《计算机视觉-openCV应用技术》

第三章 使用OpenCV 3处理图像

李策

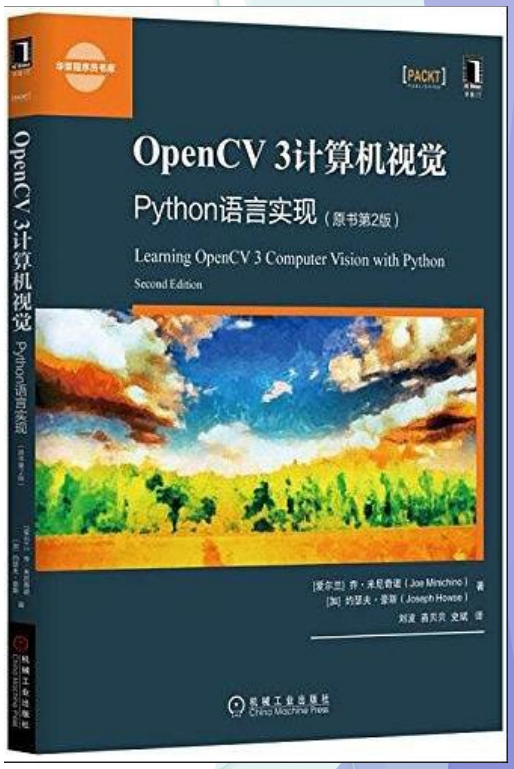
中国矿业大学（北京）计算机科学与技术系

E-mail: celi@cumtb.edu.cn

提纲

第三章 使用OpenCV 3处理图像

- 3.1 不同色彩空间的转换
- 3.2 傅里叶变换
- 3.3 创建模块
- 3.4 边缘检测
- 3.5 用定制内核做卷积
- 3.6 修改应用
- 3.7 Canny边缘检测
- 3.8 轮廓检测
- 3.9 边界框、最小矩形区域和最小闭圆的轮廓
- 3.10 凸轮廓与Douglas-peucker算法
- 3.11 直线和圆检测
- 3.12 检测其他形状



3.1 不同色彩空间的转换

计算机视觉中三种常用的色彩空间：灰度、BGR、HSV。

- 灰度色彩空间，是通过去除彩色信息来将其转换成灰阶
- BGR色彩空间，即蓝-绿-红色彩空间，每一个像素点都由一个三元数组来表示，分别代表蓝、绿、红三种颜色。
- HSV色彩空间，H(Hue)是色调、S（Saturation）是饱和度、V（Value）表示黑暗的程度(或光谱另一端的明亮程度)。

3.1 不同色彩空间的转换

关于BGR的说明:

当第一次处理BGR色彩空间的时候，可以不要其中的一个色彩分量，

比如像素值[0,255,255]表示黄色

颜色模型具有可加性并且处理的是光照

运行在计算机上的软件所使用的色彩模型是加色模型。

3.2 傅里叶变换

Joseph Fourier（约瑟夫.傅里叶）认为一切都可以用波形来描述。

看到的波形都是由其它波形叠加结果。

通过傅里叶变换来介绍图像的幅度谱:

- 把一幅图像中最明亮的像素放到图中央，
- 然后逐渐变暗，在边缘的像素最暗。

可以发现图像中有多少亮的像素和暗的像素，以及其分布的百分比。

引入两个概念：

- 高通滤波器
- 低通滤波器

3.2.1 高通滤波器

高通滤波器(High-pass filter,HPF)，是检测图像的某个区域，根据像素与周围像素的亮度差值来提升该像素亮度的滤波器。

通常，在计算完中央像素和周围邻近像素的亮度差值之和后，如果亮度变化很大，中央像素的亮度会增加（反之则不会）。也就是说，如果一个像素比它周围的像素更突出，就会提升它的亮度。

高通滤波器的特例：高频提升滤波器(High Boost Filter,HBF)在边缘检测上尤其有效。

高通和低通滤波器都有半径(radius)属性，决定多大面积的邻近像素参与滤波运算。

3.2.1 高通滤波器

核是指一组权重的集合，它会应用在源图像的一个区域，并由此产生目标图像的一个像素。比如，大小为7的核意味着每49(7*7)个源图像的像素会产生目标图像的一个像素。

高通滤波器的例子如下，注意其中所有值总和为0。

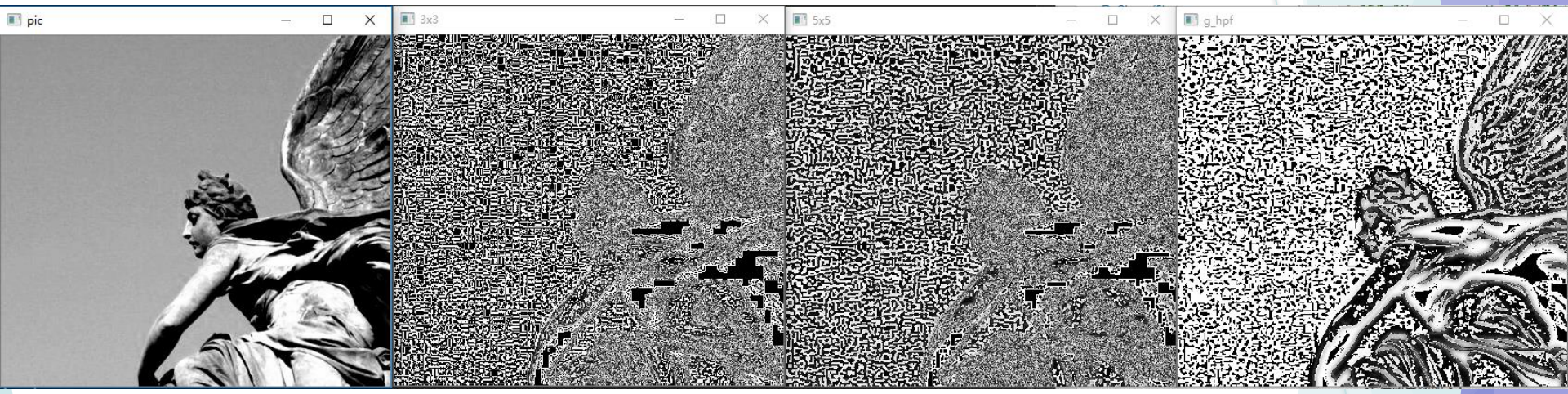
```
kernal_3x3 = np.array([[-1, -1, -1],
                       [-1, 8, -1],
                       [-1, -1, -1]])
kernal_5x5 = np.array([[-1, -1, -1, -1, -1],
                       [-1, 1, 2, 1, -1],
                       [-1, 2, 4, 2, -1],
                       [-1, 1, 2, 1, -1],
                       [-1, -1, -1, -1,-1]])
```

3.2.1 高通滤波器

```
img = cv2.imread("../images/statue_small.jpg", 0)

k3 = ndimage.convolve(img, kernel_3x3)
k5 = ndimage.convolve(img, kernel_5x5)

blurred = cv2.GaussianBlur(img, (17,17), 0)
g_hpf = img - blurred

cv2.imshow("3x3", k3)
cv2.imshow("5x5", k5)
cv2.imshow("g_hpf", g_hpf)
cv2.waitKey()
cv2.destroyAllWindows()
```

3.2.1 高通滤波器

运行效果图如下：



3.2.2 低通滤波器

高通滤波器(High Pass Filter,HPF)是根据像素与周围像素的亮度差值，来提升该像素的亮度。

低通滤波器(Low Pass Filter,LPF)是在像素与周围像素的亮度差值小于一个特定值时，平滑该像素的亮度。

低通滤波器主要用于去噪和模糊化。

例如，高斯模糊是最常用的模糊滤波器(平滑滤波器)之一，它是一个削弱高频信号强度的低通滤波器

3.3 创建模块

在cameo.py的同一目录下创建一个filters.py文件，导入如下模块，添加滤波函数和类。

```
import cv2
import numpy
import utils
```

在cameo.py的同一目录下创建一个utils.py的文件，导入如下模块，添加通用的数学函数。

```
import cv2
import numpy
import scipy.interpolate
```

3.4 边缘检测

- OpenCV提供的边缘检测滤波函数：

  Laplacian()、Sobel()、Scharr()

以上滤波函数会将非边缘区域转为黑色，将边缘区域转为白色或其他饱和的颜色，但容易将噪声错误地识别为边缘。解决办法是在找到边缘之前先对图像做模糊处理。

- OpenCV提供的模糊滤波函数：

  blur() 简单算术平均

  medianBlur() 去除彩色图像噪声

  GaussianBlur()

边缘检测函数和模糊滤波函数有很多参数，但总会有一个ksize参数，它是一个奇数，表示滤波核的宽和高(以像素为单位)。

3.4 边缘检测

```
strokeEdges (见课本40页)
if blurKsize >= 3:
        blurredSrc = cv2.medianBlur(src,blurKsize)
        graySrc =  cv2.cvtColor(blurredSrc, cv2.COLOR_BGR2GRAY)
    else:
        graySrc = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    cv2.Laplacian(graySrc,cv2.CV_8U,graySrc,ksize=edgeKsize)
    normalizedInverseAlpha = (1.0/255)*(255-graySrc)
    channels =  cv2.split(src)
```

3.5 用定制内核做卷积

核是一组权重，它决定如何通过临近像素点来计算新的像素点。

核也称为卷积矩阵，它对一个区域的像素做调和(mix up)或卷积运算，通常基于核的滤波器被称为卷积滤波器。

OpenCV提供了一个非常通用的filter2D(),它运用由用户指定的任意核或卷积矩阵。

什么是卷积矩阵？

卷积矩阵是一个二维数组，有奇数行、奇数列，中心的元素对应于感兴趣的像素，其它的元素对应于这个像素周围的邻近像素，每个元素都有一个整数或浮点数的值，这些值就是应用在像素上的权重。

```
kernel = numpy.array([[-1, -1 , -1],
                      [-1,  9,  -1],
                      [-1, -1, -1]])
```

3.5 用定制内核做卷积

在源图像和目标图像上分别使用卷积矩阵：

cv2.filter2D(src, -1, kernel, dst). 第二个参数指定了目标图像每个通道的位深度（比如，位深度cv2.CV_8U表示每个通道为8位）

对于模糊滤波器，为了达到模糊效果，通常权重和应该为1，而且邻近像素的权重全为正。

```
# 一般的滤波器
class VConvolutionFilter(object):

    def __init__(self, kernel):
        self._kernel = kernel

    def apply(self, src, dst):
cv2.filter2D(src, -1, self._kernel, dst)
```

```
# 特定的锐化滤波器
Class
SharpenFilter(VConvolutionFilter):

    def __init__(self):
        kernel = numpy.array([[-1, -1, -1],
                              [-1, 9, -1],
                              [-1, -1, -1]])
        VConvolutionFilter.__init__(self,
kernel)
```

3.5 用定制内核做卷积

```
# 边缘检测滤波器
class FindEdgesFilter(VConvolutionFilter):
    def __init__(self):
        kernel = numpy.array([[-1, -1, -1],
                              [-1, 8, -1],
                              [-1, -1, -1]])
        VConvolutionFilter.__init__(self, kernel)
```

3.5 用定制内核做卷积

```
# 邻近平均滤波器
class BlurFilter(VConvolutionFilter):
    def __init__(self):
        kernel = numpy.array([[0.04, 0.04, 0.04, 0.04, 0.04],
                              [0.04, 0.04, 0.04, 0.04, 0.04],
                              [0.04, 0.04, 0.04, 0.04, 0.04],
                              [0.04, 0.04, 0.04, 0.04, 0.04],
                              [0.04, 0.04, 0.04, 0.04, 0.04]])
        VConvolutionFilter.__init__(self, kernel)
```

3.7 Canny边缘检测

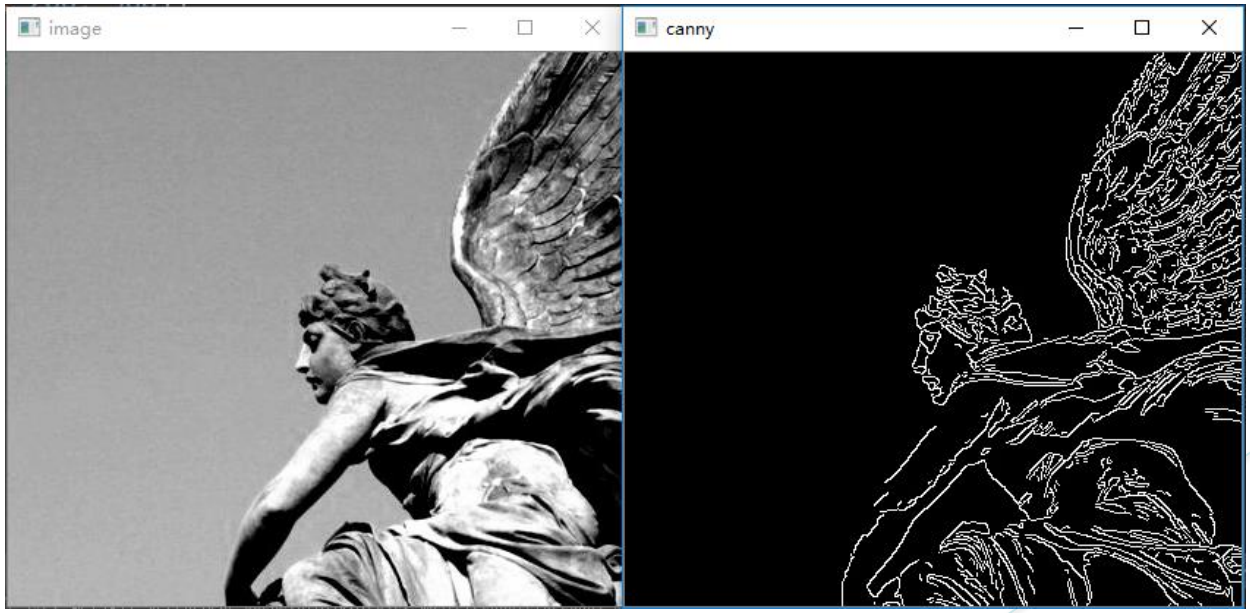
OpenCV提供了Canny边缘检测函数来识别边缘。

它有5个步骤：

- 高斯滤波器对图像进行去噪
- 计算梯度
- 在边缘上使用非最大抑制(NMS)
- 在检测到的边缘上使用双阈值去除假阳性(false positive)
- 分析出所有的边缘及其之间的连接，以保留真正的边缘并消除不明显的边缘

3.7 Canny边缘检测

```
img = cv2.imread("../images/statue_small.jpg", 0)
cv2.imwrite("canny.jpg", cv2.Canny(img, 200, 300))
cv2.imshow("canny", cv2.imread("canny.jpg"))
cv2.waitKey()
cv2.destroyAllWindows()
```



3.8 轮廓检测

在计算机视觉中，轮廓检测是另一个比较重要的任务：

- 检测图像或视频帧中物体的轮廓
- 计算多边形边界、形状逼近
- 计算感兴趣区域

NumPy中的矩形区域可以使用数组切片(slice)来定义

在介绍物体检测（包括人脸）和物体跟踪的概念时会大量使用这种技术。

我们先通过下面的例子熟悉API

3.8 轮廓检测

```
img = np.zeros((200, 200), dtype = np.uint8)
img[50:150, 50:150] = 255

ret, thresh = cv2.threshold(img, 127, 255, 0)

contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
img = cv2.drawContours(color, contours, -1, (0, 0, 255), 5)
cv2.imshow("contours",color)
cv2.waitKey()
cv2.destroyAllWindows()
```

3.8 轮廓检测

findContours()函数:

- 有三个参数

a) 输入图像
b) 层次类型
c) 轮廓逼近方法

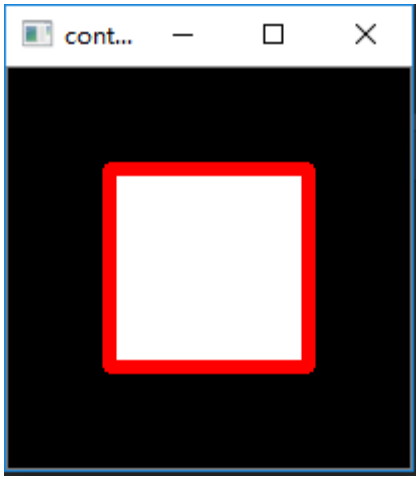
- 由函数返回的层次树相当重要：cv2.RETR_TREE参数会得到图像中轮廓的整体层次结构，以此来建立轮廓之间的“关系”。如果只想得到最外面的轮廓，可使用cv2.RETR_EXTERNAL。

- 这对消除包含在其他轮廓中的轮廓很有用（如在大多数情形下，不需要检测一个目标包含在另一个与之相同的目标里面）

- 有三个返回值：修改后的图像、图像的轮廓以及它们的层次。使用轮廓来画出图像的彩色版本（即把轮廓画成绿色），并显示出来。

3.8 轮廓检测

运行结果：



3.9 边界框、最小矩形区域和最小闭圆的轮廓

找到一个正方形轮廓很简单，要找到不规则的、歪斜的以及旋转的形状可用 OpenCV的cv2.findContours函数，它能得到最好的结果，下面来看一幅图像：

3.9 边界框、最小矩形区域和最小闭圆的轮廓

将cv2.findContours函数与少量的OpenCV的功能相结合就能非常容易地实现这些功能：

在导入模块后，加载图像，在灰度图像上执行计算轮廓的操作：

1. 计算出一个简单的边界框：

   x, y, w, h = cv2.boundingRect(c)

2. 将轮廓信息转换成(x, y)坐标，并加上矩形的高度和宽度， 画出矩形

   cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 2)

3. 计算包围目标的最小矩形区域：

   rect = cv2.minAreaRect(c)

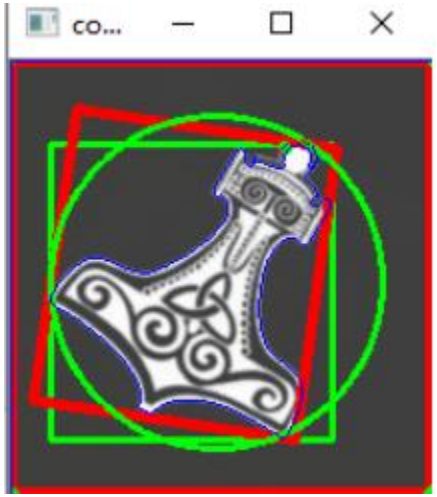
   box = cv2.boxPoints(rect)

   box =np.int0(box)

3.9 边界框、最小矩形区域和最小闭圆的轮廓

4. OpenCV没有函数能直接从轮廓信息中计算出最小矩形顶点的坐标。所以需要计算出最小矩形区域，然后计算这个矩形的顶点。由于计算出来的顶点坐标是浮点型，但是所得像素的坐标值是整数（不能获取像素的一部分）所以需要做一个转换

```
# draw contours
cv2.drawContours(img, [box], 0, (0, 0, 255), 3)  # 画出该矩形
```



3.10 凸轮廓与Douglas-peucker算法

大多数处理轮廓的时候，物体的形状（包括凸形状）都是变化多样的。凸形状内部的任意两点之间的连线都在该形状里面。

cv2.approxPloyDP函数: 它用来计算近似的多边形框。

该函数有三个参数：

- 第一个参数为“轮廓”；
- 第二个参数为“ε值”，它表示源轮廓与近似多边形的最大差值（这个值越小，近似多边形与源轮廓越接近）；
- 第三个参数为“布尔标记”，它表示这个多边形是否闭合。

ε值对获取有用的轮廓非常重要，是为所得到的近似多边形周长与源轮廓周长之间的最大差值，这个差值越小，近似多边形与源轮廓就越相似。

3.10 凸轮廓与Douglas-peucker算法

可通过OpenCV的cv2.arcLength函数来得到轮廓的周长信息：

epsilon = 0.01 * cv2.arcLength(cnt, True)

approx = cv2.approxPolyDP(cnt, epsilon, True)

可通过OpenCV来有效地计算一个近似多边形，多边形周长与源轮廓周长之比就为ε。

为了计算凸形状，需要用OpenCV的cv2.convexHull函数来获取处理过的轮廓信息，代码为：

hull = cv2.convexHull(cnt)

3.11 直线和圆检测

检测边缘和轮廓是构成其他复杂操作的基础。

直线和形状检测与边缘和轮廓检测有密切的关系。

Hough变换是直线和形状检测背后的理论基础。

下面介绍OpenCV中Hough变换中的API函数：

3.11.1 直线检测

直线检测可通过HoughLines和HoughLinesP函数来完成，它们仅有的差别是：第一个函数使用标准的Hough变换，第二个函数使用概率Hough变换。

HoughLinesP函数之所以称为概率版本的Hough变换，是因为它只通过分析点的子集并估计这些点都属于一条直线的概率，这是标准Hough变换的优化版本，该函数的计算代价会少一些，执行会变得更快。

3.11.1 直线检测

HoughLines函数会接收由Canny边缘检测滤波器处理过的单通道二值图像

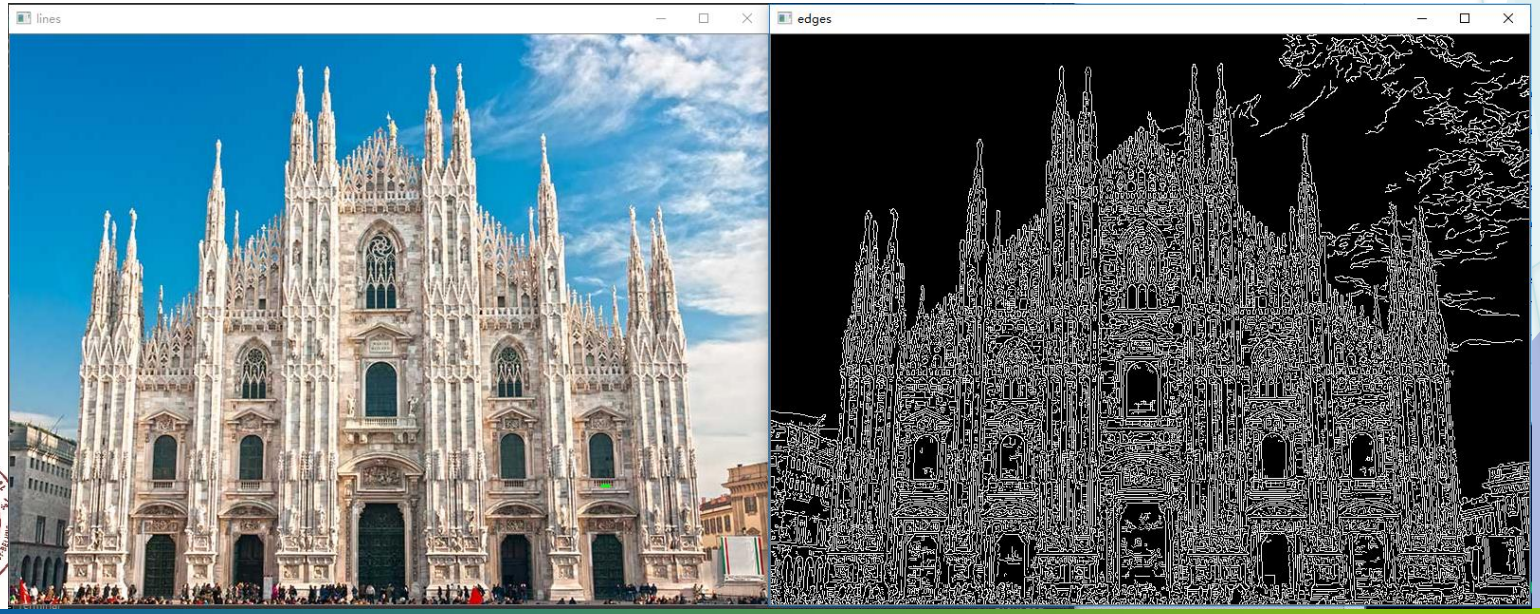
HoughLinesP的参数：

需要处理的图像；

线段的几何表示rho和theta，一般分别取1和np.pi/180；

阈值

minLineLength和maxLineGap

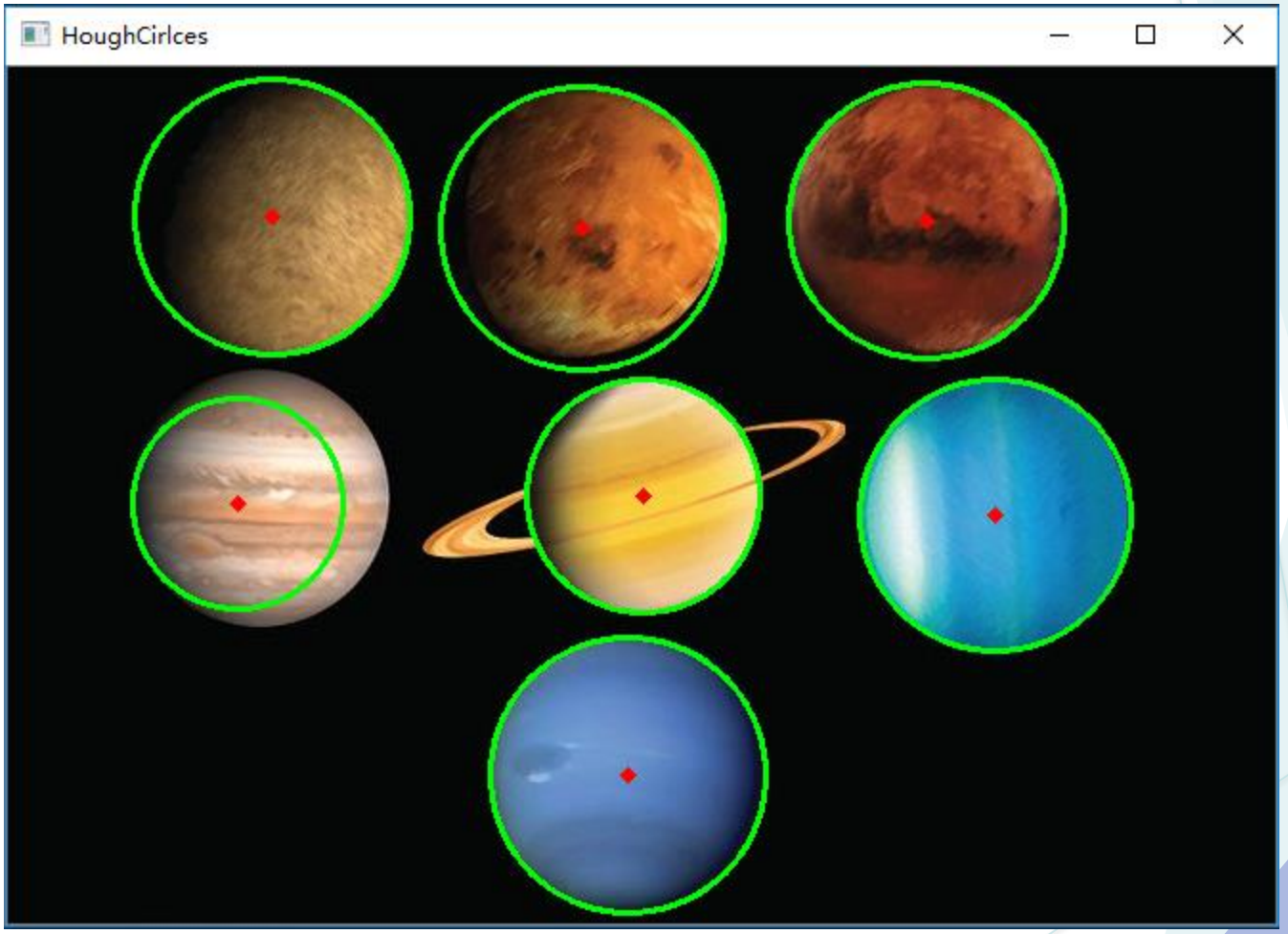


3.11.2 圆检测

OpenCV的HoughCircles函数可用来检测圆，它与使用HoughLines函数类似。像用来决定删除或保留直线的两个参数minLineLength和maxLineGap一样HoughCircles有一个圆心间的最小距离和圆的最小及最大半径。

```
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(planets, (i[0], i[1]), i[2],(0, 255, 0),2)
    # draw the center of the circle
    cv2.circle(planets, (i[0], i[1]), 2, (0, 0,255), 3)
```

3.11.2 圆检测



3.12 检测其他形状

Hough变换能检测的形状仅限于圆，前面提到过检测任何形状的方法，特别是用approxPloyDP函数来检测，该函数提供多边形的近似，所以如果你的图像有多边形，再结合cv2.findContours函数和cv2.approxPloyDP函数，就能相当准确地检测出来。

3.13 总结

本章介绍了：

- 色彩空间、傅里叶变换和多种由OpenCV提供的处理图像的滤波器
- 检测边缘、直线、圆和一些普通形状
- 寻找轮廓，并由此得到关于图像中所包含的目标信息

这些概念都是后续章节的基础