

## 问题四十一至五十

### 问题四十一：Canny边缘检测：第一步——边缘强度

问题四十一至问题四十三是边缘检测方法中的一种——Canny边缘检测法的理论介绍。

1. 使用高斯滤波；
2. 在 $x$ 方向和 $y$ 方向上使用Sobel滤波器，在此之上求出边缘的强度和边缘的梯度；
3. 对梯度幅值进行非极大值抑制（Non-maximum suppression）来使边缘变得更细；
4. 使用滞后阈值来对阈值进行处理。

上面就是图像边缘检测的方法了。在这里我们先完成第一步和第二步。按照以下步骤进行处理：



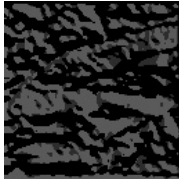
1. 将图像进行灰度化处理；
2. 将图像进行高斯滤波（ $5 \times 5$ ， $s = 1.4$ ）；
3. 在 $x$ 方向和 $y$ 方向上使用Sobel滤波器，在此之上求出边缘梯度 $f_x$ 和 $f_y$ 。边缘梯度可以按照下式求得：

$$\text{edge} = \sqrt{f_x^2 + f_y^2}$$
$$\tan = \arctan\left(\frac{f_y}{f_x}\right)$$

4. 使用下面的公式将梯度方向量化：

$$\text{angle} = \begin{cases} 0 & (\text{if } -0.4142 < \tan \leq 0.4142) \\ 45 & (\text{if } 0.4142 < \tan < 2.4142) \\ 90 & (\text{if } |\tan| \geq 2.4142) \\ 135 & (\text{if } -2.4142 < \tan \leq -0.4142) \end{cases}$$

请使用 `numpy.pad()` 来设置滤波器的 `padding` 吧！

输入(imori.jpg)	输出(梯度幅值)(answers_image/answer_41_1.jpg)	输出(梯度方向)(answers_image/answer_41_2.jpg)
		

答案

- Python >> [answers\\_py/answer\\_41.py](#)
- C++ >> [answers\\_cpp/answer\\_41.cpp](#)

### 问题四十二：Canny边缘检测：第二步——边缘细化

在这里我们完成Canny边缘检测的第三步。



我们从在第四十二问中求出的边缘梯度进行非极大值抑制，来对边缘进行细化。

非极大值抑制是对除去非极大值以外的值的操作的总称（这个术语在其它的任务中也经常出现）。

在这里，我们比较我们我们所关注的地方梯度的法线方向邻接的三个像素点的梯度幅值，如果该点的梯度值不比其它两个像素大，那么这个地方的值设置为0。

也就是说，我们在注意梯度幅值 $edge(x, y)$ 的时候，可以根据下式由梯度方向 $angle(x, y)$ 来变换 $edge(x, y)$ ：

- $angle(x, y) = 0$   
如果在 $edge(x, y)$ 、 $edge(x - 1, y)$ 、 $edge(x + 1, y)$ 中 $edge(x, y)$ 不是最大的，那么 $edge(x, y) = 0$ ；
- $angle(x, y) = 45$   
如果在 $edge(x, y)$ 、 $edge(x - 1, y)$ 、 $edge(x + 1, y)$ 中 $edge(x, y)$ 不是最大的，那么 $edge(x, y) = 0$ ；
- $angle(x, y) = 90$   
如果在 $edge(x, y)$ 、 $edge(x - 1, y)$ 、 $edge(x + 1, y)$ 中 $edge(x, y)$ 不是最大的，那么 $edge(x, y) = 0$ ；
- $angle(x, y) = 135$   
如果在 $edge(x, y)$ 、 $edge(x - 1, y)$ 、 $edge(x + 1, y)$ 中 $edge(x, y)$ 不是最大的，那么 $edge(x, y) = 0$ ；

输入 (imori.jpg)	输出 (answers_image/answer_42.jpg)
	

答案

- Python >> [answers\\_py/answer\\_42.py](#)
- Python >> [answers\\_cpp/answer\\_42.cpp](#)



## 问题四十三：Canny 边缘检测：第三步——滞后阈值

在这里我们进行 Canny 边缘检测的最后一步。

在这里我们将通过设置高阈值（HT：high threshold）和低阈值（LT：low threshold）来将梯度幅值二值化。

1. 如果梯度幅值 $edge(x, y)$ 大于高阈值的话，令 $edge(x, y) = 255$ ；
2. 如果梯度幅值 $edge(x, y)$ 小于低阈值的话，令 $edge(x, y) = 0$ ；
3. 如果梯度幅值 $edge(x, y)$ 介于高阈值和低阈值之间并且周围8邻域内有比高阈值高的像素点存在，令 $edge(x, y) = 255$ ；

在这里，我们使高阈值为100，低阈值为20。顺便说一句，阈值的大小需要边看结果边调整。  
上面的算法就是Canny边缘检测算法了。

输入 (imori.jpg)	输出 (answers_image/answer_43.jpg)
	

- 答案
  - Python >> [answers\\_py/answer\\_43.py](#)
  - C++ >> [answers\\_cpp/answer\\_43.cpp](#)

## 问题四十四：霍夫变换（Hough Transform） / 直线检测——第一步：霍夫变换

第四十四问到第四十六问进行霍夫直线检测算法。

霍夫变换，是将坐标由直角坐标系变换到极坐标系，然后再根据数学表达式检测某些形状（如直线和圆）的方法。当直线上的点变换到极坐标中的时候，会交于一定的 $r$ 、 $t$ 的点。这个点即为要检测的直线的参数。通过对这个参数进行逆变换，我们就可以求出直线方程。

方法如下：

1. 我们用边缘图像来对边缘像素进行霍夫变换。
2. 在霍夫变换后获取值的直方图并选择最大点。
3. 对极大点的 $r$ 和 $t$ 的值进行霍夫逆变换以获得检测到的直线的参数。


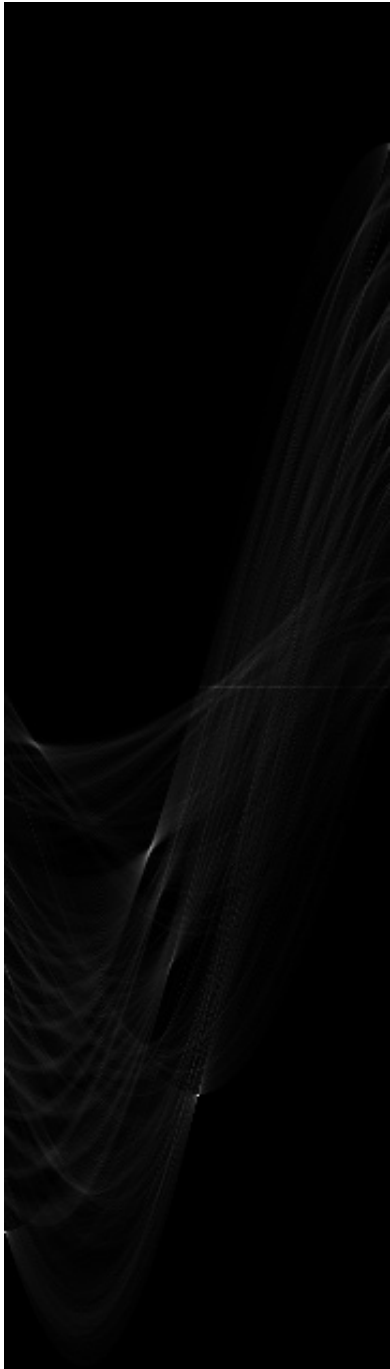
在这里，进行一次霍夫变换之后，可以获得直方图。算法如下：

1. 求出图像的对角线长 $r_{max}$ ；
2. 在边缘点 $(x, y)$ 处， $t$ 取遍 $[0, 179]$ ，根据下式执行霍夫变换：

$$r_{ho} = x \cos(t) + y \sin(t)$$

3. 做一个 $180 \times r_{max}$ 大小的表，将每次按上式计算得到的表格 $(t, r)$ 处的值加1。换句话说，这就是在进行投票。票数会在一定的地方集中。

这一次，使用 torino.jpg 来计算投票之后的表。使用如下参数进行 Canny 边缘检测：高斯滤波器 $(5 \times 5, s = 1.4)$ ， $HT = 100$ ， $LT = 30$ 。

输入 (thorino.jpg)	输出 (answers/answer_44.jpg)
	

答案

- Python >> [answers\\_py/answer\\_44.py](#)
- C++ >> [answers\\_cpp/answer\\_44.cpp](#)

## 问题四十五：霍夫变换（Hough Transform） / 直线检测——第二步：NMS

我们将在这里进行第2步。

在问题44获得的表格中，在某个地方附近集中了很多票。这里，执行提取局部最大值的操作。

这一次，提取出投票前十名的位置，并将其表示出来。

NMS 的算法如下：

- 1. 在该表中，如果遍历到的像素的投票数大于其8近邻的像素值，则它不变。
- 2. 如果遍历到的像素的投票数小于其8近邻的像素值，则设置为0。

输入 (thorino.jpg)	输出 (answers/answer_45.jpg)
	

答案

- Python >> [answers\\_py/answer\\_45.py](#)
- C++ >> [answers\\_cpp/answer\\_45.cpp](#)

问题四十六：霍夫变换（Hough Transform） / 直线检测——第三步：霍夫逆变换


这里是将问题45中得到的极大值进行霍夫逆变换之后画出得到的直线。在这里，已经通过霍夫变换检测出了直线。

算法如下：

1. 极大值点(r,t)通过下式进行逆变换：

$$y = -\frac{\cos(t)}{\sin(t)}x + \frac{r}{\sin(t)}$$
$$x = -\frac{\sin(t)}{\cos(t)}y + \frac{r}{\cos(t)}$$

2. 对于每个局部最大点，使 $y = 0 - H - 1$ ， $x = 0 - W - 1$ ，然后执行1中的逆变换，并在输入图像中绘制检测到的直线。请将线的颜色设置为红色( $R, G, B = (255, 0, 0)$ )。

输入 (thorino.jpg)	输出 (answers/answer_46.jpg)
	

答案

- Python >> [answers\\_py/answer\\_46.py](#)
- C++ >> [answers\\_cpp/answer\\_46.cpp](#)

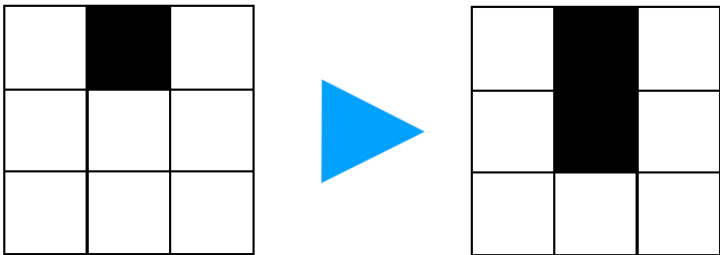
## 问题四十七：膨胀（Dilate）

将 `imori.jpg` 大津二值化之后，进行两次形态学膨胀处理。

在形态学处理的过程中，二值化图像中白色（255）的部分向4-近邻（上下左右）膨胀或收缩一格。

反复进行膨胀和收缩操作，可以消除独立存在的白色像素点（见问题四十九：开操作）；或者连接白色像素点（见问题五十：闭操作）。

形态学处理中的膨胀算法如下。对于待操作的像素 $I(x, y) = 0$ ， $I(x, y - 1)$ ， $I(x - 1, y)$ ， $I(x + 1, y)$ ， $I(x, y + 1)$ 中不论哪一个为255，令 $I(x, y) = 255$ 。





换句话说，如果将上面的操作执行两次，则可以扩大两格。

在实际进行形态学处理的时候，待操作的像素4—近邻与矩阵  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$  相乘，结果大于255的话，将中心像素设为255。

输入 (imori.jpg)	大津の二值化(answers_image/answer_4.jpg)	输出 (answers_image/answer_47.jpg)
		

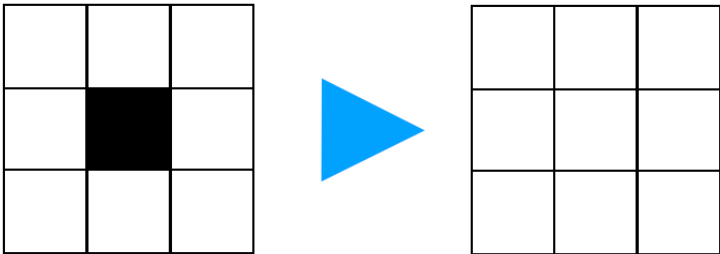
答案

- Python >> [answers\\_py/answer\\_47.py](#)
- C++ >> [answers\\_cpp/answer\\_47.cpp](#)




## 问题四十八：腐蚀（Erode）

将 `imori.jpg` 大津二值化之后，进行两次形态学腐蚀处理。

形态学处理中腐蚀操作如下：对于待操作的像素  $I(x, y) = 255$ ,  $I(x, y - 1)$ ,  $I(x - 1, y)$ ,  $I(x + 1, y)$ ,  $I(x, y + 1)$  中不论哪一个不为255，令  $I(x, y) = 0$ 。



在实际进行形态学处理的时候，待操作的像素4—近邻与矩阵  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$  相乘，结果小于  $255 \times 4$  的话，将中心像素设为0。

输入 (imori.jpg)	大津の二值化(answers/answer_4.jpg)	输出 (answers/answer_48.jpg)
		

答案

- Python >> [answers\\_py/answer\\_48.py](#)
- C++ >> [answers\\_cpp/answer\\_48.cpp](#)




# 问题四十九：开运算（Opening Operation）

大津二值化之后，进行开运算（ $N=1$ ）吧。

开运算，即先进行 $N$ 次腐蚀再进行 $N$ 次膨胀。



开运算可以用来去除仅存的小块像素。

输入 (imori.jpg)	大津の二值化(answers/answer_4.jpg)	输出 (answers/answer_49.jpg)
		

答案

- Python >> [answers\\_py/answer\\_49.py](#)
- C++ >> [answers\\_cpp/answer\\_49.cpp](#)




# 问题五十：闭运算（Closing Operation）

Canny边缘检测之后，进行 $N = 1$ 的闭处理吧。

闭运算，即先进行 $N$ 次膨胀再进行 $N$ 次腐蚀。



闭运算能够将中断的像素连接起来。

输入 (imori.jpg)	Canny(answers/answer_43.jpg)	输出 (answers/answer_50.jpg)
		



## 答案

- Python >> [answers\\_py/answer\\_50.py](#)
- C++ >> [answers\\_cpp/answer\\_50.cpp](#)