OpenCV 4 for Secret Agents

Second Edition

Use OpenCV 4 in secret projects to classify cats, reveal the unseen, and react to rogue drivers

Joseph Howse

Packt
www.packt.com

OpenCV 4 for Secret Agents
Second Edition

Use OpenCV 4 in secret projects to classify cats, reveal the unseen, and react to rogue drivers

Joseph Howse

Packt

BIRMINGHAM - MUMBAI

OpenCV 4 for Secret Agents
Second Edition

Commissioning Editor: Richa Tripathi
Acquisition Editor: Chaitanya Nair
Content Development Editors: Digvijay Bagul
Technical Editor: Riddesh Dawne
Copy Editor: Safis Editing
Project Coordinator: Prajakta Naik
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Graphics: Tom Scaria
Production Coordinator: Nilesh Mohite

First published: January 2015
Second edition: April 2019

Production reference: 1300419

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

ISBN 978-1-78934-536-0

www.packtpub.com

I dedicate my work to Sam, Jan, Bob, Bunny, and the cats, who have been my lifelong guides and companions.

Let us remember Plasma Tigerlily Zoya (2004-2017), a pioneer of feline computer vision, and a cat of great virtue.

– Joseph Howse

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the author

Joseph Howse lives in a Canadian fishing village with four cats; the cats like fish, but they prefer chicken.

Joseph provides computer vision expertise through his company, Nummist Media. His books include *OpenCV 4 for Secret Agents*, *OpenCV 3 Blueprints*, *Android Application Programming with OpenCV 3*, *iOS Application Development with OpenCV 3*, *Learning OpenCV 3 Computer Vision with Python*, and *Python Game Programming by Example*, published by Packt.

I want to thank the readers – such as Dan and Cindy Davis of Farmington, Utah – who shared their enthusiasm for this book's first edition and told me of their own adventures in computer vision. I am grateful to the editors, technical reviewers, and marketers of both editions, as well as my colleagues at Market Beat (El Salvador) and at General Motors who gave feedback on drafts. Above all, my family makes my work possible and I dedicate it to them.

About the reviewers

Christian Stehno studied computer science, receiving his diploma from Oldenburg University, Germany, in 2000. Since then, he's worked in different fields of computer science, first as a researcher on theoretical computer science at an academic institution, before switching later on to embedded system design at a research institute. In 2010, he started his own company, CoSynth, which develops embedded systems and intelligent cameras for industrial automation. In addition, he is a long-time member of the Irrlicht 3D engine developer team.

Arun Ponnusamy works as a computer vision research engineer at an AI start-up in India. He is a lifelong learner, passionate about image processing, computer vision, and machine learning. He is an engineering graduate from PSG College of Technology, Coimbatore. He started his career at MulticoreWare Inc., where he spent most of his time on image processing, OpenCV, software optimization, and GPU computing.

Arun loves to build his understanding of computer vision concepts clearly, allowing him to explain it in an intuitive way on his blog and in meetups. He has created an open source Python library for computer vision, named *cvlib*, which is aimed at simplicity and user friendliness. He is currently working on object detection, action recognition, and generative networks.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Section 2: Section 2: The Chase

Preface

Computer vision systems are deployed in the Arctic Ocean to spot icebergs at night. They are flown over the Amazon rainforest to create aerial maps of fires, blights, and illegal logging. They are set up in ports and airports worldwide to scan for suspects and contraband. They are sent to the depths of the Marianas Trench to guide autonomous submarines. They are used in operating rooms to help surgeons visualize the planned procedure and the patient's current condition. They are launched from battlefields as the steering systems of heat-seeking, anti-aircraft rockets.

We might seldom—or never—visit these places. However, stories often encourage us to imagine extreme environments and a person's dependence on tools in these unforgiving conditions. Perhaps fittingly, one of contemporary fiction's most popular characters is an almost ordinary man (handsome, but not *too* handsome; clever, but not *too* clever) who wears a suit, works for the British Government, always chooses the same drink, the same kind of woman, the same tone for delivering a pun, and is sent to do dangerous jobs with a peculiar collection of gadgets.

> *Bond. James Bond.*

This book discusses seriously useful technologies and techniques, with a healthy dose of inspiration from spy fiction. The Bond franchise is rich in ideas about detection, disguise, smart devices, image capture, and sometimes, even computer vision specifically. With imagination, plus dedication to learning new skills, we can become the next generation of gadget makers to rival Bond's engineer, Q!

Who this book is for

This book is for tinkerers (and spies) who want to make computer vision a practical and fun part of their lifestyle. You should already be comfortable with 2D graphical concepts, object-oriented languages, GUIs, networking, and the command line. This book does not assume experience with any specific libraries or platforms. Detailed instructions cover everything from setting up the development environment to deploying finished apps.

A desire to learn multiple technologies and techniques, and then integrate them, is highly beneficial! This book will help you branch out to understand several types of systems and application domains where computer vision is relevant, and will help you to apply several approaches to detect, recognize, track, and augment faces, objects, and motions.

What this book covers

`Chapter 1`, *Preparing for the Mission*, helps us to install OpenCV, a Python development environment, and an Android development environment on Windows, macOS, or Linux systems. In this chapter, we also install a Unity development environment on Windows or macOS.

`Chapter 2`, *Searching for Luxury Accommodations Worldwide*, helps us to classify images of real estate based on color schemes. Are we outside a luxury dwelling or inside a Stalinist apartment? In this chapter, we use the classifier in a search engine that labels its image results.

`Chapter 3`, *Training a Smart Alarm to Recognize the Villain and His Cat*, helps us to detect and recognize human faces and cat faces as a means of controlling an alarm. Has Ernst Stavro Blofeld returned, with his blue-eyed Angora cat?

`Chapter 4`, *Controlling a Phone App with Your Suave Gestures*, helps us to detect motion and recognize gestures as a means of controlling a guessing game on a smartphone. The phone knows why Bond is nodding, even if no one else does.

`Chapter 5`, *Equipping Your Car with a Rearview Camera and Hazard Detection*, helps us to detect car headlights, classify their color, estimate distances to them, and provide feedback to the driver. Is that car tailing us?

`Chapter 6`, *Creating a Physics Simulation Based on a Pen and Paper Sketch*, helps us to draw a ball-in-a-maze puzzle on paper, and see it come to life as a physics simulation on a smartphone. Physics and timing are everything!

`Chapter 7`, *Seeing a Heartbeat with a Motion-Amplifying Camera*, helps us to amplify motion in live video, in real time, so that a person's heartbeat and breathing become clearly visible. See the passion!

`Chapter 8`, *Stopping Time and Seeing like a Bee*, helps us improve the previous chapter's project by adopting specialized cameras for high-speed, infrared, or ultraviolet imaging. Surpass the limits of human vision!

`Appendix A`, *Making WxUtils.py Compatible with Raspberry Pi*, helps us solve a compatibility issue that affects the wxPython GUI library in some Raspberry Pi environments.

`Appendix B`, *Learning More about Feature Detection in OpenCV*, helps us discover more of OpenCV's feature-detection capabilities, beyond the ones we use in this book's projects.

`Appendix C`, *Running with Snakes (or, First Steps with Python)*, helps us learn to run Python code and test an OpenCV installation in a Python environment.

To get the most out of this book

This book supports several operating systems as development environments, including Windows 7 SP 1 or later, macOS X 10.7 (Lion) or later, Debian Jessie, Raspbian, Ubuntu 14.04 or later, Linux Mint 17 or later, Fedora 28 or later, **Red Hat Enterprise Linux** (**RHEL**) 8 or a later version, CentOS 8 or later, openSUSE Leap 42.3, openSUSE Leap 15.0 or later, and openSUSE Tumbleweed.

The book contains six projects with the following requirements:

- Four of these six projects run on Windows, macOS, or Linux, and require a webcam. Optionally, these projects can use Raspberry Pi or another single-board computer that runs Linux.
- One project runs on Android 5.0 (Lollipop) or a later version, and requires a front-facing camera (which most Android devices have).
- One project runs on Android 4.1 (Jelly Bean) or a later version, and requires a rear-facing camera and gravity sensor (which most Android devices have). For development, it requires a Windows or macOS machine and approximately $95 worth of game development software.

Setup instructions for all required libraries and tools are covered in the book. Optional setup instructions for Raspberry Pi are also included.

Download the example code files

You can download the example code files for this book from your account at `www.packtpub.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

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 book is also hosted on GitHub at `https://github.com/PacktPublishing/OpenCV-4-for-Secret-Agents-Second-Edition`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `http://www.packtpub.com/sites/default/files/downloads/9781789345360_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "You can edit `/etc/modules` to check whether `bcm2835-v4l2` is already listed there."

A block of code is set as follows:

```
set PYINSTALLER=pyinstaller

REM Remove any previous build of the app.
rmdir build /s /q
rmdir dist /s /q

REM Train the classifier.
python HistogramClassifier.py
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
            <activity
                android:name=".CameraActivity"
                android:screenOrientation="landscape"
                android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />

                    <category android:name="android.intent.category.LAUNCHER"
    />
                </intent-filter>
            </activity>
```

Any command-line input or output is written as follows:

```
$ echo "bcm2835-v4l2" | sudo tee -a /etc/modules
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Click the **Android** platform and then the **Switch Platform** button."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Contacting the author: You can email Joseph Howse directly at `josephhowse@nummist.com`. He maintains this book's GitHub repository at `https://github.com/PacktPublishing/OpenCV-4-for-Secret-Agents-Second-Edition`, as well as his own support webpage for his books at `http://nummist.com/opencv`, so you may want to look for updates from him on these sites.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

1
Section 1: The Briefing

Set up a multi-platform development environment. Integrate OpenCV with other libraries to make an application that classifies images from the web.

The following chapters will be covered in this section:

- `Chapter 1`, *Preparing for the Mission*
- `Chapter 2`, *Searching for Luxury Accommodations Worldwide*

1
Preparing for the Mission

"Q: I've been saying for years, sir, that our special equipment is obsolete. And now, computer analysis reveals an entirely new approach: miniaturization."

– On Her Majesty's Secret Service (1969)

James Bond is not a pedestrian. He cruises in a submarine car; he straps on a rocket belt; and, oh, how he skis, how he skis! He always has the latest stuff, and he is never afraid to put a dent in it, much to the dismay of Q, the engineer.

As software developers in the 2010s, we have witnessed an explosion of the adoption of new platforms. Under one family's roof, we might find a mix of Windows, Mac, iOS, and Android devices. Mom and Dad's workplaces provide different platforms. The kids have three game consoles, or five, if you count the mobile versions. The toddler has a LeapFrog learning tablet. Smart glasses are becoming more affordable.

We must not be afraid to try new platforms and consider new ways to combine them. After all, our users do.

This book embraces multiplatform development. It presents weird and wonderful applications that we can deploy in unexpected places. It uses several of the computer's senses, but especially computer vision, to breathe new life into the humdrum, heterogeneous clutter of devices that surrounds us.

Before Agent 007 runs amok with the gadgets, he is obligated to listen to Q's briefing. This chapter will performs Q's role. It is the setup chapter.

By the end of this chapter, you will have obtained all the tools to develop OpenCV applications in Python for Windows, Mac, or Linux, and in Java for Android. You will also be the proud new user of a Raspberry Pi single-board computer. (This additional hardware is optional.) You will even know a bit about Unity, a game engine into which we can integrate OpenCV. Specifically, this chapter will cover the following approaches to setting up a development environment:

- Setting up Python and OpenCV on Windows. Optionally, this will include configuring and building OpenCV from a source with CMake and Visual Studio.
- Setting up Python and OpenCV on Mac. This will include using either MacPorts or Homebrew as a package manager.
- Setting up Python and OpenCV on Debian Jessie or one of its derivatives, such as Raspbian, Ubuntu, or Linux Mint. This will include using the **Advanced Package Tool** (**APT**) package manager. Optionally, it will also include configuring and building OpenCV from a source with CMake and GCC.
- Setting up Python and OpenCV on Fedora or one of its derivatives, such as **Red Hat Enterprise Linux** (**RHEL**) or CentOS. This will include using the yum package manager.
- Setting up Python and OpenCV on openSUSE. This will include using the yum package manager.
- Setting up Android Studio and OpenCV's Android libraries on Windows, Mac, or Linux.
- Setting up Unity and OpenCV on Windows or Mac.
- Setting up a Raspberry Pi.

If you find yourself a bit daunted by the extent of this setup chapter, be reassured that not all of the tools are required, and no single project uses all of them in combination. Although Q and I live for the big event of setting up multiple technologies at once, you can just skim this chapter and refer back to it later when the tools become useful, one by one, in our projects.

Technical requirements

This is the setup chapter. There are no particular software prerequisites at the outset; we will set up everything as we go along.

Basic instructions for running Python code are covered in `Appendix C`, *Running with Snakes (or, First Steps with Python)*. After we set up a Python environment with OpenCV, you may want to refer to this appendix so that you know how to minimally test the environment.

Setting up a development machine

We can develop our OpenCV applications on a desktop, a notebook, or even the humble Raspberry Pi (covered later, in the *Setting up a Raspberry Pi* section). Most of our apps have a memory footprint of less than 128 MB, so they can still run (albeit slowly) on old or low-powered machines. To save time, develop on your fastest machine first and test on slower machines later.

This book assumes that you have one of the following operating systems on your development machine:

- Windows 7 SP 1, or a later version.
- Mac OS 10.7 (Lion), or a later version.
- Debian Jessie, a later version, or a derivative such as the following:
    - Raspbian 2015-09-25, or a later version
    - Ubuntu 14.04, or a later version
    - Linux Mint 17, or a later version
- Fedora 28, a later version, or a derivative such as the following:
    - RHEL 8, or a later version
    - CentOS 8, or a later version
- openSUSE Leap 42.3, openSUSE Leap 15.0, or a later version; openSUSE Tumbleweed, or a derivative.

Other Unix-like systems might also work, but they will not be covered in this book.

You should have a USB webcam and any necessary drivers. Most webcams come with instructions for installing drivers on Windows and Mac. Linux distributions typically include the **USB Video Class** (**UVC**) Linux driver, which supports many webcams, listed at `http://www.ideasonboard.org/uvc/#devices`.

We are going to set up the following components:

- On Mac, a third-party package manager to help us install libraries and their dependencies; we will use either MacPorts or Homebrew.
- A Python development environment—at the time of writing this book, OpenCV supports Python 2.7, 3.4, 3.5, 3.6, and 3.7. The Python code in this book supports all of these versions. As part of the Python development environment, we will use Python's package manager, pip.
- Popular Python libraries, such as NumPy (for numeric functions), SciPy (for numeric and scientific functions), Requests (for web requests), and wxPython (for cross-platform GUIs).

- PyInstaller, a cross-platform tool for bundling Python scripts, libraries, and data as redistributable apps, such that user machines do not require installations of Python, OpenCV, and other libraries. For this book's purposes, building redistributables of Python projects is an optional topic. We will cover the basics in `Chapter 2`, *Searching for Luxury Accommodations Worldwide*, but you might need to do your own testing and debugging, as PyInstaller (like other Python bundling tools) does not show entirely consistent behavior across operating systems, Python versions, and library versions. It is not well supported on Raspberry Pi or other ARM devices.
- Optionally, we can use a C++ development environment to enable us to build OpenCV from a source. On Windows, we use Visual Studio 2015 or later. On Mac, we use Xcode. On Linux, we use GCC, which comes as standard.
- A build of OpenCV and `opencv_contrib` (a set of extra OpenCV modules) with Python support, plus optimizations for certain desktop hardware. At the time of writing this book, OpenCV 4.0.x is the latest stable branch, and our instructions are tailored for this branch. However, generally, the code in this book also works with the previous stable branch, OpenCV 3.4.x, which is more widely available from package managers for users who prefer a prepackaged build.
- Another build of OpenCV with Java support, plus optimizations for certain Android hardware. At the time of writing, OpenCV 4.0.1 is the most recent release.
- An Android development environment, including Android Studio and Android SDK.
- On 64-bit Windows or Mac, a three-dimensional game engine called **Unity**.

Android Studio has a big memory footprint. Even if you want to use Raspberry Pi for developing desktop and Pi apps, use something with more RAM for developing Android apps.

Let's break this setup down into three sets of platform dependent steps for a Python and OpenCV environment, plus a set of platform independent steps for an Android Studio and OpenCV environment, and another set of platform independent steps for a Unity and OpenCV environment.

Setting up Python and OpenCV on Windows

On Windows, we have the option of setting up a 32-bit development environment (to make apps that are compatible with both 32-bit and 64-bit Windows) or a 64-bit development environment (to make optimized apps that are only compatible with 64-bit Windows). OpenCV is available in 32-bit and 64-bit versions.

We also have a choice of either using binary installers or compiling OpenCV from source. For our Windows apps in this book, the binary installers provide everything we need. However, we also discuss the option of compiling from source because it enables us to configure additional features, which may be relevant to your future work or to our projects in other books.

Regardless of our approach to obtaining OpenCV, we need a general-purpose Python development environment. We will set up this environment using a binary installer. The installers for Python are available from `http://www.python.org/getit/`. Download and run the latest revision of Python 3.7, in either the 32-bit variant or the 64-bit variant.

To make Python scripts run using our new Python 3.7 installation by default, let's edit the system's Path variable and append `;C:\Python3.7` (assuming Python 3.7 is installed in the default location). Remove any previous Python paths, such as `;C:\Python2.7`. Log out and log back in (or reboot).

Python comes with a package manager called `pip`, which simplifies the task of installing Python modules and their dependencies. Open Command Prompt and run the following command to install `numpy`, `scipy`, `requests`, `wxPython`, and `pyinstaller`:

```
> pip install --user numpy scipy requests wxPython pyinstaller
```

Now, we have a choice. We can either install the binaries of `OpenCV` and `opencv_contrib` as a prebuilt Python module, or we can build this module from source. To install a prebuilt module, simply run the following command:

```
> pip install --user opencv-contrib-python
```

Alternatively, to build `OpenCV` and `opencv_contrib` from source, follow the instructions in the section *Building OpenCV on Windows with CMake and Visual Studio*, as follow.

After either installing a prebuilt `OpenCV` and `opencv_contrib` module or building it from source, we will have everything we need to develop OpenCV applications for Windows. To develop for Android, we need to set up Android Studio as described in the section *Setting up Android Studio, and OpenCV*, later in this chapter.

Building OpenCV on Windows with CMake and Visual Studio

To compile OpenCV from source, we need a general purpose C++ development environment. As our C++ development environment, we will use Visual Studio 2015 or later. Use any installation media you may have purchased, or go to the downloads page at `https://visualstudio.microsoft.com/downloads/`. Download and run the installer for one of the following:

- Visual Studio Community 2017, which is free
- Any of the paid Visual Studio 2017 versions, which have 30-day free trials

If the installer lists optional C++ components, we should opt to install them all. After the installer runs to completion, reboot.

OpenCV uses a set of build tools called **CMake**, which we must install. Optionally, we may install several third-party libraries in order to enable extra features in OpenCV. As an example, let's install Intel **Thread Building Blocks** (**TBB**), which OpenCV can leverage in order to optimize some functions for multicore CPUs. After installing TBB, we will configure and build OpenCV. Lastly, we will ensure that our C++ and Python environments can find our build of OpenCV.

Here are the detailed steps:

1. Download and install the latest stable version of CMake from `https://cmake.org/download/`. CMake 3 or a newer version is required. Even if we are using 64-bit libraries and compilers, 32-bit CMake is compatible. When the installer asks about modifying PATH, select either **Add CMake to the system PATH for all users** or **Add CMake to the system PATH for current user**.
2. If your system uses a proxy server to access the internet, define two environment variables, `HTTP_PROXY` and `HTTPS_PROXY`, with values equal to the proxy server's URL, such as `http://myproxy.com:8080`. This ensures that CMake can use the proxy server to download some additional dependencies for OpenCV. (If in doubt, do not define these environment variables; you are probably not using a proxy server.)

3. Download the OpenCV Win pack from `http://opencv.org/releases.html`. (Choose the latest version.) The downloaded file may have an `.exe` extension, but actually, it is a self-extracting ZIP. Double-click on the file and, when prompted, enter any destination folder, which we will refer to as `<opencv_unzip_destination>`. A subfolder, `<opencv_unzip_destination>/opencv`, will be created.
4. Download `opencv_contrib` as a ZIP from `https://github.com/opencv/opencv_contrib/releases`. (Choose the latest version.) Unzip it to any destination folder, which we will refer to as `<opencv_contrib_unzip_destination>`.
5. Download the latest stable version of TBB from `https://www.threadingbuildingblocks.org/download`. It includes both 32-bit and 64-bit binaries. Unzip it to any destination, which we will refer to as `<tbb_unzip_destination>`.
6. Open Command Prompt. Create a folder to store our build:

```
> mkdir <build_folder>
```

   Change directories to the newly created build folder:

```
> cd <build_folder>
```

7. Having set up our dependencies, we can now configure OpenCV's build system. To understand all the configuration options, we can read the code in `<opencv_unzip_destination>/opencv/sources/CMakeLists.txt`. However, as an example, we will just use the options for a release build that includes Python bindings and multiprocessing through TBB:

    - To create a 32-bit project for Visual Studio 2017, run the following command (but replace the angle brackets and their contents with the actual paths):

```
> CALL <tbb_unzip_destination>\bin\tbbvars.bat ia32 vs2017
> cmake -DCMAKE_BUILD_TYPE=RELEASE DWITH_OPENGL=ON -
DWITH_TBB=ON
-DOPENCV_SKIP_PYTHON_LOADER=ON
-DPYTHON3_LIBRARY=C:/Python37/libs/python37.lib
-DPYTHON3_INCLUDE_DIR=C:/Python37/include -
DOPENCV_EXTRA_MODULES_PATH="<opencv_contrib_unzip_destination>/
modules" -G "Visual Studio 15 2017"
"<opencv_unzip_destination>/opencv/sources"
```

- Alternatively, to create a 64-bit project for Visual Studio 2017, run the following command (but replace the angle brackets and their contents with the actual paths):

```
> CALL <tbb_unzip_destination>\bin\tbbvars.bat intel64 vs2017
> cmake -DCMAKE_BUILD_TYPE=RELEASE DWITH_OPENGL=ON -
DWITH_TBB=ON
-DOPENCV_SKIP_PYTHON_LOADER=ON
-DPYTHON3_LIBRARY=C:/Python37/libs/python37.lib
-DPYTHON3_INCLUDE_DIR=C:/Python37/include -
DOPENCV_EXTRA_MODULES_PATH="<opencv_contrib_unzip_destination>/
modules" -G "Visual Studio 15 2017 Win64"
"<opencv_unzip_destination>/opencv/sources"
```

- CMake will produce a report on the dependencies that it did or did not find. OpenCV has many optional dependencies, so do not panic (yet) about missing dependencies. However, if the build does not finish successfully, try installing missing dependencies. (Many are available as prebuilt binaries.) Then, repeat this step.

8. Now that our build system is configured, we can compile OpenCV. Open `<build_folder>/OpenCV.sln` in Visual Studio. Select the **Release** configuration and build the solution. (You may get errors if you select another build configuration besides **Release**, because most Python installations do not include debug libraries.)
9. We should ensure that our Python installation does not already include some other version of OpenCV. Find and delete any OpenCV files in your Python `DLLs` folder and your Python `site-packages` folder. For example, the paths to these files might match the `C:\Python37\DLLs\opencv_*.dll`, `C:\Python37\Lib\site-packages\opencv`, and `C:\Python37\Lib\site-packages\cv2.pyd` patterns.

10. Finally, we need to install OpenCV to a location where Python and other processes can find it. To do this, right-click on the OpenCV solution's **INSTALL** project (in the **Solution Explorer** pane of Visual Studio) and build it. When this build finishes, quit Visual Studio. Edit the system's `Path` variable and append `;<build_folder>\install\x86\vc15\bin` (for a 32-bit build) or `;<build_folder>\install\x64\vc15\bin` (for a 64-bit build), which is the location where the OpenCV DLL files are located. Also, append `;<tbb_unzip_destination>\lib\ia32\vc14` (32-bit) or `;<tbb_unzip_ destination>\lib\intel64\vc14`(64-bit), which is the location where the TBB DLL files are located. Log out and log back in (or reboot). The OpenCV Python module is located at a path such as `C:\Python37\Lib\site-packages\cv2.pyd`. Python will find it there, so you do not need to take any further steps.

If you are using Visual Studio 2015, replace `vs2017` with `vs2015`, replace `Visual Studio 15 2017` with `Visual Studio 14 2015`, and replace `vc15` with `vc14`. However, for TBB, note that the folder named `vc14` contains the DLL files that work for both Visual Studio 2015 and Visual Studio 2017.

You might want to look at the code samples in `<opencv_unzip_destination>/opencv/sources/samples/python`.

At this point, we have everything we need to develop OpenCV applications for Windows. To also develop for Android, we need to set up Android Studio, as described in the *Setting up Android Studio and OpenCV* section, later in this chapter.

Setting up Python and OpenCV on Mac

Mac comes with Python preinstalled. However, the preinstalled Python has been customized by Apple for the system's internal needs. Normally, we should not install any libraries on top of Apple's Python. If we do, our libraries might break during system updates, or worse, they might conflict with preinstalled libraries that the system requires. Instead, we should install standard Python 3.7 and then install our libraries on top of it.

For Mac, there are several possible approaches to obtaining standard Python 3.7 and Python-compatible libraries, such as OpenCV. All approaches ultimately require some components to be compiled from source, using Xcode developer tools. However, depending on the approach we choose, the task of building these components is automated for us by third-party tools in various ways.

Let's begin by setting up Xcode and the Xcode command-line tools, which give us a complete C++ development environment:

1. Download and install Xcode from the Mac App Store or `https://developer.apple.com/xcode/`. If the installer provides an option to install command-line tools, select it.
2. Open Xcode. If a license agreement is presented, accept it.
3. If command-line tools were not already installed, we must install them now. Go to **Xcode** | **Preferences** | **Downloads** and click on the **Install** button next to command-line tools. Wait for the installation to finish. Then, quit Xcode. Alternatively, if you do not find an option to install command-line tools from inside Xcode, open the Terminal and run the following command:

```
$ xcode-select install
```

Now, we will look at ways to automate our builds using MacPorts or Homebrew. These two tools are package managers, which help us resolve dependencies and separate our development libraries from the system libraries.

Generally, I recommend MacPorts. Compared to Homebrew, MacPorts offers more patches and configuration options for OpenCV. On the other hand, Homebrew offers more timely updates for OpenCV. At the time of writing, Homebrew offers a package for OpenCV 4.0.1, but MacPorts is still lagging at OpenCV 3.4.3. Homebrew and MacPorts can coexist with the Python package manager, pip, and we can use pip to get OpenCV 4.0.1, even though MacPorts does not package this version yet. Normally, MacPorts and Homebrew should not be installed on the same machine.

Our installation methods for Mac do not give us the OpenCV sample projects. To get these, download the latest source code archive from `https://opencv.org/releases.html` and unzip it to any location. Find the samples in `<opencv_unzip_destination>/samples/python`.

Now, depending on your preference, let's proceed to either the *Mac with MacPorts* section or the *Mac with Homebrew* section.

Mac with MacPorts

MacPorts provides Terminal commands that automate the process of downloading, compiling, and installing various pieces of **open source software** (**OSS**). MacPorts also installs dependencies, as needed. For each piece of software, the dependencies and build recipe are defined in a configuration file called a **Portfile**. A MacPorts repository is a collection of Portfiles.

Starting from a system where Xcode and its command-line tools are already set up, the following steps will give us an OpenCV installation through MacPorts:

1. Download and install MacPorts from `http://www.macports.org/install.php`.
2. Open the Terminal and run the following command to update MacPorts:

```
$ sudo port selfupdate
```

   When prompted, enter your password.

3. Run the following commands to install Python 3.7, pip, NumPy, SciPy, and Requests:

```
$ sudo port install python37
$ sudo port install py37-pip
$ sudo port install py37-numpy
$ sudo port install py37-scipy
$ sudo port install py37-requests
```

4. The Python installation executable is named python3.7. To link the default python executable to python3.7, and to link the default pip executable to this Python pip installation, let's also run the following:

```
$ sudo port install python_select
$ sudo port select python python37
$ sudo port install pip_select
$ sudo port select pip pip37
```

5. At the time of writing, MacPorts only has packages for relatively old versions of wxPython and PyInstaller. Let's use the following `pip` command to install more recent versions:

```
$ pip install --user wxPython pyinstaller
```

6. To check whether MacPorts has an OpenCV 4 package, run `$ port list opencv`. At the time of writing, this produces the following output:

```
opencv                              @3.4.3          graphics/opencv
```

   - Here, `@3.4.3` means that OpenCV 3.4.3 is the latest available package from MacPorts. However, if your output shows `@4.0.0` or a more recent version, you can use MacPorts to configure, build, and install OpenCV 4, by running a command such as the following:

```
$ sudo port install opencv +avx2 +contrib +opencl +python37
```

- By adding `+avx2 +contrib +opencl +python37` to the command, we are specifying that we want the `opencv` variant (build configuration) with AVX2 CPU optimizations, `opencv_contrib` extra modules, OpenCL GPU optimizations, and Python 3.7 bindings. To see the full list of available variants before installing, we can enter the following:

```
$ port variants opencv
```

- Depending on our customization needs, we can add other variants to the `install` command.
- On the other hand, if the output from `$ port list opencv` showed that MacPorts does not have an OpenCV 4 package yet, we can instead install OpenCV 4 and the `opencv_contrib` extra modules with pip, by running the following command:

```
$ pip install --user opencv-contrib-python
```

Now, we have everything we need to develop OpenCV applications for Mac. To also develop for Android, we need to set up Android Studio, as we will describe in the following *Android Studio* section.

Mac with Homebrew

Like MacPorts, Homebrew is a package manager that provides Terminal commands to automate the process of downloading, compiling, and installing various pieces of open source software.

Starting from a system where Xcode and its command-line tools are already set up, the following steps will give us an OpenCV installation through Homebrew:

1. Open Terminal and run the following command to install Homebrew:

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Unlike MacPorts, Homebrew does not automatically put its executables in `PATH`. To do so, create or edit the `~/.profile` file and add this line at the top:

```
export PATH=/usr/local/bin:/usr/local/sbin:$PATH
```

- Save the file and run this command to refresh `PATH`:

```
$ source ~/.profile
```

- Note that executables installed by Homebrew now take precedence over executables installed by the system.

3. For Homebrew's self-diagnostic report, run:

```
$ brew doctor
```

   Follow any troubleshooting advice it gives.

4. Now, update Homebrew:

```
$ brew update
```

5. Run the following command to install Python 3.7:

```
$ brew install python
```

6. Now, we can use Homebrew to install OpenCV and its dependencies, including NumPy. Run the following command:

```
$ brew install opencv --with-contrib
```

7. Similarly, run the following command to install SciPy:

```
$ pip install --user scipy
```

8. At the time of writing, Homebrew does not have packages for `requests` and `pyinstaller`, and its `wxPython` package is a relatively old version, so instead, we will use `pip` to install these modules. Run the following command:

```
$ pip install --user requests wxPython pyinstaller
```

Now, we have everything we need to develop OpenCV applications for Mac. To also develop for Android, we need to set up **Tegra Android Development Pack** (**TADP**), as described in the following *Tegra Android Development Pack* section.

Setting up Python and OpenCV on Debian Jessie and its derivatives, including Raspbian, Ubuntu, and Linux Mint

For information on setting up the Raspbian operating system, see the *Setting up a Raspberry Pi* section, later in this chapter.

On Debian Jessie, Raspbian, Ubuntu, Linux Mint, and their derivatives, the python executable is Python 2.7, which comes preinstalled. We can use the system package manager, apt, to install NumPy, SciPy, and Requests from the standard repository. To update the apt package index and install the packages, run the following commands in Terminal:

```
$ sudo apt-get update
$ sudo apt-get install python-numpy python-scipy python-requests
```

The standard repository's latest packaged version of wxPython varies, depending on the operating system. On Ubuntu 14.04 and its derivatives, including Linux Mint 17, the latest packaged version is wxPython 2.8. Install it by running the following command:

```
$ sudo apt-get install python-wxgtk2.8
```

On Ubuntu 18.04 and newer versions, as well as derivatives such as Linux Mint 19, the latest packaged version is wxPython 4.0. Install it by running the following command:

```
$ sudo apt-get install python-wxgtk4.0
```

On most other systems in the Debian Jessie family, wxPython 3.0 is the latest packaged version. Install it by running the following command:

```
$ sudo apt-get install python-wxgtk3.0
```

The standard repository does not offer a PyInstaller package. Instead, let's use Python's own package manager, `pip`, to obtain PyInstaller. First, to ensure that `pip` is installed, run the following command:

```
$ sudo apt-get install python-pip
```

Now, install PyInstaller by running the following command:

```
$ pip install --user pyinstaller
```

The standard repository contains a `python-opencv` package, but it is an old version (3.2.0 or older, depending on the operating system) and it is missing the `opencv_contrib` modules, so it lacks some of the functionality used in this book. Thus, we have a choice of either using `pip` to obtain OpenCV 4 with the `opencv_contrib` modules, or building the same from source. To install a prebuilt version of OpenCV 4 and `opencv_contrib` with `pip`, run the following command:

```
$ pip install --user opencv-contrib-python
```

If you prefer to use the `python3` executable, which is Python 3.4 or a newer version (depending on the operating system), modify all the `apt-get` commands in the preceding instructions to use package names like `python3-numpy`, instead of names like `python-numpy`. Similarly, replace the `pip` commands with the `pip3` commands.

Alternatively, to build `OpenCV` and `opencv_contrib` from source, follow the instructions in the following *Building OpenCV on Debian Jessie and its derivates with CMake and GCC* section.

After either installing a prebuilt `OpenCV` and `opencv_contrib` module or building it from source, we will have everything we need to develop OpenCV applications for Debian Jessie or a derivative. To also develop for Android, we need to set up Android Studio, as described in the *Setting up Android Studio and OpenCV* section, later in this chapter.

Building OpenCV on Debian Jessie and its derivatives with CMake and GCC

To compile OpenCV from source, we need a general-purpose C++ development environment. On Linux, the standard C++ development environment includes the g++ compiler and the Make build system, which defines build instructions in a file format known as **Makefile**.

OpenCV uses a set of build tools called **CMake**, which automates the use of Make, g++, and other tools. CMake 3 or a newer version is required, and we must install it. Also, we will install several third-party libraries. Some of these are required for standard OpenCV features, while others are optional dependencies that enable extra features.

As an example, let's install the following optional dependencies:

- `libdc1394`: This is a library to programmatically control IEEE 1394 (FireWire) cameras, which are quite common in industrial use. OpenCV can leverage this library to capture photos or video from some of these cameras.

- `libgphoto2`: This is a library to programmatically control photo cameras through a wired or wireless connection. The `libgphoto2` library supports a large number of cameras from Canon, Fuji, Leica, Nikon, Olympus, Panasonic, Sony, and other manufacturers. OpenCV can leverage this library to capture photos or video from some of these cameras.

After installing dependencies, we will configure, build, and install `OpenCV`. Here are the detailed steps:

1. On Ubuntu 14.04 and its derivatives, including Linux Mint 17, the `cmake` package in the standard repository is CMake 2, which is too old for our purposes. We need to ensure that the `cmake` package is not installed, and then we need to install the `cmake3` package, as well as other essential development and packaging tools. To accomplish this, run the following commands:

   ```
   $ sudo apt-get remove cmake
   $ sudo apt-get install build-essential cmake3 pkg-config
   ```

   On more recent versions of Ubuntu and its derivatives, and on Debian Jessie, the `cmake3` package does not exist; rather, the `cmake` package is CMake 3. Install it, as well as other essential development and packaging tools, by running the following command:

   ```
   $ sudo apt-get install build-essential cmake pkg-config
   ```

2. If your system uses a proxy server to access the internet, define two environment variables, `HTTP_PROXY` and `HTTPS_PROXY`, with values equal to the proxy server URL, such as `http://myproxy.com:8080`. This ensures that CMake can use the proxy server to download some additional dependencies for OpenCV. (If in doubt, do *not* define these environment variables; you are probably *not* using a proxy server.)

3. Run the following command to install OpenCV's dependencies for Python bindings and for video capture from **Video4Linux** (**V4L**)-compatible cameras, including most webcams:

```
$ sudo apt-get install python-dev libv4l-dev
```

If you prefer to use Python 3, replace `python-dev` with `python3-dev` in the preceding command.

4. Run the following command to install optional OpenCV dependencies:

```
$ sudo apt-get install libdc1394-22-dev libgphoto2-dev
```

5. Download the OpenCV source's ZIP from `http://opencv.org/releases.html`. (Choose the latest version.) Unzip it to any destination folder, which we will refer to as `<opencv_unzip_destination>`.
6. Download `opencv_contrib` as a ZIP from `https://github.com/opencv/opencv_contrib/releases`. (Choose the latest version.) Unzip it to any destination folder, which we will refer to as `<opencv_contrib_unzip_destination>`.
7. Open Command Prompt. Create a folder to store our build:

```
$ mkdir <build_folder>
```

Change the directory to the newly created build folder:

```
$ cd <build_folder>
```

8. Having set up our dependencies, we can now configure OpenCV's build system. To understand all the configuration options, we can read the code in `<opencv_unzip_destination>/opencv/sources/CMakeLists.txt`. However, as an example, we will just use the options for a release build that includes Python bindings, support for OpenGL interoperability, and support for extra camera types and video types. To create Makefiles for OpenCV with Python 2.7 bindings, run the following command (but replace the angle brackets and their contents with the actual paths):

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D BUILD_EXAMPLES=ON -D
WITH_1394=ON
-D WITH_GPHOTO2=ON -D BUILD_opencv_python2=ON
-D PYTHON2_EXECUTABLE=/usr/bin/python2.7
-D PYTHON_LIBRARY2=/usr/lib/python2.7/config-x86_64-linux-
```

```
gnu/libpython2.7.so -D PYTHON_INCLUDE_DIR2=/usr/include/python2.7 -
D BUILD_opencv_python3=OFF
-D
OPENCV_EXTRA_MODULES_PATH=<opencv_contrib_unzip_destination>/module
s <opencv_unzip_destination>
```

Alternatively, to create Makefiles for OpenCV with bindings for Python 3, run the following command (but replace the angle brackets and their contents with the actual paths, and replace `3.6` with your actual Python 3 version, if it is not 3.6):

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D BUILD_EXAMPLES=ON -D
WITH_1394=ON -D WITH_GPHOTO2=ON -D BUILD_opencv_python2=OFF
-D BUILD_opencv_python3=ON -D PYTHON3_EXECUTABLE=/usr/bin/python3.6
-D PYTHON3_INCLUDE_DIR=/usr/include/python3.6
-D PYTHON3_LIBRARY=/usr/lib/python3.6/config-3.6m-x86_64-linux-
gnu/libpython3.6.so -D
OPENCV_EXTRA_MODULES_PATH=<opencv_contrib_unzip_destination>
<opencv_unzip_destination>
```

9. Run the following commands to build and install OpenCV in the manner specified by the Makefiles:

```
$ make -j8
$ sudo make install
```

At this point, we have everything we need to develop OpenCV applications for Debian Jessie or a derivative. To also develop for Android, we need to set up Android Studio, as described in the *Setting up Android Studio and OpenCV* section, later in this chapter.

Setting up Python and OpenCV on Fedora and its derivatives, including RHEL and CentOS

On Fedora, RHEL, and CentOS, the `python` executable is Python 2.7, which comes preinstalled. We can use the system package manager, yum, to install NumPy, SciPy, Requests, and wxPython from the standard repository. To do this, open Terminal and run the following command:

```
$ sudo yum install numpy scipy python-requests wxPython
```

The standard repository does not offer a PyInstaller package. Instead, let's use Python's own package manager, `pip`, to obtain PyInstaller. First, to ensure that `pip` is installed, run the following command:

```
$ sudo yum install python-pip
```

Now, install PyInstaller by running the following command:

```
$ pip install --user pyinstaller
```

The standard repository contains an `opencv` package, which includes the `opencv_contrib` modules and Python bindings, but it is an old version (3.4.4 or older, depending on the operating system). Thus, we want to use `pip` to obtain OpenCV 4 with the `opencv_contrib` modules. Run the following command:

```
$ pip install --user opencv-contrib-python
```

If you prefer to use the `python3` executable, which is Python 3.6 or 3.7 (depending on the operating system), replace the preceding `pip` commands with the `pip3` commands. You do not need to modify the `yum` commands, as the relevant packages, such as `numpy`, include sub-packages for both Python 2 and Python 3.

Now, we have everything we need to develop OpenCV applications for Fedora or a derivative. To also develop for Android, we need to set up Android Studio, as described in the following *Setting up Android Studio and OpenCV* section.

Setting up Python and OpenCV on openSUSE and its derivatives

On openSUSE, the `python` executable is Python 2.7, which comes preinstalled. We can use the system package manager, yum, to install NumPy, SciPy, Requests, and wxPython from the standard repository. To do this, open Terminal and run the following command:

```
$ sudo yum install python-numpy python-scipy python-requests python-
wxWidgets
```

Although openSUSE and Fedora both use the yum package manager, they use different standard repositories with different package names.

The standard repository does not offer a PyInstaller package. Instead, let's use Python's own package manager, `pip`, to obtain PyInstaller. First, to ensure that pip is installed, run the following command:

```
$ sudo yum install python-pip
```

Now, install PyInstaller by running the following command:

```
$ pip install --user pyinstaller
```

The standard repository contains a `python2-opencv` package (and a `python3-opencv` package for Python 3), but it is an old version of OpenCV (3.4.3 or older, depending on the operating system). Thus, we want to use pip to obtain OpenCV 4 with the `opencv_contrib` modules. Run the following command:

```
$ pip install --user opencv-contrib-python
```

If you prefer to use the `python3` executable, which is Python 3.4, 3.6, or 3.7 (depending on the operating system), replace the preceding `pip` commands with the `pip3` commands. You do not need to modify the `yum` commands, as the relevant packages, such as `python-numpy`, include sub-packages for both Python 2 and Python 3.

Now, we have everything we need to develop OpenCV applications for openSUSE or a derivative. Next, we need to follow the cross-platform steps for setting up an Android development environment.

Setting up Android Studio and OpenCV

Android Studio is Google's official **integrated development environment** (**IDE**) for Android app development. Since its first stable release in 2014, Android Studio has grown in popularity and has replaced Eclipse as the IDE of choice for Android developers. Although some of the OpenCV documentation still contains outdated tutorials on Android development in Eclipse, nowadays, the OpenCV Android library and Android sample projects are primarily intended for use with Android Studio, instead.

Google provides a good cross-platform tutorial on Android Studio installation at `https://developer.android.com/studio/install`. Follow the part of the tutorial that is relevant to your operating system.

Download the latest version of the OpenCV Android pack from `https://opencv.org/releases.html`. Unzip it to any destination, which we will refer to as `<opencv_android_pack_unzip_destination>`. This has two subfolders:

- `<opencv_android_pack_unzip_destination>/sdk` contains the `OpenCV4Android` SDK. This consists of Java and C++ libraries, as well as build instructions, which we can import into Android Studio projects.

- `<opencv_android_pack_unzip_destination>/samples` contains sample projects that can be built in Android Studio. Unfortunately, as of OpenCV 4.0.1, these samples are outdated. On Android 6.0 and newer versions, the samples fail to access the camera, because they do not request user permission at runtime in the required manner.

At this point, we have obtained the core components of our development environment for the OpenCV Android apps. Next, let's look at Unity, a game engine that can deploy to Android and other platforms.

Setting up Unity and OpenCV

Unity (`https://unity3d.com`) is a three-dimensional game engine that supports development on 64-bit Windows or Mac, and deployment to many platforms, including Windows, Mac, Linux, iOS, Android, WebGL, and several game consoles. For one of our projects, we will use Unity along with a plugin called **OpenCV for Unity**, which is developed by Enox Software (`http://enoxsoftware.com/`.) The main programming language for Unity projects is C#, and OpenCV for Unity provides a C# API that is modeled on the OpenCV Java API for Android.

Unity has three license plans, Personal, Plus, and Pro, which all support the plugin that we want to use. The different editions are for different sizes of companies, as described in the licensing information at `https://store.unity.com`. The Personal license is free. The Plus and Pro licenses have subscription costs. If you are not already a Unity subscriber, you may want to wait and purchase a subscription when you are ready to start working on our Unity project in `Chapter 6`, *Creating a Physics Simulation Based on a Pen and Paper Sketch*. Once you are ready, obtain your license from `https://store.unity.com` and download Unity Hub from `https://unity3d.com/get-unity/download`. Unity Hub is an application to manage your Unity licenses and installations. Use it to set up Unity on your system. You can purchase OpenCV for Unity from the Unity Asset Store at `https://assetstore.unity.com/packages/tools/integration/opencv-for-unity-21088`, but we will cover details about obtaining the plugin in `Chapter 6`, *Creating a Physics Simulation Based on a Pen and Paper Sketch,* when we set up a Unity project.

Even before installing Unity, we can get inspiration from the demos at `https://unity3d.com/unity/demos/`. These demos include videos, articles about the development process, and in some cases, playable games that you can download for various platforms. They also include source code and art assets that you can open in Unity. After installing Unity, we can learn from these and other demo projects. Take a look at the resources available for download at `https://unity3d.com/learn/resources/downloads`. Also, check out the tutorials, videos, and documentation at `https://unity3d.com/learn`.

As you can see, there are a lot of official resources for Unity beginners, so I will let you explore these on your own for now.

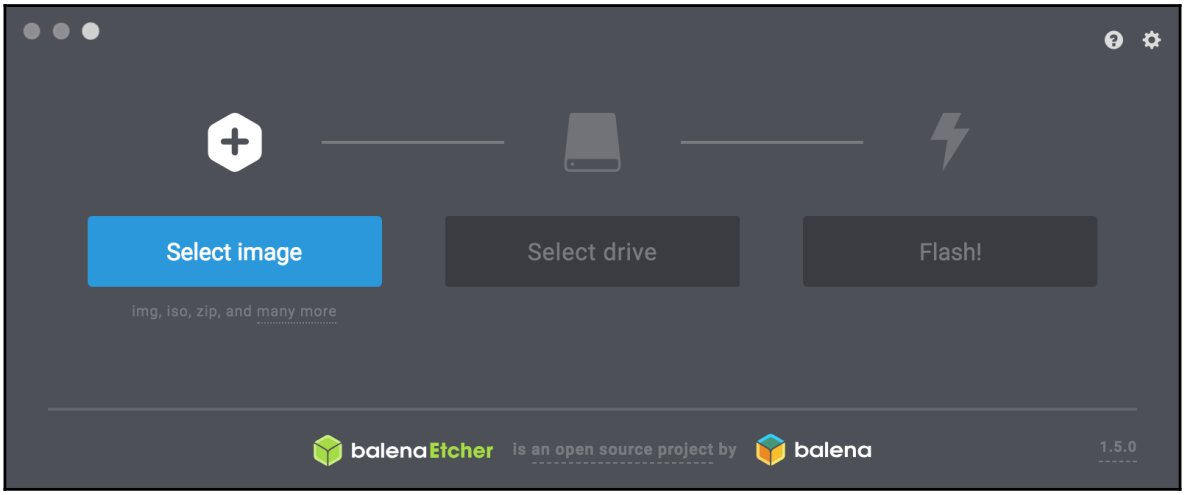
Setting up a Raspberry Pi

Raspberry Pi is a **single-board computer** (**SBC**) with a low cost and low power consumption. It can be used as a desktop, a server, or an embedded system that controls other electronics. The Pi comes in several models. Currently, the flagship is Model 3 B+, which costs about $35. Compared to other models, it offers a faster CPU, more memory, and a faster Ethernet port, so it is a good candidate for computer vision experiments that rely on either local computing resources or the cloud. However, other models are also usable for computer vision projects.

Several operating systems are available for Raspberry Pi. We will use Raspbian, which is a port of Debian Stretch (a major Linux distribution) to ARM.

Download the latest Raspbian disk image from `http://downloads.raspberrypi.org/raspbian_latest`. You do not need to unzip the downloaded file. At the time of writing, the ZIP is called `2018-11-13-raspbian-stretch.zip` . Since your filenames may differ, we will refer to the file as `<raspbian_zip>`.

The `<raspbian_zip>` file contains a disk image, which we need to burn to an SD card with a capacity of at least 4 GB. (Note that 8 GB or larger is preferable, to allow plenty of room for OpenCV and our projects.) Any existing data on the card will be lost in the process. To burn the card, let's use a cross-platform, open source application called **Etcher**. Download it from `https://www.balena.io/etcher/` and set it up. (An installer is available for Windows or Mac. Alternatively, a portable application is available for Windows, Mac, or Linux). Insert an SD card and open Etcher. You should see the following window:



Etcher's user interface is so self-explanatory that even James Bond might struggle to find a double meaning in it. Click on the **Select image** button and select `<raspbian_zip>`. Click on the **Select drive** button and select your SD drive. Click on the **Flash!** button to burn the image to the SD card. Wait for the burning process to finish. Quit Etcher and eject the card.

Now, let's turn our attention to the Raspberry Pi hardware. Ensure that the Raspberry Pi micro-USB power cable is disconnected. Connect an HDMI monitor or TV, USB keyboard, USB mouse, and (optionally) Ethernet cable. Then, insert the SD card firmly into the slot on the bottom of the Pi. Connect the Pi power cable. The Pi should start booting from the SD card. During this first bootup, the filesystem expands to fill the entire SD card. When the Raspbian desktop appears for the first time, the system displays a series of setup dialogs. Follow the instructions in the dialogs in order to set your login password, and select an appropriate locale, time zone, and keyboard. Raspbian defaults to a UK keyboard layout, which will cause problems if you have a US or other keyboard. If you have an internet connection, you can also use the setup dialogs to perform a system update.

Once the setup is complete, take some time to admire the Raspbian desktop wallpaper and explore the system's infinite horizons, as I am doing in the following photograph:

In its heart (or in its seeds), Raspbian is just Debian Linux with an LXDE desktop and some special developer tools. If you are familiar with Debian or derivatives such as Ubuntu, you should feel right at home. Otherwise, you might want to explore the documentation and guides for beginners that are posted on the Raspberry Pi website, at `https://www.raspberrypi.org/help/`.

Now, as an exercise, let's share our Raspbian desktop through **Virtual Network Computing** (**VNC**) so that we can control it from a Windows, Mac, or Linux machine.

On the Pi, we need to first determine our local network address, which we will refer to as `<pi_ip_address>`. Open LXTerminal and run the following command:

```
$ ifconfig
```

The output should include a line beginning with something like `inet addr:192.168.1.93`, although the numbers will probably differ. In this example, `<pi_ip_address>` is `192.168.1.93`.

Now, we need to install a VNC server on the Pi by running the following command:

```
$ sudo apt-get install tightvncserver
```

To start the server, run this command:

```
$ tightvncserver
```

When prompted, enter a password, which other users must enter when connecting to this VNC server. Later, if you want to change the password, run this command:

```
$ vncpasswd
```

Unless the Pi (or the Ethernet socket to which it is connected) has a static IP address, the address may change whenever we reboot. Thus, upon reboot, we would need to run `ifconfig` again to determine the new address. Also, after rebooting, we need to run `tightvncserver` to relaunch the VNC server. For instructions on making the Pi's IP address static and automatically running `tightvncserver` upon boot, see Neil Black's online Raspberry Pi Beginner Guide, at `http://www.neil-black.co.uk/raspberrypi/raspberry-pi-beginners-guide/`.

Now, on another machine on the same local network, we can access the Pi's desktop through a VNC client. The steps are platform dependent, as follows:

1. On Windows, download VNC Viewer from `https://www.realvnc.com/download/`. Unzip it to any destination and run the executable file (such as `VNC-Server-6.4.0-Windows.exe`), which is inside the unzipped folder. Enter `vnc://<pi_ip_address>:5901` in the VNC server field and click on the **Connect** button. When prompted, enter the VNC password that you created earlier.
2. On Mac, open Safari and enter `vnc://<pi_ip_address>:5901` in the address bar. A window, **Connect to Shared Computer**, should appear. Click on the **Connect** button. When prompted, enter the VNC password that you created earlier.

3. Ubuntu normally comes with a VNC client called **Vinagre**. However, if we do not already have Vinagre, we can install it on Ubuntu or any Debian-based system by running the following command in Terminal:

```
$ sudo apt-get install vinagre
```

   Open Vinagre (it might be listed as **Remote Desktop Viewer** in the system's applications menu or launcher). Click on the **Connect** button in the toolbar. Enter `vnc://<pi_ip_address>:5901` in the **Host**: field. Click on the **Connect** button in the lower-right corner.

Now you know how to prepare and serve Pi.

Setting up the Raspberry Pi camera module

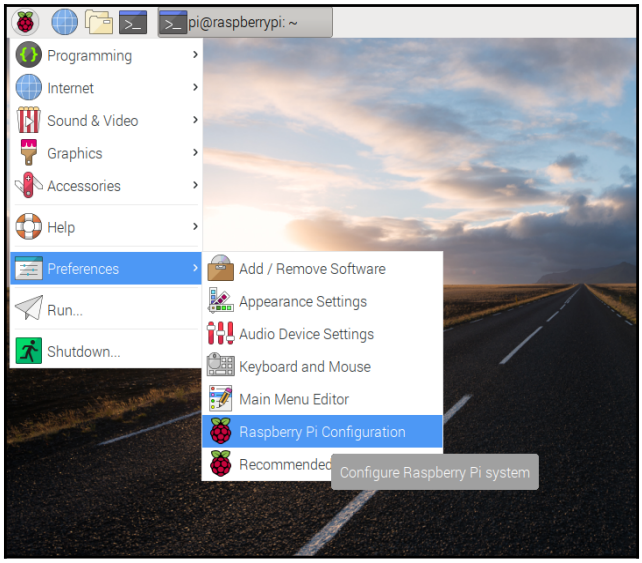
Raspbian supports most USB webcams out of the box. Also, it supports the following **Camera Serial Interface** (**CSI**) cameras, which offer faster transfer speeds:

- **Raspberry Pi camera module**: A $25 RGB camera
- **Pi NoIR**: A $30 variant of the same camera, with the **infrared radiation** (**IR**) block filter removed so that it is sensitive to not only visible light, but also the nearest-to-visible part of the infrared spectrum, **near infrared** (**NIR**)

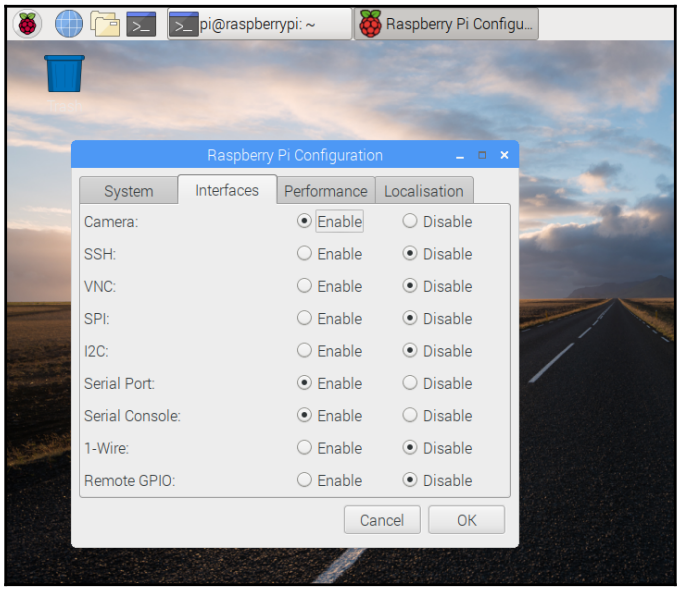
Compared to a USB webcam, the camera module or NoIR improves our chances of achieving high enough frame rates for interactive computer vision on the Pi. For this reason, I recommend these Pi-specific CSI cameras. However, in accordance with the low price, they have poor color rendition, mediocre auto-exposure, and fixed focus.

If in doubt, choose the camera module over the NoIR, because depending on the subject and lighting, NIR may interfere with vision, rather than aid it.

See the official tutorial at `http://www.raspberrypi.org/help/camera-module-setup/` for details about setting up either the camera module or the NoIR. Once the hardware is set up, you need to configure Raspbian to use the camera. From the Raspbian launch menu, select **Preferences** | **Raspbian Pi Configuration**, as shown in the following screenshot:



The **Raspberry Pi Configuration** window should appear. Go to the **Interfaces** tab and select **Camera: Enable**, as shown in the following screenshot:



Click on **OK** and, when prompted, reboot the system.

At the time of writing, the camera module and NoIR do not work out of the box with OpenCV. We need to load a kernel module that adds support for the cameras through the **Video for Linux 2** (**V4L2**) drivers. To do this for a single session, run the following command in Terminal:

```
$ sudo modprobe bcm2835-v4l2
```

Alternatively, to always load the kernel module upon bootup, run the following command, which appends the module to the `/etc/modules` file:

```
$ echo "bcm2835-v4l2" | sudo tee -a /etc/modules
```

Future versions of Raspbian (later than version 2018-11-13) might be preconfigured to use this kernel module. You can edit `/etc/modules` to check whether `bcm2835-v4l2` is already listed there.

Reboot the system again so that the kernel module is loaded. Now, we can use the `Camera` module or the NoIR with any camera software that supports V4L2 drivers, including OpenCV.

Finding OpenCV documentation, help, and updates

OpenCV's documentation is online, at `https://docs.opencv.org/master/`. The documentation includes a combined API reference for the latest OpenCV C++ API and its latest Python API (which is based on the C++ API). The latest Java API documentation is online, at `http://docs.opencv.org/master/javadoc/`.

If the documentation seems to leave your questions unanswered, try reaching out to the OpenCV community instead. The following sites are good venues for questions, answers, and shared experience:

- The official OpenCV forum, at `http://answers.opencv.org/questions/`
- The PyImageSearch site, where Adrian Rosebrock teaches computer vision and machine learning, at `https://www.pyimagesearch.com/`
- The support site for my OpenCV books, at `http://nummist.com/opencv/`

Finally, if you are an advanced user who wants to try new features, bug-fixes, and sample scripts from the latest (unstable) OpenCV source code, have a look at the project's repository at `https://github.com/opencv/opencv/`, as well as the repository for the `opencv_contrib` modules at `https://github.com/opencv/opencv_contrib/`.

Alternatives to Raspberry Pi

Besides Raspberry Pi, many other low-cost SBCs are suitable for running a desktop Linux distribution and OpenCV applications. The Raspberry Pi 3 models offer a quad-core ARMv8 processor and 1 GB RAM. However, some of the competing SBCs offer octa-core ARM processors and 2 GB RAM and can run more complex computer vision algorithms in real time. Moreover, unlike any current model of Pi, some of the competitors offer a USB 3 interface, which supports a wide range of high-resolution or high-speed cameras. These advantages tend to come with a higher price tag and higher power consumption. Here are some examples:

- Odroid XU4 (`https://www.hardkernel.com/shop/odroid-xu4-special-price/`): An octa-core SBC with 2 GB RAM and a USB 3 interface. It can run Ubuntu and other Linux distributions. At the time of writing, it is available from Odroid at a promotional price of $49.
- Banana Pi M3 (`http://www.banana-pi.org/m3.html`): An octa-core SBC with 2 GB RAM and a SATA interface for fast storage devices. It is compatible with many Raspberry Pi accessories and can run Ubuntu, Raspbian, and other Linux distributions. It typically costs around $75, if ordered factory-direct.
- Orange Pi 3 (`http://www.orangepi.org/Orange%20Pi%203/`): A quad-core SBC with 2 GB RAM and a USB 3 interface. It can run Ubuntu and other Linux distributions. It typically costs around $40, if ordered factory-direct.

If you would like to share your experience with using SBCs in computer vision projects, please write to me at `josephhowse@nummist.com`. I will post the community's wisdom to `http://nummist.com/opencv/`.

Summary

This was all a setup! I hear you gasp. Yes, but we did it for good reason. Now, we have a diverse set of development tools that will enable us to explore OpenCV in many contexts. Besides, it never hurts to learn some things about a lot of application frameworks and to have them all set up, in case someone asks us to do a project in a hurry.

Remember, James Bond has encyclopedic knowledge. In a highly symbolic conversation about rare and deadly fish, he goes toe-to-toe with Karl Stromberg, the diabolical oceanographer (*The Spy Who Loved Me*, 1977). Although we never see Bond studying the fish books, he must do it as bedtime reading after the camera cuts out.

The moral is, be prepared. Next, we will use OpenCV, along with several of the Python libraries and tools that we have installed, to build a GUI application that finds and classifies images on the web.