应用型高校产教融合系列教材

大电类专业系列

计算机视觉入门与综合实践案例

栾新源　张荣琪　黄奇志 ◎ 编著

清華大學出版社

北 京

内 容 简 介

本书是针对应用型本科或者职教本科课程偏重应用的特点编写的一本入门级的计算机视觉实践教材。本书介绍 OpenCV、HALCON、VisionMaster 等常用的计算机视觉项目开发环境，讲解基础常用算法并开发配套例程，通过亲自运行修改例程，读者对知识点掌握更加深刻。还提供多个如陶瓷外观缺陷检测、风电桨叶外观缺陷检测、激光充电目标跟踪等具有代表性的综合案例，讲解项目开发基本流程并提供项目程序代码。读者学完本课程后基本能独立完成一般性缺陷检测类项目，成为智能制造技术人才。

图书在版编目（CIP）数据

计算机视觉入门与综合实践案例 / 栾新源，张荣琪，黄奇志编著. 北京 ：清华大学出版社，2025. 1. --（应用型高校产教融合系列教材）.
ISBN 978-7-302-68081-9

Ⅰ. TP302.7

中国国家版本馆 CIP 数据核字第 20259LR898 号

责任编辑：王　欣
封面设计：何凤霞
责任校对：薄军霞
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-83470000　　**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 装 者：小森印刷霸州有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：9.75　　**字　　数**：234 千字
版　　次：2025 年 1 月第 1 版　　**印　　次**：2025 年 1 月第1 次印刷
定　　价：48.00 元

产品编号：109003-01

总 编 委 会

编 委 会

丛书序

FOREWORD

教材是知识传播的主要载体、教学的根本依据、人才培养的重要基石。《国务院办公厅关于深化产教融合的若干意见》明确提出，要深化“引企入教”改革，支持引导企业深度参与职业学校、高等学校教育教学改革，多种方式参与学校专业规划、教材开发、教学设计、课程设置、实习实训，促进企业需求融入人才培养环节。随着科技的飞速发展和产业结构的不断升级，高等教育与产业界的紧密结合已成为培养创新型人才、推动社会进步的重要途径。产教融合不仅是教育与产业协同发展的必然趋势，更是提高教育质量、促进学生就业、服务经济社会发展的有效手段。

上海工程技术大学是教育部“卓越工程师教育培养计划”首批试点高校、全国地方高校新工科建设牵头单位、上海市“高水平地方应用型高校”试点建设单位，具有 40 多年的产学合作教育经验。学校坚持依托现代产业办学、服务经济社会发展的办学宗旨，以现代产业发展需求为导向，学科群、专业群对接产业链和技术链，以产学研战略联盟为平台，与行业、企业共同构建了协同办学、协同育人、协同创新的“三协同”模式。

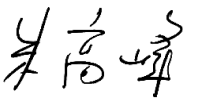
在实施“卓越工程师教育培养计划”期间，学校自 2010 年开始陆续出版了一系列卓越工程师教育培养计划配套教材，为培养出具备卓越能力的工程师作出了贡献。时隔 10 多年，为贯彻国家有关战略要求，落实《国务院办公厅关于深化产教融合的若干意见》，结合《现代产业学院建设指南（试行）》《上海工程技术大学合作教育新方案实施意见》文件精神，进一步编写了这套强调科学性、先进性、原创性、适用性的高质量应用型高校产教融合系列教材，深入推动产教融合实践与探索，加强校企合作，引导行业企业深度参与教材编写，提升人才培养的适应性，旨在培养学生的创新思维和实践能力，为学生提供更加贴近实际、更具前瞻性的学习材料，使他们在学习过程中能够更好地适应未来职业发展的需要。

在教材编写过程中，始终坚持以习近平新时代中国特色社会主义思想为指导，全面贯彻党的教育方针，落实立德树人根本任务，质量为先，立足于合作教育的传承与创新，突出产教融合、校企合作特色，校企双元开发，注重理论与实践、案例等相结合，以真实生产项目、典型工作任务、案例等为载体，构建项目化、任务式、模块化、基于实际生产工作过程的教材体系，力求通过与企业的紧密合作，紧跟产业发展趋势和行业人才需求，将行业、产业、企业发展的新技术、新工艺、新规范纳入教材，使教材既具有理论深度，能够反映未来技术发展，又具有实践指导意义，使学生能够在学习过程中与行业需求保持同步。

系列教材注重培养学生的创新能力和实践能力。通过设置丰富的实践案例和实验项目，引导学生将所学知识应用于实际问题的解决中。相信通过这样的学习方式，学生将更加

具备竞争力，成为推动经济社会发展的有生力量。

本套应用型高校产教融合系列教材的出版，既是学校教育教学改革成果的集中展示，也是对未来产教融合教育发展的积极探索。教材的特色和价值不仅体现在内容的全面性和前沿性上，更体现在其对于产教融合教育模式的深入探索和实践上。期待系列教材能够为高等教育改革和创新人才培养贡献力量，为广大学生和教育工作者提供一个全新的教学平台，共同推动产教融合教育的发展和创新，更好地赋能新质生产力发展。



中国工程院院士、中国工程院原常务副院长

2024 年 5 月

前 言

PREFACE

本书是一本针对应用型本科或职教本科课程偏重应用的特点编写的入门级计算机视觉实践教材。本书介绍 OpenCV、HALCON、VisionMaster 等常用的计算机视觉项目开发环境，讲解基础常用算法并开发配套例程，通过亲自运行修改例程，使读者能够对知识点掌握更加深刻。还提供了多个如陶瓷外观缺陷检测、风电桨叶外观缺陷检测、激光充电目标跟踪等具有代表性的综合案例，讲解项目开发基本流程并提供项目程序代码。读者学习后基本能独立完成一般性缺陷检测类项目，提升智能制造技术水平。


本书例程源代码

本书共 7 章，第 1 章简介计算机视觉发展历史、开发环境；第 2 章介绍计算机视觉要用到的图像基础知识，包含图像定义、颜色空间、图像卷积原理等；第 3 章介绍图像几何变换的知识；第 4 章介绍常用的图形特征检测方法，包含边缘检测、USAN 算子、哈里斯角点检测、霍夫变换、轮廓提取；第 5 章介绍目标图像分割方法及综合示例；第 6 章介绍目标跟踪；第 7 章介绍项目综合案例。

本书主要由电子电气工程学院人工智能产业研究院编写，上海文化广播影视集团有限公司和中国物资再生协会纤维复合材料再生分会参与编写，栾新源等老师编写第 1～7 章，张荣琪、黄奇志等参与编写第 7 章的案例。

感谢上海工程技术大学“应用型高校产教融合系列教材”总编委会、“大电类系列”编委会提供参与编写该系列教材的机会。

由于编者水平有限，书中难免存在疏漏和不足之处，恳请广大读者批评指正。

编　者

2024 年 7 月

目 录

CONTENTS

第1章 计算机视觉概述 / 1

1.1 计算机视觉简介 / 1
1.2 项目开发典型软件环境 / 2
1.2.1 OpenCV / 2
1.2.2 HALCON / 3
1.2.3 VisionMaster / 6
1.3 计算机视觉趣味范例 / 8
1.4 本书构成框架 / 9
习题 / 10

第2章 图像预处理 / 11

2.1 图像基础 / 12
2.1.1 图像定义 / 12
2.1.2 图像文件格式 / 13
2.1.3 颜色空间 / 13
2.1.4 像素邻域 / 16
2.2 直方图均衡化 / 16
2.2.1 直方图均衡原理 / 16
2.2.2 直方图均衡化的缺点 / 16
2.2.3 直方图均衡化程序示例 / 16
2.3 图像卷积 / 18
2.3.1 卷积原理 / 18
2.3.2 卷积运算 / 19
2.4 图像滤波 / 20
2.4.1 线性平滑滤波 / 20
2.4.2 非线性平滑滤波 / 20

2.4.3 线性锐化滤波 / 20
2.4.4 滤波函数示例 / 21
2.5 图像形态学 / 21
2.5.1 图像膨胀 / 22
2.5.2 图像腐蚀 / 22
2.5.3 开闭运算 / 22
2.5.4 形态学梯度 / 22
2.5.5 图像形态学示例 / 22
2.6 小结 / 24
习题 / 24

第3章 图像几何变换 / 25

3.1 边界链码表达 / 26
3.2 基于曲率的形状分析 / 26
3.2.1 曲率与几何特征 / 26
3.2.2 曲面曲率 / 27
3.3 图像仿射变换 / 27
3.3.1 仿射变换概念 / 27
3.3.2 仿射变换公式 / 27
3.3.3 图像平移及例程 / 28
3.3.4 图像旋转缩放及例程 / 29
3.3.5 图像翻转及例程 / 32
3.3.6 函数直接生成转换矩阵及例程 / 34
3.4 图像透视变换及例程 / 35
3.5 重映射及例程 / 36
3.6 图像缩放 / 38
3.6.1 仿射变换缩放 / 38
3.6.2 resize 函数缩放 / 38
3.6.3 图像金字塔 / 40
3.7 图像翻转 / 43
3.8 小结 / 44
习题 / 44

第4章 基元检测 / 45

4.1 边缘检测 / 46
4.1.1 检测原理 / 46
4.1.2 一阶导数算子 / 46

4.1.3 二阶导数算子 / 47
4.1.4 边缘检测算子比较 / 48
4.1.5 边缘检测示例 / 48
4.2 USAN 算子 / 50
4.2.1 USAN 原理 / 50
4.2.2 USAN 算子的特点 / 51
4.3 哈里斯角点检测 / 51
4.3.1 哈里斯角点检测特点 / 51
4.3.2 哈里斯角点检测示例 / 51
4.4 霍夫变换 / 52
4.4.1 霍夫变换检测直线原理 / 52
4.4.2 霍夫变换检测直线示例 / 53
4.4.3 改进霍夫变换检测直线 / 55
4.4.4 改进霍夫变换检测直线示例 / 55
4.4.5 霍夫变换检测圆原理 / 56
4.4.6 改进霍夫变换检测圆 / 57
4.4.7 霍夫变换检测圆示例 / 57
4.5 轮廓提取 / 59
4.5.1 轮廓提取相关函数 / 59
4.5.2 轮廓提取示例 / 60
4.6 小结 / 62
习题 / 62

第5章 图像分割 / 63

5.1 基于边缘的分割方法 / 63
5.2 基于阈值的分割方法 / 63
5.2.1 阈值分割原理与种类 / 63
5.2.2 全局阈值选取 / 64
5.2.3 自动获取阈值 / 65
5.3 米粒图像分割综合示例 / 65
5.3.1 软件环境 / 65
5.3.2 实验 1：转灰度图 / 66
5.3.3 实验 2：边缘检测与形态学 / 67
5.3.4 实验 3：阈值分割 / 70
5.3.5 实验 4：米粒计数 / 72
5.4 小结 / 75
习题 / 76

第6章 目标识别与跟踪 / 77

6.1 背景建模 / 78
6.1.1 建模原理 / 78
6.1.2 典型背景建模方法 / 78
6.1.3 高斯混合建模示例 / 78
6.2 粒子滤波器 / 80
6.3 运动光流 / 80
6.4 卷积神经网络 / 81
6.4.1 卷积操作 / 82
6.4.2 激活函数 / 82
6.4.3 池化 / 83
6.4.4 深度神经网络 / 83
6.4.5 全连接层 / 83
6.4.6 卷积神经网络 / 84
6.5 小结 / 84
习题 / 84

第7章 综合实践案例 / 85

7.1 检测盒结果自动读取 / 85
7.1.1 研究背景 / 85
7.1.2 软件编写 / 85
7.1.3 项目总结 / 89
7.2 陶瓷马桶外观缺陷检测 / 89
7.2.1 总体实施方案 / 89
7.2.2 开发环境配置 / 90
7.2.3 VS 中设计执行界面 / 93
7.2.4 基于 VM 检测缺陷代码设计 / 93
7.2.5 基于 OpenCV 检测代码设计 / 96
7.2.6 检测效果 / 97
7.2.7 项目总结 / 98
7.3 药瓶激光雕刻编码识别 / 98
7.3.1 项目背景 / 98
7.3.2 图像接入 / 99
7.3.3 图像预处理 / 99
7.3.4 仿射变换 / 99
7.3.5 字符识别 / 100

7.3.6 项目效果 / 102
7.3.7 项目总结 / 103
7.4 无人机白激光充电 / 103
7.4.1 项目背景 / 103
7.4.2 YOLOv5 网络 / 103
7.4.3 激光充电系统 / 104
7.4.4 软件系统设计 / 105
7.4.5 软件测试 / 106
7.4.6 软件测试结果分析 / 107
7.4.7 项目代码 / 109
7.4.8 项目总结 / 118
7.5 风机叶片表面缺陷检测 / 119
7.5.1 项目背景 / 119
7.5.2 缺陷种类 / 119
7.5.3 扩充图像数据集 / 119
7.5.4 YOLOv8 算法 / 121
7.5.5 YOLOv8 添加 ECA / 121
7.5.6 检测效果 / 123
7.5.7 项目总结 / 124
7.6 基于视觉的光通信平衡码编解码设计 / 124
7.6.1 项目背景 / 124
7.6.2 光通信平衡码编解码方法 / 125
7.6.3 YOLOv8 识别定位 / 128
7.6.4 图像处理解码 / 129
7.6.5 项目代码 / 132
7.6.6 项目总结 / 137

参考文献 / 138

第 1 章　计算机视觉概述

1.1　计算机视觉简介

计算机视觉是指对图像、视频、3D 点云或多维数据分析得到的特征进行分析，提取场景的语义表示，让计算机或其他计算系统具有人眼和人脑的能力。它涉及多个领域，如图像处理、模式识别、机器学习、深度学习等。

近年来，中国的计算机视觉研究与应用迎来了快速发展时期，涌现出一批如华为海思、海康威视、大华等世界级公司。中国在计算机视觉领域拥有丰富的人才资源和巨大的市场需求，在机器视觉、人脸识别、智能物流、无人驾驶等领域处于世界先进地位。图 1-1 所示为我国月球车涉及的视觉导航。

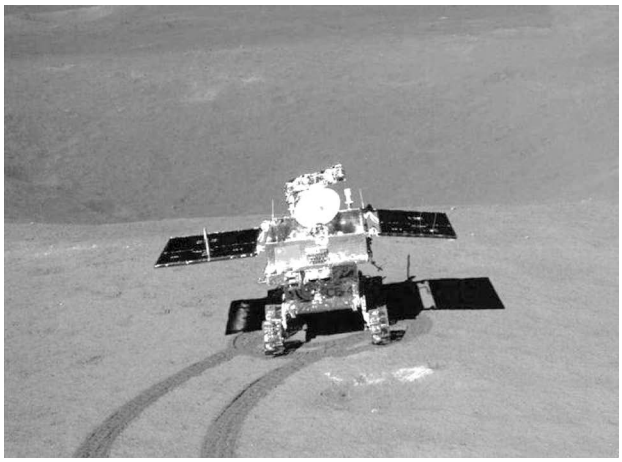


图 1-1　月球车自主导航

总体来说，计算机视觉技术的发展历程是一个算法不断创新及算力持续提升推动的过程。随着技术的不断进步和应用场景的不断扩展，计算机视觉将会在未来的科技发展中发挥更加重要的作用。

1.2 项目开发典型软件环境

1.2.1 OpenCV

OpenCV(Open Source Computer Vision Library)是一个基于 Apache 2.0 许可(开源)发行的跨平台计算机视觉和机器学习软件库。

OpenCV 主要用于开发图像处理、计算机视觉以及模式识别程序。该软件库的主要特点如下。

(1) 编程语言：OpenCV 基于 C++实现，同时提供 Python、Ruby、MATLAB 等语言的接口。OpenCV-Python 是 OpenCV 的 Python API(应用程序编程接口)，结合了 OpenCV C++ API 和 Python 语言的最佳特性。

(2) 跨平台：OpenCV 可以在不同的系统平台上使用，包括 Windows、Linux、OS X、Android 和 iOS。

(3) 广泛应用：OpenCV 广泛应用于计算机视觉和机器学习领域，包括人脸识别、物体检测、图像分割、运动跟踪等。

OpenCV 有多个版本，每个版本都有一些改进和新特性。

OpenCV 4.x 版带来了更多的新特性和改进，如全新的 ONNX(开放的机器学习模型表示格式)层，大大提高了 DNN(深度神经网络)代码的卷积性能。OpenCV 4.0 引入了大量新功能和改进，包括更好的性能和稳定性，以及一些新的 API 和模块。例如，它引入了改进的 DNN 模块，支持深度学习，还引入了改进的图像处理和计算机视觉算法。此外，OpenCV 4.0 还支持更多种类的 GPU 加速，可提高处理速度。

1. OpenCV 的下载、安装与配置

安装 Python 后，推荐使用 PyCharm 作为代码编辑、调试环境。本书后续 Python 代码皆在 PyCharm 编辑器中编写、调试。

安装 OpenCV 的具体步骤如下。

使用 pip 安装 OpenCV 的 Python 库。在命令行中输入以下命令安装 OpenCV：

```
pip install
opencv - python
```

如果要安装 OpenCV 的完整版，包括额外的功能和算法，可以使用以下命令：

```
pip install
opencv - contrib - python
```

这将安装 OpenCV 及其所有附加组件。

安装完成后，可以在 Python 代码中导入 OpenCV 库，如下所示：

```
import cv2
```

现在就可以使用 OpenCV 的功能处理图像和视频。例如，使用以下代码读取一幅图像

并显示：

```
img = cv2.imread('image.jpg')          # 也可以是 BMP 格式
cv2.imshow('Image', img)               # 显示图像
cv2.waitKey(0) # 按任意键关闭窗口
cv2.destroyAllWindows() # 关闭所有窗口
```

直接通过 OpenCV 官网下载资源，通常网络速度比较慢。国内有许多可用的 OpenCV 镜像源，例如清华大学开源软件镜像站、阿里云、中国科学技术大学镜像站等。这些镜像源提供了快速下载，并且通常与官方源同步。

要使用 OpenCV 镜像源，需要在安装过程中指定镜像源的 URL(统一资源定位器)。例如，如果使用清华大学的镜像源安装 OpenCV，可以按照以下步骤进行操作。

输入以下命令安装 OpenCV：

```
pip install opencv-python -i https://pypi.tuna.tsinghua.edu.cn/simple
```

上述命令中的-i 选项用于指定镜像源的 URL。也可将 https://pypi.tuna.tsinghua.edu.cn/simple 替换为其他镜像源的 URL。使用镜像源可以大大加快 OpenCV 的下载、安装速度，特别是在网络条件不佳的情况下。

其他常用插件介绍如下。

numpy：OpenCV 绑定 Python 时依赖的库，这意味着必须安装 numpy。

Tabnine：用于自动填充代码。

Rainbow Brackets：将括号以不同的颜色标注出来。

Indent Rainbow：为不同层级缩进的空格标注不同的颜色。

此外，可能还需要安装其他一些依赖项，具体取决于应用需求和系统环境。例如，如果需要使用特定的摄像头或传感器，可能需要安装相应的驱动程序或库。

2. OpenCV 官方例程

打开浏览器，访问 OpenCV 官方网站下载例程。完成后，将安装文件(通常是一个压缩文件)解压到选择的目录中。解压文件后，在相应的文件夹中找到“samples”目录，该目录包含 OpenCV 的官方例程。

Canny 边缘检测：例程演示如何使用 Canny 算法进行边缘检测。

霍夫直线变换：例程演示如何使用霍夫变换检测图像中的直线。

特征匹配：例程演示如何使用特征匹配算法在两幅图像之间进行特征匹配。

摄像头标定和三维重建：例程演示如何使用 OpenCV 进行摄像头标定和三维重建。

目标跟踪：例程演示如何进行目标跟踪，如使用 MeanShift(均值漂移)算法。

图像拼接：例程演示如何使用 OpenCV 进行图像拼接，如全景图像拼接。

OCR(光学字符识别)：例程演示如何使用 OCR 引擎进行光学字符识别。

动态分析：例程演示如何使用 OpenCV 进行动态分析，如运动跟踪和光流法。

1.2.2 HALCON

HALCON 是一款由德国 MVTec 公司开发的机器视觉软件，广泛应用于自动化技术、质量检验、医疗分析和工业测量等领域。该软件以强大的图像处理和分析能力著称，提供一

系列复杂的算法支持，包括 2D 和 3D 图像分析、机器学习以及深度学习功能，如物体识别、缺陷检测、光学字符识别（OCR）和条形码读取。它通过一个名为 HDevelop 的集成开发环境，允许用户以视觉方式创建和测试图像分析程序，同时支持 C、C++、C＃和 Python 等编程语言，为用户提供极大的灵活性。Halcon 的另一个特点是其高度的可扩展性和兼容性，支持常见厂家的工业相机和 3D 传感器。

1. HALCON 下载与安装

进入 HALCON 中文官网下载界面，如图 1-2 所示，选择需要的版本及深度学习工具下载。

图 1-2　HALCON 中文官网下载界面

例如，下载 HALCON 22.11 的安装包压缩文件，并解压到 HALCON-22.11.0.0-x64-win64 文件夹，右击 som.exe 文件，选择以管理员身份运行。单击右上角环境按钮，设置程序和数据路径（建议选择 D 盘）。

组件选择：选择安装包，建议选择全部，如图 1-3 所示。

图 1-3　安装组件

许可协议：单击接受。

安装证书：将下载好的证书复制到“…\\HALCON-XX.XX-Progress\\license”文件夹

下即可，图 1-4 所示，授权激活完成，软件可正常使用。正式 license(许可证书)需要购买，如图 1-4 所示为试用版 license。MVTec 每月释放一次试用 license，可从 CSDN 等网站获取。

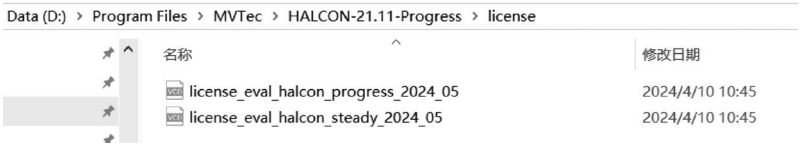


图 1-4　license 路径

2. HALCON 官方例程

启动软件后，如图 1-5 所示，在文件菜单下可看到示例程序(快捷键 Ctrl+E)。

图 1-5　示例程序

图 1-6 是关于光学字符识别的实例程序(以下简称“例程”)。

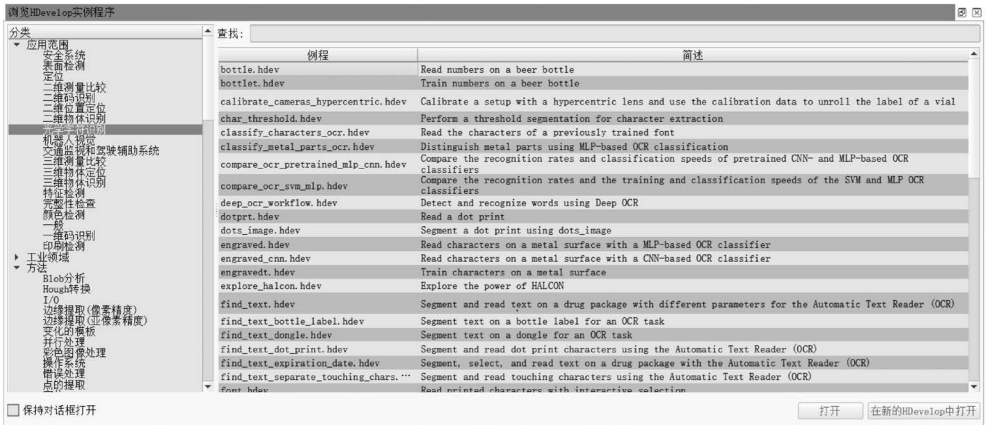


图 1-6　丰富的实例程序

1.2.3 VisionMaster

中国海康威视(Hikvision)开发的 VisionMaster(简称 VM)软件是一款功能强大的机器视觉软件,专为工业自动化和智能检测应用设计。VM 提供了丰富的视觉工具和算法,能够满足各种复杂的视觉检测需求,广泛应用于制造、电子、半导体、食品和饮料等行业,成为工业自动化和智能检测领域的重要工具。

1. VisionMaster 下载与安装

安装包可从官网下载,如图 1-7 所示,单击选择“服务支持→下载中心→软件”。

图 1-7 VM 下载

如图 1-8 所示,从“类型”下拉菜单中选择“平台”,下载 VM 基础安装包、深度学习安装包、VM 示例程序。

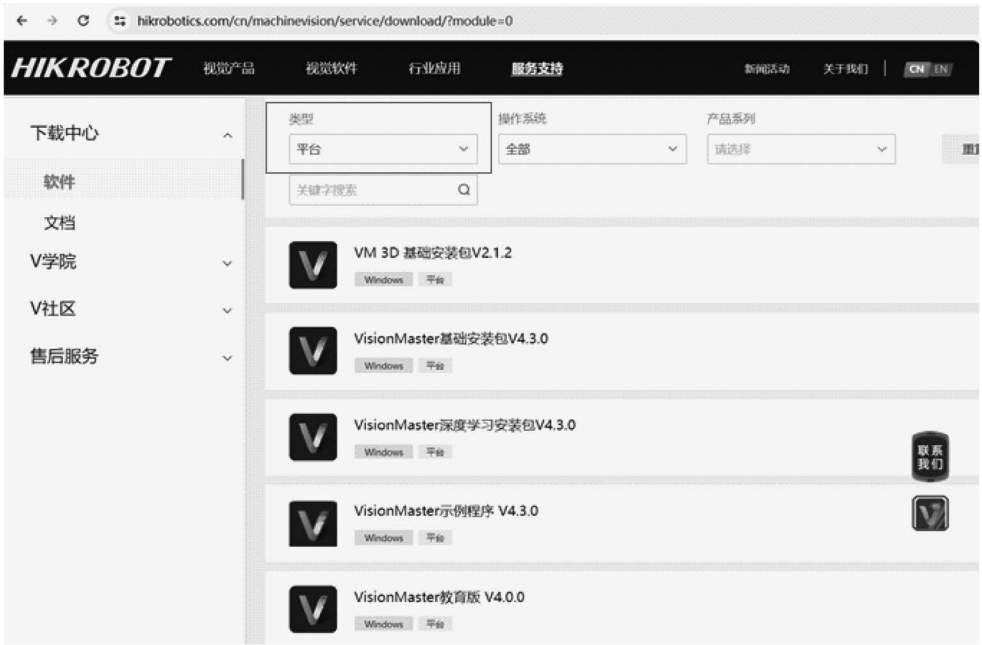


图 1-8 VM 下载界面

如图 1-9 所示,在“安装选项”加密方式中选择软加密或者加密狗。软加密有试用版,正式版或者加密狗需要购买。

图 1-9　加密方式

2. VisionMaster 官方例程

在 VM 主菜单的“文件”中单击“打开示例”即可看到例程；V 学院提供了大量学习视频，对入门用户非常友好；V 社区提供了交流学习空间以及竞赛方案。

VM 算法开发平台提供了完全图形化的交互界面，如图 1-10 所示，功能图标直观易懂，简单好用的交互逻辑以及拖拽式操作能够快速搭建视觉方案。软件优秀的交互和视觉效果设计，出众的用户体验，吸引了大量行业用户。

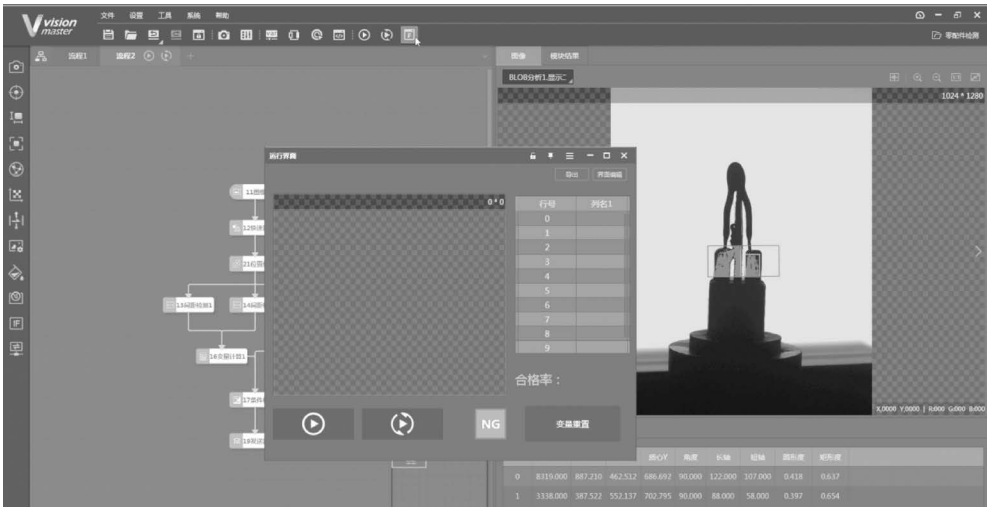


图 1-10　打开示例及拖拽式开发

VM 算法开发平台配备了高性能深度学习算法，经过大量案例验证、优化后的算法对常见检测品都有良好的适应性。深度学习算法提供了图像分割、分类，模板检测，字符定位与识别，图像检索，异常检测等算法模块；还提供了独立训练工具进行图像打标训练，可高效完成深度学习模块的应用。

1.3 计算机视觉趣味范例

下面的 Python 例程使用颜色直方图匹配器进行目标跟踪。它首先读取一个视频文件，然后创建一个颜色直方图匹配器。在循环中，它读取每一帧，计算颜色直方图匹配，并进行目标跟踪。如果跟踪成功，则在窗口中绘制一个矩形表示目标，如图 1-11 所示。最后释放资源并关闭窗口。可以根据实际情况修改颜色直方图的阈值和跟踪窗口的大小等参数以适应不同的应用场景。

彩图 1-11

图 1-11　颜色跟踪

例程 1-1　颜色目标跟踪

```
import cv2
import numpy as np
# 打开视频文件
cap = cv2.VideoCapture('video.mp4')
# 读取视频的第一帧
ret, frame = cap.read()
# 将图像从 BGR 颜色空间转换到 HSV 颜色空间
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
# 创建一个掩码，筛选出特定 HSV 值范围内的区域
# 设置 HSV 的范围以识别橙色
# H: 5～17 (橙色到黄色的范围)
# S: 100～255 (排除较低的饱和度，避免灰色地板干扰)
# V: 50～255 (亮度不要太低，以避免在非常暗的区域误识别)
```

```
mask = cv2.inRange(hsv, np.array([5, 100, 50]), np.array([17, 255, 255])) # 根据掩码计算
HSV图像色调(H通道)的直方图
hist = cv2.calcHist([hsv], [0], mask, [180], [0, 180])
# 归一化直方图,使其范围为0～255
cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX)
# 设置初始跟踪窗口的位置和大小
track_window = (0, 0, frame.shape[1], frame.shape[0])
# 循环读取视频帧
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # 转换颜色空间
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # 计算反向投影
    dst = cv2.calcBackProject([hsv], [0], hist, [0, 180], 1)
    # 应用CamShift算法进行目标跟踪
    ret, track_window = cv2.CamShift(dst, track_window, (cv2.TERM_CRITERIA_EPS | cv2.TERM_
CRITERIA_COUNT, 10, 1))
    # 获取旋转矩形的4个顶点
    pts = cv2.boxPoints(ret)
    pts = np.int0(pts)
    # 在图像上绘制轮廓
    cv2.polylines(frame, [pts], True, 255, 2)
    # 显示跟踪结果
    cv2.imshow('Tracking', frame)
    # 按'q'键退出循环
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break
# 释放视频资源并关闭窗口
cap.release()
cv2.destroyAllWindows()
```

1.4 本书构成框架

本书共 7 章,各章内容如图 1-12 所示,具体安排如下。

第 1 章,简介计算机视觉发展历史、开发环境及趣味范例,让读者有一个初步的学习轮廓。

第 2 章和第 3 章的知识在图像预处理阶段都会用到。

第 2 章,介绍计算机视觉要用到的图像基础知识,包含图像定义、颜色空间、图像卷积原理等。还介绍直方图均衡化、图像滤波、图像形态学图像预处理方法。

第 3 章,介绍图像几何变换的知识;介绍仿射变换中通过自定义转换矩阵及用函数生成转换矩阵两种方式实现平移、旋转、缩放、翻转;介绍透视变换和重映射;介绍实现图像缩放、翻转的其他 3 种函数。

第 4 章,介绍常用的图形特征检测方法,包含边缘检测、USAN 算子、哈里斯角点检测、霍夫变换、轮廓提取。

学习完第 2～4 章的知识后，可具备将感兴趣区域从图像中分割出来的能力。

第 5 章，介绍目标图像分割方法及综合示例。

第 6 章，进一步介绍目标跟踪。

第 7 章，介绍项目综合案例。

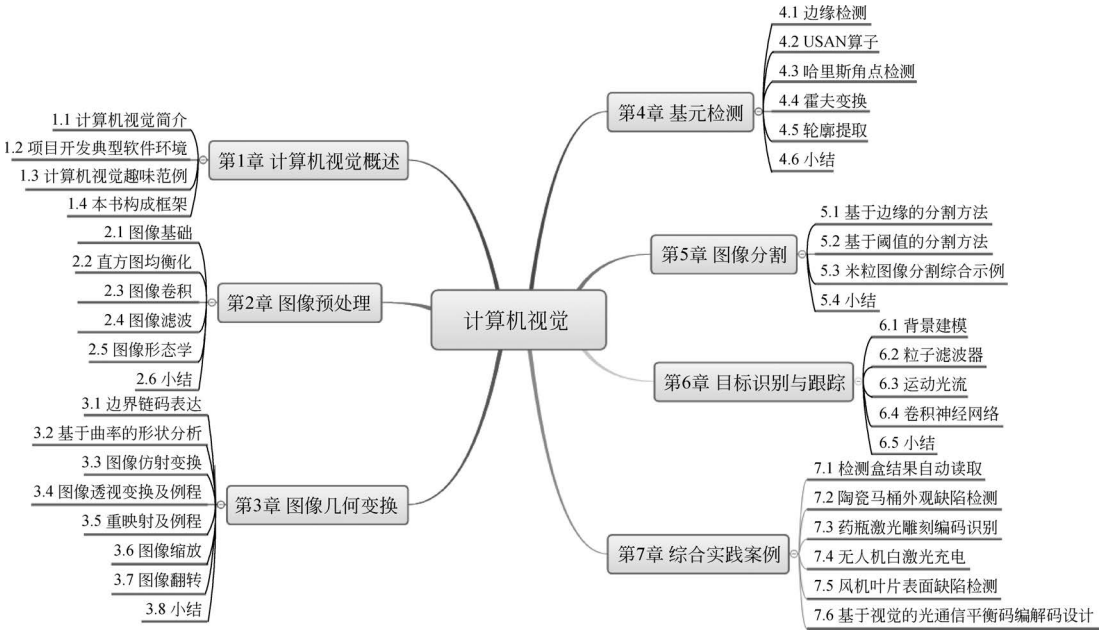


图 1-12　各章内容

习题

1.1　计算机视觉有哪些应用领域？

1.2　计算机视觉项目开发用到的典型软件有哪些？

第 2 章　图像预处理

本章主要介绍计算机视觉要用到的图像基础知识，如图 2-1 所示，包含图像定义、图像文件格式、颜色空间、像素邻域、图像卷积原理等。还介绍图像预处理方法，包含直方图均衡化、图像滤波、图像形态学。本章是计算机视觉课程中需要重点掌握的知识。

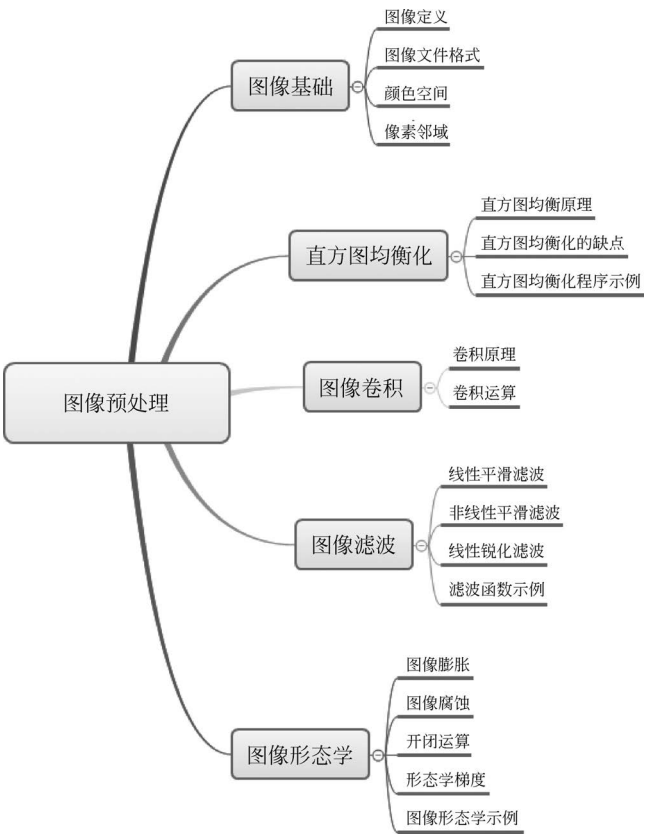


图 2-1　图像预处理基础知识

2.1 图像基础

2.1.1 图像定义

1. 模拟图像

模拟图像(analog image)是指用模拟信号表示的图像,它与数字图像相对。在模拟图像中,图像信息是通过连续变化的信号表示的,而不是通过离散的数字值。以下是模拟信号图像的一些关键特点。

(1) 连续性:模拟图像的信号是连续的,没有像素的概念,图像的亮度和颜色信息通过信号的强度连续变化表示。

(2) 无量化误差:由于信号是连续的,模拟图像没有数字图像中的量化误差,理论上可以提供无限的分辨率。

(3) 易受干扰:模拟信号容易受到噪声和干扰的影响,导致图像质量下降。

(4) 不易存储和传输:与数字信号相比,模拟信号不易长期存储和远距离传输,因为信号会随着距离的增加而衰减。

(5) 处理复杂:模拟图像的处理通常需要专门的硬件,如模拟电路,这使处理过程比数字图像更复杂、昂贵。

(6) 应用领域:模拟图像主要应用于传统的电视广播、视频监控和一些医学成像技术(如 X 光片)等领域。

2. 数字图像

数字图像(digital image)是通过将模拟图像数字化得到的图像或者由数字化设备创建的图像,如图 2-2 所示,图像中的 1 个像素(pixel)一般包含 3 个子像素(sub-pixel),由一系列数值表示,包含颜色和亮度信息。数字图像可以是二维的,也可以是三维的(如 TIFF 图)。


彩图 2-2

B	G	R	B	G	R	B	G	R
0	0	255	0	128	255	255	128	0
0	0	255	0	128	255	255	128	0
0	0	255	0	128	255	255	128	0

图 2-2　数字图像子像素值与颜色

数字图像的基本属性如下。

(1) 分辨率:由图像中的像素数量决定,通常以水平像素×垂直像素(如 1920×1080)表示。

(2) 颜色深度:决定每个像素可以表示的颜色数量,常见的有 8 位、16 位、24 位或 32 位。例如,8 位图像可以表示 256 种颜色,而 24 位图像可以表示 1600 万种以上的颜色。

(3) 颜色空间:图像中使用的颜色模型,如 RGB(红、绿、蓝)、CMYK(青色、品红、黄色、黑色)或灰度。

(4) 压缩:为减少存储空间、加快传输速度,数字图像经常使用各种压缩算法,如

JPEG、PNG、GIF 等。

(5) 文件格式：存储图像数据的文件类型，如 JPEG、PNG、BMP、TIFF 等。

2.1.2 图像文件格式

1. BMP 格式

BMP(全称 bitmap)是 Windows 操作系统中的标准图像文件格式，可以分为两类：设备相关位图(DDB)和设备无关位图(DIB)，应用非常广。采用位映射存储格式，除了图像深度可选以外，不进行其他任何压缩，因此，BMP 文件所占的空间很大。BMP 文件的图像深度可选 lb、4b、8b 及 24b。BMP 文件存储数据时，图像的扫描方式按照从左到右、从下到上的顺序。

2. GIF 格式

GIF 是图形交换格式(graphics interchange format)的简称。GIF 的图像深度从 lb 到 8b，即 GIF 最多支持 256 种色彩的图像。GIF 格式的另一个特点是在一个 GIF 文件中可以存储多幅彩色图像，如果把存储于一个文件中的多幅图像数据逐幅读出并显示到屏幕，就可构成一种最简单的动画。由于其具有压缩比高、解码速度快、支持透明背景和动画效果等优点，在网络和多媒体中得到广泛应用。

3. TIFF 格式

TIFF(tag image file format)是一种灵活的位图格式，主要用于存储包括照片和艺术图在内的图像。最初由 Aldus 公司与微软公司一起为 PostScript 打印开发。TIFF 与 JPEG 和 PNG 一起成为流行的高位彩色图像格式。TIFF 格式在业界得到了广泛的支持，桌面印刷和页面排版应用，扫描、传真、文字处理、光学字符识别和其他一些应用等都支持这种格式。

4. JPEG 格式

JPEG(joint photographic experts group)是一种有损压缩的图像文件格式。这种格式的文件扩展名为. jpg 或. jpeg。

JPEG 是一种广泛应用于静态图像压缩的标准，特别适用于连续色调静态图像的压缩。它采用预测编码、离散余弦变换(DCT)以及熵编码等联合编码方式，去除冗余的图像和彩色数据，在较小的储存空间内一定程度地损失图像数据。其压缩比率通常为 10～40 倍，压缩比越大，图像品质越低。

5. PNG 格式

PNG(portable network graphics)是一种采用无损压缩算法的位图格式，其设计目的是替代 GIF 和 TIFF 文件格式，同时增加一些 GIF 文件格式不具备的特性。PNG 格式支持索引、灰度、RGB 三种颜色方案以及 Alpha 通道等特性，并具有透明背景。它还支持 Gamma 校正和 16 位通道，可以在不同的系统和应用程序中保持一致的颜色与亮度。

2.1.3 颜色空间

图像颜色空间，也称为彩色模型或彩色空间，是描述和表示图像中颜色的一种方式。它是一种坐标系统和子空间的阐述，每种颜色在颜色空间中由单个点表示。采用的大多数颜色模型都是面向硬件或面向应用的。在计算机图像处理和相关领域中，有多种常用的颜色空间，包括 RGB、HSV/HSL、YUV 等。

1. RGB 颜色空间

RGB(red,green,blue)颜色空间是最常见的面向硬件设备的彩色模型,如图 2-3 所示。它与人的视觉系统紧密相连,根据人眼的结构,所有的颜色都可以看作三种基本颜色——红色、绿色和蓝色的不同比例的组合。在 RGB 颜色空间中,每幅图像包括三个独立的基色子图像,每种颜色的亮度用 0～255 表示。通过改变这三种颜色通道的值及其之间的叠加,可以得到超过 1670 万种颜色。

彩图 2-3

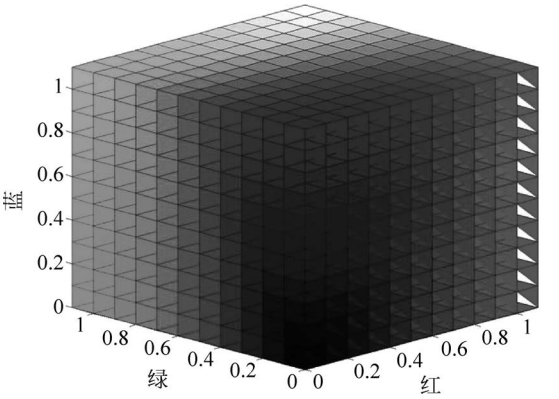


图 2-3　RGB 颜色空间

24 位数字图像按 B、G、R 顺序存储三个分量,B、G、R 三个分量值不同,就会显示不同的色彩。

图像矩阵从左上角开始,左上角是(0,0)位置,向右是 X 坐标,向下是 Y 坐标,X 坐标对应图像的列,也就是图像的宽度;Y 坐标对应图像的行,也就是图像的高度。

2. HSV 和 HSL 颜色空间

这两种颜色空间更接近人对颜色的感知方式。HSV(hue,saturation,value)代表色调、饱和度与亮度,而 HSL(hue,saturation,lightness)代表色调、饱和度与明度,如图 2-4 所示。

彩图 2-4

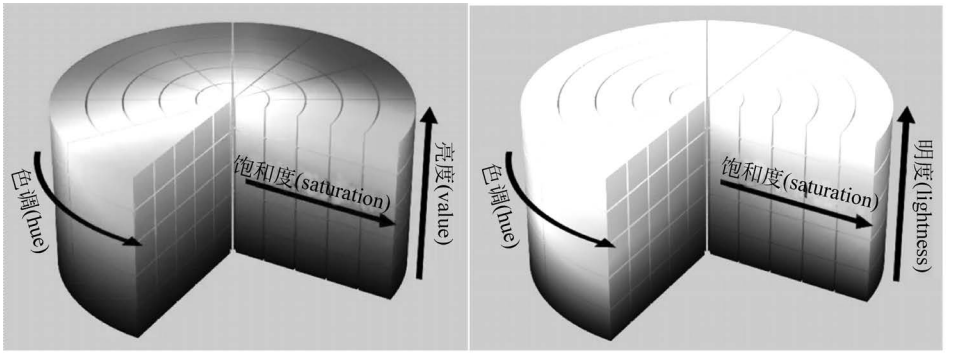


图 2-4　HSV 和 HSL 颜色空间

HSV 颜色空间是为了更好地数字化处理颜色而提出的。在 HSV 颜色空间下,比在 BGR 颜色空间下更容易跟踪某种颜色的物体,常用于分割指定颜色的物体。

Hue 用角度度量,取值范围为 0～360°,表示色彩信息,即所处的光谱颜色的位置。表

示如下。

颜色圆环上所有的颜色都是光谱上的颜色，从红色开始按逆时针方向旋转，Hue＝0 表示红色，Hue＝120 表示绿色，Hue＝240 表示蓝色，等等。

在 RGB 中，颜色由三个值共同决定，比如黄色为(255，255，0)；在 HSV 中，黄色只由一个值决定，Hue＝60 即可。

其中，水平方向表示饱和度，饱和度表示颜色接近光谱色的程度。饱和度越高，说明颜色越深，越接近光谱色；饱和度越低，说明颜色越浅，越接近白色。饱和度为 0 表示纯白色。取值范围为 0～100％，值越大，颜色越饱和。

垂直方向表示明度，决定颜色空间中颜色的明暗程度，明度越高，表示颜色越明亮，范围是 0～100％。明度为 0 表示纯黑色(此时颜色最暗)。在 Hue 一定的情况下，饱和度减小，就是向光谱色中添加白色，光谱色所占的比例也会减小，饱和度减为 0，表示光谱色所占的比例为零，导致整个颜色呈现白色。明度减小，就是向光谱色中添加黑色，光谱色所占的比例也会减小，明度减为 0，表示光谱色所占的比例为零，导致整个颜色呈现黑色。

3. XYZ 颜色空间

国际照明委员会(CIE)进行了大量正常人视觉测量和统计，1931 年建立了“标准色度观察者”，从而奠定了现代 CIE 标准色度学的定量基础。由于“标准色度观察者”用于标定光谱色时会出现负刺激值，计算不便，也不易理解，因此 1931 年 CIE 在 RGB 系统基础上，改用三种假想的原色 X、Y、Z 建立了一个新的色度系统，叫作“CIE1931 标准色度系统”。CIE XYZ 颜色空间稍加变换就可得到 Yxy 色彩空间，其中 Y 取三刺激值中 Y 的值表示亮度，x、y 反映颜色的色度特性。但是，在这一空间中，两种不同颜色间的距离值并不能正确地反映人们色彩感觉差别的大小，也就是说，在 CIE Yxy 中，不同位置不同方向上颜色的宽容量是不同的，这就是 Yxy 颜色空间的不均匀性。这一缺陷的存在，导致在 Yxy 及 XYZ 空间不能直观地评价颜色。

4. CMYK 颜色空间

CMYK(cyan，magenta，yellow，black)颜色空间应用于印刷工业。印刷业通过青(C)、品(M)、黄(Y)三原色油墨的不同网点面积率的叠印表现丰富多彩的颜色和阶调，这便是三原色的 CMY 颜色空间。实际印刷中，一般采用青 (C)、品(M)、黄(Y)、黑(BK)四色印刷，在印刷的中间色调至暗色调间增加黑色，这种模型称为 CMYK。

5. YUV 颜色空间

在现代彩色电视系统中，通常采用三管彩色摄像机或彩色 CCD(电荷耦合器件)摄像机，它把摄得的彩色图像信号，经分色、分别放大校正得到 RGB，再经过矩阵变换电路得到亮度信号 Y 和两个色差信号 R-Y、B-Y，最后发送端将亮度和两个色差信号分别进行编码，用同一信道发送出去。这就是我们常用的 YUV 色彩空间。采用 YUV 色彩空间的重要性在于它的亮度信号 Y 和色度信号 U、V 是分离的。如果只有 Y 信号分量而没有 U、V 分量，那么这样表示的图就是黑白灰度图。彩色电视采用 YUV 空间正是为了用亮度信号 Y 解决彩色电视机与黑白电视机的兼容问题，使黑白电视机也能接收彩色信号。当白光的亮度用 Y 表示时，它与红、绿、蓝三色光的关系可用下式描述：$Y=0.3R+0.59G+0.11B$，这就是常用的亮度公式。

6. 颜色空间相互转换 OpenCV 函数

将图像转换为 HSV 色彩空间：

```
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

再将 HSV 图像转换回 BGR 色彩空间：

```
bgr_image_eq = cv2.cvtColor(hsv_image_eq, cv2.COLOR_HSV2BGR)
```

2.1.4 像素邻域

(1) 邻域像素：像素 p 周围最邻近的像素，称为邻域像素。

(2) 像素之间的邻接性。

4 邻域：像素 p 的坐标是(x,y)，那么它的 4 邻域坐标 N_4 是$(x+1,y)$、$(x-1,y)$、$(x,y+1)$、$(x,y-1)$。

对角邻域：点 p 的对角邻域像素坐标 N_D 为$(x+1,y+1)$、$(x+1,y-1)$、$(x-1,y+1)$、$(x-1,y-1)$。

8 邻域：$N_D+N_4=N_8$。

2.2 直方图均衡化

2.2.1 直方图均衡原理

直方图均衡化的基本思想是把原始图的直方图变换为在整个灰度范围内均匀分布的形式，以增加像素灰度值的动态范围，从而达到增强图像整体对比度的效果。

直方图均衡化是一种简单、有效的图像增强技术，通过改变图像的直方图改变图像中各像素的灰度，主要用于增强动态范围偏小的图像对比度。由于原始图像灰度分布可能集中在较窄的区间，造成图像不够清晰。例如，过曝光图像的灰度级集中在高亮度范围内，而曝光不足将使图像灰度级集中在低亮度范围内。

直方图均衡化的基本原理是：对在图像中像素数多的灰度值(对画面起主要作用的灰度值)进行展宽，而对像素数少的灰度值(对画面不起主要作用的灰度值)进行归并，从而增大对比度，使图像清晰。如图 2-5 所示，左图为原始图像，右图为直方图均衡化后的图像。

2.2.2 直方图均衡化的缺点

如果一幅图像整体偏暗或者偏亮，那么直方图均衡化的方法很适用。但直方图均衡化是一种全局处理方式，它对处理的数据不加选择，可能会增加背景干扰信息的对比度并降低有用信号的对比度(如果图像某些区域对比度很高，而另一些区域对比度不高，采用直方图均衡化就不一定适用)。此外，均衡化后图像的灰度级减少，某些细节将消失；某些图像(如直方图有高峰)经过均衡化后对比度不自然地过分增强。

2.2.3 直方图均衡化程序示例

以下是灰度图直方图均衡化的一个 Python 示例，展示了如何使用 OpenCV 的

cv2.equalizeHist()函数对图像进行直方图均衡化。

例程 2-1 灰度图直方图均衡化

```
import cv2
# 读取图像
image = cv2.imread('image3.2.3.jpg', cv2.IMREAD_GRAYSCALE)
# 进行直方图均衡化
equ_image = cv2.equalizeHist(image)
# 显示原始图像和均衡化后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Equalized Image', equ_image)
# 等待用户按键退出
cv2.waitKey(0)
cv2.destroyAllWindows()
```

这段代码首先导入必要的库,然后读入一个灰度图像。如果图像成功加载,它将使用cv2.equalizeHist()函数进行直方图均衡化。然后,它将显示原始图像和均衡化后的图像,如图 2-5 所示,并等待用户按键操作后退出。显然,偏暗的原始图像均衡化后效果更好。

(a) (b)

图 2-5 直方图均衡化效果

(a) 原始图像;(b) 均衡化后的图像

如果想要对彩色图像进行直方图均衡化,可以对每个颜色通道分别进行均衡化,如以下例程所示。

例程 2-2 彩色图像直方图均衡化

```
import cv2
# 读取图像
image = cv2.imread('image3.2.3.jpg')
# 将图像转换为 HSV 色彩空间
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# 分离 HSV 图像的各个通道
h, s, v = cv2.split(hsv_image)
# 对 V 通道进行直方图均衡化
equ_v = cv2.equalizeHist(v)
```

```
# 合并均衡化后的 V 通道回到 HSV 图像
hsv_image_eq = cv2.merge([h, s, equ_v])
# 将均衡化后的 HSV 图像转换回 BGR 色彩空间
bgr_image_eq = cv2.cvtColor(hsv_image_eq, cv2.COLOR_HSV2BGR)
# 显示原始图像和均衡化后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Equalized Image', bgr_image_eq)
# 等待用户按键退出
cv2.waitKey(0)
cv2.destroyAllWindows()
```

在这个 Python 示例中，首先将原始图像（图 2-6(a)）转换为 HSV 色彩空间，其次对 V(value，亮度)通道进行直方图均衡化，最后将结果转换回 BGR 色彩空间（图 2-6(b)），原来偏暗的图像经过均衡化后图像细节更清晰。这种方法在某些情况下可以提供比在 BGR 空间直接进行均衡化更好的视觉效果。

彩图 2-6

(a) (b)

图 2-6 彩色图直方图均衡化

(a) 原始图像；(b) 均衡化后的图像

2.3 图像卷积

2.3.1 卷积原理

卷积模板(mask)，又称卷积核(kernel)、滤波器或算子。卷积核是一个小的矩阵，通常为二维(2D)，用于与图像进行卷积运算，以产生新的图像或特征图。

图像卷积是指一个卷积模板和另一个待处理图像矩阵进行卷积，卷积模板的锚点(中心点)对准待处理矩阵元素，卷积模板整体覆盖在待处理矩阵上面。然后计算被覆盖的元素值与卷积模板中值的乘积，再相加求和。将这个和赋给当前元素，就是卷积的过程。

假设有待处理矩阵 src，模板是 kernel。

$$\mathrm{dst}(x,y)=\sum_{\substack{0\leqslant i<\mathrm{kernel.\,cols}\\0\leqslant j<\mathrm{kernel.\,rows}}}\mathrm{kernel}(i,j)*\mathrm{src}(x+i-\mathrm{anchor}.x,y+j-\mathrm{anchor}.y)\qquad(2\text{-}1)$$

anchor(x,y)是卷积模板(kernel)锚点(中心点)在待处理矩阵 src 上的坐标。

图像卷积模板的特点如下。

(1) 卷积模板像素数一般是奇数,按照核中心对称,一般是 3×3、5×5 或者 7×7。中心到边称为半径,例如 5×5 大小的核的半径就是 2。

(2) 为了与原始图像的亮度保持大体一致,模板所有的元素之和一般等于 1。相对地,如果模板所有元素之和大于 1,那么卷积后的图像就会比原图像亮;反之,如果小于 1,那么得到的图像就会变暗。

(3) 卷积运算后,可能出现负数或者大于 255 的数值,直接截断到 0～255 即可。对于负数,也可以取绝对值。

2.3.2 卷积运算

假设卷积模板 kernel 如图 2-7 所示。

待处理矩阵 src 如图 2-8 所示。

求 kernel * src,步骤如下。

(1) 将卷积模板旋转 180°,如图 2-9 所示。

1	2	1
0	0 anchor	0
−1	−2	−1

图 2-7 卷积模板 kernel

col

row

1 col=0,row=0	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

图 2-8 待处理矩阵 src

−1	−2	−1
0	0	0
1	2	1

图 2-9 卷积模板旋转 180°

(2) 将卷积模板 kernel 的中心对准 src 的第一个元素,然后 kernel 与 src 重叠的元素相乘,kernel 中不与 src 重叠的地方用 0 代替,再将相乘后 kernel 对应的元素相加,得到结果矩阵中 dst 的第一个元素。如图 2-10 所示。

所以结果矩阵中的第一个元素 dst(0,0)=−1×0+(−2)×0+(−1)×0+0×0+0×1+0×2+1×0+2×5+1×6=16。

(3) src 中的每个元素都用这样的方法计算,得到的卷积结果矩阵如图 2-11 所示。

−1×0	−2×0	−1×0		
0×0	0×1	0×2	3	4
1×0	2×5	1×6	7	8
	9	10	11	12
	13	14	15	16

图 2-10 卷积运算

16	24	28	23
24	32	32	24
24	32	32	24
−24	−40	−44	−35

图 2-11 卷积结果矩阵

实际应用中,负数截止为 0。平滑、模糊、锐化、去噪、边缘提取等工作都可以通过卷积操作完成。

2.4 图像滤波

图像滤波是指在尽量保留图像细节特征的条件下，改善图像质量、去除噪声或实现特定效果。

2.4.1 线性平滑滤波

OpenCV 线性平滑滤波包括均值滤波、高斯滤波、方框滤波三种。

1. 均值滤波(mean filtering)

系数均为正值，接近模板中心的系数比较大，而模板边界附近的系数比较小。通过将每个像素的值替换为其邻域内像素值的平均数减少噪声。实现简单，但可能会模糊图像细节。

cv2.blur()：对图像进行均值滤波，去除噪声。它使用一个滑动窗口，计算窗口内所有像素值的平均数。

2. 高斯滤波(Gaussian filtering)

使用高斯函数作为权重计算邻域内像素的平均值。比均值滤波更平滑，常用于减少图像噪声。

cv2.GaussianBlur()：使用高斯权重的均值滤波器，对图像进行平滑处理。

3. 方框滤波(box filter)

cv2.boxFilter()：实现简单的方框滤波器，可以作为均值滤波器使用。

2.4.2 非线性平滑滤波

OpenCV 非线性平滑滤波包括中值滤波和双边滤波两种。

1. 中值滤波(median filtering)

用一个窗口内所有像素值的中值替换窗口中心的像素值。对于去除椒盐噪声(salt-and-pepper noise)特别有效，且能保留图像细节。

cv2.medianBlur()：使用中值代替均值以减少噪声。

中值滤波可通过如下步骤完成。

(1) 使模板在图中漫游，并将模板中心与图中某个像素位置重合。

(2) 读取模板覆盖下的目标图像像素的灰度值。

(3) 将这些灰度值从小到大排成一列。

(4) 找出这些灰度值中排在中间的一个。

(5) 将这个中间值赋给对应模板中心位置的像素。

2. 双边滤波(bilateral filtering)

cv2.bilateralFilter()：在保留边缘信息的同时减少噪声，通过像素的强度和空间邻近度计算权重。

2.4.3 线性锐化滤波

利用求导的方法可以对图像进行锐化滤波。线性锐化滤波的模板仅中心系数为正，周围的系数均为负值。典型的拉普拉斯(Laplacian)模板如图 2-12 所示。

−1	−1	−1
−1	8	−1
−1	−1	−1

图 2-12　拉普拉斯模板示例

cv2. Laplacian()：用于图像锐化和边缘检测，计算图像的二阶导数。

2.4.4 滤波函数示例

以下是使用 OpenCV 进行平滑滤波的 Python 示例代码。

例程 2-3 平滑滤波

```
import cv2
# 读取图像
image = cv2.imread('image3.4.4.png', cv2.IMREAD_GRAYSCALE)
# 均值滤波
mean_filtered = cv2.blur(image, (3, 3))
# 中值滤波
median_filtered = cv2.medianBlur(image, 3)
# 高斯滤波
gaussian_filtered = cv2.GaussianBlur(image, (5, 5), 0)
# 显示原始图像和滤波后的图像
cv2.imshow('Original Image', image)
cv2.imshow('Mean Filtered', mean_filtered)
cv2.imshow('Median Filtered', median_filtered)
cv2.imshow('Gaussian Filtered', gaussian_filtered)
# 等待用户按键退出
cv2.waitKey(0)
cv2.destroyAllWindows()
```

如图 2-13 所示，从滤波效果来看，中值滤波后图像清晰，均值滤波后图像有些模糊，高斯滤波后图像比较模糊。

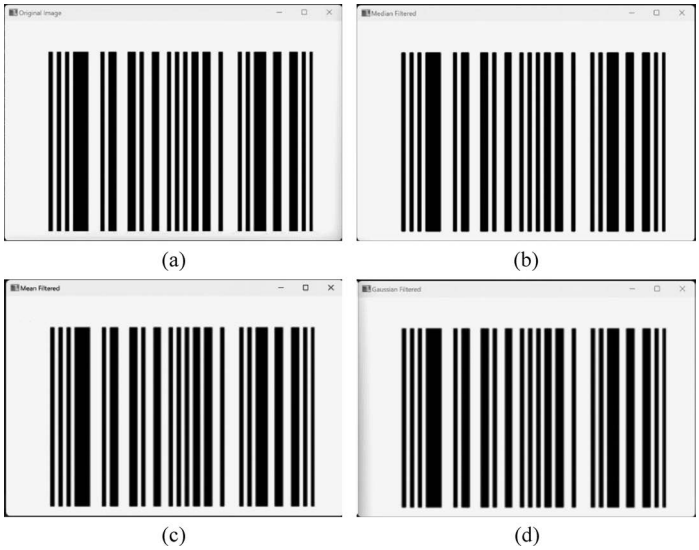


图 2-13 滤波效果

(a) 原始图像；(b) 中值滤波；(c) 均值滤波；(d) 高斯滤波

2.5 图像形态学

两种基本形态学操作包括膨胀与腐蚀；5 种高级形态学滤波操作包括开运算、闭运算、形态学梯度、顶帽及黑帽。

2.5.1 图像膨胀

图像膨胀(dilation)是一种增加图像中白色(前景)物体大小的操作。它通过使用一个称为结构元素的模板实现,该模板在图像上滑动,如果结构元素与图像重叠的区域完全由白色像素组成,则在结构元素的中心位置将黑色像素变为白色。这样物体的边界会向外扩展,从而增加物体的面积。

膨胀操作具有如下功能。

(1) 连接相邻物体:通过膨胀可以使原本分离的物体连接起来。

(2) 增加物体尺寸:可以使物体的边界向外扩展,从而增加其尺寸。

(3) 填补小的间隙:可以填补物体内部的小孔或间隙。

2.5.2 图像腐蚀

与膨胀相反,图像腐蚀(erosion)是一种减小图像中白色物体大小的操作。它使用与膨胀相同的结构元素,但是规则不同:只有当结构元素完全覆盖在白色像素上时,结构元素中心位置的黑色像素才会变为白色。这意味着物体的边界会向内收缩。

腐蚀操作具有如下功能。

(1) 移除小的物体或细节:可以移除图像中小的白色物体或细节。

(2) 减小物体尺寸:可以使物体的边界向内收缩,减小其尺寸。

(3) 揭示物体的确切边界:通过腐蚀可以更清晰地看到物体的边界。

2.5.3 开闭运算

膨胀和腐蚀操作经常结合使用,以实现更复杂的图像处理任务。

开运算(opening):先腐蚀后膨胀。有助于移除小的物体或细节,并可以平滑物体的边界。

闭运算(closing):先膨胀后腐蚀。有助于填充小的间隙,并可以使物体的边界更清晰。

2.5.4 形态学梯度

形态学梯度(morphological gradient):形态学膨胀与腐蚀之差。对二值图进行这一操作可以突出团块(blob)的边缘,进而用于保留物体的边缘轮廓。

顶帽(top hat):原始图像与开运算结果图之差。得到的效果图突出了比原图轮廓更明亮的区域。

黑帽(black hat):闭运算结果图与原始图像之差。得到的效果图突出了比原图轮廓更暗的区域。

2.5.5 图像形态学示例

以下 Python 代码将显示原始图像及经过膨胀、腐蚀、开运算和闭运算后的图像。

例程 2-4 形态学示例

```
import cv2
# 读取图像
image = cv2.imread('image3.5.5.png', cv2.IMREAD_GRAYSCALE)
# 创建结构元素,这里使用 5×5 的矩形
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
# 图像膨胀操作
dilated_image = cv2.dilate(image, kernel, iterations = 2)
# 图像腐蚀操作
eroded_image = cv2.erode(image, kernel, iterations = 2)
# 开运算
opened_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
# 闭运算
closed_image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
# 显示结果
cv2.imshow('Original', image)
cv2.imshow('Dilated', dilated_image)
cv2.imshow('Eroded', eroded_image)
cv2.imshow('Opened', opened_image)
cv2.imshow('Closed', closed_image)
# 等待用户按键退出
cv2.waitKey(0)
cv2.destroyAllWindows()
```

上述形态学示例代码中函数含义如下。

cv2. dilate 函数用于图像膨胀。

cv2. erode 函数用于图像腐蚀。

iterations＝1 参数表示操作执行的次数，可以根据需要进行调整。

cv2. morphologyEx 函数用于执行开运算和闭运算，其中 cv2. MORPH_OPEN 和 cv2. MORPH_CLOSE 是指定操作类型的参数。

如图 2-14 所示，原始图像及经过膨胀后白色区域变大，腐蚀后白色区域变小。开运算后轮廓清晰，闭运算后扩大白色区域连通域。

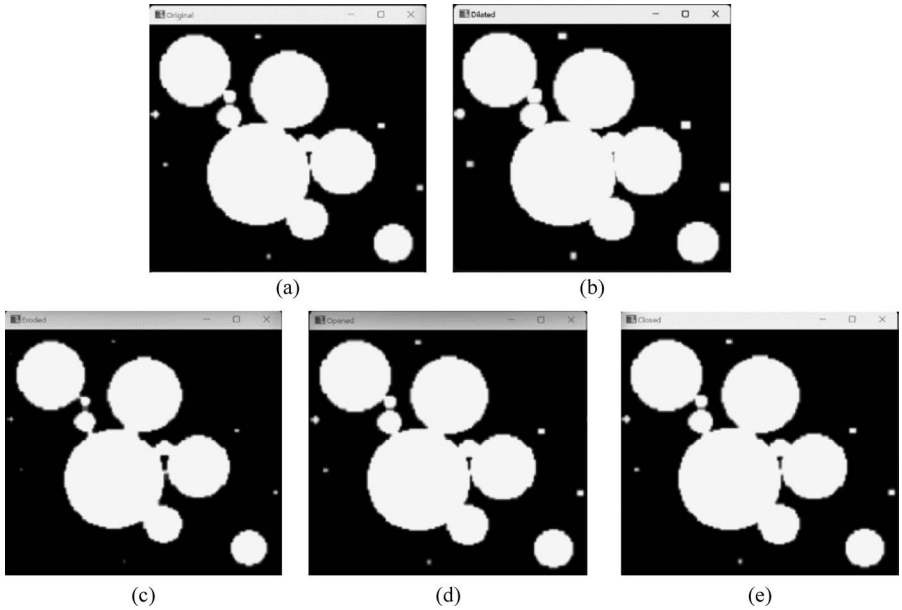


图 2-14　形态学

(a) 原始图像；(b) 图像膨胀操作；(c) 图像腐蚀操作；(d) 开运算；(e) 闭运算

2.6 小结

本章介绍了计算机视觉入门基础知识及图像预处理方法，读者应当掌握多种图像滤波方法、图像均衡方法、图像形态学方法。

习题

2.1 总结对比几种颜色空间的适用场景。

2.2 简述中值滤波特点与步骤。

2.3 对图 2-8 所示矩阵分别进行均值滤波(用图 2-7 模板)、中值滤波(3×3 模板)和拉普拉斯滤波(图 2-12)。

2.4 开闭运算的作用是什么?

2.5 计算机存储图像的原点坐标在什么位置?

第 3 章　图像几何变换

本章主要介绍图像几何变换的知识，如图 3-1 所示，包括图像边界链码表达及曲率与几何关系，仿射变换的公式推导，仿射变换实现平移、旋转、缩放、翻转，用函数生成转换矩阵，透视变换，重映射，resize 函数图像缩放，图像金字塔，flip 函数图像翻转。

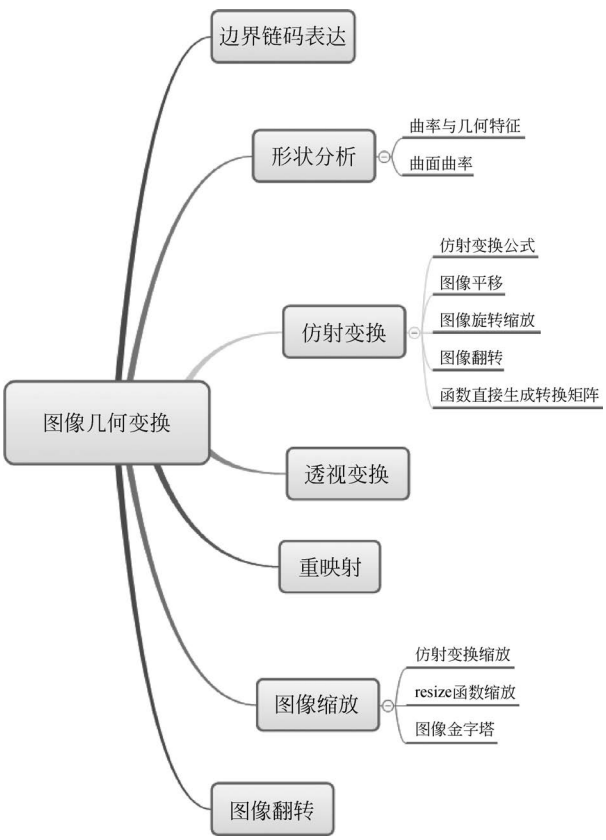


图 3-1　图像几何变换的知识

3.1 边界链码表达

只有边界的起点需用(绝对)坐标表示,其余点都可只用接续方向代表偏移量。4-方向链码和 8-方向链码的共同特点是线段的长度固定,方向数有限。

1. 链码起点归一化

给定一个从任意点开始而产生的链码,把它看作一个由各个方向数构成的自然数。将这些方向数按一个方向循环,使它们构成的自然数的值最小,然后将这样转换后的链码起点作为归一化链码的起点。

2. 链码旋转归一化

利用链码的一阶差分重新构造一个序列,该序列具有旋转不变性。若差分为负值,则加上 4(4-方向链码)或 8(8-方向链码)。

3. 边界形状数

形状数是值最小的链码差分码。

阶数定义为形状数序列的长度。

图 3-2 以 4-方向链码为例,给出图 3-2(b)的原始链码、起点归一化码等。

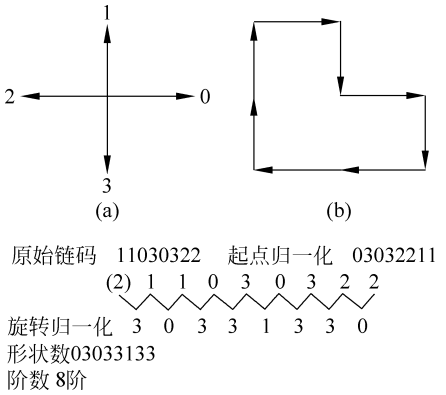


图 3-2 链码示例

3.2 基于曲率的形状分析

3.2.1 曲率与几何特征

(1) 斜率:轮廓点的(切线)指向。

(2) 曲率:斜率的改变率。曲率大于零时,曲线凹向朝着法线正向;曲率小于零时,曲线凹向朝着法线负向。

(3) 角点:曲率的局部极值点。

表 3-1 列出了几种曲率表征的几何特征。

表 3-1 曲率与几何特征

曲　率	几 何 特 征	曲　率	几 何 特 征
连续零曲率	线段	局部最大曲率正值	凸角点
连续非零曲率	弧线段	局部最大曲率负值	凹角点
局部最大曲率绝对值	(一般) 角点	曲率过零点	拐点

3.2.2 曲面曲率

主曲率：在曲面上的某点可找出一个具有最大曲率的方向 k_1，再找出一个具有最小曲率的方向 k_2，它们是互相正交的。

高斯曲率：$G=k_1k_2$

平均曲率：$H=(k_1+k_2)/2$

其中，k_1 和 k_2 分别是该点的主曲率。表 3-2 列出了不同曲面曲率 G、H 表征的不同形状。

高斯曲率的几何意义是：反映曲面在某一点处的“弯曲程度”。如果高斯曲率是正的，那么曲面在该点是“椭球形”弯曲；如果高斯曲率是负的，那么曲面在该点是“双曲面形”弯曲；如果高斯曲率为零，那么曲面在该点是“抛物面形”弯曲。

平均曲率的几何意义可以理解为：如果将曲面在该点处的切平面稍微扭曲，使其与曲面相切，那么这个扭曲量的平均值就是平均曲率。

表 3-2 曲面曲率与形状

曲　率	$H>0$	$H=0$	$H<0$
$G>0$	鞍脊	反向鞍脊	鞍谷
$G=0$	脊面	平　面	谷面
$G<0$	顶面	—	凹坑

3.3 图像仿射变换

3.3.1 仿射变换概念

仿射变换(affine transformation)是指图像可以通过仿射变换矩阵运算后实现平移、旋转、缩放、倾斜、翻转多种操作。

仿射变换可以看作两种简单变换的叠加：线性变换和平移变换。该变换能够保持图像的平直性和平行性。

(1) 平直性：指图像经过仿射变换后，直线仍然是直线。

(2) 平行性：若两条线变换前平行，则变换后仍然平行。

(3) 共线性：若几个点变换前在一条线上，则仿射变换后仍然在一条线上。

(4) 共线比例不变性：变换前一条线上两条线段的比例，变换后保持不变。

3.3.2 仿射变换公式

对于原始图像中的一点 $P(x,y)$，经过仿射变换后得到在目标图像中的坐标 $P'(x'$，

y'),可以通过以下公式计算:

$$\begin{cases} x' = w_{00}x + w_{01}y + t_x \\ y' = w_{10}x + w_{11}y + t_y \end{cases} \tag{3-1}$$

写成矩阵形式:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & t_x \\ w_{10} & w_{11} & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3-2}$$

仿射变换矩阵 $\boldsymbol{M}$ 的维度是 2×3,由矩阵 $\boldsymbol{w}$ 和 $\boldsymbol{t}$ 构成。

$$\boldsymbol{M} = \begin{bmatrix} w_{00} & w_{01} & t_x \\ w_{10} & w_{11} & t_y \end{bmatrix} \tag{3-3}$$

以上公式中:

(x,y)是原始坐标系中点的坐标。

(x',y')是仿射变换后点的新坐标。

w_{00}、w_{01}、w_{10} 和 w_{11} 是控制旋转、缩放、翻转和倾斜的矩阵元素。

t_x 和 t_y 是平移的量。

如图 3-3、图 3-4 所示,使用不同矩阵元素的矩阵 $\boldsymbol{M}$,就会获得不同的仿射变换效果。

不变 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$　平移 $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$　缩放 $\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \end{bmatrix}$　原点旋转 $\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \end{bmatrix}$

沿原点翻转 $\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$　沿X轴翻转 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$　沿Y轴翻转 $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

图 3-3　不同效果的矩阵 $\boldsymbol{M}$

沿X轴倾斜 $\begin{bmatrix} 1 & \tan\varphi & 0 \\ 0 & 1 & 0 \end{bmatrix}$　沿Y轴倾斜 $\begin{bmatrix} 1 & 0 & 0 \\ \tan\varphi & 1 & 0 \end{bmatrix}$

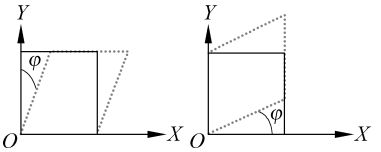


图 3-4　倾斜效果的矩阵 $\boldsymbol{M}$

在 OpenCV 中,仿射变换也可以先使用 cv2.getAffineTransform()函数计算仿射变换矩阵,然后使用 cv2.warpAffine()函数将变换矩阵应用于图像。

3.3.3　图像平移及例程

平移是一种简单空间变换。其表达式为

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \\ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{cases} \tag{3-4}$$

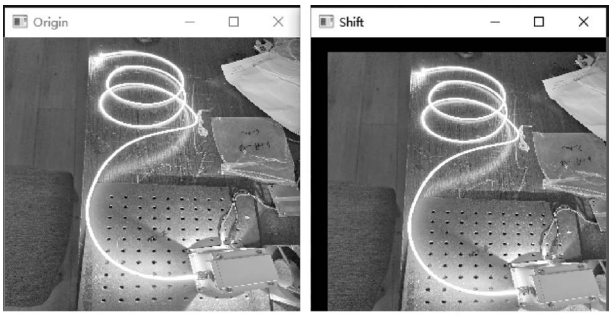
如果向右平移 150 个像素,向下平移 200 个像素,那么变换矩阵 $\boldsymbol{M}$ 可以为

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 150 \\ 0 & 1 & 200 \end{bmatrix}$$

例程 3-1 图像平移

```
import cv2
import numpy as np
fibe = cv2.imread("./File.jpg")                    # 读取原图
height, width = fibe.shape[:2]                     # 获取图像的高度和宽度
x = 150 #向右移动 150 个像素
y = 200 #向下移动 200 个像素
M = np.float32([[1, 0, x], [0, 1, y]])             #转换矩阵 M
Panned_fibe = cv2.warpAffine(fibe, M, (width, height))
cv2.namedWindow('Origin', cv2.WINDOW_NORMAL)
cv2.namedWindow('Shift', cv2.WINDOW_NORMAL)
cv2.imshow("Origin", fibe)
cv2.imshow("Shift", Panned_fibe)
cv2.waitKey()
cv2.destroyAllWindows()
```

图像平移后的效果如图 3-5(b)所示，图像整体向右、向下平移，左上角出现黑边。



(a) (b)

图 3-5 图像平移后的效果

(a) 原始图像；(b) 平移后的图像

3.3.4 图像旋转缩放及例程

1. 利用 OpenCV 自带函数旋转

在使用函数 cv2.warpAffine()对图像进行旋转时，可以通过函数 cv2.getRotationMatrix2D()获取转换矩阵 $\boldsymbol{M}$。

该函数的语法格式为

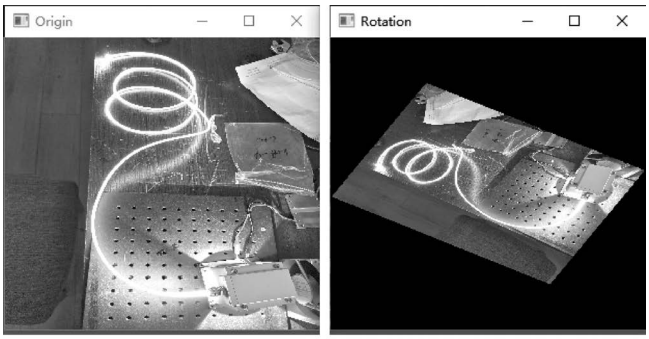
```
retval = cv2.getRotationMatrix2D (center,angle,scale)
```

center：旋转中心点。

angle：旋转角度，正数表示逆时针旋转，负数表示顺时针旋转。

scale：变换尺度(缩放大小)。

以下例程以图像中心为圆点，逆时针旋转 60°，并将目标图像缩小为原始图像的 3/5。例程中图像旋转后的效果如图 3-6 所示，左图是原始图，右图是旋转后的图。



(a) (b)

图 3-6 图像旋转后的效果

(a) 原始图像；(b) 旋转后的图像

例程 3-2 图像旋转

```
import cv2
fibe = cv2.imread("./File.jpg")                          # 读取原图
height, width = fibe.shape[:2]                           # 获取图像的高度和宽度
# 以图像中心为圆点,逆时针旋转 60°,并将目标图像缩小为原始图像的 3/5
M = cv2.getRotationMatrix2D((width/2, height/2), 60, 0.6) #生成转换矩阵 M
rotate_fibe = cv2.warpAffine(fibe, M, (width, height))
cv2.namedWindow('Origin', cv2.WINDOW_NORMAL)
cv2.namedWindow('Rotation', cv2.WINDOW_NORMAL)
cv2.imshow("Origin", fibe)
cv2.imshow("Rotation", rotate_fibe)
cv2.waitKey()
cv2.destroyAllWindows()
```

2. 使用变换矩阵旋转

如图 3-7 所示，围绕原点进行旋转，公式推导如下：

$$\begin{cases} x = r\cos\phi \\ y = r\sin\phi \\ x' = r\cos(\phi+\theta) \\ x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ x' = x\cos\theta - y\sin\theta \\ y' = r\sin(\phi+\theta) \\ y' = r\sin\phi\cos\theta + r\cos\phi\sin\theta \\ y' = y\cos\theta + x\sin\theta \end{cases} \tag{3-5}$$

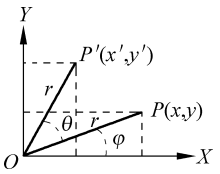


图 3-7 沿原点旋转

由此可推导得到 $\boldsymbol{M}$ 矩阵：

$$\begin{cases} \alpha = \cos\theta \\ \beta = \sin\theta \\ \boldsymbol{M} = \begin{bmatrix} \alpha & -\beta & 0 \\ \beta & \alpha & 0 \end{bmatrix} \end{cases} \tag{3-6}$$

进行公式推导的时候，参照的原点在左下角，而在 OpenCV 中，图像的原点在图像的左上角，所以实际使用需对 θ 取反。

围绕任意点旋转：可以先把当前图像的旋转中心点平移到原点处，绕原点旋转后再平移回去，使用如下变换矩阵 $\boldsymbol{M}$ 实现：

$$\begin{cases}\alpha = \text{scale}\cos(\text{angle}) \\ \beta = \text{scale}\sin(\text{angle}) \\ \boldsymbol{M} = \begin{bmatrix} \alpha & \beta & (1-\alpha)\text{center}.x - \beta\text{center}.y \\ -\beta & \alpha & \beta\text{center}.x - (1-\alpha)\text{center}.y \end{bmatrix}\end{cases} \tag{3-7}$$

其中，center 为原始图像中心点坐标，scale 为放大比例，angle 为旋转角度。

例程 3-3 使用 $\boldsymbol{M}$ 矩阵实现图像旋转

```
import cv2
import numpy as np
from math import cos, sin, radians
img = cv2.imread('file.jpg')
height, width, channel = img.shape
theta = 45
def getRotationMatrix2D(angle, tx=0, ty=0):
    # 将角度值转换为弧度值
    # 因为图像的左上角是原点,需要×(-1)
    angle = radians(-1 * angle)
    M = np.float32([
        [cos(angle), -sin(angle), (1 - cos(angle)) * tx + sin(angle) * ty],
        [sin(angle), cos(angle), -sin(angle) * tx + (1 - cos(angle)) * ty]])
    return M
# 求得图像中心点,作为旋转的轴心
cx = int(width / 2)
cy = int(height / 2)
# 进行 2D 仿射变换
# 围绕原点逆时针旋转 30°
M = getRotationMatrix2D (30, tx=cx, ty=cy)
rotated_30 = cv2.warpAffine(img, M, (width, height))
# 围绕原点逆时针旋转 45°
M = getRotationMatrix2D (45, tx=cx, ty=cy)
rotated_45 = cv2.warpAffine(img, M, (width, height))
# 围绕原点逆时针旋转 60°
M = getRotationMatrix2D (60, tx=cx, ty=cy)
rotated_60 = cv2.warpAffine(img, M, (width, height))
cv2.namedWindow('Origin', cv2.WINDOW_NORMAL)
cv2.namedWindow('Rotated 30 Degree', cv2.WINDOW_NORMAL)
cv2.imshow("Origin", img)
cv2.imshow("Rotated 30 Degree", rotated_30[:, :, ::-1])
cv2.namedWindow('Rotated 45 Degree', cv2.WINDOW_NORMAL)
cv2.imshow("Rotated 45 Degree", rotated_45[:, :, ::-1])
cv2.namedWindow('Rotated 60 Degree', cv2.WINDOW_NORMAL)
cv2.imshow("Rotated 60 Degree", rotated_60[:, :, ::-1])
cv2.waitKey()
cv2.destroyAllWindows()
```

图 3-8 展示了左上角的原始图像依次旋转 30°、45°及 60°后的效果。

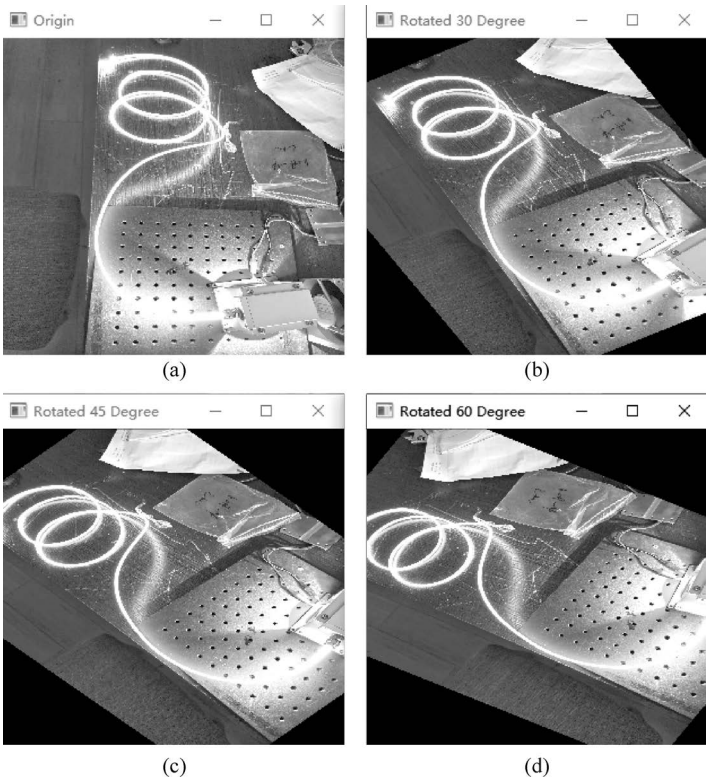


图 3-8　使用 $\boldsymbol{M}$ 矩阵旋转

(a) 原始图像；(b) 旋转 30°；(c) 旋转 45°；(d) 旋转 60°

3.3.5　图像翻转及例程

设 width 代表图像的宽度，height 代表图像的高度。

水平翻转的 $\boldsymbol{M}$ 变换矩阵如下：

$$\boldsymbol{M}=\begin{bmatrix}-1 & 0 & \text{width}\\ 0 & 1 & 0\end{bmatrix} \tag{3-8}$$

垂直翻转的 $\boldsymbol{M}$ 变换矩阵如下：

$$\boldsymbol{M}=\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & \text{height}\end{bmatrix} \tag{3-9}$$

同时进行水平翻转与垂直翻转 $\boldsymbol{M}$ 变换矩阵如下：

$$\boldsymbol{M}=\begin{bmatrix}-1 & 0 & \text{width}\\ 0 & -1 & \text{height}\end{bmatrix} \tag{3-10}$$

以下例程使用 $\boldsymbol{M}$ 矩阵实现图像垂直、水平翻转。

例程 3-4　使用 $\boldsymbol{M}$ 矩阵实现图像翻转

```
import cv2
import numpy as np
img = cv2.imread('file.jpg')
height, width, channel = img.shape
```

```
# 水平翻转
M1 = np.float32([[-1, 0, width], [0, 1, 0]])
flip_h = cv2.warpAffine(img, M1, (width, height))
# 垂直翻转
M2 = np.float32([[1, 0, 0], [0, -1, height]])
flip_v = cv2.warpAffine(img, M2, (width, height))
# 水平垂直同时翻转
M3 = np.float32([[-1, 0, width], [0, -1, height]])
flip_hv = cv2.warpAffine(img, M3, (width, height))
def bgr2rbg(img):
    return img[:, :, ::-1]
cv2.namedWindow('Origin', cv2.WINDOW_NORMAL)
cv2.imshow("Origin", bgr2rbg(img))
cv2.namedWindow('Horizontally', cv2.WINDOW_NORMAL)
cv2.imshow("Horizontally", bgr2rbg(flip_h))
cv2.namedWindow('Vertically', cv2.WINDOW_NORMAL)
cv2.imshow("Vertically", bgr2rbg(flip_v))
cv2.namedWindow('Horizontally & Vertically', cv2.WINDOW_NORMAL)
cv2.imshow("Horizontally & Vertically", bgr2rbg(flip_hv))
cv2.waitKey()
cv2.destroyAllWindows()
```

图 3-9 展示了例程使用 $\boldsymbol{M}$ 矩阵对左上角的原始图像分别进行水平翻转，垂直翻转，水平、垂直同时翻转后的效果。

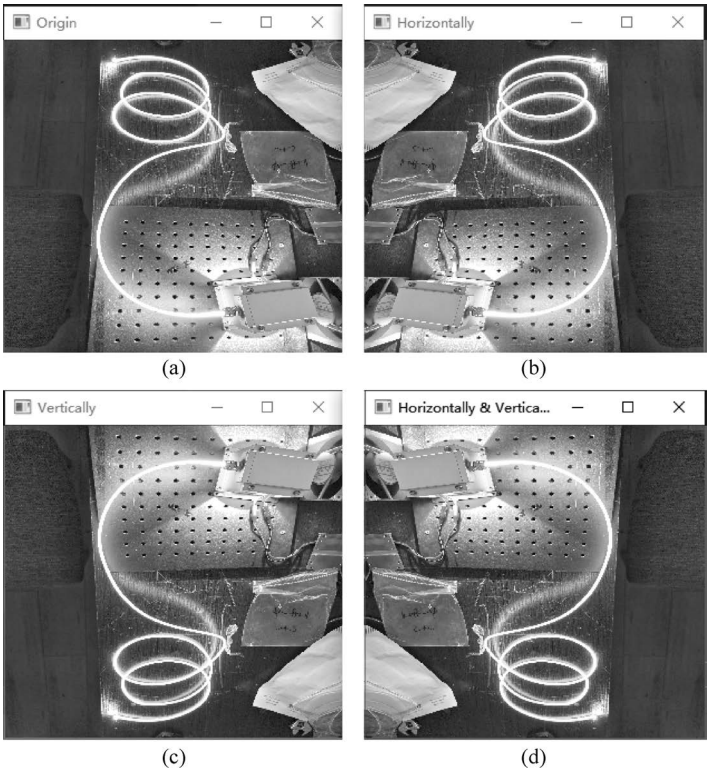


图 3-9 使用 $\boldsymbol{M}$ 矩阵实现图像翻转

(a) 原始图像；(b) 水平翻转；(c) 垂直翻转；(d) 水平、垂直同时翻转

3.3.6 函数直接生成转换矩阵及例程

OpenCV 提供了函数 cv2. getAffineTransform()以生成仿射函数 cv2. warpAffine()使用的转换矩阵 $\boldsymbol{M}$。

该函数的语法格式为

```
retval = cv2.getAffineTransform(src,dst)
```

src：输入图像中的三个点坐标。

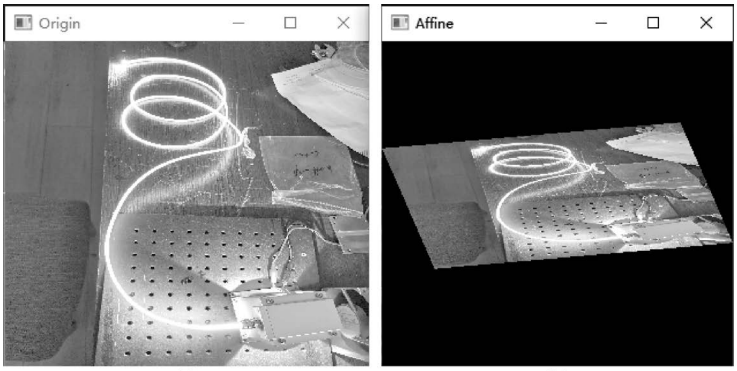
dst：输出图像中的三个点坐标。

在该函数中，参数 src 和 dst 是包含三个二维数组(x,y)点的数组。上述参数通过函数 cv2. getAffineTransform()定义了两个平行四边形。src 和 dst 中的三个点分别对应平行四边形左上角、右上角、左下角的三个点。函数 cv2. warpAffine()以函数 cv2. getAffineTransform()获取的转换矩阵 $\boldsymbol{M}$ 为参数，将 src 中的点仿射到 dst 中。函数 cv2. getAffineTransform()完成指定点的映射后，按照指定点的关系计算确定所有其他点的映射关系。

例程 3-5 函数生成转换矩阵

```
import cv2
import numpy as np
fibe = cv2.imread("./File.jpg")                    # 读取原图
rows, cols, ch = fibe.shape                        # 获取图像的行数、列数和色彩通道数
p1 = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])
p2 = np.float32([[0, rows * 0.33], [cols * 0.85, rows * 0.25], [cols * 0.15, rows * 0.7]])
M = cv2.getAffineTransform(p1, p2)                 # 转换矩阵 M
dst = cv2.warpAffine(fibe, M, (cols, rows))# 按照指定点的关系计算确定所有其他点的映射关系
cv2.namedWindow('Origin', cv2.WINDOW_NORMAL)
cv2.namedWindow('Affine', cv2.WINDOW_NORMAL)
cv2.imshow("Origin", fibe)
cv2.imshow("Affine", dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

图 3-10 展示了例程使用函数生成 $\boldsymbol{M}$ 矩阵，对原始图像进行仿射变换后的效果。



(a) (b)

图 3-10 函数生成转换矩阵对图像进行仿射变换后的效果

(a) 原始图像；(b) 仿射变换后的图像