



Community Experience Distilled

# Emgu CV Essentials

Develop your own computer vision application using the power of Emgu CV

**Shin Shi**

**[PACKT]**  
PUBLISHING

# Emgu CV Essentials

Develop your own computer vision application using the power of Emgu CV

**Shin Shi**



BIRMINGHAM - MUMBAI

# Emgu CV Essentials

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 author, 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: November 2013

Production Reference: 1071113

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

ISBN 978-1-78355-952-7

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Aniket Sawant ([aniket\\_sawant\\_photography@hotmail.com](mailto:aniket_sawant_photography@hotmail.com))

# Credits

**Author**

Shin Shi

**Project Coordinator**

Suraj Bist

**Reviewers**

Yashaswita R. Bhoir

Luca Del Tongo

**Proofreader**

Faye Coulman

**Acquisition Editors**

Saleem Ahmed

Antony Lowe

**Indexer**

Priya Subramani

**Commissioning Editor**

Neil Alexander

**Graphics**

Ronak Dhruv

Abhinash Sahu

**Technical Editors**

Shiny Poojary

Faisal Siddiqui

Sonali S. Vernekar

**Production Coordinator**

Arvindkumar Gupta

**Cover Work**

Arvindkumar Gupta

**Copy Editors**

Alisha Aranha

Mradula Hegde

Dipti Kapadia

Gladson Monteiro

Sayanee Mukherjee

Aditya Nair

Alfida Paiva

Adithi Shetty

# About the Author

**Shin Shi** is a member of the Digital Art Lab at Shanghai Jiao Tong University. When he built his first robot at the age of fourteen in 2005, he became interested in the field of computer vision. Over the last two years, he created a real-time face recognition attendance system using Emgu CV at Huazhong University of Science and Technology. He obtained a B.E. in Software Engineering from HUST and is currently an M.S. candidate in DA Lab from SJTU. Now he is focusing on the field of computer graphics, such as creating physics-based simulations and the use of animation.

---

Foremost, I would like to give a very special thanks to my soul mate, Licheng Yin, for his invaluable support and patience. And I would like to express my sincere gratitude to Prof. ZP Qin, for his motivation and immense knowledge.

Last but not least, I would like to thank my family for supporting me spiritually throughout my life.

---

# About the Reviewers

**Yashaswita R. Bhoir** is a full-time working professional, employed in an MNC and currently working on various technologies related to J2EE and application servers. She had her share of experience working on Emgu CV for a Human-Computer Interaction based project for a year. In addition to various projects on Emgu CV, she has a few conference presentations to her name.

When not working on something or the other, Yashaswita likes spending her time cooking, analyzing political affairs or re-reading the Harry Potter series. You can get in touch with her on [ykombinator@gmail.com](mailto:ykombinator@gmail.com)

**Luca Del Tongo** is a computer engineer with a strong passion for algorithms, computer vision, and image-processing techniques. He's the co-author of a free e-book called "Data Structures and Algorithms (DSA)" with 100k downloads and has published several image-processing tutorials on his YouTube channel, using Emgu CV. During his Masters' thesis, he developed forensic algorithm for use on images, published in a scientific paper called "Copy Move forgery detection and localization by means of robust clustering with J-Linkage". Currently, he works as a software engineer in the ophthalmology field developing corneal topography-processing algorithms and IOL calculations. He loves practicing sports and follows MOOC courses in his spare time. You can contact him through his blog at <http://blogs.ugidotnet.org/wetblog>.

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://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](http://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](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://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.



<http://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](http://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

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Introduction to Emgu CV</b>	<b>5</b>
<b>What is Emgu CV?</b>	<b>5</b>
<b>Comparing image-processing libraries</b>	<b>6</b>
License agreement	6
Documentation and other material	7
Ease of use	7
Performance	8
Summary of the comparison	9
<b>Advantages of Emgu CV</b>	<b>9</b>
Cross-platform	9
Cross-language support with examples	9
Other advantages	10
<b>Summary</b>	<b>10</b>
<b>Chapter 2: Installing Emgu CV</b>	<b>11</b>
<b>Downloading Emgu CV</b>	<b>11</b>
<b>Installing Emgu CV</b>	<b>11</b>
Installing on Windows	11
Installing on Linux	16
Getting the dependency	16
Building Emgu CV from source	17
Installing on OS X	18
Getting the dependency	19
Building Emgu CV from source	19
<b>Troubleshooting</b>	<b>20</b>
Windows	20
Linux	21
OS X	22
<b>Summary</b>	<b>22</b>



---

<b>Chapter 3: Hello World</b>	<b>23</b>
<b>Hello World in C#</b>	<b>23</b>
Creating a new project	24
Designing our form	27
Coding	27
Output	29
<b>Hello World in VB.NET</b>	<b>30</b>
<b>Hello World in C++</b>	<b>31</b>
<b>Summary</b>	<b>32</b>
<b>Chapter 4: Wrapping OpenCV</b>	<b>33</b>
<b>Architecture overview</b>	<b>33</b>
OpenCV	33
Emgu CV	34
<b>Function mapping</b>	<b>36</b>
<b>Structure mapping</b>	<b>36</b>
<b>Enumeration mapping</b>	<b>37</b>
<b>Summary</b>	<b>37</b>
<b>Chapter 5: Working with Images</b>	<b>39</b>
<b>Digital image representation</b>	<b>39</b>
Pixels and data	39
Pixel resolution	40
Color image representation	41
Color depth	43
<b>Working with images</b>	<b>44</b>
Creating an image	44
Loading an image from a file	46
Operations with pixels	47
Method naming rules	49
Using operator overload	50
Generic operations support	51
Garbage collection	51
XML serialization	52
<b>Summary</b>	<b>53</b>
<b>Chapter 6: Working with Matrices</b>	<b>55</b>
<b>Matrix and the Image class</b>	<b>55</b>
<b>Definition and parameters</b>	<b>56</b>
<b>Working with matrices</b>	<b>56</b>
Creating a matrix	57
Operations with elements	58
<b>Summary</b>	<b>59</b>

<b>Chapter 7: Shape Detection</b>	<b>61</b>
<b>Canny Edge Detector</b>	<b>61</b>
<b>Hough transforms</b>	<b>63</b>
Hough Line transform	63
Hough Circle transform	65
<b>Contour</b>	<b>67</b>
Contour finding	68
Representation of contours	68
Sequences of vertexes	68
Free chain codes	69
Drawing contours	69
Polygon approximations	70
A contours example	70
<b>Summary</b>	<b>72</b>
<b>Chapter 8: Face Detection</b>	<b>73</b>
<b>Biometric systems</b>	<b>73</b>
<b>Camera captures</b>	<b>75</b>
<b>Machine learning</b>	<b>76</b>
<b>Face detection or the Haar classifier</b>	<b>77</b>
Boosting theory and supervised learning	78
Haar-like features	78
Code for face detection	81
<b>Summary</b>	<b>83</b>
<b>Chapter 9: License Plate Recognition</b>	<b>85</b>
<b>License Plate Recognition</b>	<b>85</b>
Algorithms for LPR	86
<b>OCR</b>	<b>87</b>
<b>Tesseract-OCR</b>	<b>88</b>
<b>Code for License Plate Recognition</b>	<b>88</b>
Assumption	89
Source code	89
GetWhitePixelMask	90
DetectLicensePlate	91
FindLicensePlate	92
Output	93
<b>Summary</b>	<b>93</b>
<b>Chapter 10: Image Stitching</b>	<b>95</b>
<b>Image stitching</b>	<b>95</b>
<b>Algorithms for image stitching</b>	<b>96</b>
Image matching	96

*Table of Contents*

---

Image calibration	97
Image blending	97
<b>Code</b>	<b>98</b>
<b>Summary</b>	<b>99</b>
<b>Index</b>	<b>101</b>

---

# Preface

This book serves as a simple working guide to the Emgu CV library, which is a .NET wrapper to the OpenCV image-processing library. Main features and sample codes are provided for understanding Emgu CV and a general background of the computer vision field is covered.

## What this book covers

*Chapter 1, Introduction to Emgu CV*, gives a brief introduction to Emgu CV and provides a comparison of different libraries.

*Chapter 2, Installing Emgu CV*, is an installation reference for Emgu CV users.

*Chapter 3, Hello World*, is a guide to creating our first Emgu CV project. We also see that Emgu CV can be used with several different languages.

*Chapter 4, Wrapping OpenCV*, examines and compares the architectures of OpenCV and Emgu CV.

*Chapter 5, Working with Images*, will explain the basic concept about images and how to deal with them.

*Chapter 6, Working with Matrices*, will explain what matrices are and how to deal with them.

*Chapter 7, Shape Detection*, will introduce how to transform an image into a representation of data.

*Chapter 8, Face Detection*, will explain the basis of machine learning and face detection.

*Chapter 9, License Plate Recognition*, shows how to do a license plate recognition and optical character recognition in C# using Emgu CV.

*Chapter 10, Image Stitching*, will explain image stitching, which is very easy to use, but essential to Emgu CV.

## What you need for this book

The chapters in this book assume that you have basic knowledge about C#. It will be better if you also know something about OpenCV. If you are new to computer vision, this book offers a friendly guidance. Visual Studio and Emgu library are required in your computer. However, if you don't know them, don't worry because they will be explained in the chapters.

Essential tools:

- Visual Studio 2008 or later
- Emgu CV

## Who this book is for

If you are a C# programmer and working on the projects about computer vision, this book is just for you. You should have some prior exposure to C#, at least through basic examples. This book contains basic introductions and working coded examples that will help you enjoy the fun of computer vision.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "For Windows, the latest source distribution is the `libemgucv-windows-universal-gpu-xxxx.exe` file."


A block of code is set as follows:


```
using Emgu.CV;  
using Emgu.CV.CvEnum;  
using Emgu.CV.Structure;
```

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

```
cmake -DBUILD_NEW_PYTHON_SUPPORT:BOOL=FALSE -DWITH_TBB:BOOL=TRUE -DBUILD_TESTS:BOOL=FALSE -DBUILD_DOCS:BOOL=FALSE.
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

[  Warnings or important notes appear in a box like this. ]

[  Tips and tricks appear like this. ]

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code and graphics

You can download the example code files for all Packt books and the colored graphics of this book you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

# 1

## Introduction to Emgu CV

Welcome to the world of Emgu CV. Nowadays, developers have to deal with lots of media files such as images and videos. They can hardly start from scratch in defining every operation, so the usual solution is to find an image-processing library to improve the efficiency. Emgu CV is just a new library that can give us a helping hand. In this chapter, we will get to know it.

### What is Emgu CV?

Emgu CV is a cross-platform image-processing library. It is closely related to OpenCV because Emgu CV is a .NET wrapper to OpenCV. We can say Emgu CV is OpenCV in .NET. The amazing wrapper makes it possible for OpenCV functions to be called from .NET programming languages. C#, VB, IronPython, and VC++ are some of the languages supported. Emgu CV can be compiled in Mono, and it runs under Linux, Windows, Mac OS X, and popular mobile platforms such as Android devices, iPhone, iPod Touch, and iPad.



OpenCV, developed by Intel, is a library of programming functions. It's used for real-time computer vision. Developers can use it freely under the open source BSD license. It is written in optimized C and C++ and contains over 500 functions, which span different areas in computer vision. Also, it can take advantage of multicore processors to keep the computational efficiency. But it is not so friendly to the .NET programmers.


Because of the fact that .NET is an interpreted framework that cannot directly call functions or methods written in native C or C++, .NET developers can use Emgu CV to solve the problem. It is possible using P/Invoke and a lot of hand-made plumbing code to call C++ from C#. But Emgu CV makes it more transparent to the user.



One of Emgu CV's goals is to provide a simple-to-use computer vision infrastructure for .NET programmers that helps them build fairly sophisticated vision applications quickly. The Emgu CV library spans many areas in vision, including factory product inspection, medical imaging, user interfaces, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand in hand, Emgu CV also wraps a full, general-purpose machine learning library from the OpenCV image-processing library.

## Comparing image-processing libraries

At present, there are some well-known and fully functional image-processing libraries, such as OpenCV, Emgu CV and AForge.NET. This part gives a brief comparison of these image-processing libraries with regards to license agreement, documentation, ease of use, performance, and so on.

[  AForge.NET is an artificial intelligence and computer vision library originally developed by *Andrew Kirillov* for the .NET framework. ]

## License agreement

The following table shows the comparison of the ease of using licenses' agreement:

Library	License	Brief Introduction
OpenCV	BSD	You can freely use it in any application but remember to keep the BSD license file in it
Emgu CV	GPLv3	Your product or application must be open source and free with GPL license on it
	Commercial License	Application can be closed source after you pay for the commercial license
AForge.NET	LGPLv3	Product can be closed source if you do not change the source code of the library

## Documentation and other material

The following table shows the comparison of documentation and other material:

Library	Book	Website	Documentation	Sample	Forum
OpenCV	Many	Multilanguage	Multilanguage	Many	Hot
Emgu CV	Few	English	English	Less	Poor
AForge.NET	Few	English	English	Many	Poor

Since its alpha release in January 1999, OpenCV has been used in many applications, products, and research efforts. So, it is easier for a beginner to find plenty of tutorials. Emgu CV is a .NET wrapper to the OpenCV image-processing library. Maybe we cannot find many samples of Emgu CV, but a better understanding of OpenCV also helps a lot. These two are just like brothers and share the same spirit. The AForge.NET computer vision framework is provided not only with different libraries and their sources but also with a lot of sample applications, which demonstrate the use of the library. AForge.NET documentation's help files can be downloaded in HTML help format.

## Ease of use

The following table shows the comparison of ease of use:

Library	Ease of use
OpenCV	Not so friendly
Emgu CV	Good
AForge.NET	Very good

Most of OpenCV's functions are provided in C style, but a few of them have a C++ class. After the 2.0 edition, more features have been encapsulated into the C++ classes. Emgu CV is a .NET wrapper to the OpenCV, which encapsulated most of the OpenCV features. AForge.NET is a pure .NET library, which means it is more user-friendly.

Ease of use differs from user ability and user experience. If the developer is more fluent in C and C++, OpenCV also can be a better choice.

## Performance

An image processing library can do plenty of things. Here, we select the most basic functions to test their performance. One is grayscale processing and the other is binarization. These two algorithms can be considered to be representatives of the average image-processing operations because most of the work in image processing involves memory read/write and matrix operations.

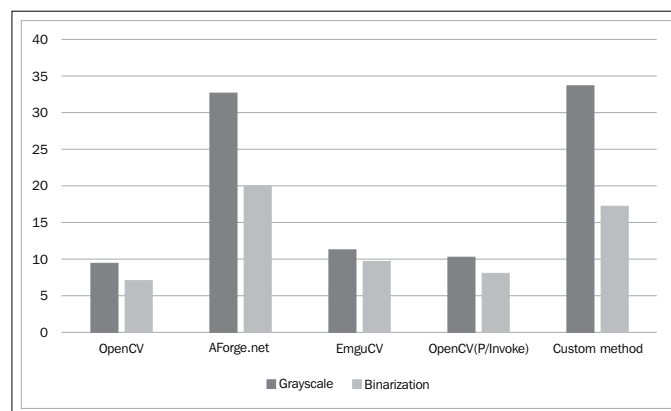
Here, we use five ways to achieve grayscale processing and binarization:

- Call OpenCV functions with C
- Call AForge.NET functions with C#
- Call Emgu CV with C#
- Use P/Invoke to call OpenCV in C# code
- Custom processing method in C#

Record the time and calculate the average of ten times of the execution period. Each method will get a runtime. The result is shown in the following table.

Code	Library	Grayscale (ms)	Binarization (ms)	Rank
C	OpenCV	9.3432	6.9332	1
C#	AForge.NET	32.6548	19.8743	5
C#	Emgu CV	11.2369	9.6853	3
C#	OpenCV (P/Invoke)	10.2355	8.0332	2
C#	Custom method	33.6742	17.2009	4

This data can be graphically represented to make things clearer, as shown in the following diagram:



This result comes from an X86 computer with Pentium 4 processor and 512 MB RAM. The obvious conclusion is that OpenCV with C runs the best. The two pure C# methods cost a lot of time. Emgu CV ranks third but does not fall too far behind OpenCV.

## Summary of the comparison

Summary of all the information is shown in the following table:

Library	OpenCV	Emgu CV	AForge.NET
License	BSD	GPLv3 or Commercial	LGPLv3
Documentation	Great	Good	Poor
Ease of use	Poor	Good	Great
Performance	Awesome	Great	Very poor

In summary, Emgu CV is the better image-processing library than the other two, especially for a C# programmer. Emgu CV improves the ease of use and readability, which makes it more transparent to the user in managed C# code.

## Advantages of Emgu CV

The following are the major advantages to using Emgu CV.

### Cross-platform

Emgu CV is totally written in C#. The advantage it brings is that it is compatible with Mono so that our Emgu CV program can be launched on any platform with Mono, such as Linux PC, Android, iOS, and Mac OS X. Headers can be simply included in managed C++ implementation, but great effort has been taken to build this pure C# implementation to achieve cross-platform. However, it's still quite considerable if Emgu CV is able to run on your Ubuntu or Fedora PC, let alone the fact that you'll always be comfortable knowing that your code is compatible and cross-platform.

### Cross-language support with examples

Emgu CV is able to run in multiple language environments, including VB.NET, IronPython, C#, and C++. Several instances have been provided on the official website for all those languages, which can be accessed from the **Examples** section on the **Tutorial** page (<http://www.emgu.com/wiki/index.php/Tutorial>). You can also visit the Emgu CV Discussion Forum (<http://www.emgu.com/forum>) to solve your language-related questions.

## Other advantages

Some other advantages are as follows:

- Provides generic color and depth image class
- Automatic memory management (garbage collection)
- XML-serializable image
- Both, invoking the image class and using OpenCV functions are directly supported
- Generic operations are provided for image pixels

## Summary

This chapter gives a brief introduction to Emgu CV and then provides a comparison of different image-processing libraries. From now on, we should know why to choose Emgu CV. In the next chapter, we are going to install Emgu CV step-by-step.

# 2

## Installing Emgu CV

This chapter serves as a quick installation reference for users new to Emgu CV.

### Downloading Emgu CV

The main Emgu CV site on SourceForge is <http://sourceforge.net/projects/emgucv/>. For Windows, the latest source distribution is the `libemgucv-windows-universal-gpu-xxxx.exe` file. (Here, `xxxx` is the edition number, but if you want to download the old release, you will see a different pattern for filename.) For Linux, the extension you need is `tar.gz`. For OS X and Linux, it's better to build the library file from the source.

### Installing Emgu CV

Once you download the libraries, you must install them. This section shows a brief way to install Emgu CV on Windows, Linux, or Mac OS X.

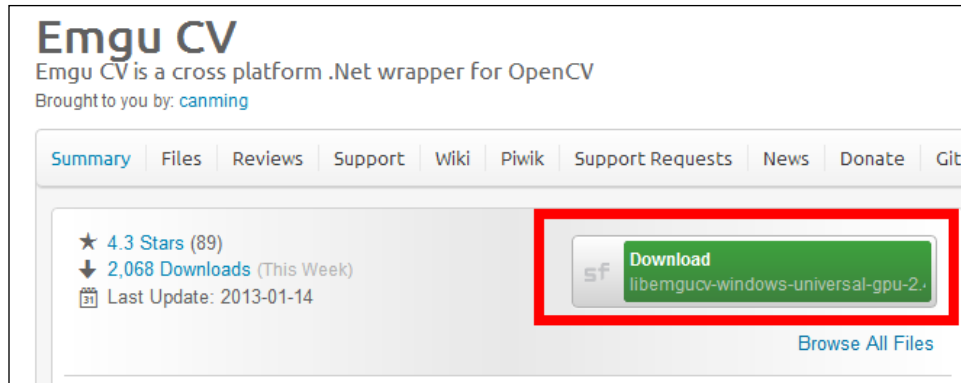
### Installing on Windows

For installation on Windows, some essential tools must be downloaded. We will be using the latest version, though it may have some issues. Using the latest version will get a quick support as we come across a problem.


The essential tools are:

- Visual Studio 2008 or later
- Emgu CV

We can visit the Emgu CV site on SourceForge to download the Windows executable file to start a quick installation. As shown in the following screenshot, click on the **Download** button to download a package. The file name is `libemgucv-windows-universal-gpu-2.4.9.1847.exe`.

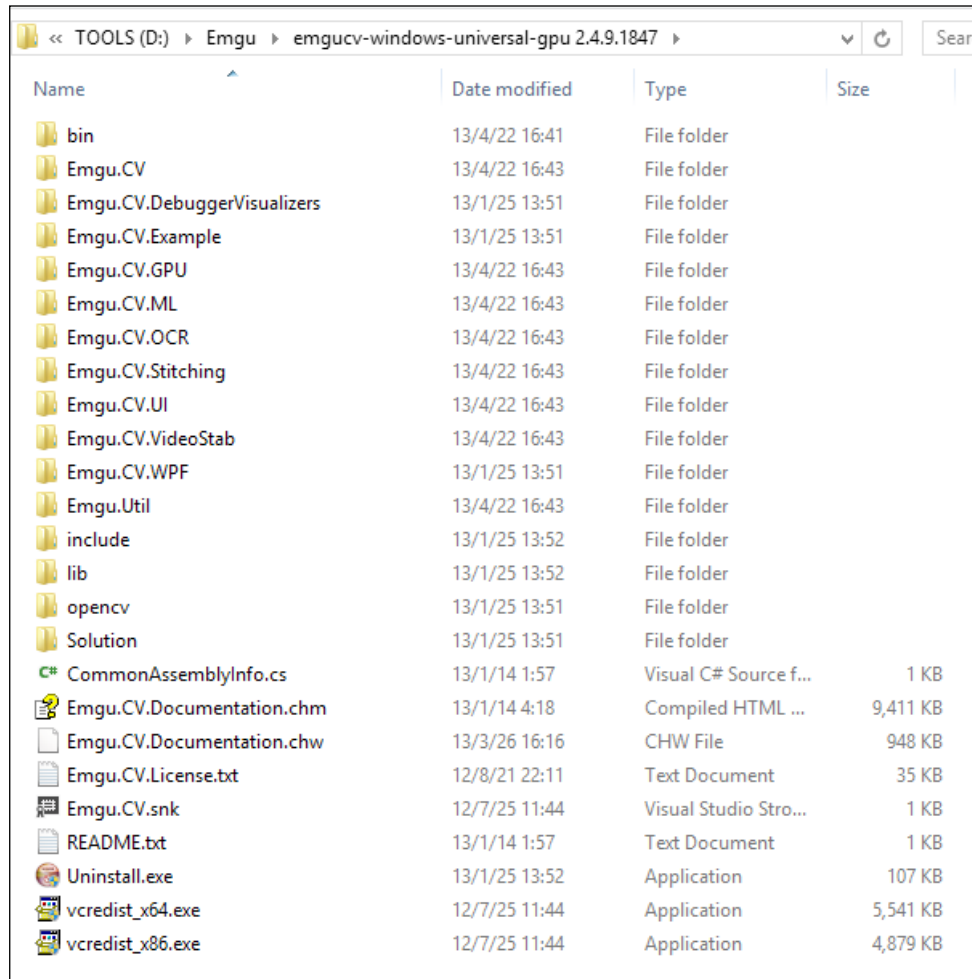


The latest update was on **2013-1-14**, as shown in the preceding screenshot. The edition number should be 2.4.9.1847. This is an alpha version with many new features. We can also choose a stable one like 2.3.0, which was released on 2011-08-20. Here, we download the latest version to do all the work in this book:

 <code>libemgucv-windows-universal-gpu-2.4.9.1847.exe</code>	Application	223,283 KB
---	-------------	------------

Open the executable file and we can continue with the installation. This operation needs a UAC confirmation. All the Emgu CV package files need 1.8 GB space on your disk.

Here we set the destination folder as `D:\Emgu\emgucv-windows-universal-gpu 2.4.9.1847`. All the files and folders are shown in the following screenshot. All the things in the screenshot will be explained later:



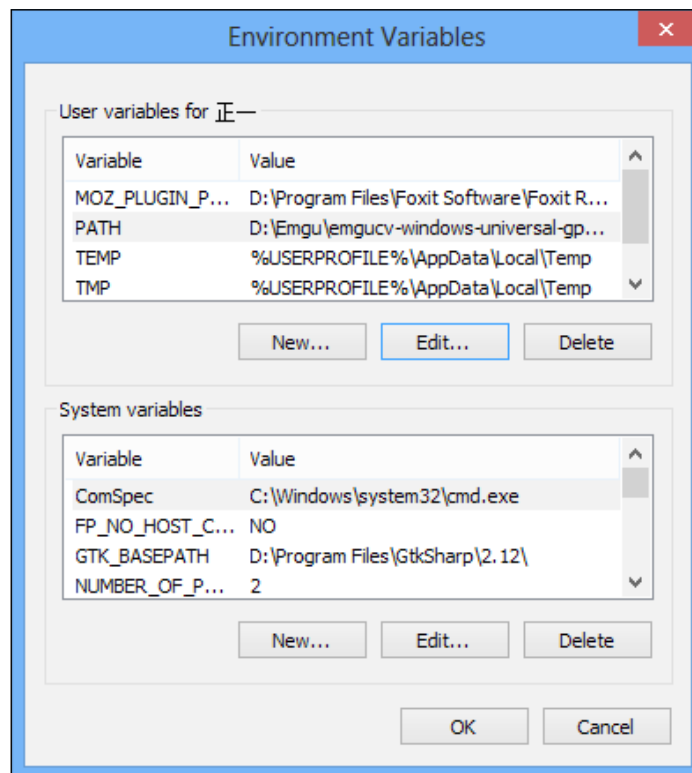


Second, we will configure the Emgu CV. We need to add a new system environment variable to our Windows. We have to do this because it affects the way our Emgu CV programs will behave. For example, `PATH` specifies the directories in which our libraries are located so that our program can be executed successfully.

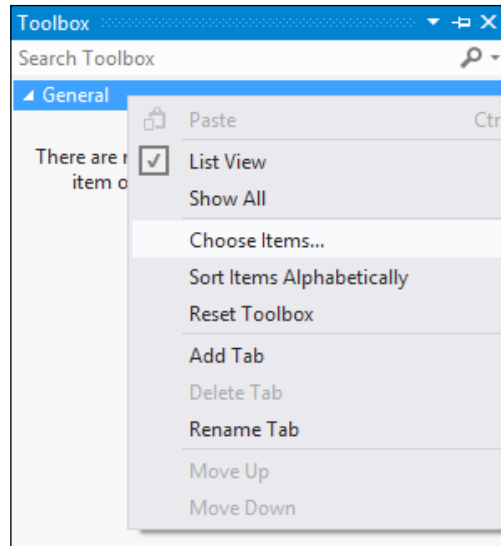


Here my PC runs Windows 8 Pro 32 bit as the example. It also has Visual Studio 2012 Ultimate. If your computer runs Windows 7 and another edition of Visual Studio, every step given here will also be fine. If you are using a 64-bit OS, select the x64 one instead of x86 folder.

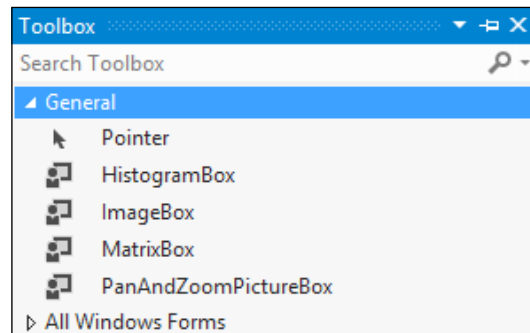
1. On the desktop, right-click on the **Computer** icon and choose the **Properties** option. Click on **Advanced system settings** in the left-hand side pane. Select the **Advanced** tab, then click on the **Environment Variables...** button at the bottom. We need to change the **PATH** variable. Carefully add a semicolon (;) and the path to the Emgu CV bin folder (by what we set here, it should be `D:\Emgu\emgucv-windows-universal-gpu 2.4.9.1847\bin\x86`) as shown in the following screenshot:



2. Now the system environment is okay, and we are going to add the useful Emgu custom controls to our Visual Studio toolbox. Right-click on the **General** tab and select **Choose Items...** in **Toolbox** as shown in the following screenshot:



3. Click on the **Browse...** button at the bottom and browse for `Emgu.CV.UI.dll`. (This file is also located in the Emgu bin folder and the path will be `D:\Emgu\emgucv-windows-universal-gpu 2.4.9.1847\bin\`.)
4. Now the useful Emgu controls are available in our Visual Studio as shown in the following screenshot:



We have successfully set up the basic environment on the Windows PC for working with Emgu CV. More details about dependencies will be introduced in this chapter later.

## Installing on Linux

The following steps are taken from the official site. As Emgu CV is good for cross-platform development, it is quite easy to develop Emgu solutions using Mono on our Linux PC. If your computer runs Fedora or Ubuntu, the steps to set up the environment are given in the following sections.

## Getting the dependency

Prebuilt files for Linux are not included with the Linux version because some of the dependencies, such as GCC and Mono, are too large. We have to get the dependency first. Here are the steps and command lines.

### Fedora

The installation process for Fedora is a multi-stage process starting with the installation of each separate module, which is as follows:

- **OpenCV:** A custom version of OpenCV will be built in the following step. If you have installed any version of OpenCV on your machine, I recommend you to remove it first. By running as root, type the following command line to remove it:  
`yum remove opencv`
- **Mono:** Mono is required on our Linux machine. It is necessary because Emgu CV needs **WCF (Windows Communication Foundation)** when it is running. Mono 2.6 or later will be okay.



[ Mono is an open source implementation of .NET. For more information, see [http://www.mono-project.com/Main\\_Page](http://www.mono-project.com/Main_Page). ]

- **CMake:** CMake is used to compile Emgu CV and OpenCV from its source code.
- **Installing all the prerequisites:** We will need root authority to install all the prerequisites. Now input the following command line and press *Enter*:

```
yum install svn gcc-c++ cmake gtk2-devel libpng-devel libjpeg-  
turbo-devel jasper-devel libtiff-devel libgeotiff-devel OpenEXR-  
devel qt-devel libv4l-devel eigen2-devel libdc1394-devel  
tesseract-devel tbb-devel mono-core mono-extras mono-devel mono-  
wcf
```

## Ubuntu

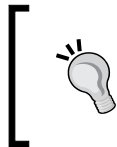
The installation process for Ubuntu is a multi-stage process starting with the installation of each separate module, which is as follows:

- **OpenCV:** A custom version of OpenCV will be built in the following step. If you have installed any version of OpenCV on your machine, I recommend you to remove it first.
- **Mono:** Mono 2.6 or later is required here. Emgu CV library uses WCF so that it also requires `libmono-wcf3.0-cil`.
- **CMake:** CMake is used to compile Emgu CV and OpenCV from its course code.
- **Installing the prerequisites:** Run the following command line to install the prerequisites:

```
sudo apt-get install git cmake-curses-gui libgtk2.0-dev build-essential libtiff5-dev libgeotiff-dev libgstreamer0.10-dev libswscale-dev libavcodec-dev libavformat-dev libjasper-dev libopenexr-dev libv4l-dev libqt4-opengl-dev libeigen2-dev libdc1394-22-dev libtbb-dev libtesseract-dev monodevelop mono-gmcs libmono-wcf3.0-cil
```

## Building Emgu CV from source

Emgu CV needs to be built on our Linux machine. Before we build it, a Git client is required because it is used to check out the source code from SourceForge.



For software developers, Git is a well-known source code management and distributed version control system. When you are using Git, collaborative software development is supported and changes of the source code can be tracked over time.

The steps to be performed are as follows:

1. Start to check out the source by typing the following command:  
`git clone git://git.code.sf.net/p/emgucv/code emgucv`
2. After that, go to the directory where Emgu CV is located:  
`cd emgucv`
3. Initialize OpenCV and some submodules:  
`git submodule update --init --recursive`

4. Use the `cmake` command to generate the make file:  


```
cmake -DBUILD_NEW_PYTHON_SUPPORT:BOOL=FALSE -DWITH_TBB:BOOL=TRUE  
-DBUILD_TESTS:BOOL=FALSE -DBUILD_DOCS:BOOL=FALSE.
```
5. Emgu CV will be generated and then use the `make` command to build Emgu CV:  

```
make
```
6. It takes a few minutes to build OpenCV and Emgu CV after calling that command. Essential files such as `Emgu.CV.dll`, `Emgu.Util.dll`, `Emgu.CV.ML.dll`, and `Emgu.CV.UI.dll` will be created.
7. Then open the `bin` folder:  

```
cd bin
```
8. By typing the following command, we can make sure that Mono will load the DLLs from the current location:  

```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```
9. The last step is to try running an example:  

```
mono Example.PlanarSubdivision.monoexe
```
10. A planar subdivision window will be shown if you were successful.

 As the environment has been set up by `cmake`, we can compile with `Monodevelop`. If you want to go further on your Linux machine, check out: [http://www.emgu.com/wiki/index.php/Compiling\\_with\\_Monodevelop](http://www.emgu.com/wiki/index.php/Compiling_with_Monodevelop).

Now we have succeeded in setting up the environment for working with Emgu CV on a Linux machine. If you encounter any exceptions, solutions will be introduced in this chapter later.

## Installing on OS X

The steps for installation on OS X are almost the same as in the *Installing on Linux* section.

## Getting the dependency

We need to get the dependencies for each separate module, which is as follows:

- **Mono:** Mono is required on our Mac. It is necessary because Emgu CV needs WCF (Windows Communication Foundation) when it is running. Mono 2.6 or later will be okay.
- **CMake:** Make sure we have CMake installed because CMake is used to compile Emgu CV and OpenCV from its source code. We have to build from source to generate all the library files.
- **Xcode:** Xcode and command-line tools are needed. Xcode is a powerful IDE containing a suite of development tools for your development for OS X and iOS. It can be the easiest way to develop your Emgu solutions.

## Building Emgu CV from source

Now we need to build Emgu CV on our Mac. The steps are as follows:

1. Before we build Emgu CV, Git client is required because it is used to check out the source code from SourceForge:  

```
git clone git://git.code.sf.net/p/emgucv/code emgucv
```
2. After that, go to the directory where Emgu CV is located:  

```
cd emgucv
```
3. Initialize OpenCV and some submodules:  

```
git submodule update --init --recursive
```
4. Use the `cmake` command to generate the make file:  

```
cmake -DBUILD_PERF_TESTS=FALSE -DBUILD_NEW_PYTHON_
SUPPORT:BOOL=FALSE -DCMAKE_OSX_ARCHITECTURES=i386 -DBUILD_
TESTS:BOOL=FALSE -DBUILD_DOCS:BOOL=FALSE -DBUILD_PNG=TRUE -DBUILD_
JPEG=TRUE -DBUILD_TIFF=TRUE .
```
5. Call `make` to build Emgu CV:  

```
make
```
6. It will take a few minutes to build OpenCV and Emgu CV after calling that. Essential files such as `Emgu.CV.dll`, `Emgu.Util.dll`, `Emgu.CV.ML.dll`, and `Emgu.CV.UI.dll` will be created.

7. Then open the bin folder:

```
cd bin
```

8. The last step is to try running an example:

```
mono Example.PlanarSubdivision.monoexe
```

9. A planar subdivision window will be shown if this is successful.

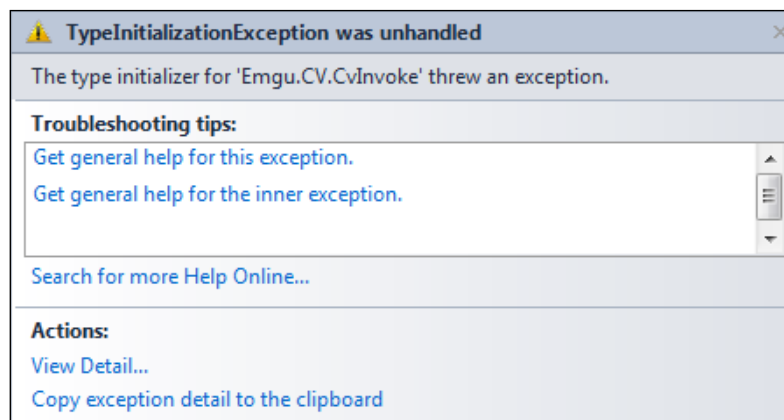
Now we have succeeded in setting up the environment for working with Emgu CV on a Mac. If you encounter any exceptions, solutions will be introduced in this chapter later.

## Troubleshooting

This part lists several basic Emgu troubleshooting steps.

### Windows

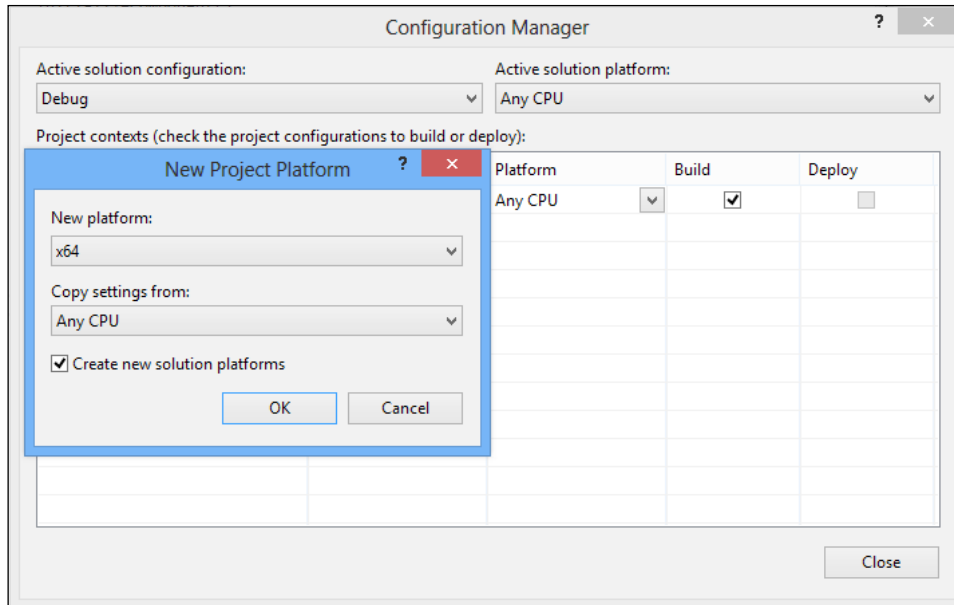
One of the most common issues is **The type initializer for 'Emgu.CV.CvInvoke' threw an exception** as shown in the following screenshot:



If you see this exception, please check the following:

- Make sure you have installed the MS Visual C++ Redistributable Package. If you haven't, go to the Emgu CV folder and open `vcredist_x86.exe` (for 32-bit OS) or `vcredist_x64.exe` (for 64-bit OS).
- Make sure you have copied all the OpenCV DLLs to the execution directory.

- Make sure you are doing everything right with the 32-bit or 64-bit Windows. If you are running 64-bit Windows, make sure you have selected x64 in every step. Go to **BUILD** then **Configuration manager**, and add an **x64** platform as shown in the following screenshot:



- Make sure you don't miss any dependency. You can download the Dependency Walker (<http://www.dependencywalker.com/>) to check it. Use it to open the `cvextern.dll` file to get the result.
- If you still get the exception, try to find out `InnerException` and then report it to the Emgu discussion forum (<http://www.emgu.com/forum/>).

## Linux

One of the most common issues for Linux users is that you may encounter `System.DllNotFoundException` while executing the following command:

```
mono Example.PlanarSubdivision.monoexe
```

That is because the built OpenCV might not be complete. Call the following command to trace the cause:

```
MONO_LOG_LEVEL=debug mono Example.PlanarSubdivision.monoexe
```

Try to check the messages and fix the error.



## OS X

One of the most common issues for Mac users is that you may encounter `System.DllNotFoundException` while executing the following command:

```
mono Example.PlanarSubdivision.monoexe
```

That is because the built OpenCV might not be complete. Call the following command to trace the cause:

```
MONO_LOG_LEVEL=debug mono Example.PlanarSubdivision.monoexe
```

Try to check the messages and fix the error.

## Summary

This chapter is an installation reference for Emgu CV users. Follow the steps to get ready for the next chapter, and we will start our first Emgu CV project. Make sure that everything has been arranged before we start the next chapter.

# 3

## Hello World

After installing the Emgu CV library, our first task is to get started and make something interesting happen. Here, we are going to start our first Emgu CV project. Using Emgu CV is not as difficult as it may seem; all we have to do is add certain references and make use of Emgu controls.

[  We assume that the user has basic knowledge of C# and is able to create a new C# project. ]

Emgu CV is a C# wrapper for OpenCV. Unlike other wrappers, it is written purely in C# and doesn't use unsafe code. Emgu CV opens the OpenCV library of programming functions – mainly for real-time computer vision – to C# programmers.

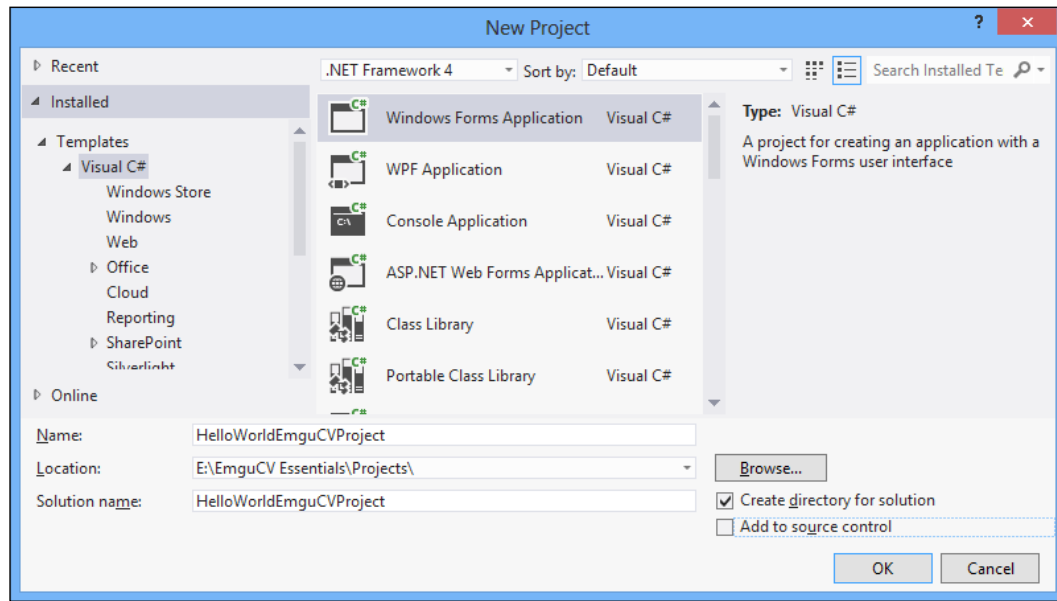
As Emgu CV is a wrapper for C++ code, two main types of **Dynamic-link libraries (DLLs)** are included. The first type is the Emgu CV DLL, with Emgu CV syntax always referenced in the name. The second type of DLLs are the OpenCV ones that vary. Setting up the first project can be a little tough for a beginner, but once you're ready, you can start your first program right away. Let's rock.

## Hello World in C#

As with some C# libraries, we need to reference some essential DLL files within our project. Create a new C# Windows form application and name it whatever you want. Emgu wrapping OpenCV has a natural dependency on its DLLs. All the DLL files mentioned are in the Emgu installation `bin` folder; we must keep this in mind. To start with, we need to create a new project.

## Creating a new project

Let's start creating a new .NET Framework 4 Windows forms application. The page will look like the following screenshot:

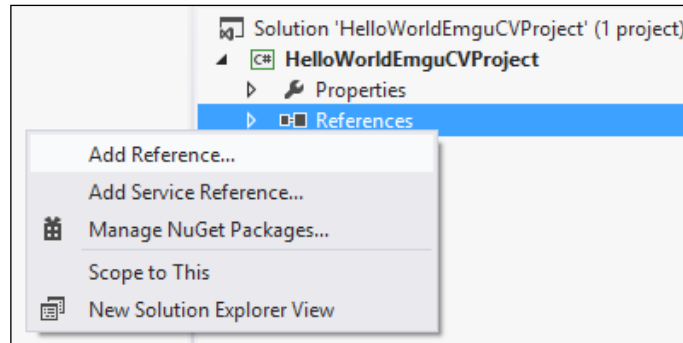


Now we will get a new Visual C# project created for us. The very first thing we must do is add references. As with any C# library, some essential DLLs are required to be referenced within each of our applications. Go to the `bin` folder (the path should be this: `D:\Emgu\emgucv-windows-universal-gpu 2.4.9.1847\bin`); there are several unused DLLs in that folder. All we have to do is add them into our project with the names of the files starting with `Emgu.CV` (select only one among `Emgu.CV.DebuggerVisualizers.VS2012.dll`, `Emgu.CV.DebuggerVisualizers.VS2010.dll`, and `Emgu.CV.DebuggerVisualizers.VS2008.dll`, depending on the edition of Visual Studio on the machine; in this case, we will use Visual Studio 2012, so we choose `Emgu.CV.DebuggerVisualizers.VS2012.dll`).

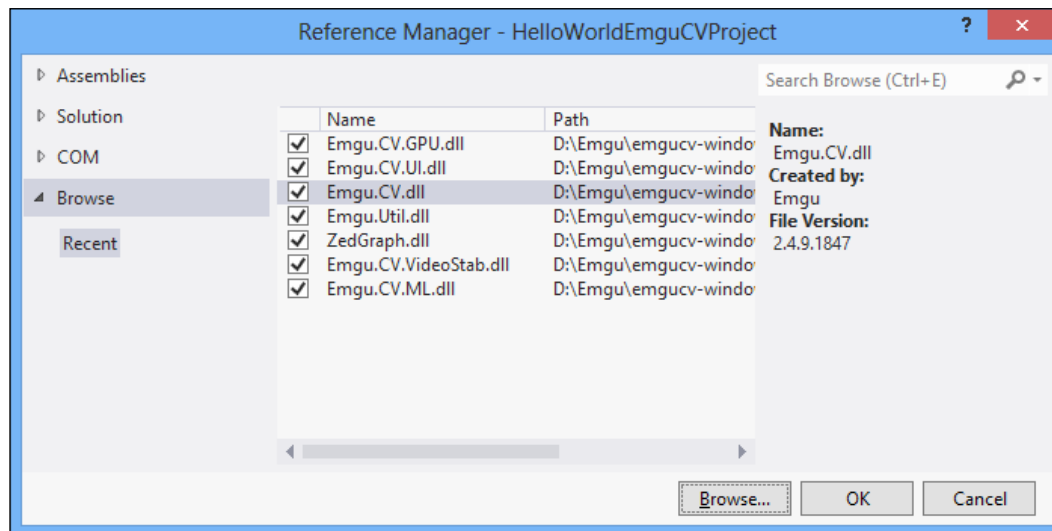


Not all of these DLLs can be called in one Emgu CV project, so we won't need to reference all of them as we gain more experience. Basically, three Emgu CV DLLs must be referenced in any simple Emgu CV project: `Emgu.CV.dll`, `Emgu.CV.Util.dll`, and `Emgu.UI.dll`.

Right-click on the **References** option of our project. We will see the **Add Reference...** as shown in the following screenshot:

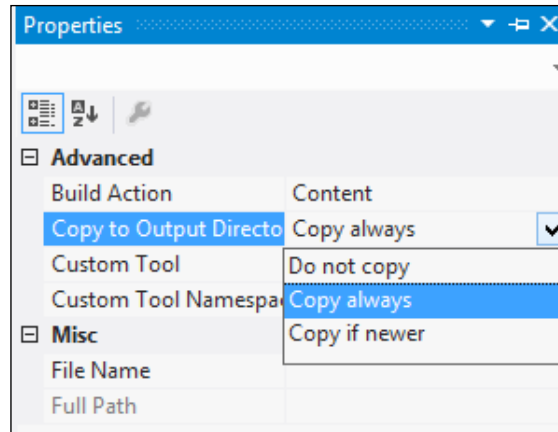



Click on **Add Reference....** We will see **Reference Manager** as shown in the following screenshot:



What we need to do now is to add some existing items. Let's go to **Project** and then **Add Existing Items**. Browse for the **bin** folder of Emgu CV again and this time pick up all the DLLs with names starting with **opencv\_**. These OpenCV DLLs are required each time the program's output is produced via Emgu. That is the reason why we have to add them to our project directory.

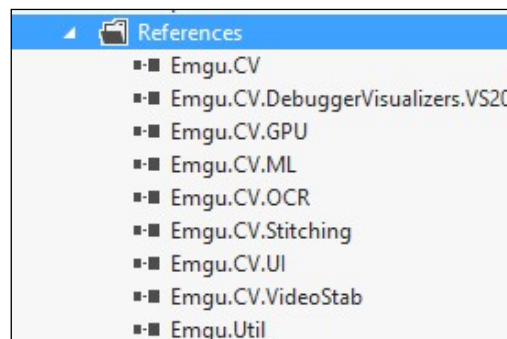
We also need to change the project's properties in such a way that those DLLs can always be copied to the output folder. So make sure all the DLLs are added and go to the **Properties** window and modify the **Copy to Output Directory** option to **Copy always**. This change in options is shown in the following screenshot:



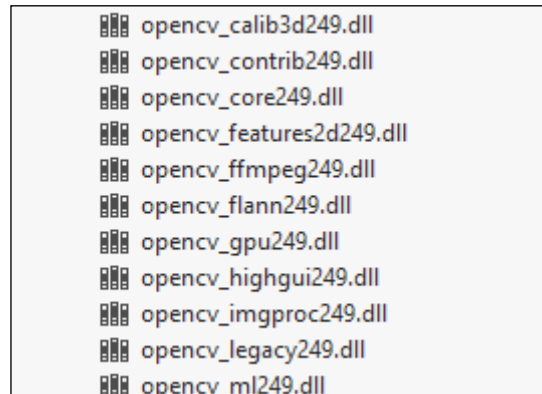
[  Also, not all these OpenCV DLLs can be called in every project; we can select some of them when we know each DLL. Basically, two OpenCV DLLs must be added in any simple Emgu CV project: opencv\_core249.dll and opencv\_imgproc249.dll. ]

If you want to use the x64 compilation to go through your project, options must be modified to allow compiling project to the x64 framework before you start the image processing. After doing this, no error message will jump out, even if you are in the debug or release configuration. All the required files will always be copied and available in the project; that is why we prefer this kind of method here.

Now, in the **Solution Explorer**, we can see references like the following:

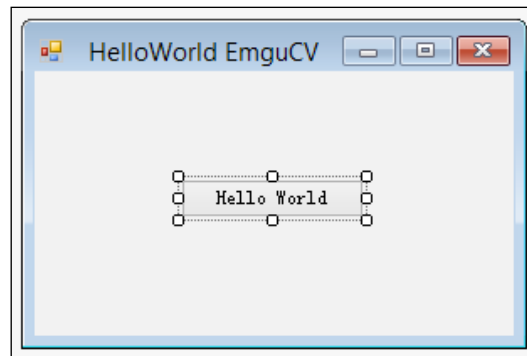


Additionally, we can see the DLLs as shown in the following screenshot:



## Designing our form

Now let's begin to design our first form. It's very simple because this is our first project. We just put a button in the middle of our form. We want a "Hello World" window to pop up when we click on this button, as shown in the following screenshot:



## Coding

Go to the code view of `form1.cs`. The most important thing is to add the following namespaces:

```
using Emgu.CV;  
using Emgu.CV.CvEnum;  
using Emgu.CV.Structure;
```

Next, create a member variable in the class `Form1`.

```
//Set the name of pop-up window
String winname = "First Window";
```

Now let's come to the coding part of the **Hello World** button.

```
private void button1_Click(object sender, EventArgs e)
{
    //Create a window with the specific name
    CvInvoke.cvNamedWindow(winname);

    //Create an image of 480x200 with color yellow
    using (Image<Bgr, Byte> img1 = new Image<Bgr, byte>(480, 200,
        new Bgr(0, 255, 255)))
    {
        //Create a font
        MCvFont font = new
        MCvFont(Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX, 1.0,
            1.0);
        //Draw "Hello, world" on the yellow image; Start point is
        (25, 100) with color blue
        img1.Draw("Hello, world", ref font, new Point(25, 100),
            new Bgr(255, 0, 0));

        //Show the img1 in the window
        CvInvoke.cvShowImage(winname, img1.Ptr);
        //A key pressing event
        CvInvoke.cvWaitKey(0);
        //Destroy the window
        CvInvoke.cvDestroyWindow(winname);
    }
}
```

#### Downloading the example code

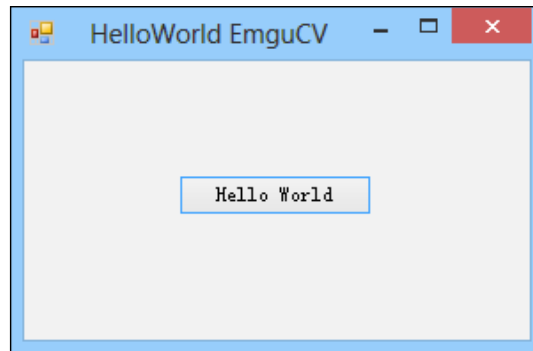


You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

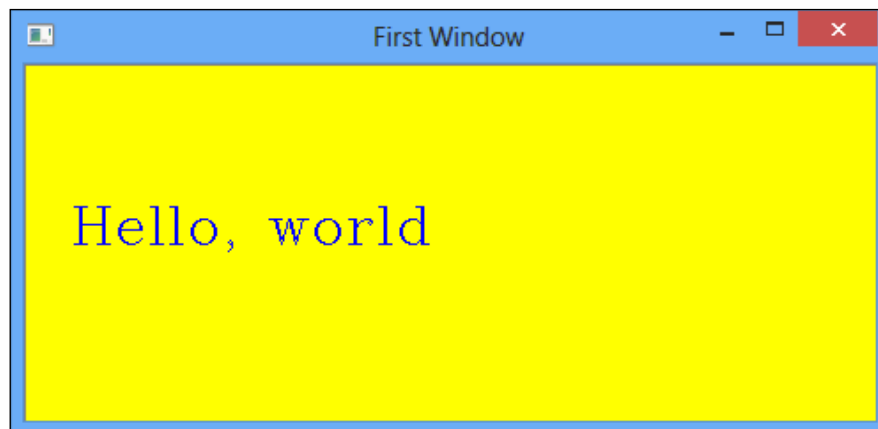
Left-click on **button1**. The event will create a yellow image with the text **Hello World** in blue inside it. The size of the image should be 480 x 200, as we set here. This yellow-blue image should be displayed in the **First Window**.

## Output

The **HelloWorld EmguCV** window will look like the following screenshot:



Click on the button. The **First Window** should pop up as shown in the following screenshot:



The **HelloWorld EmguCV** window will be displayed after compiling. Click on the **Hello World** button and we will get **First Window**.



## Hello World in VB.NET

The Emgu CV library can be called in several programming languages. We can also use VB.NET to do what we did previously. The following VB.NET code is a C# source conversion. This conversion was done manually from C# and it has been smoothly built and executed with Visual Studio 2012 on Windows 8. The process of creating and designing a project is the same as in a C# project. The VB code is given here:

```
Imports Emgu.CV
Imports Emgu.CV.CvEnum
Imports Emgu.CV.Structure

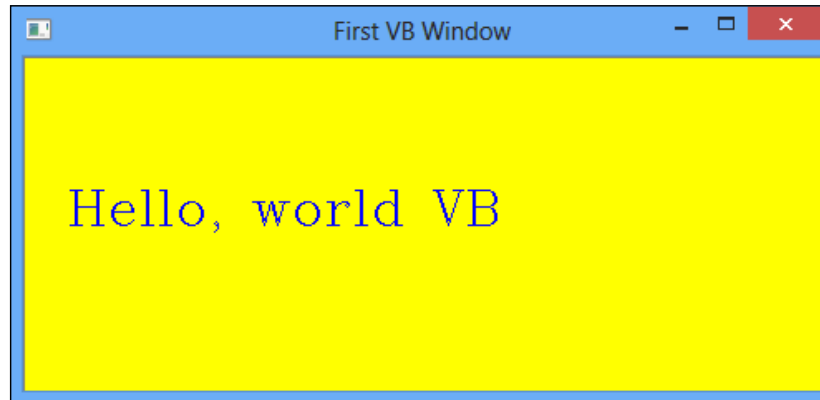
Public Class Form1
    'Set the name of pop-up window
    Dim winname = "First VB Window"
    Private Sub Button1_Click(sender As Object, e As EventArgs)
        Handles Button1.Click
            'Create a window with the specific name
            CvInvoke.cvNamedWindow(winname)

            'Create an image of 480x200 with color yellow
            Using img As Image(Of Bgr, Byte) = New Image(Of Bgr,
                Byte)(480, 200, New Bgr(0, 255, 255))

                'Create a font
                Dim font = New
                    MCvFont(Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX,
                        1.0, 1.0)
                '//Draw "Hello, world VB" on the yellow image; Start
                point is (25, 100) with color blue
                img.Draw("Hello, world VB", font, New Point(25, 100),
                    New Bgr(255, 0, 0))

                'Show the img1 in the window
                CvInvoke.cvShowImage(winname, img.Ptr)
                'A key pressing event
                CvInvoke.cvWaitKey(0)
                'Destroy the window
                CvInvoke.cvDestroyWindow(winname)
            End Using
        End Sub
    End Class
```

The output is as shown in the following screenshot:



## Hello World in C++

We can only compile our Emgu CV C++ projects on Windows, because Mono, the Linux and OSX .NET platform, cannot compile managed C++ code. We can find a project called Hello World written in C++ at `D:\Emgu\emgucv-windows-universal-gpu 2.4.9.1847\Emgu.CV.Example\CPlusPlus`.

Here we will generate a C++/CLR example. Create a new C++/CLR empty project and add references and a new Windows form. Add a button on the form and edit the click event.

The most crucial part of the source code is given here:

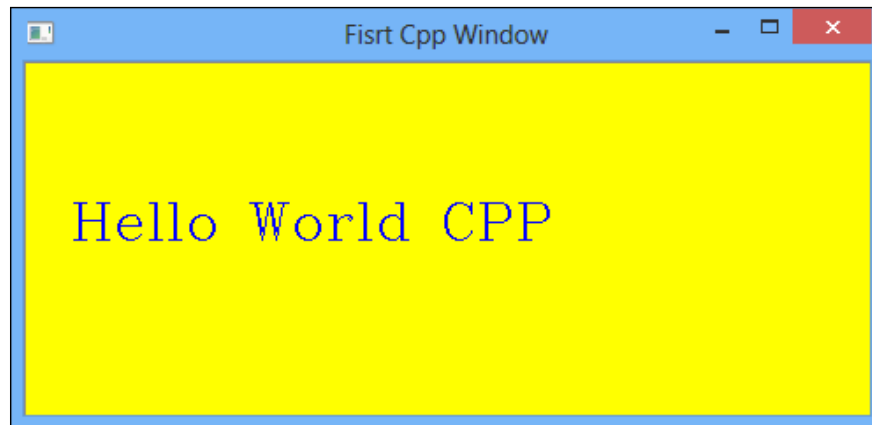
```
String ^ winname = "First Cpp Window";
//Create a window with the specific name
CvInvoke::cvNamedWindow(winname);
//Create an image of 480x200 with color yellow
Emgu::CV::Image<Bgr,Byte>^ img1 = gcnew
    Emgu::CV::Image<Bgr,Byte>(480, 200, Bgr(0, 255, 255));
//Create a font
MCvFont font =
    MCvFont(Emgu::CV::CvEnum::FONT::CV_FONT_HERSHEY_COMPLEX, 1.0,
        1.0);
//Draw "Hello, world" on the yellow image; Start point is (25,
    100) with color blue
String ^ message = "Hello World CPP";
img1->Draw(message, font, Point(25,100), Bgr(255,0,0));
//Show the img1 in the window
CvInvoke::cvShowImage(winname, img1);
```

```
//A key pressing event
CvInvoke::cvWaitKey(0);
//Destroy the window
CvInvoke::cvDestroyWindow(winname);
```

Now we need to set the entry function of this program. Go to **Project | Properties... | Linker | System** and choose **Windows(SUBSYSTEM:WINDOWS)**. Go to **Advanced** and edit main in the **Entry Point**. Open the cpp file and use the project namespace; then insert the following code:

```
int main(array<System::String^>^ args)
{
    MyForm ^ f = gcnew MyForm;
    Application::Run(f);
    return 0;
}
```

On compiling and running, the output will be as shown in the following screenshot:



Now we are okay with our first Emgu CV project in different programming languages. Although Emgu CV supports several languages, we use C# as the first one in the coming chapters. It is strongly recommended to code your Emgu CV project in C#.

## Summary

This chapter can be used as a guide to creating our first Emgu CV project. We also see that Emgu CV can be used with several different languages. But a problem remains; there is no clear explanation for how it works. In the next chapter, we will get to know the fundamental principles of Emgu CV.

# 4

## Wrapping OpenCV

In the last chapter, we succeeded in building our first Emgu CV project. But we do not actually know how it works. As stated in the previous chapters, Emgu CV is special because it is a .NET wrapper to OpenCV. The wrapper is so amazing that it lets OpenCV functions be called from .NET compatible languages such as C#, VB, VC++, and IronPython. This chapter shows the user a better understanding of wrapping OpenCV.

### Architecture overview

In this section we will examine and compare the architectures of OpenCV and Emgu CV.

### OpenCV

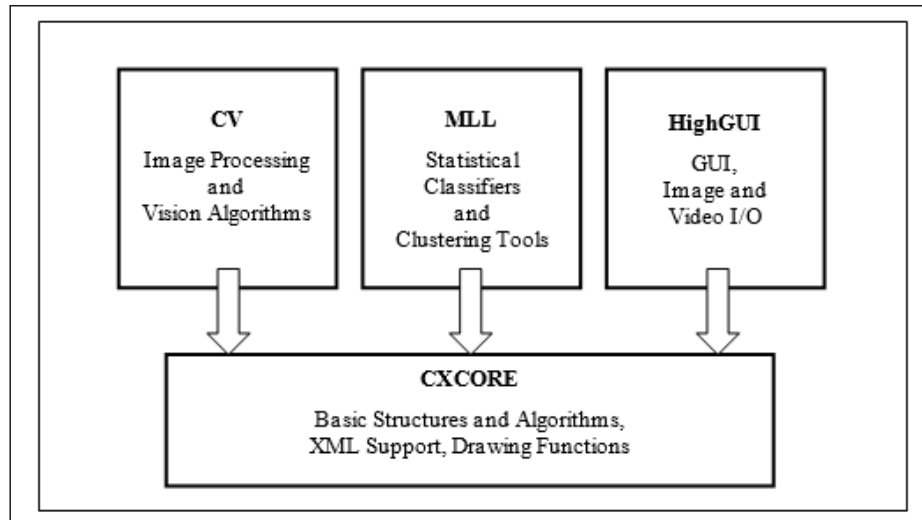
In the `hello-world` project, we already knew our code had something to do with the `bin` folder in the Emgu library that we installed. Those files are OpenCV DLLs, which have the filename starting with `opencv_`. So the Emgu CV users need to have some basic knowledge about OpenCV.

OpenCV is broadly structured into five main components. Four of them are described in the following section:

- The first one is the CV component, which includes the algorithms about computer vision and basic image processing. All the methods for basic processes are found here.
- **ML** is short for **Machine Learning**, which contains popular machine learning algorithms with clustering tools and statistical classifiers.

- HighGUI is designed to construct user-friendly interfaces to load and store media data.
- CXCore is the most important one. This component provides all the basic data structures and contents.

The components can be seen in the following diagram:



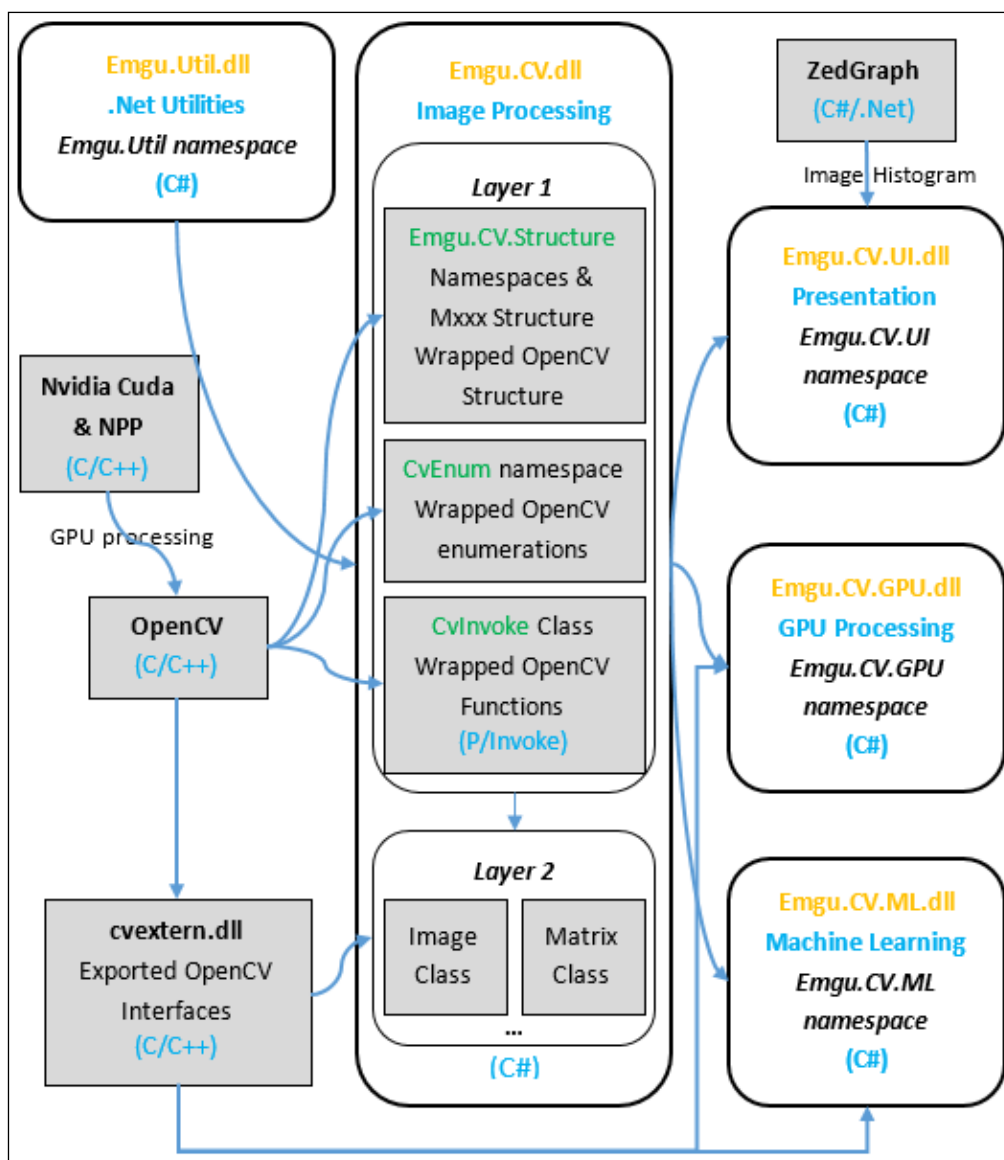
The preceding structure map does not include CvAux, which contains many areas. It can be divided into two parts: defunct areas and experimental algorithms. CvAux is not particularly well documented in the Wiki, but it covers many features. Some of them may migrate to CV in the future, others probably never will.

## Emgu CV

Emgu CV can be seen as two layers on top of OpenCV, which are explained as follows:

- Layer 1 is the basic layer. It includes enumeration, structure, and function mappings. The namespaces are direct wrappers from OpenCV components.
- Layer 2 is an upper layer. It takes good advantage of .NET framework and mixes the classes together. It can be seen in the bridge from OpenCV to .NET.

The architecture of Emgu CV can be seen in the following diagram, which includes more details:



In *Chapter 3, Hello World*, after we create our new Emgu CV project, the first thing we will do is add references. Now we can see what those DLLs are used for:

- `Emgu.Util.dll`: A collection of .NET utilities
- `Emgu.CV.dll`: Basic image-processing algorithms from OpenCV
- `Emgu.CV.UI.dll`: Useful tools for Emgu controls
- `Emgu.CV.GPU.dll`: GPU processing (Nvidia CUDA)
- `Emgu.CV.ML.dll`: Machine learning algorithms

## Function mapping

`Emgu.CV.Invoke` class introduces an easy way to directly call OpenCV functions in .NET programming languages. Every method in the `Emgu.CV.Invoke` class keeps the original name in OpenCV. An example of `cvCreateImage` is given here. In Emgu CV, we write it like this:

```
IntPtr img1 = CvInvoke.cvCreateImage(new Size(256, 256),  
    IPL_DEPTH_16U, 1);
```

It shows the same effects in OpenCV code:

```
IplImage* img1 = cvCreateImage(cvSize(256, 256), IPL_DEPTH_16U, 1);
```

Both of them will create an image with the size of 256 x 256 pixels, the depth of 16 bit unsigned, and the channel 1.

According to Emgu CV library documentation, the `CvInvoke` type exposes the many members and methods. All those hundreds of methods are very useful because they are base functions. We can see how those functions work in the following chapters. Please see the Emgu CV library documentation for more details.

## Structure mapping

`Emgu.CV.Structure` namespace contains all the direct mappings to OpenCV basic structures. Examples are shown in the following table:

Emgu CV structure	OpenCV structure
<code>Emgu.CV.Structure.MIplImage</code>	<code>IplImage</code>
<code>Emgu.CV.Structure.MCvMat</code>	<code>CvMat</code>
...	...
<code>Emgu.CV.Structure.Mxxxx</code>	<code>xxxx</code>

The prefix `M` in the table means managed structure.

The best thing is that Emgu CV mixes .NET graphics and the OpenCV structures together. Parts of them are shown in the following structure:

<b>.NET structure</b>	<b>OpenCV structure</b>
<code>System.Drawing.Point</code>	<code>CvPoint</code>
<code>System.Drawing.PointF</code>	<code>CvPoint2D32f</code>
<code>System.Drawing.Size</code>	<code>CvSize</code>
<code>System.Drawing.Rectangle</code>	<code>CvRect</code>

This is amazing because it is just like a bridge from OpenCV to .NET structure. If we know something about .NET graphics programming, we can code our Emgu project more easily because of the structure mapping.

## Enumeration mapping

`Emgu.CV.CvEnum` namespace contains all the direct mappings to OpenCV enumerations. An example is given here:

```
CvEnum.IPL_DEPTH.IPL_DEPTH_16U
```

This enumeration shares the same value in OpenCV given here:

```
IPL_DEPTH_16U
```

Both of them are 16. We can see that each Emgu CV and OpenCV enum type has the same name. That is what direct mapping means. OpenCV programmers will get quick access to Emgu CV because of it. Please see the Emgu CV library documentation for more details.

## Summary

In this chapter, we examined and compared the architectures of OpenCV and Emgu CV. By the end of the chapter, the user has a preliminary knowledge of Emgu CV. We will start to learn some basic things about images and deal with them in the next chapter, which is more fun.





# 5

## Working with Images

As mentioned in the previous chapter, image-processing libraries make many image-processing operations easy to code because of the extensive set of functions they have. In this chapter, the basic knowledge about images and how to deal with them will be introduced. Fundamental properties and operations will be discussed to enhance the power of Emgu CV. Thus, this chapter can be seen as the basis for most image-processing operations in this book.

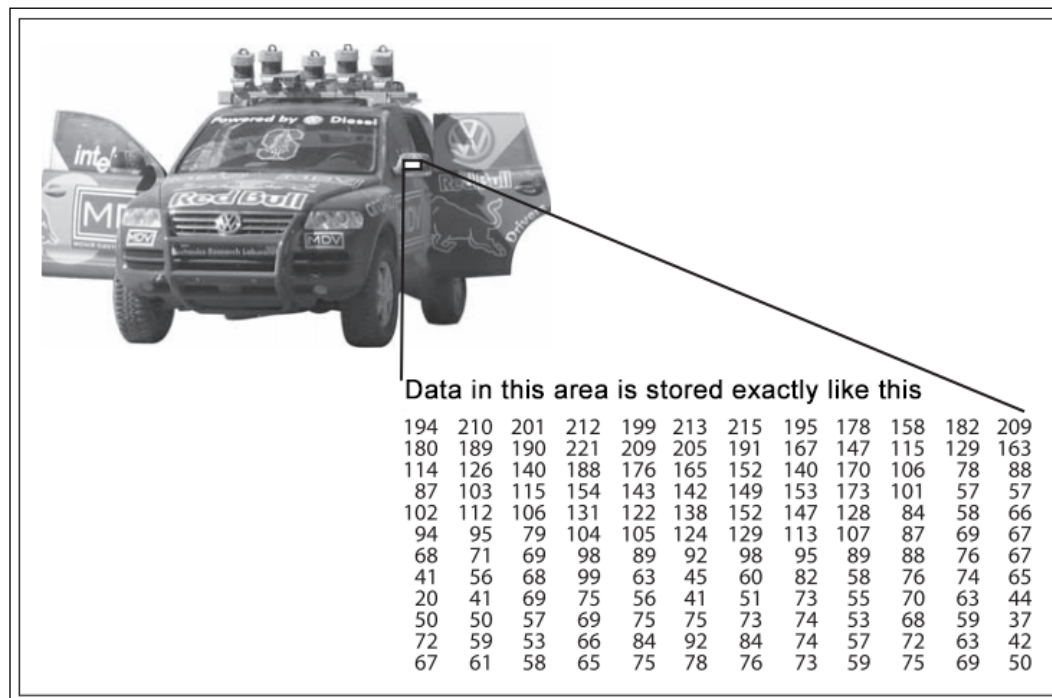
### Digital image representation

Images need to be represented as a series of numbers before it can be stored or used by our computer. In this section, we will discuss the basic concept of digital images.

### Pixels and data

A digital grayscale image can be defined as a simple function:  $f(x, y)$ . It has two dimensions:  $x$  and  $y$ , which mean the coordinates, and  $f$  is the gray level at this point. Color images are composed of different component images; as we know, red, green, and blue can form all the colors we want. Three layers can form a color image. This is called the RGB color system, which we will talk about later. Thus, monochromatic images almost share the same process techniques as the color ones.

However, computers can only load and store a grid of numbers instead of natural pictures. After sampling and quantization, each (x, y) point records an integer, which is commonly known as a pixel. A pixel is composed of both the spatial coordinates and the color level. The following diagram shows how digital data value is stored in each precise location.



Every digital image is made up of pixels, and the greater the number of pixels, the more detail (or resolution) the image will have.

## Pixel resolution

Resolution is known as the pixel count in a digital image. If an image is N pixels high and M pixels wide, we can say this image's pixel resolution is M x N. For example, an image with the dimensions of 5184 x 3456, consists of 5184 pixels in each row and 3456 rows. An image like this has a total of  $5184 \times 3456 = 17,915,904$  pixels. An image size is defined by the width and the height.

The properties of one image on Windows are shown here as an example:

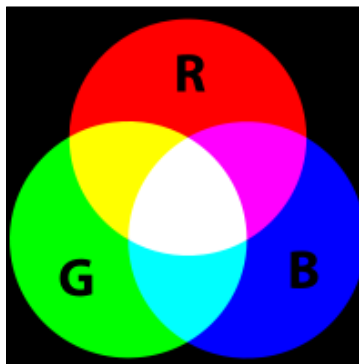
Image	
Image ID	
Dimensions	5184 x 3456
Width	5184 pixels
Height	3456 pixels
Horizontal resolution	72 dpi
Vertical resolution	72 dpi
Bit depth	24
Compression	
Resolution unit	2
Color representation	sRGB
Compressed bits/pixel	

These properties are very important. One of them is color representation, and we are going to talk about it in the next section.

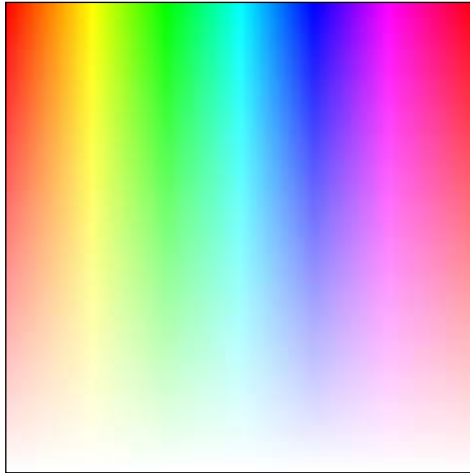
## Color image representation

A color model is a way to represent colors as a set of numbers. It is a mathematical concept and we won't discuss it further here. What concerns us are the color types we will use and the ones that Emgu CV supports.

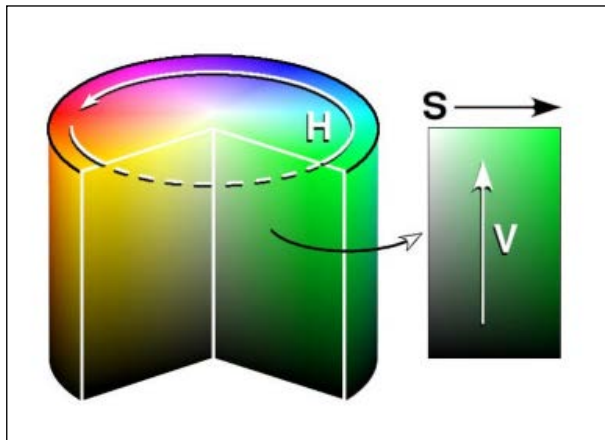
RGB (red, green, and blue) is the most common color model, which uses additive color mixing. An RGB image is an  $M \times N \times 3$  array of color pixels. As we know, mixture of these three primary colors can cover a large part of color space. That's the reason why color screens only need to produce these three kinds of colors and make mixtures. The basic RGB color mixing is given here:



RGBA stands for Red, Green, Blue, and Alpha, which includes extra alpha information on the basis of the RGB color model. The alpha channel here is always used to record the opacity level. For example, if alpha channel equals 0 percent, it is completely transparent, whereas a value of 100 percent shows an opaque image. If the value of alpha channel is between 0 percent and 100 percent, it is a translucent image. PNG is one of the well-known image formats that uses RGBA color space. An example of a translucent RGBA image is given here:



HSV stands for Hue, Saturation, and Value, which is considerably more natural than the RGB way to describe color sensations in a human's experience. This color system is used by artists to get colors from a color wheel, and hue, saturation, and value refer to tint, shade, and tone, respectively. An example of this color model is given here:



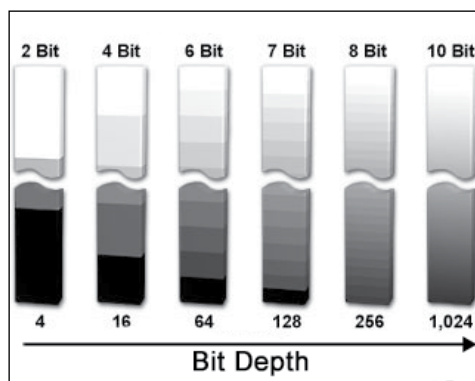
After introducing these three most-frequently used color models, what we're most concerned about is which color types are supported by Emgu CV. A parameter called `TColor` of the image class records the color model of an image. All the values of `TColor` in Emgu CV are given here:

<b>TColor</b>	<b>Color model</b>
Bgr	Defines a Bgr (Blue Green Red) color
Bgr565	Defines a Bgr565 (Blue Green Red) color
Bgra	Defines a Bgra (Blue Green Red Alpha) color
Gray	Defines a Gray color
Hls	Defines a Hls (Hue Lightness Saturation) color
Hsv	Defines a Hsv (Hue Saturation Value) color
Lab	Defines a CIE Lab color
Luv	Defines a CIE Luv color
Rgb	Defines a Rgb (Red Green Blue) color
Rgba	Defines a Rgba (Red Green Blue Alpha) color
Xyz	Defines a Xyz color (CIE XYZ.Rec 709 with D65 white point)
Ycc	Defines a Ycc color (YCrCb JPEG)

Color models define how we choose a color.

## Color depth

Color depth, or bit depth is the number of bits used to indicate the color of a single pixel in a bitmapped image. In a color image, it records the number of bits for each color component of a single pixel. For example, in a monochrome image, if four bits are utilized, this image can have sixteen brightness levels and each level can be seen as a different gray shade between pure white and black, as shown in the following figure:



In a color image, we use  $n$  bit color to describe the depth of color, which means there are  $2^n$  colors that can be chosen for each pixel. For example, a 16 bit color image supports 16 bits for three RGB colors. There can be 5 bits for R, 6 bits for green, and 5 bits for blue so that it has 65,536 colors. True color supports 24 bit for RGB colors. Usually, there can be 8 bits for each of the R, G, and B components. 8 bit equals 1 byte. So, true color has 16,777,216 ( $2^{24}$ ) color variations.

Image color depth is specified using a `TDepth` parameter in Emgu CV. Here we list all the values of the depth that Emgu CV supported:

TDepth	Value
Byte	8 bits for each color component (0-255)
SByte	8 bits for each color component (-128-127)
Single	32 bits for each color component (C# 32 bit float)
Double	64 bits for each color component (C# 64 bit double)
UInt16	16 bits for each color component (unsigned int16)
Int16	16 bits for each color component (signed int16)
Int32	32 bits for each color component (signed int32)

Now, we are done with all the basic concepts of an image, so we are going to work with images in the next section.

## Working with images

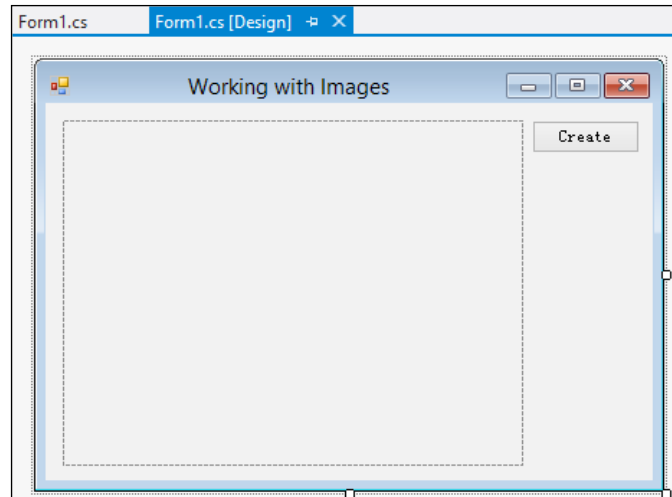
After understanding the basics of an image, the first thing is to create a new image with specific parameters.

### Creating an image

We can create an image by two different callings. Although it is possible to do it by calling `CvInvoke.CreateImage`, like we did in *Chapter 3, Hello World*, it is suggested to construct an `Image<TColor, TDepth>` object instead from now on. The advantages of using this object are as follows:

- Automatic garbage collection
- Can be debugged more easily with debugger visualizer
- More advanced methods

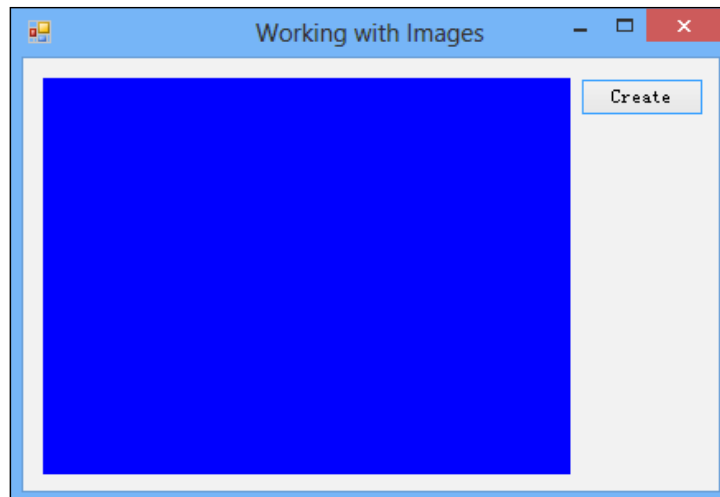
Now we are going to create a 320 x 240 image of Bgr (blue, green, and red) color with 8 bits for each color component. We want the background color to be blue. Start a new project and add the references. In the design view, let's add a picturebox control and a button like this:



Double-click on the create button and insert the following C# code:

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(320, 240, new  
    Bgr(255, 0, 0));  
pictureBox1.Image = img1.ToBitmap();
```

Build the project and click on the button, the output should be like this:





The `Image(TColor, TDepth)` type exposes the following constructors:

Name	Description
<code>Image(TColor, TDepth)()</code>	Creates an empty image
<code>Image(TColor, TDepth)(Bitmap)</code>	Obtains the image from the specific bitmap
<code>Image(TColor, TDepth)(Size)</code>	Creates a blank image of the specific size
<code>Image(TColor, TDepth)(String)</code>	Reads image from a file
<code>Image(TColor, TDepth)(TDepth[, ,])</code>	Creates image from the specific multidimensional data, where the first dimension is the number of rows (height), the second dimension is the number of columns (width), and the third dimension is the channel
<code>Image(TColor, TDepth)(Image(Gray, TDepth)[])</code>	Creates a multichannel image from multiple grayscale images
<code>Image(TColor, TDepth)(Int32, Int32)</code>	Creates a blank image of the specified width and height
<code>Image(TColor, TDepth)(SerializationInfo, StreamingContext)</code>	Constructor used to deserialize runtime serialized object
<code>Image(TColor, TDepth)(Int32, Int32, TColor)</code>	Creates a blank image of the specified width, height, and color
<code>Image(TColor, TDepth)(Int32, Int32, Int32, IntPtr)</code>	Creates an image from unmanaged data

More often, we open an existing image to do further processing. The preceding table shows a way to deal with it.

## Loading an image from a file

From the preceding table, we can load an image in a simple way. Add an **Open...** button and insert the following C# code:

```
//Loading image from file
private void button2_Click(object sender, EventArgs e)
{
    string strFileName = string.Empty;
    //prompts the user to open a file
    OpenFileDialog ofd = new OpenFileDialog();
    if (ofd.ShowDialog() == DialogResult.OK)
    {

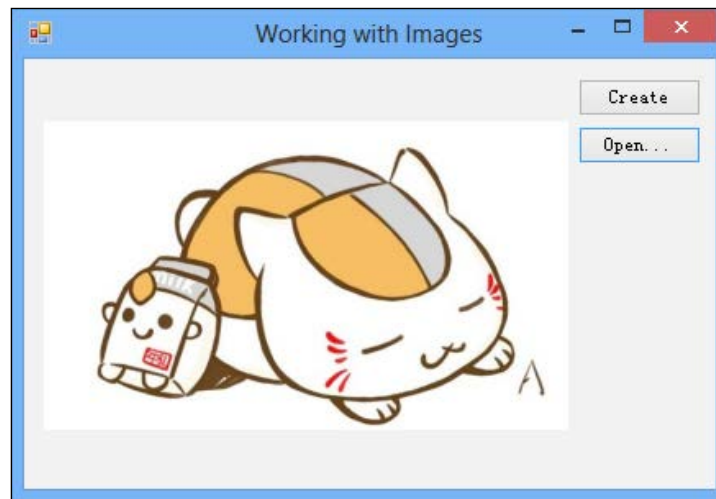
```

```

        Image<Bgr, Byte> img1 = new Image<Bgr,
        Byte>(ofd.FileName);
        pictureBox1.Image = img1.ToBitmap();
    }
}

```

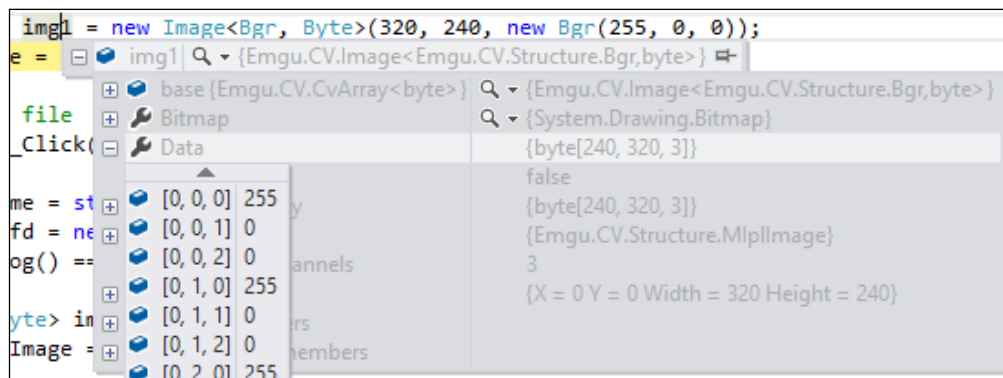
Build the project and click on the newly created button, an open window will jump out, after choosing the image path, the output should be like this:



Now we will try to edit the image.

## Operations with pixels

The values of pixels are stored in the data properly, which is a three-dimensional array as shown in the following screenshot:

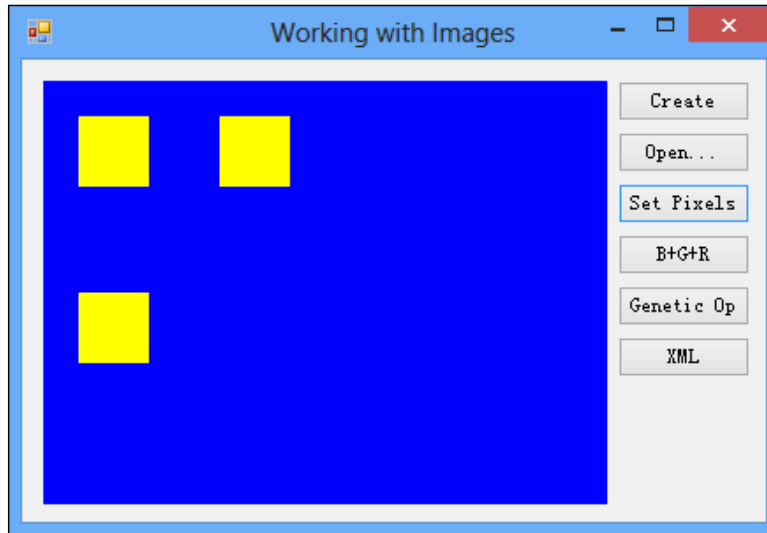


Now, we will try to change the pixel value via the data property. We are going to create a blue image and try to iterate the color of some pixels to yellow.

Add a **Set Pixels** button and insert the following C# code:

```
private void button3_Click(object sender, EventArgs e)
{
    //Creating an image
    Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(320, 240, new
    Bgr(255, 0, 0));
    //yellow(0,255,255)
    Byte b1 = 255;
    Bgr yellow = new Bgr(0, 255, 255);
    //Change the color by iterating Data
    for (int i = 20; i < 60; i++)
    {
        for (int j = 20; j < 60; j++)
        {
            img1.Data[i, j, 0] = 0;
            img1.Data[i, j, 1] = b1;
            img1.Data[i, j, 2] = b1;
        }
    }
    //Change the color by setting an Bgr color
    for (int i = 120; i < 160; i++)
    {
        for (int j = 20; j < 60; j++)
        {
            img1[i, j] = yellow;
        }
    }
    //The best practice to reduce performance penalties
    byte[, ,] data = img1.Data;
    for (int i = 20; i < 60; i++)
    {
        for (int j = 100; j < 140; j++)
        {
            //Avoid using c# property inside a loop can have a huge
            performance boost
            data[i, j, 0] = 0;
            data[i, j, 1] = b1;
            data[i, j, 2] = b1;
        }
    }
    //Show the result
    pictureBox1.Image = img1.ToBitmap();
}
```

Build and compile again, then click on the newly created button, the result is given in the following screenshot:



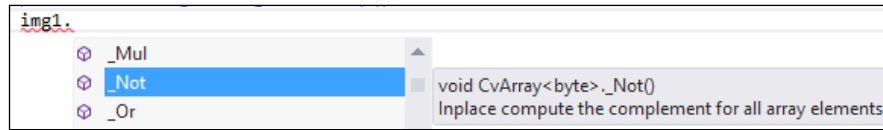
It is very easy to get or set pixels because Emgu CV is a nice OpenCV wrapper so that operations and methods are easy to be called. We will talk about the method naming rules in the next section.

## Method naming rules

All the methods come from OpenCV functions, but it's not a big problem if you know little about the functions in OpenCV. In the `Image<TColor, TDepth>` class, there is a one-to-one match between the names of functions of OpenCV and Emgu CV:

- Method `ABC` corresponds to the OpenCV function `cvABC`. For example, the `Image<TColor, TDepth>.Not()` method corresponds to the `cvNot` function, and both of them compute the complement image.
- Method `_ABC` acts almost the same as method `ABC` except that the operation is performed in place rather than returning a value.

With the help of an IDE, you can find the methods you want very quickly if you know little about the original OpenCV functions:



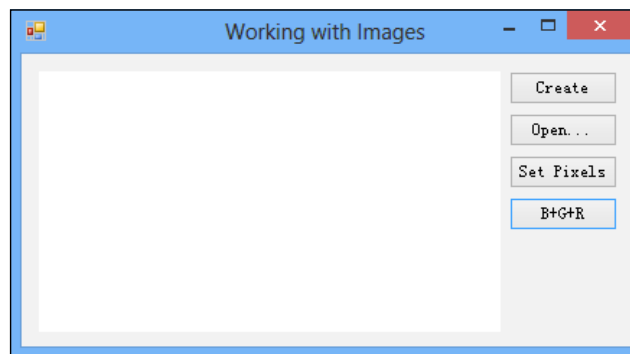
It improves the efficiency and accuracy when you are coding. Operators overload can help you more in your functions.

## Using operator overload

Using overload operators makes coding more easy. The operators +, -, \*, and / have been overloaded in Emgu CV. Here comes an example that will help us understand it:

```
private void button4_Click(object sender, EventArgs e)
{
    Image<Bgr, Byte> imgBlue = new Image<Bgr, Byte>(320, 240, new
    Bgr(255, 0, 0));
    Image<Bgr, Byte> imgGreen = new Image<Bgr, Byte>(320, 240, new
    Bgr(0, 255, 0));
    Image<Bgr, Byte> imgRed = new Image<Bgr, Byte>(320, 240, new
    Bgr(0, 0, 255));
    //Blue + Green + Red = White
    //Operators Overload
    Image<Bgr, Byte> img1 = imgBlue + imgGreen + imgRed;
    pictureBox1.Image = img1.ToBitmap();
}
```

We can get a pure white image like this:



It makes the code more understandable and natural.

## Generic operations support

The advantage of generic operations is making our code more flexible and user friendly with only minor performance penalty. Fortunately, Emgu CV has the ability to perform generic operations.

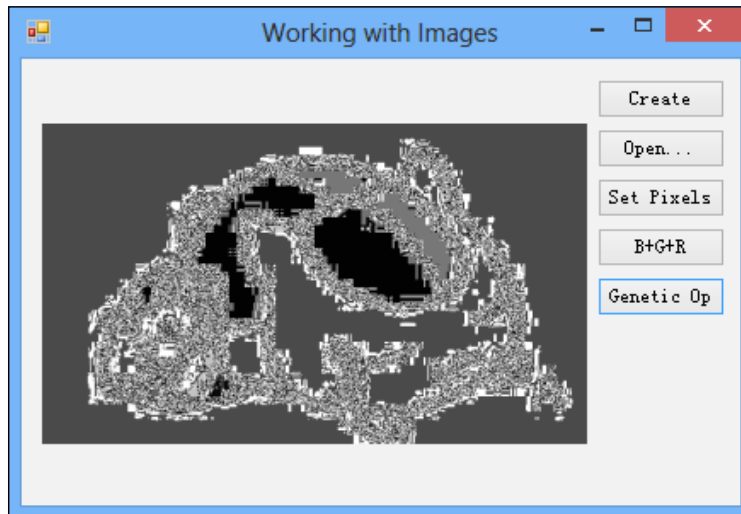
Assume that we have `Image<Gray, Byte> imgGray` with pixels already set:

```
Image<Gray, Byte> imgGray = new Image<Gray, Byte>("Gray.jpg");
```

What we want is to do a converting like this:

```
Image<Gray, Single> img1 = imgGray.Convert<Single>(delegate(Byte b) { return (Single)Math.Sin(b * b / 255.0); });
```

We convert it to a new single grayscale image. Each pixel of `img1` corresponds to `imgGray` with a sine transformation. The output is as follows:



This is amazing because it is very hard to perform this kind of operation in OpenCV. Immense flexibility is obvious here.

## Garbage collection

Garbage collection is a form of automatic memory management. The `Image<TColor, TDepth>` class will be disposed when the garbage collector considers there is no more reference to this object. Then the unmanaged structure will be released from the memory, conserving the limited memory on streaming applications.

We propose to call the `Dispose()` method by ourselves if we are dealing with a huge image. Don't forget that we have the `using` keyword to limit the scope of the object like this:

```
using (Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(320, 240))
{
    ... //something happens here
} //img1 will be automatically disposed without calling
    img1.Dispose()
```

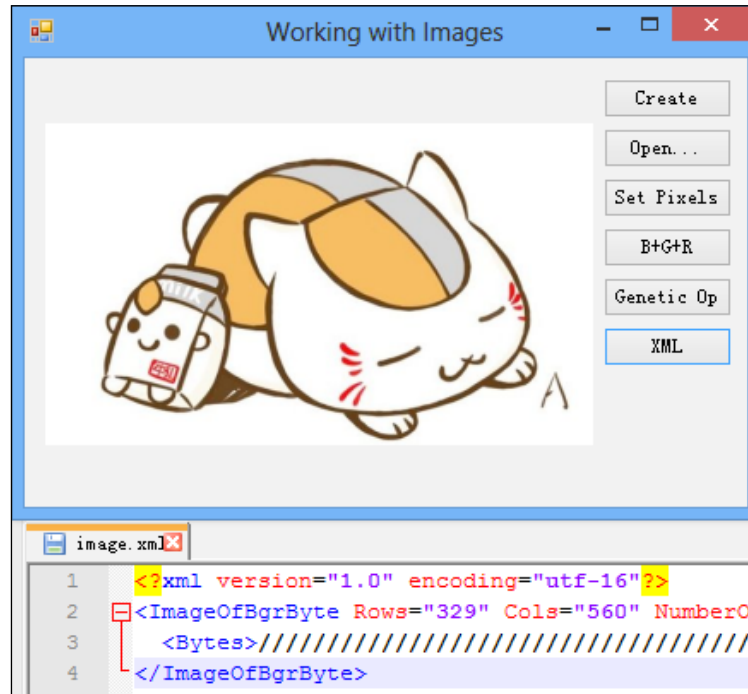
## XML serialization

XML serialization can be seen as an advantage of Emgu CV. Standardized XML format can be used for many web services. Emgu CV provides a simple method to convert image to XML or XML to image.

The code of a new button will convert any image we want to an XML file:

```
private void button6_Click(object sender, EventArgs e)
{
    string strFileName = string.Empty;
    OpenFileDialog ofd = new OpenFileDialog();
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        Image<Bgr, Byte> img1 = new Image<Bgr,
        Byte>(ofd.FileName);
        pictureBox1.Image = img1.ToBitmap();
        //Convert an Image to XmlDocument
        StringBuilder sb1 = new StringBuilder();
        (new XmlSerializer(typeof(Image<Bgr,
        Byte>))).Serialize(new StringWriter(sb1),img1);
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(sb1.ToString());
        //Save the XML file
        xmlDoc.Save("image.xml");
    }
}
```

The output is given in the following screenshot:



Also, we can convert an XML image to a normal image. We can use the following code to deal with the conversion:

```
Image<Bgr, Byte> img1 = (Image<Bgr, Byte>)(
    new XmlSerializer(typeof(Image<Bgr, Byte>))).Deserialize(new
        XmlNodeReader(xDoc));
```

It can convert an XML document `xDoc` to `Image<Bgr, Byte>`.

## Summary

In this chapter, we talked about the basic concepts of images. At this point, we have almost all of the basics at our disposal. We understand the structure of Emgu CV library as well as image representation. We can code and actually handle the pixels. We will move on to the matrix in the next chapter.





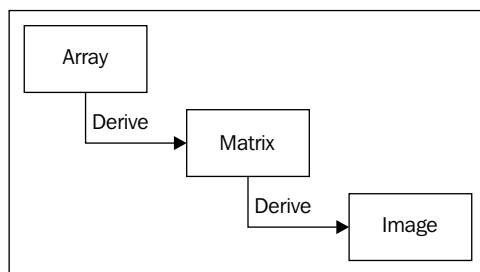
# 6

## Working with Matrices

In the previous chapter, we introduced how to work with images. In the field of computer vision, matrices, which are used to record data and do mathematical calculations, are as important as images. The latest OpenCV release implements the image class as a two-dimensional matrix, which really illustrates the importance of matrix operations in image processing. Thus, this chapter serves as the introduction to the operations on matrices with Emgu CV.

### Matrix and the Image class

Now if we are going to deal with an image, we may repeatedly encounter the `Image` (`TColor`, `TDepth`) class, which has already been used often in the previous chapter. The truth is that an `Image` class can be seen as being derived from the class `Matrix`. The latest OpenCV release implements the image class as a matrix (`mat` class). Therefore, `Matrix` and `Image` share the same members and methods. A matrix can be seen as a rectangular array of data, but the matrix implements more operations for developers. The following figure shows the relationship between these three structures or classes:



Let's get to know more about the `Matrix` class in Emgu CV.

## Definition and parameters

As a rectangular array, a matrix is defined by its size and depth. For instance, there is a real matrix A with 3 rows and 2 columns as in the following figure:

$$A = \begin{bmatrix} -1.3 & 0.6 \\ 20.4 & 5.5 \\ 9.7 & -6.2 \end{bmatrix}$$

We can call the numbers in the matrix elements. The elements of a matrix can be integers, floating point literals, and so on. There is a generic parameter called `TDDepth`, which defines the element type. We list all the values of the depth that Emgu CV supports as follows, and they are almost the same as the `Image` class we introduced in the previous chapter:

TDDepth	Value
Byte	8 bits for each element (0 - 255)
SByte	8 bits for each element (-128 - 127)
Single	32 bits for each element (C# 32-bit float)
Double	64 bits for each element (C# 64-bit double)
UInt16	16 bits for each element (unsigned int16)
Int16	16 bits for each element (signed int16)
Int32	32 bits for each element (signed int32)

The size of a matrix is defined by its width and height, or we can call them columns and rows, respectively. Sometimes we can use matrices to construct a vector, which is why matrices are widely used. Matrices that have only one single row are called row vectors, and those that have only one single column are called column vectors. Whenever we create a vector, we usually use a matrix with only one column, which will be described in detail in the subsequent chapters.

## Working with matrices

After introducing the definition of matrices, we'd like to create the first matrix in our project.

## Creating a matrix

Now, if we want to create a Double matrix of a specific size in Emgu CV, it can be done by using the following line of code:

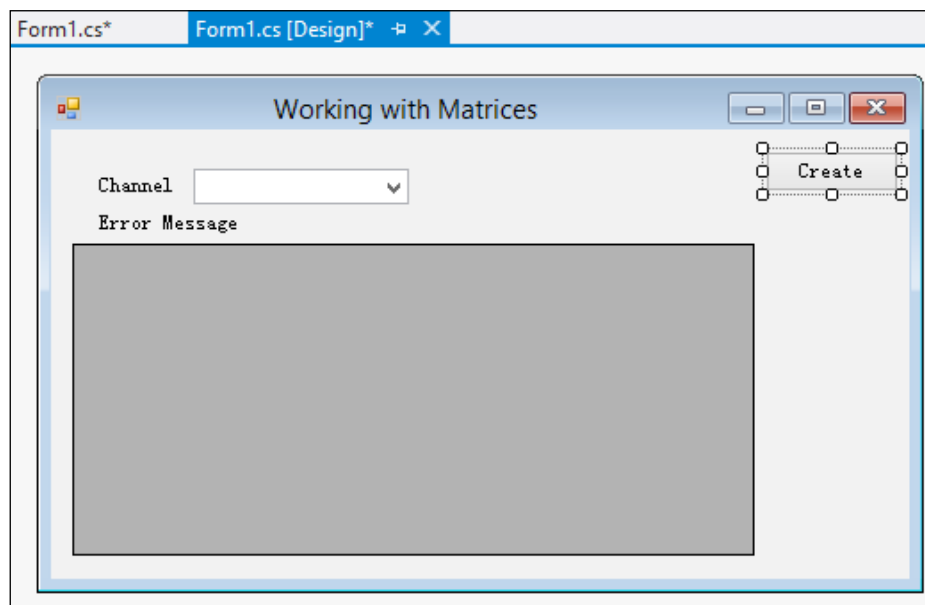
```
Matrix<Double> matrix = new Matrix<Double>(height, width);
```

The `Double` here is the type of the elements. `width` and `height` decide the size of this matrix.



We can also create this matrix by calling `CvInvoke.cvCreateMat`. This process is similar to that of creating an image, which was covered in the previous chapter. But it is also used to construct a `Matrix<TDepth>` object instead. The reason behind this can be found in *Chapter 5, Working with Images*.

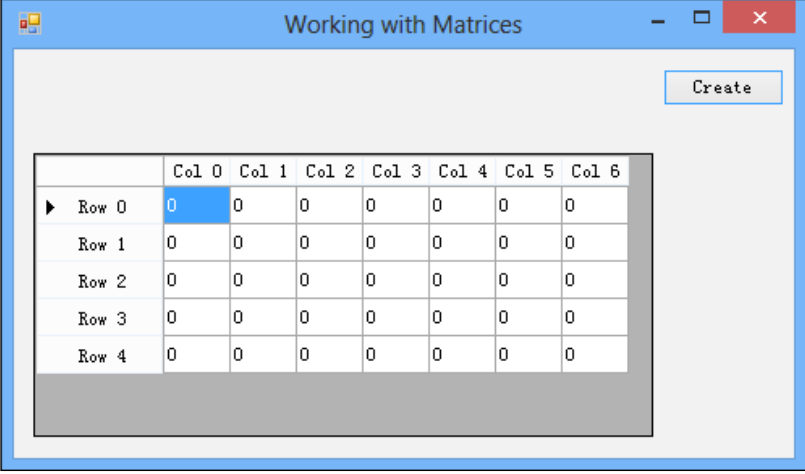
Now we are going to create a 5x7 matrix, which means it has 5 rows and 7 columns. Start a new project and add the references. In the design view, we need to add a `MatrixBox` control (if you don't see it in the toolbox, please refer to *Chapter 2, Installing Emgu CV*) and a **Create** button, as shown in the following figure:



Double-click on the **Create** button and insert the following C# code in the click event:

```
Matrix<Double> matrix1 = new Matrix<Double>(5, 7);
matrixBox1.Matrix = matrix1;
```

Build the project and click on the **Create** button; the output should be as shown in the following figure:



	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 0	0	0	0	0	0	0	0
Row 1	0	0	0	0	0	0	0
Row 2	0	0	0	0	0	0	0
Row 3	0	0	0	0	0	0	0
Row 4	0	0	0	0	0	0	0

We can see all the elements here are zero. In the next step, we are going to fill the matrix with some data.

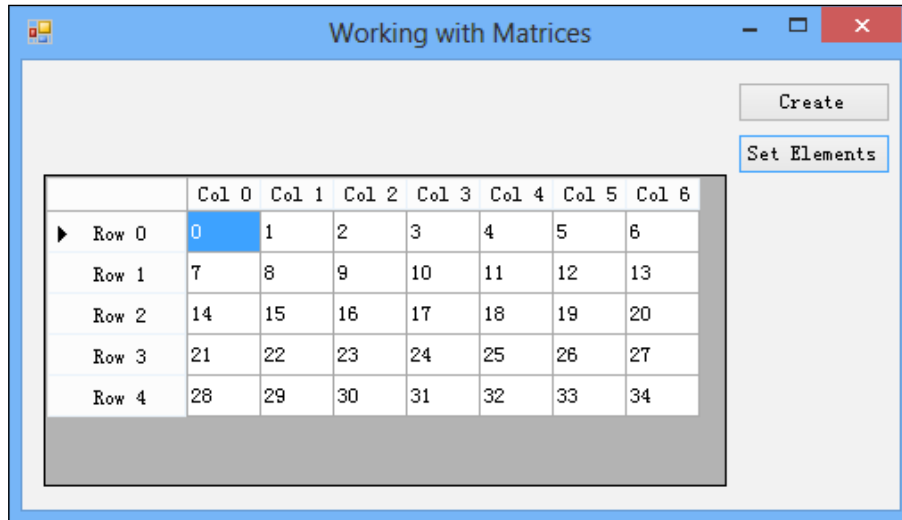
## Operations with elements

The values of elements are stored in the Data property. One way to access the Data property of matrix is to set the values of each element.

Add a **Set Elements** button and insert the following C# code:

```
private void button2_Click(object sender, EventArgs e)
{
    //Create a matrix
    Matrix<Double> matrix1 = new Matrix<Double>(5, 7);
    double element = 0;
    //Set the elements
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 7; j++)
        {
            matrix1.Data[i, j] = element;
            element++;
        }
    }
    //Show the result
    matrixBox1.Matrix = matrix1;
}
```

Build and compile the matrix again, and then click on the new **Set Elements** button; the result is shown in the following figure:



	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 0	0	1	2	3	4	5	6
Row 1	7	8	9	10	11	12	13
Row 2	14	15	16	17	18	19	20
Row 3	21	22	23	24	25	26	27
Row 4	28	29	30	31	32	33	34

You can try matrix operations such as matrix multiplication, as you have learned how to set the elements. Also, since matrices can be seen as abstract images, the operations and methods are almost the same; you can try them all by yourself now.

## Summary

In this chapter, we have learned what matrices are and how to deal with them. What we have not yet seen is how to use matrices in a real application. In the next few chapters, we are going to create some projects that are more complex, so readers need to be fluent with the matrix operations.



# 7

## Shape Detection

In the previous chapters, we have learned the basis of Emgu CV and explored a variety of functions related to images. The majority of the methods presented so far are used to convert one image to a new but similar one. From this chapter onwards, our main idea is to introduce some real and interesting projects to see the power of Emgu CV. What we are going to do is transform an image into a representation of the data. One of these applications is Shape Detection. Emgu CV provides many methods to deal with this problem, and we'd like to demonstrate Canny and Hough Transform to implement the detections.

### Canny Edge Detector

In 1986, John F. Canny developed an edge detection operator. It is a multi-stage image algorithm that is applied to a great extent to digital images. We shall not talk much about the mathematical theory of it. However, the Canny algorithm provides a way to assemble the candidate pixels into edges, which we can call contours. A hysteresis threshold is used to form contours from pixels, which means that the Canny algorithm has two thresholds. In Emgu CV, the syntax of the Canny method is as follows:

```
public Image<Gray, byte> Canny(  
    double thresh,  
    double threshLinking  
)
```

To put it simply, a pixel will be accepted as a candidate edge pixel, only if it has a gradient that is larger than the upper threshold. And the pixel will be rejected if its gradient is below the lower one. If it is between the two thresholds, the pixel will be accepted when it's next to a pixel that has already been accepted. The input image of the Canny method should be grayscale, and the output is also a gray scale (Boolean actually) image.

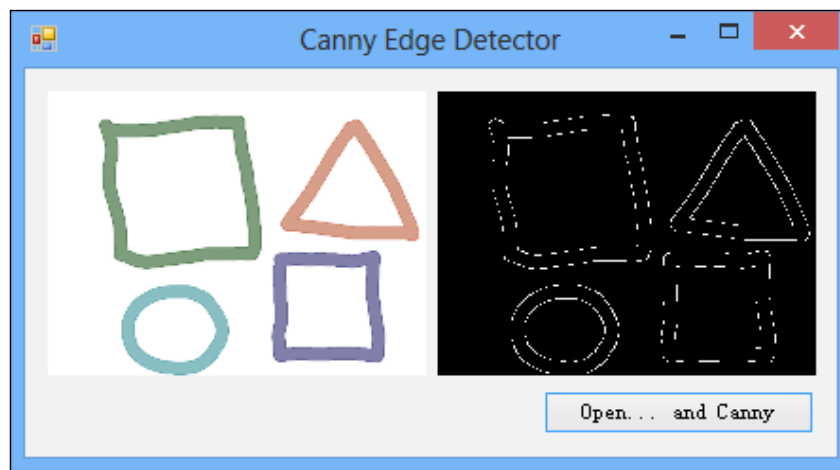


A threshold that is too high will cause loss of important information. On the other hand, a threshold that is too low will regard irrelevant pixels as important ones. It is hard to give a universal threshold to adapt to all the images. All in all, we should try to get a better result.

The following is an example code of Canny Edge Detector:

```
private void button1_Click(object sender, EventArgs e)
{
    string strFileName = string.Empty;
    OpenFileDialog ofd = new OpenFileDialog();
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(ofd.FileName);
        pictureBox1.Image = img1.ToBitmap();
        //Convert the img1 to grayscale and then filter out the noise
        Image<Gray, Byte> gray1 = img1.Convert<Gray,
            Byte>().PyrDown().PyrUp();
        //Canny Edge Detector
        Image<Gray, Byte> cannyGray = gray1.Canny(120, 180);
        pictureBox2.Image = cannyGray.ToBitmap();
    }
}
```

The output of this program is shown in the following figure:



It is easy, but our work is not completed yet. We can see the edges clearly but the computer did not catch them at all. What we are going to do next is learn something about Hough transforms.

## Hough transforms

The Hough transform is a feature-extraction technique that is used to detect lines and circles in an image. In many cases, an edge detector is used as a preprocessing step to get contours. And then lines will be searched by Hough transforms.

### Hough Line transform

In Emgu CV, we can call the `HoughLines` or the `HoughLinesBinary` method to deal with the lines in images. They do not show the steps of complex computation to the users. Instead, its return value and the parameters are our first concern. The difference in these two methods is explained as follows:

- `Image (TColor, TDepth).HoughLines`: Applies Canny Edge Detector followed by the Probabilistic Hough transform to find line segments in the image
- `Image (TColor, TDepth).HoughLinesBinary`: Applies the Probabilistic Hough transform to find line segments. The current image must be a binary image (for example, the edges as a result of Canny Edge Detector)

Since we have introduced Canny Edge Detector, we want to focus more on the Hough Line Transform. So we will take the `HoughLinesBinary` method as an example. It is defined as follows:

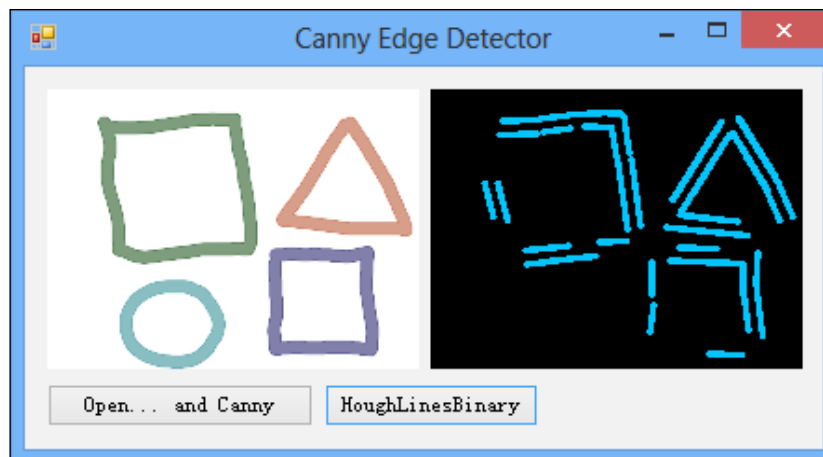
```
public LineSegment2D[][] HoughLinesBinary(
    double rhoResolution,
    double thetaResolution,
    int threshold,
    double minLineWidth,
    double gapBetweenLines
)
```

The first two arguments, `rhoResolution` and `thetaResolution`, set the resolution desired for the lines in the binary image. A line can be regarded as a 2-D histogram with a slope and an intercept, so the unit of `rhoResolution` should be pixels and the unit of `thetaResolution`, radians. The value of the threshold is the minimum pixel number of the lines. When the pixel count is larger than the threshold, this line will be recorded during detection. The last two parameters are the minimum line width and the gap between the lines. Their units are pixels. The return value, `LineSegment2D`, is a place where the results are stored.

Now let's try the Hough line transform using the following code:

```
private void button2_Click(object sender, EventArgs e)
{
    //Load the image that has been pre-processed
    Image<Gray, Byte> cannyGray = new Image<Gray,
        byte>(new Bitmap(pictureBox2.Image));
    //Call HoughLinesBinary method
    LineSegment2D[] lines = cannyGray.HoughLinesBinary(1,
        Math.PI / 45.0, 20, 30, 10)[0];
    //Draw lines
    Image<Bgr, Byte> imageLines = new Image<Bgr,
        byte>(cannyGray.Width, cannyGray.Height);
    foreach (LineSegment2D line in lines)
    {
        imageLines.Draw(line, new Bgr(Color.DeepSkyBlue), 5);
    }
    //Show result
    pictureBox2.Image = imageLines.ToBitmap();
}
```

The result should be like the following figure:



We got all the straight lines in the new image, but it had nothing to do with the circle in our image. Now we come to another Hough Transform. Hough Circle Transform will help us to solve the problem of detecting the edges of the circle in our image.

## Hough Circle transform

The process of identifying circular objects works almost the same as line transforms. In its mathematical theory, some of the parameters and steps are different to get circles instead of lines. We do not talk much about the complicated theory. What we focus on is implementation. If you want to know more about the basis of these functions, searching Wikipedia would be a good way to understand the detailed steps during calculation.

- `Image (TColor, TDepth).HoughCircles`: First, apply Canny Edge Detector on the current image, then apply Hough transform to find circles.

In the `HoughCircles` method, Canny Edge Detector is included. The input image should be a gray scale one. With these points in mind, let's see the arguments of this method shown as follows:

```
public CircleF[] [] HoughCircles(
    TColor cannyThreshold,
    TColor accumulatorThreshold,
    double dp,
    double minDist,
    int minRadius,
    int maxRadius
)
```

This function has similar parameters to the `HoughLinesBinary` method. The `CircleF` is an array to store the result, which encodes the radius and the location of the circle. The parameters, `cannyThreshold` and `accumulatorThreshold`, are the Canny edge threshold and the accumulator threshold, respectively. Usually `cannyThreshold` is larger than `accumulatorThreshold`. The parameter `accumulatorThreshold` works exactly the same as the threshold argument of the `HoughLinesBinary` method.

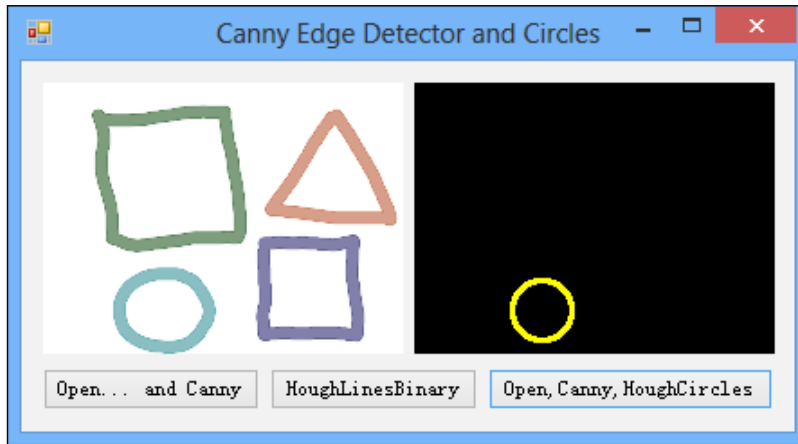
The next parameter, `dp`, decides the resolution of the accumulator image during calculation. We can change the value of `dp` to lower the resolution of the input image. This is necessary because there is no reason to expect the circles that exist in the image to fall naturally into the same number of categories as the width or the height of the image itself. When `dp` equals to 1, the resolution will not be changed. If the value of `dp` is greater than 1, the resolution will be smaller. And it cannot be less than 1.

The fourth parameter, `minDist`, is used to avoid duplicate recording. It is the minimum distance between circles.

The last two parameters, `minRadius` and `maxRadius`, are easy to understand. These two values set the biggest and smallest circles that will be detected. Circles with a radius between the maximum and minimum radius can be recorded into the result. An example code of how to use the `HoughCircles` method is as follows:

```
private void button3_Click(object sender, EventArgs e)
{
    string strFileName = string.Empty;
    OpenFileDialog ofd = new OpenFileDialog();
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(ofd.FileName);
        pictureBox1.Image = img1.ToBitmap();
        //Convert the img1 to grayscale and then filter out the noise
        Image<Gray, Byte> gray1 = img1.Convert<Gray,
            Byte>().PyrDown().PyrUp();
        //Call HoughCircles (Canny included)
        CircleF[] circles = gray1.HoughCircles(
            new Gray(180), //cannyThreshold
            new Gray(120), //accumulatorThreshold
            2.0,           //dp
            15.0,          //minDist
            5,             //minRadius
            0              //maxRadius
        )[0];
        //Draw circles
        Image<Bgr, Byte> imageCircles = img1.CopyBlank();
        foreach (CircleF circle in circles)
        {
            imageCircles.Draw(circle, new Bgr(Color.Yellow), 5);
        }
        //Show result
        pictureBox2.Image = imageCircles.ToBitmap();
    }
}
```

The expected result is shown in the following figure:



It seems that we get all the information from this image, but our program cannot tell us anything about the shapes, such as squares, rectangles, or triangles. Now we are ready to learn about contours to deal with this problem.

## Contour

Canny Edge Detector can help us get the edge pixels, but it cannot combine these separate segments into a cohesive whole. Emgu CV provides a convenient method that will do exactly what we want. Before we start using it, we need to know what a contour means. The understanding may be different in different circumstances, but here we'd like to define a contour as a list of pixels or points that can represent a curve in a digital image, in a structured way. In Emgu CV, a contour is stored as a sequence of pixels or points.

## Contour finding

The `FindContours` method in the `Image<TColor, TDepth>` class, is the key to access all the contours of a binary image. So before we call this function, we need to use Canny Edge Detector to do the transformation. An example code is as follows:

```
Image<Bgr, Byte> img1 = new Image<Bgr, Byte>(sourceFileName);
//Convert the img1 to grayscale and then filter out the noise
Image<Gray, Byte> gray1 = img1.Convert<Gray, Byte>().PyrDown().
PyrUp();
//Canny Edge Detector
Image<Gray, Byte> cannyGray = gray1.Canny(120, 180);
//Call FindContours method
Contour<Point> contour1 = cannyGray.FindContours();
```

The preceding code can get the default contour of our source image, and the data is stored in the `Contour<Point>` structure. We need to handle its operations.

## Representation of contours

In Emgu CV, there are two ways to represent the contour.

### Sequences of vertexes

We can use the vertexes to determine a shape. For example, if we have a rectangle from point (0, 0) to point (2, 2), there are the following two ways to store it on our computer:

- **Use points:** P1 (0, 0), P2 (1, 0), P3 (2, 0), P4 (2, 1), P5 (2, 2), P6 (1, 2), P7 (0, 2), P8 (0, 1)
- **Use vertexes:** V1 (0, 0), V2 (2, 0), V3 (2, 2), V4 (0, 2)

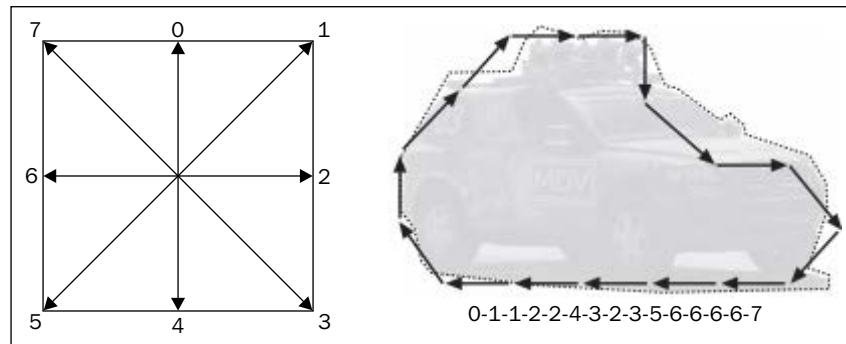
The following code can access the points on the contour:

```
for (int i = 0; i < contour1.Total; i++)
{
    sbContour.AppendFormat("{0},", contour1[i]);
}
```

This is very simple, let's get to the other way to represent a contour.

## Free chain codes

The contours created by the `FindContours` method are sequences of points. If we add some parameters, such as `CHAIN_APPROX_METHOD`, into the method, we can change the way the result is stored. A `CV_CHAIN_CODE` value can output contours in the Freeman chain code. When using a Freeman chain, a polygon is represented as a sequence of steps in directions, which has eight values, from zero to seven. The following figure can help you understand this better:



But the support for the Freeman chain code is not very good in Emgu CV. We do not recommend using this method in your project. Freeman chain code has its advantages, and if you want to handle it anyway, the Open CV documentation will help you.

## Drawing contours

Drawing contours is a basic task for us. After we get all the points in the contour, we can simply write the following line of code to complete it:

```
imageResult.Draw(contours1, externalColor, holeColor, maxLevel,
thickness)
```

The preceding line of code draws contour outlines in the image if the thickness equals or is more than 0, or fills the area bounded by the contours if the thickness is less than 0.



## Polygon approximations

In this chapter, we want to do shape analysis. Emgu CV provides an easy implementation of representing a contour to a polygon. We can reduce the points of a contour after polygon approximations, and the new sequence of points will have fewer vertices. `Contour<Point>.ApproxPoly` can do the job well:

```
contour2 = contour1.ApproxPoly(accuracy, new MemStorage());
```

Here, we need to use a memory storage to handle memory allocation for dynamic objects. It is important and we will see it in many complicated projects.

## A contours example

Now we are going to combine all the things about contours that we have talked about. Our goal is to detect the shape in the image. The following code is based on the idea that we apply Canny Edge Detector first and then find the contours. After that, we do polygon approximations and get the shapes by counting the vertices.

```
private void button2_Click(object sender, EventArgs e)
{
    Image<Bgr, Byte> img1 = new Image<Bgr, byte>(new
        Bitmap(pictureBox1.Image));
    //Convert the img1 to grayscale and then filter out the noise
    Image<Gray, Byte> gray1 = img1.Convert<Gray,
        Byte>().PyrDown().PyrUp();
    //Canny Edge Detector
    Image<Gray, Byte> cannyGray = gray1.Canny(120, 180);
    //Lists to store triangles and rectangles
    List<Triangle2DF> triangleList = new List<Triangle2DF>();
    List<MCvBox2D> boxList = new List<MCvBox2D>();
    //New Memstorage
    using (MemStorage storagel = new MemStorage())
    for (Contour<Point> contours1 = cannyGray.FindContours();
        contours1 != null; contours1 = contours1.HNext)
    {
        //Polygon Approximations
        Contour<Point> contoursAP = contours1.ApproxPoly(
            contours1.Perimeter * 0.05, storagel);
        //Use area to wipe out the unnecessary result
        if (contours1.Area >= 200)
        {
            //Use vertices to determine the shape
            if (contoursAP.Total == 3)
            {

```

---

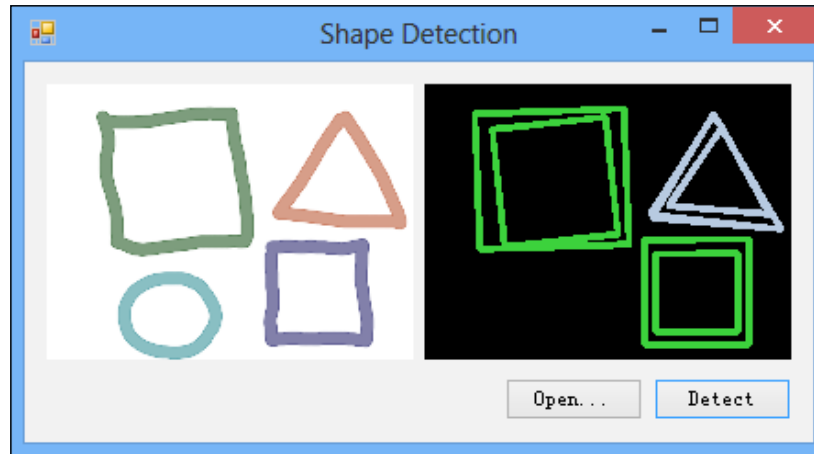
```

        //Triangle
        Point[] points = contoursAP.ToArray();
        triangleList.Add(new Triangle2DF(
            points[0],
            points[1],
            points[2]
        ));
    }
    else if (contoursAP.Total == 4)
    {
        //Rectangle
        bool isRectangle = true;
        Point[] points = contoursAP.ToArray();
        LineSegment2D[] edges = PointCollection.
            PolyLine(points, true);
        //degree within the range of [75, 105] will be detected
        for (int i = 0; i < edges.Length; i++)
        {
            double angle = Math.Abs(edges[(i + 1) %
                edges.Length].GetExteriorAngleDegree(edges[i]));
            if (angle < 75 || angle > 105)
            {
                isRectangle = false;
                break;
            }
        }
        if (isRectangle)
        {
            boxList.Add(contoursAP.GetMinAreaRect());
        }
    }
}

//Draw result
Image<Bgr, Byte> imageResult = img1.CopyBlank();
foreach (Triangle2DF triangle in triangleList)
    imageResult.Draw(triangle, new Bgr(Color.LightSteelBlue), 5);
foreach (MCvBox2D box in boxList)
    imageResult.Draw(box, new Bgr(Color.LimeGreen), 5);
//Show result
pictureBox2.Image = imageResult.ToBitmap();
}

```

The result should be like the following figure:



## Summary

In this chapter, we introduced how to transform an image into a representation of the data. Canny Edge Detector, Hough transform, and Contours algorithm were implemented in detail, respectively. At last, we can combine all we have learned to detect the edges, lines, circles, triangles, and rectangles in an image. More interesting things are waiting for you to explore. In the following chapter, we will deal with a more difficult problem.

# 8

## Face Detection

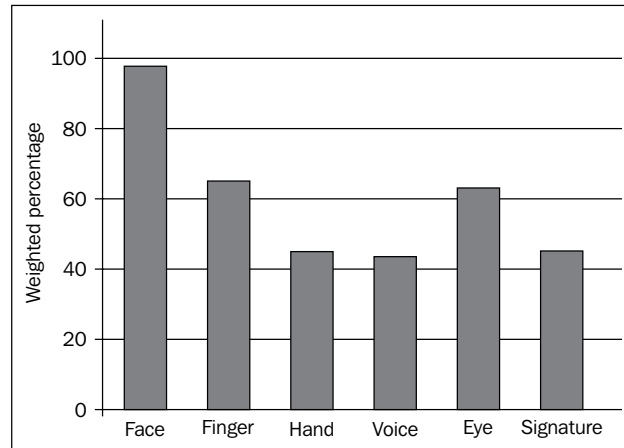
In the previous chapter, we learned how to detect shapes in Emgu CV. Now we turn to a more complicated technique, Face Detection. It is generally considered as a machine-learning problem, but in Emgu CV and OpenCV, Face Detection has a different format from all the other machine-learning libraries. Face detection or face recognition can be regarded as a self-contained application, and it was developed earlier than other ML functions. In this chapter, we will cover face detection and other related techniques in detail, and show you how to use Emgu CV to detect faces.

### **Biometric systems**

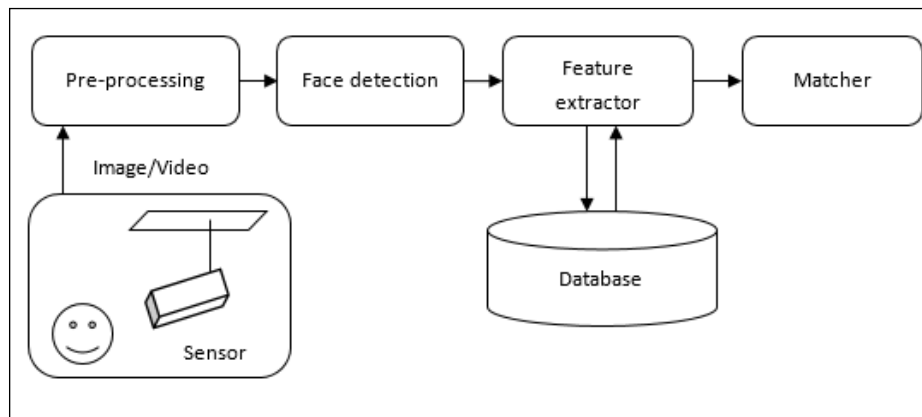
Face detection or face recognition is one of the problems in biometrics. Biometrics refers to identifying human traits or characteristics. In computer science and the computer vision area, biometric systems are developed to automatically verify a person from input data, which can be digital images, voice waves, infrared data, and so on.

Face detection and recognition has several advantages over other biometric systems. It is more natural and nonintrusive, because data collection is contactless. Face images can be captured in a covert manner.

There is a study among different biometric systems and face-based features scored the highest, as shown in the following figure:



A typical biometric system can be divided into several parts, for example, a face detection system generally consists of the following six modules:



In order to complete face recognition, firstly, we have to resolve issues, such as image/video capture, preprocessing, and face detection. Now we are going to deal with them.

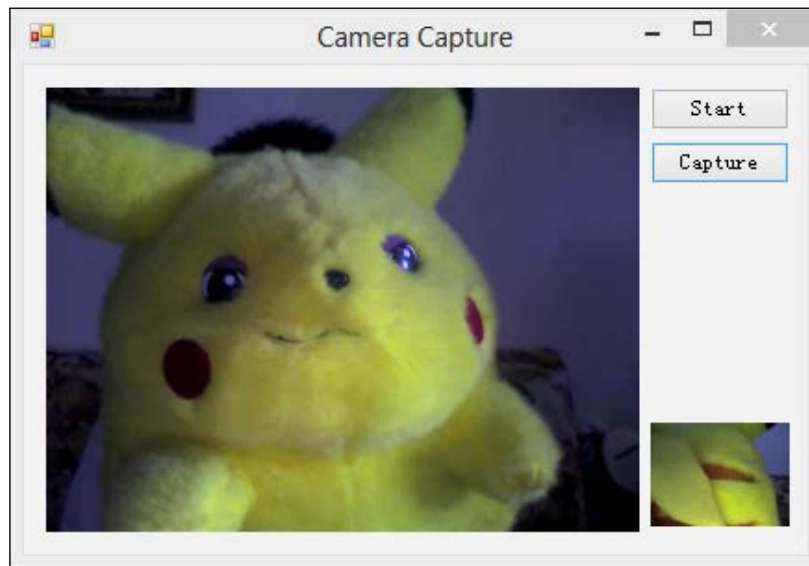
## Camera captures

In the previous chapters, we have learned how to load, process, and save images with Emgu CV and .NET controls in our project. In this section, we are going to use the video camera of our computer to capture an image.

Create a new project, add the using statement in the beginning, and insert the following lines of code in the Csharp file:

```
public partial class Form1 : Form
{
    //Frame
    Image<Bgr, byte> current;
    //Webcam
    Capture webcam = new Capture();
    public Form1()
    {
        InitializeComponent();
    }
    //Start button
    private void button1_Click(object sender, EventArgs e)
    {
        //run until close
        Application.Idle += new EventHandler(processCamera);
    }
    private void processCamera(object sender, EventArgs e)
    {
        current = webcam.QueryFrame();
        // Flip because forecam is MIRRORED
        current = current.Flip(Emgu.CV.CvEnum.FLIP.HORIZONTAL);
        pictureBox1.Image = current.Bitmap;
    }
    //Capture button
    private void button2_Click(object sender, EventArgs e)
    {
        //Get the current frame
        pictureBox2.Image = pictureBox1.Image;
    }
}
```

Make sure that your computer has at least one camera to run this example. The `Capture` class is new to us, which contains the capture device. The `current` variable stores the current frame of the capture device. It is very easy to understand and use it because Emgu CV does a lot of work behind the scenes. The output after running this project should be like the following figure:



Now we can use a camera to capture images or simply load an image from our disk, which means we have no problem with the image acquisition module. If you want to know more about camera capture, such as how to set the properties of the camera, please take a look at [http://www.emgu.com/wiki/index.php?title=Camera\\_Capture](http://www.emgu.com/wiki/index.php?title=Camera_Capture). As for preprocessing, please refer to *Chapter 5, Working with Images*. In the next section, we are going to talk about the main part that is machine learning and face detection.

## Machine learning

Machine learning is a branch of artificial intelligence. Its goal is that we want our machine to learn something from the data set and then answer the question about the data we give. Machine learning turns data into useful information and can always be used for prediction. For example, a machine-learning system can be trained on digital photos to be able to distinguish between face and non-face images. After that, this system can be used to predict the possible face area of a new image.

Emgu CV provides many famous machine-learning algorithms. All the algorithms are included in `Emgu.CV.ML Namespace`. The following table shows the supported machine-learning algorithms in Emgu CV:

Class	Algorithm
ANN_MLP	Neural network
Boost	Boost tree
DTree	Decision tree
EM	Expectation maximization model
EMLegacy	
EMParams	
ERTreess	Extreme random tree
GBTrees	Gradient boosting tree
KNearest	The KNearest classifier
NormalBayesClassifier	A normal Bayes classifier
RTrees	Random tree
StatModel	A statistic model
SVM	Support vector machine
SVMPParams	
HaarCascade( <b>Obsolete</b> )	HaarCascade for object detection
CascadeClassifier	The Cascade classifier
Facerecognizer	Face recognizer

The last three algorithms are in `Emgu.CV Namespace` but they are object-detection applications based on a clever use of the boosting algorithm. Here we do not want to introduce every algorithm in detail, because it may occupy a tremendous number of pages. What we are most concerned about in this chapter are Haar Cascades, which are used for object detection.

## Face detection or the Haar classifier

The Haar classifier builds a boosted rejection cascade, and it can be used to detect faces of other rigid objects. As seen in the preceding table, the Haar classifier is not included in `Emgu.CV.ML` but it is a machine-learning algorithm. The computer vision field changes very fast and some parts of the library are implemented individually, to reduce the risk of running out of data. Face detection techniques advance with time and as we see, the `HaarCascade` class is obsolete so we are going to study the `CascadeClassifier` class.



In OpenCV and Emgu CV, the first implementation of the face detection technique was developed by Paul Viola and Michael Jones, and later extended by Jochen Maydt and Rainer Lienhart. The name of the `HaarCascade` class comes from the usage of Haar features, which calculate rectangular image regions and threshold the result. Haar features or Haar-like wavelets more precisely, can be trained into cascade files by us but Emgu CV also ships a series of pretrained ones that can be used directly.

The pretrained cascade files are saved under `D:\Emgu\emgucv-windows-universal-gpu 2.4.9.1847\opencv\data\haarcascades`. There are two default XML files in the folder. One is for eye detection and the other is for front-face detection.

## **Boosting theory and supervised learning**

Data can be divided into two kinds; one is labeled data, such as sex, age, weight, and so on, and the other one has no labels. For example, the faces of a family where the sons and daughters look like their parents. Supervised learning is the machine-learning task of inferring a function from labeled training data. Every example in supervised learning must be a pair that has two parts. One is an input data vector, and the other is our desired result, or more precisely, a supervisory signal.

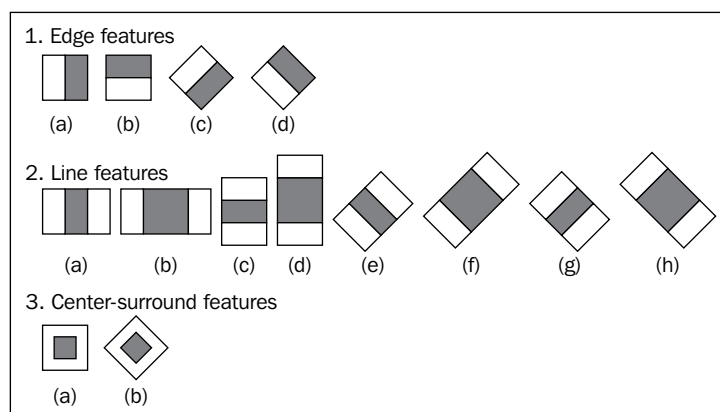
The Haar classifier is a supervised classifier. So we have to prepare images that have faces (or no faces, called negative samples) that have the same size to train our cascade.

Boosting theory is a little complicated. Here it may be simply interpreted as building a decision tree. Each node in the Viola-Jones detector is a rejection cascade, which has a high pass rate and a low rejection rate. After building a big and appropriate decision tree with a large number of nodes, an input image can be judged during computation.

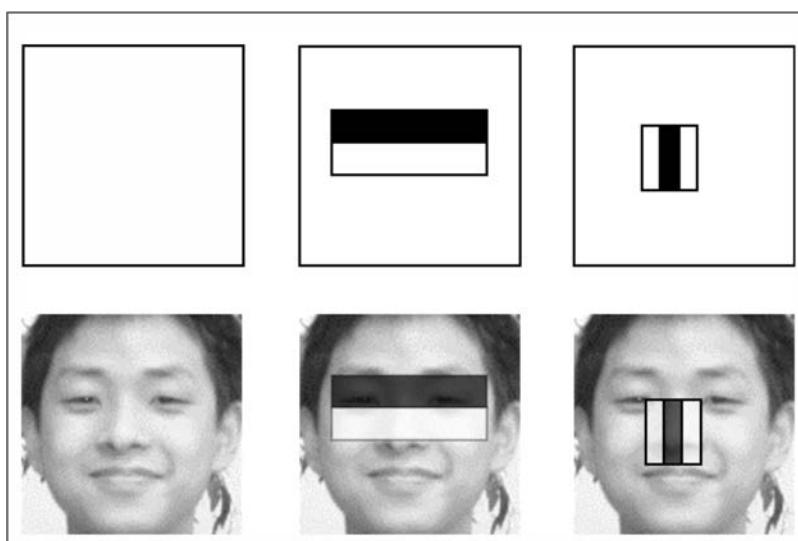
## **Haar-like features**

The Viola-Jones classifier theory cannot be explained very clearly in a short amount of space. Here we'd like to talk about the basics, which are easy to understand. If you want to know the theory in detail, please refer to research papers.

In the algorithm, the Viola-Jones classifier uses Haar-like input features, a threshold applied to sums and differences of rectangular image regions. Haar-like features are shown in the following figure:

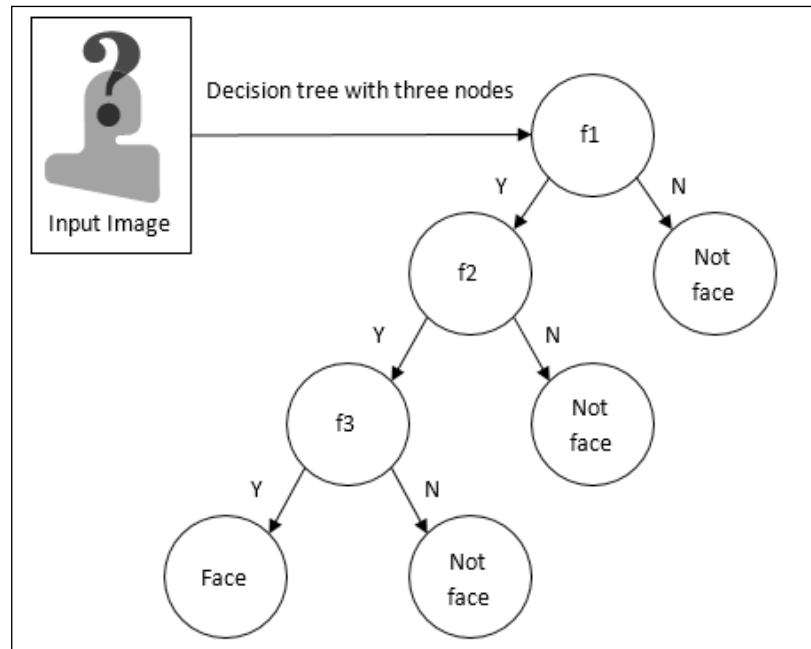


The light region, in the preceding features, is interpreted as "add this area," and the dark region as "subtract this area." The preceding figure looks confusing, but if we put the features into a face image, it may look meaningful, as in the following figure:



Through the observation of average people, we can find a rule such as, the color of the human eye area looks deeper than the cheek area. So, if we use a Haar feature that can be placed in the adjacent rectangle of the eyes and the cheek area, the rectangle can be used for face detection, as in the preceding figure.

Here, we omit the mathematical computation part, which uses integral image technique to accelerate computation. The next step is to create binary classification nodes of a decision tree, like the following figure:



Each of the non-leaf nodes represents a judgment, each path indicates the result of the last judgment, and each leaf represents a kind of output, face or not face. After all the Haar features calculation, the image that passes all the nodes is regarded as a face image.

Now, what we are most interested in is how to use it to detect faces of an input image.

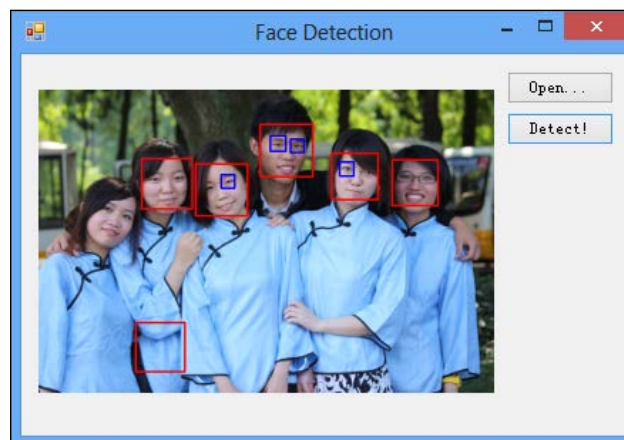
## Code for face detection

If we want to build a face detection application, the core part is to use the `CascadeClassifier` class in our project. The following code presumes that we have a trained classifier cascade (we can use pretrained cascade in the Emgu folder) and then open an image to start the detection:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            pictureBox1.Image = Image.
                FromFile(openFileDialog1.FileName);
        }
    }
    private void button2_Click(object sender, EventArgs e)
    {
        //Load the image
        Image<Bgr, Byte> image = new Image<Bgr,
            Byte>(openFileDialog1.FileName);
        //Use List to store faces and eyes
        List<Rectangle> faces = new List<Rectangle>();
        List<Rectangle> eyes = new List<Rectangle>();
        //Pre-trained cascade
        CascadeClassifier face = new CascadeClassifier(
            "haarcascade_frontalface_default.xml");
        CascadeClassifier eye = new CascadeClassifier("
            haarcascade_eye.xml");
        //The input image of CascadeClassifier must be grayscale
        Image<Gray, Byte> gray = image.Convert<Gray, Byte>();
        //Face detection
        Rectangle[] facesDetected = face.DetectMultiScale(
            gray,                //image
            1.1,                 //scaleFactor
            10,                  //minNeighbors
            new Size(20, 20),    //minSize
            Size.Empty);         //maxSize
        faces.AddRange(facesDetected);
        //Eyes detection
```

```
foreach (Rectangle f in facesDetected)
{
    gray.ROI = f;
    Rectangle[] eyesDetected = eye.DetectMultiScale(
        gray,
        1.1,
        10,
        new Size(20, 20),
        Size.Empty);
    gray.ROI = Rectangle.Empty;
    foreach (Rectangle ey in eyesDetected)
    {
        Rectangle eyeRect = ey;
        eyeRect.Offset(f.X, f.Y);
        eyes.Add(eyeRect);
    }
}
//Draw detected area
foreach (Rectangle face1 in faces)
    image.Draw(face1, new Bgr(Color.Red), 2);
foreach (Rectangle eye1 in eyes)
    image.Draw(eye1, new Bgr(Color.Blue), 2);
//Show image
pictureBox1.Image = image.Bitmap;
}
}
```

After compiling and running the project, we can get what we want, as shown in the following figure:



As we see, the first girl was not detected and some of the eyes in the image were rejected by our program. Also, there is a mistaken-detected area. A more appropriate cascade will do a better job.

## Summary

In this chapter, we have learned the basis of machine learning and face detection. Emgu CV did a lot of work in the background, so we do not introduce the theory in detail, but we did learn how to use the methods and data structures to finish the task of face detection. In the following chapter, we are going to deal with the recognition algorithm, which means more work will be done after detection.

If you want to go further with face detection and recognition, please refer to the `FaceRecognizer` class.



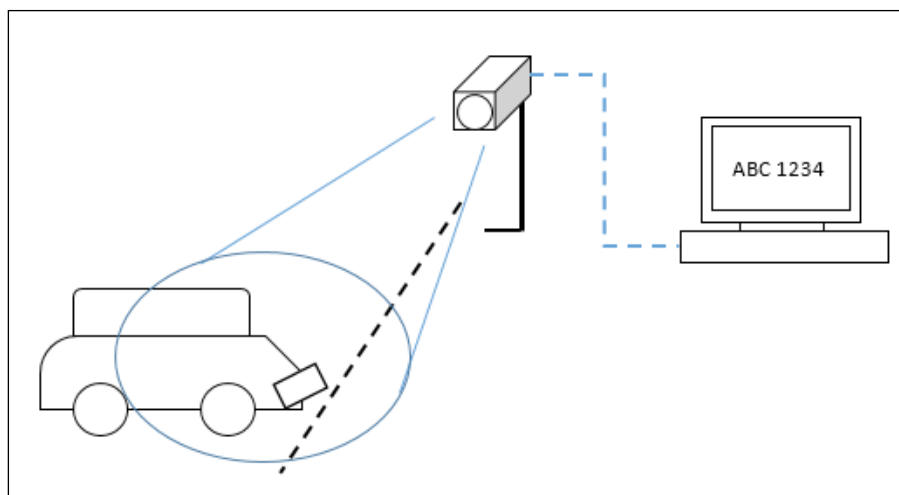
# 9

## License Plate Recognition

In the previous chapters, we learned how to detect shapes or faces in the given image, but we did not go further than detection. In this chapter, we will do license plate recognition. It is more complicated work because we will translate the license plate image into text after locating its position in the image. In this chapter, we will cover Optical Character Recognition and other related techniques in detail. In the end, we will build our project to do automatic recognition with Emgu CV.

### License Plate Recognition

**License Plate Recognition (LPR)** is a kind of mass surveillance technique that involves **OCR (Optical Character Recognition)** in the given videos or images. It is widely used by security departments or systems. For the police, it can be used to collect information and catalog the traffic on roads in real time as shown in the following figure.





An LPR system can use an existing traffic enforcement camera to load and store the captured images. License plates are reflective and cameras use infrared lighting or a flashlight to make sure these systems can capture and process the image at any time of the day. One of the uncertainties is that the plate varies from country to country or even from one state to another, as seen in the following figure, so we must design a specific algorithm during processing and recognition:



For a general case, the common algorithm is presented as follows.

## Algorithms for LPR

To identify a license plate, the key idea can be divided into the following two stages:

- Firstly, license plate region detection should be performed
- Secondly, use optical character recognition on the region to get the license character and number

To be more specific, in a complete LPR system, there should be the following five primary procedures to identify a license plate:

- **License plate localization:** This method detects and isolates the plate region on the given image
- **Normalization:** This method adjusts the size, dimensions, contrast, and brightness of the plate region
- **Character segmentation:** This method gets each individual character of the region
- **Optical character recognition:** This method recognizes the characters
- **Syntactical analysis:** This method makes a comparison of country-specific rules

The following figure shows what happens during normalization, character segmentation, and OCR:



The implementation of the algorithms will be introduced later.

## OCR

OCR is the conversion of images of text into characters. Generally, the scanned image of the text can include printed text, handwritten text, and so on. It is the process of scanning the text, analyzing the image, and getting the text. It is widely used to digitize printed or handwritten text so that they can be stored, searched, and displayed on the computer. OCR is also a part of computer vision and pattern recognition research.

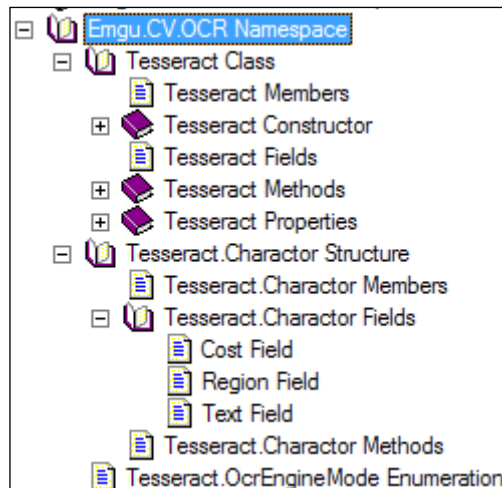
In the earlier optical character recognition system, words can only be recognized character by character and are only compatible with one kind of font during one process. Now, most OCR systems have a very high degree of accuracy for almost all kinds of fonts. With the development of pattern recognition technology, some leading OCR systems can deal with non textual components in the text in a very nice format. The non-textual components can be paragraphs, columns, images, and so on.

## Tesseract-OCR

Tesseract is a famous OCR engine developed by Ray Smith from 1985 to 1995 at HP Labs. It is available at <http://code.google.com/p/tesseract-ocr/>. The Tesseract OCR engine supports over 60 languages and different image formats. It was one of the top three OCR engines in the 1995 UNLV accuracy test. It improved a little after that, but since 2006, it has been improved extensively by Google. It is the most accurate open-source optical character recognition engine now.

Tesseract can work on Windows, Mac OS X, Linux, Android, and iOS. However, as an engine, it only provides a command-line tool. Emgu CV provides the .NET wrapper, `tessnet2` (<http://www.pixel-technology.com/freeware/tessnet2/>), to the OCR engine Tesseract so that we can use Tesseract through the Emgu CV library in our project. However, this OCR wrapper can only be run on Windows and is not compatible with Linux and other operating systems.

The following figure shows the structure of `Emgu.CV.OCR Namespace`. It is all about Tesseract, and we will learn how to use it in our project in the next section.



## Code for License Plate Recognition

As we introduced previously, the main idea to do license plate recognition has two stages. First, we detect and locate the position of the plate in the given image. After this, the program normalizes the image and uses an OCR engine to translate the text.

There are two ways to do the detection related to the previous chapters. One is to train a Haar cascade (see *Chapter 8, Face Detection*) to distinguish plate and non plate image areas. Another way of thinking is that characters can be seen as regular geometric figures, and we can find the contours (see *Chapter 7, Shape Detection*) to get the license plate.

Here, we recommend the second way to be the first choice because contours can also help with image normalization and character segmentation.

## Assumption

In our project, we assume that the given image includes a European license plate. In this case, we only consider the rectangles with the width-height ratio in the range of (3, 10) and the character must be from A to Z and 0 to 9. If the width-height ratio is not in the specified range, it is not a license plate.

The definition of ratio in our project is given as follows:

```
double ratio = (double)width / height;
if (!(3<ratio && ratio<10))
{...}
```

Initialize the OCR engine using the following code:

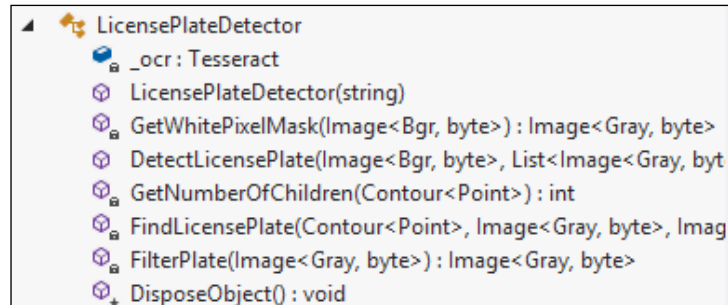
```
private Tesseract _ocr;
public LicensePlateDetector(String dataPath)
{
    _ocr = new Tesseract(dataPath, "eng",
        Tesseract.OcrEngineMode.OEM_TESSERACT_CUBE_COMBINED);
    //specify the inclusive range of characters
    _ocr.SetVariable("tessedit_char_whitelist",
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ-1234567890");
}
```

However, if you want to use this system in another region whose plate is different from the assumption, you have to change the preceding code to satisfy the conditions.

## Source code

Create a new project and add the references to it. The original files as well as the `Emgu.CV.OCR.dll` file must be included. We want our code to be suitable for different environments, and we also want it to be easy to edit. To do this, first create a new class and name it `LicensePlateDetector`.

The methods in the class are as shown in the following screenshot. Now, we are going to talk about all these methods in detail.



## GetWhitePixelMask

This method computes the white pixel mask for the given image. The goal of this method is to convert the color image to a gray scale one to better accommodate the changes. This example works only on a white background license plate because of the threshold we set in the following code. If you want to process a different kind of number plate, you can use other color ranges in that scenario. Also, this method is not necessary because we have the Convert method to do the same work but with less flexibility.

```
private static Image<Gray, Byte> GetWhitePixelMask(
    Image<Bgr, byte> image)
{
    using (Image<Hsv, Byte> hsvimage = image.Convert<Hsv, Byte>())
    {
        //split current Image (hsvimage) into an array of
        gray scale images where each element in the array
        represent a single color channel of the original image
        Image<Gray, Byte>[] channels = hsvimage.Split();
        try
        {
            //channels[1] is the mask for saturation less than 50,
            this is the mask for either white or black pixels
            channels[1]._ThresholdBinaryInv(new Gray(50), new
            Gray(255));
            //channels[2] is the mask for bright pixels
            channels[2]._ThresholdBinary(new Gray(190), new Gray(255));
            //combine the channels to a grayscale image, see cvAnd
            CvInvoke.cvAnd(channels[1], channels[2],
            channels[0], IntPtr.Zero);
        }
    }
}
```

```

        finally
        {
            channels[1].Dispose();
            channels[2].Dispose();
        }
        return channels[0];
    }
}

```

The return value of the preceding method is the white pixel mask, which is a gray scale image. After doing this, we can use Canny Edge Detector to get the contours so that we can find the license plate.

## DetectLicensePlate

As we introduced earlier, there are two ways to do detection. Here, we use the Canny detector. You should have intimate knowledge of it if you have read the previous chapters. The code is given as follows:

```

public List<String> DetectLicensePlate(
    Image<Bgr, byte> img,
    List<Image<Gray, Byte>> licensePlateImagesList,
    List<Image<Gray, Byte>> filteredLicensePlateImagesList,
    List<MCvBox2D> detectedLicensePlateRegionList)
{
    List<String> licenses = new List<String>();
    //Both Convert method or GetWhitePixelMask will be OK
    //using (Image<Gray, byte> gray = img.Convert<Gray, Byte>())
    using (Image<Gray, byte> gray = GetWhitePixelMask(img))
    using (Image<Gray, Byte> canny = new Image<Gray,
        byte>(gray.Size))
    using (MemStorage stor = new MemStorage())
    {
        CvInvoke.cvCanny(gray, canny, 100, 50, 3);
        Contour<Point> contours = canny.FindContours(
            Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_SIMPLE,
            Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_TREE,
            stor);
        FindLicensePlate(contours, gray, canny,
            licensePlateImagesList, filteredLicensePlateImagesList,
            detectedLicensePlateRegionList, licenses);
    }
    return licenses;
}

```

The parameters of this method are explained as follows:

- **img**: It is the source image and contains the license plate
- **licensePlateImagesList**: It is the image list that stores the detected plate regions
- **detectedLicensePlateRegionList**: It is an MCvBox2D list that stores the regions.

The return value is a string list that stores the words in each license plate.

## FindLicensePlate

This method is used to locate the license plate region in the given image. It is almost like the code for rectangle detection in *Chapter 7, Shape Detection*. We will not paste the same code owing to space constraints. The implementation is explained briefly as follows:

- Calculate the contours' area. Omit the value that is too small. The left out ones can be regarded as a possible license plate area.
- Calculate the point sum of the contours. Omit the value that is too small. A license plate has at least six or seven characters so that the left out ones can go to the next step.
- Normalize the remaining areas for further calculation.
- Only consider the rectangles with the width-height ratio in the range of appropriate value (for example, 3 to 10).
- Start the OCR engine and store the characters.

In fact, there are many different ways to filter the rectangular area and locate the license plate. Contours' area, sum of points, and ratio are the simplest ideas and are easy to perform.

Other methods, such as `FilterPlate` and the control event, are so easy that we will not talk about them in detail. The `FilterPlate` method is used to filter the noise of the given image, which means it only contains the basic image process functions.

## Output

Compile and start the program; the result should turn out like the following figure. The OCR result can be seen in the right panel as well as the license plate area and filter area. You can change the input image by clicking on the **Open...** button.



Now, if you put a camera right on the road in front of your house, it can monitor all the vehicles that pass by. Have fun!

## Summary

This chapter shows how to do License Plate Recognition and Optical Character Recognition in C# using Emgu CV. Some of the tasks such as basic image processing, Canny Edge Detector, and contours were involved in the previous chapters, but this one involves a more complicated combination of them. The `Emgu.CV.OCR.dll` file can help us with many interesting things. Try to build your own OCR applications and have fun with them.





# 10

## Image Stitching

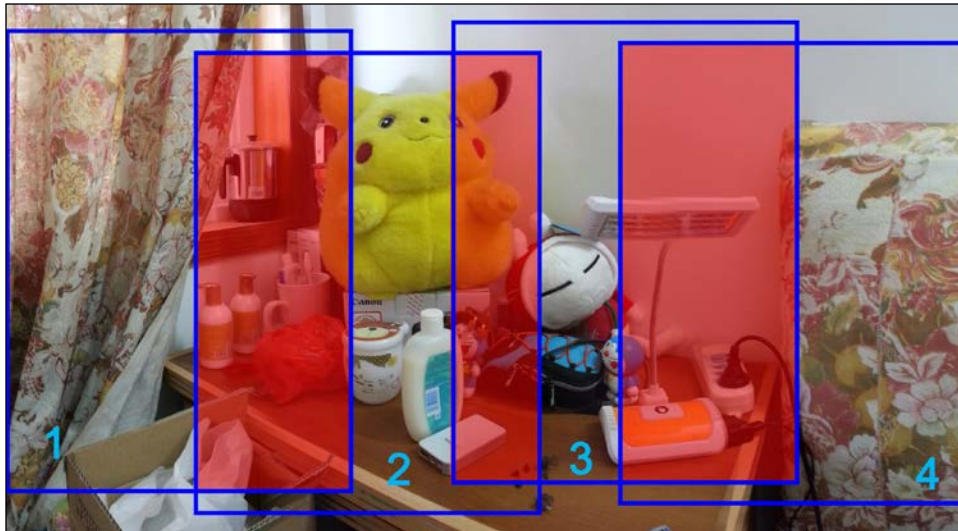
In the previous chapters, we got access to most of Emgu CV's dynamic-link libraries and succeeded in referencing them in our projects. In this chapter, `Emgu.CV.Stitching` will be introduced. As the name suggests, it has something to do with image stitching. Thus, this chapter can be seen as a tutorial of using Emgu CV to stitch images together.

### Image stitching

Image stitching, or sometimes called photo stitching, is defined as the process of assembling several images (usually photographic ones) with overlapping fields of view, to create a new high-resolution image or segmented panorama. A lot of image-processing software allow users to stitch images to a panorama. Some smart phones or digital cameras can also stitch the photos after taking a series of continuous photos. The following figure is an example of a desk's panorama, which is created by a smart phone internally:



Most approaches to image stitching need overlaps between images, so that a seamless panorama can be created. The preceding figure is combined with four different but neighboring images shown as follows:



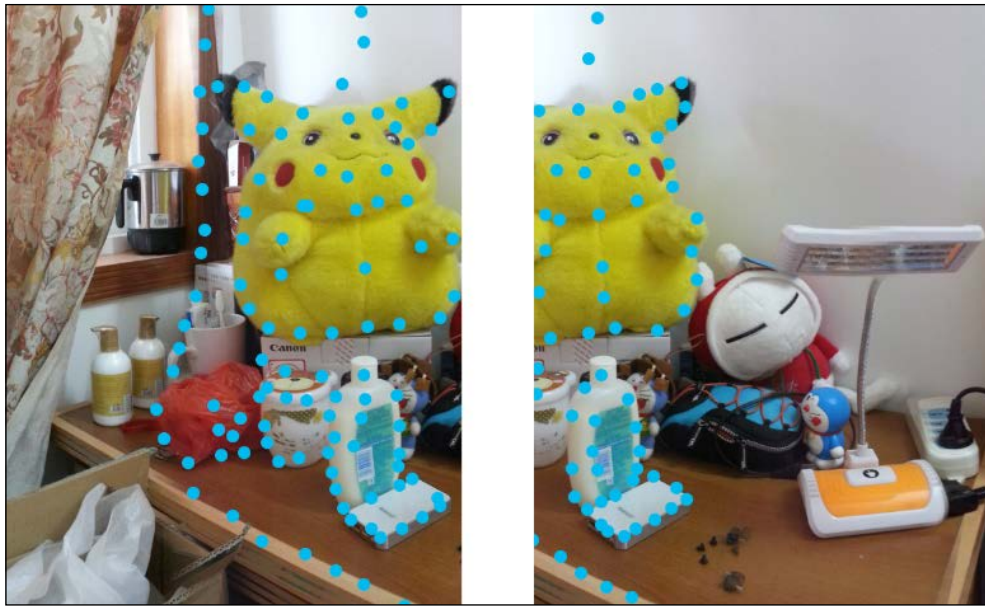
Now we are going to talk about how our computer combines neighboring images together.

## Algorithms for image stitching

There are three main stages during one image stitching process – image matching, image calibration, and image blending.

### Image matching

Neighboring images always share the same parts that exactly overlap. Image matching, also called image registration, aims at searching for image alignments. Here we use matching features to describe local features in an image so that overlapping pixels can be found. The Matching feature is another computer-vision concept, and it is used to compute abstractions of image information at every point. The following figure shows how computers search for the overlaps:



After computing the image features of every point in two images individually, common ones to both images will be found. The next step is calibration.

## Image calibration

During image calibration, differences between neighboring images will be minimized. Optical differences are the most common ones, such as exposure and distortion. In order to minimize them, the data for geometric optimization of all the images will be used, which can be calculated by the image features.

## Image blending

The last stage is image blending, which involves executing the adjustments during calibration, and remapping all the segments into a new panorama. The differences are further minimized. If applicable, the **HDR (High Dynamic Range)** technique can represent more accurately the range of intensity levels. In the end, all the neighboring images will be blended together and all the adjustments will be done so that there is no crack between them.

## Code

Let's create a new project and remember to add the new using the `Emgu.CV.Stitching`. Use the `dataGridView` control to show the input images, and add a button to select the image files. The code of how to store and stitch images is shown as follows:

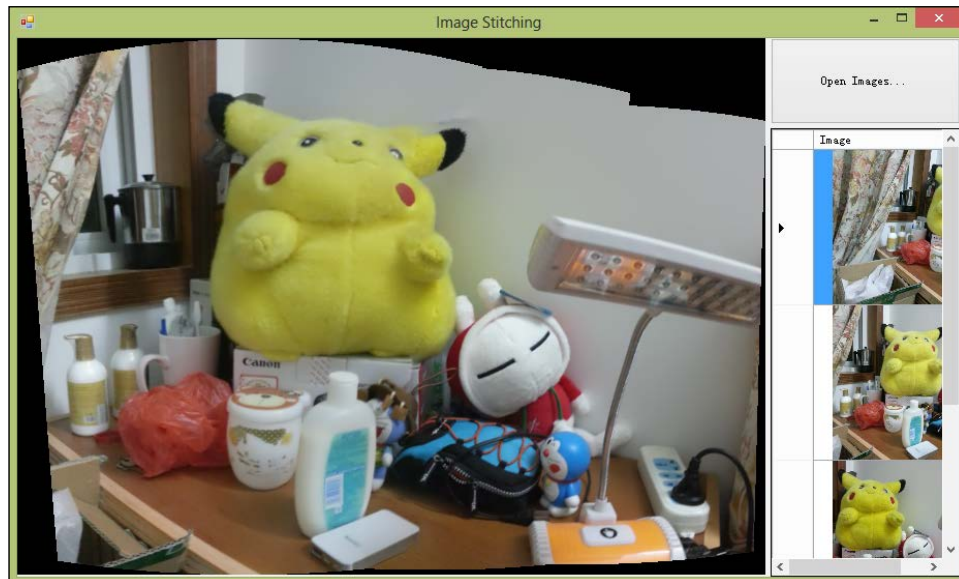
```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Multiselect = true;
    ofd.CheckFileExists = true;
    if (ofd.ShowDialog() == DialogResult.OK)
    {
        //Clear data
        dataGridView1.Rows.Clear();
        //Store input images
        Image<Bgr, Byte>[] images = new Image<Bgr,
            Byte>[ofd.FileNames.Length];
        for (int i = 0; i < images.Length; i++)
        {
            images[i] = new Image<Bgr, Byte>(ofd.FileNames[i]);
            using (Image<Bgr, Byte> thumbnail = images[i].
                Resize(150, 150, Emgu.CV.CvEnum.INTER.
                    CV_INTER_CUBIC, true))
            {
                DataGridViewRow row = dataGridView1.Rows[
                    dataGridView1.Rows.Add()];
                row.Cells["Image"].Value = thumbnail.ToBitmap();
                row.Height = 150;
            }
        }
        //Try Image Stitching
        try
        {
            //Core Part
            using (Stitcher stitcher = new Stitcher(
                // GPU boost enable or disable
                // Must specify false because it will cause error if true
                // The bug is from OpenCV
                false))
            {
                Image<Bgr, Byte> result = stitcher.Stitch(images);
                imageBox1.Image = result;
            }
        }
    }
}
```

```

    }
    finally
    {
        foreach (Image<Bgr, Byte> image in images)
        {
            image.Dispose();
            //or the code below that will make the code efficient
            //((IDisposable)image).Dispose();
        }
    }
}
}

```

Click on the button and select all the neighboring images; the result should turn out to be like the following image:



You can cut the black edge out with further processing. A simple way is to get the four corner pixels that are not black and record the coordinates. Do a straight cut with these four coordinates. Try it yourself.

## Summary

In this last chapter, we introduced image stitching, which is very easy to use but it's also an essential part of Emgu CV. As this book comes to an end, I hope you see the benefits. Try to explore more with Emgu CV. Have fun!



# Index

## A

assumption, LPR 89

## B

biometric systems 73, 74

Browse... button 15

## C

camera capture 75, 76

camera properties

URL 76

Canny Edge Detector 61, 62

CascadeClassifier class 81

CMake command 16-19

code

about 98, 99

for face detection 81-83

coding, Hello World 27, 28

color depth 43, 44

color image representation 41-43

contour

drawing 69

example 70, 72

finding 68

polygon approximation 70

representing 68

contour representation

free chain codes 69

vertex sequence 68

Create button 57

CV component 33

CXCore component 34

## D

data 39, 40

Data property 58

Dependency Walker

URL 21

detectedLicensePlateRegionList

parameter 92

DetectLicensePlate method 91, 92

digital image representation

color depth 43, 44

color image representation 41-43

data 39, 40

pixel 39, 40

pixel resolution 40, 41

Dispose() method 52

Download button 12

## E

elements

operations, using with 58, 59

Emgu CV

about 5, 6, 23

advantages 9, 10

building, from source 17, 18

downloading 11

installing 11

installing, on Linux 16

installing, on OS X 18

installing, on Windows 11-15

Layer 1 34

Layer 2 34

Emgu CV Discussion Forum

URL 9



- Emgu.CV.dll 36
- Emgu.CV.GPU.dll 36
- Emgu.CV.Invoke class 36
- Emgu.CV.ML.dll 36
- Emgu CV troubleshooting
  - Linux 21
  - OS X 22
  - Windows 20, 21
- Emgu.CV.UI.dll 36
- Emgu discussion forum
  - URL 21
- Emgu.Util.dll 36
- enumeration mapping 37

## F

- face detection 77, 78
- Fedora
  - CMake 16
  - installing 16
  - Mono 16
  - OpenCV 16
- file
  - image, loading from 46, 47
- FilterPlate method 92
- FindContours method 68, 69
- FindLicensePlate method 92
- form
  - designing 27
- free chain codes 69
- function mapping 36

## G

- garbage collection 51
- generic operations 51
- GetWhitePixelMask method 90, 91

## H

- HaarCascade class 77
- Haar classifier 77, 78
- Haar-like features 78-80
- HDR (High Dynamic Range) 97
- Hello World
  - in C# 23
  - in C++ 31, 32
  - in VB.NET 30

- Hello World, in C#
  - coding 27, 28
  - form, designing 27
  - new project, creating 24-27
  - output 29
- HighGUI component 34
- HoughCircles method 65, 66
- Hough Circle Transform 65-67
- HoughLinesBinary method 63, 65
- Hough Line Transform 63, 64
- Hough transform
  - about 63
  - Hough Line Transform 63, 64

## I

- image
  - creating 44-46
  - garbage collection 51
  - generic operations 51
  - loading, from file 46, 47
  - method naming rules 49, 50
  - operations, with pixels 47-49
  - operators overload, using 50
  - XML serialization 52, 53
- image blending 97
- image calibration 97
- Image class 55
- image matching 96, 97
- image-processing libraries
  - comparison of 6
  - documentation, comparing 7
  - ease of use 7
  - license agreement 6
  - performance 8, 9
- image stitching
  - about 95, 96
  - image blending 97
  - image calibration 97
  - image matching 96, 97
- Image(TColor, TDepth)() 46
- Image(TColor, TDepth)(Bitmap) 46
- Image(TColor, TDepth)(Image(Gray, TDepth)) 46
- Image(TColor, TDepth)(Int32, Int32) 46
- Image(TColor, TDepth)(Int32, Int32, Int32, IntPtr) 46

**Image(TColor, TDepth)(Int32, Int32, TColor)** 46  
**Image(TColor, TDepth)(SerializationInfo, StreamingContext)** 46  
**Image(TColor, TDepth)(Size)** 46  
**Image(TColor, TDepth)(String)** 46  
**Image(TColor, TDepth)(TDepth[,])** 46  
**img** parameter 92

## L

**Layer 1** 34  
**Layer 2** 34  
**license agreement** 6  
**licensePlateImagesList** parameter 92  
**License Plate Recognition**. *See* **LPR**  
**Linux**  
    dependency, obtaining 16  
    Emgu CV, building from source 17, 18  
    Emgu CV, installing on 16  
    troubleshooting 21

## LPR

    about 85, 86  
    algorithm 86, 87  
    assumption 89  
    code for 88  
    output 93  
    procedures, to identify license plate 86  
    source code 89, 90

## M

**Machine Learning**. *See* **ML**

### matrix

    creating 57, 58  
    operations, with elements 58, 59  
    working with 56

**Matrix class** 55

**method naming rules** 49, 50

**ML** 33, 76, 77

### Mono

    about 16  
    URL 16

**Monodevelop**

    URL 18

## O

**OCR** 85, 87

**Open...** button 46, 93

### OpenCV

    about 16  
    CV component 33  
    CXCore component 34  
    HighGUI component 34  
    ML component 33

### operations

    used, with elements 58, 59

### operators overload

    using 50

**Optical Character Recognition**. *See* **OCR**

### OS X

    building, from source 19, 20  
    dependencies, obtaining 19  
    Emgu CV, installing on 18  
    troubleshooting 22

**output, Hello World** 29

## P

**parameters** 56

### pixel

    about 39, 40  
    operations, using with 47-49

**pixel resolution** 40, 41

**polygon approximation** 70

### project

    creating 24-27

## S

**Set Elements button** 58, 59

**Set Pixels button** 48

### source code

    DetectLicensePlate method 91, 92  
    FindLicensePlate method 92  
    GetWhitePixelMask method 90, 91

### SourceForge

    URL 11

**structure mapping** 36, 37

**supervised learning** 78

## **T**

### **Tesseract-OCR**

about 88

URL 88

### **tessnet2**

URL 88

### **theory**

boosting 78

### **Tutorial page**

URL 9

## **U**

### **Ubuntu**

installing 17

using keyword 52

## **V**

Variables... button 14

vertex sequence 68

## **W**

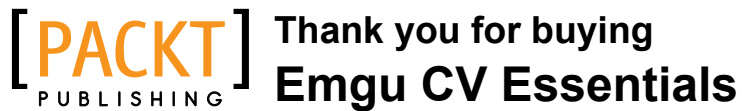
### **Windows**

Emgu CV, installing on 11-15

Windows, troubleshooting 20, 21

## **X**

XML serialization 52, 53



## 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.

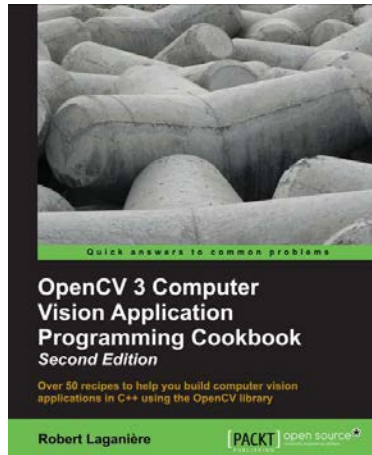
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books 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](http://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](mailto: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.

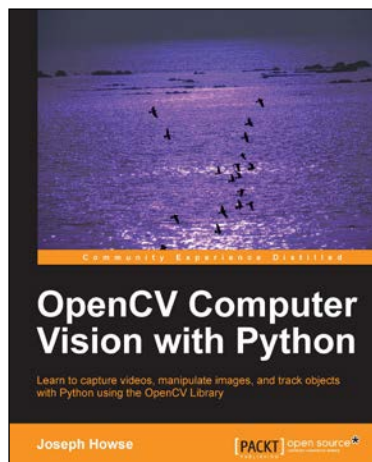


## OpenCV 3 Computer Vision Application Programming Cookbook *Second Edition*

ISBN: 978-1-78216-148-6      Paperback: 350 pages

Over 50 recipes to help you build computer vision applications in C++ using the OpenCV library

1. Master OpenCV, the open source library of the computer vision community
2. Master fundamental concepts in computer vision and image processing
3. Learn the important classes and functions of OpenCV with complete working examples applied on real images



## OpenCV Computer Vision with Python

ISBN: 978-1-78216-392-3      Paperback: 122 pages

Learn to capture videos, manipulate images, and track objects with Python using the OpenCV Library

1. Set up OpenCV, its Python bindings, and optional Kinect drivers on Windows, Mac or Ubuntu
2. Create an application that tracks and manipulates faces
3. Identify face regions using normal color images and depth images

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles



## Instant OpenCV Starter

ISBN: 978-1-78216-881-2

Paperback: 56 pages

Get started with OpenCV using practical, hands-on projects

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results
2. Step by step installation of OpenCV in Windows and Linux
3. Examples and code based on real-life implementation of OpenCV to help the reader understand the importance of this technology
4. Codes and algorithms with detailed explanations



## 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](http://www.PacktPub.com) for information on our titles