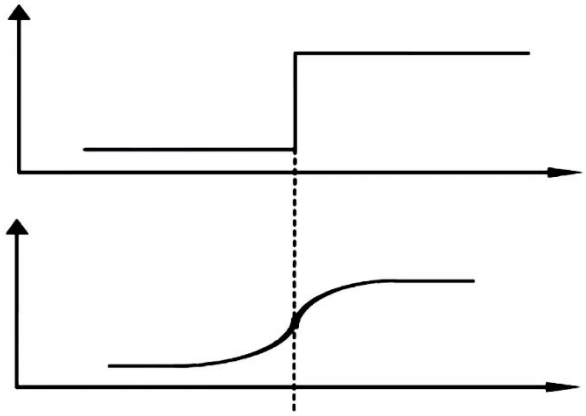
第 10 课 图像处理—边缘检测

1.边缘检测原理

边缘检测（Edge Detection）是图像处理和计算机视觉中的基本问题，其目的在于标识数字图像中亮度变化明显的点。图像属性中的显著变化通常反映了属性的重要事件和变化。边缘的表现形式如下图所示：



图像边缘检测大幅度地减少了数据量，并且剔除了可看作不相关的信息，保留了图像重要的结构属性。边缘检测的方法可大致划分为两类：

1） 基于搜索：通过寻找图像一阶导数中的最大值和最小值来检测边界，通常是将边界定位在梯度最大的方向，代表算法是 Sobel 算子和 Scharr 算子。

2） 基于零穿越：通过寻找图像二阶导数零穿越来寻找边界，通常是 Laplacian 过零点或者非线性差分表示的过零点，代表算法是 Laplacian 算子。

2.Canny边缘检测算法

Canny 边缘检测算法是一种非常流行的边缘检测算法，是 John F.Canny 于 1986 年提出的，被认为是最优的边缘检测算法。它可分为 4 个步骤：去除噪声、计算图像梯度、非极大值抑制和滞后阈值。

◆ **去除噪声**

边缘检测极易受到噪声影响，因此需要使用高斯滤波器去除噪声，具体操作可参考目录**“第 4 章 OpenCV 计算机视觉学习->图像处理进阶篇->第 3 课 图像处理——平滑”**下的文档。

◆ **计算图像梯度幅值与方向**

在数学中，梯度的本意是一个向量，表示函数在某点的方向导数沿着该方向取得最大值，即函数在该点处沿着该方向变化最快，变化率最大。

图像中用梯度表示灰度值的变化程度与方向，而边缘就是指灰度强度变化最强的位置。梯度方向与边缘方向呈垂直关系。

此处采用 Sobel 滤波器计算梯度的大小和方向。Sobel 算子 G_X 是水平方向的一阶导数，用于检测 Y 轴方向的边缘，而 G_Y 则是竖直方向的一阶导数，用于检测 X 轴方向的边缘。

梯度大小的计算公式如下：

$$G = \sqrt{(G_X^2 + G_Y^2)}$$

梯度方向的计算公式如下：

$$\theta = arctan\frac{G_Y}{G_X}$$

◆ **非极大值抑制**

非极大值抑制（Non-Maximum Suppression，NMS），即保留局部最大值，抑制非局部最大值的所有值。简单而言，就是对图像的所有像素点进行检测，如果某点的梯度强度大于其梯度方向的正负方向的像素点，则该点保留；否则，该点被抑制。

Canny 边缘检测算法是沿着梯度方向对幅值进行非极大值抑制的，而非边缘方向。

◆ **滞后阈值**

为了确定真正的边界，需要设定两个阈值“**minVal**”和“**maxVal**”。

当图像某点的灰度梯度大于阈值“**maxVal**”，该点被视为真的边界点；

当某点的灰度梯度小于阈值“**minVal**”，则该点不看作边界点；

当某点的灰度梯度介于两个阈值之间，根据该点是否与真的边界点相连来进行判断，相连则将该点也看作边界点，否则将其抛弃。

3.实验步骤

本节例程会**对图像进行边缘检测**。

开始操作前，需要先将目录“**第 4 章 OpenCV 计算机视觉学习->图像处理进阶篇->第 9 课 图像处理——边缘检测->例程源码**”下的例程“**edge_detection.py**”和示例图片“**luna.jpg**”复制到共享文件夹。

关于共享文件夹的配置方法，可参考目录“**第 2 章 Linux 系统简介及使用入门->Linux 基础课程->第 3 课 Linux 系统安装及换源方法**”下的文档。

注意：输入指令时需严格区分大小写，且可以使用“Tab”键补齐关键字。

1） 启动虚拟机，点击系统任务栏的图标  ，并点击图标  ，或使用快捷键“**Ctrl+Alt+T**”，打开命令行终端。

2） 输入指令“**cd /mnt/hgfs/Share/**”，并按下回车，进入共享文件夹。

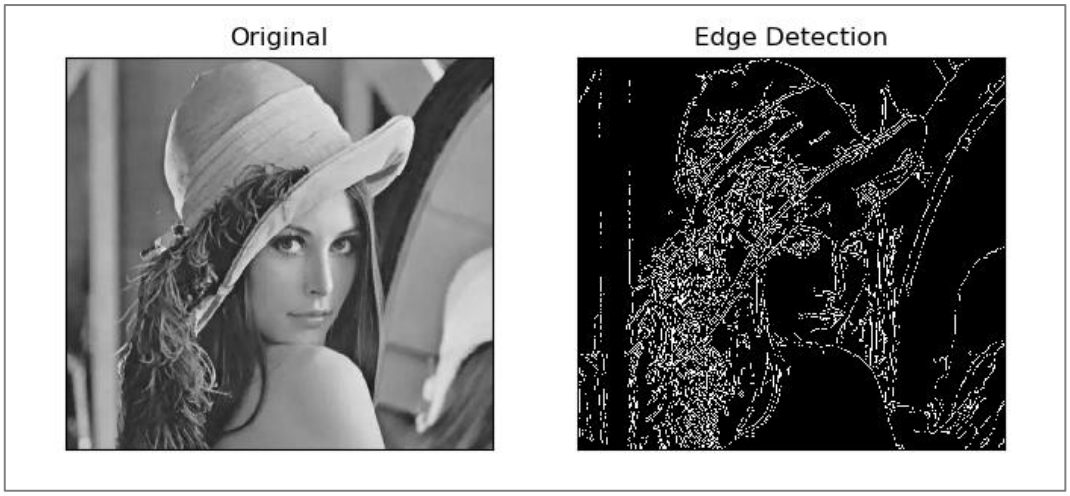
```
hiwonder@ubuntu:~$ cd /mnt/hgfs/Share/
```

3） 输入指令“**python3 edge_detection.py**”，并按下回车，运行例程。

```
hiwonder@ubuntu:/mnt/hgfs/Share$ python3 edge_detection.py
```

4.实验效果

执行程序后，图像处理结果如下图所示：



“**Original**”是原始图像；“**Edge Detection**”是对原图进行边缘检测的结果图像。

5.程序分析

可以在目录“**第 4 章 OpenCV 计算机视觉学习->图像处理进阶篇->第 9 课 图像处理——边缘检测->例程源码**”下查看例程“**edge_detection.py**”。

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # 图像读取
6  img = cv2.imread('luna.jpg')
7
8  # Canny边缘检测
9  lowThreshold = 1
10 max_lowThreshold = 80
11 canny = cv2.Canny(img, lowThreshold, max_lowThreshold)
12
13 # 图像展示
14 plt.figure(figsize=(8, 5), dpi=100)
15 plt.rcParams['axes.unicode_minus'] = False
16 plt.subplot(121), plt.imshow(img, cmap=plt.cm.gray), plt.title("Original")
17 plt.xticks([]), plt.yticks([])
18 plt.subplot(122), plt.imshow(canny, cmap=plt.cm.gray), plt.title("Edge Detection")
19 plt.xticks([]), plt.yticks([])
20 plt.show()
```

5.1 图像处理

◆ **导入模块**

首先，通过 import 语句导入所需模块。

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
```

◆ **读取图像**

通过调用 cv2 模块中的 imread()函数，读取用于边缘检测的图像。

```
6  img = cv2.imread('luna.jpg')
```

函数括号内的参数是图像名称。

◆ **设置阈值**

设置两个阈值，用于确定最终的边界。

```
9  lowThreshold = 1
10 max_lowThreshold = 80
```

◆ **边缘检测**

通过调用 cv2 模块中的 Canny()函数，对指定图像进行边缘检测。

```
11 canny = cv2.Canny(img, lowThreshold, max_lowThreshold)
```

Canny()函数的语法格式如下：

```
Canny( image, threshold1, threshold2)
```

第一个参数“**image**”是输入图像；

第二个参数“**threshold1**”是低阈值“**minVal**”；

第三个参数“**threshold2**”是高阈值“**maxVal**”。

5.2 图像显示

◆ **创建自定义画像**

通过调用 matplotlib.pyplot 模块中的 figure()函数，创建一个自定义画像（Figure），用于显示滤波处理的结果图像。

```
14  plt.figure(figsize=(8, 5), dpi=100)
```

figure()函数的语法格式如下：

figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs)

第一个参数“**num**”是画像的唯一标识符，即画像编号（数字）或名称（字符串）；

第二个参数“**figsize**”是画像的宽和高，单位为英寸；

第三个参数“**dpi**”是画像的分辨率，即每英寸的像素数；

第四个参数“**facecolor**”是画像的背景颜色；

第五个参数“**edgecolor**”是画像的边框颜色；

第六个参数“**frameon**”用于决定是否绘制画像，默认值为“**True**”；

第七个参数“**FigureClass**”是生成画像时选用的自定义 Figure；

第八个参数“**clear**”用于决定当画像存在时是否清除原有画像；

第九个参数“****kwargs**”是画像的其他属性。

◆ **修改 matplotlib 配置**

matplotlib 是 Python 的绘图库，用户可以通过参数字典“**rcParams**”访问与修改 matplotlib 的配置项。

```
15  plt.rcParams['axes.unicode_minus'] = False
```

上图所示代码用于控制正常字符显示。

◆ **设置图像显示参数**

通过调用 matplotlib.pyplot 模块中的 subplot()、imshow()和 title()函数，指定画像（Figure）内的子图位置分布、子图颜色类型和子图标题。

```
16  plt.subplot(121), plt.imshow(img, cmap=plt.cm.gray), plt.title("Original")
```

1） subplot()函数用于设置子图的位置分布，其语法格式如下：

```
subplot(nrows, ncols, index, **kwargs)
```

第一个参数“**nrows**”和第二个参数“**ncols**”分别是 subplot 的行、列数；

第三个参数“**index**”是索引位置，从“**1**”开始累计（左上角为“**1**”），依次向右递增。

当行、列数都小于“**10**”，可以将两个数值缩写为一个整数，例如，代码“**subplot(1, 2, 1)**”和“**subplot(121)**”的含义相同，都表示将画像分为 1 行 2 列，当前子图排在第 1 位，即第 1 行的第 1 列。

2） imshow()函数用于设置子图颜色类型，其语法格式如下：

```
imshow(X, cmap=None)
```

第一个参数“**X**”是图像数据；

第二个参数“**cmap**”是颜色图谱（colormap），默认为 RGB（A）颜色空间。

3） title()函数用于设置子图标题，函数括号内的参数是子图名称，其语法格式如下：

```
title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)
```

第一个参数“**label**”是标题文本，类型为字符串；

第二个参数“**fontdict**”是标题文本的字体属性，类型为字典；

第三个参数“**loc**”是标题位置，可取值“**left**”、“**center**”或“**right**”，默认值为“**center**”；

第四个参数“**pad**”是标题与子图的填充距离（内边距），默认值为“**6.0**”；

第五个参数“**y**”是标题在子图中的垂直距离，单位为子图高度的百分比，默认值为

“**None**”，即自动确定标题位置，避免与其他元素重叠。值为“**1.0**”表示在子图顶端；

第六个参数“****kwargs**”是文本对象关键字属性，用于控制文本的外观属性，如字体、文本颜色等。

◆ **设置坐标轴刻度**

通过调用 matplotlib.pyplot 模块中的 xticks()和 yticks()函数，设置 X、Y 轴的刻度及标签，由于此例程在显示图像时无需使用坐标轴信息，此处将列表设为空，即不显示坐标轴。

```
17  plt.xticks([]), plt.yticks([])
```

xticks()函数的语法格式如下：

```
xticks(ticks=None, labels=None, **kwargs)
```

当参数为空，函数会返回当前 X 轴的刻度及标签；否则，函数用于设置当前 X 轴的刻度及标签。

第一个参数“**ticks**”是 X 轴刻度的位置列表，若列表为空，X 轴刻度将清空；

第二个参数“**labels**”是 X 轴刻度的标签，当参数“**ticks**”不为空，此参数才会传递；

第三个参数“****kwargs**”用于控制刻度标签的外观。

yticks()函数的语法格式和 xticks()函数的相同，区别在于 yticks()函数的作用对象是 Y 轴。

◆ **显示图像**

通过调用 matplotlib.pyplot 模块中的 show()函数，在窗口显示图像。

```
20  plt.show()
```

图像显示部分的完整代码如下：

```
14  plt.figure(figsize=(8, 5), dpi=100)
15  plt.rcParams['axes.unicode_minus'] = False
16  plt.subplot(121), plt.imshow(img, cmap=plt.cm.gray), plt.title("Original")
17  plt.xticks([]), plt.yticks([])
18  plt.subplot(122), plt.imshow(canny, cmap=plt.cm.gray), plt.title("Edge Detection")
19  plt.xticks([]), plt.yticks([])
20  plt.show()
```