OpenCV By Example

Whether you are completely new to the concept of Computer Vision or have a basic understanding of it, this book will be your guide to understanding the basic OpenCV concepts and algorithms through real-world examples and projects.

Starting with the installation of OpenCV on your system and understanding the basics of image processing, we swiftly move on to creating optical flow video analysis or text recognition in complex scenes, and will take you through the commonly used Computer Vision techniques to build your own OpenCV projects from scratch.

By the end of this book, you will be familiar with the basics of OpenCV, such as matrix operations, filters, and histograms, as well as more advanced concepts such as segmentation, machine learning, complex video analysis, and text recognition.

Who this book is written for

If you are a software developer with a basic understanding of Computer Vision and image processing and want to develop interesting Computer Vision applications with OpenCV, this is the book for you. Knowledge of C++ is required.

What you will learn from this book

- Install OpenCV 3 on your operating system
- Create the required CMake scripts to compile the C++ application and manage its dependencies
- Get to grips with Computer Vision workflows and understand the basic image matrix format and filters
- Understand the segmentation and feature extraction techniques
- Remove backgrounds from a static scene to identify moving objects for video surveillance
- Track different objects in a live video using various techniques
- Use the new OpenCV functions for text detection and recognition with Tesseract

OpenCV By Example

Enhance your understanding of Computer Vision and image processing by developing real-world projects in OpenCV 3
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Getting Started with OpenCV'
- A synopsis of the book's content
- More information on OpenCV By Example
Prateek Joshi is a Computer Vision researcher and published author. He has over eight years of experience in this field with a primary focus on content-based analysis and deep learning. His work in this field has resulted in multiple patents, tech demos, and research papers at major IEEE conferences. He is the author of *OpenCV with Python By Example*, Packt Publishing.

He has won many hackathons using a wide variety of technologies related to image recognition. His blog has been visited by users in more than 200 countries, and he has been featured as a guest author in prominent tech magazines. He enjoys blogging on topics, such as artificial intelligence, abstract mathematics, and cryptography. You can visit his blog at [www.prateekvjoshi.com](http://www.prateekvjoshi.com).

He is an avid coder who is passionate about building game-changing products. He is particularly interested in intelligent algorithms that can automatically understand the content to produce scene descriptions in terms of constituent objects. He graduated from the University of Southern California and has worked for such companies as Nvidia, Microsoft Research, Qualcomm, and a couple of early stage start-ups in Silicon Valley. You can learn more about him on his personal website at [www.prateekj.com](http://www.prateekj.com).
David Millán Escrivá was eight years old when he wrote his first program on an 8086 PC with BASIC language, which enabled the 2D plotting of BASIC equations. He started with his computer development relationship and created many applications and games.

In 2005, he completed his studies in IT from the Universitat Politècnica de Valencia with honors in human-computer interaction supported by Computer Vision with OpenCV (v0.96). He had a final project based on this subject and published it on HCI Spanish Congress.

In 2014, he completed his Master’s degree in artificial intelligence, computer graphics, and pattern recognition, focusing on pattern recognition and Computer Vision.

He participated in Blender source code, an open source and 3D-software project, and worked in his first commercial movie, Plumíferos – Aventuras voladoras, as a computer graphics software developer.

David now has more than 13 years of experience in IT, with more than nine years of experience in Computer Vision, computer graphics, and pattern recognition, working on different projects and start-ups, applying his knowledge of Computer Vision, optical character recognition, and augmented reality.

He is the author of the DamilesBlog (http://blog.damiles.com), where he publishes research articles and tutorials on OpenCV, Computer Vision in general, and optical character recognition algorithms. He is the co-author of Mastering OpenCV with Practical Computer Vision Projects Book and also the reviewer of GnuPlot Cookbook by Lee Phillips, OpenCV Computer Vision with Python by Joseph Howse, Instant Opencv Starter by Jayneil Dalal and Sohil Patel, all published by Packt Publishing.
Vinícius Godoy is a computer graphics university professor at PUCPR. He started programming with C++ 18 years ago and ventured into the field of computer gaming and computer graphics 10 years ago. His former experience also includes working as an IT manager in document processing applications in Sinax, a company that focuses in BPM and ECM activities, building games and applications for Positivo Informática, including building an augmented reality educational game exposed at CEBIT and network libraries for Siemens Enterprise Communications (Unify).

As part of his Master’s degree research, he used Kinect, OpenNI, and OpenCV to recognize Brazilian sign language gestures. He is currently working with medical imaging systems for his PhD thesis. He was also a reviewer of the *OpenNI Cookbook*, *Packt Publishing*.

He is also a game development fan, having a popular site entirely dedicated to the field called *Ponto V* (http://www.pontov.com.br). He is the cofounder of a start-up company called Blackmuppet. His fields of interest includes image processing, Computer Vision, design patterns, and multithreaded applications.
Preface

OpenCV is one of the most popular libraries used to develop Computer Vision applications. It enables us to run many different Computer Vision algorithms in real time. It has been around for many years, and it has become the standard library in this field. One of the main advantages of OpenCV is that it is highly optimized and available on almost all the platforms.

This book starts off by giving a brief introduction of various fields in Computer Vision and the associated OpenCV functionalities in C++. Each chapter contains real-world examples and code samples to demonstrate the use cases. This helps you to easily grasp the topics and understand how they can be applied in real life. To sum it up, this is a practical guide on how to use OpenCV in C++ and build various applications using this library.

What this book covers

Chapter 1, Getting Started with OpenCV, covers installation steps on various operating systems and provides an introduction to the human visual system as well as various topics in Computer Vision.

Chapter 2, An Introduction to the Basics of OpenCV, discusses how to read/write images and videos in OpenCV, and also explains how to build a project using CMake.

Chapter 3, Learning the Graphical User Interface and Basic Filtering, covers how to build a graphical user interface and mouse event detector to build interactive applications.

Chapter 4, Delving into Histograms and Filters, explores histograms and filters and also shows how we can cartoonize an image.

Chapter 5, Automated Optical Inspection, Object Segmentation, and Detection, describes various image preprocessing techniques, such as noise removal, thresholding, and contour analysis.
Chapter 6, *Learning Object Classification*, deals with object recognition and machine learning, and how to use Support Vector Machines to build an object classification system.

Chapter 7, *Detecting Face Parts and Overlaying Masks*, discusses face detection and Haar Cascades, and then explains how these methods can be used to detect various parts of the human face.

Chapter 8, *Video Surveillance, Background Modeling, and Morphological Operations*, explores background subtraction, video surveillance, and morphological image processing and describes how they are connected to each other.

Chapter 9, *Learning Object Tracking*, covers how to track objects in a live video using different techniques, such as color-based and feature-based tracking.


Chapter 11, *Text Recognition with Tesseract*, delves deeper into the Tesseract OCR Engine to explain how it can be used for text detection, extraction, and recognition.
Getting Started with OpenCV

Computer Vision applications are interesting and useful, but the underlying algorithms are computationally intensive. With the advent of cloud computing, we are getting more processing power to work with. The OpenCV library enables you to run Computer Vision algorithms efficiently in real time. It has been around for many years and it has become the standard library in this field. One of the main advantages of OpenCV is that it is highly optimized and available on almost all platforms. The discussions in this book will cover everything, including the algorithm we are using, why we are using it, and how to implement it in OpenCV.

In this chapter, we are going to learn how to install OpenCV on various operating systems. We will discuss what OpenCV offers out of the box and the various things that we can do using the in-built functions.

By the end of this chapter, you will be able to answer the following questions:

- How do humans process visual data and how do they understand image content?
- What can we do with OpenCV and what are the various modules available in OpenCV that can be used to achieve those things?
- How to install OpenCV on Windows, Linux, and Mac OS X?

Understanding the human visual system

Before we jump into OpenCV functionalities, we need to understand why those functions were built in the first place. It's important to understand how the human visual system works so that you can develop the right algorithms. The goal of the Computer Vision algorithms is to understand the content of images and videos. Humans seem to do it effortlessly! So, how do we get machines to do it with the same accuracy?
Let’s consider the following figure:

The human eye captures all the information that comes along such as color, shapes, brightness, and so on. In the preceding image, the human eye captures all the information about the two main objects and stores it in a certain way. Once we understand how our system works, we can take advantage of this to achieve what we want. For example, here are a few things we need to know:

- Our visual system is more sensitive to low frequency content than high frequency content. Low frequency content refers to planar regions where pixel values don’t change rapidly and high frequency content refers to regions with corners and edges, where pixel values fluctuate a lot. You will have noticed that we can easily see if there are blotches on a planar surface, but it’s difficult to spot something like that on a highly textured surface.
- The human eye is more sensitive to changes in brightness as compared to changes in color.
- Our visual system is sensitive to motion. We can quickly recognize if something is moving in our field of vision even though we are not directly looking at it.
• We tend to make a mental note of salient points in our field of vision. Let's consider a white table with four black legs and a red dot at one of the corners of the table surface. When you look at this table, you'll immediately make a mental note that the surface and legs have opposing colors and there is a red dot on one of the corners. Our brain is really smart that way! We do this automatically so that we can immediately recognize it if we encounter it again.

To get an idea of our field of view, let's take a look at the top view of a human and the angles at which we see various things:

Our visual system is actually capable of a lot more things, but this should be good enough to get us started. You can explore further by reading up on Human Visual System Models on the internet.

**How do humans understand image content?**

If you look around, you will see a lot of objects. You may encounter many different objects every day, and you recognize them almost instantaneously without any effort. When you see a chair, you don't wait for a few minutes before realizing that it is, in fact, a chair. You just know that it's a chair right away! Now, on the other hand, computers find it very difficult to do this task. Researchers have been working for many years to find out why computers are not as good as we are at this.
To get an answer to this question, we need to understand how humans do it. The visual data processing happens in the ventral visual stream. This ventral visual stream refers to the pathway in our visual system that is associated with object recognition. It is basically a hierarchy of areas in our brain that helps us recognize objects. Humans can recognize different objects effortlessly, and we can cluster similar objects together. We can do this because we have developed some sort of invariance toward objects of the same class. When we look at an object, our brain extracts the salient points in such a way that factors such as orientation, size, perspective, and illumination don't matter.

A chair that is double the normal size and rotated by 45 degrees is still a chair. We can easily recognize it because of the way we process it. Machines cannot do this so easily. Humans tend to remember an object based on its shape and important features. Regardless of how the object is placed, we can still recognize it. In our visual system, we build these hierarchical invariances with respect to position, scale, and viewpoint that help us to be very robust.

If you look deeper in our system, you will see that humans have cells in their visual cortex that can respond to shapes, such as curves and lines. As we move further along our ventral stream, we will see more complex cells that are trained to respond to more complex objects, such as trees, gates, and so on. The neurons along our ventral stream tend to show an increase in the size of the receptive field. This is coupled with the fact that the complexity of their preferred stimuli increases as well.

**Why is it difficult for machines to understand image content?**

We now understand how visual data enters the human visual system and how our system processes it. The issue is that we still don't completely understand how our brain recognizes and organizes this visual data. We just extract some features from images and ask the computer to learn from them using machine learning algorithms. We still have those variations such as shape, size, perspective, angle, illumination, occlusion, and so on. For example, the same chair looks very different to a machine when you look at it from the side view. Humans can easily recognize that it's a chair regardless of how it's presented to us. So, how do we explain this to our machines?
One way to do this would be to store all the different variations of an object, including *sizes, angles, perspectives,* and so on. But this process is cumbersome and time-consuming! Also, it's actually not possible to gather data that can encompass every single variation. The machines will consume a huge amount of memory and a lot of time to build a model that can recognize these objects. Even with all this, if an object is partially occluded, computers still won't be able to recognize it. This is because they think that this is a new object. So, when we build a Computer Vision library, we need to build the underlying functional blocks that can be combined in many different ways to formulate complex algorithms. OpenCV provides a lot of these functions and they are highly optimized. So, once we understand what OpenCV provides out of the box, we can use it effectively to build interesting applications. Let's go ahead and explore this in the next section.

What can you do with OpenCV?

Using OpenCV, you can pretty much do every Computer Vision task that you can think of. Real-life problems require you to use many blocks together to achieve the desired result. So, you just need to understand what modules and functions to use to get what you want. Let's understand what OpenCV can do out of the box.

In-built data structures and input/output

One of the best things about OpenCV is that it provides a lot of in-built primitives to handle operations related to image processing and Computer Vision. If you have to write something from scratch, you will have to define things, such as an *image, point, rectangle,* and so on. These are fundamental to almost any Computer Vision algorithm. OpenCV comes with all these basic structures out of the box, and they are contained in the *core* module. Another advantage is that these structures have already been optimized for speed and memory, so you don't have to worry about the implementation details.

The *imgcodecs* module handles reading and writing image files. When you operate on an input image and create an output image, you can save it as a *JPG* or a *PNG* file with a simple command. You will be dealing with a lot of video files when you are working with cameras. The *videoio* module handles everything related to the input/output of video files. You can easily capture a video from a webcam or read a video file in many different formats. You can even save a bunch of frames as a video file by setting properties such as frames per second, frame size, and so on.
Image processing operations

When you write a Computer Vision algorithm, there are a lot of basic image processing operations that you will use over and over again. Most of these functions are present in the `imgproc` module. You can do things such as image filtering, morphological operations, geometric transformations, color conversions, drawing on images, histograms, shape analysis, motion analysis, feature detection, and so on.

Let's consider the following figure:

![Image processing example](image.png)

The right-hand side image is a rotated version of the left-hand side image. We can do this transformation with a single line in OpenCV. There is another module called `ximgproc` that contains advanced image processing algorithms such as structured forests for edge detection, domain transform filters, adaptive manifold filters, and so on.

Building GUI

OpenCV provides a module called `highgui` that handles all the high-level user interface operations. Let's say that you are working on a problem and you want to check what the image looks like before you proceed to the next step. This module has functions that can be used to create windows to display images and/or video. There is also a waiting function that will wait until you hit a key on your keyboard before it goes to the next step. There is a function that can detect mouse events as well. This is very useful to develop interactive applications. Using this functionality, you can draw rectangles on these input windows and then proceed based on the selected region.
Consider the following image:

As you can see, we have drawn a green rectangle on the image and applied a negative film effect to that region. Once we have the coordinates of this rectangle, we can operate only on that region.

**Video analysis**

Video analysis includes tasks such as analyzing the motion between successive frames in a video, tracking different objects in a video, creating models for video surveillance, and so on. OpenCV provides a module called video that can handle all of this. There is a module called videostab that deals with video stabilization. Video stabilization is an important part of video cameras. When you capture videos by holding the camera in your hands, it's hard to keep your hands perfectly steady. If you look at that video as it is, it will look bad and jittery. All modern devices use video stabilization techniques to process the videos before they are presented to the end user.
3D reconstruction

3D reconstruction is an important topic in Computer Vision. Given a set of 2D images, we can reconstruct the 3D scene using the relevant algorithms. OpenCV provides algorithms that can find the relationship between various objects in these 2D images to compute their 3D positions. We have a module called calib3d that can handle all this. This module can also handle camera calibration, which is essential to estimate the parameters of the camera. These parameters are basically the internal parameters of any given camera that uses them to transform the captured scene into an image. We need to know these parameters to design algorithms, or else we might get unexpected results. Let's consider the following figure:

As shown in the preceding image, the same object is captured from multiple poses. Our job is to reconstruct the original object using these 2D images.
Chapter 1

Feature extraction
As discussed earlier, the human visual system tends to extract the salient features from a given scene so that it can be retrieved later. To mimic this, people started designing various feature extractors that can extract these salient points from a given image. Some of the popular algorithms include SIFT (Scale Invariant Feature Transform), SURF (Speeded Up Robust Features), FAST (Features from Accelerated Segment Test), and so on. There is a module called features2d that provides functions to detect and extract all these features. There is another module called xfeatures2d that provides a few more feature extractors, some of which are still in the experimental phase. You can play around with these if you get a chance. There is also a module called bioinspired that provides algorithms for biologically inspired Computer Vision models.

Object detection
Object detection refers to detecting the location of an object in a given image. This process is not concerned with the type of object. If you design a chair detector, it will just tell you the location of the chair in a given image. It will not tell you whether it's a red chair with a high back or a blue chair with a low back. Detecting the location of objects is a very critical step in many Computer Vision systems. Consider the following image:
If you run a chair detector on this image, it will put a green box around all the chairs. It won't tell you what kind of chair it is! Object detection used to be a computationally intensive task because of the number of calculations required to perform the detection at various scales. To solve this, Paul Viola and Michael Jones came up with a great algorithm in their seminal paper in 2001. You can read it at https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf. They provided a fast way to design an object detector for any object. OpenCV has modules called objdetect and xobjdetect that provide the framework to design an object detector. You can use it to develop detectors for random items such as sunglasses, boots, and so on.

**Machine learning**

Computer Vision uses various machine learning algorithms to achieve different things. OpenCV provides a module called ml that has many machine learning algorithms bundled into it. Some of the algorithms include Bayes Classifier, K-Nearest Neighbors, Support Vector Machines, Decision Trees, Neural Networks, and so on. It also has a module called flann that contains algorithms for fast-nearest-neighbor searches in large datasets. Machine learning algorithms are used extensively to build systems for object recognition, image classification, face detection, visual searches, and so on.

**Computational photography**

Computational photography refers to using advanced image processing techniques to improve the images captured by cameras. Instead of focusing on optical processes and image capture methods, computational photography uses software to manipulate visual data. Some applications include high dynamic range imaging, panoramic images, image relighting, light field cameras, and so on.

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you. Instructions for running example are available in the README.md file present in the root folder of each project.
Let's take a look at the following image:

![Image](https://pixabay.com/en(hdr-high-dynamic-range-landscape-806260/)

Look at those vivid colors! This is an example of a high dynamic range image and it wouldn't be possible to get this using conventional image capture techniques. To do this, we have to capture the same scene at multiple exposures, register those images with each other, and then blend them nicely to create this image. The `photo` and `xphoto` modules contain various algorithms that provide algorithms pertaining to computational photography. There is a module called `stitching` that provides algorithms to create panoramic images.

The preceding image can be found at [https://pixabay.com/en(hdr-high-dynamic-range-landscape-806260/).](https://pixabay.com/en(hdr-high-dynamic-range-landscape-806260/)
Shape analysis

The notion of shape is crucial in Computer Vision. We analyze the visual data by recognizing various different shapes in the image. This is actually an important step in many algorithms. Let's say you are trying to identify a particular logo in an image. Now, you know that it can appear in various shapes, orientations, sizes, and so on. One good way to get started is to quantify the characteristics of the shape of the object. The module shape provides all the algorithms required to extract different shapes, measure similarities between them, transform shapes of objects, and so on.

Optical flow algorithms

Optical flow algorithms are used in videos to track features across successive frames. Let's say you want to track a particular object in a video. Running a feature extractor on each frame would be computationally expensive; hence, the process would be slow. So, you just need to extract the features from the current frame and then track these features in successive frames. Optical flow algorithms are heavily used in video-based applications in Computer Vision. The optflow module contains a number of algorithms required to perform optical flow. There is also a module called tracking that contains more algorithms that can be used to track features.

Face and object recognition

Face recognition refers to identifying the person in a given image. This is not the same as face detection where you identify the location of a face in the given image. So, if you want to build a practical biometric system that can recognize the person in front of the camera, you first need to run the face detector that can identify the location of the face, and then, run a face recognizer that can recognize who that person is. There is a module called face that deals with face recognition.

As discussed earlier, Computer Vision tries to model algorithms based on how humans perceive the visual data. So, it would be helpful to find salient regions and objects in the images that can help different applications, such as object recognition, object detection and tracking, and so on. There is a module called saliency that's designed for this purpose. It provides algorithms that can detect salient regions in static images and videos.
Surface matching
We are increasingly interacting with devices that can capture the 3D structure of the objects around us. These devices basically capture the depth information along with the regular 2D color images. So, it's important for us to build algorithms that can understand and process 3D objects. Kinect is a good example of a device that captures the depth information along with the visual data. The task at hand is to recognize the input 3D object by matching it with one of the models in our database. If we have a system that can recognize and locate objects, then it can be used for many different applications. There is a module called surface_matching that contains algorithms for 3D object recognition and a pose estimation algorithm using 3D features.

Text detection and recognition
Identifying text in a given scene and recognizing the content is becoming increasingly important. Some applications include nameplate recognition, recognizing road signs for self-driving cars, book scanning to digitize the contents, and so on. There is a module called text that contains various algorithms to handle text detection and recognition.

Installing OpenCV
Let's see how to get OpenCV up-and-running on various operating systems.

Windows
To keep things easy, let's install OpenCV using prebuilt libraries. Let's go to http://opencv.org and download the latest version for Windows. The current version is 3.0.0, and you can go to the OpenCV homepage to get the latest link to download the package.

You need to make sure you have admin rights before you proceed. The downloaded file will be an executable file, so just double-click on it to start the installation process. The installer expands the content into a folder. You will be able to choose the installation path and check the installation by inspecting the files.
Once you are done with the previous step, we need to set the OpenCV environment variables and add it to the system path to complete the installation. We will set up an environment variable that will hold the build directory of the OpenCV library. We will be using this in our projects. Open the terminal and type the following command:

```
C:\> setx -m OPENCV_DIR D:\OpenCV\Build\x64\vc11
```

We are assuming that you have a 64-bit machine with Visual Studio 2012 installed. If you have Visual Studio 2010, replace `vc11` with `vc10` in the preceding command. The path specified earlier is where we will have our OpenCV binaries, and you will see two folders inside this path called `lib` and `bin`. If you are using Visual Studio 2015, you should be able to compile OpenCV from scratch.

Let's go ahead and add the path to the `bin` folder of our system's path. The reason we need to do this is because we will be using the OpenCV library in the form of **Dynamic Link Libraries (DLLs)**. Basically, all the OpenCV algorithms are stored here, and our operating system will only load them during runtime. In order to do this, our operating system needs to know where they are located. The system's `PATH` variable will contain a list of all the folders where it can find the DLLs. So, naturally, we need to add the path to the OpenCV library to this list. Now, why do we need to do all this? Well, the other option is to copy the required DLLs to the same folder as the application's executable file (the `.exe` file). This is an unnecessary overhead, especially when we are working with many different projects.

We need to edit the `PATH` variable in order to add it to this folder. You can use software such as Path Editor to do this. You can download it from [https://patheditor2.codeplex.com](https://patheditor2.codeplex.com). Once you install it, start it and add the following new entry (you can right-click on the path to insert a new item):

```
%OPENCV_DIR%\bin
```

Go ahead and save it to the registry. We are done!

### Mac OS X

In this section, we will see how to install OpenCV on Mac OS X. Precompiled binaries are not available for Mac OS X, so we need to compile OpenCV from scratch. Before we proceed, we need to install CMake. If you don't have CMake already installed, you can download it from [https://cmake.org/files/v3.3/cmake-3.3.2-Darwin-x86_64.dmg](https://cmake.org/files/v3.3/cmake-3.3.2-Darwin-x86_64.dmg). It's a DMG file! So, once you download it, just run the installer.
Download the latest version of OpenCV from opencv.org. The current version is 3.0.0, and you can download it from https://github.com/Itseez/opencv/archive/3.0.0.zip.

Unzip the contents into a folder of your choice. OpenCV 3.0.0 also has a new package called opencv_contrib that contains user contributions that are not yet considered stable. One thing to keep in mind is that some of the algorithms in the opencv_contrib package are not freely available for commercial use. Also, installing this package is optional. OpenCV will work just fine if you don't install opencv_contrib. Since we are installing OpenCV anyway, it's good to install this package so that you can experiment with it later on (as opposed to going through the whole installation process again). This package is a great way to learn and play around with new algorithms. You can download it from https://github.com/Itseez/opencv_contrib/archive/3.0.0.zip.

Unzip the contents of the ZIP file into a folder of your choice. For convenience, unzip it into the same folder, as mentioned earlier, so that the opencv-3.0.0 and opencv_contrib-3.0.0 folders are in the same main folder.

We are now ready to build OpenCV. Open your terminal and navigate to the folder where you unzipped the contents of OpenCV 3.0.0. Run the following commands after substituting the right paths in the commands:

```bash
$ cd /full/path/to/opencv-3.0.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/full/path/to/opencv-3.0.0/build -D INSTALL_C_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D OPENCV_EXTRA_MODULES_PATH=/full/path/to/opencv_contrib-3.0.0/modules ../
```

It's time to install OpenCV 3.0.0. Go inside the /full/path/to/opencv-3.0.0/build directory and run the following commands on your terminal:

```bash
$ make -j4
$ make install
```

In the preceding command, the -j4 flag indicates that it is using four cores to install it. It's faster this way! Now, let's set the library path. Open your ~/.profile file in your terminal using the vi ~/.profile command, and add the following line:

```bash
export DYLD_LIBRARY_PATH=/full/path/to/opencv-3.0.0/build/lib:$DYLD_LIBRARY_PATH
```
Getting Started with OpenCV

We need to copy the pkg-config file opencv.pc to /usr/local/lib/pkgconfig and name it opencv3.pc. This way, if you already have an existing OpenCV 2.4.x installation, there will be no conflicts. Let's go ahead and do this:

```
$ cp /full/path/to/opencv-3.0.0/build/lib/pkgconfig/opencv.pc /usr/local/lib/pkgconfig/opencv3.pc
```

We need to update our PKG_CONFIG_PATH variable as well. Open your ~/.profile file, and add the following line:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:/$PKG_CONFIG_PATH
```

Reload your ~/.profile file using the following command:

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

We are done! Let's see if it's working:

```
$ cd /full/path/to/opencv-3.0.0/samples/cpp
$ g++ -ggdb `pkg-config --cflags --libs opencv3` opencv_version.cpp -o /tmp/opencv_version && /tmp/opencv_version
```

If you see Welcome to OpenCV 3.0.0 printed on your terminal, you are good to go. We will be using CMake to build our OpenCV projects throughout this book. We will cover this in more detail in the next chapter.

**Linux**

Let's see how to install OpenCV on Ubuntu. We need to install some dependencies before we begin. Let's install them using the package manager by running the following command on your terminal:

```
$ sudo apt-get -y install libopencv-dev build-essential cmake libopencv-devel libavcodec-dev libavfilter-dev libavformat-dev libswscale-dev libv4l-dev libxine-dev libxvidcore-dev x264 v4l-utils
```

Now that you have installed the dependencies, let's download, build, and install OpenCV:

```
$ wget "https://github.com/Itseez/opencv/archive/3.0.0.zip" -O opencv.zip
$ wget "https://github.com/Itseez/opencv_contrib/archive/3.0.0.zip" -O opencv_contrib.zip
$ unzip opencv.zip -d.
```
Chapter 1

$ unzip opencv_contrib.zip -d .
$ cd opencv-3.0.0
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/full/path/to/opencv-3.0.0/build -D INSTALL_C_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D OPENCV_EXTRA_MODULES_PATH=/full/path/to/opencv_contrib-3.0.0/modules ../
$ make -j4
$ sudo make install

Let's copy the pkg-config file's opencv.pc to /usr/local/lib/pkgconfig and name it opencv3.pc:

$ cp /full/path/to/opencv-3.0.0/build/lib/pkgconfig/opencv.pc /usr/local/lib/pkgconfig/opencv3.pc

We are done! We will now be able to use it to compile our OpenCV programs from the command line. Also, if you already have an existing OpenCV 2.4.x installation, there will be no conflicts. Let's check whether the installation is working properly:

$ cd /full/path/to/opencv-3.0.0/samples/cpp
$ g++ -ggdb 'pkg-config --cflags --libs opencv3' opencv_version.cpp -o /tmp/opencv_version
$ /tmp/opencv_version

If you see Welcome to OpenCV 3.0.0 printed on your terminal, you are good to go. In the following chapters, you will learn how to use CMake to build your OpenCV projects.

Summary

In this chapter, we learned how to install OpenCV across various operating systems. We discussed the human visual system and how humans process visual data. We understood why it's difficult for machines to do the same and what we need to consider while designing a Computer Vision library. We learned what can be done using OpenCV and the various modules that can be used to do those tasks.

In the next chapter, we will discuss how to operate on images and how we can manipulate them using various functions. We will also learn how to build a project structure for our OpenCV applications.
Where to buy this book

You can buy OpenCV By Example from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.