

INSTANT

Short | Fast | Focused

OpenCV Starter

Get started with OpenCV using practical, hands-on projects

Jayneil Dalal

Sohil Patel

[PACKT]
PUBLISHING

Instant OpenCV Starter

Get started with OpenCV using practical, hands-on projects

Jayneil Dalal

Sohil Patel



BIRMINGHAM - MUMBAI

Instant OpenCV Starter

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: May 2013

Production Reference: 1170513

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

ISBN 978-1-78216-881-2

www.packtpub.com

Credits

Authors

Jayneil Dalal

Sohil Patel

Reviewers

David Millán Escrivá

Sagar Patel

Acquisition Editor

Usha Iyer

Commissioning Editor

Sharvari Tawde

Technical Editor

Vrinda Nitesh Bhosale

Project Coordinator

Esha Thakker

Proofreader

Paul Hindle

Graphics

Abhinash Sahu

Production Coordinator

Prachali Bhiwandkar

Cover Work

Prachali Bhiwandkar

Cover Image

Conidon Miranda

About the Authors

Jayneil Dalal is an Embedded Linux Engineer/Technical Writer who loves to explore different open source technologies, and he has been a key member of the PandaBoard.org and Beagleboard.org projects at Texas Instruments. He has been selected to give a talk at Linuxfest Northwest 2013 and OSCON 2013. He has previously presented at Linuxcon North America 2012, Drodicon 2012 in Berlin, Southeast Linuxfest 2012, Indiana Linuxfest 2012, Northwest Linuxfest 2012, Scipy 2011, and Opensource bridge 2012. Jayneil can be reached at jayneil.dalal@gmail.com.

I would like to thank my family members, Harish, Atul, Deena, and Nayshil, for encouraging me to write this book. Finally, I would like to acknowledge my co-author Sohil Patel's immense hardwork and dedication that made this book a reality.

Sohil Patel is a Computer Vision Engineer and FOSS advocate who loves to explore different open source technologies. His areas of interest include Python, Computer Vision, Augmented Reality, Linux, HCI, Pandaboard, and other open source hardware platforms. He is currently working at Azoï, which is a startup based in India.

I would like to thank my family members, Sanjay, Hema, and Hetvi, for encouraging me to write this book.

About the Reviewers

David Millán Escrivá was 8 years old when he wrote his first program on an 8086 PC with Basic language, which enabled the 2D plotting of basic equations. This was only the beginning of his computer development relationship, and he has since created many applications and games.

In 2005, he finished his studies in IT through the Universitat Politècnica de Valencia with honors in human-computer interaction supported by computer vision with OpenCV (vo.g6). He wrote a final project based on this subject and published it on HCI Spanish congress.

He also participated in Blender source code, an open source 3D-software project, and worked in his first commercial movie, Plumíferos – Aventuras Voladoras, as a Computer Graphics Software Developer.

David now has more than 10 years of experience in IT, with more than 7 years experience in computer vision, computer graphics, and pattern recognition working on different projects and startups and applying his knowledge of computer vision, optical character recognition, and Augmented Reality.

He is the author of DamilesBlog (<http://blog.damiles.com>), where he publishes research articles and tutorials about OpenCV, computer vision in general, and Optical Character Recognition algorithms. He is also the co-author of *Mastering OpenCV with Practical Computer Vision Projects*. He has also reviewed the following Packt books:

- ◆ *GnuPlot Cookbook*, Lee Phillips
- ◆ *OpenCV Computer Vision with Python*, Joseph Howse

I thank my wife, Izaskun, and my daughter, Eider, for their patience and support. Love you.

Sagar Patel is pursuing a Bachelors degree in Electronics and Communication Engineering from Nirma University, India. His main areas of interest include Digital Sound Processing, Augmented Reality, Artificial Intelligence, and Machine Vision. His research is on the bionics based project The Third Eye, which has been approved by Idea Lab. As far as physics is concerned, he loves to explore the idea of the *Theory of Relativity*. He believes that, "Thousands of miles of long journey begin with a single step, and that step begins with a tiny thought".

I am thankful to the authors, Jayneil Dalal and Sohil Patel, for giving me the opportunity to review this book. I would like to specially thank my parents and my grandfather for giving me the motivation. I am also thankful to Packt Publishing for providing the direction to the reviewing process.

www.packtpub.com

Support files, eBooks, discount offers and more

You might want to visit www.packtpub.com for support files and downloads related to your book.

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

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

packtlib.packtpub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ◆ Fully searchable across every book published by Packt
- ◆ Copy and paste, print and bookmark content
- ◆ On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.packtpub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.



Table of Contents

Instant OpenCV Starter	1
So, what is OpenCV?	3
Installation	4
OpenCV installation procedure for Linux	4
Approach 1	4
Approach 2	4
OpenCV uninstallation procedure for Linux	11
OpenCV installation procedure for Windows	12
Step 1 – installing the dependencies	12
Step 2 – installing OpenCV	12
Step 3 – configuring OpenCV with Code::Blocks	12
And that's it	15
Quick start – OpenCV fundamentals	16
Task 1 – image basics	17
Greyscale	17
Color/RGB	18
Task 2 – reading and displaying an image	19
Code	19
Code explanation	19
Output	21
Task 3 – resizing and saving an image	21
Code	21
Code explanation	22
Output	23
Top 5 features you need to know about	24
Pixel manipulation	24
Task	24
Algorithm	24
Code	25
Code explanation	26
Output	28

Table of Contents

Image conversions	28
Task	28
Code	28
Code explanation	29
Output	30
Image steganography	30
Part 1 – encode	31
Code explanation	32
Part 2 – Decode	34
Code explanation	35
Edge detection	36
Task	36
Code	37
Code explanation	37
Output	38
Real-time video processing via webcam	39
Task	39
Algorithm	39
Code	39
Code explanation	40
Output	40
People and places you should get to know	41
Official sites	41
Tutorials/cheat sheets/answers	41
Community	41
Twitter	41

Instant OpenCV Starter

Welcome to *Instant OpenCV Starter*. This book has been specifically created to provide you with all the information that you need to set up OpenCV. You will learn the basics of OpenCV, get started with building your first program, and discover some tips and tricks for using OpenCV.

This book contains the following sections:

So, what is OpenCV? find out what OpenCV actually is, what you can do with it, and why it's so great.

Installation learn how to download and install OpenCV with minimum fuss and then set it up so that you can use it as soon as possible.

Quick start – OpenCV fundamentals this section will show you how to perform a few of the basic tasks in OpenCV as well as how to write your first program.

Top 5 features you need to know about here you will learn how to perform image conversions and pixel manipulations.

People and places you should get to know every open source project is centered around a community. This section provides you with many useful links to the OpenCV project page and forums.

So, what is OpenCV?

OpenCV is the world's most popular open source computer vision library, with more than 500 optimized algorithms for image and video analysis. In the digital age of image and video sharing, the need for computer vision is at an all-time high. Take a look around you, and you will see that computer vision is being implemented everywhere. It's in cars to assist drivers with parking in tight spots; most manufacturers ship laptops these days with facial recognition software for additional security; even Facebook and Google+ use it to identify individual persons in the large photo albums we upload so that we don't have to tag each person multiple times in every single photograph. The list is endless. This is where OpenCV comes in to the picture. You can use the wide range of image and video algorithms provided by the OpenCV library for your particular computer vision application. It saves time and energy by providing you a tested, well-known reference platform to start so that you don't end up writing everything from scratch.

OpenCV is distributed with a BSD license, which means that you can make a commercial application without revealing your source code. But, there are a few algorithms, despite being provided with complete source code inside OpenCV, that are patented.

OpenCV has C++, C, Python, and Java interfaces, and it supports Windows, Linux, Mac OS, iOS, and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multicore processing.

The URL of the project's website is as follows:

<http://opencv.org/>

Installation

This section will cover the installation procedure for OpenCV in Windows as well as Linux. Newer versions of the OpenCV library are released periodically. For the purpose of this book, the 2.4.2 version has been the reference. Also, we will be using the open source Code::Blocks **Integrated Development Environment (IDE)** for writing all our codes. More information about this can be found at <http://www.codeblocks.org/>.

OpenCV installation procedure for Linux

We have used Ubuntu 12.04 LTS for this installation. Before you install OpenCV, you will need to check that you have all of the required elements, listed as follows:

- ◆ **Disk space:** 300 MB free (min). You will require more free space to store your teaching materials.
- ◆ **Memory:** 256 MB (min); 1 GB (recommended).

Approach 1

If you are not interested in installing the latest version of OpenCV, you can install it from the Ubuntu software repositories by running the following command in the terminal (by pressing *Ctrl + Alt + T*):

```
sudo apt-get install libopencv-dev
```

Approach 2

Use this approach if you want the latest version of OpenCV. In this approach, OpenCV will be built from source and you will have to install the dependencies prior to that.

Step 1 – updating the system

Make sure that your system is updated. To update your system, run the following commands in the terminal:

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

Step 2 – installing the dependencies

OpenCV is a library, and it needs various other components to function properly. So, the various other dependencies that we would need to install first are as follows:

- ◆ **Essentials:** These are the libraries and tools required by OpenCV. Use the following command to install essentials:

```
sudo apt-get install build-essential checkinstall cmake pkg-config yasm
```

- ◆ **Image I/O:** These are libraries for reading and writing various image types. If you do not install them, then the versions supplied by OpenCV will be used.
`sudo apt-get install libtiff4-dev libjpeg-dev libjasper-dev`
- ◆ **Video I/O:** You need some or all of these packages to add video capturing/encoding/decoding capabilities to the highgui module.
`sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev libxine-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev`
- ◆ **Python:** These are packages needed to build the Python wrappers.
`sudo apt-get install python-dev python-numpy`
- ◆ **Other dependencies:**
`sudo apt-get install checkinstall gstreamer1.2-gst-plugins-base-0.10 gstreamer1.2-gst-plugins-base-0.10-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libxine-dev libxine1-bin libxml2-dev`
- ◆ **Other third-party libraries:** Install Intel TBB to enable parallel code in OpenCV.
`sudo apt-get install libtbb-dev`
- ◆ **GUI:** You can optionally install QT instead of the default GTK and later enable it in the configuration.
`sudo apt-get install libgtk2.0-dev libqt4-dev`

Step 3 - configuring OpenCV Version 2.4.2

1. Download OpenCV from the following URL:

<http://downloads.sourceforge.net/project/opencvlibrary/opencv-unix/2.4.2/OpenCV-2.4.2.tar.bz2>

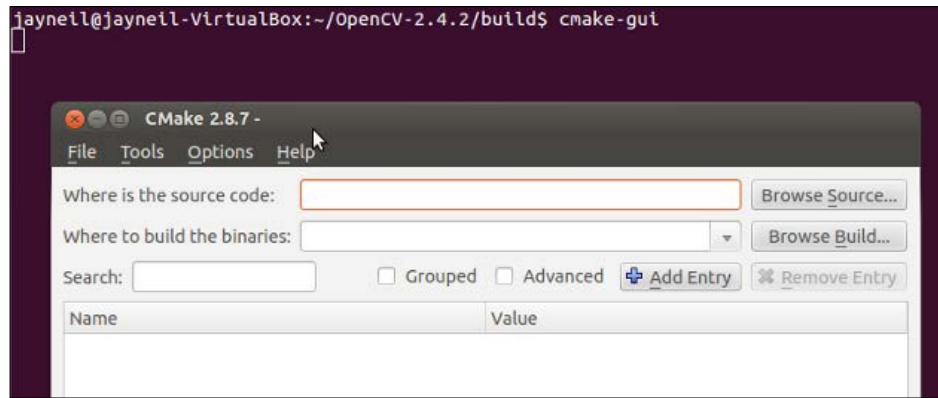
```
jayneil@jayneil-VirtualBox:~/OpenCV-2.4.2$ ls
3rdparty  apps      CMakeLists.txt  doc      index.rst  modules  samples
android   cmake    data            include  ios        README
jayneil@jayneil-VirtualBox:~/OpenCV-2.4.2$
```

2. Extract the downloaded file to your home folder and navigate to the extracted folder using the terminal.

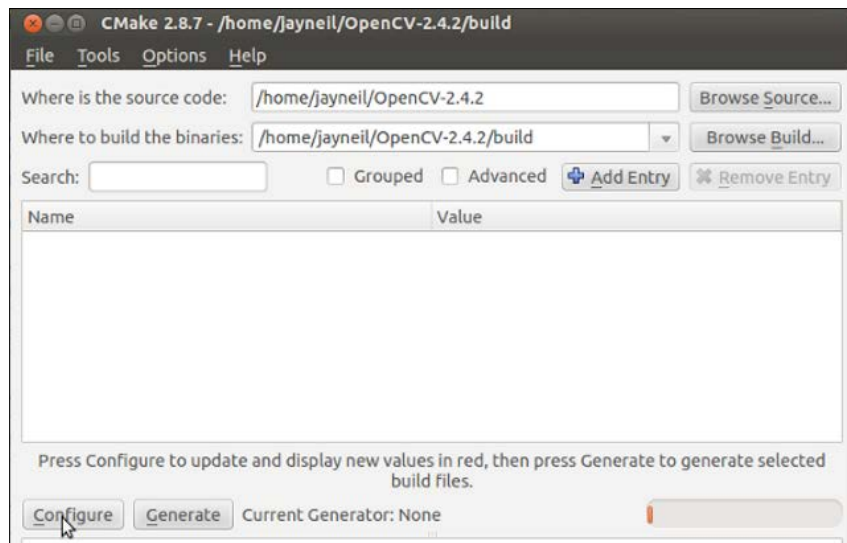
Instant OpenCV Starter

3. Make a sub folder and name it `build`. Navigate to this folder using the terminal and run the following command:

```
sudo apt-get install cmake-gui  
cmake-gui
```

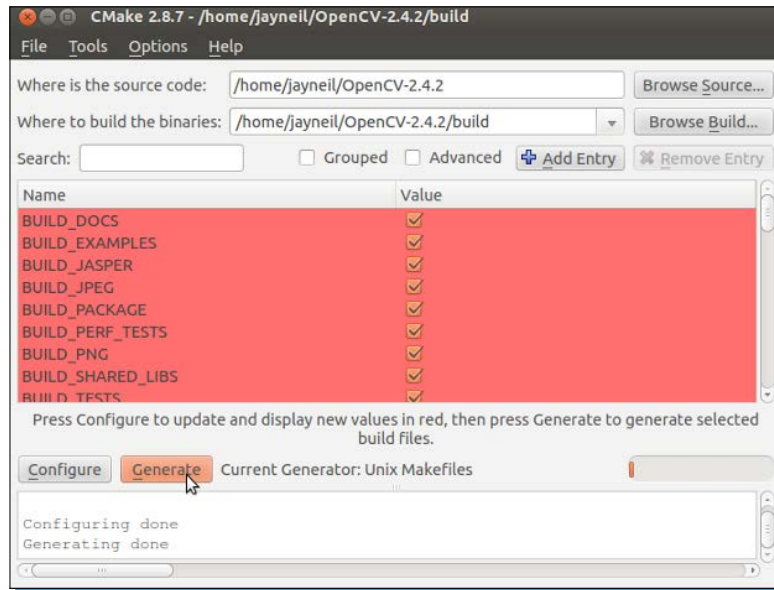


4. Provide the source folder and in the binary folder option, provide the `build` folder path.
5. Click on **Configure**.



6. Now check the checkboxes to include those functionalities and click on **Configure** again to update.

7. Once you are sure, click on **Generate**.



Step 4 – compiling OpenCV

Run the following command:

make

```
jayneil@jayneil-VirtualBox:~/OpenCV-2.4.2/build$ make
Scanning dependencies of target zlib
[ 0%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/adler32.c.o
[ 0%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/compress.c.o
[ 0%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/crc32.c.o
/home/jayneil/OpenCV-2.4.2/3rdparty/zlib/crc32.c: In function 'get_crc32':
/home/jayneil/OpenCV-2.4.2/3rdparty/zlib/crc32.c:218:5: warning: dereferencing pointer of incomplete type [-Wstrict-aliasing]
    218 |     *p = 0;
        |     ~~~~^
[ 0%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/deflate.c.o
[ 0%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/gzclose.c.o
[ 0%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/gzlib.c.o
[ 1%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/gzread.c.o
[ 1%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/gzwrite.c.o
[ 1%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/inflate.c.o
[ 1%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/inflate.c.o
[ 1%] Building C object 3rdparty/zlib/CMakeFiles/zlib.dir/inftrees.c.o
```

Instant OpenCV Starter

The above compilation process can take some time depending on your system configuration. It took approximately about 10 minutes on our Intel i7 laptop, while it took a bit more than an hour on our Intel i3 laptop. The following screenshot shows the output after the successful completion of the process.

```
type-punned pointer might break strict-aliasing rules [-Wstrict-aliasing]
Linking CXX executable ../../bin/stereo_multi_gpu
[100%] Built target example_gpu_stereo_multi
Scanning dependencies of target example_gpu_surf_keypoint_matcher
[100%] Building CXX object samples/gpu/CMakeFiles/example_gpu_surf_keypoint_matcher.cpp.o
In file included from /home/jayneil/OpenCV-2.4.2/modules/core/include/opencv2/core/operation
from /home/jayneil/OpenCV-2.4.2/samples/gpu/surf_keypoint_matcher.cpp:
/home/jayneil/OpenCV-2.4.2/modules/core/include/opencv2/core/operation
operator*(const cv::Matx_4_4d&, const cv::Scalar_4d&):
/home/jayneil/OpenCV-2.4.2/modules/core/include/opencv2/core/operation
type-punned pointer might break strict-aliasing rules [-Wstrict-aliasing]
Linking CXX executable ../../bin/surf_keypoint_matcher_gpu
[100%] Built target example_gpu_surf_keypoint_matcher
jayneil@jayneil-VirtualBox:~/OpenCV-2.4.2/build$
```

Step 5 – installing OpenCV

Run the following command:

```
sudo make install
```

```
jayneil@jayneil-VirtualBox:~/OpenCV-2.4.2/build$ sudo make install
[sudo] password for jayneil:
[ 2%] Built target zlib
[ 7%] Built target libtiff
[ 13%] Built target libjpeg
[ 17%] Built target libjasper
[ 19%] Built target libpng
[ 22%] Built target opencv_core
[ 22%] Built target opencv_ts
[ 27%] Built target opencv_imgproc
[ 30%] Built target opencv_highgui
[ 32%] Built target opencv_perf_core
[ 33%] Built target opencv_test_core
```

The above installation process can take some time depending on your system configuration. It took approximately about 2 minutes on our Intel i7 laptop, while it took around 15 to 20 minutes on our Intel i3 laptop. The following screenshot shows the output after the successful completion of the process.

```
-- Installing: /usr/local/share/OpenCV/samples/gpu/surf_keypoint_match
-- Installing: /usr/local/share/OpenCV/samples/gpu/alpha_comp.cpp
-- Installing: /usr/local/share/OpenCV/samples/gpu/aloeL.jpg
-- Installing: /usr/local/share/OpenCV/samples/gpu/aloeR.jpg
-- Installing: /usr/local/share/OpenCV/samples/gpu/basketball2.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/rubberwhale2.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/basketball1.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/tsucuba_right.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/rubberwhale1.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/tsucuba_left.png
-- Installing: /usr/local/share/OpenCV/samples/gpu/road.png
jayneil@jayneil-VirtualBox:~/OpenCV-2.4.2/build$
```

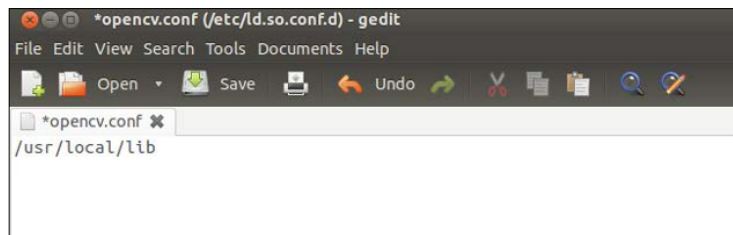
Step 6 – configuring Linux

1. Edit `/etc/ld.so.conf.d/opencv.conf` and add `/usr/local/lib` to it:

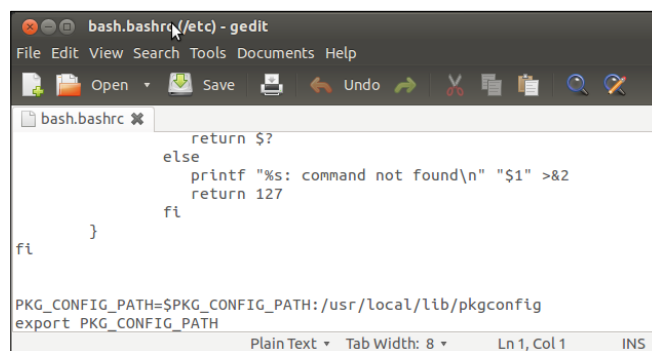
```
sudo gedit /etc/ld.so.conf.d/opencv.conf
sudo ldconfig
```

2. Edit the `bash.rc` file and add the following to it:

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



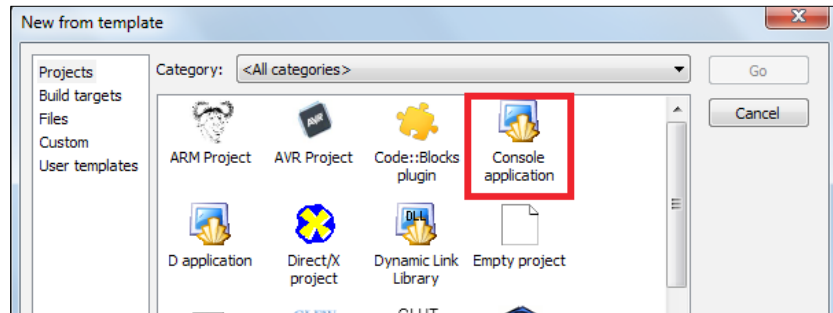
```
sudo gedit /etc/bash.bashrc
```



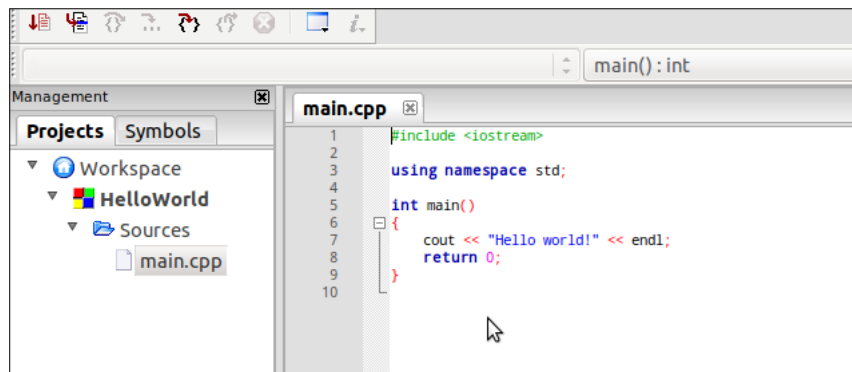
Now, log out of the system or restart it.

Step 7 – configuring OpenCV with Code::Blocks

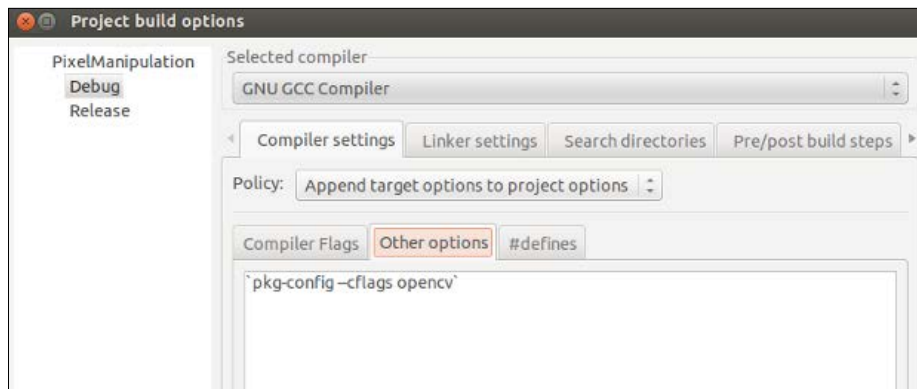
1. Install Code::Blocks IDE from the Ubuntu Software Center.
2. Start Code::Blocks IDE.
3. Create a new project by clicking on the **Create a New Project** tab.
4. Select **Console application** from **Projects**:



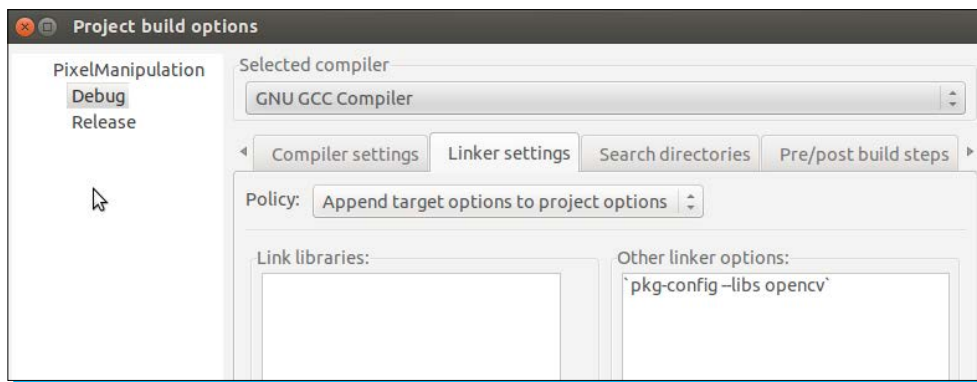
5. Select the language as **C++** and click on **Next**.
6. Give a name to your project and click on **Next**.
7. Make sure you are using GNU GCC Compiler. Click on **Finish**.



8. Find your project under the **Projects** tab of the **Management** view. To use OpenCV in your project the following preconfigurations are required:
 - i. Go to **Build options** by right-clicking on your project.
 - ii. Jump to the **Other options** tab in **Compiler settings**.
 - iii. Add 'pkg-config --cflags opencv' in the blank space.



- iv. Jump to the **Linker settings** tab.
- v. Add 'pkg-config --libs opencv' in the blank space under **Other linker options**.



To compile your code from the command line using gcc, use the following command:

```
g++ 'pkg-config --cflags --libs opencv' -o main main.cpp
```

OpenCV uninstallation procedure for Linux

1. Go to the `build` folder (inside the OpenCV folder).
2. Now run the following command in the terminal:
`sudo make uninstall`
3. Delete the entire OpenCV folder.

4. Run the following command in the terminal, to delete every file containing opencv in its name:

```
sudo find / -name "*opencv*" -exec rm -i {} \;
```
5. Edit the opencv.conf file under /etc/ld.so.conf.d and remove /usr/local/lib from it:

```
sudo gedit /etc/ld.so.conf.d/opencv.conf
```

```
sudo ldconfig
```
6. Edit the bash.rc file and remove the following lines from it:

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

```
export PKG_CONFIG_PATH
```

```
sudo gedit /etc/bash.bashrc
```
7. To make sure you have been successful, check the following paths:
 - /usr/local/bin
 - /usr/local/lib

OpenCV installation procedure for Windows

To install OpenCV in Windows, follow the steps mentioned in the next sections.

Step 1 – installing the dependencies

1. Download MinGW from <http://sourceforge.net/projects/mingw/files/> and install it in the c drive.
2. Download and install Path Editor from <http://www.redfernplace.com/software-projects/patheditor/>.

Step 2 – installing OpenCV

Download the latest version of OpenCV from the following link and run the executable file:

<http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.2/>

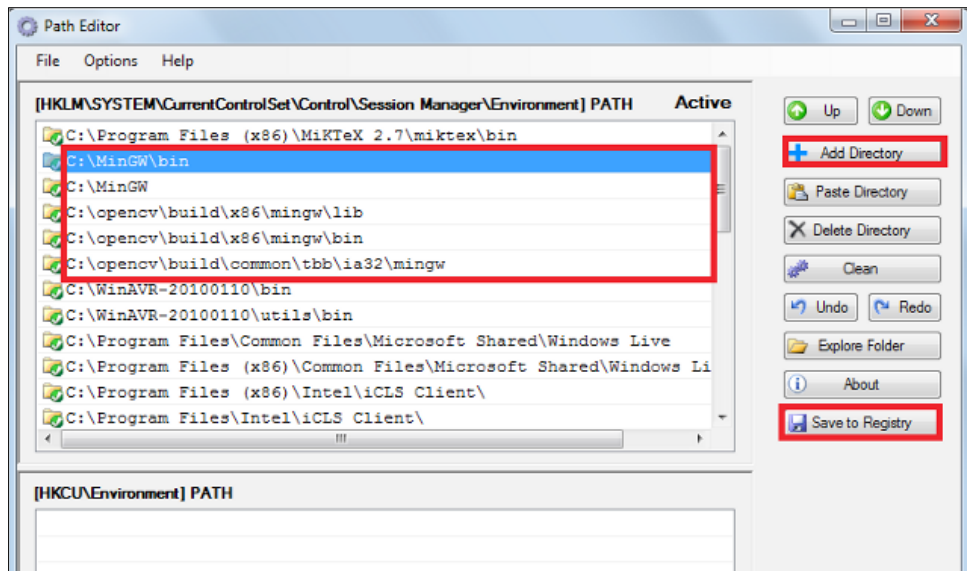


Make sure that you are installing OpenCV in your c drive.

Step 3 – configuring OpenCV with Code::Blocks

1. Download and install Code::Blocks from the following link:
<http://www.codeblocks.org/downloads/26>

2. Run Path Editor and click on **Add Directory**, and then include all the following locations one by one and click on **Save to Registry**.
 - C:\MinGW\bin
 - C:\MinGW
 - C:\opencv\build\x86\mingw\lib
 - C:\opencv\build\x86\mingw\bin
 - C:\opencv\build\common\tbb\ia32\mingw



Add the path of the bin folder for MinGW and OpenCV.

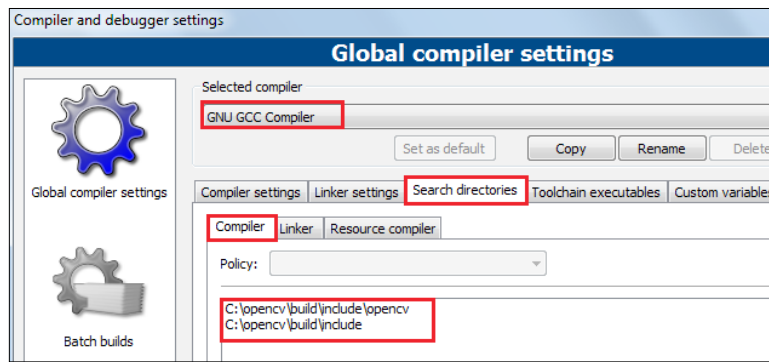


3. Start Code::Blocks IDE.
4. Create a new project by clicking on the **Create a New Project** tab.
5. Select **Console application** from **Projects**, as shown in the first screenshot in *Step 7 – configuring OpenCV with Code::Blocks* discussed earlier.
6. Select the language as **C++** and click on **Next**.
7. Give a suitable name to your project and click on **Next**.
8. Make sure that you are using GNU GCC Compiler and click on **Finish**.

9. Find your project under the **Projects** tab of the **Management** view.

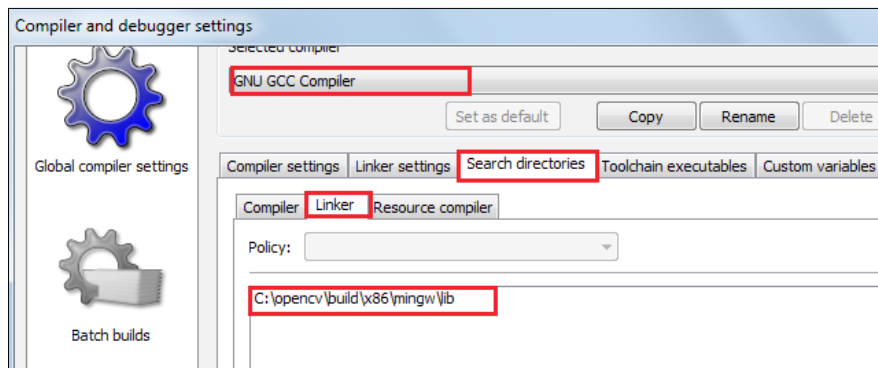



10. To use OpenCV in your project, the following preconfigurations are required:
 - i. Go to the **Settings | Compiler** option.
 - ii. Open the **Search Directories** tab and choose the **compiler** option.
 - iii. Add the following paths by clicking on the **Add** button:
 - C:\opencv\build\include\opencv
 - C:\opencv\build\include



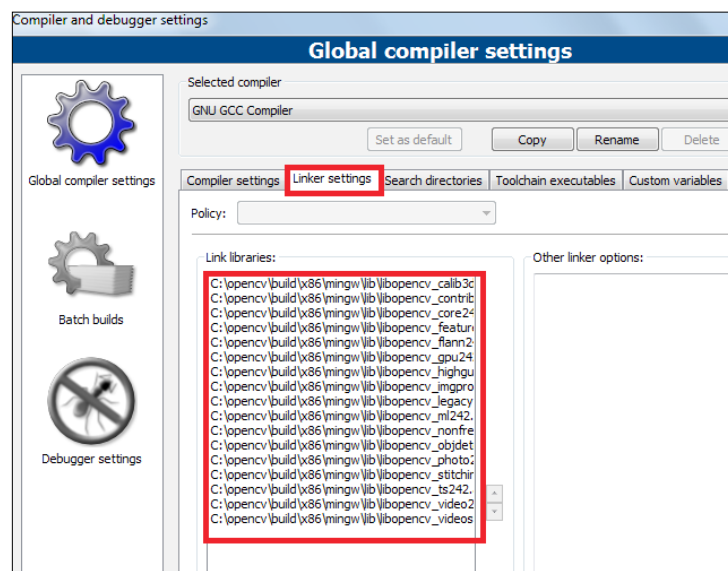
Set the path of the `include` folders from the installed OpenCV folder.

11. Jump to the **Linker** tab under **Search Directories** and add the following path by clicking on the **Add** button:
C:\opencv\build\x86\mingw\lib



 Set the path of the library files from the installed OpenCV folder.

12. Go to the **Linker Settings** tab and click on the **Add** button in the lower-left corner. Browse to `c:\opencv\build\x86\mingw\lib` and include all the available libraries.



13. Click on the **OK** button to save your settings.

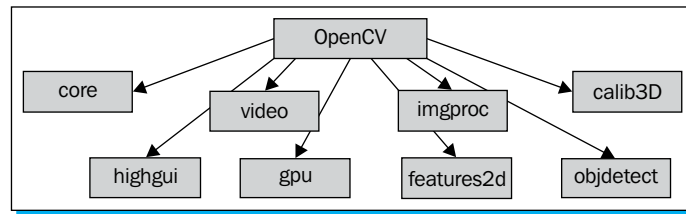
And that's it

By this point, you should have a working installation of OpenCV on your machine. Feel free to play around and discover more about it.

Quick start – OpenCV fundamentals

In this section we will be covering the fundamentals of image processing and help you write your first program in OpenCV by performing a few trivial tasks. All the examples throughout the book have been written in the C++ programming language.

The OpenCV library has a modular structure, and the following diagram depicts the different modules available in it:



A brief description of all the modules is as follows:

Module	Feature
Core	A compact module defining basic data structures, including the dense multidimensional array Mat and the basic functions used by all other modules.
Imgproc	An image processing module that includes linear and non-linear image filtering, geometrical image transformations (resizing, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
Video	A video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
Calib3d	Basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
Features2d	Salient feature detectors, descriptors, and descriptor matchers.
Objdetect	Detection of objects and instances of the predefined classes; for example, faces, eyes, mugs, people, cars, and so on.
Highgui	An easy-to-use interface for video capturing, image and video codecs, as well as simple UI capabilities.
Gpu	GPU-accelerated algorithms from different OpenCV modules.

Task 1 – image basics

When trying to recreate the physical world around us in digital format via a camera, for example, the computer just sees the image in the form of a code that just contains the numbers 1 and 0. A digital image is nothing but a collection of pixels (picture elements) which are then stored in matrices in OpenCV for further manipulation. In the matrices, each element contains information about a particular pixel in the image. The pixel value decides how bright or what color that pixel should be. Based on this, we can classify images as:

- ◆ Greyscale
- ◆ Color/RGB

Greyscale

Here the pixel value can range from 0 to 255 and hence we can see the various shades of gray as shown in the following diagram. Here, 0 represents black and 255 represents white:

144	143	143	143	154	150	150	147	153
147	148	149	142	148	147	153	151	150
144	146	145	148	146	145	151	150	152
144	149	141	145	143	144	149	151	149
144	142	149	148	146	148	147	149	149
141	148	146	144	145	145	152	146	148
140	146	145	142	145	142	148	147	147
138	142	141	147	142	143	145	141	145
137	144	139	145	149	145	144	143	146

A special case of grayscale is the binary image or black and white image. Here every pixel is either black or white, as shown in the following diagram:

0	0	0	0	255	255	255	255	255
0	0	255	255	255	255	255	0	0
0	255	255	255	255	255	255	0	0
255	255	255	255	255	255	0	0	255
255	255	255	255	0	0	255	255	255
255	255	255	0	255	255	255	255	255
255	255	0	0	255	255	255	255	255
255	0	0	255	255	255	255	255	255
0	255	255	255	255	255	255	255	255

Color/RGB

Red, Blue, and Green are the primary colors and upon mixing them in various different proportions, we can get new colors. A pixel in a color image has three separate channels—one each for Red, Blue, and Green. The value ranges from 0 to 255 for each channel, as shown in the following diagram:

1	113	108	114	109	117	115	112	119	126	122	1
5	224	222	224	224	226	224	228	226	232	227	2
5	114	110	116	113	118	124	113	118	126	127	1
9	119	117	117	118	111	124	119	115	129	128	1
2	220	225	229	226	227	225	228	226	229	227	2
8	111	113	113	115	119	120	112	115	123	120	1
2	119	109	111	116	116	121	112	111	128	115	1
4	225	226	224	227	227	228	229	229	231	229	2
8	112	108	110	115	113	113	114	117	125	122	1
3	115	116	113	118	117	117	117	111	121	121	1
2	224	224	225	224	225	227	227	227	230	227	2
9	107	110	108	111	116	105	113	120	120	123	1
4	111	116	110	115	117	112	117	117	120	128	1
5	224	224	225	224	223	227	226	230	227	229	2
7	114	115	104	108	112	112	121	121	123	117	1
7	117	116	107	114	114	118	117	116	131	109	1
5	226	221	225	226	225	225	228	228	229	226	2
3	107	111	110	112	118	117	124	118	124	119	1
9	111	122	114	118	123	125	121	123	126	118	1
4	222	224	223	228	226	227	228	229	231	228	2
7	105	104	110	108	116	110	121	121	120	122	1
4	113	109	113	112	119	105	123	115	118	128	1
9	222	224	226	229	226	226	229	233	228	228	2
3	105	106	103	106	111	107	112	116	116	120	1
5	109	108	108	110	106	108	117	108	116	113	1
4	220	225	227	226	225	224	227	226	229	228	2

Task 2 – reading and displaying an image

We are now going to write a very simple and basic program using the OpenCV library to read and display an image. This will help you understand the basics.

Code

A simple program to read and display an image is as follows:

```
// opencv header files
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/core/core.hpp"
// namespaces declaration
using namespace cv;
using namespace std;
// create a variable to store the image
Mat image;
int main( int argc, char** argv )
{
    // open the image and store it in the 'image' variable
    // Replace the path with where you have downloaded the image
    image=imread("<path to image>/lena.jpg");
    // create a window to display the image
    namedWindow( "Display window", CV_WINDOW_AUTOSIZE );
    // display the image in the window created
    imshow( "Display window", image );
    // wait for a keystroke
    waitKey(0);
    return 0;
}
```

Code explanation

Now let us understand how the code works. Short comments have also been included in the code itself to increase the readability.

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/core/core.hpp"
```

The preceding two header files will be a part of almost every program we write using the OpenCV library. As explained earlier, the `highgui` header is used for window creation, management, and so on, while the `core` header is used to access the `Mat` data structure in OpenCV.

```
using namespace cv;
using namespace std;
```

Instant OpenCV Starter

The preceding two lines declare the required namespaces for this code so that we don't have to use the `::` (scope resolution) operator every time for accessing the functions.

```
Mat image;
```

With the above command, we have just created a variable `image` of the datatype `Mat` that is frequently used in OpenCV to store images.

```
image=imread("<path to image>/lena.jpg");
```

In the previous command, we opened the image `lena.jpg` and stored it in the `image` variable. Replace `<path to image>` in the preceding command with the location of that picture on your PC.

```
namedWindow( "Display window", CV_WINDOW_AUTOSIZE );
```

We now need a window to display our image. So, we use the above function to do the same. This function takes two parameters, out of which the first one is the name of the window. In our case, we would like to name our window `Display Window`. The second parameter is optional, but it resizes the window based on the size of the image so that the image is not cropped.

```
imshow( "Display window", image );
```

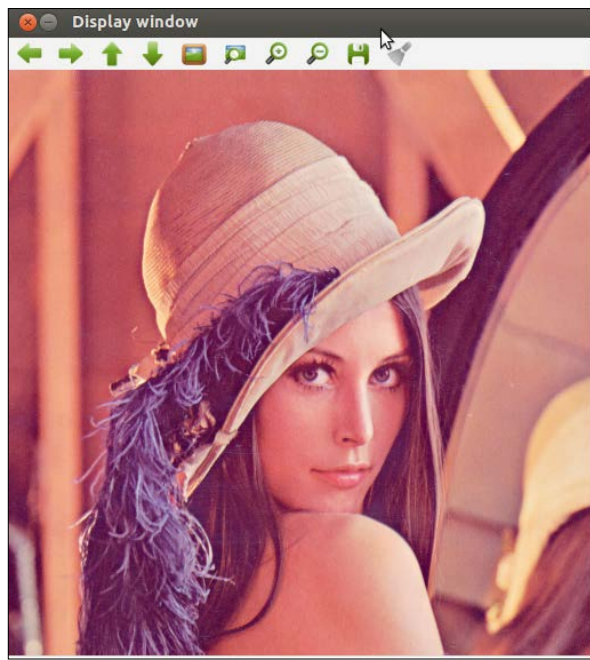
Finally, we are ready to display our image in the window we just created by using the preceding function. This function takes two parameters out of which the first one is the window name in which the image has to be displayed. In our case, obviously, that will be **Display Window**. The second parameter is the image variable containing the image that we want to display. In our case, it's the `image` variable.

```
waitKey(0);
```

Last but not least, it is advised that you use the preceding function in most of the codes that you write using the OpenCV library. If we don't write this code, the image will be displayed for a fraction of a second and the program will be immediately terminated. It happens so fast that you will not be able to see the image. What this function does essentially is that it waits for a keystroke from the user and hence it delays the termination of the program. The delay here is in milliseconds.

Output

The image can be displayed as follows:



Task 3 – resizing and saving an image

We are now going to write a very simple and basic program using the OpenCV library to resize and save an image.

Code

The following code helps you to resize a given image:

```
// opencv header files
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
// namespaces declaration
using namespace std;
using namespace cv;
int main(int argc, char** argv)
```



```
{
    // create variables to store the images
    Mat org, resized,saved;
    // open the image and store it in the 'org' variable
    // Replace the path with where you have downloaded the image
    org=imread("<path to image>/lena.png");
    //Create a window to display the image
    namedWindow("Original Image",CV_WINDOW_AUTOSIZE);
    //display the image
    imshow("Original Image",org);
    //resize the image
    resize(org,resized,Size(),0.5,0.5,INTER_LINEAR);
    namedWindow("Resized Image",CV_WINDOW_AUTOSIZE);
    imshow("Resized Image",resized);
    //save the image
    //Replace <path> with your desired location
    imwrite("<path>/saved.png",resized);
    namedWindow("Image saved",CV_WINDOW_AUTOSIZE);
    saved=imread("<path to image>/saved.png");
    imshow("Image saved",saved);
    //wait for a keystroke
    waitKey(0);
    return 0;
}
```

Code explanation

Since most of the code in the above program is similar to the one in the *Image conversions* section, only the new functions/concepts will be explained in this case.

```
#include "opencv2/imgproc/imgproc.hpp"
```

Imgproc is another useful header that gives us access to the various transformations, color conversions, filters, histograms, and so on.

```
Mat org, resized;
```

We have now created two variables, `org` and `resized`, to store the original and resized images respectively.

```
resize(org,resized,Size(),0.5,0.5,INTER_LINEAR);
```

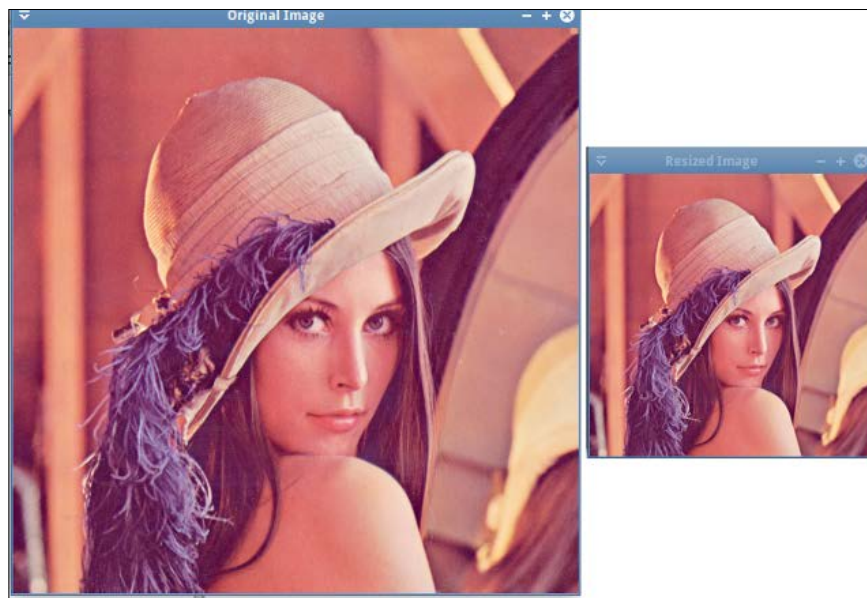
We have used the preceding function to resize the image. The preceding function takes six parameters, out of which the first one is the variable containing the source image to be modified. The second one is the variable to store the resized image. The third parameter is the output image size. In this case we have not specified this, but we have instead used the `Size()` function, which will automatically calculate it based on the values of the fourth and fifth parameters. The fourth and fifth parameters are the scale factors along the horizontal and vertical axes respectively. The sixth parameter is for choosing the type of interpolation method. We have used the bilinear interpolation, which is the default method.

```
imwrite("<path>/saved.png", final);
```

Finally, using the preceding function, you can save an image to a particular location on our PC. The function takes two parameters, out of which the first one is the location where you want to store the image and the second is the variable in which the image is stored. This function is very useful when you want to perform multiple operations on an image and save the image on your PC for future reference. Replace `<path>` in the preceding function with your desired location.

Output

Resizing can be demonstrated through the following output:



Top 5 features you need to know about

Let us do a quick recap of the things we have learnt and done so far. We have had a brief overview of OpenCV and performed a few trivial tasks such as reading, displaying, and saving an image. Now, we will gradually step it up a notch and learn a few useful and more advanced features, such as pixel manipulation and image conversion.

From here onwards, remaining parts of the code will not be published in the book, but the entire codes can be downloaded from the Packt Publishing website.

Pixel manipulation

If you look closely, you will notice that all the codes that we have executed so far perform operations on the image as a whole, and not its individual constituent elements, called **pixels**. There could be many applications where individual pixel manipulation could be useful. For example, you went on a trip to India and took a lot of pictures at various exotic locations. After returning from the trip, you had more than three thousand pictures with your digital camera. Suddenly, you want that beautiful picture you took of the desert. Now, you could flip through all the three thousand images and hope to find the picture or, you could use OpenCV to help you out. Since you know the color of the sand in the desert, you can write an OpenCV program to search for pictures with similar backgrounds by comparing each pixel value in the picture to the color of the desert sand. This will narrow down your search just to a couple of pictures!

So, we will now see how we can perform pixel manipulation in OpenCV.

Task

Given a grayscale or a color image, perform pixel manipulation.

Algorithm

The algorithm is quite similar for grayscale and color images. Let us first understand the algorithm in regards to a grayscale image.

Grayscale

Our program would compare the values of each and every pixel in the image to a predetermined threshold value and then, based on a preset logic, change the existing value of the selected pixels in the image to another desired value. So, for example, let us assume that the threshold value is 100 and our preset logic is designed such that every pixel value that is greater than the threshold value should be made white in the image. Now our program would compare each pixel value in the image to the threshold value, and whichever pixel satisfies the preset logic criteria would be turned white and no change will be made to the remaining pixels.

Color

The logic extends similarly to a color image, but there is a slight modification. Unlike the grayscale image, a pixel in a color image has three components, which are red, green, and blue. It is the combination of the values of these three individual components which decides the resulting color in the image at that pixel. Each of these three components has values ranging from 0 to 255, where 0 would represent black and 255 would represent white. So, we have to take into consideration the values of all three components. Our program in this case would first obtain the value of the three components, compute the average value (let us call it the **average pixel value**), and then use this value to compare with the predetermined threshold value. Then, based on the preset logic, it would change the existing value of the three components for selected pixels in the image to another desired value. So, for example, let us assume that the threshold value is 100 and our preset logic is designed such that every average pixel with a value that is greater than the threshold value should be made white in the image. Now, our program would first obtain the values of the **RGB** (short for RED, GREEN, and BLUE) components of the pixel and then compute the average value. So, this newly computed average value will represent the pixel value and reduce our burden, as we don't have to deal with three separate values. Now our program would compare the average pixel value of each pixel in the image to the threshold value and whichever pixels satisfy the preset logic criteria would be turned white and no change will be made to the remaining pixels.

Code

To perform a pixel manipulation, we will use the following code:

```
void thresholding(Mat &aImage, uchar aThreshValue)
{
    int numberRows = aImage.rows; // Number of Rows in Image(Height)
    int numberCols = aImage.cols; // Number of Cols in Image(Width)
    for(int j = 0; j < numberRows; j++)
        for(int i = 0; i < numberCols; i++)
        {
            if (aImage.channels() == 1)
            {
                // grayscale image
                // Get the value of each pixel
                uchar tValue = aImage.at<uchar>(j,i);
                if(tValue > aThreshValue)
                { // If the pixel value is greater than the threshold value then
make it WHITE
                    aImage.at<uchar>(j,i)= 255;
                }
            }
            else if (aImage.channels() == 3)
            { // color image
```

```
        // Sum of RGB components of the pixel
        int tSum = aImage.at<Vec3b>(j,i)[0] + aImage.at<Vec3b>(j,i)[1]
+ aImage.at<Vec3b>(j,i)[2];
        uchar averageValue = static_cast<uchar>(tSum/3);
        if(averageValue > aThreshValue)
        { // If the average value of RGB components of the pixel is
greater than the threshold value then make the components WHITE
            aImage.at<Vec3b>(j,i)[0]= 255; // Blue component of the
pixel
            aImage.at<Vec3b>(j,i)[1]= 255; // Green component of the
pixel
            aImage.at<Vec3b>(j,i)[2]= 255; // Red component of the
pixel
        }
    }
}
```

Code explanation

Now let us understand the working of the program.

```
void thresholding(Mat &aImage, uchar aThreshValue)
```

This is our function that will perform the pixel manipulation. It accepts two arguments: `aImage` and `aThreshValue`. The `aImage` argument is the matrix containing the image on which we want to perform the pixel manipulation while `aThreshValue` is the variable that contains the threshold value. Note that `uchar` in OpenCV is an 8-bit unsigned integer.

```
    int numberOfRows = aImage.rows; // Number of Rows in Image(Height)
    int numberCols = aImage.cols; // Number of Cols in Image(Width)
```

Here, `numberOfRows` and `numberCols` will store the height (rows) and width (columns) of the image respectively.

```
    if (aImage.channels() == 1)
```

Now, we will check whether the given image is a grayscale image or not. As we discussed earlier, grayscale images only have one channel—only one component that represents the value of each pixel in the image.

```
        uchar tValue = aImage.at<uchar>(j,i);
```

tValue is an 8-bit unsigned integer that will store the value of the pixel at the point (j, i) in the image.

```
if(tValue > aThreshValue)
    aImage.at<uchar>(j,i) = 255;
```

Here, we will check whether the value of a particular pixel is greater than the threshold value or not. If a particular pixel value satisfies the above preset logic, its value is greater than the threshold value, so we will change that pixel to white (255).

```
else if (aImage.channels() == 3)
```

Here, we check whether the given image is a color image or not. As we discussed earlier, color images have three channels (three components), which are red, green, and blue.

```
int tSum = aImage.at<Vec3b>(j,i)[0] + aImage.at<Vec3b>(j,i)[1] +
aImage.at<Vec3b>(j,i)[2];
```

tSum will store the arithmetic sum of the RGB components of the pixel at the point (j, i) in the image. Also, Vec3b is a template class for storing numerical vectors. Here, the numbers [0], [1], [2] are used to access the blue, green, and red components respectively.

```
uchar averageValue = static_cast<uchar>(tSum/3);
```

We now calculate the average value of the pixels.

```
if(averageValue > aThreshValue)
{ // If the average value of RGB components of the pixel is
greater than the threshold value then make it white
    aImage.at<Vec3b>(j,i)[0] = 255; // Blue component of the
pixel
    aImage.at<Vec3b>(j,i)[1] = 255; // Green component of the
pixel
    aImage.at<Vec3b>(j,i)[2] = 255; // Red component of the
pixel
}
```

Here, we will check whether the average value of each pixel is greater than the threshold value or not. If a particular pixel value satisfies the preset logic, its value is greater than the threshold value, so we will change the value of all the components of that pixel to white (255). We also would like to point out that OpenCV also has its own built in threshold function but we have created our own in this case for the sake of simplicity.

Output

The following output shows pixel manipulation:

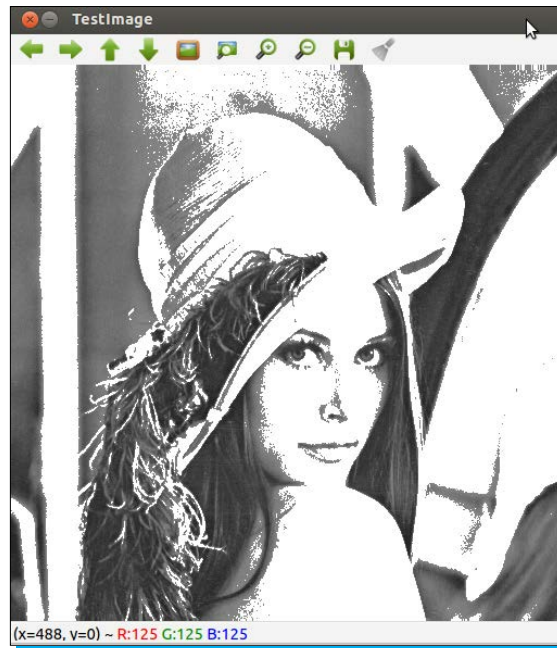


Image conversions

Image conversion is another very powerful tool, which is very essential to learn. There could be various scenarios where dealing with grayscale images compared to color images is much easier. For example, let's say that we wanted to write a program that would detect whether a particular object such as a square or rectangle is present in a given image or not. Here, we are concerned with the shape of the object and not its color. So, converting the image to grayscale from color will reduce a lot of computations for us, as we only have to work with one channel.

Task

Given a color image, convert it to a grayscale image.

Code

Use the following code to convert a color image into a grayscale image:

```
// opencv header files
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
```

```

// namespaces declaration
using namespace cv;
using namespace std;
// create a variable to store the image
Mat image;
Mat gray;

int main( int argc, char** argv )
{
// open the image and store it in the 'image' variable
// Replace the path with where you have downloaded the image
image=imread("<Path to Image>/RGB.png");
// create single channel image to store output grayscale image
gray.create(image.rows, image.cols, CV_8UC1);
// convert RGB image to GrayScale
cvtColor(image, gray, CV_BGR2GRAY);
// create a window to display the image
namedWindow( "Color Image", CV_WINDOW_AUTOSIZE );
namedWindow("GrayScale Image", CV_WINDOW_AUTOSIZE);
// display the image in the window created
imshow( "Color Image", image );
imshow( "GrayScale Image", gray);
// wait for a keystroke
waitKey(0);
return 0;
}

```

Code explanation

The code is quite similar to the ones we have looked at before. So, only the portions that need further explanation have been discussed in the following.

```
gray.create(image.rows, image.cols, CV_8UC1);
```

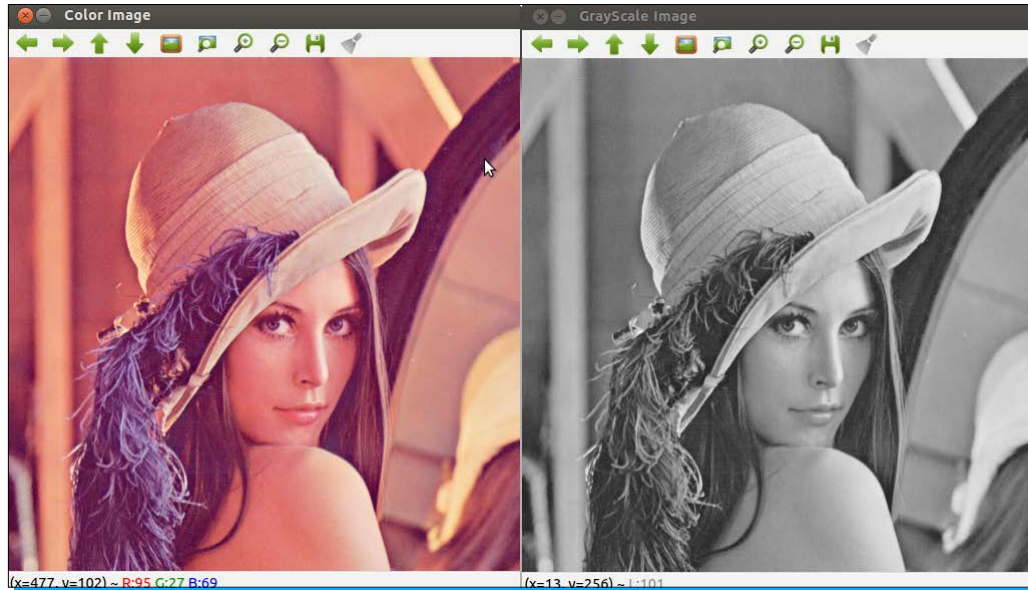
The preceding function is used to create a matrix with a single channel to store the resulting grayscale image.

```
cvtColor(image, gray, CV_BGR2GRAY);
```

This OpenCV function performs various image conversions. In this case, it converts a given color image to grayscale. The function takes three arguments. The first one is the matrix containing the source image (the image that we want to convert), while the second one is the matrix in which we want to store the converted image. The third argument specifies the type of conversion that we want perform and in our case, it is `CV_BGR2GRAY`. We can also use the above function to convert an image from a given color space model like RGB to other models such as HSV and CIELAB.

Output

The following output shows the comparison between the original color image and the grayscale image that we created:



What good is learning a technology when you don't do some awesome, real-life application based projects on it? So, now we will do three awesome projects that will hopefully highlight the importance and the various possible usages of OpenCV.

Image steganography

This is a very interesting technology that is being used for the wrong reasons in the world today. It is used in drug smuggling, trafficking, terrorism, and so on. Let us first understand this technique. For example, let's say you have a treasure map that leads to the hidden treasures of King Asoka the Great! Due to the sensitive and secretive nature of the information, you cannot just pass it on to someone without the wrong people getting their hands on it. You could send the image of the treasure map via e-mail but there are always hackers who could hack your account and obtain it. Meeting that person face to face is also risky. Here is where image steganography can help you out. What if you could send the image of the treasure map hidden inside the image of a house or a cat? This is exactly what image steganography is! Modern research has led to many advanced forms of image steganography that are heavily encrypted and not easily detectable.

So, now let us understand how to write an OpenCV code to perform the above operation.

For better understanding this process has been split into two parts.

Part 1 – encode

This is the first part of the program.

Task

Hide a given image in another image using image steganography.

Algorithm

Think of a number, for example, 126. We can say that the most important digit in this number is 1 which is in the hundredth place. It carries maximum weightage compared to the other digits 2 and 6. Even if the other digits were dropped, it still conveys the information that it is a number greater than or equal to 100. The same cannot be said if the digit 1 was dropped and the remaining were retained. The same reasoning can be extended to computer data. If you have an 8-bit integer, the first bit on the left-hand side is called the **Most Significant Bit (MSB)** while the last bit on the right is called the **Least Significant Bit (LSB)**. The priority/weightage decreases as we go from left to right (from MSB to LSB). So, if we store an 8-bit image in OpenCV, each pixel in the image is represented by an 8-bit integer and the most valuable information about the pixel is in the MSB of that pixel. Now, let us call the image that we want to hide (hidden image) and the image in which the hidden image will be stored (front image). So, the hidden image will be hidden inside the front image! We can store the MSB of the pixels of the hidden image in the LSB of the front image. This provision does not disrupt the information content of either of the images. Image steganography can lead to a loss of a lot of information. Therefore, in our case, we have stored the first four bits (starting from the MSB) of each pixel in the hidden image in the last four bits (ending at the LSB) of each pixel in the front image.

Code

We use the following code to hide a given image in another image:

```
void steganographMyImage(Mat& aFrontImage, Mat& aHiddenImage, Mat&
aStegedImage)
{
    // check for size and type of both the given images
    if(aFrontImage.type() != aHiddenImage.type() || aFrontImage.size()
!= aHiddenImage.size())
    {
        printf("Given Image types are different \n");
        return;
    }
    int numberRows = aFrontImage.rows; // Number of Rows in
Image(Height)
    int numberCols = aFrontImage.cols; // Number of Cols in
Image(Width)
    // create output Image
    aStegedImage.create(numberRows, numberCols, aFrontImage.type());
```

```
Mat tFront_image, tHidden_image;

Mat front_mask(numberRows, numberCols, aFrontImage.type(),
Scalar(0xF0, 0xF0, 0xF0));
Mat hidden_mask(numberRows, numberCols, aFrontImage.type(),
Scalar(0xF0, 0xF0, 0xF0));

bitwise_and(aFrontImage, front_mask, tFront_image);
bitwise_and(aHiddenImage, hidden_mask, tHidden_image);

for(int j = 0; j < numberRows; j++)
    for(int i = 0; i < numberCols; i++){
        tHidden_image.at<Vec3b>(j,i)[0] = tHidden_image.
at<Vec3b>(j,i)[0] >> 4;
        tHidden_image.at<Vec3b>(j,i)[1] = tHidden_image.
at<Vec3b>(j,i)[1] >> 4;
        tHidden_image.at<Vec3b>(j,i)[2] = tHidden_image.
at<Vec3b>(j,i)[2] >> 4;
    }

bitwise_or(tFront_image, tHidden_image, aStegedImage);

}
```

Code explanation

The function which will perform the image steganography is as follows:

```
void steganographMyImage(Mat& aFrontImage, Mat& aHiddenImage, Mat&
aStegedImage)
```

This function takes three parameters. The first one is the matrix containing the image in which we want to hide out desired image. The second argument is the matrix containing the image that we want to hide and the third and last argument is the matrix which will store the resulting steganograph image.

```
if(aFrontImage.type() != aHiddenImage.type() || aFrontImage.size() !=
aHiddenImage.size())
```

This code has been written for the case when both the front image and hidden image are of the same size for the sake of easier understanding and simplicity. Here, we will check whether both the images have the same type (bit size, that is, 8 bit, 16 bit, and so on) and size. If they are not equal in either size or type, we terminate the execution of our function.

```
aStegedImage.create(numberRows, numberCols, aFrontImage.type());
```

With the help of the above function, we make sure that the attributes (columns, rows, and type) of the `aStegedImage` matrix which will store the steganograph image are the same as that of the `aFrontImage` matrix which contains the carrier image in which we will hide our desired image.

```
Mat front_mask(numberRows, numberCols, aFrontImage.type(),
Scalar(0xF0, 0xF0, 0xF0));
Mat hidden_mask(numberRows, numberCols, aFrontImage.type(),
Scalar(0xF0, 0xF0, 0xF0));
```

The above functions create a matrix with the specified number of rows, columns, and type, and even initializes each element with the provided value. In our case, we have first created a matrix, `front_mask` with rows, columns, and type the same as that of the `aFrontImage` matrix. We then initialized all the matrix elements with the value `0xF0` or `11110000` in binary. Why three times? It is because this is a color image, so it has three channels or three components. We then perform a similar operation to the matrix `hidden_mask`.

```
bitwise_and(aFrontImage, front_mask, tFront_image);
bitwise_and(aHiddenImage, hidden_mask, tHidden_image);
```

The above functions perform bitwise ANDing of two matrices and store the result in a third matrix. In our case, we have first performed the bitwise ANDing of the `aFrontImage` and `front_mask` matrices, and stored it in the `tFront_image` matrix. So, what have we achieved with this operation? Well, now the resulting `tFront_image` matrix contains only the first four important bits of each pixel in `aFrontImage`. The remaining four bits are zero padded. The second line performs a similar operation as the first one.

```
tHidden_image.at<Vec3b>(j,i)[0] = tHidden_image.at<Vec3b>(j,i)[0] >>
4;
tHidden_image.at<Vec3b>(j,i)[1] = tHidden_image.at<Vec3b>(j,i)[1] >>
4;
tHidden_image.at<Vec3b>(j,i)[2] = tHidden_image.at<Vec3b>(j,i)[2] >>
4;
```

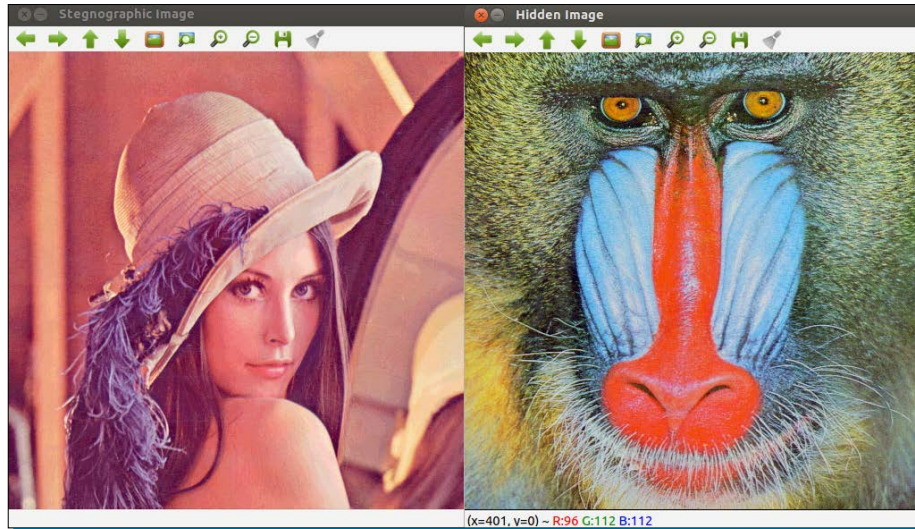
The preceding functions right-shift the pixel components of the `tHidden_image` matrix by 4 bits, and hence the first four bits are zero padded.

```
bitwise_or(tFront_image, tHidden_image, aStegedImage);
```

Finally, we perform the bitwise addition of the `tFront_image` and `tHidden_image` matrices to obtain `aStegedImage`, which is our steganograph image.

Output

The output showing image steganography is as follows:



Part 2 – Decode

This is the second part of the program.

Task

Obtain the original image from the steganograph image.

Algorithm

The algorithm here is quite similar to the one we used previously in the case of performing image steganography. Here we already have the steganograph image and we need to obtain the hidden image and carrier image (the image in which the hidden image is hidden) from it. So, we need to reverse engineer the algorithm we had used earlier. In the given steganograph image, we know that the first four bits of the pixel represent the first four bits of the carrier image. The remaining four bits represent the first four bits of the hidden image. So, we have to separate these bits and store them in their corresponding images.

Code

Use the following code to retrieve the original image from a steganograph image:

```
void getOriginalImages(Mat& aStegedImage, Mat& aFrontImage, Mat&
aHiddenImage)
{
    int numberRows = aStegedImage.rows; // Number of Rows in
    Image(Height)
```

```

    int numberCols = aStegedImage.cols; // Number of Cols in
    Image(Width)

    aFrontImage.create(numberRows, numberCols, aStegedImage.type());
    aHiddenImage.create(numberRows, numberCols, aStegedImage.type());

    Mat tFront_image, tHidden_image;

    Mat front_mask(numberRows, numberCols, aStegedImage.type(),
    Scalar(0xF0, 0xF0, 0xF0));
    Mat hidden_mask(numberRows, numberCols, aStegedImage.type(),
    Scalar(0x0F, 0x0F, 0x0F));

    bitwise_and(aStegedImage, front_mask, aFrontImage);
    bitwise_and(aStegedImage, hidden_mask, aHiddenImage);

    for(int j = 0; j < numberOfRows; j++)
        for(int i = 0; i < numberCols; i++){
            aHiddenImage.at<Vec3b>(j,i)[0] = aHiddenImage.
at<Vec3b>(j,i)[0] << 4;
            aHiddenImage.at<Vec3b>(j,i)[1] = aHiddenImage.
at<Vec3b>(j,i)[1] << 4;
            aHiddenImage.at<Vec3b>(j,i)[2] = aHiddenImage.
at<Vec3b>(j,i)[2] << 4;
        }
    }

```

Code explanation

Since the code is quite similar to the one which performs the actual steganograph, only the portions which were different and need explanation have been discussed here.

```

void getOriginalImages(Mat& aStegedImage, Mat& aFrontImage, Mat&
aHiddenImage)

```

This is our function which will help us obtain the carrier image and the hidden image from the steganograph image and it takes three arguments. The first argument is the matrix that contains the steganograph image. The second one is the matrix containing the image in which we have hidden our desired image. The third argument is the matrix containing the image that is desired or hidden.

```

    Mat front_mask(numberRows, numberCols, aStegedImage.type(),
    Scalar(0xF0, 0xF0, 0xF0));

```

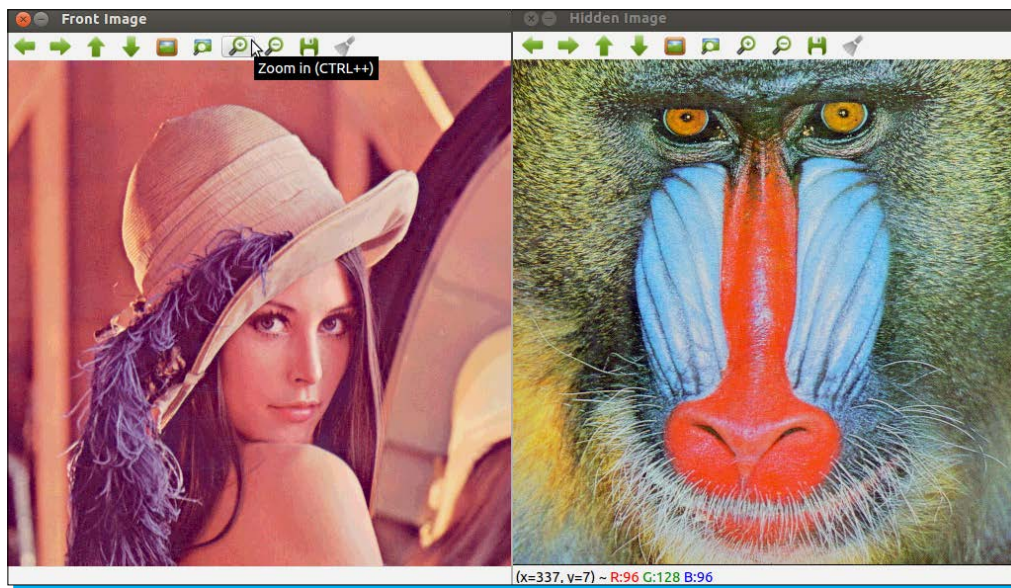

Unlike in the previous case, here we initialize each element of the `front_mask` matrix with the value `0x0F` or `11110000` in binary. Why? Because we want the first four bits of the steganograph image as they correspond to the first four bits of the carrier image.

```
aHiddenImage.at<Vec3b>(j,i)[0] = aHiddenImage.at<Vec3b>(j,i)[0] << 4;  
aHiddenImage.at<Vec3b>(j,i)[1] = aHiddenImage.at<Vec3b>(j,i)[1] << 4;  
aHiddenImage.at<Vec3b>(j,i)[2] = aHiddenImage.at<Vec3b>(j,i)[2] << 4;
```

The preceding functions left-shift the pixel components of `aHidden_image` by 4 bits, because the first four bits are zero padded and the actual information is stored in the last 4 bits.

Output

The following screenshot shows the original image when retrieved from the steganograph image:



Edge detection

Edge detection is another very important technique used a lot in computer vision. We have seen a very interesting application of this in real life. A biscuit manufacturing company has to manufacture thousands of biscuits daily and maintain the standard and quality; it cannot afford to have workers check each and every biscuit to make sure that each one is rectangular in shape. There will of course be defective pieces. So, the company uses edge detection and robotics to filter out and remove the defective pieces from the main lot.

Task

For a given image, detect the edges/boundaries in it.

Code

The following code will detect the edges or boundaries in a given image:

```
int main( int argc, char** argv )
{

    // create a variable to store the image
    Mat image, gray, edge, cedge;
    // Threshold value for canny edge detection
    int edgeThresh = 10;

    // open the image and store it in the 'image' variable
    // Replace the path with where you have downloaded the image
    image=imread("<Path to Image>/RGB.png");

    // create single channel image to store output gray image
    gray.create(image.rows, image.cols, CV_8UC1);
    // convert RGB image to GrayScale
    cvtColor(image, gray, CV_BGR2GRAY);
    // create image to store final edge detected image.
    cedge.create(gray.size(), gray.type());

    // Run the edge detector on grayscale
    Canny(gray, edge, edgeThresh, edgeThresh*3, 3);
    cedge = Scalar::all(0);

    image.copyTo(cedge, edge);

    namedWindow("Output Image", CV_WINDOW_AUTOSIZE);
    // display the image in the window created
    imshow("Output Image", cedge);
    waitKey(0);
    return 0;
}
```

Code explanation

The code here is quite similar to the ones we have looked at before. So, only the portions that need further explanation have been discussed, as follows:

```
Canny(gray edge, edgeThresh, edgeThresh*3, 3);
```


We have used the Canny function in OpenCV to implement the Canny algorithm for edge detection. This function accepts usually five parameters, but the last one is optional. The first parameter is the matrix containing the source image while the second is the matrix in which we want to store the resulting output. The third and fourth parameters are threshold 1 and threshold 2 respectively. The smallest value between `threshold1` and `threshold2` is used for edge linking. The largest value is used to find the initial segments of strong edges. It is usually recommended to make the value of `threshold2` three times the value of `threshold1`. The fourth parameter is the aperture size for the Sobel operator, and it is also known as the kernel size.

```
cedge = Scalar::all(0);
```

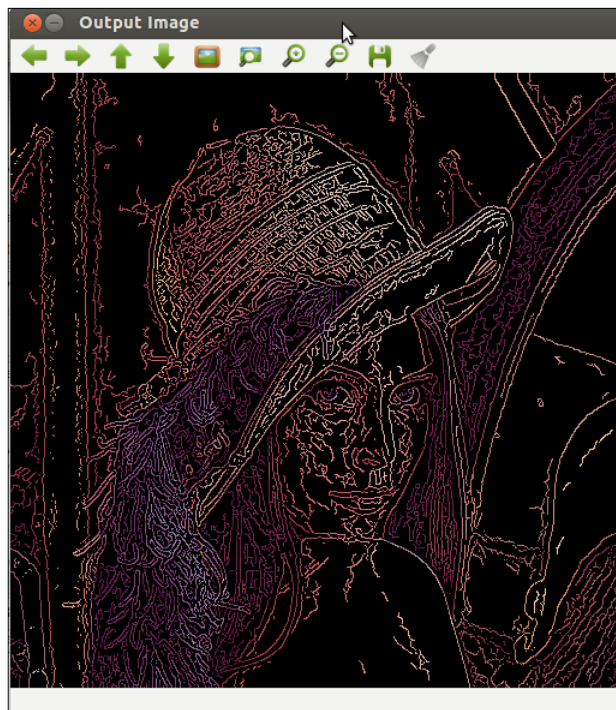
This is to make sure that all the elements of the `cedge` matrix are initialized with the value 0 and not any other random value.

```
image.copyTo(cedge, edge);
```

The preceding function copies one matrix to another. So, we have basically copied the image contained in the `edge` matrix to the `cedge` matrix.

Output

The following output shows the edges and boundaries in a given image:



Real-time video processing via webcam

Video processing is a very important technique used these days by the police to find or locate suspects after a crime. Next time you visit a small shop in your neighborhood, don't be surprised to see CCTV cameras installed inside it. This is to help small shop owners catch shoplifters. Next, just to get a brief idea of video processing, we will do a very trivial task.

Task

Convert the real-time color video feed from your webcam to grayscale and display it in a window.

Algorithm

A video is nothing but a sequence/collection of frames/images. So, to process it, we can split it into its constituent frames and perform the desired operations on those frames.

Code

The following code converts a colored image into a grayscale image:

```
int main( int argc, char** argv )
{
    Mat Image;
    Mat gray;
    char key = 0;

    // open the default camera
    VideoCapture capture(0);

    // check for failure
    if (!capture.isOpened()) {
        printf("Failed to open a video device or video file!\n");
        return 1;
    }
    // Set Capture device properties.
    capture.set(CV_CAP_PROP_FRAME_WIDTH, 640);
    capture.set(CV_CAP_PROP_FRAME_HEIGHT, 480);

    namedWindow("Camera Video", CV_WINDOW_AUTOSIZE);
    namedWindow("Processed Video", CV_WINDOW_AUTOSIZE);

    while( key != 'q')
    {
        // get a new frame from camera
        capture >> Image;

        cvtColor(Image, gray, CV_BGR2GRAY);
```

Instant OpenCV Starter

```
        imshow("Camera Video", Image);
        imshow("Processed Video", gray);

        key = waitKey(25);
    }
    return 0;
}
```

Code explanation

Since the code is quite similar to the ones we have studied earlier, only the portions which were different and need explanation have been discussed here.

```
VideoCapture capture(0);
```

The above line of code opens the default camera on your computer. VideoCapture is a class in OpenCV which provides a C++ video capturing API.

```
if (!capture.isOpened())
```

This checks for failure—the case where no camera interface could be opened on the computer.

```
capture.set(CV_CAP_PROP_FRAME_WIDTH, 640);
capture.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
```

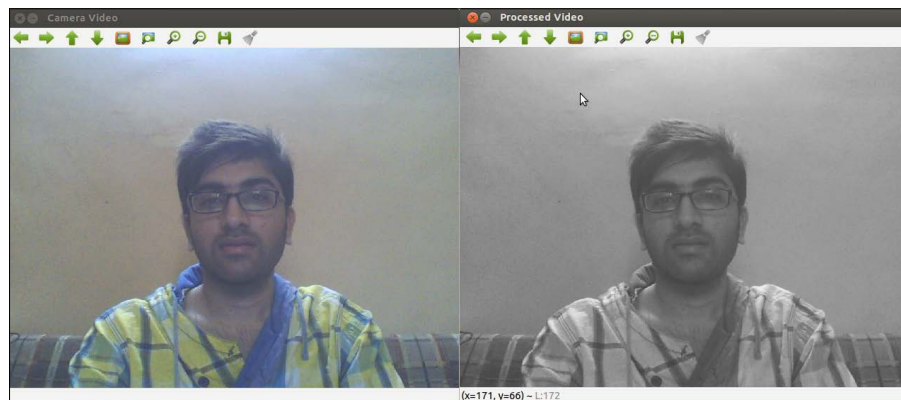
The above lines of code set the capture device properties. To know the other properties of the videoCapture class refer to the reference document of this class.

```
capture >> Image;
```

This gets a new frame of the camera and stores it in the Image matrix.

Output

The following output shows the comparison between a colored and a grayscale image:



People and places you should get to know

If you need help with OpenCV, here are some people and places that will prove invaluable.

Official sites

- ◆ **OpenCV home page:** www.opencv.org
- ◆ **OpenCV documentation:** <http://docs.opencv.org/>
- ◆ **OpenCV user guide:** http://docs.opencv.org/doc/user_guide/user_guide.html

Tutorials/cheat sheets/answers

- ◆ **Official OpenCV tutorials:** <http://docs.opencv.org/doc/tutorials/tutorials.html>
- ◆ **OpenCV cheat sheet:** http://docs.opencv.org/trunk/opencv_cheatsheet.pdf
- ◆ **OpenCV answers:** <http://answers.opencv.org/questions/>

Community

- ◆ **OpenCV Yahoo group:** <http://tech.groups.yahoo.com/group/OpenCV/>
- ◆ **OpenCV Google Plus group:** <https://plus.google.com/communities/106558044109618648316>

Twitter

- ◆ **Official OpenCV Twitter page:** <https://twitter.com/opencvlibrary>
- ◆ For more open source information, follow Packt at <http://twitter.com/#!/packtopensource>



Thank you for buying
Instant OpenCV Starter

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

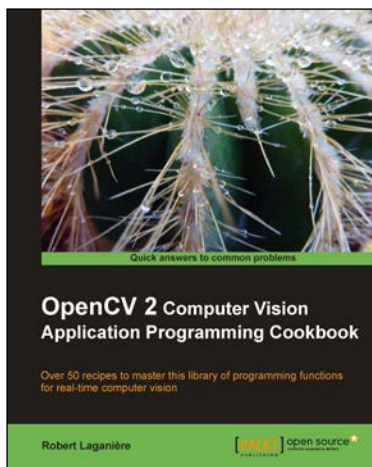


Mastering OpenCV with Practical Computer Vision Projects

ISBN: 978-1-84951-782-9 Paperback: 340 pages

Step-by-step tutorials to solve common real-world computer vision problems for desktop or mobile, from augmented reality and number plate recognition to face recognition and 3D head tracking

1. Allows anyone with basic OpenCV experience to rapidly obtain skills in many computer vision topics, for research or commercial use
2. Each chapter is a separate project covering a computer vision problem, written by a professional with proven experience on that topic
3. All projects include a step-by-step tutorial and full source-code, using the C++ interface of OpenCV



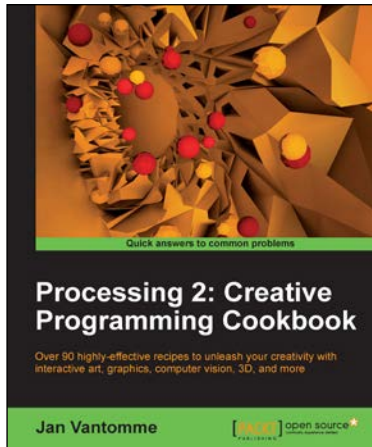
OpenCV 2 Computer Vision Application Programming Cookbook

ISBN: 978-1-84951-324-1 Paperback: 304 pages

Over 50 recipes to master this library of programming functions for real-time computer vision

1. Teaches you how to program computer vision applications in C++ using the different features of the OpenCV library
2. Demonstrates the important structures and functions of OpenCV in detail with complete working examples
3. Describes fundamental concepts in computer vision and image processing

Please check www.PacktPub.com for information on our titles

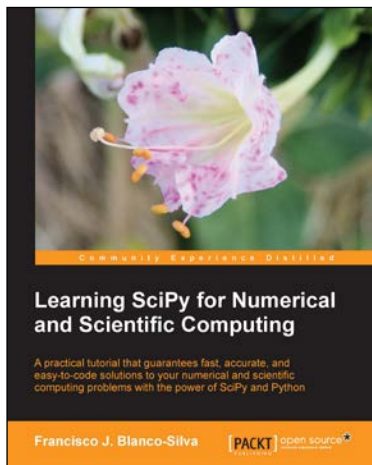


Processing 2: Creative Programming Cookbook

ISBN: 978-1-84951-794-2 Paperback: 306 pages

Over 80 recipes for creating native mobile applications specifically for iPhone and Android smartphones - no Objective-C or Java required

1. Explore the Processing language with a broad range of practical recipes for computational art and graphics
2. Wide coverage of topics including interactive art, computer vision, visualization, drawing in 3D, and much more with Processing
3. Create interactive art installations and learn to export your artwork for print, screen, Internet, and mobile devices



Learning SciPy for Numerical and Scientific Computing

ISBN: 978-1-78216-162-2 Paperback: 150 pages

A practical tutorial that guarantees fast, accurate, and easy-to-code solutions to your numerical and scientific computing problems with the power of SciPy and Python

1. Perform complex operations with large matrices, including eigenvalue problems, matrix decompositions, or solution to large systems of equations
2. Step-by-step examples to easily implement statistical analysis and data mining that rivals in performance any of the costly specialized software suites
3. Plenty of examples of state-of-the-art research problems from all disciplines of science, that prove how simple, yet effective, is to provide solutions based on SciPy

Please check www.PacktPub.com for information on our titles