Make:

Jumpstarting Raspberry Pi Vision

Machine Learning and Facial Recognition on a Single-Board Computer

SANDY ANTUNES | JAMES WEST

Make:

JUMPSTARTING
Raspberry Pi Vision

MACHINE LEARNING AND FACIAL RECOGNITION ON A SINGLE-BOARD COMPUTER

Sandy Antunes and James West

Maker Media, Inc.
San Francisco

Published by
Maker Media, Inc.
1700 Montgomery Street, Suite 240
San Francisco, CA 94111

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

Publisher: Roger Stewart
Editor: Patrick DiJusto
Copy Editor: Elizabeth Welch
Proofreader: Scout Festa
Interior and Cover Designer: Maureen Forys, Happenstance Type-O-Rama

October 2018: First Edition

Revision History for the First Edition

2018-10-08 First Release

See *oreilly.com/catalog/errata.csp?isbn=9781680455427* for release details.

978-1-680-45542-7

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business. Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training. Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals. Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us
Please address comments and questions to the publisher:

Maker Media, Inc.
1700 Montgomery Street, Suite 240
San Francisco, CA 94111

You can send comments and questions to us by email at
books@makermedia.com.

Maker Media unites, inspires, informs, and entertains a growing community of resourceful people who undertake amazing projects in their backyards, basements, and garages. Maker Media celebrates your right to tweak, hack, and bend any Technology to your will. The Maker Media audience continues to be a growing culture and community that believes in bettering ourselves, our environment, our educational system—our entire world. This is much more than an audience, it's a worldwide movement that Maker Media is leading. We call it the Maker Movement.

To learn more about Make: visit us at *make.co*.

CONTENTS

PREFACE

If a camera takes a picture and no one notices, did it really happen? In this book, we're going to show you how to make a working Raspberry Pi–based camera system so that you can capture time-lapse images and view via WiFi, trigger the camera if motion is detected, and even carry out basic facial recognition as an introduction to machine learning methods.

Take a Raspberry Pi and add a camera module, and you have a programmable camera. Add some software, and you can start to do interesting surveillance and automatic object recognition work with it. Activate the Pi as a WiFi node and you can do all these wonderful things from a distance.

A good surveillance system does more than take pictures. It should also turn those pictures into actionable information that increases your knowledge. That's now easily done in software, and we're going to show you how.

This book pulls together a set of little tricks—setting up Pi cameras, making a Pi broadcast as a WiFi device, adding time lapse and motion detection and face recognition, and sticking a battery pack on it so it can function anywhere—to create portable spy cameras. We've used these rigs in everything from "Find the Pi" party contests, to implementing privacy-respecting security in our lab, to showing off modern tech like facial recognition. Other uses could include monitoring deer and wildlife, checking your house mailbox for mail arrival, and capturing time-lapse sequences of natural events or traffic patterns.

Our first project will be to create a portable "SpyPi" camera rig that broadcasts images via WiFi without needing a network—you just have to log into it to see what the camera sees . It's great for a "find the camera" hunting challenge or stand-alone security.

The second project will involve setting up a time-lapse camera that can do simple motion detection—perfect for monitoring a location. The system will include automatic deletion of old images after a week.

The third project will be to add facial detection and facial recognition to your SpyPi. In the process, you'll learn about the tools and software installed by the end of these projects so that you can go beyond this book and explore additional machine learning methods.

So grab a Pi, an SD card, a USB camera, and a USB battery pack, and let's go!

1

Introduction to the Raspberry Pi

In this book, we will be discussing how to take a Raspberry Pi and turn it into a do-it-yourself security system. The following chapters will cover projects ranging from taking time-lapse photos to using facial recognition. Before we get to any of those, though, we need to spend some time preparing the Pi so that it's ready for those projects.

These setup processes may not be the most exciting information covered in this book, but they are arguably the most crucial for you to know. The Raspberry Pi is a versatile device, providing a good base for a variety of projects. It is for that very reason that knowing how to modify and customize your Pi to suit a given project is critical.

This chapter will cover, among other things, how to install software packages and connect to your Pi remotely. This knowledge is useful in any Pi project and helps open up possibilities for taking your projects further.

FIGURE 1.1: Raspberry Pi 3 with USB webcam

GEAR

You will need the following:

- A Raspberry Pi (the Pi 3 model is recommended, but any model will work)—$35
- A microSD card (8GB minimum)—$6–$20
- A USB camera (or Pi-specific ribbon camera)—$10–$20
- A 5V battery pack (cell phone recharger, USB charge pack, etc.)—$5–$10
- A USB WiFi module (only if you're not using Pi 3 or Pi Zero models, which have built-in WiFi)—$10–$15

For the initial setup, you will also need

- HDMI monitor and cable
- 5V Pi-compatible power supply (a microUSB phone charger should work)
- USB mouse and keyboard

The following are strongly recommended but not mandatory:

- A USB power supply (2000mA or more)—$10
- A case—$5, or 3D-print your own with plans from *thingiverse.com*

raspberrypi.org lists online sites by country, so that's a good place to start. You can also buy kits that include everything listed (and often more add-ons).

To make your project portable, you'll need a standard USB battery pack (the kind used to recharge phones and devices; you can find them for $5–$10 at electronics or convenience stores). The larger the pack, the longer the Pi will last. A five-dollar 4000mAH (milliamp hours) device powered our project for over four hours at an outdoor conference.

For the SD card, we recommend you get a name brand like SanDisk, Kingston, PNY, Sony, Toshiba, or Samsung. Don't get

Older Pi Models

Just to complicate things, if you have an older Pi Model A or B (not an A+ or B+ or Pi 2 or Pi 3 or Pi Zero), it will require the older, larger SD instead of a microSD card, but at this point the older Pis are hard to find so you probably don't have to worry about it. The rule is simple: look at what physical form factor card (SD or microSD) fits in your Pi, and buy that one. Typically, you want to buy the latest model, which as of this writing is the Pi 3 Model B+, because the newest model will be faster and have more memory yet will be the same price as the other models. But you can use an older one, especially if you already have it lying around. There are different models of Pi. These are all fine: Pi 3 Model B or Model B+, Pi 2 Model B, or even Pi 1 Model B+—they all have four USB plugs so they're easy to hook up to lots of things. Avoid the "weird" ones (Pi Zero, Pi Compute Module, or the old Model A) unless you really know about Pi. But it's easy to tell them apart—buy a $35 Pi and you're likely good. Fun fact: Raspberry Pis are always $35 because there's only one vendor and Raspberry Pis are intended for educational and maker use, so they don't change in price. Since all the models we can use are the same price, there's no reason not to buy the latest.

a cheap generic card; they aren't as reliable. The card must be marked as SDHC (secure digital high capacity). The cards are available at speeds of Class 2, 4, 6, or 10—the higher the number, the faster the card. As long as it's at least Class 4, the actual speed isn't very important—the Raspberry Pi Foundation recommends Class 6 or higher, but Class 4 work well for everything in this book. For more details, you can visit *www.raspberrypi.org/documentation/installation/sd-cards.md*.

For the camera, if it's USB and it works on your PC or Mac or laptop, it'll most likely work on a Pi (we tested 11 different cameras, including super-cheap generic ones!). Cameras for surveillance don't need to be fancy; they are just commodities. It's the software you'll be installing that does the magic.

To begin, you're going to hook up your Pi to a monitor, keyboard, and mouse, and use it like an ordinary computer. After you finish getting everything set up and are ready to deploy the device, you'll ditch the keyboard, mouse, and HDMI monitor and power cable; add the battery pack; and shift to "headless" mode, where the Pi is accessible only via the Internet. In this mode, you'll interact with the Pi via a web browser, remote access tools like VLC, or an SSH terminal.

We'll cover a base loadout and setup, and we'll include sidebars on using other forms of gear. For example, any Pi models (including the Pi Zero or original Pi Model A) can do everything in this book; they just require a little extra setup. As for cameras, a Pi can use either USB cameras, its own model of ribbon-cable cameras, or special cameras that connect via the general-purpose input/output (GPIO) pins that stick off the Pi board. As we said, we recommend an inexpensive USB camera that you can buy just about anywhere, but we'll include notes on using other cameras if you want to play around with alternatives.

SETTING UP

Two ways to set up your Pi (based on the official Raspberry Pi Software Guide at *www.raspberrypi.org/learning/software-guide*) are as follows:

* Buy a preloaded SD card with NOOBS (New Out-of-Box Software) or Raspbian.
* Buy a blank SD card and load either NOOBS or Raspbian from *http://raspberrypi.org/downloads*.

Both lead to the same result—you first boot your new Pi with your microSD card that has the operating system installed, and then you follow its installation menu. Pro tip: If you're in the United States, you'll have to reset the keyboard from UK style to US style or else you'll find all the punctuation marks are in the wrong place (on the Pi, type `sudo raspi-config` and use the location or keyboard settings menu to do this).

You'll need to hook up the Pi to an HDMI monitor as well as a keyboard and mouse for setup. You can also plug in a wired network cable, or just wait and configure the built-in WiFi (Pi Model 3 or later, or add in a WiFi USB dongle if you have an earlier Pi without built-in WiFi) to connect to your local network. Once we finish this, you won't need to use the monitor, keyboard, and mouse to work with it, which is a great advantage of a Raspberry Pi. You can work headless, with just the Pi itself and a power source, wirelessly and with no extra attachments needed.

NOTE **Plug in the USB camera, ideally before you power up the system. The Pi will auto-detect the camera upon boot; it might not always detect the camera if you plug it in or remove it while the Pi is running.**

ADDING SOFTWARE

We'll install all the free software you'll need for this book at once. Also, we're not going to use the GUI or have you click any icons; instead, you'll run commands in the terminal by choosing Terminal from the Start menu. (If you didn't install a GUI, you're already in the terminal and ready to go.)

The command for installing software in Raspbian Linux is `sudo apt-get install` *`SOFTWARENAME`*, which automatically finds and installs the software via the Internet.

First make sure your operating system is current and up to date by entering the following in the terminal on your Pi:

```
sudo apt-get update
sudo apt-get upgrade
```

```
pi@raspberrypi:/ $ sudo apt-get update
Ign:1 http://ftp.debian.org/debian stretch InRelease
Get:2 http://security.debian.org stretch/updates InRelease [94.3 kB]
Get:3 http://archive.raspberrypi.org/debian stretch InRelease [25.3 kB]
Get:4 http://ftp.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:5 http://archive.raspberrypi.org/debian stretch/main amd64 Packages [106 kB]
Get:6 http://ftp.debian.org/debian stretch Release [118 kB]
Get:7 http://ftp.debian.org/debian stretch-updates/main amd64 Packages [11.6 kB]
Get:8 http://ftp.debian.org/debian stretch-updates/main i386 Packages [11.6 kB]
Get:9 http://archive.raspberrypi.org/debian stretch/main i386 Packages [106 kB]
Get:10 http://ftp.debian.org/debian stretch-updates/main Translation-en [7,761 B
]
Get:11 http://archive.raspberrypi.org/debian stretch/ui i386 Packages [33.1 kB]
Get:12 http://security.debian.org stretch/updates/main i386 Packages [375 kB]
Get:13 http://archive.raspberrypi.org/debian stretch/ui amd64 Packages [33.3 kB]
Get:14 http://ftp.debian.org/debian stretch Release.gpg [2,434 B]
Get:15 http://security.debian.org stretch/updates/main amd64 Packages [374 kB]
Get:16 http://ftp.debian.org/debian stretch/main i386 Packages [7,078 kB]
Get:17 http://security.debian.org stretch/updates/main Translation-en [175 kB]
Get:18 http://security.debian.org stretch/updates/contrib amd64 Packages [1,760
B]
Get:19 http://security.debian.org stretch/updates/contrib i386 Packages [1,760 B
]
Get:20 http://ftp.debian.org/debian stretch/main amd64 Packages [7,099 kB]
Get:21 http://ftp.debian.org/debian stretch/main Translation-en [5,393 kB]
Get:22 http://ftp.debian.org/debian stretch/contrib amd64 Packages [50.9 kB]
Get:23 http://ftp.debian.org/debian stretch/contrib i386 Packages [48.0 kB]
Get:24 http://ftp.debian.org/debian stretch/contrib Translation-en [45.9 kB]
Get:25 http://ftp.debian.org/debian stretch/non-free i386 Packages [69.7 kB]
Get:26 http://ftp.debian.org/debian stretch/non-free amd64 Packages [78.7 kB]
Get:27 http://ftp.debian.org/debian stretch/non-free Translation-en [80.6 kB]
Fetched 21.5 MB in 41s (524 kB/s)
Reading package lists... Done
```

FIGURE 1.2: `sudo apt-get update`

Next, install the software you'll need to complete Chapter 1:

```
sudo apt-get install fswebcam -y
sudo apt-get install feh -y
sudo apt-get install python-picamera python3-picamera
```

```
pi@raspberrypi:/ $ sudo apt-get install fswebcam
Reading package lists... Done
Building dependency tree
Reading state information... Done
fswebcam is already the newest version (20140113-1+b1).
0 upgraded, 0 newly installed, 0 to remove and 288 not upgraded.
```

FIGURE 1.3: Installing `fswebcam`

Then, install the software for Chapter 2:

```
sudo apt-get install apache2 -y
sudo apt-get install imagemagick -y
```

Continue with the Chapter 3 software next:

```
sudo apt-get install motion -y
```

Finally, install the Chapter 4 software:

```
sudo apt-get install python-opencv -y
sudo apt-get install python-pip python3-pip -y
pip install numpy
```

Keep that terminal window open, because you'll type more commands in it. As a maker/hacker, you'll find it's quicker to type and edit in this window rather than use a GUI.

SETTING UP THE CAMERA

Time to plug in your USB camera and do some initial diagnostics. To capture images, you'll use `fswebcam` as the primary program. For more details (beyond our usage in this book), you can learn more at *www.raspberrypi.org/documentation/usage/webcams*, but let's skip right to the real work.

When you typed **`sudo apt-get install fswebcam`** in the terminal, you installed a package of programs that can control a USB camera. You need to test them to make sure that they installed correctly. Type the following in the terminal to check:

```
fswebcam image.jpg
```

```
pi@raspberrypi:~ $ fswebcam image.jpg
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
No input was specified, using the first.
Adjusting resolution from 384x288 to 424x240.
--- Capturing frame...
Captured frame in 0.00 seconds.
--- Processing captured image...
Writing JPEG image to 'image.jpg'.
```

FIGURE 1.4: Taking a test image with `fswebcam`

This command will tell the camera to take a snapshot of whatever it sees and save it to the current directory. If you are using the GUI, you should see the image once you open the File Manager. If you are working via the terminal, simply check the directory with the `ls` (list) command.

```
pi@raspberrypi:~ $ ls
Desktop    Downloads  Music     Public        Templates
Documents  image.jpg  Pictures  python_games  Videos
```

FIGURE 1.5: Finding the test image in the current directory

"Timed Out" Error

If `fswebcam` isn't working and you're getting the error message `Timed out waiting for frame! No frames captured`, we have a solution. The problem is that the system can't tell what resolution the camera is. To fix this, just force the resolution of the camera in any call of `fswebcam` as follows:

```
fswebcam -r 320x240 image.jpg
```

If you are working through the Pi's GUI, it's a good idea to open the test image and check to make sure that it is passable. Not all cameras are the same in terms of quality, and a cheaper model may be more prone to taking unusable photos. That being

said, once you see your image and are able to recognize what the image is of, your camera should be good enough for the next step.

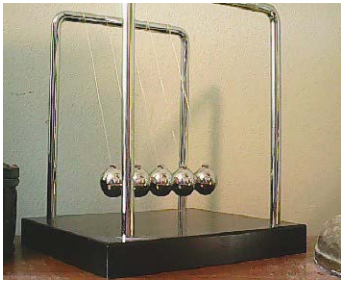


FIGURE 1.6: Test image captured with `fswebcam`

Overexposed

A common problem found when using `fswebcam` with cheaper USB cameras is that the photos taken have a tendency to be overexposed. Fortunately, `fswebcam` is capable of automatically adjusting its exposure by taking multiple frames. To tell `fswebcam` to do this, use the following:

```
fswebcam image2.jpg -F n
```

where *n* is the number of frames `fswebcam` is to take. Most people recommend taking 50 frames, though as few as 20 frames should result in passable images without taking up too much processing time.

GOING HEADLESS: ACCESSING YOUR PI

All Raspberry Pis start with the default network name/ID of `raspberrypi`, but you'll change your machine name. This makes it

easier to find (in case other people stick a Raspberry Pi on the same network) and more secure (since people won't automatically know the name of your machine). Start up the configuration program:

```
sudo raspi-config
```

Click Change Hostname and give your machine a new name. We suggest **`spypi`**, and that's the example name we'll use in this book. Once you set this, it always stays that way. And now, on the network, your machine can be found as `spypi.local` (thanks to a built-in package called Avahi, if you were curious).

So instead of going to, say, Google or Makermedia.com, you can point web browsers and other tools to `spypi.local` and find it. Bear in mind this is only *local*; you won't be able to connect to your Raspberry Pi if you're not on the same home or work network you plugged your Pi into. The rest of the Internet can't see your PI, and that's just what you want—private, not public, access.

We'll also make your Pi remotely accessible, so even if it isn't hooked up to a monitor and keyboard you can still access it from any Internet-connected device on the network with an SSH client (on Windows, PuTTY; on Mac/Linux, `ssh`) as long as you set the Pi to accept that. On the Pi, run `sudo raspi-config` and choose Interfaces, then choose SSH and "Yes, enable SSH." This allows remote logins from authenticated users who know the Pi username and password (as you do). The PuTTY or `ssh` client will let you remotely access your Pi, removing the need for a keyboard, mouse, and monitor.

To log into your Pi remotely, you'll use SSH:

* In Windows: Install PuTTY from *putty.org* (freeware), and then run the program. Enter **`spypi.local`** as the machine you want to log into, with the username **`pi`**.

* In macOS or Linux: Call up a terminal and log in with the following:

```
ssh -X pi@spypi.local
```

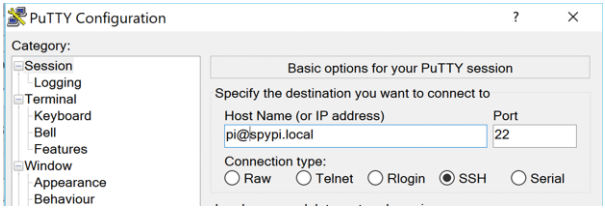


FIGURE 1.7: Connecting with PuTTY

In either case, you'll be prompted for your `pi` password. You can change that password on your Pi by again entering **`sudo raspi-config`** and using the Change Password option. (Part of the ease of the Pi is that controlling the various options is pretty simple.)

Using raspi-config

Perhaps in the future, beyond this book, you might get into adding attachments and wires to your Pi. When you add switches, sensors, motors, and other systems, the instructions will generally say something like "Enable the I²C interface" or "Make the serial port accessible." Again, those are just options in the raspi-config tool that you can turn on or off. They are turned off by default to make your Pi a little more secure. A core rule of computer security is to *not* turn on a service or software or port if you aren't actively using it, because anything "open" is also a potential opening a malicious program can exploit. The Pi smartly keeps most options turned off, and then lets you easily toggle options with raspi-config as needed.

Multiple Cameras

Yes, you can put more than one USB camera on a Pi—as many as there are open USB ports, in fact. The `fswebcam` program defaults to the camera it finds using the device location `/dev/video0`, but you can add other cameras. Once they are plugged in and (hopefully) auto-recognized, see which ones exist by entering

```
ls /dev/video*
```

Sample output for two cameras might look like

```
/dev/video0 /dev/video1
```

And now you can use `fswebcam` to point to a specific camera like this:

```
fswebcam -d /dev/video0
fswebcam -d /dev/video1
```

Easy!

Using the Pi Ribbon Camera

You can buy a special camera just for the Pi and use it instead of a USB webcam. It plugs into the Pi using a ribbon cable. To access it, instead of using `fswebcam` you can use the built-in utilities `raspistill` (for single pictures) and `raspivid` (for video).

2

Time-Lapse Imaging Viewed via the Web

The official title of this project is "A Surveillance Pi with Time-Lapse Image Capture Accessed Remotely via the Web," but we'll stick with the SpyPi name. In the first chapter we got the Raspberry Pi to produce a single image; now we're going to have it do some bona fide surveillance. You'll set up your SpyPi to automatically take a picture every minute and also make a web page so that you can remotely access the most current image at any time. And we'll talk about how you can combine the images to make a single time-lapse animated GIF of the activity!

You'll create all of this via the terminal, as in the previous chapter. In this case, you'll write scripts using the nano editor, save them, and then run them with the Python language interpreter. Meanwhile, you'll set up the web server in the background. Finally, you'll use a stand-alone tool to make the animated GIF.

A typical use case would be to set up your Pi in a forest or over an area where you need to maintain security, or even to capture traffic at a busy street intersection. Really, the legal uses are up to you. Everything we make is configurable, so you can make your system take one picture of road traffic every second, or one picture of the sunset every minute, or one picture of your house every hour. Best of all, you don't need to be near your Pi to view these images; just point your web browser to the Raspberry Pi's web page and watch the images from the luxury of your own home.

FIGURE 2.1: Time series of a SpyPi surveillance of our basement floor, showing a stuffed animal crossing. (Image brightness is due to camera.)

CAPTURING TIME SEQUENCES AUTOMATICALLY

First, you'll write a short script file using the Python language to automatically run your camera with a regular time sequence. To do this, you'll use the nano editor to enter the program. Type the following commands into the terminal:

```
nano timelapse.py
```

And enter the following code exactly as you see it. Python is *very* touchy about indentations—be sure to indent the code exactly as shown here. We'll use `60 seconds` (1 minute) as a delay time, but you can change that value in the code to whatever you need. Lines preceded by `#` are comments that explain the code.

```
# Spy-Pi code version 1
# this code takes an image every minute (60 seconds),
# and overwrites the current image with the new one
```

```
import os
import time
# choose a delay time in seconds by modifying the next line
delay = 60
# below are the default location to put the files, and the name to
use
directory = "/home/pi/"
filename = "spycam"
stem = ".jpg"
webfile = directory + filename + stem
# and this is the command to run.
mycommand = "fswebcam -d /dev/video0 -r 640x480 --no-banner"
# this is the actual 'do stuff' part. It runs forever
while True:
    runme = mycommand + " " + webfile
    os.system(runme)
    time.sleep(delay)
```

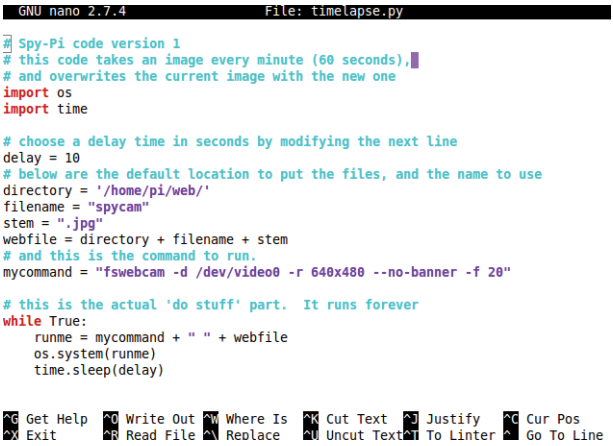


FIGURE 2.2: What you'll see in nano when you make this first Python script

This is a general structure we'll use often in this book—using Python as a wrapper around our external programs like `fswebcam`. We'll later add in a package called openCV for facial recognition. Python is easy to learn and modify, and it will let us set time-lapse loops and other tricks.

Once you've typed in this program, save it by pressing Ctrl-X and name it **`timelapse.py`**. At the terminal prompt run it with

`python timelapse.py`. Your gadget is now taking a new picture every 10 seconds, overwriting the old one. You can see the image on the desktop or look at it within the window by typing **`feh spycam.jpg`**.

FIGURE 2.3: Our first image: a stuffed animal on a piano!

This code will run forever, as long as the Raspberry Pi has power. You can stop it at any time, fortunately, by pressing Ctrl-C. There's also a fancy way to run it in the background so that you can type other things in the terminal—when you call it up, add an `&`, as in

```
python timelapse.py &
```

Don't do this yet. In the next chapter, we'll talk about using a routine called `crontab` to automatically run your code in the background. If you do want to play around with using `&` to run the program in the background, know that if you type **`fg`** it will bring the program up again, at which point you can then press Ctrl-C and kill it. Playing around with running scripts in the background or bringing them to the foreground so you can nix them is a very common Pi/Linux way to work on and run scripts.

MAKE THE PI INTO A WEB SERVER

To make your Pi into a web server so that you can point a browser to it, you...well, actually, you already did this. Back in Chapter 1 when you typed **`sudo apt-get install apache2 -y`** you told the Raspberry Pi to install a complete web server package. We're just

going to make it easier to modify things. (You can also read more at *www.raspberrypi.org/documentation/remote-access/web-server/apache.md*.)

Here's your default homepage, if you didn't change your Pi's name:

http://raspberrypi.local/

If you did change it as we recommended, it's

http://spypi.local/

and, if you are on your Pi, it can also see the page as

http://localhost/

FIGURE 2.4: The default web page auto-installed on your Pi (that you'll soon delete)

To make life easier, we'll show you how to make a link to where the Pi stores web pages so it will be easy for you to drop stuff into your home directory. Type these two lines

```
ln -s /var/www/html /home/pi/web
sudo chown pi . *
```

to make a new directory in your home directory called `web`, which is the location where any web page is stored. The `chown` line makes it so you can write and edit stuff in that (otherwise, it'll tell you `permission denied`). Anything you put into the `web` directory will be visible to the outside world via a browser, if the individual knows the address of your Raspberry Pi. There's already a default page called `index.html` there. Let's change it!

Go to the web directory:

```
cd web
```

Move the original `index.html` file to `index-orig.html`:

```
mv index.html index-orig.html
```

And create a new `index.html` file:

```
nano index.html
```

And now that you've invoked the nano editor, type the following into the file:

```
Hi
```

and then save the file and look at it in a browser.

FIGURE 2.5: The world's simplest web page

Okay, it's a very boring web page, but as you can see, anything you put into `index.html` appears to your web visitors.

Now you'll make a new index page that works with the camera. You'll also make a new Python script that automatically triggers the camera. Together, the two work like this: Every 1 minute, the Python script tells the Pi "Take a new picture and stick it in the `web` directory." Meanwhile, people visiting your new Pi web page will see the most recent picture—and their browser will automatically refresh itself every minute so it can catch when the picture changes.

Also, that 1 minute is an arbitrary time setting that you can change. If you think what the Pi can see will change faster or slower, make it a different time (every 5 minutes, every 30 seconds, every 5 seconds). Similarly, you can change the web page to update more rapidly—but beware, it takes time to load an image over the network via the web, so any web page that refreshes faster than 1 minute might mean your browser hasn't even finished loading a picture before the next refresh happens. Put another way, you should not take pictures faster than you can serve them.

First, modify our previous `timelapse.py` script by changing the one line that reads

```
directory = '/home/pi/'
```

to instead read

```
directory = '/home/pi/web/'
```

This will make the camera program write the image file into the web-visible area.

We now need to put an HTML web file in that web area that points to the picture. At the terminal prompt, type these two lines:

```
cd web
nano index.html
```

and then, when nano comes up, type this in the editor:

```
<html>
<head><title>SpyPi</title>
<meta http-equiv="refresh" content="60" />
</head>
<body>
<img src='spycam.jpg' alt='pi image'>
Image refreshes every minute!
</body>
</html>
```

Now save this code (in nano, press Ctrl-X and name the file **`index.html`**). Feel free to change `SpyPi` in the title and the `Image refreshes every minute!` text. That's just text for this demo and it can be changed to read anything you want. The key part is the meta keyword in the `head` tag, which sets the auto-refresh to every 60 seconds. This triggers any visiting web browser to reload the page every minute. You can set any time interval; one minute is useful for tracking slow state changes such as the camera being moved. Set it longer if you find it takes a long time to load an image in your browser.

Let's give it a try. Start the infinitely looping Python script:

```
python timelapse.py
```

and look at what's in the web directory by pointing your smartphone or tablet to the web page *http://spypi.local/*.

You should see what the Pi sees in your web browser. It's the same test image as before, but now you can access it via your web browser from a remote machine. Success!

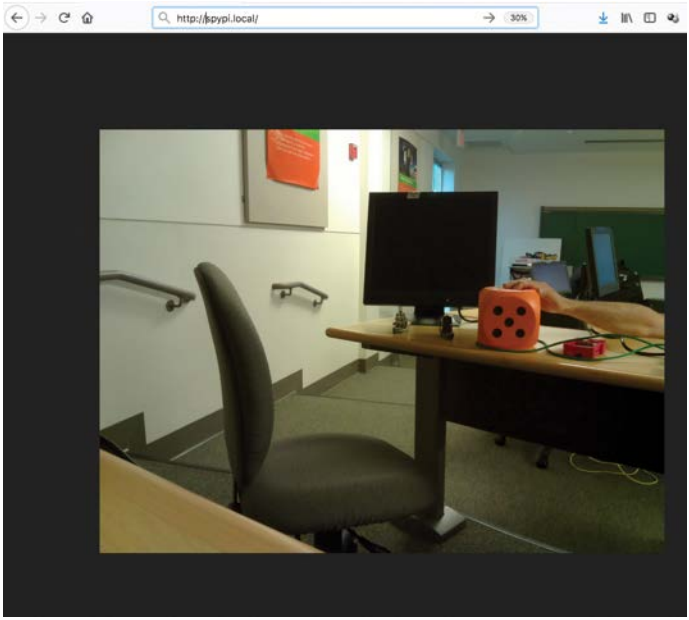


FIGURE 2.6: Sample image from our SpyPi, framed in your web browser

THE PI AS A REMOTE WIFI HOTSPOT

Right now, the Pi is running the web server and technically, the web pages are visible to any other machine on its network. You can find it from other machines via a web browser at *http://spypi.local/*.

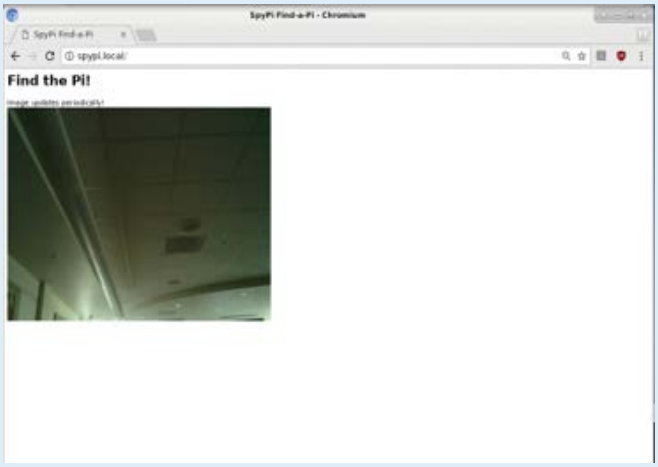
This is great for a SpyPi that is able to log into your local WiFi network. You can use the SpyPi at home or work, and you'll be able to check it from any machine on that network. You can place the Pi so that it creates its own hotspot that enables people to log into it, even if the Pi can't connect to an external network. This is great if you want to, say, place it in a forest to monitor wildlife and still want to be able to log into it remotely.

The "Find the Pi" Game

With the basic rig you've built, you now can play Find the Pi. One person hides the Raspberry Pi rig—the Pi, camera, and power supply, powered up and serving images—somewhere in a given area, such as a park, a school, or a backyard, for instance. The other players log into it with their smartphones or tablets and, based purely on the image they see from the Pi, try to figure out what the Pi is looking at and therefore where the Pi must be hidden.

Good hiding places should be somewhat generic: players must be able to see some clue that creates a puzzle. For example, a totally dark image is useless, but a Pi that can barely see an "exit" sign or indoor plant gives players a chance.

The person who finds the Pi can then choose a new hiding place for it, and the game continues. Here's an example of a picture that makes you think, "I bet I could walk around the author's basement and figure out where the Pi is looking out."



In this section, we're going to set up the Raspberry Pi so it doesn't have to be connected to an existing network. Instead, the Pi itself will create a WiFi hotspot that you can log into. This means anyone close enough to be in WiFi range of the Pi can connect to it.

To access your Pi, even without an Internet connection, follow any of these three tutorials:

www.raspberrypi.org/documentation/configuration/wireless/access-point.md

www.instructables.com/id/Make-Your-Pi-a-Local-Cloud-Server/

https://thepi.io/how-to-use-your-raspberry-pi-as-a-wireless-access-point/

The software installed is two packages called `HostAPD` (that makes your Pi a stand-alone Internet device) and `DHCP` (a network standard for automatically assigning addresses and names to devices that connect to your Pi). We'll also toss in a firewall called `iptables`. You'll install them with the usual:

```
sudo apt-get install hostapd isc-dhcp-server iptables-persistent
```

and then follow any of the tutorial links. Once done, you can log into your Pi as if it were its own Internet.

CLEANING AND ARCHIVING

At this point you are always serving up the current image. However, archiving past images moves the SpyPi past "toy" into "genuine surveillance." Let's modify our Python script to save the previous `N` days' worth of images, where `N` is however many days you want to keep.

Type **nano timelapse2.py** in the terminal, and enter the following code as is. Remember to maintain the proper indentations.

```
# Spy-Pi code version 2
# takes an image every minute, and saves 1 week (7 days) of data
# it also makes the most current image available
import os
import time
import shutil
# choose a delay time in seconds by modifying the next line
delay = 60
# this says to save 7 days (1 week) at a time, feel free to change
to a different duration
savedays = 7
# do not change this math code, it converts our savedays to how
often to retake a pic in secs
# note that 1 day at 1 picture/minute is 10,080 files per week
waitcount = savedays * 24 * 60 * 60 / delay
# below are the default location to put the files, and the name to
use
directory = '/home/pi/web/'
filename = "spycam"
stem = ".jpg"
# also, this is the file the webserver expects
webfile = directory + filename + stem
# and this is the command to run.
mycommand = "fswebcam -d /dev/video0 -r 640x480 --no-banner"
# this is the actual 'do stuff' part. It runs forever
icount = 0
while True:
    icount += 1
    myfile = directory + filename + "_" + str(icount) + stem
    runme = mycommand + " " + myfile
    os.system(runme)
    # copy new file to the webfile location so it is web-visible
    shutil.copy(myfile, webfile)
    # new purge routine removes older images after expiration time
    inix = icount - waitcount
    if inix > 0:
        deleteme = directory + str(inix) + filename + stem
        os.remove(deleteme)
        time.sleep(delay)
```

Again, running `python timelapse2.py` will do three things:

1. Take a picture at the given interval (here, 1 minute).
2. Save all the pictures it takes, up to the given period (here, 1 week), deleting older ones.
3. As written, it restarts the image count at `0` whenever you power up, so you lose old data and start afresh each time.

You may want to clean up when you stop or restart the time-lapse routine. These two commands, typed into the terminal, will get rid of all old files:

```
cd web
rm spycam*.jpg
```

Now you can rerun `python timelapse.py` without the clutter from previous runs.

Using the Pi Ribbon Camera?

If you are using the Pi ribbon camera instead of a USB web camera, just change the call from `fswebcam` (which automatically finds the USB camera) to `raspistill` (which automatically uses the Pi ribbon camera).

ANIMATING A TIME-LAPSE GIF

To view the archive of images, log into your Pi and either download the set or view the images on the Pi. Or you can choose to make them into an animated time-lapse GIF! Type in your terminal the following sequence to build a GIF at any time.

First, if you didn't install `imagemagick` in Chapter 1, type this:

```
sudo apt-get install imagemagick
```

Now, using your terminal, type these commands to go into the directory with the files and turn all the images into a time-lapse animated GIF file. Note the unit: `delay` is in hundredths of a second, so `10` = 0.1 second between frames, and `loop 0` = loop infinitely:

```
cd web
convert -delay 10 -loop 0 spycam*.jpg timelapse.gif
```

Let's look at it via the web. As before, you need to make a web page that calls it. So in the `web` directory, make the file `timelapse.html`:

```
cd web
nano timelapse.html
```

In that file, put these lines:

```
<html><body>
<img src="timelapse.gif" alt="timelapse">
</body></html>
```

Now, with your web browser, point to

http://spypi.local/timelapse.html

And you can see your amazing animated time-lapse image! The following is a tiled version of eight time-lapse images.

FIGURE 2.7: Tiled version of eight consecutive SpyPi time-lapse images

3

Pi Motion Detection

In this chapter, we will be discussing how to turn your Raspberry Pi into a motion-detecting surveillance system. This is simpler to accomplish than it sounds, since the software we need is ready in a downloadable package for the Pi. Once it's installed, we can dive right into operating and fine-tuning the program.

As with the previous project, we'll be working mostly from within the terminal, though this chapter will cover how to view the program's output with the GUI. We will be using the nano editor to help customize our program to suit our needs.

Motion detection allows for more selective and sophisticated image capturing than just taking a photo every 10 seconds. Now you can set up a wildlife camera that takes routine snapshots *and* automatically records video whenever an animal walks by. We'll also show you how to set up filters so you can quickly find who's been knocking on your front door without having to look through scores of images with the same view of your welcome mat. Last but not least, you can connect your motion-detection system to the Internet and watch what it sees wherever you are!

WHAT IS MOTION DETECTION?

Motion detection, in essence, is the search for changes in an environment caused by an object moving. A program does this by comparing two sets of input data about the environment and deciding whether there is a significant difference between them. In this chapter, we will compare literal images to try to detect whenever something has moved.

It's important to understand how one image is compared to the last taken. Essentially, the program counts how many pixels have changed since the last image. With this in mind, we must set a *threshold* for the differences to be reported. If the threshold is set at a low percentage of changed pixels, then the system will flag minor events like branches or papers rustling in a breeze as movement. If the threshold is set higher, it will take a more drastic change to count as movement.

FIGURE 3.1: Environment before movement

FIGURE 3.2: Environment after movement

But not all changes to an image are caused by movement. Suppose your camera is looking at a deserted street at twilight. At the appropriate moment, the street lights will flicker to life. The scene will change drastically—formerly dark shadows will become warm pools of light. But does this count as movement?

This is where *sensitivity* comes in. Sensitivity refers to certain parameters designed to help reduce the number of false positives reported by the system. If sensitivity is not taken into consideration, the system will report any time there's *any* significant change between images. This is fine when someone walks in or out of a room—but not when the light in the room is turned off. We can screen out such events by including a contrast setting with the camera.

HARDWARE

This chapter uses the same hardware listed in Chapter 1.

SET UP MOTION DETECTION

You will need to download the software needed for motion detection. If you didn't do so in Chapter 1, enter this command:

```
sudo apt-get install motion
```

While the packages are downloading, a prompt will come up asking if you wish to continue with the installation. Just press Y and then Enter to finish the download.

TESTING *MOTION* WITH GUI

The `motion` package is fairly ready to use once it's installed. The easiest way to test whether the program is operational is to work

using the Raspberry GUI. To get the package running, first enter the command:

```
sudo motion
```

This command will start `motion` and have it record images. To view what your camera is seeing, open the Pi's web browser and type in the address **`http://localhost:8081`**. This will establish a connection to `motion` and display its images in the browser.

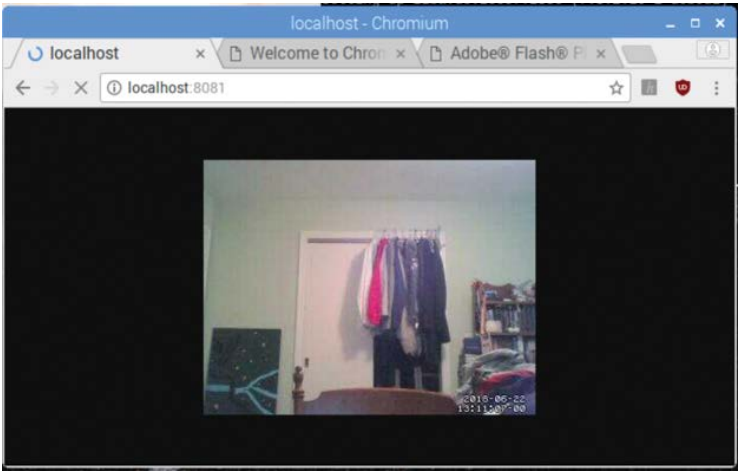


FIGURE 3.3: Streaming `motion` from `localhost`

When you first connect to the service, don't be alarmed if the images are small or the frame rate seems slow. These are the default settings assigned to `motion`. We'll go over how to reconfigure the service in the next section.

TESTING *MOTION* WITH THE COMMAND LINE

If you are only using the command line, testing `motion` isn't too complicated. The first thing you need to do is start `motion`:

```
sudo motion
```

Now, because you don't have the web browser to watch the live feed from the camera, let the program run for a few seconds before signaling it to stop. After running `motion` for about 30 seconds, enter the following to kill the program:

```
sudo service motion stop
```

Once `motion` is stopped, you can then try to look up the images it took while it was running. By default, `motion` should save its images to the current working directory. To check, type the following:

```
ls
```

Once you execute this command, you should see a list of JPEG files. Most are named with a time code followed by the word `snapshot`. These are the snapshot images that `motion` takes at a set rate to look for changes. There should also be an image named `lastsnap.jpg`. This is the latest image taken by `motion` and is updated each time `motion` captures a new photo. It is also what you see when viewing `motion` in the browser.

The point of this test is to make sure `motion` is running as it should in its default state. If the program is displaying what it sees in the browser and it is writing files as described, then that means `motion` is running fine. This test also acts as a quick introduction into how to operate the program. You're ready to move to the next step!

CUSTOMIZING *MOTION*

We mentioned earlier that it is possible to customize the settings in `motion` in order to have the program suit your needs. Fortunately, almost all these changes can be made and applied in a single file: `motion.conf`.

To access `motion.conf`, open the file using nano:

```
sudo nano /etc/motion/motion.conf
```

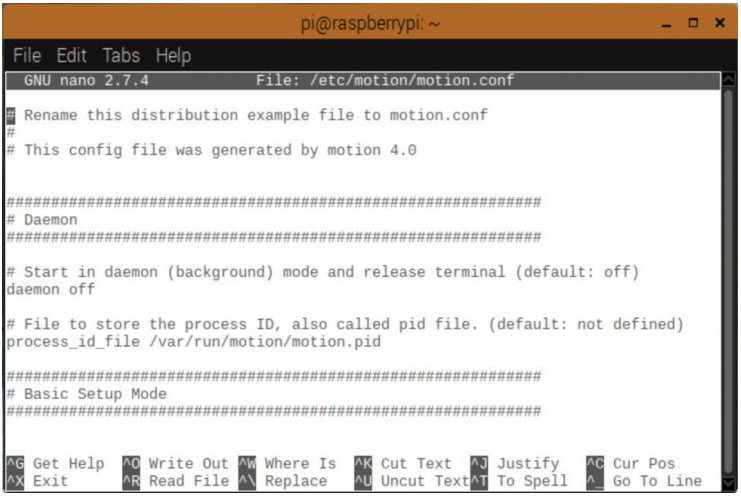


FIGURE 3.4: `motion.conf` opened in nano

The configuration file for `motion` is lengthy to say the least, but don't be daunted. The different settings are documented and explained within the file, and if you need additional information, you can find plenty of online resources.

One of the perks of using nano to look at the file is that you can search for keywords by pressing Ctrl-W. This makes navigating the file and finding specific settings much easier than having to scroll down through the various configurations.

Let's focus on some of the more basic options available.

NOTE **It is important to keep track of changes you make in the configuration file. Needs vary from project to project and mistakes are always a possibility, so unless you like the idea of combing through the entire file for that one line of code, a record of what you changed and what you changed it from is a good thing to have on hand.**

A good place to start in `motion.conf` is the `height` and `width` values. As you saw, the default image in the browser is a bit on the small side. That's because the default values of `height` and `width` are 240 and 320 pixels, respectively. If you want to view a larger image, you simply need to replace those values with larger numbers.

> **NOTE** **To avoid weird dimensions with your images, you should maintain the ratio between height and width (3:4). Also, the valid range of dimensions is limited by your camera: you're never going to get a 1280×960 image from a camera with a maximum resolution of 640×480.**

For practice, let's double `height` and `width`. Find those lines in the file and edit them to say

```
width 640
height 480
```

Once you've specified the new dimensions, exit the file to test them. To save the changes you made, press Ctrl-X followed by Y, and then press Enter to reach the command line. Run `motion` as you did in the previous section and see whether the image is larger.

If the image in the browser is the same as before, you may need to reboot the Pi to have the new settings take effect. To initiate a reboot, at the command line type

```
sudo reboot
```

Once the Pi is rebooted and you are logged back in, you should be able to start `motion` and see the changes.

One other value we should look at is `threshold`. The `threshold` value determines how many pixels need to have changed

between two snapshots to trigger an event. By default, `threshold` is set to trigger if 1,500 pixels have been changed.

Now recall what we said about the main principles of motion detection at the beginning of this chapter. Although 1,500 may sound like a high number, that is 1,500 pixels out of the *entire* image. Using the default settings in `motion`, let's calculate what percentage 1,500 is of a 240×320 image:

$$240 \times 320 = 76{,}800$$

$$1{,}500 / 76{,}800 = 0.02$$

Using the default settings, `motion` triggers an event whenever there is a 2 percent difference between photos. That is far too sensitive! With a threshold that low, an event could be triggered by a fly buzzing by the camera. The sensitivity is increased even more for a 480 × 640 image: 1,500 pixels represents a threshold of 0.5 percent.

To make `motion` less sensitive and to help avoid drowning in false alarms, let's increase `threshold`. You can set `threshold` to whatever you want, but for the sake of this project, shoot to make it a manageable 25 percent. To find the correct number, simply do the following calculations with your doubled dimensions:

$$480 \times 640 = 307{,}200$$

$$0.25 \times 307{,}200 = 76{,}800$$

Now that you know how many pixels equal a quarter of the image, you can plug that number in the configuration file for `motion`:

```
sudo nano /etc/motion/motion.conf
```

Find `threshold` and edit the line to say

```
threshold 76800
```

Now `motion` will trigger an event when you wave your hand in front of the camera—but not when a bug flies around on the other side of the room.

The rest of the configuration settings for motion are certainly worth checking out and experimenting with, though doing so would be a bit outside the scope of this book. If you're interested in learning about the various settings, you can find a good reference here: *www.lavrsen.dk/foswiki/bin/view/Motion/ConfigFileOptions*.

STREAMING *MOTION* REMOTELY

Viewing `motion` remotely operates much like viewing it locally on the Pi. Once it's set up, you can stream what `motion` sees from any web-enabled device, be it a phone, a laptop, or even another Pi. Before you can do this, however, there are a few things you'll need.

First of all, you must enable remote streaming with `motion`. Start by opening `motion.conf`:

```
sudo nano /etc/motion/motion.conf
```

Once the configuration file is open, find the `stream_localhost` line. This value determines whether or not `motion` allows remote streaming. To enable streaming, change it to read

```
stream_localhost off
```

With streaming now enabled, you just need to find the Pi's IP address so that you can connect to it. This is easy to do by typing the following at the command line:

```
hostname -I
```

The output the command gives is the IP address of the Pi. Armed with this knowledge, you can now stream `motion` from a remote device. Making sure that the Pi is connected to the Internet, open the web browser on your device and type the Pi's IP address in the address bar, followed by the port number **`8081`**. The bar should resemble the following:

```
http://192.165.76.3:8081
```

Start `motion`:

```
sudo motion
```

Enter the IP address and your browser should look just as it did when you viewed `motion` locally on the Pi.

GOING HEADLESS

Streaming `motion` headlessly—that is, without connecting the Pi to a monitor or keyboard—is simple enough to do. You know how to watch the stream from `motion` remotely at this point, which is half the battle. The other half is getting `motion` to run while the Pi is headless.

There are two ways to accomplish this. The first is to connect to the Pi from a remote computer via SSH and tell `motion` to run.

The other way is to have `motion` run automatically when the Pi powers up. One of the best ways to do this is by using `crontab`. The `crontab` (short for cron table) file is used for scheduling different commands and programs so that they run automatically without the system administrator having to enter the same command every day.

To use `crontab`, first open the file for editing:

```
sudo crontab -e
```

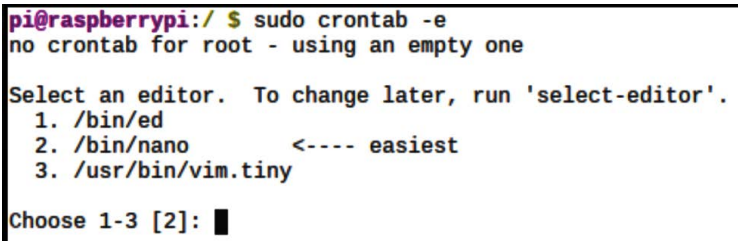


FIGURE 3.5: Selecting an editor for `crontab`

When you first open `crontab`, the system will ask you which text editor you wish to use. At the prompt, select nano by typing **2** and pressing Enter. The `crontab` file should then open in the command window.

Since you want to run `motion` at startup and not at any specific time, we will ignore the traditional scheduling format. Instead, scroll to the bottom of the file and enter the following:

```
@reboot motion
```

Now `motion` will run every time you boot up your Pi. Save your edit and exit the file.

To test and make sure the changes you made work, reboot your Pi:

```
sudo reboot
```

Then try to stream `motion` via the web browser. If everything is operational, you should see the feed from `motion` in the window without having to start the program manually.

4

Machine Learning: Identifying People in Images

So you've posted a photo to your favorite social media site and it suddenly puts little boxes around each person's face and asks you to tag who they are. The first part—recognizing there's a face in the image—is facial detection. The second part is facial recognition—putting a name to a face. The reason the site is asking you to name the people in the image? That's so it can learn to recognize that face in other pictures. Next thing you know, it'll be auto-suggesting people in other pictures—Facebook, for example, is learning faces.

Now that you have a smart camera set up, we're going to add both facial detection ("there is a human face in the picture") and facial recognition ("and that face is my face!"). There are two steps to this. First, we'll install software that can detect whether

there is a face-shaped object in the image (facial detection); then we'll try to extract that object and see whose face it is (facial recognition).

The same software does both—the first operation is quick, and the second takes a little more effort. Just as with the previous projects, the software for doing the hard work is easy to

OpenCV

OpenCV, also known as the Open Source Computer Vision Library (at *https://opencv.org*), is a library of over 2,500 image-handling algorithms. They can be used for everything from interactive art, to mines inspection, to stitching maps on the web or to advanced robotics. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high-resolution image of an entire scene, find similar images from an image database, remove red-eye from images taken using flash, follow eye movements, and recognize scenery and establish markers to overlay it with augmented reality.

In short, it's free industry-grade software that is easily set up via Python. We'll use OpenCV to do facial detection as well as facial recognition. If you decide to delve deeper into image processing, the best part is that OpenCV can do a lot more than just faces. If you want to dig deeper on the background theory and advanced capabilities of machine learning as it applies to images, you can read up on its full abilities at *https://opencv.org*. We'll work off their specific section on facial recognition here:

https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html

install. It's a package called OpenCV, and you probably already installed it in Chapter 1. But if not, install it with

```
sudo apt-get install opencv
pip install numpy
```

(Numpy is a numerical library that is required by and speeds up OpenCV.)

We'll pull out the core parts of the tutorial for our specific SpyPi work. To begin, we'll get going with the core functionality for your Pi system. After that, if you want to explore deeper, you have all the tools you'll need in place to do so.

STEP 1: QUICK FACE BOXING

Let's start by downloading a ready-to-go facial recognition script (from a user named Shantnu Tiwari, username shantnu) that finds faces and draws boxes around them in real time. Navigate to this website from your Pi:

https://github.com/shantnu/Webcam-Face-Detect

Click the Download button to download the zip file. Then, in your terminal, unpack it:

```
unzip ../Downloads/Webcam-Face-Detect-master.zip
```

Then, to run it:

```
cd Webcam-Face-Detect-master
python webcam.py
```

Figure 4.1 shows the code in that `webcam.py` file.

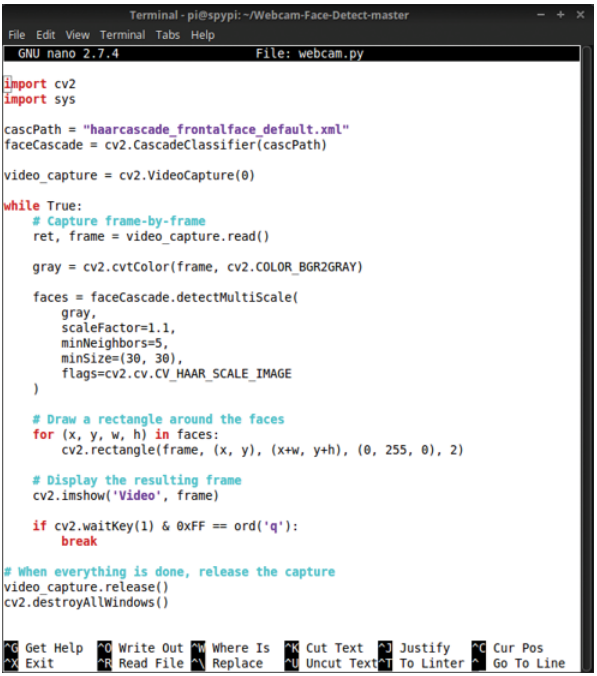


FIGURE 4.1: Code from `Webcam-Face-Detect-master/webcam.py`

That's it. You will now see the view from your webcam, and as the software detects a face, it will draw a box around the face. The video is choppy, with slight delays between what you do and what it shows; this is because it takes a while for the software to run the facial detection. We'll tackle what the software is doing and add coolness to it, but this script is a great way to show how easily you can do facial recognition on the Pi.

NOTE Shantnu Tiwari, who provided this initial face-detection script, has other code bits you can run on his GitHub site (*https://github.com/shantnu/*). He runs the website *pythonforengineers.com*, which includes the free online book *Python for Scientists and Engineers*, as well as other ebooks he's written.

Figure 4.2 shows the first result, the author "found" by the software.

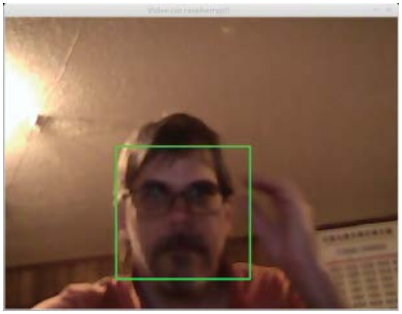


FIGURE 4.2: Author's face found and boxed by SpyPi

It runs without you needing to understand it, but where is the fun in that? Let's go deeper. First, look at the script (click it to open it in an editor, or if you use the terminal, type **`less webcam.py`**). The script is very short and written in Python; it loads the `opencv` library that handles the image processing as well as an XML file that contains mathematical definitions created by researchers that define what a computer will recognize as "a face." The script then calls up the web camera and runs a loop to apply the face-finding algorithm.

The first five lines of text set up the OpenCV package and the XML file. The `while True:` line says "loop forever." The lines marked with `#` are comments that describe what the following lines in the code do. First the script captures a frame; then it draws a rectangle around the frame and displays that frame in the rectangle. The final two lines do cleanup. If you press Q at any point during the demo, the program exits, which is handy (and means you don't need to press Ctrl-C as in earlier chapters).

This script is good enough to tell a person from a stuffed animal or pet, as you can see in Figure 4.3. The accuracy of facial recognition software (even in industrial uses) is still a work in

progress, so we'll look at the sorts of errors you can expect, as well as let you try different methods.

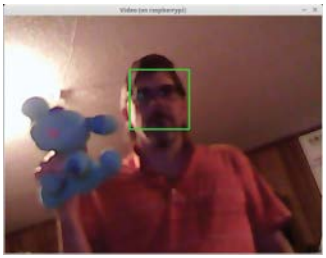


FIGURE 4.3: Face-finding test: person vs. stuffed animal. The software correctly found the person and did not match on the stuffed animal.

An interesting point about any image detection is that you get to choose between a false positive—it matches a face even if one isn't there—and a false negative—there's a face but it doesn't find it. Let's look at some image examples of this basic face-finding problem. Figure 4.4 contains both a false positive—it "finds" a face in some wall tiles that isn't really a person—and a false negative—it fails to find the author's face.

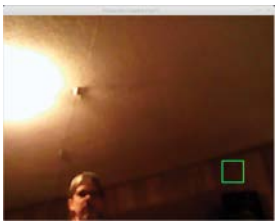


FIGURE 4.4: False positive (box but no face) and false negative (face and no box)

It's also worth noting that software relies on patterns, so if the only thing in the field of view has some face-like aspects, such as eyes, it might find a match. If a stuffed animal is the only thing in the camera's field of view, it's not surprising if the software sometimes (but not always) tries to tag it as a face. We do this as

humans—draw any two dots plus a line, and we'll see a face. This phenomenon, called *pareidolia*, means humans tend to see faces in objects. Computer code can have the same problem. A classic example is the face seen in an outlet (Figure 4.5).

FIGURE 4.5: Pareidolia—seeing faces in inanimate objects

We can run into this phenomenon with software, particularly if we're trying to fool it. The software doesn't know what a "person" or "object" is, and it's looking for patterns. If you mess up the patterns—put on Juggalo makeup, for example—you can confuse facial recognition. Likewise, an object such as a toy made to look like it has a face might find the software latching onto part of its face-like structure to falsely match a face.

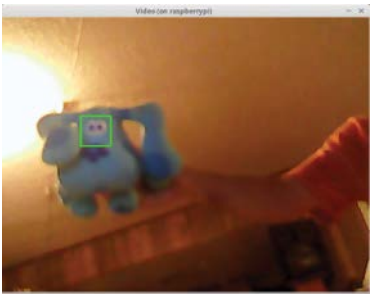


FIGURE 4.6: The stuffed animal region around the eyes fools the software into thinking it's a face.

Either way, you now have real-time facial detection working on your SpyPi. Let's build on this to do more.

Face Algorithms

OpenCV can automatically find a face-like object using one of two built-in methods. Why two? One method (the Haar feature) doesn't find faces easily, but when it finds a face you can trust that it's actually a face (which means fewer false positives). The other method (LBP) is faster and finds more faces, but it also is more likely to tag something as a face that isn't really a face (which means more false positives). Since the code we started with uses Haar, we'll keep going with that.

A LITTLE IMAGE THEORY

How do you recognize someone? The most common human way is to notice a specific feature or set of features that identifies a specific person: that person with the dark hair and big nose, that other person with the eyeglasses and red lips. We latch onto these for quick identification.

When you do this with a group of people you've just met, you compare them to one another. It's no good saying "she's the one with dark hair" if everyone in the group has dark hair, so you make quick comparatives for just that group—the one redhead, the one with the narrow eyes, the one with glasses. This kind of quick sorting out of a group is the Eigenfaces method in OpenCV—within a group of known photos, it's how you tell one from another. The downside of this approach is that you can later get confused if a person changes her hairstyle or takes off his glasses (Clark Kent style!), or even if we see that person in different lighting.

More often, you'll want to make a list of features that let you distinguish any face from the others: not just "the person with glasses" but a checklist—"Glasses: yes/no. Dark hair: yes/no" and so on. Then each person has a profile of features that you can use to compare with others. That's the Fisherfaces method in OpenCV. Sure, the computer isn't smart enough to know what "glasses" or "hair" are, but it can teach itself that the "dark blob at the top of the face-shape" is different from the "light blob at the top of the face-shape"—and that's practically the same thing for our purposes.

Finally, you can go full-on machine thinking and instead of extracting components of faces to make a checklist, you let the computer do a pixel-by-pixel mapping of how faces change as you scan left to right and up and down. So where we see an "eye," the computer just says, "There is a white zone with a non-white dot in the middle." That's the Local Binary Pattern Histogram (LBPH) method in OpenCV.

In Python, these modules are called as follows:

Eigenfaces: `createEigenFaceRecognizer()`

Fisherfaces: `createFisherFaceRecognizer()`

Local Binary Patterns Histograms: `createLBPHFaceRecognizer()`

These are the three different underlying algorithms available in OpenCV, each with its own pluses and minuses. We'll be using LBPH, but (as you'll see) you can change just one line in the code to use the other algorithms.

STEP 2: TRAINING YOUR CAMERA

Once you have an algorithm, you need to train your camera with the set of faces you think it will see. For example, if you want the camera to identify any visiting students and you have a set of student photos, you first make the software aware of what the students look like; then the camera can not only detect that a face exists but tag it and label it as whose face it is.

Now that your camera can detect a face, we want to make it tell you whether that's a specific known face, thus creating an ID system. To do this, you'll need an image of any and all faces you want it to recognize so that you can "train" your camera system to spot specific people.

So, take a bunch of pictures of the faces of people you want to enter in its system. You'll then train the system by loading those in. Once trained, the program can then automatically recognize known individuals.

An untrained system can still put a rectangle around a face and say "There is a face," but for identification purposes, training it is useful.

To get this started, first you're going to modify the previous code to make it easier to rework. We already did this and put it up on GitHub at

https://github.com/sandyfreelance/SpyPi

Go get it and put it on your Pi, and we'll walk through what it does. This code will 1) display a new image every second, 2) draw rectangles around faces, 3) match any faces it "knows," and 4)

let you also train it by telling it to add a face to its memory. The opening lines just set up the variables and load OpenCV and other programs:

```
import cv2
import sys
import numpy
import time
import pickle
cascPath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascPath)
face_recog = cv2.createLBPHFaceRecognizer()
vid_cap = cv2.VideoCapture(0)
all_faces = []  # list to store 'known' faces in
face_count = 0
savefile = 'webfaces.dat'
```

After that, the code asks if you want to reload previously saved "known" faces:

```
yn = raw_input("Do you want to import previously saved faces? y/n")
try:
        if yn[0] == "Y" or yn[0] == "y":
                infile = open(savefile,"rb")
                all_faces = pickle.load(infile)
                infile.close()
                face_count = len(all_faces)
                labels = range(face_count)
                face_recog.train(all_faces,numpy.array(labels))
                print face_count,"faces loaded"
except:
        print "No faces loaded"
        pass
```

Enough preamble.

STEP 3: IDENTIFYING WHOSE FACE YOU FOUND

The meat of the code starts with the rectangle-drawing functions from before and adds a little face recognition. First, we use the

same routines from the previous code that grab an image and draw a box around it:

```
def detect_face(vid_cap, all_faces, face_recog, face_count):
    faces_found = []  # empty holder for storing what we find
    # Capture frame-by-frame
    ret, frame = vid_cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
          gray,
          scaleFactor=1.1,
          minNeighbors=5,
          minSize=(30, 30),
          flags=cv2.cv.CV_HAAR_SCALE_IMAGE
          )
    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            # also can put a trigger here
```

What does the comment `#also can put trigger here` mean? This is where you can put code that triggers when any face is found, in case you want to make an automatic door opener or something similar. It's just a comment indicating that this is a good place to put instructions for facial detection (before they are recognized).

Now for the next bit. If we can draw a rectangle around a face, we can extract just the face and then compare it with any previous faces we might have stored in a list. We always convert faces to grayscale, which removes color imbalance concerns (and is also the format OpenCV expects). So do this:

```
# extract just the face as its own image
thisface = frame[y:y+w, x:x+h]
grayface = cv2.cvtColor(thisface, cv2.COLOR_BGR2GRAY)
faces_found.append(grayface)
```

Here's the fun part. It's only one line that compares a new face found with "known" faces, using OpenCV's `.predict` method:

```
       if face_count > 0:
          id = face_recog.predict(grayface)
          print "Found face number",id[0]
```

And to close off this part, we again display it onscreen and return any faces we found to our main calling program:

```
# Display the resulting frame
cv2.imshow('Video', frame)
return faces_found
```

That wraps up the code for the new subroutine. Next we'll put in some bookkeeping and add in the OpenCV functions that do the recognition. We again use a "keep looping until we tell you otherwise" construct, and we allow for two different keypresses:

Q quits the program.

S saves the currently rectangle-highlighted face as a "known" face.

So we train the system by pressing S when a known person is there, and then the system automatically matches it to any new faces found. You could move the training to its own program, but having the script allow you to both train and predict made it smaller and easier.

```
while True:
       faces_found = detect_face(vid_cap, all_faces, face_recog,
       face_count)
       checkme = cv2.waitKey(100)
       if checkme & 0xFF == ord('q'):
          print "Exiting"
          break
       if checkme & 0xFF == ord('s') and len(faces_found) > 0:
           # assumes we are storing one and only one face
           print "Storing a new face"
           face_count = face_count + 1
           all_faces.append(faces_found[0])
           labels = range(face_count)
           face_recog.train(all_faces,numpy.array(labels))
```

Yet again, all the real work is being done in the OpenCV `.train()` method call. You pass it a list of faces you want it to learn, and it handles the rest.

Now to wrap everything up by cleanly closing out our windows (once the user presses the Q key), and then also saving any faces you marked as known (by pressing S, perhaps multiple times):

```
# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
# also save any 'trained' faces
outfile = open(savefile,'wb')
pickle.dump(all_faces, outfile)
outfile.close()
```

How well does it work? In Figure 4.7 we pressed the S key to tell the software to store this fine face it found.

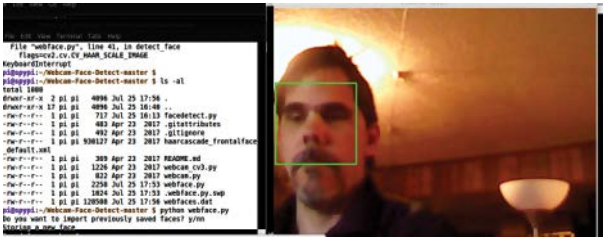


FIGURE 4.7: Training your SpyPi to recognize me (runtime on left, video capture on right)

Then I'll move around and see if it can still find me. It can! It keeps listing "Found face number 0" in its runtime window. You can also modify the code and replace that line with, well, whatever you want your Pi to do. You can have it trigger an alarm, send a message, and many other things, by combining other Jumpstart guides with this one. We provide the face recognition; you add the hardware.

So you can see how it saves trained data, Figure 4.8 shows a new run that loads the automatically saved data from the previous run. It still recognizes me, because now I'm in its saved trained data set.

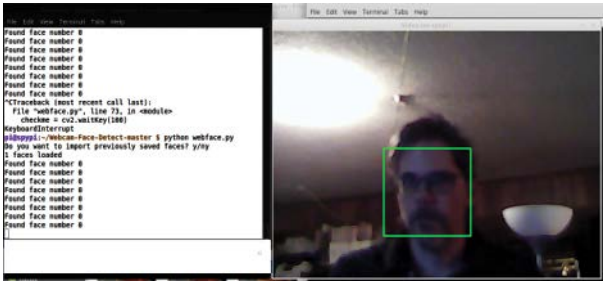


FIGURE 4.8: Trained system keeps finding me (runtime on left, video capture on right).

And, to complete the tests, we check whether I can fool it with a cunning disguise.

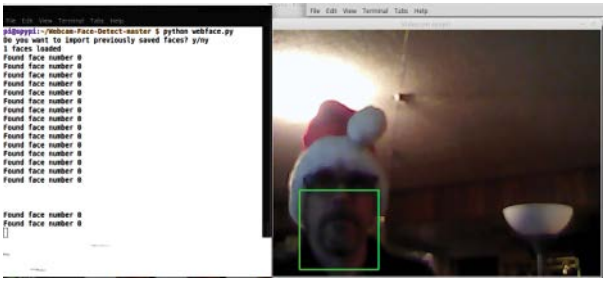


FIGURE 4.9: Even using a Santa hat does not fool the SpyPi face recognizer.

TRIGGERING AND FUTURE STEPS

Once you have categorized found faces as either known or unknown, you can add triggers to your code to make your Pi react. Maybe you want it to flash an LED (or lock a door) if it sees a face it doesn't recognize. Or flash an LED (or open a door) if it sees a face it does know. With the previous code and a little work in Python, you can add in these "triggered" effects by editing the code provided. Just take the line that prints "face not found" and have it call a Python routine to carry out your automated wishes.

Using the Pi Ribbon Camera

As usual, if you are using the Pi ribbon camera instead of a USB web camera, you just need to change a few lines in our examples to repoint our code to the correct camera. We're not going to rewrite all the code here, but you should be able to figure out the mods easily enough. To capture with a web camera, we were using code with this type of call:

```
camera = cv2.VideoCapture(0)
retval,image=camera.read()
if retval:
    cv2.imwrite('test.jpg',image)
```

To use the Pi ribbon camera instead, you use constructs of this form:

```
from picamera import *
camera = PiCamera()
camera.capture('image.jpg')
```

The `picamera` package is installed on the Pi automatically, and full documentation is at *http://picamera.readthedocs.io/*.

Put simply, once code can find a face, it can act on that knowledge. Building this system into a security system is one possibility. Saving only images of faces is another surveillance-type activity you can use it for. There's not a lot of use for, for example, one week of raw video, even if you can find faces. However, if you capture faces only when they appear and put that together, you have a video log of all the people who traveled past your system.

Add in our earlier motion-detection and time-lapse work, and you have a system that can capture time-lapse images only when there's activity involving people with faces:

* Have it take pictures only when there's motion (Chapter 3).

- For those pictures, if there's a face, have it save the image of the face (this chapter).
- Build those into a time-lapse movie of "people who visited us" (Chapter 2).

Faces are only the start. Using the same OpenCV libraries, you can perform recognition of other objects, edge detection, real-time color filtering or background subtraction, conversion to black and white, motion trails, and all sorts of other image manipulation (see *https://pythonprogramming.net/loading-images-python-opencv-tutorial/*)—even converting images in near real time to artistic styles like Van Gogh or A-ha/Take-On-Me stylings (see *https://larseidnes.com/2015/12/18/painting-videos-with-neural-networks/*).

At this point, armed with a Pi and some simple Python software, you can bend image reality to do everything from recognition to alteration. Enjoy capturing and manipulating visual data!