
Preface for Chinese Translation

Since the writing of book, OpenCV is now actively supported by Willow Garage (<http://www.willowgarage.com>), a robotics research institute located in Menlo Park, California. During the time when OpenCV had less support, Shiqi Yu helped out by producing a Chinese translation of OpenCV documentation. It is therefore fitting that Shiqi has continued on to produce the Chinese translation of this book. The Chinese translation timing is also nicely aligned with the new release 2.0 of OpenCV in September 2009. You can find links to current information on OpenCV at the main wiki page at <http://opencv.willowgarage.com> which links to the detailed page at <http://opencv.willowgarage.com/wiki/FullOpenCVWiki>. The new OpenCV releases are detailed in <http://opencv.willowgarage.com/wiki>Welcome#Announcements>.

Applications for computer vision and machine perception are growing rapidly. For example, many people are familiar with face detection now available on consumer cameras. Many of those face detection techniques are adapted from the face detection algorithms developed in OpenCV. But, many people are not aware of just how important computer vision already is for manufacturing. Almost nothing is manufactured these days without making use of video inspection equipment, cameras now monitor fruits and vegetables for blemishes, make sure that the labels on products are put on in the right place, watch to make sure cloth has no flaws, or that each pixel works on an LCD screen and much more. Many of these applications make use of OpenCV routines, many such systems are deployed in China.

Computer vision also has growing uses in monitoring and safety. Some people are aware of security cameras in airports and train stations, but fewer people know that cameras are also increasingly used to monitor mine equipment, prevent drownings in swimming pools and watch traffic flow and accidents on freeways. OpenCV, with its BSD license, encourages commercial use and so is deployed on many of these monitoring systems. When you search the web, many of the image processing routines run by Google make use of OpenCV. These uses range from helping stitch satellite and airplane images together in Google Earth and Google Maps, but also to stitch street scenes together and align the imagery with laser scans in Google Street View.

Computer vision has many uses on the Web, such as in the Video Summary produced by Video Surf(<http://www.videosurf.com/>) or for image retrieval by all the major search engines. There is a positive feedback here because the more images there are on the web, the more training data becomes available such as LabelMe(<http://labelme.csail.mit.edu/> data bases) or Tiny Images(<http://people.csail.mit.edu/torralba/tinyimages/collection>). Additionally, more and more researchers are using Amazon's Mechanical Turk service(<https://www.mturk.com/mturk/welcome>) to label images databases for pennies per image and tools are appearing to make such labeling tasks easier to run (<http://pr.willowgarage.com/wiki/ROS/mturk>). The training data from these databases and services helps improve computer vision algorithms.

By coincidence, both authors of the OpenCV book now work in robotics where sensor perception is the main obstacle to enable wide deployment of robotics. Robots will be useful for elderly care, for agriculture, for services and for manufacturing. Robots have been doing well at navigation and mapping(<http://www.youtube.com/watch?v=qRrMHaO6NpE>) but the real key to unleashing a mobile robotics industry is perception for manipulation. Robots need to be able to see objects reliably in order to manipulate and build things. There is still much work to do to make perception reliable, and that is one of the reasons why OpenCV is open — to collect the best work of the top people in order to enable seeing machines. Recently there have been many advances in mobile manipulation(http://www.youtube.com/watch?v=0S2dc_B-6Kg) but much more work remains to be done. Fortunately, OpenCV is keeping up and now does a major release every 6 months with daily improvements available from the source code repository (http://sourceforge.net/scm/?type=svn&group_id=22870).

I thank Shiqi Yu and Ruizhen Liu for their translation of this book. People who work in open source do their work for all of humanity and not for any specific country, company or group. OpenCV was designed to accelerate human knowledge and capability by making advances in knowledge available to everyone. This is an optimistic view of humanity but it depends on contributions back. We encourage top scientists and developers in China to learn OpenCV and to contribute their advances back to the effort.

Gary Bradski
Senior Scientist, Willow Garage
Consulting Professor, Stanford University, CS Dept.
garybradski@gmail.com
September 2009

译者

学习 OpenCV

(中文版)

Gary Bradski Adrian Kaehler 著

于仕琪 刘瑞祯 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权清华大学出版社出版

清华大学出版社

北京

内 容 简 介

计算机视觉是在图像处理的基础上发展起来的新兴学科。OpenCV 是一个开源的计算机视觉库，是英特尔公司资助的两大图像处理利器之一。它为图像处理、模式识别、三维重建、物体跟踪、机器学习和线性代数提供了各种各样的算法。

本书由 OpenCV 发起人所写，站在一线开发人员的角度用通俗易懂的语言解释了 OpenCV 的缘起和计算机视觉基础结构，演示了如何用 OpenCV 和现有的自由代码为各种各样的机器进行编程，这些都有助于读者迅速入门并渐入佳境，兴趣盎然地深入探索计算机视觉领域。

本书可作为信息处理、计算机、机器人、人工智能、遥感图像处理、认知神经科学等有关专业的高年级学生或研究生的教学用书，也可供相关领域的研究工作者参考。

Copyright © 2008 Gary Bradski and Adrian Kaehler. All rights reserved.

Authorized Simplified Chinese translation edition, by O'Reilly Media, Inc., is published by Tsinghua University Press, 2009. Authorized translation of the original English edition, 2008 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书之英文原版由 O'Reilly Media, Inc. 于 2008 年出版。

本书中文简体版由 O'Reilly Media, Inc. 授权清华大学出版社出版 2009 年出版。此翻译版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未经书面许可，本书的任何部分和全部不得以任何形式复制。

北京市版权局著作权合同登记号 图字：01-2009-5150

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

学习 OpenCV(中文版)/(美)布拉德斯基(Bradski, G), (美)克勒(Kaehler, A.)著；于仕琪, 刘瑞祯译。
—北京：清华大学出版社，2009.10

书名原文：Learning OpenCV

ISBN 978-7-302-20993-5

学… II.①布… ②克… ③于… ④刘… III. 图像处理—应用软件 IV. TP391.41

中国版本图书馆 CIP 数据核字(2009)第 151959 号

责任编辑：文开琪

封面设计：Ellie Volckhausen 张 健

版式设计：北京东方人华科技有限公司

责任印制：李红英

出版发行：清华大学出版社 地址：北京清华大学学研大厦 A 座

http://www.tup.com.cn 邮 编：100084

社 总 机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969,c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015,zhiliang@tup.tsinghua.edu.cn

印 刷 者：清华大学印刷厂

装 订 者：北京市密云县京文制本装订厂

经 销：全国新华书店

开 本：178×233 印 张：39.5 字 数：769 千字

版 次：2009 年 10 月第 1 版 印 次：2009 年 10 月第 1 次印刷

印 数：1~4000

定 价：75.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：(010)62770177 转 3103 产品编号：029292-01

目录

出版前言	VI
译者序	XI
写在前面的话	XIII
前言	XV
第 1 章 概述	1
什么是 OpenCV	1
OpenCV 的应用领域	1
什么是计算机视觉	2
OpenCV 的起源	6
下载和安装 OpenCV	8
通过 SVN 获取最新的 OpenCV 代码	11
更多 OpenCV 文档	12
OpenCV 的结构和内容	14
移植性	16
练习	16
第 2 章 OpenCV 入门	18
开始准备	18
初试牛刀——显示图像	19
第二个程序——播放 AVI 视频	21
视频播放控制	23
一个简单的变换	26
一个复杂一点的变换	28

从摄像机读入数据	30
写入 AVI 视频文件	31
小结	33
练习	34
第 3 章 初探 OpenCV.....	35
OpenCV 的基本数据类型	35
CvMat 矩阵结构	38
IplImage 数据结构.....	48
矩阵和图像操作	54
绘图	91
数据存储	98
集成性能基元.....	102
小结	103
练习	103 ..
第 4 章 细说 HighGUI.....	106
一个可移植的图形工具包	106
创建窗口	107
载入图像	108
显示图像	110
视频的处理	120
ConvertImage 函数	125
练习	126
第 5 章 图像处理.....	128
综述	128
平滑处理	128
图像形态学	134
漫水填充算法.....	146
尺寸调整	149
图像金字塔	150

阈值化.....	155
练习	162
第 6 章 图像变换.....	165
概述	165
卷积	165
梯度和 Sobel 导数.....	169
拉普拉斯变换.....	172
Canny 算子.....	173
霍夫变换.....	175
重映射.....	183
拉伸、收缩、扭曲和旋转	185
CartToPolar 与 PolarToCart	196
LogPolar.....	197
离散傅里叶变换(DFT).....	200
离散余弦变换(DCT).....	205
积分图像	206
距离变换	208
直方图均衡化.....	211
练习	213
第 7 章 直方图与匹配.....	216
直方图的基本数据结构	219
访问直方图	221
直方图的基本操作	223
一些更复杂的策略	231
练习	244
第 8 章 轮廓	246
内存	246
序列	248
查找轮廓	259

Freeman 链码	266
轮廓例子	268
另一个轮廓例子	270
深入分析轮廓	271
轮廓的匹配	279
练习	290
第 9 章 图像局部与分割	293
局部与分割	293
背景减除	294
分水岭算法	328
用 Inpainting 修补图像	329
均值漂移分割	331
Delaunay 三角剖分和 Voronoi 划分	333
练习	347
第 10 章 跟踪与运动	350
跟踪基础	350
寻找角点	351
亚像素级角点	353
不变特征	355
光流	356
mean-shift 和 camshift 跟踪	371
运动模板	376
预估器	383
condensation 算法	399
练习	403
第 11 章 摄像机模型与标定	406
摄像机模型	407
标定	414
矫正	430

一次完成标定	432
罗德里格斯变换	437
练习	438
第 12 章 投影与三维视觉	441
投影	441
仿射变换和透视变换	443
POSIT：3D 姿态估计	449
立体成像	452
来自运动的结构	493
二维和三维下的直线拟合	494
练习	498
第 13 章 机器学习	499
什么是机器学习	499
OpenCV 机器学习算法	502
Mahalanobis 距离	516
K 均值	519
朴素贝叶斯分类	524
二叉决策树	527
boosting	537
随机森林	543
人脸识别和 Haar 分类器	549
其他机器学习算法	559
练习	560
第 14 章 OpenCV 的未来	564
过去与未来	564
发展方向	565
OpenCV 与艺术家	568
后记	570

参考文献	571
索引	586
关于作者和译者	599
封面图片	601

出版前言

在 CMU(卡内基·梅隆大学，全球计算机专业三强之一)，A.纽维尔教授时常饱含热情地对学生说：“世界上有这么多‘为什么’‘要是能解决那些问题该有多好啊？’这样的问题仿佛时时刻刻都在呼唤：‘解决我吧，弄清我吧！’像等待着恋人那样在等着我们这些研究者去解决它们。”作为出版工作者，我们时常也能听到这样的声音，吸引着时常以“超级好奇宝宝”自诩的我们循声而去并付诸实践。

在一次偶然的事件中，我们对计算机视觉发生了浓厚的兴趣。这是专门研究如何让机器(即摄像机和计算机)“看”的科学，这些机器可用来定性或定量地分析图像中各目标之间的相互联系，并通过对这些图像内容含义的理解来解释场景。无论是研究人员或从业人员，还是门外汉，这都是一件多么有趣又富有挑战的事情啊！

循着计算机视觉这一主线，OpenCV 自然成为我们感兴趣的焦点。作为一个跨平台的计算机视觉库，OpenCV(Open Source Computer Vision Library，开源的计算机视觉库)最初由 Intel 公司发起并开发，以 BSD 许可证授权发行，可免费用于商业和研究领域。它包含许多常用的算法，已经广泛应用于对实时性要求较高的计算机视觉和模式识别系统的开发。截至 2009 年 8 月，在 sourceforge.net 的下载次数已经超过 2 200 000 次，大量用户来自中国。OpenCV 中文网站(<http://www.opencv.org.cn>)通过提供丰富的中文资料为 OpenCV 在中国的推广做出了巨大贡献，吸引着越来越多的新手参与 OpenCV 的学习、使用和贡献中。

随着对 OpenCV 的深入了解，我们把眼光投向 O'Reilly Media。它是一家在技术圈内享有盛誉的出版公司。我们一向倾慕于其创始人 Tim O'Reilly 的出版理念：*“All of our editors are expected to get their hands dirty with the technology we publish about. Many are former programmers, system administrators, technical writers, or*

practicing scientists, and all are expected to have written at least one successful book of their own. Because we're close to the industry, we know what books are really needed, and we make sure they tell people what they really need to know."

他们在适当的时候推出了 Learning OpenCV 这本优秀的教材。在浮夸之风盛行的当下，书名中的 Learning 显得格外清新、朴实，一种久闻的亲切感油然而生。书中广泛探讨的计算机视觉算法与理论，丰富的实例，清晰的结构，简繁适当的写作风格，无不引人入胜。对于更注重技术细节的专业人士而言，本书作者的背景和本书内容的组织和呈现方式可能更具有吸引力。关于 Gary 和 Adrian 的介绍，可参见书后的“关于作者和译者”。

顺利引进此书之后，更关键的工作之一便是物色“双优”译者。何为双优呢？优秀专业知识背景+优秀的中英文文字功底。我们何其幸运，一发出邀请，国内 OpenCV 的先行者刘瑞祯和于仕琪两位博士便爽快地答应出手相助。感谢他们能在百忙之中贡献自己的休息时间参与本书的翻译，这是源于他们对于 OpenCV 的一往情深，源于他们对于计算机视觉领域难以割舍的情结，源于他们肩负的知识传播的使命感。

在翻译过程中，译者所表现出来的严谨、认真给我们留下了深刻的印象。整个沟通过程是令人愉快的。对于编辑提出的疑问，他们充分体现出他们的专业精神，以科学的态度负责任地加以肯定或否定。在这个互动过程中，我们受益良多。相信在他们的帮助下，学习 OpenCV 将成为一件轻松的事情。

为保证此书的尽善尽美，我们还有幸邀请到清华大学电子工程系博士研究生段菲对本书进行审阅。他曾经翻译过非常畅销的《DirectX 3D 游戏开发编程基础》和《精通 3D 图形编程》。在我们的印象中，他是一个一丝不苟、地道的 science guy，对技术抱有超常的激情。在解决疑问的时候，他会现场进行验算。对于本书，他以专业的眼光进行了仔细审阅。在此向他表示衷心的感谢！

在编辑此书过程中，为方便读者快速定位，为方便读者快速定位自己希望了解的知识点，我们保留了原书索引，并在正文中相应位置标注了原书页码(见标记符【】)，希望能为读者提供少许帮助。

OpenCV 在国内的应用情况如何呢？我们有幸从大恒王亚鹏先生那里获得了答案。作为行业领跑者，他抽出宝贵的休息时间与我们分享了他们最终选用 OpenCV 的历程，也让我们对本书能为 OpenCV 做出贡献大有信心。在此也向他致以诚挚的谢意！

任何一种技术，仅有赞助者是不够的，仅有充满激情的开拓者也不够的，还必须有

执着的“传教士”，还必须有忠实的跟随者。OpenCV 何其幸运，有优秀的公司（早期的 Intel 和现在的 Willow Garage）做支撑，有 Gary Bradski, Adrian Kaehler, 刘瑞祯和于仕琪这样乐于分享的“知识传播者”，有 Google 等优秀企业的开发人员积极参与和无私奉献，它的前景是可以预知的。计算机视觉是一个新兴领域，一个可以由天马行空自由创造的天地，一块等待着您留下脚印的“尚未凝固的水泥地”（注——明可夫斯基教授曾为迷茫中的爱因斯坦开“处方”，指导他大胆创新和开拓）。

亲爱的读者朋友们，这本书是否也能点燃您对您产生这样的激情呢？拿起它，开始令人心动的新旅程吧！正如济慈所说：“Now it appears to me that almost any Man may like the spider spin from his own inwards his own airy Citadel - the points of leaves and twigs on which the spider begins her work are few, and she fills the air with a beautiful circuiting.（在我看来，几乎人人都可以像蜘蛛那样，从体内吐出丝来结成自己的空中堡垒。她开始工作时，只凭借着树叶和树枝的几个尖儿，然后来回兜转，最后竟使空中布满了美丽迂回的路线。）”希望我们也能借助于简单的“树叶和树枝的几个尖儿”，构筑起自己的城堡，计算机相关领域的城堡，共同共享自己微薄的力量。作为这一战线上的盟友，我们期待着您的任何意见和建议，电子邮箱 coo@netease.com 期待着您与我们分享这个旅程中的点点滴滴！

清华大学出版社
2009 年 9 月

译者序

计算机视觉是在图像处理的基础上发展起来的新兴学科，在计算机科学和工程、信号处理、物理学、应用数学和统计学，神经生理学和认知科学等研究方面，在制造业、检验、文档分析、医疗诊断，和军事等领域等各种智能 / 自主应用方面，都有非常广阔的发展前景。

由于涉及到如此多的专业知识，对普通的研究人员而言，计算机视觉颇有些阳春白雪的意味。其实这种意味来自于两个方面，即它是学术研究与工程开发的集合体。纯粹的研究人员，在有好的想法或者概念情况下，需要一个工程开发工具来验证自己的想法，这个开发工具必须是简单而易用的；工程人员则由于专业背景知识的缺乏，非常难以介入到计算机视觉领域。而 OpenCV 恰恰为这两者的结合提供了一个得心应手的开发工具或者应用平台。

OpenCV 作为一个开放源代码的应用平台，最大程度上体现出“众人拾柴火焰高”的开放精神。有大量的 OpenCV 学习资源可以在互联网上找到，这里译者深深感谢互联网的发展，一言以蔽之，没有互联网，就没有 OpenCV。因此 OpenCV 发展到今天，已经快速从少数人的兴趣爱好逐步转变为一个系统的、有科研和商业应用价值的研发平台。

这几年在中国，译者很欣喜地看到越来越多的学生、科研人员和应用开发人员开始在计算机视觉的研究和工程应用领域使用 OpenCV，并逐步把 OpenCV 作为自己所从事职业的一个忠实伙伴。

作为 OpenCV 项目的发起人，Gary Bradski 和 Adrain Kaehler 所撰写的 *Learning OpenCV* 一书，对 OpenCV 的很多基本算法函数都给出了详细的阐述，并且对函数算法的说明也非常到位。在阅读本书的过程中，读者不但有“知其然”，而且有

“知其所以然”的感受。

本书在介绍计算机视觉各个算法思想的同时，通过大量的程序样例，给读者以启发和引导，始终体现出“学以致用”的精神。特别是每章之后的练习，让读者在浏览各章节内容的基础上，借此做更进一步的思考，对读者在视觉算法思想的领悟和视野的拓展大有裨益。“桃李不言，下自成蹊”，对本书真实价值的最有效评判，其实是来自于广大的读者。

翻译本书的过程对于每位译者而言，既是再次学习和思考的历程，也是追寻作者提出问题、分析问题、解决问题的思维过程。“嘤其鸣矣，求其友声”，本书翻译的过程虽然并不短暂，译者却无过多艰辛之感，原因大致是在翻译的路途上，我们既体味到作者在本书中所展现的灵动思维，也感受到广大同行对本书进展的热情关注。换言之，译者不是在独自前行。

参与翻译本书的人员还有徐明亮、孙涛、柴树杉、吴佳、周磊、罗明、武思远、马长正、陈瑞卿等人。感谢他们的辛勤工作。本书的翻译与其说是几个人的工作，毋宁说它是 OpenCV 爱好者集体工作的结晶。译者感谢清华大学出版社给予我们这样一个难得的机会。另外特别感谢文开琪女士在本书译员确定、翻译质量、进度控制等方面指导和工作。

刘瑞祯

2009 年 9 月于北京

写在前面的话

“工欲善其事，必先利其器”，古代的剑客会像爱护自己的手足一样珍惜自己的剑，因为他懂得在决斗中拥有适合自己的武器往往是克敌制胜的关键。对于从事机器视觉应用技术开发的工程师来说，他们所追求的是功能强大同时又快捷高效的工具，既能保证开发出来的视觉系统足以满足复杂应用现场的实际需求，又能快速完成一系列复杂算法的开发。毫无疑问，每个优秀的视觉技术开发人员都会认真地考虑自己所选用的开发工具。如果说 VC++ 是视觉技术开发人员不可或缺、随身必备的军刀，那么 OpenCV 就是他们冲锋陷阵时渴望拥有的冲锋枪，它带给开发人员两个重要的法宝——威力、速度，它对企业和开发人员具有两大“致命”诱惑——开放源码、完全免费。

中科院中国大恒集团下属的北京大恒图像视觉有限公司作为国内最早成立的专业从事机器视觉产品开发的公司，一直专注于自有产品、自有技术的研发，也经历了从最初在 DOS 操作系统下的汇编语言、C 语言一直到目前 Vista 操作系统下的 Visual Studio 等基础开发工具的升级换代过程，到现在形成了 VC++、IPP、OpenCV、Halcon 等多种工具并用的局面。说到 OpenCV，就不得不提起 Intel 公司在 1996 年发布的著名的奔腾处理器和 MMX (Multi Media Extended) 技术，也可以说正是奔腾处理器和 MMX 技术的出现把机器视觉技术在各领域中的实际应用发展推向了快车道。我们都知道基于数字图像处理和模式识别等技术的算法运算量一般都非常之大，所以在早期用计算机对一幅图像做个基本的处理都要花费很长的时间，这一瓶颈严重制约了机器视觉技术在实际应用领域的发展，所以评价一个视觉算法程序开发质量的重要指标之一就是运算速度，一直到现在的多核处理器时代仍然如此。而 Intel 公司的 MMX 技术以及后来的 SSE (Streaming SIMD Extensions) 技术的出现使得机器视觉算法的开发人员看到了希望的曙光，这种基于单指令多数据的多媒体

指令集技术可以使得图像处理算法的运行速度几倍甚至十几倍的提高，然而要想使用好该技术就必须面对令人头疼的汇编语言，算法开发和优化需要花费比较多的时间才能完成。

对于追求开发效率的机器视觉应用开发企业来讲，希望的是既能开发出性能优越的视觉系统，又能尽量提高开发效率、降低成本，大恒图像也在这方面经历了若干次选择。最初是选择了 Intel 公司的 IPL 及 IPP，这里面的函数都是采用了 MMX 或 SSE 技术优化的，是很优秀的图像处理库，但这里面大都是比较基础的图像处理函数，不能满足复杂的应用技术快速开发的要求，而且还有一点就是不能开放源代码。OpenCV 的出现使得每个机器视觉技术的开发人员都眼前一亮，它不仅是完全免费的开源软件，更可贵的是它包含的各类图像处理及识别的函数非常丰富，而且一般都利用 MMX 及 SSE 技术进行了很好的优化！我是从 2001 年开始接触 OpenCV，虽然公司里的算法工程师都很快喜欢上了 OpenCV，虽然我们从 2002 年起就正式地在产品的开发中使用了 OpenCV，虽然 OpenCV 已经成为视觉算法开发部必备的开发工具之一，但说句实在话我一直心有疑虑，我担心的是会不会哪一天 Intel 公司突然宣布 OpenCV 要收费，当然我的担心也是缘于我计划把我们公司自己开发的算法库建立在 OpenCV 的基础之上。幸运的是我的这种担心被 Intel 公司的 IPP 首席设计师李信宏先生化解了，这还要感谢本书的两位译者刘瑞祯博士和于仕琪博士，正式在他们组织的一次 OpenCV 的研讨会上我结识了李信宏先生和来自 OpenCV 开发组的 Vadim Pisarevsky 先生，李信宏先生亲口告诉我说 Intel 公司不会这样做，我信了，我相信 Intel 公司是可以用他们强大的 CPU 的赢利来支持 OpenCV 的，我也因为 OpenCV 成为了 Intel CPU 的忠实拥护者。

当然，除了 IPP 和 OpenCV 之外还有一些非常优秀的职业机器视觉开发软件包，比如大家熟悉的 Matlab、Halcon、Sapera、VisionPro、EVision 等，Matlab 主要是高校里在视觉算法研究方面用的比较广泛，其他几个主要是针对商业应用开发的，虽然这些商业软件对于初级的开发者更容易掌握，但都是收费软件且不开源，所以专业的开发人员更喜欢 OpenCV，因此 OpenCV 目前成为了在从事机器视觉技术开发的企业中广泛使用的开发工具。我相信本书的出版将有助于机器视觉算法开发人员更容易地掌握 OpenCV 这一独特的开发工具，希望有更多的开发人员借此了解 OpenCV，也衷心祝愿 OpenCV 能走得更远、做得更好！

王亚鹏

北京大恒图像视觉有限公司(副总经理)

前言

本书为使用开放源代码计算机视觉库(OpenCV)提供了一个实战指南，同时还介绍了大量计算机视觉领域的背景知识以帮助读者充分使用 OpenCV。

目的

计算机视觉是一个迅速发展的领域，摄像机价格不断降低且功能越来越强、计算能力的普及以及视觉算法的日臻成熟都带动了该领域的发展。OpenCV 在计算机视觉的发展中扮演着重要的角色，它使得数千名研究人员在视觉领域能够获得更高的生产力。由于 OpenCV 专注于实时视觉应用，因此十分有助于学生和专业人员高效完成项目和加快研究进展，这是通过它提供的一个计算机视觉和机器学习基础架构来实现的，这个基础架构过去只是少数设备完善的实验室的专利。本书目的如下。

- 为 OpenCV 提供一份更好的文档——详细说明函数调用约定以及如何正确使用这些函数。
- 快速帮助读者对计算机视觉的算法原理获得直观的理解。
- 让读者认识到可以使用哪些算法，以及应用这些算法的场合。
- 通过许多可用的代码实例，让读者循序渐进地学会如何实现计算机视觉和机器学习算法。
- 培养读者具有一定的直觉，使其在出现问题的时候能够对一些 OpenCV 源代码中更高级的子程序进行修正。

简言之，本书既是在学校时希望使用的教材，也是我们在工作时希望翻阅的参考书。

本书为 OpenCV 这个工具提供了注解，旨在帮助读者快速在计算机视觉领域中开展有趣的工作。本书能帮助读者直观地理解算法的原理，这样可以帮助读者设计和调试视觉系统，并使得其他教材中对计算机视觉和机器学习算法的形式化描述更易于理解和记忆。

总而言之，如果直观地领会了算法的原理，便容易理解复杂的算法和与这些算法相关的数学知识。

本书面向的读者

本书包括算法描述、可运行的例程代码以及对 OpenCV 库中的计算机视觉工具的解释，因此，它应该会对多种类型的读者提供有益的帮助。

专业人员

对于需要迅速实现计算机视觉系统的专业人员来说，例程代码为开始工作提供了一个快速上手的框架。我们对算法原理的直观描述可以迅速教会读者或提示读者其用法。

学生

如我们所说，本书是我们当年在学校时希望使用的教材。直观的解释、详细的文档和例程代码都有助于读者在计算机视觉领域获得迅速成长，完成更多有趣的课堂项目，并且最终为计算机视觉领域贡献新的研究成果。

教师

计算机视觉是一个迅速发展的领域。我们发现，在需要时讲解一些经典的理论、当前的论文或专家的讲稿，学生会迅速地掌握一本课本。同时，学生也可以更早开始一些课程项目，尝试更多有挑战性的任务。

业余爱好者

计算机视觉非常有趣，可任由你天马行空地“创造”！

我们对于为读者提供充分的直观感受、文档以及可运行的代码给予了强烈的关注，

目的是使读者能够迅速实现实时计算机视觉应用程序。

本书声明

本书并不是一本正规教材。毋庸讳言，本书的许多知识点都涉及了大量数学细节^①，但这样做的目的是加深读者对算法的理解，或者讲清楚算法中所用的前提条件。在这里，我们并不打算进行严格的数学推导，这也许会让一些一直用严格数学表达的人感到不习惯。

本书不是为理论研究人员所写，因为它更多地关注应用。本书针对视觉提供通用的知识，而不是仅仅针对计算机视觉的某些特定应用(例如医学图像或遥感分析)。

也就是说，作者深信一点：读完这里的解释之后，学生不仅会更好地学习理论知识，还会将这些知识铭记于心。因此，本书是针对理论课程的理想辅导书，也适用于入门课程或实战性较强的课程。

关于本书中的程序

本书所有的例程都基于 OpenCV 1.0 版本。代码可以在 Linux 或 Windows 下运行，也可能在 OS-X 下运行。本书的例程源代码可以本书的网站 (<http://www.oreilly.com/catalog/9780596516130>) 下载。OpenCV 可以从它的代码管理网站 (<http://sourceforge.net/projects/opencvlibrary>) 下载。

OpenCV 仍在不断发展，每年都会发布一到两个正式版本。一般来说，可以从代码管理网站的 SVN 服务器 (http://sourceforge.net/scm/?type=svn&group_id=22870) 获得最新代码。

预备知识

在大多数情况下，读者只需要知道如何用 C 语言编程，也许需要知道一些 C++ 编程知识。许多数学相关的内容属于选读，并带有特定标记。书中涉及的数学知识包括简单的代数和基本的矩阵代数，并且假定读者较熟悉最小二乘优化问题的求解方法，以及高斯分布、贝叶斯定律和简单函数的求导等一些基本知识。

① 深入数学细节的部分都有一个提示，指出一般读者可以直接跳过此部分内容。

这些数学知识用于帮助读者加深对算法的直观理解。读者可以跳过数学和算法描述，只通过函数定义和范例代码，便可让计算机视觉应用程序启动和运行。

如何充分使用本书

本书不需要按照顺序从头到尾地阅读。它可以作为一种用户手册：在需要的时候，可以从中查找函数；如果想知其所以然，可以阅读函数的描述。然而，本书的设计初衷是更偏向于教程。它帮助读者基本了解计算机视觉，如何以及何时使用所选定的算法。

本书可以作为计算机视觉领域本科生或研究生的辅导书或主要教材。学生阅读本书可迅速了解计算机视觉，然后再辅以其他教材中的理论知识以及本领域内的学术论文，更深入地学习。每一章后面都有练习题，可以帮助测试学生对知识的掌握情况，并加深理解。

您可以通过下面三种方式之一阅读本书。

仅选有用部分

开始阅读本书时，请先阅读第 1 章～第 3 章，然后根据自己需要阅读其他章节。本书不一定要按照顺序阅读，不过第 11 章和第 12 章除外。

最佳进度

一个星期只读两章，直到用六个星期读完第 1 章～第 12 章(第 13 章有些特殊，详见下文讨论)。然后开始项目，着手解决具体问题，并阅读其他教材和相关的论文。

快速掌握

在充分理解内容的前提下尽快浏览本书第 1 章～第 12 章。然后开始项目，着手解决具体问题，并阅读其他的教材和相关的论文。该方法可供专业人员选用，同时也适用于比较高级的计算机视觉课程。

第 13 章的篇幅较多，介绍了机器学习的背景知识、OpenCV 中实现的机器学习算法背后的细节，以及如何使用这些算法。当然，机器学习与物体识别以及计算机视觉的很多方面相关，详细描述需要一本书的篇幅。专业人员会发现，这是未来阅读文献(或直接使用 OpenCV 库中的代码立项)的一个理想起点。对于一般的计算机视

觉课程来说，本章可作为选学内容。

这是作者所希望的教授计算机视觉的方法：学生掌握要点后，快速学完课程内容，然后动手做一些有意义的课堂项目，同时指导老师通过其他教材或论文提供该领域的一些有深度的知识。该方法对小学期、整个学期或两个学期的课程都适用。学生的兴趣和创造力可以被迅速激发起来，很好地将自己所理解的知识和可运行的代码结合起来。当他们开始更有挑战性且更耗时的项目时，指导老师可帮助他们开发和调试复杂的系统。对于课时较多的课程，项目本身可以以项目管理的方式变成教育方式。首先建立其一个可以运行的系统，并优化改进该系统，然后进行研究。课程的目标是每个项目可以发表一篇会议论文，并且在随后(课程结束之后)的工作中发表更多相关论文。

本书所用约定

本书采用如下印刷约定。

斜体

表示新名词，URL，电子邮件地址，文件名，文件扩展名，路径名，目录和Unix实用程序。

等宽字体

表示命令、选项、开关、变量、属性、键值、函数、类型、类、命名空间、方法、模块、参数、参数、值、对象、事件、事件句柄、XML标签、HTML标签、文件内容或者命令输出。

等宽粗体

显示需要用户逐字输入的命令或者其他文字。也用于代码中的强调。

等宽斜体

显示应该被用户输入值代替的文字。

[...]

表示引用参考文献。

注意：该图标表示一个技巧，建议或一般注解。

警告：该图标表示警告或注意事项。

使用例程代码

OpenCV 是免费的，可用于商业和研究，因此对本书的例程代码，我们也持同样的态度。本书例程代码可以用于课程作业、科研或商业产品。如果在使用 OpenCV 时能在参考文献中引用本书，我们将很高兴，但这不是必须的。它如何帮助你完成课程作业(最好保密)这方面的细节可以不必告诉我们，但在借助于 OpenCV 时，我们希望知道您是如何将计算机视觉用于科学研究，课堂教学以及商业产品的。再次强调，这不是必须的，但我们总期待着您能跟我们讲几句。

联系我们

对于本书，如果有任何意见或疑问，请按照以下地址联系本书出版商：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室(100035)
奥莱利技术咨询(北京)有限公司

本书也有相关的网页，我们在上面列出了勘误表、范例以及其他一些信息。你可以访问：

[http://www.oreilly.com/catalog/9780596516246\(英文版\)](http://www.oreilly.com/catalog/9780596516246)
[http://www.oreilly.com.cn/book.php?bn=978-7-?????????-?\(中文版\)](http://www.oreilly.com.cn/book.php?bn=978-7-?????????-?(中文版))

对本书做出评论或者询问技术问题，请发送 E-mail 至：

bookquestions@oreilly.com

希望获得关于本书、会议、资源中心和 O'Reilly 网络的更多信息，请访问：

<http://www.oreilly.com>
<http://www.oreilly.com.cn>

致谢

一个长期的开源项目见证了许多人的参与和离开，每个人都以自己不同的方式做出贡献。OpenCV 的贡献者列表实在太长，无法在此列出，但可以通过随 OpenCV 一起发布的文件 .../opencv/docs/HTML/Contributors/doc_contributors.html 看到所有贡献者。

感谢对 OpenCV 提供帮助的所有人士

Intel 是 OpenCV 的诞生地，它对该项目的全程支持理应得到感谢。开放源代码项目需要一个领军人物和充足的开发支持才能获得突破和迅速发展。Intel 提供了这两个关键条件。无论境况如何，一个公司能够启动这样一个项目并坚持不懈地进行维护，着实难能可贵。自诞生以来，OpenCV 帮助发起了 Intel 的高性能多媒体函数库(IPP)，Intel 的高性能多媒体函数库是一系列手工精心编制的汇编语言子程序，用于计算机视觉、信号处理、语音处理、线性代数等其他领域。现在，OpenCV 也可以通过 Intel 的高性能多媒体函数库提高效率(可选选项)。因此，一个伟大的商业产品和一个开源产品的发展历程是互相关联的。

Mark Holler 是 Intel 的一位研发主管。在比较早的时候，大量时间投入这个非正式的项目，他对此睁一只眼闭一只眼。他的好心得到了好报，他现在在加州酒乡 Napa 的 Mt. Veeder 地区经营一家葡萄酒厂，可尽情享用美酒。Intel 高性能多媒体函数库小组的 Stuart Taylor 允许我们“借用”他的俄罗斯软件团队来帮助 OpenCV。在 OpenCV 发展和存活下来的过程中，Richard Wirt 发挥了关键性作用。作为 Intel 实验室的主要负责人，实验室主任 Bob Liang(梁兆柱博士)使 OpenCV 蓬勃发展；当 Justin Rattner 成为 CTO 时，软件技术实验室为 OpenCV 确立了更加坚定的支持，这时获得了软件大师 Shinn-Horng Lee(李信弘)的支持和以及他的经理 Paul Wiley 的间接支持。在早期，Omid Moghadam 帮助 OpenCV 做了很多宣传工作。Mohammad Haghigat 和 Bill Butera 在技术咨询委员会中做了优秀的工作。Nuriel Amir、Denver Dash、John Mark Agosta 和 Marzia Polito 在启动机器学习库的过程中发挥了关键作用。Rainer Lienhart、Jean-Yves Bouquet、Radek Grzeszczuk 和 Ara Nefian 是 OpenCV 的关键贡献者和优秀的合作者；Rainer Lienhart 现在是一个教授，Jean-Yves Bouquet 现在是研究实验室人员并已经上任。技术贡献者的名字实在太多，无法一一列举。

在软件方面，一些人员特别突出，所以必须提到，特别是俄罗斯软件团队。这些人

的领导者是俄罗斯优秀的程序员 Vadim Pisarevsky，他开发了 OpenCV 的很大一部分，并且在项目从繁荣转为艰难时刻，挤出时间对这个项目进行管理并“抚育”。如果 OpenCV 有一个真英雄的话，那么这个人就是他。他的技术洞察力对本书的写作给予了巨大帮助。在支持不足的时期，Valery Kuriakin 给予了管理支持和保护，他是一个具有伟大天才和智慧的人。还有 Victor Eruhimov，他几乎一直在参与 OpenCV 项目。我们也感谢 Boris Chudinovich 完成了所有轮廓组件的工作。

最后，特别感谢 Willow Garage[WG]公司，不仅因为它对 OpenCV 未来发展的坚实资金支持，而且在本书最后阶段为一个作者提供支持(并提供了点心和饮料)。

对本书帮助的致谢

当准备本书时，有几个关键人物贡献了他们的建议、审阅和意见。非常感谢《纽约时报》的技术记者 John Markoff 的鼓励、关键沟通和实用的写作建议。对于我们的评阅人，要特别感谢加州理工学院的物理学博士后 Evgeniy Bar，每一章他都给出了很多有用的建议；Applied Minds 的 Kjerstin Williams 进行了详细的证明和验证，直至本书完成；Willow Garage 的 John Hsu 测试了所有的例程代码；还有 Vadim Pisarevsky，他仔细阅读了每一章，验证了函数调用和代码，并提供了几个例程代码。还有其他几位评阅人进行了部分章节的评阅，Google 的 Jean-Yves Bouguet 在摄像机标定和立体视觉章节的讨论中给予了巨大帮助。斯坦福大学的 Andrew Ng 教授为机器学习的章节提供了有用的建议。还有数目众多的其他评阅人评阅了不同章节，在此一并对他们表示感谢。当然，如果因为我们的大意或者误解造成的错误，是我们的责任，而不是由于我们收到的建议造成的。

最后，非常感谢我们的编辑 Michael Loukides 的早期支持、大量的编辑工作以及长时间里一直具有的激情。

Gary 谢辞

家里有三个年幼的孩子，我的妻子 Sonya 为本书的出版比我付出了更多的劳动。虽然在人脸识别的例程图像中，OpenCV 让她受到关注，但是仍要向她表达我衷心的感谢和爱意。从更久远来讲，我的技术生涯始于俄勒冈大学物理系，然后是转入加州大学伯克利分校读本科期间。对于读研究生期间，我感谢我的导师，波士顿大学自适应系统中心的 Steve Grossberg 和 Gail Carpenter，我从他们那儿开始了我的学术生涯。虽然他们专注的方向是大脑的数学模型，我已经结束了该研究而专注于人工智能的工程领域，但是我认为我在那儿学到的眼光使我有所不同。在研究生院的

一些前任同事，他们依然是我的亲密朋友并且为本书提供了一些建议、支持甚至进行了一些编辑工作：感谢 Frank Guenther、Andrew Worth、Steve Lehar、Dan Cruthirds、Allen Gove 和 Krishna Govindarajan。

我要特别感谢斯坦福大学，目前我是该大学人工智能和机器人实验室的顾问教授。跟世界上最有头脑的人近距离接触深深影响了我，我曾与 Sebastian Thrun 和 Mike Montemerlo 一起工作把 OpenCV 应用到 Stanley(一个从美国国防部高级研究计划署赢得二百万美元的机器人)，与 Andrew Ng 一起参与 STAIR(最先进的个人机器人之一)，这些团队合作比一个人做有趣得多。这是一个做事全力以赴的实验室，是一个优秀的环境。除了 Sebastian Thrun 和 Andrew Ng，我还要感谢 Daphne Koller 设置了高的科技标准，并让我雇佣一些关键的实习生和学生；还要感谢 Kunle Olukotun 和 Christos Kozyrakis，与他们一起讨论并一起工作。我还要感谢 Oussama Khatib，他在控制方面的工作激发了我现在对虚拟导航机器人控制的兴趣。Intel 的 Horst Haussecker 是一个优秀的同事，他的写书经验帮助我完成了本书。

最后，再次感谢 Willow Garage 允许我在这个世界一级的天才环境里追求我毕生的机器人梦，并且支持我写本书以及支持 OpenCV。

Adrian 谢辞

我最初的学习专业是理论物理，然后是超级计算机设计和数字计算，最后到机器学习和计算机视觉，这是一条很长的经历曲线。在这条学习之路中，很多人给了我巨大的帮助。有许多优秀的教师帮助我，有些是正式的导师，其他的是非正式的指路人。我要特别指出加州大学圣克鲁兹分校的 David Dorfan 教授和斯坦福大学国家加速器实验室的 Hartmut Sadrozinski 教授，在开始阶段他们给了我很大的鼓励，Norman Christ 利用简单的话语“如果你不能用计算机实现，你就不知道你自己在讲什么”教会了我计算的精髓。谨向 James Guzzo 致以特别的感谢，他允许我在 Intel 做一些任务之外的事情，这些年还鼓励我参加 DARPA 无人驾驶汽车大赛。最后，我感谢 Danny Hillis 创造了一个好的环境，在这儿所有的技术可以获得飞跃，并且在 Applied Minds 时鼓励我写本书。

另外要感谢斯坦福大学在这些年里对我特别的支持。从我与 Sebastian Thrun 一起参加无人驾驶汽车大赛团队，到与 Andrew Ng 一起参加 STAIR 机器人项目，斯坦福大学人工智能实验室一直慷慨地提供办公室，资金支持，大部分重要的创意，富有启发性的谈话，并在需要时在视觉、机器人和机器学习方面提供指导。我深深地感激那些在我成长和学习过程中提供重要帮助的人。

除了一份特别的感谢，没有其他的感谢能够表达对我的妻子 Lyssa 的谢意，她一直毫不犹豫地鼓励我参与这个项目，心甘情愿地陪伴我来回出差使我能与 Gary 一起写书。非常感谢她。

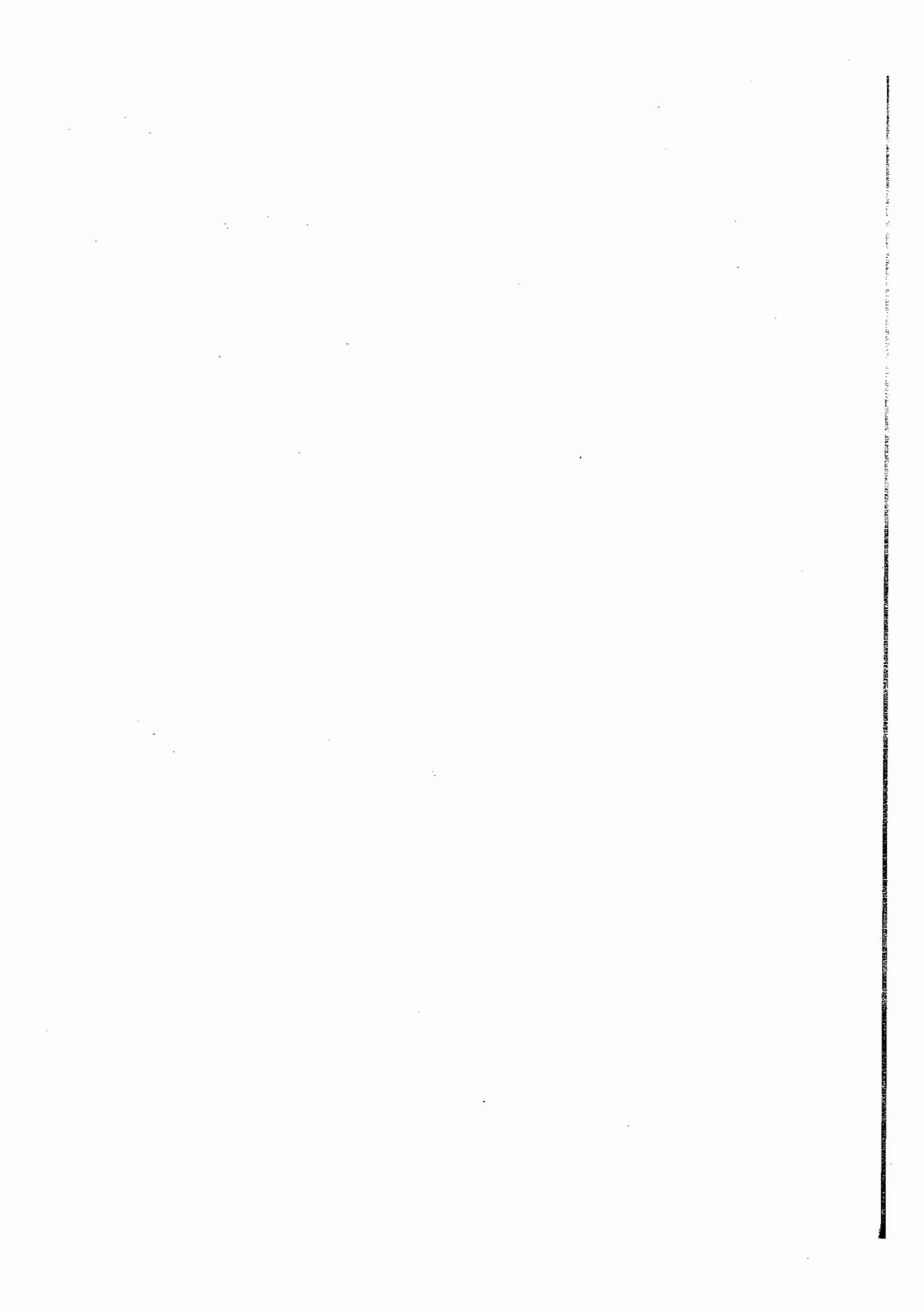
O'Reilly Media, Inc.介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构 O'Reilly Media, Inc. 授权清华大学出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的 The Whole Internet User's Guide & Catalog(被纽约公共图书馆评为 20 世纪最重要的 50 本书之一)到 GNN(最早的 Internet 门户和商业网站)，再到 WebSite(第一个桌面 PC 的 Web 服务器软件)，O'Reilly Media, Inc. 一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc. 是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着，所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。



第 1 章

概述

什么是 OpenCV

OpenCV 是一个开源(参见 <http://opensource.org>)的计算机视觉库，项目主页为 <http://SourceForge.net/projects/opencvllibrary>。OpenCV 采用 C/C++语言编写，可以运行在 Linux/Windows/Mac 等操作系统上。OpenCV 还提供了 Python、Ruby、MATLAB 以及其他语言的接口。

OpenCV 的设计目标是执行速度尽量快，主要关注实时应用。它采用优化的 C 代码编写，能够充分利用多核处理器的优势。如果是希望在 Intel 平台上得到更快的处理速度，可以购买 Intel 的高性能多媒体函数库 IPP(Integrated Performance Primitives)。IPP 库包含许多从底层优化的函数，这些函数涵盖多个应用领域。如果系统已经安装了 IPP 库，OpenCV 会在运行时自动使用相应的 IPP 库。

OpenCV 的一个目标是构建一个简单易用的计算机视觉框架，以帮助开发人员更便捷地设计更复杂的计算机视觉相关应用程序。OpenCV 包含的函数有 500 多个，覆盖了计算机视觉的许多应用领域，如工厂产品检测、医学成像、信息安全、用户界面、摄像机标定、立体视觉和机器人等。因为计算机视觉和机器学习密切相关，所以 OpenCV 还提供了 MLL(Machine Learning Library)机器学习库。该机器学习库侧重于统计方面的模式识别和聚类(clustering)。MLL 除了用在视觉相关的任务中，还可以方便地应用于其他的机器学习场合。

OpenCV 的应用领域

大多数计算机科学家和程序员已经意识到计算机视觉的重要作用。但是很少有人知

道计算机视觉的所有应用。例如，大多数人或多或少地知道计算机视觉可用在监控方面，也知道视觉被越来越多地用在网络图像和视频方面。少数人也了解计算机视觉在游戏界面方面的应用。但是很少有人了解大多数航空和街道地图图像(如 Google 的 Street View)也大量使用计算机定标和图像拼接技术。一些人知道安全监控、无人飞行器或生物医学分析等方面的应用，但是很少人知道机器视觉是多么广泛地被用在工厂中：差不多所有的大规模制造的产品都在流水线上的某个环节上自动使用视觉检测。

【1~2】

OpenCV 所有的开放源代码协议允许你使用 OpenCV 的全部代码或者 OpenCV 的部分代码生成商业产品。使用了 OpenCV 后，你不必对公众开放自己的源代码或改善后的算法，虽然我们非常希望你能够开放源代码。许多公司(IBM, Microsoft, Intel, SONY, Siemens 和 Google 等其他公司)和研究单位(例如斯坦福大学、MIT、CMU、剑桥大学和 INRIA)中的人都广泛使用 OpenCV，其部分原因是 OpenCV 采用了这个宽松的协议。Yahoo groups 里有一个 OpenCV 论坛 (<http://groups.yahoo.com/group/OpenCV>)，用户可以在此发帖提问和讨论；该论坛大约有 20 000 个会员。OpenCV 在全世界广受欢迎，在中国、日本、俄罗斯、欧洲和以色列都有庞大的用户群。

自从 OpenCV 在 1999 年 1 月发布 alpha 版本开始，它就被广泛用在许多应用领域、产品和研究成果中。相关应用包括卫星地图和电子地图的拼接，扫描图像的对齐，医学图像去噪(消噪或滤波)，图像中的物体分析，安全和入侵检测系统，自动监视和安全系统，制造业中的产品质量检测系统，摄像机标定，军事应用，无人飞行器，无人汽车和无人水下机器人。将视觉识别技术用在声谱图上，OpenCV 可以进行声音和音乐识别。在斯坦福大学的 Stanley 机器人项目中，OpenCV 是其视觉系统的关键部分。Stanley 在 DARPA 机器人沙漠挑战赛中，赢得了二百万美元奖金[Thrun06]。

什么是计算机视觉

计算机视觉^①是将来自静止图像或视频的数据转换成一个决策或者一种新的表达方式的过程，所有的这些转换都是为了达到某个目标。输入数据可以包含一些辅助信息，如“摄像机架在汽车上”或“激光扫描仪在 1 米处发现一个物体”。最终的决

① 计算机视觉是一个很广的领域，本书只涉及该领域的一些基本知识。我们推荐的参考书有 Trucco 的教科书[Trucco98](了解计算机视觉)、Forsyth 的教科书[Forsyth03](全面参考)以及 Hartley 的教科书[Hartley06]和 Faugeras 的教科书[Faugeras93](了解三维视觉的工作原理)。

策可能是“场景中有一个人”或“在这个切片中有 14 个肿瘤细胞”。一种新的表达方式可以是将一张彩色照片转为灰度照片，或者从图像序列中去除摄像机晃动影响。

因为人类是视觉动物，所以会误以为可以很容易地实现计算机视觉。当你凝视图像时，从中找到一辆汽车会很困难么？你凭直觉会觉得很容易。人脑将视觉信号划分成很多个通道，将各种不同的信息输入你的大脑。你的大脑有一个关注系统，会根据任务识别出图像的重要部分，并做重点分析，而其他部分则分析得较少。在人类视觉流中存在大量的反馈，但是目前我们对之了解甚少。肌肉控制的传感器以及其他所有传感器的输入信息之间存在广泛的关联，这使得大脑可以依赖从出生以来所学到的信息。大脑中的反馈在信息处理的各个阶段都存在，在传感器硬件(眼睛)中也存在。在眼睛中通过反馈来调节通过瞳孔的进光量，以及调节视网膜表面上的接收单元。

【2~3】

在计算机视觉系统中，计算机接收到的是来自摄像机或者磁盘文件的一个数值矩阵。一般来说，没有内置的模式识别系统，没有自动控制的对焦和光圈，没有多年来经验的积累。视觉系统通常很低级。图 1-1 显示了一辆汽车的图像。在此图中，我们可以看到车的一侧有一个反光镜，而计算机“看”到的只是一个数值的矩阵。矩阵中的每个数值都有很大的噪声成分，所以它仅仅给出很少的信息，这个数值矩阵就是计算机“看”到的全部。我们的任务是将这个具有噪声成分的数值矩阵变成感知：“反光镜”。图 1-2 形象地解释了为什么计算机视觉如此之难。

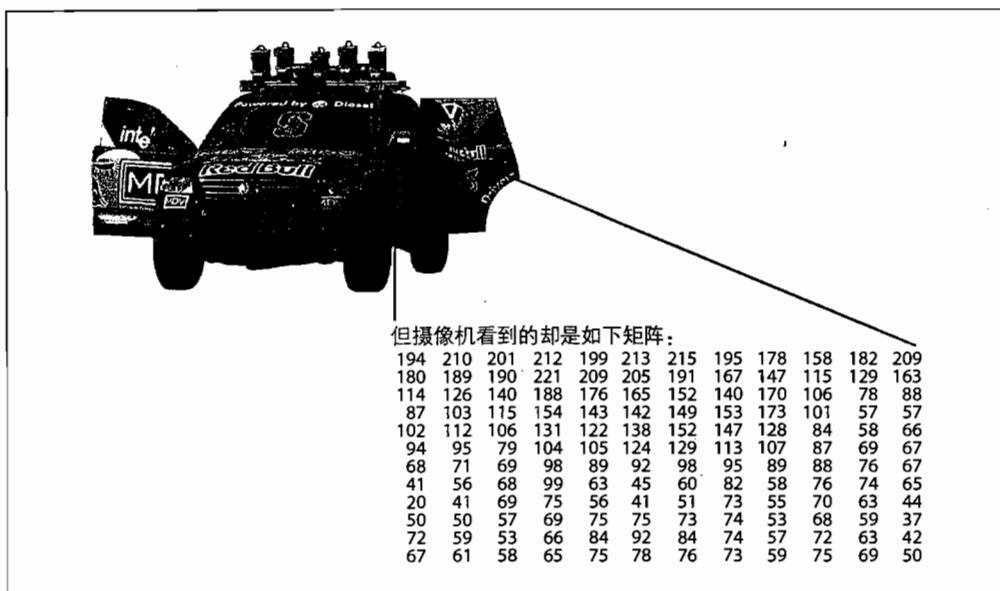


图 1-1：对一个计算机来说，汽车的反光镜只是一个数值矩阵

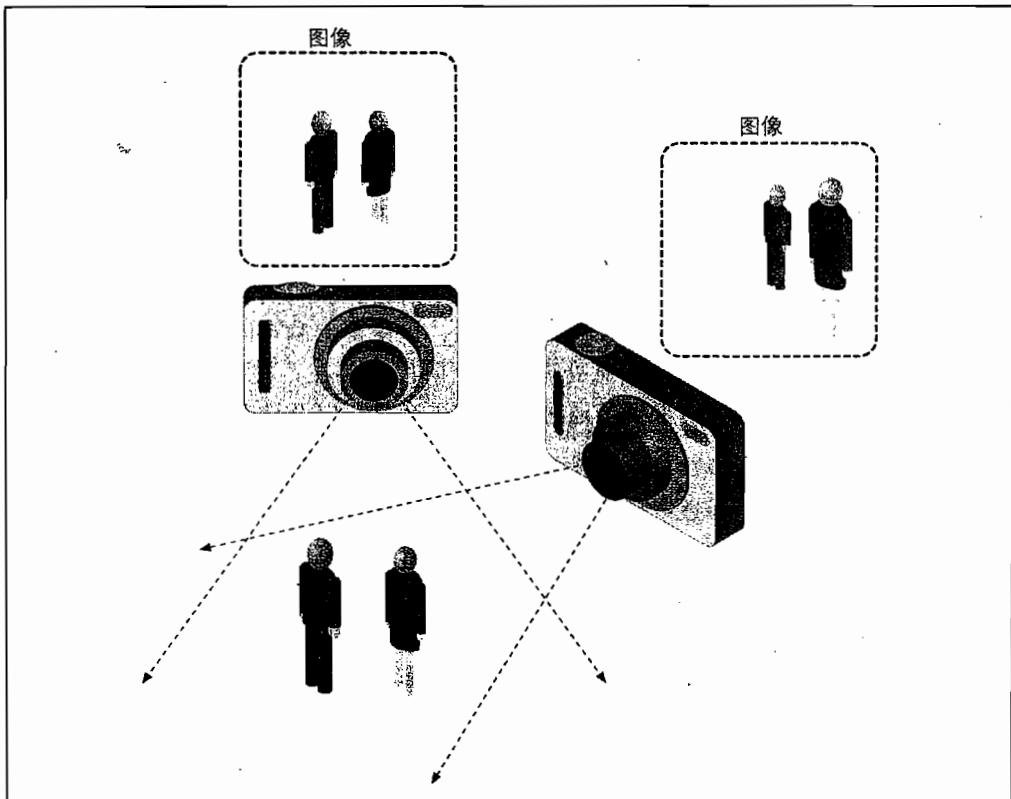


图 1-2：视觉问题的病态本质：随着视点的变化，物体的二维外观会变化很大

实际上，计算机视觉问题比前面我们提到的更糟糕，它是不可解的。给出三维世界的二维视图，是没有固定方法来重建出三维信息的。在理论上，此类病态问题没有唯一和确定的解。即使数据很完美，同一张二维图像也可以表示多种三维场景。然而，如前面提到的，数据会被噪声和形变影响。这些影响来自真实世界的变化(天气、光线、反射、运动)，镜头和机械结构的不完美，传感器上的长时间感应(运动模糊)，传感器上和其他电子器件上的电子噪声，以及图像采集后的图像压缩引入的变化。有如此多令人头疼的问题，我们如何取得进展？

【3~4】

设计实际系统时，为了克服视觉传感器的限制，通常会使用一些其他的上下文知识。考虑这样一个例子，移动机器人在室内寻找并捡起订书机。机器人可以利用这个先验知识：可在办公室内发现桌子，订书机最可能在桌子上被找到。这给出了一个隐含的尺寸参考或参照，也就是订书机能够放在桌子上。这也可用于消除在不可能的地方(例如在天花板或者窗户上)错误识别出订书机的可能性。机器人也完全可以忽略一个 200 英尺大小的跟订书机形状类似的广告飞艇，因为飞艇周围没有桌

子的木纹背景。与之相反，在图像检索中，数据库中的所有订书机图像都是对真正的订书机拍摄的，而且尺寸很大和形状不规划的订书机图像一般不可能被拍到。也就是拍摄者一般只拍摄真正的、普通大小的订书机图像。而且人们拍照时一般会将被拍物体置于中心，且将物体放在最能表现其特征的方向上。因此在由人拍摄的图像中，具有相当多的隐含信息。

【4~5】

我们也可以使用机器学习技术对上下文信息进行显式建模。隐含的变量(例如物体大小、重力方向及其他变量)都可以通过标记好的训练数据里的数值来校正。或者，也可以通过其他的传感器来测量隐含的变量。使用激光扫描仪可以精确测量出一个物体的大小。计算机视觉面临的另一个难题是噪声问题。我们一般使用统计的方法来克服噪声。例如，一般来说不可能通过比较一个点和它紧密相邻的点来检测图像里的边缘。但是如果观察一个局部区域的统计特征，边缘检测会变得容易些。由局部区域卷积的响应连成的点串，构成边缘。另外可以通过时间维度上的统计来抑制噪声。还有一些其他的技术，可以从数据中学习显式模型，来解决噪声和畸变问题。例如镜头畸变，可以通过学习一个简单多项式模型的参数来描述这种畸变，然后可以几乎完全校正这种畸变。

计算机视觉拟根据摄像机数据来采取行动或者做出决策，这样的行动或决策是在一个指特定目的或任务的环境中来解决。我们从图像去除噪声和损坏区域，可以让监控系统在有人爬过栅栏时给出报警，或者在一个游乐园里监控系统能够数出总共有多少人通过了某个区域。在办公室巡游的机器人的视觉软件所采用的方法与固定摄像机的不同，因为这两个系统有不同的应用环境和目标。通用的规律是：对计算机视觉应用环境的约束越多，则越能够使用这些约束来简化问题，从而使最终的解决方案越可靠。

OpenCV 的目标是为解决计算机视觉问题提供基本工具。在有些情况下，它提供的高层函数可以高效地解决计算机视觉中的一些很复杂的问题。当没有高层函数时，它提供的基本函数足够为大多数计算机视觉问题创建一个完整的解决方案。对于后者，有几个经过检验且可靠的使用 OpenCV 的方法；所有这些方法都是首先大量使用 OpenCV 函数来解决问题。一旦设计出解决方案的第一个版本，便会了解它的不足，然后可以使用自己的代码和知识来解决(更为广知的一点是“解决实际遇到的问题，而不是你想像出来的问题”)。你可以使用第一个版本的解决方案作为一个基准，用之评价解决方案的改进程度。解决方案所存在的不足可以通过系统所用的环境限制来解决。

【5~6】

OpenCV 的起源

OpenCV 诞生于 Intel 研究中心，其目的是为了促进 CPU 密集型应用。为了达到这一目的，Intel 启动了多个项目，包括实时光线追踪和三维显示墙。一个在 Intel 工作的 OpenCV 作者在访问一些大学时，注意到许多顶尖大学中的研究组(如 MIT 媒体实验室)拥有很好的内部使用的开放计算机视觉库——(在学生们之间互相传播的代码)，这会帮助一个新生从高的起点开始他/她的计算机视觉研究。这样一个新生可以在以前的基础上继续开始研究，而不用从底层写基本函数。

因此，OpenCV 的目的是开发一个普遍可用的计算机视觉库。在 Intel 的性能库团队的帮助下^①，OpenCV 实现了一些核心代码以及算法，并发给 Intel 俄罗斯的库团队。这就是 OpenCV 的诞生之地：在与软件性能库团队的合作下，它开始于 Intel 的研究中心，并在俄罗斯得到实现和优化。

俄罗斯团队的主要负责人是 Vadim Pisarevsky，他负责管理项目、写代码并优化 OpenCV 的大部分代码，在 OpenCV 中很大一部分功劳都属于他。跟他一起，Victor Eruhimov 帮助开发了早期的架构，Valery Kuriakin 管理俄罗斯实验室并提供了很大的支持。在开始时，OpenCV 有以下三大目标。

- 为基本的视觉应用提供开放且优化的源代码，以促进视觉研究的发展。能有效地避免“闭门造车”。
- 通过提供一个通用的架构来传播视觉知识，开发者可以在这个架构上继续开展工作，所以代码应该是非常易读的且可改写。
- 本库采用的协议不要求商业产品继续开放代码，这使得可移植的、性能被优化的代码可以自由获取，可以促进基于视觉的商业应用的发展。

这些目标说明了 OpenCV 的缘起。计算机视觉应用的发展会增加对快速处理器的需求。与单独销售软件相比，促进处理器的升级会为 Intel 带来更多收入。这也许是为什么这个开放且免费的库出现在一家硬件生产企业中，而不是在一家软件公司中。从某种程度上说，在一家硬件公司里，在软件方面会有更多创新的空间。

【6】

任何开放源代码的努力方面，达到一定的规模使项目自己能够发展是非常重要的。

① Shinn Lee(李信弘)提供了主要的帮助。

目前 OpenCV 已经有大约二百万的下载量，这个数字仍然在以平均每个月 26 000 的下载量递增。OpenCV 用户组大约有 20 000 个会员。OpenCV 吸纳了许多用户的贡献，核心开发工作已经从 Intel 转移到别处^①。OpenCV 过去的开发历程如图 1-3 所示。在发展中，OpenCV 受到网络经济泡沫破裂的影响，也受到无数次管理和发展方向变化的影响。在这些变故中，OpenCV 曾经有多次缺乏 Intel 公司人员的支持。然而，随着多核时代的到来，以及计算机视觉的更多应用的出现，OpenCV 的价值开始提升。现在 OpenCV 在几个研究所中的开发都很活跃，所以不久应该会看到更多的功能出现，如多摄像机标定、深度信息感知、视觉与激光扫描的融合、更好的模式识别算法，同时还会支持机器人视觉的需求。关于 OpenCV 未来的发展，请参考第 14 章。

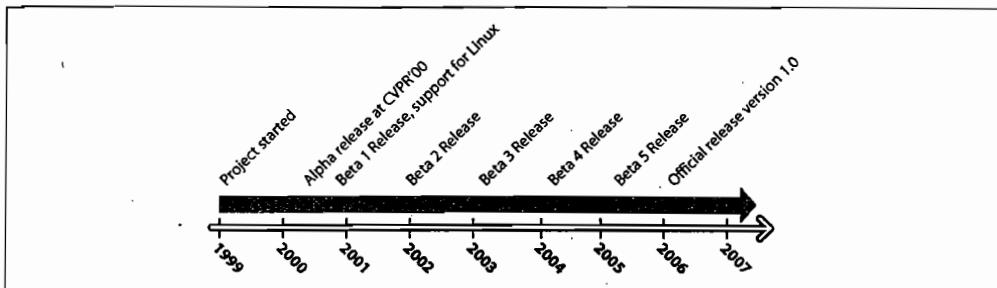


图 1-3：OpenCV 发展路线图

用 IPP 给 OpenCV 加速

因为 OpenCV 曾由 Intel 性能基元(IPP)团队主持，而且几个主要开发者都与 IPP 团队保持着良好的关系，所以 OpenCV 利用了 IPP 高度手工优化的代码来实现加速。使用 IPP 获得的提速是非常显著的。图 1-4 比较了另外两个视觉库 LTI[LTI] 和 VXL[VXL] 与 OpenCV 以及 IPP 优化的 OpenCV 的性能。请注意，性能是 OpenCV 追求的一个关键目标；它需要实时运行代码的能力。

OpenCV 使用优化了的 C 和 C++ 代码实现。它对 IPP 不存在任何依赖。但如果安装了 IPP，那么 OpenCV 将会通过自动载入 IPP 动态链接库来获取 IPP 的优势，来提升速度。

【6~7】

① 撰写此书时，一个机器人研究所和孵化器 Willow Garage [WG] (www.willowgarage.com) 开始积极支持日常的 OpenCV 维护以及在机器人应用领域的开发。

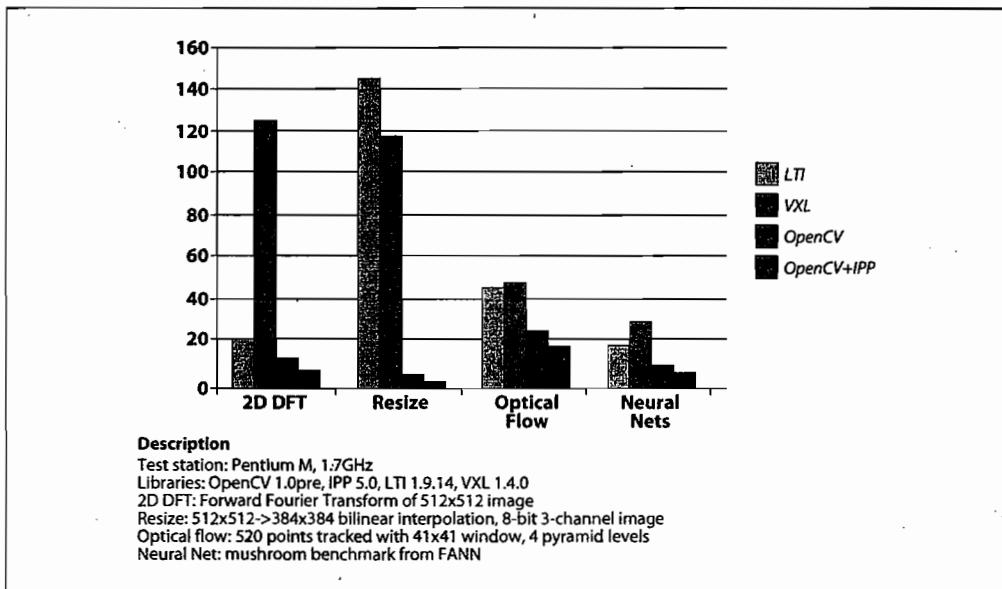


图 1-4：另外两个视觉库(LTI 和 VXL)与 OpenCV(不使用和使用 IPP)的四个不同性能指标的比较：每个指标的四个柱图分别表示四个库的得分，得分与运行时间成正比；在所有指标中，OpenCV 均优于其他的两个库，且用 IPP 优化的 OpenCV 优于没有使用 IPP 优化的 OpenCV

OpenCV 属于谁

虽然 OpenCV 项目是 Intel 发起的，但这个库一直致力于促进商业和研究使用。它是开放源代码且免费的，无论是商业使用还是科研使用，OpenCV 的代码可用于或者嵌入(整体或部分)其他的应用程序中。它不强迫你开放或者免费发放你的源代码。它也不要求你将改进的部分提交到 OpenCV 库中——但我们希望你能够提交。

下载和安装 OpenCV

OpenCV 项目主页在 SourceForge 网站 <http://SourceForge.net/projects/opencvlibrary>，对应的 Wiki 在 <http://opencv.willowgarage.com>。对于 Linux 系统，源代码发布文件为 opencv-1.0.0.tar.gz；对于 Windows 系统，则为 OpenCV_1.0.exe 安装程序。然而，最新的版本始终都在 SourceForge 的 SVN 仓库中。

安装

下载 OpenCV 库之后，就可以安装了。Linux 和 Mac OS 系统的安装细节可以查看.../opencv/目录下的 *INSTALL* 文本文件中的说明。*INSTALL* 文件中还描述了如何编译 OpenCV 和运行测试程序。对于 OpenCV 开发人员，*INSTALL* 还列出了所需的 *autoconf*、*automake*、*libtool* 和 *swig* 其他开发工具。

【8~9】

Windows

从 SourceForge 网站下载 OpenCV 安装程序，然后运行安装程序。安装程序将安装 OpenCV，注册 DirectShow filter，然后进行一些安装后的处理。现在你就可以使用 OpenCV 了。你还可以进入目录.../opencv/_make，使用 MSVC++ 或者 MSVC.NET 2005 打开 *opencv.sln*，或者使用低版本的 MSVC++ 打开 *opencv.dsw*，然后生成 Debug 版的库，也可以重新生成 Release 版的库^①。

如果需要使用 IPP 的优化功能，首先需要从 Intel 网站(<http://www.intel.com/software/products/ipp/index.htm>)获得 IPP 并安装；请使用 5.1 或更新的版本。请确认二进制文件路径(例如 *c:/program files/intel/ipp/5.1/ia32/bin*)被添加到系统环境变量 PATH 中。现在 OpenCV 就能够自动探测到 IPP，并在运行时装载 IPP 了(详细信息请参考第 3 章)。

Linux

因为在 Linux 系统的各个发行版(SuSE, Debian, Ubuntu 等)的 GCC 和 GLIBC 版本并不一样，OpenCV 的 Linux 版本并不包含可直接使用的二进制库。如果发行版没有提供 OpenCV，则需要从源代码重新编译 OpenCV，具体的细节请参考.../opencv/*INSTALL* 文件。

如果要编译库和演示程序，需要版本为 2.x 或更高版本的 GTK+ 及其头文件。除此之外还要需要具有开发文件的 *pkgconfig*, *libpng*, *zlib*, *libjpeg*, *libtiff* 和 *libjasper*。同时还要安装版本为 2.3、2.4 或 2.5 的 Python 及其头文件(开发包)。同时还需要依赖 *ffmpeg* 0.4.9-pre1 或更高的版本中 *libavcodec* 和 *libav** 系列的库，*ffmpeg* 的最新版可以用以下命令获取：*svn checkout svn://svn.mplayerhq.hu/ffmpeg/trunk ffmpeg*。

① 注意，Windows 版本的 OpenCV 只包含 release 版的库，并不包含 debug 版的库。如果要在 debug 模式下使用 OpenCV，则需要自己重新编译 debug 模式的 OpenCV 库。

从 <http://ffmpeg.mplayerhq.hu/download.html> 下载 ffmpeg 库^①， ffmpeg 库的授权协议为 GNU 宽通用公共许可证(LGPL)。非 GPL 软件(如 OpenCV)使用 ffmpeg 库，需要生成和调用共享的 ffmpeg 库：

```
$> ./configure --enable-shared  
$> make  
$> sudo make install
```

编译完成后会生成以下系列库文件：`/usr/local/lib/libavcodec.so.*`，`/usr/local/lib/libavformat.so.*`，`/usr/local/lib/libavutil.so.*`，及其对应的头文件`/usr/local/include/libav*`。

【9】

下载了 OpenCV 后就可以编译 OpenCV 了^②：

```
$> ./configure  
$> make  
$> sudo make install  
$> sudo ldconfig
```

安装完成后，OpenCV 会被默认安装在以下目录：`/usr/local/lib/` 和 `/usr/local/include/opencv/`。因此，用户需要将`/usr/local/lib/`添加到`/etc/ld.so.conf`文件(之后需要执行`ldconfig`命令)，或者将该路径添加到`LD_LIBRARY_PATH`环境变量中。

同样在 Linux 平台也可以用 IPP 给 OpenCV 加速，IPP 的安装细节在前面已经提过。我们现在假设 IPP 安装在以下路径：`/opt/intel/ipp/5.1/ia32/`。修改初始化配置文件，添加`<your install_path>/bin/` 和`<your install_path>/bin/linux32` 到`LD_LIBRARY_PATH`环境变量(可以直接编辑`.bashrc`配置文件)：

```
LD_LIBRARY_PATH=/opt/intel/ipp/5.1/ia32/bin:/opt/intel/ipp/5.1/  
ia32/bin/linux32:$LD_LIBRARY_PATH  
export LD_LIBRARY_PATH
```

另一个方法是将`<your install_path>/bin` 和`<your install_path>/bin/linux32` 添加到

-
- ① 可以用以下命令获取 ffmpeg：`svn checkout svn://svn.mplayerhq.hu/ff_mpeg/trunk ffmpeg`。
 - ② 可以用 Red Hat 的包管理工具(RPMs)编译 OpenCV，编译命令为`rpmbuild -ta OpenCV-x.y.z.tar.gz`(4.X 以上的 rpm) 或`rpm -ta OpenCV-x.y.z.tar.gz`(早期版本的 rpm)，`OpenCV-x.y.z.tar.gz` 应该放在`/usr/src/redhat/SOURCES/`目录或其他类似的目录中。然后使用`rpm -i OpenCV-x.y.z.*.rpm`命令安装 OpenCV。

/etc/ld.so.conf 文件，每个文件占一行，完成后在 root 权限下(也可以使用 sudo 命令)执行 *ldconfig* 命令。

现在 OpenCV 就可以找到并能使用 IPP 的共享库了，具体的细节请参考 */opencv/INSTALL*。

Mac OS X

当写此书的时候，所有的功能都可以在 Mac OS X 下使用，但是仍然有一些限制(如 AVI 文件的写操作)；文件.../*/opencv/INSTALL* 中详细描述了这些限制。

在 Mac OS X 下的编译需求和编译步骤跟 Linux 下类似，但是有如下不同。

- 默认情况下是使用 Carbon 而不是 GTK+。
- 默认情况下是使用 QuickTime 而不是 ffmpeg。
- pkg-config 是非必需的(它只在脚本 *samples/c/build_all.sh* 中用到)
- 默认情况下不支持 RPM 和 ldconfig。使用命令 configure+make+sudo make install 来编译和安装 OpenCV；如果不是使用 ./configure --prefix=/usr 命令来配置的话，需要更新 DYLD_LIBRARY_PATH 变量。

如果要使用全部功能，需要使用 *darwinports* 来安装 *libpng*、*libtiff*、*libjpeg* 和 *libjasper*，然后使它们能够被脚本 *./configure* 检测到(详细帮助请运行 *./configure --help*)。对于大多数信息，可以参考 *OpenCV Wiki*(网址为 <http://opencv.willowgarage.com/>)和 Mac 相关的页面(网址为 http://opencv.willowgarage.com/Mac_OS_X_OpenCV_Port)。

通过 SVN 获取最新的 OpenCV 代码

OpenCV 是一个相对活跃的开发项目，如果提交了 bug 的详细描述以及出错的代码，该 bug 会被很快修复。然而，OpenCV 一般一年才会发布一个或两个官方版本。如果用 OpenCV 开发比较重要的应用，你可能想获得修复了最新 bug 的最新 OpenCV 代码。如果要获取 OpenCV 的最新代码，需要通过 SourceForge 网站上的 OpenCV 库的 SVN(Subversion)获得。

【10~11】

这里并不是一个 SVN 的完整教程。如果你参与过其他的开源项目，也许很熟悉 SVN。如果不了解 SVN，可以参考 Ben Collins-Sussman 等人所著的 *Version*

Control with Subversion(O'Reilly 出版)。SVN 的命令行客户端一般被打包在 Linux、OS X 和大部分类 UNIX 系统中。对于 Windows 系统的用户，可以选择 TortoiseSVN(<http://tortoisessvn.tigris.org/>)客户端，很多命令被集成到 Windows 资源管理器的右键菜单中，使用很方便。

对于 Windows 用户，可使用 TortoiseSVN 检出最新源代码，检出地址为 <https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary/trunk>。

对于 Linux 用户，可以使用如下命令检出最新源代码：

```
svn co  
https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary  
/trunk
```

更多 OpenCV 文档

OpenCV 的主要文档是随源代码一起发布的 HTML 帮助文件。除此之外，网上的参考文档还有 OpenCV Wiki 网站和以前的 HTML 帮助。

HTML 帮助文档

安装 OpenCV 后，在`.../opencv/docs`子目录中有相应的 HTML 格式的帮助文件，打开 `index.htm` 文件，其中包含以下链接。

CXCORE

包含数据结构、矩阵运算、数据变换、对象持久(object persistence)、内存管理、错误处理、动态装载、绘图、文本和基本的数学功能等。

CV

包含图像处理、图像结构分析、运动描述和跟踪、模式识别和摄像机标定。

Machine Learning (ML)

包含许多聚类、分类和数据分析函数。

HighGUI

包含图形用户界面和图像/视频的读/写。

CVCAM

摄像机接口，在 OpenCV 1.0 以后的版本中被移除。

Haartraining

如何训练 boosted 级联物体分类器。文档在文件 `.../opencv/apps/HaarTraining/doc/haartraining.htm` 中。

目录`.../opencv/docs`中还有一个 *IPLMAN.pdf* 文件，它是 OpenCV 的早期文档。这个文档已经过时，阅读时一定要注意。但是这个文档中详细描述了一些算法以及某些算法中应该使用何种类型的图像。当然，本书是详细描述这些图像和算法的最佳参考资料。

Wiki 帮助文档

OpenCV 的文档 Wiki 所包含的内容比 OpenCV 安装文件自带的 HTML 帮助更新，并包含自带文档没有的一些内容。Wiki 网址为 <http://opencv.willowgarage.com>，它包含以下内容：

- 用 Eclipse 集成开发环境编译 OpenCV 的帮助
- 使用 OpenCV 进行人脸识别
- 视频监控
- 使用向导
- 摄像机支持
- 中文和韩文网站链接

另外一个 Wiki 地址为 <http://opencv.willowgarage.com/wiki/CvAux>，是下一节“OpenCV 架构和内容”提到的辅助函数的唯一文档。CvAux 模块包含以下信息：

- 双目匹配
- 多摄像机情况下的视点渐变
- 立体视觉中的三维跟踪
- 用于物体识别的 PCA 方法
- 嵌入隐马尔可夫模型(HMM)

OpenCV 中文的 Wiki 地址为 <http://www.opencv.org.cn/>。

刚才提到的帮助文档并没有解释下面的问题：

- 哪些类型(浮点、整数、单字节；1-3 通道)的图像适用于某个函数？
- 哪些函数可以以 in place 模式(输入和输出使用同一个图像结构)调用？
- 一些复杂函数的调用细节(如 contours)？
- 目录 .../opencv/samples/c/中各个例子的运行细节？
- 为什么要这样使用函数，而不仅仅是如何使用？
- 怎么样设置某些函数的参数？

本书的目的是解答上述问题。

OpenCV 的结构和内容

OpenCV 主体分为五个模块，其中四个模块如图 1-5 所示。OpenCV 的 CV 模块包含基本的图像处理函数和高级的计算机视觉算法。ML 是机器学习库，包含一些基于统计的分类和聚类工具。HighGUI 包含图像和视频输入/输出的函数。CXCore 包含 OpenCV 的一些基本数据结构和相关函数。

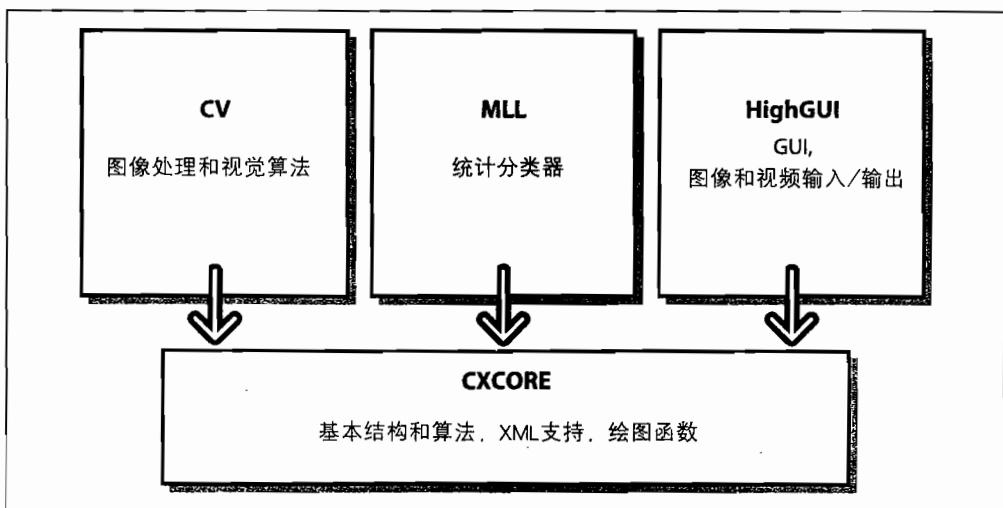


图 1-5：OpenCV 的基本结构

图 1-5 中并没有包含 CvAux 模块，该模块中一般存放一些即将被淘汰的算法和函数(如基于嵌入式隐马尔可夫模型的人脸识别算法)，同时还有一些新出现的实验性的算法和函数(如背景和前景的分割)。CvAux 在 Wiki 中并没有很完整的文档，而在.../opencv/docs 子目录下的 CvAux 文档也不是很完整。CvAux 包含以下一些内容。

- 特征物体，它是一个模式识别领域里用于降低计算量的方法，本质上，依然是模板匹配。
- 一维和二维隐马尔可夫模型(HMM)，它是一个基于统计的识别方法，用动态规划来求解。
- 嵌入式 HMM(一个父 HMM 的观测量本身也符合 HMM)
- 通过立体视觉来实现的动作识别
- Delaunay 三角划分、序列等方法的扩展
- 立体视觉
- 基于轮廓线的形状匹配
- 纹理描述
- 眼睛和嘴跟踪
- 3D 跟踪
- 寻找场景中的物体的骨架(中心线)
- 通过两个不同视角的图像合成中间图像
- 前景/背景分割
- 视频监控(请参考 Wiki 的 FAQ 获得更多资料)
- 摄像机标定的 C++类(C 函数和引擎已经在 CV 模块中)

未来一些特性可能被合并到 CV 模块，还有一些可能永远留在 CvAux 中。

【13~14】

移植性

OpenCV 被设计为可移植的库。它的代码可以用 Borland C++, MS VC++, Intel 等编译器编译。为了使得跨平台更容易实现，C/C++代码必须按照通用的标准来编写。图 1-6 显示了目前已知的可以运行 OpenCV 在各种系统平台。基于 32 位 Intel 架构(IA32)的 Windows 系统支持最好，然后是 IA32 架构的 Linux 平台。对于 Mac OS X 平台的支持，只有在 Apple 采用 Intel 处理器后才提上议程。(在 OS X 平台上的移植目前还不像 Windows 和 Linux 平台上一样成熟，但是已在快速完善中。)成熟度次之的是在扩展内存上 64 位(EM64T)和 64 位 Intel 架构(IA64)。最不成熟的是 Sun 的硬件和其他操作系统。

	IA32	EM64T	IA64	Other (PPC, Sparc)
Windows	✓ (w. IPP; MSVC6, .NET2005+OMP, ICC, GCC, BCC)	✓ (w. IPP; MSVC6+PSDK.NET2005+OMP, PSDK)	✗ (w. IPP; PSDK, some tests fail)	N/A
Linux	✓ (w. IPP; GCC, BCC)	✓ (w. IPP; GCC, BCC)	✓ (GCC, ICC)	✗
MacOSX	✓ (w. IPP, GCC, native APIs)	✗ (not tested)	N/A	✓ (iMac G5, GCC, native APIs)
Others (BSD, Solaris...)	✗	✗	✗	Reported to build on UltraSparc Solaris

图 1-6：OpenCV 1.0 移植指南

如果某个 CPU 架构或操作系统没有出现在图 1-6 中，并不意味着在那上面不能使用 OpenCV。OpenCV 几乎可用于所有的商业系统，从 PowerPC Mac 到机器狗。OpenCV 同样可以很好的运行在 AMD 处理器上，IPP 也会采用 AMD 处理器里的多媒体扩展技术(MMX 等)技术进行加速。

【14~15】

练习

1. 下载并安装最新的 OpenCV 版本，然后分别在 debug 和 release 模式下编译 OpenCV。

2. 通过 SVN 下载 OpenCV 的最新代码，然后编译。
3. 在三维信息转换为二维表示时，存在一些有歧义的描述，请描述至少三个歧义描述，并提供克服这些问题的方法。

OpenCV 入门

开始准备

安装完 OpenCV 开发包后，我们的首要任务自然是立即用它来做一些有趣的事情。为此，还需要搭建编程环境。

在 Visual Studio 开发环境中，我们需要先创建一个项目(project)，然后配置好它的各项设置，以使 OpenCV 开发包中的 *highgui.lib*, *cxcore.lib*, *ml.lib* 和 *cv.lib* 库能被正确链接^①，并保证编译器的预处理器能搜索到“*OpenCV .../opencv/*/include*”目录下的各个头文件。OpenCV 开发包中有许多“*include*”目录，它们一般都位于“*C:/program files/opencv/cv/include*”^②，“*.../opencv/cxcore/include*”，“*.../opencv/ml/include*”和“*.../opencv/otherlibs/ highgui*”路径下。完成这些工作后，便可新建一个 C 程序文件并编写你的第一个程序。

注意：有一些重要的头文件可以使工作变得轻松。在头文件“*.../opencv/cxcore/include/cxtypes.h*”和“*cxmisc.h*”中，包含许多有用的宏定义。使用这些宏，仅用一行程序便可完成结构体和数组的初始化、对链表进行排序等工作。在编译时，有几个头文件非常重要，它们分别是：机器视觉中所要用到的

-
- ① 在 debug 模式下编译程序，需要将链接库设置为 *highguid.lib*, *cxcored.lib*, *mld.lib* 和 *cvd.lib*。
 - ② 在 Windows 系统下，OpenCV 的安装目录默认为“*C:/program files/*”，但也可以选择安装到其他目录下。在表述中为了避免混淆，本书从现在开始都使用“*.../opencv/*”表示读者本机上的 opencv 安装目录路径。

“`.../cv/include/cv.h`” 和 “`.../cxcore/include/cxcore.h`”，I/O 操作中所要用到的 “`.../otherlibs/highgui/include/highgui.h`”；机器学习中所要用到的 “`.../ml/include/ml.h`”。

初试牛刀——显示图像

OpenCV 开发包提供了读取各种类型的图像文件、视频内容以及摄像机输入的功能。这些功能是 OpenCV 开发包中所包含的 HighGUI 工具集的一部分。我们将使用其中的一些功能编写一段简单的程序，用以读取并在屏幕上显示一张图像。如例 2-1 所示。

例 2-1：用于从磁盘加载并在屏幕上显示一幅图像的简单 OpenCV 程序

```
#include "highgui.h"

int main( int argc, char** argv ) {
    IplImage* img = cvLoadImage( argv[1] );
    cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE );
    cvShowImage( "Example1", img );
    cvWaitKey(0);
    cvReleaseImage( &img );
    cvDestroyWindow( "Example1" );
}
```

当以上程序编译后，我们就可以在命令行模式下通过输入一个参数执行它。执行时，该程序将向内存加载一幅图像，并将该图像显示于屏幕上，直至按下键盘的任意一个键后它才关闭窗口并退出程序。下面我们将对以上代码做逐行分析，并仔细讲解每个函数的用法以帮助读者理解这段程序。

```
IplImage* img = cvLoadImage( argv[1] );
```

该行程序的功能是将图像文件^①加载至内存。`cvLoadImage()`函数是一个高层调用接口，它通过文件名确定被加载文件的格式；并且该函数将自动分配图像数据结构所需的内存。需要指出的是，`cvLoadImage()`函数可读取绝大多数格式类型的图像文件，这些类型包括 BMP, DIB, JPEG, JPE, PNG, PBM, PGM, PPM, SR, RAS 和 TIFF。该函数执行完后将返回一个指针，此指针指向一块为描述该图像文件的数

① 在实际代码中，我们应该检查命令行参数 `argv[1]` 所指示的文件是否存在，若不存在，应向用户发送一条错误提示信息。在本书中，我假定读者都已了解此类用于异常处理的代码的重要性和必要性，所以省略了这些代码。

据结构(IplImage)而分配的内存块。IplImage 结构体将是我们在使用 OpenCV 时会最常用到的数据结构。OpenCV 使用 IplImage 结构体处理诸如单通道(single-channel)、多通道(multichannel)、整型的(integer-valued)、浮点型的(floating-point-valued)等所有类型的图像文件。

```
cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE );
```

cvNamedWindow() 函数也是一个高层调用接口，该函数由 HighGUI 库提供。cvNamedWindow() 函数用于在屏幕上创建一个窗口，将被显示的图像包含于该窗口中。函数的第一个参数指定了该窗口的窗口标题(本例中为"Example1")，如果要使用 HighGUI 库所提供的其他函数与该窗口进行交互时，我们将通过该参数值引用这个窗口。

cvNamedWindow() 函数的第二个参数定义了窗口的属性。该参数可被设置为 0(默认值)或 CV_WINDOW_AUTOSIZE，设置为 0 时，窗口的大小不会因图像的大小而改变，图像只能在窗口中根据窗口的大小进行拉伸或缩放；而设置为 CV_WINDOW_AUTOSIZE 时，窗口则会根据图像的实际大小自动进行拉伸或缩放，以容纳图像。

```
cvShowImage( "Example1", img );
```

只要有一个与某个图像文件相对应的 IplImage*类型的指针，我们就可以在一个已创建好的窗口(使用 cvNamedWindow() 函数创建)中使用 cvShowImage() 函数显示该图像。cvShowImage() 函数通过设置其第一个参数确定在哪个已存在的窗口中显示图像。cvShowImage() 函数被调用时，该窗口将被重新绘制，并且图像也会显示在窗口中。如果该窗口在创建时被指定 CV_WINDOW_AUTOSIZE 标志作为 cvNamedWindow() 函数的第二个参数，该窗口将根据图像的大小自动调整为与图像一致。

```
cvWaitKey(0);
```

【17~18】

cvWaitKey() 函数的功能是使程序暂停，等待用户触发一个按键操作。但如果将该函数参数设为一个正数，则程序将暂停一段时间，时间长为该整数值个毫秒单位，然后继续执行程序，即使用户没有按下任何键。当设置该函数参数为 0 或负数时，程序将一直等待用户触发按键操作。

```
cvReleaseImage( &img );
```

一旦用完加载到内存的图像文件，我们就可以释放为该图像文件所分配的内存。我们通过为 cvReleaseImage() 函数传递一个类型为 IplImage* 的指针参数调用该函数，用以执行内存释放操作。对 cvReleaseImage() 函数的调用执行完毕后，img 指针将被设置为 NULL。

```
cvDestroyWindow( "Example1" );
```

最后，可以销毁显示图像文件的窗口。cvDestroyWindow() 函数将关闭窗口，并同时释放为该窗口所分配的所有内存(包括窗口内部的图像内存缓冲区，该缓冲区中保存了与 img 指针相关的图像文件像素信息的一个副本)。因为当应用程序的窗口被关闭时，该应用程序窗口所占用的一切资源都会由操作系统自动释放，所以对一些简单程序，不必调用 cvDestroyWindow() 或 cvReleaseImage() 函数显式释放资源。但是，养成习惯每次都调用这些函数显式释放资源总是有好处的。

尽管现在已经有了这个可供我们随意把玩的简单程序，但我们并不能就此停滞不前。我们的下一个任务将要去创建一个几乎与例 2-1 一样非常简单的程序——编写程序读取并播放 AVI 视频文件。完成这个新任务后，需要开始深入思考一些问题了。

第二个程序——播放 AVI 视频

使用 OpenCV 播放视频，几乎与使用它来显示图像一样容易。播放视频时只需要处理的新问题是如何循环地顺序读取视频中的每一帧，以及如何从枯燥的电影视频的读取中退出该循环操作。具体如例 2-2 所示。

例 2-2：一个简单的 OpenCV 程序，用于播放硬盘中的视频文件

```
#include "highgui.h"

int main( int argc, char** argv ) {
    cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );
    CvCapture* capture = cvCreateFileCapture( argv[1] );
    IplImage* frame;
    while(1) {
        frame = cvQueryFrame( capture );
        if( !frame ) break;
        cvShowImage( "Example2", frame );
        char c = cvWaitKey(33);
```

```
    if( c == 27 ) break;
}
cvReleaseCapture( &capture );
cvDestroyWindow( "Example2" );
}
```

【18】

这里我们还是通过前面的方法创建一个命名窗口，在“例 2”中，事情变得更加有趣了。

```
CvCapture* capture = cvCreateFileCapture( argv[1] );
```

函数 `cvCreateFileCapture()` 通过参数设置确定要读入的 AVI 文件，返回一个指向 `CvCapture` 结构的指针。这个结构包括了所有关于要读入 AVI 文件的信息，其中包含状态信息。在调用这个函数后，返回指针所指向的 `CvCapture` 结构被初始化到所对应 AVI 文件的开头。

```
frame = cvQueryFrame( capture );
```

一旦进入 `while(1)` 循环，我们便开始读入 AVI 文件，`cvQueryFrame` 的参数为 `CvCapture` 结构的指针。用来将下一帧视频文件载入内存（实际是填充或更新 `CvCapture` 结构中）。返回一个对应当前帧的指针。与 `cvLoadImage` 不同的是，`cvLoadImage` 为图像分配内存空间，而 `cvQueryFrame` 使用已经在 `cvCapture` 结构中分配好的内存。这样的话，就没有必要通过 `cvReleaseImage()` 对这个返回的图像指针进行释放，当 `CvCapture` 结构被释放后，每一帧图像所对应的内存空间即会被释放。

```
c = cvWaitKey(33);
if( c == 27 ) break;
```

当前帧被显示后，我们会等待 33 ms。^①如果其间用户触发了一个按键，`c` 会被设置成这个按键的 ASCII 码，否则，`c` 会被设置成 -1。如果用户触发了 ESC 键 (ASCII 27)，循环被退出，读入视频停止。否则 33 ms 以后继续执行循环。

需要指出的是，在这个简单的例子程序中，我们没有使用任何准确的方法来控制视频帧率。我们只是简单的通过 `cvWaitKey` 来以固定时间间隔载入帧图像，在一个精度要求更高的程序中，通过从 `CvCapture` 结构体中读取实际帧率是一个更好的

① 可以等待任意数量的单位时间。在这里，我们只是简单假设每秒需要播放 30 帧并且在任意两帧之间允许用户中断（所以我们每帧之间停顿 33 ms）。事实上，我们可以通过 `cvCaptureFromCamera()` 返回的 `CvCapture` 结构来更准确地确定视频的帧率（详情请参见第 4 章）。

方法！

```
cvReleaseCapture( &capture );
```

退出循环体(视频文件已经读入结束或者用户触发了 Esc 键)后，我们应该释放为 cvCapture 结构开辟的内存空间，这同时也会关闭所有打开的 AVI 文件相关的文件句柄。

视频播放控制

完成了前面的程序以后，现在我们可对其加以改进，进一步探索更多可用的功能。首先会注意到，例 2-2 所实现的 AVI 播放器无法在视频播放时进行快速拖动。我们的下一个任务就是通过加入一个滚动条来实现这个功能。【19】

HighGUI 工具包不仅提供了我们刚刚使用的一些简单的显示函数，还包括一些图像和视频控制方法。其中一个经常使用的就是滚动条，滚动条可以使我们方便地从视频的一帧跳到另外一帧。我们通过调用 cvCreateTrackbar() 来创建一个滚动条，并且通过设置参数确定滚动条所属于的窗口。为了获得所需的功能，只需要提供一个回调函数。具体如例 2-3 所示。

例 2-3：用来添加滚动条到基本浏览窗口的程序：拖动滚动条，函数 onTrackSlide() 便被调用并被传入滚动条新的状态值

```
#include "cv.h"
#include "highgui.h"

int g_slider_position = 0;
CvCapture* g_capture = NULL;

void onTrackbarSlide(int pos) {
    cvSetCaptureProperty(
        g_capture,
        CV_CAP_PROP_POS_FRAMES,
        pos
    );
}

int main( int argc, char** argv ) {
    cvNamedWindow( "Example3", CV_WINDOW_AUTOSIZE );
    g_capture = cvCreateFileCapture( argv[1] );
}
```

```

int frames = (int) cvGetCaptureProperty(
    g_capture,
    CV_CAP_PROP_FRAME_COUNT
);
if( frames!= 0 ) {
    cvCreateTrackbar(
        "Position",
        "Example3",
        &g_slider_position,
        frames,
        onTrackbarSlide
    );
}
IplImage* frame;
// While loop (as in Example 2) capture & show video
...
// Release memory and destroy window
...
return(0);
}

```

从本质上说，这种方法是通过添加一个全局变量来表示滚动条位置并且添加一个回调函数更新变量以及重新设置视频读入位置。我们通过一个调用来创建滚动条和确定回调函数^①。下面让我们看看细节。

```

int g_slider_position = 0;
CvCapture* g_capture = NULL;

```

【20~21】

首先为滚动条位置定义一个全局变量。由于回调函数需要使用 CvCapture 对象，因此我们将它定义为全局变量。为了使我们的程序可读性更强，我们在所有全局变量名称前面加上 g_。

```

void onTrackbarSlide(int pos) {
    cvSetCaptureProperty(
        g_capture,
        CV_CAP_PROP_POS_FRAMES,
        pos
    );
}

```

① 这段代码并没有实现滚动条随着视频播放自动移动；我们把它作为一个练习留给读者。另外需要注意，一些 mpeg 编码的视频是不可以后退拖动的。

现在我们定义一个回调函数，使其在滚动条被拖动时调用。滚动条的位置会被作为一个 32 位整数以参数形式传入。

后面我们会常常看到函数 `cvSetCaptureProperty()` 被调用，同时与之配套的函数 `cvGetCaptureProperty()` 也经常会被调用。这些函数允许我们设置(或查询)CvCapture 对象的各种属性。在本程序中我们设置参数 `CV_CAP_PROP_POS_FRAMES`(这个参数表示我们以帧数来设置读入位置，如果我们想通过视频长度比例来设置读入位置，我们可以通过用 `AVI_RATIO` 代替 `FRAMES` 来实现)。最后，我们把新的滚动条位置作为参数传入。因为 HighGUI 是高度智能化的，它会自动处理一些问题，比如滚动条对应位置不是关键帧，它会从前面一个关键帧开始运行并且快进到对应帧，而不需要我们来处理这些细节问题。

```
int frames = (int) cvGetCaptureProperty(  
    g_capture,  
    CV_CAP_PROP_FRAME_COUNT  
) ;
```

正如前面所说，当需要从 CvCapture 结构查询数据时，可使用 `cvGetCaptureProperty` 函数。在本程序中，我们希望获得视频文件的总帧数以对滚动条进行设置(具体实现在后面)。

```
if( frames!= 0 ) {  
    cvCreateTrackbar(  
        "Position",  
        "Example3",  
        &g_slider_position,  
        frames,  
        onTrackbarSlide  
    );  
}
```

【21】

前面的代码用来创建滚动条，借助函数 `cvCreateTrackbar()`，我们可设置滚动条的名称并确定滚动条的所属窗口。我们将一个变量绑定到这个滚动条来表示滚动条的最大值和一个回调函数(不需要回调函数时置为空，当滚动条被拖动时触发)。仔细分析，你会发现一点：`cvGetCaptureProperty()` 返回的帧数为 0 时，滚动条不会被创建。这是因为对于有些编码方式，总的帧数获取不到，在这种情况下，我们

只能直接播放视频文件而看不到滚动条^①。

值得注意的是，通过 HighGUI 创建的滚动条不像其他工具提供的滚动条功能这么全面。当然，也可以使用自己喜欢的其他窗口开发工具包来代替 HighGUI，但是 HighGUI 可以较快地实现一些基本功能。

最后，我们并没有将实现滚动条随着视频播放移动功能的代码包含进来，这可作为一个练习留给读者。

一个简单的变换

很好，现在你已经可以使用 OpenCV 创建自己的视频播放器了，但是我们所关心的是计算机视觉，所以下面会讨论一些计算机视觉方面的工作。很多基本的视觉任务包括对视频流的滤波。我们将通过修改前面程序，实现随着视频的播放而对其中每一帧进行一些简单的运算。

一个简单的变化就是对图像平滑处理，通过对图像数据与高斯或者其他核函数进行卷积有效的减少图像信息内容。OpenCV 使得这个卷积操作非常容易。我们首先创建一个窗口“Example4-out”用来显示处理后的图像。然后，在我们调用 cvShowImage() 来显示新捕捉的图像以后，我们可以计算和在输出窗口中显示平滑处理后的图像。具体如例 2-4 所示。

例 2-4：载入一幅图像并进行平滑处理

```
#include "cv.h"
#include "highgui.h"

void example2_4( IplImage* image )

    // Create some windows to show the input
    // and output images in.
    //
    cvNamedWindow( "Example4-in" );
```

① 因为 HighGUI 是一个轻巧易用的开发工具包，所以 cvCreateTrackbar() 将滚动条的名称和标签名称设为相同的。你可能已经注意到，cvNamedWindow() 还将窗口名称与界面上的窗口标签设为相同的。

```

cvNamedWindow( "Example4-out" );

// Create a window to show our input image
//
cvShowImage( "Example4-in", image );

// Create an image to hold the smoothed output
//
IplImage* out = cvCreateImage(
    cvGetSize(image),
    IPL_DEPTH_8U,
    3
);

// Do the smoothing
//
cvSmooth( image, out, CV_GAUSSIAN, 3, 3 );

// Show the smoothed image in the output window
//
cvShowImage( "Example4-out", out );

// Be tidy
//
cvReleaseImage( &out );

// Wait for the user to hit a key, then clean up the windows
//
cvWaitKey( 0 );
cvDestroyWindow( "Example4-in" );
cvDestroyWindow( "Example4-out" );

}

```

【22 ~ 23】

第一个 `cvShowImage()` 调用跟前面的例子中没有什么不同。在下面的调用，我们为另一个图像结构分配空间。前面依靠 `cvCreateFileCapture()` 来为新的帧分配空间。事实上，`cvCreateFileCapture()` 只分配一帧图像的空间，每次调用时覆盖前面一次的数据(这样每次调用返回的指针是一样的)。在这种情况下，我们想分配自己的图像结构空间用来存储平滑处理后的图像。第一个参数是一个 `CvSize` 结构，这个结构可以通过 `cvGetSize(image)` 方便地获得；第一个参数说明了当前图像结构的大小。第二个参数告诉了我们各通道每个像素点的数据类型，最后一个参数说明了通道的总数。所以从程序中可以看出，当前图像是 3 个通道(每个通道 8

位), 图像大小同 `image`。

平滑处理实际上只是对 OpenCV 库函数的一个调用: 我们指定输入图像, 输出图像, 光滑操作的方法以及平滑处理的一些参数。在本程序中, 我们通过使用每个像素周围 3×3 区域进行高斯平滑处理。事实上, 输入、输出图像可以是相同的, 这将使得我们的程序更加有效, 不过我们为了介绍 `cvCreateImage()` 函数而没有这样使用。

现在我们可以在我们新窗口中显示处理后的图像然后释放它: `cvReleaseImage()` 通过给定一个指向 `IplImage*` 的指针来释放与图像对应的内存空间。 【23~24】

一个复杂一点的变换

前面的工作很棒, 我们学会了做一些更有趣的事情。在例 2-4 中, 我们选择了重新创建一个 `IplImage` 结构, 并且在新建的结构中写入了一个单个变换的结果。正如前面所提到的, 我们可以以这样一种方式来应用某个变换, 即用输出来覆盖输入变量, 但这并非总是行得通的。具体说来, 有些操作输出的图像与输入图像相比, 大小、深度、通道数目都不一样。通常, 我们希望对一些原始图像进行一系列操作并且产生一系列变换后的图像。

在这种情况下, 介绍一些封装好的函数是很有用的, 这些函数既包含输出图像内存空间的分配, 同时也进行了一些我们感兴趣的变换。例如, 可以考虑对图像进行缩放比例为 2 的缩放处理[Rosenfeld80]。在 OpenCV 中, 我们通过函数 `cvPyrDown()` 来完成上述功能。这在很多重要的视觉算法中都很有用。我们在例 2-5 中执行这个函数。

例 2-5: 使用 `cvPyrDown()` 创建一幅宽度和高度为输入图像一半尺寸的图像

```
IplImage* doPyrDown(  
    IplImage* in,
    int filter = IPL_GAUSSIAN_5x5
) {
    // Best to make sure input image is divisible by two.
    //
    assert( in->width%2 == 0 && in->height%2 == 0 );
    IplImage* out = cvCreateImage(
```

```

        cvSize( in->width/2, in->height/2 ),
        in->depth,
        in->nChannels
    );
    cvPyrDown( in, out );
    return( out );
}

```

细心的读者会发现，我们分配新的图像空间时是从旧的图像中读取所需的信息。在 OpenCV 中，所有的重要数据结构都是以结构体的形式实现，并且以结构体指针的形式传递。OpenCV 中没有私有数据！现在让我们来看一个与前类似但已加入了 Canny 边界检测的例子[Canny86](见例 2-6)。在这个程序中，边缘检测器产生了一个与输入图像大小相同但只有一个通道的图像。

【24~25】

例 2-6：Canny 边缘检测将输出写入一个单通道(灰度级)图像

```

IplImage* doCanny(
    IplImage*    in,
    double       lowThresh,
    double       highThresh,
    double       aperture
) {
    if(in->nChannels != 1)
        return(0); //Canny only handles gray scale images

    IplImage* out = cvCreateImage(
        cvGetSize( in ),
        IPL_DEPTH_8U,
        1
    );
    cvCanny( in, out, lowThresh, highThresh, aperture );
    return( out );
}

```

前面的实现使得我们更加容易进行一些连续的变换。例如，如果我们想缩放图像两次，然后在缩放后的图像中寻找边缘，我们可以很简单的进行操作组合，具体如例 2-7 所示。

例 2-7：在一个简单的图像处理流程中进行两次缩放处理与 Canny 边缘检测

```

IplImage* img1 = doPyrDown( in, IPL_GAUSSIAN_5x5 );
IplImage* img2 = doPyrDown( img1, IPL_GAUSSIAN_5x5 );

```

```
IplImage* img3 = doCanny( img2, 10, 100, 3 );
// do whatever with 'img3'
//
...
cvReleaseImage( &img1 );
cvReleaseImage( &img2 );
cvReleaseImage( &img3 );
```

注意，对前面的函数进行嵌套调用是一个很不好的主意，因为那样，内存管理的释放将会成为一个很困难的问题。如果想偷懒儿，我们可以将下面这行代码插入我们的每个封装。

```
cvReleaseImage( &in );
```

这种“自清理”模式会很干净，但是它同样有一些缺点。比如，如果想对其中任何一个中间步骤的图像进行处理，便无法找到此图像的入口。为了解决前面的问题，代码可以简化(如例 2-8 所示)。【25】

例 2-8：通过每个独立阶段释放内存来简化例 2-7 中图像处理流程

```
IplImage* out;
out = doPyrDown( in, IPL_GAUSSIAN_5x5 );
out = doPyrDown( out, IPL_GAUSSIAN_5x5 );
out = doCanny( out, 10, 100, 3 );

// do whatever with 'out'
//
...
cvReleaseImage ( &out );
```

对于自清理方法的最后一个提醒：在 OpenCV 中，我们必须确认被释放的空间必须是我们显式分配的。参考前面从 cvQueryFrame() 返回的 IplImage* 指针。这个指针指向的结构是 CvCapture 结构的一部分，而 CvCapture 在初始时就已经被初始化了而且只初始化一次。通过调用 cvReleaseImage() 释放 IplImage* 会产生很多难以处理的问题。前面这个例子主要是想说明，虽然内存垃圾处理在 OpenCV 中很重要，但我们只需要释放自己显式分配的内存空间。

从摄像机读入数据

在计算机世界里，“视觉”这个词的含义非常丰富。在一些情况下，我们要分析从其他地方载入的固定图像。在另外一些情况下，我们要分析从磁盘中读入的视频文

件。在更多的情况下，我们想处理从某些摄像设备中实时读入的视频流。

OpenCV——更确切地说，OpenCV 中的 HighGUI 模块——为我们提供了一种简单的方式来处理这种情况。这种方法类似于读取 AVI 文件，不同的是，我们调用的是 `cvCreateCameraCapture()`，而不是 `cvCreateFileCapture()`。后面一个函数参数为摄像设备 ID 而不是文件名。当然，只有存在多个摄像设备时这个参数才起作用。默认值为 -1，代表“随机选择一个”；自然，它更适合当有且仅有一个摄像设备的情况(详细内容参考第 4 章)。

函数 `cvCreateCameraCapture()` 同样返回相同的 `CvCapture*` 指针，这使得我们后面可以使用完全类似于从视频流中获取帧的方法。当然，HighGUI 做很多工作才使得摄像机图像序列看起来像一个视频文件，但是我们不需要考虑那些问题。当我们需要处理摄像机图像序列时我们只需要简单地从摄像机获得图像，像视频文件一样处理。为了便于开发工作，大多程序实时处理的程序同样会有视频文件读入模式，`CvCapture` 结构的通用性使得这非常容易实现。具体如例 2-9 所示。【26】

例 2-9：capture 结构初始化后，从视频文件或摄像设备读入图像没有区别

```
CvCapture* capture;

if( argc==1 ) {
    capture = cvCreateCameraCapture(0);
} else {
    capture = cvCreateFileCapture( argv[1] );
}
assert( capture != NULL );

// Rest of program proceeds totally ignorant
...
```

由此可见，这种处理是很理想的。

写入 AVI 视频文件

在很多程序中，我们想将输入视频流或者捕获的图像序列记录到输出视频流中，OpenCV 提供了一个简洁的方法来实现这个功能。就像可以创建一个捕获设备以便每次都能从视频流中提取一帧一样，我们同样可以创建一个写入设备以便逐帧将视频流写入视频文件。实现这个功能的函数是 `cvCreateVideoWriter()`。

当输出设备被创建以后，我们可以通过调用 `cvWriteFrame()` 逐帧将视频流写入文

件。写入结束后，我们调用 `cvReleaseVideoWriter()` 来释放资源。例 2-10 描述了一个简单的程序。这个程序首先打开一个视频文件，读取文件内容，将每一帧图像转换为对数极坐标格式(就像你的眼睛真正看到的，在第 6 章会有详细描述)，最后将转化后的图像序列写入新的视频文件中。

例 2-10：一个完整的程序用来实现读入一个彩色视频文件并以灰度格式输出这个视频文件

```
// Convert a video to grayscale
// argv[1]: input video file
// argv[2]: name of new output file
//
#include "cv.h"
#include "highgui.h"
main( int argc, char* argv[] ) {
    CvCapture* capture = 0;
    capture = cvCreateFileCapture( argv[1] );
    if(!capture){
        return -1;
    }
    IplImage *bgr_frame=cvQueryFrame(capture); //Init the video read
    double fps = cvGetCaptureProperty (
        capture,
        CV_CAP_PROP_FPS
    );
    CvSize size = cvSize(
        (int)cvGetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH),
        (int)cvGetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT)
    );
    CvVideoWriter *writer = cvCreateVideoWriter(
        argv[2],
        CV_FOURCC('M','J','P','G'),
        fps,
        size
    );
    IplImage* logpolar_frame = cvCreateImage(
        size,
        IPL_DEPTH_8U,
        3
    );
    while( (bgr_frame=cvQueryFrame(capture)) != NULL ) {
        cvLogPolar( bgr_frame, logpolar_frame,
                    cvPoint2D32f(bgr_frame->width/2,
```

```

        bgr_frame->height/2),
        40,
        CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS );
cvWriteFrame( writer, logpolar_frame );
}
cvReleaseVideoWriter( &writer );
cvReleaseImage( &logpolar_frame );
cvReleaseCapture( &capture );
return(0);
}

```

【27~28】

仔细阅读这个程序，会发现它使用了一些我们很熟悉的函数。首先，打开一个视频文件；通过 `cvQueryFrame()` 函数读入视频；然后，我们使用 `cvGetCaptureProperty()` 来获得视频流的各种重要属性。打开一个视频文件进行此操作，并将各帧图像转换为对数极坐标格式，将转换后的图像逐帧写入视频文件，直到读入结束。最后释放各种资源，程序结束。

在对函数 `cvCreateVideoWriter()` 进行调用时，有几个参数是我们必须了解的。第一个参数是用来指定新建视频文件的名称。第二个参数是视频压缩的编码格式。目前有很多流行的编解码器格式，但是无论采用哪种格式，都必须确保自己的电脑中有这种编解码器(编解码器的安装独立于 OpenCV)。在本程序中，我们选用比较流行的 MJPG 编码格式；OpenCV 用宏 `CV_FOURCC()` 来指定编码格式，`CV_FOURCC()` 包含 4 个字符参数。这 4 个字符构成了编解码器的“4 字标记”，每个编解码器都有一个这样的标记，Motion JPEG 编码格式的四字标记就是 MJPG，所以如此指定宏的 4 个字符参数 `CV_FOURCC('M','J','P','G')`。后面两个参数用来指定播放的帧率和视频图像的大小。在本程序中，我们将它们设置成原始视频文件(彩色视频文件)的帧率和图像的大小。

小结

在开始学习第 3 章前，我们需要先总结一下前面的知识，并展望后面要讲的东西。我们已经知道，OpenCV 为程序开发提供了很多简便易用的接口，可以用来载入图像、从磁盘或者从摄像设备中捕获视频。我们也知道，OpenCV 库中包含很多基本的函数用来处理这些图像。但是更多更强大的功能我们还没有看到。它可以对抽象数据类型进行更多更复杂的处理，这些对解决实际的视觉问题有很大的帮助。

在后面的几章里，我们会更深入地了解 OpenCV 的一些基础结构并对界面相关函数和图像数据类型做更详细的分析。我们会系统地讲解一些基本的图像处理操作，

并在后面讲解一些高级的图像变换。最后，我们会探索更多专业的应用，如摄像机标定、跟踪、识别。OpenCV 为这些应用提供了许多 API。准备好了吗？让我们开始吧！

【29】

练习

如果还没有在本机上安装 OpenCV，请先下载并安装。浏览目录，在 docs 目录中，可以找到 *index.htm* 文件，它链接到库文件的一些主要文档。下面我们进一步了解一些库文件，*Cvcore* 包含一些基本数据结构与算法，*cv* 包含图像处理和视觉的一些算法，*ml* 由一些机器学习和聚类算法组成，*otherlibs/highgui* 包含一些输入/输出函数。*_make* 文件夹下包含 OpenCV 的工程文件，程序实例的代码在 *samples* 文件夹里。

1. 进入目录 .../opencv/_make。在 Windows 环境下，打开解决方案 *opencv.sln*，在 Linux 环境下，打开对应的 *makefile*，分别建立 *debug* 和 *release* 版的库文件。这会花一些时间，但是后面会用到这些库文件和动态链接库。
2. 进入目录 .../opencv/samples/c/。建立一个工程 (project)，导入并编译 *lkdemo.c*(这是一个运动跟踪程序实例)。安装上摄像机以后运行程序，选中运行窗口，键入“r”进行跟踪初始化，也可以在视频位置上单击鼠标以添加跟踪点。还可以通过键入“n”切换为只观察点(而非图像)。再次键入“n”可在“夜间”和“白天”这两个视图之间进行切换。
3. 使用例 2-10 中的视频捕捉和存储方法，结合例 2-5 中的 *doPyrDown()* 创建一个程序，使其从摄像机读入视频数据并将缩放变换后的彩色图像存入磁盘。
4. 修改练习 3 的代码，结合例 2-1，将变换结果显示在窗口中。
5. 对练习 4 中的代码进行修改，参考例 2-3，给程序加入滚动条，使得用户可以动态调节缩放比例，缩放比例的取值为 2~8 之间。可以跳过写入磁盘操作，但是必须将变换结果显示在窗口中。

初探 OpenCV



OpenCV 的基本数据类型

OpenCV 提供了多种基本数据类型。虽然这些数据类型在 C 语言中不是基本类型，但结构都很简单，可将它们作为原子类型。可以在“…/OpenCV/cxcore/include”目录下的 `cxtypes.h` 文件中查看其详细定义。

在这些数据类型中最简单的就是 `CvPoint`。`CvPoint` 是一个包含 `integer` 类型成员 `x` 和 `y` 的简单结构体。`CvPoint` 有两个变体类型：`CvPoint2D32f` 和 `CvPoint3D32f`。前者同样有两个成员 `x`, `y`，但它们是浮点类型；而后者却多了一个浮点类型的成员 `z`。

`CvSize` 类型与 `CvPoint` 非常相似，但它的数据成员是 `integer` 类型的 `width` 和 `height`。如果希望使用浮点类型，则选用 `CvSize` 的变体类型 `CvSize2D32f`。

`CvRect` 类型派生于 `CvPoint` 和 `CvSize`，它包含 4 个数据成员：`x`, `y`, `width` 和 `height`。(正如你所想的那样，该类型是一个复合类型)。

下一个(但不是最后一个)是包含 4 个整型成员的 `CvScalar` 类型，当内存不是问题时，`CvScalar` 经常用来代替 1, 2 或者 3 个实数成员(在这个情况下，不需要的分量被忽略)。`CvScalar` 有一个单独的成员 `val`，它是一个指向 4 个双精度浮点数数组的指针。

所有这些数据类型具有以其名称来定义的构造函数，例如 `cvSize()`。(构造函数通

常^①具有与结构类型一样的名称，只是首字母不大写）。记住，这是 C 而不是 C++，所以这些构造函数只是内联函数，它们首先提取参数列表，然后返回被赋予相关值的结构。

【31】

各数据类型的内联构造函数被列在表 3-1 中：`cvPointXXX()`，`cvSize()`，`cvRect()`和`cvScalar()`。这些结构都十分有用，因为它们不仅使代码更容易编写，而且也更易于阅读。假设要在(5, 10)和(20, 30)之间画一个白色矩形，只需简单调用：

```
cvRectangle(  
    myImg,  
    cvPoint(5,10),  
    cvPoint(20,30),  
    cvScalar(255,255,255)  
) ;
```

表 3-1：points, size, rectangles 和 scalar 三元组的结构

结构	成员	说明
CvPoint	int x, y	图像中的点
CvPoint2D32f	float x, y	二维空间中的点
CvPoint3D32f	float x, y, z	三维空间中的点
CvSize	int width, height	图像的尺寸
CvRect	int x, y, width, height	图像的部分区域
CvScalar	double val[4]	RGBA 值

`cvScalar` 是一个特殊的例子：它有 3 个构造函数。第一个是`cvScalar()`，它需要一个、两个、三个或者四个参数并将这些参数传递给数组`val[]`中的相应元素。第二个构造函数是`cvRealScalar()`，它需要一个参数，它被传递给给`val[0]`，而`val[]`数组别的值被赋为 0。最后一个有所变化的是`cvScalarAll()`，它需要一个参数并且`val[]`中的 4 个元素都会设置为这个参数。

① 我们之所以说“通常”是因为这里有一些特殊情况。我们有`cvScalarAll(double)`和`cvRealScalar(double)`两个构造函数，前者将返回一个向量，该向量的 4 个成员变量全部初始化为同一参数值；而后者所返回的向量中，仅第一个成员变量初始化为参数值，其余变量则都初始化为 0。

矩阵和图像类型

图 3-1 为我们展示了三种图像的类或结构层次结构。使用 OpenCV 时，会频繁遇到 IplImage 数据类型，第 2 章已经出现多次。IplImage 是我们用来为通常所说的“图像”进行编码的基本结构。这些图像可能是灰度，彩色，4 通道的 (RGB+alpha)，其中每个通道可以包含任意的整数或浮点数。因此，该类型比常见的、易于理解的 3 通道 8 位 RGB 图像更通用^①。

OpenCV 提供了大量实用的图像操作符，包括缩放图像，单通道提取，找出特定通道最大最小值，两个图像求和，对图像进行阈值操作，等等。本章我们将详细介绍这类操作。

【32】

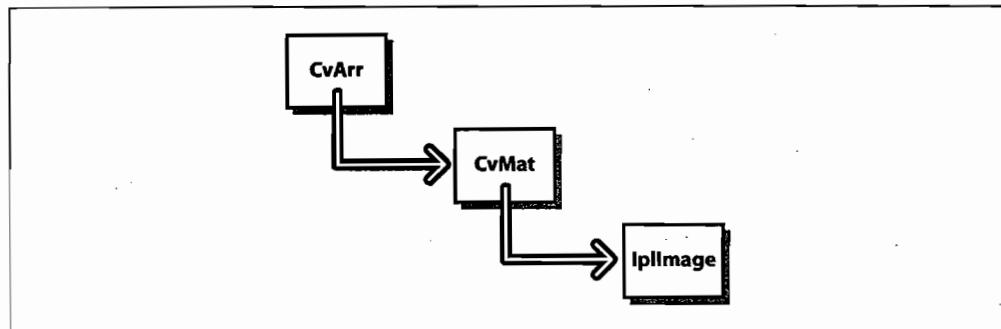


图 3-1：虽然 OpenCV 是由 C 语言实现的，但它使用的结构体也是遵循面向对象的思想设计的。实际上，IplImage 由 CvMat 派生，而 CvMat 由 CvArr 派生

在开始探讨图像细节之前，我们需要先了解另一种数据类型 CvMat，OpenCV 的矩阵结构。虽然 OpenCV 完全由 C 语言实现，但 CvMat 和 IplImage 之间的关系就如同 C++ 中的继承关系。实质上，IplImage 可以被视为从 CvMat 中派生的。因此，在试图了解复杂的派生类之前，最好先了解基本的类。第三个类 CvArr，可以被视为一个抽象基类，CvMat 由它派生。在函数原型中，会经常看到 CvArr(更准确地说，CvArr*)，当它出现时，便可以将 CvMat* 或 IplImage* 传递到程序。

① 若从严谨的角度来讲，你可以认为 OpenCV 虽然是基于 C 语言实现的，但其设计思想却融入了面向对象和面向模板的设计方法。

CvMat 矩阵结构

在开始学习矩阵的相关内容之前，我们需要知道两件事情。第一，在 OpenCV 中没有向量(vector)结构。任何时候需要向量，都只需要一个列矩阵(如果需要一个转置或者共轭向量，则需要一个行矩阵)。第二，OpenCV 矩阵的概念与我们在线性代数课上学习的概念相比，更抽象，尤其是矩阵的元素，并非只能取简单的数值类型。例如，一个用于新建一个二维矩阵的例程具有以下原型：

```
cvMat* cvCreateMat ( int rows, int cols, int type );
```

这里 type 可以是任何预定义类型，预定义类型的结构如下：`CV_<bit_depth>(S|U|F)C<number_of_channels>`。于是，矩阵的元素可以是 32 位浮点型数据(CV_32FC1)，或者是无符号的 8 位三元组的整型数据(CV_8UC3)，或者是无数的其他类型的元素。一个 CvMat 的元素不一定就是个单一的数字。在矩阵中可以通过单一(简单)的输入来表示多值，这样我们可以在一个三原色图像上描绘多重色彩通道。对于一个包含 RGB 通道的简单图像，大多数的图像操作将分别应用于每一个通道(除非另有说明)。

实质上，正如例 3-1 所示，CvMat 的结构相当简单，(可以自己打开文件…/opencv/cxcore/include/cxtypes.h 查看)。矩阵由宽度(width)、高度(height)、类型(type)、行数据长度(step，行的长度用字节表示而不是整型或者浮点型长度)和一个指向数据的指针构成(现在我们还不能讨论更多的东西)。可以通过一个指向 CvMat 的指针访问这些成员，或者对于一些普通元素，使用现成的访问方法。例如，为了获得矩阵的大小，可通过调用函数 `vGetSize(CvMat*)`，返回一个 CvSize 结构，便可以获取任何所需信息，或者通过独立访问高度和宽度，结构为 `matrix->height` 和 `matrix->width`。
【33~34】

例 3-1：CvMat 结构：矩阵头

```
typedef struct CvMat {
    int type;
    int step;
    int* refcount; // for internal use only
    union {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
```

```

        double* db;
    } data;
union {
    int rows;
    int height;
};
union {
    int cols;
    int width;
};
} CvMat;

```

此类信息通常被称作矩阵头。很多程序是区分矩阵头和数据体的，后者是各个 data 成员所指向的内存位置。

矩阵有多种创建方法。最常见的方法是用 cvCreateMat()，它由多个原函数组成，如 cvCreateMatHeader() 和 cvCreateData()。cvCreateMatHeader() 函数创建 CvMat 结构，不为数据分配内存，而 cvCreateData() 函数只负责数据的内存分配。有时，只需要函数 cvCreateMatHeader()，因为已因其他理由分配了存储空间，或因为还不准备分配存储空间。第三种方法是用函数 cvCloneMat(CvMat*)^①，它依据一个现有矩阵创建一个新的矩阵。当这个矩阵不再需要时，可以调用函数 cvReleaseMat(CvMat*) 释放它。

【34】

例 3-2 概述了这些函数及其密切相关的其他函数。

例 3-2：矩阵的创建和释放

```

//Create a new rows by cols matrix of type 'type'.
//
CvMat* cvCreateMat( int rows, int cols, int type );

//Create only matrix header without allocating data
//
CvMat* cvCreateMatHeader( int rows, int cols, int type );

//Initialize header on existiong CvMat structure
//
CvMat* cvInitMatHeader(

```

① 函数 cvCloneMat() 和其他的 OpenCV 包含单词 “clone”的函数，不仅创建一个和输入头同样的头，也分配各自的数据区并将源数据复制到新对象。

```

CvMat* mat,
int rows,
int cols,
int type,
void* data = NULL,
int step = CV_AUTOSTEP
);

//Like cvInitMatHeader() but allocates CvMat as well.
//
CvMat cvMat(
    int rows,
    int cols,
    int type,
    void* data = NULL
);

//Allocate a new matrix just like the matrix 'mat'.
//
CvMat* cvCloneMat( const cvMat* mat );
// Free the matrix 'mat', both header and data.
//
void cvReleaseMat( CvMat** mat );

```

与很多 OpenCV 结构类似，有一种构造函数叫 cvMat，它可以创建 CvMat 结构，但实际上不分配存储空间，仅创建头结构(与 cvInitMatHeader() 类似)。这些方法对于存取到处散放的数据很有作用，可以将矩阵头指向这些数据，实现对这些数据的打包，并用操作矩阵的函数去处理这些数据，如例 3-3 所示。

例 3-3：用固定数据创建一个 OpenCV 矩阵

```

//Create an OpenCV Matrix containing some fixed data.
//
float vals[] = { 0.866025, -0.500000, 0.500000, 0.866025 };

CvMat rotmat;

cvInitMatHeader(
    &rotmat,
    2,
    2,
    CV_32FC1,

```

```
    vals  
);
```

一旦我们创建了一个矩阵，便可用它来完成很多事情。最简单的操作就是查询数组定义和数据访问等。为查询矩阵，我们可以使用函数 `cvGetElemType(const CvArr* arr)`, `cvGetDims(const CvArr* arr, int* sizes=NULL)` 和 `cvGetDimSize(const CvArr* arr, int index)`。第一个返回一个整型常数，表示存储在数组里的元素类型(它可以为 `CV_8UC1` 和 `CV_64FC4` 等类型)。第二个取出数组以及一个可选择的整型指针，它返回维数(我们当前的实例是二维，但是在后面我们将遇到的 N 维矩阵对象)。如果整型指针不为空，它将存储对应数组的高度和宽度(或者 N 维数)。最后的函数通过一个指示维数的整型数简单地返回矩阵在那个维数上矩阵的大小。^①

【35~36】

矩阵数据的存取

访问矩阵中的数据有 3 种方法：简单的方法、麻烦的方法和恰当的方法。

简单的方法

从矩阵中得到一个元素的最简单的方法是利用宏 `CV_MAT_ELEM()`。这个宏(参见例 3-4)传入矩阵、待提取的元素的类型、行和列数 4 个参数，返回提取出的元素的值。

例 3-4：利用 `CV_MAT_ELEM()` 宏存取矩阵

```
CvMat* mat = cvCreateMat( 5, 5, CV_32FC1 );  
float element_3_2 = CV_MAT_ELEM( *mat, float, 3, 2 );
```

更进一步，还有一个与此宏类似的宏，叫 `CV_MAT_ELEM_PTR()`。`CV_MAT_ELEM_PTR()`(参见例 3-5)传入矩阵、待返回元素的行和列号这 3 个参数，返回指向这个元素的指针。该宏和 `CV_MAT_ELEM()` 宏的最重要的区别是后者在指针解引用之前将其转化成指定的类型。如果需要同时读取数据和设置数据，可以直接调用 `CV_MAT_ELEM_PTR()`。但在这种情况下，必须自己将指针转化成恰当的类型。

① 这里讨论的是二维矩阵，所以第 0 维，通常表示宽度；第 1 维，通常表示高度。

例 3-5：利用宏 CV_MAT_ELEM_PTR()为矩阵设置一个数值

```
CvMat* mat = cvCreateMat( 5, 5, CV_32FC1 );
float element_3_2 = 7.7;
*( (float*)CV_MAT_ELEM_PTR( *mat, 3, 2 ) ) = element_3_2;
```

【36】

遗憾的是，这些宏在每次调用的时候都重新计算指针。这意味着要查找指向矩阵基本元素数据区的指针、计算目标数据在矩阵中的相对地址，然后将相对位置与基本位置相加。所以，即使这些宏容易使用，但也不是存取矩阵的最佳方法。在计划顺序访问矩阵中的所有元素时，这种方法的缺点尤为突出。下面我们将讲述怎么运用最好的方法完成这个重要任务。

麻烦的方法

在“简单的方法”中讨论的两个宏仅仅适用于访问 1 维或 2 维的数组(回忆一下，1 维的数组，或者称为“向量”实际只是一个 $n \times 1$ 维矩阵)。OpenCV 提供了处理多维数组的机制。事实上，OpenCV 可以支持普通的 N 维的数组，这个 N 值可以取值为任意大的数。

为了访问普通矩阵中的数据，我们可以利用在例 3-6 和例 3-7 中列举的 cvPtr*D 和 cvGet*D... 等函数族。cvPtr*D 家族包括 cvPtr1D(), cvPtr2D(), cvPtr3D() 和 cvPtrND()...。这三个函数都可接收 CvArr*类型的矩阵指针参数，紧随其后的参数是表示索引的整数值，最后是一个可选的参数，它表示输出值的类型。函数返回一个指向所需元素的指针。对于 cvPtrND()来说，第二个参数是一个指向一个整型数组的指针，这个数组中包含索引的合适数字。后文会再次介绍此函数(在这之后的原型中，也会看到一些可选参数，必要时会有讲解)。

例 3-6：指针访问矩阵结构

```
uchar* cvPtr1D(
    const CvArr* arr,
    int           idx0,
    int*          type = NULL
);

uchar* cvPtr2D(
    const CvArr* arr,
    int           idx0,
```

```

int          idx1,
int*        type = NULL
);

uchar* cvPtr3D(
    const CvArr* arr,
    int          idx0,
    int          idx1,
    int          idx2,
    int*        type = NULL
);
uchar* cvPtrND(
    const CvArr* arr,
    int*        idx,
    int*        type      = NULL,
    int          create_node = 1,
    unsigned*   precalc_hashval = NULL
);

```

【37~38】

如果仅仅是读取数据，可用另一个函数族 `cvGet*D`。如例 3-7 所示，该例与例 3-6 类似，但是返回矩阵元素的实际值。

例 3-7：CvMat 和 IplImage 元素函数

```

double cvGetReal1D( const CvArr* arr, int idx0 );
double cvGetReal2D( const CvArr* arr, int idx0, int idx1 );
double cvGetReal3D( const CvArr* arr, int idx0, int idx1, int idx2 );
double cvGetRealND( const CvArr* arr, int* idx );

CvScalar cvGet1D( const CvArr* arr, int idx0 );
CvScalar cvGet2D( const CvArr* arr, int idx0, int idx1 );
CvScalar cvGet3D( const CvArr* arr, int idx0, int idx1, int idx2 );
CvScalar cvGetND( const CvArr* arr, int* idx );

```

`cvGet*D` 中有四个函数返回的是整型的，另外四个的返回值是 `CvScalar` 类型的。这意味着在使用这些函数的时候，会有很大的空间浪费。所以，只是在你认为用这些函数比较方便和高效率的时候才用它们，否则，最好用 `cvPtr*D`。

用 `cvPtr*D()` 函数族还有另外一个原因，即可以用这些指针函数访问矩阵中的特定的点，然后由这个点出发，用指针的算术运算得到指向矩阵中的其他数据的指

针。在多通道的矩阵中，务必记住一点：通道是连续的，例如，在一个 3 通道 2 维的表示红、绿、蓝(RGB)矩阵中。矩阵数据如下存储 `rgbrgbrgb . . .`。所以，要将指向该数据类型的指针移动到下一通道，我们只需要将其增加 1。如果想访问下一个“像素”或者元素集，我们只需要增加一定的偏移量，使其与通道数相等。

另一个需要知道的技巧是矩阵数组的 `step` 元素(参见例 3-1 和例 3-3)，`step` 是矩阵中行的长度，单位为字节。在那些结构中，仅靠 `cols` 或 `width` 是无法在矩阵的不同行之间移动指针的，出于效率的考虑，矩阵或图像的内存分配都是 4 字节的整数倍。所以，三个字节宽度的矩阵将被分配 4 个字节，最后一个字节被忽略。因此，如果我们得到一个字节指针，该指针指向数据元素，那么我们可以用 `step` 和这个指针相加以使指针指向正好在我们的点的下一行元素。如果我们有一个整型或者浮点型的矩阵，对应的有整型和浮点型的指针指向数据区域，我们将让 `step/4` 与指针相加来移到下一行，对双精度型的，我们让 `step/8` 与指针相加(这里仅仅考虑了 C 将自动地将差值与我们添加的数据类型的字节数相乘)。

【38】

例 3-8 中的 `cvSet*D` 和 `cvGet*D` 多少有些相似，它通过一次函数调用为一个矩阵或图像中的元素设置值，函数 `cvSetReal*D()` 和函数 `cvSet*D()` 可以用来设置矩阵或者图像中元素的数值。

例 3-8：为 CvMat 或者 IplImage 元素设定值的函数

```
void cvSetReal1D( CvArr* arr, int idx0, double value );
void cvSetReal2D( CvArr* arr, int idx0, int idx1, double value );
void cvSetReal3D(
    CvArr* arr,
    int idx0,
    int idx1,
    int idx2,
    double value
);
void cvSetRealND( CvArr* arr, int* idx, double value );

void cvSet1D( CvArr* arr, int idx0, CvScalar value );
void cvSet2D( CvArr* arr, int idx0, int idx1, CvScalar value );
void cvSet3D(
    CvArr* arr,
    int idx0,
```

```
int idx1,  
int idx2,  
CvScalar value  
);  
void cvSetND( CvArr* arr, int* idx, CvScalar value );
```

为了方便，我们也可以使用 `cvmSet()` 和 `cvmGet()`，这两个函数用于处理浮点型单通道矩阵，非常简单。

```
double cvmGet( const CvMat* mat, int row, int col )  
void cvmSet( CvMat* mat, int row, int col, double value )
```

以下函数调用 `cvmSet()`：

```
cvmSet( mat, 2, 2, 0.5000 );
```

等同于 `cvSetReal2D` 函数调用：

```
cvSetReal2D( mat, 2, 2, 0.5000 );
```

恰当的方法

从以上所有那些访问函数来看，你可能会想，没有必要再介绍了。实际上，这些 `set` 和 `get` 函数很少派上用场。大多数时候，计算机视觉是一种运算密集型的任务，因而你想尽量利用最有效的方法做事。毋庸置疑，通过这些函数接口是不可能做到十分高效的。相反地，应该定义自己的指针计算并且在矩阵中利用自己的方法。如果打算对数组中的每一个元素执行一些操作，使用自己的指针是尤为重要的（假设没有可以为你执行任务的 OpenCV 函数）。

要想直接访问矩阵，其实只需要知道一点，即数据是按光栅扫描顺序存储的，列（“x”）是变化最快的变量。通道是互相交错的，这意味着，对于一个多通道矩阵来说，它们变化的速度仍然比较快。例 3-9 显示了这一过程。【39~40】

例 3-9：累加一个三通道矩阵中的所有元素

```
float sum( const CvMat* mat ) {  
  
    float s = 0.0f;  
    for( int row=0; row<mat->rows; row++ ) {  
        const float* ptr=(const float*)(mat->data.ptr + row * mat->step);  
        for( col=0; col<mat->cols; col++ ) {  
            s += *ptr++;  
        }  
    }  
}
```

```
    }
}
return( s );
}
```

计算指向矩阵的指针时，记住一点：矩阵的元素 `data` 是一个联合体。所以，对这个指针解引用的时候，必须指明结构体中的正确的元素以便得到正确的指针类型。然后，为了使指针产生正确的偏移，必须用矩阵的行数据长度(`step`)元素。我们以前曾提过，行数据元素的是用字节来计算的。为了安全，指针最好用字节计算，然后分配恰当的类型，如浮点型。`CvMat` 结构中为了兼容 `IplImage` 结构，有宽度和高度的概念，这个概念已经被最新的行和列取代。最后要注意，我们为每行都重新计算了 `ptr`，而不是简单地从开头开始，尔后每次读的时候累加指针。这看起来好像很繁琐，但是因为 `CvMat` 数据指针可以指向一个大型数组中的 ROI，所以无法保证数据会逐行连续存取。

点的数组

有一个经常提到但又必须理解的问题是，包含多维对象的多维数组(或矩阵)和包含一维对象的高维数组之间的不同。例如，假设有 n 个三维的点，你想将这些点传递到参数类型为 `CvMat*` 的一些 OpenCV 函数中。对此，有四种显而易见的方式，记住，这些方法不一定等价。一是用一个二维数组，数组的类型是 `CV32FC1`，有 n 行，3 列($n \times 3$)。类似地，也可以用一个 3 行 n 列($3 \times n$)的二维数组。也可以用一个 n 行 1 列($n \times 1$)的数组或者 1 行 n 列($1 \times n$)的数组，数组的类型是 `CV32FC3`。这些例子中，有些可以自由转换(这意味着只需传递一个，另一个便可可以计算得到)，有的则不能。要想理解原因，可以参考图 3-2 中的内存布局情况。

从图中可以看出，在前三种方式中，点集以同样的方式被映射到内存。但最后一种方式则不同。对 N 维数组的 c 维点，情况变得更为复杂。需要记住的最关键的一点是，给定点的位置可以由以下公式计算出来。

$$\delta = (\text{row}) \cdot N_{\text{cols}} \cdot N_{\text{channels}} + (\text{col}) \cdot N_{\text{channels}} + (\text{channel})$$

n-by-1 (32FC3)							1-by-n (32FC3)						
points	x_0	y_0	z_0	x_1	y_1	z_1	x_0	y_0	z_0	x_1	y_1	z_1	
($x_0 \ y_0 \ z_0$)	x_2	y_2	z_2	x_3	y_3	z_3	x_2	y_2	z_2	x_3	y_3	z_3	
($x_1 \ y_1 \ z_1$)	x_4	y_4	z_4	x_5	y_5	z_5	x_4	y_4	z_4	x_5	y_5	z_5	
($x_2 \ y_2 \ z_2$)	x_6	y_6	z_6	x_7	y_7	z_7	x_6	y_6	z_6	x_7	y_7	z_7	
($x_3 \ y_3 \ z_3$)	x_8	y_8	z_8	x_9	y_9	z_9	x_8	y_8	z_8	x_9	y_9	z_9	
($x_4 \ y_4 \ z_4$)													
($x_5 \ y_5 \ z_5$)													
n-by-3 (32FC1)	x_0	y_0	z_0	x_1	y_1	z_1	x_0	x_1	x_2	x_3	x_4	x_5	
($x_6 \ y_6 \ z_6$)	x_2	y_2	z_2	x_3	y_3	z_3	x_6	x_7	x_8	x_9	y_0	y_1	
($x_7 \ y_7 \ z_7$)	x_4	y_4	z_4	x_5	y_5	z_5	y_2	y_4	y_4	y_5	y_6	y_7	
($x_8 \ y_8 \ z_8$)	x_6	y_6	z_6	x_7	y_7	z_7	y_8	y_9	z_0	z_1	z_2	z_3	
($x_9 \ y_9 \ z_9$)	x_8	y_8	z_8	x_9	y_9	z_9	z_4	z_5	z_6	z_7	z_8	z_9	

图 3-2：有 10 个点，每个点由 3 个浮点数表示，这 10 个点被放在 4 个结构稍有不同的数组中。前 3 种情况下，内存布局情况是相同的，但最后一种情况下，内存布局不同

其中， N_{cols} 和 $N_{channels}$ 分别表示列数和通道数^①。总的来说，从这个公式中可以看出一点，一个 c 维对象的 N 维数组和一个一维对象的 $(N+c)$ 维数组不同。至于 $N=1$ （即把向量描绘成 $n \times 1$ 或者 $1 \times n$ 数组），有一个特殊之处（即图 3-2 显示的等值）值得注意，如果考虑到性能，可以在有些情况下用到它。

关于 OpenCV 的数据类型，如 CvPoint2D 和 CvPoint2D32f，我们要说明的最后一点是：这些数据类型被定义为 C 结构，因此有严格定义的内存布局。具体说来，由整型或者浮点型组成的结构是顺序型的通道。那么，对于一维的 C 语言的对象数组来说，其数组元素都具有相同的内存布局，形如 CV32FC2 的 $n \times 1$ 或者 $1 \times n$ 数组。这和申请 CvPoint3D32f 类型的数组结构也是相同的。【41】

① 在前面的公式中，我们利用术语“通道”来表示运行最快的索引。这个索引与 CV32FC3 中的 C3 部分有关。简而言之，当我们讨论图像时，“通道”将完全等同于这里所用的“通道”。

IplImage 数据结构

掌握了前面的知识，再来讨论 IplImage 数据结构就比较容易了。从本质上讲，它是一个 CvMat 对象，但它还有其他一些成员变量将矩阵解释为图像。这个结构最初被定义为 Intel 图像处理库(IPL)^①的一部分。IplImage 结构的准确定义如例 3-10 所示。

例 3-10：IplImage 结构

```
typedef struct _IplImage {
    int             nSize;
    int             ID;
    int             nChannels;
    int             alphaChannel;
    int             depth;
    char            colorModel[4];
    char            channelSeq[4];
    int             dataOrder;
    int             origin;
    int             align;
    int             width;
    int             height;
    struct _IplROI* roi;
    struct _IplImage* maskROI;
    void*           imageId;
    struct _IplTileInfo* tileInfo;
    int             imageSize;
    char*           imageData;
    int             widthStep;
    int             BorderMode[4];
    int             BorderConst[4];
    char*           imageDataOrigin;
} IplImage;
```

① IPL 是现在的英特尔高性能多媒体函数库(IPP)的前身，在第 1 章已经讨论过。OpenCV 的很多函数实际上可调用 IPL 或 IPP 中的相同功能函数。因此如果已安装了高性能的 IPP 库，OpenCV 便可以轻松地自动调用 IPP 函数。

我们试图讨论这些变量的某些功能。有些变量不是很重要，但是有些变量非常重要，有助于我们理解 OpenCV 解释和处理图像的方式。

`width` 和 `height` 这两个变量很重要，其次是 `depth` 和 `nchannels`。`depth` 变量的值取自 `ipl.h` 中定义的一组数据，但与在矩阵中看到的对应变量不同。因为在图像中，我们往往将深度和通道数分开处理，而在矩阵中，我们往往同时表示它们。可用的深度值如表 3-2 所示。

【42】

表 3-2：OpenCV 图像类型

宏	图像像素类型
IPL_DEPTH_8U	无符号 8 位整数 (8u)
IPL_DEPTH_8S	有符号 8 位整数(8s)
IPL_DEPTH_16S	有符号 16 位整数(16s)
IPL_DEPTH_32S	有符号 32 位整数(32s)
IPL_DEPTH_32F	32 位浮点数单精度(32f)
IPL_DEPTH_64F	64 位浮点数双精度(64f)

通道数 `nChannels` 可取的值是 1, 2, 3 或 4。

随后两个重要成员是 `origin` 和 `dataOrder`。`origin` 变量可以有两种取值：`IPL_ORIGIN_TL` 或者 `IPL_ORIGIN_BL`，分别设置坐标原点的位置于图像的左上角或者左下角。在计算机视觉领域，一个重要的错误来源就是原点位置的定义不统一。具体而言，图像的来源、操作系统、编解码器和存储格式等因素都可以影响图像坐标原点的选取。举例来说，你或许认为自己正在从图像上面的脸部附近取样，但实际上却在图像下方的裙子附近取样。避免此类现象发生的最好办法是在最开始的时候检查一下系统，在所操作的图像块的地方画点东西试试。

`dataOrder` 的取值可以是 `IPL_DATA_ORDER_PIXEL` 或 `IPL_DATA_ORDER_PLANE`^①，前者指明数据是将像素点不同通道的值交错排在一起(这是常用的交错排列方式)，后者是把所有像素同通道值排在一起，形成通道平面，再把平面排列起来。

① 我们说 `dataOrder` 的取值可以是 `IPL_DATA_ORDER_PIXEL` 或 `IPL_DATA_ORDER_PLANE`，但实际上 OpenCV 只支持 `IPL_DATA_ORDER_PIXEL`。这两个值都受 IPL/IPP 支持，但是 OpenCV 通常使用交错排列的图像。

参数 `widthStep` 与前面讨论过的 `CvMat` 中的 `step` 参数类似，包括相邻行的同列点之间的字节数。仅凭变量 `width` 是不能计算这个值的，因为为了处理过程更高效每行都会用固定的字节数来对齐，因此在第 i 行末和第 $i+1$ 行开始处可能会有些冗于字节。参数 `imageData` 包含一个指向第一行图像数据的指针。如果图像中有些独立的平面(如当 `dataOrder = IPL_DATA_ORDER_PLANE`)那么把它们作为单独的图像连续摆放，总行数为 `height` 和 `nChannels` 的乘积。但通常情况下，它们是交错的，使得行数等于高度，而且每一行都有序地包含交错的通道。

最后还有一个实用的重要参数——感兴趣的区域(ROI)，实际上它是另一个 IPL/IPP 结构 `IplROI` 的实例。`IplROI` 包含 `xOffset`, `yOffset`, `height`, `width` 和 `coi` 成员变量，其中 COI 代表 `channel of interest`(感兴趣的通道)。ROI 的思想是：^①一旦设定 ROI，通常作用于整幅图像的函数便会只对 ROI 所表示的子图像进行操作。如果 `IplImage` 变量中设置了 ROI，则所有的 OpenCV 函数就会使用该 ROI 变量。如果 COI 被设置成非 0 值，则对该图像的操作就只作用于被指定的通道上了^②。不幸的是，许多 OpenCV 函数都忽略参数 COI。

访问图像数据

通常，我们需要非常迅速和高效地访问图像中的数据。这意味着我们不应受制于存取函数(如 `cvSet*D` 之类)。实际上，我们想要用最直接的方式访问图像内的数据。现在，应用已掌握的 `IplImage` 内部结构的知识，我们知道怎样做才是最佳的方法。

虽然 OpenCV 中有很多优化函数帮助我们完成大多数的图像处理的任务，但是还有一些任务，库中没有预先包装好的函数可以帮我们解决。例如，如果我们有一个三通道 HSV 图像[Smith78]^③，在色度保持不变的情况下，我们要设置每个点的饱和度和亮度为 255(8 位图像的最大值)，我们可以使用指针遍历图像，类似于例 3-9 中的矩阵遍历。然而，有一些细微的不同，是源于 `IplImage` 和 `CvMat` 结构的差异。例 3-11 演示了最高效的方法。

-
- ① 与 ROI 不同，并不是所有的 OpenCV 函数都使用 COI，现在应该记住，COI 并不像 ROI 那样在很多函数中可用。
 - ② COI，这个变量可取的值是 1, 2, 3, 4 通道，并且为了使 COI 无效而保留了 0 取值(有点像“忽略”)。
 - ③ 在 OpenCV 中，除了对通道这个术语解释的不同外，HSV 图像与 RGB 没有什么区别。所以，从 RGB 图像中构造一个 HSV 图像仅仅需要对“data”数据区做一些变化，在头的定义中没有什么表示是特意为某个数据通道的。

例 3-11：仅最大化 HSV 图像“S”和“V”部分

```
void saturate_sv( IplImage* img ) {
    for( int y=0; y<img->height; ; y++ ) {
        uchar* ptr = (uchar*) (
            img->imageData + y * img->widthStep
        );
        for( int x=0; x<img->width; x++ ) {
            ptr[3*x+1] = 255;
            ptr[3*x+2] = 255;
        }
    }
}
```

在以上程序中，我们用指针 `ptr` 指向第 `y` 行的起始位置。接着，我们从指针中析出饱和度和高度在 `x` 维的值。因为这是一个三通道图像，所以 `C` 通道在 `x` 行的位置是 `3*x+c`。

与 `CvMat` 的成员 `data` 相比，`IplImage` 和 `CvMat` 之间的一个重要区别在于 `imageData`。`CvMat` 的 `data` 元素类型是联合类型，所以你必须说明需要使用的指针类型。`imageData` 指针是字节类型指针(`uchar *`)。我们已经知道是种类型的指针指向的数据是 `uchar` 类型的，这意味着，在图像上进行指针运算时，你可以简单地增加 `widthStep` (也以字节为单位)，而不必关心实际数据类型。在这里重新说明一下：当要处理的是矩阵时，必须对偏移并进行调整，因为数据指针可能是非字节类型；当要处理的是图像时，可以直接使用偏移，因为数据指针总是字节类型，因此当你要用到它的时候要清楚是怎么回事。

对 ROI 和 widthStep 的补充

`ROI` 和 `widthStep` 在实际工作中有很重要的作用，在很多情况下，使用它们会提高计算机视觉代码的执行速度。这是因为它们允许对图像的某一小部分进行操作，而不是对整个图像进行运算。在 OpenCV 中^①，普遍支持 `ROI` 和 `widthStep`，函数的操作被限于感兴趣区域。要设置或取消 `ROI`，就要使用 `cvSetImageROI()` 和 `cvResetImageROI()` 函数。如果想设置 `ROI`，可以使用函数 `cvSetImageROI()`，

① 至少在理论上，任何对 `widthStep` 和 `ROI` 的不支持都可以被视为 bug，可能被发布到 SourceForge 网站并被归入“待修复”列表。这与感兴趣的绿色通道 COI 相对比，后者只在明确说明的时候才被支持。

并为其传递一个图像指针和矩形。而取消 ROI，只需要为函数 cvResetImageROI() 传递一个图像指针。

```
void cvSetImageROI( IplImage* image, CvRect rect );
void cvResetImageROI( IplImage* image );
```

为了解释 ROI 的用法，我们假设要加载一幅图像并修改一些区域，如例 3-12 的代码，读取了一幅图像，并设置了想要的 ROI 的 x, y, width 和 height 的值，最后将 ROI 区域中像素都加上一个整数。本例程中通过内联的 cvRect() 构造函数设置 ROI。通过 cvResetImageROI() 函数释放 ROI 是非常重要的，否则，将忠实地只显示 ROI 区域。

例 3-12：用 imageROI 来增加某范围的像素

```
// roi_add <image> <x> <y> <width> <height> <add>
#include <cv.h>
#include <highgui.h>

int main(int argc, char** argv)
{
    IplImage* src;
    if( argc == 7 && ((src=cvLoadImage(argv[1],1)) != 0 ))
    {
        int x = atoi(argv[2]);
        int y = atoi(argv[3]);
        int width = atoi(argv[4]);
        int height = atoi(argv[5]);
        int add = atoi(argv[6]);
        cvSetImage ROI(src, cvRect(x,y,width,height));
        cvAddS(src, cvScalar(add), src);
        cvResetImageROI(src);
        cvNamedWindow( "Roi_Add", 1 );
        cvShowImage( "Roi_Add", src );
        cvWaitKey();
    }
    return 0;
}
```

使用例 3-12 中的代码把 ROI 集中于一张猫的脸部，并将其蓝色通道增加 150 后的效果如图 3-3 所示。【45~46】



图 3-3：在猫脸上用 ROI 增加 150 像素的效果

通过巧妙地使用 `widthStep`，我们可以达到同样的效果。要做到这一点，我们创建另一个图像头，让它的 `width` 和 `height` 的值等于 `interest_rect` 的 `width` 和 `height` 的值。我们还需要按 `interest_rect` 起点设置图像起点(左上角或者左下角)。下一步，我们设置子图像的 `widthStep` 与较大的 `interest_img` 相同。这样，即可在子图像中逐行地步进到大图像里子区域中下一行开始处的合适位置。最后设置子图像的 `imageData` 指针指向兴趣子区域的开始，如例 3-13 所示。

例 3-13：利用其他 `widthStep` 方法把 `interest_img` 的所有像素值增加 1

```
// Assuming IplImage *interest_img; and
// CvRect interest_rect;
// Use widthStep to get a region of interest
//
// (Alternate method)
//
IplImage *sub_img = cvCreateImageHeader(
    cvSize(
        interest_rect.width,
```

```

    interest_rect.height
),
interest_img->depth,
interest_img->nChannels
);

sub_img->origin = interest_img->origin;

sub_img->widthStep = interest_img->widthStep;

sub_img->imageData = interest_img->imageData +
interest_rect.y * interest_img->widthStep +
interest_rect.x * interest_img->nChannels;

cvAddS( sub_img, cvScalar(1), sub_img );

cvReleaseImageHeader(&sub_img);

```

看起来设置和重置 ROI 更方便一些，为什么还要使用 widthStep？原因在于有些时候在处理的过程中，想在操作过程中设置和保持一幅图像的多个子区域处于活动状态，但是 ROI 只能串行处理并且必须不断地设置和重置。

最后，我们要在此提到一个词——掩码或模板，在代码示例中 cvAddS() 函数允许第四个参数默认值为空：const CvArr* mask=NULL。这是一个 8 位单通道数组，它允许把操作限制到任意形状的非 0 像素的掩码区，如果 ROI 随着掩码或模板变化，进程将会被限制在 ROI 和掩码的交集区域。掩码或模板只能在指定了其图像的函数中使用。

矩阵和图像操作

表 3-3 列出了一些操作矩阵图像的函数，其中的大部分对于图像处理非常有效。它们实现了图像处理中的基本操作，例如对角化、矩阵变换以及一些更复杂的诸如计算图像的统计操作。【47】

表 3-3：矩阵和图像基本操作

函数名称	描述
cvAbs	计算数组中所有元素的绝对值
cvAbsDiff	计算两个数组差值的绝对值

续表

函数名称	描述
cvAbsDiffS	计算数组和标量差值的绝对值
cvAdd	两个数组的元素级的加运算
cvAddS	一个数组和一个标量的元素级的相加运算
cvAddWeighted	两个数组的元素级的加权相加运算(alpha 融合)
cvAvg	计算数组中所有元素的平均值
cvAvgSdv	计算数组中所有元素的绝对值和标准差
cvCalcCovarMatrix	计算一组 n 维空间向量的协方差
cvCmp	对两个数组中的所有元素运用设置的比较操作
cvCmpS	对数组和标量运用设置的比较操作
cvConvertScale	用可选的缩放值转换数组元素类型
cvConvertScaleAbs	计算可选的缩放值的绝对值之后再转换数组元素的类型
cvCopy	把数组中的值复制到另一个数组中
cvCountNonZero	计算数组中非 0 值的个数
cvCrossProduct	计算两个三维向量的向量积(叉积)
cvCvtColor	将数组的通道从一个颜色空间转换另外一个颜色空间
cvDet	计算方阵的行列式
cvDiv	用另外一个数组对一个数组进行元素级的除法运算
cvDotProduct	计算两个向量的点积
cvEigenVV	计算方阵的特征值和特征向量
cvFlip	围绕选定轴翻转
cvGEMM	矩阵乘法
cvGetCol	从一个数组的列中复制元素
cvGetCols	从数据的相邻的多列中复制元素值
cvGetDiag	复制数组中对角线上的所有元素
cvGetDims	返回数组的维数
cvGetDimSize	返回一个数组的所有维的大小
cvGetRow	从一个数组的行中复制元素值
cvGetRows	从一个数组的多个相邻的行中复制元素值
cvGetSize	得到二维的数组的尺寸, 以 CvSize 返回
cvGetSubRect	从一个数组的子区域复制元素值
cvInRange	检查一个数组的元素是否在另外两个数组中的值的范围内
cvInRangeS	检查一个数组的元素的值是否在另外两个标量的范围内

续表

函数名称	
cvInvert	求矩阵的转置
cvMahalanobis	计算两个向量间的马氏距离
cvMax	在两个数组中进行元素级的取最大值操作
cvMaxS	在一个数组和一个标量中进行元素级的取最大值操作
cvMerge	把几个单通道图像合并为一个多通道图像
cvMin	在两个数组中进行元素级的取最小值操作
cvMinS	在一个数组和一个标量中进行元素级的取最小值操作
cvMinMaxLoc	寻找数组中的最大最小值
cvMul	计算两个数组的元素级的乘积
cvNot	按位对数组中的每一个元素求反
cvNorm	计算两个数组的正态相关性
cvNormalize	将数组中元素进行规一化
cvOr	对两个数组进行按位或操作
cvOrS	在数组与标量之间进行按位或操作
cvReduce	通过给定的操作符将二维数组约简为向量
cvRepeat	以平铺的方式进行数组复制
cvSet	用给定值初始化数组
cvSetZero	将数组中所有元素初始化为 0
cvSetIdentity	将数组中对角线上的元素设为 1, 其他置 0
cvSolve	求出线性方程组的解
cvSplit	将多通道所组分割成多个单通道数组
cvSub	两个数组元素级的相减
cvSubS	元素级的从数组中减去标量
cvSubRS	元素级的从标量中减去数组
cvSum	对数组中的所有元素求和
cvSVD	二维矩阵的奇异值分解
cvSVBkSb	奇异值回代计算
cvTrace	计算矩阵迹
cvTranspose	矩阵的转置运算
cvXor	对两个数组进行按位异或操作
cvXorS	在数组和标量之间进行按位异或操作
cvZero	将所有数组中的元素置为 0

cvAbs, cvAbsDiff 和 cvAbsDiffS

```
void cvAbs(
    const CvArr* src,
    const         dst
);
void cvAbsDiff(
    const CvArr* src1,
    const CvArr* src2,
    const         dst
);
void cvAbsDiffS(
    const CvArr* src,
    CvScalar      value,
    const         dst
);
```

【50】

这些函数计算一个数组的绝对值或数组和其他对象的差值的绝对值，cvAbs()函数计算 src 里的值的绝对值，然后把结果写到 dst；cvAbsDiff()函数会先从 src1 减去 src2，然后将所得差的绝对值写到 dst；除了从所有 src 元素减掉的数是常标量值外，可以看到 cvAbsDiffS() 函数同 cvAbsDiff() 函数基本相同。

cvAdd, cvAddS, cvAddWeighted 和 alpha 融合

```
void cvAdd(
    const CvArr* src1,
    const CvArr* src2,
    CvArr*       dst,
    const CvArr* mask = NULL
);
void cvAddS(
    const CvArr* src,
    CvScalar      value,
    CvArr*       dst,
    const CvArr* mask = NULL
);
void cvAddWeighted(
    const CvArr* src1,
```

```

    double      alpha,
    const CvArr* src2,
    double      beta,
    double      gamma,
    CvArr*     dst
);

```

`cvAdd()`是一个简单的加法函数，它把 `src1` 里的所有元素同 `src2` 里的元素对应进行相加，然后把结果放到 `dst`，如果 `mask` 没有被设为 `NULL`，那么由 `mask` 中非零元素指定的 `dst` 元素值在函数执行后不变。`cvAddS()` 与 `cvAdd()` 非常相似，惟一不同的是被加的数量标量 `value`。

`cvAddWeighted()` 函数同 `cvAdd()` 类似，但是被写入 `dst` 的结果是经过下面的公式得出的：

$$dst_{x,y} = \alpha \cdot src1_{x,y} + \beta \cdot src2_{x,y} + \gamma \quad \text{【50】}$$

这个函数可用来实现 *alpha* 融合 [Smith79; Porter84]；也就是说，它可以用一个图像同另一个图像的融合，函数的形式如下：

```

void cvAddWeighted(
    const CvArr* src1,
    double      alpha,
    const CvArr* src2,
    double      beta,
    double      gamma,
    CvArr*     dst
);

```

在函数 `cvAddWeighted()` 中我们有两个源图像，分别是 `src1` 和 `src2`。这些图像可以是任何类型的像素，只要它们属于同一类型即可。它们还可以有一个或三个通道(灰度或彩色)，同样也要保持类型一致。结果图像 `dst`，也必须同 `src1` 和 `src2` 是相同的像素类型。这些图像可能是不同尺寸，但是它们的 ROI 必须统一尺寸，否则 OpenCV 就会产生错误，参数 `alpha` 是 `src1` 的融合强度，`beta` 是 `src2` 的融合强度，`alpha` 融合公式如下：

$$dst_{x,y} = \alpha \cdot src1_{x,y} + \beta \cdot src2_{x,y} + \gamma$$

可以通过设置 α 从 0 到 1 区间取值， $\beta = 1 - \alpha$ ， γ 为 0，将前面公式转换为标准 *alpha* 融合方程。这就得出下式：

$$dst_{x,y} = \alpha \cdot src1_{x,y} + (1 - \alpha) \cdot src2_{x,y}$$

但是，在加权融合图像，以及目标图像的附加偏移参数 γ 方面，cvAddWeighted() 提供了更大的灵活性。一般而言，你或许想让 alpha 和 beta 不小于 0，并且两者之和不大于 1，gamma 的设置取决于像素所要调整到的平均或最大值。例 3-14 展示了 alpha 融合的用法。

例 3-14：src2 中 alpha 融合 ROI 以(0,0)开始，src1 中 ROI 以(x,y)开始

```
// alphablend <imageA> <image B> <x> <y> <width> <height>
//           <alpha> <beta>
#include <cv.h>
#include <highgui.h>

int main(int argc, char** argv)
{
    IplImage *src1, *src2;
    if( argc == 9 && ((src1=cvLoadImage(argv[1],1)) != 0
        )&&((src2=cvLoadImage(argv[2],1)) != 0 ))
    {
        int x = atoi(argv[3]);
        int y = atoi(argv[4]);
        int width = atoi(argv[5]);
        int height = atoi(argv[6]);
        double alpha = (double)atof(argv[7]);
        double beta = (double)atof(argv[8]);
        cvSetImage ROI(src1, cvRect(x,y,width,height));
        cvSetImageROI(src2, cvRect(0,0,width,height));
        cvAddWeighted(src1, alpha, src2, beta, 0.0,src1);
        cvResetImageROI(src1);
        cvNamedWindow( "Alpha_blend", 1 );
        cvShowImage( "Alpha_blend", src1 );
        cvWaitKey();
    }
    return 0;
}
```

【51~52】

例 3-14 中的代码用两个源图像：初始的(src1)和待融合的(src2)。它从矩形的 ROI 中读取 src1，然后将同样大小的 ROI 应用到 src2 中，这一次设在原始位置，它从命令行读入 alpha 和 beta 的级别但是把 gamma 设为 0。Alpha 融合使用函数 cvAddWeighted()，结果被放到 src1 并显示，例子输出如图 3-4 所示，一个小孩

的脸同一个猫的脸和身体被融合到了一起，值得注意的是，代码采用和图 3-3 不同的方法，像图 3-3 的例子一样。这次我们使用了 ROI 作为目标融合区域。



图 3-4：一个小孩的脸被 alpha 融合到一只猫的脸上

cvAnd 和 cvAndS

```
void cvAnd(
    const CvArr* src1,
    const CvArr* src2,
    CvArr*       dst,
    const CvArr* mask = NULL
);
void cvAndS(
    const CvArr* src1,
```

```
CvScalar      value,  
CvArr*       dst,  
const CvArr* mask = NULL  
);
```

这两个函数在 `src1` 数组上做按位与运算，在 `cvAnd()` 中每个 `dst` 元素都是由相应的 `src1` 和 `src2` 两个元素进行位与运算得出的。在 `cvAndS()` 中，位与运算由常标量 `value` 得出。同一般函数一样，如果 `mask` 是非空，就只计算非 0 `mask` 元素所对应的 `dst` 元素。

尽管支持所有的数据类型，但是对于 `cvAnd()` 来说，`src1` 和 `src2` 要保持相同的数据类型。如果元素都是浮点型的，则使用该浮点数的按位表示。 【52】

cvAvg

```
CvScalar cvAvg(  
    const CvArr* arr,  
    const CvArr* mask = NULL  
);
```

`cvAvg()` 计算数组 `arr` 的平均像素值，如果 `mask` 为非空，那么平均值仅由那些 `mask` 值为非 0 的元素相对应的像素算出。

此函数还有别名 `.cvMean()`，但不推荐使用。

cvAvgSdv

```
cvAvgSdv(  
    const CvArr* arr,  
    CvScalar*   mean,  
    CvScalar*   std_dev,  
    const CvArr* mask     = NULL  
);
```

【53】

此函数同 `cvAvg()` 类似，但除了求平均，还可以计算像素的标准差。

函数现在有不再使用的别名 `cvMean_StdDev()`。

cvCalcCovarMatrix

```
void cvCalcCovarMatrix(
    const CvArr** vrets,
    int count,
    CvArr* cov_mat,
    CvArr* avg,
    int flags
);
```

给定一些向量，假定这些向量表示的点是高斯分布，`cvCalcCovarMatrix()`将计算这些点的均值和协方差矩阵。这当然可以运用到很多方面，并且 OpenCV 有很多附加的 `flags` 值，在特定的环境下会起作用(参见表 3-4)。这些标志可以用标准的布尔或操作组合到一起。

表 3-4: `cvCalcCovarMatrix()`可能用到的标志参数的值

标志参数的名称及值	意义
<code>CV_COVAR_NORMAL</code>	计算均值和协方差
<code>CV_COVAR_SCRAMBLED</code>	快速 PCA “Scrambled” 协方差
<code>CV_COVAR_USE_AVERAGE</code>	输入均值而不是计算均值
<code>CV_COVAR_SCALE</code>	重新缩放输出的协方差矩阵

在所有情况下，在 `vrets` 中是 OpenCV 指针数组(即一个指向指针数组的指针)，并有一个指示多少数组的参数 `count`。在所有情况下，结果将被置于 `cov_mat`，但是 `avg` 的确切含义取决于标志的值(参见表 3-4)。

标识 `CV_COVAR_NORMAL` 和 `CV_COVAR_SCRAMBLED` 是相互排斥的；只能使用其中一种，不能两者同时使用。如果为 `CV_COVAR_NORMAL`，函数便只计算该点的均值和协方差。

$$\sum_{\text{normal}}^2 = z \begin{bmatrix} v_{0,0} - \bar{v}_0 & \cdots & v_{m,0} - \bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n} - \bar{v}_n & \cdots & v_{m,n} - \bar{v}_n \end{bmatrix} \begin{bmatrix} v_{0,0} - \bar{v}_0 & \cdots & v_{m,0} - \bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n} - \bar{v}_n & \cdots & v_{m,n} - \bar{v}_n \end{bmatrix}^T$$

因此，标准的协方差 \sum_{normal}^2 由长度为 n 的 m 个向量计算，其中 \bar{v}_n 被定义为平均向量 \bar{v} 的第 n 个元素，由此产生的协方差矩阵是一个 $n \times n$ 矩阵，比例 z 是一个可选的缩放比例，除非使用 `CV_COVAR_SCALE` 标志，否则它将被设置为 1。 【54】

如果是 CV_COVAR_SCRAMBLED 标志，cvCalcCovarMatrix() 将如下计算：

$$\sum_{\text{scrambled}}^2 = z \begin{bmatrix} v_{0,0} - \bar{v}_0 & \cdots & v_{m,0} - \bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n} - \bar{v}_n & \cdots & v_{m,n} - \bar{v}_n \end{bmatrix}^T \begin{bmatrix} v_{0,0} - \bar{v}_0 & \cdots & v_{m,0} - \bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n} - \bar{v}_n & \cdots & v_{m,n} - \bar{v}_n \end{bmatrix}$$

这种矩阵不是通常的协方差矩阵(注意转置运算符的位置)，这种矩阵的计算来自同样长度为 n 的 m 个向量，但由此而来的协方差矩阵是一个 $m \times m$ 矩阵。这种矩阵是用在一些特定的算法中，如针对非常大的向量的快速 PCA 分析法(人脸识别可能会用到此运算)。

如果已知平均向量，则使用标志 CV_COVAR_USE_AVG，在这种情况下，参数 avg 用来作为输入而不是输出，从而减少计算时间。

最后，标志 CV_COVAR_SCALE 用于对计算得到的协方差矩阵进行均匀缩放。这是前述方程的比例 z ，同标志 CV_COVAR_NORMAL 一起使用时，应用的缩放比例将是 $1.0/m$ (或等效于 $1.0/\text{count}$)。如果不使用 CV_COVAR_SCRAMBLED，那么 z 的值将会是 $1.0/n$ (向量长度的倒数)，cvCalcCovarMatrix() 的输入输出矩阵都应该是浮点型，结果矩阵 cov_mat 的大小应当是 $n \times n$ 或者 $m \times m$ ，这取决于计算的是标准协方差还是 scrambled 的协方差。应当指出的是，在 vcts 中输入的“向量”并不一定要是一维的；它们也可以是二维对象(例如图像)。

cvCmp 和 cvCmpS

```
void cvCmp(
    const CvArr* src1,
    const CvArr* src2,
    CvArr* dst,
    int cmp_op
);
void cvCmpS(
    const CvArr* src,
    double value,
    CvArr* dst,
    int cmp_op
);
```

这两个函数都是进行比较操作，比较两幅图像相应的像素值或将给定图像的像素值与某常标量值进行比较。cvCmp() 和 cvCmpS() 的最后一个参数的比较操作符可以是表 3-5 所列出的任意一个。

【55】

表 3-5: cvCmp() 和 cvCmpS() 使用的 cmp_op 值以及由此产生的比较操作

cmp_op 的值	由该值表示的比较方法
CV_CMP_EQ	(src1i == src2i)
CV_CMP_GT	(src1i > src2i)
CV_CMP_GE	(src1i >= src2i)
CV_CMP_LT	(src1i < src2i)
CV_CMP_LE	(src1i <= src2i)
CV_CMP_NE	(src1i != src2i)

表 3-5 列出的比较操作都是通过相同的函数实现的，只需传递合适的参数来说明你想怎么做，这些特殊的功能操作只能应用于单通道的图像。

这些比较功能适用于这样的应用程序，当你使用某些版本的背景减法并想对结果进行掩码处理但又只从图像中提取变化区域信息时(如从安全监控摄像机看一段视频流)。

cvConvertScale

```
void cvConvertScale(
    const CvArr*     src,
    CvArr*           dst,
    double           scale = 1.0,
    double           shift = 0.0
);
```

cvConvertScale() 函数实际上融多种功能于一体，它能执行几个功能中的任意之一，如果需要，也可以一起执行多个功能。第一个功能是将源图像的数据类型转变成目标图像的数据类型。例如，如果我们有一个 8 位的 RGB 灰度图像并想把它变为 16 位有符号的图像，就可以调用函数 cvConvertScale() 来做这个工作。

cvConvertScale() 的第二个功能是对图像数据执行线性变换。在转换成新的数据类型之后，每个像素值将乘以 scale 值，然后将 shift 值加到每个像素上。

至关重要的是要记住，尽管在函数名称中“Convert”在“Scale”之前，但执行这些操作的顺序实际上是相反的。具体来说，在数据类型转变之前，与 scale 相乘和 shift 的相加已经发生了。【56】

如果只是传递默认值 (scale = 1.0 和 shift = 0.0)，则不必担心性能；OpenCV 足够聪明，能意识到这种情况而不会在无用的操作上浪费处理器的时间。澄清一下（如果你想添加一些），OpenCV 还提供了宏指令 cvConvert()，该指令同 cvConvertScale()一样，但是通常只适用于 scale 和 shift 参数设为默认值时。

对于所有数据类型和任何数量通道 cvConvertScale () 都适用，但是源图像和目标图像的通道数量必须相同。（如果你想实现彩色图像与灰度图的相互转换，可以使用 cvCvtColor()，之后我们将会提到。）

【56~57】

cvConvertScaleAbs

```
void cvConvertScaleAbs(
    const CvArr*     src,
    CvArr*          dst,
    double           scale = 1.0,
    double           shift = 0.0
);
```

cvConvertScaleAbs() 与 cvConvertScale() 基本相同，区别是 dst 图像元素是结果数据的绝对值。具体说来，cvConvertScaleAbs() 先缩放和平移，然后算出绝对值，最后进行数据类型的转换。

cvCopy

```
void cvCopy(
    const CvArr*     src,
    CvArr*          dst,
    const CvArr*     mask = NULL
);
```

用于将一个图像复制到另一个图像。cvCopy() 函数要求两个数组具有相同的数据类型、相同的大小和相同的维数。可以使用它来复制稀疏矩阵，但这样做时，不支持 mask。对于非稀疏矩阵和图像，mask 如果为非空，则只对与 mask 中与非 0 值相对应的 dst 中的像素赋值。

cvCountNonZero

```
int cvCountNonZero( const CvArr* arr );
```

cvCountNonZero()返回数组 arr 中非 0 像素的个数。

cvCrossProduct

```
void cvCrossProduct(
    const CvArr* src1,
    const CvArr* src2,
    CvArr* dst
);
```

这个函数的主要功能是计算两个三维向量的叉积[Lagrange1773]。无论向量是行或者列的形式函数都支持。(实际上对于单通道矩阵，行向量和列向量的数据在内存中的排列方式完全相同)。src1 和 src2 都必须是单道数组，同时 dst 也必须是单道的，并且长度应精确为 3。所有这些阵列的数据类型都要一致。 【57】

cvCvtColor

```
void cvCvtColor(
    const CvArr* src,
    CvArr* dst,
    int code
);
```

此前介绍的几个函数用于把一个数据类型转换成另一个数据类型，原始图像和目标图像的通道数目应保持一致。另外一个函数是 cvCvtColor()，当数据类型一致时，它将图像从一个颜色空间(通道的数值)转换到另一个[Wharton71]。具体转换操作由参数 code 来指定，表 3-6^①列出了此参数可能的值。

① 用过 IPL 的用户会注意到一点：cvCvtColor() 会忽略 IplImage 头中的 colorModel 和 channelSeq 属性。转换根据所提供的参数 code 来执行。

表 3-6: cvCvtColor()的转换

代码	解释
CV_BGR2RGB	在 RGB 或 BGR 色彩空间之间转换(包括或者不包括 alpha 通道)
CV_RGB2BGR	
CV_RGBA2BGRA	
CV_BGRA2RGBA	
CV_RGB2RGBA	在 RGB 或 BGR 图像中加入 alpha 通道
CV_BGR2BGRA	
CV_RGBA2RGB	从 RGB 或 BGR 图像中删除 alpha 通道
CV_BGRA2BGR	
CV_RGB2BGRA	加入或者移除 alpha 通道时, 转换 RGB 到 BGR 色彩空间
CV_RGBA2BGR	
CV_BGRA2RGB	
CV_BGR2RGBA	
CV_RGB2GRAY	转换 RGB 或者 BGR 色彩空间为灰度空间
CV_BGR2GRAY	
CV_GRAY2RGB	转换灰度为 RGB 或者 BGR 色彩空间(在进程中选择移除 alpha 通道)
CV_GRAY2BGR	
CV_RGBA2GRAY	
CV_BGRA2GRAY	
CV_GRAY2RGBA	转换灰度为 RGB 或者 BGR 色彩空间并且加入 alpha 通道
CV_GRAY2BGRA	
CV_RGB2BGR565	在从 RGB 或者 BGR 色彩空间转换到 BGR565 彩色图画时, 选择加入或者移除 alpha 通道 (16 位图)
CV_BGR2BGR565	
CV_BGR5652RGB	
CV_BGR5652BGR	
CV_RGBA2BGR565	
CV_BGRA2BGR565	
CV_BGR5652RGBA	
CV_BGR5652BGRA	
CV_GRAY2BGR565	转换灰度为 BGR565 彩色图像或者反变换(16 位图)
CV_BGR5652GRAY	

续表

转换代码	功能描述
CV_RGB2BGR555	在从 RGB 或者 BGR 色彩空间转换到 BGR555 色彩空间时，选择加入或者移除 alpha 通道(16 位图)
CV_BGR2BGR555	
CV_BGR5552RGB	
CV_BGR5552BGR	
CV_RGBA2BGR555	
CV_BGRA2BGR555	
CV_BGR5552RGBA	
CV_BGR5552BGRA	
CV_GRAY2BGR555	转换灰度到 BGR555 色彩空间或者反变换(16 位图)
CV_BGR5552GRAY	
CV_RGB2XYZ	转换 RGB 或者 BGR 色彩空间到 CIE XYZ 色彩空间或者反变换(Rec 709 和 D65 白点)
CV_BGR2XYZ	
CV_XYZ2RGB	
CV_XYZ2BGR	
CV_RGB2YCrCb	转换 RGB 或者 BGR 色彩空间到 luma-chroma (aka YCC)色彩空间
CV_BGR2YCrCb	
CV_YCrCb2RGB	
CV_YCrCb2BGR	
CV_RGB2HSV	转换 RGB 或者 BGR 色彩空间到 HSV(hue, saturation, value)色彩空间或反变换
CV_BGR2HSV	
CV_HSV2RGB	
CV_HSV2BGR	
CV_RGB2HLS	转换 RGB 或者 BGR 色彩空间到 HLS(hue, Lightness, saturation)色彩空间或反变换
CV_BGR2HLS	
CV_HLS2RGB	
CV_HLS2BGR	
CV_RGB2Lab	转换 RGB 或者 BGR 色彩空间到 CIE LAB 色彩空间或反变换
CV_BGR2Lab	
CV_Lab2RGB	
CV_Lab2BGR	

续表

转换代码	解释
CV_RGB2Luv	转换 RGB 或者 BGR 色彩空间到 CIE Luv 色彩空间
CV_BGR2Luv	
CV_Luv2RGB	
CV_Luv2BGR	
CV_BayerBG2RGB	转换 Bayer 模式(单通道)到 RGB 或者 BGR 色彩空间
CV_BayerGB2RGB	
CV_BayerRG2RGB	
CV_BayerGR2RGB	
CV_BayerBG2BGR	
CV_BayerGB2BGR	
CV_BayerRG2BGR	
CV_BayerGR2BGR	

这里不再进一步阐述 CIE 色彩空间中 Bayer 模式的细节，但许多这样的转换是很有意义的。我们的目的是，了解 OpenCV 能够在哪些色彩空间进行转换，这对用户来说很重要。

色彩空间转换都用到以下约定：8 位图像范围是 0~255，16 位图像范围是 0~65536，浮点数的范围是 0.0~1.0。黑白图像转换为彩色图像时，最终图像的所有通道都是相同的；但是逆变换(例如 RGB 或 BGR 到灰度)，灰度值的计算使用加权公式：

$$Y=(0.299)R+(0.587)G+(0.114)B$$

就 HSV 色彩模式或者 HLS 色彩模式来说，色调通常是在 0~360 之间。^①在 8 位图中，这可能出现问题，因此，转换到 HSV 色彩模式，并以 8 位图的形式输出时，色调应该除以 2。

cvDet

```
double cvDet(const CvArr* mat);
```

cvDet() 用于计算一个方阵的行列式。这个数组可以是任何数据类型，但它必须是

① 当然不包括 360。

单通道的，如果是小的矩阵，则直接用标准公式计算。然而对于大型矩阵，这样就不是很有效，行列式的计算使用高斯消去法。

值得指出的是，如果已知一个矩阵是对称正定的，也可以通过奇异值分解的策略来解决。欲了解更多信息，请参阅“cvSVD”一节。但这个策略是将 U 和 V 设置为 NULL，然后矩阵 W 的乘积就是所求正定矩阵。

cvDiv

```
void cvDiv(
    const CvArr* src1,
    const CvArr* src2,
    CvArr* dst,
    double scale = 1
);
```

cvDiv 是一个实现除法的简单函数，它用 src2 除以 src1 中对应元素，然后把最终的结果存到 dst 中。如果 mask 非空，那么 dst 中的任何与 mask 中 0 元素相对应的元素都不改变。如果对数组中所有元素求倒数，则可以设置 src1 为 NULL，函数将假定 src1 是一个元素全为 1 的数组。

cvDotProduct

```
double cvDotProduct(
    const CvArr* src1,
    const CvArr* src2
);
```

【58~60】

这个函数主要计算两个 N 维向量^①的点积[Lagrange1773]。与叉积函数相同，点积函数也不太关注向量是行或者是列的形式。src1 和 src2 都应该是单通道的数组，并且数组的数据类型应该一致。

① 事实上，cvDotProduct() 函数有更多用途，给出任意一对 $n \times m$ 的矩阵，cvDotProduct() 都会返回一个对应元素的乘积之和。

cvEigenVV

```
double cvEigenVV(
    CvArr* mat,
    CvArr* evecs,
    CvArr* evals,
    double eps = 0
);
```

对对称矩阵 `mat`, `cvEigenVV()`会计算出该矩阵的特征值和相应的特征向量。函数实现的是雅可比方法[Bronshtein97], 对于小的矩阵是非常高效的^①, 雅可比方法需要一个停止参数, 它是最终矩阵中偏离对角线元素最大尺寸^②。可选参数 `eps` 用于设置这个值。在计算的过程中, 所提供的矩阵 `mat` 的数据空间被用于计算, 所以, 它的值会在调用函数后改变。函数返回时, 你会在 `evecs` 中找到以行顺序保存的特征向量。对应的特征值被存储到 `evals` 中。特征向量的次序以对应特征值的重要性按降序排列。该 `cvEigenVV()` 函数要求所有三个矩阵具有浮点类型。

正如 `cvDet()`(前述), 如果被讨论的向量是已知的对称和正定矩阵^③, 那么最好使用 SVD 计算 `mat` 的特征值和特征向量。

cvFlip

```
void cvFlip(
    const CvArr* src,
    CvArr* dst = NULL,
    int flip_mode = 0
);
```

本函数是将图像绕着在 X 轴或 Y 轴或者绕着 X 轴或 Y 轴上同时旋转。当参数 `flip_mode` 被设置为 0 的时候, 图像只会绕 X 轴旋转。 【61】

-
- ① 一个好的经验法则是, 选用矩阵 10×10 或更小的矩阵, 使用雅可比方法是有效的。如果矩阵是大于 20×20 , 可能无法进行下去。
 - ② 从原则上说, 一旦雅可比方法完成, 那么原始矩阵转化为一个对角阵, 且仅包含特征值; 然而, 该方法可以在非对角线元素都为 0 之前终止, 以节省计算量。在实践中, 它通常将此值设置为 `DBL_EPSILON`, 或大约 10^{-15} 。
 - ③ 这是协方差矩阵情况的举例。详见 `cvCalcCovarMatrix()`。

`flip_mode` 被设置为正值时(例如, +1), 图像会围绕 Y 轴旋转, 如果被设置成负值(例如, -1), 图像会围绕 X 轴和 Y 轴旋转。

在 Win32 运行视频处理系统时, 你会发现自己经常使用此功能来进行图像格式变换, 也就是坐标原点在左上角和左下角的变换。

cvGEMM

```
double cvGEMM(
    const CvArr* src1,
    const CvArr* src2,
    double alpha,
    const CvArr* src3,
    double beta,
    CvArr* dst,
    int tABC = 0
);
```

广义矩阵乘法(generalized matrix multiplication, GEMM)在 OpenCV 中是由 `cvGEMM()` 来实现的, 可实现矩阵乘法、转置后相乘、比例缩放等。最常见的情况下, `cvGEMM()` 计算如下:

$$\mathbf{D} = \alpha \cdot \text{op}(\mathbf{A}) \cdot \text{op}(\mathbf{B}) + \beta \cdot \text{op}(\mathbf{C})$$

其中 \mathbf{A} , \mathbf{B} 和 \mathbf{C} 分别是矩阵 `src1`, `src2` 和 `src3`, α 和 β 是数值系数, `op()` 是附在矩阵上的可选转置。参数 `src3` 可以设置为空。在这种情况下, 不会参与计算。转置将由可选参数 `tABC` 来控制, 它的值可以是 0 或者(通过布尔 OR 操作)`CV_GEMM_A_T`、`CV_GEMM_B_T` 和 `CV_GEMM_C_T` 的任何组合(每一个标志都有一个矩阵转换相对应)。

过去的 OpenCV 包含 `cvMatMul()` 和 `cvMatMulAdd()` 方法, 但是, 它们很容易和 `cvMul()` 混淆, 其实它们的功能是完全不一样的(即两个数组的元素与元素相乘)。这个函数以宏的形式继续存在, 它们直接调用 `cvGEMM()`。两者对应关系如表 3-7 所示。

表 3-7: `cvGEMM()`一般用法的宏别名

<code>cvMatMul(A, B, D)</code>	<code>cvGEMM(A, B, 1, NULL, 0, D, 0)</code>
<code>cvMatMulAdd(A, B, C, D)</code>	<code>cvGEMM(A, B, 1, C, 1, D, 0)</code>

只有大小符合约束的矩阵才能进行乘法运算，并且所有的数据类型都应该是浮点型。cvGEMM()函数支持双通道矩阵，在这种情况下，它将双通道视为一个复数的两个部分。

cvGetCol 和 cvGetCols

```
CvMat* cvGetCol(
    const CvArr* arr,
    CvMat* submat,
    int col
);
CvMat* cvGetCols(
    const CvArr* arr,
    CvMat* submat,
    int start_col,
    int end_col
);
```

cvGetCol()函数被用作提取矩阵中的某一列，并把它以向量的形式返回(即只有一列的矩阵)。在这种情况下，矩阵指针 submat 将被修改为指向 arr 中的特定列，必须指出的是，该指针在修改过程中并未涉及内存的分配或数据的复制；submat 的内容仅仅是作了简单修改以使它正确地指出 arr 中所选择的列。它支持所有数据类型。

cvGetCcols()函数的工作原理与 cvGetCols 完全一致，区别只在于前者将选择从 start_col 到 end_col 之间的所有列。这两个函数都返回一个与被调用的特定列或者多列(即，submat)相对应的头指针。

cvGetDiag

```
CvMat* cvGetDiag(
    const CvArr* arr,
    CvMat* submat,
    int diag= 0
);
```

cvGetDiag ()类似于 cvGetCol()；它能从一个矩阵选择某一条对角线并将其作为向量返回。submat 是一个矩阵类型的头指针。函数 cvGetDiag()将填充该向量头指针中的各分量，以使用指向 arr 中的正确信息。注意，调用 cvGetDiag()会修

改输入的头指针，将数据指针指向 arr 对角线上的数据，实际上，并没有复制 arr 的数据。可选参数 diag 表明 submat 指向哪一条对角线的。如果 diag 被设置为默认值 0，主对角线将被选中。如果 diag 大于 0，则始于(diag, 0)的对角线将被选中，如果 diag 小于 0，则始于(0, -diag)的对角线将被选中。cvGetDiag()并不要求矩阵 arr 是方阵，但是数组 submat 长度必须与输入数组的尺寸相匹配。当该函数被调用时，最终的返回结果与输入的 submat 相同。

cvGetDims 和 cvGetDimSize

```
int cvGetDims(
    const CvArr* arr,
    int* sizes=NULL
),
int cvGetDimSize(
    const CvArr* arr,
    int index
);
```

【63】

您一定还记得 OpenCV 中的矩阵维数可以远远大于 2。函数 cvGetDims()返回指定数组的维数并可返回每一个维数的大小。如果数组 sizes 非空，那么大小将被写入 sizes。如果使用了参数 sizes，它应该是一个指向 n 个整数的指针，这里的 n 指维数。如果无法事先获知维数，为了安全起见，可以把 sizes 大小指定为 CV_MAX_DIM。

函数 cvGetDimSize()返回一个由 index 参数指定的某一维大小。如果这个数组是矩阵或者图像，那么 cvGetDims()将一直返回为 2。^①对于矩阵和图像，由 cvGetDims()返回的 sizes 的次序将总是先是行数然后是列数。

cvGetRow 和 cvGetRows

```
CvMat* cvGetRow(
    const CvArr* arr,
    CvMat* submat,
```

① 请记住，OpenCV 视“向量”为 $n \times 1$ 或 $1 \times n$ 的矩阵。

```
    int row  
);  
CvMat* cvGetRows(  
    const CvArr* arr,  
    CvMat* submat,  
    int start_row,  
    int end_row  
);
```

`cvgetRow()` 获取矩阵中的一行让它作为向量(仅有一行的矩阵)返回。跟 `cvGetCol()` 类似, 矩阵头指针 `submat` 将被修改为指向 `arr` 中的某个特定行, 并且对该头指针的修改不涉及内存的分配和数据的复制; `submat` 的内容仅是作为适当的修改以使它正确地指向 `arr` 中所选择的行。该指针所有数据类型。

`cvGetRows()` 函数的工作原理与 `cvgetRow()` 完全一致, 区别只在于前者将选择从 `start_row` 到 `end_row` 之间的所有行。这两个函数都返回一个的头指针, 指向特定行或者多个行。

cvGetSize

```
CvSize cvGetSize( const CvArr* arr );
```

它与 `cvGetDims()` 密切相关, `cvGetDims()` 返回一个数组的大小。主要的不同是 `cvGetSize()` 是专为矩阵和图像设计的, 这两种对象的维数总是 2。其尺寸可以以 `CvSize` 结构的形式返回, 例如当创建一个新的大小相同的矩阵或图像时, 使用此函数就很方便。 【64】

cvGetSubRect

```
cvGetSubRect  
CvSize cvGetSubRect(  
    const CvArr* arr,  
    CvArr* submat,  
    CvRect rect  
);
```

`cvGetSubRect()` 与 `cvGetColumns()` 或 `cvGetRows()` 非常类似, 区别在于 `cvGetSubRect()` 通过参数 `rect` 在数组中选择一个任意的子矩阵。与其他选择数组子区域的函数一样, `submat` 仅仅是一个被 `cvGetSubRect()` 函数填充的头, 它将指向用户期望的子矩阵数据, 这里并不涉及内存分配和数据的复制。

cvInRange 和 cvInRangeS

```
void cvInRange(const CvArr* src,
               const CvArr* lower,
               const CvArr* upper,
               CvArr*       dst
);
void cvInRangeS(
               const CvArr* src,
               CvScalar    lower,
               CvScalar    upper,
               CvArr*      dst
);
```

这两个函数可用于检查图像中像素的灰度是否属于某一指定范围。cvInRange() 检查，src 的每一个像素点是否落在 lower 和 upper 范围中。如果 src 的值大于或者等于 lower 值，并且小于 upper 值，那么 dst 中对应的对应值将被设置为 0xff，否则，dst 的值将被设置为 0。

cvInRangeS() 的原理与之完全相同，但 src 是与 lower 和 upper 中的一组常量值（类型 CvScalar）进行比较。对于这两个函数，图像 src 可以是任意类型；如果图像有多个通道，那么每一种通道都将被分别处理。注意，dst 的尺寸和通道数必须与 src 一致，且必须为 8 位的图像。

cvInvert

```
double cvInvert(
               const CvArr* src,
               CvArr*      dst,
               Int         method = CV_LU
);
```

cvInvert() 求取保存在 src 中的矩阵的逆并把结果保存在 dst 中。这个函数支持使用多种方法来计算矩阵的逆（见表 3-8），但默认采取的是高斯消去法。该函数的返回值与所选用的方法有关。【65】

表 3-8: cvInvert()函数中指定方法的参数值

方法的参数值	含义
CV_LU	高斯消去法 (LU 分解)
CV_SVD	奇异值分解(SVD)
CV_SVD_SYM	对称矩阵的 SVD

就高斯消去法(method=CV_LU)来说，当函数执行完毕，src 的行列式将被返回。如果行列式是 0，那么事实上不进行求逆操作，并且数组 dst 将被设置为全 0。

就 CV_SVD 或者 CV_SVD_SYM，来说，返回值是矩阵的逆条件数(最小特征值跟最大特征值的比例)。如果 src 是奇异的，那么 cvInvert()在 SVD 模式中将进行伪逆计算。

cvMahalonobis

```
CvSize cvMahalonobis(  
    const CvArr* vec1,  
    const CvArr* vec2,  
    CvArr*      mat  
) ;
```

Mahalonobis 距离(Mahal)被定义为一点和高斯分布中心之间的向量距离，该距离使用给定分布的协方差矩阵的逆作为归一化标准。参见图 3-5。直观上，这是与基础统计学中的标准分数(Z-score)类似，某一点到分布中心的距离是以该分布的方差作为单位。马氏距离则是该思路在高维空间中的推广。

cvMahalonobis()计算的公式如下：

$$r_{\text{Mahalonobis}} = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

假设向量 vec1 对应 \mathbf{x} 点，向量 vec2 是分布的均值^①。 mat 是协方差矩阵的逆。

实际上，这个协方差矩阵通常用 cvCalcCovarMatrix()(前面所述)来进行计算，然后用 cvInvert()来求逆。使用 SV_SVD 方法求逆是良好的程序设计习惯，因为其中一个特征值为 0 的分布这种情况在所难免！

① 其实，Mahalonobis 距离可定义于任意两个向量之间；在任何可能的情况下，向量 vec2 减去向量 vec1。在 cvMahalonobis()中的 mat 和协方差矩阵的逆之间也没有任何固定的联系；任何指标都可以适当的施加在这里。

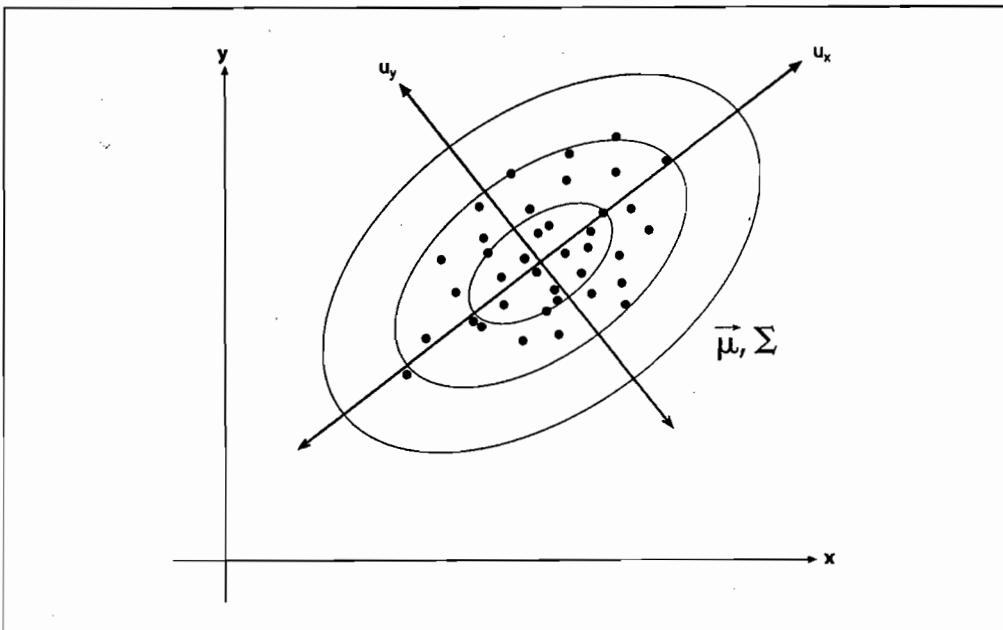


图 3-5：数据在 2D 空间分布，3 个叠加在一起的椭圆分别对应到分布中心的马氏距离为 1.0, 2.0 和 3.0 所有点

cvMax 和 cvMaxS

```

void cvMax(
    const CvArr* src1,
    const CvArr* src2,
    CvArr*      dst
);
void cvMaxS(
    const CvArr* src,
    double       value,
    CvArr*      dst
);

```

【66】

cvMax() 计算数组 src1 和 src2 中相对应的每像素一对中的最大值。而 cvMaxS(), 将数组 src 与常量参数 value 进行比较。通常, 如果 mask 非空, 那么只有与非 0 参数相对应的 dst 中的元素参与计算。

cvMerge

```
void cvMerge(
    const CvArr* src0,
    const CvArr* src1,
    const CvArr* src2,
    const CvArr* src3,
    CvArr* dst
);
```

【67】

cvMerge()是 cvSplit()的逆运算。数组 src0, src1, src2, 和 src3 将被合并到数组 dst 中。当然, dst 应该与源数组具有相同的数据类型和尺寸, 但它可以有两个, 三个或四个通道。未使用的源图像参数可设置为 NULL。

cvMin 和 cvMinS

```
void cvMin(
    const CvArr* src1,
    const CvArr* src2,
    CvArr* dst
);
void cvMinS(
    const CvArr* src,
    double value,
    CvArr* dst
);
```

cvMin()计算数组 src1 和 src2 中相对应的每一对像素中的最小值。而 cvMaxS(), 将数组 src 与常量标量 value 进行比较。同样的, 如果 mask 非空的话, 那么只有与 mask 的非 0 参数相对应的 dst 中的元素进行计算。

cvMinMaxLoc

```
void cvMinMaxLoc(
    const CvArr* arr,
    double*      min_val,
```

```
    double*      max_val,
    CvPoint*    min_loc = NULL,
    CvPoint*    max_loc = NULL,
    const CvArr* mask   = NULL
);
```

该例程找出数组 arr 中的最大值和最小值，并且(有选择性地)返回它们的地址。计算出的最大值和最小值赋值给 max_val 和 min_val。或者，如果极值的位置参数非空，那极值的位置便会写入 min_loc 和 max_loc。

通常，如果参数 mask 非空，那么只有图像 arr 中与参数 mask 中的非零的像素相对应的部分才被考虑。cvMinMaxLoc()例程仅仅处理单通道数组，如果有一个多通道的数组，则应该使用 cvSetCOI() 来对某个特定通道进行设置。

cvMul

```
void cvMul(
    const CvArr* src1,
    const CvArr* src2,
    CvArr* dst,
    double scale=1
);
```

【68】

cvMul() 是一个简单的乘法函数。它将 src1 中的元素与 src2 中相对应的元素相乘，然后把结果赋给 dst。如果 mask 是空，那么与其中 0 元素相对应的 dst 元素都不会因此操作而改变。OpenCV 中没有函数 cvMulS()，因为该功能已经由函数 cvScale() 或 cvCvtScale() 提供。

除此之外，有一件事情要记住：cvMul() 执行的是元素之间的乘法。有时候，在进行矩阵相乘时，可能会错误使用 cvMul()，但这无法奏效；记住，cvGEMM() 才是处理矩阵乘法的函数，而不是 cvMul()。

cvNot

```
void cvNot(
    const CvArr* src,
```

```
CvArr* dst  
);
```

函数 cvNot() 会将 src 中的每一个元素的每一位取反，然后把结果赋给 dst。因此，一个值为 0x00 的 8 位图像将被映射到 0xff，而值为 0x83 的图像将被映射到 0x7c。

cvNorm

```
double cvNorm(  
    const CvArr* arr1,  
    const CvArr* arr2 = NULL,  
    int norm_type = CV_L2,  
    const CvArr* mask = NULL  
);
```

这一函数可用于计算一个数组的各种范数，当为该函数提供了两个数组作为参数时，可选用各种不同的公式来计算相对的距离。在前一种情况下，计算的范数如表 3-9 所示。

表 3-9：当 arr2=NULL 时，对于不同的 norm_type 由 cvNorm() 计算范数的公式

norm_type	结果
CV_C	$\ \text{arr} \ _c = \max_{x,y} \text{abs}(\text{arr1}_{x,y})$
CV_L1	$\ \text{arr} \ _{L1} = \sum_{x,y} \text{abs}(\text{arr1}_{x,y})$
CV_L2	$\ \text{arr} \ _{L2} = \sqrt{\sum_{x,y} \text{arr1}_{x,y}^2}$

如果第二个数组参数 arr2 非空，那么范数的计算将使用不同的公式，就像两个数组之间的距离。^①前三种情况的计算公式如表 3-10 所示，这些范数是绝对范数；在后三个情况下，将会根据第二个数组 arr2 的幅度进行重新调整。【69~70】

① 至少在计算 L2 范数的情况下，在与图像素数相同的空间中，差分范数可理解为欧氏距离。

表 3-10: arr2 非空, 且 norm_type 不同值时函数 cvNorm()计算范数的计算公式

norm_type	结果
CV_C	$\ \text{arr1} - \text{arr2} \ _c = \max_{x,y} \text{arr1}_{x,y} - \text{arr2}_{x,y} $
CV_L1	$\ \text{arr1} - \text{arr2} \ _{L1} = \sum_{x,y} \text{arr1}_{x,y} - \text{arr2}_{x,y} $
CV_L2	$\ \text{arr1} - \text{arr2} \ _{L2} = \sqrt{\sum_{x,y} (\text{arr1}_{x,y} - \text{arr2}_{x,y})^2}$
CV_RELATIVE_C	$\frac{\ \text{arr1} - \text{arr2} \ _c}{\ \text{arr2} \ _c}$
CV_RELATIVE_L1	$\frac{\ \text{arr1} - \text{arr2} \ _{L1}}{\ \text{arr2} \ _{L1}}$
CV_RELATIVE_L2	$\frac{\ \text{arr1} - \text{arr2} \ _{L2}}{\ \text{arr2} \ _{L2}}$

在所有情况下, arr1 和 arr2 必须具有相同的大小和通道数。当通道数大于 1 时, 将会对所有通道一起计算范数(即是说, 在表 3-9 和表 3-10 中, 不仅是针对 x 和 y , 也针对通道数求和)。

cvNormalize

```
cvNormalize(
    const CvArr*     src,
    CvArr*           dst,
    double            a          = 1.0,
    double            b          = 0.0,
    int               norm_type = CV_L2,
    const CvArr*     mask      = NULL
);
```

与许多 OpenCV 函数一样, cvNormalize() 的功能比表面看更多。根据 norm_type 的不同值, 图像 src 将被规范化, 或者以其他方式映射到 dst 的一个特定的范围内。表 3-11 列举了 norm_type 可能出现的值。

表 3-11: 函数 cvNormalize() 的参数 norm_type 可能的值

norm_type	结果
CV_C	$\ \text{arr1} \ _c = \max_{dst} \text{arr1}_{x,y} = a$
CV_L1	$\ \text{arr1} \ _{L1} = \sum_{dst} \text{arr1}_{x,y} = a$

续表

norm_type	结果
CV_L2	$\ arr1\ _{l_2} = \sum_{x,y} /^2 = a$
CV_MINMAX	映射到 [a, b] 的范围内

【70】

计算 C 范数时，数组 src 将被进行比例变标，使其中绝对值最大的值等于 a。当计算 L1 范数或 L2 范数时，该数组也将被缩放，如使其范数为 a。如果 norm_type 的值设置为 CV_MINMAX，那么将会对数组的所有的值进行转化，使它们线性映射到 a 和 b 之间(包括 a 和 b)。

与以前一样，如果参数 mask 非空，那么只有与掩码非 0 值对应的像素会对范数的计算有贡献，并且只有那些像素会被 cvNormalize() 改变。

cvOr 和 cvOrS

```

void cvOr(
    const CvArr*      src1,
    const CvArr*      src2,
    CvArr*            dst,
    const CvArr*      mask=NULL
);
void cvOrS(
    const CvArr*      src,
    CvScalar          value,
    CvArr*            dst,
    const CvArr*      mask = NULL
);

```

这两个函数将对数组 src1 进行按位或计算。在函数 cvOr() 中，dst 中的每一个元素都是由 src1 和 src2 中相对应的元素按位做或运算的结果。在 cvOrS() 函数中，将对 src 和常量 value 进行或运算。像往常一样，如果 mask 非空，则只计算 dst 中与 mask 中非 0 元素对应的元素。

该函数支持所有的数据类型，但在 cvOr() 中，src1 和 src2 必须有相同的数据类型。如果数组元素是浮点类型，则使用浮点按位表示形式。

cvReduce

```
CvSize cvReduce(
    const CvArr*     src,
    CvArr*          dst,
    int              dim,
    int              op = CV_REDUCE_SUM
);
```

约简是指使用一些 `op` 所代表的组合规则，对输入的矩阵 `src` 的每一行(或列)进行系统的转化，使之成为成向量 `dst`，直到只剩一行(或列)为止(见表 3-12)。^①参数 `op` 决定如何进行约简，总结如表 3-13 所示。

【71】

表 3-12：参数 `op` 在 `cvReduce()` 中所代表的转化操作

<code>op</code> 的值	结果
<code>CV_REDUCE_SUM</code>	计算所有向量的总和
<code>CV_REDUCE_AVG</code>	计算所有向量的平均值
<code>CV_REDUCE_MAX</code>	计算所有向量中的最大值
<code>CV_REDUCE_MIN</code>	计算所有向量中的最小值

表 3-13：参数 `dim` 在 `cvReduce()` 中控制转化的方向

<code>dim</code> 的值	结果
+1	合并成一行
0	合并成一列
-1	转化成对应的 <code>dst</code>

`cvReduce()` 支持浮点型的多通道数组。它也允许在 `dst` 中使用比 `src` 更高精度的数据类型。这关键在于要有正确的 `CV_REDUCE_SUM` 和 `CV_REDUCE_AVG` 参数，否则那里可能有溢出和累积问题。

① 从严格意义上来说，求平均并不是一个真正的约简方法。OpenCV 从更实用的角度定义约简，因此在函数 `cvReduce` 中包括这一有用的功能。

cvRepeat

```
void cvRepeat(
    const CvArr*     src,
    CvArr*          dst
);
```

这一函数是将 `src` 的内容复制到 `dst` 中，重复多次，直到 `dst` 没有多余的空间。具体而言，`dst` 相对于 `src` 可以是任何大小。它可能比 `src` 大或小，它们在大小和维数之间不需要有任何的数值关系。

cvScale

```
void cvScale(
    const CvArr*     src,
    CvArr*          dst,
    double           scale
);
```

从宏观上讲，函数 `cvScale()` 实际上是 `cvConvertScale()` 的一个宏，它会将 `shift` 参数设置为 0.0。因此，它可以用来重新调整数组的内容，并且可以将参数从一种数据类型转换为另一种。

cvSet 和 cvSetZero

```
void cvSet(
    CvArr*          arr,
    CvScalar        value,
    const CvArr*    mask   = NULL
);
```

【72】

这些函数能将数组的所有通道的所有值设置为指定的参数 `value`。该 `cvSet()` 函数接受一个可选的参数：如果提供参数，那么只有那些与参数 `mask` 中非 0 值对应的像素将被设置为指定的值。函数 `cvSetZero()` 仅仅是 `cvSet(0.0)` 别名。

cvSetIdentity

```
void cvSetIdentity( CvArr* arr );
```

cvSetIdentity()将会把数组中除了行数与列数相等以外的所有元素的值都设置为 0；行数与列数相等的元素的值都设置为 1。cvSetIdentity()支持所有数据类型，甚至不要求数组的行数与列数相等。

cvSolve

```
int cvSolve(
    const CvArr*      src1,
    const CvArr*      src2,
    CvArr*            dst,
    int                method = CV_LU
);
```

基于 cvInvert() 函数 cvSolve() 为求解线性方程组提供了一条捷径。它的计算公式如下：

$$C = \operatorname{argmin}_X \|A \cdot X - B\|$$

其中 **A** 是一个由 src1 指定的方阵，**B** 是向量 src2，然后 **C** 是由 cvSolve() 计算的结果，目标是寻找一个最优的向量 **X**。最优结果向量 **X** 将返回给 **dst**。cvInvert() 支持同样的方法(前述)；不过只支持浮点类型的数据。该函数将会返回一个整型值，当返回的值是一个非 0 值的话，这表明它能够找到一个解。

应当指出的是，cvSolve() 可以用来解决超定的线性方程组。超定系统将使用所谓的伪逆方法进行解决，它是使用 SVD 方法找到方程组的最小二乘解的。

cvSplit

```
void cvSplit(
    const CvArr*      src,
    CvArr*            dst0,
    CvArr*            dst1,
    CvArr*            dst2,
```

```
    CvArr*          dst3  
);
```

有些时候处理多通道图像时不是很方便。在这种情况下，可以利用 cvSplit() 分别复制每个通道到多个单通道图像。如果需要，cvSplit() 函数将复制 src 的各个通道到图像 dst0, dst1, dst2 和 dst3 中。目标图像必须与源图像在大小和数据类型上相匹配，当然也应该是单通道的图像。

如果源图像少于 4 个通道(这种情况经常出现)，那么传递给 cvSplit() 的不必要的目标参数可设置为 NULL。【73】

cvSub, cvSubS 和 cvSubRS

```
void cvSub(  
    const CvArr*    src1,  
    const CvArr*    src2,  
    CvArr*          dst,  
    const CvArr*    mask = NULL  
);  
void cvSubS(  
    const CvArr*    src,  
    CvScalar        value,  
    CvArr*          dst,  
    const CvArr*    mask = NULL  
);  
void cvSubRS(  
    const CvArr*    src,  
    CvScalar        value,  
    CvArr*          dst,  
    const CvArr*    mask = NULL  
);
```

cvSub() 是一个简单的减法函数，它对数组 src2 和 src1 对应的元素进行减法运算，然后把结果赋给 dst。如果数组 mask 非空，那么 dst 中元素对应位置的 mask 中的 0 元素不会因此而改变。相关的函数 cvSubS() 执行相类似的功能，但它会对 src 的每一个元素减去一个常量 value。函数 cvSubRS() 的功能和 cvSubS() 相似，但不是 src 的每个元素减去一个常量，而是常量减去的 src 中的每一元素。

cvSum

```
CvScalar cvSum(
    CvArr* arr
);
```

`cvSum()`计算数组 `arr` 各个通道的所有的像素的总和。注意，函数的返回类型是 `CvScalar`，这意味着 `cvSum()` 提供多通道数组计算。在这种情况下，每个通道的和都会赋给类型为 `CvScalar` 的返回值中相应的分量。

cvSVD

```
void cvSVD(
    CvArr* A,
    CvArr* W,
    CvArr* U = NULL,
    CvArr* V = NULL,
    int flags = 0
);
```

奇异值分解(SVD)是将一个 $m \times n$ 的矩阵 **A** 按如下公式分解：

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T$$

其中，**W** 是一个对角矩阵，**U** 和 **V** 分别是 $m \times m$ 和 $n \times n$ 的酉矩阵。当然，矩阵 **W** 也是一个 $m \times n$ 的矩阵，所以在里面的“对角线”是指任何行数和列数不相等的位置的元素值一定是 0。因为 **W** 必须是对角矩阵，OpenCV 允许它表示为一个 $m \times n$ 阶矩阵或一个 $n \times 1$ 向量(在这种情况下，该向量将只包含对角线上的“奇异”值)。

对于函数 `cvSVD()` 来说，**U** 和 **V** 是可选参数，如果它们的值设置为 `NULL`，则不会返回它们的内容。最后的参数 `flags` 可以是表 3-14 所示三个选项中任何一个或全部(视情况进行布尔型或计算合并)。

表 3-14: cvSVD()中 flags 参数的取值

参数	结果
CV_SVD MODIFY_A	允许改变矩阵 A
CV_SVD_U_T	返回 U^T 而不是 U
CV_SVD_V_T	返回 V^T 而不是 V

cvSVBkSb

```
void cvSVBkSb(
    const CvArr* W,
    const CvArr* U,
    const CvArr* V,
    const CvArr* B,
    CvArr* X,
    int     flags = 0
);
```

这个函数一般不会被直接调用。与刚才所描述的 cvSVD()一起，本函数构成了基于 SVD 的方法 cvInvert() 和 cvSolve() 的基础。也就是说，如果你想自己实现矩阵求逆，可用这两个函数(这可以为你节省在 cvInvert() 或 cvSolve() 中为临时矩阵分配的一大堆内存空间)。

【75】

函数 cvSVBkSb() 对矩阵 A 进行反向替代计算，A 已分解为矩阵 U, W 和 V(即 SVD)的结构中描述出来。矩阵 X 的结果可由如下公式计算得出：

$$X = V \cdot W^* \cdot U^T \cdot B$$

矩阵 B 是可选的，如果设置为 NULL，它将会被忽略。当 $\lambda_i \geq \varepsilon$ 时矩阵 W* 中的对角线元素定义如下：

$$\lambda'_i = \lambda_i^{-1}$$

ε 这个值是一个奇异性阈值，一个非常小的数值，通常与 W 的对角线元素的总和成正比(即 $\varepsilon \propto \sum_i \lambda_i$)。

cvTrace

```
CvScalar cvTrace( const CvArr* mat );
```

矩阵的迹是对角线元素的总和。在 OpenCV 中，该功能在函数 `cvGetDiag()` 基础上实现，因此输入的数组不需要是方阵。同样支持多通道数组，但是数组 `mat` 必须是浮点类型。

cvTranspose 与 cvT

```
void cvTranspose(
    const CvArr* src,
    CvArr*       dst
);
```

`cvTranspose()` 将 `src` 中每一个元素的值复制到 `dst` 中行号与列号相调换的位置上。这个函数不支持多通道数组；然而，如果你用两通道数组表示复数，那么记住一点：`cvTranspose()` 不执行复共轭（依靠函数 `cvXorS()` 是实现该功能的一个快速方法，它可以直接翻转数组中虚数部分中的符号位）。宏 `cvT()` 是函数 `cvTranspose()` 的缩写。

cvXor 和 cvXorS

```
void cvXor(
    const CvArr* src1,
    const CvArr* src2,
    CvArr*       dst,
    const CvArr* mask=NULL
);
void cvXorS(
    const CvArr* src,
    CvScalar     value,
    CvArr*       dst,
    const CvArr* mask=NULL
);
```

【76】

这两个函数在数组 `src1` 上按位进行异或(XOR)运算。在函数 `cvXor()` 中，`dst` 的每个元素是由 `src1` 和 `src2` 中对应的元素按位进行异或运算所得到的。在函数 `cvXorS()` 中，是与常量 `value` 进行按位异或运算。再次说明，如果参数 `mask` 非空，则只计算与 `mask` 中非 0 值相对应的 `dst` 元素。

计算支持所有的数据类型，但 `src1` 和 `src2` 在函数 `cvXor()` 中必须是相同的数据类型。如果数组的元素是浮点类型的，那么使用浮点数的二进制表示。

cvZero

```
void cvZero( CvArr* arr );
```

这个函数会将数组中的所有通道的所有元素的值都设置为 0。

绘图

我们经常需要绘制图像或者在已有的图像上方绘制一些图形。为此，OpenCV 提供了一系列的函数帮助我们绘制直线、方形和圆形等。

直线

`cvLine()` 是绘图函数中最简单的，只需用 Bresenham 算法[Bresenham65]画一条线：

```
void cvLine(
    CvArr*      array,
    CvPoint     pt1,
    CvPoint     pt2,
    CvScalar   color,
    int        thickness = 1,
    int        connectivity = 8
);
```

`cvLine()` 函数中的第一个属性是 `CvArr*`。在这里，它一般为一个图像类型的指针 `IplImage*`。随后两个 `CvPoint` 是一种简单数据结构，它只包括整型变量 `x` 和 `y`。我们可以用 `CvPoint(int x, int y)` 函数快速地构造一个 `CvPoint` 类型的变量，这样可以方便地把两个整型变量值赋给 `CvPoint` 数据结构。

下一个属性是 CvScalar 类型的颜色变量。CvScalar 也是一种数据结构，定义如下所示：

```
typedef struct {
    double val[4];
} CvScalar;
```

可以看出，这种结构只是四个双精度浮点型变量的集合。在这里，前三个分别代表红、绿、蓝通道；没有用到第四个(它只在适当的时候用于 alpha 通道)。一个常用的便捷宏指令是 CV_RGB(*r*, *g*, *b*)，该指令采用三个数字作为参数并将其封装到 CvScalar。

接下来的两个属性是可选的。*thickness* 是线的粗细(像素)，*connectivity* 被设为反走样模式，默认值为“8 连通”，这种是较为平滑不会走样的线型。也可以设置为“4 连通”，这样的话，斜线会产生重叠以致看上去过于粗重，不过画起来速度要快得多。

cvRectangle()和 cvLine()几乎同样便捷。cvRectangle()用于画矩形。除了没有 connectivity 参数，它和 cvLine()的其他参数都是一样的。因为由此产生的矩形总是平行与 *X* 和 *Y* 轴。利用 cvRectangle()，我们只需给出两个对顶点，OpenCV 便于画出一个矩形。

```
void cvRectangle(
    CvArr* array,
    CvPoint pt1,
    CvPoint pt2,
    CvScalar color,
    int thickness = 1
);
```

圆形和椭圆

画圆同样简单，其参数与前相同。

```
void cvCircle (
    CvArr* array,
    CvPoint center,
    int radius,
    CvScalar color,
    int thickness = 1,
```

```
    int      connectivity = 8  
);
```

对圆形和矩阵等很多封闭图形来说，thickness 参数也可以设置为 CV_FILL，其值是-1，其结果是使用与边一样的颜色填充圆内部。

椭圆函数比 cvCircle() 略微复杂一些：

```
void cvEllipse(  
    CvArr*    img,  
    CvPoint   center,  
    CvSize    axes,  
    double    angle,  
    double    start_angle,  
    double    end_angle,  
    CvScalar  color,  
    int       thickness = 1,  
    int       line_type = 8  
);
```

【79】

这里，主要的新成员是 axes 属性，其类型为 CvSize。CvSize 函数与 CvPoint 和 CvScalar 非常相似；这是一个仅包含宽度和高度的简单结构。同 CvPoint 和 CvScalar 一样，CvSize 也有一个简单的构造函数 cvSize(int height, int width)，在需要的时候返回一个 CvSize 数据。在这种情况下，height 和 width 参数分别代表椭圆的长短半轴长。

angle 是指偏离主轴的角度，从 X 轴算起，逆时针方向为正。同样，start_angle 和 end_angle 表示弧线开始和结束位置的角度。因此，一个完整的椭圆必须分别将这两个值分别设为 0° 和 360°。

使用外接矩形是描述椭圆绘制的另一种方法：

```
void cvEllipseBox(  
    CvArr*    img,  
    CvBox2D   box,  
    CvScalar  color,  
    int       thickness = 1,  
    int       line_type = 8,  
    int       shift     = 0  
);
```

这里用到 OpenCV 的另一个结构 CvBox2D:

```
typedef struct {
    CvPoint2D32f center;
    CvSize2D32f size;
    float angle;
} CvBox2D;
```

CvPoint2D32f 是 CvPoint 的浮点形式，同时 CvSize2D32f 也是 CvSize 的浮点形式。这些，连同倾斜角度，可以有效地描述椭圆的外接矩形。

多边形

最后，我们有一系列绘制多边形的函数。

```
void cvFillPoly(
    CvArr* img,
    CvPoint** pts,
    int* npts,
    int contours,
    CvScalar color,
    int line_type = 8
);

void cvFillConvexPoly(
    CvArr* img,
    CvPoint* pts,
    int npts,
    CvScalar color,
    int line_type = 8
);

void cvPolyLine(
    CvArr* img,
    CvPoint** pts,
    int* npts,
    int contours,
    int is_closed,
    CvScalar color,
    int thickness = 1,
```

```
    int          line_type = 8  
};
```

【79~80】

上述三种方法依据同一思路又略有不同，其主要区别是如何描述点。

在 `cvFillPoly()` 中，点是由 `CvPoint` 数组提供的。它允许 `cvFillPoly()` 在一次调用中绘制多个多边形。同样地，`npts` 是由记数点构成的数组，与多边形对应。如果把变量 `is_closed` 设为 `true`，那么下一个多边形的第一个线段就会从上一多边形最后一点开始。`cvFillPoly()` 很稳定，可以处理自相交多边形，有孔的多边形等复杂问题。然而不幸的是，函数运行起来相对缓慢。

`cvFillConvexPoly()` 和 `cvFillPoly()` 类似。不同的是，它一次只能画一个 多边形，而且只能画凸多边形^①。好处是，`cvFillConvexPoly()` 运行得更快。

第三个 `cvPolyLine()`，其参数与 `cvFillPoly()` 相同，但因为只需画出多边形的 边，不需处理相交情况。因此，这种函数运行速度远远超过 `cvFillPoly()`。

字体和文字

最后一种形式的绘图是绘制文字。当然，文字创建了一套自己的复杂格式，但是，在这类事情上，OpenCV 一如既往地更关心提供一个简单的“一招解决问题”的方案，这个方案只适用于一些简单应用，而不适用于一个稳定的和完整的应用(这将降低由其他库提供的功能)。

OpenCV 有一个主要的函数，叫 `cvPutText()`。这个函数可以在图像上输出一些 文本。参数 `text` 所指向的文本将打印到图像上，参数 `origin` 指定文本框左下角位 置，参数 `color` 指定文本颜色。

```
void cvPutText(  
    CvArr*      img,  
    const char*   text,  
    CvPoint      origin,  
    const CvFont* font,
```

① 严格来讲不是这样，它可以绘制并填充任何单调(monotone)多边形，单调多边形比凸 多边形范围更大一点。

```
CvScalar color  
);
```

【80~81】

总有一些琐事使我们的工作比预期复杂，此时是 CvFont 指针表现的机会了。

概括地说，获取 CvFont* 指针的方式就是调用函数 cvInitFont()。该函数接受一组参数配置一些用于屏幕输出的基本个特定字体。如果熟悉其他环境中的 API 程序，势必会觉得 cvInitFont 似曾相识，但只需更少的参数。

为了建立一个可以传值给 cvPutText() 的 CvFont，首先必须声明一个 CvFont 变量，然后把它传递给 cvInitFont()。

```
void cvInitFont(  
    CvFont* font,  
    int font_face,  
    double hscale,  
    double vscale,  
    double shear = 0,  
    int thickness = 1,  
    int line_type = 8  
) ;
```

观察本函数与其他相似函数的不同。正如工作在 OpenCV 环境下的 cvCreateImage()。调用 cvInitFont() 时，初始化一个已经准备好的 CvFont 结构(这意味着你创建了一个变量，并传给 cvInitFont() 函数一个指向新建的变量指针)，而不是像 cvCreateImage() 那样创建一个结构并返回指针。

font_face 参数列在表 3-15 中(效果在图 3-6 中画出)，它可与 CV_FONT_ITALIC 组合(通过布尔或操作)。

表 3-15：可用字体(全部可选变量)

标志名称	描述
CV_FONT_HERSHEY_SIMPLEX	正常尺寸 sanserif 字体
CV_FONT_HERSHEY_PLAIN	小尺寸 sanserif 字体
CV_FONT_HERSHEY_DUPLEX	正常尺寸 sanserif, 比 CV_FONT_HERSHEY_SIMPLEX 更复杂
CV_FONT_HERSHEY_COMPLEX	正常尺寸 serif, 比 CV_FONT_HERSHEY_DUPLEX 更复杂

续表

标志名称	描述
CV_FONT_HERSHEY_TRIPLEX	正常尺寸 serif, 比 CV_FONT_HERSHEY_COMPLEX 更复杂
CV_FONT_HERSHEY_COMPLEX_SMALL	小尺寸的
CV_FONT_HERSHEY_SCRIPT_SIMPLEX	CV_FONT_HERSHEY_COMPLEX 手写风格
CV_FONT_HERSHEY_SCRIPT_COMPLEX	比 CV_FONT_HERSHEY_SCRIPT_SIMPLEX 更复杂的风格

【81】



图 3-6：表 3-15 中的 8 个字体，绘制时设置 `hscale = vscale = 1.0`，且每行的垂直间距为 30 像素

`hscale` 和 `vscale` 只能设为 1.0 或 0.5。字体渲染时选择全高或半高(宽度同比缩放)，绘制效果与指定字体的基本定义有关。

参数 `shear` 创建斜体字，如果设置为 0.0，字体不倾斜。当设置为 1.0 时，字符倾斜范围接近 45 度。

参数 `thickness` 与 `Line_type` 的定义与其他绘图函数相同。

数据存储

OpenCV 提供了一种机制来序列化(serialize)和去序列化(de-serialize)其各种数据类型，可以从磁盘中按 YAML 或 XML 格式读/写。在第 4 章中，我们将专门介绍存储和调用常见的对象 `IplImage` 的函数(`cvSaveImage()`和`cvLoadImage()`)。此外，第 4 章将讨论读/写视频的特有函数：可以从文件或者摄影机中读取数据的函数`cvGrabFrame()`以及写操作函数`cvCreateVideoWriter()`和`cvWriteFrame()`。本小节将侧重于一般对象的永久存储：读/写矩阵、OpenCV 构造、配置与日志文件。

首先，我们从有效且简便的 OpenCV 矩阵的保存和读取功能函数开始。函数是`cvSave()`和`cvLoad()`。例 3-15 展示了如何保存和读取一个 5×5 的单位矩阵(对角线上是 1，其余地方都是 0)。

例 3-15：存储和读取 CvMat

```
CvMat A = cvMat( 5, 5, CV_32F, the_matrix_data );

cvSave( "my_matrix.xml", &A );

...
// to load it then in some other program use ...
CvMat* A1 = (CvMat*) cvLoad( "my_matrix.xml" );
```

CxCore 参考手册中有整节内容都在讨论数据存储。首先要知道，在 OpenCV 中，一般的数据存储要先创建一个 `CvFileStorage` 结构(如例 3-16)所示，该结构将内存对象存储在一个树形结构中。然后通过使用 `CV_STORAGE_READ` 参数的 `cvOpenFileStorage()` 从磁盘读取数据，创建填充该结构，也可以通过使用 `CV_STORAGE_WRITE` 的 `cvOpenFileStorage()` 创建并打开 `CvFileStorage` 对象，而后使用适当的数据存储函数来填充它。在磁盘上，数据的存储格式为 XML 或者 YAML。

例 3-16：CvFileStorage 结构，数据通过 CxCore 数据存储函数访问

```
typedef struct CvFileStorage
{
    ...
    // hidden fields
} CvFileStorage;
```

`CvFileStorage` 树内部的数据是一个层次化的数据集合，包括标量、CxCore 对象、

(矩阵、序列和图)以及用户定义的对象。

假如有一个配置文件或日志文件。配置文件告诉我们视频有多少帧(10)，画面大小(320×240)并且将应用一个 3×3 的色彩转换矩阵。例 3-17 展示了如何从磁盘中调出 cfg.xml 文件。

例 3-17：往磁盘上写一个配置文件 cfg.xml

```
CvFileStorage* fs = cvOpenFileStorage(
    "cfg.xml",
    0,
    CV_STORAGE_WRITE
);
cvWriteInt( fs, "frame_count", 10 );
cvStartWriteStruct( fs, "frame_size", CV_NODE_SEQ );
cvWriteInt( fs, 0, 320 );
cvWriteInt( fs, 0, 200 );
cvEndWriteStruct(fs);
cvWrite( fs, "color_cvt_matrix", cmatrix );
cvReleaseFileStorage( &fs );
```

请留意这个例子中的一些关键函数。我们可以定义一个整型变量通过 cvWriteInt() 向结构中写数据。我们也可以使用 cvStartWriteStruct() 来创建任意一个可以任选一个名称(如果无名称请输入 0 或 NULL)的结构。这个结构有两个未命名的整型变量，使用 cvEndWriteStruct() 结束编写结构。如果有更多的结构体，我们用相似的方法来解决；这种结构可以进行任意深度的嵌套。最后，我们使用 cvWrite() 编写处色彩转换矩阵。将这个相对复杂的矩阵程序与例 3-15 中简单的 cvSave() 程序进行对比。便会发现 cvSave() 是 cvWrite() 在只保存一个矩阵时的快捷方式。当写完数据后，使用 cvReleaseFileStorage() 释放 CvFileStorage 句柄。例 3-18 显示了 XML 格式的输出内容。

例 3-18：磁盘中的 cfg.xml 文件

```
<?xml version="1.0"?>
<opencv_storage>
<frame_count>10</frame_count>
<frame_size>320 200</frame_size>
<color_cvt_matrix type_id="opencv-matrix">
    <rows>3</rows> <cols>3</cols>
    <dt>f</dt>
    <data>...</data></color_cvt_matrix>
</opencv_storage>
```

我们将会在例 3-19 中将这个配置文件读入。

例 3-19：磁盘中的 cfg.xml 文件

```
CvFileStorage* fs = cvOpenFileStorage(
    "cfg.xml",
    0,
    CV_STORAGE_READ
);

int frame_count = cvReadIntByName(
    fs,
    0,
    "frame_count",
    5 /* default value */
);

CvSeq* s = cvGetFileNodeByName(fs, 0, "frame_size")->data.seq;

int frame_width = cvReadInt(
    (CvFileNode*)cvGetSeqElem(s, 0)
);

int frame_height = cvReadInt(
    (CvFileNode*)cvGetSeqElem(s, 1)
);

CvMat* color_cvt_matrix = (CvMat*) cvReadByName(
    fs,
    0,
    "color_cvt_matrix"
);

cvReleaseFileStorage( &fs );
```

在阅读时，我们像例 3-19 中那样用 `cvOpenFileStorage()` 打开 XML 配置文件。然后用 `cvReadIntByName()` 来读取 `frame_count`，如果有没有读到的数，则赋一个默认值。在这个例子中默认的值是 5。然后使用 `cvGetFileNodeByName()` 得到结构体 `frame_size`。在这里我们用 `cvReadInt()` 读两个无名称的整型数据。随后

使用 `cvReadByName()` 读出我们已经定义的色彩转换矩阵。^① 将本例与例 3-15 中的 `cvLoad()` 进行对比。如果我们只有一个矩阵要读取，那么可以使用 `cvLoad()`，但是如果矩阵是内嵌在一个较大的结构中，必须使用 `cvRead()`。最后，释放 `CvFileStorage` 结构。

数据函数存储与 `CvFileStorage` 结构相关的表单列在表 3-16 中。想了解更多细节，请查看 `CxCore` 手册。

表 3-16：数据存储函数

函数名称	描述
打开并释放	
<code>cvOpenFileStorage</code>	为读/写打开存储文件
<code>cvReleaseFileStorage</code>	释放存储的数据
写入	
<code>cvStartWriteStruct</code>	开始写入新的数据结构
<code>cvEndWriteStruct</code>	结束写入数据结构
<code>cvWriteInt</code>	写入整数型
<code>cvWriteReal</code>	写入浮点型
<code>cvWriteString</code>	写入字符串
<code>cvWriteComment</code>	写一个 XML 或 YAML 的注释字符串
<code>cvWrite</code>	写一个对象，例如 <code>CvMat</code>
<code>cvWriteRawData</code>	写入多个数值
<code>cvWriteTreeNode</code>	将文件节点写入另一个文件存储器
读取	
<code>cvGetRootTreeNode</code>	获取存储器最顶层的节点
<code>cvGetTreeNodeByName</code>	在映图或存储器中找到相应节点
<code>cvGetHashedKey</code>	为名称返回一个唯一的指针
<code>cvGetTreeNode</code>	在映图或文件存储器中找到节点
<code>cvGetTreeNodeName</code>	返回文件的节点名
<code>cvReadInt</code>	读取一个无名称的整数型
<code>cvReadIntByName</code>	读取一个有名称的整数型
<code>cvReadReal</code>	读取一个无名称的浮点型

① 我们还可以使用 `cvRead()` 来读取矩阵，但它只能在获取 `CvTreeNode()` 之后调用，比如使用 `cvGetTreeNodeByName()`。

续表

函数	功能描述
cvReadRealByName	读取一个有名称的浮点型
cvReadString	从文件节点中寻找字符串
cvReadStringByName	找到一个有名称的文件节点并返回它
cvRead	将对象解码并返回它的指针
cvReadByName	找到对象并解码
cvReadRawData	读取多个数值
cvStartReadRawData	初始化文件节点序列的读取
cvReadRawDataSlice	读取文件节点的内容

集成性能基元

Intel 公司有一个产品叫集成性能基元(Integrated Performance Primitives, IPP)库。这个库实际上是一个有着高性能内核的工具箱，它主要用于多媒体处理以及其他计算密集型应用，可发掘处理器架构的计算能力。(其他厂商的处理器也有类似的架构，只不过规模较小。)

就像第一章所探讨的，无论从软件层面还是公司内组织层面 OpenCV 都与 IPP 有着紧密的联系。最终，OpenCV 被设计成能够自动识别^①IPP 库，自动将性能较低的代码切换为 IPP 同功能的高性能代码。IPP 库允许 OpenCV 依靠它获得性能提升，IPP 依靠单指令多数据(SIMD)指令以及多核架构提升性能。

学会这些基础知识，我们就可以执行各种各样的基本任务。在本书随后的内容中，我们会发现 OpenCV 具有许多高级功能，几乎所有这些功能都可切换到 IPP 运行。图像处理经常需要对大量数据做同样的重复操作，许多是可并行处理的。因此如果任何利用并行处理方式(MMX, SSE, SSE2 等)的代码获得了巨大性能提升，您不必感到惊讶。

① 这个自动识别功能的先决条件是 IPP 的二进制目录必须在系统路径中。所以，在 Windows 系统中，如果 IPP 安装在目录 *C:/Program Files/Intel/IPP*，则应确保 *C:/Program Files/Intel/IPP/bin* 在系统路径中。

验证安装

用来检查 IPP 库是否已经正常安装并且检验运行是否正常的方法是使用函数 cvGetModuleInfo()，如例 3-20 所示。这个函数将检查当前 OpenCV 的版本和所有附加模块。

例 3-20：使用 cvGetModuleInfo() 检查 IPP

```
char* libraries;
char* modules;
cvGetModuleInfo( 0, &libraries, &modules );
printf("Libraries: %s/nModules: %s/n", libraries, modules );
```

例 3-20 中的代码将打印出描述已经安装的库和模块的文本。结果如下所示：

```
Libraries cxcore: 1.0.0
Modules: ippcv20.dll,ippi20.dll,ipps20.dll,ippvm20.dll
```

此处输出的模块列表就是 OpenCV 从 IPP 库中调用的模块的信息。实际上这些模块是更底层的 CPU 库函数的封装。它的运行原理将远远超出这本书的范围，但是如果在 modules 字符串中看到了 IPP 库，那么你会感到很自信，因为所有的工作都在按您预期的进行。当然你可以运用这个信息来确认 IPP 在您的系统中能正常运行。你也许会用它来检查 IPP 是否已经安装，并根据 IPP 存在与否来自动调整代码。

小结

本章介绍了我们经常遇到的一些基本数据结构。具体说来，我们了解了 OpenCV 矩阵结构和最重要的 OpenCV 图像结构 IplImage。经过对两者的仔细研究，我们得出一个结论：矩阵结构和图像结构非常相似，如果某函数可使用 CvMat，那也可使用 IplImage，反之亦然。

练习

在下面的练习中，有些函数细节在本书中未介绍，可能需要参考与 OpenCV 一起安装的或者网上 OpenCV Wiki 中的 CxCore 手册。

1. 找到并打开 `.../opencv/cxcore/include/cxtypes.h` 通读并找到可进行如下操作的函数。
 - a. 选取一个负的浮点数，取它的绝对值，四舍五入后，取它的极值。
 - b. 产生一些随机数。
 - c. 创建一个浮点型 `CvPoint2D32f`，并转换成一个整数型 `CvPoint`。
 - d. 将一个 `CvPoint` 转换成一个 `CvPoint2D32f`。
2. 下面这个练习是帮助掌握矩阵类型。创造一个三通道二维矩阵，字节类型，大小为 100×100 ，并设置所有数值为 0。
 - a. 在矩阵中使用 `void cvCircle(CvArr* img, CvPoint center, int radius, CvScalar color, int thickness=1, int line type=8, int shift=0)` 画一个圆。
 - b. 使用第 2 章所学的方法来显示这幅图像。
3. 创建一个拥有三个通道的二维字节类型矩阵，大小为 100×100 ，并将所有值赋为 0。通过函数 `cvPtr2D` 将指针指向中间的通道（“绿色”）。以(20,5)与(40,20)为顶点间画一个绿色的长方形。
4. 创建一个大小为 100×100 的三通道 RGB 图像。将它的元素全部置 0。使用指针算法以(20, 5)与(40, 20)为顶点绘制一个绿色平面。
5. 练习使用感兴趣区域(ROI)。创建一个 210×210 的单通道图像并将其归 0。在图像中使用 ROI 和 `cvSet()` 建立一个增长如金字塔状的数组。也就是说：外部边界为 0，下一个内部边界应该为 20，再下一个内部边界为 40 依此类推，直到最后内部值为 200；所有的边界应该为 10 个像素的宽度。最后显示这个图形。
6. 为一个图像创建多个图像头。读取一个大小至少为 100×100 的图像。另创建两个图像头并设置它们的 `origin`, `depth`, `nChannels` 和 `widthHeight` 属性同之前读取的图像一样。在新的图像头中，设置宽度为 20，高度为 30。最后，将 `imageData` 指针分别指向像素(5, 10)和(50, 60)像素位置。传递这两个新的图像头给 `cvNot()`。最后显示最初读取的图像，在那个大图像中应该有两个矩形，矩形内的值是原始值的求反值。
7. 使用 `cvCmp()` 创建一个掩码。加载一个真实的图像。使用 `cvSplit()` 将图像分割成红，绿，蓝三个单通道图像。
 - a. 找到并显示绿图。

- b. 克隆这个绿图两次(分别命名为 clone1 和 clone2)。
 - c. 求出这个绿色平面的最大值和最小值。
 - d. 将 clone1 的所有元素赋值为 `thresh=(unsigned char)((最大值-最小值)/2.0)`。
 - e. 将 clone2 所有元素赋值为 0, 然后调用函数 `cvCmp(green_image, clone1, clone2, CV_CMP_GE)`。现在 clone2 将是一个标识绿图中值超过 thresh 的掩码图像。
 - f. 最后, 使用 `cvSubS(green_image, thresh/2, green_image, clone2)` 函数并显示结果。
8. 创建一个结构, 结构中包含一个整数, 一个 CvPoint 和一个 CvRect; 称结构为 "my_struct"。
- a. 写两个函数 : `void write_my_struct(CvFileStorage * fs, const char * name, my_struct * ms)` 和 `void read_my_struct(CvFileStorage* fs, CvTreeNode* ms_node, my_struct* ms)`。用它们读/写 my_struct。
 - b. 创建一个元素为 my_struct 结构体且长度为 10 的数组, 并将数组写入磁盘和从磁盘读入内存。

细说 HighGUI

一个可移植的图形工具包

OpenCV 将与操作系统、文件系统和摄像机之类的硬件进行交互的一些函数纳入 HighGUI(*high-level graphical user interface*)库中。有了 HighGUI，我们可以方便地打开窗口、显示图像、读出或者写入图像相关的文件(包含图像与视频)、处理简单的鼠标、光标和键盘事件。我们也可以使用 HighGUI 创建其他一些很有用的控件，比如滑动条，并把它们加入窗口。如果对自己所使用的系统的图形用户界面非常熟悉，也许会认为 HighGUI 提供的很多功能是没有必要的，但即使如此，HighGUI 的跨平台性对你也会有很大的帮助。

我们最初的观点是，OpenCV 中的 HighGUI 可以分为 3 部分：硬件相关部分、文件部分以及图形用户界面部分^①。在仔细研究 HighGUI 之前，我们先用一些时间分别浏览一下每部分的内容。

硬件部分最主要的就是对于摄像机的操作，在大多操作系统下，与摄像机交互是一件很复杂并且很痛苦的工作。HighGUI 提供了一种从摄像机中获取图像的简单方法，所有繁琐的工作都在 HighGUI 内部完成了，这让我们很开心。

文件系统部分的主要工作是载入与保存图像文件。HighGUI 一个很好的特点就是

① 事实上，HighGUI 的结构实现与我们所表述的不尽相同，但是将 HighGUI 分为硬件相关部分、文件部分以及图形用户接口部分有助于我们更清晰地理解。HighGUI 函数实际被划分为“视频输入／输出”、“图像输入／输出”和“GUI 工具”三部分，分别在 cvcap*、grfmt* 和 window* 源文件中实现。

可以用与读取摄像机视频相同的方法读入视频文件。这使得我们可以省去处理从各种特定设备中读入数据的麻烦，而专心于我们感兴趣的代码部分。同样地，HighGUI 为我们提供了一对函数来读入与保存图像，这两个函数根据文件名的后缀，自动处理所有编码和解码工作。

【90】

HighGUI 的第三部分是窗口系统(或者称为 GUI)。HighGUI 提供了一些简单的函数用来打开窗口以及将图像显示在窗口中。它同时给我们提供了为窗口加入鼠标、键盘响应的方法。这些函数为我们快速建立一个简单的应用程序提供了很大的帮助。一种变通的方式，我们可以用滑动条实现切换功能^①。我们发现使用 HighGUI 可以实现很多实用的程序。

在本章后面的具体讲解中，我们不会对 HighGUI 的三个部分分别介绍；而是通过实现一些功能来讲解 HighGUI。在这种方式下，你会以最快的速度了解到如何使用 HighGUI。

创建窗口

首先，我们要做的是利用 HighGUI 将一幅图像显示到屏幕上。我们使用 cvNamedWindow() 来实现这个功能。这个函数接受两个参数，第一个参数用来表示新窗口的名称，这个名称显示在窗口的顶部，同时用作 HighGUI 中其他函数调用窗口的句柄。第二个参数是一个标志，用来表示是否需要使窗口大小自动适应读入的图像大小。下面是这个函数的定义：

```
int cvNamedWindow(
    const char* name,
    int         flags = CV_WINDOW_AUTOSIZE
);
```

需要留意的是参数 flags，到目前为止，惟一有效的设置是 0 或者保持默认设置 CV_WINDOW_AUTOSIZE。如果使用 CV_WINDOW_AUTOSIZE，HighGUI 会根据图像的大小调整窗口大小。这样，窗口大小会随着图像的载入而根据图像大小调整，用户没办法手动调整窗口大小。当然，如果不想窗口大小自动调整，也可以将参数值设置为 0，这样的话，用户就可以随意调整窗口的大小了。

① OpenCV 的 HighGUI 中没有包含一些类似于按钮的控件。一般的实现方法是用一个只有两个可选位置的滑动条来模拟按钮功能(后面会更详细介绍)。

当窗口被创建以后，我们通常是想加入一些东西到里面。不要着急，在做那些事情之前，我们先看看当不需要这些窗口时，如何释放它们。为了释放窗口，我们需要使用 `cvDestroyWindow()`，这个函数接收一个字符串参数，这个字符串是窗口创建时所指定的名字。

在 OpenCV 中，窗口根据名称来引用(操作系统独立的)而不是一些“不好好”的句柄。句柄与窗口名称之间的转换都由 HighGUI 在后台处理，我们不用为这些问题操心。

话虽如此，还是有些人担心 HighGUI 内部的处理，那没关系。HighGUI 提供了以下函数：

```
void* cvGetWindowHandle( const char* name );
const char* cvGetWindowName( void* window_handle );
```

【91】

这些函数允许我们在 OpenCV 所使用的窗口名称与各个窗口系统^①所使用的窗口句柄之间进行转换。

HighGUI 提供了 `cvResizeWindow()` 用来调整窗口的大小：

```
void cvResizeWindow(
    const char* name,
    int width,
    int height
);
```

这里在宽度与高度是以像素为单位的，指定了窗口中可以显示部分(这部分的大小可能才是真正关心的)的大小。

载入图像

为了在窗口中显示图像，我们需要了解如何从磁盘中载入图像。OpenCV 为我们提供了 `cvLoadImage()`，如下所示：

```
IplImage* cvLoadImage(
    const char* filename,
```

① 在 Win32 窗口系统下返回一个 HWND 句柄，在 Mac 窗口操作系统返回一个 Carbon WindowRef，在 X 窗口情况下，返回一个 Widget* 指针(例如 GtkWidget*)。

```
int iscolor = CV_LOAD_IMAGE_COLOR  
);
```

当打开一幅图像时，cvLoadImage()并不分析文件扩展名，而是通过分析图像文件的前几个字节来确定图像的编码格式。第二个参数 iscolor 有几个值可以选择。默认情况下，图像是以每个通道 8 位，3 个通道的形式被读入；可以通过设置 CV_LOAD_IMAGE_ANYDEPTH 来读入非 8 位的图像。默认情况下的通道为 3，因为参数 iscolor 的默认值是 CV_LOAD_IMAGE_COLOR，这意味着不管原始图像的通道数为多少，都将被转换为 3 个通道读入。相对于 CV_LOAD_IMAGE_COLOR，iscolor 也可以被设置成 CV_LOAD_IMAGE_GRAYSCALE 和 CV_LOAD_IMAGEANYCOLOR。类似于 CV_LOAD_IMAGE_COLOR 将读入图像强制转换为 3 个通道，CV_LOAD_IMAGE_GRAYSCALE 将读入图像强制转换为单通道。CV_LOAD_IMAGEANYCOLOR 则以保持原始图像通道数的方式读入。这样，为了读入 16 位的彩色图像，我们需要设置 iscolor 为 CV_LOAD_IMAGE_COLOR | CV_LOAD_IMAGE_ANYDEPTH。如果想读入数据与原始图像通道数以及位数保持一致，也可以使用 CV_LOAD_IMAGE_UNCHANGED。需要注意的是，当 cvLoadImage() 读入失败，并不会产生一个运行时错误，而是返回一个空指针。

与 cvLoadImage() 对应的函数是 cvSaveImage()，实现了保存图像功能，cvSaveImage() 有两个参数：

```
int cvSaveImage(  
    const char* filename,  
    const CvArr* image  
);
```

【92】

第一个参数表示文件名，其中后缀部分用来指定图像存储的编码格式。第二个参数指向要存储的图像数据。回忆前一章讲到的，CvArr 是一种 C 语言风格的，功能与面向对象语言中基类类似的结构。当你看到 CvArr* 时，你可以用 IplImage* 参数传入。对于大部分文件格式，cvSaveImage() 只能存储 8 位单通道或者 8 位 3 个通道格式的图像。新的文件格式像 PNG，TIFF 或者 JPEG2000 允许存储 16 位甚至浮点类型格式，同样也部分支持 4 个通道格式(BGRA)的图像。当存储成功时，返回 1，否则返回 0^①。

① 在一些操作系统中，存储文件时有可能会产生系统异常。一般情况下，都会返回 0 表示存储失败。

显示图像

准备工作做完了，现在让我们来做一些我们一直想做的事情，从磁盘中载入一张图像，并在窗口中显示出来，让我们来享受这一过程吧！为了完成这个功能，我们还需要了解一个简单的函数 `cvShowImage()`：

```
void cvShowImage(  
    const char* name,  
    const CvArr* image  
) ;
```

第一个参数用来指定用来显示图像的窗口，第二个参数指向需要显示的图像。

现在让我们用前面提到的一些函数完成一个简单的程序。这个程序通过命令行读入文件名，创建窗口并且将图像显示的窗口中。它包括注释和内存清理部分在内，它共有 25 行！

```
int main(int argc, char** argv)  
{  
  
    // Create a named window with the name of the file.  
    cvNamedWindow( argv[1], 1 );  
  
    // Load the image from the given file name.  
    IplImage* img = cvLoadImage( argv[1] );  
  
    // Show the image in the named window  
    cvShowImage( argv[1], img );  
  
    // Idle until the user hits the "Esc" key.  
    while( 1 ) {  
        if( cvWaitKey( 100 ) == 27 ) break;  
    }  
  
    // Clean up and don't be piggies  
    cvDestroyWindow( argv[1] );  
    cvReleaseImage( &img );  
    exit(0);  
}
```

【93 ~ 94】

方便起见，我们使用文件名来表示窗口名称。这样处理很好，因为 OpenCV 默认将窗口名称显示在窗口的顶部，这使得我们可以知道正在看的是哪一幅图像(见图 4-1)。这是很容易的一件事。

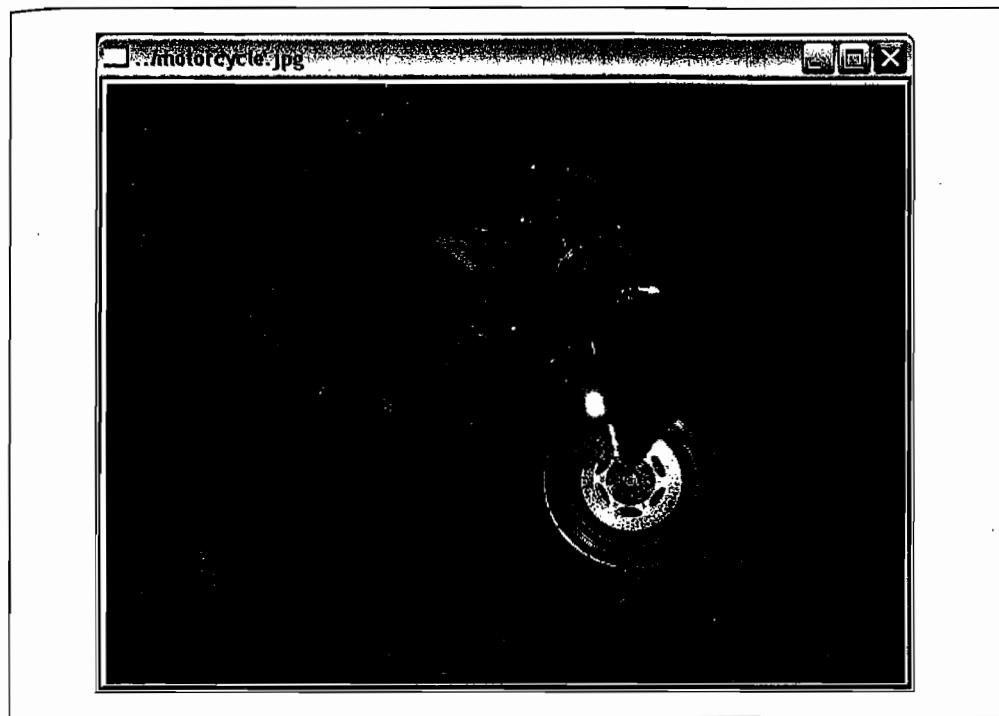


图 4-1：用 cvShowImage() 来实现的一个简单图像显示

在进入下一部分前，有一些其他窗口相关的函数需要了解，具体如下：

```
void cvMoveWindow( const char* name, int x, int y );
void cvDestroyAllWindows( void );
int cvStartWindowThread( void );
```

cvMoveWindow() 将窗口移动到其左上角为 x,y 的位置。

cvDestroyAllWindows() 是一个很有用的清理函数，用来关闭所有窗口并释放窗口相关的内存空间。

在 Linux 或者 Mac 操作系统中，cvStartWindowThread() 用来创建一个线程用来自动更新窗口以及处理其他窗口触发事件。返回值为 0 表示没有创建出线程，如果目前使用的 OpenCV 版本并不支持这个功能则返回 0。注意，如果没有创建一个独立的窗口线程，OpenCV 只会在明显式指定的时间里面处理用户交互(这需要程序

WaitKey

在前面的创建窗口例子的循环语句中，有一个新的函数我们前面没有看到过：cvWaitKey()。这个函数在一个给定的时间内(单位 ms)等待用户按键触发。如果在给定时间内用户按下了一个键，函数返回用户按下的键^①，否则，返回 0。具体使用如下：

```
while( 1 ) {
    if( cvWaitKey(100)==27 ) break;
}
```

在这个程序中，我们告诉 OpenCV 等待用户触发事件 100ms，如果在 100ms 内没有用户触发，则继续循环；如果有用户触发且按键 ASCII 码为 27(Esc 键)，则退出循环。这使得用户可以从容地仔细查看图像，直到按 Esc 键退出。

提到 cvWaitKey()，需要介绍的是 cvWaitKey() 可以接收 0 作为参数。在这种情况下，cvWaitKey() 会无限期地等待，直到用户触发一个按键。在前面的程序中，可以用 cvWaitKey(0) 来代替我们的实现。这两者实现的区别在显示一个视频的时候会体现出来，因为在视频播放过程中，我们希望播放下一帧，即使用户没有触发事件。

鼠标事件

现在我们已经可以显示一张图像给用户，我们现在希望给用户添加一些交互功能。由于我们在窗口环境下工作，而且我们已经知道如何使用 cvWaitKey() 来捕捉单次用户触发事件，我们下一步需要做的是如何捕捉以及响应鼠标事件。

① 细心的读者也许会问，返回用户触发的按键具体是返回什么？简单的回答就是“按键的 ASCII 码”，但实际上与操作系统有关，在 Win32 环境下，cvWaitKey() 等待一个 WM_CHAR 的消息类型，在接收消息以后返回消息的 wParam 参数值(wParam 并不一定是一个字符)。在类 UNIX 的操作系统下，cvWaitKey 使用 GTK，返回值是(event->keyval | (event->state<<16))，其中 event 是一个 GdkEventKey 结构。再一次说明，这并不一定是一个字符。state 包含 Shift、Control 键等信息；具体而言，当你希望获得一个大写的 Q，或者将 cvWaitKey() 的返回值强制截断成字符类型，或者将它与 0xff 求与。因为 shift 值将表示在高位里(例如组合键 Shift-Q 返回 0x10051)。

与键盘事件响应不同，鼠标事件响应采用回调函数的方式来处理。即，为了可以响应鼠标点击事件，首先必须创建一个回调函数，使鼠标点击事件发生时 OpenCV 可以调用这个函数。创建这个函数以后，需要在 OpenCV 中注册这个函数，以便特定窗口被触发鼠标事件以后，OpenCV 可以正确调用这个函数。 【95~96】

让我们从这个回调函数开始，回调函数 `callback` 可以是满足指定输入参数以及返回参数类型的任何函数。这里，我们必须清楚告诉回调函数触发的事件以及触发位置。函数还需要被告知，用户是否在触发鼠标事件时同时触发了 Shift 或者 Alt 等键。下面是回调函数必须符合的格式：

```
void CvMouseCallback(  
    int event,  
    int x,  
    int y,  
    int flags,  
    void* param  
) ;
```

现在，当回调函数被调用，OpenCV 会给函数传入合适的值。第一个参数 `event` 必须为表 4-1 中的一个值。

表 4-1：鼠标事件类型

事件名称	数值
CV_EVENT_MOUSEMOVE	0
CV_EVENT_LBUTTONDOWN	1
CV_EVENT_RBUTTONDOWN	2
CV_EVENT_MBUTTONDOWN	3
CV_EVENT_LBUTTONUP	4
CV_EVENT_RBUTTONUP	5
CV_EVENT_MBUTTONUP	6
CV_EVENT_LBUTTONDOWNDBLCLK	7
CV_EVENT_RBUTTONDOWNDBLCLK	8
CV_EVENT_MBUTTONDOWNDBLCLK	9

第二个以及第三个参数会被设置成事件发生时鼠标位置的 `x,y` 坐标值。值得指出的是，这些坐标代表窗口中图像的像素坐标，与窗口的大小没有关系。

第四个参数 `flags`，每一位指定了在事件发生时的不同状态。例如，`CV_EVENT_`

FLAG_SHIFTKEY 的值为 16(flags 的第五位为 1)，如果想知道 Shift 键是否被触发，我们可以用 flags 与位掩码(1<<4)求与。表 4-2 列出了所有的标志。

表 4-2：鼠标事件标志

标志名称	数值
CV_EVENT_FLAG_LBUTTON	1
CV_EVENT_FLAG_RBUTTON	2
CV_EVENT_FLAG_MBUTTON	4
CV_EVENT_FLAG_CTRLKEY	8
CV_EVENT_FLAG_SHIFTKEY	16
CV_EVENT_FLAG_ALTKEY	32

【96~97】

最后一个参数是一个 void 指针，可以用来以任何结构方式传递额外的参数信息。通常情况下，当回调函数为一个类的静态成员时。在这种情况下，可能需要传递这个类的指针以确定对哪一个类实例产生影响。

下面需要注册回调函数到 OpenCV 中，实现注册的函数是 cvSetMouseCallback()，该函数需要三个参数。

```
void cvSetMouseCallback(
    const char*      window_name,
    CvMouseCallback  on_mouse,
    void*            param     = NULL
);
```

第一个参数指定了回调函数需要注册到的窗口，也就是产生事件的窗口。只有在这个指定的窗口中触发的事件才会调用回调函数。第二个参数为回调函数。最后，第三个参数用来传递额外的信息给前面提到的 void* param 参数。

在例 4-1 中，我们实现了一个简单的程序，使得用户可以通过鼠标来画方形。函数 my_mouse_callback() 用来响应鼠标事件，并且根据 event 来确定给出的响应。

例 4-1：用鼠标在窗口中画方形的程序

```
// An example program in which the
// user can draw boxes on the screen.
//
#include <cv.h>
#include <highgui.h>
```

```

// Define our callback which we will install for
// mouse events.
//
void my_mouse_callback(
    int event, int x, int y, int flags, void* param
);

CvRect box;
bool drawing_box = false;

// A little subroutine to draw a box onto an image
//
void draw_box( IplImage* img, CvRect rect ) {
    cvRectangle (
        img,
        cvPoint(box.x,box.y),
        cvPoint(box.x+box.width,box.y+box.height),
        cvScalar(0xff,0x00,0x00) /* red */
    );
}

int main( int argc, char* argv[] ) {

    box = cvRect(-1,-1,0,0);

    IplImage* image = cvCreateImage(
        cvSize(200,200),
        IPL_DEPTH_8U,
        3
    );
    cvZero( image );
    IplImage* temp = cvCloneImage( image );

    cvNamedWindow( "Box Example" );

    // Here is the crucial moment that we actually install
    // the callback. Note that we set the value 'param' to
    // be the image we are working with so that the callback
    // will have the image to edit.
    //

    cvSetMouseCallback(

```

```

"Box Example",
my_mouse_callback,
(void*) image
);

// The main program loop. Here we copy the working image
// to the 'temp' image, and if the user is drawing, then
// put the currently contemplated box onto that temp image.
// display the temp image, and wait 15ms for a keystroke,
// then repeat...
//
while( 1 ) {

    cvCopyImage( image, temp );
    if( drawing_box ) draw_box( temp, box );
    cvShowImage( "Box Example", temp );

    if( cvWaitKey( 15 )==27 ) break;
}

// Be tidy
//
cvReleaseImage( &image );
cvReleaseImage( &temp );
cvDestroyWindow( "Box Example" );
}

// This is our mouse callback. If the user
// presses the left button, we start a box.
// when the user releases that button, then we
// add the box to the current image. When the
// mouse is dragged (with the button down) we
// resize the box.
//
void my_mouse_callback(
    int event, int x, int y, int flags, void* param
) {

    IplImage* image = (IplImage*) param;

    switch( event ) {
        case CV_EVENT_MOUSEMOVE: {
            if( drawing_box ) {

```

```

        box.width = x-box.x;
        box.height = y-box.y;
    }
}
break;
case CV_EVENT_LBUTTONDOWN: {
    drawing_box = true;
    box = cvRect(x, y, 0, 0);
}
break;
case CV_EVENT_LBUTTONUP: {
    drawing_box = false;
    if(box.width<0) {
        box.x+=box.width;
        box.width *=-1;
    }
    if(box.height<0) {
        box.y+=box.height;
        box.height *=-1;
    }
    draw_box(image, box);
}
break;
}
}

```

【97~99】

Sliders, Trackbars 和 Switches

HighGUI 为我们提供了 slider(滑动条)的实现，在 OpenCV 中 slider 称为 trackbar(滑动条)。这是因为在 OpenCV 中加入 slider 的最初目的是为了实现视频的拖放。当然，在 HighGUI 中加入了 slider 以后，trackbar 被用作实现各种其他功能(见下一节“无按钮”)!

【99~100】

与窗口一样，滑动条被指定了一个独立的名称(字符串形式)，并且后面使用此名称来指定这个滑动条。HighGUI 中创建滑动条的函数如下：

```

int cvCreateTrackbar(
    const char*         trackbar_name,
    const char*         window_name,
    int*                value,

```

```
    int          count,
    CvTrackbarCallback  on_change
);
```

前两个参数分别指定了滑动条的名字以及滑动条附属窗口的名字。当滑动条被创建后，滑动条会被创建在窗口的顶部或者底部^①。另外，滑动条不会遮挡窗口中的图像。

随后的两个参数之一为 `value`，它一个整数指针，当滑动条被拖动时，OpenCV 会自动将当前位置所代表的值传给指针指向的整数；另外一个参数 `count` 是一个整数值，为滑动条所能表示的最大值。

最后一个参数是一个指向回调函数的指针，当滑动条被拖动时，回调函数会自动被调用。这跟鼠标事件的回调函数实现类似。回调函数必须为 `CvTrackbarCallback` 格式，`CvTrackbarCallback` 定义如下：

```
void (*callback)( int position )
```

这个回调函数不是必须的，所以如果不需要一个回调函数，可以将参数设置为 `NULL`，没有回调函数，当滑动条被拖动时，惟一的影响就是改变指针 `value` 所指向的整数值。

最后，HighGUI 提供了两个函数分别用来读取与设置滚动条的 `value` 值，不过前提是必须知道滑动条的名称。

```
int cvGetTrackbarPos(
    const char* trackbar_name,
    const char* window_name
);

void cvSetTrackbarPos(
    const char* trackbar_name,
    const char* window_name,
    int        pos
);
```

这两个函数可以用在程序的任何地方来读取或设置滑动条的值。

【100】

① 滑动条具体显示的窗口的顶部或者底部取决于当前的操作系统，但是在同一台计算机上，滑动条会显示在窗口的同一个位置。

无按钮

不幸的是，HighGUI 没有显示提供任何形式的按钮。我们经常使用的方法是，用只有两个取的滑动条来代替按钮。当然也有其他一些取巧的方法^①。另外一个在.../opencv/samples/c/ 目录下很多例子中使用的方法是用键盘快捷键来取代按钮。

开关(switch)事实上就是只有两个状态的滑动条，这两个状态是“on”(1)和“off”(0)(当 count 被设置成 1 时)。你可以发现使用滑动条来实现按钮功能以及开关是很简单的事情。根据你自己的要求，通过滑动条的回调函数，可以将滑动条的 value 每次自动设置成 0(如例 4-2，这跟大多数按钮的行为类似)，也可以将其他滑动条的值设置为 0(实现类似单选按钮的功能)。

例 4-2：使用滑动条实现一个开关功能，用户可以选择打开或关闭

```
// We make this value global so everyone can see it.  
//  
int g_switch_value = 0;  
  
// This will be the callback that we give to the  
// trackbar.  
//  
void switch_callback( int position ) {  
    if( position == 0 ) {  
        switch_off_function();  
    } else {  
        switch_on_function();  
    }  
}  
  
int main( int argc, char* argv[] ) {  
  
    // Name the main window  
    //  
    cvNamedWindow( "Demo Window", 1 );
```

① 另外一个常用的方法是用图像来代替按钮，当触发了鼠标事件，判断鼠标当前位置的(x,y)是否在代表按钮的图像范围内，当(x,y)在图像范围内，按钮事件被触发。在这种情况下，所有的按钮实际上是通过窗口的鼠标回调函数实现的。

```

// Create the trackbar. We give it a name,
// and tell it the name of the parent window.
//
cvCreateTrackbar(
    "Switch",
    "Demo Window",
    &g_switch_value,
    1,
    Switch_callback
);

// This will just cause OpenCV to idle until
// someone hits the "Escape" key.
//
while( 1 ) {
    if( cvWaitKey(15)==27 ) break;
}

}

```

可以看出，程序实现了类似灯开关一样的打开和关闭功能。在示例程序中，滑动条的 value 被设为 0 时，回调函数调用 switch_off_function()；设为 1 时，回调函数调用 switch_on_function() 函数。

【101~102】

视频的处理

处理视频相关问题的时候需要一些函数，当然，首先要用到的就是读/写视频文件的函数。我们也要知道如何在屏幕上播放视频。

在 OpenCV 中，处理视频时，我们最先要了解的就是 CvCapture。CvCapture 结构包含从摄像机或视频文件中读取帧所需的信息。根据视频来源，我们可以使用下面两个函数之一来初始化 CvCapture 结构。

```

CvCapture* cvCreateFileCapture( const char* filename );
CvCapture* cvCreateCameraCapture( int index );

```

当使用 cvCreateFileCapture() 时，我们只需要将 MPG 或 AVI 视频文件名告诉 cvCreateFileCapture()，OpenCV 会打开并准备读取视频。如果打开成功，将返回一个指向已经初始化了的 CvCapture 结构的指针，随后便可以读入视频的帧。

很多人不习惯检查函数返回的结果，想当然的认为不会有问题。请一定要检查返回值。在一些情况下(比如文件不存在)无法打开文件，函数会返回 NULL 指针；当压缩视频的编码未知时 cvCreateFileCapture() 也会返回 NULL 指针。视频压缩编码的具体问题超出了本书的范围，但是为了使视频可以被成功读入，必须确保这种视频解码库已经安装在系统中。比如你想在 Windows 中读入一些 DIVX 或者 MP4 压缩的文件，需要有实现视频解码的 DLL 文件。这样一来，即使程序可以在一台电脑(安装了所需的 DLL 文件)上正常运行，但是在其他电脑(未安装所需的 DLL 文件)上可能会出现问题。这也就是为什么必须要检查 cvCreateFileCapture() 函数的返回结果。创建 CvCapture 结构之后，即可开始读入视频和做很多其他事情。在此之前，让我们看看如何从摄像机中得到图像。

【102】

函数 cvCreateCameraCapture() 的用法与 cvCreateFileCapture() 非常类似，但是没有处理视频编码解码的麻烦^①。在这种情况下，可以用 identifier 指定我们需要使用的摄像机，告诉操作系统如何与摄像机交互。对于前者，我们通过一个整数值来确定要使用的摄像机，当只有一个摄像机时，参数值取 0。而与操作系统交互的问题，identifier 的 domain 通过一个整数来告诉 OpenCV 所使用摄像机的类型。Domain 的值可参见表 4-3。

表 4-3：摄像机参数指定 HighGUI 如何连接到摄像机

摄像机捕获常数	数值
CV_CAP_ANY	0
CV_CAP_MIL	100
CV_CAP_VFW	200
CV_CAP_V4L	200
CV_CAP_V4L2	200
CV_CAP_FIREWIRE	300
CV_CAP_IEEE1394	300
CV_CAP_DC1394	300
CV_CAP_CMU1394	300

调用函数 cvCreateCameraCapture() 时，我们将前面提到的两个整数之和传给

① 当然，公平地说，不同视频编码解码带来的麻烦被系统支持或者不支持的摄像机类型的麻烦所取代。

`cvCreateCameraCapture()`。例如：

```
CvCapture* capture = cvCreateCameraCapture( CV_CAP_FIREWIRE );
```

这段函数表示，`cvCreateCameraCapture()`会打开第一个 Firewire 摄像机。只有一个摄像机时，大多数情况下没有必要使用 domain；使用 `CV_CAP_ANY` 就很方便。在我们进入下一部分前，提示一个有用的小敲门，当 `cvCreateCameraCapture()` 的参数被设置为 -1 时，OpenCV 会打开一个窗口让用户选择需要摄像机。

读视频

```
int          cvGrabFrame( CvCapture* capture );
IplImage*   cvRetrieveFrame( CvCapture* capture );
IplImage*   cvQueryFrame( CvCapture* capture );
```

创建一个有效的 `CvCapture` 结构之后，便可以开始读视频帧。这里有两种方法。第一种方法是使用 `cvGrabFrame()`，该函数以 `CvCapture*` 指针为参数，返回一个整数，当读取帧成功时返回 1，否则返回 0。`cvGrabFrame()` 将视频帧复制到了一个用户不可见的内存空间里。为什么 OpenCV 会将视频帧复制到一个用户不可见的空间里？这是因为获取的视频帧数据是未经过处理的，`cvGrabFrame()` 被设计为用于快速将视频帧读入内存。

【103~104】

在 `cvGrabFrame()` 以后，可以调用 `cvRetrieveFrame()` 来处理 `cvGrabFrame()` 读入的视频数据。这个函数会对读入帧做所有必须的处理（包括图像解码操作），并且返回一个 `IplImage*` 指针，该指针指向另一块内部内存空间（不要过于依赖这个指针指向的图像，因为 `cvGrabFrame()` 下一次调用时，指针所指向的空间会被新的图像覆盖）。如果想对这幅图像做一些特别处理，先将图像数据复制到其他地方。因为这个指针所指向的结构空间由 OpenCV 管理，所以不要试图释放这个空间，否则会产生一些不可预测的错误。

接下来介绍第二种方法。这种方法比较简单，使用 `cvQueryFrame()` 函数。`cvQueryFrame()` 实际上是 `cvGrabFrame()` 与 `cvRetrieveFrame()` 的一个组合。它与 `cvRetrieveFrame()` 返回同样的指针。

值得注意的是，对于一个视频文件，当调用 `cvGrabFrame()` 时，视频帧会自动前进一步。所以下一次调用会自动读入下一帧视频。

`CvCapture` 结构使用结束后，可以调用 `cvReleaseCapture()` 来释放 `CvCapture`

结构。与 OpenCV 中大部分释放函数类似，这个函数读入一个指向 CvCapture* 结构的指针。

```
void cvReleaseCapture( CvCapture** capture );
```

事实上，我们还可以对 CvCapture 结构执行很多其他的操作，比如，我们可以查询与设置视频的各种属性：

```
double cvGetCaptureProperty(
    CvCapture* capture,
    int property_id
);

int cvSetCaptureProperty(
    CvCapture* capture,
    int property_id,
    double value
);
```

函数 cvGetCaptureProperty() 可以接受的参数见表 4-4。

表 4-4：视频捕捉属性设置

视频捕捉属性	参数	数值
CV_CAP_PROP_POS_MSEC	0	
CV_CAP_PROP_POS_FRAME	1	
CV_CAP_PROP_POS_AVI_RATIO	2	
CV_CAP_PROP_FRAME_WIDTH	3	
CV_CAP_PROP_FRAME_HEIGHT	4	
CV_CAP_PROP_FPS	5	
CV_CAP_PROP_FOURCC	6	
CV_CAP_PROP_FRAME_COUNT	7	

【104~105】

大部分属性都可以从名称中看出它的意思。POS_MSEC 是指向视频的当前位置，以毫秒为单位。POS_FRAME 是以帧为单位的当前位置。POS_AVI_RATIO 是用介于 0 至 1 间的数，表示位置(这在用户使用滑动条拖放视频时很有用处)。FRAME_WIDTH 与 FRAME_HEIGHT 是当前读取的帧(或者当前从摄像机捕获的帧)的宽度与高度。FPS 是针对视频文件来说的，它记录了视频录入时每秒钟的帧数。这个参数用来以

正确的速度播放视频。FOURCC 由四个字节组成，表示视频文件的压缩方法。`FRAME_COUNT` 表示视频文件的总帧数，但是这个数据不是非常可靠。

前面所有参数都以 `double` 类型返回，这对大部分都适用，除了参数为 FOURCC (FourCC) [FourCC85] 的情况。此时需要转化参数的类型，具体见例 4-3。

例 4-3：获得视频编码格式

```
double f = cvGetCaptureProperty(  
    capture,  
    CV_CAP_PROP_FOURCC  
) ;  
  
char* fourcc = (char*) (&f);
```

对于前面每一个视频捕捉属性，有一个对应的函数 `cvSetCaptureProperty()` 来设置这些属性。这些不是全部有效的，比如，读入视频文件时不可以设置 FOURCC 属性。通过设置当前位置来从不同位置读入视频应该是可行的，但是只对一部分编码格式有效(第 5 章将更详细地介绍视频编码和解码问题)。

写视频

我们对视频的另一个操作就是将视频写入磁盘。OpenCV 使这件事情非常简单。除了个别细节，大致与读入视频一致。

首先必须创建一个 `CvVideoWriter` 结构。`CvVideoWriter` 是一个类似于 `CvCapture` 的视频写入结构。此结构与下面的函数一起使用。

```
CvVideoWriter* cvCreateVideoWriter(  
    const char* filename,  
    int fourcc,  
    double fps,  
    CvSize frame_size,  
    int is_color = 1  
) ;  
int cvWriteFrame(  
    CvVideoWriter* writer,  
    const IplImage* image  
) ;  
void cvReleaseVideoWriter(  
) ;
```

```
CvVideoWriter** writer  
);
```

【105~106】

你会注意到，相比视频读入，视频写入需要一些额外的参数。除了视频文件名称，我们需要告诉视频写入结构视频编码格式、帧率以及每一帧的大小。还可以告诉OpenCV 图像是否彩色(默认为彩色)。

这里，四个字符标记来表示编码格式。(不了解视频编码的读者此时只需知道这是一个标识编码的四字节惟一性标志)int 类型的 cvCreateVideoWriter() 参数 fourcc 是将四个字符被打包在一个整数中。由于经常使用，OpenCV 提供了一个宏 CV_FOURCC(c0,c1,c2,c3) 来实现这个打包操作。

视频写入结构建好之后，需要调用 cvWriteFrame() 函数，并且为 cvWriteFrame() 函数传入一个 CvVideoWriter* 指针和一个 IplImage* 指针来写入文件。

写入完成后，为了关闭写入结构，必须调用 CvReleaseVideoWriter() 函数。如果平时不习惯释放不再使用的空间和结构，对这个结构千万别偷懒，因为不显式地释放写入结构，视频文件可能会被损坏。

ConvertImage 函数

完全由于历史原因，HighGUI 中有一个孤立的函数，独立于前面的任何一类。但此函数使用广泛，其重要性不言而喻，它便是 cvConvertImage()。

```
void cvConvertImage(  
    const CvArr* src,  
    CvArr* dst,  
    int flags = 0  
) ;
```

cvConvertImage() 用于在常用的不同图像格式之间转换。文件格式在 src 与 dst 图像的头文件中指出(该函数还允许使用比 IplImage 更通用的 CvArr 类型)。

源图像可以是单个、3 个或者 4 个通道，可以是 8 位或者浮点类型像素格式。目标图像必须是 8 位的单通道或者 3 个通道。函数也可以将彩色图像转换为灰度图或者将单通道的灰度图转换为 3 个通道的灰度图(彩色)。最后，参数 flags 可以垂直旋转图像。这是很有用的，有时候摄像机图像格式与图像显示格式会反转。设置这个参数可以在内存中彻底旋转图像。

【106~107】

练习

1. 本章完整讲述了基本的输入/输出编程以及 OpenCV 的数据结构。下面的练习是基于前面的知识做一些应用，为后面大程序的实现提供帮助。
 - a. 创建一个程序实现以下功能：(1)从视频文件中读入数据；(2)将读入的数据转换为灰度图；(3)对图像做 Canny 边缘检测。将三个过程的处理结果显示到不同的窗口中，每个窗口根据其内容合理命名。
 - b. 将所有三个步骤实现显示在一个图像中。

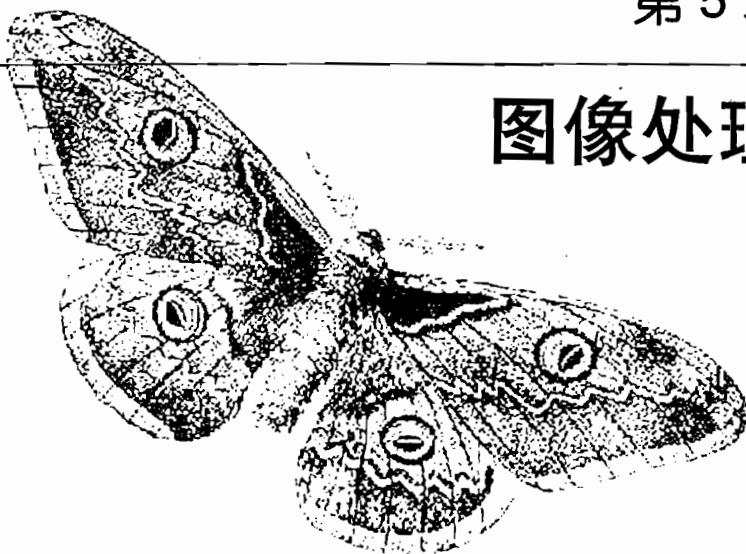
提示：创建一个新的图像，其高度与原始图像相同，宽度为原来视频帧的 3 倍，将 3 幅图像分别复制到新的图像中：可使用指针；或者更巧妙地创建三个图像头，三个图像头分别指向图像数据的开始处， $1/3$ 处和 $2/3$ 处，然后使用函数 cvCopy() 复制。
 - c. 在图像的三个不同部分写上合适的文字标签。
2. 创建一个程序，使其读入并显示一幅图像。当用户鼠标点击图像时，获取图像对应像素的颜色值(BGR)，并在图像上点击鼠标处用文本将颜色值显示出来。
 - a. 对于练习 1b，在三幅图像中任何地方点击鼠标，将当前图像的坐标显示于点击鼠标处。
3. 创建一个程序读入并显示一幅图像。
 - a. 允许用户在图像中选择一个矩形区域，然后通过按住鼠标按键画一个矩形。当鼠标键放开，高亮显示矩形框。注意，在内存中保留一个原始图像的副本，以防止修改图像时改变原始图像的数据。在下一次点击鼠标按键时，图像恢复为原始图像并重新开始绘矩形。
 - b. 在另一个独立的窗口中，使用画图函数画一个图表，分别用蓝、绿和红色表示选中区域中每种颜色的像素数量。这是选定区域的颜色直方图。x 坐标系表示像素值范围在 0–31, 32–63, …, 223–255; y 坐标表示在选定区域中对应像素的数量。对每一个颜色通道(BGR)都进行统计。
4. 创建一个程序读入和显示视频文件，并可以使用滑动条控制视频文件的播放。一个滑动条用来控制视频播放位置，以 10 为步长跳进。另一个滑动条用来控制停止/播放。

5. 创建一个简单画图程序。
 - a. 创建一个程序，可以新建一个图像，并将图像所有像素值设置为 0，然后将其显示出来。允许用户通过鼠标左键画线、圆、椭圆以及多边形。并使其能在右击鼠标时，实现橡皮擦功能。
 - b. 实现“逻辑画图”，用户通过滑动条选择 AND、OR 或者 XOR。例如，如果用户选择 AND，那么在用户画图时，只显示原始像素数据为 1 的像素。
6. 写一个程序，使其创建一幅图像，然后将图像所有像素值设置为 0，显示出来。当用户点击一个位置，便可以在此位置输入一个标签。允许使用 Enter 键进行编辑，以及实现一个退出键。按 Enter 键可以修改当前位置的标签。
7. 透视变换
 - a. 写一个程序读入一幅图像，并使用数字键 1~9 控制变换矩阵(参考第 6 章中的 cvWarpPerspective() 函数)。按住任意 1~9 中任意一个按键，透视变换矩阵中对应数据会变大。同时按下 Shift 键时，对应数据减小(最小为 0)。每次改变一个数据，在两个窗口中显示图像：原始图像和变换后的图像。
 - b. 添加放大、缩小功能。
 - c. 添加旋转图像功能。
8. 有趣的人脸程序。进入目录/samples/c/ 编译 facedetect.c 代码。画一个骷髅头(或者从网上下载)并保存在磁盘文件中。修改 facedetect 程序，使之能读入骷髅头图像。
 - a. 检测到人脸，便将人脸矩形区域中的图像用骷髅头图像代替。

提示：cvConvertImage() 可以改变图像大小，或者使用 cvSize() 函数。可以将矩形区域设置成 ROI，然后使用 cvCopy() 复制缩放后的图像。
 - b. 加入一个滑动条，在范围 0.0~1.0 之间设置 10 位置。使用滑动条来调节骷髅头像与矩形区域内图像的 alpha 融合(使用 cvAddWeighted() 函数)。
9. 图像稳定。进入目录/samples/c/，编译 lkdemo 代码(运动跟踪或光流法代码)。在一个更大的窗口中创建并显示视频图像。轻轻移动摄像机，并用光流法的向量将图像显示在大窗口中，并保持画面稳定。这是一个最基本的图像稳定技术。

第 5 章

图像处理



综述

到这里，我们已经掌握了关于图像处理的所有基础知识。我们了解了 OpenCV 库的结构，也知道了通常用来表示图像的基本数据结构。通过熟悉 HighGUI 接口，我们可以运行程序并将结果显示在屏幕上。掌握了这些用来控制图像结构的基本方法，我们就可以学习更复杂的图像处理方法了。

现在，我们来继续学习高级的处理方法，即把图像以“图像”的方式来处理，而不仅是颜色值(或灰度值)组成的数组。在提到“图像处理”时，它的意思是：使用图像结构中所定义的高层处理方法来完成特定任务，这些任务是图形和视觉范畴的任务。

平滑处理

“平滑处理”也称“模糊处理”(blurring)，是一项简单且使用频率很高的图像处理方法。平滑处理的用途有很多，但最常见的是用来减少图像上的噪声或者失真。降低图像分辨率时，平滑处理是很重要的(在本章的“图像金字塔”部分会详细介绍这一点)。

目前 OpenCV 可以提供五种不同的平滑操作方法，所有操作都由 `cvSmooth` 函数实

现^①，该函数可以将用户所期望的平滑方式作为参数。

```
void cvSmooth(  
    const CvArr* src,  
    CvArr* dst,  
    int smoothtype = CV_GAUSSIAN,  
    int param1 = 3,  
    int param2 = 0,  
    double param3 = 0,  
    double param4 = 0  
) ;
```

【109~110】

src 和 dst 分别是平滑操作的输入图像和结果，cvSmooth() 函数包含 4 个参数：param1, param2, param3 和 param4。这些参数的含义取决于 smoothtype 的值，这些值可能是表 5.1 中列出的任何一个^②。(请注意，有些操作不支持 in place 方式的输入。in place 方式意味着输入图像与结果图像是同一个图像。)

表 5-1：平滑操作的各种类型

平滑类型						
	方法	No.	输入数据类型	输出数据类型	说明	示例
CV_BLUR	简单模糊	是	1,3	8u, 32f	8u, 32f	对每个像素 $\text{param1} \times \text{param2}$ 邻域求和，并做 缩放 $1/(\text{param1} \times \text{param2})$
CV_BLUR_NO_SCALE	简单无缩放变换的模糊	否	1	8u	16s (占 8u 的资源) 或 32f (占 32f 的资源大小)	对每个像素的 param1 \times param2 邻域求和

- ① 注意，与 MATLAB 不同，用 OpenCV 的滤波操作(例如 cvSmooth(), cvErode() 和 cvDilate()) 处理后的图像与输入图像的大小是相同的。为了达到这个效果，OpenCV 在图像边界之外创造了“虚拟”像素，默认情况下通过边缘复制来完成，也就是说 $\text{input}(-dx, y) = \text{input}(0, y)$, $\text{input}(w+dx, y) = \text{input}(w-1, y)$ ，以此类推。
- ② 有时，我们会用 8u 作为 8 位无符号的图像深度(IPL_DEPTH_8U)的缩写，其他缩写见表 3-2。

续表

函数名	功能	参数	返回值	说明
CV_MEDIAN	中值模糊	否 1,3	8u 8u	对图像进行核大小为 $\text{param1} \times \text{param1}$ 的中值滤波
CV_GAUSSIAN	高斯模糊	是 1,3	8u, 32f 8u (占 8u 的资源) 或 32f (占 32f 的资源)	对图像进行核大小为 $\text{param1} \times \text{param2}$ 的高斯卷积
CV_BILATERAL	双边滤波	否 1,3	8u 8u	应用双线性 3×3 滤波, 颜色 $\text{sigma} = \text{param1}$, 空间 $\text{sigma} = \text{param2}$

图 5.1CV_BLUR 所例举的 simple blur 是最简单的一项操作。输出图像的每一个像素是窗口中输入图像对应像素的简单平均值。simple blur 支持 1~4 个图像通道, 可以处理 8 位图像或者 32 位的浮点图像。

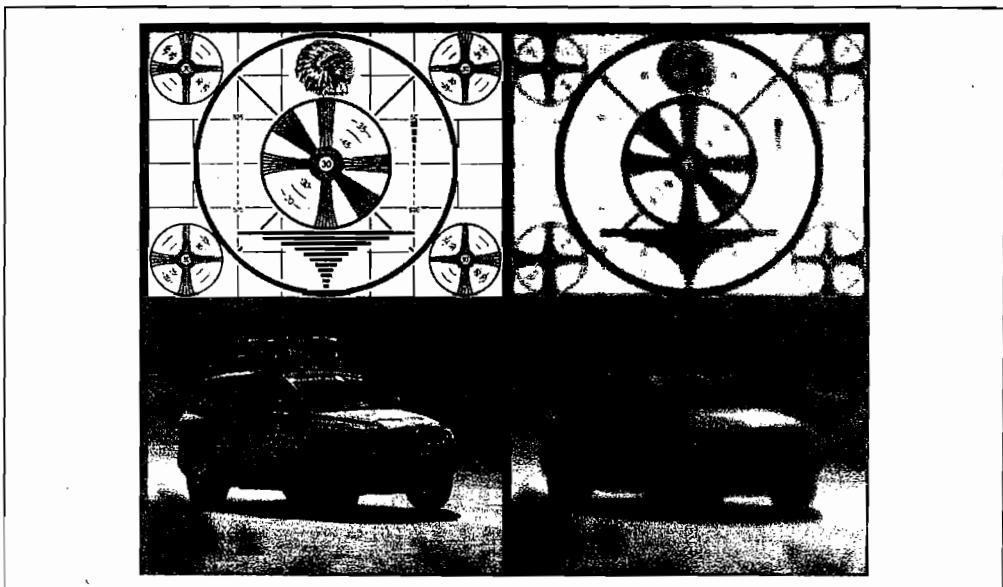


图 5-1: 简单图像平滑处理: 左边为输入图像, 右边为结果图像

不是所有的模糊操作的输入和结果图像类型都相同, CV_BLUR_NO_SCALE(不缩放比例而进行模糊处理)与 simple blur 本质上是相同的, 但并没有计算其平均值的操

作。因此，输入图像和结果图像必须有不同的数值精度，才能保证模糊操作不会导致错误溢出。不缩放比例的 simple blur 支持 8 位的输入图像，结果图像的数据类型必须是 IPL_DEPTH_16S (CV_16S) 或 IPL_DEPTH_32S (CV_32S)。同样的操作也可以在 32 位浮点图像上进行，结果图像也应该是 32 位浮点类型。简单无缩放变换的模糊不支持 in place 方式：输入图像与结果图像必须不同。(在 8 位和 16 位情况下，很明显不能用 in place 方式；用 32 位图像时，也保持这一规定。) 用户选择不缩放比例的模糊操作是因为其比缩放比例的模糊操作要快一些。【110~111】

中值滤波器(CV_MEDIAN) [Bardyn84]将中心像素的正方形邻域内的每个像素值用中间像素值(不是平均像素值)替换，它可以用来处理单个通道、三个通道或者四个通道 8 位的图像，但不可以 in place 操作。中值滤波的结果见图 5-2。基于平均算法的 simple blur 对噪声图像特别是有大的孤立点(有时被称为“镜头噪声”)的图像非常敏感，即使有极少量点存在较大差异也会导致平均值的明显波动，因此中值滤波可以通过选择中间值避免这些点的影响。

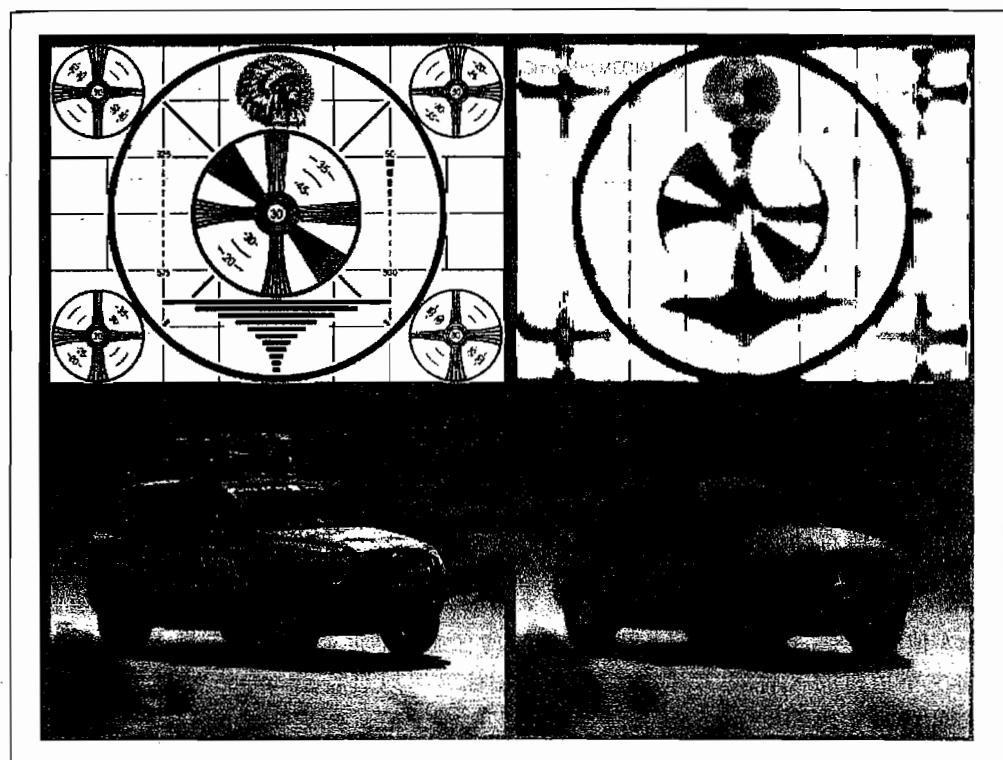


图 5-2：用中值滤波进行的图像模糊处理

下一个平滑滤波器是 Gaussian filter (CV_GAUSSIAN)，虽然它不是最快的，但它

是最有用的滤波器。高斯滤波用卷积核与输入图像的每个点进行卷积，将最终计算结果之和作为输出图像的像素值。

对于高斯模糊(图 5-3)，前两个参数代表滤波器窗口的宽度和高度，可选择的第三个参数代表高斯卷积核的 sigma 值(是最大宽度的四分之一)。如果第二个参数未指定，系统将会根据窗口尺寸通过下面的方程来自动确定高斯核的各个参数。

$$\sigma_x = \left(\frac{n_x}{2} - 1 \right) \cdot 0.30 + 0.80, n_x = \text{param1}$$

$$\sigma_y = \left(\frac{n_y}{2} - 1 \right) \cdot 0.30 + 0.80, n_y = \text{param2}$$

【111-112】

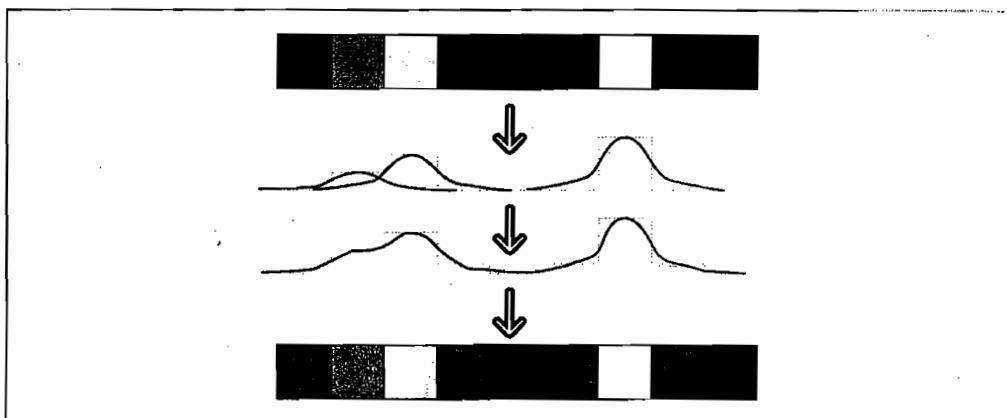


图 5-3：对一维像素数组进行高斯模糊

如果用户希望高斯核不对称，那么可以引入第四个参数。这样，第三个和第四个参数分别为水平方向和垂直方向的 sigma 值。

如果第三个和第四个参数已经给出，但是前两个参数被设为 0，那么窗口的尺寸会根据 sigma 值自动确定。

高斯滤波的 OpenCV 的实现还为几个常见的核提供了更高的性能优化。具有标准 sigma 值的 3×3 , 5×5 和 7×7 比其他核具有更优的性能。高斯模糊支持单个通道或者三个通道的 8 位或 32 位的浮点格式图像，可以进行 in place 方式操作。高斯模糊的效果见图 5-4。

OpenCV 支持的第五个也是最后一个平滑操作被称作双边滤波(bilateral filtering)[Tomasi98]，举例见图 5-5。双边滤波是“边缘保留滤波”的图像分析方法中的一种。将它与高斯平滑对比后会更容易理解。进行高斯滤波的通常原因是真实

在空间内的像素是缓慢变化的，因此邻近点的像素变化不会很明显。但是随机一个点就可能形成很大的像素差(也就是说空间上噪声点不是相互联系的)。正是这一点，高斯滤波在保留信号的条件下减少噪声。遗憾的是，这种方法在接近处就无效了，在那儿你不希望像素与相邻像素相关。因此，高斯滤波会磨平边缘而双边滤波能够提供一种不会将边缘的平滑掉的方法，但作为代价，需要更多时间。

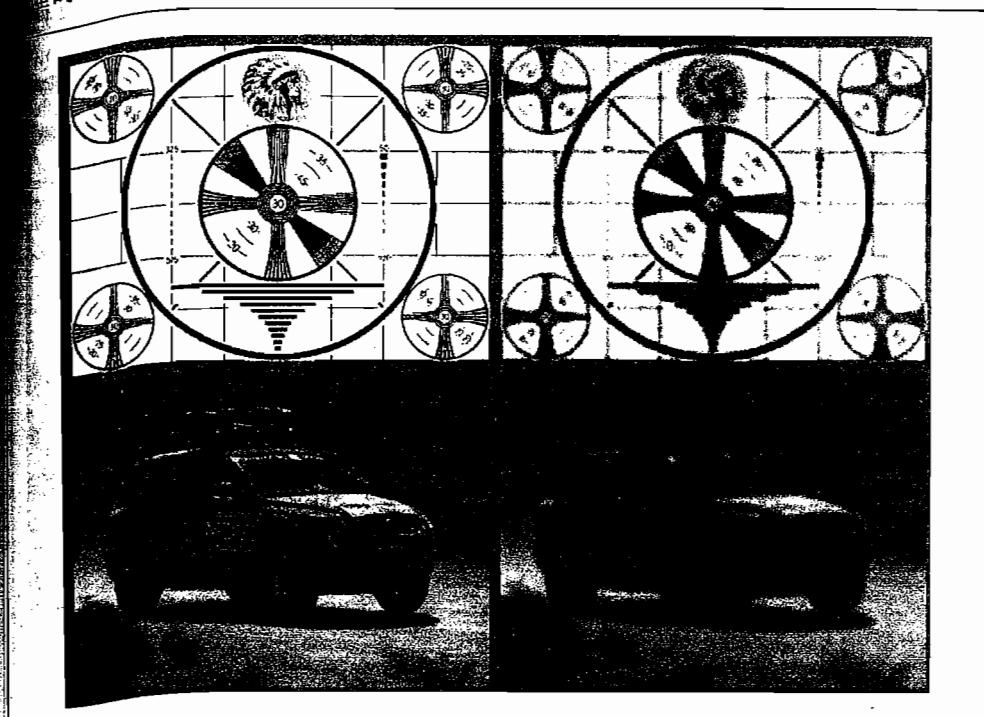


图 5-4：高斯模糊

与高斯滤波类似，双边滤波会依据每个像素及其邻域构造一个加权平均值，加权计算包括两个部分，其中第一部分加权方式与高斯平滑中的相同，第二部分也属于高斯加权，但不是基于中心像素点与其他像素点的空间距离之上的加权，而是基于其他像素与中心像素^①的亮度差值^②的加权。可以将双边滤波视为高斯平滑，对相似

-
- ① 严格来讲，高斯分布函数的应用不是双边滤波的必要条件。尽管可以采用多种加权方法，OpenCV 中实现的双边滤波依然采用了高斯加权。
 - ② 对于多通道图像，也就是彩色图像，使用各种颜色差值的加权和来代替亮度差值。这种加权法被用来确定 CIE 颜色空间的 Euclidean 距离。
-

的像素赋予较高的权重，不相似的像素赋予较小的权重。这种滤波的最终效果就是使处理过的图像看上去像是一幅源图的水彩画^①，可用于图像的分割。

双边滤波含有两个参数。第一个参数代表空域中所使用的高斯核的宽度，与高斯滤波的 sigma 参数类似。第二个参数代表颜色域内高斯核的宽度。第二个参数越大，表明待滤波的强度(或颜色)范围越大(因此不连续的程度越高，以便保留)

【113—114】

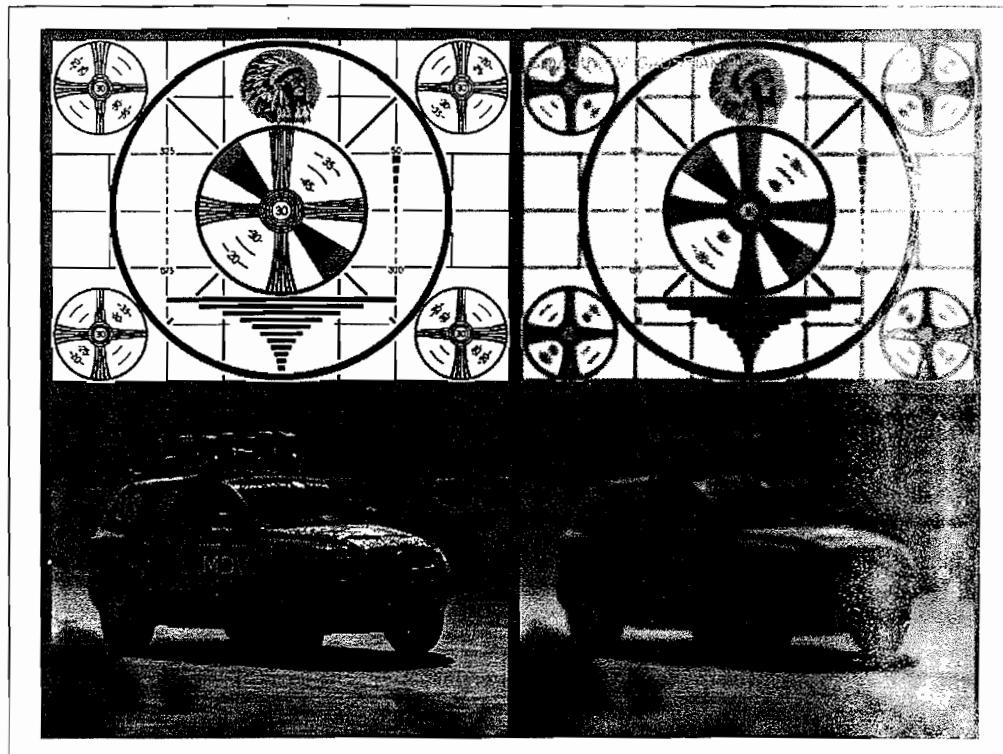


图 5-5：双边滤波结果

图像形态学

OpenCV 为进行图像的形态学变换[Serra83]提供了快速、方便的函数。基本的形态学转换是膨胀与腐蚀，它们能实现多种功能：例如消除噪声、分割出独立的图像元素

① ① 迭代执行多次双边滤波后，这种效果尤为明显。

以及在图像中连接相邻的元素。形态学也常被用于寻找图像中的明显的极大值区域或极小值区域以及求出图像的梯度。

膨胀和腐蚀

膨胀是指将一些图像(或图像中的一部分区域，称之为 A)与核(称之为 B)进行卷积。核可以是任何的形状或大小，它拥有一个单独定义出来的参考点(anchor point)。多数情况下，核是一个小的中间带有参考点的实心正方形或圆盘。核可以视为模板或掩码，膨胀是求局部最大值的操作。核 B 与图像卷积，即计算核 B 覆盖的区域的像素点最大值，并把这个最大值赋值给参考点指定的像素。这样就会使图像中的高亮区域逐渐增长，如图 5-6 所示。这样的增长就是“膨胀操作”的初衷。【115】

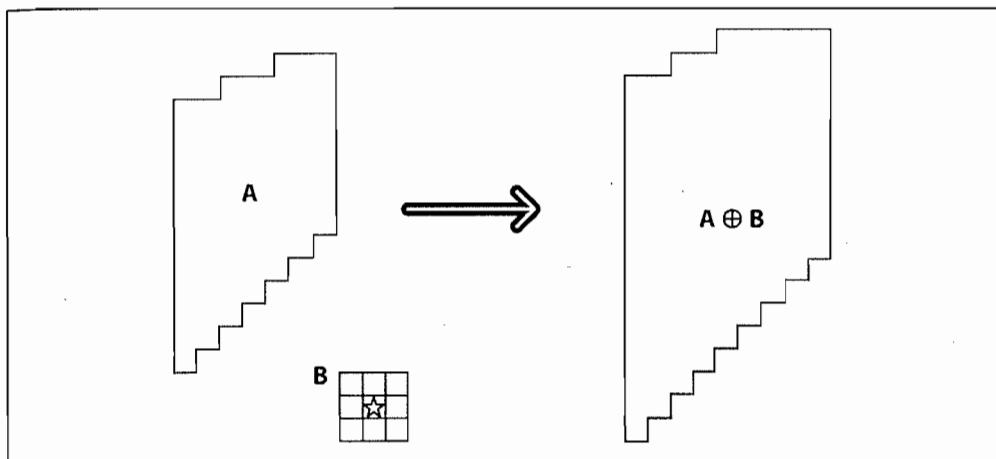


图 5-6：形态学膨胀：在核 B 下取最大像素值

腐蚀是膨胀的反操作。腐蚀操作要计算核区域像素的最小值。腐蚀可以通过下面的算法生成一个新的图像：当核 B 与图像卷积时，计算被 B 覆盖区域的最小像素值，并把这个值放到参考点上。^① 腐蚀后的图像如图 5-7 所示。

注意：图像形态学经常在通过阈值化获得的二进制图像中完成。因为膨胀是求最大值操作，腐蚀是求最小值的操作，所以形态学也可以用于灰度图像。

一般来说，膨胀扩展了区域 A，而腐蚀缩小了区域 A。此外，膨胀可以填补凹洞，

① 确切地讲，是把目标图像像素点的值设为源图像核区域 B 中像素的最小值。

腐蚀能够消除细的凸起。当然，准确的效果将取决于核，但当使用凸核时前面的说法一般是对的。

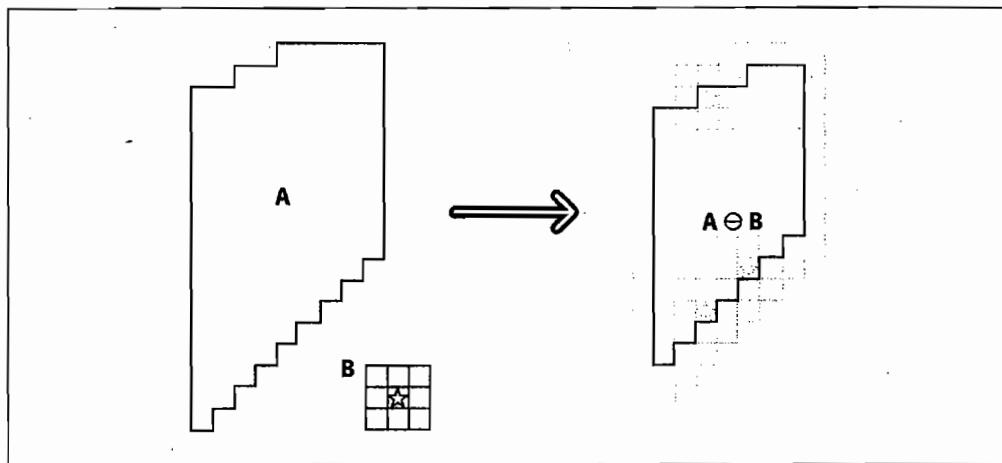


图 5-7：形态腐蚀：在核 B 之下取最小像素值

在 OpenCV，我们利用 cvErode() 和 cvDilate() 函数实现上述变换。

```
void cvErode(
    IplImage*         src,
    IplImage*         dst,
    IplConvKernel*   B      = NULL,
    int              iterations = 1
);
void cvDilate(
    IplImage*         src,
    IplImage*         dst,
    IplConvKernel*   B      = NULL,
    int              iterations = 1
);
```

【116~117】

cvErode() 和 cvDilate() 都有源图像和目标图像参数，它们都支持“in-place”操作（源图像和目标图像是同一个图像）。第三个参数是核，默认值为 NULL。当为空时，所使用的是参考点位于中心的 3×3 核（我们将简单讨论如何构造核）。最后，第四个参数是迭代的次数。如果未将它设置为默认值(1)，将在一次函数的调用中执行多次操作。腐蚀操作的结果如图 5-8 所示，膨胀操作的结果如图 5-9 所示。腐蚀操作通常是用来消除图像中“斑点”噪声。腐蚀可以将斑点腐蚀掉，且能确保图像内的较大区域依然存在。在试图找到连通分支(即具有相似颜色或强度的像素点的大块的互相分离的区域)时通常使用膨胀操作。因为在大多数情况下一个大区域

可能被噪声、阴影等类似的东西分割成多个部分，而一次轻微的膨胀又将使这些部分“融合”在一起。

综上所述：当 OpenCV 执行 cvErode () 函数时，将某点 p 的像素值设为与 p 对应的核覆盖下所有点中的最小值，同样的，对于执行膨胀操作时，将取最小值换为取最大值：

$$\text{erode}(x, y) = \lim_{(x', y') \in \text{kernel}} \text{src}(x+x', y+y')$$

$$\text{dilate}(x, y) = \max_{(x', y') \in \text{kernel}} \text{src}(x+x', y+y')$$

【117~118】

大家可能会想，既然之前的算法描述已经能解释清楚，为什么还需要引入一个复杂的公式呢？实际上，有些读者更喜欢这样的公式，更重要的是，公式可以阐明一些定性描述表达不清楚的一般性问题。我们能够注意到，如果图像不是二值的，那么膨胀和腐蚀操作起到的作用不是很明显。再看一看图 5-8 和图 5-9，分别展示了对两个图像进行腐蚀和膨胀操作的效果。

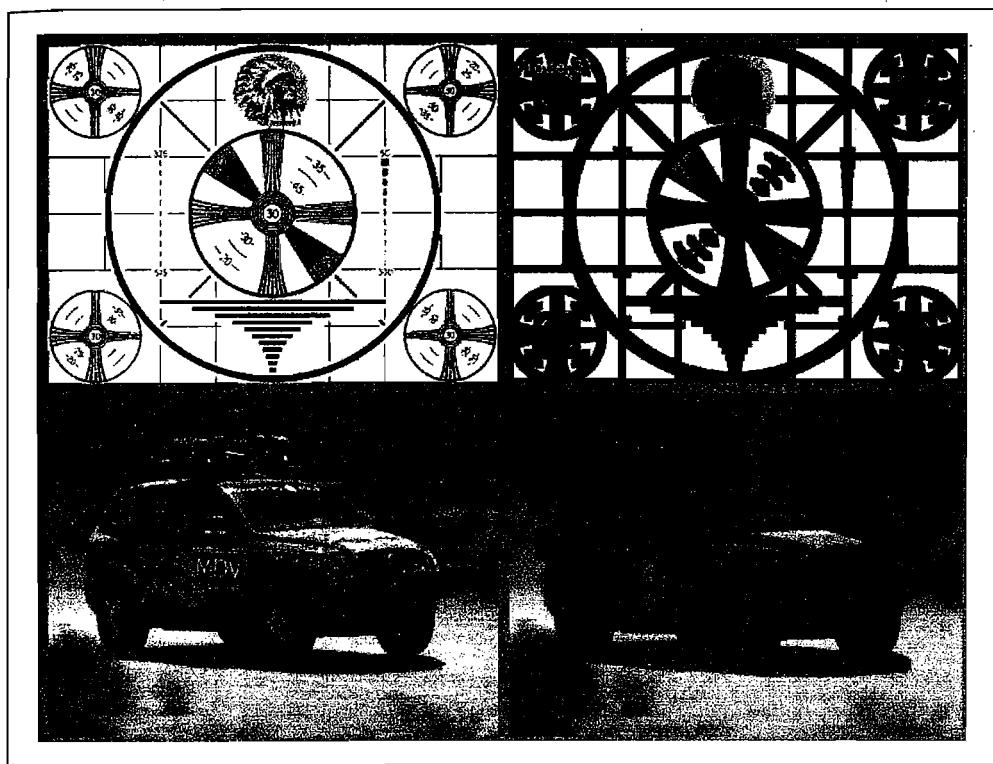


图 5-8：腐蚀的结果或者“最小化”操作，亮的区域被隔离并且缩小

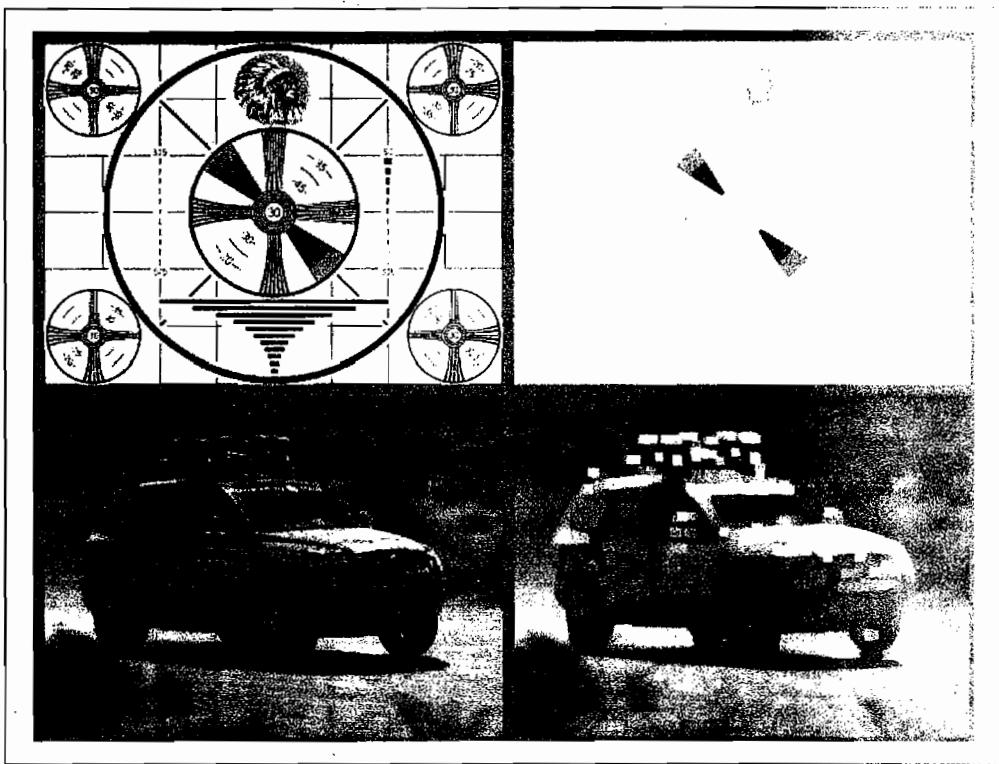


图 5-9：膨胀的“最大化”操作：亮的区域得到了扩展和连接

自定义核

你不必局限于选择 3×3 方形的核。可以创建自定义的 IplConvKernel 核(即我们之前提到的“核 B”)。这样的核由 cvCreateStructuringElementEx() 函数创建，由 cvReleaseStructuringElement() 函数释放。

```
IplConvKernel* cvCreateStructuringElementEx(
    int          cols,
    int          rows,
    int          anchor_x,
    int          anchor_y,
    int          shape,
    int*         values=NULL
);
void cvReleaseStructuringElement( IplConvKernel** element );
```

© 2005, 人民邮电出版社

形态核与卷积核不同，不需要任何的数值填充核。当核在图像上移动时，核的元素只需简单标明应该在哪个范围里计算最大值或最小值。参考点指定核与源图像的位置关系，同时也锁定了计算结果在目标图像中的设置。当构造核时，行与列确定了所构造的矩形大小(矩形内含有结构元素)，下两个参数 anchor_x 和 anchor_y，是核的封闭矩形内参考点的横纵坐标(x, y)。第五个参数，形状 shape 可以取表 5-2 中所列的值。如果使用 CV_SHAPE_CUSTOM，那么使用整数向量 value 在封闭矩形内定义核的形状。使用光栅扫描法读取向量，使每个元素代表封闭矩形中的不同像素。所有非零值指定在核中对应的各个像素点。如果值为空，通常会构造一个所有值为非空的矩形核^①。

【119~120】

表 5-2：IplConvKernel 的形状取值

形状值	含义
CV_SHAPE_RECT	核是矩形
CV_SHAPE_CROSS	核是十字交叉形
CV_SHAPE_ELLIPSE	核是椭圆形
CV_SHAPE_CUSTOM	核是用户自定义的值

更通用的形态学

在处理布尔图像和图像掩码时，基本的腐蚀和膨胀操作通常是足够的。然而，在处理灰度或彩色图像时，往往需要一些额外的操作。更通用的 cvMorphologyEx() 函数提供了更多有用的操作。

```
void cvMorphologyEx(
    const CvArr*      src,
    CvArr*            dst,
    CvArr*            temp,
    IplConvKernel*    element,
    int                operation,
    int                iterations = 1
```

除了以往操作中使用过的参数：如 src, dst, element 和 iterations 外，cvMorphologyEx() 函数增加了两个新的参数。第一个是 temp 数组，它在一些操作可能会用到(参见表 5-3)。使用该数组时，它应与源图像同样大小。第二个参数

① 这个整型向量的使用风格与在 OpenCV 其他函数的使用风格不一致，很多人对此感到疑惑。正如该结构使用 IPL 前缀一样，这种用法来源于“远古遗风”风格的代码。

operation 很有趣，它指定形态学操作的方法。

表 5-3: cvMorphologyEx() 操作选项

操作名称	形态学操作	是否需要源图像
CV_MOP_OPEN	开运算	否
CV_MOP_CLOSE	闭运算	否
CV_MOP_GRADIENT	形态梯度	总是
CV_MOP_TOPHAT	“礼帽”	in-place 情况下(src = dst)需要
CV_MOP_BLACKHAT	“黑帽”	in-place 情况下(src = dst)需要

开运算与闭运算

表 5-3 中的前两个操作开运算和闭运算包含腐蚀和膨胀操作。在开运算的情况下，我们首先将其腐蚀然后再膨胀(图 5-10)。开运算通常可以用来统计二值图像中的区域数。若已将显微镜载玻片上观察到的细胞图像作了阈值化处理，可以使用开运算将相邻的细胞分离开来，然后再计算图像中的区域(细胞)数目。在闭运算的情况下，我们首先将其膨胀然后再腐蚀(图 5-12)。在大多数好的连通区域分析算法中，都会用到闭运算来去除噪声引起的区域。对于连通区域分析，通常先采用腐蚀或闭运算来消除纯粹由噪声引起的部分，然后用开运算来连接邻近的区域。(注意，虽然使用开运算或闭运算的结果与使用腐蚀和膨胀的结果类似，但这两个新的操作能更精确地保存源图像连接的区域。)

【120 ~ 121】

开运算和闭运算操作几乎都是“保留区域”形式的：最显著的效果是，闭运算消除了低于其邻近点的孤立点，而开运算是消除高于其邻近点的孤立点。开运算的结果如图 5-11 所示，闭运算的结果如图 5-13 所示。

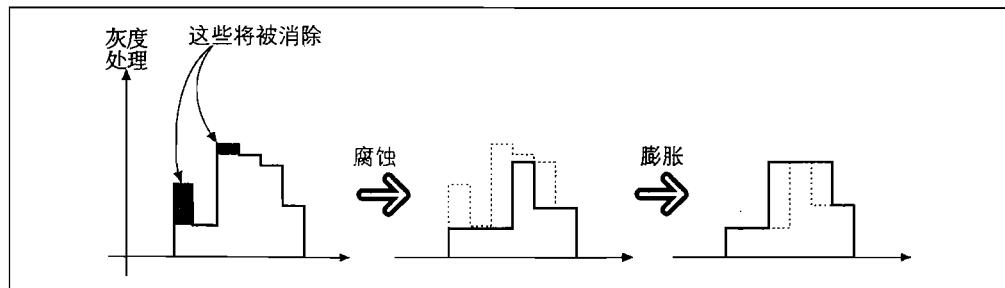


图 5-10: 形态的开运算(向上的孤立点被消除)

开运算和闭运算操作几乎都是“面积保持”形式的：最显著的效果是，闭运算消除了低于其邻近点的孤立点，而开运算是消除高于其邻近点的孤立点。开运算的结果

如图 5-11 所示，闭运算的结果如图 5-13 所示。



图 5-11：形态开运算的结果(去除小的明亮区域，并且剩余的明亮区域被隔绝，但其大小不变)

关于开运算和闭运算，最后一点需要说明的是 `iterations` 参数的意思。您可能会认为闭操作执行两次，相当于执行膨胀—腐蚀—膨胀—腐蚀。但事实上，这并不是必要的。真正需要的(并且所能得到的)是膨胀—膨胀—腐蚀—腐蚀这样的过程。通过这种方式，不仅是单一的孤立点会消失，而且邻近的孤离点群也会消失。

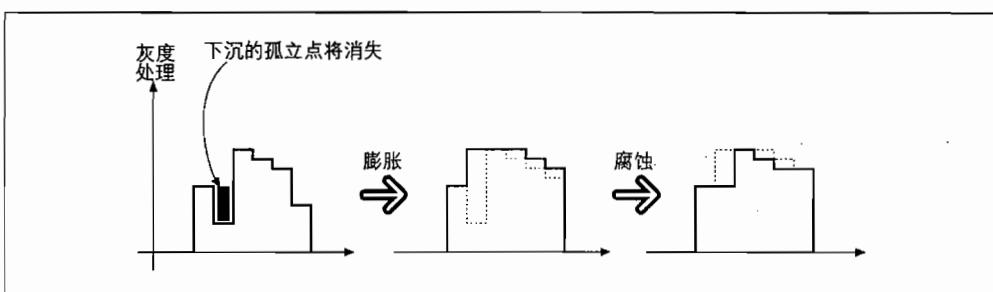


图 5-12：形态闭运算的操作(消除低亮度值的孤立点)



图 5-13：形态学闭运算的结果(亮的区域连在一起，但他们基本的大小不变)

形态学梯度

下一个可用的操作是形态学梯度。在这里我们先给出其公式，然后再解释公式的含义：

```
gradient(src) = dilate(src)-erode(src)
```

对二值图像进行这一操作可以将团块(blob)的边缘突出出来。图 5-14 解释了这一操作，在测试图像上进行操作的效果如图 5-15 所示。

【121】

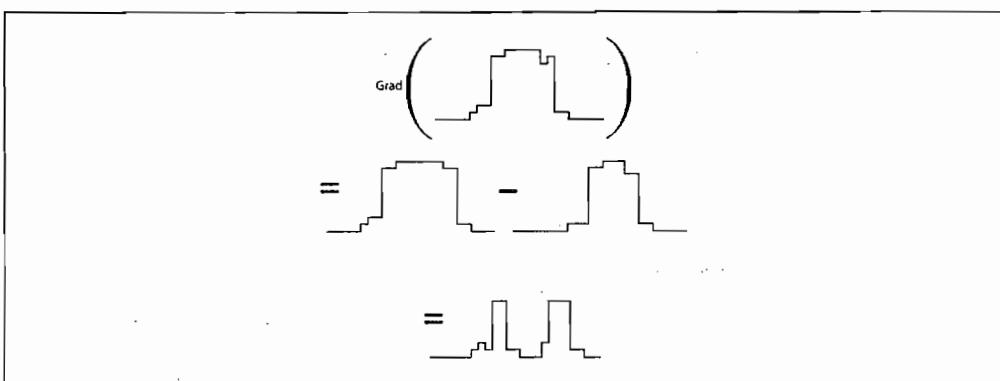


图 5-14：形态梯度被应用于灰度图(正如所料，在灰度值变化最剧烈的区域得到的结果数值最大)

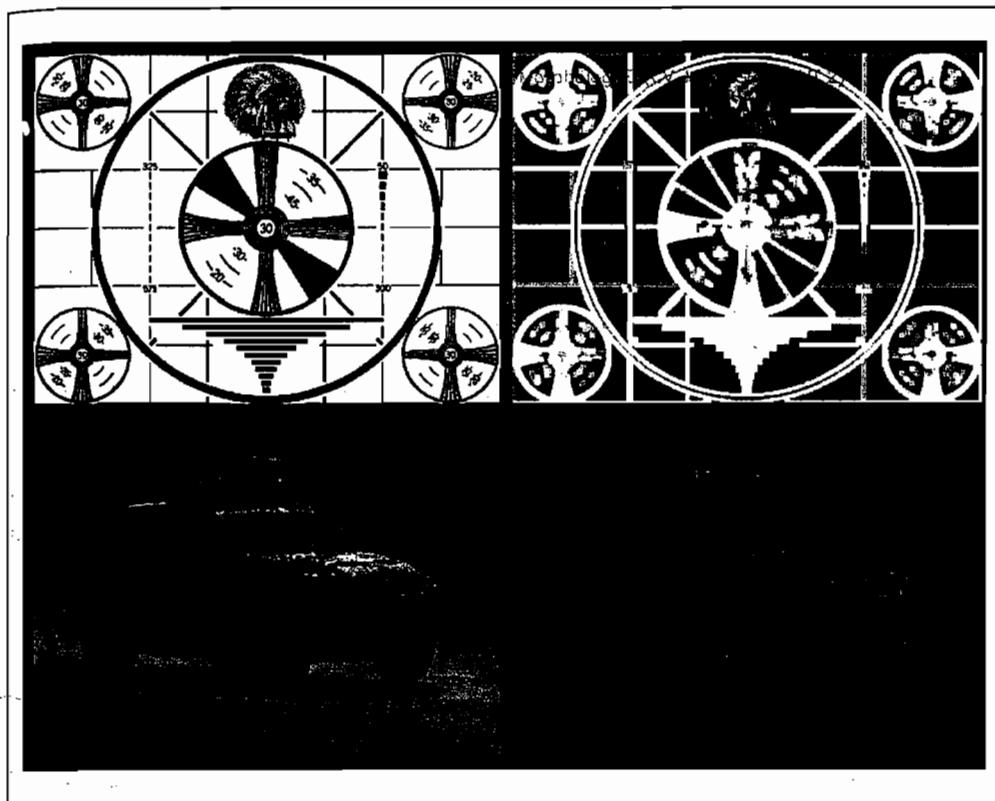


图 5-15：形态学梯度操作结果(边缘以高亮区域突出)

观察对灰度图像的处理结果，可以获知形态学梯度操作能描述图像亮度变化的剧烈程度；这就是为什么把其称为“形态学梯度”的原因。当我们想突出高亮区域的外围时，通常可使用形态学梯度，这样我们可以把高亮的看成一个整体(或物体的一整部分)。因为从原区域的膨胀中减去了原区域的收缩，所以留下了完整的外围边缘。这与计算梯度有所不同，梯度一般不能获得物体的外围边缘^①。【122~123】

礼帽和黑帽

最后两个操作被称为礼帽(Top Hat)和黑帽变换(Black Hat)[Meyer78]。这些操作分别用于分离比邻近的点亮或暗的一些斑块。当试图孤立的部分相对于其邻近的部分有亮度变化时，就可以使用这些方法。例如常用与处理有机组织或细胞的显微镜图

① 第6章介绍 Sobel 和 Scharr 时将再一次讨论梯度。

像。这是两个操作都是基本的操作组合，定义如下：

$$\text{TopHat(src)} = \text{src} - \text{open(src)}$$

$$\text{BlackHat(src)} = \text{close(src)} - \text{src}$$

可以看出，礼帽操作从 A 中减去了 A 的开运算。开运算带来的结果是放大裂缝或局部低亮度区域，因此，从 A 中减去 open(A) 可以突出比 A 周围的区域更明亮的区域，并跟核的大小相关(参见图 5-16)；相反地，黑帽操作突出比 A 的周围的区域黑暗的区域(图 5-17)。在本章讨论的所有形态操作的结果均可参见图 5-18^①。

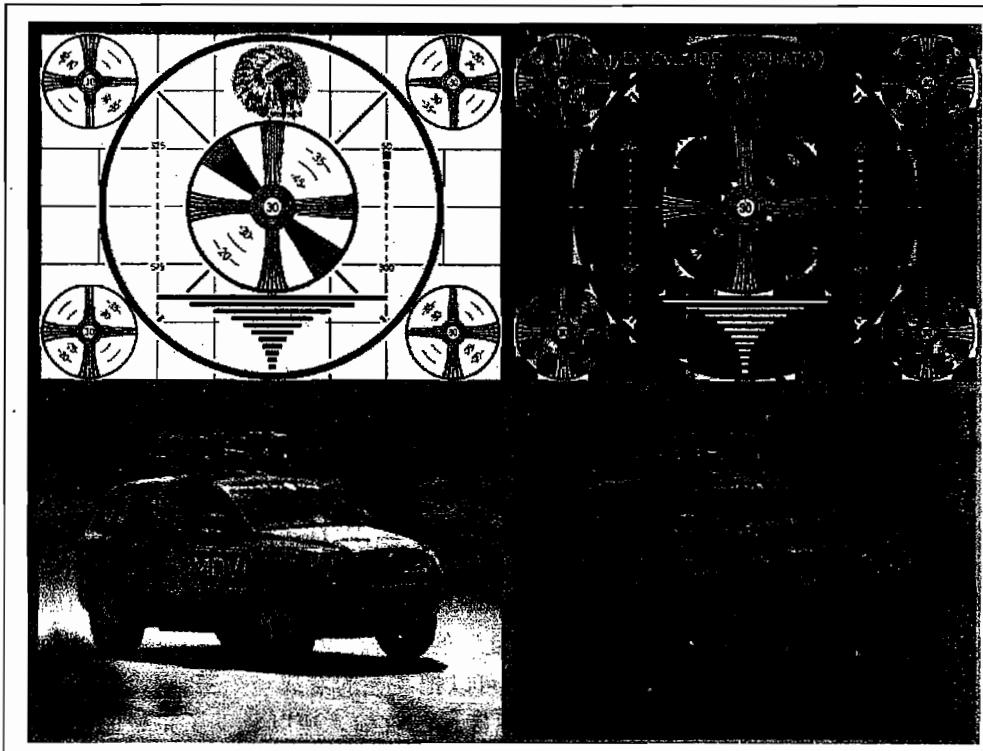


图 5-16：形态学“礼帽”操作的效果(局部亮度极大点被分割出)

① 这两项操作(礼帽和黑帽)在灰度形态学中有着重要的意义，结构元素是一个实数的矩阵(而不只是一个二值掩码)，而且可以在取最大或最小值前，将此矩阵加入到当前像素的邻域中。遗憾的是，在 OpenCV 中这还没有实现。

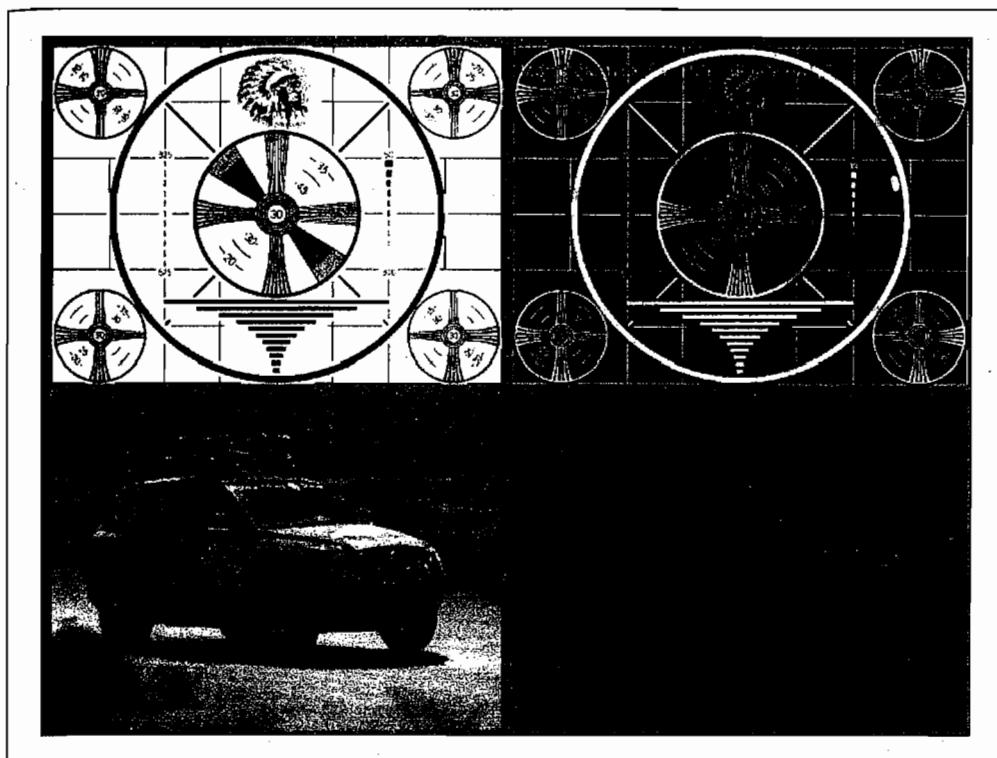


图 5-17：形态学“黑帽”操作的效果(黑色“洞”被分割出)

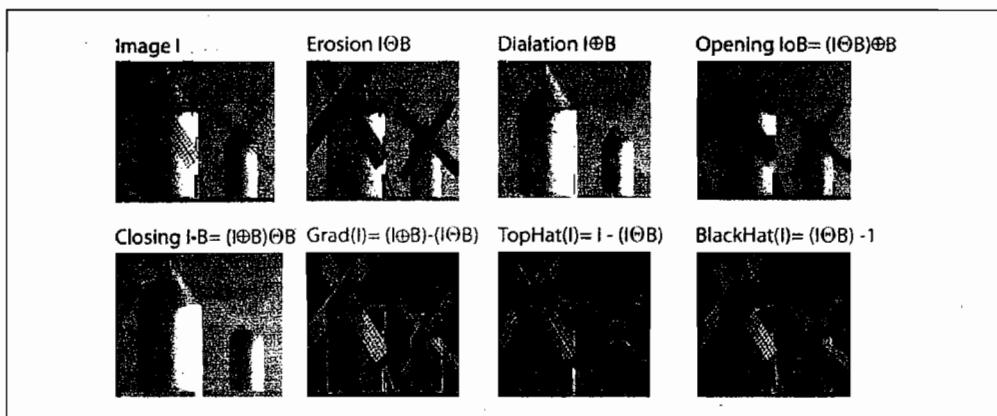


图 5-18：所有形态学算子操作的效果汇总

【123 ~ 124】

漫水填充算法

漫水填充(Flood Fill)[Heckbert00; Shaw04; Vandevenne04]是一个非常有用的功能，它经常被用来标记或分离图像的一部分以便对其进行进一步处理或分析。漫水填充也可以用来从输入图像获取掩码区域，掩码会加速处理过程，或只处理掩码指定的像素点。cvFloodFill 函数本身也包含一个可选的掩码参数，用来进一步控制哪些区域将被填充颜色(例如当对同一图像进行多次填充时)。

在 OpenCV 里，漫水填充是填充算法中最通用的方法，也许你看到这里已经将其与典型的计算机绘图程序联系起来。对于两种程序来说，都必须在图像上选择一个种子点，然后把邻近区域所有相似点填充上同样的颜色，不同的是不一定将所有的邻近像素点都被染成同一颜色^①，漫水填充操作的结果总是某个连续的区域。当邻近像素点位于给定的范围(从 loDiff 到 upDiff)内或在原始 seedPoint 像素值范围内时，cvFloodFill() 函数将为这个点涂上颜色。可选参数 mask 也可以用来控制漫水法填充。该填充方法的函数原型如下所示：

```
void cvFloodFill(
    IplImage*           img,
    CvPoint             seedPoint,
    CvScalar            newVal,
    CvScalar            loDiff      = cvScalarAll(0),
    CvScalar            upDiff      = cvScalarAll(0),
    CvConnectedComp*    comp        = NULL,
    int                 flags       = 4,
    CvArr*              mask        = NULL
);
```

img 参数代表输入图像，该图像可以是 8 位或浮点类型的单通道或三通道图像。漫水法填充从点 seedPoint 开始，newVal 是像素点被染色的值。如果一个像素点的值不低于被染色的相邻点减去 loDiff 且不高于其加上 upDiff，那么该像素点就会被染色。如果 flags 参数包含 CV_FLOODFILL_FIXED_RANGE，这时每个像素点都将与种子点而不是相邻点相比较。如果 comp 不是 NULL，那

① 使用当前流行的绘图程序的用户会注意到，大多数填充算法都与 cvFloodFill() 函数类似。

么该 CvConnectedComp 结构将被设置为被填充区域的统计属性^①。flags 参数(下文会有简短论述)有些复杂，这些参数决定填充的连通性、相关性、是否只填充掩码区域及用来填充的值。我们第一个漫水填充例子见图 5-19。

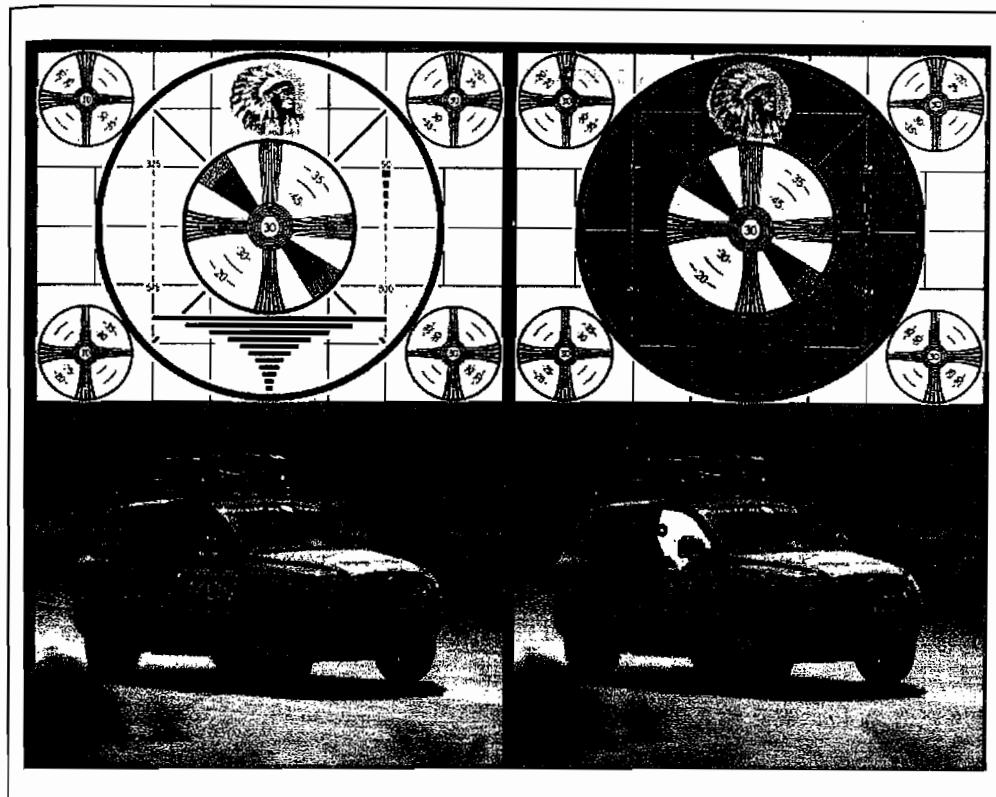


图 5-19：漫水填充的效果(上方图像用灰色填充，下方图像用白色填充)，从位于两个图像中心旁的黑色圆形区域开始填充；此处 hiDiff 与 loDiff 参数均设为 7.0

这里 mask 参数所代表的掩码既可以作为 cvFloodFill() 函数的输入值(此时它控制可以被填充的区域)，也可以作为 cvFloodFill() 函数的输出值(此时它指已经被填充的区域)。如果 mask 非空，那么它必须是一个单通道、8 位、像素宽度和高度均比源图像大两倍的图像(这是为了使内部运算更简单快速)。mask 图像的像素($x + 1, y + 1$)与源图像的像素(x, y)相对应。注意，cvFloodFill() 不会覆盖 mask 的非 0 像

① 在“图像金字塔”一节将对“连通分量”进行具体讲解，现在将其标识图像分子区域的掩码即可。

素点，因此如果不希望 mask 阻碍填充操作时，将其中元素设为 0。源图像 img 和掩码图像 mask 均可以用漫水填充来染色。

【125~127】

注意：如果漫水填充的掩码不为空，那么要用 flags 参数的中间比特值(第 8~15 位)来填充掩码图像(参考下文)。如果没有设置值 flags 中间比特值，则取默认值 1。如果填充了掩码后显示出来是黑色，不要感到奇怪，因为所设置的值(如果 flags 的中间值没有被设置)为 1，所以如果要显示它，必须需要放大这个掩码图像的数值。

下面讲一下 flags 参数。此参数包含三部分，因此它比较复杂。低 8 位部分(第 0~7 位)可以设为 4 或 8，这个参数控制填充算法的连通性。如果设为 4，填充算法只考虑当前像素水平方向和垂直方向的相邻点；如果设为 8，除上述相邻点外，还会包含对角线方向的相邻点。高 8 位部分(第 16~23 位)可以设为 CV_FLOODFILL_FIXED_RANGE(如果设置为这个值，则只有当某个相邻点与种子像素之间的差值在指定范围内才填充，否则考虑当前点与其相邻点的差是否落在指定范围)或者 CV_FLOODFILL_MASK_ONLY(如果设置，函数不填充原始图像，而去填充掩码图像)。很明显，如果设为 CV_FLOODFILL_MASK_ONLY，必须输入符合要求的掩码。flags 的中间比特(第 8~15 位)的值指定填充掩码图像的值。但如果中间比特值为 0，则掩码将用 1 填充。所有 flags 可以通过 OR 操作连接起来。例如，如果想用 8 邻域填充，并填充固定像素值范围，是填充掩码而不是填充源图像，以及设填充值为 47，那么输入的参数应该是：

```
flags = 8  
| CV_FLOODFILL_MASK_ONLY  
| CV_FLOODFILL_FIXED_RANGE  
| (47<<8);
```

图 5-20 显示了对示例图像的填充操作结果。在这个填充中使用了 CV_FLOODFILL_FIXED_RANGE 并选择了较大的范围，结果图像的大部分区域被填充(从中心处开始填充)。注意，newVal, loDiff 和 upDiff 都是 CvScalar 的类型，所以它们可以同时处理三个通道(可以通过 CV_RGB() 宏设置 RGB 三色值)。例如令 lowDiff = CV_RGB(20, 30, 40)，则三种颜色的 lowDiff 分别设为红色值 20，绿色值 30，蓝色值 40。

【128~129】

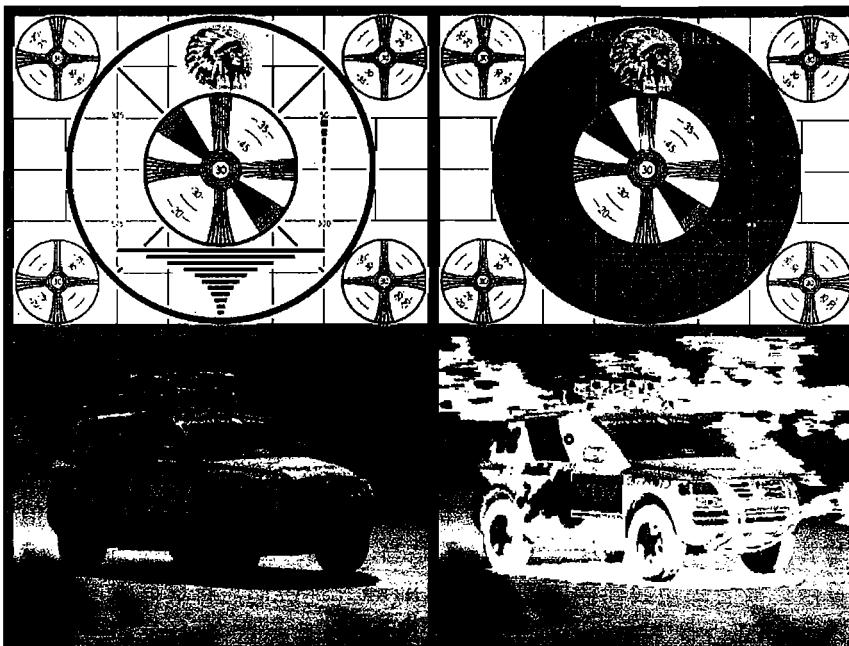


图 5-20：漫水填充的效果(上方图像用灰色填充，下方图像用白色填充)，填充从位于两个图像中心旁的黑色圆形区域开始；此处指定颜色填充法是在固定范围内进行的，`loDiff` 和 `upDiff` 的取值都是 25.0

尺寸调整

我们经常会将某种尺寸的图像转换为其他尺寸的图像，如放大或者缩小图像。我们可以用 `cvResize()` 函数来放大或缩小图像。该函数可以将源图像精确转换为目标图像的尺寸。如果源图像中设置了 ROI，那么 `cvResize()` 将会对 ROI 区域调整尺寸，以匹配目标图像，同样，如果目标图像中已设置 ROI 的值，那么 `cvResize()` 将会将源图像进行尺寸调整并填充到目标图像的 ROI 中。

```
void cvResize(  
    const CvArr* src,  
    CvArr*      dst,  
    int         interpolation = CV_INTER_LINEAR  
) ;
```

【129~130】

最后一个参数指定插值方法，默認為线性插值法。可用的插值方法如表 5-4 所示。

表 5-4: cvResize() 插值方法

插值方法	含义
CV_INTER_NN	最近邻插值
CV_INTER_LINEAR	线性插值
CV_INTER_AREA	区域插值
CV_INTER_CUBIC	三次样条插值

一般情况下，我们期望源图像和重采样后的目标图像之间的映射尽可能地平滑。参数 `interpolation` 控制如何进行映射。当缩小图像时，目标图像的像素会映射为源图像中的多个像素，这时需要进行插值。当放大图像时，目标图像上的像素可能无法在源图像中找到精确对应的像素，也需要进行插值。在任何一种情况下，都有很多 z 种计算像素值的方法。其中最简单的办法是将目标图像各点的像素值设为源图像中与其距离最近的点的像素值，这就是当 `interpolation` 设为 `CV_INTER_NN` 时用的算法。或者采用线性插值算法(`CV_INTER_LINEAR`)，将根据源图像附近的 4 个(2×2 范围)邻近像素的线性加权计算得出，权重由这 4 个像素到精确目标点的距离决定。我们也可以用新的像素点覆盖原来的像素点，然后求取覆盖区域的平均值，这种插值算法称为区域插值。^①最后一种选择是三次样条插值(`CV_INTER_CUBIC`)。首先对源图像附近的 4×4 个邻近像素进行三次样条拟合，然后将目标像素对应的三次样条值作为目标图像对应像素点的值。

图像金字塔

图像金字塔[Adelson84]被广泛用于各种视觉应用中。图像金字塔是一个图像集合，集合中所有的图像都源于同一个原始图像，而且是通过对原始图像连续降采样获得，直到达到某个中止条件才停止降采样。(当然，降为一个像素肯定是中止条件。)

【130】

有两种类型的图像金字塔常常出现在文献和应用中：高斯金字塔[Rosenfeld80]和拉普拉斯[Burt83]金字塔[Adelson84]。高斯金字塔用来向下降采样图像，而拉普拉斯金字塔(后面会简单讨论)则用来从金字塔低层图像中向上采样重建一个图像。

① 至少用 `cvResize()` 函数来缩小图像时是这样的。放大图像时，`CV_INTER_AREA` 方法与 `CV_INTER_NN` 方法相同。

要从金字塔第 i 层生成第 $i+1$ 层(我们表示第 $i+1$ 层为 G_{i+1})，我们先要用高斯核对 G_i 进行卷积，然后删除所有偶数行和偶数列。当然，新得到的图像面积会变为源图像的四分之一。按上述过程对输入图像 G_0 循环执行操作就可产生整个金字塔。OpenCV 为我们提供了从金字塔中上一级图像生成下一级图像的方法：

```
void cvPyrDown(
    IplImage*    src,
    IplImage*    dst,
    IplFilter    filter = IPL_GAUSSIAN_5x5
);
```

目前，最后一个参数 `filter` 仅支持 `CV_GAUSSIAN_5x5`(默认选项)。

同样，我们可以通过下面相似的函数(但不是降采样的逆操作！)将现有的图像在每个维度上都放大两倍：

```
void cvPyrUp(
    IplImage*    src,
    IplImage*    dst,
    IplFilter    filter = IPL_GAUSSIAN_5x5
);
```

在这种情况下，图像首先在每个维度上扩大为原来的两倍，新增的行(偶数行)以 0 填充。然后给指定的滤波器进行卷积(实际上是一个在每一维上都扩大为两倍的过滤器)^①去估计“丢失”像素的近似值。

我们之前注意到函数 `PyrUp()` 并不是函数 `PyrDown()` 的逆操作。之所以这样是因为 `PyrDown()` 是一个会丢失信息的函数。为了恢复原来(更高的分辨率)的图像，我们需要获得由降采样操作丢失的信息。这些数据形成了拉普拉斯金字塔。下面是拉普拉斯金字塔的第 i 层的数学定义：

$$L_i = G_i - \text{UP}(G_{i+1}) \otimes \mathcal{G}_{5 \times 5}$$

这里的 `UP()` 操作将原始图像中位置为 (x, y) 的像素映射到目标图像的 $(2x+1, 2y+1)$ 位置；符号 \otimes 代表卷积操作， $\mathcal{G}_{5 \times 5}$ 是 5×5 高斯核。OpenCV 提供的函数 `PyrUp()` 实现的功能就如 $G_i - \text{UP}(G_{i+1}) \otimes \mathcal{G}_{5 \times 5}$ 所定义。因此，我们可以使用 OpenCV 直接进行拉普拉斯运算：

① 此滤波器中所有元素的和被规范化为 4，而不是 1。因为在进行卷积操作前，插入行的像素值都为 0，所以这是合理的。

$$L_i = G_i - \text{PyrUp}(G_{i+1})$$

【131~132】

高斯金字塔和拉普拉斯金字塔如图 5-21 所示，这也显示了从小图恢复原始图像这个逆过程。注意：拉普拉斯是就像在前面方程中和图表中揭示的那样，它可通过高斯进行近似。

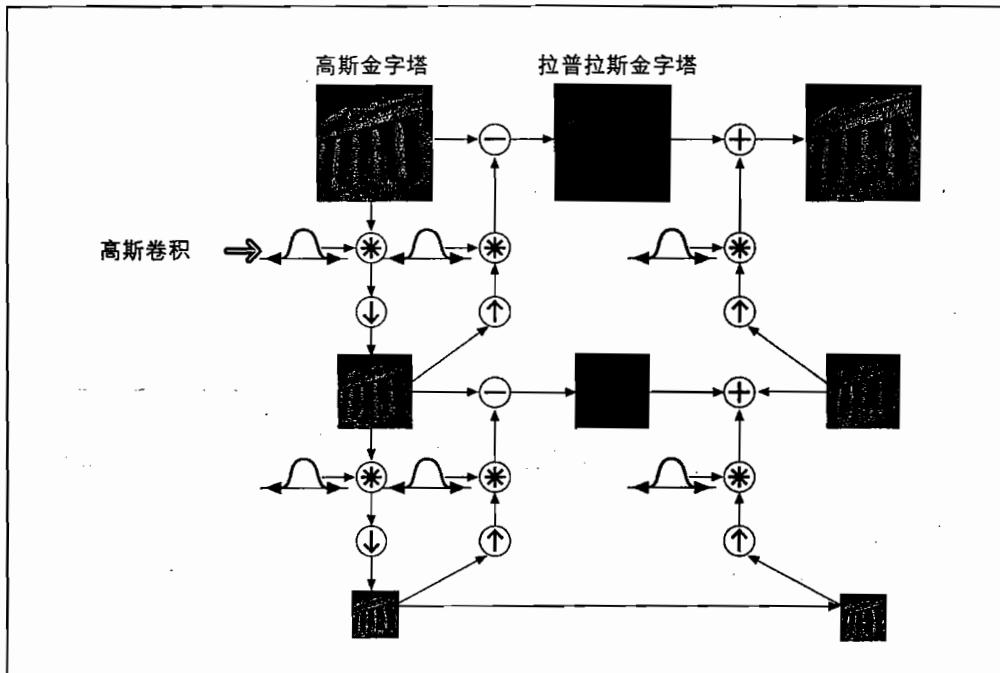


图 5-21：高斯金字塔及其逆形式——拉普拉斯金字塔

有许多操作广泛使用高斯金字塔和拉普拉斯金字塔，但一个特别重要的应用就是利用金字塔实现图像分割(见图 5-22)。图像分割需要先建立一个图像金字塔，然后在 G_i 的像素和 G_{i+1} 的像素直接依照对应关系，建立起“父—子”关系。通过这种方式，快速初始分割可以先在金字塔高层的低分辨率图像上完成，然后逐层对分割加以优化。

OpenCV 的函数 `cvPyrSegmentation()` 实现了该算法(可参考 B.Jaehne 的论文 [Jaehne95; Antonisse82])：

```
void cvPyrSegmentation(
    IplImage*          src,
    IplImage*          dst,
    CvMemStorage*      storage,
```

```

CvSeq** comp,
int level,
double threshold1,
double threshold2
);

```

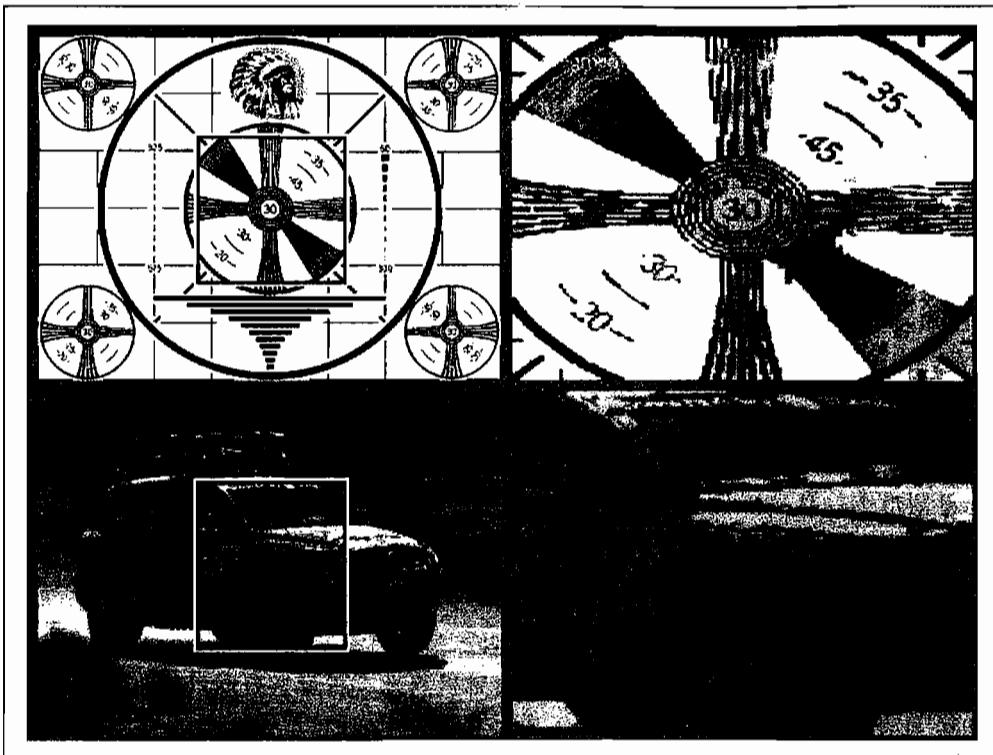


图 5-22：金字塔分割的阈值 `threshold1` 设置为 150，阈值 `threshold2` 设置为 30；右边的图像只包含左边图像的一部分，因为用金字塔分割图像需要将图像进行 n 次降采样，其中 n 是要计算出的金字塔的层数(右图都是从源图像分割出的 512×512 大小的区域)

像以前介绍的一样，`src` 和 `dst` 分别是源图像和目标图像，它们都必须是 8 位的，且具有相同的图像大小和通道的数量(1 或 3)。大家可能会想：“会产生什么样的目标图像？”这个问题并不是没有道理。目标图像 `dst` 可作为执行算法时的所需的临时空间，并对分割结果可视化。如果观察此图像，可以看出每个分割出的区域都被涂为单一的颜色(分割区域中像素的颜色)。由于这个图像是执行算法时所需的临时计算空间，所以不能设为 `NULL`；即使不想要分割结果，也必须提供一个图像。关于 `src` 和 `dst`，需要特别注意一点：由于图像金字塔各层的长和宽都必须是整

数，所以必须要求起始图像的长和宽都能够被 2 整除，并且能够被 2 整除的次数不少于金字塔总的层数。例如，对于 4 层金字塔的高度或宽度为 80($2 \times 2 \times 2 \times 5$)是满足要求的，而为 90 时($2 \times 3 \times 3 \times 5$)就不符合要求了^①。

【132~133】

指针 storage 用来指向 OpenCV 的存储区。第 8 章将详细讨论此内容，但现在应该知道一点，以下命令可以分配存储区域^②：

```
CvMemStorage * storage = cvCreateMemStorage( );
```

comp 参数用于存储分割结果更详细的信息——存储区里一序列相连的组成部分。具体工作机制将在第 8 章详细讲解。但为方便起见，这里简要介绍一下 cvPyrSegmentation() 有关的知识。

首先要强调的是，序列(Sequence)是某个特定类型的结构列表。给定一个序列，如果已知元素的类型及其在序列中的位置，便可以获得此序列中元素的个数以及这个元素。访问序列的方法如例 5-1 所示。

例 5-1：对序列中的每个元素进行操作，此序列的元素是由 cvPyrSegmentation() 返回的连续区域。

```
void f(
    IplImage* src,
    IplImage* dst
) {
    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSeq* comp = NULL;
    cvPyrSegmentation( src, dst, storage, &comp, 4, 200, 50 );
    int n_comp = comp->total;
    for( int i=0; i<n_comp; i++ ) {
        CvConnectedComp* cc = (CvConnectedComp*) cvGetSeqElem(comp, i);
        do_something_with( cc );
    }
    cvReleaseMemStorage( &storage );
}
```

在例有以下几点应该注意。首先，观察存储的分配情况：cvPyrSegmentation() 函数从中为它要创建的连续区域申请内存。然后将指针类型设为 CvSeq*，并被初

-
- ① 一定要留意上面提到的要求！否则，你会得到一个毫无用处的错误消息，而且会花费很多时间去了解错误的原因。
 - ② 实际上，目前的 cvPyrSegmentation() 函数并不完善，因为它返回的不是操作后的分割，而只是分割区域的外接矩形(是 CvSeq <CvConnectedComp>)。

始化为 NULL，因为其当前值意味着无内容。我们将传递指向 comp 的指针给 cvPyrSegmentation() 函数，这样 comp 就可以设置为 cvPyrSegmentation() 创建序列的位置。调用分割操作之后，通过成员变量 total，就会知道序列中的元素个数。随后，我们可以使用通用的 cvGetSeqElem() 以获取 comp 的第 i 个元素；然而，由于 cvGetSeqElem() 是通用的，所以我们必须将返回的指针强制转化为适当的类型(这里是 CvConnectedComp* 类型)。

【134】

最后，我们需要知道连续区域是 OpenCV 中的基本结构类型之一。可以把它视为描述图像中“团块”(blob)的一种方法。它具有以下定义：

```
typedef struct CvConnectedComponent {  
    double area;  
    CvScalar value;  
    CvRect rect;  
    CvSeq* contour;  
};
```

【134~135】

参数 area 是区域的面积。参数 value 是区域颜色的平均值^①，参数 rect 是一个区域的外接矩形(定义在父图像的坐标系中)。最后一个参数 contour 是一个指向另一个序列的指针。这个序列可以用来储存区域的边界，通常是点的序列(元素类型为 CvPoint)。

在 cvPyrSegmentation() 函数中，没有设参数 contour 的值。因此，如果想要区域中的像素，就只能自己计算。当然得根据自己的想法来选择函数。通常希望获得一个二值掩码图像，连续区域由掩码中的非 0 元素指定。可以使用连续区域的 rect 作为掩码，然后使用 cvFloodFill() 来选择矩形内所需的像素。

阈值化

完成许多处理步骤之后，通常希望对图像中的像素做出最后的决策，或直接剔除一些低于或高于一定值的像素。在 OpenCV 中，函数 cvThreshold() 可以完成这些任务(见综述[Sezgin04])。其基本的思想是，给定一个数组和一个阈值，然后根据数组中的每个元素的值是低于还是高于阈值而进行一些处理。

① 其实 value 的含义依赖于上下文的内容，并可以是任何内容的，但它通常是与区域相关的颜色。在 cvPyrSegmentation() 中，value 是分割区域的颜色平均值。

```

double cvThreshold(
    CvArr*    src,
    CvArr*    dst,
    double    threshold,
    double    max_value,
    int       threshold_type
);

```

如表 5-5 所示，每个阈值类型对应于一个特定的比较操作，该比较操作在源图像第 i 个像素(src_i)和阈值(表中表示为 T)之间进行。根据源图像的像素和阈值之间的关系，目标图像的像素 dst_i 可能被设置为 0, src_i 或 max_value (表中表示为 M)。

【135】

表 5-5: cvThreshold()中阈值类型选项和对应的操作

阈值类型	操作
CV_THRESH_BINARY	$dst_i = (src_i > T) ? M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) ? 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) ? M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) ? 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) ? src_i : 0$

图 5-23 有助于我们理解关于每一个阈值类型的确切操作。

我们来看一个简单的例子。在例 5-2 中，我们对图像中的三个通道求和，然后在值为 100 处对结果图像进行截断。

例 5-2: cvThreshold()函数的用法

```

#include <stdio.h>
#include <cv.h>
#include <highgui.h>
void sum_rgb( IplImage* src, IplImage* dst ) {

    // Allocate individual image planes.
    IplImage* r = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
    IplImage* g = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
    IplImage* b = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );

    // Split image onto the color planes.
    cvSplit( src, r, g, b, NULL );
}

```

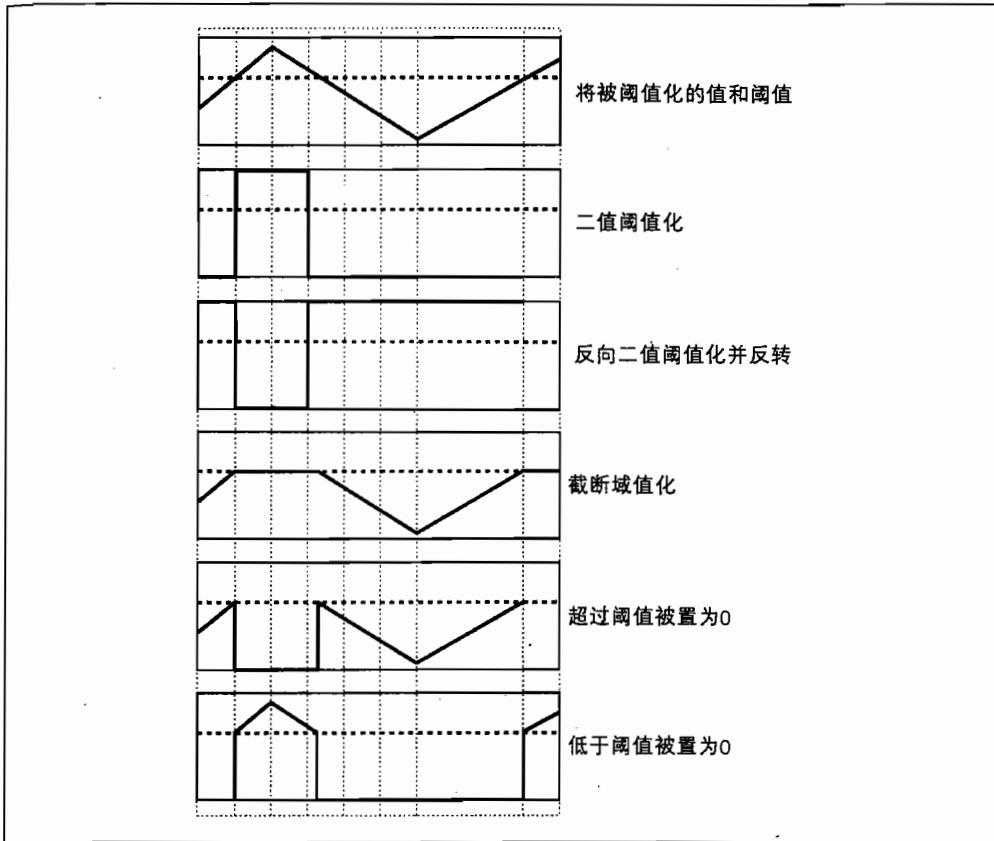


图 5-23：在 cvThreshold () 中不同阈值类型的操作结果。每个图表的水平虚线代表应用到最上方图的阈值，5 种阈值类型的操作结果列于其后 【136】

```

// Temporary storage.
IplImage* s = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );

// Add equally weighted rgb values.
cvAddWeighted( r, 1./3., g, 1./3., 0.0, s );
cvAddWeighted( s, 2./3., b, 1./3., 0.0, s );

// Truncate values above 100.
cvThreshold( s, dst, 100, 100, CV_THRESH_TRUNC );

cvReleaseImage( &r );
cvReleaseImage( &g );
cvReleaseImage( &b );
cvReleaseImage( &s );

```

```

}

int main(int argc, char** argv)
{
    // Create a named window with the name of the file.
    cvNamedWindow( argv[1], 1 );

    // Load the image from the given file name.
    IplImage* src = cvLoadImage( argv[1] );
    IplImage* dst = cvCreateImage( cvGetSize(src), src->depth, 1 );
    sum_rgb( src, dst );

    // Show the image in the named window
    cvShowImage( argv[1], dst );

    // Idle until the user hits the "Esc" key.
    while( 1 ) { if( (cvWaitKey( 10 )&0x7f) == 27 ) break; }

    // Clean up and don't be piggies
    cvDestroyWindow( argv[1] );
    cvReleaseImage( &src );
    cvReleaseImage( &dst );
}

```

【137~138】

这里包含几个重要的思想。第一，我们通常不会对 8 位数组进行加法运算，因为较高的位可能会溢出。取而代之，我们使用加权加法算法(`cvAddWeighted()`)对三个通道求和；然后对结果以 100 为阈值进行截断处理然后返回。`cvThreshold()`函数只能处理 8 位或浮点灰度图像。目标图像必须与源图像的类型一致，或者为 8 位图像。事实上，`cvThreshold()`还允许源图像和目标图像是同一图像。如果我们在例 5-2 中使用了临时浮点图像 `s`，例 5-3 中有等价的代码。注意，`cvAcc()`可以将 8 位整数类型图像累加为浮点图像；然而，`cvADD()`却不能将整数与浮点数相加。

例 5-3：另一种组合不同通道，并阈值化图像的方法

```

IplImage* s = cvCreateImage(cvGetSize(src), IPL_DEPTH_32F, 1);
cvZero(s);
cvAcc(b,s);
cvAcc(g,s);
cvAcc(r,s);
cvThreshold( s, s, 100, 100, CV_THRESH_TRUNC );

```

```
cvConvertScale( s, dst, 1, 0 );
```

自适应阈值

这是一种改进了的阈值技术，其中阈值本身是一个变量。在 OpenCV 中，这种方法由函数 cvAdaptiveThreshold() [Jain86] 来实现：

```
void cvAdaptiveThreshold(
    CvArr*      src,
    CvArr*      dst,
    double      max_val,
    int         adaptive_method= CV_ADAPTIVE_THRESH_MEAN_C
    int         threshold_type = CV_THRESH_BINARY,
    int         block_size     = 3,
    double      param1        = 5
);
```

cvAdaptiveThreshold() 有两种不同的自适应阈值方法，可以用参数 adaptive_method 进行设置。在这两种情况下，自适应阈值 $T(x, y)$ 在每个像素点都不同。通过计算像素点周围的 $b \times b$ 区域的加权平均，然后减去一个常数来得到自适应阈值， b 由参数 block_size 指定，常数由 param1 指定。如果使用 CV_ADAPTIVE_THRESH_MEAN_C 方法，那么对区域的所有像素平均加权。如果使用了 CV_ADAPTIVE_THRESH_GAUSSIAN_C 方法，那么区域中的 (x, y) 周围的像素根据高斯函数按照它们离中心点的距离进行加权计算。

【138】

最后，参数 threshold_type 和表 5-5 所示的 cvThreshold() 的参数 threshold_type 是一样的。

针对有很强照明或反射梯度的图像，需要根据梯度进行阈值化时，自适应阈值技术非常有用。此函数只能处理单通道 8 位图像或浮点图像，它要求源图像和目标图像不能使用同一图像。

使用函数 cvAdaptiveThreshold() 和 cvThreshold() 进行比较的源代码在例 5-4 中。图 5-24 显示了对有很强的照明梯度图像的处理结果。图的左下部分显示了使用单一的全局阈值方法 cvThreshold() 的结果；图的右下部分显示了使用自适应局部阈值 cvAdaptiveThreshold() 的结果。我们通过自适应阈值得到了整个棋盘格，而在使用单一的阈值时是不能获得。请注意例 5-4 中代码顶部的使用说明；图 5-24 所用的参数如下：

```
./adaptThresh 15 1 1 71 15 ..../Data/cal3-L.bmp
```

【139】

例 5-4：单一阈值与自适应阈值

```
// Compare thresholding with adaptive thresholding
// CALL:
// ./adaptThreshold      Threshold 1binary 1adaptiveMean \
//                                blocksize offset filename
#include "cv.h"
#include "highgui.h"
#include "math.h"
IplImage *Igray=0, *It = 0, *Iat;
int main( int argc, char** argv )
{
    if(argc != 7){return -1; }

    //Command line
    double threshold = (double)atof(argv[1]);
    int threshold_type = atoi(argv[2]) ?
        CV_THRESH_BINARY : CV_THRESH_BINARY_INV;
    int adaptive_method = atoi(argv[3]) ?
        CV_ADAPTIVE_THRESH_MEAN_C : CV_ADAPTIVE_THRESH_GAUSSIAN_C;
    int block_size = atoi(argv[4]);
    double offset = (double)atof(argv[5]);

    //Read in gray image
    if((Igray = cvLoadImage(argv[6],CV_LOAD_IMAGE_GRAYSCALE)) == 0){
        return -1; }

    // Create the grayscale output images
    It = cvCreateImage(cvSize(Igray->width,Igray->height),
                      IPL_DEPTH_8U, 1);
    Iat = cvCreateImage(cvSize(Igray->width,Igray->height),
                       IPL_DEPTH_8U, 1);
    //Threshold
    cvThreshold(Igray,It,threshold,255,threshold_type);
    cvAdaptiveThreshold(Igray, Iat, 255, adaptive_method,
                        threshold_type, block_size, offset);
    //PUT UP 2 WINDOWS
    cvNamedWindow("Raw",1);
    cvNamedWindow("Threshold",1);
    cvNamedWindow("Adaptive Threshold",1);
```

```

//Show the results
cvShowImage("Raw",Igray);
cvShowImage("Threshold",It);
cvShowImage("Adaptive Threshold",Iat);

cvWaitKey(0);
//Clean up
cvReleaseImage(&Igray);
cvReleaseImage(&It);
cvReleaseImage(&Iat);
cvDestroyWindow("Raw");
cvDestroyWindow("Threshold");
cvDestroyWindow("Adaptive Threshold");
return(0);
}

```

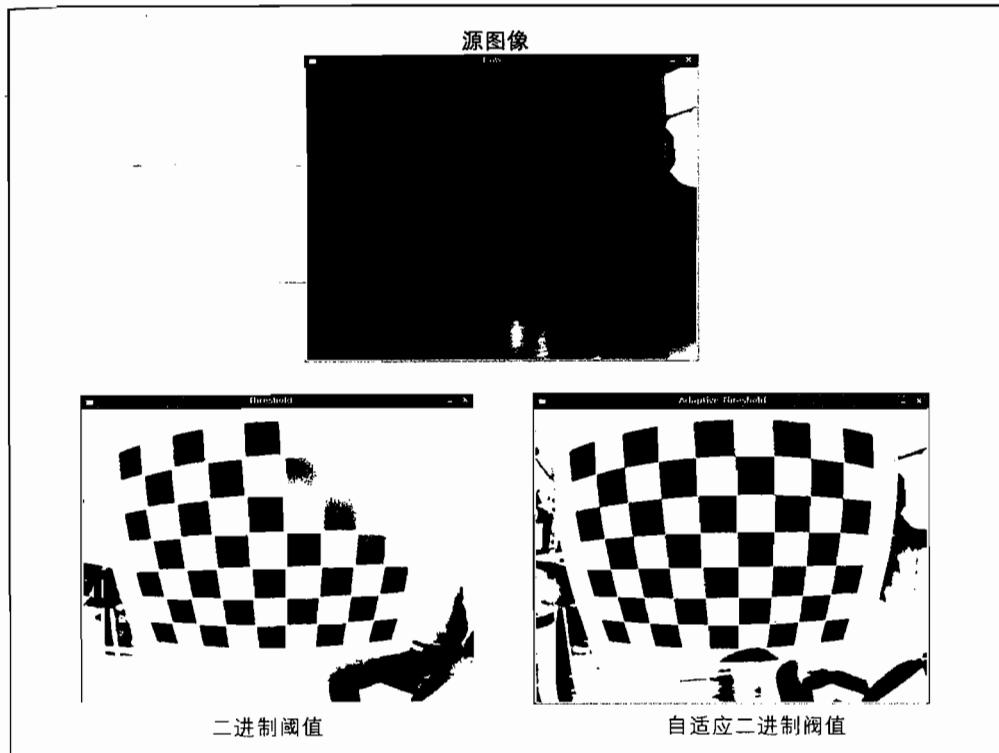


图 5-24：二值阈值化与自适应二值阈值化。输入图像(上图)使用全局阈值时的二值图像(左下图)和使用自适应阈值时的二值图像(右下图)[感谢 Kurt Konolidge 提供原始图像]

【140~141】

练习

1. 载入一个带有有趣纹理的图像。使用 `cvSmooth()` 函数以多种方法平滑图像，参数为 `smoothtype=CV_GAUSSIAN`。
 - a. 使用对称的平滑窗口，大小依次为 3×3 , 5×5 , 9×9 和 11×11 ，并显示出结果。
 - b. 用 5×5 高斯滤波器平滑图像两次和用两个 11×11 平滑器平滑一次的输出结果是接近相同吗？为什么？
2. 显示滤波器的效果。建立一个 100×100 的单通道图像。将图像全部像素置零，然后设置中心像素值等于 255。
 - a. 利用 5×5 高斯滤波器平滑此图像并显示结果。你发现了什么？
 - b. 改用 9×9 高斯滤波器重复 a 操作。
 - c. 如果你重新对原始图像用 5×5 过滤器平滑两次，会出现什么结果？与用 9×9 过滤器的结果对比，它们相似吗？为什么？
3. 加载一个有趣的图像。再次使用高斯滤波器通过 `cvSmooth()` 函数对其进行模糊操作。
 - a. 设 `param1=param2=9`。依次将 `param3` 设为几个不同的值(例如，1、4 和 6)，并显示结果。
 - b. 设 `param1=param2=0`，然后也设 `param3` 分别等于 1、4 和 6，并显示结果。这时和上题的结果有什么不同吗？为什么？
 - c. 再一次设 `param1=param2=0`，但这时令 `param3=1`, `param4=9`，对图像进行平滑操作并显示结果。
 - d. 令 `param3=9`, `param4=1`，重复操作 c，显示操作结果。
 - e. 将图像用 c 方法平滑一次，然后再用 d 方法平滑一次，显示结果。
 - f. 对图像依次进行两次平滑操作，第一次的参数为 `param1=param2=0`、`param3=param4=9`，第二次的参数为 `param1=param2=0`、`param3=param4=0`。将其结果与 e 操作的结果进行比较。它们相同吗？为什么？
4. 用相摄像机拍摄同一场景的两张照片，两张张照片的拍摄位置有轻微不同。将两张照片载入电脑，分别命名为 `src1` 和 `src2`。

- a. 将 `src1` 减去 `src2` 并求绝对值，将结果记为 `diff12` 并显示。在理想情况下，`diff12` 将是黑色的，为什么？
 - b. 对 `diff12` 先进行腐蚀操作 `cvErode()`，然后进行膨胀操作 `cvDilate()`，记结果为 `cleandiff`，并显示出来。
 - c. 对 `diff12` 先进行膨胀操作 `cvDilate()`，然后进行腐蚀操作 `cvErode()`，记结果为 `dirtydiff`，并显示出来。
 - d. 解释 `cleandiff` 与 `dirtydiff` 的区别。
5. 首先拍摄一张某场景的照片，然后摄像机不动，在此场景中心位置放一个咖啡杯，再拍摄一张照片，将其载入电脑并都转换为 8 位灰度图像。
- a. 取其差的绝对值并显示结果，它应该是一个带有噪声的咖啡杯掩码。
 - b. 对结果图像进行二值化阈值操作，剔除噪声的同时并保留咖啡杯。超过阈值的像素值应该设为 255。显示结果。
 - c. 在图像上进行 `CV_MOP_OPEN` 操作，以进一步清除噪声。
6. 从噪声图像中创建一个清晰的掩码。完成练习 5 后，保留图像中最大的图形区域。在图像的左上角设置一个指针，然后让它遍历图像。当你发现像素值为 255 的时候，存储其位置，然后对其进行漫水填充操作，新颜色值为 100。读出漫水填充法返回的连续区域，并记录下面积。如果图像中有另一个较大的区域，那么用 0 值对这个相对较小的区域对其进行颜色填充，然后删除已记录的面积。如果新的区域大于之前的区域，那么以 0 值填充之前的区域并删除它的位置。最后以颜色值 255 填充剩余的最大的区域。显示结果。现在你会得到一个实心的咖啡杯的掩码，它是惟一的连续区域。
7. 在这个练习里，使用练习 6 生成的掩码或者你自己制造的掩码(可以画张图像，或者简单的使用一个方形的区域)。载入一张外景图像，然后在 `cvCopy()` 函数中使用这个掩码，将一个杯子的图像复制到外景图像中。
8. 生成一个有较小方差的随机图像(用一个随机变量，使其大部分数值的差异不超过 3，并且大部分数值接近 0)。将此图像载入一个绘制程序例如 PowerPoint，在图像上画一些辐射状相交于一个交点的线条。对图像使用双边滤波，并解释所得到的结果。
9. 读入一幅风景图，然后将其转化为灰度图像。
- a. 对图像进行形态学“礼帽”操作，并显示其结果。
 - b. 将结果图像转换为 8 位的掩码。

- c. 复制灰度值到礼帽块中，显示结果。
10. 读入一幅具有很多细节的图。
- a. 利用 `cvResize()` 函数缩小图像，每个维度上缩小比例为 2(因此整张图像缩小比例为 4)重复三次并显示结果。
 - b. 下面利用 `cvPyrDown()` 函数在原始图像上进行三次降采样操作，显示结果。
 - c. 以上两个结果有何不同？使用的方法有何不同？
11. 载入一幅风景图。执行 `cvPyrSegmentation()` 操作并显示结果。
12. 载入一幅有趣或者场景丰富的图像，使用 `cvThreshold()` 函数对其进行操作，设置阈值为 128。依次使用表 5-5 中的设置类型并显示结果。你应该熟练的掌握阈值化处理，因为它非常有用。
- a. 用函数 `cvAdaptiveThreshold()` 设 `param1=5`，重复此练习。
 - b. 先设 `param1=0` 重复练习 a，然后设 `param1=-5`，重复练习 a。

第6章

图像变换

概述

第5章介绍了许多类型的图像处理方法。迄今为止，所介绍的大多数操作都用于增强、修改或者“处理”图像，使之成为类似但全新的图像。

本章将关注图像变换(image transform，即将一幅图像转变成图像数据)的另一种表现形式。变换最常见的例子就也许是傅里叶变换(Fourier transform)，即将图像转换成源图像数据的另一种表示。这类操作的结果仍然保存为OpenCV图像结构的形式，但是新图像的每个单独像素表示原始输入图像的频谱分量而不是我们通常所考虑的空间分量。

计算机视觉中经常会用到许多有用的变换。OpenCV提供了一套完整的实现工具和方法，其中一些普通的工具和方法就像积木一样可帮助我们实现各种图像转换。

卷积

卷积(convolution)是本章所讨论的很多变换的基础。从抽象的层次说，这个术语意味着我们要对图像的每一个部分进行操作。从这个意义上讲，我们在第5章所看到的许多操作可以被理解成普通卷积的特殊情况。一个特殊卷积所实现的功能是由其卷积核的形式决定的。这个核本质上是一个大小固定、由数值参数构成的数组，数组的参考点(anchor point)通常位于数组的中心。数组的大小称为核支撑(support of the kernel)。单就技术而言，核支撑实际上仅仅由核数组的非0部分组成。[【144】](#)

图 6-1 描述了以数组中心为参考点的 3×3 卷积核。若要计算一个特定点的卷积值，首先将核的参考点定位到图像的第一个像素点，核的其余元素覆盖图像中其相对应的局部像素点。对于每一个核点，我们可以得到这个点的核的值以及图像中相对应图像点的值。将这些值相乘并求和，并将这个结果放在与输入图像参考点所相对应的位置。通过在整个图像上扫描卷积核，对图像的每个点重复此操作。

【144~145】

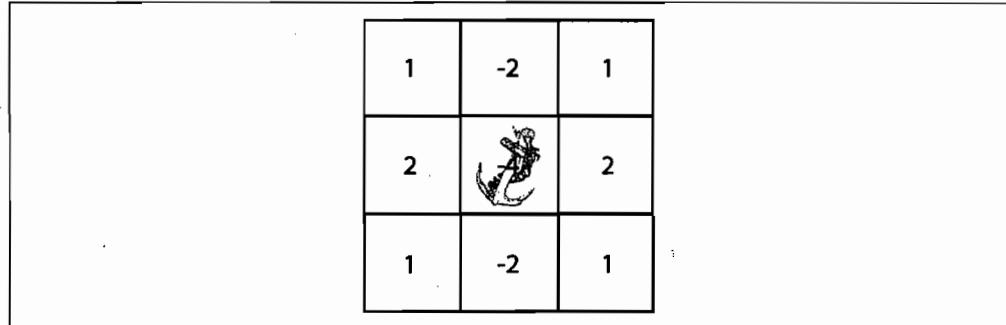


图 6-1: Sobel 导数的 3×3 核，注意，参考点在核的中心

当然，我们可以用方程来表示这个过程。如果我们定义图像为 $I(x,y)$ ，核为 $G(i,j)$ （其中 $0 < i < M_i - 1$ 和 $0 < j < M_j - 1$ ），参考点位于相应核的 (a_i, a_j) 坐标上，则卷积 $H(x,y)$ 定义如下：

$$H(x,y) = \sum_{i=0}^{M_i-1} \sum_{j=0}^{M_j-1} I(x+i-a_i, y+j-a_j) G(i,j)$$

注意观察运算次数，至少初看上去，它似乎等于图像的像素数乘以核的像素数^①。这其实是很大的计算量并且也不是仅仅用其中的一些 for 循环以及许多指针再分配就能做的事情。类似这种情况，最好让 OpenCV 来做这个工作以利用 OpenCV 已编程实现的最优方法。其函数为 cvFilter2D ()；

```
void cvFilter2D(
    const CvArr*      src,
    CvArr*            dst,
    const CvMat*      kernel,
```

① 这里我们说“初看上去”的意思是在频域中也可能进行卷积操作。在这种情况下，对于一个 $N \times N$ 的图像和一个 $M \times M$ 的核 ($N > M$)，计算复杂度将会正比于 $N^2 \log(N)$ 而增长，而不是在空域内预计的 $N^2 M^2$ 。这是因为频域的计算量与核的大小是相对独立的，对于大核更加高效。OpenCV 会根据核的大小自动决定是否做频域内的卷积。

```
CvPoint anchor = cvPoint(-1,-1)
};
```

【145~146】

这里我们创建一个适当大小的矩阵，将系数连同源图像和目标图像一起传递给 `cvFilter2D()`。我们还可以有选择地输入一个 `CvPoint` 指出核的中心位置，默认值(`cvPoint(-1,-1)`)会设参考点为核的中心。如果定义了参考点，核的大小可以是偶数，否则只能是奇数。

源图像 `src` 和目标图像 `dst` 大小应该相同，有些人可能认为考虑到卷积核的额外的长和宽，源图像 `src` 应该大于目标图像 `dst`。但是在 OpenCV 里源图像 `src` 和目标图像 `dst` 的大小是可以一样的，因为在默认情况下，在卷积之前，OpenCV 通过复制源图像 `src` 的边界创建了虚拟像素，这样以便于目标图像 `dst` 边界的像素可以被填充。复制是通过 `input(-dx, y) = input(0, y)`, `input(w + dx, y) = input(w - 1, y)` 等实现的，此默认行为还有一些替代方法，我们将在下一节讨论。

提示一下，这里我们所讨论的卷积核的系数应该是浮点类型的，这就意味着我们必须用 `CV_32F` 来初始化矩阵。

卷积边界

对于卷积，一个很自然的问题是如何处理卷积边界。例如，在使用刚才所讨论的卷积核时，当卷积点在图像边界时会发生什么？许多使用 `cvFilter2D()` 的 OpenCV 内置函数必须用各种方式来解决这个问题。同样在做卷积时，有必要知道如何有效解决这个问题。

这个解决方法就是使用 `cvCopyMakeBorder()` 函数，它可以将特定的图像轻微变大，然后以各种方式自动填充图像边界。

```
void cvCopyMakeBorder(
    const CvArr* src,
    CvArr* dst,
    CvPoint offset,
    int bordertype,
    CvScalar value = cvScalarAll(0)
);
```

`Offset` 变量告诉 `cvCopyMakeBorder()` 将源图像的副本放到目标图像中的位置。典型情况是，如果核为 $N \times N$ (N 为奇数)时，那么边界在每一侧的宽度都应是 $(N - 1)/2$ ，即这幅图像比源图像宽或高 $N - 1$ 。在这种情况下，可以把 `Offset` 设置为

`cvPoint((N-1)/2,(N-1)/2)`, 使得边界在每一侧都是偶数^①。

【146】

`Bordertype` 既可以是 `IPL_BORDER_CONSTANT`, 也可以是 `IPL_BORDER_REPLICATE`(见图 6-2)。在第一种情况下, `value` 变量被认为是所有在边界的像素应该设置的值。在第二种情况下, 原始图像边缘的行和列被复制到大图像的边缘。注意, 测试的模板图像边缘是比较精细的(注意图 6-2 右上角的图像)。在测试的模板图像中, 除了在圆图案边缘附近的像素变白外, 有一个像素宽的黑色边界。这里定义了另外两种边界类型, `IPL_BORDER_REFLECT` 和 `IPL_BORDER_WRAP`, 目前还没有被 OpenCV 所实现, 但以后可能会在 OpenCV 中实现。



图 6-2: 扩大的图像边界, 左列显示的是 `IPL_BORDER_CONSTANT`, 边界是用 0 值填充的, 右列是 `IPL_BORDER_REPLICATE`, 在水平和垂直两个方向复制边界像素

① 当然, 参考点在中心、 $N \times N$ 大小且 N 是奇数时的情形为最简单。在一般情况下, 如果核是 $N \times M$ 并且参考点在 (a_x, a_y) , 那么目标图像将比源图像宽 $N-1$ 、高 $M-1$ 个像素。`Offset` 的值仅仅是 (a_x, a_y) 。

我们在前面已经提到，当调用 OpenCV 库函数中的卷积功能时，`cvCopyMakeBorder()` 函数就会被调用。在大多数情况下，边界类型为 `IPL_BORDER_REPLICATE`，但有时并不希望用它。所以在另一种场合，可能用到 `cvCopyMakeBorder()`。你可以创建一幅具有比想要得到的边界稍微大一些的图像，无论调用任何常规操作，接下来就可以剪切到对源图像所感兴趣的部分。这样一来，OpenCV 的自动加边就不会影响所关心的像素。

【147】

梯度和 Sobel 导数

一个最重要并且最基本的卷积是导数的计算(或者是其近似值)，有许多方法可以做到，但是只有少数方法适合于给定情况。

通常来说，用来表达微分的最常用的操作是 Sobel 微分算子(见图 6-3 和图 6-4)。Sobel 算子包含任意阶的微分以及融合偏导(例如 $\partial^2/\partial x\partial y$)。

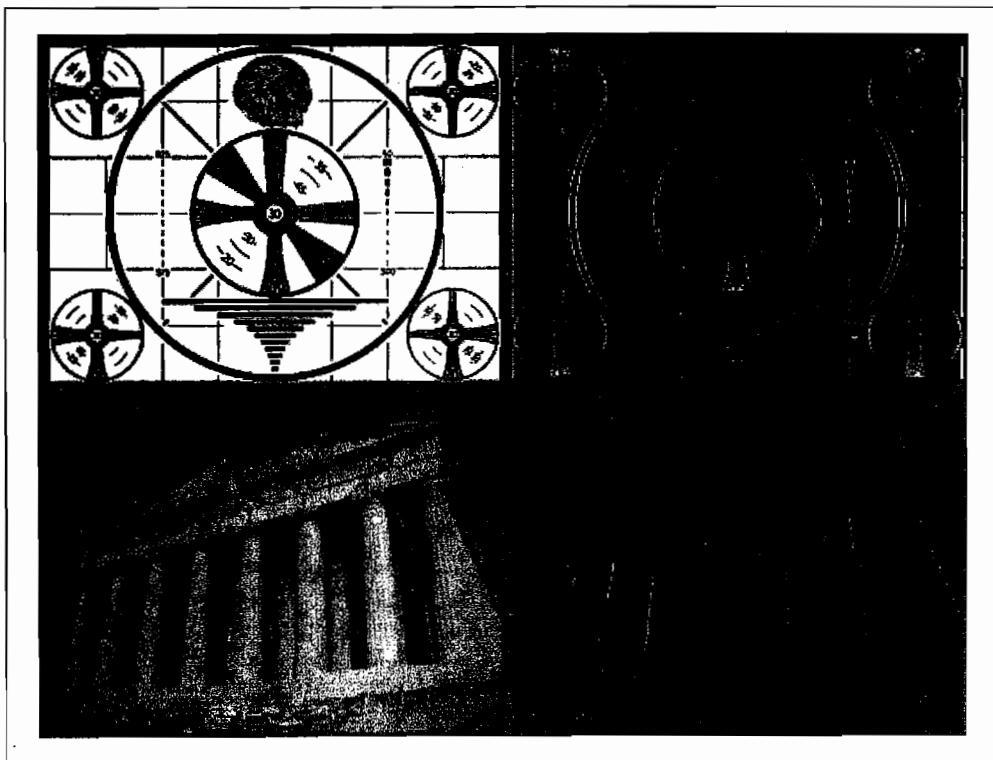


图 6-3：逼近 x 方向上一阶微分的 Sobel 算子的效果

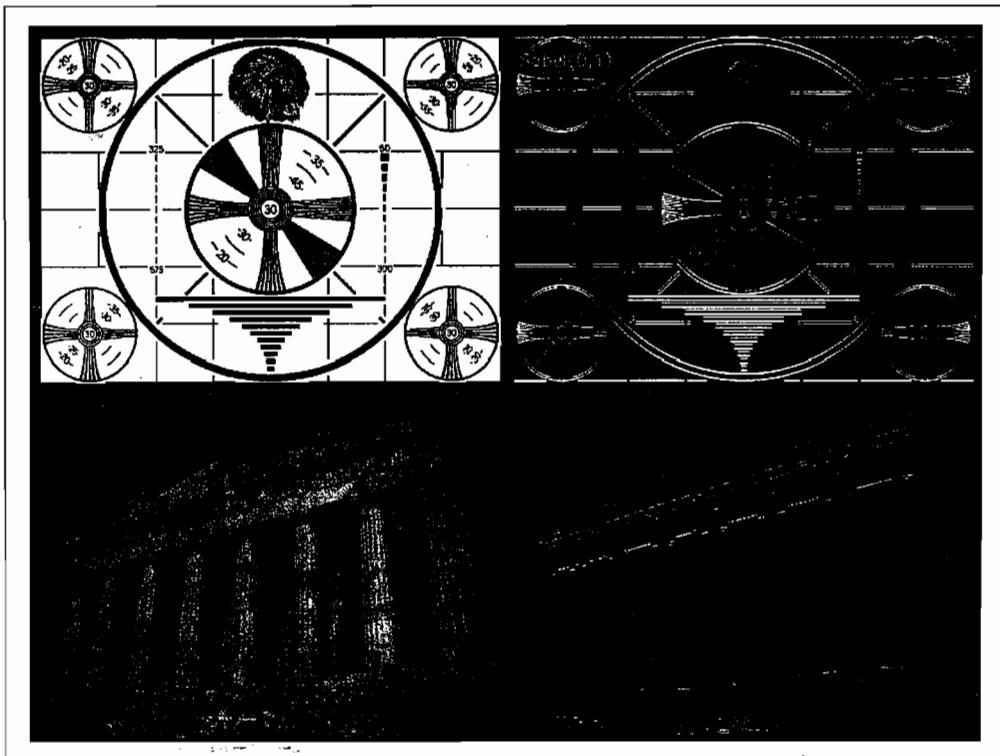


图 6-4：逼近 y 方向上一阶微分的 Sobel 算子的效果

```
cvSobel(
    const CvArr* src,
    CvArr* dst,
    int xorder,
    int yorder,
    int aperture_size = 3
);
```

【148】

这里，`src` 和 `dst` 分别是输入图像和输出图像，`xorder` 和 `yorder` 是求导的阶数。通常只可能用到 0, 1, 最多 2。值为 0 表明在这个方向上没有求导^①，`aperture_size` 参数是方形滤波器的宽(或高)并且应该是奇数。目前，`aperture_size` 支持 1, 3, 5, 7。如果源图像 `src` 是 8 位的，为避免溢出，目标图像的深度必须是 `IPL_DEPTH_16S`。

Sobel 导数有一个非常好的性质，即可以定义任意大小的核，并且这些核可以用快

① 无论是 `xorder` 还是 `yorder`，都必须非 0。

速且迭代方式构造。大核对导数有更好的逼近，因为小核对噪声更敏感。

为了更好地理解以上内容，我们必须认识到一点，即 Sobel 导数并不是真正的导数，因为 Sobel 算子定义于一个离散空间之上。Sobel 算子真正表示的是多项式拟合，也就是说， x 方向上的二阶 Sobel 导数并不是真正的二阶导数。它是对抛物线函数的局部拟合。这说明人们为什么想用较大的核，因为较大核是在更多像素上计算这种拟合。

【149】

Scharr 滤波器

事实上，在离散网格的场合下有很多方法可以近似地计算出导数。对于小一点的核而言，这种使用于 Sobel 算子近似计算导数的缺点是精度比较低。对于大核，由于在估计时使用了更多的点，所以这个问题并不严重。这种不精确性并不会直接在 cvSobel() 中使用的 X 和 Y 滤波器中表现出来，因为它们完全沿 x 轴和 y 轴排列。当试图估计图像的方向导数(*directional derivative*，即，使用 y/x 滤波器响应的反正切得到的图像梯度的方向)时，难度就出现了。

为了承上启下，这里有一个这类图像度量的具体例子。这个例子为从一个物体收集形状信息的过程，而这个过程是通过求取目标周围梯度角度的直方图来实现的。这样的直方图是许多通用或常见分类器训练和工作的基础。在这种情况下，对梯度角不准确的度量将会降低分类器识别的性能。

对于一个 3×3 的 Sobel 滤波器，当梯度角度越接近水平或者垂直方向时，这样的不准确就更加明显。OPENCV 通过在 cvSobel() 函数中一些特殊 `aperture_size` 值 `CV_SCHARR` 的隐性使用，以解决小(但是快)的 3×3 Sobel 导数滤波器不准确的问题。Scharr 滤波器同 sobel 滤波器一样快，但是准确率更高，所以当你利用 3×3 滤波器实现图像度量的时候应该使用 Scharr 滤波器。Scharr 滤波器的滤波系数如图 6-5 所示[Scharr00]。

<table border="1"><tr><td>-3</td><td>0</td><td>3</td></tr><tr><td>-10</td><td>0</td><td>10</td></tr><tr><td>-3</td><td>0</td><td>3</td></tr></table>	-3	0	3	-10	0	10	-3	0	3	<table border="1"><tr><td>-3</td><td>-10</td><td>-3</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>3</td><td>10</td><td>3</td></tr></table>	-3	-10	-3	0	0	0	3	10	3
-3	0	3																	
-10	0	10																	
-3	0	3																	
-3	-10	-3																	
0	0	0																	
3	10	3																	

图 6-5：使用标记 `CV_SCHARR` 的 3×3 Scharr 滤波器

拉普拉斯变换

OpenCV 的拉普拉斯函数(第一次被 Marr [Marr82]应用于视觉领域)实现了拉普拉斯算子的离散模拟。^①

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

因为拉普拉斯算子可以用二次导数的形式定义，可假设其离散实现类似于二阶 Sobel 导数。事实的确如此，OpenCV 在计算拉普拉斯算子时直接使用 Sobel 算子。

```
void cvLaplace(
    const CvArr* src,
    CvArr*       dst,
    int          apertureSize = 3
);
```

cvLaplace() 函数通常把源图像和目标图像以及中孔大小作为变量。源图像既可以是 8 位(无符号)图像，也可以是 32 位(浮点)图像。而目标图像必须是 16 位(有符号)或者 32 位(浮点)图像。这里的中孔与 Sobel 导数中出现的中孔完全一样，事实上主要给出区域大小，在二次求导的计算中，采样这个区域的像素。

拉普拉斯算子可用于各种情况。一个通常的应用是检测“团块”。联想到拉普拉斯算子的形式是沿着 X 轴和 Y 轴的二次导数的和，这就意味着周围是更高值的单点或者小块(比中孔小)会将使这个函数值最大化。反过来说，周围是更低值的点将会使函数的负值最大化。

基于这种思想，拉普拉斯算子也可以用作边缘检测。为达此目的，只需要考虑当函数快速变化时其一阶导数变大即可。同样重要的是，当我们逼近类似边缘的不连续地方时导数会快速增长，而穿过这些不连续地方时导数又会快速减小。所以导数会在此范围内有局部极值。这样我们可以期待局部最大值位于二阶导数为 0 的地方。明白了吗？原始图像的边缘位于拉普拉斯的值等于 0 的地方。不幸的是，在拉普拉斯算子中，所有实质性的和没有意义的边缘的检测都是 0。但这并不是什么问题，因为我们可以过滤掉这些点，它们的一阶(Sobel)导数值也很大。图 6-6 显示对图像应用拉普拉斯和一次、二次导数以及它们的 0 交叉的例子。

① 注意，拉普拉斯算子与第 5 章的拉普拉斯金字塔完全不同。

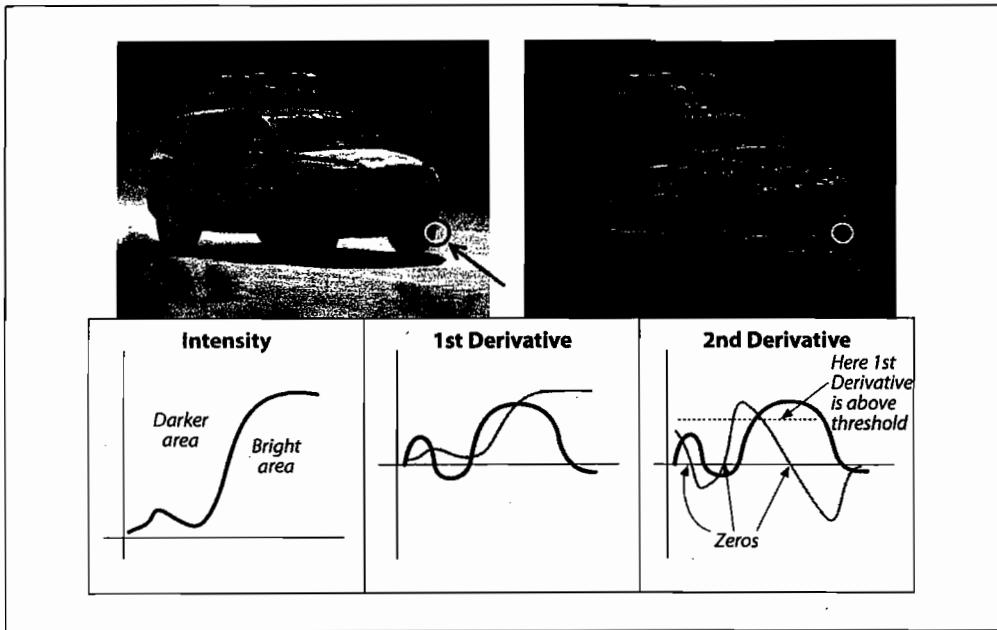


图 6-6：赛车图像的拉普拉斯变换(右上角)。放大轮胎(白圈)并且只考虑 X 方向，我们(定性)展示了具有代表性的亮度、一次和二次导数(下面 3 个子图)，二次导数的 0 值对应着边，相对较大的一次导数的 0 对应的是较强的边缘

Canny 算子

以上所描述的边缘检测的方法在 1986 年由 J. Canny 得到完善，也就是通常所称的 Canny 边缘检测法[Canny86]。Canny 算法同 6.4 节提到的简单的基于拉普拉斯算法的不同点之一是在 Canny 算法中，首先在 x 和 y 方向求一阶导数，然后组合为 4 个方向的导数。这些方向导数达到局部最大值的点就是组成边缘的候选点。【151】

然而，Canny 算法最重要的一个新特点是其试图将独立边的候选像素拼装成轮廓。^①轮廓的形成是对这些像素运用滞后性阈值。这意味着有两个阈值，上限和下限。如果一个像素的梯度大于上限阈值，则被认为是边缘像素，如果低于下限阈值，则被抛弃，如果介于二者之间，只有当其与高于上限阈值的像素连接时才会被接受。

① 我们将在以后对轮廓进行更多地阐述。但要记住，cvCanny() 实际上通常并不返回 CvContour 的对象。如果我们想要在 cvFindContours() 中使用它们，就必须在 cvCanny() 的输出中建立它们。所有轮廓知识将在第 8 章全面讨论。

Canny 推荐的上下限阈值比为 2 : 1 到 3 : 1 之间。图 6-7 和图 6-8 分别显示了利用 cvCanny() 以及上下限阈值比为 5 : 1 和 3 : 2 的测试图案和图像的结果。

```
void cvCanny(  
    const CvArr* img,  
    CvArr* edges,  
    double lowThresh,  
    double highThresh,  
    int apertureSize = 3  
) ;
```

【152】

cvCanny() 函数需要输入一幅灰度图，输出图也一定是灰度的(实际上是布尔图像)。接下来两个参数是下限阈值和上限阈值，最后一个参数是另一个中孔。通常，这个被 Sobel 算子用到的中孔是 cvCanny() 在内部使用的。

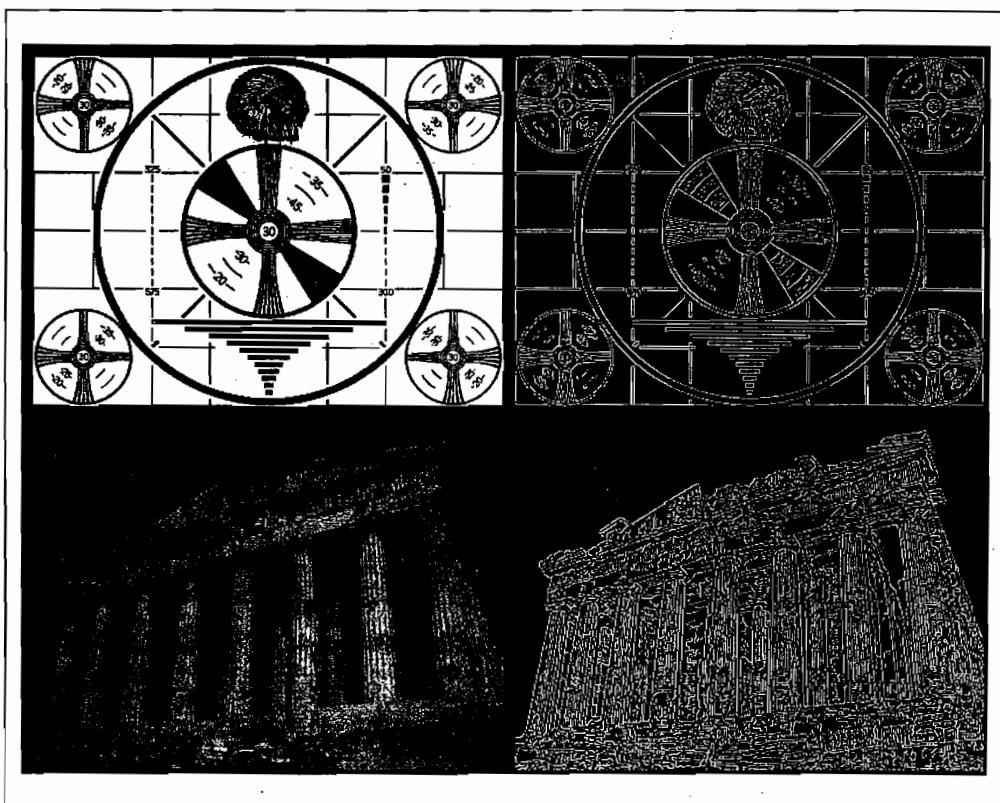


图 6-7：上下限分别为 50 和 10 时用 Canny 算子对两幅不同图像进行边缘检测的结果

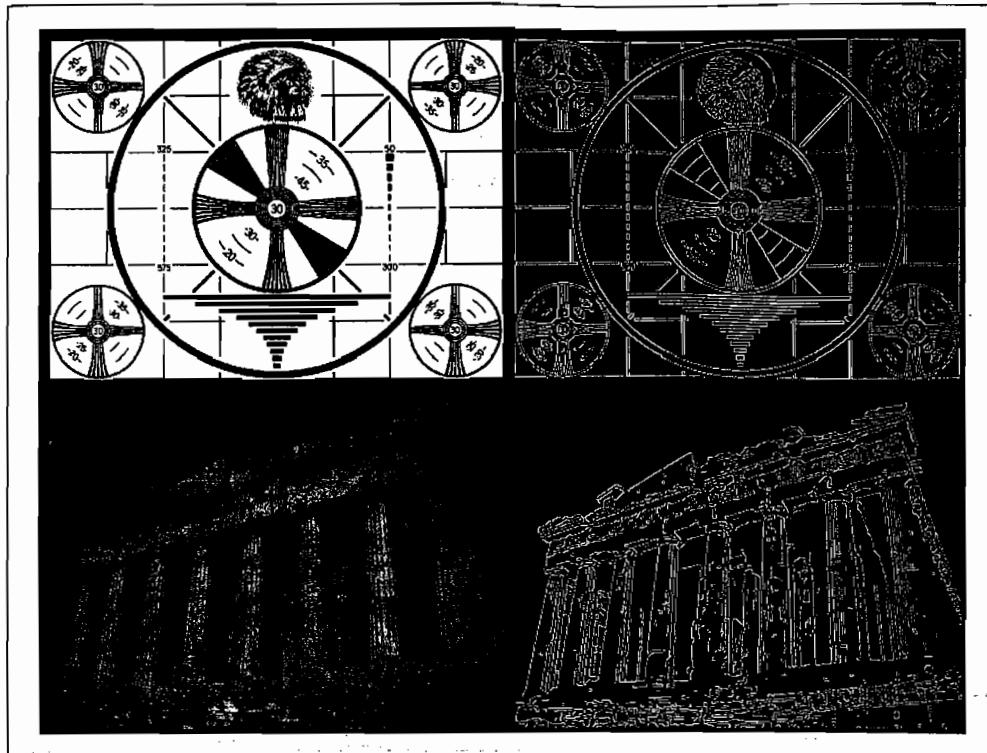


图 6-8：上下限分别为 150 和 100 时用 Canny 算子对两幅不同图像进行边缘检测的结果

霍夫变换

霍夫变换^①是一种在图像中寻找直线、圆及其他简单形状的方法。原始的霍夫变化是一种直线变换，即在二值图像中寻找直线的一种相对快速方法。变换可以推广到其他普通的情况，而不仅仅是简单的直线。

霍夫线变换

霍夫直线变换的基本理论是二值图像中的任何点都可能是一些候选直线集合的一部

① 霍夫首先在物理实验中发展了该变换[Hough59]; Duda 和 Hart 将其应用在视觉领域 [Duda72]。

分。如果要确定每条线进行参数化，例如一个斜率 a 和截距 b ，原始图像中的一点会变换为 (a, b) 平面上的轨迹，轨迹上的点对应着所有过原始图像上点的直线（见图 6-9）。如果我们将输入图像中所有非 0 像素转化成输出图像中的这些点集并且将其贡献相加，然后输入图像（例如 (x, y) 平面）出现的直线将会在输出图像（例如 (a, b) 平面）以局部最大值出现。因为我们将每个点的贡献相加，因此 (a, b) 平面通常被称为累加平面（accumulator plane）。

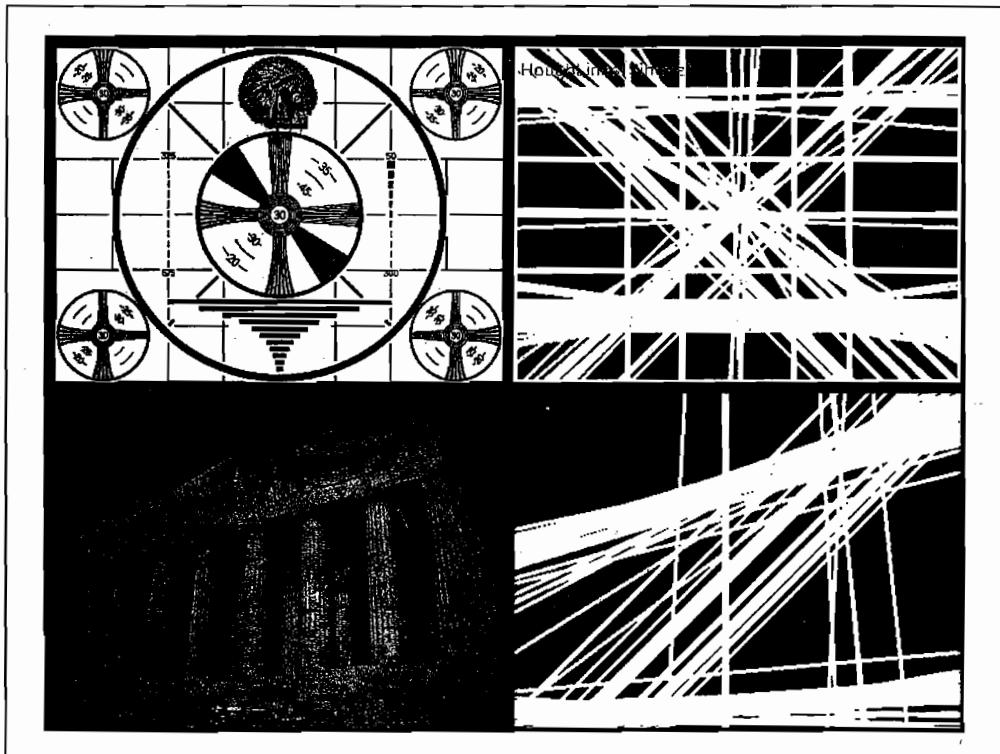


图 6-9：霍夫变换找到了图像中的许多线；其中一些是想要的，而另一些不是

你可能认为用斜率-截距的形式来代表所有通过的点并不是一种最好的方式（因为作为斜率函数，直线的密度有相当的差异，以及相关的事实是可能的斜率间隔的范围是从 $-\infty$ 到 $+\infty$ ）。正是由于这个原因，在实际数值计算中使用的变换图像的参数化略微有些不同。首选的参数化方式是每一行代表极坐标 (ρ, θ) 中的一个点，并且隐含的直线是通过象征点，垂直于远点到此点的半径。如图 6-10 所示，此直线的方程如下：

$$\rho = x \cos \theta + y \sin \theta$$

【153 ~ 154】

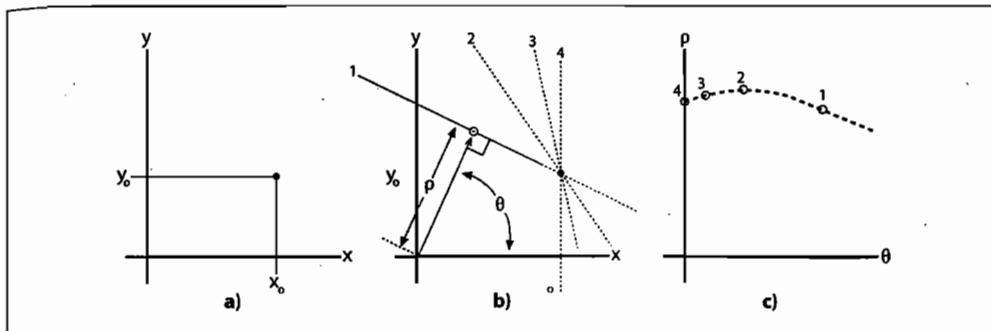


图 6-10：a) 显示图像平面的一个点 (x_0, y_0) ，b) 显示 a) 图中参数 ρ 和 θ 不同时的许多线，这些线隐含着在 (ρ, θ) 平面内的点，放在一起就形成了一条特征曲线 (c 图)

【155】

OpenCV 的霍夫变换算法并没有将这个算法显式地展示给用户。而是简单地返回 (ρ, θ) 平面的局部最大值。然而，需要了解这个过程以便更好地理解 OpenCV 霍夫变换函数中的参数。

OpenCV 支持两种不同形式的霍夫变换：标准霍夫变换(SHT) [Duda72]和累计概率霍夫变换(PPHT)^①。刚才所说的是 SHT 算法。PPHT 是这种算法的一个变种，计算单独线段的方向以及范围(如图 6-11 所示)。之所以称 PPHT 为“概率”的，是因为并不将累加器平面内的所有可能点累加，而只累加其中的一部分。该想法是如果峰值将要足够高，只用一小部分时间去寻找它就足够了。这个猜想的结果可以实质性地减少计算时间。尽管有一些变量的含义是取决于用哪个算法，但可以使用 OpenCV 中的同一个函数来访问这两个算法。

```
CvSeq* cvHoughLines2(
    CvArr* image,
    void* line_storage,
    int method,
    double rho,
    double theta,
    int threshold,
    double param1      = 0,
    double param2      = 0
);
```

① Kiryati, Eldar 和 Bruckshtein 在 1991 年 [Kiryati91] 提出了“概率霍夫变换”(PHT)，PPHT 是在 1999 年由 Matas, Galambosy 和 Kittler 提出的。

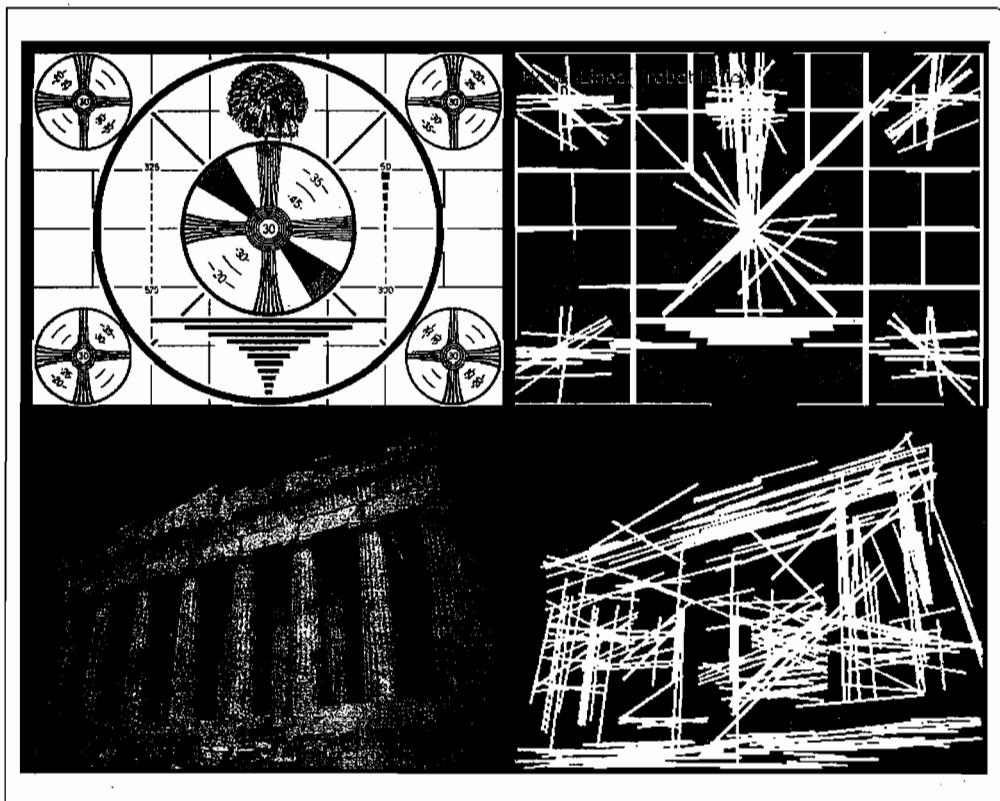


图 6-11：首先运行 Canny 边缘检测(参数 1=50，参数 2=150)，结果以灰度图显示。然后运行累计概率的霍夫变换(参数 1=50，参数 2=10)，结果被白色覆盖，可以看到由霍夫变换产生的粗线

第一个变量是输入图像，必须是 8 位的，但输入信息可以被看成是二值的(即所有非 0 像素被认为是相等的)。第二个参数是指向保存结果位置的指针，既可以是内存块(见第 8 章的 CvMemoryStorage)，也可以是 $N*1$ 的矩阵数列(行数 N 将有助于限制直线的最大数量)。下一个参数 method 可以是 CV_HOUGH_STANDARD，CV_HOUGH_PROBABILISTIC 或者 CV_HOUGH_MULTI_SCALE，分别对应 SHT，PPHT 或 SHT 的多尺度变种。

下面两个变量，rho 和 theta 是用来设置直线所需要的分辨率(例如，累加平面所需要的分辨率)。Rho 的单位是像素而 theta 的单位是弧度。因此，累加平面可以看成是由 rho 像素和 theta 弧度组成的二维直方图。threshold 是认定为一条直线时在累加平面中必须达到的值。最后一个参数在实际应用中有点棘手，它没有被归一化，所以应该将其根据 SHT 中的图像进行尺度化。请记住，这个变量实际上

表示支持所返回的直线的(在边缘图像的)点的数量。

【156】

在 SHT 中没有用到 param1 和 param2 参数。对于 PPHT，param1 设置为将要返回的线段的最小长度，param2 设置为一条直线上分离线段不能连成一条直线的分隔像素点数。对于多尺度的 HT(Hough Transform)，这两个参数是用来指明应被计算的直线参数中较高的分辨率。多尺度的 HT 首先根据 rho 和 theta 参数准确计算直线的位置，然后分别通过 param1 和 param2 等比例继续细化结果(例如，rho 中最终的分辨率是 param1 分割 rho 产生的，theta 中最终的分辨率是 param2 分割 theta 产生的)。

函数的返回内容依赖于调用方式。如果 line_storage 是矩阵数组，最终的返回值为空。在这种情形下，使用 SHT 或者多尺度的 HT 时，矩阵应该是 CV_32FC2 类型，当使用 PPHT 时，矩阵应为 CV_32SC4 类型。在头两种情形下，每一行中 ρ - 和 θ - 的值应在数组中的两个通道里。在 PPHT 情形下，四个通道保留的是返回线段开始点和结束点的 x - 和 y - 的值。在所有的情形下，数组的行数将会被 cvHoughLines2() 更新以便正确反映返回直线的数量。

【157】

如果 line_storage 是指向内存块的一个指针^①，返回值将是一个指向 CvSeq 序列结构的指针。在这种情况下，可以用下面的类似命令从序列中得到每一条直线或者线段。

```
float* line = (float*) cvGetSeqElem( lines , i );
```

其中 lines 是从 cvHoughLines2() 中得到的返回值，i 是所关心的线的索引。在这种情形下，line 是指向这条直线数据的指针，对于 SHT 和 MSHT，line[0] 和 line[1] 是浮点类型的 ρ 和 θ ，对于 PPHT，是线段终点的 CvPoint 结构。

霍夫圆变换

霍夫圆变换[Kimme75](如图 6-12)与之前所描述的霍夫直线变换是大体上是类似的。说“大体上类似”的原因是——如果想要尝试完全类似——累加平面会被三维的累加容器所代替：在这三维中，一维是 x ，一维是 y ，另一维是圆的半径 r 。这就意味着需要大量的内存但速度却很慢。在 OpenCV 的应用中可以通过一个比较灵活的霍夫梯度法来解决圆变换的这一问题。

霍夫梯度法的原理如下。首先对图像应用边缘检测(这里用 cvCanny())。然后，对

① 我们还没有介绍内存或者序列的概念，这个主题主要由第 8 章完成。

边缘图像中每一个非 0 点，考虑其局部梯度(通过 cvSobel() 函数计算 x 和 y 方向的 Sobel 一阶导数得到梯度)。利用得到的梯度，由斜率指定的直线上的每一个点都在累加器中被累加，这里斜率是从一个指定的最小值到指定的最大值的距离。同时，标记边缘图像中每一个非 0 像素的位置。然后从(二维)累加器中这些点中选择候选的中心，这些中心都大于给定阈值并且大于其所有近邻。这些候选的中心按照累加值降序排列，以便于最支持像素的中心首先出现。接下来对每一个中心，考虑所有的非 0 像素(回想一下这个清单在早期已经建立)。这些像素按照其与中心的距离排序。从到最大半径的最小距离算起，选择非 0 像素最支持的一条半径。如果一个中心受到边缘图像非 0 像素最充分的支持，并且到前期被选择的中心有足够的距离，它将会被保留。

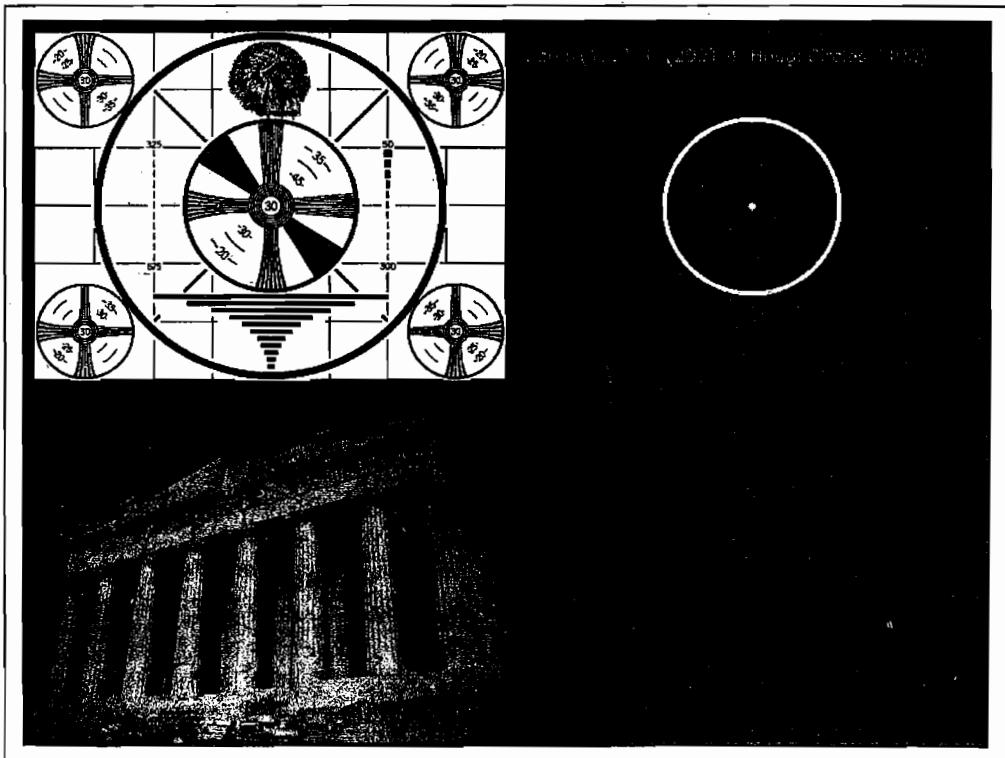


图 6-12：霍夫圆变换可以在测试图案中找到圆但在照片中没有(正确)找到

这个实现可以使算法执行起来更快，或许更重要的是，能够帮助解决三维累加器中其他稀疏分布问题，这个问题会产生许多噪声并使结果不稳定。另一方面，这个算法有许多需要注意的缺点。

【158】

首先，使用 Sobel 导数计算局部梯度——随之而来的假设是这个可以看作等同于一

条局部切线——并不是一个数值稳定做法。在大多数时间这个命题可能是真的，但你可能认为这样会在输出中产生一些噪声。

其次，在边缘图像中的整个非 0 像素集被认为是每个中心的候选。因此，如果把累加器的阈值设置偏低，算法将要消耗比较长的时间。第三，因为每一个中心只选择一个圆，如果有同心圆，就只能选择其中的一个。

最后，因为中心是被认为是按照与其关联的累加器的值升序排列的，并且如果新的中心过于接近以前接受的中心将不会被保留。这里有一个倾向就是当有许多同心圆或者是近似同心圆时，保留最大的一个圆。(只是个“偏见”，因为从 Sobel 导数产生了噪声；若是在无穷分辨率的平滑图像，这才是确定的)。

把所有以上的情况记录在大脑内，接下来看 OpenCV 程序能为我们做的：

```
CvSeq* cvHoughCircles(  
    CvArr* image,  
    void* circle_storage,  
    int method,  
    double dp,  
    double min_dist,  
    double param1 = 100,  
    double param2 = 300,  
    int min_radius = 0,  
    int max_radius = 0  
) ;
```

【159~160】

霍夫圆变换函数 cvHoughCircles() 与直线变换有相似的变量。输入的 image 也是 8 位的。cvHoughCircles() 与 cvHoughLines2() 一个明显不同是后者需要二值图像。cvHoughCircles() 函数将会在内部(自动)调用 cvSobel()^①，所以可以提供更加普通的灰度图。

circle_storage 既可以是数组，也可以是内存存储器(memory storage)，这取决于我们希望返回什么结果。如果使用数组，则应该是 CV_32FC3 类型的单列数组，三个通道分别存储圆的位置及其半径。如果使用内存存储器(memory storage)，圆将会变成 OpenCV 的一个序列 CvSeq，由 cvHoughCircles() 返回一个指向这个序列的指针(给定一个指向圆存储的数组指针值，cvHoughCircles() 的返回值将为空)。在这个方法中，参数必须设置为 CV_HOUGH_GRADIENT。

① 内部调用的是函数 cvSobel()，而不是 cvCanny()。因为 cvHoughCircles() 需要估计每一个像素梯度的方向，但二值边缘图像的处理是比较难的。

参数 dp 是指累加器图像的分辨率。这个参数允许创建一个比输入图像分辨率底的累加器。(这样做是因为没有理由认为图像中存在的圆会自然降低到与图像宽高相同数量的范畴)。如果 dp 设置为 1 时，则分辨率是相同的；如果设置为更大的值(比如 2)，累加器的分辨率受此影响会变小(此情况下为一半)。dp 的值不能比 1 小。

参数 min_dist 是让算法能明显区分的两个不同圆之间的最小距离。

当方法(现在必须的)设置成 CV_HOUGH_GRADIENT 时，后面的两个参数 param 1 和 param 2 分别是边缘阈值(Canny)和累加器阈值。Canny 边缘检测实际上本身只用到两个不同的阈值。当内部调用 cvCanny() 时，第一个(高的)阈值被设成输入到 cvHoughCircles() 的 param1，第二个(低的)阈值被严格设为此值的一半。param2 是用作累加器的阈值恰好与 cvHoughLines() 的阈值参数类似。

最后两个参数是所能发现的圆半径的最小值和最大值。这就意味着累加器中圆的半径具有代表性。例 6-1 展示了利用 cvHoughCircles() 的示例程序。

例 6-1：使用 cvHoughCircles 返回在灰度图中找到的圆序列

```
#include <cv.h>
#include <highgui.h>
#include <math.h>

int main(int argc, char** argv) {
    IplImage* image = cvLoadImage(
        argv[1],
        CV_LOAD_IMAGE_GRAYSCALE
    );

    CvMemStorage* storage = cvCreateMemStorage(0);
    cvSmooth(image, image, CV_GAUSSIAN, 5, 5);

    CvSeq* results = cvHoughCircles(
        image,
        storage,
        CV_HOUGH_GRADIENT,
        2,
        image->width/10
    );

    for( int i = 0; i < results->total; i++ ) {
        float* p = (float*) cvGetSeqElem( results, i );
    }
}
```

```

CvPoint pt = cvPoint( cvRound( p[0] ), cvRound( p[1] ) );
cvCircle(
    image,
    pt,
    cvRound( p[2] ),
    CV_RGB(0xff,0xff,0xff)
);
}
cvNamedWindow( "cvHoughCircles", 1 );
cvShowImage( "cvHoughCircles", image );
cvWaitKey(0);
}

```

现在值得考虑的一个事实是，不管我们用什么技巧，都不能绕过圆必须被描述成三个自由度的需求(x , y 和 r)，与此形成对比的是直线只需要两个自由度(ρ 和 θ)。必然出现的结果是寻找圆的算法比我们以前看到的寻找直线的算法需要更多的内存和计算量。考虑到这一点，为了让这些消耗能在掌控之内，在环境允许范围内牢固的限制半径参数是一个好的主意^①。通过考虑物体是梯度边缘的集合，1981 年 Ballard[Ballard81]将霍夫圆变换拓展到任意形状。

【161】

重映射

在幕后，许多变换都有一个共同点。具体说来，它们会把一幅图像中一个位置的像素重映射到另一个位置。在这种情况下，就始终需要一些平滑的映射(我们希望实现的)，但并不总是能做到像素一一对应。

我们有时想以编程方式完成这类插值，即想用一些已知的算法来确定映射。然而在其他一些情况下，我们想自定义映射。在深入计算映射的方法之前，我们首先看一下其他方法所依赖的可解决映射问题的函数。在 OpenCV 程序中，这样的函数是 `cvRemap()`：

```

void cvRemap(
    const CvArr* src,
    CvArr*       dst,

```

① 尽管 `cvHoughCircles()` 能很好地捕获到圆心，但有时也会找不到圆的半径。因此，在只需找到圆的中心(或者有其他一些技术可以准确找到圆的半径)的实际应用中，`cvHoughCircles()` 返回的圆半径可以忽略。

```

const CvArr* mapx,
const CvArr* mapy,
int flags = CV_INTER_LINEAR | CV_WARP_FILL_OUTLIERS,
CvScalar fillval = cvScalarAll(0)
);

```

`cvRemap()`的前两个参数分别是源图像和目标图像。显然它们的大小和通道数必须相同，但可以是任意的数据类型。重要的是要知道它们不一定是同一幅图像^①。后两个参数 `mapx` 和 `mapy` 指明任意具体像素重新分配的位置。它们应该与源图像和目标图像的大小相同，但为单通道并且通常是浮点类型(`IPL_DEPTH_32F`)。非整型映射确定后，`cvRemap()`将会自动计算插值。`cvRemap()`的一个通常的用途是校正(纠正失真)标定和立体的图像。我们将在第 11 章和第 12 章看到这些函数，这些函数计算摄像机失真并且排列为 `mapx` 和 `mapy` 参数。最后一个参数包括一个标志位 `flags`，明确告诉 `cvRemap()` 插值如何进行。表 6-1 中给出的所有值都可用。

表 6-1: `cvWarpAffine()` 附加标志位的值

标志位的值	意义
<code>CV_INTER_NN</code>	最近邻
<code>CV_INTER_LINEAR</code>	双线性(默认)
<code>CV_INTER_AREA</code>	像素区域重新采样
<code>CV_INTER_CUBIC</code>	双三次插值

【162】

在这里，插值是一个很重要的问题。源图像的像素在整数网格内，例如，我们可以说像素位于(20, 17)的位置。当这些整数位置映射到新的图像中时，可能会有一些差异——源图像中像素的位置是整型的而目标图像是浮点型的，就必须将其四舍五入到最相近的整型，因为可能映射后的位置完全就是没有像素的(联想一下通过拉伸将图像扩大两倍，其他的每一个目标像素将会是空白)。这些问题一般被称为正投影问题。为了解决这些四舍五入和目标差异的问题，实际上我们可以反过来解决：通过目标图像中的每一个像素去问：“哪个像素需要来填补这个目标像素？”这些源像素的位置几乎都是小数(非整数)，所以必须对这些源像素进行插值以得到目标位置的正确值。默认方法是双线性插值，但也可以选择其他方法(如表 6-1

① 稍加思考就会弄清楚为什么最有效的重映射策略会与写入源图像不匹配。毕竟，如果把一个像素从 A 位置移动到 B 位置，然后当你到达位置 B 又想到位置 C 时，你会发现你已经把 A 的初值写入了 B！

所示)。

也可以增加(用或操作)标志位 `CV_WARP_FILL_OUTLIERS`, 其效果是可以用最后一个变量 `fillval` 设置的值填充目标图像的像素, 而这些像素在原始输入图像中没有任意像素与之对应。通过这种方式, 如果将所有图映射到一个位于中心的圆里, 圆的外面会自动填充成黑色(或者你想像的任意颜色)。

拉伸、收缩、扭曲和旋转

本小节介绍图像的几何操作^①。这些操作包括各种方式的拉伸, 包括一致性缩放和非一致性缩放(后者被称为扭曲)。有许多理由来执行这些操作: 例如, 扭曲和旋转一幅图像使得它可以叠加在场景中的墙上, 或者人为地扩大一组训练图像用于物体识别^②。这些拉伸、扭曲、旋转图像的函数叫做几何转换函数(早期的阐述见 [Semple79])。对于平面区域, 有两种方式的几何转换: 一种是基于 2×3 矩阵进行的变换, 也叫仿射变换; 另一种是基于 3×3 矩阵的变换, 又称透视变换或者单应性映射。可以把后一种变换当作一个三维平面被一个特定观察者感知的计算方法, 而该观察者也许不是垂直观测该平面。

【163】

一个任意的仿射变换可以表达为乘以一个矩阵再加上一个向量的形式。在 OpenCV 里, 代表这种变换的标准形式是 2×3 矩阵。定义如下:

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad B = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \quad T = \begin{bmatrix} A & B \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \end{bmatrix} \quad X' = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

容易看出, 仿射变换 $A \cdot X + B$ 在效果上就等于将向量 X 拓展成 X' , 并且只是将 X' 左乘 T 。

仿射变换可以形象地表示成以下形式。一个平面内的任意平行四边形 $ABCD$ 可以被仿射变换映射为另一个平行四边形 $A'B'C'D'$ 。如果这些平行四边形的面积不等于 0, 这个隐含的仿射变换就被这两个平行四边形(其中的三个顶点)惟一定义。如果

-
- ① 我们将在这里详细讲述这些变换; 在讨论如何在三维视觉技术中应用这些变换时(第 11 章), 将再次阐述它们。
 - ② 这个行为似乎有点迷惑。一句话, 如果使用对局部仿射扭曲具有不变性的识别方法, 岂不更好? 不过尽管如此, 这个方法有着很长的历史而且在实践中仍然非常有用。

愿意，可以将仿射变换想像成把一幅图像画到一个胶版上^①，在胶版的角上推或拉以使其变形而得到不同类型的平行四边形。

当有多幅图像时，我们知道在不同角度观察相同的物体会略有不同，所以可能希望计算在不同角度下的实际变换时会与不同的视角有一些关系。在这种情况下，仿射变换通常用单应性(homography)建模^②，这是因为其参数少，有利于问题的解决。它的缺点是真正投影畸变只能通过单应性建模来解决，所以仿射变换所产生的表示并不能适应视角间所有的关系。另一方面，由视角微小的变化导致的失真就是仿射，所以在有些环境下，仿射变换也许够用了。

仿射变换可以将矩形转换成平行四边形。它可以将矩形的边压扁但必须保持边是平行的，也可以将矩形旋转或者按比例变化。透视变换提供了更大的灵活性，一个透视变换可以将矩形转变成梯形。当然，因为平行四边形也是梯形，所以仿射变换是透视变换的子集。图 6-13 展示了各种仿射变换和透视变换的例子。

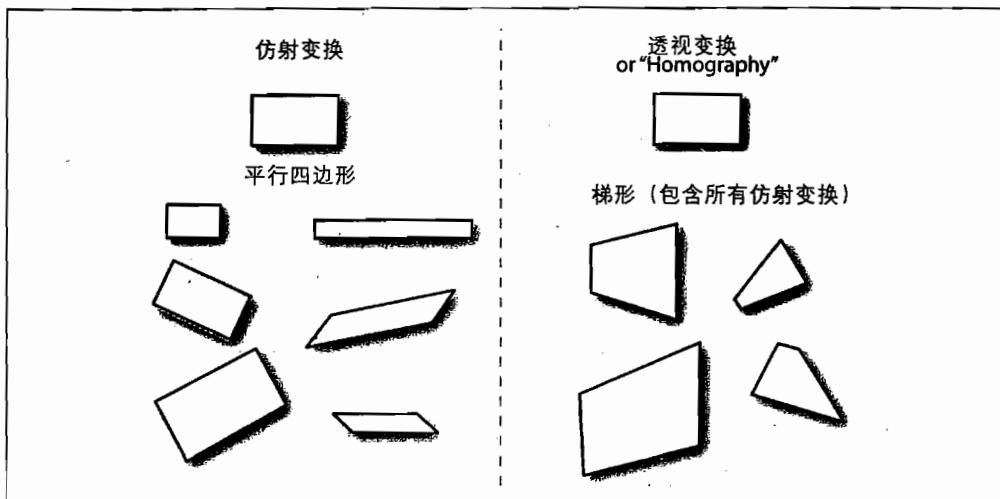


图 6-13：仿射变换和透视变换

-
- ① 甚至我们可以用拉的方式以颠倒这个平行四边形。
 - ② “单应性”是一个数学名词，指的是将一个表面上的点集映射到另一个表面上的点集。在这个意义上，这是一个比这里所用的更加一般的术语。在计算机视觉里，单应性通常是指在两幅图上点之间的映射，这些点与现实世界中的平面物体的位置是相对应的。可以看出这样的映射可以用一个 3×3 的正交矩阵表示(详情参见第 11 章)。

仿射变换

有两种情况会用到仿射变换。第一种是有一幅想要转换的图像(或者感兴趣区域),第二种是我们有一个点序列并想以此计算出变换。

【164】

稠密仿射变换

在第一种情况下,很明显输入/输出的格式是图像,并且隐含的要求是扭曲假设对于所使用的图像,其像素必须是其稠密的表现形式。这意味着图像扭曲必须进行一些插值运算以使输出的图像平滑并且看起来自然一些。OpenCV 中为稠密变换提供的转换函数是 `cvWarpAffine()`。

```
void cvWarpAffine(
    const CvArr* src,
    CvArr*       dst,
    const CvMat* map_matrix,
    int          flags     = CV_INTER_LINEAR |
    CV_WARP_FILL_OUTLIERS,
    CvScalar     fillval   = cvScalarAll(0)
);
```

这里 `src` 和 `dst` 表示数组或图像,它们可为单通道或三通道的任意类型(假定它们大小和类型是相同的)^①。`map_matrix` 是一个我们前面介绍过的对所需要的变换进行量化的 2×3 矩阵。倒数第二个参数 `flags` 控制插值的方法以及下面一个或两个附加选项(通常用布尔或操作来组合)。

- **CV_WARP_FILL_OUTLIERS** 通常,变换的 `src` 图像不能和 `dst` 图像完美匹配——从源图像映射的像素可能实际上并不存在。如果设置了标志位,这些失去的值就会由 `fillval` 补充(前面已经介绍)。
- **CV_WARP_INVERSE_MAP** 这个标志位用于方便地进行从 `dst` 到 `src` 的逆向变形,而不是从 `src` 到 `dst` 的变换。

【165】

① 因为将一幅图像旋转通常会使其外接盒变大,这将导致图像被剪切。若要避免这种情况,可以将图像缩小(就像在代码范例中的)或者将第一幅图像在变换之前复制到较大源图像里感兴趣区域(ROI)的中心。

cvWarpAffine 的性能

有必要知道 cvWarpAffine() 涉及开销问题。另一种方法是利用 cvGetQuadrangleSubPix()。这个函数的选项更少但优点更多。尤其是，它的开销比较小而且可以处理源图像是 8 位而目标图像是 32 位浮点图像的特殊情况。同时它能处理多通道图像。

```
void cvGetQuadrangleSubPix(
    const CvArr*      src,
    CvArr*            dst,
    const CvMat*      map_matrix
);
```

cvGetQuadrangleSubPix() 所做的就是计算从 src 图像的点(通过插值)映射到 dst 图像上的所有的点，这个映射是通过仿射变换即乘一个 2×3 矩阵实现的。(目标图像的位置乘以齐次坐标的转换是自动完成的)。

$$dst(x, y) = src(a_{00}x'' + a_{01}y'' + b_0, a_{10}x'' + a_{11}y'' + b_1)$$

cvGetQuadrangleSubPix() 的一个特点是函数提供了附加映射。具体来说，目标图像中点的结果是通过下式计算的：

$$dst(x, y) = src(a_{00}x'' + a_{01}y'' + b_0, a_{10}x'' + a_{11}y'' + b_1)$$

其中：

$$M_{\text{map}} \equiv \begin{bmatrix} a_{00} & a_{01} & b_0 \\ a_{10} & a_{11} & b_1 \end{bmatrix} \text{ and } \begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x - \frac{(\text{width(dst)}-1)}{2} \\ y - \frac{(\text{height(dst)}-1)}{2} \end{bmatrix}$$

观察到从 (x, y) 到 (x'', y'') 的映射产生的效果是一一即使映射 M 是恒等映射——目标图像中心点会从原点的源图像获得。如果 cvGetQuadrangleSubPix() 需要图像外面的点，它会利用复制来重构这些值。

仿射映射矩阵的计算

OpenCV 提供了两个函数以帮助生成映射矩阵 map_matrix。如果已有两幅图像要通过仿射变换发生关联或希望以那种方式来逼近则可以使用第一个函数如下所示：

```
CvMat* cvGetAffineTransform(
    const CvPoint2D32f* pts_src,
```

```
    const CvPoint2D32f* pts_dst,
    CvMat*           map_matrix
}
```

【166】

这里 `src` 和 `dst` 是包含三个二维点(x, y)的数组，`map_matrix` 所表示的仿射变换就是通过这些点来计算。

`cvGetAffineTransform()`中的 `pts_src` 和 `pts_dst` 是其中包含三个点的数组，它们定义了两个平行四边形。描述仿射变换的简单方法是把 `pts_src` 设置为源图像的三个角——例如，源图像的左上角和左下角以及右上角。从源图像到目标图像的映射完全由特定的 `pts_dst` 定义，这三个点的位置将会被映射到目标图像。一旦这三个独立点(实际上是指定一个“有代表性”的平行四边形)^①的映射完成，所有其他点会依次变形。

例 6-2 显示了使用这些函数的一些代码。在这个例子里，我们首先通过创建点(我们表示平行四边形的顶点)的两个三元数组，然后利用 `cvGetAffineTransform()` 将其转换为具体的变换矩阵，从而得到 `cvWarpAffine()` 的矩阵参数。接下来，做一次仿射变换，紧接着将图像旋转。对代表源图像点的数组被称为 `srcTri[]` 的点集，我们提取其中的三个点:(0,0), (0,height-1) 和 (width-1,0)。接着我们指定位置，这些点会在相应数组 `srcTri[]` 中被映射到这些位置。

例 6-2：仿射变换

```
// Usage: warp_affine <image>
//
#include <cv.h>
#include <highgui.h>

int main(int argc, char** argv)
{
    CvPoint2D32f srcTri[3], dstTri[3];
    CvMat* rot_mat = cvCreateMat(2,3,CV_32FC1);
    CvMat* warp_mat = cvCreateMat(2,3,CV_32FC1);
    IplImage *src, *dst;

    if( argc == 2 && ((src=cvLoadImage(argv[1],1)) != 0) ) {
        dst = cvCloneImage( src );

```

① 只需要三个点，因为对于仿射变换来说，只需要表示一个平行四边形。涉及透视变换时，就需要四个点以表示一个普通梯形。

```

dst->origin = src->origin;
cvZero( dst );

// Compute warp matrix
//
srcTri[0].x = 0;                                //src Top left
srcTri[0].y = 0;
srcTri[1].x = src->width - 1;                  //src Top right
srcTri[1].y = 0;
srcTri[2].x = 0;                                //src Bottom left offset
srcTri[2].y = src->height - 1;
dstTri[0].x = src->width*0.0;                   //dst Top left
dstTri[0].y = src->height*0.33;
dstTri[1].x = src->width*0.85;                 //dst Top right
dstTri[1].y = src->height*0.25;
dstTri[2].x = src->width*0.15;                 //dst Bottom left offset
dstTri[2].y = src->height*0.7;

cvGetAffineTransform( srcTri, dstTri, warp_mat );
cvWarpAffine( src, dst, warp_mat );
cvCopy( dst, src );

// Compute rotation matrix
//
CvPoint2D32f center = cvPoint2D32f(
    src->width/2,
    src->height/2
);
double angle = -50.0;
double scale = 0.6;
cv2DRotationMatrix( center, angle, scale, rot_mat );

// Do the transformation
//
cvWarpAffine( src, dst, rot_mat );

cvNamedWindow( "Affine_Transform", 1 );
cvShowImage( "Affine_Transform", dst );
cvWaitKey();
}
cvReleaseImage( &dst );
cvReleaseMat( &rot_mat );

```

```

    cvReleaseMat( &warp_mat );
    return 0;
}
}

```

【167~168】

计算 `map_matrix` 的第二种方法是用 `cv2DRotationMatrix()`，它用来计算围绕任意点的旋转的映射矩阵和一个可选择的尺度。虽然这只是仿射变换可能出现的一种，但代表一个重要的子集，有一个我们更容易理解的替代(且更直观)的表示：

```

CvMat* cv2DRotationMatrix(
    CvPoint2D32f center,
    double      angle,
    double      scale,
    CvMat*     map_matrix
);

```

第一个参数 `center` 是旋转中心。第 2 和 3 个参数给出了旋转的角度和缩放尺度。最后一个参数 `map_matrix` 是输出的映射矩阵，总是一个浮点类型的 2×3 矩阵。

【168】

如果我们定义 $\alpha = \text{scale} \cdot \cos(\text{angle})$ 和 $\beta = \text{scale} \cdot \sin(\text{angle})$ ，那么该函数将如下计算 `map_matrix`：

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot \text{center}_x - \beta \cdot \text{center}_y \\ -\beta & \alpha & \beta \cdot \text{center}_x + (1-\alpha) \cdot \text{center}_y \end{bmatrix}$$

可以结合使用这些设置 `map_matrix` 的方法获得图像的旋转、缩放或变形效果，例如，一幅旋转，缩放并且扭曲的图像。

稀疏仿射变换

如前所述，`cvWarpAffine()` 是解决密集映射的正确方法。对于稀疏映射(例如，对一系列独立点的映射)，最好的办法是用 `cvTransform()`：

```

void cvTransform(
    const CvArr*      src,
    CvArr*            dst,
    const CvMat*      transmat,
    const CvMat*      shiftvec = NULL
);

```

一般情况下，`src` 是 D_s 通道， $N \times 1$ 的数组， N 是将要转换的点的数量， D_s 是这些源图像点的维数。输出的目标数组 `dst` 必须相同大小但可以有不同的通道数 D_d 。

转换矩阵 transmat 是一个 $D_s \times D_d$ 的矩阵，被应用到源图像的每一个元素，结果被置入 dst 中。可选向量 shiftvec ，如果非空，则必须是一个 $D_s \times 1$ 的数组，在结果被置入目标图像之前将其加到 dst 中。

在前面仿射变换的例子中，`cvTransform()` 有两种用法，如何选择取决于我们想如何表达我们的转换。第一种方法中，我们将变换分解成 2×2 部分(完成旋转，缩放和扭曲)以及 2×1 部分(完成转换)。这里我们的输入是一个二通道的 $N \times 1$ 的数组， transmat 是我们的局部齐次变换， shiftvec 包括一些需要的位移。第二种方法是利用我们通常的 2×3 形式的仿射变换。在这种情况下，输入数组 src 是一个 3 通道数组，其中我们必须将全部第三个通道的元素设为 1。(例如，这些点必须在齐次坐标里提供)。当然，输出数组仍将是 2 通道数组。

透视变换

为了获得透视变换(单应性)所提供的更大灵活性，我们需要一个新的函数，使之能实现广泛意义上的变换。首先我们注意到，即使一个透视投影由一个单独矩阵完全确定，但这个投影实际上并不是线性变换。这是因为变换需要与最后一维(通常是 Z ，见第 11 章)元素相除，因此在过程中会失去一维。

【169】

与仿射变换一样，图像操作(密集变换)是由不同的函数来处理的相较于点集变换(稀疏变换)所用的函数，而不是通过对点集的变换(稀疏变换)。

密集透视变换

密集透视变换用到的 OpenCV 函数与提供的密集仿射变换是类似的。特别地，`cvWarpPerspective()` 的参数与 `cvWarpAffine()` 相同，但是有一个小的但很重要的区别是所采用的映射矩阵必须是 3×3 的。

```
void cvWarpPerspective(
    const CvArr*    src,
    CvArr*          dst,
    const CvMat*    map_matrix,
    int              flags      = CV_INTER_LINEAR +
    CV_WARP_FILL_OUTLIERS,
    CvScalar         fillval   = cvScalarAll(0)
);
```

这里的标志与仿射情形的一样。

计算透视映射矩阵

对于仿射变换，在前面的代码里，针对在代码执行期间填充 `map_matrix`，我们有一个方便的函数，可以通过对应点列表计算变换矩阵：

```
CvMat* cvGetPerspectiveTransform(
    const CvPoint2D32f* pts_src,
    const CvPoint2D32f* pts_dst,
    CvMat*             map_matrix
);
```

这里 `pts_src` 和 `pts_dst` 是四个点的数组(而不是三个)，所以我们将能独立地控制如何将 `pts_src` 中矩形的(典型的)四个角映射到 `pts_dst` 中的普通菱形上。我们的变换完全由源图像上的四个点所指定的目标定义。如前所述，对透视变换，我们必须为 `map_matrix` 分配一个 2×3 数组，见例 6-3 的代码。除了 3×3 矩阵和三个控点变为四个控点外，透视变换在其他方面与仿射变换完全类似。

例 6-3：透视变换代码

```
// Usage: warp_affine <image>
//
#include <cv.h>
#include <highgui.h>

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

    CvPoint2D32f srcQuad[4], dstQuad[4];
    CvMat*         warp_matrix = cvCreateMat(3,3,CV_32FC1);
    IplImage*      *src, *dst;
    if( argc == 2 && ((src=cvLoadImage(argv[1],1)) != 0) ) {

        dst = cvCloneImage(src);
        dst->origin = src->origin;
        cvZero(dst);

        srcQuad[0].x = 0;                      //src Top left
        srcQuad[0].y = 0;
        srcQuad[1].x = src->width - 1;        //src Top right
        srcQuad[1].y = 0;
        srcQuad[2].x = 0;                      //src Bottom left
        srcQuad[2].y = src->height - 1;
        srcQuad[3].x = src->width - 1;        //src Bot right
        srcQuad[3].y = src->height - 1;
```

```

dstQuad[0].x = src->width*0.05;           //dst Top left
dstQuad[0].y = src->height*0.33;
dstQuad[1].x = src->width*0.9;            //dst Top right
dstQuad[1].y = src->height*0.25;
dstQuad[2].x = src->width*0.2;            //dst Bottom left
dstQuad[2].y = src->height*0.7;
dstQuad[3].x = src->width*0.8;            //dst Bot right
dstQuad[3].y = src->height*0.9;

cvGetPerspectiveTransform(
    srcQuad,
    dstQuad,
    warp_matrix
);
cvWarpPerspective( src, dst, warp_matrix );
cvNamedWindow( "Perspective_Warp", 1 );
cvShowImage( "Perspective_Warp", dst );
cvWaitKey();
}
cvReleaseImage(&dst);
cvReleaseMat(&warp_matrix);
return 0;
}
}

```

【170~171】

稀疏透视变换

这里有一个特殊函数 `cvPerspectiveTransform()`，在一系列的点上完成透视变换。我们不能用 `cvTransform()`，因为它只局限于线性操作。因为如此，它不能处理透视变换因为透视变换需要齐次表达式的第三坐标来除各项($x = f * X/Z$, $y = f * Y/Z$)。这个特殊的函数 `cvPerspectiveTransform()` 需要引起注意。

```

void cvPerspectiveTransform(
    const CvArr*      src,
    CvArr*            dst,
    const CvMat*      mat
);

```

通常，`src` 和 `dst` 参数(分别)是被变换的原始点数组以及目标点数组。这些数组必须是三通道、浮点类型的。矩阵 `mat` 既可以是 3×3 的，也可以是 4×4 的。如果为 3×3 ，便投影从 2 维变成 2 维；如果是 4×4 的，投影就是从 4 维变成 3 维。

当前，我们是将一幅图像中的点集转换成另一幅图的点集，听起来类似于将二维映射到另外两维。但这并不是很准确的，因为透视变换实际上是一个嵌入在三维空间的二维平面上的实际映射点映射回一个不同的二维子空间。把这个情况想像为摄像机的行为(后文后讨论到摄像机时我们将详细讨论)。摄像机得到三维空间的点，然后利用摄像机成像仪将其映射到二维空间。这实际上是当原始点必须采取“齐次坐标”时的意思。我们为这些点增加额外的一维，即引入 Z 维，并将所有 Z 的值设为 1。投影变换然后将该空间反投影到我们输出的二维空间。这里我们用相当大的篇幅解释了当一个点从一幅图映射到另一幅图，为何需要一个 3×3 矩阵的原因。

图 6-14 显示出例 6-3 中代码对仿射变换和投影变换的输出。将此结果与图 6-13 对比，看看它是如何作用于实际图像的。在图 6-14 中，我们转换了整个图像。但这并不是必须的。我们只需要用 `src_pts` 来定义源图像中一个小(或大)的区域用来转换就行。也可以用源图像或者目标图像的 ROI 区域来限制这个转换。

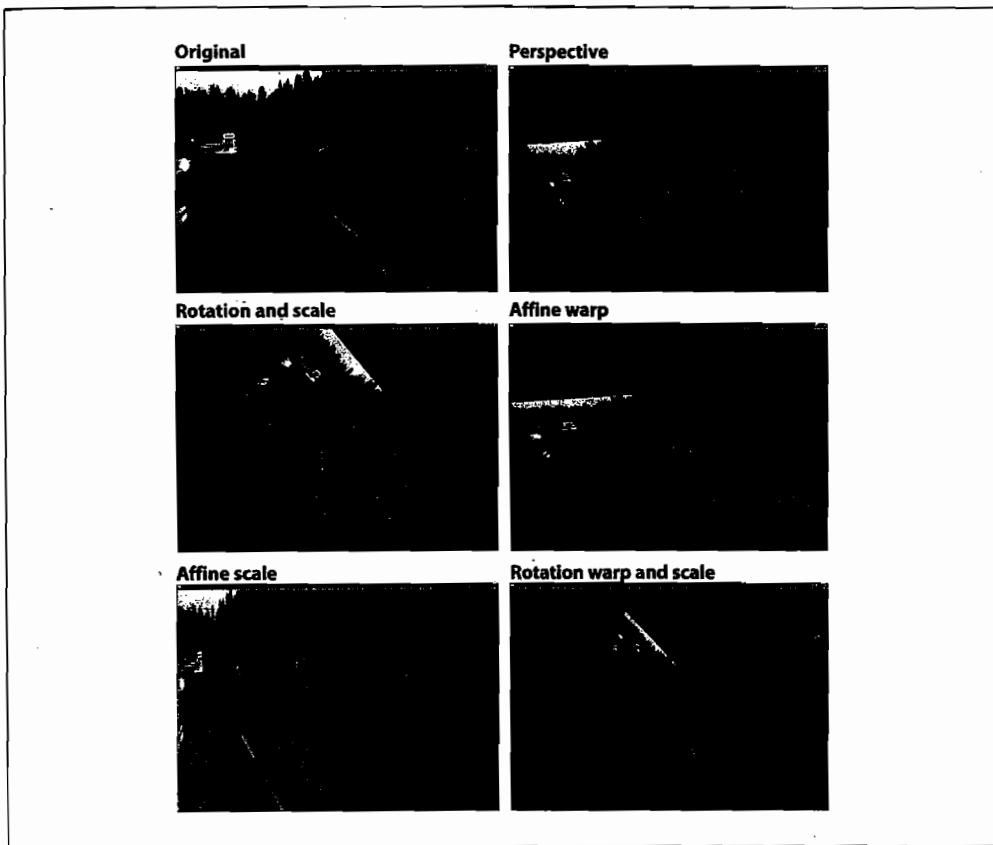


图 6-14：一幅图像的透视变换和仿射变换

CartToPolar 与 PolarToCart

函数 `cvCartToPolar()` 和 `cvPolarToCart()` 会被一些更加复杂的例程诸如 `cvLogPolar()`(后面谈到)用到，但它们本身也是有用的。这些函数将数值在笛卡儿(x,y)空间和极性或者径向(r, θ)空间之间进行映射。(例如，从笛卡儿坐标到极坐标或相反)。函数格式如下：

```
void cvCartToPolar(
    const CvArr* x,
    const CvArr* y,
    CvArr* magnitude,
    CvArr* angle           = NULL,
    int      angle_in_degrees = 0
);
void cvPolarToCart(
    const CvArr* magnitude,
    const CvArr* angle,
    CvArr*       x,
    CvArr*       y,
    int angle_in_degrees = 0
);
```

在这些函数中，前两个二维数组或者图像是输入，后两个是输出。如果输出指针设为空，它将不会被计算。这些数组必须为浮点或者双精度且是相互匹配的(大小、通道数和类型)。最后一个参数指定是用角度($0, 360$)还是用弧度($0, 2\pi$)计算。

这里给出一个可能用到此函数的例子。假定已经对图像的 x 和 y 方向求导，要么使用 `cvSobel()`，要么使用 `cvDFT()` 或 `cvFilter2D()` 做卷积运算。如果将图像的 x 方向导数存储于 `dx_img` 而将 y 方向导数存储于 `dy_img`，那么就可以创建一个边缘-角度识别直方图。就是说，你可以收集所有的角度，假定边缘像素的幅值或强度在一定阈值之上。为了计算它们，我们创建了两个类型一致(整型或浮点型)的目标图像作为求导图像并且调用 `img_mag` 和 `img_angle`。如果想让结果以角度返回，可以用函数 `cvCartToPolar(dx_img, dy_img, img_mag, img_angle, 1)`。这样一来，只要是大于阈值的 `img_mag` 中的相关像素，就可以用之填充来自 `img_angle` 的直方图。

【172~174】

LogPolar

对于二维图像，Log-polar 转换 [Schwartz80] 表示从笛卡儿坐标到极坐标的变换。 $(x, y) \leftrightarrow re^{i\theta}$ ，其中 $r = \sqrt{x^2 + y^2}$ 并且 $\exp(i\theta) = \exp(i \cdot \arctan(y/x))$ 为了将极坐标分离到 $a(\rho, \theta)$ 空间，而这又与一些中心点 (x_c, y_c) 有关，我们用对数运算，于是 $\rho = \log(\sqrt{(x - x_c)^2 + (y - y_c)^2})$ 并且 $\theta = \arctan((y - y_c)/(x - x_c))$ 出于图像的考虑——当需要将感兴趣的元素“填充”到图像缓存中——我们应用一个缩放比例 m 到 ρ 。图 6-15 左边显示了一个正方形对象以及其在极坐标系的编码。

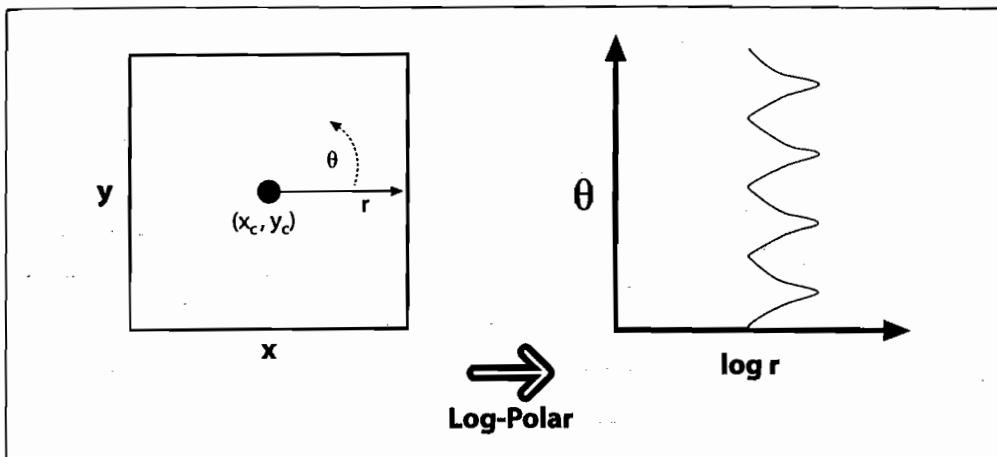


图 6-15：从 (x, y) 到 $(\log(r), \theta)$ 映射的对数极坐标变换；正方形显示在对数极坐标系里如右图所示

下一个问题是“为什么要如此费事？”对数极坐标变换的灵感来自人的视觉系统。你眼睛的中心有一个小但很密集的光感受器(中央凹)，并且受体的密度从这里快速(指数级)下降。尝试着盯着墙上的一个污点，在你的视线中以手臂的长度举着你的手指。接下来，保持对污点的凝视并且缓慢移动你的手指，在手指从你的中央凹移动开的时候，注意到图像的细节迅速下降。这个结构同时也有一个相当优美的数学性质(在本书范围之外)即关注保留交叉线段的角度。

对我们来说更重要的是，对数极坐标变换是对物体视场的一种不变表示，即当变换图像的质心移动到对数极坐标平面的某个固定点时。见图 6-16，左边是三个我们想要识别成“正方形的”形状。问题是它们看起来很不一样。有一个比其余的大很多而另一个是旋转的。对数极坐标变换见图 6-16 的右侧。观察到在 (x, y) 平面内尺寸的差异被旋换为对数极坐标平面内沿着 $\log(r)$ 轴的位移，旋转差异被转换成对数

极坐标平面沿 θ 轴的位移。如果我们将对数极坐标平面上每一个变换方形的转换中心，重新移动到一个固定的中心位置，那么所有正方形都是一样的。这就产生了一类二维旋转和缩放的不变性^①。

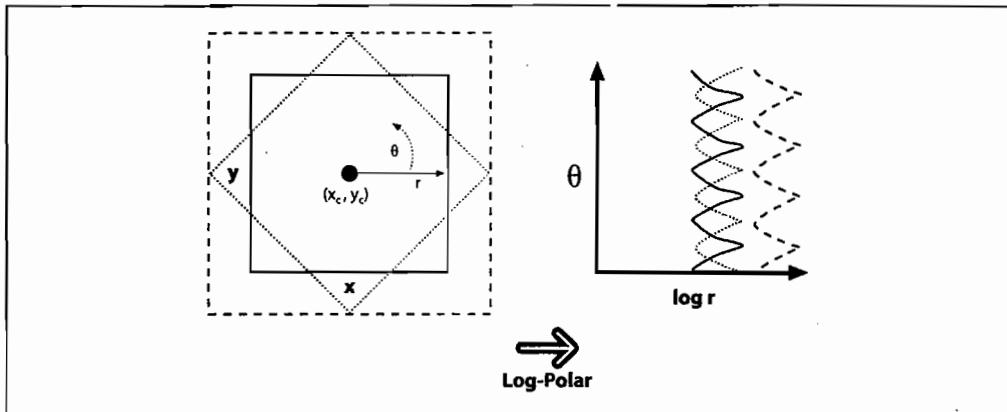


图 6-16：旋转和缩放正方形的对数极坐标变换：尺寸转到 $\log(r)$ 轴的位移，旋转转到 θ 轴的位移

OpenCV 中对于对数极坐标转换的函数是 `cvLogPolar()`:

```
void cvLogPolar(
    const CvArr* src,
    CvArr* dst,
    CvPoint2D32f center,
    double m,
    int flags = CV_INTER_LINEAR | CV_WARP_FILL_OUTLIERS
);
```

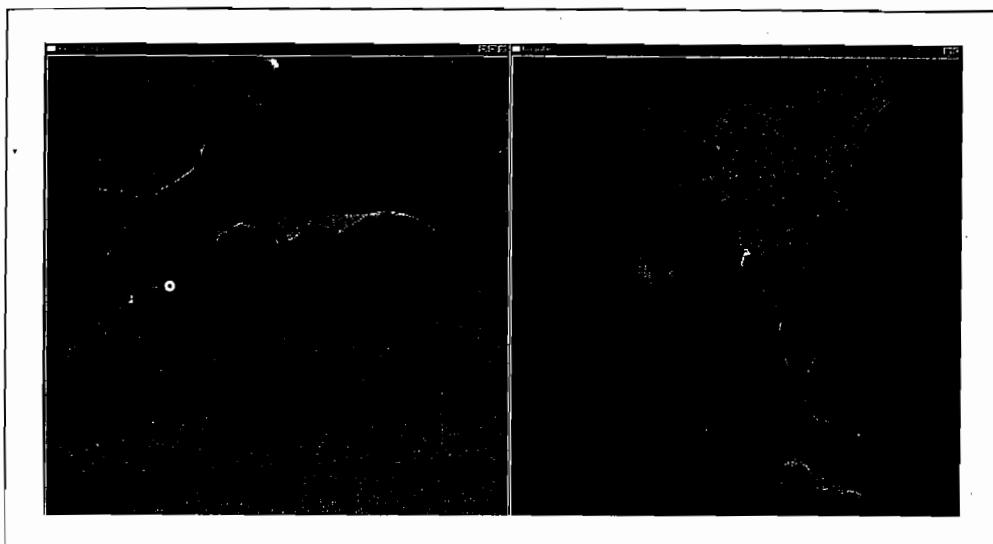
`src` 和 `dst` 是单通道或三通道的彩色或灰度图像。参数 `center` 是对数极坐标变换的中心点 (x_c, y_c) ；`m` 是缩放比例，它应被事先设置使得感兴趣的特征在可用的图像区域内占主导。插值方法的设置与 OpenCV 提供的标准插值方法是一致的（表 6-1）。这些插值方法可以与标志位 `CV_WARP_FILL_OUTLIERS`（填充点否则会未

① 在第 13 章我们将学习识别。现在只需简单地注意到，对于整个物体来说推导其对数极坐标变换并不是一个好的想法，因为这样的变换对于提取它们中心点的位置是很敏感的。更有可能用于物体识别的是检测物体周围的一些关键点（比如角或块的位置），截断这些视线的外围，只利用这些关键点的中心，将其作为对数极坐标的中心。这些局部的对数极坐标变换接下来可以被用作创建局部特征，这些特征是（部分）缩放和旋转常量以及能和可视物体联系起来的一些常量。

定义)和 CV_WARP_INVERSE_MAP(计算从对数极坐标到笛卡儿坐标的逆向映射)中的一个或者二者结合使用。

【174~178】

例 6-4 给出了对数极变换的一些代码示例，论证了向前和向后(逆向)对数极坐标变换。图 6-17 显示了对于一幅摄影图像的变换结果。



例 6-17：一个对数极变换的例子，左图是变换的中心(在麋鹿上的白圈上)，右图是输出结果

例 6-4：对数极变换示例

```
// logPolar.cpp: Defines the entry point for the console application.
#include <cv.h>
#include <highgui.h>
int main(int argc, char** argv) {
    IplImage* src;
    double M;
    if( argc == 3 && ((src=cvLoadImage(argv[1],1)) != 0 )) {
        M = atof(argv[2]);
        IplImage* dst = cvCreateImage( cvGetSize(src), 8, 3 );
        IplImage* src2 = cvCreateImage( cvGetSize(src), 8, 3 );
        cvLogPolar(
            src,
            dst,
            cvPoint2D32f(src->width/4,src->height/2),
            M,
```

```

    CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS
);
cvLogPolar(
    dst,
    src2,
    cvPoint2D32f(src->width/4, src->height/2),
    M,
    CV_INTER_LINEAR | CV_WARP_INVERSE_MAP
);
cvNamedWindow( "log-polar", 1 );
cvShowImage( "log-polar", dst );
cvNamedWindow( "inverse log-polar", 1 );
cvShowImage( "inverse log-polar", src2 );
cvWaitKey();
}
return 0;
}

```

176 - 177】

离散傅里叶变换(DFT)

对任意通过离散(整型)参数索引的数值集合，都可能用类似于连续函数的傅里叶变换的形式定义一个离散傅里叶变换(DFT)^①。对于 N 个复数： x_0, \dots, x_{N-1} ，一维的 DFT 定义为下式(其中 $i = \sqrt{-1}$)：

$$f_k = \sum_{n=0}^{N-1} x_n \exp\left(-\frac{2\pi i}{N} kn\right), k = 0, \dots, N-1$$

对于二维数组(当然高维也相似)，也可定义类似的变换：

$$f_{k_x k_y} = \sum_{n_x=0}^{N_x-1} \sum_{n_y=0}^{N_y-1} x_{n_x n_y} \exp\left(-\frac{2\pi i}{N_x} k_x n_x\right) \exp\left(-\frac{2\pi i}{N_y} k_y n_y\right)$$

- ① Joseph Fourier [Fourier]是第一个发现某些函数可以分解成其他函数的无穷级数，这种做法产生了著名的傅里叶分析。将函数分解成傅里叶级数的一些主要方法参见 Morse 的物理分析[Morse53]以及 Papoulis 的更一般分析[Papoulis62]。快速傅里叶变换是 Cooley 和 Tukey 在 1965 年 [Cooley65] 发明的，尽管 Carl Gauss 早在 1805 年 [Johnson84] 就找出了其中的关键步骤。傅里叶变换早期在计算机视觉中的应用由 Ballard 和 Brown [Ballard82] 首先引入。

通常，人们可能估计 N 个不同的项 f_k 需要 $O(N^2)$ 次操作的计算量。实际上，这里有许多快速傅里叶算法(FFT)能够以 $O(N \log N)$ 的时间复杂度计算这些值。OpenCV 函数 cvDFT() 实现了这样的 FFT 算法。函数 cvDFT() 可以计算输入是一维或二维数组时的 FFT。在后一种情况下，则是计算二维数组的傅里叶变换，如果必要，可以对数组的每一行做一维变换(这个操作比单独多次调用 cvDFT() 快许多)。

```
void cvDFT(
    const CvArr* src,
    CvArr* dst,
    int flags,
    int nonzero_rows = 0
);
```

输入数组和输出数组必须是浮点类型并且通常是单通道或双通道。对单通道情形，输入被设定为实数，输出封装了一种节省空间的格式(从同样是 IplImage 格式的旧 IPL 库里继承来的)。如果源图像和通道是双通道矩阵或图像，这两个通道会被解释成输入数据的实部和虚部。在这种情形下，结果将没有特殊的封装，在输入和输出数组中，一些有许多 0 的空间会被浪费^①。

【177】

这个单通道输出用到的特殊封装结果值如下。

对于一维数组：

$\text{Re } Y_0$	$\text{Re } Y_1$	$\text{Im } Y_1$	$\text{Re } Y_2$	$\text{Im } Y_2$...	$\text{Re } Y_{(Nx/2)-1}$	$\text{Im } Y_{(Nx/2)-1}$	$\text{Re } Y_{(Nx/2)}$
------------------	------------------	------------------	------------------	------------------	-----	---------------------------	---------------------------	-------------------------

对于二维数组：

$\text{Re } Y_{00}$	$\text{Re } Y_{01}$	$\text{Im } Y_{01}$	$\text{Re } Y_{02}$	$\text{Im } Y_{02}$...	$\text{Re } Y_{0(Nx/2)-1}$	$\text{Im } Y_{0(Nx/2)-1}$	$\text{Re } Y_{0(Nx/2)}$
$\text{Re } Y_{10}$	$\text{Re } Y_{11}$	$\text{Im } Y_{11}$	$\text{Re } Y_{12}$	$\text{Im } Y_{12}$...	$\text{Re } Y_{1(Nx/2)-1}$	$\text{Im } Y_{1(Nx/2)-1}$	$\text{Re } Y_{1(Nx/2)}$
$\text{Re } Y_{20}$	$\text{Re } Y_{21}$	$\text{Im } Y_{21}$	$\text{Re } Y_{22}$	$\text{Im } Y_{22}$...	$\text{Re } Y_{2(Nx/2)-1}$	$\text{Im } Y_{2(Nx/2)-1}$	$\text{Re } Y_{2(Nx/2)}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\text{Re } Y_{(Ny/2-1)0}$	$\text{Re } Y_{(Ny/2-1)1}$	$\text{Im } Y_{(Ny/2-1)1}$	$\text{Re } Y_{(Ny/2-1)2}$	$\text{Im } Y_{(Ny/2-1)2}$...	$\text{Re } Y_{(Ny-1)(Nx/2)-1}$	$\text{Im } Y_{(Ny-1)(Nx/2)-1}$	$\text{Re } Y_{(Ny-1)(Nx/2)}$
$\text{Im } Y_{(Ny/2-1)0}$	$\text{Re } Y_{(Ny/2-1)1}$	$\text{Im } Y_{(Ny/2-1)1}$	$\text{Re } Y_{(Ny/2-1)2}$	$\text{Im } Y_{(Ny/2-1)2}$...	$\text{Re } Y_{(Ny-1)(Nx/2)-1}$	$\text{Im } Y_{(Ny-1)(Nx/2)-1}$	$\text{Re } Y_{(Ny-1)(Nx/2)}$
$\text{Re } Y_{(Ny/2)0}$	$\text{Re } Y_{(Ny/2)1}$	$\text{Im } Y_{(Ny/2)1}$	$\text{Re } Y_{(Ny/2)2}$	$\text{Im } Y_{(Ny/2)2}$...	$\text{Re } Y_{(Ny-1)(Nx/2)-1}$	$\text{Im } Y_{(Ny-1)(Nx/2)-1}$	$\text{Re } Y_{(Ny-1)(Nx/2)}$

① 使用这种方法时，若以双通道表示，则必须显式地把虚部都设为 0。一个简单的方法是利用 cvZero() 为虚部创建一个都是 0 的矩阵，然后利用一个实值矩阵调用 cvMerge() 以建立一个临时复数数组来运行 cvDFT()(可能就在本地)。这个过程将会产生频谱的全尺寸、未封装的复数矩阵。

需要花些时间仔细研究这些数组的下标。这里的问题是某些值必须保证是 0(更准确的是, f_k 的某些值必须保证是实数)。还必须注意, 表中最后一行和最后一列分别只有在 N_y 和 N_x 是奇数时才会出现。(在二维数组的情况下, 其被看成是 N_y 一维数组而不是完整的二维数组, 所有结果行将与一维数组中输出的单独一行类似)。

【178】

第三个参数称为 flags, 明确指出要进行什么操作。开始的变换称为正变换, 选择的标志位是 CV_DXT_FORWARD。除了指数的符号和缩放比例有变化外, 逆变换^①的定义方式几乎相同。为了执行没有缩放比例的变换, 要用到 CV_DXT_INVERSE 标志位。缩放比例的标志位是 CV_DXT_SCALE, 输出的结果都被比例 $1/N$ 缩放(二维变换是 $1/N_x N_y$)。如果对序列应用正变换, 然后用逆变换回到初始的位置, 那么这个缩放是必须的。由于经常把 CV_DXT_INVERSE 和 CV_DXT_SCALE 结合起来使用, 所以对这样的操作有许多速记符号。另外除了刚才使用“或”将这两个操作结合起来之外, 也可以用 CV_DXT_INV_SCALE(如果手边必备的不用那个简短的方式, 可以用 CV_DXT_INVERSE_SCALE)。最后一个想随手使用的标志位是 CV_DXT_ROWS, 它告诉 cvDFT() 将一个二维数组看做一维数组的集合, 即把它们看成长为 N_x 的 N_y 个不同的一维向量分别进行转换。这大大减少了变换的系统开销(特别是用英特尔的优化库 IPP 时)。通过用 CV_DXT_ROWS, 可能实现对三维(或者更高)DFT 的转换。

为了理解最后一个参数 nonzero_rows, 我们必须先偏离一下正题。通常情况下, DFT 算法特别喜欢用一些有特殊长度的向量或者特殊尺寸的数组。在大多数 DFT 算法中, 首选的尺寸是 2 的幂次(一些整型数 n 的 2^n 次幂)。当 OpenCV 使用该算法时, 首选的向量长度或者说数组维数为 $2^p 3^q 5^r$, p, qr 是指数。因此, 常用方法是创建一个稍微大一些的数组(有一个很好用的函数 cvGetOptimalDFTSize() 可达此目的, 它获取向量的长度, 返回一个相等或较大的合适尺寸值), 接下来用 cvGetSubRect(), 将数组复制到这个比较大些的 0 填充的数组中。尽管需要这样的填充, 还是有可能向 cvDFT() 表示你并不在意这些必须添加实际数据的行的转换(或者, 如果正在做逆变换, 则不在意结果中的行)。在这两种情况下, 可以用 nonzero_rows 来指明有多少行可以被安全地忽略。这样做可以节省一些计算时间。

① 随着逆变换的进行, 输入被以一种前面描述的特殊格式封装。这样做的原因是我们首先调用正向 DFT, 接下来对结果运行逆 DFT, 并希望得到原始数据——不要忘记使用 CV_DXT_SCALE 标志位!

频谱乘法

在许多包含计算 DFT 的应用中，还必须将两个频谱中的每个元素分别相乘。由于 DFT 的结果是以其特殊的高密度格式封装，并且通常是复数，解除它们的封装以及通过“普通”矩阵操作来进行乘法运算是很乏味的。幸运的是，OpenCV 提供了一个很方便的程序 `cvMulSpectrums()`，它可以准确执行这个函数以及执行一些其他方便的事情。

【179】

```
void cvMulSpectrums(
    const CvArr* src1,
    const CvArr* src2,
    CvArr*       dst,
    int          flags
);
```

注意，前两个输入变量是常见的输入数组，尽管这里下它们是被 `cvDFT()` 调用的频谱。第三个参数必须是一个数组的指针——与前两个类型大小一致——将用于结果中。最后一个参数 `flags`，告诉 `cvMulSpectrums()` 想做什么。具体而言，执行上面两个乘法时，它可以被设为 0(`CV_DXT_FORWARD`)，如果第一个数组的元素与第二个数组的对应元素为复共轭时，它可以被设为 `CV_DXT_MUL_CONJ`。如果每一个数组行 0 被看作是单独的频谱，在二维情况下，这个标志位也可以与 `CV_DXT_ROWS` 组合。(记住，如果用 `CV_DXT_ROWS` 创建一个频谱数组，封装数据与不用这个函数创建有一些轻微的不同，所以必须坚持这种调用 `cvMulSpectrums` 的方法)。

卷积和 DFT

利用 DFT 可以大大加快卷积运算的速度[Titchmarsh26]，因为卷积定理说明空间域的卷积运算可以转换为频域的乘法运算[Morse53; Bracewell65; Arfken85]^①。为了完成这个任务，首先计算图像的傅里叶变换，接着计算卷积滤波器的傅里叶变换。一旦完成这个，可以在变换域中以相对于图像像素数目的线性时间内进行卷积运算。值得看此类卷积运算的源代码，它也将为我们提供许多 `cvDFT()` 的用法范例。见例 6-5，直接取自 OpenCV 参考文档。

① 如前所述，OpenCV 的 DFT 算法实现了 FFT，无论数据的大小，都可以使 FFT 更快。

例 6-5：用 cvDFT() 加快卷积的计算

```
// Use DFT to accelerate the convolution of array A by kernel B.  
// Place the result in array V.  
void speedy_convolution()  
{  
    const CvMat* A, // Size: M1xN1:  
    const CvMat* B, // Size: M2xN2:  
    CvMat* C // Size:(A->rows+B->rows-1)x(A->cols+B->cols-1)  
  
    int dft_M = cvGetOptimalDFTSize( A->rows+B->rows-1 );  
    int dft_N = cvGetOptimalDFTSize( A->cols+B->cols-1 );  
  
    CvMat* dft_A = cvCreateMat( dft_M, dft_N, A->type );  
    CvMat* dft_B = cvCreateMat( dft_M, dft_N, B->type );  
    CvMat tmp;  
  
    // copy A to dft_A and pad dft_A with zeros  
    cvGetSubRect( dft_A, &tmp, cvRect(0,0,A->cols,A->rows));  
    cvCopy( A, &tmp );  
    cvGetSubRect(  
        dft_A,  
        &tmp,  
        cvRect( A->cols, 0, dft_A->cols-A->cols, A->rows )  
    );  
    cvZero( &tmp );  
  
    // no need to pad bottom part of dft_A with zeros because of  
    // use nonzero_rows parameter in cvDFT() call below  
    cvDFT( dft_A, dft_A, CV_DXT_FORWARD, A->rows );  
  
    // repeat the same with the second array  
    cvGetSubRect( dft_B, &tmp, cvRect(0,0,B->cols,B->rows) );  
    cvCopy( B, &tmp );  
    cvGetSubRect(  
        dft_B,  
        &tmp,  
        cvRect( B->cols, 0, dft_B->cols-B->cols, B->rows )  
    );  
    cvZero( &tmp );  
  
    // no need to pad bottom part of dft_B with zeros because of
```

```

// use nonzero_rows parameter in cvDFT() call below
cvDFT( dft_B, dft_B, CV_DXT_FORWARD, B->rows );

// or CV_DXT_MUL_CONJ to get correlation rather than convolution
cvMulSpectrums( dft_A, dft_B, dft_A, 0 );

// calculate only the top part
cvDFT( dft_A, dft_A, CV_DXT_INV_SCALE, C->rows );
cvGetSubRect( dft_A, &tmp, cvRect(0,0,conv->cols,C->rows) );
cvCopy( &tmp, C );
cvReleaseMat( dft_A );
cvReleaseMat( dft_B );

```

在例 6-5 中，我们看到输入数组首先被创建并初始化。接下来，创建两个对于 DFT 算法维数最佳的新数组。原始数组被复制到新的数组，接着计算变换，最后，将所有频谱元素相乘，对乘积进行逆变换。FFT 变换是这个操作里最慢^①的部分：一个 $N \times N$ 图像用 $O(N^2 \log N)$ 的时间，所以整个过程也基本上是在这个时间内完成的(对于一个 $M \times M$ 的卷积核，假定 $N > M$)。这个时间要比使用非 DFT 卷积所花费的时间 $O(N^2 M^2)$ 快很多。

【180~182】

离散余弦变换(DCT)

对于实数，通常计算离散傅里叶变换的一半就已经足够了。离散余弦变换(DCT) [Ahmed74; Jain77]与 DFT 类似，被定义为下式：

$$c_k = \sum_{n=0}^{N-1} \sqrt{n = \begin{cases} \frac{1}{N} & \text{if } n=0 \\ \frac{2}{N} & \text{else} \end{cases}} \cdot x_n \cdot \cos\left(-\pi \frac{(2k+1)n}{N}\right)$$

注意，通常，归一化比例可以同时应用于余弦变换和它的逆变换。当然，对于高维也有相似的变换。

把 DFT 的基本思想应用于 DCT，但现在所有系数都是实数。机敏的读者可能反对

① 这里“最慢”的意思是“渐近的最慢”——也就是说，算法中的这个部分对于每个很大的 N 花了大部分的时间。这是一个非常重要的差别。在实际中，如前面的卷积，变换到傅里叶空间所涉及的开销并不总是最优的。通常，使用小的核做卷积时，费尽周折做这个变换并不值得。

把余弦变换应用到向量，因为它不是一个明显的偶函数。然而，对于 cvDCT()，算法只是简单地把向量做镜像处理，使之具有负的下标即可。

实际的 OpenCV 调用如下：

```
void cvDCT(
    const CvArr* src,
    CvArr*       dst,
    int          flags
);
```

cvDCT() 函数的预期参数与 cvDFT() 预期的一样，因为结果是实值的没有必要将结果数组以特殊形式封装(或者在逆变换里的输入数组)。flags 参数可以设成 CV_DXT_FORWARD 或者 CV_DXT_INVERSE，也可以与 CV_DXT_ROWS 结合，效果与 cvDFT() 类似。因为采用了不同的归一化方式，所有的正向和逆向余弦变换通常都包含它们各自对变换整体归一化的贡献。因此 CV_DXT_SCALE 在 cvDCT 没有发挥作用。

积分图像

OpenCV 可用于轻松计算积分图像，只要用一个具有相应名称 cvIntegral() 的函数。积分图[Viola04]是一个数据结构，可实现子区域的快速求和。这样的求和在许多应用中是很有用的，最显著的是在人脸识别及相关算法中应用的 Haar 小波(Haar wavelet)。

```
void cvIntegral(
    const CvArr* image,
    CvArr*       sum,
    CvArr*       sqsum     = NULL,
    CvArr*       tilted_sum = NULL
);
```

cvIntegral() 的参数是原始图像，也是结果中目标图像的指针。参数 sum 是需要的，其他的 sqsum 和 tilted_sum 如果需要也是可以提供(实际上，这些参数不一定是图像，可以使矩阵，但在实际中，它们通常都是图像)。当输入图像是 8 位无符号类型时，sqsum 和 tilted_sum 可以是 32 位整型或浮点型数组。对于其他情况，sum 和 tilted_sum 必须是浮点类型的(32 位或者 64 位)。结果 “images” 必须始终是浮点型。如果输入图像大小是 $W \times H$ ，输出图像的大小必

须为 $(W + 1) \times (H + 1)$ ^①。

积分图求和形式如下：

$$\text{sum}(X, Y) = \sum_{x \leq X} \sum_{y \leq Y} \text{image}(x, y)$$

可选的 `sqsum` 图是和的平方如下：

$$\text{sum}(X, Y) = \sum_{x \leq X} \sum_{y \leq Y} (\text{image}(x, y))^2$$

`tilted_sum` 除了求和之外还将图像旋转了 45 度：

$$\text{tilt_sum}(X, Y) = \sum_{x \leq X} \sum_{\text{abs}(x - X) \leq Y} \text{image}(x, y)$$

利用这些积分图，可以计算图像的任意直立或“倾斜”的矩形区域的之和、均值和标准差。例如一个简单的例子，计算一个简单矩形区域的和，这个区域是通过角点 (x_1, y_1) 和 (x_2, y_2) 定位的，这里 $x_2 > x_1$ 并且 $y_2 > y_1$ ，计算如下：

$$\begin{aligned} & \sum_{x_1 \leq x \leq x_2} \sum_{y_1 \leq y \leq y_2} [\text{image}(x, y)] \\ &= [\text{sum}(x_2, y_2) - \text{sum}(x_1 - 1, y_2) - \text{sum}(x_2, y_1 - 1) + \text{sum}(x_1 - 1, y_1 - 1)] \end{aligned}$$

在这种方式下，就可能进行快速模糊、梯度估计、计算均值和标准差，甚至为各种窗口大小执行快速的可变窗块相关计算。【182~183】

为了更加清晰地加以描述，考虑图 6-18 显示的 7×5 的图像，这个区域以柱状图显示，高度代表关联的像素值的亮度。相同的信息显示在图 6-19 中，左边是数字形式，右边是积分形式。依次遍历每行计算积分图(I')，利用前期计算的积分图的值按行加上现在的原始图像(I)像素值 $I(x, y)$ 来计算下一个积分图像值，如下：

$$I'(x, y) = I(x, y) + I'(x - 1, y) + I'(x, y - 1) - I'(x - 1, y - 1)$$

最后一项是消减，因为这个值在加上第二和第三项时重复计数了。可以通过测试图 6-19 中的一些数据来加以验证。

利用积分图像计算一个区域时，通过图 6-19 可以看到，为了计算原始图像中以 20 为界的中心矩形区域，我们如此计算 $398 - 9 - 10 + 1 = 380$ 。因此，利用 4 个变量就可以计算任意尺寸的矩形(产生 $O(1)$ 的计算复杂度)。

① 因为我们需要沿着 X 轴和 Y 轴建立一个 0 值的缓冲区以提高计算效率。

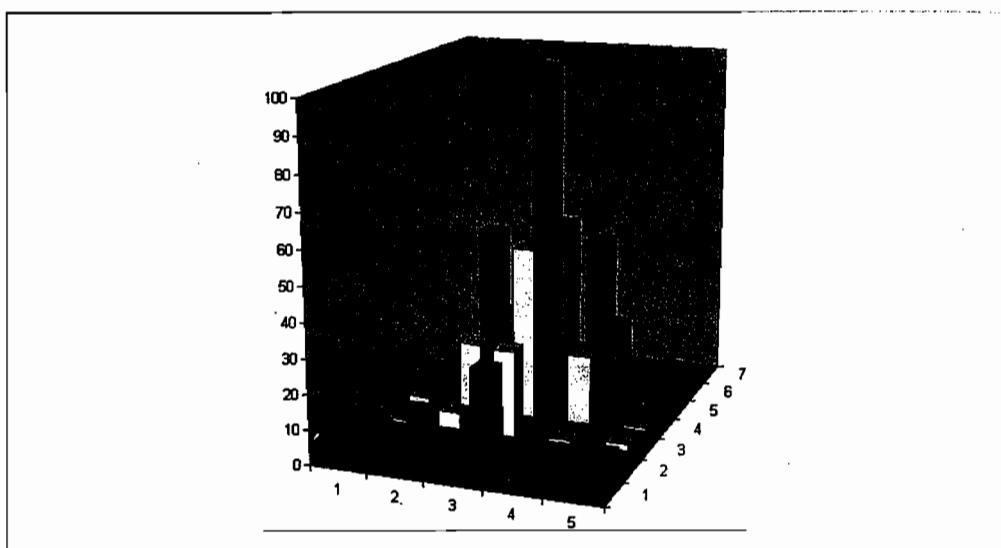


图 6-18：简单的 7×5 图像，它的 x, y 和高等于像素值，以条形图的形式显示

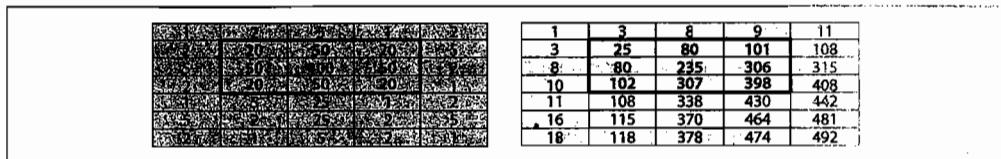


图 6-19：左侧是用数字显示的是图 6-18 中的 7×5 图像(假定左上角是原点)，右侧是转换的积分图像

【184 ~ 185】

距离变换

图像的距离变换被定义为一幅新图像，该图像的每个输出像素被设成与输入像素中 0 像素最近的距离。显然，典型的距离变换的输入应为某些边缘图像。在多数应用中，距离变换的输入是例如 Canny 边缘检测的检测图像的转换输出(即边缘的值是 0，非边缘的值非 0)。

在实际中，距离变换通常是利用 3×3 或 5×5 数组掩膜进行的。数组中的每个点被定义为这个特殊位置同其相关的掩膜中心的距离。较大的距离以由整个掩膜定义的“动作”序列的形式被建立(这样近似)。这就意味着要用更大的掩膜将生成更准确的距离。

根据预期的距离准则，会自动从 OpenCV 的已知的集合中选择适当的掩膜。同样

也可能告诉 OpenCV 根据与选取矩阵相适应的公式去计算“准确”距离，当然这种方式要慢一些。

距离准则可以有许多不同的形式，包括经典的 L2(笛卡儿)距离准则；见表 6-2 所列出的。另外，可以定义一个特殊的准则及与之相关的特殊矩阵。

表 6-2: cvDistTransform()中可能采用的距离准则

distance_type 的值	准则
CV_DIST_L2	$\rho(r) = \frac{r^2}{2}$
CV_DIST_L1	$\rho(r) = r$
CV_DIST_L12	$\rho(r) = 2\left[\sqrt{1 + \frac{r^2}{2}} - 1\right]$
CV_DIST_FAIR	$\rho(r) = c^2\left[\frac{r}{c} - \log\left(1 + \frac{r}{c}\right)\right], c = 1.3998$
CV_DIST_WELSCH	$\rho(r) = \frac{c^2}{2}\left[1 - \exp\left(-\left(\frac{r}{c}\right)^2\right)\right], c = 2.9846$
CV_DIST_USER	用户定义的距离

调用 OpenCV 的距离变换函数时，输出图像必须是 32 位浮点类型图像(例如 IPL_DEPTH_32F)。

```
Void cvDistTransform(
    const CvArr* src,
    CvArr* dst,
    int distance_type = CV_DIST_L2,
    int mask_size = 3,
    const float* kernel = NULL,
    CvArr* labels = NULL
);
```

调用 cvDistTransform()时有许多可选择的参数。第一个是 distance_type，它指定距离准则。这个参数可用的值是由 Borgefors (1986) [Borgefors86] 定义的。

distance_type 之后是 mask_size，可以是 3(选择 CV_DIST_MASK_3)或者 5(选择 CV_DIST_MASK_5)；二者选一，相应地，距离的计算也可以没有核(选择 CV_DIST_MASK_PRECISE)^①。cvDistanceTransform()中的 kernel 参数是在特殊

① 准确的方法来自 Pedro F. Felzenszwalb 和 Daniel P. Huttenlocher [Felzenszwalb63]。

准则情况下的距离掩膜。这些核是根据 Gunilla Borgefors 的方法建立起来的，图 6-20 给出了两个这样的例子。最后一个参数 `labels`，说明每个点与由 0 像素组成的最近连接部分之间的关系。当 `label` 是 `NULL` 时，它必须是一个指向整型值数组的指针，这个数组与输入输出图像的大小相同。当函数返回时，读取这幅图像可以确定哪个目标同考虑中的特殊像素点最接近。图 6-21 显示了对一个测试图案和一幅相片进行距离变换的输出。

用户定义的 3×3 掩码 $(a=1, b=1.5)$	用户定义的 5×5 掩码 $(a=1, b=1.5, c=2)$
	

图 6-20：两个自定义距离掩码

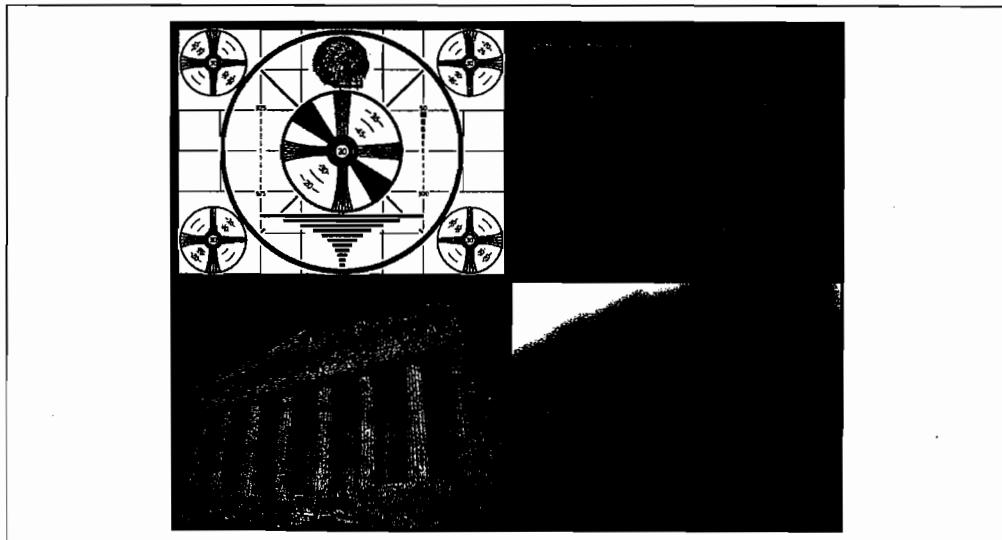


图 6-21：首先运行参数 `1=100`，参数 `2=200` 的 Canny 边缘检测算子，然后运行距离变换，输出按照比例为 5 的比例来缩放以增强可视性

直方图均衡化

通常，摄像机和图像传感器必须不仅仅处理场景的对比度，还必须让图像传感器将这个场景的灯光曝光到结果里。在一个标准的摄像机里，快门和透镜孔的设置会改变传感器的曝光量。通常传感器要处理的对比度范围往往过大而无法处理，因此在捕获黑暗区域(例如：阴影)里和明亮区域之间有一个折衷，黑暗区域需要一个长的曝光时间，而明亮区域需要较短的曝光以避免过饱和。

【186~187】

在指摄完照片之后，对于传感器的记录，我们已经没有什么可做。然而，我们仍然可以对图像做点事情并且尝试扩大图像的动态范围。对这个操作最常用的技术是直方图均衡化。^①在图 6-22 中我们看到左边的图像比较暗因为其数值范围变化较小，这点可以从右边其亮度数值的直方图里明显看出。因为我们处理的是 8 位图像，其亮度值是从 0 到 255，但是直方图显示实际亮度值却集中在亮度范围的中间附近。直方图均衡化就是一种将这个范围拉伸的方法，

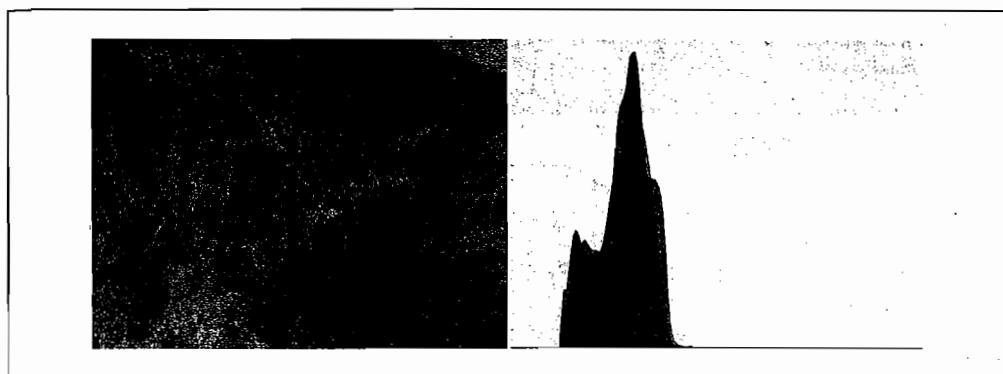


图 6-22：左边图像的对比度比较弱，右边的亮度直方图对此进行了验证

直方图均衡化后面潜在的数学原理是一个分布(输入的亮度直方图)被映射到另一个分布(一个更宽，理想统一的亮度值分布)。也就是说，我们希望把原始分布中 y 轴的值在新分布中尽可能展开。这说明对拉伸数值分布的问题我们有一个比较好的答案：映射函数应该是一个累积分布函数。图 6-23 是一个累积密度函数的例子，这

① 可能你会奇怪为什么直方图均衡化不在第 7 章讲述，其原因是直方图均衡化并不是直接显式地利用任何直方图数据类型。尽管已用到直方图，但这个函数(从用户的角度)根本不需要任何直方图。

直方图均衡化是一种较老的数学技术，各种教科书[Jain86; Russ02; Acharya05]、会议论文[Laughlin81]甚至生物视觉[Laughlin81]都描述了它在图像处理中的应用。

个函数是单纯高斯分布的理想化情况。然而，累积密度可以应用到任何分布，它只是运算从负边界到正边界的原始分布的和。

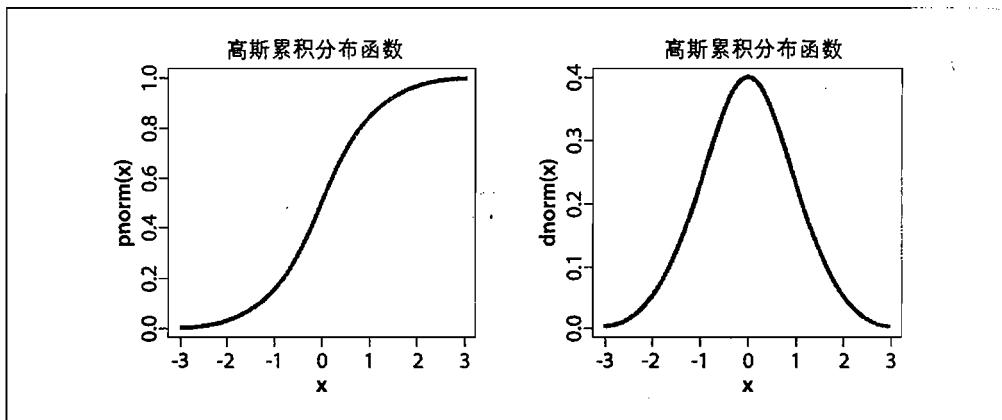


图 6-23：高斯分布(右)中累积分布函数(左)的结果

我们可以用累积分布函数来对原始分布进行映射以作为均衡拉伸分布(见图 6-24)，只需要看原始分布的 y 值在均衡分布中位于何处。

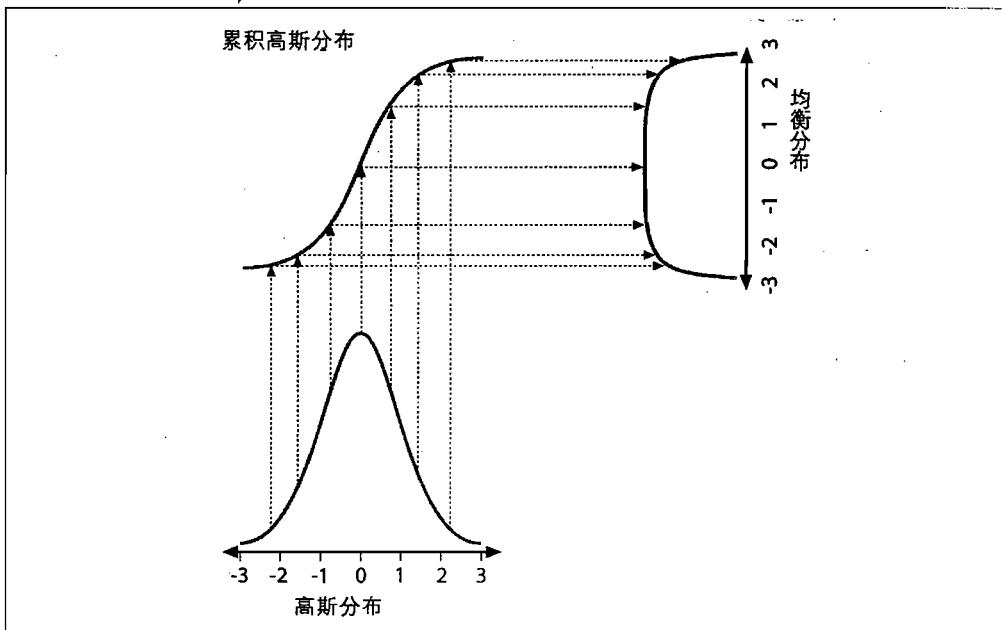


图 6-24：利用累积密度函数均衡化高斯分布

对于连续分布，结果将是准确的均衡化，但对于数字/离散分布，结果可能会有很

大差异。

将均衡化过程应用到图 6-22，就产生均衡化的亮度分布直方图以及结果图像(见图 6-25)。整个过程封装在一个简洁的函数里：

```
void cvEqualizeHist(  
    const CvArr* src,  
    CvArr*       dst  
) ;
```

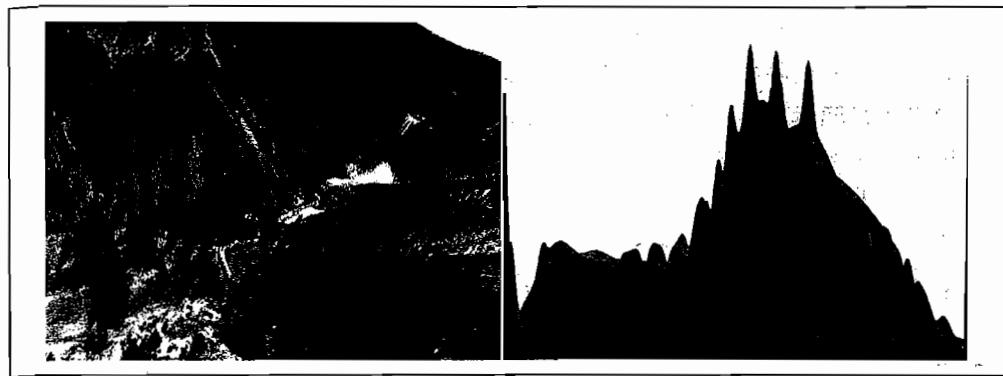


图 6-25：直方图均衡化结果：频谱被展开

在 `cvEqualizeHist()` 中，原始图像及目标图像必须是单通道，大小相同的 8 位图像。对于彩色图像，必须先将每个通道分开，再分别进行处理。 【188~189】

练习

1. 用 `cvFilter2D()` 创建一个滤波器，只检测一幅图像里 60° 角的直线。将结果显示在一个有趣的图像场景上。
2. 可分核。利用行 $[(1/16, 2/16, 1/16), (2/16, 4/16, 2/16), (1/16, 2/16, 1/16)]$ 和在中间的参考点创建一个 3×3 的高斯核。
 - a. 在一幅图像上运行此核并且显示这个结果。
 - b. 现在创建参考点在中心的两个核：一个“交叉” $(1/4, 2/4, 1/4)$ ，另一个下降 $(1/4, 2/4, 1/4)$ 。载入相同的原始图像，利用 `cvFilter2D()` 对图像做两次卷积，第一次用第一个一维核，第二次用第二个一维核。描述结果。
 - c. 描述在 a 和 b 核的复杂度(操作次数)。差别就是可分核和所有高斯类滤波

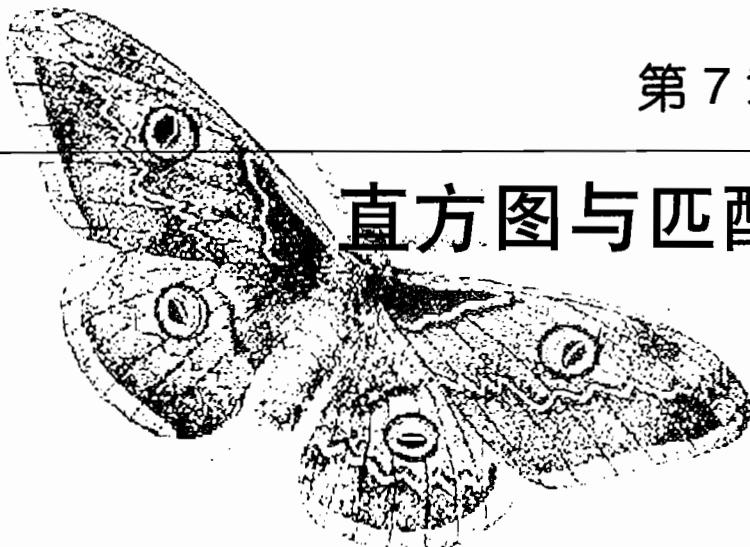
器的优点——或任何线性可分滤波器的优点，因为卷积是一个线性操作。

3. 你能从图 6-5 显示的滤波器里建立一个可分核吗？如果可以，显示一下。
4. 在一个画图程序里，例如 PowerPoint，画一系列同心圆形成一个靶心。
 - a. 构造一系列进入靶心的直线，保存图像。
 - b. 用一个 3×3 的中孔大小，对你的图像运行并显示一阶 x 和 y 方向的梯度，然后将中孔大小增加到 5×5 , 9×9 和 13×13 。描述结果。
5. 创建一幅新图像，其中只有 45° 直线，背景为黑，直线为白。给出一系列中孔尺寸，我们将要得到图像的一阶 x 方向导数(dx)和一阶 y 方向导数(dy)，然后对这条直线采取以下测量方法。 d_x 和 d_y 图像组成了输入图像的梯度。在点 (i,j) 的大小是 $(i,j) = \sqrt{dx^2(i,j) + dy^2(i,j)}$ ；且角度是 $\theta(i,j) = \arctan(dy(i,j)/dx(i,j))$ 。扫描整幅图像并寻找幅值最大或者最大附近的位置。记录这些位置的角度。将角度求平均，记录为直线的测量角。
 - a. 用 3×3 中孔的 Sobel 滤波器完成。
 - b. 用 5×5 滤波器完成。
 - c. 用 9×9 滤波器完成。
 - d. 结果变了吗？如果变了，为什么？
6. 找到并载入一幅正面人脸图，眼睛是睁开的，并且占了图像的大部分区域。写代码找出眼镜的瞳孔。

一个拉普拉斯算子“像”黑暗中的一个中心亮点。瞳孔正好相反，用一个足够大的拉普拉斯算子进行转换和卷积。
7. 这个练习，我们通过设置 cvCanny() 中好的低阈值和高阈值，对参数进行试验。载入一幅合适的线性结构图像。我们用三种不同的高：低的阈值设置，分别是 $1.5 : 1$, $2.75 : 1$ 和 $4 : 1$ 。
 - a. 当高阈值设置小于 50 时你看到了什么。
 - b. 当高阈值设置大于 50 小于 100 时你看到了什么。
 - c. 当高阈值设置在 100 和 150 之间时你看到了什么。
 - d. 当高阈值设置在 150 和 200 之间时你看到了什么。
 - e. 当高阈值设置在 200 和 250 之间时你看到了什么。
 - f. 总结结果，尽量提供清楚的解释。

8. 载入一个包括清晰直线和圆的图像，比如一辆侧面看的自行车。调用霍夫直线变换和霍夫圆变换看这幅图像如何反应。
9. 你能想出办法利用霍夫变换来识别不同周长的任意形状吗？解释具体做法。
10. 参见 log-polar 函数将正方形转换为波浪线的图表。
 - a. 如果 log-polar 的中心点在正方形的一角，画出 log-polar 的结果。
 - b. 如果中心点在圆里边靠近边缘，那么 log-polar 变换中的圆的外观？
 - c. 如果中心点正好在圆的外边，画出这个变换的外观。
11. 一个对数极坐标变换采取不同方向和大小的形状到一个空间，这个空间是相应的 θ 轴和 $\log(r)$ 轴的位移。傅里叶变换是平移不变的。我们怎样利用上述事实迫使不同大小和方向的形状自动在对数极坐标域下给出等价表现形式？
12. 分别画出大矩形、小矩形、大旋转角度和小旋转角度的矩形。分别对它们进行对数极坐标变换。编写二维转换器，这个转换器在结果对数极坐标范围内取中心点，将各种性状尽可能转换为可识别形式。
13. 对一个小的高斯分布和图像分别做傅里叶变换。将其相乘，并对结果进行逆傅里叶变换。会得到什么结果？随着滤波器的变大，你会发现在傅里叶空间运行要比在普通空间快很多。
14. 载入一幅感兴趣的图像，转换成灰度图，然后得到它的积分图。现在利用积分图的性质找到图像里的横向和纵向边缘。
利用细长的矩形，在适当的位置减去和加上它们。
15. 当比例已知且保持固定时，解释一下如何用距离变换将一个测试形状自动排列一个已知形状。如果是多尺度，又将如何进行？
16. 载入一幅图像练习直方图均衡化，报告结果。
17. 载入一幅图像，进行透视变换，然后旋转。这些变换可以在一步之内完成吗？

直方图与匹配



在分析图像、物体和视频信息的过程中，我们常常想把眼中看到的对字用直方图(histogram)表示。直方图可以用来描述各种不同的事情，如物体的色彩分布、物体边缘梯度模板[Freeman95]，以及表示目标位置的当前假设(目标当前位置的假设?)的概率分布。图 7-1 显示了利用直方图进行快速姿态识别。从“上”，“右”，“左”，“停”和“OK”等手势中得到边缘梯度。然后设置一个摄像机，该摄像机观察人的各种手势以控制网络视频。在每帧中，从输入的视频中检测感兴趣的色彩区域，然后计算这些感兴趣区域周围的边缘梯度方向，将得到的边缘梯度方向放到一个方向直方图相应的 bin 中，然后将该直方图与手势模板进行匹配，从而识别出各种手势。图 7-1 的垂直条显示不同手势的匹配程度。灰色的水平线为可接受阈值，表示对应某一手势模型胜出的垂直条。

直方图广泛应用于很多计算机视觉应用中。通过标记帧与帧之间显著的边缘和颜色的统计变化，直方图被用来检测视频中场景的变换。通过为每个兴趣点设置一个有相近特征的直方图所构成的“标签”，用以确定图像中的兴趣点。边缘、色彩、角等直方图构成了可以被传递给目标识别分类器的一个通用特征类型。色彩和边缘的直方图序列还可以用来识别网络视频是否被复制等。直方图是计算机视觉中最经典的工具之一。

简单地说，直方图就是对数据进行统计，将统计值组织到一系列事先定义好的 bin 中。bin 中的数值是从数据中计算出的特征的统计量，这些数据可以是诸如梯度、方向、色彩或任何其他特征。无论如何，直方图获得的是数据分布的统计图。通常直方图的维数要低于原始数据。图 7-2 刻画了一个典型情况。图中显示了一个二维分布的点集(左上)，施加一个网格(右上)并且统计每一个网格单元的数据点，然后产生一个一维直方图(右下)。由于原始数据点可以表征任何事情，直方图实际上是一

一个方便表示图像特征的手段。

【194】

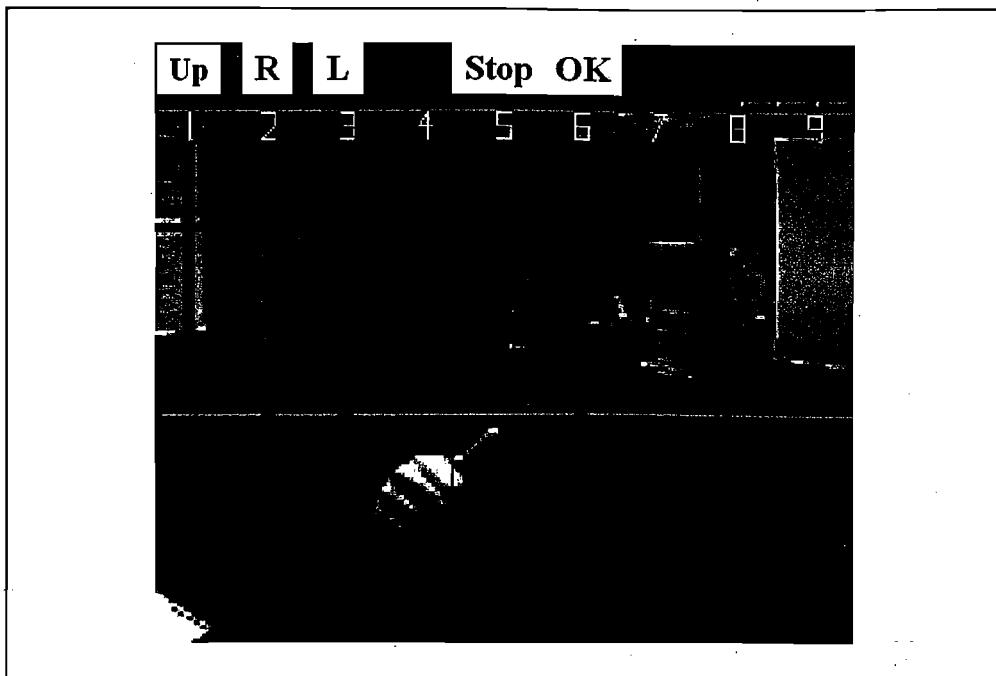


图 7-1：方向梯度的局部直方图，用以寻找手及其特征(姿态)。这里“胜出”的姿态(最长的垂直条)就是正确识别的“L”(向左移动)

表示连续分布的直方图通过隐式计算每个网格单元中点的均值来实现前面的功能^①。这就会产生一个问题，如图 7-3 所示。如果网格太宽(左上)，则参与计算平均值的点太多，就会丧失分布的结构。如果网格太窄(右上)，则没有足够的点来准确表示分布而且我们会得到小而尖锐的单元。

OpenCV 有表征直方图的数据类型。该直方图数据结构能够以一维或者多维的方式表示直方图，并且包含所有可能跟踪的均匀或非均匀的 bin 中的数据。并且，如我们所期待的，它可以配属各种有用的函数，使得我们能够在直方图上容易地进行各种常见操作。

【195】

① 当直方图所用的 bin 少于需要或要求的合乎常情的描述更少的 bin 时，其表示的图像信息很自然的落到离散的 bin 里。举例来说，用一个 10 个 bin 的直方图来表示 8 位亮度值的图像：每个 bin 会把对应于大约 25 个不同的灰度级的点放在一起，(有可能是错误地)将这些不同灰度级同等对待。

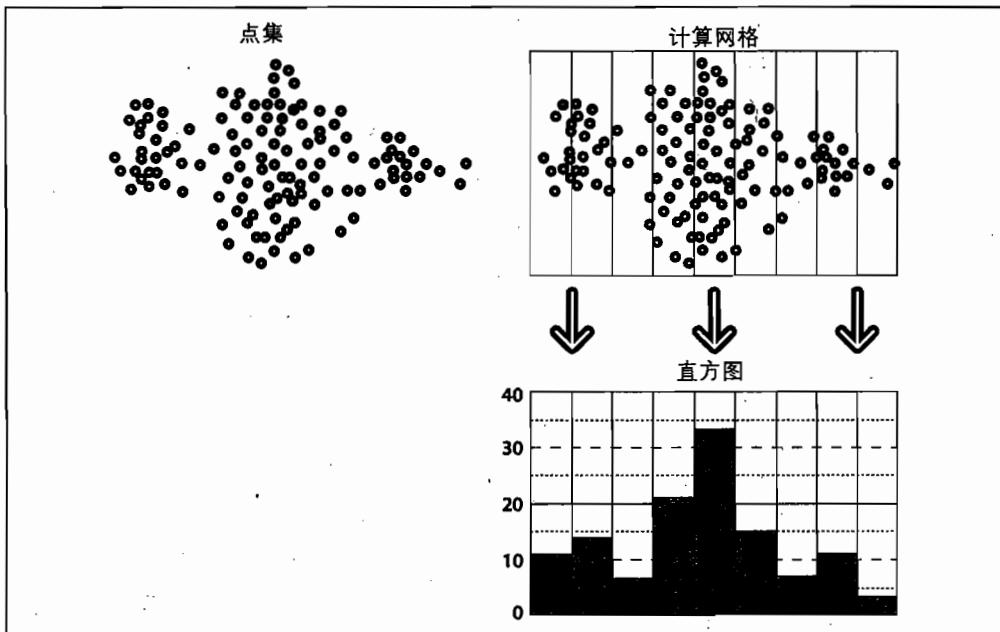


图 7-2：典型的直方图例子：从一个点集开始(左上)，施加计算网格(右上)，产生点集的一维直方图(右下)

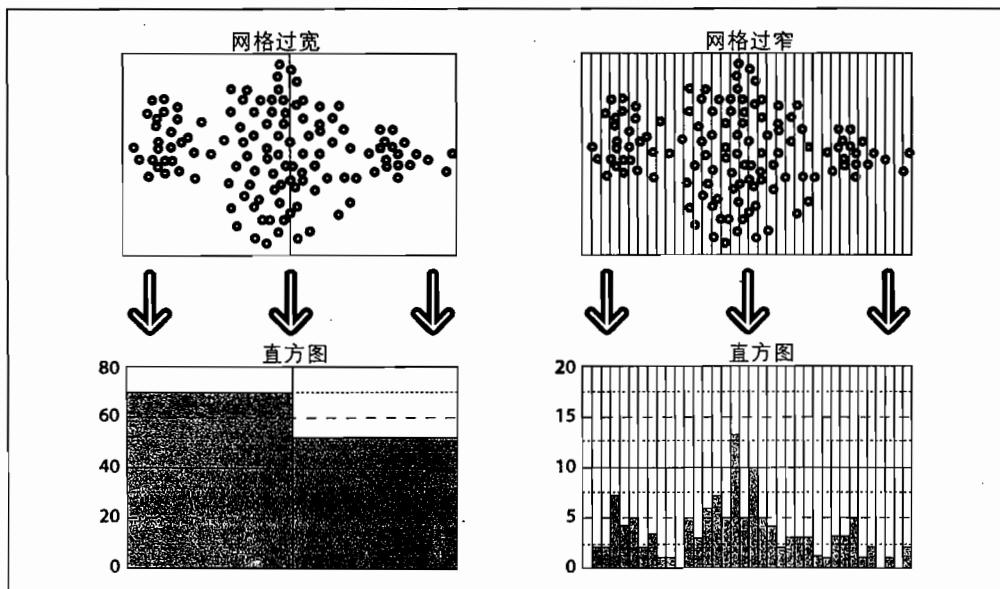


图 7-3：直方图的正确性依赖于网格大小：如果网格太宽，则直方图统计中有太多的空间平均(左)，如果网格太窄，则因太小的平均产生尖锐和单个效果(右)

直方图的基本数据结构

首先直接查看 CvHistogram 的数据结构。

```
typedef struct CvHistogram
{
    int      type;
    CvArr*  bins;
    float   thresh[CV_MAX_DIM][2];  for uniform histograms
    float** thresh2;               for nonuniform histograms
    CvMatND mat;                 embedded matrix header
                                  for array histograms
}
CvHistogram;
```

这个定义看起来很简单但并非如此，因为直方图的很多内部数据都被存储于 CvMatND 结构中。我们用下面的程序创建一个新的直方图：

```
CvHistogram* cvCreateHist(
    int      dims,
    int*    sizes,
    int      type,
    float** ranges = NULL,
    int      uniform = 1
);
```

【195】

变量 dims 表示直方图包含的维数。sizes 变量必须为整数数组，数组长度等于 dims。数组中的每一个整数表示分配给对应维数的 bin 的个数。type 既可以是 CV_HIST_ARRAY，用来表示使用密集多维矩阵结构(如 CvMatND)存储多维直方图，也可以是 CV_HIST_SPARSE^①，当数据以稀疏矩阵(CvSparseMat)方式存储时。变量 ranges 可以是两个形式中的任一种。对均匀直方图来说，ranges 是浮点数对构成的数组^②，数组的个数等于维数。而对非均匀直方图来说，则用包含分割非均匀 bin 的数据所构成的数组来代替均匀直方图所使用的数据对。如果某维数有 N 个 bin，那么这个子数组有 N+1 个元素。每个数组的值均起始于最低 bin 的底边，终结于

① 以前的 CV_HIST_TREE 值也是支持的，但这等同于 CV_HIST_SPARSE。
② 这些数据对是只有两个元素的 C 数组。

最高 bin 的顶边^①。布尔类型变量 uniform 说明直方图是否有均匀的 bin，因此也说明了 rangs 的值该如何解析^②；如果设置为非 0 值，则直方图是均匀的。也可以设置 rangs 为 NULL，这时意味 rangs 是“未知的”（它们也可以在后面使用特殊函数 cvSetHistBinRanges() 来设置）。很明显，最好在使用直方图之前给 rangs 设置数值。

```
void cvSetHistBinRanges(
    CvHistogram*      hist,
    float**           ranges,
    int               uniform = 1
);
```

cvSetHistRanges() 中的变量与 cvCreateHist() 中的相应变量完全一致。如果想重用直方图，可以对其进行清零操作（即设置所有 bins 为 0），或者使用通常的释放函数释放直方图。

```
void cvClearHist(
    CvHistogram* hist
);
void cvReleaseHist(
    CvHistogram** hist
);
```

通常，这些释放函数通过一个指针被调用，该指针指向一个直方图的指针，后者取自创建直方图的函数。一旦直方图被释放，直方图指针便被设置为 NULL。

另外一个有用的函数是根据已给出的数据创建直方图：

```
CvHistogram*      cvMakeHistHeaderForArray(
    int            dims,
    int*          sizes,
    CvHistogram*   hist,
    float*         data,
    float**        ranges = NULL,
```

-
- ① 请明一下，在均匀直方图的情况下，如果 rangs 的上下界分别设置为 0 和 10，且有两个 bin，那么 bin 将分别被指定为区间 [0,5] 和 [5,10]。对非均匀直方图，如果相应维数 *i* 的 size 等于 4，并且相应区间设置为 (0, 2, 4, 9, 10)，那么 bins 的最终结果将被指定为区间：[0,2], [2,4], [4,9] 和 [9,10]。
 - ② 不必担心该变量类型为整数，因为有意义的区分仅仅是 0 和非 0。

```
    int          uniform = 1  
);
```

在这种情况下，hist 是指向 CvHistogram 数据结构的指针，data 是指向存储直方图 bins 的大小为 sizes[0]*sizes[1]*...*sizes[dims-1] 的区域的指针。注意，data 是浮点数指针，因为直方图的内部数据类型描述永远是浮点数。返回值与我们输入的 hist 值一样。与 cvCreateHist() 程序不同，没有 type 变量。所有由 cvMakeHistHeaderForArray() 创建的直方图都是密集直方图。最后一点：由于（假设）为直方图的 bins 分配了数据存储空间，则没有理由为 CvHistogram 结构再次调用 cvReleaseHist()。你只需要清除头结构（如果没有在堆栈中分配它），当然，也要清除你自己创建的数据；但是，由于这些变量是你自己定义的，你应该用自己的方式处理它们。

【196~197】

访问直方图

访问直方图数据的方式有好几种。最直接的是使用 OpenCV 的访问函数：

```
double cvQueryHistValue_1D(  
    CvHistogram* hist,  
    int         idx0  
);  
double cvQueryHistValue_2D(  
    CvHistogram* hist,  
    int         idx0,  
    int         idx1  
);  
double cvQueryHistValue_3D(  
    CvHistogram* hist,  
    int         idx0,  
    int         idx1,  
    int         idx2  
);  
double cvQueryHistValue_nD(  
    CvHistogram* hist,  
    int*        idxN  
);
```

每个函数都返回相应 bin 中的值的浮点数。同样地，可以利用函数返回的 bin 的指针（不是 bin 的值）来设置（或者获得）直方图的 bin 的值。

```

float* cvGetHistValue_1D(
    CvHistogram* hist,
    int         idx0
);
float* cvGetHistValue_2D(
    CvHistogram* hist,
    int         idx0,
    int         idx1
);
float* cvGetHistValue_3D(
    CvHistogram* hist,
    int         idx0,
    int         idx1,
    int         idx2
);
float* cvGetHistValue_nD(
    CvHistogram* hist,
    int*        idxN
);

```

这些函数与 `cvGetReal*D` 和 `cvPtr*D` 等系列函数非常相似，事实上，它们几乎相同。在函数里传入的矩阵 `hist->bins` 被调用的方式本质上与其被直接的矩阵访问模式是一样的。类似地，稀疏直方图函数继承了它所对应的稀疏矩阵函数的一些行为。在稀疏直方图中，如果想利用函数 `GetHist*()` 来访问不存在的 bin，这个不存在的 bin 会被自动创建，并且其值被设为 0。注意，函数 `QueryHist*()` 不会创建不存在的 bin。

【198~199】

这使我们转向访问直方图的更一般论题。在许多情况下，对于密集直方图来说，我们想直接访问直方图中的 bin 成员。当然，我们可以将它视为数据访问的一部分。例如，我们想依次访问密集直方图中所有的数据元素，或者基于效率原因想直接访问 bin，在这种情况下，我们可以利用 `hist->mat.data.fl`(对于密集型直方图来说)来访问。访问直方图的其他原因包括想知道直方图的维数或每个单独的 bin 表示的区域等。对这类信息，我们可以使用下面的小技巧来访问 `CvHistogram` 数据结构中的实际数据，或者访问嵌入在 `CvMatND` 数据结构中的信息来获得。

```

int n_dimension           = histogram->mat.dims;
int dim_i_nbins          = histogram->mat.dim[ i ].size;

// uniform histograms
int dim_i_bin_lower_bound = histogram->thresh[ i ][ 0 ];
int dim_i_bin_upper_bound = histogram->thresh[ i ][ 1 ];

```

```
// nonuniform histograms  
int dim_i_bin_j_lower_bound = histogram->thresh2[ i ][ j ];  
int dim_j_bin_j_upper_bound = histogram->thresh2[ i ][ j+1 ];
```

可以看出，访问直方图数据结构是有多种方法的。

直方图的基本操作

现在，我们有了直方图这个重要的数据结构，自然要用它来做一些有趣的事情。首先来看一些反复用到的基本操作，然后转到为了更特殊的任务而用到的一些更复杂的特性。

当处理直方图的时候，一般会向直方图不同的 bin 中累积信息。完成信息的积累后，通常希望使用有着归一化形式的直方图，这样每个 bin 应该表示分配给整个直方图的所有事件的一部分。

```
cvNormalizeHist( CvHistogram* hist, double factor );
```

这里的 hist 表示直方图，factor 表示直方图归一化后的数值(通常情况下设为 1)。注意，factor 是一个 double 类型的数据，尽管函数 CvHistogram() 的内部数据类型通常都是 float — 这进一步说明了 OpenCV 是一个不断改进的项目！

接下来的一个很方便的函数是阈值函数：

```
cvThreshHist( CvHistogram* hist, double factor );
```

变量 factor 是一个开关阈值。进行直方图阈值化处理之后，小于给定阈值的各个 bin 的值都被设为 0。回忆图像阈值函数 cvThreshold()，直方图阈值函数与参数 threshold_type 设置为 CV_THRESH_TOZERO 的图像阈值函数类似。不幸的是，没有方便的直方图阈值函数来提供其他 threshold 类型的类似操作。然而，实际上函数 cvThreshHist() 或许是最常用的函数，因为我们在使用实数类型数据的时候常常将包含极少数据点的 bin 去除掉。这些 bins 通常是噪声，因此将这些 bins 的值设为 0。
【199】

另一个有用的函数是 cvCopyHist()，表示将一个直方图的信息复制到另一个直方图。

```
void cvCopyHist(const CvHistogram* src, CvHistogram** dst );
```

可以通过两种方式来用这个函数。如果目标直方图 *dst 和源直方图有相同的大

小，那么 `src` 中所有的数据和 `bin` 的范围都被复制给 `*dst`。另一种使用 `cvCopyHist()` 的方式是设置 `*dst` 为 `NULL`。这种情况下，生成一个与 `src` 一样的新直方图，所有数据和 `bin` 的范围都被复制到该直方图中(与图像函数 `cvCloneImage()` 相似)。当第二个参数 `dst` 是一个指向直方图的指针的指针，这种复制是允许的。与 `src` 不同，`src` 仅仅是一个指向直方图的指针。当函数 `cvCopyHist()` 被调用时，如果 `*dst` 为 `NULL`，则当该函数返回时，`*dst` 被分配给一个指向新分配的直方图的指针。

让我们继续了解有用的直方图函数。下一个函数是 `cvGetMinMaxHistValue()`，它输出直方图中找到的最小值和最大值。

```
void cvGetMinMaxHistValue(
    const CvHistogram* hist,
    float*           min_value,
    float*           max_value,
    int*            min_idx = NULL,
    int*            max_idx = NULL
);
```

因此，给定直方图 `hist`，函数 `cvGetMinMaxHistValue()` 将计算其最小值和最大值。当该函数返回时，`*min_value` 和 `*max_value` 将分别被设置为这两个极值。如果不需其中的一个(或两个)值，可以设置相应变量为 `NULL`。接下来的两个参数是可选的；如果使用默认值(`NULL`)，则不做任何事情。如果为非 `NULL` 的指针，则函数返回最小和最大值的索引值。对于多维直方图，参数 `min_idx` 和 `max_idx`(如果非 `NULL`)被设为指向一个整数数组的指针，数组长度等于直方图的维数。如果一个直方图中有许多个 `bin` 的值都为最小值(或最大值)，则返回具有最小索引(对多维直方图则按字母顺序)的 `bin`。

在收集直方图数据后，通常利用函数 `cvGetMinMaxHistValue()` 来寻求最小值，同时利用函数 `cvThreshHist()` 在最小值附近进行阈值化操作，最终通过函数 `cvNormalizeHist()` 来归一化直方图。

最后是从图像中自动计算直方图，函数 `cvCalcHist()` 完成了这一关键任务：

```
void cvCalcHist(
    IplImage**      image,
    CvHistogram*    hist,
    int             accumulate = 0,
    const CvArr*    mask      = NULL
);
```

【200~201】

第一个参数 `image` 是一个指向数组的 `IplImage*` 类型指针。^① 这允许利用多个图像通道。对于多通道图像(如 HSV 或 RGB)，在调用函数 `cvCalcHist()` 之前，先要用函数 `cvSplit()`(见第 3 章或第 5 章)将图像分为单通道的。这诚然有点痛苦，但是应该考虑到多个图像通道很常见，它们常常包含一个图像的不同过滤形态，如梯度通道、*YUV* 中的 *U* 或 *V* 通道。可以想像，如果试图处理包含多通道的多个图像，情形将有多么混乱(需要确定谁在哪儿用这些图像的哪个通道)。为了避免这种混淆，所有传递给 `cvCalcHist()` 的图像都被假设(或被要求)为单通道图像。得到直方图后，`bin` 的值由通过多幅图像确定的多元组来确定。参数 `hist` 是一个适当维数的直方图(比如，维数等于输入图像的通道数)。后两个参数时可选的。参数 `accumulate` 如果非 0，表示直方图 `hist` 在读入图像之前没有被清零，注意，变量 `accumulation` 允许 `cvCalcHist()` 在一个数据采集循环中被多次调用。最后一个参数 `mask` 是一个可选的布尔变量；如果被设为非 `NULL`，则只有与 `mask` 非 0 元素对应的像素点会被包含在计算直方图中。

对比两个直方图

然而，对于直方图来说，另一个不可或缺的工具是用某些具体的标准来比较两个直方图的相似度。这些工具首先由 Swain 和 Ballard[Swain91]引入，后经 Schiele 和 Crowley[Schiele96]推广。函数 `cvCompareHist()` 用于对比两个直方图的相似度。

```
double cvCompareHist(
    const CvHistogram* hist1,
    const CvHistogram* hist2,
    int method
);
```

前两个参数是要比较的大小相同的直方图，第三个变量是所选择的距离标准，有下面四种可用选项。

相关(`method= CV_COMP_CORREL`)

$$d_{\text{correl}}(H_1, H_2) = \frac{\sum_i H'_1(i) \cdot H'_2(i)}{\sqrt{\sum_i H'^2_1(i) \cdot \sum_i H'^2_2(i)}}$$

其中 $H'_k(i) = H_k(i) - (1/N) \left(\sum_j H_k(j) \right)$ 且 N 等于直方图中 bin 的个数。

【201】

① 实际上，可以利用 `CvMat*` 矩阵指针。

对于相关(correlation)，数值越大则越匹配。完全匹配的数值为 1，完全不匹配是 -1，值为 0 则表示无关联(随机组合)。

卡方(method = CV_COMP_CHISQR)

$$d_{\text{chi-square}}(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)}$$

对于 chi-square^①，低分比高分匹配的匹配程度高。完全匹配的值为 0，完全不匹配为无限值(依赖于直方图的大小)。

直方图相交(method=CV_COMP_INTERSECT)

$$d_{\text{intersection}}(H_1, H_2) = \sum_i \min(H_1(i), H_2(i))$$

对于直方图相交，高分表示好匹配，而低分则表示坏匹配。如果两个直方图都被归一化到 1，则完全匹配是 1，完全不匹配是 0。

Bhattacharyya 距离(method = CV_COMP_BHATTACHARYYA)

$$d_{\text{Bhattacharyya}}(H_1, H_2) = \sqrt{1 - \sum_i \frac{\sqrt{H_1(i) \cdot H_2(i)}}{\sum_i H_1(i) \cdot \sum_i H_2(i)}}$$

对于 Bhattacharyya 匹配[Bhattacharyya43]，低分数表示好匹配，而高分数表示坏的匹配。完全匹配是 0，完全不匹配是 1。

对于 CV_COMP_BHATTACHARYYA，公式中的一个特定比例用来对输入的直方图进行归一化操作，然而一般情况下，在对比直方图之前，都应该自行进行归一化操作，因为如果不归一化，像直方图相交等概念就没有任何意义(即使允许)。

图 7-4 所描述的简单情况应该能够说明问题。事实上，这或许是我们所能想像到的最简单的情况：只有两个 bin 的一维直方图。模板直方图左边 bin 的值为 1.0，而右边 bin 的值为 0.0。最后三列给出了用来对比的直方图以及对它们使用不同的度量后得到的值(EMD 度量将马上介绍)。

图 7-4 给出了不同匹配方式的快速参考，但是这有点令人不安。如果直方图的 bin 仅仅偏移一个位置——就像图中第一个和第三个直方图那样，那么所有这些匹配方法(除了 EMD)都会产生完全不匹配的结果，即使这两个直方图有相似的形状。

① chi-square 分布是 Karl Pearson[Pearson]发明的，他是数理统计领域的奠基人。

图 7-4 最右边一列说明 EMD 的返回值，EMD 是一种距离的度量方式。对比第三个和模板直方图，EMD 方法准确量化了这样的情况：第三个直方图向右移动了一个单位。下将继续探讨这一主题。

直方图 模型	匹配衡量				
	相关	卡方	相交	Bhattacharyya:	EMD:
精确匹配 	1.0	0.0	1.0	0.0	0.0
半匹配 	0.7	0.67	0.5	0.55	0.5
完全不匹配 	-1.0	2.0	0.0	1.0	1.0

图 7-4: 直方图匹配度量

以作者的经验来说，在快速但不怎么精确匹配的情况下，intersection 方法的效果好。而在慢速但较精确的情况下，用 chi-square 或 Bhattacharyya 方法的效果好。EMD 方法给出最直观的匹配，但更慢一些。

直方图用法示例

现在是提供一些有用例子的时候了。程序如例 7-1 所示(改写自 OpenCV 代码)，该例给出了我们怎样利用刚刚讨论过的一些函数。该程序根据输入的图像计算出一个色相饱和度(hue-saturation)直方图，然后利用网格的方式将该直方图以网格形式显示。

例 7-1: 直方图的计算与显示

```
#include <cv.h>
#include <highgui.h>

int main( int argc, char** argv ) {
    IplImage* src;
```

```

if( argc == 2 && (src=cvLoadImage(argv[1], 1))!= 0) {

    // Compute the HSV image and decompose it into separate planes.
    IplImage* hsv = cvCreateImage( cvGetSize(src), 8, 3 );
    cvCvtColor( src, hsv, CV_BGR2HSV );

    IplImage* h_plane = cvCreateImage( cvGetSize(src), 8, 1 );
    IplImage* s_plane = cvCreateImage( cvGetSize(src), 8, 1 );
    IplImage* v_plane = cvCreateImage( cvGetSize(src), 8, 1 );
    IplImage* planes[] = { h_plane, s_plane };
    cvCvtPixToPlane( hsv, h_plane, s_plane, v_plane, 0 );

    // Build the histogram and compute its contents.
    int h_bins = 30, s_bins = 32;
    CvHistogram* hist;
    {
        int hist_size[] = { h_bins, s_bins };
        float h_ranges[] = { 0, 180 };           // hue is [0,180]
        float s_ranges[] = { 0, 255 };
        float* ranges[] = { h_ranges, s_ranges };
        hist = cvCreateHist(
            2,
            hist_size,
            CV_HIST_ARRAY,
            ranges,
            1
        );
    }
    cvCalcHist( planes, hist, 0, 0 ); //Compute histogram
    cvNormalizeHist( hist[i], 1.0 ); //Normalize it
    // Create an image to use to visualize our histogram.
    int scale = 10;
    IplImage* hist_img = cvCreateImage(
        cvSize( h_bins * scale, s_bins * scale ),
        8,
        3
    );
    cvZero( hist_img );

    // populate our visualization with little gray squares.
    float max_value = 0;
    cvGetMinMaxHistValue( hist, 0, &max_value, 0, 0 );
}

```

```

        for( int h = 0; h < h_bins; h++ ) {
            for( int s = 0; s < s_bins; s++ ) {
                float bin_val = cvQueryHistValue_2D( hist, h, s );
                int intensity = cvRound( bin_val * 255 / max_value );
                cvRectangle(
                    hist_img,
                    cvPoint( h*scale, s*scale ),
                    cvPoint( (h+1)*scale - 1, (s+1)*scale - 1 ),
                    CV_RGB(intensity,intensity,intensity),
                    CV_FILLED
                );
            }
        }

        cvNamedWindow( "Source", 1 );
        cvShowImage( "Source", src );

        cvNamedWindow( "H-S Histogram", 1 );
        cvShowImage( "H-S Histogram", hist_img );

        cvWaitKey(0);
    }
}

```

【203~205】

在这个例子中，我们花费了相当一部分时间来准备函数 `cvCalcHist()` 的参数，而这通常是必须的。我们还选择了对可视化显示中的颜色进行归一化处理，而不是归一化直方图本身，尽管对于某些应用而言，归一化直方图比归一化颜色值会好一些。这样一来，我们就有理由调用函数 `cvGetMinMaxHistValue()`，而如果归一化直方图本身，我们就没有理由这样做。

让我们看一些更实际的例子：在不同光照条件下人手的颜色直方图。图 7-5 的左列给出了室内环境下的人手、室外阴影环境下的手以及室外阳光照射环境下的人手。中间一列给出了手的肤色对应的蓝、绿、红(BGR)直方图。右列给出了对应的 HSV 直方图，其中 V(亮度值，value)是垂直轴，S(饱和度，saturation)是半径，H(色调，hue)是角度。注意，室内环境下的最黑，室外阴影环境下的稍微亮点儿，而室外阳光照射环境下的最亮。注意，颜色的变化在某种程度上是光学的色彩变化造成的。

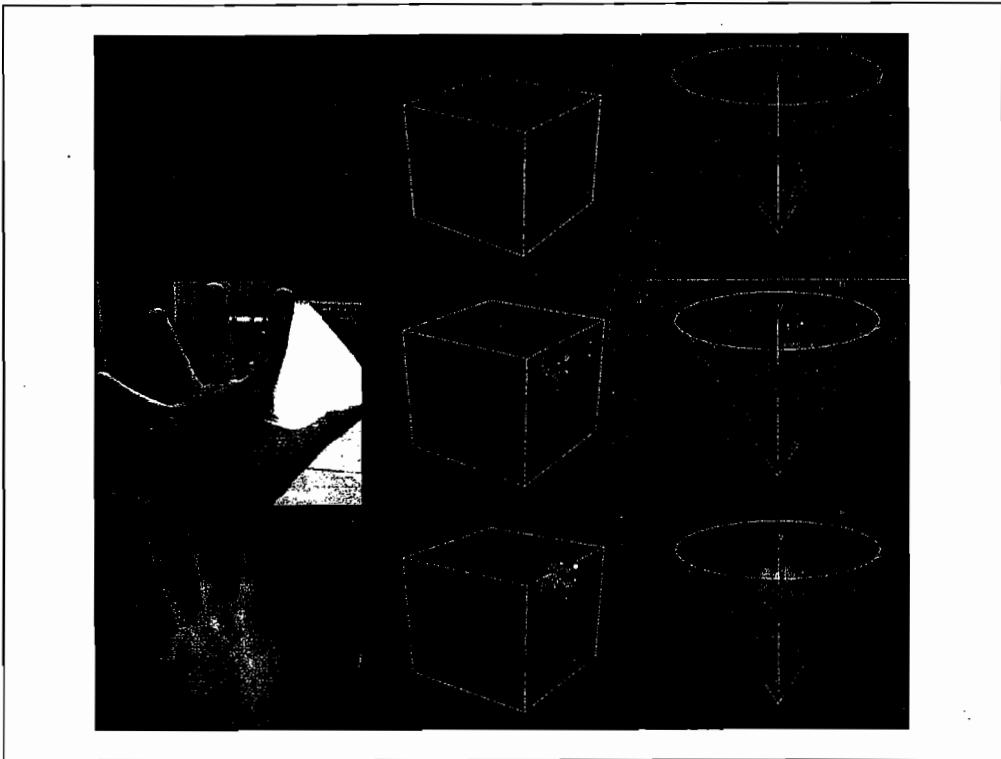


图 7-5：室内环境(左上)、室外阴影环境(左中)、室外阳光照射环境(左下)中的肤色直方图；中间一列和右列分别给出相关的 BGR 和 HSV 直方图

作为直方图比较的测试，我们利用手掌的一部分(即室内环境下手掌的上半部分)，分别将该图像中色彩的直方图与该图像区域部分的直方图以及其他两幅手部图像中直方图进行对比。新鲜肤色通常比较容易在 HSV 色彩空间中提取。这提示对我们色调(hue)和饱和度(saturation)通道中的限制不仅足够，而且有助于识别人的新鲜肤色。表 7-1 显示出我们实验的匹配结果，这证实了光线会在色彩中导致严重的误匹配。有时，归一化的 BGR 在光线变化时会比 HSV 工作得更好。 【205】

表 7-1：通过四种匹配方法，室内手掌上半部肤色与列出的手掌肤色的直方图对比

比较	相关	卡方	相交	BHATTACHARYYA
室内下半部分	0.96	0.14	0.82	0.2
室外阴影下	0.09	1.57	0.13	0.8
室外阳光下	-0.0	1.98	0.01	0.99

一些更复杂的策略

迄今讨论的所有东西都是相当基本的。每一个函数都有其相对明显的需要。总的说来，这些函数为计算机视觉(或许还有领域)应用直方图提供了一个良好基础。从这点上讲，我们需要看一些更复杂的 OpenCV 程序，这些程序在特定的应用下非常有用。它们还包括一个更复杂的直方图比较方法和工具，该工具可以计算/或显示图像的哪个部分与直方图中的给定部分相关。

陆地移动距离

光线变化能引起图像颜色值的漂移(图 7-5)，尽管这些漂移没有改变颜色直方图的形状，但是这些漂移引起了颜色值位置的变化，从而导致前述匹配策略失效。如果利用直方图的距离测量来代替直方图的匹配策略，那么我们仍然可以像直方图对比一样对比两个直方图的距离，即使第二个直方图发生漂移，也能找到最小的距离度量。陆地移动距离(Earth Mover's Distance, EMD)[Rubner00]是一种度量准则；它实际上度量的是怎样将一个直方图的形状转变为另一个直方图的形状，包括移动直方图的部分(或全部)到一个新的位置，可以在任何维的直方图上进行这种度量。

再看看图 7-4。我们可以在最右边一列看到 EMD 距离度量的特性。0 表示最精确的匹配。半匹配是成功地将直方图的一半转换，将左边直方图的一半转换到下一个直方图。最终，移动整个直方图到右边需要整个单位的距离(即将模板直方图转换为完全不匹配直方图)。

EMD 算法本身是个通用算法，它允许用户自己设置距离度量或者自己的移动代价矩阵。用户可以记录直方图“素材”从一个直方图什么位置滚动到另一个直方图后，也可以利用此前数据信息所产生的非线性距离度量。OpenCV 中的 EMD 函数是 `cvCalcEMD2()`，如下所示。

```
float cvCalcEMD2(
    const CvArr*      signature1,
    const CvArr*      signature2,
    int               distance_type,
    CvDistanceFunction distance_func   = NULL,
    const CvArr*      cost_matrix    = NULL,
    CvArr*            flow           = NULL,
    float*            lower_bound    = NULL,
```

```
    void*           userdata      = NULL  
);
```

函数 CvCalcEMD2() 有许多令人头疼的参数，所以它直观上看起来是一个很复杂的函数，但其复杂性来源于该算法的精确的配置选项^①。幸运的是，这个函数可以有更基本、更直观的形式，不必有这么多参数(注意：所有的“=NULL”默认值)。例 7-2 给出了其简单的形式。

【207】

例 7-2：简单的 EMD 接口

```
float cvCalcEMD2(  
    const CvArr* signature1,  
    const CvArr* signature2,  
    int          distance_type  
);
```

cvCalcEMD2() 的简单形式可以由参数 distance_type 确定，可以是 Manhattan 距离(CV_DIST_L1)，也可以是 Euclidean 距离(CV_DIST_2)。尽管只是在直方图中应用 EMD，但接口使用名称为 signatures 的两个数组参数更方便。

Signature 数组通常是浮点型的，每行包括直方图的 bin 及其坐标，对图 7-4 的一维的直方图来说，直方图左边一列(忽略模型)的 signatures(列于数组的每一行)定义如下：上面，[1,0;0,1]；中间，[0.5,0;0.5,1]；底部，[0,0;1,1]。如果三维直方图中的一个 bin 在(x,y,z)索引为(7,43,11)处的值为 537，那么这个 bin 中的 signature 行应该是[537,7, 43,11]。这就是我们将直方图转换为 signature 的方式。

例如，假设我们现在有两个直方图，hist1 和 hist2，我们想要将其转化为两个 signature，sig1 和 sig2。如果只是想使情况更难一点，可假设有一个 h_bins × s_bins 的二维直方图(如前面的代码示例)。例 7-3 演示了怎样将两个直方图转换为两个 signature。

【208】

例 7-3：从 EMD 直方图创建 signature

```
//Convert histograms into signatures for EMD matching  
//assume we already have 2D histograms hist1 and hist2  
//that are both of dimension h_bins by s_bins (though for EMD,  
// histograms don't have to match in size).
```

① 如果想深入了解所有细节，建议读 S.Pelg, M.Werman 和 H.Rom 在 1989 年发表的论文“*A Unified Approach to the Change of Resolution:Space and Gray-Level*”，再阅读 OpenCV 用户手册中的相关内容，见...\\opencv\\docs\\ref\\opencvref_cv.htm.

```

//  

CvMat* sig1,sig2;  

int numrows = h_bins*s_bins;  

//Create matrices to store the signature in  

//  

sig1 = cvCreateMat(numrows, 3, CV_32FC1); //1 count + 2 coords = 3  

sig2 = cvCreateMat(numrows, 3, CV_32FC1); //sigs are of type float.  

//Fill signatures for the two histograms  

//  

for( int h = 0; h < h_bins; h++ ) {  

    for( int s = 0; s < s_bins; s++ ) {  

        float bin_val = cvQueryHistValue_2D( hist1, h, s );  

        cvSet2D(sig1,h*s_bins + s,0,cvScalar(bin_val)); //bin value  

        cvSet2D(sig1,h*s_bins + s,1,cvScalar(h)); //Coord 1  

        cvSet2D(sig1,h*s_bins + s,2,cvScalar(s)); //Coord 2  

        bin_val = cvQueryHistValue_2D( hist2, h, s );  

        cvSet2D(sig2,h*s_bins + s,0,cvScalar(bin_val)); //bin value  

        cvSet2D(sig2,h*s_bins + s,1,cvScalar(h)); //Coord 1  

        cvSet2D(sig2,h*s_bins + s,2,cvScalar(s)); //Coord 2  

    }  

}

```

【208~209】

注意，在这个例子中^①，函数 `cvSet2D()` 利用 `CvScalar()` 设置数组中的数值，即使该矩阵中的每一个元素都是单一的浮点型数据。我们利用内联宏 `cvScalar()` 来完成这个任务。一旦将直方图转换为 `signature`，我们就可以得到这个距离测量。选择 Euclidean 距离，现在增加例 7-4 中的这部分代码。

例 7-4：利用 EMD 来度量两个分布之间的相似性

```

// Do EMD AND REPORT
//  

float emd = cvCalcEMD2(sig1,sig2,CV_DIST_L2);
printf("%f", emd);

```

① 利用函数 `cvSetReal2D()` 或函数 `cvmSet()` 也许更紧凑、更有效，但是这样的例子更加清楚。与 EMD 中实际的距离计算相比，这样的额外开销非常小。

反向投影

反向投影(back projection)是一种记录像素点(为 `cvCalcBackProject()`)或像素块(为 `cvCalcBackProjectPatch()`)如何适应直方图模型中分布的方式。例如，我们有一个颜色直方图，可以利用反向投影在图像中找到该区域。该函数调用需要执行如下查找：

```
void cvCalcBackProject(
    IplImage**           image,
    CvArr*                back_project,
    const CvHistogram*   hist
);
```

我们已经在函数 `cvCalcHist()` 见到过单通道的图像数组 `IplImage** image`(见 7.3 节)。数组中的图像数与用来创建直方图模型 `hist` 中的图像数完全相同——且具有相同的顺序。例 7-1 演示了怎样将一幅图像转换为单一通道然后再建立单通道图像数组。`image` 或数组 `back_project` 与数组的输入图像大小一样，而且也是单通道，8 位或浮点图像。`back_project` 的值设置为 `hist` 中的相关 bin 的值。如果直方图是归一化的，此值便与一个条件概率值相关(即图像中像素点为直方图 `hist` 所表征的某种成员的概率)^①。在图 7-6 中，用肤色直方图来推导肤色图像的概率。

注意：当 `back_project` 是一个 8 位图像而不是一个浮点数图像时，不必归一化该直方图或重新定义其尺寸。因为，标准化后的直方图的最大值为 1，在 8 位图像中，任何小于该值的数都会被四舍五入至 0。如果想看到该值，还需重新映射 `back_project` 的值，这取决于直方图中的最大数值。

- ① 具体说来，在肤色色调的 H-S 直方图中，如果 C 为像素点的颜色值，F 是像素点为肤色的概率值，那么概率映射给出 $p(C|F)$ ，表示绘制肤色的概率(如果像素点为肤色)。它和给定颜色时像素点是肤色的概率 $p(F|C)$ 不一样。但根据贝叶斯理论 [Bayes1763]，这两个概率是有关系的。所以，如果我们知道场景中有一个肤色对象的总概率和肤色范围内所有物体的总概率，便从 $p(C|F)$ 中计算 $p(F|C)$ 。具体说来，贝叶斯理论建立如下关系：

$$p(F|C) = \frac{p(F)}{p(C)} p(C|F)$$

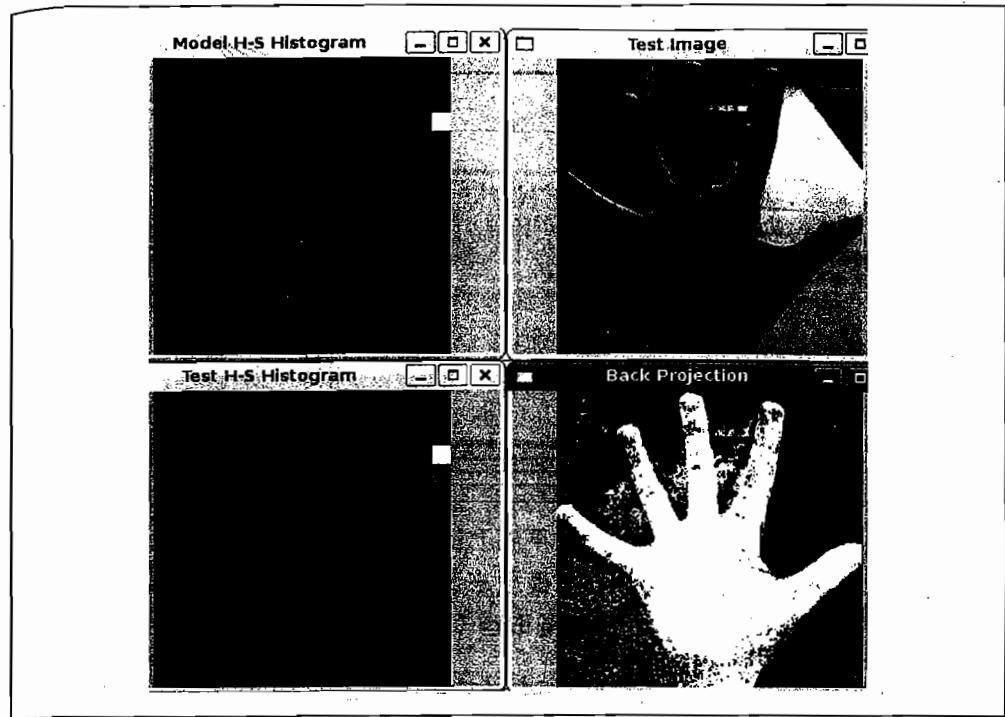


图 7-6：在每个依赖于颜色的像素点上的直方图反向投影：HSV 肤色直方图（左上）；被用来转换的手的图像（右上）到肤色概率图像（右下）；左下是手的颜色直方图

【209~210】

基于块的反向投影

我们可以利用基本的 `back_project` 方法来为一个特定像素是否可能是一个特定目标类型的成员(以一个直方图作为该目标类型的模型)建模。这与某个特定目标是否存在的概率计算不完全相同。另一种方法是考虑图像子区域以及子区域的特征(比如颜色)直方图，并且想知道子区域特征的直方图是否与模型的直方图匹配。然后，我们把每一个这样的子区域与目标是否位于该子区域的概率联系起来。

因此，正如我们可以用函数 `cvCalcBackProject()` 计算一个像素是否是一个已知目标的一部分，也可以用函数 `cvCalcBackProjectPatch()` 计算一块区域是否包含已知的目标。函数 `cvCalcBackProjectPatch()` 在整个输入图像使用一个滑动窗口，如图 7-7 所示。在输入图像矩阵的每一个位置，块中所有的像素点都被设置为在目标图像中对应的块中心位置的像素点。这一点非常重要，因为图像的许多特性(如纹理)在单一的像素级别上无法确定，但可从一组像素确定。

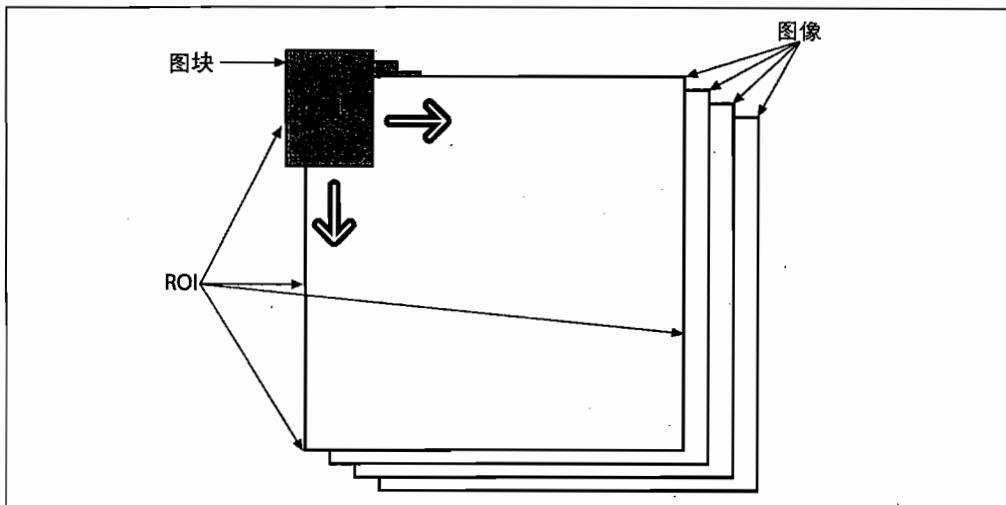


图 7-7：反向投影。使用在输入图像平面上的滑动块来设置目标图像上相应的像素(块的中心)；对于归一化直方图模型来说，结果图像可以被解释为一个概率图，它表示目标是否可能出现(此图取自 OpenCV 参考手册)

在这些例子中，为简单起见，我们对颜色进行采样以建立直方图模型，所以图 7-6 给出了整个手，因为手的像素能很好地与肤色直方图模型匹配。利用块，我们可以检测局部区域的统计特性，如局部亮度的变化弥补整个目标纹理最多的特性。使用局部块的时候，函数 `cvCalcBackProjectPatch()` 有两种用法：当采样窗口小于目标时，作为一个区域检测器，当采样窗口和目标一般大时，作为目标检测器。图 7-8 显示如何利用函数 `cvCalcBackProjectPatch()` 作为区域检测器。我们从手掌肤色的直方图模型开始，在整个图像上移动小窗口，使得反向投影图像上的每个像素点，在给定原始图像上窗口周围的所有像素点时，都表示该像素点的肤色概率。图 7-8 中手的尺寸远大于扫描窗口的尺寸，所以手掌区域被优先检测出来。图 7-9 从一个蓝色杯子的直方图模型开始。与图 7-8 的检测区域相反，图 7-9 展示函数 `cvCalcBackProjectPatch()` 可以被用作目标检测器。当窗口的大小和目标的大小大致相同时，我们希望在反向投影图像中找到整个目标。在反向投影图像中，找到该峰值对应的位置(图 7-9 中，杯子)，即为我们要找到目标的位置。

【211】

OpenCV 提供的图块的反向投影函数如下：

```
void cvCalcBackProjectPatch(
    IplImage**      images,
    CvArr*          dst,
    CvSize           patch_size,
    CvHistogram*     hist,
```

```
    int          method,
    float        factor
);
```

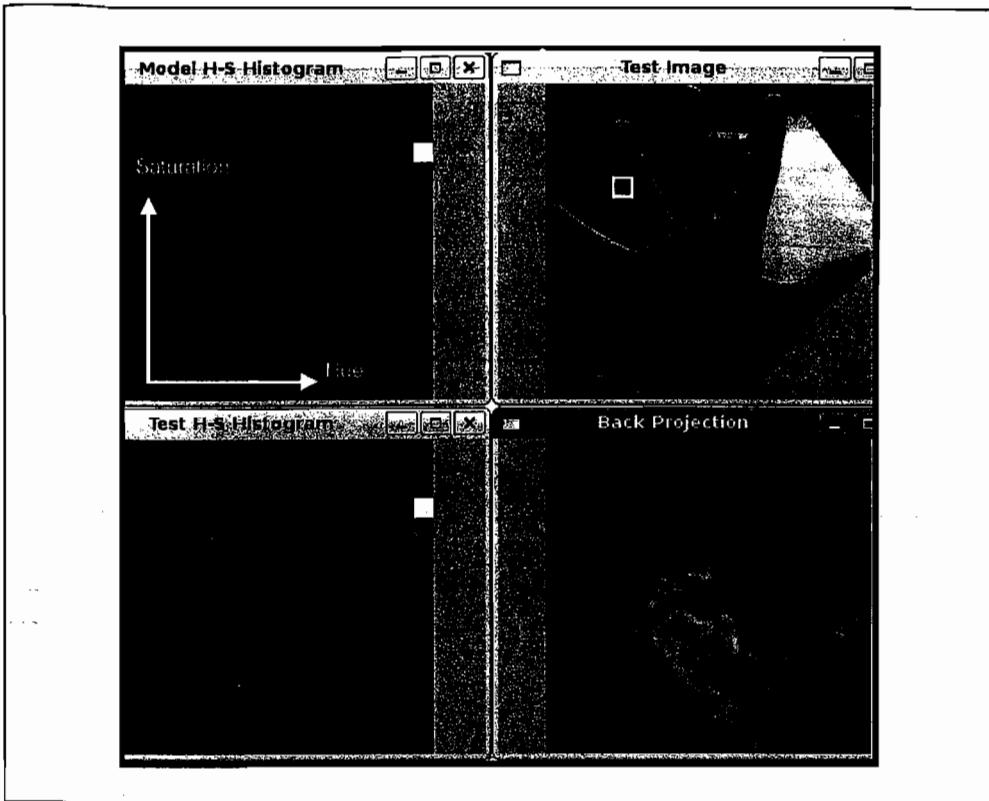


图 7-8：当窗口(在右上图的白色小框)比手要小时，利用肤色的直方图目标模型得到的反向投影；这里直方图模型是一个手的颜色分布，峰值的位置对应的是手的中心

这里，同一个单通道图像数组被 `cvCalcHist()` 用来创建直方图。但目标图像 `dst` 不一样，它仅仅是一个单通道浮点型图像，其大小为 `(images[0][0].width-patch_size.x+1, images[0][0].height -patch_size.y+1)`。该尺寸的解释(见图 7-7)是块的中心像素设置为 `dst` 中相对应的位置，所以在图像的每一侧，都丢失了半个块维数大小的数据。参数 `patch_size` 等同于所期望的块的大小，可以利用宏 `cvSize(width,height)` 来设定。我们已经熟悉直方图的参数了；如同函数 `cvCalcBackProject()` 一样，这个是要比较每个窗口的直方图模型。比较的参数

与函数 `cvCompareHist()` 中的方法类型参数一样(见 7.3.1 节)^①。最后的参数 `factor`, 是归一化水平, 这个参数与前面讲到的函数 `cvNormalizeHist()` 中的参数一致, 可以设置为 1 或可视化的更大值。由于其适应性, 通常在调用函数 `cvCalcBackProjectPatch()` 之前可以自由选择是否归一化立方图模型。

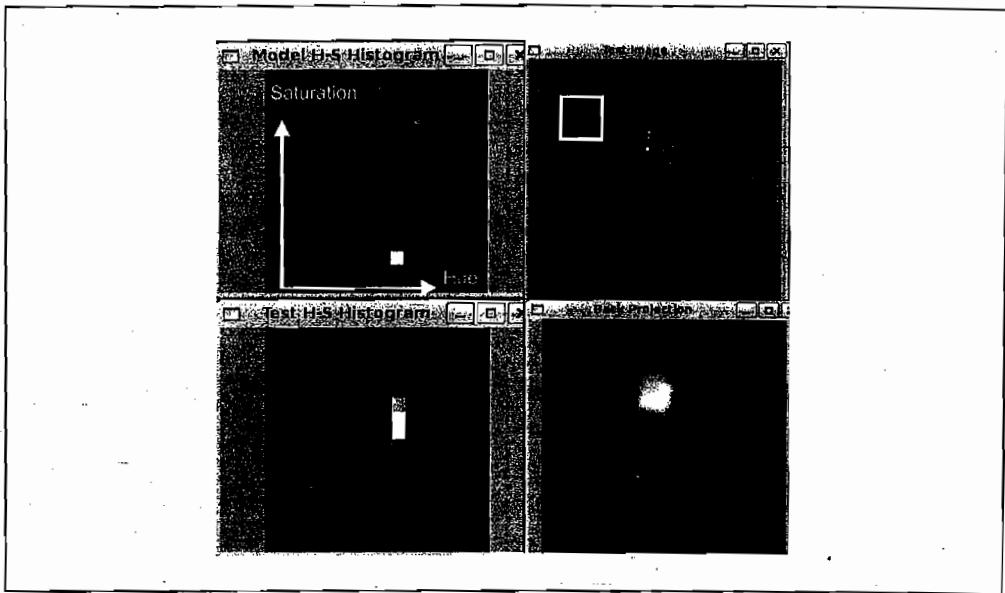


图 7-9: 利用函数 `cvCalcBackProjectPatch()` 定位目标(这里是一个咖啡杯), 其大小近似等于块大小(上图的白色矩形区域)。用色调-饱和度直方图(左上)来为寻找的建模, 然后与整幅图像的 HS 直方图比较(左下); `cvCalcBackProjectPatch()` 得到的结果(右下)就是凭借其颜色即可轻易从图像中找出的目标

最后的问题出现了: 一旦我们得到了目标图像的概率值, 又应怎样利用该图像来找到我们要找到目标呢? 我们可以利用第 3 章讲到的函数 `cvMinLoc()` 来寻找, 最大值的位置(第一步先平滑)最有可能是目标在图像中的位置。这使我们转到有点离题的模板匹配。
【212~214】

模板匹配

通过 `cvMatchTemplate()` 做模板匹配不是基于直方图的; 事实上, 使用本节介绍

① 选择方法时要慎重, 因为有的返回值为 1 表示最好的匹配, 而有的则是返回值 0 表示最好的匹配。

的匹配方法之一，通过在输入图像上滑动图像块对实际的图像块和输入图像进行匹配。

如图 7-10 所示，我们有一个包含人脸的图像块，那么我们在整幅输入图像上移动该脸来寻找表示可能有另一张脸存在的最优匹配。该函数的调用方法与 `cvCalcBackProjectPatch()` 很相似。

```
void cvMatchTemplate(
    const CvArr*      image,
    const CvArr*      templ,
    CvArr*            result,
    int               method
);
```

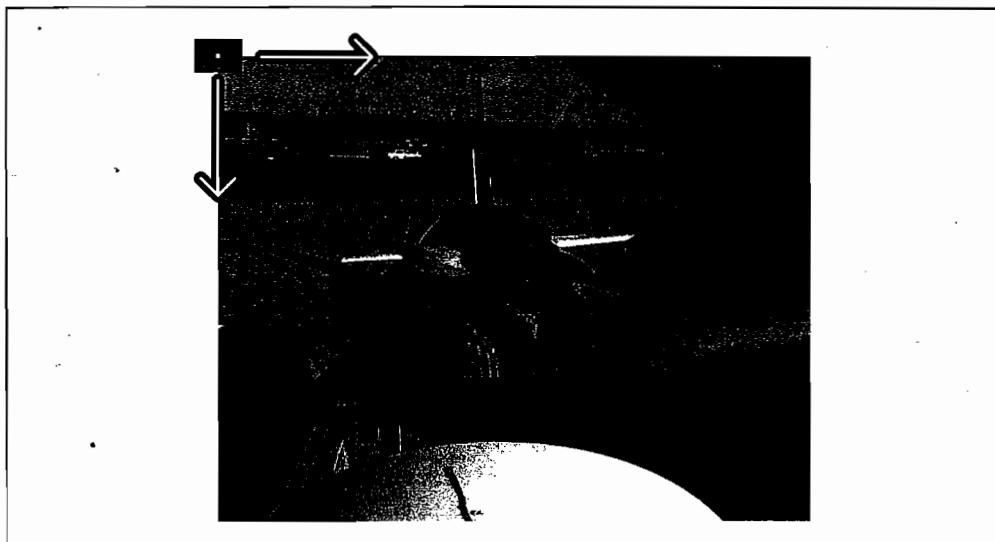


图 7-10：函数 `cvMatchTemplate()` 在另一幅图像上移动模板图像块来寻找匹配

与在 `cvCalcBackProjectPatch()` 中看到的输入图像数组不同，这里使用一个 8 位或浮点型单通道图像或彩色图像作为输入。Temp1 型中的匹配模型就是来自于包含欲寻找的目标的相似图像块。输出目标图像将被放入 `result` 图像中，它是单通道 8 位或浮点型图像(图像大小为 `images->width-patch_size.x+1, images->height - patch_size.y + 1`)，如同我们前面在 `cvCalcBackProjectPatch()` 中看到的那样。匹配的方法稍微有些复杂，将要在下文解释，这里用 *I* 来表示输入图像，*T* 为模板，*R* 为结果。

平方差匹配法(method = CV_TM_SQDIFF)

这类方法利用平方差来进行匹配，最好匹配为 0。匹配越差，匹配值越大。

$$R_{\text{sq_diff}}(x, y) = \sum_{x', y'} [(x', y') - I(x+x', y+y')]^2$$

相关匹配法(method = CV_TM_CCORR)

这类方法采用模板和图像之间的乘法操作，所以较大的数表示匹配程度较高，0 表示最坏的匹配效果。

$$R_{\text{corr}}(x, y) = \sum_{x', y'} [(x', y') \cdot I(x+x', y+y')]^2$$

相关匹配法(method = CV_TM_CCOEFF)

这类方法将模板对其均值的相对值与图像对其均值的相对值进行匹配，1 表示完美的匹配，-1 表示最糟糕的匹配，0 表示没有任何相关性(随机序列)。

$$R_{\text{coeff}}(x, y) = \sum_{x', y'} [(x', y') \cdot I(x+x', y+y')]^2$$

$$T(x', y') = T(x', y') - \frac{1}{(w \cdot h) \sum_{x'', y''} T(x'', y'')}$$

$$I'(x+x', y+y') = I(x+x', y+y') - \frac{1}{(w \cdot h) \sum_{x'', y''} I(x+x'', y+y'')}$$

归一化方法

对于刚才描述的三种方法中的任一个方法来说，都有其归一化的形式，根据 Rodgers[Rodgers88]的描述，它们最早由 Galton[Galton]提出。归一化的方法很有用，因为，像前面提到的那样，它们可以帮助我们减少模板和图像上光线变化所产生影响。在每一种情况下，归一化的系数都如下：

$$Z(x, y) = \sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x+x', y+y')^2}$$

表 7-2 给出了标准模板匹配的各种方法的参数值。

表 7-2：标准模板匹配的各种方法的参数值

方法的参数值	计算结果
CV_TM_SQDIFF_NORMED	$R_{sq_diff_normed}(x, y) = \frac{R_{sq_diff}(x, y)}{Z(x, y)}$
CV_TM_CCORR_NORMED	$R_{cor_normed}(x, y) = \frac{R_{cor}(x, y)}{Z(x, y)}$
CV_TM_CCOEFF_NORMED	$R_{coeff_normed}(x, y) = \frac{R_{coeff}(x, y)}{Z(x, y)}$

通常，随着从简单的测量(平方差)到更复杂的测量(相关系数)，我们可获得越来越准确的匹配(同时也意味着越来越大的计算代价)。最好的办法是对所有这些设置多做一些测试实验，以便为自己的应用选择同时兼顾速度和精度的最佳方案。

【216】

注意：再次重申，解释结果时要谨慎。平方差的方法中，最小值表示最好匹配，而相关和相关系数的方法，则是最大值表示最好匹配。

至于函数 `cvCalcBackProjectPatch()`，一旦我们利用函数 `cvMatchTemplate()` 获得一个匹配的结果图像，便可以利用函数 `cvMinMaxLoc()` 找到最匹配的位置。同样，我们希望保证在某个点附近有一些好的匹配区域，以避免随机模板恰好完整匹配。一个好的匹配位置附近应该有许多好的匹配位置，因为模板的轻度变化不应该有不同的匹配位置。在寻找最小(对于平方差度量来说)或最大值(对于互相关或互相关系数来说)之前要对结果图像进行平滑操作。

可通过例 7-5 看出不同模板匹配技术的行为，此程序首先读入一个模板和要匹配的图像，然后通过前面讨论的方法进行匹配。

例 7-5：模板匹配

```
// Template matching.  
// Usage: matchTemplate image template  
//  
#include <cv.h>  
#include <cxcvcore.h>  
#include <highgui.h>  
#include <stdio.h>  
int main( int argc, char** argv ) {  
    IplImage *src, *templ,*ftmp[6]; //ftmp will hold results  
    int i;
```

```

if( argc == 3){
    //Read in the source image to be searched:
    if((src=cvLoadImage(argv[1], 1))== 0) {
        printf("Error on reading src image %s\n",argv[i]);
        return(-1);
    }
    //Read in the template to be used for matching:
    if((templ=cvLoadImage(argv[2], 1))== 0) {
        printf("Error on reading template %s\n",argv[2]);
        return(-1);
    }
    //ALLOCATE OUTPUT IMAGES:
    int iwidth = src->width - templ->width + 1;
    int iheight = src->height - templ->height + 1;
    for(i=0; i<6; ++i){
        fttmp[i] = cvCreateImage(
            cvSize(iwidth,iheight),32,1);
    }
    //DO THE MATCHING OF THE TEMPLATE WITH THE IMAGE:
    for(i=0; i<6; ++i){
        cvMatch Template( src, templ, fttmp[i], i);
        cvNormalize(fttmp[i],fttmp[i],1,0,CV_MINMAX)®[101];
    }
    //DISPLAY
    cvNamedWindow( "Template", 0 );
    cvShowImage( "Template", templ );
    cvNamedWindow( "Image", 0 );
    cvShowImage( "Image", src );
    cvNamedWindow( "SQDIFF", 0 );
    cvShowImage( "SQDIFF", fttmp[0] );
    cvNamedWindow( "SQDIFF_NORMED", 0 );
    cvShowImage( "SQDIFF_NORMED", fttmp[1] );
    cvNamedWindow( "CCORR", 0 );
    cvShowImage( "CCORR", fttmp[2] );
    cvNamedWindow( "CCORR_NORMED", 0 );
    cvShowImage( "CCORR_NORMED", fttmp[3] );
    cvNamedWindow( "CCOEFF", 0 );
}

```

- ① 利用匹配值的指数形式，获得更明显的匹配结果(即 $cvPow(ftmp[i],ftmp[i], 5);$)。如果结果归一化到 0.0 到 1.0 之间，很快便能看到最好匹配 0.99。如果采取指数的 5 次方，匹配值没有降多少($0.99^5=0.95$)，而不好匹配值 0.20 却大大降低($0.50^5=0.03$)。

```

        cvShowImage( "CCOEFF", ftmp[4] );
        cvNamedWindow( "CCOEFF_NORMED", 0 );
        cvShowImage( "CCOEFF_NORMED", ftmp[5] );
        //LET USER VIEW RESULTS:
        cvWaitKey(0);
    }
    else { printf("Call should be:
        "matchTemplate image template \n");}
}

```

注意本代码中 `cvNormalize()` 的使用，这可以使我们的结果显示具有一致性（回顾到一些匹配方法会返回负值）。我们可以使用 `CV_MINMAX` 标志，该标志告诉函数对浮点图像进行平移和缩放，使得所有的返回值都在 0 和 1 之间。图 7-11 给出了使用 `cvMatchTemplate()` 的各种匹配方法将人脸模板在源图像（图 7-10）上移动的结果。对室外场景，归一化的方法几乎是最好的。在这些方法中，相关系数给出了最明确的描述匹配，但是和预期一样，计算成本很高。对于特定应用来说，如视频应用中的自动部分检测或跟踪，可以试试所有这些方法，然后找到最合适的方法。

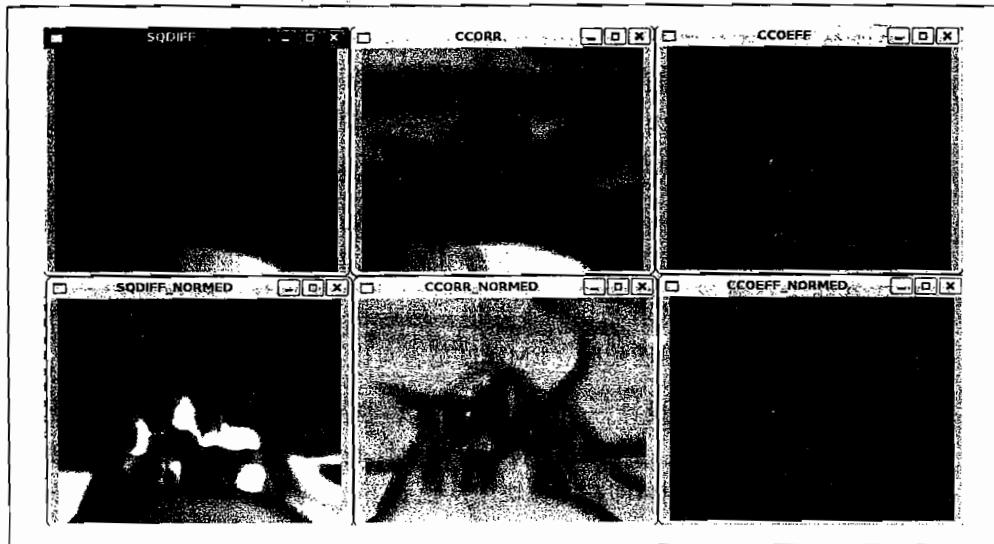


图 7-11：对于图 7-10 中描述的模板搜索，给出 6 种匹配策略的匹配结果。最好的匹配方差是 0，所以，通过左列的黑色区域和右边两列的亮点来表示匹配的相似性

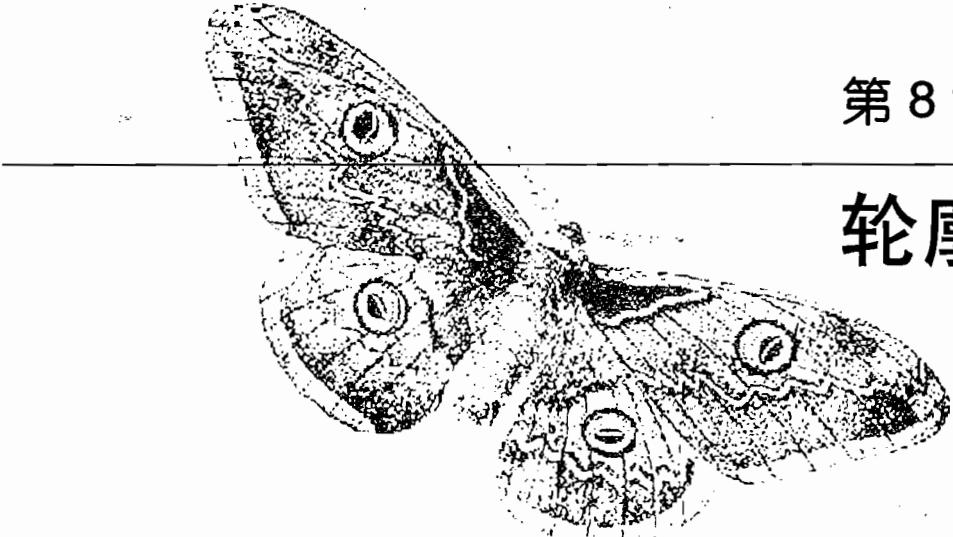
【217~219】

练习

1. 在 $0 \sim 1$ 之间生成 1000 个随机值 r_i , 定义一个 bin 的大小, 并且建立一个直方图 $1/r_i$ 。
 - a. 在每一个 bin 中, 元素数是否相同(即在 ± 10 之间)?
 - b. 提出一个处理高度非线性分布的方法, 使得每一个 bin 中在缩放比例为 10 的情况有相同数量的数据。
2. 给定三幅在书中讨论的不同光照条件下的手图像, 利用 `cvCalcHist()` 来获得室内拍照的手的肤色直方图。
 - a. 依次尝试用少量的 bin(如每维有 2 个), 中等数目的 bin(每维有 16 个)和很多 bin(每维 256 个)。然后对各种室内光线下的图像运行匹配程序(使用所有的直方图匹配方法)。描述所看到的结果。
 - b. 现在加上每维为 8 个和 32 个 bin, 试图在各种光线条件下进行匹配(室内训练和室外测试), 描述所看到的结果。
3. 和练习 2 一样, 收集手的肤色直方图。以其中的一个室内直方图样本作为模型, 并计算与另一个室内直方图、第一个室外阴影直方图以及第一个室外阳光直方图的 EMD 距离, 利用这些测量值设置一个距离阈值。
 - a. 利用该 EMD 阈值, 看看它是否能够很好地在第三幅室内、第二幅室外阴影和第二幅室外阳光下的直方图中检测到肤色直方图。报告结果。
 - b. 随机选择不是肤色的背景块的直方图, 然后观察 EMD 值是如何变化的。与真实的肤色直方图匹配时, EMD 值是否可以拒绝背景?
4. 利用收集的手的图像, 设计一个直方图, 可以判断给定的图像是在何种光线下被捕捉到的。然后, 需要建立特征——也许是整幅图像的部分采样、亮度值采样或相对亮度值采样(即在图像块中从上到下)或是中心到边缘的梯度。
5. 在三种条件下建立三类肤色模板直方图。
 - a. 利用从室内、室外阴影和室外阳光下得到的第一类直方图作为模型, 用其中的每一个分别跟第二类图进行测试, 以检验肤色匹配效果。给出匹配报告。
 - b. 利用练习 a 设计的“场景检测器”建立一个“转换直方图”模型。首先,

利用场景检测器确定要使用何种直方图模型：室内、室外阴影还是室外阳光下的。然后，使用相应的肤色模型接受或拒绝三种条件下的第二类肤色块。这种转换模型工作效果如何？

6. 建立感兴趣的肤色区域检测器。
 - a. 在室内条件下，利用一些手和脸来建立 RGB 直方图。
 - b. 利用函数 `cvCalcBackProject()` 找到肤色区域。
 - c. 利用第 5 章中的函数 `cvErode()` 来清除噪声，并利用函数 `cvFloodFill()`(第 5 章)找到图像中的肤色最大区域，这就是感兴趣区域。
7. 尝试识别手势。从摄像机中获取一个 2 英尺的手的图像，建立一些手势(不能动)：手指朝上、手指朝左和手指朝右。
 - a. 利用练习 6 得到的结果，在手周围的肤色区域求取梯度并对三种手势建立直方图模型。同时建立脸部直方图(如果图像中有人脸出现)，则可以得到一个没有任何姿势的大的肤色域模型。也可以用一些相近但是手不做姿态的区域得到直方图模型，以保证它们不会和实际的姿态混淆。
 - b. 利用网络摄像机做识别：利用感兴趣的肤色区域找到“潜在的手”；在每一个区域求取其梯度；利用直方图匹配的阈值化来检测手势。如果两个模型都在阈值之上，取最好的那个。
 - c. 将手移动 1~2 英尺，然后回来，看看梯度直方图是否依然识别这种手势？
8. 重复练习 7，但利用 EMD 匹配策略。看看将手移回来之后会发生什么现象？
9. 利用和前面一样的图像，但捕获手的一部分而不是整只手的直方图，利用 `cvMatchTemplate()` 而不是直方图匹配。当你移动该手时，在模板匹配时会产生什么问题？



第 8 章

轮廓

虽然 Canny 之类的边缘检测算法可以根据像素间的差异检测出轮廓边界的像素，但是它并没有将轮廓作为一个整体。下一步是要把这些边缘像素组装成轮廓。现在你也许希望 OpenCV 中有一个方便的函数能实现这一步，事实上这个函数就是 cvFindContours()。为了演示如何使用这些函数，本章从一些基本的问题入手。具体说来，我们会详细介绍内存存储器(memory storage)的概念，这是 OpenCV 在创建动态对象时存取内存的技术；然后是序列(sequence)基本介绍，在处理轮廓的时候通常需要使用序列。了解这些基本的概念后，我们就可以深入讨论轮廓检测的某些细节。最后我们将讨论轮廓检测的一些实际应用。

内存

OpenCV 使用内存存储器(memory storage)来统一管理各种动态对象的内存。内存存储器在底层被实现为一个有许多相同大小的内存块组成的双向链表，通过这种结构，OpenCV 可以从内存存储器中快速地分配内存或将内存返回给内存存储器。OpenCV 中基于内存存储器实现的函数，经常需要向内存存储器申请内存空间(特别是那些返回动态结果的函数)。

内存存储器可以通过以下四个函数访问：

```
CvMemStorage* cvCreateMemStorage()
    int block_size = 0
    );
```

```
void cvReleaseMemStorage(
    CvMemStorage** storage
);
void cvClearMemStorage(
    CvMemStorage* storage
);
void* cvMemStorageAlloc(
    CvMemStorage* storage,
    size_t size
);
```

【222~223】

`cvCreateMemStorage` 用于创建一个内存存储器。参数 `block_size` 对应内存存储器中每个内存块的大小。

如果 `block_size` 为 0，则表示内存块采用默认的大小，内存块默认的大小为 64 KB。该函数返回一个新创建的内存存储器指针。

`cvReleaseMemStorage` 函数通过 `storage` 获取有效的内存存储器的地址，然后释放该内存存储器的所有空间。该函数的用法和 OpenCV 中释放图像、释放矩阵或者释放其他结构的函数方法类似。

`cvClearMemStorage` 函数则用于清空内存存储器。注意，该函数是仅有的一种释放内存存储器中分配的内存的方法。该函数和通常释放内存的函数区别是，它只是将释放的内存返还给内存存储器，而并不返还给系统。实际上，通过 `cvClearMemStorage`，我们可以很方便地重复使用内存存储器中内存空间^①。注意，删除任何动态对象(如 `CvSeq`、`CvSet` 等)并不会将内存返还到内存存储器(这些结构通过在内部创建一个内存存储器以达到内存重复利用的目的)。

就像 `malloc()` 可以从堆中分配空间一样，OpenCV 中的 `cvMemStorageAlloc` 也可以从一个内存存储器中申请空间。只需要向 `cvMemStorageAlloc` 指定一个内存存储器和要申请的内存空间大小，然后返回分配内存的地址(返回值和 `malloc` 一样为 `void` 指针)。

① 实际上，还有一个 `cvRestoreMemStoragePos` 函数，可以复原内存到内存存储器。但是这个函数主要用于内部使用，超出本书的范围。

序列

序列是内存存储器中可以存储的一种对象。序列是某种结构的链表。OpenCV 中，序列可以存储多种不同的结构。你可以将序列想像为许多编程语言中都存在的容器类或者容器类模板(如 C++ 中的 vector)。序列在内存被实现为一个双端队列(deque)。因此序列可以实现快速的随机访问，以及快速删除顶端的元素，但是从中间删除元素则稍慢些。

序列中有一些重要的属性(参考例 8-1)需要了解。首先，最常用到的是 total 成员，total 存储序列中保存的数据的个数。其次是 h_prev, h_next, v_prev, 和 v_next，它们是 CV_TREE_NODE_FIELDS 的一部分，指向其他的序列(分别为上下左右四个方向)。这 4 个指针不是用来访问序列中的元素，而是用来链接不同的序列。OpenCV 中也有其他的结构包含 CV_TREE_NODE_FIELDS，我们可以使用包含 CV_TREE_NODE_FIELDS 的结构构造出更复杂的结构，例如队列、树、图等。仅仅使用变量 h_prev 和 h_next，可实现一个简单的链表。另外两个变量 v_prev 和 v_next 可以用于创建那些比较密切的复杂的拓扑结构。通过这四个变量，函数 cvFindContours 可以将图像中的复杂的轮廓构造为轮廓树。 【223~224】

例 8-1：结构 CvSeq 的定义

```
typedef struct CvSeq {
    int flags;           // miscellaneous flags
    int header_size;    // size of sequence header
    CvSeq* h_prev;      // previous sequence
    CvSeq* h_next;      // next sequence
    CvSeq* v_prev;      // 2nd previous sequence
    CvSeq* v_next;      // 2nd next sequence
    int total;          // total number of elements
    int elem_size;       // size of sequence element in byte
    char* block_max;    // maximal bound of the last block
    char* ptr;           // current write pointer
    int delta_elems;    // how many elements allocated
                        // when the sequence grows
    CvMemStorage* storage; // where the sequence is stored
    CvSeqBlock* free_blocks; // free blocks list
    CvSeqBlock* first;    // pointer to the first sequence block
}
```

创建序列

前面已经介绍了序列的结构。实际上，很多 OpenCV 函数可以返回序列。当然，我们也可以自己用 `cvCreateSeq` 函数手工创建序列。跟 OpenCV 中的许多对象一样，有一个分配函数能够创建一个序列，并返回指向所创建数据结构的指针。这个函数是 `cvCreateSeq()`。

```
CvSeq* cvCreateSeq(  
    int seq_flags,  
    int header_size,  
    int elem_size,  
    CvMemStorage* storage  
) ;
```

调用这个函数首先需要知道一些信息，这些信息用于控制创建的序列采用何种方式来组织数据。还需要序列的头大小(通常为 `sizeof(CvSeq)`^①)，以及序列要存储的元素的大小。最后，还需要为序列指定一个内存存储器，这样当序列要添加元素时，便会从内存存储器申请空间。【224】

`flags` 变量可由 3 个类值组成，不同类之间的标志可以用或运算来组合。第一类确定序列中元素的类型^②，多数类型用户可能不熟悉，另外一些类型则为 OpenCV 的内部函数使用。还有一些标志仅仅用于特定的元素类型，例如 `CV_SEQ_FLAG_CLOSED` 通常用于表示一个闭合的多边形。

- `CV_SEQ_ELTYPE_POINT` 点坐标: `(x,y)`
- `CV_SEQ_ELTYPE_CODE` Freeman: 0..7
- `CV_SEQ_ELTYPE_PPOINT` 指向一个点的指针: `&(x,y)`
- `CV_SEQ_ELTYPE_INDEX` 点的整数索引: `#{(x,y)}`

-
- ① 很明显，可以将这个参数设置为其他的值，否则它就没有存在的必要了。有时候，我们需要自己扩展序列的功能，向序列头中增加扩展变量。这时序列的头大小将需要明确指定。在扩展自己的序列的时候可以通过在定义中添加 `CV_SEQUENCE_FIELDS()` 宏，该宏的后面是扩展的成员变量。注意，当使用扩展的结构时，结构大小必须传递给函数。这是一个非常巧妙的技巧，只有非常细心的用户才会使用。
- ② 第一类的标志很少用到，它们只有在创建元素为数字组合的时候使用，如使用 `CV_32SC2`, `CV_32FC4` 等类型。如果是创建用户自己的序列，则简单地传递 0 即可。

- **CV_SEQ_ELTYPE_GRAPH_EDGE** &next_o,&next_d,&vtx_o,&vtx_d
- **CV_SEQ_ELTYPE_GRAPH_VERTEX** first_edge, &(x,y)
- **CV_SEQ_ELTYPE_TRIAN_ATR** 二叉树的节点
- **CV_SEQ_ELTYPE_CONNECTED_COMP** 联通的区域
- **CV_SEQ_ELTYPE_POINT3D** 三维的点坐标: (x,y,z)

第二类表示序列本身的性质，它可以是下面任意一个。

- **CV_SEQ_KIND_SET** 元素的集合
- **CV_SEQ_KIND_CURVE** 元素所定义的曲线
- **CV_SEQ_KIND_BIN_TREE** 二叉树
- **CV_SEQ_KIND_GRAPH** 图，其节点为序列内元素

第三类表示序列的其他属性。

- **CV_SEQ_FLAG_CLOSED** 序列是闭合的(多边形)
- **CV_SEQ_FLAG_SIMPLE** 序列是简单的(多边形)
- **CV_SEQ_FLAG_CONVEX** 序列是凸的(多边形)
- **CV_SEQ_FLAG_HOLE** 序列是一个嵌套的(多边形)

【225~226】

删除序列

```
void cvClearSeq(
    CvSeq* seq
);
```

`cvClearSeq` 可以清空序列中的所有元素。不过该函数不会将不再使用的内存返回到内存存储器中，也不会释放给系统。但是当重新向序列中添加元素时，可以重复使用这里面的内存块。如果你想回收序列中的内存块，必须使用 `cvClearMemStore` 来实现。

直接访问序列中的元素

有时候常常需要访问序列中的某个元素，这里有几种访问的方法。最常用同时也比较正确的方法是通过 cvGetSeqElem() 函数来随机访问某个元素。

```
char* cvGetSeqElem( seq, index )
```

当然还需要将 cvGetSeqElem 返回的指针转换为序列中实际存储的元素的类型。下面例子是将一个保存 CvPoint 序列中所有的点元素打印出来(序列可能是 cvFindContours 返回的轮廓):

```
for( int i=0; i<seq->total; ++i ) {
    CvPoint* p = (CvPoint*)cvGetSeqElem( seq, i );
    printf("( %d, %d )\n", p->x, p->y );
}
```

同样可以检测一个元素是否在序列中，cvSeqElemIdx 函数可以实现该功能：

```
int cvSeqElemIdx(
    const CvSeq* seq,
    const void* element,
    CvSeqBlock** block = NULL
);
```

cvSeqElemIdx 函数是一个相对耗时的操作，因此使用的时候需要慎重(所花的时候跟序列的大小成正比)^①。参数分别为序列的指针和元素的指针。另外，还有一个可选的参数 block，如果 block 不为空则存放包含所指元素的块的地址。

【226 ~ 227】

切片、复制和移动序列中的数据

cvCloneSeq 深度复制一个序列，并创建一个完全独立的序列结构。

```
CvSeq* cvCloneSeq(
    const CvSeq* seq,
```

① 准确地说，cvSeqElemIdx 是返回元素的地址。这是因为它并不是寻找序列中内容相同的元素，而且查找元素的位置是否在序列中。

```
    CvMemStorage* storage = NULL  
)  
;
```

该函数是对 cvSeqSlice() 进行简单的包装。cvSeqSlice() 函数可以为序列中的子序列生成一个新的序列(深度复制)；也可以仅为子序列创建一个头，和原来序列共用元素空间。

```
CvSeq* cvSeqSlice(  
    const CvSeq* seq,  
    CvSlice slice,  
    CvMemStorage* storage = NULL,  
    int copy_data = 0  
);
```

cvSeqSlice 中有一个 CvSlice 类型的参数，对应一个切片。我们可以用 cvSlice(a,b) 函数，或 CV_WHOLE_SEQ 宏来定义切片，其中 a 对应开始，b 对应结尾。在创建子序列的时候，只有切片之间的元素才会被复制(b 如果为 CV_WHOLE_SEQ_END_INDEX 则表示序列在 a 位置后面的所有元素)。参数 copy_data 表示是否进行深度复制，如果进行深度复制则要复制每个元素。

切片同样可以用来定义要删除或添加的序列，分别对应 cvSeqRemoveSlice 和 cvSeqInsertSlice 函数(参数一致)。

```
void cvSeqRemoveSlice(  
    CvSeq* seq,  
    CvSlice slice  
);  
void cvSeqInsertSlice(  
    CvSeq* seq,  
    int before_index,  
    const CvArr* from_arr  
);
```

【277 ~ 228】

使用指定的比较函数，我们可以对序列中的元素进行排序，或者搜索一个(排序过的)序列。元素间的比较函数需要有以下函数原型：

```
typedef int (*CvCmpFunc)(const void* a, const void* b, void*  
userdata );
```

a 和 b 为两个要排序的元素的指针，userdata 对应一个扩展数据，可以用来定义排序或查找的条件。当 a 小于 b 时比较函数返回 -1，大于 b 时返回 1，相等则返回 0。

定义了比较函数之后，就可以用 cvSeqSort 对序列进行排序，也可以用 cvSeqSearch 来搜查序列中的元素。如果序列已经排序的话，搜索一个元素的时间复杂度为 $O(\log n)$ 。如果序列没有排序，则搜索元素的时间复杂度为 $O(n)$ 。如果元素搜索成功，*elem_idx 将保存元素在序列中的索引，然后返回对应元素的指针。如果没有找到相同的元素则返回 NULL。

```
void cvSeqSort(
    CvSeq* seq,
    CvCmpFunc func,
    void* userdata = NULL
);
char* cvSeqSearch(
    CvSeq* seq,
    const void* elem,
    CvCmpFunc func,
    int is_sorted,
    int* elem_idx,
    void* userdata = NULL
);
```

cvSeqInvert 可以用于将序列进行逆序操作。该函数不会修改元素的内容，但是会将序列中的元素重新组织，改为逆序。

```
void cvSeqInvert(
    CvSeq* seq
);
```

根据用户设定的标准，OpenCV 可以通过 cvSeqPartition() 函数拆分序列^①。该函数使用类似于前面提到比较函数，但是不同的是如果两个参数相同，则返回非 0 值；如果不同则返回 0(例如用于搜索和排序的函数的求反)。 【228】

```
int cvSeqPartition(
    const CvSeq* seq,
    CvMemStorage* storage,
    CvSeq** labels,
    CvCmpFunc is_equal,
    void* userdata
);
```

① 要想进一步了解拆分，请参考论文 Hastie, Tibshirani, and Friedman [Hastie01]。

拆分操作需要申请新的内存用于存储结果。参数 `labels` 是指向序列指针的指针。当 `cvSeqPartition` 调用结束时，参数 `labels` 中是一个整数序列，序列中元素是跟序列 `seq` 中的元素一一对应。这些整数的值从 0 开始递增，是拆分后的元素类别标志。参数 `is_equal` 对应比较函数，`userdata` 对应比较函数的 `userdata` 参数。

图 8-1 为在一个 100×100 画板上随机生成的 100 个点。调用 `cvSeqPartition` 对这些点进行处理，比较函数为 2 个点之间的欧几里得距离，当 2 个点之间的距离小于等于 5 时返回 1，否则返回 0。聚类结果以参数 `labels` 中的值作为名字标出。

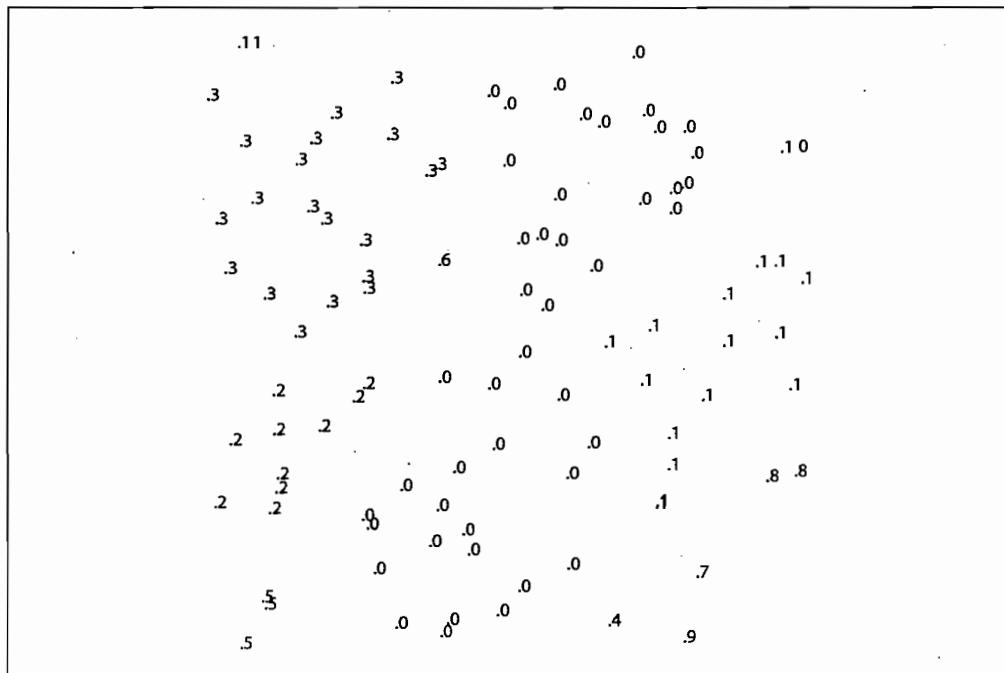


图 8-1：100x100 画布上由 100 个点组成的序列在距离小于 5 时的划分

将序列作为栈来使用

如前所述，序列在内部其实对应一个双端队列。因此，我们可以高效地从序列的任意一段(开头和结尾)访问序列。这样我们可以很自然地将序列当作一个栈使用。与 `CvSeq` 结构一起使用，下面的六个函数可将序列封装成一个栈(准确地说，是双端队列，因此它们可从两端操作元素)。

```
char* cvSeqPush(
    CvSeq* seq,
```

```

    void* element = NULL
);
char* cvSeqPushFront(
    CvSeq* seq,
    void* element = NULL
);
void cvSeqPop(
    CvSeq* seq,
    void* element = NULL
);
void cvSeqPopFront(
    CvSeq* seq,
    void* element = NULL
);
void cvSeqPushMulti(
    CvSeq* seq,
    void* elements,
    int count,
    int in_front = 0
);
void cvSeqPopMulti(
    CvSeq* seq,
    void* elements,
    int count,
    int in_front = 0
);

```

主要的操作函数是 `cvSeqPush()`, `cvSeqPushFront()`, `cvSeqPop()` 和 `cvSeqPopFront()`。因为它们操作序列两端, 可以在 $O(1)$ 时间完成操作(和序列大小没有关系)。`Push` 函数返回压栈的元素指针。`Pop` 函数弹出栈顶元素, 并且可以选择是否保存弹出的栈顶元素(`element` 不为 `NULL` 则保存弹出的元素)。`cvSeqPushMulti` 和 `cvSeqPopMulti` 用于一次将多个元素压栈和出栈, 还有一个参数用于指定对应序列的开头还是结尾。可以用宏来表示开头和结尾: `CV_FRONT` (1) 对应开头; `CV_BACK` (0) 对应结尾。

【228~230】

插入和删除元素

```

char* cvSeqInsert(
    CvSeq* seq,
    int before_index,

```

```
    void* element = NULL  
);  
void cvSeqRemove(  
    CvSeq* seq,  
    int index  
);
```

可以用 `cvSeqInsert()` 和 `cvSeqRemove()` 在序列的中间添加和删除元素。但是请注意，它们的执行效率不是很高，具体的时间依赖序列中元素的数目。

序列中块的大小

当刚开始看文档时，`cvSetSeqBlockSize()` 函数的作用可能不是很明显。它的参数为序列和新内存块的大小，当序列中内存空间不足以分配新的块时，将按照参数分配块的大小。块越大，序列中出现内存碎片的可能性就越小，但是内存中更多的内存可能被浪费。默认的内存块大小为 1K 字节，不过我们可以随时改变块的大小^①。

```
void cvSetSeqBlockSize(  
    CvSeq* seq,  
    Int delta_elems  
);
```

序列的读取和写入

当你使用序列时，如果你需要最高的性能，你可以使用一些特殊的函数修改序列（使用时需要特别小心）。这些函数通过专门的结构来保存序列的当前状态，这样使得很多后续操作可以在更短的时间内完成。

保存序列写状态的结构为 `CvSeqWriter`。`CvSeqWriter` 结构通过 `cvStartWriteSeq` 函数初始化，然后由 `cvEndWriteSeq` 关闭写状态。当序列写状态被打开的时候，可以通过 `CV_WRITE_SEQ()` 宏向序列写入元素。通过 `CV_WRITE_SEQ()` 宏写元素比 `cvSeqPush` 添加元素的效率更高，但是序列头中的一些信息并没有被即时刷新。换言之，刚写入的元素对用户来说可能并不能访问。写操作只有在执行 `cvEndWriteSeq` 函数后，才会真正的写到序列中（可以认为之前是在缓存中）。

① 在 OpenCV beta 5 之后的版本中，块的大小是随着序列的增长动态增加的，因此在一般情况下用户不必太关注底层细节。

冲中)。

【231】

如果必要，用户也可以通过 cvFlushSeqWriter() 函数来显式刷新写操作，而不需要关闭写状态。

```
void cvStartWriteSeq(
    int seq_flags,
    int header_size,
    int elem_size,
    CvMemStorage* storage,
    CvSeqWriter* writer
);
void cvStartAppendToSeq(
    CvSeq* seq,
    CvSeqWriter* writer
);
CvSeq* cvEndWriteSeq(
    CvSeqWriter* writer
);
void cvFlushSeqWriter(
    CvSeqWriter* writer
);
CV_WRITE_SEQ_ELEM( elem, writer )
CV_WRITE_SEQ_ELEM_VAR( elem_ptr, writer )
```

这些函数参数的含义都是比较明了的。cvStartWriteSeq 函数中的 seq_flags, header_size 和 elem_size 参数和 cvCreateSeq 函数的参数含义相同。cvStartAppendToSeq 初始化写状态结构到序列的末尾。CV_WRITE_SEQ_ELEM 宏则需要一个要写入的元素(如一个 CvPoint 元素)和对应的写状态结构指针；该宏首先添加一个新元素到序列，并将参数指定的元素复制到新创建的元素。

为了演示上面相关函数的用法，我们创建一个序列，并写入 100 个 320×240 矩形区域中随机点：

```
CvSeqWriter writer;
cvStartWriteSeq( CV_32SC2, sizeof(CvSeq), sizeof(CvPoint),
storage, &writer );
for( i = 0; i < 100; i++ )
{
    CvPoint pt; pt.x = rand()%320; pt.y = rand()%240;
```

```
        CV_WRITE_SEQ_ELEM( pt, writer );
    }
CvSeq* seq = cvEndWriteSeq( &writer );
```

同样，读操作也有对应的一组函数：

```
void cvStartReadSeq(
    const CvSeq* seq,
    CvSeqReader* reader,
    int reverse = 0
);
int cvGetSeqReaderPos(
    CvSeqReader* reader
);
void cvSetSeqReaderPos(
    CvSeqReader* reader,
    int index,
    int is_relative = 0
);
CV_NEXT_SEQ_ELEM( elem_size, reader )
CV_PREV_SEQ_ELEM( elem_size, reader )
CV_READ_SEQ_ELEM( elem, reader )
CV_REV_READ_SEQ_ELEM( elem, reader )
```

【232~233】

CvSeqReader 结构和 CvSeqWriter 结构类似，用来保存序列的读状态，可以用 cvStartReadSeq 函数初始化。参数 reverse 可以用于指定用正序还是逆序读序列，0 表示正序，1 表示逆序。cvGetSeqReaderPos 返回读状态在序列中的当前位置。cvSetSeqReaderPos 则可以用来设置读操作的新位置。参数 is_relative 如果非 0，则表示 cvSetSeqReaderPos 设置的新位置是当前位置的相对地址，否则是序列的绝对地址。

CV_NEXT_SEQ_ELEM() 和 CV_PREV_SEQ_ELEM() 这两个宏则用于在序列中向后或向前移动。它们并不进行错误检查，因此如果是无意越过序列边界则不会有错误提示。CV_READ_SEQ_ELEM() 和 CV_REV_READ_SEQ_ELEM() 中读元素，它们会复制当前的元素并将读状态中的位置(向前或向后)移动一个元素。这两个宏只需输入需要复制变量的名称；变量的地址会在宏内计算。

序列和数组

有时候可能需要将序列(通常是点序列)转换成数组。

```
void* cvCvtSeqToArray(
    const CvSeq* seq,
    void* elements,
    CvSlice slice = CV_WHOLE_SEQ
);
CvSeq* cvMakeSeqHeaderForArray(
    int seq_type,
    int header_size,
    int elem_size,
    void* elements,
    int total,
    CvSeq* seq,
    CvSeqBlock* block
);
```

函数 `cvCvtSeqToArray` 复制序列的全部或部分到一个连续内存数组中。如果你需要将一个保存 20 个 `CvPoint` 的序列转换成数组，那么需要的内存(`elements` 指针指向此内存)大小为 40 个整数大小(`20*sizeof(CvPoint)`)的空间。第三个可选参数 `slice` 对应序列的切片，或者默认参数 `CV_WHOLE_SEQ`。如果使用 `CV_WHOLE_SEQ`，那么整个序列被转换。

和 `cvCvtSeqToArray` 函数相对的功能是通过 `cvMakeSeqHeaderForArray()` 将数组转换为序列。这种情况下，你可以从已有的数组数据建立一个序列。函数开始的几个参数和 `cvCreateSeq()` 中用到的一样。除了要求需要复制的序列的数据(`elements`)和数组元素的个数(`total`)之外，还需要你提供序列头(`seq`)和序列的内存块结构(`block`)。这种方法生成的序列和其他方法生成的序列不太一样。具体说来就是无法改变已生成的序列的数据。

【233 ~ 234】

查找轮廓

现在终于可以讨论轮廓的问题了。首先我们需要了解轮廓到底是什么？一个轮廓一般对应一系列的点，也就是图像中的一条曲线。表示方法可能根据不同情况而有所不同。有多种方法可以表示曲线。在 OpenCV 中一般用序列来存储轮廓信息。序

列中的每一个元素是曲线中一个点的位置。关于序列表示的轮廓细节将在后面讨论，现在只要简单把轮廓想像为使用 CvSeq 表示的一系列的点就可以了。

函数 cvFindContours() 从二值图像中寻找轮廓。cvFindContours() 处理的图像可以是从 cvCanny() 函数得到的有边缘像素的图像，或者是从 cvThreshold() 及 cvAdaptiveThreshold() 得到的图像，这时的边缘是正和负区域之间的边界^①。

在介绍函数原型之前，还需要深入理解轮廓的定义。首先，我们需要了解轮廓树的概念，这对于理解 cvFindContours() 如何表达它的返回值非常重要(该方法请参考论文 Suzuki [Suzuki85])。 【234】

图 8-2 描述了 cvFindContours 的函数功能，图像的上半部分是深色背景和白色区域(被从 A 到 E 标记)的测试图像^②。下半部分是使用 cvFindContours() 函数后会得到的轮廓的说明。这些轮廓被标记为 cX 或 hX，“c”表示“轮廓(contour)”，“h”表示“孔(hole)”，“X”表示数字。其中一些轮廓用虚划线表示；表明它们是白色区域的外部边界(例如，非零区域)。孔(hole)的外部边界(例如，非零区域)即白色区域的内部边界。在图中是用点线表示外部边界的。OpenCV 的 cvFindContours() 函数可区分内部或外部边界。

包含的概念在很多应用中都非常重要。因此，OpenCV 允许得到的轮廓被聚合成一个轮廓树^③，从而把包含关系编码到树结构中。这个测试图的轮廓树在根节点的轮廓叫 c0，孔 h00 和 h01 是它的子节点。这些轮廓中直接包含的轮廓成为它们的子节点，依此类推。

注意：通过图 8-1，可以看到如何处理二值图像，由此可以推测出 cvCanny() 或者其他的边缘检测算法得到的图像使用 cvFindContours() 会得到怎样的结果。实际上，cvFindContours() 并不知道图像的边缘(edge)信息。也就是说，对 cvFindContours() 而言，“边缘”只是很细的“白色”区域。因此，对于每个外部轮廓，必然有个孔的轮廓和它同时产生。这个孔只是在外部边界(exterior boundary)的里侧。可以把它想像成一个标志着内部边缘的白-黑过渡。 【235】

- ① 给 cvFindContours() 函数传递边缘图像(edge images)和二值图像(binary images)是有细微差别的。我们会简要讨论一下。
- ② 为了确保轮廓线能够清晰绘制，图中深色的区域被描画成灰色，可以简单假设这幅图像被进行阈值操作，传递给 cvFindContours() 函数之前灰色区域会被改变成黑色。
- ③ 轮廓树最早在 Reeb [Reeb46] 中被提出，随后在 [Bajaj97], [Kreveld97], [Passucci02] 和 [Carr04] 中被改进。

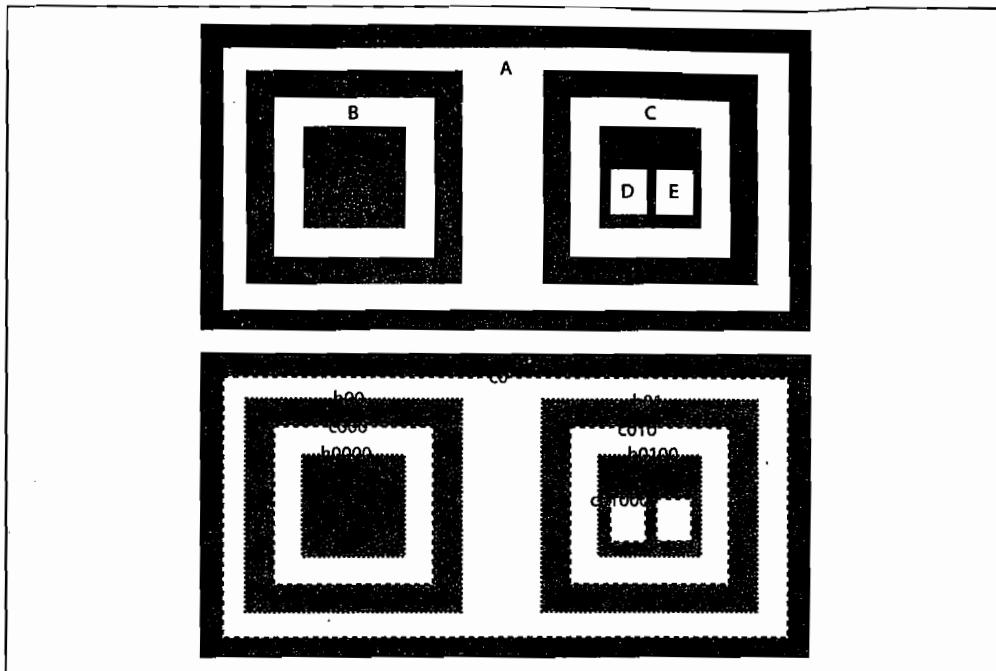


图 8-2：传递给 `cvFindContours()` 测试图(上图)，得到的轮廓(下图)：得到的轮廓只可能有两种，外部轮廓(虚划线)或者孔(点线)

现在来看 `cvFindContours()` 函数：下面将准确阐述如何定义及如何理解返回值。

```
int cvFindContours(
    IplImage*        img,
    CvMemStorage*    storage,
    CvSeq**          firstContour,
    int              headerSize = sizeof(CvContour),
    CvContourRetrievalMode mode = CV_RETR_LIST,
    CvChainApproxMethod method = CV_CHAIN_APPROX_SIMPLE
);
```

第一个参数是输入图像，图像必须是 8 位单通道图像，并且应该被转化成二值的(例如，所有非 0 像素的值都是一个定值)。`cvFindContours()` 运行的时候，这个图像会被直接涂改，因此如果是将来还有用的图像，应该复制之后再传给 `cvFindContours()`。下一个参数是内存存储器，`cvFindContours()` 找到的轮廓记录在此内存里。正如之前所说，这个存储的空间应该由 `cvCreateMemStorage()` 分配。接着是指向 `CvSeq*` 的一个指针 `firstContour`。无需动手，`cvFindContours()` 会自动分配该指针。实际上，只要在这里传一个指针就可以，

函数会自动设置。不要分配和释放(new/delete 或者 malloc/free)。就是这个指针(例如, *firstContour)指向轮廓树的首地址(head)。cvFindContours()的返回值是找到的所有轮廓的个数^①。

```
CvSeq* firstContour = NULL;  
cvFindContours( ..., &firstContour, ... );
```

headerSize 告诉 cvFindContours()更多有关对象分配的信息, 它可以被设定为 sizeof(CvContour) 或者 sizeof(CvChain)(当近似方法参数 method 被设定为 CV_CHAIN_CODE 时使用后者)^②。最后是 mode 和 method 参数, 它们分别指定计算方法和如何计算。

mode 变量可以被设置为以下四个选项之一: CV_RETR_EXTERNAL、CV_RETR_LIST、CV_RETR_CCOMP 或 CV_RETR_TREE。mode 的值向 cvFindContours()说明需要的轮廓类型, 和希望的返回值形式。具体说来, mode 的值决定把找到的轮廓如何挂到轮廓树节点变量(h_prev, h_next, v_prev 和 v_next)上。图 8-3 展示了四种可能的 mode 值所得到的结果的拓扑结构。

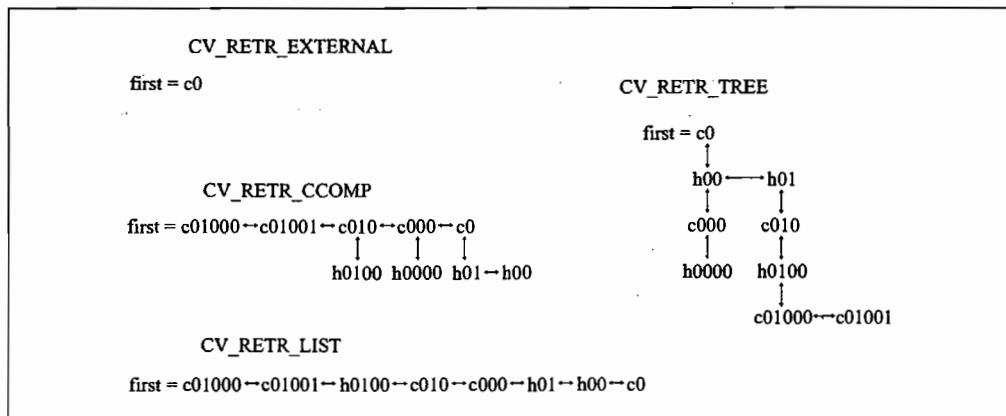


图 8-3: 树的节点变量与 cvFindContours()得到的轮廓“连接”的方法

每种情况下, 结构都可以看成是被“横向”连接(`h_next` 和 `h_prev`)联系和被“纵向”连接(`v_next` 和 `v_prev`)不同“层次”。

【236】

-
- ① 我们很快就能看到, 轮廓树只是 cvFindContours()组织所获得轮廓的一种方法。任何情况下, 它们都是通过使用轮廓的 CV_TREE_NODE_FIELDS 被组织起来的。CV_TREE_NODE_FIELDS 在开始讲解序列的时候就被介绍了。
 - ② 事实上, headerSize 可以是等于或大于上述给出的值的任意数。

- **CV_RETR_EXTERNAL** 只检测出最外的轮廓。图 8-2 中，只有一个最外轮廓，因此图 8-3 中第一个轮廓指向最外的序列，除此之外没有别的连接。
- **CV_RETR_LIST** 检出所有的轮廓并将它们保存到表(list)中。图 8-3 描绘了从图 8-2 的样图中得到的表。在这个例子中，有 8 条轮廓被找到，它们相互之间由 `h_prev` 和 `h_next` 连接(这里并没有使用 `v_prev` 和 `v_next`)。
- **CV_RETR_CCOMP** 检出所有的轮廓并将它们组织成双层结构(two-level hierarchy)，顶层边界是所有成分的外界边界，第二层边界是孔的边界。图 8-3 中，我们能看到 5 个外部边界，其中 3 个包含孔。孔被 `v_next` 和 `v_prev` 与它们的外部边界连接起来。最外层的边界 `c0` 中有两个孔，因为 `v_next` 可以只包括一个值，此节点可以只有一个子节点。`c0` 之内的所有的孔相互间由 `h_prev` 和 `h_next` 指针连接。
- **CV_RETR_TREE** 检出所有的轮廓并且重新建立网状的轮廓结构。在我们给出的例子(图 8-2 和 8-3)中，这意味着根节点是最外的轮廓 `c0`。`c0` 之下是孔 `h00`，在同一层次中与另一个孔 `h01` 相连接。同理，每个孔都有子节点(相对应的是 `c000` 和 `c010`)，这些子节点与父节点被垂直连接起来。这个步骤一直持续到图像最内层的轮廓，这些轮廓会成为树叶节点。 【237】

以下的五个值与方法相关(例如，轮廓如何被近似)。

- **CV_CHAIN_CODE** 用 Freeman 链码输出轮廓^①；*其他方法输出多边形(顶点的序列)^②。†
- **CV_CHAIN_APPROX_NONE** 将链码编码中的所有点转换为点。
- **CV_CHAIN_APPROX_SIMPLE** 压缩水平，垂直或斜的部分，只保存最后一个点。
- **CV_CHAIN_APPROX_TC89_L1** 或 **CV_CHAIN_APPROX_TC89_KCOS** 使用 Teh-Chin 链逼近算法中的一个。
- **CV_LINK_RUNS** 与上述的算法完全不同的算法，连接所有的水平层次的轮

① Freeman 链码将在“使用序列表示轮廓”一节中讨论。

② 这里顶点的意思为 CvPoint 类型的点。由 cvFindContours() 函数创建的序列，跟使用 `CV_SEQ_ELTYPE_POINT` 标志由 cvCreateSeq() 函数创建的序列相同。(cvCreateSeq() 函数和标志的意义将在本章的后面部分讨论)。

廓。此方法只可与 CV_RETR_LIST 搭配使用。

使用序列表示轮廓

序列和轮廓有太多的内容需要介绍。我们这里只关注它们基本的特性。当调用 cvFindContours 函数的时候，返回多个序列。序列的类型依赖于调用 cvFindContours 时所传递的参数。默认情况下使用 CV_RETR_LIST 和 CV_CHAIN_APPROX_SIMPLE 参数。

序列中保存一系列的点，这些点构成轮廓，轮廓是本章的重点。但是需要注意的是，轮廓只是序列所能表示物体的一种^①。轮廓是点的序列，可以用来表示图像空间中的曲线。这种点的序列很常用，所以需要有专门的函数来帮助我们对它们进行处理。下面是一组这样的处理函数。

```
int cvFindContours(
    CvArr*           image,
    CvMemStorage*    storage,
    CvSeq**          first_contour,
    int              header_size      = sizeof(CvContour),
    int              mode            = CV_RETR_LIST,
    int              method          = CV_CHAIN_APPROX_SIMPLE,
    CvPoint          offset          = cvPoint(0, 0)
);
CvContourScanner cvStartFindContours(
    CvArr*           image,
    CvMemStorage*    storage,
    int              header_size      = sizeof(CvContour),
    int              mode            = CV_RETR_LIST,
    int              method          = CV_CHAIN_APPROX_SIMPLE,
    CvPoint          offset          = cvPoint(0, 0)
);
CvSeq* cvFindNextContour(
    CvContourScanner scanner
);
```

① 除此之外，还需要指出一点不同的地方，为了不使正文部分因为一些技术细节而跑题，在脚注中解释这个问题。CvContour 类型并不完全等价于 CvSeq，CvContour 是从 CvSeq 扩展而来，它还含有几个其他的成员，例如颜色、外包矩形区域等。

```

void cvSubstituteContour(
    CvContourScanner scanner,
    CvSeq* new_contour
);
CvSeq* cvEndFindContour(
    CvContourScanner* scanner
);
CvSeq* cvApproxChains(
    CvSeq* src_seq,
    CvMemStorage* storage,
    int method = CV_CHAIN_APPROX_SIMPLE,
    double parameter = 0,
    int minimal_perimeter = 0,
    int recursive = 0
);

```

【238~239】

第一个函数是 `cvFindContours()`，在前面我们已经提到过。接着是 `cvStartFindContours()` 函数，它和 `cvFindContours()` 的功能类似。但是 `cvStartFindContours()` 每次返回一个轮廓，而不是像 `cvFindContours()` 那样一次查找所有的轮廓然后统一返回。调用 `cvStartFindContours()` 函数后，返回一个 `CvSequenceScanner` 结构。`CvSequenceScanner` 结构中包含一些状态信息，这些信息不可读^①。你可以通过在 `CvSequenceScanner` 结构上依次调用 `cvFindNextContour()` 来查找剩余的轮廓。当全部轮廓都被找完之后，`cvFindNextContour()` 将返回 `NULL`。

`cvSubstituteContour()` 函数用于替换 `scanner` 指向的轮廓。该函数的一个特性是，如果参数 `new_contour` 为 `NULL`，那么当前的轮廓将被从 `scanner` 指定的树或链表中删除(受影响的序列会作适当更新，来保证不会有指针指向不存在的物体)。

函数 `cvEndFindContour()` 结束轮廓查找，并且将 `scanner` 设置为结束状态。注意，`scanner` 并没有被删除，实际上该函数返回的是指针所指序列的第一个元素。

【239】

① 注意混淆 `CvSequenceScanner` 结构和 `CvSeqReader` 结构，它们的作用类似，不同的是后者从序列中读出元素，而前者是从序列列表中依次读出每个序列。

最后一个函数 `cvApproxChains()` 函数。该函数将 Freeman 链转换为多边形表示(精确转换或者近似拟合)。我们将在下一节中详细讲述 `cvApproxPoly()` 函数(请参考“多边形逼近”小节)。

Freeman 链码

一般情况下，通过 `cvFindContours` 获取的轮廓是一系列顶点的序列。另一种不同的表达是设置 `method` 参数为 `CV_CHAIN_CODE`，然后生成轮廓。当选择 `CV_CHAIN_CODE` 标志的时候，检测的轮廓通过 Freeman 链码[Freeman67](图 8-4)的方式返回。在 Freeman 链码中，多边形被表示为一系列的位移，每一个位移有 8 个方向，这 8 个方向使用整数 0 到 7 表示。Freeman 链码对于识别一些形状的物体很有帮助。如果得到的是 Freeman 链码，可以通过以下两个函数读出每个点：

```
void cvStartReadChainPoints(
    CvChain* chain,
    CvChainPtReader* reader
);
CvPoint cvReadChainPoint(
    CvChainPtReader* reader
);
```

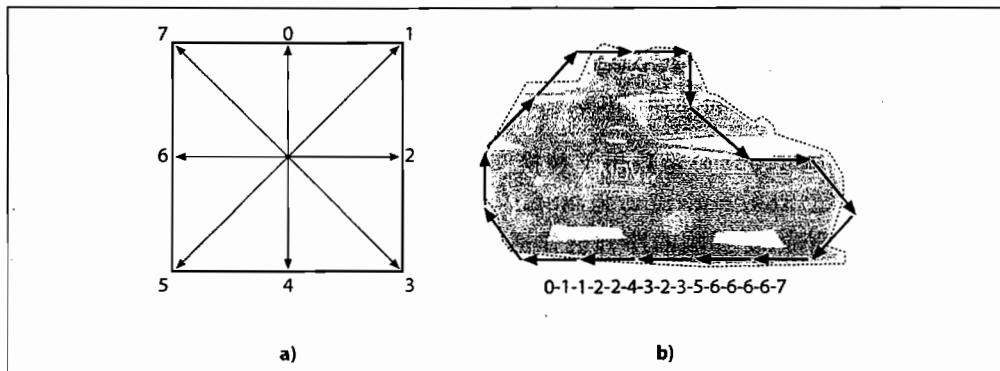


图 8-4：a)Freeman 链码对应的八个方向；b)用 Freeman 链码表示轮廓，从汽车后保险杠处开始

第一个函数用来初始化 Freeman 链 `CvChainPtReader` 结构，第二个函数通过 `CvChainPtReader` 来读每个点，`CvChainPtReader` 对应当前状态。结构 `CvChain`

从 CvSeq 扩展得来^①。和 CvContourScanner 从多个轮廓间迭代一样，CvChainPtReader 用于迭代一个使用 Freeman 链码表示的轮廓中的每个点。CvChainPtReader 和 CvSeqReader 的用法类似。如您所期望，当所有点都读完后，返回的 CvChainPtReader 值为 NULL。

【240~241】

绘制轮廓

一个经常使用的功能是在屏幕上绘制检测到的轮廓。绘制可以用 cvDrawContours 函数完成：

```
void cvDrawContours(
    CvArr* img,
    CvSeq* contour,
    CvScalar external_color,
    CvScalar hole_color,
    int max_level,
    int thickness = 1,
    int line_type = 8,
    CvPoint offset = cvPoint(0,0)
);
```

第一个参数为要绘制轮廓的图像。第二个参数是要绘制的轮廓，它不像乍看上去那样简单，它是轮廓树的根节点。其他的参数(主要是 max_level)将会控制如何绘制轮廓树。下一个参数很容易理解，是绘制轮廓所用的颜色。但是 hole_color 呢？请回忆轮廓的分类，有外轮廓，也有“洞”(图 8-2 中的虚划线和点线)。无论绘制单个轮廓还是轮廓树中的所有轮廓，标记为“洞”的轮廓都会使用 hole_color 指定的颜色绘制。

通过 max_level 变量可以告诉 cvDrawContours() 如何处理通过节点树变量连结到一个轮廓上的其他任何轮廓。此变量可以被设置为便利轮廓的最大深度。因此，max_level=0 表示与输入轮廓属于同一等级的所有轮廓(更具体地说，输入轮廓和与其相邻的轮廓)被画出，max_level=1 表示与输入轮廓属于同一等级的所有轮廓

① 前面也有关于扩展 CvSeq 扩展的讨论，CvChain 就是类似的扩展。CvChain 通过 CV_SEQUENCE_FIELDS 宏以及几个其他的元素来定义，其中有一个表示起点的 CvPoint 变量。你可以认为 CvChain 是从 CvSeq 派生而来。所以，cvApproxChains 虽然返回 CvSeq 指针，但实际上是一个 CvChain 结构，并不是普通的序列。

与其子节点被画出，以此类推。如果想要画的轮廓是由 cvFindContours() 的 CV_RETR_CCOMP 或 CV_RETR_TREE 模式得到的话，max_level 的负值也是被支持的。在这种情况下，max_level=-1 表示只有输入轮廓会被画出，max_level=-2 表示输入轮廓与其直系(仅直接相连的)子节点会被画出，以此类推。可参考例子代码.../opencv/samples/c/contours.c 来了解具体使用。

参数 thickness 和 line_type 就如其字面含义所示^①。最后，我们可以给绘图程序一个偏移量，这样轮廓可以被画在指定的精确的坐标上。当轮廓坐标被转换成质心坐标系或其他局部坐标系的时候，这个特性非常有用。

如果在图像上的不同感兴趣区域(ROI)多次执行 cvFindContour()，然后又想将所有结果在原来的大图像上显示出来，偏移量 offset 也很有用。相反，也可以先从大图提出一个轮廓，然后再用 offset 和填充，在小图像上形成和轮廓对应的蒙版(mask)。

【241~242】

轮廓例子

例 8-2 摘自 OpenCV 安装包。首先创建一个窗口用于显示图像，滑动条(trackbar)用于设置阈值，然后对采二值化后的图像提取轮廓并绘制轮廓。当控制参数的滑动条变化时，图像被更新。

例 8-2：根据滑动条参数检测轮廓，在滑动条变化的时候重新检测

```
#include <cv.h>
#include <highgui.h>

IplImage* g_image = NULL;
IplImage* g_gray = NULL;
int g_thresh = 100;
CvMemStorage* g_storage = NULL;
void on_trackbar(int) {
    if( g_storage==NULL ) {
        g_gray = cvCreateImage( cvGetSize(g_image), 8, 1 );
```

-
- ① 具体说来，thickness=-1(即 CV_FILLED)可以采用填充的方式来把轮廓树(或者单个轮廓)绘制，绘制的结果跟提取轮廓所用的二值图一样。这个特性与偏移量变量相结合，可以被用来处理一些相当复杂的情形：轮廓的交叉和融合，测试点与轮廓关系，形态学操作(腐蚀/膨胀)等。

```

    g_storage = cvCreateMemStorage(0);
} else {
    cvClearMemStorage( g_storage );
}
CvSeq* contours = 0;
cvCvtColor( g_image, g_gray, CV_BGR2GRAY );
cvThreshold( g_gray, g_gray, g_thresh, 255, CV_THRESH_BINARY );
cvFindContours( g_gray, g_storage, &contours );
cvZero( g_gray );
if( contours )
    cvDrawContours(
        g_gray,
        contours,
        cvScalarAll(255),
        cvScalarAll(255),
        100
    );
    cvShowImage( "Contours", g_gray );
}
int main( int argc, char** argv )
{
    if( argc != 2 || !(g_image = cvLoadImage(argv[1])) )
        return -1;
    cvNamedWindow( "Contours", 1 );
    cvCreateTrackbar(
        "Threshold",
        "Contours",
        &g_thresh,
        on_trackbar
    );
    on_trackbar(0);
    cvWaitKey();
    return 0;
}

```

【242~243】

我们关注的重点在 `on_trackbar` 函数内。如果全局的参数 `g_storage` 为 `NULL` 的话，则用 `cvCreateMemStorage(0)` 创建一个内存存储器。`g_gray` 被初始化为和 `g_image` 同样大小的黑色图像，但是为单通道图像。如果 `g_storage` 非空的话，则先用 `cvClearMemStorage` 清空内存存储器中的空间，这样以便于重复利用内存存储器中的资源。然后创建一个 `CvSeq*` 指针，该指针用来保存 `cvFindContours()`

检测到的轮廓。

然后 `g_image` 被转换为灰度图像，接着用 `g_thresh` 为参数进行二值化处理，得到的二值图像保存在 `g_gray` 中。`cvFindContours` 从二值图像 `g_gray` 查找轮廓，然后将得到的轮廓用 `cvDrawContours()` 函数绘制为白色到灰度图像。最终图像在窗口中显示出来，并将在回调函数开始处申请的结构释放。

另一个轮廓例子

在例 8-3 中，我们检测出输入图像的轮廓，然后逐个绘制每个轮廓。从这个例子中，我们可以了解到轮廓检测方法(如代码中是 `CV_RETR_LIST`)以及 `max_depth`(代码中是 0)等参数的细节。如果设置的 `max_depth` 是一个比较大的值，你可以发现 `cvFindContours()` 返回的轮廓是通过 `h_next` 连接被遍历。对于其他一些拓扑结构(`CV_RETR_TREE`, `CV_RETR_CCOMP` 等)，你会发现有些轮廓被画过不止一次。

例 8-3：在输入图像上寻找并绘制轮廓

```
int main(int argc, char* argv[]) {
    cvNamedWindow( argv[0], 1 );
    IplImage* img_8uc1=cvLoadImage(argv[1],CV_LOAD_IMAGE_GRAYSCALE );
    IplImage* img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
    IplImage* img_8uc3 = cvCreateImage( cvGetSize(img_8uc1), 8, 3 );
    cvThreshold( img_8uc1, img_edge, 128, 255, CV_THRESH_BINARY );
    CvMemStorage* storage = cvCreateMemStorage();
    CvSeq* first_contour = NULL;
    int Nc = cvFindContours(
        img_edge,
        storage,
        &first_contour,
        sizeof(CvContour),
        CV_RETR_LIST // Try all four values and see what happens
    );
    int n=0;
    printf( "Total Contours Detected: %d\n", Nc );
    for( CvSeq* c=first_contour; c!=NULL; c=c->h_next ) {
        cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
        cvDrawContours(
            img_8uc3,
            c,
            0, 255, 0, 2, CV_FILLED );
    }
}
```

```

CVX_RED,
CVX_BLUE,
0, // Try different values of max_level, and see what happens
2,
8
);
printf("Contour # %d\n", n );
cvShowImage( argv[0], img_8uc3 );
printf(" %d elements:\n", c->total );
for( int i=0; i<c->total; ++i ) {
CvPoint* p = CV_GET_SEQ_ELEM( CvPoint, c, i );
printf(" (%d,%d)\n", p->x, p->y );
}
cvWaitKey(0);
n++;
}

printf("Finished all contours.\n");
cvCvtColor( img_8uc1, img_8uc3, CV_GRAY2BGR );
cvShowImage( argv[0], img_8uc3 );
cvWaitKey(0);
cvDestroyWindow( argv[0] );
cvReleaseImage( &img_8uc1 );
cvReleaseImage( &img_8uc3 );
cvReleaseImage( &img_edge );
return 0;
}

```

【243~244】

深入分析轮廓

当分析一个图像的时候，针对轮廓我们也许有很多事情要做。我们对轮廓常用的操作有识别和处理，另外相关的还有多种对轮廓的处理，如简化或拟合轮廓，匹配轮廓到模板，等等。

在这一节中，我们将介绍一些 OpenCV 函数，这些函数或者可以完成我们的任务，或者可以使得工作变得容易。

多边形逼近

当我们绘制一个多边形或者进行形状分析的时候，通常需要使用多边形逼近一个轮廓，使得顶点数目变少。有多种方法可以实现这个功能，OpenCV 实现了其中的一种逼近算法^①。函数 cvApproxPoly 是该算法的一种实现，可以处理轮廓序列。

```
CvSeq* cvApproxPoly(
    const void*     src_seq,
    int             header_size,
    CvMemStorage*   storage,
    int             method,
    double          parameter,
    int             recursive = 0
);
```

我们可以传递一个列表或数状序列给 cvApproxPoly，然后对其表示的轮廓进行处理。函数的返回值对应第一个轮廓，同样我们也可用通过 h_next(以及 v_next)来访问返回的其他的轮廓。

因为 cvApproxPoly 在返回结果的时候需要创建新的对象，因此需要指定一个内存存储器以及头结构大小(一般为 sizeof(CvContour))。

逼近的算法目前只可使用 CV_POLY_APPROX_DP(如果其他算法也被实现的话，可以选择其他算法)。另外两个参数为逼近算法参数(目前只用到第一个)。parameter 参数指定逼近的精度。如果想了这个参数如何起作用的，必须仔细了解具体的算法^②。最后一个参数指定是否针对全部的轮廓(通过 h_next 和 v_next 可达的)进行逼近。如果为 0，则表示只处理 src_seq 指向轮廓。

下面简要介绍一下算法的工作原理。参考图 8-5，算法先从轮廓(图 b)选择 2 个最远的点，然后将 2 个连成一个线段(图 c)，然后再查找轮廓上到线段距离最远的点，添加到逼近后的新轮廓(图 d)。算法反复迭代，不断将最远的点添加到结果中。直到所有的点到多边形的最短距离小于 parameter 参数指定的精度(图 f)。从这里可以看出，精度和轮廓的周长，或者外包矩形周长的几分之一比较合适。【245】

-
- ① OpenCV 采用 Douglas-Peucker(DP) 逼近算法 [Douglas73]，其他的常用算法有 Rosenfeld-Johnson[Rosenfeld73] 和 Teh-Chin[Teh89]。
 - ② 如果没有太大问题，可以设置这个参数为整个曲线长度的 n 分之一。

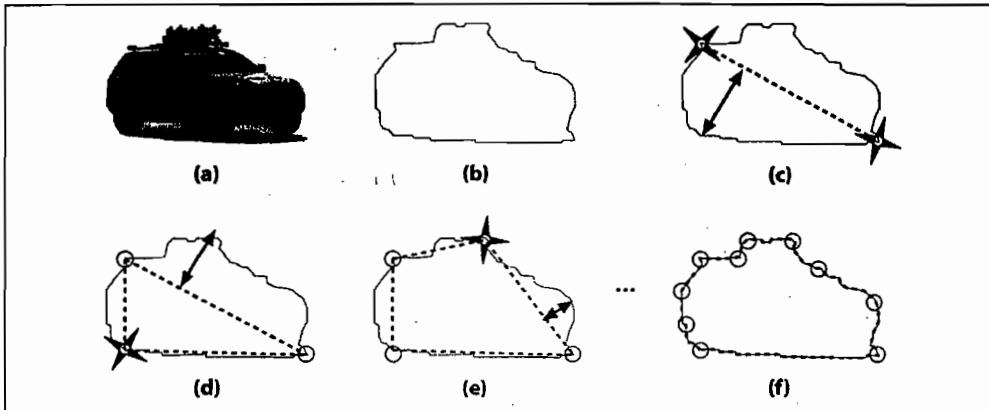


图 8-5: cvApproxPoly 实现的 DP 算法的示意图。(a)为原始图像, (b)为提取的轮廓, 从 2 个距离最远的点开始(c), 其他的点的选择过程如(d)-(f)所示

曲线逼近的过程和寻找关键点(dominant points)的过程密切相关。跟曲线上的其他点相比, 关键点是那些包含曲线信息比较多的点。关键点在逼近算法以及其他应用中都会涉及。函数 cvFindDominantPoints() 实现了被称为 IPAN* [Chetverikov99] 的算法。

```

CvSeq* cvFindDominantPoints(
    CvSeq* contour,
    CvMemStorage* storage,
    int method = CV_DOMINANT_IPAN,
    double parameter1 = 0,
    double parameter2 = 0,
    double parameter3 = 0,
    double parameter4 = 0
);

```

本质上, IPAN 算法通过扫描轮廓上并在曲线内部使用可能顶点构造三角形来实现。对于三角形的大小和张角有特殊要求(图 8-6)。在比某一特定的全局阈值和它的相邻点的张角小的情况下, 具有大张角的点被保留^①。【246】

函数 cvFindDominantPoints() 按照惯例使用参数 CvSeq* 和 CvMemStorage*。并且要求指定一个方法, 和 cvApproxPoly() 相同, 目前可供选择的方法只有一个, 就是 CV_DOMINANT_IPAN。

^① “图像模式分析小组”, 匈牙利科学院。算法在被引用的时候通常写为“IPAN99”, 因为该算法最早在 1999 年发表。

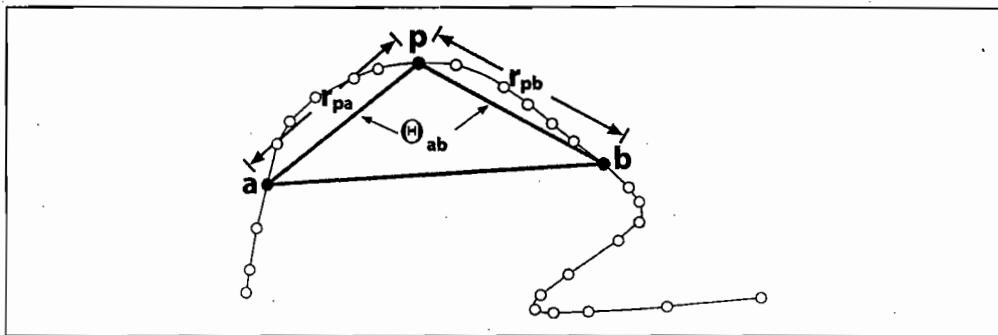


图 8-6: IPAN 算法使用三角形 abp 来描述点 p

接下来的四个参数是: 最短距离 d_{\min} , 最长距离 d_{\max} , 相邻距离 d_n 和最大角度 θ_{\max} 。如图 8-6 所示, 算法首先把所有两边距离 r_{pa} 和 r_{pb} 在 d_{\min} 和 d_{\max} 之间, $\theta_{ab} < \theta_{\max}$ 的三角形找出来。然后保留对于距离 d_n (d_n 的大小不得超过 d_{\max})有最小夹角 θ_{ab} 的所有点 p 。 d_{\min} , d_{\max} , d_n , 和 θ_{\max} 的典型值可以是 7, 9, 9, 150(最后一个参数是以度数为单位的角的大小)。

特性概括

轮廓处理中经常遇到的另一个任务是计算一些轮廓变化的概括特性。这可能包括长度或其他一些反映轮廓整体大小的量度。另一个有用的特性是轮廓矩(contour moment), 可以用来概括轮廓的总形状特性(我们下一节讨论)。

长度

函数 `cvContourPerimeter()` 作用于一个轮廓并返回其长度。事实上, 此函数是一个调用通用函数 `cvArcLength()` 的宏。

```
double cvArcLength(
    const void* curve,
    CvSlice slice = CV_WHOLE_SEQ,
    int is_closed = -1
);
#define cvContourPerimeter( contour ) \
    cvArcLength( contour, CV_WHOLE_SEQ, 1 )
```

`cvArcLength()` 的第一个参数是轮廓, 其形式可以是点的序列(`CvContour*`或`CvSeq*`)或任一 $n \times 2$ 的点的数组。后边的参数是 `slice`, 以及表明是否将轮廓视为闭合的一个布尔类型(例如, 是否将轮廓的最后一个点视为和第一个点有连接)。

`slice` 可以让我们只选择曲线(curve)^①上的点的部分集合。

【247~248】

一个和 `cvArcLength()` 有紧密关系的函数是 `cvContourArea()`，如其名称所示，这个函数同于计算轮廓的面积。函数的参数 `contour` 和 `slice` 和 `cvArcLength()` 一样。

```
double cvContourArea(
    const CvArr* contour,
    CvSlice slice = CV_WHOLE_SEQ
);
```

边界框

当然长度和面积只是轮廓的简单特性，更复杂一些的特性描述应该是矩形边界框、圆形边界框或椭圆形边界框。有两种方法可以得到矩形边界框，圆形与椭圆形边界框各只有一种方法。

```
CvRect cvBoundingRect(
    CvArr* points,
    int update = 0
);
CvBox2D cvMinAreaRect2(
    const CvArr* points,
    CvMemStorage* storage = NULL
);
```

最简单的方法是调用函数 `cvBoundingRect()`；它将返回一个包围轮廓的 `CvRect`。第一个参数 `points` 可以是由点组成的序列，一个轮廓(`CvContour*`)或者一个， $n \times 1$ 双通道的矩阵(`CvMat*`)。为了理解第二个参数 `update`，我们需要想想前面的描述。当时说 `CvContour` 并不完全等于 `CvSeq`，`CvSeq` 能实现的 `CvContour` 都可以实现，`CvContour` 甚至能做的更多一点。其中一个附加功能就是 `CvRect` 成员可以记载轮廓自己的边界框。如果调用函数 `cvBoundingRect()` 时参数 `update` 设置为 0，便可以直接从 `CvContour` 的成员中获取边界框；如果将 `update` 设置为 1，边界框会被计算出(`CvContour` 成员的内容也会被更新)。

① 基本上总是默认值 `CV_WHOLE_SEQ` 被使用。结构体 `CvSlice` 只包含两个元素：`start_index` 和 `end_index`，你可以使用帮助创建函数 `cvslice(int start, int end)` 在这里创建自己的 `slice`。请注意 `CV_WHOLE_SEQ` 是以 0 开始，以一个很大数结束的 `slice` 的缩写。

`cvBoundingRect()`得到的长方形的一个问题是，`CvRect`只能表现一个四边水平和竖直的长方形。然而函数 `cvMinAreaRect2()`可以返回一个包围轮廓最小的长方形，这个长方形很可能是倾斜的；请看图 8-7，该函数的参数和 `cvBoundingRect()` 的相似。OpenCV 的数据类型 `CvBox2D` 就是用来表述这样的长方形的。【248】

```
typedef struct CvBox2D {  
    CvPoint2D32f center;  
    CvSize2D32f size;  
    float angle;  
} CvBox2D;
```

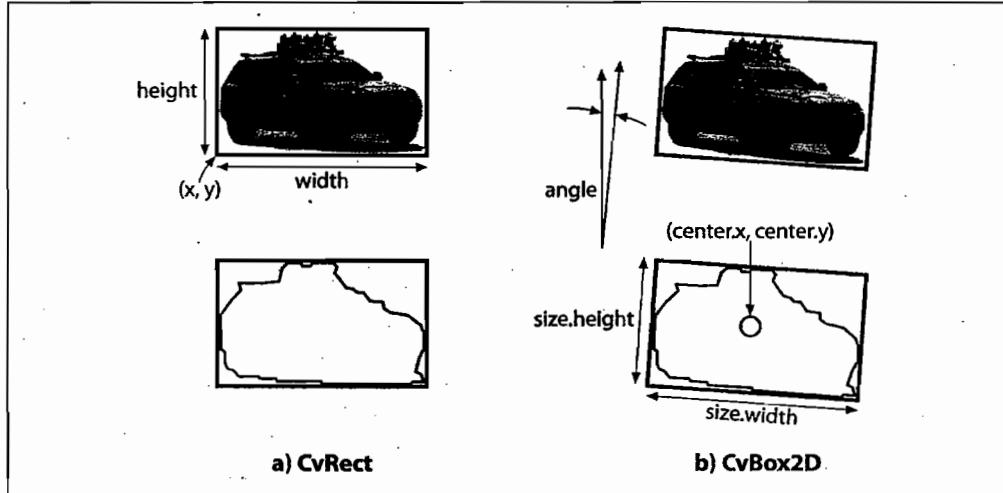


图 8-7：`CvRect` 只能表示一个方正的正方形，但 `CvBox2D` 可以表示任何倾斜度的正方形

圆形和椭圆形边界

接着我们来看函数 `cvMinEnclosingCircle()`^①。该函数和矩形边界框的作用基本相同，输入同样很灵活，可以是点的序列，也可是一维点的数组。

```
int cvMinEnclosingCircle(  
    const CvArr* points,  
    CvPoint2D32f* center,
```

① 要想进一步了解这些拟合方法的内部机制，请参考 Fitzgibbon and Fisher [Fitzgibbon95] 和 Zhang [Zhang96]。

```
    float* radius  
);
```

OpenCV 里没有专门用来表示圆的结构，因此需要给函数 `cvMinEnclosingCircle()` 传递中心和浮点型半径的两个指针来获取计算结果。

与最小包围圆一样，OpenCV 提供一函数来拟合一组点，以获取最佳拟合椭圆。

```
CvBox2D cvFitEllipse2(  
    const CvArr* points  
);
```

【249】

`cvMinEnclosingCircle()` 和 `cvFitEllipse2()` 的细微差别在于，前者只简单计算完全包围已有轮廓的最小圆，而后者使用拟合函数返回一个与轮廓最相近似的椭圆。这意味着并不是轮廓中所有的点都会被包在 `cvFitEllipse2()` 返回的椭圆中。该拟合由最小二乘拟合方法算出。

椭圆的拟合的结果由 `CvBox2D` 结构体返回，给出的矩形正好完全包围椭圆，如图 8-8 所示。

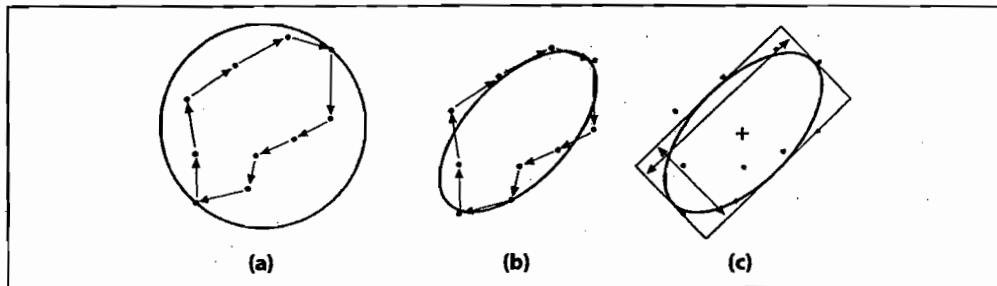


图 8-8: a)10 个点的轮廓与最小包围圆； b)与最佳拟合椭圆； c)OpenCV 给出的表示该椭圆的矩形

几何

在处理 `CvBox2D` 或多边形边界的时候，经常需要进行多边形以及边界框的重叠判断。OpenCV 提供了一组方便的小函数用于此类几何测试。

```
CvRect cvMaxRect(  
    const CvRect* rect1,  
    const CvRect* rect2
```

```

);
void cvBoxPoints(
    CvBox2D box,
    CvPoint2D32f pt[4]
);
CvSeq* cvPointSeqFromMat(
    int seq_kind,
    const CvArr* mat,
    CvContour* contour_header,
    CvSeqBlock* block
);
double cvPointPolygonTest(
    const CvArr* contour,
    CvPoint2D32f pt,
    int measure_dist
);

```

【250】

第一个函数 `cvMaxRect()` 根据输入的 2 个矩形计算，它们的最小外包矩形。

下一个实用函数 `cvBoxPoints()` 用于计算 `CvBox2D` 结构表示矩形的 4 个顶点。当然你也可以自己通过三角函数计算，不过这很令人头大，而简单调用一下这个函数则可求出。

第三个实用函数 `cvPointSeqFromMat` 从 `mat` 中初始化序列。这在你需要使用轮廓相关的函数，但是函数又不支持矩阵参数的时候使用。第一个参数用于指定点序列的类型，`seq_kind` 可以为以下类型：点集为 0；曲线为 `CV_SEQ_KIND_CURVE`；封闭曲线为 `CV_SEQ_KIND_CURVE | CV_SEQ_FLAG_CLOSED`。第二参数是输入的矩阵，该参数是连续的 1 维向量。矩阵的类型必须为 `CV_32SC2` 或 `CV_32FC2`。

下面的两个参数是指针，指针指向的内容通过该函数来填充。`contour_header` 参数对应轮廓结构，一般需要事先创建，不过由该函数负责初始化。`block` 参数同样如此，也是由该函数负责初始化^①。最后，该函数返回一个类型为 `CvSeq*` 的序列指针，指向你输入的序列头 *`contour_header`。返回值跟输入参数相同只是为了使用该函数时更方便，因为这样你就可以将该函数当作某个轮廓函数的参数使用，代码写入同一行。

① 你可能永远都不会用到 `block` 参数，它存在是因为 `cvPointSeqFromMat()` 并没有进行内存复制，相反它是通过在矩阵上构造一个虚拟的内存块。参数 `block` 用来创建该内存的一个引用，它可能在序列或轮廓内部使用。

最后一个平面几何相关的函数是 `cvPointPolygonTest()`，用于测试一个点是否在多边形的内部。如果参数 `measure_dist` 非零，函数返回值是点到多边形的最近距离。如果 `measure_dist` 为 0，函数返回 +1、-1、0，分别表示在内部、外部、在多边形边上。参数 `contour` 可以是序列，也可以是 2 通道矩阵向量。

轮廓的匹配

前面介绍了轮廓的一些基础知识，并且演示了如果在 OpenCV 中操作轮廓对象。现在我们来研究一下如何在实际应用中使用轮廓。一个跟轮廓相关的最常用到的功能是匹配两个轮廓。如果有两个轮廓，如何比较它们；或者如何比较一个轮廓和一个抽象模板。这两种情况随后都会讨论。【251】

矩

比较两个轮廓最简洁的方式是比较它们的轮廓矩。这里先简短介绍一下矩的含义。简单地说，矩是通过对轮廓上所有点进行积分运算(或者认为是求和运算)而得到的一个粗略特征。通常，我们如下定义一个轮廓的(p, q)矩：

$$m_{p,q} = \sum_{i=1}^n I(x_i, y_i) x_i^p y_i^q$$

在公式中 p 对应 x 维度上的矩， q 对应 y 维度上的矩，阶数表示对应的部分的指数。该计算是对轮廓边界上所有像素(数目为 n)进行求和。如果 p 和 q 全部为 0，那么 m_{00} 实际上对应轮廓边界上点的数目^①。

下面的函数用于计算这些轮廓矩：

```
void cvContoursMoments(  
    CvSeq* contour,  
    CvMoments* moments  
)
```

① 纯数学论者可能不同意 m_{00} 为轮廓边界长度，他们认为而应该是面积。但是因为此处我们处理的是一个轮廓，而不是一个被填充的多边形，因此多边形的边长和边界上的像素在离散的像素空间里是相同的(至少在像素空间里是跟距离相关的)。同样，也有用于计算 `IplImage` 图像的矩的函数，对于图像， m_{00} 一般对应非 0 像素的面积。

第一个参数是我们要处理的轮廓，第二个参数指向一个结构，该结构用于保存生成的结果。CvMoments 结构定义如下：

```
typedef struct CvMoments {
    // spatial moments
    double m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;
    // central moments
    double mu20, mu11, mu02, mu30, mu21, mu12, mu03;
    // m00 != 0 ? 1/sqrt(m00) : 0
    double inv_sqrt_m00;
} CvMoments;
```

在 cvContoursMoments() 函数中，只用到了 m00, m01, ..., m03 几个参数；以 mu 开头的参数在其他函数中使用。

在使用 CvMoments 结构的时候，我们可以使用以下的函数来方便地获取一个特定的矩：

```
double cvGetSpatialMoment(
    CvMoments* moments,
    Int x_order,
    int y_order
);
```

调用 cvContoursMoments() 函数会计算所有 3 阶的矩(m_{21} 和 m_{12} 会被计算，但是 m_{22} 不会被计算)。

再论矩

刚刚描述的矩计算给出了一些轮廓的简单属性，可以用来比较两个轮廓。但是在很多实际使用中，刚才的计算方法得到的矩并不是做比较时最好的参数。具体说来，经常会用到归一化的矩(因此，不同大小但是形状相同的物体会有相同的值)。同样，刚才的小节中的简单的矩依赖于所选坐标系，这意味着物体旋转后就无法正确匹配。

OpenCV 提供了计算 Hu 不变矩[Hu62]以及其他归一化矩的函数。CvMoments 结构可以用 cvMoments 或者 cvContourMoments 计算。并且，cvContourMoments 现在只是 cvMoments 的一个别名。

一个有用的小技巧是用 cvDrawContour() 描绘一幅轮廓的图像后，调用一个矩的

函数处理该图像。使用无论轮廓填充与否，你都能用同一个函数处理。

以下是 4 个相关函数的定义：

```
void cvMoments(
    const CvArr* image,
    CvMoments* moments,
    int      isBinary = 0
)
double cvGetCentralMoment(
    CvMoments* moments,
    int      x_order,
    int      y_order
)
double cvGetNormalizedCentralMoment(
    CvMoments* moments,
    int      x_order,
    int      y_order
);
void cvGetHuMoments(
    CvMoments* moments,
    CvHuMoments* HuMoments
);
```

第一个函数除了使用的是图像(而不是轮廓)作为参数，其他方面和 `cvContours-Moments()` 函数相同，另外还增加了一个参数。增加的参数 `isBinary` 如果为 `CV_TRUE`，`cvMoments` 将把图像当作二值图像处理，所有的非 0 像素都当作 1。当函数被调用的时候，所有的矩被计算(包含中心矩，请看下一段)。除了 `x` 和 `y` 的值被归一化到以 0 为均值，中心距本质上跟刚才描述的矩一样。 【253~254】

$$\mu_{pq} = \sum_{i=0}^n I(x_i, y_i) (x - x_{\text{avg}})^p (y - y_{\text{avg}})^q$$

这儿 $x_{\text{avg}} = m_{10} / m_{00}$ 且 $y_{\text{avg}} = m_{01} / m_{00}$

归一化矩和中心矩也基本相同，除了每个矩都要除以 m_{00} 的某个幂^①：

① 此处矩通过 m_{00} 的某次幂来归一化，以使得计算出的矩能够跟物体的大小无关。例如平均值是 N 个数的和除以 N ，跟求平均值的道理类似，高阶矩需要与之对应的归一化比例。

$$\eta_{p,q} = \frac{\mu_{p,q}}{m_{\infty}^{(p+q)/2+1}}$$

最后来介绍 Hu 矩，Hu 矩是归一化中心距的线性组合。之所以这样做是为了能够获取代表图像某个特征的矩函数，这些矩函数对某些变化如缩放、旋转和镜像映射（除了 h_1 ）具有不变性。

Hu 矩是从中心距中计算得到，其计算公式如下所示：

$$\begin{aligned} h_1 &= \eta_{20} + \eta_{02} \\ h_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\ h_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\ h_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\ h_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2] \\ &\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(3\eta_{21} + \eta_{03})^2 - (\eta_{21} + \eta_{03})^2] \\ h_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\ h_7 &= (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ &\quad - (\eta_{30} - \eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \end{aligned}$$

参考图 8-9 和表 8-1，我们可以直观地看到每个图像对应的 7 个 Hu 矩。通过观察可以发现当阶数变高时，Hu 矩一般会变小。对于这一点不必感到奇怪，因为根据定义，高阶 Hu 矩由多个归一化矩的高阶幂计算得到，而归一化矩都是小于 1 的，所以指数越大，计算所得的值越小。



图 8-9：五个字符的图像；观察它们的 Hu 矩可以获取跟字符相关的一些直观理解

表 8-1：在图 8-9 中所示的五个字符的 Hu 矩的值

	h_1	h_2	h_3	h_4	h_5	h_6	h_7
A	2.837e-1	1.961e-3	1.484e-2	2.265e-4	-4.152e-7	1.003e-5	-7.941e-9
I	4.578e-1	1.820e-1	0.000	0.000	0.000	0.000	0.000
O	3.971e-1	2.623e-4	4.501e-7	5.858e-7	1.529e-13	7.775e-9	-2.591e-13
M	2.465e-1	4.775e-4	7.263e-5	2.617e-6	-3.607e-11	-5.718e-8	-7.218e-24
F	3.186e-1	2.914e-2	9.397e-3	8.221e-4	3.872e-8	2.019e-5	2.285e-6

需要特别注意的是“I”，它对于 180 度旋转和镜面反射都是对称的，它的 h3 到 h7 矩都是 0；而“O”具有同样的对称特性，所有的 Hu 矩都是非零的。对于这个问题，我们留给读者去思考。您可以仔细查看字符图像，比较不同的 Hu 矩，以获得这些矩到底如何表达的直观感受。

使用 Hu 矩进行匹配

```
double cvMatchShapes(
    const void* object1,
    const void* object2,
    int method,
    double parameter = 0
);
```

很自然，使用 Hu 矩我们想要比较两个物体并且判明它们是否相似。当然，可能有很多“相似”的定义。为了使比较过程变得简单，OpenCV 的函数 `cvMatchShapes()` 允许我们简单地提供两个物体，然后计算它们的矩并根据我们提供的标准进行比较。

这些物体可以是灰度图图像或者轮廓。如果你提供了图像，`cvMatchShape()`会在对比的进程之前为你计算矩。`cvMatchShapes()`使用的方法是表 8-2 中列出的三种中的一种。【255】

表 8-2: `cvMatchShapes()`使用的匹配方法

<code>method</code> 取值	<code>cvMatchShapes()</code> 返回值
<code>CV_CONTOURS_MATCH_11</code>	$I_1(A, B) = \sum_{i=1}^7 \left \frac{1}{m_i^A} - \frac{1}{m_i^B} \right $
<code>CV_CONTOURS_MATCH_12</code>	$I_2(A, B) = \sum_{i=1}^7 m_i^A - m_i^B $
<code>CV_CONTOURS_MATCH_13</code>	$I_2(A, B) = \sum_{i=1}^7 \left \frac{m_i^A - m_i^B}{m_i^A} \right $

在表中 m_i^A 和 m_i^B 被定义为

$$m_i^A = \text{sign}(h_i^A) \cdot \log |h_i^A|$$

$$m_i^B = \text{sign}(h_i^B) \cdot \log |h_i^B|$$

在这里 h_i^A 和 h_i^B 分别是 A 和 B 的 Hu 矩。

关于对比度量标准(metric)是如何被计算的，表 8-2 中的三个常量每个都用了不同的方法。这个度量标准最终决定了 cvMatchShapes() 的返回值。最后一个参数变量现在不能使用，因此我们可以把它设成默认值 0。

等级匹配

我们经常想要匹配两个轮廓，然后用一个相似度量度来计算轮廓所有匹配的部分。使用概括参数的方法(比如矩)是相当快的，但是它们能够表达的信息却不是很多。

为了找一个更精确的相似度量度，首先考虑一下轮廓树的结构应该会有帮助。请注意，此处的轮廓树(contour tree)不是 cvFindContours() 函数返回的多个轮廓的继承描述；此处轮廓树是用来描述一个特定形状(不是多个特定形状)内各部分的等级关系。

类似于 cvFindContours() 这样的函数返回多个轮廓，轮廓树(contour tree)并不会把这些等级关系搞混，事实上，它正是对于某个特定轮廓形状的等级描述。

理解了轮廓树的创建会比较容易理解轮廓树。从一个轮廓创建一个轮廓树是从底端(叶节点)到顶端(根节点)的。首先搜索三角形突出或凹陷的形状的周边(轮廓上的每一个点都不是完全和它的相邻点共线的)。每个这样的三角形被一条线段代替，这条线段通过连接非相邻点的两点得到；因此，实际上三角形或者被削平(例如，图 8-10 的三角形 D)或者被填满(三角形 C)。每个这样的替换把轮廓的顶点减少 1，并且给轮廓树创建一个新节点。如果这样一个三角形的两侧有原始的边，那么它就是得到的轮廓树的叶子；如果一侧是已存在三角形，那么它就是那个三角形的父节点。这个过程的迭代最终把物体的外形减成一个四边形，这个四边形也被剖开；得到的两个三角形是根节点的两个子节点。

【256~257】

结果的二分树(图 8-11)最终将原始轮廓的形状信息编码。每个节点被它所对应的信息(比如三角形的大小，它的生成是被切出来还是被填进去的，这样的信息)所注释。

这些树一旦被建立，就可以很有效的对比两个轮廓^①。这个过程开始于定义两个树节点的对应关系，然后比较对应节点的特性。最后的结果就是两个树的相似度。

事实上，我们基本不需要理解这个过程。OpenCV 提供一个函数从普通的

① 一些轮廓的等级匹配的早期工作在 [Mokhtarian86] 和 [Neveu86] 中有描述，3D 匹配在 [Mokhtarian88] 中有描述。

CvContour 对象自动生成轮廓树并转化返回，还提供一个函数用来对比两个树。不幸的是，建立的轮廓树并不太鲁棒(例如，轮廓上很小的改变可能会彻底改变结果的树)。同时，最初的三角形(树的根节点)是随意选取的。因此，为了得到较好的描述需要首先使用函数 cvApproxPoly()之后将轮廓排列(运用循环移动)成最初的三角形不怎么受到旋转影响的状态。

```
CvContourTree* cvCreateContourTree(
    const CvSeq* contour,
    CvMemStorage* storage,
    double threshold
);
CvSeq* cvContourFromContourTree(
    const CvContourTree* tree,
    CvMemStorage* storage,
    CvTermCriteria criteria
);
double cvMatchContourTrees(
    const CvContourTree* tree1,
    const CvContourTree* tree2,
    int method,
    double threshold
);
```

这个代码提到了 CvTermCriteria()，该函数的细节将在第 9 章给出。现在你可以用下面的默认值(或相似的)使用 cvTermCriteria() 简单建立一个结构体。

```
CvTermCriteria termcrit = cvTermCriteria(
    CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 5, 1
);
```

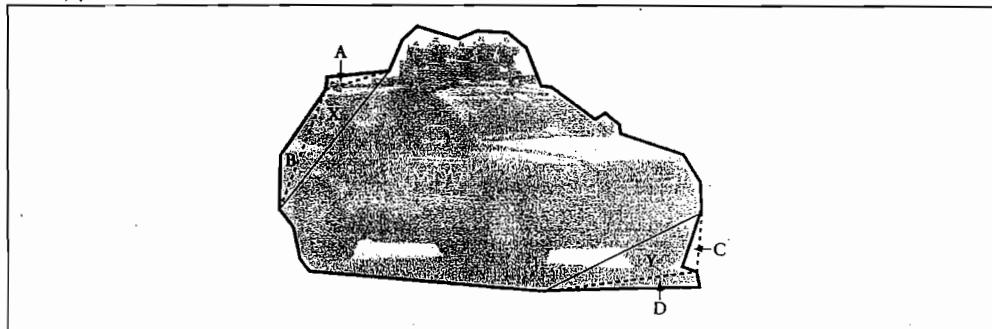


图 8-10：建立一个轮廓树：第一个回合，车的外围轮廓产生叶节点 A,B,C 和 D；第二个回合，产生 X 和 Y(X 是 A 和 B 的父节点，Y 是 C 和 D 的父节点)

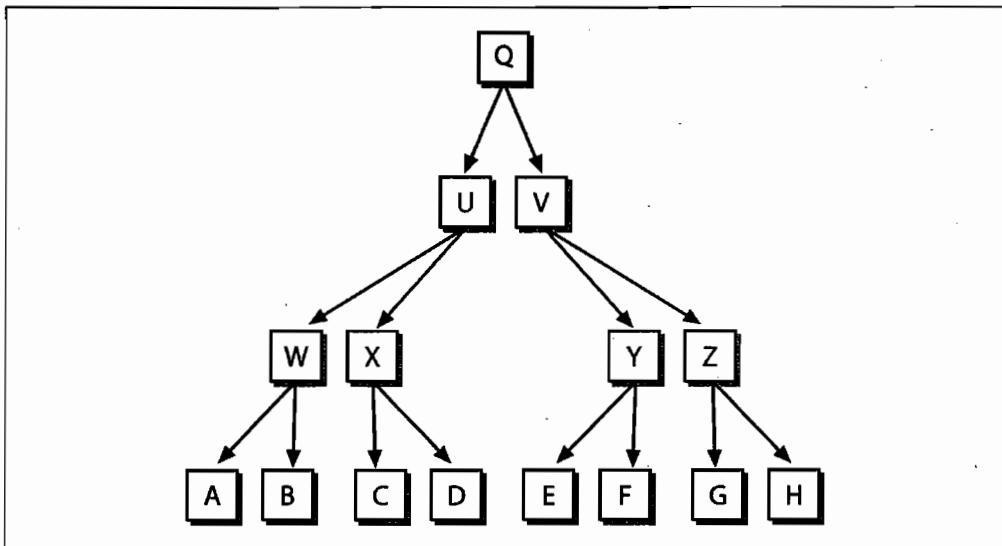


图 8-11：二分树描述的是像图 8-10 那样的轮廓

轮廓的凸包和凸缺陷

另一个理解物体形状或轮廓的有用的方法是计算一个物体的凸包(convex hull)然后计算其凸缺陷(convexity defects)[Homma85]。很多复杂物体的特性能很好的被这种缺陷表现出来。

图 8-12 用人手图举例说明了凸缺陷这一概念。手周围深色的线描画出了凸包，A 到 H 被标出的区域是凸包的各个“缺陷”。正如所看到的，这些凸度缺陷提供了手以及手状态的特征表现的方法。【258】

```
#define CV_CLOCKWISE 1
#define CV_COUNTER_CLOCKWISE 2
CvSeq* cvConvexHull2(
    const CvArr* input,
    void* hull_storage = NULL,
    int orientation = CV_CLOCKWISE,
    int return_points = 0
);
int cvCheckContourConvexity(
    const CvArr* contour
);
```

```

CvSeq* cvConvexityDefects(
    const CvArr* contour,
    const CvArr* convexhull,
    CvMemStorage* storage = NULL
);

```

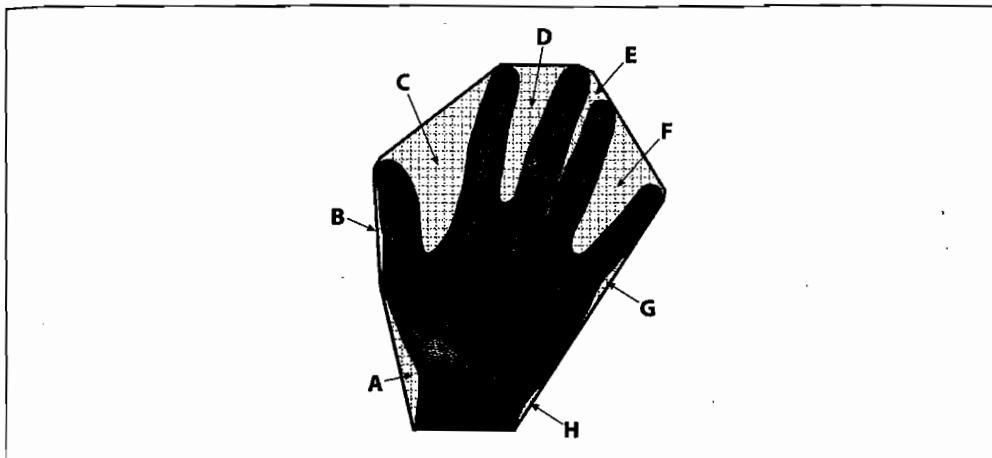


图 8-12：凸缺陷。深色的轮廓是围绕手的凸包；格子区域(A-H)是手的轮廓相对于凸包的凸缺陷

OpenCV 有三个关于凸包和凸缺陷的重要函数。第一个函数简单计算已知轮廓的凸包，第二个函数用来检查一个已知轮廓是否是凸的。第三个函数在已知轮廓是凸包的情况下计算凸缺陷。

函数 `cvConvexHull2()` 的第一个参数是点的数组，这个数组是一个 n 行 2 列的矩阵($n \times 2$)，或者是一个轮廓。如果是点矩阵，点应该是 32 位整型(CV_32SC1)或者是浮点型(CV_32FC1)。下一个参数是指向内存存储的一个指针，为结果分配内存空间。下一个参数是 CV_CLOCKWISE 或者 CV_COUNTERCLOCKWISE 中的一个，这个参数决定了程序返回的点的排列方向。最后一个参数 `returnPoints`，可以是 0 或 1。如果设置为 1，点会被存储在返回数组中。如果设置为 0，只有索引被存储在返回数组中，索引是传递给 `cvConvexHull2()` 的原始数组的索引^①。 【259 ~ 260】

这时，聪明的读者可能要问：“如果参数 `hull_storage` 是内存存储，为什么它的类型是 `void*` 而不是 `CvMemStorage*`？”问得好！这是因为很多时候作为凸包返回的点的形式，数组可能比序列更加有用。考虑到这一点，参数 `hull_storage` 的另

^① 如果输入是 `CvSeq*` 或者 `CvContour*`，则存储指向点的指针。

一个可能性是传递一个指向矩阵的指针 `CvMat*`。这种情况下，矩阵应该是一维的且和输入点的个数相同。当 `cvConvexHull2()` 被调用的时候，它会修改矩阵的头来指明当前的列数^①。

有时候，已知一个轮廓但并不知道它是否是凸的。这种情况下，我们可以调用函数 `cvCheckContourConvexity()`。这个测试简单快速^②，但是如果传递的轮廓自身有交叉的时候不会得到正确结果。

第三个函数 `cvConvexityDefects()`，计算凸缺陷并返回一个缺陷的序列。为了完成这个任务，`cvConvexityDefects()` 要求输入轮廓，凸包和内存空间，从这个内存空间来获得存放结果序列的内存。前两个参数是 `CvArr*`，和传递给 `cvConvexHull2()` 的参数 `input` 的形式相同。

```
typedef struct CvConvexityDefect {
    // point of the contour where the defect begins
    CvPoint* start;
    // point of the contour where the defect ends
    CvPoint* end;
    // point within the defect farthest from the convex hull
    CvPoint* depth_point;
    // distance between the farthest point and the convex hull
    float depth;
} CvConvexityDefect;
```

函数 `cvConvexityDefects()` 返回一个 `CvConvexityDefect` 结构体的序列，其中包括一些简单的参数用来描述凸缺陷。`start` 和 `end` 是凸包上的缺陷的起始点和终止点。`depth_point` 是缺陷中的距离凸包的边(跟该缺陷有关的凸包边)最远的点。最后一个参数 `depth` 是最远点和包的边(edge)的距离。

【260】

成对几何直方图

之前我们简要介绍过 Freeman 链码编码(FCCs)。Freeman 链码编码是对一个多边形的序列如何“移动”的描述，每个这样的移动有固定的长度和特定的方向。但是，我们并没有更多说明为什么需要用到这种描述。

-
- ① 你应该清楚为矩阵的数据分配的内存不会被重新分配，因此不要指望内存会被释放。不管怎样，既然这些是 C 的数组，只有在矩阵自身被释放以后内存才会被重新分配。
 - ② 其时间复杂度为 $O(N)$ ，只比创建凸包需要的时间复杂度 $O(N \log N)$ 快一点。

Freeman 链码编码的用处很多，但最常见的一种值得深入了解一下，因为它支持了成对几何直方图(pairwise geometrical histogram, PGH)的基本思想^①。

PGH 实际上是链码直方图(chain code histogram, CCH)的一个扩展或延伸。CCH 是一种直方图，用来统计一个轮廓的 Freeman 链码编码每一种走法的数字。这种直方图有一些良好的性质。最显著的是，将物体旋转 45 度，那么新的直方图是老直方图的循环平移(请看图 8-13)。这就提供了一个不被此类旋转影响的形状识别方法。

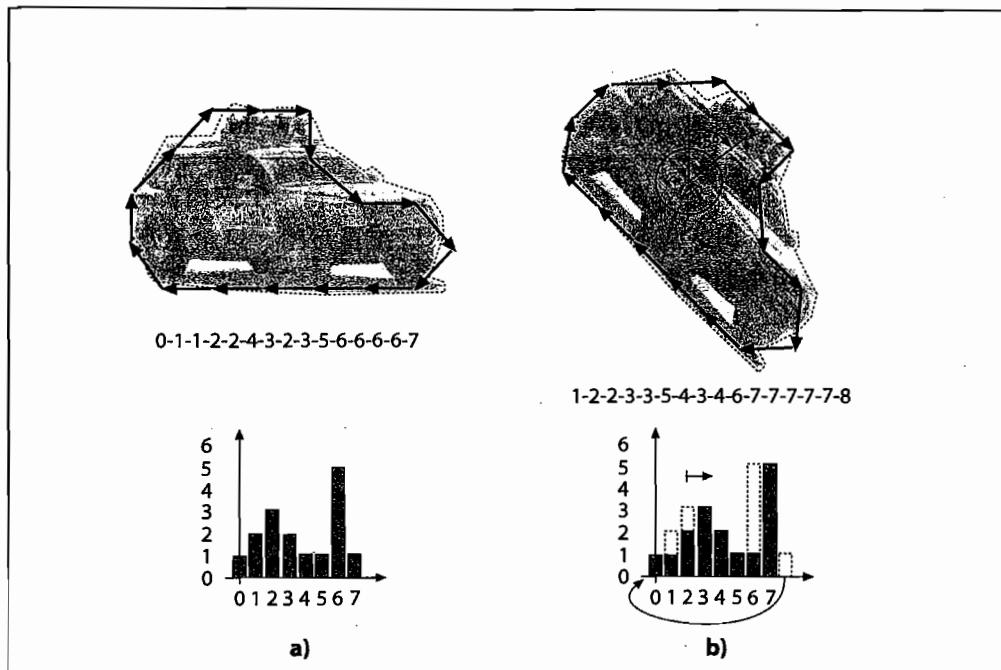


图 8-13：Freeman 链码编码对一个轮廓地描述(上)和它们的链码编码直方图(下)；当原始的轮廓 a)被顺时针旋转 45 度 b)的时候，原来的直方图向右移动一个单位便可得到的旋转后物体的链码编码直方图

【261】

PGH 的构成如下图所示(请看图 8-14)。多边形的每一个边被选择成为“基准边”。之后考虑其他的边相对于这些基础边的关系，并且计算三个值： d_{\min} , d_{\max} 和 θ 。 d_{\min} 是两条边的最小距离， d_{\max} 是最大距离， θ 是两边的夹角。PGH 是一个二维直方图，其两个维度分别是角度和距离。对于每一对边，有两个 bin，一个 bin 为 (d_{\min}, θ)

① 在 OpenCV 中实现的是 Iivarinen, Peura, Särelä, and Visa [Iivarinen97] 的方法。

另一个 b_{in} 为 (d_{max}, θ) 。对于这样的每一组边，这两个 bin 都被增长，中间值 d (d_{min} 和 d_{max} 之间的值)同样也被增长。

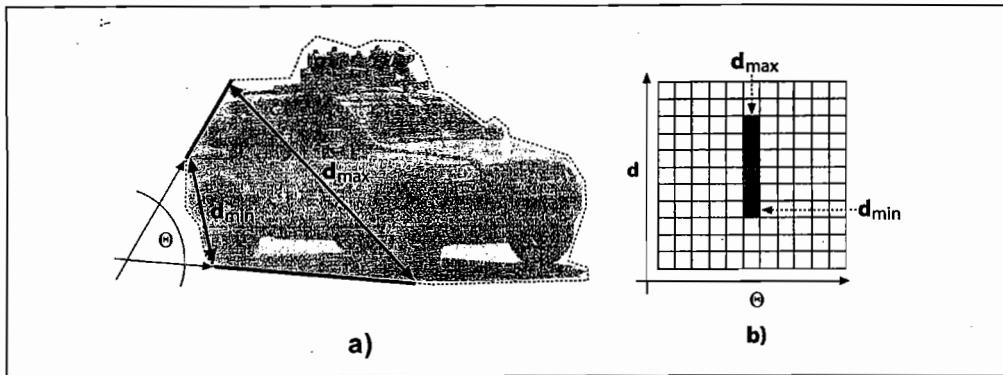


图 8-14：成对几何直方图：a)多边形上的每两个边都有一个夹角、一个最小距离和一个最大距离；b)这些数字被编码到一个二维直方图，这个直方图有旋转不变性，可以被用来匹配物体

PGH 的使用和 FCC 相似。一个重要的不同是，PGH 的描述能力更强，因此在尝试解决复杂问题的时候很有用，比如说大量的形状需要被辨识，并且/或者有很多背景噪声的时候。用来计算 PGH 的函数是

```
void cvCalcPGH(
    const CvSeq* contour,
    CvHistogram* hist
);
```

在这里轮廓可以包含整数值的点的坐标；当然直方图必须是二维的。

【262】

练习

1. 不考虑图像的噪声，如果我们缩放图像，IPAN 算法返回的结果是否相同？旋转图像呢？
 - a. 给出答案，并解释为什么是这样。
 - b. 实际动手试一下！使用 PowerPoint 或者其他类似的程序，在黑色背景上画一个白色的物体，然后转成图像并保存。缩放该图像几次，并通过多次旋转改变位置，保存每次的结果作为测试图像。然后用 OpenCV 读出图像，

转换为灰度图像并二值化，然后查找轮廓。然后用 `cvFindDominantPoints` 查找关键点，检查每个图像得到的结果是否相同。

2. 在一个由 N 个点组成的闭合轮廓中找出极端的点(比如，两个相距最远的点)，这可以通过比较每个点和其他各点的距离得到。
 - a. 这种算法的复杂度如何？
 - b. 解释一下如何才能算得更快。
3. 用 `CvSeq` 的函数创建一个圆，这个圆用点序列来表示。
4. 适合一个 4×4 图像的最大闭合轮廓长度是多少？面积是多少？
5. 使用 PowerPoint 或者其他类似程序，在黑色背景上画一个半径 20 的白色的圆(圆的周长大约是 $2\pi \cdot 20 \approx 126.7$)。把这个图作为图像保存。
 - a. 读入图像，转换成灰度图，使用域值进行二值化，检测轮廓。轮廓的长度是多少？和刚才计算出的长度一样吗(内环)？
 - b. 将 126.7 作为轮廓的基本长度，分别使用 $1/90$ 、 $1/66$ 、 $1/11$ 和 $1/10$ 作为精度参数，使用 `cvApproxPoly()` 逼近，然后计算轮廓长度，并画出结果。
6. 使用练习 5 画出的圆，用如下方法查看 `cvFindDominantPoints()` 得到的结果。
 - a. 改变 d_{min} 和 d_{max} 距离并画出结果。
 - b. 接着改变相邻距离并描述结果的改变。
 - c. 最后改变最大角度的域值并计算结果。
7. 亚像素级角点检测。用 PowerPoint 或类似的绘图工具在黑色背景上创建一个白色的拐角，使得这个角正好处在整数值的坐标上，然后保存这个图像并用 OpenCV 打开。
 - a. 找出并输出拐角的确切坐标。
 - b. 改变原始图像：在拐角上画一个小的黑色圆形遮掉原来的角。保存并打开此图，找出这个角亚像素级的位置。和刚才相同吗？为什么？
8. 假如我们建立一个瓶子的探测器并且希望创建一个“瓶子”的特征。我们有很多容易分割并能找到轮廓的瓶子的图像，但是瓶子被旋转并且大小被改变。我们可以画出轮廓并且找出 Hu 矩，从而得到不变瓶子特性的向量。前面提到的这些条件都满足。请问为了提取特征，我们应该画出被填充的轮廓还是仅仅是轮廓

轮廓？解释你的答案。

9. 练习 8 中，当使用 `cvMoments()` 抽出瓶子的轮廓矩时，如何设置 `isBinary`？解释你的答案。
10. 对前面提到的用于提取 Hu 矩的字符图像，然后通过旋转不同的角度、缩小或放大图像、或者旋转缩放相结合来生成不同的图像。请问哪些 Hu 矩对旋转稳定，那些对缩放稳定，以及哪些对这二者都稳定。
11. 用 PowerPoint 或其他绘图程序绘制一个形状，然后保存图像。然后再将图像缩放并保存，旋转并保存，缩放和旋转后保存。用 `cvMatchContourTrees()` 和 `cvConvexityDefects()` 处理这些图像，查看哪个匹配效果最好，为什么？

第 9 章

图像局部与分割

局部与分割

本章重点讲述如何从图像中将目标或部分目标分割出来。这样做的原因很明显，比如在视频安全应用中，摄像机经常观测一个不变的背景，而实际上我们对这些背景并不感兴趣。我们所感兴趣的只是当行人或车辆进入场景时或者当某些东西被遗留在场景中时。我们希望分离出这些事件而忽略没有任何事件发生的时间段。

除了从图像中分割出前景目标之外，在很多情况下我们也希望将感兴趣的目标区域分割出来，比如将一个人的脸或手分割出来。我们可以把图像预处理成有意义的超像素(super pixel)所组成的诸如包含类似四肢、头发、脸、躯干、树叶、湖泊、道路、草坪等物体的图像区域。这些超像素的使用节省了计算量。比如，对一幅图像做目标分类器处理的时候，我们只需要在包含每个超像素的一个区域进行搜索。这样可能只需要跟踪这些大的区域，而不是区域中的每一个像素点。

在第 5 章介绍图像处理时我们已经讨论了几个图像分割的算法。这些程序涵盖了图像形态学、种子填充法、阈值算法以及金字塔分割法等方面。本章将研究其他用于查找、填充和分离一幅图像中的目标以及部分目标物体的算法。先从已知背景的场景中分割出前景目标开始。这些背景的建模函数并没有内置在 OpenCV 函数中，更确切地说，它们仅仅用来自说明如何利用 OpenCV 函数实现更加复杂的例子。

背景减除

由于背景减除简单而且摄像机在很多情况下是固定的，在视频安全应用领域，背景减除(又名背景差分)也许是最基本的图像处理操作。Toyama, Krumm, Brumitt 和 Meyers 对背景提取做了很好的概述并与许多技术进行了对比[Toyama99]。要实现背景减除，我们必须首先“学习”背景模型。

【265】

一旦背景模型建立，将背景模型和当前的图像进行比较，然后减去这些已知的背景信息，则剩下的目标物大致就是所求的前景目标了。

当然，“背景”在不同的应用场合下是一个很难定义的问题。例如，若正在观测一条高速公路，那么或许平均流动的车流应该被认为是背景。通常情况下，背景被认为是在任何所感兴趣的时期内，场景中保持静止或周期运动的目标。整个场景可以包含随时间变动的单元，比如竖立在原地但却从早到晚摇曳在风中的树。两个容易碰到的常见但却不同的环境类别是室内和室外场景。如果有工具在这两种环境中对我们有所帮助，我们将很感兴趣。首先，我们将讨论经典背景模型的不足，再转向阐述更高级的场景模型。接下来，我们提出一个快速的背景建模方法，该方法对于光照条件变化不大的室内的静止背景的场景效果很好。然后将介绍 codebook 方法，该方法虽然速度稍慢，但在室内外环境下都能工作得很好；也能适应周期性的运动(比如树在风中摇曳)以及灯光缓慢变化或有规律的变化，并且对偶尔有前景目标移动的背景学习有很好的适应性。我们将随后在清除前景物体检测的内容中另行讨论连通物体(首次见于第 5 章)的相关内容。最后将会比较快速背景建模方法和 codebook 背景方法。

背景减除的缺点

虽然背景建模方法在简单的场景中能够达到较好的效果，但该方法受累于一个不常成立的假设：所有像素点是独立的。我们所描述的这种建模方法在计算像素变化时并没有考虑它相邻的像素。为了考虑到周围的像素，我们需要建立一个多元模型，它把基本的像素独立模型扩展为包含了相邻像素的亮度的基本场景。在这种情况下，我们用相邻像素的亮度来区别相邻像素值的相对明暗。然后对单个像素的两种模型进行有效的学习：一个其周围像素是明亮的，另一个则是周围像素是暗淡的。这样，我们就有了一个考虑到了周围因素的模型。但是，这样需要消耗两倍的内存和更多的计算量，因为当周围的像素是亮或暗的时，需要用不同的亮度值来表示，而且还要两倍的数据来填充这个双状态模型。我们将这种“高低”关系的思路归纳到单个像素及其周围像素亮度的多维直方图中，并且可以通过一系列的操作步骤使

该思路更加复杂。当然，这种更完整的空间和时间模型需要更多的内存、收集数据样本更多的以及更多的计算资源。

由于这些额外的开销，通常会避免使用复杂的模型。当像素独立假设不成立情况下，我们可以更有效地把精力投入到清除那些错误的检测结果中。清除采用图像处理的方式(主要是 cvErode()，cvDilate() 和 cvFloodFill() 等)去掉那些孤立的像素。在前面的章节中(第 5 章)我们已经讨论过如何在噪声数据中寻找大块紧凑连通域的程序^①。本章我们将再次使用连通域，就目前而言，我们将把讨论的方法严格限制在像素变化独立的假设基础上。

场景建模

如何定义背景和前景呢？如果在监视一个停车场时，一辆汽车开了进来，那么这辆车就是一个新的前景目标，但是，这辆轿车一直应该是前景吗？对于场景中物体移动后的情况呢？这里将显示两个地方为前景：一是物体移动到的位置，另一个则是物体离开后留下的“空洞”，如何分辨出两者的不同？再者，留下的“空洞”能在前景状态保持多长时间？如果给一个黑暗屋子建模，突然有人打开了一盏灯，那么整个屋子都变成前景了吗？为了回答这些问题，我们需要更高级的场景建模，在此模型中，要对前景状态和背景状态定义多重指标，以时间为基准将不变的前景模块缓慢转换为背景模块。当场景完全发生变化时我们还必须检测并建立一个新的模型。

通常，一个场景模型可能包含许多层次，从“新的前景”到旧的前景再到背景。还可能有一些运动检测，这样，当一个目标移动时，我们可以识别其“真的”的前景(新位置)和“假的”的前景(其旧的位置，“空洞”)。

这样，一个新的前景目标就会放进“新前景”目标级别，标识为一个真目标或一个空洞。在没有任何前景物体的地方，我们将继续更新我们的背景模型。如果一个前景物体在给定的时间内没有发生移动，就将它降级为“旧的前景”，这里它的像素统计特性还将暂时学习直到它的学习模型融合进学习背景模型之中。

对于像在屋里打开一盏灯时的全局改变的检测，我们可以使用全局帧差来计算。比如，一次有许多像素发生变化，我们就能够将它归类为全局的变化而不是局部的变化，而后转用一种适用于这种新情况的模型。

① 这里，我们使用数学中的“紧凑”定义，与大小无关。

像素片段

在转到为像素变化建模之前，先要对图像中的像素点在一段时间内如何变化有个概念。考虑一个摄像机从窗口拍摄一棵被风吹拂着的树的场景。图 9-1 显示了图像中给定线段的像素在 60 帧图像中的变化。我们希望给这种波动建立模型。然而在建模之前，我们作一个小小的离题讨论，如何对这条线进行采样，通常这是创建特征和调试都很有用的手段。

【267】

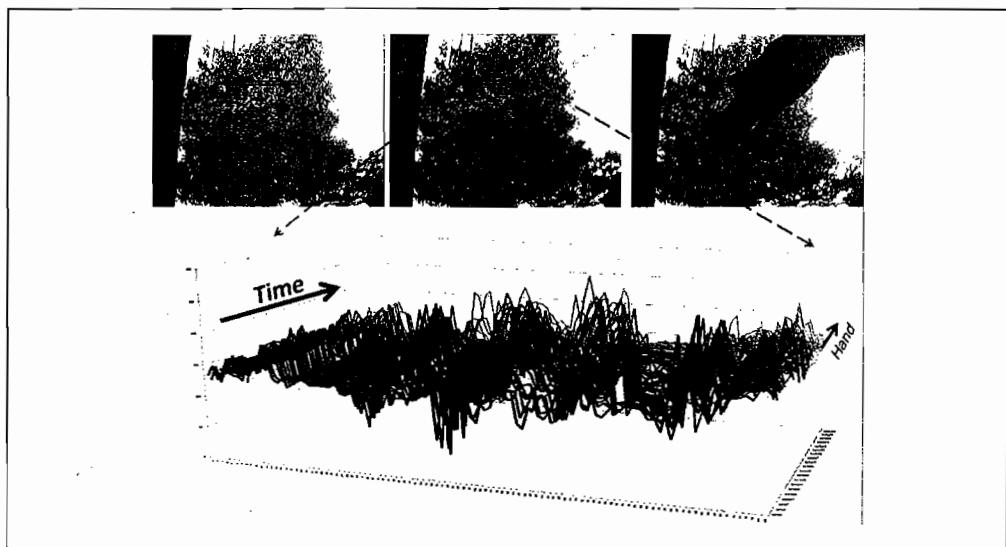


图 9-1：树在风中移动的场景中一条线上的像素在 60 帧内的波动。一些黑色区域(左上)很稳定，而移动的树枝(中上)变化范围很大

OpenCV 有这样的函数，它能够很容易对任意直线上的像素进行采样。线采样函数是 `cvInitLineIterator()` 和 `CV_NEXT_LINE_POINT()`，前者原型如下：

```
int cvInitLineIterator(
    const CvArr*      image,
    CvPoint          pt1,
    CvPoint          pt2,
    CvLineIterator*   line_iterator,
    int              connectivity = 8,
    int              left_to_right = 0
);
```

输入参数 `image` 可以是任意数据类型和任何通道数，点 `pt1` 和 `pt2` 是线段的两个

端点。迭代器 line_iterator 说明直线上两个像素之间的移动步长。如果图像是多通道的，每次调用 CV_NEXT_LINE_POINT() 函数都使 line_iterator 指向下一个像素。每个通道的像素值可同时用 line_iterator.ptr[0], line_iterator.ptr[1] 等依次得到。连通性可以是 4(直线可以是沿着右、左、上、下方向)和 8(再增加沿着对角线方向)。最后，如果把 left_to_right 设置为 0(false)，line_iterator 将从 pt1 扫描到 pt2；否则，它将从最左边的点扫描到最右边的点^①。函数 cvInitLineIterator() 将返回直线上迭代的点的个数。伴随宏(CV_NEXT_LINE_POINT (line_iterator))使迭代器从一个像素到另一个像素。

【268~269】

让我们花点时间看一下这种方法是如何从一个文件提取数据的(例 9-1)，接下来，我们再检查下图 9-1 中来自视频文件的结果数据。

例 9-1：从视频的一行中读出所有像素的 RGB 值，收集这些数值并将其分成三个文件

```
// STORE TO DISK A LINE SEGMENT OF BGR PIXELS FROM pt1 to pt2
//
CvCapture* capture = cvCreateFileCapture( argv[1] );
int max_buffer;
IplImage* rawImage;
int r[10000],g[10000],b[10000];
CvLineIterator iterator;

FILE *fptrib = fopen("blines.csv","w"); // Store the data here

FILE *fptrgb = fopen("glines.csv","w"); // for each color channel
FILE *fprr = fopen("rlines.csv","w");

// MAIN PROCESSING LOOP:
//
for(;;){
    if( !cvGrabFrame( capture ) )
        break;
    rawImage = cvRetrieveFrame( capture );
    max_buffer = cvInitLineIterator(rawImage,pt1,pt2,&iterator,8,0);
```

① left_to_right 标识符的引入因为从 pt1 到 pt2 的不连续线段并不完全与从 pt2 到 pt1 的相吻合。因此，设置标识位为用户提供一致的光栅，不管 pt1 和 pt2 的顺序如何。

```

for(int j=0; j<max_buffer; j++){

    fprintf(fptrb,"%d,", iterator.ptr[0]); //Write blue value
    fprintf(fptrg,"%d,", iterator.ptr[1]); //green
    fprintf(fptrr,"%d,", iterator.ptr[2]); //red

    iterator.ptr[2] = 255; //Mark this sample in red

    CV_NEXT_LINE_POINT(iterator); //Step to the next pixel
}

// OUTPUT THE DATA IN ROWS:
//
fprintf(fptrb,"/n");fprintf(fptrg,"/n");fprintf(fptrr,"/n");
}

// CLEAN UP:
//
fclose(fptrb); fclose(fptrg); fclose(fptrr);
cvReleaseCapture( &capture );

```

也可以用下面的函数更轻松地对直线采样：

```

int cvSampleLine(
    const CvArr*    image,
    CvPoint         pt1,
    CvPoint         pt2,
    void*           buffer,
    int             connectivity = 8
);

```

【269~270】

该函数只是函数 `cvInitLineIterator()` 和宏 `CV_NEXT_LINE_POINT (line_iterator)` 的简单封装。从 `pt1` 到 `pt2` 取样，之后，传递给它一个指针，指向正确类型和长度为 $N_{\text{channels}} \times \max(|pt1_x - pt2_x| + 1, |pt1_y - pt2_y| + 1)$ 的缓冲区。正如 `line_iterator` 一样，在转向下一个像素之前，`cvSampleLine()` 会访问多通道图像的每一个像素通道。函数返回缓冲区中的实际元素的数目。

我们现在转到图 9-1 中像素波动的一些建模方法。当我们从简单的模型向越来越复杂的模型过程中，将主要关注那些能实时运行且内存消耗可接受的模型上。

帧差

最简单的背景减除方法就是用一帧减去另一帧(也可能是后几帧), 然后将足够大的差别标为前景。这种方法往往能捕捉运动目标的边缘。简而言之, 假如我们有三个单通道的图像: frameTime1, frameTime2 和 frameForeground。图像 frameTime1 用上一时刻的灰度图像填充, frameTime2 用当前灰度图像填充。则用下面的代码检测图像 frameForeground 中前景差别的幅值(绝对值):

```
cvAbsDiff(
    frameTime1,
    frameTime2,
    frameForeground
);
```

由于像素值总会受到噪声和波动的影响, 我们应该忽略(将结果设为 0)很小的差异(小于 15), 标识其余的作为较大的差别(将结果设为 255)

```
cvThreshold(
    frameForeground,
    frameForeground,
    15,
    255,
    CV_THRESH_BINARY
);
```

在图像 frameForeground 中候选的前景目标值为 255, 背景值为 0. 我们需要清除前面讨论过的噪声; 可以调用 cvErode() 函数或者用连通域去噪。对于彩色图像, 我们用相同的代码对每个颜色通道分别处理, 之后再调用 cvOr() 函数将所有通道拼接在一起。如果不仅仅是检测运动区域, 这种方法有些简单。对更有效的背景模型, 我们需要保留场景中像素的均值和平均差等统计特征。在后面的“快速测试”小节的图 9-5 和图 9-6 中, 可以看到作帧差的例子。

【270】

平均背景法

平均背景法的基本思路是计算每个像素的平均值和标准差(或相似的, 但计算速度更快的平均差值)作为它的背景模型。考虑图 9-1 的像素直线。我们可以通过视频中的平均值和平均差来描述每一个像素的变化(图 9-2), 而不是对每一帧图像绘出一个像素值序列(像我们在那幅图像中做的那样)。在同一个视频中, 一个前景目标

(实际上，可能是一只手)经过摄像机的前面。这个前景目标显然不如背景中的天空和树明亮，手的亮度在图 9-2 中也表示出来了。

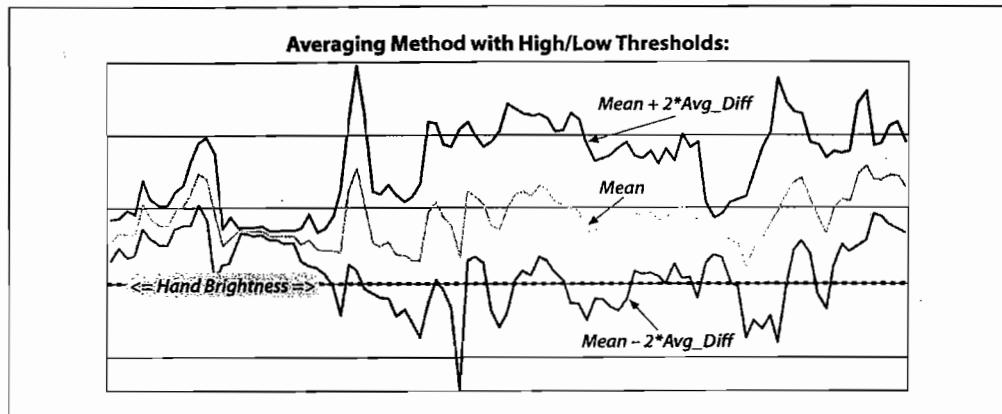


图 9-2：表示的是一组平均差分。一个目标(一只手)从摄像机前掠过，目标亮度相对较暗，图中表示了它的亮度

平均背景法使用四个 OpenCV 函数：`cvAcc()`，累积图像；`cvAbsDiff()`，计算一定时间内的每帧图像之差；`cvInRange()`，将图像分割成前景区域和背景区域(背景模型已经学习的情况下)；函数 `cvOr()`，将不同的彩色通道图像中合成为一个掩模图像。由于这个例子代码比较长，我们将它分开成几块，对每一块分别讨论。

首先，我们为需要的不同临时图像和统计属性图像创建指针。这样有助于根据图像的不同类型对以后使用的图像指针排序。

```
//Global storage
//
//Float, 3-channel images
//
IplImage *IavgF,*IdiffF, *IprevF, *IhiF, *IlowF;

IplImage *Iscratch,*Iscratch2;

//Float, 1-channel images
//
IplImage *Igray1,*Igray2, *Igray3;
IplImage *Ilow1, *Ilow2, *Ilow3;
IplImage *Ihi1, *Ihi2, *Ihi3;

// Byte, 1-channel image
```

```
//  
IplImage *Imaskt;  
  
//Counts number of images learned for averaging later.  
//  
float Icount;
```

【271~272】

接下来，我们创建一个函数来给需要的所有临时图像分配内存。为了方便，我们传递一幅图像(来自视频)作为大小参考来分配临时图像。

```
// I is just a sample image for allocation purposes  
// (passed in for sizing)  
//  
void AllocateImages( IplImage* I ){  
  
    CvSize sz = cvGetSize( I );  
  
    IavgF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IdiffF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IprevF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IhiF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IlowF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    Ilowl = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ilow2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ilow3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ihi1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ihi2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ihi3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    cvZero( IavgF );  
    cvZero( IdiffF );  
    cvZero( IprevF );  
    cvZero( IhiF );  
    cvZero( IlowF );  
    Icount = 0.00001; //Protect against divide by zero  
  
    Iscratch = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    Iscratch2 = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    Igray1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Igray2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Igray3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Imaskt = cvCreateImage( sz, IPL_DEPTH_8U, 1 );
```

```
    cvZero( Iscratch );
    cvZero( Iscratch2 );
}
```

【272】

在接下来的代码片段中，我们学习累积背景图像和每一帧图像差值的绝对值(一个计算更快的学习图像像素的标准偏差的替代)^①。这通常需要 30 至 1000 帧，有时每秒几帧或者有时需要所有有价值的帧图像。函数调用需要通道为 3，深度为 8 的彩色图像。

```
// Learn the background statistics for one more frame
// I is a color sample of the background, 3-channel, 8u
//
void accumulateBackground( IplImage *I ){

    static int first = 1;                      // nb. Not thread safe
    cvCvtScale( I, Iscratch, 1, 0 );           // convert to float
    if( !first ){
        cvAcc( Iscratch, IavgF );
        cvAbsDiff( Iscratch, IprevF, Iscratch2 );
        cvAcc( Iscratch2, IdiffF );
        Icount += 1.0;
    }
    first = 0;
    cvCopy( Iscratch, IprevF );
}

}
```

我们首先用函数 `cvCvtScale()` 将原始的每通道 8 位，3 颜色通道的彩色图像转换成一个浮点型的 3 通道图像。之后我们积累原始的浮点图像为 `IavgF`。接下来，我们用函数 `cvAbsDiff()` 计算每帧图像之间的绝对差图像，并将其积累为图像 `IdiffF`。每次积累图像之后，增加图像计数器 `Icount` 的值，该计数器是一个全局变量用于接下来计算平均值。

一旦我们积累了足够多的帧图像之后，就将其转化成为一个背景的统计模型。这就

① 注意，我们用“替代”(proxy)这个词。平均差分并不等价于算术上的标准差，但是，在这里它会产生相似的结果。平均差分的优势在于它比标准差的计算速度稍快。仅需要对例子中的代码稍作改变，就可以用标准差代替来比较结果的好坏；我们将在后面对此作更详细的讨论。

是说，计算每一个像素的均值和方差观测(平均绝对差分)。

```
void createModelsFromStats() {  
  
    cvConvertScale( IavgF, IavgF, (double)(1.0/Icount) );  
    cvConvertScale( IdiffF, IdiffF, (double)(1.0/Icount) );  
  
    //Make sure diff is always something  
    //  
    cvAddS( IdiffF, cvScalar( 1.0, 1.0, 1.0 ), IdiffF );  
    setHighThreshold( 7.0 );  
    setLowThreshold( 6.0 );  
}
```

【273】

在这段代码中，函数 `cvConvertScale()` 通过除以输入图像累积的数目计算平均原始图像和绝对差分图像。预防起见，我们确保平均差分图像的值最小是 1；当计算前景和背景阈值以及避免前景阈值和背景阈值相等而出现的退化情况时，我们要缩放这个因素。

函数 `setHighThreshold()` 和 `setLowThreshold()` 都是基于每一帧图像平均绝对差设置阈值的有效函数。函数 `setHighThreshold(7.0)` 固定一个阈值，使得对于每一帧图像的绝对差大于平均值 7 倍的像素都认为是前景；同样的，函数 `setLowThreshold(6.0)` 设置一个阈值，认为每一帧图像的绝对差小于平均值 6 倍的像素认为是前景。像素平均值范围内，认为目标为背景，阈值函数如下：

```
void setHighThreshold( float scale )  
{  
    cvConvertScale( IdiffF, Iscratch, scale );  
    cvAdd( Iscratch, IavgF, IhiF );  
    cvSplit( IhiF, Ihi1, Ihi2, Ihi3, 0 );  
}  
  
void setLowThreshold( float scale )  
{  
    cvConvertScale( IdiffF, Iscratch, scale );  
    cvSub( IavgF, Iscratch, IlowF );  
    cvSplit( IlowF, Ilow1, Ilow2, Ilow3, 0 );  
}
```

再者，使用函数 `setLowThreshold()` 和函数 `setHighThreshold()` 时，我们用函

数 cvConvertScale()乘以预先设定的值来增加或减小与 IavgF 相关的范围。这个操作通过函数 cvSplit()为图像的每个通道设置 IhiF 和 IlowF 的范围。

一旦我们有了自己的背景模型，同时给出了高、低阈值，我们就能用它将图像分割成前景(不能被背景模型“解释”的图像部分)和背景(在背景模型中，任何在高低阈值之间的图像部分)。图像分割通过调用下面的函数来完成。

```
// Create a binary: 0,255 mask where 255 means foreground pixel
// I      Input image, 3-channel, 8u
// Imask Mask image to be created, 1-channel 8u
//
void backgroundDiff(
    IplImage *I,
    IplImage *Imask
) {
    cvCvtScale(I,Iscratch,1,0); // To float;
    cvSplit( Iscratch, Igray1,Igray2,Igray3, 0 );

    //Channel 1
    //
    cvInRange(Igray1,Ilow1,Ihi1,Imask);

    //Channel 2
    //
    cvInRange(Igray2,Ilow2,Ihi2,Imaskt);
    cvOr(Imask,Imaskt,Imask);

    //Channel 3
    //
    cvInRange(Igray3,Ilow3,Ihi3,Imaskt);
    cvOr(Imask,Imaskt,Imask)

    //Finally, invert the results
    //
    cvSubRS( Imask, 255, Imask);
}
```

【274~275】

函数首先通过调用函数 cvCvtScale()将输入图像 I(用于分割的图像)转换成浮点型的图像。之后调用函数 cvSplit()将 3 通道图像分解为单通道图像。最后通过函数 cvInRange()检查这些单通道图像是否在平均背景像素的高低阈值之间，该函

数将 8 位深度的灰度图像 `Imaskt` 中在背景范围内的像素设为最大值(255)，否则设为 0。对于每一颜色通道，理论上，我们都能将分割结果转变成掩模图像 `Imask`，在任何颜色通道中非常大的差别都可认为是前景像素。最后，利用函数 `cvSubRS()` 将其转化为 `Imask` 图像，因为前景的颜色值在背景阈值范围之外。掩模图像就是函数的输出值。

完成背景建模后，我们需要将内存释放。

```
void DeallocateImages()
{
    cvReleaseImage( &IavgF );
    cvReleaseImage( &IdiffF );
    cvReleaseImage( &IprevF );
    cvReleaseImage( &IhiF );
    cvReleaseImage( &IlowF );
    cvReleaseImage( &Ilow1 );
    cvReleaseImage( &Ilow2 );
    cvReleaseImage( &Ilow3 );
    cvReleaseImage( &Ihi1 );
    cvReleaseImage( &Ihi2 );
    cvReleaseImage( &Ihi3 );
    cvReleaseImage( &Iscratch );
    cvReleaseImage( &Iscratch2 );
    cvReleaseImage( &Igray1 );
    cvReleaseImage( &Igray2 );
    cvReleaseImage( &Igray3 );
    cvReleaseImage( &Imaskt );
}
```

我们刚才已经介绍了一种学习背景场景和分割前景目标的简单方法。这种方法只能用于背景场景中不包含运动的部分(比如摆动的窗帘和在风中摇曳的树)。而且，这种方法还要求光线保持不变(如在室内静止的场景)。你可以用后面的图 9-5 来验证这种平均方法的性能。

【275~276】

累积均值、方差和协方差

刚刚描述的平均背景法利用一个累积函数 `cvAcc()`。它只是所有一组用于累计图像的函数，如平方图像，乘图像或者平均图像操作等中的一个函数。通过这些操作，我们可以计算得到整个场景或部分场景的基本统计特性(均值，方差和协方差)。在

这一节中，将要看到本组函数中的其他一些函数。

任何给定函数中的图像必须有相同的宽度和高度。对于每一个函数，输入图像命名为 `image`, `image1` 或 `image2`, 它们可以是 8 位深度的单通道或 3 通道图像，也可以是浮点型(32F)的图像数组。输出的累积图像命名为 `sum`, `s1sum` 或 `acc`, 可能是单精度的数组(32F)，也可以是双精度的数组(64F)。在累积函数中，掩模图像(如果存在)的处理严格限制操作在掩模像素不为 0 的区域。

均值漂移值 通过大量图像计算每个像素的均值的最简单的方法就是调用函数 `cvAcc()` 把它们加起来再除以图像总数来获得均值。

```
void cvAcc(
    const CvArr* image,
    CvArr* sum,
    const CvArr* mask = NULL
);
```

另外也可以选择均值漂移。

```
void cvRunningAvg(
    const CvArr* image,
    CvArr* acc,
    double alpha,
    const CvArr* mask = NULL
);
```

均值漂移值由下式给出：

$$acc(x, y) = (1 - \alpha) \cdot acc(x, y) + \alpha \cdot image(x, y) \quad \text{if } mask(x, y) \neq 0$$

对于常量 `a`, 均值漂移值并不等于利用函数 `cvAcc()` 得到的和。为了说明这个，简单地考虑一种情况，就是假设把三个数(2,3,4)相加，并设 `a` 为 0.5. 如果用函数 `cvAcc()` 来累积，得到的结果是 9，均值为 3. 如果我们用函数 `cvRunningAverage()` 来累积，首先得到的和为 $0.5 \times 2 + 0.5 \times 3 = 2.5$ ，接下来加上第三项，可得 $0.5 \times 2.5 + 0.5 \times 4 = 3.25$ 。第二个数字大的原因是最近值给了较大的权值。因此这样的均值漂移又被称作跟踪器。因为前一帧图像褪色的影响，参数 `a` 本质上是设置所需的时间。

【276】

计算方差 我们可以累积平方图像，这将有助于我们快速计算单个像素的方差。

```
void cvSquareAcc(
    const CvArr* image,
```

```

    CvArr* sqsum,
    const CvArr* mask = NULL
);

```

想一下统计的最后一节，方差定义如下：

$$\sigma^2 = \frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^2$$

其中 \bar{x} 是 N 个样本 x 的均值，该公式的问题在于计算 \bar{x} 时遍历一次整幅图像，计算 σ^2 时需要再次遍历整幅图像。下面的一个简单代数公式可以实现相同功能：

$$\sigma^2 = \left(\frac{1}{N} \sum_{i=0}^{N-1} x_i^2 \right) - \left(\frac{1}{N} \sum_{i=0}^{N-1} x_i \right)^2$$

使用这种形式，仅需一次遍历图像就能够累积出像素值和它们的平方值。单个像素的方差正好是平方的均值减去均值的平方。

计算协方差 我们可以通过选择一个特定的时间间隔来观测图像是怎么变化的，然后用当前图像乘以和特定时间间隔相对应的图像。用函数 `cvMultiplyAcc()` 来实现两幅图像之间的像素相乘，之后将结果与 `acc` 累加。

```

void cvMultiplyAcc(
    const CvArr* image1,
    const CvArr* image2,
    CvArr* acc,
    const CvArr* mask = NULL
);

```

对于协方差的计算，有一个和方差相似的公式。该式子也是只需要遍历一次图像，因为它是从标准形式经数学推导而来，所以，不需要两次遍历图像。

$$\text{Cov}(x, y) = \left(\frac{1}{N} \sum_{i=0}^{N-1} (x_i y_i) \right) - \left(\frac{1}{N} \sum_{i=0}^{N-1} x_i \right) \left(\frac{1}{N} \sum_{j=0}^{N-1} y_j \right)$$

这里 x 表示 t 时刻的图像， y 表示 $t-d$ 时刻的图像， d 是时间间隔。

【277】

我们可以用这里描述的累积函数创建各种基于统计学的背景模型。文献中有很多和我们的例子类似的基本模型的变型。你可能会发现，在应用过程中，可能更倾向于将这种最简单的模型扩展到稍微专业的版本。比如一个常用的提高是设置阈值以适应某些观测的全局状态变化。

高级背景模型

很多背景场景都包含复杂的运动目标，诸如，摇曳在风中的树，转动的风扇，摆动的窗帘等等。通常这样的场景中还包含光线的变化，比如，云彩掠过，门窗中照进来不同的光线。

解决这种问题的较好方法是得到每个像素或一组像素的时间序列模型。这种模型能很好地处理时间起伏，缺点是需要消耗大量的内存[Toyama99]。如果我们使用频率为两秒 30Hz 的输入图像序列，就意味着对于每个像素都需要 60 个样本。每个像素的结果模型都会被学习的 60 种不同的适合权值进行编码。常常我们需要获取比 2 秒更长时间的背景统计，这就意味着该方法在现有硬件设施下显得不实际。

为了获得与自适应滤波相当接近的性能，我们从视频压缩技术中得到灵感，我们试图形成一个 codebook(编码本)^①以描绘背景中感兴趣的状态^②。最简单的方法就是将一个像素现在的观测值和先前的观测值作比较。如果两个值很接近，它被建模为在那种颜色下的扰动。如果两个值不接近，它可以产生与该像素相关的一组色彩。结果可以想像为一束漂浮在 RGB 空间的斑点，每一个斑点代表一个考虑到背景的分离的体积。

实际应用中，选择 RGB 空间模型并不是最优的。选择轴与亮度联系在一起的颜色空间效果更好，比如 YUV 空间(YUV 是最通常的选择，选择 HSV 空间也行，V 实际上是亮度)。原因是，从经验角度看绝大部分背景中的变化倾向于沿着亮度轴，而不是颜色轴。

下一个细节是如何给“blobs”建模。我们与以前的简单模型有相同的选择。比如，将 blob 的模型建立为包含均值和方差的高斯分布类。最简单的情况，当这些“blobs”就是一些包含三个颜色空间每个轴的学习的范围时，模型给出的结果也相当好。这种模型消耗内存最少，而计算速度取决于新的像素是否在已经学习的模块里面。

【278~279】

-
- ① OpenCV 中实现的该方法的实现起源于 Kim, Chalidabhongse, Harwood 和 Davis [Kim05]，为了更高的速度，这些作者使用 YUV 空间的轴向盒子，而不使用 RGB 空间的学习型管子。清除背景图像的最快的方法可以在 Martins 中找到[Martins99]。
 - ② 关于背景模型和分割的文献有很多。用于为了收集设置分类器的数据而搜索前景目标，OpenCV 的使用主要是努力提高速度和增强鲁棒性。最近的背景减除方法允许摄像机的任意运动[Farin04;Cplombardi07]以及利用 mean-shift 算法[Liu07]处理动态背景模型。

让我们通过一个例子(图 9-3)解释 codebook。codebook 由一些 boxes 组成，这些 boxes 包含很长时间不变的像素值。图 9-3 的上一层显示了一个随着时间变化的波形。下一层中，boxes 覆盖了一个新的值，之后渐渐覆盖附近的值。如果当前值和历史值相差比较大，就会产生一个新的 box 来覆盖它，同样慢慢地接近新的值。

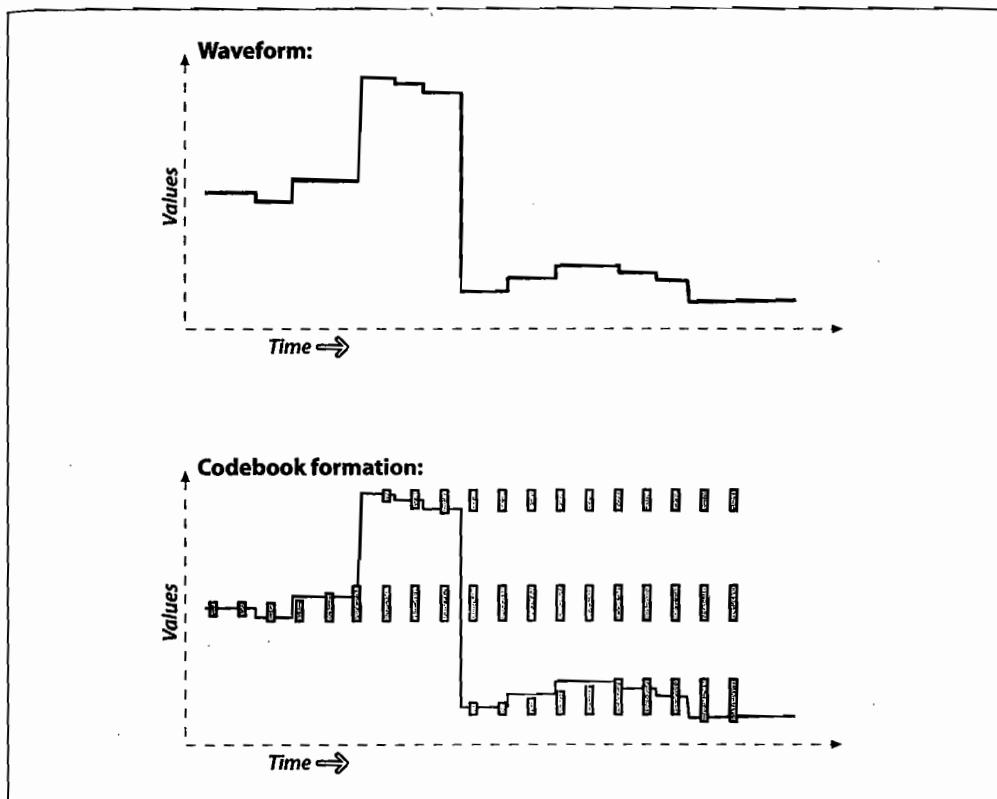


图 9-3：codebook 是“boxes”划定的强度值。形成的 box 覆盖新的值并逐渐变为覆盖附近的值。如果该值离得太远，便形成一个新的 box(见正文)

在这种背景模型情况下，我们将学习一个覆盖三维的 codebook：组成图像每个像素的三个通道。图 9-4 将 codebook 形象化了，它是从图 9-1 数据中学习的 6 个不同像素^①。这种 codebook 方法能够解决像素剧烈变化的问题(例如，被风吹的树的像素，它可能在很多树叶的颜色和树之间的蓝天颜色之间交替出现)。有了这个更精确的模型方法，我们就能够探测有不同像素值的前景目标。和图 9-2 比较，平均

① 在这种情况下，我们选择了几个扫描线上的随机像素以避免过多的杂乱。当然，实际上每个像素都有一个 codebook。

法不能从波动的像素中把手的值(图中虚线)辨别出来。先看看下一节的标题，我们稍后将看到比图 9-7 中的平均法有更好性能的方法，即 codebook 方法。

【279 ~ 280】

在背景学习模型的 codebook 方法中，在每一个三颜色轴上，每一个 box 用两个阈值(最大和最小)定义。如果新的背景模型落到学习的阈值(learnHigh 和 learnLow)之间，这些 box 的边界将膨胀(最大阈值变大，最小阈值变小)。如果新的背景样本在 box 和学习阈值外，将开始生成一个新的 box。在背景差分模型中，也能容纳 maxMod 和 minMod 阈值。使用这些阈值，可以说，如果一个像素和 box 边界的最大值和最小值非常接近，我们就认为它在 box 里面。再次调整阈值，允许模型适应特殊情形。

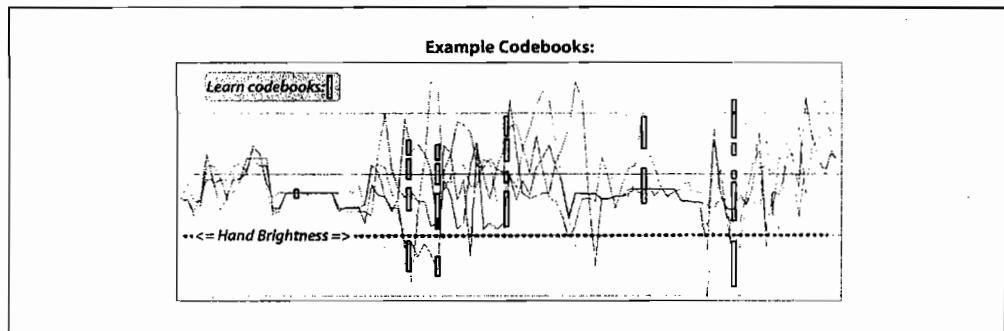


图 9-4：学习的 codebook 的亮度部分输入 6 个所选择的像素(如垂直的方块所示)。codebook box 容纳呈现多维不连续分布的像素，所以能更好地模拟像素的不连续分布；因此它们能探测到前景目标一手(虚线所示)，手的平均值在背景像素值之间可以假定。在这种情况下，codebook 的空间是一维的，仅能描述强度的变化

注意：我们没有讨论到的一个情况是由于摄像机的运动得到的场景，当这种情况发生的时候，有必要将摄像机的旋转和倾斜的角度考虑进去。

【280】

结构

现在是要看更多细节的时候了，让我们创建一个 codebook 算法的执行程序。首先，看 codebook 的结构，在 YUV 颜色空间，codebook 结构会简单地指向一串 box。

```
typedef struct code_book {  
    code_element **cb;
```

```
    int numEntries;
    int t;          //count every access
} codeBook;
```

跟踪在 `numEntries` 中有多少 `codebook`。变量 `t` 记录了从开始或最后一次清除操作之间累积的像素点的数目。实际 `codebook` 的原理的描述如下：

```
#define CHANNELS 3
typedef struct ce {
    uchar learnHigh[CHANNELS]; //High side threshold for learning
    uchar learnLow[CHANNELS]; //Low side threshold for learning
    uchar max[CHANNELS];      //High side of box boundary
    uchar min[CHANNELS];      //Low side of box boundary
    int t_last_update;        //Allow us to kill stale entries
    int stale;                //max negative run (longest period of inactivity)
} code_element;
```

每一个 `codebook` 索引消耗每个通道的 4 个字节加上 2 个整数，也就是：`CHANNELS` \times 4 + 4 + 4 个字节(如果用 3 通道就是 20 个字节)。我们可以设置 `CHANNELS` 为任何小于或等于图像通道数的整数，但是通常是指它为 1(“Y”，仅表示亮度)或者 3(YUV, HSV 空间)。在这个结构中，对于每一个通道，最大值和最小值就是 `codebook` box 的分界线。参数 `learnHigh[]` 和 `learnLow[]` 触发产生一个新的码元素的阈值。具体来说，如果一个像素值的每个通道都不在 `min - learnLow` 和 `max + learnHigh` 之间，就会生成一个新的码元素。距离上次更新和陈旧的时间(`t_last_update`)用于删除过程中学习的很少使用的码本条目。现在我们可以着手研究在这结构中使用的函数，并训练动态的背景。

背景学习

我们为每一个像素设置一个码元 `code_elements`，需要有与训练的图像像素数目长度一样的一组码本。对每一个不同的像素，调用函数 `update_codebook()` 以捕捉背景中相关变化的图像。训练可以自始至终定期更新，同时函数 `clear_stale_entries()` 用于训练有移动的前景目标(数目很小)的背景。这是有可能的，因为由移动的前景目标引起的很少使用的“陈旧”条目会被删除，函数 `update_codebook()` 的接口如下：

```
///////////
// int update_codebook(uchar *p, codeBook &c, unsigned
// cbBounds)
```

```

// Updates the codebook entry with a new data point
//
// p           Pointer to a YUV pixel
// c           Codebook for this pixel
// cbBounds    Learning bounds for codebook(Rule of thumb:10)
// numChannels Number of color channels we're learning
//
// NOTES:
//      cvBounds must be of length equal to numChannels
//
// RETURN
//      codebook index
//
int update_codebook(
    uchar*      p,
    codeBook&   c,
    unsigned*   cbBounds,
    int         numChannels
) {
    unsigned int high[3],low[3];
    for(n=0; n<numChannels; n++)
    {
        high[n] = *(p+n)+*(cbBounds+n);
        if(high[n] > 255) high[n] = 255;
        low[n] = *(p+n)-*(cbBounds+n);
        if(low[n] < 0) low[n] = 0;
    }
    int matchChannel;

    // SEE IF THIS FITS AN EXISTING CODEWORD
    //
    for(int i=0; i<c.numEntries; i++) {
        matchChannel = 0;
        for(n=0; n<numChannels; n++){
            if((c.cb[i]->learnLow[n] <= *(p+n)) &&
               //Found an entry for this channel
               (*(p+n) <= c.cb[i]->learnHigh[n]))
            {
                matchChannel++;
            }
        }
    }
}

```

```

if(matchChannel == numChannels) //If an entry was found
{
    c.cb[i]->t_last_update = c.t;
    //adjust this codeword for the first channel
    for(n=0; n<numChannels; n++){
        if(c.cb[i]->max[n] < *(p+n))
        {
            c.cb[i]->max[n] = *(p+n);
        }
        else if(c.cb[i]->min[n] > *(p+n))
        {
            c.cb[i]->min[n] = *(p+n);
        }
    }
    break;
}
}

```

【281~283】

当像素 p 超出现存的 codebook box 时，函数将增加一个 codebook 条目。当这个像素是在现存的方块里面时，方块边界将增大。如果一个像素在方块的边界距离之外，将创建一个新的 codebook。随后程序将设置一个高低阈值；然后通过每一个 codebook 条目去检查像素值 p 是否在学习的 codebook “box” 边界内；如果该像素值在所有通道都在学习的边界内，则调整阈值最大值和最小值以使该元素被包括在 codebook box，同时设置最后一次的更新时间为当前时间 c.t；接下来，update_codebook() 将统计每个码本条目多长时间被访问一次。

```

// OVERHEAD TO TRACK POTENTIAL STALE ENTRIES
//
for(int s=0; s<c.numEntries; s++){

    // Track which codebook entries are going stale:
    //
    int negRun = c.t - c.cb[s]->t_last_update;
    if(c.cb[s]->stale < negRun) c.cb[s]->stale = negRun;

}

```

在这里，变量 stale 包含最大的消极时间(即 codebook 没有数据进入的最长时间)。追踪 stale 的索引使我们能够删除那些由噪声或移动前景目标形成的从此随着时间推移变为陈旧的 codebook。背景学习的下一个阶段是，update_

codebook() 函数根据所需添加一个新的 codebook。

```
// ENTER A NEW CODEWORD IF NEEDED
//
if(i==c.numEntries) //if no existing codeword found, make one
{
    code_element **foo = new code_element* [c.numEntries+1];
    for(int ii=0; ii<c.numEntries; ii++) {
        foo[ii] = c.cb[ii];
    }
    foo[c.numEntries] = new code_element;
    if(c.numEntries) delete [] c.cb;
    c.cb = foo;
    for(n=0; n<numChannels; n++) {
        c.cb[c.numEntries]->learnHigh[n] = high[n];
        c.cb[c.numEntries]->learnLow[n] = low[n];
        c.cb[c.numEntries]->max[n] = *(p+n);
        c.cb[c.numEntries]->min[n] = *(p+n);
    }
    c.cb[c.numEntries]->t_last_update = c.t;
    c.cb[c.numEntries]->stale = 0;
    c.numEntries += 1;
}
```

【283~284】

最后，如果发现像素在 box 阈值之外，但仍然在其高和低范围内，函数 update_codebook() 将慢慢调整 learnHigh 和 learnLow 的学习界限(通过加 1)。

```
// SLOWLY ADJUST LEARNING BOUNDS
//
for(n=0; n<numChannels; n++)
{
    if(c.cb[i]->learnHigh[n] < high[n]) c.cb[i]->learnHigh[n]
+= 1;
    if(c.cb[i]->learnLow[n] > low[n]) c.cb[i]->learnLow[n] -=
1;
}
return(i);
}
```

函数结果返回修改的码本的索引。我们已经看到了 codebook 是如何学习的。为了学习现存的移动的前景目标，避免学习噪声的 codebook，我们需要一种删除在学习过程中很少有访问的 codebook 条目。

学习有移动前景目标的背景

以下程序，使用函数 `clear_stale_entries()` 允许我们训练有移动前景条件下的背景。

```
/////
//int clear_stale_entries(codeBook &c)
// During learning, after you've learned for some period of time,
// periodically call this to clear out stale codebook entries
//
// c  Codebook to clean up
//
// Return
// number of entries cleared
//
int clear_stale_entries(codeBook &c){
    int staleThresh = c.t>>1;
    int *keep = new int [c.numEntries];
    int keepCnt = 0;
    // SEE WHICH CODEBOOK ENTRIES ARE TOO STALE
    //
    for(int i=0; i<c.numEntries; i++){
        if(c.cb[i]->stale > staleThresh)
            keep[i] = 0; //Mark for destruction
        else
        {
            keep[i] = 1; //Mark to keep
            keepCnt += 1;
        }
    }
    // KEEP ONLY THE GOOD
    //
    c.t = 0;           //Full reset on stale tracking
    code_element **foo = new code_element* [keepCnt];
    int k=0;
    for(int ii=0; ii<c.numEntries; ii++){
        if(keep[ii])
        {
            foo[k] = c.cb[ii];
            //We have to refresh these entries for next clearStale
```

```

        foo[k]->t_last_update = 0;
        k++;
    }
}

// CLEAN UP
//
delete [] keep;
delete [] c.cb;
c.cb = foo;
int numCleared = c.numEntries - keepCnt;
c.numEntries = keepCnt;
return(numCleared);
}

```

【284~285】

函数由定义参数 `staleThresh` 开始，该参数设置为总运行时间的一半(经验值)。这意味着，在训练背景过程中，如果 `codebook` 的条目 `i` 在总时间一半的时间段没有被访问，该条目(`keep[i] = 0`)将被删除。

向量 `keep[]`使我们能够标识每个输入码，因此它的长度是 `c.numEntries`。变量 `keepCnt` 统计将要保持的 `codebook` 的数目。随着记录保持了哪个 `codebook` 条目，就可以创建一个新的指针 `foo`，一个 `code_element` 向量的指针，用于指向 `keepCnt` 的长度，然后把非陈旧的条目复制给该指针。最后，删除旧的指向 `codebook` 向量的指针，而用非陈旧的指针取代它。

背景差分：寻找前景目标

我们已经看到如何创建一个背景 `codebook` 模型，以及如何清除很少使用的码本条目。下面，我们来看看函数 `background_diff()`，使用经验模型在先前的背景中将前景目标的像素分割出来。

```

///////////
// uchar background_diff( uchar *p, codeBook &c,
//                         int minMod, int maxMod)
// Given a pixel and a codebook, determine if the pixel is
// covered by the codebook
//
// p          Pixel pointer (YUV interleaved)
// c          Codebook reference

```

```

// numChannels Number of channels we are testing
// maxMod      Add this (possibly negative) number onto
//               max level when determining if new pixel is foreground
// minMod      Subtract this (possibly negative) number from
//               min level when determining if new pixel is
//               foreground
//
// NOTES:
// minMod and maxMod must have length numChannels,
// e.g. 3 channels=>minMod[3], maxMod[3]. There is one min and
//      one max threshold per channel.
//
// Return
// 0 => background, 255 => foreground
//
uchar background_diff(
    uchar*      p,
    codeBook&   c,
    int         numChannels,
    int*        minMod,
    int*        maxMod
) {
    int matchChannel;

    // SEE IF THIS FITS AN EXISTING CODEWORD
    //
    for(int i=0; i<c.numEntries; i++) {
        matchChannel = 0;
        for(int n=0; n<numChannels; n++) {
            if((c.cb[i]->min[n] - minMod[n] <= *(p+n)) &&
               (*(p+n) <= c.cb[i]->max[n] + maxMod[n])) {
                matchChannel++; //Found an entry for this channel
            } else {
                break;
            }
        }
        if(matchChannel == numChannels) {
            break; //Found an entry that matched all channels
        }
    }
}

```

```
    if(i >= c.numEntries) return(255);
    return(0);
}
```

背景差分函数有一个类似于学习程序 `update_codebook` 的内循环，但在这里，我们给每个 `codebook` box 的已训练的最小和最大边界分别加上偏移量 `maxMod` 和 `minMod`。如果 `box` 中的像素在每个通道的高的部分加上 `maxMod` 或者在低的部分减去 `minMod`，则 `matchChannel` 计数将增加。当 `matchChannel` 和通道数目相同时，我们已经搜索了每一维，而且知道已经有了一个匹配。如果像素在一个训练的 `box` 内，则返回 255(一个正的检测的前景目标)，否则返回 0(背景)。`update_codebook()`, `clear_stale_entries()`, 和 `background_diff()` 这三个函数构成了一个从训练背景中将前景分割出来的 `codebook`。

【285~286】

使用 `codebook` 背景模型

使用 `codebook` 背景分割技术，通常有以下步骤。

- (1) 使用函数 `update_codebook()` 在几秒钟或几分钟时间内训练一个基本的背景模型。
- (2) 调用函数 `clear_stale_entries()` 清除 stale 索引。
- (3) 调整阈值 `minMod` 和 `maxMod` 对已知前景达到最好的分割。
- (4) 保持一个更高级别的场景模型(如以前讨论)
- (5) 通过函数 `background_diff()` 使用训练好的模型将前景从背景中分割出来。
- (6) 定期更新学习的背景像素。
- (7) 在一个频率较慢的情况下，用函数 `clear_stale_entries()` 定期清理 stale 的 `codebook` 索引。

关于 `codebook` 的一些更多的思考

通常，`codebook` 在大多数条件下效果都很好，且训练和运行速度相对较快，但它不能很好处理不同模式的光(如早晨、中午和傍晚的阳光，或在室内有人打开和熄灭灯)。这种全局变化的类型可以考虑用几种不同的在每个条件下的 `codebook` 模型来处理，然后让这种模型是条件可控的。

用于前景清除的连通部分

在比较均值法和 codebook 之前，我们应该停下来讨论使用连通成分分析来清理原始分割图像的方法。这种分析的方式包含噪声输入掩模图像；然后利用形态学“开”操作将小的噪声缩小至 0，紧接着用“闭”操作重建由于“开”操作丢失的边缘部分。然后我们可以找到“足够大”存在的部分轮廓，并可以选择地对这些片段进行统计。接着就可以恢复最大的轮廓或者大于设置阈值的所有轮廓。在如下程序中，我们实现了尽可能多的功能来处理连通区域：

- 采用多边形拟合存在的轮廓部分，或凸包设置连通轮廓有多大
- 设置连通轮廓的大小以保证不被删除
- 设置返回的连通轮廓的最大数目
- 可选返回存活的连通轮廓的外接矩形
- 可选返回存活连通轮廓的中心

【287】

实现这些操作的连通成分的头文件定义如下。

```
//////////  
////  
// void find_connected_components(IplImage *mask, int poly1_hull0,  
//                                 float perimScale, int *num,  
//                                 CvRect *bbs, CvPoint *centers)  
// This cleans up the foreground segmentation mask derived from  
// calls to backgroundDiff  
//  
// mask      Is a grayscale (8-bit depth) "raw" mask image that  
//           will be cleaned up  
//  
// OPTIONAL PARAMETERS:  
// poly1_hull0 If set, approximate connected component by  
//               (DEFAULT) polygon, or else convex hull (0)  
// perimScale Len = image (width+height)/perimScale. If contour  
//               len < this, delete that contour (DEFAULT: 4)  
// num       Maximum number of rectangles and/or centers to  
//               return; on return, will contain number filled  
//               (DEFAULT: NULL)  
// bbs       Pointer to bounding box rectangle vector of
```

```

//           length num. (DEFAULT SETTING: NULL)
// centers     Pointer to contour centers vector of length
//           num (DEFAULT: NULL)
//
void find_connected_components(
    IplImage* mask,
    int      poly1_hull0 = 1,
    float    perimScale = 4,
    int*     num        = NULL,
    CvRect*  bbs        = NULL,
    CvPoint* centers    = NULL
);

```

函数体将在下面列出。首先，我们为连通域轮廓声明存储空间。然后利用形态学“开”操作和“闭”操作来清除小的像素噪声，之后利用“开”操作来重建受到腐蚀的区域。程序有两个额外的参数，它们是由#define宏定义的固定值。定义的值一般会得到不错的结果，通常不需要改变它们。这些额外的参数控制前景区域的边界的简单程度(数目越大越简单)，以及形态学运算应执行多少次。迭代次数越多，则在“闭”操作膨胀之前，就会有越多的“开”操作腐蚀^①。越多的腐蚀操作消除越大的斑点，其代价是侵蚀了较大的边界地区。再强调一下，这个例子使用的参数会取得不错的结果，但在试验中使用其他数值也没有坏处。

```

// For connected components:
// Approx.threshold - the bigger it is, the simpler is the boundary
//
#define CVCONTOUR_APPROX_LEVEL 2

// How many iterations of erosion and/or dilation there should be
//
#define CVCLOSE_ITR 1

```

【288~289】

现在，我们讨论了连通区域算法本身。函数执行的第一部分就是形态上的开和闭操作。

```

void find_connected_components(
    IplImage *mask,

```

① 观察到的 CVCLOSE_ITR 的值实际上取决于图像的分辨率。如果是一张极高分辨率的图像把它设置为 1 不太可能产生令人满意的结果。

```

int poly1_hull0,
float perimScale,
int *num,
CvRect *bbs,
CvPoint *centers
) {

    static CvMemStorage*      mem_storage = NULL;
    static CvSeq*             contours     = NULL;

    //CLEAN UP RAW MASK
    //
    cvMorphologyEx( mask, mask, 0, 0, CV_MOP_OPEN, CVCLOSE_ITR );
    cvMorphologyEx( mask, mask, 0, 0, CV_MOP_CLOSE, CVCLOSE_ITR );

```

现在，噪声已经被从掩模图像上清除了，我们找到了所有的轮廓。

```

//FIND CONTOURS AROUND ONLY BIGGER REGIONS
//
if( mem_storage==NULL ) {
    mem_storage = cvCreateMemStorage(0);
} else {
    cvClearMemStorage(mem_storage);
}

CvContourScanner scanner = cvStartFindContours(
    mask,
    mem_storage,
    sizeof(CvContour),
    CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_SIMPLE
);

```

下一步，我们丢弃太小的轮廓，用多边形或凸包拟合剩下的轮廓(它的复杂度由CVCONTOUR_APPROX_LEVEL设置)。

```

CvSeq* c;
int numCont = 0;
while( (c = cvFindNextContour( scanner )) != NULL ) {

    double len = cvContourPerimeter( c );

```

```

// calculate perimeter len threshold:
//
double q = (mask->height + mask->width)/perimScale;

//Get rid of blob if its perimeter is too small:
//
if( len < q ) {
    cvSubstituteContour( scanner, NULL );
} else {

    // Smooth its edges if its large enough
    //
    CvSeq* c_new;
    if( poly1_hull0 ) {

        // Polygonal approximation
        //
        c_new = cvApproxPoly(
            c,
            sizeof(CvContour),
            mem_storage,
            CV_POLY_APPROX_DP,
            CVCONTOUR_APPROX_LEVEL,
            0
        );

    } else {

        // Convex Hull of the segmentation
        //
        c_new = cvConvexHull2(
            c,
            mem_storage,
            CV_CLOCKWISE,
            1
        );
    }
    cvSubstituteContour( scanner, c_new );
    numCont++;
}

```

```
contours = cvEndFindContours( &scanner );
```

【289~290】

在前面的代码，`CV_POLY_APPROX_DP`使得 Douglas-Peucker 拟合算法被调用，并且 `CV_CLOCKWISE` 是默认的凸形轮廓方向。所有这些处理将产生一系列轮廓。在将轮廓绘制到掩模图像之前，我们定义一些简单的绘制颜色：

```
// Just some convenience variables
const CvScalar CVX_WHITE    = CV_RGB(0xff,0xff,0xff)
const CvScalar CVX_BLACK   = CV_RGB(0x00,0x00,0x00)
```

在下面的代码中，我们用这些定义，首先，把掩模外的部分剔除，然后在掩模图像上绘出完整的轮廓。我们还要检查用户是否想收集轮廓的统计信息(外接矩形和中心)。

```
// PAINT THE FOUND REGIONS BACK INTO THE IMAGE
//
cvZero( mask );
IplImage *maskTemp;
// CALC CENTER OF MASS AND/OR BOUNDING RECTANGLES
//
if(num != NULL) {

    //User wants to collect statistics
    //
    int N = *num, numFilled = 0, i=0;
    CvMoments moments;
    double M00, M01, M10;
    maskTemp = cvCloneImage(mask);
    for(i=0, c=contours; c != NULL; c = c->h_next, i++ ) {

        if(i < N) {
            // Only process up to *num of them
            //
            cvDrawContours(
                maskTemp,
                c,
                CVX_WHITE,
                CVX_WHITE,
                -1,
                CV_FILLED,
                8
```

```

    );
    // Find the center of each contour
    //
    if(centers != NULL) {
        cvMoments(maskTemp,&moments,1);
        M00 = cvGetSpatialMoment(&moments,0,0);
        M10 = cvGetSpatialMoment(&moments,1,0);
        M01 = cvGetSpatialMoment(&moments,0,1);
        centers[i].x = (int)(M10/M00);
        centers[i].y = (int)(M01/M00);
    }

    //Bounding rectangles around blobs
    //
    if(bbs != NULL) {
        bbs[i] = cvBoundingRect(c);
    }
    cvZero(maskTemp);
    numFilled++;
}

// Draw filled contours into mask
//
cvDrawContours(
    mask,
    c,
    CVX_WHITE,
    CVX_WHITE,
    -1,
    CV_FILLED,
    8
);
}                                //end looping over contours
*num = numFilled;
cvReleaseImage( &maskTemp);
}

```

【290～292】

如果用户不需要掩模图像中生成区域的外接矩形和中心，我们只需要将代表背景中足够大的连通部分的已处理的轮廓在掩模图像中画出来。

```

// ELSE JUST DRAW PROCESSED CONTOURS INTO THE MASK
//
else {
    // The user doesn't want statistics, just draw the contours
    //
    for( c=contours; c != NULL; c = c->h_next ) {
        cvDrawContours(
            mask,
            c,
            CVX_WHITE,
            CVX_BLACK,
            -1,
            CV_FILLED,
            8
        );
    }
}

```

到这儿我们已得到了一个很有用的程序，它能从原始噪声掩模图像创建出完整的掩模图像。现在让我们看一下与背景减除法的简短比较。

一个快速测试

我们用一个例子来看看这个程序是如何处理实际视频的。让我们继续使用在窗口外面的树的视频。回顾(图 9-1)一只手穿过场景时的某些点。人们可能期望，我们可以用相对容易的技术找出这只手，如帧差分(以前在讨论它自己的一节)。帧差的基本思路是从前面的帧减去当前帧，然后再将该差值图像进行阈值化。在一段视频中连续的帧中往往是相似的。

因此，人们可能预见，如果我们考虑到当前帧和滞后帧的简单的差异，除非有一些前景目标经过场景，否则，将见不到太多的差异^①。但是，在这里，“见不到太多”是什么含义呢？其实，它的意思就是“噪声”，当然，实际中的问题是当前景目标出现时如何从信号中清理掉噪声。【292】

为更好理解噪声，我们将首先看视频中没有任何前景目标的两帧，仅仅只是背景和结果噪声。图 9-5 显示了视频(左上角)和前一帧(右上)。图像还显示在阈值为 15

^① 在帧差的相关内容里，一个目标主要是通过其速度被确定为“前景”。这在场景通常是非静态的或前景目标比背景目标更接近摄像机(从而根据摄像机的射影几何出现物移动的速度更快)的情况下是合理的。

(左下)的帧差分的结果。可以看到树叶的移动导致大量的噪声，然而，连通成分法可以相当不错清理这些离散噪声^①(右下角)。这并不惊讶，因为没有任何理由相信噪声有很大的空间相关性，这些信号有大量的非常小的区域来描述。

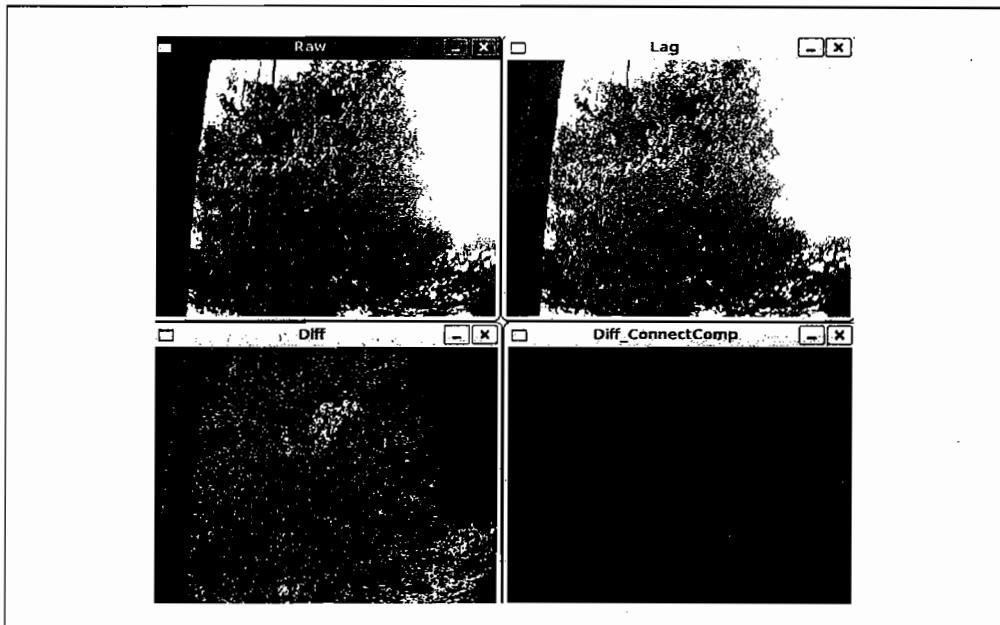


图 9-5：帧差分。一棵当前(左上角)和以前(右上)帧图像中摇曳的树。通过连通域法完全清理(右下角)不同的图像(左下)

现在考虑一个前景目标(我们无处不在的手)经过摄像机的场景情况。除了现在这只手是从左移动到右，图 9-6 显示与图 9-5 相似的两帧。与以前一样，当前帧(左上角)和前一帧(右上)的帧差分(左下)响应，且连通器法清理的结果(右下)相当不错。

【293 ~ 294】

我们还可以清楚地看到帧差的不足：它不能区分目标物体离开的位置(“空穴”)和目标物体现在的位置。此外，因为“同物相减”是 0(至少在阈值以下)，在重叠的区域往往出现一个缺口。

因此，我们看到，连通域法是一个功能强大的在背景减去中去除噪声的技术。通过这个例子，我们还能够看到帧差分一些优点和缺点。

① 在这些空帧中，连通域的阈值的大小已调为 0 了。真正的问题是感兴趣的前景目标(手)是否在这个阈值下能免予修剪。我们将看到它做得很精细(图 9-6)。

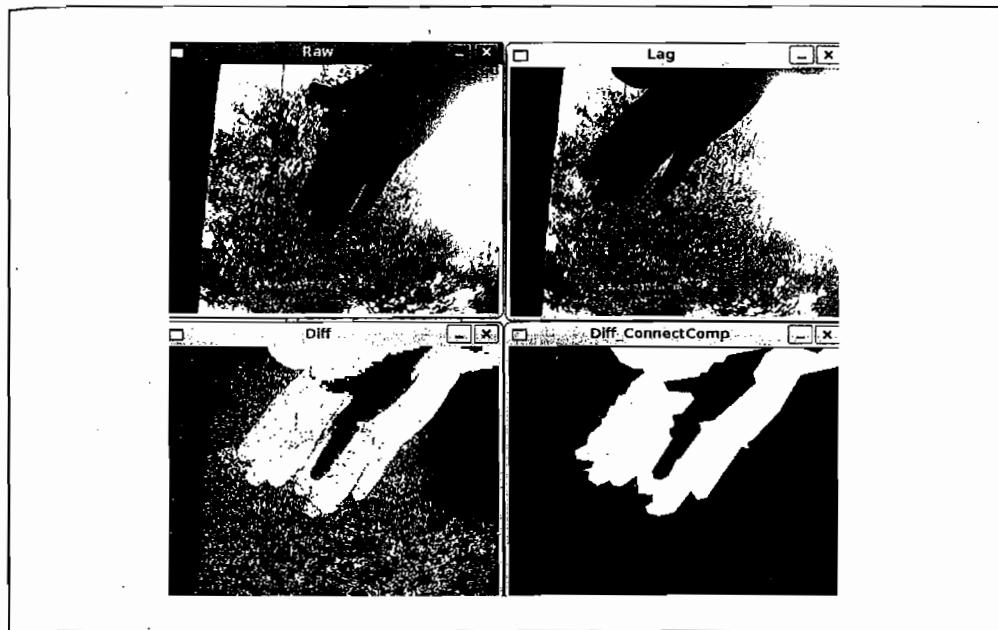


图 9-6：检测手的帧差分方法，手作为前景物体从左移动到右(上面的两个图像)；差别图像(左下)显示了朝向左的“空穴”(手以前在的位置)和朝向右的前沿，连通域图像显示了清理的差异(右下)

背景比较法

本章我们已经讨论了两种背景建模技术：平均距离法和 codebook 法。您可能想知道哪种方法更好，或者至少你想知道什么情况下，运用哪种方法更简单。在这种情况下，最好就是直接对两种有用的方法进行食物烘烤竞赛 bakeoff^①。【294~295】

我们将继续讨论本章中我们一直讨论的有树的视频。除了移动的树以外，视频还有大量从一个建筑物打到右侧的强光，在左边的内墙上剩下部分。对这样的背景建模确实非常具有挑战性。

在图 9-7 中，我们比较平均差分法(上图)和 codebook 法(下图)；左边的是原始前景图像，右边的则是经处理后的连通部分。如所预料的，平均差分方法留下了一个不整洁的掩模图像，将手分成两部分。在图 9-2，我们看到使用的平均差分法的均

① 对于不知情的人来说，“食物烘烤竞赛”实际上是一个用来在预定义数据集上的多个算法比较的习惯术语。

值作为背景模型通常包括与手相关的像素值(图中虚线所示)。图 9-4 给出了对比, codebook 法能更准确地模拟波动的枝叶和更准确地从背景像素中将前景手(虚线)识别出来。从图 9-7 可以看出不仅仅是背景建模产生少量的噪声, 连通部分也能够生成一个相对精确的目标轮廓。

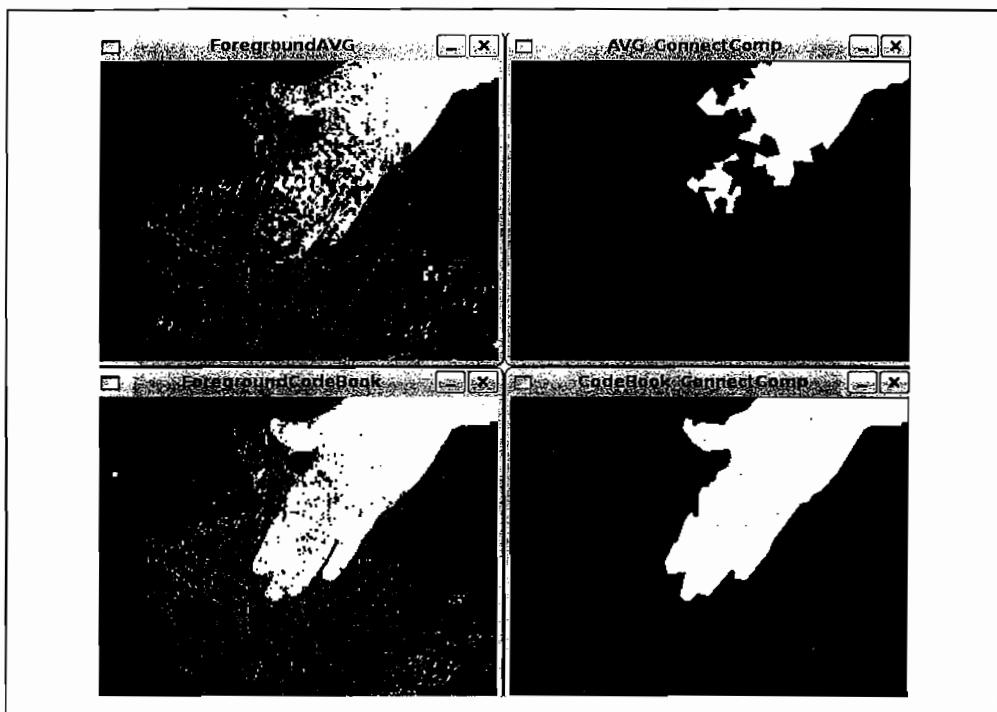


图 9-7: 运用平均法(第一行), 区域连接消除法, 手指部分缺失(上右); codebook 法(第二排)的分割效果更好些, 且产生一个清晰地连通部分掩模(下右)

分水岭算法

在许多实际情况下, 我们要分割图像, 但无法从背景图像中获得有用信息。分水岭算法(watershed algorithm)在这方面往往是有效的[Meyer92]。该算法可以将图像中的边缘转化成“山脉”, 将均匀区域转化为“山谷”, 这样有助于分隔目标。分水岭算法首先计算灰度图像的梯度; 这对山谷或没有纹理的盆地(亮度值低的点)的形成有效, 也对山头或图像中有主导线段的山脉(山脊对应的边缘)的形成有效。然后开始从用户指定点(或者算法得到点)开始持续“灌注”盆地直到这些区域连在一起。基于这样产生的标记就可以把区域合并到一起, 合并后的区域又通过聚集的方

式进行分割，好像图像被“填充”起来一样。通过这种方式，与指示点相连的盆地就为指示点“所拥有”。最终我们把图像分割成相应的标记区域。

更确切地说，分水岭算法允许用户(或其他算法！)来标记目标的某个部分为目标，或背景的某个部分为背景。用户或算法可以通过画一条简单的线，有效地告知分水岭算法把这些点像这样组合起来。接着分水岭算法通过允许在梯度图像中和片段连接的标识区域“拥有”边沿定义的山谷来分割图像。图 9-8 描述了这一算法。分水岭算法的函数定义如下：

```
void cvWatershed(  
    const CvArr* image,  
    CvArr* markers  
) ;
```

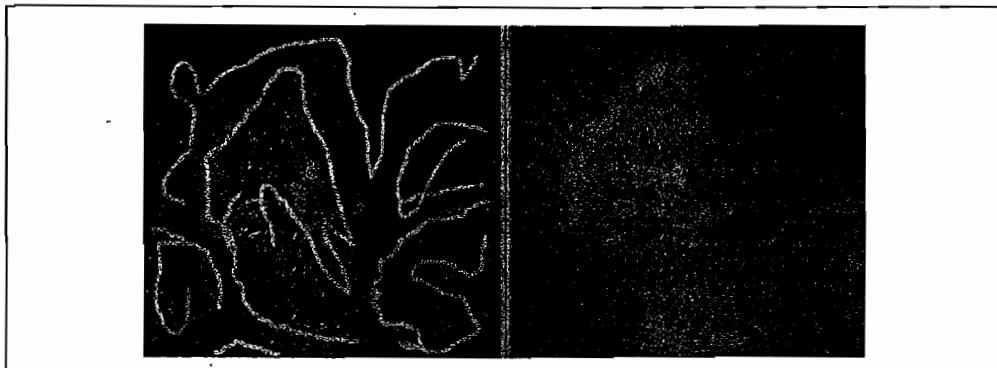


图 9-8：分水岭算法。在用户标记物体上的区域(左图)以后，算法将标记区域合成到了分割片段中(右图)

这里 `image` 是一个 8 位(三通道)的彩色图像，而 `markers` 是单通道整型(IPL_DEPTH_32S)，具有相同维数(x, y)的图像。除非用户(或算法)用正整数标记属于同一部分的区域，`markers` 的值都是 0。例如，在图 9-8 中的左边部分，橙子被标记为 1，柠檬为 2，柳橙为 3，上边的前景为 4 等等。这样的处理，生成了右侧同样效果的分割。

【295~297】

用 Inpainting 修补图像

图像常常被噪声腐蚀。这些噪声也许是镜头上的灰尘或水滴造成的，也可能是旧照片上的划痕，或者图像的部分已经被破坏了。Inpainting[Telea04]是修复这些损害

(实际上，可能是一只手)经过摄像机的前面。这个前景目标显然不如背景中的天空和树明亮，手的亮度在图 9-2 中也表示出来了。

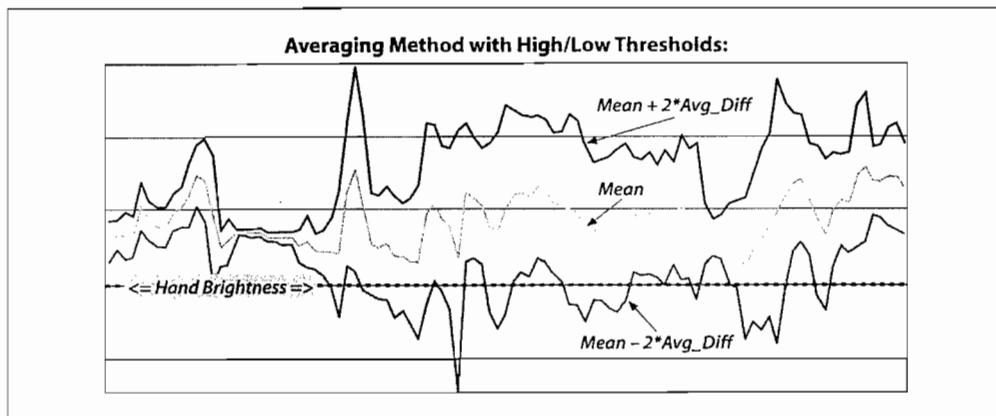


图 9-2：表示的是一组平均差分。一个目标(一只手)从摄像机前掠过，目标亮度相对较暗，图中表示了它的亮度

平均背景法使用四个 OpenCV 函数：`cvAcc()`，累积图像；`cvAbsDiff()`，计算一定时间内的每帧图像之差；`cvInRange()`，将图像分割成前景区域和背景区域(背景模型已经学习的情况下)；函数 `cvOr()`，将不同的彩色通道图像中合成为一个掩模图像。由于这个例子代码比较长，我们将它分开成几块，对每一块分别讨论。

首先，我们为需要的不同临时图像和统计属性图像创建指针。这样有助于根据图像的不同类型对以后使用的图像指针排序。

```
//Global storage
//
//Float, 3-channel images
//
IplImage *IavgF,*IdiffF, *IprevF, *IhiF, *IlowF;

IplImage *Iscratch,*Iscratch2;

    Float, 1-channel images
    /
IplImage *Igray1,*Igray2, *Igray3;
IplImage *Ilow1,  *Ilow2, *Ilow3;
IplImage *Ihi1,   *Ihi2, *Ihi3;

    Byte, 1-channel image
```

```
//  
IplImage *Imaskt;  
  
//Counts number of images learned for averaging later.  
//  
float Icount;
```

【271~272】

接下来，我们创建一个函数来给需要的所有临时图像分配内存。为了方便，我们传递一幅图像(来自视频)作为大小参考来分配临时图像。

```
// I is just a sample image for allocation purposes  
// (passed in for sizing)  
//  
void AllocateImages( IplImage* I ){  
  
    CvSize sz = cvGetSize( I );  
  
    IavgF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IdiffF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IprevF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IhiF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    IlowF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    Ilow1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ilow2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ilow3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ihi1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ihi2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Ihi3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    cvZero( IavgF );  
    cvZero( IdiffF );  
    cvZero( IprevF );  
    cvZero( IhiF );  
    cvZero( IlowF );  
    Icount = 0.00001; //Protect against divide by zero  
  
    Iscratch = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    Iscratch2 = cvCreateImage( sz, IPL_DEPTH_32F, 3 );  
    Igray1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Igray2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Igray3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );  
    Imaskt = cvCreateImage( sz, IPL_DEPTH_8U, 1 );
```

```

void cvPyrMeanShiftFiltering(
    const CvArr* src,
    CvArr* dst,
    double spatialRadius,
    double colorRadius,
    int max_level = 1,
    CvTermCriteria termcrit = cvTermCriteria(
        CV_TERMCRIT_ITER | CV_TERMCRIT_EPS,
        5,
        1
    )
);

```

在 `cvPyrMeanShiftFiltering()` 里，我们拥有一个输入图像 `src` 和一个输出图像 `dst`。均为 8 位，三通道，且大小相等的彩色图像。`spatialRadius` 和 `colorRadius` 定义了均值漂移算法如何均衡颜色和空间以分割图像。对于一个分辨率为 640×480 的彩色图像，将 `spatialRadius` 设为 2，`colorRadius` 设为 40，效果很好。而算法的下一个参数则是 `max_level`，表示在分割中使用多少级金字塔。同样对于一个分辨率为 640×480 的彩色图像，`max_level` 设置为 2 或 3 效果很好。

最后一个参数是我们在第八章所见到的 `CvTermCriteria`。`CvTermCriteria` 在所有的 OpenCV 迭代算法中都会被使用。当参数默认时，均值漂移分割函数有很好的默认值。否则 `cvTermCriteria` 将有如下构造器：

```

cvTermCriteria(
    int type; // CV_TERMCRIT_ITER, CV_TERMCRIT_EPS,
    int max_iter,
    double epsilon
);

```

`cvTermCriteria()` 函数的典型运用是产生所需要的 `CvTermCriteria` 结构。第一个参数是 `CV_TERMCRIT_ITER` 或者 `CV_TERMCRIT_EPS`，它们告诉算法在一定的迭代次数之后或当收敛矩阵达到某个很小的值(各自)时，终止计算。接下来的两个参数设定了其中一个或者两个终止算法的准则。我们之所以有两种选择，是因为可以设置 `CV_TERMCRIT_ITER` 的类型。参数 `max_iter` 限制了所设 `CV_TERMCRIT_ITER` 的重复次数。当然 `epsilon` 的精确意义依赖于算法。

【299~300】

图 9-11 显示了使用下面参数值的均值漂移分割的结果：

```
cvPyrMeanShiftFiltering( src, dst, 20, 40, 2);
```

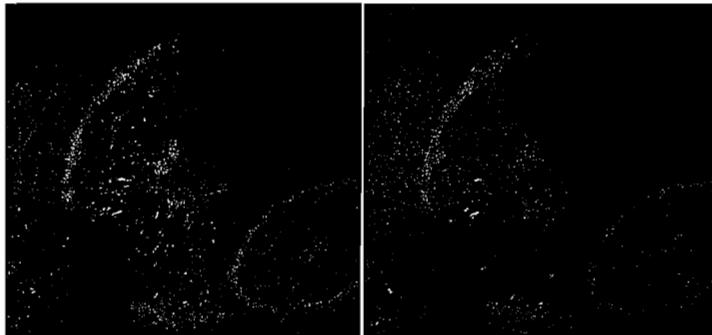


图 9-11：均值漂移分割 通过比例运用 cvPyrMeanShiftFiltering(), 参数 max_level=2, spatialRadius=20 和 colorRadius=40;；相似区域具有相近值，因此可以视为超像素，能显著增加随后的处理速度

Delaunay 三角剖分和 Voronoi 划分

Delaunay 三角剖分是 1934 年发明的将空间点连接为三角形，使得所有三角形中最小的角最大的一个技术。这意味着 Delaunay 三角剖分力图避免出现瘦长三角形。如图 9-12 可以看到三角剖分的要点，那就是任何三角形的外接圆都不包含任何其他顶点，这叫外接圆性质(图 C)。

从计算效率讲，Delaunay 算法从一个远离三角形边界的外轮廓开始运行。图 9-12(b) 代表虚构的外三角的由虚线引至最高点。图 9-12(c) 表明一些外接圆性质的例子，包括一个连接两个界外点的实时数据的一个顶点虚拟外部三角形。 【300】

现在有很多种算法计算 delaunay 三角剖分，其中一些很有效，但内部描述却很难描述。其中一个较简单算法的要点如下。

- (1) 添加外部三角形，并从它的一个顶点(在这里会产生一个确定的外部起点)处开始。
- (2) 加入一个内部点；在三角形的外接圆内搜寻该点，去掉包含该点的三角剖分。
- (3) 重新构造三角图，在刚刚去掉的三角形外接圆内包括新的点。
- (4) 返回第二步，直到没有新的点加入。

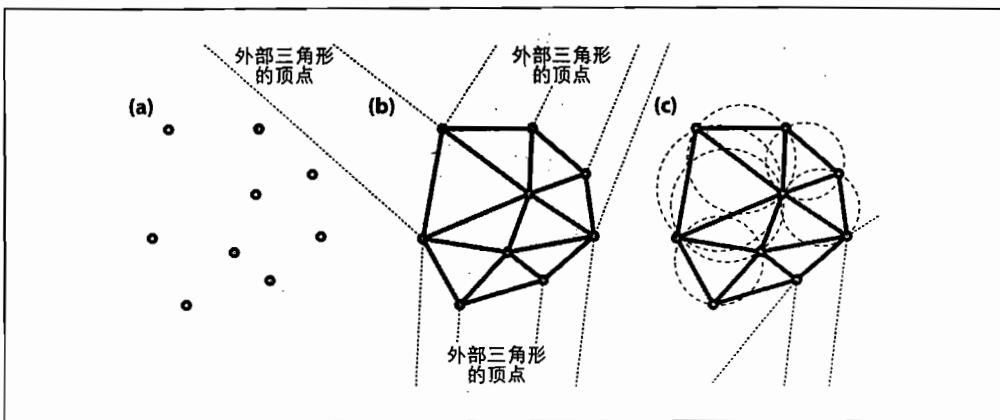


图 9-12: delaunay 三角剖分。(a)点集合; (b)外面边界三角形带有尾巴的点集的 delaunay 三角剖分; (c)演示外接圆效应的特征圆

该算法的复杂度是 $O(n^2)$ 。最好的算法的复杂度是 $O(n \log \log n)$ (平均值)。

很好，但是它究竟有什么用处呢？算法是开始于一个虚拟的外部三角形，所以事实上，所有的外部实点都连接着两个三角的顶点。现在，让我们回顾一下外接圆特点：经过任意两个外部点和虚拟顶点的圆不包含任何内部点。这就意味着，通过检查哪个点与外部的三个虚拟凸顶点相连接，计算机可以直接找出哪些点在点集中构成外部轮廓。换句话说，通过 Delaunay 三角剖分，我们可以直接找到一组点的外部轮廓。

我们同样可以找到，在点与点之间的空间被谁“拥有”，也就是说哪个坐标是距离 Delaunay 顶点最近的。因此，运用原始点的 Delaunay 三角测量，你可以快速搜索新点的最近邻居。这种划分叫 Voronoi Tessellation(见图 9-13)。划分是 Delaunay 三角剖分的对偶图像，因为 Delaunay 线定义了已存在点之间的距离，这样 Voronoi 线就“知道”在哪里要插入 Delaunay 线以保证点和点之间的等距离。利用这两种方法，可以计算出凸形外界和最近的外部轮廓，而这对于点的聚类和分类操作是非常重要的！

【301 ~ 302】

如果你熟悉计算机图形学，你便会知道 Delaunay 三角剖分是表现三维形状的基础。如果我们在三维空间渲染一个，我们可以通过这个物体的投影来建立二维视觉图，并用二维 Delaunay 三角剖分来分析识别该物体，或者将它与实物相比较。Delaunay 三角剖分就是连接计算机视觉与计算机图形学的桥梁。然而，使用 OpenCV 实现 Delaunay 三角的一个不足之处(我们希望以后的版本会更正，见第 14 章)就是 OpenCV 仅实现了二维的 Delaunay 剖分。如果我们能够对三维点云进行三角剖分，也就是说构成立体视觉(见第 11 章)，那么我们可以在三维的计算机图形

和计算机视觉之间进行无缝的转换。然而，二维三角剖分通常用于计算机视觉中标记空间目标的排列特征或运动场景跟踪，目标识别，或两个不同的摄像机的场景匹配（如同从立体图像中获得深度信息）。图 9-14 显示 Delaunay 三角剖分 [Gokturk01; Gokturk02] 在跟踪和识别中的应用，其中关键的面部特征点根据它们的三角空间分布。

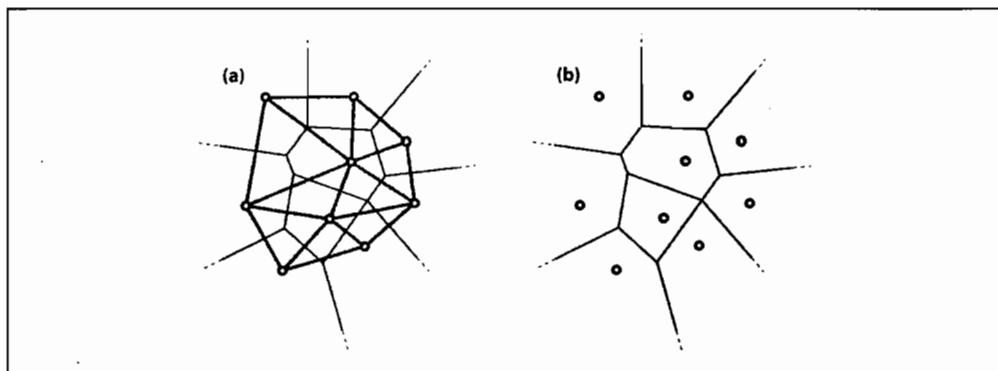


图 9-13：Voronoi 划分，任何包含在 Voronoi 单元中的点都比其他 Delaunay 点更接近于它们自己的 Delaunay 点：(a)粗线 Delaunay 三角剖分以及与细线表示 Voronoi 划分；(b)围绕着每一个 Delaunay 点的 Voronoi 泡子

现在一旦给出一组点，我们就确定了 Delaunay 的有用性，那么如何得到三角形呢？OpenCV 在 `.../opencv/samples/c/delaunay.c` 文件中有这些代码例子。OpenCV 把 Delaunay 三角的函数归为 Delaunay 的一部分，我们接下来将会讨论这些关键性的可重复使用的部分。

【302】



图 9-14：Delaunay 点可用于追踪目标，这里，面部通过有意义的点的实现跟踪，由此可以识别表情

创立一个 Delaunay 或 Voronoi 细分

首先，我们需要储存 Delaunay 的内存空间。还需要一个外接矩形(记住为了要加速计算，算法需要用由一个外接矩形盒子所确定的虚拟三角形)，为了设置这些参数，假设点都位于 600×600 的图像中。

```
// STORAGE AND STRUCTURE FOR DELAUNAY SUBDIVISION
//
CvRect      rect = { 0, 0, 600, 600 }; //Our outer bounding box
CvMemStorage* storage; //Storage for the Delaunay subdivision
storage = cvCreateMemStorage(0); //Initialize the storage
CvSubdiv2D*    subdiv;           //The subdivision itself
subdiv = init_delaunay( storage, rect); //See this function
//below
```

代码调用 `init_delaunay()` 函数，它不仅是一个 OpenCV 函数，更是一个包含一些 OpenCV 函数的函数包。

```
//INITIALIZATION CONVENIENCE FUNCTION FOR DELAUNAY SUBDIVISION
//
CvSubdiv2D* init_delaunay(
    CvMemStorage* storage,
    CvRect rect
) {
    CvSubdiv2D* subdiv;
    subdiv = cvCreateSubdiv2D(
        CV_SEQ_KIND_SUBDIV2D,
        sizeof(*subdiv),
        sizeof(CvSubdiv2DPoint),
        sizeof(CvQuadEdge2D),
        storage
    );
    cvInitSubdiv Delaunay2D(subdiv, rect); //rect sets the bounds
    return subdiv;
}
```

【303~304】

接下来，我们需要知道如何插入点。这些点必须是 32 位浮点型的

```
CvPoint2D32f fp; //This is our point holder
```

```

for( i = 0; i < as_many_points_as_you_want; i++ ) {

    // However you want to set points
    //
    fp = your_32f_point_list[i];

    cvSubdivDelaunay2DInsert( subdiv, fp );
}

```

可以通过宏 `cvPoint2D32f(double x, double y)` 或者 `cxtypes.h` 源里的 `cvPointTo32f(CvPoint point)` 函数将整型点方便地转换为 32 位浮点数。当可以输入点来得到 Delaunay 三角剖分后，接下来我们用以下两个函数设置和清除相关的 Voronoi 划分。

```

cvCalcSubdivVoronoi2D( subdiv ); // Fill out Voronoi data in
                                // subdiv
cvClearSubdivVoronoi2D( subdiv ); // Clear the Voronoi from
                                // subdiv

```

在两个函数中，`subdiv` 的类型是 `CvSubdiv2D*`。现在我们可以创立二维点集的 Delaunay 细分，然后加入和清除 Voronoi 划分。但是，怎样做才能获得这些结构里面的有用信息呢？我们可以在分解中逐步由边缘到点，或由边缘到边缘来完成这个步骤；见图 9-15，从给定边缘及其原始点开始基本操作。接下来我们以两种不同的方式发现细分中的边缘或点：(1)通过利用外部点来定位一个边缘或顶点；(2)逐步遍历一系列点或边缘。我们将首先描述如何遍历图像中的边缘和点，接下来遍历整个图像。

Delaunay 细分遍历

图 9-15 结合了我们用于遍历整个细分图的两种数据结构。`cvQuadEdge2D` 结构包含了两个 Delaunay 点和两个 Voronoi 点以及连接它们的边缘(假设 Voronoi 点和边缘已经由函数计算出来)，如图 9-16 所示。`CvSubdiv2DPoint` 结构包含 Delaunay 边缘及其相连的顶点，如图 9-17 所示。`quad-edge` 结构的定义代码在图下面。【304】

```

// Edges themselves are encoded in long integers. The lower two
// bits are its index (0..3) and upper bits are the quad-edge
// pointer.
//
typedef long CvSubdiv2DEdge;

```

```

// quad-edge structure fields:
//
#define CV_QUADEDGE2D_FIELDS()           /
    int flags;                         /
    struct CvSubdiv2DPoint* pt[4];      /
    CvSubdiv2DEdge next[4];
}

typedef struct CvQuadEdge2D {
    CV_QUADEDGE2D_FIELDS()
} CvQuadEdge2D;

```

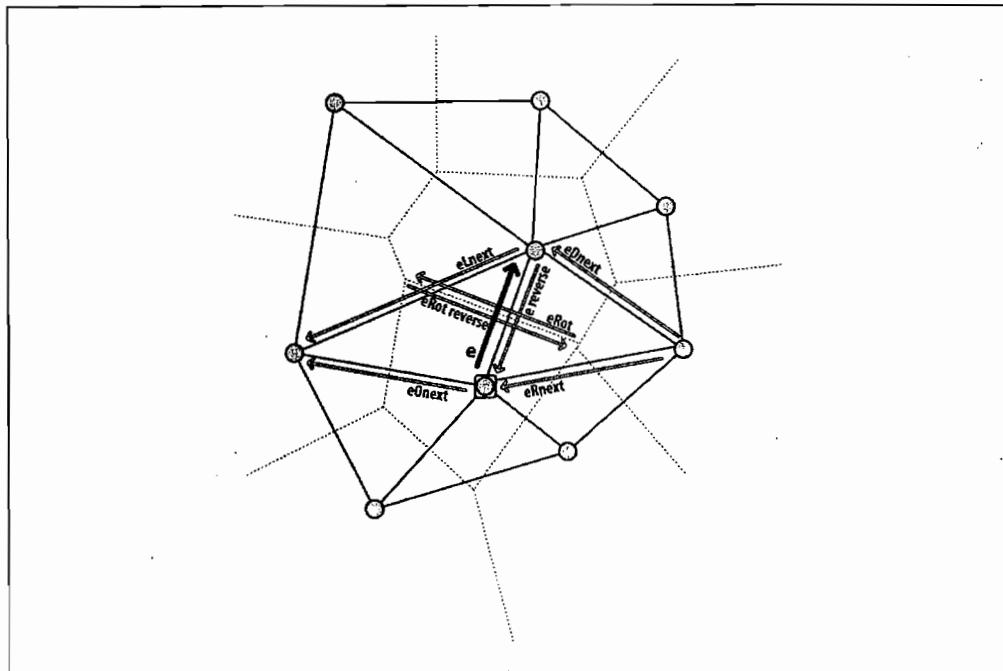


图 9-15：给定一个边缘的相关边，标为 e，其顶点(标记为一个正方形)

Delaunay 细分的点和相关的边缘结构定义如下：

```

#define CV_SUBDIV2D_POINT_FIELDS() /
    int             flags;          /
    CvSubdiv2DEdge first;         /*The edge "e" in the figures.*/
    CvPoint2D32f   pt;
#define CV_SUBDIV2D_VIRTUAL_POINT_FLAG (1 << 30)

typedef struct CvSubdiv2DPoint

```

```

{
    CV_SUBDIV2D_POINT_FIELDS()
}
CvSubdiv2DPoint;

```

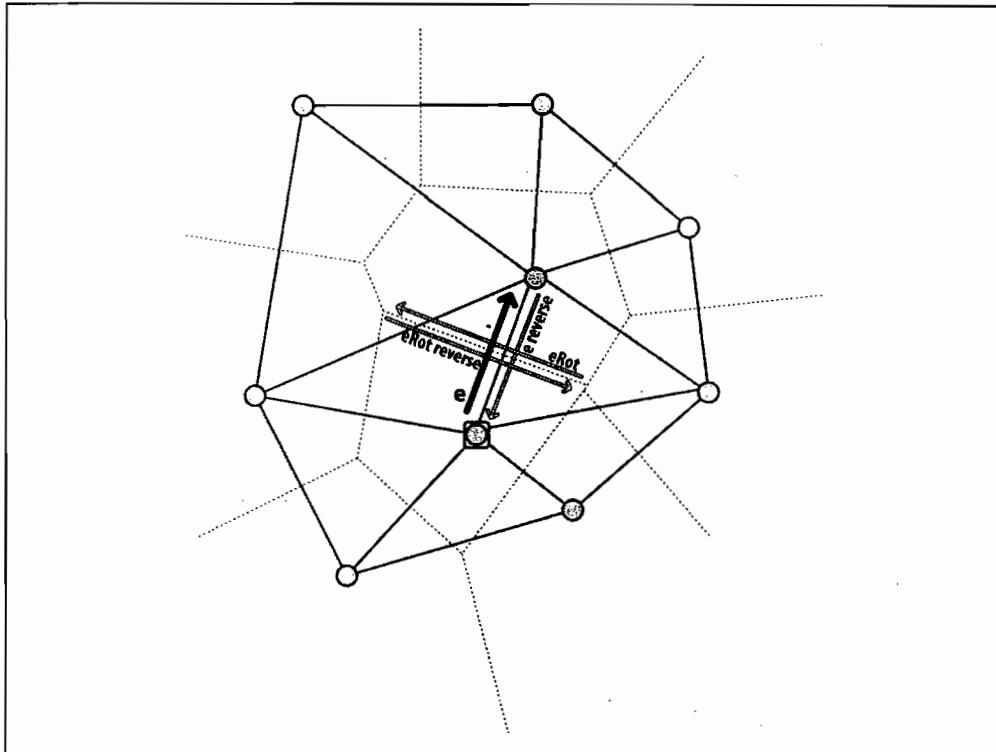


图 9-16：用包含 Delaunay 边缘，其对边(连同相关的顶点)以及相关的 Voronoi 边和点 cvSubdiv2DRotateEdge ()获得的四边形

利用这些已知结构，我们可以检查移动的不同方式。

【305 ~ 306】

边缘遍历

如图 9-16 所示，我们可通过使用下面的函数遍历四边形：

```

CvSubdiv2DEdge cvSubdiv2DRotateEdge(
    CvSubdiv2DEdge edge,
    int          type
);

```

【306】

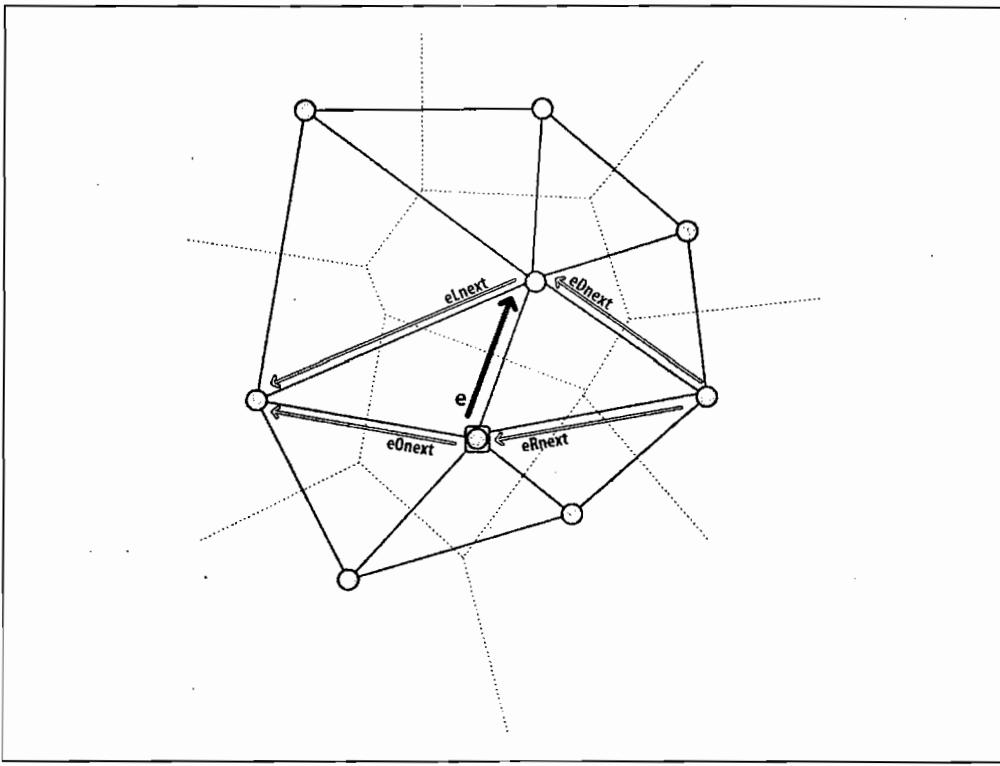


图 9-17：一个 CvSubdiv2DPoint 顶点及其连接边 e 连同其他连的边缘，可由 cvSubdiv2DGetEdge() 来求取

给定一个边，我们可以使用 `type` 变量得到下一个边缘，其中 `type` 可以采用如下参数：

- 输入边(e ，如果 e 是输入边，它必须在图像中)
- 旋转边缘($eRot$)
- 相对边缘($reversed\ e$)
- 颠倒的旋转边缘($reversed\ eRot$)

参照图 9-17，我们也可以使用 `CvSubdiv2DEdge` 遍历 Delaunay 图。

```
CvSubdiv2DEdge cvSubdiv2DGetEdge(
    CvSubdiv2DEdge edge,
    CvNextEdgeType type
);
```

```
#define cvSubdiv2DNextEdge( edge ) /  
    cvSubdiv2DGetEdge( /  
        edge, /  
        CV_NEXT_AROUND_ORG /  
    )
```

【307】

这里详细介绍以下遍历的类型：

- CV_NEXT_AROUND_ORG 下一个边缘原点(图 9-17 中 eOnext, 如果 e 是输入边)
- CV_NEXT_AROUND_DST 下一个边缘顶点
- CV_PREV_AROUND_ORG 前一个边缘起点(反向 eRnext)
- CV_PREV_AROUND_DST 前一个边缘终点(反向 eLnext)
- CV_NEXT_AROUND_LEFT 下一个左平面(eLnext)
- CV_NEXT_AROUND_RIGHT 下一个右平面(eRnext)
- CV_PREV_AROUND_LEFT 前一个左平面(反向 eOnext)
- CV_PREV_AROUND_RIGHT 前一个右平面(反向 eDnext)

请注意，给定一个与顶点连接的边缘，我们可以利用宏 `cvSubdiv2DnextEdge`(edge)找到连接该顶点的所有其他边缘。这有助于从外接三角的顶点(虚线)开始寻找类似凸包的东西。

其他重要的遍历类型是 CV_NEXT_AROUND_LEFT 和 CV_NEXT_AROUND_RIGHT。我如果我们在一个 Delaunay 边缘上就可以使用这些类型遍历 Delaunay 三角形，或者在 Voronoi 边缘上遍历 Voronoi 单元。

来自边缘的点

我们还需要知道如何从 Delaunay 或 Voronoi 顶点中获得实际点。每个 Delaunay 或 Voronoi 边缘有两个与之相连的点：org 为原始点，dst 为终点。可以轻易地通过如下函数得到：

```
CvSubdiv2DPoint* cvSubdiv2DEdgeOrg( CvSubdiv2DEdge edge );  
CvSubdiv2DPoint* cvSubdiv2DEdgeDst( CvSubdiv2DEdge edge );
```

这里是几个将 CvSubdiv2DPoint 转化为更熟悉的方法：

```

CvSubdiv2Dpoint ptSub;           //Subdivision vertex point
CvPoint2D32f    pt32f = ptSub->pt; // to 32f point
CvPoint         pt      = cvPointFrom32f(pt32f);
                           // to an integer point

```

我们现在知道细分结构是什么样的了，也知道了如何遍历它的点和边缘。再回到从 Delaunay/Voronoi 细分得到的第一条边或点的两种方法。

方法 1：使用一个外部点定位边缘或顶点

第一种方法是取任意一点，然后在细分中定位该点。该点不一定是三角剖分中的点，而可以为任意点。cvSubdiv2DLocate() 函数填充三角形的边缘和顶点(如果必要)或者填充该点所处在的 Voronoi 面。

```

CvSubdiv2DPointLocation cvSubdiv2DLocate(
    CvSubdiv2D*          subdiv,
    CvPoint2D32f          pt,
    CvSubdiv2DEdge*       edge,
    CvSubdiv2DPoint**     vertex = NULL
);

```

请注意，这些不必是最接近的边缘或顶点，它们只需要在三角形或表面上。此函数的返回值按下列方式说明点的位置。

- **CV_PTLOC_INSIDE** 点落入某些面；*edge 将包含该面的一个边缘。
- **CV_PTLOC_ON_EDGE** 点落于边缘；*edge 含有这个边缘。
- **CV_PTLOC_VERTEX** 该点符合一个细分顶点重合；*vertex 将包含该顶点指针。
- **CV_PTLOC_OUTSIDE_RECT** 点处于细分参考矩形之外；该函数返回后不填补点。
- **CV_PTLOC_ERROR** 输入变量无效

方法 2：遍历一系列点或边缘

为方便起见，当创建点集的 Delaunay 细分时，开始的三点和边缘构成了假定外接三角形的顶点和三个边。由此，我们可以直接访问来自实际数据点凸包的外点和边缘。一旦我们建立了一个 Delaunay 细分(称为 subdiv)，我们还将需要调用

`cvCalcSubdivVoronoi2D(subdiv)` 函数计算相关的 Voronoi 划分。然后，我们就能用 `CvSubdiv2DPoint* outer_vtx[3]` 得到所用的外接三角形的三个顶点。

```
CvSubdiv2DPoint* outer_vtx[3];
for( i = 0; i < 3; i++ ) {
    outer_vtx[i] =
        (CvSubdiv2DPoint*)cvGetSeqElem( (CvSeq*)subdiv, I );
}
```

【309】

同样可以得到外界三角形的三个边：

```
CvQuadEdge2D* outer_qedges[3];
for( i = 0; i < 3; i++ ) {
    outer_qedges[i] =
        (CvQuadEdge2D*)cvGetSeqElem( (CvSeq*)(my_subdiv->edges), I );
}
```

一旦知道如何遍历图形，我们也能知道何时处于外部边缘或点的边界。

确定凸包的外接三角形或边缘并遍历凸包

回想一下我们通过调用 `cvInitSubdivDelaunay2D(subdiv, rect)` 来初始化 Delaunay 三角剖分。在这种情况下，下面的论述成立。

- (1) 如果边缘的起点和终点都在矩形之外，那么此边缘也在细分的外接三角形上。
- (2) 如果边缘的一端在矩形内，一端在矩形边界外，那么矩形边界上的点落在凸集上，凸集上的每个点与虚外接三角形的两顶点相连，这两边相继出现。

从第二个条件可知，可以使用宏 `cvSubdiv2DNextEdge()` 移到第一条边上，这条边的 `dst` 在边界内。两端点都在边界上的第一条边在凸集点上，记下该点或边。一旦它在凸集上，就可以如下遍历所有顶点。

- (1) 将凸包遍历一周后，通过 `cvSubdiv2DRotateEdge(CvSubdiv2DEdge edge, 0)` 函数移动到凸包的下一条边。
- (2) 接着，两次调用宏 `cvSubdiv2DNextEdge()` 就到了凸包的下一条边。跳转到第一步。

我们现在知道如何初始化 Delaunay 和 Voronoi 细分，如何找到初始边缘，以及如何遍历图形中的边和点。下一节将阐述一些实际应用。

使用实例

我们可以用函数 `cvSubdiv2DLocate()` 遍历 Delaunay 三角剖分的边。

```
void locate_point(
    CvSubdiv2D* subdiv,
    CvPoint2D32f fp,
    IplImage* img,
    CvScalar active_color
) {
    CvSubdiv2DEdge e;
    CvSubdiv2DEdge e0 = 0;
    CvSubdiv2DPoint* p = 0;
    cvSubdiv2DLocate( subdiv, fp, &e0, &p );
    if( e0 ) {
        e = e0;
        do // Always 3 edges -- this is a triangulation, after all.
        {
            // [Insert your code here]
            //
            // Do something with e ...
            e = cvSubdiv2DGetEdge(e,CV_NEXT_AROUND_LEFT);
        }
        while( e != e0 );
    }
}
```

【310~311】

也可能通过下面函数找到距离输入点最近的点：

```
CvSubdiv2DPoint* cvFindNearestPoint2D(
    CvSubdiv2D* subdiv,
    CvPoint2D32f pt
);
```

与 `cvSubdiv2DLocate()` 函数不同，`cvFindNearestPoint2D()` 返回 Delaunay 细分的最近顶点。该点不一定落在该点所在的面或三角形内。

同样地，我们能够通过使用如下语句逐步遍历(本例中也绘出了)Voronoi 面。

```
void draw_subdiv_facet(
```

```

IplImage *img,
CvSubdiv2DEdge edge
) {

    CvSubdiv2DEdge t = edge;
    int i, count = 0;
    CvPoint* buf = 0;

    // Count number of edges in facet
    do{
        count++;
        t = cvSubdiv2DGetEdge( t, CV_NEXT_AROUND_LEFT );
    } while (t != edge );

    // Gather points
    //
    buf = (CvPoint*)malloc( count * sizeof(buf[0]) );
    t = edge;
    for( i = 0; i < count; i++ ) {
        CvSubdiv2DPoint* pt = cvSubdiv2DEdgeOrg( t );
        if( !pt ) break;
        buf[i] = cvPoint( cvRound(pt->pt.x), cvRound(pt->pt.y));
        t = cvSubdiv2DGetEdge( t, CV_NEXT_AROUND_LEFT );
    }

    // Around we go
    //
    if( i == count ){
        CvSubdiv2DPoint* pt = cvSubdiv2DEdgeDst(
            cvSubdiv2DRotateEdge( edge, 1 ));
        cvFillConvexPoly( img, buf, count,
            CV_RGB(rand()&255,rand()&255,rand()&255), CV_AA, 0 );
        cvPolyLine( img, &buf, &count, 1, 1, CV_RGB(0,0,0),
            1, CV_AA, 0 );
        draw_subdiv_point( img, pt->pt, CV_RGB(0,0,0));
    }
    free( buf );
}

```

【311~312】

最后，获得细分结构的另一种方法是使用 CvSeqReader 逐步遍历边。这里介绍如

何遍历所有 Delaunay 或者 Voronoi 边：

```
void visit_edges( CvSubdiv2D* subdiv) {
    CvSeqReader reader;                                //Sequence reader
    int i, total = subdiv->edges->total;             //edge count
    int elem_size = subdiv->edges->elem_size;          //edge size

    cvStartReadSeq( (CvSeq*)(subdiv->edges), &reader, 0 );

    cvCalcSubdivVoronoi2D( subdiv ); //Make sure Voronoi exists

    for( i = 0; i < total; i++ ) {

        CvQuadEdge2D* edge = (CvQuadEdge2D*)(reader.ptr);

        if( CV_IS_SET_ELEM( edge ) ) {

            // Do something with Voronoi and Delaunay edges ...
            //

            CvSubdiv2DEdge voronoi_edge = (CvSubdiv2DEdge)edge + 1;
            CvSubdiv2DEdge delaunay_edge = (CvSubdiv2DEdge)edge;

            // ...OR WE COULD FOCUS EXCLUSIVELY ON VORONOI...

            // left
            //
            voronoi_edge = cvSubdiv2DRotateEdge( edge, 1 );

            // right
            //
            voronoi_edge = cvSubdiv2DRotateEdge( edge, 3 );
        }
        CV_NEXT_SEQ_ELEM( elem_size, reader );
    }
}
```

最后，我们以一个方便的内联宏指令结束：只要我们找到了 Delaunay 三角的剖分顶点，就可以使用如下函数计算它的面积。

```
double cvTriangleArea(
    CvPoint2D32f a,
    CvPoint2D32f b,
```

```
CvPoint2D32f c  
})
```

【312】

练习

1. 使用 `cvRunningAvg()` 函数，再次实现背景去除的平均法。为了实现这一方法，需要知道场景中像素的均值漂移值从而得出绝对差分的均值和均值漂移，该均值漂移代替图像的标准偏差。
2. 阴影是背景减除中经常遇到的一个问题，因为它们常像前景目标一样出现。使用平均或背景去除的 `codebook` 方法学习背景。让一个人走进前景，这样阴影就从前景目标的底部映射出来。
 - a. 户外场景中，阴影比它周围更暗更蓝一些，可以利用这个事实来消除阴影。
 - b. 室内场景，阴影比它周围更暗一些，可以利用这个事实来消除阴影。
3. 本章介绍的简单背景模型对阈值参数很敏感。在第十章，我们将看到如何追踪运动轨迹，这可以作为“真实值”用来对背景模型及其阈值进行检查。当知道有人在摄像机前做“标定游走”时，也可以使用它来寻找运动目标，调整参数直到前景目标符合运动边界。已知背景的一部分遮挡时，我们可以使用标定物体本身的特定模式(或在背景上)做实际检测并且做参数调谐指导。
 - a. 更改代码，包含自动校准模型。学习背景模型，然后在场景中放置一个明亮颜色的目标。利用颜色特征找出目标，然后在背景程序中利用该目标自动设定阈值，以此分割目标。注意：可以将目标留在场景中以对阈值作连续调整。
 - b. 使用已修正的代码解答练习 2 阴影去除的问题。
4. 利用背景分割法分割一个伸开双臂的人，研究在 `find_connected_components()` 函数中使用不同参数和默认值时的影响。对如下参数设置不同的值，给出你的结果：
 - a. `poly1_hull0`
 - b. `perimScale`
 - c. `CVCONTOUR_APPROX_LEVEL`

- d. CVCLOSE_ITR
5. 在 2005 年 DARPA 机器人挑战竞赛中，斯坦福队的队员们用一种颜色聚类算法将公路与不是公路的区域分离开来。这些颜色从车前面的小片公路的激光探测的梯形里采样出来。图像中其他接近这小片颜色的颜色区域——以及原来梯形连接的部分颜色区域——被认为是公路。如图 9-18 所示，图中先用梯形标记公路，用倒“U”字标记公路的外边部分，然后用分水岭算法分割公路。假设我们能够自动产生这些标记，那么用这个方法分割公路会出现什么错误呢？提示：请认真观察图 9-8，然后思考在梯形中用一个像什么的东西来扩展公路梯形。

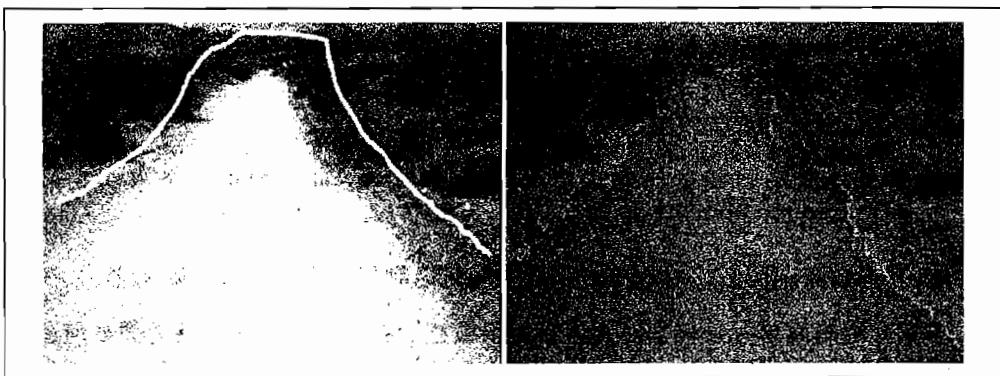


图 9-18：使用分水岭算法来识别路面。将标记置于原图中(左图)，算法分割出了路面(右图)

6. Inpainting 对修复纹理区域的划痕效果很好。如果图像中划痕使实际目标边缘模糊，那么将会是什么样的结果？试试看。
7. 虽然可能有点慢，但是当使用 `cvPyrMeanShiftFiltering()` 将视频输入预先分割时，试试使用背景分割。也就是说，输入视频先进行均值漂移分割，然后用 codebook 背景分割程序进行背景学习，之后测试前景。
- 与没有进行均值漂移分割的结果进行比较。
 - 系统地改变均值移动分割的 `ax_level`, `spatialRadius`, 和 `colorRadius` 等参数，比较结果。
8. Inpainting 以固定的笔迹对经过均值漂移分割的图像效果如何？试试不同的设置，给出结果。
9. 修改代码 `.../opencv/samples/delaunay.c`，加入点击鼠标选择点的位置(而不是通过现有的随机选中点的方法)，用三角化法对结果进行实验。

10. 再次修改 *delaunay.c* 代码，使之可以用键盘画点集的凸包。
11. 同一直线上的三点是否有 Delaunay 三角剖分？
12. 图 9-19(a)所示的三角形是一个 Delaunay 三角剖分吗？如果是，说明原因。如果不是，你怎么改变图形使它成为 Delaunay 三角剖分？
13. 对图 9-19(b)中的点手工实现 Delaunay 三角化，通过这个练习，你不用另外增加一个虚拟外接三角。

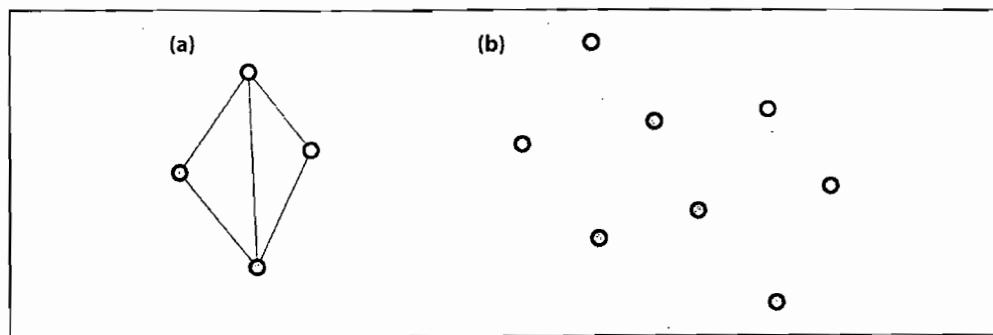


图 9-19：练习 12 和练习 13

跟踪与运动



跟踪基础

当我们在处理一段视频而非某张静止的图像时，通常视频中有一个或几个特定物体是我们想在整个视野范围内关注的。在前一章中，我们知道了如何在逐帧(frame-by-frame)分离出特定的形状，例如人或者车；现在，我们尝试理解这个物体的运动。理解物体的运动则主要包含两个部分：识别(identification)和建模。

识别指在视频流后续的帧中找出之前某帧中的感兴趣物体。前面章节中讲到的矩和颜色直方图可以帮助识别我们关注的物体。一个相关的问题就是跟踪不明物体。当我们需要根据运动来确定什么是感兴趣的，或者当物体的运动使得其成为感兴趣物体时，跟踪不明物体就很重要了。经典的跟踪不明物体的方法是跟踪视觉上重要的关键点，并不是整个物体。OpenCV 提供了两种方法实现跟踪关键点：Lucas-Kanade^①[Lucas81]和 Horn-Schunk [Horn81]方法。这两种方法分别代表了通常提到的稀疏(sparse)和稠密(dense)光流。

上述方法实际上只能给出物体实际位置的初步计算。理解物体运动的第二个部分，也就是建模，则可以帮助我们解决这个问题。为估计由这种粗略测量得出的物体的轨迹，人们设计了许多有力的数学方法。这些方法适用于二维和三维物体或物体位置的模型。

① 很奇怪，OpenCV 中实现的金字塔框架下的 Lucas-Kanade 光流方法是根据一篇未正式发表的论文实现的，这篇论文是 Bouguet 的 [Bouguet04]。

寻找角点

有多种局部特征可以用来进行跟踪。花一点时间去思考究竟角点(corner)用什么样的特征来描述是值得的。很明显，如果从一面很大的空墙上选择一个点，那么将很难从视频的下一帧中再找出这一点。如果墙上的所有点都是一样的或者是相似的，我们就不会有太好的运气能在随后的视频帧中跟踪到这个点了。相反，如果选择一个独一无二的点，那么再找到这点的几率就非常大。实际上，被选择的点或特征应该是独一无二的，或者至少接近独一无二，并且在与另一张图像的其他点可以进行参数化的比较。如图 10-1 所示。

【316~317】

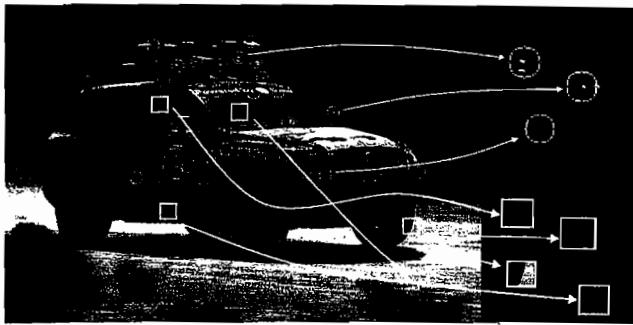


图 10-1：圆圈圈出的特征点是易于跟踪的点，而矩形圈出的特征——甚至是明显的边缘——却不然

回到那面大而空的墙，我们想要找本身具有明显变化的点——例如导数值比较明显的地方。虽然仅此是不够的，但这是一个开始。一个导数值比较明显的点可能在某种类型的边缘上，但却和此边缘上的其他点看起来一样(参考图 10-8 和“Lucas-Kanade 方法”一节中讨论的孔径问题)。

如果一个点在两个正交的方向上都有明显的导数，则我们认为此点更倾向是独一无二的，所以，许多可跟踪的特征点都称为角点。从直观上讲，角点(而非边缘)是一类含有足够信息且能从当前帧和下一帧中都能提取出来的点。

最普遍使用的角点定义是由 Harris[Harris88]提出的。定义的基础是图像灰度强度的二阶导数($\partial^2x, \partial^2y, \partial x \partial y$)矩阵。考虑到图像所有的像素点，我们可以想像图像的二阶导数即形成一幅新的“二阶导数图像”，或融合在一起形成一幅新的 Hessian 图像。这个术语源自于一个点的如下定义的二维 Hessian 矩阵：

$$H(p) = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix}_p$$

【317】

对于 Harris 角点，我们使用每点周围小窗口的二阶导数图像的自相关矩阵。这个自相关矩阵的定义如下：

$$M(x, y) = \begin{bmatrix} \sum_{-K \leq i, j \leq K} w_{i,j} I_x^2(x+i, y+j) & \sum_{-K \leq i, j \leq K} w_{i,j} I_x(x+i, y+j) I_y(x+i, y+j) \\ \sum_{-K \leq i, j \leq K} w_{i,j} I_x(x+i, y+j) I_y(x+i, y+j) & \sum_{-K \leq i, j \leq K} w_{i,j} I_y^2(x+i, y+j) \end{bmatrix}$$

这里 $w_{i,j}$ 是可以归一化的权重比例，但是通常被用作产生圆形窗口或高斯权重。Harris 定义的角点位于图像二阶导数的自相关矩阵有两个最大特征值的地方，这在本质上表示以此点为中心周围存在至少两个不同方向的纹理(或者边缘)，正如实际的角点是由至少两个边缘相交于一点而产生。之所以采用二阶导数是由于它对均匀梯度不产生响应^①。角点的这个定义还有另一个优点。被跟踪的物体在移动过程中也可能会旋转，找到同时对移动和旋转不变的量是很重要的。只考虑自相关矩阵的特征值可达到这个目的。这两个最大特征值不仅可以判定一个点是否一个好的可跟踪的特征点，同时也提供了对这个点进行识别的一个标识。

Harris 最原始的定义是将矩阵 $H(p)$ 的行列式值与 $H(p)$ 的迹(带权重系数)相减，再将差值同预先给定的阈值进行比较。后来 Shi 和 Tomasi[Shi94]发现，若两个特征值中较小的一个大于最小阈值，则会得到强角点。Shi 和 Tomasi 的方法比较充分，并且在很多情况下可以得到比使用 Harris 方法更好的结果。

函数 cvGoodFeaturesToTrack() 采用 Shi 和 Tomasi 提出的方法，先计算二阶导数(利用 Sobel 算子)，再计算特征值，它返回满足易于跟踪的定义的一系列点。

```
void cvGoodFeaturesToTrack(
    const CvArr*           image,
    CvArr*                 eigImage,
    CvArr*                 tempImage,
    CvPoint2D32f*           corners,
    int*                   corner_count,
    double                 quality_level,
    double                 min_distance,
```

① 梯度是一阶导数而来。若一阶导数是均匀的(常数)，则二阶导数为 0。

```
    CvArr*          mask      = NULL,  
    block_size     = 3,  
    use_harris    = 0,  
    k              = 0.4
```

【318】

数中，输入图像 `image` 须为 8 位或 32 位(也就是，`IPL_DEPTH_8U` 或者 `IPL_DEPTH_32F`)单通道图像。第二和第三个参数是大小与输入图像相同的 32 位单精度浮点数。参数 `tempImage` 和 `eigImage` 在计算过程中被当作临时变量使用，计算出的值在 `eigImage` 中是有效的。特别地，每个元素包含了输入图像中该点处的最小特征值。`corners` 是函数的输出，为检测到的 32 位(`CvPoint2D32f`)角点位置。在调用函数 `cvGoodFeaturesToTrack()` 之前需要给这个数组分配内存。`corner_count` 表示可以返回的最大角点数目。很明显，只能为此数组分配有限的内存。`corner_count` 表示可以返回的实际角点数目。函数调用结束后，`corner_count` 输出实际检测到的角点数目。`quality_level` 表示一点被认为是角点的可接受的最小特征值。实际用于过滤角点的最小特征值是 `quality_level` 与图像中最大特征值的乘积。所以 `quality_level` 的值不应超过 1(常用的值为 0.10 或者 0.01)。检测完所有的角点后，要进一步剔除掉一些距离较近的角点。`min_distance` 保证返回的角点之间至少相隔 `min_distance` 个像素。

参数 `mask` 是可选参数，是一幅像素值为布尔类型的图像，用于指定输入图像中参与角点检测的像素点。若 `mask` 的值设为 `NULL`，则选择整个图像。`block_size` 是计算自相关矩阵时指定点的领域，采用小窗口计算的结果比单点(也就是 `block_size` 为 1)计算的结果要好。若 `use_harris` 的值为非 0，则函数使用 Harris 角点定义；若为 0，则使用 Shi-Tomasi 的定义。当 `use_harris` 为 `k` 且 `k > 0` 时，`k` 为用于设置 Hessian 自相关矩阵即对 Hessian 行列式的相对权重的权重。

函数 `cvGoodFeaturesToTrack()` 的输出结果为角点的位置数组，我们往往会希望在另一幅相似的图像中寻找这些角点。就本章而言，我们关注的是在后续的视频帧中找到这些特征点，但是这个函数也有其他的应用，例如，对从微小不同的视角拍摄的多幅图像进行匹配。这个问题将在后面讨论立体视觉的小节中遇到。

亚像素级角点

如果我们进行图像处理的目的不是提取用于识别的特征点而是进行几何测量，则通

常需要更高的精度，而函数 `cvGoodFeaturesToTrack()` 只能提供简单的像素的坐标值，就是说，我们有时会需要实数坐标值而不是整数坐标值，例如(8.25, 117.16)。

您也许需要确定图像中一个尖锐的峰值点的位置，但是峰值的位置一般都不会恰好位于一个像素的正中心，使得确定其位置比较困难。要解决这个问题，需要先用一条曲线(例如一条抛物线)拟合图像的值，再用数学的方法计算位于像素之间的峰值点的位置。亚像素检测方法就是一些有关峰值点位置的计算技巧(了解更多此方面知识以及最新的方法，请参考 Lucchese [Lucchese02] 和 Chen [Chen05])。图像测量常用的领域为三维重建、摄像机标定、图像拼接以及在卫星图像中查找特定信号，如一栋建筑的精确位置。

【319~320】

亚像素级角点的位置在摄像机标定、跟踪并重建摄像机的轨迹或者重建被跟踪目标的三维结构时是一个基本的测量值。我们已经知道如何对整数网格像素来检测角点并得到角点的位置，下面所要讨论的是如何将求得的角点位置精确到亚像素级精度。一个向量和与其正交的向量的点积为 0，角点则满足这种情况，如图 10-2 所示。

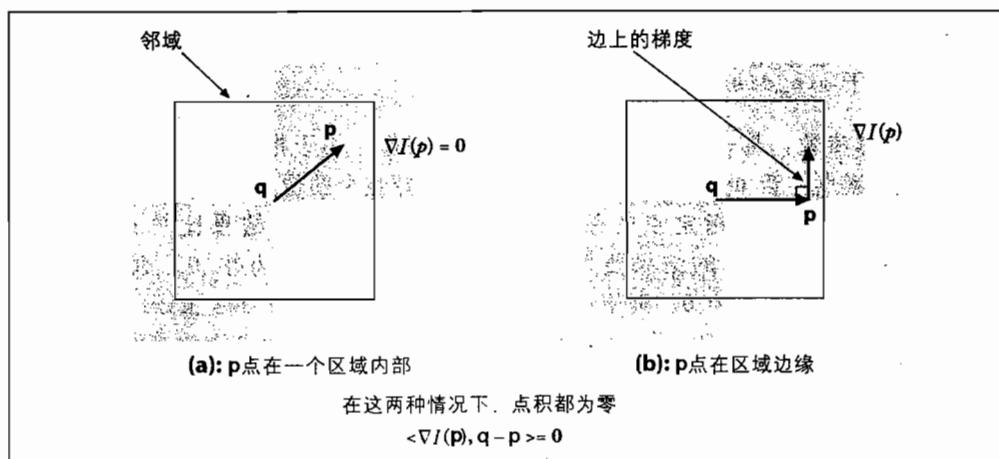


图 10-2：计算亚像素级精度的角点：(a)点 p 附近的图像是均匀的，其梯度为 0；(b)边缘的梯度与沿边缘方向的 q-p 向量正交。在图中两种情况下，p 点的梯度与 q-p 向量的点积均为 0(参考正文)

图 10-2 假设起始角点 q 在实际亚像素级角点的附近。检测所有的 $q-p$ 向量。若点 p 位于一个均匀的区域，则点 p 处的梯度为 0。若 $q-p$ 向量的方向与边缘的方向一致，则此边缘上 p 点处的梯度与 $q-p$ 向量正交，在这两种情况下， p 点处的梯度与 $q-p$ 向量的点积为 0。我们可以在 p 点周围找到很多组梯度以及相关的向量 $q-p$ ，令

其点集为 0，然后可以通过求解方程组，方程组的解即为角点 q 的亚像素级精度的位置，也就是精确的角点位置。

【320】

函数 `cvFindCornerSubPix()` 用于发现亚像素精度的角点位置。

```
void cvFindCornerSubPix(
    const CvArr*          image,
    CvPoint2D32f*         corners,
    int                  count,
    CvSize                win,
    CvSize                zero_zone,
    CvTermCriteria        criteria
);
```

输入图像 `image` 是 8 位单通道的灰度图像。`corners` 为整数值的像素位置，例如是由 `cvGoodFeatures()` 得到的位置坐标。`corners` 设定了角点的初始位置。`count` 为需要计算的角点数目。

实际计算亚像素级的角点位置时，解的是一个点积的表达式为 0 的方程组(参考图 10-2)，其中每一个方程都是由 q 邻域的一个点产生。`win` 指定了等式产生的窗口的尺寸。搜索窗口的中心是整数坐标值的角点并从中心点在每个方向上扩展 `win` 指定的像素(也就是说，如果 `win.width = 4`，则实际的窗口大小为 $4+1+4=9$ 个像素宽)。这些等式构成一个可用自相关矩阵的逆(非前文讨论 Harris 角点时涉及的自相关矩阵)来求解的线形方程组。实际上，由于非常接近 p 的像素产生了很小的特征值，所以这个自相关矩阵并不总是可逆的。为了解决这个问题，一般可以简单地剔除离 p 点非常近的像素。输入参数 `zero_zone` 定义了一个禁区(与 `win` 相似，但通常比 `win` 小)，这个区域在方程组以及自相关矩阵中不被考虑。如果不需要这样一个禁区，则 `zero_zone` 应设置为 `cvSize(-1, -1)`。

当找到一个 q 的新位置时，算法会以这个新的角点作为初始点进行迭代直到满足用户定义的迭代终止条件。迭代过程的终止条件可以是最大迭代次数 `CV_TERMCRIT_ITER` 类型，或者是设定的精度 `CV_TERMCRIT_EPS` 类型(或者是两者的组合)。终止条件的设置在极大程度上影响最终得到的亚像素值的精度。例如，如果指定 0.10，则求得的亚像素级精度为像素的十分之一。

不变特征

继 Harris 提出角点以及后来 Shi 和 Tomasi 提出角点后，许多其他类型的角点和相关局部特征点也被提出来。SIFT(“scale-invariant feature transform”)是其中一种

广泛应用的类型[Lowe04]。SIFT 特征正如其名称一样是缩放不变的。SIFT 在一点处检测主要梯度方向，根据这个方向记录局部梯度直方图结果，所以 SIFT 也是旋转不变的。正是由于这些特点，SIFT 特征在小的仿射变换中有相对不错的表现。虽然 OpenCV 库中还没有实现 SIFT 算法(参考第 14 章)，但可以用 OpenCV 的基本函数实现它。我们将不在这个问题上花更多的时间，但需要记住的是，用前面已经介绍过的 OpenCV 函数可以实现计算机视觉领域的文献中提出的绝大部分特征。

【321 ~ 322】

光流

如前所述，我们经常不知道任何关于视频内容的先验知识，但是需要估计两帧之间(或一个帧序列)的运动。运动本身就说明了存在运动的感兴趣的目标。在图 10-3 中解释了光流。

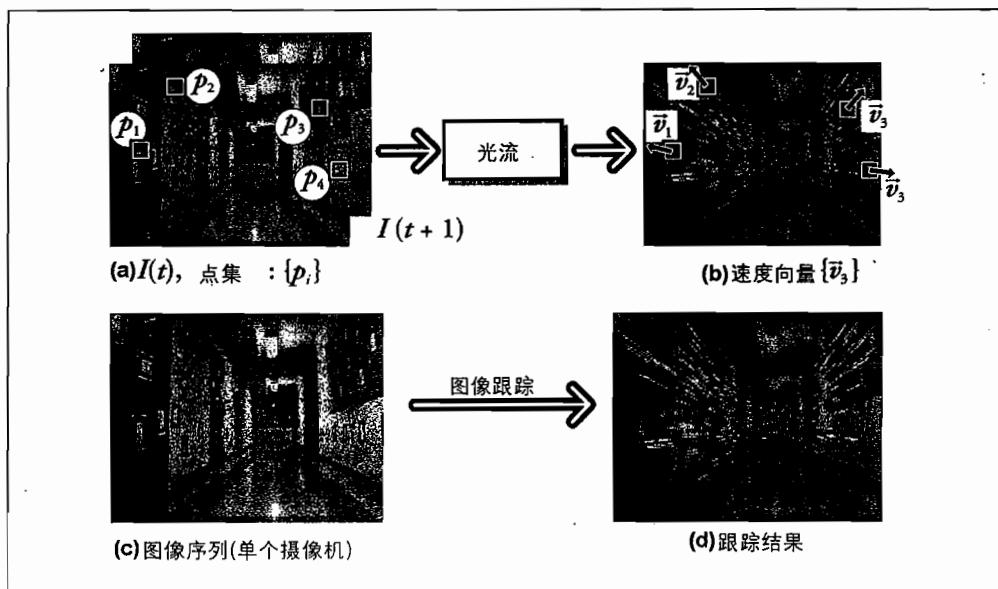


图 10-3：光流。目标特征点(a)被连续跟踪，它们的运动被转换成速度矢量(b)；下面的两幅图分别表示一幅走廊的图像(c)和摄像机沿着走廊运动时的光流矢量(d)(原图由 Jean-Yves Bouguet 提供)

可以将图像中的每个像素与速度关联，或者等价地，与表示像素在连续两帧之间的位移关联。这样得到的是稠密光流(dense optical flow)，即将图像中的每个像素都

与速度关联。Horn-Schunck 方法[Horn81]计算的就是稠密光流的速度场。OpenCV 中实现了一种比较简单直接的方法，即对前后连续两帧的一个像素的邻域进行匹配。这种方法被称为块匹配(block matching)。这两种方法都会在“稠密跟踪方法”一节中讨论。

【322~323】

实际上计算稠密光流并不容易。我们先来看看一张白纸的运动。上一帧中的白色的像素在下一帧中仍然为白色，而只有边缘的像素并且是与运动方向垂直的像素才会产生变化。稠密光流的方法需要使用某种插值方法在比较容易跟踪的像素之间进行插值以解决那些运动不明确的像素，从中可以清楚地看到稠密光流相当大的计算量。

于是我们找到了另一种方法，稀疏光流(sparse optical flow)。稀疏光流的计算需要在被跟踪之前指定一组点。如果这些点具有某种明显的特性，例如前面讲到的“角点”，那么跟踪就会相对稳定和可靠。OpenCV 可以帮助我们找到最适合跟踪的特征点。在很多的实际应用中，稀疏跟踪的计算开销比稠密跟踪小得多，以至于后者只有在理论中研究^①。

下面的几节介绍了几种不同的跟踪的方法。我们从最流行的稀疏跟踪方法 Lucas-Kanade(LK)光流开始介绍；这种方法与图像金字塔一起，可以跟踪更快的运动。接下来介绍两种稠密跟踪方法，Horn-Schunck 方法和块匹配方法。

Lucas-Kanade 方法

Lucas-Kanade 算法(LK)[Lucas81]是用于求稠密光流的，最初于 1981 年提出。由于算法易于应用在输入图像中的一组点上，其后来成为求稀疏光流的一种重要方法。LK 算法只需要每个感兴趣点周围小窗口的局部信息，所以它可以应用于稀疏内容，这与 Horn 和 Schunck 的算法全局性是不同的(后面会详细讲述)。但是，使用小窗口的 LK 算法存在不足之处，较大的运动会将点移出这个小窗口，从而造成算法无法再找到这些点。金字塔的 LK 算法可以解决这个问题，即从图像金字塔的最高层(细节最少)开始向金字塔的低层(丰富的细节)进行跟踪。跟踪图像金字塔允许小窗口捕获较大的运动。

这个算法是一个重要而有效的方法，我们将会具体介绍其中的数学问题。不想阅读

① Black 和 Anandan 提出的计算稠密光流的方法[Black93; Black96]被广泛用在电影制作中，因为为了追求视觉的质量，电影工作室愿意牺牲时间开销来获得详细的光流信息。OpenCV 未来的版本中会将这些方法包含进去(参考第 14 章)。

这些细节的读者可以直接阅读函数描述和代码部分，但是建议读者至少浏览一遍其中描述 Lucas-Kanade 光流假设条件的文字和插图部分，这样在失去头绪的时候知道应该如何去解决。

Lucas-Kanade 算法原理

LK 算法基于以下三个假设。

- (1) 亮度恒定。图像场景中目标的像素在帧间运动时外观上保持不变。对于灰度图像(LK 算法也可用于彩色图像)，需要假设像素被逐帧跟踪时其亮度不发生变化。
- (2) 时间连续或者运动是“小运动”。图像的运动随时间的变化比较缓慢。实际应用中指的是时间变化相对图像中运动的比例要足够小，这样目标在帧间的运动就比较小。
- (3) 空间一致。一个场景中同一表面上邻近的点具有相似的运动，在图像平面上的投影也在邻近区域。

图 10-4 说明了这三点假设如何构建一个有效的跟踪算法。第一点假设：亮度恒定，指对被跟踪部分(patch)像素不随时间变化的要求：

$$f(x, t) \equiv I(x(t), t) = I(x(t + dt), t + dt)$$

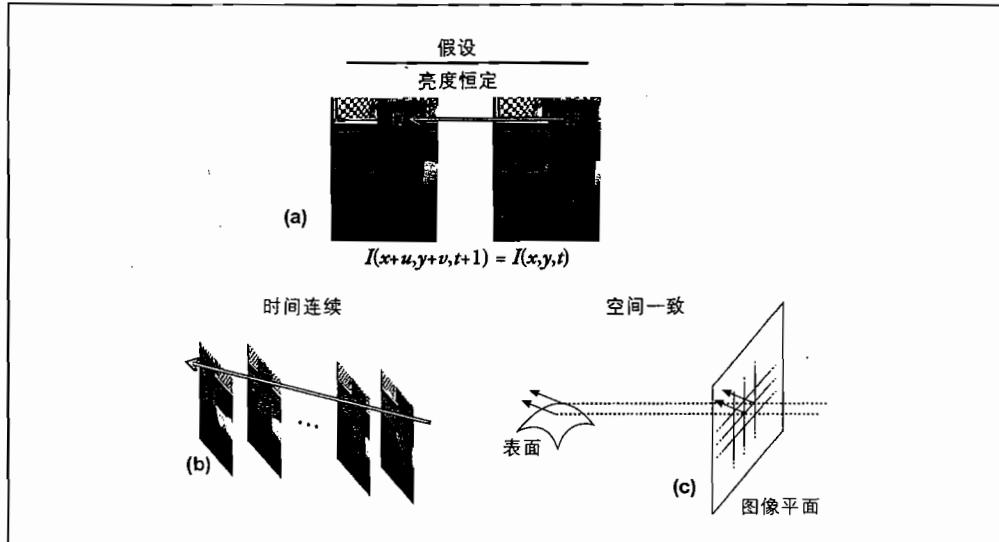


图 10-4: Lucas-Kanade 光流假设：(a)场景中物体被跟踪的部分的亮度不变；
(b)运动相对于帧率是缓慢的；(c)[相邻的点保持相邻(图由 Michael Black
[Black82]提供)]

【324】

很简单，公式的意思是被跟踪像素的灰度不随时间变化而变化：

$$\frac{\partial f(x)}{\partial t} = 0$$

第二点假设：时间连续，指相邻帧之间的运动较小。换句话，可以将运动的变化看成是亮度对时间的导数(即断言一个序列中相邻帧间的变化是小微分的)。我们先来看一下一维空间的例子，以理解第二点假设所蕴含的意思。

由亮度恒定的等式开始，考虑隐含的 x 为 t 的函数这点，将亮度的定义 $f(x, t)$ 用 $I(x(t), t)$ 替换，再应用偏微分的链式规则(chain rule)，即：

$$\underbrace{\frac{\partial I}{\partial x}}_{I_x} \left(\underbrace{\frac{\partial x}{\partial t}}_v \right) + \underbrace{\frac{\partial I}{\partial t}}_{I_t}_{|_{x(t)}} = 0$$

其中， I_x 是图像的偏导数， I_t 是图像随时间的导数， v 是要求的速度。简单一维空间中的光流速度的等式为：

$$v = -\frac{I_t}{I_x}$$

下面我们对一维跟踪问题进行一些直观的解释。图 10-5 表示的是左边值大右边值小、沿 x 轴向右运动的一个“边缘”。我们要求出图的上半部分标出的边缘运动的速度 v 。在图的下半部分，对这个速度的测量正是“rise over run”，即 rise 是随时间变化的而 run 是斜率(空间的导数)。负号纠正了 x 的斜率。

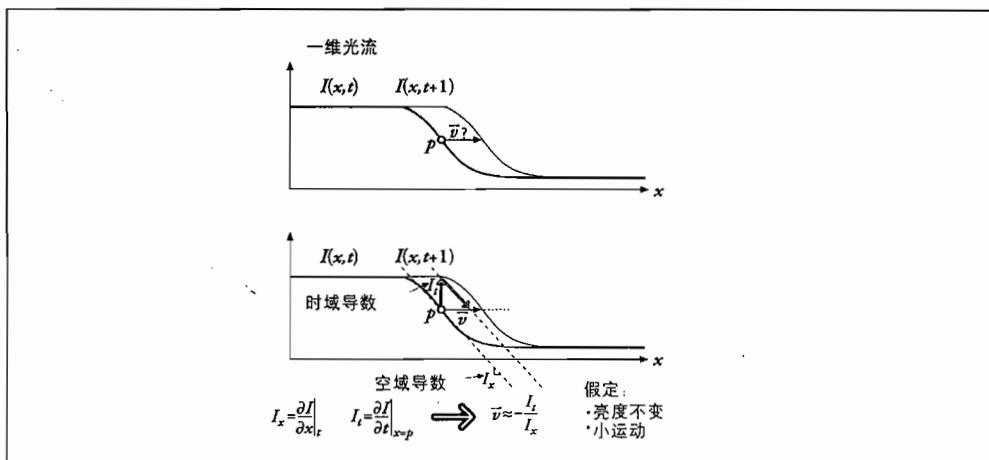


图 10-5：一维空间中的 Lucas-Kanade 光流。通过计算亮度对时域导数与亮度对空域导数的比值，可以估计出运动边缘(图的上半部分)的速度

图 10-5 揭示了光流公式的另一个问题：我们的假设可能不是十分正确，即图像的亮度实际上不是恒定的，时间步长(由摄像机设定)也不是如我们期望的相对于运动足够短。所以我们求解的速度并不准确。但是，如果求解到的速度与实际的速度“足够接近”，就可以用迭代的方法来解决这个问题，如图 10-6 所示，将第一次估计的速度(不准确的)作为初始值进行下一次迭代并重复这个过程。根据沿 x 运动的像素不变的亮度恒定假设，在迭代过程中，可以保持使用由第一帧计算得到的 x 的空间导数。这种重复使用已经计算出的空间导数极大地节省了计算开销。时间导数仍需要在每次迭代和每一帧中重新计算，但如果初始值与实际值足够接近，迭代过程在 5 次迭代之内就会收敛到接近的准确值，这就是所谓的牛顿法。如果初始估计值与实际值相差比较大，则牛顿法实际上也是发散的。

【325~326】

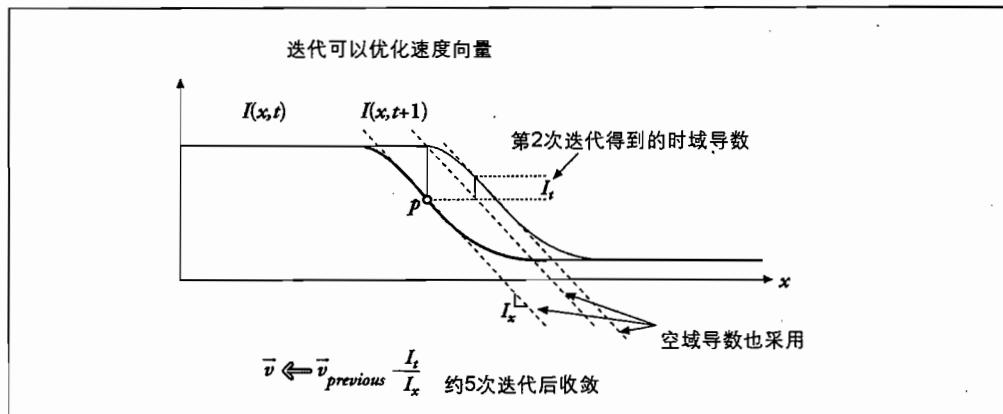


图 10-6：通过迭代来优化光流的求解(牛顿法)。使用同样的两个图像和同样的空域导数，重新求解时域导数，通常可通过几次迭代收敛到稳定解

前面是 LK 算法在一维空间的用法，下面我们将其扩展应用到二维空间的图像上。乍看起来好像比较简单，只需要加入 y 坐标即可。改变一下符号，速度的 y 分量为 v ， x 分量为 u ，则得到：

$$I_x u + I_y v + I_t = 0$$

不幸的是，这个等式对任意一个像素都含有两个未知量。这就是说对于单个像素，等式的约束条件过少，不能得到此点二维运动的定解。我们只能求得与光流方程描述的方向垂直的运动分量。图 10-7 给出了数学和几何的细节。

垂直光流由孔径问题产生，即用小孔或小窗口去测量运动。这种情况下，我们通常只能观测到边缘而观测不到角点，而只依靠边缘是不足以判断整个物体是如何运动(也就是朝哪个方向运动)的，如图 10-8 所示。

从一维到二维光流

公式

但是，只能求得一条直线，而不是一个点

$$I_x u + I_y v + I_t = 0$$

$$\nabla I^T u = -I_t$$

$$u = \begin{bmatrix} u \\ v \end{bmatrix}, \nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

$$I_x u + I_y v = -I_t$$

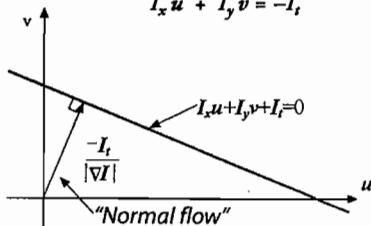


图 10-7：单个像素的二维光流：单个像素的光流是无法求解的，最多只能求出光流方向，因为光流方向与光流方程描述的线垂直(图由 Michael Black 提供)

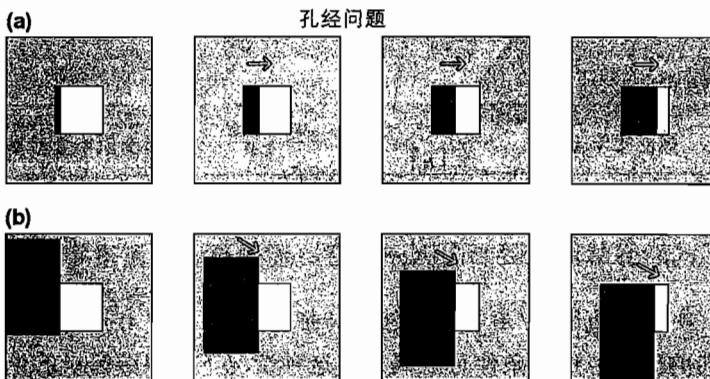


图 10-8：孔径问题：从 aperture window(a)我们可以观测到边缘向右运动，但是无法观察到边缘也在向下运动(b)

那么，如何解决单个像素不能求解整个运动的问题呢？这时需要利用光流的最后一点假设。若一个局部区域的像素运动是一致的，则可以建立邻域像素的系统方程来求解中心像素的运动。例如，如果用当前像素 5×5 邻域的像素的亮度值(彩色像素的光流只需增加两倍)来计算此像素的运动，则可以建立如下的 25 个方程。

【326 ~ 327】

① 窗口可以是 3×3 、 7×7 或者任何被指定的值。若窗口太大则会由于违背运动一致的假设而不能进行较好的跟踪。若窗口太小，则又会产生孔径问题。

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}_{\frac{A}{25 \times 2}} \begin{bmatrix} u \\ v \end{bmatrix}_{\frac{d}{2d}} = \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}_{\frac{b}{25 \times 1}}$$

现在我们得到一个约束条件过多的系统方程，若在 5×5 的窗口中包含两条或以上边缘则可以解此系统方程。为了解这个系统方程，需要建立一个该方程的最小平方，通过下面方程来求解最小化的 $\|Ad - b\|^2$ ：

$$\underbrace{(A^T A)}_{\geq 2} \underbrace{d}_{2d} = \underbrace{A^T b}_{\geq 2}$$

由这个关系式可以得到 u 和 v 运动分量。这个关系更详细的表述如下：

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix}_{\tilde{A}^T \tilde{A}} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}_{\tilde{A}^T b}$$

当 $(A^T A)$ 可逆时，方程的解如下：

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b$$

【328】

当 $(A^T A)$ 满秩(秩为 2)也即 $(A^T A)$ 有两个较大特征向量时， $(A^T A)$ 可逆。图象中纹理至少有两个方向的区域，这个条件可以满足。这种情况下，跟踪窗口的中心在图象的角点区域时， $(A^T A)$ 的特性最好。由这一点我们可以联系到前面讨论的 Harris 角点检测器。事实上，正因为 $(A^T A)$ 在角点处有两个大的特征向量，所以这些角点是“可用于跟踪的良好特征点”(参考前面对 cvGoodFeaturesToTrack() 的介绍)，后文即将介绍的 cvCalcOpticalFlowLK() 实现了这个计算。

理解了小而连贯运动的假设的读者现在会感到不解：对于大多数 30Hz 的摄像机，大而不连贯的运动是普遍存在的。而 Lucas-Kanade 光流正因为这个原因在实际中的跟踪效果并不是很好：我们需要一个大的窗口来捕获大的运动，而大窗口往往违背运动连贯的假设！图象金字塔可以解决这个问题，即最初在较大的空间尺度上进行跟踪，再通过对图象金字塔向下直到图象像素的处理来修正初始运动速度的假定。

所以，建议的跟踪方法是：在图象金字塔的最高层计算光流，用得到的运动估计结

果作为下一层金字塔的起始点，重复这个过程直到到达金字塔的最底层。这样就将不满足运动假设的可能性降到最小从而实现对更快和更长的运动的跟踪。这个更精致的算法叫金字塔 Lucas-Kanade 光流，其原理在图 10-9 中描述。实现金字塔 Lucas-Kanade 光流的函数是 `cvCalcOpticalFlowPyrLK()`，后面将介绍这个函数。

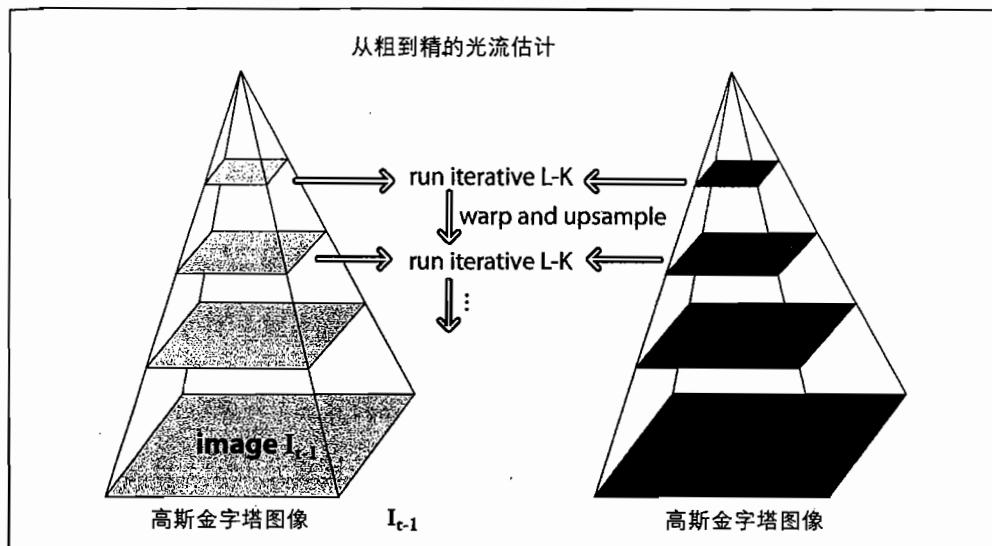


图 10-9：金字塔 Lucas-Kanade 光流。为了减轻由小而连贯的运动假设引起的问题，首先在金字塔顶层计算光流；在上一次估计到的运动将作为下一层的起始点，进行进一步估计

Lucas-Kanade 代码

下面的函数实现了非金字塔的 Lucas-Kanade 稠密光流算法：

```
void cvCalcOpticalFlowLK(  
    const CvArr*     imgA,  
    const CvArr*     imgB,  
    CvSize          winSize,  
    CvArr*          velx,  
    CvArr*          vely  
) ;
```

OpenCV 这个函数的输出只记录了可以计算最小误差的像素。对于最小误差不能被可靠地计算出的像素，相关的速度被设置为 0。多数情况下我们不会使用这个函数，而用下面介绍的基于图像金字塔的方法。

金字塔 Lucas-Kanade 代码

现在来介绍 OpenCV 中在图像金字塔中计算 Lucas-Kanade 光流的算法 cvCalcOpticalFlowPyrLK()。我们将会看到，这个计算光流的函数使用了“易于跟踪的特征点”并返回每个点被跟踪的情况。

【329~330】

```
void cvCalcOpticalFlowPyrLK(
    const CvArr* imgA,
    const CvArr* imgB,
    CvArr* pyrA,
    CvArr* pyrB,
    CvPoint2D32f* featuresA,
    CvPoint2D32f* featuresB,
    int count,
    CvSize winSize,
    int level,
    char* status,
    float* track_error,
    CvTermCriteria criteria,
    int flags
);
```

此函数有很多输入参数，我们现花一点时间来弄清楚它们是什么。一旦掌握了这个函数，我们就可以到下一个问题，即跟踪哪些点和怎样计算。

cvCalcOpticalFlowPyrLk() 的前两个参数代表初始图像和最终图像，两幅图像都是 8 位的单通道图像。第三、四个参数是申请存放两幅输入图像(pyrA 和 pyrB)金字塔的缓存。这两个缓存的大小至少为 $(\text{img}.width+8)*\text{img}.height/3$ 个字节^①。(如果这两个指针设置为空，则当函数被调用时会自动分配合适的内存，使用并释放之，但这样会降低函数的执行效率。)featuresA 数组存放的是用于寻找运动的点，featuresB 与 featuresA 相似，存放 featuresA 中点的新位置；count 是 featuresA 中点的数目。win_size 定义了计算局部连续运动的窗口尺寸。参数 level 用于设置构建的图像金字塔的栈的层数。若 level 设置为 0 则不使用金字塔。数组 status 的长度是 count；函数调用结束时，status 中的每个元素被置 1(对应点在第二幅图像中被发现)或 0(对应点在第二幅图像中未被发现)。参数 track_error 为可选参数，它是表示被跟踪点的原始图像小区域与此点在第二幅图像的小区域间的差的数组。track_error 可以删除那些局部外观小区域随点的

① 分配这个大小的缓存是因为演算空间不仅存放图像本身，还要存放整个金字塔。

运动变化剧烈的点。

我们还需要一个迭代终止的条件，即 `criteria`。`CvTermCriteria` 是被许多含有迭代过程的 OpenCV 算法使用的一个结构：

```
cvTermCriteria(  
    int type, // CV_TERMCRIT_ITER, CV_TERMCRIT_EPS, or both  
    int max_iter,  
    double epsilon  
);
```

我们使用 `cvTermCriteria()` 函数来生成需要的这个结构。函数的第一个参数为 `CV_TERMCRIT_ITER` 或者 `CV_TERMCRIT_EPS`。这个参数分别告诉算法是根据迭代次数来终止迭代过程，还是根据收敛误差是否达到一个较小值来终止迭代。后面的两个参数则设置终止算法的一个或两个准则的值。有两个选项的原因使我们可以设置迭代终止类型为 `CV_TERMCRIT_ITER|CV_TERMCRIT_EPS`，这样当任一条件满足时迭代终止(大多数的实际代码就是这样设置的)。

最后一个参数 `flags` 允许对函数内部 bookkeeping 进行一些细微的控制，它可以设置为下面的任何一个或全部(使用位 OR 运算)。

- **CV_LKFLOW_PYR_A_READY** 在调用和存储到 `pyrA` 之前，先计算第一帧的金字塔。
- **CV_LKFLOW_PYR_B_READY** 在调用和存储到 `pyrB` 之前，先计算第二帧的金字塔。
- **CV_LKFLOW_INITIAL_GUESSES** 在函数调用之前，数组 `B` 已包含特征点的初始坐标值。

【331~332】

这些标志特别是在处理视频时会用到。图像金字塔的计算量较大，所以应该尽可能避免重复计算金字塔。计算得到的图像对的后面一帧被作为下次计算的图像对的初始帧。如果调用者为函数分配了缓存(而不是让函数在内部分配)，则函数返回时，每幅图像的金字塔将被存储在其中。告诉函数金字塔已经被计算，则函数不会再进行重复计算。类似地，如果从前一帧中计算得到点的运动，则在计算这些点在下一帧中的位置就有了一个好的初始估计。

应用的基本过程很简单：输入图像，在 `featuresA` 中列出需要跟踪的点，然后调用函数。函数返回后，检查 `status` 数组以确定哪些点被成功跟踪，再检查 `featuresB` 得到这些点新的位置。

现在我们再来讨论前面未讨论的问题：如何确定哪些特征点是易于被跟踪的。前面介绍过 OpenCV 函数 `cvGoodFeaturesToTrack()`，这个函数使用 Shi 和 Tomasi 提出的方法可靠地解决了这个问题。在多数情况下，使用 `cvGoodFeaturesToTrack()` 和 `cvCalcOpticalFlowPyrLK()` 的组合可以获得很好的结果。当然，读者可以使用自己的准则来确定被跟踪的点。

下面来看一个简单的使用 `cvGoodFeaturesToTrack()` 和 `cvCalcOpticalFlowPyrLK()` 的组合的例子(例 10-1)；也可参考图 10-10。

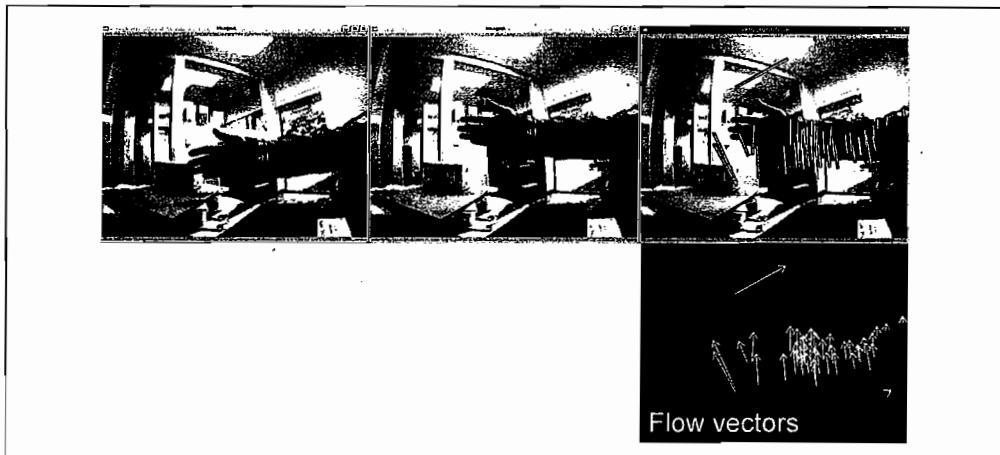


图 10-10：金字塔 Lucas-Kanade 稀疏光流。中间的图像是左边图像的下一帧视频；右边的图像表示计算出的“易于跟踪特征点”的运动(右下图为了增加可视性在黑色背景中标识出光流矢量)

例 10-1：金字塔 Lucas-Kanade 光流代码

```
// Pyramid L-K optical flow example
//
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

const int MAX_CORNERS = 500;

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

    // Initialize, load two images from the file system, and
    // allocate the images and other structures we will need for
```

```

// results.
//
IplImage* imgA=cvLoadImage("image0.jpg",CV_LOAD_IMAGE_GRAYSCALE);
IplImage* imgB=cvLoadImage("image1.jpg",CV_LOAD_IMAGE_GRAYSCALE);

CvSize img_sz = cvGetSize( imgA );
int win_size = 10;

IplImage* imgC = cvLoadImage(
"../Data/OpticalFlow1.jpg",
CV_LOAD_IMAGE_UNCHANGED
);
// The first thing we need to do is get the features
// we want to track.
//
IplImage* eig_image = cvCreateImage( img_sz, IPL_DEPTH_32F, 1 );
IplImage* tmp_image = cvCreateImage( img_sz, IPL_DEPTH_32F, 1 );

int corner_count = MAX_CORNERS;
CvPoint2D32f* cornersA = new CvPoint2D32f[ MAX_CORNERS ];

cvGoodFeaturesToTrack(
    imgA,
    eig_image,
    tmp_image,
    cornersA,
    &corner_count,
    0.01,
    5.0,
    0,
    3,
    0,
    0.04
);

cvFindCornerSubPix(
    imgA,
    cornersA,
    corner_count,
    cvSize(win_size,win_size),
    cvSize(-1,-1),

```

```

    cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03)
};

// Call Lucas Kanade algorithm
//
char features_found[ MAX_CORNERS ];
float feature_errors[ MAX_CORNERS ];

CvSize pyr_sz = cvSize( imgA->width+8, imgB->height/3 );

IplImage* pyrA = cvCreateImage( pyr_sz, IPL_DEPTH_32F, 1 );
IplImage* pyrB = cvCreateImage( pyr_sz, IPL_DEPTH_32F, 1 );

CvPoint2D32f* cornersB = new CvPoint2D32f[ MAX_CORNERS ];

cvCalcOpticalFlowPyrLK(
    imgA,
    imgB,
    pyrA,
    pyrB,
    cornersA,
    cornersB,
    corner_count,
    cvSize( win_size,win_size ),
    5,
    features_found,
    feature_errors,
    cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 ),
    0
);

// Now make some image of what we are looking at:
//
for( int i=0; i<corner_count; i++ ) {
    if( features_found[i]==0|| feature_errors[i]>550 ) {
        printf("Error is %f/n",feature_errors[i]);
        continue;
    }
    printf("Got it/n");
    CvPoint p0 = cvPoint(
        cvRound( cornersA[i].x ),

```

```

    cvRound( cornersA[i].y )
);

CvPoint p1 = cvPoint(
    cvRound( cornersB[i].x ),
    cvRound( cornersB[i].y )
);
cvLine( imgC, p0, p1, CV_RGB(255,0,0),2 );
}

cvNamedWindow("ImageA",0);
cvNamedWindow("ImageB",0);
cvNamedWindow("LKpyr_OpticalFlow",0);
cvShowImage("ImageA",imgA);
cvShowImage("ImageB",imgB);
cvShowImage("LKpyr_OpticalFlow",imgC);

cvWaitKey(0);

return 0;
}

```

稠密跟踪方法

OpenCV 中还有另外两种光流方法，但现在已经很少用到了。这些函数比 Lucas-Kanade 方法慢很多；另外，它们不支持图像金字塔匹配而不能用于跟踪大幅度的运动。本节我们简单讨论一下这些函数。

【332~334】

Horn-Schunck 方法

Horn 和 Schunck 于 1981 年提出这种方法。此方法是首次使用亮度恒定假设和推导出基本的亮度恒定方程的方法之一。Horn 和 Schunck 求解方程方法是假定一个速度 v_x 和 v_y 的平滑约束。该约束是通过对光流速度分量的二阶导数进行规则化获得：

$$\frac{\partial}{\partial x} \frac{\partial v_x}{\partial x} - \frac{1}{\alpha} I_x (I_x v_x + I_y v_y + I_t) = 0$$

$$\frac{\partial}{\partial y} \frac{\partial v_y}{\partial y} - \frac{1}{\alpha} I_y (I_x v_x + I_y v_y + I_t) = 0$$

这里 α 是不变的权重系数，称为规则化常数(regularization constant)。 α 的值较大可

以获得更平滑(也就是更局部一致)的运动流向量。这是强制进行平滑的一个简单的约束，其效果是惩罚光流变化剧烈的区域。与 Lucas-Kanade 算法一样，Horn-Schunck 方法也要通过迭代来解微分方程。

```
void cvCalcOpticalFlowHS(
    const CvArr* imgA,
    const CvArr* imgB,
    int usePrevious,
    CvArr* velx,
    CvArr* vely,
    double lambda,
    CvTermCriteria criteria
);
```

【335~336】

其中 `imgA` 和 `imgB` 必须是 8 位的单通道图像。`velx` 和 `vely` 存放 x 和 y 方向的速度，它们必须是 32 位的浮点数单通道图像。参数 `usePrevious` 指定算法使用从前一帧计算的 `velx` 和 `vely` 速度作为计算新的速度的初始值。参数 `lambda` 是与 Lagrange 乘子相关的权重。读者可能会问：“什么是 Lagrange 乘子？”^① Lagrange 乘子出现在当我们(同时)最小化运动-亮度方程和平滑方程时；它表示当最小化时赋予每一个方程误差的相对权重。

块匹配方法

读者可能会想：“光流到底有什么用？只是匹配上一帧和下一帧的像素而已。”有一些人的确是这样用的。“块匹配”是对一类将图像分割成被称为“块”[Huang95; Beauchemin95]的小区域的相似算法的统称。典型的块是正方形的，包含了一定数目的像素。这些块可以重叠，实际中它们通常是重叠的。块匹配算法将前一帧图像和当前图像划分成小块，然后计算这些块的运动。这一类算法在视频压缩算法和计算机视觉的光流中扮演重要角色。

块匹配算法对像素的集合进行处理而非单个像素，所以返回的“速度图像”通常比输入图像的分辨率低，但也有例外，这取决于块之间的重叠程度。下面的公式给出了结果图像的尺寸：

① 这种情况下最好忽略这里的描述而设置 `lambda` 为 1。

$$W_{result} = \left\lfloor \frac{\frac{W_{prev} - W_{block} + W_{shiftsize}}{W_{shiftsize}}}{floor} \right\rfloor$$

$$H_{result} = \left\lfloor \frac{\frac{H_{prev} - H_{block} + H_{shiftsize}}{H_{shiftsize}}}{floor} \right\rfloor$$

OpenCV 中的实现算法用螺旋搜索，找出(前一帧图像的)原始块的位置，然后再将候选的新块同原始块进行比较。比较的结果是像素绝对差值的和(也就是 L1 距离)。找到很好的匹配后就终止搜索。函数原型如下：

【336】

```
void cvCalcOpticalFlowBM(
    const CvArr*      prev,
    const CvArr*      curr,
    CvSize            block_size,
    CvSize            shift_size,
    CvSize            max_range,
    int               use_previous,
    CvArr*           velx,
    CvArr*           vely
);
```

参数表示的意思简明易懂。参数 `prev` 和 `curr` 是前一帧图像和当前图像，它们都是 8 位单通道的图像。`block_size` 是块的尺寸，`shift_size` 是块间移动的步长(这个参数决定了块与块之间是否重叠以及如果重叠，重叠的程度又如何)。参数 `max_range` 是一个块周围邻域的尺寸，函数在下一帧中搜索此邻域以找到对应的块。如果设置了参数 `use_previous`，则 `velx` 和 `vely` 的值将作为块搜索的初始值^①。参数 `velx` 和 `vely` 是存储计算得的块的运动的 32 位的单通道图像。如前文所说，运动的计算是在块级上进行的，所以输出结果的图像的坐标是对块而言的(就是说，像素的集合体)，而不是原始图像中的每个像素。

mean-shift 和 camshift 跟踪

这一节中将要介绍两种跟踪方法，`mean-shift` 和 `camshift`(“camshaitf”是“continuously adaptive mean-shift”的缩写)。前者是可用于多种应用的通用的数据

① 如果 `use_previous == 0`，块搜索区域与原始块的位置的距离为 `max_range`。如果 `use_previous != 0`，则块搜索区域的中心将被偏移 $\Delta x = vel_x(x, y)$ 和 $\Delta y = vel_y(x, y)$ 。

分析方法(在第 9 章的分割问题中讨论过), 计算机视觉正是这些应用之一。在介绍 mean-shift 的基本原理之后, 会介绍 OpenCV 是如何用它在图像中进行跟踪运动的。后一种方法 camshift 是建立在 mean-shift 之上, 它可以跟踪视频中尺寸可能产生变化的目标。

mean-shift

mean-shift 算法^①是一种在一组数据的密度分布中寻找局部极值的稳定的方法。若分布是连续的, 处理过程就比较容易, 这种情况下本质上只需要对数据的密度直方图应用爬山算法即可^②。然而, 对于离散的数据集, 这个问题在某种程度上是比较麻烦的。

【337】

这里用“稳定”这个形容词是指统计意义上的稳定, 因为 mean-shift 忽略了数据中的 outliers, 即忽略远离数据峰值的点。mean-shift 仅对数据局部窗口中的点进行处理, 处理完成后再移动窗口。

mean-shift 算法的步骤如下。

- (1) 选择搜索窗口。
 - 窗口的初始位置;
 - 窗口的类型(均匀、多项式、指数或者高斯类型);
 - 窗口的形状(对称的或歪斜的, 可能旋转的, 圆形或矩形);
 - 窗口的大小(超出窗口大小则被截去)。
- (2) 计算窗口(可能是带权重的)的重心。
- (3) 将窗口的中心设置在计算出的重心处。
- (4) 返回第(2)步, 直到窗口的位置不再变化(通常会)^③。

mean-shift 算法更加正式的描述是, mean-shift 算法与核密度估计的规则有关。

-
- ① mean-shift 是一个相当深奥的话题, 我们在这里讨论的目的主要是使读者有个直观的印象。参考 Fukunaga[Fukunaga90]、Comaniciu 和 Meer[Comaniciu99]可以了解 mean-shift 的由来。
 - ② 在这里使用“本质上”这个词是由于 mean-shift 会受尺度变化的影响。准确说就是: mean-shift 等价于先对连续分布用 mean-shift 核进行卷积, 然后再应用爬山算法。
 - ③ 迭代过程通常由最大迭代次数或者两次迭代中心变化的程度进行限制; 虽然如此, 迭代过程最后都会收敛。

“核”是一个局部函数(例如高斯分布)。如果在足够的点处有足够的带权重和尺度的核，数据的分布便可以完全根据这些核来表示。与核密度估计不同，mean-shift 只估计数据分布的梯度(变化的方向)。若变化为 0 的地方则表示是这个分布的峰值(虽然可能是局部的)。当然在附近或其他尺度上可能还有峰值。

图 10-11 是 mean-shift 算法中的计算公式。

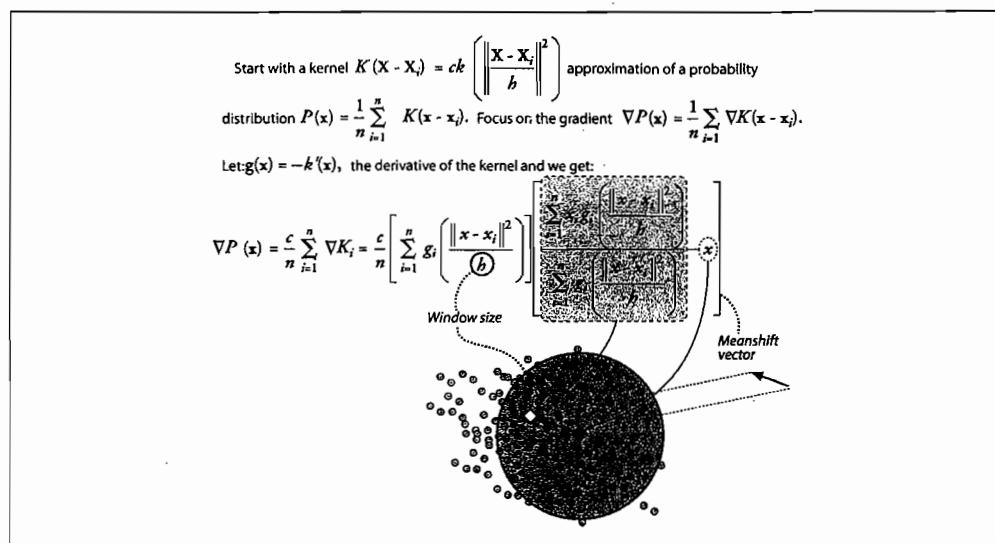


图 10-11：mean-shift 等式及其含义

这些计算公式可用一个矩形的核^①进行简化，将 mean-shift 矢量等式简化为计算图像像素分布的重心：

$$x_c = \frac{M_{10}}{M_{00}}, \quad y_c = \frac{M_{01}}{M_{00}}$$

这里，零阶矩的计算如下：

$$M_{00} = \sum_x \sum_y I(x, y)$$

一阶矩的计算为：

【338】

$$M_{10} = \sum_x \sum_y xI(x, y) \quad \text{和} \quad M_{01} = \sum_x \sum_y yI(x, y)$$

① 矩形核并不随着到中心的距离下降，而是一个突然变成零的突然转换。这个与高斯核的指数衰减不同，与 Epanechnikov 核的随着到中心的距离的开方衰减也不同。

mean-shift 矢量告诉我们如何将 mean-shift 窗口的中心重新移动到由计算得出的此窗口的重心的位置。很显然，窗口的移动造成了窗口内容的改变，于是我们又重复刚才重新定位窗口中心的步骤。窗口中心重定位的过程通常会收敛到 mean-shift 矢量为 0(也就是，窗口不能再移动)。收敛的位置在窗口中像素分布的局部最大值(峰值)处。由于“峰值”本身是一个对尺度变化敏感的量，所以窗口大小不同，峰值的位置也不一样。

图 10-12 是一个二维数据分布及初始化窗口(此例中，窗口为矩形)的例子。箭头表示迭代过程最终收敛到分布的局部峰值。正如前文所说的，我们可以看到由于处于 mean-shift 窗口外部点不影响 mean-shift 的收敛性，所以寻找峰值的这一过程在统计意义上是稳定的。

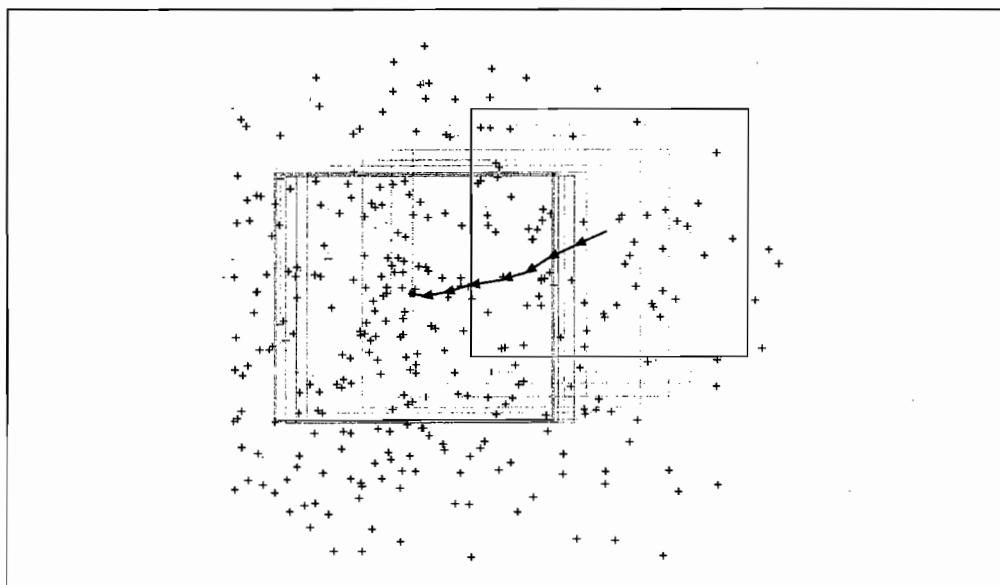


图 10-12：mean-shift 算法的过程。将初始窗口放置在数据点的二维数组上，连续重定位(窗口)中心到数据分布的局部峰值处，直到收敛

1998 年，人们认识到这种寻优的算法可以用于跟踪视频中的运动物体[Bradski98a; Bradski98b]，此后这个算法被大大地扩展[Comaniciu03]。OpenCV 中运用 mean-shift 算法的函数是在图像分析的层面上实现的。这也就是说，OpenCV 中实现的 mean-shift 把一幅代表将要被分析的密度分布的图像作为输入，而不是一组任意的点(可能是任意的维数)作为输入。

【339 ~ 340】

```
int cvMeanShift(
    const CvArr*      prob_image,
```

```
CvRect          window,  
CvTermCriteria criteria,  
CvConnectedComp* comp  
);
```

cvMeanShift()函数中，参数 prob_image 是单通道图像，代表可能位置的密度，其类型可以是 byte 或 float。参数 window 的位置设置在为指定初始位置，大小为核窗口的大小。终止条件 criteria 主要由 mean-shift 移动的最大迭代次数和可视为窗口位置收敛的最小移动距离组成，其定义在其他章节中有描述^①。连接部件 comp 在 comp->rect 中包含收敛后搜索窗口的位置，在 comp->area 中包含了窗口内部所有像素点的和。

函数 cvMeanShift() 使用的是矩形窗口的 mean-shift 算法，但这同样可以用于跟踪。在这种情况下，需要首先选择代表物体的特征的分布(例如，颜色+纹理)，然后在物体的特征分布上开始 mean-shift 窗口搜索，最后计算下一帧视频中所选择的特征的分布。从当前的窗口位置开始，mean-shift 算法寻找特征分布的新的峰值或者 mode，它们(被假定)设置在最初产生颜色和纹理的物体的中心。这样，mean-shift 窗口就可以逐帧跟踪物体的运动。

CamShift

另一个相关的算法是 CamShift 跟踪器。与 mean-shift 不同的是，CamShift 搜索窗口会自我调整尺寸。如果有一个易于分割的分布(例如保持紧密的人脸特征)，此算法可以根据人在走近或远离摄像机时脸的尺寸而自动调整窗口的尺寸。CamShift 算法的形式如下：

```
int cvCamShift(  
    const CvArr*      prob_image,  
    CvRect            window,  
    CvTermCriteria    criteria,  
    CvConnectedComp*  comp,  
    CvBox2D*          box = NULL  
);
```

前四个参数的含义与 cvMeanShift() 算法中的含义相同。如果参数 box 不为空，

① 强调一下 mean-shift 总是会收敛，但是如果一个分布在局部峰值附近非常“平坦”，收敛将会变得非常慢。

则它包含了新的尺寸的 box，其中也包含根据二阶矩计算出的物体的方向。在跟踪的应用中，会把由前一帧计算出的新尺寸的 box 作为下一帧的 window。

注意：很多人认为 mean-shift 和 camshift 利用颜色特征进行跟踪，但实际上不完全是这样。两种算法跟踪在 prob_image 中所代表的任意的分布，所以它们是轻量级的、稳定和有效的跟踪器。

运动模板

运动模板(motion template)是由 MIT 媒体实验室的 Bobick 和 Davis 提出的 [Bobick96, Davis97]，并且其中的一人又参与了它进一步的开发 [Davis99, Bradski00]。而这一步工作为运动模板在 OpenCV 中的实现提供了基础。运动模板是一种有效的跟踪普通运动的方法，尤其可应用在姿态识别中。运用运动模板需要知道物体的轮廓(或者轮廓的一部分，此处轮廓指 silhouette，即实心轮廓)，而轮廓的获取有不同的方法。

【341~342】

- (1) 最简单的获取物体轮廓的方法是用一个静止的摄像机，再使用帧间差(见第 9 章论述)得到物体的运动边缘，这种信息就足够让运动模板发挥效用。
- (2) 利用颜色信息。例如，如果已知背景的颜色，比如亮绿色，那么可以很简单地将不是亮绿色的任何物体看作前景。
- (3) 另一种方法(在第 9 章中讨论过)是学习一个背景模型，从此背景中可以将新的前景物体/人以轮廓的形式分割出来。
- (4) 使用主动轮廓技术。例如，创建一个近红外光的墙，将能感应近红外线的摄像机对着墙，那么任何介于墙与摄像机之间的物体都会以轮廓的形式出现。
- (5) 使用热感图像。任何温度高的物体(如人脸)都可以被认为是前景。
- (6) 最后，可以用第 9 章中谈到的分割技术(例如金字塔分割或 mean-shift 分割)来生成轮廓。

从现在开始，假设我们有一个如图 10-13(A)中白色矩形所代表的被较好的分割出的物体轮廓。白色表示所有像素都被设置为最新的系统时间的浮点数值。随着矩形的运动，新的轮廓将被捕获并且被(新的)当前轮廓覆盖；新的轮廓为图 10-13(B)和图 10-13(C)中的白色矩形所示。较早的运动在图 10-13 中表示为亮度渐暗的矩形。这些连续变暗的轮廓记录了早前运动的历史，所以被称为“运动历史图像(motion history image)”。

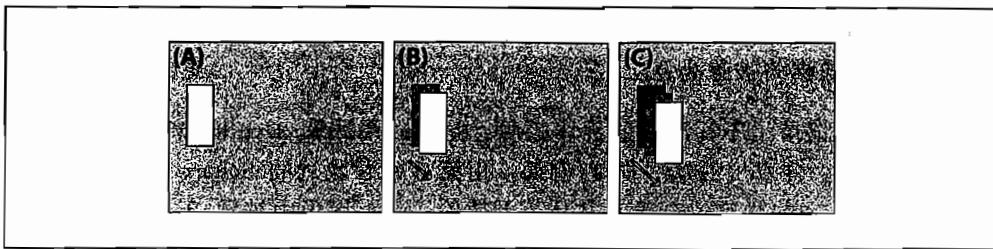


图 10-13：运动模板图：(A)当前时间分割出的物体(白色)；(B)在下一个时间点，物体运动并以(新的)当前时间标记，将前面的分割边界留在后面；(C)在下一个时间点，物体继续运动并将此前的分割标记为渐暗的矩形，这些包含运动的序列就产生运动历史图像

【342】

若轮廓的时间超过设定的比当前时间早的持续时间，则其被置 0，如图 10-14 所示。OpenCV 中完成运动模板构建的函数是 `cvUpdateMotionHistory()`：

```
void cvUpdateMotionHistory(
    const CvArr* silhouette,
    CvArr* mhi,
    double timestamp,
    double duration
);
```

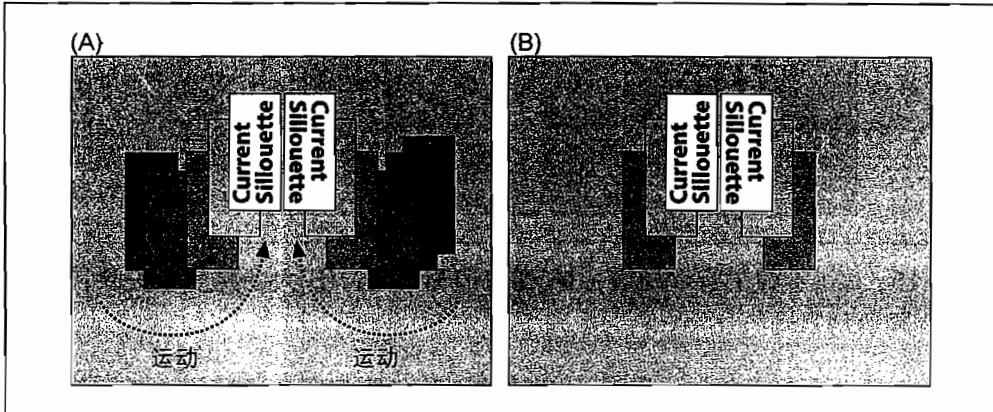


图 10-14：两个物体的运动模板轮廓(A)；超过设定持续时间的轮廓被置 0(B)

在 `cvUpdateMotionHistory()` 中，所有图像都是单通道图像。图像 `silhouette` 是一幅单字节图像，其中非 0 像素代表前景物体的最新的分割轮廓。图像 `mhi` 是一幅浮点值的图像，代表运动模板(也就是运动历史图像)。`timestamp` 是当前系统时间(典型地以毫秒为单位)，`duration` 如刚才谈到的表示运动历史像素的允许保

存在 mhi 中的最大持续时间。换句话说，mhi 中任何比 timestamp 减去 duration 的值早(少)的像素将被置为 0。

一旦运动模板记录了不同时间的物体轮廓，就可以用计算 mhi 图像的梯度来获取全局运动信息。计算出的这些梯度(例如，用第 6 章讨论过的 Scharr 或 Sobel 梯度函数来计算)，一些梯度值会很大并且是无效的。mhi 图像中被置 0 的旧的或没有运动的部分的梯度是无效的，因为它们会在轮廓的外部边缘区域人为地产生很大的梯度值，如图 10-15(A)所示。由于使用 cvUpdateMotionHistory() 将新的轮廓引进 mhi 的时间步长是已知的，所以梯度值应该为多大也就知道了(即为 dx 和 dy step derivatives)。那么我们可以用梯度幅值来消除特别大的梯度，如图 10-15(B)所示。最终，我们就得到全局运动的测量，如图 10-15(C)。图 10-15(A)和(B)是由 cvCalcMotionGradient() 函数得到的：

【343】

```
void cvCalcMotionGradient(
    const CvArr* mhi,
    CvArr* mask,
    CvArr* orientation,
    double delta1,
    double delta2,
    int aperture_size=3
);
```

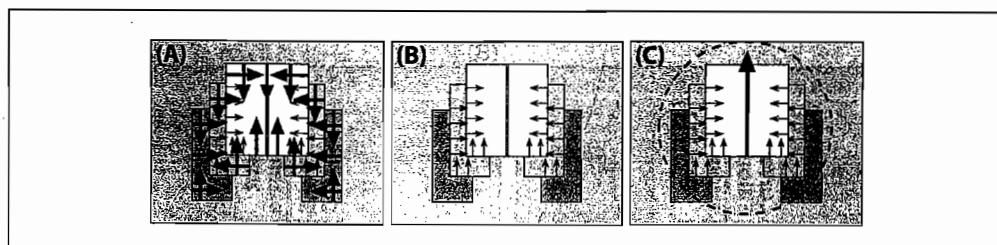


图 10-15：mhi 图像的运动梯度：(A)梯度值和方向；(B)大的梯度被去除；(C)得到全局运动方向

函数 cvCalcMotionGradient 中，所有的图像矩阵都是单通道的。函数的输入 mhi 是一幅浮点数值的运动历史图像，delta1 和 delta2 则分别是允许的最小和最大梯度值。期望的梯度值是连续调用 cvUpdateMotionHistory() 的每个轮廓间当前时间标记的平均数；将 delta1 设置为小于此平均值的一半，delta2 设置为大于此平均值的一半比较合适。变量 aperture_size 设置梯度算子的宽和高，其值可以为 -1(3×3 CV_SCHARR 梯度滤波器)、3(默认的 3×3 Sobel 滤波器)、5(5×5 Sobel 滤波器)或者 7(7×7 滤波器)。函数的输出 mask 是一幅 8 位的单通道图像，其中的

非 0 值代表此点处的梯度有效，orientation 是一幅浮点值的图像，给出每一点梯度方向的角度。

函数 cvCalcGlobalOrientation() 计算有效梯度方向矢量和来获得全局运动方向。

```
double cvCalcGlobalOrientation(
    const CvArr* orientation,
    const CvArr* mask,
    const CvArr* mhi,
    double timestamp,
    double duration
);
```

函数的输入为由 cvCalcMotionGradient() 计算得到的 orientation 和 mask 图像以及 timestamp、duration 和 cvUpdateMotionHistory() 得出的 mhi；函数的输出为如图 10-15(c) 所示的矢量和的全局方向。timestamp 和 duration 给函数设定了 mhi 和 orientation 图像中需要考虑的运动量。用每个 mhi 轮廓的重心可以计算得到全局运动，但是对已计算出的运动矢量求和会快很多。

我们还可以将运动模板 mhi 图像分割为独立的区域，并计算区域里的局部运动，如图 10-16 所示。图中，搜索 mhi 图像寻找当前的轮廓区域。当找到被标记为最新当前时间的区域时，继续搜索区域的边界以寻找紧邻边界外围的最近近的运动(最近的轮廓)。若找到这样的运动，就会用逐级洪水泛滥法将从感兴趣物体的当前位置分割出来，计算分割区域局部运动的梯度方向，再移除此区域。重复此过程直到所有的区域都被找到(如图 10-16 所示)。

【344~345】

分割和计算局部运动的函数为 cvSegmentMotion()：

```
CvSeq* cvSegmentMotion(
    const CvArr* mhi,
    CvArr* seg_mask,
    CvMemStorage* storage,
    double timestamp,
    double seg_thresh
);
```

函数中，mhi 是单通道、浮点值的输入图像。我们还传入 storage，它是一个由 cvCreateMemStorage() 分配的 CvMemoryStorage 结构。另一个输入为 timestamp，它的值是要从其中分割出局部运动的 mhi 图像中最新轮廓的值。最后，还必须传入一个参数 seg_thresh，它是认为可以接受为相关联运动的最大

从当前时间到早先的运动。之所以要输入这个参数，是由于最近的和早前的运动的轮廓可能存在重叠，但你并不想将它们连接在一起。

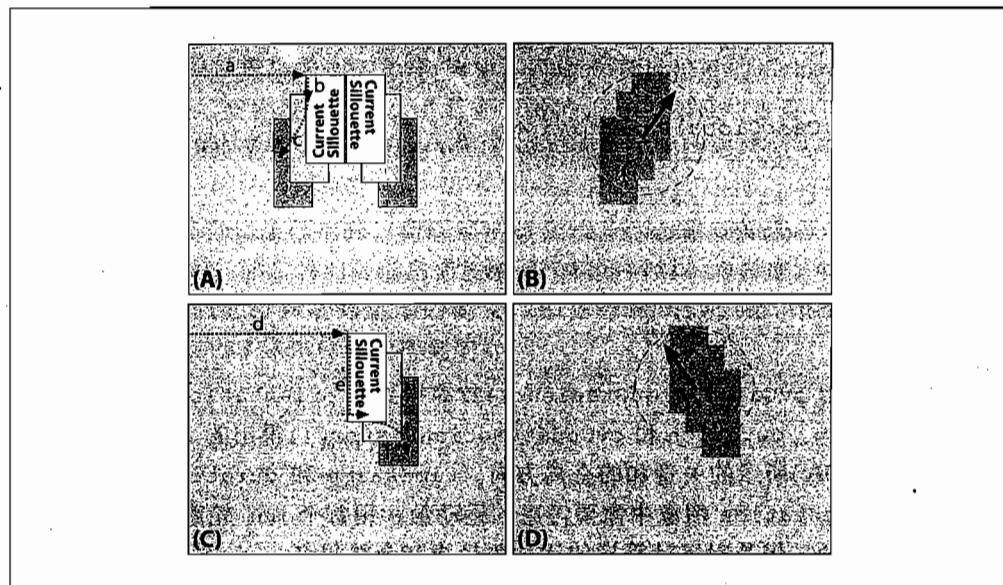


图 10-16：分割 mhi 图像中的局部运动区域：(A)搜索 mhi 图像寻找当前轮廓 a。当搜索到此轮廓，沿其边界搜索其他较近的轮廓(b)；若找到，用逐级洪水泛滥法 c 分割出局部运动；(B)计算分割出的局部运动区域的梯度，并用此梯度计算局部运动；(C)移除此区域并搜寻下一个当前轮廓的区域 d，沿着其搜索(e)，再用逐级洪水泛滥法填充 f；(D)计算新分割区域内的运动，重复(A)(C)过程直到没有剩余的当前轮廓

【345】

通常最好将 `seg_thresh` 的值设置为轮廓时间平均差别的 1.5 倍。函数返回 `CvConnectedComp` 结构的 `CvSeq` 序列，其中每一个代表搜索到的一个描述局部运动区域的分割的运动；函数还返回一个单通道的浮点值图像，`seg_mask`，其中每个被分割出的运动的区域被标记为不同的非 0 值(`seg_mask` 中值为 0 的像素表示无运动)。用从 `CvConnectedComp` 或者 `seg_mask` 中特定值中选择的掩码区域，调用 `cvCalcGlobalOrientation()`一次来计算一个局部运动，如下所示：

```
cvCmpS(  
    seg_mask,  
    // [value_wanted_in_seg_mask],  
    // [your_destination_mask],  
    CV_CMP_EQ  
)
```

讨论到这里，现在应该能够理解 OpenCV 的 .../opencv/samples/c/ 目录中的 *motempl.c* 例子。我们现在从 *motempl.c* 中 *update_mhi()* 函数抽出一些关键点来解释。*update_mhi()* 函数使用有阈值的帧间差来获取模板，然后得到的轮廓传递给 *cvUpdateMotionHistory()*：

```
...
cvAbsDiff( buf[idx1], buf[idx2], silh );
cvThreshold( silh, silh, diff_threshold, 1, CV_THRESH_BINARY );
cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION );
...

```

计算得到的 *mhi* 图像的梯度，再由 *cvCalcMotionGradient()* 获得有效梯度的掩码。分配 *CvMemStorage*(或者，如果已经存在则释放)，分割出的局部运动存储在 *CvConnectedComp* 结构中，而 *CvConnectedComp* 在包含 *CvSeq* 结构的 *seq* 中：

```
...
cvCalcMotionGradient(
    mhi,
    mask,
    orient,
    MAX_TIME_DELTA,
    MIN_TIME_DELTA,
    3
);
if( !storage )
    storage = cvCreateMemStorage(0);
else
    cvClearMemStorage(storage);
seq = cvSegmentMotion(
    mhi,
    segmask,
    storage,
    timestamp,
    MAX_TIME_DELTA
);

```

【346~347】

我们用一个 *for* 循环重复从 *sep->total* 个 *CvConnectedComp* 结构中提取每个运动的边界矩形。循环从 -1 开始，这样指定为的是要寻找整幅图像的全局运动。对于局部运动块，忽略面积小的分割后，用 *cvCalcGlobalOrientation()* 计算方向。函数将运动的计算限制在包含局部运动的兴趣区域(ROI)，而不是提取掩码；然

后，计算在局部 ROI 中有效运动的实际位置。任何太小的这样的运动区域将被丢弃。最后，函数将运动绘制出来。图 10-17 表示了对一个挥动手臂的人使用此函数的输出结果，即为两行每行四帧连续原始图像上方的图像。(完整的代码可以参考 .../opencv/samples/c/motempl.c。)序列中，“Y”姿态由第 8 章讨论的形状算子(Hu 矩)识别得出。samples 代码中还未包含形状识别。

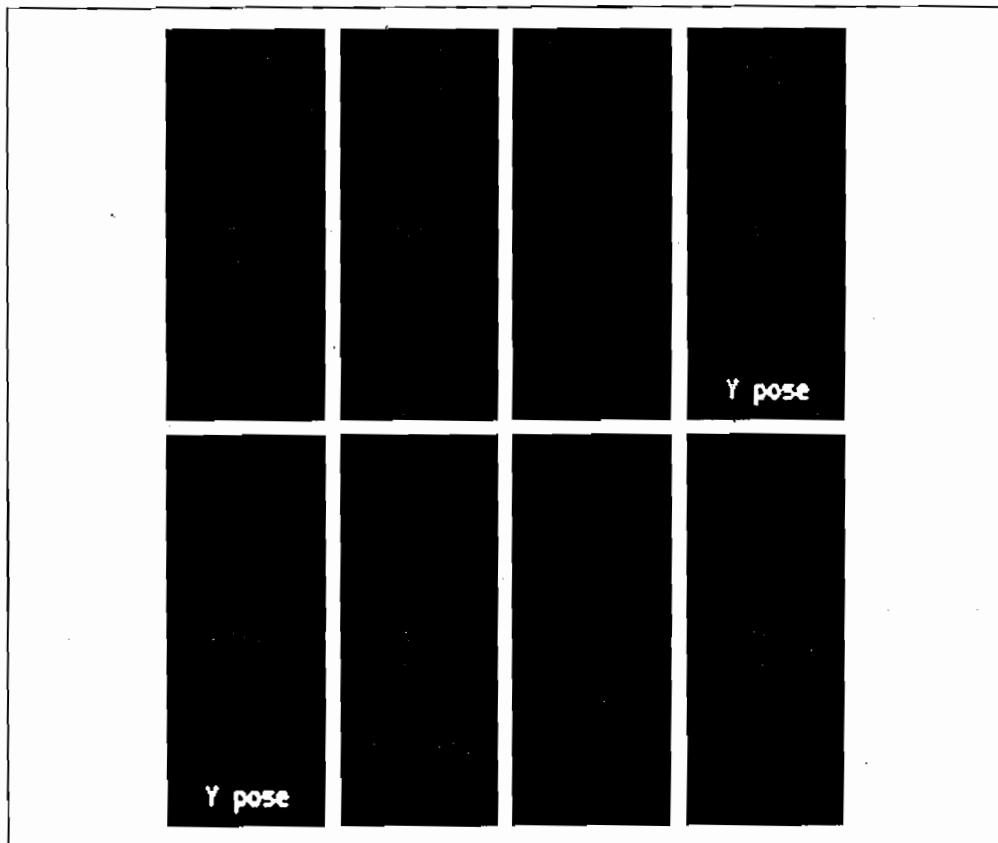


图 10-17：运动模板函数运行结果。从左至右、从上到下，一个人在运动，输出结果中大的八边形表示全局运动，小的八边形表示局部运动；“Y”姿态可由形状算子(Hu 矩)识别

```
for( i = -1; i < seq->total; i++ ) {  
    if( i < 0 ) { // case of the whole image  
        // ...[does the whole image]...  
    } else { // i-th motion component  
        comp_rect = ((CvConnectedComp*)cvGetSeqElem( seq, i ))->rect;  
        // [reject very small components]...  
    }  
}
```

```
}

...[set component ROI regions]...
angle = cvCalcGlobalOrientation( orient, mask, mhi,
timestamp, MHI_DURATION);
...[find regions of valid motion]...
...[reset ROI regions]...
...[skip small valid motion regions]...
...[draw the motions]...
}
```

【347】

预估器

假设我们跟踪一个穿过摄像机视场的人，每一帧都要确定此人的位置。正如我们知道的，有多种方法可以达到这个目的，但是我们发现每种情况下得到的只是此人在每帧中位置的估计。这个估计，不太可能相当的准确。造成不准确的原因有很多。可能是传感器的不精确，早期处理阶段的近似，遮挡或者阴影，又或者是由于腿和手臂在行走中的摆动造成的明显的形状变化。不管源头是什么，我们期望这些测量对可能从理想传感器获得的“实际”值会发生变化，而且可能是任意的。整体考虑这些影响因素，可以将这些不准确简单地看作向跟踪过程添加噪声。【348~349】

我们希望可以最大限度地使用测量结果来估计此人的运动。所以，多个测量的累积可以让我们检测出不受噪声影响的部分观测轨迹。一个关键的附加要素即此人运动的模型。例如，我们可能用下面的方式建立此人的运动模型：“一个人从图像的一边进入并以恒定的速度穿过图像。”有了这个模型，我们不仅可以知道此人在什么位置，同时还可以知道我们的观察支持模型的什么参数。

这个任务分成两个阶段(如图 10-18 所示)。在第一阶段，即预测阶段，用从过去得到的信息进一步修正模型以取得人(或物体)的下一个将会出现的位置。在第二阶段，即校正阶段，我们获得一个测量，然后与基于前一次测量的预期值(即模型)进行调整。

完成两个阶段估计任务的方法被归入预估器(estimator)这个标题下，其中 Kalman 滤波器[Kalman60]是最广泛使用的。除了 Kalman 滤波器，另一重要的方法是 condensation 算法，它是更为广泛的一类方法粒子滤波在计算机视觉领域的实现。Kalman 滤波器和 condensation 算法主要的区别在于状态概率密度是如何描述的。后面将探究这个区别的含义。

【349~350】

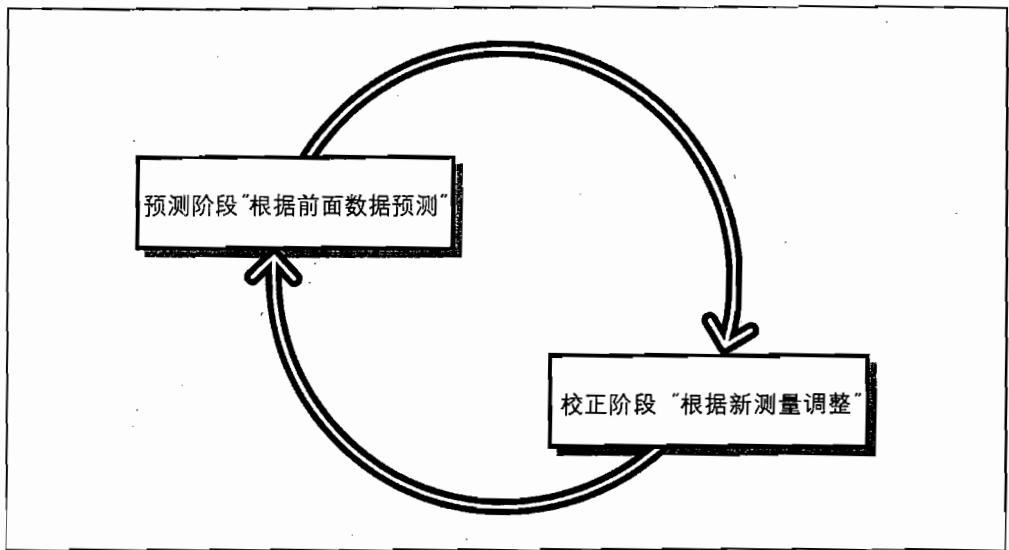


图 10-18：两阶段的预估器循环：基于前面数据的预测和根据最新测量的调整

Kalman 滤波器

Kalman 滤波器(即卡尔曼滤波器)最初于 1960 年提出，之后发展成为信号处理领域中不同方面的重要方法。Kalman 滤波器的基本思想是，若有一组强而合理的^①假设，给出系统的历史测量值，则可以建立最大化这些早前测量值的后验概率的系统状态模型^②。Welsh 和 Bishop[Welsh95]做了比较好的介绍。另外，无需存储很长的早前测量历史，我们也可以最大化后验概率，即重复更新系统状态模型，并只为下一次更新保存模型。这样就大大地简化了这个方法的计算机实现。

在详细介绍这些在实际应用中的含义之前，我们花一点时间来看看提到的假设是什么。理论上，Kalman 滤波器需要三个重要的假设：(1)被建模的系统是线性的；(2)影响测量的噪声属于白噪声；(3)噪声本质上是高斯分布的。第一条假设的意思是 k 时刻的系统状态可以用某个矩阵与 $k-1$ 时刻的系统状态的乘积表示。余下两条假

- ① “合理的”在这里的意思是“限制非常宽松使得这个方法对真实世界中出现的相当多的实际问题都有用”。“合理的”只是仅次于“完美的”。
- ② 修饰词“后验的”是学术领域用于描述“事后解释”的一个术语。当我们说这样一个分布“最大化后验概率”时，意思是虽然本质上是对“实际发生过的”一种可能的解释，但结合已经观测到的数据，这个分布实际上是最可能的一个，也就是以回顾的方式分析。

设，即假设噪声是高斯分布的白噪声，其含义为噪声与时间不相关，且只用均值和协方差(也就是噪声完全由一阶矩和二阶矩描述)就可以准确地为幅值建模。这些假设看起来似乎非常苛刻，但它们实际上却应用在非常广泛的普通环境中^①。

“最大化前期测量值的后验概率”是什么意思？意思就是在获得测量值——考虑早前的模型和新测量值的不确定性——之后新建立的模型是具有最高正确概率的模型。给定三条假设，Kalman 滤波器是将从不同来源获得的数据或从同一来源不同时间获得的数据结合的最好的方法。从我们知道的信息开始，获取新的信息，然后根据对旧信息和新信息的确定程度，用新旧信息带权重的结合对我们知道的信息进行更新。

【350~351】

我们需要用一点数学和一维运动的情况来说明上文所讲的概念，你可以跳过下一节。线性系统和高斯分布是非常易懂的，如果你看都不看一眼，Kalman 博士可能会不开心。

Kalman 滤波器相关的一些数学知识

Kalman 滤波器的核心是什么？是信息融合。假设你想知道某一点在一条直线的什么位置(这就是一维的情况)^②，由于噪声的影响，将会得到两个不可靠的(从高斯概率的角度)物体位置的变量： x_1 和 x_2 。因为这些测量值中存在高斯不确定性，所以有均值 \bar{x}_1 和 \bar{x}_2 以及标准差 σ_1 和 σ_2 。标准差其实是反映了我们对于测量值好坏程度的不确定性。以位置为变量的概率分布为下面的高斯分布：

$$p_i(x) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x - \bar{x}_i)^2}{2\sigma_i^2}\right) (i=1,2)$$

若有这样两个具有各自高斯概率分布的测量，我们期望在同时给出两个测量的条件下某一 x 值的概率密度与 $p(x) = p_1(x)p_2(x)$ 成比例。这个乘积的结果也是一个高斯分布，可以按照下面的方式来计算这个新的分布的均值和标准差。由于有

-
- ① 再多说一句。在这里我们实际上又用了另一个假设，即初始分布在本质上也是高斯分布的。通常应用中的初始状态是准确地知道的，或者至少可以这样处理，于是这就满足了我们的要求。如果初始状态为(举例说)在卧室或者在浴室的概率各占一半，则我们会很不幸运，则需要比单一的 Kalman 滤波器更有好的方法。
 - ② 更多详细的解释以及一个曲线轨迹的例子，可以参考 J. D. Schutter, J. De Geeter, T. Lefebvre 和 H. Bruyninckx 所著的 “Kalman Filters: A Tutorial” (<http://citeseer.ist.psu.edu/443226.html>)。

$$p_{12}(x) \propto \exp\left(-\frac{(x - \bar{x}_1)^2}{2\sigma_1^2}\right) \exp\left(-\frac{(x - \bar{x}_2)^2}{2\sigma_2^2}\right) = \exp\left(-\frac{(x - \bar{x}_1)^2}{2\sigma_1^2} - \frac{(x - \bar{x}_2)^2}{2\sigma_2^2}\right)$$

并且高斯分布在平均值处最大，我们可以简单地用计算关于 x 的 $p(x)$ 的导数来获得平均值。由于函数在其导数为 0 处值最大，所以有

$$\left. \frac{dp_{12}}{dx} \right|_{\bar{x}_{12}} = -\left[\frac{\bar{x}_{12} - \bar{x}_1}{\sigma_1^2} + \frac{\bar{x}_{12} - \bar{x}_2}{\sigma_2^2} \right] \cdot p_{12}(\bar{x}_{12}) = 0$$

由于概率分布函数 $p(x)$ 不会为 0，所以括号中的项必须为 0。解这个等式后，可得到一个非常重要的关系式：

$$\bar{x}_{12} = \left(\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \right) \bar{x}_1 + \left(\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \right) \bar{x}_2$$

【351】

于是，新分布的均值即为两个测量到的均值的加权组合，而权重则由两个测量相关的不确定性决定。可以看到，例如，如果第二个测量的不确定性 σ_2 特别大，则新的均值与更确定的前期测量的均值 \bar{x}_1 本质上是相同的。

知道了新的均值 \bar{x}_{12} ，将其代入 $p_{12}(x)$ 的表达式中进行整理^①，则不确定性 σ_{12}^2 即为：

$$\sigma_{12}^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

计算到这里，你可能想知道这个说明了什么。实际上，当我们用新的均值和不确定性进行新的测量时，可以将新的测量与已有的均值和不确定性结合来获得由更新的均值和不确定性确定的新状态。(这些过程现在有数值表达式，我们马上会讲到)。

两个高斯测量相结合等价于一个高斯测量(其均值和不确定性是可计算的)这个性质对我们来说是最重要的一个。这个性质的意思是，当有 M 个高斯测量时，可以先合并前两个，再将前两个合并的结果与第三个进行合并，然后再将合并的结果与第四个合并，以此类推。计算机视觉中的跟踪就是这样使用的；由一个测量推出下一个再推出其下面的一个。

^① 整理的过程有些繁杂。如果你想验证整个过程，这样做会容易很多：(1)根据 \bar{x}_{12} 和 σ_{12}^2 由高斯分布 $p_{12}(x)$ 的等式开始，(2)将等式中的与 \bar{x}_{12} 相关的用 \bar{x}_1 和 \bar{x}_2 代替，与 σ_{12} 相关的用 σ_1 和 σ_2 代替，最后(3)验证替换后的结果是否能分离成最开始的高斯的乘积。

由于测量值 (x_i, σ_i) 是时间离散的，可以如下面所述的方法来计算 $(\hat{x}_i, \hat{\sigma}_i^2)$ 估计值的当前状态。在时间点 1，只有第一个测量 $\hat{x}_1 = x_1$ 和它的不确定性 $\hat{\sigma}_1^2 = \sigma_1^2$ 。在最优估计等式中将它们替换，则得到一个迭代等式：

$$\hat{x}_2 = \frac{\sigma_2^2}{\hat{\sigma}_1^2 + \sigma_2^2} x_1 + \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2} x_2$$

重新改写一下等式，则可得到如下有用的形式：

$$\hat{x}_2 = \hat{x}_1 + \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2} (x_2 - \hat{x}_1)$$

在我们思考这个公式如何使用之前，先来计算一下 $\hat{\sigma}_2^2$ 的另一种表达形式。首先，令 $\hat{\sigma}_1^2 = \sigma_1^2$ ，则有：

$$\hat{\sigma}_2^2 = \frac{\sigma_2^2 \hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}$$

像处理 \hat{x}_2 那样重新整理一下等式，则得到新的测量的估计协方差的迭代公式：

$$\hat{\sigma}_2^2 = \left(1 - \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}\right) \hat{\sigma}_1^2$$

用这些等式当前的形式，我们可以很清晰地分辨出“旧”信息(在新的测量结果得到之前已知的)和“新”信息(最新的测量结果)。在第二个时间点 2，新的信息 $(x_2 - \hat{x}_1)$ 叫“变化(innovation)”。现在我们也可以得到最优迭代更新比例：

$$K = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}$$

这个比例叫更新率(update gain)。用 K 的这个定义，我们可以得到下面的比较方便的递归形式：

$$\hat{x}_2 = \hat{x}_1 + K(x_2 - \hat{x}_1)$$

$$\hat{\sigma}_2^2 = (1 - K)\hat{\sigma}_1^2$$

在 Kalman 滤波器文献中，如果讨论的是普通的测量，则第二个时间点“2”通常就表示为 k ，而第一个时间点则表示为 $k-1$ 。

动态系统

在简单的一维情况的例子中，我们认为目标在某一点 x ，并且在这一点进行连续的

测量。在这种情况下，我们并没有特别地去考虑到在测量间物体实际上可能在运动的情况。对于这种情况，我们使用预测。在预测阶段，在新的测量添加进来之前，我们利用已知去计算系统的期望值。

实际上，预测阶段发生在新的测量完成之后和新测量的值被添加进系统状态的估计中之前。举例来说，我们在时间 t 测量汽车的位置，再在时间 $t+dt$ 测量一次。若汽车的速度为 v ，则我们不直接合并第二次测量的结果。首先，用在时间 t 得到的模型向前推，得到系统在时间 $t+dt$ ，即新的测量值被合并之前的时刻的模型。这样，在时间 $t+dt$ 获得的新的测量值与由旧模型向前映射到时间 $t+dt$ 的模型融合，而不是直接与这个旧的模型融合。这就是图 10-18 中描绘的循环所表示的意思。在 Kalman 滤波器应用中，我们将要考虑三种运动。

【353~354】

第一种运动是动态运动(dynamical motion)。这种运动是我们期望的前次测量时系统状态的直接结果。如果我们在位置 x 、时间 t 时测量速度为 v 的系统，则在时间 $t+dt$ 我们可以预期此系统在位置 $x+v\times dt$ ，速度可能仍为此速度。

第二种形式的运动叫控制运动(control motion)。这种运动是我们期望的，由于某种已知的外部因素以某种原因施加于系统。正如运动名称所暗示的，控制运动最普遍的一个例子是，当我们施加了控制的系统估计其状态时，我们知道我们的控制会使系统产生什么样的运动。这种情况对机器人系统来说尤其是这样。在机器人系统中，这种控制就是系统告诉机器人(例如)加速或者向前。很明显，在这种情况下，如果机器人在时间 t 时在位置 x 并且运动速度为 v ，那么由于我们命令机器人加速，在时间 $t+dt$ 我们预期机器人运动的距离不是 $x+v\times dt$ (这个值是系统没有外在控制时的预期值)，而是比这更远。

最后一类运动是随机运动(random motion)。即便是在简单的一维情况的例子中，如果观测的目标有因任一原因而产生运动的可能性，那么就需要在预测阶段包含这种随机运动。这种随机运动的影响是简单地增加状态估计随时间的协方差。随机运动包含未知的或不在我们控制中的任何运动。但是，在 Kalman 滤波器中，随机运动被假设成为高斯模型(也就是一种随机)或者至少是可以用高斯模型有效地来建模。

因此，我们在融合一个新的测量值前先进行“更新”以便将动态变化包含进仿真模型中。在更新阶段首先将我们所已知的目标的运动信息应用到其先前的状态上，然后再应用任何由我们施加的控制或从某个外部代理施加的已知的控制得到的任何附加信息，最后将在上次观测完成后可能改变系统状态的符合定义的随机事件融合进来。应用这些因素之后，我们便可以融合下一个新的测量值。

实际上，在系统“状态”比我们的仿真模型复杂得多的时候，动态运动尤其重要。通常，当目标在运动时，系统的“状态”有多个组成成分，例如位置和速度。当然

在这种情况下，系统状态随我们认为其具有的速度而发展。在下一节中将讨论如何处理具有多个状态成分的系统。我们还将会用一些稍微复杂的符号来处理这些问题的新方面。

【354】

Kalman 方程

现在我们来推广简单模型中的运动等式。这个更普遍的讨论中的模型是目标状态的线性函数 F 。比如说，这样的一个模型可能会考虑到先前运动一阶和二阶导数的结合。我们也将看到如何在模型中处理控制输入 u_k 。最后，我们将得到一个更符合实际的观测模型 z ，在这个模型中只需要测量几个模型的状态变量，并且测量值与状态变量间没有直接联系^①。

首先，我们看一下前一节中提到的更新率 K 如何影响状态估计。若新的测量值的不确定性非常大，则此新的测量值对状态估计不起作用，等式还原为我们已知的在时间 $t-1$ 的结果。相反地，如果开始时原始测量的协方差非常大，那么我们进行新的更准确的测量，然后“相信”这个值。若两个测量值具有相同的确定性（协方差），则新的期望值必将在它们之间。以上的论述与我们的合理的期望是相符的。

图 10-19 表示不确定性是如何根据新的观测随时间变化。

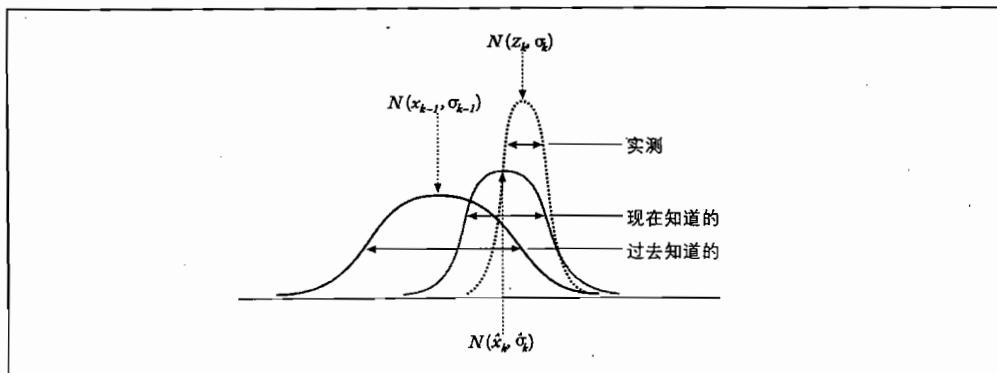


图 10-19：将先验知识 $N(x_{k-1}, \sigma_{k-1})$ 与观测 $N(z_k, \sigma_k)$ 结合；结果为新的估计 $N(\hat{x}_k, \hat{\sigma}_k)$

更新，对不确定性敏感，可以推广到其他状态变量。最简单的例子就是在视频跟踪的应用中，视频中的物体可以具有两个或三个方向的运动。一般来说，状态可能包

① 观察从 x_k 到 z_k 在符号上的变化。后者是文献中的标准，意思是 z_k 不仅是（有时是不是）对位置 x_k 的测量，可能是对模型多个参数的更一般的测量。

含附加的元素，例如被跟踪的物体的速度。在这些任意的一般情况下，为了说明我们所要讨论的问题，需要再引入一些符号。我们将时间 k 的状态描述推广为时间 $k-1$ 的状态的函数。

$$x_k = Fx_{k-1} + Bu_k + w_k$$

【355~356】

这里， x_k 现在是一个状态元素的 n 维向量， F 是一个与 x_k 相乘的 $n \times n$ 矩阵，其有时也被称作传递矩阵。向量 u_k 是新添加的。它的作用是允许外部控制施加于系统，由表示输入控制的 c 维向量组成。 B 是一个联系输入控制和状态改变的 $n \times c$ 矩阵。变量 w_k 是一个关联直接影响系统状态的随机事件或外力的随机变量(通常称为过程噪声)。假设 w_k 的元素具有高斯分布 $N(0, Q_k)$ ， $n \times n$ 协方差矩阵 Q_k (Q 可以随时间变化，但通常不这样做)。

聪明的读者，或者已经对 Kalman 滤波器有一定了解的读者会注意到漏掉的另一个重要假设，也就是在控制 u_k 和状态变化间存在线性关系(通过矩阵乘法)。在实际应用中，这条假设常常首先被违反。

一般来说，测量值 z_k 有可能是但也有可能不是状态变量的直接测量 x_k 。(例如，如果想要知道一辆汽车行驶得多快，可以用雷达枪测量它的速度，也可以测量排气管传出的声音，在前一种情况中， z_k 为 x_k 加上测量噪声，而后一种情况，这种关系并不这么直接)。我们将这种情况总结为测量下面等式给出的测量值 z_k 的 m 维向量：

$$z_k = H_k x_k + v_k$$

这里， H_k 是 $m \times n$ 矩阵， v_k 是测量误差，也被假设为具有高斯分布 $N(0, R_k)$ 和 $m \times m$ 协方差矩阵 R_k 。^①

在我们还没有彻底被弄糊涂之前，我们来看一个具体的测量在停车场行驶的汽车的实际例子。我们可以将汽车的状态用两个位置变量 x 和 y ，以及两个速度变量 v_x 和 v_y 表示。这四个变量组成状态向量 x_k 的元素。这就是说 F 的正确形式为：

$$x_k = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

【356】

① 这些项中的 k 表示它们可以随时间变化，但是并不需要这样。在实际的应用中， H 和 R 通常不随时间变化。

但是，当使用摄像机去测量汽车的状态时，可能只能测量到位置变量：

$$z_k = \begin{bmatrix} z_x \\ z_y \end{bmatrix}_k$$

这就暗示了 H 结构类似于下面这样：

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

在这个例子中，我们可能并不真正认为汽车的速度是不变的，所以要设置一个值 Q_k 来反映这个问题。我们对视频流用图像分析方法来测量汽车的位置，再根据对测量精确程度的估计来选择 R_k 。

现在剩下所要做的就是将这些表达式嵌入推广形式的更新等式中。基本思想是相同的，但是，首先要计算一下状态的先验估计 \bar{x}_k 。在文献中，比较常见的(虽然不是普遍的)是用符号作为上标来表示“在新的测量之前”；在此我们也采用这个惯例。这个先验估计如下：

$$\bar{x}_k = F\bar{x}_{k-1} + Bu_{k-1} + w_k$$

用 P_k^- 表示误差协方差，此协方差在时间 k 的先验估计由其在时间 $k-1$ 的值获得：

$$P_k^- = FP_{k-1}F^T + Q_{k-1}$$

这个等式就构成了预估器预测部分的基础，它告诉我们根据我们已经看到的“我们可以期望什么”。由此，我们可以给出(并没有引出)所谓的 Kalman 更新率或者混合比例，它告诉我们如何给定新信息相对已经知道的信息的权重：

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$$

虽然这个等式看起来有些勉强，但实际上它也不是很恐怖。若思考一下不同的简单的情形，就能更容易地理解这个等式。在直接测量一个位置变量的一维的例子中， H_k 仅仅是只有 1 的 1×1 矩阵！所以，如果测量误差是 σ_{k+1}^2 ，那么 R_k 也是只含有 1 值的 1×1 矩阵。类似的， P_k 也正是这个协方差 σ_k^2 。于是，那个庞大的等式就简化为如下：

$$K = \frac{\sigma_k^2}{\sigma_k^2 + \sigma_{k+1}^2}$$

【357】

注意，这正是我们期望的。在前一节中首次提到的更新率让我们在获得新的测量值时，可以计算 x_k 和 P_k 最优的更新值：

$$x_k^- = x_k^- + K_k(z_k^- - H_k x_k^-)$$

$$P_k^- = (I - K_k H_k) P_k^-$$

这些方程又一次让人乍一看很头疼，但是在所讨论的简单的一维情况中，它们也不是这么糟的。最优权重和更新率以与一维情形相同的方法获得，除了此次我们在解之前设置对 x 的偏导数为 0 来最小化位置状态 x 的不确定性。可以先设置 $F=I$ (I 是单位矩阵)， $B=0$ 和 $Q=0$ ，来表示更简单的一维情况。在更一般的等式中作如下的替换，同时也揭示出与一维滤波器导数的相似之处： $x_k \leftarrow \hat{x}_2$ ， $x_k^- \leftarrow \hat{x}_1$ ， $K_k \leftarrow K$ ， $z_k \leftarrow x_2$ ， $H_k \leftarrow 1$ ， $P_k \leftarrow \hat{\sigma}_2^2$ ， $I \leftarrow 1$ ， $P_k^- \leftarrow \hat{\sigma}_1^2$ 和 $R_k \leftarrow \hat{\sigma}_2^2$ 。

OpenCV 和 Kalman 滤波器

根据我们知道的这一切，你可能会觉得不需要 OpenCV 为我们做什么事情或者迫切地需要 OpenCV 为我们完成一切工作。幸运的是，OpenCV 可以满足两种需求。OpenCV 提供了四个与应用 Kalman 滤波器直接相关的函数。

```
cvCreateKalman(
    int      nDynamParams,
    int      nMeasureParams,
    int      nControlParams
);
cvReleaseKalman(
    CvKalman** kalman
);
```

第一个函数产生和返回一个 CvKalman 数据结构的指针，第二个函数删除这个结构。

```
typedef struct CvKalman {
    int MP;                      // measurement vector dimensions
    int DP;                      // state vector dimensions
    int CP;                      // control vector dimensions
    CvMat* state_pre;            // predicted state:
                                // x_k = F x_k-1 + B u_k
    CvMat* state_post;           // corrected state:
                                // x_k = x_k' + K_k (z_k' - H x_k')
    CvMat* transition_matrix;    // state transition matrix
                                // F
```

```

CvMat* control_matrix; // control matrix
    // B
    // (not used if there is no control)
CvMat* measurement_matrix;
    // measurement matrix
    // H
CvMat* process_noise_cov;
    // process noise covariance
    // Q
CvMat* measurement_noise_cov;
    // measurement noise covariance
    // R
CvMat* error_cov_pre; // prior error covariance:
    // (P_k' = F P_k-1 Ft) + Q
CvMat* gain; // Kalman gain matrix:
    // K_k = P_k' H^T (H P_k' H^T + R)^-1
CvMat* error_cov_post; // posteriori error covariance
    // P_k = (I - K_k H) P_k'
CvMat* temp1;           // temporary matrices
CvMat* temp2;
CvMat* temp3;
CvMat* temp4;
CvMat* temp5;
} CvKalman;

```

【358~359】

下面两个函数是 Kalman 滤波器的实现。一旦 CvKalman 结构被赋值，则调用 cvKalmanPredict() 计算下一个时间点的预期值，然后调用 cvKalmanCorrect() 校正新的测量值。这些函数每个运行完成之后，我们就可以获得被跟踪系统的状态。cvKalmanCorrect() 的结果存储在 state_post 中，cvKalmanPredict() 的结果存储在 state_pre 中。

```

cvKalmanPredict(
    CvKalman*    kalman,
    const        CvMat*  control = NULL
);
cvKalmanCorrect(
    CvKalman*    kalman,
    CvMat*       measured
);

```

Kalman 滤波器示例代码

很明显现在是给出一个例子的时间了。我们举一个相对简单的例子并实现。假设有一个做圆周运动的点，就像一辆在赛道上行驶的汽车。汽车沿着赛道以几乎恒定的速度行驶，但是也存在一些波动(也就是过程噪声)。我们使用某种方法测量汽车的位置，例如用视觉的方法对其进行跟踪。这也会产生一些(不相关的并且可能不同的)噪声(也就是测量噪声)。

这样，我们的模型就比较简单：汽车在任意时刻都具有位置和角速度。这两个元素合起来形成一个二维的状态向量 x_k 。但是，我们只能测量汽车的位置，所以便得到一个一维的“向量” z_k 。

编写一段程序(例 10-2)，其输出显示了汽车的圆形运动(红色)，我们的测量值(黄色)和由 Kalman 滤波器预测的位置(白色)。

我们在调用的开始包含几个库的头文件。同时，定义一个将被证明是有用的宏，用于将汽车的位置从角度转换为笛卡儿坐标以便在屏幕上绘图。 【359】

例 10-2：Kalman 滤波器示例代码

```
// Use Kalman Filter to model particle in circular trajectory.  
//  
#include "cv.h"  
#include "highgui.h"  
#include "cvx_defs.h"  
  
#define phi2xy(mat) /  
    cvPoint( cvRound(img->width/2 + img->width/3*cos(mat->  
        data.fl[0])), /  
    cvRound( img->height/2 - img->width/3*sin(mat->data.fl[0])) )  
  
int main(int argc, char** argv) {  
  
    // Initialize, create Kalman Filter object, window, random number  
    // generator etc.  
    //  
    cvNamedWindow( "Kalman", 1 );  
    ...continued below
```

接下来，我们要创建一个随机数产生器，一幅用于绘制的图像和 Kalman 滤波器结构。注意，需要指定 Kalman 滤波器状态变量的维数(2)和测量变量的维数(1)。

```

...continued from above
CvRandState rng;
cvRandInit( &rng, 0, 1, -1, CV_RAND_UNI );

IplImage* img = cvCreateImage( cvSize(500,500), 8, 3 );
CvKalman* kalman = cvCreateKalman( 2, 1, 0 );
. . .continued below

```

一旦有了这些块之后，我们为状态 x_k 、过程噪声 w_k 、测量值 z_k 和最重要的传递矩阵 F 创建矩阵(实际上是向量，但在 OpenCV 中我们把所有的都称为矩阵)。状态需要被初始化，我们将其初始化为一些分布在 0 附近的合理的随机数。

由于传递矩阵联系着系统在时间 t 和时间 $t+1$ 的状态，所以它是至关重要的。在此例中，传递矩阵是 2×2 的(因为状态向量是二维的)。实际上，传递矩阵赋予了状态向量元素意义。我们将 x_k 视为代表汽车的角度位置(ϕ)和角速度(ω)。这样，传递矩阵就为 $\begin{bmatrix} 1, dt \\ 0, 1 \end{bmatrix}$ 。因此，当乘以 F 之后，状态 (ϕ, ω) 变成 $(\phi + \omega dt, \omega)$ ——即角速度保持不变而角度位置增加角速度乘以时间步长。在我们的例子中，为了方便选取 $dt=1.0$ ，但是实际上需要选取类似连续视频帧间的时间的值。

```

. . .continued from above
// state is (phi, delta_phi) - angle and angular velocity
// Initialize with random guess.
//
CvMat* x_k = cvCreateMat( 2, 1, CV_32FC1 );
cvRandSetRange( &rng, 0, 0.1, 0 );
rng.disttype = CV_RAND_NORMAL;
cvRand( &rng, x_k );

// process noise
//
CvMat* w_k = cvCreateMat( 2, 1, CV_32FC1 );

// measurements, only one parameter for angle
//
CvMat* z_k = cvCreateMat( 1, 1, CV_32FC1 );
cvZero( z_k );

// Transition matrix 'F' describes relationship between
// model parameters at step k and at step k+1 (this is
// the "dynamics" in our model)
//

```

```
const float F[] = { 1, 1, 0, 1 };
memcpy( kalman->transition_matrix->data.fl, F, sizeof(F));
. . .continued below
```

【360~361】

Kalman 滤波器还有必须被初始化的其他内部参数。特别地， 1×2 的测量矩阵 H 被一个不直接的恒等函数初始化为 $[1, 0]$ 。过程噪声和测量噪声的协方差被设置为合理的但是比较有趣的值(你可以自己试一试)，后验误差协方差也被初始化单位矩阵(这样做是为了保证第一次迭代有意义；它的值随后会被改写)。

类似地，由于在此时没有任何信息，我们将(第一个前假象阶段的)后验状态初始化为一个随机值。

```
. . .continued from above
// Initialize other Kalman filter parameters.
//
cvSetIdentity( kalman->measurement_matrix,
cvRealScalar(1) );
cvSetIdentity( kalman->process_noise_cov, cvRealScalar(1e-5) );
cvSetIdentity( kalman->measurement_noise_cov, cvRealScalar(1e-
1) );
cvSetIdentity( kalman->error_cov_post, cvRealScalar(1));

// choose random initial state
//
cvRand( &rng, kalman->state_post );

while( 1 ) {
. . .continued below
```

最后，我们可以在实际动态系统上开始预测了。首先，用 Kalman 滤波器预测它所认为在此阶段会产生的什么(也就是在获得任何新的信息之前)；我们称为 y_k 。然后为这次迭代产生 z_k (测量值)的新值。根据定义，这个值是“实际的”值 x_k 乘以测量矩阵 H 再加上随机测量噪声。必须在这里提一下，除了类似这个简单应用之外，不能从 x_k 产生 z_k ，而是由环境状态或传感器来产生一个生成函数。在这个模拟的例子中，我们添加随机噪声从一个基本的“实际”数据模型中产生测量值；这样就可以看到 Kalman 滤波器的作用了。

【361~362】

```
. . .continued from above
// predict point position
const CvMat* y_k = cvKalmanPredict( kalman, 0 );
```

```

// generate measurement (z_k)
//
cvRandSetRange(
    &rng,
    0,
    sqrt(kalman->measurement_noise_cov->data.fl[0]),
    0
);
cvRand( &rng, z_k );
cvMatMulAdd( kalman->measurement_matrix, x_k, z_k, z_k );
. . .continued below.

```

画出由先前合成的观测值、由 Kalman 滤波器预测的位置和基本状态(在这个模拟的例子中我们恰巧知道)所代表的三个点。

```

. . .continued from above
// plot points (eg convert to planar coordinates and draw)
//
cvZero( img );
cvCircle( img, phi2xy(z_k), 4, CVX_YELLOW );
// observed state
cvCircle( img, phi2xy(y_k), 4, CVX_WHITE, 2 );
// "predicted" state
cvCircle( img, phi2xy(x_k), 4, CVX_RED ); // real state
cvShowImage( "Kalman", img );
. . .continued below

```

到此，我们可以开始下一次迭代了。首先要做的是在此调用 Kalman 滤波器并赋予其最新的测量值。接下来就是产生过程噪声。然后对 x_k 乘以时间传递矩阵 F 完成一次迭代并加上我们产生的过程噪声，现在，我们又可以开始新一轮计算。

```

. . .continued from above
// adjust Kalman filter state
//
cvKalmanCorrect( kalman, z_k );

// Apply the transition matrix 'F' (e.g., step time forward)
// and also apply the "process" noise w_k.
//
cvRandSetRange(
    &rng,

```

```

    0,
    sqrt(kalman->process_noise_cov->data.f1[0]),
    0
);
cvRand( &rng, w_k );
cvMatMulAdd( kalman->transition_matrix, x_k, w_k, x_k );

// exit if user hits 'Esc'
if( cvWaitKey( 100 ) == 27 ) break;
}

return 0;
}

```

【362~363】

可以看出，Kalman 滤波部分并不十分复杂，有一半所需的代码只是用来产生一些填充信息。在任何情况下，我们都应该总结做过的所有事情，为的是确保所有的都有意义。

首先创建表示系统状态的矩阵和将要获得的测量值的矩阵开始，定义传递矩阵和测量矩阵，然后初始化噪声协方差矩阵和滤波器的其他参数。

在初始化状态向量为一随机值后，调用 Kalman 滤波器并让其做出第一次预测。一旦获得此预测值(虽然此值在第一次中没有太大的意义)，将其在屏幕上绘制出来。同时也合成一个新的观测并在屏幕上绘制出来与滤波器的预测值进行比较。然后，以新测量值的形式传递新信息给滤波器，而这个新的测量值会被合并到滤波器的内部模型中。最后，给模型合成一个新的“真实的”状态，这样我们就可以再次重复这个循环。

运行代码，可以看到小的红色球绕轨道一圈一圈地运动。小的黄色球围绕红色球出现和消失，这表示 Kalman 滤波器正尝试去“看透”的噪声。白色的球收敛在红色球周围的小空间中运动，这说明 Kalman 滤波器在我们的模型中对物体(汽车)的运动做出了合理的估计。

在例子中还没有讨论的一个问题是控制输入的使用。例如，如果这辆车是一辆无线控制的汽车，同时我们知道人会控制其进行什么操作，那么就可以将这个信息加入模型中。这样，速度就由控制器来设定。我们需要补充一个矩阵 $B(kalman->control_matrix)$ ，同时还要为 `cvKalmanPredict()` 调节控制向量 u 提供第二个参数。

扩展 Kalman 滤波器简述

你可能会注意到，要求系统的动力学性能在参数上呈线性是相当苛刻的。当动力学性能不是线性时，Kalman 滤波器对我们仍然有用，OpenCV 中的 Kalman 滤波器函数也可以使用。

再说一下，“线性”的意思(实际上)是 Kalman 滤波器定义中的不同的阶段可以用矩阵来表示。什么时候情况又不是这样的呢？实际有许多可能。例如，假设我们的控制测量是汽车的油门被踩下次数的总数：汽车速度和油门被踩的次数之间的关系就不是线性关系。另一个比较常见的问题是施加在汽车上的力容易用笛卡儿坐标系来表示，而汽车的运动(如我们举的这个例子)更容易用极坐标来表示。这个问题可能在这样的情况中出现，即当用船代替汽车在规则的水流中做圆形运动但同时又向着某一特定的方向前进时。

【363~364】

在所有这样的情况中，仅仅使用 Kalman 滤波器是不够的。处理这些非线性(或者试图去处理)的一种方法是将相关的过程(例如，更新 F 或者控制输入响应 B)线性化。这样，我们需要根据状态 x 在每一个时间步长计算 F 和 B 的新值。这些值只是在某个特定 x 值的附近接近真实的更新和控制函数，但是在实际中已经足够。这个对 Kalman 滤波器的扩展被简单地叫作扩展 Kalman 滤波器[Schmidt66]。

OpenCV 没有提供专门的程序来实现扩展 Kalman 滤波器，但实际上我们不需要这样一个函数。我们所需要做的只是在每次更新前重新计算和设置 `kalman->update_matrix` 和 `kalman->control_matrix` 的值。Kalman 滤波器后来由一个 *unscented* 粒子滤波[Merwe00]的公式被巧妙地扩展到非线性系统。[Thrun05]非常清楚地介绍了 Kalman 滤波器的全部信息，包括最新的发展。

condensation 算法

Kalman 滤波器是基于单个假设的建模。由于此假设的概率分布的基本模型是单高斯的，所以不可能用 Kalman 滤波器同时表达多个假设。要解决这一问题，可以使用一个更先进一点的方法，即 condensation 算法。condensation 算法是建立在称为粒子滤波器的一类更广泛的预估器的基础上的。

为了理解 condensation 算法的意图，假想有一个物体以恒定的速度运动(如 Kalman 滤波器建模的那样)。任何测量的数据实质上都被融合进模型中，假定它们符合这个假设。现在假设有一个被遮挡的运动的物体。我们并不清楚这个物体怎样在运

动，它可能以恒定的速度继续运动，可能停止然后/或者改变运动方向。简单地增大物体位置(高斯)分布的不确定性并不能使 Kalman 滤波器描述这些多种可能性。Kalman 滤波器必须为高斯的，所以它不能表示这种多态分布。

和 Kalman 滤波器相同，有两个函数(分别)用于创建和销毁表示 condensation 滤波器的数据结构。惟一的不同在于对于 condensation 滤波器，创建函数 cvCreateConDensation() 多了一个参数。输给参数的值为滤波器在任意给定时刻都保持不变的假设(也就是“粒子”的数量。这个数字应该相对比较大(50 或者 100，对于复杂的情况可能更大)，因为这些单个的假设将替代 Kalman 滤波器的参数化高斯概率分布。如图 10-20 所示。

【364~365】

```
CvConDensation* cvCreateConDensation(
    int dynam_params,
    int measure_params,
    int sample_count
);

void cvReleaseConDensation(
    CvConDensation** condens
);
```

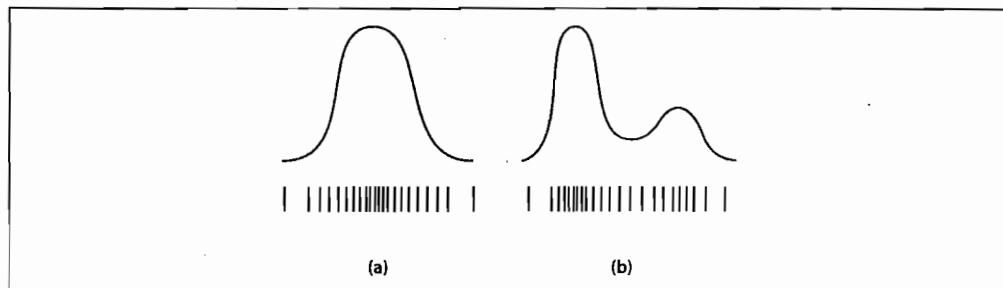


图 10-20：可以(a)和不可以(b)用均值和协方差参数化的连续高斯分布表示的分布；两个分布都可以一组粒子来代替，这组粒子用密度来近似概率分布

这个数据结构含有如下内部元素：

```
typedef struct CvConDensation
{
    int MP;           // Dimension of measurement vector
    int DP;           // Dimension of state vector
    float* DynamMatr; // Matrix of the linear Dynamics system
    float* State;     // Vector of State
```

```

    int SamplesNum;           // Number of Samples
    float** flSamples;       // array of the Sample Vectors,
    float** flNewSamples;    // temporary array of the Sample
                            // Vectors
    float* flConfidence;    // Confidence for each Sample
    float* flCumulative;    // Cumulative confidence
    float* Temp;             // Temporary vector
    float* RandomSample;    // RandomVector to update sample set
    CvRandState* RandS;     // Array of structures to generate
                            // random vectors
} CvConDensation;

```

分配了 condensation 滤波器数据结构后需要初始化这个结构。我们用 cvConDensIniSampleSet() 来实现。在创建 CvConDensation 结构时，需要指出粒子的数目并为每个粒子指定维数。初始化所有的粒子是相当麻烦的^①。幸运的是，cvConDensInitSampleSet() 以一种方便的方法完成了这个任务；我们只需要为每一个维度指定一个范围。

```

void cvConDensInitSampleSet(
    CvConDensation* condens,
    CvMat*           lower_bound,
    CvMat*           upper_bound
);

```

这个函数需要我们初始化两个 CvMat 结构。这两个结构均是向量(意思是它们都只有一列)，并且每个都有与系统状态维数相同的元素个数。这两个向量将被用于设置初始化 CvConDensation 结构中粒子向量的范围。

下面的代码创建了两个大小为 Dim 的矩阵并将它们分别初始化为 -1 和 +1。当调用 cvConDensInitSampleSet() 时，初始的粒子每个将被初始化为在 -1 到 +1 间() 的随机数。所以，如果 Dim 为 3，则滤波器被初始化为均匀分布在中心在原点、边长为 2 的立方体内部的粒子。

```

CvMat LB = cvMat(Dim, 1, CV_MAT32F, NULL);
CvMat UB = cvMat(Dim, 1, CV_MAT32F, NULL);
cvmAlloc(&LB);
cvmAlloc(&UB);

```

^① 当然，如果你了解粒子滤波，那么你就知道这里正是我们用有关系统状态的先验知识(或先验假设)初始化滤波器的地方。初始化滤波器的这个函数只是帮助我们产生一个点集的均匀分布(也就是概率密度函数是平的)。

```

ConDens = cvCreateConDensation(Dim, Dim, SamplesNum);
for( int i = 0; i<Dim; i++ ) {
    LB.data.fl[i] = -1.0f;
    UB.data.fl[i] = 1.0f;
}
cvConDensInitSampleSet(ConDens, &LB, &UB);

```

最后，一个用于更新 condensation 滤波器状态的函数：

```
void cvConDensUpdateByTime( CvConDensation* condens );
```

使用这个函数还需要比我们表面所看到的多做些工作。具体说来，必须根据前一次更新之后可获得的信息来更新所有粒子的置信度。不好的是，OpenCV 中没有为此而设置的方便的函数。原因是粒子新的置信度和新的信息间的关系取决于应用的环境。这里有一个更新的例子，它对滤波器中的每个粒子的置信度进行了简单更新^①。

```

// Update the confidences on all of the particles in the filter
// based on a new measurement M[]. Here M has the
// dimensionality of
// the particles in the filter.
//
void CondProbDens(
    CvConDensation* CD,
    float* M
) {
    for( int i=0; i<CD->SamplesNum; i++ ) {
        float p = 1.0f;
        for( int j=0; j<CD->DP; j++ ) {
            p *= (float) exp(
                -0.05*(M[j] - CD->flSamples[i][j])*(M[j]-CD->
                    flSamples[i][j])
            );
        }
        CD->flConfidence[i] = Prob;
    }
}

```

① 细心的读者会注意到这个更新实际上暗含了一个高斯概率分布，但是你当然可以根据你专门的应用环境选用一个更为复杂的更新。

当置信度被更新后就可以调用 `cvCondensUpdateByTime()` 去更新粒子了。这里“更新”的意思是重采样，即产生与计算出的置信度一致的一组新的粒子。更新之后，所有的置信度又将变成 1.0f，但是粒子分布将会把前一次修改的置信度直接包含进下一次迭代粒子的密度中。

【366~367】

练习

在 `.../opencv/samples/c/` 目录中是一些示例代码程序，演示了本章讨论的很多算法：

- `lkdemo.c` (光流)
 - `camshiftdemo.c` (用 mean-shift 根据颜色跟踪某个区域)
 - `motempl.c` (运动模板)
 - `kalman.c` (Kalman 滤波器)
1. `cvGoodFeaturesToTrack()` 中的协方差 Hessian 矩阵是在图像中的正方形区域上计算得出的正方形区域大小由此函数的 `block_size` 设置。
 - a. 从概念上来讲，当 `block_size` 增加是会出现什么情况？我们会得到更多的还是更少的“好的特征”？为什么？
 - b. 仔细阅读 `lkdemo.c` 代码，找到 `cvGoodFeaturesToTrack()`，改变 `block_size` 的大小，然后运行查看有何不同。
 2. 参考图 10-12，思考实现寻找亚像素级角点的函数 `cvFindCornerSubPix()`。
 - a. 在图 10-12 中，如果棋盘格扭曲了，直的黑白线形成交于一点的曲线时，会发生什么情况？亚像素级角点还能找到吗？请给出解释。
 - b. 如果扩大扭曲的棋盘格的角点周围窗口的尺寸(扩大 `win` 和 `zero_zone` 参数)，寻找亚像素会更精确呢还是它会开始发散？请解释你的答案。
 3. 光流
 - a. 描述一个用块匹配跟踪比 Lucas-Kanade 光流跟踪更好的物体。
 - b. 描述一个用 Lucas-Kanade 光流跟踪比块匹配更好的物体。
 4. 编译 `lkdemo.c`。连上一个网络摄像机(或者用以前拍摄的一个有纹理的运动物体的序列)。在运行此程序中，“r”是自动初始化跟踪，“c”是清除跟踪，单击鼠标添加一个新的点或删除一个已有的点。运行 `lkdemo.c`，同时键入

“r” 初始化点的跟踪。观察效果。

- a. 现在进入代码，去掉亚像素级角点定位函数 `cvFindCornerSubPix()`。这样做影响结果吗？怎样影响的？

- b. 再次进入代码，不调用 `cvGoodFeaturesToTrack()`，在物体周围的 ROI 中标记一些格子的点。描述这些点会产生什么变化并说明为什么。

提示：所发生的部分是由于孔径问题造成的——给定一个固定窗口尺寸和一条直线，我们分辨不出这条线是如何运动的。

5. 修改 `lkdemo.c` 程序，使其对轻微的摄像机运动进行的简单图像稳定。在比摄像机的输出大得多的窗口中心显示稳定的结果(当第一组点保持稳定时帧画面可能抖动)。

6. 用一个网络摄像机或一个运动的有颜色的物体的彩色视频，编译和运行 `camshiftdemo.c`。用鼠标画一个(刚好)包含这个运动物体的方框；这个程序将会跟踪此物体。

- a. 在 `camshiftdemo.c` 中，用 `cvMeanShift()` 替换 `cvCamShift()` 函数。描述一个跟踪器比另一个跟踪器跟踪效果更好的情况。

- b. 编写一个在初始的 `cvMeanShift()` 区域中标记一组点的函数。立即运行两个跟踪器。

- c. 这两个跟踪器如何结合使用会让跟踪更稳定？解释原因并且演示。

7. 使用网络摄像机或者事先存储的视频文件，编译并运行运动模板代码 `motempl.c`。

- a. 修改 `motempl.c`，让它可以进行简单的姿态识别。

- b. 如果摄像机在运动，阐述一下怎样使用练习 5 中的运动稳定的代码让运动模板对轻微运动的摄像机也有效。

8. 描述一下用一个线性状态模型(非扩展)Kalman 滤波器怎样去跟踪圆周(非线性)运动。

提示：怎样预处理这种情况来得到线性的动态？

9. 建立一个当前状态取决于先前状态的位置和速度的运动模型。将 `lkdemo.c`(只使用一些 click 点)结合 Kalman 滤波器更好地跟踪 Lucas-Kanade 点。显示每一个点的不确定性。这个跟踪在哪里会失败？

提示：将 Lucas-Kanade 作为 Kalman 滤波器的观测模型，调整噪声使其可以跟

踪。保持运动的合理性。

10. Kalman 滤波器依赖于线性动态性和 Markov 独立性(也就是它假设当前状态只依赖于刚过去的状态而不是所有过去的状态)。假设要跟踪一个运动与先前的位置和速度相关的物体，但是你错误地使用了一个动态项使状态依赖于先前的位置——换句话说，遗漏了先前的速度项。
 - a. Kalman 的假设仍然还成立吗？如果成立，说明为什么？如果不成立，说明为什么不满足假设。
 - b. 怎样建立 Kalman 滤波器使其在动态项的某些部分被漏掉时仍可以进行跟踪？
提示：想一想噪声模型。
11. 使用一个网络摄像机或者一段视频，内容为一个人挥舞双手，每个手中都有一个明亮颜色的物体。用 condensation 去跟踪两只手。

摄像机模型与标定

视觉开始于检测大千世界的自然光线。从某种发射源(如灯管、太阳等)发出的射线形成光线，然后穿过空间照射到某些物体上。照射到物体上的大部分光线被物体表面吸收，而只有少部分没有吸收的光线被我们所察觉，并形成光线的颜色。反射的光线进入眼睛(或者摄像机)，为视网膜(或图像采集器)吸收。光线从物体发射开始，通过透镜到达眼睛或摄像机，然后到达视网膜或者图像采集器的这个过程的几何研究，是实用计算机视觉中的一个尤其重要的方面。

一个简单而有用的解释这种现象的模型就是针孔摄像机模型^①。针孔是一堵想像中的墙(中心有一个微型小孔)，光线只能从这个开口中通过，而其余的都被墙所阻挡。在本章，我们将从一个针孔摄像机模型开始，处理基本几何中的投影射线。遗憾的是，真实的针孔由于不能为快速曝光收集足够的光线，因此它不是一个得到图像的好方法。这也是为什么眼睛和摄像机都要使用透镜而不是仅仅只用一个点来收集更多光线的原因。然而，这种利用透镜得到更多的光线的缺点是，不仅使我们背离了所使用的简单针孔几何模型，而且引入来自透镜的畸变。

在本章中，我们将学习如何利用摄像机标定(camera calibration)，来矫正(数学方式)因使用透镜而给针孔模型带来的主要偏差。摄像机标定的重要性还在于它是摄像机

^① 基于透镜的知识至少可以上溯到古罗马时期。针孔摄像机模型则至少上溯到公元 987 年哈桑[1021](译者注：阿拉伯著名数学家)的理论，它是介绍视觉几何方面的经典方法。在十七、十八世纪，伴随着笛卡儿、开普勒、伽利略、牛顿、霍克、欧拉、费马、施耐尔等人的贡献，数学和物理学不断发展并成长(见 O'Connor [O'Connor02])。一些几何视觉方面的关键内容包括 Trucco [Trucco98], Jaehne (有时也称 Jähne) [Jaehne95; Jaehne97], Hartley 与 Zisserman [Hartley06], Forsyth 与 Ponce [Forsyth03], Shapiro 与 Stockman [Shapiro02]，和 Xu 与 Zhang [Xu96]等。

测量与真实三维世界测量的联系桥梁，场景不仅仅是三维的，也是用物理单位度量的空间。因此，摄像机的自然单位(像素)和物理世界的单位(米)的关系对三维场景的重构至关重要。

【370~371】

摄像机标定的过程既给出摄像机几何模型，也给出透镜的畸变模型。这两个模型定义了摄像机的内参数(intrinsic parameter)。本章将应用这些模型来矫正透镜畸变。在第 12 章，我们将用这些模型来阐述物理场景。

我们将从摄像机模型和透镜畸变开始。在这里，我们将研究单应变换(homograph transform)，它是一种能够描述摄像机基本行为特性和各种失真、矫正特性的数学工具。我们将花费一点时间来仔细讨论如何以数学方式描述和计算这种变换。一旦我们理解这个工具，就可以转移到 OpenCV 函数完成大部分工作。

本章阐述的所有知识是为了建立足够的理论基础，以便读者能够完整理解 OpenCV 函数 cvCalibrateCamera2()，并且知道其外表下的内涵。这对正确使用该函数是非常重要的。这里说明一下，如果读者已经是一个专家或者只是想简单地了解如何使用 OpenCV 的相关函数，那么可以直接转到后面的“标定函数”一节。

摄像机模型

我们先看摄像机模型中最简单的针孔模型。在此模型中，想像着光线是从场景或很远的物体发射过来的，但仅仅是来自某点的一条光线。在实际针孔摄像机中，该点被“投影”到成像表面。其结果是在图像平面(也称为投影平面，projective plane)上，图像被聚焦。因此与远处物体相关的图像大小只用一个摄像机参数来描述：焦距(focal length)。对于假想的针孔摄像机，从针孔到屏幕的距离就是焦距。如图 11-1 所示， f 是摄像机焦距， Z 是摄像机到物体的距离， X 是物体长度， x 是图像平面上的物体图像。其数值可以通过相似三角形 $-x/f = X/Z$ 得到，或

$$-x = f \frac{X}{Z}$$

重新把针孔摄像机模型整理为另一种等价形式，使其数学形式更简单一些。在图 11-2 中，我们交换针孔和图像平面^①，主要的差别是现在物体出现在等式右边。针孔中的点被理解为投影中心(center of projection)。这样，每一条光线，从远处物体

① 这种数学抽象无法在物理上重构。图像平面可以简单地设想为所有抵达投影中心的光束的“切片”。这种处理更容易绘制而且数学上的处理也更容易。

的某个点出发，到达投影平面的中心。光轴与图像平面的交点被称为主点(principal point)。在这个与旧的投影平面(或图像平面)等价的新前端图像平面上(见图 11-2)，远处物体的图像与图 11-1 中的图像大小完全一致。光束与图像平面的相交生成图像，而平面到投影中心的距离是 f 。这样形成更容易理解的三角形相似关系 $x/f = X/Z$ 。负号被去掉了，因为目标图像不再是倒立的。

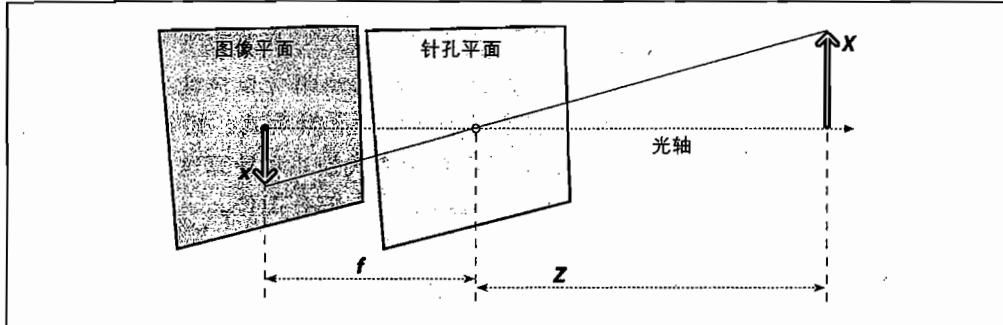


图 11-1：针孔摄像机模型。过空间某特定点的光线才能通过针孔，这些光束被投影到图像平面形成图像

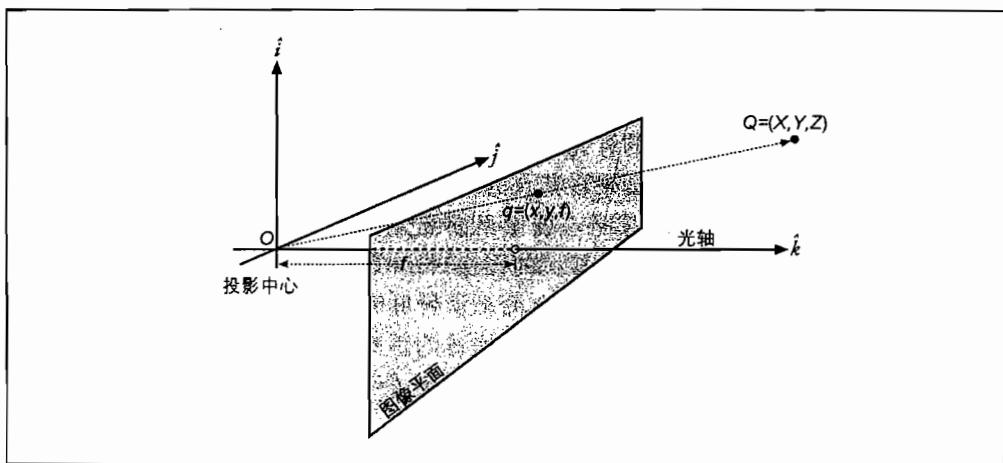


图 11-2：点 $Q = (X, Y, Z)$ 由通过投影中心的光线投影到图像平面上，相应的图像点为 $q = (x, y, f)$ 。图像平面实际上就是把投影屏幕放置到针孔的前方(数学上等价，但形式简单些) 【371~372】

你也许认为主点即等于成像仪的中心，但这意味着某些人拿着镊子和胶水要把摄像机里面的成像仪以微米级别的精度安装。实际上，芯片的中心通常不在光轴上。我们因此引入两个新的参数 c_x 和 c_y 。对可能的偏移(对光轴而言)进行建模。这样物

理世界中的点 Q , 其坐标为 (X, Y, Z) , 以某些偏移的方式投影为点 $(x_{\text{screen}}, y_{\text{screen}})$, 如下所示^①:

$$x_{\text{screen}} = f_x \left(\frac{X}{Z} \right) + c_x, \quad y_{\text{screen}} = f_y \left(\frac{Y}{Z} \right) + c_y$$

注意, 我们引入了两个不同的焦距。原因是单个像素点在低价成像仪上是矩形而不是正方形的。例如, 焦距 f_x 实际上是透镜的物理焦距长度与成像仪每个单元尺寸 s_x 的乘积(这样做的意义在于 s_x 的单位是像素/每毫米^②, 而 F 的单位是毫米, 这意味着 f_x 的单位是像素)。同样的道理也适用于 f_y 和 s_y 。请记住, s_x 和 s_y 以及物理焦距 F 均不能在摄像机标定过程中直接测量, 只有组合量 $f_x=Fs_x$ 和 $f_y=Fs_y$ 可以直接计算出来而不必拆除摄像机去直接测量其部件。

基本投影几何

将坐标为 (X_i, Y_i, Z_i) 的物理点 Q_i 映射到投影平面上坐标为 (x_i, y_i) 的点的过程叫投影变换 (projective transform)。采用这种变换, 可以方便地使用我们所熟知的齐次坐标。齐次坐标把维数为 n 投影空间上的点用 $(n+1)$ 维向量(如 x, y, z 变为 x, y, z, w)表示, 其额外限制是任何两点的交比不变。在这里, 图像平面是一个二维投影空间, 因此可以用一个三维向量 $q=(q_1, q_2, q_3)$ 来表示该平面上的点。因为投影空间上所有点的交比不变, 因此可以通过除以 q_3 计算实际的像素坐标值。这样允许我们将定义摄像机的参数(如 f_x, f_y, c_x 和 c_y)重新排列为一个 3×3 的矩阵, 该矩阵称为摄像机的内参数矩阵(camera intrinsics matrix), 在 OpenCV 中, 摆像机内参数矩阵的计算来自 Heikkila 和 Silven [Heikkila97])。那么将物理世界中的点投影到摄像机上, 可以用下式表示:

$$q = MQ, \text{ 其中 } q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

展开该式, 可以发现 $w=Z$, 并且点 q 是齐次坐标形式。除以 w (或 Z), 就可以恢复以前的定义。(没有减号是因为我们是在针孔前面来观看投影平面的非倒立图像, 而不是在针孔后面看投影平面的倒立图像)。

【373~374】

- ① 这里下标 “screen” 是说明计算出的坐标是屏幕(或成像仪)上面的坐标。方程式中 $(x_{\text{screen}}, y_{\text{screen}})$ 和图 11-2 中 (x, y) 的差别在于 c_x 和 c_y 所描述的点。所以, 我们用 “screen” 下标来描述成像仪坐标。
- ② 当然, “毫米” 仅仅是一个独立的度量单位, 也可以是“米”或者其他。关键在于 s_x 将物理单位转换为像素单位。

讨论齐次坐标的时候，OpenCV 库中有一个函数 `cvConvertPointsHomogenous()`^①，使用该函数可以很方便地对齐次坐标进行转换。该函数在很多其他场合也非常有用。

```
void cvConvertPointsHomogenous(
    const CvMat* src,
    CvMat* dst
);
```

不要被简单的变量所迷惑。程序做了一整套有用的事情。输入的数组 `src` 可以是 $M_{src} \times N$ 或 $N \times M_{src}$ (这里 $M_{src} = 2, 3$ 或 4)，也可以是 $1 \times N$ 或者 $N \times 1$ ，其中 $M_{src} = 2, 3$ 或 4 通道(N 可以是任意数，它实际上是 `src` 中要转换的点的个数)。输出数组 `dst` 可以是任意形式，只是 M_{dst} 的维数必须与 $M_{src}, M_{src} - 1$ 或 $M_{src} + 1$ 一致。

当输入维数 M_{src} 等于输出维数 M_{dst} ，数据就是简单的复制(而且如果必要，则进行转置)。如果 $M_{src} > M_{dst}$ ，那么用 `dst` 的元素除以 `src` 中对应向量的所有但最后一个元素除外(也就是说，`src` 假设为齐次坐标)。如果 $M_{src} < M_{dst}$ ，那么所有点被复制到 `dst`，并且将 1 插入到 `dst` 数组中每个向量的末尾。在这些场合，如同偶尔出现的 $M_{src} = M_{dst}$ 情况下，都做必要的转置。

注意：关于该函数，需要提醒一下，在某种情况(如 $N < 5$)下输入和输出维数是不确定的。这样函数会抛出异常。如果发现这种现象，可以用某些伪造的数据排列到矩阵后面。另外，有时用户会传入多通道 $N \times 1$ 或 $1 \times N$ 矩阵，其中通道数目为 M_{src} (M_{dst})，此时可以使用函数 `cvReshape()` 将单通道矩阵转换为多通道矩阵而不用复制任何数据。

【374】

采用理想针孔，我们有了一个对视觉中的三维几何有用的模型。但请记住，由于只有很少量的光线通过针孔，这导致实际情况下因曝光不足使得图像生成得很慢。对要快速生成图像的摄像机而言，必须利用大面积且弯曲的透镜，让足够多的光线能够收敛聚焦到投影点上。为了实现该目的，我们用透镜。透镜可以聚焦足够多的光线到某个点上，使得图像的生成更加迅速。其代价就是引入了畸变。

透镜畸变

理论上讲是可能定义一种透镜而不引入任何畸变的。然而现实世界没有完美的透

① 是的，“Homogenous”拼写错误。

镜。这主要是制造上的原因，因为制作一个“球形”透镜比制作一个数学上理想的透镜更容易。而且从机械方面也很难把透镜和成像仪保持平行。这里我们主要描述两种主要的透镜畸变并且为它们建模^①。径向畸变来自于透镜形状，而切向畸变则来自于整个摄像机的组装过程。

首先从径向畸变开始。实际摄像机的透镜总是在成像仪的边缘产生显著的畸变。这个头痛现象来源于“筒形”或“鱼眼”影响(参见图 11-12 顶端的房子的分割线)。

图 11-3 给出一些产生径向畸变的直观提示。对某些透镜，光线在远离透镜中心的地方比靠近中心的地方更加弯曲。对常用的普通透镜来说，这种现象更加严重。筒形畸变在便宜的网络摄像机中非常厉害，但在高端摄像机中不明显，因为这些透镜系统做了很多消除径向畸变的工作。

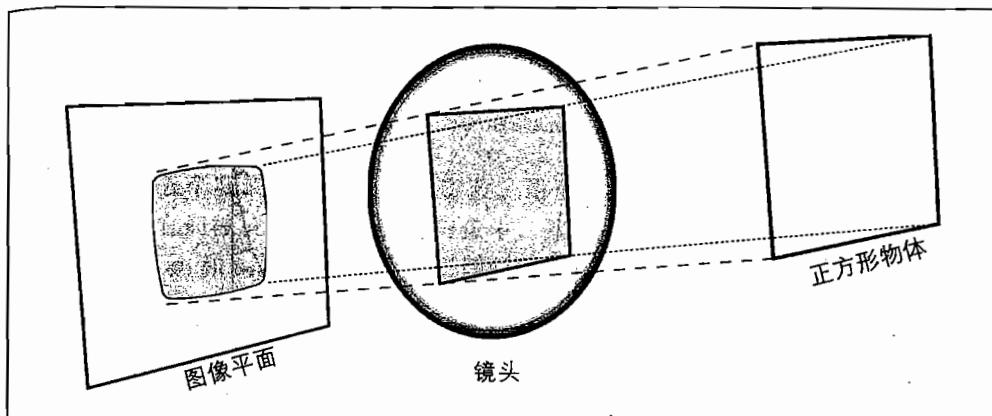


图 11-3：径向畸变。远离透镜中心的光线弯曲比靠近中心的严重。因此正方形的边在图像平面上为弯曲(即筒形畸变)

对径向畸变，成像仪中心(光学中心)的畸变为 0，随着向边缘移动，畸变越来越严重。实际情况中，这种畸变比较小，而且可以用 $r = 0^\circ$ 位置周围的泰勒级数展开的前几项来定量描述。对便宜的网络摄像机，我们通常使用前两项，其中第一项通常

- ① 透镜畸变建模方法主要来自于 Brown [Brown71] 以及更早的 Fryer 和 Brown [Fryer86]。
- ② 如果不懂泰勒级数的定义，也不要过于担心。泰勒级数是用多项式形式在某些特定点附近来表示一个复杂函数的数学技术(或者说多项式序列，序列越长，逼近的精度越高)。在我们所应用的场合，我们在 $r = 0$ 附近把畸变函数展开为多项式形式，这个多项式的通用形式是 $f(r) = a_0 + a_1 r + a_2 r^2 + \dots$ ，在这里，当 $r = 0$ 时， $f(r) = 0$ ，这意味着 $a_0 = 0$ ，因为函数必须是关于 r 径向对称，所以，只有 r 的偶次幂项不为 0。因此，畸变函数只需要用 r^2, r^4 和 (有时) r^6 项的系数来描述。

为 k_1 , 而第二项为 k_2 。对畸变很大的摄像机, 比如鱼眼透镜, 我们可以使用第三径向畸变项 k_3 。通常, 成像仪某点的径向位置按下式进行调节:

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

这里 (x, y) 是畸变点在成像仪上的原始位置, $(x_{\text{corrected}}, y_{\text{corrected}})$ 是校正后的新位置。图 11-4 显示矩形网格因径向畸变而产生的位移。从前面看, 光心越向外, 矩形网格上的点位移越大。

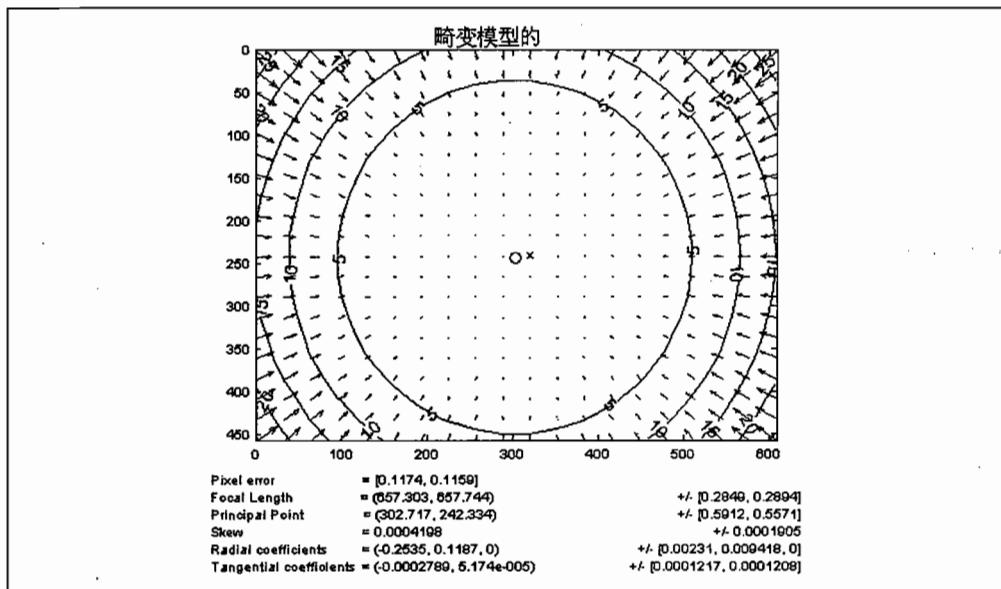


图 11-4: 某个摄像机透镜的镜像畸变图, 箭头显示径向畸变图像上外部矩形网格的偏移(经 Jean-Yves Bouguet 授权)

第二大常用畸变是切向畸变。这种畸变是由于透镜制造上的缺陷使得透镜本身与图像平面不平行而产生的。参见图 11-5。

【375 ~ 376】

切向畸变可以用两个额外参数 p_1 和 p_2 来描述, 如下^①:

^① 这些公式的推导超出本书的范围。有兴趣的读者可以参考“铅锤”模型, 见 D. C. Brown 的, “Decentering Distortion of Lenses”, Photometric Engineering 32(3) (1966), 444–462.

$$x_{\text{corrected}} = x + [2p_1y + p_2(r^2 + 2x^2)]$$

$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2x]$$

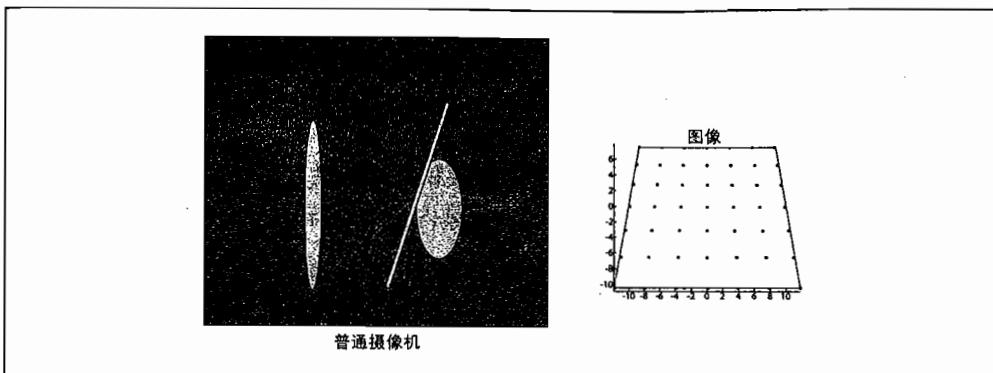


图 11-5：当透镜不完全平行于图像平面的时候产生切向畸变，这种现象发生于成像仪被粘贴在摄像机的时候(经 Sebastian Thrun 授权)

因此总共有五个我们所需要的畸变参数。由于在 OpenCV 程序中五个参数都是必需的，所以它们被放置到一个畸变向量中，这是一个 5×1 的矩阵，按顺序依次包含 k_1 , k_2 , p_1 , p_2 和 k_3 。图 11-6 给出了切向畸变在前面外部矩形网格点的影响。这些点明显在位置和半径上有位移。

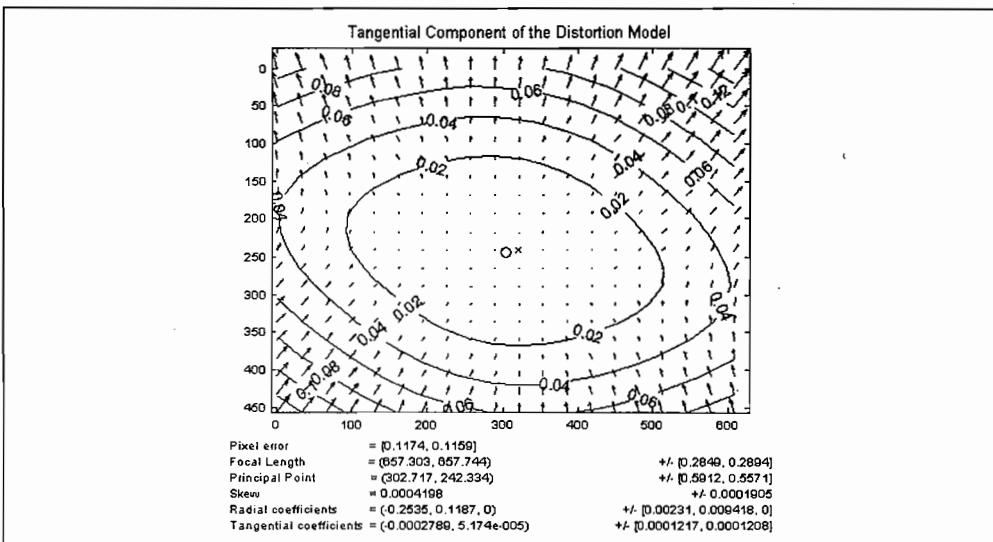


图 11-6：某摄像机透镜的切向畸变图，箭头显示切向畸变图像上外部矩形网格的偏移(经 Jean-Yves Bouguet 授权)

在图像系统中还有许多其他类型的畸变，不过都没有径向和切向畸变显著。因此不论我们，还是 OpenCV，都不打算处理它们。

【376~377】

标定

现在，我们已经在数学意义上对摄像机内参数和畸变特性有了一些概念。接下来一个很自然的问题是，如何利用 OpenCV 来计算内参数矩阵和畸变向量^①。

OpenCV 提供了好几种算法来计算摄像机的内参数。实际的标定是通过 cvCalibrateCamera2() 完成的。在这个程序中，标定方法是把摄像机对准一个有很多独立可标识点的物体。通过在不同角度观看这个物体，可以利用通过每个图像计算摄像机的相对位置和方向以及摄像机的内参数(见图 11-9)。为了提供不同视角，我们旋转和移动物体。这样，让我们先中断一下，先来学点旋转与平移的知识。

【378】

旋转矩阵与平移向量

对每一幅摄像机得到的特定物体的图像，我们可以在摄像机坐标系统上用旋转和平移来描述物体的相对位置，见图 11-7。

通常，任何维的旋转可以表述为坐标向量与合适尺寸的方阵的乘积。最终一个旋转等价于在另一个不同坐标系下对点位置的重新表述。坐标系旋转角度 θ 则等同于将目标点围绕坐标原点反方向旋转同样的角度 θ 。图 11-8 显示用矩阵乘法对二维旋转的描述。在三维空间中，旋转可以分解为绕各自坐标轴的二维旋转，其中旋转轴线的度量保持不变。如果依次绕 x , y 和 z 轴旋转角度 ψ , φ 和 θ° ，那么总的旋转矩阵 R 是三个矩阵 $R_x(\psi)$, $R_y(\varphi)$ 和 $R_z(\theta)$ 的乘积，其中：

$$R_x(\psi) \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix}$$

-
- ① 关于摄像机标定的一个优秀网上教程，请见 Jean-Yves Bouguet 的标定网站 (http://www.vision.caltech.edu/bouguetj/calib_doc)。
 - ② 需要清楚一点，这里所描述的旋转是先绕 z 轴，然后在新位置绕 y 轴，最后再绕 x 轴旋转。

$$R_y(\varphi) \begin{bmatrix} \cos \varphi & 0 & -\sin \varphi \\ 0 & 1 & 0 \\ \sin \varphi & 0 & \cos \varphi \end{bmatrix}$$

$$R_z(\theta) \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

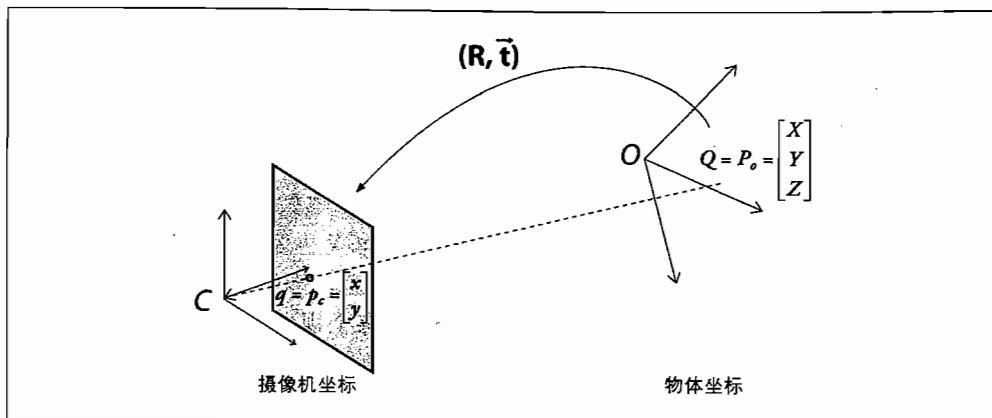


图 11-7：从物体到摄像机坐标系统的变换：物体上的点 P 对应图像平面上的点 p 。点 p 通过旋转矩阵 R 和平移向量 t 的应用与点 P 相连

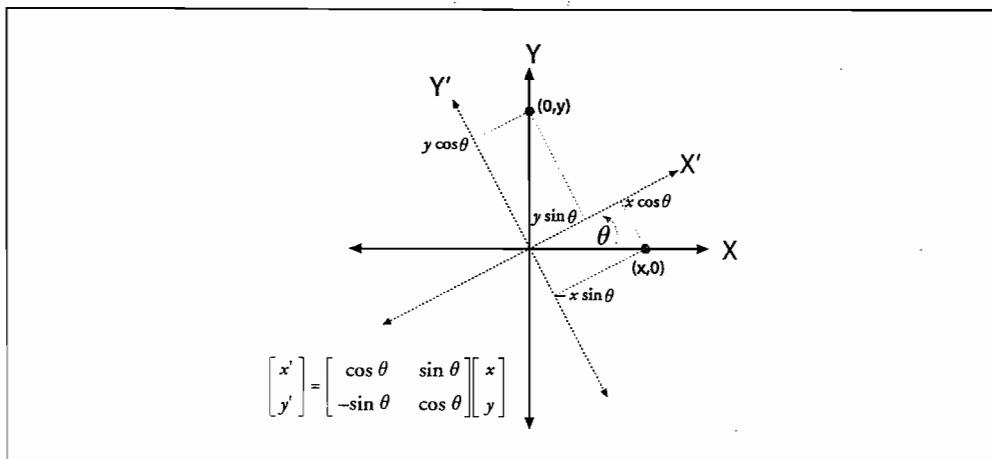


图 11-8：把点旋转 θ (这里是绕 z 轴)等价于坐标轴反向旋转 θ 。通过简单的三角计算，我们可以看出旋转如何改变点的坐标

因此 $R = R_z(\theta), R_y(\varphi), R_x(\psi)$ 。旋转矩阵 R 的特性是它的逆阵就是它的转置阵(我们只

需转回来)。因此有 $R^T R = R^T I = I$, 其中 I 是对角元素为 1, 其余为 0 的单位矩阵。

平移向量用来表示怎样将一个坐标系的原点移动到另一个坐标系的原点, 或者说, 平移向量是第一个坐标系原点与第二个坐标系原点的偏移量。因此, 从以目标中心为原点的坐标系移动到以摄像机中心为原点的另一个坐标系, 相应的平移向量为 $T = \text{目标原点} - \text{摄像机原点}$ 。那么有图 11-7, 点在世界坐标系中的坐标 P_o 到在摄像机坐标系中的坐标 P_c :

$$P_c = R(P_o - T)$$

【379~380】

结合上面计算 P_c 的公式以及摄像机内参数校正, 我们构造出 OpenCV 所能解决的基本方程式。这些方程的解将是我们所寻找的摄像机标定参数。

我们已经看到可以用三个角度来表示三维旋转, 用三个参数(x, y, z)来表示三维平移。因此我们总共有 6 个参数。对摄像机而言, OpenCV 内参数矩阵有 4 个参数(f_x, f_y, c_x 和 c_y), 因此对每个视场的解需要 10 个参数(注意, 摄像机内参数在不同视场保持不变)。使用一个平面物体, 我们很快可以看到每个视场固定 8 个参数。因为不同视角下旋转和平移的 6 个参数会变化, 对每一个视角用来求解摄像机内参数矩阵的两个额外参数需要约束。因此求解全部几何参数至少需要两个视角。

本章稍后将进一步讨论这些参数和相关约束。但首先需要讨论标定物(calibration object)。OpenCV 所使用的标定物是一个用不同黑白方块构成的平面格子, 通常称为“棋盘”(即使它没有 8 个方块, 甚至每个方向的方块个数不相等)。

棋盘

原理上, 任何合适的表征物体都可以用作标定物体, 而实际上都选用诸如棋盘这样的规则模式^①【174】。文献中一些标定方法是基于三维物体的(如有标记点的盒子), 但是平面棋盘模式更容易处理, 因为精确的三维标定物体难以制作(存储和发布)。因此 OpenCV 不是使用基于 3D 构造物体的视场, 而是使用平面物体(如棋盘)的多个视场。我们使用黑白方块交替排列的模式(见图 11-9), 这保证在测量上任何一边都没有偏移。而且, 格线角点也让亚像素定位函数(在第 10 章讨论)的使用更为自然。

给定一个棋盘图像(或者一个人手持棋盘, 或任何包含棋盘的视场以及没有任何干

① 标定物的具体使用以及更多的标定方法来自 Zhang [Zhang99; Zhang00] 和 Sturm [Sturm99]。

的背景), 可以使用 OpenCV 函数 `cvFindChessboardCorners()` 来定位棋盘的角点。

```
int cvFindChessboardCorners(
    const void*      image,
    CvSize           pattern_size,
    CvPoint2D32f*    corners,
    int*             corner_count = NULL,
    int              flags        = CV_CALIB_CB_ADAPTIVE_THRESH
);
```

【381】

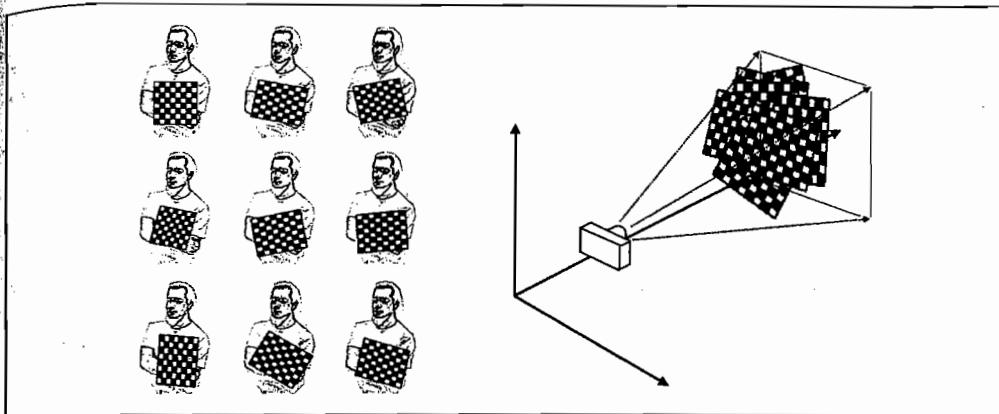


图 11-9：手持棋盘以各种方向得到的棋盘图像，以确保为完全求解这些图像在整个坐标系(相对于摄像机)的位置和摄像机内参数提供足够的信息

该函数输入的变量是包含棋盘的单幅图像。此图像必须是 8 位灰度(单通道)图像。第二个变量 `pattern_size`, 表示棋盘的每行和每列有多少个角点, 该数值是内角点个数, 因此对一个标准象棋棋盘, 正确的值应该是 `cvSize(7, 7)`^①。下一个变量 `corners`, 是存储角点位置的数组指针。该数组必须事先分配空间, 而且至少必须大于棋盘的所有角点数(对一个标准象棋棋盘为 49)。角点位置的每个数字都是以像素坐标保存。`corner_count` 变量是可选的, 如果不为 `NULL`, 则它是一个指向所记录角点数目的整数指针。如果函数成功找到所有的角点^②, 那么返回非 0, 反

-
- ① 实际使用中, 使用棋盘格为不对称的偶奇维数更方便——比如(5, 6)。这样的偶-奇不对称模式使得棋盘只有一个对称轴, 从而棋盘的方向总是能唯一确定。
 - ② 实际上, 要求会更加严格一些: 不仅所有的角点必须被找到, 而且必须按照期望的按行和列排列。只有找到角点而且正确排列, 函数才正确返回非 0 值。

之，返回 0。最后的 flag 变量可以用来定义额外的滤波步长以有助于寻找棋盘角点。所有变量都可以单独或者以逻辑或的方式组合使用。

- **CV_CALIB_CB_ADAPTIVE_THRESH** cvFindChessboardCorners() 的默
认方式是，首先根据平均亮度对图像进行二值化，但如果设置此标志，则使用
自适应二值化法。【382】
- **CV_CALIB_CB_NORMALIZE_IMAGE** 如果设置该标志，则会在二值化之
前应用函数 cvEqualizeHist() 来归一化图像。
- **CV_CALIB_CB_FILTER_QUADS** 一旦二值化图像以后，算法试图根据棋盘
上黑色方块的投影视场中定位四边形。这是一个逼近的过程，因为四边形的每
个边都被假设为直的，而实际上由于图像的径向畸变，这个不完全成立。如果
这个标志被设置，那么将使用对这些四边形使用其他额外的约束以拒绝错误的
四边形。

亚像素角点

cvFindChessboardCorners() 返回的角点仅仅是近似值。这意味着实际上位置的精度受限于图像设备的精度，即小于一个像素。必须单独使用另外的函数来计算角点的精确位置(用近似位置和图像作为输入)以达到亚像素精度。这个函数与在第 10 章中所使用的用于跟踪的 cvFindCornerSubPix() 一样。在本节中使用该函数并不奇怪，因为棋盘内角点只是更通用的 Harris 角点的一种简单形式。之所以使用棋盘角点仅仅是因为这样做是寻找和跟踪的特别简单方式。如果在第一次定位角点时忽略调用此亚像素精度函数，那么会导致标定的实际错误。

绘制棋盘角点

尤其是在调试程序时，需要在图像上绘制找到的棋盘角点(通常我们用来第一次计算角点图像)。这样我们可以看到投影的角点是否与观察的角点匹配。基于这种思路，OpenCV 提供了一个方便程序来处理这种常用的需求。函数 cvDrawChessboardCorners() 将函数 cvFindChessboardCorners() 发现的所有角点绘制到所提供的图像上。如果没有发现所有的角点，那么得到的角点将用红色圆圈绘制。如果所有的角点被找到，那么角点将使用不同颜色绘制(每行使用单独颜色)，并且把角点以一定顺序用线连接起来。

```
void cvDrawChessboardCorners(  
    CvArr*           image,  
    CvSize            pattern_size,
```

```
CvPoint2D32f* corners,  
int count,  
int pattern_was_found  
);
```

`cvDrawChessboardCorners()`的第一个参数是欲绘制的图像。因为角点是以颜色圆圈表示的，该图像必须是 8 位的彩色图像。在大多数场合下，它是 `cvFindChessboardCorners()` 所使用图像的复制品(但必须预先转换为 3 通道图像)。

【383】

接下来的两个参数 `pattern_size` 和 `corners`，与 `cvFindChessboardCorners()` 中的定义一样。参数 `count` 是一个等于角点数目的整数，最后的参数 `pattern_was_found` 表示是否所有的棋盘模式都被成功找到。这可以设置为 `cvFindChessboardCorners()` 的返回值。图 11-10 显示了在棋盘图像上应用函数 `cvDrawChessboardCorners()` 所得到的结果。

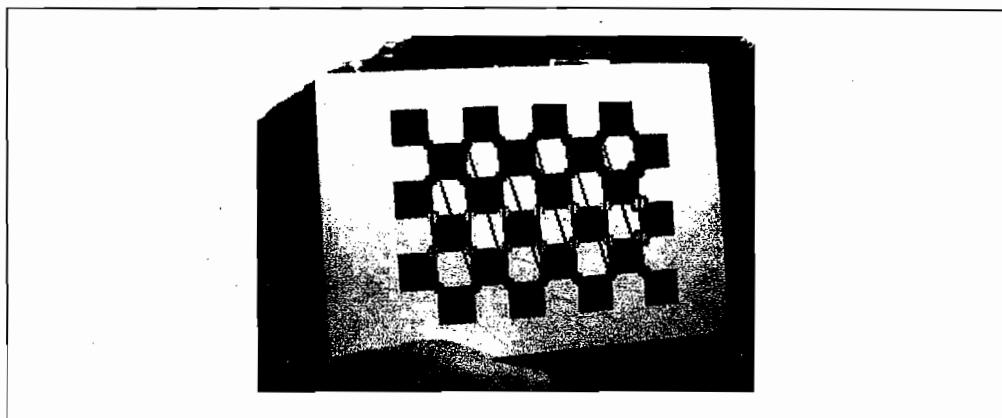


图 11-10：函数 `cvDrawChessboardCorners()` 的结果，一旦用函数 `cvFindChessboardCorners()` 找到角点，便可以将在找到的角点位置(在角点上的小圆圈)以及按顺序(用圆圈之间的线来表示)作投影

现在开始讨论平面物体能够做些什么。当通过针孔和透镜时，平面上的点集是经过透视变换的。该变换的各种参数将被存储到 3×3 单应性矩阵中，即我们将要讨论的内容。

单应性

在计算机视觉中，平面的单应性被定义为从一个平面到另一个平面的投影映

射^①。因此一个二维平面上的点映射到摄像机成像仪上的映射就是平面单应性的例子。如果对点 Q 到成像仪上的点 q 的映射使用齐次坐标，这种映射可以用矩阵相乘的方式表示。若有以下定义：

$$\begin{aligned}\tilde{Q} &= [X \ Y \ Z \ 1]^T \\ \tilde{q} &= [x \ y \ 1]^T\end{aligned}$$

则可以将单应性简单表示为：

$$\tilde{q} = sH\tilde{Q}$$

这里引入参数 s ，它是一个任意尺度比例(目的是使得单应性被定义到该尺度比例)。通常根据习惯放在 H 的外面。

稍微利用一点几何和矩阵代数的知识，便可以求解这个变换矩阵。最重要的是 H 有两部分：用于定位观察的物体平面的物理变换和使用摄像机内参数矩阵的投影。参见图 11-11。

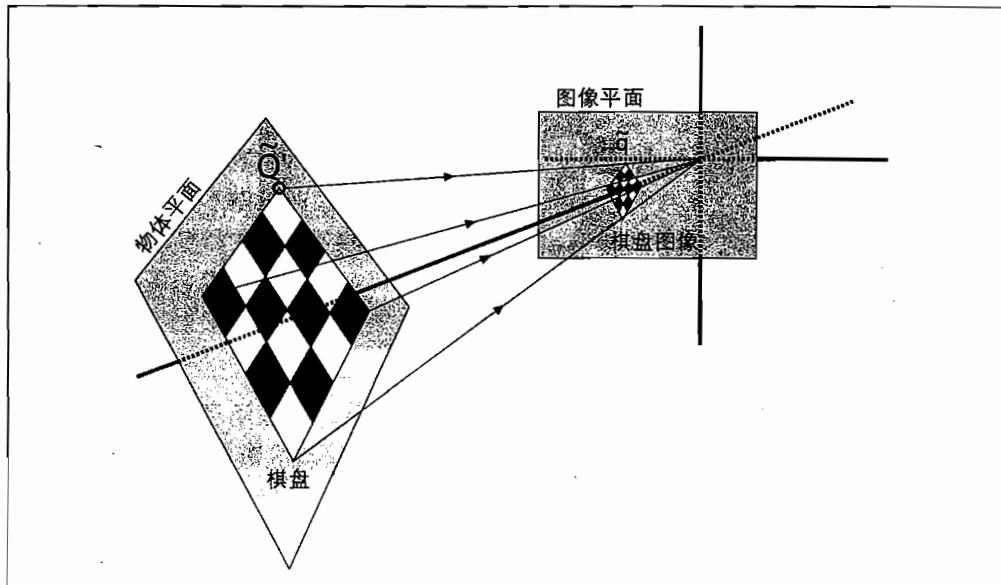


图 11-11：用单应性来描述平面物体的观测：从物体平面到图像平面的映射，同时表征了这个两个平面的相对位置和摄像机投影矩阵

【384~385】

① 术语“单应性”在不同学科上有各种不同的含义。例如，在数学上，它有更通用的意思。计算机视觉中，对单应性最感兴趣的部分只是其他意义中的一个子集。

物理变换部分是与观测到的图像平面相关的部分旋转 R 和部分平移 t 的影响之和。因为使用齐次坐标，我们可以把它们组合到一个单一矩阵中，如下所示^①：

$$W = [R \quad t]$$

然后，通过乘以 $W\tilde{Q}$ ，得到摄像机矩阵 M (用来表示影射坐标)，即：

$$\tilde{q} = sMW\tilde{Q}, \text{ 其中 } M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

这看起来已经完成了。但实际上，我们的关注点不是表征所有空间的坐标 \tilde{Q} ，而只是定义我们所寻找的平面的坐标 \tilde{Q}' 。这需要简化。

考虑到一般性，我们可以选择定义这个物体平面，使得 $Z=0$ 。这样做的原因是如果把旋转矩阵也分解为 3 个 3×1 向量(即 $R = [r_1 \ r_2 \ r_3]$ ，那么其中的一个列向量就不需要了。具体如下：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM[r_1 \ r_2 \ r_3 \ t] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = sM[r_1 \ r_2 \ t] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

映射目标点到成像仪的单应性矩阵 H 可以完全用 $H = sM[r_1 \ r_2 \ t]$ 表述，其中：

$$\tilde{q} = sH\tilde{Q}'$$

注意， H 现在是 3×3 矩阵。

OpenCV 使用上述公式来计算单应性矩阵。它使用同一物体的多个图像来计算每个视场的旋转和平移，同时也计算摄像机的内参数(对所有视场不变)。如我们所讨论的，旋转和平移分别用三个角度和三个偏移量定义，因此对每个视场，有 6 个未知量。这没有问题，因为一个已知平面物体(如棋盘)能够提供 8 个方程，即映射一个正方形到四边形可以用 4 个(x, y)点来描述。每个新的图像帧为计算新的 6 个未知量提供 8 个方程，因此若给定足够图像，我们能够计算出未知内参数的任何值(或者更多)。

【386】

通过下面的简单方程，单应性矩阵 H 把源图像平面上的点集位置与目标图像平面

^① 这里 $W = [R \ t]$ 是一个 3×4 矩阵，前三列包含 R 的 9 个元素，最后一列由拥有三个元素的向量 t 组成。

(通常为成像仪平面)上的点集位置联系起来：

$$p_{\text{dst}} = Hp_{\text{src}}, \quad p_{\text{src}} = H^{-1}p_{\text{dst}}$$

$$p_{\text{dst}} = \begin{bmatrix} x_{\text{dst}} \\ y_{\text{dst}} \\ 1 \end{bmatrix}, \quad p_{\text{src}} = \begin{bmatrix} x_{\text{src}} \\ y_{\text{src}} \\ 1 \end{bmatrix}$$

注意到，我们可以在不知道摄像机内参数的情况下计算 H 。事实上，OpenCV 正是利用从多个视场计算多个单应性矩阵的方法来求解摄像机内参数，如下文所示。

OpenCV 提供了一个方便的函数 `cvFindHomography()`，以对应点序列作为输入，返回最佳描述这些对应点的单应性矩阵。我们最少需要 4 个点来求解 H ，但是如果有，通常提供更多的点^①(如我们提供的大于 3×3 的棋盘)。使用更多点的好处是尽可能减少噪声和其他不确定因素所带来的干扰。

```
void cvFindHomography(
    const CvMat*    src_points,
    const CvMat*    dst_points,
    CvMat*          homography
);
```

输入数组 `src_points` 和 `dst_points` 即可以是 $N \times 2$ ，也可以是 $N \times 3$ 矩阵。对于前者，点是以像素坐标表示，对于后者，希望是齐次坐标。最后的变量 `homography`，其值为函数所填写的 3×3 矩阵，以保证反向投影误差最小。因为在单应性矩阵中只有 8 个独立参数，我们选择归一化，使得 $H_{33}=1$ 。但通常的方法是对整个单应性矩阵乘以一个尺度比例。

摄像机标定

最后我们为摄像机内参数和畸变参数进行摄像机标定。在本节中，我们首先学习使用 `cvCalibrateCamera2()` 来计算这些值，并且利用这些模型对来自标定摄像机的图像畸变进行矫正。首先讨论需要多少棋盘视场图像来求解内参数和畸变。然后在转到简化此工作的代码之前，概述 OpenCV 是如何求解这个系统的。【387】

① 自然，要保证精确解我们只需要 4 个对应点。如果提供更多，我们的计算结果便是最小二乘误差意义上的最优解。

棋盘角点个数和参数个数

首先回顾我们的未知因素，即标定过程求解究竟需要多少参数？在 OpenCV 里，我们有 4 个内参数(f_x, f_y, c_x, c_y)和 5 个畸变参数——三个径向(k_1, k_2, k_3)和两个切向(p_1, p_2)。内参数直接与棋盘所在空间的 3D 几何相关(即外参数)，而畸变参数则与点集如何畸变的 2D 集合相关。因此我们要分别处理两大类参数。为求解 5 个畸变参数(为了增加鲁棒性自然也可以有更多参数)，需要已知模式的三个角点所产生的 6 组信息(在理论上是所有需要解决的 5 个畸变参数(当然，可以利用更多的参数以达到鲁棒性))。因此，在内参数计算中，可以利用一个棋盘视场，接下来考虑外参数。对外参数，我们需要知道棋盘的位置。对棋盘的 6 个不同视场图像，需要三个旋转参数(ψ, φ, θ)和三个平移参数(T_x, T_y, T_z)，因为在每幅中棋盘是移动的。总而言之，在每个视场中，我们必须计算 4 个内参数和 6 个外参数。

假设有 N 个角点和 K 个棋盘图像(不同位置)，需要多少视场图像和角点才能提供足够的约束来求解所有这些参数？

- K 个棋盘图形提供 $2NK$ 个约束(乘以 2 是因为每个点都由 x 和 y 两个坐标值组成的)。
- 忽略每次的畸变参数，我们有 4 个内参数和 $6K$ 个外参数(因为我们需要在每 K 个视场中的找到棋盘位置的 6 个参数)
- 有解的前提是 $2NK \geq 6K + 4$ 成立(或者等价地， $(N - 3)K \geq 2$)

似乎当 $N=5$ 且 $K=1$ 时成立。且慢，对于我们， K (图像个数)必须大于 1。要求 $K > 1$ 是因为我们使用棋盘来为适合每个视场图像的单应性矩阵标定。如前面所讨论的，一个单应性矩阵可以从 4 组(x, y)坐标对中产生多达 8 个参数。这是因为为了表示平面投影视场的所有目标只需要 4 个点：即一次性在四个方向伸展正方形的边，把它变成任意四边形(见第 6 章的投影图像)。因此无论在一个平面上检测到多少个角点，我们只能得到四个有用的角度信息。鉴于每个棋盘视场，方程只能给我们四个角度信息或者 $(4 - 3)K > 1$ ，即 $K > 1$ 。这意味着一个 3×3 棋盘(只计算内部角点)最少需要两个视场来求解标定问题。考虑到噪声和数值稳定性要求，对大棋盘需要收集更多的图像。实际使用中，为了得到高质量结果，至少需要 10 幅 7×8 或者更大棋盘的图像(而且只在移动棋盘在不同图像中足够大以从视场图像中得到更加丰富的信息)。

【388】

内幕探秘

本小节是为求知欲旺盛的人准备的。如果你只想调用标定函数，完全可以略过不

读。如果想继续跟随我们的脚步，就会想到这个问题：数学是怎么应用于标定的？尽管有很多求解摄像机参数的方法，OpenCV 选择那些能够很好工作于平面物体的方法。OpenCV 中使用的求解焦距和偏移的算法是基于 zhang(张) 的方法 [Zhang00]，但求解畸变参数则是另外的基于 Brown [Brown71] 的方法。

首先，我们假定在求解标定参数时摄像机没有畸变。对每一个棋盘视场，我们得到一个前面描述的单应性矩阵 H 。将 H 转写为列向量形式， $H = [h_1 \ h_2 \ h_3]$ ，每个 h 是 3×1 向量。则通过前面对单应性矩阵的讨论，我们可以设置 H 等于摄像机内参数矩阵 M 乘以前两个旋转矩阵 r_1 和 r_2 与平移向量 t 的组合矩阵，再加上缩放因子 s ，即有：

$$H = [h_1 \ h_2 \ h_3] = sM[r_1 \ r_2 \ t]$$

分解方程，得到：

$$h_1 = sMr_1 \quad \text{或} \quad r_1 = \lambda M^{-1}h_1$$

$$h_2 = sMr_2 \quad \text{或} \quad r_2 = \lambda M^{-1}h_2$$

$$h_3 = sMt \quad \text{或} \quad t = \lambda M^{-1}h_3$$

这里 $\lambda = 1/s$ 。

旋转向量在构造中是相互正交的，将缩放因子提到外面，则有 r_1 和 r_2 相互正交。正交有两个含义：旋转向量的点积为 0，而且向量的长度相等。从点积开始，我们有：

$$r_1^T r_2 = 0$$

对任何向量 a 和 b ，我们有 $(ab)^T = b^T a^T$ ，因此替换 r_1 和 r_2 ，得到第一个约束：

$$h_1^T M^{-T} M^{-1} h_2 = 0$$

其中 A^{-T} 是 $(A^{-1})^T$ 的简写形式，已知旋转向量的长度相等：

$$\|r_1\| = \|r_2\| \quad \text{或} \quad r_1^T r_1 = r_2^T r_2$$

代替 r_1 和 r_2 ，我们得到第二个约束：

$$h_1^T M^{-T} M^{-1} h_1 = h_2^T M^{-T} M^{-1} h_2$$

为了使事情更容易些，设置 $B = M^{-T} M^{-1}$ ，展开有：

【389】

$$B = M^{-T} M^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$

事实上，矩阵 B 有通用形式的封闭解：

$$B = \begin{bmatrix} \frac{1}{f_x^2} & 0 & \frac{-c_x}{f_x^2} \\ 0 & \frac{1}{f_y^2} & \frac{-c_y}{f_y^2} \\ \frac{-c_x}{f_x^2} & \frac{-c_y}{f_y^2} & \frac{c_x^2}{f_x^2} + \frac{c_y^2}{f_y^2} + 1 \end{bmatrix}$$

使用 B 矩阵，两个约束有其通用的形式 $h_i^T B h_j$ 。将矩阵乘开看看每个元素的形式。

因为 B 是对称的，它可以写成 6 个元素向量的点积。重新排列 B 的元素得到一个新的向量 b ，我们有：

$$h_i^T B h_j = v_g^T b = \begin{bmatrix} h_{i1} h_{j1} \\ h_{i1} h_{j2} + h_{i2} h_{j1} \\ h_{i2} h_{j2} \\ h_{i3} h_{j1} + h_{i1} h_{j3} \\ h_{i3} h_{j2} + h_{i2} h_{j3} \\ h_{i3} h_{j3} \end{bmatrix}^T \begin{bmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{bmatrix}$$

使用 v_g^T 的定义，两个约束可以写为：

$$\begin{bmatrix} v_{i2}^T \\ (v_{i1} - v_{22})^T \end{bmatrix} b = 0$$

如果我们同时得到 K 个棋盘图像，堆叠这些方程，有：

$$Vb = 0$$

其中 V 是 $2K \times 6$ 的矩阵。如前所述，如果 $K \geq 2$ ，那么方程有解 $b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$ 。摄像机内参数可以从 B 矩阵的封闭解中直接得到：

$$\begin{aligned} f_x &= \sqrt{\lambda / B_{11}} \\ f_y &= \sqrt{\lambda B_{11} / (B_{11} B_{22} - B_{12}^2)} \\ c_x &= -B_{13} f_x^2 / \lambda \\ c_y &= (B_{12} B_{13} - B_{11} B_{23}) / (B_{11} B_{22} - B_{12}^2) \end{aligned}$$

其中：

$$\lambda = B_{33} - [B_{13}^2 + c_y(B_{12}B_{13} - B_{11}B_{23})]/B_{11}$$

外参数(旋转和平移)可以由单应性条件计算得到：

$$r_1 = \lambda M^{-1} h_1$$

$$r_2 = \lambda M^{-1} h_2$$

$$r_3 = r_1 \times r_2$$

$$t = \lambda M^{-1} h_3$$

这里比例比例由正交条件确定 $\lambda = 1/\|M^{-1}h_1\|$ 。

需要小心的是，当我们使用真实数据求解时，将 r 向量放在一起($R = [r_1 \quad r_2 \quad r_3]$)，我们无法得到一个精确的旋转矩阵，即 $R^T R = R R^T = I$ 成立。

要解决这个问题，常用的技巧是使用 R 的奇异值分解(SVD)。如第 3 章所讨论的，SVD 是将矩阵分解为一个对角阵 D 和两个正交阵 U 和 V 的数学方法。这允许我们将 R 转化为 $R = UDV^T$ 。因为 R 本身是正交的，矩阵 D 必须是单位阵 I ，使得 UV^T 。我们可以通过对 R 的奇异值分解，设置 D 为单位阵，用 SVD 方法再重新合成 R ，将 R 强制计算为吻合的旋转矩阵 R' 。

除了所有的这些工作，我们还没有处理透镜畸变。我们是用前面得到摄像机内参数——连同畸变参数都设置为 0——作为我们的初始值，然后来求解大的系统方程。

由于畸变而在图像上得到的感知点的位置是不真实的。如果针孔模型是完美的，令 (x_p, y_p) 为点的位置，令 (x_d, y_d) 为畸变位置，那么有：

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} f_x X''/Z'' + c_x \\ f_y X''/Z'' + c_y \end{bmatrix}$$

通过下面的替换，可以得到没有畸变的标定结果：

【390~391】

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2 p_1 x_d y_d + p_2 (r^2 + 2x_d^2) \\ p_1 (r^2 + 2y_d^2) + 2 p_2 x_d y_d \end{bmatrix}$$

这样在重新估计内外参数后，我们得到的这些大量方程可以找到畸变参数。这对单个函数 cvCalibrateCamera2()^①而言是一个繁重的工作。

① cvCalibrateCamera2() 为第 12 章中的立体标定函数内部所使用。对立体标定，我们需要在同一时间内标定两个摄像机，并且希望将它们通过一个旋转矩阵和平移向量联系到一起。

标定函数

一旦我们有多个图像的角点，就可以调用函数 `cvCalibrateCamera2()`。此函数将要做大量分解工作以提供我们所需的信息。具体来说，我们会得到摄像机内参数矩阵、畸变系数、旋转向量和平移向量。前两个构成摄像机的内参数，后两个构成了物体(如棋盘)位置和方向的摄像机外参数。畸变系数(k_1 , k_2 , p_1 , p_2 和 k_3)^①[181]来自于我们前面遇到的径向和切向畸变方程。它们有助于矫正这些畸变。摄像机内参数矩阵也许是最有用的结果，因为它可以让我们将 3D 坐标转换为 2D 图像坐标。我们也可以使用摄像机矩阵来做相反操作，但这种时候，我们只能得到与图像点对应的三维世界中的一条线。回头再讨论。

让我们检查一下摄像机标定程序本身。

```
void cvCalibrateCamera2(
    CvMat* object_points,
    CvMat* image_points,
    int* point_counts,
    CvSize image_size,
    CvMat* intrinsic_matrix,
    CvMat* distortion_coeffs,
    CvMat* rotation_vectors = NULL,
    CvMat* translation_vectors = NULL,
    int flags = 0
);
```

调用 `cvCalibrateCamera2()` 时，虽然很多变量的本身意思清晰，但我们还是尽量阐述它们，以期能被准确理解。

【392】

第一个参数 `object_points` 是一个 $N \times 3$ 矩阵，包含物体的每 k 个点在每 M 个图像上的物理坐标(即 $N=K \times M$)。这些点位于物体的坐标平面上^②。该变量实际上比看起来的更微妙一些，这在与我们描述物体上的点的方式隐含着坐标系统的结构和物理单位都已经被事先定义。例如，在使用棋盘的场合，我们定义了坐标系使得所有棋盘上的点的 z 坐标值为 0，而 x 和 y 坐标用厘米度量。如果选用英寸，所有参

-
- ① 第三个径向畸变参数 k_3 最后出现是因为它是 OpenCV 后加的一个条件，用于更好地矫正因鱼眼透镜而产生的畸变。我们即刻看到 k_3 的初始值可以设置为 0，然后设置标志位为 `CV_CALIB_FIX_K3`。
 - ② 当然，在不同图像中使用同一个物体很正常，所以 N 个点实际上是同一个物体的 K 个点的 M 次重复列表。

数的计算结果也是用英寸表示。类似地，如果设置所有 x 坐标值（`object_points` 的 x 值）为 0，那么意味着与摄像机相关的棋盘位置将主要在 x 方向上而不是在 z 方向上。正方形定义了一个单位，即如果正方形的边长是 90mm，那么在摄像机世界中，物体和摄像机坐标单位应该是 mm/90。理论上，可以使用棋盘以外的任何其他物体，因此所有物体上的点不必一定在一个平面上。但是使用平面更容易标定摄像机。最简单的方式是，我们简单定义棋盘的每个方块为一个单位，这样棋盘角点的坐标横竖排列都是整数。定义 S_{width} 为棋盘宽度方向的方块个数表示宽度， S_{height} 为棋盘高度方向的方块个数表示高度：

$$(0,0), (0,1), (0,2), \dots, (1,0), (2,0), \dots, (1,1), \dots, (S_{\text{width}}-1, S_{\text{height}}-1)$$

第二个变量是 `image_points`，它是一个 $N \times 2$ 矩阵，包含 `object_points` 所提供的所有点的像素坐标。如果使用棋盘进行标定，那么这个变量简单地由 M 次调用 `cvFindChessboardCorners()` 的返回值构成。但是现在它被以另外一种稍微不同的格式重新排列。

变量 `point_counts` 表示每个图像的点的个数，以 $M \times 1$ 矩阵形式提供。`Image_size` 是以像素衡量的图像尺寸，图像点就是从该图像中提取(例如自己摇晃的棋盘图像)。

下面两个变量 `intrinsic_matrix` 和 `distortion_coeffs` 构成了摄像机的内参数。这些变量既可以是输出参数(标定的主要原因就是填充它们)也可以是输入参数。当被用作输入参数，函数调用时这些矩阵的数值将会影响计算结果，而是否被当作输入参数使用则由后面的标志位参数决定，见随后的讨论。如前面所讨论，内参数矩阵完全定义了理想摄像机模型的摄像机行为，而畸变系数则更多表征摄像机的非理想行为。摄像机矩阵总是 3×3 的，而畸变系数总是 5 个。所以 `distortion_coeffs` 变量是一个指向 5×1 矩阵的指针(记录顺序是 k_1, k_2, p_1, p_2, k_3)。 【393~394】

鉴于前两个变量概括的是摄像机的内参数信息，接下来的两个变量将概况外参数信息。也就是，它们说明与在每幅图像中相对于摄像机来说的标定物体(即棋盘)的位置。物体位置由旋转和平移表征^①。旋转 `rotation_vectors` 由 M 个三元素向量排列为 $M \times 3$ 的矩阵(M 即为图像个数)。需要小心的是，这些向量不是我们前面所讨

-
- ① 行文至此，在运行最优化算法之前，平面的标定物体需要对内参数进行自动初始化。这意味着如果你有一个非平面模板，必须为主点和焦距提供初始的猜测值(见后面讨论的 `CV_CALIB_USE_INTRINSIC_GUESS`)。
 - ② 可以想像，棋盘位置被描述为：(1)在摄像机坐标原点创建一个棋盘；(2)用一定角度绕某个轴旋转棋盘；(3)将旋转的棋盘移到某个特定位置。这些类似 OpenGL 的操作，大家应该不会陌生。

论的 3×3 旋转矩阵，而是代表棋盘围绕摄像机坐标系统下三维空间的坐标轴的旋转，其中每个向量的长度表示逆时针旋转的角度。每个旋转向量可以通过调用 cvRodrigues2() 转换为 3×3 的旋转矩阵，这将在后文讨论。平移 translation_vectors 则简单排列为第二个 $M \times 3$ 矩阵，同样是在摄像机坐标系下。如前所述，摄像机坐标系的单位以棋盘的方块单位为准，即如果棋盘方块边长为 1 英寸，那么单位就是英寸。

通过优化方法找到这些参数是一个富有技巧性的工作。如果设置的初始值位置远离实际解，有时试图一次性求解所有参数会导致结果不精确或者不收敛。因此，通常需要猜测解以得到更好的初始值。因此，我们常常固定某些参数而求解另外一些参数，然后再固定另外的参数求解原始的固定参数，依次往复，最后可以认为所有参数都接近真实解，然后使用所有的参数作为初始值一次性输入。OpenCV 允许你控制所有的标志位。标志位变量是用来做某些细微的控制使得标定能够完成的更好。如有必要，下面的值可以综合使用逻辑运算“或”。

- **CV_CALIB_USE_INTRINSIC_GUESS** cvCalibrateCamera2() 计算内参数矩阵的时候通常不需要额外信息。具体说来，参数 c_x 和 c_y (图像中心)的初始值可以直接从变量 image_size 中得到。如果设置该变量，那么 intrinsic_matrix 假设包含正确值，并被用作初始猜测，为 cvCalibrateCamera2() 做优化时所用。【394】
- **CV_CALIB_FIX_PRINCIPAL_POINT** 这个标志既可以与 CV_CALIB_USE_INTRINSIC_GUESS 一起使用，也可以单独使用。如果单独使用，则设置图像中心为主点。如果一起使用，则设置主点位置为 intrinsic_matrix 提供的初始值。
- **CV_CALIB_FIX_ASPECT_RATIO** 如果设置该标志，那么在调用标定程序时，优化过程只同时改变 f_x 和 f_y ，而固定 intrinsic_matrix 的其他值(如果 CV_CALIB_USE_INTRINSIC_GUESS 也没有被设置，则 intrinsic_matrix 中的 f_x 和 f_y 可以为任何值，但比例相关。)
- **CV_CALIB_FIX_FOCAL_LENGTH** 该标志在优化时候，直接使用 intrinsic_matrix 中传递过来的 f_x 和 f_y 值。
- **CV_CALIB_FIX_K1, CV_CALIB_FIX_K2 and CV_CALIB_FIX_K3** 固定径向畸变 k_1, k_2 和 k_3 。径向畸变参数可以通过组合这些标志设置为任意值。一般地，最后一个参数应设置为 0，除非使用鱼眼透镜。
- **CV_CALIB_ZERO_TANGENT_DIST** 该标志在标定高级摄像机的时候比较重要，因为精确制作导致很小的切向畸变。试图将参数拟合 0 会导致噪声干扰

和数值不稳定。通过设置该标志可以切向畸变参数 p_1 和 p_2 的拟合，即设置两个参数为 0。

只计算外参数

某些情况下我们已经得到了摄像机的内参数，因此只需要计算观察物体的位置。这种场景与一般的摄像机标定明显不同，但是也是一项值得完成的有用工作。

```
void cvFindExtrinsicCameraParams2(
    const CvMat* object_points,
    const CvMat* image_points,
    const CvMat* intrinsic_matrix,
    const CvMat* distortion_coeffs,
    CvMat* rotation_vector,
    CvMat* translation_vector
);
```

`cvFindExtrinsicCameraParams2()` 的变量与 `cvCalibrateCamera2()` 的相应变量相同，只是内参数矩阵和畸变系数是直接提供而非计算出来的。旋转输出的格式是 1×3 或 3×1 旋转向量，表示棋盘或者点绕坐标轴的旋转，而向量长度则表示逆时针的旋转角度。这个旋转向量可以通过 `cvRodrigues2()` 转换为 3×3 矩阵，如前所讨论的。平移向量是摄像机坐标系中棋盘原点的偏移量。 【395~396】

矫正

如我们已经提过的，标定摄像机通常是想做两件事，一个是矫正畸变效应，另一个是根据获得的图像重构三维场景。在深入第 12 章讨论第二个复杂任务之前，我们先花点时间考虑第一个任务。

OpenCV 提供一个直接使用的校正算法，即输入原始图像和由函数 `cvCalibrateCamera2()` 得到的畸变系数，生成矫正后的图像(见图 11-12)。我们既可以一次性通过函数 `cvUndistort2()` 使用该算法完成所有事项，也可以通过一对函数 `cvInitUndistortMap()` 和 `cvRemap()` 来更有效率的处理此事，这通常适合视频或者从同一摄像机中得到多个图像的应用^①。

① 我们花点功夫清晰区分一下“矫正”(undistortion)与“校正”(rectification)的关系，“矫正”是在数学上去掉透镜畸变，而“校正”是数学上将图像排列整齐。

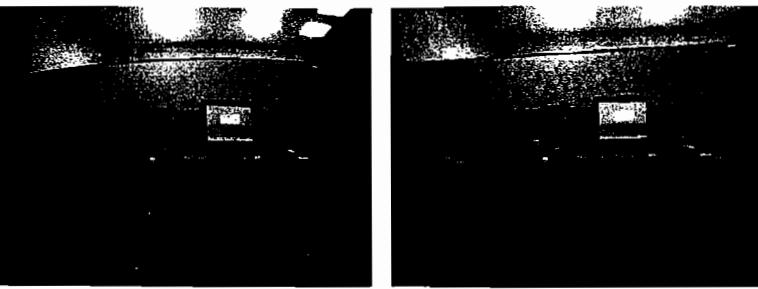


图 11-12：矫正前(左)和矫正后(右)的摄像机图像

基本方法是先计算畸变映射，再矫正图像。函数 `cvInitUndistortMap()` 用于计算畸变映射，而函数 `cvRemap()` 表示在任意图像应用该映射^①。函数 `cvUndistort2()` 是在一次调用中先后完成两个步骤。但是计算畸变映射是一个耗时的操作，所以，当畸变映射不变的时候随时调用函数 `cvUndistort2()` 是一种不聪明的做法。最后，如果有一系列的 2D 点，我们可以调用函数 `cvUndistortPoints()` 从原始坐标变换到矫正后的坐标。

【396】

```
// Undistort images
void cvInitUndistortMap(
    const CvMat*      intrinsic_matrix,
    const CvMat*      distortion_coeffs,
    CvArr*           mapx,
    CvArr*           mapy
);
void cvUndistort2(
    const CvArr*      src,
    CvArr*           dst,
    const cvMat*      intrinsic_matrix,
    const cvMat*      distortion_coeffs
);
// Undistort a list of 2D points only
void cvUndistortPoints(
    const CvMat*      _src,
    CvMat*            dst,
    const CvMat*      intrinsic_matrix,
    const CvMat*      distortion_coeffs,
```

① 我们最先在图像变换(第 6 章)的上下文中遇到 `cvRemap()`。

```
const CvMat* R = 0,  
const CvMat* Mr = 0;  
);
```

函数 `cvInitUndistortMap()` 计算畸变映射，该映射将图像中的每个点与其映射位置关联。前两个变量是摄像机内参数矩阵和畸变系数，全部是来自函数 `cvCalibrateCamera2()` 的期望形式。生成的畸变映射由两个单独的 32 位单通道矩阵所表示：第一个给出点被映射的 x 值，第二个给出 y 值。你可能会疑惑，为何不直接用一个双通道数组代替。因为来自 `cvInitUndistortMap()` 的结果可以直接传递给函数 `cvRemap()`。

函数 `cvUndistort2()` 一次完成所有事情。它输入初始量(畸变图像)以及摄像机内参数矩阵和畸变系数，输出同样尺寸的矫正图像。如前所述，如果你有一系列来自原始图像的 2 维点坐标并且计算相应的矫正点坐标，那么使用 `cvUndistortPoints()`。它有两个额外的参数与使用在第 12 章讨论的立体校正中相关。这些参数是 R ，两个摄像机之间的旋转矩阵，和 M ，矫正后摄像机的内参数矩阵(在第 12 章中仅仅当有两个摄像机时被使用)。矫正后的摄像机矩阵 M 可以是 3×3 矩阵或者 3×4 矩阵，即分别是从 `cvStereoRectify()` 返回的摄像机矩阵 P_1 和 P_2 (有关左或右摄像机，参见第 12 章)的数值中得到的前 3 列或者整个 4 列。这些参数的默认值是 `NULL`，表示函数被解释为单位矩阵。

一次完成标定

好了，现在是时候用例子来把所有东西放在一起了。我们通过程序来完成如下功能：它先寻找用户指定维数的棋盘，然后捕捉到用户需要的许多完整图像(即能找到棋盘的所有角点)，计算摄像机内参数和畸变参数，最后进入显示模式，以显示矫正后的摄像机图像，见例 11-1。当使用这个算法，在连续的捕获之间想要充分的改变棋盘的视角，否则求解标定参数的点集会是病态矩阵(非满秩)，其结果是要么得到一个坏解，要么干脆无解。

【397~398】

例 11-1：读入棋盘的宽度和高度，读入收集到的不同场景图像，然后标定摄像机

```
// calib.cpp  
// Calling convention:  
// calib board_w board_h number_of_views  
//  
// Hit 'p' to pause/unpause, ESC to quit  
//
```

```

#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <stdlib.h>

int n_boards = 0; //Will be set by input list
const int board_dt = 20; //Wait 20 frames per chessboard view
int board_w;
int board_h;

int main(int argc, char* argv[]) {

    if(argc != 4){
        printf("ERROR: Wrong number of input parameters\n");
        return -1;
    }
    board_w = atoi(argv[1]);
    board_h = atoi(argv[2]);
    n_boards = atoi(argv[3]);
    int board_n = board_w * board_h;
    CvSize board_sz = cvSize( board_w, board_h );
    CvCapture* capture = cvCreateCameraCapture( 0 );
    assert( capture );

    cvNamedWindow( "Calibration" );
    //ALLOCATE STORAGE
    CvMat* image_points      = cvCreateMat(n_boards*board_n,2,
                                             CV_32FC1);
    CvMat* object_points     = cvCreateMat(n_boards*board_n,3,
                                             CV_32FC1);
    CvMat* point_counts      = cvCreateMat(n_boards,1,CV_32SC1);
    CvMat* intrinsic_matrix  = cvCreateMat(3,3,CV_32FC1);
    CvMat* distortion_coeffs = cvCreateMat(5,1,CV_32FC1);

    CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
    int corner_count;
    int successes = 0;
    int step, frame = 0;
    IplImage *image = cvQueryFrame( capture );
    IplImage *gray_image = cvCreateImage(cvGetSize(image),8,1);
                                //subpixel

```

```

// CAPTURE CORNER VIEWS LOOP UNTIL WE'VE GOT n_boards
// SUCCESSFUL CAPTURES (ALL CORNERS ON THE BOARD ARE FOUND)
//
while(successes < n_boards) {
    //Skip every board_dt frames to allow user to move chessboard
    if(frame++ % board_dt == 0) {
        //Find chessboard corners:
        int found = cvFindChessboardCorners(
            image, board_sz, corners, &corner_count,
            CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
    }

    //Get Subpixel accuracy on those corners
    cvCvtColor(image, gray_image, CV_BGR2GRAY);
    cvFindCornerSubPix(gray_image, corners, corner_count,
        cvSize(11,11),cvSize(-1,-1), cvTermCriteria(
        CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));

    //Draw it
    cvDrawChessboardCorners(image, board_sz, corners,
        corner_count, found);
    cvShowImage( "Calibration", image );

    // If we got a good board, add it to our data
    if( corner_count == board_n ) {
        step = successes*board_n;
        for( int i=step, j=0; j<board_n; ++i,++j ) {
            CV_MAT_ELEM(*image_points, float,i,0) = corners[j].x;
            CV_MAT_ELEM(*image_points, float,i,1) = corners[j].y;
            CV_MAT_ELEM(*object_points, float,i,0) = j/board_w;
            CV_MAT_ELEM(*object_points, float,i,1) = j%board_w;
            CV_MAT_ELEM(*object_points, float,i,2) = 0.0f;
        }
        CV_MAT_ELEM(*point_counts, int,successes,0) = board_n;
        successes++;
    }
} //end skip board_dt between chessboard capture

//Handle pause/unpause and ESC

```

```

int c = cvWaitKey(15);
if(c == 'p'){
    c = 0;
    while(c != 'p' && c != 27){
        c = cvWaitKey(250);
    }
}
if(c == 27)
    return 0;
image = cvQueryFrame( capture ); //Get next image
} //END COLLECTION WHILE LOOP.

//ALLOCATE MATRICES ACCORDING TO HOW MANY CHESSBOARDS FOUND
CvMat* object_points2 = cvCreateMat(successes*board_n,3,CV_32FC1);
CvMat* image_points2 = cvCreateMat(successes*board_n,2,CV_32FC1);
CvMat* point_counts2 = cvCreateMat(successes,1,CV_32SC1);
//TRANSFER THE POINTS INTO THE CORRECT SIZE MATRICES
//Below, we write out the details in the next two loops. We could
//instead have written:
//image_points->rows = object_points->rows = \
//successes*board_n; point_counts->rows = successes;
//
for(int i = 0; i<successes*board_n; ++i) {
    CV_MAT_ELEM( *image_points2, float, i, 0 ) =
        CV_MAT_ELEM( *image_points, float, i, 0 );
    CV_MAT_ELEM( *image_points2, float, i, 1 ) =
        CV_MAT_ELEM( *image_points, float, i, 1 );
    CV_MAT_ELEM( *object_points2, float, i, 0 ) =
        CV_MAT_ELEM( *object_points, float, i, 0 );
    CV_MAT_ELEM( *object_points2, float, i, 1 ) =
        CV_MAT_ELEM( *object_points, float, i, 1 );
    CV_MAT_ELEM( *object_points2, float, i, 2 ) =
        CV_MAT_ELEM( *object_points, float, i, 2 );
}
for(int i=0; i<successes; ++i){ //These are all the same number
    CV_MAT_ELEM( *point_counts2, int, i, 0 ) =CV_MAT_ELEM(
        *point_counts, int, i, 0 );
}
cvReleaseMat(&object_points);
cvReleaseMat(&image_points);
cvReleaseMat(&point_counts);

```

```

// At this point we have all of the chessboard corners we need.
// Initialize the intrinsic matrix such that the two focal
// lengths have a ratio of 1.0
//
CV_MAT_ELEM( *intrinsic_matrix, float, 0, 0 ) = 1.0f;
CV_MAT_ELEM( *intrinsic_matrix, float, 1, 1 ) = 1.0f;

//CALIBRATE THE CAMERA!
cvCalibrateCamera2(
    object_points2, image_points2,
    point_counts2, cvGetSize( image ),
    intrinsic_matrix, distortion_coeffs,
    NULL, NULL, 0 //CV_CALIB_FIX_ASPECT_RATIO
);

// SAVE THE INTRINSICS AND DISTORTIONS
cvSave("Intrinsics.xml",intrinsic_matrix);
cvSave("Distortion.xml",distortion_coeffs);
// EXAMPLE OF LOADING THESE MATRICES BACK IN:
CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics.xml");
CvMat *distortion = (CvMat*)cvLoad("Distortion.xml");

// Build the undistort map that we will use for all
// subsequent frames.
//
IplImage* mapx = cvCreateImage( cvGetSize(image), IPL_DEPTH_32F,
1 );
IplImage* mapy = cvCreateImage( cvGetSize(image), IPL_DEPTH_32F,
1 );
cvInitUndistortMap(
    intrinsic,
    distortion,
    mapx,
    mapy
);
// Just run the camera to the screen, now showing the raw and
// the undistorted image.
//
cvNamedWindow( "Undistort" );
while(image) {

```

```

IplImage *t = cvCloneImage(image);
cvShowImage( "Calibration", image ); // Show raw image
cvRemap( t, image, mapx, mapy ); // Undistort image
cvReleaseImage(&t);
cvShowImage("Undistort", image); // Show corrected image

//Handle pause/unpause and ESC
int c = cvWaitKey(15);
if(c == 'p') {
    c = 0;
    while(c != 'p' && c != 27) {
        c = cvWaitKey(250);
    }
}
if(c == 27)
    break;
image = cvQueryFrame( capture );
}
return 0;
}

```

【398 ~ 401】

罗德里格斯变换

当处理三维空间的时候，常常需要用 3×3 矩阵表征空间旋转。这种表示方法通常是最方便的，因为一个向量乘以该矩阵等价于该向量某种方式的旋转。不便之处是它不能直观显示 3×3 矩阵的旋转含义。另外一个容易可视化的表示方式^①[187]是用向量形式表示旋转，而该旋转每次用单个角度来操作。这种情况下，最标准的方式是仅用一个向量来说明绕坐标轴的旋转，向量长度表示绕轴逆时针旋转的角度。这个很容易实现，由于方向可以用任意长度的向量表示，因此我们选择向量的长度表示旋转角度。两种表述方式的关系是矩阵和向量可以用罗德里格斯变换关联^②。设 r 为三维向量 $r=[r_x \ r_y \ r_z]$ ，这个向量含蓄地定义 θ ，旋转量用 r 的长度表示。我们能够将这种以坐标轴-标量形式表示的旋转转换为一个旋转矩阵 R ：

① “简单”的表示方式对人不合适。三维空间的旋转只有三个变量。对数值优化程序，处理用罗德里格斯表示的三个元素比用 3×3 旋转矩阵的 9 个元素更有效率。

② 罗德里格斯是十九世纪法国数学家。

$$R = \cos(\theta) \cdot I + (1 - \cos(\theta)) \cdot rr^T + \sin(\theta) \cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ r_y & r_x & 0 \end{bmatrix}$$

我们也能反向从坐标轴表现形式得到旋转矩阵：

$$\sin(\theta) \cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ r_y & r_x & 0 \end{bmatrix} = \frac{(R - R^T)}{2}$$

因此我们发现一种表示形式(矩阵表示)更方便地计算，而另一种表现形式(罗德里格斯表示)更易于理解。OpenCV 为我们提供了相互转换的函数：

```
void cvRodrigues2(
    const CvMat* src,
    CvMat* dst,
    CvMat* jacobian = NULL
);
```

假定我们有向量 r 和对应的旋转矩阵 R ，设置 src 为 3×1 向量 r ， dst 为 3×3 旋转矩阵 R 。相反，我们可以设置 src 为 3×3 旋转矩阵 R ，以及 dst 为 3×1 向量 r 。不论在哪种情况下，函数 `cvRodrigues2()` 都会正确执行。最后的参数是可选的。如果 $jacobian$ 不为 `NULL`，那么它应该是一个 3×9 或 9×3 矩阵的指针，其元素是对应输入数组元素的输出数组元素的偏微分。 $jacobian$ 输出主要用于函数 `cvFindExtrinsicCameraParameters2()` 和函数 `cvCalibrateCamera2()` 的内部优化算法。函数中 $jacobian$ 的应用通常会局限于仅仅将 `cvFindExtrinsicCameraParameters2()` 和 `cvCalibrateCamera2()` 的输出从罗德里格斯的 1×3 或 3×1 坐标轴角度向量格式转换为旋转矩阵。如果是这样，请将 $jacobian$ 设置为 `NULL`。

练习

- 根据图 11-2，利用摄像机中心位置偏移的相似三角形推导方程得出方程 $x = f_x \cdot (X/Z) + c_x$ 和 $y = f_y \cdot (Y/Z) + c_y$
- 真实中心位置(c_x, c_y)的估计错误是否会影响其他参数的估计，如焦距的估计？
提示：参见方程 $q = MQ$ 。
- 分别根据以下条件绘制一个正方形的图像：

- a. 在径向畸变下；
 - b. 在切向畸变下；
 - c. 在两种畸变下。
4. 参考图 11-13，对投影变换，对以下问题进行解释。
- a. 无穷远的线来自何处？
 - b. 为何目标平面的平行线收敛到图像平面的一个点？
 - c. 假定物体平面和图像平面相互垂直。在物体平面上，从一点 p_1 开始，从图像平面开始，远离其 10 个单位到 p_2 ，在图像平面上的相应移动距离是多少？

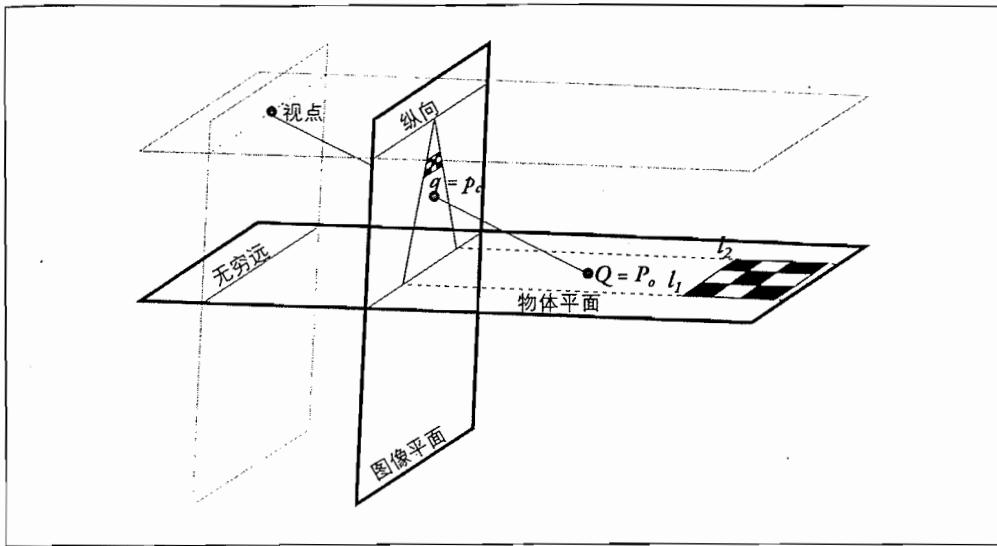


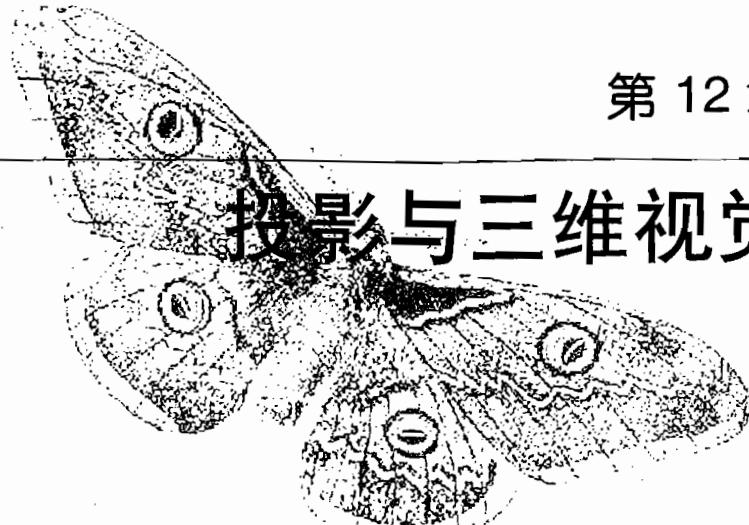
图 11-13：显示物体平面与图像平面相交以及投影中心视场表示的单应性示意图

5. 图 11-3 显示向外凸出的“筒形畸变”效应，它在图 11-12 中则更为明显。一些透镜是否能产生向内弯曲的效果？如何能做到？
6. 使用一个便宜的网络摄像机或者手机，创建有径向和切向畸变的同心正方形或棋盘图像。
7. 在练习 6 中标定摄像机。显示矫正前后的图像。
8. 通过收集棋盘的多幅图像以及对所有图像进行恰当的标定，做数值稳定性和噪声实验。然后，当减少棋盘图像数量时，看看校准参数是怎样变化的？将结果

用图表的形式表示出来：摄像机参数作为棋盘图像数量的一个函数。

9. 参考练习 8，当使用 10 幅大小为 3×5 , 4×6 和 5×7 的棋盘时，标定参数怎样变化？将结果用图表的形式表示出来。
10. 高端摄像机显然有可以物理矫正图像畸变的透镜系统。如果你强行对该摄像机使用多项矫正模型会发生什么现象？
提示：此条件被称为过拟合。
11. 三维游戏杆策略。使用视频，摇晃棋盘并使用 `cvFindExtrinsicCameraParams2()` 作为三维游戏杆，来标定一个摄像机。记住 `cvFindExtrinsicCameraParams2()` 输出 3×1 或 1×3 的旋转向量和三维平移向量，其中旋转向量长度表示沿着 3D 变换向量的方向而逆时针的旋转角度。
 - a. 输出棋盘的坐标轴和当移动棋盘时真实世界的旋转角度。处理棋盘不在视场中的情况。
 - b. 使用函数 `cvRodrigues2()` 将 `cvFindExtrinsicCameraParams2()` 的输出转换为 3×3 的旋转矩阵和平移向量。并利用它动态模拟一个飞机被实时渲染到图像上的简单三维棒图，如同在视频摄像机的视场中移动棋盘一样。

第 12 章



投影与三维视觉

本章我们将转入到三维视觉部分，依次介绍投影和多摄像机深度感知的相关内容。为达此目的我们将继续沿用第 11 章的一些原理：摄像机内参数矩阵 M 、畸变参数、旋转矩阵 R 、平移向量 T 以及单应性矩阵 H 。

首先讨论使用标定后的摄像机到三维世界的投影，并回顾仿射和投影变换(第一次在第 6 章遇见)，然后给出一个获得地平面鸟瞰图的程序实例^①。接下来，我们还将对 POSIT 算法进行讨论，该算法可以让我们从一幅图像中查找获得已知三维物体的三维姿态(位置和旋转角度)。

然后我们转到多幅图像的三维几何问题。一般情况下，没有可靠的方法可以做到不依赖多幅图像就可以进行标定或提取 3D 信息。利用多幅图像重建三维场景的最常见情形就是立体视觉。在立体视觉中，同时在不同位置上拍摄两幅图像(或者更多)中的特征，然后对图像中的相应特征进行匹配，分析其中的差异，从而获得深度信息。另一个情形是从运动中得到结构。这种情况下，我们可能只用一个摄像机，但是要在不同时间从不同的地方拍摄多幅图像。对前者而言，我们主要对视差效应(三角剖分)感兴趣，并作为计算距离的一种方法；而后者，则是通过计算基础矩阵(将两个不同场景联系到一起)来获得场景理解的数据源。下面我们开始介绍投影的相关内容。

投影

一旦对摄像机完成了标定(见第 11 章)，就有能将现实物理世界中的点无歧义地投

① 这是在机器人技术以及其他很多视觉应用中常见的问题。

影到图像上。这意味着给定对于摄像机三维物理坐标框架下的位置，我们可以计算该三维点在成像仪中的坐标，即像素坐标。在 OpenCV 中，这个转换过程通过函数 cvProjectPoints2() 来实现的。

```
void cvProjectPoints2(
    const CvMat* object_points,
    const CvMat* rotation_vector,
    const CvMat* translation_vector,
    const CvMat* intrinsic_matrix,
    const CvMat* distortion_coeffs,
    CvMat* image_points,
    CvMat* dpdrot = NULL,
    CvMat* dpdt = NULL,
    CvMat* dpdf = NULL,
    CvMat* dpdc = NULL,
    CvMat* dpddist = NULL,
    double aspectRatio = 0
);
```

仅仅从参数个数来看，这个函数可能有一些吓人，但实际上，它是一个使用简单的函数。函数 cvProjectPoints2() 被设计用来适应这样的场景(非常普通)，即欲投影的点位于某些刚体上。这种情况下，就自然地不是用摄像机坐标系统表示这些点的位置，而是用以物体本体为中心的坐标系来表示这些点的位置。然后用旋转和平移定义物体坐标系和摄像机坐标系之间的关系。事实上，函数 cvProjectPoints2() 在函数 cvCalibrateCamera2() 的内部被调用，这也是函数 cvCalibrate-Camera2() 组织自身内部操作的方式。所有可选参数都是被函数 cvCalibrate-Camera2() 使用，但熟练的用户可以根据自己的需求来方便地使用这些参数。

第一个参数 object_points 是需要投影的点序列，它是一个包含点位置的 $N \times 3$ 矩阵。可以给这些点赋以物体自身的坐标系，然后提供一个 3×1 的旋转矩阵 rotation_vector^① 和 translation_vector 来建立两个坐标系的联系。如果在特定的环境下，直接使用摄像机坐标系更方便，可以只设定该坐标系下的 object_points，并把 rotation_vector 和 translation_vector 内的元素置为 0^②。

① 旋转向量通常以弧度表示。

② 记住，这个旋转向量是按轴旋转来表示，所以如果设置为全 0，则意味着它的长度为 0，故无旋转。

参数 `intrinsic_matrix` 和 `distortion_coeffs` 与来自第 11 章中讨论的函数 `cvCalibrateCamera2()` 里面的摄像机内参数和畸变系数是一样的。参数 `image_points` 是一个 $N \times 2$ 的矩阵，将被写入计算结果。

最后，长长的可选参数 `dprerot`, `dprdt`, `dprdf`, `dprdc` 和 `dprddist` 都是偏导数的雅可比矩阵，这些矩阵将图像点和每个不同的输入参数联系起来。具体来说，`dprerot` 是与旋转向量部分相关的图像点的偏导数，为 $N \times 3$ 矩阵；`dprdt` 是关于与平移向量部分相关的图像点的偏导数，为 $N \times 3$ 矩阵；`dprdf` 是关于 f_x 和 f_y 的图像点的偏导数，为 $N \times 2$ 矩阵；`dprdc` 是关于 c_x 和 c_y 的图像点的偏导数，为 $N \times 2$ 矩阵；`dprddist` 是关于畸变系数的图像点的偏导数，为 $N \times 4$ 矩阵。大多数情况下，这些参数被默认设置为 `NULL`，不参与计算。最后一个参数 `aspectRatio` 也是可选的，在函数 `cvCalibrateCamera2()` 或者函数 `cvStereoCalibrate()` 中，当方向比固定时，它被用来导数运算。如果 `aspectRatio` 不为 0，导数 `dprdf` 被调整。

【406~407】

仿射变换和透视变换

在 OpenCV 中，经常出现和讨论的两个变换就是仿射变换和透视变换(在其他的自己编写的应用软件中也同样如此)。我们第一次是在第 6 章遇到了这些变换。如 OpenCV 所实现的那样，这些程序作用于点序列或整幅图像。它们把图像上的点从一个位置映射到另外一个位置，通常还伴随着亚像素的插值。回忆一下，仿射变换可以将矩形映射为任意平行四边形；而更一般的情形是，透视变换将矩形映射为任意四边形。

透视变换(perspective transformation)与透视投影(perspective projection)关系密切。透视投影是使用中心投影法，沿着一系列最终汇聚到一个被称为投影中心(center of projection)的点的投影线，将三维物理世界中的点投影变换到二维图像平面中。投影变换是一种特定的单应性变换^①，是将同一个三维物体分别投影到两个不同投影平面下的两幅图像联系起来(因此，针对的是与三维物体相交的平面，具有不同投影中心的非退化配置)。

与这些投影变换相关的函数在第 6 章中已经做了详细的论述，为方便起见，我们仅仅在这里做个简单的汇总，见表 12.1。

① 回想第 11 章，这种特殊的单应性变换也称为平面单应性变换。

表 12-1：仿射变换和透视变换函数

函数名称	用途
cvTransform()	点序列的仿射变换
cvWarpAffine()	整个图像的仿射变换
cvGetAffineTransform()	得到仿射变换矩阵参数
cv2DRotationMatrix()	得到仿射变换矩阵参数
cvGetQuadrangleSubPix()	低开销的图像仿射变换
cvPerspectiveTransform()	点序列的透视变换
cvWarpPerspective()	整个图像的透视变换
cvGetPerspectiveTransform()	得到透视变换矩阵参数

【407】

鸟瞰图变换实例

机器人导航的一项常见工作就是将机器人场景的摄像机视图转换到从上到下的“俯视”视图。在图 12-1 中，场景的机器人视图转换成了鸟瞰图，使得视图就可以被随后的由扫描激光测距仪创建的世界场景覆盖。结合我们已学的知识，接下来我们将详细地来说明如何利用标定的摄像机计算这样的一个鸟瞰图。

【408】

为了获得鸟瞰图^①，我们需要从标定程序中得到摄像机内参数和畸变参数矩阵。仅仅为了程序的多选择性，我们选择从硬盘读取文件。在地面上放置一个棋盘，从机器人车上获得它的地平面图像，然后将图像重新映射为鸟瞰图。具体算法流程如下。

1. 读取摄像机的内参数和畸变参数模型。
2. 查找地平面上的已知物体(如本例中的棋盘)，获得最少 4 个亚像素精度上的点。
3. 将找到的点输入到函数 cvGetPerspectiveTransform()中(参见第 6 章)，计算地平面视图的单应矩阵 H 。
4. 使用函数 cvWarpPerspective()(还是参见第 6 章)，设置标志 CV_INTER_LINEAR + CV_WARP_INVERSE_MAP + CV_WARP_FILL_OUTLIERS，获得地平面的前向平行视图(鸟瞰图)。

① 眼视图技术也可以应用于将任意平面(如墙或天花板)透视变换到前向平行视图。



图 12-1：鸟瞰图。安装在机器人轿车里的摄像机对道路场景进行扫描，激光测距仪识别定位出车前面的“道路”区域，并用框线标记(上图)；视觉算法对平坦的“似道路”区域进行分割(中图)；分割后的道路区域转换成鸟瞰图，并和鸟瞰图上的激光图进行图像融合(下图)

例 12-1 给出了鸟瞰图的全部代码。

例 12-1：鸟瞰图

```
//Call:  
// birds-eye board_w board_h instrinics distortion image_file  
// ADJUST VIEW HEIGHT using keys 'u' up, 'd' down. ESC to quit.  
  
int main(int argc, char* argv[]) {  
    if(argc != 6) return -1;  
  
    //INPUT PARAMETERS:  
    //
```

```

int      board_w      = atoi(argv[1]);
int      board_h      = atoi(argv[2]);
int      board_n      = board_w * board_h;
CvSize* board_sz     = cvSize( board_w, board_h );
CvMat*  intrinsic    = (CvMat*)cvLoad(argv[3]);
CvMat*  distortion   = (CvMat*)cvLoad(argv[4]);
IplImage* image       = 0;
IplImage* gray_image = 0;
if( (image = cvLoadImage(argv[5])) == 0 ) {
    printf("Error: Couldn't load %s\n", argv[5]);
    return -1;
}
gray_image = cvCreateImage( cvGetSize(image), 8, 1 );
cvCvtColor(image, gray_image, CV_BGR2GRAY );

//UNDISTORT OUR IMAGE
//
IplImage* mapx = cvCreateImage(cvGetSize(image), IPL_DEPTH_32F, 1 );
IplImage* mapy = cvCreateImage(cvGetSize(image), IPL_DEPTH_32F, 1 );
//This initializes rectification matrices
//
cvInitUndistortMap(
    intrinsic,
    distortion,
    mapx,
    mapy
);
IplImage *t = cvCloneImage(image);

//Rectify our image
//
cvRemap( t, image, mapx, mapy );

//GET THE CHESSBOARD ON THE PLANE
//
cvNamedWindow("Chessboard");
CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
int corner_count = 0;
int found = cvFindChessboardCorners(
    image,
    board_sz,

```

```

corners,
&corner_count,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS
);
if(!found){
printf("Couldn't aquire chessboard on %s, "
"only found %d of %d corners\n",
argv[5],corner_count,board_n
);
return -1;
}
//Get Subpixel accuracy on those corners:
cvFindCornerSubPix(
gray_image,
corners,
corner_count,
cvSize(11,11),
cvSize(-1,-1),
cvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1 )
);

//GET THE IMAGE AND OBJECT POINTS:
// We will choose chessboard object points as (r,c):
// (0,0), (board_w-1,0), (0,board_h-1), (board_w-1,board_h-1).
//
CvPoint2D32f objPts[4], imgPts[4];
objPts[0].x = 0;           objPts[0].y = 0;
objPts[1].x = board_w-1;   objPts[1].y = 0;
objPts[2].x = 0;           objPts[2].y = board_h-1;
objPts[3].x = board_w-1;   objPts[3].y = board_h-1;
imgPts[0] = corners[0];
imgPts[1] = corners[board_w-1];
imgPts[2] = corners[(board_h-1)*board_w];
imgPts[3] = corners[(board_h-1)*board_w + board_w-1];

//DRAW THE POINTS in order: B,G,R,YELLOW
//
cvCircle( image, cvPointFrom32f(imgPts[0]), 9, CV_RGB(0,0,255), 3 );
cvCircle( image, cvPointFrom32f(imgPts[1]), 9, CV_RGB(0,255,0), 3 );
cvCircle( image, cvPointFrom32f(imgPts[2]), 9, CV_RGB(255,0,0), 3 );
cvCircle( image, cvPointFrom32f(imgPts[3]), 9, CV_RGB(255,255,0), 3 );

```

```

//DRAW THE FOUND CHESSBOARD
//
cvDrawChessboardCorners(
    image,
    board_sz,
    corners,
    corner_count,
    found
);
cvShowImage( "Chessboard", image );

//FIND THE HOMOGRAPHY
//
CvMat *H = cvCreateMat( 3, 3, CV_32F);
cvGetPerspectiveTransform( objPts, imgPts, H);

//LET THE USER ADJUST THE Z HEIGHT OF THE VIEW
//
float Z = 25;
int key = 0;
IplImage *birds_image = cvCloneImage(image);
cvNamedWindow("Birds_Eye");

//LOOP TO ALLOW USER TO PLAY WITH HEIGHT:
//
// escape key stops
//
while(key != 27) {
    // Set the height
    //
    CV_MAT_ELEM(*H, float, 2, 2) = Z;

    //COMPUTE THE FRONTAL PARALLEL OR BIRD'S-EYE VIEW:
    // USING HOMOGRAPHY TO REMAP THE VIEW
    //
    cvWarpPerspective(
        image,
        birds_image,
        H,
        CV_INTER_LINEAR | CV_WARP_INVERSE_MAP | CV_WARP_FILL_OUTLIERS

```

```

);
cvShowImage( "Birds_Eye", birds_image );

key = cvWaitKey();
if(key == 'u') Z += 0.5;
if(key == 'd') Z -= 0.5;
}

cvSave("H.xml",H); //We can reuse H for the same camera mounting
return 0;
}

```

一旦知道单应矩阵和高度参数集后，就能去除图像中的棋盘和车，制作出道路的鸟瞰图视频，如何制作鸟瞰图视频留给读者作为练习。图 12-2 是鸟瞰图代码的输入图像(左图)和输出图像(右图)。

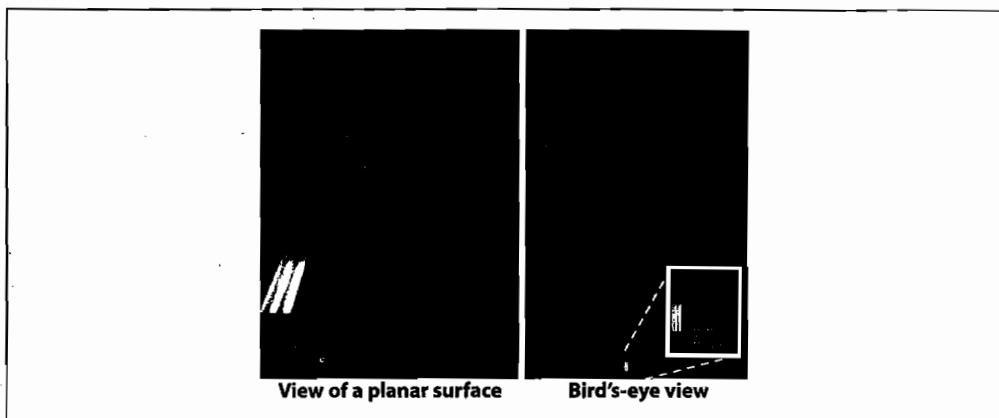


图 12-2：鸟瞰图例子

POSIT：3D 姿态估计

在进入立体视觉之前，我们首先了解一下一个能够估计已知物体三维位置的算法，即 POSIT 算法。POSIT(英文 “Pose from Orthography and Scaling with Iteration”的缩写)是 1992 年首次提出的用于计算 3D 物体(其精确的维数是已知的)的姿态的一种算法(位置 T 和方向 R 由 6 个参数描述[DeMenthon92])。为了计算这个姿态，必须找到物体表面的至少 4 个非共面点在相应二维图像上的位置。算法的第一部分，

由正交投影和尺寸变换提取姿态(POS)，并假设物体上的点具有相同的有效深度^①而且原始模型的大小变化只与其距摄像机的远近比例相关。在这种情况下，基于尺度变换的物体三维姿态的解是闭合形式的。所有物体点都具有相同有效深度的假设意味着物体距离摄像机足够远，使得我们可以忽略了各个点在物体内部的深度差异，这个假设称为“弱透视近似”。

给定已知摄像机的内参数，我们就能求取已知物体的透视缩放比例，从而计算它的近似姿态。这样计算的精度不高，但是如果真实的三维物体位于通过 POS 计算出的近似位置，我们就可以将 4 个观测点投影到预期的位置上，然后我们将这些新的点作为 POS 的输入参数，重新运行一遍 POS 算法。如此的反复迭代，四五次后算法便可收敛到真实的物体姿态，这也是该算法为何被称为“迭代 POS 算法”的原因。记住，上述过程是基于这样的假设，即物体的内部深度远远小于物体与摄像机的距离。如果假设不成立，则算法可能要么不收敛，要么收敛到一个“糟糕的姿态”。OpenCV 所实现的这个算法可以允许我们跟踪四个以上的非共面物体点，从而提高姿态估计的精度。

POSIT 算法在 OpenCV 中有三个相关的函数：一个是对每一个单一的物体的姿态分配数据结构，一个释放该数据结构，一个是 POSIT 算法的执行函数。

```
CvPOSITObject* cvCreatePOSITObject(
    CvPoint3D32f*    points,
    int              point_count
);
void cvReleasePOSITObject(
    CvPOSITObject** posit_object
);
```

函数 `cvCreatePOSITObject()` 的输入为 `points`(三维点集合)和 `point_count`(表示点个数的整数)，并返回分配好了内存的指向 POSIT 对象结构的指针。而函数 `cvReleasePOSITObject()` 是一个指向上述结构的指针，并且释放该数据结构(在处理过程中，设置该指针为 `NULL`)。

```
void cvPOSIT(
    CvPOSITObject* posit_object,
    CvPoint2D32f*  image_points,
```

① 该结构查找一个通过与图像平面平行的穿过物体的参考平面，该平面与图像平面有单一距离值 Z。物体上的三维点首先投影到该参考平面上，然后再利用透视映射投影到图像平面上。这个结果就是比例正交投影，它可以特别容易地将物体大小与深度关联起来。

```
    double          focal_length,  
    CvTermCriteria criteria,  
    float*         rotation_matrix,  
    float*         translation_vector  
);
```

【411~412】

现在，回到 POSIT 函数本身，函数 `cvPOSIT()` 的参数列表其他大部分函数不同，这是因为它使用的是早期版本 OpenCV 中所用的“旧的”参数风格^①。这里，`posit_object` 是指向欲跟踪的 POSIT 对象结构的指针，`image_points` 为图像平面上对应点的位置列表(注意，这些是运行亚像素精度的 32 位浮点数数值)。函数 `cvPOSIT()` 的当前实现是假设方形像素，因此只使用一个值表示参数 `focal_length`，而不是原来使用的 `x` 和 `y` 方向上的两个值。因为函数 `cvPOSIT()` 是个迭代算法，它需要一个迭代终止准则：该准则提示何时结果已经足够好，迭代可以结束。最后两个参数 `rotation_matrix` 和 `translation_vector` 与早期函数的同名参数意义相同，它们是指向浮点型的指针，因此可以通过调用函数 `cvCalibrateCamera2()` 来获得矩阵的数据部分信息。这种情况下，给定矩阵 `M`，便可以使用 `M->data.fl` 作为函数 `cvPOSIT()` 中的参数。

使用 POSIT 时，请记住，增加物体表面上的共面点，并不能使算法执行得更好。若一个点位于其他三个点定义的平面上，那么这样的点对算法不会有任何贡献。实际上，多余的共面点往往会导致算法性能降低。而多余的非共面点，则对算法有好处。图 12-3 给出了 POSIT 算法在玩具飞机上的应用[Tanguay00]，飞机上有四条标记线，用来定义 4 个非共面点。这些点被传入到函数 `cvPOSIT()` 中，输出的 `rotation_matrix` 和 `translation_vector` 就用来控制飞行模拟器。



图 12-3：POSIT 算法的应用。用玩具飞机的四个非共面点控制飞行模拟器

【414】

① 可能已注意到很多函数都是以“2”结尾的。这往往是因为当前版本库中的函数参数与前一版本相比通常发生了改变，因而利用的是较新的参数类型。

立体成像

现在我们正式开始进入立体成像(stereo imaging)^①。所有对眼睛的立体成像能力都很熟悉，但是在计算机系统内我们可以多大程度地模仿这种立体成像能力呢？计算机可以在两个成像仪上寻找对应点来完成立体成像。通过这样的对应点和摄像机之间已知的基线间隔，我们就可以计算这些点的三维位置。虽然对应点的搜寻需要很高的计算成本，但是我们可以根据已有系统的几何结构来尽可能地减小搜索空间。在实际应用中，两台摄像机的立体成像过程包括以下四个步骤。

1. 消除畸变：使用数学方法消除径向和切线方向上的镜头畸变，第 11 章已对此进行了详细描述。这一步输出的是无畸变图像。
2. 摄像机校正：调整摄像机间的角度和距离，输出行对准^②的校正图像。
3. 图像匹配：查找左右摄像机视场中的相同特征^③。这一步输出的是视差图，差值是指左右图像上的相同特征在 x 坐标上的差值 $x_l - x_r$ 。
4. 重投影：当知道了摄像机的相对几何位置后，就可以将视差图通过三角测量的方法转成距离。本步骤输出等深度图。

我们先从最后一步说起，然后分析前三步。

三角测量

假设我们已有一套无畸变、对准、已测量好的完美标准立体实验台，如图 12-4 所示，两台摄像机的像平面精确位于同一平面上，光轴严格平行(光轴是从投影中心 O 朝像主点 c 方向引出的一条射线，也称为主光线^④)，距离一定，焦距相同 $f_l = f_r$ 。

-
- ① 这里我们仅仅给出高层的理解，更详细论述请参考下面的文献：Trucco 和 Verri [Trucco98]，Hartley 与 Zisserman [Hartley06]，Forsyth 与 Ponce [Forsyth03]，以及 Shapiro 与 Stockman [Shapiro02]。这些书中的立体校正部分将给出本章所引用文献的背景知识。
 - ② 我们说的“行对准”是指两图像在同一个平面上，并且图像的每一行是严格对齐的(具有相同的方向和 y 坐标)。
 - ③ 我们每次提到左右摄像机时，也表示可能是竖直方向的上下摄像机，不同的只是 x 方向上的视差变成了 y 方向的。
 - ④ 两条平行主光线是指这两条线在无穷远处相交。

并且假设主点 c_x^{left} 和 c_x^{right} 已经校准，在左右图像上具有相同的像素坐标。注意，不要将主点和图像中心混为一谈，主点是主光线与像平面的交点，该交点在镜头的光轴上。我们在第 11 章提到过，像平面很少与镜头完美地重叠，因此图像中心也几乎不会和像主点重合。

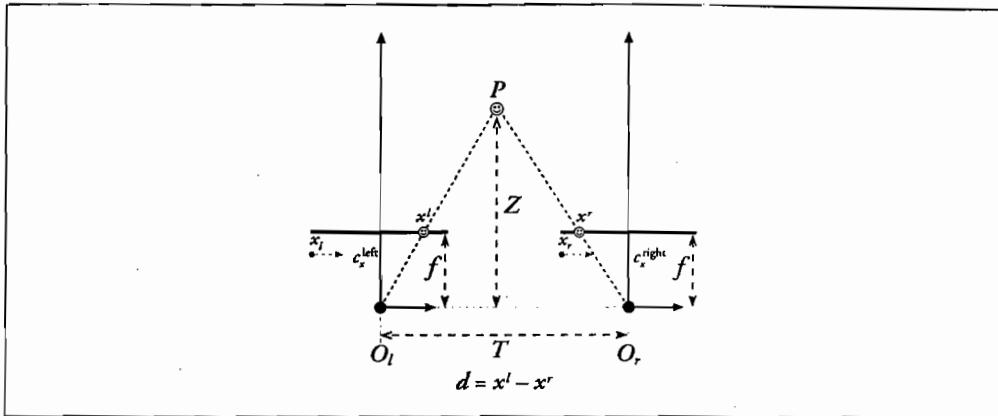


图 12-4：利用无畸变对准的标准立体实验台和已知的特征对应点，深度 Z 可以通过相似三角形计算出来；成像仪的主光线从投影中心 O_1 和 O_2 出发，并经过两个像平面的像主点 c_l 和 c_r

进一步假设两幅图像是行对准的，并且一台摄像机的像素行与另一台完全对准^①，我们称为摄像机前向平行排列。再假设物理世界中的点 P 在左右图像上的成像点为 p_l 和 p_r ，相应的横坐标分别为 x_l 和 x_r 。

在这个简单化的例子中， x_l 和 x_r 分别表示点在左右成像仪上的水平位置，这使得深度与视差成反比关系，视差简单的定义为 $d = x_l - x_r$ 。如图 12-4 所示，利用相似三角形可以很容易推导出 Z 值。由图可知^②：

$$\frac{T - (x' - x')}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x' - x'} \quad \boxed{[415 \sim 417]}$$

- ① 这里提出了好几个假设，但是我们现在只考虑基本假设。请记住，校正的过程(接下来要说的)就是当这些假设在现实中不成立时用数学方法解决问题。类似地，接下来也做了临时的假设来分析匹配问题。
- ② 这个公式说明主光线在无穷远处相交。然而，如同本章后文所示，我们推导出与主点 c_x^{left} 和 c_x^{right} 相关的立体校正。在推导过程中，如果主光线在无穷远处相交，主点则具有相同的坐标，计算深度的公式成立；但是如果主光线在有限距离内相交的话，则主点不会相等，而求深度公式修改为 $Z = f_T / (d - (c_x^{\text{left}} - c_x^{\text{right}}))$ 。

因为深度与视差成反比，二者是明显的非线性关系。当视差接近 0 时，微小的视差变化会导致很大的深度变化；当视差较大时，微小的视差变化几乎不会引起深度多大的改变。结果就是，立体视觉系统仅仅对于物体与摄像机相距较近时具有较高的深度精度，如图 12-5 所示。

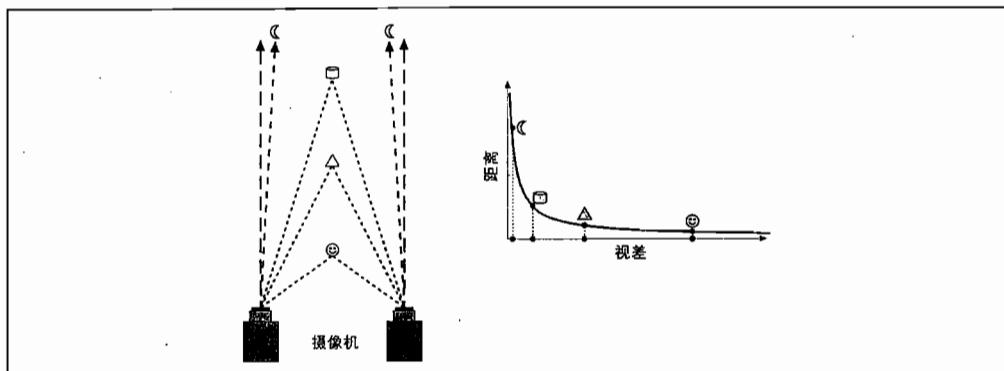


图 12-5：深度与视差成负相关，故精度较高的深度估计被限制于针对附近的物体

在第 11 章对标定的讨论中，我们已经遇到多个坐标系统。图 12-6 显示立体视觉在 OpenCV 中使用到的二维和三维坐标系统。注意，它是一个右手坐标系：你用右手食指指向 X 方向，弯曲中指指向 Y 方向，拇指指向的就是主光线的方向。左右成像仪的像素原点都在图像的左上角，像素坐标分别记为 (x_l, y_l) 和 (x_r, y_r) 。投影中心为 O_l 和 O_r ，主光线与图像平面相交于主点 (c_x, c_y) （非图像中心）。经过数学校正，摄像机就是行对准的（共面且水平对准），且相距为 T ，相同的焦距为 f 。

这样安排可以相对容易地求取距离。现在，我们必须花一些精力去弄明白，怎么样才能让现实世界中摄像机设备映射到理想安排的几何状态。在真实世界中，摄像机几乎不可能会像图 12-4 那样的严格的前向平行对准。不过，我们可以通过数学方法计算投影图和畸变图，从而将左右图像校正成为前向平行对准。在设计立体实验台的时候，最好近似地将摄像机放置成前向对准，并尽量让摄像机水平对准。这种实际的对准可以使得数学变换更易处理。如果摄像机没有尽可能对准，那么数学转换的结果会导致很大的图像畸变，而且也会减小甚至消除结果图像的立体重叠区域^①。要想要一个好的结果，也需要同步的摄像机。如果摄像机不能同时捕获图像，

① 针对想在很近范围内获得高精度的应用则例外；在这种情况下，让摄像机相互微微倾斜，以保证摄像机的主光线能在一个有限的距离内相交。经过数学整理，摄像机向内倾斜的影响就是引入了一个从视差减去的 x 偏移量。这可能会导致负的视差，但我们可以感兴趣的深度范围内获得更好的深度精度。

则一旦场景(包括摄像机本身)内有任意物体在移动，就会出问题。如果没有同步摄像机，就只能使用固定摄像机观察静态的立体场景了。

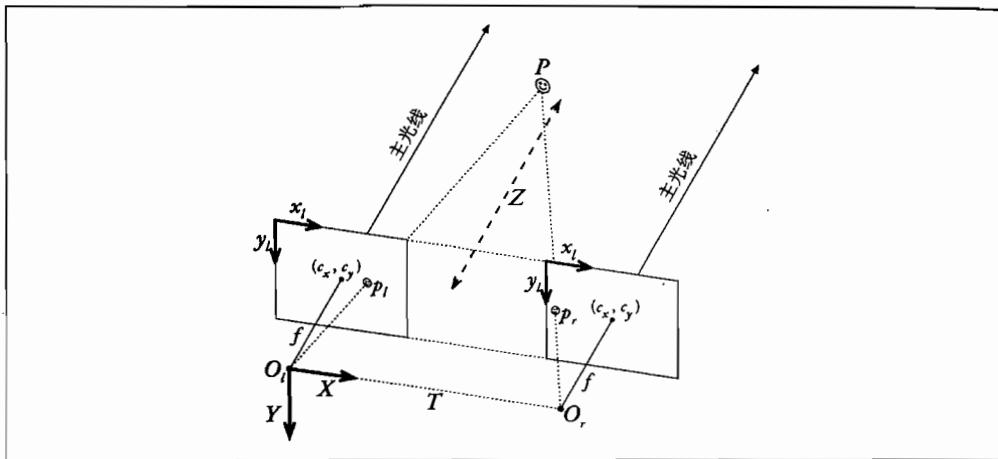


图 12-6：OpenCV 中的未畸变校正摄像机的立体坐标系。像素坐标系以图像的左上角为原点，两个平面行对准；摄像机坐标系以左摄像机的投影中心为原点

图 12-7 描述了两个摄像机间的真实情况和想要实现的数学对准。为了完成数学对准，我们需要知道观测一个场景的两台摄像机间更多的几何关系。一旦知道它们的几何定义和一些可以描述的术语与符号，我们就可以回到对准的问题上。

【417~418】

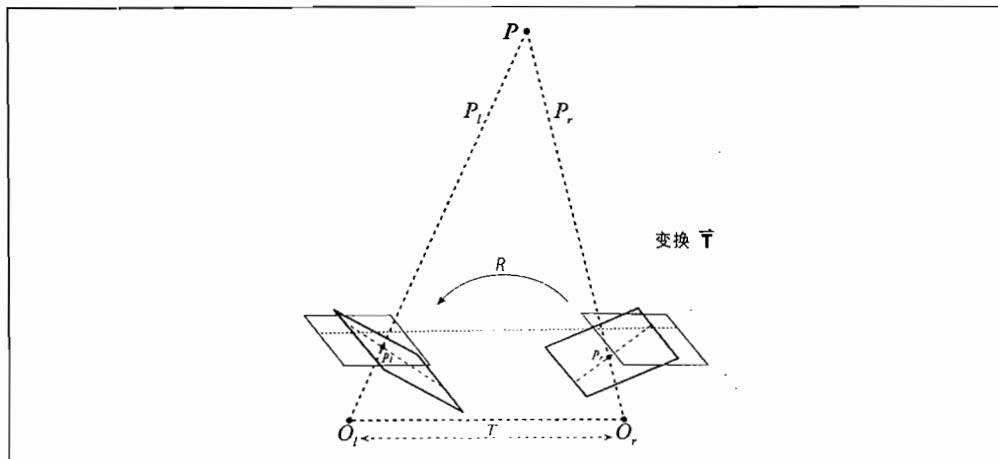


图 12-7：我们的目的是想要将摄像机在数学上对准(而不是物理对准)到同一个观察平面上，从而使得摄像机之间的像素行是严格的互相对准

对极几何

立体成像的基本几何学就是对极几何。从本质上来说，对极几何就是将两个针孔模型(每个摄像机就是一个针孔^①)和一些新的被称为极点(epipole)的兴趣点(见图 12-8)结合起来。在解释这些极点有何用处之前，我们先花点时间来明确定义它们，再了解一些相关的术语。完成这些之后，我们将会对整个对极几何有一个简明的理解，然后会发现可以尽可能地缩小两台立体摄像机上匹配点出现的可能范围。这个额外的发现将对实际的立体实现具有重要的作用。

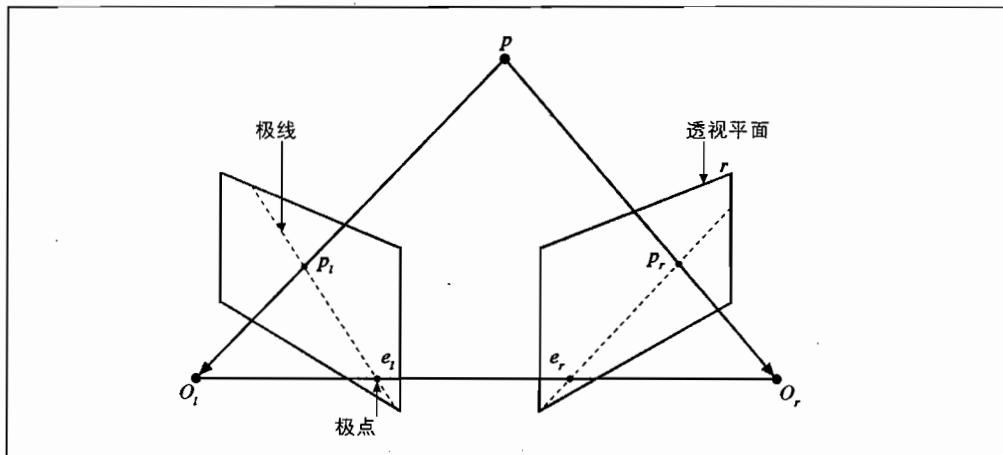


图 12-8：极面是观测点 P 与两个投影中心 O_l 和 O_r 确定的面；极点是投影中心的连线与两个投影面的交点

每台摄像机各自都有一个独立的投影中心，分别为(O_l 和 O_r)以及相应的投影平面 Π_l 和 Π_r 。物理世界中的点 P 在每个投影面上的投影点，记为 p_l 和 p_r 。新感兴趣的点叫极点，像平面 Π_l (或 Π_r)上的一个极点 e_l (或 e_r)被定义成另一台摄像机 O_r (相应地， O_l)的投影中心的成像点。由实际点 P 和两个极点 e_l 和 e_r (或者投影中心 O_r 和 O_l)确定的平面叫极面。线 $plel$ 和 $prer$ (投影点与对应极点之间的连线)称为极线^②。

【419】

-
- ① 由于我们处理的实际上 是真实镜头而不是针孔摄像机，因此左右两幅图像的畸变处理非常重要，详见第 11 章。
 - ② 你可能想到为什么极点以前没有出现过。这是因为当平面是完美平行时，极点为无穷远。

为了理解前面提到过的极点的用处，当我们看到物理世界中的一个点投影到右(或左)图像平面时，这个点实际上会落在沿着由 O_r 指向 p_r (或者由 O_l 指向 p_l)的这条射线的任何位置上，这是因为我们仅仅靠一个摄像机是无法知道与观测点的距离。准确地说，假设点 P 是从右图像上看到，因为这摄像机仅仅看到 p_r (P 在 Π_r 的投影)，实际的点 P 可以是在 p_r 和 O_r 所在直线的任何位置。很明显这条直接包含点 P ，但也同样包含其他的很多点。感兴趣读者的会问，这条线如果被投影到左图像 Π_l 上会是怎么样；实际上，这就是 p_l 和 e_l 确定的那条极线。翻译过来就是，在一个成像仪上看到的所有可能位置的点都是穿过另一台成像仪的极点和对应点的直线图像。

现在，我们总结一下立体摄像机对极几何的一些事实。

- 摄像机视图内的每个 3D 点都包含在极面内，极面与每幅图像相交的直线是极线。
- 给定一幅图像上的一个特征，它在另一幅图像上的匹配视图一定在对应的极线上。这就是“对极约束”。
- 对极约束意味着，一旦我们知道立体试验台的对极几何之后，对两幅图像间匹配特征的二维搜索就转变成了沿着极线的一维搜索。这不仅仅节省了大量的计算，还允许我们排除许多导致虚假匹配的点。
- 次序是保留的。如果两个点 A 和 B 在两幅成像仪上都可见，按顺序水平出现在其中成像仪上，那么在另一个成像仪上也是水平出现^①。 【420~421】

本征矩阵和基础矩阵

你可能想下一步就应该介绍计算这些极线的 OpenCV 函数了。但在介绍这些函数之前，我们实际上还需要两个因素。这两个因素就是本征矩阵 E 和基础矩阵 F ^②。矩阵 E 包含在物理空间中两个摄像机相关的旋转和平移信息(见图 12-9)，矩阵 F 除

-
- ① 由于碰撞和视图叠加区域，自然出现两个摄像机都看不见的点。但是次序却是保留的。假设左边成像仪上的三个点 A 、 B 、 C 从左到右排列，而 B 由于遮挡在右边成像仪上不可见，这时候我们仍然可以在右图像上看到从左到右的点 A 和 C 。
 - ② 接下来的一小部分与数学相关。如果不喜欢数学，可以直接略过不读；至少当某时某刻全部理解这些材料后会增强你的信心。就简单的应用而言，可以仅仅使用 OpenCV 提供的框架而无需下面几页所提供的细节。

了包含 E 的信息外还包含了两个摄像机的内参数^①。由于 F 包含了这些内参数，因此它可以在像素坐标系将两台摄像机关联起来。

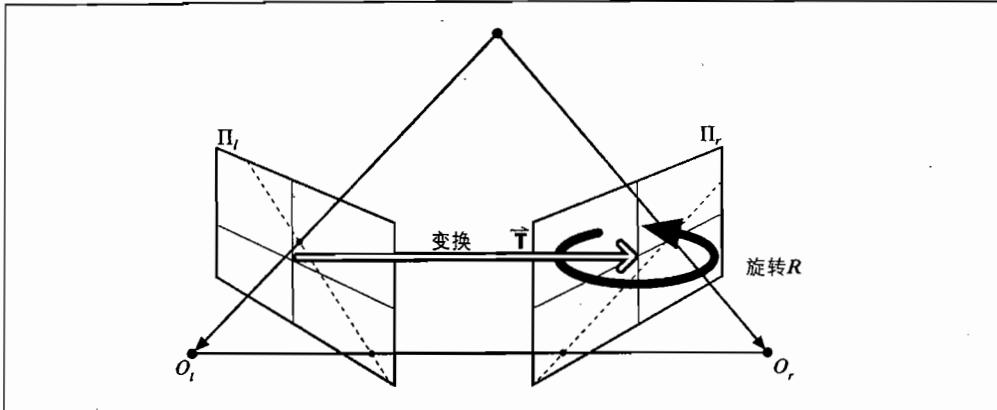


图 12-9：立体成像的几何本质是通过本征矩阵 E 得到的，它包含了关于平移 T 和旋转 R 的所有信息， T 和 R 描述了一台摄像机相对另一台摄像机在全局坐标系中的相对位置

我们再补充说明一下 E 和 F 的区别。本征矩阵 E 是单几何意义上的，与成像仪无关，它将左摄像机观测到的点 P 的物理坐标与右摄像机观测到的相同点的位置关联起来(例如将 p_l 和 p_r 关联)。基础矩阵 F 则是将一台摄像机的像平面上的点在图像坐标(像素)上的坐标和另一台摄像机的像平面上的点关联起来(在这里，我们标记为 q_l 和 q_r)。

【421~422】

本征矩阵数学

我们接下来将引入一些数学内容，这样可以更好地理解这些 OpenCV 函数，以解决立体几何的一些难题。

给定点 P ，我们将要推导点 P 在两个成像仪的观测位置 p_l 和 p_r 之间的关系。该关系将最终转变成本征矩阵的定义。从考虑 p_l 和 p_r 的关系开始， p_l 和 p_r 是在两个摄像机坐标系中观察到的点的物理坐标。利用对极几何，可以将它们联系起来，如已

① 聪明的读者会认识到，矩阵 E 和前一节描述的单应矩阵 M 几乎完全相同。尽管它们都是由相同的信息构建的，但是二者还是有些差异，不能混为一谈。 H 定义的本质是我们所考虑的为摄像机的观察平面，从而把该平面上的点与摄像机平面上的匹配点关联起来。而矩阵 E 没有这个假设，它仅仅是将一幅图像上的点和另外一幅图像的点联系起来。

经所阐述的那样^①。

我们选择一列坐标，左或右都行，开始引入计算。任何一组都没有问题，但是我们将选择左摄像机 O_l 为中心的坐标。在这些坐标里面，观测点的位置是 P_l ，而另一台摄像机的原点为 T 。点 P 在右摄像机的观测坐标(右摄像机坐标系)是 P_r , $P_r = R(P_l - T)$ 。关键的一步是引入极面，如我们所知，它联系所有的相关事务。当然，我们能够用多种方式去描述一个平面，但对当前目的而言，最有用的就是用通过法向量 n 和平面上的所有点 x 表示：

$$(x-a) \cdot n=0$$

回顾向量 P_l 和 T 在极面上，因此，如果有一个向量与 P_l 和 T 都垂直(比如 $P_l \times T$)^②，那就可以用以表示求取平面公式中的 n 。这样，通过点 T 的所有可能点 P_l 以及包含这两个向量的方程表示为^③：

$$(P_l - T)^T (T \times P_l) = 0$$

记住，我们的目的是通过 P_l 和 P_r 的联系来获得 q_l 和 q_r 之间的关系。通过等式 $P_r = R(P_l - T)$ 将点 P_r 绘制到图像上，这样便可方便地将等式重写为 $(P_l - T) = R^{-1}P_r$ ，并代入到 $R_T = R^{-1}$ ，可得：

$$(R^T P_r)^T (T \times P_l) = 0$$

总是可以将叉积写成矩阵相乘的形式，故定义矩阵 S 为：

$$T \times P_l = S P_l \quad \Rightarrow \quad S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}$$

这样就推导出第一个结果，将上式代入到叉积公式中，得到：

$$(P_r)^T R S P_l = 0$$

乘积 RS 就是我们定义的本征矩阵，可以写成下面的简洁形式：

-
- ① 请不要把在投影的像平面上的点 p_l 、 p_r 与和在两台摄像机坐标系下的点 P 的位置 p_l 、 p_r 混为一谈。
 - ② 用两个向量的叉积得到的第三个向量与这两个向量都垂直。叉积的方向可以通过“右手规则”定义：食指指向方向 a ，中指弯曲指向方向 b ，那么 a 和 b 的叉积 $a \times b$ 与 a 和 b 垂直，它的方向就是拇指指向的方向。
 - ③ 这里我们用通过将正交向量转置后矩阵乘法代替了点积(点乘)。

$$(P_r)^T EP_r = 0$$

当然了，我们真正想要的是获得成像仪上的观测点之间的关系，这仅仅是其中的一步。我们可以通过投影方程 $p_i = f_i P_i / Z_i$ 和 $p_r = f_r P_r / Z_r$ 将上式简化，并利用 $Z_i Z_r / f_i f_r$ 分解得到我们的最终结果：

$$P_r^T EP_r = 0$$

首先，看起来，如果已知其他项，我们就完全可以确定 p 项，但是 E 是一个秩亏矩阵^①(一个秩等于 2 的 3×3 矩阵)，因此方程实际上有无穷解。本征矩阵中有五个参数——三个旋转参数、两个平移参数(没有设置缩放)以及两个其他约束。本征矩阵中的两个其他约束是：(1)行列式值等于 0，因为是秩亏矩阵(一个 3×3 、秩为 2 的矩阵)；(2)两个非零奇异值相等，因为 S 是反对称矩阵，而 R 是旋转矩阵。这总共产生七个约束。再次注意， E 不包括摄像机的内参数，因此它联系的是点的物理坐标或者摄像机坐标而不是像素坐标。

【422~423】

基础矩阵数学

矩阵 E 包含两台摄像机相关的所有几何信息，但不包括摄像机本身任何信息。在实际应用中，我们通常只对像素坐标感兴趣。为了探究一幅图像上的像素和它在另一幅图像上的对应极线的联系，必须引入两台摄像机的内参数信息。为此，用 p (像素坐标)来代替 q ，二者通过摄像机内参数矩阵相关联。已知 $q = Mp$ (M 是摄像机内参数矩阵)或等价的， $p = M^{-1}q$ ，因此关于 E 的等式就变成：

$$q_r^T (M_r^{-1}) E M_i^{-1} q_i = 0$$

虽然看起来有点混乱，我们可以通过定义基础矩阵 F 来整理一下：

$$F = (M_r^{-1}) E M_i^{-1}$$

因此有

$$q_r^T F q_i = 0$$

表面上，除了基础 F 操作的是图像像素坐标而 E 操作的是物理坐标之外，基础矩

① 对于类似 E 的 $n \times n$ 方阵，秩亏意味着它的非 0 特征值个数小于 n 。所以秩亏矩阵说明线性方程组没有唯一解。如果秩(非 0 特征值的个数)为 $n-1$ ，方程的所有解会在同一条直线上；如果秩为 $n-2$ ，就可以确定一个面；以此类推。

阵 F 与本征矩阵 E 没什么差别^①。与 E 一样，基础矩阵 F 的秩也为 2。基础矩阵 F 有 7 个参数，其中两个参数表示对极，三个参数表示两个像平面的单应矩阵(通常的 4 个参数中没有比例因素)。

OpenCV 的处理方法

类似地，可以像前一节计算图像单应矩阵的方法，通过已知的匹配点来计算矩阵 F 。这种情况下，甚至不需要单独为摄像机进行标定，因为我们可以直接求解 F ，它其实隐含了两台摄像机的基础矩阵。处理整个过程的这个算法是 `cvFindFundamentalMat()`。

```
int cvFindFundamentalMat(
    const CvMat* points1,
    const CvMat* points2,
    CvMat* fundamental_matrix,
    int method = CV_FM_RANSAC,
    double param1 = 1.0,
    double param2 = 0.99,
    CvMat* status = NULL
);
```

函数 `cvFindFundamentalMat()` 的前两个参数是 $N \times 2$ 或者 $N \times 3$ ^② 的浮点型(单精度或双精度)矩阵，它包含的是收集的 N 个匹配点(也可以是 $N \times 1$ 的 2 或 3 个通道的多通道矩阵)。输出结果是 `fundamental_matrix`，是与输入点相同精度的 3×3 矩阵(个别情况下是 9×3 的矩阵，下面将阐述)。【424~425】

函数的下一个参数是利用对应点计算基础矩阵的方法选项，你可以从四个之中选择一个。对每个数值，对点 `points1` 和 `points2` 所需要的(或允许的)点数有限制，如表 12-2 所示。

-
- ① 请注意，该公式是将本征矩阵和基础矩阵联系在一起。如果有了校正图像和用焦距归一化所有点，这时内参数矩阵 M 就变成了单位矩阵，并且 $F=E$ 。
 - ② 你可能想知道， $N \times 3$ 或者 3 通道矩阵有何用途。这个算法只适用于标定物体测量到的 3D 坐标 (x, y, z) 。三维点最后将被转换成 $(x/z, y/z)$ ，或者你也可以用齐次坐标 $(x, y, 1)$ 的形式输入 3D 坐标，这样的处理方式是一样的。如果输入 $(x, y, 0)$ ，算法将会忽略 0。一般情况下，由于我们只能探测到标定物体上的 2D 点，所以实际上很少会出现 3D 点的情况。

表 12-2: cvFindFundamentalMat()中方法参数的约束条件

方法	点数	约束
CV_FM_7POINT	$N = 7$	7 点算法
CV_FM_8POINT	$N \geq 8$	8 点算法
CV_FM_RANSAC	$N \geq 8$	RANSAC 算法
CV_FM_LMEDS	$N \geq 8$	LMedS 算法

7 点算法只使用 7 个点，其原理是，矩阵 F 的秩必须为 2，这样才可以完全地约束矩阵。这个约束的好处就是，矩阵 F 的秩总是为 2，因此它没有不等于 0 的很小特征值。不足之处在于，约束不是绝对的唯一，所以函数有可能返回三种不同的矩阵（这就是为什么前面说到了要将 fundamental_matrix 创建为 9×3 的矩阵，这是考虑了要顾及所有的三种结果）。8 点算法通过线性方程来求解 F 。如果输入的点数目超过 8，则求所有点的最小二乘解。7 点算法和 8 点算法都存在的一个问题是对异常点非常敏感（即使 8 点算法的输入点数目超过 8 也是如此）。RANSAC 与 LMedS 算法一般被划分为鲁棒算法，这是因为他们有一定的识别和剔除异常点的能力^①。对这两种方法而言，都要求最少输入 8 个点，且多多益善。

下面的两个参数仅为方法 RANSAC 和 LmedS 所使用。第一个参数 param1 是点到极线（像素）的最大距离，超出该值则认为该点是异常点。第二个参数 param2 是期望可信度（0~1 之间），实际上是告诉算法的迭代次数。

最后一个参数 status 为可选，如果使用则应该它是一个 $N \times 1$ 的 CV_8UC1 类型的矩阵，其中 N 与 points1 和 points2 的长度相同。如果矩阵非空，RANSAC 和 LMedS 就会用它来存储哪些是异常点和哪些不是异常点的信息。具体说来，如果为异常点则其元素置为 0，否则置为 1。对其他两种方法而言，如果数组存在，则所有值都置为 1。
【425~426】

cvFindFundamentalMat() 的返回值是一个整数，表示查找出的矩阵个数。它对 7 点算法而言可以是 0、1 和 3，而对其他所有方法只能是 0 或者 1。如果为 0，说明没有矩阵被计算。OpenCV 手册上的例子代码见例 12-2。

① RANSAC 和 LMedS 算法的内部工作原理超出了本书的范围，但是 RANSAC 算法的基本原理是使用点的随机子集来多次地求解问题，然后选出最接近平均或中间的结果。LMedS 算法采用点的子集去估计一个解，然后把剩余的那些与解“相容”的点中加入到子集中，并剔除其他的所谓的“异常点”。更多有关 RANSAC 的内容，请参考 Fischler 和 Bolles 的原始论文，有关最小中值平方，请参考 Rousseeuw [Rousseeuw84]，使用 LMedS 的线性拟合，请参考 Inui, Kaneko 和 Igarashi [Inui03]。

例 12-2：使用 RANSAC 算法计算基本矩阵

```
int point_count = 100;
CvMat* points1;
CvMat* points2;
CvMat* status;
CvMat* fundamental_matrix;

points1 = cvCreateMat(1,point_count,CV_32FC2);
points2 = cvCreateMat(1,point_count,CV_32FC2);
status = cvCreateMat(1,point_count,CV_8UC1);

/* Fill the points here ... */
for( int i = 0; i < point_count; i++ )
{
    points1->data.fl[i*2] = <x1,i>; //These are points such as
found
    points1->data.fl[i*2+1] = <y1,i>; // on the chessboard
calibration
    points2->data.fl[i*2] = <x2,i>; // pattern.
    points2->data.fl[i*2+1] = <y2,i>;
}

fundamental_matrix = cvCreateMat(3,3,CV_32FC1);
int fm_count = cvFindFundamentalMat( points1, points2,
                                      fundamental_matrix,
                                      CV_FM_RANSAC,1.0,0.99,status );
```

一点警告(与返回 0 时的可能性有关)：如果这些点形成了退化形态，会导致算法失败。当提供的点少于所需的信息时，如一个点出现的次数多于一次，或者当多个点共线或与其他点共面时，这些退化形态就会产生。因此，检查 cvFindFundamentalMat() 的返回值显得十分重要。

极线的计算

一旦有了基础矩阵，我们就能够计算极线。OpenCV 中的 cvComputeCorrespondEpilines() 函数根据一幅图像中的点列，计算其在另一幅图像中对应的极线。注意，给定一幅图像上的任意点，在另外一幅图像总是有对应的极线。每条计算的极线以三点(a, b, c)的形式编码，这样极线就定义如下：

$$ax + by + c = 0$$

【426】

为了计算这些极线，函数需要前面用 `cvFindFundamentalMat()` 算出的基础矩阵。

```
void cvComputeCorrespondEpilines(
    const CvMat* points,
    int which_image,
    const CvMat* fundamental_matrix,
    CvMat* correspondent_lines
);
```

这里第一个参数 `points`，通常的是一个 $N \times 2$ 或者 $N \times 3$ 阵列的点(也可以是 $N \times 1$ 的 2 或 2 通道的多通道矩阵)。参数 `which_image` 必须是 1 或者 2，它标明哪幅图像的点被定义(相对于 `cvFindFundamentalMat()` 中的 `points1` 和 `points2`)。当然，参数 `fundamental_matrix` 是函数 `cvFindFundamentalMat()` 返回的 3×3 的矩阵。最后的参数 `correspondent_lines` 是一个 $N \times 3$ 的浮点数矩阵，表示将写入的结果极线。容易看出，极线方程 $ax + by = c = 0$ 与参数 a 、 b 、 c 的所有归一化无关。它们都默认地被归一化了，故 $a_2 + b_2 = 1$ 。

立体标定

我们已经在摄像机和三维点的背后建立许多可以利用的理论机制。本节将介绍立体标定，下一节则介绍立体校正。立体标定是计算空间上两台摄像机几何关系的过程。相反地，立体校正则是对个体图像进行纠正的过程，这样保证这些图像可以从像平面行对准的两幅图像获得(回顾图 12-4 和图 12-7)。通过这样的校正，两台摄像机的光轴(或者主光线)就是平行的，即所谓在无穷远处相交。当然，我们也可以将两台摄像机的图像标定成许多其他的结果，但是这里(在 OpenCV 中)我们将注意力集中在一些更常见更简单的实例来使主光线在无穷远处相交。

立体标定依赖于查找两台摄像机之间的旋转矩阵 R 和平移向量 T ，如图 12-9 所示。 R 和 T 都是通过函数 `cvStereoCalibrate()` 来计算的，这个函数与我们在第 11 章中看到的 `cvCalibrateCamera2()` 相似，所不同是现在我们有了两台摄像机和一些新的函数用以计算(利用以前的计算)摄像机的失真矩阵、本征矩阵或者基础矩阵。立体标定和单摄像机标定的另一主要不同之处在于，在函数 `cvCalibrateCamera2()` 中，我们以摄像机和棋盘视图之间的一些列旋转和平移结束，而在函数 `cvStereoCalibrate()` 中我们则是寻求单个旋转矩阵和平移向量来联系左右摄像机。

前面我们已经说明如何计算本征矩阵和基础矩阵。但是如何计算左右摄像机的 R 和 T 呢？给定物体坐标系中的任意 3D 点 P ，我们可以分别用两台摄像机的单摄像机标定来将点 P 输入到左右摄像机的摄像机坐标系： $P_l = R_l P + T_l$ 和 $P_r = R_r P + T_r$ 。相应地，从图 12-9 中也同样能明显看出点 P (在两台摄像机上)的两个视图可以用 $P_l = R^T(P_r - T)$ 关联^①，其中 R 和 T 分别两个摄像机之间的旋转矩阵和平移向量。利用这三个等式分别求解旋转和平移，就可以推出下面的简单关系^②：

$$R = R_r(R_l)T$$

$$T = T_r - RT_l$$

【427~428】

给定棋盘角点的多个联合视图，函数 `cvStereoCalibrate()` 利用 `cvCalibrateCamera2()` 来单独求解每个摄像机棋盘视图的旋转和平移参数(请参见第 11 章)。然后将这些旋转和平移结果代入到公式中，就可以求出两台摄像机之间的旋转和平移参数。由于图像噪声和舍入误差，每一对棋盘都会使得 R 和 T 的结果出现细小不同。这时，`cvStereoCalibrate()` 就选用 R 和 T 参数的中值来作为真实结果的初始近似值，然后再运行稳健的 Levenberg-Marquardt 迭代算法查找棋盘角点在两个摄像机视图上的(本地的)最小投影误差，并返回 R 和 T 的结果。为了使得到的立体标定显示更清晰，旋转矩阵被作用到与左摄像机共面的右摄像机上，这样就保证两个图像平面共面并且行不对准(在下面小节中我们将看到如何完成行对准的)

函数 `cvStereoCalibrate()` 有许多参数，但是它们都相当直观易懂，而且大部分都与第 11 章中的 `cvCalibrateCamera2()` 函数相同。

```
bool cvStereoCalibrate(
    const CvMat* objectPoints,
    const CvMat* imagePoints1,
    const CvMat* imagePoints2,
    const CvMat* npoints,
    CvMat* cameraMatrix1,
    CvMat* distCoeffs1,
```

- ① 注意这些项的意义： P_l 和 P_r 分别是左右摄像机相应的坐标系下 3D 点 P 的位置； R_l 和 T_l (或者 R_r 和 T_r) 表示左摄像机(对右摄像机)上由摄像机到 3D 点的旋转和平移矩阵； R 和 T 则是将右摄像机坐标系转换到左摄像机的旋转和平移。
- ② 左右摄像机可以通过两个公式中的下标进行翻转，也可以翻转下标并只在转换公式中将 R 转置来完成。

```
CvMat*           cameraMatrix2,
CvMat*           distCoefs2,
CvSize          imageSize,
CvMat*           R,
CvMat*           T,
CvMat*           E,
CvMat*           F,
CvTermCriteria termCrit,
int             flags=CV_CALIB_FIX_INTRINSIC
);
```

【428~429】

第一个参数 `objectPoints` 是一个 $N \times 3$ 的矩阵，它包含三维物体在 M 幅图像中的每一幅图像上的 K 个点的物理坐标，因此 $N = K \times M$ 。当使用棋盘作为三维物体时，这些点就落在物体的坐标系上，棋盘的左上角作为原点(棋盘平面上点的 Z 坐标通常置为 0)，但任意已知的 3D 点都可能被用到 `cvCalibrateCamera2()` 中。

现在我们有两台摄像机，分别标记为“1”和“2”^①。因此有 `imagePoints1` 和 `imagePoints2`，它们是 $N \times 2$ 的矩阵，分别存储由 `objectPoints` 提供的所有物体参考点在左右坐标系上的位置。如果使用一个棋盘来完成两个摄像机的标定，那么 `imagePoints1` 和 `imagePoints2` 就正好分别是调用 `cvFindChessboardCorners()` 求取左右视图的返回值。

参数 `npoints` 是每幅图像上的点数目，为 $M \times 1$ 的矩阵。

参数 `cameraMatrix1` 和 `cameraMatrix2` 是 3×3 的摄像机矩阵，而参数 `distCoefs1` 和 `distCoefs2` 分别是摄像机 1 和摄像机 2 的 5×1 畸变矩阵。记住，这些矩阵中前两个径向参数首先出现，然后是两个切向参数，最后是第三个径向参数(参见第 11 章对畸变系数的讨论)，因为第三个像像畸变参数是 OpenCV 发展中新加入的成员，因此放在最后；它主要是用于广角(鱼眼)镜头。这些摄像机内参数的使用由参数 `flags` 控制，如果 `flags` 设置成 `CV_CALIB_FIX_INTRINSIC`，这些参数便仅用于标定过程中；如果 `flags` 为 `CV_CALIB_USE_INTRINSIC_GUESS`，参数便被用作优化摄像机的内参数和畸变参数的初始值，而且在 `cvStereoCalibrate()` 返回时被改进值所更换。结合 `flags` 的其他设置，使之具有和函数 `cvCalibrateCamera2()` 完全相同的可能值，这样这些参数在

① 简单而言，就是用“1”来表示左摄像机，“2”来表示右摄像机。你也可以交换它们以本书讨论的相反方式来描述结果旋转和平移结果。最重要的是保持摄像机的物理对准，从而保证它们的扫描线能近似对应，以获得更好的标定结果。

`cvStereoCalibrate()` 中就从 0 开始计算得到。换句话说，可以利用 `cvStereoCalibrate()` 函数一次性计算内参数、外参数和立体参数^①。

参数 `imageSize` 是以像素为单位的图像大小。它仅用于细化或者计算内参数且 `flags` 不等于 `CV_CALIB_FIX_INTRINSIC` 时。

R 和 T 项是输出参数，在函数返回时被填入待求的(联系左右摄像机的)旋转矩阵和平移向量。参数 E 和 F 为可选，如果不为空，`cvStereoCalibrate()` 就会计算和填充这些 3×3 的本征矩阵和基础矩阵。之前我们已经见过参数 `termCrit` 多次了，它设置内部最优化准则，或者指示计算参数的变化在结构 `termCrit` 中的阈值内时，迭代一定次数后将程序终止或者暂停。这个函数的一个典型调用是 `cvTermCriteria(CV_TERMCRIT_ITER + CV_TERMCRIT_EPS, 100, 1e-5)`。

最后，我们已经稍微讨论了参数 `flags`。如果已经将两台摄像机标定，并确认了结果，你便可以使用 `CV_CALIB_FIX_INTRINSIC` “强行重置”先前的单摄像机标定结果。如果认为两台摄像机的初始标定不够好，可以设置参数 `flags` 为 `CV_CALIB_USE_INTRINSIC_GUESS` 来，用它来细化内参数和畸变参数。如果摄像机没有分别进行标定，你可以设置第 11 章 `cvCalibrateCamera2()` 函数的参数 `flags` 为相同的设置。

一旦有了旋转或者平移值(R, T)或者基础矩阵 F ，我们就可以利用这些结果来校正两幅立体图像，使得极线沿着图像行对准，并且穿过两幅图像的扫面线也是相同的。尽管 R 和 T 并不决定唯一的立体校正，但是在下一小节中，我们就会看到怎样利用这些项(以及其他约束条件)。

立体校正

当两个像平面是完全的行对准时(见图 12-4)，计算立体视差是最简单的。不幸的是，如前面所讨论的，由于两台摄像机几乎不可能有准确的共面和行对准的成像平面，完美的对准结构在真实的立体系统中几乎不存在。图 12-7 显示了立体校正的目的：我们要对两台摄像机的图像平面重投影，使得它们精确落在同一个平面上，而且图像的行完全地对准到前向平行的结构上。如何选择特定的平面使摄像机保持数学对准依赖于所使用的算法。在下文中，我们讨论使用 OpenCV 提及的两个

① 注意：尝试一次性求解多个参数有时会导致结果发散到无意义的值。求解方程组是一项技术活，必须校验结果。可以看到在标定中一些这样的考虑和校正源码实例，其中便利用对极约束来检查标定结果。

情况。

我们要保证两个摄像机的图像行在校正之后是对准的，使得立体匹配(在不同摄像机视场中发现相同点)更可靠，计算更可行。注意，只在图像的一行上面搜索另一图像的匹配点能够提高可靠性和算法效率。让每个图像平面都落在一个公共成像面上并水平对准的结果是极点都位于无穷远。即一幅图像上的投影中心成像与另一个像平面平行。但是由于可选择的前向平行平面个数是有限的，我们需要加入更多的约束，包括视图重叠最大化和畸变最小化，接下来我们讨论如何选择算法。

【430】

对准两个图像平面后的结果有八项，左右摄像机各四项。对每个摄像机，我们都会有一个畸变向量 `distCoeffs`、一个旋转矩阵 `R_rect`(应用于摄像机)、校正和为未校正的摄像机矩阵(`M_rect` 和 `M`)。从这些项里使用函数 `cvInitUndistortRectifyMap()`(接下来就简要论述)创建一个映射，该函数从原始图像插值出一幅新的校正图像^①。

有很多算法可以计算我们的校正项，OpenCV 实现了其中的两种：一是 Hartley 算法，它只使用基础矩阵来生成非标定立体视觉；二是 Bouguet 算法^②，它使用两台标定摄像机的旋转和平移参数。Hartley 算法可以通过单个摄像机记录的运动推导出立体结构，虽然单个摄像机会(当立体校正后)比 Bouguet 标定算法产生更多的畸变图像。在可以使用标定模式的情形下，如机器人臂或者安全摄像机装备上，Bouguet 算法更简单自然。

非标定立体校正：Hartley 算法

Hartley 算法旨在找到最小化两幅立体图像的计算视差时将对极点映射到无穷远处的单应矩阵。这可通过匹配两幅图像之间的对应点简单完成。因此，就可以绕过计算两个摄像机的摄像机内参数，因为这样的内参数信息式隐含在匹配点对之中。所以我们只需计算基础矩阵，它可以通过前面描述的 `cvFindFundamentalMat()` 函数所获得的两个场景视图上的 7 个以上匹配点来获得。

Hartley 算法的优点是，在线立体标定可以简单地通过场景中的观察点来完成。缺点是场景的图像比例未知。例如，如果使用棋盘来生成点对，就无法知道棋盘从一

① 在 OpenCV 中，只有当对极在图像矩形框外面时才能对图像进行立体校正。因此，当立体构型的基线很宽或者摄像机对视过多时，这个校正算法就可能没有用处了。

② Bouguet 算法是由 Tsai[Tsa87]第一次提出的方法的简化实现。Jean-Yves Bouguet 从来没有在其著名的 MATLAB 摄像机标定工具箱之外公布这个算法。

端到另一端是 100 米还是 100 厘米。这样我们就不能确定摄像机的内参数矩阵，因为摄像机可能有不同的焦距、倾斜像素、不同的投影中心和/或不同的主点。因此，我们只能根据投影变换来重构 3D 物体。也就是说，一个目标的不同比例尺或者投影看起来都是相同的(比如说，尽管 3D 物体不同，但它们具有相同的 2D 坐标)。这些问题如图 12-10 所示。

【43】

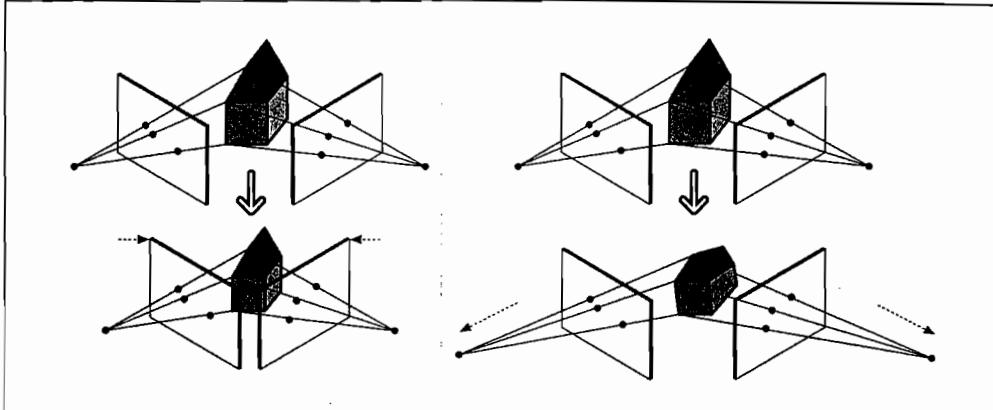


图 12-10：立体重建的非惟一性：如果不知道物体的大小，那么根据和(左)摄像机距离的远近，不同大小的物体看起来有可能是相同的；如果不知道摄像机内参数，不同的投影看起来也是相同的——比如它们有不同的焦距和主点

1. 假设我们有基础矩阵 F (该矩阵需要至少 7 个点来计算)，Hartley 算法流程如下(更多细节，请参见 Hartley 的原作[Hartley98])。通过下面的关系式用基础矩阵来计算两个极点及其关系： $Fe_r = 0$ 和 $(e_r)^T F$ 上面两式分别对应左右两极点。
2. 我们先求第一个单应矩阵 H_r ，它将右极点映射到无穷远 $(1, 0, 0)^T$ 处的二维齐次点。由于一个单应矩阵有 7 个常数(没有比例尺)，其中的三个用来做无穷远的映射，剩下的四个自由度来选择 H_r 。因为 H_r 的大部分选择都会导致非常扭曲的图像，所以这四个自由度最容易导致混乱。为了找到一个好的 H_r ，我们选择图像上的一点，使得这一点上有最小的畸变，即只允许刚性旋转和平移而不是剪切。对这样的点的一个合理选择就是图像原点，我们进一步假设极点落在 x 轴上(下面的一个旋转矩阵将会完成这个)： $(e_r)^T = (f, 0, 1)$ 给定这些坐标，如下矩阵：会将这样的一个极点映射到无穷远。

$$G = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/k & 0 & 1 \end{pmatrix}$$

3. 对右图像上所选择的一个感兴趣点(这里选择原点)，我们计算点到图像原点的

平移矩阵 T 和将极点指向 $(e_r)^T = (f, 0, 1)$ 的旋转矩阵 R , 则需要的单应矩阵就是 $H_r = GRT$ 。 【432】

4. 接下来搜索匹配的单应矩阵 H_l , 它将左极点发送到无穷远, 并保证两幅图像的行对准。通过步骤 2 中的三个约束可以容易地将左极点转到无穷远。为了能够行对准, 仅仅的事实是行对准使两幅图像的所有匹配点间距离和最小。也就是说, 我们查找 H_l , 使得左右匹配点的总的视差最小。 $\sum_i d(H_l p'_i, H_r p'_i)$ 这两个单应矩阵就定义了立体校正。

虽然说这个算法的细节有点拐弯抹角, 但是 `cvStereoRectifyUncalibrated()` 完成了所有的复杂工作。这个函数的叫法有点不妥当, 因为它不是校正非标定立体图像, 而是计算可以用于校正的单应矩阵。算法调用如下:

```
int cvStereoRectifyUncalibrated(
    const CvMat* points1,
    const CvMat* points2,
    const CvMat* F,
    CvSize imageSize,
    CvMat* Hl,
    CvMat* Hr,
    double threshold
);
```

函数 `cvStereoRectifyUncalibrated` 中, 算法输入为左右图像间的 $2 \times K$ 匹配点数组 `points1` 和 `points2`。上面计算出来的基础矩阵 `F` 作为参数传入。我们对 `imageSize` 已经熟悉了, 它只是描述了在标定中使用到图像的宽和高。函数变量 `Hl` 和 `Hr` 返回校正的单应性矩阵。最后, 如果点和对应的极线之间的距离超过了设定的阈值, 算法会删除对应点^①。

如果摄像机大致有相同的参数, 并且是设置为近似水平对准的向前平行构型, 那么 Hartley 算法的最终校正输出就非常像我们接下来描述的标定情形。如果知道了场景中物体的大小和 3D 几何, 我们就能够获得和标定情形相同的结果。

标定立体校正: Bouguet 算法

给定立体图像间的旋转矩阵和平移(R, T), 立体校正的 Bouguet 算法就是简单地使

① Hartley 算法对已经由单摄像机标定法校正的图像最有效, 对高度畸变的图像则完全无效。有点讽刺意味的是, 我们的“免标定”方法只是对无畸变图像(图像的参数来自以前的标定)有用。另一个非标定 3D 方法请参见 Pollefeys[Pollefeys99a]。

两图像中的每一幅重投影次数最小化(从而也使重投影畸变最小化), 同时使得观测面积最大化。

为了使图像重投影畸变最小化, 将右摄像机图像平面旋转到左摄像机图像平面的旋转矩阵 R 被分离成图像之间的两部分, 我们称之为左右摄像机的两个合成旋转矩阵 r_l 和 r_r 。每个摄像机都旋转一半, 这样其主光线就平行地指向其原主光线指向的向量和方向。如所标记的, 这样的旋转可以让摄像机共面但是行不对准。为了计算将左摄像机极点变换到无穷远并使极线水平对准的矩阵 R_{rect} , 我们创建一个由极点 e_1 方向开始的旋转矩阵。让主点(c_x, c_y)作为左图像的原点, 极点的方向就是两台摄像机投影中心之间的平移向量方向:

$$e_1 = \frac{T}{\|T\|}$$

下一个向量 e_2 必须与 e_1 正交, 没有其他限制。对 e_2 而言, 很好的一个选择就是选择与主光线正交的方向(通常沿着图像平面)。这可以通过计算 e_1 和主光线方向的叉积来得到, 然后将它归一化到单位向量:

$$e_2 = \frac{[-T_y T_x 0]^\top}{\sqrt{T_x^2 + T_y^2}}$$

第三个向量只与 e_1 和 e_2 正交, 它可以通过叉积得到:

$$e_3 = e_1 \times e_2$$

此时, 将左摄像机的极点转换到无穷远处的矩阵如下:

$$R_{rect} \begin{bmatrix} (e_1)^\top \\ (e_2)^\top \\ (e_3)^\top \end{bmatrix}$$

这个矩阵将左图像绕着投影中心旋转, 使得极线变成水平, 并且极点在无穷远处。两台摄像机的行对准通过设定来实现:

$$\begin{aligned} R_l &= R_{rect} r_l \\ R_r &= R_{rect} r_r \end{aligned}$$

我们同样可以计算校正后的左右摄像机矩阵 M_{rect_l} 和 M_{rect_r} , 但是与投影矩阵 P_l 和 P_r 一起返回:

$$P_l = M_{\text{rect}_l} P'_l \begin{bmatrix} f_{x_{-l}} & \alpha_l & c_{x_{-l}} \\ 0 & f_{y_{-l}} & c_{y_{-l}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

和

$$P_r = M_{\text{rect}_r} P'_r \begin{bmatrix} f_{x_r} & \alpha_r & c_{x_r} \\ 0 & f_{y_r} & c_{y_r} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad 【433 \sim 435】$$

(其中, α_l 和 α_r 是像素畸变比例, 它们在现代摄像机中几乎总是等于 0)。投影矩阵将齐次坐标中的 3D 点转换到如下齐次坐标系下的 2D 点:

$$P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

其中, 屏幕坐标为 $(x/w, y/w)$ 。如果给定屏幕坐标和摄像机内参数矩阵, 二维点同样可以重投影到三维中, 重投影矩阵如下:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)T_x \end{bmatrix}$$

这里, 除 c'_x 外的所有参数都来自于左图像, c'_x 是主点在右图像上的 x 坐标。如果主光线在无穷远处相交, 那么 $cx = c'_x$, 并且右下角的项为 0。给定一个二维齐次点和其关联的视差 d , 我们可以将此点投影到三维中:

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

三维坐标就是 $(X/W, Y/W, Z/W)$ 。

应用刚刚描述的 Bouguet 校正方法即可生成图 12-4 中的理想立体构型。为旋转图像选择新图像中心和边界从而使叠加视图的面积最大化。大体上来说, 这正好生成一个相同的摄像机中心和两个图像区域共有的最大高度和宽度作为新的立体视图平面。

```

void cvStereoRectify(
    const CvMat* cameraMatrix1,
    const CvMat* cameraMatrix2,
    const CvMat* distCoeffs1,
    const CvMat* distCoeffs2,
    CvSize imageSize,
    const CvMat* R,
    const CvMat* T,
    CvMat* Rl,
    CvMat* Rr,
    CvMat* Pl,
    CvMat* Pr,
    CvMat* Q=0,
    int flags=CV_CALIB_ZERO_DISPARITY
);

```

【435~436】

对函数 `cvStereoRectify()`^① 来说，输入的是我们所熟悉的由 `cvStereoCalibrate()` 返回的原始摄像机矩阵和畸变向量。接下来的参数 `imageSize`，是用来执行标定的棋盘图像的大小。同样传入由 `cvStereoCalibrate()` 返回的左右摄像机间旋转矩阵 `R` 和平移向量 `T`。

返回参数是 3×3 矩阵 R_l 和 R_r ，是从前述公式推导而来的左右摄像机平面间的行对准的校正旋转矩阵。同样地，我们也获得了 3×4 的左右投影方程 P_l 和 P_r 。一个可选返回参数为 `Q`，是前面叙述过的 4×4 的重投影矩阵。

`Flags` 参数的默认设置为无穷远处的视差，如图 12-4 中的通常情形。不设置 `flags` 参数，意味着我们想要摄像机相互重合(比如轻微的“交叉注视”)，从而使得有限距离内的视差为 0(这可能在特定距离附近的更高深度分辨率的情形下需要)

如果参数 `flags` 不为 `CV_CALIB_ZERO_DISPARITY`，那我们要对如何完成校正系统更加小心。回顾以前，我们是相对于左右摄像机的主点(c_x , c_y)来校正系统的，因此图 12-4 的测量值也是和这些点的位置相关。基本上说，我们必须修正这些距离，使得： $\tilde{x}' = x' - c_x^{\text{right}}$ 和 $\tilde{x}' = x' - c_x^{\text{left}}$ 当视差设置成无穷大时，我们有 $c_x^{\text{left}} = c_x^{\text{right}}$ (比如，当 `CV_CALIB_ZERO_DISPARITY` 传递到 `cvStereoRectify()` 中)，并且我们可以传递平面像素坐标(或者视差)到公式中以计算深度。但是如果 `cvStereoRectify()` 被

① 再次说明，`cvStereoRectify()` 的名称不妥当是因为此函数是计算可以用来校正的参数项，而不是真实的对立体图像进行校正。

调用时没有传入 `CV_CALIB_ZERO_DISPARITY`, 一般有 $c_x^{\text{left}} \neq c_x^{\text{right}}$ 。因此, 即使公式 $Z = fT/(x_l - x_r)$ 仍然保持不变, 但要切记, x_l 和 x_r 不是针对图像中心, 而是针对各自的主点 c_x^{left} and c_x^{right} , 它们与 x_l 和 x_r 不同。所以, 如果计算视差 $d = x_l - x_r$, 那么在计算 Z 之前就应该将其修正为: $Z fT/(d - (c_x^{\text{left}} - c_x^{\text{right}}))$ 。

校正映射

一旦有了立体标定项, 我们就可以单独调用 `cvInitUndistortRectifyMap()` 来事先计算左右视图的校正查找映射表。对任何图像到图像的映射函数, 由于目标位置是浮点型的缘故, 正向映射(即根据原始图像上的点计算其到目标图像上的点), 不会命中目标图像对应的像素位置, 因为目标图像看起来像瑞士硬干酪。因此我们采用逆向映射: 对目标图像上的每个整型的像素位置, 首先查找出其对应源图像上的浮点位置, 然后利用周围源像素的整型值插值出新的值来。这种查找一般使用双线性插值方法, 在第 6 章中的 `cvRemap()` 函数中遇到。 【436~437】

这个校正的过程如图 12-11 所示。图中公式流程就是真实的从(c)到(a)的后向校正过程, 称为逆向映射。对校正图像(c)中的每个整型像素, 我们查找它在非畸变图像(b)上的坐标, 并用这些坐标回溯在原始图像(a)上的真实(浮点)坐标。浮点坐标上的像素值通过在原始图像上的邻近整型像素位置插值得到, 这个值被赋给目的图像(c)上的校正后整型像素位置。在校正图像都被赋值之后, 我们经常将它剪切以增大左右图像间的叠加面积。

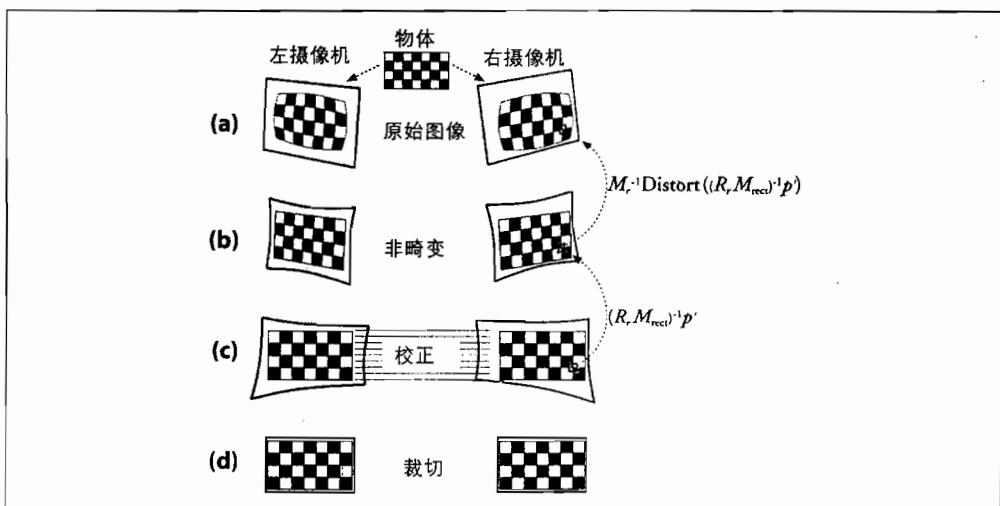


图 12-11: 立体校正。对左右摄像机而言, (a)原始图像、(b)非畸变、(c)校正和(d)最后裁切成两幅图像间的重叠区域。校正实际上是由(c)到(a)的反向过程

图 12-11 中数学描述的函数实现是 `cvInitUndistortRectifyMap()`。这个函数被调用了两次，左右图像各调用一次。

```
void cvInitUndistortRectifyMap(
    const CvMat* M,
    const CvMat* distCoeffs,
    const CvMat* Rrect,
    const CvMat* Mrect,
    CvArr* mapx,
    CvArr* mapy
);
```

`cvInitUndistortRectifyMap()` 使用的输入参数是 3×3 的摄像机矩阵 M 、校正后的 3×3 摄像机矩阵 M_{rect} 、 3×3 的旋转矩阵 R_{rect} 和 5×1 的摄像机畸变参数 $distCoeffs$ 。

如果使用 `cvStereoRectify()` 去标定立体摄像机，我们可以从 `cvStereoRectify()` 直接读出 `cvInitUndistortRectifyMap()` 的输入参数，使用左参数校正左图像，右参数校正右图像。对 R_{rect} ，可用 `cvStereoRectify()` 的 R_l 和 R_r ，对 M ，使用 `cameraMatrix1` 或者 `cameraMatrix2`。对 M_{rect} ，我们可以用 `cvStereoRectify()` 中的 $3 \times 4 P_l$ 或 P_r 的前面三列，但为方便起见，函数允许我们直接输入 P_l 和 P_r ，并从中读出 M_{rect} 。

另一方面，如果使用 `cvStereoRectifyUncalibrated()` 校正立体摄像机，那我们必须预处理一下单应性矩阵。尽管在理论和实际中可以不使用摄像机内参数来校正立体视觉，但 OpenCV 中没有这样的直接处理函数。假设没有从先前的标定中获得 M_{rect} ，那正确的流程应该是令 M_{rect} 等于 M 。然后对 `cvInitUndistortRectifyMap()` 中的 M_{rect} 来说，我们需要分别计算左右校正的 $R_{rect_l} = M_{rect_l}^{-1} H_l M_l$ （或无法得到 $R_{rect_l} = M_l^{-1} H_l M_l M_{rect_l}^{-1}$ ）以及 $R_{rect_r} = M_{rect_r}^{-1} H_r M_r$ （或无法得到 $R_{rect_r} = M_r^{-1} H_r M_r M_{rect_r}^{-1}$ ）。最后，我们还需要每台摄像机的畸变系数来赋给 5×1 的 $distCoeffs$ 参数。

函数 `cvInitUndistortRectifyMap()` 返回的输出值是查找映射表 `mapx` 和 `mapy`。对目标图像上的每个像素来说，映射表说明应该从什么位置插值源像素；并且映射表可以直接被 `cvRemap()` 嵌入使用，我们第一次在第 6 章中见到过该函数。如我们提到的，函数 `cvInitUndistortRectifyMap()` 被左右摄像机分别调用，这样可以获得它们的各自不同的重映射参数 `mapx` 和 `mapy`。当每次我们有新的左右立体图像需要校正时可以使用左右映射表时，函数 `cvRemap()` 也可能被调用。图 12-12 是

图像立体对的立体非畸变化和校正的结果。注意，特征点在非畸变校正图像上是如何变成水平对准的。

【437~438】

立体匹配

立体匹配——匹配两个不同的摄像机视图的 3D 点——只有在两摄像机的重叠视图内的可视区域上才能被计算。再重复一下，这就是为什么如果将摄像机尽量靠近前向平行(至少在你成为立体视觉的专家之前)以获得更好结果的一个原因。一旦知道了摄像机的物理坐标或者场景中物体的大小，就可以通过两个不同摄像机视图中的匹配点之间的三角测量视差值 $d = x_l - x_r$ (当主光线在有限距离内相交时 $d = x_l - x_r(c_x^{\text{left}} - c_x^{\text{right}})$)来求取深度值。没有这样的物理信息，我们就只能计算深度的比例关系。如果我们没有摄像机内参数，如当使用 Hartley 算法时，我们只能在投影变换意义上计算点的位置(见图 12-10)。

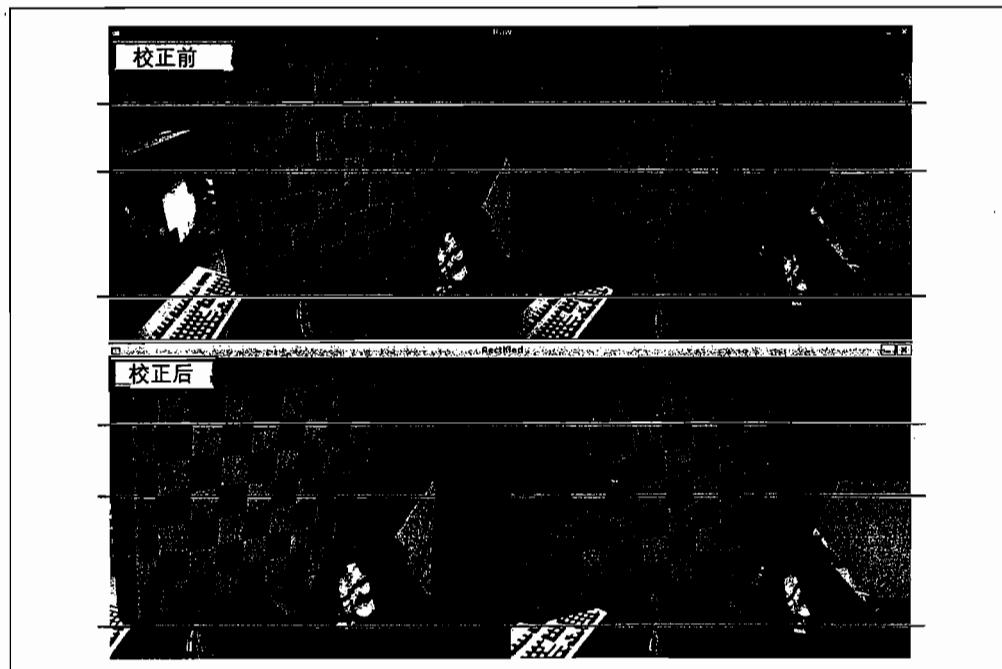


图 12-12：立体校正：原始的左右像对(上)和校正后的左右像对(下)；注意，桶形畸变(棋盘图形顶部)被纠正了，并且校正图像中的扫描行也已经被对准了

OpenCV 实现了一个快速有效的块匹配立体算法 `cvFindStereoCorrespondenceBM()`，它与 Kurt Konolige 提出的算法相似。它使用了一个叫“绝对误差累计”的

小窗口(SAD)来查找左右两幅立体校正图像之间的匹配点^①。这个算法只查找两幅图像之间的强匹配点(强纹理)。因此,在一个强纹理场景中(例如出现在森林户外),每个像素都有可计算的深度。而在弱纹理场景里(比如室内的走廊),则只需要计算少数点的深度。对于处理非畸变的校正立体图像,块匹配立体匹配算法有以下三个步骤。

【438~439】

1. 预过滤,使图像亮度归一化并加强图像纹理。
2. 沿着水平极线用 SAD 窗口进行匹配搜索。
3. 再过滤,去除坏的匹配点。

在预过滤中,输入图像被归一化处理,从而减少了亮度差异,也增强了图像纹理。这个过程通过在整幅图像上移动窗口来实现的,窗口的大小可以是 5×5 、 7×7 (默认值)、…… 21×21 (最大)。窗口的中心像素 I_c 由 $\min[\max(I_c - \bar{I}, -I_{cap}), I_{cap}]$ 来代替,其中 \bar{I} 是窗口的平均值, I_{cap} 是一个正数范围,默认值为 30。这个方法通过设置 `CV_NORMALIZED_RESPONSE` 标志被激活。另外一个标志是 `CV_LAPLACIAN_OF_GAUSSIAN`,它在图像的平滑版本上运行峰值检波器。

匹配过程通过滑动 SAD 窗口来完成。对左图像上的每个特征而言,我们搜索右图像中的对应行以找到最佳匹配。校正之后,每一行就是一条极线,因此右图像上的匹配位置就一定会在左图像的相同行上(即具有同样的 y 坐标)。如果特征有足够多可检测的纹理,并且位于右摄像机视图内,就可以找出该匹配位置(如图 12-16 所示)。如果左特征像素位于 (x_0, y_0) ,那么对水平前向平行的摄像机排列情形而言,它的匹配点(如果有的话)就一定与 X_0 在同一行,或者是在 x_0 的左边,见图 12-13。对前向平行的摄像机来说, x_0 是 0 视差,并且左边的视差更大。对两个摄像机之间有夹角的情况,匹配点则可能出现负的视差(x_0 以右)。控制匹配搜索的第一个参数是 `minDisparity`,它说明了匹配搜索从哪里开始,默认值为 0。这时,程序在 `numberOfDisparities` 设置的(默认为 64)视差内开始搜索。视差是离散的,亚像素精度通过参数 `subPixelDisparities`(默认值 16)来设定。我们可以通过限制极线上匹配点的搜索长度来减少搜索的视差个数,从而缩减计算的时间。记住,视差越大表示距离越近。

通过设置最小视差和搜索的视差个数就可以建立起一个双眼视界,这个 3D 体被立体算法的搜索范围所覆盖。图 12-14 显示了由三种不同视差(20、17、16)限制开始的 5 像素的视差搜索范围。如图所示,每组不同的视差限制和视差个数都产生了不同的深度可知的双眼视界。在这个范围之外就不能获得深度,在深度图上会出现一

① 这个算法可以从 Videre 的 FPGA 立体硬件系统中得到(见[Videre])。

个深度未知的“空洞”。缩小摄像机间的极线距离 T 、减小焦距长度、增加立体视差的搜索范围或者增大像素宽度，都可以使双眼视差变得更大。

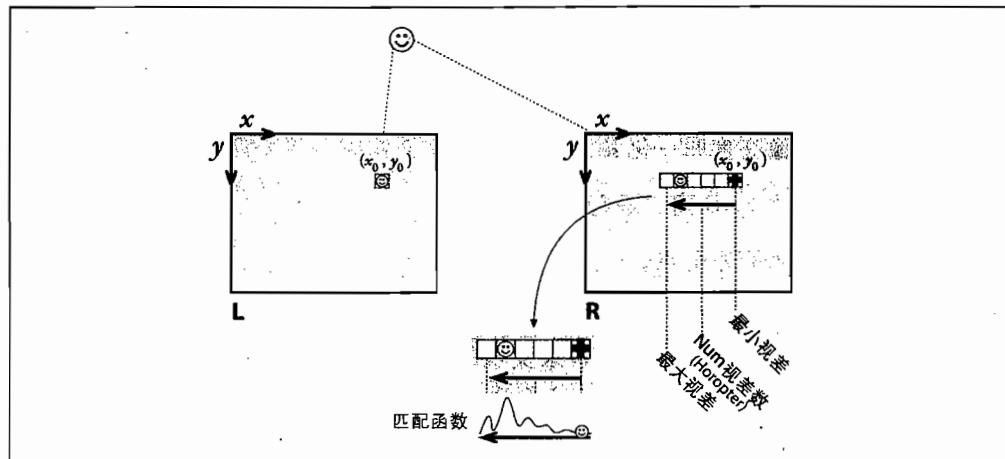


图 12-13：左图像特征的右图像匹配一定出现在相同的行上，并且在相同的坐标点上(或者左边)，这个点就是匹配搜索从 $minDisparity$ 点(here, 0)开向左移动视差个数的位置。图的下半部分是基于窗口特征匹配的特性匹配函数

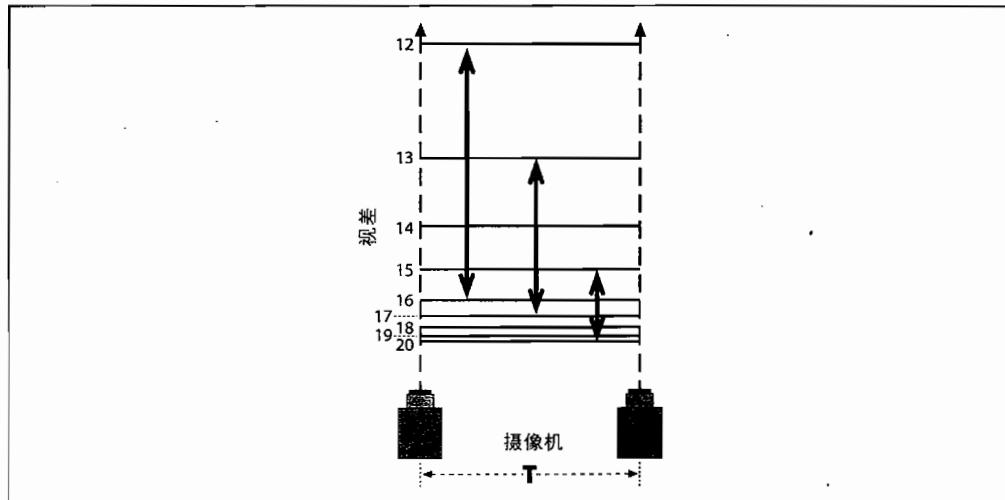


图 12-14：每条直线表示整型像素从 20 到 12 变换时的恒等视差平面；5 个像素内的视差搜索范围包含了不同的两眼视界范围(图中的垂直箭头)，并且不同的最大视差产生不同的两眼视界

双眼视界的匹配有一个隐含的约束，叫顺序约束，它简单地规定了特征从左视图到

右视图转换时顺序保持一致。可能会有特征缺失，这是因为有遮挡和噪声的缘故，使得左图像上的特征在右图像上没有发现，但是发现它们的顺序保持不变。同样地，右图像上也可能有一些特征在左摄像机上不能识别(称为插入)，但是插入不会改变特征的顺序，即使这些特征可能会扩散。图 12-16 中的过程反映了水平扫描线上匹配特征的顺序约束。

【440~442】

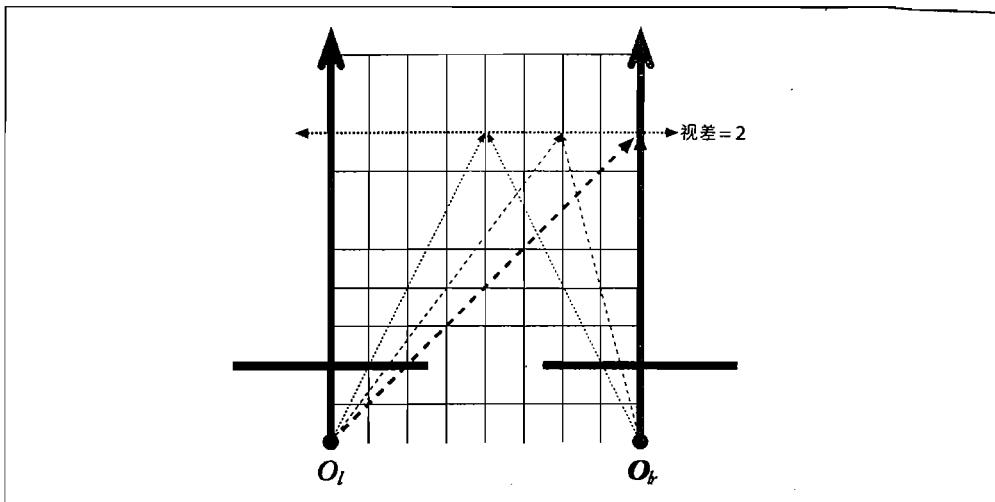


图 12-15：一个固定视差形成了离摄像机距离不变的平面

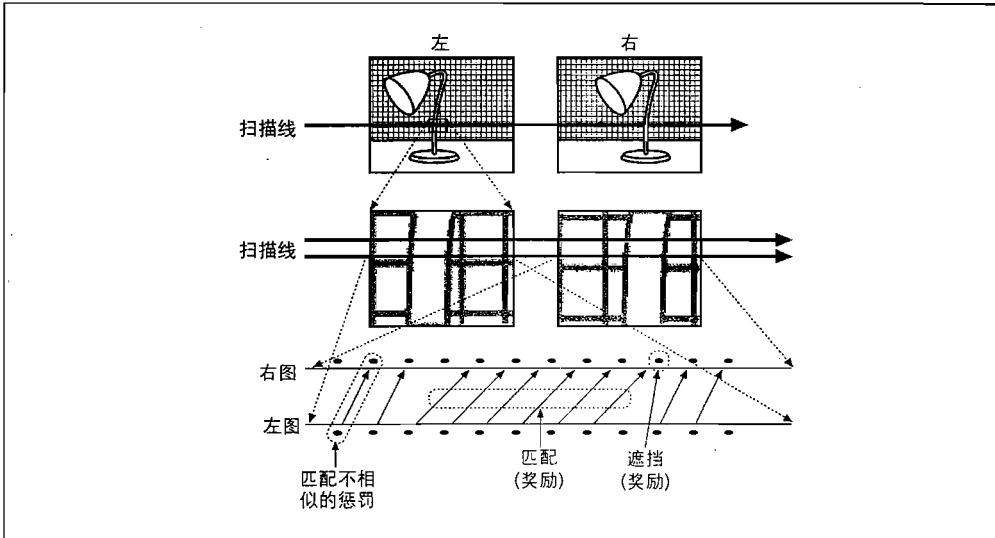


图 12-16：立体匹配由分配左右图像匹配行之间的点匹配开始：台灯的左右图像(上图)、单一扫描线放大(中图)、分派匹配的可视化(下图)

给定允许的最小视差增量，通过下面的公式，我们能确定可以获得的最小深度范围精度。

$$\Delta Z = \frac{Z^2}{fT} \Delta d$$

牢记这个公式是很有用的，因为你可以知道从立体实验台中预期获得的是哪种深度精度。

在匹配之后，我们就开始后过滤处理。图 12-13 的下部显示了一个典型的匹配功能响应，因为特征是从最小视差里扫描出来的最大视差。请注意，匹配值经常具有一个特点，就是强烈的中央峰被副瓣所包围。一旦确定了两个视图的待选特征匹配，就可用后过滤来预防虚匹配。OpenCV 通过一个叫 `uniquenessRatio` 的参数(默认值为 12)来使用匹配功能模式，这个参数可以在 `uniquenessRatio > (match_val - min_match) / min_match` 的地方过滤掉匹配值。【440~442】

OpenCV 使用了 `textureThreshold` 参数来保证有足够的纹理以克服随机噪声。这是对 SAD 窗口响应的一个限制，它使得任何响应小于 `textureThreshold` 的匹配不予考虑。最后，由于匹配窗口捕捉的是物体一侧的前景和另一侧的背景，基于块的匹配在物体的边界附近会有一些问题。这会导致同时产生大小视差的局部区域，我们称它为散斑。为了避免出现这种边界匹配，可以通过设置参数 `speckleWindowSize` 来在散斑窗口(大小从 5×5 到 21×21)上设置一个散斑探测器，这个参数的默认值为 9，窗口大小默认值为 9×9 。在散斑窗口内部，只有探测到的最大最小视差在 `speckleRange` 范围(默认范围是 4)内的匹配才能被接受。

立体视觉对监控系统、导航、机器人技术这类有实时性能要求的系统来说，变得尤其重要。所以要把立体匹配过程设计成能够快速运行才行。因此，我们不能为每次调用 `cvFindStereoCorrespondenceBM()` 的匹配过程持续分配内部离散缓存。

基于块的匹配参数和内部离散缓存保存在一个叫 `CvStereoBMState` 的结构体内。

```
typedef struct CvStereoBMState {
    //pre filters (normalize input images):
    int     preFilterType;
    int     preFilterSize; //for 5x5 up to 21x21
    int     preFilterCap;
    //correspondence using Sum of Absolute Difference (SAD):
    int     SADWindowSize; // Could be 5x5, 7x7, ..., 21x21
    int     minDisparity;
    int     number_of_Disparities; //Number of pixels to search
    //post filters (knock out bad matches):
```

```

int      textureThreshold; //minimum allowed
float    uniquenessRatio;// Filter out if:
           // [ match_val - min_match <
           // uniqRatio*min_match ]
           // over the corr window area
int      speckleWindowSize;//Disparity variation window
int      speckleRange;//Acceptable range of variation in window
// temporary buffers
CvMat*  preFilteredImg0;
CvMat*  preFilteredImg1;
CvMat*  slidingSumBuf;
} Cv_StereoBMState;
typedef struct CvStereoBMState {
//pre filters (normalize input images):
int      preFilterType;
int      preFilterSize;//for 5x5 up to 21x21
int      preFilterCap;
//correspondence using Sum of Absolute Difference (SAD):
int      SADWindowSize; // Could be 5x5,7x7, ..., 21x21
int      minDisparity;
int      numberofDisparities;//Number of pixels to search
//post filters (knock out bad matches):
int      textureThreshold; //minimum allowed
float    uniquenessRatio;// Filter out if:
           // [ match_val - min_match <
           // uniqRatio*min_match ]
           // over the corr window area
int      speckleWindowSize;//Disparity variation window
int      speckleRange;//Acceptable range of variation in window
// temporary buffers
CvMat*  preFilteredImg0;
CvMat*  preFilteredImg1;
CvMat*  slidingSumBuf;
} CvStereoBMState;

```

【443~444】

这个固定的结构在函数 `cvCreateStereoBMState()` 中分配并返回。这个函数使用的参数 `preset` 可以设置为下面的任意一个。

- `CV_STEREO_BM_BASIC` 将所有参数设置为默认值。

- **CV_STEREO_BM_FISH_EYE** 将参数设置成可以处理广角镜头。
- **CV_STEREO_BM_NARROW** 将参数设置成可适用窄视场的立体摄像机。

这个函数也有一个可选参数 `numberOfDisparities`, 如果为 0, 它就从 `preset` 中获得默认值, 下面是规格说明:

```
CvStereoBMState* cvCreateStereoBMState(
    int presetFlag=CV_STEREO_BM_BASIC,
    int numberOfDisparities=0
);
```

状态结构 `CvStereoBMState()` 通过调用函数来释放:

```
void cvReleaseBMState(
    CvStereoBMState **BMState
);
```

在调用 `cvFindStereoCorrespondenceBM` 期间, 可以通过为状态结构的各个字段分配新属性值随时调整任何的立体匹配参数。

最后, `cvFindStereoCorrespondenceBM()` 输入校正图像对, 并输出视差映射, 其结构如下:

```
void cvFindStereoCorrespondenceBM(
    const CvArr      *leftImage,
    const CvArr      *rightImage,
    CvArr           *disparityResult,
    Cv StereoBMState *BMState
);
```

【444~445】

立体标定、校正及相应的示例代码

让我们使用一个实例程序以及其中的代码来把所有的这些知识综合起来, 实例程序将从一个叫 `list.txt` 的文件中读入棋盘图形。文件中包含交替的左右立体(棋盘)像对序列, 可以用来标定摄像机和校正图像。再次注意, 我们是假设已经将摄像机排列好了, 其图像扫描线是粗略的物理对齐, 从而使得每台摄像机本质上都具有相同的

视场。这可以避免极点在图像内^①的问题，还可以在重投影视差最小化的同时使立体重叠面积最大。

在代码中(例 12-3)，我们首先读入左右图像对，找出亚像素精度的棋盘角点，然后在能够看到所有棋盘的图像上设定目标的图像点。这个过程是可以随时显示的。给定这些在合格棋盘图像上的查找点序列后，代码调用函数 `cvStereoCalibrate()` 来标定图像。标定后我们就有了摄像机矩阵`_M`、两台摄像机的畸变向量`_D`、平移向量`_T`、本征矩阵`_E`和基础矩阵`_F`。

下面是一个小插曲，通过检查图像上点与另一幅图像的极线的距离远近来评价标定的精度。为了实现这个目的，我们使用 `cvUndistortPoints()`(参见第 11 章)来对原始点做去畸变处理，使用 `cvComputeCorrespondEpilines()`来计算极线，然后计算这些点和线的点积(理想情形中，这些点积都为 0)。累计的绝对距离形成了误差。

代码继续往前，选择非标定(Hartley)方法 `cvStereoRectifyUncalibrated()`或者标定(Bouguet)方法 `cvStereoRectify()`来计算校正映射。如果选择了非标定纠正，代码下一步就从头开始计算需要的基础矩阵或者只计使用立体标定得到的基础矩阵。这时就调用 `cvRemap()`来计算校正图像。在我们的例子中，绘制直线穿过图像对，这有助于观察一个好的校正图像是如何排列的。实例的一个结果可参见图 12-12，从图中可以看到，原始图像的桶形畸变主要是从上到下纠正的，并且图像按照水平扫描线对齐。

最后，如果校正了图像，便使用 `cvCreateBMState()`来初始化块状匹配状态(内部分配和参数)。这时候便可以使用 `cvFindStereoCorrespondenceBM()`来计算视差图。这里的代码例子允许你使用水平对齐(从左到右)或者垂直对齐(从上到下)的摄像机；但要注意，对于摄像机垂直对齐的情形而言，如果没有自己添加图像转置的代码，函数 `cvFindStereoCorrespondenceBM()`就只能计算非标定校正的视差。对于摄像机水平对齐的情形，函数 `cvFindStereoCorrespondenceBM()`可以计算标定和非标定的校正立体像对的视差。(参见后面图 12-17 中的实例视差结果)

【445~446】

例 12-3：立体标定、校正、匹配

```
#include "cv.h"
#include "cxmisc.h"
```

① OpenCV(至今还)没有处理极点在图像框内的立体图像校正。对这种情况以及实例的讨论，可参见 Pollefey, Koch, and Gool [Pollefey99b]。

```

#include "highgui.h"
#include "cvaux.h"
#include <vector>
#include <string>
#include <algorithm>
#include <stdio.h>
#include <ctype.h>

using namespace std;

// Given a list of chessboard images, the number of corners (nx, ny)
// on the chessboards, and a flag called useCalibrated (0 for
Hartley
// or 1 for Bouguet stereo methods). Calibrate the cameras and
display the
// rectified results along with the computed disparity images.

static void
StereoCalib(const char* imageList, int nx, int ny, int
useUncalibrated)
{
    int displayCorners = 0;
    int showUndistorted = 1;
    bool isVerticalStereo = false; //OpenCV can handle left-right
                                    //or up-down camera arrangements
    const int maxScale = 1;
    const float squareSize = 1.f; //Set this to your actual square size
    FILE* f = fopen(imageList, "rt");
    int i, j, lr, nframes, n = nx*ny, N = 0;
    vector<string> imageNames[2];
    vector<CvPoint3D32f> objectPoints;
    vector<CvPoint2D32f> points[2];
    vector<int> npoints;
    vector<uchar> active[2];
    vector<CvPoint2D32f> temp(n);
    CvSize imageSize = {0,0};
    // ARRAY AND VECTOR STORAGE:
    double M1[3][3], M2[3][3], D1[5], D2[5];
    double R[3][3], T[3], E[3][3], F[3][3];
    CvMat _M1 = cvMat(3, 3, CV_64F, M1 );
    CvMat _M2 = cvMat(3, 3, CV_64F, M2 );

```

```

CvMat _D1 = cvMat(1, 5, CV_64F, D1 );
CvMat _D2 = cvMat(1, 5, CV_64F, D2 );
CvMat _R = cvMat(3, 3, CV_64F, R );
CvMat _T = cvMat(3, 1, CV_64F, T );
CvMat _E = cvMat(3, 3, CV_64F, E );
CvMat _F = cvMat(3, 3, CV_64F, F );
if( displayCorners )
    cvNamedWindow( "corners", 1 );
// READ IN THE LIST OF CHESSBOARDS:
if( !f )
{
    fprintf(stderr, "can not open file %s\n", imageList );
    return;
}
for(i=0;;i++)
{
    char buf[1024];
    int count = 0, result=0;
    lr = i % 2;
    vector<CvPoint2D32f>& pts = points[lr];
    if( !fgets( buf, sizeof(buf)-3, f ))
        break;
    size_t len = strlen(buf);
    while( len > 0 && isspace(buf[len-1]))
        buf[--len] = '\0';
    if( buf[0] == '#')
        continue;
    IplImage* img = cvLoadImage( buf, 0 );
    if( !img )
        break;
    imageSize = cvGetSize(img);
    imageNames[lr].push_back(buf);
//FIND CHESSBOARDS AND CORNERS THEREIN:
    for( int s = 1; s <= maxScale; s++ )
    {
        IplImage* timg = img;
        if( s > 1 )
        {
            timg = cvCreateImage(cvSize(img->width*s, img-
                >height*s),
                img->depth, img->nChannels );

```

```

        cvResize( img, timg, CV_INTER_CUBIC );
    }

    result = cvFindChessboardCorners( timg, cvSize(nx, ny),
        &temp[0], &count,
        CV_CALIB_CB_ADAPTIVE_THRESH |
        CV_CALIB_CB_NORMALIZE_IMAGE);
    if( timg != img )
        cvReleaseImage( &timg );
    if( result || s == maxScale )
        for( j = 0; j < count; j++ )
    {
        temp[j].x /= s;
        temp[j].y /= s;
    }
    if( result )
        break;
}
if( displayCorners )
{
    printf("%s\n", buf);
    IplImage* cimg = cvCreateImage( imageSize, 8, 3 );
    cvCvtColor( img, cimg, CV_GRAY2BGR );
    cvDrawChessboardCorners( cimg, cvSize(nx, ny), &temp[0],
        count, result );
    cvShowImage( "corners", cimg );
    cvReleaseImage( &cimg );
    if( cvWaitKey(0) == 27 ) //Allow ESC to quit
        exit(-1);
}
else
    putchar('.');
N = pts.size();
pts.resize(N + n, cvPoint2D32f(0,0));
active[lr].push_back((uchar)result);
//assert( result != 0 );
if( result )
{
    //Calibration will suffer without subpixel interpolation
    cvFindCornerSubPix( img, &temp[0], count,
        cvSize(11, 11), cvSize(-1,-1),
        cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS,

```

```

        30, 0.01) );
    copy( temp.begin(), temp.end(), pts.begin() + N );
}
cvReleaseImage( &img );
}
fclose(f);
printf("\n");
// HARVEST CHESSBOARD 3D OBJECT POINT LIST:
nframes = active[0].size(); //Number of good chessboards found
objectPoints.resize(nframes*n);
for( i = 0; i < ny; i++ )
    for( j = 0; j < nx; j++ )
        objectPoints[i*nx + j] =
            cvPoint3D32f(i*squareSize, j*squareSize, 0);
for( i = 1; i < nframes; i++ )
    copy( objectPoints.begin(), objectPoints.begin() + n,
          objectPoints.begin() + i*n );
npoints.resize(nframes,n);
N = nframes*n;
CvMat _objectPoints = cvMat(1, N, CV_32FC3, &objectPoints[0] );
CvMat _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0] );
CvMat _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0] );
CvMat _npoints = cvMat(1, npoints.size(), CV_32S, &npoints[0] );
cvSetIdentity(&_M1);
cvSetIdentity(&_M2);
cvZero(&_D1);
cvZero(&_D2);
// CALIBRATE THE STEREO CAMERAS
printf("Running stereo calibration ...");
fflush(stdout);
cvStereoCalibrate( &_objectPoints, &_imagePoints1,
    &_imagePoints2, &_npoints,
    &_M1, &_D1, &_M2, &_D2,
    imageSize, &_R, &_T, &_E, &_F,
    cvTermCriteria(CV_TERMCRIT_ITER+
        CV_TERMCRIT_EPS, 100, 1e-5),
    CV_CALIB_FIX_ASPECT_RATIO +
    CV_CALIB_ZERO_TANGENT_DIST +
    CV_CALIB_SAME_FOCAL_LENGTH );
printf(" done\n");
// CALIBRATION QUALITY CHECK

```

```

// because the output fundamental matrix implicitly
// includes all the output information,
// we can check the quality of calibration using the
// epipolar geometry constraint: m2^t*F*m1=0
    vector<CvPoint3D32f> lines[2];
    points[0].resize(N);
    points[1].resize(N);
    _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0] );
    _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0] );
    lines[0].resize(N);
    lines[1].resize(N);
    CvMat _L1 = cvMat(1, N, CV_32FC3, &lines[0][0]);
    CvMat _L2 = cvMat(1, N, CV_32FC3, &lines[1][0]);
//Always work in undistorted space
    cvUndistortPoints( &_imagePoints1, &_imagePoints1,
        &_M1, &_D1, 0, &_M1 );
    cvUndistortPoints( &_imagePoints2, &_imagePoints2,
        &_M2, &_D2, 0, &_M2 );
    cvComputeCorrespondEpilines( &_imagePoints1, 1, &_F, &_L1 );
    cvComputeCorrespondEpilines( &_imagePoints2, 2, &_F, &_L2 );
    double avgErr = 0;
    for( i = 0; i < N; i++ )
    {
        double err = fabs(points[0][i].x*lines[1][i].x +
            points[0][i].y*lines[1][i].y + lines[1][i].z)
            + fabs(points[1][i].x*lines[0][i].x +
            points[1][i].y*lines[0][i].y + lines[0][i].z);
        avgErr += err;
    }
    printf( "avg err = %g\n", avgErr/(nframes*n) );
//COMPUTE AND DISPLAY RECTIFICATION
    if( showUndistorted )
    {
        CvMat* mx1 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );
        CvMat* my1 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );
        CvMat* mx2 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );
        CvMat* my2 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );

```

```

CvMat* img1r = cvCreateMat( imageSize.height,
    imageSize.width, CV_8U );
CvMat* img2r = cvCreateMat( imageSize.height,
    imageSize.width, CV_8U );
CvMat* disp = cvCreateMat( imageSize.height,
    imageSize.width, CV_16S );
CvMat* vdisp = cvCreateMat( imageSize.height,
    imageSize.width, CV_8U );
CvMat* pair;
double R1[3][3], R2[3][3], P1[3][4], P2[3][4];
CvMat _R1 = cvMat(3, 3, CV_64F, R1);
CvMat _R2 = cvMat(3, 3, CV_64F, R2);
// IF BY CALIBRATED (BOUGUET'S METHOD)
if( useUncalibrated == 0 )
{
    CvMat _P1 = cvMat(3, 4, CV_64F, P1);
    CvMat _P2 = cvMat(3, 4, CV_64F, P2);
    cvStereoRectify( &_M1, &_M2, &_D1, &_D2, imageSize,
        &_R, &_T,
        &_R1, &_R2, &_P1, &_P2, 0,
        /*CV_CALIB_ZERO_DISPARITY*/ );
    isVerticalStereo = fabs(P2[1][3]) > fabs(P2[0][3]);
//Precompute maps for cvRemap()
    cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_P1,mx1,my1);
    cvInitUndistortRectifyMap(&_M2,&_D2,&_R2,&_P2,mx2,my2);
}
//OR ELSE HARTLEY'S METHOD
else if( useUncalibrated == 1 || useUncalibrated == 2 )
// use intrinsic parameters of each camera, but
// compute the rectification transformation directly
// from the fundamental matrix
{
    double H1[3][3], H2[3][3], iM[3][3];
    CvMat _H1 = cvMat(3, 3, CV_64F, H1);
    CvMat _H2 = cvMat(3, 3, CV_64F, H2);
    CvMat _iM = cvMat(3, 3, CV_64F, iM);
//Just to show you could have independently used F
    if( useUncalibrated == 2 )
        cvFindFundamentalMat( &_imagePoints1,
            &_imagePoints2, &_F);
    cvStereoRectifyUncalibrated( &_imagePoints1,

```

```

        &_imagePoints2, &_F,
        imageSize,
        &_H1, &_H2, 3);
    cvInvert(&_M1, &iM);
    cvMatMul(&_H1, &_M1, &_R1);
    cvMatMul(&iM, &_R1, &_R1);
    cvInvert(&_M2, &iM);
    cvMatMul(&_H2, &_M2, &_R2);
    cvMatMul(&iM, &_R2, &_R2);
//Precompute map for cvRemap()
    cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_M1,mx1,my1);
    cvInitUndistortRectifyMap(&_M2,&_D1,&_R2,&_M2,mx2,my2);
}
else
    assert(0);
cvNamedWindow( "rectified", 1 );
// RECTIFY THE IMAGES AND FIND DISPARITY MAPS
if( !isVerticalStereo )
    pair = cvCreateMat( imageSize.height, imageSize.width*2,
    CV_8UC3 );
else
    pair = cvCreateMat( imageSize.height*2, imageSize.width,
    CV_8UC3 );
//Setup for finding stereo correspondences
CvStereoBMState *BMState = cvCreateStereoBMState();
assert(BMState != 0);
BMState->preFilterSize=41;
BMState->preFilterCap=31;
BMState->SADWindowSize=41;
BMState->minDisparity=-64;
BMState->numberOfDisparities=128;
BMState->textureThreshold=10;
BMState->uniquenessRatio=15;
for( i = 0; i < nframes; i++ )
{
    IplImage* img1=cvLoadImage(imageNames[0][i].c_str(),0);
    IplImage* img2=cvLoadImage(imageNames[1][i].c_str(),0);
    if( img1 && img2 )
    {
        CvMat part;
        cvRemap( img1, img1r, mx1, my1 );

```

```

        cvRemap( img2, img2r, mx2, my2 );
        if( !isVerticalStereo || useUncalibrated != 0 )
        {
// When the stereo camera is oriented vertically,
// useUncalibrated==0 does not transpose the
// image, so the epipolar lines in the rectified
// images are vertical. Stereo correspondence
// function does not support such a case.
        cvFindStereoCorrespondenceBM( img1r, img2r, disp,
            BMState );
        cvNormalize( disp, vdisp, 0, 256, CV_MINMAX );
        cvNamedWindow( "disparity" );
        cvShowImage( "disparity", vdisp );
    }
    if( !isVerticalStereo )
    {
        cvGetCols( pair, &part, 0, imageSize.width );
        cvCvtColor( img1r, &part, CV_GRAY2BGR );
        cvGetCols( pair, &part, imageSize.width,
            imageSize.width*2 );
        cvCvtColor( img2r, &part, CV_GRAY2BGR );
        for( j = 0; j < imageSize.height; j += 16 )
            cvLine( pair, cvPoint(0,j),
                cvPoint(imageSize.width*2,j),
                CV_RGB(0,255,0));
    }
    else
    {
        cvGetRows( pair, &part, 0, imageSize.height );
        cvCvtColor( img1r, &part, CV_GRAY2BGR );
        cvGetRows( pair, &part, imageSize.height,
            imageSize.height*2 );
        cvCvtColor( img2r, &part, CV_GRAY2BGR );
        for( j = 0; j < imageSize.width; j += 16 )
            cvLine( pair, cvPoint(j,0),
                cvPoint(j,imageSize.height*2),
                CV_RGB(0,255,0));
    }
    cvShowImage( "rectified", pair );
    if( cvWaitKey() == 27 )
        break;

```

```

    }
    cvReleaseImage( &img1 );
    cvReleaseImage( &img2 );
}
cvReleaseStereoBMState(&BMState);
cvReleaseMat( &mx1 );
cvReleaseMat( &my1 );
cvReleaseMat( &mx2 );
cvReleaseMat( &my2 );
cvReleaseMat( &img1r );
cvReleaseMat( &img2r );
cvReleaseMat( &disp );
}
}

int main(void)
{
    StereoCalib("list.txt", 9, 6, 1);
    return 0;
}

```

【446~452】

从三维重投影获得深度映射

很多算法直接使用视差映射——比如检测目标是否在工作台之上(或者从工作台伸出)。但是对三维形状匹配、三维模型学习和机器人抓取等应用，我们需要真正的三维重建或者深度映射。幸运的是，我们至今建立的许多立体机制都能使这项任务变得简单。回顾前文介绍的 4×4 的重投影矩阵 Q 。也请回顾，如果给定视差 d 和 2D 点 (x, y) ，我们就能用下面的公式来计算 3D 深度：

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

其中三维坐标就是 $(X/W, Y/W, Z/W)$ 。显然 Q 可以解析出摄像机视线是否收敛以及摄像机基线和两幅图像的主点。因此，我们不需要明确解释收敛或者前向平行摄像机，而只需要简单地通过矩阵相乘就提取深度信息。OpenCV 中有两个函数帮助我们完成这项工作。第一个函数是你之前就熟悉的，操作的是一序列点及其关联视差，函数名称是 `cvPerspectiveTransform`：

```

void cvPerspectiveTransform(
    const CvArr *pointsXYD,
    CvArr* result3DPoints,
    const CvMat *Q
);

```

第二个函数(也是新函数)cvReprojectImageTo3D()是操作整幅图像:

```

void cvReprojectImageTo3D(
    CvArr *disparityImage,
    CvArr *result3DImage,
    CvArr *Q
);

```

这个程序输入的是单通道的 `disparityImage`, 并利用 4×4 重投影矩阵 Q , 将每个像素的 (x, y) 坐标和视差(比如矢量 $[x \ y \ d]^T$)转换到匹配的三维点 $(X/W, Y/W, Z/W)$ 。输出的是大小与输入相同的 3 通道浮点型(捉着 16 为整型)图像。

当然, 这两个函数都允许你传入由 `cvStereoRectify` 计算出的任意投影变换(比如标准转换)或者它们的叠合、任意的 3D 旋转、变换等。

图 12-17 给出了利用 `cvReprojectImageTo3D()` 处理一幅杯子和椅子图像的结果。

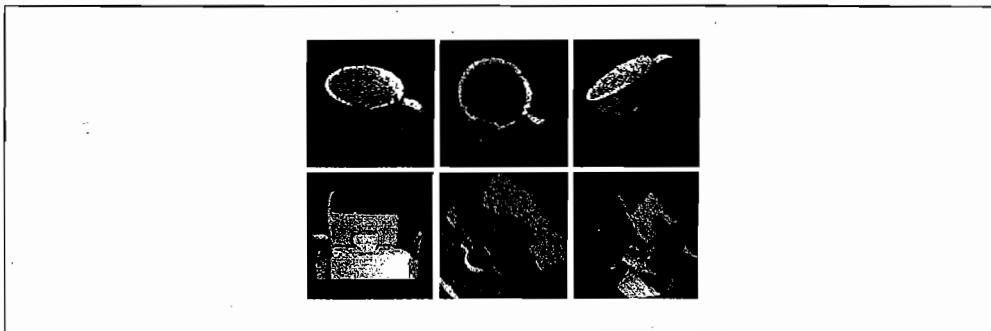


图 12-17: 使用函数 `cvFindStereoCorrespondenceBM()` 和函数 `cvReprojectImageTo3D()`(图像来源于 Willow Garage)计算的深度映射输出实例(对于一个杯子或椅子来说)

来自运动的结构

来自运动的结构是移动机器人技术以及更广泛的视频图像分析, 比如来自手持摄像

机等领域中非常重要的主题。由运动中得到的结构是一个很广的领域，并且在这个领域中，已经做了大量的研究工作。然而，通过简单的观测可以完成更多的内容：在一个静态的场景中，由移动摄像机拍摄的一幅图像和任何另一台摄像机拍摄的图像没有什么不同。所以，所有的直觉以及数学和算法机制，都可以迅速地应用于这种情形中。当然，描述词“静态”是比较苛刻的，但是在很多实际情形中，场景要么是静止的，要么是近似静止的，即场景中只有很少的移动点，这可以利用鲁棒性拟合方法，作为异常点处理。

考虑摄像机在建筑物内移动的情形，如果环境中有相对较多的可识别特征可以被光流技术找到(比如 `cvCalcOpticalFlowPyrLK()`)，这时我们就应该可以计算足够多的点之间——帧到帧——的匹配，来重建摄像机轨迹(此信息通过本征矩阵 E 来表示， E 可以通过基础矩阵 F 和摄像机内参数矩阵 M 来计算获得)、建筑物的全部 3D 结构、建筑物中的上述所有特征的位置。`cvStereoRectifyUncalibrated()` 程序只需要基础矩阵就可以根据缩放比例来计算场景上的基本结构。

二维和三维下的直线拟合

本章中最后一个兴趣主题是常规的直线拟合。直线拟合的出现来自很多原因，也导致很多相关内容。我们在这里选择它来介绍，是因为在分析三维点的时候频繁与直线拟合相关(尽管这里描述的函数也能拟合二维直线)。直线拟合一般使用统计的鲁棒算法。OpenCV 中的直线拟合算法 `cvFitLine()` 适用于需要直线拟合的任何时候：

```
void cvFitLine(
    const CvArr* points,
    int         dist_type,
    double      param,
    double      reps,
    double      aeps,
    float*     line
);
```

【453~455】

点序列可以是 $N \times 3$ 或 $N \times 2$ 的浮点值矩阵(可以是 2D 和 3D 上的点)，也可以是 `cvPointXXX` 结构序列^①。参数 `dist_type` 表示对所有点列进行最小化的距离准

① 这里 XXX 表示诸如 2D32F 或者 3D64f 等可以被替换的字符。

则。(如表 12-3 所示)

表 12-3：用来计算 dist_type 值的距离准则

dist_type 值	准则
CV_DIST_L2	$\rho(r) = \frac{r^2}{2}$
CV_DIST_L1	$\rho(r) = r$
CV_DIST_L12	$\rho(r) = \left[\sqrt{1 + \frac{r^2}{2}} - 1 \right]$
CV_DIST_FAIR	$\rho(r) = c^2 \left[\frac{r}{2} - \log\left(1 + \frac{r}{2}\right)\right], c = 1.3998$
CV_DIST_WELSCH	$\rho(r) = \frac{c^2}{2} \left[1 - \exp\left(\frac{r}{2}\right)^2 \right], c = 2.9846$
CV_DIST_HUBER	$\rho(r) = \begin{cases} r^2/2 & r < c \\ c(r - c/2) & r \geq c \end{cases}, c = 2.9846$

参数 param 用于设定表 12-3 中的参数 C 的值。如果选择使用表中的 C 值，需要把参数 param 设置为 0。在描述直线之后，我们就回到 reps 和 aeps。

参数 line 是存储结果的位置。如果点存在 $N \times 2$ 数组中，line 就是一个指向 4 个浮点数数组的指针(如 float array[4])；如果点存在 $N \times 3$ 数组中，line 就是一个指向 6 个浮点数数组的指针(比如 float array[6])。在第一种情形中，返回值是 (v_x, v_y, x_0, y_0) ，其中 (v_x, v_y) 是和拟合直线平行的归一化向量， (x_0, y_0) 是直线上的一点。类似地，后一种(3D)情形中的返回值是 $(v_x, v_y, v_z, x_0, y_0, z_0)$ ，其中 (v_x, v_y, v_z) 是和拟合直线平行的归一化向量， (x_0, y_0, z_0) 是直线上的一点。给定了直线的形式，精度评价参数 reps 和 aeps 就是：reps 是 x_0, y_0, z_0 估计的请求精度，aeps 是 v_x, v_y, v_z 的角度精度。OpenCV 文档对这两个精度值的推荐值都是 0.01。

cvFitLine() 可以拟合 2D 和 3D 中的直线。因为通常只需要 2D 中的直线拟合，而 OpenCV 中的 3D 技术则变得更重要了(见第 14 章)，我们最后将以一个直线程序来结束，见例 12-4^①。代码中，我们首先假设直线周围有一些 2D 噪声点，然后增加一些和直线无关的随机点(称之为异常点)，最后将这些点拟合成直线并显示出来。cvFitLine() 很好地忽略了离群点，这在测量值被高噪声、传感器故障等破坏的实际应用中非常重要。

【455 ~ 456】

① 感谢 Vadim Pisarevsky 生成的这个实例。

例 12-4：2D 直线拟合

```
#include "cv.h"
#include "highgui.h"
#include <math.h>

int main( int argc, char** argv )
{
    IplImage* img = cvCreateImage( cvSize( 500, 500 ), 8, 3 );
    CvRNG rng = cvRNG(-1);

    cvNamedWindow( "fitline", 1 );

    for(;;) {

        char key;
        int i;
        int count      = cvRandInt(&rng)%100 + 1;
        int outliers   = count/5;
        float a        = cvRandReal(&rng)*200;
        float b        = cvRandReal(&rng)*40;
        float angle    = cvRandReal(&rng)*CV_PI;
        float cos_a    = cos(angle);
        float sin_a    = sin(angle);
        CvPoint pt1, pt2;
        CvPoint* points = (CvPoint*)malloc( count * sizeof(points[0]));
        CvMat pointMat = cvMat( 1, count, CV_32SC2, points );
        float line[4];
        float d, t;

        b = MIN(a*0.3, b);
        // generate some points that are close to the line
        for( i = 0; i < count - outliers; i++ ) {
            float x = (cvRandReal(&rng)*2-1)*a;
            float y = (cvRandReal(&rng)*2-1)*b;
            points[i].x = cvRound(x*cos_a - y*sin_a + img->width/2);
            points[i].y = cvRound(x*sin_a + y*cos_a + img->height/2);
        }

        // generate "completely off" points
        for( ; i < count; i++ ) {
            points[i].x = cvRandInt(&rng) % img->width;
```

```

    points[i].y = cvRandInt(&rng) % img->height;
}

// find the optimal line
cvFitLine( &pointMat, CV_DIST_L1, 1, 0.001, 0.001, line );
cvZero( img );

// draw the points
for( i = 0; i < count; i++ )
    cvCircle(
        img,
        points[i],
        2,
        (i < count - outliers) ? CV_RGB(255, 0, 0) : CV_RGB(255,255,0),
        CV_FILLED, CV_AA,
        0
    );
// ... and the line long enough to cross the whole image
d = sqrt((double)line[0]*line[0] + (double)line[1]*line[1]);
line[0] /= d;
line[1] /= d;
t = (float)(img->width + img->height);
pt1.x = cvRound(line[2] - line[0]*t);
pt1.y = cvRound(line[3] - line[1]*t);
pt2.x = cvRound(line[2] + line[0]*t);
pt2.y = cvRound(line[3] + line[1]*t);
cvLine( img, pt1, pt2, CV_RGB(0,255,0), 3, CV_AA, 0 );

cvShowImage( "fitline", img );

key = (char) cvWaitKey(0);
if( key == 27 || key == 'q' || key == 'Q' ) // 'Esc'
    break;
free( points );
}
cvDestroyWindow( "fitline" );
return 0;
}

```

【456~457】

练习

1. 使用函数 `cvCalibrateCamera2()` 和至少 15 幅棋盘图像来对摄像机进行标定，然后调用函数 `cvProjectPoints2()`，使用由摄像机标定得到的旋转和平移向量，将一个正交箭头投影到每幅图像的棋盘(曲面法线)上。
2. 三维操纵杆。利用一个简单的已知目标，用至少四个不共面的可跟踪的测量点作为 `POSIT` 的输入参数，并以此目标作为一个三维操纵杆在图像上移动小棒图形。
3. 在文中的鸟瞰图实例中，通过平面上空的摄像机水平的对周围成像，我们看到了地平面的单应性有一条水平线，而超过了该水平线，单应性是无效的，怎么使无穷平面上具有水平线？为什么它不会永远出现？
提示：在远离摄像机的平面上的等间隔点序列上绘制直线。摄像机平面到下一点的角度与上一点的角度是如何变化的？
4. 在观测地平面的视频摄像机内应用鸟瞰图。实时运行它，当目标围绕鸟瞰图图像和常规图像移动时会发生什么？
5. 安装两个摄像机或者一台移动摄像机来拍摄两幅图像，然后完成以下练习。
 - a. 计算、存储并检测基础矩阵。
 - b. 重复几次计算基础矩阵，看看计算的稳定性。

6. 如果有一个标定好的立体摄像机，并且跟踪两个摄像机中的移动点，请问你如何使用基础矩阵来获得跟踪误差？
7. 计算和绘制两台摄像机装置的极线来建立立体视觉。
8. 安装两台视频摄像机，应用立体校正，并根据以下条件对深度精度进行实验。
 - a. 在场景中放入一块镜子时会发生什么？
 - b. 改变场景中的纹理并汇报结果。
 - c. 尝试不同的视差方法并汇报结果。
9. 安装立体摄像机，并在手臂上放一些纹理，使用所有的 `dist_type` 方法对手臂进行直线拟合。比较不同方法的精度性和可靠性。

机器学习

什么是机器学习

机器学习^①的目的是把数据转换成信息。在学习了一系列数据之后，我们需要机器能够回答与这些数据有关的问题：其他还有哪些数据和本数据最相似？图像中有没有汽车？哪个广告最能得到消费者的响应？由于消费者经常考虑价格因素，这个问题会变为“在我们销售的所有产品中，如果要做广告，哪个产品会最热销？”机器学习通过从数据里提取规则或模式来把数据转换成信息。

训练集和测试集

机器学习针对的是温度、股票价格、色度等之类的数据。这些数据经常被预处理成特征。举个例子，我们有一个数据库，它是对 10 000 张人脸图像进行边缘检测，然后收集特征，如边缘方向、边缘长度、边缘相对脸中心的偏移度。我们从每张脸中获得一个含有 500 个数据的 500 项特征向量。然后我们通过机器学习技术根据这些特征数据创建某种模型。如果我们仅仅想把这些数据分成不同的类(粗分、细分或其他)，一个“聚类”算法就足够了。如果想学习从人脸的边缘的模式预测他的

① 机器学习是一个非常大的题目，OpenCV 仅仅涉及统计性机器学习，而不包括贝叶斯网络、马尔可夫随机场和图模型等内容。一些比较好的机器学习教材有 Hastie、Tibshirani 和 Friedman 的[Hastie01]，Duda 和 Hart 的[Duda73]，Duda、Hart 和 Stork 的[Duda00]以及 Bishop 的[Bishop07]；如果要了解机器学习问题的并行解决方法，请参考 Ranger 等人的[Ranger07]和 Chu 等人的[Chu07]。

年龄，我们需要一个“分类”算法。为了达到目的，机器学习算法分析我们收集的数据，分配权重、阈值和其他参数来提高性能。这个调整参数来达到目的的过程被称为“学习”(learning)。

【459】

知道机器学习方法的性能很重要，而且这是一个精细的问题。一般情况下，我们把原始数据分成一个大的训练集(我们的例子中，大约 9000 张脸)和一个小的测试集(剩下的 1000 张脸)。我们利用训练集来训练我们分类器，学习年龄预测模型。当训练结束，我们用测试集来测试年龄预测分类器。

测试集不能被用来训练分类器，我们不让分类器知道测试集的年龄标签。我们让分类器对测试集的数据预测，记录预测结果，考察预测年龄和真实年龄的符合程度。如果分类器效果不好，就尝试增加新的特征或者考虑不同类型的分类器。本章将描述多种分类器和多种训练分类器的方法。

如果分类器效果好，我们就有了一个有潜在价值的模型，我们可以在真实世界里应用它。这个系统可能被用来根据用户的年龄来设置视频游戏的行为。人们准备玩游戏的时候，他们的脸的图像将被处理成 500 个特征。这个数据被分类器分类，得到的预测年龄将被用来设置游戏行为。在系统部署之后，分类器看到以前没有见过的脸就会根据学习结果做出判断。

最后，在开发训练分类系统的时候，经常使用验证集。有的时候，在最后测试整个系统是一个太大的工作。我们需要在提交分类器进行最终测试之前调整参数。为此，我们可以把原始的 10 000 个数据分为 3 部分，8000 个训练集，1000 个验证集，1000 个测试集。现在，训练完之后，我们尝试性地提前用验证集来测试分类器的效果。只有对分类器的性能感到满意，才能用测试集来做最后的判断。

监督数据和无监督数据

有的时候，数据没有标签。我们仅仅想根据边缘信息看看人脸倾向于哪个组。有的时候，数据有标签，比如年龄。这类有标签的机器学习是有监督的(即可利用数据特征向量附带的“信号”或“标签”)。如果数据向量没有标签，则表明这种机器学习是无监督的。

有监督的学习可以来判断种类，比如把名字和脸对应起来，或赋予一个数字标签(如年龄)。当数据以名字(种类)作为标签，则表明我们在做分类(classification)；如果输出的是数值，则表明我们在进行回归，回归是通过类别的或数值的输入数据来拟合一个数值输出。

有监督的学习也有一些模糊领域：它包括数据向量和标签的一一匹配，也包括强化学习(deferred learning)。在强化学习中，数据标签(也叫奖励或惩罚)能在单独的数据向量被观察之后存在很久。当一个老鼠走迷宫寻找食物的时候，它经历一系列的转弯，最后得到了食物，这是对它的奖励。奖励必须反馈回去，对老鼠寻找食物的过程产生影响。强化学习也是一样：系统获得一个延迟的信号(奖励或者惩罚)，根据这个信号推断下一步的策略(如在迷宫中的每一步怎么走)。监督学习也适用于不完整标签，此时意味着标签缺损(这种情况下称作半监督学习)，或者噪声标签(标签是错的)。大部分机器学习的方法只能处理以上讨论的一种或两种情况。例如：某算法可以处理分类，但不可以处理回归；某个可以处理半监督学习，但不能进行强化学习；某算法可以处理数字数据，但是不可以处理类别数据；等等。

【460~461】

有的时候，数据是没有标签的，但我们对数据是否能自然归类感兴趣。这种无监督的学习算法叫聚类算法(clustering algorithm)。目的是把这些无标签的数据按照它们的距离(预先确定的或者感觉上的)远近归类。我们仅仅想看脸是怎么分布的：它们是胖是瘦，是长是方？如果我们观察癌症方面的数据，能否将拥有不同化学信号的癌症方面的数据聚集在一起？无监督的聚类数据经常形成一个特征向量供更高层的有监督的分类器使用。我们可以先把脸聚类成脸型(宽，窄，长，方)，然后把脸型作为输入，可能再加上其他的一些数据(声音的平均频率)，以此预测人的性别。

这两个典型的机器学习任务(分类和聚类)与计算机视觉中两个基本任务(识别和分割)有共通之处。我们需要计算机判断物体是什么的时候，我们使用识别；我们需要计算机说出物体的位置的时候，我们使用分割。因为计算机视觉大量依靠机器学习，OpenCV 在 ML 部分包含大量的机器学习算法，库位于`.../opencv/ml` 目录之中。

注意：OpenCV 机器学习的代码是通用的。尽管在视觉应用中经常使用此部分的代码，代码的应用范围并不局限于视觉。例如可以某些函数来研究染色体序列。当然，我们的关注重点还是物体识别，其中特征向量是从图像中提取的。

生成模型和判别模型

很多算法被设计用于解决学习和聚类问题。OpenCV 囊括了一些当前用得最多的统计机器学习方法。概率机器学习方法，像贝叶斯网络和图模型，在 OpenCV 中支持得比较少，部分原因是这些算法还很新，还在成长中，变化会很大。OpenCV 倾向于支持判别算法，即通过给定数据来判断类别($P(L|D)$)，而不倾向于产生式算

法，产生式算法是通过给定类别来生成数据的分布($P(D|L)$)。虽然这二者的区别不是非常清晰，但是判别模型在根据给定的数据做出预测上有优势，而生成模型则是在为你提供更强大的数据表达或者有条件地生成新数据时有优势(比如，想象一个大象的样子；你是在根据条件“大象”来生成数据)。

【461~462】

产生式模型很容易理解，是因为它是对数据的产生进行建模。判别式学习算法经常根据一些看似武断的阈值做出判断。例如：假设一小片区域在某个场景中被定义为路仅仅因为它的红色分量小于 125。但是是否意味着红色分量等于 126 就不是路？这类问题很难解释的清楚。产生式模型中经常给定类别，得到数据的条件分布，所以你可以很容易看出这个分布跟真正的分布是否相似。

OpenCV 机器学习算法

OpenCV 中的机器学习算法如表 13-1 所示。除了 Mahalanobis 和 K 均值算法在 CXCORE 库中，人脸检测算法在 CV 库中，其他算法都在 ML 库中。

表 13-1：OpenCV 支持的机器学习算法，算法描述后为参考文献

算法名称	简要描述
Mahalanobis	通过除以协方差来对数据空间进行变换，然后计算距离。如果协方差矩阵是单位矩阵，那么该度量等价于欧氏距离 [Mahalanobis36]
K 均值	这是一种非监督的聚类方法，使用 K 个均值来表示数据的分布，其中 K 的大小由用户定义。该方法跟期望最大化方法(expectation maximization)的区别是 K 均值的中心不是高斯分布，而且因为各个中心竞争去“俘获”最近的点，所以聚类结果更像肥皂泡。聚类区域经常被用作稀疏直方图的 bin，用来描述数据。该方法由 Steinhaus [Steinhaus56]发明，并由 Lloyd [Lloyd57]推广使用
正态/朴素贝叶斯分类器	这是一个通用的分类器，它假设特征是高斯分布而且统计上互相独立。这个假定过于苛刻，在很多条件下不能满足，正因为此，它也被称作“朴素贝叶斯”分类器。然而，在许多情况下，这个分类器的效果却出奇得好。该方法最早出自[Maron61; Minsky61]
决策树	这是一个判别分类器。该树在当前节点通过寻找数据特征和一个阈值，来最优划分数据到不同的类别。处理流程是不停地划分数据并向下到树的左侧子节点或右侧子节点。虽然它一般不具有最优性能，但是往往是测试算法的第一选择，因为它速度比较快，

续表

算法名称	描述
决策树	而且具有不错的功能[Breiman84]
Boosting	多个判别子分类器的组合。最终的分类决策是由各个子分类器的加权组合来决定。在训练时，逐个训练子分类器，且每个子分类器是一个弱分类器(只是优于随机选择的性能)。这些弱分类器由单变量决策树构成，被称作“树桩”。在训练时，“树桩”不仅从数据中学习分类决策，而且还根据识别精度学习“投票”的权重。当逐个训练分类器的时候，数据样本的权重被重新分配，使之能够给予分错的数据更多的注意力。训练过程不停地执行，直到总错误(加权组合所有决策树组成的分类器产生的错误)低于某个已经设置好的阈值。为了达到好的效果，这个方法通常需要很大数据量的训练数据[Freund97]
随机森林	这是由许多决策树组成的“森林”，每个决策树向下递归以获取最大深度。在学习过程中，每棵树的每个节点只从特征数据的一个随机子集中选择。这保证了每棵树是统计上不相关的分类器。在识别过程中，将每棵树的结果进行投票来确定最终结果，每棵树的权重相同。这个分类方法经常很有效，而且对每棵树的输出进行平均，可以处理回归问题[Ho95][Breiman01]
人脸检测/Haar 分类器	这个物体检测方法巧妙地使用了 boosting 算法。OpenCV 提供了正面人脸的检测器，它的检测效果非常好。你也可使用 OpenCV 提供的软件训练算法，使之能够检测其他物体。该算法非常擅长检测特定视角的刚性物体[Viola04]
期望最大化 (EM)	一种用于聚类的非监督生成算法，它可以拟合 N 个多维高斯到数据上，此处 N 的大小由用户决定。这样仅仅使用几个参数(均值和方差)就可以非常有效的表达一个比较负责的分布。该方法经常用于分科。它与 K 均值的比较可参考文献[Dempster77]
K 近邻	K 近邻可能是最简单的分类器。训练数据跟类别标签存放在一起，离测试数据最近的(欧氏距离最近) K 个样本进行投票，确定测试数据的分类结果。这可能是你想到的最简单的方法，该方法通常比较有效，但是速度比较慢且对内存的需求比较大[Fix51]
神经网络/多层 传感器	该分类算法在输入节点和输出节点之间具有隐藏节点，这可以更好的表示输入信号。训练该分类器很慢，但是识别时很快。对于一些识别问题，如字符识别，它具有非常不错的性能[Werbos74; Rumelhart88]

续表

算法名称	描述
支持向量机(SVM)	它可以进行分类，也可以进行递归。该算法需要定一个高维空间中任两点的距离函数。(将数据投影到高维空间会使数据更容易地线性可分。)该算法可以学习一个分类超平面，用来在高维空间里实现最优分类。当数据有限的时候，该算法可以获得非常好的性能；而 boosting 和随机森林只能在拥有大量训练数据时才有好的效果[Vapnik95]

【462~463】

视觉中使用机器学习

基本上，表 13-1 的所有算法都使用特征构成的数据向量作为输入，特征的个数可以以千计。假设你的任务是识别特定的物体(例如，人)。你遇到的第一个问题是怎么样采集数据并给数据定标签(场景里有人还是没有人)。你会发现人以不同的方式出现：人的图像可能只包含一点点像素，或者你只能看到一只耳朵出现在整个屏幕上。更坏的情况下，人可能处于遮挡状态下：在车中的人，女人的脸，树后面的一条腿。你需要定义有人在场景中的确切含义。

下面将遇到采集数据的问题。是用自己的摄像机采集，还是使用公用数据库 <http://www.flickr.com> 尝试寻找“人物”标签，或者两种方法都使用？采集运动信息吗？采集其他信息吗？如场景中的门是开的，时间，季节，温度。用于海滩上检测人的算法说不定不适用于滑雪场。需要捕获数据的各种变化：不同视角，不同光线，不同天气，阴影，等等。

获取了很多数据之后，该怎么给数据定标签？首先需要决定标签的意义。需要知道场景中的人在什么位置吗？运动信息重要吗？如果有大量的图像，又应该怎样给所有的图像定标签？有很多诀窍，比如在约束情况下中做背景剪除，收集运动前景。你可以雇人帮你分类，如可以通过 Amazon 的“mechanical turt”网站 (<http://www.mturk.com/mturk/welcome>) 雇人帮你定标签。如果安排的工作很简单，可以把价格降到每个标签一分钱。

数据定好标签之后，需要决定需要提取哪些特征。再一次，必须知道自己的目的。如果人们总是正面出现，就没有必要使用旋转不变的特征，也没有必要预先旋转物体。总之，必须寻找表达物体固有属性的特征，比如梯度直方图、色彩、或目前流

形的 SIFT 特征^①。如果有背景信息，可能想首先把背景去除，提取出物体；然后进行图像处理(归一化图像、尺度改变、旋转、直方图均衡)，计算很多特征。物体的特征向量将与物体的标签对应。

一旦数据被转换特征向量，便需要把数据分成训练集、验证集和测试集。在这里，我推荐交叉使用训练集、验证集和测试集。也就是说：所有的数据分成 K 个子集，然后每次随机选取其中一部分来训练；剩余的用来测试^②。测试结果求平均，得到最终的性能结果。使用交叉验证，我们能够更清楚地看到处理异常数据时分类器的性能。

【463 ~ 465】

数据已经准备好，下面是选择分类器。一般分类器的选择需要考虑计算速度、数据形式和内存大小。一些应用中，在线用户优先选择建模，所以分类器需要能够快速完成训练。在这种情况下，最邻近算法、正态贝叶斯和决策树是不错的选择。如果需要考虑内存因素，决策树和神经网络是理想的选择。如果不需要很快训练，而需要很快判断，那么神经网络可以满足要求，正态贝叶斯和 SVM 也不错。如果不需要训练很快，但是需要精确度很高，可选择 boosting 和随机森林。如果选取的特征比较好，仅仅需要一个简单易懂的分类器，就选择决策树和最邻近算法。要获得最好的性能，还是离不开 boosting 和随机森林算法。

注意：不存在“最好”的分类器（请参考 http://en.wikipedia.org/wiki/No_free_lunch_theorem）。综合考虑所有的数据分布类型，所有的分类器都是一样的。因此我们无法清楚说明表 13-1 中的算法哪个“最好”。但是如果给定某个特定的数据分布，或者特定的一些数据分布，通常存在一个最好的分类器。所以在实际应用中，最好多尝试一下各种不同的分类器。考虑一下你的目的：只是为了得到正确的匹配分数，还是去理解数据？你追求的是快速计算，少的内存需求，还是结果的可信度？在这些方面，不同的分类器有不同的能力。

变量的重要性

表 13-1 中有两个算法可用来评估变量的重要性。拿到一个特征向量，我们怎么才知道哪个特征对分类器的准确性有较大贡献？二进制决策树可以解决这个问题：通过在每个节点选择最能够分裂出数据的变量。最上面的变量是最重要的变量，第二层的变量第二重要，以此类推。随机森林采用 Leo Breiman 发明的一项技术来测量变量的重要性。这个技术可以用于任何分类器，但是在 OpenCV 中，仅仅决策树

① 请参考 Lowe 的 SIFT 特征 demo(<http://www.cs.ubc.ca/~lowe/keypoints/>)。

② 一般进行 5~10 次训练(或验证)和测试。

和随机森林实现了这个技术。

变量重要性的一个用途是减少分类器需要考虑的特征的个数。一开始有很多特征，你训练分类器，发现其中一个特征与另外一些特征有关联。你可以去掉其中一些不重要的特征。排除不重要的特征可以提高速度(因为不需要处理不重要的特征)，可以使训练和测试更快。还有，如果没有足够的数据，去除不重要的变量可以提高分类器的准确率。这也可以使处理更快，效果更好。

【465~466】

Breiman 的变量重要性算法的步骤如下。

1. 用训练集训练一个分类器。
2. 使用验证集或测试集来确定分类器的准确率。
3. 对于每一个数据样本和一个选择的特征，随机选择其他特征数据中的该特征的值来替代(替补抽样法)。这样可以保证特征的分布与原始数据集的一样，但是特征的结构或者意义被抹去了(因为它的值是从剩余的数据中随机选择的)。
4. 用改变后的训练集训练分类器，然后用改变后的测试集或者验证集来评价分类器。如果完全打乱特征使正确率降低很多，那么这个特征很重要。如果完全打乱特征并没有使正确率下降多少，那么这个特征并没有多重要，可以考虑删除。
5. 把原始数据重新换一个特征按照 3 和 4 操作，直到所有的特征全部分析完毕，得到的结果就是每个特征的重要性。

这个过程被内置到随机森林和决策树中，所以可以使用它们来确定使用哪些变量做特征，然后使用被删减的特征向量来训练分类器。

诊断机器学习中的问题

将机器学习用好，不仅仅是一门技术，更是一门艺术。算法经常“有些时候”能用，但又不能完全与要求一致。这就是艺术的所在；需要指出哪些地方出问题了并解决问题。尽管在这里我不能详细说明，但我将列举出一些常见问题^①。首先介绍一些重要的规律：大量数据比少量数据好；好的特征比好的算法更重要。如果选取的特征好，最大化它们的独立性，最小化它们在不同环境之下的变化，那么大部分算法都可以获得比较好的效果。除此之外，还有两个经常遇到的问题。

- 欠拟合 模型假设太严格，所以模型不能拟合到实际数据上。

^① 斯坦福大学的 Andrew Ng 教授在他的网上讲稿“Advice for Applying Machine Learning”给出了详细介绍(<http://www.stanford.edu/class/cs229/materials/ML-advice.pdf>)。

- **过拟合** 算法不仅学习了数据，而且把噪声也当作信号学习了，这样算法的推广能力很差。

图 13-1 为统计类机器学习的基本结构。我们的目的是根据输入和输出对判别函数 f 建模。这个函数可以是一个回归问题，也可以是一个分类预测问题。现实生活中的问题中，噪声和没有考虑过的影响会导致观测结果与理论结果不同。例如，在人脸识别中，我们学习一个模型，它通过人眼、嘴、鼻子的标准距离来定义人脸。但是附近闪烁的光线变化会在测量中引入噪声，或者一个工艺不好的摄像机镜头会在测量中引入系统畸变。这些都是模型中没有考虑的。这些干扰会影响精确度。

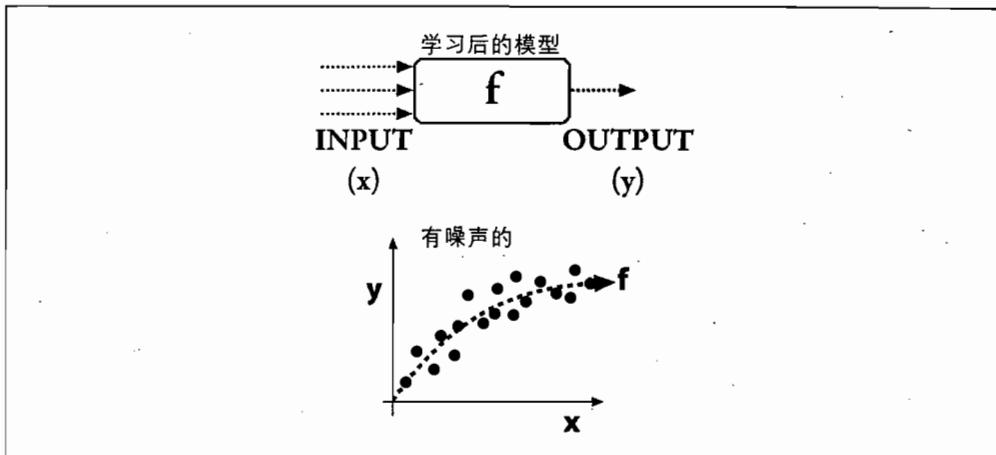


图 13-1：统计机器学习的结构。训练一个分类器来拟合一个数据集，真正的模型 f 会被噪声和一些未知的因素“淹没”

图 13-2 中的上两幅图分别展示了数据的欠拟合和过拟合，下两幅图展示了对应的最终检测错误率与训练集大小的关系。在图 13-2 左部，我们尝试对图 13-1 的底部的数据进行训练和预测。如果使用很严格的模型(粗虚线)，我们就不能很好地拟合真实的抛物线函数(细虚线)。因此，尽管有大量的数据，训练集和测试集的拟合程度都不好。在这个例子中，因为欠拟合，训练集和测试集都无法预测得很好。在图 13-2 右部，我们的训练严格符合训练集，但这使函数包含了噪声。同样，测试的结果不好。尽管训练错误率很低，但是测试错误率也很高，说明存在过拟合。

有些时候，需要注意是否正在解决自己想要解决的问题。如果训练和测试结果都很好，但是算法在实际应用中效果不好，则表明数据集可能是从非实际条件下获得的，也许是因为这些条件可使收集式模拟这些数据更容易。如果算法不能重现训练集或者测试集，表明问题出在算法，或者从数据中提取的特征是无效的，或者有用的信息不在收集的数据中。表 13-2 列出了我们提到的一些可能的解决方法。当

然，这并不是所有的解决方案。为了使机器学习方法发挥作用，必须仔细思考并设计应该采集哪些数据，计算哪些特征。机器学习方法出现问题时，还必须做一些系统开发级的诊断。

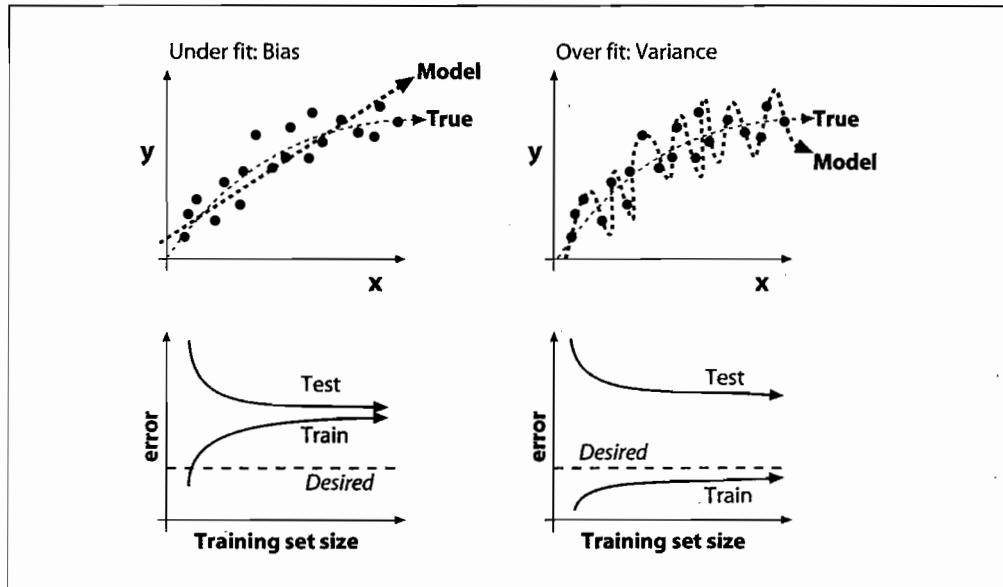


图 13-2：机器学习中比较差的拟合模型以及在训练和测试时对性能的影响，而真正的模型函数是上图中的细虚线：欠拟合模型(左上图)在训练数据和测时数据上都有很大的误差，而过拟合模型(右上图)的在训练数据上的误差比较小，但是在测试数据上的误差很大

表 13-2：机器学习中遇到的问题以及可能的解决方法，采用更好的特征对任何问题都有帮助

问题	可能的解决方案
欠拟合	使用更多的特征有利于拟合 选用一个学习能力更好的拟合算法
过拟合	增加训练数据的数量可使得拟合曲线更光滑 减少特征的数量可降低过拟合程度 使用一个学习能力差一点的算法
训练和测试比较好，但实际应用效果不好	采集更加真实的数据

续表

问题	可能的解决方案
模型无法学习数据	重新选择特征，使特征更能表达数据的不变特性
	采集更新、更相关的数据
	选用一个学习能力更好的拟合算法

【466~468】

交叉验证、自抽样法、ROC 曲线和混淆矩阵

最后，介绍一些用来评估机器学习结果的工具。在监督学习中，最基本的问题之一是知道算法的性能：分类器准确吗？你可能会说“简单，我只需要在测试集或者验证集上运行一下，就可以获得结果。”但是在实际情况中，我们必须考虑噪声，采样误差和采样错误。测试集或验证集可能并不能精确地反映数据的实际分布。为了更准确评估分类器性能，我们可以采用交叉验证(cross-validation)的技术或者与之比较相近的自抽样法(bootstrapping)^①。

交叉验证首先把数据分为 k 个不同的子集。然后用 $k-1$ 个子集进行训练，用没有用来训练的子集进行测试。这样做 k 次(每个子集都有一次机会作为测试集)，然后把结果平均。

自抽样法跟交叉验证法类似，但是验证集是从训练集中随机选取的。选择的点仅用于测试，不在训练中使用。这样做 N 次，每次随机选择一些验证集，最后把得到的结果平均。这意味着一些数据样本会出现在不同的验证集中，自抽样法的效果一般胜于交叉验证。

使用其中任意一种技术都可以使实际效果的评估更准确，更高的精确度可以用来指导学习系统的参数调整。再训练、再评估、再调整，如此继续。

在评估和调整分类器的方法中，还有两个比较有用的方法，它们是画 ROC (receiver operating characteristic)曲线图和填充混淆矩阵，请参考图 13-3。ROC 曲线评估了分类器参数的变化对分类器性能的影响。假设这个参数是一个阈值。更具体一点，假设我们需要在图中识别黄色的花，探测器使用一个阈值来定义黄色。黄色的阈值设置得非常高，则意味着我们的分类器的错误识别率为 0，同时将正确识别率也为 0(图 13-3 的左下部分)。另一方面，如果黄色的阈值设置得非常低，则意味着所有的信号都被识别成黄色。分类器的错误识别率为 100%，同时正确识别率

① 关于此类方法的更多信息，请参考 What Are Cross-Validation and Bootstrapping? 网址为 <http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-12.html>。

也为 100%(图 13-3 的右上部分)。最好的 ROC 曲线是从原点沿 Y 轴上升到 100%，再沿 X 轴到达右上角的折线。如果做不到这样，那么，曲线越靠近左上角越好。我们可以计算 ROC 曲线下方的面积和整个 ROC 图的面积之比，越接近 1，就表明分类器越好。

【469】

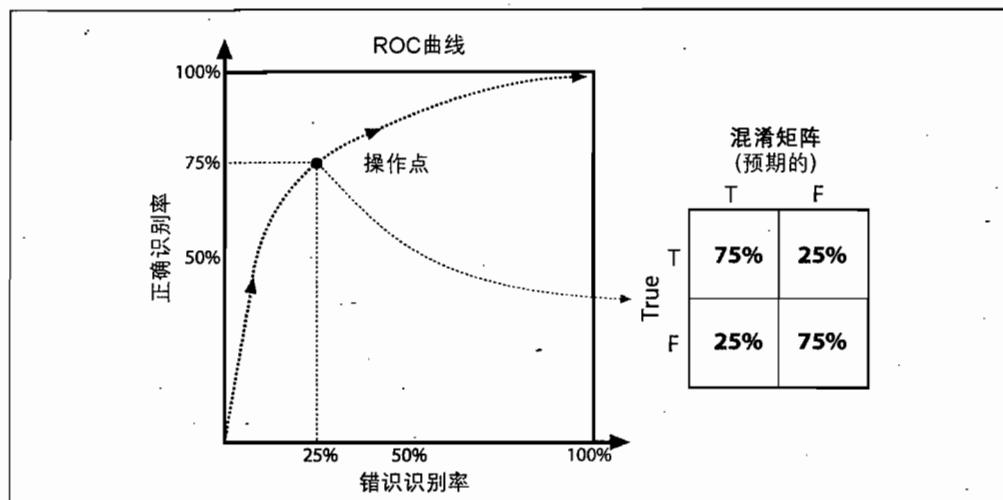


图 13-3：ROC 曲线和与之相关的混淆矩阵：ROC 曲线显示分类器参数在所有取值范围内变化时，正确识别率和错误识别率的关系；混淆矩阵显示错误识别率和错误拒绝率的关系

图 13-3 还展示了一个混淆矩阵。这是一个包含错误识别率、正确识别率、错误拒绝率、正确拒绝率的图。这是评估分类器性能的快速方法：理想情况下，我们将看到左上-右下的对角线上的两个值是 100%，而其他的两个值是 0%。如果我们的分类器可以处理多类问题(多层感知神经网络和随机森林)，那么混淆矩阵将推广到多类，你需要跟踪每一个带标签的数据样本所被分配的类别。

分错类的代价 下面我们讨论分错类的代价。假设我们要设计分类器来检测毒蘑菇，我们希望错误拒绝率(可食用的蘑菇被认为有毒)很大，错误识别率(有毒的蘑菇却被认为可食用)很小。ROC 曲线可以处理这种情况；我们设置 ROC 的参数使操作点(operation point)下降——靠近图 13-3 的左下角。还有一种方法是生成 ROC 曲线的时候对错误识别赋更大的权值。可以把每个错误识别的权值设为错误拒绝权值的 10 倍^①。一些 OpenCV 算法(如决策树和 SVM)，能够通过指定类别的先验概

① 有两类错误的相对代价将会很有用。例如在超市的收银台处将产品分错类会更容易纠正。

率或者指定个别的训练样本的权重，来自动平衡“击中率和误报率”。

【470~471】

特征的方差不一致 另一个训练分类器的问题是特征向量中，特征的方差不一致。例如，如果一个特征是由小写的 ASCII 码字母表示的，那么它最多有 25 个值。与之不同的是，显微镜载物片上的生物细胞数目是以亿计的。类似 k 均值算法的一类算法会认为第一个特征相比于第二个特征可以忽略。解决这个问题的方法是预处理每个特征变量，使它们的方差一致。如果特征不相关，这个步骤很重要；如果特征相关，你可以用它们的协方差或平均方差来归一化。决策树等一类算法不受方差不一致的影响^①，所以不需要预先处理。如果算法以距离度量为准则，就需要预先将方差归一化。我们可以利用 Mahalanobis 距离来归一化特征的方差，这在本章的后面会提到^②。

我们现在正式开始讨论 OpenCV 支持一些机器学习算法，这些算法的大部分都在路径 `.../opencv/ml` 下。我们从 ML 库最基础的一些类开始。

ML 库的通用类

本章的目的是使读者开始使用机器学习算法。即使已经比较了解各种方法，也需要参考 `.../opencv/docs/ref/opencvref_ml.htm` 手册，或者访问 OpenCV 在线维基文档 (<http://opencv.willowgarage.com/>)。因为这个库还在发展，通过手册或者在线维基文档可以了解更多更新的内容。

ML 库中^③所有的程序都是用 C++ 写的，它们都继承于 `CvStatModel` 类。`CvStatModel` 类包含所有算法都需要的一些函数，如表 13-3 所示。注意，`CvStatModel` 中有两套方法来对磁盘进行模型的读/写操作：保存操作的 `save()` 和 `write()`，读操作的 `load()` 和 `read()`。对于机器学习的模型，应该使用简单的 `save()` 和 `load()`，它们实际上把复杂的 `write()` 和 `read()` 函数进行了封装，能够从硬盘读/写 XML 和 YAML 文件。`CvStatModel` 中还有机器学习中两个最重要的方法，即 `predict()` 和 `train()`，它们的形式随算法变化，放到以后讨论。

【471~472】

-
- ① 因为每个变量都有与之对应的独立阈值，所以决策树不受特征方差不一致的影响。换而言之，只有有一个很清晰的区分界限，变量的范围大小是无所谓的。
 - ② 熟悉机器学习或者信号处理的读者可以将视为“白化滤波”。
 - ③ 请注意 Haar 分类器、Mahalanobis 和 k 均值算法是在 ML 库创建之前实现的，所以目前在 `cv` 和 `cxcore` 库中。

表 13-3：ML 库中的算法的基类

基类 CvStatModel 上的成员方法	描述
save(const char* filename, const char* name = 0)	将训练出的模型以 XML 或 YAML 格式保存到硬盘。此方法用于存储
load(const char* filename, const char* name=0) ;	首先需要调用 clear() 函数，然后通过此函数装载 XML 或 YAML 格式的模型。此方法用于调用
clear()	释放所有内存。为重用做准备
bool train(-data points-, [flags] -responses-, [flags etc]) ;	此训练函数从输入的数据集上学习一个模型。算法不同时，训练函数的输入参数也有所不同
float predict(const CvMat* sample [,<prediction_params>]) const;	训练之后，使用此函数来预测一个新数据样本的标签或者值
构造函数、析构函数：	
CvStatModel(); CvStatModel(const CvMat* train_data ...) ;	默认构造函数，以及将创建并训练模型功能集成的构造函数
CvStatModel::~CvStatModel();	机器学习模型的析构函数
读写支持(实际应用时请使用 save 和 load 函数)	
write(CvFileStorage* storage, const char* name) ;	通用的写 CvFileStorage 结构的函数，位于 cxcore 库中(请参考第 3 章)，在此处被 save() 函数调用

续表

基类 CvStatModel 中的函数	描述
read(CvFileStorage* storage, CvTreeNode* node) ;	通用的读 CvFileStorage 结构的函数，位于 cxcore 库中，在此处被 load() 函数调用

训练

训练函数的原型如下。

```
bool CvStatModel::train(  
    const CvMat* train_data,  
    [int tflag,] ...,  
    const CvMat* responses, ...,  
    [const CvMat* var_idx,] ...,  
    [const CvMat* sample_idx,] ...,  
    [const CvMat* var_type,] ...,  
    [const CvMat* missing_mask,]  
<misc_training_alg_params> ...  
) ;
```

【472~473】

机器学习的 train() 方法根据具体的算法呈现不同的形式。所有的算法都以一个指向 CvMat 矩阵的指针作为训练数据。矩阵必须是 32FC1(32 位浮点单通道)类型。CvMat 结构可以支持多通道矩阵，但是机器学习只采用单通道矩阵。一般情况下，这个矩阵以行排列数据样本，每个数据样本被表示为一个行向量。矩阵的每一列，表示某个变量在不同的数据样本中的不同取值，这样就组成了一个二维的数据矩阵。请时刻牢记输入数据矩阵的构成方式(行，列)=(数据样本，特征变量)。但是，有的算法可以直接处理转置后的输入矩阵。针对这些算法，可以使用参数 tflag 向算法表明训练数据是以列排列的。这样便不需要转置一个大的数据矩阵，如果算法可以处理行排列和列排列两种情况，可以使用下面的标记。

- tflag = CV_ROW_SAMPLE 特征向量以行排列(默认)
- tflag = CV_COL_SAMPLE 特征向量以列排列

读者可能会问：如果我的数据不是浮点型，而是字符或者整型，怎么办？答案是：只要在写入 CvMat 的时候，把它们转换成 32 位浮点数。如果特征或标签是字符，

就把 ASCII 字符转换成浮点数。整型也一样。只要转换是唯一的，就没有问题——但是请记住，有些算法对特征的方差不一致比较敏感。最好先归一化特征的方差。基于树的算法(决策树，随机森林和 boosting)支持类别变量和数值变量，OpenCV ML 库中的其他算法只支持数值输入。使数值输入的算法能够处理类别数据的常用方法是把类别数据表示成 0-1 编码。例如，如果输入的颜色变量有 7 个不同的值，则把它们替换成 7 个二进制变量，变量有且仅有一位是 1。

返回的参数 `responses` 可以是类别标签(如蘑菇分类中的“有毒”和“无毒”标签)，也可以是连续的数值(如温度计测得的人体温度)。返回值常常是一个一维向量，向量的每个值对应一个输入数据样本；但是神经网络例外，神经网络的每个数据样本都返回一个向量。返回值有两种类型：对于分类问题，类型是整型(32SC1)，对于回归问题，类型是 32 位浮点型(32FC1)。显然，一些算法只能处理分类问题，一些算法只能处理回归问题，一些算法两类问题都能处理。在最后一种情况下，输出变量的类型要么以一个单独的参数返回，要么以向量 `var_type` 的最后一个元素返回。可以如下设置：

- **CV_VAR_CATEGORICAL** 输出的值是离散类型标签
- **CV_VAR_ORDERED (= CV_VAR_NUMERICAL)** 输出的值是数值类型标签，就是说，不同的值可以作为数来比较，这是一个回归问题 【472~474】

输入数据的类型也可以用 `var_type` 指定。但是，回归类型的算法只能处理有序输入。有的时候，需要把类别数据一一映射到有序数据，但是这样做有的时候会出问题，因为“伪”有序数据会因物理基础的缺乏而可能大幅振荡。

ML 库中的很多模型可以选择训练特定数据子集和/或特定特征子集。为了方便用户，`train()` 方法包含了向量参数 `var_idx` 和 `sample_idx`。默认值为 `NULL`，即使用所有的数据。可以用它们来指定使用哪些数据样本和使用哪些特征。这两个向量要么是单通道整型(CV_32SC1)向量，要么是单通道 8 位数据(CV_8UC1)——非 0 代表有效。读入一堆数据却只想使用一部分数据来训练，一部分数据来分类，不想把数据分成 2 个矩阵的时候，参数 `sample_idx` 尤其有用。

另外，一些算法可以处理数据丢失的情况。例如，当用户使用的是人工采集的数据的时候，很有可能会丢失测量数据。这种情况下，参数 `missing_mask`——一个与 `train_data` 同样大小的 8 位矩阵——将被用来标记丢失的数据(非 0 的标记元素)。一些算法不能处理数据丢失，所以丢失的数据样本必须由用户采用插值的方法合成，或者在训练中抛弃不完整的数据样本。其他算法(像决策树和朴素贝叶斯)通过不同的方法来处理数据丢失。决策树使用“代理分裂”，朴素贝叶斯推断数据值。

通常在训练之前，前面提到的模型都可以用 `clear()` 来清除。但有些算法可以随时用新数据更新模型，而不是每次训练都从头处理所有数据。

【474】

预测

指定训练函数 `train()` 中使用哪些特征的参数 `var_idx` 将被程序记住。调用预测函数 `predict()` 的时候，并随后被用来从输入参数中抽取出需要的部分数据。`Predict()` 函数的基本形式如下：

```
float CvStatMode::predict(
    const CvMat* sample
    [, <prediction_params>]
) const;
```

这个函数可对新输入的数据向量进行预测。如果是分类问题，它会返回一个类别标签。如果是回归问题，它会返回一个数值标签。注意，输入的样本格式必须和训练样本的格式完全一样。可选参数 `prediction_params` 由算法指定，可以用来在基于树的预测方法中指定缺失的特征的值等。函数后缀 `const` 告诉我们：预测不影响模型的内部状态。因此，这个方法可以并行调用。它可以用在网络服务器从多客户端获取图像之中，也可以用在机器人加速场景扫描之中。

训练迭代次数

尽管迭代控制结构体 `CvTermCriteria` 已经在其他章谈论过，但由于它被广泛应用于很多机器学习算法中，所以这里再重复一下。

```
typedef struct CvTermCriteria {
    int type; /* CV_TERMCRIT_ITER and/or CV_TERMCRIT_EPS */
    int max_iter; /* maximum number of iterations */
    double epsilon; /* stop when error is below this value */
}
```

整型参数 `max_iter` 设置了算法最大的迭代次数。参数 `epsilon` 设置了误差停止标准，当误差在此参数之下，则算法停止。最后，参数 `type` 设置了应该使用哪个标准，也可以同时使用 (`CV_TERMCRIT_ITER | CV_TERMCRIT_EPS`)。`term_crit.type` 可以使用的值定义如下：

```
#define CV_TERMCRIT_ITER 1  
#define CV_TERMCRIT_NUMBER CV_TERMCRIT_ITER  
#define CV_TERMCRIT_EPS 2
```

现在我们转向 OpenCV 中具体的算法。我们首先讨论最常用的 Mahalanobis 距离矩阵，然后深入介绍一个无监督算法(K 均值)；这两个算法在 cxcore 库中。然后我们讨论机器学习库中的正态贝叶斯分类器和决策树算法(决策树，boosting，随机森林，Haar 级联)。对于其他的函数，我们仅给出简短描述和用法示例。 【475】

Mahalanobis 距离

Mahalanobis 距离是数据所在的空间的协方差的度量，或者是认为把数据所在空间进行“扭曲拉伸”然后进行度量。如果你知道 Z-score，就可以把 Mahalanobis 距离看作多维空间中 Z-score 的类似物。图 13-4(a)展示了三个数据集的初始分布，看起来竖直方向上的那两个集合比较接近。在我们根据数据的协方差归一化空间之后，如图 13-4(b)，实际上水平方向上的两个集合比较接近。这种情况经常发生；如果我们以米为单位来测量人的身高，以天为单位测量人的年龄，我们看到身高的范围很小，而年龄的范围很大。通过方差归一化，变量之间的关系便会更加符合实际情况。像 K 邻近之类的算法很难处理方差相差很大的数据，但是决策树等其他算法则不受这种情况影响。

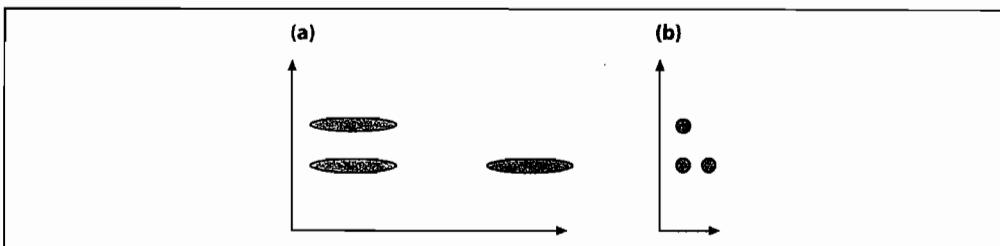


图 13-4：Mahalanobis 距离让我们理解数据的协方差为数据空间的“拉伸”：
(a)原始数据的垂直距离要比水平距离小；(b)通过对空间进行方差归一化，数据之间的水平距离小于垂直距离

我们通过图 13-4^①大致了解了什么是 Mahalanobis 距离。测量距离的时候必须考虑

① 注意，图 13-4 的协方差矩阵是对角阵，这样的协方差矩阵可以认为变量 X 和 Y 是独立的，之所以采用这样的协方差矩阵是为了使问题更简单。实际数据往往不会这么简单，会有很多种分布方式。

数据的协方差。首先，我们回忆一下协方差的概念。给定一个列表 X , X 包含 N 个数据样本，每个数据样本有 K 维，每一维的均值分别为 μ_1, \dots, μ_K 。它们的 $K \times K$ 协方差矩阵为：

$$\Sigma = E[(X - \mu)(X - \mu)^T]$$

其中 $E[\cdot]$ 是期望操作符。OpenCV 使用下面函数来计算协方差矩阵：

```
void cvCalcCovarMatrix (
    const CvArr** vvects,
    int count,
    CvArr* cov_mat,
    CvArr* avg,
    int flags
);
```

这个函数有一点点迷惑性。注意，参数 `vvects` 是一个指向 `CvArr` 的指针的指针。这意味着我们可以最多拥有 `vvects[0] ~ vvects[count-1]` 个 `CvArr`。到底有多少个 `CvArr`，实际上由参数 `flag` 决定。一般有以下两种情况。

1. 如果 `flag` 中 `CV_COV_ROWS` 和 `CV_COV_COLS` 都没有设置，`vvects` 则是一个指针数组，数组里的元素指向一维或者二维矩阵。具体说来，每一个 `vvects[i]` 可以指向一个一维或二维向量。这种情况下，如果 `CV_COVAR_CSCALL` 被设置，累积协方差的计算时，将除以 `count` 指定的数据点的个数。
2. 如果 `CV_COV_ROWS` 或者 `CV_COV_COLS` 其中之一被设置，那么在通常只有一个输入向量的时候，就只用 `vvects[0]`, `count` 被忽略。所有的输入都包含在向量 `vvects[0]` 中。
 - a. `CV_COV_ROWS`, 所有的输入向量在 `vvects[0]` 中以行排列。
 - b. `CV_COV_COLS`, 所有的输入向量在 `vvects[0]` 中以列排列。不可以同时设置行标志和列标志(详情参考标志相关描述)。

【476~477】

`Vvects` 可以存储 `8UC1`, `16UC1`, `32FC1` 和 `64FC1` 几种数据类型。任何情况下，`vvects` 包含了一个 K 维数据样本的列表。`cov_mat` 是得到的 $K \times K$ 协方差矩阵，它有 `CV_32FC1` 和 `CV_64FC1` 两种数据类型。`Avg` 是否使用向量取决于 `flag` 的设置。如果使用了 `avg`，则它与 `vvects` 的数据结构一致，包含 `vvects` 中的所有向量的 K 维均值。参数 `flags` 可以有很多种组合，总体来说，可以设置 `flag` 为下面的任何一个。

- **CV_COVAR_NORMAL** 按照前面所示的最一般的协方差矩阵计算公式来计

算。如果 CV_COVAR_SCALE 没有设置，则把结果用 count 平均，否则用 vcts[0] 的数据点个数来平均结果。

- CV_COVAR_SCALE 归一化计算得出的协方差矩阵。
- CV_COVAR_USE_AVG 使用 avg 矩阵而不通过每个特征来计算均值。如果已经得到均值(如通过调用 CVAvg())，设置这个参数可以节省计算时间。否则，程序会计算均值^①。
【477~478】

通常把数据结合成一个大矩阵，假设以行排列，那么，标志将被设置为 flags = CV_COVAR_NORMAL | CV_COVAR_SCALE | CV_COVAR_ROWS。

现在我们有了协方差矩阵，为了计算 Mahalanobis 距离，我们需要除以空间的方差，所以需要计算协方差矩阵的逆矩阵。可以通过下面这个函数来计算：

```
double cvInvert(
    const CvArr*     src,
    CvArr*          dst,
    int method =     CV_LU
);
```

在 cvInvert 中，源矩阵 src 是前面计算的协方差矩阵，des 是计算得到的逆矩阵。如果没有指定参数 method，则默认为 CV_LU，但是最好使用 CV_SVD_SYM 作为参数^②。

得到了协方差矩阵的逆矩阵，我们就可以求两个向量的 Mahalanobis 距离。Euclidean 距离是计算两个向量的对应数据的差的平方和，Mahalanobis 距离与 Euclidean 距离类似，只不过它还需要除以空间的协方差矩阵：

$$D_{\text{Mahalanobis}}(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$$

Mahalanobis 距离是一个数值。如果协方差矩阵是单位矩阵，则 Mahalanobis 距离退化为 Euclidean 距离。最后介绍 OpenCV 中计算 Mahalanobis 距离的函数，它输

-
- ① 如果用户有一个统计上更合理的均值，或者协方差矩阵由块来计算，那么应该输入事先计算好的均值向量。
 - ② 在这种情况下，也可以使用 CV_SVD，但是它比 CV_SVD_SYM 更慢且精度低。如果特征的维数远小于数据样本的数目，即使比 CV_LU 慢，也应该使用 CV_SVD_SYM。在这种情况下，总的计算时间中大部分由 cvCalcCovarMatrix() 消耗，所以稍微多花一点时间来获得更精确的协方差的逆矩阵是明智的(如果数据点集中于一个低维子空间中，可以更精确)。因此在这儿，CV_SVD_SYM 通常是该任务的最佳选择。

入两个向量(vec1,vec2)和一个逆协方差矩阵(mat)，返回一个双精度的距离。

```
double cvMahalanobis(
    const CvArr*  vec1,
    const CvArr*  vec2,
    CvArr*        mat
);
```

Mahalanobis 距离是多维空间中两点相似性的度量，它本身不是聚类或者分类算法。下面，我们开始讲解最常用的聚类算法：K 均值。【478】

K 均值

K 均值聚类算法在 cxcore 中，因为它在 ML 库诞生之前就存在了。K 均值尝试找到数据的自然类别。用户设置类别个数，K 均值迅速地找到“好的”类别中心。“好的”意味着聚类中心位于数据的自然类别中心。K 均值是最常用的聚类技术之一，与高斯混合中的期望最大化算法(在 ML 库中实现为 CvEM)很相似，也与第 9 章中讨论过的均值漂移算法(在 CV 库中实现为 cvMeanShift())相似。K 均值是一个迭代算法，在 OpenCV 中采用的是 Lloyd 算法^①，也叫 Voronoi 迭代。算法运行如下。

1. 输入数据集合和类别数 K(由用户指定)。
2. 随机分配类别中心点的位置。
3. 将每个点放入离它最近的类别中心点所在的集合。
4. 移动类别中心点到它所在集合的中心。
5. 转到第 3 步，直到收敛。

图 13-5 展示了 K 均值是怎么工作的。在这个例子中，只用两次迭代就达到了收敛。在现实数据中，算法经常很快地收敛，但是有的时候需要的迭代的次数比较多。

^① S. P. Lloyd, “Least Squares Quantization in PCM,” IEEE Transactions on Information Theory 28 (1982), 129–137.

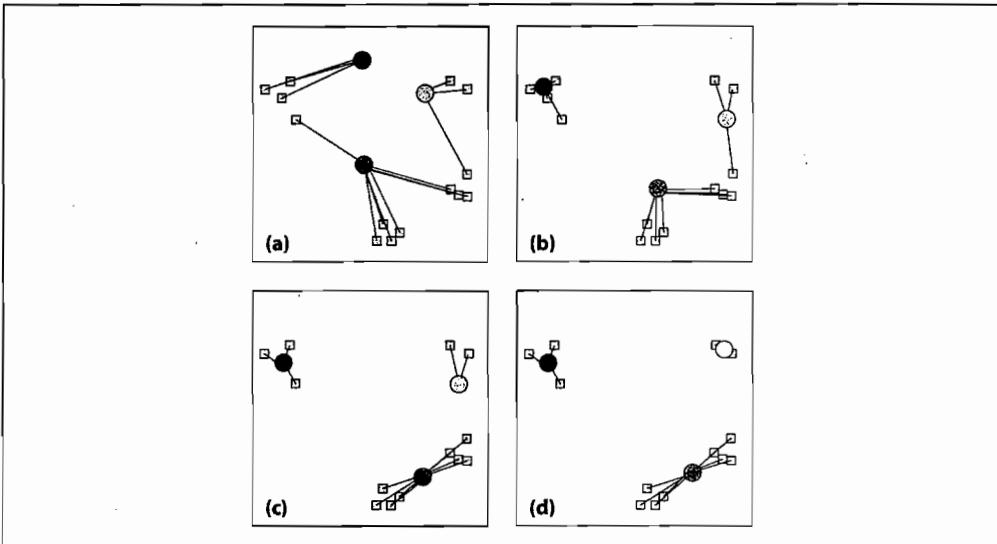


图 13-5：K 均值算法的迭代中有两步：(a)随机放聚类中心然后将数据样本聚到离它最近的中心；(b)数据中心移动到它所在类别的中心；(c)数据点根据最近邻规则重新聚到类别中心；(d)聚类中心再次移动到它所在的类别中心

问题和解决方案

K 均值是一个极其高效的聚类算法。但是它也有以下 3 个问题。

1. 它不保证能找到定位聚类中心的最佳方案，但是它能保证能收敛到某个解决方案(例如迭代不再无限继续下去)。
2. K 均值无法指出应该使用多少个类别。在同一个数据集中，例如对于图 13-5 中的例子，选择 2 个类别和选择 4 个类别，得到的结果是不一样的，甚至是不合理的。
3. K 均值假设空间的协方差矩阵不会影响结果，或者已经归一化(参考 Mahalanobis 距离的讨论)。

这三个问题都有“解决办法”。这些解决方法中最好的两种都是基于“解释数据的方差”。在 K 均值中，每个聚类中心拥有它的数据点，我们计算这些点的方差，最好的聚类在不引起太大的复杂度的情况下使方差达到最小。所以，列出的问题可以改进如下。

1. 多运行几次 K 均值，每次初始的聚类中心点都不一样(很容易做到，因为

OpenCV 随机选择中心点), 最后选择方差最小的那个结果。

2. 首先将类别数设为 1, 然后提高类别数(到达某个上限), 每次聚类的时候使用前面提到的方法 1。一般情况下, 总方差会很快下降, 直到到达一个拐点; 这意味着再加一个新的聚类中心不会显著地减少总方差。在拐点处停止, 保存此时的类别数。
3. 将数据乘以逆协方差矩阵。例如: 如果输入向量是按行排列, 每行一个数据点, 则通过计算新的数据向量 D^* , $D^* = D\Sigma^{-1/2}$ 来归一化数据空间。

【479 ~ 480】

K 均值代码

K 均值函数的调用很简单:

```
void cvKMeans2(
    const CvArr*      samples,
    int               cluster_count,
    CvArr*            labels,
    CvTermCriteria    termcrit
);
```

数组 sample 是一个多维的数据样本矩阵, 每行一个数据样本。这里有一点点微妙, 数据样本可以是浮点型 CV_32FC1 向量, 也可以是 CV_32FC2, CV_32FC3 和 CV_32FC(K)^① 的多维向量。参数 cluster_count 指定类别数, 返回向量 labels 包含每个点最后的类别索引。前面已经讨论了 termcrit。

阅读下面的 K 均值代码(例 13-1), 它的数据生成部分可以用于其他机器学习程序。

例 13-1: K 均值用法示例

```
#include "cxcore.h"
#include "highgui.h"

void main( int argc, char** argv )
{
    #define MAX_CLUSTERS 5
```

① 这完全等价于 $N \times K$ 矩阵, 其中有 N 行数据样本, 每个样本有 K 列, 数据类型是 32FC1。这两种表示的内存使用是完全相同的。

```

CvScalar color_tab[MAX_CLUSTERS];
IplImage* img = cvCreateImage( cvSize( 500, 500 ), 8, 3 );
CvRNG rng = cvRNG(0xffffffff);

color_tab[0] = CV_RGB(255,0,0);
color_tab[1] = CV_RGB(0,255,0);
color_tab[2] = CV_RGB(100,100,255);
color_tab[3] = CV_RGB(255,0,255);
color_tab[4] = CV_RGB(255,255,0);

cvNamedWindow( "clusters", 1 );

for(;;)
{
    int k, cluster_count = cvRandInt(&rng)%MAX_CLUSTERS + 1;
    int i, sample_count = cvRandInt(&rng)%1000 + 1;
    CvMat* points = cvCreateMat( sample_count, 1, CV_32FC2 );
    CvMat* clusters = cvCreateMat( sample_count, 1, CV_32SC1 );

    /* generate random sample from multivariate Gaussian
distribution */

    for( k = 0; k < cluster_count; k++ )
    {
        CvPoint center;
        CvMat point_chunk;
        center.x = cvRandInt(&rng)%img->width;
        center.y = cvRandInt(&rng)%img->height;
        cvGetRows( points, &point_chunk,
                   k*sample_count/cluster_count,
                   k == cluster_count - 1 ? sample_count :
                   (k+1)*sample_count/cluster_count );
        cvRandArr( &rng, &point_chunk, CV_RAND_NORMAL,
                   cvScalar(center.x,center.y,0,0),
                   cvScalar(img->width/6, img->height/6,0,0) );
    }

    /* shuffle samples */
    for( i = 0; i < sample_count/2; i++ )
    {
        CvPoint2D32f* pt1 = (CvPoint2D32f*)points->data.fl +

```

```

        cvRandInt (&rng) %sample_count;
CvPoint2D32f* pt2 = (CvPoint2D32f*)points->data.fl +
        cvRandInt (&rng) %sample_count;
        CvPoint2D32f temp;
        CV_SWAP( *pt1, *pt2, temp );
    }

cvKMeans2( points, cluster_count, clusters,
            cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,
                            10, 1.0 ));

cvZero( img );
for( i = 0; i < sample_count; i++ )
{
    CvPoint2D32f pt = ((CvPoint2D32f*)points->data.fl)[i];
    int cluster_idx = clusters->data.i[i];
    cvCircle( img, cvPointFrom32f(pt), 2,
              color_tab[cluster_idx], CV_FILLED );
}
cvReleaseMat( &points );
cvReleaseMat( &clusters );

cvShowImage( "clusters", img );

int key = cvWaitKey(0);
if( key == 27 ) // 'ESC'
    break;
}
}

```

在这段代码中，包含 `highgui.h` 来使用窗口输出，包含 `cxcore.h` 是因为它包含了 `Kmeans2()`。在 `main` 函数中，我们设置了返回的类别显示的颜色；设置类别个数上界 `MAX_CLUSTERS`（这儿是 5），类别的个数是随机产生的，存储在 `cluster_count` 中；设置数据样本的个数的上界(1000)，数据样本的个数也是随机产生的，被存储在 `sample_count` 中。在最外层循环 `for()` 中，我们分配一个浮点数双通道矩阵 `point` 来存储 `sample_count` 个数据样本，我们还分配了一个整型矩阵 `clusters` 来存储数据样本的聚类标签，从 0 ~ `cluster_count-1`。

【482~483】

然后，我们进入数据生成 `for()` 循环，这个循环也可以用于其他算法中使用。我们给每个类别填写 `sample_count/cluster_count` 个数据样本，这些 2 维数据样本

服从正态分布，正态分布的中心是随机选择的。

下一个 `for()` 循环仅仅打乱了数据样本的顺序。然后我们使用 `cvKMeans2()`，直到聚类中心的最大移动小于 1。

最后的 `for()` 循环画出结果，在图中显示。然后释放数据数组，等待用户的输入进入下一次计算或者通过 Esc 键退出。

朴素贝叶斯分类

前面提到的函数都在 `cxcore` 库中。现在我们开始讨论 OpenCV 的机器学习(ML)库。首先看到 OpenCV 最简单监督学习分类器 `CvNormalBayesClassifier`，也叫正态贝叶斯分类器或朴素贝叶斯分类器。它简单是因为它假设所有的特征之间相互独立，而这在现实中很少见(如，找到一只眼睛常常意味着另一只眼睛在附近)。Zhang 讨论了这个分类器有时能获得惊人性能的原因[Zhang04]。朴素贝叶斯分类器不能处理回归问题，但是它能有效地处理多类问题，而不仅仅是两类问题。这个分类器是当前快速发展的贝叶斯网络(也叫概率图模型)的最简单情况。贝叶斯网络是因果模型。在图 13-6 中，脸的存在产生了图像中的脸的特征。在使用中，脸是一个隐含变量，通过对输入图像的处理得到的脸部特征，组成了脸的观测证据。我们把这个叫做产生式模型，因为脸生成了脸部特征。它的逆过程，我们假设脸是存在的，然后在脸存在的前提下，随机采样生成了哪些特征^①。这种自上而下的由原因模型生成数据的方法可以处理纯判断型的模型不能处理的问题。比如，可以生成人脸用于计算机图形显示，机器人可以逐条想象怎样来生成场景、物体并让物体交互。而与图 13-6 形成对照的是，判断型模型会将图中的所有箭头翻转。

【481~483】

贝叶斯网络很深奥，一开始学习的时候很难理解。但是朴素贝叶斯算法衍生于贝叶斯法则的一个简单应用。在这个例子中，给定特征下脸的条件概率 p 有：

$$p(\text{face} | \text{LE}, \text{RE}, \text{N}, \text{M}, \text{H}) = \frac{p(\text{LE}, \text{RE}, \text{N}, \text{M}, \text{H} | \text{face})p(\text{face})}{p(\text{LE}, \text{RE}, \text{N}, \text{M}, \text{H})}$$

也可以写成：

① 因为朴素贝叶斯算法假设特征是独立的，所以使用它来生成人脸并不明智。但是一个更通用的贝叶斯网络可以特征相关时生成。

条件概率=可能性*先验概率/证据

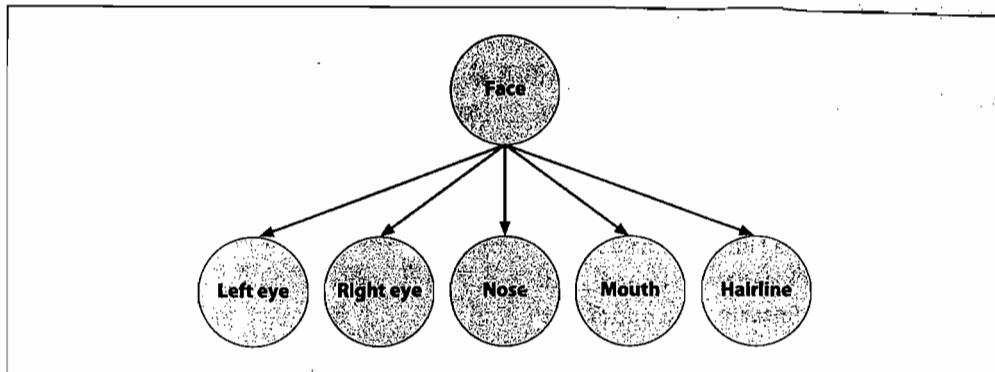


图 13-6：一个(朴素)贝叶斯网络，因为一个物体(人脸)的出现，产生底层特征

实际情况下，我们计算证据，然后决定产生证据的原因。因为用于计算的证据相同，我们可以不考虑这个条件。如果有很多模型，则需要找到有最大分子的那个。分子是模型和证据的联合概率： $p(\text{face}, \text{LE}, \text{RE}, \text{N}, \text{M}, \text{H})$ 。我们可以使用条件概率来表示这个联合概率：

$$\begin{aligned} p(\text{face}, \text{LE}, \text{RE}, \text{N}, \text{M}, \text{H}) \\ = p(\text{face})p(\text{LE}|\text{face})p(\text{RE}|\text{face}, \text{LE})p(\text{N}|\text{face}, \text{LE}, \text{RE}) \\ \times p(\text{M}|\text{face}, \text{LE}, \text{RE}, \text{N})p(\text{H}|\text{face}, \text{LE}, \text{RE}, \text{N}, \text{M}) \end{aligned}$$

应用特征独立的假设，我们获得简化的等式：

$$p(\text{object}, \text{all features}) = p(\text{object}) \prod_{i=1}^{\text{all features}} p(\text{feature}_i | \text{object}) \quad \text{【484】}$$

使用这个分类器的时候，要学习所有我们需要的物体的模型。运行的时候，我们计算特征，找到使上面等式最大的物体。然后，我们测试获得胜利的物体的概率是否大于某个给定值，如果是，我们就宣布物体被检测出来了，如果否，我们宣布没有任何一个物体被检测出来。

注意：如果只有一个感兴趣的物体，通常也是这样，那么你会问：“我正在计算的概率与什么有关？”在这种情况下，一直会有一个暗含的第二个物体：背景。背景是除去我们感兴趣的要学习和识别的物体之外的所有物体。

学习所有物体的模型比较简单。我们照很多张照片；计算这些照片的特征，计算每个物体的训练集中某个特征出现的比例。在实际计算中，我们不允许 0 概率，因为这样会除去某个存在物体；因此，0 概率被赋予一个很小的值。一般情况下，如果

没有很多的数据，简单的模型(朴素贝叶斯)会比很多复杂模型获得很好的性能，因为复杂模型使用了太多的假设，以致产生欠拟合。

朴素贝叶斯代码

朴素贝叶斯分类器的训练方法如下：

```
bool CvNormalBayesClassifier::train(
    const CvMat* _train_data,
    const CvMat* _responses,
    const CvMat* _var_idx = 0,
    const CvMat* _sample_idx = 0,
    bool           update = false
);
```

它与前面讲的一般训练方法一样，但是它只支持 CV_ROW_SAMPLE 类型的数据。输入变量 `_train_data` 应该是有序的(CV_VAR_ORDERED，数字)CV_32FC1 向量。输出标签 `_responses` 只能是 CV_VAR_CATEGORICAL(整数数值，即使是浮点向量)的向量列。参数 `_var_idx` 和 `_sample_idx` 可选，它们允许你标记你想使用哪些特征和数据样本。很多情况下会使用所有数据样本，因而赋值为 NULL，但是 `_sample_idx` 还可以用来区分训练集和测试集。这两个向量都是单通道整型数据(CV_32SC1)，0 意味着跳过。最后，当 `update` 变量为 `true` 的时候，使用新数据样本更新模型，`update` 为 `false` 的时候，从零开始训练模型。

朴素贝叶斯分类器的预测方法计算了它的输入向量的最大可能性的类别。一个或者多个数据按行排列在输入矩阵 `sample` 中。预测在 `results` 向量中返回对应的类别。如果只有一个输入向量，则只返回一个浮点值，`results` 将被设为 NULL。预测函数的格式如下：

```
float CvNormalBayesClassifier::predict(
    const CvMat* samples,
    CvMat*       results = 0
) const;
```

【485~486】

下面将讨论基于树的分类器。

二叉决策树

我们将具体讨论二叉决策树，它们是最常用的，且实现了机器学习库中大量的功能，因此将被作为指导性的例子来讲解。二叉决策树由 Leo Breiman 和他的同事提出^①。他们称之为“分类和回归树(CART)”。OpenCV 实现的就是“分类回归树”。算法的要点是给树的每个节点定义一个衡量标准。比如：当我们拟合一个函数的时候，我们使用真实值和预测值的差的平方和，这就是衡量标准。算法的目的是使差的平方和最小。对于分类问题，我们定义一个度量，使得当一个节点的大多数值都属于同一类时，这个度量最小。三个最常用度量是：熵(entropy)、吉尼系数(Gini index)和错分类(misclassification)，这些度量都会在本节中介绍。一旦我们定义了度量，二叉树搜寻整个特征向量，搜寻哪个特征和哪个阈值可以正确分类数据或正确拟合数据(在本书中称为使数据“纯净”)。根据惯例，我们说特征值大于这个阈值的数据为“真”，被分配到左分支；其他的点放到右分支。从二叉树的每个节点递归使用这个过程直到数据都“纯净”了，或者节点里的数据样本数达到最小值。

随后会给出节点的不纯度 $i(N)$ 。我们首先需要分两种情况来分析：回归和分类。

回归不纯度

回归或函数拟合中，节点不纯度仅仅是节点值 y 和数据值 x 的差的平方。我们需要最小化这个等式：

$$i(N) = \sum_j (y_j - x_j)^2$$

分类不纯度

分类中，决策树经常使用熵不纯度、吉尼不纯度和错分类不纯度这三个度量中的某一个来衡量。这些方法中，我们使用 $P(\omega_j)$ 来表示节点 N 中类别 ω_j 所占的比例。不同的方法中，不纯度有一些细微的差别。吉尼不纯度最经常用，所有的方法都努力使节点中的不纯度最小。图 13-7 显示了我们想要最小化的不纯度。

① L. Breiman, J. Friedman, R. Olshen, and C. Stone, Classification and Regression Trees (1984), Wadsworth.

$$\text{熵不纯度: } i(N) = \sum_j P(\omega_j) \log P(\omega_j)$$

$$\text{吉尼不纯度: } i(N) = \sum_{j \neq i} P(\omega_i)P(\omega_j)$$

$$\text{错分类不纯度: } i(N) = 1 - \max_i P(\omega_i)$$

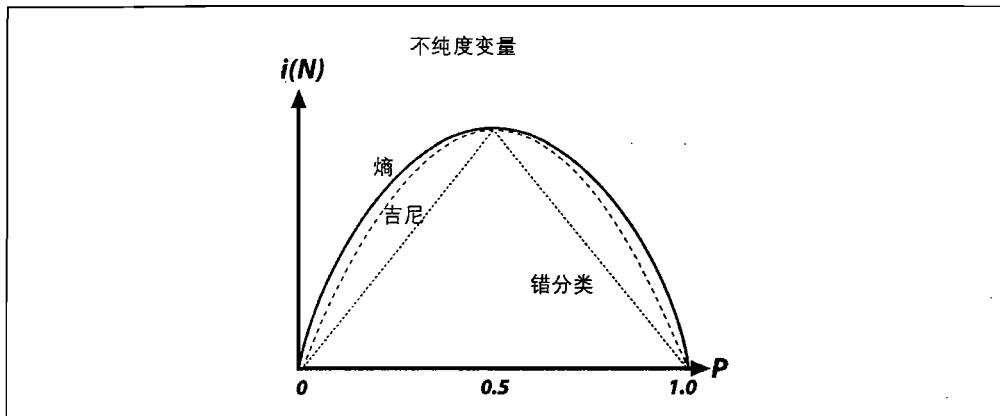


图 13-7: 决策树不纯度

决策树可能是最常用的分类方法。因为他们执行简单，容易解释结果，适应不同的数据类型(包括类别数据、数值数据、未归一化的和混合的数据)，能够处理数据丢失，通过分裂的顺序能够给数据特征赋不同的重要性。决策树构成了其他算法的基础，比如 boosting 和随机数，我们会简单地介绍这些算法。

决策树的使用

下面描述怎样使用决策树。有很多方法访问节点、改变分裂等。如果想要知道这些细节，请参考用户手册 [.../opencv/docs/ref/opencvref_ml.htm](#)。特别注意看类 CvDTree{}、训练类 CvDTreeTrainData{}、节点 CvDTreeNode{} 和分裂 CvDTreeSplit{}。
【487~488】

我们用实例来介绍决策树。在路径 [.../opencv/samples/c](#) 下，*mushroom.cpp* 文件对 *agaricuslepiota.data* 里的数据进行了决策树的分类。这个数据文件包含标签 p 和 e(分别表示有毒和无毒)，还有 22 个属性，每个属性只有一个字母。在 *mushroom.cpp* 中，函数 *mushroom_read_database()* 从文件 *agaricuslepiota.data* 中读取数据。这个函数过于特殊，并不具有通用性，它只对三个数组进行了赋值：
(1) 浮点数组 *data[][]*，数组的行数是数据样本的数目，列数是特征的数目(这里

是 22), 所有的特征都从类别字母转换为浮点数字; (2)单字节类型矩阵 missing[][]], 用“true”或者“1”表示数据文件里的缺失特征, 缺失特征在数据文件里用问号表示, 本数组的其他值为 0; (3)浮点向量 responses[], 里面包含有毒的字符“p”和无毒的字符“e”的对应浮点值。大多数情况下, 可以写一个更通用的数据输入程序。现在我们讨论 *mushroom.cpp* 最主要的工作, 也就是在 main 函数中被直接或者间接调用的部分。

训练树

为了训练树, 我们首先填充树的参数结构体 CvDTreeParams{}:

```
struct CvDTreeParams {

    int      max_categories;
                    //Until pre-clustering
    int      max_depth;       //Maximum levels in a tree
    int      min_sample_count;
                    //Don't split a node if less
    int      cv_folds;
                    //Prune tree with K fold cross-
validation
    bool     use_surrogates; //Alternate splits for missing data
    bool     use_lse_rule;   //Harsher pruning
    bool     truncate_pruned_tree;
                    //Don't "remember" pruned branches
    float    regression_accuracy;
                    //One of the "stop splitting" criteria
    const float* priors; //Weight of each prediction category

    CvDTreeParams() : max_categories(10), max_depth(INT_MAX),
                      min_sample_count(10), cv_folds(10), use_surrogates(true),
                      use_lse_rule(true), truncate_pruned_tree(true),
                      regression_accuracy(0.01f), priors(NULL) { ; }

    CvDTreeParams(
        int      _max_depth,
        int      _min_sample_count,
        float    _regression_accuracy,
        bool     _use_surrogates,
        int      _max_categories,
```

```

    int      _cv_folds,
    bool     _use_lse_rule,
    bool     _truncate_pruned_tree,
    const float* _priors
};

}

```

【488~489】

在结构体中，`max_categories` 默认为 10。这个值限制了特征类别取值的数目，决策树会事先将这些类别聚类，使其可以测试不超过 $2^{max_categories}-2$ 种可能性^①。这个参数对有序的或者数字的特征没有影响，因为对于有序的或者数字的特征，算法仅需要找到分裂的阈值。类别个数超过 `max_categories` 的变量将自动聚类到 `max_categories` 个类别个数。这样，决策树每次只需要测试不多于 `max_categories` 个层。当这个参数设置得比较小的时候，将减少计算复杂度，但也损失了精确度。

其他参数的名称不言自明。最后一个参数 `priors` 非常重要。它设置了错分类的代价。这是说，如果第 1 个类别的代价是 1，第 2 个类别的代价是 10，则预测第 2 个类别出现一次错误则等于预测第 1 个类别出现 10 次错误。在这个程序中，我们判断有毒和无毒的蘑菇，所以我们惩罚把毒蘑菇认为无毒 10 倍于把无毒的蘑菇认为有毒。

训练决策树的方法如下，有两个函数：第一个为了直接使用决策树；第二个为了全体(用法如 boosting)或森林(在随机数使用)。

```

//Work directly with decision trees:
bool CvDTree::train(
    const CvMat* _train_data,

```

① 更多关于标签和数值分裂的介绍：对于数值变量，分裂的形式是“如果 $x < a$ ，则到左子节点，否则到右子节点”；如果对于标签变量，分裂的形式是“如果 $x \in \{v_1, v_2, v_3, \dots, v_k\}$ ，则到左子节点，否则到右子节点，此处 v_i 是变量的可能取值”。因此，如果一个特征变量有 N 种可能取值，那么为了找到此变量最优分裂，需要尝试 2^N-2 种子集(不包括空集和满集)。一个算法往往需要将 N 种取值聚为 $K \leq max_categories$ 个类(通过 K 均值聚类)。所以算法尝试各种不同的聚类组合，然后取最好的分裂，这一般可以获得很好的结果。对于两个经常使用的任务，两类问题的分类和回归，不需要聚类就可以方便地找到最优分割(如取值的最优组合)。所以只有在这种情况下才会使用聚类： $n > 2$ 类的分类问题，且类别变量 $N > max_categories$ 。所以将 `max_categories` 设置为大于 20 的值时，请务必三思而后行，因为每次分裂需要的操作超过一百万次操作。

```

int _tflag,
const CvMat* _responses,
const CvMat* _var_idx = 0,
const CvMat* _sample_idx = 0,
const CvMat* _var_type = 0,
const CvMat* _missing_mask = 0,
CvDTreeParams params = CvDTreeParams()
);

//Method that ensembles of decision trees use to call
//individual training for each tree in the ensemble
bool CvDTree::train(
CvDTreeTrainData* _train_data,
const CvMat* _subsample_idx
);

```

【489~490】

在 train 方法中，我们输入 2 维浮点型矩阵 _train_data[][]。在矩阵中，如果 _tflag 设置为 CV_ROW_SAMPLE，每一行是一个包含特征向量的数据点。如果 _tflag 设置为 CV_COL_SAMPLE，则行和列对调。参数 _responses [] 是一个浮点型向量，包含数据点对应的类别。其他的参数可选。向量 _var_idx 指出需要考虑的特征，_sample_idx 指出需要包含的数据点；这 2 个向量要么是基于 0 的整数列，要么是 8 位的标记，1 预示有用，0 预示跳过(具体请参考本章前面的 train() 介绍)。字节型向量 _var_type 是一个各个特征类型的基于 0 的标记(特征类型是 CV_VAR_CATEGORICAL 还是 CV_VAR_ORDERED)^①，它的长度是特征的个数加 1，最后一个数指定学习结果的类型。字节型的矩阵 _miss_mask [][] 指定哪些是丢失的值(如果相对于的位置标记为 1)。例 13-2 是决策树创建和训练的例子。

例 13-2：创建并训练一个决策树

```

float priors[] = { 1.0, 10.0 }; // Edible vs poisonous weights
CvMat* var_type;
var_type = cvCreateMat( data->cols + 1, 1, CV_8U );
cvSet( var_type, cvScalarAll(CV_VAR_CATEGORICAL) ); // all
these vars are categorical
CvDTree* dtree;
dtree = new CvDTree;
dtree->train(

```

① CV_VAR_ORDERED 跟 CV_VAR_NUMERICAL 完全相同。

```

    data,
    CV_ROW_SAMPLE,
    responses,
    0,
    0,
    var_type,
    missing,
    CvDTreeParams(
        8,           // max depth
        10,          // min sample count
        0,           // regression accuracy: N/A here
        true,         // compute surrogate split,
                      // since we have missing data
        15,          // max number of categories
                      // (use suboptimal algorithm for
                      // larger numbers)
        10,          // cross-validations
        true,         // use 1SE rule => smaller tree
        true,         // throw away the pruned tree branches
        priors // the array of priors, the bigger
                      // p_weight, the more attention
                      // to the poisonous mushrooms
    )
);

```

【490~491】

在这段代码中，声明并创建了决策树 dtree。调用了 dtree 的训练算法。在这个例子中，向量 responses[] 设置为 ASCII 码的“p”和“e”。在训练结束之后，dtree 就可以用来预测新数据了。决策树也可以通过 save 方法保存到磁盘，通过 load 方法从磁盘读取。在下面的例子中，在保存和加载之间，我们使用了 clear 方法来将决策树重置并清空。

```

dtree->save("tree.xml", "MyTree");

dtree->clear();

dtree->load("tree.xml", "MyTree");

```

上面代码保存和加载了一个叫 “tree.xml”的树文件。选项 “MyTree” 在 tree.xml 中标记了一个树。和机器学习中其他的统计模式一样，使用 save 的时候并不能存储多个对象；多对象存储的时候要使用 cvOpenFileStorage() 和

`write()`^①。但是，`load` 函数不一样，如果一个文件中存储了多了对象，它可以根据名称提取出对应的对象。

决策树的预测函数如下：

```
CvDTreeNode* CvDTree::predict(  
    const CvMat* _sample,  
    const CvMat* _missing_data_mask = 0,  
    bool raw_mode = false  
) const;
```

此处 `_sample` 是一个需要预测的浮点型的特征向量。`_missing_data_mask` 是一个同样长度和同样大小^②的字节向量，它的非 0 值预示着对应的特征丢失。最后，`raw_mode` 用 `false` 指出输入类别数据没有归一化，`true` 指出输入类别数据经过了归一化，主要作用是提高预测速度。在(0,1)区间内归一化数据可以加速计算，因为算法可以知道数据的波动。这样的归一化对精确度没有影响。预测方法返回决策树的一个节点，可以通过 `(CvDTreeNode *)->value` 得到预测值：

```
double r = dtree->predict( &sample, &mask )->value; 【491~492】
```

最后，我们调用 `var_importance()` 得到初始特征的重要性。这个函数返回一个 $N \times 1$ 的双精度(CV_64FC1)向量，向量包含每个特征的重要性，1 代表最重要，0 代表完全不重要。不重要的特征在 2 次训练中可以被排除。(图 13-12 是变量重要性的图示)这个函数的调用如下：

```
const CvMat* var_importance = dtree->get_var_importance();
```

如 `.../opencv/samples/c/mushroom.cpp` 文件所示，每个独立特征的重要性可以直接由下式访问：

```
double val = var_importance->data.db[i];
```

大多数用户只训练并使用决策树，但是高级用户或者研究人员可能想测试树节点、改变树节点或者改变分裂标准。如本节开始提到的，可以参考 OpenCV 文档 `...opencv/docs/ref/opencvref_ml.htm#ch_dtrees`，还可以通过 OpenCV 维基

-
- ① 如前所述，函数 `save()` 和 `load()` 是复杂函数 `write()` 和 `read()` 的一个易于使用的包装。
 - ② 此处如果 `_sample` 是一个 $1 \times N$ 向量，则此样本也必须是 $1 \times N$ ；如果 `_sample` 是 $N \times 1$ ，那么此样本必须是 $N \times 1$ 。

(<http://opencvlibrary.sourceforge.net/>)获得。深入分析的重点是类结构 CvDtree{}, 训练 CvDTreeTrainData{}, 节点结构 CvDTreeNode{} 和分裂结构 CvDTreeSplit{}。

决策树结果

用前面代码，我们可以从 *agaricus-lepiota.data* 文件中学习蘑菇有毒还是无毒。如果我们训练决策树而不经过修剪，它可以很好地学习数据，结构如图 13-8 所示。尽管完全的决策树可以很好地学习训练集，但是会产生图 13-2 所描述的现象(过拟合)。我们在图 13-8 中不仅学习了数据，也学习了噪声和错误。因此，它在实际数据中的效果可能并不好。这就是为什么 OpenCV 决策树和 CART 树需要包含一个附加的步骤来通过修剪树来达到复杂度和性能的平衡。其他的有些树通过在建立树的同时考虑复杂度，以把修剪融合到训练环节里面。但是，在开发 ML 库的时候，发现先建立树再修剪(OpenCV 是这样实现的)的效果胜于边建立树边修建好。

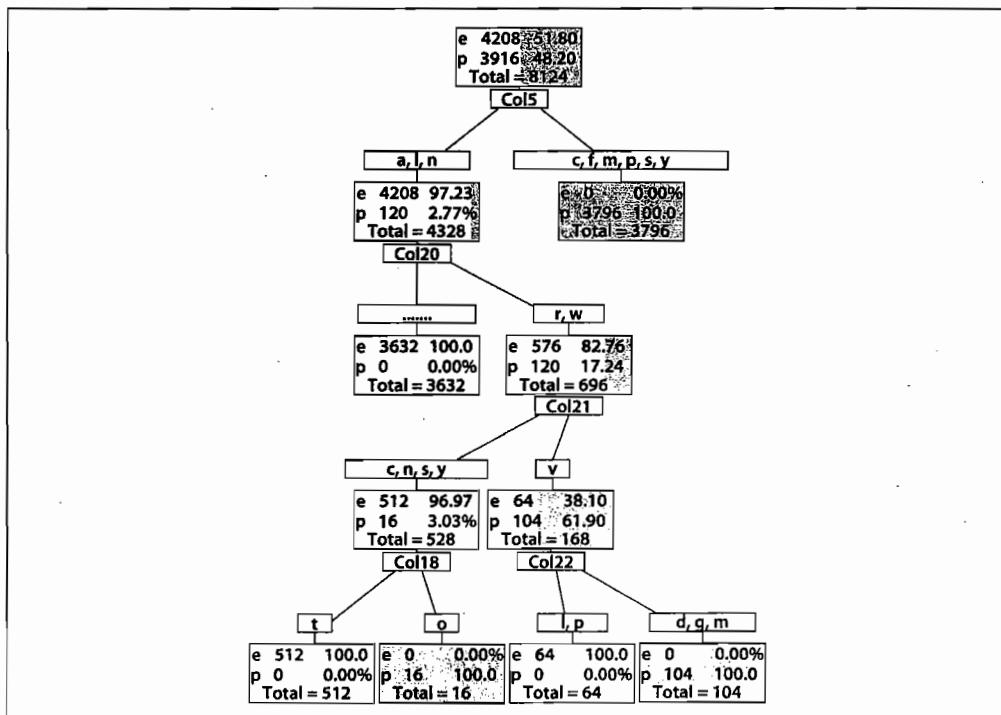


图 13-8：区分有毒(p)和无毒(e)蘑菇的完整决策树：该决策树非常复杂，使训练集上的错误率为 0%，但是有可能不能处理测试数据或者真实数据的变化(矩形中的深色部分表示在那一步有毒蘑菇的比例)

图 13-9 显示了一个在训练集上并不是非常好的，但是在实际数据中会有更好的性能的树，因为它在过拟合和欠拟合中达到了平衡。但是这种分类器有一个缺点：尽管它在数据的总误差上获得比较好的性能，它仍有 1.23% 的概率将有毒的蘑菇判断为没有毒。可能我们更希望得到一个性能略差一点、尽管会标记错很多没有毒的蘑菇，但是不会标记错有毒的蘑菇的分类器。这样的分类器可以通过故意歧视分类器或者故意歧视数据来得到。这种方法会给分类器的分类错误增加代价。在我们的例子中，我们认为错分有毒蘑菇的代价很高，而错分无毒蘑菇的代价较小。OpenCV 允许调整训练参数 CvDTreeParams{} 中的 prior 来达到目的。我们也可以重复使用“坏”数据来加强 prior 的代价。重复使用“坏”数据暗中增加“坏”数据的权重。

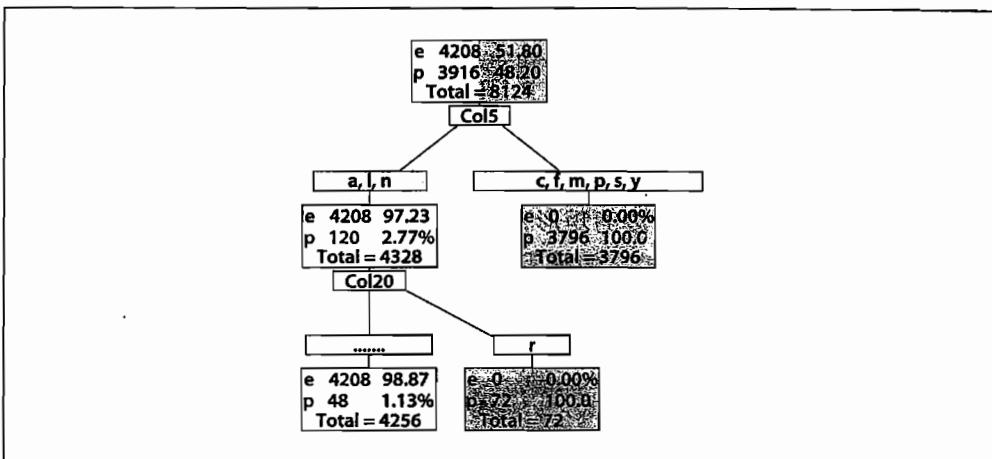


图 13-9：区分有毒(p)和无毒(e)蘑菇的修剪后的决策树：虽然被修剪导致该决策树在训练集上有一个较低的错误率，但是它肯定能够较好地处理真实数据

图 13-10 展示了一个给毒蘑菇分错加了 10 倍权值的树。这个树以牺牲判断无毒蘑菇的准确率来使判断毒蘑菇完全正确。“安全比犯错好”。无偏见和有偏见的树的混淆矩阵如图 13-11 所示。

最后，我们讨论怎么利用决策树产生的变量重要性，OpenCV 的树分类器中包含变量重要性^①。变量重要性技术度量了每个变量对分类器性能的贡献。哪些特征丢弃后使性能下降越多，哪些特征就越重要。决策树可以直接通过数据的分裂来表示重要性：第一层一般会比第二层重要。分裂是重要性的有效指示，但它是以贪婪方式

① 变量重要性技术可用于任何分类器，但是目前 OpenCV 只在树分类器中实现了变量重要性。

来实现的，即找到当前使数据最纯净的分裂。但有些时候，先做不重要的分裂会得到更好的效果，但是目前决策树不会这样做^①。毒蘑菇的变量重要性如图 13-12 所示，它包括无偏见的树和有偏见的树。可以看到变量重要性也与树的偏见有关。

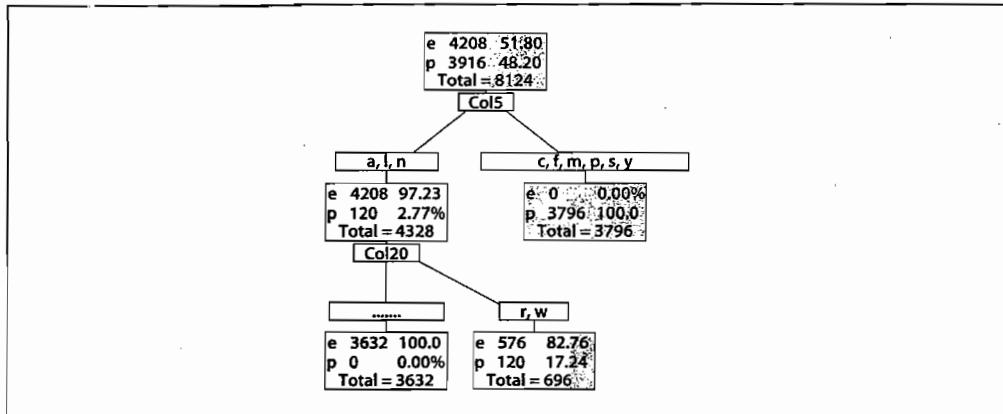


图 13-10：一个食用蘑菇分类器，如果将有毒蘑菇错分为可食用蘑菇，则有 10 倍的惩罚。请注意最下面右边的矩形，虽然大部分是可食用蘑菇，但是达不到有毒蘑菇的 10 倍，所以仍被认为是不可食用的

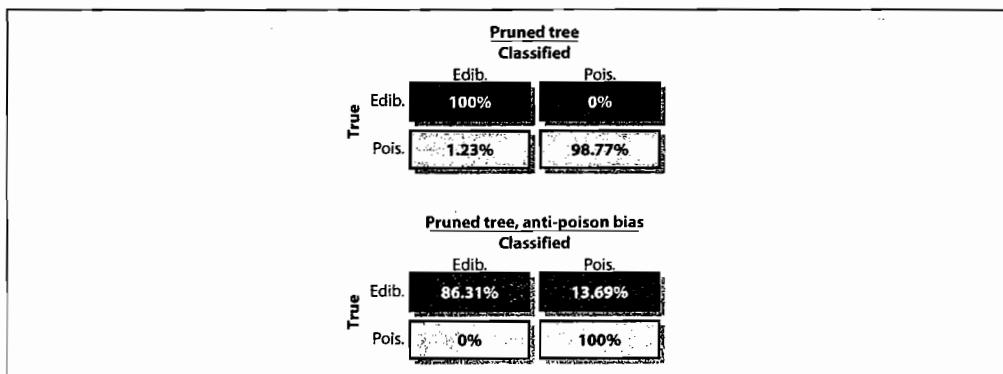


图 13-11：用于区分可食用蘑菇的修剪决策树的混淆矩阵(confusion matrices)。无惩罚的分类获得更好的总识别率(上图)，但是将一些有毒蘑菇误分为可食用的；有惩罚的分类虽然识别率不是那么高(下图)，但是不会将有毒蘑菇分为可食用的

^① OpenCV 在所有的分裂中都计算变量重要性(根据 Breiman 的方法实现)，包含代理分裂也是这样，这降低了 CART 贪婪分裂算法在变量重要性上的负面影响。

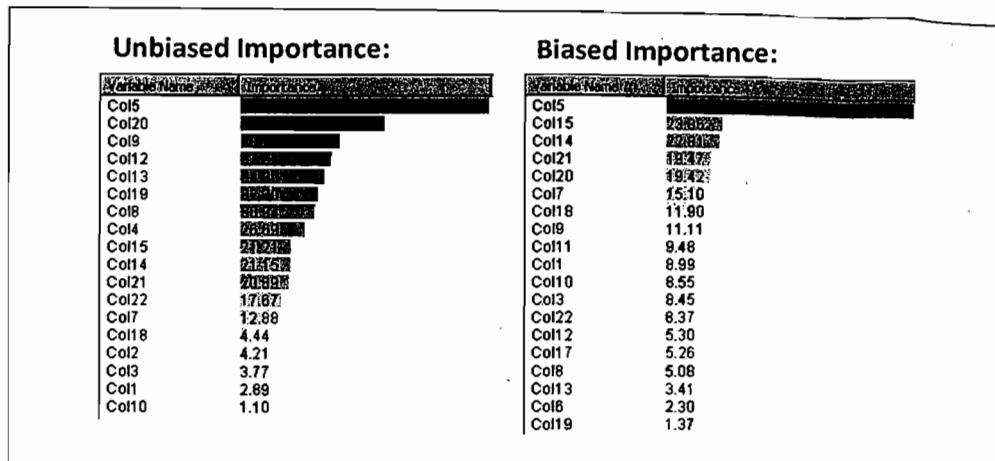


图 13-12：由无惩罚的树(左图)分类食用蘑菇的变量重要性，以及由有惩罚的树(右图)的分类重要性

【494~495】

boosting

决策树很有用，但是它们并不是最好的分类器。在本节和下节，我们讲述两种技术：boosting 和随机森林。它们在内部使用了决策树，所以继承了树的很多有用的性质(能够处理混合数据类型、没有归一化的数据、特征缺失)。这些技术能够获得相当好的性能，因此它们通常是 ML 库中最好的监督算法^①。

在监督学习领域，有一个叫统计提升(meta-learning)的学习算法(由 Michael Keans 首次提出)。Keans 想知道可不可能从很多弱分类器中学习到一个强分类器^②。第一个 boosting 算法，叫 AdaBoosting，由 Freund 和 Schapire 提出^③。OpenCV 包含 4 种类型的 boosting：

1. CvBoost::DISCRETE(discrete AdaBoost)

- ① 请回顾“没有免费的午餐”法则，它告诉我们没有总是最优的分类器。但是在计算机视觉方面的很多数据上，boosting 和随机数可以获得很好的性能。
- ② 一个“弱分类器”的输出结果只是比随机分类略好一些，“强分类器”的输出结果几乎接近完全正确分类。因此弱分类器和强类器只是统计意义上的定义。
- ③ Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm”, in Machine Learning: Proceedings of the Thirteenth International Conference (Morgan Kauman, San Francisco, 1996), 148–156.

2. CvBoost::REAL(real AdaBoost)
3. CvBoost::LOGIT(LogitBoost)
4. CvBoost::GENTLE(gentle AdaBoost)

它们都是由原始 AdaBoost 变化而来。我们发现 real AdaBoost 和 gentle AdaBoost 的效果最好。Real AdaBoost 利用置信区间预测，在标签数据上有很好的性能。Gentle AdaBoost 对外围数据(outlier data)赋较小的值，所以在处理回归问题上效果很好。LogitBoost 也可以很好的处理回归问题。只需设置一下标志参数，就可以使用不同的 AdaBoost 算法，所以在处理实际问题的时候，可以尝试所有的算法，然后挑选最好的那个^①。下面我们讨论原始的 AdaBoost 算法。注意，OpenCV 执行的 boosting 算法，是一个两类分类器^②（不像决策树和随机森林）。还有，LogitBoost 和 GentleBoost 不仅可以用来解决两类分类，还可以用于回归问题(详见后文)。

【495~497】

AdaBoost

boosting 算法训练了 T 个弱分类器 h_t , $t \in \{1, \dots, T\}$ 。这些弱分类器很简单。大多数情况下，它们是只包含一次分裂(称为决策 stumps)或仅有几次分裂(可能到 3 次)的决策树。最后做决定的时候将赋值权重 α_t 给每个分类器。AdaBoost 训练时输入的特征向量是 x_i ，向量的类别标签是 y_i (这儿 $i=1, \dots, M$, M 是样本总数)，且 $y_i \in \{1, -1\}$ 。首先，我们初始化数据样本的权值 $D_1(i)$ 来告诉分类器将一个数据点分类错误的代价是多少。boosting 算法的主要特征是，在训练过程之中，这个代价将会更新，使得后来的弱分类器更加关注与前面的分类器没有分对的数据点。算法如下：

1. $D_1(i)=1/m$, $i=1, \dots, m$
2. 针对 $t = 1, \dots, T$:
 - a. 寻找使得权重为 $D_t(i)$ 的总错误最小的分类器 h_t 。
 - b. 求 $h_t = \operatorname{argmin}_{h_j \in H} \varepsilon_j$ ，这儿 $\varepsilon_j = \sum_{i=1}^m D_t(i) (\text{其中 } y_i \neq h_j(x_i))$ ，如果最小错误满

-
- ① 这种处理方式可以称为“魔法学习”或者“魔法规划”。虽然没有太多道理，但是通常可以获得最高的准确率。有时候，通过深入思考，可以发现某个方法最好的原因，并能更深刻的理解数据；但是有时候却无法找出原因。
 - ② 有一个技巧称为 unrolling，它可以将任何两类分类器转换为多类分类器，但这会使得训练和识别的计算代价很高。请参考例程 .../opencv/samples/c/letter_recog.cpp。

足 $\varepsilon_j < 0.5$, 则继续; 否则退出。

- c. 设置 h_t 的权重 $\alpha_t = \log[(1 - \varepsilon_t)/\varepsilon_t]$, 这儿 ε_t 是步骤 2b 中的最小错误。
- d. 更新数据点权重: $D_{t+1}(i) = [D_t(i) \exp(-\alpha_t y_i h_t(x_i))] / Z_t$, 这儿 Z_t 将所有数据点的权重归一化。

注意, 在 2b 步, 如果找不到小于 50% 错误率的分类器, 则停止, 表明可能需要更好的特征。

当前面讲的训练算法结束之后, 最后的强分类器接受输入向量 x , 使用所有弱分类器的加权和来进行分类。

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

这里, 符号函数把所有正数变为 1, 把所有的负数变为 -1(0 仍然为 0)。【497~498】

boosting 代码

.../opencv/samples/c/letter_recog.cpp 展示了怎么使用 boosting、随机森林和反向传播网络(即多层传感网络, MLP)。boosting 的代码与决策树的代码相似, 但是除了决策树的参数, 它还拥有自己的控制参数。

```
struct CvBoostParams : public CvDTreeParams {
    int boost_type; // CvBoost:: DISCRETE, REAL, LOGIT, GENTLE
    int weak_count; // How many classifiers
    int split_criteria;
    // CvBoost:: DEFAULT, GINI, MISCLASS, SQERR
    double weight_trim_rate;
    CvBoostParams();
    CvBoostParams(
        int          boost_type,
        int          weak_count,
        double       weight_trim_rate,
        int          max_depth,
        bool         use_surrogates,
        const float* priors
    );
};
```

在 CvBoostParams 中, boost_type 从前面列出的四个算法中做出选择。

`split_criteria` 为下面的值之一：

- `CvBoost :: DEFAULT`(默认配置)
- `CvBoost :: GINI`(real AdaBoost 的默认配置)
- `CvBoost :: MISCLASS`(discrete AdaBoost 的默认配置)
- `CvBoost :: SQERR`(最小平方误差；只能在 `LogitBoost` 和 `gentle AdaBoost` 时使用)

最后一个参数 `weight_trim_rate` 用于保存计算结果。当训练进行时，很多数据点变得不重要。也就是说，第 i 个点的权值 $D_i(i)$ 变得很小。`weight_trim_rate` 是一个 0 到 1(包含)的阈值，在给定的 `boosting` 迭代中暗中丢弃一些数据点。例如：`weight_trim_rate` 是 0.95。这意味着样本总权值小于 $1.0 - 0.95 = 0.05$ 的点将不参加训练的下一次迭代。这些点并不是永久删除。当下一个弱分类器训练结束，所有的点的权值将被重新计算，有些前面被丢弃的样本则有可能进入下一个训练集合。关闭这个功能，只需要将 `weight_trim_rate` 置为 0 即可。

我们观察到 `CVBoostParams()` 是继承于 `CvDTreeParams()`，所以可以设置其他与决策树有关的参数。实际情况下，如果处理的特征有丢失^①，可以设置 `CvDTreeParams::use_surrogates`，它可以保证每次分裂选取的特征将被存储在相应的节点中。另一个重要的参数是使用 `priors` 来设置将非物体认为是物体的代价。再说一次，如果我们训练有毒和无毒的蘑菇，我们可以设置 `priors` 为 `float priors={1.0, 10.0}`；如此一来，把有毒蘑菇标记为无毒的错误代价则比把无毒的蘑菇标记为有毒的代价大 10 倍。

【498~499】

`CvBoost` 类包含成员变量 `weak`，它是一个 `CvSeq*` 类型的指针，指向继承于 `CvDTree` 决策树的弱分类器^②。`LogitBoost` 和 `GentleBoost` 中，决策树是回归树，其他的 `boosting` 算法中决策树则只判断类别 0 和类别 1。`CvBooatTree` 的声明如下：

-
- ① 请注意，对于计算机视觉来说，特征是从一幅图像中计算得来，然后传递给分类器，所以几乎不会有数据丢失。数据丢失现象经常发生在由人采集数据时，例如测量病人一天中的体温。
 - ② 对象的命名有时候没有道理。`CvBoost` 类型的对象是 `boosting` 树分类器。`CvBoostTree` 是组成总的 `boosting` 强分类器的弱分类器。分析起来可能是这样的原因：类型为 `CvBoostTree` 的弱分类器继承于 `CvDTree`(例如它们是很小的树，如此之小以至于只能称为 `stump`)。`CvBoost` 的成员变量 `weak` 指向一个序列，序列里的元素是类型为 `CvBoostTree` 的弱分类器。

```

class CvBoostTree: public CvDTree {

public:
    CvBoostTree();
    virtual ~CvBoostTree();
    virtual bool train(
        CvDTreeTrainData* _train_data,
        const CvMat* subsample_idx,
        CvBoost* ensemble
    );
    virtual void scale( double s );
    virtual void read(
        CvFileStorage*      fs,
        CvTreeNode*         node,
        CvBoost*            ensemble,
        CvDTreeTrainData*   _data
    );
    virtual void clear();

protected:
    ...
    CvBoost* ensemble;
};


```

boosting 算法的训练和决策树的训练几乎一样。不同的是多了一个外部参数 update，它的默认值为 0。这样，我们重新训练所有的弱分类器。如果 update 设置为 1，我们就把新训练出的弱分类器加到已经存在的弱分类器集合中。训练 boosting 分类器的函数原型如下：

```

bool CvBoost::train(
    const CvMat*      _train_data,
    int               _tflag,
    const CvMat*      _responses,
    const CvMat*      _var_idx = 0,
    const CvMat*      _sample_idx = 0,
    const CvMat*      _var_type = 0,
    const CvMat*      _missing_mask = 0,
    CvBoostParams     params = CvBoostParams(),
    bool              update = false
);

```

【499~500】

.../opencv/samples/c/letter_recog.cpp 中有一个训练 boosting 分类器的例子。训练部分的代码片段如下。

例 13-3：训练 boosting 分类器的代码片段

```
var_type = cvCreateMat( var_count + 2, 1, CV_8U );

cvSet( var_type, cvScalarAll(CV_VAR_ORDERED) );

// the last indicator variable, as well
// as the new (binary) response are categorical
//
cvSetReal1D( var_type, var_count, CV_VAR_CATEGORICAL );
cvSetReal1D( var_type, var_count+1, CV_VAR_CATEGORICAL );

// Train the classifier
//
boost.train(
    new_data,
    CV_ROW_SAMPLE,
    responses,
    0,
    0,
    var_type,
    0,
    CvBoostParams( CvBoost::REAL, 100, 0.95, 5, false, 0 )
);

cvReleaseMat( &new_data );
cvReleaseMat( &new_responses );
```

boosting 算法的预测函数也与决策树很相似：

```
float CvBoost::predict(
    const CvMat* sample,
    const CvMat* missing = 0,
    CvMat* weak_responses = 0,
    CvSlice slice = CV_WHOLE_SEQ,
    bool raw_mode = false
) const;
```

在预测中，我们传入一个特征向量 sample，predict() 函数则返回一个预测值。

当然，有一些可选参数。第一个参数是 `missing`，与决策树相同，它包含一个与 `sample` 向量维度相同的字节类型向量，非 0 标志着特征的丢失。(注意，这个标志只有在训练分类器时使用了 `CvDTreeParams::use_surrogates` 的前提下才有效)

如果想得到每个弱分类器的输出，可以传入一个浮点型的 `CvMat` 向量 `weak_response`，它的长度和弱分类器的个数一样。如果向量 `weak_response` 被传入，预测函数将根据每个分类器的输出填充这个向量。

```
CvMat* weak_responses = cvCreateMat(
    1,
    boostedClassifier.get_weak_predictors()->total,
    CV_32F
);
```

预测函数的下一个参数 `slice` 指定使用弱分类器的那个相邻子集，它可以这样设置：

```
inline CvSlice cvSlice( int start, int end );
```

但是，我们通常使用默认值，把 `slice` 设置为“所有弱分类器”(`CvSlice slice=CV_WHOLE_SEQ`)。最后，我们看 `raw_mode`，它的默认值为 `false`，但是可以设置为 `true`。这个参数的用法与决策树完全一样，标志着数据是否已经归一化。正常情况下，我们不使用这个参数。调用 `boost` 预测的例子如下：

```
boost.predict(temp_sample, 0, weak_response);
```

最后，一些辅助函数可以使用。我们可以使用 `void CvBoost::prune(CvSlice slice)` 来从模型中去除一个弱分类器。

我们也可以使用 `CvSeq* CvBoost::get_weak_predictor()` 来获得所有的弱分类器。这个函数返回一个类型为 `CvSeq*` 的指针序列，序列中元素为 `CvBoostTree` 类型。【500~501】

随机森林

OpenCV 包含随机森林(random tree)类，它是根据 Leo Breiman 的随机森林方法(random forest)^① 执行的。随机森林可以通过收集很多树的子节点对各个类别的投

① Breiman 的大部分关于随机森林的方法被收集到一个网页上(http://www.stat.berkeley.edu/users/breiman/RandomForests/cc_home.htm)。

票，然后选择获得最多投票的类别作为判断结果。通过计算“森林”的所有子节点上的值的平均值来解决回归问题。随机森林包含随机选择的一些决策树，在ML库建立的时候，它是当时最好的分类器之一。即使是在非共享内存的系统上，随机森林在并行执行方面也很大的潜力可挖。这个特征使它在未来发展占据优势。随机森林建立时的基本子系统也是决策树。在建立决策树时会一直继续下去直到数据纯净。因此(图13-2的上半部分，过拟合)，尽管每个树都很好学习了训练数据，但是各个树之间仍有很大的不同。为了消除这些不同，我们把这些树放在一起平均(因此叫随机森林)。

当然，如果所有的树都很相似，随机森林就没有很大的作用。为了克服这点，随机森林通过在树的建立过程中随机选择特征子集来使各个树不相同。例如，一个目标识别树可以有很多可能的特征：颜色、质地、倾斜度、倾斜方向、变量、值的比等。树的每个节点可以从这些特征中随机的选择子集来决定怎样最好地分裂数据。每个后来的节点获得新的、随机选择的特征子集来分裂。随机子集的规模一般是特征数量的开方。因此，如果我们有100个可能的特征，每个对接点将随机选择10个特征，并利用这10个特征对数据进行最好地分裂。为了提高鲁棒性，随机森林使用袋外(out of bag)方法来检验分裂。给定任何节点，训练是发生在一个随机选择然后替换的数据子集上进行的^①；没有选到的数据被叫做“out of bag(OOB)”数据，将用于估计分裂的性能。OOB数据一般为所有数据的1/3。

如同其他基于树的方法，随机森林继承了树的很多属性：丢失值的代理分裂，可以处理标签和数值特征，不需要归一化数据，容易获得对预测很重要的变量。随机森林还使用OOB错误来估计它在处理未见过数据时的性能。如果训练数据与测试数据的分布相似，OOB性能预测会更精确。

最后，随机森林可以用来确定两个数据样本的亲近度(是相似度，而不是距离)。方法如下：(1)将数据样本放入判别树；(2)计算它们到达同样的叶子(子节点)的次数；(3)用相同的叶子数除以树的总数。亲近度为1意味着完全亲近，0意味着完全不亲近。亲近度可以用来检测异常(样本与其他的很不相似)，或者用来聚类(把相似的样本聚在一起)。

【501~502】

随机森林代码

我们已经熟悉了ML库的工作方法，随机森林的方法也不例外。它首先是一个参数

① 这意味着一些数据样本可能随机重复。

结构 CvRTParams，该结构继承于决策树：

```
struct CvRTParams : public CvDTreeParams {

    bool calc_var_impo_rtance;
    int nactive_vars;
    CvTermCriteria term_crit;

    CvRTParams() : CvDTreeParams(
        5, 10, 0, false,
        10, 0, false, false,
        0
    ), calc_var_importance(false), nactive_vars(0) {

        term_crit = cvTermCriteria(
            CV_TERMCRIT_ITER | CV_TERMCRIT_EPS,
            50,
            0.1
        );
    }

    CvRTParams(
        int _max_depth,
        int _min_sample_count,
        float _regression_accuracy,
        bool _use_surrogates,
        int _max_categories,
        const float* _priors,
        bool _calc_var_importance,
        int _nactive_vars,
        int max_tree_count,
        float forest_accuracy,
        int termcrit_type,
    );
};


```

CvRTParams 中的主要参数是 calc_var_importance，这是一个在训练中计算每个特征的变量重要性的开关(会稍微增加计算时间)。图 13-13 展示了由.../opencv/samples/c/agaricus-lepiota.date 文件中的一部分蘑菇数据计算得到的特征重要性。参数 nactive_vars 设置了随机选择的特征的大小，它一般设置为特征总数的平方根；term_crit(本章其他地方讨论过的结构体)控制着树的最大数

目。随机森林训练时，`term_crit` 中的 `max_iter` 控制着树的总数；当 `epsilon` 值小于 OOB 错误时，则停止增加新的树；`type` 告诉程序使用哪种停止标准 (`CV_TERMCRIT_ITER` | `CV_TERMCRIT_EPS`)。

Variable Name	RandomTrees	Boosting	DecisionTree
Col5			
Col20			
Col21	45.77	6.11	
Col19	39.55	4.57	
Col9	19.01		
Col13	10.02	2.24	
Col8	9.52		
Col12	9.09	0.66	
Col22	8.29	0.28	
Col7	8.08	0.10	
Col15	4.06	1.84	
Col11	3.52	0.44	
Col4	3.12		
Col14	2.98	0.25	
Col18	2.88		0.70
Col3	2.56	0.11	9.15
Col2	2.22	0.39	12.91
Col10	1.78		2.67
Col1	0.41	0.24	7.26
Col17	0.18	0.32	0.54
Col0			
Col6			
Col16			

图 13-13：使用随机森林，boosting 和决策树计算出的蘑菇数据的变量重要性：随机森林使用更重要的变量，获得了最好的预测性能(包含 20% 数据上随机选择测试机得到 100% 准确率)

随机森林训练算法和决策树训练差不多(请参考前面对 `CvDTree::train()` 的描述)，不同的是这儿使用结构 `CvRTParams`：

```
bool CvRTrees::train(
    const CvMat*      train_data,
    int               tflag,
    const CvMat*      responses,
    const CvMat*      comp_idx = 0,
    const CvMat*      sample_idx = 0,
    const CvMat*      var_type = 0,
    const CvMat*      missing_mask = 0,
    CvRTParams        params = CvRTParams()
);
```

【503 ~ 504】

OpenCV 的实例目录中提供了一个调用训练函数的例子，用于多类学习问题，参见 `.../opencv/samples/c/letter_cecog.cpp` 文件，其中的随机森林分类器名为

forest.

```
forest.train(
    data,
    CV_ROW_SAMPLE,
    responses,
    0,
    sample_idx,
    var_type,
    0,
    CvRTParams(10,10,0,false,15,0,true,4,100,0.01f,
    CV_TERMCRIT_ITER)
);
```

随机森林预测函数的形式也和决策树差不多，但是它的返回值是一个森林中所有的树的返回值的平均值。参数 missing 是可选的，维数与 sample 向量一致，非 0 值表明 sample 中丢失特征。

```
double CvRTrees::predict(
    const CvMat* sample,
    const CvMat* missing = 0
) const;
```

【504~505】

文件 *letter_recog.cpp* 中的预测函数调用如下。

```
double r;
CvMat sample;

cvGetRow( data, &sample, i );

r = forest.predict( &sample );
r = fabs((double)r - responses->data.fl[i]) <= FLT_EPSILON ?1: 0;
```

在这段代码中，返回值 r 被转换成整数来标识是否正确预测。

最后，有一些随机森林的分析和工具函数。假设在训练时已设置 CvRTParams::cal_var_importance，则可以通过如下函数获得每一个变量的相对重要性。

```
const CvMat* CvRTrees::get_var_importance() const;
```

图 13-13 展示了随机森林计算出来的蘑菇数据的变量重要性一个例子。我们还可以通过以下函数调用获得两个数据样本在随机森林模型中的相似度。

```

float CvRTrees::get_proximity(
    const CvMat* sample_1,
    const CvMat* sample_2
) const;

```

前面已经说明，返回值为 1 标志着数据样本是同一的，0 意味着数据样本完全不同。从与训练集近似的分布中取得的两个点的相似度一般在 0 和 1 之间。

另外两个函数可以获得树的总数以及指定决策树的数据结构：

```

int get_tree_count() const; // How many trees are in the forest
CvForestTree* get_tree(int i) const; // Get an individual
decision tree

```

使用随机森林

我们已经说过：随机森林算法经常获得最好(或者最好的之一)的性能，但是最好的策略仍然是在训练集上尝试很多分类器。我们在蘑菇数据集合中使用了随机森林、boosting 和决策树。从 8124 个数据样本中，我们随机抽取了 1624 个测试样本，其他的作为训练样本。在训练了这三个基于树的分类器之后，我们获得了表 13-4 的测试结果。蘑菇数据集合相当简单，所以尽管随机森林的性能最好，我们也不能言之凿凿地说随机数分类器在这个实例中是最好的。

【505~506】

表 13-4：三个基于树的分类器对 OpenCV 蘑菇数据集(使用 1624 个随机选择的样本作为测试样本，没有对错误分类毒蘑菇进行惩罚)的实验结果

分类器	识别率
随机森林	100%
AdaBoost	99%
决策树	98%

如图 13-13 所示，最有意思的是变量重要性(也可通过分类器测得)。图中可以看出，随机森林和 boosting 各自所用的重要变量都明显少于决策树。重要性超过 15% 的特征中，随机森林只使用了 3 个变量，boosting 使用了 6 个，决策树却使用了 13 个。因此，我们可以收缩特征集大小，在不损失性能的前提下减少计算量和内存使用。当然，在决策树中，只需要建立一棵树，而在随机森林和 AdaBoost 算法中需要建立多棵树；因此，哪个算法拥有最少的开销与数据有关。

人脸识别和 Haar 分类器

现在，我们转到 OpenCV 中最后一个基于树的技术：Haar 分类器，它建立了 boost 篩选式级联分类器。它与 ML 库中其他部分相比，有不同的格局，因为它是在早期开发的，并完全可用于人脸检测。因此，我们仔细研究它，并说明它是怎么识别出人脸和其他刚性物体的。

计算机视觉是一个涉及广泛而且发展迅速的领域，所以 OpenCV 中某个特定技术很容易过时。人脸检测就用这样的过时风险。但是人脸检测有巨大的需求，因此需要有一个不错的基线技术供使用；而且人脸检测是建立在最经常使用的分类器 boosting 上，因此更加通用。事实上，一些公司使用了 OpenCV 中的“人脸”检测器来检测“基本刚性的”物体(脸，汽车，自行车，人体)。他们通过成千上万的物体各个角度的训练图像，训练出新的分类器。这个技术被用来设计目前最优的检测算法。因此，对于此类识别任务，Haar 分类器是一个有用的工具。

OpenCV 实现了人脸检测技术的其中一个版本。它是首先由 Paul Viola 和 Michael Jones 设计的，称为 Viola-Jones 检测器^①。接着，它由 Rainer Lienhart 和 Jochen Maydt 用对角特征扩展(超出本文分析范围)^②。OpenCV 称这个检测器为“Haar 分类器”是因为它使用 Haar 特征^③或更准确的描述是类 Haar 的小波特征，该特征由矩形图像区域的加减组成。OpenCV 包含一系列的预先训练好的物体识别文件，但是代码允许你训练并存储新的物体模型。除了人脸外，训练方法(`createsamples()` 和 `haartraining()`)和识别方法(`cvHaarDetectObjects()`)可以用于具有纹理的近似刚性的任何物体。

【506~507】

OpenCV 中预先训练好的物体检测器位于`.../opencv/data/haarcascades`，其中正面人脸识别效果最好的模型文件为 `haarcascade_frontalface_alt2.xml`。而侧脸却难以用该方法获得准确的检测结果(随后会简单介绍原因)。如果你训练了一个很好的物体检测器，也许可以考虑把它开源。

-
- ① P. Viola and M. J. Jones, “Rapid Object Detection Using a Boosted Cascade of Simple Features,” IEEE CVPR (2001).
 - ② R. Lienhart and J. Maydt, “An Extended Set of Haar-like Features for Rapid Object Detection,” IEEE ICIP (2002), 900–903.
 - ③ 严格来说，这种说法不对的。分类器使用矩形区域内像素的和或者差，然后通过阈值化来生成特征检测器。这里使用“类 Haar”来表明这种区别。

监督学习和 boosting 算法

OpenCV 中的 Haar 分类器是一个监督分类器(这点已经在本章开始时提到)。先对图像进行直方图均衡化并归一化到同样大小，然后标记里面是否包含要检测的物体，大多数情况下要检测的物体是人脸。

Viola-Jones 识别器使用 AdaBoost，但是把它组织为筛选式的级联分类器，每个节点是多个树构成的分类器，且每个节点的正确识别率很高(例如 99.9%，也就是很低的错误拒绝率，一般不会把人脸丢掉)，但正确拒绝率很低(接近 50%，也就是高的错误接收率，很多非人脸不会被检测出来)。在任一级计算中，如果一旦获得目标“不在类别中”的结论，则计算终止，算法也宣布该位置上没有人脸。因此只有通过分类器中的所有级别，才会认为物体被检测到。这样的优点是当目标出现频率较低的时候(例如一幅大图里只有一幅小人脸)，筛选式的级联分类器可以显著地降低计算量，因为大部分被检测的区域都可以很早被筛选掉，迅速判断出此处无人脸。

Haar 级联中的 boosting

boosting 分类器算法在本章前面已经讨论过。Viola-Jones 的筛选式级分类器中，弱分类器是一个个多数情况下只有一层的决策树(例如“决策 stump”)。一层决策树允许下面形式的决策：判断特征 f 的值 v 大于某个阈值 t ；yes 表示可能是人脸，no 表示不是人脸：

$$f_i = \begin{cases} +1 & v_i \geq t_i \\ -1 & v_i < t_i \end{cases} \quad \text{【507~508】}$$

训练中，Viola-Jones 分类器在每个弱分类器中使用的类 Haar-like 特征个数可以设置。但是大多数情况下，我们使用 1 个特征(一个只有一个分裂的树)，最多 3 个。然后提升算法迭代地建立一个由那些弱分类器的加权和组成的强分类器。Viola-Jones 使用如下分类函数：

$$F = \text{sign}(w_1 f_1 + w_2 f_2 + \dots + w_n f_n)$$

如果加权和小于 0，则符号函数返回 -1；0，则返回 0；大于 0，则返回 1。在第一次遍历数据的时候，我们训练得到 f_1 的阈值 t_1 ，它最好地区分了输入数据。然后 boosting 算法使用得到的错误来计算投票权值 w_1 。根据传统的 AdaBoost，每个特

征向量将根据是否被正确分类，重新赋予或高或低的权值^①。一旦一个节点训练完成，剩余的数据将被用来训练下一个节点，以此类推。

Viola-Jones 分类算法

Viola-Jones 分类器在级联的每个节点中使用 AdaBoost 来学习一个高检测率低拒绝率的多层树分类器。这个算法使用了如下一些创新的特征。

1. 它使用类 Haar 输入特征：对矩形图像区域的和或者差进行阈值化。
2. 它的积分图像技术加速了矩形图像区域或矩形区域的 45 度旋转(参考第 6 章)的值的计算，这个图像结构被用来加速类 Haar 输入特征的计算。
3. 它使用统计 boosting 来创建两类问题(人脸与非人脸)的分类器节点(高通过率、低拒绝率)。
4. 它把弱分类器节点组成筛选式级联。换句话说：第一组分类器是最优，能通过包含物体的图像区域，同时允许一些不包含物体的图像通过；第二组分类器^②次优分类器，也是有较低的拒绝率；以此类推。在测试模式下，只要图像区域通过了整个级联，则认为里面有物体^③。【508】

类 Haar 特征如图 13-14 所示。在所有缩放尺度下，这些特征组成了 boosting 分类器使用的全部“原材料”。它们从原始灰度图像的积分图(参考第 6 章)中快速计算得出。

Viola 和 Jones 把每个 boosting 分类器组合成筛选式级联的一个节点，如图 13-15 所示。图中，每个节点 F_j 包含一组使用类 Haar 特征训练有没有人脸的决策树。典型情况下，节点由简单到复杂排列(先尝试简单节点)，这样可最小化拒绝图像的简单区域时的计算量。每个节点的 boosting 使节点具有高通过率。例如，在检测人脸的时候，几乎所有的人脸 99.9% 都被检测出并允许通过，但是 50% 的非人脸也得以通过。这没关系，因为 20 个节点使总识别率为 $0.999^{20} \approx 98\%$ ，而错误接收率仅为 $0.5^{20} \approx 0.0001\%$ 。

-
- ① 对正确分类的样本降低权重，对错误分类的样本增大权重，有时候这会令人困惑。其原因是 boosting 希望更加关注无法分类的样本，忽略已经指导如何分类的样本。一个更加技术的描述是 boosting 是最大化间距的。
- ② 记住，在筛选式级联里，一个节点是 Adaboost 类型的一组分类器。
- ③ 这保证了级联的运行速度可以很快，因为它一般可以在前几步就可以拒绝不包含物体的图像区域，而不必走完整个级联。

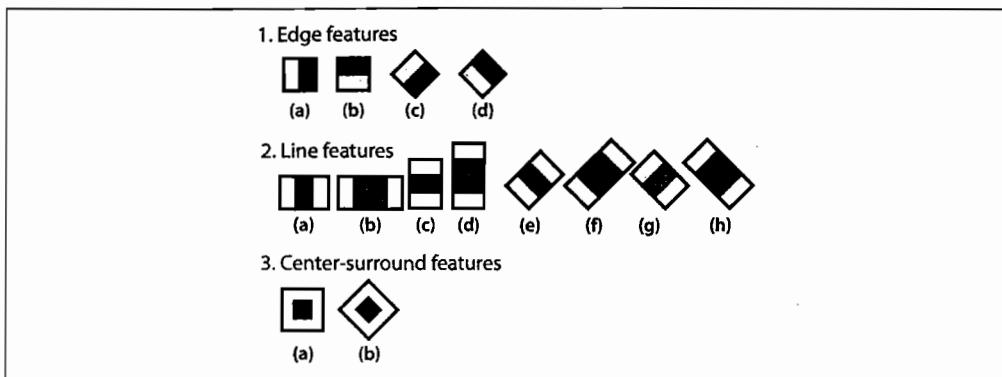


图 13-14: OpenCV 中的类 Haar 特征(矩形和旋转矩形特征都很容易从积分图中算出)。在这些小波示意图中, 浅色区域表示“累加数据”, 深色区域表示“减去该区域的数据”

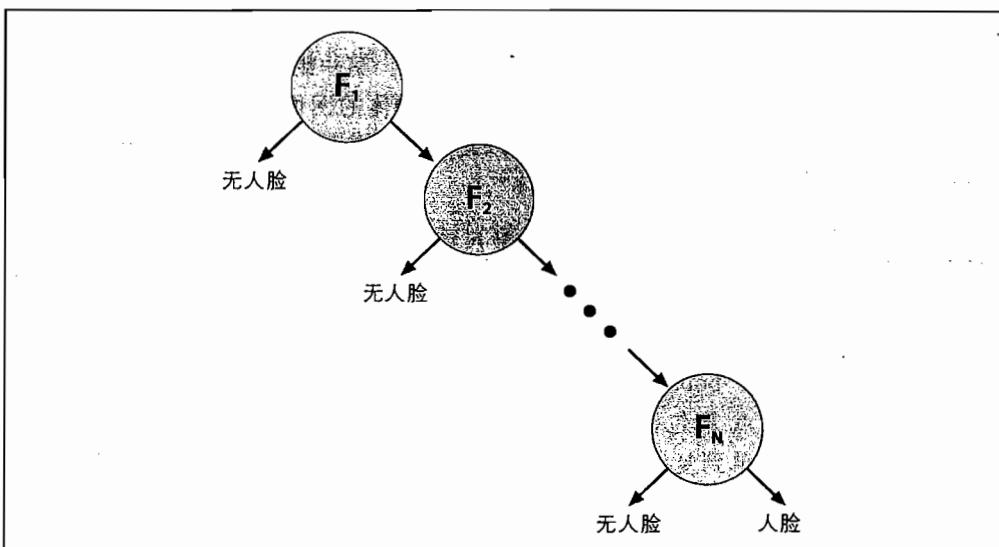


图 13-15: Viola-Jones 分类器中用到的筛选式级联。每个节点都由多个 boosting 分类器组成, 只要有人脸, 它基本上都可以检测到, 同时它只拒绝一小部分非人脸; 但是到最后一个节点, 几乎所有的非人脸都被拒绝掉, 只剩下人脸区域

识别时, 原始图像的不同区域都会被扫描。70%~80%的非人脸区域在筛选式级联的前两个节点被拒绝, 每个节点使用大约 10 个决策树。这种快速的早期拒绝提高了人脸检测的速度。

方法适用范围

此技术虽然可以用于人脸检测，并不限于人脸检测；它适用于其他外表有区别的(接近刚性的)物体的检测。正面人脸、车的前部、侧部和后部都可以用它来检测，但如果是人的侧脸和车的斜视角，则效果不好。主要是因为这些视角在模板中有很多变化，而“块特征”无法很好地处理这种变化。例如，为了让模型学习侧脸的曲线，人的侧脸肯定会包含变化的背景到模型中。可以使用 `haarcascade_profileface.xml` 来检测人的侧脸，但是如果想要得到更好的结果，需要收集更多的数据(相较于训练这个模型的数据)，也许还要包含不同背景下的侧脸数据。再次强调，侧脸是很难用这个分类器识别的，因为此分类器使用块特征，侧脸边缘外的背景也会被当作有用信息进行学习。在训练中，如果只学习右侧脸会更有效率一些。在测试时，(1)首先运行右侧脸检测器，(2)然后沿着图像的垂直轴翻转图像，再次运行右侧脸探测来检测左侧脸。

如前所述，这些类 Haar 特征对于“块特征”(眼睛、嘴、发际线)具有比较好的效果，但对树枝或者物体主要靠外形(如咖啡杯)的物体。就像我们讨论的，基于类 Haar 特征的识别器适用于固有特征——眼睛、嘴、发际线。但是不适用于树枝，或者那些外形是最有区别的特型(coffee 杯)。

如果你愿意收集有关刚性物体的大量好的，已完美分割的数据，那么这个分类器仍然可以达到很好的效果，它的筛选式级联结构使其运行速度很快(当然不是指训练时的运行速度)。这里的“大量的数据”意味着数以千计的物体实例和数以万计的非物体实例。“好数据”意味着不应该把倾斜的脸和竖直的脸混在一起；解决方法是训练两个分类器，一个用来判断倾斜，一个用来判断竖直。“完美分割”圈定物体的矩形边界要保持一致。如果物体边界四处漂移，那么分类器不得不去学习这些变化。例如，眼睛在矩形边界内的位置不同，会使分类器认为眼睛位置不是脸的固定的几何特征，是可移动的。当分类器尝试去适应实际并存在的东西时，性能一般都会变差。

人脸检测代码

例 13-4 的代码 `detect_and_draw()` 可以检测人脸，并在图中用不同颜色的矩形画出它们的位置。按照下面第 4 行到第 7 行注释，这段代码假设已经加载预先训练好的分类器，也已经为检测到的人脸创建了内存。

例 13-4：检测并标识人脸的代码

```
// Detect and draw detected object boxes on image
// Presumes 2 Globals:
// Cascade is loaded by:
// cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name,
// 0, 0, 0 );
// AND that storage is allocated:
// CvMemStorage* storage = cvCreateMemStorage(0);
//
void detect_and_draw(
    IplImage* img,
    Double scale = 1.3
){
    static CvScalar colors[] = {
        {{0,0,255}}, {{0,128,255}}, {{0,255,255}}, {{0,255,0}},
        {{255,128,0}}, {{255,255,0}}, {{255,0,0}}, {{255,0,255}}
    }; //Just some pretty colors to draw with

    // IMAGE PREPARATION:
    //
    IplImage* gray=cvCreateImage(cvSize(img->width,img->height),8,1 );
    IplImage* small_img = cvCreateImage(
        cvSize(cvRound(img->width/scale), cvRound(img->height/scale)),8,1
    );
    cvCvtColor( img, gray, CV_BGR2GRAY );
    cvResize( gray, small_img, CV_INTER_LINEAR );
    cvEqualizeHist( small_img, small_img );

    // DETECT OBJECTS IF ANY
    //
    cvClearMemStorage( storage );
    CvSeq* objects = cvHaarDetectObjects(
        small_img,
        cascade,
        storage,
        1.1,
        2,
        0 /*CV_HAAR_DO_CANNY_PRUNING*/,
        cvSize(30, 30)
    );
}
```

```

// LOOP THROUGH FOUND OBJECTS AND DRAW BOXES AROUND THEM
//
for(int i = 0; i < (objects ? objects->total : 0); i++ ) {
    CvRect* r = (CvRect*)cvGetSeqElem( objects, i );
    cvRectangle(
        img,
        cvPoint(r.x,r.y),
        cvPoint(r.x+r.width,r.y+r.height),
        colors[i%8]
    )
}
cvReleaseImage( &graygray );
cvReleaseImage( &small_img );
}

```

【511~512】

代码中函数 `detect_and_draw()` 有一个颜色向量的静态数组 `colors[]`，可以用不同的颜色标识出的人脸。分类器在灰度图像上进行检测，所以 RGB 图像首先需要通过 `cvCvtColor()` 转换成灰度图像，还可以选择 `cvResize()` 调整大小，然后通过 `cvEqualizeHist()` 进行直方图均衡。它可以平衡亮度值，因为积分图像特征基于不同的矩形区域的差别，如果直方图没有均衡，这些差别可能由于测试图像的成像条件或过分曝光而偏离正常值。因为分类器识别出的目标矩形在 `CvSeq` 序列中返回，我们需要先通过 `cvClearMemStorage()` 清除内存。实际的检测在 `for()` 循环的前面，下面将详细讲述 `cvHaarDetectObjects()` 的参数。这个循环的作用是逐个取出人脸的矩形区域，然后用不同的颜色通过函数 `cvRectangle()` 画出来。让我们先仔细看一下检测函数调用：

```

CvSeq* cvHaarDetectObjects(
    const CvArr*           image,
    CvHaarClassifierCascade* cascade,
    CvMemStorage*          storage,
    double                  scale_factor = 1.1,
    int                     min_neighbors = 3,
    int                     flags = 0,
    CvSize                 min_size = cvSize(0,0)
);

```

`CvArr image` 是一个灰度图像，如果对它设置了感兴趣的区域(ROI)，那么函数将只处理这个区域。因此，一个提高人脸识别速度的方法是使用 ROI 裁减图像边界。分类器 `cascade` 是我们通过 `cvLoad()` 加载的 Haar 特征级联。参数 `storage`

是这个算法的工作缓存。它由 `cvCreateMemStorage()` 来分配，由 `cvClearMemStorage()` 释放。函数 `cvHaarDetectObjects()` 以不同的窗口扫描输入图像寻找人脸。设置 `scale_factor` 参数可以决定每两个不同大小的窗口之间有多大的跳跃；这个参数设置得大，则意味着计算会变快，但如果窗口错过了某个大小的人脸，则可能丢失物体。参数 `min_neighbors` 控制着误检测。现实图像中的脸会被多次检测到，因为周围的像素和不同大小的窗口也会检测到人脸。在人脸识别代码中设置这个参数为默认值(3)表明只有至少有 3 次重叠检测，我们才认为人脸确实存在。参数 `flag` 有 4 个可用的数值，它们可以用位或操作结合使用。第一个是 `CV_HAAR_DO_CANNY_PRUNING`，这个值告诉分类器跳过平滑(无边缘)区域。第二个值是 `CV_HAAR_SCALE_IMAGE`，这个值告诉分类器不要缩放分类器，而是缩放图像(处理好内存和缓存的使用问题，这可以提高性能)。第三个值是 `CV_HAAR_FIND_BIGGEST_OBJECTS`，告诉分类器只返回最大的目标(这样返回的物体个数只可能是 1 个或者 0 个)^①。最后是 `CV_HAAR_DO_ROUGH_SEARCH`，它只可与 `CV_HAAR_FIND_BIGGEST_OBJECTS` 一起使用，这个标志告诉分类器在任何窗口，只要第一个候选者被发现则结束寻找(当然需要足够的相邻区域来说明真正找到了目标)。参数 `min_size` 指示寻找人脸的最小区域。设置这个参数过大，会以丢失小物体为代价减少计算量。图 13-16 展示了在包含人脸的场景中使用人脸识别代码的结果。

【512~513】

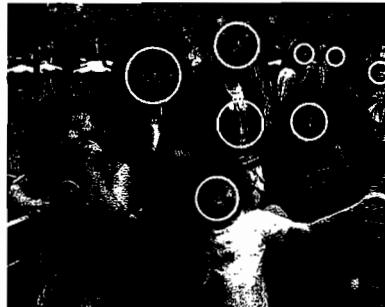


图 13-16：公园场景的人脸检测。一些旋转的脸没有被检测到，另外也有误检测(中间的衣服部分)；对于这个 1054x851 图像，在 2 GHz 的计算机上运行了 1.5 秒，完成百万次搜索后得到了这个结果

① 最好不要使用 `CV_HAAR_DO_CANNY_PRUNING` 和 `with CV_HAAR_FIND_BIGGEST_OBJECT`，虽然这两个标志可以降低计算量，但是引起的副作用往往会导致性能降低。

学习新的物体

我们已经知道怎么加载和运行一个预先训练并存储在 xml 文件中的沉降分类器。我们首先使用 cvLoad() 来加载它，然后使用 cvHaarDetectObjects() 来找到与训练目标相似的物体。现在我们转到怎么训练分类器(如眼睛、行人、车等)。我们可以使用 OpenCV 的 haartraining 应用程序，它从给定的训练集训练出分类器。训练一个分类器的 4 个步骤如下。(更多问题，请参考 haartraining 参考文档 *opencv/apps/HaarTraining/doc/haartraining*)

1. 收集打算学习的物体的数据集(如正面人脸图，汽车侧面图)，把它们存储在一个或多个目录下面。这些文件通过一个文本文件以下面的形式建立索引：

```
<path>/img_name_1 count_1 x11 y11 w11 h11 x12 y12 . . .
<path>/img_name_2 count_2 x21 y21 w21 h21 x22 y22 . . .
. . .
```

文本文档的每一行包含路径(如果有路径的话)和文件名，紧接着是物体的个数和物体的矩形列表，矩形的格式是(左上角的 x 和 y 坐标，宽度，高度)。

具体说来，如果数据集在路径 *data/faces/* 下，那么索引文件 *face.idx* 将如下所示：

```
Data/faces/face_000.jpg 2 73 100 25 37 133 123 30 45
Data/faces/face_001.jpg 1 155 200 55 78
. . .
```

如果希望分类器充分发挥作用，则需要收集很多高质量的数据(1000~10 000 个正样本)。高质量是指已经把所有不需要的变量从数据中除掉。举个例子，如果你学习人脸，需要尽量对齐眼睛(最好加上鼻子和嘴巴)。除非告诉分类器眼睛不可以移动，否则它会认为眼睛可以出现在任意区域内。但是这样是不符合实际情况的，分类器将会无法取得好的效果。一个策略是首先训练一个容易锁定的子集(如眼睛)的级联。然后使用这个级联来寻找眼睛，可以旋转/改变图像大小直到眼睛被对齐。对于非对称数据，可以使用在前面讨论的沿竖直轴翻转图像的窍门。

【513~515】

2. 使用辅助工具 *createsamples* 来建立正样本的向量输出文件。通过这个文件，便可以重复训练过程，使用同一个向量输出文件尝试各种参数。例如：

```
createsamples -vec face.vec -info face.idx -w 30 -h 40
```

这个程序读入第一步所说的 *face.idx* 文件，输出一个格式化的训练文件 *face.vec*。然后 *createsamples* 提取图像中的正样本，再归一化并调整到指定的大小(这里是 30×40)。注意，*createsamples* 还可以对数据进行几何变换、增加噪声、改变颜色等来合成数据。这个程序还可以学习一个公司的 logo，只需要照一张照片，然后让它进行很多真实世界存在的变化。更多细节请参考 OpenCV 的 *haartraining* 手册/*apps/HaarTraining/doc/haartraining*。

3. Viola-Jones 级联是一个两类分类器：它只判断图像中的物体是否(“是”还是“否”)与训练集相似。我们已经说明怎么收集和处理正样本。现在我们说明怎么收集和处理反样本。任何没有我们感兴趣的物体的图像都可以作为反样本。最好从我们需要测试的数据中选取反样本图像。即，如果我们想从在线视频中学习人脸，最好从视频的不包含人脸的帧中获得反样本。然而从其他地方(光盘上或者互联网相册中)获得反样本也可以获很好的最终结果。同样，我们把反样本图像放到一个或者几个路径下面，并由图像文件名组成索引文件，一个文件名占一行。比如，一个叫 *background.idx* 的图像索引文件可以包含下面的路径和文件名：

```
data/vacations/beach.jpg  
data/nonfaces/img_043.bmp  
data/nonfaces/257-5799_IMG.JPG  
..
```

4. 训练。下例从命令行输入或者使用批处理文件创建的训练。

```
Haartraining /  
-data face_classifier_take_3 /  
-vec faces.vec -w 30 -h 40 /  
-bg backgrounds.idx /  
-nstages 20 /  
-nsplits 1 /  
[-nonsym] /  
-minhitrate 0.998 /  
-maxfalsealarm 0.5
```

【515】

执行之后，获得的分类器将存储在文件 *face_classifier_take_3.xml* 中。这里 *face.vec* 是正样本(归一化为宽 30，高 40)，反样本是从 *backgrounds.idx* 随机抽取的图像。级联被设置为有 20 层(*-nstages*)，每一层的最低正确检测率(*-minhitrate*)是 99.8%，错误接受率(*-maxfalsealarm*)是 50%。弱分类器是被指定为“stumps”，即只有一层分裂(*-nsplits*)的决策树，如果设置为多

层，在某些情况下是可以提高准确率的。对于一些复杂物体，最多可以使用 6 层决策树，但是大多数情况下，我们将这个参数设置得比较小，一般不超过 3 层。

即使在一台比较块的计算机上面，根据数据集的大小，训练可能需要几个小时，甚至一天。训练程序需要在每个正样本和反样本上测试大约 100 000 个特征。这个程序可以在多核机器上并行执行(通过 Intel 编译器和 OpenMP 实现)。OpenCV 中已经包含了并行版本。

其他机器学习算法

现在我们已经熟悉 OpenCV ML 库的工作原理。ML 库的架构使新算法和新技术比较容易集成到库里面去。更多其他的新算法会很快添加进去。本节将简要介绍刚刚添加到 OpenCV 的 4 个新机器学习算法。每个算法都是一个广泛使用的机器学习算法。这些算法被人们广泛研究，有很多书、论文和网络资料都是关于这些算法的。想了解更多算法细节，请参考相应的文献和参考手册 [.../opencv/docs/ref/opencvref_ml.htm](http://opencv/docs/ref/opencvref_ml.htm)。

期望值最大化

期望值最大化(expectation maximization, EM)算法是另一个常用的聚类技术。OpenCV 仅支持混合高斯模型的期望值最大化，EM 算法本身更通用。EM 算法通过迭代先找到给定模型时的最大可能性的猜想，然后调整模型使猜想正确率最大化。OpenCV 中 EM 算法实现在类 CvEM{} 中，仅包括用混合高斯来拟合数据。用户需要提供高斯的个数，所以此算法与 K 均值很相似。

K 近邻

K 近邻(K-nearest neighbor, KNN)是最简单的分类器技术之一。它只存储所有训练样本数据，如果需要分类一个新的数据样本，只需要找到它的 K(K 是整数)个最相邻的点，然后统计哪个类在这 K 近邻点中频率最高，然后把该点标记为出现频率最高的类。这个算法实现在类 CvKNeast{} 中。KNN 分类器技术很有效，但是它需要存储所有训练集，因此它占用很大的内存，速度比较慢。使用这个算法之前把训练集聚类来降低数据的大小。读者如果对大脑中的(和机器学习中的)动态适应最近

邻类型技术感兴趣，可以阅读 Grossberg 的论文[Grossberg87]，或者 Carpenter 和 Grossberg 的最新的综述[Carpenter03]。

【516~517】

多层感知器

多层感知器(MLP，也叫反向传播)是一种神经网络，是性能最好的分类器之一，尤其在文字识别方面性能卓越。它训练的时候很慢，因为它使用梯度下降调整神经网络层节点之间的连接权来最小化误差。测试模式时，它的速度很快：仅仅一些点乘运算，然后是一个压缩函数。OpenCV 中，它实现在类 CvANN_MLP{} 中，用法可以参见文件 .../opencv/samples/c/letter_recog.cpp。有兴趣的读者可以阅读 LeCun, Botton, Bengio 和 Haffner[LeCun98a]的论文来了解 MLP 在文字识别和物体识别方面的效能。LeCun, Botton 和 Muller[LeCun98b]的论文中介绍了实现和调节细节。分等级的类脑网络的最新工作可以参考 Hinton, Osindero 和 Teh 的论文 [Hinton06]。

支持向量机

有很多数据的时候，boosting 和随机森林算法常常是最好的分类器。但是当数据集和比较小的时候，支持向量机(SVM)效果常常最好。这个 N 类分类算法首先把数据映射到高维空间(创建超过特征空间的组合的新空间)，然后在高维空间找到类别间最优的线性分类器。在原始数据的原始空间中，这种高维线性分类器可能是非线性的。因此我们采用基于最大类间隔的线性分类技术，得到在某种意义上较优地区分类别的非线性分类器。有了足够的增加的维数，我们几乎可以总能把数据很好地分成不同类别。这个技术实现在类 CvSVM{} 中。

这些工具与很多计算机视觉算法紧密相连，这些算法包括从通过训练好的分类器寻找特征样本实现跟踪，到分割场景，还包括更直接的任务：物体分类和图像数据聚类。

【516~517】

练习

1. 考虑根据以前的股票价格来学习未来的股票价格。假设有 20 年的每日股票数据。在把数据转换成训练集和测试集时，下面的转换方法各有哪些优缺点？
 - a. 把偶数的点作为训练集，奇数的点作为测试集。

- b. 随机选择测试集和训练集。
- c. 把数据分为两部分，一半用来训练，另一半用来测试。
- d. 把数据分为很多小窗口，每个小窗口包含一些过去的数据样本和一个预测样本。
2. 图 13-17 描述了一个“true”和“false”两类分布，图中也展示了一些可以设置阈值的点(a, b, c, d, e, f, g)。
- 在 ROC 曲线上标记出点 a 到 g。
 - 如果“true”类是有毒的蘑菇，阈值应设置为哪个数字？
 - 决策树对这些数据样本进行分类，效果会怎样？

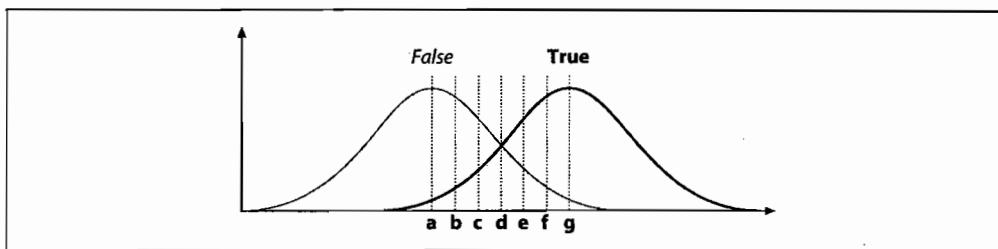


图 13-17：两个类别(“true” 和 “false”)的高斯分布

3. 参考图 13-1，回答下列问题。
- 画出决策树怎样用三次分裂拟合真实的函数(虚线)。这是一个回归问题，不是分类问题。
 - 画出决策树怎样用七次分裂拟合真实数据。
 - 画出决策树怎样用七次分裂拟合噪声数据。
 - 讨论 b 和 c 在过拟合方面的区别。
4. 为什么在单个决策树学习多类问题的时候，分裂方法(例如基尼——Gini)仍然有效？
5. 参见图 13-4，它描述了不等方差(左图)和相等方差(右图)下的二维空间。把它们视为与分类问题有关的特征值。假如这些特征值用于分类问题。在左图的空

间和右图的空间中，使用如下分类器时，变量重要性会有所不同么？

- a. 决策树。
 - b. K 近邻。
 - c. 朴素贝叶斯。
6. 修改例 13-1(在 K 均值部分，最外层 `for()` 循环前面)的数据生成代码，以生成一个有随机标签的数据集。我们使用一个以(63,63)为中心，标准差为(`img->width/6, img->height/6`)的正态分布来生成 10 000 个数据样本，图像大小是 128×128 。给数据作标签的时候，我们把空间分成 4 个象限。 $x < 64$ 的时候，20%的概率标记为 A； $x > 64$ 的时候，90%的概率标记为 A。 $y < 64$ 的时候，40%的概率标记为 A； $y > 64$ 的时候，60%的概率标记为 A。然后对 x 和 y 维度上的概率相乘，得到类别 A 在 4 个象限内的概率。如果不标记为 A，则标记为 B。例如当 $x < 64$ 且 $y < 64$ 时，样本有 8% 的可能性为 A，92% 的可能性为 B。则 4 个象限中标记为 A 的概率如下：
- | | |
|-------------------------|-------------------------|
| $0.2 \times 0.6 = 0.12$ | $0.9 \times 0.6 = 0.54$ |
| $0.2 \times 0.4 = 0.08$ | $0.9 \times 0.4 = 0.36$ |
- 对每一个数据样本，首先决定它的象限，再生成一个随机数，如果随机数小于等于象限概率，则标记为 A；否则标记为 B。这样，我们就获得了以 x 和 y 作为特征的带标签数据样本。读者可能意识到 x 轴比 y 轴更适合用来划分数据类别。训练一个随机森林来验证 x 的变量重要性确实比 y 重要。
7. 使用与练习 6 一样的数据集，使用离散 AdaBoost 学习两个模型：一个设置 `weak_count` 为 20，一个设置为 500。从 10 000 个数据样本中随机选取测试集和训练集。通过下面指定的样本数训练模型并记录结果：
- a. 150 个数据样本。
 - b. 200 个数据样本。
 - c. 1200 个数据样本。
 - d. 5000 个数据样本。
 - e. 对结果做出解释。
8. 重复练习 7，使用随机森林分类器，树的个数为 50 个和 500 个。
9. 重复练习 7，但是使用 60 棵树，并比较随机森林和 SVM。
10. 在什么情况下，对于过拟合，随机森林比决策树具有更强的鲁棒性？

11. 参考图 13-2, 你能想象什么时候测试错误会小于训练错误吗?
12. 图 13-2 是一个回归问题。把第一个点标记为 A , 第二个点 B , 第三个点 A , 第四个点 B , 如此继续。画出 A 和 B 的一条分界线, 分别演示:
 - a. 欠拟合。
 - b. 过拟合。
13. 参考图 13-3。
 - a. 画出最好可能情况下的 ROC 曲线。
 - b. 画出最坏可能情况下的 ROC 曲线。
 - c. 画出在测试集上性能随机的 ROC 曲线。
14. “没有免费的午餐” 法则的意思是没有哪个分类器在各种分布的标签数据上都是最优的。请描述一个有标签的数据分布, 本章描述的分类器没有一个能在这方面取得好结果。
 - a. 朴素贝叶斯分类器难以处理哪种分布?
 - b. 决策树难以处理哪种分布?
 - c. 怎样预先处理(a)和(b)中的分布使分类器能够更容易地学习数据?
15. 在 USB 摄像机上设置并运行 Haar 分类器来检测人脸:
 - a. 它可以处理多少次尺度变化?
 - b. 它可以处理多大的噪声?
 - c. 它能最大能处理头多大的倾斜?
 - d. 它能最大能处理下巴多大角度的上下运动?
 - e. 它能最大能处理头多大角度的左右旋转?
 - f. 它能能处理头的三维姿势变化吗?
16. 使用蓝色或绿色背景采集展开的手姿态(静态姿态)。收集其他的手的姿态, 并用随机背景采集。收集上百张图像训练 Haar 分类器来检测展开的手姿态。实时测试分类器并估计检测率。
17. 根据你的知识和从练习 16 中学到的, 改进其结果。

OpenCV 的未来

过去与未来

在第 1 章中，我们了解了 OpenCV 的过去。随后的第 2 章~第 13 章详细介绍了 OpenCV 现有的功能。现在我们介绍 OpenCV 的未来。计算机视觉应用最近迅速发展，从产品检测到互联网图像和视频的检索，到医学应用甚至在火星上的局部导航。OpenCV 也在成长以适应这些领域里的发展。

OpenCV 在很长一段时间里一直得到 Intel 公司的支持，最近得到 Willow Garage(www.willowgarage.com)的支持。Willow Garage 是一个新成立的私立机器人研究所和技术孵化器。Willow Garage 将通过开发开放且有支持的硬件和基础软件来促进民用机器人的迅速发展，基础软件包括但并不仅限于 OpenCV。这使得 OpenCV 获得了新的资源，能够更加快速地更新以及提供支持。OpenCV 的几个原始开发者愿意继续帮助维护和促进这个库的发展。这些新的资源可以加快代码评估和集成周期，使得更多的社区贡献代码进入 OpenCV。

OpenCV 的一个关键的新领域是机器人感知。OpenCV 将致力于三维感知。另外，既然数据类型的混合能够为物体检测、分割和识别提取出更好的特征，那么同样也会关注二维加三维的物体识别。机器人感知非常依赖于三维传感，所以正在扩展这些功能：摄像机标定、多摄像机的校正和匹配，以及摄像机和激光测距仪的融合（参考图 14-1）^①。

① 在本书编写过程中，这些方法依然在开发中，尚未进入 OpenCV。

如果商业上有硬件支持，“雷达 + 摄像机标定”工作将会被推广到一些设备中，如激光雷达和红外波前设备等。另外还会针对使用结构光或者激光进行三角测量，以获取精确的深度信息。大部分深度测量方法获得的原始输出都是三维点云。把从三维深度测量中获得的原始点云转换为三维面片是下一步的工作。三维面片可以用来获取物体的三维模型，从三维中分割出物体，因此可以让机器人具有理解和处理这些物体的能力。通过外部三维感知到内部三维图形，机器人能够规划路径，使自己能够平滑移动；或者让机器人能够识别、处理和移动物体。

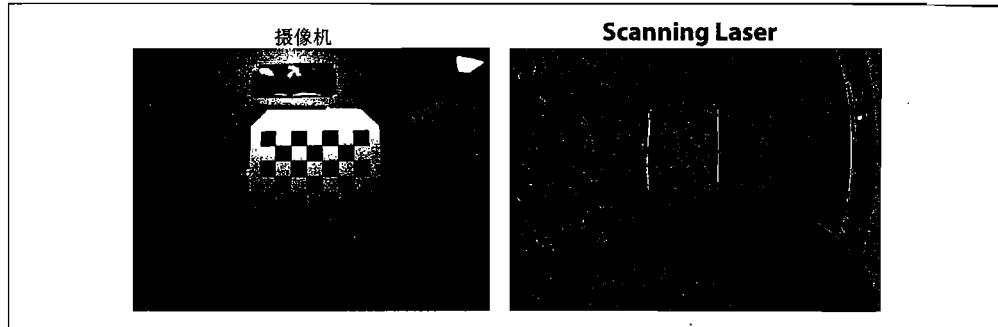


图 14-1：新的三维成像融合方法：使用从激光深度扫描仪(右图)返回的深度信息标定一个摄像机(左图)。(所用图像已获得 Hai Nguyen 和 Willow Garage 许可)

【521 ~ 522】

在探测三维物体的同时，机器人还需要识别三维物体和物体的三维姿态。为了实现这些功能，将会设计几个可扩展的二维加三维的物体识别方法。要制作强大的机器人，需要的知识会涵盖计算机视觉和人工智能的大部分领域，从精确的三维重建到跟踪、识别别人的身份、物体识别和图像拼接，以及学习、控制、路径规划和决策。如果能够迅速并精确的感知深度信息和识别物体，那么任何高层的任务，例如路径规划，将会变得更容易。在这些领域里，特别是 OpenCV 关注的领域，可以通过鼓励很多小组贡献和使用更好的方法来解决实际应用中的感知、识别和学习方面的难题，来实现迅速发展。

OpenCV 也将支持许多其他领域的应用，例如网络上的图像和视频索引，安全监控系统，医学图像分析。整个社区的意愿将会影响 OpenCV 的方向和发展。

发展方向

虽然 OpenCV 关注的不仅仅是实时的算法，但是它将会继续倾向实时处理技术。

没有人能够非常肯定地讲出未来的计划，但是可以提一下下面这些高优先级的领域。

- **应用** 与底层功能相比，真正可行的应用有更大的需求。例如，更多的人将使用一个全自动的立体视觉解决方案而不是一个更好的亚像素级角点检测方法。几个更完整的应用将被加入，例如可扩展的单摄像机到多摄像机标定和校正以及三维深度显示界面。
- **三维** 如前面所述，OpenCV 中将会有更好的三维深度探测，以及二维摄像机和三维测量仪器的融合，还会有更好的立体视觉算法。还可能会增加结构光算法。
- **密集光流** 因为我们想了解整个物体是如何移动的(并部分支持三维)，OpenCV 一直在计划实现 Black 的密集光流算法[Black96]。
- **特征** 为了能够更好的实现物体识别功能，大家都在期待有一个全功能的工具套件，这个套件能够有一个好的框架，可以方便的换用各种不同的关键点检测算法，以及关键点描述算法。这里面将会包含一些流行的特征，如 SURF、HoG、Shape Context、MSER、Geometric Blur、PHOG、PHOW 及其他特征。二维和三维特征的支持也在计划中。
- **基础功能** 将增加一个类封装^①，一个好的 Python 接口，增强的 GUI，更好的文档，更好的错误处理机制，增强对 Linux 的支持，等等。
- **摄像机接口** 正在计划更多摄像机的无缝支持，还有最后可能的宽动态摄像机的支持。目前大多数摄像机每个颜色通道只支持 8 位，但是一些新的摄像机可以支持每个通道 10 或 12 位^②。[207]对于物体识别和立体视觉匹配，这些摄像机的宽动态可以取得更好的结果，因为它能够检测精细的纹理和颜色，而很多老式的窄动态摄像机却看不到这些。

【523】

① Daniel Filip 和 Google 已经为 OpenCV 贡献了一个很快的轻量级图像类封装 WImage，他们开发这个类用于内部使用。本书出版的时候，该类将进入 OpenCV，但是目前无法在本书中给出该类的文档。

② 许多昂贵的摄像机宣称能够支持 16 位，但是作者还没有发现超过 10 位的有用分辨率，其余的数据位为噪声。

具体发展方向

计算机视觉的许多物体识别技术需要检测显著区域，显著区域是在不同视图中变化很小的区域。可以使用一些特征来标记这些显著区域^①[271]，例如一个显著点周围的图像梯度方向直方图。虽然本节中提到所有技术都可以使用现有的 OpenCV 函数来实现，但是 OpenCV 目前缺少流行的感兴趣区域检测和特征描述算法的完整实现。

OpenCV 包含一个高效的 Harris 兴趣角点检测，但是缺少 David Lowe[Lowe04]设计的流行的“缩放不变的最大拉普拉斯”检测，最大稳定极值区域(MSER)[Matas02]检测，以及其他检测算法。

OpenCV 还缺少许多可以识别显著区域的流行算法，如 SURF 梯度直方图网格[Bay06]。另外，我们希望能够包含一些其他的特征，比如梯度方向直方图(HoG)[Dalai05]、几何模糊[Berg01]和偏移图像块(Torralba07)，能够快速计算的密集高斯尺度变化梯度(DAISY)[Tola08]，梯度位置和方向直方图(GLOH)[Mikolajczyk04]；另外，虽然申请了专利，我们仍希望添加尺度不变特征变换(SIFT)[Lowe04]描述子作为参考。其他的一些通过学习得到的不错的特征描述子有学习获取的具有方向的小块[Hinterstoisser08]和学习获得的比例点[Ozusy07]。我们也希望能够增加上下文特征或元特征有金字塔匹配核[Grauman05]，其他特征的金字塔直方图内嵌，PHOW[Bosch07]，Shape Context[Belongie00; Mori05]，或者其他通过统计空间分布来定位特征的方法[Fei-Fei98]。最后，是一些全局特征，它们可描述整个场景，可以通过上下文来级联识别[Oliva06]。这些有很高的优先级，希望 OpenCV 社区能够为这些功能或其他功能开发和贡献代码。

通过大数据量学习，使用简单有效的最近邻匹配便可识别出物体，很多研究组在这个方面获得了令人鼓舞的结果[Nister06; Philbin07; Torralba08]，所以可考虑加入一个高效的最近邻框架。

对于机器人应用，我们需要物体识别(什么)和物体定位(哪里)。所以需要考虑增加 Shi 和 Malik 的分割方法[Shi00]，并可以实现快速算法[Sharon06]。最近一些算法可以通过学习的方法来同时实现识别和分割[Oppelt08; Schroff08; Sivic08]。光照的方向[Sun98]和形状信息也可提供很大帮助[Zhang99; Prados05]。

① 也被称作兴趣点。

除了开发好的特征算法，还要开发三维传感算法，它可以用来实现虚拟测量和虚拟 SLAM(同时定位与构建地图)。如果能够获取足够精确的深度信息以及用于识别的特征，便可以实现更好的导航和三维物体处理。为了生成更好的三维物体训练数据，也可创建一个特制的视觉接口，来连接光线跟踪(ray-tracing)软件(例如开源的 Manta 光线跟踪软件[Manta])。

【524~525】

机器人、安全系统以及网络图像和视频搜索都需要拥有识别物体的能力，所以 OpenCV 必须优化机器学习库中的模式匹配算法。具体说来，OpenCV 应该首先简化学习算法的接口，然后设置好的默认参数，以使得算法易于上手。一些新的学习算法需要加入，它们中的一些需要能够同时处理两类和多类问题(就像现在 OpenCV 中的随机森林那样)。另外需要可扩展的识别技术，这样用户就不需要为每一个类别学习一个全新的模型。还有就是应该设计更多的功能，使得机器学习分类器能够处理深度信息和三维特征。

在计算机视觉中，马尔可夫随机场(MRFs)和条件随机场(CRFs)正越来越流行。但这些方法通常仅仅对具体某个问题可用，我们正在尝试让它们能够更灵活地使用。

我们希望开发出能够学习互联网数据库或移动机器人采集的数据库的方法，也许可以根据 Zisserman 的建议采用近似最近邻(approximate nearest neighbor)技术来处理百万或数亿个数据样本，这一点在前面提到过。类似地，我们需加速的 boosting 和 Haar 特征训练方法来处理更大的数据库。目前 ML 部分的几个方法需要把所有数据放入内存中处理，这严重影响了它们在大数据库上的应用。OpenCV 需要打破此类限制。

OpenCV 也需要比目前更好的文档。当然此书对于丰富文档有帮助，但是 OpenCV 参考手册需要一次彻底整理，并增加搜索功能。处于高优先级的还有增强 Linux 支持，以及更好的外部语言接口——特别是用 Python 和 Numpy 方便地进行视觉编程。我们也要保证机器学习库能够直接从 Python 直接调用，也能从 Python 的 SciPy 包和 Numpy 包调用。

为了在开发者社区有更好的互动，也许应该在一个重要的视觉会议上召开开发者研讨会。另外需要大量的工作来推动“grand challenge”竞赛，也需要一定数量的奖金。

OpenCV 与艺术家

目前有一个世界范围的互动艺术家社区，参观者可以动态地与他们的作品交互。这

一应用最常用的功能是人脸检测、光流和跟踪。我们希望本书能够让艺术家们更好的理解 OpenCV，并使用 OpenCV 创作出更好的作品。我们相信以后的好的深度探测会使用户交互更丰富、更稳定。因为物体能被用于模态控制，更好的物体识别功能将会产生多种与作品的交互方式。如果能够获取三维网格，那么就可以将参观者“导入”到艺术作品中，艺术家可以通过识别用户动作获取更好的感觉；同样也可以用来提升动态交互。在 OpenCV 的未来发展中，将会越来越重视艺术家社区使用计算机视觉的要求和期望。

【525~526】

后记

本书涉及很多理论和很多实际问题，我们也描述了未来的计划。当然当我们开发软件的时候，硬件也在发展。摄像机现在更便宜了，已经装在手机到交通灯等各种设备中。一些厂商正在生产手机投影仪，这对机器人是一件好事，因为大部分手机都很轻且功耗低，手机的电路大都集成了摄像机。手机投影仪可以发射出结构光，因此可以获取精确的深度图。精确的深度图是操纵机器人和三维物体扫描必不可少的。

本书的两位作者都参与了斯坦福大学的机器人运动员 Stanley 的视觉系统的设计，Stanley 曾在 2005 年 DARPA Grand Challenge 竞赛中获胜。在 Stanley 上，视觉系统与一个激光扫描仪配合，完美地跑完了七个半小时的沙漠路程[Dahlkamp06]。这让我们意识到视觉与其他感知系统结合的力量：通过将视觉与其他方式的感知结合，之前不可解的道路感知问题变成了一个可解问题。通过本书让视觉变得更容易且易于使用，其他人可以将视觉加入到他们自己的解决问题的工具中，然后发现一个解决重要问题的新方法，这就是我们的希望。也就是使用普通的摄像机和 OpenCV，人们可以解决实际问题，例如汽车倒车安全系统、新游戏控制和新安全系统中使用的立体视觉。请尽情天马行空地修改代码来解决问题！

计算机视觉未来将大有发展，它看上去似乎是二十一世纪关键技术之一。同样地，OpenCV 看上去似乎是(至少部分是)计算机视觉的关键推动力之一。在前方有无尽的创新和贡献的机会。我们希望本书能够鼓励所有有志加入计算机视觉领域的人，激发他们的兴趣，让他们加入这令人兴奋的领域。

【526】

参考文献

- [Acharya05] T. Acharya and A. Ray, *Image Processing: Principles and Applications*, New York: Wiley, 2005.
- [Adelson84] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, “Pyramid methods in image processing,” *RCA Engineer* 29 (1984): 33–41.
- [Ahmed74] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” *IEEE Transactions on Computers* 23 (1974): 90–93.
- [Al-Haytham1038] I. al-Haytham, *Book of Optics*, circa 1038.
- [AMI] Applied Minds, <http://www.appliedminds.com>.
- [Antonissee82] H. J. Antonisse, “Image segmentation in pyramids,” *Computer Graphics and Image Processing* 19 (1982): 367–383.
- [Arfken85] G. Arfken, “Convolution theorem,” in *Mathematical Methods for Physicists*, 3rd ed. (pp. 810–814), Orlando, FL: Academic Press, 1985.
- [Bajaj97] C. L. Bajaj, V. Pascucci, and D. R. Schikore, “The contour spectrum,” *Proceedings of IEEE Visualization 1997* (pp. 167–173), 1997.
- [Ballard81] D. H. Ballard, “Generalizing the Hough transform to detect arbitrary shapes,” *Pattern Recognition* 13 (1981): 111–122.
- [Ballard82] D. Ballard and C. Brown, *Computer Vision*, Englewood Cliffs, NJ: Prentice-Hall, 1982.
- [Bardyn84] J. J. Bardyn et al., “Une architecture VLSI pour un opérateur de filtrage median,” *Congrès reconnaissance des formes et intelligence artificielle* (vol. 1, pp. 557–566), Paris, 25–27 January 1984.
- [Bay06] H. Bay, T. Tuytelaars, and L. V. Gool, “SURF: Speeded up robust features,” *Proceedings of the Ninth European Conference on Computer Vision* (pp. 404–417), May 2006.
- [Bayes1763] T. Bayes, “An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F.R.S. communicated by Mr. Price, in a letter to John

- Canton, A.M.F.R.S.,” *Philosophical Transactions, Giving Some Account of the Present Undertakings, Studies and Labours of the Ingenious in Many Considerable Parts of the World* 53 (1763): 370–418.
- [Beauchemin95] S. S. Beauchemin and J. L. Barron, “The computation of optical flow,” *ACM Computing Surveys* 27 (1995): 433–466.
- [Belongie00] S. Belongie, J. Malik, and J. Puzicha, “Shape context: A new descriptor for shape matching and object recognition,” NIPS 2000, Computer Vision Group, University of California, Berkeley, 2000.
- [Berg01] A. C. Berg and J. Malik, “Geometric blur for template matching,” *IEEE Conference on Computer Vision and Pattern Recognition* (vol. 1, pp. 607–614), Kauai, Hawaii, 2001.
- [Bhattacharyya43] A. Bhattacharyya, “On a measure of divergence between two statistical populations defined by probability distributions,” *Bulletin of the Calcutta Mathematical Society* 35 (1943): 99–109.
- [Bishop07] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York: Springer-Verlag, 2007.
- [Black92] M. J. Black, “Robust incremental optical flow” (YALEU-DCS-RR-923), Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT, 1992.
- [Black93] M. J. Black and P. Anandan, “A framework for the robust estimation of optical flow,” *Fourth International Conference on Computer Vision* (pp. 231–236), May 1993.
- [Black96] M. J. Black and P. Anandan, “The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields,” *Computer Vision and Image Understanding* 63 (1996): 75–104.
- [Bobick96] A. Bobick and J. Davis, “Real-time recognition of activity using temporal templates,” *IEEE Workshop on Applications of Computer Vision* (pp. 39–42), December 1996.
- [Borgefors86] G. Borgefors, “Distance transformations in digital images,” *Computer Vision, Graphics and Image Processing* 34 (1986): 344–371.
- [Bosch07] A. Bosch, A. Zisserman, and X. Muñoz, “Image classification using random forests and ferns,” *IEEE International Conference on Computer Vision*, Rio de Janeiro, October 2007.
- [Bouguet] J.-Y. Bouguet, “Camera calibration toolbox for Matlab,” retrieved June 2, 2008, from http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.
- [BouguetAlg] J.-Y. Bouguet, “The calibration toolbox for Matlab, example 5: Stereo rectification algorithm” (code and instructions only), http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.html.

- [Bouguet04] J.-Y. Bouguet, "Pyramidal implementation of the Lucas Kanade feature tracker description of the algorithm," http://robots.stanford.edu/cs223b04/algo_tracking.pdf.
- [Bracewell65] R. Bracewell, "Convolution" and "Two-dimensional convolution," in *The Fourier Transform and Its Applications* (pp. 25–50 and 243–244), New York: McGraw-Hill, 1965.
- [Bradski00] G. Bradski and J. Davis, "Motion segmentation and pose recognition with motion history gradients," *IEEE Workshop on Applications of Computer Vision*, 2000.
- [Bradski98a] G. R. Bradski, "Real time face and object tracking as a component of a perceptual user interface," *Proceedings of the 4th IEEE Workshop on Applications of Computer Vision*, October 1998.
- [Bradski98b] G. R. Bradski, "Computer video face tracking for use in a perceptual user interface," *Intel Technology Journal* Q2 (1998): 705–740.
- [Breiman01] L. Breiman, "Random forests," *Machine Learning* 45 (2001): 5–32.
- [Breiman84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Monteray, CA: Wadsworth, 1984.
- [Bresenham65] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal* 4 (1965): 25–30.
- [Bronshtein97] I. N. Bronshtein, and K. A. Semendyayev, *Handbook of Mathematics*, 3rd ed., New York: Springer-Verlag, 1997.
- [Brown66] D. C. Brown, "Decentering distortion of lenses," *Photogrammetric Engineering* 32 (1966): 444–462.
- [Brown71] D. C. Brown, "Close-range camera calibration," *Photogrammetric Engineering* 37 (1971): 855–866.
- [Burt81] P. J. Burt, T. H. Hong, and A. Rosenfeld, "Segmentation and estimation of image region properties through cooperative hierarchical computation," *IEEE Transactions on Systems, Man, and Cybernetics* 11 (1981): 802–809.
- [Burt83] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications* 31 (1983): 532–540.
- [Canny86] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8 (1986): 679–714.
- [Carpenter03] G. A. Carpenter and S. Grossberg, "Adaptive resonance theory," in M. A. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks*, 2nd ed. (pp. 87–90), Cambridge, MA: MIT Press, 2003.
- [Carr04] H. Carr, J. Snoeyink, and M. van de Panne, "Progressive topological simplification using contour trees and local spatial measures," *15th Western Computer Graphics Symposium*, Big White, British Columbia, March 2004.

- [Chen05] D. Chen and G. Zhang, “A new sub-pixel detector for x-corners in camera calibration targets,” *WSCG Short Papers* (2005): 97–100.
- [Chetverikov99] D. Chetverikov and Zs. Szabo, “A simple and efficient algorithm for detection of high curvature points in planar curves,” *Proceedings of the 23rd Workshop of the Austrian Pattern Recognition Group* (pp. 175–184), 1999.
- [Chu07] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, “Map-reduce for machine learning on multicore,” *Proceedings of the Neural Information Processing Systems Conference* (vol. 19, pp. 304–310), 2007.
- [Clarke98] T. A. Clarke and J. G. Fryer, “The Development of Camera Calibration Methods and Models,” *Photogrammetric Record* 16 (1998): 51–66.
- [Colombari07] A. Colombari, A. Fusiello, and V. Murino, “Video objects segmentation by robust background modeling,” *International Conference on Image Analysis and Processing* (pp. 155–164), September 2007.
- [Comaniciu99] D. Comaniciu and P. Meer, “Mean shift analysis and applications,” *IEEE International Conference on Computer Vision* (vol. 2, p. 1197), 1999.
- [Comaniciu03] D. Comaniciu, “Nonparametric information fusion for motion estimation,” *IEEE Conference on Computer Vision and Pattern Recognition* (vol. 1, pp. 59–66), 2003.
- [Conrady1919] A. Conrady, “Decentering lens systems,” *Monthly Notices of the Royal Astronomical Society* 79 (1919): 384–390.
- [Cooley65] J. W. Cooley and O. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation* 19 (1965): 297–301.
- [Dahlkamp06] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski, “Self-supervised monocular road detection in desert terrain,” *Robotics: Science and Systems*, Philadelphia, 2006.
- [Dalai05] N. Dalai, and B. Triggs, “Histograms of oriented gradients for human detection,” *Computer Vision and Pattern Recognition* (vol. 1, pp. 886–893), June 2005.
- [Davis97] J. Davis and A. Bobick, “The representation and recognition of action using temporal templates” (Technical Report 402), MIT Media Lab, Cambridge, MA, 1997.
- [Davis99] J. Davis and G. Bradski, “Real-time motion template gradients using Intel CVLib,” *ICCV Workshop on Framerate Vision*, 1999.
- [Delaunay34] B. Delaunay, “Sur la sphère vide,” *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk* 7 (1934): 793–800.
- [DeMenthon92] D. F. DeMenthon and L. S. Davis, “Model-based object pose in 25 lines of code,” *Proceedings of the European Conference on Computer Vision* (pp. 335–343), 1992.

- [Dempster77] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B* 39 (1977): 1–38.
- [Det] "History of matrices and determinants," http://www-history.mcs.st-and.ac.uk/history/HistTopics/Matrices_and_determinants.html.
- [Douglas73] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required for represent a digitized line or its caricature," *Canadian Cartographer* 10(1973): 112–122.
- [Duda72] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the Association for Computing Machinery* 15 (1972): 11–15.
- [Duda73] R. O. Duda and P. E. Hart, *Pattern Recognition and Scene Analysis*, New York: Wiley, 1973.
- [Duda00] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, New York: Wiley, 2001.
- [Farin04] D. Farin, P. H. N. de With, and W. Effelsberg, "Video-object segmentation using multi-sprite background subtraction," *Proceedings of the IEEE International Conference on Multimedia and Expo*, 2004.
- [Faugeras93] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*, Cambridge, MA: MIT Press, 1993.
- [Fei-Fei98] L. Fei-Fei, R. Fergus, and P. Perona, "A Bayesian approach to unsupervised one-shot learning of object categories," *Proceedings of the Ninth International Conference on Computer Vision* (vol. 2, pp. 1134–1141), October 2003.
- [Felzenszwalb63] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions" (Technical Report TR2004-1963), Department of Computing and Information Science, Cornell University, Ithaca, NY, 1963.
- [FFmpeg] "Ffmpeg summary," <http://en.wikipedia.org/wiki/Ffmpeg>.
- [Fischler81] M. A. Fischler and R. C. Bolles, "Random sample concensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the Association for Computing Machinery* 24 (1981): 381–395.
- [Fitzgibbon95] A. W. Fitzgibbon and R. B. Fisher, "A buyer's guide to conic fitting," *Proceedings of the 5th British Machine Vision Conference* (pp. 513–522), Birmingham, 1995.
- [Fix51] E. Fix, and J. L. Hodges, "Discriminatory analysis, nonparametric discrimination: Consistency properties" (Technical Report 4), USAF School of Aviation Medicine, Randolph Field, Texas, 1951.
- [Forsyth03] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, Englewood Cliffs, NJ: Prentice-Hall, 2003.

- [FourCC] “FourCC summary,” <http://en.wikipedia.org/wiki/Fourcc>.
- [FourCC85] J. Morrison, “EA IFF 85 standard for interchange format files,” <http://www.szonye.com;bradd/iff.html>.
- [Fourier] “Joseph Fourier,” http://en.wikipedia.org/wiki/Joseph_Fourier.
- [Freeman67] H. Freeman, “On the classification of line-drawing data,” *Models for the Perception of Speech and Visual Form* (pp. 408–412), 1967.
- [Freeman95] W. T. Freeman and M. Roth, “Orientation histograms for hand gesture recognition,” *International Workshop on Automatic Face and Gesture Recognition* (pp. 296–301), June 1995.
- [Freund97] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences* 55 (1997): 119–139.
- [Fryer86] J. G. Fryer and D. C. Brown, “Lens distortion for close-range photogrammetry,” *Photogrammetric Engineering and Remote Sensing* 52 (1986): 51–58.
- [Fukunaga90] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Boston: Academic Press, 1990.
- [Galton] “Francis Galton,” http://en.wikipedia.org/wiki/Francis_Galton.
- [GEMM] “Generalized matrix multiplication summary,” <http://notvincenz.blogspot.com/2007/06/generalized-matrix-multiplication.html>.
- [Göktürk01] S. B. Göktürk, J.-Y. Bouguet, and R. Grzeszczuk, “A data-driven model for monocular face tracking,” *Proceedings of the IEEE International Conference on Computer Vision* (vol. 2, pp. 701–708), 2001.
- [Göktürk02] S. B. Göktürk, J.-Y. Bouguet, C. Tomasi, and B. Girod, “Model-based face tracking for view-independent facial expression recognition,” *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition* (pp. 287–293), May 2002.
- [Grauman05] K. Grauman and T. Darrell, “The pyramid match kernel: Discriminative classification with sets of image features,” *Proceedings of the IEEE International Conference on Computer Vision*, October 2005.
- [Grossberg87] S. Grossberg, “Competitive learning: From interactive activation to adaptive resonance,” *Cognitive Science* 11 (1987): 23–63.
- [Harris88] C. Harris and M. Stephens, “A combined corner and edge detector,” *Proceedings of the 4th Alvey Vision Conference* (pp. 147–151), 1988.
- [Hartley98] R. I. Hartley, “Theory and practice of projective rectification,” *International Journal of Computer Vision* 35 (1998): 115–127.
- [Hartley06] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge, UK: Cambridge University Press, 2006.

- [Hastie01] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, New York: Springer-Verlag, 2001.
- [Heckbert90] P. Heckbert, *A Seed Fill Algorithm* (Graphics Gems I), New York: Academic Press, 1990.
- [Heikkila97] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition* (p. 1106), 1997.
- [Hinterstoesser08] S. Hinterstoesser, S. Benhimane, V. Lepetit, P. Fua, and N. Navab, "Simultaneous recognition and homography extraction of local patches with a simple linear classifier," *British Machine Vision Conference*, Leeds, September 2008.
- [Hinton06] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation* 18 (2006): 1527–1554.
- [Ho95] T. K. Ho, "Random decision forest," *Proceedings of the 3rd International Conference on Document Analysis and Recognition* (pp. 278–282), August 1995.
- [Homma85] K. Homma and E.-I. Takenaka, "An image processing method for feature extraction of space-occupying lesions," *Journal of Nuclear Medicine* 26 (1985): 1472–1477.
- [Horn81] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence* 17 (1981): 185–203.
- [Hough59] P. V. C. Hough, "Machine analysis of bubble chamber pictures," *Proceedings of the International Conference on High Energy Accelerators and Instrumentation* (pp. 554–556), 1959.
- [Hu62] M. Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory* 8 (1962): 179–187.
- [Huang95] Y. Huang and X. H. Zhuang, "Motion-partitioned adaptive block matching for video compression," *International Conference on Image Processing* (vol. 1, p. 554), 1995.
- [Iivarinen97] J. Iivarinen, M. Peura, J. Särelä, and A. Visa, "Comparison of combined shape descriptors for irregular objects," *8th British Machine Vision Conference*, 1997.
- [Intel] Intel Corporation, <http://www.intel.com/>.
- [Inui03] K. Inui, S. Kaneko, and S. Igarashi, "Robust line fitting using LmedS clustering," *Systems and Computers in Japan* 34 (2003): 92–100.
- [IPL] Intel Image Processing Library (IPL), www.cc.gatech.edu/dvfx/readings/iplman.pdf.
- [IPP] Intel Integrated Performance Primitives, <http://www.intel.com/cd/software/products/asmo-na/eng/219767.htm>.

- [Isard98] M. Isard and A. Blake, “CONDENSATION: Conditional density propagation for visual tracking,” *International Journal of Computer Vision* 29 (1998): 5–28.
- [Jaehne95] B. Jaehne, *Digital Image Processing*, 3rd ed., Berlin: Springer-Verlag, 1995.
- [Jaehne97] B. Jaehne, *Practical Handbook on Image Processing for Scientific Applications*, Boca Raton, FL: CRC Press, 1997.
- [Jain77] A. Jain, “A fast Karhunen-Loeve transform for digital restoration of images degraded by white and colored noise,” *IEEE Transactions on Computers* 26 (1997): 560–571.
- [Jain86] A. Jain, *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1986.
- [Johnson84] D. H. Johnson, “Gauss and the history of the fast Fourier transform,” *IEEE Acoustics, Speech, and Signal Processing Magazine* 1 (1984): 14–21.
- [Kalman60] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering* 82 (1960): 35–45.
- [Kim05] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis, “Real-time foreground-background segmentation using codebook model,” *Real-Time Imaging* 11 (2005): 167–256.
- [Kimme75] C. Kimme, D. H. Ballard, and J. Sklansky, “Finding circles by an array of accumulators,” *Communications of the Association for Computing Machinery* 18 (1975): 120–122.
- [Kiryati91] N. Kiryati, Y. Eldar, and A. M. Bruckstein, “A probabilistic Hough transform,” *Pattern Recognition* 24 (1991): 303–316.
- [Konolige97] K. Konolige, “Small vision system: Hardware and implementation,” *Proceedings of the International Symposium on Robotics Research* (pp. 111–116), Hayama, Japan, 1997.
- [Kreveld97] M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. R. Schikore, “Contour trees and small seed sets for isosurface traversal,” *Proceedings of the 13th ACM Symposium on Computational Geometry* (pp. 212–220), 1997.
- [Lagrange1773] J. L. Lagrange, “Solutions analytiques de quelques problèmes sur les pyramides triangulaires,” in *Oeuvres* (vol. 3), 1773.
- [Laughlin81] S. B. Laughlin, “A simple coding procedure enhances a neuron’s information capacity,” *Zeitschrift für Naturforschung* 9/10 (1981): 910–912.
- [LeCun98a] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE* 86 (1998): 2278–2324.
- [LeCun98b] Y. LeCun, L. Bottou, G. Orr, and K. Muller, “Efficient BackProp,” in G. Orr and K. Muller (Eds.), *Neural Networks: Tricks of the Trade*, New York: Springer-Verlag, 1998.
- [Lens] “Lens (optics),” [http://en.wikipedia.org/wiki/Lens_\(optics\)](http://en.wikipedia.org/wiki/Lens_(optics)).

- [Liu07] Y. Z. Liu, H. X. Yao, W. Gao, X. L. Chen, and D. Zhao, “Nonparametric background generation,” *Journal of Visual Communication and Image Representation* 18 (2007): 253–263.
- [Lloyd57] S. Lloyd, “Least square quantization in PCM’s” (Bell Telephone Laboratories Paper), 1957. [“Lloyd’s algorithm” was later published in *IEEE Transactions on Information Theory* 28 (1982): 129–137.]
- [Lowe04] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision* 60 (2004): 91–110.
- [LTI] LTI-Lib, Vision Library, <http://ltilib.sourceforge.net/doc/homepage/index.shtml>.
- [Lucas81] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” *Proceedings of the 1981 DARPA Imaging Understanding Workshop* (pp. 121–130), 1981.
- [Lucchese02] L. Lucchese and S. K. Mitra, “Using saddle points for subpixel feature detection in camera calibration targets,” *Proceedings of the 2002 Asia Pacific Conference on Circuits and Systems* (pp. 191–195), December 2002.
- [Mahal] “Mahalanobis summary,” http://en.wikipedia.org/wiki/Mahalanobis_distance.
- [Mahalanobis36] P. Mahalanobis, “On the generalized distance in statistics,” *Proceedings of the National Institute of Science* 12 (1936): 49–55.
- [Manta] Manta Open Source Interactive Ray Tracer, http://code.sci.utah.edu/Manta/index.php/Main_Page.
- [Maron61] M. E. Maron, “Automatic indexing: An experimental inquiry,” *Journal of the Association for Computing Machinery* 8 (1961): 404–417.
- [Marr82] D. Marr, *Vision*, San Francisco: Freeman, 1982.
- [Martins99] F. C. M. Martins, B. R. Nickerson, V. Bostrom, and R. Hazra, “Implementation of a real-time foreground/background segmentation system on the Intel architecture,” *IEEE International Conference on Computer Vision Frame Rate Workshop*, 1999.
- [Matas00] J. Matas, C. Galambos, and J. Kittler, “Robust detection of lines using the progressive probabilistic Hough transform,” *Computer Vision Image Understanding* 78 (2000): 119–137.
- [Matas02] J. Matas, O. Chum, M. Urba, and T. Pajdla, “Robust wide baseline stereo from maximally stable extremal regions,” *Proceedings of the British Machine Vision Conference* (pp. 384–396), 2002.
- [Meer91] P. Meer, D. Mintz, and A. Rosenfeld, “Robust regression methods for computer vision: A review,” *International Journal of Computer Vision* 6 (1991): 59–70.
- [Merwe00] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan, “The unscented particle filter,” *Advances in Neural Information Processing Systems*, December 2000.

- [Meyer78] F. Meyer, “Contrast feature extraction,” in J.-L. Chermant (Ed.), *Quantitative Analysis of Microstructures in Material Sciences, Biology and Medicine* [Special issue of Practical Metallography], Stuttgart: Riederer, 1978.
- [Meyer92] F. Meyer, “Color image segmentation,” *Proceedings of the International Conference on Image Processing and Its Applications* (pp. 303–306), 1992.
- [Mikolajczyk04] K. Mikolajczyk and C. Schmid, “A performance evaluation of local descriptors,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (2004): 1615–1630.
- [Minsky61] M. Minsky, “Steps toward artificial intelligence,” *Proceedings of the Institute of Radio Engineers* 49 (1961): 8–30.
- [Mokhtarian86] F. Mokhtarian and A. K. Mackworth, “Scale based description and recognition of planar curves and two-dimensional shapes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8 (1986): 34–43.
- [Mokhtarian88] F. Mokhtarian, “Multi-scale description of space curves and three-dimensional objects,” *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 298–303), 1988.
- [Mori05] G. Mori, S. Belongie, and J. Malik, “Efficient shape matching using shape contexts,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (2005): 1832–1837.
- [Morse53] P. M. Morse and H. Feshbach, “Fourier transforms,” in *Methods of Theoretical Physics* (Part I, pp. 453–471), New York: McGraw-Hill, 1953.
- [Neveu86] C. F. Neveu, C. R. Dyer, and R. T. Chin, “Two-dimensional object recognition using multiresolution models,” *Computer Vision Graphics and Image Processing* 34 (1986): 52–65.
- [Ng] A. Ng, “Advice for applying machine learning,” <http://www.stanford.edu/class/cs229/materials/ML-advice.pdf>.
- [Nistér06] D. Nistér and H. Stewénius, “Scalable recognition with a vocabulary tree,” *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.
- [O’Connor02] J. J. O’Connor and E. F. Robertson, “Light through the ages: Ancient Greece to Maxwell,” http://www-groups.dcs.st-and.ac.uk/~history/HistTopics/Light_1.html.
- [Oliva06] A. Oliva and A. Torralba, “Building the gist of a scene: The role of global image features in recognition visual perception,” *Progress in Brain Research* 155 (2006): 23–36.
- [Opelt08] A. Opelt, A. Pinz, and A. Zisserman, “Learning an alphabet of shape and appearance for multi-class object detection,” *International Journal of Computer Vision* (2008).
- [OpenCV] Open Source Computer Vision Library (OpenCV), <http://sourceforge.net/projects/opencvlibrary/>.

- [OpenCV Wiki] Open Source Computer Vision Library Wiki, <http://opencvlibrary.sourceforge.net/>.
- [OpenCV YahooGroups] OpenCV discussion group on Yahoo, <http://groups.yahoo.com/group/OpenCV>.
- [Ozuysal07] M. Ozuysal, P. Fua, and V. Lepetit, “Fast keypoint recognition in ten lines of code,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.
- [Papoulis62] A. Papoulis, *The Fourier Integral and Its Applications*, New York: McGraw-Hill, 1962.
- [Pascucci02] V. Pascucci and K. Cole-McLaughlin, “Efficient computation of the topology of level sets,” *Proceedings of IEEE Visualization 2002* (pp. 187–194), 2002.
- [Pearson] “Karl Pearson,” http://en.wikipedia.org/wiki/Karl_Pearson.
- [Philbin07] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Object retrieval with large vocabularies and fast spatial matching,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.
- [Pollefeys99a] M. Pollefeys, “Self-calibration and metric 3D reconstruction from uncalibrated image sequences,” Ph.D. thesis, Katholieke Universiteit, Leuven, 1999.
- [Pollefeys99b] M. Pollefeys, R. Koch, and L. V. Gool, “A simple and efficient rectification method for general motion,” *Proceedings of the 7th IEEE Conference on Computer Vision*, 1999.
- [Porter84] T. Porter and T. Duff, “Compositing digital images,” *Computer Graphics* 18 (1984): 253–259.
- [Prados05] E. Prados and O. Faugeras, “Shape from shading: A well-posed problem?” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005.
- [Ranger07] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating mapreduce for multi-core and multiprocessor systems,” *Proceedings of the 13th International Symposium on High-Performance Computer Architecture* (pp. 13–24), 2007.
- [Reeb46] G. Reeb, “Sur les points singuliers d’une forme de Pfaff complètement intégrable ou d’une fonction numérique,” *Comptes Rendus de l’Academie des Sciences de Paris* 222 (1946): 847–849.
- [Rodgers88] J. L. Rodgers and W. A. Nicewander, “Thirteen ways to look at the correlation coefficient,” *American Statistician* 42 (1988): 59–66.
- [Rodrigues] “Olinde Rodrigues,” http://en.wikipedia.org/wiki/Benjamin_Olinde_Rodrigues.
- [Rosenfeld73] A. Rosenfeld and E. Johnston, “Angle detection on digital curves,” *IEEE Transactions on Computers* 22 (1973): 875–878.

- [Rosenfeld80] A. Rosenfeld, "Some Uses of Pyramids in Image Processing and Segmentation," *Proceedings of the DARPA Imaging Understanding Workshop* (pp. 112–120), 1980.
- [Rousseeuw84] P. J. Rousseeuw, "Least median of squares regression," *Journal of the American Statistical Association*, 79 (1984): 871–880.
- [Rousseeuw87] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, New York: Wiley, 1987.
- [Rubner98a] Y. Rubner, C. Tomasi, and L. J. Guibas, "Metrics for distributions with applications to image databases," *Proceedings of the 1998 IEEE International Conference on Computer Vision* (pp. 59–66), Bombay, January 1998.
- [Rubner98b] Y. Rubner and C. Tomasi, "Texture metrics," *Proceeding of the IEEE International Conference on Systems, Man, and Cybernetics* (pp. 4601–4607), San Diego, October 1998.
- [Rubner00] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision* 40 (2000): 99–121.
- [Rumelhart88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in D. E. Rumelhart, J. L. McClelland, and PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition* (vol. 1, pp. 318–362), Cambridge, MA: MIT Press, 1988.
- [Russ02] J. C. Russ, *The Image Processing Handbook*, 4th ed. Boca Raton, FL: CRC Press, 2002.
- [Scharr00] H. Scharr, "Optimal operators in digital image processing," Ph.D. thesis, Interdisciplinary Center for Scientific Computing, Ruprecht-Karls-Universität, Heidelberg, <http://www.fz-juelich.de/icg/icg-3/index.php?index=195>.
- [Schiele96] B. Schiele and J. L. Crowley, "Object recognition using multidimensional receptive field histograms," *European Conference on Computer Vision* (vol. 1, pp. 610–619), April 1996.
- [Schmidt66] S. Schmidt, "Applications of state-space methods to navigation problems," in C. Leondes (Ed.), *Advances in Control Systems* (vol. 3, pp. 293–340), New York: Academic Press, 1966.
- [Schroff08] F. Schroff, A. Criminisi, and A. Zisserman, "Object class segmentation using random forests," *Proceedings of the British Machine Vision Conference*, 2008.
- [Schwartz80] E. L. Schwartz, "Computational anatomy and functional architecture of the striate cortex: A spatial mapping approach to perceptual coding," *Vision Research* 20 (1980): 645–669.
- [Schwarz78] A. A. Schwarz and J. M. Soha, "Multidimensional histogram normalization contrast enhancement," *Proceedings of the Canadian Symposium on Remote Sensing* (pp. 86–93), 1978.

- [Semple79] J. Semple and G. Kneebone, *Algebraic Projective Geometry*, Oxford, UK: Oxford University Press, 1979.
- [Serra83] J. Serra, *Image Analysis and Mathematical Morphology*, New York: Academic Press, 1983.
- [Sezgin04] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging* 13 (2004): 146–165.
- [Shapiro02] L. G. Shapiro and G. C. Stockman, *Computer Vision*, Englewood Cliffs, NJ: Prentice-Hall, 2002.
- [Sharon06] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt, "Hierarchy and adaptivity in segmenting visual scenes," *Nature* 442 (2006): 810–813.
- [Shaw04] J. R. Shaw, "QuickFill: An efficient flood fill algorithm," <http://www.codeproject.com/gdi/QuickFill.asp>.
- [Shi00] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000): 888–905.
- [Shi94] J. Shi and C. Tomasi, "Good features to track," *9th IEEE Conference on Computer Vision and Pattern Recognition*, June 1994.
- [Sivic08] J. Sivic, B. C. Russell, A. Zisserman, W. T. Freeman, and A. A. Efros, "Unsupervised discovery of visual object class hierarchies," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
- [Smith78] A. R. Smith. "Color gamut transform pairs," *Computer Graphics* 12 (1978): 12–19.
- [Smith79] A. R. Smith, "Painting tutorial notes," Computer Graphics Laboratory, New York Institute of Technology, Islip, NY, 1979.
- [Sobel73] I. Sobel and G. Feldman, "A 3×3 Isotropic Gradient Operator for Image Processing," in R. Duda and P. Hart (Eds.), *Pattern Classification and Scene Analysis* (pp. 271–272), New York: Wiley, 1973.
- [Steinhaus56] H. Steinhaus, "Sur la division des corp matériels en parties," *Bulletin of the Polish Academy of Sciences and Mathematics* 4 (1956): 801–804.
- [Sturm99] P. F. Sturm and S. J. Maybank, "On plane-based camera calibration: A general algorithm, singularities, applications," *IEEE Conference on Computer Vision and Pattern Recognition*, 1999.
- [Sun98] J. Sun and P. Perona, "Where is the sun?" *Nature Neuroscience* 1 (1998): 183–184.
- [Suzuki85] S. Suzuki and K. Abe, "Topological structural analysis of digital binary images by border following," *Computer Vision, Graphics and Image Processing* 30 (1985): 32–46.
- [SVD] "SVD summary," http://en.wikipedia.org/wiki/Singular_value_decomposition.

- [Swain91] M. J. Swain and D. H. Ballard, “Color indexing,” *International Journal of Computer Vision* 7 (1991): 11–32.
- [Tanguay00] D. Tanguay, “Flying a Toy Plane,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (p. 2231), 2000.
- [Teh89] C. H. Teh, R. T. Chin, “On the detection of dominant points on digital curves,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11 (1989): 859–872.
- [Telea04] A. Telea, “An image inpainting technique based on the fast marching method,” *Journal of Graphics Tools* 9 (2004): 25–36.
- [Thrun05] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics: Intelligent Robotics and Autonomous Agents*, Cambridge, MA: MIT Press, 2005.
- [Thrun06] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. “Stanley, the robot that won the DARPA Grand Challenge,” *Journal of Robotic Systems* 23 (2006): 661–692.
- [Titchmarsh26] E. C. Titchmarsh, “The zeros of certain integral functions,” *Proceedings of the London Mathematical Society* 25 (1926): 283–302.
- [Tola08] E. Tola, V. Lepetit, and P. Fua, “A fast local descriptor for dense matching,” *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, June 2008.
- [Tomasi98] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” *Sixth International Conference on Computer Vision* (pp. 839–846), New Delhi, 1998.
- [Torralba07] A. Torralba, K. P. Murphy, and W. T. Freeman, “Sharing visual features for multiclass and multiview object detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (2007): 854–869.
- [Torralba08] A. Torralba, R. Fergus, and Y. Weiss, “Small codes and large databases for recognition,” *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, June 2008.
- [Toyama99] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, “Wallflower: Principles and practice of background maintenance,” *Proceedings of the 7th IEEE International Conference on Computer Vision* (pp. 255–261), 1999.
- [Trace] “Matrix trace summary,” [http://en.wikipedia.org/wiki/Trace_\(linear_algebra\)](http://en.wikipedia.org/wiki/Trace_(linear_algebra)).
- [Trucco98] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision*, Englewood Cliffs, NJ: Prentice-Hall, 1998.
- [Tsai87] R. Y. Tsai, “A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses,” *IEEE Journal of Robotics and Automation* 3 (1987): 323–344.

- [Vandevenne04] L. Vandevenne, "Lode's computer graphics tutorial, flood fill," <http://student.kuleuven.be/~m0216922/CG/floodfill.html>.
- [Vapnik95] V. Vapnik, *The Nature of Statistical Learning Theory*, New York: Springer-Verlag, 1995.
- [Videre] Videre Design, "Stereo on a chip (STOC)," <http://www.videredesign.com/templates/stoc.htm>.
- [Viola04] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision* 57 (2004): 137–154.
- [VXL] VXL, Vision Library, <http://vxl.sourceforge.net/>.
- [Welsh95] G. Welsh and G. Bishop, "An introduction to the Kalman filter" (Technical Report TR95-041), University of North Carolina, Chapel Hill, NC, 1995.
- [Werbos74] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioural sciences," Ph.D. thesis, Economics Department, Harvard University, Cambridge, MA, 1974.
- [WG] Willow Garage, <http://www.willowgarage.com>.
- [Wharton71] W. Wharton and D. Howorth, *Principles of Television Reception*, London: Pitman, 1971.
- [Xu96] G. Xu and Z. Zhang, *Epipolar Geometry in Stereo, Motion and Object Recognition*, Dordrecht: Kluwer, 1996.
- [Zhang96] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting," *Image and Vision Computing* 15 (1996): 59–76.
- [Zhang99] R. Zhang, P.-S. Tsi, J. E. Cryer, and M. Shah, "Shape form shading: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21 (1999): 690–706.
- [Zhang99] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," *Proceedings of the 7th International Conference on Computer Vision* (pp. 666–673), Corfu, September 1999.
- [Zhang00] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000): 1330–1334.
- [Zhang04] H. Zhang, "The optimality of naive Bayes," *Proceedings of the 17th International FLAIRS Conference*, 2004.

索引

A

absolute value, 48, 49–50, 57
accumulation functions, 276, 278
accumulator plane, 154, 156
AdaBoost, 496–498, 506–508
affine transforms, 163–169, 173, 407
allocation of memory, 222, 472
alpha blending, 50–52
AMD processors, 15
anchor points, 115, 144–145
aperture problem, 327, 328
arrays, 54
 accessing members of, 36, 37
 merging, 67–68
 norm, total, computing, 69
 operators, table of, 48–49
 of points, 40–41
 row/column index, reversing, 76
 sequences and, 233
 setting elements of, 72–73, 77
 splitting, 73–74
 square, 60
artistic community, OpenCV needs of, 525
averaging background method, 271–278

B

background
 defined, 266
 learning, 275, 282
 statistical model, 273
 subtraction (differencing), 265–267,
 270, 278
 versus foreground, 267
background-foreground segmentation, 14
back projection, 209–213, 386
back-propagation (MLP), 498, 517
barrel (fish-eye) effect, 375, 376

Bayer pattern, 59
Bayes classifier, naïve (normal), 462, 474,
 483–486
Bayesian network, 461–462, 483, 484
Bayes' theorem, 210
Bhattacharyya matching, 202, 206
bias (underfitting)
 intentional, 493–495, 496
 overview of, 466–468
bilateral filter, 110–115
bird's-eye view transform, 408–412
bitwise AND operation, 52
bitwise OR operation, 71
bitwise XOR operation, 77
Black Hat operation, 120, 123–124, 127
block matching method, 322, 336, 439, 443–444
blurring (see smoothing)
Boolean images, 120, 121, 153
Boolean mask, 135
boosted rejection cascade, 506
boosting classifiers, 463, 495–501, 506, 508
bootstrapping, 469
Borgefors (Gunilla) method, 186
Bouquet, Jean-Yves, website, 378
Bouquet algorithm, 433–436, 445
boundaries
 box, 279
 convolution, 146
 exterior, 234
 interior, 234
Breiman binary decision trees, 486
Breiman random forests theory, 501
Breiman variable importance algorithm,
 465, 495
Bresenham algorithm, 77
brightness constancy, 324, 325, 326, 335
Brown method, 376, 389
buttons (simulating), 101

C

- calibration, 14, 320, 370, 378, 397–401
- callback, defined, 95–96
- cameras
 - artifact reduction, 109
 - CVCAM interface, 12
 - domains, 103
 - focal length, 371, 373
 - format reversed, 107
 - identifiers, 103
 - input from, 16, 19, 26, 102–105
 - intrinsics matrix, defined, 373, 392, 454
 - manufacturing defects, 375–377, 467
 - path, reconstructing, 320
 - pinhole model, 370, 371–373, 391
 - projection matrix, 385
 - properties, checking and setting, 104
 - stereo imaging, overview of, 415
 - whiteouts, avoiding, 186
 - (see also calibration)
- canshift tracking, 337, 341
- Canny edge detector, 25, 151–160, 187, 234
- Canny, J., 151
- Cartesian to polar coordinates, 172–174
- CCH (chain code histogram), 262
- cell-phone projectors, 525
- center of projection, 371, 407
- chain code histogram (CCH), 262
- channel of interest (COI), 44, 45
- channel, defined, 41
- chessboards (calibration object)
 - corners, drawing, 383
 - corners, finding, 382–384, 388, 392
 - overview of, 381, 428
 - stereo rectification, 439
- chi-square method, histograms, 202
- Chinese wiki site, 12
- circles, 78–79, 249
- circle transform (Hough), 158–161
- circum-circle property, 300
- classification and regression tree (CART)
 - algorithms, 486, 492, 495
- classification, machine learning, 459–461
- classifiers
 - Bayes, 483–486
 - Haar, 506–509
 - strong, 496, 499
 - Viola-Jones, 506–511, 515
 - weak, 463, 496–501, 507, 516
- clone functions, defined, 34
- clustering algorithms, 459–461, 479
- codebook method, 266, 278–287
- codecs, 28, 92, 102, 105, 106
- COI (channel of interest), 44, 45
- color conversions, 48, 58–60, 106
- color histograms, 205, 206
- color similarity, 298
- color space, 58–60, 278
- compilers, 14
- compression codecs, 102, 105, 106
- computer vision (see vision, computer; vision, human)
- Concurrent Versions System (CVS), 10–11
- condensation algorithm, 349–350, 364–367
- conditional random field (CRF), 525
- configuration and log files, reading and writing, 83
- confusion matrices, 469–471, 493, 495
- connected components
 - closing and, 121
 - defined, 117, 126, 135
 - foreground cleanup and, 287–293, 294
- constructor methods, defined, 31
- container class templates (see sequences)
- containment, 235
- contour
 - area, computing, 248
 - bounding, 248, 249
 - Canny and, 152
 - convexity, 258–260
 - drawing, 241, 243
 - finding, 234–238, 243
 - foreground cleanup and, 287
 - length, computing, 247
 - matching, 251–258
 - moments, 247, 252–256
 - tree, 235–237, 256, 257
- control motion, 354
- convex hull, defined, 259
- convexity defects, 258–260
- convolutions, 144–147
- convolution theorem, 180–182
- correlation methods, 201–202, 215–219
- correspondence
 - calibration and, 445–452
 - defined, 415, 416
 - stereo, 438–445
- covariance matrix, computing, 54
- CRF (conditional random field), 525
- cross-validation, 469
- cumulative distribution function, 188, 189
- CV, component of OpenCV, 13

CVaux, component of OpenCV, 13–14
Cvcore, 29
CVS (Concurrent Versions System), 10–11
CvXX OpenCV classes
 CvANN_MLP, 517
 CvBoost, 498–501
 CvDTree, 488, 493, 498
 CvKNearest, 516
 CvStatModel, 472, 472–475
 CvSVM, 517
CvXX OpenCV data structures
 CvArr, 33
 CvBox2D, 249
 CvConDensation, 364
 CvConnectedComponent, 135
 CvConvexityDefect, 260
 CvFileStorage, 83
 CvHistogram, 195
 CvKalman, 358
 CvMat, 33–41, 44, 83
 CvMoments, 252
 CvPointXXX, 31, 32, 41, 77
 CvRect, 31, 32
 CvScalar, 31, 32, 77, 78
 CvSeq, 224
 CvSize, 23, 31, 32
 CvStereoBMState, 443–444
 CvTermCriteria, 475
 CvTrackbarCallback, 100
 CvVideoWriter, 105–106
cvXX OpenCV functions
 cv2DRotationMatrix(), 168, 407
 cvAbs(), cvAbsDiff(), cvAbsDiffS(), 49–50,
 270–273
 cvAcc(), 138, 271, 276
 cvAdaptiveThreshold(), 138–141, 234
 cvADD(), 138
 cvAdd(), cvAddS(), cvAddWeighted(),
 50–52
 cvAddWeighted(), 138
 cvAnd(), cvAndS(), 52
 cvApproxChains(), 240
 cvApproxPoly(), 245, 246, 258
 cvArcLength(), 247
 cvAvg(), 53
 cvBoundingRect(), 248
 cvBoxPoints(), 221
 cvCalcBackProject(), cvCalcBack
 ProjectPatch(), 209–215
 cvCalcCovarMatrix(), 54, 61, 66, 476, 478
 cvCalcEMD2(), 207
 cvCalcGlobalOrientation(), 344, 347
 cvCalcHist(), 200–201, 205, 209, 212
 cvCalcMotionGradient(), 343–344, 346–347
 cvCalcOpticalFlowBM(), 337
 cvCalcOpticalFlowHS(), 335–336
 cvCalcOpticalFlowLK(), 329
 cvCalcOpticalFlowPyrLK(), 329–332, 454
 cvCalcPGH(), 262
 cvCalcSubdivVoronoi2D(), 304, 309
 cvCalibrateCamera2(), 371, 378, 387,
 392–397, 403, 406, 427–430
 cvCamShift(), 341
 cvCanny(), 25, 152–154, 158–160, 234
 cvCaptureFromCamera(), 19
 cvCartToPolar(), 172–174
 cvCheckContourConvexity(), 260
 cvCircle(), 78–79
 cvClearMemoryStorage, cvClearMem
 Storage(), 223, 226, 512
 cvClearSeq(), 226
 cvClearSubdivVoronoi2D(), 304
 cvCloneImage(), 200
 cvCloneMat(), 34
 cvCloneSeq(), 227
 cvCmp(), cvCmpS(), 55–56
 cvCompareHist(), 201, 213
 cvComputeCorrespondEpilines(),
 426–427, 445
 cvConDensInitSampleSet(), 365
 cvCondensUpdateByTime(), 366
 cvContourArea(), 248
 cvContourPerimeter(), 247
 cvContoursMoments(), 252–253
 cvConvert(), 56
 cvConvertImage(), 106
 cvConvertPointsHomogenous(), 374
 cvConvertScale(), 56, 69, 274
 cvConvertScaleAbs(), 57
 cvConvexHull2(), 259
 cvConvexityDefects(), 260
 cvCopy(), 57
 cvCopyHist(), 200
 cvCopyMakeBorder(), 146
 cvCreateBMState(), 445
 cvCreateCameraCapture(), 26, 102
 cvCreateConDensation(), 365
 cvCreateData(), 34
 cvCreateFileCapture(), 19, 23, 26, 102
 cvCreateHist(), 195
 cvCreateImage(), 24, 81
 cvCreateKalman(), 358

cvXX OpenCV functions (*continued*)

- cvCreateMat(), 34
- cvCreateMatHeader(), 34
- cvCreateMemoryStorage(), cvCreateMemStorage(), 223, 236, 243
- cvCreatePOSITObject(), 413
- cvCreateSeq(), 224, 232–234
- cvCreateStereoBMState(), 444
- cvCreateStructuringElementEx(), 118
- cvCreateTrackbar(), 20, 22, 100
- cvCreateVideoWriter(), 27, 106
- cvCrossProduct(), 57
- cvCvtColor(), 58–60, 512
- cvCvtScale(), 273, 275
- cvCvtSeqToArray(), 233
- cvDCT(), 182
- cvDestroyAllWindows(), 94
- cvDestroyWindow(), 18, 91
- cvDet(), 60, 61
- cvDFT(), 173, 177–178, 180–182
- cvDilate(), 116, 117
- cvDistTransform(), cvDistanceTransform(), 185
- cvDiv(), 60
- cvDotProduct(), 60–61
- cvDrawChessboardCorners(), 383, 384
- cvDrawContours(), 241, 253
- cvDTreeParams(), 488
- cvEigenVV(), 61
- cvEllipse(), 78–79
- cvEndFindContour(), 239
- cvEndWriteSeq(), 231
- cvEndWriteStruct(), 84
- cvEqualizeHist(), 190, 512
- cvErode(), 116, 117, 270
- cvFillPoly(), cvFillConvexPoly(), 79–80
- cvFilter2D(), 145, 173
- cvFindChessboardCorners(), 381–384, 393
- cvFindContours(), 152, 222–226, 234–243, 256
- cvFindCornerSubPix(), 321, 383
- cvFindDominantPoints(), 246
- cvFindExtrinsicCameraParameters2(), cvFindExtrinsicCameraParams2(), 395, 403
- cvFindFundamentalMat(), 424–426, 431
- cvFindHomography(), 387
- cvFindNearestPoint2D(), 311
- cvFindNextContour(), 239
- cvFindStereoCorrespondenceBM(), 439, 443, 444, 454
- cvFitEllipse2(), 250
- cvFitLine(), 455–457
- cvFloodFill(), 124–129, 135
- cvFlushSeqWriter(), 232
- cvGEMM(), 62, 69
- cvGet*D() family, 37–38
- cvGetAffineTransform(), 166–167, 407
- cvGetCaptureProperty(), 21, 28, 104
- cvGetCentralMoment(), 253
- cvGetCol(), cvGetCols(), 62–63, 65
- cvGetDiag(), 63
- cvGetDims(), cvGetDimSize(), 36, 63–64
- cvGetElemType(), 36
- cvGetFileNodeByName(), 85
- cvGetHistValue_XX(), 198
- cvGetHuMoments(), 253
- cvGetMinMaxHistValue(), 200, 205
- cvGetModuleInfo(), 87
- cvGetNormalizedCentralMoment(), 253
- cvGetOptimalDFTSize(), 179
- cvGetPerspectiveTransform(), 170, 407, 409
- cvGetQuadrangleSubPix(), 166, 407
- cvGetRow(), cvGetRows(), 64, 65
- cvGetSeqElem(), 134, 226
- cvGetSeqReaderPos(), 233
- cvGetSize(), 23, 34, 64
- cvGetSpatialMoment(), 253
- cvGetSubRect(), 65, 179
- cvGetTrackbarPos(), 100
- cvGetWindowHandle(), 91
- cvGetWindowName(), 91
- cvGoodFeaturesToTrack(), 318–321, 329, 332
- cvGrabFrame(), 103
- cvHaarDetectObjects(), 507, 513
- cvHistogram(), 199
- cvHoughCircles(), 159–161
- cvHoughLines2(), 156–160
- cvInitFont(), 81
- cvInitLineIterator(), 268
- cvInitMatHeader(), 35
- cvInitSubdivDelaunay2D(), 304, 310
- cvInitUndistortMap(), 396
- cvInitUndistortRectifyMap(), 436–437
- cvInpaint(), 297
- cvInRange(), cvInRangeS(), 65, 271, 275
- cvIntegral(), 182–183
- cvInvert(), 65–66, 73, 75–76, 478
- cvKalmanCorrect(), 359
- cvKalmanPredict(), 359
- cvKMeans2(), 481–483
- cvLaplace(), 151
- cvLine(), 77

cvLoad(), 83, 85, 512
cvLoadImage(), 17, 19, 92
cvLogPolar(), 172, 175
cvMahalonobis(), 66, 478
cvMakeHistHeaderForArray(), 197
cvMakeSeqHeaderForArray(), 234
cvMat(), 35
cvMatchShapes(), 255, 256
cvMatchTemplate(), 214, 215, 218
cvMatMul(), cvMatMulAdd(), 62
cvMax(), cvMaxS(), 66–67
cvMaxRect(), 251
cvMean(), 53
cvMean_StdDev(), 53–54
cvMeanShift(), 298, 340, 479
cvMemStorageAlloc(), 223
cvMerge(), 67–68, 178
cvmGet(), 39
cvMin(), cvMinS(), 68
cvMinAreaRect2(), 248
cvMinEnclosingCircle(), 249
cvMinMaxLoc(), 68, 213, 217
cvMoments(), 253
cvMorphologyEx(), 120
cvMouseCallback(), 96
cvMoveWindow(), 94
cvmSet(), 39, 209
cvMul(), 68–69
cvMulSpectrums(), 179, 180
cvMultiplyAcc(), 277
cvNamedWindow(), 17, 22, 91
cvNorm(), 69–70
cvNormalBayesClassifier(), 485–486
cvNormalize(), 70–71, 218
cvNormalizeHist(), 199, 213
cvNot(), 69
cvOpenFileStorage(), 83, 491
cvOr(), cvOrS(), 71, 270
cvPerspectiveTransform(), 171, 407, 453
cvPoint(), 32, 146
cvPoint2D32f(), cvPointTo32f(), 304
cvPointPolygonTest(), 251
cvPointSeqFromMat(), 251
cvPolarToCart(), 172
cvPolyLine(), 80
cvPOSIT(), 413
cvPow(), 218
cvProjectPoints2(), 406
cvPtr*D() family, 37–38
cvPutText(), 80, 81
cvPyrDown(), 24, 131, 299
cvPyrMeanShiftFiltering(), 298, 299, 300
cvPyrSegmentation(), 132–135, 298
cvPyrUp(), 131, 132, 299
cvQueryFrame(), 19, 28, 104
cvQueryHistValue_XX(), 198
cvRead(), 85
cvReadByName(), 85
cvReadInt(), 85
cvReadIntByName(), 85
cvRealScalar(), 31, 32
cvRect(), 32, 45
cvRectangle(), 32, 78, 512
cvReduce(), 71–72
cvReleaseCapture(), 19, 104
cvReleaseFileStorage(), 84
cvReleaseHist(), 197
cvReleaseImage(), 18, 19, 24, 25
cvReleaseKalman(), 358
cvReleaseMat(), 34
cvReleaseMemoryStorage(), cvRelease
 MemStorage(), 223
cvReleasePOSITObject(), 413
cvReleaseStructuringElementEx(), 118
cvReleaseVideoWriter(), 27, 106
cvRemap(), 162, 396, 438, 445
cvRepeat(), 72
cvReprojectImageTo3D(), 453, 454
cvResetImageROI(), 45
cvReshape(), 374
cvResize(), 129–130, 512
cvResizeWindow(), 92
cvRestoreMemStoragePos(), 223
cvRetrieveFrame(), 104
cvRodrigues2(), 394, 402
cvRunningAverage(), 276
cvSampleLine(), 270
cvSave(), 83
cvSaveImage(), 92
cvScalar(), 32, 209
cvScalarAll(), 31, 32
cvScale(), 69, 72
cvSegmentMotion(), 346
cvSeqElemIdx(), 226, 227
cvSeqInsert(), 231
cvSeqInsertSlice(), 227
cvSeqInvert(), 228
cvSeqPartition(), 228
cvSeqPop(), cvSeqPopFront(),
 cvSeqPopMulti(), 229
cvSeqPush(), cvSeqPushFront(),
 cvSeqPushMulti(), 229, 231
cvSeqRemove(), 231
cvSeqRemoveSlice(), 227

- cvXX OpenCV functions (*continued*)**
- cvSeqSearch(), 228
 - cvSeqSlice(), 227
 - cvSeqSort(), 228
 - cvSet(), cvSetZero(), 72–73
 - cvSet2D(), 209
 - cvSetCaptureProperty(), 21, 105
 - cvSetCOI(), 68
 - cvSetHistBinRanges(), 197
 - cvSetHistRanges(), 197
 - cvSetIdentity(), 73
 - cvSetImageROI(), 45
 - cvSetMouseCallback(), 97
 - cvSetReal2D(), 39, 209
 - cvSetSeqBlockSize(), 231
 - cvSetSeqReaderPos(), 233
 - cvSetTrackbarPos(), 100
 - cvShowImage(), 17, 22, 93, 94
 - cvSize(), 32, 212
 - cvSlice(), 248
 - cvSobel(), 148–150, 158–160, 173
 - cvSolve(), 73, 75–76
 - cvSplit(), 73–74, 201, 275
 - cvSquareAcc(), 277
 - cvStartAppendToSeq(), 232
 - cvStartFindContours(), 239
 - cvStartReadChainPoints(), 240, 241
 - cvStartReadSeq(), 233, 241
 - cvStartWindowThread(), 94
 - cvStartWriteSeq(), 231
 - cvStartWriteStruct(), 84
 - cvStereoCalibrate(), 407, 427–431, 436, 445
 - cvStereoRectify(), 397, 436–438, 445
 - cvStereoRectifyUncalibrated(), 433, 437, 445, 454
 - cvSub(), 74
 - cvSubdiv2DGetEdge(), 306, 307
 - cvSubdiv2DLocate(), 309, 310–311
 - cvSubdiv2DNextEdge(), 308–310
 - cvSubdiv2DPoint(), 307
 - cvSubdiv2DRotateEdge(), 306, 306, 310–312
 - cvSubdivDelaunay2DInsert(), 304
 - cvSubS(), cvSubRS function, 74, 275
 - cvSubstituteContour(), 239
 - cvSum(), 74–75
 - cvSVBkSb(), 75–76
 - cvSVD(), 75
 - cvTermCriteria(), 258, 299–300, 321, 331
 - cvThreshHist(), 199
 - cvThreshold(), 135–141, 199, 234, 270
- cvTrace(), 76
- cvTransform(), 169, 171, 407
- cvTranspose(), cvT(), 76
- cvTriangleArea(), 312
- cvUndistort2(), 396
- cvUndistortPoints(), 396, 445
- cvUpdateMotionHistory(), 343–346
- cvWaitKey(), 18, 19, 95, 482
- cvWarpAffine(), 162–170, 407
- cvWarpPerspective(), 170, 407, 409
- cvWatershed(), 295
- cvWrite(), 84
- cvWriteFrame(), 27, 106
- cvWriteInt(), 84
- cvXor(), cvXorS(), 76–77
- cvZero(), 77, 178
- CxCore, OpenCV component, 11, 13, 83, 85
- ## D
- DAISY (dense rapidly computed Gaussian scale variant gradients), 524
- DARPA Grand Challenge race, 2, 313–314, 526
- data persistence, 82–86
- data structures
- constructor methods, defined, 31
 - converting, 72
 - handling of, 24
 - image, 32, 42–44
 - matrix, 33–41
 - primitive, 31
 - serializing, 82
 - (see also CvXX OpenCV data structures)
- data types (see CvXX OpenCV data structures; data structures)
- DCT (discrete cosine transform), 182
- de-allocation of memory, 222, 472
- debug builds, 9, 16
- debugging, 267, 383
- decision stumps, 497, 507–509, 516
- decision trees
- advanced analysis, 492
 - binary, 486–495
 - compared, 506
 - creating and training, 487–491
 - predicting, 491
 - pruning, 492–495
 - random, 463, 465, 474, 501–506
- deep copy, 227
- deferred (reinforcement) learning, 461
- degenerate configurations, avoiding, 274, 426

Delaunay triangulation, 14, 301–304, 310–312
dense rapidly computed Gaussian scale variant
 gradients (DAISY), 524
depth maps, 415, 452, 453
deque, 223
detect_and_draw() code, 511
dilation, 115–121
directories, OpenCV, 16
discrete cosine transform (DCT), 182
discriminative models, 462, 483
disparity effects, 405
disparity maps, 415
distance transforms, 185, 187
distortion
 coefficients, defined, 392
 lens, 375–377, 378
documentation, OpenCV, 11–13, 471, 525
dominant point, 246
Douglas-Peucker approximation, 245, 246,
 290–292
download and installation, OpenCV, 8–11
dynamical motion, 354

E

earth mover's distance (EMD), 203, 207–209
edges
 Delaunay, 304–312
 detection, 5, 25, 151–154
 Voronoi, 304–312
 walking on, 306
edible mushrooms example, 470, 488–495, 496,
 499, 503–506
Eigen objects, 13
eigenvalues/eigenvectors, 48, 61, 318–321, 329,
 425
ellipses, 78–79, 120, 248–250
EM (expectation maximization), 462, 463,
 479, 516
EMD (earth mover's distance), 203, 207–209
entropy impurity, 487
epipolar geometry, overview of, 419–421
epipolar lines, 426–427
erosion, 115–121
Eruhimov, Victor, 6
essential matrices, 421–423, 445, 454
estimators (see condensation algorithm;
 Kalman filter)
Euclidean distance, 208, 462
expectation maximization (EM), 462, 463,
 479, 516

F

face recognition
 Bayesian algorithm, 484
 Delaunay points, 303
 detector classifier, 463, 506, 511
 eigenfaces, 55
 Haar classifier, 183, 463, 471, 506–510
 template matching, 214
 training and test set, 459
face recognition tasks, examples of
 shape, clustering by, 461
 orientations, differing, 509, 514
 sizes, differing, 341, 513
 emotions, 303
eyes, 14, 510, 513, 514
features, using, 483
samples, learning from, 515
mouth, 14, 467, 510, 514
age, predicting, 460, 467
temperature difference, using, 342
fast PCA, 54, 55
file
 configuration (logging), 83
 disk, writing to, 27, 105
 header, 16, 31
 information about file, reading, 19
 moving within, 19
 playing video, 18, 27, 105
 pointers, checking, 102
 properties, checking and setting, 104
 querying, 36
 reading images from, 16, 19, 27, 103–105
 signature, 92
Filip, Daniel, 523
filter pipeline, 25
fish-eye (barrel) effect, 375, 376
fish-eye camera lenses, 429
flood fill, 124–129
fonts, 80–82
foreground
 finding objects, 285
 overview of, 265
 segmentation into, 274
foreground versus background, 267
forward projection, problems, 163
forward transform, 179
FOURCC (four-character code), 28, 105
Fourier, Joseph, 177
Fourier transforms, 144, 177–182
frame differencing, 270, 292–294
Freeman chains, 240, 261

Freund, Y., 496
frontal parallel configuration, 416, 417–418,
438, 453
functions (see cvXX OpenCV functions)
fundamental matrix, 405, 421, 423–426, 454

G

Galton, Francis, 216
Gauss, Carl, 177
Gaussian elimination, 60, 65, 66
Gaussian filter, 110–114
Gaussian smooth, 22, 24
GEMM (generalized matrix multiplication),
48, 62, 69
generalized matrix multiplication (GEMM),
48, 62, 69
generative algorithms, 462, 483
geometrical checking, 250
Geometric Blur, 523, 524
geometric manipulations, 163–171
gesture recognition, 14, 193, 194, 342
Gini index (impurity), 487
GLOH (gradient location and orientation
histogram), 524
Google, 1, 523
gradient location and orientation histogram
(GLOH), 524
gradients
 Hough, 158
 morphological, 120, 121–122, 123, 124, 125
 Sobel derivatives and, 148
grayscale morphology, 124
grayscale, converting to/from color, 27, 58–60,
92, 106

H

Haar classifier, 183, 463, 471, 506–510
haartraining, 12, 513–515
Harris corners, 317–319, 321, 329, 383, 524
Hartley's algorithm, 431–433, 439
Hessian image, 317
HighGUI, OpenCV component, 11, 13, 16–19,
21, 90
high-level graphical user interface
 (see HighGUI)
hill climbing algorithm, 337
histogram of oriented gradients (HoG),
523, 524
histograms, 193–213
 accessing, 198
 assembling, 150, 199

chain code (CCH), 262
color, 205, 206
comparing, 201–203, 205
converting to signatures, 208
data structure, 194, 195
defined, 193
dense, 199
equalization, 186–190
grid size problems, 194
intersection, 202
matching methods, 201–202, 206
overview of, 193
pairwise geometrical (PGH), 261–262
homogeneous coordinates, 172, 373, 385–387
homographies
 defined, 163, 371
 dense, 170
 flexibility of, 164, 169
 map matrix, 170, 453
 overview of, 407
 planar, 384–387
 sparse, 171
Horn-Schunk dense tracking method, 316,
322, 335
horopter, 440–442
Hough transforms, 153–160
Hu moments, 253–256, 347, 348
hue saturation histogram, 203–205
human vision (see vision, human)

I

illuminated grid histogram, 203–205
image (projective) planes, 371, 407
Image Processing Library (IPL), 42
image pyramids, 25, 130–135
images
 copying, 57
 creating, 23
 data types, 43
 data types, converting, 56
 displaying, 17, 23, 93
 flipping, 61–62, 107
 formats, 17, 62, 106
 loading, 17, 92
 operators, table of, 48–49
impurity metrics, 486–487
inpainting, 297
installation of OpenCV, 8–11, 16, 31, 87
integral images, 182–185, 508
Integrated Performance Primitives (IPP),
1, 7–10, 86, 179

- Intel Compiler, 516
Intel Corporation, 521
Intel Research, 6
Intel website for IPP, 9
intensity bumps/holes, finding, 115
intentional bias, 493–495, 496
interpolation, 130, 162, 163, 176
intersection method, histograms, 202
intrinsic parameters, defined, 371
intrinsics matrix, defined, 373, 392
inverse transforms, 179
IPAN algorithm, 246, 247
IPL (Image Processing Library), 42
IplImage data structure
 compared with RGB, 32
 element functions, 38, 39
 overview of, 42
 variables, 17, 42, 45–47
IPP (Integrated Performance Primitives),
 1, 7–10, 86, 179
- J**
- Jacobi's method, 61, 406
Jaehne, B., 132
Jones, M. J., 506–511, 515
- K**
- K-means algorithm, 462, 472, 479–483
K-nearest neighbor (KNN), 463, 471, 516
Kalman filter, 350–363
 blending factor (Kalman gain), 357
 extended, 363
 limitations of, 364
 mathematics of, 351–353, 355–358
 OpenCV and, 358–363
 overview of, 350
kernel density estimation, 338
kernels
 convolution, 144
 custom, 118–120
 defined, 115, 338
 shape values, 120
 support of, 144
Kerns, Michael, 495
key-frame, handling of, 21
Konolige, Kurt, 439
Kuriakin, Valery, 6
- L**
- Lagrange multiplier, 336
Laplacian operator, 150–152
- Laplacian pyramid, defined, 131, 132
learning, 459
Lee, Shinn, 6
lens distortion model, 371, 375–377, 378,
 391, 416
lenses, 370
Levenberg–Marquardt algorithm, 428
licensing terms, 2, 8
Lienhart, Rainer, 507
linear transformation, 56
lines
 drawing, 77–78
 epipolar, 454–457
 finding, 25, 153
 (see also Delaunay triangulation)
link strength, 298
Linux systems, 1, 8, 9, 15, 94, 523, 525
Lloyd algorithm, 479
LMedS algorithm, 425
log-polar transforms, 174–177
Lowe, David, 524
Lowe SIFT demo, 464
Lucas–Kanade (sparse) method, 316, 317,
 323–334, 335
- M**
- Machine Learning Library (MLL), 1, 11–13,
 471–475
machine learning, overview of, 459–466
MacOS systems, 1, 10, 15, 92, 94
MacPowerPC, 15
Mahalanobis distance, 49, 66, 462–471,
 476–478
malloc() function, 223
Manhattan distance, 208
Manta open source ray-tracing, 524
Markov random fields (MRFs), 525
masks, 47, 120, 124, 135
matching methods
 Bhattacharyya, 202
 block, 322, 336, 439, 443–444
 contours, 251–259
 hierarchical, 256–259
 histogram, 201–206
 Hu moments, 253–256, 347, 348
 template, 214–219
Matlab interface, 1, 109, 431
matrix
 accessing data in, 34, 36–41
 array, comparison with, 40
 creating, 34, 35

- matrix (*continued*)
 - data types, 32–41
 - element functions, 38, 39
 - elements of, 33
 - essential, 421–423, 445, 454
 - fundamental, 405, 421, 423–426, 454
 - header, 34
 - inverting, 65–66
 - multiplication, 48, 62, 68–69
 - operators, table of, 48–49
- maximally stable external region (MSER), 523, 524
- Maydt, Jochen, 507
- mean-shift segmentation/tracking, 278, 298–300, 337–341, 479
- mechanical turk, 464
- median filter, 110–112
- memory
 - allocation/de-allocation, 222, 472
 - layout, 40, 41
 - storage, 222–234
- misclassification, cost of, 470–471, 487
- missing values, 474, 499
- MIT Media Lab, 6, 341
- MJPEG (motion jpeg), 28
- MLL (Machine Learning Library), 1, 11–13, 471–475
- MLP (multilayer perceptron), 463, 498, 517
- moments
 - central, 254
 - defined, 252
 - Hu, 253–256, 347, 348
 - normalized, 253
- morphological transformations, 115–129
 - Black Hat operation, 120, 123–124, 127
 - closing operation, 120–121, 123
 - custom kernels, 118–120
 - dilation, 115–121
 - erosion, 115–121
 - gradient operation, 120–123, 124, 125
 - intensity images, 116
 - opening operation, 120–121, 122
 - Top Hat operation, 123–124, 126
- motion
 - control, 354
 - dynamical, 354
 - random, 354
- motion jpeg (MJPEG), 28
- motion templates, 341–348
- mouse events, 95–99
- MRFs (Markov random fields), 525
- MSER (maximally stable external region), 523, 524
- multilayer perception (MLP), 463, 498, 517
- mushrooms example, 470, 488–495, 496, 499, 503–506
- ## N
- Newton's method, 326
- Ng, Andrew (web lecture), 466
- nonpyramidal Lucas-Kanade dense optical flow, 329
- normalized template matching, 216
- Numpy, 525
- ## O
- object silhouettes, 342–346
- offset image patches, 524
- onTrackbarSlide() function, 20
- OOB (out of bag) measure, 502
- OpenCV
 - definition and purpose, 1, 5
 - directories, 16
 - documentation, 11–13, 471, 525
 - download and installation, 8–11, 16, 31, 87
 - future developments, 7, 14, 521–526
 - header files, 16, 31
 - history, 1, 6, 7
 - how to use, 5
 - libraries, 16, 23
 - license, 2, 8
 - optimization with IPP, 7, 8, 86
 - portability, 14–15
 - programming languages, 1, 7, 14
 - setup, 16
 - structure and content, 13
 - updates, most recent, 11
 - user community, 2, 6–7
- OpenMP, 516
- operator functions, 48–49
- optical flow, 322–334, 335, 454, 523
- order constraint, 440
- out of bag (OOB) measure, 502
- overfitting (variance), 466–468, 471, 493
- ## P
- pairwise geometrical histogram (PGH), 261–262
- Pearson, Karl, 202
- Peleg, S., 207

- perspective transformations
(see homographies)
- PGH (pairwise geometrical histogram), 261–262
- PHOG, 523
- PHOW (pyramid histogram embedding of other features), 523, 524
- pinhole camera model, 370, 371–373, 391
- pipeline, filter, 25
- Pisarevsky, Vadim, 6
- pixel types, 43
- pixels, virtual, 109, 146
- planar homography, defined, 384
- plumb bob model, 376
- point, dominant, 246
- pointer arithmetic, 38–41, 44
- polar to Cartesian coordinates, 172–174
- polygons, 79–80, 245
- portability guide, 14
- pose, 379, 405, 413
- POSIT (Pose from Orthography and Scaling with Iteration), 412–414
- PPHT (progressive probabilistic Hough transform), 156
- prediction, 349
- primitive data types, 31
- principal points, 372, 415
- principal rays, 415
- probabilistic graphical models, 483
- progressive probabilistic Hough transform (PPHT), 156
- projections, overview of, 405
- projective planes, 371, 407
- projective transforms, 172, 373, 407
- pyramidal Lucas-Kanade optical flow, 329–334, 335
- pyramid histogram embedding of other features (PHOW), 523, 524
- pyramids, image, 25, 130–135
- Python, 1, 9, 523, 525
- R**
- radial distortions, 375–377, 392, 429
- random forests, 501
- random motion, 354
- RANSAC algorithm, 425
- receiver operating characteristic (ROC), 469, 470
- recognition, defined, 461
- recognition by context, 524
- recognition tasks, examples of
blocky features, 510
car in motion, 356
copy detection, 193
depth perception, 522
edible mushrooms, 470, 488–495, 496, 499, 503–506
flesh color, 205, 209–213
flight simulator, 414
flowers, yellow, 469
gestures, 14, 193, 194
hand, 271
local navigation on Mars, 521
microscope slides, processing, 121, 124, 471
novel information from video stream, 56, 265
- object, 175, 212, 214
- person, identity of, 467
- person, motion of, 348–349
- person, presence of, 464, 522
- product inspection, 218, 521
- road, 526
- shape, 262
- text/letter, 463, 517
- tree, windblown, 266–268
- (see also face recognition tasks, examples of; robot tasks, examples of)
- rectangles
bounding, 248
drawing, 78, 107
- parallelogram, converting to, 164
- trapezoid, converting to, 164
- rectification, 430–438
- region of interest (ROI), 43–46, 52
- regression, defined, 461
- regularization constant, 335
- reinforcement (deferred) learning, 461
- remapping, 162
- reprojection, 428, 433–436, 452
- resizing, 129–130, 163
- RGB images, 44, 269
- robot tasks, examples of
camera on arm, 431
car on road, 408
cart, bird's-eye view, 409
objects, grasping, 452, 522
office security, 5
planning, 483, 522
scanning a scene, 475
staples, finding and picking up, 4
- robotics, 2, 7, 405, 453, 521, 524–526

- ROC (receiver operating characteristic), 469, 470
Rodrigues, Olinde, 402
Rodrigues transform, 401–402, 406
ROI (region of interest), 43–46, 52
Rom, H., 207
Rosenfeld-Johnson algorithm, 245
rotation matrix, defined, 379–381
rotation vector, defined, 392
Ruby interface, 1
running average, 276
- S**
- SAD (sum of absolute difference), 439, 443
salient regions, 523
scalable recognition techniques, 524
scalar tuples, 32
scale-invariant feature transform (SIFT), 321, 464, 524
scene modeling, 267
scene transitions, 193
Schapire, R. E., 496
Scharr filter, 150, 343
SciPy, 525
scrambled covariance matrix, 54–55
seed point, 124
segmentation, overview of, 265
self-cleaning procedure, 25
sequences, 134, 223–234
 accessing, 134, 226
 block size, 231
 converting to array, 233
 copying, 227–229
 creating, 224–226
 deleting, 226
 inserting and removing elements from, 231
 moving, 227–229
 partitioning, 229, 230
 readers, 231–233
 sorting, 228
 stack, using as, 229
 writers, 231–233
setup, OpenCV, 16
Shape Context, 523, 524
Shi and Tomasi corners, 318, 321
SHT (standard Hough transform), 156
SIFT (scale-invariant feature transform), 321, 464, 524
silhouettes, object, 342–346
simultaneous localization and mapping (SLAM), 524
singularity threshold, 76
singular value decomposition, 60, 61, 75, 391
SLAM (simultaneous localization and mapping), 524
slider trackbar, 20–22, 99–102, 105, 242
smoothing, 22–24, 109–115
Sobel derivatives, 145, 148–151, 158, 318, 343
software, additional needed, 8
Software Performance Libraries group, 6
SourceForge site, 8
spatial coherence, 324
speckle noise, 117, 443
spectrum multiplication, 179
square differences matching method, 215
stack, sequence as a, 229
standard Hough transform (SHT), 156
Stanford’s “Stanley” robot, 2, 526
statistical machine learning, 467
stereo imaging
 calibration, 427–430, 445–452
 correspondence, 438–445
 overview of, 415
 rectification, 427, 433, 438, 439, 452
stereo reconstruction ambiguity, 432
strong classifiers, 496, 499
structured light, 523
subpixel corners, 319–321, 383, 523
summary characteristics, 247
sum of absolute difference (SAD), 439, 443
Sun systems, 15
superpixels, 265
supervised/unsupervised data, 460
support vector machine (SVM), 463, 470, 517
SURF gradient histogram grids, 523, 524
SVD (singular value decomposition), 60, 61, 75, 391
SVM (support vector machine), 463, 470, 517
switches, 101
- T**
- tangential distortions, 375–377, 378
Taylor series, 375
Teh-Chin algorithm, 245
temporal persistence, 324
test sets, 460–464
text, drawing, 80–82
texture descriptors, 14
textured scene, high and low, 439
thresholds
 actions above/below, 135
 adaptive, 138–141

binary, 139
hysteresis, 152
image pyramids, 133–135
singularity, 76
types, 135
timer function (wait for keystroke), 18, 19
Top Hat operation, 123–124, 126
trackbar slider, 20–22, 99–102, 105, 242
tracking
 corner finding, 316–321
 CvAux, features in, 14
 Horn-Schunk dense method, 316, 322, 335
 identification, defined, 316
 modeling, defined, 316
training sets, 459–464
transforms
 distance, 185–187
 forward, 179
 inverse, 179
 overview of, 144
 perspective, 163
 remapping, 162
 (see also individual transforms)
translation vectors, overview of, 379–381, 392
trees, contour, 235–237, 256, 257
triangulation, 301–304, 310–312, 415–418, 419

U

underfitting (see bias)
undistortion, 396, 445
Unix systems, 95
updates, latest OpenCV, 11
user community, 2, 6–7
user input
 marked objects, 296
 mouse, 95–98
 trackbar, 99–103
 wait for keyboard, 17–21, 95, 483
window functions, 91

V

validation sets, 460
variable importance, 465, 492–495, 496,
 503–506
variables
 global, naming convention, 21
 IplImage, 17, 42, 45–47
variance, finding, 277
variance (overfitting), 466–468, 471, 493

Viola, Paul, 506–511, 515
Viola-Jones rejection cascade (detector),
 506–511, 515
virtual pixels, 109, 146
vision, computer
 applications of, 1–5, 121, 265, 267
 challenges of, 2–5, 370, 464
 defined, 2
 (see also recognition tasks, examples of)
vision, human, 2–3, 14, 174, 370, 517
Visual Studio, 16
Voronoi iteration, 479
Voronoi tessellation, 301–312

W

walking on edges, 306
warping, 163–166
watercolor effect, 114
watershed algorithm, 295–297
weak classifiers, 463, 496–501, 507, 516
weak-perspective approximation, 413
Werman, M., 207
whitening, data, 471
widthStep image parameter, 43–47
Wiki sites, OpenCV, 8, 12, 471
Willow Garage, 7, 521
Win32 systems, 62, 92, 95
Windows
 OpenCV installation, 8–11
 portability, 15
windows
 clean up, 18, 91, 94
 closing, 18, 91, 94
 creating, 17, 22, 91, 242
 moving, 94
 names versus handles, 92
 properties of, defining, 17
 resizing, 92
wrapper function, 24

Y

Yahoo groups forum, 2

Z

Zhang's method, 389
Zisserman's approximate nearest neighbor
 suggestion, 525
zooming in/out, 129

关于作者和译者

Gary Rost Bradski 博士是斯坦福大学计算机科学系人工智能实验室的顾问教授，负责指导机器人、机器学习和计算机视觉方面的研究。同时，他还是 Willow Garage(<http://www.willowgarage.com>)的高级科学家。Willow Garage 是最近新成立的一个机器人研究所/孵化器。Gary 拥有加州大学伯克利分校学士学位，波士顿大学博士学位。他在机器学习和计算机视觉方面有 20 年的从业经验。先后在美国第一联合国家银行从事期权交易操作，在 Intel 研究院从事计算机视觉方面的工作，在 Intel 制造中心从事机器学习方面的工作，期间还参与创办过几个公司。

Gary 是开放源代码计算机视觉库(OpenCV，<http://sourceforge.net/projects/opencvlibrary>)的发起人，该库目前已被广泛应用于世界各地的研究所、政府部门和商业应用。他还发起了统计机器学习库(已包含入 OpenCV)和概率网络库(PNL)。计算机视觉库帮助开发了商业的 Intel 集成性能基元(IPP，<http://tinyurl.com/36ua5s>)很重要的一部分。Gary 还帮助组织了 Stanley 视觉团队。Stanley 是斯坦福的一个机器人，它在 DARPA 的无人驾驶大挑战赛中穿越沙漠赢得了两百万美金。他与 Andrew Ng 教授一起，在斯坦福建立了斯坦福人工智能机器人项目(<http://www.cs.stanford.edu/group/stair>)。Gary 发表了 50 多篇论文，有已经授权的专利 13 项，正在申请中的专利 18 项。他和妻子以及三个女儿居住于美国加州的 Palo Alto，闲暇时钟情于在公路或山区骑自行车。

Adrian Kaehler 博士是 Applied Minds 公司的高级科学家。他目前主要研究机器学习、统计建模、计算机视觉和机器人。Adrian 于 1998 年从哥伦比亚大学获得理论物理博士学位。随后他在 Intel 公司和斯坦福大学人工智能实验室任职。他也是赢得 DARPA 大挑战赛的 Stanley 机器人团队成员之一。他在物理、电气工程、计算机科学和机器人领域有许多论文和专利。

于仕琪博士供职于中国科学院深圳先进技术研究院，担任助理研究员。OpenCV 中文网站 (<http://www.opencv.org.cn>) 的主要维护人。曾担任 2007 年北京 OpenCV 研讨会组织者，邀请 OpenCV 开发者 Vadim Pisarevsky 和其他一些专业人员前来布道。目前主要研究方向是计算机视觉和模式识别。

刘瑞桢博士是中国国内 OpenCV 推广的先行者，第一个 OpenCV 中文论坛的创办人。毕业于中国科学院自动化研究所模式识别国家重点实验室，目前从事智能图像识别与机器视觉方面的产业化工作。

封面图片

《学习 OpenCV(中文版)》的封面是一只大型、漂亮的孔雀蝶。原产于欧洲，孔雀蝶分布在法国南部和意大利、伊比利亚半岛、西伯利亚和北非部分地区。它们栖息于地形开阔的零星分布的树与灌木丛中，因此人们通常可以在稀树草原、果园和葡萄园中看到它们的踪影，它们白天就在这些遮阴树林下休息。

作为欧洲最大的一种蝶，孔雀蝶的翼展长达 6 英寸，它们的大小和喜欢夜出的本性使得某些观察者经常把它们误认为是蝙蝠。它们的翅膀是灰色和浅灰色的，上面洒着白色和黄色的圆点。在孔雀蝶每一个翅膀的中间，都有一个很大的眼点，这是一种与众不同的图案，与孔雀蝶这一名称密切相关。

本书封面图片来自 Cassell 的 *Natural History*(第 5 卷)。

