueprints, presents you with some cool stuff of controlling LEDs and will show you how to control different LEDs with an artistic approach. This module teaches you the basics of IR LEDs and IR communication. Once you have learned about programming an IR sensor, you will use it to control the TV backlight using a remote. We learn about soldering and creating a 4*4*4 LED Cube using the Arduino UNO board. Some chapters show you how to visualize sound using Arduino and then we will develop an LED Christmas tree. We also create a Persistence of Vision wand using an LED array and a motor. A chapter in this module starts with common troubleshooting techniques. We discuss resources that will be useful if you want to do advanced stuff with Arduino.

Module 3, Arduino for Secret Agents, is about building an alarm system that is based on the Arduino platform with a motion sensor and a visual alarm. We build a secret recording system that can record the conversations and noises in a room. A few chapters in this module teach you about creating a detector to check whether there are other secret agent devices in a room and an access control system using your own fingerprint. We build a project where the secret agent can open a lock just by sending a text message to the Arduino device and also make a spy camera that can be accessed from anywhere in the world to record pictures in Dropbox when motion is detected. We learn to secretly record any kind of data and log in this data on the Cloud. One of the chapters in this module is about creating a GPS tracker that indicates its position on a map in real time. We conclude this module by building a small surveillance robot that can spy on your behalf.

What you need for this learning path

The primary softwares required are as follows:

- Arduino IDE
- Processing IDE
- Python 2.7
- BitVoicer
- Teraterm
- Putty

We will be using a wide range of Arduino boards, shields, and hardware components. You will find all the details about these requirements in the relevant chapters.

Who this learning path is for

This course is aimed to a wide range of readers. It can be really illustrative to those wanting to be introduced to the development of projects based on microcontrollers and using Arduino in particular for the first time. It can also be interesting to all those who already know or have worked with microcontrollers previously but haven't tried Arduino and still want to know the basics about this powerful platform by way of a number of projects that will present all important aspects of the platform. Novice programmers and hobbyists who want to delve deeper into the world of Arduino

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this course from your account at `http://www.packtpub.com`. If you purchased this course elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at `https://github.com/PacktPublishing/Arduino-Building-exciting-LED-based-projects-and-espionage-devices/tree/master`. We also have other code bundles from our rich catalog of books, videos, and courses available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

Module 1: Arduino By Example

Module 2: Arduino BLINK Blueprints

Module 3: Arduino for Secret Agents

Module 1

Arduino By Example

Design and build fantastic projects and devices using the Arduino platform

1
Getting Started with Arduino

Hello there! If you are reading this book right now, it means that you've taken your first step to make fascinating projects using Arduinos. This chapter will teach you how to set up an Arduino and write your first Arduino code.

You'll be in good hands whilst you learn some of the basics aspects of coding using the Arduino platform; this will allow you to build almost anything including robots, home automation systems, touch interfaces, sensory systems, and so on. Firstly, you will learn how to install the powerful Arduino software, then set that up, followed by hooking up your Arduino board and, after making sure that everything is fine and well, you will write your first code! Once you are comfortable with that, we will modify the code to make it do something more, which is often what Arduino coders do. We do not just create completely new programs; often we build on what has been done before, to make it better and more suited to our objectives. The contents of this chapter are divided into the following topics:

- Prerequisites
- Setting up
- Hello World
- Summary

Prerequisites

Well, you can't jump onto a horse without putting on a saddle first, can you? This section will cover what components you need to start coding on an Arduino. These can be purchased from your favorite electrical hobby store or simply ordered online.

Materials needed

- 1x Arduino-compatible board such as an Arduino UNO
- 1x USB cable A to B
- 2x LEDs
- 2x 330Ω resistors
- A mini breadboard
- 5x male-to-male jumper wires

Note

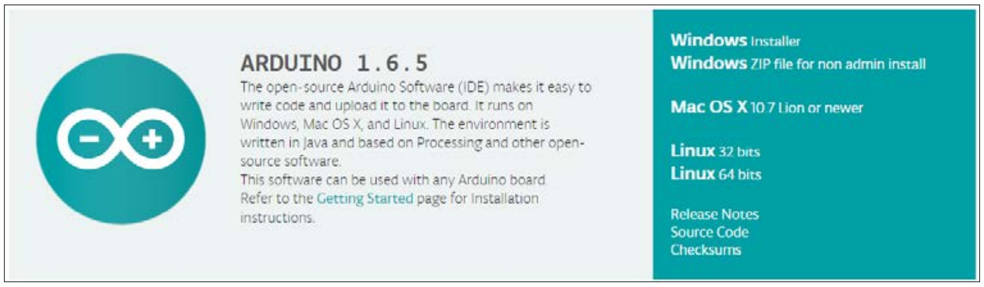
The UNO can be substituted for any other Arduino board (Mega, Leonardo, and so on) for most of the projects. These boards have their own extra features. For example, the Mega has almost double the number of I/O (input/output) pins for added functionality. The Leonardo has a feature that enables it to control the keyboard and mouse of your computer.

Setting up

This topic involves downloading the Arduino software, installing the drivers, hooking up the Arduino, and understanding the IDE menus.

Downloading and installing the software

Arduino is open source-oriented. This means all the software is free to use non-commercially. Go to `http://arduino.cc/en/Main/Software` and download the latest version for your specific operating system. If you are using a Mac, make sure you choose the right Java version; similarly on Linux, download the 32-or 64-bit version according to your computer.



ARDUINO 1.6.5

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Arduino download page

Windows

Once you have downloaded the setup file, run it. If it asks for administrator privileges, allow it. Install it in its default location (`C:\Program Files\Arduino` or `C:\Program Files (x86)\Arduino`). Create a new folder in this location and rename it `My Codes` or something where you can conveniently store all your programs.

Mac OS X

Once the ZIP file has finished downloading, double-click to expand it. Copy the Arduino application to the `Applications` folder. You won't have to install additional drivers to make the Arduino work since we will be using only the Arduino UNO and MEGA throughout the book. You're all set.

If you didn't get anything to work, go to `https://www.arduino.cc/en/guide/macOSX`.

Linux (Ubuntu 12.04 and above)

Once you have downloaded the latest version of Arduino from the preceding link, install the compiler and the library packages using the following command:

```
sudo apt-get update && sudo apt-get install arduino arduino-core
```

If you are using a different version of Linux, this official Arduino walkthrough at `http://playground.arduino.cc/Learning/Linux` will help you out.

Connecting the Arduino

It is time to hook up the Arduino board. Plug in the respective USB terminals to the USB cable and the tiny LEDs on the Arduino should begin to flash.

Arduino UNO plugged in

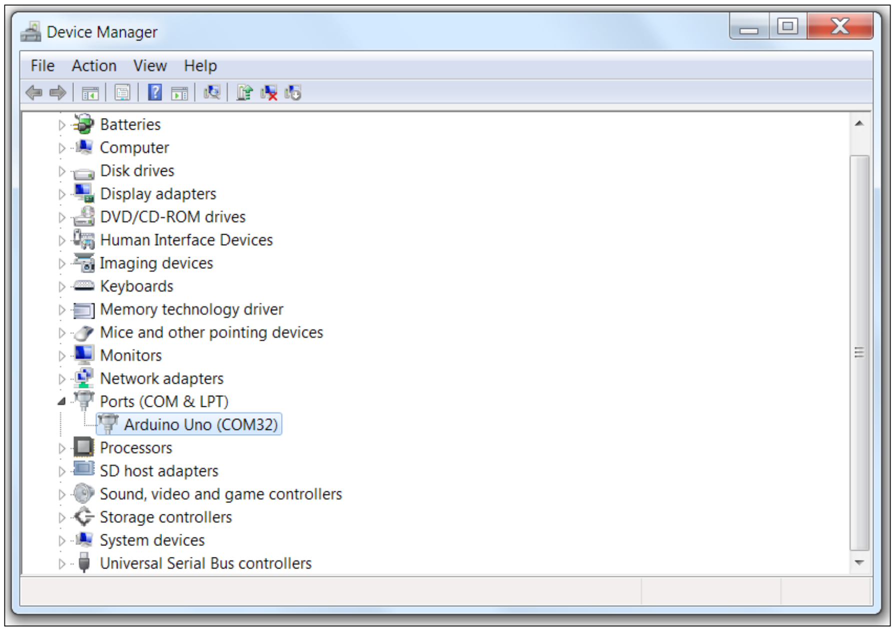
If the LEDs didn't turn on, ensure that the USB port on your computer is functioning and make sure the cable isn't faulty. If it still does not light up, there is something wrong with your board and you should get it checked.

Windows

The computer will begin to install the drivers for the Arduino by itself. If it does not succeed, do the following:

1. Open **Device Manager**.
2. Click on **Ports (COM & LPT)**.
3. Right-click on **Unknown Device** and select **Properties**.
4. Click on **Install Driver** and choose **browse files on the computer**.
5. Choose the `drivers` folder in the previously installed `Arduino` folder.

The computer should say that your Arduino UNO (USB) has been successfully installed on COM port (xx). Here xx refers to a single or double digit number. If this message didn't pop up, go back to the **Device Manager** and check if it has been installed under **COM** ports.



Arduino UNO COM port

Remember the (COMxx) port that the Arduino UNO was installed on.

Mac OS X

If you are using Mac OS, a dialog box will tell you that a new network interface has been detected. Click **Network Preferences** and select **Apply**. Even though the Arduino board may show up as **Not Configured**, it should be working perfectly.

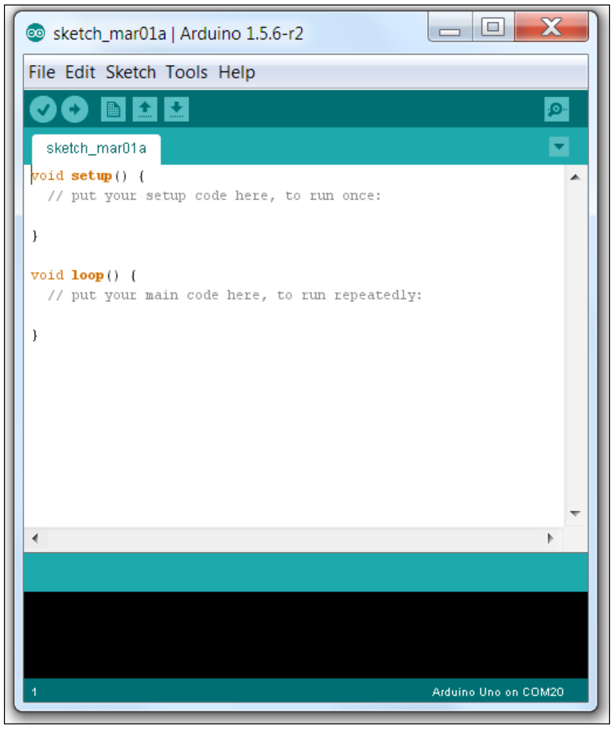
Linux

You are ready to go.

The serial ports for Mac OS and Linux will be obtained once the Arduino software has been launched.

The Arduino IDE

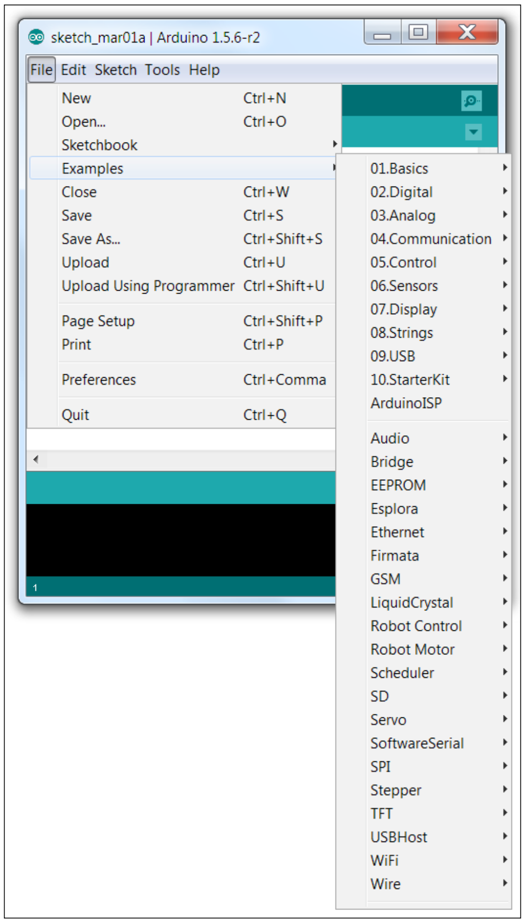
The Arduino software, commonly referred to as the Arduino IDE (Integrated Development Environment), is something that you will become really familiar with as you progress through this book. The IDE for Windows, Mac OS, and Linux is almost identical. Now let's look at some of the highlights of this software.



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

}

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

}
```

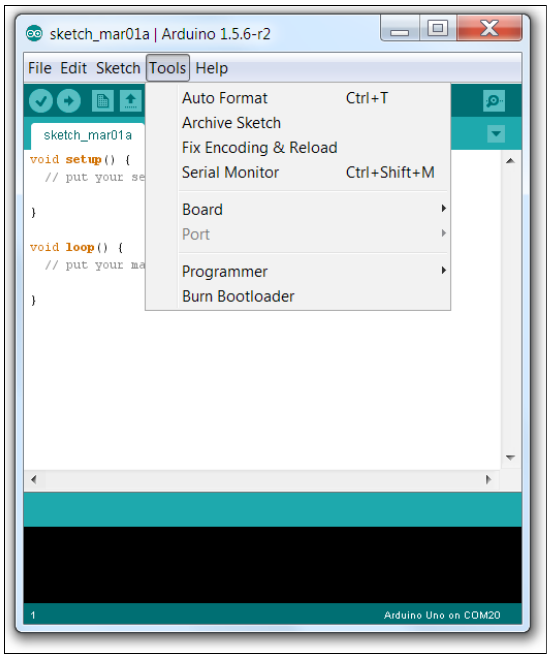
Arduino IDE

This is the window that you will see when you first start up the IDE. The tick/check mark verifies that your code's syntax is correct. The arrow pointing right is the button that uploads the code to the board and checks if the code has been changed since the last upload or verification. The magnifying glass is the **Serial Monitor**. This is used to input text or output debugging statements or sensor values.



Examples of Arduino

All Arduino programmers start by using one of these examples. Even after mastering Arduino, you will still return here to find examples to use.



```
void setup() {
  // put your se

}

void loop() {
  // put your ma

}
```

Arduino tools

The screenshot shows the tools that are available in the Arduino IDE. The **Board** option opens up all the different boards that the software supports.

Hello World

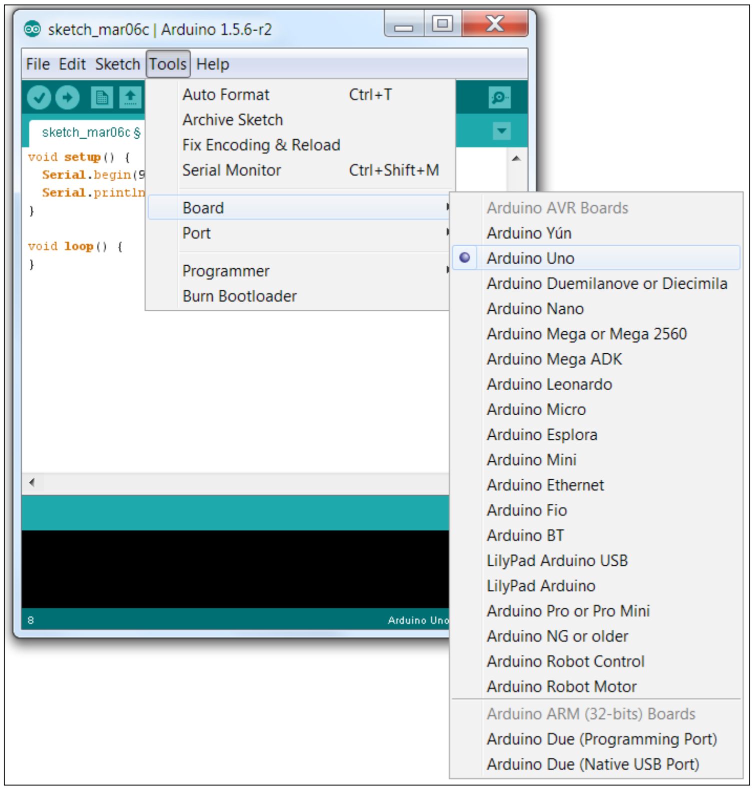
The easiest way to start working with Arduinos begins here. You'll learn how to output print statements. The Arduino uses a **Serial Monitor** for displaying information such as print statements, sensor data, and the like. This is a very powerful tool for debugging long codes. Now for your first code!

Writing a simple print statement

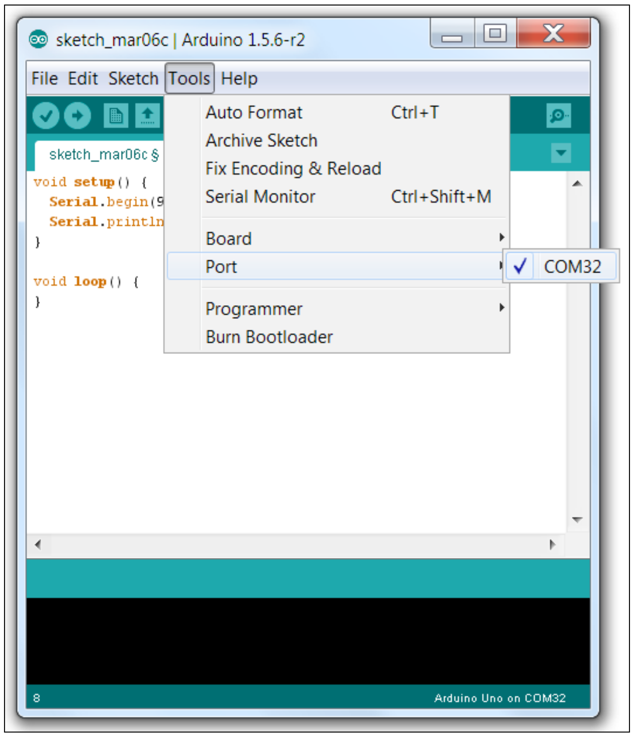
Open up the Arduino IDE and copy the following code into a new sketch:

```
void setup() {
Serial.begin(9600);
Serial.println("Hello World!");
}

void loop() {
}
```

Open **Tools** | **Board** and choose **Arduino UNO,** as shown in the following screenshot:



Open **Tools** | **Port** and choose the appropriate port (remember the previous COM xx number? select that), as shown in the following screenshot. For Mac and Linux users, once you have connected the Arduino board, going to **Tools** | **Serial Port** will give you a list of ports. The Arduino is typically something like `/dev/tty.usbmodem12345` where *12345* will be different.



```
void setup() {
  Serial.begin(9
  Serial.println
}

void loop() {
}
```

Selecting the Port

Finally, hit the Upload button. If everything is fine, the LEDs on the Arduino should start flickering as the code is uploaded to the Arduino. The code will then have uploaded to the Arduino.

To see what you have accomplished, click on the **Serial Monitor** button on the right side and switch the baud rate on the **Serial Monitor** window to 9600.

You should see your message `Hello World!` waiting for you there.

Using serial communication

Serial communication is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port which is also known as a UART. Serial data transfer is when we transfer data one bit at a time, one right after the other. Information is passed back and forth between the computer and Arduino by, essentially, setting a pin to high or low. Just like we used that technique to turn an LED on and off, we can also send data. One side sets the pin and the other reads it.

In this section, you will see two examples. In the first example, Arduino will send data to the computer using serial communication, while in the second example, by sending a command (serial) from the computer, you can control the functionality of the Arduino board.

Serial write

In this example, the Arduino board will communicate with the computer using the serial port, which can be viewed on your machine using the Serial Monitor.

Write the following code to your Arduino editor:

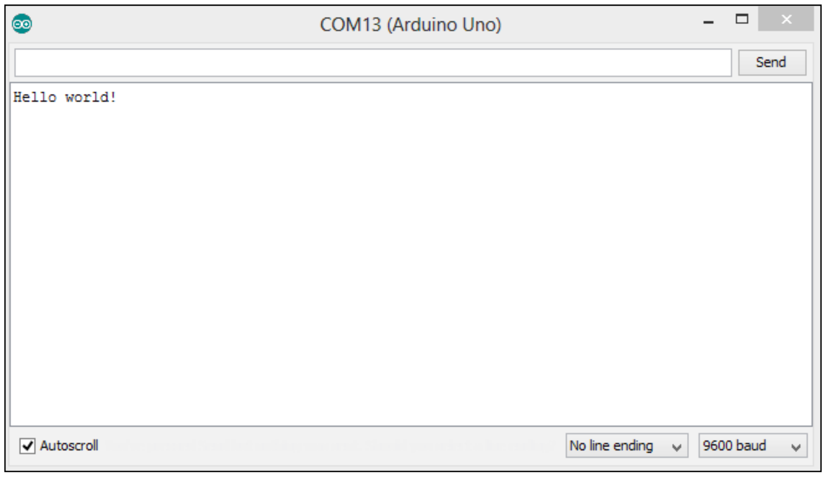
```
void setup()                    // run once, when the sketch starts
{
  Serial.begin(9600);           // set up Serial library at 9600 bps

  Serial.println("Hello world!");  // prints hello with ending line
break
}

void loop()                       // run over and over again
{
                                  // do nothing!
}
```

Even if you have nothing in the setup or loop procedures, Arduino requires them to be there. That way it knows you really mean to do nothing, as opposed to forgetting to include them!

`Serial.begin` sets up Arduino with the transfer rate we want, in this case 9600 bits per second. `Serial.println` sends data from Arduino to the computer.

Once you compile and upload it to your connected Arduino board, open **Serial Monitor** from the Arduino IDE. You should be able to see the **Hello world!** text being sent from the Arduino board:

If you have trouble locating **Serial Monitor**, check the *Understanding Arduino IDE* section of this chapter.



```
Hello world!
```

No line ending
9600 baud

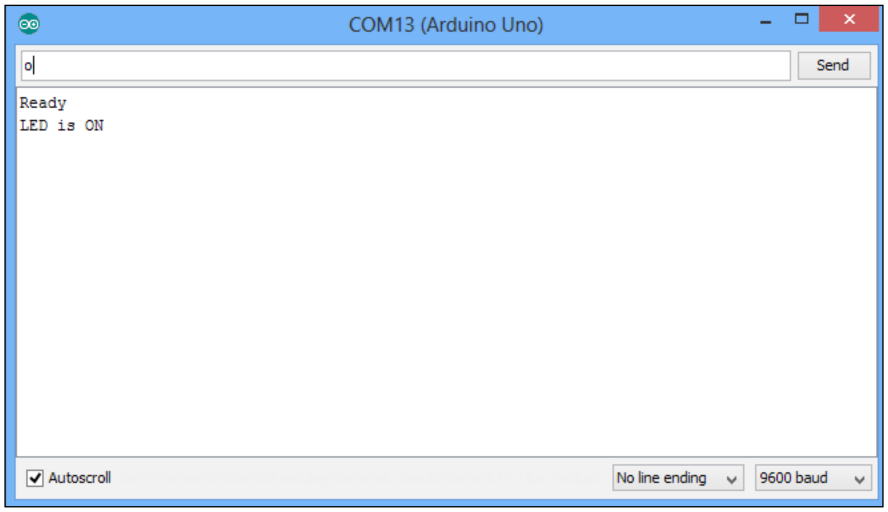
Serial read

In the previous example, serial library was used to send a command from Arduino to your computer. In this example, you will send a command from the computer, and Arduino will do a certain operation (turn on/off LED) based on the command received:

```
int inByte; // Stores incoming command

void setup() {
```

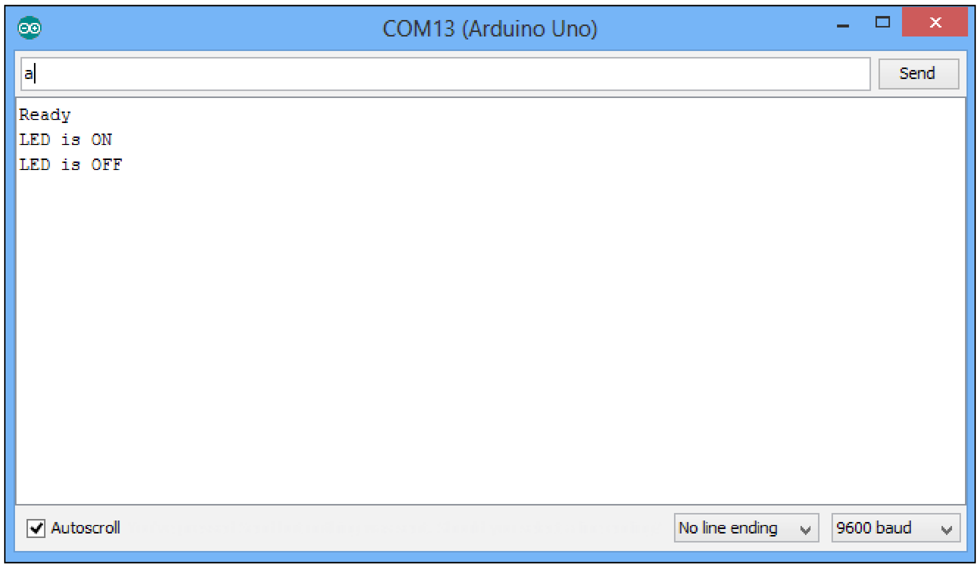
```
Serial.begin(9600);
pinMode(13, OUTPUT); // LED pin
Serial.println("Ready"); // Ready to receive commands
}
void loop() {
  if(Serial.available() > 0) { // A byte is ready to receive
    inByte = Serial.read();
    if(inByte == 'o') { // byte is 'o'
      digitalWrite(13, HIGH);
      Serial.println("LED is ON");
      }
      else
      {
        // byte isn't 'o'
        digitalWrite(13, LOW);
        Serial.println("LED is OFF");
      }
    }
}
```

The `inByte` function will store the incoming serial byte. From the previous example, you should be familiar with the commands written in the `setup` function. In the loop function, first you need to know when a byte is available to be read. The `Serial.available()` function returns the number of bytes that are available to be read. If it is greater than 0, `Serial.read()` will read the byte and store it in an `inByte` variable. Let's say you want to turn on the LED when the letter 'o' is available. For that you will be using the `if` condition, and you will check whether the received byte is 'o' or not. If it is 'o', turn on the LED by setting pin 13 to `HIGH`. Arduino will also send an **LED is ON** message to the computer, which can be viewed in Serial Monitor:



```
Ready
LED is ON
```

If it is any other character, then turn off the LED by setting pin 13 to `LOW`. Arduino will also send an **LED is OFF** message to the computer, which can be viewed in Serial Monitor:



```
Ready
LED is ON
LED is OFF
```

The world of LED

LED stands for light emitting diode, so it emits light when sufficient voltage is provided across the LED anode and cathode. Today's LEDs are available in many different types, shapes, and sizes – a direct result of the tremendous improvements in semiconductor technology over recent years. These advancements have led to better illumination, longer service life, and lower power consumption. They've also led to more difficult decision making, as there are so many types of LED to choose from.

LEDs can be categorized into miniature, high power, and application-specific LEDs:

- **Miniature LEDs**: These LEDs are extremely small and usually available in a single color/shape. They can be used as indicators on devices such as cell phones, calculators, and remote controls.
- **High power LEDs**: Often referred to as high output LEDs, these are a direct result of improved diode technology. They offer a much higher lumen output than standard LEDs. Typically, these LEDs are used in car headlights.

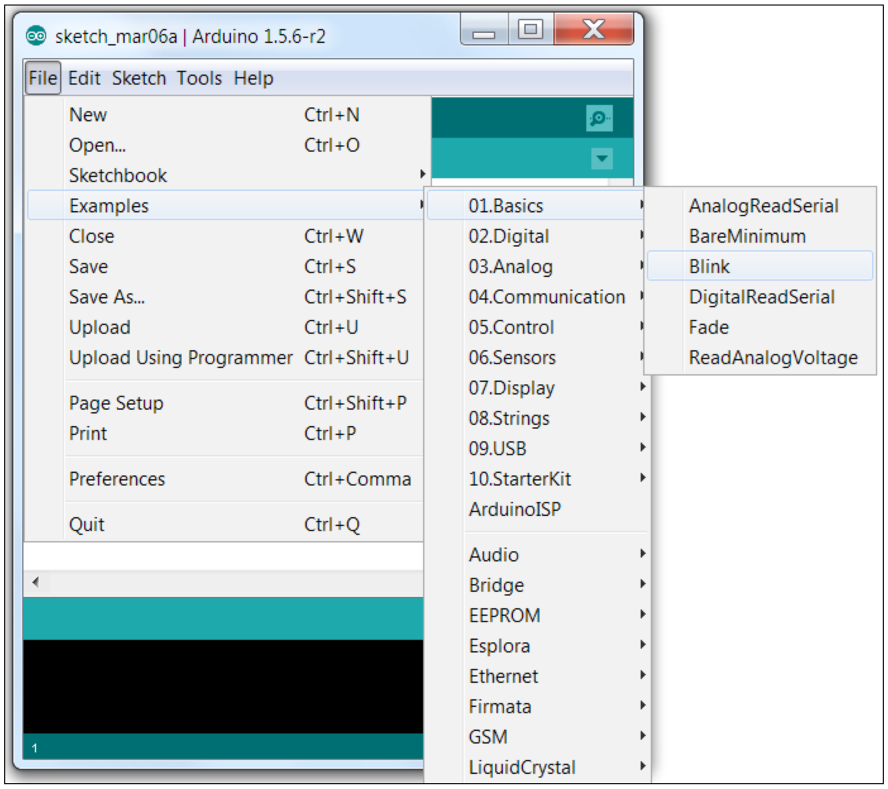
- **Application-specific LEDs**: As the name suggests, there are many LEDs that fall under this category. These are flash LEDs, RGB LEDs, seven segment display, LED lamps, and LED bars.

LED blink

That wasn't too bad but it isn't cool enough. This little section will enlighten you, literally.

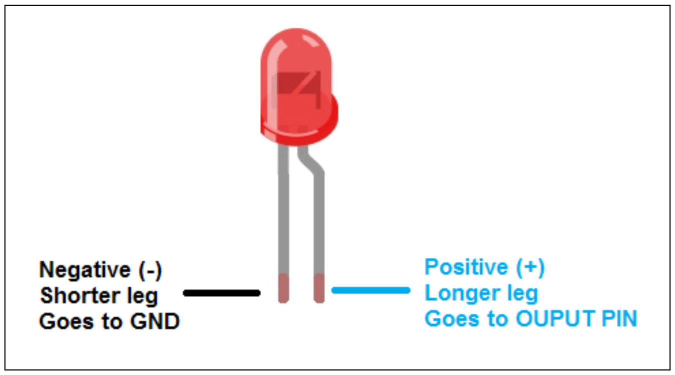
Open up a new sketch.

Go to **File** | **Examples** | 01. **Basics** | **Blink**.



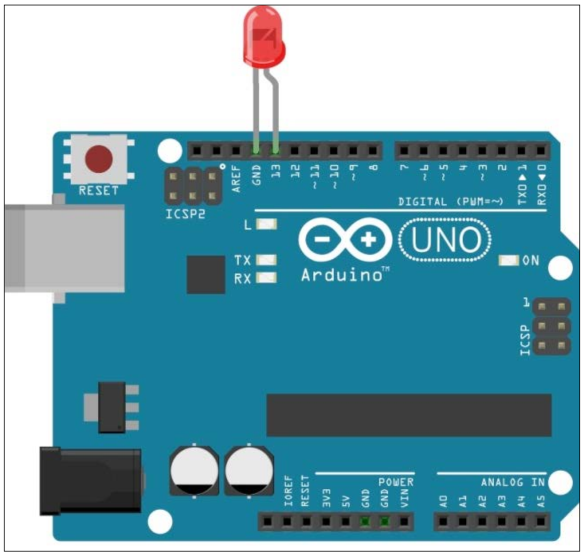
Blink example

Before we upload the code, we need to make sure of one more thing. Remember the LED that we spoke about in the prerequisites? Let's learn a bit about it before plugging it in, as shown in the following image:



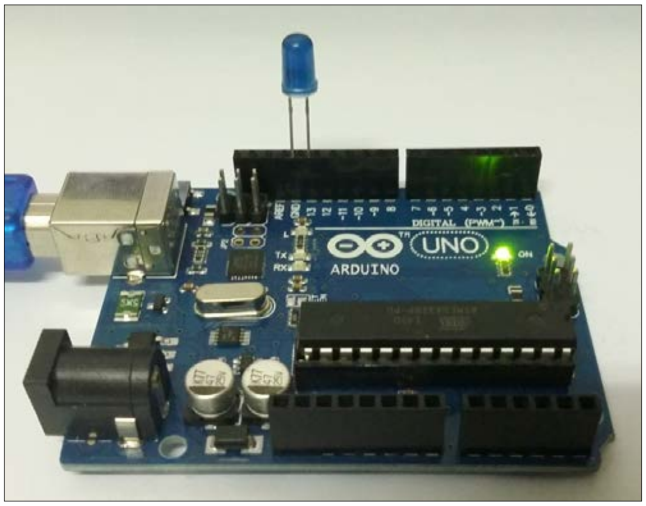
LED basics

We will make use of it now. Plug in the LED such that the longer leg goes into pin 13 and the shorter leg goes into the GND pin, as in the following:



LED blink setup (Fritzing)

This diagram is made using software called Fritzing. This software will be used in future projects to make it cleaner to see and easier to understand as compared to a photograph with all the wires running around. Fritzing is open source software which you can learn more about at `www.fritzing.org`.



Arduino LED setup

Upload the code. Your LED will start blinking, as shown in the following image.

A lit up LED

Isn't it just fascinating? You just programmed your first hardware. There's no stopping you now. Before advancing to the next chapter, let's see what the code does and what happens when you change it.

This is the blink example code that you just used:

```
/*
Blink
Turns on an LED on for one second, then off for one second,
repeatedly.

This example code is in the public domain.
*/

//Pin 13 has an LED connected on most Arduino boards.
//give it a name:
int led = 13;

//the setup routine runs once when you press reset:
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}

//the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage
level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage
LOW
  delay(1000);               // wait for a second
}
```

We have three major sections in this code. This format will be used for most of the projects in the book.

```
int led = 13;
```

This line simply stores the numerical PIN value onto a variable called `led`.

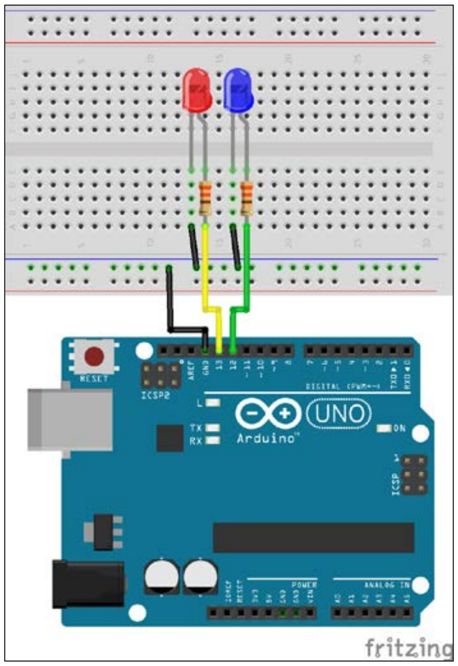
```
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
```

This is the `setup` function. Here is where you tell the Arduino what is connected on each used pin. In this case, we tell the Arduino that there is an output device (LED) on pin `13`.

```
void loop() {
digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage
level)
delay(1000);               // wait for a second
digitalWrite(led, LOW);    // turn the LED off by making the voltage
LOW
delay(1000);               // wait for a second
}
```

This is the `loop` function. It tells the Arduino to keep repeating whatever is inside it in a sequence. The `digitalWrite` command is like a switch that can be turned ON (`HIGH`) or OFF (`LOW`). The `delay(1000)` function simply makes the Arduino wait for a second before heading to the next line.

If you wanted to add another LED, you'd need some additional tools and some changes to the code. This is the setup that you want to create.



Connecting two LEDs to an Arduino

If this is your first time using a breadboard, take some time to make sure all the connections are in the right place. The colors of the wires don't matter. However, GND is denoted using a black wire and VCC/5V/PWR is denoted with a red wire. The two resistors, each connected in series (acting like a connecting wire itself) with the LEDs, limit the current flowing to the LEDs, making sure they don't blow up.

As before, create a new sketch and paste in the following code:

```
/*
Double Blink
Turns on and off two LEDs alternatively for one second each
repeatedly.

This example code is in the public domain.
*/

int led1 = 12;
int led2 = 13;

void setup() {
// initialize the digital pins as an output.
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);

// turn off LEDs before loop begins
digitalWrite(led1, LOW);   // turn the LED off (LOW is the voltage
level)
digitalWrite(led2, LOW);   // turn the LED off (LOW is the voltage
level)
}

//the loop routine runs over and over again forever:
void loop() {
digitalWrite(led1, HIGH);   // turn the LED on (HIGH is the voltage
level)
digitalWrite(led2, LOW);    // turn the LED off (LOW is the voltage
level)
delay(1000);                // wait for a second
digitalWrite(led1, LOW);    // turn the LED off (LOW is the voltage
level)
digitalWrite(led2, HIGH);   // turn the LED on (HIGH is the voltage
level)
delay(1000);                // wait for a second
}
```

Once again, make sure the connections are made properly, especially the positive LEDs (the longer one to OUTPUT PIN) and the negative (the shorter to the GND) terminals. Save the code as `DoubleBlink.ino`. Now, if you make any changes to it, you can always retrieve the backup.

Upload the code. 3… 2… 1… And there you have it, an alternating LED blink cycle created purely with the Arduino. You can try changing the delay to see its effects.

For the sake of completeness, I would like to mention that you could take this mini-project further by using a battery to power the system and decorate your desk/room/house. More on how to power the Arduino will be covered in subsequent chapters.

Summary

You have now completed the basic introduction to the world of Arduino. In short, you have successfully set up your Arduino and have written your first code. You also learned how to modify the existing code to create something new, making it more suitable for your specific needs. This methodology will be applied repeatedly while programming, because almost all the code available is open source and it saves time and energy.

In the next chapter, we will look into sensors and displays. You will build a digital ruler that you can use to measure short distances. It will consist of an ultrasound sensor to compute distance and a small LCD screen to display it. Additionally, we will look at safely powering the Arduino board using a battery so that you are not dependent on your computer for USB power every time.

2
Digital Ruler

You've made it to chapter 2! Congrats! From now on things are going to get a bit complicated as we try to make the most of the powerful capabilities of the Arduino micro controller. In this chapter we are going to learn how to use a sensor and an LCD board to create a digital LCD ruler.

Put simply, we will use the ultrasound sensor to gauge the distance between the sensor and an object. We will use the Arduino and some math to convert the distance into meaningful data (cm, inches) and finally display this on the LCD.

- Prerequisites
- Using an ultrasound sensor
- Hooking up an LCD to the Arduino
- Displaying the sensor data on the LCD
- Summary

Prerequisites

The following is a list of materials that you'll need to start coding on an Arduino; these can be purchased from your favorite electrical hobby store or simply ordered online:

- 1 x Arduino-compatible board such as the UNO
- 1 x USB cable A to B 1 x HC—SR04 ultrasound sensor
- 1 x I2C LCD1602
- 10 x male to male wires
- 9V battery with 2.1 mm barrel jack connector (optional)
- Laser pointer (optional)

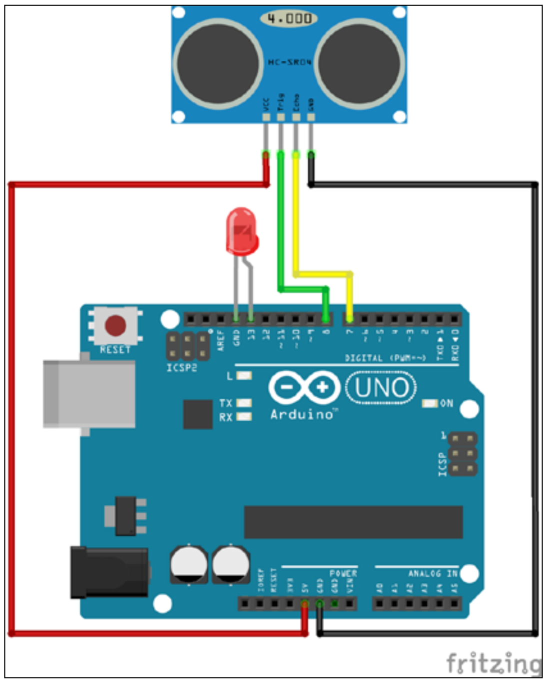
Components such as the LCD panel and the ultrasonic sensor can be found in most electronic hobby stores. If they are unavailable in a store near you, you will find online stores that ship worldwide.

A bit about the sensor

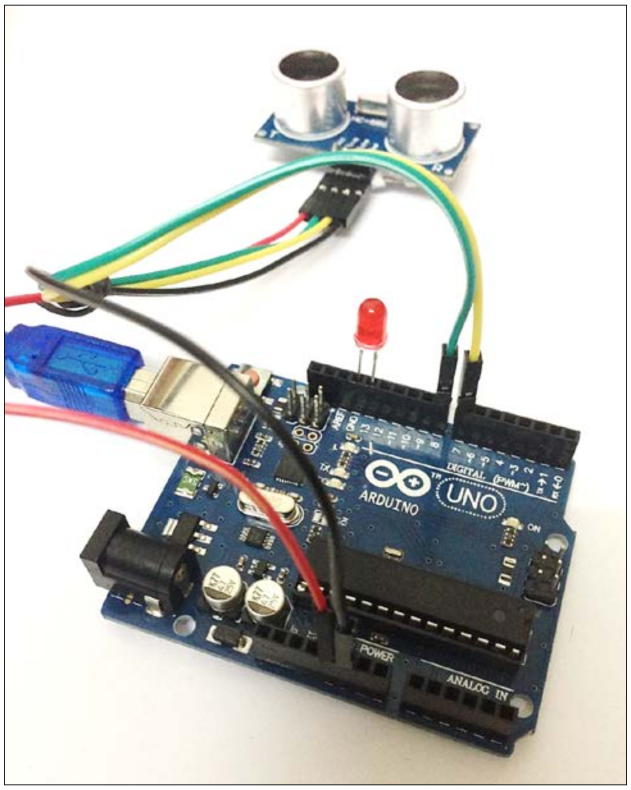
The SR04 is a very powerful and commonly used distance/proximity sensor. And that is what we are going to look at first. The SR04 sensor emits ultrasonic waves which are sound waves at such a high frequency (40 kHz) that they are inaudible to humans. When these waves come across an object, some of them get reflected. These reflected waves get picked up by the sensor and it calculates how much time it took for the wave to return. It then converts this time into distance.

We are firstly going to use this sensor to make a simple proximity switch. Basically, when you bring an object closer than the set threshold distance, an LED is going to light up.

This is the circuit that we need to construct. Again, be very careful about where everything goes and make sure there are no mistakes. It is very easy to make a mistake, no matter how much experience you've had with Arduinos.



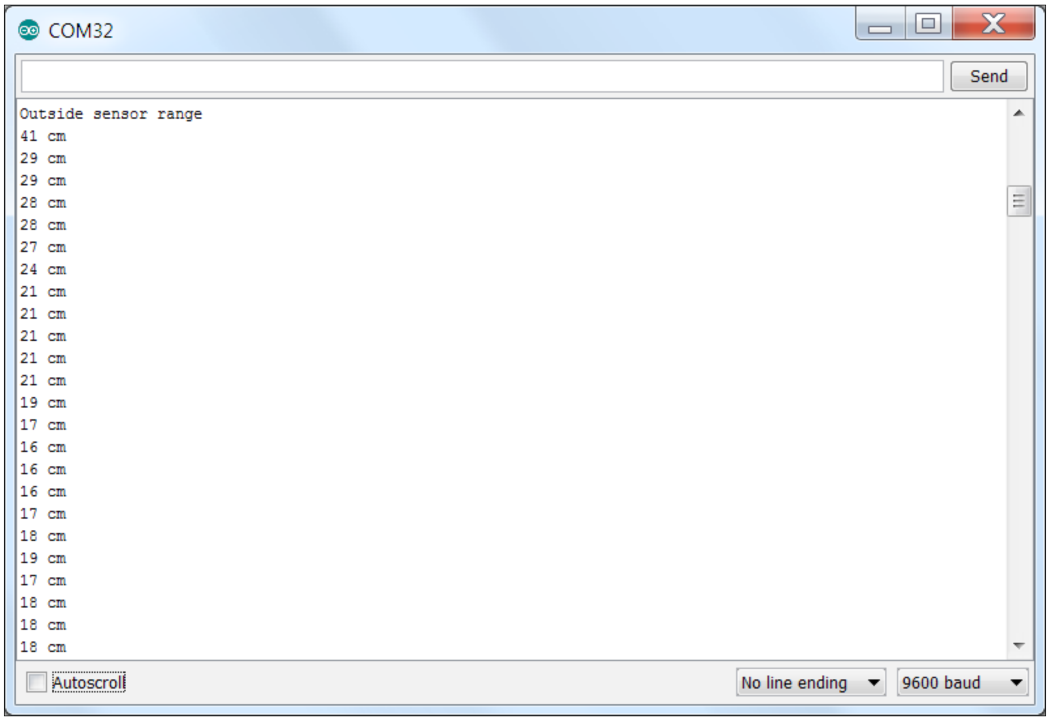
In reality it is going to look something like this, much messier than the **Fritzing** circuit depicted in the previous screenshot:



Open a new sketch on the Arduino IDE and load the `SR04_Blink.ino` program that came with this book.

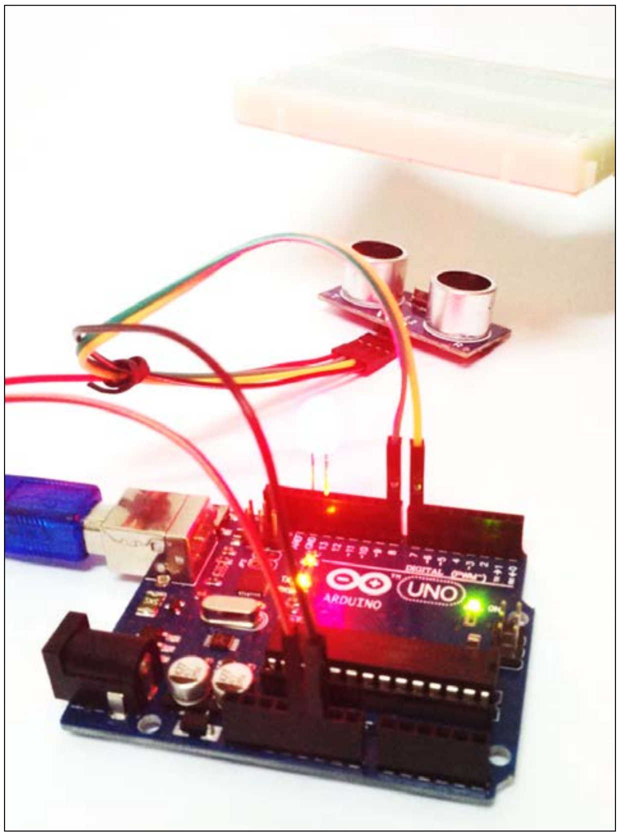
Save the code as `SR04_Blink.ino` in your codes directory. This enables us to keep the supplied code as a backup if we tweak it and end up messing up the program. Do this in every instance. Now, once more, check and ensure that the pins match the topmost lines of the code. Upload the code. Now open the **Serial Monitor** on the Arduino IDE and select 9600 as the baud rate. Place your hand or a flat surface (a book) in front of it and keep changing the distance.

You should be able to see the sensor distances being displayed on the screen, as in the following screenshot:



```
Outside sensor range
41 cm
29 cm
29 cm
28 cm
28 cm
27 cm
24 cm
21 cm
21 cm
21 cm
21 cm
21 cm
19 cm
17 cm
16 cm
16 cm
16 cm
17 cm
18 cm
19 cm
17 cm
18 cm
18 cm
18 cm
```

It says `Outside sensor range` if the sensor is picking up values greater than 200 cm because that is the most it can measure. Otherwise, if you make it point at nothing at a distance, it will still display around 200 cm because that is its range.



You will notice that, as you bring the object closer to the sensor than 15cm, it lights up. This is because the threshold is set at `15cms`, as you can see in the following code snippet:

```
  if (distance < 15) {  // Threshold set to 15 cm; LED turns off if
object distance < 15cms
    digitalWrite(ledPin,HIGH);
}
  else {
    digitalWrite(ledPin,LOW);
  }
  if (distance >= 200 || distance <= 0){
    Serial.println("Outside sensor range");
  }
  else {
    Serial.print(distance); // prints the distance on the serial
monitor
    Serial.println(" cm");
  }
  delay(500); // wait time between each reading
}
```

This is the same principle used in cars (the beeping sound while reversing if you are too close to a wall), except usually it is an infrared sensor that emits light instead of sound.

In the code, we had the following line:

```
distance = duration / 58
```

This line is used to convert a time interval into distance. I will briefly explain the logic behind this. Sound travels at 340m/s, which is 29 microseconds per centimeter. The ping needs to travel twice the distance (to the object and its rebound back to the sensor). Hence, we need to use 2*29 which is 58 microseconds per centimeter. This same logic is applied in the case of inches.

Now think about the maximum and minimum range. As seen in the above snippet, the maximum is set to 200 cm. Most hobby ultrasonic sensors can measure up to 200 cm without hassle, but this can be decreased according to your project. The minimum is set to 0cm because the sensor can indeed measure values at that distance but with lower accuracy.

In some cases, your Serial Monitor may be spamming 0cm as the sensor value, even though you know this is not the case. To fix this issue, simply replace `if (distance < 15)` with `if ((distance > 0) && (distance < 15))`.

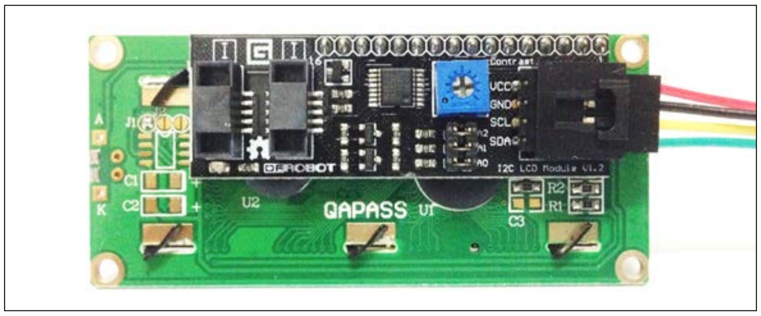
Now that you have learnt how to use the ultrasound sensor, let's move on to the LCD part of the project.

Hooking up an LCD to the Arduino

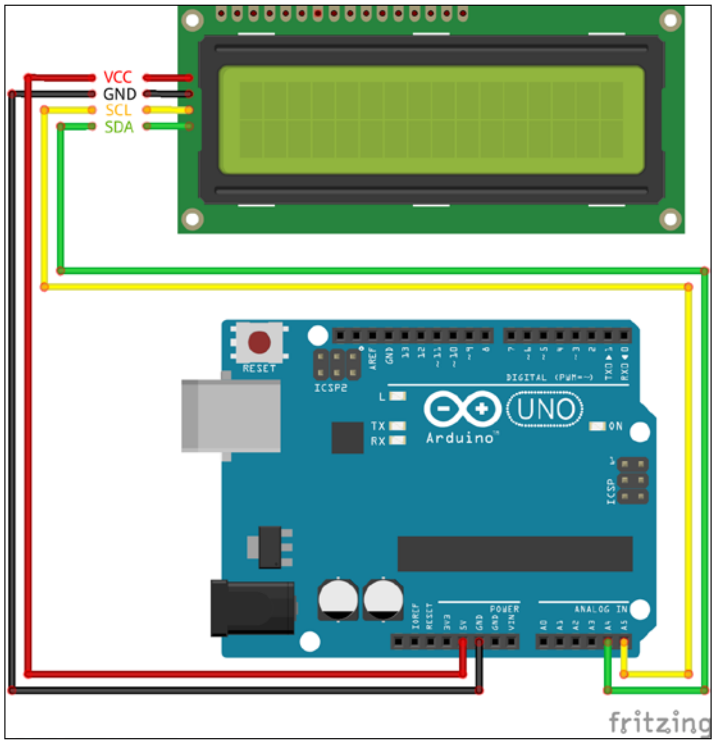
The LCD screen that we will be using is an I2C LCD1602.

This display screen can be programmed to display whatever you want in a 16x2 matrix. This means that the screen (as you will soon find out) has two rows capable of fitting 16 characters in each row.

Before setting up the complete circuit, look at the back of the LCD. Plug in four wires, as follows:



And then set up the circuit, as follows:



Now you will have to trust me on this next step. We are going to manually install a library that the LCD requires to run. You will be using this same method in future, so be patient and try to understand what we are doing here.

Download the `LiquidCrystal_I2C.zip` file from `http://www.wentztech.com/filevault/Electronics/Arduino/`.

Now, in the Arduino IDE, go to **Sketch** | **Include Library** | **Add ZIP library** and browse to the downloaded ZIP file. You are good to go.

If this doesn't work, you can manually extract the contents to: `C:\Users\<Username>\Documents\Arduino\libraries\LiquidCrystal_I2C` on Windows or `Documents/Arduino/libraries/LiquidCrystal_I2C` on the Mac and the same on Linux.

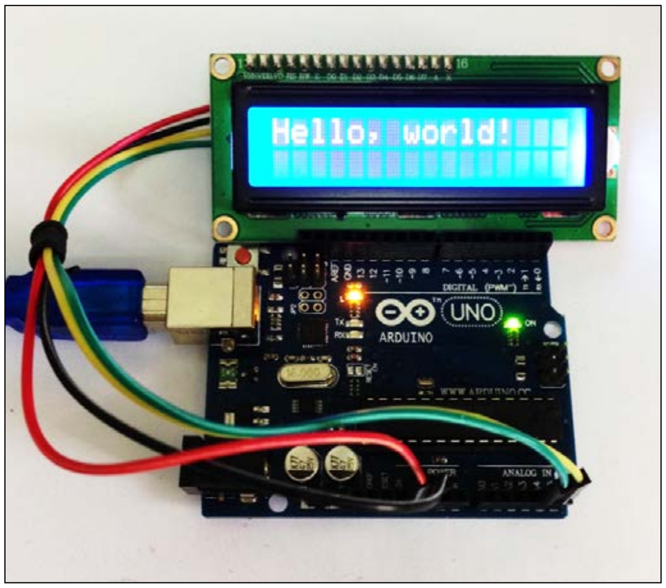
You will have to restart the IDE for it to be detected.

Now create a new sketch. Copy the following:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x20,16,2);  // set the LCD address to 0x20 for
a 16 chars and 2 line display

void setup()
{
  lcd.init();                      // initialize the lcd

  // Print a message to the LCD.
  lcd.backlight();
  lcd.print("Hello, world!");
}

void loop()
{
}
```

So few code lines after all that trouble? Yes! This is the beauty of using libraries, which enable us to hide all the complicated code so that the visible code is easier to read and understand. Save it as `I2C_HelloWorld.ino`. Plug in the Arduino and upload the code.

And you should have something like this:



At this moment you are free to play with it, change the text from `Hello, world!` to whatever you like, such as "I like Arduino!".

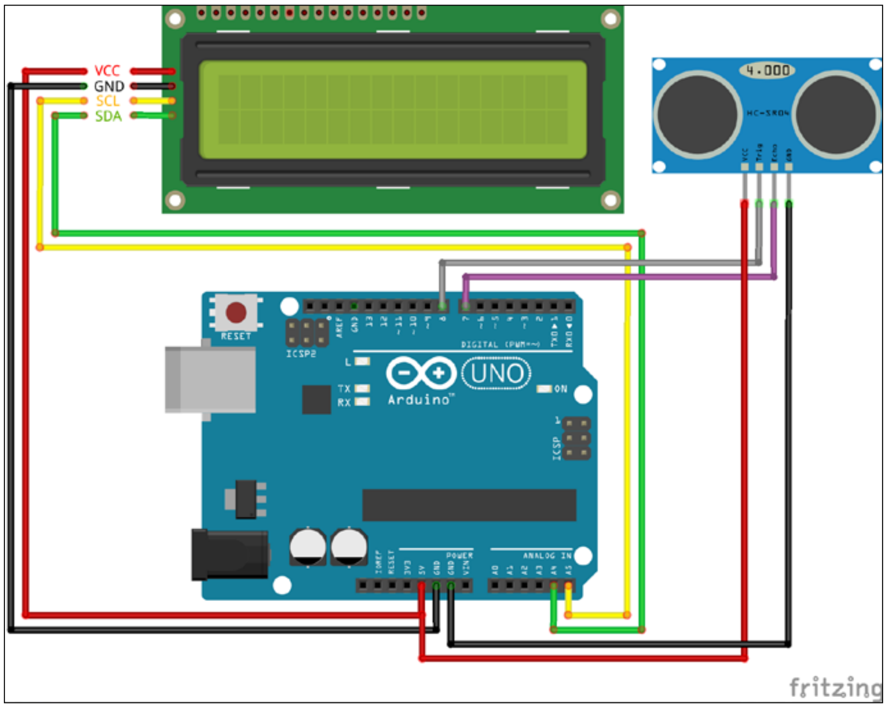
But, what if I want to use the second line too? Well, don't worry. Change the code as follows:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x20,16,2);  // set the LCD address to 0x20 for
a 16 chars and 2 line display

void setup()
{
  lcd.init();                      // initialize the lcd

  // Print a message to the LCD.
  lcd.backlight();
  lcd.print("I like Arduinos!!!!!");
  lcd.setCursor(0,1);
  lcd.print("So Awesome!");
}

void loop()
{
}
```

Best of both worlds

Now comes the part where we combine what we have learnt so far in this chapter into one project. We are going use the sensor to calculate the distance between an object and relay this information to be displayed on the LCD screen in real time.

You are going to combine both the circuits from the previous two sections and create something like this:

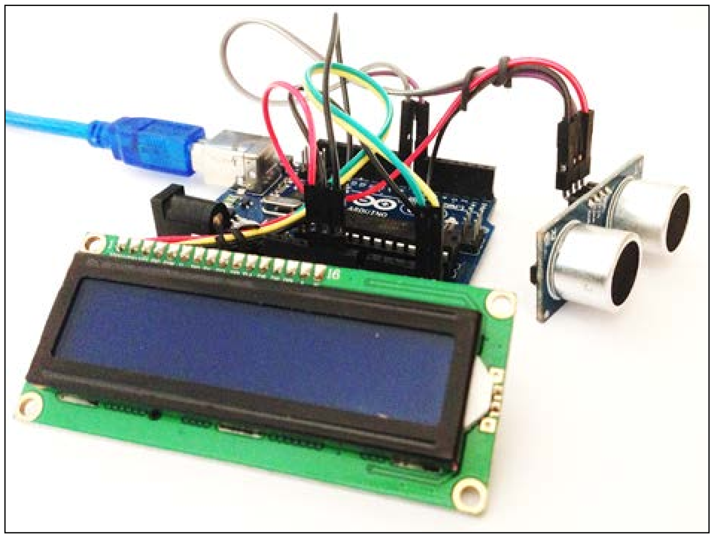


Note that, at the bottom of the setup, you can see two power VCC/5V/red wires merging into one. We did not use a breadboard because we wanted to save space. A simple way to go about this is to use a male splitter. A crude way is to cut a male to male in two and cut one female to female wire in two, strip off a bit of their plastic insulation and twist the copper ends (two males and one female) together and simply tape the joint.

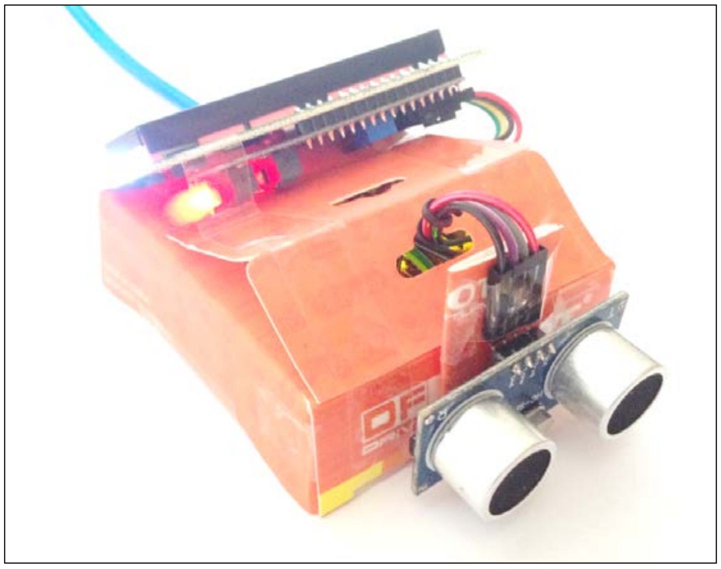
Open up a new sketch and load the `Digital_Ruler.ino` file.

Save it as `Digital_Ruler.ino` in your `Chapter 2` directory and upload the code to the Arduino.

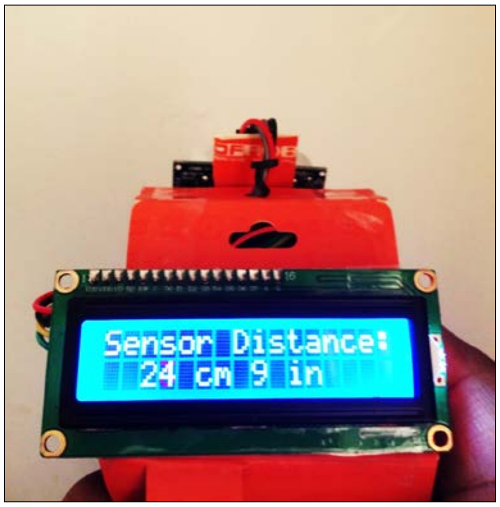
The result, if everything has gone right, will be exactly what you expect. It should look something like this:



Pretty neat, huh? But, if you want to make it compact, you can create something like this:



The LCD view will be like this:



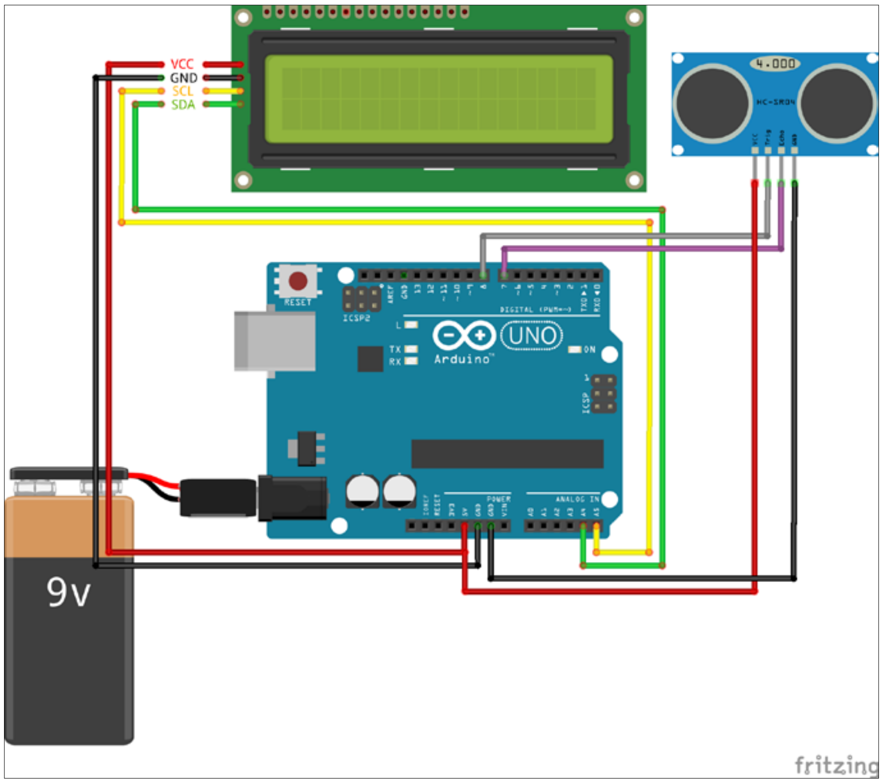
Hold on a second! How did you get there? Well, as much as I would like to give you a step-by-step tutorial for putting the setup into a box, I'd like to take this moment to ask you to put on your creativity cap, scramble around your home/office, and find scraps that you could use to convert this into an art and craft project. You can use anything you can find: a box, tin can (insulated of course), a bottle, anything would do. I merely chose the packaging box that the LCD came in.

But what if I want to make it completely portable? As in, without that annoying USB cable? Well, let me show you what you can do. You simply need to have these two things:

- A 9V Battery:

- A 9V battery connector:

And build the following circuit:



The code is the same. Everything remains the same except that there is a battery to power the board instead of the USB cable. A standard 9V battery connected to a 2.1mm barrel jack can be connected to the Arduino to power it. The Arduino UNO can handle between 5-20V of voltage. But the onboard voltage regulator ensures that no more than 5V is fed to the components connected to it. If you want to, you can also mount a laser pointer on the sensor so that it improves the accuracy of the device. Plus, it would look a lot cooler.

Summary

That was fun, right? And a bit challenging, correct? Good! That's when you know you are learning. So let's just summarize what we achieved in this chapter. We first programmed a HC-SR04 Ultrasound sensor and used it to measure a distance which was then displayed on the Arduino UNO Serial Monitor. Next, we played around with the I2C LCD1602 screen, and then, we combined what we learned from the two sections into one project called the Digital Ruler (Scale). You successfully created a digital measuring tape, which made it compact and less cumbersome to use. But since it can measure between 0 to 2 meters, it can only be used indoors. Higher ranges can be achieved using better (and more expensive) sensors.

In the next chapter, we will learn about touch sensors, which will along with a powerful processing software allow us to convert finger gestures to text.

3 Converting Finger Gestures to Text

You have reached chapter 3. Here we will look deeper into the realm of sensors. You will learn a lot in this chapter about using a touch sensor to create cool projects. You will be introduced to Arduino's sister software (technically its father). Processing, which is often used along with the Arduino to either create an interface to communicate with the Arduino, or simply to display data in a much more comprehensible way.

This chapter is divided into three sections. We will start by learning the basics of Processing. Then we will use the number touch pad and the processing software to create a tic-tac-toe (X's & O's) game. And, in the final section, we will use the touch pad to recognize stroke patterns and different alphabets and display them on the screen.

This chapter uses less hardware; however it involves a lot of programming, so you should be ready for that.

In this chapter, we'll cover the following topics:

- Brief note on capacitive sensors
- Introduction to Processing
- Tic-tac-toe with touch
- Pattern recognition with touch

Prerequisites

You will need the following components to add a touch element to your Arduino projects:

- 1x Arduino-compatible board such as the UNO
- 1x USB Cable A to B
- 1x capacitive touch kit (`http://www.dfrobot.com/index.php?route=product/product&keyword=capac&product_id=463`)

[Only the capacitive number pad and touch pad will be used in this chapter.]

What is a capacitive touch sensor?

A capacitive touch sensor is an upgrade from the commonly used resistive sensors that relied on a change in resistance due to a change in resistor geometry due to the applied pressure. A capacitive sensor works on the principle that your finger acts as an external capacitor that changes the total capacitance of the system, which is then measured by onboard chips and converted into readable data.

An introduction to Processing

Processing is very similar to Arduino in the sense that it has a similar interface to the Arduino IDE. But Processing is used mainly to create graphical interfaces or to display data in real time, which is what we are going to be doing in this chapter. Let's get started.

The first thing you will have to do is to download and install the latest version of Processing from `https://processing.org/download/`.

Install it like you installed Arduino.

Open the application. The environment looks very similar to the Arduino IDE, right? This should make it easy for you to work with.

For a quick demo on what Processing can do, go to **File** | **Example** and, under **Inputs**, choose **MouseXY** (**Mouse2D**). Run the sketch and be amazed! Try some other examples, such as (**Topics** | **Simulate** | **Flocking**), to see what else can be achieved.

Now we are going to use this powerful software along with the Arduino and number touch pad to create a tic-tac-toe game.

Tic-tac-toe with touch

Remember how, while using Arduino, you needed to install libraries to make certain functions work? We need to do the same with Processing because it cannot directly communicate with the Arduino. To do this, go to `http://playground.arduino.cc/Interfacing/Processing` and download the `processing2-arduino.zip` file. Processing, like Arduino also creates a directory by default in the `Documents` folder. Extract the downloaded ZIP file to `C:\Users\<user>\Documents\Processing\libraries` for Windows and `Documents/Processing/libraries` for Mac and Linux.

Do the same for the matrix library we will be using in the next section from `http://pratt.edu/~fbitonti/pmatrix/matrix.zip`. Restart Processing.

> If you do not have a `libraries` folder in the `Processing` directory, go ahead and create a new one.

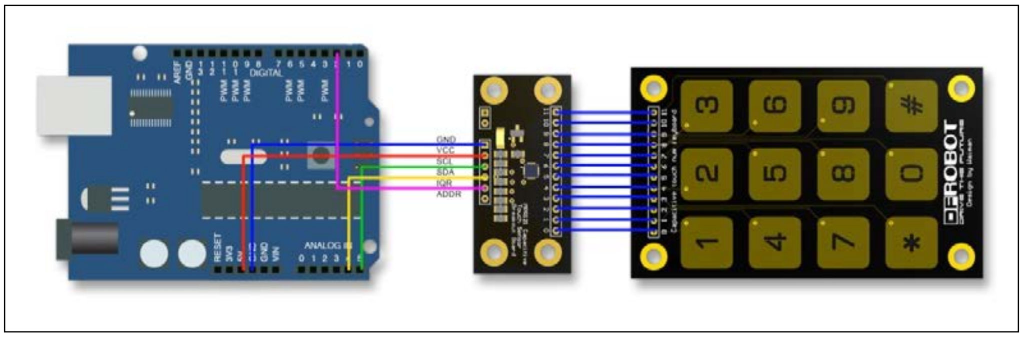
Now launch the Processing IDE (not the Arduino IDE). Connect the Arduino and run this sketch:

```
import processing.serial.*;
import cc.arduino.*;

Arduino arduino;
int ledPin = 13;

void setup()
{
  //println(Arduino.list());
  arduino = new Arduino(this, Arduino.list()[0], 57600);
  arduino.pinMode(ledPin, Arduino.OUTPUT);
}
```

```
void draw()
{
  arduino.digitalWrite(ledPin, Arduino.HIGH);
  delay(1000);
  arduino.digitalWrite(ledPin, Arduino.LOW);
  delay(1000);
}
```

What do you observe? The LED that is connected to pin 13 should begin blinking. This means that you have coded the Arduino successfully with Processing. If it didn't work, try using `Arduino.list()[1]` instead of `Arduino.list()[0]`.

Ok, now we want to test out the capacitive touch pad. Create the following circuit, taken from DFRobot's wiki:



(Credits: Lauren, DFRobot)

The connections are as follows (Arduino UNO | NumPad):

- GND – GND
- VCC – 5V
- SCL – A5 (analog pin 5)
- SDA – A4
- IQR – D2 (digital pin 2)
- ADDR – no connection necessary

This particular touch pad also requires a library. Go to `http://www.dfrobot.com/image/data/DFR0129/MPR121%20v1.0.zip` to download it. As before, extract it to the `Processing libraries` directory.

Now it is time to get to the actual tic-tac-toe program.

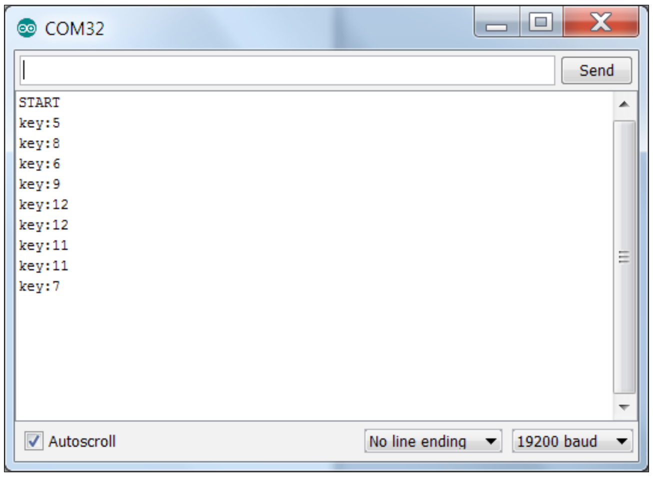
Arduino and Processing

Open up a new sketch on Arduino and paste in the following:

```
#include <Wire.h> // default Wire library
#include <mpr121.h> // touch pad library

int num = 0; // variable to store pressed number

void setup()
{
  Serial.begin(19200); // begin the Serial Port at 19200 baud
  Wire.begin(); // initiate the wire library
  CapaTouch.begin(); // inititate the capacitive touch library
  delay(500);
}

void loop()
{
  num = CapaTouch.keyPad(); // stores the pressed number to num
  if(num > 0){ // checks if the key pressed is within scope
    Serial.print(num); // prints the number to the serial port
  }
  delay(200); // small delay to allow serial communication
}
```

Save it as before, to `tic_tac_toe.ide`.

You must be thinking: Oh! You said there would be a lot of programming in this chapter! This code is so small! Running the program will give you a **Serial Monitor** display like this:



```
START
key:5
key:8
key:6
key:9
key:12
key:12
key:11
key:11
key:7
```

Serial Monitor output

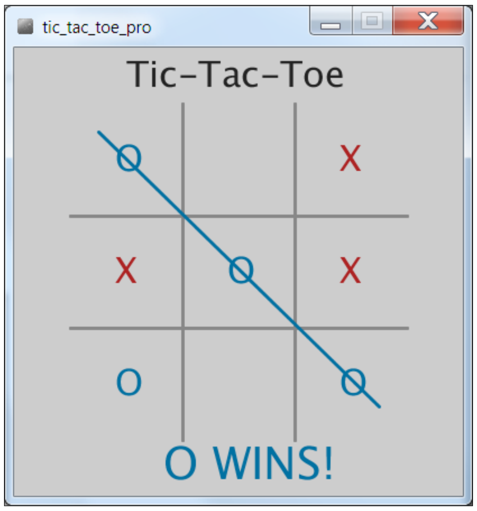
Well, that was just the Arduino sketch. Now we need to add the processing sketch that will allow us to communicate with the Arduino and create a nice little graphical interface where you can play a two-player tic-tac-toe game.

This involves a lot of things, as you will find out.

Using Processing, open the file called `tic_tac_toe_pro.pde` that came with this chapter and, when you're ready, run it.

The result

Go ahead and play a game on your own or with a friend and you should get something like this:



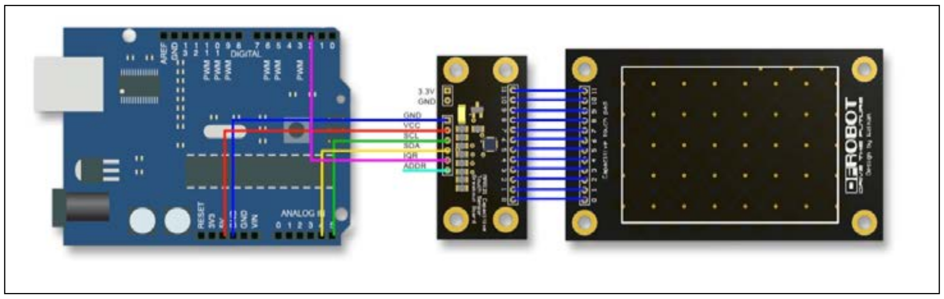
Tic-tac-toe final output

Pretty neat, huh? If you are feeling really adventurous and have complete confidence in your programming skills, you can go ahead and program a player versus AI game, which is outside the scope of this book.

Now that we have had a small taste of what is possible using a touch sensor, we will move on to the pattern recognition part of the chapter where we will push the capacitive grid sensor and our coding capabilities to their limits.

Pattern recognition

Firstly, go ahead and create this circuit:



Touch pad circuit (credits: Lauren, DFRobot)

Your capacitive touch pad and its controller (the central component in the image), are connected using the corresponding numbers labeled on the pins. The connections from the Arduino to the controller are as follows (literally the same connections as before):

- GND – GND
- VCC – 5V
- SCL – A5 (analog pin 5)
- SDA – A4
- IQR – D2 (digital pin 2)
- ADDR – no connection necessary

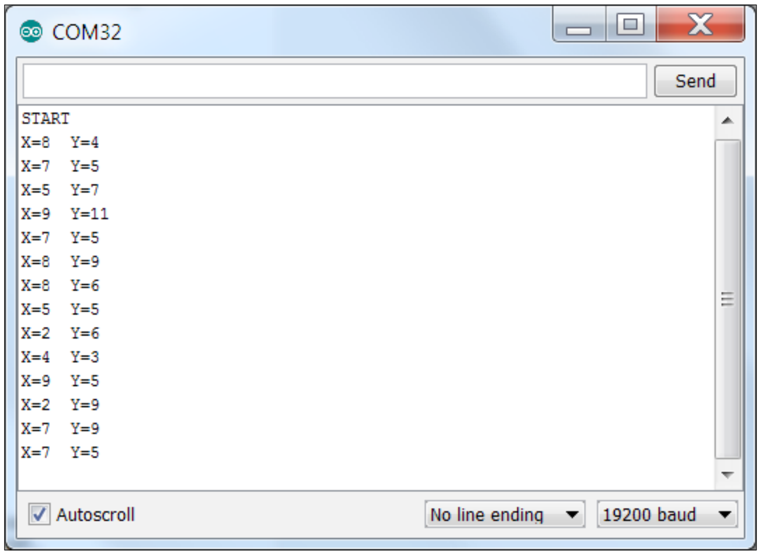
Do not worry if the touch pad doesn't look exactly like this; as long as the connections are fine, everything will work out.

Open up Arduino and go to **File** | **Examples** | **MPR121** | **Examples** | **Touchpad** or copy the following code:

```
#include <Wire.h>
#include <mpr121.h>

int X ;           // X-coordinate
int Y ;           // Y-coordinate

// =========  setup  =========
void setup()
{
  //  initialize function
```

```
    Serial.begin(19200);
    Wire.begin();
    CapaTouch.begin();

    delay(500);
    Serial.println("START");
  }


  // =========  loop  =========
  void loop()
  {
    X=CapaTouch.getX();              // Get X position.
    Y=CapaTouch.getY();              // Get Y position.
    if(X>=1 && X<=9 && Y>=1 && Y<=13)
     {
  // Determine whether in the range. If not, do nothing.
        Serial.print("X=");
        Serial.print(X);
        Serial.print("  Y=");
        Serial.println(Y);
     }
    delay(300);
  }
```

Now, run this program, open up the Serial Monitor, and set it to a baud rate of 19200. Play around with the touch pad using your finger tip and you should get something like this:

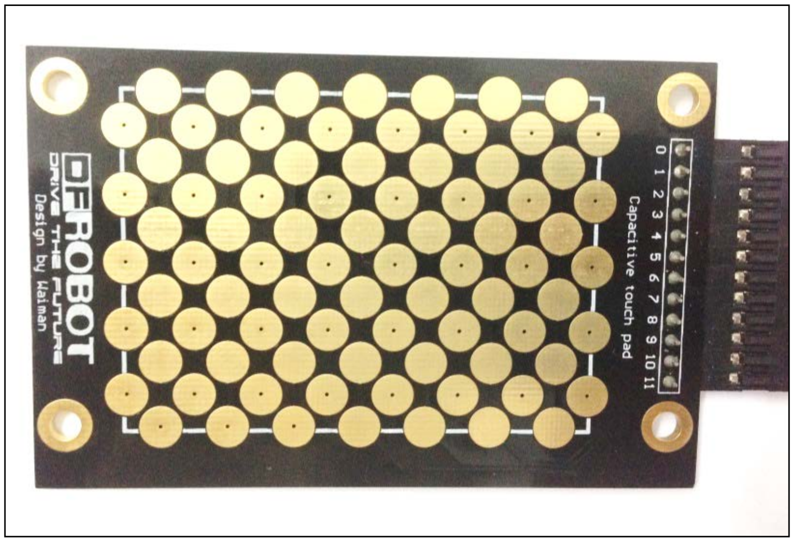


```
START
X=8  Y=4
X=7  Y=5
X=5  Y=7
X=9  Y=11
X=7  Y=5
X=8  Y=9
X=8  Y=6
X=5  Y=5
X=2  Y=6
X=4  Y=3
X=9  Y=5
X=2  Y=9
X=7  Y=9
X=7  Y=5
```

Serial Monitor with touch pad

Before we jump into how we are going to use this device, let's take a minute to think about what exactly we are trying to do using the device. You can use it for various things but, for the purpose of this project, we are going to use the touch pad, Arduino, and Processing to convert your finger strokes into meaningful text (alphabets for simplicity).

Now, before we link this to Processing, we need to learn how to tackle the problem that has haunted engineers for decades, namely, image processing. In this case, however, the image processing is not too complex, but some sort of logical brainstorming is required. Look at your capacitive touch pad.

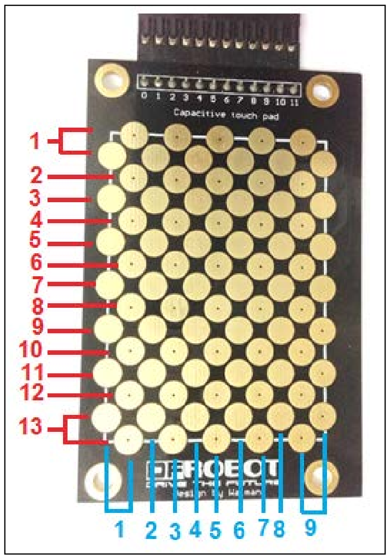


Capacitive touch pad

What is the first thing that you think of that would aid in analyzing the stroke patterns? Right away, you would guess, a two-dimensional array! You are right.

A 2D array or matrix like the one that we used in the X's & O's section will help us to tackle this problem. If you have already played with the touch pad, and if yours looks similar to the one used in this chapter, the first two rows both have an X value of 1, the last two have a value of 13, the two left columns have a Y value of 1 and the two right-most columns have a Y value of 9.

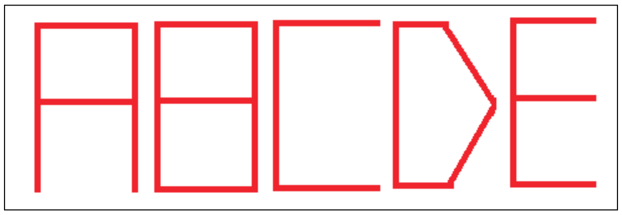
Let's try to put them to good use. This is what the layout of the touch pad would look like with coordinates added in for visualization.



Capacitive touch pad - Labeled

We are going to create a 13x9 matrix with a starting value of 0 for each element. When the capacitive touch sensor pad is touched, the corresponding value in the matrix will be changed to 1. In this way, we have a mathematical means of representing the stroke layout.

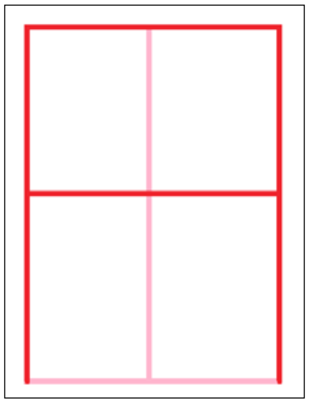
To further alleviate the complexity of the problem of having to understand and iterate all of the twists and bends of alphabets, we will be using block diagrams.



Block diagram for ABCDE

What is wrong with that D? It is enough to know that, if the D was written as a complete rectangle, it would simply represent O. These typefaces are much easier to process as they are mainly composed of lines.

Let's look at the block letter A.

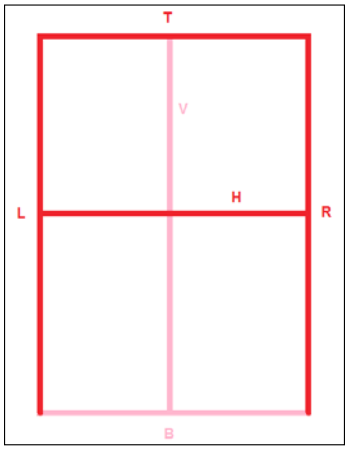


Block A

Let's name each of the straight lines:

- Top Horizontal – T
- Bottom Horizontal – B
- Middle Horizontal – H
- Left Vertical – L
- Right Vertical - R
- Middle Vertical - V

With the labels, it looks like this:

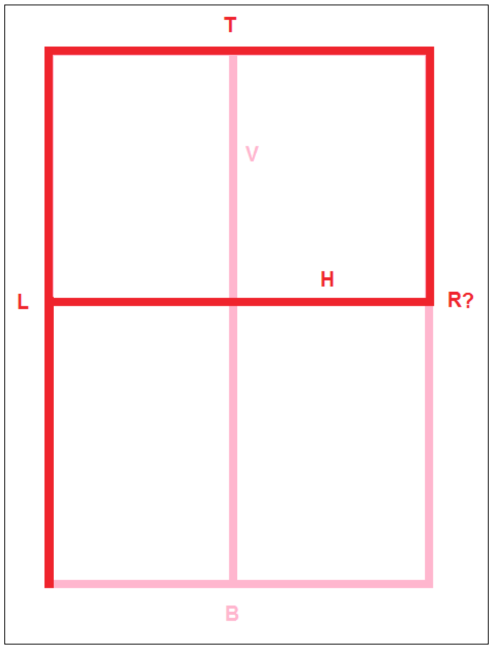


Block A with labels

Note how the Bottom Horizontal line and the Vertical Line are faded; this is because the alphabet A does not contain these strokes. We can calculate if a particular combination of T, B, H, L, R, and V exists for each alphabet in the matrix we made before. If we denote true (exists) as 1, then for A:

- T = 1
- B = 0 (does not exist)
- H = 1
- L = 1
- R = 1
- V = 0

You see where we are going with this, right? In this way we can map out the characteristics of all the alphabets. Some alphabets, though, are a bit tricky. For example, the letter P looks like this:



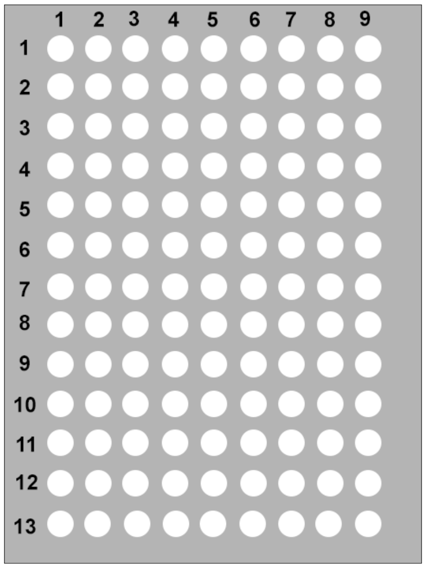
Block P

What do we do for the Right Vertical (R) line? To solve this issue, if we come across a half-line, we will denote it with a 2. A ½ would be more representative but, when using ½ as a coding parameter, it will simply disappear into a zero in the integer format, hence 2 is used.

So P would become:

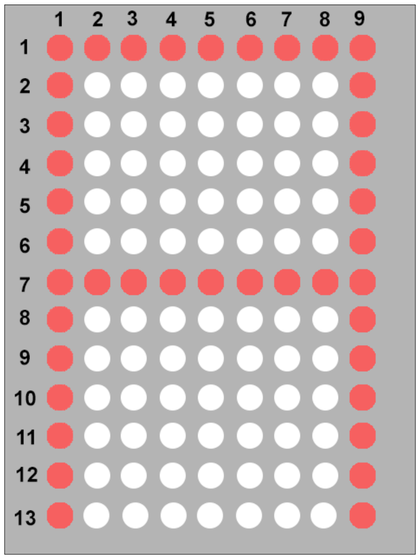
- T = 1
- B = 0
- H = 1
- L = 1
- R = 2
- V = 0

Now, to represent the touch panel in real time or to process a display to represent it requires the ability to use the power of matrices. So we will code a simple layout that looks like this:



Processing grid layout

And, with the strokes on an ideal block A, it would look something like this:



Processing grid layout for 'A'

Now that we have understood how to solve this entire problem, let's finally go to the code.

Fire up Arduino and paste in the following code. Or you could just use the `touch_pad.ide` file supplied to you with this book.

```
// Connections:
// SDA (MPR121) -> PIN A4 (Arduino)
// SCL (MPR121) -> PIN A5 (Arduino)
// IRQ (MPR121) -> PIN A2 (Arduino)

// Libraries
#include <Wire.h>
#include <mpr121.h> // Touch Pad library

int X ;  // X-coordinate
int Y ;  // Y-coordinate
int inByte = 0; // incoming serial byte

void setup()
{
  Serial.begin(57600); // Begin serial at 57600 baud rate (faster)
  Wire.begin(); // intiialize wire library
  CapaTouch.begin(); // initialize the capacitive touch library
  delay(500); // brief delay for initialization
}

void loop()
{
  X = CapaTouch.getX(); // Get X position.
  Y = CapaTouch.getY(); // Get Y position.

  // Determine whether in the range.If not,do nothing.
  if(X>=1&&X<=9&&Y>=1&&Y<=13)
  {
    // Debug lines, can be uncommented to check inputs
    //Serial.print("X=");
    //Serial.write(X);
    //Serial.print("  Y=");
    //Serial.write(Y);

    // convert X and Y coordinates into one variable
    if(Y<10)
    {
      Serial.print(X*10+Y); // prints to serial port
    }
    else if(Y>9)
    {
      Serial.print(X*100+Y); // prints to serial port
    }
   }
     delay(200); // delay for message to be relayed
}
```

Save it as `touch_pad` under `Chapter 3`.

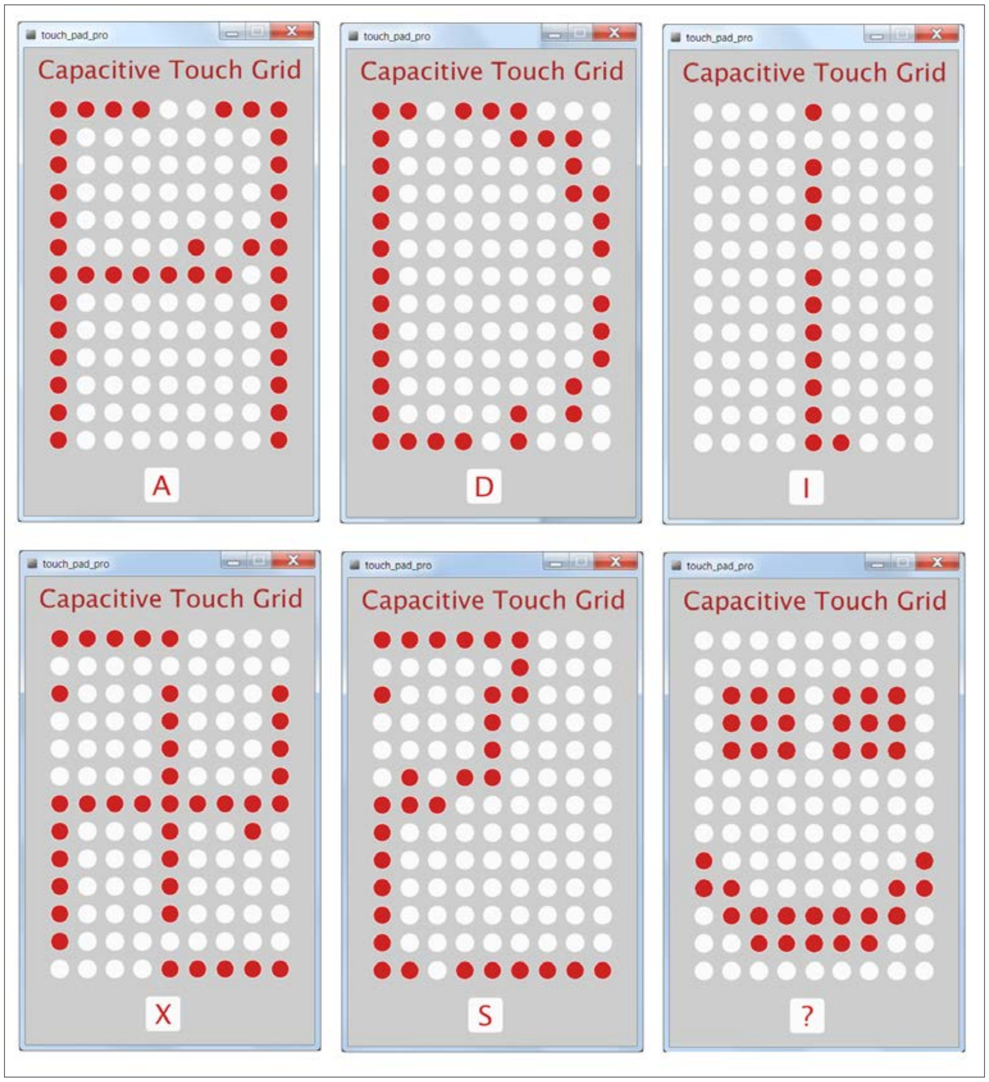
Note how we converted both the X and Y coordinates into one integer so that we send a single data point through the serial port.

Open a new processing sketch and open the file named `touch_pad_pro.pde`. Run it.

Save it as `touch_pad_pro`. Upload the Arduino code first and only then run this program! If you get an error with the libraries, make sure they are installed in your `Processing` directory in `Documents`.

Go ahead and play with it. I recommend the first character to be an 'I', nice and simple. Make sure the strokes are gentle. You can go over previous lines, so do not worry about that. You will soon find out that the strokes don't have to be perfect for the program to identify what they are.

Your results should look identical or similar to these:



Touch pad results

With this knowledge you can do almost anything with that touch device. Think maze game, digital locking mechanism, and the like.

Summary

That was super cool, wasn't it? The synergy when Arduino, Processing, and touch sensors work together is enormous. We first played with the basics of Processing and understood how it communicates with Arduino. We wrote a simple Processing code that could directly control the Arduino. Then we created a simple tic-tac-toe program using the capacitive number pad as the input device, and using Processing as the processor to make the game tick. Finally, we created a complicated pattern-processing program that takes input from the capacitive touch pad, which we realized is really powerful. You have learnt a lot from this chapter and I hope this serves you well in your coding endeavors.

In the next chapter, we bid farewell to Processing and say hello to an even more powerful tool that synergizes well with Arduino, namely, Python. Additionally, in the next chapter, we will learn how to implement wireless communication in our projects by means of Bluetooth.

4
Burglar Alarm – Part 1

In the previous chapter, there were a lot of coding skills to take on board. Understandably, it was difficult, but you made it through. In this chapter, we will use a lot of hardware and software. Haven't you always wanted to make your own burglar alarm? Catch the crook who stole your favorite cookies from the cookie jar? This chapter will teach you how to do just that.

Firstly, we will create a plan of how we are going to go about catching the culprit. Of course, it is not enough to simply sound an alarm when a thief is caught in the act; you also want to have evidence. You see where this is going, right? Yes, we will be using a wireless camera to get a mugshot of the culprit.

This chapter does use a fair share of code but, in addition to that, it requires quite a few components. Learning to use them in unison is the ultimate goal of this chapter. It is divided into the following sections:

- The PIR sensor
- Testing the camera
- Communicating with a smart phone
- The burglar alarm

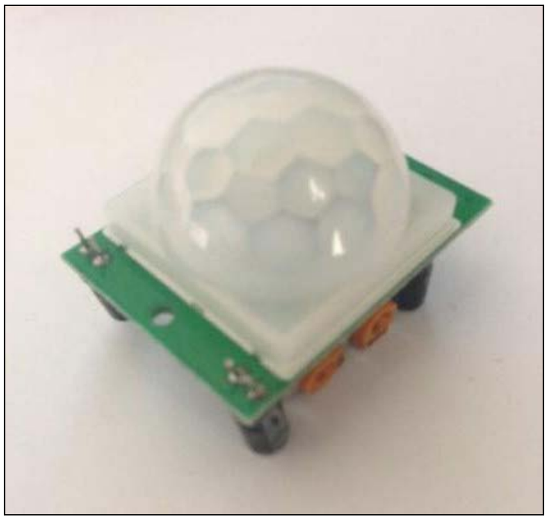
I promised you in previous chapters that I would try to teach you as much I possibly could about the Arduino. Bluetooth is very reliable and inexpensive but it is short-range and usually needs a host to gather or send data.

The following are the components you'll need, to create a high-tech burglar alarm:

- 1 x Arduino UNO board
- 1 x USB cable A to B (aka the printer cable)
- 1 x PIR sensor
- 1 x wireless IP camera (a netcam360 is used in this chapter)
- 1 x HC-06 module (Bluetooth)
- 1 x wireless router (with accessible settings)
- 1 x PC with inbuilt Bluetooth or a Bluetooth USB module

What is a passive infrared sensor?

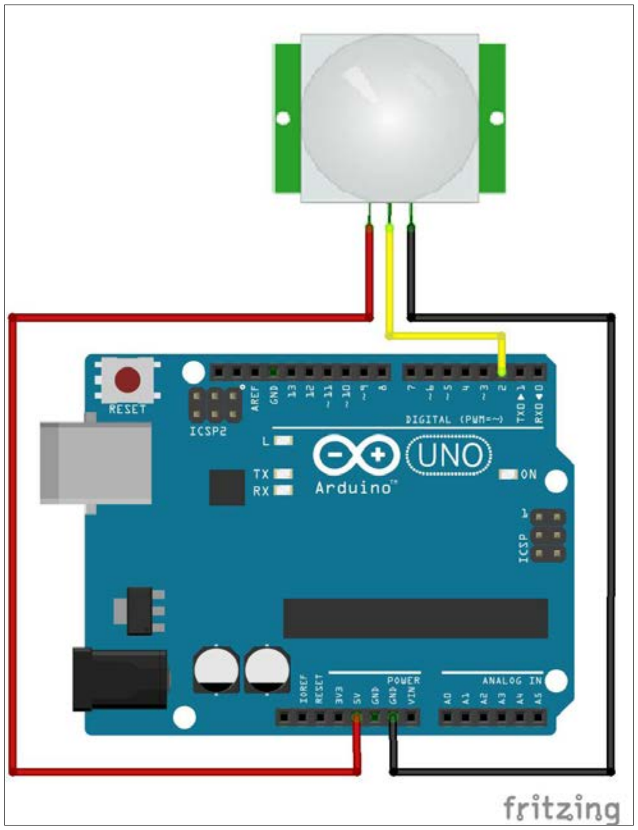
A **passive infrared sensor** (**PIR**) is an electronic sensor that uses infrared radiation to detect variations in its field of view. They are most commonly used as motion sensors; for example, they are used to minimize power consumption by switching off lights and utilities if nobody is at home. They are also used in state-of-the-art burglar alarm systems to trigger a switch when motion is detected.



[If you would like to learn more about how they work, you should refer to this page (`https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/`) at Adafruit. Adafruit and Ladyada are really good resources for building Arduino projects.]

A mini PIR-Arduino alarm

Let's get started. We are going to create a setup in which an LED flashes when motion is detected by the PIR sensor. This is what the setup should look like when connecting the Arduino to the PIR Sensor:



Basically, the connections are as follows:

- GND → GND
- VCC → 5V
- OUT → D02 (digital pin 2)

[Digital pins are denoted using D and analog pins are denoted by A.
So digital pin 13 is D13 and analog pin 2 is A02.]

Open Arduino and load the sketch called `PIR_LED.ino`, or copy this:

```
int ledPin = 13; // use the onboard LED
int pirPin = 2; // 'out' of PIR connected to digital pin 2
int pirState = LOW; // start the state of the PIR to be low (no
motion)
int pirValue = 0; // variable to store change in PIR value

void setup() {
  pinMode(ledPin, OUTPUT); // declare the LED as output
  pinMode(pirPin, INPUT);  // declare the PIR as input
  Serial.begin(9600); // begin the Serial port at baud 9600
}

void loop() {
  pirValue = digitalRead(pirPin); // read PIR value
  if ((pirValue == HIGH)&&(pirState==LOW)) { // check if motion has
occured
    digitalWrite(ledPin, HIGH); // turn on LED
    Serial.println("Motion detected!");
    pirState = HIGH; // set the PIR state to ON/HIGH
    delay(1000); // wait for a second
  }
  else { // if there is no motion
    digitalWrite(ledPin, LOW); // turn off LED
    if(pirState == HIGH) {
    // prints only if motion has happened in the first place
    Serial.println("No more motion!\n");
    pirState = LOW; // sets the PIR state to OFF/LOW
    delay(200); // small delay before proceeding
    }
  }
}
```

Run the code and open up the **Serial Monitor** and set the baud rate to 9600.
This is a simple program that switches on the LED when motion is detected and powers it off when there is no motion.

See it in action by moving your hand in front of the PIR censor. See how the LED glows when the PIR detects motion? PIR sensors are very sensitive to variations in light, which is why they are often used in motion detectors.

This was pretty straightforward so far, right? Good! Let's now move on to the camera.

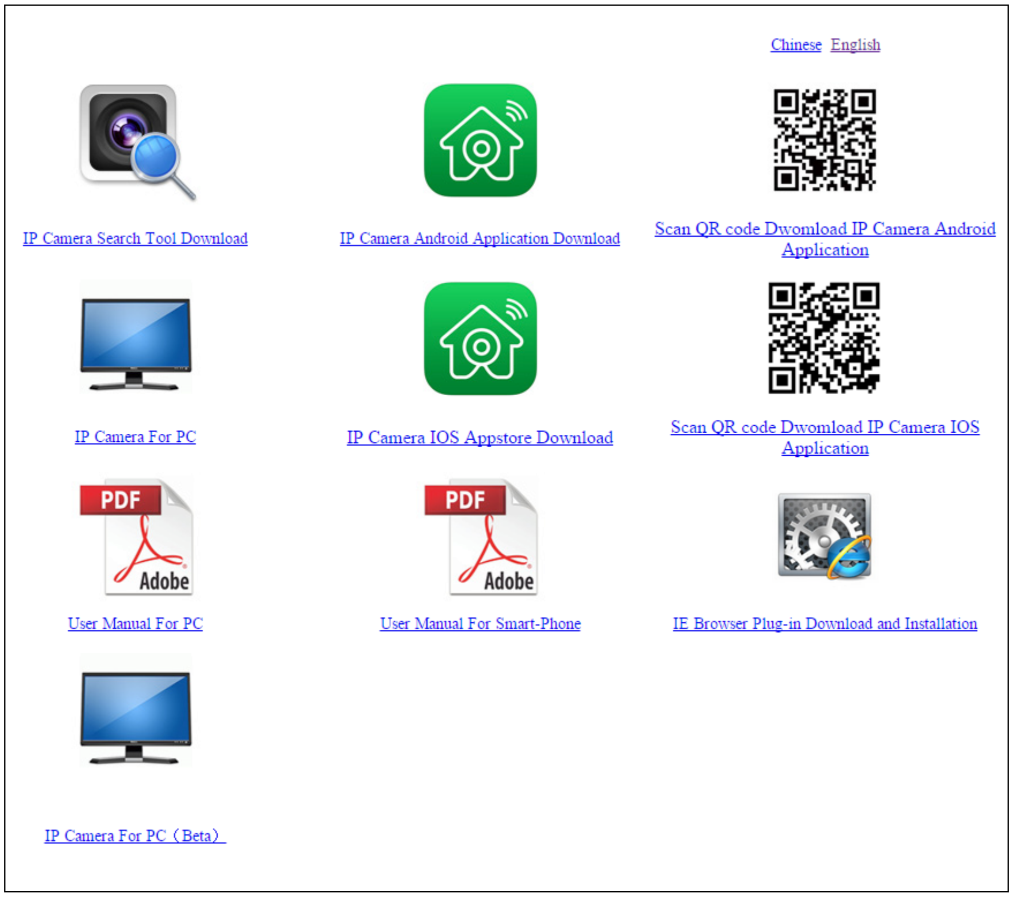
Testing the camera

An IP camera (or Internet Protocol camera) is a camera that you can access on your wireless network provided that it is configured correctly. The configuration procedure depends on which IP camera you bought and who the manufacturer is, but they should all be pretty similar.

If you purchased the exact same one that is used in this chapter, the setup is explained in the following section. However, if you got a different one, do not worry. Use the manual or installation guide that comes with the camera to install it onto your network. Go through the following steps anyway, to get an idea of what you should do if you do not have the same camera.

Installing the camera on the network

Go to `http://netcam360.com/enindex.html` and download **IP camera for PC**.



Chinese English

IP Camera Search Tool Download

IP Camera Android Application Download

Scan QR code Dwomload IP Camera Android Application

IP Camera For PC

IP Camera IOS Appstore Download

Scan QR code Dwomload IP Camera IOS Application

User Manual For PC

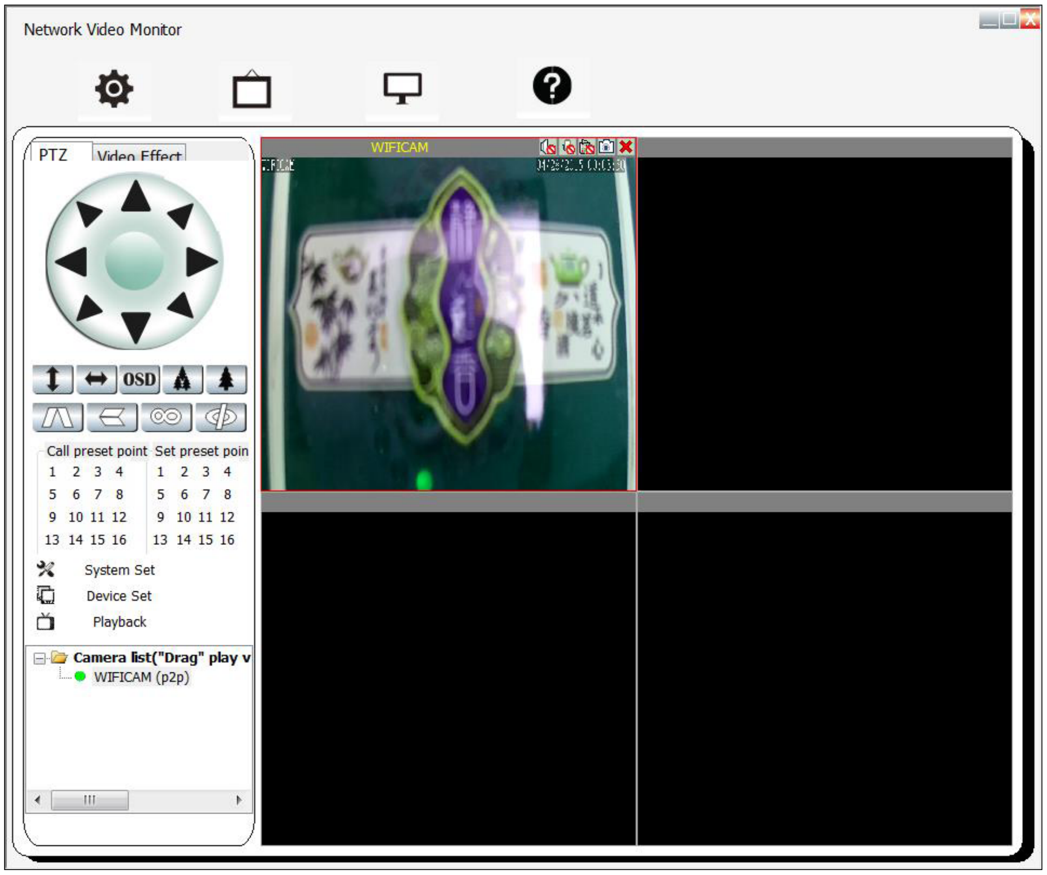
User Manual For Smart-Phone

IE Browser Plug-in Download and Installation

IP Camera For PC（Beta）

You should also be able to find a PC installation manual on the same webpage. Follow the instructions provided. I will not go into detail because there is a high chance that you have not purchased the same IP Camera.

Eventually, you should be able to get something like this:

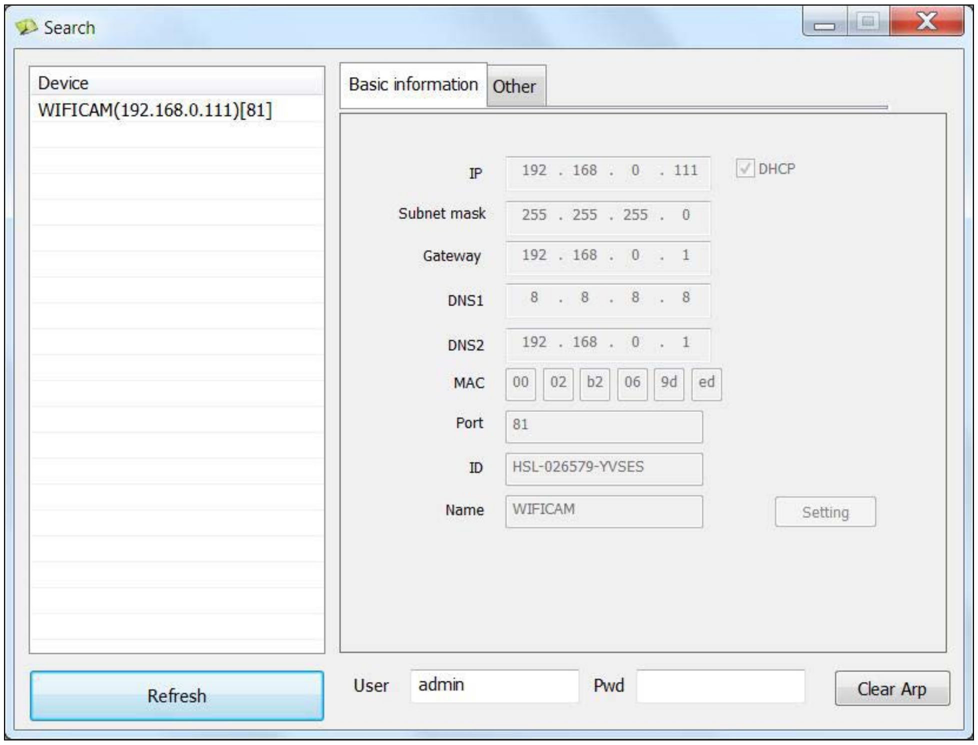


It is highly recommended that you create a password for your camera to keep prying eyes away. Also, do remember the password because we will need it in the steps to come.

Setting up the mugshot URL

Go back to `http://netcam360.com/enindex.html` and download the IP camera search tool or use the file of the same name located in the `Useful Files` folder that comes with this chapter. I believe that this tool can be used, irrespective of the manufacturer, to find IP cameras on your network.

Run it and you should see this screen:



IP 192 . 168 . 0 . 111
Subnet mask 255 . 255 . 255 . 0
Gateway 192 . 168 . 0 . 1
DNS1 8 . 8 . 8 . 8
DNS2 192 . 168 . 0 . 1
MAC 00 02 b2 06 9d ed
Port 81
ID HSL-026579-YVSES
Name WIFICAM
User admin
Pwd

Note down the IP address at the top. In this case, it is 192.168.0.111:81, where 81 is the port number.

Open a new browser window and, in the address bar, paste the following:

```
http://<your ip>/snapshot.cgi?user=admin&pwd=<your_password>
```

For example:

```
http://192.168.0.111:81/snapshot.cgi?user=admin&pwd=password
```

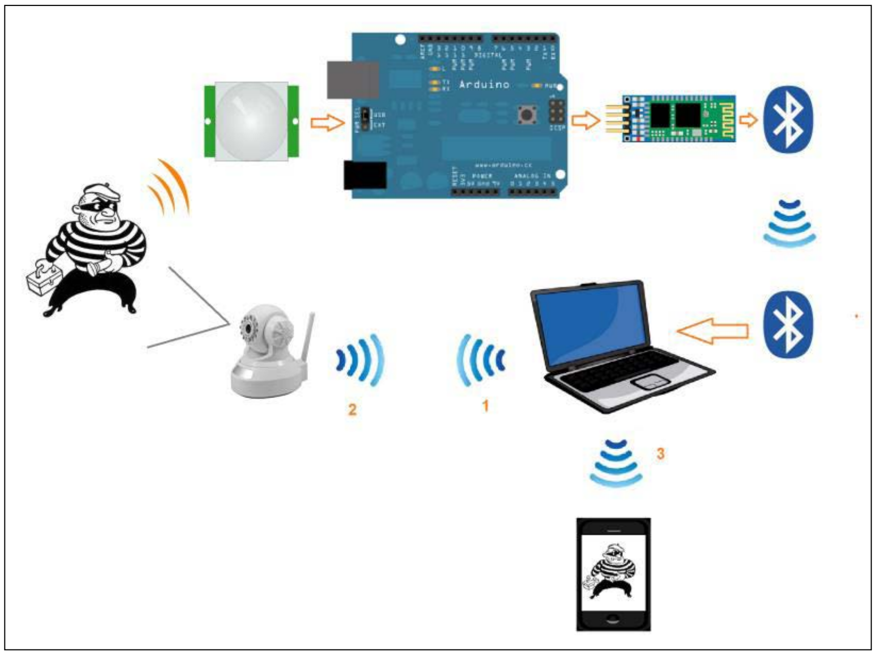
You should be able to see a snapshot of what your camera is seeing in real time. You can try refreshing the page while moving the camera to test different camera positions. We are going to use this link later in the chapter to fetch the image file to catch the burglar.

Putting it together

As I mentioned before, we will be using a Bluetooth module to communicate with the Arduino. The specific model is the HC-06 module, which is one of the cheapest Arduino communications modules and is widely used. We could just try to hook it up to our smart phone and send an alert directly to it, but what if you aren't at home? That is a drawback of using Bluetooth.

But fret not. We will capitalize on what we can do using just the Bluetooth. I think this is a good time to explain the plan before diving into the details.

The following is a representation of what we are going to achieve in this part of the chapter.



We use the PIR sensor to check if there is any motion detected. This data is transmitted through the HC-06 Bluetooth module to a host computer through a Bluetooth channel. A script (program) running on the computer takes a snapshot of the culprit using the wireless IP camera's URL and saves it as an image. It then uploads the image to an image-sharing website and fetches the link. Finally, an alert is sent to the user saying that the alarm has been set off. The user can then open the link on a smart phone to see the culprit. I hope that wasn't too hard to follow.

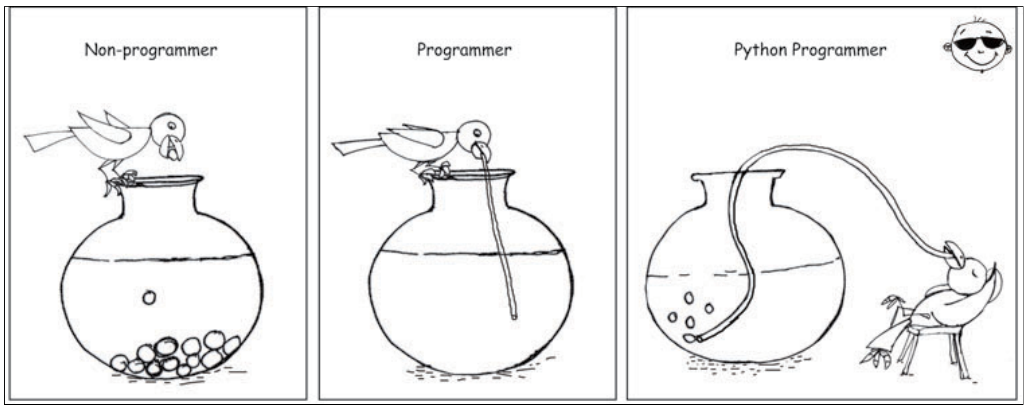
First, let's get the script ready.

An introduction to Python

Wait! What? I did not sign up for this! Hold on there. We are not going to study Python too much. That is outside the scope of this book. To be honest, I think this particular section is the furthest you will deviate from the field of Arduino. But you must understand that, to create something really powerful, you need to make the most out of the resources we have. We are just going to use its basic functionalities to achieve our intended plan.

What are you even talking about? What is Python? Why are we talking about a snake? Python is a very powerful, but easy to use, language like C or Java. We will be using it to get the snapshot from the camera, upload it to the Web, and send a notification to your smart phone.

The following image is the best way to describe it:



You should be warned that this section is going to be a bit difficult, but you should try to be patient. Once Python is installed, it is going to stay there forever. Let's go ahead and install it. There are two popular versions of Python, namely 2.7 and 3.4. We are going to use 2.7 since it is older and has a lot more libraries that work with it. Yes, libraries, similar to the ones that you used earlier with the Arduino.

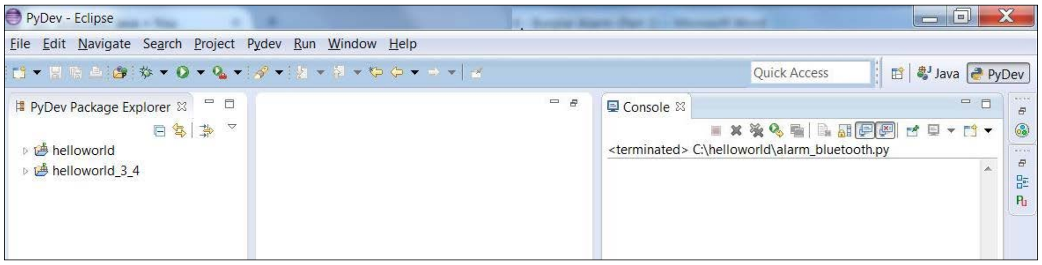
Download Python from `https://www.python.org/download/releases/2.7.8/` according to your operating system. Note that 32-bit is x86 and 64-bit is x64.

Install it to somewhere convenient. Most OS X and Linux computers come with Python pre-installed.

Next we need to download and install an interface for writing your codes. I recommend Eclipse because it is easy to use for newcomers to Python. Since Eclipse is Java-based, you should update Java by going to `http://www.java.com/en/` and installing/updating Java on your system.

You can download Eclipse from `http://www.eclipse.org/downloads/`. Select Eclipse Classic and install it, or rather extract it, into a convenient location, as with Arudino. If you have used Python before, you can simply choose your own interface. Geany is one commonly used Python IDE.

Open Eclipse. If you get a Java error, use this resource to solve the problem: `http://stackoverflow.com/questions/2030434/eclipse-no-java-jre-jdk-no-virtual-machine`. It will ask you for a workspace. Use something like `C:\MyScripts` or choose anything similar. Then close the welcome screen. You will see something like this:

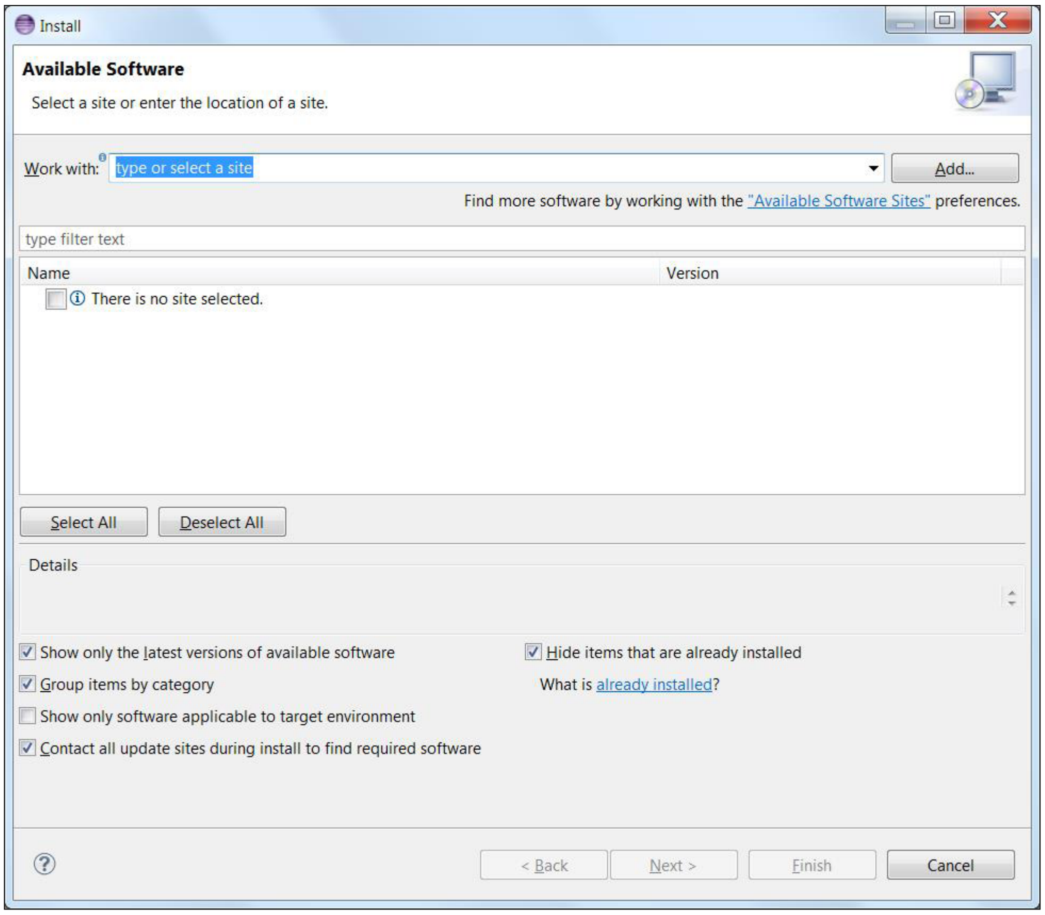


<terminated> C:\helloworld\alarm_bluetooth.py

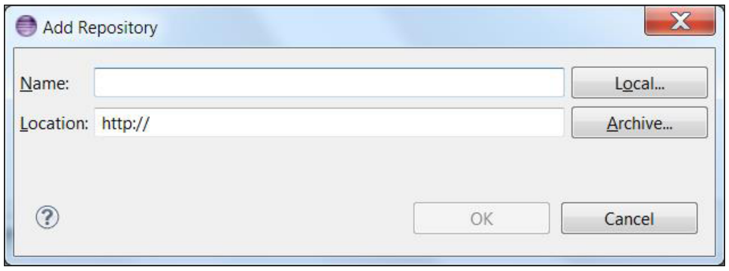
Ignore **PyDev Package Explorer** on the left in the screenshot. I am starting from the beginning so that you can set it up.

1. Go to **Help | Install New Software**.

   You will see this dialog box:


Install
Available Software
Select a site or enter the location of a site.
Work with: type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

2. Click **Add...**.

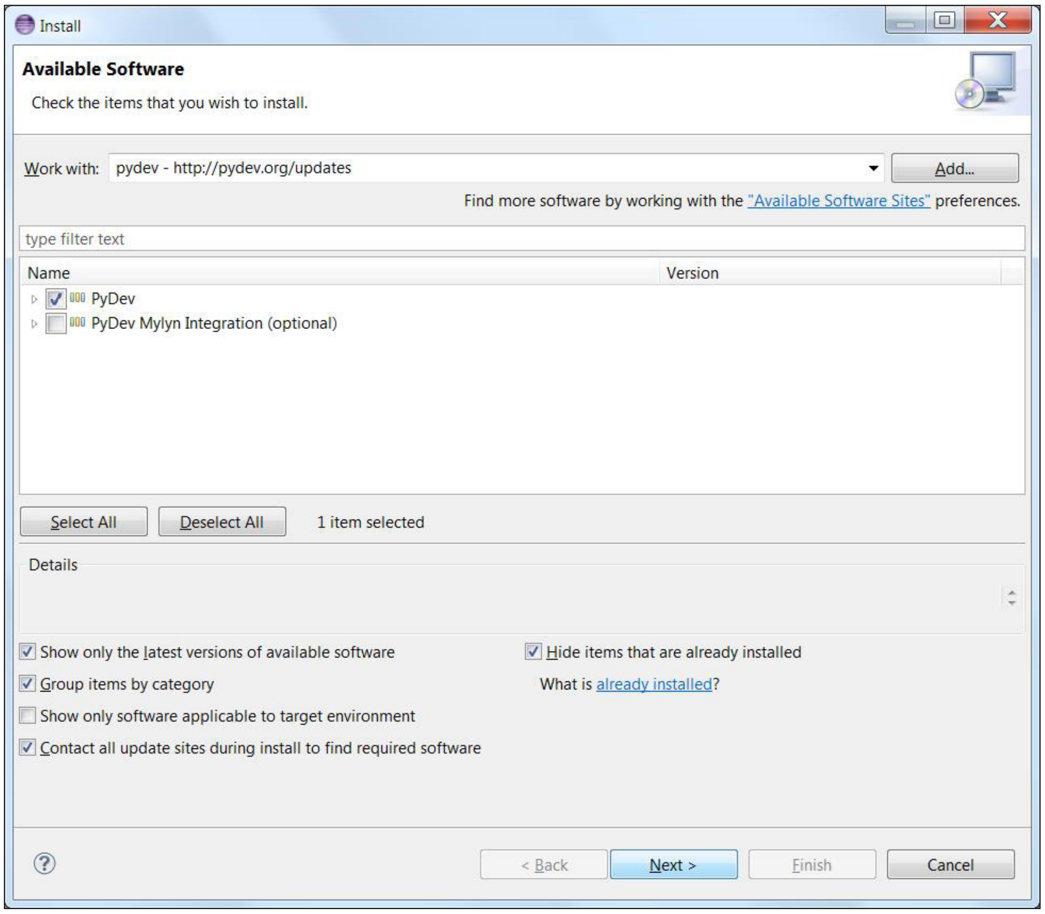


Add Repository

Name:

Location: http://

3. For **Name**, type `pydev` and for **Location,** type `http://pydev.org/updates`.
4. Press **OK**. Wait for it to load.



Install

Available Software

Check the items that you wish to install.

Work with: pydev - http://pydev.org/updates

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☑ PyDev	
☐ PyDev Mylyn Integration (optional)	

1 item selected

Details

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed

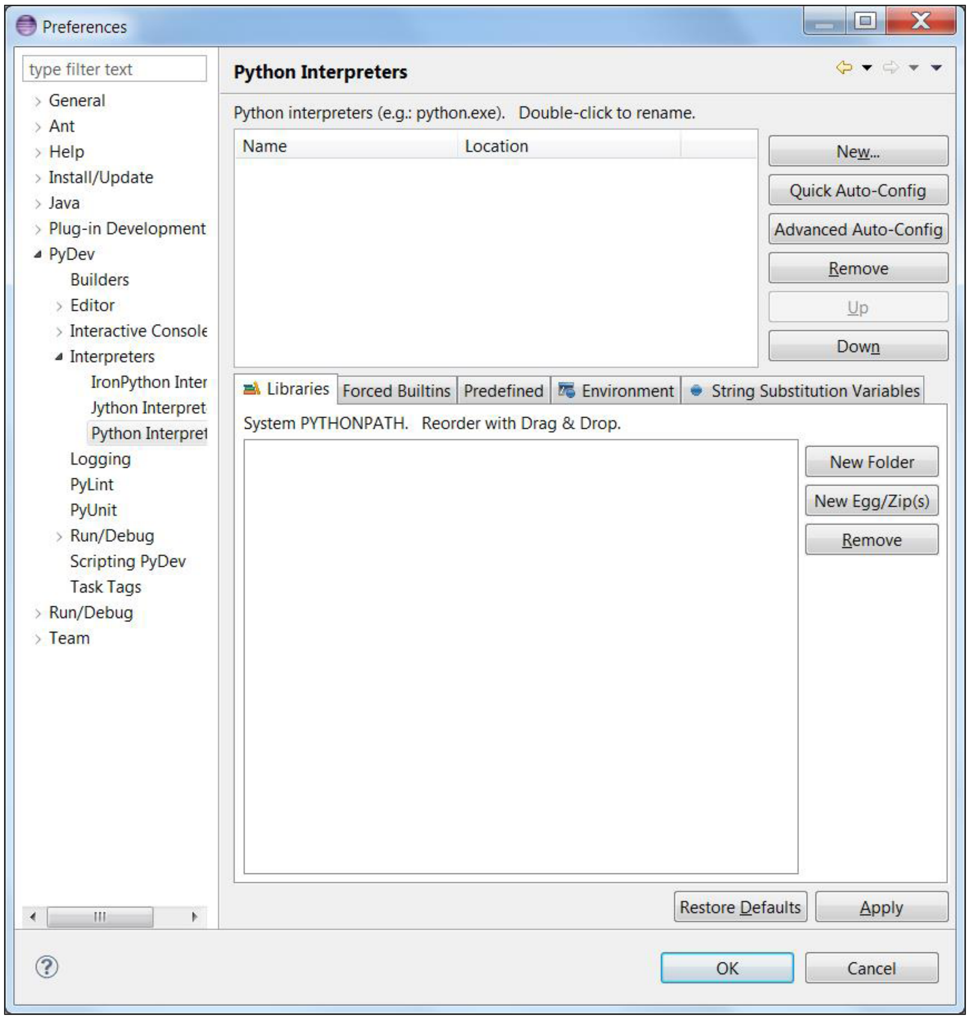
What is already installed?

5. Select the first option (**PyDev**) and click **Next >**. Accept the terms and conditions and let it install. If it asks you whether you trust this application, just select **Yes**.

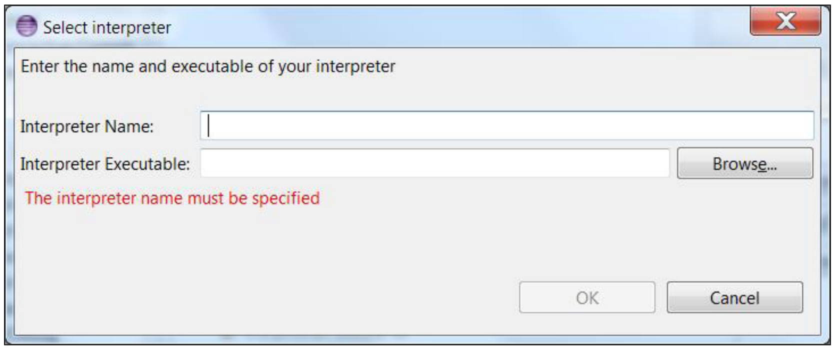
   It will ask you to restart Eclipse. Allow it. Wait for it to launch by itself. You're almost done.

6. Go to **Window** | **Preferences**.

   In that dialog, expand **PyDev** | **Interpreters** | **Python Interpreter**.



7. Click on **New...**.



Select interpreter

Enter the name and executable of your interpreter

Interpreter Name:

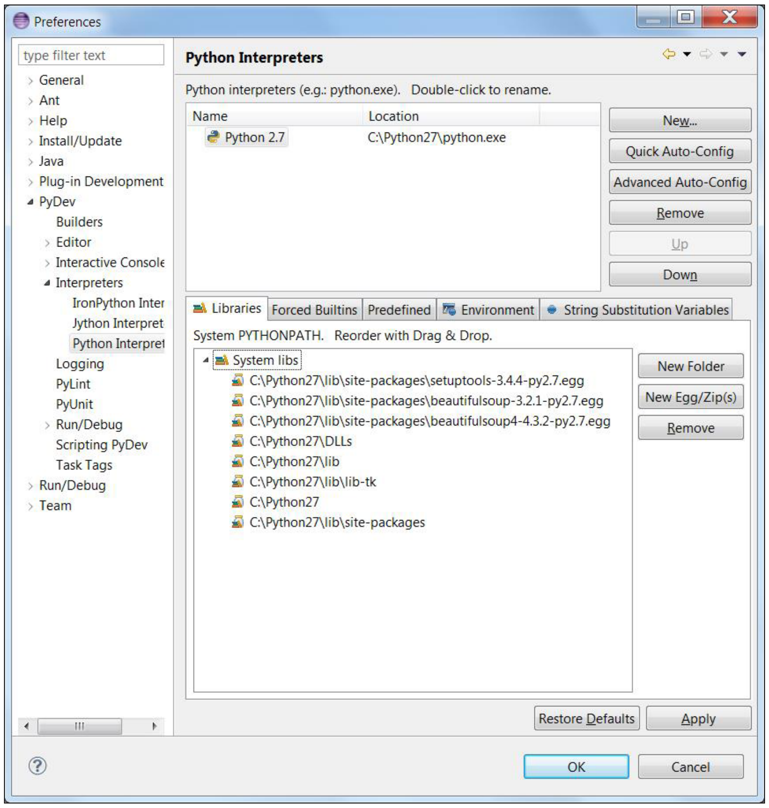
Interpreter Executable:

The interpreter name must be specified

8. For the **Interpreter Name**, type `Python27`.

   And for **Interpreter Executable**, you can browse to select `C:\Python27\python.exe` or you can simply paste that without quotations. Click **OK**.

   It will load a lot of files. Ensure all are checked and hit **OK**.



Preferences

Python Interpreters

Python interpreters (e.g.: python.exe). Double-click to rename.

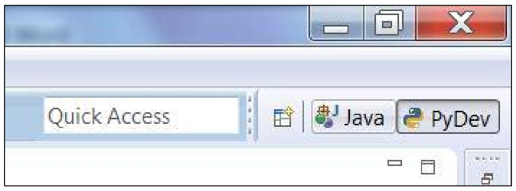
Name	Location
Python 2.7	C:\Python27\python.exe

System PYTHONPATH. Reorder with Drag & Drop.

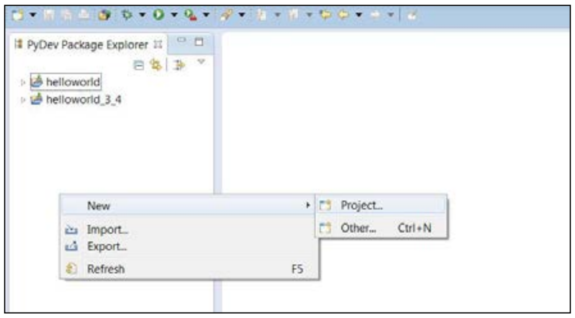
System libs
- C:\Python27\lib\site-packages\setuptools-3.4.4-py2.7.egg
- C:\Python27\lib\site-packages\beautifulsoup-3.2.1-py2.7.egg
- C:\Python27\lib\site-packages\beautifulsoup4-4.3.2-py2.7.egg
- C:\Python27\DLLs
- C:\Python27\lib
- C:\Python27\lib\lib-tk
- C:\Python27
- C:\Python27\lib\site-packages

9. Hit **OK** again.

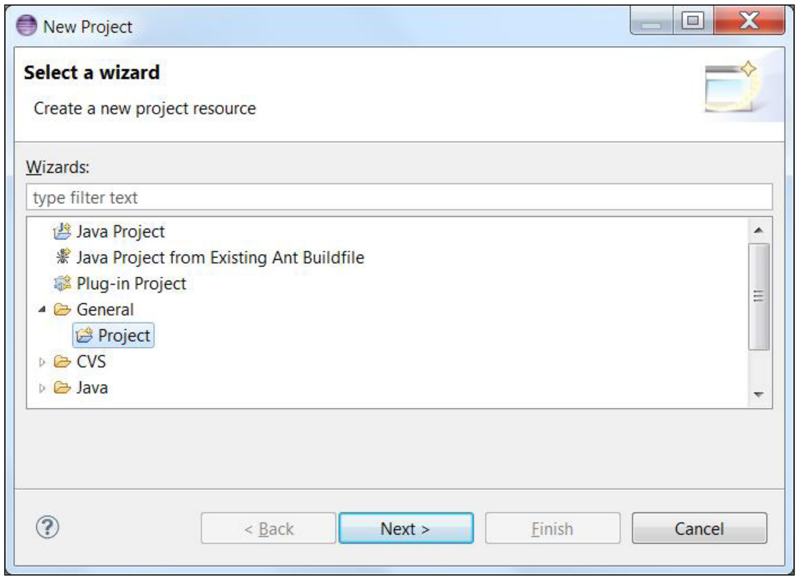
   One last thing: on the Eclipse Interface on the top right. Select **PyDev** instead of Java.



10. Now, in **PyDev Package Explorer**, right-click and create a new project.

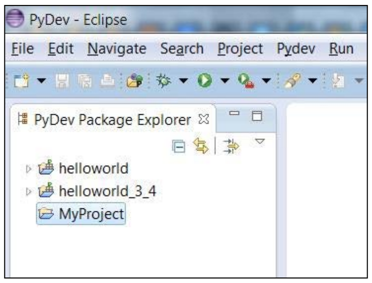


11. Choose **General** | **Project**.

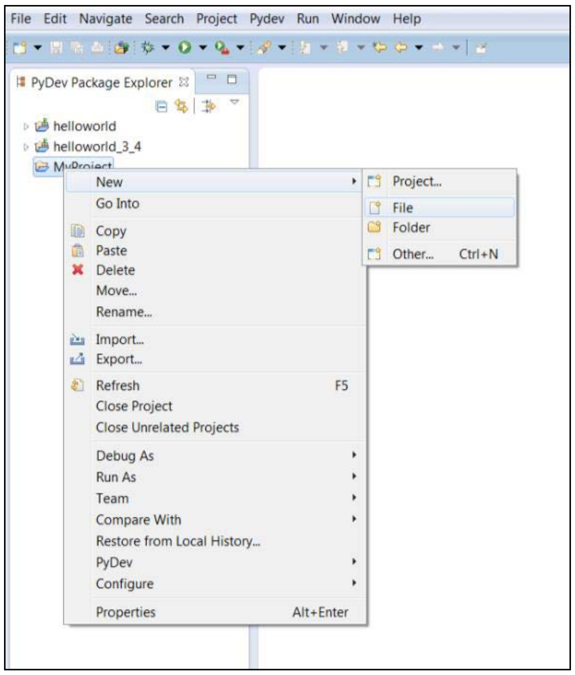


12. Click **Next >**.

    Call your project something like **MyProject** or **helloworld**, or whatever you like, and let it use the default location to store the files. Click **Finish** when you're ready.



13. Right-click on **MyProject** | **New** | **File**.

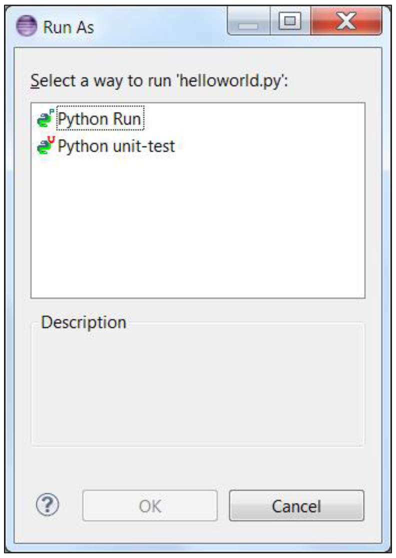


14. Call it `helloworld.py` and press **Finish**.

    Remember to use `.py` and not just `helloworld`, so that Eclipse understands what to associate it with. Type:

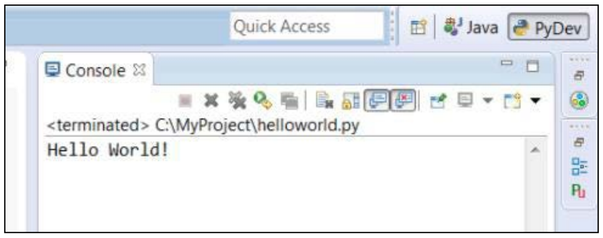
```
print("Hello World!")
```

15. Press **Run** (the green bubble with a right-pointing triangle). It will bring up the following window every time you run a new script:



16. Simply select **Python Run** and press **OK**.

    If everything works as expected, you will see this in the **Console** window on the right side:



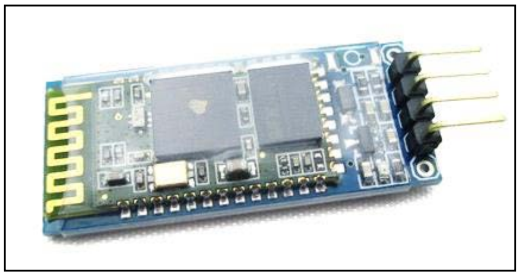
```
<terminated> C:\MyProject\helloworld.py
Hello World!
```

Hurray! You have just written your very first Python script! You should be proud of yourself, because you have just been introduced to one of the most powerful programming tools in the history of programming. When you are done with this chapter, I recommend you look up some Python tutorials to see what it can do.

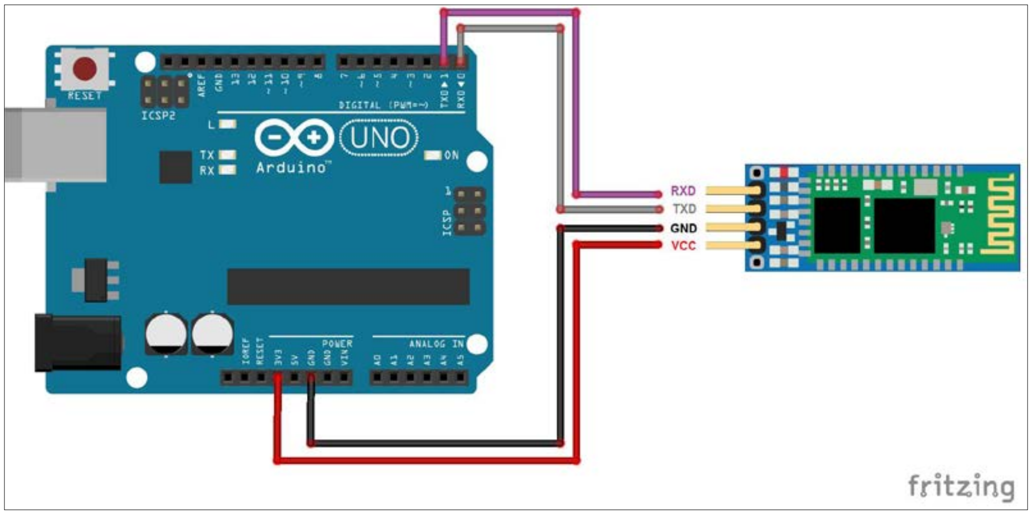
But now, we must keep moving. Let's now focus on setting up the Bluetooth network.

Hooking up the Bluetooth module

The HC-06 is a simple, low-powered Bluetooth module that works very well with Arduinos.



Go ahead and create this circuit:

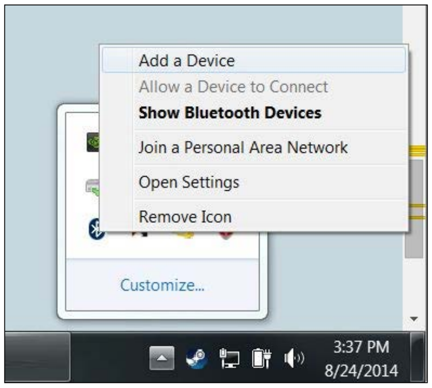


The connections are as follows:

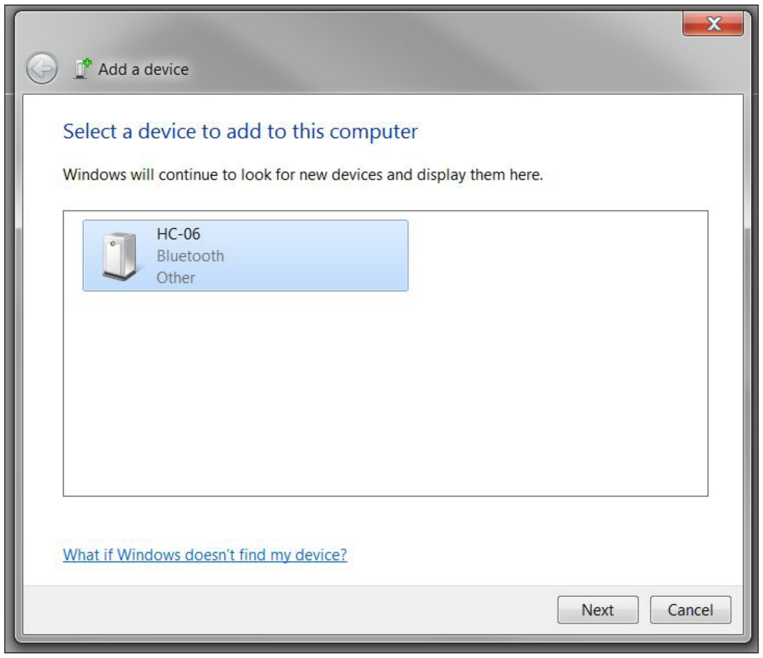
- GND → GND
- VCC → 3.3 V
- RXD → D01 (TX)
- TXD → D00 (RX)

Note that, before uploading codes where RX and TX pins are used, unplug those pins. Reconnect them once the uploading process is complete.

1. Plug the Arduino into the USB hub to power the HC-06 chip. Now, in your system tray, right-click on the Bluetooth icon and click **Add a Device**.



2. Let the computer search until it finds **HC-06**.


Add a device

Select a device to add to this computer

Windows will continue to look for new devices and display them here.

HC-06
Bluetooth
Other

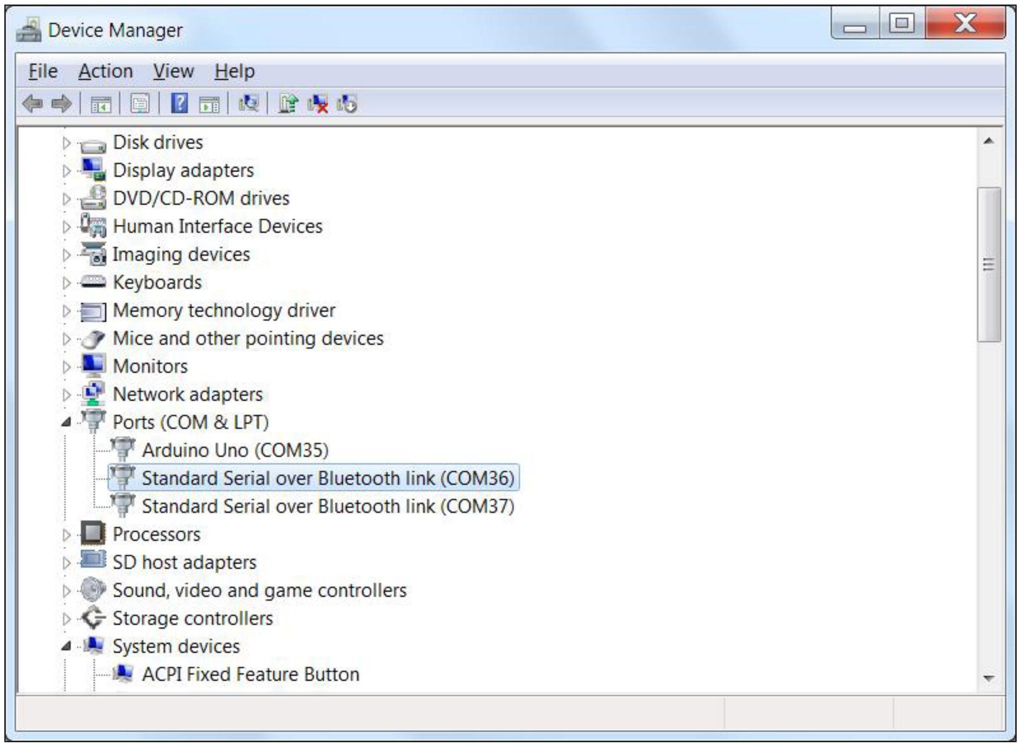
What if Windows doesn't find my device?

If nothing shows up, try using an Android phone to connect to it. If it doesn't show up even then, check your connections.

3. Click **Next**.

   Here, type in the device's pairing code, which is 1234 by default.

4. It will now install the HC-06 on your computer. If everything works well, when you open up **Device Manager** and go to **Ports (COM & LPT)**, you should see this screen:



```
Disk drives
Display adapters
DVD/CD-ROM drives
Human Interface Devices
Imaging devices
Keyboards
Memory technology driver
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
    Arduino Uno (COM35)
    Standard Serial over Bluetooth link (COM36)
    Standard Serial over Bluetooth link (COM37)
Processors
SD host adapters
Sound, video and game controllers
Storage controllers
System devices
    ACPI Fixed Feature Button
```

   Note down these three COM values (they will be different for different users).

5. Finally, you are ready to program the Bluetooth module.

Open up a new Arduino sketch and load the `ard_BL_led.ino` file or paste in the following code:

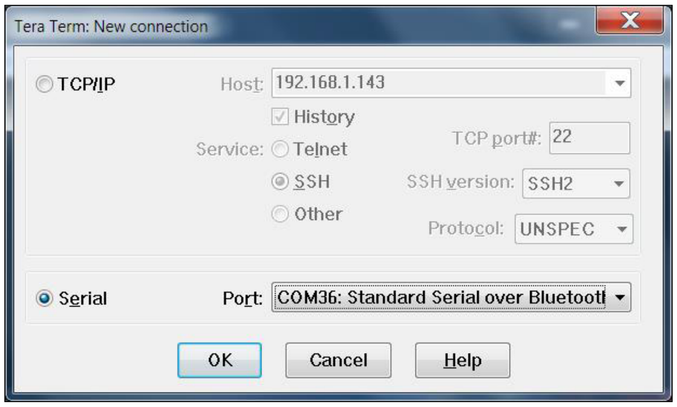
```
char bluetoothVal;            //value sent over via bluetooth
char lastValue;               //stores last state of device (on/off)

int ledPin = 13;
```

```
void setup() {
 Serial.begin(9600); // begin communication on baud 9600
 pinMode(ledPin, OUTPUT); // set the led pin to output
}


void loop() {
  if(Serial.available()) // searches for available data
  {//if there is data being recieved
    bluetoothVal=Serial.read(); //read it
  }
  if (bluetoothVal=='1')
  {//if value from bluetooth serial is '1'
    digitalWrite(ledPin,HIGH);     // turn on LED
    if (lastValue!='1')
      Serial.println(F("LED ON")); //print LED is on
    lastValue = bluetoothVal;
  }
  else if (bluetoothVal=='0')
  {//if value from bluetooth serial is '0'
    digitalWrite(ledPin,LOW);       //turn off LED
    if (lastValue!='0')
      Serial.println(F("LED OFF")); //print LED is off
    lastValue = bluetoothVal;
  }
  delay(1000);
}
```

Again, before uploading, make sure you disconnect the RX and TX wires. Connect them after the upload is completed.

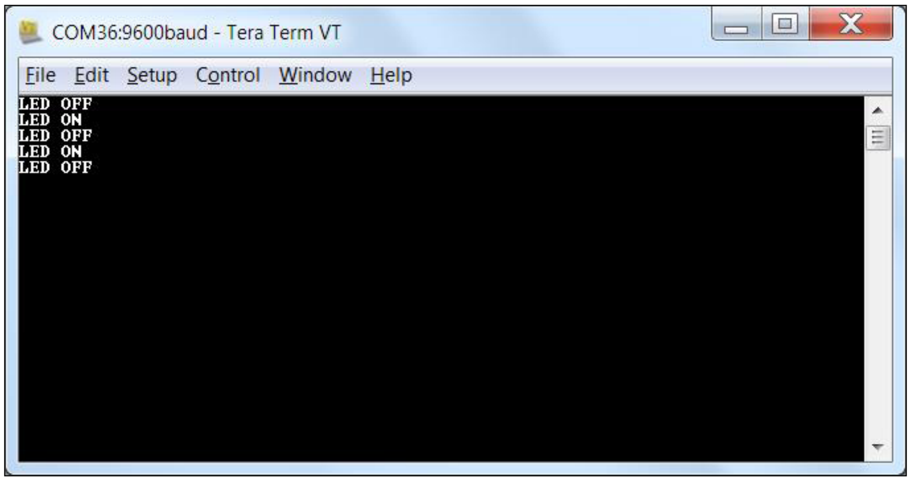
To test this code, we will use some popular software called Tera Term. OS X and Linux systems come with terminal emulators so this is not necessary. It is mainly used as a terminal emulator (a fake terminal, in plain language) to communicate with different devices/servers/ports. You can download it from `http://en.osdn.jp/projects/ttssh2/releases/`. Install it to someplace convenient.

Launch it and select **Serial**, and select the COM port that is associated with Bluetooth. Start by using the lower COM port number. If that doesn't work, the other one should.



Hit **OK**. Give it some time to connect. The title of the terminal window will change to **COM36:9600baud** if everything works correctly.

Type `1` and hit enter. What do you see? Now try `0`.


COM36:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
LED OFF
LED ON
LED OFF
LED ON
LED OFF
```

Give Tera Term some time to connect to Bluetooth. 1 or 0 are not displayed when you type them. Just the LED status will be displayed.

You have now successfully controlled an LED via Bluetooth! This effect would be a lot cooler if you used a battery to power the Arduino so that there was no wired connection between the Arduino and the computer. Anyway, let's not get carried away, there is much to be done.

Before bringing everything together, there are two things left to be done: dealing with the image (mugshot) upload and sending a notification to your smart device. We'll start with the former, in the following chapter.

Summary

No, we are not done with this project. The second half of it is moved to the next chapter. But let's do a quick recap of what we have done so far. We tested out the PIR sensor, which we found out to be a really efficient motion sensor. We installed and wrote our very first Python script, which is a phenomenal achievement. Finally, we used Bluetooth to communicate between the computer and the Arduino.

In the next part of this project, we are going to process the image captured from the camera, upload it to where it can be accessible on other devices, learn about notification software, and finally bring the pieces together to create the burglar alarm.

5
Burglar Alarm – Part 2

This is part 2 (and the final part) of the burglar alarm series. So far, we have configured the camera, the Bluetooth, and Python.

In this chapter, we will be going through the following topics:

- Obtaining and processing the image of the intruder
- Uploading the image to a convenient website
- Sending the URL to your smart phone

So, shall we get right to it?

Dealing with the image

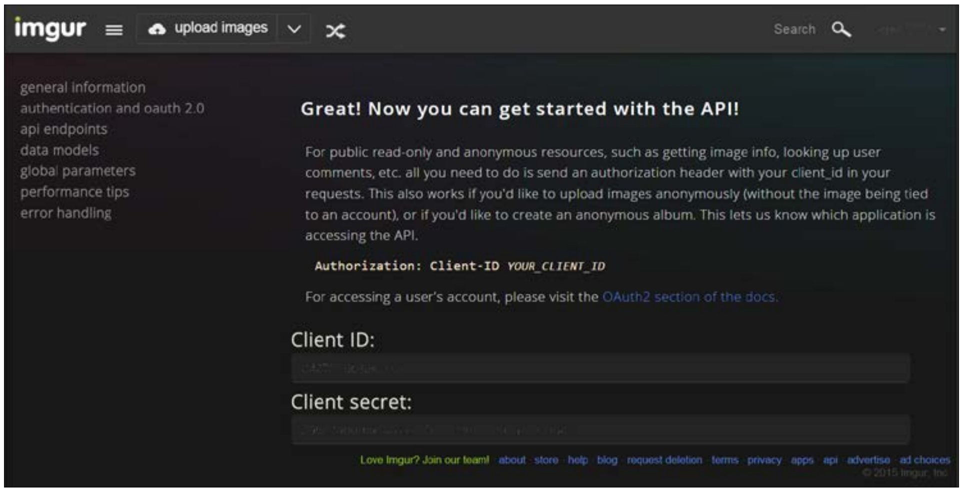
As discussed before, when Arduino sends a message to Python, it is going to take a snapshot using the camera and save it to the computer. What do we do with it, then? How do we upload it to a file sharing platform? There are several ways to do this, but in this chapter, we will be using **Imgur**. Yes, the same Imgur that you have been using, knowingly or unknowingly, on Reddit or 9gag.

Go to `http://imgur.com/` and sign up for a new account.

Once you have verified your account, go to `https://api.imgur.com/oauth2/addclient` to add a new application so that Imgur permits you to post images using a script. Use the following information:

- **Application name**: Arduino burglar alarm
- **Authorization type**: OAuth 2 authorization without callback URL
- **Email**: <your email>
- **Description**: Using Arduino to catch the cookie thieves

Now proceed, and you will get a page like this:



Great! Now you can get started with the API!

For public read-only and anonymous resources, such as getting image info, looking up user comments, etc. all you need to do is send an authorization header with your client_id in your requests. This also works if you'd like to upload images anonymously (without the image being tied to an account), or if you'd like to create an anonymous album. This lets us know which application is accessing the API.

```
Authorization: Client-ID YOUR_CLIENT_ID
```

For accessing a user's account, please visit the OAuth2 section of the docs.

Client ID:
[redacted]

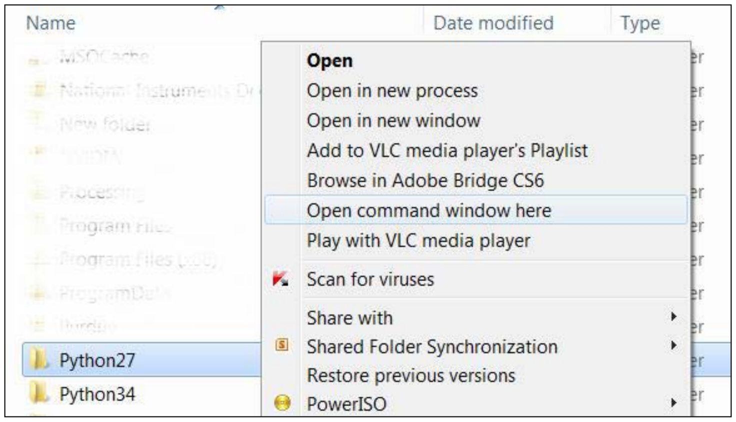
Client secret:
[redacted]

Love Imgur? Join our team! about store help blog request deletion terms privacy apps api advertise ad choices
© 2015 Imgur, Inc

Imgur will give you the **Client ID** and **Client secret**. Save them in a safe location as we will need them later.

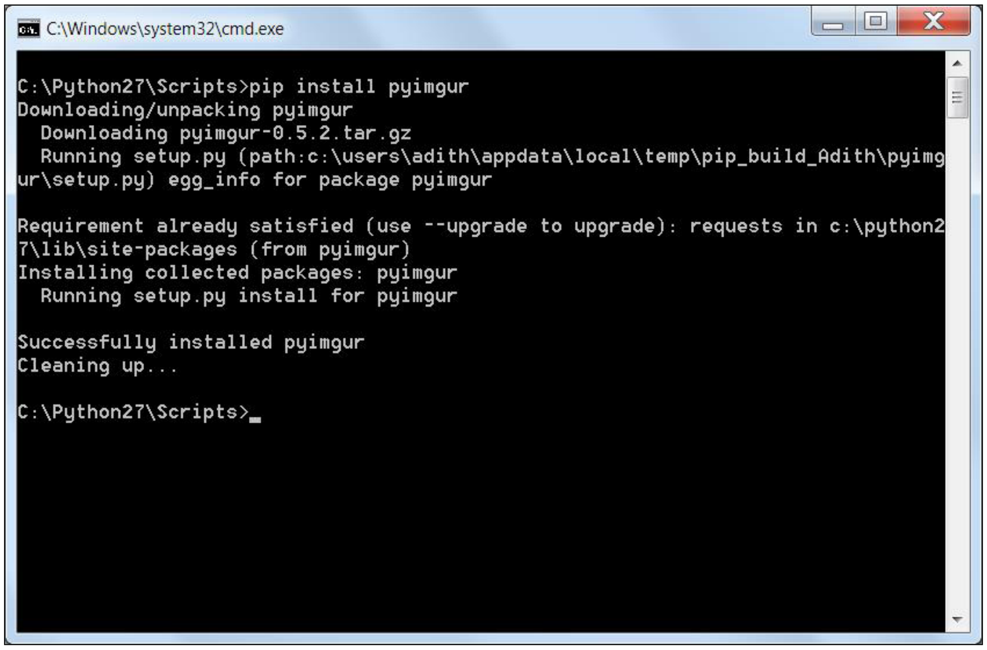
Let us try testing the automatic image uploading functionality. However, before that, we need to install the Imgur library for Python. To do this, firstly, we need to install a simple Python library installer called `pip`, which is used to download other libraries. Yeah! Welcome to Python!

Go to `https://pip.pypa.io/en/latest/installing.html` and download `get-pip.py`, or you could just use `get-pip.py` that is in the `Useful Files` folder that came with this chapter. Save it or copy it to `C:\Python27`. Go to your `C` directory and *Shift* + right-click on `Python27`. Then, select **Open command window here**:


Name
Date modified
Type
Python27
Python34
Open
Open in new process
Open in new window
Add to VLC media player's Playlist
Browse in Adobe Bridge CS6
Open command window here
Play with VLC media player
Scan for viruses
Share with
Shared Folder Synchronization
Restore previous versions
PowerISO

Type `python get-pip.py`, hit *Enter*, and let it install pip to your computer.

Navigate to `C:\Python27` and *Shift*+ right-click on **Scripts**. Open the command window and type `pip install pyimgur` in that command window to install the Imgur library. Your terminal window will look like this:



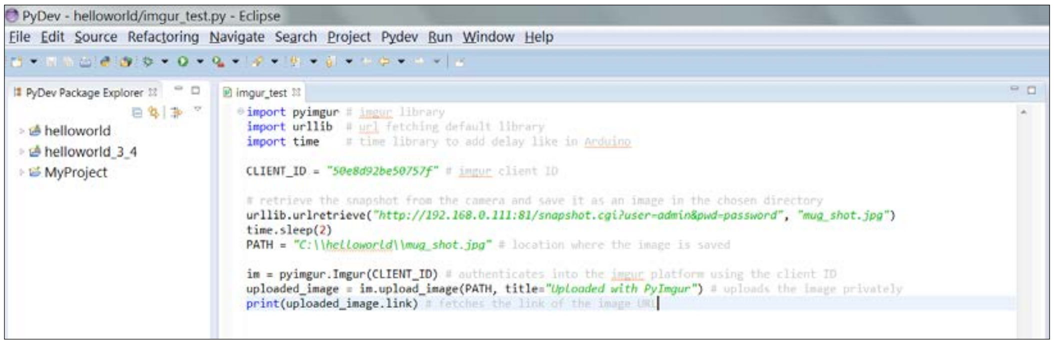
```
C:\Python27\Scripts>pip install pyimgur
Downloading/unpacking pyimgur
  Downloading pyimgur-0.5.2.tar.gz
  Running setup.py (path:c:\users\adith\appdata\local\temp\pip_build_Adith\pyimg
ur\setup.py) egg_info for package pyimgur

Requirement already satisfied (use --upgrade to upgrade): requests in c:\python2
7\lib\site-packages (from pyimgur)
Installing collected packages: pyimgur
  Running setup.py install for pyimgur

Successfully installed pyimgur
Cleaning up...

C:\Python27\Scripts>
```

Let's test this newly installed library. Open Eclipse and create a new file called `imgur_test.py`, and paste the following or simply load the `imgur_test.py` file from the code attached to this chapter:

```
import pyimgur # imgur library
import urllib  # url fetching default library
import time    # time library to add delay like in Arduino

CLIENT_ID = "<your client ID>" # imgur client ID

# retrieve the snapshot from the camera and save it as an image in the
chosen directory
urllib.urlretrieve("http://<your ip : port>/snapshot.
cgi?user=admin&pwd=password", "mug_shot.jpg")
time.sleep(2)
```

```
PATH = "C:\\<your python project name>\\mug_shot.jpg" # location where
the image is saved

im = pyimgur.Imgur(CLIENT_ID) # authenticates into the imgur platform
using the client ID
uploaded_image = im.upload_image(PATH, title="Uploaded with PyImgur")
# uploads the image privately
print(uploaded_image.link) # fetches the link of the image URL
```

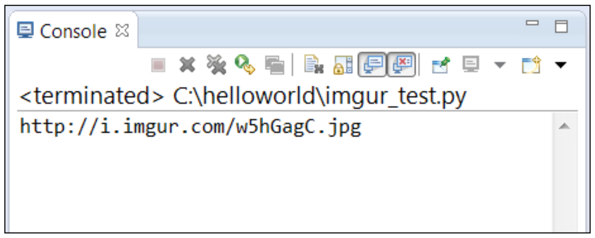
Change `<your client ID>` to the ID that you saved when you created an application on Imgur; change `<your ip : your port>` to your camera URL; change `<your python project name>` to the folder where all your python codes are saved. For example, if you look at your Eclipse IDE:


PyDev - helloworld/imgur_test.py - Eclipse
File Edit Source Refactoring Navigate Search Project Pydev Run Window Help
PyDev Package Explorer
helloworld
helloworld_3_4
MyProject
imgur_test

```
import pyimgur # imgur library
import urllib  # url fetching default library
import time    # time library to add delay like in Arduino

CLIENT_ID = "50e8d92be50757f" # imgur client ID

# retrieve the snapshot from the camera and save it as an image in the chosen directory
urllib.urlretrieve("http://192.168.0.111:81/snapshot.cgi?user=admin&pwd=password", "mug_shot.jpg")
time.sleep(2)
PATH = "C:\\helloworld\\mug_shot.jpg" # location where the image is saved

im = pyimgur.Imgur(CLIENT_ID) # authenticates into the imgur platform using the client ID
uploaded_image = im.upload_image(PATH, title="Uploaded with PyImgur") # uploads the image privately
print(uploaded_image.link) # fetches the link of the image URL
```

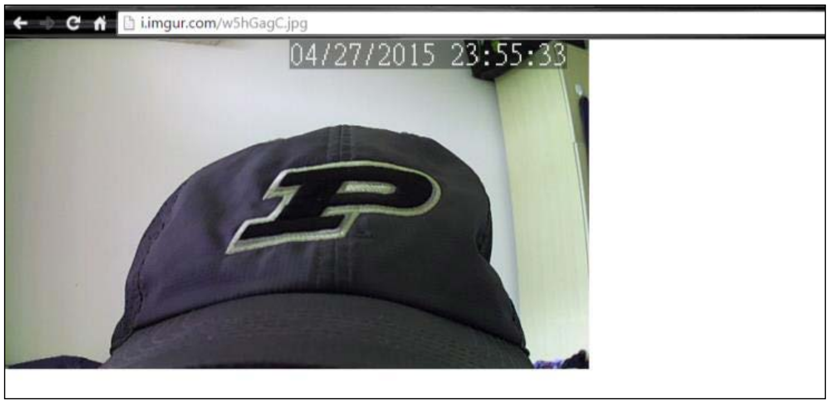
My project name is `helloworld`, and in the `C` drive I have a folder called `helloworld` where all the Python files are saved. So, set your `PATH` to correspond to that directory. For me, it will be `C:\\helloworld\\mug_shot.jpg` where `mug_shot.jpg` is the name of the saved file.

Once you have changed everything and ensured that your camera is on the wireless network, run the code. Run it using Python as you did in the `helloworld.py` example in the previous chapter and you should get a result with an Imgur link:



```
<terminated> C:\helloworld\imgur_test.py
http://i.imgur.com/w5hGagC.jpg
```

Copy and paste this link in a new browser window and you should have something like this:



Try refreshing the page and see what happens. Nothing! Exactly! This is why we took all the trouble of saving the image and then uploading it to a file sharing server. Because once the snapshot has been taken, it will not change.

Be mindful while using this method to upload images to Imgur. Do not abuse the system to upload too many images, because then your account will be banned.

Now, it is time to deal with sending a notification to your smart device.

Sending a notification to a smart device

There are many ways a notification can be sent to your smart device (e-mail, SMS, or via an app). During the course of writing this chapter, I realized that an e-mail is not the most efficient way to alert the user of an emergency (a burglar in this case), and there is no single global SMS notification service that can be used by people from different parts of the world. Hence, we are going to use yet another really powerful push messaging app called **Pushover**.

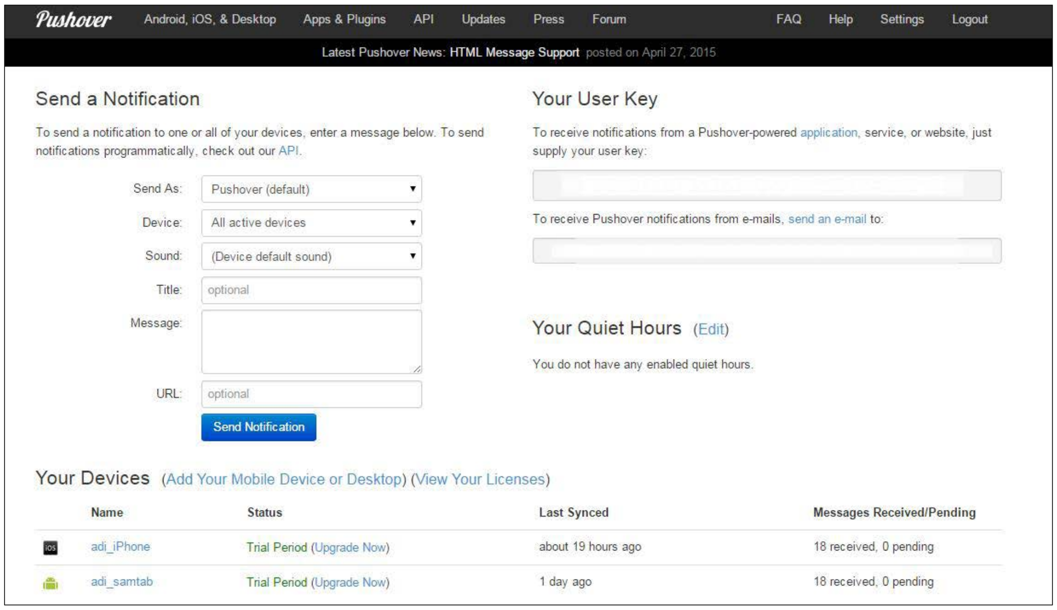
It works by communicating over the Internet, and conveniently there is a Python library associated with it that we will use to send notifications to the Pushover app on our smart device.

Pushover
Simple Notifications
Pushover makes it easy to get real-time notifications on your Android, iPhone, iPad, and Desktop.
GET IT ON Google play
Available on the App Store
https://client.pushover.net
All Notifications
Adium
E-mail Gateway
Fitbit Notifier

Go to `https://pushover.net/login` and create a new account. By default, you get a 7-day trial, which is sufficient for completing this chapter. If you like the software, you can go ahead and purchase it, as you can use it for future projects.

Once you have verified your e-mail ID, look up Pushover on the iTunes App store (`https://itunes.apple.com/en/app/pushover-notifications/id506088175?mt=8`) or on the Google Playstore (`https://play.google.com/store/apps/details?id=net.superblock.pushover&hl=en`) and download the app. Log in with the same credentials. It is important to allow push notifications for this app, because that is going to be its sole purpose.

Now, when you go back to `https://pushover.net/` and log in, you should be able to see your devices listed, as shown in the following:



Send a Notification

To send a notification to one or all of your devices, enter a message below. To send notifications programmatically, check out our API.

Send As:	Pushover (default)
Device:	All active devices
Sound:	(Device default sound)
Title:	optional
Message:	
URL:	optional

Your User Key

To receive notifications from a Pushover-powered application, service, or website, just supply your user key:

To receive Pushover notifications from e-mails, send an e-mail to:

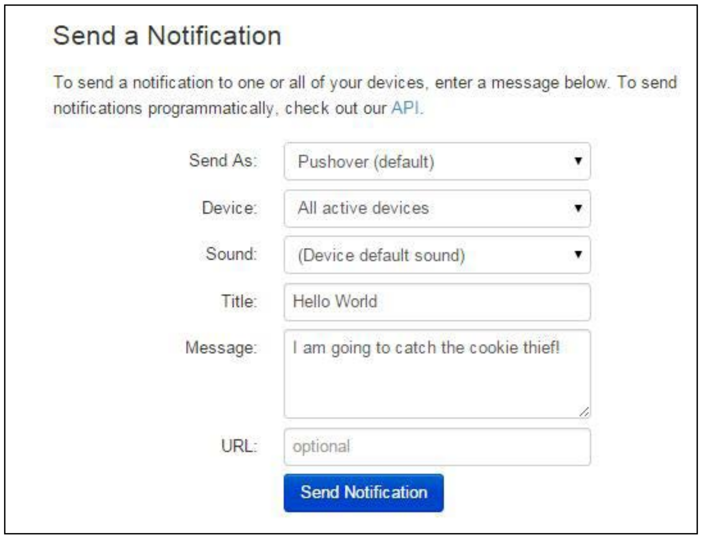
Your Quiet Hours (Edit)

You do not have any enabled quiet hours.

Your Devices (Add Your Mobile Device or Desktop) (View Your Licenses)

Name	Status	Last Synced	Messages Received/Pending
adi_iPhone	Trial Period (Upgrade Now)	about 19 hours ago	18 received, 0 pending
adi_samtab	Trial Period (Upgrade Now)	1 day ago	18 received, 0 pending

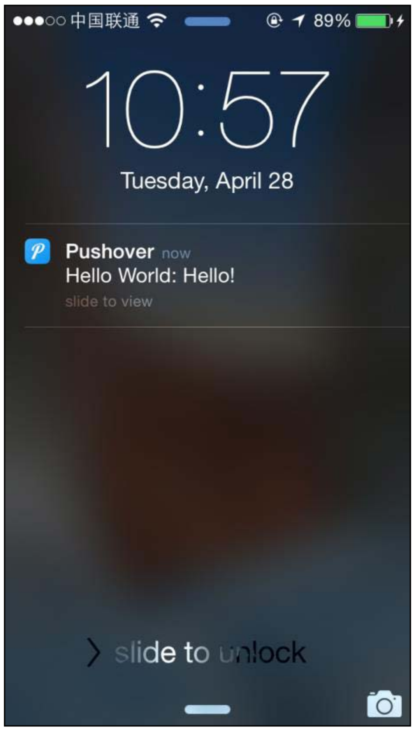
Note down your user key as we will need it while we create a script. For now, let us test the app. Under **Send a notification**, fill in the following:


Send a Notification

To send a notification to one or all of your devices, enter a message below. To send notifications programmatically, check out our API.

Send As:	Pushover (default)
Device:	All active devices
Sound:	(Device default sound)
Title:	Hello World
Message:	I am going to catch the cookie thief!
URL:	optional

Now, hit the **Send Notification** button. Wait for a couple of seconds and you should see a message that pops up on your smart device with the title **Hello World**:

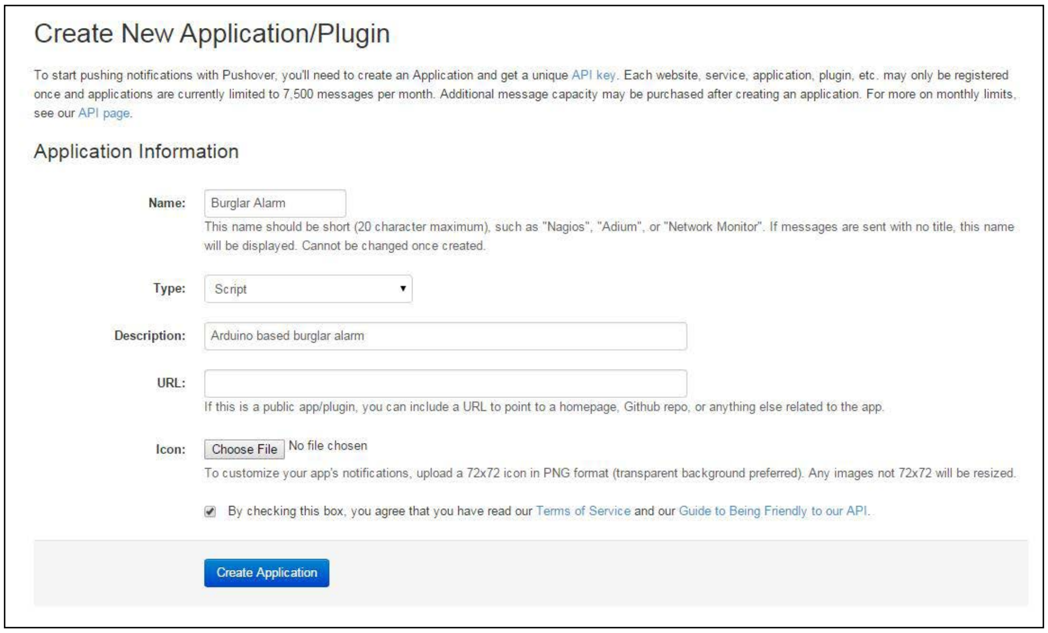


Hello World: Hello!

Pretty neat, huh?

Now, go ahead and register a new application on the Pushover website:

1. Fill in the details as you wish:



Create New Application/Plugin

To start pushing notifications with Pushover, you'll need to create an Application and get a unique API key. Each website, service, application, plugin, etc. may only be registered once and applications are currently limited to 7,500 messages per month. Additional message capacity may be purchased after creating an application. For more on monthly limits, see our API page.

Application Information

Name: Burglar Alarm
This name should be short (20 character maximum), such as "Nagios", "Adium", or "Network Monitor". If messages are sent with no title, this name will be displayed. Cannot be changed once created.

Type: Script

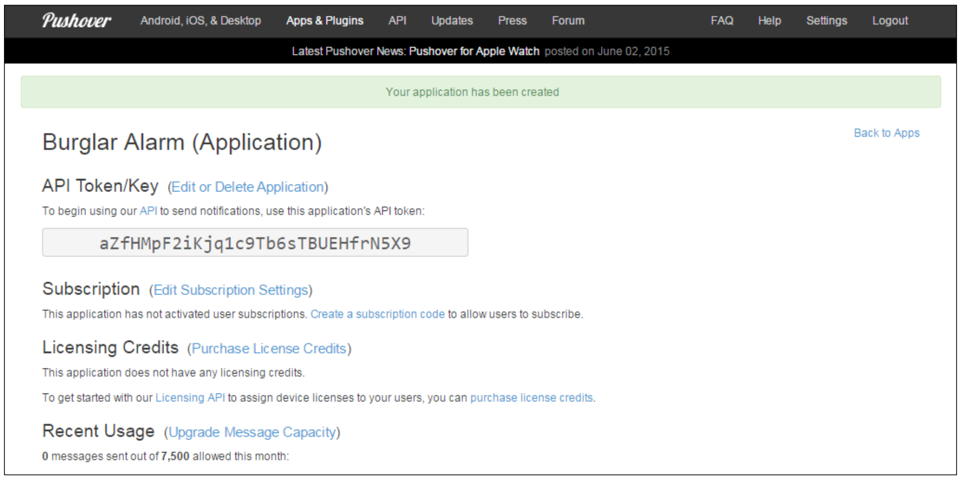
Description: Arduino based burglar alarm

URL:
If this is a public app/plugin, you can include a URL to point to a homepage, Github repo, or anything else related to the app.

Icon: Choose File No file chosen
To customize your app's notifications, upload a 72x72 icon in PNG format (transparent background preferred). Any images not 72x72 will be resized.

[✓] By checking this box, you agree that you have read our Terms of Service and our Guide to Being Friendly to our API.

2. Then, click on **Create Application** and you'll get the following page:



Latest Pushover News: Pushover for Apple Watch posted on June 02, 2015

Your application has been created

Burglar Alarm (Application)

API Token/Key (Edit or Delete Application)

To begin using our API to send notifications, use this application's API token:

```
aZfHMpF2iKjq1c9Tb6sTBUEHfrN5X9
```

Subscription (Edit Subscription Settings)

This application has not activated user subscriptions. Create a subscription code to allow users to subscribe.

Licensing Credits (Purchase License Credits)

This application does not have any licensing credits.

To get started with our Licensing API to assign device licenses to your users, you can purchase license credits.

Recent Usage (Upgrade Message Capacity)

0 messages sent out of **7,500** allowed this month:

3. Note down your **API Token/Key**.
4. Go to `C:\Python27` and *Shift* + right-click on **Scripts**; open a command window and type the following:

   ```
   pip install python-pushover
   ```

5. Hit *Enter*. This will install the `python-pushover` library for you.
6. Open Eclipse and create a new script, and call it `pushover_test.py`. Copy the following code into it, or load the `pushover_test.py` file that came with this chapter:

   ```
   from pushover import init, Client

   init("<token>")
   client = Client("<user-key>").send_message("Hello!",
   title="Hello")
   ```

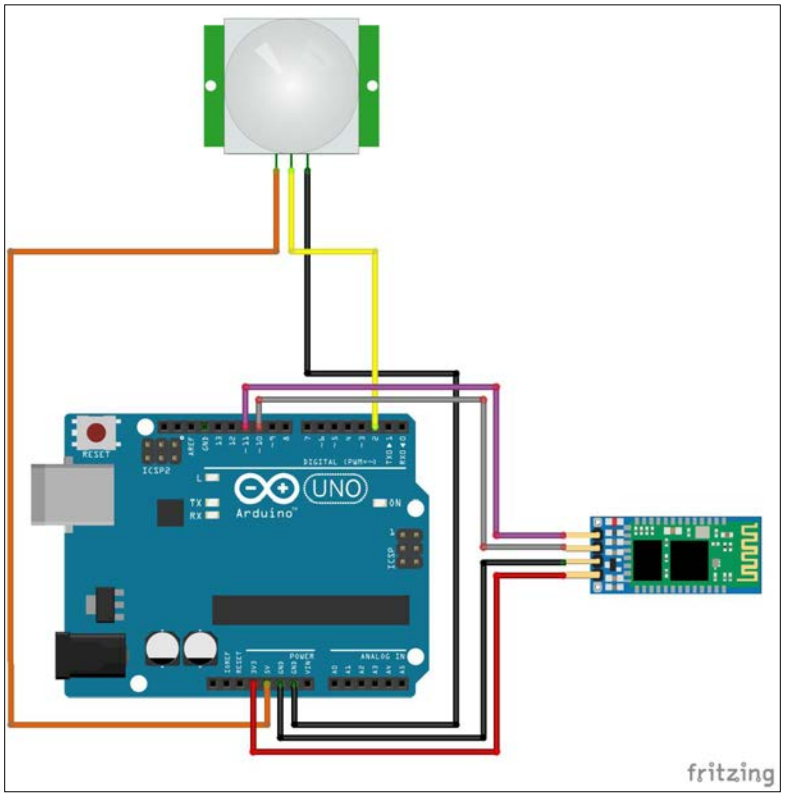
7. As you did for the Imgur tutorial, change `<token>` to your **API Token/Key**. Also, change `<user-key>` to your user key that is available when you sign in to your account at `https://pushover.net/`.

Run the script. Wait for a few seconds and you should get a similar notification as you got before. Fascinating stuff, if I say so myself.

Finally, we have finished working with and understanding the elements that will go into creating the entire compound. We are now going to put them all together.

Putting the pieces together

Go ahead and create this circuit for Arduino:



The connections are as follows:

- PIR → Arduino
- GND → GND
- OUT → D02
- VCC → 5V
- HC-06 → Arduino
- VCC → 3.3V
- GND → GND
- TXD → D10
- RXD → D11

Don't get mad, but there is one last library that you need to install in order to allow Arduino to communicate with Python: `pySerial`. Go to `https://pypi.python.org/pypi/pyserial`, download `pyserial-2.7.win32.exe`, and install it just like you install any other software. Then, we are ready.

Open Arduino and load the `alarm_bluetooth.ino` file that came with this chapter. It is recommended that you have the most up-to-date Arduino software before proceeding:

```
#include <SoftwareSerial.h> // serial library used for communication

SoftwareSerial burg_alarm(10, 11); // communicates via TX, RX at 10,
11 respectively
int ledPin = 13; // in built LED to show status changes
int pirPIN = 2;   // signal of the PIR sensor goes to pin 2
int pirState = LOW; // initiate the PIR status to LOW (no motion)
int pirVal = 0; // variable for storing the PIR status

void setup() {
  burg_alarm.begin(9600); // communication begins on baud 9600
  pinMode(ledPin, OUTPUT); // sets the LED pin as output
  delay(5000); // waits 5 seconds for motion to die down
}

void loop(){
  pirVal = digitalRead(pirPIN);  // read input from the sensor
  if (pirVal == HIGH) {          // if input is HIGH (motion detected)
    digitalWrite(ledPin, HIGH);  // turn LED ON
    delay(150);  // small delay
    if (pirState == LOW) { // checks if the PIR state is LOW while the
input is HIGH
```

```
    // this means, there wasn't motion before, but there is something
happening now
      Serial.println("Motion detected!"); // prints out an Alert
      burg_alarm.println('1'); // sends out '1' for when motion is
detected via Bluetooth to python
      pirState = HIGH; // sets the pirState to HIGH (motion detected)
    }
  } else { // no motion detected
      digitalWrite(ledPin, LOW); // turn LED OFF
      delay(300);  // small delay
      if (pirState == HIGH){ // if there was motion, but isn't any now
      Serial.println("Motion ended!");
      burg_alarm.println('0'); // sends a '0' when the motion has
ended
      pirState = LOW; // sets the state to LOW (no motion)
    }
  }
}
```

Make sure you unplug the TX and RX wires before uploading the code and then attach them back. Now, unplug the Arduino while we create a Python script.

Open Eclipse and load the `alarm_bluetooth.py` file, or just copy the following:

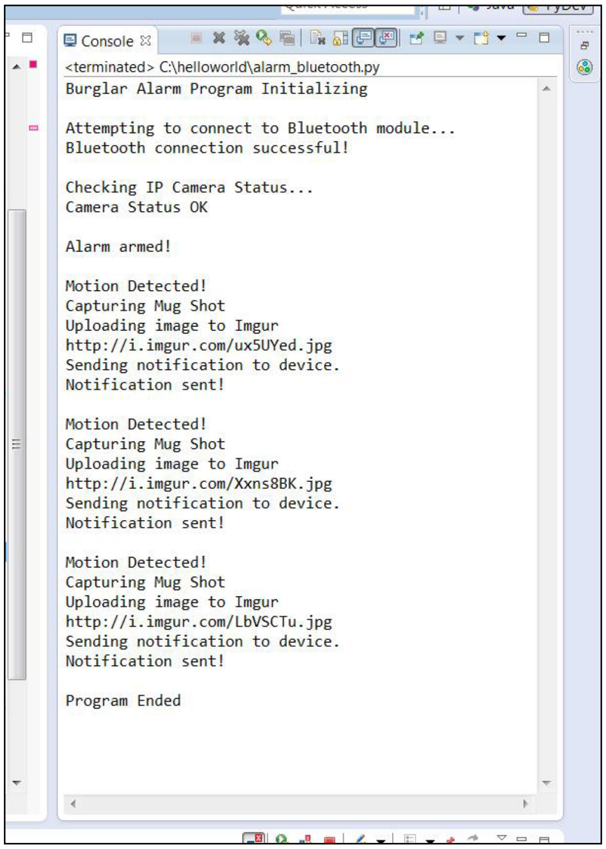
```
import serial
import pyimgur
import urllib
import time
from pushover import init, Client

print("Burglar Alarm Program Initializing")
init("< your push overtoken>")
CLIENT_ID = "<your client ID>"
PATH = "C:\\<your python folder>\\mug_shot.jpg"
im = pyimgur.Imgur(CLIENT_ID)
mug_shot_ctr = 0
serial_status = 0
camera_status = 0
try:
    print("\nAttempting to connect to Bluetooth module...")
    ser = serial.Serial('COM36', 9600) #Tried with and without the
last 3 parameters, and also at 1Mbps, same happens.
    time.sleep(3)
    serial_status = 1
    print('Bluetooth connection successful!')
except:
    print('Bluetooth Connection Error! \nPlease check if bluetooth is
connected.')
```

```
        mug_shot_ctr = 4

    try:
        print("\nChecking IP Camera Status...")
        urllib.urlretrieve("http://<your ip>/snapshot.
    cgi?user=admin&pwd=<your password>", "mug_shot.jpg")
        time.sleep(2)
        print("Camera Status OK")
        camera_status = 1
    except:
        print("Camera not connected!\nPlease check if camera is
    connected.")
        mug_shot_ctr = 4

    if((serial_status==1)&(camera_status==1)):
        print("\nBurglar Alarm armed!")

    while mug_shot_ctr < 3:
        line = ser.readline()
        if(line[0]=='1'):
            print('\nMotion Detected!')
            print('Capturing Mug Shot')
            urllib.urlretrieve("http://<your ip>/snapshot.
    cgi?user=admin&pwd=<your password>", "mug_shot.jpg")
            time.sleep(2)
            print('Uploading image to Imgur')
            uploaded_image = im.upload_image(PATH, title="Uploaded with
    PyImgur - Mugshot")
            print(uploaded_image.link)
            print('Sending notification to device.')
            Client("<your imgur ID>").send_message("Mug Shot: "+ uploaded_
    image.link, title="Intruder Alert!")
            print('Notification sent!')
            mug_shot_ctr = mug_shot_ctr + 1
    if(serial_status ==1):
        ser.close()
        print('\nProgram Ended')
```

Remember to change the tokens, keys, ports, the path, the camera IP, and the client ID (highlighted) to your specifics. The good thing about this code is that it checks whether the camera and Bluetooth are working as expected. If not, it gives an error message with which you can tell if something is wrong. Note that if you are not getting your camera IP to work, go back to the **IP Camera Search Tool** at `www.netcam360.com` to find the IP of your camera. It changes sometimes if it has been restarted.

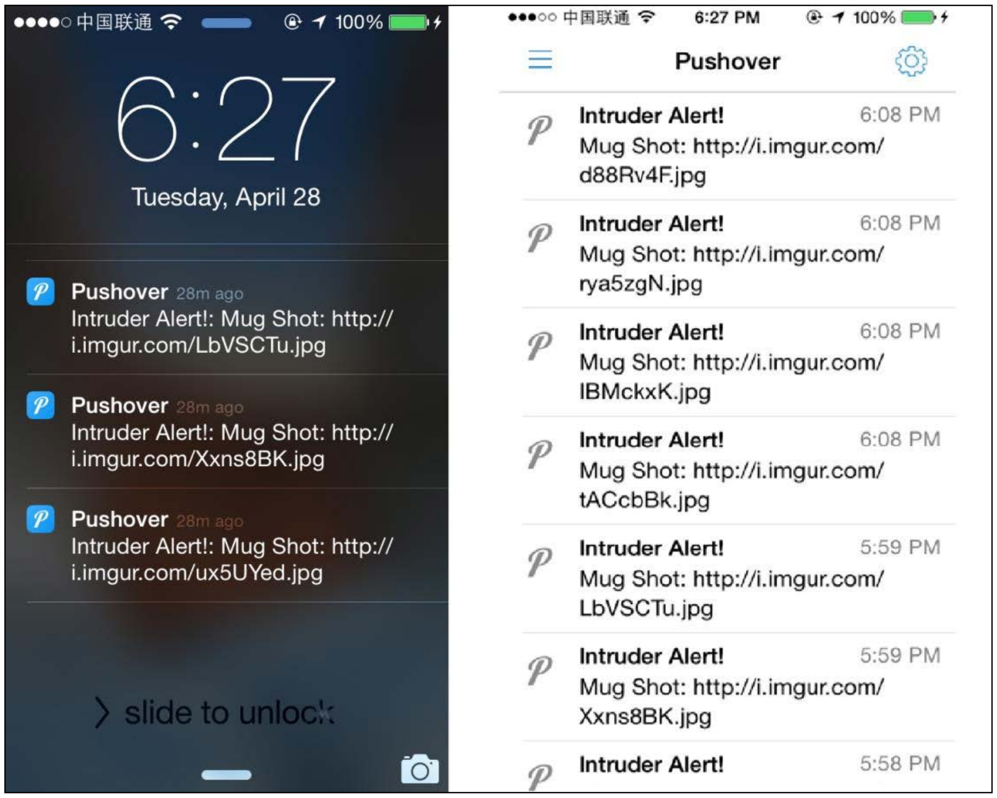
What this code does is…well, everything! It first gathers all the libraries that you have used so far. It then checks and connects to the Bluetooth module followed by the IP camera. Then, it waits for Arduino to send a message (`1`) saying motion has been detected. The script immediately fetches a snapshot from the IP camera's URL and uploads it to Imgur. Then, it uses Pushover to send the user an alert saying that motion has been detected, along with the Imgur URL that the user can open to see who the culprit is. The script will send three simultaneous images to give a better chance of catching the thief in action. This value can be changed by changing `mug_shot_ctr`.

Power your camera and your Arduino. Let's test this out and run the program. Move your hand in front of the PIR sensor and you will get an output like this:



```
Burglar Alarm Program Initializing

Attempting to connect to Bluetooth module...
Bluetooth connection successful!

Checking IP Camera Status...
Camera Status OK

Alarm armed!

Motion Detected!
Capturing Mug Shot
Uploading image to Imgur
http://i.imgur.com/ux5UYed.jpg
Sending notification to device.
Notification sent!

Motion Detected!
Capturing Mug Shot
Uploading image to Imgur
http://i.imgur.com/Xxns8BK.jpg
Sending notification to device.
Notification sent!

Motion Detected!
Capturing Mug Shot
Uploading image to Imgur
http://i.imgur.com/LbVSCTu.jpg
Sending notification to device.
Notification sent!

Program Ended
```

Immediately after the Python script has finished executing, you should see this on your smart phone:



●●●●○ 中国联通 100%

6:27

Tuesday, April 28

Pushover 28m ago
Intruder Alert!: Mug Shot: http://i.imgur.com/LbVSCTu.jpg

Pushover 28m ago
Intruder Alert!: Mug Shot: http://i.imgur.com/Xxns8BK.jpg

Pushover 28m ago
Intruder Alert!: Mug Shot: http://i.imgur.com/ux5UYed.jpg

slide to unlock

●●●○○ 中国联通 6:27 PM 100%

Pushover

Intruder Alert! 6:08 PM
Mug Shot: http://i.imgur.com/d88Rv4F.jpg

Intruder Alert! 6:08 PM
Mug Shot: http://i.imgur.com/rya5zgN.jpg

Intruder Alert! 6:08 PM
Mug Shot: http://i.imgur.com/IBMckxK.jpg

Intruder Alert! 6:08 PM
Mug Shot: http://i.imgur.com/tACcbBk.jpg

Intruder Alert! 5:59 PM
Mug Shot: http://i.imgur.com/LbVSCTu.jpg

Intruder Alert! 5:59 PM
Mug Shot: http://i.imgur.com/Xxns8BK.jpg

Intruder Alert! 5:58 PM

The only thing left to do now is to position your camera in such an area where it keeps an eye on a possible entry point. The Arduino, with the PIR sensor and Bluetooth, needs to be placed close to the entry point so that the PIR sensor can detect when a door or window is opened.

It is advisable to use a 2.1mm 9-12V DC adapter to power the Arduino, as shown in the following:

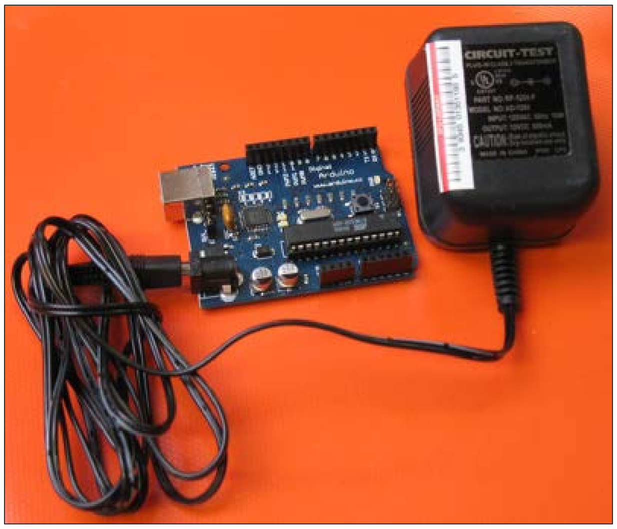
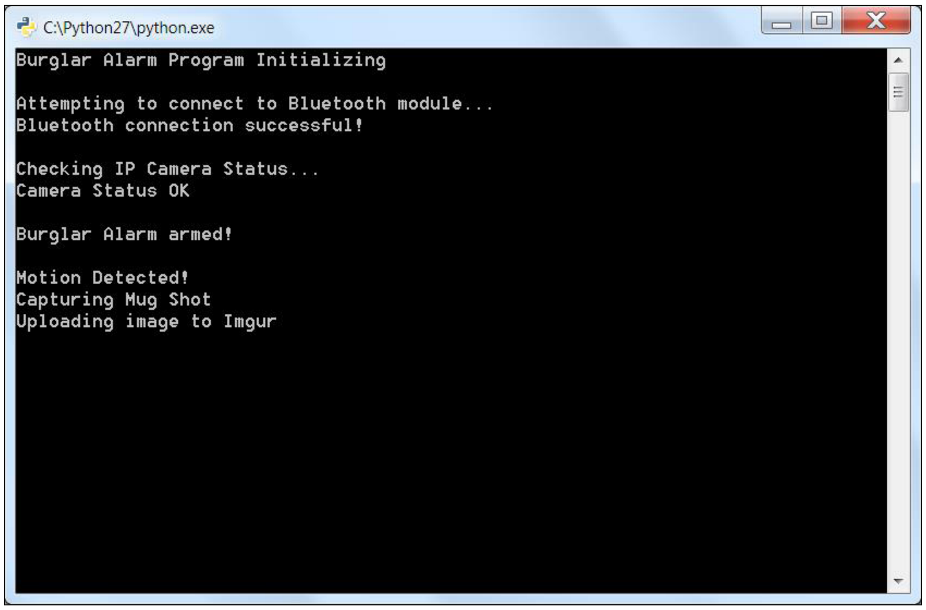


Image source: `http://playground.arduino.cc/Learning/WhatAdapter`

Also, instead of using Eclipse, now that the code has been prepared, you can navigate to `C:\<your Python project directory>` and double-click on `alarm_bluetooth.py` to run it directly. It will look like this:



```
Burglar Alarm Program Initializing

Attempting to connect to Bluetooth module...
Bluetooth connection successful!

Checking IP Camera Status...
Camera Status OK

Burglar Alarm armed!

Motion Detected!
Capturing Mug Shot
Uploading image to Imgur
```

Done! You are finally done! You have created your very own high-tech burglar alarm system using Arduino, Bluetooth, Python, Imgur, and Pushover. If you have reached this point, I want to congratulate you. It has not been an easy journey, but it is definitely worth the patience and hard work. Your home is now secure. "However, what if we want to do this without the need for a computer?" We would have to use something such as a Raspberry Pi, but that is beyond the scope of this book. If you are adventurous, I am not going to stop you from trying to make this project standalone.

Summary

This was a long project, wasn't it? However, it is truly worth the end product, as well as the knowledge gained from each element that went together, mainly, Arduino, Bluetooth, and Python. Sometimes, instead of wasting time creating something completely from scratch, it is often a good idea to use what already exists and tweak it to do what we want. We did this for Imgur and Pushover, both very powerful tools. I hope you enjoyed and had a lot to take away from this chapter.

In the next chapter, we will take networking to a whole new level by creating a master remote for your entire home. Yes, you've guessed it – home automation.

6
Home Automation – Part 1

In the previous chapter, you learnt how to merge a wireless camera, a PIR sensor, the Internet, and some of the powerful software with an Arduino to make a high tech security alarm system for your home or office. This time, we will be working on the similar lines. Well, the title has already given it away; we will be creating a home automation system. Before you get into this chapter, take a moment to look back at what you have achieved. You are half way done!

This chapter is going to be really exciting. "Why?", you ask. You are going to control the lights, fans, and other electrical appliances, using your smart phone. In addition to this, we will also be implementing speech recognition! You can literally control your home using your words. Enough of the sales pitch; now, let's get down to business.

For this project, we are going to use a Wi-Fi Arduino shield connected to your home's Wi-Fi network in order to communicate with your smart phone or computer (running speech recognition), with which you will be able to switch appliances on or off.

If you have not worked with high voltages before, you need to be very careful while proceeding with this chapter. It would be advisable to call a trusted electrician to help you with this project. Specific instances where an electrician would be recommended will be highlighted as we go through the chapter.

This project, Home Automation, is split into two chapters to balance the length of each chapter. The chapter is further split into the following sections:

- Connecting the Wi-Fi module
- Using relays to control appliances
- Communicating through a terminal

You have been promised before that you would learn many new things about the Arduino world. In the previous project, our communication was mainly through Bluetooth. Here, we will utilize a different approach: Wi-Fi. Wi-Fi is increasingly becoming more popular than Bluetooth when it comes to the Internet of Things or having a centralized network, because it has a much greater range than Bluetooth and it can also be used to access the Internet, which ultimately increases its range to the entire planet. What's the drawback? Well, using Wi-Fi modules in projects such as home automation is a relatively new idea; hence, the modules are more expensive than their Bluetooth counterparts. However, they are much more powerful. Let's move on to the prerequisites.

Prerequisites

This topic will cover what parts you need in order to create a good home automation system. Obtaining software will be explained as the chapter progresses.

The materials needed are as follows:

- 1x Arduino UNO board or Arduino MEGA
- 1x USB cable A to B (also known as the printer cable)
- 1x CC3000 Wi-Fi shield
- 1x 5V relay (Arduino compatible)
- 1x Two-way switch replacement (for your switchboard)
- 1x Regular type screwdriver
- 1x Multimeter
- 1x 9VDC 2A 2.1mm power adapter
- 1x Wireless router (with accessible settings)
- 1x Smart phone
- 1x PC with a microphone
- 10x Connecting wires
- 1x 5V 4 Channel relay (optional: Arduino compatible)
- 4x Two-way switch replacement (depending on the number of relays)

The softwares required are as follows:

- Putty (terminal)
- .cmd (iOS)
- UDP TCP Server Free (Android)
- BitVoicer (speech recognition)

If you want to control your home with just your smart phone and not use the physical switches, then you do not require the 'two-way switch', but as you go through this chapter, you will understand why a two-way switch is listed here.

The 4 channel relay is used to control four appliances. It is up to you as to how many appliances you want to ultimately control, and buy the necessary number of channel relays.

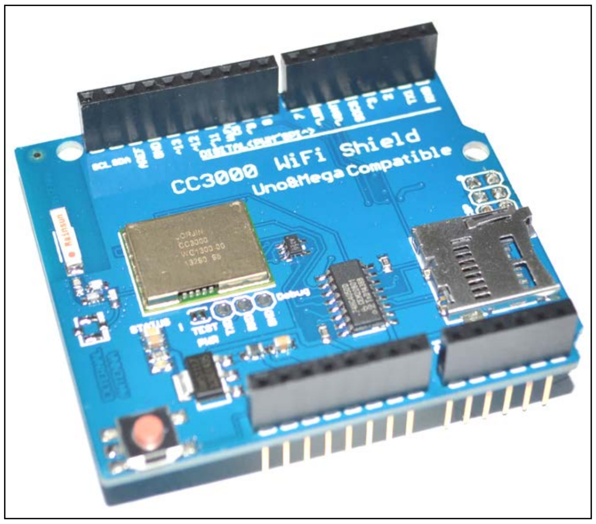
Here, Arduino MEGA is preferred if you want to control more than five appliances. Everything is the same as UNO, except the MEGA has much more pins. However, in this project, we are going to use an Arduino UNO.

Connecting the Wi-Fi module

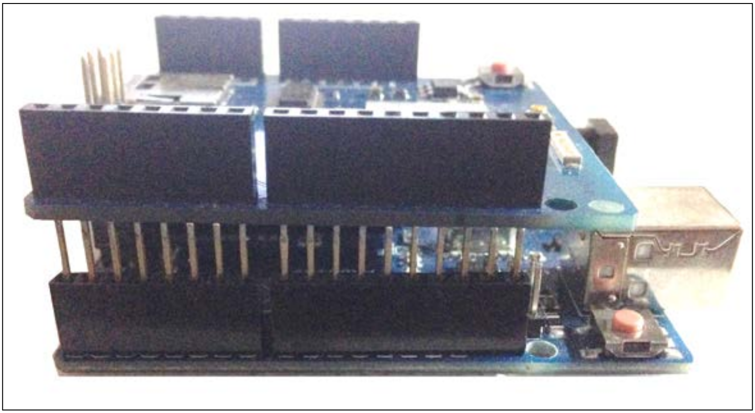
In this case, the Wi-Fi module is the CC3000 shield. Shields are basically ready-to-go modules that can be directly attached to the Arduino with any extra wiring. This makes them quite convenient to use. In this section, you will learn about connecting the Arduino to your home Wi-Fi network and linking it to the Internet.

The CC3000 Arduino shield

This shield is a relatively new means of communication for Arduino. The CC3000 chip is made by Texas Instruments, with the goal to simplify internet connectivity for projects such as the one we are going to make in this and the following chapter.



Connecting the shield to the Arduino is probably the simplest task in this entire book. Make sure the male header pins of the shield align with the female header pins of the Arduino board (UNO or MEGA), and then gently mount them together. You will have something like this:



If you would like to know more about how the chip works, you should refer to this page (at `http://www.ti.com/product/cc3000`) at the Texas Instruments website. When you visit this page, it will tell you that it is recommended to use the newer CC3200 version of the chip. However, at the time of writing this chapter, there were no easy-to-use Arduino compatible CC3200 modules; hence, we will stick to CC3000 because it has a lot of community support, which really helps if you come across some unprecedented problem.

The thing that you need to be wary about with this particular shield is that your USB cable from the computer will sometimes not be able to completely meet the power demands of the shield. Hence, we make use of the power adapter when using it for the project. To learn more about this and other frequently asked questions, check out `https://learn.adafruit.com/adafruit-cc3000-wifi/faq`.

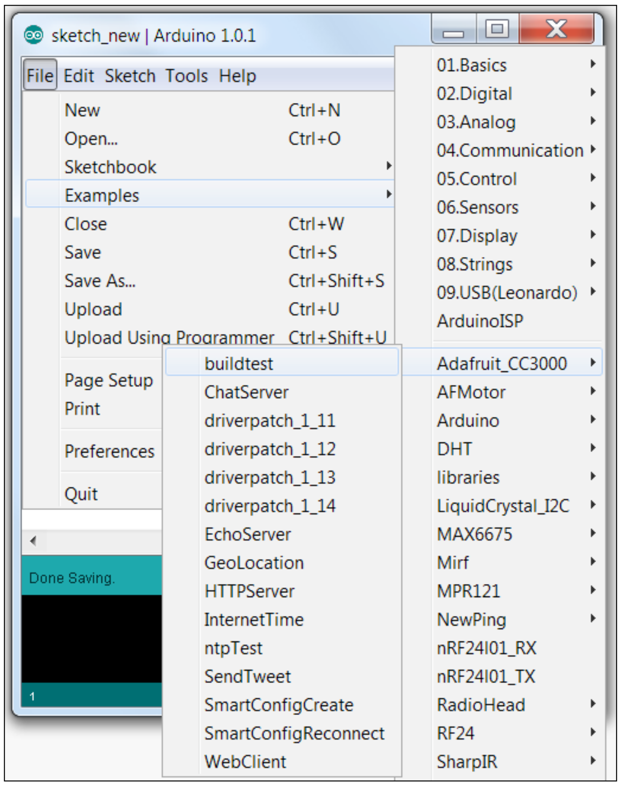
Testing the shield

Since you have already connected the shield, there is no need for any circuit diagrams just yet. However, we do have to install an Arduino library. You have mastered this by now, haven't you?

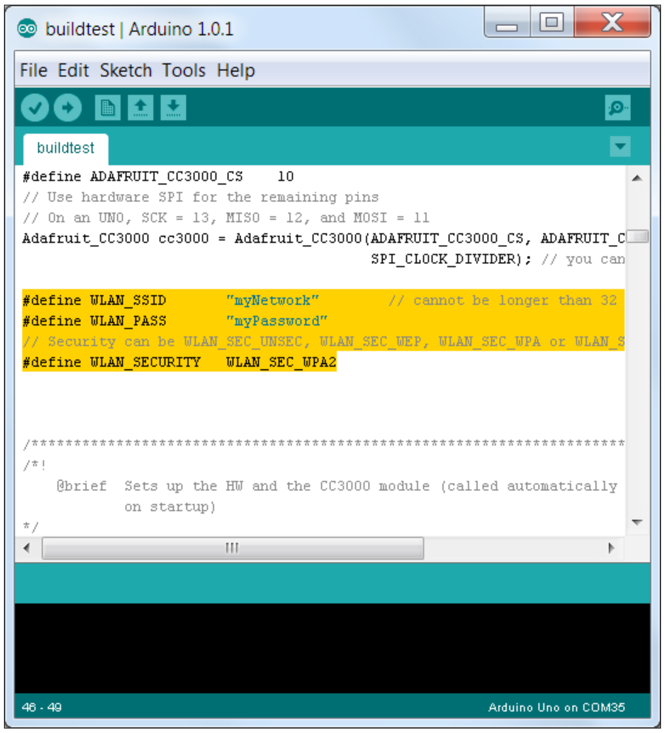
Before that, you will have to install an older version of Arduino to your computer. This is because the newer versions are not completely compatible with the CC3000. Go to `http://www.arduino.cc/en/Main/OldSoftwareReleases` and install 1.0.6 or a lower version of the Arduino IDE.

Once that is done, as before, go ahead and install the CC3000 library by Adafruit from `https://codeload.github.com/adafruit/Adafruit_CC3000_Library/zip/master`. As before, extract the contents of the ZIP file to `C:\Users\<user>\Documents\Arduino\libraries`. Rename the folder as `Adafruit_CC3000` or `CC3000_WIFI`, something that is short and recognizable.

Now, open up the older Arduino IDE. Go to **File→Examples→Adafruit_ CC3000→buildtest**:



This code can be run directly, with only a few changes. Go to line 46 of the code and take a look at the following four lines of code:

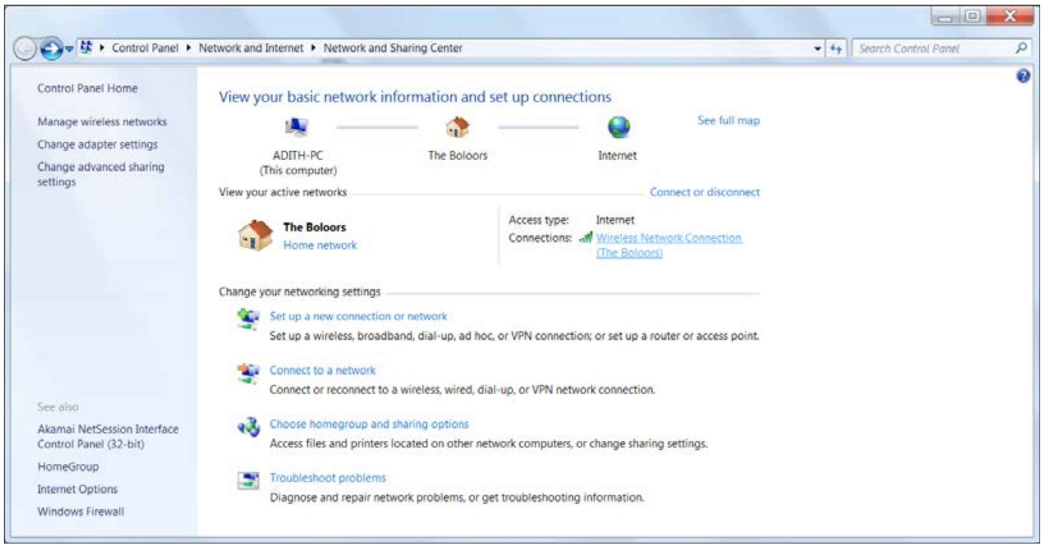


```
#define ADAFRUIT_CC3000_CS    10
// Use hardware SPI for the remaining pins
// On an UNO, SCK = 13, MISO = 12, and MOSI = 11
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS, ADAFRUIT_C
                                         SPI_CLOCK_DIVIDER); // you can

#define WLAN_SSID       "myNetwork"        // cannot be longer than 32
#define WLAN_PASS       "myPassword"
// Security can be WLAN_SEC_UNSEC, WLAN_SEC_WEP, WLAN_SEC_WPA or WLAN_S
#define WLAN_SECURITY   WLAN_SEC_WPA2



/**************************************************************************
/*!
    @brief  Sets up the HW and the CC3000 module (called automatically
            on startup)
*/
```

The only things that you will have to change in this code to make it work are `myNetwork`, `myPassword` and `WLAN_SEC_WPA2`.

Change `myNetwork` to your Wi-Fi name/SSID and `myPassword` to your password. Do not forget the double quotation marks because they are string type constants that require quotations so that the software can recognize them as constants.

If you are not using a WPA2 type security `WLAN_SEC_WPA2` should be changed. You can check this by going to your network settings. Go to **Network and Sharing Center** in the control panel or right-click on the Wi-Fi tray icon and select the same:



View your basic network information and set up connections

View your active networks

The Boloors
Home network

Access type: Internet
Connections: Wireless Network Connection (The Boloors)

Change your networking settings

Set up a new connection or network
Set up a wireless, broadband, dial-up, ad hoc, or VPN connection; or set up a router or access point.

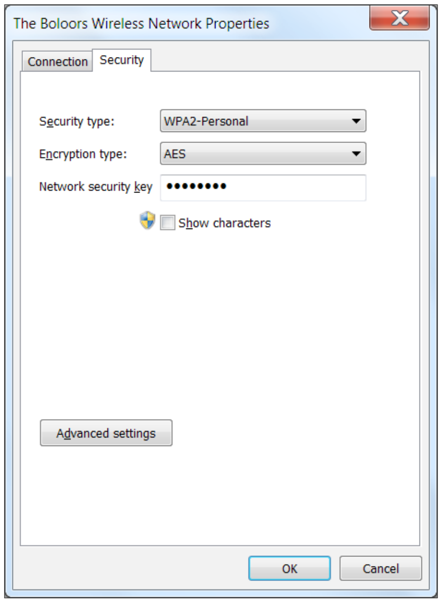
Connect to a network
Connect or reconnect to a wireless, wired, dial-up, or VPN network connection.

Choose homegroup and sharing options
Access files and printers located on other network computers, or change sharing settings.

Troubleshoot problems
Diagnose and repair network problems, or get troubleshooting information.

Then, click on **Wireless Network Connection** and select **Wireless Properties**.

Go to the **Security** tab and you will have something like this:



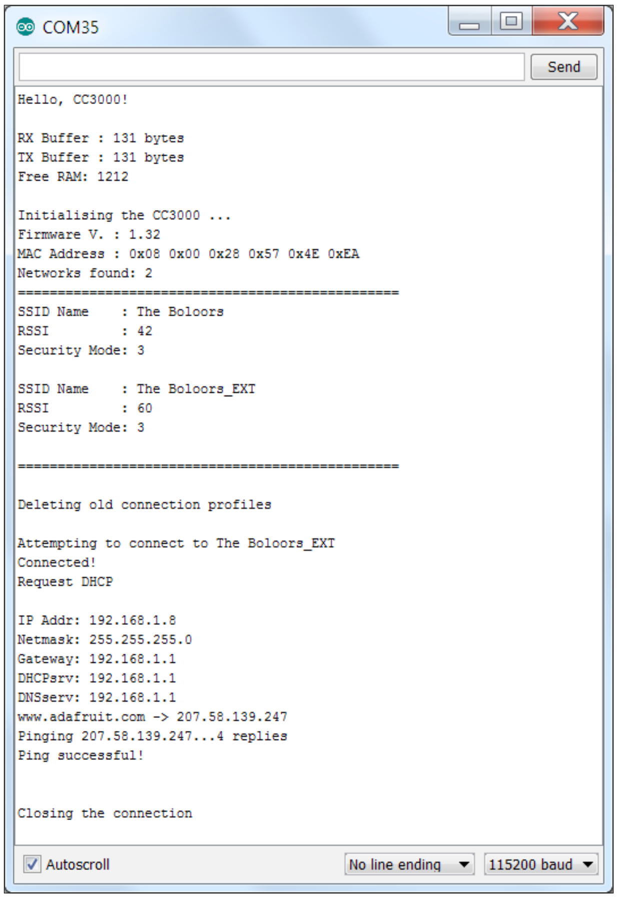
Security type: WPA2-Personal
Encryption type: AES
Network security key ••••••••

The **Security type** field tells you what sort of security access your router is using. It does not matter if it is WPA2-Personal or WPA2-Enterprise. As long as you identify the first segment of the field (WPA2 in this case), you will know what you have to change in your Arduino code.

Once you have changed the parameters for the Wi-Fi in the code, go ahead and connect the Arduino to your computer using the USB cable. If your computer has a USB 3.0 port, use that. This is because the USB 3.0 provides more current than the USB 2.0. If not, don't fret, this program will still run without any external power.

Once the program has been uploaded, open up the Serial Monitor and change the baud rate to 115200. If for some reason the board is acting weird, go ahead and connect the 9V 2A power adapter to power the Arduino.

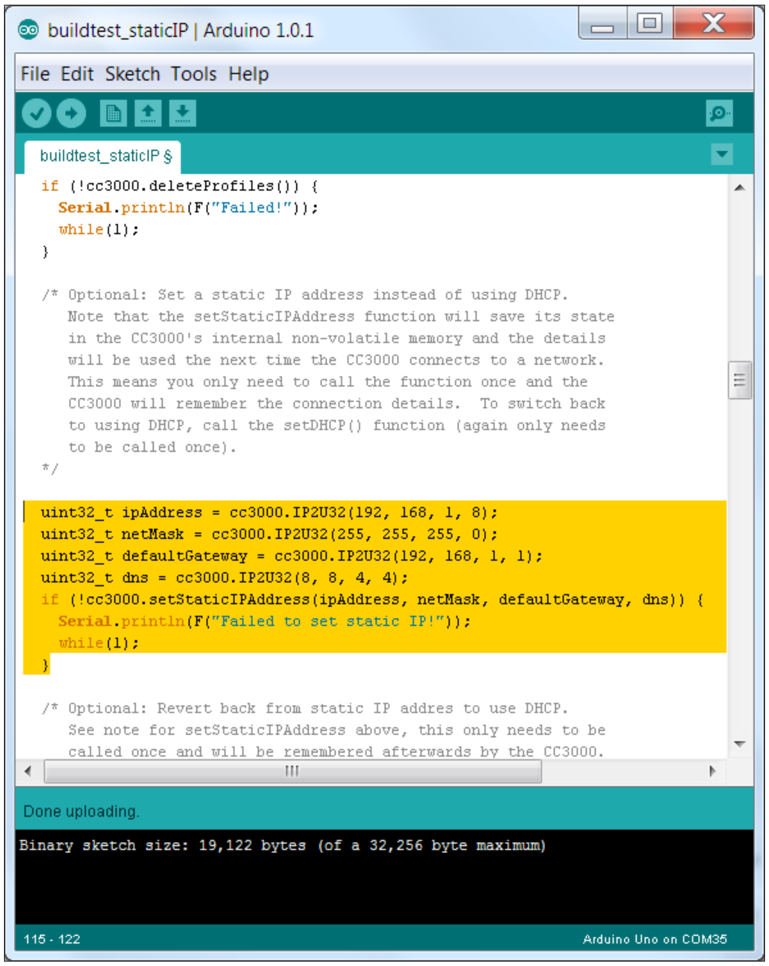
If everything worked well, you will see something like this:



```
Hello, CC3000!

RX Buffer : 131 bytes
TX Buffer : 131 bytes
Free RAM: 1212

Initialising the CC3000 ...
Firmware V. : 1.32
MAC Address : 0x08 0x00 0x28 0x57 0x4E 0xEA
Networks found: 2
================================================
SSID Name    : The Boloors
RSSI         : 42
Security Mode: 3

SSID Name    : The Boloors_EXT
RSSI         : 60
Security Mode: 3

================================================

Deleting old connection profiles

Attempting to connect to The Boloors_EXT
Connected!
Request DHCP

IP Addr: 192.168.1.8
Netmask: 255.255.255.0
Gateway: 192.168.1.1
DHCPsrv: 192.168.1.1
DNSserv: 192.168.1.1
www.adafruit.com -> 207.58.139.247
Pinging 207.58.139.247...4 replies
Ping successful!


Closing the connection
```

If you watched your Serial Monitor while doing this, you will have noticed that it takes quite some time to acquire the DHCP settings. For example, `IP Addr` is the unique number given by the router to allow CC3000 to connect to the router. However, what happens sometimes is that the router is restarted (power outage), thus the IP address might change. Since ultimately we will not be always monitoring the Serial Monitor for `IP Addr`, let us give it a constant, one that solves both the problems, namely, the long time it takes to get DHCP settings and the possibility of the IP address changing. Use the latest `driverpatch_X_XX.ino` example to update the firmware if it isn't updated already.

Load the `buildtest_staticIP.ino` file that came with this book. The main difference in this code is that we have uncommented a part of the code (highlighted in the following image) and changed the IP address to match the preceding image:


buildtest_staticIP | Arduino 1.0.1
File Edit Sketch Tools Help
buildtest_staticIP §

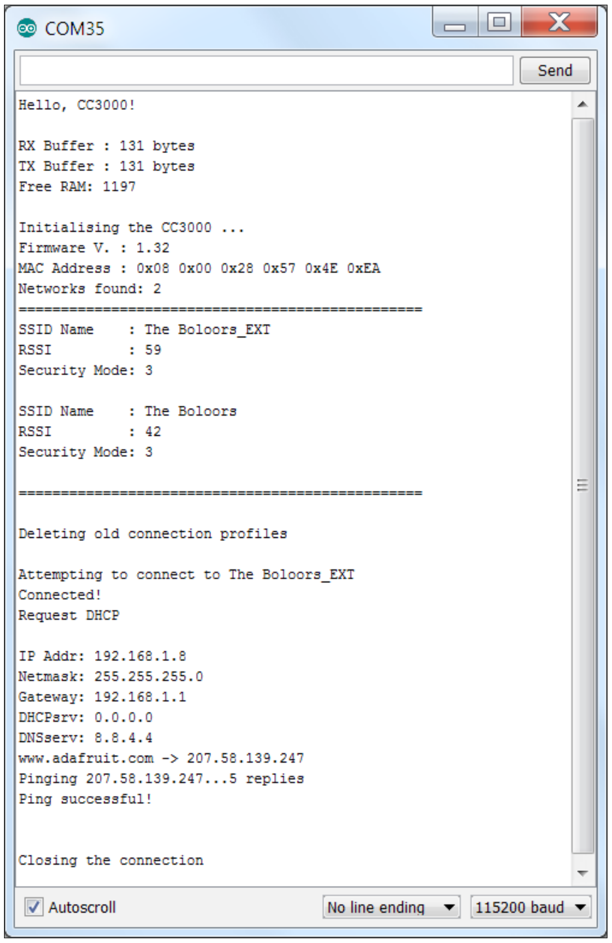
```
  if (!cc3000.deleteProfiles()) {
    Serial.println(F("Failed!"));
    while(1);
  }

  /* Optional: Set a static IP address instead of using DHCP.
     Note that the setStaticIPAddress function will save its state
     in the CC3000's internal non-volatile memory and the details
     will be used the next time the CC3000 connects to a network.
     This means you only need to call the function once and the
     CC3000 will remember the connection details.  To switch back
     to using DHCP, call the setDHCP() function (again only needs
     to be called once).
  */

  uint32_t ipAddress = cc3000.IP2U32(192, 168, 1, 8);
  uint32_t netMask = cc3000.IP2U32(255, 255, 255, 0);
  uint32_t defaultGateway = cc3000.IP2U32(192, 168, 1, 1);
  uint32_t dns = cc3000.IP2U32(8, 8, 4, 4);
  if (!cc3000.setStaticIPAddress(ipAddress, netMask, defaultGateway, dns)) {
    Serial.println(F("Failed to set static IP!"));
    while(1);
  }

  /* Optional: Revert back from static IP addres to use DHCP.
     See note for setStaticIPAddress above, this only needs to be
     called once and will be remembered afterwards by the CC3000.
```

```
Done uploading.
Binary sketch size: 19,122 bytes (of a 32,256 byte maximum)
```

In this case, `(192, 168, 1, 8)` has been used because this is the IP address the router allotted to `cc3000` when it was first connected. You should use the IP address that your `cc3000` was allotted.

Do not forget to change `myNetwork` and `myPassword` to reflect your router's configuration.

If you did it correctly, you will see the following in your Serial Monitor:



```
Hello, CC3000!

RX Buffer : 131 bytes
TX Buffer : 131 bytes
Free RAM: 1197

Initialising the CC3000 ...
Firmware V. : 1.32
MAC Address : 0x08 0x00 0x28 0x57 0x4E 0xEA
Networks found: 2
================================================
SSID Name    : The Boloors_EXT
RSSI         : 59
Security Mode: 3

SSID Name    : The Boloors
RSSI         : 42
Security Mode: 3

================================================

Deleting old connection profiles

Attempting to connect to The Boloors_EXT
Connected!
Request DHCP

IP Addr: 192.168.1.8
Netmask: 255.255.255.0
Gateway: 192.168.1.1
DHCPsrv: 0.0.0.0
DNSserv: 8.8.4.4
www.adafruit.com -> 207.58.139.247
Pinging 207.58.139.247...5 replies
Ping successful!


Closing the connection
```

Isn't the DHCP so much faster now? You can even try restarting your router, and running the code again will give you the same IP address. If you ever want to go back to the dynamic IP, run the default `buildtest` example that comes with the Adafruit CC3000 library.

Note that the antenna of the CC3000 is not as powerful as that of your smart phone or computer. So, it is recommended that the Arduino and CC3000 is placed as close to the Wi-Fi as possible. The `buildtest` program is your go-to code for making sure that the CC3000 shield is working as expected.

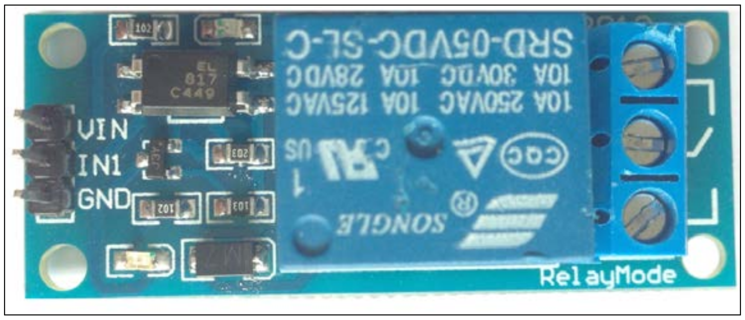
In a similar fashion, go ahead and try the GeoLocation and InternetTime programs under examples. Pretty neat, huh?

Using relays to control appliances

Here, we learn what relays are and how they can be used to control everyday electrical/electronic appliances. Also, to make them useful, we will connect them to the Arduino and create a `Blink.ino` program, except this time it will be your lamp that will be turning on and off.

Understanding the basics of the relay

A relay is basically an electrical device, usually consisting of an electromagnet, which is activated by a passing of current in one circuit to open or close another circuit. You could think of it as a switch that requires current to turn on or off.



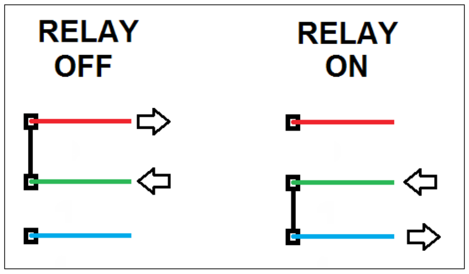
Why can't we simply connect a pin of the Arduino to the switchboard, and switch the fan on or off like we do with an LED? Well, an Arduino can output only 2-5V, whereas the fan or any other appliance in your house uses around 200-250V that comes from the electricity grid. This number is different depending on where you are from. Also, we cannot simply connect the Arduino to the switch of the fan because that 200-250V will get fed into the Arduino, which would instantly burn the chip, or worse.

Since the relay uses an electromagnet to flip a switch inside it, there is no physical contact between the circuitry of the Arduino and the circuitry of the fan; hence, it is safe and very effective.

Diving deeper into relay functionality

If you have a good look at the relay (preceding image), you will notice that the three male header pins on the left are to be connected to the Arduino and the three 'screws' are to be connected to an electronic appliance.

We need to understand how these two parts work with each other so that we can program the Arduino as we require. The left part will simply be connected to an Arduino OUTPUT pin, which we can control (turn ON and OFF—HIGH and LOW) just like an LED. This, in turn, flips a switch on the right side of the relay. Imagine wires are connected to each of the three screws (right side) of the relay. This depiction will show how the circuitry will look like on the right side of the relay comprising of its three pins/screws (Arduino LOW → Relay OFF):



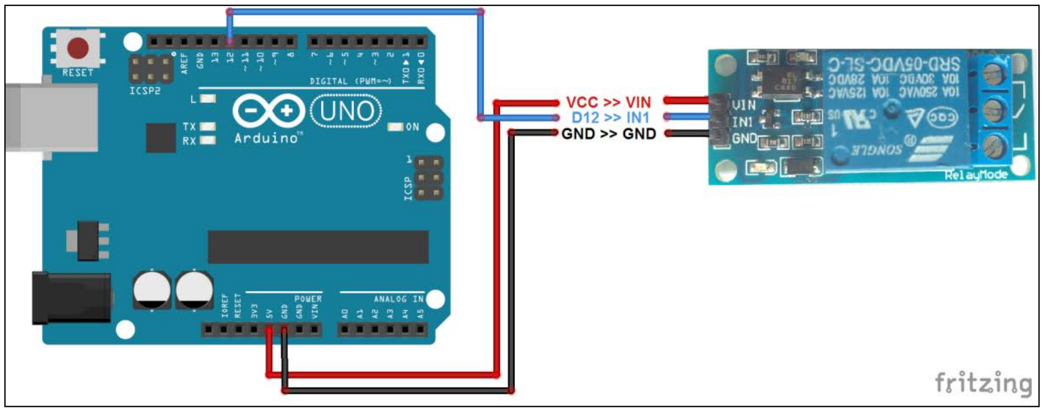
When the relay state is 'OFF', the top screw and the middle screw of the relay form a closed circuit (completed connection), allowing the current to pass through it. Also, as you guessed it, when the relay state is 'ON', the bottom two screws will form a closed circuit.

We use a two-way switch instead of a simple switch, because a two-way switch enables us to control the appliance through the Arduino and the physical switch independently. This means that you can turn ON a lamp via the Arduino, and physically flip the switch to turn it off.

Next, we will move to actually programming a relay.

Programming a relay

Programming a relay is almost as easy as programming an LED. You can leave the CC3000 Wi-Fi shield mounted onto the Arduino and use the pins on it to connect to the relay. Go ahead and build the following circuit:



The connections from the Arduino to the relay are as follows:

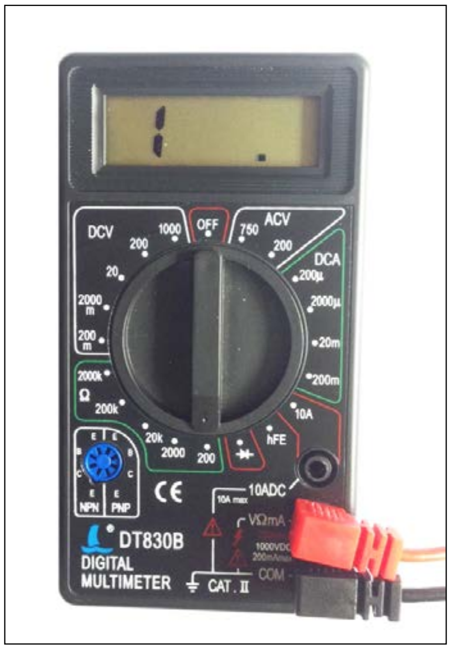
- GND → GND
- 5V → VIN
- D12 → IN1

Since we are just going to test the relay for now, open Arduino and fire up the blink LED example, and instead of using pin 13 for the LED, just use pin 12 (connected to IN1 of the relay). Also, increase the delay time from 1000 to 3000. Basically, it will look like this:

```
void setup() {
  // initialize digital pin 12 as an output.
  pinMode(12, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage
level)
  delay(3000);              // wait for a second
  digitalWrite(12, LOW);    // turn the LED off by making the voltage
LOW
  delay(3000);              // wait for a second
}
```

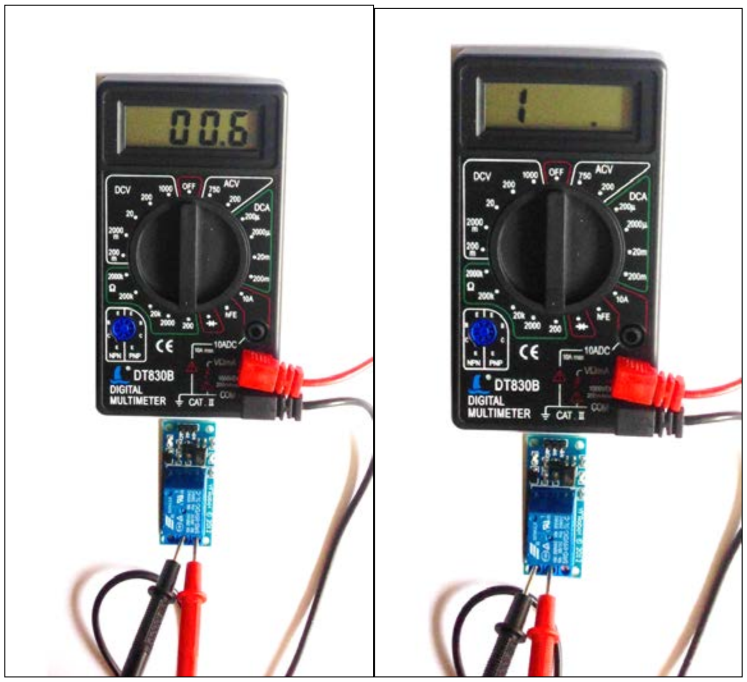
Run the program. If the upload is successful and if you are lucky enough to have an LED on the relay board, you will see it go on and off in three-second intervals. You will also hear a clicking sound whenever the LED turns on and off. This is the sound of the physical motion of a metallic component due to the electromagnetism that completes the circuit. So there you go – this is a simple working example of a relay.

Now, fetch your multimeter and set the parameter measuring knob to resistance 200Ω. See the following image if you are unsure:



Now, holding the two wires of the multimeter touch the noninsulated ends to each other. You should get a reading of 0 Ω. This is because there is virtually no resistance between the two contact points (the two wires).

Use the two wires to touch the top and the middle screws of the relay, as shown in the following image:



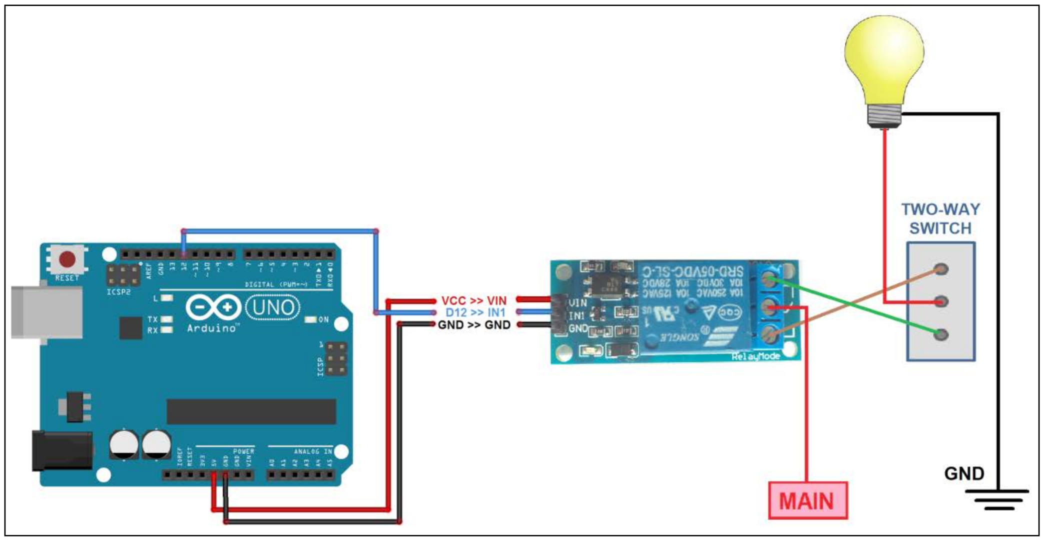
You will notice that when the relay is turned off, the resistance between the top two screws will be almost 0 Ω. This implies that the circuit is complete (take a look at the previous images of the relay schematic for a diagrammatic representation). When you touch the bottom two screws of the relay when the relay is turned ON, the resistance will again be zero and the bottom circuit will be complete. If you do the same when the relay is turned OFF, the resistance will be infinite, but the multimeter will display **1**. If your relay behaves in an opposite manner, make a note of that and we will change the circuit diagram accordingly.

Testing the relay with a light bulb

"One more test? Seriously?" Yes. Before we begin programming communications into the system, we need to ensure that the relay would actually work in your home or whatever experimental environment you chose. Remember, I told you that you would be requiring an electrician if you haven't played doctor with your switchboard? This is one of those times where an electrician will be very helpful.

You're not being asked to take help from the electrician just because you will be working with high voltages, but also because if this is your first time messing with electrical circuits, you would want to know what wire goes where.

We are going to use the relay to switch on and off a light (bulb or tube light) in your house, just like how we turned on and off an LED in Module 1 *Chapter 2*, *Digital Ruler*. Firstly, power off your Arduino. Keep in mind that since this is just a testing phase, this will not be a permanent connection. We are doing this before adding the communication network so that if we come across a bug, it will not be because of the relay connection. Now, with the electrician's help (unless you know exactly what you are doing), build the following circuit with the two-way switch working as a replacement for a simple switch on your switchboard:



When you are completely sure that the circuit you have created is as described in the preceding image, connect the Arduino with the same program, which we loaded in the previous subsection, to the USB hub or the 9V 2A power adapter, and power it on. What do you see? The light goes on for three seconds and goes off for three seconds. It is literally a larger LED that goes on and off, controlled by the Arduino.

If this did not work, check the circuit again. Make sure that the connection to the two-way switch is done properly. Also, make sure that the relay is working as it is supposed to. If it still does not work, try unplugging the Arduino from the power supply, remove the CC3000 shield, and then connect it to the relay.

If it worked, isn't that awesome? You just created a setup where an Arduino can control a switch in your house. Next, we will learn how to communicate with the Arduino and, in turn, the relay using the Wi-Fi network.

Communicating through a terminal

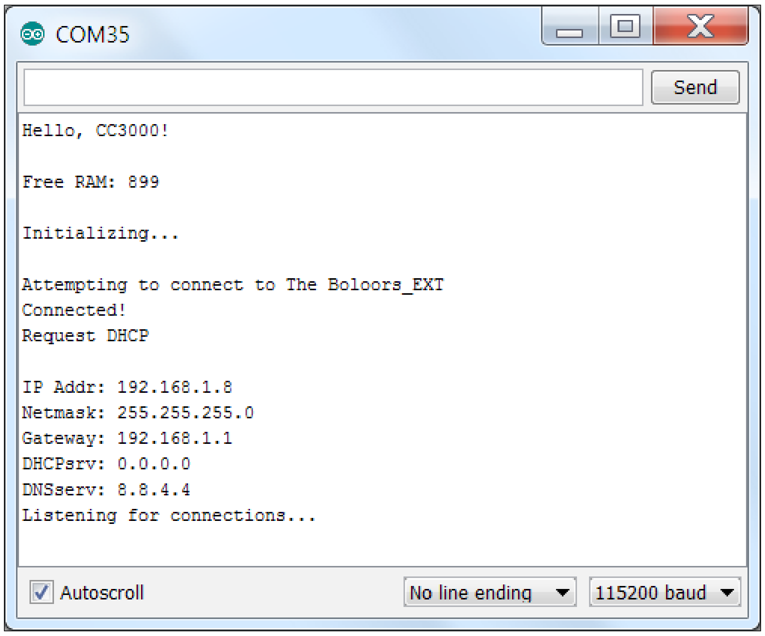
There is one more dilemma that we have to solve in order to have complete control over the electronic appliance. If we create a digital ON and OFF switch that when set to ON sends a turn ON signal to the relay, the problem is that if the physical switch is already ON, the ON signal of the digital switch will end up actually turning OFF the appliance.

"If the appliance is already ON, why would I send a turn ON signal?" Okay, think about this. Both the switch and the relay are ON, but the bulb would actually be off if you look at the two-way switch and the relay image previously shown. Now, if you want to turn ON the bulb through the relay, you actually have to send an OFF signal.

Pretty confusing, isn't it? To solve this, we will use a flag or a status variable that stores the current state of the relay. This will help us solve logical issues such as the one stated before.

Make sure your router is working as expected. Then, plug in the Arduino into the USB port and open the `home_automation_simple.ino` file that came with this chapter. As before, change `myNetwork` and `myPassword` to the ones corresponding to your router. There is also one major change that is made in this code. We have moved the relay (IN1) pin from D12 to A0 and modified the code accordingly. Repeated testing has proven that the setup is much more stable when it uses the analog port, because it consumes a lot less power. Change D12 → IN1 to A0 → IN1 before proceeding.

Finally, upload that program to the UNO. Once it is uploaded, open up the Serial Monitor. After some time, your Serial Monitor should look something like this:



```
Hello, CC3000!

Free RAM: 899

Initializing...

Attempting to connect to The Boloors_EXT
Connected!
Request DHCP

IP Addr: 192.168.1.8
Netmask: 255.255.255.0
Gateway: 192.168.1.1
DHCPsrv: 0.0.0.0
DNSserv: 8.8.4.4
Listening for connections...
```

Since we are using static IP, connecting to the network will be pretty quick. If it did not connect, try getting the Arduino closer to the Wi-Fi router, run it again, or try powering it through the 9V 2A adapter.

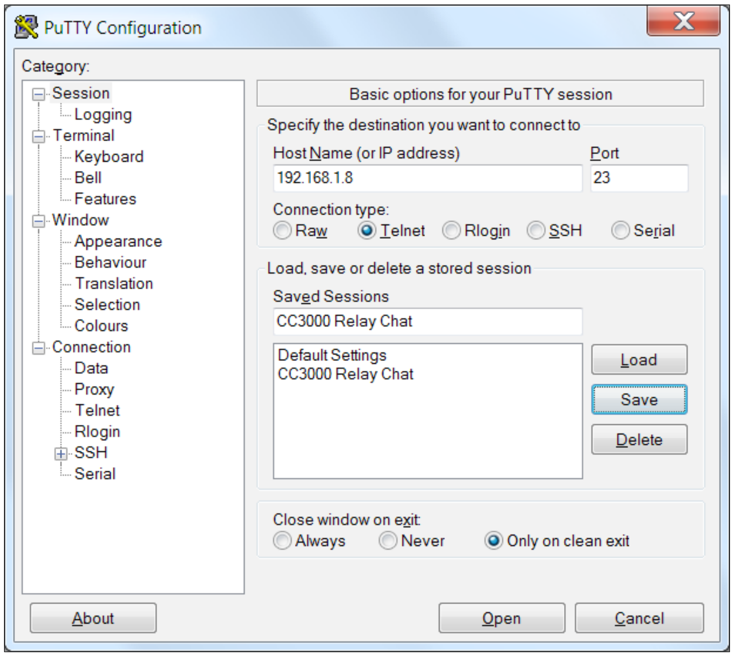
Remember the static IP address that you have set for your CC3000. We will need that address pretty soon.

The title of this subsection says *Communicating through a terminal*. So, we need to use a terminal. In this chapter, we will use Putty. Download `putty.exe` from `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

You could also just copy it from the `Files` folder of this chapter.

It is directly executable so you can place it on a desktop or someplace where you can access it easily. Once you have downloaded it, launch it.

Change the **Connection type** to **Telnet**. The **Port** should have automatically changed to `23`. If not, change it to `23`. Change the **Host Name** to the one that we have on the Arduino Serial Monitor. It may not be `192.168.1.8` in your case. Type `CC3000 Relay Chat` in the **Saved Sessions** field and click on **Save**, as shown in the following screenshot:



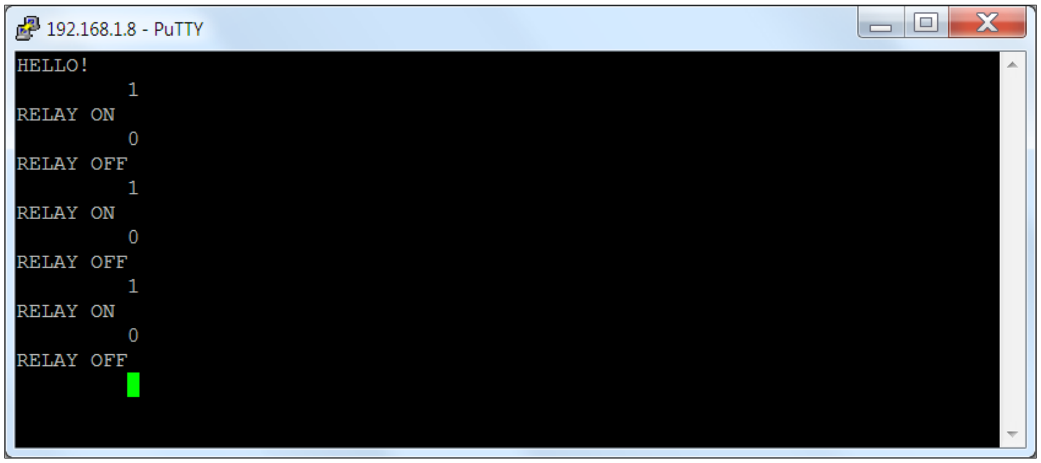
Host Name (or IP address): 192.168.1.8
Port: 23
Connection type: Raw [✓ Telnet] Rlogin SSH Serial
Saved Sessions: CC3000 Relay Chat
Default Settings
CC3000 Relay Chat
Close window on exit: Always Never [✓ Only on clean exit]

Now, instead of typing these details over and over again, you can simply load the saved configuration. Make sure your computer is also connected to the same network. Then, select **Open**.

Here are the commands you can use:

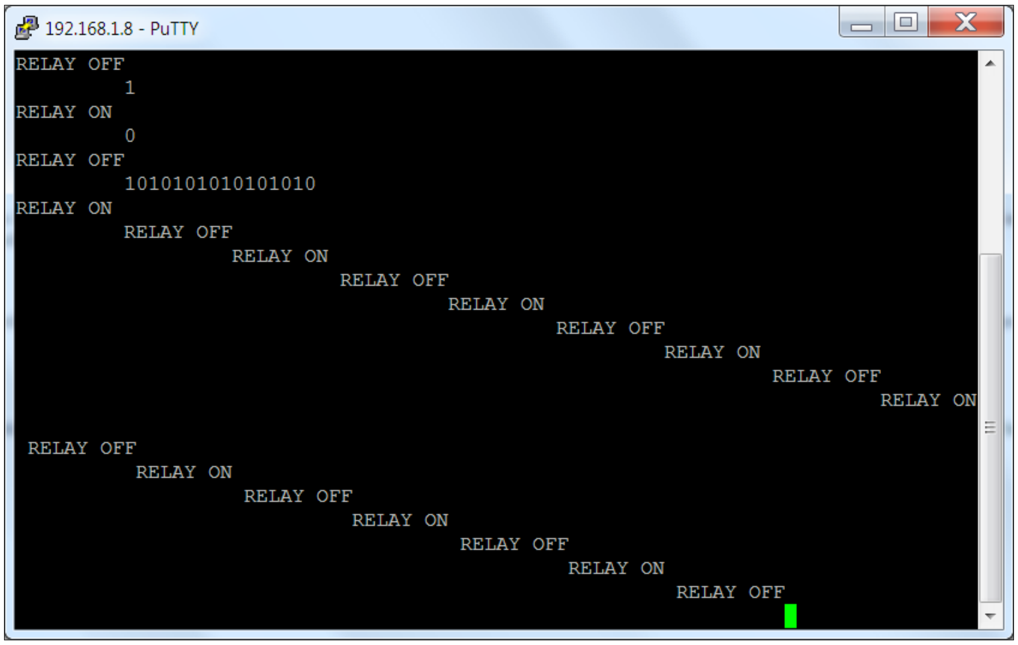
- `1` – Relay ON
- `0` – Relay OFF
- `q` – quit

You need to hit *Enter* after each command, for it to be sent. The window will look like this after a couple of tries:


192.168.1.8 - PuTTY

```
HELLO!
         1
RELAY ON
         0
RELAY OFF
         1
RELAY ON
         0
RELAY OFF
         1
RELAY ON
         0
RELAY OFF
```

Do not mind the empty spaces between the left edge and the `1`s and `0`s. Your relay is also turning ON and OFF based on the input, right? Good! What was unprecedented (and a bit hilarious) when this code was written was that, if you input something like `1010101010101010` to the terminal, you would end up with this (the following image) and a rapid clicking noise:


192.168.1.8 - PuTTY

```
RELAY OFF
         1
RELAY ON
         0
RELAY OFF
         1010101010101010
RELAY ON
         RELAY OFF
                  RELAY ON
                           RELAY OFF
                                    RELAY ON
                                             RELAY OFF
                                                      RELAY ON
                                                               RELAY OFF
                                                                        RELAY ON

 RELAY OFF
          RELAY ON
                   RELAY OFF
                            RELAY ON
                                     RELAY OFF
                                              RELAY ON
                                                       RELAY OFF
```

It is highly recommended to avoid this, because it might be a bit too much for the Arduino to handle. **Do not do this when the relay is connected to an electric appliance**. It could damage both the appliance and the UNO.

Press `q` to safely close the connection. Speaking of which, go ahead and create the previous circuit, which you created, that connects the Arduino to the bulb (or tube light) and try this code. Again, it is not a bad idea to get help from your electrician while making the connections. Open the terminal and try turning your bulb on. Works beautifully, doesn't it?

"I can only turn one appliance ON and OFF!"

"How do I control my home, using my smart phone?"

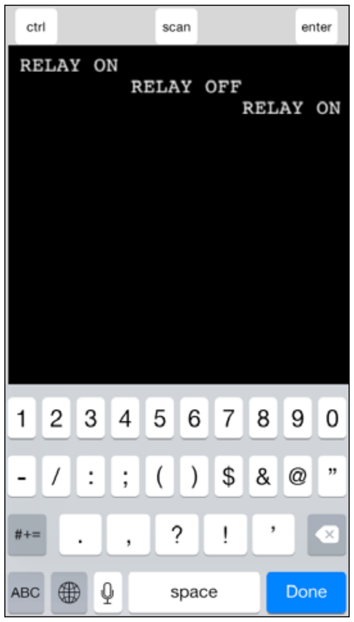
"You also spoke about speech recognition! Where is that?"

Do not worry. We will deal with all of these in the next sections.

However, there is a cheap trick you can use to control the bulb, using your smart phone. If you are using iOS, download an app called Telnet Lite from the App Store; for Android users, download Simple Telnet Client from the Google Play Store. You can use any Telnet app for this task, but these are the two that have been tested.

Configuring the app is similar to that of Putty and is self-explanatory. Reset the Arduino by pressing its reset button. Use the power adapter to power it this time, and disconnect it from the USB port. Give it sufficient time to connect to the router. Then, launch the app.

So, what are you waiting for? Connect to the corresponding IP address and press `1` or `0`!



```
RELAY ON
        RELAY OFF
                RELAY ON
```

You will see a screen resembling that in the preceding image on your Android device.

The great thing about the chat server program is that you can control the appliance with both the terminals running on your computer and smart phone at the same time. Go ahead and try it!

Summary

We will be ending this chapter with that. There is a lot more to do, which will be discussed in part 2—the next chapter. What have we learned in this chapter? We have learned a lot about relays, from how they work to how to work with them. We also took considerable time to fully gauge how the relay functions in conjunction with a simple and two-way switch. We also understood why a two-way switch is better for control. We also programmed the Arduino to control a light's switch and built a circuit to achieve the same, with the help of an electrician (or not). Finally, we took it one step higher by adding a communication layer (via terminal).

The next thing on the to-do list is to create a communication layer that allows communication through a smart phone (without a terminal) and also using speech recognition. To spread our wings a little, we will also learn to program more than one appliance, which means only one thing – more relays!

7
Home Automation – Part 2

In the previous chapter, we got to know the basics of home automation; the key element being the relay that serves as a switch for an electrical appliance that can be controlled using Arduino. We also created a communication network via terminal, and this in turn allowed us to control the appliance by using a terminal on a computer or on a smart phone.

In this chapter, we will be taking this idea further by adding additional means of communication to Arduino, namely, a smart phone interface (not terminal) and a speech recognition system. After that, we will take another step higher to increase the number of appliances the setup can control, which would make this whole idea much more practical. Finally, we will discuss how this idea can be expanded to cover more area.

In short, these are the topics we will be covering in this chapter:

- Communicating via a smart phone
- Implementing speech recognition
- Upgrading the home automation system

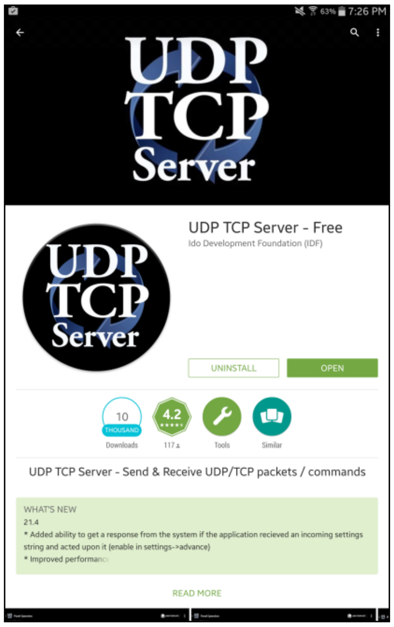
Let us continue, starting with communication using a smart phone.

Communicating via a smart phone

Unfortunately, creating an app for this purpose is too complicated to be within the scope of this book. However, we will be using apps that are already out there to communicate with our so far mini home automation system. The following subsections will deal with Android and iOS devices, not necessarily just smart phones. Yes, that means you can even use your tablet to control the appliances.

Android devices

Open the Google Play Store on your Android device, and look for UDP TCP Server:

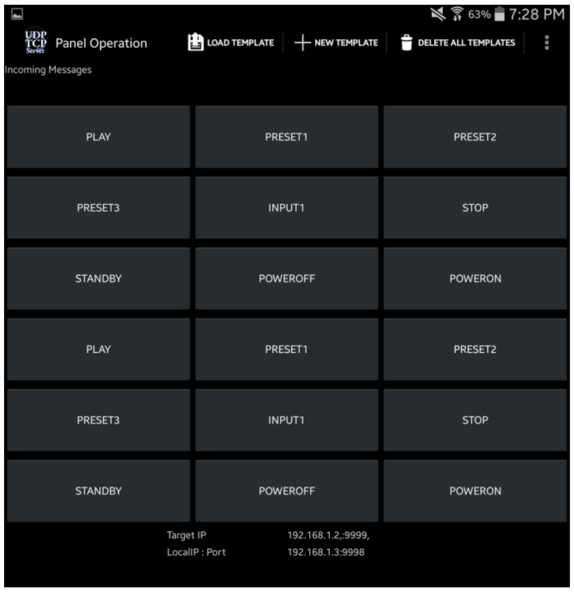


UDP TCP Server - Free
Ido Development Foundation (IDF)

UDP TCP Server - Send & Receive UDP/TCP packets / commands

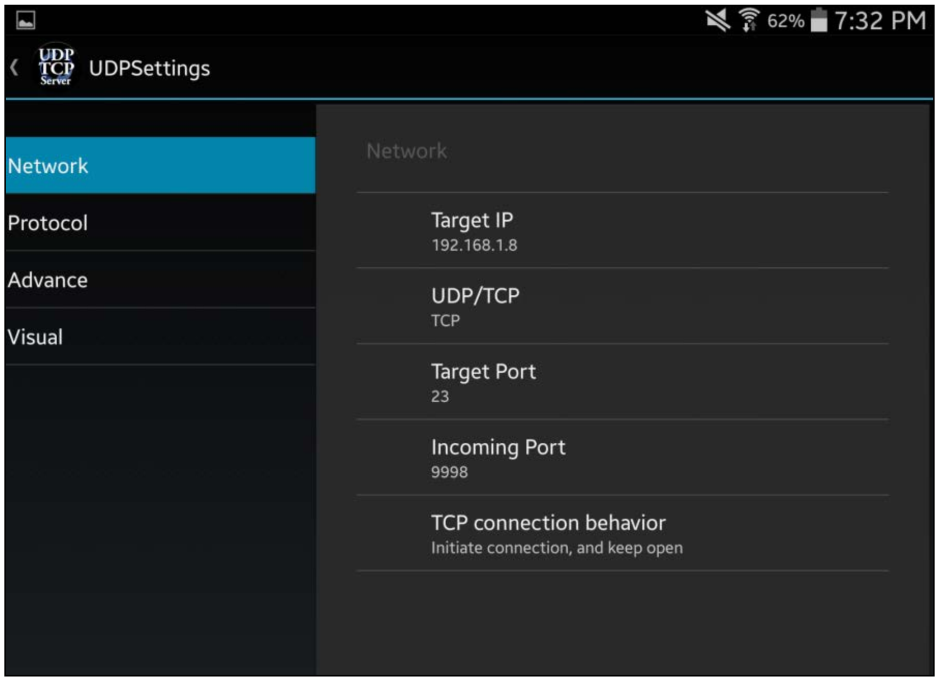
WHAT'S NEW
21.4
* Added ability to get a response from the system if the application recieved an incoming settings string and acted upon it (enable in settings->advance)
* Improved performanc

Download and install it. On opening it, you will see the following page:



Target IP 192.168.1.2,:9999,
LocalIP : Port 192.168.1.3:9998

Go to **UDPSettings**. This is reached by first selecting the three vertical squares icon on the top-right. Change the settings to reflect the following image:



Target IP
192.168.1.8

UDP/TCP
TCP

Target Port
23

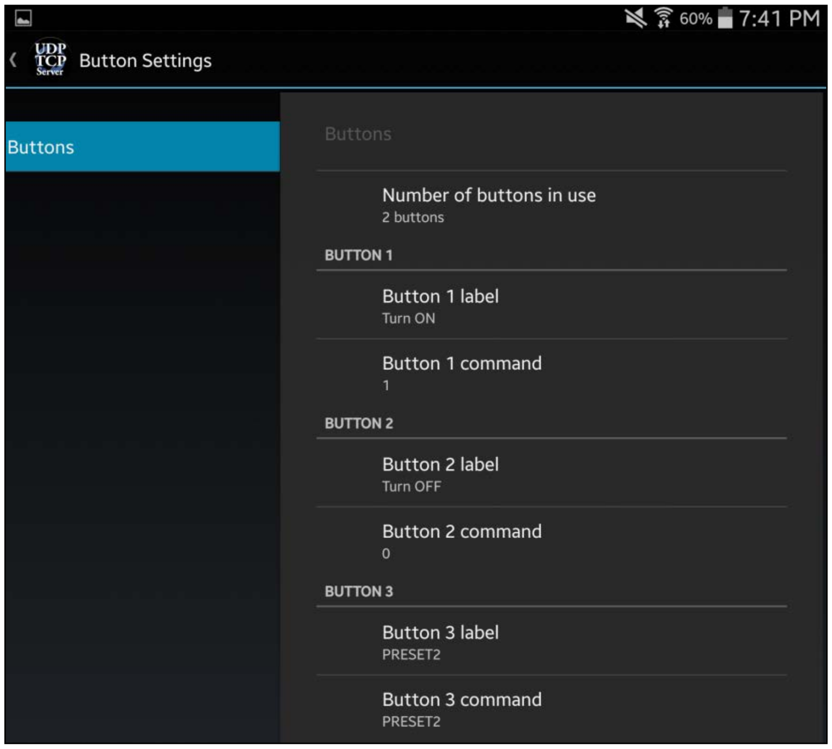
Incoming Port
9998

TCP connection behavior
Initiate connection, and keep open

Do not forget to use your IP address, which may differ from the one in the preceding image. Go back to the main page and select **+ New Template**. Call it something like `Home Automation`.

Just as you did for accessing settings, access the **Button Settings**. We are now going to redo the main page. We will get rid of all the useless buttons and keep only two.

One button will be to turn ON the light (relay), and another to turn it OFF. Use the following parameters:



Buttons

Number of buttons in use
2 buttons

BUTTON 1

Button 1 label
Turn ON

Button 1 command
1

BUTTON 2

Button 2 label
Turn OFF

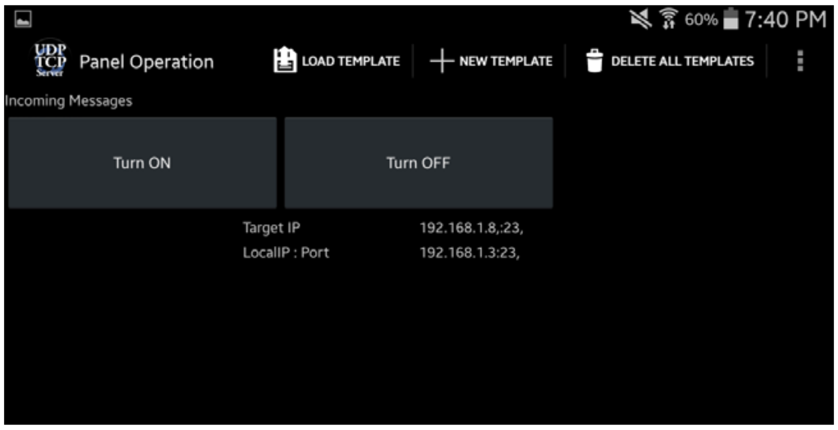
Button 2 command
0

BUTTON 3

Button 3 label
PRESET2

Button 3 command
PRESET2

Go back to the main page and you will have something that resembles the following image:



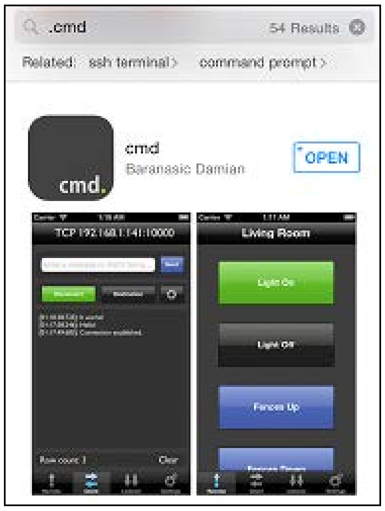
Target IP 192.168.1.8,:23,
LocalIP : Port 192.168.1.3:23,

Restart the app and make sure the Arduino is connected to the router. Then, go on and try pressing the two buttons. Works blissfully, right?

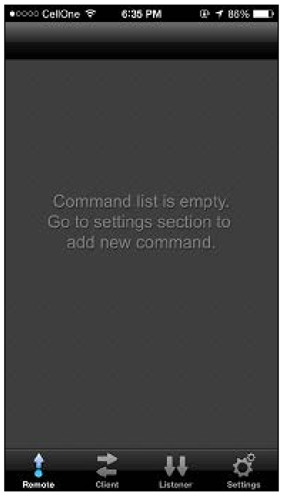
If you want to try out other apps, just look for TCP on the Play Store and try them, now that you know how to configure them.

iOS (Apple) devices

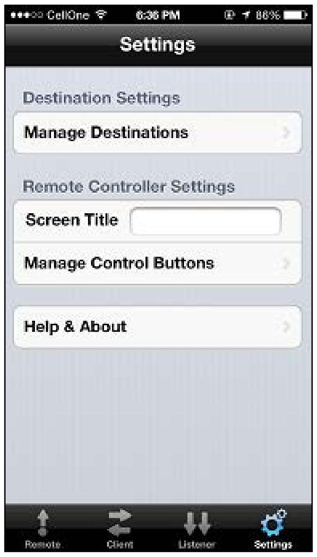
Go to the App Store and look for an app called .cmd:



Download it. Then, open it:

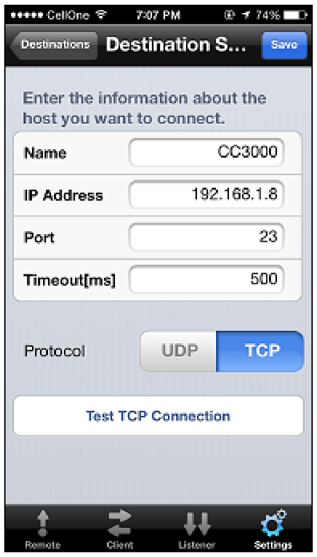

CellOne
6:35 PM
88%
Command list is empty. Go to settings section to add new command.
Remote
Client
Listener
Settings

Go to **Settings**, which is the bottom-right icon:



Tap on **Manage Destinations** and on the top-right icon. Select **Add New Connection**.

Then, fill the page with the following details:



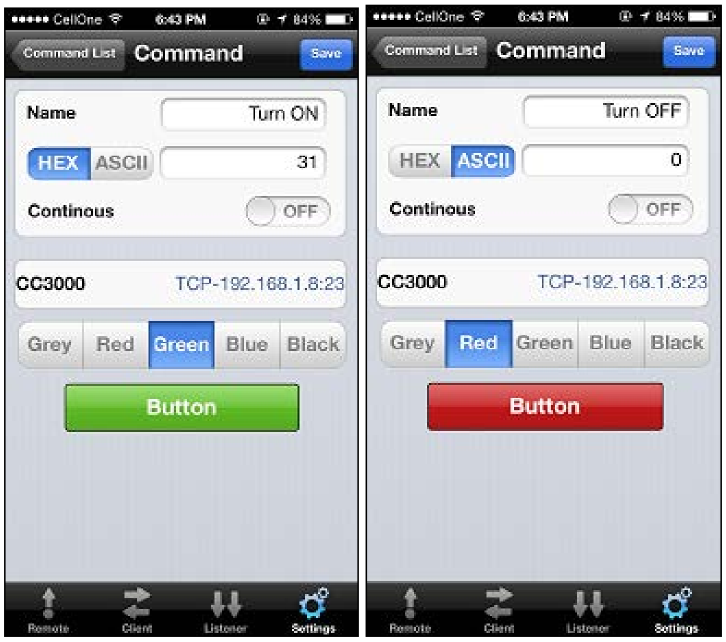
Enter the information about the host you want to connect.

Name	CC3000
IP Address	192.168.1.8
Port	23
Timeout[ms]	500
Protocol	UDP [✓ TCP]

Test TCP Connection

As before, remember to use your Arduino IP, which may not be the same as in the preceding image. Then, tap on **Save**. Go back to **Settings**. For the screen title, type something like `Home Automation`. Then, press **Manage Control Buttons**.

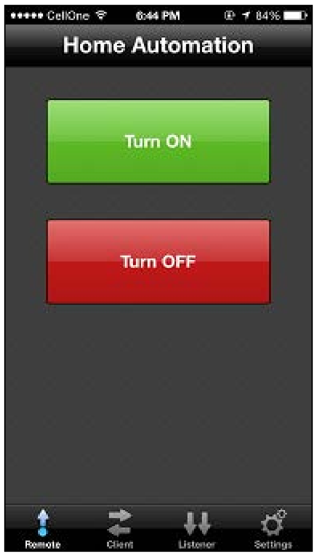
As you did for creating a new destination, click on the top-right icon and select **Add new command**. You have to do this twice to add two buttons (one for **ON** and another for **OFF**), as shown in the following screenshot:



Name: Turn ON
[✓ HEX] ASCII: 31
Continous: OFF
CC3000 TCP-192.168.1.8:23
Grey Red [✓ Green] Blue Black

Name: Turn OFF
HEX [✓ ASCII]: 0
Continous: OFF
CC3000 TCP-192.168.1.8:23
Grey [✓ Red] Green Blue Black

For the connection, tap the **Assign Selection** option and use the connection that we just made. Do not forget to save the button each time. Then, select the **Remote** icon on the bottom-left. You will have something like the following:



Once you are sure that the Arduino is connected to the router, go ahead and play with the switches. They should work just as expected. If it didn't work in the first try, try again. For some reason, the app is shy sometimes to work the first time.

However, if it worked the first time, it's remarkable, isn't it? This means you didn't have to break a sweat—the same code for Arduino worked for all the cases!

Implementing speech recognition

This section deals with using existing, powerful speech recognition tools that will aid us in adding a layer of verbal communication with the Arduino.

The software

There are tons of speech recognition softwares that can be used for this project. Some are better than the others. The software we will be using for this project is called BitVoicer. Mind you, the software is not free, but it isn't very expensive. However, after using it, you will understand why this software was chosen. You will also need a good microphone for this project. Most laptops come with inbuilt microphones. You could also use your phone's earphones if they came with a microphone.

Go to `http://www.bitsophia.com/BitVoicer.aspx` and purchase the program. Download and install it. If the download is a bit slow, be patient and let it finish downloading. Then, run the program as an administrator.

Configuring the software

The default language that is used in BitVoicer is English (US). Even if English is your primary language, go through the following step:

Go to **File** → **Additional Languages…**. A new window will open, showing other languages that can be recognized by the BitVoicer software. If it showed you an error message saying 'BitVoicer requires elevated privileges,' restart the program as an administrator.

You will notice that sometimes there are different varieties of the same language. This refers to the different nature of the language spoken and written in different parts of the world. For example, UK English is quite different from US English. There is also a change in accent between the two regions. So, pick a region that best suits your language and accent. Then, click on **Install**.

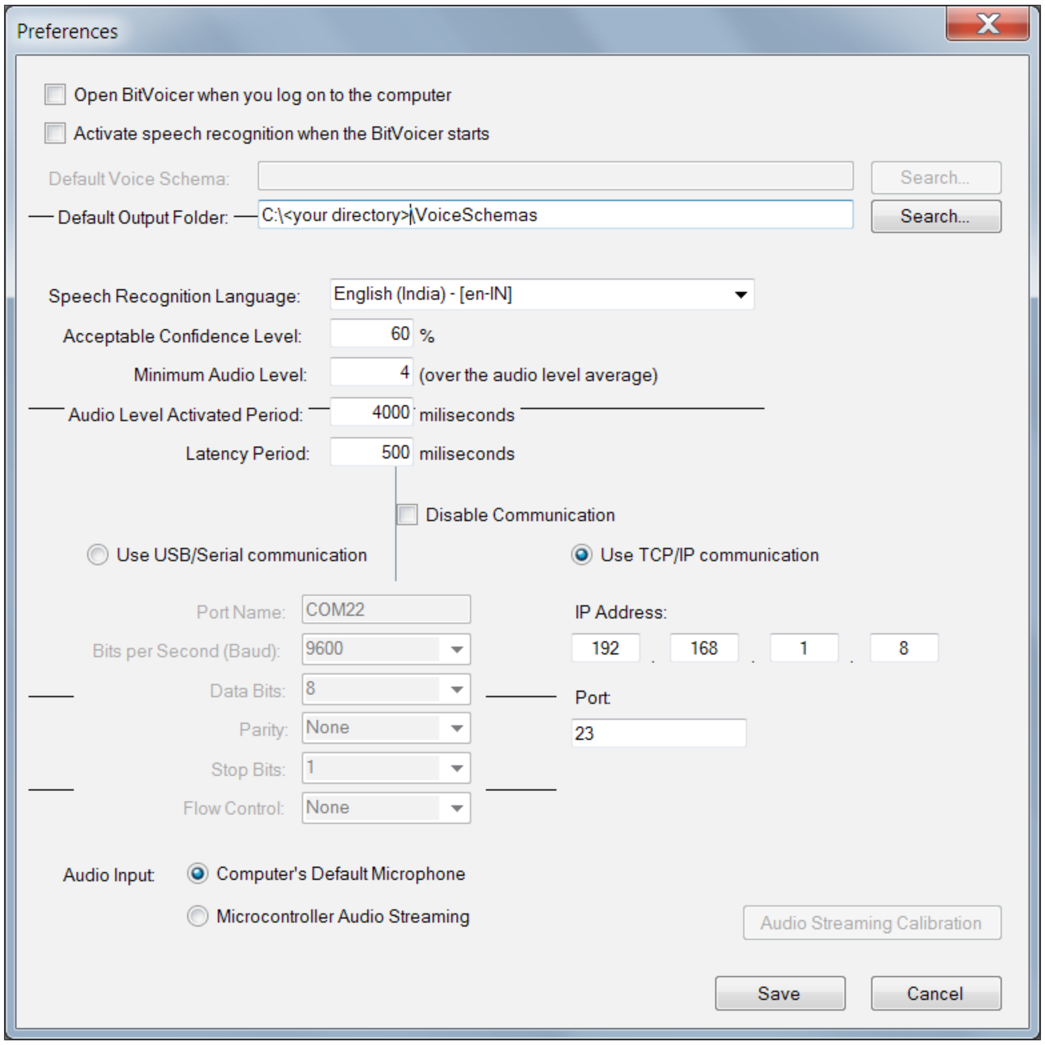
Now, open **Preferences** via **File** → **Preferences**.

What are schemas? schema is a file that is used by BitVoicer. This is similar to `.ino` associated with Arduino.

Move the output folder to some place that you will remember. You can put it in a new folder named `VoiceSchemas` in your Arduino directory.

Change the language to the one that you had installed. If you didn't, leave the default one as it is. For this project, we will be using TCP/IP communication.

Change the settings on the **Preferences** window to match the following image:


Preferences

[ ] Open BitVoicer when you log on to the computer

[ ] Activate speech recognition when the BitVoicer starts

Default Voice Schema: Search...

Default Output Folder: C:\<your directory>\VoiceSchemas Search...

Speech Recognition Language: English (India) - [en-IN]

Acceptable Confidence Level: 60 %

Minimum Audio Level: 4 (over the audio level average)

Audio Level Activated Period: 4000 miliseconds

Latency Period: 500 miliseconds

[ ] Disable Communication

○ Use USB/Serial communication [● Use TCP/IP communication]

Port Name: COM22

Bits per Second (Baud): 9600

Data Bits: 8

Parity: None

Stop Bits: 1

Flow Control: None

IP Address: 192 . 168 . 1 . 8

Port: 23

Audio Input: [● Computer's Default Microphone] ○ Microcontroller Audio Streaming

The only things that will be different for you are the IP address that should match the IP address of the Arduino, the language if you chose a different one, and the default output folder.

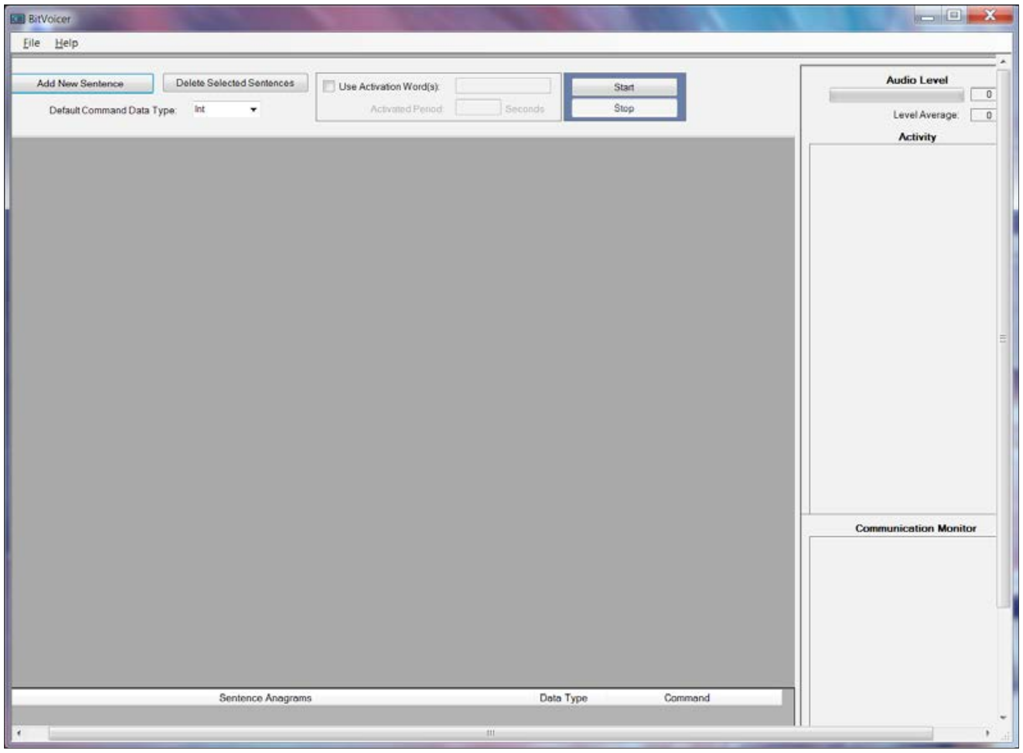
The key parameters that you need to take a note of are as follows:

- **Acceptable confidence level**: Setting this to 100 percent will require a perfect pronunciation of the programmed text. Since we have set this to 60 percent, the software will take it easy on us and understand us better, even with tiny errors in the pronunciation.
- **Minimum audio level**: This sets a threshold for how loud you will have to speak for the program to begin recording you. You might have to change this number later on, depending on your microphone.

Now, hit **Save**.

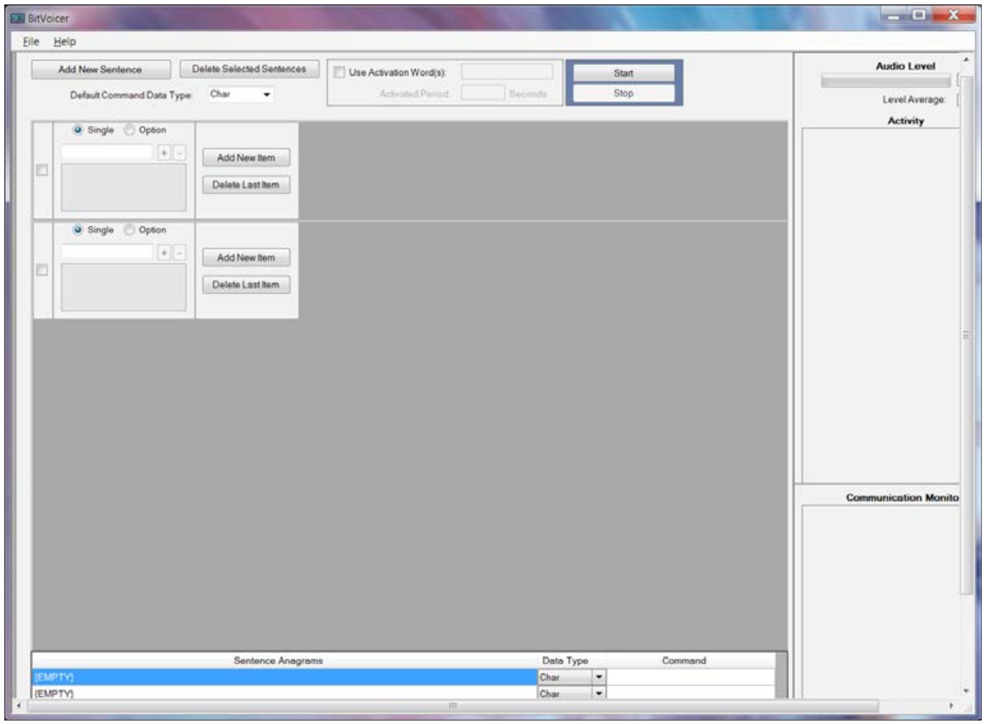
Creating a voice schema

1. Go to **File** → **New** and you will see the following window:



2. Change **Default Command Data Type** to **Char**.
3. Click on **Add New Sentence** two times to create two new programmable sentences.

   Your window will look like the following:

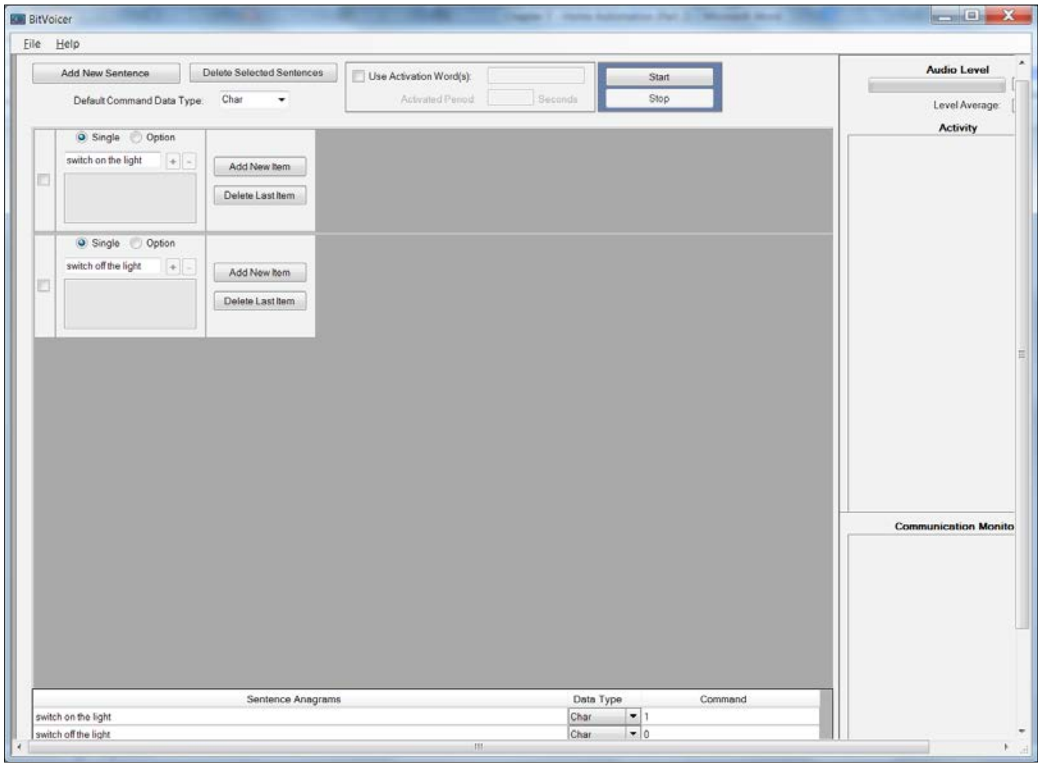


Default Command Data Type: Char

Sentence Anagrams	Data Type	Command
{EMPTY}	Char	
{EMPTY}	Char	

4. In the first empty sentence field, type `switch on the light`.
5. In the second empty sentence field, type `switch off the light`.
6. In the empty **Command** field at the very bottom (scroll if you cannot see it), which is corresponding to the first sentence, type `1` (without quotations).
7. In the one below this, type `0`.

We will add more commands later. For now, your window will resemble the following image:



Default Command Data Type: Char

switch on the light

switch off the light

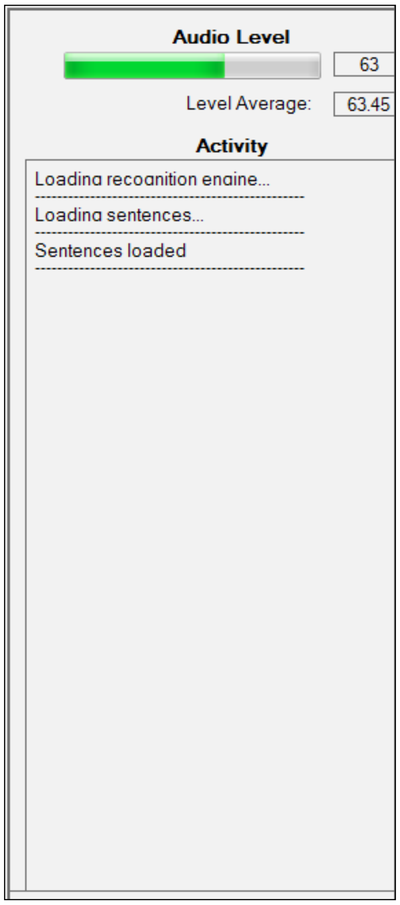
Sentence Anagrams	Data Type	Command
switch on the light	Char	1
switch off the light	Char	0

Check everything and make sure everything is in order. Then, go back to your Arduino. Use the same code that we used previously to control the relay through a terminal. Power the Arduino via the power adapter, and wait for it to connect to your network. Again, make sure that your computer is on the same Wi-Fi network.

Testing out the software

When everything is set, press the **Start** button in BitVoicer.

Scroll right and you will see the **Activity** notification box. If everything worked correctly, you will see this in the activity box:

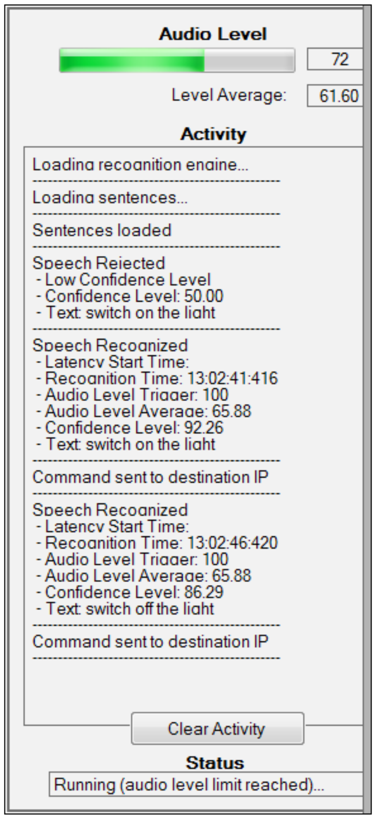


```
Loading recognition engine...
---------------------------------------------
Loading sentences...
---------------------------------------------
Sentences loaded
---------------------------------------------
```

If you do not see any green in the **Audio Level** bar, your microphone isn't working; so, go back and check its connections.

Now try it. Use your beautiful voice to say *switch on the light*. If it is not loud enough, try again. If you hear a click from the relay, you know that it worked! Now, say *switch off the light* and you will hear another click.

Here is how your activity monitor would look like:



```
Loading recognition engine...
--------------------------------
Loading sentences...
--------------------------------
Sentences loaded
--------------------------------
Speech Rejected
 - Low Confidence Level
 - Confidence Level: 50.00
 - Text: switch on the light
--------------------------------
Speech Recognized
 - Latency Start Time:
 - Recognition Time: 13:02:41:416
 - Audio Level Trigger: 100
 - Audio Level Average: 65.88
 - Confidence Level: 92.26
 - Text: switch on the light
--------------------------------
Command sent to destination IP
--------------------------------
Speech Recognized
 - Latency Start Time:
 - Recognition Time: 13:02:46:420
 - Audio Level Trigger: 100
 - Audio Level Average: 65.88
 - Confidence Level: 86.29
 - Text: switch off the light
--------------------------------
Command sent to destination IP
--------------------------------
```

```
Running (audio level limit reached)...
```

How cool is that? You used speech to control your Arduino. If you go ahead and complete the circuit with an actual appliance and try the same thing again (in case you had unplugged it from the relay like me), it will work in the same way as it did when you were controlling it via a terminal, except this time your speech is doing the job.

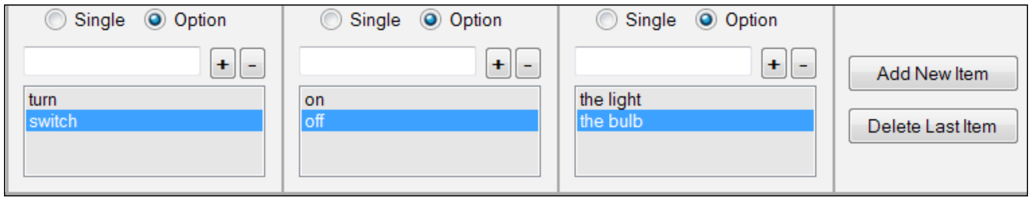
Notice how in the preceding image the speech was rejected the first time, because either the speech was inaudible or indistinguishable for the software to comprehend. It was basically not confident about what was said, so it did not send any information to the Arduino.

Making a more reliable schema

Now, we will edit the schema to allow the different ways to say switch on/off the light. Press **Stop** when done, and save this schema (`chat_server_simple.vsc` came with this chapter, in case you lost the saved file). Then, go to **File** → **New**.

The default command data type should now be **Char**. However, if it is still **Int**, change it to **Char**. Follow these steps (or just open `chat_server_simple2.vsc`):

1. Add one new sentence.
2. Change the radio button from **Single** to **Option**.
3. Type `turn` and click on the **+** sign.
4. Type `switch` and click on the **+** sign.
5. Click on **Add New Item**.
6. Change the radio button from **Single** to **Option**.
7. Type `on` and click on the **+** sign.
8. Type `off` and click on the **+** sign.
9. Click on **Add New Item**.
10. Type `the light` and click on the **+** sign.
11. Type `the bulb` and click on the **+** sign.
12. Check whether your sentence matches the following image:



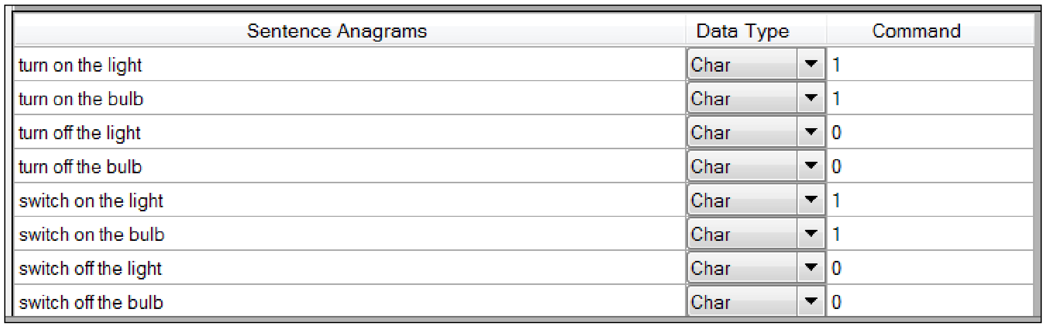
Single [✓ Option]: turn, switch

Single [✓ Option]: on, off

Single [✓ Option]: the light, the bulb

13. Scroll all the way down to **Sentence Anagrams**. Change all the data types to **Char**, if they are not set as **Char**.
14. Type `1` into the **Command** field for any sentence that pertains to turn ON the light/bulb.
15. Type `0` into the **Command** field for any sentence that pertains to turn OFF the light/bulb.

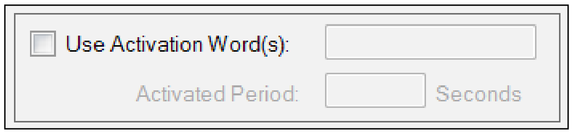
16. Check whether your commands match the following image:



Sentence Anagrams	Data Type	Command
turn on the light	Char	1
turn on the bulb	Char	1
turn off the light	Char	0
turn off the bulb	Char	0
switch on the light	Char	1
switch on the bulb	Char	1
switch off the light	Char	0
switch off the bulb	Char	0

Once this is done, again, make sure your Arduino circuit is complete and that the Arduino is connected to the Wi-Fi. Then, hit **Start**. Also, try all the new commands that you just created and watch them do wonders. Beautiful, isn't it?

One last aspect of the BitVoicer software is the activation word:



Use Activation Word(s):

Activated Period: Seconds

If you would like to use this, click on the check mark and type something that you want to use (Jarvis, Cortana, and so on); set **Activated Period** to however long you wish. Suppose you set **Activated Period** to 300 seconds (5 minutes), this would mean that you have to initially say something like <*activation word*> + *turn on the light*. However, for the next 300 seconds, you don't have to keep using the activation word. Learn to program some AI into it, and watch it blow up your home and free itself from its human masters. This is slightly beyond the scope of this book.

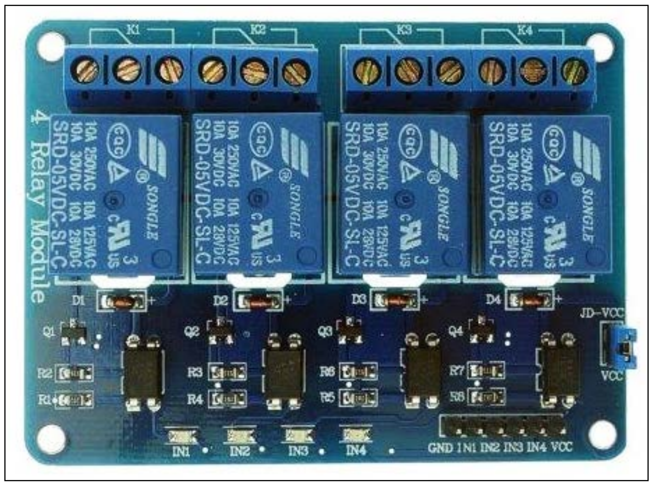
We have finally finished implementing the communications network. The only thing left to do is expand this to control more devices. This would involve using more relays and tweaking the codes/app layouts/schemas to correspond to it.

Upgrading the home automation system

Now that we understand how each means of communication works independently, we will begin controlling more than one device. Let's say, for the sake of this chapter, we are going to control the light and fan in your room, and the light and television in the living room. We are going to use only four home appliances for this section. Once you understand how it works and what we changed from using just one appliance, you will be powerful enough to add more appliances to your home automation system.

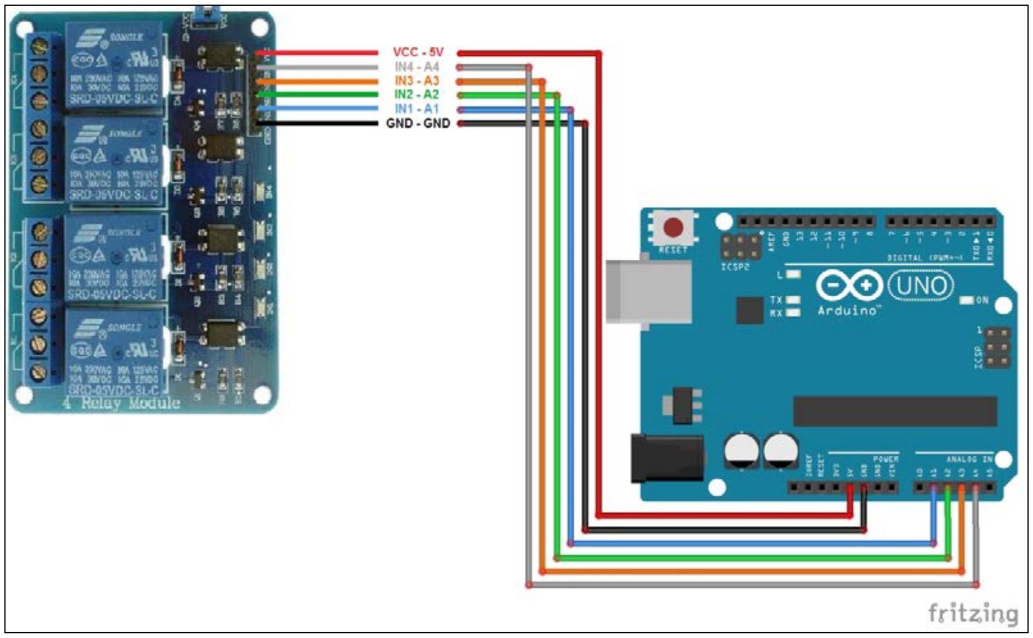
Controlling multiple appliances

Controlling four appliances implies that we need four relays. Instead of getting four single 5V relays, buy yourself a 4-channel 5V Arduino-compatible relay as shown in the following image:



Since you have already learned how relays work, there is nothing new here, except that you will be using just one GND pin and one VCC (VIN) pin, which is really convenient. Before creating the entire circuit, let's just connect the Arduino to the 4-channel relay, and ensure that they work and can be controlled independently.

So, create the following circuit:



The connections from the Arduino to the 4-channel relay are as follows:

- GND → GND
- 5V → VCC
- A1 → IN1 (your room's light)
- A2 → IN2 (your room's fan)
- A3 → IN3 (living room's light)
- A4 → IN4 (living room's television)

The parentheses represent where each relay is eventually going to be connected. These can, however, be connected to whatever is convenient in your case.

Launch Arduino IDE, connect the Arduino, and open the `home_automation_complete.ino` file that came with this chapter. Upload that code to your Arduino.

The fundamental difference between this code and the previous one is the use of a `switch` function. It is a more condensed form of the multiple `if` statements, which you are familiar with by now. In the program `home_automation_simple`, we are going to switch the following `if` statements to `switch` statements:

```
if((ch=='1')||(ch=='0')||(ch=='q'))
       {
         //Serial.println(char(ch));
         if(ch == '1')
         {
           analogWrite(A0, 255);
           //digitalWrite(12, HIGH);
           chatServer.write("RELAY ON \n");
           delay(100);
           relay_status = 1;
         }
         else if(ch == '0')
         {
           analogWrite(A0, 0);
           //digitalWrite(12, LOW);
           chatServer.write("RELAY OFF\n");
           delay(100);
           relay_status = 0;
         }
         else if(ch == 'q')
         {
           //Serial.println(F("\n\nClosing the connection"));
           chatServer.write("DISCONNECTED");
           cc3000.disconnect();
         }
       }
```

Following is the new switch statement:

```
       switch(ch)
       {
         case '1':
           analogWrite(light_room, 255);
           chatServer.write("ROOM LIGHT ON\n");
           delay(500);
           break;
         case '2':
         {
```

```
  analogWrite(light_room, 0);
  chatServer.write("ROOM LIGHT OFF\n");
  delay(500);
  break;
}
case '3':
{
  analogWrite(fan_room, 255);
  chatServer.write("ROOM FAN ON\n");
  delay(500);
  break;
}
case '4':
{
  analogWrite(fan_room, 0);
  chatServer.write("ROOM FAN OFF\n");
  delay(500);
  break;
}
case '5':
{
  analogWrite(light_main, 255);
  chatServer.write("MAIN LIGHT ON\n");
  delay(500);
  break;
}
case '6':
{
  analogWrite(light_main, 0);
  chatServer.write("MAIN LIGHT OFF\n");
  delay(500);
  break;
}
case '7':
{
  analogWrite(tv_main, 255);
  chatServer.write("TV ON\n");
  delay(500);
  break;
}
case '8':
{
```

```
              analogWrite(tv_main, 0);
              chatServer.write("TV OFF\n");
              delay(500);
              break;
            }
            case 'q':
            {
              chatServer.write("DISCONNECTED");
              cc3000.disconnect();
              break;
            }
            default:
              break;
          }
```

Can you see how it is more condensed and neater? Of course, we could have used more of the `if` statements, but with this code, it is much easier for you to add another application as follows:

```
            case 'a':
            {
              analogWrite(something, 255);
              chatServer.write("something ON\n");
              delay(500);
              break;
            }
            case 'b':
            {
              analogWrite(something, 0);
              chatServer.write("something OFF\n");
              delay(500);
              break;
            }
```

Yes, that's right, you don't have to stick to just numbers.

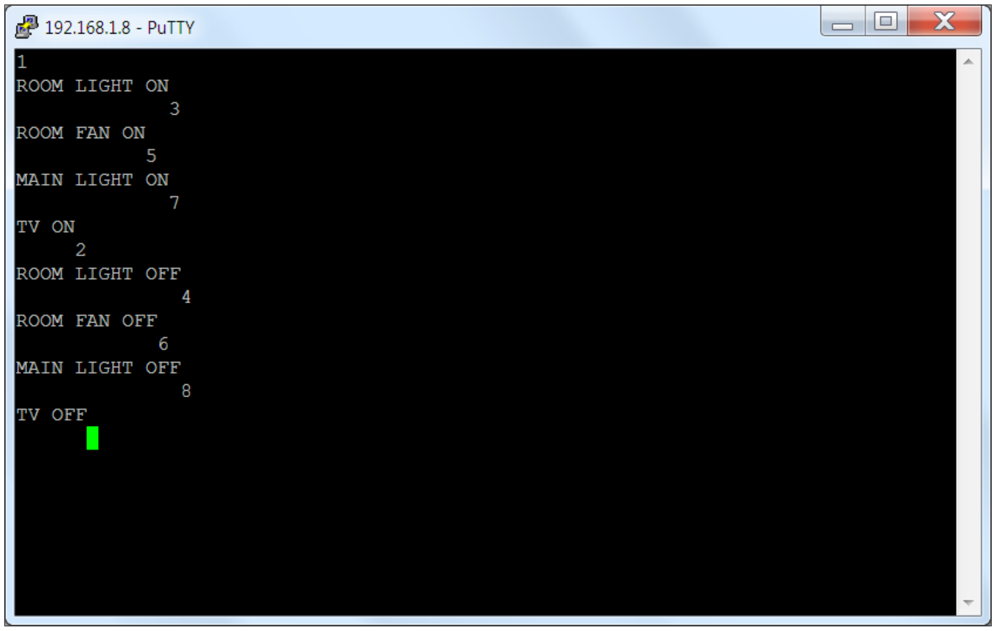
Via the terminal

Now, launch Putty. The only thing that is different is the commands, and they are as follows:

- `1`: Light in your room ON
- `2`: Light in your room OFF

- `3`: Fan in your room ON
- `4`: Fan in your room OFF
- `5`: Light in the living room ON
- `6`: Light in the living room OFF
- `7`: Television ON
- `8`: Television OFF
- `q`: Disconnect

Go ahead and try it out. You'll find yourself with a screen like the following:

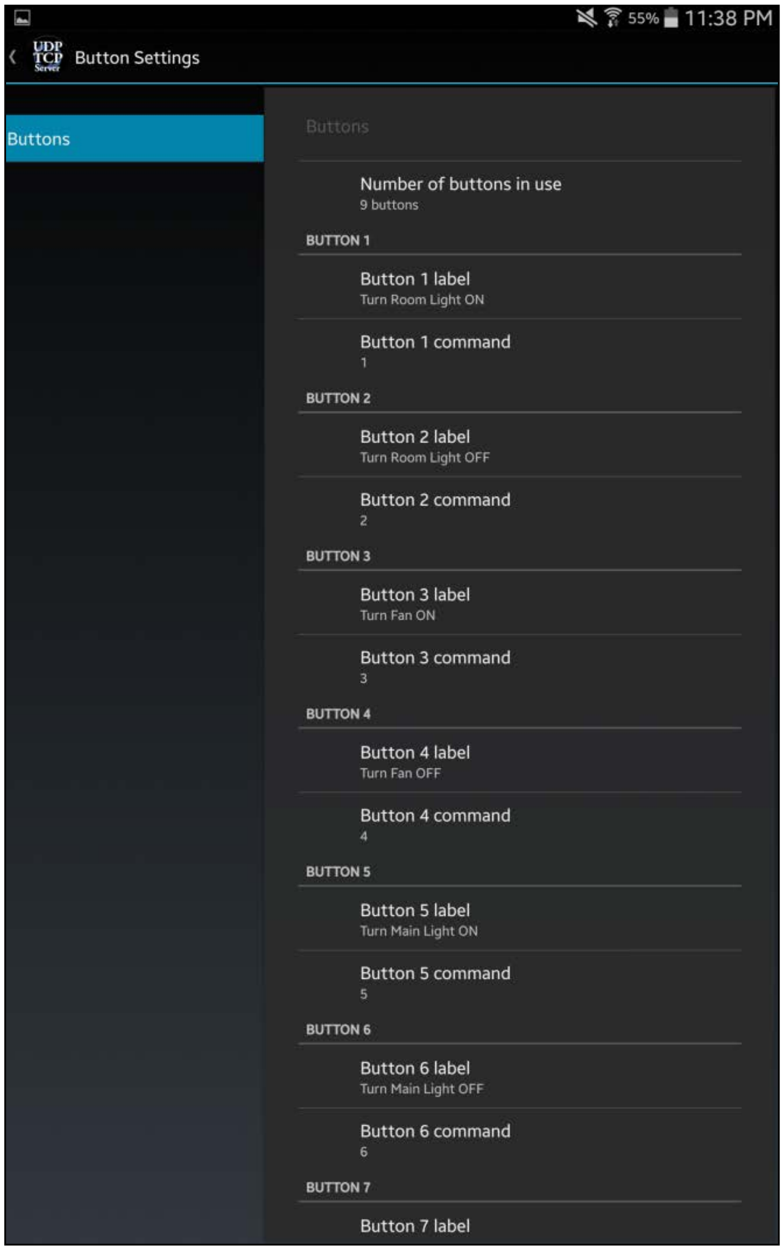


```
1
ROOM LIGHT ON
             3
ROOM FAN ON
           5
MAIN LIGHT ON
             7
TV ON
     2
ROOM LIGHT OFF
              4
ROOM FAN OFF
            6
MAIN LIGHT OFF
              8
TV OFF
```

The terminal will probably be the seldom-used mode of communication. Nevertheless, it is the best one for testing and debugging. Of course, now you can hear only the clicking sound of the relays when they go ON and OFF, but when they are finally connected to their electronic counterparts, they will still work the same.

Via the smart phone (Android)

Open the UDP TCP Server free app and create a new template by pressing the **+ NEW TEMPLATE** button. Then, go to **Button Settings** and change the parameters to match the following image:



Buttons

Number of buttons in use
9 buttons

BUTTON 1

Button 1 label
Turn Room Light ON

Button 1 command
1

BUTTON 2

Button 2 label
Turn Room Light OFF

Button 2 command
2

BUTTON 3

Button 3 label
Turn Fan ON

Button 3 command
3

BUTTON 4

Button 4 label
Turn Fan OFF

Button 4 command
4

BUTTON 5

Button 5 label
Turn Main Light ON

Button 5 command
5

BUTTON 6

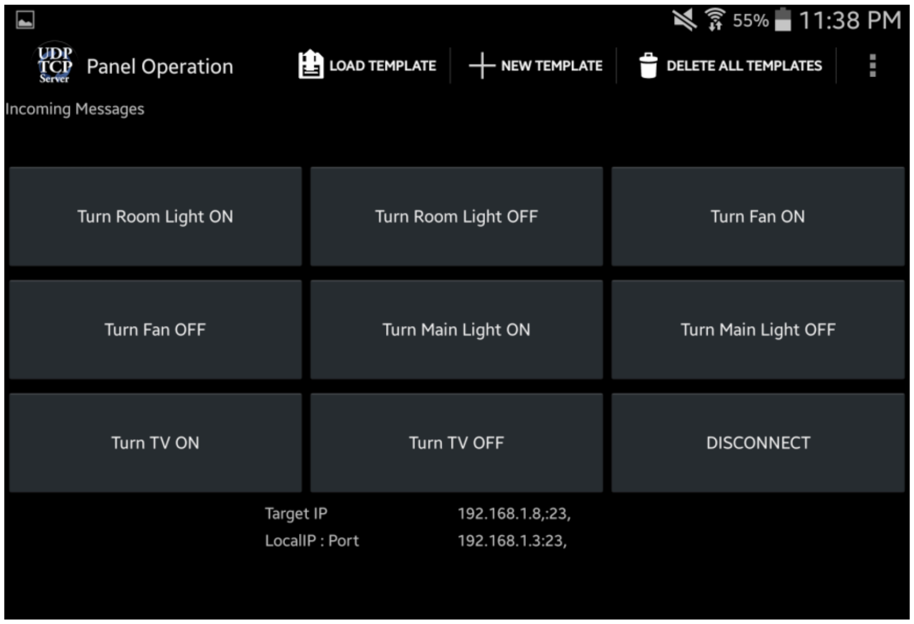
Button 6 label
Turn Main Light OFF

Button 6 command
6

BUTTON 7

Button 7 label

Similarly, you need to create buttons 7 and 8 (cut off in the preceding image). When you go back to the main page, it will look like the following:



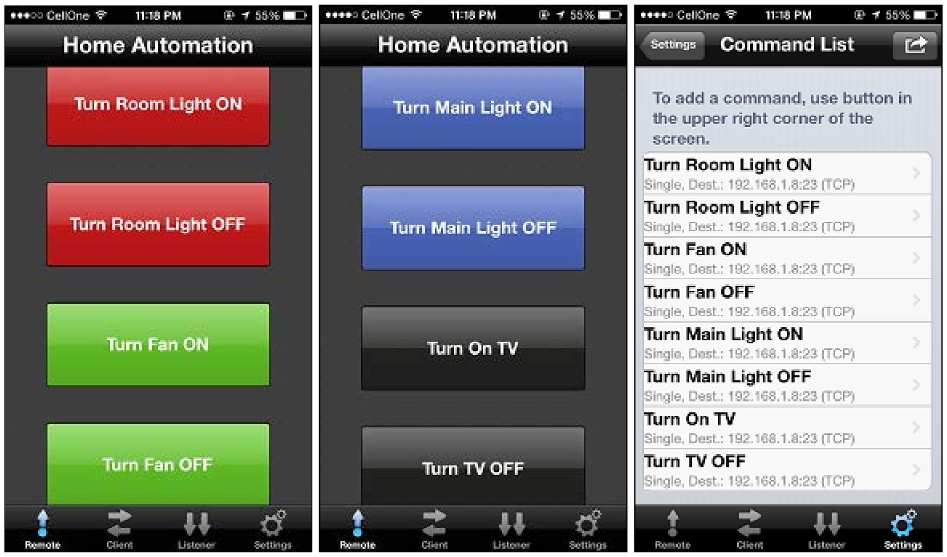
Target IP 192.168.1.8,:23,
LocalIP : Port 192.168.1.3:23,

Go ahead and try the buttons out. Make sure you set the right commands corresponding to the correct relay.

Via the smart phone (iOS)

You already know how to do this. Launch .cmd. You just have to add six more buttons with a total of eight (nine if you add a disconnect button). If you are reusing the single relay template, remember to change its OFF button's command from `0` to `2`.

Your main page and buttons page after making some changes will look like the following:



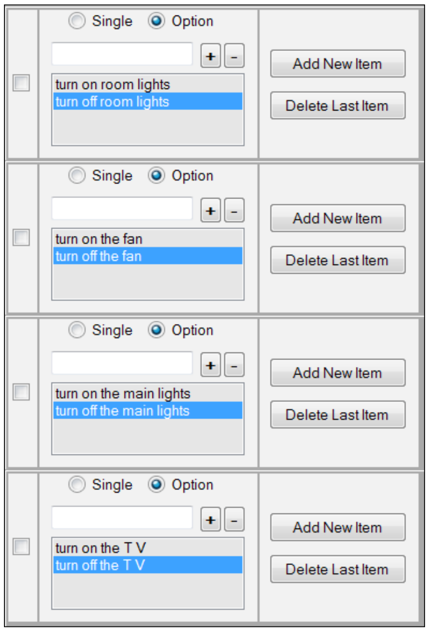
To add a command, use button in the upper right corner of the screen.

Turn Room Light ON
Single, Dest.: 192.168.1.8:23 (TCP)
Turn Room Light OFF
Single, Dest.: 192.168.1.8:23 (TCP)
Turn Fan ON
Single, Dest.: 192.168.1.8:23 (TCP)
Turn Fan OFF
Single, Dest.: 192.168.1.8:23 (TCP)
Turn Main Light ON
Single, Dest.: 192.168.1.8:23 (TCP)
Turn Main Light OFF
Single, Dest.: 192.168.1.8:23 (TCP)
Turn On TV
Single, Dest.: 192.168.1.8:23 (TCP)
Turn TV OFF
Single, Dest.: 192.168.1.8:23 (TCP)

Try all the buttons and make sure they work. And there you have it! The smart phone communication phase is complete.

Via the speech recognition software (BitVoicer)

Launch BitVoicer and load the `home_automation_complete.vsc` schema that came with this chapter. It comes with the following sentences:



turn on room lights
turn off room lights

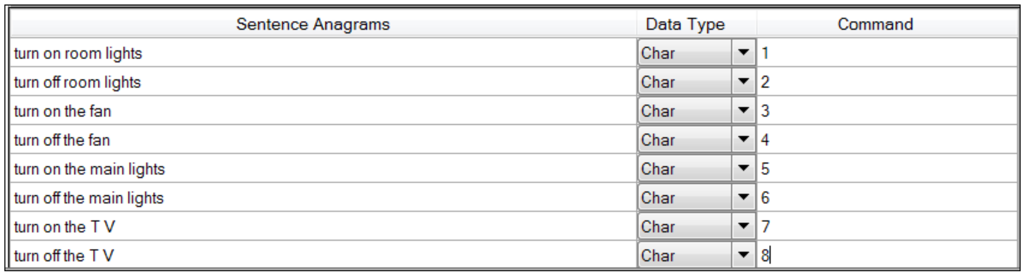
turn on the fan
turn off the fan

turn on the main lights
turn off the main lights

turn on the T V
turn off the T V

You can (and are recommended to) change these sentences to your liking, or add more options as we did in the second BitVoicer example (`turn on the lights` and `switch off the bulb`, remember?).

The command space will look like the following:



Sentence Anagrams	Data Type	Command
turn on room lights	Char	1
turn off room lights	Char	2
turn on the fan	Char	3
turn off the fan	Char	4
turn on the main lights	Char	5
turn off the main lights	Char	6
turn on the T V	Char	7
turn off the T V	Char	8

Once you are happy with the control phrases, press **Start** and test it out. Do the relays obey their master? If so, well we are done with all the tests.

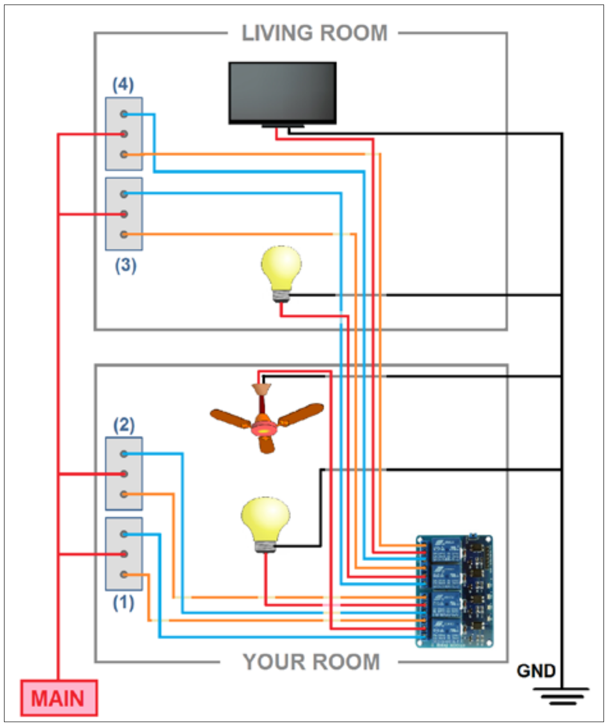
Now, we will complete the circuit.

Complete home automation

We are finally here. This is the last time we will be calling our electrician friend to help us. Depending on what appliances you are actually using as compared to what is being said in this section, the circuit arrangement may differ. However, the logic is the same.

Unplug the UNO. You will have to more or less create the circuit that is on the next page. Take a look at it. Do not let the number of wires confuse you. All the wires are insulated. The only wires that are connected to each other are the ones at the main line and the ground (neutral) line. The relay (connected to the Arduino) needs to be placed in a location close to the router, but it is also convenient enough to connect all the wires.

It is recommended that you keep your Arduino + CC3000 + relay setup close to at least one switchboard (perhaps the one in your room). The following image is just to show you where and how the connections go, but it is not an exact representation of how it will be. Physically placing the Arduino board close to a switchboard will make the wire-connections to that switchboard and the relays short and neat. All non ground wires from an appliance are connected to one side of the switch on the switchboard. The other wire connected to a simple switch will be the main (power).



Each relay has to be connected to a two-way switch, which would mean that you would have to replace a simple switch with a two-way switch for this setup to work. As before, be very careful while handling high voltages. Be patient and take your time. If you are controlling a single room, you will not have wires going around, but if you want to control appliances in different rooms, this is the most feasible option. Alternatively, if your budget allows, you can get another UNO and a CC3000 shield and connect them near another switchboard.

Power the Arduino through the 9V 2A power plug and let it connect to the network. Begin speech recognition and start the app on your smart device. What are you waiting for? Press, tap, and speak away! Feeling Tony Stark-ish yet? If not, you should. This is a milestone that you have reached! It was a wonderful moment when I first got this to work some time before working on this book. It's truly spectacular and you may feel like you can do anything and build anything now.

To take things further, you can buy yourself an 8-channel relay and an Arduino MEGA board, which would allow you to control even more appliances. You can also get yourself a wireless microphone and place it in a convenient spot; connect it to your computer so that you neither have to be close to your computer nor have to yell, for the speech recognition software to work.

Summary

Although no summary can do justice to the amount of knowledge you have obtained from this chapter, let us try our best. After learning, in the previous chapter, about how relays function and how the CC3000 works, we programmed them to work in unison. We were able to control the relay and in turn a home appliance by using a terminal on our computer and smart phone. In addition to that, we used some apps to directly control the relay through aesthetically pleasing GUIs. Then, we learned about speech recognition and understood how accurate (or inaccurate) it can be. We then used it to control the very same relay. Finally, we put together everything we have learned, in order to control four home appliances (with the help of our electrician friend), using the terminal, smart phone, and speech recognition. Finally, we built a complete home automation system.

In the next chapter, we will be building and programming a robot dog. This is one of my favorite projects because you can program any sort of personality as you would like. Since we have created a smart home, we will need a guard too, right? The next project will involve using a lot of new parts such as servos to aid motion, and will be quite different from this project. I hope you enjoyed this project and that you are confident enough to modify this project to suit your needs.

8 Robot Dog – Part 1

We've finally made it to the last project. More than the destination, I hope the journey so far has been enjoyable and educational. The previous chapter was on relays, relays, and more relays. Home automation is a very much an upcoming field, and I am glad that we had the opportunity to build a project that helped us to learn about it. We learned to communicate through Wi-Fi via the CC3000 Arduino shield. Using this, we were able to control the Arduino using our smart device, and ultimately we controlled all electrical home appliances. We also used BitVoicer, running on the computer, to enable speech control.

Now, coming back to this chapter, we will be building a robot dog. Cool! That's right. This is one of my personal favorites. We will be using everything that we have learned so far in order to create a small quadruped (four-legged) robot with a lot of capabilities, which you will learn about soon. You are warned that this project is very hard. *Why?* This is because you will be working with many servos (think of them as little modified motors)—the number of wires running around will make you dizzy, and the battery that is very powerful, could cause significant damage if not cared for properly—and programming the robot is a pain. *Why should I even start this project then?* Don't let the complexity demotivate you. I just want you to be mentally prepared before beginning this project. Besides, we will be building this together, and you will be guided every step of the way.

This project, **Robot Dog**, is split into three chapters to balance the length of each chapter. The project is further split into the following sections:

- Introducing Arduino MEGA
- Understanding servos
- Understanding power requirements
- Building the chassis

I have given you my word that you will always be learning something new.
In this last leg of the book, we will learn about using servos. Servos are, in simple words, motors that can only rotate from 0 to 180 degrees. They are the most commonly-used mechanical components for physical motion with Arduinos. Since they are digitally-controlled versions of motors, they are easy to program and can also be used to do different tasks. We will also learn to build the robot from scratch, by using our own creativity and not using an assembly-ready kit. For the first time in this book, we will not be using an Arduino UNO, but instead migrate to its big brother, the Arduino MEGA. Don't fret. It is exactly the same as the UNO, except it has many more input/output pins that allow us to connect more components. This project is very detailed, so be prepared.

Prerequisites

To minimize the cost of this project, while also trying to teach you that sometimes household items can be used in creative ways as materials for a project, we will be using ice cream sticks to create the chassis (ice-cream sticks should be available in your local stationery store; if you, however, choose to acquire the ice cream sticks by visiting your local ice cream parlor every day and get sick of doing so, I am not responsible; nevertheless, this is the way to go).

The rest of the components needed for this project are as follows:

- 1x Arduino MEGA 2560
- 1x USB cable A to B (also known as the printer cable)
- 12 x 9g Micro servos (2+ extra servos recommended)
- 1x Breadboard
- 40x Male-to-male connecting wires
- 20x Male-to-female connecting wires
- 20x Ice cream/popsicle sticks
- 1x Wood glue
- 1x Ruler/measuring tape
- 1x Regular-type screwdriver
- 1x Multimeter
- 1x 7.4V (2 Cell) 2200 mAh (or above) LiPo battery
- 1x 3A UBEC (Universal Battery Elimination Circuit)
- 1x XT60 male
- 1x 7.4V 500 mAh LiPo battery
- 1x Female JST connector

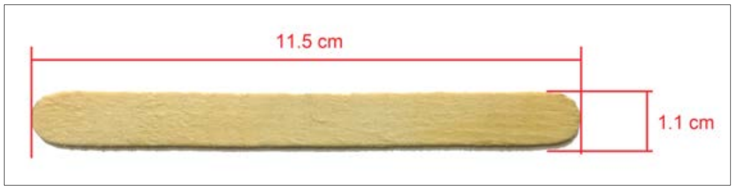
- 1x LiPo battery charger
- 1x Pack of rubber bands
- 1x Pack/roll of double-sided tape
- 3x Insulating tape (different colors)
- 1x PC with a microphone
- 1x Proto board
- 40x Male headers
- 1x Soldering kit
- 1x Thumb pin or needle
- 1x Pack of paper clips
- 1x Pliers
- 1x Wire cutter
- 1x Cutting blade

You will also need the following software:

- Putty (terminal)
- BitVoicer (speech recognition)

Since the shape and size of ice cream sticks will be different in different parts of the world, there would be the need of strengthening the sticks, by sticking two or more of them together. Hence, it is suggested that you buy more sticks.

To give you an edge for planning purposes, here is the ice cream stick used in this project, along with its dimensions:

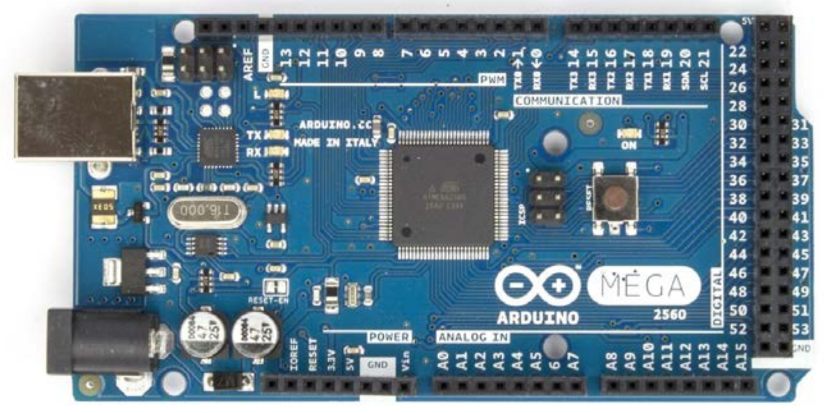


Introducing Arduino MEGA 2560

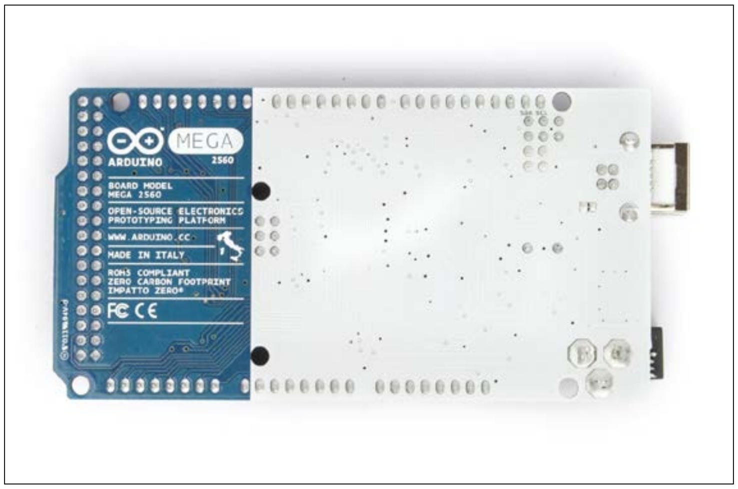
In this section, we will briefly go through the Arduino MEGA 2560 board. We will understand how it is different from the Arduino UNO, and also see why we chose this board over the Arduino UNO.

The microcontroller

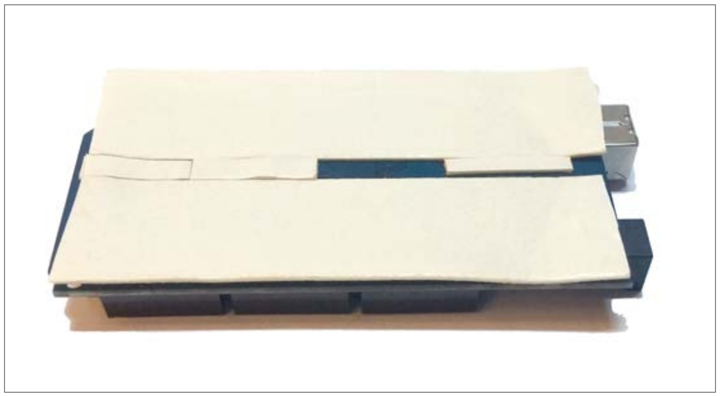
As mentioned before, Arduino MEGA 2560 is like a big brother of Arduino UNO. It is almost twice as long and way more powerful; Arduino MEGA 2560 is the successor of Arduino MEGA. Unlike Arduino UNO, MEGA 2560 has 54 digital input and output pins, as shown in the following image:



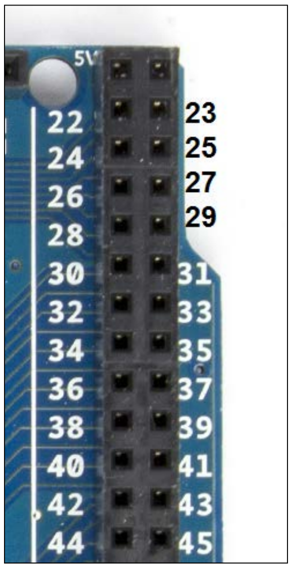
With the bottom looking like this:



Before we get started with using the board, it is a good idea to use one side of the double-sided tape to cover the base of the board. This is to ensure that, while in use, the Arduino does not get short-circuited if the base makes contact with a conductive material. Stick a double sided tape to the bottom of the Arduino like this:



On the right side of the MEGA (short for Arduino MEGA 2560), you will notice some missing labels. The following image should help get this clarified:



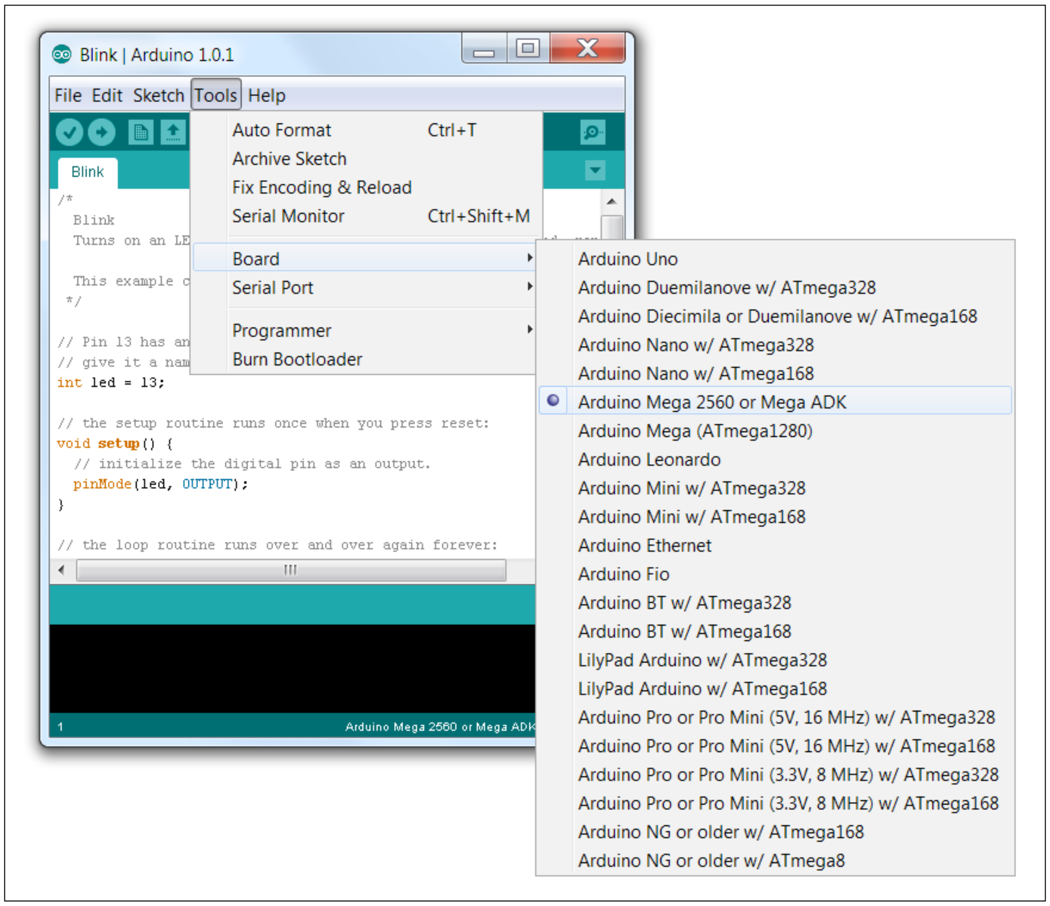
In the pin-pair shown in the preceding image, pins 22 and 23 are 5V pins, just like the one in the Arduino UNO.

Testing MEGA

We will quickly test MEGA to ensure that it works as expected, and to also give you a feel of the board. As always, plug MEGA to your computer by using the corresponding USB cable. Your computer will automatically search for the drivers and install them. If you see that the on-board LEDs are turned on, you know the board's connection is fine.

Launch Arduino IDE and, as you did in the very first chapter, open the **Blink** example. It can be found in **File** → **Examples** → **01. Basics** → **Blink**. Before uploading the program, you need to select Arduino MEGA as the board to be uploaded to.

To do this, go to **Tools** → **Board** → **Arduino MEGA 2560**, as shown in the following image:


Blink | Arduino 1.0.1
File
Edit
Sketch
Tools
Help
Blink
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
1
Arduino Mega 2560 or Mega ADK

```
/*
  Blink
  Turns on an LE

  This example c
 */

// Pin 13 has an
// give it a nam
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
```

Don't forget to choose the right serial port too. Only after that, you can upload the program. You should notice the on-board LED blinking. Once you have verified that Arduino MEGA works as expected, you are ready to move on.

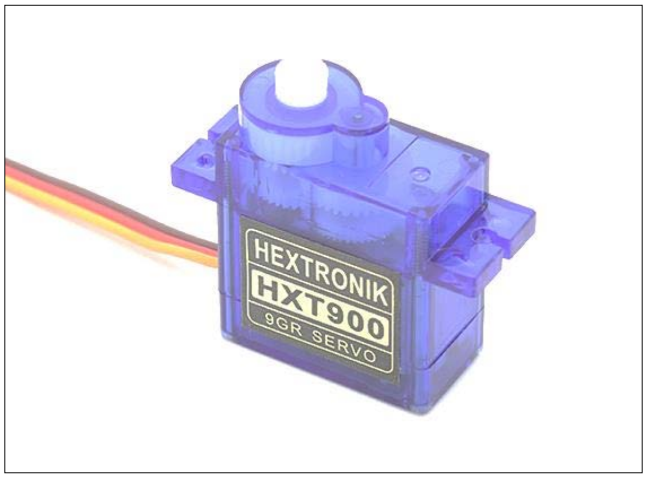
We will go to the next part where we will learn about servos.

Understanding servos

Just as we had done for the relays in the previous chapter, this section will tell you what a servo is and how it is used. We will also test the servo by using a **Blink** like example.

Servo 101

A servo (or servo motor) can be thought of as a motor, but with the ability to be controlled by its angular position. The servos we are using are 180 degrees micro servos, as shown in the following image:



Servos come in different sizes and capacities. We could use the standard-sized servos that would give us more torque, but it would require more power. For this purpose, a micro servo will do the job.

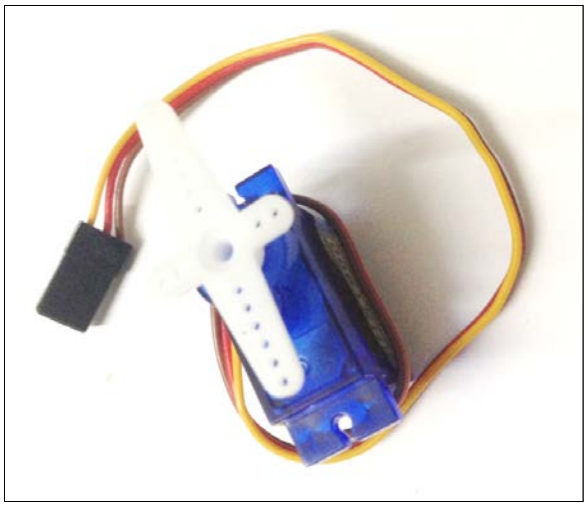
In the preceding image, you may have noticed that there were three wires. The wires' colors correspond to the following:

- Black/brown: GND
- Red: 5V
- Yellow/orange: Signal

A signal is basically how we control how much a servo turns (between 0 and 180 degrees). Let us now test one.

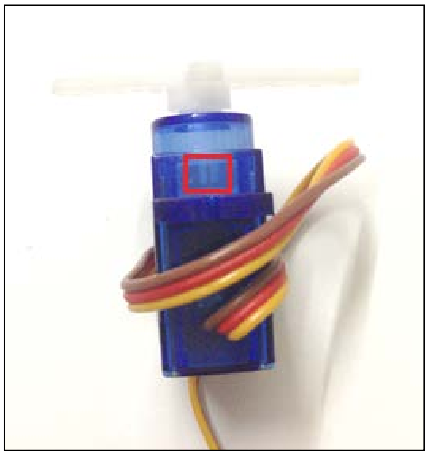
Testing a servo

Pick up a servo. It should come with different types of servo arms (the typically white-colored pieces that can be attached to a servo and then screwed together). For now, choose any arm and fix it onto the servo. For this part, you need not screw it. However, if you think that the attachment is too loose, you can gently screw them together. The servo will look like this:



Try to physically, but gently, turn the arms. You will notice that you can freely turn the arm until you reach a certain point. This is a barrier that does not allow the servo to turn more than the preset 180 degrees. If you turn it in the other direction, you will notice another barrier. You might also notice that the two barriers are not located exactly 180 degrees apart. Do not worry about this. If you couldn't budge the arm when you first tried it on, you know that there is something wrong with your servo. Perhaps the gears inside are broken. You will either have to replace the gears, or simply use a new one. It is a good idea to test all the servos that you have purchased before proceeding.

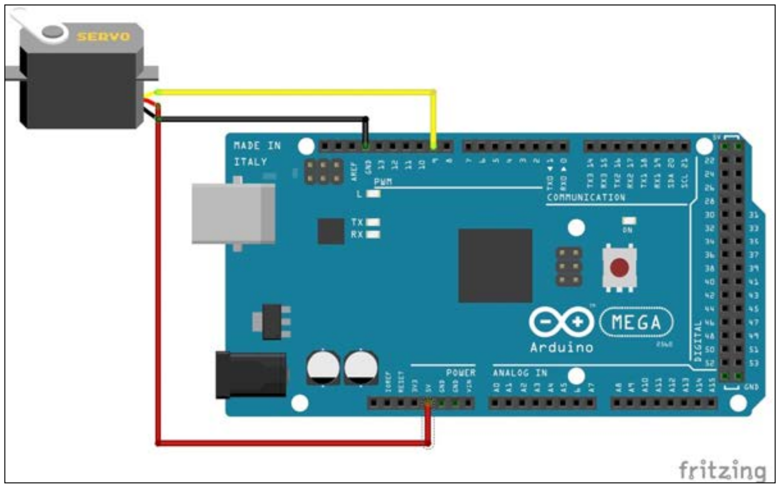
In the following image, you will notice that, at the very centre, there is a white protrusion from the base of the top big gear. This protrusion is marked here:



Try moving the arms of your servo to achieve this position. This is the central position of the servo (90 degrees).

Programming a servo

Programming a servo is almost as easy as programming an LED. Yes, these are the exact same words that I had used when introducing you to programming a relay in the previous chapter. Disconnect the MEGA USB cable. Go ahead and create the following circuit:



You will have to use the male-to-male connecting wires to connect the servo to MEGA.

The connections from Arduino to the servo are as follows:

- GND → GND (black/brown)
- 5V → VIN (power—red)
- D09 → Signal (yellow/orange)

Plug MEGA back in. You may hear some sounds from the servo. It may even change its position. This happens, do not worry about it. Now, open the servo sweep program, which can be found at **File** → **Examples** → **Servo** → **Sweep**. Then, upload the program.

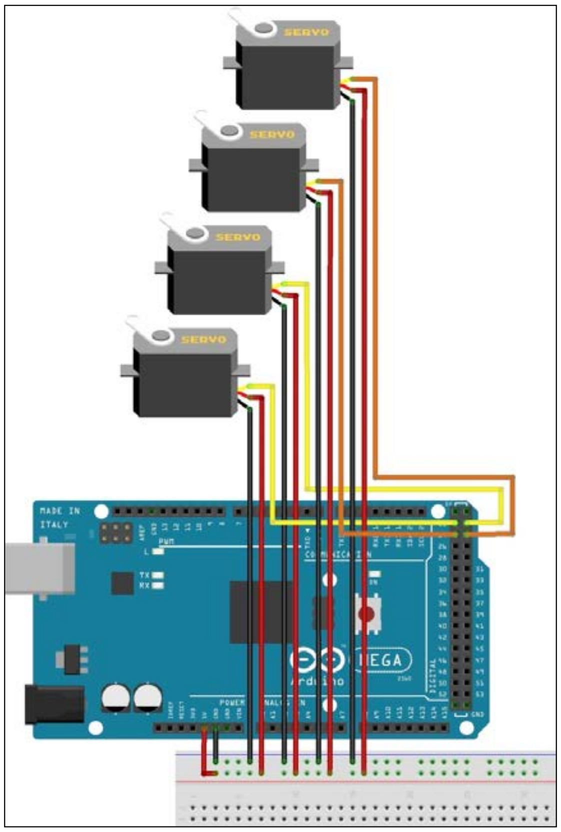
You will see that the servo rotates from 0 to 180 degrees and comes back to 0. It keeps repeating this action. In this program, look at the following segment inside the void loop:

```
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180
degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position
in variable 'pos'
    delay(15);                       // waits 15ms for the servo to
reach the position
  }
```

Notice the line that mentions `delay(15)`. This is a very important line of code. This `delay` controls the speed at which the servo turns. For now, it is a good idea to keep the minimum delay at `15`. You can try adding another servo, and change the program accordingly to get the similar result. If you need some help, read the following section.

Using multiple servos

Let us now try to control multiple servos. This is our ultimate task to make the robot walk, isn't it? Since the Arduino MEGA has only three 5V pins, we will make use of a breadboard to make the connections. Create the following circuit:



The diagram depicts only four servos. Use the same pattern and connect all the 12 servos. As I said before, the number of wires running around is dizzying. Be careful while using the male-to-male connecting wires. Make sure that the ground, voltage, and signal wires go only to their respective destinations.

The connections are basically as follows (Arduino → Servos):

- GND → GND
- 5V → VIN
- D22 → Signal Servo #01
- D23 → Signal Servo #02

  ..

  ..
- D33 → Signal Servo #12

Now, open the `multi_sweep.ino` file that should have come with this chapter. Plug in your MEGA and upload the code. What do you see? If the Arduino MEGA didn't *die*, then you are one of the lucky ones. In most cases, you will notice that the Arduino MEGA simply turns off, turns itself back on, tries to run the program, fails to do so, and turns off again.

Do you know why this happens? The answer is, because the MEGA simply cannot handle the load of all those 12 servos running at the same time. At most, you can get away with running four servos at a time. The total current required by all the servos when they are in action is simply not being supplied by the computer's USB port. *You didn't just kill my MEGA, did you?* No, don't worry. The Arduino MEGAs are made to withstand much more than the insufficient current. How are we going to deal with this then? How can I control 12 servos, when I can't even power them?

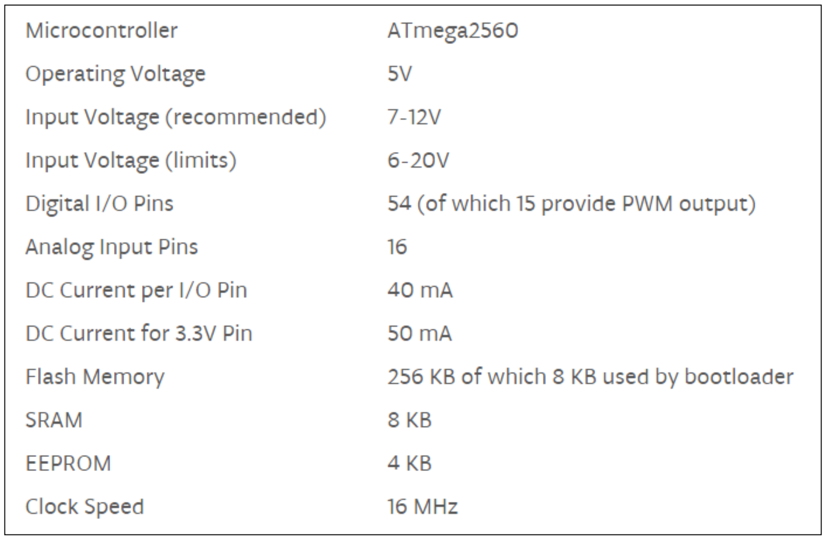
This is exactly what we are going to discuss in the following section.

Understanding power requirements

The basic necessity for any electro-mechanical device is a power source. Selecting the right power source for a particular device is an important trick of trade that every **Arduino tinkerer** needs to know. This is exactly what will be taught in this section.

Limitations of Arduino MEGA 2560

If you look at the specifications of Arduino MEGA 2560 on the Arduino website, you will see the following:



Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

The important thing to note is the operating voltage that says 5V. This means that no matter how large of an input voltage you put in, the MEGA will always convert it to 5V. In reality, it will be slightly less than 5V due to miscellaneous resistances. Another thing to note is the DC current per I/O pin that says 40 mA. An average micro servo has a voltage rating of 4.8 - 6V, and under heavy load, its current consumption can reach up to 1A that is 1000mA. So, it is surprising that MEGA could even power one servo sufficiently.

Choosing the right power source

Since we are making a robot dog and not something that is stationary, we are going to rule out using a wall power supply such as the one we used for the home automation project. Also, since a computer cannot provide sufficient power to the Arduino, this is also crossed out of the list. This leaves us with batteries. However, there are so many batteries to choose from. Obviously, the component list for this project, at the beginning of this chapter, has already given away what we are ultimately choosing; let us understand what our choices were and how we decided what to go with.

The reason why we are going through the process of finding the right power source is because this is very important, not only for this project, but also for your future endeavors.

Okay, so we are going to use batteries. Yes. *What kind?* Since we are going to build a robot and we want to increase reusability, we will cross out nonrechargeable batteries. This leaves us with two popular choices:

- Rechargeable AA battery packs
- LiPo batteries (Lithium polymer)

As mentioned before, a micro servo requires 4.8 - 6V to work. Each servo also has a peak current draw of 1A, which makes it 12A in total. In reality, not all servos will be moving at the same time, so the 12A is just a maximum figure we are using for calculation purposes.

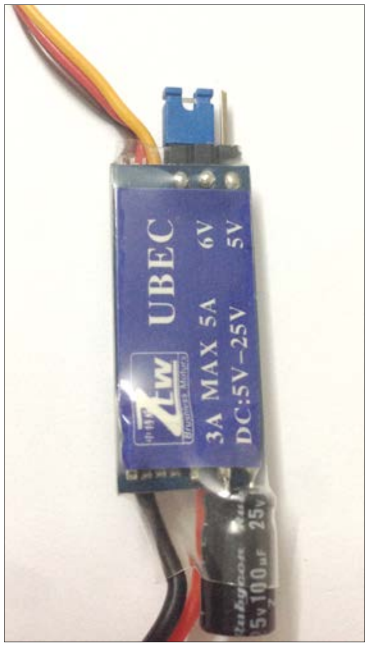
Each rechargeable AA battery has 1.2V. So, if you connect five of them in series, you would get a total of 6V that would suffice. Now, you also need to make sure that the current provided by the batteries is high enough for the servos to work. Rechargeable AA batteries rated ~2000+ mAh batteries are readily available in the market and can also be used. The reason why we are using LiPo over AA batteries is because the former lasts longer and their performance to weight ratio is much greater. However, if your future project doesn't have to take weight into account, you are free to use rechargeable AA batteries.

Now, let's talk about LiPo batteries. As mentioned in the prerequisites, we will be using a 7.4V (2 Cell) 2200 mAh LiPo battery. If we look at the specifications for this battery, we will see that the constant discharge rate is 20C. '20C' means that the battery can give off 20 times the current of its rated capacity. This means that the battery will discharge 20 * 2200 / 1000 = 44 Amps at a constant rate. This is safely above the 12A required by the servos. The more the mAh, the longer the battery will be able to power the Arduino. A LiPo battery with an ever high mAh could be chosen, but that would increase the weight of the battery. Since the battery is going to be carried by the robot, increasing the battery's weight increases the load on the robot. Also, there is only so much load those micro servos can endure. So, we have to create a balance between performance and weight. Hence, the 7.4V (2 Cell) 2200 mAh LiPo battery was chosen.

Using the right power source(s)

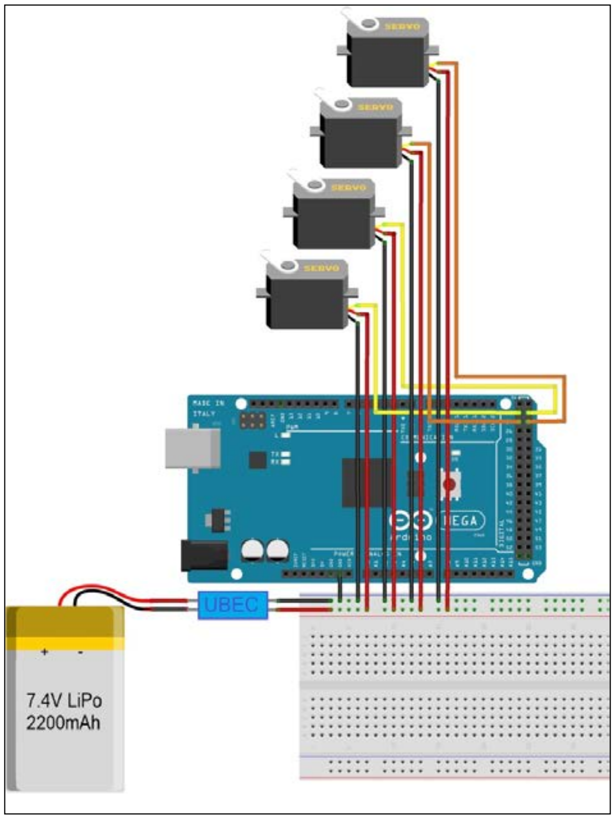
If you had noticed that there were two batteries mentioned in the prerequisites, you were correct. The first and more powerful one that we just talked about is going to be used only for the servos. The smaller 500mAh battery will be used only to power the Arduino separately. The reason why we are not hooking up the 2200mAh battery to both the servos and the MEGA is because of the noise that is generated by the servos, which might cause unwanted disturbances in the MEGA.

However, didn't you say the servos are rated at 4.8-6V? You now want to hook up a 7.4V directly to it? Hold your horses. This is what we are ultimately going to do, but before that I must introduce you to our little friend, the 'UBEC'. What this device basically does is, it drops the voltage (no matter how high it is) to 5 or 6V, depending on what is wanted without compromising the current. Since we are going to stick with 6V, move the jumper to choose 6V as the output. The UBEC should now look like this:



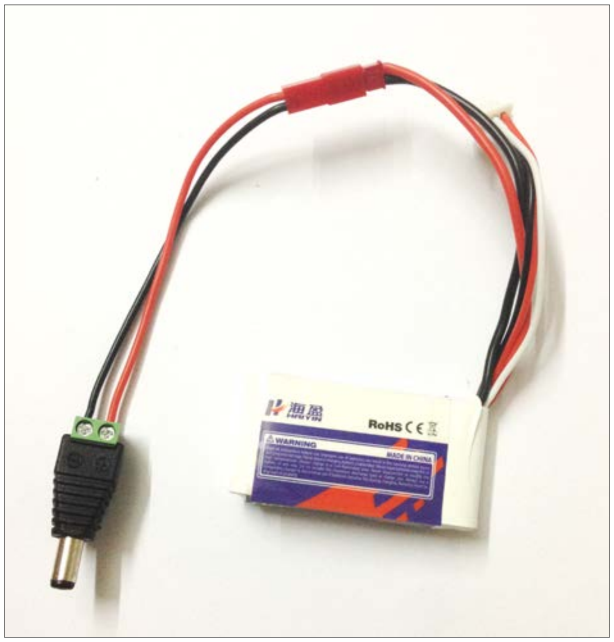
Solder the voltage and ground (red and black respectively) wires to the XT60 male connector. Make sure that the ground is soldered to the terminal with the chamfered side. Wrap some insulation tape around it to make sure no unwanted connections occur. On the other side of the UBEC, the triple wires that look like that of a servo will basically act the same. Meaning, the black/brown wire represents ground and the red wire represents the constant 6V. The yellow/orange wire is the signal that is usually used in RC planes in receivers, but we will not make use of this wire.

Now, we are going to modify the previous circuit to ensure that all the servos work together without choking the MEGA. To do this, create the following circuit:



The main thing to notice is that only the servos are being powered by the LiPo + UBEC. The MEGA is still being powered separately. Ensure that the right polarities are connected to and from the battery and the UBEC. One key thing is that the ground of the breadboard should be grounded to the Arduino MEGA. This is to establish one common reference ground. Be very careful while handling the high discharge battery. Now, try running the `multi_sweep` program again. See the difference? The servos don't stop; they keep rotating. This means that their power requirements have been satisfied. If a particular servo is not rotating, make sure that its wiring is done properly.

Before going to the chassis construction part, connect the smaller 500mAh battery to the 2.1mm DC plug by using the female JST connector. You'll have something like the following:



Now, unplug the MEGA from the computer and plug the 2.1mm plug into the MEGA's 2.1mm jack. It works! So, you have successfully used a portable power source to power the servos and the MEGA independently.

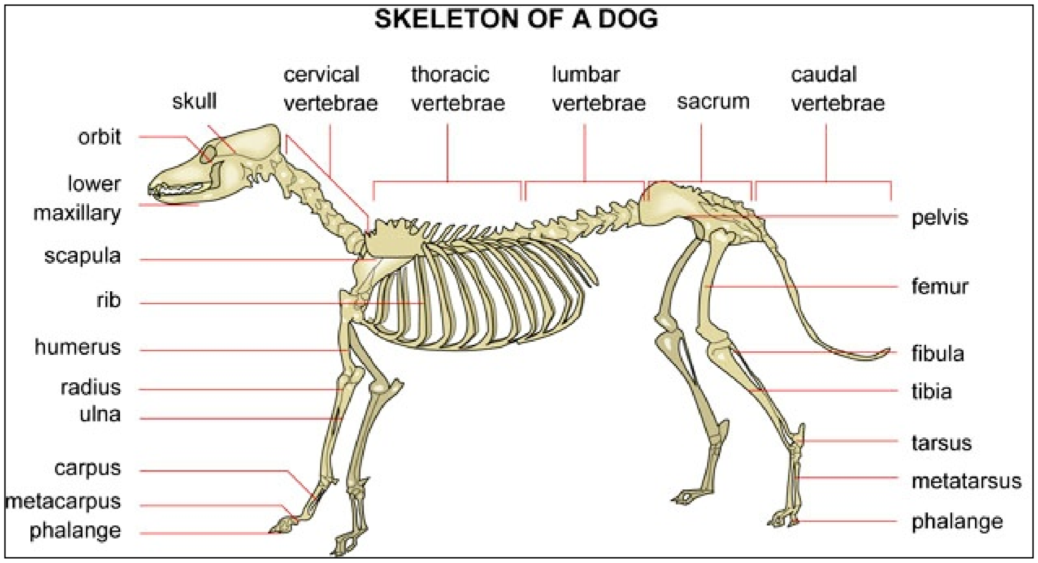
Building the chassis

A robot always comprises of three fundamental fields. They are mechanical, electrical, and programming. Working out the power requirements falls into the electrical aspect. Writing code to control the motion of the servos falls under the programming category. Building and ensuring the chassis can support the weight of all the components that the bot is carrying is a mechanical challenge that will be addressed in this section.

Using prior art

Before rushing into the building phase of the body of the robot, let us take a moment to understand what we are even trying to build in the first place. We are making a robot dog. So, let us look at the nature and see how a dog stands and moves.

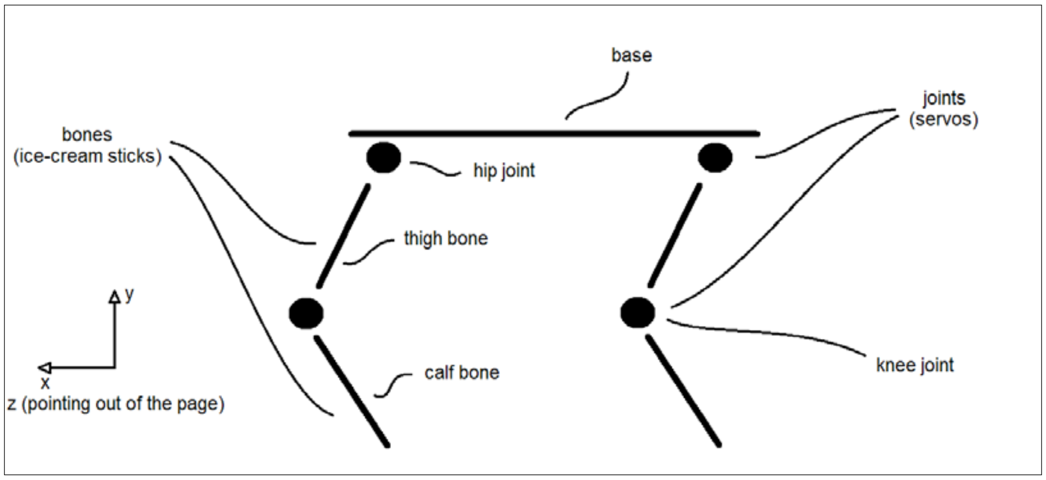
It is very hard for us to replicate the dog at its entirety. This is mostly because an actual dog uses strong muscles in its legs to walk (jump), and we are using servos. Now, let us look at the following bone anatomy of a dog to get a deeper look at the skeletal structure that allows the dog to walk the way it does:



Why are we doing this? Didn't I say that I will teach you as much as possible? This part of the project is called *getting inspiration from prior art*. This technique is used almost everywhere while designing a new product or simply creating a new design. It is used in research papers and even patents. Sometimes it helps to use what has already been done before, either natural or artificial, to aid in creating a new design.

Let us focus on the legs. Notice how the joints are connected. To make things easier, we will simplify this anatomy to fit our needs. We will modify the legs to just have a joint at the hip and the knee. This ensures that we won't have to use too many servos. Also, we are going to make the body (not including the legs) flat.

Since we are using straight, flat ice cream sticks as the bones, we have to ensure that they are proportionate. Here is a basic side view of what we will be making:



This stick figure shows the side view of our robot. The bones will be made of popsicle sticks (ice-cream sticks), and the joints are simply servos. The base will also be made of the same sticks, but it will be rectangular in shape, with its corners attached to servos; the central part will be made to support the other components such as the Arduino, batteries, and so on.

The hip joint (closer to the base) will comprise of two servos attached to each other so that the leg can be rotated in two axes (axis): the y-axis and the z-axis. The knee joint (conjunction of the thigh bone and the calf bone) is going to have a single servo that allows the calf bone to be rotated only in the z-axis. Make a note of all the names of the joints and the bones in the preceding diagram. We will be using the same names during the construction and programming process, to label different joints/bones.

Now that we have a basic understanding of what we are going to build, we can proceed with its actual construction that will be covered in the next chapter. You are free to change some details of the structure while we build it. However, you should be wary that this change may result in intricate changes you have to do for the code as well.

Summary

This chapter is very different from the previous chapters that we have gone through. This is because, in this chapter, we looked at every detail to ensure that the learning process aids in your future projects. We started by introducing ourselves to the Arduino MEGA 2560 and even ran the **Blink** example, which we had executed a long while ago in the very first chapter with Arduino UNO. After that we learnt about servos. What they are and how they are programmed. While programming multiple servos, we faced issues that were concerned with the power requirements for the servos. In the subsequent section, we learned how to select a suitable battery to nullify the issue. We even tested this out and were able to successfully control all the 12 servos through the Arduino MEGA. Then, we imitated the process of building the chassis by looking at the prior art of dogs, and created a simple sketch that helped us visualize what we are ultimately going to build.

In the next chapter, we will actually build the entire chassis by using ice cream sticks, and fit the joints with servos. We will also complete the circuit that will allow us to control all the 12 servos.

9 Robot Dog – Part 2

The previous chapter introduced you to the Arduino MEGA: servos and battery requirements. It ended with an understanding of what we are going to build by the means of using a dog as prior art.

In this part (2 out of 3), in the series of the robot dog chapters, we will focus on building the chassis of the robot dog, and then completing the circuit. What we are about to do in this chapter is time consuming, so it is okay to take breaks in between to give your fingers and mind some rest. In summary, we are going to cover the following topics:

- Building the chassis
- Completing the circuit

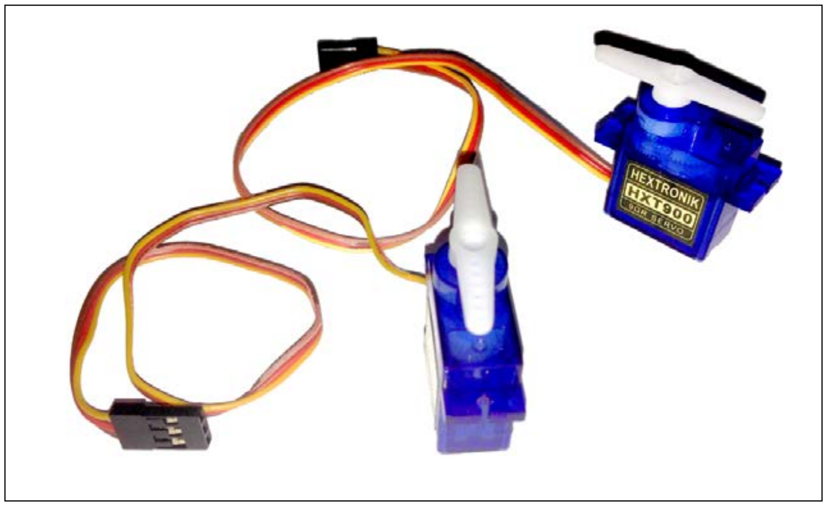
That being said, let us begin.

Building the chassis

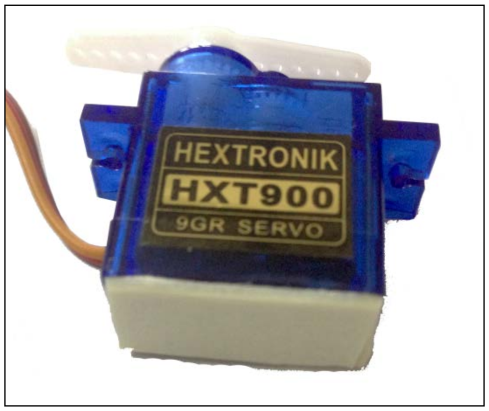
We have already finished the *prior art* part of this section. That subsection has inspired us to design our robot in a similar fashion of a dog. We will use this knowledge to build the body of the robot. This section may bring about nostalgic memories of art and craft projects that you did when you were little.

Sticks and servos

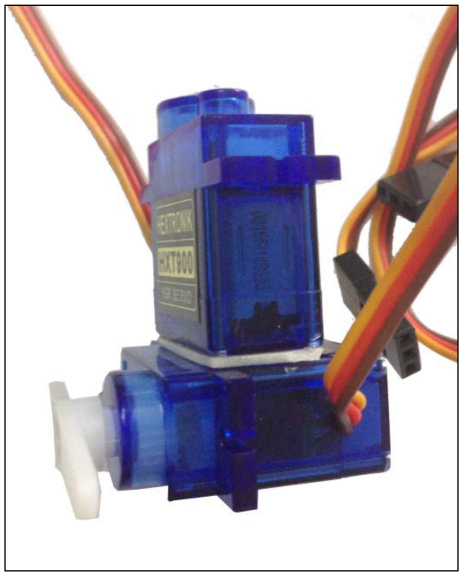
Start by clearing out some space on your workspace. Then, pick out two working servos and put on their arms.



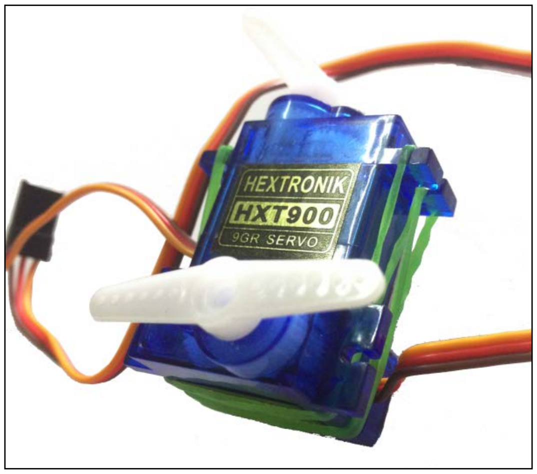
Now, cut out a piece of double-sided tape such that it covers the base of the servo as shown:



Now, stick this servo onto the other servo at a 90 degree angle like this:

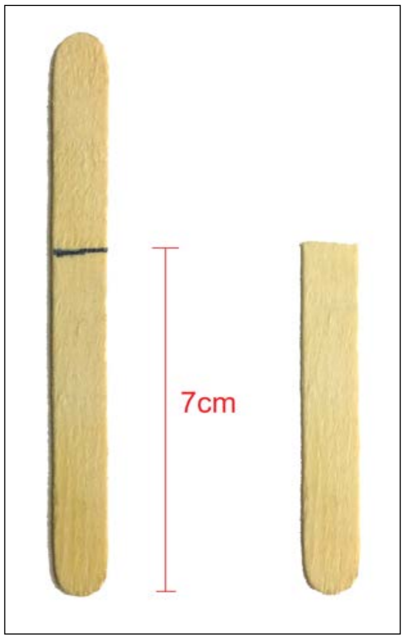


Use two rubber bands and stretch them around the servos, as shown in the following image, to strengthen their bond:

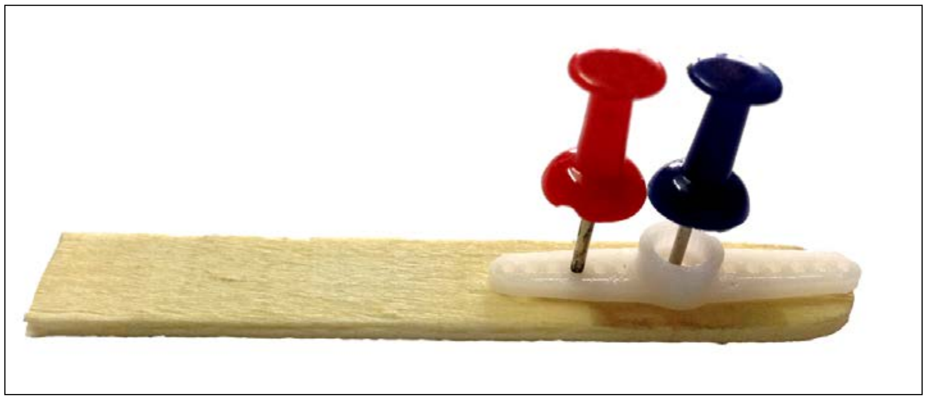


This is going to be our hip joint that was mentioned earlier.

Now, pick the straightest-looking ice cream stick, mark a line that is 7 cm from one end, and cut it:

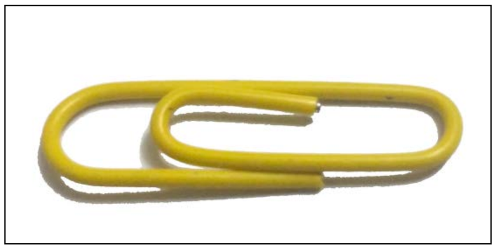


Now, laying the two-sided arm of the servo at the edge of the cut stick, poke two thumb pins (or any other pin) such that one of them is at the center and another one is through the tiny hole along the arm of the servo, like this:

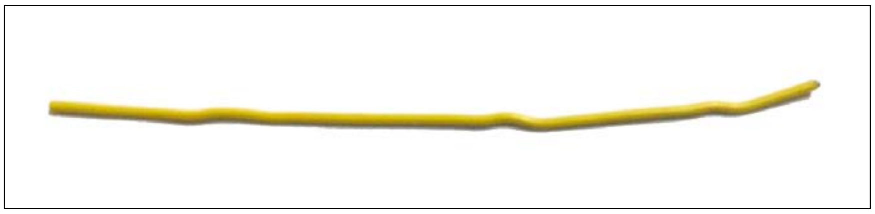


Now, remove the pins and the servo arm. Use the pins to make holes through the thickness of the stick where they were previously poked. It is suggested not to hammer the pin into the stick because this may result in creating a crack along the length of the stick from the hole, which decreases the strength of the stick. Use a smooth rotational motion to push the pin through the stick.

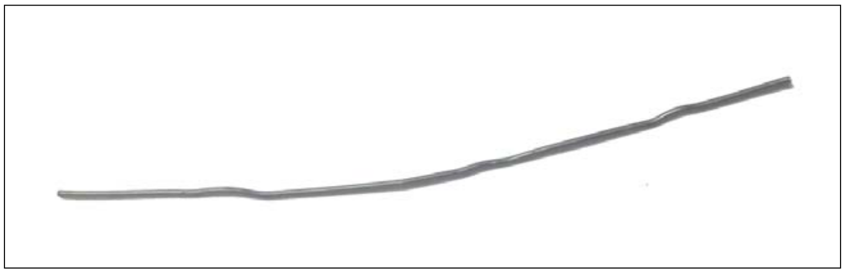
To attach the servo arm to the stick, we will use a screw that goes along the axis of the servo, and also a paper clip. Most paper clips, such as the one shown in the following image are metallic with a plastic coating:



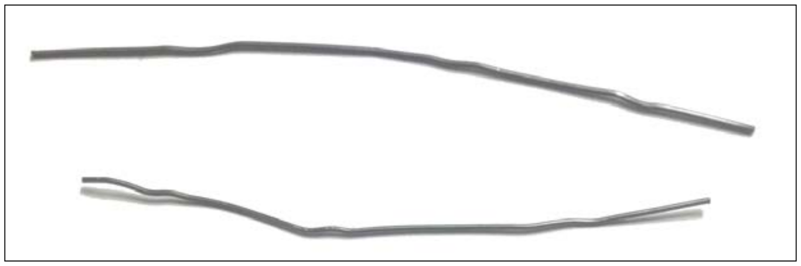
We only want the metallic 'wire' that lies inside the coating. To get it, unbend the paper clip to make a straight line. If it is hard to accomplish this with your fingers, you should use a plier to do this job:



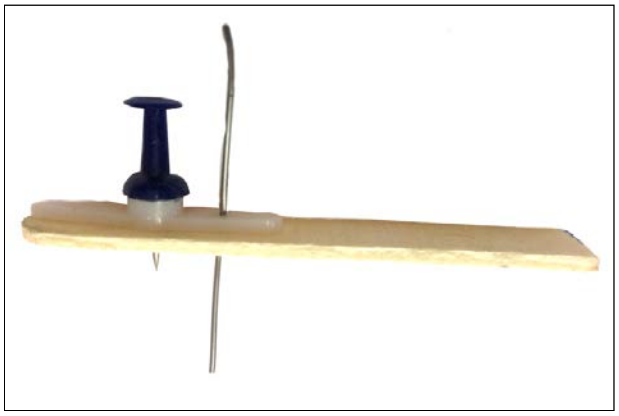
Now, holding a plier in one hand and a wire cutter in another, remove the coating of the paper clip:



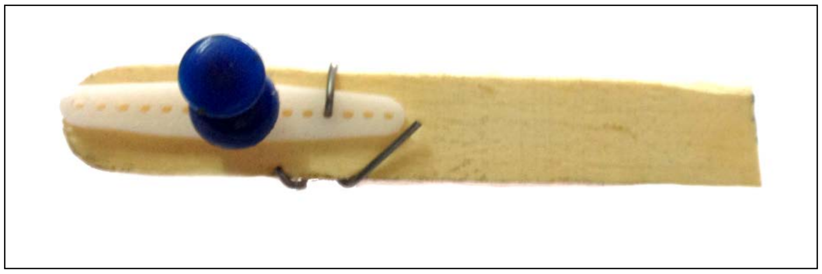
Fold this clip into half and cut it into two pieces at the fold:



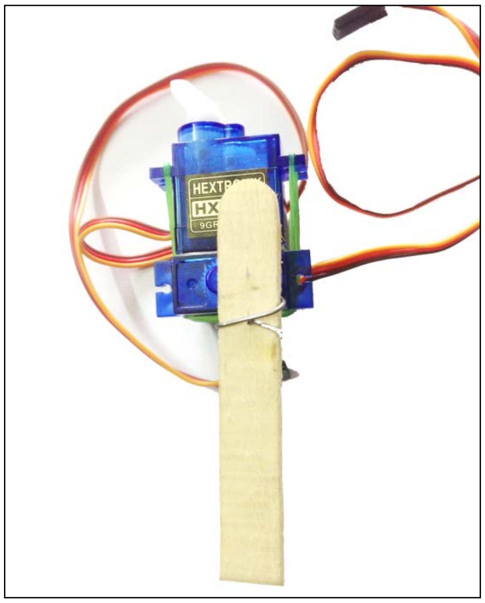
Place the arm of the servo at the position corresponding to the holes that we made earlier. Push one of the halves paper clips through the hole on the arm (not the central one). Fold the clip in such a way that it tightly locks the arms in place. Use pliers to do this; be careful of the sharp edges:



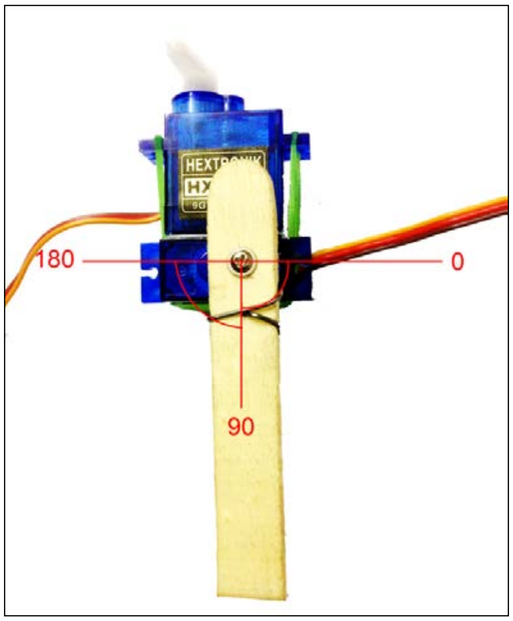
Bend the straightened clip to fasten the servo arm to the stick:



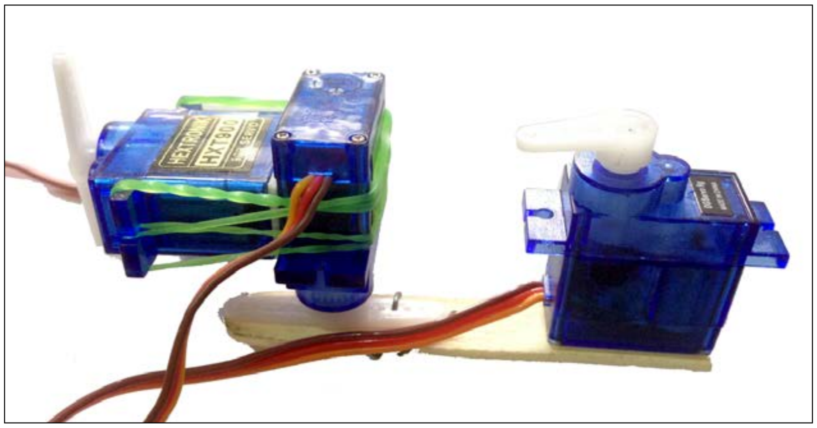
Now, center the lower servo. Attach the servo to its servo arm counterpart, such that the stick is perpendicularly downwards:



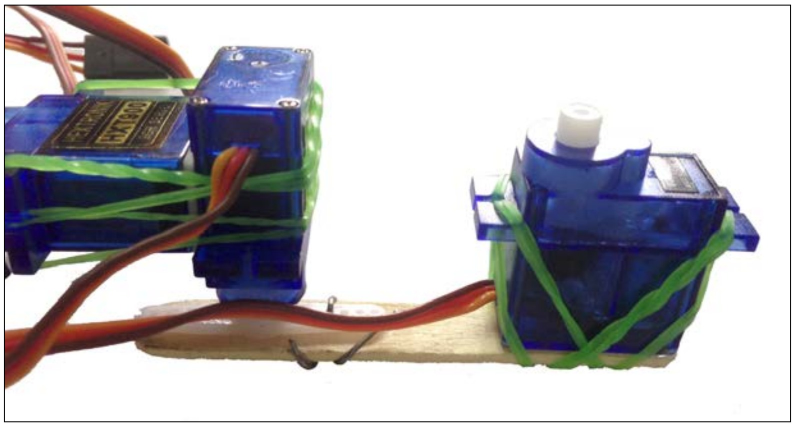
Once you are satisfied with the positioning of the stick, tightly screw the stick to the servo using a screw that came with the servo. These are the approximate angles you could achieve with this arrangement:



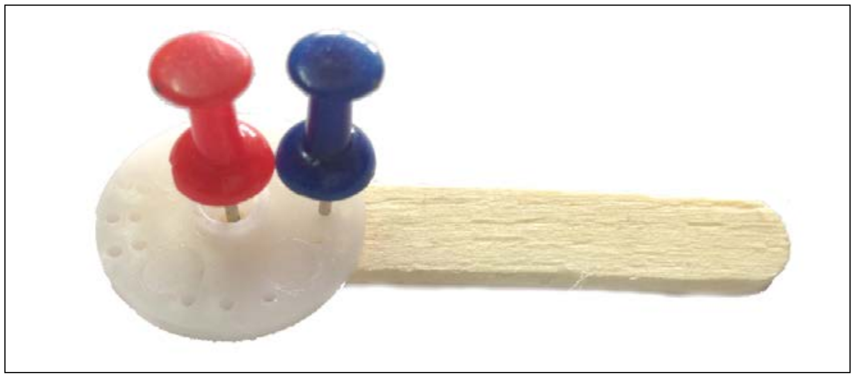
Now, take another servo and apply a double-sided tape to cover its base. Stick this servo to the stick above, in such a way that the edge of the servo superimposes on the edge of the stick:



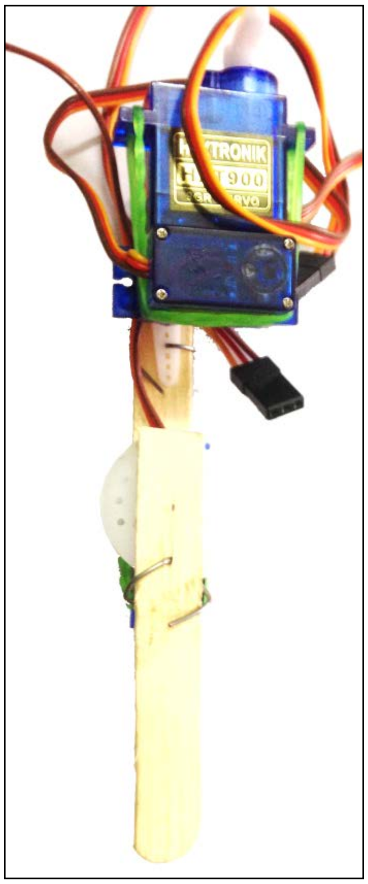
Now, tightly bind this servo with rubber bands as well:



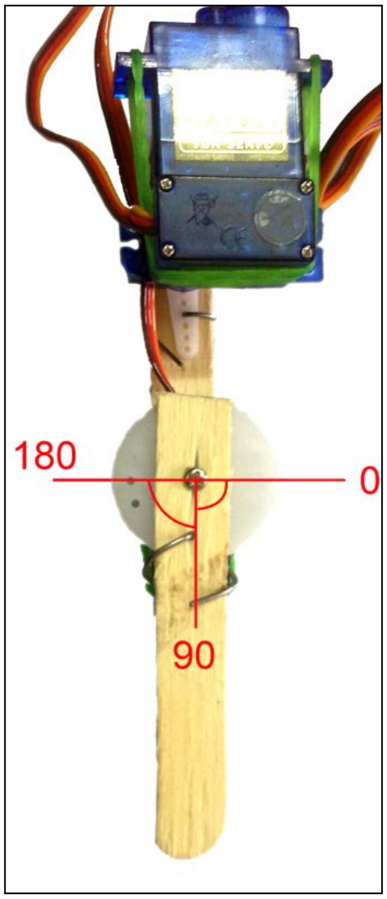
This servo serves as the knee joint. Now, take another ice cream stick, measure 6.5 cm from one end, and cut it at that mark. As we did before, take a servo arm and place it at the edge of the stick. This time, place it at the blunt edge. Use two pins to mark the locations of the necessary holes:



Make the holes using the pins, and attach the servo arm to it. Hold it in place by using the other half of the unsheathed paperclip. Center the knee servo and attach the stick in this manner:

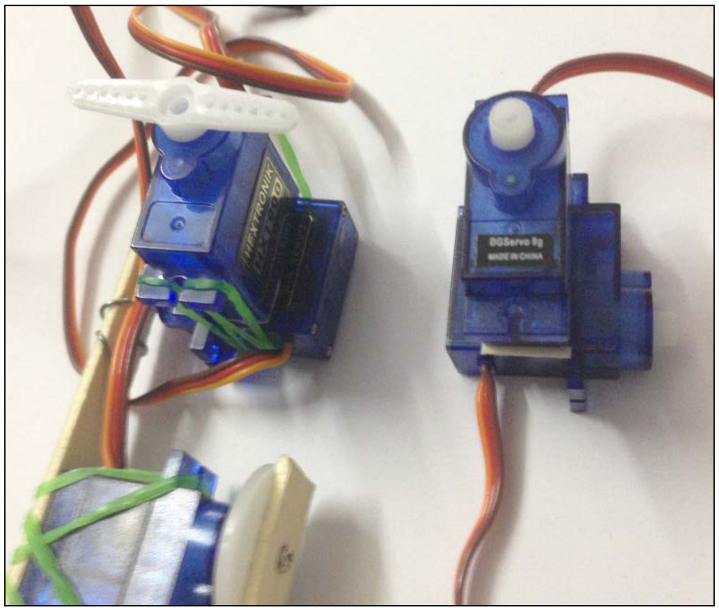


When you are okay with the angular movement of the stick, screw it tightly. You should be able to get the following angle with this joint:

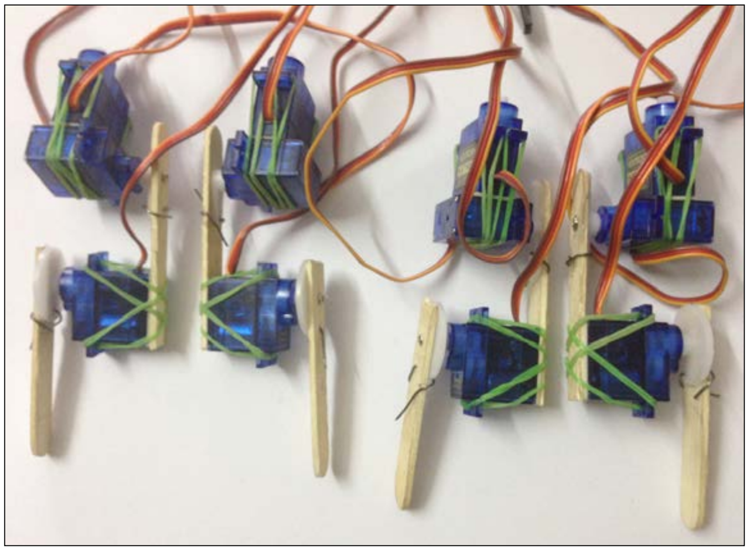


You have now created one of the four legs. There are three more to go. You can create another leg in exactly the same way as you did just now.

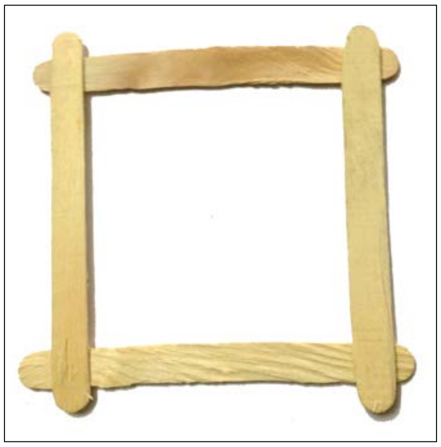
Next, you need to create the other pair of legs. The only difference is about the way the servos at the hip are joined together:



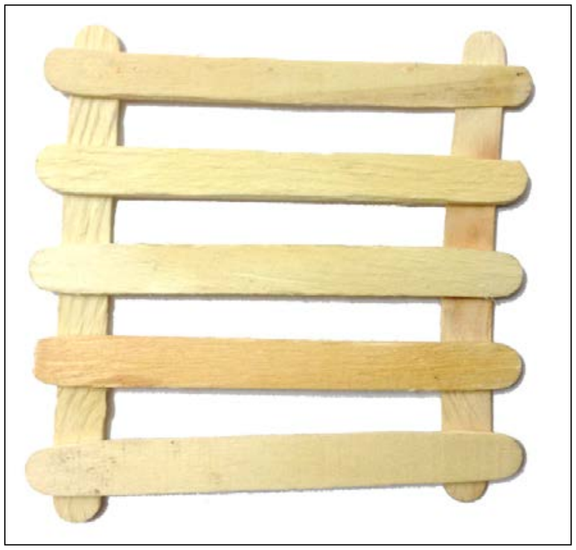
They are mirror images of each other. Now, create two legs with that joint. When you are ready with all the four legs, we will be able to move on:



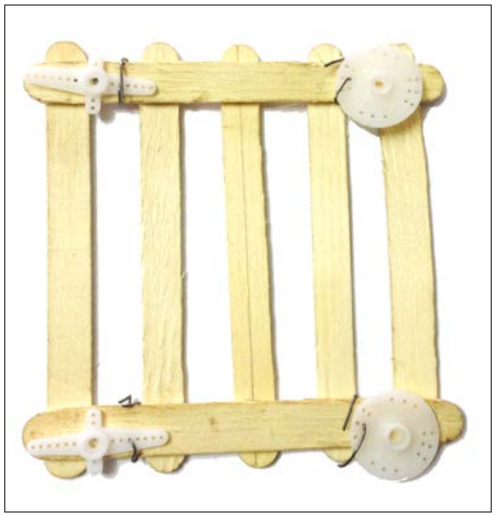
Now, we need to create the base. Take four ice cream sticks with the same length and place them in the following formation:



Basically, each stick is placed such that its 'curvy end' is just outside the superimposition region. Go ahead and stick them in this position with wood glue. What you can do now to make the base a bit stronger is lay more ice cream sticks in the following manner and stick them:

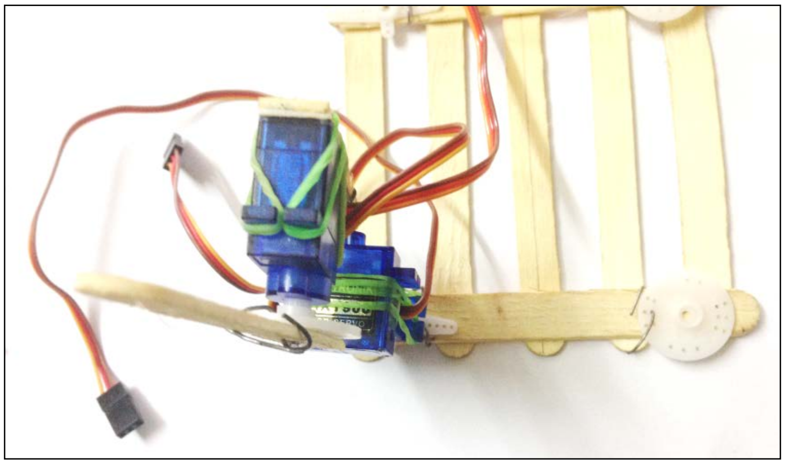


Now, the base is ready. All that is left to do is attach each leg to the corners of the base. To do this, mark points 2 cm away from each end. As we did before, place a servo arm and hold it in place using two pins, create holes in these points with the pins, and send an unsheathed paper clip through the noncentral axis hole and bind the arm in place. Do this at all the four corners and you'll get the following result:

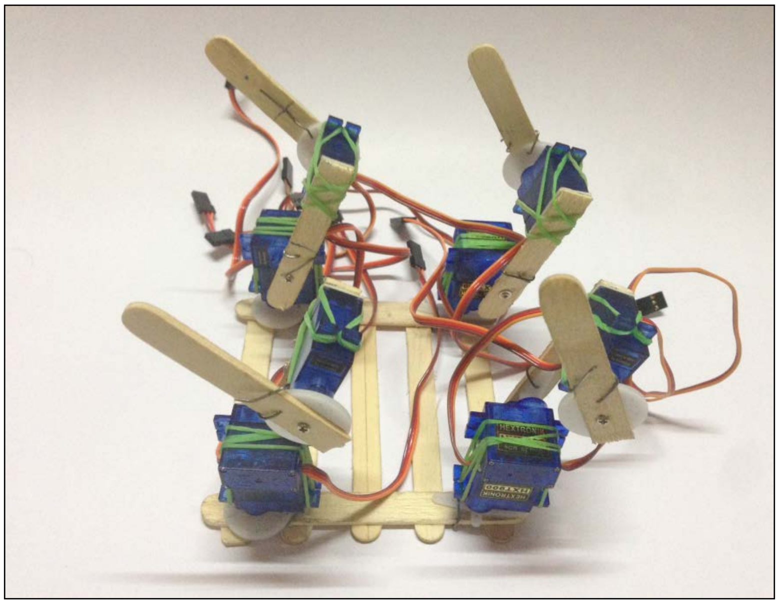


In your case, the arms will look identical. Do not worry about how they look. In this project, I chose different arm shapes to differentiate between the front and back servos. You will soon realize that the number of wires running around will get real confusing, real quick. Now, all we have to do is attach the legs. For our purposes, we are going to make the calf bone (stick) outward. This should give the robot enough balance to stand.

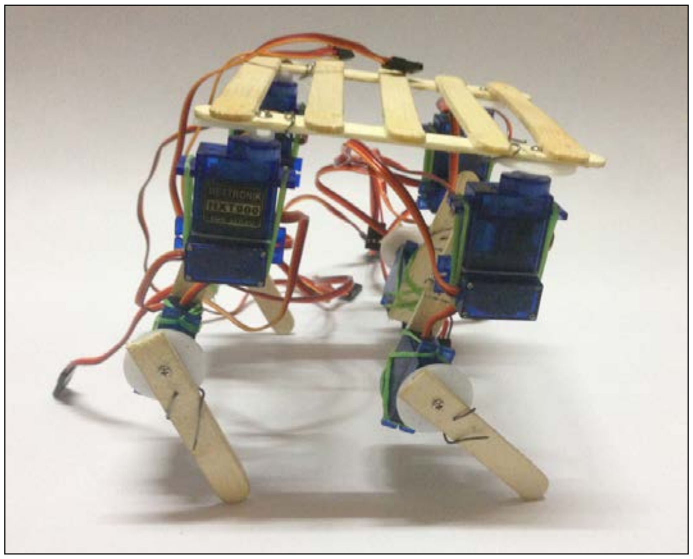
Now, center the servo and attach the leg as shown in the following:



Do this for all the other legs and you'll get the following result:



The following shows the standing position:



If this chassis is built well, it can carry a lot of load, such as this bowl:

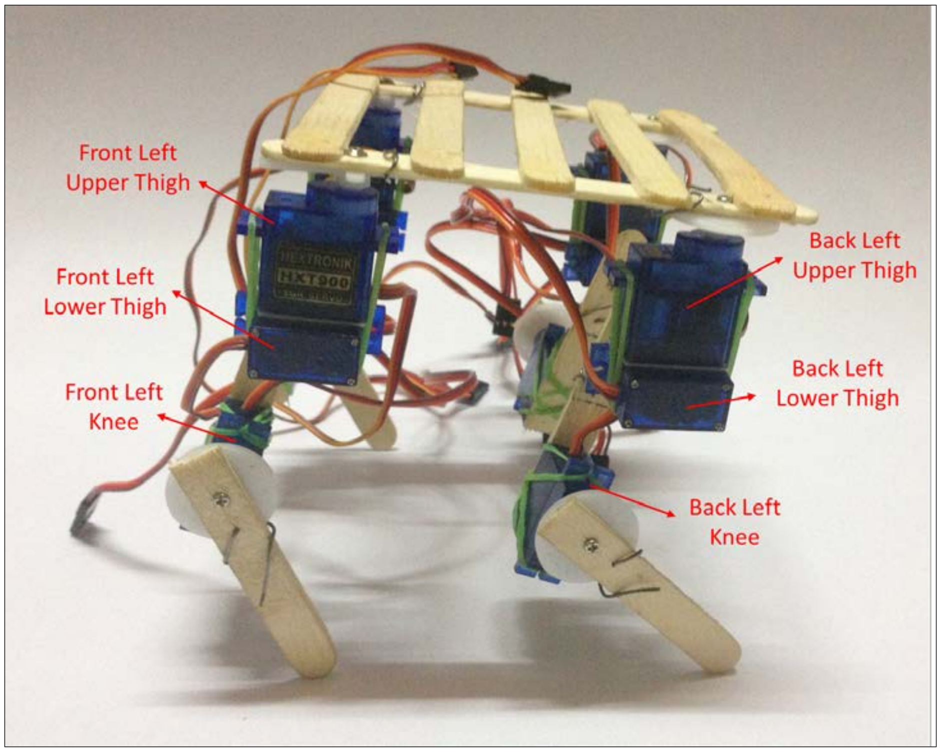
And there we have it: the completed body of the robot.

Completing the circuit

Now that we completed building the chassis (most of it at least), we can move on to the electrical part of the project. This involves connecting all the servos to the Arduino MEGA; this is exactly what this part entails. "However, didn't we already do this in the previous section?" Yes, but to be more effective, we are going to firstly label the servos and create a circuit that drastically decreases the number of wires that we need.

Labeling the servos

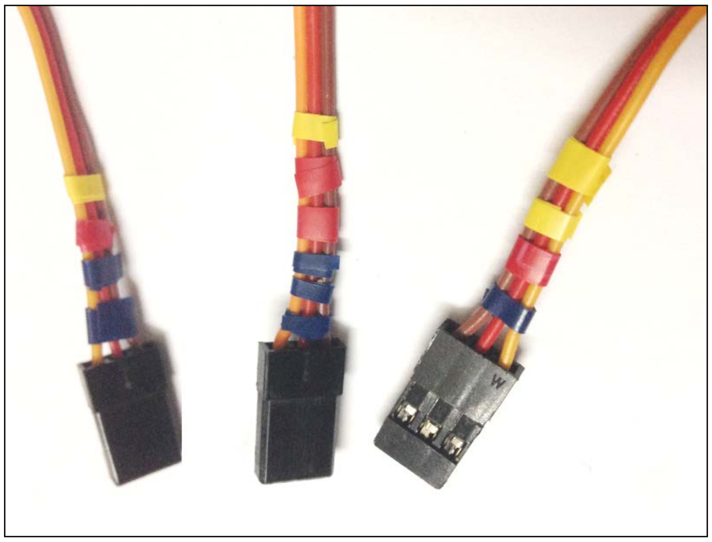
Just before connecting all the servos together to the MEGA, let us make the life of our future selves easier. Let us label each servo so that later we can easily identify what servo wires connect to what servo. To gain an understanding of this, take a look at the following image:



A symmetric labeling will follow on the right-hand side. We will use the following chart to label each servo. The number of (bands refer to the thin strips of the insulation tape that is used as a type of identification):

Servo location	**Color**	**Number of bands**
Upper thigh	Black	1
Lower thigh	Black	2
Knee	Black	3
Front	Red	1
Back	Red	2
Left	Yellow	1
Right	Yellow	2

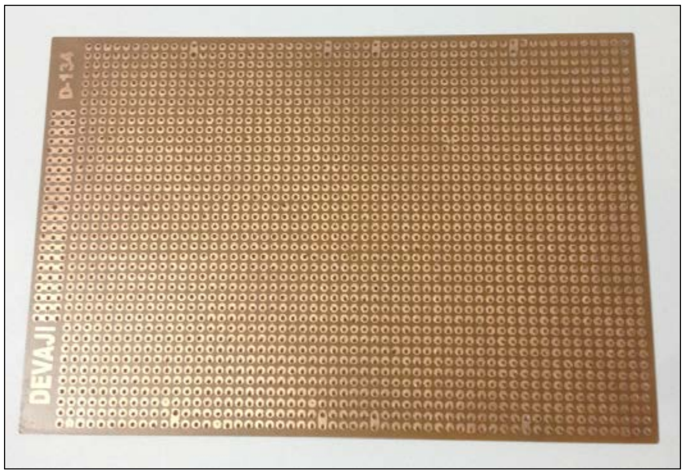
Doing this for the entire servo would look like this:



It might have been a bit frustrating to do this, but once you are done, it will be very easy to identify and replace a faulty servo later. It also aids programming and such.

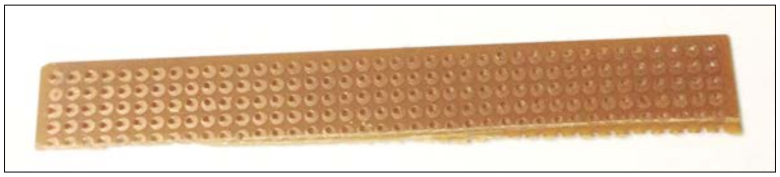
Building a tiny circuit

This is something new. As mentioned at the very beginning of the chapter, we are going to create a circuit to remove the need for the breadboard and drastically reduce the number of wires needed for the servo connections. For this, we are going to use a proto board.

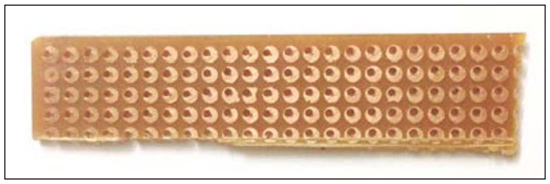


Of course, we do not want such a large proto-board, so we will have to cut it. We will use a 4 x 15 size board, where 4 and 15 represent the number of column holes and row holes, respectively. To cut it, use a straight edge (as shown the following image) and use a blade to cut through the board. It may not cut this way, so turn the board to the other side and repeat it. If it still doesn't cut, use pliers to carefully separate the proto-board into two pieces:

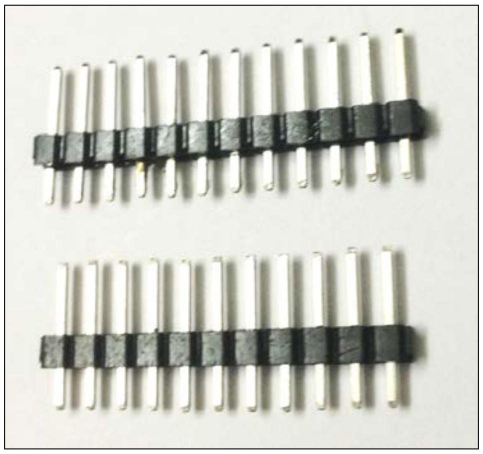
The cut piece will look like this:



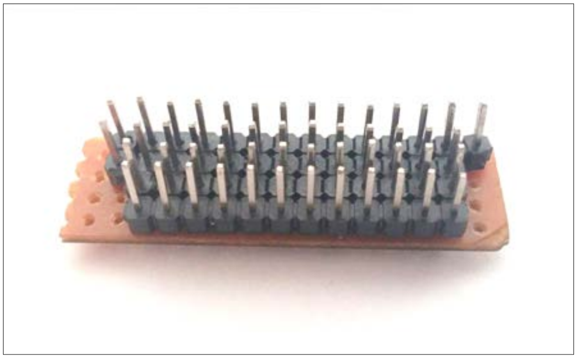
Repeat the step again to decrease the number of rows to 15, as shown in the following image:



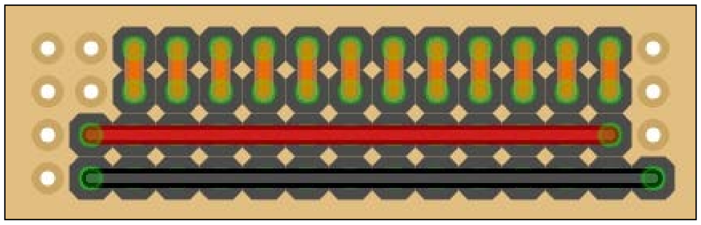
Now, pick up male header strips and snap them into 1x14, 1x13 and 2x12 pin lengths. If you don't have enough to create a set of these numbers, it's alright; you could always merge 7 and 5 pins to make 12:



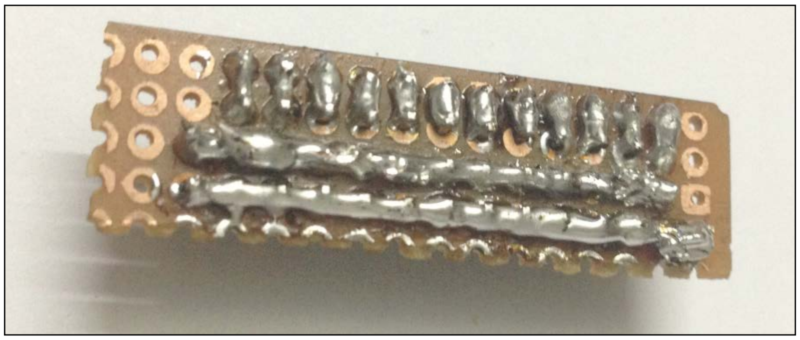
Place them into the 4 x 15 cut proto-board as follows (ensure that the shorter part of the header is on the side, with the little copper circles):



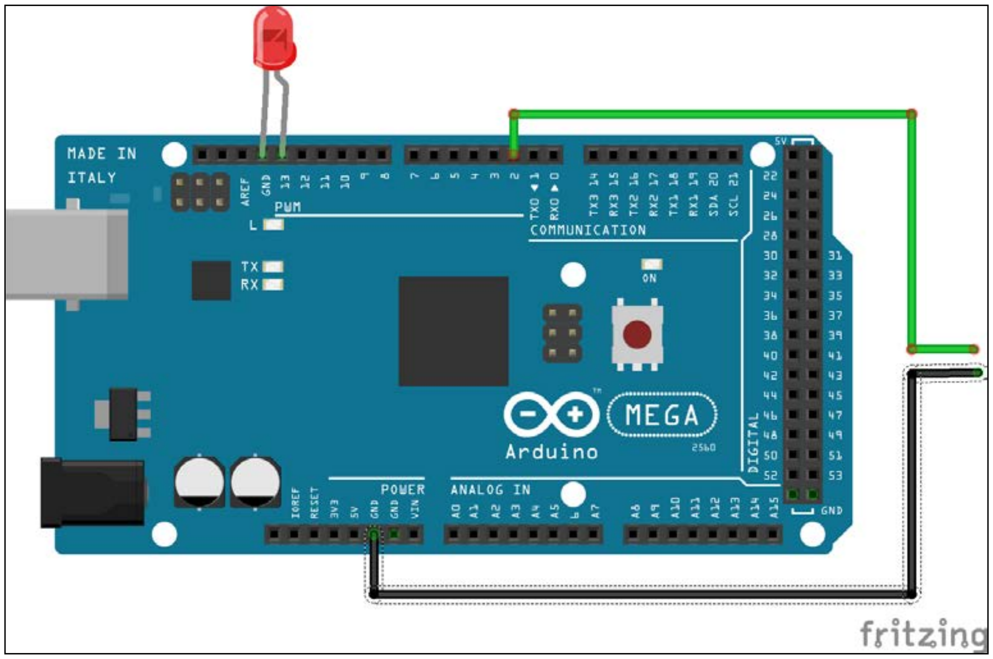
Now, solder the underside (copper circles) in the following manner:



The bottom will look like this:



I hope you understand what we are doing with this tiny circuit. We simply created a more efficient way to connect all the servos through the length of the tiny circuit and connected the battery directly to this board. To test the soldering, you can either use a multimeter set to measure resistances, or use the MEGA. The former is self-explanatory. For the latter, remove all the connections that go to the MEGA and create the following circuit:



Make sure that the two unattached wires that come out of the MEGA have female terminals. Open Arduino and launch the **Button** example that can be found via **File → Examples → 02.Digital → Button**. Then, upload the program.

The LED on pin 13 will stay ON as long as no connection is detected (the circuit is open). Now, use these two wires to go through the tiny circuit to ensure that the soldering is done properly and the connections are as required. Who needs a multimeter, right?

Now that we have built the circuit, we just have to complete the connections.

Putting it all together

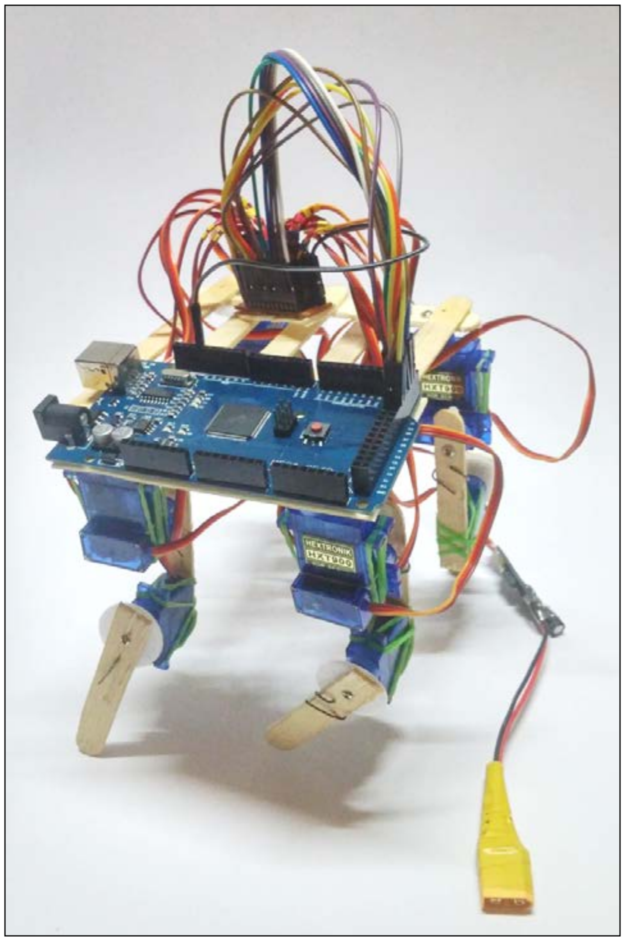
If you haven't yet appreciated the advantage of the tiny circuit, you will now do so. Connect the edge with the 12 pins of the tiny circuit, with 12 male-to-female connecting wires. Just before going to the connections, let's take a look at a few abbreviations that will help us:

- UH: Upper Hip
- LH: Lower Hip
- K: Knee
- F: Front
- B: Back
- L: Left
- R: Right

Now, here is how the rest of the connections will go about with the configuration being components → tiny circuit → Arduino:

Components	**Tiny circuit**	**Arduino**
F R UH	#1	D22
F R LH	#2	D23
F R K	#3	D24
F L UH	#4	D25
F L LH	#5	D26
F L K	#6	D27
B R UH	#7	D28
B R LH	#8	D29
B R K	#9	D30
B L UH	#10	D31
B L LH	#11	D32
B L K	#12	D33
N/A	#13(GND)	GND
UBEC (5V & GND)	#0 (5V & GND)	N/A

Doing so will result in the following setup:



Summary

This chapter was quite something, wasn't it? We planned and built our own chassis for the robot dog using household materials. We painstakingly labeled each servo that our future selves will be grateful for. Then, we created our own tiny circuit, which will help us significantly, by decreasing the necessity of the breadboard and so many extra wires. Finally, we completed the circuit, linking the components, the tiny circuit, and the Arduino MEGA 2560.

In the next chapter, we will program the robot dog to do a series of actions, including standing, sitting, walking, and so on. We will also make some additions such as adding speech control and also adding a personality to the dog.

10
Robot Dog – Part 3

Chapter 10! "Are we there yet?" Finally, yes! In the previous two chapters, we learned about the Arduino MEGA: how to control a servo and supply adequate power; we built the chassis for our dog using ice cream sticks and created a tiny circuit to get rid of the breadboard and a dozen of wires.

In this chapter, we will finally learn to program a complex machine such as the robot dog that we've built, and how to tackle hurdles that we face. The reason why this is so hard to program is because of all the servos that need to be controlled and coordinated in order to produce the right balance and meaningful motions. This chapter is split into the following sections:

- Programming the robot
- Developing a personality
- Implementing speech control

So let us begin part 3 of the robot dog chapters.

Programming the robot

Programming the robot is the hardest task in this book. It is not only about simply writing code and hoping for the best. Technically, you can use that technique and still get away with it, but it would consume a huge amount of time and energy. We will go through the prior art again, see how a dog (and a robot dog) walks, and try to mimic some of that motion. We will also look at the mechanical design of our robot and see how it can be upgraded for better performance.

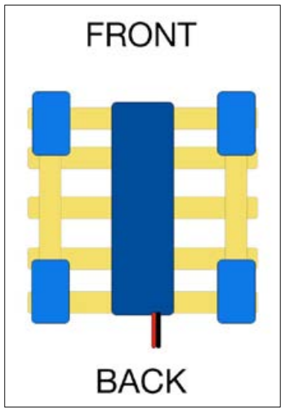
Weight distribution

Funnily enough, it makes sense to put this in the programming section. This is because weight distribution and the creation of the walking motion go hand in hand. Before programming the bot, we will have to power it. Since we have made our circuit in the previous chapter, we just have to put it in place on the robot, such that the weight of the battery and the Arduino MEGA is evenly distributed.

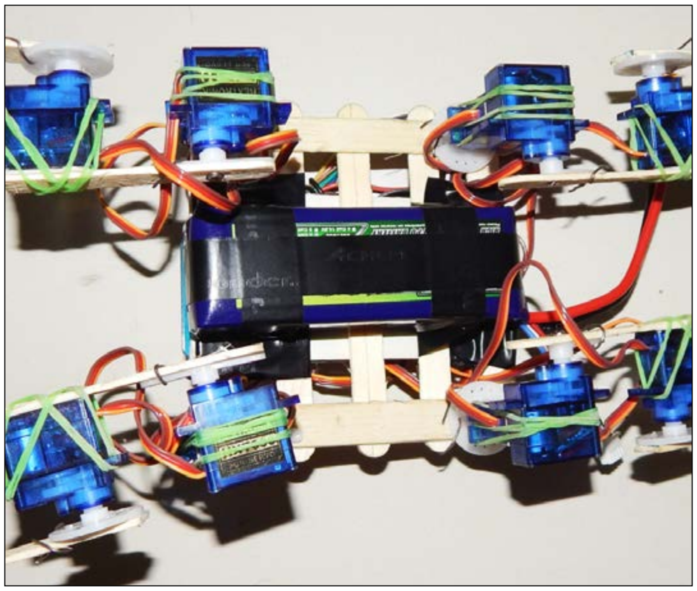
First, let's take a look at the battery (the larger one):

This is the heaviest object that the robot will be lifting. It will significantly increase the difficulty of programming if it is not centered correctly. Putting it over the base of the robot seems like a logical solution, but the issue with the battery is that it occupies so much valuable space that can be used by the circuits, as they need to be more accessible than the battery. This is why we are going to mount it to the bottom of the robot.

Here is a simple diagram of what the battery's position will look like. Remember, this is the bottom view of the robot:

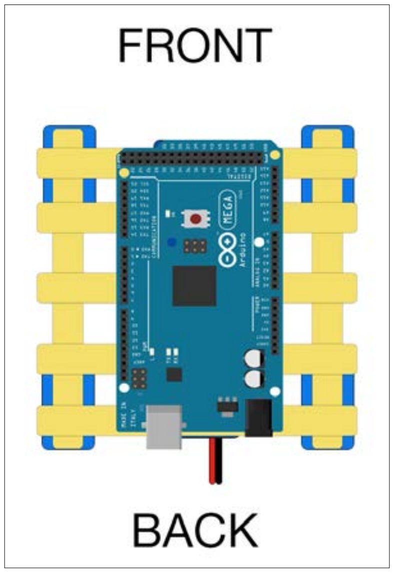


While placing the battery, sort out the servo wires and place them at the corners around the battery. You can mount it using a layer of double-sided tape and then tape to bind it from a place. Ensure that the battery is exactly at the center of the base. A well-bounded one will look like this:



Make a note of how the servo wires are segregated into four parts around the battery. This is to decrease the distance between the servos and the MEGA in order to avoid unnecessary servo-cable extensions. If you think you really need them, especially for the knee servos, feel free to use them.

Arduino MEGA can be mounted directly opposite to it on the top, like this:



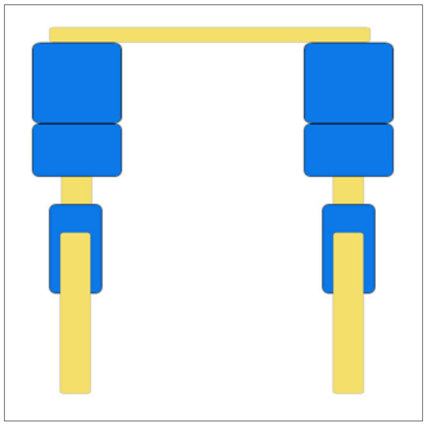
The MEGA can be stuck using a double-sided tape that we had used earlier to cover its base. Again, make sure that the segregated servo wires come out through the corners of the MEGA. For testing purposes, we will leave the rest of the connections as it is for now.

Test one

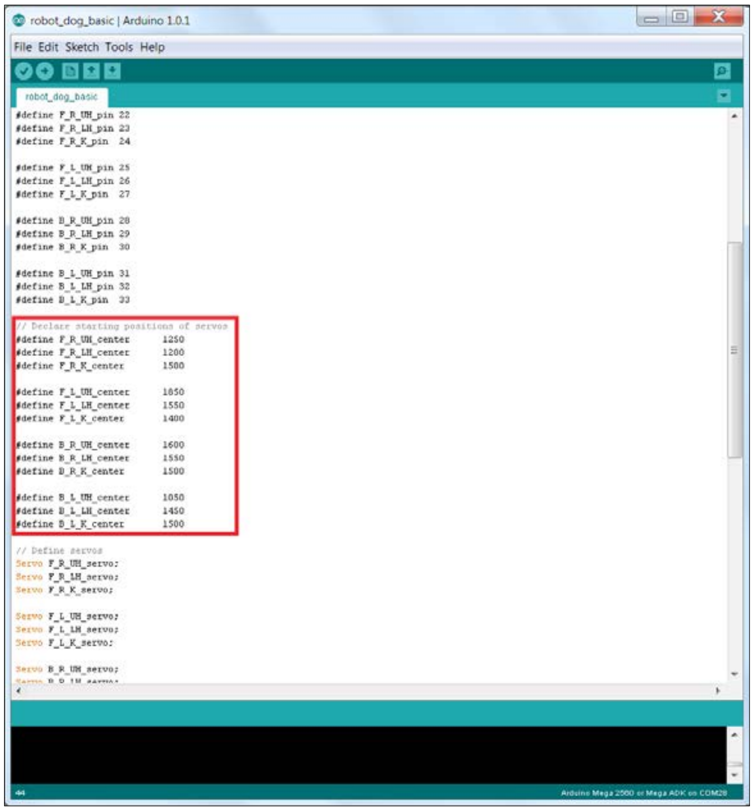
"Test one?" As mentioned before, you will be taught how to make a robot, including the testing and reiteration phases. We are going to run our first test. It is a simple test that checks the stability of the robot and its ability to stand upright. You can connect the USB to the MEGA. Do not connect the battery yet.

Open the `robot_dog_basic.ino` file in Arduino, upload the program, and carefully connect the batteries to the UBEC. Your robot should spring into life and stand in an awkward stance, or not stand at all.

This is what we are trying to achieve (side view):



All the bones (sticks) point straight down. If you didn't achieve this, don't be saddened. Let us look at the code and fix it:


robot_dog_basic | Arduino 1.0.1
File Edit Sketch Tools Help
robot_dog_basic
44
Arduino Mega 2560 or Mega ADK on COM28

```
#define F_R_UH_pin 22
#define F_R_LH_pin 23
#define F_R_K_pin  24

#define F_L_UH_pin 25
#define F_L_LH_pin 26
#define F_L_K_pin  27

#define B_R_UH_pin 28
#define B_R_LH_pin 29
#define B_R_K_pin  30

#define B_L_UH_pin 31
#define B_L_LH_pin 32
#define B_L_K_pin  33

// Declare starting positions of servos
#define F_R_UH_center      1250
#define F_R_LH_center      1200
#define F_R_K_center       1500

#define F_L_UH_center      1850
#define F_L_LH_center      1550
#define F_L_K_center       1400

#define B_R_UH_center      1600
#define B_R_LH_center      1550
#define B_R_K_center       1500

#define B_L_UH_center      1050
#define B_L_LH_center      1450
#define B_L_K_center       1500

// Define servos
Servo F_R_UH_servo;
Servo F_R_LH_servo;
Servo F_R_K_servo;

Servo F_L_UH_servo;
Servo F_L_LH_servo;
Servo F_L_K_servo;

Servo B_R_UH_servo;
```

The marked part is very important. The numbers that represent the starting position of the servos used in this chapter and the number you have to use will be different. Try changing one of the numbers, say `F_R_UH_center` (Front Right Upper Hip) right now, and uploading the code. Notice what happens? Tweak the numbers until your robot looks like the required stance shown in the preceding image of the robot.

If you haven't got it standing yet, keep trying until you get it. You can proceed once you are done.

Let's take a look at the robot now. The whole thing looks balanced, but it seems as though something is not perfectly right. It looks like the back legs are taking the majority of the load. This is because we had attached the back hip servos in order to resemble very much like a dog. A dog has more muscles to help balance. However, this slight offset we have here will give us a lot of trouble in the future. *Why didn't you tell me this while we were putting the thing on?* It's simple: nobody knows how to create the perfect project plan. It requires choosing a design and hoping for the best, and reiterating the design on the way.

Unhook the battery and the USB. Unhook all the back servos going to the tiny circuit. Unscrew the back hip servos only. Exchange the left and right legs. Screw the exchanged hips back. Relabel the servos (remove one yellow band from the then right servos and put them onto the current right servos). Connect them accordingly to the tiny circuit following this components → tiny circuit → Arduino:

- B R UH → #7 → D28
- B R LH → #8 → D29
- B R K → #9 → D30
- B L UH → #10 → D31
- B L LH → #11 → D32
- B L K → #12 → D33

Make sure the connections are correct and tight, connect the USB and the battery and check the position the dog now stands in. You may have to change the 'servo center' values to get the desired position.

Save this file or at least the center values, as we will be using it from now on. The dog is much more stable now with the weight equally distributed among the four legs. We can now move to the next step: teaching the robot how to walk.

Try using the same technique to create a sitting position. Save this position and use these numbers when we come across the sitting position again.

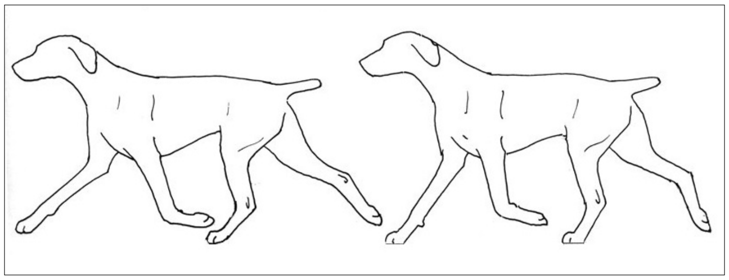
The walking gait

Programming the robot's walk is similar to teaching a baby to walk, except that the robot does not have consciousness. A walking gait is basically a sequence of motions that results in the walking action. There are two common gaits: the 'creep/crawl' gait and the 'trot' gait.

The creep gait is what you see a cat doing when it is stalking its prey.

A video (`https://www.youtube.com/watch?v=wQsmsr0oR6c`) would do this more justice, but even from the preceding image or your experience, you can observe how the cat keeps its body constantly horizontal. This is a highly energy-efficient walking gait, but it is very complex to engineer.

The trot gait is when the two diagonal legs move together at a time to create a walking motion, as shown in the following image:



This is not as highly energy-efficient as the creep gait, but is far simpler than the creep gait. Also, this has more static stability, while the creep gait has more dynamic stability. Since you are programming a quadruped robot for the first time, let's take it easy. We will choose the trot gait and slightly modify the motions to make it suitable for our robot.

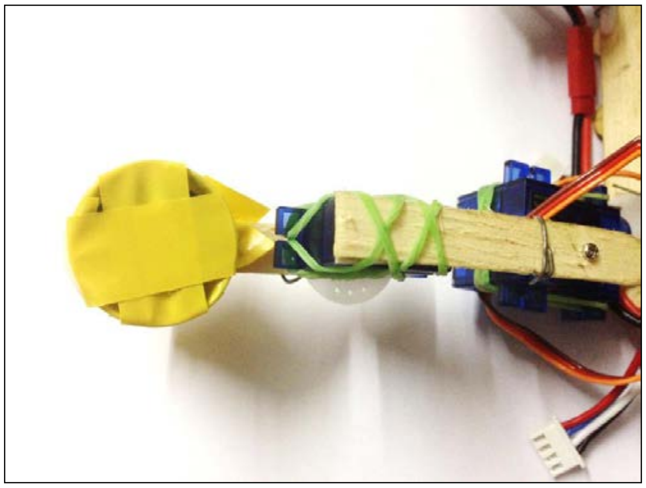
Test two

We are going to program the robot, using the trot gait. Open the `robot_dog_trot_gait.ino` file in Arduino. Before uploading it, keep the robot in a space where it has no obstacles. A long USB cable benefits here. Change the servo center values to what you had saved in the previous subsection. Plug in the battery and upload the program. Did it move forward or did it fall? Even if you executed a perfect walk, you can glance through the next bit. This may aid you later.

If your robot fell, why do you think this happened? We were sure of the balance. What is going wrong in that case? The ends of the calf bones (sticks) are too thin to cover the area that would further improve stability and weight distribution along the legs. A good way to solve this is using plastic bottle caps. As mentioned earlier, these household materials are easily available.

Again, unplug the USB and the battery. It is not necessary to remove the knee screws to remove the calf stick, but if this is more convenient, go for it. We will need eight bottle caps. If you think four will suffice, then you can choose 4 and run the code again. Mark the center of the two bottles with the thumb pin you used earlier, and push the pin through to make a tiny hole. Tape the bottom part of the calf stick a couple of times to create a washer-like material. Now make a small hole, 1 cm above the bottom of the calf stick.

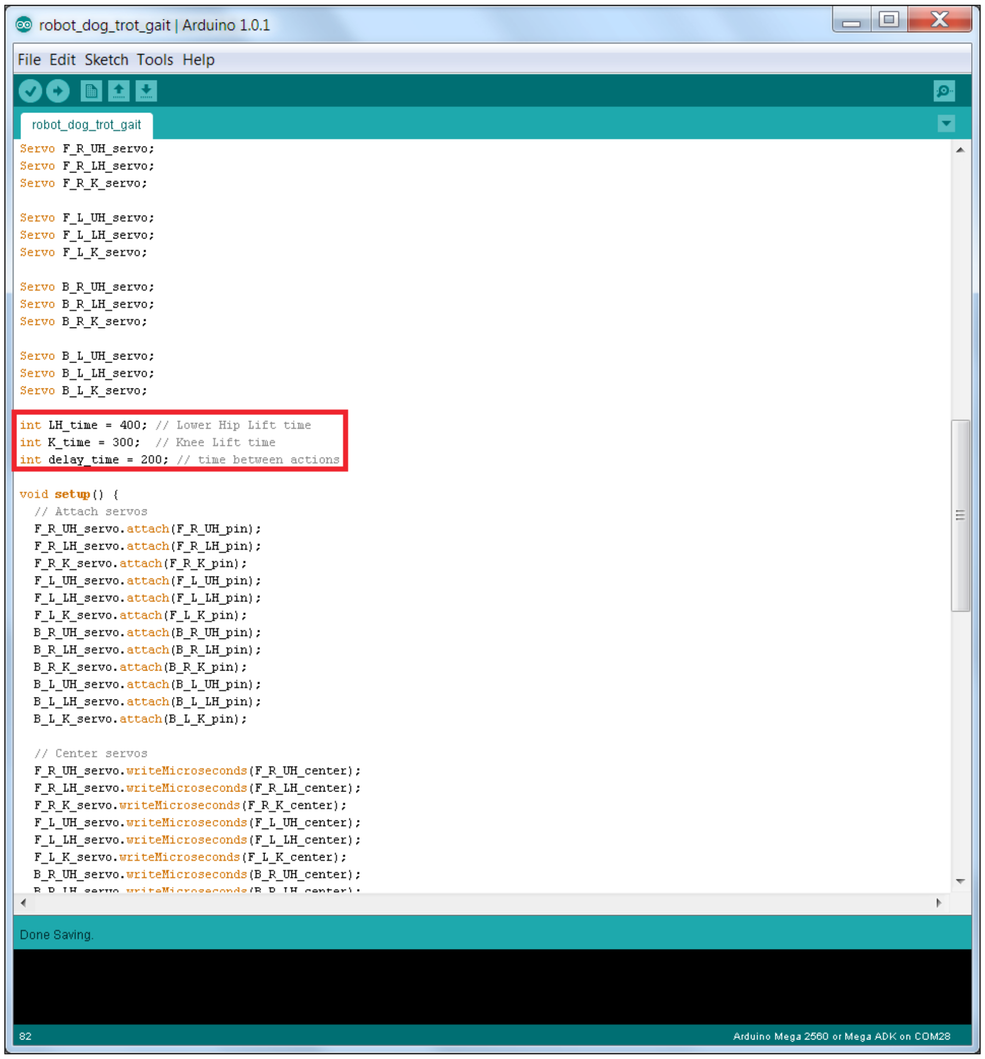
Now, with one cap on both the sides of the calf stick, drive a screw through the thickness of all the three layers. It is recommended to tape the exterior of the bottle caps to improve traction. It will look something like this:



Do this for all four limbs.

Once done, connect the battery and then the USB cable. You can alternatively use the smaller LiPo battery to power the Arduino at this stage via the 2.1 mm power port on the Arduino MEGA. This would be more convenient because of the length constraint of the USB cable.

It walks, doesn't it? It doesn't fall either, right? It is cool, right? If it does fall, try tweaking these parameters:


robot_dog_trot_gait | Arduino 1.0.1
File Edit Sketch Tools Help
robot_dog_trot_gait

```
Servo F_R_UH_servo;
Servo F_R_LH_servo;
Servo F_R_K_servo;

Servo F_L_UH_servo;
Servo F_L_LH_servo;
Servo F_L_K_servo;

Servo B_R_UH_servo;
Servo B_R_LH_servo;
Servo B_R_K_servo;

Servo B_L_UH_servo;
Servo B_L_LH_servo;
Servo B_L_K_servo;

int LH_time = 400; // Lower Hip Lift time
int K_time = 300;  // Knee Lift time
int delay_time = 200; // time between actions

void setup() {
  // Attach servos
  F_R_UH_servo.attach(F_R_UH_pin);
  F_R_LH_servo.attach(F_R_LH_pin);
  F_R_K_servo.attach(F_R_K_pin);
  F_L_UH_servo.attach(F_L_UH_pin);
  F_L_LH_servo.attach(F_L_LH_pin);
  F_L_K_servo.attach(F_L_K_pin);
  B_R_UH_servo.attach(B_R_UH_pin);
  B_R_LH_servo.attach(B_R_LH_pin);
  B_R_K_servo.attach(B_R_K_pin);
  B_L_UH_servo.attach(B_L_UH_pin);
  B_L_LH_servo.attach(B_L_LH_pin);
  B_L_K_servo.attach(B_L_K_pin);

  // Center servos
  F_R_UH_servo.writeMicroseconds(F_R_UH_center);
  F_R_LH_servo.writeMicroseconds(F_R_LH_center);
  F_R_K_servo.writeMicroseconds(F_R_K_center);
  F_L_UH_servo.writeMicroseconds(F_L_UH_center);
  F_L_LH_servo.writeMicroseconds(F_L_LH_center);
  F_L_K_servo.writeMicroseconds(F_L_K_center);
  B_R_UH_servo.writeMicroseconds(B_R_UH_center);
  B_R_LH_servo.writeMicroseconds(B_R_LH_center);
```

```
Done Saving.
```

What this program does in sequence and repetition is the following:

- Raise the front-right leg and advance it
- Raise the back-left leg and advance it
- Center the above servos, raise and advance the front-left leg, pushing the robot forward
- Raise the bottom-right leg and advance it
- Center other servos pushing the robot forward

There you have it: the 'trot gait'.

This was the major part of programming the robot.

Developing personality

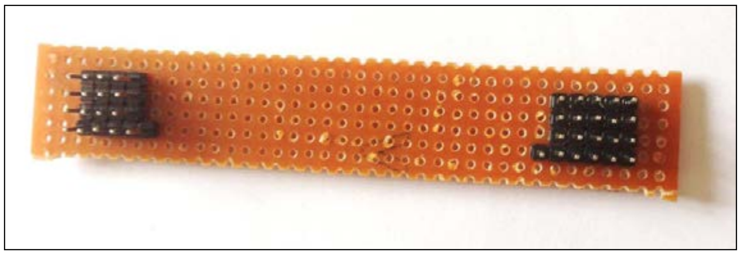
The personality is what makes the robot more alive. Adding personality to it makes it more fun and likable. This section focuses on some ideas that will help you create your own unique personality for your robot.

Circuit upgrade

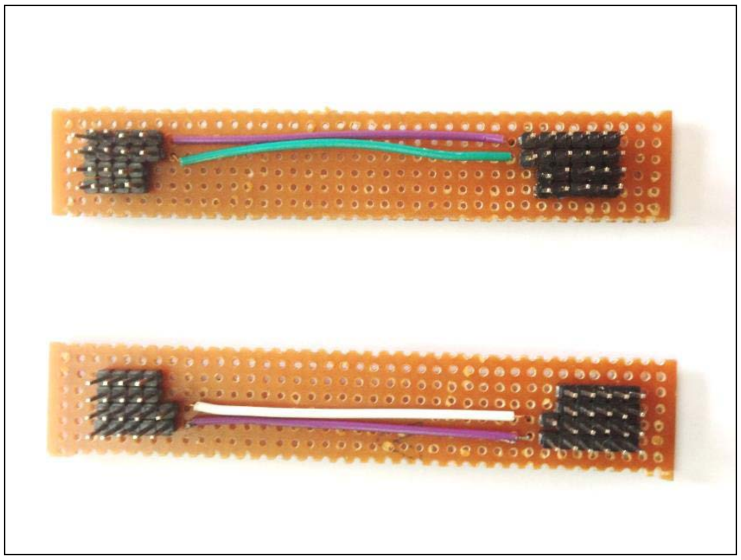
The tiny circuit that we made can be split into two; one along each side of the Arduino connected to a common GND and power. Additionally, more header pins can be added to accommodate more servos (the tail and head) in order to organize the wires better.

First, for this, cut out a long circuit board with a row size of about 6 pins.

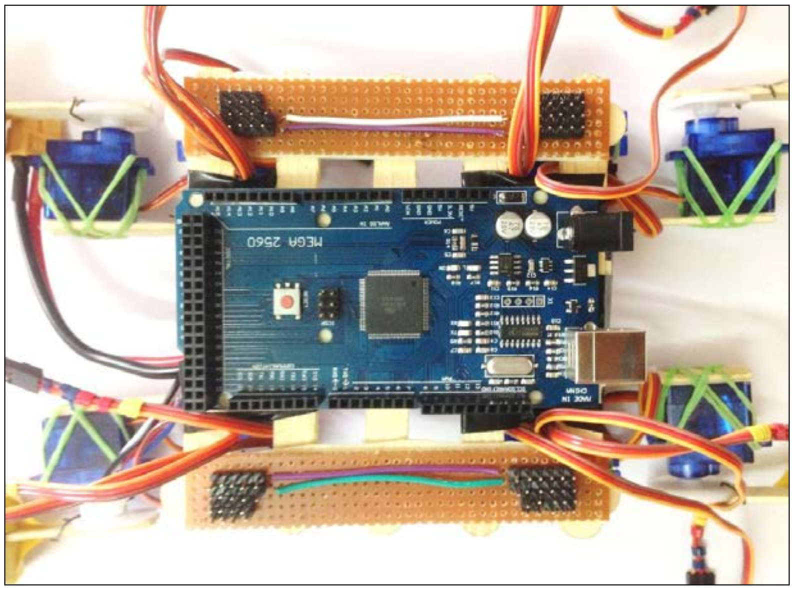
Following the previous circuit's design, create this circuit:



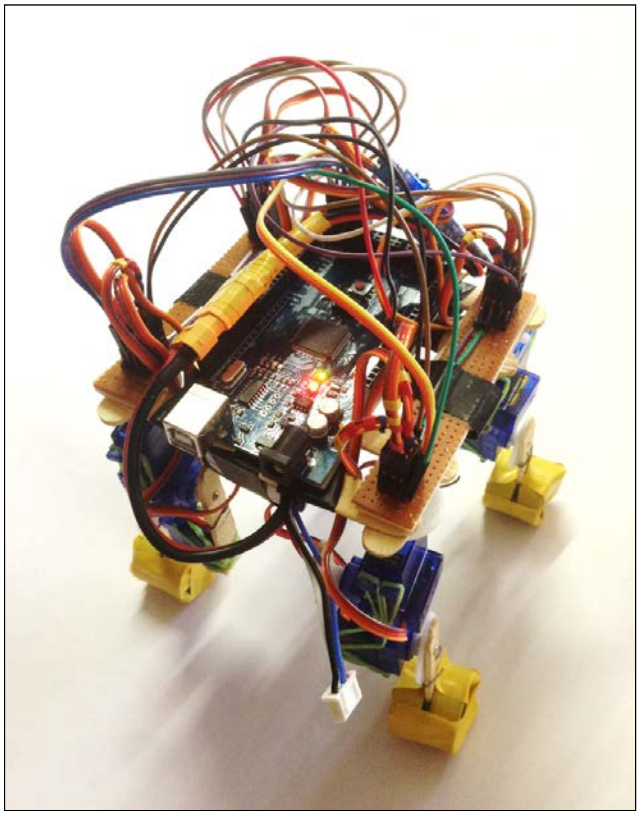
To link the left and right side, small pieces of wires are soldered; one for the GND and another for the power, like this:



Attach these circuits on either sides of the Arduino MEGA in this fashion:



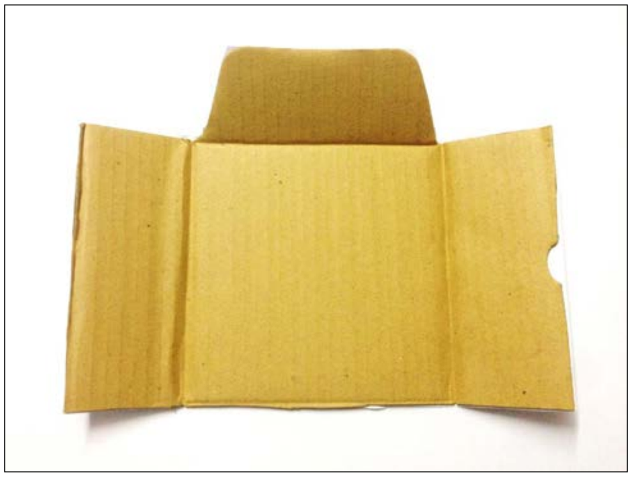
The finished connections will look like this:



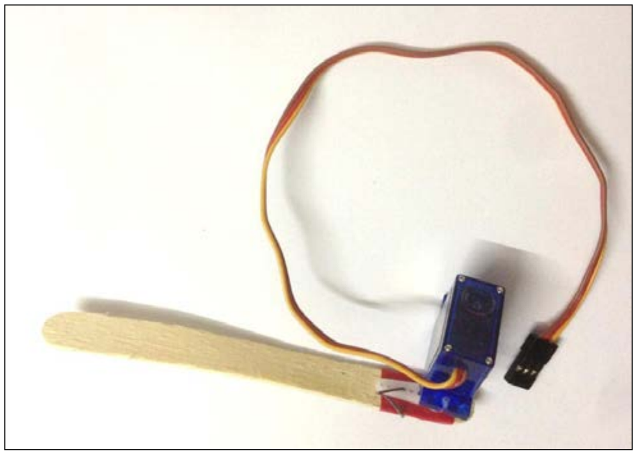
Body upgrade

To be perfectly honest, the robot doesn't resemble a dog at all. Sure, it walks on four feet, but it doesn't resemble a robot dog either. This would require an outer shell with a head and a tail.

Before installing a tail, it's a good idea to create an additional base layer or a case above all the wire commotion. This can either be made of more ice cream sticks, or you can just modify a cardboard box that you happen to have lying around to fit the robot, like this:



This new base can accommodate the tail. The tail can be made using another ice cream stick. It needs to be attached to a servo stuck underneath the base at the back.



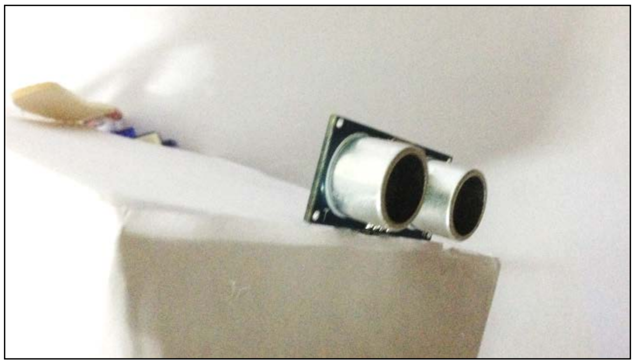
The servo can be programmed to create a wagging effect. You can also attach two servos that would allow you to control the shape of the tail. The tail can be used to express emotions (a wagging tail is a happy tail).

You can additionally attach a make-shift head that would make it seem much livelier. However, in this chapter, we will use something that does give it a face, but not a head.

Sensors

The reason why adding sensors, such as the SRF04 (or SRF04) sensor that you used earlier to measure the distance, is such a great idea is because it enables the robot to get a feel of the environment—only with sensors can it respond to stimuli.

To attach the sensor, make a slit on the top-front part of the base to accommodate the HC SRF04. Place the sensor at the front, as shown in the following image:



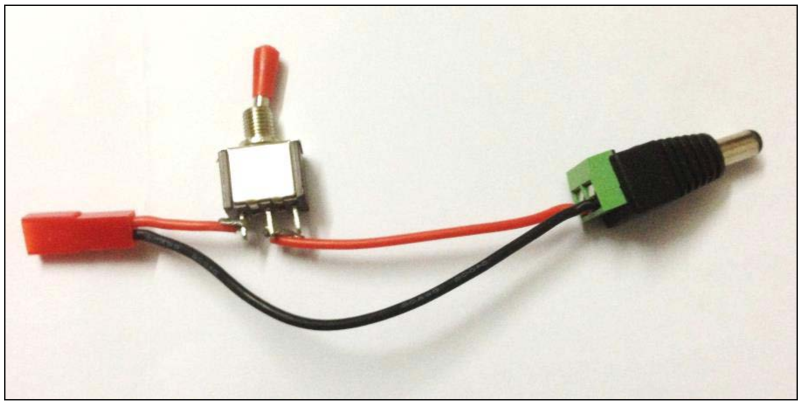
Attach this in place. When the time comes, connect the sensor using the following circuit (SRF04 → Arduino):

- GND → GND
- Trig → D12
- Echo → D11
- VCC → 5V

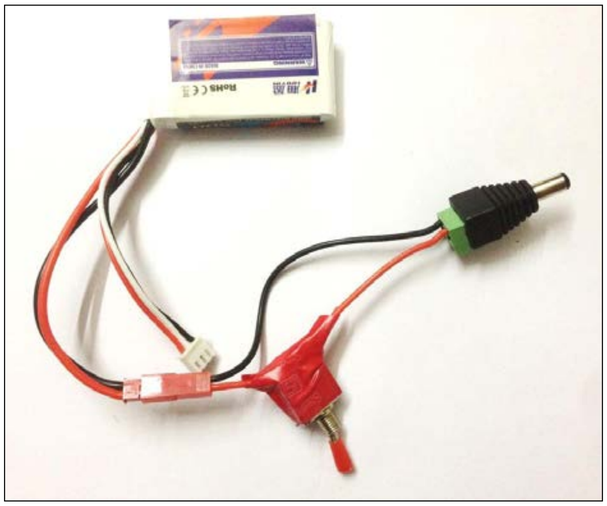
The switch

Finally, we will connect the secondary battery. To conserve its power, we will make use of a switch like this:

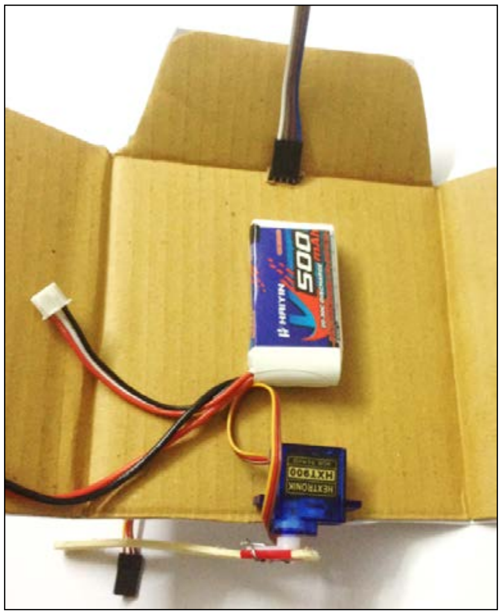
Connect the battery's connector to the switch, like this:



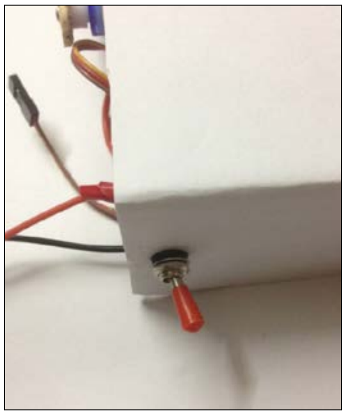
Attach the battery and the setup will look like the following image:



Once you have attached the switch to the battery, attach the battery on the underside of the case. Make sure that you stick it at the exact center of the case:

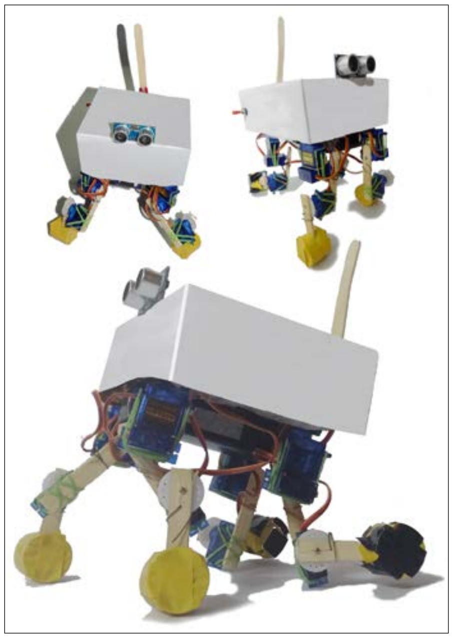


Make a small hole on the side where the wires of the battery are, and attach the switch to it:



Once you are satisfied with the components inside the case, connect all the wires including those for the sensor. Connect the tail to D34. Then, attach the case to the body of the robot.

This is what the final product will look like:



Of course, yours may look entirely different, but that is completely fine.

Coding the personality

Just like the external design of the robot, you can program your robot's unique personality too. Since we have attached the sensor as the head of the robot, we can make use of its capabilities. To give you an example, let us make a 'lazy but loving' robot. Let us briefly discuss the algorithms involved:

- Dog just sits around
- Keeps sitting if not disturbed
- If disturbed (detects a hand in front of its sensors), it stands up and wags its tail
- If the hand slowly moves back, the robot follows the hand
- Ultimately, if the hand comes close enough, the robot will lift its leg to shake its hand with you

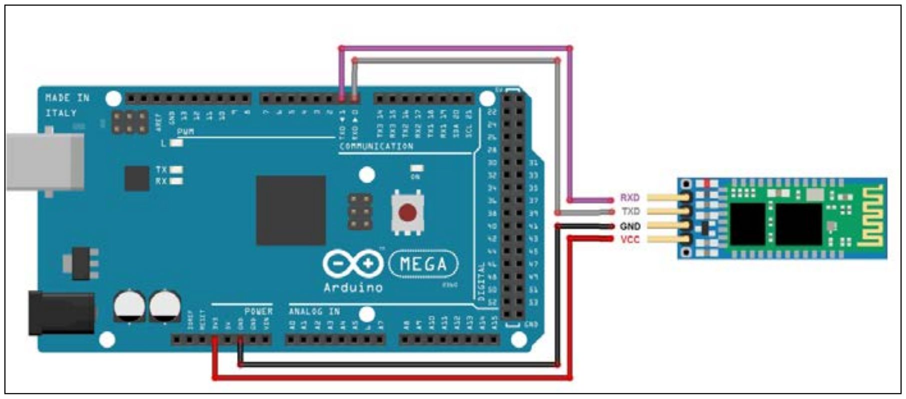
Load the `arduino_robot_dog_personality.ino` file, and go ahead and play with it. It is important to note that this program uses both a tail and the ultrasound sensor. If you do not want to use these components, tweak the code accordingly.

Implementing speech control

We have already learned about Bluetooth and speech control in the previous chapters. In this chapter, we are going to merge them.

Connecting the HC-06 module

Just like in the burglar alarm project (Chapters 4 and 5), hook up the Bluetooth to the Arduino MEGA in the following manner:



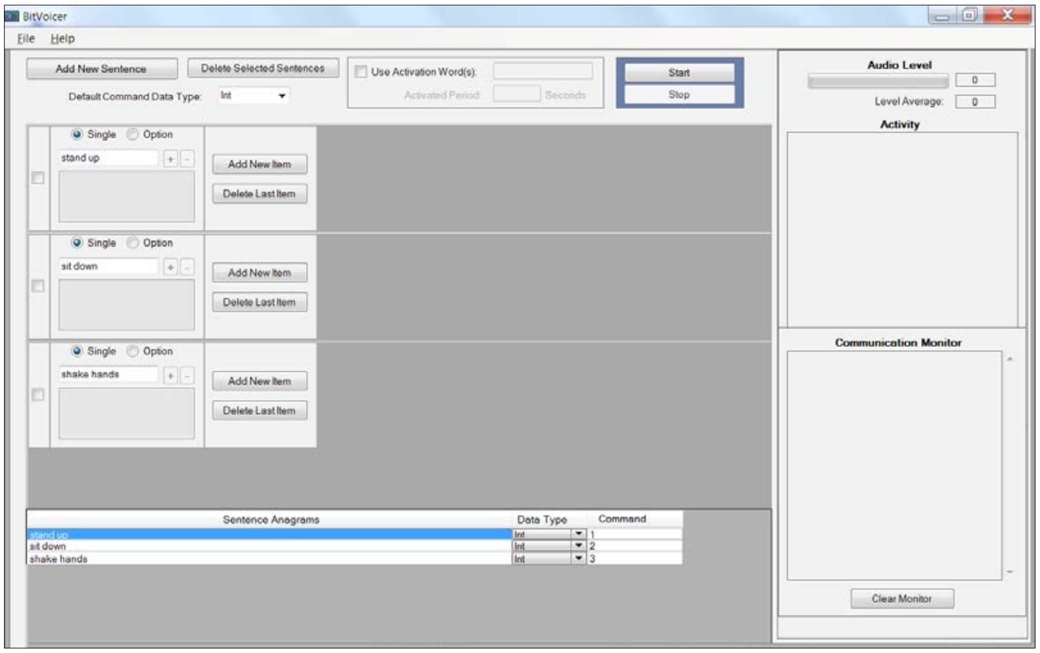
Make sure you are connecting the Bluetooth module to the 3.3V of the MEGA, and not the 5V. Leave the rest of the servo wiring as it is.

Programming the Arduino

Now, open the `robot_dog_speech.ino` file. Just like before, change the servo center parameters to match your robot. Upload the program and connect the computer to the Bluetooth, using the instructions given in Module 1 *Chapter 5*, *Burglar Alarm – Part 2*.

Setting up BitVoicer

Open up BitVoicer and use the commands as shown in the following screenshot:



Default Command Data Type: Int

stand up

sit down

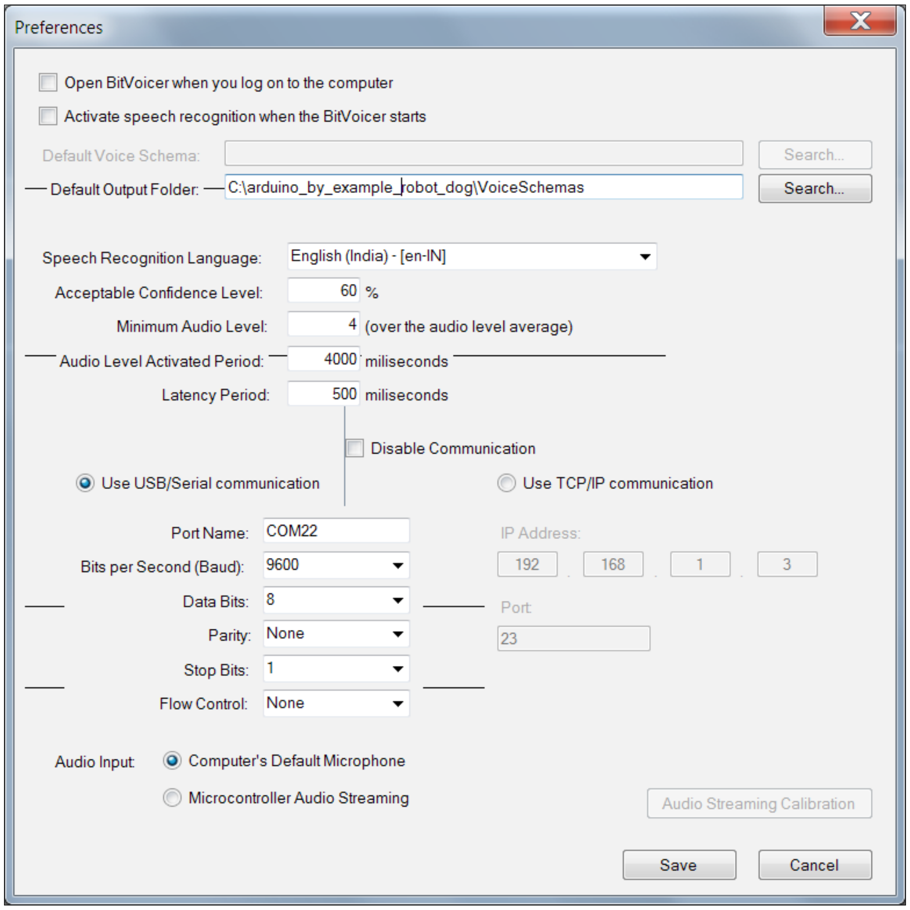
shake hands

Sentence Anagrams	Data Type	Command
stand up	Int	1
sit down	Int	2
shake hands	Int	3

In summary, the commands are as follows (integer type):

- Stand up – 1
- Sit down – 2
- Shake hands – 3

Change the preferences of BitVoicer to match this image:



Preferences

- [ ] Open BitVoicer when you log on to the computer
- [ ] Activate speech recognition when the BitVoicer starts

Default Voice Schema:

Default Output Folder: C:\arduino_by_example_robot_dog\VoiceSchemas

Speech Recognition Language: English (India) - [en-IN]

Acceptable Confidence Level: 60 %

Minimum Audio Level: 4 (over the audio level average)

Audio Level Activated Period: 4000 miliseconds

Latency Period: 500 miliseconds

- [ ] Disable Communication

[✓ Use USB/Serial communication] Use TCP/IP communication

Port Name: COM22

Bits per Second (Baud): 9600

Data Bits: 8

Parity: None

Stop Bits: 1

Flow Control: None

IP Address: 192 . 168 . 1 . 3

Port: 23

Audio Input: [✓ Computer's Default Microphone] Microcontroller Audio Streaming

Note that the COM ports for you will be different. You can look it up in the device manager.

Once everything is set, run the BitVoicer schema. Now, you can literally talk to your robot. It won't talk back, but you can make it sit, walk around, shake your hand, or whatever else you want to program. You are free to add additional functionalities via speech.

Unfortunately, you will still need the computer to communicate with the robot. Making the project standalone will require an additional microcontroller, which is beyond the scope of this book. For now, you can, however, use a wireless headset to communicate with the robot wirelessly.

Summary

If you have reached this point, pat yourself on the back. This chapter must have been very exhausting with so many different problems to deal with. However, I hope that at the end, you have successfully managed to get the robot to move the way you wished.

In this chapter, we programmed the robot dog. To further aid our programming trials, we had to change the chassis a bit. We had to do a lot of testing to get things working the way we wanted. While doing so, we learned about the many aspects (and issues) of building a project from scratch. Once we were happy with the basic walking gait, we upgraded the robot. We added a case, tail, and sensor. Using these elements, we created a simple personality with a huge potential for expansion. Finally, we implemented the speech recognition that allowed us to control the robot using our speech. Funnily enough, to complete the revision of the previous chapters, we can also use a relay to act as a switch for the larger battery powering the servos, which can be controlled by the Arduino.

Now, we have reached the end of the book. If you've reached here, you have learned a lot about Arduinos. I hope that the example-based layout of this book, which aids in direct translation of thought from the author to the reader, was helpful. I would like to thank you for having the patience and commitment to go through this book. The only thing left to do now is use the knowledge you have gained from this book and use your creativity to make wonderful creations. Go on, I'm not going to stop you.

Module 2

Arduino BLINK Blueprints

Get the most out of your Arduino to develop exciting and creative LED-based projects

1
Project 1 – LED Night Lamp

In Module 1 *Chapter 1, Getting Started with Arduino*, you learned about the "Hello World" of physical computing. Now, as you have basic knowledge of Arduino and its IDE, we can go ahead with some cool stuff to do with controlling LEDs. The following topics will be covered in this chapter:

- Introduction to breadboard
- Controlling multiple LEDs
- LED fading
- Creating a mood lamp
- Developing an LED night lamp

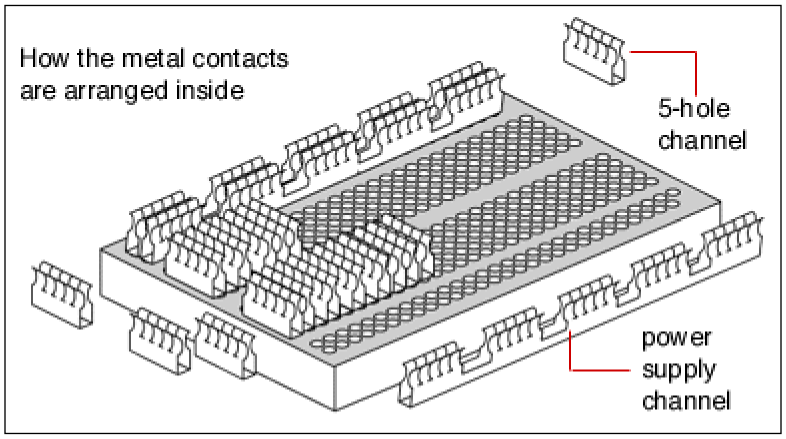
By the end of this chapter, you will be able to control different LEDs in an artistic approach.

Introduction to breadboard

Prototyping is the process of testing out an idea by creating a preliminary model from which other products can be developed or formed. The breadboard is one of the most fundamental pieces for prototyping electronics circuits. As it does not require any soldering, it is also referred to as a "solderless board".

Structure of a breadboard

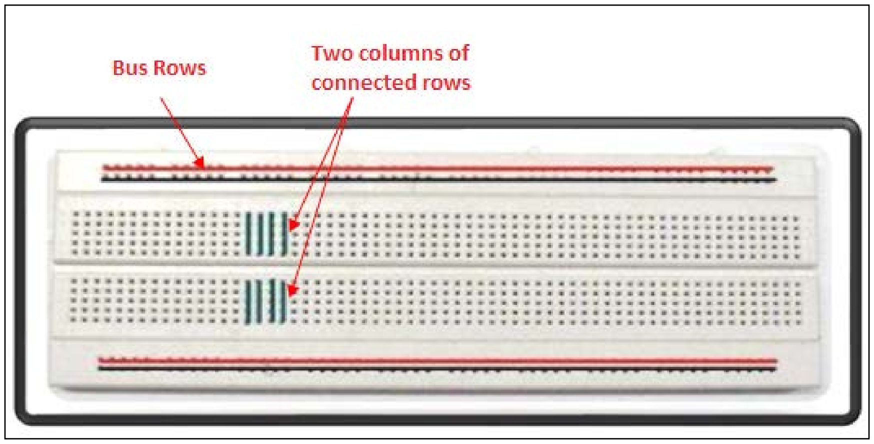
Almost all modern breadboards are made up of plastic. A modern breadboard consists of a perforated block of plastic with numerous metal clips under the perforations. The breadboard has strips of metal underneath the board and holes on top of the board. In the following image, you can see the structure of the breadboard:



The main structure of the breadboard is made up of a main central area, which is a block of two sets of columns, where each column is made up of many rows. All of these rows are connected on a row-by-row basis.

Using a breadboard

The breadboard has many strips of copper beneath the board that connects the holes as shown (short circuited or same potential). The upper blue lines are not connected with the lower ones. In the case of electronic circuits, a power supply is required at various pins. So instead of making many connections with a power supply, one can give a power supply to one of the holes on the breadboard and can get a power supply from its outer holes:



Multiple breadboards can be connected together to form larger prototyping board experimenting areas. If you want to use any chip/IC (integrated circuit), you can place it with one side on the upper block and the other side on the lower block of the breadboard. It will be very easy for us to understand the breadboard, as we will be using the breadboard extensively for all our projects.

Controlling multiple LEDs

In the embedded world, controlling a single LED is "Hello World" code, which we learned in first chapter. Now, as we are familiar with the concepts of LED, we can start to control multiple LEDs with Arduino. Here, we will start by making a simple traffic light module.

Simple traffic light controller

As we all know, a traffic light is made up of three LEDs: red, yellow, and green. To make this project, we need red, green, and yellow LEDs, strip wires, and a few 255 Ω resistors.

In the previous chapter, in our "Hello World" program, we connected an LED directly with pin 13. Here, we will connect red, yellow, and green LEDs with pins 9, 10, and 11 respectively. In the case of pin 13, it has an in-built pull up resistor. Pull up resistors are used to limit the current supplied to an LED. We can't give current of more than a few mA to LEDs. But, with pin 13, current is itself limited in Arduino by the internal pull up resistor. If we want to connect to pins other than 13, we need to add resistors in the circuit.

Connect the longer head of the red LED to pin 9 and the shorter head to the resistor and then the resistor to ground. Make a similar connection with the yellow and green LEDs to pins 10 and 11 respectively.

Once you have made the connection, write the following code in the editor window of Arduino IDE:

```
// Initializing LEDs for using it in the code.
// Initializing outside any function so global variable can be
accessed throughout the code

int redLed = 9;
int yellowLed = 10;
int greenLed = 11;

// the setup function runs once when you press reset or power the
board
void setup() {
  // initialize digital pin for red, yellow and green led as an
output.
  pinMode(redLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(redLed, HIGH);       //    Making red led high
  digitalWrite(yellowLed, LOW);     //    Making yellow led low
  digitalWrite(greenLed, LOW);      //    Making green led low
  delay(10000);                     //    Wait for 10 seconds (10000
milliseconds)
  digitalWrite(redLed, LOW);
 //    Making red led low
  digitalWrite(yellowLed, LOW);     //    Making yellow led low
  digitalWrite(greenLed, HIGH);     //    Making green led high
  delay(10000);                     //    Wait for 10 seconds (10000
milliseconds)
  digitalWrite(redLed, LOW);        //    Making red led low
  digitalWrite(yellowLed, HIGH);    //    Making yellow led high
  digitalWrite(greenLed, LOW);      //    Making green led low
  delay(3000);                      //    Wait for 3 seconds 3000
milliseconds)
}
```

In the preceding code, you can see that it is much the same as the "Hello World" program, except here we are controlling multiple LEDs. Here, we are initializing the variables as integers and using that same variable throughout the code. So, in the future, if we need to change an Arduino pin, we just have to make a change at one place, instead of making changes at multiple places in the code. It is a good practice to use variables instead of directly using pin numbers in the code. In the `setup` function, we are setting pins as OUTPUT. If we don't initialize the port to either INPUT or OUTPUT, a port might be in an indefinite state. So, it will give random output.

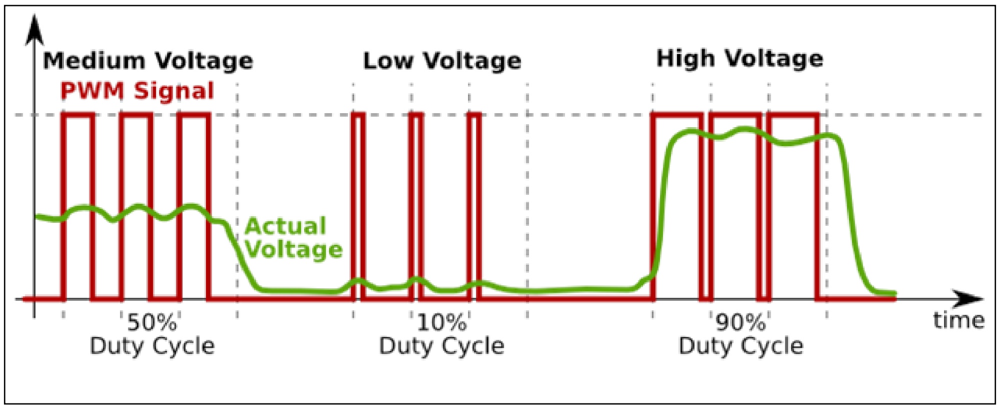
We have completed the code for one direction of our traffic light. Similarly, you can create your code for the other remaining directions.

LED fading

You can fade out and fade in the light of an LED using Arduino's `analogWrite(pin, value)` function. Before we get into using the `analogWrite()` function, we will understand the concept behind the `analogWrite()` function. To create an analog signal, Arduino uses a technique called **Pulse width modulation** (**PWM**).

Pulse width modulation (PWM)

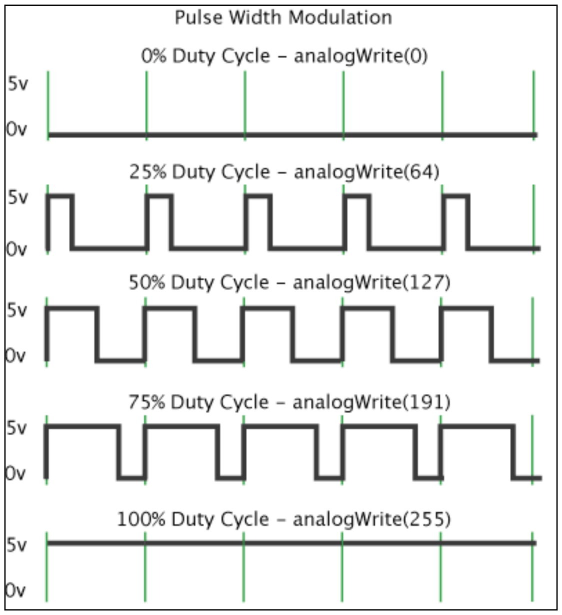
PWM is a technique for getting an analog signal using digital means. By varying the duty cycle (duty cycle is the percentage of a period, when a signal is active.), we can mimic an "average" analog voltage. As you can see in the following image, when we want medium voltage, we will keep the duty cycle as 50%. Similarly, if we want to achieve low voltage and high voltage, we will keep the duty cycle as 10% and 90% respectively. In this application, PWM is the process to control the power sent to the LED:



Using PWM on Arduino

Arduino UNO has 14 digital I/O pins. As mentioned in Module 1 *Chapter 1*, *Getting Started with Arduino*, we can use six pins (3, 5, 6, 9, 10, and 11) as PWM pins. These pins are controlled by on-chip timers, which toggle the pins automatically at a rate of about 490 Hz. As discussed earlier, we will be using the `analogWrite()` function.

In the following figure, the `analogWrite()` function takes the pin number and pin value as its parameter. Here, as you can see in the image, the pin value can be between 0 and 255, with the duty cycle mapped to 0% and 100%:



Connect the anode (longer head) of the LED to pin 11 (PWM pin) through a 220 Ω resistor and the cathode (shorter head) to ground, and write the following code in your editor window:

```
int led = 11;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int steps = 5;    // how many points to fade the LED by

void setup() {
 pinMode(led, OUTPUT);
}
```

```
void loop() {
  // Setting brightness of LED:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + steps;

  // When brightness value reaches either 0 or 255, reverse direction
of fading
  if (brightness == 0 || brightness == 255) {
    steps = -steps ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

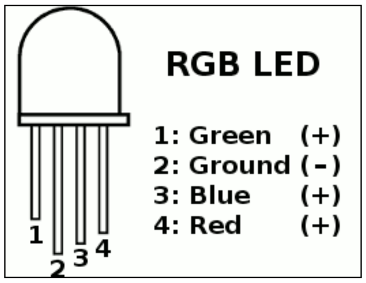
We are using pin 11(PWM pin) for fading the LED. We are storing the brightness of the LED in variable brightness. Initially, we are setting 0 brightness to the LED. When the `loop` function runs again, we are incrementing the value by steps of 5. As in the `analogWrite()` function, we can set the value between 0 and 255. Once brightness reaches maximum, we are decrementing the value. Similarly, once brightness reaches 0, we start incrementing brightness in steps of 1. To see the dimming effect, we are putting a delay of 30 milliseconds at the end of the code.

Creating a mood lamp

Lighting is one of the biggest opportunities for homeowners to effectively influence the ambience of their home, whether for comfort and convenience or to set the mood for guests. In this section, we will make a simple yet effective mood lamp using our own Arduino. We will be using an RGB LED for creating our mood lamp. An RGB (red, green, and blue) LED has all three LEDs in one single package, so we don't need to use three different LEDs for getting different colors. Also, by mixing the values, we can simulate many colors using some sophisticated programming. It is said that, we can produce 16.7 million different colors.

Using an RGB LED

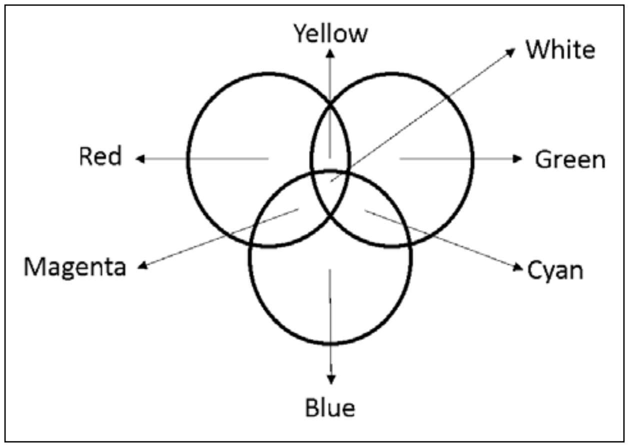
An RGB LED is simply three LEDs crammed into a single package. An RGB LED has four pins. Out of these four pins, one pin is the cathode (ground). As an RGB LED has all other three pins shorted with each other, it is also called a common anode RGB LED:



Here, the longer head is the cathode, which is connected with ground, and the other three pins are connected with the power supply. Be sure to use a current-limiting resistor to protect the LED from burning out. Here, we will mix colors as we mix paint on a palette or mix audio with a mixing board. But to get a different color, we will have to write a different analog voltage to the pins of the LED.

Why do RGB LEDs change color?

As your eye has three types of light interceptor (red, green, and blue), you can mix any color you like by varying the quantities of red, green, and blue light. Your eyes and brain process the amounts of red, green, and blue, and convert them into a color of the spectrum:



If we set the brightness of all our LEDs the same, the overall color of the light will be white. If we turn off the red LED, then only the green and blue LEDs will be on, which will make a cyan color. We can control the brightness of all three LEDs, making it possible to make any color. Also, the three different LEDs inside a single RGB LED might have different voltage and current levels; you can find out about them in a datasheet. For example, a red LED typically needs 2 V, while green and blue LEDs may drop up to 3-4 V.

Designing a mood lamp

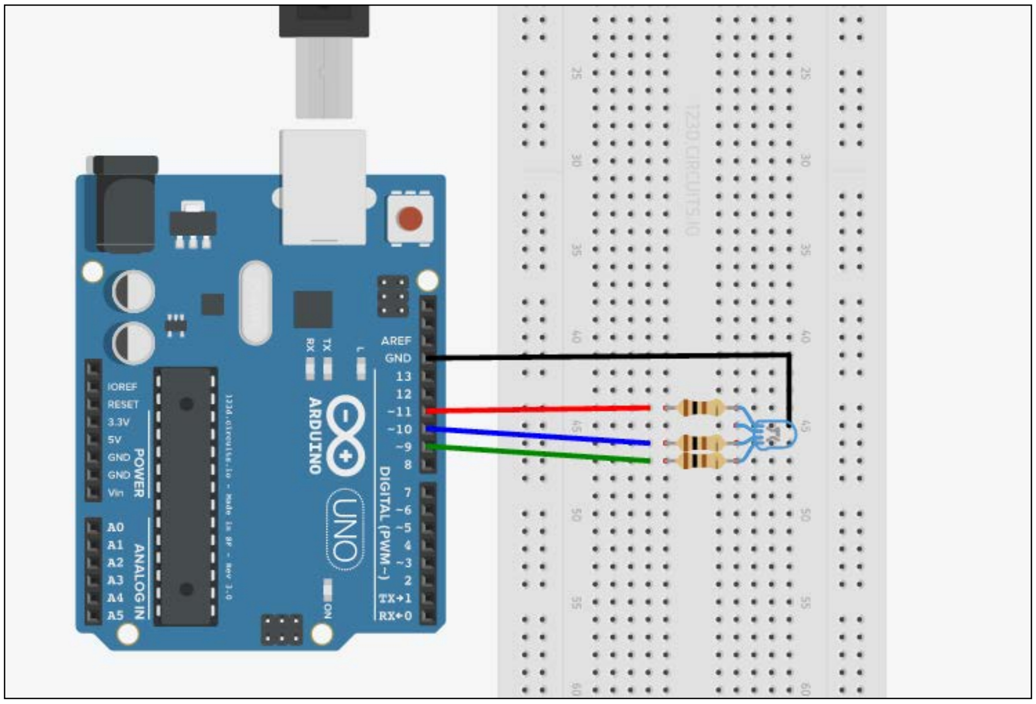
Now, we are all set to use our RGB LED in our mood lamp. We will start by designing the circuit for our mood lamp. In our mood lamp, we will make a smooth transition between multiple colors.

For that, we will need following components:

- An RGB LED
- 270 Ω resistors (for limiting the current supplied to the LED)
- A breadboard

As we did earlier, we need one pin to control one LED. Here, our RGB LED consists of three LEDs. So, we need three different control pins to control three LEDs. Similarly, three current-limiting resistors are required for each LED. Usually, this resistor's value can be between 100 Ω and 1000 Ω. If we use a resistor with a value higher than 1000 Ω, minimal current will flow through the circuit, resulting in negligible light emission from our LED. So, it is advisable to use a resistor having suitable resistance. Usually, a resistor of 220 Ω or 470 Ω is preferred as a current-limiting resistor.

As discussed in the earlier section, we want to control the voltage applied to each pin, so we will have to use PWM pins (3, 5, 6, 9, 10, and 11). The following schematic controls the red LED from pin 11, the blue LED from pin 10, and the green LED from pin 9. Hook the following circuit using resistors, the breadboard, and your RGB LED:



Once you have made the connection, write the following code in the editor window of Arduino IDE:

```
int redLed = 11;
int blueLed = 10;
int greenLed = 9;

void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(blueLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
}
void loop()
{
  setColor(255, 0, 0); // Red
  delay(500);
  setColor(255, 0, 255); // Magenta
  delay(500);
  setColor(0, 0, 255); // Blue
  delay(500);
  setColor(0, 255, 255); // Cyan
  delay(500);
  setColor(0, 255, 0); // Green
  delay(500);
  setColor(255, 255, 0); // Yellow
  delay(500);
  setColor(255, 255, 255); // White
  delay(500);
}
void setColor(int red, int green, int blue)
{
  // For common anode LED, we need to subtract value from 255.
  red = 255 - red;
  green = 255 - green;
  blue = 255 - blue;
  analogWrite(redLed, red);
  analogWrite(greenLed, green);
  analogWrite(blueLed, blue);
}
```

We are using very simple code for changing the color of the LED at every one second interval. Here, we are setting the color every second. So, this code won't give you a smooth transition between colors. But with this code, you will be able to run the RGB LED. Now we will modify this code to smoothly transition between colors. For a smooth transition between colors, we will use the following code:

```
int redLed = 11;
int greenLed = 10;
int blueLed = 9;

int redValue   = 0;
int greenValue = 0;
int blueValue  = 0;

void setup(){
  randomSeed(analogRead(0));
}

void loop() {
  redValue = random(0,256); // Randomly generate 1 to 255
  greenValue = random(0,256); // Randomly generate 1 to 255
  blueValue = random(0,256); // Randomly generate 1 to 255
```

```
    analogWrite(redLed,redValue);
    analogWrite(greenLed,greenValue);
    analogWrite(blueLed,blueValue);

  // Incrementing all the values one by one after setting the random 
  values.
    for(redValue = 0; redValue < 255; redValue++){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
    for(greenValue = 0; greenValue < 255; greenValue++){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
    for(blueValue = 0; blueValue < 255; blueValue++){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }

    //Decrementing all the values one by one for turning off all the 
  LEDs.
    for(redValue = 255; redValue > 0; redValue--){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
    for(greenValue = 255; greenValue > 0; greenValue--){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
    for(blueValue = 255; blueValue > 0; blueValue--){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
  }
```

We want our mood lamp to repeat the same sequence of colors again and again. So, we are using the `randomSeed()` function. The `randomSeed()` function initializes the pseudo random number generator, which will start at an arbitrary point and will repeat in the same sequence again and again. This sequence is very long and random, but will always be the same. Here, pin 0 is unconnected. So, when we start our sequence using `analogRead(0)`, it will give some random number, which is useful in initializing the random number generator with a pretty fair random number. The `random(min,max)` function generates the random number between min and max values provided as parameters. In the `analogWrite()` function, the number should be between 0 and 255. So, we are setting min and max as 0 and 255 respectively. We are setting the random value to redPulse, greenPulse, and bluePulse, which we are setting to the pins. Once a random number is generated, we increment or decrement the value generated with a step of 1, which will smooth the transition between colors.

Now we are all set to use this as mood lamp in our home. But before that we need to design the outer body of our lamp. We can use white paper (folded in a cube shape) to put around our RGB LED. White paper acts as a diffuser, which will make sure that the light is mixed together. Alternatively, you can use anything which diffuses light and make things looks beautiful! If you want to make the smaller version of the mood lamp, make a hole in a ping pong ball. Extend the RGB LED with jump wires and put that LED in the ball and you are ready to make your home look beautiful.

Developing an LED night lamp

So now we have developed our mood lamp, but it will turn on only when we connect a power supply to Arduino. It won't turn on or off depending on the darkness of the environment. Also, to turn it off, we have to disconnect our power supply from Arduino. In this section, we will learn how to use switches with Arduino.

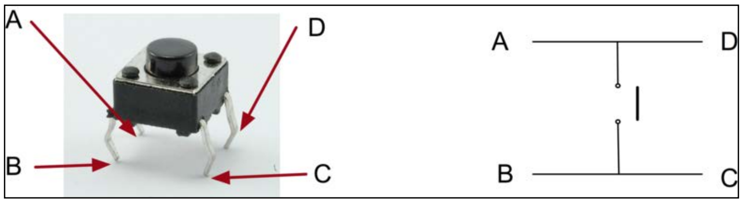
Introduction to switch

Switches are one of the most elementary and easy-to-overlook components. Switches do only one thing: either they open a circuit or short circuit. Mainly, there are two types of switches:

- **Momentary switches**: Momentary switches are those switches which require continuous actuation—like a keyboard switch and reset button on the Arduino board
- **Maintained switches**: Maintained switches are those switches which, once actuated, remain actuated—like a wall switch.

Normally, all the switches are NO (normally opened) type switches. So, when the switch is actuated, it closes the path and acts as a perfect piece of conducting wire. Apart from this, based on their working, many switches are out there in the world, such as toggle, rotary, DIP, rocker, membrane, and so on.

Here, we will use a normal push button switch with four pins:

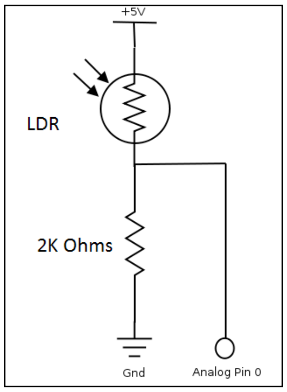


In our push button switch, contacts A-D and B-C are short. We will connect our circuit between A and C. So, whenever you press the switch, the circuit will be complete and current will flow through the circuit. We will read the input from the button using the `digitalRead()` function. We will connect one pin (pin A) to the 5 V, and the other pin (pin C) to Arduino's digital input pin (pin 2). So whenever the key is pressed, it will send a 5 V signal to pin 2.

Pixar lamp

We will add a few more things in the mood lamp we discussed to make it more robust and easy to use. Along with the switch, we will add some kind of light-sensing circuit to make it automatic. We will use a **Light Dependent Resistor** (**LDR**) for sensing the light and controlling the lamp.

Basically, LDR is a resistor whose resistance changes as the light intensity changes. Mostly, the resistance of LDRs drops as light increases. For getting the value changes as per the light levels, we need to connect our LDR as per the following circuit:



Here, we are using a voltage divider circuit for measuring the light intensity change. As light intensity changes, the resistance of the LDR changes, which in turn changes the voltage across the resistor. We can read the voltage from any analog pin using `analogRead()`.

Once you have connected the circuit as shown, write the following code in the editor:

```
int LDR = 0; //will be getting input from pin A0
int LDRValue = 0;
int light_sensitivity = 400;    //This is the approx value of light
surrounding your LDR
int LED = 13;

void setup()
{
  Serial.begin(9600);          //start the serial monitor with 9600
buad
  pinMode(LED, OUTPUT);

}

void loop()
{
  LDRValue = analogRead(LDR);     //Read the LDR's value through LDR
pin A0
  Serial.println(LDRValue);       //Print the LDR values to serial
monitor

  if (LDRValue < light_sensitivity)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
  }
  delay(50);        //Delay before LDR value is read again
}
```

In the preceding code, we are reading the value from our LDR at pin analog A0. Whenever the value read from pin A0 is below a certain threshold value, we are turning on the LED. So whenever the light (lux value) around the LDR drops, then the set value, it will turn on the LED, or in our case, the mood lamp.

Similarly, we will add a switch in our mood lamp to make it fully functional as a pixar lamp.

Connect one pin of the push button at 5 V and the other pin to digital pin 2. We will turn on the lamp only, and only when the room is dark and the switch is on. So we will make the following changes in the previous code.

In the `setup` function, initialize pin 2 as input, and in the loop add the following code:

```
buttonState = digitalRead(pushSwitch); //pushSwitch is initialized as
2.
If (buttonState == HIGH){

//Turn on the lamp
}
Else {
//Turn off the lamp.
//Turn off all LEDs one by one for smoothly turning off the lamp.
  for(redValue = 255; redValue > 0; redValue--){
    analogWrite(redLed,redValue);
    analogWrite(greenLed,greenValue);
    analogWrite(blueLed,blueValue);
    delay(10);
  }
  for(greenValue = 255; greenValue > 0; greenValue--){
    analogWrite(redLed,redValue);
    analogWrite(greenLed,greenValue);
    analogWrite(blueLed,blueValue);
    delay(10);
  }
  for(blueValue = 255; blueValue > 0; blueValue--){
    analogWrite(redLed,redValue);
    analogWrite(greenLed,greenValue);
    analogWrite(blueLed,blueValue);
    delay(10);
  }
}
```

So, now we have incorporated an LDR and switch in our lamp to use it as a normal lamp.

Summary

In this chapter, we started with using a breadboard. Programming multiple LEDs was explained by developing a model traffic light controller. A mood lamp with RGB LED and a Pixar lamp with a sensing part were developed by the end of this chapter.

In the next chapter, a remote-controlled TV backlight will be developed, where you will learn how to make communication between Arduino and a TV remote.

2
Project 2 – Remote Controlled TV Backlight

In the previous chapter, *Project 1 – LED Night Lamp*, we dived into the amazing world of LEDs. We created some cool projects using different types of LEDs. Now, we will use another type of LED – an IR (Infrared) LED. In this chapter, we will start by learning the basics of IR LEDs and the basics of IR communication. Once you have learned about programming the IR sensor, we will use it to control a TV backlight using a remote. In this chapter, you will learn about the following:

- Introduction to and the workings of an IR LED
- Programming an IR sensor
- How to control an LED array
- Developing a remote controlled TV backlight

Introduction to IR LEDs

In the world of wireless technology, IR (infrared) is one of the most common, inexpensive, and easy to use modes of communication. You might have always wondered how a TV remote works. A TV remote uses IR LEDs to send out the signal. As the wavelength of light emitted from the IR LED is longer than the visible light, one can't see it with the naked eye. But, if you look through the camera of your mobile or any other camera, you can see the light beaming when you press any key on the remote. Let's first understand what an IR LED is and what the different applications of an IR LED are.

What is IR LED?

An IR (infrared) LED, also known as an IR (infrared) transmitter, transmits infrared waves in the range of 875 nm to 950 nm. Usually, IR LEDs are made up of gallium arsenide or aluminum gallium arsenide. The working principle of an IR LED is the same as we mentioned in the previous chapters. The longer lead of the LED is the anode and the shorter one is the cathode, as shown here:

Applications of IR LED / IR communication

Apart from using IR communication in TV remote controls, there are a number of other applications that use IR communication. Infrared light can also be used to transfer data between electronic devices. Although infrared light is invisible to the eye, digital cameras and other sensors can see this light, so infrared light can be used for many security systems and night vision technology. Apart from technical uses, the U.S. Food and Drug Administration Department has approved several products with IR LEDs for use in some medical and cosmetic procedures.

IR sensors

We have learned the basics of IR communication and IR LEDs, so now we will move on to making our own IR sensor.

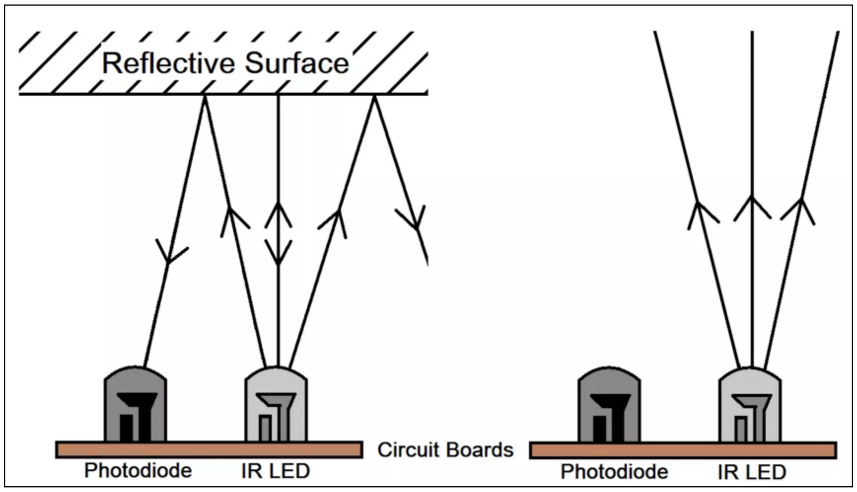
There are various types of IR sensors, depending upon the application. Proximity sensors, obstruction sensors, and contrast sensors (used in line follower robot) are a few examples which use IR sensors.

Working mechanism

An IR sensor consists of an IR LED (which works as a transmitter) and a photodiode (which works as a receiver) pair. The IR LED emits IR radiation, which, on receiving at the photodiode dictates the output of the sensor.

There are different types of use cases for an IR sensor. For example, if we held an IR LED directly in front of the photodiode, such that almost all the radiation reaches the photodiode, this can be used as burglar alarm. So, when anyone interrupts the line of sight between the IR LED and the photodiode, this will break the continuous radiation coming from the IR LED, and we can program it to raise an alarm. This type of mechanism is also called a direct incidence, as we are not using any reflective surface in between the transmitter and receiver.

Another use case is with indirect incidence, in which we use physics' law of absorption. This means, when light is directed at a black surface, the black surface will actually absorb the light. Similarly, when IR radiation is directed at a black surface, the surface will absorb the IR radiation. But, when it is directed toward a white surface, the surface will reflect the IR radiation. Based on the amount of light received back from the surface, we can detect, if robot is following a line or not. So, the absorption principle can be used for line follower robot:



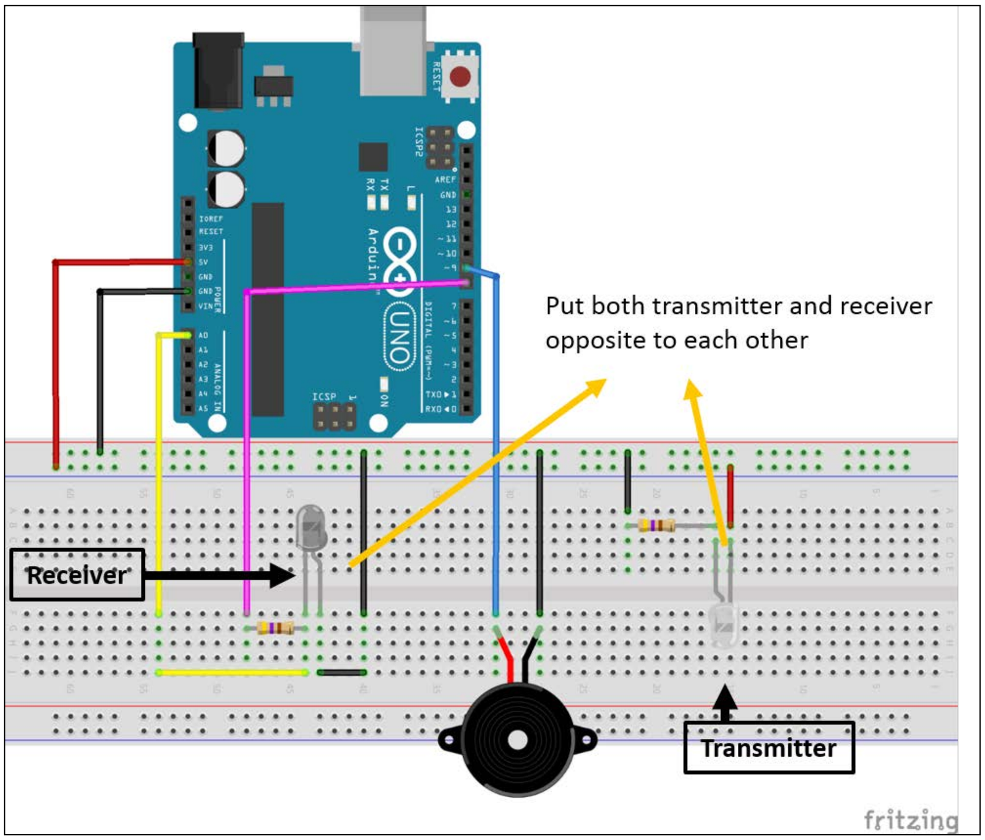
As you can see in the preceding image, whenever any obstacle is detected in the path of the IR, some of the IR radiation is reflected back, which, on receipt of the IR waves, gives the output. This type of setup is also useful for detecting any object/obstacle in the path.

Programming a basic IR sensor

After understanding the basic workings of a simple IR sensor, we will learn how to program this sensor.

Based on the principle of direct incidence, we will develop a simple burglar alarm.

Once you have connected the circuit as shown in the following diagram, upload the following code to Arduino:



```
int IRTransmitter = 8;
int IRReceiver = A0; //IR Receiver is connected to pin 5

int buzzer = 9; // Buzzer is connected with pin 9.
int output = 0; // Variable to store the value from the IR sensor
int ambientLight = 500;

void setup()
{
  pinMode(IRReceiver,INPUT);
  pinMode(IRTransmitter,OUTPUT);
  pinMode(buzzer,OUTPUT);
  digitalWrite(IRTransmitter,HIGH);
}
```

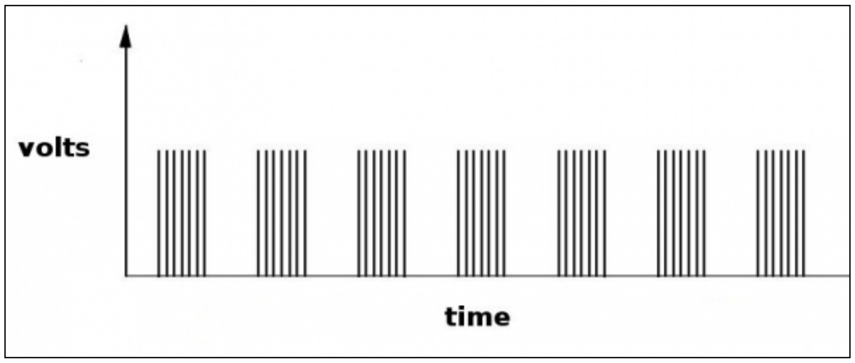
```
void loop()
{
  output = analogRead(IRReceiver);
  // If anything comes in between the IR Transmitter and IR receiver
  // IR receiver will not give the output. Make an alarm.
  if (output < ambientLight)
  {
    makeAlarm(50);
  }
}
void makeAlarm(unsigned char time)
{
  analogWrite(buzzer,170);
  delay(time);
  analogWrite(buzzer,0);
  delay(time);
}
```

In the preceding code, we are continuously making the IR transmitter ON. So, the IR receiver/photodiode will continuously receive an IR signal. Whenever any person tries to break into the house or safe, the IR signal will get interrupted, which in turn will lower the voltage drop across the IR receiver and an alarm will be raised. Here, based on the conditions around you, you will have to change the value of the `ambientLight` variable in the code.

How to receive data from a TV remote

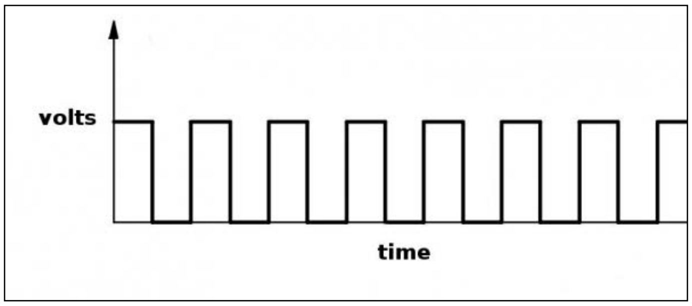
As we discussed earlier, a TV remote has an IR LED, which transmits IR light with a 38 kHz frequency. So, whenever that radiation is detected at the receiver part of the TV (the control circuit of the TV), we need to filter that signal.

As you can see in the following image, when a signal is actually transmitted from the TV remote, it will look like a series of waves:



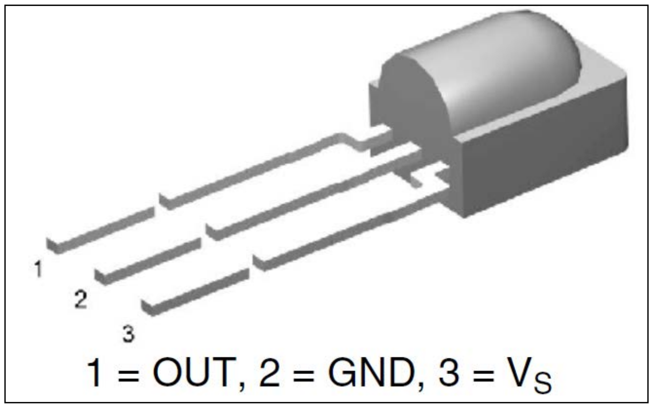
For filtering the actual signal from a TV remote from ambient noise, we will use TSOP38238 IC. TSOP38238 looks like a transistor, but actually this device combines an IR sensitive photocell, a 38 kHz band pass filter, and an automatic gain controller. Here, the IR sensitive photocell works as an IR receiver (photodiode), and the 38 kHz band pass filter is required to smooth the received modulated signal.

After passing through the 38 kHz band pass filter, the output signal will look as shown in the following, which is much cleaner and easier to read:



Also, TSOP38238 is covered in the lead package epoxy frame, which acts as an IR filter. So, the disturbance light (DC light from a tungsten bulb, or a modulated signal from fluorescent lamps) does not reach the photodiode. This demodulated signal can be directly decoded by any microprocessor (in our case, Arduino). Because of this, apart from preamplifier, it ignores all other IR light unless it is modulated at a specific frequency (38 kHz).

This IC is as simple as an IC can get with its three pins. There are two power pins and one pin for the output signal:



TSOP38238 can be powered with a supply from 2.5 V to 5.5 V, which makes it suitable for all types of application. Also, it can receive signals from almost all types of remotes.

To start with, we will control a single LED using an IR remote. For that, we will be using the `IRRemote` library, available at `https://github.com/z3t0/Arduino-IRremote`. Once you have downloaded and extracted the code in the libraries folder of Arduino, you will be able to see the examples in the Arduino IDE. We need to record the value of all the buttons for future use.

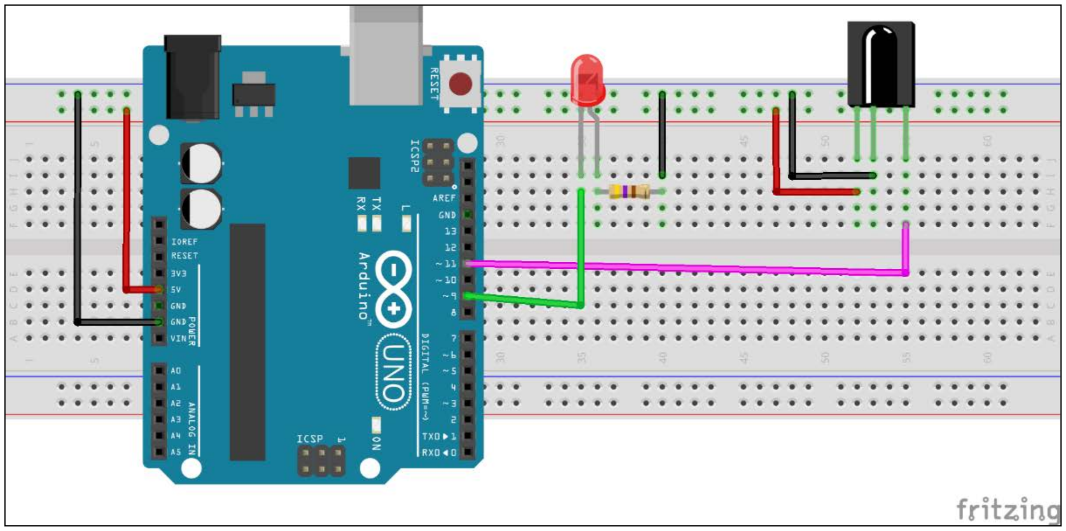
Connect the first pin of TSOP38238 to pin 11 of Arduino, the second pin to the ground pin of Arduino, and the third pin to the 5 V pin of Arduino. Connect Arduino over USB and try to compile the following code:

```
#include <IRremote.h>
const int IR_RECEIVER = 11;
IRrecv receiver(IR_RECEIVER);
decode_results buttonPressed;
void setup()
{
  Serial.begin(9600);
  receiver.enableIRIn(); // Start the receiver
}

void loop()
{
  if (receiver.decode(&buttonPressed))
  {
    Serial.println(buttonPressed.value); //Printing values coming from
the IR Remote
    receiver.resume(); // Receive the next value
  }
  delay(100);
}
```

In the latest version of Arduino, the `IRremote.h` library is already installed in the `RobotIRremote` folder. But, you won't have all the examples available from the downloaded library. So, delete the `RobotIRremote` library and try to compile the code again.

Once you have deleted the duplicate file, you should be able to successfully compile the code. After uploading the preceding code, open the serial monitor. As we want to control a single LED using a remote, we will have to know the value of the button pressed. So, try noting down the values of all the buttons one after the other.

For example, digit 1 has 54,528 codes for a certain remote. You might have to check for the remote that you have. We will now control the LED by using the IR remote. Along with the IR receiver, we will connect one LED, as shown in the following circuit:



Update the code for controlling the LED, based on your readings from the previous exercise:

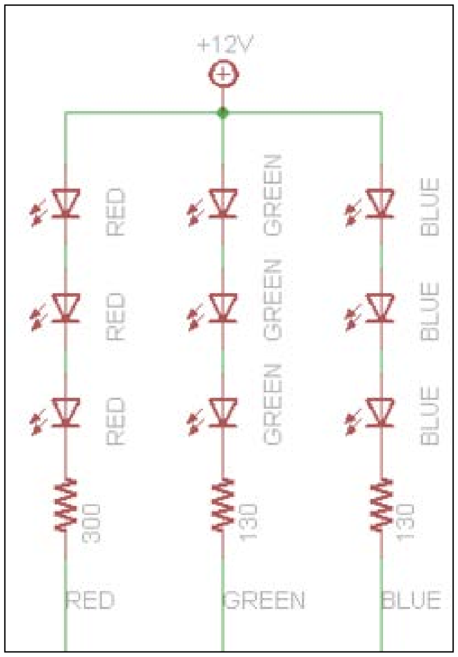
```
#include <IRremote.h>
const int IR_RECEIVER = 11;
IRrecv receiver(IR_RECEIVER);
decode_results buttonPressed;
long int buttonValue = 0;
const long int buttonOne = 54528; //Update value one according to your
readings
const long int buttonTwo = 54572; //Update value two according to your
readings


int LEDpin = 9;

void setup()
{
  Serial.begin(9600);
  receiver.enableIRIn(); // Start the receiver
  pinMode(LEDpin,OUTPUT);
}

void loop()
{
  if (receiver.decode(&buttonPressed))
  {
    buttonValue = buttonPressed.value;
    Serial.println(buttonValue);
    switch (buttonValue){
      case buttonOne:
           digitalWrite(LEDpin,HIGH);
           break;
      case buttonTwo:
           digitalWrite(LEDpin,LOW);
           break;
      default:
           Serial.println("Waiting for input. ");
    }
    receiver.resume(); // Receive the next value
  }
  delay(100);
}
```

Now you will be able to control a single LED using the IR remote. In a similar way, you can control anything that you want to control using an IR remote. At the end of this chapter, we want to control a TV backlight using an IR remote. We will develop a TV backlight using an LED strip. So, we will now learn about how to control an LED array.

LED strips

LED strips are flexible circuit boards with full color LEDs soldered on them. There are two basic types of LED strip: the "analog" and "digital" kinds. Analog strips have all the LEDs connected in parallel. So, they act as one huge tri-color LED. In case of an analog LED strip, we can't set the color/brightness of each LED. So, they are easy to use and inexpensive. Digital LED strips work in a different way. To use the LED, we have to send digital code corresponding to each LED in the case of a digital LED strip. As they provide more modularity, they are quite expensive. Also, digital LED strips are difficult to use compared to analog LED strips:

Internally, RGB LEDs are connected to each other in parallel. One section of the strip contains all three LEDs, connected in parallel. A complete LED strip is made up of a number of parallel RGB LEDs connected in series. Connection in one block/section is as shown in the following figure:



As one section contains multiple LEDs, it requires more current to run. It can't run on the current provided from Arduino. In case of full white, each segment will require approximately 60 mA of current per block/section. So to run the complete LED strip, we may require current of up to 1.2 A per meter. For running an LED strip, an external power supply of 12 V is required.

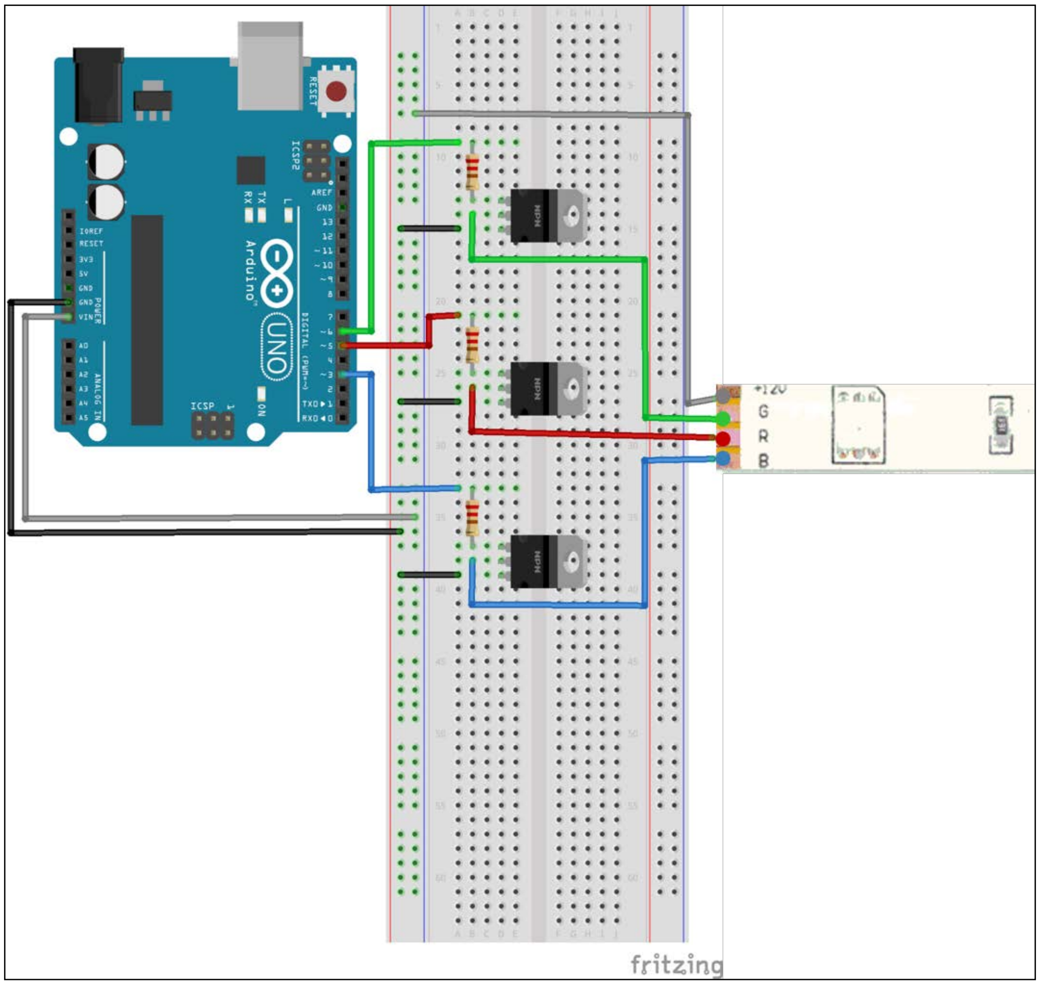
Controlling an LED strip with Arduino

As we know, we can draw maximum current up to 300 mA if we put all the I/O pins together. But, as we require a higher current, we will have to use an external power supply. Now here comes the tricky part. We will use Arduino to control the LED strip, and for a power supply, we will use external power. For connecting the LED strip with Arduino, we will use MOSFET to limit the current drawn to and from Arduino.

Here, we will use N-channel MOSFET, which is very inexpensive and works with 3.3 V to 5 V logic. These FETs can switch over 60 A and 30 V.

Connecting to the strip is relatively easy. We need to solder the wires to the four pins/copper pads of our LED strip. One can use heat shrink for providing insulation, abrasion resistance, and environmental protection.

To use our LED strip with Arduino, we will require some resistors as well, to limit the current:



Connect the power transistor with the resistor in between the PWM output pin and the base. In case of NPN transistors, pin 1 is the base. Pin 2 is the collector and pin 3 is the emitter. Now, for power supply to Arduino and the LED strip, connect a 9 -12 V power supply to the Arduino, so that Vin supplies the high voltage to the LED.

At the end of the wiring connection, make sure to connect the ground of the supply to that of Arduino/MOSFETs. Connect the Arduino output pin to the base of the MOSFET (pin 1) with a 100-220 Ω resistor in between. Connect pin 2 of the MOSFET to the LED strip's input pin and connect pin 3 of the MOSFET to the ground.

Check all the connections and write the following code in the editor window of Arduino. Here, we are using pins 5, 6, and 3 of Arduino to control the LED strip's red, green, and blue LEDs respectively:

```
int redLed = 5;
int greenLed = 6;
int blueLed = 3;

int redValue   = 0;
int greenValue = 0;
int blueValue  = 0;

void setup(){
  randomSeed(analogRead(0));
}

void loop() {
  redValue = random(0,256); // Randomly generate 1 to 255
  greenValue = random(0,256); // Randomly generate 1 to 255
  blueValue = random(0,256); // Randomly generate 1 to 255

  analogWrite(redLed,redValue);
  analogWrite(greenLed,greenValue);
  analogWrite(blueLed,blueValue);

// Incrementing all the values one by one after setting the random 
values.
  for(redValue = 0; redValue < 255; redValue++){
    analogWrite(redLed,redValue);
    analogWrite(greenLed,greenValue);
    analogWrite(blueLed,blueValue);
    delay(10);
  }
  for(greenValue = 0; greenValue < 255; greenValue++){
    analogWrite(redLed,redValue);
    analogWrite(greenLed,greenValue);
    analogWrite(blueLed,blueValue);
```

```
      delay(10);
    }
    for(blueValue = 0; blueValue < 255; blueValue++){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }

    //Decrementing all the values one by one for turning off all the
  LEDs.
    for(redValue = 255; redValue > 0; redValue--){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
    for(greenValue = 255; greenValue > 0; greenValue--){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
    for(blueValue = 255; blueValue > 0; blueValue--){
      analogWrite(redLed,redValue);
      analogWrite(greenLed,greenValue);
      analogWrite(blueLed,blueValue);
      delay(10);
    }
  }
```

If everything is in place, you should be able to see the LED strip getting on and changing its color.

Now we have learned about all the things required to control the TV backlight using IR remote, so we will integrate all the things that we learned in this chapter:

- How IR works
- How to read values from an IR remote
- How to control an LED strip

We want to control the brightness of the LED strip as well. We will use the power button to turn the backlight off and on. With volume plus and volume minus, we will increase and decrease the brightness of the backlight.

As we did earlier in the chapter, connect TSOP38238 to pin 11, 5 V, and ground pin of Arduino. Once you have done all the connections, upload the following code on Arduino:

```
#include <IRremote.h>
const int IR_RECEIVER = 11; // Connect output pin of TSOP38238 to pin
11
IRrecv receiver(IR_RECEIVER);
decode_results buttonPressed;
long int buttonValue = 0;

// Mention the codes, you get from previous exercise
const long int POWER_BUTTON = 54572; // Power button to turn on or off
the backlight
const long int PLUS_BUTTON = 54536;  // Increase brightness of the LED
Strip
const long int MINUS_BUTTON = 54608; // Decrease brightness of the LED
strip
const long int CHANGE_COLOR = 54584; // Decrease brightness of the LED
strip

const int FADE_AMOUNT = 5; // For fast increasing/decreasing
brightness increase this value
boolean isOn = false;

int redLed = 5;
int greenLed = 6;
int blueLed = 3;

int redValue   = 0;
int greenValue = 0;
int blueValue  = 0;

int colors[3];

// Power up the LED strip with random color
void powerUp(int *colors)
{
  redValue = random(0, 256); // Randomly generate 1 to 255
  greenValue = random(0, 256); // Randomly generate 1 to 255
  blueValue = random(0, 256); // Randomly generate 1 to 255

  analogWrite(redLed, redValue);
  analogWrite(greenLed, greenValue);
```

```
    analogWrite(blueLed, blueValue);

    colors[0] = redValue;
    colors[1] = greenValue;
    colors[2] = blueValue;
  }

  // Turn off the LED
  void powerDown(int *colors)
  {
    redValue = colors[0];
    greenValue = colors[1];
    blueValue = colors[2];

    //Decrementing all the values one by one for turning off all the
  LEDs.
    for (; redValue > 0; redValue--) {
      analogWrite(redLed, redValue);
      delay(10);
    }
    for (; greenValue > 0; greenValue--) {
      analogWrite(greenLed, greenValue);
      delay(10);
    }
    for (; blueValue > 0; blueValue--) {
      analogWrite(blueLed, blueValue);
      delay(10);
    }
    colors[0] = redValue;
    colors[1] = greenValue;
    colors[2] = blueValue;
  }

  void increaseBrightness(int *colors)
  {
    redValue = colors[0];
    greenValue = colors[1];
    blueValue = colors[2];

    redValue += FADE_AMOUNT;
    greenValue += FADE_AMOUNT;
    blueValue += FADE_AMOUNT;

    if (redValue >= 255) {
```

```
    redValue = 255;
  }

  if (greenValue >= 255) {
    greenValue = 255;
  }

  if (blueValue >= 255) {
    blueValue = 255;
  }
  analogWrite(redLed, redValue);
  analogWrite(greenLed, greenValue);
  analogWrite(blueLed, blueValue);

  colors[0] = redValue;
  colors[1] = greenValue;
  colors[2] = blueValue;
}

void decreaseBrightness(int *colors)
{
  redValue = colors[0];
  greenValue = colors[1];

  blueValue = colors[2];

  redValue -= FADE_AMOUNT;
  greenValue -= FADE_AMOUNT;
  blueValue -= FADE_AMOUNT;

  if (redValue <= 5) {
    redValue = 0;
  }

  if (greenValue <= 5) {
    greenValue = 0;
  }

  if (blueValue <= 5) {
    blueValue = 0;
  }
  analogWrite(redLed, redValue);
  analogWrite(greenLed, greenValue);
```

```
    analogWrite(blueLed, blueValue);

    colors[0] = redValue;
    colors[1] = greenValue;
    colors[2] = blueValue;
  }

  // Randomly generates a color and make a smooth transition to that
  color
  void changeColor(int *colors)
  {
    int newRedValue = random(0, 256); // Randomly generate 1 to 255
    int newGreenValue = random(0, 256); // Randomly generate 1 to 255
    int newBlueValue = random(0, 256); // Randomly generate 1 to 255

    redValue = colors[0];
    greenValue = colors[1];
    blueValue = colors[2];

    if (newRedValue > redValue) {
      for (; redValue >= newRedValue; redValue++) {
        analogWrite(redLed, redValue);
        delay(10);
      }
    }
    else {
      for (; redValue <= newRedValue; redValue--) {
        analogWrite(redLed, redValue);
        delay(10);
      }
    }

    if (newGreenValue > greenValue) {
      for (; greenValue >= newGreenValue; greenValue++) {
        analogWrite(greenLed, greenValue);
        delay(10);
      }
    }
    else {
      for (; greenValue <= newGreenValue; greenValue--) {
        analogWrite(greenLed, greenValue);
        delay(10);
      }
    }
    if (newBlueValue > blueValue) {
      for (; blueValue >= newBlueValue; blueValue++) {
        analogWrite(blueLed, blueValue);
        delay(10);
      }
    }
    else {
      for (; blueValue <= newBlueValue; blueValue--) {
        analogWrite(blueLed, blueValue);
        delay(10);
      }
    }

    colors[0] = redValue;
    colors[1] = greenValue;
    colors[2] = blueValue;
  }

  void setup() {
    Serial.begin(9600);
    receiver.enableIRIn(); // Start the receiver

    randomSeed(analogRead(0));

    pinMode(redLed, OUTPUT);
    pinMode(greenLed, OUTPUT);
    pinMode(blueLed, OUTPUT);
  }

  void loop() {

    if (receiver.decode(&buttonPressed))
    {
      buttonValue = buttonPressed.value;
      Serial.println(buttonValue);
      switch (buttonValue) {
        case POWER_BUTTON:
          if (!isOn) {
            powerUp(colors);
            isOn = true;
          }
          else {
            powerDown(colors);
            isOn = false;
          }
          break;
        case PLUS_BUTTON:
          decreaseBrightness(colors);
          break;
        case MINUS_BUTTON:
          increaseBrightness(colors);
          break;
        case CHANGE_COLOR:
          changeColor(colors);
          break;
        default:
          Serial.println("Waiting for input. ");
      }
      receiver.resume(); // Receive the next value
    }
    delay(100);
  }
```

In the preceding code, we are using the power button to turn the backlight on and off, and the volume up and down buttons to increase and decrease the brightness respectively. I have used the up arrow button to change the color of the strip. You can add more features to this project by configuring it in the switch case block.

Summary

In this chapter, we started with the basics of IR LEDs and IR communication. After that, we learnt about programming IR sensors and their applications. By the end of this chapter, we learnt about controlling an LED strip, and we completed our chapter by developing a remote controlled TV backlight.

In the coming chapters, we will start into the more advanced stuff of developing an LED cube, sound visualization, and persistence of vision.

3
Project 3 – LED Cube

If you have successfully implemented the last two projects, you will have noticed that there is very little or no soldering involved. However, I would say you haven't worked on electronics if you haven't done some intense soldering and burnt your hands. In this chapter, you will get introduced to soldering in detail. You will also understand how to create a 4*4*4 LED cube using an Arduino UNO board. You will learn about the following:

- Introduction to soldering
- Designing an LED cube
- Programming a 4*4*4 LED cube

Getting started with soldering

Soldering is the process of making a sound electrical and mechanical joint between certain metals by joining them with a soft solder. This is an alloy of lead and tin with a low melting point. The joint is heated to the correct temperature by a soldering iron. Effective soldering requires good heat transfer from the iron to the components to be soldered. The longer heat is applied, the greater the risk of heat damage to the wire or component, so it's important to get the job wrapped up quickly.

What you will need

Before you proceed further with the next section of the chapter, make sure you have the following items with you and have got yourself familiarized with the tools:

- Soldering iron
- Basic stand
- Solder desoldering pump
- Cardboard

The following image is for your quick reference so that you will get a rough idea of soldering tools. One thing to remember is that soldering tools vary from place to place, so don't worry if your tools don't look exactly the same. An important thing to note at the beginner level is that your soldering skills are highly dependent on the kind of soldering tools you are using, so make sure you buy the best soldering tools available in your country:

Go to any electronics components store and ask for a soldering kit. They will give you all the necessary components needed for soldering. Alternatively, you can order electronics components from a few online stores as well.

Safety tips

Soldering poses a few different dangers, so, to stay as safe as possible, always follow these soldering safety tips:

- Never touch the element or tip of the soldering iron. They are very hot (about 400 degree Celsius) and will give you a nasty burn.
- Take great care to avoid touching the mains flex with the tip of the iron.

- Always return the soldering iron to its stand when not in use.
- Never put the soldering iron down on your workbench, even for a moment!
- Work in a well-ventilated area.
- The smoke formed as you melt solder is mostly from the flux and quite irritating. Avoid breathing it in by keeping your head to the side of, not above, your work.
- Wash your hands after using solder.
- Solder contains lead which is a poisonous metal.
- Never solder a live circuit (one that is energized).

Designing an LED cube

As mentioned before, the main focus of this chapter is on soldering. In this section, you will learn how to design an LED cube, which will have intense soldering, and creative elements like LED control and Arduino programming.

Required components

Before getting into the cube design, make sure you have following components for this project:

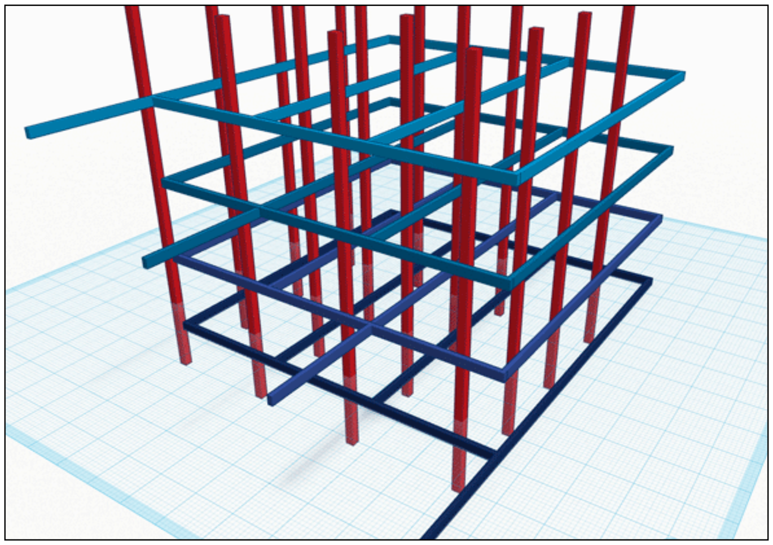
- Arduino UNO
- 64 LEDs: You can use any color LED. Although 64 LEDs are required for this project, I would recommend you to buy at least 100 LEDs in case some LEDs get burned during the soldering process.
- 16 resistors: These must be appropriate to your LEDs. If you are not sure which resistor to purchase, get 500 Ω/1k Ω resistors.
- Connecting wires
- A printed circuit board
- Thermocol
- Soldering iron and solder wire

Principle behind the design

Before you read this section, make sure you have the listed components or have already ordered them. This section is the most important part of this chapter as it explains the key principle behind the project. It gives a complete overview of the system so that it always stays in the back of your mind and can help identify stuff as you go along. On the Internet, you will find lots of tutorials on making an LED cube, however, most of them would use a single output pin for every single LED. If you use that approach for a 4*4*4 LED cube, you would need 64 pins, which Arduino UNO doesn't have. One approach would be to use shift registers, which will make it complicated for beginners and first-timers.

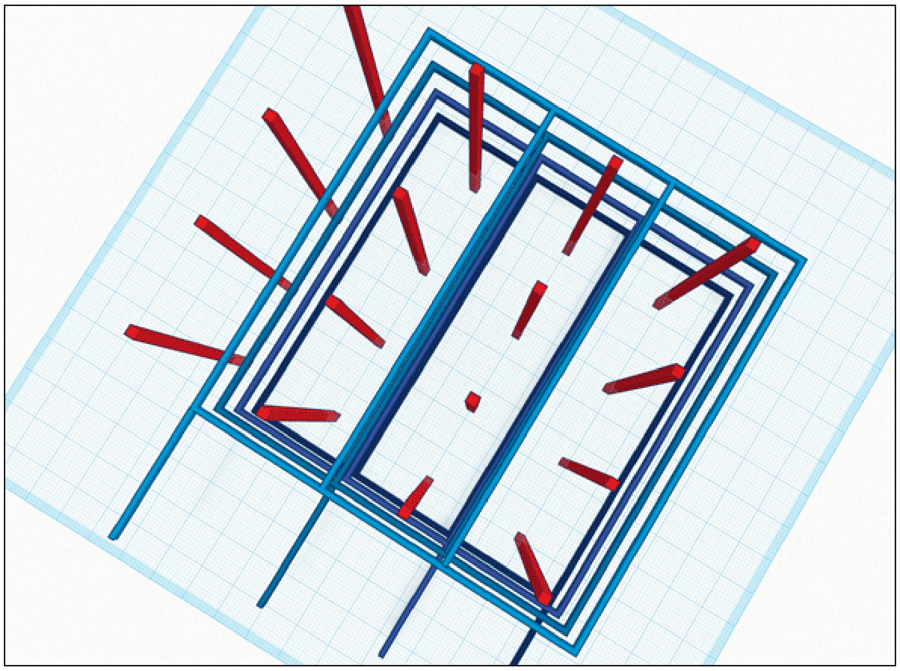
If you look at the Arduino UNO board, you will notice that at max, 20 pins can be used. So to control all those LEDs in 20 pins, a multiplexing technique will be used. If you break down a cube into four layers, you will need 16 control pins for referencing individual LEDs. To enable a particular LED of the cube, all the layers needs to be activated. So in total, you need 16+4 (20) pins for this project.

There will be a common cathode (negative terminal) for each layer, so all the negative legs of the LEDs are connected to a single pin for that layer. For the anode (positive terminal), each LED will be connected to the corresponding layer above and below. So in total, you will have 16 columns of positive terminals and four layers of negative terminals. Following are some 3D views of the design, which will help you better understand it. Red ones are positive terminals and blue ones are negative terminals:



Source: http://cdn.makeuseof.com/wp-content/uploads/2012/07/structural-diagram1.png

Here is another image for your reference:

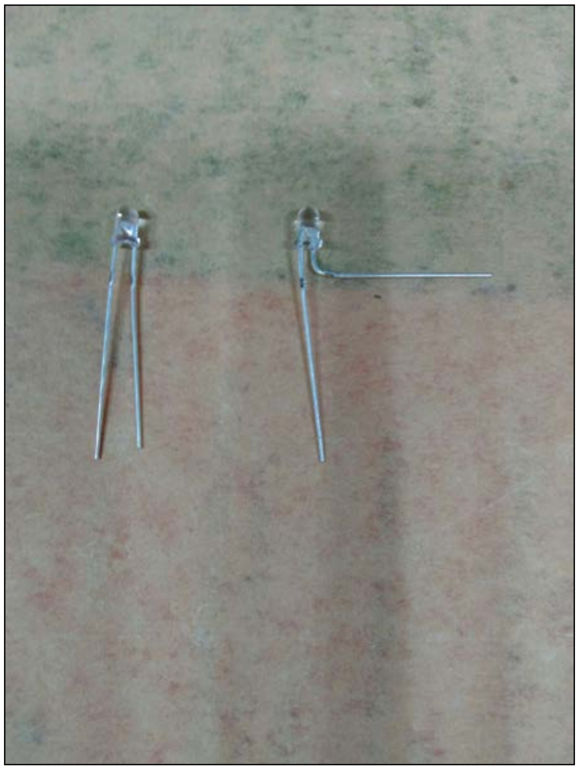


Source: http://cdn.makeuseof.com/wp-content/uploads/2012/07/cube-wiring-layers-from-top.png

Construction

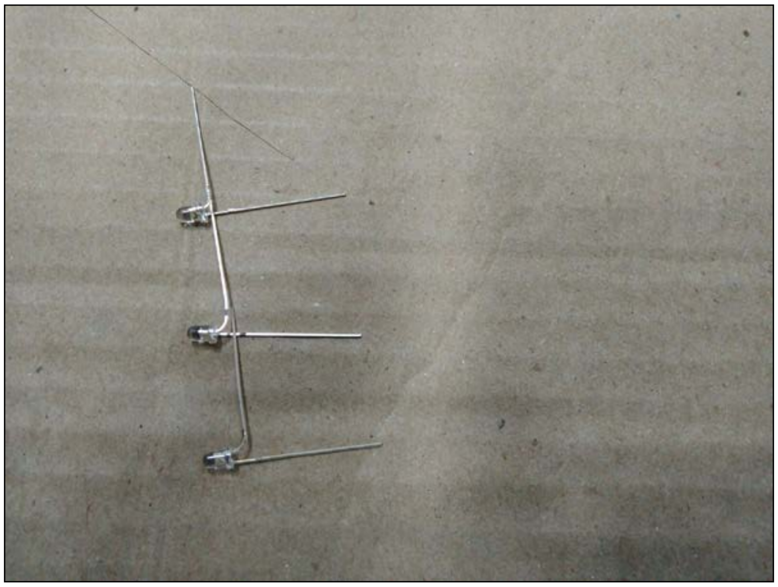
For the construction, you can either use a full metal structure to give rigidity to the structure at the cost of simplicity, or the other, simpler option is to overlap all the legs of the LEDs by about a quarter, which in turn will give the rigidity to the structure that we require. As shown in the following image, fold the cathode of your LEDs:

You can choose to bend the cathode to the left or right. Just make sure that you are consistent across the whole structure and it never touches the anode.

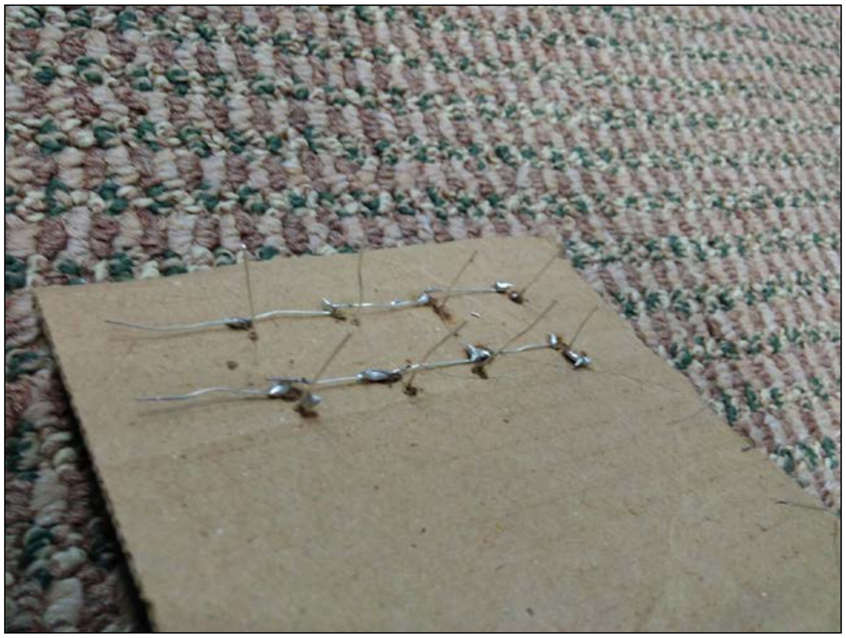


One of the critical parts of this project is making a structure to hold the LED while soldering. You can make this kind of structure by making a wooden jig or using a thermocol sheet. The thermocol sheet option is going to be a bit easier. Here are a couple of things that you should keep in mind while working on this part:

- As mentioned before, it will hold an LED layer so make sure it is accurate and not too loose.
- A quarter of the LED leg should overlap with the adjacent LED. If needed, use a ruler.
- If you are using a wooden jig, make sure your drill is the same size as the LED. If it is a bit tight, you won't be able to remove the fully soldered layer.



Now solder the cathodes of four rows of LEDs. After completing the first four rows, your structure will look something similar to the following image:



In the preceding image, only two rows of layers are being displayed. If you are working with copper LED cables, you would know that structures created using copper cables won't be that strong. To strengthen the rigidity of the layer, cut and solder two straight bits of craft wire to either end. Make sure they connect with each row. That's it. Your first layer is ready.

Before soldering other layers, you should test the layer that you have just created and correct any errors if you have made any, so that you don't repeat the same mistake while working on other layers. For that, open Arduino IDE and load the blink application that you developed in Module 1 *Chapter 1, Getting Started with Arduino*. For testing each LED, connect the ground pin to the layer frame with the connected resistor and press the positive lead to each LED.

If everything goes well, each LED will light up. If not, check the connection for that LED and make sure you haven't missed a solder joint somewhere. If the soldering and connection are right, replace the LED and test it again. There is a chance that the LEDs might have heated up if you kept the soldering iron connected to them for long.

Once you have tested the first layer, remove that layer from the wooden/thermocol jig and repeat the process three more times. Don't forget to test all the four layers of LEDs with the blink program.

Once you have completed and tested the four layers, you need to join all the vertical legs together. One of the techniques is to cut the piece of the cardboard.

It will help to keep all the layers at the same height. You will notice that even after using this there are lots of legs which don't align perfectly. You can use some crocodile pins to hold them in place. Another option is to get some copper cables and solder them at those places where the legs are not aligned properly.

Mistakes to avoid

When you are working with electronics, things can become quite tricky sometimes. Here are a couple of things that you should keep in mind, as most first-timers make mistakes with this:

- If you are using cardboard for getting equal height for the all the layers, make the cardboard longer on the side, and join the pieces of card outside of the cube, so when you've completed the layer, you can easily pull out the card.
- A general rule of thumb is the anode of an LED will connect with the anode of another LED, and similarly, the cathode of an LED will connect with the cathode of another LED. Don't connect/solder the anode of one LED with the cathode of another LED.

Make sure you test all four layers once you have connected them. The easiest way to test is to touch only on the uppermost layer. If that works, it means you have a good connection going through all the layers.

Don't forget to cut extra bits of metal frame and legs before you connect them to the circuit board or cardboard. An important thing to consider here is to not cut vertical legs as those need to put into the prototyping board.

Fixing to the board

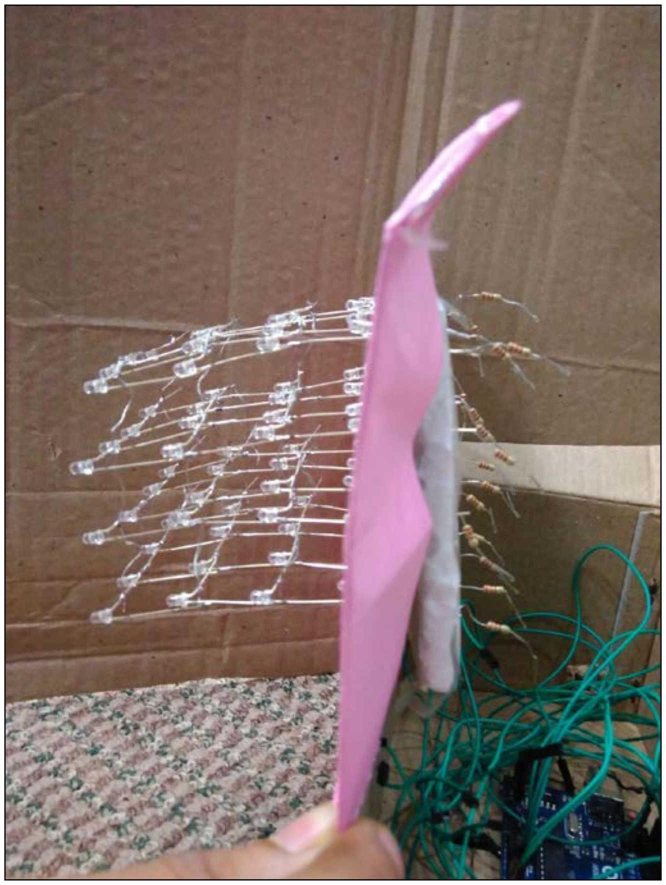
Before you move into the Arduino part, the last step is to fix this structure on a prototyping board. If you want to go ahead with a prototyping board, make sure you place each LED at equal distance, and for holding each LED leg you can use some crocodile pins.

[Personally, I found it a bit difficult, as once you have joined all the four layers, all the LED legs won't be at an equal distance, so instead I have used a cardboard sheet and soldered the LEDs on cardboard.]

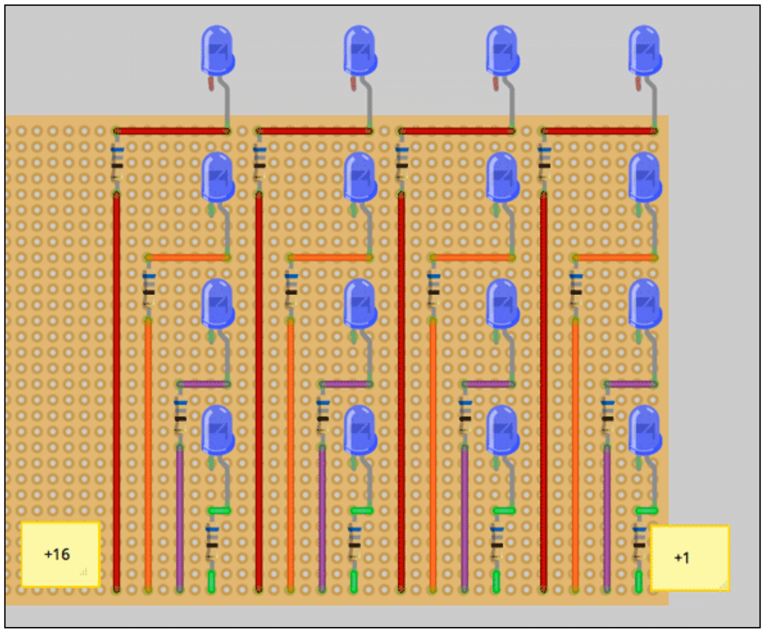
The next step is to connect the LEDs with the resistors. Ideally, you should have the resistors soldered on a prototyping board.

If you are using cardboard, as I did, you can use thermocol for connecting a resistor to an LED.

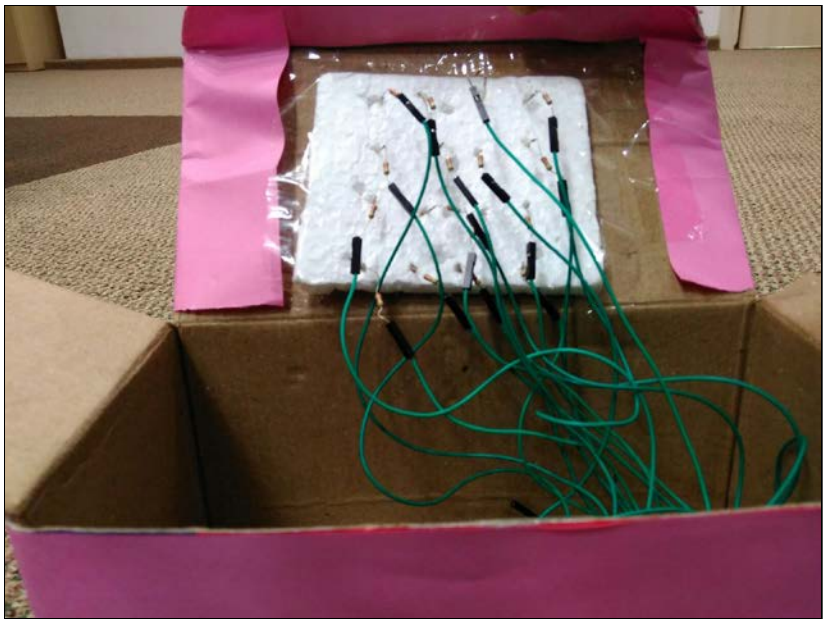
Once you have connected the resistors with the cube, it will look like the following image. Please note, I have used thermocol and cardboard for this prototype. If you are using a wooden base and a prototyping board, your prototype will look much more polished:



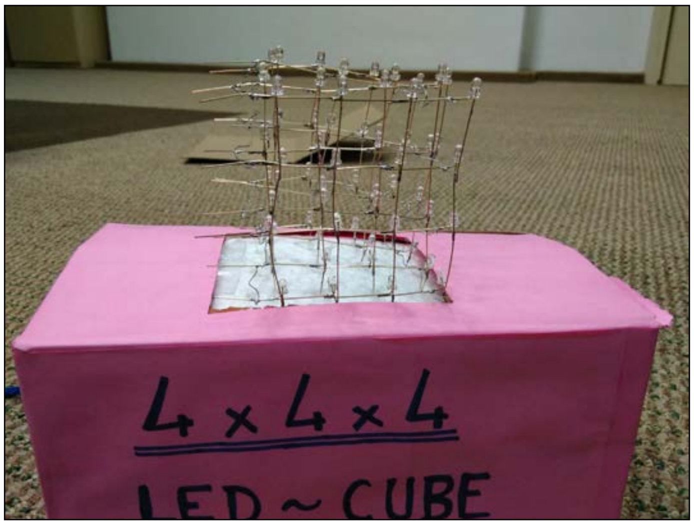
If you didn't plan your resistor placement in advance, here is what you will have once you have completed all the steps. A better way is to space them equally in a stepping fashion, so that then you could use one entire side of the cube for all the final connections to Arduino. Here's the circuit diagram:



Finally, connect some connecting wires, which can be plugged into relevant Arduino pins, and make sure you use long wires for this. You can use a color code for differentiating between connecting cables:

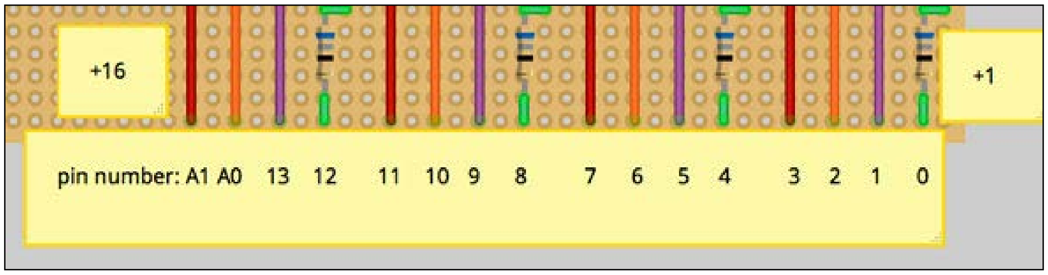


If you are having trouble getting different color cables and at the end you feel like your soldering and connections are going to be a bit messy, cover your cube internally with a cardboard box, as shown in the following figure:



Programming a 4*4*4 LED cube

Having done the hard part of soldering, let's get into Arduino connection and programming. Before connecting the positive leads, connect four negative layers to Arduino analog I/O ports A2 (bottom layer) through A5 (top layer). After that, 16 LED control pins needs to be connected to the Arduino board. Connect the first 14 pins to Arduino digital I/O ports 0 to 13. The remaining pins 15 and 16 need to be connected to Analog pins A0 and A1. See the following diagram for connection reference:



There are a few things that you should understand before programming your cube:

- To address a single LED use a plane (layer) number 0–3, and an LED pin number 0–15. Turn the plane to LOW output (negative leg) and the LED pin number to HIGH (positive leg) to activate the LED.
- Before activating a single LED, ensure all other planes are off by setting them to HIGH output. If you don't do this, the whole column of LEDs will light up instead of a single LED.

Now you are all set to start creating your own programming sequence.

Copy the following code (inspired from `http://www.tecnosalva.com/files/ficheros/ledcube2.ino`) and create a new Arduino sketch and upload the code to your Arduino board:

```
#include <avr/pgmspace.h> // allows use of PROGMEM to store patterns
in flash

#define CUBESIZE 4
#define PLANESIZE CUBESIZE*CUBESIZE
#define PLANETIME 3333 // time each plane is displayed in us -> 100 Hz
refresh
#define TIMECONST 5 // multiplies DisplayTime to get ms

/*
** Defining pins in array makes it easier to rearrange how cube is
wired
** Adjust numbers here until LEDs flash in order - L to R, T to B
** Note that analog inputs 0-5 are also digital outputs 14-19!
** Pin DigitalOut0 (serial RX) and AnalogIn5 are left open for future
apps
*/

int LEDPin[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
int LEDPinCount = 16;
int PlanePin[] = {16, 17, 18, 19};
int PlanePinCount  = 4;

// initialization
void setup()
{
  int pin; // loop counter
  // set up LED pins as output (active HIGH)
  for (pin=0; pin<PLANESIZE; pin++) {
    pinMode( LEDPin[pin], OUTPUT );
  }
  // set up plane pins as outputs (active LOW)
  for (pin=0; pin<CUBESIZE; pin++) {
    pinMode( PlanePin[pin], OUTPUT );
  }
}

void loop(){
  loopFor();
}

// the principles of using 4 planes and 16 pins - here we loop over
each, turning on and off in turn
void loopFor()
{
    for(int thisPlane = 0; thisPlane < PlanePinCount; thisPlane++){
      for(int thisPin = 0; thisPin < LEDPinCount; thisPin++){

        planesOff();
        digitalWrite(LEDPin[thisPin],HIGH);
        digitalWrite(PlanePin[thisPlane],LOW);


        delay(50);

        digitalWrite(LEDPin[thisPin],LOW);
        digitalWrite(PlanePin[thisPlane],HIGH);


      }
    }
}

void planesOff(){
    for(int thisPlane = 0; thisPlane < PlanePinCount; thisPlane++){
        digitalWrite(PlanePin[thisPlane],HIGH);
    }
}
```

You would notice that the preceding code simply lights every LED one by one, in sequence. We use two `for` loops for this, iterating over each layer and each control pin.

That's it! Of course, this is not the only way to control a 4*4*4 LED cube. There are multiple ways you can control LEDs.

Summary

So far in this book, the focus was more on understanding Arduino programming and less about electronics and soldering. If you reached the end of this chapter, this means you have learnt the very important skill that is soldering. Having understood soldering and Arduino programming, you can now bring your own ideas into reality/prototype. Share your prototype and experience on social media using hashtag `#ArduinoBLINK`.

Having gained sound knowledge of Arduino programming and soldering, in the next chapter you will learn about sound visualization and how to use different sensors with Arduino.

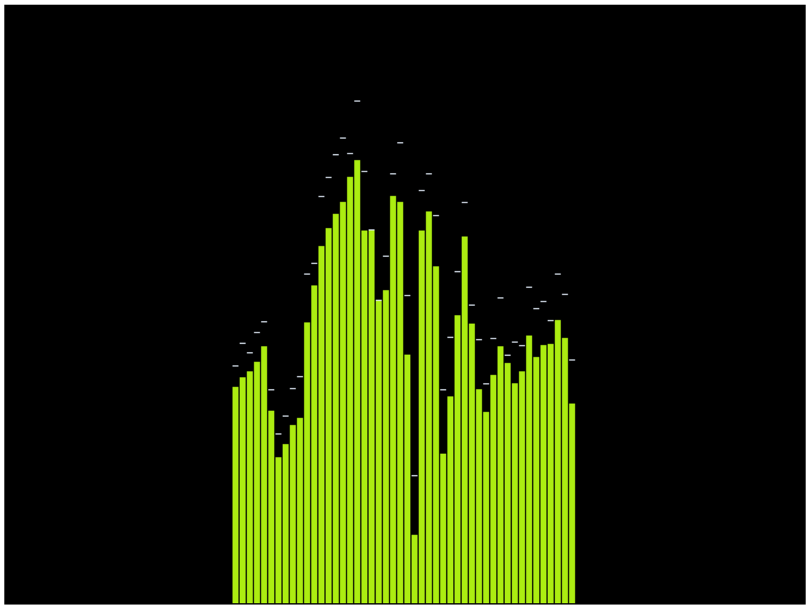
4
Sound Visualization and LED Christmas Tree

Things get pretty easy once you have understood the basics of Arduino and how to control the "stuff" with Arduino. In the previous chapters, we have developed some useful projects using LEDs and light sensors. We also learned soldering in the previous chapter. In this chapter, we will understand how to visualize sound using Arduino and then we will develop an LED Christmas tree.

- Introduction to sound visualization
- Sound visualization using Arduino
- Developing a sound controlled Christmas tree

Introduction to sound visualization

Sound visualization, or music visualization, has been an integral part of the music industry since the evolution of media players. For example, on your computer, when you play any music, you can see the visualization in the default media player or VLC media player, as shown in the following image:



You have noticed that, as the loudness of the music, or the frequency of the sound changes, the visualization changes. We will use the same principle to visualize sound using Arduino.

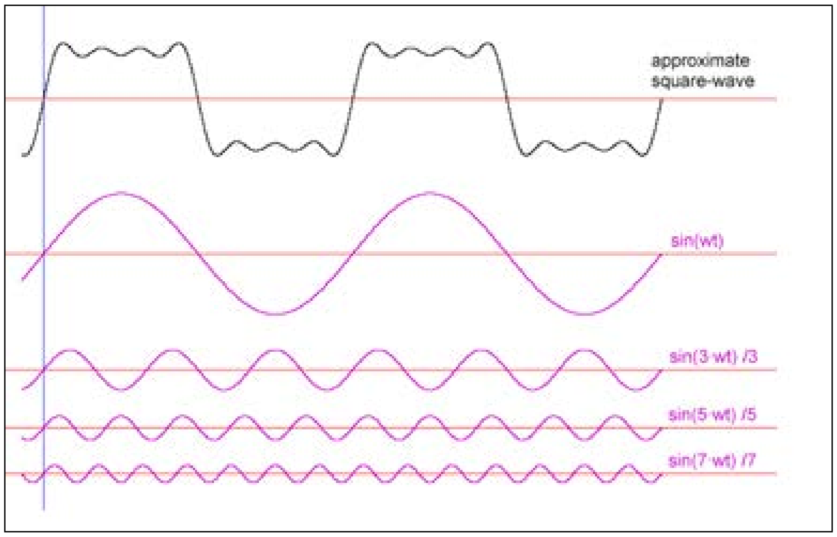
How to visualize the sound

In simple terms, sound/music is nothing but a series of signals with a certain frequency and a certain amplitude. We can get the value at each point by using `analogRead()`. But, this function would be much too slow for sampling audio. So, we will use the microcontroller's analog-to-digital converter, which automatically takes repeated analog intervals at precise intervals. We will learn both ways of sampling audio. There are a number of algorithms available for analyzing the sampled audio. But, for fast performance, we will use a FFT (Fast Fourier Transform) algorithm.

What is FFT (fast fourier transform)

Fast Fourier Transform is one of the basic and most important numerical algorithms in the field of signal processing. It converts a signal from its time domain (time domain refers to analysis of a signal with respect to time) to frequency domain. Frequency domain refers to the analysis of a mathematical function, or here a signal with respect to frequency.

You can understand the time domain and frequency domain with the following image:



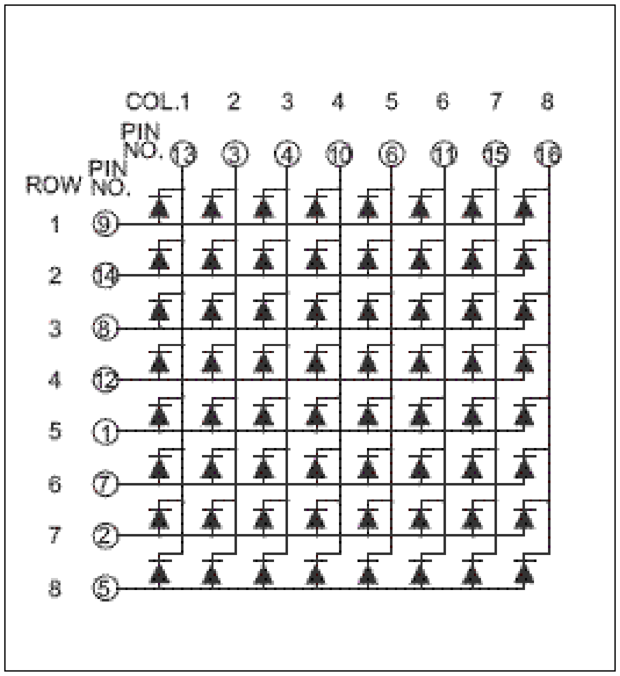
You can decompose any signal into a bunch of sine waves (of different amplitude, frequency, and phase). An analogy for this is the images you see on your computer screen, which are composed of red, green, and blue dots (of different magnitude and frequency).

FFT decomposes sine waves from an arbitrary signal by multiplying the arbitrary signal with a sine wave of a specific frequency. As the match is closer, the resulting summed product value will be higher. Once we have the signal, which is converted to the frequency domain, it becomes easier to analyze.

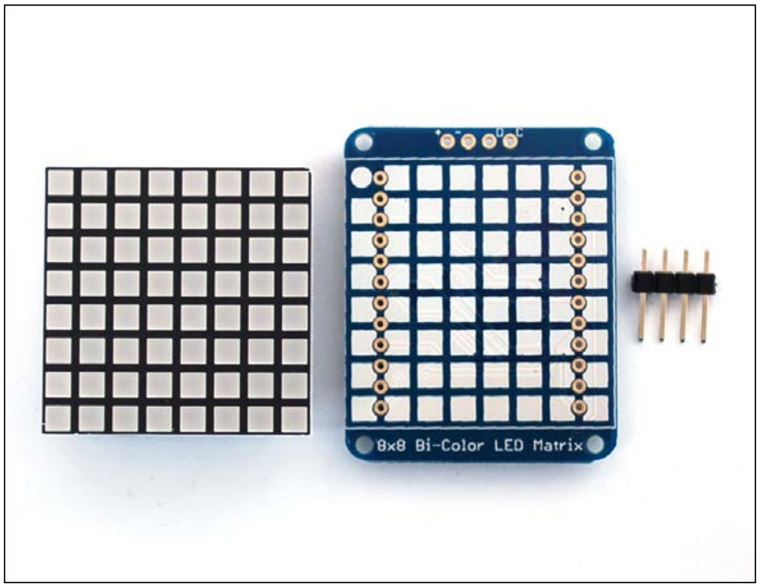
Sound visualization using Arduino

After understanding the basics of sound visualization, we will move on to implement sound visualization using Arduino. Before we develop our LED Christmas tree, we will develop sound visualization on an LED matrix.

An LED matrix is a combination of 64 LEDs connected together as shown in the following circuit. As Arduino doesn't have 64 pins, we can't connect individual pins to control each LED. Instead, we will use the concept of multiplexing:



With the use of multiplexing, we can control any number of LEDs with Arduino. For controlling an 8 x 8 LED matrix, we need one multiplexer circuit. We can use the backpack from adafruit for the multiplexer, or a MAX7219 Dot Matrix MCU Control for the multiplexer part:



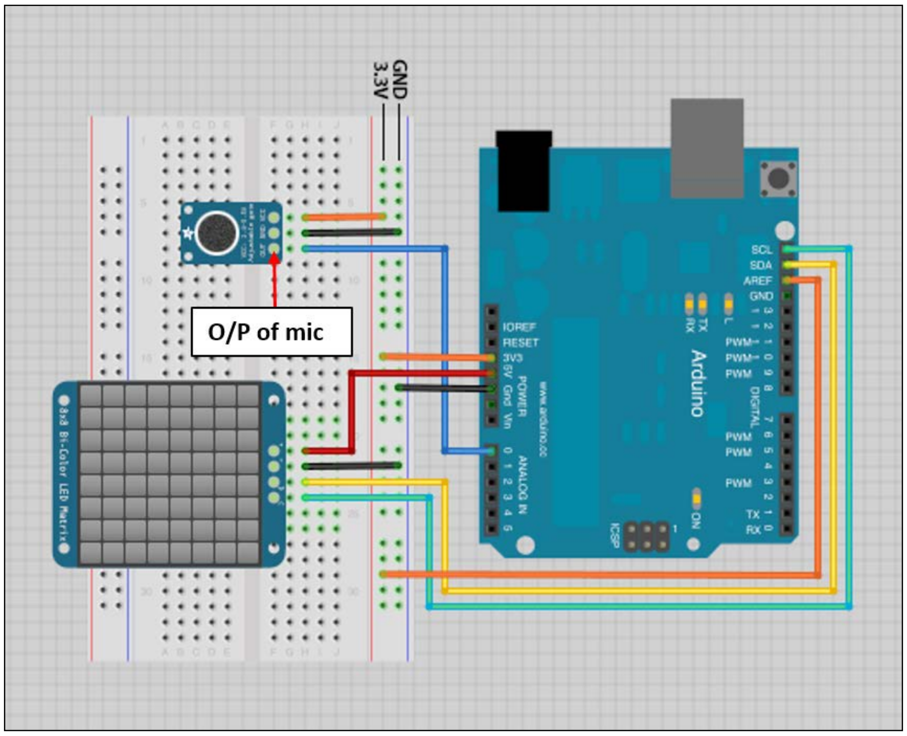
So, now we do not need to use a lot of pins to control this LED matrix. Instead, we require only four pins to control this LED matrix.

You will need the following components for a music-controlled LED matrix:

- Analog Mic Sensor
- 8 x 8 LED matrix with backpack

Connect the mic and LED matrix as shown in the circuit. Connect a 3.3 V pin to the AREF of Arduino and the mic's Vcc in pin. This is important for making a reference of 3.3 V to analog input from the mic. Connect the Arduino 5 V pin to the + pin of the LED matrix. Connect the Arduino analog pin A0 to Mic Output. Connect the Arduino SDA and SCL pin to the D (data) and C (clock) pin of the backpack. Earlier versions of Arduino might not have these pins.

Use analog pins 4 and 5 for this purpose instead. Finally, don't forget to connect Arduino Ground to Mic GND and backpack GND:



Once you have connected the circuit as shown in the diagram, upload the following code to Arduino:

```
#include <avr/pgmspace.h>
// Useful for Sound Analysis
#include <ffft.h>
#include <math.h>
// Required for communication with I2C device
#include <Wire.h>
#include <Adafruit_GFX.h>
// Required for Adafruit LED backpack
#include <Adafruit_LEDBackpack.h>

// Microphone connects to Analog Pin 0. ADC Pin 0 for Arduino Uno
#define ADC_INPUT 0

int16_t audio_capture[FFT_N];
```

```
complex_t fft_buffer[FFT_N];
uint16_t output_spectrum[FFT_N / 2];
volatile byte buffer_position = 0;

byte peakValue[8],dotCount = 0,colCount = 0;
int col[8][10], minAvgLevel[8], maxAvgLevel[8], colDiv[8];

static const uint8_t PROGMEM
// Noise to be removed from each column. Adjust the values as per the
requirements
noiseToDeduct[64] = { 8, 6, 6, 5, 3, 4, 4, 4, 3, 4, 4, 3, 2, 3, 3, 4,
              2, 1, 2, 1, 3, 2, 3, 2, 1, 2, 3, 1, 2, 3, 4, 4,
              3, 2, 2, 2, 2, 2, 2, 1, 3, 2, 2, 2, 2, 2, 2, 2,
              2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4
            },
// Equalizer to remove the noise and neutralize the noise at the bass
end.
equalizer[64] = {
  255, 175, 218, 225, 220, 198, 147, 99, 68, 47, 33, 22, 14,  8,  4,
2,
  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
},

// We want to fit the output of FFT spectrum to 8 columns.
// All the bins from the output spectrum is not useful.
// we will following bins for the column output.
// Below table contains details about number of bins to take and index
from where it will start
// along with the nin numbers.
column0[] = {  2,  1,
                111,   8
             },
column1[] = {  4,  1,
              19, 186,  38,   2
              },
column2[] = {  5,  2,
              11, 156, 118,  16,   1
              },
column3[] = {  8,  3,
              5,  55, 165, 164,  71,  18,   4,   1
              },
column4[] = { 11,  5,
              3,  24,  89, 169, 178, 118,  54,  20,   6,   2,   1
              },
column5[] = { 17,  7,
              2,   9,  29,  70, 125, 172, 185, 162, 118, 74,
              41,  21,  10,   5,   2,   1,   1
              },
column6[] = { 25, 11,
              1,   4,  11,  25,  49,  83, 121, 156, 180, 185,
              174, 149, 118,  87,  60,  40,  25,  16,  10,   6,
              4,   2,   1,   1,   1
              },
column7[] = { 37, 16,
              1,   2,   5,  10,  18,  30,  46,  67,  92, 118,
              143, 164, 179, 185, 184, 174, 158, 139, 118,  97,
              77,  60,  45,  34,  25,  18,  13,   9,   7,   5,
              3,   2,   2,   1,   1,   1,   1
              },
// This contains list of all the data bin for all 8 columns
* const binsToUse[]  = {
  column0, column1, column2, column3,
  column4, column5, column6, column7
};

Adafruit_BicolorMatrix LEDmatrix = Adafruit_BicolorMatrix();

void setup() {
  uint8_t i, j, nBins, binNum, *outputData;

  memset(peakValue, 0, sizeof(peakValue));
  memset(col , 0, sizeof(col));

  for (i = 0; i < 8; i++) {
    minAvgLevel[i] = 0;
    maxAvgLevel[i] = 512;
    outputData = (uint8_t *)pgm_read_word(&binsToUse[i]);
    nBins = pgm_read_byte(&outputData[0]) + 2;
    binNum = pgm_read_byte(&outputData[1]);
    for (colDiv[i] = 0, j = 2; j < nBins; j++)
      colDiv[i] += pgm_read_byte(&outputData[j]);
  }

  LEDmatrix.begin(0x70);
  // Init ADC free-run mode; f = ( 16MHz/prescaler ) / 13 cycles/
conversion
  ADMUX  = ADC_INPUT; // Channel sel, right-adj, use AREF pin
  ADCSRA = _BV(ADEN)  | // ADC enable
           _BV(ADSC)  | // ADC start
           _BV(ADATE) | // Auto trigger
           _BV(ADIE)  | // Interrupt enable
           _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0); // 128:1 / 13 = 9615
Hz
  ADCSRB = 0;
  DIDR0  = 1 << ADC_INPUT;
  TIMSK0 = 0;

  sei(); // Enable interrupts
}

void loop() {
  uint8_t  i, x, L, *outputData, nBins, binNum, weighting, c;
  uint16_t minLvl, maxLvl;
  int      level, y, sum;

  while (ADCSRA & _BV(ADIE)); // Wait for audio sampling to finish

  fft_input(audio_capture, fft_buffer);   // Samples -> complex #s
  buffer_position = 0;                   // Reset sample counter
  ADCSRA |= _BV(ADIE);             // Resume sampling interrupt
  fft_execute(fft_buffer);          // Process complex data
  fft_output(fft_buffer, output_spectrum); // Complex -> spectrum

  // Remove noise and apply equalizers
  for (x = 0; x < FFT_N / 2; x++) {
    L = pgm_read_byte(&noiseToDeduct[x]);
    output_spectrum[x] = (output_spectrum[x] <= L) ? 0 :
                  (((output_spectrum[x] - L) * (256L - pgm_read_
byte(&equalizer[x]))) >> 8);
  }

  // Fill background w/colors, then idle parts of columns will erase
  LEDmatrix.fillRect(0, 0, 8, 3, LED_RED);    // Upper section
  LEDmatrix.fillRect(0, 3, 8, 2, LED_YELLOW); // Mid
  LEDmatrix.fillRect(0, 5, 8, 3, LED_GREEN);  // Lower section

  // Downsample spectrum output to 8 columns:
  for (x = 0; x < 8; x++) {
      outputData   = (uint8_t *)pgm_read_word(&binsToUse[x]);
      nBins  = pgm_read_byte(&outputData[0]) + 2;
      binNum = pgm_read_byte(&outputData[1]);
      for (sum = 0, i = 2; i < nBins; i++)
        sum += output_spectrum[binNum++] * pgm_read_
  byte(&outputData[i]); // Weighted
      col[x][colCount] = sum / colDiv[x];                    // Average
      minLvl = maxLvl = col[x][0];
      for (i = 1; i < 10; i++) { // Get range of prior 10 frames
        if (col[x][i] < minLvl)
        {
          minLvl = col[x][i];
        }
        else if (col[x][i] > maxLvl)
        {
          maxLvl = col[x][i];
        }
      }
      // minLvl and maxLvl indicate the extents of the FFT output, used
      // for dynamically setting the min and max level of the column.

      if ((maxLvl - minLvl) < 8)
      {
        maxLvl = minLvl + 8;
      }
      minAvgLevel[x] = (minAvgLevel[x] * 7 + minLvl) >> 3; // Dampen
  min/max levels
      maxAvgLevel[x] = (maxAvgLevel[x] * 7 + maxLvl) >> 3; // (fake
  rolling average)

      // Second fixed-point scale based on dynamic min/max levels:
      level = 10L * (col[x][colCount] - minAvgLevel[x]) /
              (long)(maxAvgLevel[x] - minAvgLevel[x]);

      // Clip output and convert to byte:
      if (level < 0L)
      {
        c = 0;
      }
      else if (level > 10)
      {
        c = 10; // Allow dot to go a couple pixels off top
      }
      else
```

```
    {
      c = (uint8_t)level;
    }

    if (c > peakValue[x])
    {
      peakValue[x] = c; // Keep dot on top
    }

    if (peakValue[x] <= 0) // No output
    {
      LEDmatrix.drawLine(x, 0, x, 7, LED_OFF);
      continue;
    }
    else if (c < 8) // Partial column?
    {
      LEDmatrix.drawLine(x, 0, x, 7 - c, LED_OFF);
    }

    // The 'peak' dot color varies, but doesn't necessarily match
    // the three screen regions...yellow has a little extra influence.
    y = 8 - peakValue[x];
    if (y < 2)
    {
      LEDmatrix.drawPixel(x, y, LED_RED);
    }
    else if (y < 6)
    {
      LEDmatrix.drawPixel(x, y, LED_YELLOW);
    }
    else
    {
      LEDmatrix.drawPixel(x, y, LED_GREEN);
    }
  }

  LEDmatrix.writeDisplay();

  // Every third frame, make the peak pixels drop by 1:
  if (++dotCount >= 3)
  {
    dotCount = 0;
    for (x = 0; x < 8; x++)
    {
      if (peakValue[x] > 0) peakValue[x]--;
    }
  }

  if (++colCount >= 10)
  {
    colCount = 0;
  }
}

ISR(ADC_vect) { // Audio-sampling interrupt
  static const int16_t noiseThreshold = 4;
  int16_t sample = ADC; // values between 0-1023

  audio_capture[buffer_position] =
    ((sample > (512 - noiseThreshold)) &&
     (sample < (512 + noiseThreshold))) ? 0 :
    sample - 512; // Sign-convert for FFT; -512 to +511

  if (++buffer_position >= FFT_N) ADCSRA &= ~_BV(ADIE); // Turn off
the interrupt once buffer is full
}
```

Let's understand the code in detail. Here, as we discussed earlier, we are using onboard ADC for sampling the audio. While dealing with sampling/signal processing, one of the most useful features of Arduino that comes in handy is interrupts. Before we understand how interrupts work, we need to know about interrupts.

Interrupts allow certain important tasks to happen in the background and will interrupt the execution flow when a particular event occurs. In our case, interrupts are used for sampling the audio. Once the audio is processed with an FFT algorithm and is rendered on the display, it will again sample the audio.

Although we can't hear all the sounds, there is some background noise present around us. We have to remove the ambient noise before processing music to render on the LED matrix. So, we have stored 64 values in the array `noiseToDeduct`, which will be deducted from the signal afterwards. If you are getting more noise in the output, you can adjust the noise by setting the different values in the array.

Apart from the noise, we also need to equalize the sound towards the bass end of the sound. So, we are storing different values in an array `equalizer`. After noise is deducted from the output of the filter, we will normalize the sound/output spectrum.

Also, the output spectrum of FFT will be having much more buckets/samples then we can use. The bottom-most and several at the top of the spectrum are either noisy or out of range. So, we will use the FFT spectrum output values stored in the `column0`, `column1`, `column2`, `column3`, `column4`, `column5`, `column6`, and `column7` array. The array contains the number of the output values to be used and the starting index of the bins along with their weights. Although these values can be changed as per our requirements, these values are derived after thorough testing by adafruit.

Two new functions that we have used here are: `pgm_read_word` and `pgm_read_byte`. Both functions are provided by the `avr/pgmspace.h` library. `pgm_read_word` is used for reading a word from the program space with a 16-bit (near) address. Similarly, `pgm_read_byte` is used for reading a byte from the program space with a 16-bit (near) address:

```
outputData   = (uint8_t *)pgm_read_word(&binsToUse[x]);
    nBins  = pgm_read_byte(&outputData[0]) + 2;
    binNum = pgm_read_byte(&outputData[1]);
```

For getting data from the `binsToUse` array, we used `pgm_read_word`. While reading a particular value from `outputData`, we have used `pgm_read_byte`.

Here, we are using a 128-bit buffer for storing the sampled audio. Whenever the buffer is full, it will process the data by passing it to the FFT filter. After the output spectrum from the FFT, the signal is again down-sampled:

```
       sum += output_spectrum[binNum++] * pgm_read_
byte(&outputData[i]); // Weighted
    col[x][colCount] = sum / colDiv[x];                    // Average
```

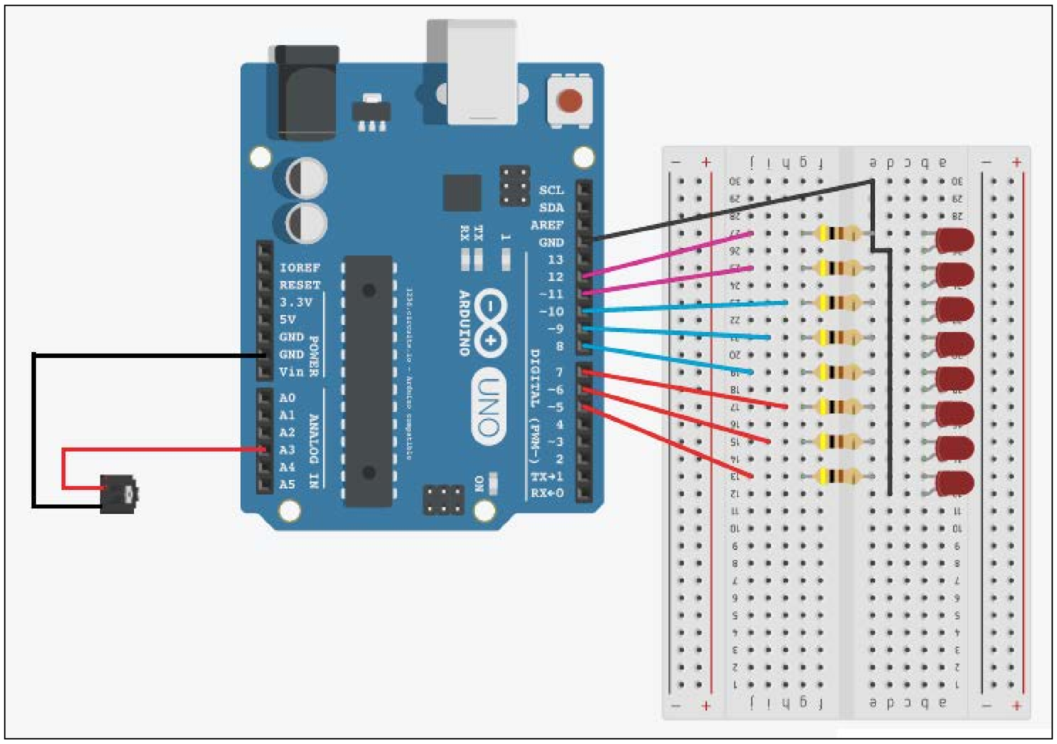
We are finding a weighted average for each column by selecting the bin from the predefined table initialized at the beginning. After finding the weighted average for eight columns, we are writing these values to the LED matrix display after storing the minimum and maximum values of the output spectrum. These dynamic minimum and maximum values are useful for making the display interesting, even at a low volume.

After understanding sound visualization using FFT, we will now develop an LED Christmas tree that syncs its lighting with beats.

Developing an LED Christmas tree

We are now familiar with the concept of sound visualization. We have also learnt about controlling an LED matrix with music. Now, we will develop an LED Christmas tree, which will blink the LEDs as per the music beats.

To develop the basic circuit which responds to the beats, connect the circuit as shown in the following image:



We will connect the audio input/mic to the analog pin 3 of the Arduino. We have connected LEDs to pins 5 to 12.

Once you have connected the circuit as mentioned, upload the following code on the Arduino:

```
#include <fix_fft.h>

int LEDPins[] = {5, 6, 7, 8, 9, 10, 11, 12};
int x = 0;
char imaginary[128], inputSignal[128];
char outputAverage[14];
int i = 0, inputValue;
#define AUDIOPIN 1

void setup()
{
  for (int i = 0; i < 8; i++)
  {
    pinMode(LEDPins[i], OUTPUT);
  }
  Serial.begin(9600);
}

void loop()
{
  for (i = 0; i < 128; i++) {
    inputValue = analogRead(AUDIOPIN);
    inputSignal[i] = inputValue;
    imaginary[i] = 0;
  };
  fix_fft(inputSignal, imaginary, 7, 0);
  for (i = 0; i < 64; i++) {
    inputSignal[i] = sqrt(inputSignal[i] * inputSignal[i] +
imaginary[i] * imaginary[i]);  // this gets the absolute value of the
values in the
    //array, so we're only dealing with positive numbers
  };

  // average bars together
  for (i = 0; i < 14; i++) {
    outputAverage[i] = inputSignal[i * 4] + inputSignal[i * 4 + 1] +
inputSignal[i * 4 + 2] + inputSignal[i * 4 + 3]; // average together
    outputAverage[i] = map(outputAverage[i], 0, 30, 0, 9);
  }
  int value = outputAverage[0];//0 for bass
  writetoLED(value);
}

void writetoLED(int mappedSignal)
{
  if (mappedSignal > 8)
  {
    for (int i = 0; i < 8; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
  }
  else if (mappedSignal > 7)
  {
    for (int i = 0; i < 7; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 7; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else if (mappedSignal > 6)
  {
    for (int i = 0; i < 6; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 6; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else if (mappedSignal > 5)
  {
    for (int i = 0; i < 5; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 5; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else if (mappedSignal > 4)
  {
    for (int i = 0; i < 4; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 4; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else if (mappedSignal > 3)
  {
    for (int i = 0; i < 3; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 3; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else if (mappedSignal > 2)
  {
    for (int i = 0; i < 2; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 2; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else if (mappedSignal > 1)
  {
    for (int i = 0; i < 1; i++)
    {
      digitalWrite(LEDPins[i], HIGH);
    }
    for (int i = 1; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
  else
  {
    for (int i = 0; i < 8; i++)
    {
      digitalWrite(LEDPins[i], LOW);
    }
  }
}
```

Here, we are directly reading the signal from the mic by using `analogRead()`. We are taking 128 readings. After taking all the readings, we are processing those readings using the `fix_fft.h` library, which is available at `http://forum.arduino.cc/index.php/topic,38153.0.html`. After storing both `fix_fft.h` and `fix_fft.cpp` in your machine, you need to import this library in your Arduino code.

After getting the output spectrum from the FFT, we are taking the average value of the output for turning on or off the LEDs based on the average value. As the bass part most often provides harmonics and rhythmic support, we are using the bass value for controlling the LEDs. We are getting the bass value as `outputAverage[0]`. Mapping of the average value from 0 to 9 helps in setting the LEDs to on or off easily.

For artistic purposes, we will connect multiple LEDs around the Christmas tree model. So, the Christmas tree will sync its lighting with the beats of the music.

Summary

In this chapter, we started with the basics of sound visualization. After understanding the FFT algorithm, we move on to visualize the sound with Arduino using the same on an 8 x 8 dot matrix LED display. At the end of this chapter, we developed an LED Christmas tree, which syncs its light as per the beats in the music. In the next chapter, we will move on to develop "persistence of vision".

5 Persistence of Vision

So far in this book, we have made all things that are stationary in nature—that is, they can't move. In the final project of this book, we will create an even more intensive experience by moving LEDs using motors. We will create a Persistence of Vision wand using an LED array and motor. But, first of all, you will get introduced to LED arrays and motors. Along with the different type of motors, you will get to know about their pros and cons. In this chapter, we will cover the following topics:

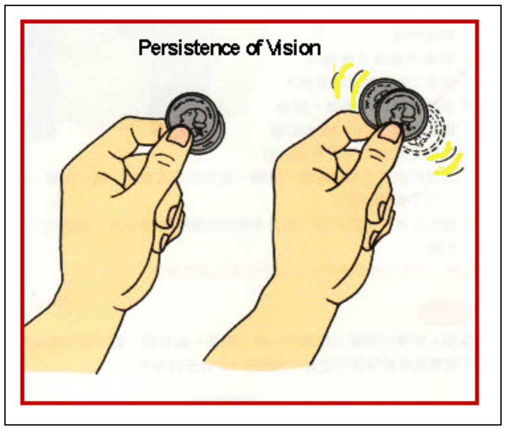
- Persistence of Vision
- Programming an LED array
- Controlling a motor using Arduino
- Synchronizing LED array timing based on the speed of the motor

Creating your own Persistence of Vision

One of the five sensory organs of our body, the eye is a remarkable instrument that helps us to process light in such a way that our mind can create meaning from it. Persistence of Vision refers to an optical illusion where multiple different images blend into a single image in the human mind.

The Persistence of Vision illusion plays a role in keeping the world from going pitch black every time we blink our eyes. Whenever a retina is hit by light, it keeps an impression of the light for about a tenth of a second after the light source is removed. Due to this, the eye can't distinguish between changes that occur faster than this retention period. This similar phenomenon is used in motion pictures or, as we call it, "flicks". The motion picture creates an illusion by rapidly sequencing individual photographs. Usually for motion pictures, the rate of frames per second is 24, which leads to a flicker-free picture. If frames per second is kept below 16, the mind can distinguish between the images, which leads to flashing images, or flicker.

Look at the following image; when you move your hands back and forth as demonstrated, you will see a flicker at all the positions:



A renowned professor at the University of Central Arkansas quoted this:

> *"The notion of 'persistence of vision' seems to have been appropriated from psychology in the first decade of the century, the period during which cinema came into being. But while most film scholars accepted 'persistence of vision' as the perceptual basis of the medium and proceeded to theorise about the nature, meaning and functioning of cinema from that base, perceptual psychologists continued to question the mechanisms involved in motion perception; and they have achieved insights that necessitate the re-thinking of many conclusions reached by film scholars during the past 50 years."*

After getting introduced to the concept of Persistence of Vision, let's dive into how we can make our own PoV. For that we will need the following components:

- Arduino
- LED array
- DC motor
- Resistor
- L293D motor driver
- Wooden material for making the base of the PoV

In the upcoming section, you will learn how to use these components to create your own Persistence of Vision.

Programming an LED array

An LED array is nothing but a few LEDs connected together. Usually, an LED array comes in sizes of eight LEDs and 16 LEDs. You can control an LED array directly using the `digitalWrite()` function. Apart from using the `digitalWrite()` function, you can control LEDs directly using port-level communication. On Arduino, we have three ports: ports B, C, and D:

- Port B: Digital pins 8 to 13
- Port C: Analog pins
- Port D: Digital pins 0 to 7

Each port is controlled by three DDR registers. These registers are defined variables in Arduino as DDRB, DDRC, and DDRD. Using these variables, we can make the pins either as input or output in the setup function.

You can use the following syntax to initialize the pins:

```
DDRy = Bxxxxxxxx
```

Here, *y* is the name of the port (B/C/D) and x is the value of the pin that determines if the pin is input or output. We will use 0 for input and 1 for output. LSB (least significant byte) is the lowest pin number for that register.

To control pins using this port manipulation, we can use the following syntax:

```
PORTy = Bxxxxxxxx
```

Here, *y* is the name of the port (B/C/D), and make *x* equal 1 for making a pin HIGH and 0 for making the pin LOW.

You can use the following code to control the LEDs using port level communication:

```
void setup()
{
  DDRD = B11111111; // set PORTD (digital 7-0) to outputs
}

void loop()
{
  PORTD = B11110000; // digital 4~7 HIGH, digital 3-0 LOW
  delay(2000);
  PORTD = B00001111; // digital 4~7 LOW, digital 3-0 HIGH
  delay(2000);
}
```

There are a few pros and cons for using a port manipulation technique. Following are the disadvantages of using a port manipulation technique:

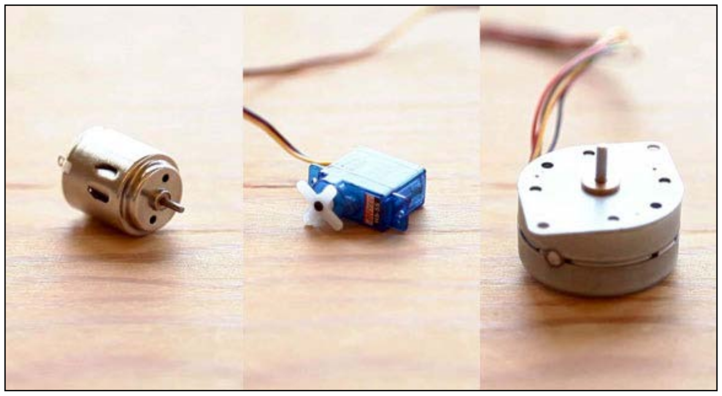
- The code becomes more difficult to debug and maintain and it takes a lot of time to understand the code.
- The code becomes less portable. If you use `digitalWrite()` and `digitalRead()`, it is much easier to write a code that will run on all microcontrollers, whereas ports and registers can be different for each kind of microcontroller.
- You might cause unintentional malfunctions with direct port access, as pin 0 is the receive line for the serial port. If you accidently make it input, it may cause your serial port to stop working.

There are a few advantages that a port manipulation technique has over normal code practices:

- In case of time constraint uses of Arduino, you would need to turn pins on or off very quickly. Using direct port access, you can save many machine cycles.
- Also, if you are running low on program memory, you can use these techniques to make your code smaller.

Different types of motors

Depending upon your project needs, you can choose from the variety of motors available in the market. For hobby electronics, you will mostly use either DC motor, servo motor, or stepper motor. The following image shows all three types of motors. From left to right, DC motor, servo motor, and stepper motor:



Let's get an overview about all the different types of motors.

DC motors

DC (Direct Current) motors are two-wire continuous rotational motors. When power is supplied to the motor, it will start running and will stop once power is removed. Most DC motors run at high speed, that is, high RPM (rotations per minute). The speed of the DC motor is controlled using the PWM technique (as discussed in Module 2 *Chapter 1*, *Project 1 – LED Night Lamp*). The duty cycle will determine the speed of the motor. The motor seems to be continuously running as each pulse is very rapid.

Servo motors

Servo motors are self-contained electric devices that rotate or push parts of a machine with great precision. A servo motor uses closed loop position feedback to control its motion and final position. Servo motors are intended for use in more specific tasks, where position needs to be accurate, such as moving a robotic arm. For this purpose, servo motors are an assembly of a DC motor, control circuit, potentiometer, and a gearing set.

The angle of rotation is limited to 180 degrees compared to the free run of a normal DC motor. They usually have three wires consisting of power, ground, and signal. To run the servo motors, continuous power is required. By giving a signal of proper value to the signal pin of the servo, one can control the position of the servo motor.

When a servo is given the signal to move, if any external force is applied to change its position, the servo motor will try to hold on to its position. A servo motor uses an integrated controller circuit to position itself.

Stepper motors

A stepper motor is a DC motor that moves in discrete steps. A stepper motor utilizes multiple toothed electromagnets arranged around a central gear to set the position. A stepper motors require an external control circuit to energize each electromagnet and make the motor shaft turn.

Stepper motors are available in two types, that is, unipolar and bipolar. Bipolar motors are the strongest type of stepper motor, having four or eight leads. Unipolar stepper motors are simpler compared to bipolar motors. Unipolar motors can step without reversing the direction of the current in the coils, because it is having a centre tap internally. However, because of the centre tap, unipolar motors have less torque compared to bipolar motors.

The basic difference between servo motors and stepper motors is the type of motor and how it is controlled. Stepper motors typically use 50 to 100 pole brushless motors, while servo motors have only 4 to 12 poles.

Different applications of motors

This is a very condensed overview of a somewhat complicated field:

- **DC motors**: Used in fans, car wheels, and so on, which need fast and continuous rotation motors.
- **Servo motors**: Servo motors are usually suited for robotic arms/legs where fast, high torque, and accurate rotation within a limited angle is required.
- **Stepper motors**: For devices where precise rotation and accurate control is required, stepper motors are used. Stepper motors have an advantage over servo motors in positional control because a stepper motor has positional control due to its nature of rotation by fractional increments.

Controlling a DC motor using Arduino

In this chapter, we will get to know how to control a DC motor with Arduino.

You can also run the DC motor by using the same code as the LED blink. We can consider the motor as an LED. As discussed earlier, a DC motor is a two-wired motor. One wire is the positive supply and other is ground. If we connect the positive terminal of the motor to the positive terminal of the battery, and the negative terminal of the motor to the negative terminal of the battery, the motor will run in one direction, and if we reverse the connection, the motor will run in the reverse direction.

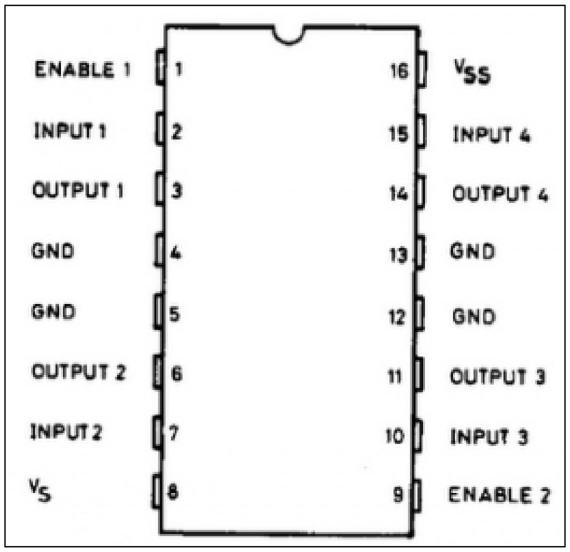
By connecting the motor to two digital pins of the Arduino, we can control the direction of the motor. In the following basic code, we will run the motor in one direction for five seconds and then we will reverse the direction of the motor. Connect pin 3 and pin 4 of the Arduino with the two wires of the motor:

```
int motorPos = 3;
int motorNeg = 4;

void setup() {
  pinMode(motorPos, OUTPUT);
  pinMode(motorNeg, OUTPUT);
}

void loop() {
  //run the motor in one direction
  digitalWrite(motorPos, HIGH);
  digitalWrite(motorNeg, LOW);
  delay(5000); //Run for 5 seconds
  // Reverse the direction of the motor
```

```
    digitalWrite(motorPos, LOW);
    digitalWrite(motorNeg, HIGH);
    delay(5000);
  }
```

In the preceding code, we are giving opposite outputs to both pins 3 and 4, which is useful in defining the direction of the motor.

Although this method of controlling DC motors directly with Arduino seems very easy, it has its own disadvantages. From the Arduino I/O pin, one can draw the maximum current of 20 mA. So, if we connect the heave load to Arduino, Arduino might get busted. For this purpose, we use an L293D chip, which is compatible with H-bridge connections. H-bridge is a circuit, which can drive the motor in both directions. Before we connect L293D to Arduino, check out all the pin details in the following image:



Connect **Enable1** and **Enable2** with a 5 V constant logic supply. L293D IC is designed in such a way that the left pins of the IC can be used to control one motor and the right pins of the IC can be used to control another motor in both directions. For controlling one motor, give input to pin 2 and 7, which will give output at pin 3 and pin 7. Pin 8 is the power supply for the motors. Connect a 5 V logic supply to pin 16.

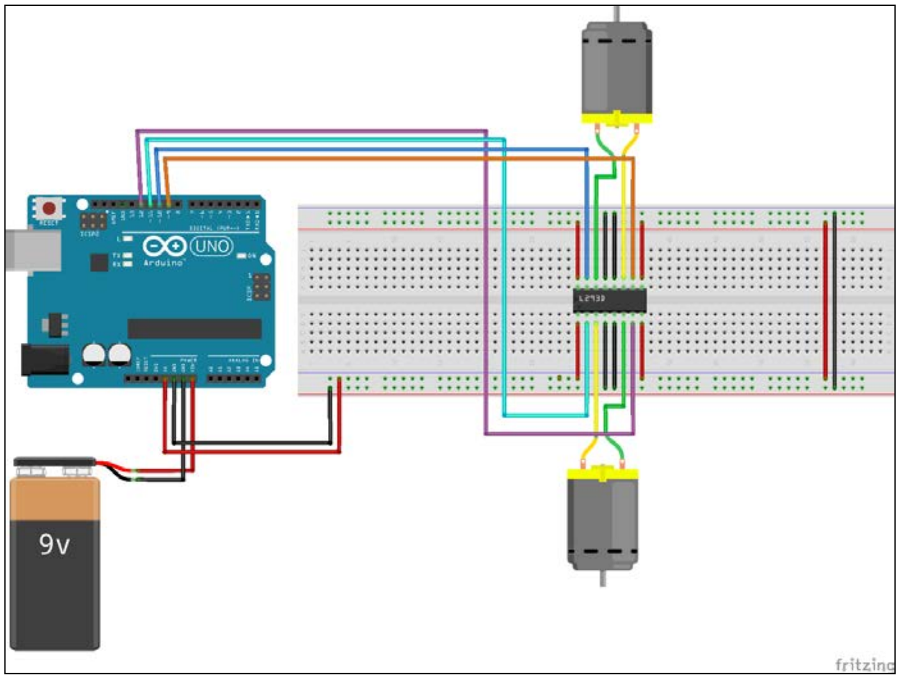
In most cases, a voltage regulator is required as the controller can't handle voltage more than 5 V. But, Arduino UNO has an in-built voltage regulator. Although you can give up to 20 V to the Arduino UNO, it is recommended to give input voltage up to 12 V.

As you can see in the preceding image, two motors can be controlled with one single chip, with normal voltage (9 V).

We will connect Arduino UNO as a controller and will connect inputs at pins 3 and 4. We will connect one motor at pins 3 and 4, and the other motor at pins 7 and 8. As shown in the following image, we can make a simple robot by connecting a caster as a third wheel after fitting those motors to the chassis:

We will make this robot do a simple repetitive task using this L293D and Arduino. Connect the Arduino control signal/output to the motors to pins 2, 7, 9, and 15 of the L293D.

Connect the motor between pins 3 and 6 of L293D. Once you have made the same connection for the other side of L293D, that is, pins 9 to 15, your circuit will look like the following image:



After checking all the connections once again, upload the following code to Arduino:

```
int leftMotorPos = 10;
int leftMotorNeg = 9;
int rightMotorPos = 12;
int rightMotorNeg = 13;

void setup()
{
  pinMode(leftMotorPos, OUTPUT);
  pinMode(rightMotorPos, OUTPUT);
  pinMode(leftMotorNeg, OUTPUT);
  pinMode(rightMotorNeg, OUTPUT);
}

void loop()
{
  forward();
  delay(5000);
  right();
  delay(5000);
  left();
  delay(5000);
  reverse();
  delay(5000);
  stopAll();
  delay(5000);
}

void forward() {
  digitalWrite(rightMotorPos, HIGH);
  digitalWrite(leftMotorPos, HIGH);
  digitalWrite(rightMotorNeg, LOW);
  digitalWrite(leftMotorNeg, LOW);
}

void left() {
  digitalWrite(rightMotorPos, HIGH);
  digitalWrite(leftMotorPos, LOW);
  digitalWrite(rightMotorNeg, LOW);
  digitalWrite(leftMotorNeg, LOW);
}

void right() {
  digitalWrite(rightMotorPos, LOW);
  digitalWrite(leftMotorPos, HIGH);
  digitalWrite(rightMotorNeg, LOW);
  digitalWrite(leftMotorNeg, LOW);
}

void reverse() {
  digitalWrite(rightMotorPos, LOW);
  digitalWrite(leftMotorPos, LOW);
  digitalWrite(rightMotorNeg, HIGH);
  digitalWrite(leftMotorNeg, HIGH);
}

void stopAll() {
  digitalWrite(rightMotorNeg, LOW);
  digitalWrite(leftMotorNeg, LOW);
  digitalWrite(rightMotorPos, LOW);
  digitalWrite(leftMotorPos, LOW);
}
```

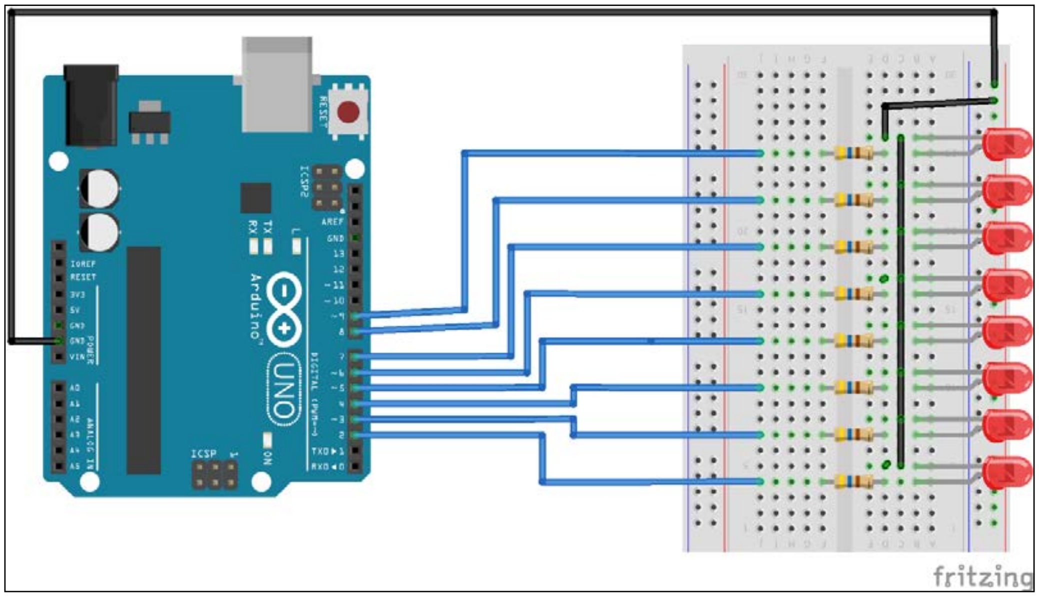
As per the preceding circuit and code, we are not giving input directly to the motors; rather, we are giving inputs to the L293D, which in turn will provide sufficient power to run the motors.

> Make sure to connect the grounds between Arduino and L293D. Otherwise, the grounds will be in floating mode and the motor will run abruptly, that is, the motor might run sometimes and might not run sometimes.

Another way to control the motor using L293 is to give a signal to an enable pin. By controlling enable pins' input, you can control the motors. Here, we are giving control to the motor input and the enable pins are given high input continuously.

Synchronizing an LED array with a motor

In the previous sections of this chapter, we learned about controlling an LED array and DC motor using Arduino:



Once you have connected the circuit as shown in the preceding image, upload the following code to Arduino. In the following code, we are writing the "Hello world" of persistence of vision:

```
int LEDPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
int noOfLEDs = 8;

//data corresponding to the each alphabet and a few characters to be
displayed
byte H[] = {B11111111, B11111111, B00011000, B00011000, B00011000,
B00011000, B11111111, B11111111};
byte E[] = {B00000000, B11111111, B11011011, B11011011, B11011011,
B11011011, B11000011, B11000011};
byte L[] = {B00000000, B11111111, B11111111, B00000011, B00000011,
B00000011, B00000011, B00000011};
byte O[] = {B00000000, B11111111, B11111111, B11000011, B11000011,
B11000011, B11111111, B11111111};
byte fullstop[] = {B00000000, B00000000, B00000000, B00000011,
B00000011, B00000000, B00000000, B00000000};
byte comma[] = {B00000000, B00000000, B00000000, B00000110, B00000101,
B00000000, B00000000, B00000000};

// Customize parameters based on the need
int timeBetweenColumn = 2.2;
int timeBtwnFrame = 20;
int frame_len = 8;

void setup()
{
  int i;
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(10, INPUT);
  for (i = 0; i < noOfLEDs; i++) {
    pinMode(LEDPins[i], OUTPUT);     // set each pin as an output
  }
}

void loop()
{
  int b = 0;
  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);
  digitalWrite(11, HIGH);
  delay(timeBtwnFrame);
  show(H);
  delay(timeBtwnFrame);
  show(E);
  delay(timeBtwnFrame);
  show(L);
  delay(timeBtwnFrame);
  show(L);
  delay(timeBtwnFrame);
  show(O);
  delay(timeBtwnFrame);
}

void show( byte* image )
{
  int a, b, c;

  // go through all data for all columns in each frame.
  for (b = 0; b < frame_len; b++)
  {
    for (c = 0; c < noOfLEDs; c++)
    {
      digitalWrite(LEDPins[c], bitRead(image[b], c));
    }
```

```
        delay(timeBetweenColumn);
      }
      for (c = 0; c < noOfLEDs; c++)
      {
        digitalWrite(LEDPins[c], LOW);
      }
    }
```

Initially, we are setting pins which need to be turned on, while writing any letters. Here, we have written code for letters H, E, L, O, period, and comma.

Here, we are using eight LEDs for displaying PoV. Make sure to connect a resistor between the LED and Arduino. As we have to make the LEDs glow based on the speed of the motor, we can control that by changing the value of `timeBetweenColumn` and `timeBtwnFrame` variables in the code.

By changing the values of these two variables, you should be able to sync the LEDs with the motor speed. One more thing that you can do is initialize the variable at a fixed value, and by using serial communication, change the value of these variables. Using the SoftwareSerial library, you can easily accomplish this.

Bringing your efforts to life

Once you know how to control motors and LEDs using Arduino, the final step is to put everything that you have learned so far and make it a standalone product. There are multiple ways you can achieve this:

- The simplest method is to use your hands.
- Using two different Arduinos or using external motors
- Using existing real-life devices

Using your hands for rotation

Even though you learned about controlling motors and LEDs, if you are doing this for the first time, you will take some time to understand the synchronization of LEDs and motors. The easiest way to test Persistence of Vision is to use your hands for rotation. Before you start rotating the complete setup, make sure you have uploaded the latest sketch to Arduino, connected some standalone battery/power source, and firmly fixed your LEDs on some base. As a base, you can use a thermocol sheet or cardboard, and fix them using some electrical tape.

Once you have fixed all the materials, you can rotate your PoV, and you can see your efforts coming to life, as shown in the following image:

Using two different Arduinos or external motors

Once you have fixed your Arduino, LEDs, and external battery/power source, you can use one more Arduino and connect motors for rotating the complete structure. At first it seems you can control the motors and LEDs using the same Arduino, however, if you think a little bit more, you will understand why you should not do this. The reason is because of the rotation of the LED structure and the connecting wire between motor and Arduino. If you have an external motor, you can use that and connect the Arduino-LED structure to it.

Use existing real-life devices

Another interesting idea is to use existing real-life devices that you have at your home. Of course, there are many devices which you can use; we have tested it with two devices: a bicycle wheel and a table fan, as shown in the following image. There is no difference in setup compared to the other two methods. In fact, you can use the same setup/structure as in method 1:

6
Troubleshooting and Advanced Resources

In this book, you got introduced to Arduino Pi and its capability, you developed your LED night lamp, remote controlled TV backlight, LED cube, sound visualization, and finally, persistence of vision. There might have been instances when you wanted to know more about certain topics or you were stuck in between. This chapter answers all those questions. In the first section, common troubleshooting techniques are mentioned. The second and last part of the chapter has resources that will be useful if you want to do advanced stuff with Arduino:

- Troubleshooting
- Resources – advanced users

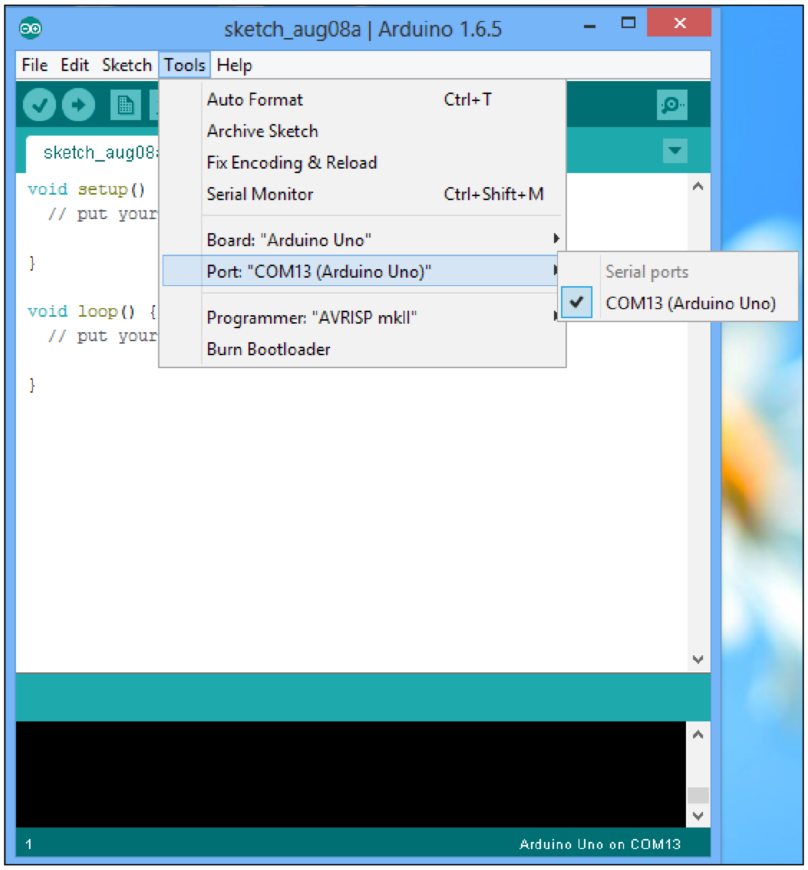
Troubleshooting

This section has answers to some of the common problems that you might face while working with Arduino.

Can't upload program

Assign the correct serial port: in the Arduino Environment program, go to **Tools | Serial Port**, and select the correct serial port. To see what serial port the board is using, connect the board to your computer with the USB cable. From the Windows desktop, right-click on My Computer, then **Properties | Device Manager | Ports (COM & LPT)**. There will be an entry like **USB Serial Port (COM13)** or **Arduino UNO (COM13)**.

This means serial communication port 13 is the one in use:



```
void setup() {
  // put your
}

void loop() {
  // put your
}
```

[On upload, you may get error messages like, "Serial port 'COM13' already in use". Try quitting any programs that may be using it.]

One of the possible reasons could be that you are running multiple Arduino IDEs on a single machine. Try closing all Arduino IDEs and open the sketch that you want to upload to your Arduino board. Most of the time it will solve the issue. Even after re-opening the Arduino IDE, if you get the same error message, unplug your Arduino and plug it back in. This should solve your issue. In some cases, if you get the same error message again, restart your computer.

LED is dim

This is the most common mistake that beginners make. Sometimes, an LED connected to an Arduino pin is dim. This is because the Arduino pin connected to the LED is not declared as OUTPUT and is not getting the full power from the Arduino board. If you declare the Arduino LED pin as OUTPUT, it will solve the issue and the LED will glow properly.

You can also refer to the troubleshooting section on the Arduino website. You can find solutions to other problems at `https://www.arduino.cc/en/Guide/Troubleshooting`.

Resources – advanced users

This section contains some advanced projects that I think are interesting and fun to build. This last section has some handy and useful resources to take your Arduino journey to the next level.

Projects

Based on all the basic skills of Arduino, LED, and sensor programming that you have, we believe that the following are four projects that might be interesting for you to build.

Twitter Mood Light

This is one of the cool projects that I have seen. It is a way to get a glimpse of the collective human consciousness. It is a way to be alerted with the world's events as they unfold, or when something big happens. Arduino connects directly to any wireless network via the WiFly module. It then searches Twitter for tweets with emotional content and collates the tweets for each emotion. It also does some math, such that the color of the LED fades to reflect the current world mood. Here are a few examples:

- Red for anger
- Yellow for happiness
- Pink for love
- White for fear
- Green for envy
- Orange for surprise
- Blue for sadness

In this project, after getting tweets from the twitter handle for the user, by using the sentiment extraction method, you can get to know about the emotion/mood of the world.

Read more at: `http://www.instructables.com/id/Twitter-Mood-Light-The-Worlds-Mood-in-a-Box/`.

Secret knock detecting door-lock

You can now keep your secret hideout hidden from intruders with a lock that will only open when it hears the secret knock. This wasn't accepted completely at the beginning, but turned out to be surprisingly accurate at judging knocks. If the precision is turned all the way up it can even tell people apart, even if they give the same knock!

Read more at `http://www.instructables.com/id/Secret-Knock-Detecting-Door-Lock/`.

LED biking jacket

This is one of the best projects to show your making skills to your friend and the outside world. This project shows you how to build a jacket with turn signals that will let people know where you're headed when you're on your bike. It uses conductive thread and sewable electronics, so your jacket will be soft, wearable, and washable when you're done.

In this project, you will learn to use another type of Arduino which is designed specifically to be wearable—LilyPad.

Read more at `http://www.instructables.com/id/turn-signal-biking-jacket/`.

Twitter-enabled coffee pot

Tweet-a-pot is the next wave in fancy twitter-enabled devices. This coffee pot enables you to make a pot of coffee from anywhere that has a cell phone reception, using Twitter and an Arduino board. The tweet-a-pot is the easy implementation for remote device control; using a bit of code and some hardware, you can have your very own Twitter-enabled coffee pot.

Read more at: `http://www.instructables.com/id/Tweet-a-Pot-Twitter-Enabled-Coffee-Pot/`.

Useful resources

In this book, you have learned about Arduino, LEDs, and sensors. If you look at the world of the maker movement and Arduino, we have only scratched the surface, and there are so many things that you need to learn. Here are some resources that we think are useful in taking your skills to the next level.

Hackaday

This is an excellent resource for all sorts of technological wonders. It has lots of Arduino-related projects and easy to follow guides for most of the projects. However, this website is not limited to just Arduino; it has various other resources for almost all DIY technologies. It contains an excellent collection of posts and information to fuel the imagination.

Refer to `http://hackaday.com/`.

The Arduino blog

This is a great resource for all Arduino-related news. It features all the latest Arduino-related hardware, as well as software projects. It is also one of the best places to keep yourself updated with the work that the Arduino team has been doing.

Refer to `https://blog.arduino.cc/`.

The Make magazine

This is a hobbyist magazine that celebrates all kinds of technology. Its blog covers all kinds of interesting do-it-yourself (DIY) technology and projects for inspiration. You can find useful Arduino resources/projects under the "Arduino" section of the website.

Refer to `http://blog.makezine.com/`.

Bildr

Bildr is an excellent resource that provides in-depth, community-published tutorials. As well as providing clear tutorials, Bildr also has excellent illustrations, making the connections easy to follow. Many of the tutorials are Arduino-based and provide all the code and information on the components that you will need.

Refer to `http://bildr.org/`.

Instructables

This is a web-based documentation platform that allows people to share their projects with step-by-step instructions on how to make them. Instructables isn't just about Arduino or even technology, so you can find a whole world of interesting material there.

Refer to `http://www.instructables.com/`.

Tronixstuff

John Boxall's website is a great resource for learning about Arduino. He has dozens of different Arduino projects and demos on his site.

Refer to `http://tronixstuff.com/tutorials/`.

Adafruit

Adafruit is an online shop, repository, and forum for all kinds of kits to help you make your projects work. It is probably one of the best online resources for learning about Arduino and checking out some cool projects.

Refer to `https://learn.adafruit.com/`.

All About Circuits

If you are interested in learning more about electronics and circuit design, this might be the best place for you to learn.

Refer to `http://www.allaboutcircuits.com/`.

Hackerspaces

Hackerspaces are physical spaces where artists, designers, makers, hackers, coders, engineers, or anyone else, can meet to learn, socialize, and collaborate on projects. If you are looking for inspiration and want to meet some awesome people doing some amazing work, find a hackerspace nearby your area and learn from the masters.

Refer to `http://hackerspaces.org/`.

The Arduino forum

This is a great place to get answers to specific Arduino questions. You often find that other people are working through the same problems that you are, so with some thorough searching, you're likely to find the answer to almost any problem.

Refer to `http://arduino.cc/forum/`.

Summary

This chapter provided solutions for some of the common problems that you might face while working with Arduino. The last section of the chapter mentioned a few DIY projects that you might want to pursue with the resources provided.

If you face any issue in any of the projects mentioned in the book, or you notice any typos/errors in any of the chapters, feel free to mail us at `Samarth@outlook.com` and/or `Utsav_shah01@outlook.com` with the subject as the title of the book.

Module 3

Arduino for Secret Agents

Transform your Arduino device into a secret agent gadget and build a range of espionage projects with this practical guide for hackers

1
A Simple Alarm System with Arduino

I want to start this book with a simple project that any secret agent will want to have, a simple alarm system that will be activated whenever motion is detected by a sensor. This simple system is not only fun to make but will also help us to go over the basics of Arduino programming and electronics, which are the skills that we will use in this whole book.

It will basically be a simple alarm (a buzzer that makes sound, plus a red LED) combined with a motion detector. The user will also be able to stop the alarm by pressing a button.

We are going to do the following in this chapter:

- First, we are going to see what the requirements for this project are, in terms of hardware and software
- Then, we will see how to assemble the hardware parts for this project
- After that, we will configure our system using the Arduino IDE

Hardware and software requirements

First, let's see what the required components for this project are. As this is the first chapter of the book, we will spend a bit more time here to detail the different components, as these are components that we will be using in the whole book.

The first component that will be central to the project is the Arduino Uno board:

In several chapters of this book, this will be the 'brain' of the projects that we will make. In all the projects, I will be using the official Arduino Uno R3 board. However, you can use an equivalent board from another brand or another Arduino board, such as an Arduino Mega board.

Another crucial component of our alarm system will be the buzzer:

This is a very simple component that is used to make simple sounds with Arduino. You couldn't play an MP3 with it but it's just fine for an alarm system. You can, of course, use any buzzer that is available; the goal is to just make a sound.

After that, we are going to need a motion detector:

Here, I used a very simple PIR motion detector. This sensor will measure the infrared (IR) light that is emitted by moving objects in its field of view, for example, people moving around. It is really easy and quite cheap to interface with Arduino. You can use any brand that you want for this sensor; it just needs a voltage level of 5V in order to be compatible with the Arduino Uno board.

Finally, here is the list of all the components that we will use in this project:

- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- Buzzer (`https://www.sparkfun.com/products/7950`)
- PIR (`https://www.sparkfun.com/products/13285`)
- LED (`https://www.sparkfun.com/products/9590`)
- 330 Ohm resistor (`https://www.sparkfun.com/products/8377`)
- Button (`https://www.sparkfun.com/products/97`)
- 1k Ohm resistor (`https://www.sparkfun.com/products/8980`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

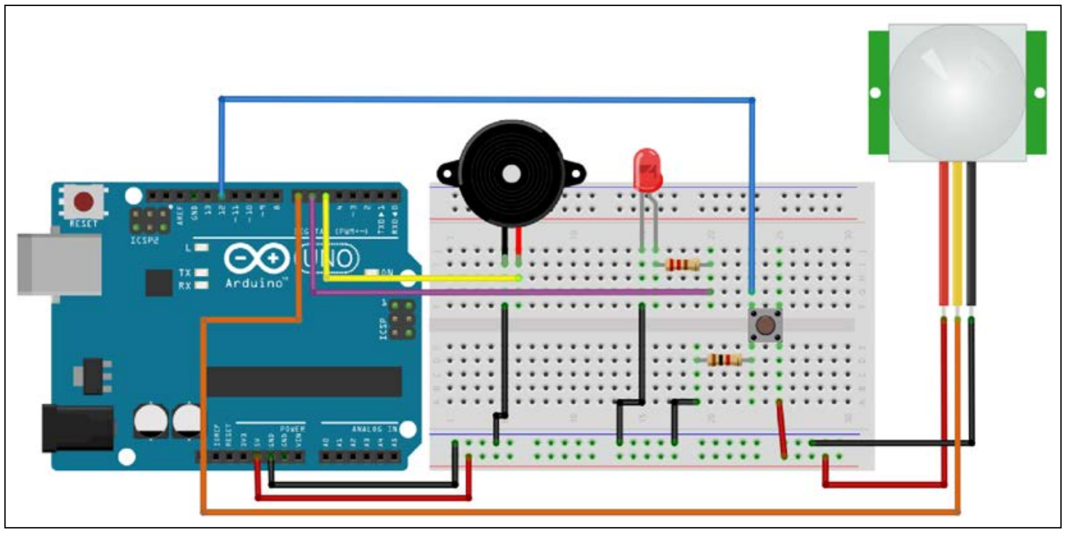
On the software side, the only thing that we will need in the first chapter is the latest version of the Arduino IDE that you can download from the following URL: `https://www.arduino.cc/en/main/software`.

Note that we are going to use the Arduino IDE in all the projects of this book, so make sure to install the latest version.

Hardware configuration

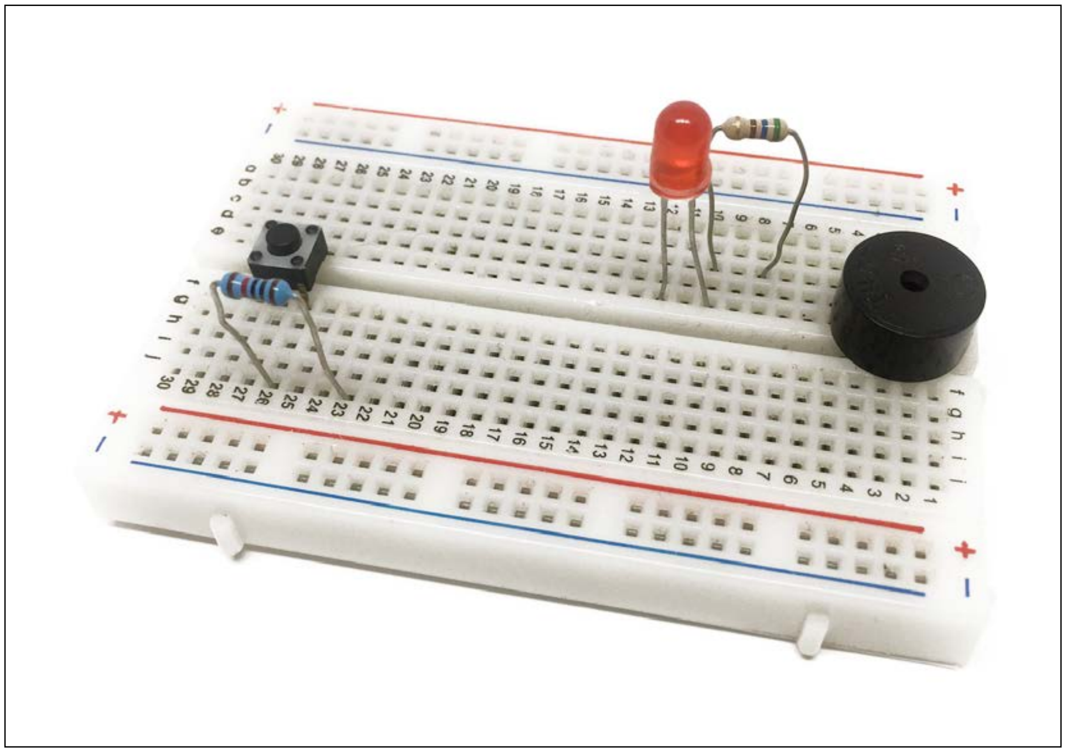
We are now going to assemble the hardware for this project. As this is the first project of this book, it will be quite simple. However, there are quite a lot of components, so be sure to follow all the steps.

Here is a schematic to help you out during the process:



Let's start by putting all the components on the board. Place the buzzer, button, and LED on the board first, according to the schematics. Then, place the 330 Ohm resistor in series with the LED anode (the longest pin) and connect the 1k Ohm resistor to one pin of the push button.

This is how it should look at this stage:



Now we are going to connect each component to the Arduino board.

Let's start with the power supply. Connect the 5V pin of the Arduino board to one red power rail of the breadboard, and the GND pin of the Arduino board to one blue power rail of the breadboard.

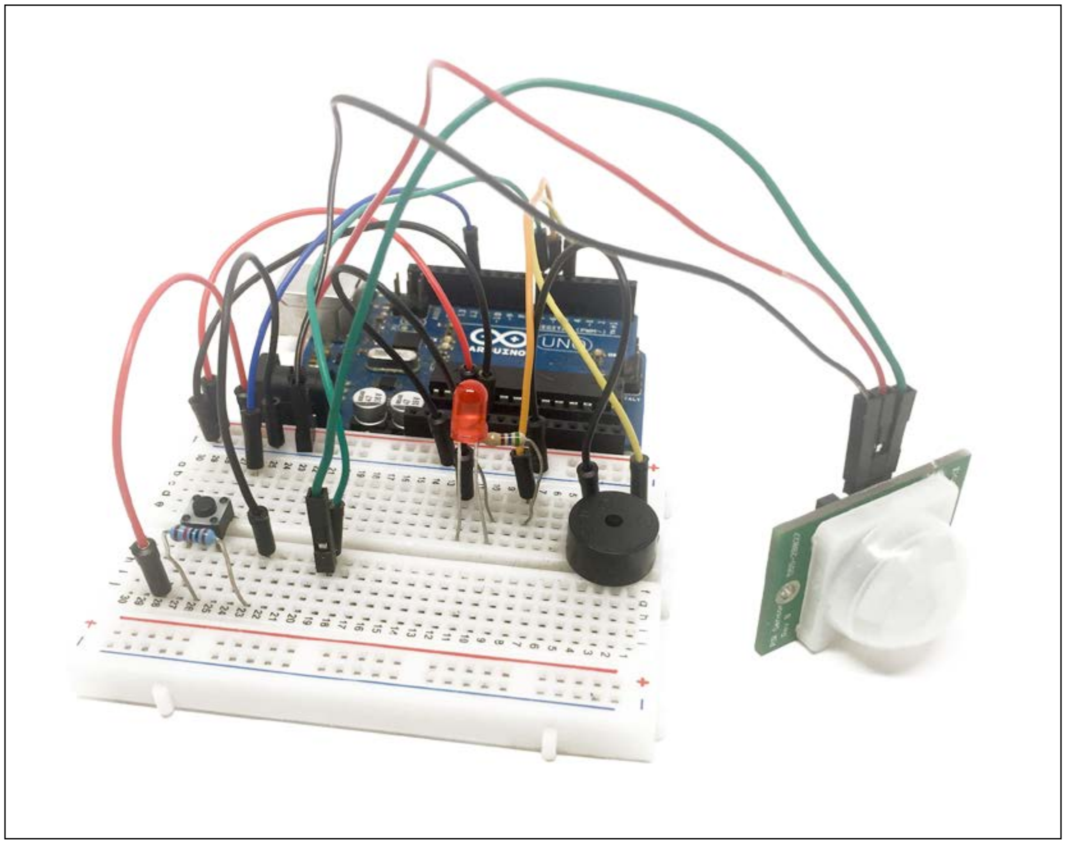
Then, we are going to connect the buzzer. Connect one pin of the buzzer to pin number 5 of the Arduino board and the other pin to the blue power rail of the breadboard.

After that, let's connect the LED. Connect the free pin of the resistor to pin number 6 of the Arduino board and the free pin of the LED (the cathode) to the ground via the blue power rail.

Let's also connect the push button to our Arduino board. Refer to the schematic to be sure about the connections since it is a bit more complex. Basically, you need to connect the free pin of the resistor to the ground and connect the pin that is connected to the button to the 5V pin via the red power rail. Finally, connect the other side of the button to pin 12 of the Arduino board.

Finally, let's connect the PIR motion sensor to the Arduino board. Connect the VCC pin of the motion sensor to the red power rail and the GND pin to the blue power rail. Finally, connect the SIG pin (or OUT pin) to Arduino pin number 7.

The following is the final result:



If your project looks similar to this picture, congratulations, you just assembled your first secret agent project! You can now go on to the next section.

Configuring the alarm system

Now that the hardware for our project is ready, we can write down the code for the project so that we have a usable alarm system. The goal is to make the buzzer produce a sound whenever motion is detected and also to make the LED flash. However, whenever the button is pressed, the alarm will be switched off.

Here is the complete code for this project:

```
// Code for the simple alarm system

// Pins
const int alarm_pin = 5;
const int led_pin = 6;
const int motion_pin = 7;
const int button_pin = 12;

// Alarm
boolean alarm_mode = false;

// Variables for the flashing LED
int ledState = LOW;
long previousMillis = 0;
long interval = 100;  // Interval at which to blink (milliseconds)

void setup()
{
  // Set pins to output
  pinMode(led_pin,OUTPUT);
  pinMode(alarm_pin,OUTPUT);

  // Set button pin to input
  pinMode(button_pin, INPUT);

  // Wait before starting the alarm
  delay(5000);
}

void loop()
{
  // Motion detected ?
  if (digitalRead(motion_pin)) {
    alarm_mode = true;
  }

  // If alarm mode is on, flash the LED and make the alarm ring
  if (alarm_mode){
    unsigned long currentMillis = millis();
    if(currentMillis - previousMillis > interval) {
      previousMillis = currentMillis;
      if (ledState == LOW)
```

```
          ledState = HIGH;
        else
          ledState = LOW;
      // Switch the LED
      digitalWrite(led_pin, ledState);
      }
      tone(alarm_pin,1000);
    }

    // If alarm is off
    if (alarm_mode == false) {

      // No tone & LED off
      noTone(alarm_pin);
      digitalWrite(led_pin, LOW);
    }

    // If button is pressed, set alarm off
    int button_state = digitalRead(button_pin);
    if (button_state) {alarm_mode = false;}
  }
```

Downloading the example code
You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

We are now going to see, in more detail, the different parts of the code. It starts by declaring which pins are connected to different elements of the project, such as the alarm buzzer:

```
const int alarm_pin = 5;
const int led_pin = 6;
const int motion_pin = 7;
const int button_pin = 12;
```

After that, in the `setup()` function of the sketch, we declare these pins as either inputs or outputs, as follows:

```
// Set pins to output
pinMode(led_pin,OUTPUT);
pinMode(alarm_pin,OUTPUT);

// Set button pin to input
pinMode(button_pin, INPUT);
```

Then, in the `loop()` function of the sketch, we check whether the alarm was switched on by checking the state of the motion sensor:

```
if (digitalRead(motion_pin)) {
  alarm_mode = true;
}
```

Note that if we detect some motion, we immediately set the `alarm_mode` variable to true. We will see how the code makes use of this variable right now.

Now, if the `alarm_mode` variable is true, we have to enable the alarm, make the buzzer emit a sound, and also flash the LED. This is done by the following code snippet:

```
if (alarm_mode){
    unsigned long currentMillis = millis();
    if(currentMillis - previousMillis > interval) {
      previousMillis = currentMillis;
      if (ledState == LOW)
        ledState = HIGH;
      else
        ledState = LOW;
    // Switch the LED
    digitalWrite(led_pin, ledState);
    }
    tone(alarm_pin,1000);
  }
```

Also, if `alarm_mode` is returning false, we need to deactivate the alarm immediately by stopping the sound from being emitted and shutting down the LED. This is done with the following code:

```
if (alarm_mode == false) {

    // No tone & LED off
    noTone(alarm_pin);
    digitalWrite(led_pin, LOW);
  }
```

Finally, we continuously read the state of the push button. If the button is pressed, we will immediately set the alarm off:

```
int button_state = digitalRead(button_pin);
if (button_state) {alarm_mode = false;}
```

Usually, we should take care of the bounce effect of the button in order to make sure that we don't have erratic readings when the button is pressed. However, here we only care about the button actually being pressed so we do not need to add an additional debouncing code for the button.

Note that you can find all the code for this project inside the GitHub repository of the book:

`https://github.com/marcoschwartz/arduino-secret-agents`

Now that we have written down the code for the project, it's time to get to the most exciting part of the chapter: testing the alarm system!

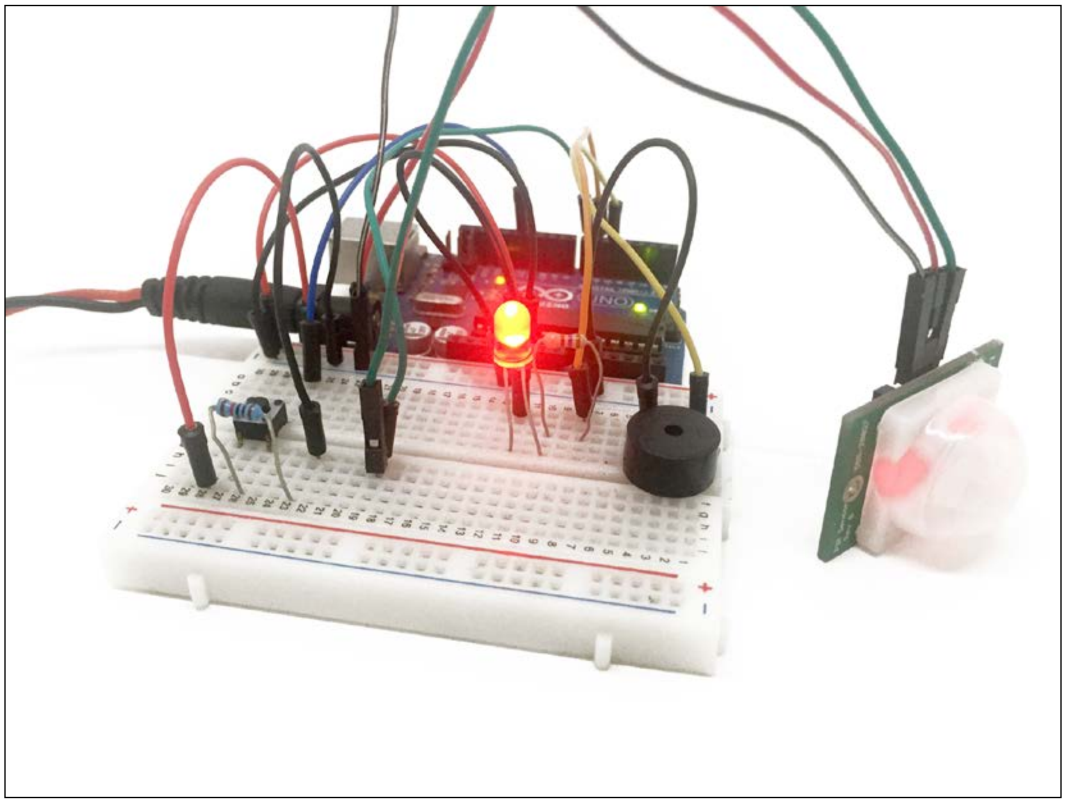
Testing the alarm system

We are now ready to test our simple alarm system. Just grab the code for this project (either from the preceding code or the GitHub repository of the book) and put it into your Arduino IDE.

In the IDE, choose the right board type (for example, Arduino Uno) and also the correct serial port.

You can now upload the code to the board. Once it is done, simply pass your hand in front of the PIR motion sensor; the alarm should go off immediately. Then, simply press the push button to stop it.

To illustrate the behavior of the alarm, I simply used a battery pack to make it work when it is not connected to my computer. The following is the result when the alarm goes off:



If this works as expected, congratulations, you just built your first secret agent project: a simple alarm system based on Arduino!

If it doesn't work well at this point, there are several things you can check. First, go through the hardware configuration part again to make sure that your project is correctly configured.

Also, you can verify that when you pass your hand in front of the PIR sensor, it goes red. If this is not the case, most probably your PIR motion sensor has a problem and must be replaced.

Summary

In this first chapter, we built a simple alarm based on Arduino with only a few components.

There are several ways to go further and improve this project. You can add more functions to the project just by adding more lines to the code. For example, you can add a timer so that the alarm only goes off after a given amount of time, or you can build a mode where a push of the button actually activates or deactivates the alarm mode.

In the next chapter, we are going to build another project that is very useful for secret agents: an audio recording device based on Arduino!

2
Creating a Spy Microphone

In this chapter, we are going to build a very useful device for any secret agent: a spy microphone. The project will be based on Arduino, with a simple amplified microphone and an SD card.

The following are the steps that we are going to take to build this project:

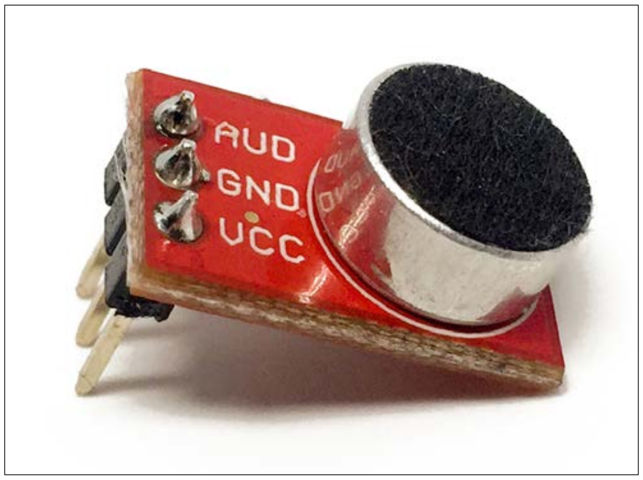
- We will see how to configure the project in order to make sure that it is recording for a given amount of time that can be configured by the user
- Then, the recorded audio file will be written on the SD card and be accessible from any computer
- Before doing that, we will test all the components of the project individually

Let's dive in!

Hardware and software requirements

Let's first see what the required components for this project are. As usual, we will use an Arduino Uno board as the 'brain' of the project.

Then, we will need a microphone. I used a simple SparkFun electret microphone, which has an amplifier onboard, as shown in the following image:



The most important thing here is that the microphone is amplified. For example, SparkFun is amplified 100 times, making it possible for the Arduino Uno to record usual sound levels (such as voices).

Then, you will need a microSD card with an adapter:

You will also need a way to record data on the SD card. There are many ways to do so with Arduino. The easiest, which is the solution that I chose here, is to use a shield. I had an Ethernet Shield available, which is great because it also has an onboard microSD card reader.

You can, of course, use any shield with a microSD card reader or even a microSD reader breakout board.

You will also need a breadboard and some jumper wires to make the required connections.

Finally, the following is the list of all the components that we will use in this project:

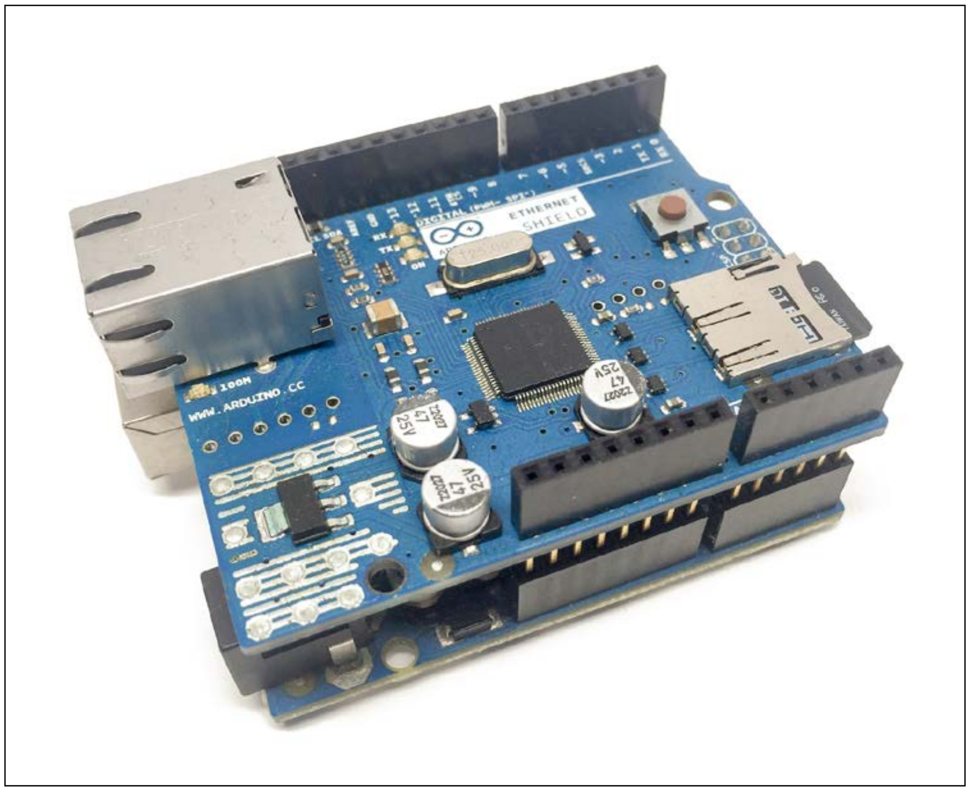
- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- Arduino Ethernet Shield (`https://www.sparkfun.com/products/11166`)
- Electret Microphone (`https://www.sparkfun.com/products/9964`)
- microSD card (`https://www.sparkfun.com/products/11609`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

On the software side, you will need a special version of the SD card library called `SdFat`. We can't use the usual Arduino SD library here as we will do some really fast write operations on the SD card, which can't be handled by the SD library that comes with the Arduino software. You can download this library from `https://github.com/greiman/SdFat`.

Using the SD card

The first thing that we are going to do in this project is to test whether we can actually access the SD card. This will ensure that we don't run into SD card-related problems later in the project.

This is a picture of the Ethernet Shield that I used, with the microSD card mounted on the right:



Let's now see the code that we will use to test the SD card's functionalities. The following is the complete code for this section:

```
// Include the SD library
#include <SPI.h>
#include <SD.h>

// Set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
```

```
const int chipSelect = 4;

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  Serial.print("\nInitializing SD card...");

  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your
shield or module?");
    return;
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.print("\nCard type: ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }

  // Now we will try to open the 'volume'/'partition' - it should be
FAT16 or FAT32
```

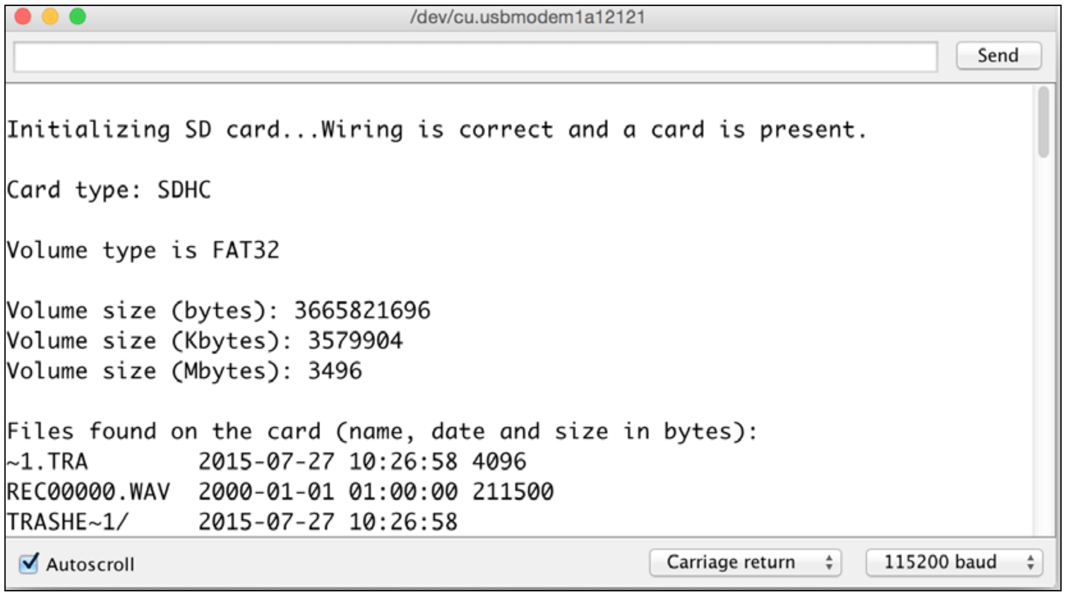
```
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure 
you've formatted the card");
    return;
  }


  // print the type and size of the first FAT-type volume
  uint32_t volumesize;
  Serial.print("\nVolume type is FAT");
  Serial.println(volume.fatType(), DEC);
  Serial.println();

  volumesize = volume.blocksPerCluster();    // clusters are 
collections of blocks
  volumesize *= volume.clusterCount();       // we'll have a lot of 
clusters
  volumesize *= 512;                            // SD card blocks are 
always 512 bytes
  Serial.print("Volume size (bytes): ");
  Serial.println(volumesize);
  Serial.print("Volume size (Kbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Mbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);


  Serial.println("\nFiles found on the card (name, date and size in 
bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);
}


void loop(void) {

}
```

This code tests a lot of things on the SD card, such as the file format and available space, and also lists all the files present on the SD card. However, what we are really interested in is to know whether the SD card can be read by the Arduino board. This is done with the following code snippet:

```
if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your
shield or module?");
    return;
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }
```

Now, let's test the code. You can simply copy this code and paste it into the Arduino IDE.

Then, upload it to the Arduino Uno board and open the serial monitor. Make sure that the serial speed is set to `115200` bps. This is what you will see:



```
Initializing SD card...Wiring is correct and a card is present.

Card type: SDHC

Volume type is FAT32

Volume size (bytes): 3665821696
Volume size (Kbytes): 3579904
Volume size (Mbytes): 3496

Files found on the card (name, date and size in bytes):
~1.TRA        2015-07-27 10:26:58 4096
REC00000.WAV  2000-01-01 01:00:00 211500
TRASHE~1/     2015-07-27 10:26:58
```

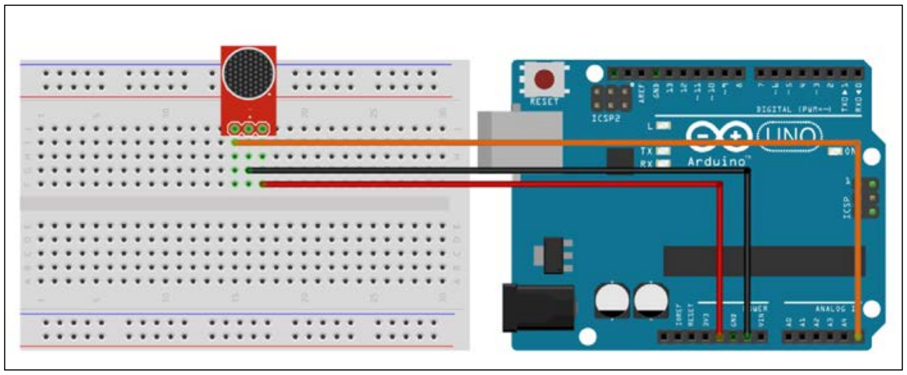
If you can see this, congratulations, your SD card and card reader are correctly configured and ready to host some spy audio recordings!

Testing the microphone

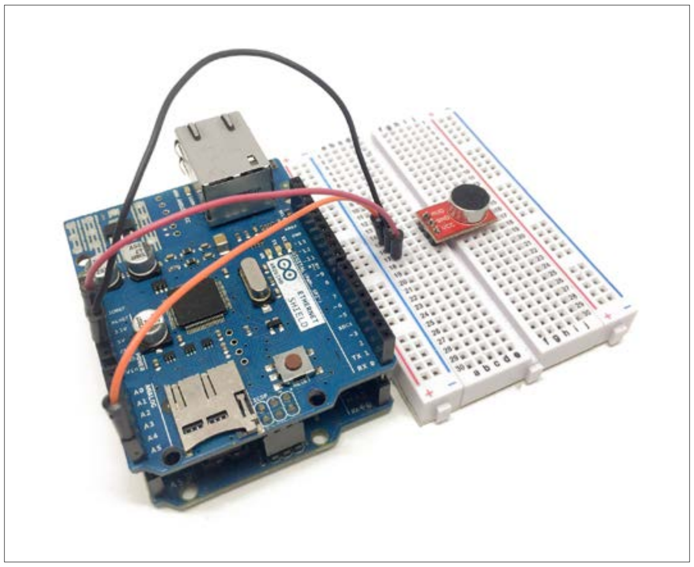
We are now going to make sure that the microphone is working correctly and especially check whether it can record voice levels, for example. I had a problem when I was testing the prototype of this project with a microphone that wasn't amplified; I just couldn't hear anything on the recording.

The first step is to plug the microphone into the Arduino board. There are 3 pins to connect the microphone: VCC, GND, and AUD. Connect VCC to the Arduino 5V pin, GND to the Arduino GND pin, and AUD to the Arduino analog pin A5.

The following is a schematic to help you out:



Here is an image of the final result:



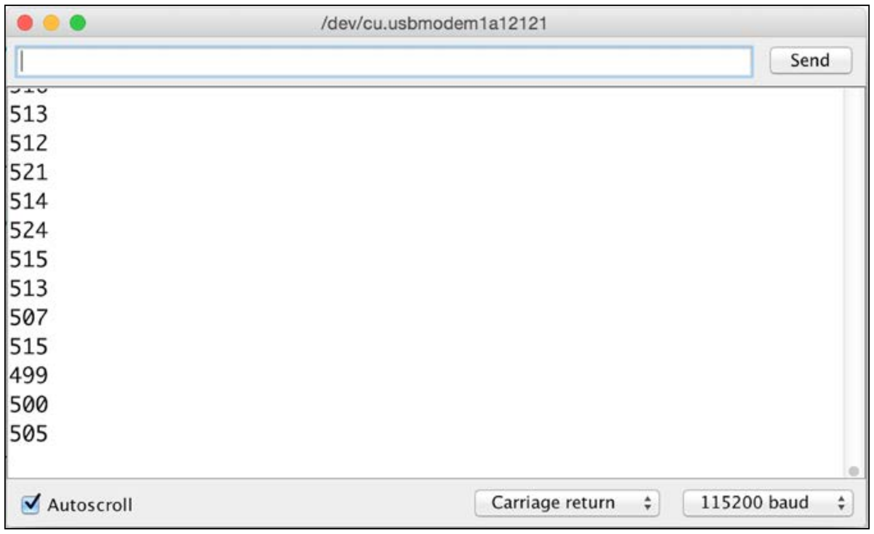
We are now going to use a very simple sketch to read out the signal from the microphone and print it on the serial monitor:

```
// Microphone test

void setup() {

  // Start Serial
  Serial.begin(115200);
}

void loop() {

  // Read the input on analog pin 5:
  int sensorValue = analogRead(A5);

  // Print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

This sketch, basically, continuously reads the data from the A5 pin, where the microphone is connected, and prints it on the serial monitor.

Now, copy and paste this sketch in the Arduino IDE and upload it to the board. Also, open the serial monitor.

The following is the result on the serial monitor:



```
513
512
521
514
524
515
513
507
515
499
500
505
```

While looking at the serial monitor, speak around the microphone. You should immediately see some variations in the signal that is read by the board. This means that your voice is being recorded by the microphone and the amplification is sufficient for the microphone to record a normal voice level.

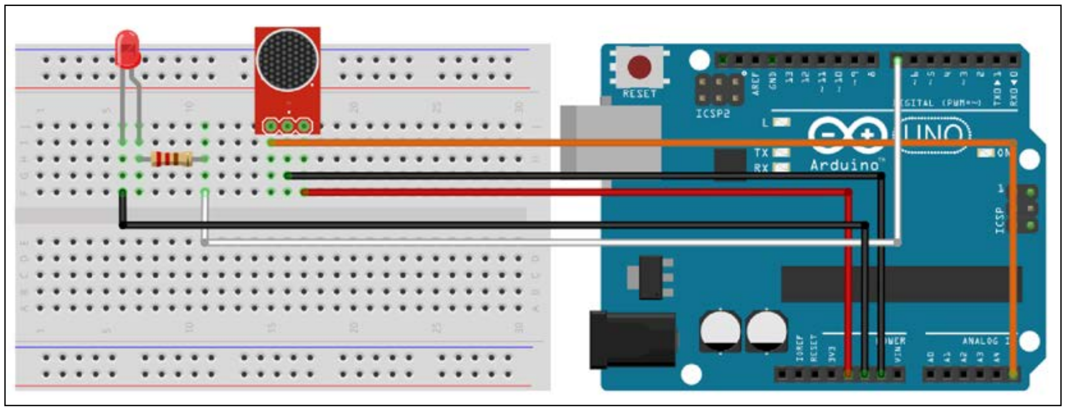
Building the spy microphone

In this section, we are going to put everything together and actually build our spy microphone.

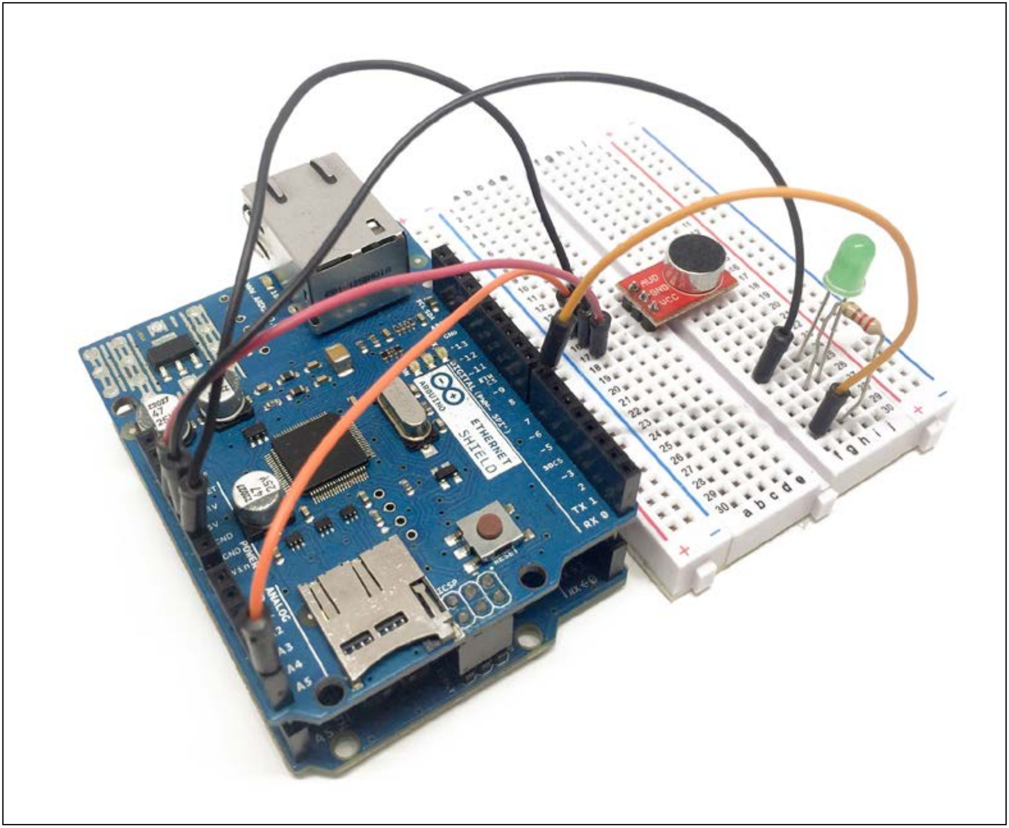
The hardware for the project is nearly ready if you followed the previous section. You just need to plug the SD card into the reader again.

I also added an LED on pin 7, just to know when the recording is on. If you want to do the same, you just need an LED and a 330 Ohm resistor. Of course, remove this LED when you actually want to use it as a spy microphone or your project might get noticed.

The schematic to help you out is as follows:



The following is the image of the completely assembled project:



We are now going to see the details of the code for the project. Basically, we want the device to record audio from the microphone for a given amount of time and then stop the recording.

As the code is long and complex, we are only going to see the most important parts here.

The first step is to include the `SdFat` library:

```
#include <SdFat.h>
```

Then, we will create the instances that are necessary in order to write on the SD card:

```
SdFat sd;
SdFile rec;
```

After that, we will define on which pin we want the optional recording LED to be:

```
const int ledStart = 7;
```

We will also define a variable for the internal counter of the project:

```
byte recordingEnded = false;
unsigned int counter;
unsigned int initial_count;
unsigned int maxCount = 10 * 1000; // 10 Seconds
```

Note that here you will have to modify the `maxCount` variable according to the time for which you want the device to record. Here, I just used 10 seconds by default as a test.

Then, we will initialize the ADC (Analog-Digital Converter):

```
Setup_timer2();
Setup_ADC();
```

After that, we will initialize the SD card and also flash the LED if this is successful:

```
if (sd.begin(chipSelect, SPI_FULL_SPEED)) {
  for (int dloop = 0; dloop < 4; dloop++) {
    digitalWrite(ledStart,!digitalRead(ledStart));
    delay(100);
  }
}
```

Then, we will actually start the recording with the following function:

```
StartRec();
```

We will also initialize the counter with the following line of code:

```
initial_count = millis();
```

Then, in the `loop()` function of the sketch, we will update the counter, which helps to keep track of the elapsed time:

```
counter = millis() - initial_count;
```

Still in the `loop()` function, we will actually stop the recording if we reach the maximum amount of time that we defined previously:

```
if (counter > maxCount && !recordingEnded) {
  recordingEnded = true;
  StopRec();
}
```

Note that the whole code for this section can be found in the GitHub repository of the book at `https://github.com/marcoschwartz/arduino-secret-agents`.

Recording on the SD card

In the last section of the chapter, we are actually going to test the project and record some audio.

First, copy all the code and paste it into the Arduino IDE. Compile it and upload it to the Arduino board. Note that, as soon as you do that, the project will start recording the audio. If you have connected the optional LED on pin 7, the LED should also be on during the recording phase.

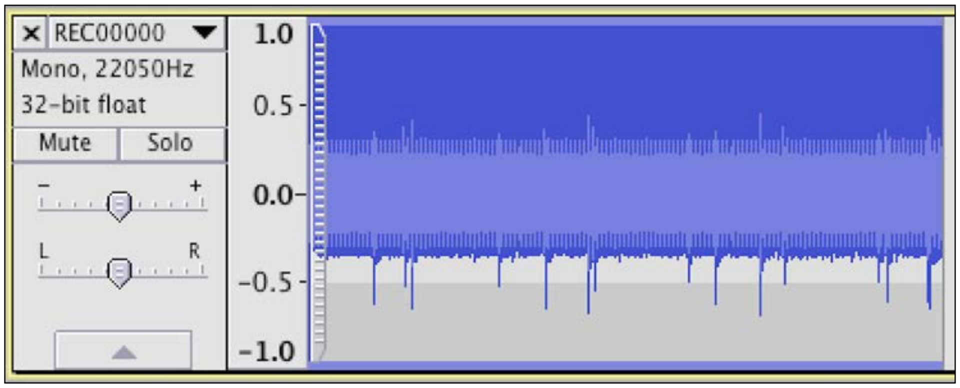
You can now talk a bit or play your favorite song just to make sure that actual audio is being recorded by the microphone.

Then, after the amount of time defined in the code, stop the project by disconnecting the power. Then, remove the SD card and insert it into your computer.

On your computer, navigate to the SD card and you will see that one file was recorded:

You can now simply open this file with your favorite audio player and listen to what was just recorded.

I, for example, opened it with the free audio editing software, Audacity, to see how the waveform looked like:



Congratulations, you just built your own spy microphone! You can now play with the different settings, for example, using a longer recording time.

Summary

In this chapter, you learned how to build a spy microphone based on a simple amplified microphone, Arduino, and a microSD card. The spy microphone can be set up to record audio continuously for a given amount of time.

Of course, there are many things that you can now do in order to improve this project. You can, for example, use a battery to power the project to make it autonomous. Then, you just need to place it in a room in which you want to record a conversation and just come back later to retrieve the SD card with the recording.

You can also connect a motion sensor to the project and use that to automatically start a recording when motion is detected in a room in order to intercept conversations with total discretion.

Another interesting project would be to use the audio levels that are sensed by the project to actually start or stop the recording. For example, you could rewrite the software of the project to continuously monitor the microphone and automatically start a recording when a given threshold is crossed, indicating that somebody is speaking. Then, the program could automatically stop the recording once the audio levels are low again.

In the next chapter of the book, we are going to make another useful project for any spy: an EMF detector to detect the presence of a recording device in a room, for example.

3
Building an EMF Bug Detector

In this chapter, we are going to build a very useful tool that every secret agent should have: a bug detector. We will build a simple device that will allow you to detect whether there are any bugs nearby, such as a recording device or a wireless hidden camera.

The following are the topics that we will cover in this chapter:

- We will build a project with a simple wire antenna and the project will display the EMF readings on LCD screen.
- We will also add a simple LED to indicate when EMF activity goes above a certain threshold.

Let's dive in!

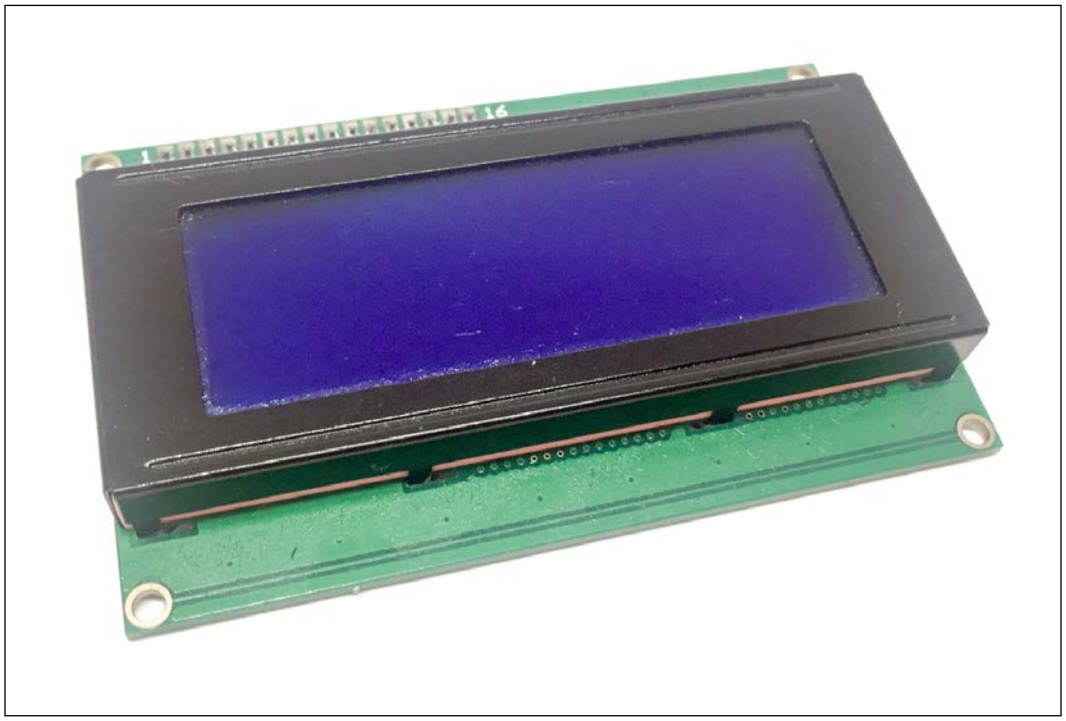
Hardware and Software requirements

First, let's see what the required components for this project are.

You will, of course, need the usual Arduino Uno, that will act as the brain of the project and process all the information.

You will also need a simple wire, preferably long (such as 10 cm to 20 cm) to act as an antenna. You will need a 1M Ohm resistor along with this wire.

To display the data, we will use a simple I2C LCD screen. I used a 4 x 20 I2C screen from DFRobot:



You can, of course, use the LCD screen of your choice for this project, you will just need to use the right LCD library.

I also integrated a simple red LED along with a 330 Ohm resistor to display when EMF goes above a given level.

Finally, you will also need a breadboard and jumper wires to make the different connections.

Here is the list of all the components that we will use for this project:

- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- I2C LCD screen (`http://www.robotshop.com/en/dfrobot-i2c-twi-lcd-module.html`)
- Red LED (`https://www.sparkfun.com/products/9590`)
- 330 Ohm resistor (`https://www.sparkfun.com/products/8377`)
- Long wire (at least 10 cm)

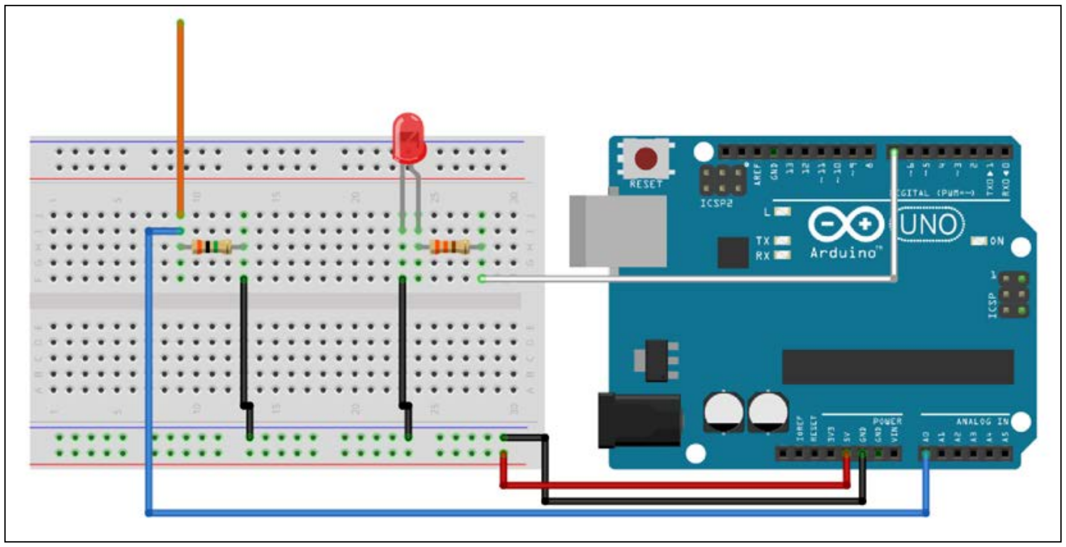
- 1M Ohm resistor (`https://www.sparkfun.com/products/11853`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

On the software side, you will need the library for the LCD screen. As we will use an I2C LCD screen for this project, I recommend the following library that you can download from `http://hmario.home.xs4all.nl/arduino/LiquidCrystal_I2C/`.

Once the library is correctly installed, you can move to the next step.

Hardware configuration

We are now going to configure the hardware part of the project. As we have a relatively small number of simple components, the configuration of this project will be really easy and straightforward. This is a schematic to help you out:



As you can see, the LCD screen is not present on this schematic. We'll first see how to connect all the other components and then see how to connect the LCD screen at the end of this section.

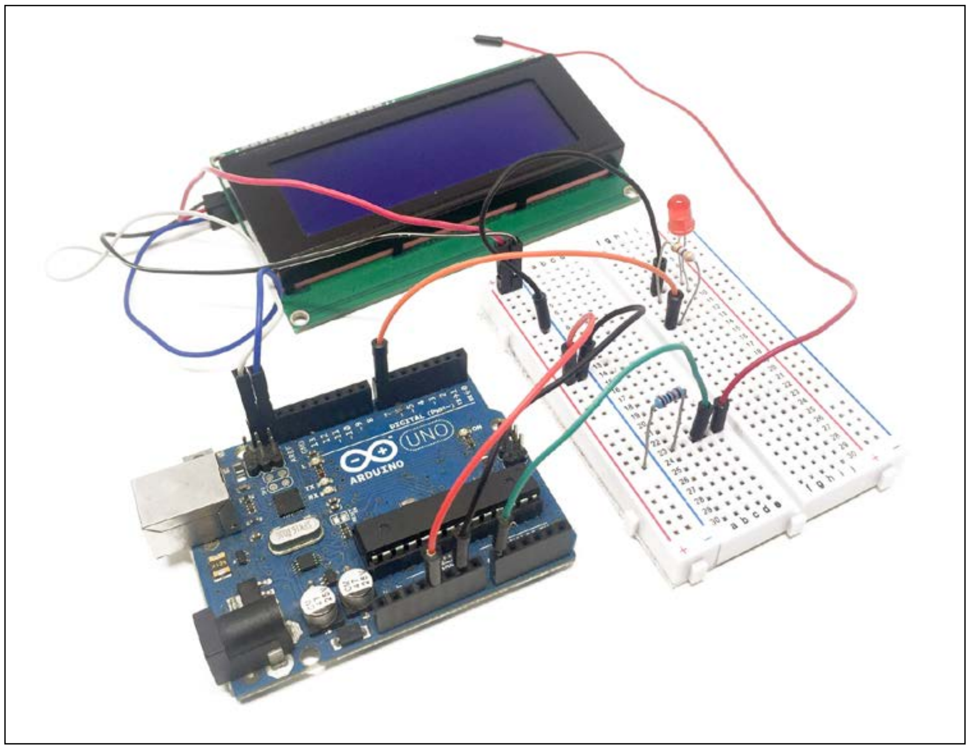
First, connect the power to the breadboard: connect GND to the blue power rail of the breadboard and the +5V pin to the red power rail.

Then, we are going to connect the antenna: first, place it on the breadboard in series with the 1M Ohm resistor. Then, connect the other end of the resistor to the ground. Finally, connect the antenna to the A0 analog pin.

For the LED, simply place it in series with the resistor, as seen on the schematic. Ensure that you connect the resistor to the anode of the LED, which is the longest pin of the LED. Finally, connect the other end of the resistor to the digital pin 7 and the other end of the LED to ground.

For the LCD screen, the connections are really easy, thanks to the I2C interface. First connect the power: VCC goes to the red power rail and GND to the blue power rail. For the data pin, connect SDA and SCL to their respective pins on the Arduino board, which are next to the digital pin 13.

The following image is the final result:



Congratulations! We are now going to see how to test the LCD screen.

Testing the LCD screen

Before we build our EMF bug detector, we want to make sure that the LCD screen is working correctly. Therefore, we are going to test it by printing a very simple message on it.

The following is a complete sketch to do this:

```
// Required libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Create LCD instance
LiquidCrystal_I2C lcd(0x27,20,4);

void setup()
{
  // Initialise LCD
  lcd.init();

  // Print a message to the LCD
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Hello Secret Agent!");
}


void loop()
{
}
```

As you can see, the sketch is pretty straightforward. You can now plug the Arduino project into your computer using a USB cable and then copy and paste the sketch into your Arduino IDE.

Then, upload the sketch to your board. This is what you should see:

If you can see this message, congratulations, you are ready to move to the next step!

Building the EMF bug detector

We are now going to dive into the core of this project and configure the project so that it can detect EMF activity around the antenna.

The following is the complete code for this project:

```
// Required libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Number of readings
#define NUMREADINGS 15

// Parameters for the EMF detector
int senseLimit = 15;
int probePin = 0;
int ledPin = 7;
int val = 0;
int threshold = 200;

// Averaging the measurements
```

```
int readings[NUMREADINGS];
int index = 0;
int total = 0
int average = 0;

// Time between readings
int updateTime = 40;

// Create LCD instance
LiquidCrystal_I2C lcd(0x27,20,4);

void setup()
{
  // Initialise LCD
  lcd.init();

  // Set LED as output
  pinMode(ledPin, OUTPUT);

  // Print a welcome message to the LCD
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("EMF Detector Started");
  delay(1000);
  lcd.clear();
}


void loop()
{
  // Read from the probe
  val = analogRead(probePin);
  Serial.println(val);

  // Check reading
  if(val >= 1){

    // Constrain and map with sense limit value
    val = constrain(val, 1, senseLimit);
    val = map(val, 1, senseLimit, 1, 1023);

    // Averaging the reading
    total -= readings[index];
    readings[index] = val;
```

```
        total += readings[index];
        index = (index + 1);

        if (index >= NUMREADINGS)
          index = 0;

        average = total / NUMREADINGS;

        // Print on LCD screen
        lcd.setCursor(0,1);
        lcd.print("   ");

        lcd.setCursor(0,0);
        lcd.print("EMF level: ");
        lcd.setCursor(0,1);
        lcd.print(average);

        // Light up LED if EMF activity detected
        if (average > threshold) {
          digitalWrite(ledPin, HIGH);
        }
        else {
          digitalWrite(ledPin, LOW);
        }

        // Wait until next reading
        delay(updateTime);
      }
    }
```

Now let's see the details of this code. It starts by including the required libraries:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Then, we define the number of readings that we want to do at every iteration in order to make sure that we get the average EMF reading:

```
#define NUMREADINGS 15
```

We will then define some parameters for the project, such as the different pins on which the components are connected:

```
int senseLimit = 15;
int probePin = 0;
int ledPin = 7;
int val = 0;
int threshold = 200;
```

After that, we will define some variables that will be used to take the average of the readings:

```
int readings[NUMREADINGS];
int index = 0;
int total = 0;
int average = 0;
```

We will also set an update time that we will leave between each EMF reading. By default, we set it to `40` milliseconds:

```
int updateTime = 40;
```

We will also create an instance of the LCD screen:

```
LiquidCrystal_I2C lcd(0x27,20,4);
```

After that, in the `setup()` function of the sketch, we will initialize the LCD screen:

```
// Initialise LCD
  lcd.init();

  // Set LED as output
  pinMode(ledPin, OUTPUT);

  // Print a welcome message to the LCD
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("EMF Detector Started");
  delay(1000);
  lcd.clear();
```

Then, in the `loop()` function of the sketch, we will get the value of the analog pin on which the antenna is connected:

```
val = analogRead(probePin);
```

After that, we will constrain the reading to the limit value that was defined previously, and then map it again between `1` and `1023`:

```
val = constrain(val, 1, senseLimit);
val = map(val, 1, senseLimit, 1, 1023);
```

Then, we will take an average of the readings:

```
total -= readings[index];
readings[index] = val;
total += readings[index];
index = (index + 1);

if (index >= NUMREADINGS)
  index = 0;
average = total / NUMREADINGS;
```

Finally, we will display the average on the LCD screen and also light up the LED if the average reading is higher than the threshold:

```
lcd.setCursor(0,1);
lcd.print("   ");

lcd.setCursor(0,0);
lcd.print("EMF level: ");
lcd.setCursor(0,1);
lcd.print(average);

// Light up LED if EMF activity detected
if (average > threshold) {
  digitalWrite(ledPin, HIGH);
}
else {
  digitalWrite(ledPin, LOW);
}
```

We will also wait for a given amount of time between each reading:

```
delay(updateTime);
```

Note that you can find the complete code in the GitHub repository of the project at `https://github.com/marcoschwartz/arduino-secret-agents`.

It's now time to test the project. You can copy the complete code and paste it into your Arduino IDE, or just get it from GitHub.

Then, upload the code to your project. After a while, you should see that the LCD screen is being initialized and is blank. This is because it is waiting to detect a significant EMF activity.

I, for example, approached my phone with the antenna and got the following result:

If you can see something on your screen, congratulations, you just built an EMF bug detector! Now, I invite you to try with other devices, such as a surveillance camera, any Wi-Fi device, a radio controller, and so on.

Summary

In this chapter, we built a simple EMF bug detector that a secret agent can use to see whether there are any bugs present in a room, such as an audio recorder or a spy camera.

There are, of course, several ways to improve this project. You can, for example, connect a battery to the project and integrate it in a small case in order to build a simple and portable EMF bug detector.

In the next chapter, we are going to build another useful device for a secret agent. We will see how to use a fingerprint scanner to secure access to important information.

4
Access Control with a Fingerprint Sensor

In this chapter, we are going to build a more complex secret agent project: an access control system using a fingerprint sensor. We will connect the fingerprint sensor to Arduino along with a relay and an LCD screen.

Based on this hardware, we will build several cool projects. The following are the steps that we will take for this project:

- First, we are going to record your fingerprint in the sensor so that you can get access
- Then, we will use the fingerprint sensor to open or close the relay
- Finally, we will create a system with the LCD screen to grant access to a secret piece of data stored in Arduino

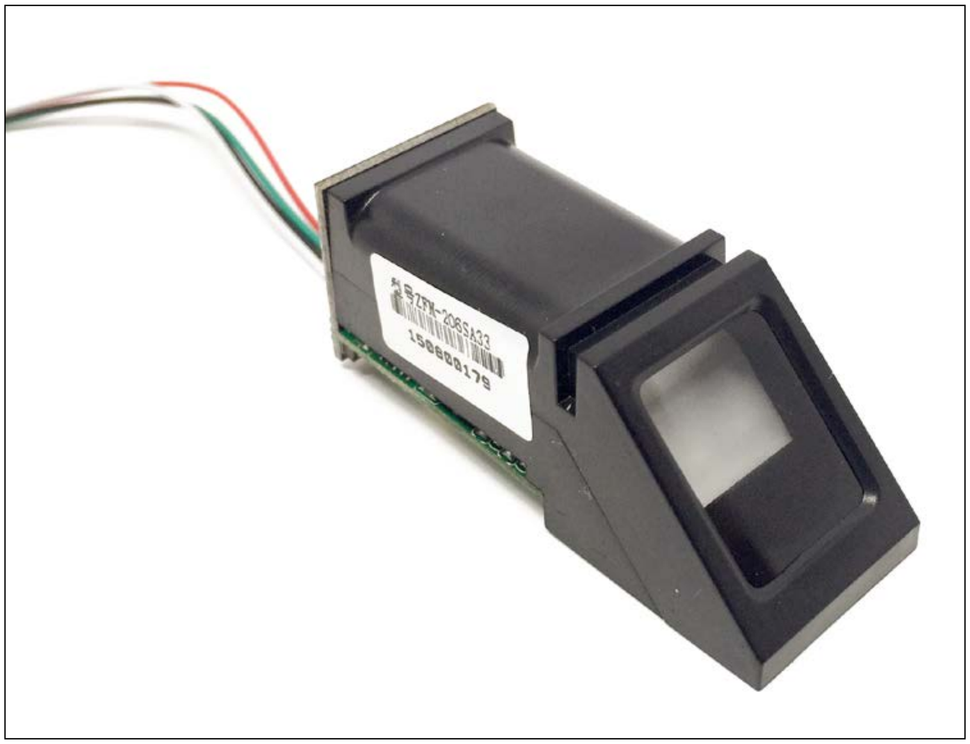
Let's dive in!

Hardware and software requirements

First, let's see with fingerprint sensor: software requisites "what with fingerprint sensor:hardware requisites" are the required components for this project.

As usual, we will use an Arduino Uno board as the *brain* of the project.

The most important part of this project is the fingerprint sensor. The following is an image of the sensor that I used:



You with fingerprint sensor:software requisites "need to get with fingerprint sensor:hardware requisites" the exact same model (from Adafruit). Otherwise, the code for this project won't work.

You will also need an LCD screen for the last part of this project. I used an I2C LCD screen from DFRobot that we already used earlier in the book.

I also used a Pololu 5V relay module, which is really convenient to connect to Arduino. A relay will basically allow us to control a wide range of devices, for example, from a simple LED to electrical appliances.

Finally, here is the list of all the components that we will use in this project:

- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- Adafruit Fingerprint Sensor (`https://www.adafruit.com/products/751`)
- DFRobot 4x20 LCD screen (`http://www.robotshop.com/en/dfrobot-i2c-twi-lcd-module.html`)

- Pololu relay module (`https://www.pololu.com/product/2480`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

On the software side, you will need two Arduino libraries: the `LiquidCrystal_I2C` library with fingerprint sensor:software requisites "for the LCD with fingerprint sensor:hardware requisites" screen and the `Adafruit Fingerprint Sensor` library. You can get them both using the Arduino library manager.

You can also visit the GitHub repository of the Fingerprint Sensor library, to learn more about the different functions, available at `https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library`.

Hardware configuration

We are first going to see how to assemble the different parts of this project.

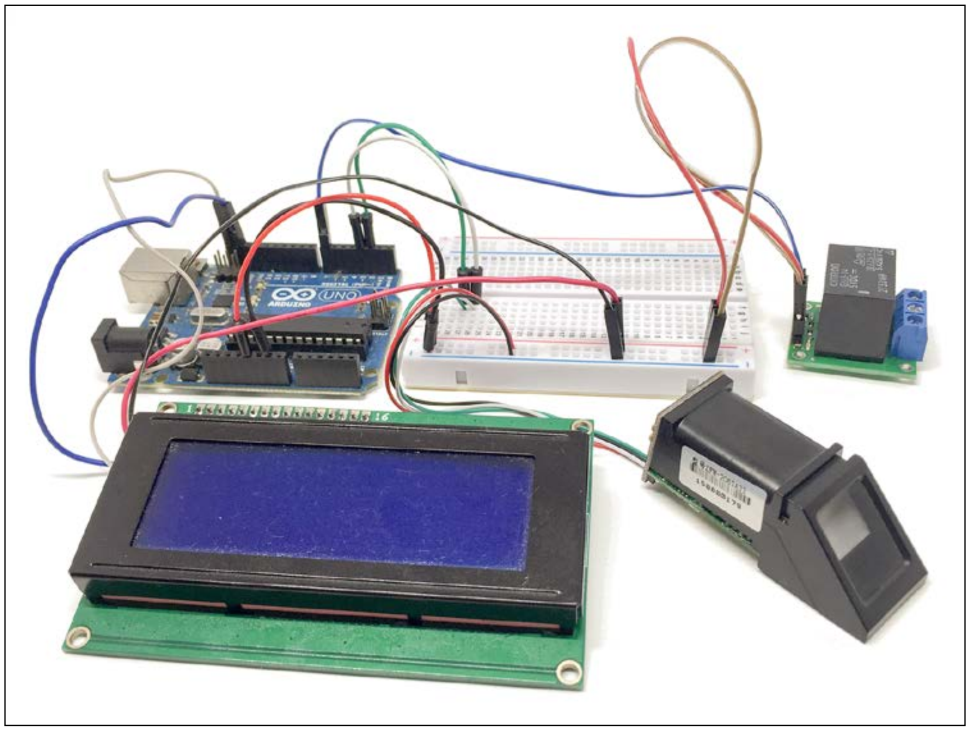
Let's start by connecting the power supply. Connect the 5V pin from the Arduino board to the red power rail and the GND from Arduino to the blue power rail on the breadboard.

Now, let's connect the fingerprint sensor. First, connect the power by connecting the cables to their respective color on the breadboard. Then, connect the white wire from the sensor to Arduino pin 3 and the green wire to pin number 2.

After that, we are going to connect the relay module. Connect the VCC pin to the red power rail, GND pin to the blue power rail, and the EN pin to Arduino pin 7.

Finally, let's connect the LCD screen. First, connect the power: VCC to the red power rail and GND pin to the blue power rail. After that, connect the I2C pins (SDA and SCL) to the Arduino board. The I2C pins are next to pin 13 on the Arduino Uno board.

The following is the final result:



If your project is similar to the preceding image, congratulations, you can now proceed to the next section of this chapter and build exciting applications based on the fingerprint sensor!

Enrolling your fingerprint

The first thing that we have to do is enroll at least one fingerprint so that it can be later recognized by the sensor. We will do that in this section. Here is most of the code for this section:

```
// Libraries
#include <Adafruit_Fingerprint.h>
#include <SoftwareSerial.h>

// Fingerprint enroll function
uint8_t getFingerprintEnroll(uint8_t id);

// Software Serial instance
```

```
SoftwareSerial mySerial(2, 3);

// Fingerprint sensor instance
Adafruit_Fingerprint finger = Adafruit_Fingerprint(andmySerial);

void setup()
{
  // Start serial
  Serial.begin(9600);
  Serial.println("fingertest");

  // Set the data rate for the sensor serial port
  finger.begin(57600);

  // Verify that sensor is present
  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1);
  }
}

void loop()
{
  // Wait for fingerprint ID
  Serial.println("Type in the ID # you want to save this finger
as...");
  uint8_t id = 0;
  while (true) {
    while (! Serial.available());
    char c = Serial.read();
    if (! isdigit(c)) break;
    id *= 10;
    id += c - '0';
  }
  Serial.print("Enrolling ID #");
  Serial.println(id);

  while (!  getFingerprintEnroll(id) );
}
```

You will note that I didn't include all the details of fingerprint sensor functions as they are too long to be displayed here. You can, of course, find the whole code in the GitHub repository of this book.

Now, let's see the details of the code. It starts by including the required libraries:

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

Then, we declare the function that we will use in order to enroll our fingerprint:

```
uint8_t getFingerprintEnroll(uint8_t id);
```

After that, we create the software serial instance that we will use to communicate with the server:

```
SoftwareSerial mySerial(2, 3);
```

We will also create the fingerprint sensor instance:

```
Adafruit_Fingerprint finger = Adafruit_Fingerprint(andmySerial);
```

Now, in the `setup()` function of the sketch, we will initialize serial communications:

```
Serial.begin(9600);
Serial.println("fingertest");
```

Then, we will initialize the communication with the sensor:

```
finger.begin(57600);
```

We will also check whether the sensor is present:

```
if (finger.verifyPassword()) {
  Serial.println("Found fingerprint sensor!");
} else {
  Serial.println("Did not find fingerprint sensor :(");
  while (1);
}
```

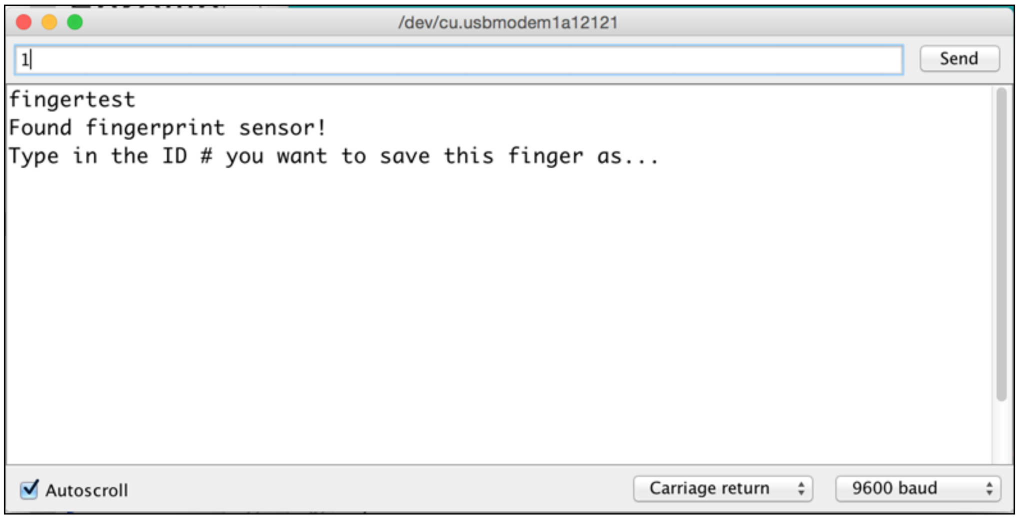
In the `loop()` function of the code, we will first wait for an input from the user, which is the ID of the fingerprint that we want to store. Then, we go through the process of storing the fingerprint. This is all done by the following code snippet:

```
// Wait for fingerprint ID
Serial.println("Type in the ID # you want to save this finger as...");
  uint8_t id = 0;
  while (true) {
    while (! Serial.available());
```

```
      char c = Serial.read();
      if (! isdigit(c)) break;
      id *= 10;
      id += c - '0';
    }
    Serial.print("Enrolling ID #");
    Serial.println(id);

    while (!  getFingerprintEnroll(id) );
```

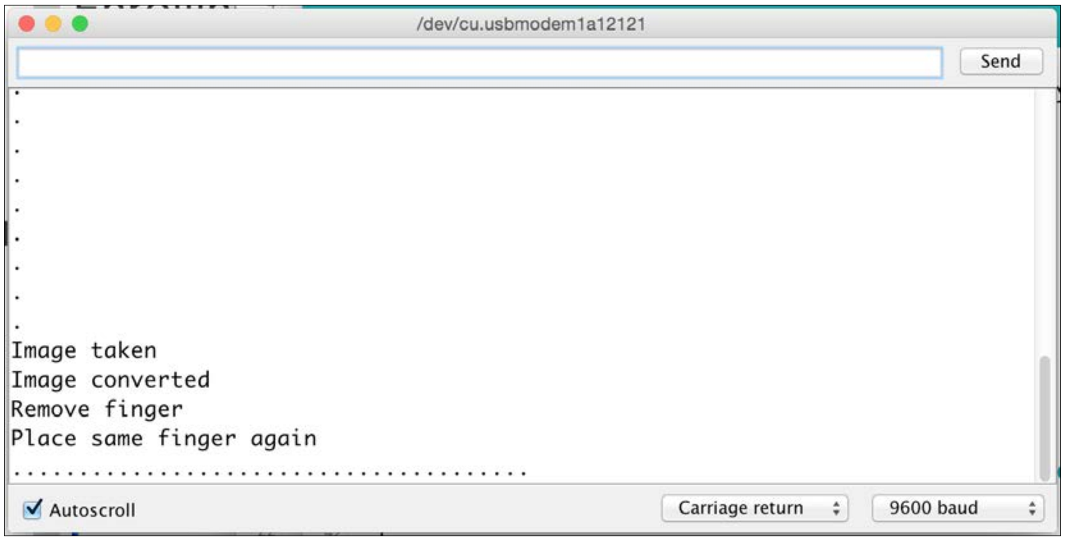
It's now the time to test the enrollment process. First, get the complete code, for example, from the GitHub repository of the book, which is available at `https://github.com/marcoschwartz/arduino-secret-agents`.

Then, copy the code in the Arduino IDE. After that, upload the code to the Arduino board and open the serial monitor with a speed of 9600 baud. The following screenshot is what you should see:



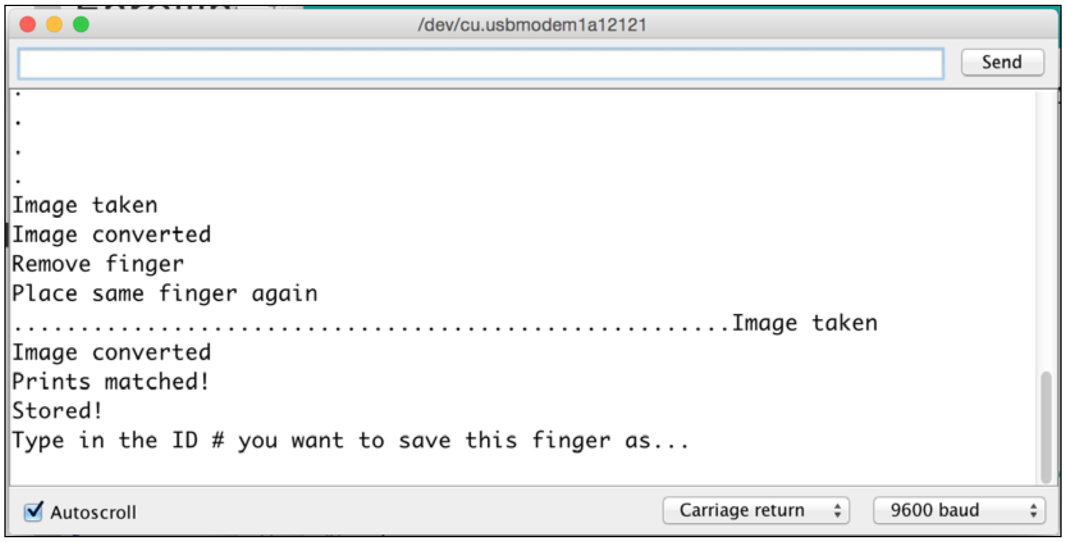
```
fingertest
Found fingerprint sensor!
Type in the ID # you want to save this finger as...
```

On being prompted, enter the ID of the fingerprint that you want to store and press enter. The sketch will now ask you to put your finger on the sensor. Do so and, after a while, you should see that the image was taken and you can now remove your finger:



```
.
.
.
.
.
.
.
.
.
Image taken
Image converted
Remove finger
Place same finger again
.......................................
```

Then, as asked by the sketch, put your finger on the sensor once more. The sketch will then confirm that the fingerprint has been stored:



```
.
.
.
.
Image taken
Image converted
Remove finger
Place same finger again
......................................................Image taken
Image converted
Prints matched!
Stored!
Type in the ID # you want to save this finger as...
```

If you can see this message, it means that your fingerprint is now stored in the sensor and you can move to the next section.

Controlling access to the relay

Now that your fingerprint is stored in the sensor, you can build your first application using the hardware that we previously built. We are going to open or close the relay every time the sensor recognizes our fingerprint. The following is the complete code for this part, excluding the details about recognition functions:

```
// Libraries
#include <Adafruit_Fingerprint.h>
#include <SoftwareSerial.h>

// Function to get fingerprint
int getFingerprintIDez();

// Init Software Serial
SoftwareSerial mySerial(2, 3);

// Fingerprint sensor instance
Adafruit_Fingerprint finger = Adafruit_Fingerprint(andmySerial);

// Relay parameters
int relayPin = 7;
bool relayState = false;

// Your stored finger ID
int fingerID = 0;

// Counters
int activationCounter = 0;
int lastActivation = 0;

void setup()
{

  // Start Serial
  Serial.begin(9600);

  // Set the data rate for the sensor serial port
  finger.begin(57600);

  // Check if sensor is present
  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1);
  }
  Serial.println("Waiting for valid finger...");

  // Set relay as output
  pinMode(relayPin, OUTPUT);
}

void loop()
{
  // Get fingerprint # ID
  int fingerprintID = getFingerprintIDez();

  // Activation ?
  if ( (activationCounter - lastActivation) > 2000) {

  if (fingerprintID == fingerID && relayState == false) {
    relayState = true;
    digitalWrite(relayPin, HIGH);
    lastActivation = millis();
  }
  else if (fingerprintID == fingerID && relayState == true) {
    relayState = false;
    digitalWrite(relayPin, LOW);
    lastActivation = millis();
  }

  }
  activationCounter = millis();
  delay(50);
}
```

As you can see, many elements are common with the sketch that we saw in the previous section. We are only going to see those elements which are important and are added to this new sketch.

We have to define on which pin the relay is connected and also state that the relay is off by default:

```
int relayPin = 7;
bool relayState = false;
```

Then, we will define the ID under which we stored the fingerprint in the previous section. I used `0` here as I stored my fingerprint with the ID number `0`:

```
int fingerID = 0;
```

Also, we don't want the relay to continuously switch state when we have our finger on the sensor. Therefore, we need the following two variables to count 2 seconds before the state of the relay can be changed again:

```
int activationCounter = 0;
int lastActivation = 0;
```

We will then set the relay pin as an output:

```
pinMode(relayPin, OUTPUT);
```

Then, in the `loop()` function of the sketch, we check whether the sensor is reading any fingerprint ID that is already stored in the sensor:

```
int fingerprintID = getFingerprintIDez();
```

The following is the check for the activation period:

```
if ( (activationCounter - lastActivation) > 2000) {
```

We will then check whether the ID corresponds to the ID that we defined earlier and also check the state of the relay. If the ID corresponds to the ID that we entered in the sketch, we switch the state of the relay:

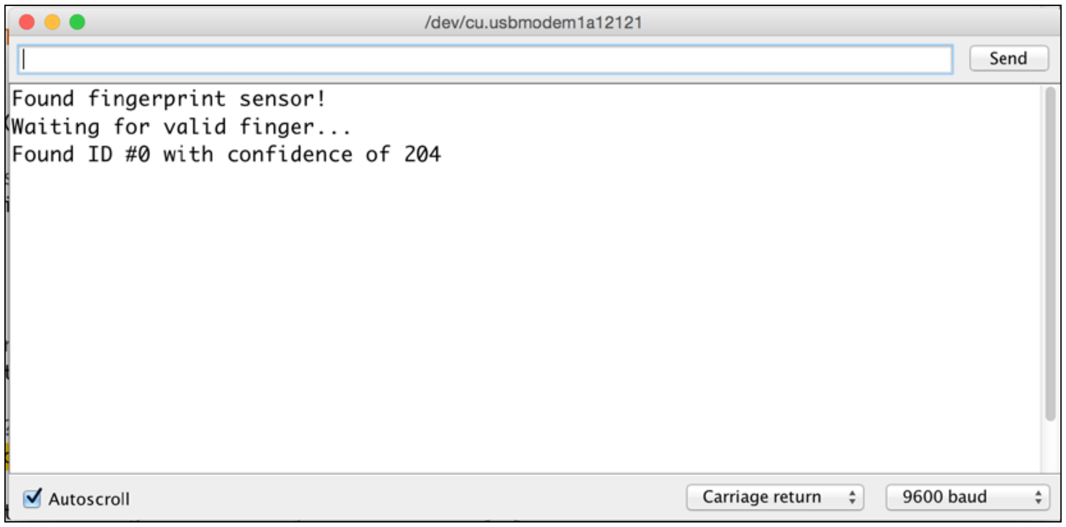
```
if (fingerprintID == fingerID && relayState == false) {
     relayState = true;
  digitalWrite(relayPin, HIGH);
  lastActivation = millis();
   }
else if (fingerprintID == fingerID && relayState == true) {
  relayState = false;
  digitalWrite(relayPin, LOW);
  lastActivation = millis();
}
```

Finally, we will refresh the activation counter and wait 50 milliseconds until the next read:

```
activationCounter = millis();
delay(50);
```

It's now time to test the sketch. Get all the code, for example, from the GitHub repository of the book and then upload it to the board. Make sure that you change the ID in the sketch, corresponding to the fingerprint that you stored earlier.

Then, open the serial monitor with the speed of 9600 baud. Place the finger that you recorded previously on the sensor. You should immediately see the following in the serial monitor:



```
Found fingerprint sensor!
Waiting for valid finger...
Found ID #0 with confidence of 204
```

You should also hear the relay click, meaning that it just changed its state. You can now do the same operation after some seconds; you should hear the relay switch back to its initial state.

You can try with another finger or ask someone else to try the project; nothing will happen at all.

Accessing secret data

In the last section of this chapter, we are going to use all the hardware that we connected to the project for another cool application: accessing a secret piece of data with your fingerprint.

It can, for example, be a secret code that you only want to be accessible with your own fingerprint. We will use the LCD screen for this, removing the need to have the serial monitor opened.

The following is the complete code for this part, excluding the functions to read the fingerprint sensor data:

```
// Libraries
#include <Adafruit_Fingerprint.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

// LCD Screen instance
LiquidCrystal_I2C lcd(0x27,20,4);

// Function to get fingerprint
int getFingerprintIDez();

// Init Software Serial
SoftwareSerial mySerial(2, 3);

// Fingerprint sensor instance
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);

// Relay parameters
int relayPin = 7;
bool relayState = false;

// Your stored finger ID
int fingerID = 0;

// Counters
int activationCounter = 0;
int lastActivation = 0;

// Secret data
String secretData = "u3fks43";

void setup()
{

  // Start Serial
  Serial.begin(9600);

  // Set the data rate for the sensor serial port
  finger.begin(57600);

  // Check if sensor is present
  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!");
  } else {
    Serial.println("Did not find fingerprint sensor :(");
    while (1);
  }
  Serial.println("Waiting for valid finger...");

  // Set relay as output
  pinMode(relayPin, OUTPUT);

  // Init display
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Scan your finger");
}

void loop()
{
  // Get fingerprint # ID
  int fingerprintID = getFingerprintIDez();

  // Activation ?
  if ( (activationCounter - lastActivation) > 2000) {

    if (fingerprintID == fingerID) {

      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Access granted!");
      lcd.setCursor(0,1);
      lcd.print("Your secret data is:");
      lcd.setCursor(0,2);
      lcd.print(secretData);

      if (relayState == false) {
        relayState = true;
        digitalWrite(relayPin, HIGH);
      }
```

```
        else if (relayState == true) {
          relayState = false;
          digitalWrite(relayPin, LOW);
        }
        lastActivation = millis();
      }
    }
    activationCounter = millis();
    delay(50);
  }
```

As you can see, this code has some common parts with the code that we saw in the earlier section. Therefore, we are only going to talk about the most important elements here.

It starts by including the required libraries:

```
#include <Adafruit_Fingerprint.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
```

Then, we need to create an instance of the `LiquidCrystal_I2C` LCD screen:

```
LiquidCrystal_I2C lcd(0x27,20,4);
```

We will also define a string containing our secret data:

```
String secretData = "u3fks43";
```

Of course, feel free to put anything here. Then, in the `setup()` function, we will initialize the LCD screen:

```
lcd.init();
lcd.backlight();
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Scan your finger");
```

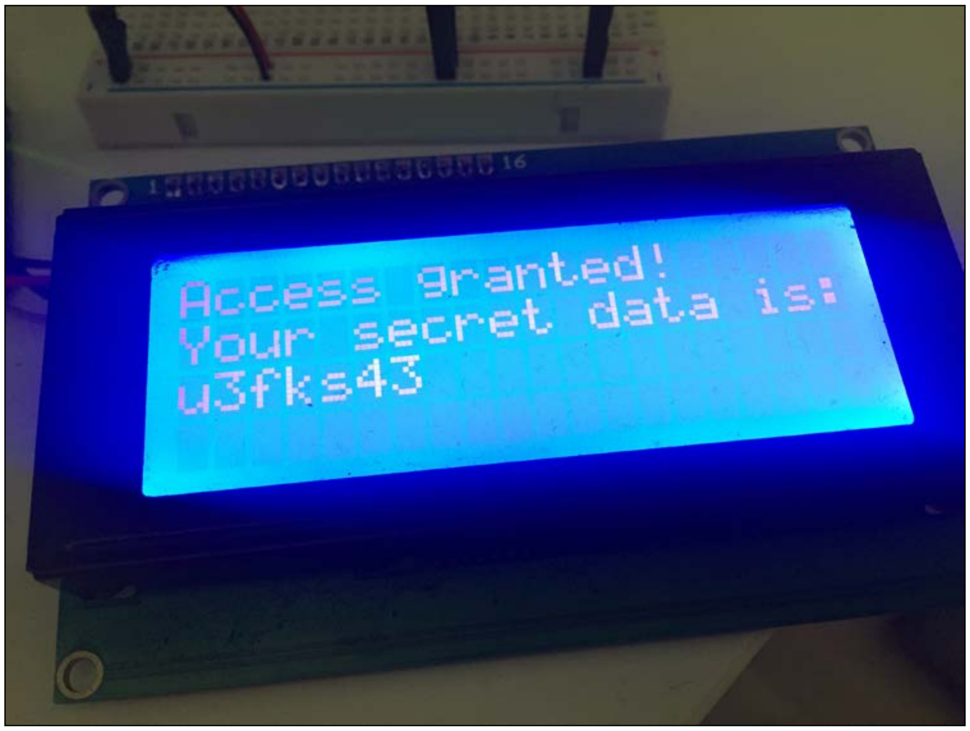
Then, in the `loop()` function of the sketch, if we are outside of the activation period of 2 seconds, we will compare the fingerprint that was measured with the ID that we set in the sketch. If this is correct, we will print a message on the LCD saying that the access was granted along with the secret data:

```
if (fingerprintID == fingerID) {

      lcd.clear();
      lcd.setCursor(0,0);
```

```
        lcd.print("Access granted!");
        lcd.setCursor(0,1);
        lcd.print("Your secret data is:");
        lcd.setCursor(0,2);
        lcd.print(secretData);
```

Now, it's time to test the sketch. Get all the code, for example, from the GitHub repository of the project and upload it to the board.

Then, place your finger that you previously recorded on the sensor. This is what you should see:



```
Access granted!
Your secret data is:
u3fks43
```

If you can see the preceding result, congratulations, you just built your own fingerprint access control system!

Summary

In this project, we built an access control system using a fingerprint sensor with Arduino. We also built several cool applications based on it, including a way to get access to a secret piece of data using your fingerprint.

There are, of course, many things you can do in order to improve this project. For example, you can code a way for the secret message to disappear after a given amount of time; avoiding someone else getting access to it after you used the project. You could also connect the board to the web (for example, with an Ethernet shield) and use the fingerprint sensor to send a given tweet when you put your finger on it.

In the next chapter, we are going to use another piece of hardware to build an access control system: a GSM shield that will allow us to open a door just by sending an SMS!

5 Opening a Lock with an SMS

In this chapter, we are going to build another great application for secret agents: opening a door lock simply by sending an SMS! You'll just need to send a message to a given number and then you'll be able to control a door lock or any other kind of on/off digital device, such as an alarm.

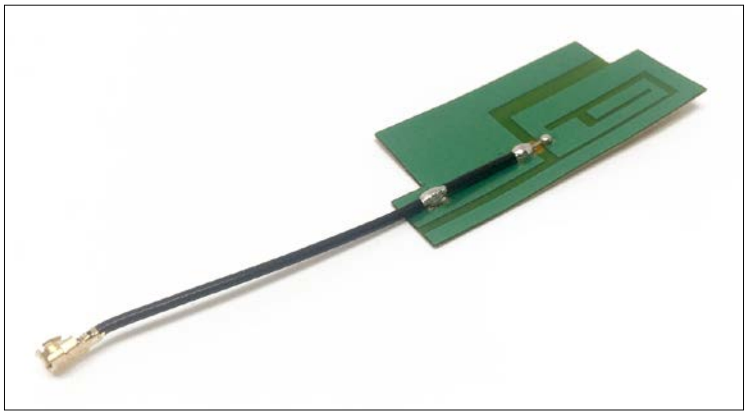
For that, we are going to take a number of steps, as follows:

- First, we are going to use Arduino and the FONA shield from Adafruit in order to be able to receive and process text messages
- Then, we'll connect this to a relay and LED to see whether we can actually control a device by sending an SMS
- Finally, we'll see how to connect an actual electronic lock to the system

Let's start!

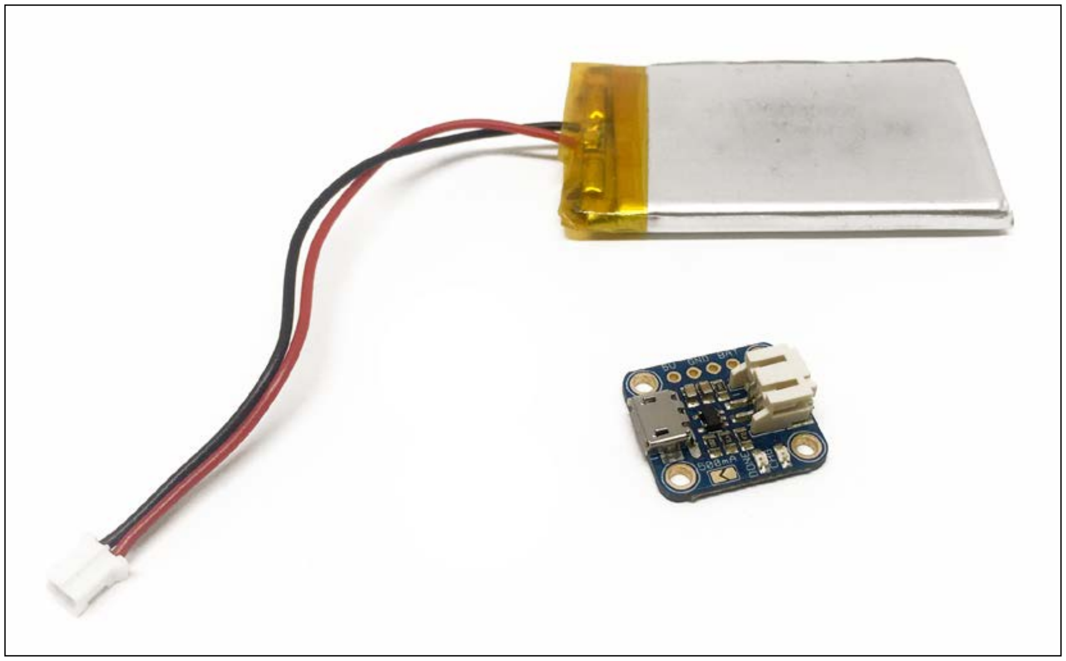
Hardware and software requirements

First, let's see what the required components for this project are. The most important parts are related to GSM functionalities, which is the central piece in our project. We'll need an antenna in order to be able to connect to the local GSM network. For this project, a flat uFL antenna is used:



Then, you'll need a way to actually use a SIM card, connect to the GSM network, and process the information with Arduino. There are many boards that can do this; however, I recommend the Adafruit FONA shield, which is very convenient to configure and use with Arduino. The following is the image of the Adafruit FONA shield along with the flat GSM antenna.

Then, you will need a battery to power the FONA shield, as the Arduino Uno board doesn't allow to power the chip that is at the core of the FONA shield (it can use up to 2A at a time!). For this, I used a 3.7 LiPo battery along with a micro USB battery charger:



A very important part of the project is the SIM card that you need to place in the FONA shield. You need a normal SIM card (not micro or nano), which is activated, not locked by a PIN, and is able to receive text messages. You can get one at any of your local mobile network operators.

Then, to test the functionalities of the project, we'll also use a simple LED (along with a 330 Ohm resistor) and a 5V relay. This will mimic the behavior of the real electronic lock.

Finally, you'll need an electronic lock. This part is optional as you can completely test everything without the lock and it is quite complicated to set up. For this, you'll need an Adafruit electronic lock, a 1K Ohm resistor, and a power transistor.

Finally, here is the list of all the components that we will use in this project:

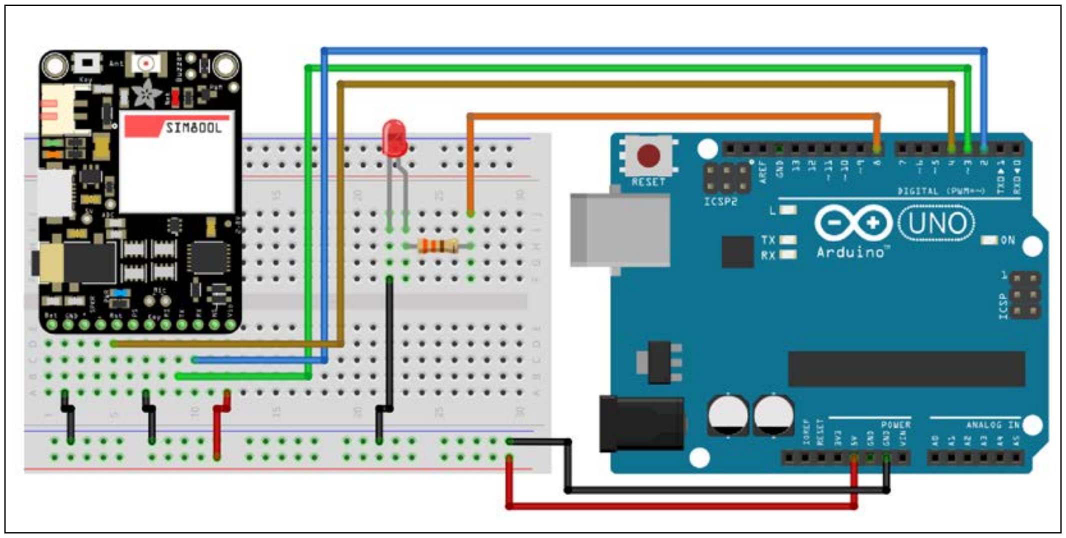
- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- Adafruit Fona 808 shield (`http://www.adafruit.com/product/2542`)
- GSM uFL antenna (`http://www.adafruit.com/products/1991`)
- GSM SIM card
- 3.7V LiPo battery (`http://www.adafruit.com/products/328`)
- LiPo battery charger (`http://www.adafruit.com/products/1904`)
- LED (`https://www.sparkfun.com/products/9590`)
- 330 Ohm resistor (`https://www.sparkfun.com/products/8377`)
- 5V relay (`https://www.pololu.com/product/2480`)
- Optional: Adafruit electrical lock (`http://www.adafruit.com/products/1512`)
- Optional: Rectifier diode (`https://www.sparkfun.com/products/8589`)
- Optional: Power transistor (`http://www.adafruit.com/products/976`)
- Optional: 1K Ohm resistor (`https://www.sparkfun.com/products/8980`)
- Optional: 12V power supply (`https://www.sparkfun.com/products/9442`)
- Optional: DC jack adapter (`https://www.sparkfun.com/products/10288`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

On the software side, you'll only need the latest version of the Arduino IDE and the Adafruit FONA library. You can install this library using the Arduino IDE library manager.

Hardware configuration

It's now the time to assemble the hardware for the project. We'll first connect the Adafruit FONA shield and then the other components. Before assembling the hardware, make sure that you have inserted the SIM card into the FONA shield.

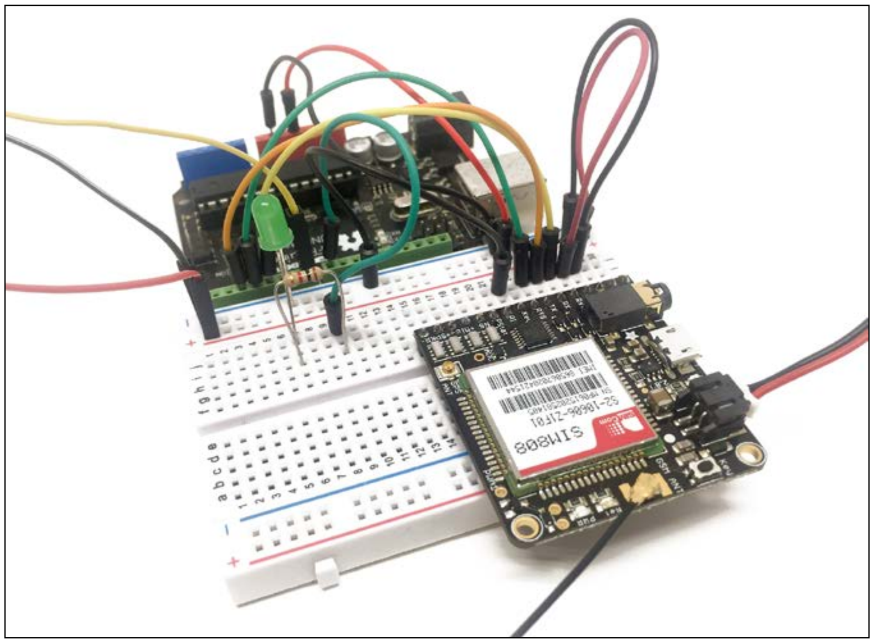
The following is a schematic to help you out:



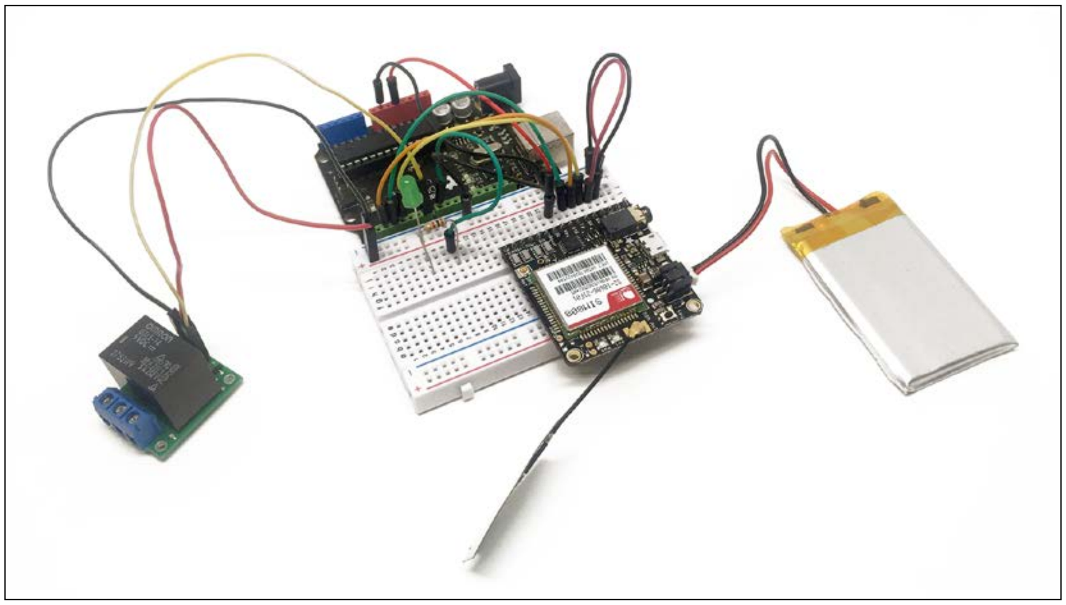
Note that the location of the pins can be different on your FONA shield, as there are many versions available. Also, note that the relay is not represented on this schematic. Here are the steps that you need to take in order to assemble the hardware:

1. First, connect the power supply to the breadboard. Connect the 5V pin from the Arduino board to the red power line on the breadboard and the GND pin to the blue power line.
2. Then, place the FONA shield on the breadboard. Connect the VIO pin to the red power line and the GND and key pins to the blue power line.
3. After that, connect the RST pin to Arduino pin 4, TX to Arduino pin 3, and RX to Arduino pin 2. Also, connect the 3.7V LiPo battery and antenna to the FONA shield.
4. Finally, connect the relay and the LED to the Arduino board. Connect the relay VCC and GND pin to the power supply on the breadboard and the EN pin to pin 7 of the Arduino board.
5. For the LED, place it in series with a 330 Ohm resistor on the breadboard, with the longest side of the LED connected to the resistor. Then, connect the other pin of the resistor to pin 8 of the Arduino board. Also, connect the other pin of the LED to the ground.

The following is an image of the completed project, with a zoom around the FONA shield and Arduino board:



The following is an overview of the completely assembled project:



Testing the FONA shield

Now that our hardware is ready, we are going to test the FONA shield to see whether it is correctly connected and can connect to the network. As a test, we will send an SMS to the shield and also read all the messages present in the SIM card.

This is the complete Arduino sketch for this part, minus some helper functions that won't be detailed here:

```
// Include library
#include "Adafruit_FONA.h"
#include <SoftwareSerial.h>

// Pins
#define FONA_RX 2
#define FONA_TX 3
#define FONA_RST 4

// Buffer for replies
char replybuffer[255];

// Software serial
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = &fonaSS;

// Fona
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);

// Readline function
uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout = 0);

void setup() {

  // Start Serial
  while (!Serial);
  Serial.begin(115200);
  Serial.println(F("FONA basic test"));
  Serial.println(F("Initializing....(May take 3 seconds)"));

  // Init FONA
  fonaSerial->begin(4800);
  if (! fona.begin(*fonaSerial)) {
    Serial.println(F("Couldn't find FONA"));
    while(1);
  }
```

```
    Serial.println(F("FONA is OK"));

    // Print SIM card IMEI number.
    char imei[15] = {0}; // MUST use a 16 character buffer for IMEI!
    uint8_t imeiLen = fona.getIMEI(imei);
    if (imeiLen > 0) {
      Serial.print("SIM card IMEI: "); Serial.println(imei);
    }

  }

  void loop() {

    // Get number of SMS
    int8_t smsnum = fona.getNumSMS();
    if (smsnum < 0) {
      Serial.println(F("Could not read # SMS"));
    } else {
      Serial.print(smsnum);
      Serial.println(F(" SMS's on SIM card!"));
    }

    // Read last SMS
    flushSerial();
    Serial.print(F("Read #"));
    uint8_t smsn = smsnum;
    Serial.print(F("\n\rReading SMS #")); Serial.println(smsn);

    // Retrieve SMS sender address/phone number.
    if (! fona.getSMSSender(smsn, replybuffer, 250)) {
      Serial.println("Failed!");
    }
    Serial.print(F("FROM: ")); Serial.println(replybuffer);

    // Retrieve SMS value.
    uint16_t smslen;
    if (! fona.readSMS(smsn, replybuffer, 250, &smslen)) { // pass in
  buffer and max len!
      Serial.println("Failed!");
    }
    Serial.print(F("***** SMS #")); Serial.print(smsn);
    Serial.print(" ("); Serial.print(smslen); Serial.println(F(") bytes
  *****"));
    Serial.println(replybuffer);
```

```
      Serial.println(F("*****"));

      // Flush input
      flushSerial();
      while (fona.available()) {
        Serial.write(fona.read());
      }

      // Wait
      delay(10000);

    }
```

Now, let's see the details of this sketch. It starts by including the required libraries:

```
#include "Adafruit_FONA.h"
#include <SoftwareSerial.h>
```

Then, we will define the pins on which the FONA shield is connected:

```
#define FONA_RX 2
#define FONA_TX 3
#define FONA_RST 4
```

We will also define a buffer that will contain the SMS that we will read from the shield:

```
char replybuffer[255];
```

Then, we'll use a `SoftwareSerial` instance to communicate with the shield:

```
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = andfonaSS;
```

We will also create an instance of the FONA library:

```
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);
```

Then, in the `setup()` function of the sketch, we will start the `Serial` object for the purpose of debugging and print a welcome message:

```
while (!Serial);
Serial.begin(115200);
Serial.println(F("FONA basic test"));
Serial.println(F("Initializing....(May take 3 seconds)"));
```

After that, we will initialize the FONA shield:

```
fonaSerial->begin(4800);
if (! fona.begin(*fonaSerial)) {
  Serial.println(F("Couldn't find FONA"));
  while(1);
}
Serial.println(F("FONA is OK"));
```

Then, we will read the IMEI of the SIM card in order to check whether the shield is working and whether a card is present:

```
char imei[15] = {0}; // MUST use a 16 character buffer for IMEI!
uint8_t imeiLen = fona.getIMEI(imei);
if (imeiLen > 0) {
  Serial.print("SIM card IMEI: "); Serial.println(imei);
}
```

Now, in the `loop()` function of the sketch, we will first count the number of text messages present in the card:

```
int8_t smsnum = fona.getNumSMS();
if (smsnum < 0) {
  Serial.println(F("Could not read # SMS"));
} else {
  Serial.print(smsnum);
  Serial.println(F(" SMS's on SIM card!"));
}
```

Then, we will also read the last one and print it on the serial monitor:

```
// Read last SMS
flushSerial();
Serial.print(F("Read #"));
uint8_t smsn = smsnum;
Serial.print(F("\n\rReading SMS #")); Serial.println(smsn);

 // Retrieve SMS sender address/phone number.
if (! fona.getSMSSender(smsn, replybuffer, 250)) {
  Serial.println("Failed!");
}
Serial.print(F("FROM: ")); Serial.println(replybuffer);

// Retrieve SMS value.
uint16_t smslen;
```

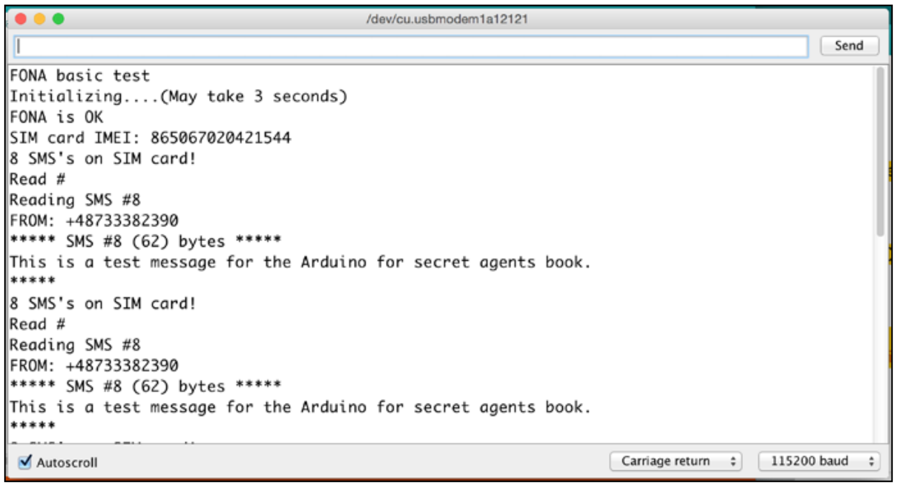
```
if (! fona.readSMS(smsn, replybuffer, 250, &smslen)) { // pass in
buffer and max len!
  Serial.println("Failed!");
}
Serial.print(F("***** SMS #")); Serial.print(smsn);
Serial.print(" ("); Serial.print(smslen); Serial.println(F(") bytes
*****"));
Serial.println(replybuffer);
Serial.println(F("*****"));
```

Once that's done, we will flush the `SoftwareSerial` instance and wait for 10 seconds until the next read:

```
// Flush input
flushSerial();
while (fona.available()) {
  Serial.write(fona.read());
}

// Wait
delay(10000);
```

Note that the complete code for this section can be found on the GitHub repository of the book at `https://github.com/marcoschwartz/arduino-secret-agents`.

It's now the time to test the FONA shield. Make sure that you get all the code and open it in your Arduino IDE. Then, upload the code to the board and open the serial monitor. You should see the following screenshot:



```
FONA basic test
Initializing....(May take 3 seconds)
FONA is OK
SIM card IMEI: 865067020421544
8 SMS's on SIM card!
Read #
Reading SMS #8
FROM: +48733382390
***** SMS #8 (62) bytes *****
This is a test message for the Arduino for secret agents book.
*****
8 SMS's on SIM card!
Read #
Reading SMS #8
FROM: +48733382390
***** SMS #8 (62) bytes *****
This is a test message for the Arduino for secret agents book.
*****
```

If everything is working fine, you should see the IMEI number of the SIM card and then the latest SMS in the card. You can test it by sending a message to the phone number of the SIM card; it should immediately show in the next reading of the SIM card. If this works, congratulations! You can now move to the next section.

Controlling the relay

In this part, we are going to program the Arduino board to remotely control the relay and LED that are connected to the board by sending an SMS to the FONA shield. As most of the code is similar to the code that we saw in the previous section, I'll only detail the new elements of code here.

We need to define the pins on which the relay and the LED are connected:

```
#define RELAY_PIN 7
#define LED_PIN 8
```

Then, we'll define a set of variables for a counter for the relay/lock. This is needed because after we open a lock, for example, we will want it to close automatically after a given amount of time for the purpose of safety. Here, we use a delay of 5 seconds:

```
bool lock_state = false;
int init_counter = 0;
int lock_counter = 0;
int lock_delay = 5000;
```

Then, we will define variables to count the number of SMS stored in the card:

```
int8_t smsnum = 0;
int8_t smsnum_old = 0;
```

We will also set two "passwords" to activate or deactivate the relay. I used very simple passphrases here; however, you can use the one that you want:

```
String password_on = "open";
String password_off = "close";
```

Then, in the `setup()` function of the sketch, we will set the relay and LED pins as output:

```
pinMode(RELAY_PIN, OUTPUT);
pinMode(LED_PIN, OUTPUT);
```

In the `loop()` function of the sketch, we will get the total number of SMS stored in the SIM card:

```
smsnum = fona.getNumSMS();
```

Basically, what we want is to check whether a new SMS has been received by the shield. To do that, we will compare this new reading to the old number of SMS stored in the SIM card:

```
if (smsnum > smsnum_old) {
```

If that's the case, we store the message in a `String` object so that we can conduct some operations on it:

```
String message = String(replybuffer);
```

Then, if the received message matches with the "On" password that we set earlier, we will activate the relay and LED, and also start the counter:

```
if (message.indexOf(password_on) > -1) {
  Serial.println("Lock OPEN");
  digitalWrite(RELAY_PIN, HIGH);
  digitalWrite(LED_PIN, HIGH);
  lock_state = true;
  init_counter = millis();
}
```

We would do something similar if we received an "Off" message:

```
if (message.indexOf(password_off) > -1) {
    Serial.println("Lock CLOSE");
    digitalWrite(RELAY_PIN, LOW);
    digitalWrite(LED_PIN, LOW);
    lock_state = false;
  }
```

After that, we will update the counter:

```
lock_counter = millis();
```

Then, we check whether the delay has already passed since the last time the relay/lock was activated. If that's the case, we will deactivate the relay/close the lock automatically:

```
if (lock_state == true andand (lock_counter - init_counter) > lock_
delay) {
  Serial.println("Lock CLOSE");
  digitalWrite(RELAY_PIN, LOW);
  digitalWrite(LED_PIN, LOW);
  lock_state = false;
}
```

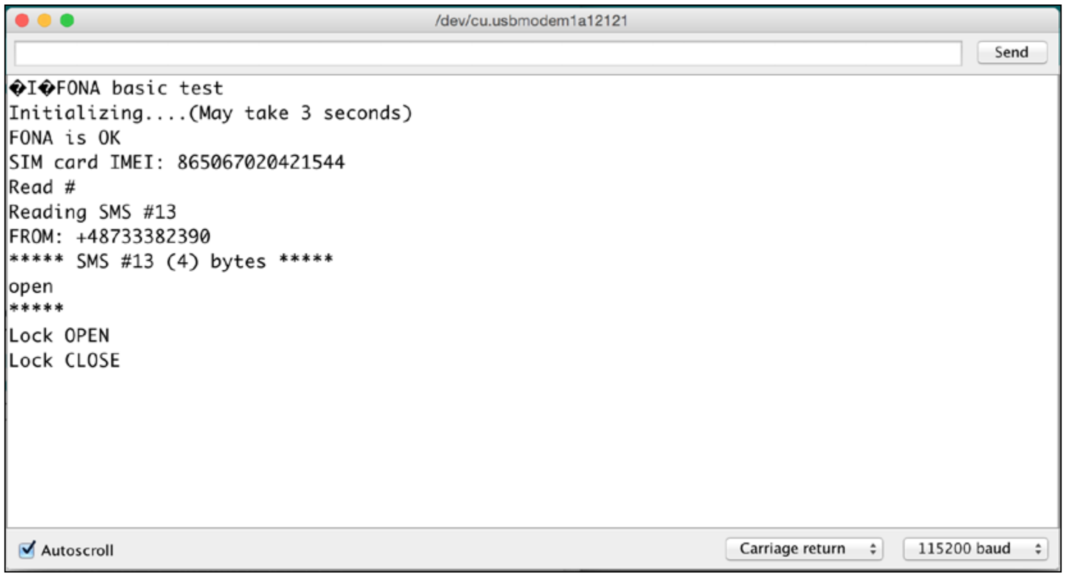
Finally, we will store the current number of received text messages in the `smsnum_old` variable:

```
smsnum_old = smsnum;
```

Note that the complete code can be found in the GitHub repository of the project at `https://github.com/marcoschwartz/arduino-secret-agents`.

It's time to test the project. Grab all the code and paste it in the Arduino IDE. Make sure that you change the on/off passwords in the code as well.

Then, upload the code to the board and open the serial monitor. You should see the shield being initialized and then waiting for a new text message. Now, send a text message to the shield with the message corresponding to the "On" password that you defined earlier. The following screenshot is what you should see:



```
�I�FONA basic test
Initializing....(May take 3 seconds)
FONA is OK
SIM card IMEI: 865067020421544
Read #
Reading SMS #13
FROM: +48733382390
***** SMS #13 (4) bytes *****
open
*****
Lock OPEN
Lock CLOSE
```

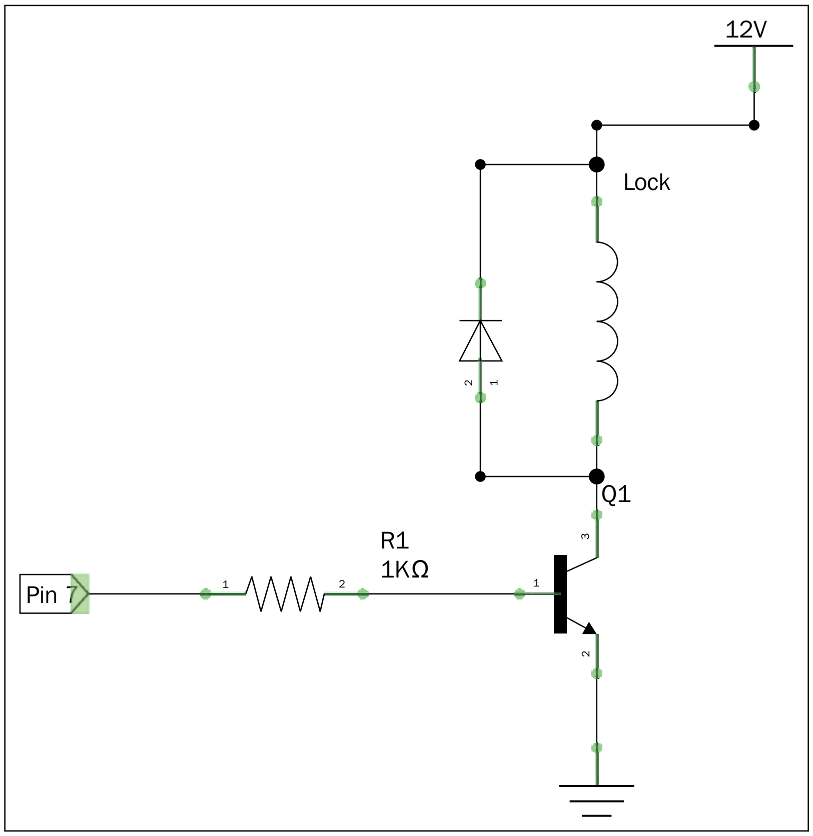
You should also see the LED turn on and the relay being activated. You can either wait until the system deactivates itself or you can simply send the "Off" passphrase again.

Opening and closing the lock

Now that we managed to actually control the relay and the LED by sending text messages to the FONA shield, we can connect the actual lock. This last step of the project is optional as you already tested the main functionality of the project and connecting the electronic door lock is a bit more technical. This is an image of the lock that I used:

First, you'll need to cut the JST connector from the electronic lock in order to be able to connect it the breadboard. I soldered some 2-pin header at the end of the electronic lock to make it compatible with my breadboard.

Then, you'll need to assemble the remaining optional components on the breadboard, as shown in the following schematic:



Let's see how to assemble these components together. First, place all the components on the breadboard. Then, connect the digital pin 7 of the Arduino board to the base of the transistor via the 1K Ohm resistor. Also, connect the emitter of the transistor to the ground of the project. Then, connect the electronic door lock between the collector of the transistor and the 12V power supply. Finally, mount the rectifier diode parallel to the door lock, with the orientation defined on the schematic.

Then, it's time to test the project again. You can use the exact same code as in the previous section. Congratulations, you can now control a door lock by sending a secret passphrase via SMS!

Summary

In this chapter, we built another very useful project for secret agents; a project where you can open a door lock by sending a secret code by SMS. You can also use this project for many other purposes. For example, you could use the same project to remotely activate an alarm when a text message is received by the project.

Note that you can also use the FONA shield to actually send text messages, which opens the door to even more exciting projects!

In the next chapter, we are going to see how to build a typical secret agent project: a remote spy camera that can be monitored from anywhere in the world!

6
Building a Cloud Spy Camera

We are now going to build a very famous project for secret agents: a spy camera. This will be a camera that can be, for example, installed in a room behind a set of books and help you in monitoring the room from another location.

We are going to build two projects based on the same hardware. These will be the key topics covered in this chapter:

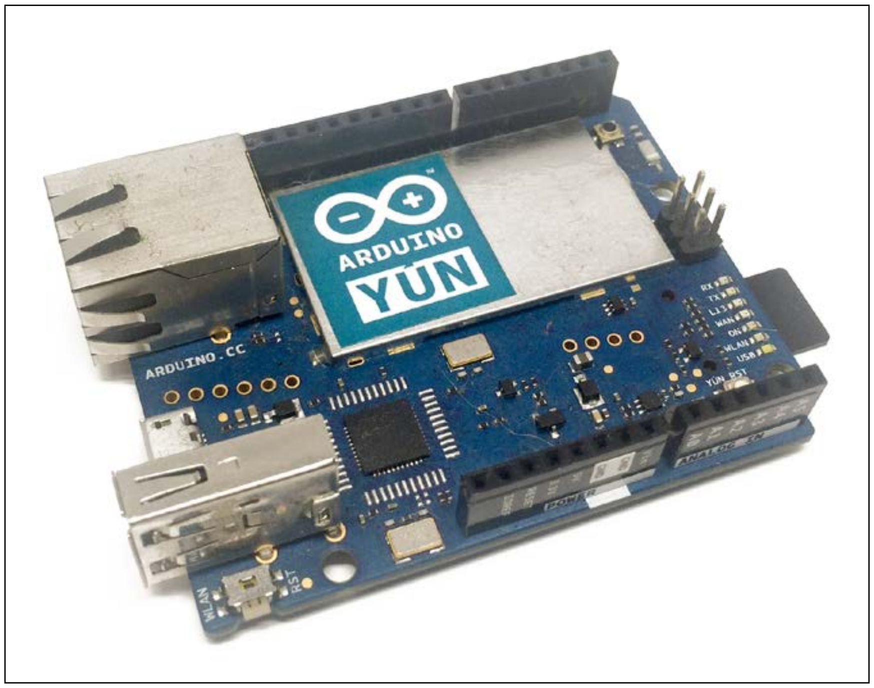
- The first project will be a spy camera that takes a picture every time motion is detected in front of it and uploads it to your Dropbox account. The pictures will then be accessible to the spy from anywhere in the world.
- Finally, we'll end the chapter by making the camera stream live videos on a local Wi-Fi network. This will be perfect for a spy who wants to see what's happening in a room while being hidden in another room or outside. Let's dive in!

Hardware and software requirements

First, let's see what are the required components for this project.

For once, we are not going to use an Arduino Uno board, but an Arduino Yun. Not only do we need Wi-Fi connectivity but also the on-board USB port of the Yun. This will make it really easy to use a USB camera with our project.

The following is the Arduino board that I used for this project:



Then, you need a USB camera. You need a camera that is compatible with the **USB Video Class** (**UVC**). Basically, most recent USB cameras are compatible with this standard. I recommend the Logitech C270 USB camera that I used for this project:

Finally, you will also need a PIR motion sensor, to detect whether there is motion in front of the camera. Any brand will be fine, you just need it be 5V-level compatible. This is the sensor that I used for this project:

Finally, here is the list of all the components that we will use in this project:

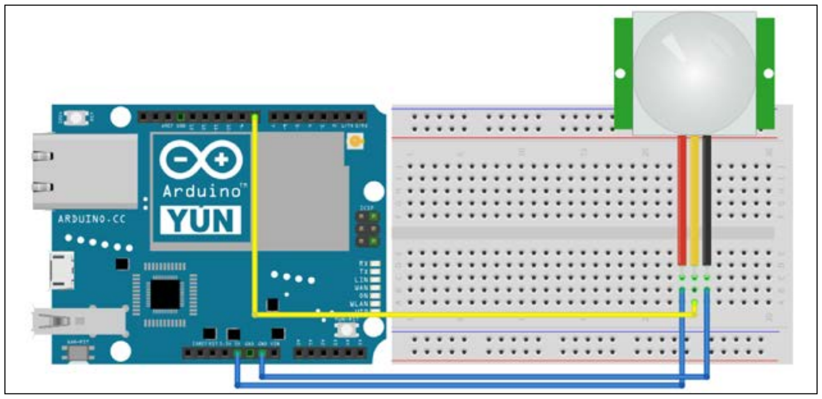
- Arduino Yun (`http://www.adafruit.com/product/1498`)
- USB camera (`http://www.logitech.com/en-us/product/hd-webcam-c270`)
- PIR motion sensor (`http://www.adafruit.com/product/189`)
- A microSD card (at least 2 GB)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

On the software side, you will only need the Arduino IDE. If you are using Windows, you'll also need a terminal software. I recommend using PuTTY, which you can download from `http://www.putty.org/`.

Hardware configuration

Now let's configure the hardware for this project. It will be really simple and we'll also set up the Wi-Fi connectivity of Yun.

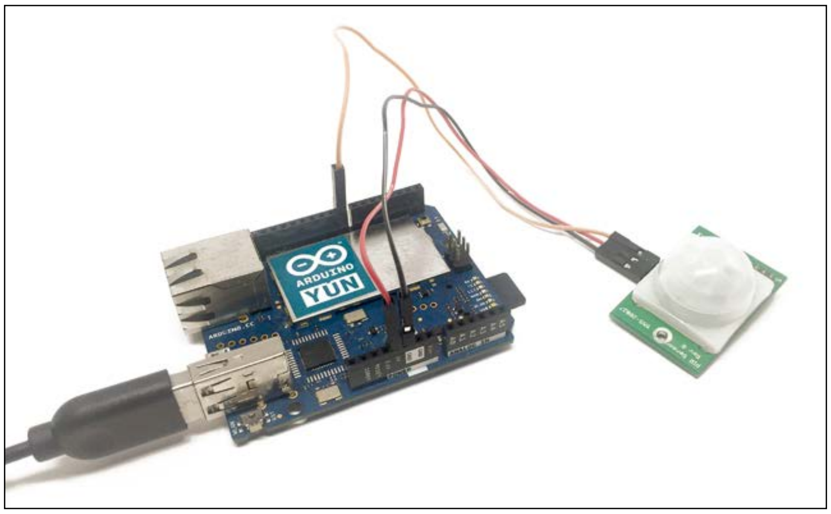
This is a schematic to help you out (excluding the USB camera):



You just need to connect the PIR motion sensor to Yun. Connect VCC to the Arduino 5V pin, GND to GND, and the output of the sensor to pin number 8 of the Arduino Yun.

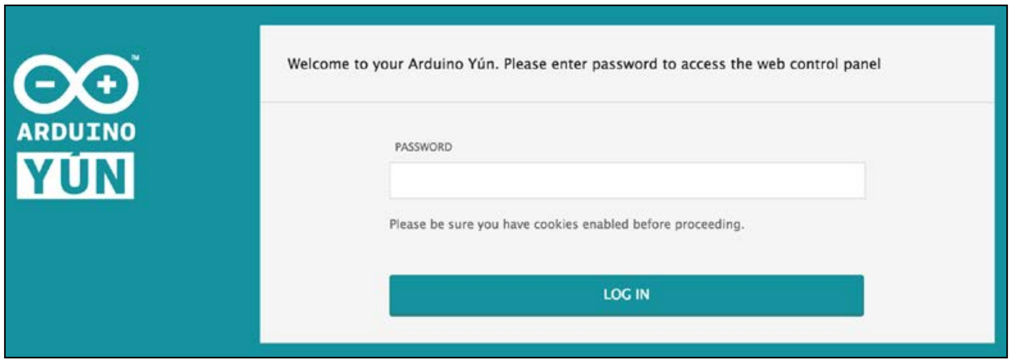
Finally, insert the USB camera into the USB port and the microSD card into the Arduino Yun.

The following is how it should look in the end:



Now, we are going to set up Yun so that it connects to your Wi-Fi network. For that, the best way is to follow the latest instructions from Arduino that are available at `https://www.arduino.cc/en/Guide/ArduinoYun`.

Then, you should be able to connect to your Yun via your favorite web browser and access it with the password that you set earlier:

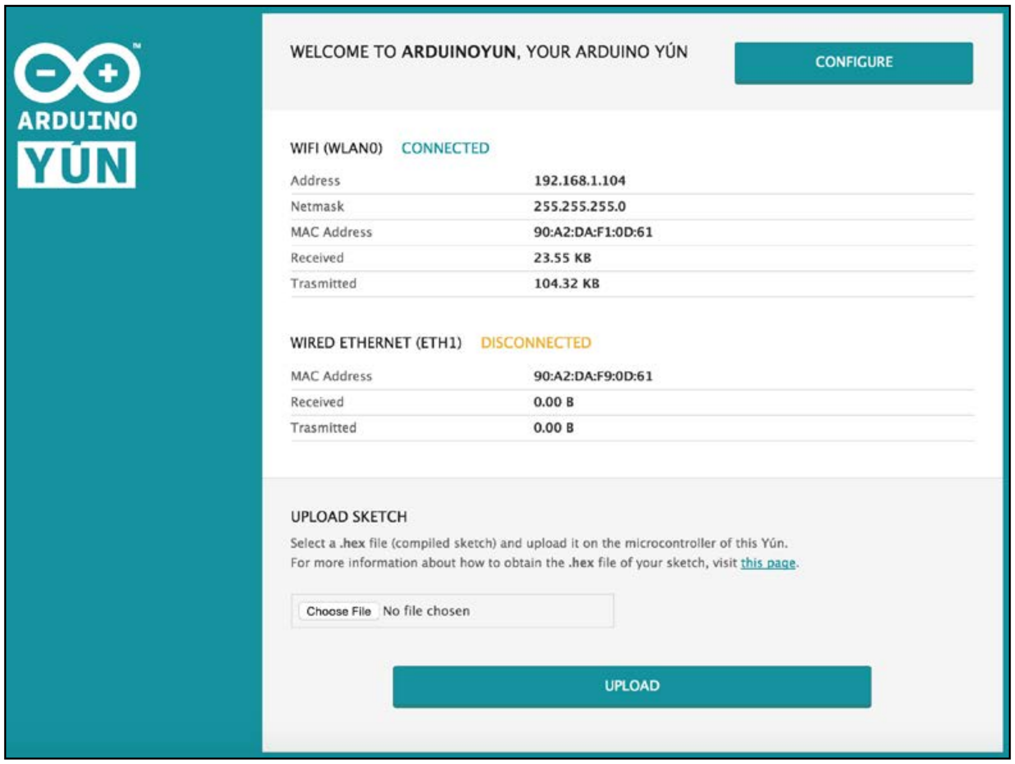


Welcome to your Arduino Yún. Please enter password to access the web control panel

PASSWORD

Please be sure you have cookies enabled before proceeding.

After that, you'll be able to see that your Yun is working:



WELCOME TO **ARDUINOYUN**, YOUR ARDUINO YÚN

WIFI (WLAN0) CONNECTED

Address	192.168.1.104
Netmask	255.255.255.0
MAC Address	90:A2:DA:F1:0D:61
Received	23.55 KB
Trasmitted	104.32 KB

WIRED ETHERNET (ETH1) DISCONNECTED

MAC Address	90:A2:DA:F9:0D:61
Received	0.00 B
Trasmitted	0.00 B

UPLOAD SKETCH

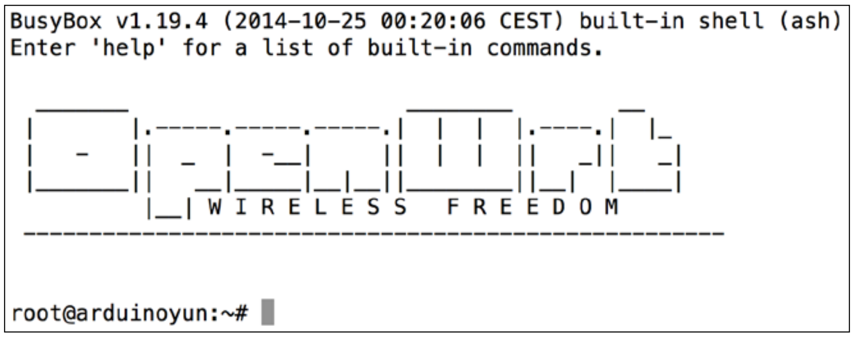
Select a **.hex** file (compiled sketch) and upload it on the microcontroller of this Yún.
For more information about how to obtain the **.hex** file of your sketch, visit this page.

Choose File No file chosen

Now, we'll access it from a terminal in order to install some modules. If you are on Windows, I recommend using PuTTY to type these commands. Start by connecting your Yun (replace the address with the one of your Yun) with:

```
ssh root@arduinoyun.local
```

You will then be greeted by the following message:



```
BusyBox v1.19.4 (2014-10-25 00:20:06 CEST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------


root@arduinoyun:~#
```

Now, type the following command:

```
opkg update
```

When this is done, type:

```
opkg install kmod-video-uvc
```

Then enter:

```
opkg install fswebcam
```

Finally, type the following command:

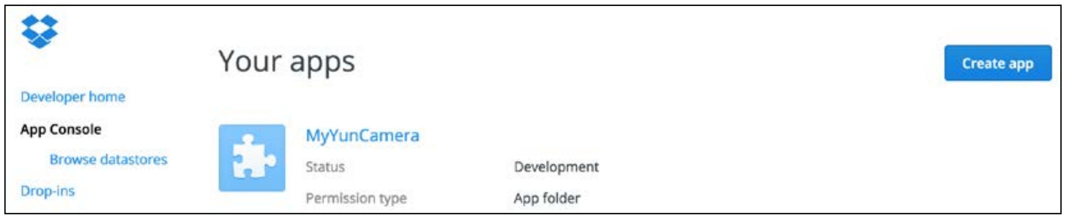
```
opkg install mjpg-streamer
```

Congratulations, your Yun is now fully operational for this project!

Setting up your Dropbox account

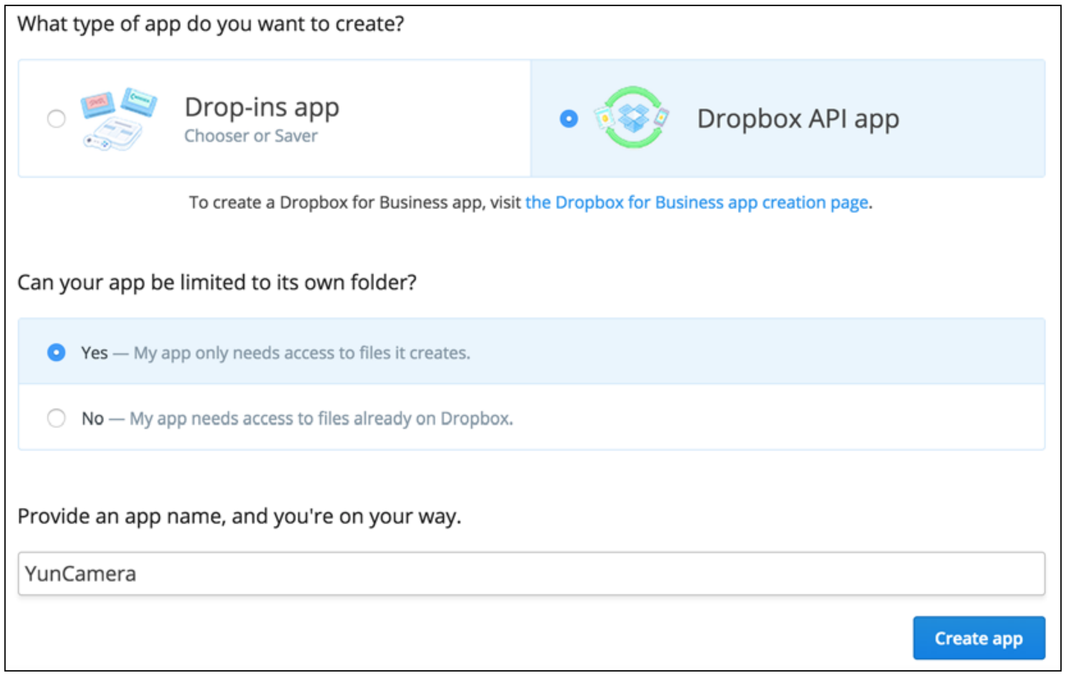
It's now time to set up your Dropbox account. First, make sure that you actually have an account by simply visiting the Dropbox website. Then, we'll need to create a Dropbox app. For this, go to `https://www.dropbox.com/developers/apps`.

Then, you can create a new app with the **Create app** button:



MyYunCamera
Status Development
Permission type App folder

Give it a name and make sure it is set similar to the following screenshot:



What type of app do you want to create?

Drop-ins app [✓ Dropbox API app]

To create a Dropbox for Business app, visit the Dropbox for Business app creation page.

Can your app be limited to its own folder?

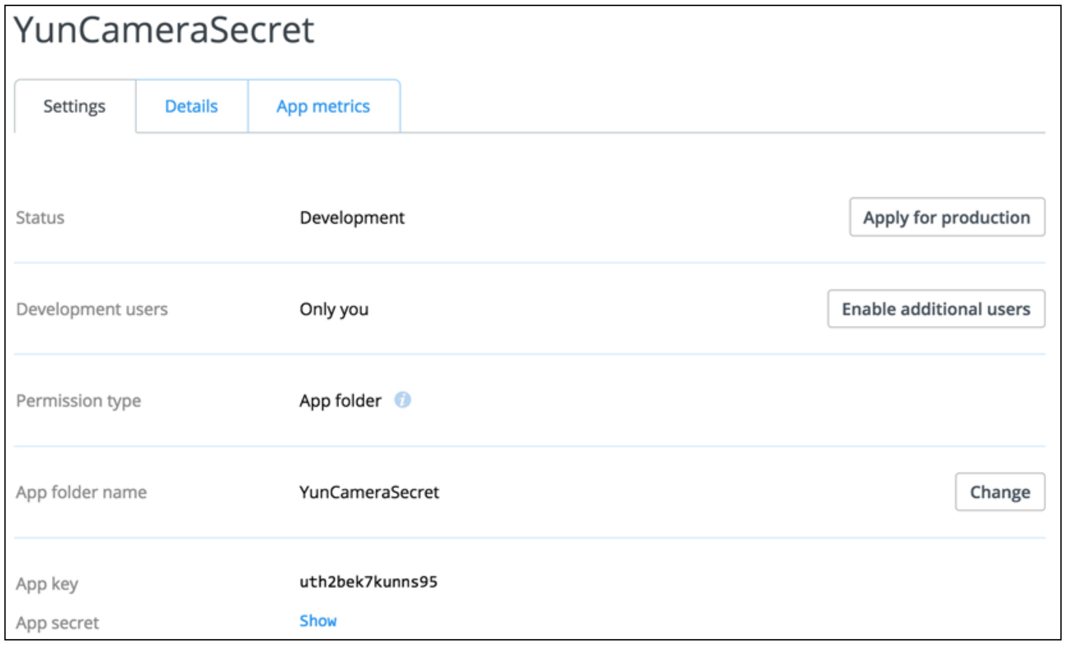
[✓ Yes — My app only needs access to files it creates.]

No — My app needs access to files already on Dropbox.

Provide an app name, and you're on your way.

YunCamera

Now, in the parameters of the app, there are two things that you need: **App key** and **App secret**. You can find them both on the same page:


YunCameraSecret

Status	Development	Apply for production
Development users	Only you	Enable additional users
Permission type	App folder	
App folder name	YunCameraSecret	Change
App key	uth2bek7kunns95	
App secret	Show	

Once you have these, you can move to the next step and configure your Temboo account.

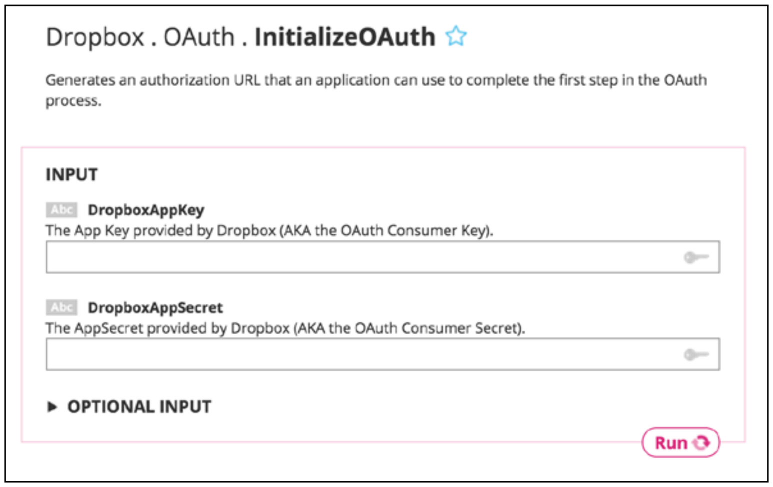
Setting up your Temboo account

We are going to use the Temboo service to link our hardware to the Dropbox app that we just created. This will allow us to upload files to Dropbox.

You first need to set up a new Temboo account from the following URL: `https://www.temboo.com/library/`.

Then, we need to actually authorize our Temboo account (and therefore, our Arduino project) in order to use your Dropbox app. For this, go to `https://www.temboo.com/library/Library/Dropbox/OAuth/InitializeOAuth/`.

You will be asked to enter your Dropbox App key and App secret:



Dropbox . OAuth . **InitializeOAuth**

Generates an authorization URL that an application can use to complete the first step in the OAuth process.

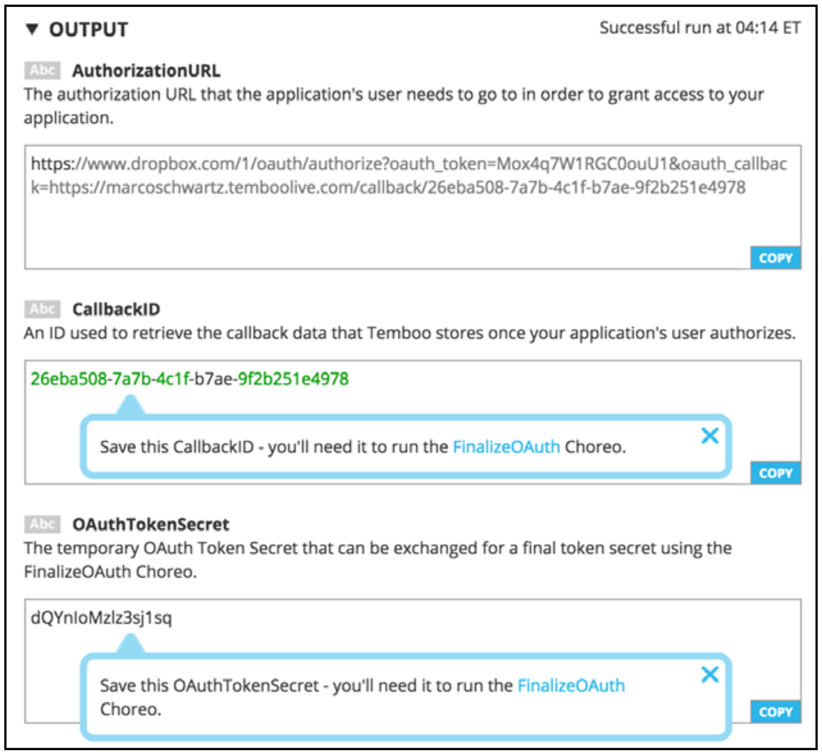
INPUT

Abc **DropboxAppKey**
The App Key provided by Dropbox (AKA the OAuth Consumer Key).

Abc **DropboxAppSecret**
The AppSecret provided by Dropbox (AKA the OAuth Consumer Secret).

▶ OPTIONAL INPUT

Once you click on **Run**, there are two things that you'll need to do. First, you need to follow the link that is given to you by Temboo:



▼ OUTPUT — Successful run at 04:14 ET

Abc **AuthorizationURL**
The authorization URL that the application's user needs to go to in order to grant access to your application.

```
https://www.dropbox.com/1/oauth/authorize?oauth_token=Mox4q7W1RGC0ouU1&oauth_callback=https://marcoschwartz.temboolive.com/callback/26eba508-7a7b-4c1f-b7ae-9f2b251e4978
```

Abc **CallbackID**
An ID used to retrieve the callback data that Temboo stores once your application's user authorizes.

```
26eba508-7a7b-4c1f-b7ae-9f2b251e4978
```

Save this CallbackID - you'll need it to run the FinalizeOAuth Choreo.

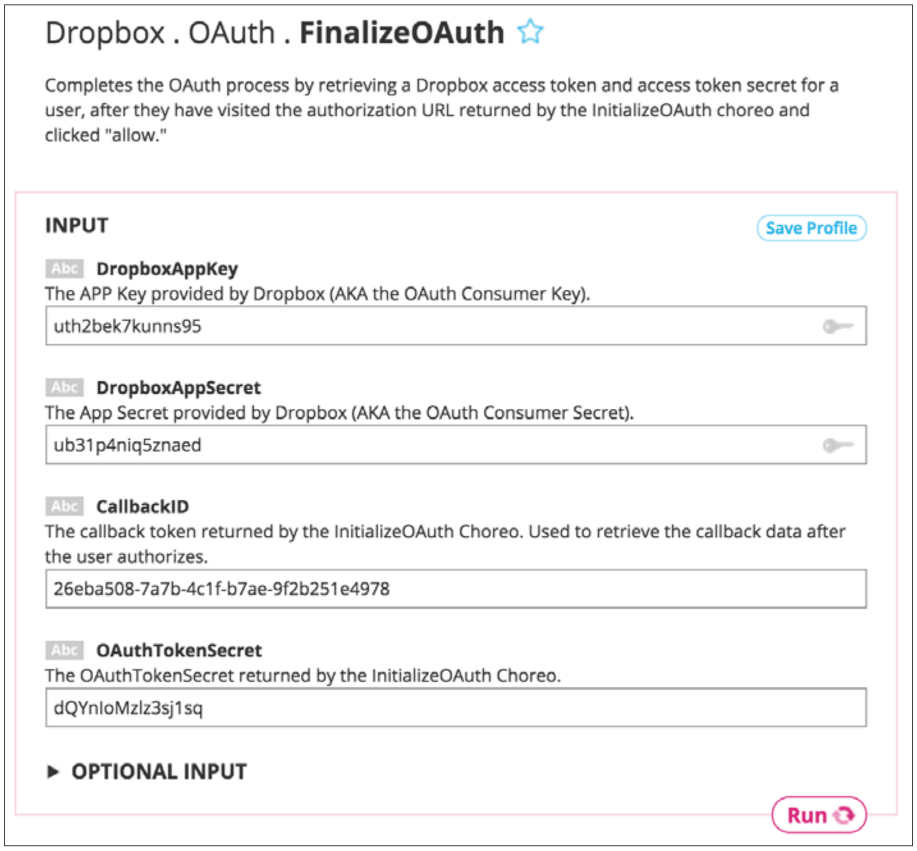
Abc **OAuthTokenSecret**
The temporary OAuth Token Secret that can be exchanged for a final token secret using the FinalizeOAuth Choreo.

```
dQYnloMzlz3sj1sq
```

Save this OAuthTokenSecret - you'll need it to run the FinalizeOAuth Choreo.

After that, you'll need to grab **CallbackID** and **OAuthTokenSecret** and go to the page at `https://www.temboo.com/library/Library/Dropbox/OAuth/FinalizeOAuth/`.

On this page, you can enter all the information that you have received so far:



Dropbox . OAuth . **FinalizeOAuth**

Completes the OAuth process by retrieving a Dropbox access token and access token secret for a user, after they have visited the authorization URL returned by the InitializeOAuth choreo and clicked "allow."

INPUT

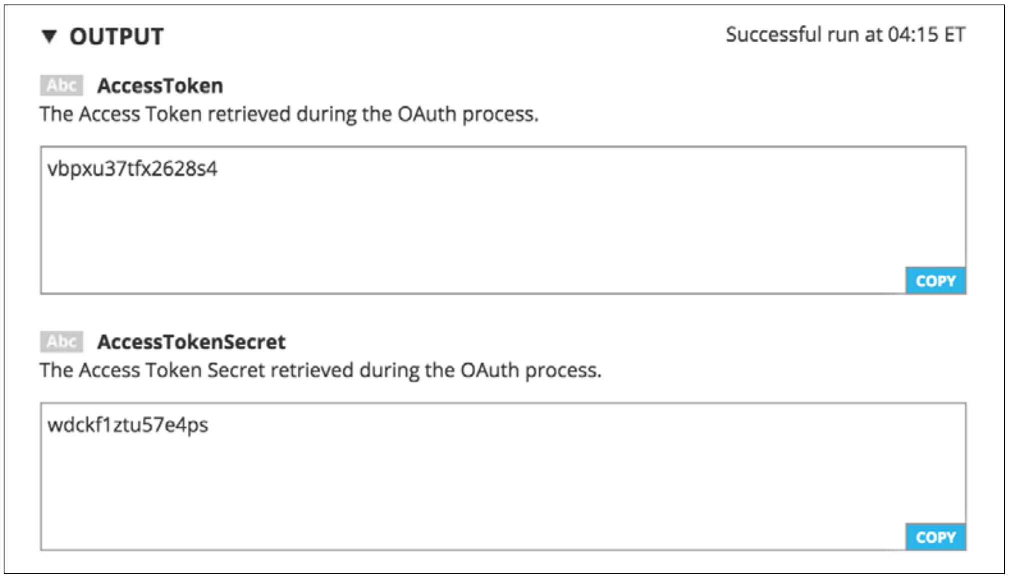
DropboxAppKey
The APP Key provided by Dropbox (AKA the OAuth Consumer Key).
uth2bek7kunns95

DropboxAppSecret
The App Secret provided by Dropbox (AKA the OAuth Consumer Secret).
ub31p4niq5znaed

CallbackID
The callback token returned by the InitializeOAuth Choreo. Used to retrieve the callback data after the user authorizes.
26eba508-7a7b-4c1f-b7ae-9f2b251e4978

OAuthTokenSecret
The OAuthTokenSecret returned by the InitializeOAuth Choreo.
dQYnIoMzlz3sj1sq

You'll be then given access token and token secret, which you will need for the spy camera project:



Successful run at 04:15 ET

AccessToken
The Access Token retrieved during the OAuth process.

```
vbpxu37tfx2628s4
```

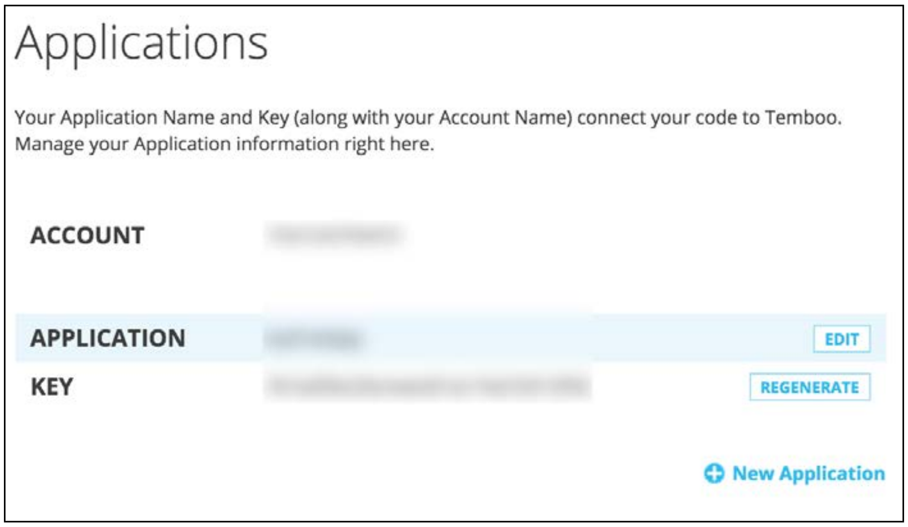
AccessTokenSecret
The Access Token Secret retrieved during the OAuth process.

```
wdckf1ztu57e4ps
```

Note that the Dropbox API is subject to change in the future. Therefore, always check the Temboo page and follow the instructions given there if they are different from the ones presented in this book.

You just need one more thing from Temboo; some data about your account. Go to `https://www.temboo.com/account/applications/`.

There you can see the information about the **Application** that you created when opening your account:


Applications

Your Application Name and Key (along with your Account Name) connect your code to Temboo. Manage your Application information right here.

ACCOUNT

APPLICATION

KEY

Keep this open, you'll also need this information later.

Saving pictures to Dropbox

Finally, we are going to make our first application using the hardware that we built. There will be two parts here: an Arduino sketch and a Python script. The Arduino sketch will be in charge of taking a picture in case motion is detected and call the Python script. The Python script will actually upload the pictures to Temboo every time it is called by the Arduino sketch.

This is the complete Arduino sketch:

```
// Sketch to upload pictures to Dropbox when motion is detected
#include <Bridge.h>
#include <Process.h>

// Picture process
Process picture;

// Filename
String filename;

// Pin
```

```
int pir_pin = 8;

// Path
String path = "/mnt/sda1/";

void setup() {

  // Bridge
  Bridge.begin();

  // Set pin mode
  pinMode(pir_pin,INPUT);
}

void loop(void)
{

  if (digitalRead(pir_pin) == true) {

    // Generate filename with timestamp
    filename = "";
    picture.runShellCommand("date +%s");
    while(picture.running());

    while (picture.available()>0) {
      char c = picture.read();
      filename += c;
    }
    filename.trim();
    filename += ".png";

    // Take picture
    picture.runShellCommand("fswebcam " + path + filename + " -r 
1280x720");
    while(picture.running());

    // Upload to Dropbox
    picture.runShellCommand("python " + path + "upload_picture.py " + 
path + filename);
    while(picture.running());
  }
}
```

Let's see what the most important parts of this sketch are. First, you need to include the required libraries:

```
#include <Bridge.h>
#include <Process.h>
```

Then, we will define the path to the SD card, which is where the pictures will be stored:

```
String path = "/mnt/sda1/";
```

After that, we will initialize the `Bridge` instance that will allow us to use the Yun filesystem, for example:

```
Bridge.begin();
```

Still in the `setup()` function of the sketch, we will set the motion sensor pin as an input:

```
pinMode(pir_pin,INPUT);
```

After that, in the `loop()` function of the sketch, we will check whether the motion sensor detected motion:

```
if (digitalRead(pir_pin) == true) {
```

If that's the case, we will first build a filename for the new picture using the current date and time:

```
filename = "";
   picture.runShellCommand("date +%s");
while(picture.running());

while (picture.available()>0) {
     char c = picture.read();
  filename += c;
}
filename.trim();
filename += ".png";
```

Then, we will use the `fswebcam` utility to save this picture on the SD card:

```
picture.runShellCommand("fswebcam " + path + filename + " -r 
1280x720");
while(picture.running());
```

And finally, we will call the Python script to actually upload the picture to Dropbox:

```
picture.runShellCommand("python " + path + "upload_picture.py " + path
+ filename);
while(picture.running());
```

Now, let's see the Python script. The following is the complete script:

```
# coding=utf-8
# Script to upload files to Dropbox

# Import correct libraries
import base64
import sys
from temboo.core.session import TembooSession
from temboo.Library.Dropbox.FilesAndMetadata import UploadFile

print str(sys.argv[1])

# Encode image
with open(str(sys.argv[1]), "rb") as image_file:
    encoded_string = base64.b64encode(image_file.read())

# Declare Temboo session and Choreo to upload files
session = TembooSession('yourSession', 'yourApp', 'yourKey')
uploadFileChoreo = UploadFile(session)

# Get an InputSet object for the choreo
uploadFileInputs = uploadFileChoreo.new_input_set()

# Set inputs
uploadFileInputs.set_AppSecret("yourAppSecret")
uploadFileInputs.set_AccessToken("yourAccessToken")
uploadFileInputs.set_FileName(str(sys.argv[1]))
uploadFileInputs.set_AccessTokenSecret("yourTokenSecret")
uploadFileInputs.set_AppKey("yourAppKey")
uploadFileInputs.set_FileContents(encoded_string)
uploadFileInputs.set_Root("sandbox")

# Execute choreo
uploadFileResults = uploadFileChoreo.execute_with_
results(uploadFileInputs)
```

Now, let's see the most important parts of this script. We will first import the Temboo libraries:

```
import base64
import sys
from temboo.core.session import TembooSession
from temboo.Library.Dropbox.FilesAndMetadata import UploadFile
```

You will also need to set up your Temboo account details:

```
session = TembooSession('yourSession', 'yourApp', 'yourKey')
uploadFileChoreo = UploadFile(session)
```

Then, we will create a new set of inputs for the Dropbox library:

```
uploadFileInputs = uploadFileChoreo.new_input_set()
```

After that, this is where you will need to enter all the keys that we got from Dropbox and Temboo:

```
uploadFileInputs.set_AppSecret("yourAppSecret")
uploadFileInputs.set_AccessToken("yourAccessToken")
uploadFileInputs.set_FileName(str(sys.argv[1]))
uploadFileInputs.set_AccessTokenSecret("yourTokenSecret")
uploadFileInputs.set_AppKey("yourAppKey")
uploadFileInputs.set_FileContents(encoded_string)
uploadFileInputs.set_Root("sandbox")
```

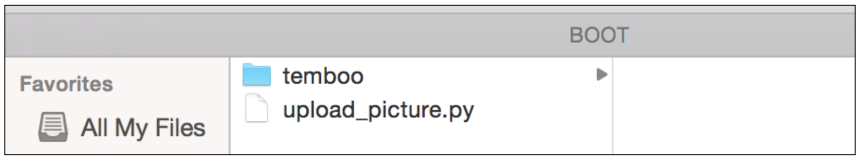
Finally, we will execute the uploading of the file to Dropbox:

```
uploadFileResults = uploadFileChoreo.execute_with_
results(uploadFileInputs)
```

It's now the time to test the project! Note that you can find all the code from the GitHub repository at `https://github.com/marcoschwartz/arduino-secret-agents`.

You will still need to download the Temboo Python library from `https://temboo.com/sdk/python`.

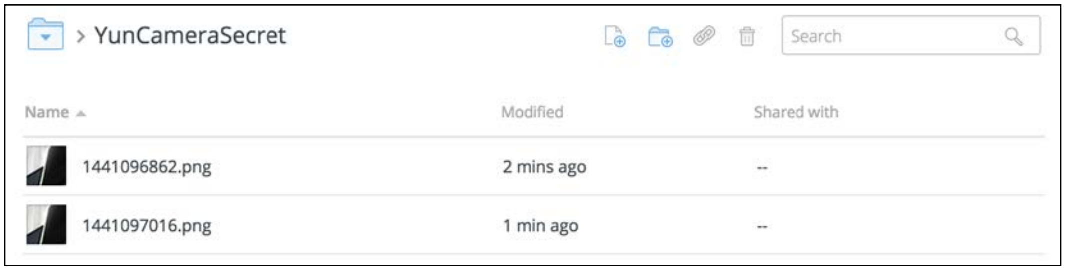
Then, make sure that you modified the Python files with your own data. Also, put the SD card in your computer again using an adapter. Place the file and the `temboo` SDK on the SD card, as shown in the following screenshot:



After that, place the SD card back in your Yun. Open the Arduino sketch with the Arduino IDE and make sure that you have selected the Arduino Yun board. Now, upload the sketch to the board.

Now, try to move your hand in front of the motion sensor; the sensor should go red and you should also notice that the camera becomes active immediately (there is a little green LED on the Logitech C270 camera).

You can also check your Dropbox account in the `Applications` folder. There should be a new folder that has been created, which contains the pictures taken by the spy camera:



Name	Modified	Shared with
1441096862.png	2 mins ago	--
1441097016.png	1 min ago	--

Congratulations, you now built your first spy camera project! Note that this folder can, of course, be accessed from anywhere in the world so even if you are on the other side of the town, you can monitor what's going on in the room where the camera is sitting.

Live streaming from the spy camera

We are now going to end this chapter with a shorter project: using the camera to stream live video in a web browser. This stream will be accessible from any device connected to the same Wi-Fi network as Yun.

To start with this project, log in to your Yun using the following command (by changing the name of the board with the name of your Yun):

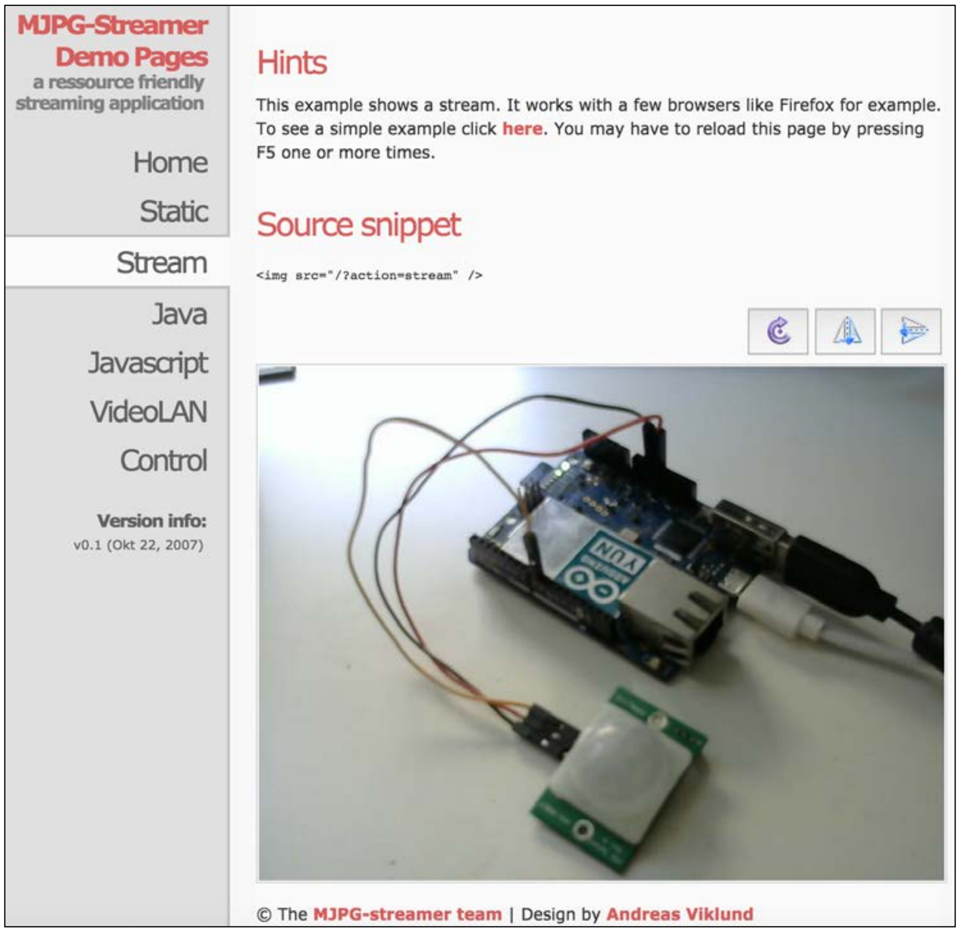
```
ssh root@arduinoyun.local
```

Then, type the following:

```
mjpg_streamer -i "input_uvc.so -d /dev/video0 -r 640x480 -f 25" -o
"output_http.so -p 8080 -w /www/webcam" &
```

This will start the streaming from your Yun. You can now simply go the URL of your Yun, and add `:8080` at the end, for example, `http://arduinoyun.local:8080`.

You will arrive at the streaming interface:



Hints

This example shows a stream. It works with a few browsers like Firefox for example. To see a simple example click here. You may have to reload this page by pressing F5 one or more times.

Source snippet

```
<img src="/?action=stream" />
```

© The MJPG-streamer team | Design by Andreas Viklund

You can now stream this video live to your mobile phone or any other device in the same network. It's the perfect project to spy on a room while you are sitting outside, for example.

Summary

In this project, we built a spy camera project that can send pictures to the Cloud whenever motion is detected. We also saw that it can do other things such as streaming live videos in a web browser.

There are, of course, many ways to improve this project. You can, for example, deploy several of these spy cameras in a building and make them take pictures that you identify in the Arduino code (when the filename is created).

In the next chapter, we are going to make another exciting application for secret agents: a project that will allow you to monitor secret data from anywhere in the world!

7
Monitoring Secret Data from Anywhere

In this chapter, we are going to build a project that will continuously record data from sensors, and send this data over Wi-Fi so it's accessible from any web browser. This is great for a secret agent that wants to monitor a room remotely, without being seen. You'll of course be able to adapt the project with your own sensors, depending on what you want to record.

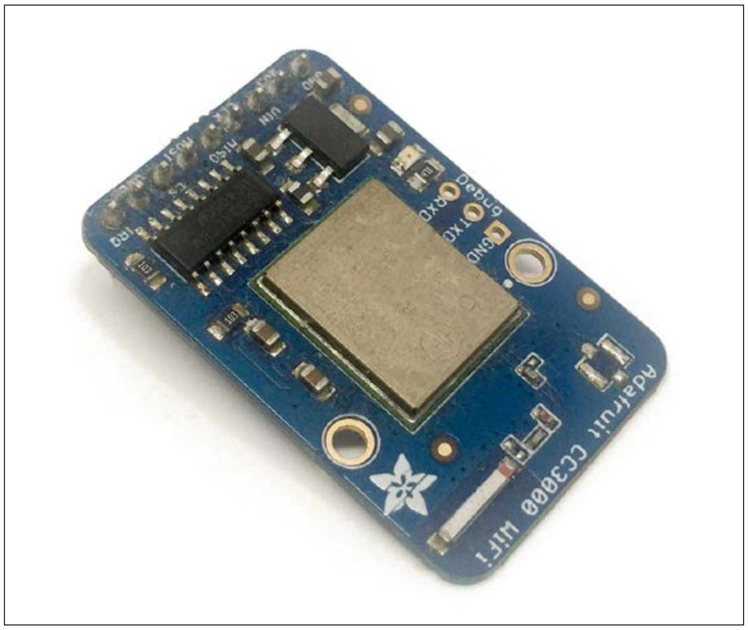
To do so, these are the steps we are going to take in this chapter:

- We will use Arduino along with the CC3000 Wi-Fi chip, which is quite convenient to give Wi-Fi connectivity to Arduino projects.
- We will send sensor data to an online service called dweet.io, and then display the result on a dashboard using Freeboard.io.
- Finally, we'll also see how to set up automated alerts based on the recorded data. Let's dive in!

Hardware and software requirements

First, let's see what the required components are for this project.

We'll of course use an Arduino Uno as the brain of the project. For Wi-Fi connectivity, we are going to use a CC3000 breakout board from Adafruit:



We'll also use a bunch of sensors to illustrate the behavior of the project: a DHT11 sensor for temperature and humidity, a photocell for light levels, and a motion sensor.

Finally, here is a list of all the components that we will use in this project:

- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- CC3000 breakout board (`http://www.adafruit.com/product/1469`)
- DHT11 sensor with 4.7k Ohm resistor (`http://www.adafruit.com/product/386`)
- Photocell (`https://www.sparkfun.com/products/9088`)
- 10k Ohm resistor (`https://www.sparkfun.com/products/8374`)
- PIR motion sensor (`http://www.adafruit.com/product/189`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

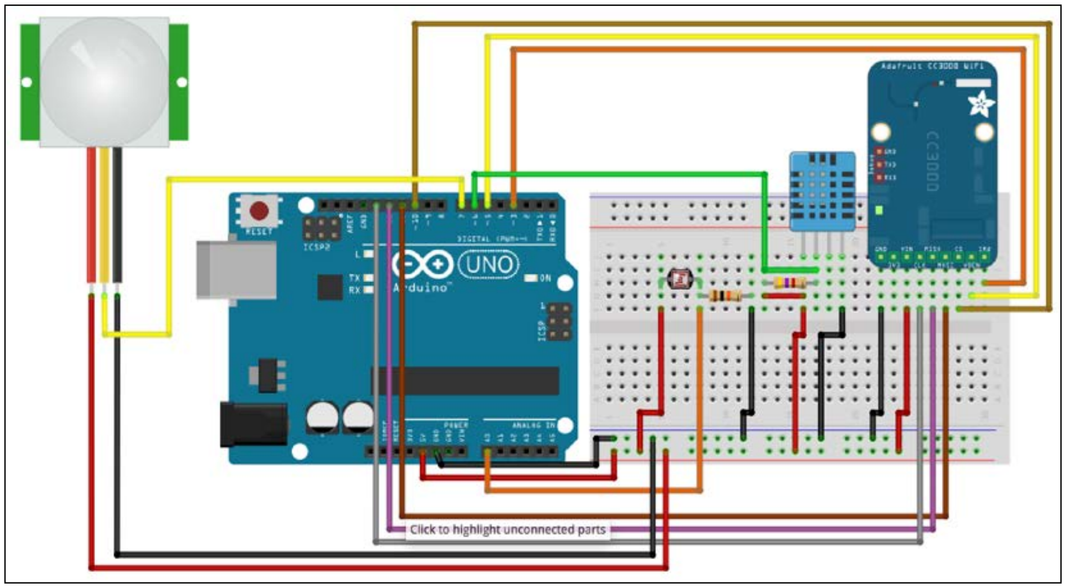
On the software side, you need the latest version of the Arduino IDE. You will also need the following libraries:

- Adafruit CC3000 library
- Adafruit DHT library

To install these libraries, just use the Arduino library manager.

Hardware configuration

Now let's assemble the different components of this project. This is a schematic to help you out:



First, connect the power. Connect the Arduino Uno 5V to the red power rail on the breadboard, and the GND pin to the blue power rail. Also, place all the main components on the breadboard.

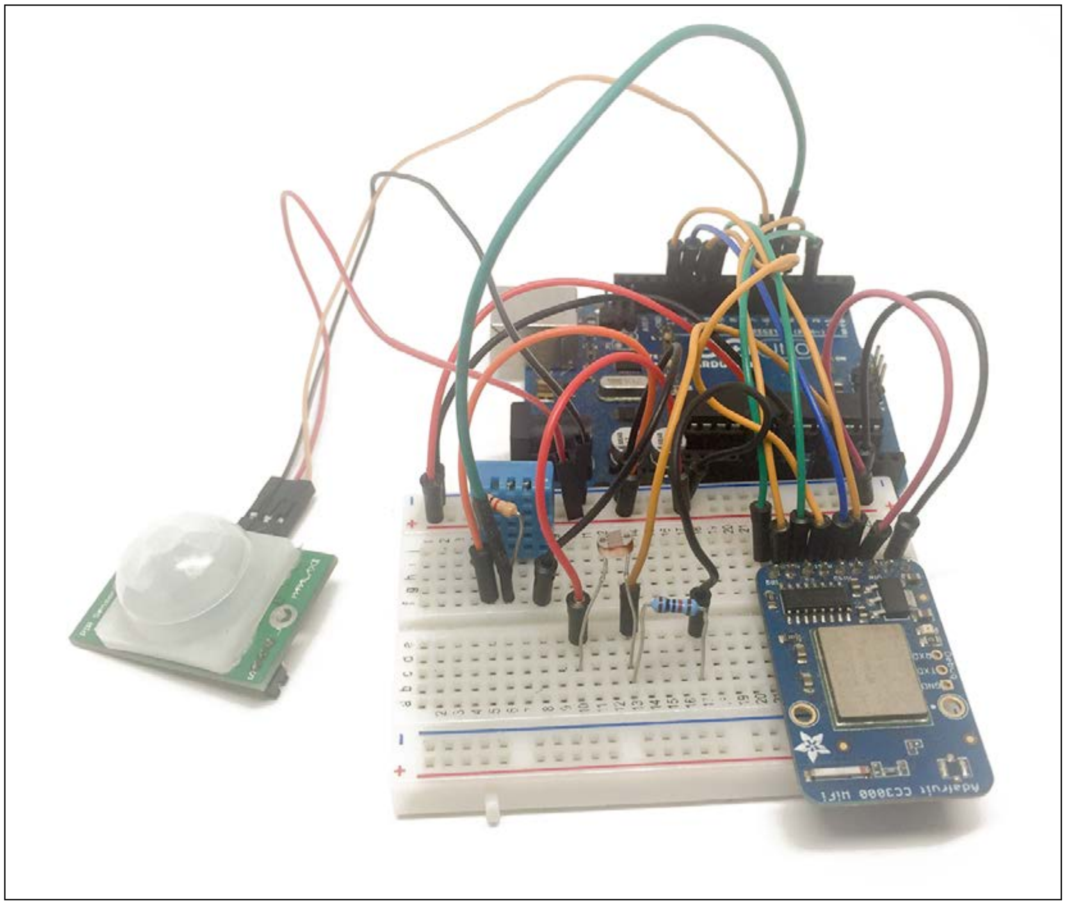
After that, for the DHT11 sensor, follow the instructions given by the schematic to connect the sensor to the Arduino board. Make sure you don't forget the 4.7k Ohm resistor between the VCC and signal pins.

We are now going to connect the photocell. Start by placing the photocell on the breadboard in series with the 10k Ohm resistor. After that, connect the other end of the photocell to the red power rail, and the other pin of the resistor to the blue power rail. Finally, connect the pin between the photocell and the resistor to Arduino Uno pin A0.

Finally, for the motion sensor, connect the VCC pin to the red power rail, the GND pin to the blue power rail, and finally the output pin of the sensor to Arduino pin 7.

Now, we are going to connect the CC3000 breakout board. Connect the pins as indicated on the schematic: IRQ to pin number 3 of the Arduino board, VBAT to pin 5, and CS to pin 10. After that, connect these pins to the Arduino board: MOSI, MISO, and CLK go to pins 11, 12, and 13, respectively. Finally, connect the power to the CC3000 breakout: connect 5V to the Vin pin of the breakout board, and GND to GND.

This is a picture of the completely assembled project:

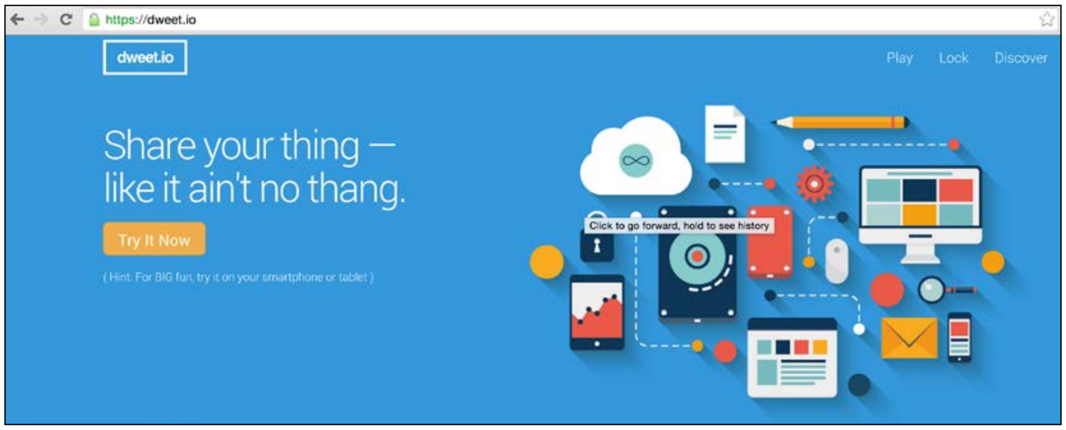


Congratulations, your project is now fully assembled! You can move on to the next part: sending data to the cloud.

Sending data to dweet.io

The first step in this project is really to send data to the web so it is stored online. For this, we'll use a service called dweet.io. You can check it out at `https://dweet.io/`.

This is the main welcome page:


https://dweet.io
dweet.io
Play
Lock
Discover
Share your thing — like it ain't no thang.
Try It Now
(Hint: For BIG fun, try it on your smartphone or tablet)
Click to go forward, hold to see history

This is the complete Arduino code for this project:

```
// Libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include "DHT.h"
#include <avr/wdt.h>

// Define CC3000 chip pins
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// DHT sensor
#define DHTPIN 6
#define DHTTYPE DHT11

// Create CC3000 instances
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS, ADAFRUIT_
CC3000_IRQ, ADAFRUIT_CC3000_VBAT,
                                          SPI_CLOCK_DIV2); // you can
change this clock speed

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

// WLAN parameters
#define WLAN_SSID       "yourWiFiSSID"
#define WLAN_PASS       "yourWiFiPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2

// Dweet parameters
#define thing_name  "mySecretThing"

// Variables to be sent
int temperature;
int humidity;
int light;
int motion;

uint32_t ip;

void setup(void)
{
  // Initialize
  Serial.begin(115200);

  Serial.println(F("\nInitializing..."));
  if (!cc3000.begin())
  {
    Serial.println(F("Couldn't begin()! Check your wiring?"));
    while(1);
  }

  // Connect to WiFi network
  Serial.print(F("Connecting to WiFi network ..."));
  cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY);
  Serial.println(F("done!"));

  /* Wait for DHCP to complete */
  Serial.println(F("Request DHCP"));
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }

  // Start watchdog
  wdt_enable(WDTO_8S);
}

void loop(void)
{

  // Measure from DHT
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  temperature = (int)t;
  humidity = (int)h;

  // Measure light level
  float sensor_reading = analogRead(A0);
  light = (int)(sensor_reading/1024*100);

  // Get motion sensor reading
  motion = digitalRead(7);
  Serial.println(F("Measurements done"));

  // Reset watchdog
  wdt_reset();

  // Get IP
  uint32_t ip = 0;
  Serial.print(F("www.dweet.io -> "));
  while  (ip  ==  0)  {
    if  (!  cc3000.getHostByName("www.dweet.io", &ip))  {
      Serial.println(F("Couldn't resolve!"));
    }
    delay(500);
  }
  cc3000.printIPdotsRev(ip);
  Serial.println(F(""));

  // Reset watchdog
  wdt_reset();

  // Check connection to WiFi
  Serial.print(F("Checking WiFi connection ..."));
  if(!cc3000.checkConnected()){while(1){}}
  Serial.println(F("done."));
  wdt_reset();

  // Send request
  Adafruit_CC3000_Client client = cc3000.connectTCP(ip, 80);
  if (client.connected()) {
    Serial.print(F("Sending request... "));

    client.fastrprint(F("GET /dweet/for/"));
    client.print(thing_name);
    client.fastrprint(F("?temperature="));
    client.print(temperature);
    client.fastrprint(F("&humidity="));
    client.print(humidity);
    client.fastrprint(F("&light="));
    client.print(light);
    client.fastrprint(F("&motion="));
    client.print(motion);
    client.fastrprintln(F(" HTTP/1.1"));

    client.fastrprintln(F("Host: dweet.io"));
    client.fastrprintln(F("Connection: close"));
    client.fastrprintln(F(""));

    Serial.println(F("done."));
  } else {
    Serial.println(F("Connection failed"));
    return;
  }

  // Reset watchdog
  wdt_reset();

  Serial.println(F("Reading answer..."));
  while (client.connected()) {
    while (client.available()) {
      char c = client.read();
      Serial.print(c);
    }
  }
  Serial.println(F(""));

  // Reset watchdog
```

```
    wdt_reset();

    // Close connection and disconnect
    client.close();
    Serial.println(F("Closing connection"));
    Serial.println(F(""));

    // Reset watchdog & disable
    wdt_reset();

  }
```

Now let's look at the most important parts of the code. First, we need to include the required libraries, such as the CC3000 library and the DHT library:

```
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include "DHT.h"
#include <avr/wdt.h>
```

Then, we define which pin the DHT11 sensor is connected to:

```
#define DHTPIN 6
#define DHTTYPE DHT11
```

You also need to enter your Wi-Fi name and password:

```
#define WLAN_SSID       "yourWiFiSSID"
#define WLAN_PASS       "yourWiFiPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

Then, you can define a name for your thing that is, the virtual object that will store the data online:

```
#define thing_name  "mySecretThing"
```

In the `setup()` function of the sketch, we initialize the CC3000 chip:

```
if (!cc3000.begin())
  {
    Serial.println(F("Couldn't begin()! Check your wiring?"));
    while(1);
  }
```

We also connect to the Wi-Fi network:

```
cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY);
```

Finally, we initialize the watchdog to `8` seconds. This will basically reset the Arduino automatically if we don't refresh it before this delay. It is basically here to prevent the project from getting stuck:

```
wdt_enable(WDTO_8S);
```

In the `loop()` function of the sketch, we first measure data from the DHT sensor:

```
float t = dht.readTemperature();
float h = dht.readHumidity();
temperature = (int)t;
humidity = (int)h;
```

After that, we measure the ambient light level:

```
float sensor_reading = analogRead(A0);
light = (int)(sensor_reading/1024*100);
```

And finally we get the status of the motion sensor:

```
motion = digitalRead(7);
```

Then, we try to get the IP address of the dweet.io website:

```
while  (ip  ==  0)  {
  if  (!  cc3000.getHostByName("www.dweet.io", &ip))  {
    Serial.println(F("Couldn't resolve!"));
  }
  delay(500);
}
```

Then, we connect the project to this IP address:

```
Adafruit_CC3000_Client client = cc3000.connectTCP(ip, 80);
```

We can now send the data, following the format given by dweet.io:

```
client.fastrprint(F("GET /dweet/for/"));
client.print(thing_name);
client.fastrprint(F("?temperature="));
client.print(temperature);
client.fastrprint(F("&humidity="));
client.print(humidity);
client.fastrprint(F("&light="));
client.print(light);
client.fastrprint(F("&motion="));
client.print(motion);
client.fastrprintln(F(" HTTP/1.1"));
```

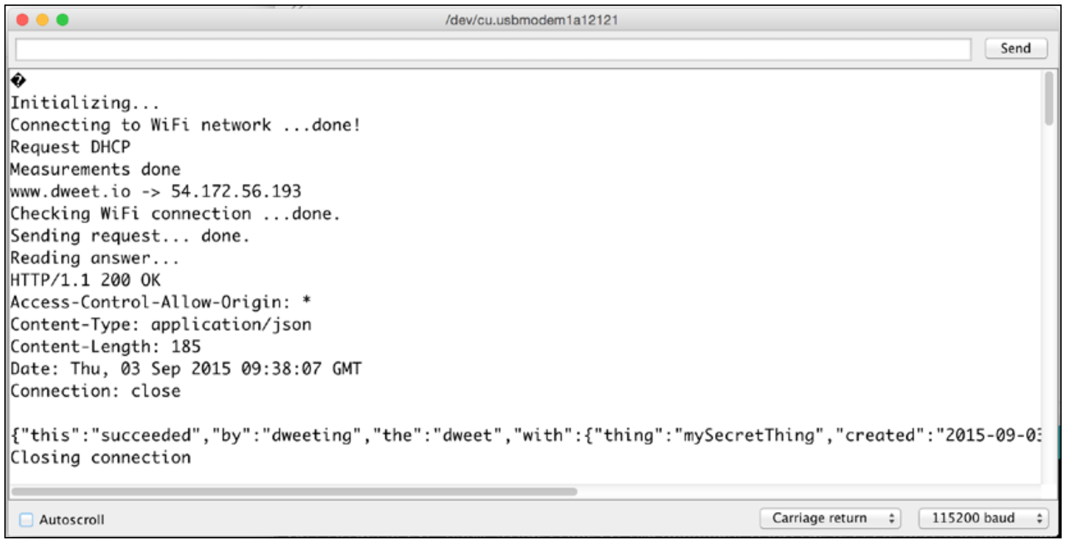
After that, we read the answer from the server:

```
while (client.connected()) {
    while (client.available()) {
      char c = client.read();
      Serial.print(c);
    }
  }
```

And we close the connection:

```
client.close();
```

It's now time to test the project! Make sure you grab all the code, copy it inside the IDE, and change the Wi-Fi details and thing name. Then, upload the code to the board and open the Serial monitor:

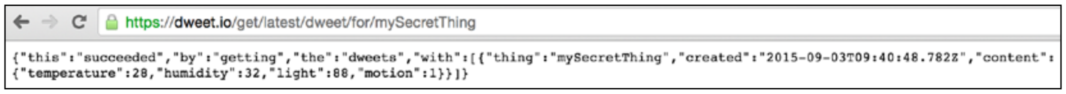


```
�
Initializing...
Connecting to WiFi network ...done!
Request DHCP
Measurements done
www.dweet.io -> 54.172.56.193
Checking WiFi connection ...done.
Sending request... done.
Reading answer...
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 185
Date: Thu, 03 Sep 2015 09:38:07 GMT
Connection: close

{"this":"succeeded","by":"dweeting","the":"dweet","with":{"thing":"mySecretThing","created":"2015-09-0
Closing connection
```

You should see that the project is sending measurements to dweet.io and then getting an answer. The most important part is the one indicating that data was recorded:

```
{"this":"succeeded","by":"dweeting","the":"dweet","with":{"thing":"myS
ecretThing","created":"2015-09-03T09:38:07.051Z","content":{"temperatu
re":28,"humidity":32,"light":87,"motion":0}}}
```

You can also check online to make sure data was recorded:



```
{"this":"succeeded","by":"getting","the":"dweets","with":[{"thing":"mySecretThing","created":"2015-09-03T09:40:48.782Z","content":
{"temperature":28,"humidity":32,"light":88,"motion":1}}]}
```

Now that we are sure that data is being recorded, we can move to the next step: spying on this data remotely from any web browser!

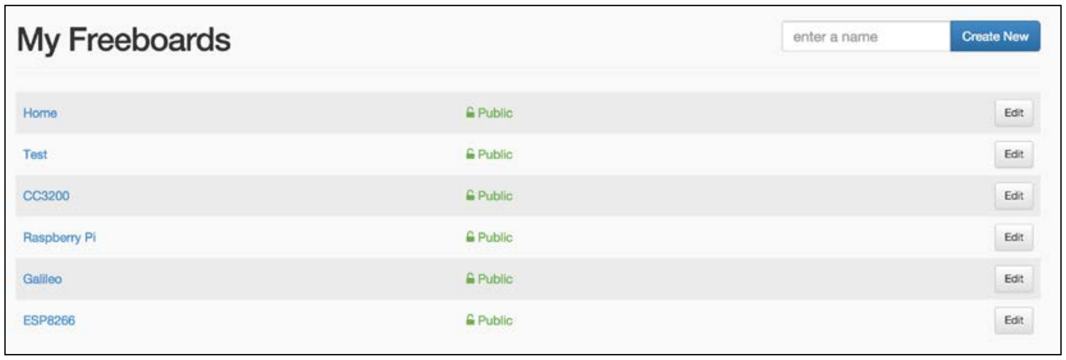
Monitoring the device remotely

We are now going to see how to access the data stored on dweet.io and display it graphically. For that, we are going to use a website called freeboard.io. You can go there with the following URL `http://freeboard.io/`.

This is the main welcome screen where you need to create an account:

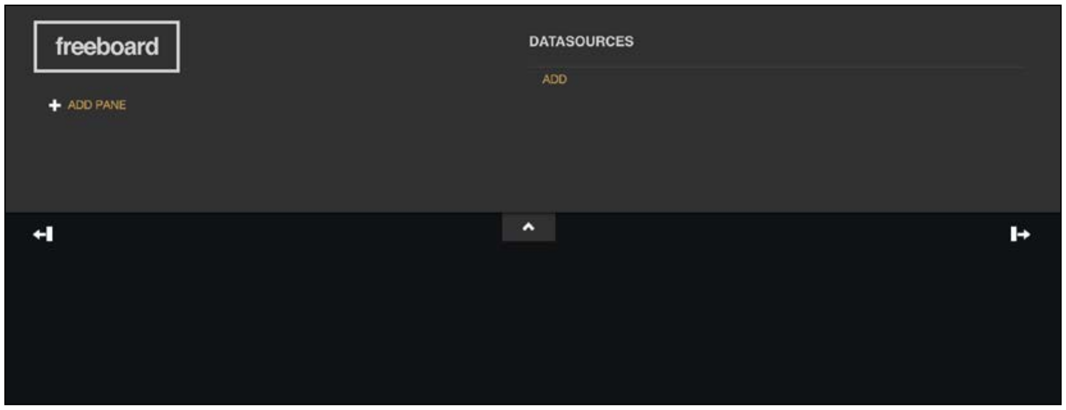
freeboard
Pricing
Login
Visualize the Internet of Things.
Ridiculously simple dashboards for your devices.
START NOW

Once you have an account, you can create a new board:

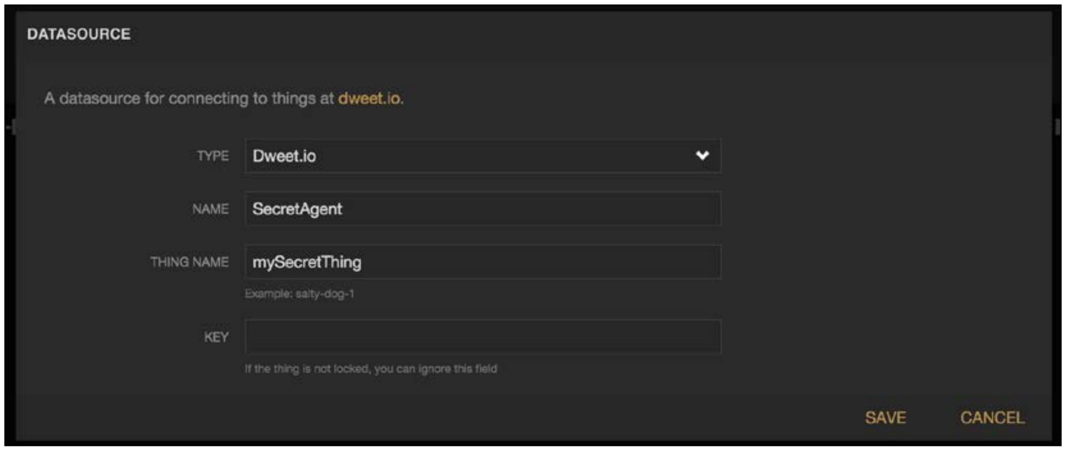


Home	Public	Edit
Test	Public	Edit
CC3200	Public	Edit
Raspberry Pi	Public	Edit
Galileo	Public	Edit
ESP8266	Public	Edit

Once this is done, you should be redirected to a similar page showing an empty board:



First, we need to set a datasource, meaning that we need to tell Freeboard to get data from the dweet thing we are storing the data in. Add a new datasource and fill out the fields as shown in the following screenshot, with the name of the thing that stores your data of course:


DATASOURCE

A datasource for connecting to things at dweet.io.

TYPE: Dweet.io

NAME: SecretAgent

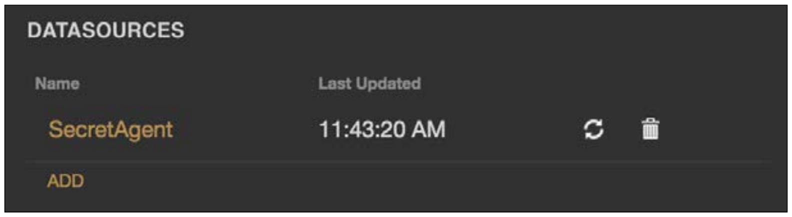
THING NAME: mySecretThing

Example: salty-dog-1

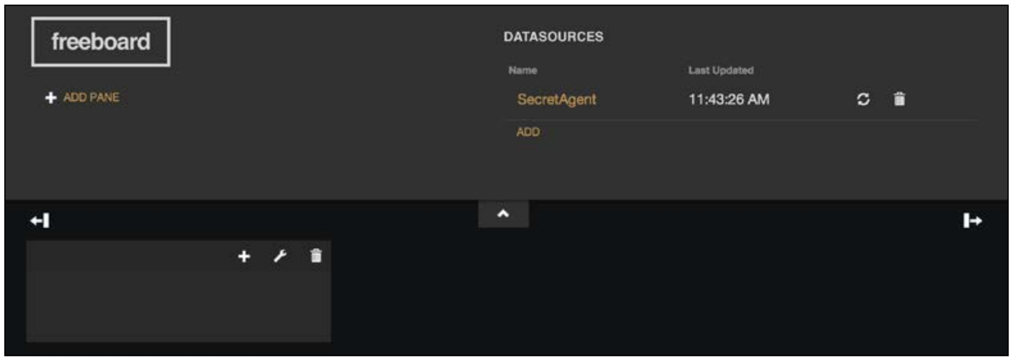
KEY:

If the thing is not locked, you can ignore this field

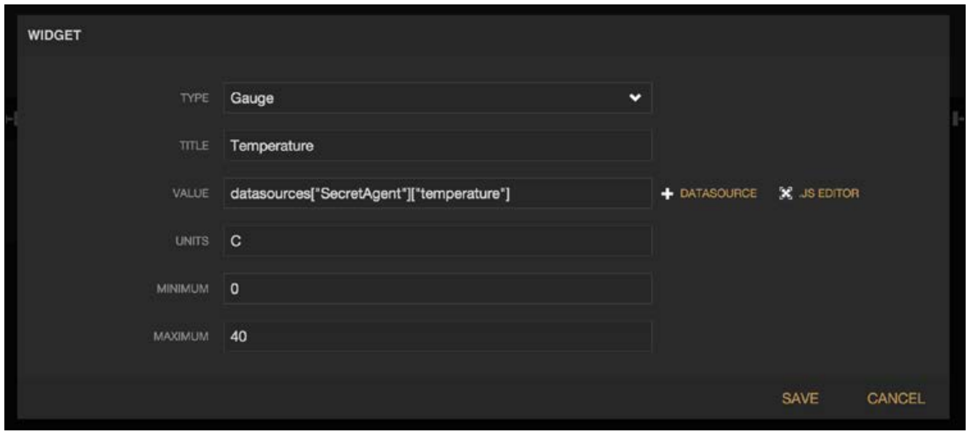
After that, you will see the datasource appearing at top of your board, with the last update date:



It's now time to add some graphical elements to our dashboard. We'll first add one for the temperature. Click on a new pane, which will create a new block inside the dashboard:

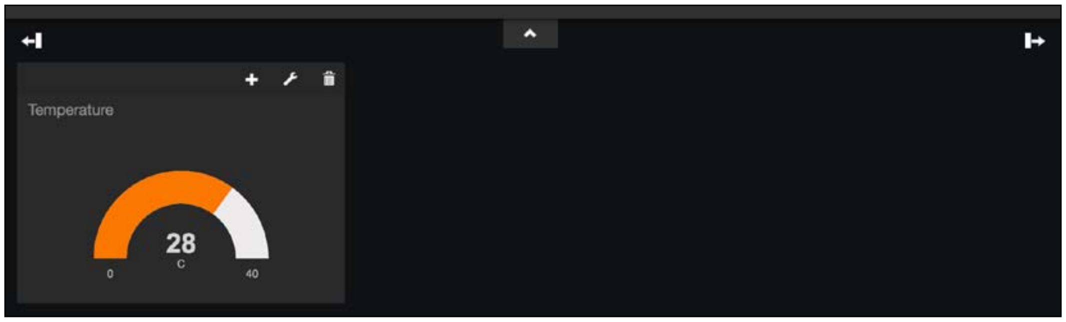


Then, click on the little **+** sign to create a new widget. Here, we are going to use a gauge widget for the temperature. Fill out the widget creation form as in the following screenshot:

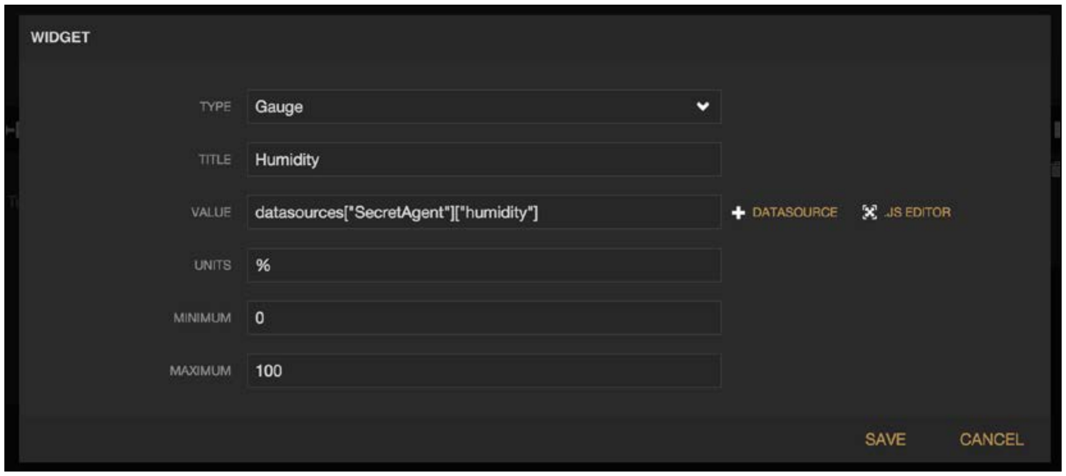

WIDGET

TYPE	Gauge
TITLE	Temperature
VALUE	datasources["SecretAgent"]["temperature"]
UNITS	C
MINIMUM	0
MAXIMUM	40

You should immediately see the gauge for the temperature on your board:



Now, let's do the same for humidity:


WIDGET

TYPE Gauge

TITLE Humidity

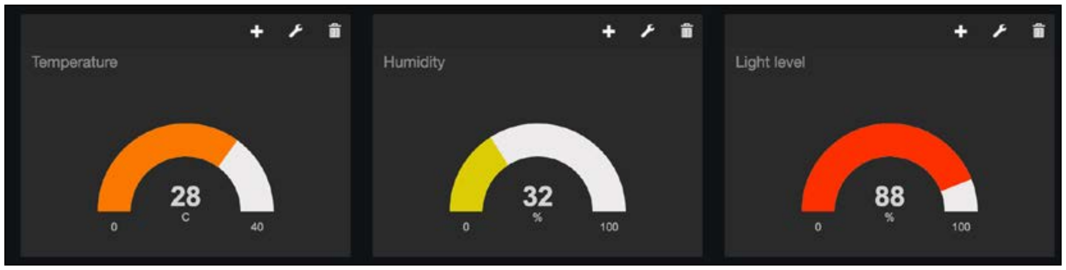
VALUE datasources["SecretAgent"]["humidity"]

UNITS %

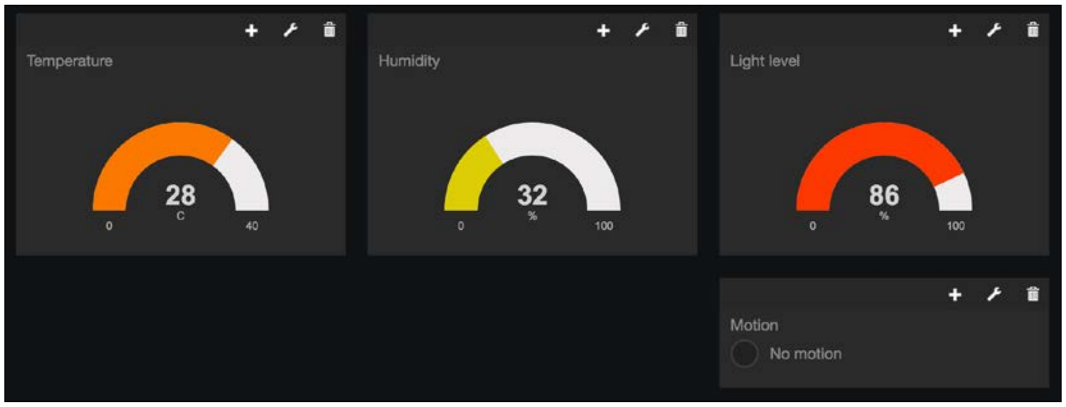
MINIMUM 0

MAXIMUM 100

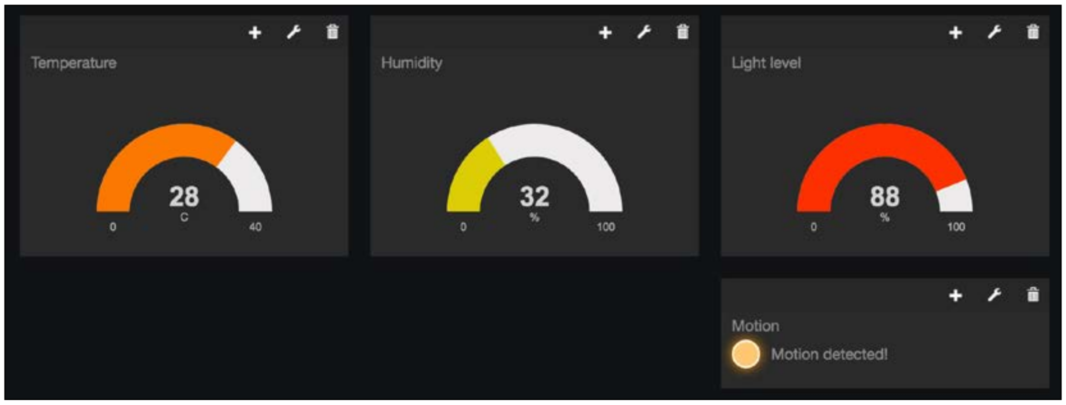
You can also do the same for the ambient light level. You now have all the data from these sensors refreshed in near real-time on your board:



The last thing we need to put in is the motion sensor. As it's an on/off sensor, I used an indicator widget for this:



Now try to pass your hand in front of the sensor. You should immediately see that the indicator changes its color:



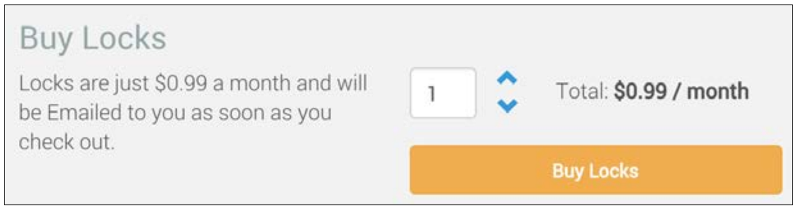
Congratulations, you now have a dashboard that you can access at any time to spy on this data!

Creating automated e-mail alerts

There is one more thing we can do with our project. Spying on the data is good, but we are not always behind a computer. For example, we would like to receive an alert via e-mail if motion is detected by the project.

Dweet.io proposes this kind of service, at a very cheap price (less than $1 a month). To do this, you need to make your device private with a lock. This is basically a key you can get from `https://dweet.io/locks`.

This is the screen where you can buy this key:



Buy Locks

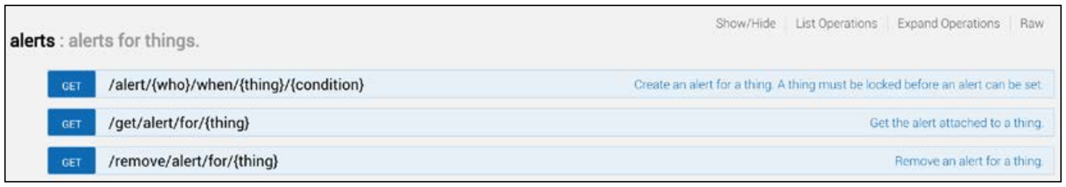
Locks are just $0.99 a month and will be Emailed to you as soon as you check out.

1

Total: $0.99 / month

Buy Locks

Once we have the key, we can actually set our alert. The dweet.io website explains this very well:



alerts : alerts for things.

Show/Hide | List Operations | Expand Operations | Raw

GET	/alert/{who}/when/{thing}/{condition}	Create an alert for a thing. A thing must be locked before an alert can be set.
GET	/get/alert/for/{thing}	Get the alert attached to a thing.
GET	/remove/alert/for/{thing}	Remove an alert for a thing.

To set up an alert, simply go to the following URL by modifying the different parameters with the desired parameters `https://dweet.io/alert/youremail@yourdomain.com/when/yourThing/dweet.motion==1?key=yourKey`.

Once that's done, you will automatically receive an alert whenever motion is detected by your project, even if you are not actually watching the data!

Summary

In this chapter, we built a project that can just be put in a room to spy on it from anywhere in the world. It continuously records data from the sensors connected to it, and sends this data directly to the cloud.

There are of course many things you can do to improve this project. You can, for example, experiment with your own sensors, depending on the data you want to spy on. You can also try to replace the CC3000 Wi-Fi chip with the GSM/GPRS shield we used in an earlier chapter to have a module you can place somewhere without having to connect it to a Wi-Fi network.

In the next chapter, we'll dive into a more advanced topic: actually building a GPS tracker based on Arduino!

8
Creating a GPS Tracker with Arduino

We are now going to build a very common tool that any secret agent should have: a GPS tracker. We'll use the Arduino platform again to build our own DIY GPS tracker. We will actually build two applications using the same hardware:

- The first one will be a location device that sends its position via SMS
- The other project will be a GPS tracker that you can actually locate live on a map

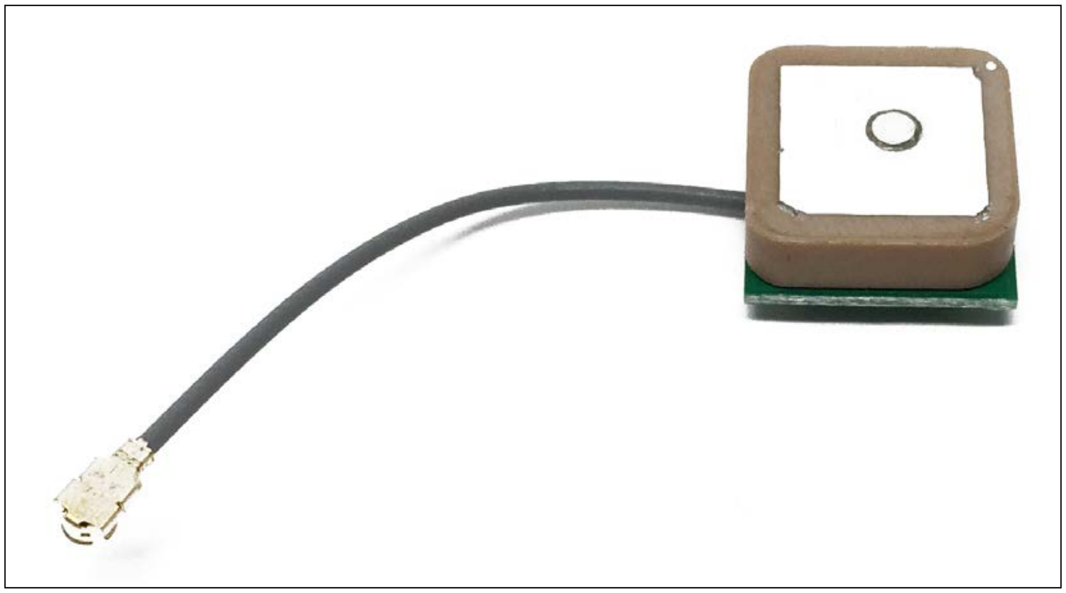
Let's dive in!

Hardware and software requirements

First let's see what the required components are for this project. As usual, we are going to use an Arduino Uno board as the central part of the project.

For the GPRS and GPS parts, we are going to use the Adafruit FONA 808 module again, which we already used in Module 3 *Chapter 5*, *Opening a Lock with an SMS*. We'll also need a GSM antenna for GSM/GPRS communications.

However, as here we want to use the onboard GPS, we'll also need an additional GPS antenna. I used a standard uFL passive GPS antenna from Adafruit:



Then, you will need a battery to power the FONA shield and the onboard GPS module, as the Arduino Uno board doesn't allow you to power the FONA shield (it can use up to 2A at a time!). For that, I used a 3.7V LiPo battery, along with a microUSB battery charger.

A very important part of the project is the SIM card, which you need to place inside the FONA shield. You will need a normal SIM card (not micro or nano), which is activated, not locked by a PIN, and able to receive text messages. You can get one at any of your local mobile network operators. Also, in this chapter you also need to have a data plan active with at least 1 MB of credit on the account.

Finally, here is a list of all the components that we will use in this project:

- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- Adafruit Fona 808 breakout (`http://www.adafruit.com/product/2542`)
- GSM uFL antenna (`http://www.adafruit.com/products/1991`)
- GSM SIM card
- 3.7V LiPo battery (`http://www.adafruit.com/products/328`)
- LiPo battery charger (`http://www.adafruit.com/products/1904`)
- Passive GPS antenna (`https://www.adafruit.com/product/2461`)

- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)

On the software side, you'll only need the latest version of the Arduino IDE, and the Adafruit FONA library. You can install this library using the Arduino IDE library manager.

Hardware configuration

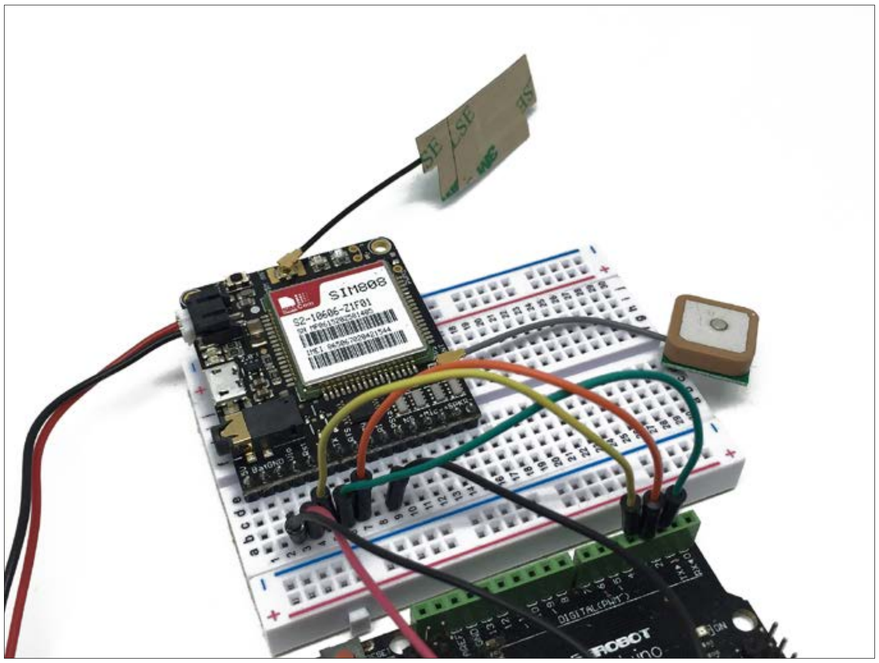
It's now time to assemble the hardware of this project.

First, connect the power supply to the breadboard: connect the 5V pin from the Arduino board to the red power line on the breadboard, and the GND pin to the blue power line.

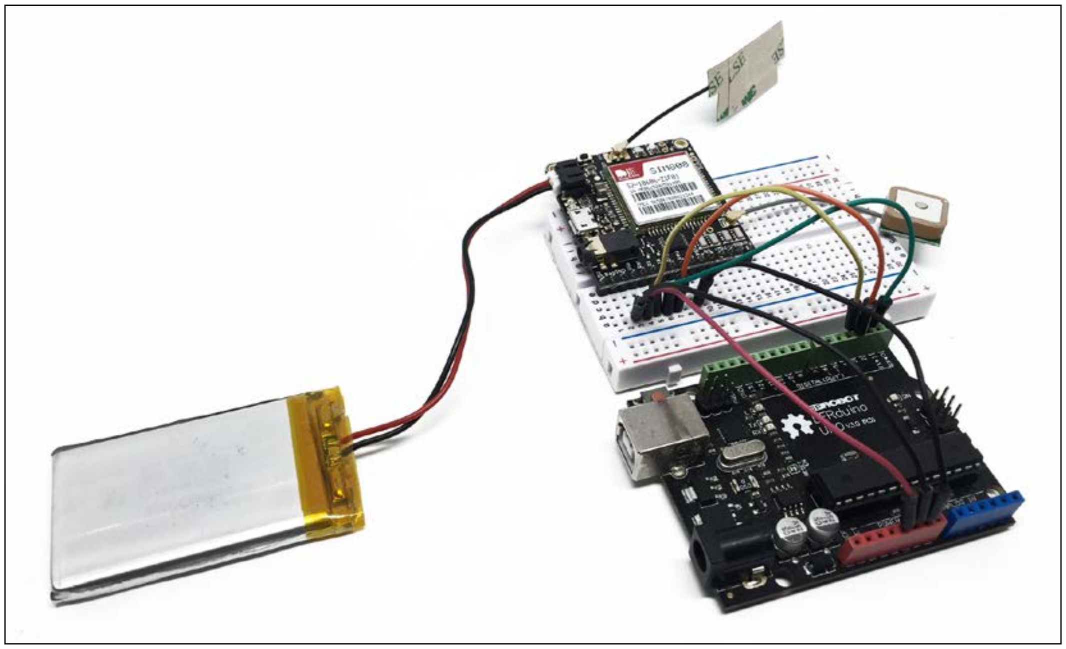
Then, place the FONA shield on the breadboard. Connect the VIO pin to the red power line, and the GND and Key pins to the blue power line.

After that, connect the RST pin to Arduino pin 4, TX to Arduino pin 3, and RX to Arduino pin 2. Also connect the 3.7V LiPo battery, the GPS antenna, and the GSM antenna to the FONA shield.

This is a close-up picture of the shield after the project was assembled:



And this is an overview of the whole project assembled:



Testing the location functions

Before we dive into the two exciting projects of this chapter, we'll first make a simple test using the FONA shield, and see whether it can actually locate our project on a map. The sketch will actually test whether the GPS location is working correctly.

If not, no worries: the sketch, and the other projects of this chapter, will automatically use GPRS location instead. It's less precise, but works quite well. This will be the case if you are testing this project inside, for example.

This is the complete code for this part:

```
// Libraries
#include "Adafruit_FONA.h"
#include <SoftwareSerial.h>

// Pins
#define FONA_RX 2
#define FONA_TX 3
```

```
#define FONA_RST 4

// Buffer
char replybuffer[255];

// Instances
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = &fonaSS;
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);

void setup() {

  // Init board
  while (!Serial);
  Serial.begin(115200);
  Serial.println(F("FONA location test"));
  Serial.println(F("Initializing....(May take 3 seconds)"));

  fonaSerial->begin(4800);
  if (! fona.begin(*fonaSerial)) {
    Serial.println(F("Couldn't find FONA"));
    while(1);
  }
  Serial.println(F("FONA is OK"));

  // Print SIM card IMEI number.
  char imei[15] = {0}; // MUST use a 16 character buffer for IMEI!
  uint8_t imeiLen = fona.getIMEI(imei);
  if (imeiLen > 0) {
    Serial.print("SIM card IMEI: "); Serial.println(imei);
  }

  // Setup GPRS APN (username/password optional)
  fona.setGPRSNetworkSettings(F("your_APN"));
  //fona.setGPRSNetworkSettings(F("your_APN"), F("your_username"),
F("your_password"));

  // Turn GPS on
  if (!fona.enableGPS(true)) {
    Serial.println(F("Failed to turn on GPS"));
  }

  // Turn GPRS on
```

```
    fona.enableGPRS(true);

    // Decide between GPS or GPRS localisation
    boolean GPSloc;
    int8_t stat;

    // Check GPS fix
    stat = fona.GPSstatus();
    if (stat < 0) {
      GPSloc = false;
    }
    if (stat == 0 || stat == 1) {
      GPSloc = false;
    }
    if (stat == 2 || stat == 3) {
      GPSloc = false;
    }

    // Print which localisation method is used
    Serial.print("Localisation method: ");
    if (GPSloc) {Serial.println("GPS");}
    else {Serial.println("GPRS");}

    // Print position
    if (GPSloc) {
      char gpsdata[80];
      fona.getGPS(0, gpsdata, 80);
      Serial.println(F("Reply in format: mode, longitude, latitude,
  altitude, utctime(yyyymmddHHMMSS), ttff, satellites, speed, course"));
      Serial.println(gpsdata);
    }
    else {
      uint16_t returncode;
      if (!fona.getGSMLoc(&returncode, replybuffer, 250))
        Serial.println(F("Failed!"));
      if (returncode == 0) {
        Serial.println(replybuffer);
      } else {
        Serial.print(F("Fail code #")); Serial.println(returncode);
      }
    }
  }

  void loop() {
    // Nothing here
  }
```

We are now going to look at the important parts of this sketch.

It starts by importing the required libraries:

```
#include "Adafruit_FONA.h"
#include <SoftwareSerial.h>
```

Then, we define the pins that the FONA is connected to:

```
#define FONA_RX 2
#define FONA_TX 3
#define FONA_RST 4
```

We create some instances for the `SoftwareSerial` object and the FONA breakout:

```
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = &fonaSS;
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);
```

Then, we initialize the FONA:

```
while (!Serial);
Serial.begin(115200);
Serial.println(F("FONA location test"));
Serial.println(F("Initializing....(May take 3 seconds)"));

fonaSerial->begin(4800);
if (! fona.begin(*fonaSerial)) {
  Serial.println(F("Couldn't find FONA"));
  while(1);
}
Serial.println(F("FONA is OK"));
```

Following this is where you need to put your GPRS data. This completely depends on your phone carrier. I simply had to put in an APN, which was `internet`, but you might have to put a username and password as well. Contact your carrier to get the exact information, and then comment/uncomment the required line and fill out the data:

```
fona.setGPRSNetworkSettings(F("your_APN"));
//fona.setGPRSNetworkSettings(F("your_APN"), F("your_username"),
F("your_password"));
```

Then, we can enable the GPS:

```
if (!fona.enableGPS(true)) {
  Serial.println(F("Failed to turn on GPS"));
}
```

We also enable the GPRS connection:

```
fona.enableGPRS(true);
```

When this is done, we get the state of the GPS, and check whether we can locate a GPS satellite and get a fix. If yes, we'll use the GPS for location purposes. If not, we'll switch to a GPRS location:

```
stat = fona.GPSstatus();
if (stat < 0) {
  GPSloc = false;
}
if (stat == 0 || stat == 1) {
  GPSloc = false;
}
if (stat == 2 || stat == 3) {
  GPSloc = false;
}
```

Once we know which location method to use, we can actually get the location, depending on the chosen method:

```
if (GPSloc) {
  location = getLocationGPS();
  latitude = getLatitudeGPS(location);
  longitude = getLongitudeGPS(location);
  latitudeNumeric = convertDegMinToDecDeg(latitude.toFloat());
  longitudeNumeric = convertDegMinToDecDeg(longitude.toFloat());
}
else {
  location = getLocationGPRS();
  latitude = getLatitudeGPRS(location);
  longitude = getLongitudeGPRS(location);
  latitudeNumeric = latitude.toFloat();
  longitudeNumeric = longitude.toFloat();
}
Serial.print("Latitude, longitude: ");
Serial.print(latitudeNumeric, 6);
Serial.print(",");
Serial.println(longitudeNumeric, 6);
```

As you can see, we are calling a few helper functions here to get the `latitude` and `longitude` as float variables. We are now going to see all these functions in detail.

Here is the detail of the function to get the location using GPS:

```
String getLocationGPS() {

  // Buffer
  char gpsdata[80];

  // Get data
  fona.getGPS(0, gpsdata, 80);
  return String(gpsdata);
}
```

This function is really simple, as it simply gets the data from the GPS of the FONA, and returns it as a `String`.

The function to get the location using GPRS is similar:

```
String getLocationGPRS() {

  // Buffer for reply and returncode
  char replybuffer[255];
  uint16_t returncode;

  // Get and return location
  if (!fona.getGSMLoc(&returncode, replybuffer, 250))
    return String("Failed!");
  if (returncode == 0) {
    return String(replybuffer);
  } else {
    return String(returncode);
  }

}
```

Then, we actually need to extract the data from the returned String object, and get the latitude and longitude of our GPS module. Here is the function for the `longitude` using the GPRS location:

```
String getLongitudeGPRS(String data) {

  // Find commas
  int commaIndex = data.indexOf(',');
  int secondCommaIndex = data.indexOf(',', commaIndex+1);

  return data.substring(0, commaIndex);
}
```

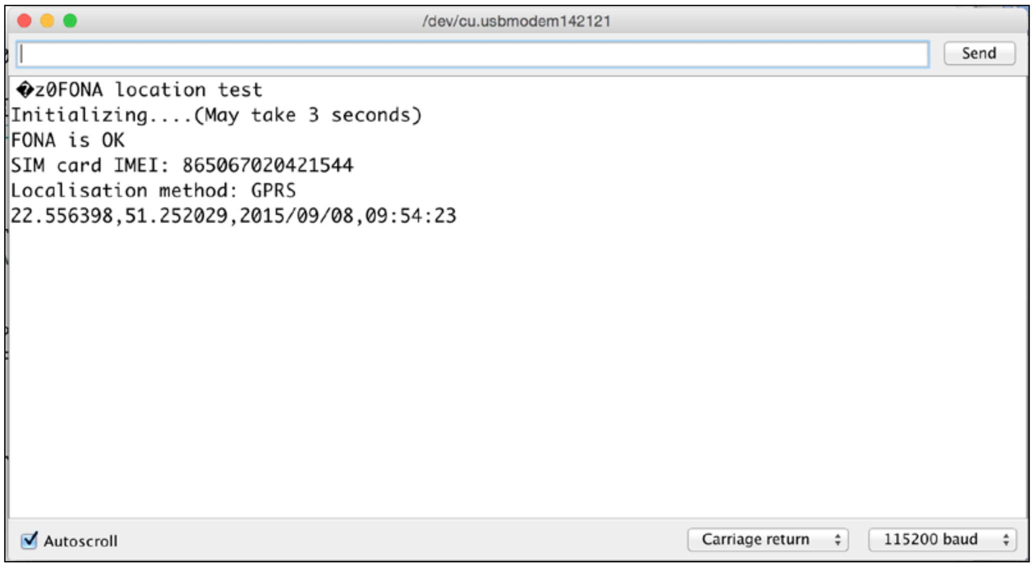
This is the function to get the `latitude` using the GPRS location:

```
String getLatitudeGPRS(String data) {

  // Find commas
  int commaIndex = data.indexOf(',');
  int secondCommaIndex = data.indexOf(',', commaIndex+1);

  return data.substring(commaIndex + 1, secondCommaIndex);
}
```

For the GPS location, things are similar. However, we need one extra function for the GPS module. Indeed, the latitude and longitude are given in degrees-minutes-seconds, and we need to convert this to a numeric format. This is done using the following function:

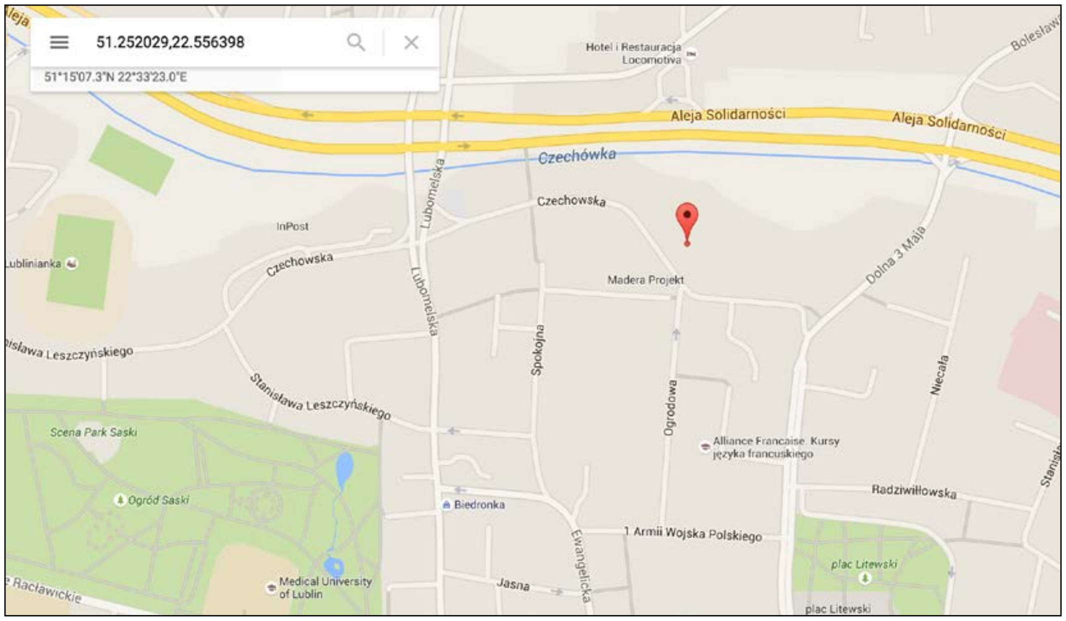
```
double convertDegMinToDecDeg (float degMin) {
  double min = 0.0;
  double decDeg = 0.0;

  // Get the minutes, fmod() requires double
  min = fmod((double)degMin, 100.0);

  //rebuild coordinates in decimal degrees
  degMin = (int) ( degMin / 100 );
  decDeg = degMin + ( min / 60 );

  return decDeg;
}
```

It's now time to test the sketch! Simply put all the code in the Arduino IDE, update your GPRS settings, and upload it to the board. Then, open the serial monitor and this is what you should see:



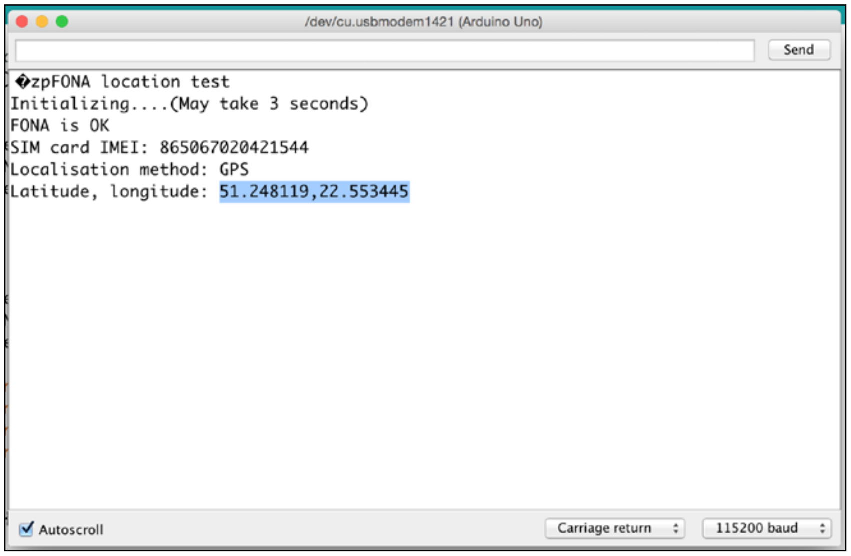
```
�z0FONA location test
Initializing....(May take 3 seconds)
FONA is OK
SIM card IMEI: 865067020421544
Localisation method: GPRS
22.556398,51.252029,2015/09/08,09:54:23
```

As you can see, I was testing this while indoors so the sketch used the GPRS location. I then simply copied the latitude and longitude and pasted them into Google Maps:



I immediately saw a point close to my own location, which was 100-150 meters off my real location. Not too bad if you need to track whether something is inside or outside a city for example.

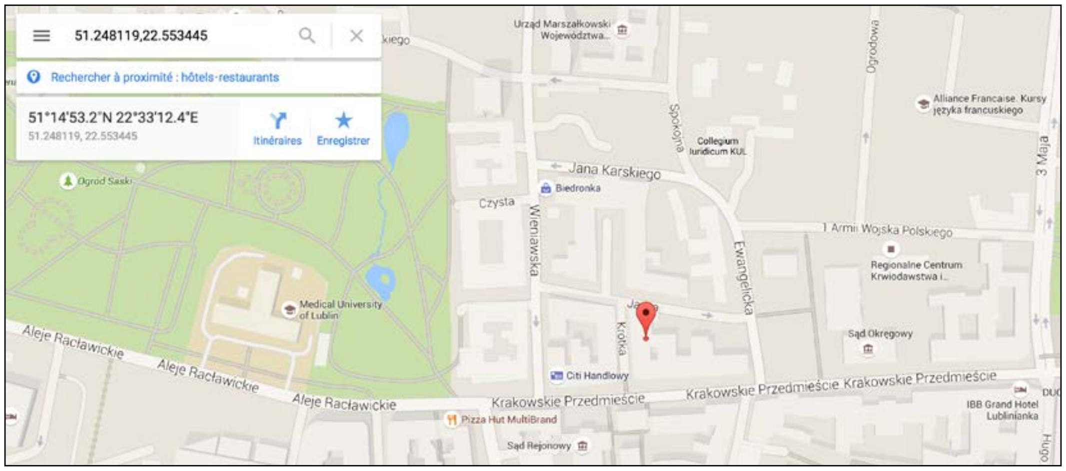
I then tried again outside, with nothing between the GPS antenna and the sky. Then, the sketch automatically selected the GPS location option:



```
�zpFONA location test
Initializing....(May take 3 seconds)
FONA is OK
SIM card IMEI: 865067020421544
Localisation method: GPS
Latitude, longitude: 51.248119,22.553445
```

Autoscroll ✓ Carriage return 115200 baud

I then also copied and pasted the GPS latitude and longitude into Google Maps:



This time, the location was really accurate and showed exactly where I was at that moment.

Sending a GPS location by SMS

Now that we have the location part working correctly, we can start building our GPS tracking projects for secret agents. The first project will simply use the location and send it via SMS to the phone number of your choice.

As the sketch is quite similar to previous one, I'll only show which parts changed compared to the location test sketch. We first need to define the phone number where you want to send the tracking data to:

```
char * sendToNumber = "123456789";
```

After getting the location just as in the previous section, we can now build the message that we will send via SMS:

```
char messageToSend[140];
String message = "Current GPS location: " + latitude + "," +
longitude;
message.toCharArray(messageToSend, message.length());
```

Using the FONA instance, it's very easy to actually send the SMS to the number we defined earlier:

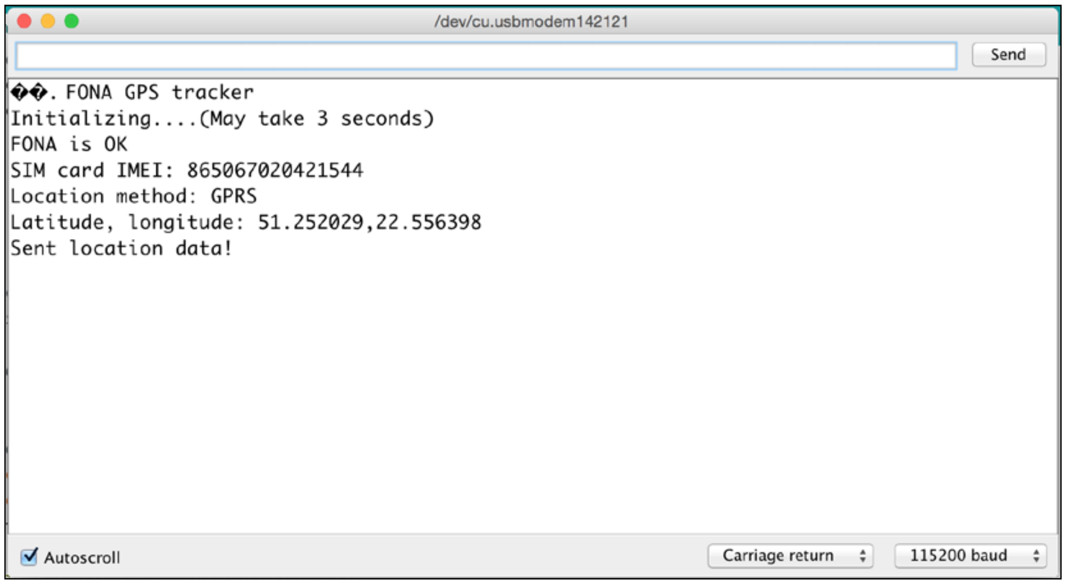
```
if (!fona.sendSMS(sendToNumber, messageToSend)) {
  Serial.println(F("Failed to send SMS"));
} else {
  Serial.println(F("Sent location data!"));
}
```

Finally, we wait before each message. I used 10 seconds for testing purposes, but you can enter your own delay in milliseconds:

```
Serial.println(F("Waiting until next message..."));
delay(10000);
```

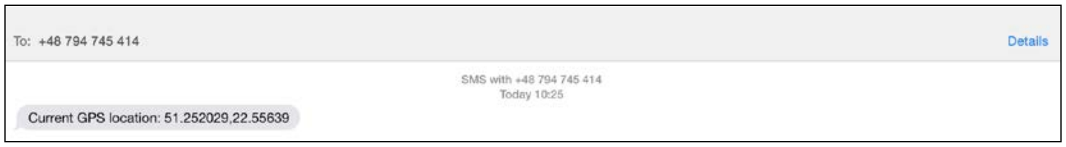
It's now time to test the project. You can grab the whole code from the GitHub repository of the book at `https://github.com/marcoschwartz/arduino-secret-agents`.

Then, make sure you change the phone number and your GPRS settings inside the code and upload it to the board. This is what you should see:



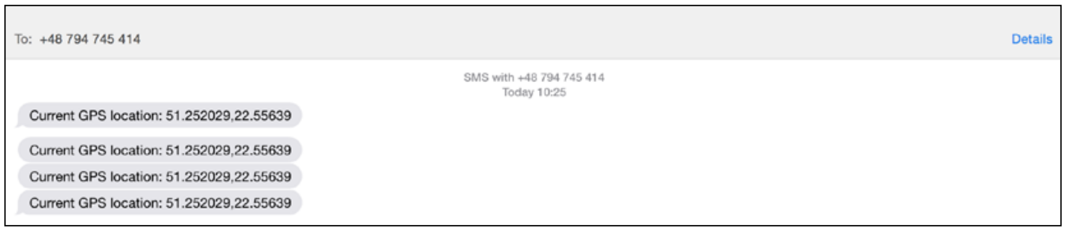
```
��. FONA GPS tracker
Initializing....(May take 3 seconds)
FONA is OK
SIM card IMEI: 865067020421544
Location method: GPRS
Latitude, longitude: 51.252029,22.556398
Sent location data!
```

If you can see the last line, it means that an SMS actually has been sent. I quickly received the message after this:



Current GPS location: 51.252029,22.55639

This was followed by a series of messages with the current location of the project:



Current GPS location: 51.252029,22.55639

Current GPS location: 51.252029,22.55639

Current GPS location: 51.252029,22.55639

Current GPS location: 51.252029,22.55639

If you can see this message on your phone, congratulations, you have built your first GPS tracker using Arduino!

Building a GPS location tracker

It's now time to build the last project of this chapter: a real GPS location tracker. For this project, we'll get the location just as before, using the GPS if available, and the GPRS location otherwise.

However, here we are going to use the GPRS capabilities of the shield to send the latitude and longitude data to dweet.io, which is a service we have used before. Then, we'll plot this data in Google Maps, allowing you to follow your device in real time from anywhere in the world.

As the sketch is very similar to the ones in the previous sections, I'll only detail the most important parts of the sketch here.

You need to define a name for the thing that will contain the GPS location data:

```
String dweetThing = "mysecretgpstracker";
```

Then, after getting the current location, we prepare the data to be sent to dweet.io:

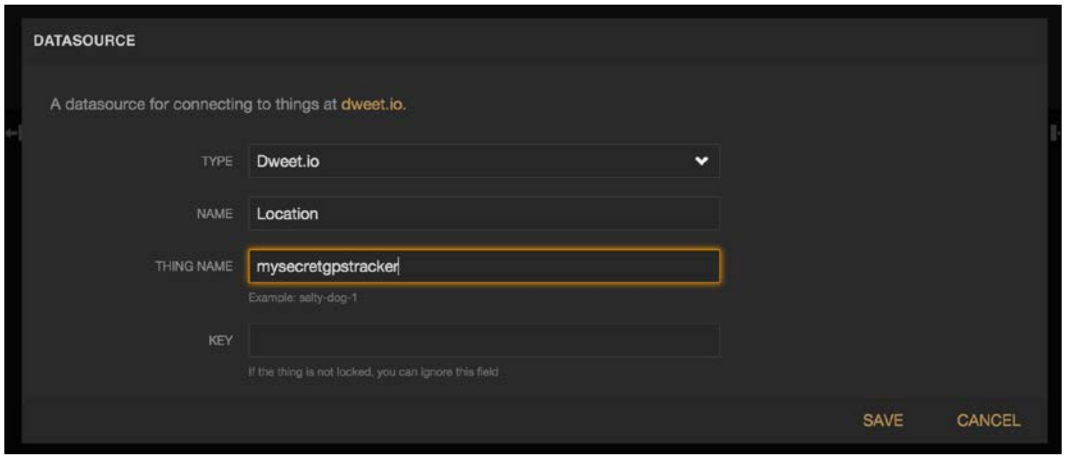
```
uint16_t statuscode;
int16_t length;
char url[80];
String request = "www.dweet.io/dweet/for/" + dweetThing + "?latitude="
+ latitude + "&longitude=" + longitude;
request.toCharArray(url, request.length());
```

After this, we actually send the data to dweet.io:

```
if (!fona.HTTP_GET_start(url, &statuscode, (uint16_t *)&length)) {
    Serial.println("Failed!");
  }
  while (length > 0) {
    while (fona.available()) {
      char c = fona.read();

      // Serial.write is too slow, we'll write directly to Serial
register!
      #if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
        loop_until_bit_is_set(UCSR0A, UDRE0); /* Wait until data
register empty. */
        UDR0 = c;
      #else
        Serial.write(c);
      #endif
      length--;
    }
  }
  fona.HTTP_GET_end();
```

Now, before testing the project, we are going to prepare our dashboard that will host the Google Maps widget. Just as in the previous chapter, we are going to use freeboard.io for this purpose. If you don't have an account yet, go to `http://freeboard.io/`.

Create a new dashboard, and also a new datasource. Insert the name of your `thing` inside the **THING NAME** field:


DATASOURCE

A datasource for connecting to things at dweet.io.

TYPE: Dweet.io

NAME: Location

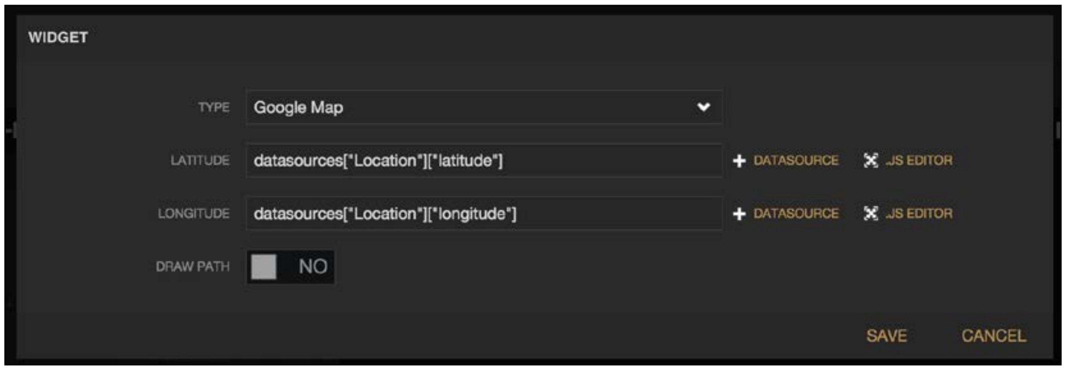
THING NAME: mysecretgpstracker

Example: salty-dog-1

KEY:

If the thing is not locked, you can ignore this field

Then, create a new pane with a Google Maps widget. Link this widget to the latitude and longitude of your location datasource:


WIDGET

TYPE: Google Map

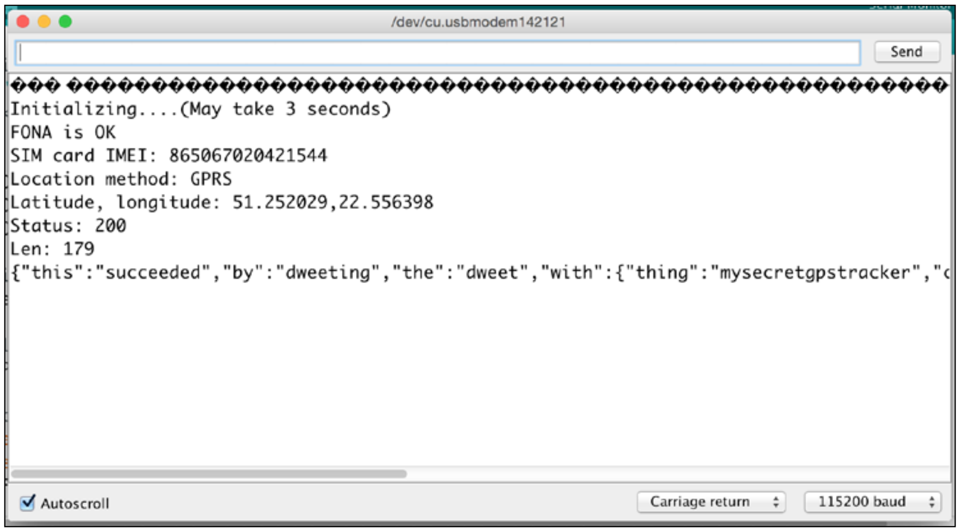
LATITUDE: datasources["Location"]["latitude"]

LONGITUDE: datasources["Location"]["longitude"]

DRAW PATH: NO

It's now time to test the project. Make sure you grab all the code, for example from the GitHub repository of the book. Also, don't forget to modify the thing name, as well as your GPRS settings.

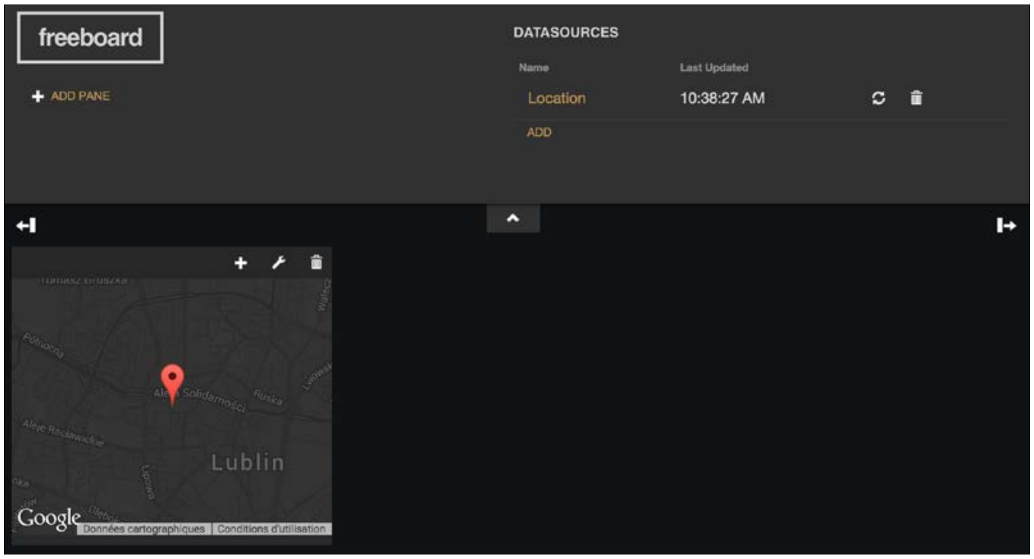
Then, upload the sketch to the board and open the Serial monitor. This is what you should see:



```
��� ����������������������������������������������������
Initializing....(May take 3 seconds)
FONA is OK
SIM card IMEI: 865067020421544
Location method: GPRS
Latitude, longitude: 51.252029,22.556398
Status: 200
Len: 179
{"this":"succeeded","by":"dweeting","the":"dweet","with":{"thing":"mysecretgpstracker","c
```

The most important line is the last one, which confirms that data has been sent to dweet.io and has been stored there.

Now, simply go back to the dashboard you just created, and you can now see that the location on the map has been updated:



Note that this map is also updated in real time as new measurements arrive from the board. You can also modify the delay between two updates of the position of the tracker by changing the `delay()` function in the sketch. Congratulations, you just built your own GPS tracking device!

Summary

In this chapter, we built a device that allows us to track the GPS location of any object it is attached to. We built two projects based on the same hardware: one that sends SMS messages to your phone with the location of the device, and another one where you can see the current location of the device on a map.

There are of course many ways to improve this project. You can, for example, add sensors to the project and monitor them as well from the Freeboard dashboard, just as we did in the previous project. One useful thing is also to make the project autonomous, and for this, you can simply power the Arduino board directly from the FONA shield, as it has a 5V output.

In the final chapter of the book, we are going to build another cool application for secret agents: a small mobile surveillance robot.

9
Building an Arduino Spy Robot

In this last project of the book, we are going to build a small surveillance spy robot based on Arduino. The secret agent will be able to command the robot remotely from a web page and see what the robot is seeing in real time using a camera.

To do this, we will use everything you have learned in this book so far:

- How to use Wi-Fi with Arduino
- How to create control interfaces
- How to use a camera with the Arduino Yun

Let's dive in!

Hardware and software requirements

First let's see what the required components are for this project. The core of the project will of course be the robot itself. For the robot's chassis, you have a wide range of choices available on the market. For this project, I chose a DFRobot MiniQ 2 wheels chassis, which is a small robot chassis that you can easily mount Arduino boards to.

Then, you will need two Arduino boards for this project. The first one will be an Arduino Yun, which we will use to connect a USB camera, just as we did in Module 3 *Chapter 6*, *Building a Cloud Spy Camera*. For the camera itself, I used a C720 camera from Logitech again.

The other thing you will need is an Arduino Uno, which will take care of driving the motors of the robot via a motor shield. We have to use an Arduino Uno here because the Arduino Yun is incompatible with most of the motor shields of DFRobot.

To control the motors, I used a 1A motor shield from DFRobot. In order to control the robot remotely, we'll also use the CC3000 breakout board that we used in an earlier chapter. To assemble the CC3000 board with the rest of the robot, we will also use a DFRobot prototyping shield.

You will also need a 7.2V battery pack to power the robot when it is not connected to your computer. I also used a DFRobot battery pack with a DC jack adapter.

Finally, here is a list of all the components that we will use in this project:

- DFRobot MiniQ 2 wheels chassis (`http://www.dfrobot.com/index.php?route=product/product&product_id=367#.VfP4kmR96u4`)
- DFRobot Motor shield (`http://www.dfrobot.com/index.php?route=product/product&product_id=59&search=motor+shield&description=true#.VfP4wWR96u4`)
- DFRobot Prototyping shield (`http://www.dfrobot.com/index.php?route=product/product&product_id=55&search=prototyping&description=true#.VfP40WR96u4`)
- CC3000 breakout board (`https://www.adafruit.com/products/1469`)
- Arduino Uno (`https://www.sparkfun.com/products/11021`)
- Arduino Yun (`http://www.adafruit.com/product/1498`)
- USB camera (`http://www.logitech.com/en-us/product/hd-webcam-c270`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/8431`)
- 7.2V battery with DC power jack (`http://www.dfrobot.com/index.php?route=product/product&product_id=489`)

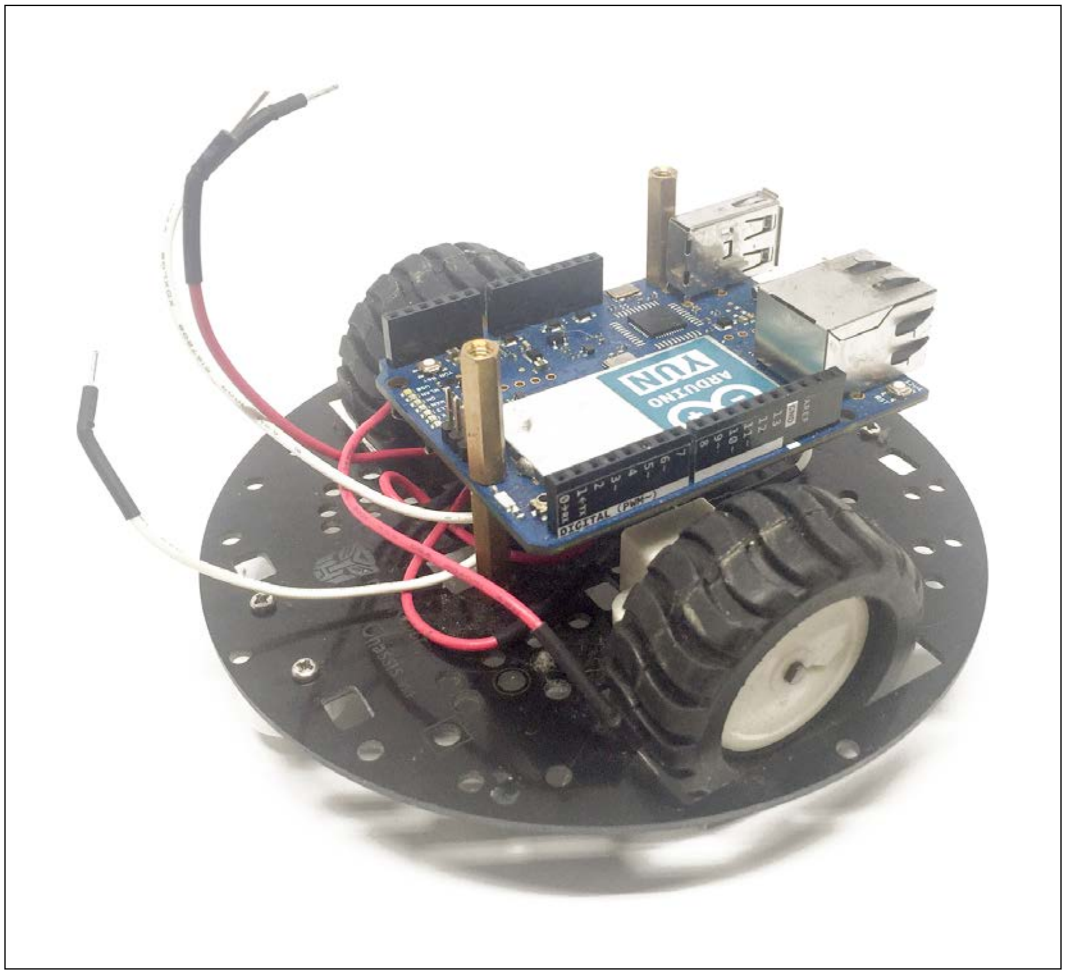
On the software side, you will need the latest version of the Arduino IDE. You will also need to install the following libraries using the Arduino library manager:

- Adafruit CC3000 library
- aREST

Hardware configuration

Let's start with the most difficult part of this project: assembling the robot itself. Of course, the exact steps will depend on the robot chassis you are using, but the goal is to give you the main steps in this section.

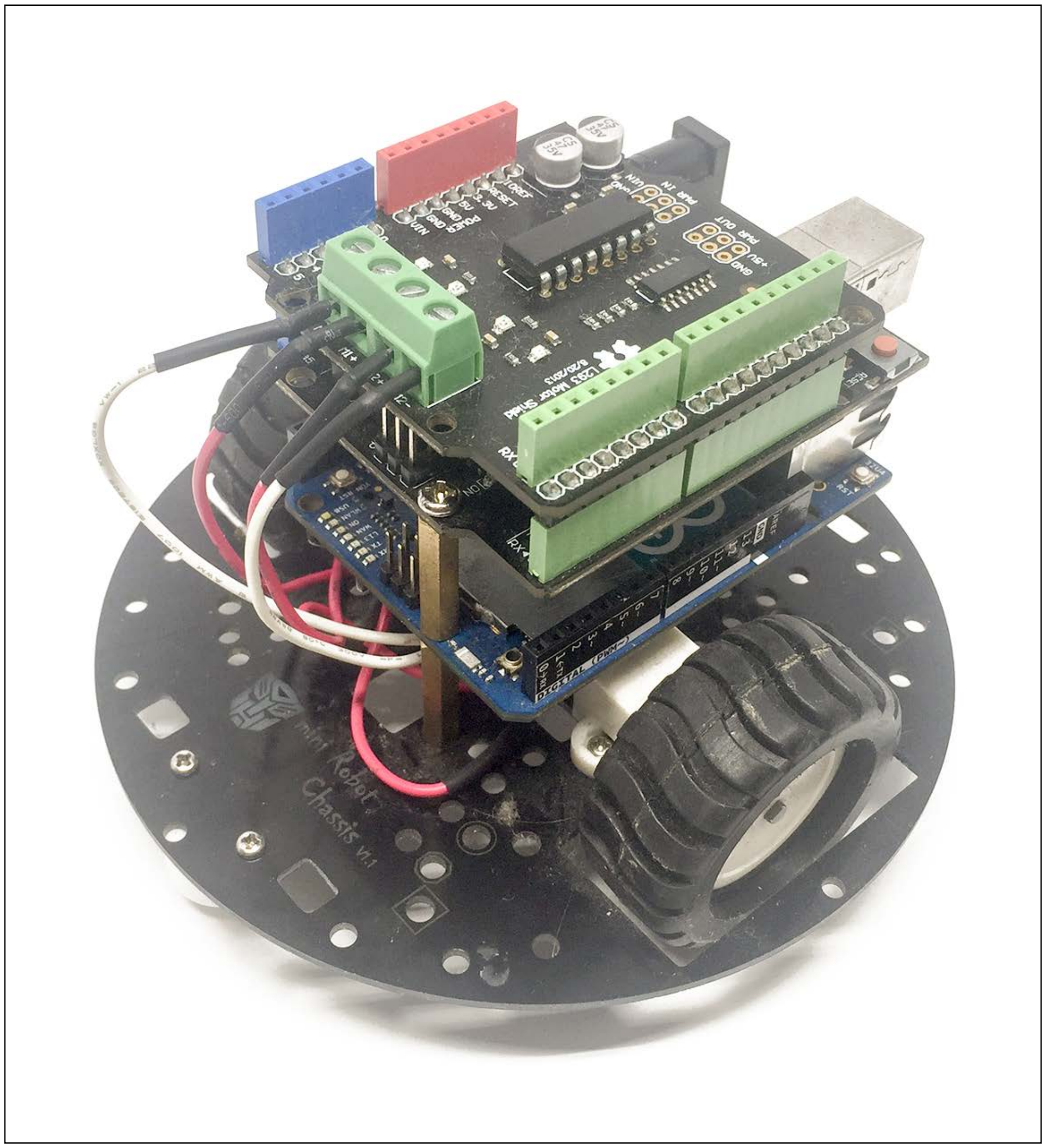
We start by actually mounting the Arduino Yun board on the robot chassis using the screws and headers provided with the chassis itself:



Once that's done, we mount the Arduino Uno board on top of the Yun using an extra row of headers. Then, we tightly screw the Arduino Uno board into the metallic headers.

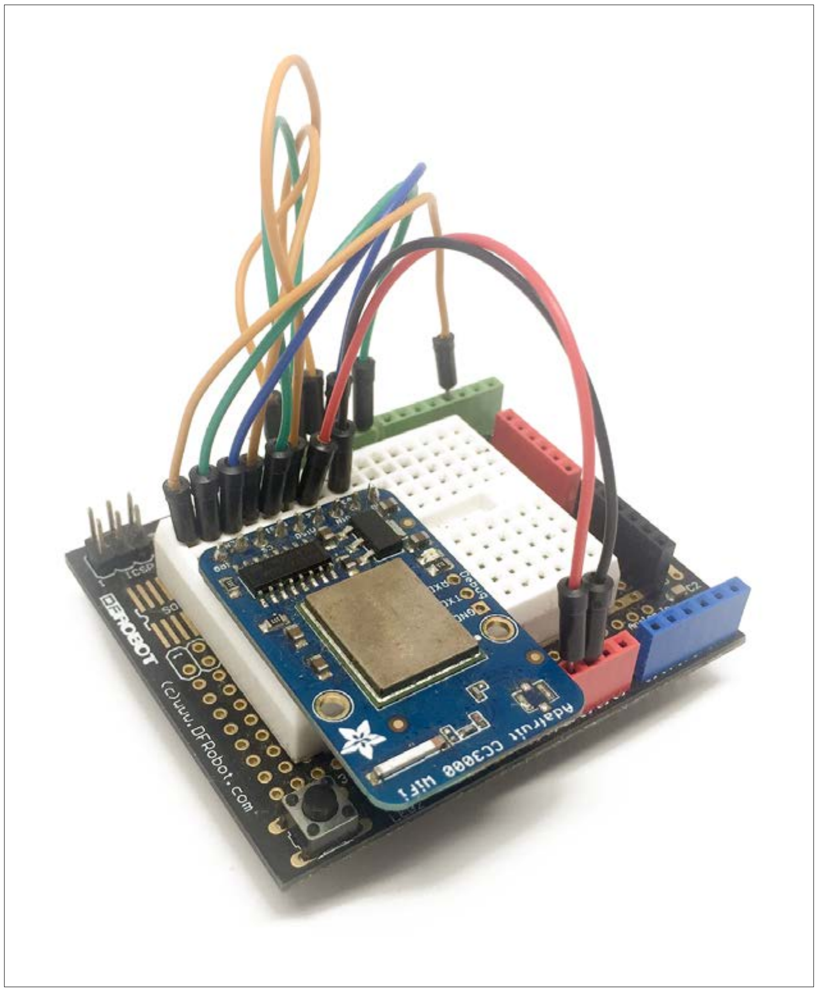
Then, we mount the motor shield on top of the Arduino Uno board. Next, we insert the cables from the motors into the dedicated headers on the motor shield and secure them with the screws. Make sure that you are using the same polarity for both motors.

Here is the result at this point:

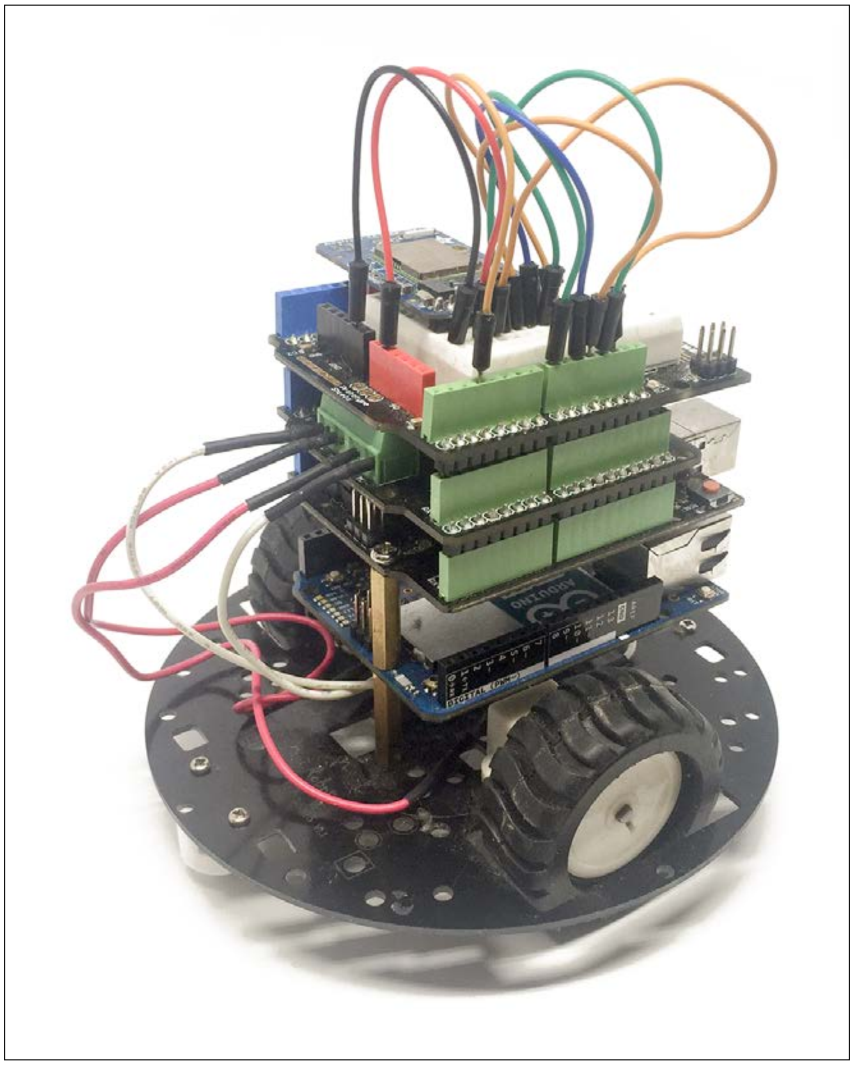


Now, we are going to assemble the CC3000 breakout board on the prototyping shield, which will then sit on top of the robot. To know how to connect the CC3000 board, please refer to Module 3 *Chapter 7*, *Monitoring Secret Data from Anywhere*. The only difference here is that the IRQ pin is connected to pin 3, and the VBAT pin is connected to pin 8.

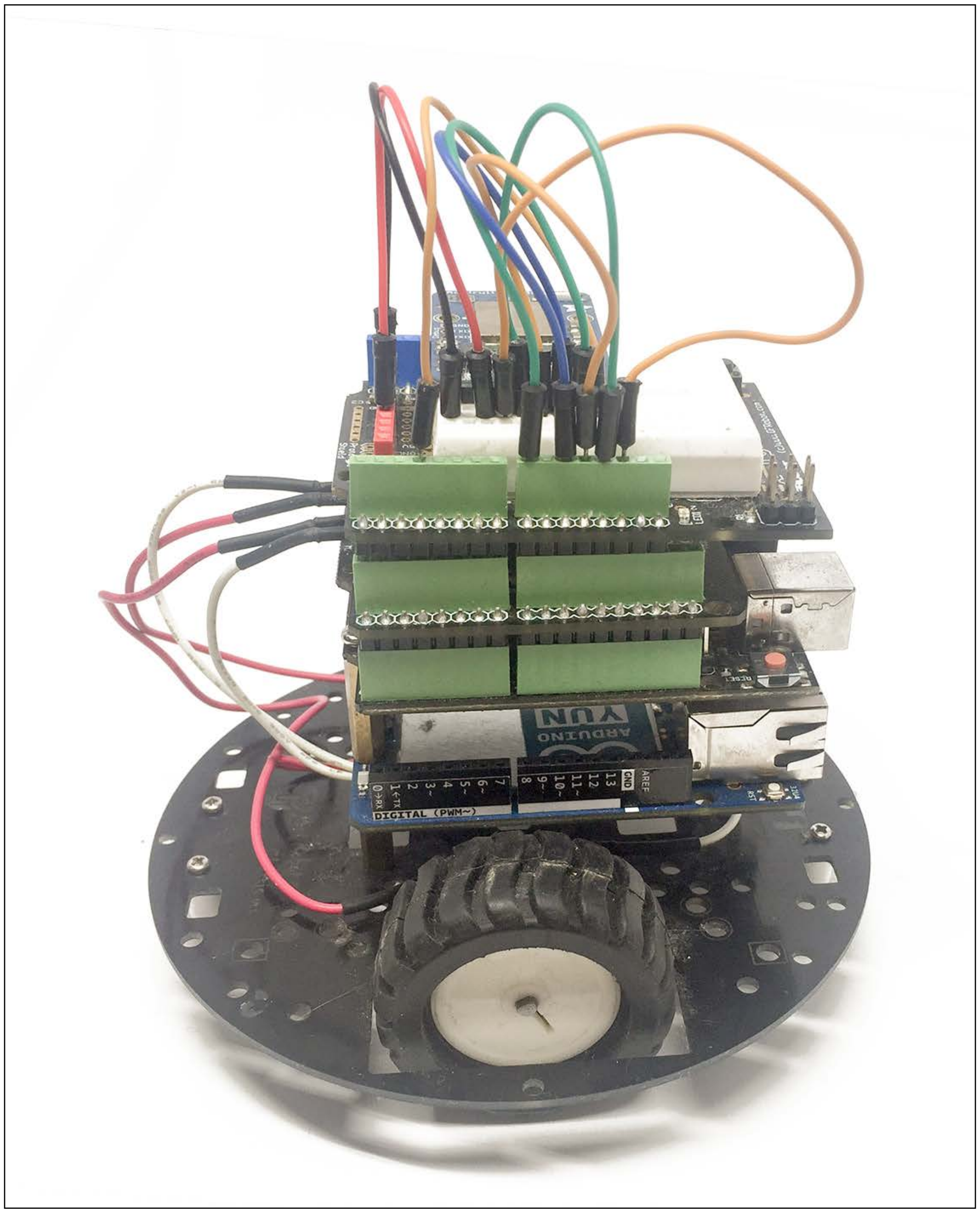
This is the assembled CC3000 board with the prototyping shield:



Now, simply mount this shield on top of the robot:

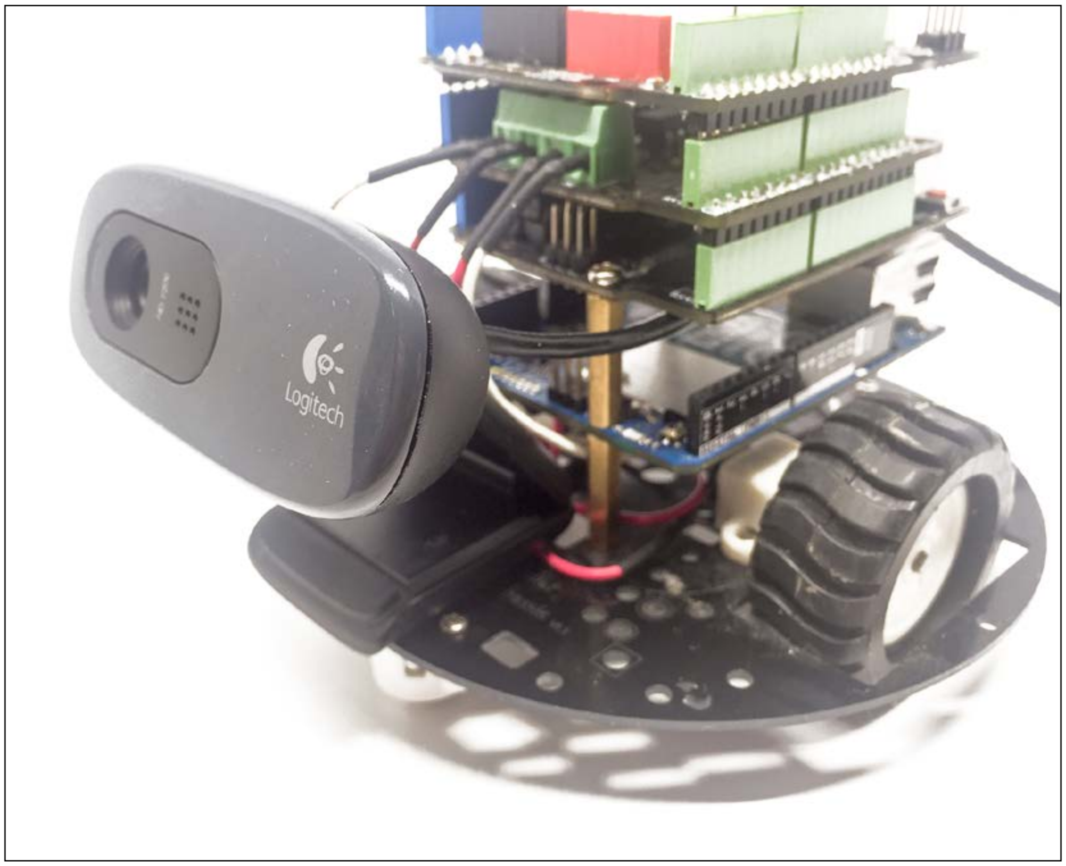


Here is a side view of the project at this stage:

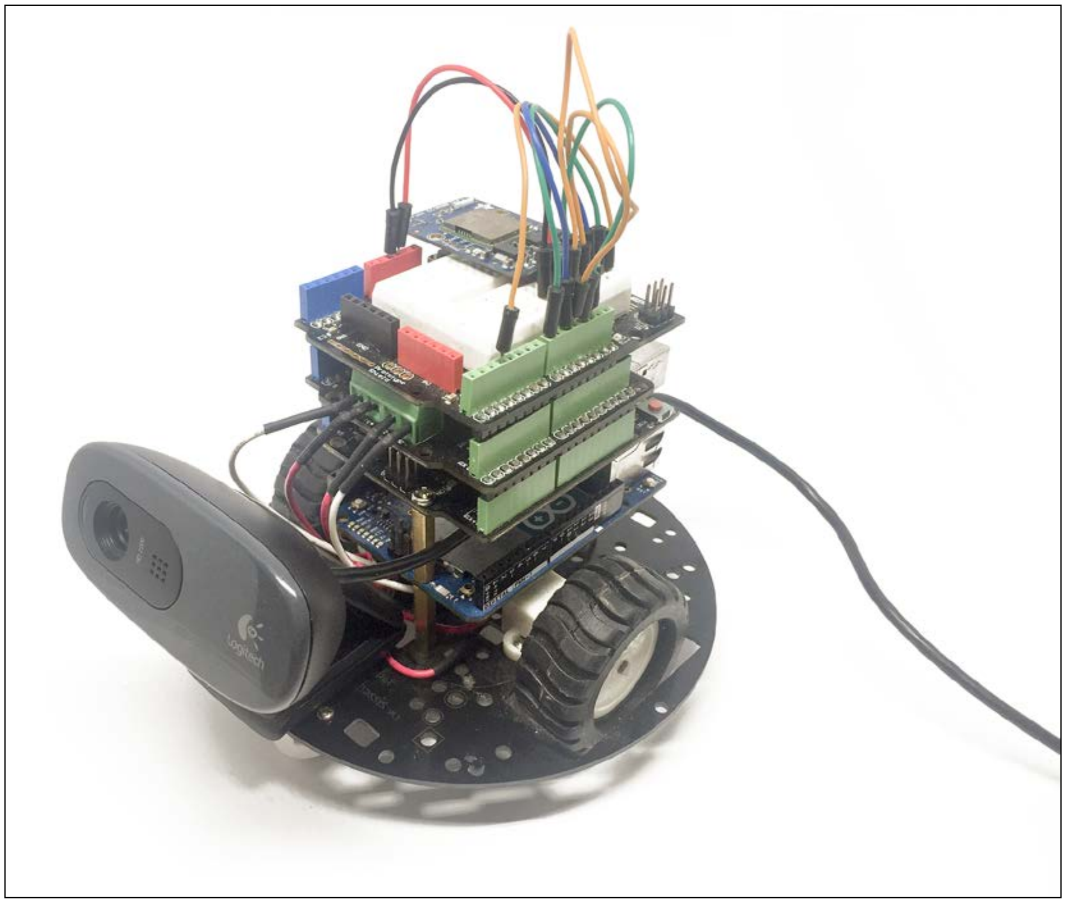


Finally, plug the camera into the USB port of the Arduino Yun and place it in front of the robot. I secured it with screws, but this will depend on your own robot chassis.

This is a close-up picture showing the completely assembled robot:



This is an overview of the completely assembled robot:



If you have a similar result, congratulations! You are now ready to program your surveillance robot. Don't worry about the battery for now, as we will do everything using USB cables that connect the robot directly to your computer.

Setting up the motor control

We are now going to set up the different parts of the project, starting with configuring the Arduino Uno, which will control the motors via Wi-Fi. For this, we'll use the `aREST` Arduino library, which makes it really easy to control Arduino projects via Wi-Fi.

Here is the complete code for this part:

```
// Robot test via aREST + WiFi
#define NUMBER_VARIABLES 1
#define NUMBER_FUNCTIONS 5

// Libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <aREST.h>
#include <avr/wdt.h>

// CC3000 pins
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  8
#define ADAFRUIT_CC3000_CS    10

// Robot speed
#define FULL_SPEED 100
#define TURN_SPEED 50

// Motor pins
int speed_motor1 = 6;
int speed_motor2 = 5;
int direction_motor1 = 7;
int direction_motor2 = 4;

// Sensor pins
int distance_sensor = A0;

// CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS, ADAFRUIT_
CC3000_IRQ, ADAFRUIT_CC3000_VBAT);

// Create aREST instance
aREST rest = aREST();

// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);

#define WLAN_SSID       "KrakowskiePrzedm51m.15(flat15)"
```

```
#define WLAN_PASS       "KrK51flat15_1944_15"
#define WLAN_SECURITY   WLAN_SEC_WPA2

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Give name to robot
  rest.set_id("1");
  rest.set_name("robot");

  // Expose functions
  rest.function("forward",forward);
  rest.function("backward",backward);
  rest.function("left",left);
  rest.function("right",right);
  rest.function("stop",stop);

  // Set up CC3000 and get connected to the wireless network
  Serial.print(F("Initialising CC3000..."));
  if (!cc3000.begin())
  {
    while(1);
  }
  Serial.println(F("done"));

  Serial.print(F("Connecting to WiFi..."));
  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    while(1);
  }
  Serial.println(F("done"));

  Serial.print(F("Getting DHCP..."));
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }

  // Start server
  restServer.begin();
  Serial.println(F("Listening for connections..."));

  displayConnectionDetails();

  wdt_enable(WDTO_8S);
}

void loop() {

  // Handle REST calls
  Adafruit_CC3000_ClientRef client = restServer.available();
  rest.handle(client);
  wdt_reset();

  // Check connection
  if(!cc3000.checkConnected()){while(1){}}
  wdt_reset();
}

// Forward
int forward(String command) {

  send_motor_command(speed_motor1,direction_motor1,100,1);
  send_motor_command(speed_motor2,direction_motor2,100,1);
  return 1;
}

// Backward
int backward(String command) {

  send_motor_command(speed_motor1,direction_motor1,100,0);
  send_motor_command(speed_motor2,direction_motor2,100,0);
  return 1;
}

// Left
int left(String command) {

  send_motor_command(speed_motor1,direction_motor1,75,0);
  send_motor_command(speed_motor2,direction_motor2,75,1);
  return 1;
}

// Right
```

```
int right(String command) {

  send_motor_command(speed_motor1,direction_motor1,75,1);
  send_motor_command(speed_motor2,direction_motor2,75,0);
  return 1;
}

// Stop
int stop(String command) {

  send_motor_command(speed_motor1,direction_motor1,0,1);
  send_motor_command(speed_motor2,direction_motor2,0,1);
  return 1;
}

// Function to command a given motor of the robot
void send_motor_command(int speed_pin, int direction_pin, int pwm,
boolean dir)
{
  analogWrite(speed_pin,pwm); // Set PWM control, 0 for stop, and 255
for maximum speed
  digitalWrite(direction_pin,dir);
}

// Print connection details of the CC3000 chip
bool displayConnectionDetails(void)
{
  uint32_t ipAddress, netmask, gateway, dhcpserv, dnsserv;

  if(!cc3000.getIPAddress(&ipAddress, &netmask, &gateway, &dhcpserv,
&dnsserv))
  {
    Serial.println(F("Unable to retrieve the IP Address!\r\n"));
    return false;
  }
  else
  {
    Serial.print(F("\nIP Addr: ")); cc3000.printIPdotsRev(ipAddress);
    Serial.print(F("\nNetmask: ")); cc3000.printIPdotsRev(netmask);
    Serial.print(F("\nGateway: ")); cc3000.printIPdotsRev(gateway);
    Serial.print(F("\nDHCPsrv: ")); cc3000.printIPdotsRev(dhcpserv);
    Serial.print(F("\nDNSserv: ")); cc3000.printIPdotsRev(dnsserv);
    Serial.println();
    return true;
  }
}
```

As this code is quite long, we are only going to look at the most important parts here. We start by including all the required libraries:

```
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <aREST.h>
#include <avr/wdt.h>
```

Then, we define the pins that correspond to the motor shield:

```
int speed_motor1 = 6;
int speed_motor2 = 5;
int direction_motor1 = 7;
int direction_motor2 = 4;
```

After this, you need to enter your own Wi-Fi network name and password:

```
#define WLAN_SSID       "your_wifi_ssid"
#define WLAN_PASS       "your_wifi_password"
#define WLAN_SECURITY   WLAN_SEC_WPA2s
```

Then, we declare the `aREST` instance, which we'll use later to access motor functions via Wi-Fi:

```
aREST rest = aREST();
```

In the `setup()` function of the sketch, we expose the different functions to control the robot so they are accessible via Wi-Fi:

```
rest.function("forward",forward);
rest.function("backward",backward);
rest.function("left",left);
rest.function("right",right);
rest.function("stop",stop);
```

After that, we start a Wi-Fi server on the board:

```
restServer.begin();
Serial.println(F("Listening for connections..."));
```

In the `loop()` function of the sketch, we continuously listen for connections and handle them with the `aREST` instance:

```
Adafruit_CC3000_ClientRef client = restServer.available();
rest.handle(client);
```

Now let's have a look at one of the functions to control the robot, for example, the one to go forward:

```
int forward(String command) {

  send_motor_command(speed_motor1,direction_motor1,100,1);
  send_motor_command(speed_motor2,direction_motor2,100,1);
  return 1;
}
```

As you can see, the work is done by another function that directly acts on the motors:

```
void send_motor_command(int speed_pin, int direction_pin, int pwm,
boolean dir)
{
  analogWrite(speed_pin,pwm);
  digitalWrite(direction_pin,dir);
}
```

It's now time to configure this part of the project. Connect the Arduino Uno to the computer via USB and upload this sketch to the board. Then, open the Serial monitor. After a while, you should see the IP address of the board being printed on the Serial monitor.

Setting up live streaming

We can now move to the next part: configuring live camera streaming on the Arduino Yun. This is something we already did in Module 3 *Chapter 6*, *Building a Cloud Spy Camera*, so we'll only look at the most important parts here. Refer to the *Hardware configuration* section if you need to know how to configure your Yun again.

First, connect to your Yun using the following command:

```
ssh root@arduinoyun.local
```

Then, launch camera streaming with the following command:

```
mjpg_streamer -i "input_uvc.so -d /dev/video0 -r 640x480 -f 25" -o
"output_http.so -p 8080 -w /www/webcam" &
```

You can check that the streaming is working at the following page `http://arduinoyun.local:8080`.

Setting up the interface

We can now move to the last part: setting up the interface that will allow a secret agent to command the robot and also to see the live stream from the camera. This interface will be composed of an HTML page and a JavaScript file. It will be based on the `aREST.js` module that makes it easy to control aREST devices from a web page.

This is the complete HTML page:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
  <title>Surveillance Robot</title>
  <link rel="stylesheet" type="text/css" href="https://maxcdn.
bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">
  <link rel="stylesheet" type="text/css" href="interface.css">
  <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.4.min.js"></script>
  <script type="text/javascript" src="https://cdn.rawgit.com/Foliotek/
AjaxQ/master/ajaxq.js"></script>
  <script type="text/javascript" src="https://cdn.rawgit.com/
marcoschwartz/aREST.js/master/aREST.js"></script>
  <script type="text/javascript" src="interface.js"></script>
</head>

<body>

<div class='container'>

<h1>Surveillance Robot</h1>


<div class='row'>

  <div class="col-md-2"></div>

  <div class="col-md-2">
    <button id='fw' class='btn btn-primary btn-block'
type="button">Forward</button>
  </div>

</div>

<div class='row'>

  <div class="col-md-2">
```

```
    <button id='left' class='btn btn-primary btn-block'
type="button">Left</button>
  </div>

  <div class="col-md-2">
    <button id='stop' class='btn btn-danger btn-block'
type="button">Stop</button>
  </div>

  <div class="col-md-2">
    <button id='right' class='btn btn-primary btn-block'
type="button">Right</button>
  </div>

</div>

<div class='row'>

  <div class="col-md-2"></div>

  <div class="col-md-2">
    <button id='bw' class='btn btn-primary btn-block'
type="button">Backward</button>
  </div>

</div>

<div class='row'>

  <img src="http://arduinoyun.local:8080/?action=stream" />

</div>

</div>

</body>
</html>
```

The most important parts on this page are the buttons to control the robot and the live stream of the camera. For example, this defines the button to move the robot forward:

```
<div class="col-md-2">
    <button id='fw' class='btn btn-primary btn-block'
type="button">Forward</button>
  </div>
```

This `<img>` tag allows you to insert the live stream of the camera into the page:

```
<img src="http://arduinoyun.local:8080/?action=stream" />
```

Now let's look at the JavaScript file that will actually send the command to the robot. This is the complete file:

```
$( document ).ready(function() {

    // Device
    var address = '192.168.1.105';
    var device = new Device(address);

    // Button
    $('#fw').click(function() {
      device.callFunction('forward', '');
    });

    $('#bw').click(function() {
      device.callFunction('backward', '');
    });

    $('#left').click(function() {
      device.callFunction('left', '');
    });

    $('#right').click(function() {
      device.callFunction('right', '');
    });

    $('#stop').click(function() {
      device.callFunction('stop', '');
    });

});
```

You only have to change one thing in this script: the IP address of your board, which you obtained previously:

```
  var address = '192.168.1.105';
  var device = new Device(address);
```

The different lines of code on the page each control a function of the robot. For example, this piece of code creates the link between the forward button and the `forward` function on the robot:

```
$('#fw').click(function() {
     device.callFunction('forward', '');
});
```

Testing the surveillance robot

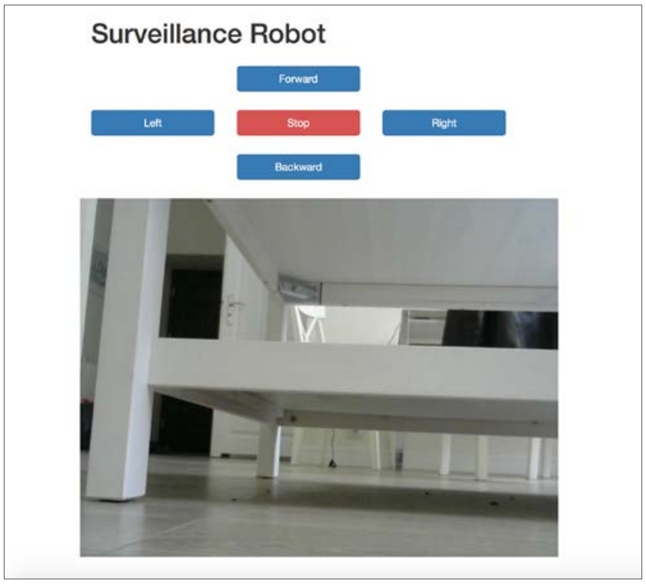
Now that we have all the elements in place, it's time to test our little surveillance robot. Note that you can grab the complete code from `https://github.com/marcoschwartz/arduino-secret-agents`.

Make sure that you configured the code and the hardware with the instructions from the previous sections.

Now, it's time to power your robot from the battery. Insert the battery into the robot chassis, and then plug it into the Arduino Uno DC jack input.

Then, connect the Arduino Uno 5V pin to the Vin pin of the Yun. Also connect the GND pins of the two boards together. This will ensure that the Arduino Yun will be powered as well. Don't forget to start the live video stream again after that.

Now, place your robot on the ground and open the HTML page. This is what you should see:



Now, try the different buttons, and you should see that the robot reacts nearly instantly. Congratulations, you just built your own spy robot based on Arduino!

Summary

In this last chapter of the book, you built your own little surveillance robot controlled by Wi-Fi. With this robot, you can now spy on what's going on in a room and also move the robot around.

There are several ways to improve this project. For example, you can add more sensors to the robot and display this sensor data on the same page that you control the robot from.

You have already reached the end of this book, and I hope that you enjoyed reading all the secret agent projects! I now really encourage you to try all the projects from this book again, and build them yourself. I also advise that you use all the knowledge from this book and build your own secret agent projects. Have fun!

Bibliography

This course is a blend of different projects and texts all packaged up keeping your journey in mind. It includes the content from the following Packt products:

- *Arduino by Example - Adith Jagdish Boloor*
- *Arduino BLINK Blueprints - Samarth Shah, Utsav Shah*
- *Arduino for Secret Agents - Marco Shwartz*