

TECHNOLOGY IN ACTION™



Raspberry Pi Image Processing Programming



Develop Real-Life Examples with
Python, Pillow, and SciPy

Ashwin Pajankar

apress®

Raspberry Pi Image Processing Programming

Develop Real-Life Examples with
Python, Pillow, and SciPy



Ashwin Pajankar

Apress®

Raspberry Pi Image Processing Programming: Develop Real-Life Examples with Python, Pillow, and SciPy

Ashwin Pajankar
Nashik, Maharashtra, India

ISBN-13 (pbk): 978-1-4842-2730-5
DOI 10.1007/978-1-4842-2731-2

ISBN-13 (electronic): 978-1-4842-2731-2

Library of Congress Control Number: 2017936370

Copyright © 2017 by Ashwin Pajankar

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr

Lead Editor: Celestin Suresh John

Technical Reviewer: Lentin Joseph

Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black,

Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John,

Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao,

Gwenan Spearing

Coordinating Editor: Sanchita Mandal

Copy Editor: Kezia Endsley

Compositor: SPi Global

Indexer: SPi Global

Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

Contents at a Glance

About the Author	xi
About the Technical Reviewer	xiii
Acknowledgments	xv
Introduction	xvii
■ Chapter 1: Introduction to Single Board Computers and Raspberry Pi.....	1
■ Chapter 2: Introduction to Python and Digital Image Processing.....	25
■ Chapter 3: Getting Started	41
■ Chapter 4: Basic Operations on Images	51
■ Chapter 5: Advanced Operations on Images	65
■ Chapter 6: Introduction to Scientific Python	81
■ Chapter 7: Transformations and Measurements	93
■ Chapter 8: Filters and Their Application.....	99
■ Chapter 9: Morphology, Thresholding, and Segmentation.....	111
Index.....	123

Contents

About the Author	xi
About the Technical Reviewer	xiii
Acknowledgments	xv
Introduction	xvii
■ Chapter 1: Introduction to Single Board Computers and Raspberry Pi.....	1
Single Board Computers (SBCs)	1
Differences Between SBCs and Regular Computers	2
System on Chips (SoCs).....	2
History of SBCs.....	3
SBC Families.....	3
The Raspberry Pi	4
Raspberry Pi Setup.....	5
Required Hardware.....	6
Preparation of the microSD Card for Raspberry Pi	9
Download the Required Free Software	10
Writing the Raspbian OS Image to the microSD Card.....	10
Altering the Contents of the config.txt File for a VGA Monitor	12
Bootting Up the Pi	13
Configuring the Pi	15
The Raspbian OS	18
The config.txt File.....	18

Connecting the Raspberry Pi to a Network and to the Internet.....	19
WiFi.....	19
Ethernet.....	21
Updating the Pi.....	22
Updating the Firmware.....	22
Updating and Upgrading Raspbian	22
Updating raspi-config	23
Shutting Down and Restarting Pi	24
Conclusion.....	24
■ Chapter 2: Introduction to Python and Digital Image Processing.....	25
A History of Python.....	25
Features of Python	26
Simple.....	26
Easy to Learn.....	27
Easy to Read.....	27
Easy to Maintain	27
Open Source	27
High-Level Language.....	27
Portable	27
Interpreted	28
Object-Oriented	28
Extensible	28
Extensive Libraries	28
Robust	28
Rapid Prototyping	29
Memory Management	29
Powerful	29
Community Support.....	29

Python 3	29
The Differences Between Python 2 and Python 3.....	30
Why Use Python 3.....	31
Python 2 and Python 3 on Raspbian.....	31
Running a Python Program and Python Modes	31
Interactive Mode.....	32
Normal Mode	32
IDEs for Python	33
IDLE	33
Geany.....	34
Introduction to Digital Image Processing	36
Signal Processing	36
Image Processing	37
Using Raspberry Pi and Python for Digital Image Processing (DIP).....	38
Conclusion.....	39
■ Chapter 3: Getting Started	41
Image Sources	41
Using the Webcam.....	42
The Pi Camera Module.....	44
Using Python 3 for Digital Image Processing	46
Working with Images.....	47
Conclusion.....	50
■ Chapter 4: Basic Operations on Images	51
Image Module.....	51
Splitting and Merging Image Channels.....	51
Image Mode Conversion	53
Image Blending.....	53
Resizing an Image	55

Rotating an Image	56
Crop and Paste Operations	57
Copying and Saving Images to a File.....	58
Knowing the Value of a Particular Pixel.....	58
ImageChops Module.....	59
ImageOps Module	61
Conclusion.....	63
■ Chapter 5: Advanced Operations on Images	65
The ImageFilter Module.....	65
The ImageEnhance Module	74
Color Quantization.....	76
Histograms and Equalization.....	77
Histogram Equalization.....	78
Conclusion.....	79
■ Chapter 6: Introduction to Scientific Python	81
The Scientific Python Stack.....	81
Installing the SciPy Stack	82
A Simple Program.....	82
Simple Image Processing	83
Introduction to NumPy	84
Matplotlib.....	86
Image Channels.....	89
Conversion Between PIL Image Objects and NumPy ndarrays.....	91
Conclusion.....	92

■ Chapter 7: Transformations and Measurements	93
Transformations	93
Measurements	95
Conclusion.....	98
■ Chapter 8: Filters and Their Application.....	99
Filters	99
Low-Pass Filters.....	100
High-Pass Filters	105
Fourier Filters	108
Conclusion.....	110
■ Chapter 9: Morphology, Thresholding, and Segmentation.....	111
Distance Transforms.....	111
Morphology and Morphological Operations.....	113
Structuring Element.....	113
Various Morphological Operations.....	113
Grayscale Morphological Operations.....	115
Thresholding and Segmentation	117
Conclusion.....	121
Book Summary.....	121
What's Next	121
Index.....	123

About the Author

Ashwin Pajankar is a polymath. He is a Science popularizer, a programmer, a maker, an author, and a YouTuber. He graduated from IIIT Hyderabad with MTech in Computer Science and Engineering. He has a keen interest in the promotion of science, technology, engineering, and mathematics (STEM) education. He has written three books with Packt Publication, six books with Leanpub, and one book with Apress Media, and he has also reviewed four books for Packt Publications. He's currently working on several more books with Apress Media as well.

His personal web site is found at www.AshwinPajankar.com.

His LinkedIn profile is found at <https://in.linkedin.com/in/ashwinpajankar>.

About the Technical Reviewer



Lentin Joseph is an author, entrepreneur, electronics engineer, robotics enthusiast, machine vision expert, embedded programmer, and the founder and the CEO of Qbotics Labs (<http://www.qboticslabs.com>) from India.

He completed his bachelor's degree in electronics and communication engineering at the Federal Institute of Science and Technology (FISAT), Kerala. For his final year engineering project, he made a social robot that interacted with people. The project was a huge success and was mentioned in many forms of

visual and print media. His robot can communicate with people and reply intelligently and has image-processing capabilities such as face, motion, and color detection. The entire project was implemented using the Python programming language. His interest in robotics, image processing, and Python started with that project.

After his graduation, he worked for three years at a startup company focusing on robotics and image processing. In the meantime, he learned to work with famous robotic software platforms such as Robot Operating System (ROS), V-REP, and Actin (a robotic simulation tool), as well as with image-processing libraries such as OpenCV, OpenNI, and PCL. He also knows about robot 3D designing and embedded programming on Arduino and Tiva Launchpad.

After three years, he started a new company called Qbotics Labs, which mainly focuses on research to build great products in domains such as robotics and machine vision. He maintains a personal web site (at <http://www.lentinjoseph.com>) and a technology blog called technolabsz (see <http://www.technolabsz.com>). He publishes his works on his tech blog. He was also a speaker at PyCon2013, India, on the topic of learning about robotics using Python.

Lentin is the author of the books, *Learning Robotics Using Python* (see <http://learn-robotics.com> to find out more) and *Mastering ROS for Robotics Programming* (see <http://mastering-ros.com> to find out more), both by Packt. The first book was about building an autonomous mobile robot using ROS and OpenCV. This book was launched at ICRA 2015 and was featured in the ROS blog, Robohub, OpenCV, the Python web site, and various other such forums. The second book is on mastering the Robot Operating System, which was also launched at ICRA 2016, and is one of the bestselling books on ROS. The third book is on ROS robotics projects (see <http://rosrobots.com>), and is expected to be published by April 2017.

He also reviewed one book about the Robot Operating System called *Effective Robotics Programming Using ROS* (<https://www.packtpub.com/hardware-and-creative/effective-robotics-programming-ros-third-edition>).

Lentin and his team were also winners of the HRATC 2016 challenge conducted as part of ICRA 2016. He was also a finalist in the ICRA 2015 challenge, HRATC (see <http://www.icra2016.org/conference/challenges/>).

Acknowledgments

Writing a book is a journey that I am glad I undertook. First, I want to thank my wife Kavitha, without whose support the journey would never have been possible. The journey spanned a few months but the experience will last a lifetime. I had my wife Kavitha with me onboard this journey and I wish to express my deepest gratitude to her. Without her unwavering support and affection, I couldn't have pulled it off.

I am grateful to the community of professionals, students, trainers, and teachers who, with their continual bombardment of queries, impelled me to learn more, simplify my knowledge and findings, and place it neatly in the book. This book is for all of them.

I want to thank my friends and colleagues—the practitioners from the industry and experts from academia—for their good counsel and filling me in with the knowledge on the latest in the fields of single board computers, computer vision, digital image processing, and Python.

I want to thank the technical reviewer for his vigilant reviews, suggestions, corrections, and expert opinion.

I consider myself very fortunate for the editorial assistance provided by Apress Media. This is my second book with Apress and collaborating with them on both the books has been fabulous. I am thankful to Celestin Suresh John, Senior Manager, Editorial Acquisition, Apress and Springer Science and Business Media Company, for giving me a long-desired opportunity to collaborate and write for Apress. I also want to acknowledge Sanchita Mandal, coordinating editor, Anila Vincent, development editor, and the team of associates from Apress Media who adeptly guided me through the entire process of preparation and publication.

Introduction

Why This Book?

I have been using Python for more than 10 years for a variety of tasks. Initially, I used it for GUI applications. Then I quickly moved to scientific uses as my academic projects demanded it. When I entered professional life, I used it for automation first and then for implementation of alert mechanisms. I have been using Python the last six years for various fields like scientific computing, the Internet of Things, and single board computers. I have written plenty of Python code all these years. I prefer Python to Bash scripting, which offers limited capabilities to users like me. Over the last 10 years, I have worked as a developer, an R&D engineer, a maker, an author, and a QA specialist. I used Python in every single role.

This is my third dedicated book on the topic of digital image processing. I have extensive work experience in the field of digital image processing and computer vision. Almost all of the digital image processing programming I did was in C++ and Python. For beginners in image processing programming, I always recommend Python, as it is easy to learn. Also, if you are working in the research areas like Medical imaging, optics, and biology, where the core expertise is not computer science but you must use digital image processing, Python is the best choice, as you won't get bogged down with its syntax.

To prepare for and write this book, I spent a lot of time writing code examples from scratch, testing them, and then checking the PEP-8 compatibility. Also, I spent numerous hours writing and editing the text in order to explain the image processing concepts in very simple and plain language. I did not keep track of the time I spent posting on various forums and discussing problems with colleagues from the industry and academia. I have poured my heart and soul into writing this book. I hope that you readers who want to get started with digital image processing and single board computers will find this book immensely valuable.

I wrote this book to share my knowledge and experiences while programming in the field of digital image processing with Python 3 and Raspberry Pi. I explored multiple techniques, frameworks, and libraries for capturing, processing, and displaying digital images in this book. I hope you will enjoy reading and following the book as much as I enjoyed writing it. The book covers the following topics:

- Introduction to single board computers, Python 3, and Raspberry Pi
- Interfacing Raspberry Pi with the Pi Camera module and a Webcam
- Exploring various image-processing libraries like Pillow and `scipy.ndimage`
- Introduction to additional libraries such as NumPy, matplotlib, and Tkinter, which assist us in image processing

Who This Book Is For

Raspberry Pi enthusiasts are the main audience of this book. This includes a large and diverse set of people such as developers, students, researchers, and novice learners. The book is for those who have some prior knowledge of the Python programming language. If you are a developer, student, or a researcher with some experience in Python programming, you can quickly learn the concepts related to digital image processing with your favorite little British computer, Raspberry Pi.

What This Book Is Not

This is not a book for learning Python 3 programming and syntax from scratch. It is more of a DIY cookbook for Raspberry Pi and digital image processing. If your understanding of coding is limited or you are not from computer science background, you will find it difficult to follow this book.

How This Book Is Organized

This book has nine chapters. Here is a sneak peek into the chapters of the book:

Chapter 1: This chapter introduces the readers to the history and the philosophy of single board computers. Then it explores Raspberry Pi basics. It teaches readers to set up the Raspberry Pi and connect it to a network.

Chapter 2: This chapter introduces the readers to the history and the philosophy of Python. It teaches you how to install Python and how to set up the environment for Python 3 programming. It also explores new features of Python 3 in brief and introduces the readers to a few popular Python 3 IDEs. The chapter concludes with a brief introduction to the concepts related to digital image processing and related areas.

Chapter 3: The aim of this chapter is to quickly get the readers started with digital image processing in Python 3. The chapter introduces the readers to capture images with a Webcam and Pi Camera. It introduces the readers to Pillow for image processing and to Tkinter for GUI.

Chapter 4: This chapter serves to introduce basic arithmetic and logical operations on Image. Readers also study the image channels in this chapter.

Chapter 5: This chapter explores advanced operations like filtering and effects on images. Readers are introduced to the concept of the histogram and its computation.

Chapter 6: This chapter introduces you to the world of scientific image processing. We will install the SciPy stack on Raspberry Pi. We also get started with SciPy, NumPy, and matplotlib in this chapter.

Chapter 7: This chapter helps readers understand the measurements and transformations using the `scipy.ndimage` module of SciPy library.

Chapter 8: This chapter introduces readers to the important concept of filtering. We will study types of filters (such as low-pass and high-pass filters) and their applications.

Chapter 9: This chapter helps readers understand the concepts related to morphology. It also covers thresholding. Finally, the chapter uses both concepts to achieve segmentation in binary images.

How Do You Get the Most Out of This Book

It is easy to leverage the book to gain the maximum amount of information you can, simply by abiding to the following:

- Read the chapters thoroughly. Perform the examples hands-on by following the step-by-step instructions stated in the code. Do *not* skip any code example. If need be, repeat the examples a second time or until the concept is firmly etched in your mind.
- Join a Python community or discussion forum.
- Read the online documentation available for various image-processing frameworks in Python 3.
- Read blogs covering computer vision, signal and image processing, and Python 3.

Where Next?

I endeavored to unleash the power of digital image processing libraries for Python 3 as an aid to the Raspberry Pi community. I recommend you read the book from cover to cover without skipping any of the chapters, text, code examples, or exercises.

I wish you well in exploring Python and Raspberry Pi!

A Quick Word for the Instructor's Fraternity

Attention has been paid in arriving at the sequence of chapters and also to the flow of topics within each chapter. This is done particularly to assist my fellow instructors and academicians in carving out a syllabus for their training from the Table of Contents (ToC).

I ensured that each concept discussed in the book includes adequate hands-on content to enable you to teach better and provide ample hands-on practice to your students.

A Quick Word for the Non-Computer Science Readers

If you are reading this book and do not belong to the computer science-related field then you might face a few hurdles to understanding the concepts and mathematics behind them. If you are working in allied fields like mathematics, electronics, signal processing, bio-medical imaging, digital imaging, or bio-informatics, and you are reading this book for work, I recommend reading a few books on the fundamentals of the topics explained here. This is essential, as the book focuses more on the practicals.

Happy learning and Pythoning!!!

Author, Ashwin Pajankar



Introduction to Single Board Computers and Raspberry Pi

We will start this exciting journey exploring the scientific domain of digital image processing with Raspberry Pi. To begin the journey, you must be comfortable with the basics of single board computers (SBCs) and with Raspberry Pi. This chapter discusses the definition, history, and philosophy behind SBCs. It compares SBCs to regular computers. Then it moves toward the most popular and best selling SBC of all time, the Raspberry Pi. By the end of this chapter, you will have adequate knowledge to set up your own Raspberry Pi independently. This chapter aims to make you comfortable with the basic concepts of SBCs and Raspberry Pi setup.

Single Board Computers (SBCs)

A single board computer (referred to as an SBC from now on) is a fully functional computer system built around a single printed circuit board. An SBC has a microprocessor(s), memory, input/output, and other features required of a minimally functioning computer. Unlike with desktop personal computers (PC), most SBCs do not have expansion slots for peripheral functions or expansion. As all the components—processor(s), RAM, and GPU, etc.—are integrated on a single printed circuit board (PCB), you cannot upgrade an SBC.

Few SBCs are made to plug into a backplane for system expansion. SBCs come in many varieties, sizes, shapes, form factors, and feature sets. Due to the advances in the electronics and semiconductor technologies, prices of most SBCs are very low. One of the most important features of SBCs is their inexpensive cost. With a price at around \$50 a piece, you have in your hand a development tool suitable for new applications, hacking, debugging, testing, hardware development, and automation systems.

SBCs are usually manufactured with the following form factors:

- Pico-ITX
- PXI
- Qseven
- VMEbus
- VPX
- VXI
- AdvancedTCA
- CompactPCI
- Embedded Compact Extended (ECX)
- Mini-ITX
- PC/104
- PICMG

Differences Between SBCs and Regular Computers

Table 1-1 lists the differences between SBCs and regular computers.

Table 1-1. *Differences Between SBCs and Regular Computers*

Single Board Computer	Regular Computer
Not modular	Modular
Components cannot be upgraded or replaced	Components can be upgraded or replaced
A System On Chip	Not a System On Chip
Has a small form factor	Has a large form factor
Is portable	Is mostly non-portable or semi-portable
Consumes less power	Consumes more power
Cheaper than a regular computer	Costs more than a SBC

System on Chips (SoCs)

All the SBCs are predominantly SoCs. A system on a chip (SoC) is an integrated circuit (IC) that has all the components of a computer on a single chip. SoCs are very common with mobile electronic devices because of their low power consumption and versatility. SoCs are widely used in mobile phones, SBCs, and embedded hardware. A SoC includes all the hardware and software needed for its operation.

SoC versus Regular CPU

The biggest advantage of using a SoC is its size. If you use a CPU, it's very hard to make a compact computer, only because of the number of individual chips and other components that you need to arrange on a board. However, when using SoCs, you can place complete application-specific computing systems in smartphones and tablets, and still have plenty of space for batteries, the antenna, and other add-ons required for remote telephony and data communication.

Due to the very high level of integration and the compact size, a SoC uses considerably less power than a regular CPU. This is a significant advantage of SoCs when it comes to mobile and portable systems. Also, reducing the number of chips by eliminating redundant ICs on a computer board results in a compact board size.

History of SBCs

Dyna-Micro was the first true SBC. It was based on the Intel C8080A and used Intel's first EPROM, the C1702A. The Dyna-Micro was rebranded and marketed by E&L Instruments of Derby, CT in 1976 as the MMD-1 (Mini-Micro Designer 1). It became famous as the leading example of microcomputers. SBCs were very popular in the earlier days of computing, as many home computers were actually SBCs. However, with the rise of PCs, the popularity of SBCs declined. Since 2010, there has been a resurgence in the popularity of SBCs due to their lower production costs.

Apart from the MMD-1, here are a few other popular historical SBCs:

- The BBC Micro was built around an MOS technology 6502A processor running at 2MHz.
- The Ferguson Big Board II was a Zilog Z80-based computer running at 4MHz.
- The Nascom was another Zilog Z80-based computer.

SBC Families

Based on the manufacturers and designers, the SBCs are grouped into families, models, and generations. Here are a few popular SBC families:

- Raspberry Pi by the Raspberry Pi Foundation
- Banana Pi and Banana Pro
- Intel Edison and Galileo
- CubieBoard
- BeagleBone and BeagleBoard

The Raspberry Pi

The Raspberry Pi is a family of credit card-sized SBCs developed in the United Kingdom by the Raspberry Pi Foundation. The Raspberry Pi Foundation formed in 2009. The aim behind developing Raspberry Pi was to promote the teaching of basic computer science in schools and developing countries by providing a low-cost computing platform.

Raspberry Pi Foundation’s Raspberry Pi was released in 2012. It was a massive hit and sold over two million units in two years. Subsequently, the Raspberry Pi Foundation revised versions of the Raspberry Pi. They also released other accessories for the Pi.

You can find more information about the Raspberry Pi foundation on the Raspberry Pi Foundation’s web site at <https://www.raspberrypi.org>.

The product page for Raspberry Pi’s current production models and other accessories is at <https://www.raspberrypi.org/products>.

I have written, executed, and tested all the code examples in this book on Raspberry Pi Models B+, 2B, and 3B. Raspberry Pi 3 Model B (also known as 3B) is the most recent model of Raspberry Pi. Table 1-2 lists the specifications of the Raspberry Pi 3, Model B.

Table 1-2. *Specifications of the Raspberry Pi 3 Model B*

Release Date	February 2016
Architecture	ARMv8
SoC broadcom	BCM2837
CPU	1.2GHz 64-bit quad-core ARM Cortex-A53
GPU	Broadcom VideoCore IV (3D part of GPU @ 300MHz, video part of GPU @ 400MHz)
Memory	1 GB (shared with GPU)
USB	2.0 ports 4
Video output	HDMI rev 1.3 and Composite Video RCA jack
On-board storage	Micro SDHC slot
On-board network	10/100 Mbps Ethernet, Bluetooth, and WiFi
Power source	5V via MicroUSB
Power ratings	800 mA (4W)

Figure 1-1 shows the top view of Raspberry Pi 3 Model B. The components relevant to this book are labeled in the image.

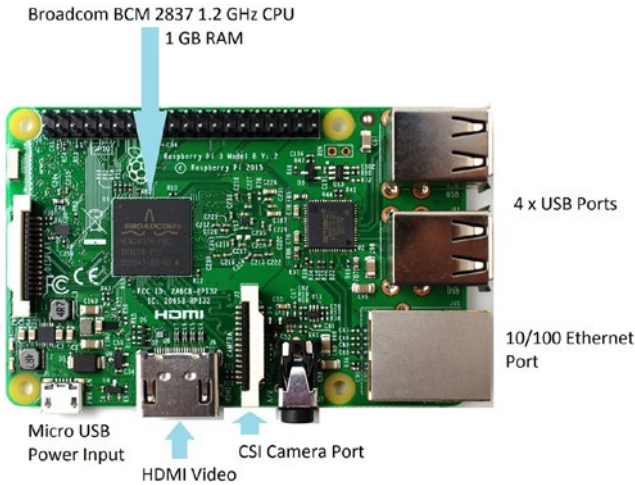


Figure 1-1. Raspberry Pi 3 Model B, top view

Figure 1-2 shows the bottom view of Raspberry Pi 3 Model B.

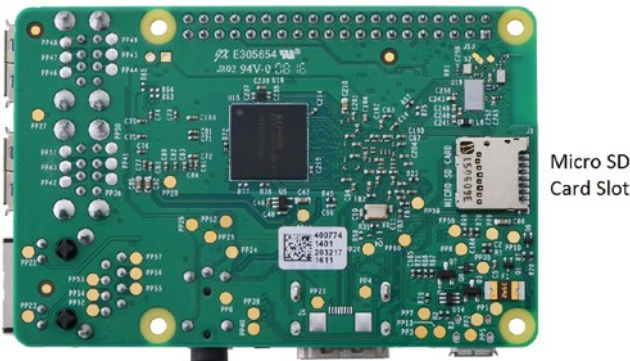


Figure 1-2. Raspberry Pi 3 Model B, bottom view

You can get more Information on Raspberry Pi 3 Model B by visiting the product page at <https://www.raspberrypi.org/products/raspberry-pi-3-model-b>.

Raspberry Pi Setup

You have to set up Raspberry Pi before you can use it for exploration and adventure. This section explains in detail how to set it up. As mentioned earlier, I am using Raspberry Pi 3 Model B for this setup. The setup process is exactly same for Raspberry Pi 2 Model B and Raspberry Pi 1 Model B+. Here is the list of hardware materials to be procured for the setup.

Required Hardware

The following hardware is required to set up the Raspberry Pi.

Raspberry Pi

You need to use Raspberry Pi 3 Model B or Raspberry Pi 2 Model B or Raspberry Pi 1 Model B+ for the setup.

Computer

A Windows computer or laptop with an Internet connection is required. You need to use a computer to prepare a microSD card with a Raspbian OS image for the Pi.

I/O Devices

A standard USB keyboard and a USB mouse are required.

microSD Card

A microSD card (see Figure 1-3) with at least 8GB of storage is needed. You'll use the card for secondary storage for the Pi. A card of Class 10 is recommended as the data transfer speed with class 10 is great. I recommend using at least an 8GB card to be on the safe side. Choosing a 16GB card will be adequate for most of the use cases.

■ **Note** Before purchasing a card, visit http://elinux.org/RPi_SD_cards to check the compatibility of the card with the Raspberry Pi.



Figure 1-3. Class 10 microSD card

Power Supply

For all the Raspberry Pi models, a 5V Micro USB power supply unit (PSU) is required. The recommended current capacity of the PSU for Raspberry Pi 3 Model B is 2.5 amp. For all the other models, a 2 amp PSU is more than enough.

You can find Raspberry Pi's official power supply (see Figure 1-4) at <https://thepihut.com/products/official-raspberry-pi-universal-power-supply>.

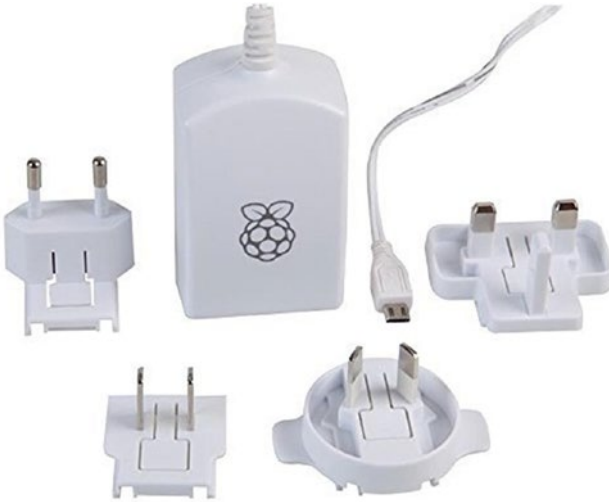


Figure 1-4. *Raspberry Pi official power supply*

Card Reader

You also need a card reader. Many laptops have a built-in SD card reader.

If the laptop or the card reader works with an SD card only, you need a additional microSD-to-SD card adapter. Figure 1-5 shows a card reader and an adapter.



Figure 1-5. Card reader and microSD-to-SD adapter

Monitor

You need an HDMI or VGA monitor.

For an HDMI monitor, you need an HDMI male-to-male cable (see Figure 1-6). It is typically packaged with the HDMI monitor.



Figure 1-6. HDMI male-to-male cable

For the VGA monitor, you need a VGA cable (see Figure 1-7). This too is usually packaged with the VGA monitor.



Figure 1-7. VGA male-to-male cable (also known as a D-SUB cable)

If you are using a VGA monitor, you need an HDMI to VGA adapter (see Figure 1-8), because Raspberry Pi has an HDMI port only for the video output.



Figure 1-8. HDMI to VGA adapter

Preparation of the microSD Card for Raspberry Pi

Manually preparing the microSD card for Pi is the best way of installing any OS into a microSD card for SBCs. Many users (including me) prefer it, because it allows the contents of microSD card to be modified manually (if needed) before it is used for booting. The other way to prepare the microSD is to use *NOOBS (New Out Of the Box Software)*, which I have not used in this book.

This approach allows you to access to the configuration files like `/boot/config.txt` before booting. You might have to modify the configuration files in a few cases (we will discuss that soon) before booting up the Pi. The default Raspbian image has two partitions, called boot and system. Be sure to use at least a 16GB microSD card for the Pi considering any possible future upgrades to the OS.

Download the Required Free Software

Let's download the required software.

Download Accelerator Plus

Download the Download Accelerator Plus setup from its download page (<http://www.speedbit.com/dap/download/downloading.asp>). This freeware is used to manage downloads. It is useful for large downloads, as you can pause and resume downloads. If your computer shuts down suddenly or the Internet is interrupted, it resumes the download from the last checkpoint. Once you download and install it, use it to manage any further downloads.

Win32 Disk Imager

Download the Win32 Disk Imager setup from its download page (<https://sourceforge.net/projects/win32diskimager/files/latest/download>). Install it.

WinZip or WinRaR

You need a file extraction utility. Download WinZip (<http://www.winzip.com/win/en/index.htm>) or WinRaR (<http://www.win-rar.com/download.html>). Install the one you chose.

Download and Extract the Raspbian OS Image

You will use the Raspbian OS for the Pi. (We will discuss Raspbian in detail in a later part of the chapter.) As of now, download the latest ZIP of the image of the Raspbian OS from <https://www.raspberrypi.org/downloads/raspbian>. Extract the image ZIP file using WinZip or WinRaR.

Writing the Raspbian OS Image to the microSD Card

Insert the microSD card into the card reader. If your computer or laptop has a built-in card reader, insert it there. You might have to use a microSD-to-SD card adapter if the card reader or your computer has a slot only for the SD card reader.

Open Win32 Disk Imager. Select the location of the image file and click the Write button (see Figure 1-9).

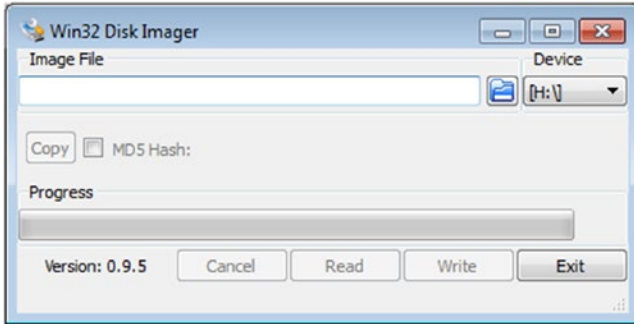


Figure 1-9. Win32 Disk Imager

If you see the warning message shown in Figure 1-10, toggle the write protection notch of the card reader or the SD card adapter (or both). Then click the Write button again.

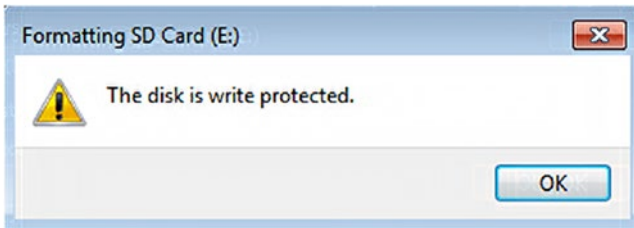


Figure 1-10. Write protection error message

Figure 1-11 shows the warning message that will be displayed. Click Yes to continue.

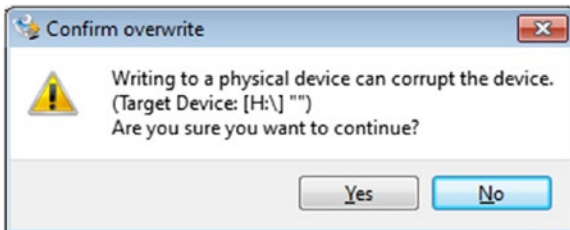


Figure 1-11. Overwrite warning message

Once the OS image has been written to the SD card, the message in Figure 1-12 will be displayed. Click OK.



Figure 1-12. *The Write Successful message*

This means the Raspbian OS has been flashed to the microSD card.

Altering the Contents of the config.txt File for a VGA Monitor

■ **Note** This step is a must if you are planning to use a VGA monitor. You should skip this step if you are using an HDMI monitor.

You need to change the contents of the config.txt file to get the Pi working with VGA monitors. You will learn more about config.txt later in this chapter.

Insert the microSD card into the card reader again and browse it in Windows Explorer. In Windows Explorer, it will be represented as a removable media drive called boot.

Open the config.txt file and make the following changes to it:

- Change `#disable_overscan=1` to `disable_overscan=1`
- Change `#hdmi_force_hotplug=1` to `hdmi_force_hotplug=1`
- Change `#hdmi_group=1` to `hdmi_group=2`
- Change `#hdmi_mode=1` to `hdmi_mode=16`
- Change `#hdmi_drive=2` to `hdmi_drive=2`
- Change `#config_hdmi_boost=4` to `config_hdmi_boost=4`

Save the file after making these changes. The microSD card is now ready for the Pi and for a VGA monitor.


Booting Up the Pi

Let's boot the Pi up with the prepared microSD card. The steps for that are as follows.

1. If you are using an HDMI monitor, connect the monitor directly to the Pi's HDMI port using the HDMI male-to-male cable. If you are using a VGA monitor, use the HDMI-to-VGA adapter to convert HDMI signals to VGA.
2. Insert the microSD card into the microSD card slot of the Pi.
3. Connect the USB mouse and the USB keyboard.
4. At this point, make sure that the power is switched off. Then connect the Pi to the power supply with a micro USB power cable discussed earlier.
5. Connect the monitor to the power supply.
6. Check all the connections. Switch on the power supply of the Pi and the monitor.

At this point, the Raspberry Pi will boot up.

For all the models of Raspberry Pi with the single core processor, the boot screen will resemble the screen in Figure 1-13.



```

devtmpfs: mounted
Freeing init memory: 132K
INIT: version 2.88 booting
[info] Using makefile-style concurrent boot in runlevel S.
[....] Starting the hotplug events dispatcher: udevdudevd[114]: starting version
175
. ok
[....] Synthesizing the initial hotplug events...udev: version magic '3.1.9+ pr
eempt mod_unload modversions ARMv6 ' should be '3.1.9+ mod_unload ARMv6 '
udev: version magic '3.1.9+ preempt mod_unload modversions ARMv6 ' should be '3
.1.9+ mod_unload ARMv6 '
ata_id[203]: HDIO_GET_IDENTITY failed for '/dev/sr0': Invalid argument
done.
[ ok ] Waiting for /dev to be fully populated...done.
Starting fake huclock: loading system time.
Tue Nov  4 16:17:01 UTC 2014
[ ok ] Setting preliminary keymap...done.
[ ok ] Setting parameters of disc: (none).
[ ok ] Activating swap...done.
EXT4-fs (sda2): re-mounted. Opts: (null)
[....] Loading kernel modules...snd_page_alloc: version magic '3.1.9+ preempt mo
d_unload modversions ARMv6 ' should be '3.1.9+ mod_unload ARMv6 '
done.
[....] Activating lvm and md swap...

```

Figure 1-13. Single-core CPU RPi model boot screen

For all the models of Raspberry Pi with the quad-core processor, the boot screen will resemble Figure 1-14.



```
[ ok ] Waiting for /dev to be fully populated...done.
Starting fake hwclock: loading system time.
Thu Feb 26 18:50:09 UTC 2015
[ ok ] Setting preliminary keymap...done.
[ ok ] Activating swap...done.
[ 6.121590] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[....] Checking root file system...fsck from util-linux 2.20.1
e2fsck 1.42.5 (29-Jul-2012)
/dev/mmcblk0p2: clean, 85289/196224 files, 648143/786640 blocks
done.
[ 6.365172] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ ok ] Cleaning up temporary files... /tmp.
[info] Loading kernel module snd-bcm2835.
[ ok ] Activating lun and md swap...done.
[....] Checking file systems...fsck from util-linux 2.20.1
dosfsck 3.0.13, 30 Jun 2012, FAT32, LFN
/dev/mmcblk0p1: 38 files, 1044/7161 clusters
done.
[ ok ] Mounting local filesystems...done.
[ ok ] Activating swapfile swap...done.
[ ok ] Cleaning up temporary files....
[ ok ] Setting kernel variables...done.
[....] Starting resize2fs, once/resize2fs 1.42.5 (29-Jul-2012)
Filesystem at /dev/root is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/root is now 3874176 blocks long.

update-rc.d: using dependency based boot sequencing
. ok
[ ok ] Configuring network interfaces...done.
[ ok ] Cleaning up temporary files....
[ ok ] Setting up ALSA...done.
[info] Setting console screen modes.
[info] Skipping font and keymap setup (handled by console-setup).
[ ok ] Setting up console font and keymap...done.
[....] Checking if shift key is held down:Error opening '/dev/input/event': No such file or directory
[ ok ] Switching to ondemand scaling governor.
[ ok ] Setting up X socket directories... /tmp/.X11-unix /tmp/.ICE-unix.
INIT: Entering runlevel: 2
[info] Using makefile-style concurrent boot in runlevel 2.
Error opening '/dev/input/event': No such file or directory
```

Figure 1-14. Quad-core CPU RPi model boot screen

Once the Pi boots up, the monitor displays the desktop, as shown in Figure 1-15.

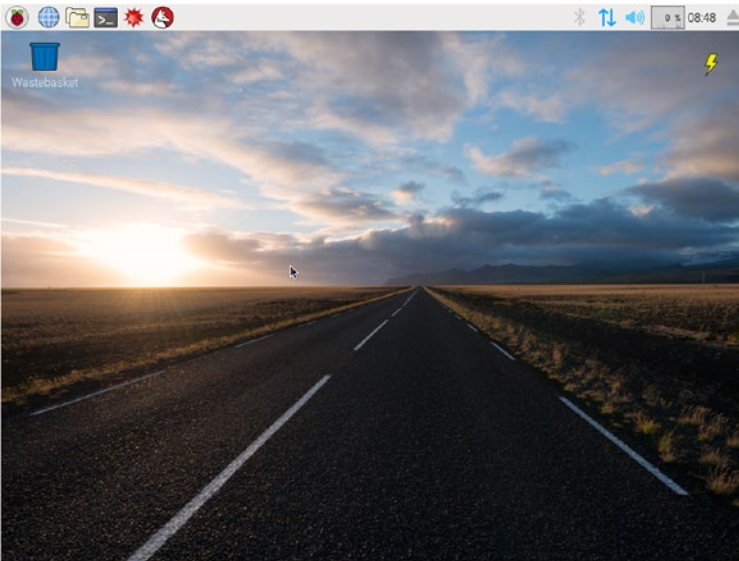


Figure 1-15. Raspbian desktop (as of February 2017)

Configuring the Pi

You need to configure the Pi for further use. Let's do that.

On the desktop, there is a taskbar. In the taskbar, you'll see the icon shown in Figure 1-16.



Figure 1-16. *LXTerminal icon*

Click the icon to open the LXTerminal window, as shown in Figure 1-17.



Figure 1-17. *The LXTerminal window*

The terminal is a desktop-independent VTE-based terminal emulator for LXDE without any unnecessary dependencies. Type `sudo raspi-config` in the prompt and press Enter. The `raspi-config` is the configuration tool for Raspberry Pi.

First expand the filesystem, as shown in Figure 1-18.

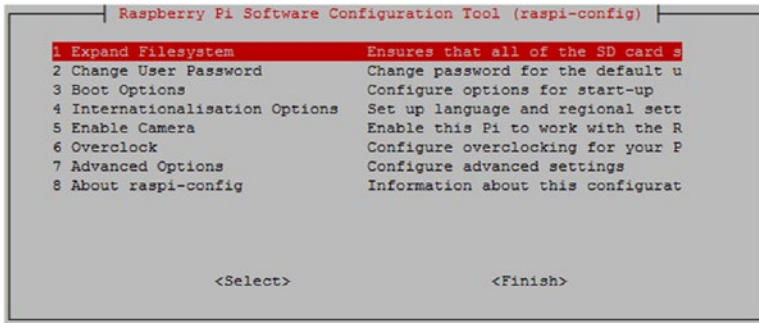


Figure 1-18. The *raspi-config* utility's main menu

Then navigate to the boot options, which are highlighted in Figure 1-19.

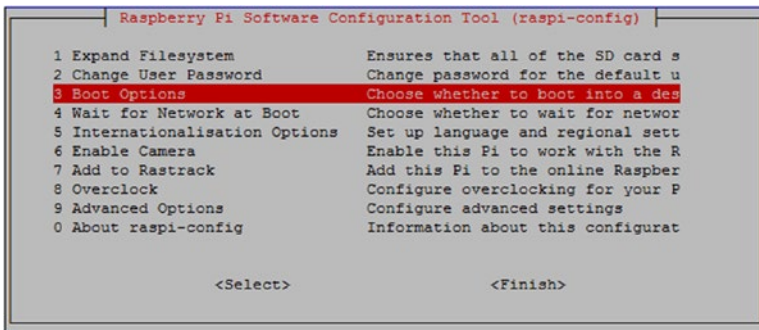


Figure 1-19. The *raspi-config* with the Boot Options highlighted

Set the Boot Options to Desktop Autologin, as shown in Figure 1-20.

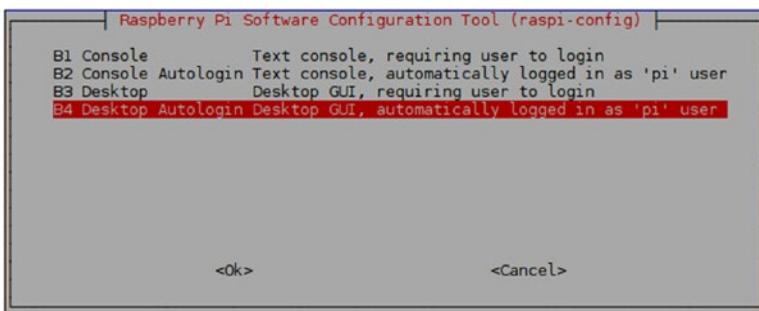


Figure 1-20. Desktop Autologin is highlighted

In the Internationalization Options section, change the time zone and the WiFi country (see Figure 1-21). Change the keyboard layout to US.

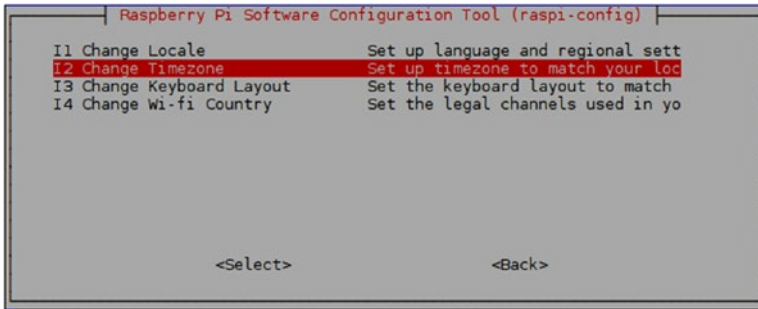


Figure 1-21. The *raspi-config* internationalization options

Once you're done, go back to the the main screen and click Finish, as shown in Figure 1-22.

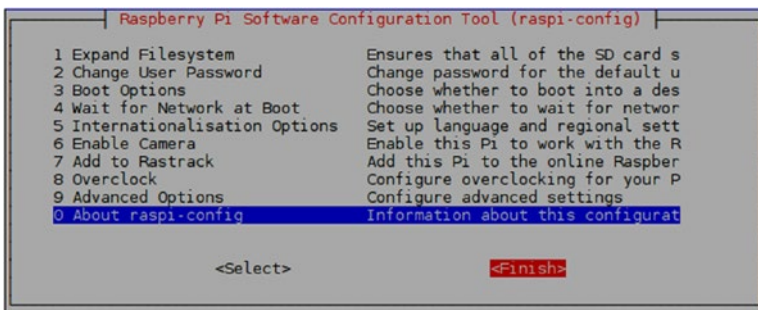


Figure 1-22. Finish

It will ask if you want to reboot at this point, as shown in Figure 1-23. Choose Yes.

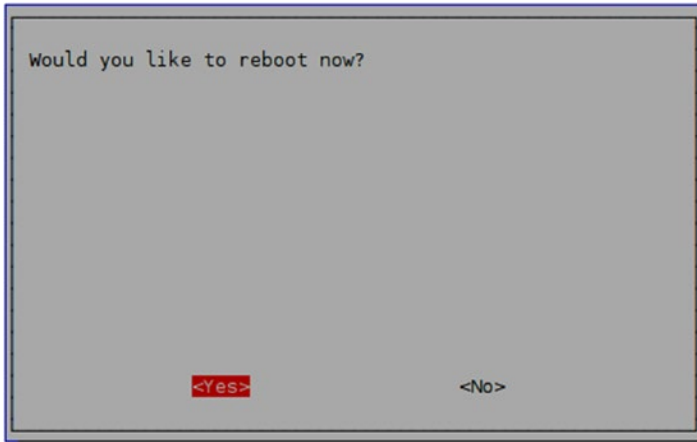


Figure 1-23. Reboot prompt

It will reboot the Pi.

Your job is not done yet. You need to learn how to connect the Pi to the Internet and how to update it.

The Raspbian OS

An operating system is the set of basic programs and utilities that make a computer work. It is an interface between the user and the computer. *Raspbian* is a free operating system based on the popular Linux distribution, *Debian*. Raspbian is optimized for the Raspberry Pi family of SBCs. It is even ported to the other similar SBCs like Banana Pro.

Raspbian has more than 35,000 packages and lots of pre-compiled software bundled for easy installation and use on the Raspberry Pi. The first build of Raspbian was completed in June of 2012. Raspbian is still under active development and updated very frequently. Visit the Raspbian home page at <https://www.raspbian.org> and the Raspbian documentation page at <https://www.raspbian.org/RaspbianDocumentation> for more information on Raspbian.

The config.txt File

Raspberry Pi does not have a conventional BIOS. The BIOS (basic input/output system) is the program that a computer's microprocessor uses to get the computer system started after it is turned on. It also manages data flow between the computer's operating system and attached peripheral devices such as the hard disk, video adapter, keyboard, mouse, and printer.

Since the Raspberry Pi does not have a BIOS, the various system configuration parameters that are normally stored and modified using the BIOS are instead stored in a text file called `config.txt`.

The Raspberry Pi `config.txt` file is on the boot partition of the Raspberry Pi. It is normally accessible as `/boot/config.txt` from Linux. However, from Windows and Mac OS, it is seen as a file in the accessible part of the microSD card. The accessible part of the card is labeled as `boot`. As you learned earlier in this chapter, you must edit the `/boot/config.txt` file if you want to connect it to a VGA display.

On Raspberry Pi, you can edit this file with the following command in the LXTerminal:

```
sudo nano /boot/config.txt
```

■ **Note** nano is a simple and easy-to-learn text-based text editor for Linux. Visit its home page at <https://www.nano-editor.org> to learn more about it. I find it easier to use than the vi or vim editors.

To learn more about `config.txt`, visit the page http://elinux.org/RPi_config. A sample configuration can also be found at http://elinux.org/R-Pi_configuration_file.

Connecting the Raspberry Pi to a Network and to the Internet

To connect the Pi to any network, you have to edit the `/etc/network/interfaces` file. If the network the Pi is connected to is connected to the Internet, the Pi can access the Internet as well.

WiFi

Raspberry Pi 3 Model B has built-in WiFi. For all the other models of Pi, you need to use a USB WiFi adapter.

Once the USB WiFi adapter is attached to the Pi, take a backup of the `/etc/network/interfaces` file using the following command:

```
sudo mv /etc/network/interfaces /etc/network/interfaces.bkp
```

The original `/etc/network/interfaces` file is safe this way, and it can be restored if something goes wrong.

Now create a new `/etc/network/interfaces` file as follows:

```
sudo nano /etc/network/interfaces
```

Type the lines from Listing 1-1 into that new file.

Listing 1-1. /etc/network/interfaces

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-ssid "ASHWIN"
wpa-psk "internet"
```

In Listing 1-1, replace ASHWIN with the SSID of your WiFi network and replace internet with the password of your WiFi network. Save the file by pressing Ctrl+X and then y.

Run the following command to restart the networking service:

```
sudo service networking restart
```

If you followed the steps correctly, the Pi should be connected to the WiFi network and to the Internet (provided that the WiFi network is connected to the Internet, of course).

To verify connectivity with the Internet, use the following command:

```
ping -c4 www.google.com
```

It should show output similar to this:

```
PING www.google.com (216.58.197.68) 56(84) bytes of data.
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=1 ttl=55
time=755 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=2 ttl=55
time=394 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=3 ttl=55
time=391 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=4 ttl=55
time=401 ms

--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 391.729/485.695/755.701/155.925 ms
```

This output indicates that the Pi is connected to the Internet.

To determine the IP address of Pi, use the `ifconfig` command. Check the output for `wlan0`. It will appear as follows:

```
wlan0    Link encap:Ethernet  HWaddr 7c:dd:90:00:e2:1e
        inet addr:192.168.0.122  Bcast:192.168.0.255  Mask:255.255.255.0
        inet6 addr: fe80::7edd:90ff:fe00:e21e/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1974 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1275 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:195049 (190.4 KiB)  TX bytes:1204336 (1.1 MiB)
```

In this output, `192.168.0.122` is IP address of the Pi. Because the IP address is allocated using the DHCP protocol, it will be different for you depending on your WiFi network settings.

Ethernet

You can also connect the Pi to a LAN network. Based on the LAN switch's settings, you can allocate an IP address to the Pi statically or dynamically.

Static IP Address

If the LAN network allocates IP addresses statically, configure the `/etc/network/interfaces` file as shown in Listing 1-2.

Listing 1-2. `/etc/network/interfaces`

source-directory `/etc/network/interfaces.d`

```
auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet static
# Your static IP
address 192.168.0.2
# Your gateway IP
gateway 192.168.0.1
netmask 255.255.255.0
# Your network address family
network 192.168.0.0
broadcast 192.168.0.255
```

In the file shown in Listing 1-2, the parameters `address`, `gateway`, `netmask`, `network`, and `broadcast` are based on the LAN's configuration. Check the manual of the LAN switch or router. If you are working for an organization, then check with the network administrator for these parameters.

Dynamic IP Address

This is an easy one. If the LAN has DHCP capability, configure the `/etc/network/interfaces` file as shown in Listing 1-3.

Listing 1-3. `/etc/network/interfaces`

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
```

This will configure the Pi to acquire the IP address automatically with DHCP.

■ **Note** All the information needed for network setup on Debian and its derivatives can be found at <https://wiki.debian.org/NetworkConfiguration>.

Updating the Pi

Pi must be connected to the Internet in order to update it successfully.

Updating the Firmware

To update the firmware, run `sudo rpi-update`. It will do the update for you.

Updating and Upgrading Raspbian

You will use APT for this. APT (Advanced Package Too) is a program that handles the installation and removal of software on the Debian and other Debian derivatives. APT simplifies the process of managing software on Debian systems by automating the fetching, configuration, and installation of software packages. You need an Internet connection for this too.

First, update the system's package list by entering the following command in the LXTerminal:

```
sudo apt-get update
```

`apt-get update` downloads the package lists from the respective remote repositories and updates them in the local computer so that information on the newest versions of packages and their dependencies is available for the installation and update. It should be run before running the `install` or `upgrade` command.

Next, upgrade all the installed packages to their latest versions using this command:

```
sudo apt-get dist-upgrade -y
```

`apt-get dist-upgrade` fetches new versions of the packages on the local machine that are marked for upgrade. It also detects and installs any dependencies. It might also remove obsolete packages.

Doing this regularly will keep your Raspbian OS up to date. After entering these commands, it will take a while to update the OS, because these commands fetch the data and the packages from remote repositories.

■ **Note** `sudo apt-get --help` will list all the options associated with `apt-get`.

Updating raspi-config

In `raspi-config`, go to the advanced options (see Figure 1-24) and choose Update.

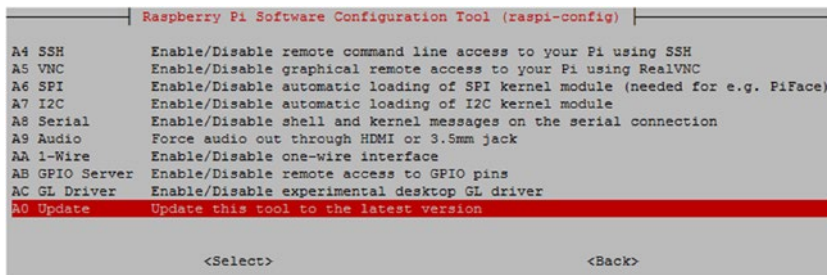


Figure 1-24. Updating `raspi-config`

Shutting Down and Restarting Pi

You can shut down Pi safely using the `sudo shutdown -h now` command. You can restart Pi using the `sudo reboot -h now` command.

Conclusion

This chapter introduced the concept and philosophy of SBCs. You also learned about a popular family of SBCs, Raspberry Pi. Now you can confidently move ahead with further exploration. In the next chapter, you will learn a few things about Python and digital image processing.

CHAPTER 2



Introduction to Python and Digital Image Processing

In the last chapter, we explored the amazing world of single board computers and Raspberry Pi. We booted up the Raspberry Pi, connected it to the Internet, and updated the Raspbian OS.

In this chapter, we will get started with Python and the concepts of digital image processing (DIP).

Let's begin this chapter with an introduction to Python. I personally find Python amazing and have been enchanted by it. Python is a simple yet powerful programming language. When programmers use Python, it's easy to focus on solving a given problem as they do not have to worry about the syntax. Python perfectly fits the philosophy of Raspberry Pi, which is programming for everyone. That's why it's the most preferred programming platform for Raspberry Pi and many other SBCs.

A History of Python

Python was designed and conceived in the late 1980s. Its actual implementation was started in late 1989 by Guido van Rossum in Centrum Wiskunde & Informatica (National Research Institute for Mathematics and Computer Science) in the Netherlands. Python is a successor to the ABC Programming Language, which itself was inspired by SETL. In February of 1991, Van Rossum publically published the Python source code to the alt.sources newsgroup. The name Python was inspired by the British television show *Monty Python's Flying Circus*. Van Rossum is a big fan of Monty Python.

Van Rossum is the principal author of the Python programming language. He plays a central role in guiding the direction of the development, enhancement, and further evolution of Python programming language. He holds the title *Benevolent Dictator for Life* for Python. He currently (as of February 2017) works for Dropbox and dedicates almost half of his time toward further development of the Python programming language.

The central philosophy of the Python programming language (the Zen of Python) is explained in PEP-20 (PEP stands for Python Enhancement Proposal), which can be found at <https://www.python.org/dev/peps/pep-0020>.

It is a collection of 20 software principles, out of which 19 have been documented. The principles are as follows:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

Features of Python

The following are the features of Python that have made it popular and beloved in the programming community.

Simple

Python is a simple language with a minimalist approach. Reading a well written and good Python program makes you think you are reading English text.

Easy to Learn

Due to its simple and English-like syntax, Python is extremely easy to learn. That is the prime reason that it is taught as the first programming language to high school and university students who take introductory programming courses. An entire new generation of programmers is learning Python as their first programming language.

Easy to Read

Unlike other high-level programming languages, Python does not obfuscate the code and make it unreadable. The English-like structure of the Python code makes it easier to read compared to the code written in other programming languages. This makes it easier to understand and easier to learn compared to other high-level languages like C and C++.

Easy to Maintain

As Python code is easy to read, easy to understand, and easy to learn, anyone maintaining the code becomes comfortable with the codebase very quickly. I can vouch for this from personal experiences of maintaining and enhancing large legacy codebases which were written in a combination of Bash and Python 2.

Open Source

Python is an open source project, which means its source code is freely available. You can make changes to it to suit your needs and use the original and modified code in your applications.

High-Level Language

While writing Python programs, you do not have to manage the low-level details like memory management, CPU timings, and scheduling processes. All these tasks are managed by the Python interpreter. You can write the code directly in the easy-to-understand English-like syntax.

Portable

The Python interpreter has been ported to many OS platforms. Python code is also portable. All the Python programs will work on the supported platform without requiring many changes if you are careful enough to avoid system-dependent coding.

You can use Python on GNU/Linux, Windows, Android, FreeBSD, Mac OS, iOS, Solaris, OS/2, Amiga, Palm OS, QNX, VMS, AROS, AS/400, BeOS, OS/390, z/OS, Psion, Acorn, PlayStation, Sharp Zaurus, RISC OS, VxWorks, Windows CE, and PocketPC.

Interpreted

Python is an interpreted language. Let's take a look at what that means. Programs written in high-level programming languages like C, C++, and Java are compiled first. This means that they are first converted into an intermediate format. When we run the program, this intermediate format is loaded from secondary storage (i.e., from the hard disk) to the memory (RAM) by the linker/loader.

So, C, C++, and Java have a separate compiler and linker/loader. This is not the case with Python. Python runs the program directly from the source code. You do not have to bother compiling and linking to the proper libraries. This makes Python programs truly portable, as you can copy the program to one computer from another and the program runs fine as long as the necessary libraries are installed on the target computer.

Object-Oriented

Python supports procedure-oriented programming as well as object-oriented programming paradigms.

All the object-oriented programming paradigms are implemented in Python. In the object-oriented programming languages, the program is built around objects that combine data and the related functionality. Python is a very simple but powerful object-oriented programming language.

Extensible

One of the features of Python is that you can call C and C++ routines from the Python programs. If you want the core functionality of the application to run faster, you can code that part in C/C++ and call it in the Python program (C/C++ programs generally run faster than Python).

Extensive Libraries

Python has an extensive standard library that comes pre-installed. The standard library has all the essential features for a modern day programming language. It has provisions for databases, unit testing (we will explore this later in this book), regular expressions, multi-threading, network programming, computer graphics, image processing, GUI, and other utilities. This is the part of Python's batteries-included philosophy.

Apart from the standard library, Python has numerous and ever-growing sets of third-party libraries. The list of these libraries can be found on the Python Package Index.

Robust

Python provides robustness by means of the ability to handle errors. The full stack trace of the encountered errors is available and makes the life of the programmer more bearable. The runtime errors are known as exceptions. The feature that handles these errors is known as an exception handling mechanism.

Rapid Prototyping

Python is used as a rapid prototyping tool. As you learned earlier, Python has extensive libraries and is easy to learn, which has led many software architects to use it as a tool to rapidly prototype their ideas into working models quickly.

Memory Management

In assembly language and in programming languages like C and C++, memory management is the responsibility of the programmer. This is in addition to the task at hand. This creates an unnecessary burden on the programmer. In Python, the Python interpreter takes care of the memory management. This helps the programmers steer clear of memory issues and focus on the task at hand.

Powerful

Python has everything in it that a modern programming language needs. It is used in applications such as computer visioning, supercomputing, drug discovery, scientific computing, simulation, and bioinformatics. Millions of programmers around the world use Python. Many big organizations like NASA, Google, SpaceX, and Cisco use Python for their applications and infrastructure.

Community Support

I find this to be the most appealing feature of Python. Recall that Python is open source. It also has community of almost a million programmers throughout the world (probably more, as today high school kids are learning Python too). There are also plenty of forums on the Internet to support programmers who encounter a roadblock. None of my queries related to Python have ever gone unanswered.

Python 3

Python 3 was released in 2008. The Python development team decided to do away with some of the redundant features of Python, simplify some more features, rectify some design flaws, and add a few much needed features.

It was decided that a major revision number was needed for this and the resultant release would not be backward compatible. Python 2.x and 3.x were supposed to coexist in parallel for the programmer community to have enough time to migrate their code and the third-party libraries from 2.x to 3.x. Python 2.x code cannot be run as-is in most cases, as there are significant differences between 2.x and 3.x.

The Differences Between Python 2 and Python 3

The following are few of the most noticeable differences between Python 2 and Python 3. We will be using many features of Python 3 related to these differences. Let's look at them in brief:

- *The print() function:* This is the most noticeable difference between Python 2 and Python 3. The `print` statement of Python 2 is replaced with the `print()` function in Python 3.
- *Integer division produces a float value:* The nature of integer division has been changed in Python 3 for the sake of mathematical correctness. In Python 2, the result of the division of two integers was an integer. However, in Python 3, it is a float value, which is mathematically correct and makes more sense to beginners. In most programming languages, the integer division is an integer.
- *Removal of xrange():* In Python 2, the `xrange()` function was used to create iterable objects. In Python 3, `range()` is implemented like `xrange()`. So, a separate `xrange()` is not required anymore in Python 3. Using `xrange()` in Python 3 raises a `NameError` exception.
- *Raising exceptions:* It is mandatory in Python 3 to enclose exception arguments, if any, in parentheses, whereas in Python 2 it is optional.
- *Handling exceptions:* In Python 3, while handling the exceptions, you must use the `as` keyword before the parameter to handle the argument. In Python 2, it is not needed.
- *New style classes:* Python 2 supports both the old style classes and new style classes, whereas Python 3 supports only the new style classes. All the classes created in Python 3 use the new style by default.

These exclusive new features of Python 3 have not yet been backported to Python 2:

- Strings are Unicode by default
- Clean Unicode/byte separation
- Exception chaining
- Function annotations
- Syntax for keyword-only arguments
- Extended tuple unpacking
- Non-local variable declarations

From this list, we will be extensively incorporating the `print()` method, new-style classes, exceptions, and exception handling in the code examples in this book.

■ **Note** Check out Python's Wiki page for differences between Python 2 and Python 3: <https://wiki.python.org/moin/Python2orPython3>.

Why Use Python 3

We will be frequently using new style classes and exceptions in the code examples in this book. Although many Python experts still advocate using Python 2, I disagree with them. Python's wiki page (see <https://wiki.python.org/moin/Python2orPython3>) says:

Python 2.x is legacy, Python 3.x is the present and future of the language.

One of the major arguments in favor of Python 2 is its extensive documentation, books, and third-party libraries. However, most of the developers are porting their custom libraries to Python 3. Almost all the major third-party libraries are ported and fully supported for Python 3. As far as books and documentation is concerned, authors like me are extensively writing for Python 3. As time passes, more documentation for Python 3 will surely be available.

A new generation of programmers is being introduced to Python 3 as the first programming language. When they are comfortable with the concept and Philosophy of Python programming, they are gradually introduced to Python 2.

Many organizations have already started migrating codebases from Python 2 to Python 3. Almost all new projects in Python are using Python 3.

I personally think that these are pretty good reasons to use Python 3.

Python 2 and Python 3 on Raspbian

Raspbian is a Debian variant. Python 2 and Python 3 interpreters are pre-installed in Raspbian. The Python 2 interpreter can be invoked by running the `python` command in the `lxterminal`. The Python 3 interpreter can be invoked by running the `python3` command in the `lxterminal`. You can check the Python 3 interpreter version by running `python3 -V` or `python --version`. You can check the location of the Python 3 binary by running `which python3` at the `lxterminal`.

Running a Python Program and Python Modes

You have set up your environment for Python programming, so let's get started with a simple concept of Python. Python has two modes—normal and interactive. Let's look at these modes in detail.

Interactive Mode

Python's interactive mode is a command-line shell. It provides immediate output for the every executed statement. It also stores the output of previously executed statements in the active memory. As new statements are executed by the Python interpreter, the entire sequence of previously executed statements is considered while evaluating the current output. You have to type `python3` in the `lxterminal` to invoke the Python 3 interpreter into interactive mode as follows:

```
pi@raspberrypi:~ $
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can execute Python statements directly in this interactive mode just like when you run commands from the OS shell/console as follows:

```
>>>print ('Hello World!')
Hello World!
>>>
```

We will not be using interactive mode in this book. However, keep it in mind, as it's the quickest way to check small snippets of code (5 to 10 lines). You can quit the interactive mode using the `exit()` statement as follows:

```
>>> exit()
pi@raspberrypi:~ $
```

Normal Mode

Normal mode is where the Python script files (`.py`) are executed by the Python interpreter.

Create a file called `test.py` and add the `print ('Hello World!')` statement to the file. Save the file and run it with the Python 3 interpreter as follows.

```
pi@raspberrypi:~ $ python3 test.py
HelloWorld!
pi@raspberrypi:~ $
```

In this example, `python3` is the interpreter and `test.py` is the filename. In case the Python `test.py` file is not in the same directory where you're invoking the `python3` interpreter, you have to provide the absolute path of the Python file.

IDEs for Python

An Integrated Development Environment (IDE) is a software suite that has all the basic tools to write and test programs. A typical IDE has a compiler, a debugger, a code editor, and a build automation tool. Most of the programming languages have various IDEs to make programmers' lives better. Python too has many IDEs. Let's look at a few of Python's IDEs.

IDLE

IDLE stands for Integrated DeveLopment Environment. It comes bundled with Python. IDLE3 is for Python 3. It's popular with Python beginners. Just run `idle3` in the `lxterminal`. Figure 2-1 shows the IDLE3 code editor and an interactive prompt.

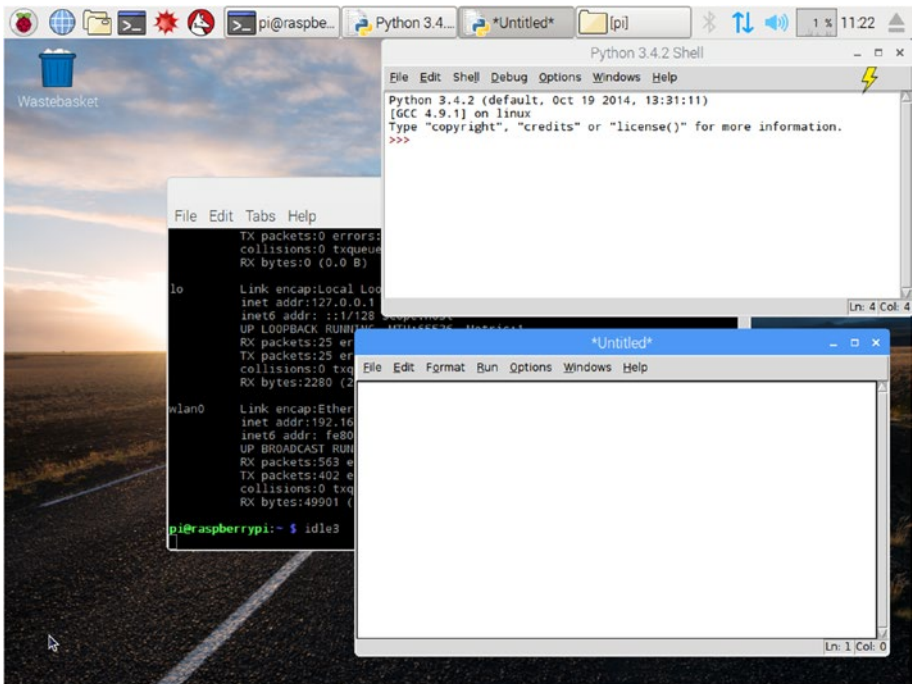


Figure 2-1. IDLE3

Geany

Geany is a text editor using the GTK+ toolkit with basic features of an integrated development environment. It supports many file types and has some nice features. Check out <https://www.geany.org> for more details. Figure 2-2 shows the Geany text editor.

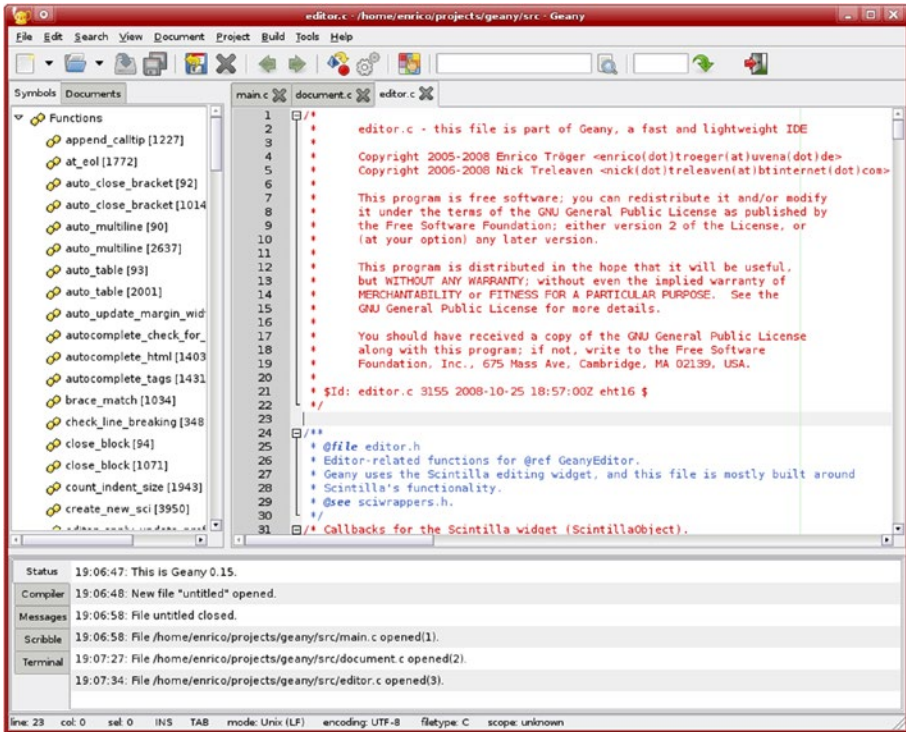


Figure 2-2. Geany

Geany is pre-installed in the latest versions of Raspbian. If your Raspbian installation does not have Geany, you can install it by running `sudo apt-get install geany` in the lterminal. Once it's installed, it can be found using the Raspbian menu by choosing Programming ► Geany Programmer's Editor, as shown in Figure 2-3.

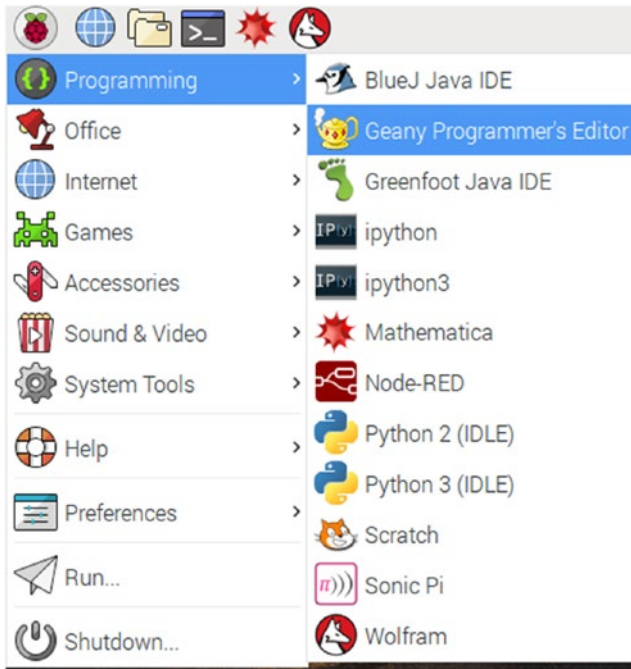


Figure 2-3. The Raspbian menu

Type `print("Hello World!")` in the code editor and save the file in the `/home/pi` directory as `test.py`. Click Build in the menu bar and then choose Execute. You can also use the F5 keyboard shortcut to execute the program. The program will execute in an lterminal window. You have to press Enter to close the execution window. The default Python interpreter for Geany is Python 2. You'll need to change it to Python 3. To do that, go to Build ► Set Build Commands. The window in Figure 2-4 will appear.

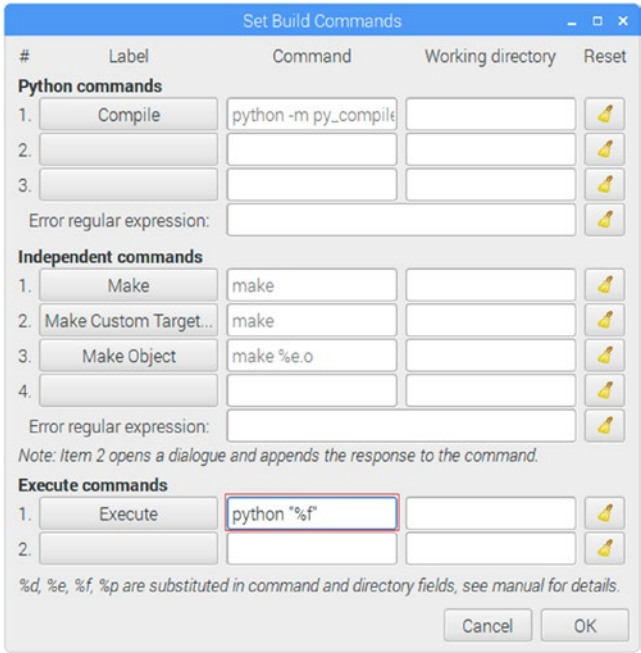


Figure 2-4. The Set Build Commands window

In the Set Build Commands window, within the Execute Commands section, change python "%f" (highlighted in the box in Figure 2-4) to python3 "%f" to set the Python 3 interpreter as the default interpreter. After that, run the program again to verify that everything works correctly.

Introduction to Digital Image Processing

To get started with digital image processing (DIP), we'll briefly review your understanding of a few basic concepts about the related topics first.

Signal Processing

Anything that carries any information, when represented mathematically, is called a *signal*. The process or technique used to extract useful and relevant information from a given signal is known as *signal processing*. The system that does this type of task is known as a *signal processing system*. The best example of a signal processing system is the human brain. It processes various types of signals from various senses. The human brain is a biological signal processing system. When the system is made up of electronic components, it is known as an electronic signal processing system. Signal processing is a discipline that combines mathematics and electrical engineering.

There are two types of electronic signals—analog and digital. The following table lists the differences between these two types of signals.

Analog	Digital
A continuous signal	Discrete in nature
Denoted by sinusoidal waves	Denoted by square waves
A continuous signal	A discrete signal
Deteriorated due to noise	Not affected by noise
Not flexible	Are flexible and can perform a variety of operations
Consume less bandwidth	Consume a lot of bandwidth
Draw a lot of power	Draw less power compared to analog
There are lots of errors in capturing, storing, and transmitting analog signals	Fewer errors compared to analog signals, and there are checksum and error correction algorithms available for digital signals.

Image Processing

An image is a signal. So, *image processing* is a type of signal processing. *Image processing systems* are types of signal processing systems. The combination of the eye and brain is an example of a biological image processing system. There are two types of image processing systems—analog and digital.

Analog Image Processing

The days of still and motion film cameras represent the analog era. The sources of analog images are film cameras (still and motion), older fax machines, and telex machines. Older television systems, CRT monitors, and film projectors represent analog image processing systems. Analog image processing involves using analog electronic systems, mechanical parts, optics, and chemistry (to develop and store films) extensively.

Digital Image Processing

With the advent of digital computers, storage systems, image sensors, and digital cameras, images can be captured, stored, and processed in digital formats. Using digital computers to process and retrieve information from an image (analog and digital images both) is known as *digital image processing*. It involves extensive usage of digital sensors (digital cameras), digital computers, and digital storage devices. Here are some applications of digital image processing systems:

- Computerized image editing, correction, enhancement, denoising, etc.
- Medical image processing and diagnostics assistance

- Space image processing (processing images from Hubble and ground-based telescopes)
- Industrial applications like product inspection and sorting
- Biometrics (finger, face, and iris recognition)
- Film-making and visual effects
- Remote sensing (processing images from aerial and satellite sources)

The following scientific disciplines significantly overlap with digital image processing:

- Signal processing
- Digital electronics
- Computer/machine vision
- Biological vision
- Artificial intelligence, pattern recognition, and machine learning
- Robotics and robot vision

Using Raspberry Pi and Python for Digital Image Processing (DIP)

Since DIP requires digital computers, you need to use a computer and an associated programming platform to implement digital image processing. Raspberry Pi fulfills all the requirements of a minimal power system required for DIP. The most appealing feature of Raspberry Pi is its low cost. Also, it can be interfaced with a variety of digital image sensors like Webcams and the Pi Camera module.

You learned that Python is an easy-to-learn programming language and helps you focus on the task at hand rather than having to worry about syntax. It is also the most preferred programming platform for Raspberry Pi. Many third-party image processing and visualization libraries are available for Python. This makes Raspberry Pi and Python the most obvious choice for beginners to get started with DIP.

EXERCISE

Complete the following exercise to better understand the Python 3 background.

- Visit and explore the Python home page at www.python.org.
- Visit and explore the Python documentation page at <https://docs.python.org/3/>.
- Check for new features of the latest releases of Python at <https://docs.python.org/3/whatsnew/index.html>.

Conclusion

In this chapter, you learned about the background, history, and features of Python. You also studied the important differences between Python 2.x and Python 3.x. You learned to use Python 3 in scripting and interpreter modes. You looked at few popular IDEs for Python and configured Geany for Python 3 on the Pi. Finally, you learned about the field of digital image processing. In the next chapter, you will get started with a popular digital image processing library in Python, called Pillow. You will also learn how to use the Tkinter library to show images.

CHAPTER 3



Getting Started

In the last chapter, you learned the philosophy of Python. You also learned why you should use Python 3 and learned the concepts related to digital image processing. This chapter starts image processing programming with Python 3 on the Raspberry Pi. You will learn how to connect a Raspberry Pi to various camera sensors to acquire images. You will be introduced to the pillow, Tkinter, and matplotlib libraries in Python 3.

Image Sources

To learn digital image processing, you are going to need digital images. There are standard image datasets used all around the world and we will use two of them. They are available in compressed and downloadable formats on the Internet:

http://imageprocessingplace.com/root_files_V3/image_databases.htm

<http://sipi.usc.edu/database/>

In this chapter, you will start programming. You need to organize all the files in a single directory, with chapter sub-directories to save the code. Open the Lxterminal and run the following commands to create a directory structure for the book code and image datasets:

```
mkdir DIP
cd DIP
mkdir code
mkdir dataset
cd code
mkdir chapter03
```

Extract all the images into the Dataset directory. Now the directory structure will look like Figure 3-1.


```
pi@raspberrypi:~/DIP $ tree
.
├── code
│   └── chapter03
└── Dataset
    ├── 4.1.01.tiff
    ├── 4.1.02.tiff
    ├── 4.1.03.tiff
    ├── 4.1.04.tiff
    ├── 4.1.05.tiff
    ├── 4.1.06.tiff
    ├── 4.1.07.tiff
    ├── 4.1.08.tiff
    ├── 4.2.01.tiff
    ├── 4.2.02.tiff
    ├── 4.2.03.tiff
    ├── 4.2.04.tiff
    ├── 4.2.05.tiff
    ├── 4.2.06.tiff
    ├── 4.2.07.tiff
    └── 5.1.09.tiff
```

Figure 3-1. Directory structure for the book

We used the `tree` command in the DIP directory to view the directory structure.

The images we downloaded and extracted are used as standard images for all the digital image processing and computer vision research all around the world.

Using the Webcam

Let's see how to capture images using a standard USB Webcam. Raspberry Pi 3 has four USB ports. You can use one of those to connect a Webcam to the Raspberry Pi. I am using a Logitech HD c310 USB Webcam (see Figure 3-2).



Figure 3-2. Logitech HD c310 USB Webcam

■ **Note** Before purchasing a Webcam, check the Webcam’s compatibility with Pi at http://elinux.org/RPi_USB_Webcams.

Attach the Webcam and run the `lsusb` command in the terminal. It displays a the list of all the USB devices connected to the computer. The output will be similar to the following:

```
Bus 001 Device 004: ID 046d:081b Logitech, Inc. Webcam C310
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514
Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

If you can see the USB Webcam in the output, it means that the Webcam is detected. There are quite a few Linux utilities to capture images using a Webcam. If you like GUI, `gview` is one of them. Use the following command to install it if it is not already installed:

```
sudo apt-get install guvcview
```

Once `guvcview` is installed, it can be accessed from the menu or from the terminal using the `guvcview` command.

The other useful utility is `fswebcam`. It's a command-line utility. Install it using the following command:

```
sudo apt-get install fswebcam
```

You can invoke the `fswebcam` utility from the terminal to capture an image with the Webcam as follows:

```
fswebcam -r 1280x960 --no-banner test.jpg
```

This will capture an image with the resolution of 1280x960 pixels. The `--no-banner` flag is for disabling the timestamp banner from the image. The image is saved as `test.jpg`. The output of the command is as follows:

```
--- Opening /dev/video0...  
Trying source module v4l2...  
/dev/video0 opened.  
No input was specified, using the first.  
--- Capturing frame...  
Captured frame in 0.00 seconds.  
--- Processing captured image...  
Disabling banner.  
Writing JPEG image to 'test.jpg'.
```

Then, check in the current directory for the image file called `test.jpg`.

The Pi Camera Module

Raspberry Pi foundation has designed dedicated Camera modules for the Raspberry Pi. There are two versions of them and they both come in normal and NoIR (No Infrared) varieties. The original versions are 5-megapixel Camera modules. The new versions are 8-megapixel Camera modules. Figure 3-3 shows an image of the new versions of the Camera modules.



Figure 3-4. Attaching the Pi Camera module to the Pi

The Pi Camera module connects directly to the GPU. As it is attached to the GPU, there is only a small impact on the CPU, leaving it available for other processing. The difference between the Pi Camera and USB Webcam is that the Pi Camera module has a higher performance and higher frame rate with h.264 video encoding.

The command-line utility you use to capture images using the Camera module is `raspistill`. You call it as follows:

```
raspistill -o test.png
```

`raspistill` does not write anything to the console or terminal, so you need to check the directory to see if `test.png` has been created or updated.

This is how you capture images with various cameras for the Pi for use in exploring the world of DIP with Pi.

Using Python 3 for Digital Image Processing

Python 3 does not have any pre-installed libraries for image processing. The Python Imaging Library (PIL) is one of the most popular beginner-friendly image processing libraries used for image processing in Python.

This library provides extensive file format support and reasonably powerful image processing capabilities. The core Image library is designed for faster access to data stored in a few basic image formats. The drawback with PIL is that after its last stable release 1.1.7 in 2009 and the last commit in 2011, there have not been any new releases. Also, PIL does not support Python 3. Officially, we have not heard that the PIL is dead. However, there is a friendly fork project that's an unofficial replacement and enhancement for PIL. It works well with Python 3 and is called Pillow. It is mostly backward compatible with PIL. You can get more information about Pillow on its home page at <https://python-pillow.org/>. Let's get started with the basics of Pillow.

You can install Pillow by running the following command at the terminal:

```
sudo pip3 install pillow
```

This will install Pillow for Python 3.

To check if it is installed properly, open Python 3 in interpreter mode and run the following sequence of commands. If Pillow is installed properly, these commands will display its version number.

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from PIL import Image
>>> print(Image.VERSION)
1.1.7
>>>
```

Working with Images

Let's work with images now. Before you begin, you have to install an image viewing utility called `xv` in order for the built-in function `show()` to work. The problem is that the `xv` utility is deprecated. So, you will use another utility, called `xli`, and point it to `xv` with a Linux command. Run the following commands:

```
sudo apt-get install xli -y
cd /usr/local/bin
sudo ln -s /usr/bin/xli xv
```

Save the program shown in Listing 3-1 as `prog01.py` in the `/home/pi/DIP/code/chapter03` directory.

Listing 3-1. `prog01.py`

```
from PIL import Image

im = Image.open("/home/pi/DIP/Dataset/4.2.04.tiff")
im.show()
```

Run this code with the following command:

```
python3 prog01.py
```

It will show the image in an xli window. This is the simplest Pillow program and it loads an image in a Python variable with `open()` and displays it with `show()`. In the first line, we're importing the Image module of Pillow. You will learn more about this module in the next chapter. The standard version of `show()` is not very efficient, since it saves the image to a temporary file and calls the xv utility to display the image. It's handy for the purposes of debugging.

Due to the limitation of the `show()` function, we will use Python's built-in GUI module called Tkinter for displaying images whenever needed, as demonstrated in Listing 3-2. In the later part of the book, you will become familiar with the matplotlib library for displaying images.

Listing 3-2. prog02.py

```
from PIL import Image, ImageTk
import tkinter as tk

im = Image.open("/home/pi/DIP/Dataset/4.2.04.tiff")

root = tk.Tk()
root.title("Test")

photo = ImageTk.PhotoImage(im)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

root.mainloop()
```

This program uses Python's built-in module for GUI, called Tkinter. We're importing it in the second line. The ImageTk module provides the functionality to convert the Pillow image to an Tk-compatible image with the `PhotoImage()` function. We're creating a Tk window with the statement `root = tk.Tk()`. The code displays the Pillow image as a label. The `title()` method sets the title of the image. In the code, the `Label()` and `pack()` functions are used to create the image label. The line `l.photo() = photo` is for Python's garbage collector. `root.mainloop()` is the main loop for the Tk GUI.

Run the program using this code:

```
python3 prog02.py
```

■ **Note** If you are encountering an error related to importing while executing this program, run the following command at the console:

```
sudo apt-get install python3-pil.imagetk
```

The output is shown in Figure 3-5.

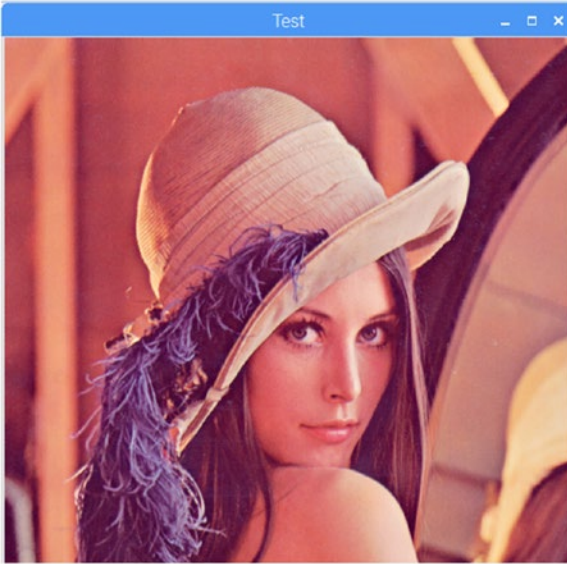


Figure 3-5. Sample Tkinter output

Image Properties

Let's take a look at the image properties using Listing 3-3.

Listing 3-3. prog03.py

```
from PIL import Image

im = Image.open("/home/pi/DIP/Dataset/4.2.04.tiff")
print(im.mode)
print(im.format)
print(im.size)
print(im.info)
print(im.getbands())
```


The following is the output of this program:

```
RGB
TIFF
(512, 512)
{'compression': 'raw', 'resolution': (1.0, 1.0)}
('R', 'G', 'B')
```

Let's look at the program line by line.

The mode of an image defines the type and depth of the pixels in an image. Here are the standard modes available in Pillow:

- 1 (1-bit pixels, black and white, stored with one pixel per byte)
- L (8-bit pixels, black and white)
- P (8-bit pixels, mapped to any other mode using a color palette)
- RGB (3x8-bit pixels, true color)
- RGBA (4x8-bit pixels, true color with transparency mask)
- CMYK (4x8-bit pixels, color separation)
- YCbCr (3x8-bit pixels, color video format)

The format of the image refers to the file format. size refers to the resolution of the image in the pixels. info refers to the auxiliary information of the image. The `getbands()` function retrieves the bands of the colors in the image.

■ **Note** You can find more information about Pillow at:

pillow.readthedocs.io

Conclusion

In this chapter, you got started with the basics of Pillow. You learned how to load and display an image. You learned a bit about image properties. You also learned how to capture images using various sensors. In the next chapter, you'll explore the Image, ImageChops, and ImageOps modules in detail.

CHAPTER 4



Basic Operations on Images

In the last chapter, you started with image processing using Pillow. You also used Tkinter for displaying images. In this chapter, you will learn various arithmetic and logical operations on images. You will explore Image, ImageChops, and ImageOps modules in Pillow for implementing arithmetic and logical operations. You will also learn how to use the slide bar in Tkinter to dynamically change the input to the Pillow methods.

Image Module

In the last chapter, you used the `open()` and `show()` functions in the Image module of Pillow. In this chapter, you will explore the module in detail. You will study and implement all the methods of this module that are used for basic operations on images.

Splitting and Merging Image Channels

In the grayscale images, there is only one image channel. This means that the grayscale image is made of only a single, two-dimensional matrix that lists the grayscale intensity of the corresponding pixels (values range from 0 to 255). For RGB images, there are three image channels—Red, Green, and Blue. You can see the channel intensities separately by splitting the images into constituent channels using the `split()` method. You can also merge separate channels into a single image with the `merge()` method. Python code that demonstrates the `split()` and `merge()` methods is shown in Listing 4-1.

Listing 4-1. prog01.py

```
from PIL import Image, ImageTk
import tkinter as tk

im = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

root = tk.Tk()
root.title("RED Channel Demo")

r, g, b = im.split()

photo1 = ImageTk.PhotoImage(r)
l1 = tk.Label(root, image=photo1)
```

```

l1.pack()
l1.photo = photo1

photo2 = ImageTk.PhotoImage(Image.merge("RGB", (r, g, b)))
l2 = tk.Label(root, image=photo2)
l2.pack()
l2.photo = photo2

root.mainloop()

```

The code in Listing 4-1 splits the image into separate channels. Then, it displays the Red channel of the image. It also displays the original image by merging previously split channels. The single channel is a matrix of intensity values. So, it is displayed as a grayscale image on the screen. The colors manifest themselves when the code combines all the channels into a single image.

The output window is shown in Figure 4-1.

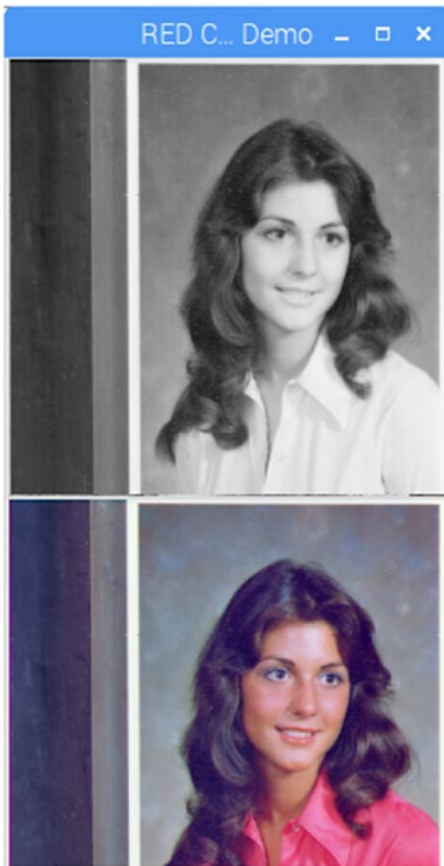


Figure 4-1. Red channel and the original image

Image Mode Conversion

You can change the mode of an image using the `convert()` method, as shown in Listing 4-2.

Listing 4-2. prog02.py

```
from PIL import Image, ImageTk
import tkinter as tk

im1 = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")
res1 = im1.convert("L")

root = tk.Tk()
root.title("Colorspace Conversion Demo")

photo = ImageTk.PhotoImage(res1)
l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

root.mainloop()
```

The code in Listing 4-2 changes the mode of the image to L. The `convert()` method supports all possible conversions between the RGB, CMYK, and L modes.

Image Blending

You can blend two images using the `blend()` method. It takes three arguments—the two images to be blended and value of alpha. The mathematical formula it uses for blending is as follows:

$$\text{output} = \text{image1} * (1.0 - \text{alpha}) + \text{image2} * \text{alpha}$$

Now, you will write a program that can change the value of alpha dynamically so that you can experience the blending effect. For that, you will use the scale widget in Tkinter.

The program in Listing 4-3 demonstrates this process.

Listing 4-3. prog03.py

```
from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(alpha):
    print('Alpha: ', alpha)

    img = Image.blend(im1, im2, float(alpha))
    photo = ImageTk.PhotoImage(img)
```

```

l['image'] = photo
l.photo = photo

root = tk.Tk()
root.title('Blending Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")
im2 = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Alpha", from_=0, to=1,
               resolution=0.01, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()

```

The code in Listing 4-3 creates a scale widget using the `tk.Scale()` statement. The statement has more than 79 characters, so in order to conform to the PEP8 standard, I wrote it to fit in two lines, each consisting less than 79 characters. The parameters passed to `tk.Scale()` are as follows:

- The Tkinter window variable name
- `label`: The label to be associated with scale
- `from_` and `to`: The range of values
- `resolution`: The resolution of the scale bar
- `command`: The function to execute when the value of the scale changes
- `orient`: The orientation of the scale

When you change the slider with the mouse pointer, it calls the custom `show_value_1()` function. You are printing the current value of the track bar position to the console for debugging purposes. The `img = Image.blend(im1, im2, float(alpha))` statement creates a blended image. This code

```

photo = ImageTk.PhotoImage(img)
l['image'] = photo
l.photo = photo

```

updates the image in the Tkinter window. The `show_value_1()` function updates every time you change the slider position and the new image is computed. This makes the program interesting and interactive, as you can change the value of alpha to see the transition effect.

The output is shown in Figure 4-2.

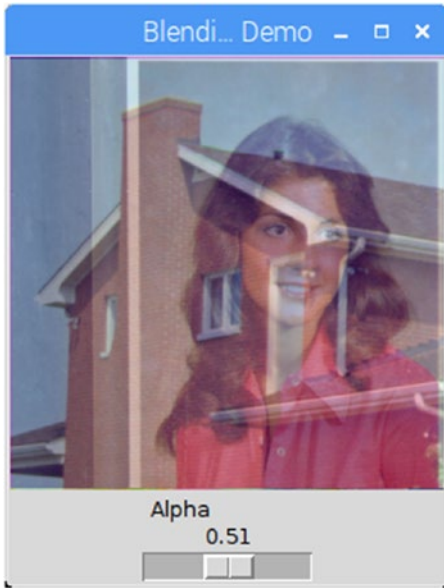


Figure 4-2. Image blending tool

You will use the same basic code skeleton to look at many of the methods in Pillow.

Resizing an Image

You can resize an image using the `resize()` function, as shown in Listing 4-4.

Listing 4-4. prog04.py

```
from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(size):
    print('Resize: ', size, ' : ', size)

    img = im.resize((int(size), int(size)))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.attributes('-fullscreen', True)
im = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

photo = ImageTk.PhotoImage(im)
```

```

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Resize", from_=128,
               to=512, resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()

```

This example varies the image size from (128, 128) to (512, 512). The `resize()` command takes the new size tuple as an argument. The code also invokes the Tkinter window in full-screen mode with the `root.attributes()` function call. To close this window, you have to press Alt+F4 from the keyboard.

Run the code and take a look at the output.

Rotating an Image

You can use `rotate()` method, which takes the angle of rotation as an argument. The code in Listing 4-5 demonstrates this idea.

Listing 4-5. prog05.py

```

from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(angle):
    print('Angle: ', angle)

    img = im.rotate(float(angle))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title("Rotation Demo")
im = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

photo = ImageTk.PhotoImage(im)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Angle", from_=0, to=90,
               resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()

```

Run this code to experience the rotation effect on an image.

You can also transpose the images using the `transpose()` function. It takes one of `PIL.Image.FLIP_LEFT_RIGHT`, `PIL.Image.FLIP_TOP_BOTTOM`, `PIL.Image.ROTATE_90`, `PIL.Image.ROTATE_180`, `PIL.Image.ROTATE_270`, or `PIL.Image.TRANSPOSE` as an argument. The code example in Listing 4-6 shows rotation at 180 degrees.

Listing 4-6. prog06.py

```
from PIL import Image, ImageTk
import tkinter as tk

root = tk.Tk()
root.title("Transpose Demo")
im = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

out = im.transpose(Image.ROTATE_180)

photo = ImageTk.PhotoImage(out)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

root.mainloop()
```

Also, if you change the argument to `transpose()` as follows:

```
out = im.transpose(Image.FLIP_TOP_BOTTOM)
```

It will flip the image vertically.

Crop and Paste Operations

You can crop a part of an image using the `crop()` method. It takes an argument of four-tuples specifying the coordinates of the box to be cropped from the image. Pillow also has a `paste()` method to paste a rectangular image to another image. The `paste()` method takes the image to be pasted and the coordinates as arguments. The program in Listing 4-7 demonstrates how you can rotate a face using a clever combination of `crop()`, `rotate()`, and `paste()`.

Listing 4-7. prog08.py

```
from PIL import Image

im = Image.open("/home/pi/DIP/Dataset/4.1.03.tiff")
face_box = (100, 100, 150, 150)
face = im.crop(face_box)
rotated_face = face.transpose(Image.ROTATE_180)
im.paste(rotated_face, face_box)
im.show()
```


The output of the code in Listing 4-7 is shown in Figure 4-3.

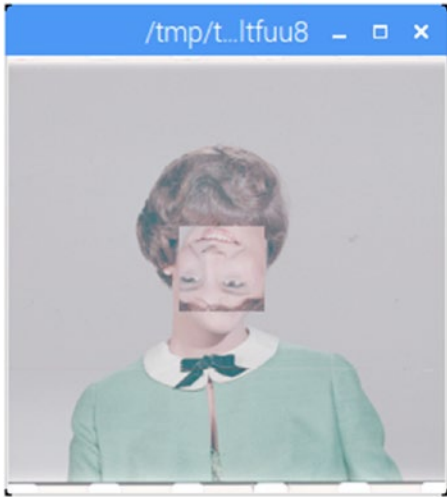


Figure 4-3. Crop and paste demo

Copying and Saving Images to a File

You can use the `copy()` method to copy an entire pillow image to another Python variable. You can save a Pillow image to a file using the `save()` method. A demonstration is shown in Listing 4-8.

Listing 4-8. prog09.py

```
from PIL import Image

im = Image.open("/home/pi/DIP/Dataset/4.1.03.tiff")

im1 = im.copy()
im1.save("test.tiff")
```

The code in Listing 4-8 opens an image from a given location, copies it into the `im1` variable, and saves it to the current location as `test.tiff`.

Knowing the Value of a Particular Pixel

You can determine the value of a particular pixel using `getpixel()`. It is usually a tuple that represents the channel intensities. With RGB images, you get the Red, Green, and Blue intensities. It is used as follows:

```
print(im.getpixel((100,100)))
```

ImageChops Module

This module contains many basic arithmetic and logical operations that you can use on your images. Let's look at them quickly one by one.

You can add two images using the `add()` method. The following is the sample code for adding images:

```
im3 = ImageChops.add(im1, im2)
```

The `add_module()` method adds two images without clipping the result:

```
im3 = ImageChops.add_modulo(im1, im2)
```

The `darker()` method compares two images, pixel-by-pixel, and returns the darker pixels.

```
im3 = ImageChops.darker(im1, im2)
```

The `difference()` method returns the difference of the absolute values of two images:

```
im3 = ImageChops.difference(im1, im2)
```

It uses the following mathematical formula for calculating the difference:

```
image3 = abs(image1 - image2)
```

You can invert an image as follows:

```
im2 = ImageChops.invert(im1)
```

Just like with `darker()`, you can use the `lighter()` method to return the set of lighter pixels:

```
im3 = ImageChops.lighter(im1, im2)
```

`logical_and()` and `logical_or()` are the logical operations on images. These are explained with the help of black-and-white images. The following are example uses:

```
im1 = Image.open("/home/pi/DIP/Dataset/5.1.13.tiff")
im2 = Image.open("/home/pi/DIP/Dataset/5.1.13.tiff")
im2 = im2.transpose(Image.ROTATE_90)
```

```
im3 = ImageChops.logical_and(im1.convert("1"), im2.convert("1"))
```

```
im3 = ImageChops.logical_or(im1.convert("1"), im2.convert("1"))
```

These examples convert the grayscale images to black-and-white images first and then perform the logical operations on them. The result is shown in Figure 4-4.

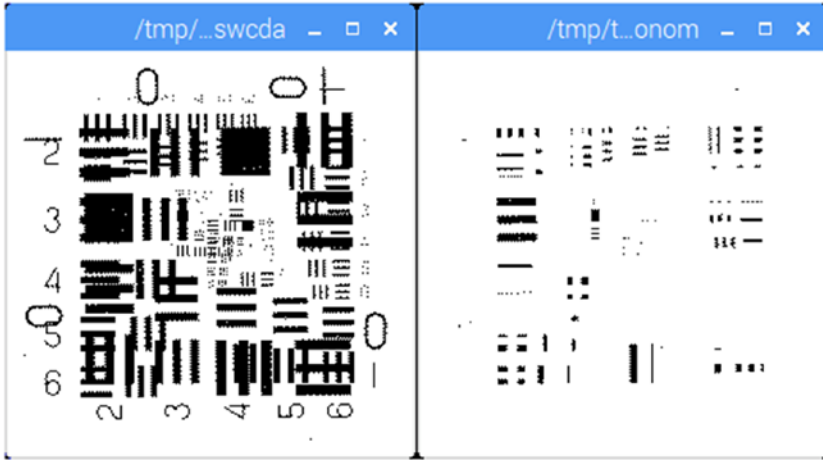


Figure 4-4. Logical operations on images

You can superimpose an image on another using the `multiply()` method:

```
im3 = ImageChops.multiply(im1, im2)
```

The `screen()` method superimposes inverted images on top of each other.

```
im3 = ImageChops.screen(im1, im2)
```

You can subtract one image from another using the `subtract()` method as follows:

```
im3 = ImageChops.subtract(im1, im2)
```

You can subtract without clipping the result as follows:

```
im3 = ImageChops.subtract_modulo(im1, im2)
```

ImageOps Module

This module has many predefined and useful operations. You can automatically adjust the contrast of an image as follows:

```
im2 = ImageOps.autocontrast(im1)
```

You can crop the borders of an image equally from all sides as follows:

```
im2 = ImageOps.crop(im1, 50)
```

The first argument of the `ImageOps.crop()` method is the image and the second argument is the width of the cropped border in pixels.

You can also expand the border of an image. Expanded borders will be filled with black pixels equally on all the sides.

```
im2 = ImageOps.expand(im1, 50)
```

You can flip an image vertically as follows:

```
im2 = ImageOps.flip(im1)
```

You can also flip it horizontally as follows:

```
im2 = ImageOps.mirror(im1)
```

You can reduce the number of bits of all the color channels using the `posterize()` method. It takes the image and the number of bits to keep for every channel as arguments. The following is an example:

```
im2 = ImageOps.posterize(im1, 3)
```

This example keeps only three bits per channel. The result is shown in Figure 4-5.

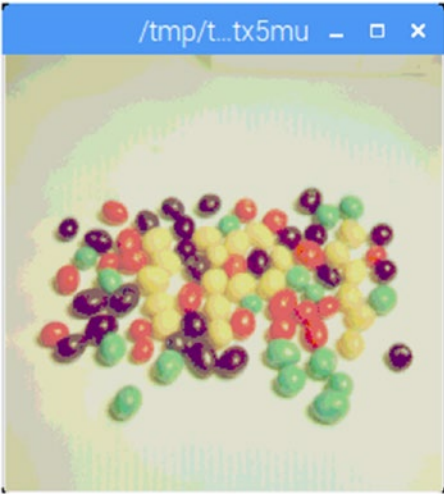


Figure 4-5. *Posterizing an image*

The `solarize()` method inverts all the pixels above a particular grayscale threshold.

```
im2 = ImageOps.solarize(im1, 100)
```

The result is shown in Figure 4-6.

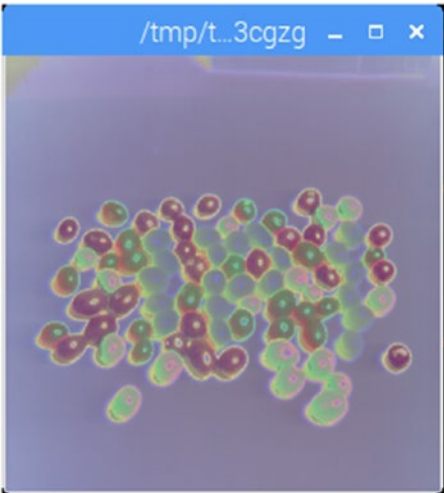


Figure 4-6. *Solarize operation*

EXERCISE

Complete the following coding exercises to gain a better understanding of Pillow.

1. Write the code to show the Green and Blue channels of an image.
 2. Write the code for image conversion between the CMYK and L modes.
-

Conclusion

This chapter explored Image, ImageChops, and ImageOps in detail. In the next chapter, you will explore a few more Pillow modules for advanced operations on images like filtering, enhancements, histograms, and quantization.



Advanced Operations on Images

The last chapter explored the arithmetic and logical operations on images with Pillow. There is more to the world of image processing than that. Pillow has a lot more functionality to offer. You can enhance and filter images. You can also calculate histogram of image and its channels. You will learn all these advanced operations with Pillow in this chapter.

The ImageFilter Module

You can use the ImageFilter module in Pillow to perform a variety of filtering operations on images. You can use filters to remove noise, to add blur effects, and to sharpen and smooth your images. Listing 5-1 shows a simple image-filtering program using the ImageFilter module.

Listing 5-1. prog01.py

```
from PIL import Image, ImageFilter

im1 = Image.open("/home/pi/DIP/Dataset/4.1.08.tiff")

im2 = im1.filter(ImageFilter.BLUR)
im2.show()
```

Figure 5-1 shows the output of this program.



Figure 5-1. *The blur operation*

Apart from BLUR, Pillow has the following filters. Modify the previous code and try all the filters.

```
CONTOUR
DETAIL
EDGE_ENHANCE
EDGE_ENHANCE_MORE
EMBOSS
FIND_EDGES
SMOOTH
SMOOTH_MORE
SHARPEN
```

You can also define custom filters. Listing 5-2 shows how to define a custom filter.

Listing 5-2. prog02.py

```
from PIL import Image, ImageFilter

im1 = Image.open("/home/pi/DIP/Dataset/4.1.08.tiff")

custom_filter = ImageFilter.GaussianBlur(radius=float(5))
im2 = im1.filter(custom_filter)
im2.show()
```


The code in Listing 5-2 uses the `GaussianBlur()` method for a custom filter. The radius of the blur is 5. You can change the radius. Let's use Tkinter to modify the code and make the blur radius dynamic by using the slider in Tkinter (see Listing 5-3).

Listing 5-3. prog03.py

```
from PIL import Image, ImageTk, ImageFilter
import tkinter as tk

def show_value_1(blur_radius):
    print('Gaussian Blur Radius: ', blur_radius)

    custom_filter = ImageFilter.GaussianBlur(radius=float(blur_radius))
    img = im1.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Gaussian Blur Filter Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Blur Radius", from_=0, to=10,
               resolution=0.2, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()
```

The output of Listing 5-3 is shown in Figure 5-2.

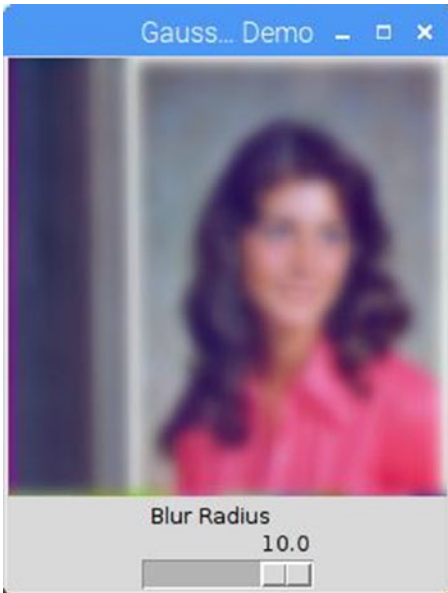


Figure 5-2. Gaussian blur demo

You can see the `GaussianBlur()` has been applied with 10 as the radius of blur in the figure.

The next filter you will explore is the convolution filter. You need to understand the concept of *kernels* for this example. Kernels are the square matrices used in image processing operations. You mostly use kernels for filtering. Kernels produce a variety of effects, including blurs, smoothing, and noise reduction. They are especially used to remove high-frequency components from an image. This process is also called low-pass filtering.

Figure 5-3 shows an example kernel.

1	1	1
1	-4	1
1	1	1

Figure 5-3. An example kernel

Listing 5-4 shows a code example of the kernel used for image convolution.

Listing 5-4. prog04.py

```
from PIL import Image, ImageTk, ImageFilter
import tkinter as tk

root = tk.Tk()
root.title('Convolution Kernel Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.05.tiff")

custom_filter = ImageFilter.Kernel((3, 3), [1, 1, 1, 1, -4, 1, 1, 1, 1])

img = im1.filter(custom_filter)

photo = ImageTk.PhotoImage(img)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

root.mainloop()
```

Run the program in Listing 5-4 and check out the output. As of now, the `ImageFilter.Kernel()` method supports (3,3) and (5,5) kernels only.

You can use the Digital Unsharp Mask filter, as shown in Listing 5-5.

Listing 5-5. prog05.py

```
from PIL import Image, ImageTk, ImageFilter
import tkinter as tk

def show_value_1(blur_radius):
    print('Unsharp Blur Radius: ', blur_radius)

    custom_filter = ImageFilter.UnsharpMask(radius=float(blur_radius))
    img = im1.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Digital Unsharp Mask Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

photo = ImageTk.PhotoImage(im1)
```

```

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Blur Radius", from_=0, to=10,
               resolution=0.2, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()

```

The digital unsharpening mask sharpens the image. Figure 5-4 shows the output with a mask radius of size 10.

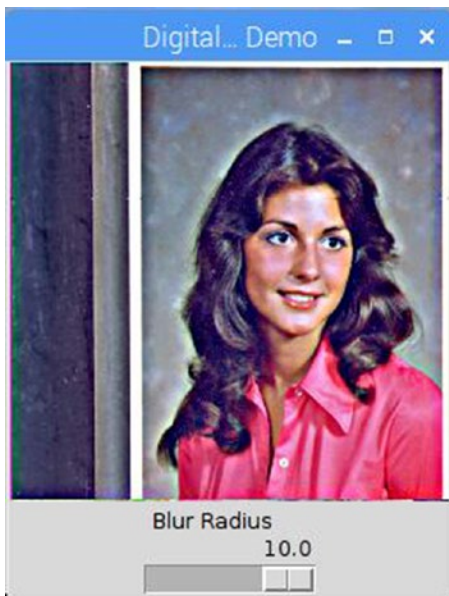


Figure 5-4. Digital unsharp demo

The radius of size 10 creates a highly sharpened image.

Let's study the median, min, and max filters in Pillow. All these filters only accept an odd number as the window size. The code in Listing 5-6 demonstrates a median filter.

Listing 5-6. prog06.py

```

from PIL import Image, ImageTk, ImageFilter
import tkinter as tk

def show_value_1(window_size):
    print('Window Size: ', window_size)

    if (int(window_size) % 2 == 0):

```

```

    print("Invalid Window Size...\nWindow Size must be an odd number...")
else:
    custom_filter = ImageFilter.MedianFilter(size=int(window_size))
    img = im1.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Median Filter Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Window Size", from_=1, to=19,
               resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()

```

The output shown in Figure 5-5 has a window size of 19.

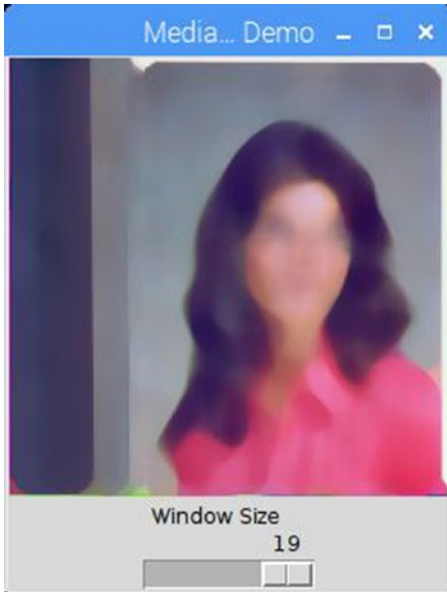


Figure 5-5. A median filter

If you change the filter to a min filter using the following code line:

```
custom_filter = ImageFilter.MinFilter(size=int(window_size))
```

You'll get the output shown in Figure 5-6.

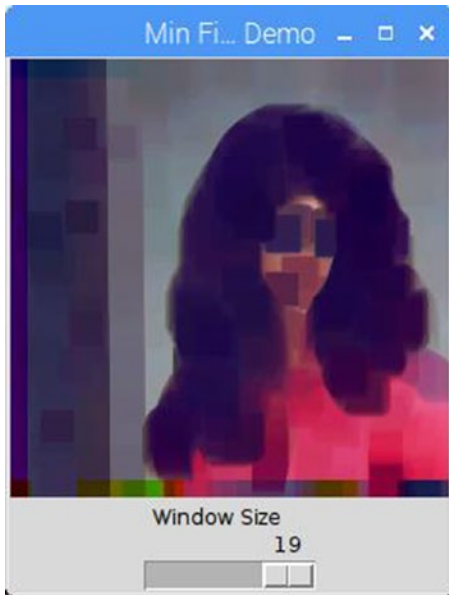


Figure 5-6. A min filter

Here's an example of the max filter:

```
custom_filter = ImageFilter.MaxFilter(size=int(window_size))
```

The output for the max filter is shown in Figure 5-7.



Figure 5-7. The max filter

Next, you'll see an example of a mode filter. The mode filter works with even and odd window sizes. Listing 5-7 shows a code example of the mode filter.

Listing 5-7. prog09.py

```
from PIL import Image, ImageTk, ImageFilter
import tkinter as tk

def show_value_1(window_size):
    print('Window Size: ', window_size)

    custom_filter = ImageFilter.ModeFilter(size=int(window_size))
    img = im1.filter(custom_filter)
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Mode Filter Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

photo = ImageTk.PhotoImage(im1)
```

```

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Window Size", from_=1, to=19,
               resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()

root.mainloop()

```

The output with a window size of 19 is shown in Figure 5-8.

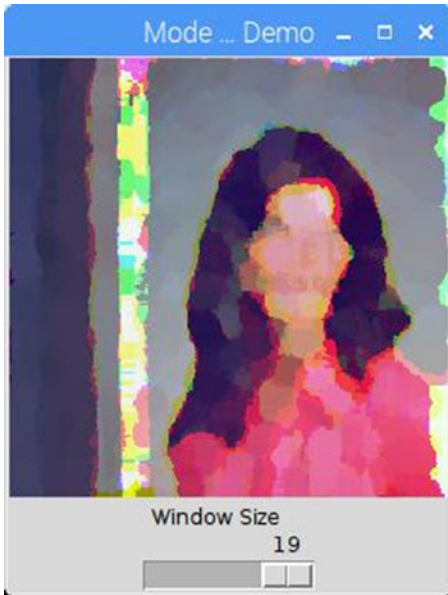


Figure 5-8. *The mode filter*

The ImageEnhance Module

You can use the ImageEnhance module in Pillow to adjust the contrast, color, sharpness, and brightness of an image just like you used to do it in old analog television sets.

Listing 5-8 shows the code for color adjustment.

Listing 5-8. prog10.py

```

from PIL import Image, ImageTk, ImageEnhance
import tkinter as tk

def show_value_1(factor):
    print('Color Factor: ', factor)

```



```

    enhancer = ImageEnhance.Color(im1)
    img = enhancer.enhance(float(factor))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Color Adjustment Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Color Factor", from_=0, to=2,
               resolution=0.1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()
w1.set(1)
root.mainloop()

```

In Listing 5-8, the image processing code is as follows:

```

enhancer = ImageEnhance.Color(im1)
img = enhancer.enhance(float(factor))

```

You can follow the same style of coding for all the other image enhancement operations. First, you create an enhancer and then you apply the enhancement factor to that. You also must have observed `w1.set(1)` in the code. This sets the scale to 1 at the beginning. Changing the argument to `set()` changes the default position of the scale pointer.

Run the program in Listing 5-8 and take a look at the output.

To change the contrast, use the code in Listing 5-9.

Listing 5-9. `prog11.py`

```

from PIL import Image, ImageTk, ImageEnhance
import tkinter as tk

def show_value_1(factor):
    print('Contrast Factor: ', factor)

    enhancer = ImageEnhance.Contrast(im1)
    img = enhancer.enhance(float(factor))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

```

```

root = tk.Tk()
root.title('Contrast Adjustment Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.04.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Contrast Factor", from_=0, to=2,
               resolution=0.1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()
w1.set(1)
root.mainloop()

```

Run the program in Listing 5-9 and take a look at the output.
The following enhancer is used to change the brightness:

```
enhancer = ImageEnhance.Brightness(im1)
```

The following enhancer is used to change the sharpness:

```
enhancer = ImageEnhance.Sharpness(im1)
```

For finer control of the sharpness, use the following code for the scale:

```

w1 = (tk.Scale(root, label="Sharpness Factor", from_=0, to=2,
               resolution=0.1, command=show_value_1, orient=tk.HORIZONTAL))

```

These were all image enhancement operations. The next section looks at more advanced image operations.

Color Quantization

Color quantization is the process of reducing the number of distinct colors in an image. The new image should be similar to the original image in appearance. Color quantization is done for a variety of purposes, including when you want to store an image in a digital medium. Real-life images have millions of colors. However, encoding them in the digital format and retaining all the color related information would result in a huge image size. If you limit the number of colors in the image, you'll need less space to store the color-related information. This is the practical application of quantization. The Image module has a method called `quantize()` that's used for image quantization.

The code in Listing 5-10 demonstrates image quantization in Pillow.

Listing 5-10. prog14.py

```
from PIL import Image, ImageTk
import tkinter as tk

def show_value_1(num_of_col):
    print('Number of colors: ', num_of_col)

    img = im1.quantize(int(num_of_col))
    photo = ImageTk.PhotoImage(img)
    l['image'] = photo
    l.photo = photo

root = tk.Tk()
root.title('Color Quantization Demo')

im1 = Image.open("/home/pi/DIP/Dataset/4.1.06.tiff")

photo = ImageTk.PhotoImage(im1)

l = tk.Label(root, image=photo)
l.pack()
l.photo = photo

w1 = (tk.Scale(root, label="Number of colors", from_=4, to=16,
               resolution=1, command=show_value_1, orient=tk.HORIZONTAL))
w1.pack()
w1.set(256)
root.mainloop()
```

Run the program in Listing 5-10 and take a look at the results of quantization by changing the slider.

Histograms and Equalization

You likely studied frequency tables in statistics in school. Well, the histogram is nothing but a frequency table visualized. You can calculate a histogram of any dataset represented in the form of numbers.

The Image module has a method called `histogram()` that's used to calculate the histogram of an image. An RGB image has three 8-bit channels. This means that it can have a staggering **256x256x256** number of colors. Drawing a histogram of such a dataset would be very difficult. So, the `histogram()` method calculates the histogram of individual channels in an image. Each channel has 256 distinct intensities. The histogram is a list of values for each intensity level of a channel.

The histogram for each channel has 256 numbers in the list. Suppose the histogram for the Red channel has the values (4, 8, 0, 19, ..., 90). This means that four pixels have the red intensity of 0, eight pixels have the red intensity of 1, no pixel has red intensity of 2, 19 pixels have the red intensity of three, and so on, until the last value, which indicates that 90 pixels have the red intensity of 255.

When you combine the histogram of all three channels, you get a list of 768 numbers. In this chapter, we will just compute the histogram. We will not show it visually. When you learn about the advanced image processing library `scipy.ndimage`, you will then learn how to represent histograms for each channel individually.

The code in Listing 5-11 calculates and stores the histograms of an image and its individual channels.

Listing 5-11. `progl5.py`

```
from PIL import Image

im1 = Image.open("/home/pi/DIP/Dataset/4.2.07.tiff")

print(len(im1.histogram()))

r, g, b = im1.split()

print(len(r.histogram()))
print(len(g.histogram()))
print(len(b.histogram()))
```

Modify this program to directly print the histograms of the image and channels.

A grayscale image (L mode image) will have a histogram of only 256 values because it has only a single channel.

Histogram Equalization

You can adjust the histogram to enhance the image contrast. This technique is known as *histogram equalization*. The `ImageOps.equalize()` method equalizes the histogram of the image. Listing 5-12 shows an example of this process.

Listing 5-12. `progl6.py`

```
from PIL import Image, ImageOps

im1 = Image.open("/home/pi/DIP/Dataset/4.2.07.tiff")

print(im1.histogram())

im2 = ImageOps.equalize(im1)

print(im2.histogram())

im2.show()
```

The program in Listing 5-12 prints the histogram of the original image after the equalization. Add the `im1.show()` statement to the program and then run it to see the difference between the images.

EXERCISE

Complete the following exercises to gain a better understanding of advanced image processing methods in Pillow.

1. Write the code to demonstrate all the predefined filters defined at the beginning of the chapter.
2. Calculate and equalize the histogram for a grayscale (L mode) image.
3. Modify the code for convolution using the following kernel:

```
custom_filter = ImageFilter.Kernel((5, 5), [1,1,1,1,1,
1,1,1,1,1, 1,1,-10,1,1, 1,1,1,1,1, 1,1,1,1,1])
```

Conclusion

In this chapter, you explored the Pillow library for advanced image processing. Pillow is good for the beginners who want to get started with an easy-to-program and less mathematical image-processing library. However, if you want a more mathematical and scientific approach, then Pillow might not be your best choice. In the following chapters, you will learn about a more powerful library for image processing, `scipy.ndimage`. It's widely used by scientific community all over the world for image processing. You will also learn the basics of the NumPy and matplotlib libraries, which come in handy when processing and displaying images.



Introduction to Scientific Python

In the last chapter, you studied advanced image processing with Pillow. Pillow is a nice starting point for image processing operations. However, it has its own limitations. When it comes to implementing elaborate image processing operations like segmentation, morphological operations, advanced filters, and measurements, Pillow is inadequate. You really need to use advanced libraries for image processing. The SciPy toolkit serves as a foundation for all the scientific usage of Python. It is extensively used for a variety of scientific operations.

SciPy stands for *Scientific Python*. It extensively uses NumPy (Numerical Python) and matplotlib for numeric operations and data visualization. This chapter explains SciPy, NumPy, and matplotlib. It also explores introductory level programming examples of NumPy, `scipy.misc`, and matplotlib.

The Scientific Python Stack

SciPy is an open source library for scientific and technical computing in Python.

SciPy has modules for ordinary differential equations solvers, fast Fourier transforms, optimization, linear algebra, integration, interpolation, signal processing, and image processing. SciPy is used extensively by scientific, mathematical, and engineering communities around the world. There are many other libraries that use core modules of SciPy and NumPy. OpenCV and SciKit are good examples of this.

The SciPy stack has the following components:

- NumPy
- SciPy library
- matplotlib
- IPython
- Pandas
- SymPy
- Nose

Figure 6-1 aptly summarizes the role of the Python SciPy stack in the world of scientific computing.

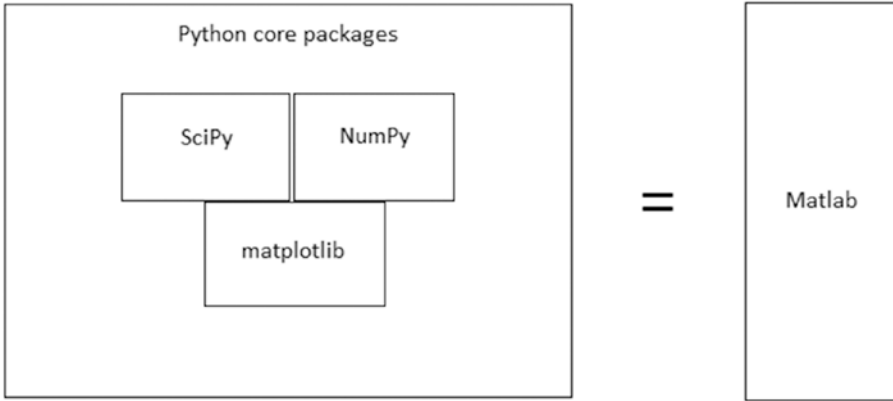


Figure 6-1. The SciPy stack

Installing the SciPy Stack

The best way to install the SciPy stack on Raspberry Pi is to use `pip`.

First, upgrade `pip` with the following command:

```
sudo python3 -m pip install --upgrade pip
```

Then install the SciPy stack with the following command:

```
pip3 install numpy scipy matplotlib ipython jupyter pandas sympy nose
```

This installs the entire SciPy stack.

A Simple Program

The `scipy.misc` module is used for basic image processing operations. Listing 6-1 shows a basic example of reading and displaying an image.

Listing 6-1. `prog01.py`

```
from scipy import misc

img = misc.imread('/home/pi/DIP/Dataset/4.2.01.tiff')

misc.imshow(img)
```

The code in Listing 6-1 reads an image from the path provided in the `imread()` method and then the `imshow()` method will display it using `xlib`.

The `scipy.misc` module has three built-in images, which can be used as shown in Listing 6-2.

Listing 6-2. `prog02.py`

```
from scipy import misc

img1 = misc.face()
img2 = misc.lena()
img3 = misc.ascent()

misc.imshow(img1)
misc.imshow(img2)
misc.imshow(img3)
```

`face()` is a face of a raccoon. `lena()` is standard test image and `ascent()` is a grayscale image.

Simple Image Processing

`scipy.misc` has three methods for simple operations. `scipy.imfilter()` applies various filters to images. Listing 6-3 shows an example.

Listing 6-3. `prog03.py`

```
from scipy import misc

misc.imshow(misc.imfilter(misc.face(), 'edge_enhance_more'))
```

The code in Listing 6-3, doesn't use an intermediate variable to store the image. It displays it directly by passing the `imshow()` method. The `imfilter()` method accepts two arguments. The first is the image to be filtered. The second is the type of pre-defined filter to be applied. Allowed values for the filter-type are 'blur', 'contour', 'detail', 'edge_enhance', 'edge_enhance_more', 'emboss', 'find_edges', 'smooth', 'smooth_more', and 'sharpen'.

You can resize the image to 50 percent, as shown in Listing 6-4.

Listing 6-4. `prog04.py`

```
from scipy import misc

misc.imshow(misc.imresize(misc.face(), 50))
```


You can also rotate the image at a certain angle, as shown in Listing 6-5.

Listing 6-5. prog05.py

```
from scipy import misc

misc.imshow(misc.imrotate(misc.face(), 45))
```

Introduction to NumPy

Let's get started with the basics of the NumPy library in Python. Consider the code in Listing 6-6.

Listing 6-6. prog06.py

```
from scipy import misc

img = misc.face()

print(type(img))
```

The output of this program is as follows:

```
<class 'numpy.ndarray'>
```

This means that the data type of the image is ndarray in NumPy. In order to get started with the scientific image processing and any type of scientific programming in general, you need to know what NumPy is.

The NumPy homepage at www.numpy.org says this:

NumPy is the fundamental package for scientific computing with Python.

It offers the following features:

- A powerful multi-dimensional array object
- Useful methods for mathematical computations
- Wrappers and tools for integration with faster C/C++ and FORTRAN code

In order to get started with the image processing with SciPy and NumPy, you need to learn basics of N-dimensional (or multi-dimensional) array objects in NumPy. Let's get started with that.

NumPy's N-dimensional array is a homogeneous (contains all the elements of the same data type) multidimensional array. It has multiple dimensions. Each dimension is known as an axis. The class corresponding to the N-dimensional array in NumPy is `numpy.ndarray`. This is what you saw in Listing 6-6. All the major image processing and computer vision libraries, like Mahotas, OpenCV, scikit-image, and `scipy.ndimage` (you will extensively study this last one in this book) use `numpy.ndarray` to represent images. All these libraries have `read()`, `open()`, and `imread()` methods for loading images from disk to a `numpy.ndarray` object.

NumPy and N-dimensional arrays in NumPy are such vast topics themselves that it would require volumes of books to explain them fully. Hence, you will learn the relevant and important features of these as and when they're needed. For now, you need to understand a few important ndarray properties that will help you understand important attributes of the images that ndarray represents.

Consider the code in Listing 6-7.

Listing 6-7. `prog07.py`

```
from scipy import misc

img = misc.face()

print(img.dtype)
print(img.shape)
print(img.ndim)
print(img.size)
```

The output is as follows:

```
uint8
(768, 1024, 3)
3
2359296
```

Let's look at what each of these means. The `dtype` attribute is for the data type of the elements that represent the image. In this case, it is `uint8`, which means an unsigned 8-bit integer. This means it can have 256 distinct values. `shape` means the dimension/size of the images. In this case, it is a color image. Its resolution is 1024x768 and it has three color channels corresponding to the colors Red, Green, and Blue. Each channel for each pixel can have one of the 256 possible values. So a combination of this can produce $256 \times 256 \times 256$ distinct colors for each pixel.

You can visualize a color image as an arrangement of three two-dimensional planes. A grayscale image is a single plane of grayscale values. `ndim` represents the dimensions. A color image has three dimensions and a grayscale image has two dimensions. `size` represents for the total number of elements in the array. It can be calculated by multiplying the values of the dimensions. In this case, it is $768 \times 1024 \times 3 = 2359296$.

You can see the RGB value corresponding to each individual pixel, as shown in Listing 6-8.

Listing 6-8. prog08.py

```
from scipy import misc  
  
img = misc.face()  
  
print(img[10, 10])
```

The code in Listing 6-8 accesses the value of the pixel located at (10, 10). The output is [172 169 188].

This concludes the basics about NumPy and image processing. You will learn more about NumPy as and when needed throughout the chapters.

Matplotlib

You have used the `misc.imshow()` method for displaying an image. While this method is useful for simple applications, it is primitive. You need to use a more advanced framework for scientific applications. Matplotlib serves this purpose. It is a Matlab-style plotting and data visualization library for Python. You installed it when you installed the SciPy stack. It is an integral part of the SciPy stack. Just like NumPy, matplotlib is too a vast topic and warrants another book. The examples in this book use the `pyplot` module in `matplotlib` for the image processing requirements. Listing 6-9 shows a simple program for the image processing.

Listing 6-9. prog09.py

```
import scipy.misc as misc  
import matplotlib.pyplot as plt  
  
img = misc.face()  
  
plt.imshow(img)  
plt.show()
```

The code in Listing 6-9 imports the `pyplot` module. The `imshow()` method adds the image to the plot window. The `show()` method shows the plot window. The output is shown in Figure 6-2.

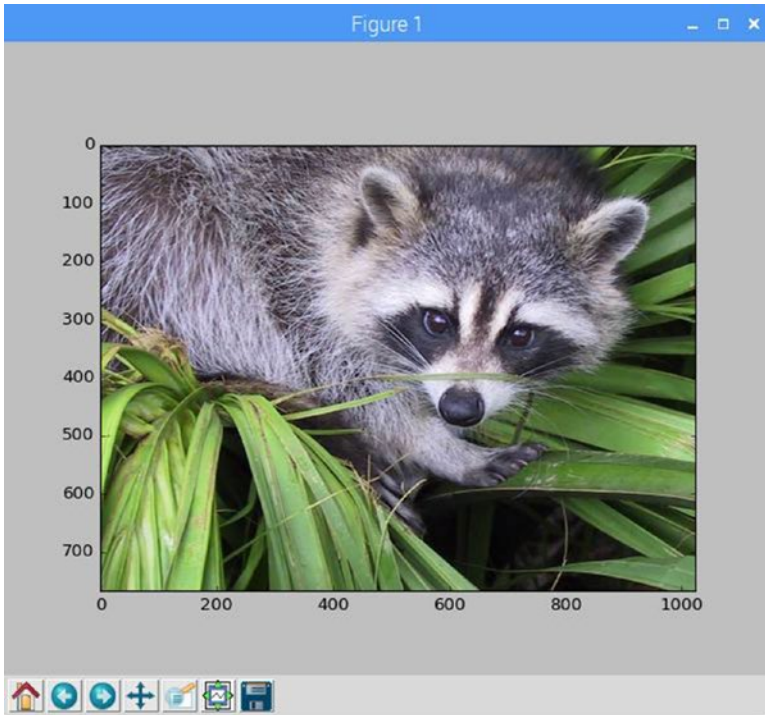


Figure 6-2. The pyplot demo

You can also turn off the axes (or the ruler) and add a title to the image, as shown in Listing 6-10.

Listing 6-10. prog10.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.lena()

plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('face')
plt.show()
```

As the image is grayscale, you have to choose a gray color map in the `imshow()` method so the image color space is properly displayed in the plot window. `axis('off')` is used to turn the axes off. The `title()` method is used to specify the title of the image. The output is shown in Figure 6-3.



Figure 6-3. *Lena image with title and axes off*

You can use `imshow()` to push multiple images to an image grid in the plot window, as shown in Listing 6-11.

Listing 6-11. `prog11.py`

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img1 = misc.face()
img2 = misc.ascent()
img3 = misc.lena()

titles = ['face', 'ascent', 'lena']
images = [img1, img2, img3]

plt.subplot(1, 3, 1)
plt.imshow(images[0])
plt.axis('off')
plt.title(titles[0])

plt.subplot(1, 3, 2)
```

```
plt.imshow(images[1], cmap='gray')
plt.axis('off')
plt.title(titles[1])

plt.subplot(1, 3, 3)
plt.imshow(images[2], cmap='gray')
plt.axis('off')
plt.title(titles[2])

plt.show()
```

You have used the `subplot()` method before, with `imshow()`. The first two arguments in the `subplot()` method specify the dimensions of the grid and the third argument specifies the position of the image in the grid. The numbering of the images in the grid starts from the top-left edge. The top-left position is the first position, the next position is the second one, and so on. The result is shown in Figure 6-4.



Figure 6-4. Multiple image grid

Image Channels

You can separate image channels of a multi-channel image. The code for that process is shown in Listing 6-12.

Listing 6-12. `Prog12.py`

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.face()

r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

titles = ['face', 'Red', 'Green', 'Blue']
images = [img, r, g, b]
```

```

plt.subplot(2, 2, 1)
plt.imshow(images[0])
plt.axis('off')
plt.title(titles[0])

plt.subplot(2, 2, 2)
plt.imshow(images[1], cmap='gray')
plt.axis('off')
plt.title(titles[1])

plt.subplot(2, 2, 3)
plt.imshow(images[2], cmap='gray')
plt.axis('off')
plt.title(titles[2])

plt.subplot(2, 2, 4)
plt.imshow(images[3], cmap='gray')
plt.axis('off')
plt.title(titles[3])

plt.show()

```

The result of the code in Listing 6-12 is shown in Figure 6-5.



Figure 6-5. Separate image channels

You can use the `np.dstack()` method, which merges all the channels, to create the original image, as shown in Listing 6-13.

Listing 6-13. `prog13.py`

```
import scipy.misc as misc
import matplotlib.pyplot as plt
import numpy as np

img = misc.face()

r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

output = np.dstack((r, g, b))

plt.imshow(output)
plt.axis('off')
plt.title('Combined')
plt.show()
```

Run the code in Listing 6-13 to see the workings of the `np.dstack()` for yourself.

Conversion Between PIL Image Objects and NumPy ndarrays

You can use the `np.asarray()` and `Image.fromarray()` methods to convert between PIL images and NumPy ndarrays, as shown in Listing 6-14.

Listing 6-14. `prog14.py`

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

img = Image.open('/home/pi/DIP/Dataset/4.2.01.tiff')

print(type(img))
img.show()

num_img = np.asarray(img)
plt.imshow(num_img)
plt.show()
print(type(num_img))
```



```
img = Image.fromarray(np.uint8(num_img))

print(type(img))
img.show()
```

The console output is as follows:

```
<class 'PIL.TiffImagePlugin.TiffImageFile'>
<class 'numpy.ndarray'>
<class 'PIL.Image.Image'>
```

You can use the `misc.fromimage()` and `misc.toimage()` methods to achieve the same conversion, as shown in Listing 6-15.

Listing 6-15. `progl5.py`

```
from PIL import Image
import scipy.misc as misc
import matplotlib.pyplot as plt

img = Image.open('/home/pi/DIP/Dataset/4.2.01.tiff')

print(type(img))
img.show()

num_img = misc.fromimage(img)

print(type(num_img))
plt.imshow(num_img)
plt.show()

img = misc.toimage(num_img)

print(type(img))
img.show()
```

The console output of Listing 6-15 is the same as the earlier example shown in Listing 6-14.

Conclusion

In this chapter, you were introduced to SciPy stack, NumPy, and matplotlib. You also explored the `scipy.misc` module for basic image processing and conversion. In the next chapter, you will start exploring the `scipy.ndimage` module for more image processing operations.

CHAPTER 7



Transformations and Measurements

In the last chapter, you were introduced to the scientific Python stack. You learned the basics of NumPy and matplotlib. You learned about the useful modules called ndarray and pyplot from NumPy and matplotlib respectively. You also learned about the `scipy.misc` module and basic image processing with it. In this chapter, you will further explore SciPy. You will learn to use the `scipy.ndimage` library for processing images. You will explore the methods for transformations and measurements. This is a short and simple chapter.

Transformations

You studied a few basic transformations with `scipy.misc` in the last chapter. Here, you will look at few more.

■ **Note** Just like with `scipy.misc`, `scipy.ndimage` has an `imread()` method that serves the same purpose as `scipy.misc.imread()`. You will be using `face()`, `lena()`, and `ascent()` throughout the rest of the book. However, if you want to use other images in the Dataset directory, you can use `imread()` from the `misc` or `ndimage` module in SciPy.

Let's get started with the simple transformation of shifting. The `shift()` method accepts the image and the values to be applied to the coordinates for shifting as arguments. The example is shown in Listing 7-1.

Listing 7-1. `prog01.py`

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt
```

```
img = misc.lena()  
  
output = ndi.shift(img, [20, -20])  
  
plt.imshow(output, cmap='gray')  
plt.title('Shift Demo')  
plt.axis('off')  
plt.show()
```

Figure 7-1 shows the output.

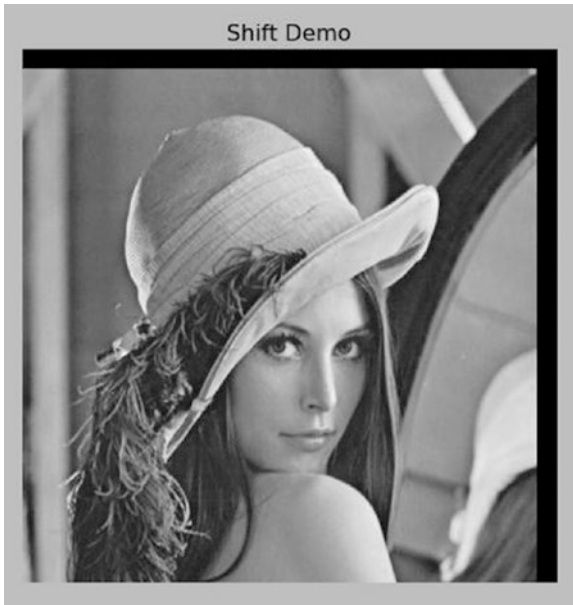


Figure 7-1. Shift demo

You can also zoom in on the image using the `zoom()` method. You have to pass the image and the scale of zooming for each of the axes as arguments to the method. Listing 7-2 shows an example of doing this.

Listing 7-2. prog02.py

```
import scipy.misc as misc  
import scipy.ndimage as ndi  
import matplotlib.pyplot as plt  
  
img = misc.lena()
```

```
plt.imshow(ndi.zoom(img, [5, 3]), cmap='gray')
plt.title('Zoom Demo')
plt.axis('off')

plt.show()
```

The output of Listing 7-2 is shown in Figure 7-2.



Figure 7-2. Zoom demo

Measurements

In Chapter 5, you learned how to create a histogram. SciPy also has a `histogram()` method that computes a histogram of image channels. You can use matplotlib to display the histogram. Listing 7-3 shows an example of a histogram computation with SciPy; it's been plotted using matplotlib.

Listing 7-3. prog03.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt

img = misc.lena()
```

```

hist = ndi.histogram(img, 0, 255, 256)

plt.plot(hist, 'k')
plt.title('Lena Histogram')
plt.grid(True)

plt.show()

```

This program passes the image, the minimum pixel value, the maximum pixel value, and the number of bins as arguments to the `histogram()` method. It passes the histogram array returned by the `histogram()` method and the color of the histogram graph ('k') as arguments to the `plot()` method. The resultant histogram is shown in Figure 7-3.

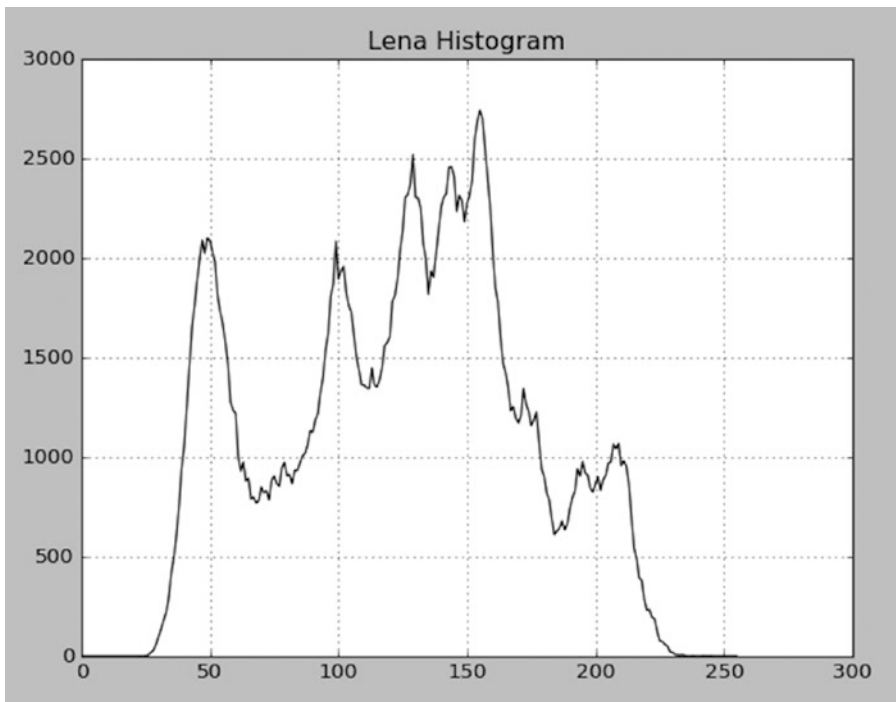


Figure 7-3. Histogram of the Lena image

The `minimum()` and `maximum()` methods return the minimum and maximum values of the pixels, respectively. The `minimum_position()` and `maximum_position()` methods return the values of the positions of pixels with the minimum and maximum intensities, respectively. The `extrema()` method combines the functionalities of the four methods. The demonstration is shown in Listing 7-4.

Listing 7-4. prog04.py

```
import scipy.misc as misc
import scipy.ndimage as ndi

img = misc.ascent()

print(ndi.minimum(img))
print(ndi.minimum_position(img))
print(ndi.maximum(img))
print(ndi.maximum_position(img))

print(ndi.extrema(img))
```

The output is as follows:

```
0
(201, 268)
255
(190, 265)
(0, 255, (201, 268), (190, 265))
```

The first line is the value of the lowest intensity pixel. The second line is its position. The next two lines refer to the value of the highest intensity pixel and its position. The next line is the output of the `extrema()` method, which combines all the previous output.

The program shown in Listing 7-5 demonstrates the statistical information about the image pixels. The methods work on the intensity values of the pixel; the names of methods are fairly self-explanatory.

Listing 7-5. prog05.py

```
import scipy.misc as misc
import scipy.ndimage as ndi

img = misc.ascent()

print(ndi.sum(img))
print(ndi.mean(img))
print(ndi.median(img))
print(ndi.variance(img))
print(ndi.standard_deviation(img))
```

The output is as follows:

```
22932324
87.4798736572
80.0
2378.9479363
48.7744598771
```

You can also calculate the center of mass based on the intensity values of the pixels, as shown in Listing 7-6.

Listing 7-6. prog06.py

```
import scipy.misc as misc
import scipy.ndimage as ndi

img = misc.ascent()

print(ndi.center_of_mass(img))
```

This will produce the following output:

```
(259.99734235396289, 254.60907272197969)
```

EXERCISE

Complete the following exercises to gain an in-depth understanding of transformations and measurements.

- Explore the `ndimage.rotate()` method. Write the code for using it.
- We calculated the histogram for a grayscale image. Calculate the histogram for a color image by computing it separately for each channel. Use the relevant color for the histogram graph for displaying using `pyplot`.
- We calculated the measurements for grayscale images. Do the same for color images.

Conclusion

In this chapter, you explored the methods for transformations and measurements. You studied the shift and zoom transformations. You calculated the histogram of a grayscale image. You also calculated statistical information about the images.

In the next chapter, you will study the image kernels and filters, their types, and their applications in the image enhancement in detail.

CHAPTER 8



Filters and Their Application

In the last chapter, you learned about the `scipy.ndimage` module of the SciPy library. You learned how to apply transformations like `shift()` and `zoom()` on an image. You also learned how to obtain statistical information about an image. You saw how to compute and plot the histogram for an image.

In Chapter 5, you studied image filters using the Pillow library. In this chapter, you will study the theory behind the filters in detail. You will also see the types of filters, kernels, and the applications of the filters in image processing.

Filters

Image filters are used to modify and/or enhance an image. All the image-filtering techniques you will study in this chapter are frequently used in image processing software programs like GIMP and Photoshop. Filtering is mostly a neighborhood operation. It is also called *local filtering*. For this, you need a matrix known as a *kernel*. A kernel is a two-dimensional matrix used to perform image processing. It is also known as a *filter*.

Let's look at a simple local filtering operation, the *convolution*. In the convolution operation, the filter (or the kernel) is applied to each pixel of the image in such a way that the center of the kernel is multiplied with the pixel. All the neighboring pixels of the pixel being processed are also multiplied by the corresponding element of the matrix. All the values are then added together to give an intensity value of the output pixel.

■ **Note** A kernel is always an odd matrix. For example, a size of a kernel could be (3, 3) or (5, 5). Bigger kernels need more time to process the image due to the number of computations involved.

Take a look at the simple example in Listing 8-1 for the convolution.

Listing 8-1. prog01.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import numpy as np
```



```
import matplotlib.pyplot as plt

img = misc.lena()

k = np.array([[1, 1, 1, 1, 1],
              [1, 1, 1, 1, 1],
              [1, 1, 1, 1, 1],
              [1, 1, 1, 1, 1],
              [1, 1, 1, 1, 1]])

output = ndi.convolve(img, k)

plt.imshow(output, cmap='gray')
plt.title('Convolution')
plt.axis('off')
plt.show()
```

Run the program for yourself and take a look at the output. You will find that the output image is blurry. Convolution kernels can produce a variety of outputs and it is a vast topic in and of itself. Try to modify the values in the kernel and see the results for yourself.

■ **Note** Visit <http://setosa.io/ev/image-kernels/> and <https://docs.gimp.org/en/plugin-in-convmatrix.html> for more and detailed information on kernels.

Low-Pass Filters

Low-pass filters filter out high-frequency information. In other words, they allow information with a frequency lower than the cutoff frequency to pass through. They are usually called smoothing, blurring, or averaging filters in the image processing domain, as they are used to blur images or remove noise. A convolution filter is a basic low-pass filter. In this section, you will study low-pass filters and their applications.

Low-Pass Filters for Blurring

This section discusses how to use the low-pass filters for blurring images. You will use Gaussian and uniform filters for this purpose. The example in Listing 8-2 shows a Gaussian low-pass filter used to blur an image.

Listing 8-2. prog02.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt
```

```

img = misc.lena()

output1 = ndi.gaussian_filter(img, sigma=3)
output2 = ndi.gaussian_filter(img, sigma=5)
output3 = ndi.gaussian_filter(img, sigma=7)

output = [output1, output2, output3]
titles = ['Sigma = 3', 'Sigma = 5', 'Sigma = 7']

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

In the code in Listing 8-2, you are applying a Gaussian filter with standard deviation (sigma) values of 3, 5, and 7, respectively. You'll see a progressively blurred image, as shown in Figure 8-1.



Figure 8-1. Gaussian filter

You can also use the uniform filter to get this blurring/smoothing effect. Listing 8-3 shows an example of this effect.

Listing 8-3. prog03.py

```

import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt

img = misc.face()

output1 = ndi.uniform_filter(img, size=19)
output2 = ndi.uniform_filter(img, size=25)

```

```

output3 = ndi.uniform_filter(img, size=31)

output = [output1, output2, output3]
titles = ['Size = 19', 'Size = 25', 'Size = 31']

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

In the code in Listing 8-3, you are applying the uniform filter of various sizes to the test image. The result is shown in Figure 8-2.



Figure 8-2. Uniform filter

Using Low-Pass Filters for Noise Removal

Another application of a low-pass filter is to remove noise from an image. You can define *noise* as any unwanted interference in a signal. All electronic devices are prone to noise. A small amount of noise is always present in all electronic signals, including electronically captured images. Noise is usually in the form of sharp and high-frequency data, which can be reduced by using low-pass filters like Gaussian and median filters. The example in Listing 8-4 demonstrates a Gaussian filter used to remove noise.

Listing 8-4. prog04.py

```

import scipy.misc as misc
import scipy.ndimage as ndi
import numpy as np
import matplotlib.pyplot as plt

img = misc.lena()

noisy = img + 0.8 * img.std() * np.random.random(img.shape)

output1 = ndi.gaussian_filter(noisy, sigma=1)

```

```

output2 = ndi.gaussian_filter(noisy, sigma=3)
output3 = ndi.gaussian_filter(noisy, sigma=5)

output = [noisy, output1, output2, output3]
titles = ['Noisy', 'Sigma = 1', 'Sigma = 3', 'Sigma = 5']

for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

In the code in Listing 8-4, the following line:

```
noisy = img + 0.8 * img.std() * np.random.random(img.shape)
```

is used to create the noisy image for the sample. As you are using the `random()` method to generate noise, the output image will be different every time the code is executed. The result is shown in Figure 8-3.

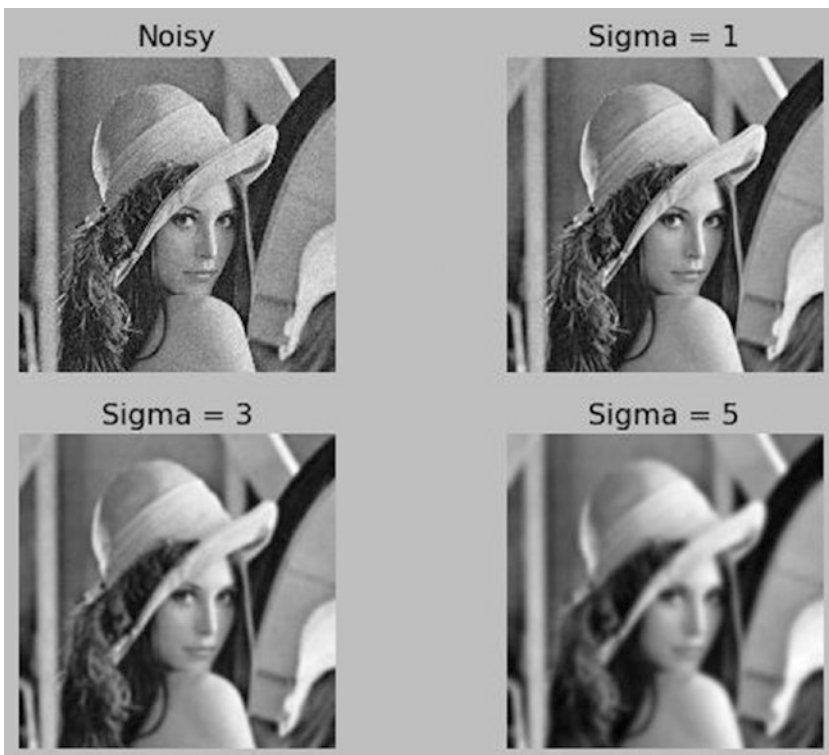


Figure 8-3. Gaussian filter for noise removal

Listing 8-5 shows an example of a median filter used to remove noise.

Listing 8-5. prog05.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import numpy as np
import matplotlib.pyplot as plt

img = misc.lena()

noisy = img + 0.8 * img.std() * np.random.random(img.shape)

output1 = ndi.median_filter(noisy, 3)
output2 = ndi.median_filter(noisy, 7)
output3 = ndi.median_filter(noisy, 9)

output = [noisy, output1, output2, output3]
titles = ['Noisy', 'Size = 1', 'Size = 3', 'Size = 5']

for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()
```

This example uses the median filters of the size 1, 3, and 5, respectively. The result shown in Figure 8-4 demonstrates that the median filter is better at noise removal than the Gaussian filter.



Figure 8-4. Median filter for noise removal

High-Pass Filters

High-pass filters allow higher frequency information to pass through. In terms of image processing, the high-frequency components include edges and boundaries in an image. Thus, applying high-pass filters results in the edge highlighting and detection in an image. Let's look at a simple high-pass filter, the *prewitt* filter. A code example is shown in Listing 8-6.

Listing 8-6. prog06.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt

img = misc.ascent()

filtered = ndi.prewitt(img)

output = [img, filtered]
titles = ['Original', 'Filtered']
```

```

for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The application of this filter results in highlighted edges, as shown in Figure 8-5.

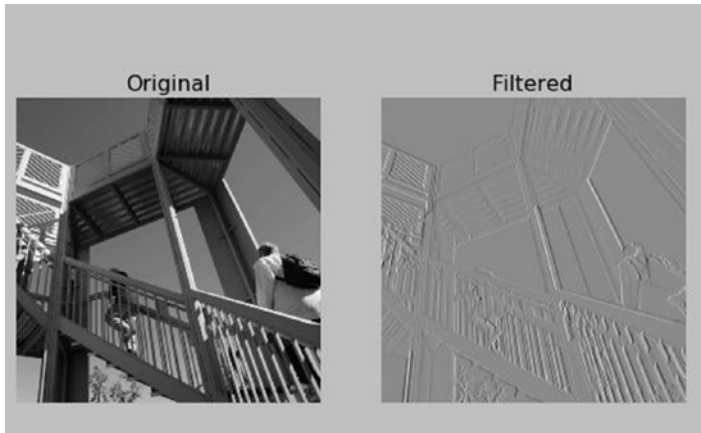


Figure 8-5. Prewitt filter output

There is a better high-pass filter, called the *Sobel* filter. As shown in Listing 8-7, it provides the horizontal and vertical edges separately.

Listing 8-7. prog07.py

```

import numpy as np
import scipy.ndimage as ndi
import matplotlib.pyplot as plt

img = np.zeros((516, 516))

img[128:-128, 128:-128] = 1

img = ndi.gaussian_filter(img, 8)

rotated = ndi.rotate(img, -20)

noisy = rotated + 0.09 * np.random.random(rotated.shape)

sx = ndi.sobel(noisy, axis=0)
sy = ndi.sobel(noisy, axis=1)
sob = np.hypot(sx, sy)

```

```

titles = ['Original', 'Rotated', 'Noisy',
          'Sobel (X-axis)', 'Sobel (Y-axis)', 'Sobel']

output = [img, rotated, noisy, sx, sy, sob]

for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(output[i])
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The code in Listing 8-7 generates a black square of dimensions 516x516 using the `np.zeros()` method. Then, the next line sets the intensity values of a 256x256 square within the generated square to white. Then it blurs and rotates the generated image. You then use this image as the source for applying the Sobel filters. First, you calculate the Sobel filter for the x axis, and then you calculate it for the y axis. `np.hypot()` is for calculation of the hypotenuse of a triangle. Passing the Sobel for the x and y axes to it gives you the combined results. When you calculate the Sobel for the x axis, it highlights the horizontal edges. When you calculate the Sobel for the y axis, it highlights the vertical edges. Combining the results returns the highlighted edges, as shown in Figure 8-6.

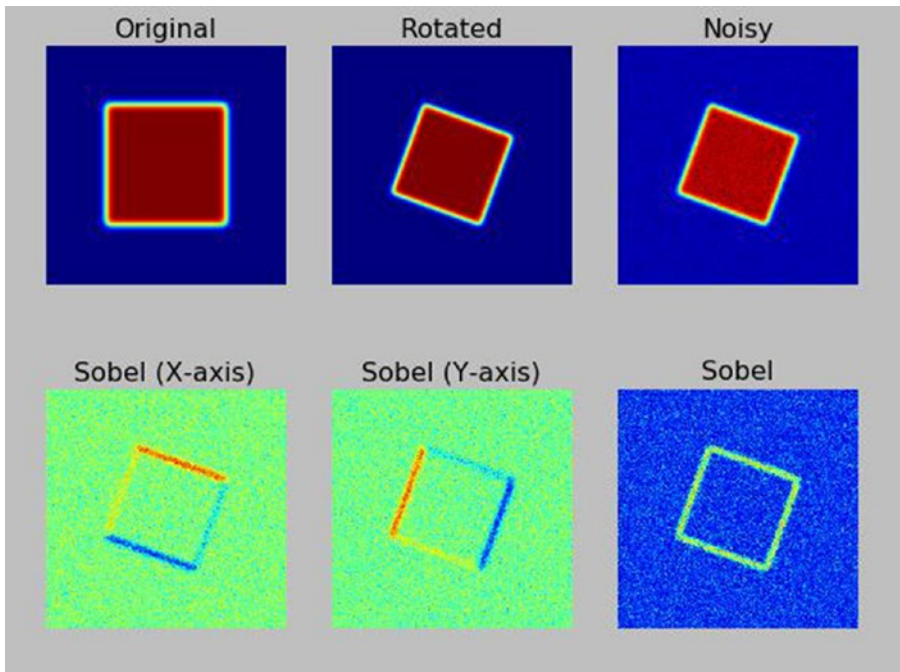


Figure 8-6. Sobel filter demo

Fourier Filters

The Fourier filter works on the frequency component of the signal. Fourier filters apply transformations in the frequency domain of the images. A Fourier filter first computes the Fourier transform of the image signals and then applies the required filtering function on the calculated Fourier transform of the image. Finally, it takes the inverse Fourier transform of the resultant output. Thus, it gives you the output on the frequency domain. It is used for a wide variety of image and signal processing tasks, including image filtering and reconstruction. Listing 8-8 shows an example of various Fourier transforms applied to an image.

Listing 8-8. prog08.py

```
import scipy.ndimage as ndi
import scipy.misc as misc
import matplotlib.pyplot as plt
import numpy as np

img = misc.ascent()

noisy = img + 0.09 * img.std() * np.random.random(img.shape)

fe = ndi.fourier_ellipsoid(img, 1)
fg = ndi.fourier_gaussian(img, 1)
fs = ndi.fourier_shift(img, 1)
fu = ndi.fourier_uniform(img, 1)

titles = ['Original', 'Noisy',
          'Fourier Ellipsoid', 'Fourier Gaussian',
          'Fourier Shift', 'Fourier Uniform']

output = [img, noisy, fe, fg, fs, fu]

for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(np.float64(output[i]), cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()
```

The result of the example in Listing 8-8 is shown in Figure 8-7.

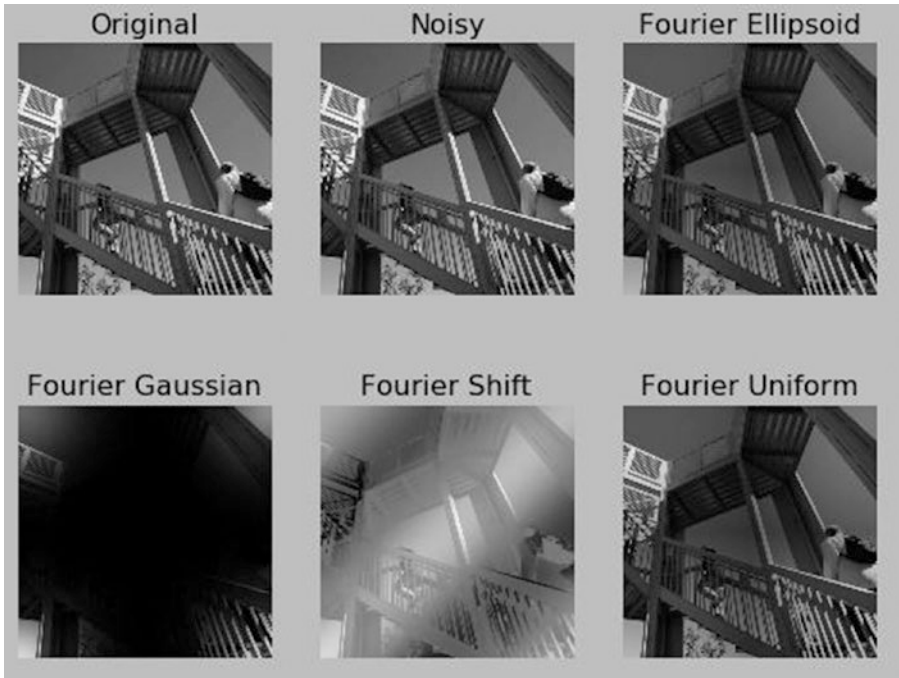


Figure 8-7. *Fourier filter demo*

■ **Note** Fourier filters are a vast area and it's difficult to cover them in a single chapter or section. You can find more information on Fourier filters at <https://terpconnect.umd.edu/~toh/spectrum/FourierFilter.html> and <http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm>.

EXERCISE

This chapter introduced you to the concepts very briefly. However, image filtering and its applications are very complicated areas. Complete the following exercises to gain a better understanding of filters and their implementation in SciPy.

- Explore more filters in `ndimage`. You can find the documentation on <https://docs.scipy.org/doc/scipy-0.18.1/reference/ndimage.html>.
 - Try different kernels on the convolution operation.
 - Visit all the reference URLs mentioned in the chapter and explore more about the kernels and Fourier filters.
 - Change the Sobel filter code example to show the results in grayscale using `cmap='gray'`.
-

Conclusion

In this chapter, you were introduced to the filters. You saw their types and looked at their applications according to the types. The image filtering area is too vast to be completely explored in a single chapter. Follow-up exercises will help you understand filtering better.

In the next chapter, we conclude the book with morphological operators, image thresholding, and basic segmentation.

CHAPTER 9



Morphology, Thresholding, and Segmentation

In the previous chapter, you studied the theory behind image filters, along with the types and practical applications of filters in enhancing images.

This is the last chapter in the book and a bit detailed one. In this chapter, you are going to study important concepts in image processing like morphology, morphological operations on images, thresholding, and segmentation. The chapter starts with the distance transform operation and then moves to the basics of morphology and structuring elements. You will see plenty of examples of morphological operations. We will then wrap up the chapter with thresholding and segmentation.

Distance Transforms

A *distance transform* is an operation on binary images. Binary images have background elements (zero value - black color) and foreground elements (white color). Distance transform replaces each foreground element with the value of the shortest distance to the background. `scipy.ndimage` has three methods for calculating the distance transform of a binary image. The code in Listing 9-1 illustrates how a distance transform can be used practically to generate test images.

Listing 9-1. prog01.py

```
import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np

img = np.zeros((32, 32))
img[8:-8, 8:-8] = 1

print(img)

dist1 = ndi.distance_transform_bf(img)
dist2 = ndi.distance_transform_cdt(img)
dist3 = ndi.distance_transform_edt(img)
```

```

output = [img, dist1, dist2, dist3]
titles = ['Original', 'Brute Force', 'Chamfer', 'Euclidean']

for i in range(4):
    print(output[i])
    plt.subplot(2, 2, i+1)
    plt.imshow(output[i], interpolation='nearest', cmap='spectral')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The code shown in Listing 9-1 calculates the distance transform by the brute force algorithm, Chamfer type algorithm, and Euclidean methods, respectively. You are going to use distance transforms in generating the test images in this chapter. The output is shown in Figure 9-1.

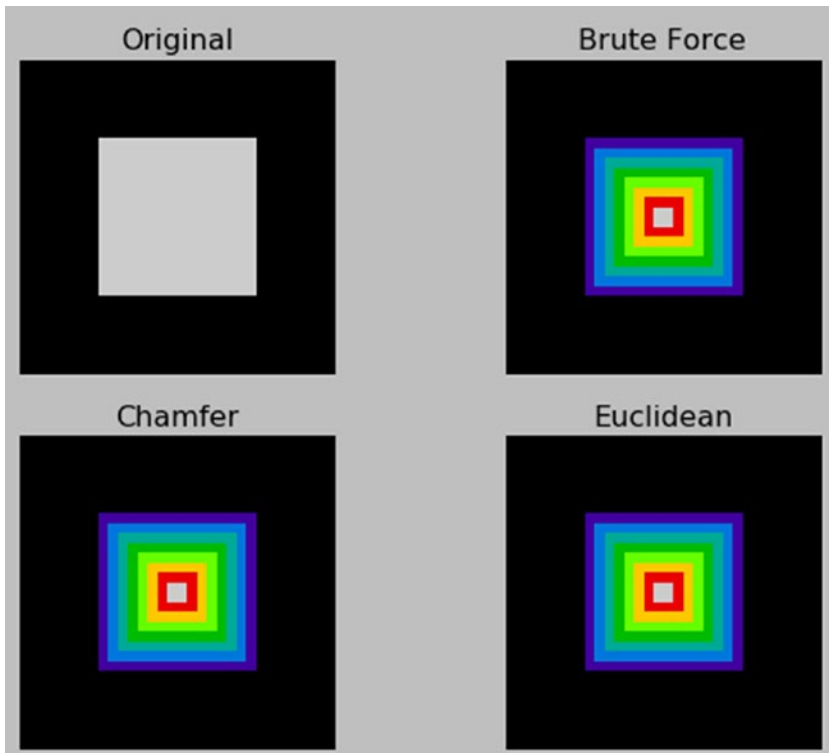


Figure 9-1. Distance transforms demo

Morphology and Morphological Operations

Morphology is the study of shapes and forms. The morphological study of images deals with shapes rather than the values of the pixels. Morphological operations are usually performed on binary images. Let's take a look at a few concepts related to the morphological operations.

Structuring Element

A structuring element is a matrix that's used to interact with a given binary image. It comes in various shapes, like a ball or a ring or a line. It can come in many shapes, like a 3x3 or 7x7 matrix. Bigger size structuring elements take more time for computation. A simple structuring element can be defined as a unity matrix of odd sizes. `np.ones((3,3))` is an example of this.

Various Morphological Operations

Let's briefly look at various morphological operations. Dilation causes the expansion of the shapes in an input image. Erosion causes the shrinkage in the shape in an input image. Opening is dilation of erosion. Closing is erosion of dilation.

It is difficult to understand these concepts just by reading about them. Take a look at the example in Listing 9-2, which demonstrates these concepts.

Listing 9-2. prog02.py

```
import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np

img = np.zeros((16, 16))
img[4:-4, 4:-4] = 1

print(img)

erosion = ndi.binary_erosion(img).astype(img.dtype)
dilation = ndi.binary_dilation(img).astype(img.dtype)
opening = ndi.binary_opening(img).astype(img.dtype)
closing = ndi.binary_closing(img).astype(img.dtype)

output = [img, erosion, dilation, opening, closing]
titles = ['Original', 'Erosion', 'Dilation', 'Opening', 'Closing']
```

```

for i in range(5):
    print(output[i])
    plt.subplot(1, 5, i+1)
    plt.imshow(output[i], interpolation='nearest', cmap='spectral')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The code example in Listing 9-2 generates a binary image and applies all the binary morphological operations to it. The output is shown in Figure 9-2.

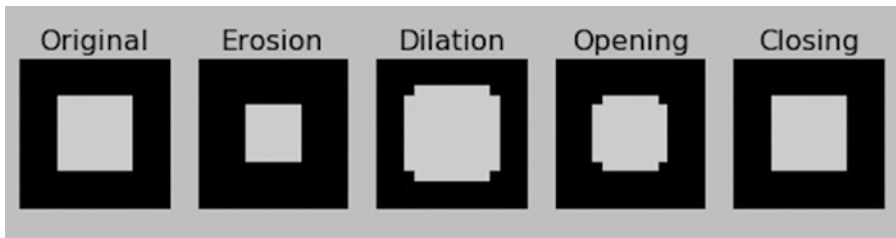


Figure 9-2. Morphological operations demo

Another important operation is `binary_fill_holes()`. It is used to fill the gaps in the binary image, as shown in Listing 9-3.

Listing 9-3. prog03.py

```

import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np

img = np.ones((32, 32))
x, y = (32*np.random.random((2, 20))).astype(np.int)
img[x, y] = 0

noise_removed = ndi.binary_fill_holes(img).astype(int)

output = [img, noise_removed]
titles = ['Original', 'Noise Removed']

for i in range(2):
    print(output[i])
    plt.subplot(1, 2, i+1)
    plt.imshow(output[i], interpolation='nearest', cmap='spectral')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The code in Listing 9-3 generates a 32x32 square matrix image with all the values as 1 (white value). Then it randomly sets a few pixels to 0 (the dark value). The dark pixels can be considered holes in the image, which can then be removed using `binary_fill_holes()`.

The output is shown in Figure 9-3.

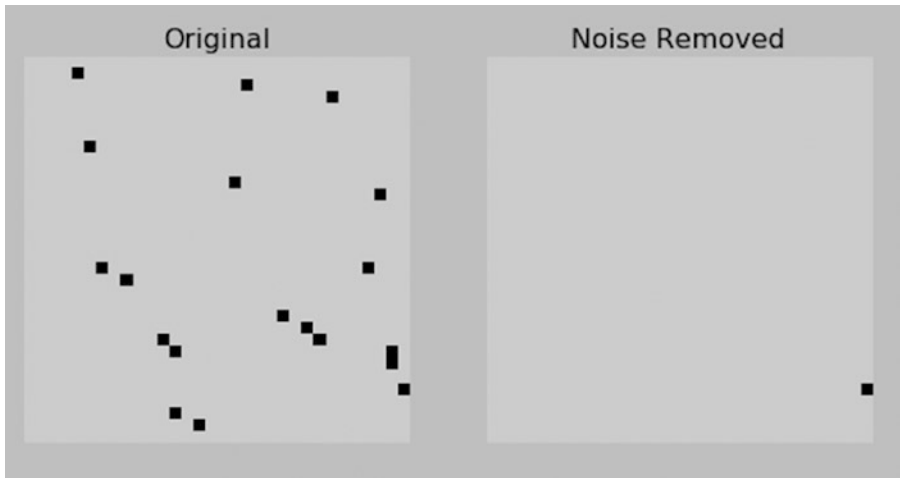


Figure 9-3. The `binary_fill_holes()` demo

Grayscale Morphological Operations

There is a set of grayscale morphological operations. The code shown in Listing 9-4 generates eight random values and assigns them to pixels on a completely dark (zero value) background. Then it uses the `grey_dilation()` method to dilate them.

Listing 9-4. prog04.py

```
import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np

img = np.zeros((64, 64))
x, y = (63*np.random.random((2, 8))).astype(np.int)
img[x, y] = np.arange(8)

dilation = ndi.grey_dilation(img, size=(5, 5),
                             structure=np.ones((5, 5)))

output = [img, dilation]
titles = ['Original', 'Dilation']
```



```

for i in range(2):
    print(output[i])
    plt.subplot(1, 2, i+1)
    plt.imshow(output[i], interpolation='nearest', cmap='spectral')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The output is shown in Figure 9-4.

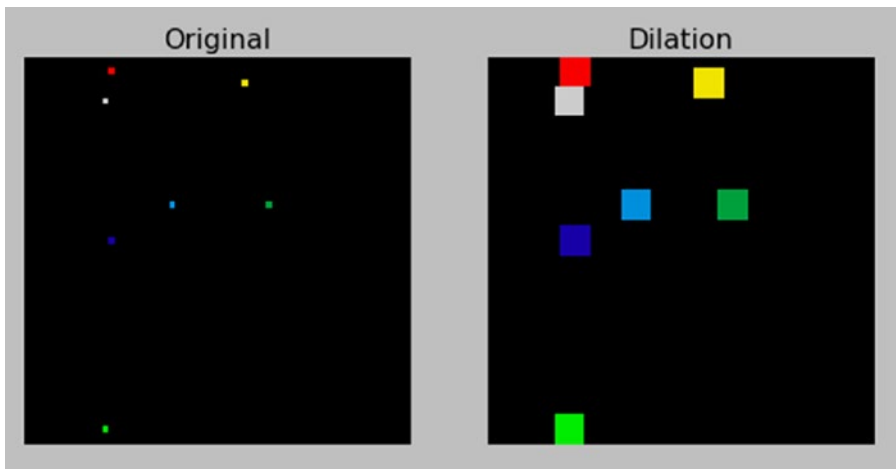


Figure 9-4. Gray dilation demo

You are using a structuring element that's 5x5 here.

■ **Note** This example uses the `random()` method. When you execute the code in the bundle, the output won't be exactly the same.

The code shown in Listing 9-5 applies gray dilation and gray erosion operations to a distance transform.

Listing 9-5. prog05.py

```

import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np

img = np.zeros((16, 16))
img[4:-4, 4:-4] = 1

```

```

img = ndi.distance_transform_bf(img)

dilation = ndi.grey_dilation(img, size=(3, 3),
                             structure=np.ones((3, 3)))

erosion = ndi.grey_erosion(img, size=(3, 3),
                           structure=np.ones((3, 3)))

output = [img, dilation, erosion]
titles = ['Original', 'Dilation', 'Erosion']

for i in range(3):
    print(output[i])
    plt.subplot(1, 3, i+1)
    plt.imshow(output[i], interpolation='nearest', cmap='spectral')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

```

The code in Listing 9-5 uses a structuring element that's 3x3 for both operations. The output is shown in Figure 9-5.

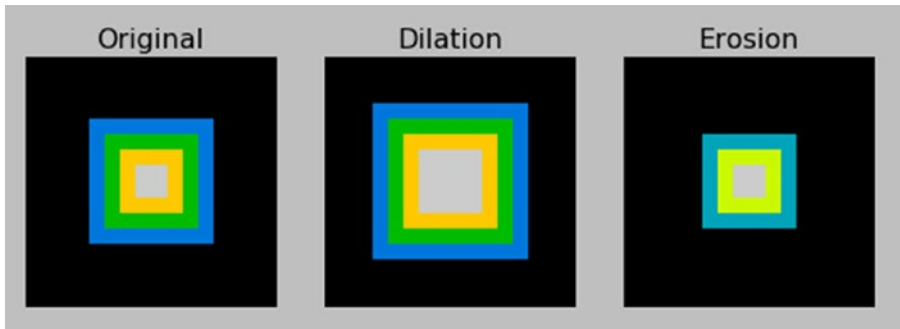


Figure 9-5. Dilation and erosion on a distance transform

Thresholding and Segmentation

This is the final part of the book and it deals with one of the most important applications of image processing, *segmentation*. In thresholding operations, you convert grayscale images to binary (black-and-white) images based on the threshold value. The pixels with intensity values greater than the threshold are assigned white and the pixels with an intensity value lower than the threshold are assigned a dark value. This is known as *binary thresholding* and it's the most basic form of thresholding and segmentation. An example is shown in Listing 9-6.

Listing 9-6. prog06.py

```
import matplotlib.pyplot as plt
import scipy.misc as misc

img = misc.ascent()

thresh = img > 127

output = [img, thresh]
titles = ['Original', 'Thresholding']

for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')

plt.show()
```

The code in Listing 9-6 sets the threshold to 127. At the grayscale, a pixel value of 127 corresponds to the gray color. The resultant thresholded image is shown in Figure 9-6.

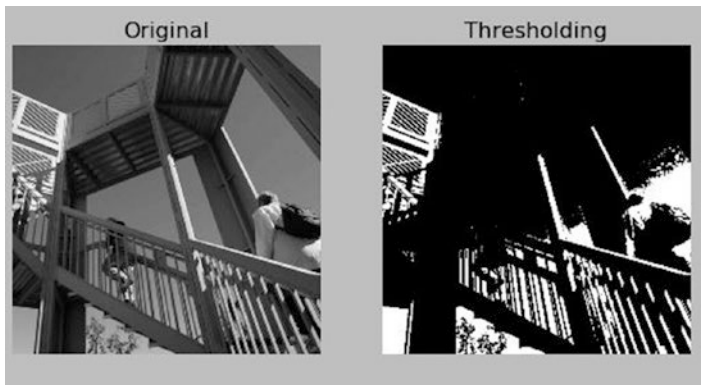


Figure 9-6. Binary thresholding

Thresholding is the most basic type of image segmentation. Image segmentation refers to dividing an image into many regions based on some property, like colors of pixels, connectivity of the region, etc. You can get the better segments of an image by applying the morphological operations on a thresholded image (see Listing 9-7).

Listing 9-7. prog07.py

```

import matplotlib.pyplot as plt
import scipy.misc as misc
import scipy.ndimage as ndi
import numpy as np

img = misc.ascent()

thresh = img > 127

dilated = ndi.binary_dilation(thresh, structure=np.ones((9, 9))).astype(int)
eroded = ndi.binary_erosion(dilated, structure=np.ones((9, 9))).astype(int)

output = [img, thresh, dilated, eroded]
titles = ['Original', 'Thresholding', 'Dilated', 'Eroded and Segmented']

for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')

plt.show()

```

The output is shown in Figure 9-7.

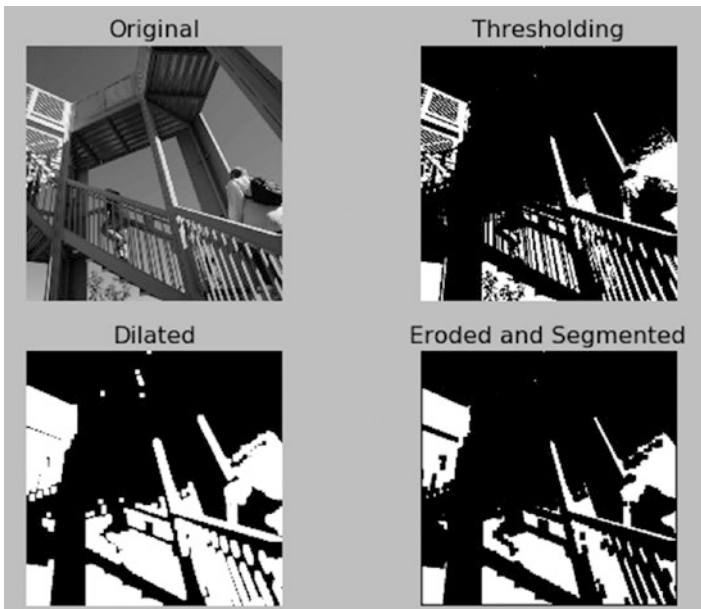


Figure 9-7. Thresholded and segmented image

Finally, here is yet another real-world application of segmentation, the processing of printed documents. The example in Listing 9-8 shows processed images of printed documents.

Listing 9-8. prog08.py

```
import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np

img = ndi.imread('/home/pi/DIP/Dataset/5.1.13.tiff')

thresh = img > 127

output = [img, thresh]
titles = ['Original', 'Thresholding']

for i in range(2):
    plt.subplot(1, 2, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()
```

The output is shown in Figure 9-8.

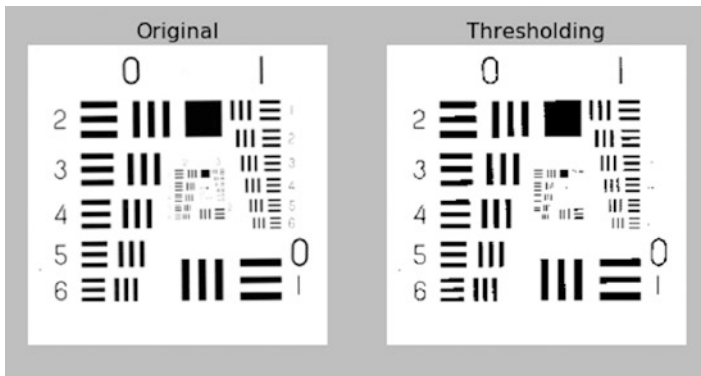


Figure 9-8. Thresholding a printed document

Conclusion

In this chapter, you studied distance transforms for generating test images. You then learned morphology and morphological operations on images. Morphological operations come in two varieties—binary and grayscale. You also studied thresholding, which is the simplest form of image segmentation. Finally, you studied how you can use thresholding on images in a document.

Book Summary

This is the final chapter of the book. You started with single board computers and Raspberry Pi. Then you learned how to set up the Pi and acquire images with various types of camera sensors. You took your first steps in the world of image processing with the help of Pillow. You also learned to create an interactive GUI for the image processing demos using Tkinter. Then you got started with scientific computing and the SciPy toolkit. You learned the basics of NumPy ndarrays and matplotlib for visualization. Finally, you learned to use `scipy.misc` and `scipy.ndimage` to process images.

I hope you enjoyed reading this book and following the examples as much as I enjoyed writing it and programming the examples.

What's Next

This is not an end but the mere beginning into the amazing world of image processing and computer vision. From here, you can follow various libraries for the image processing and data visualization. More advanced libraries for image processing include `scikit-image`, `OpenCV`, and `Mahotas`. You can explore these libraries. Consider also exploring NumPy and other aspects of scientific computing like signal processing and data visualization. The SciPy stack has suitable modules for all these applications. Happy exploring and Pythoning!

Index

■ A

add() method, 59
Analog image processing system, 37

■ B

Binary thresholding, 117–118
blend() method, 53

■ C

Channel images
 conversion, 91
 np.dstack() method, 91
 process, 89
 separate, 90
Color quantization, 76–77
config.txt file, 12, 18
convert() method, 53
copy() method, 58
crop() method, 57

■ D

darker() method, 59
difference() method, 59
Digital image processing (DIP), 36–37
 image processing systems, 37–38
 Raspberry Pi and Python, 38
 signal processing, 36
Distance transforms, 111–112

■ E

Ethernet
 dynamic IP address, 22
 static IP address, 21

■ F

Filtering operation
 convolution operation, 99
 local filtering, 99
 low-pass (*see* Low-pass filters)
Fourier filters, 108

■ G

getpixel() method, 58
grey_dilation() method, 115

■ H

High-pass filters, 105–107
histogram() method, 96
Histograms and
 equalization, 77–79

■ I, J, K

ImageChops module, 59
ImageEnhance module, 74–76
ImageFilter module, 65
 blur operation, 66
 custom filter, 66
 Digital Unsharp Mask, 69–70
 GaussianBlur() method, 67
 GaussianBlur() blur demo, 68
 kernel, 68–69
 low-pass filtering, 68
 max filter, 72
 median filter, 72
 min filter, 72
 mode filter, 73–74
 output, 68, 71
 Tkinter, 67

- ImageOps.equalize() method, 78
- ImageOps module, 60
- Image processing system
 - advanced, 79
 - analog, 37
 - color quantization, 76–77
 - digital devices, 37
 - histograms and equalization, 77–79
 - ImageEnhance module, 74–76
 - ImageFilter module, 65–74
- Image sources
 - book code and image datasets, 41
 - directory structure, 42
 - gview command, 43
 - Internet, 41
 - lsusb command, 43
 - Pi camera module, 44
 - test.jpg file, 44
 - Webcam, 42
- imfilter() method, 83
- imread() method, 93
- imshow() method, 88
- Integrated Development Environment (IDE), 33
 - Geany
 - Raspbian menu, 35
 - set build commands window, 36
 - text editor, 34
 - IDLE, 33

■ L

- Low-pass filters, 68
 - blurring, 100
 - Gaussian filter, 101
 - median filter, 105
 - remove noise, 102
 - uniform filter, 102

■ M

- Matplotlib, 86
 - image grid, 89
 - Lena image, 88
 - pyplot demo, 87
- Measurements
 - extrema() method, 96–97
 - histogram() method, 95–96
 - intensity values, 98
 - minimum() and maximum() methods, 96

- merge() method, 51
- misc.imshow() method, 86
- Modules, 51
 - blending, 53
 - copying and saving images, 58
 - crop and paste operations, 57
 - ImageChops, 59
 - ImageOps, 60
 - mode conversion, 53
 - particular pixel, 58
 - resize, 55
 - rotate() method, 56
 - Solarize operation, 62
 - split() and merge() methods, 51
- Morphological operations
 - binary_fill_holes(), 115
 - binary image, 114
 - concepts, 113
 - demo, 114
 - dilation and erosion, 117
 - erosion operations, 116
 - gray dilation demo, 116
 - grayscale, 115
 - structuring element, 113

■ N, O

- NumPy modules
 - features, 84
 - homepage, 84
 - library, 84
 - N-dimensional array, 85
 - RGB value, 86

■ P

- paste() method, 57
- Pi camera module, 44
- PIL image objects and NumPyndarrays, 91
- Power supply unit (PSU), 7
- Prewitt filter, 106
- Printed circuit board (PCB), 1
- Python
 - community support, 29
 - easy to learn, 27
 - easy to maintain, 27
 - easy to read, 27
 - extensible, 28
 - extensive libraries, 28
 - features of, 26
 - high-level language, 27

- history of, 25
- IDEs (*see* Integrated Development Environment (IDE))
- interactive mode, 32
- interpreted language, 28
- memory management, 29
- normal mode, 32
- object-oriented
 - programming, 28
- open source project, 27
- portable, 27
- powerful, 29
- principles, 26
- Python 3
 - differences, 30
 - overview, 29
 - Raspbian, 31
 - use of, 31
- rapid prototyping tool, 29
- Robust, 28
- simple language, 26
- Python Imaging Library (PIL)
 - digital image processing, 46
 - images
 - Linux command, 47
 - properties, 49
 - show() function, 48
 - Tkinter output, 49
 - version number, 47

■ Q

- quantize() method, 76

■ R

- random() method, 103
- Rapid prototyping tool, 29
- Raspberry Pi, 4
 - booting up
 - desktop, 14
 - quad-core processor, 14
 - single core processor, 13
 - steps, 13
 - bottom view, 5
 - components, 4
 - config.txt file, 12, 18
 - Ethernet
 - dynamic IP address, 22
 - static IP address, 21

- foundation of, 4
- free software
 - accelerator plus, 10
 - Raspbian OS image, 10
 - Win32 Disk Imager setup, 10
 - WinZip/WinRaR, 10
- hardware requirement
 - card reader, 7
 - computer/laptop, 6
 - I/O devices, 6
 - microSD card, 6
 - monitor, 8
 - power supply unit, 7
 - setup, 6
- LXTerminal configuration
 - boot options, 16
 - desktop autologin, 16
 - icon, 15
 - internationalization options, 17
 - main screen, 17
 - raspi-config utilities, 16
 - reboot prompt, 18
 - window, 15
- microSD card, 9
- operating system, 18
- Raspbian OS image
 - message, 12
 - microSD card, 10
 - overwrite warning message, 11
 - protection error message, 11
 - Win32 Disk Imager, 11
- shut down Pi, 24
- specifications, 4
- update
 - Firmware, 22
 - raspi-config, 23
 - upgrade Raspbian, 22
- VGA monitor, 12
- WiFi, 19
- resize() function, 55
- Robust, 28
- rotate() method, 56

■ S

- save() method, 58
- Scientific Python (SciPy), 81
 - channel image, 89
 - components, 81
 - installation, 82

Scientific Python (SciPy) (*cont.*)

Matplotlib, 86

image grid, 89

Lena image, 88

pyplot demo, 87

NumPy (*see* NumPy modules)

simple image

processing, 83

scipy.misc module, 82

stack, 81

screen() method, 60

Segmentation, 118

shift() method, 93

Signal processing system, 36

Single board computers (SBCs)

differences, 2

families, 3

form factors, 2

history of, 3

overview, 1

regular computers, 2

vs. regular CPU, 3

system on a chip, 2

Sobel filter, 107

solarize() method, 62

split() method, 51

subplot() method, 89

subtract() method, 60

System on Chips (SoCs), 2

vs. regular CPU, 3

■ **T, U, V**

Thresholding and segmentation
operations

binary thresholding, 117–118

printed document, 120

real-world application, 120

segmentation, 118

title() method, 88

Transformations, 93

shift() method, 93

zoom() method, 94

■ **W, X, Y**

Webcam, 42

WiFi, 19

■ **Z**

zoom() method, 94