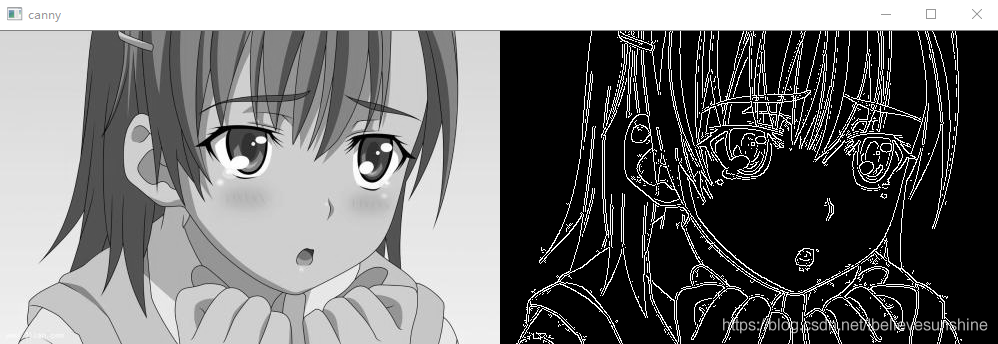
# [opencv 检测图像边缘 Canny算法应用](https://www.cnblogs.com/wojianxin/p/12597673.html)

**图解边缘检测**



**opencv 应用Canny算法进行边缘检测**

**import cv2 as cv**

**import numpy as np**

**img = cv.imread('baby\_g.jpg', 0)**

**# 二值化图像处理后，边缘检测效果更好**

**\_, thresh = cv.threshold(img, 0, 255, cv.THRESH\_BINARY + cv.THRESH\_OTSU)**

**# canny边缘检测，60以下置为0,180以上置为255,第2、3参数的作用可查看本文最后一部分内容**

**edges = cv.Canny(thresh, 60, 180)**

**cv.imshow('canny', np.hstack((img, edges)))**

**cv.waitKey(0)**

**cv.destroyAllWindows()**

**实验结果**



**Canny算法原理**

请参考Canny边缘检测算法原理详解 [Canny边缘检测算法原理](http://codec.wang/opencv-python-edge-detection/)

**Canny算法手动实现**

|  |
| --- |
| **一. 总的算法流程：**  **① 使用高斯滤波器滤波**  **② 使用 Sobel 滤波器滤波获得在 x 和 y 方向上的输出，在此基础上求出边缘的强度和边缘的角度**  https://img-blog.csdnimg.cn/20200407153103944.jpg          edge 为边缘强度，tan 为边缘角度 ↑  **③ 对边缘角度进行量化处理**  https://imgconvert.csdnimg.cn/aHR0cHM6Ly91cGxvYWQtaW1hZ2VzLmppYW5zaHUuaW8vdXBsb2FkX2ltYWdlcy8xNzIyMTQ5OS1lYmY4NzEwYjYyZmQ3Y2U2LnBuZw?x-oss-process=image/format,png          对边缘角度进行量化处理算法 ↑  **④ 根据边缘角度对边缘强度进行非极大值抑制（Non-maximum suppression），使图像边缘变得更细**  https://imgconvert.csdnimg.cn/aHR0cHM6Ly91cGxvYWQtaW1hZ2VzLmppYW5zaHUuaW8vdXBsb2FkX2ltYWdlcy8xNzIyMTQ5OS00NWUzYTk2NTVmOTE1OWMyLmpwZw?x-oss-process=image/format,png          非极大值抑制算法：0°时取(x,y)、(x+1,y)、(x-1,y) 中的最大值，其它角度类似 ↑  **⑤ 使用滞后阈值对图像进行二值化处理，优化图像显示效果**  https://imgconvert.csdnimg.cn/aHR0cHM6Ly91cGxvYWQtaW1hZ2VzLmppYW5zaHUuaW8vdXBsb2FkX2ltYWdlcy8xNzIyMTQ5OS1jMGQ5YzI0N2RiNTEwZmVjLnBuZw?x-oss-process=image/format,png          算法如上所示 ↑  **⑥ 输出图像边缘提取效果**  **二. 使用python手动实现 Canny 算法，完成图像边缘提取**  **# writer：wojianxinygcl@163.com**  **# date ：2020.3.20**  **import cv2**  **import numpy as np**  **import matplotlib.pyplot as plt**    **def Canny(img):**    **# Gray scale**  **def BGR2GRAY(img):**  **b = img[:, :, 0].copy()**  **g = img[:, :, 1].copy()**  **r = img[:, :, 2].copy()**    **# Gray scale**  **out = 0.2126 \* r + 0.7152 \* g + 0.0722 \* b**  **out = out.astype(np.uint8)**    **return out**      **# Gaussian filter for grayscale**  **def gaussian\_filter(img, K\_size=3, sigma=1.4):**    **if len(img.shape) == 3:**  **H, W, C = img.shape**  **gray = False**  **else:**  **img = np.expand\_dims(img, axis=-1)**  **H, W, C = img.shape**  **gray = True**    **## Zero padding**  **pad = K\_size // 2**  **out = np.zeros([H + pad \* 2, W + pad \* 2, C], dtype=np.float)**  **out[pad : pad + H, pad : pad + W] = img.copy().astype(np.float)**    **## prepare Kernel**  **K = np.zeros((K\_size, K\_size), dtype=np.float)**  **for x in range(-pad, -pad + K\_size):**  **for y in range(-pad, -pad + K\_size):**  **K[y + pad, x + pad] = np.exp( - (x \*\* 2 + y \*\* 2) / (2 \* sigma \* sigma))**  **#K /= (sigma \* np.sqrt(2 \* np.pi))**  **K /= (2 \* np.pi \* sigma \* sigma)**  **K /= K.sum()**    **tmp = out.copy()**    **# filtering**  **for y in range(H):**  **for x in range(W):**  **for c in range(C):**  **out[pad + y, pad + x, c] = np.sum(K \* tmp[y : y + K\_size, x : x + K\_size, c])**    **out = np.clip(out, 0, 255)**  **out = out[pad : pad + H, pad : pad + W]**  **out = out.astype(np.uint8)**    **if gray:**  **out = out[..., 0]**    **return out**      **# sobel filter**  **def sobel\_filter(img, K\_size=3):**  **if len(img.shape) == 3:**  **H, W, C = img.shape**  **else:**  **H, W = img.shape**    **# Zero padding**  **pad = K\_size // 2**  **out = np.zeros((H + pad \* 2, W + pad \* 2), dtype=np.float)**  **out[pad : pad + H, pad : pad + W] = img.copy().astype(np.float)**  **tmp = out.copy()**    **out\_v = out.copy()**  **out\_h = out.copy()**    **## Sobel vertical**  **Kv = [[1., 2., 1.],[0., 0., 0.], [-1., -2., -1.]]**  **## Sobel horizontal**  **Kh = [[1., 0., -1.],[2., 0., -2.],[1., 0., -1.]]**    **# filtering**  **for y in range(H):**  **for x in range(W):**  **out\_v[pad + y, pad + x] = np.sum(Kv \* (tmp[y : y + K\_size, x : x + K\_size]))**  **out\_h[pad + y, pad + x] = np.sum(Kh \* (tmp[y : y + K\_size, x : x + K\_size]))**    **out\_v = np.clip(out\_v, 0, 255)**  **out\_h = np.clip(out\_h, 0, 255)**    **out\_v = out\_v[pad : pad + H, pad : pad + W]**  **out\_v = out\_v.astype(np.uint8)**  **out\_h = out\_h[pad : pad + H, pad : pad + W]**  **out\_h = out\_h.astype(np.uint8)**    **return out\_v, out\_h**      **# get edge strength and edge angle**  **def get\_edge\_angle(fx, fy):**  **# get edge strength**  **edge = np.sqrt(np.power(fx.astype(np.float32), 2) + np.power(fy.astype(np.float32), 2))**  **edge = np.clip(edge, 0, 255)**    **# make sure the denominator is not 0**  **fx = np.maximum(fx, 1e-10)**  **#fx[np.abs(fx) <= 1e-5] = 1e-5**    **# get edge angle**  **angle = np.arctan(fy / fx)**    **return edge, angle**      **# 将角度量化为0°、45°、90°、135°**  **def angle\_quantization(angle):**  **angle = angle / np.pi \* 180**  **angle[angle < -22.5] = 180 + angle[angle < -22.5]**  **\_angle = np.zeros\_like(angle, dtype=np.uint8)**  **\_angle[np.where(angle <= 22.5)] = 0**  **\_angle[np.where((angle > 22.5) & (angle <= 67.5))] = 45**  **\_angle[np.where((angle > 67.5) & (angle <= 112.5))] = 90**  **\_angle[np.where((angle > 112.5) & (angle <= 157.5))] = 135**    **return \_angle**      **def non\_maximum\_suppression(angle, edge):**  **H, W = angle.shape**  **\_edge = edge.copy()**    **for y in range(H):**  **for x in range(W):**  **if angle[y, x] == 0:**  **dx1, dy1, dx2, dy2 = -1, 0, 1, 0**  **elif angle[y, x] == 45:**  **dx1, dy1, dx2, dy2 = -1, 1, 1, -1**  **elif angle[y, x] == 90:**  **dx1, dy1, dx2, dy2 = 0, -1, 0, 1**  **elif angle[y, x] == 135:**  **dx1, dy1, dx2, dy2 = -1, -1, 1, 1**  **# 边界处理**  **if x == 0:**  **dx1 = max(dx1, 0)**  **dx2 = max(dx2, 0)**  **if x == W-1:**  **dx1 = min(dx1, 0)**  **dx2 = min(dx2, 0)**  **if y == 0:**  **dy1 = max(dy1, 0)**  **dy2 = max(dy2, 0)**  **if y == H-1:**  **dy1 = min(dy1, 0)**  **dy2 = min(dy2, 0)**  **# 如果不是最大值，则将这个位置像素值置为0**  **if max(max(edge[y, x], edge[y + dy1, x + dx1]), edge[y + dy2, x + dx2]) != edge[y, x]:**  **\_edge[y, x] = 0**    **return \_edge**      **# 滞后阈值处理二值化图像**  **# > HT 的设为255，< LT 的设置0，介于它们两个中间的值，使用8邻域判断法**  **def hysterisis(edge, HT=100, LT=30):**  **H, W = edge.shape**    **# Histeresis threshold**  **edge[edge >= HT] = 255**  **edge[edge <= LT] = 0**    **\_edge = np.zeros((H + 2, W + 2), dtype=np.float32)**  **\_edge[1 : H + 1, 1 : W + 1] = edge**    **## 8 - Nearest neighbor**  **nn = np.array(((1., 1., 1.), (1., 0., 1.), (1., 1., 1.)), dtype=np.float32)**    **for y in range(1, H+2):**  **for x in range(1, W+2):**  **if \_edge[y, x] < LT or \_edge[y, x] > HT:**  **continue**  **if np.max(\_edge[y-1:y+2, x-1:x+2] \* nn) >= HT:**  **\_edge[y, x] = 255**  **else:**  **\_edge[y, x] = 0**    **edge = \_edge[1:H+1, 1:W+1]**    **return edge**    **# grayscale**  **gray = BGR2GRAY(img)**    **# gaussian filtering**  **gaussian = gaussian\_filter(gray, K\_size=5, sigma=1.4)**    **# sobel filtering**  **fy, fx = sobel\_filter(gaussian, K\_size=3)**    **# get edge strength, angle**  **edge, angle = get\_edge\_angle(fx, fy)**    **# angle quantization**  **angle = angle\_quantization(angle)**    **# non maximum suppression**  **edge = non\_maximum\_suppression(angle, edge)**    **# hysterisis threshold**  **out = hysterisis(edge, 80, 20)**    **return out**      **if \_\_name\_\_ == '\_\_main\_\_':**  **# Read image**  **img = cv2.imread("../paojie.jpg").astype(np.float32)**    **# Canny**  **edge = Canny(img)**    **out = edge.astype(np.uint8)**    **# Save result**  **cv2.imwrite("out.jpg", out)**  **cv2.imshow("result", out)**  **cv2.waitKey(0)**  **cv2.destroyAllWindows()**  **三. 实验结果：**  https://imgconvert.csdnimg.cn/aHR0cHM6Ly91cGxvYWQtaW1hZ2VzLmppYW5zaHUuaW8vdXBsb2FkX2ltYWdlcy8xNzIyMTQ5OS02OGVlYWJhZGFmZjNjZTE3LmpwZw?x-oss-process=image/format,png          原图 ↑  https://imgconvert.csdnimg.cn/aHR0cHM6Ly91cGxvYWQtaW1hZ2VzLmppYW5zaHUuaW8vdXBsb2FkX2ltYWdlcy8xNzIyMTQ5OS00NzhhYjQ0ZjZjMzk3ZmRjLmpwZw?x-oss-process=image/format,png          Canny 算法 提取图像边缘结果 ↑          可以看到，我的代码如愿以偿地提取了图像边缘，而且效果很好！  **四. 参考内容：**          ① <https://www.cnblogs.com/wojianxin/p/12533526.html>          ② <https://www.jianshu.com/p/ff4c1a6a68d8> |

**opencv中使用Canny算法**

请参考opencv中的Canny算法使用 [opencv中的Canny算法使用](https://blog.csdn.net/Ibelievesunshine/article/details/105004518)

**通过程序观察cv.Canny函数第二个，第三个参数的作用**

import cv2 as cv

import numpy as np

def track\_back(x):

pass

img = cv.imread('paojie\_g.jpg', 0)

cv.namedWindow('window')

# 创建滑动条

cv.createTrackbar('maxVal', 'window', 30, 100, track\_back)

cv.createTrackbar('minVal', 'window', 180, 255, track\_back)

while(True):

# 获取滑动条的值

max\_val = cv.getTrackbarPos('maxVal', 'window')

min\_val = cv.getTrackbarPos('minVal', 'window')

edges = cv.Canny(img, min\_val, max\_val)

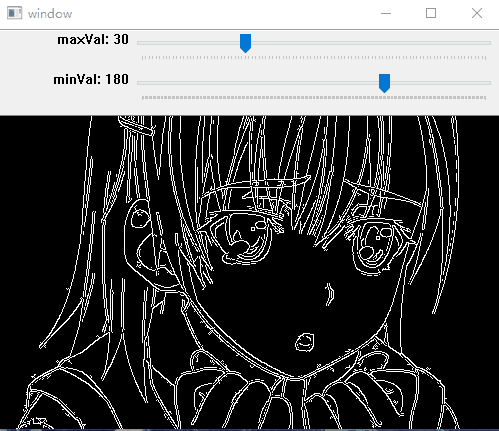
cv.imshow('window', edges)

# 按下ESC键退出

if cv.waitKey(30) == 27:

break

**结果**



# opencv canny边缘检测算法详解

一、边缘检测原理

图像的边缘由图像中两个相邻的区域之间的像素集合组成，是指图像中一个区域的结束和另外一个区域的开始。也可以这么理解，图像边缘就是图像中灰度值发生空间突变的像素的集合。梯度方向和幅度是图像边缘的两个性质，沿着跟边缘垂直的的方向，像素值的变化幅度比较平缓；而沿着与边缘平行的方向，则像素值变化幅度变化比较大。于是，根据该变化特性，通常会采用计算一阶或者二阶导数的方法来描述和检测图像边缘。

基于边缘检测的图像分割方法的基本思路是首先检测出图像中的边缘像素，然后再把这些边缘像素集合连结在一起便组成所要的目标区域边界。图像中的边缘可以通过对灰度值求导来检测确定，然而求导数可以通过计算微分算子来实现。在数字图像处理领域，微分运算通常被差分计算所近似代替。

二、 canny算法原理

在1986年，Canny边缘检测算子首次在论文《A Computational Approach to Edge Detection》中提出，目前，Canny边缘检测算子已广泛应用于各种图像处理视觉系统。由于它是从不同视觉对象中提取有用的结构信息，所以了要处理的数据量大大减少。JOHN CANNY总结出，不同视觉系统对边缘检测具有较为类似的要求，所以，发现可以采用一种应用意义广泛的边缘检测技术。

JOHN CANNY采用了如下步骤设计实现了canny算子。

\*\*(1)消除噪声。\*\*边缘检测的算法的主要思想采用了图像强度的一阶和二阶微分运算,但因为导数对噪声很敏感,所以再求导之前，先对图像源的数据进行平滑预处理再运用边缘检测算法。一般采用滤波器来改善图像的性噪比。所以Canny算子前,先通过高斯模板对原始数据进行卷积操作来抑制图像的噪声，再进行边缘检测。

\*\*(2)sobel梯度计算：\*\*平滑处理完，canny算子利用已有的一阶导数sobel微分算子来计算梯度，

Gx表示水平方向X的掩码模板，Gy表示垂直方向Y的掩码模板，采用这两个模板与图像进行卷积操作可得到图像边缘的梯度幅值和方向分别如式（4）和（5）所示：

梯度方向被归为垂直，水平，和两个对角线四类，其方向一般总是垂直于边界。

\*\*（3）对梯度幅值进行非极大值抑制。\*\*意思是遍历整个图像，将某个像素的灰度值与其梯度方向上前后两个像素的灰度值相比，判断其是否最大，如果不是那么这个像素值置为0，即不是边缘；如下图5. 1所示 ，每一列箭头的方向代表步骤二所检测出的（梯度方向与边缘垂直），数值表示对应的梯度方向的幅值。经过非极大值抑制处理之后，第一列所表示梯度方向的幅值2、4、3被置为0，第二列的所表示梯度方向的幅值3、5、4被置为0，以此类推，最终白色边框里的幅值5、6、7、6、7被当作疑似边缘像素点。

\*\*（4）使用双阈值算法检测和连接边缘。\*\*在上一个步骤得到了存在伪边缘的边缘集，因为通过单阈值处理选取边缘的操作比较难, 所以在Canny算法采用滞后阈值法减少伪边缘数量。如下图5. 1所示：Canny使用了滞后阈值,滞后阈值需要高阈值和低阈值,在进行边缘确定时依据下面的步骤第一,如果某一像素位置的幅值超过高阈值,该像素被保留为边缘像素:第二,如果某一像素位置的幅值小于低阈值,该像素被排除;第三,如果某一像素位置的幅值在两个阈值之间,该像素仅仅在连接到一个高于高阈值的像素时被保留。Canny算法的双阈值中, 大部分噪声被高阈值检测出去除了,但是也损失了有用的边缘信息, 较多的边缘信息则被低阈值检测得到的图像保留着,可取的高与低阈值比在2:1到3:1之间。

图5. 1非极大值抑制示意图图

图5. 2双阈值算法检测示意图

三、opencv 函数支持Canny()

函数原型：

CV\_EXPORTS\_W void Canny( InputArray image, OutputArray edges,

double threshold1, double threshold2,

int apertureSize = 3, bool L2gradient = false );

1

2

3

参数说明：

image：8位输入图像。

edges:输出边缘;单通道8位图像，与图像大小相同。

threshold1:迟滞过程的第一个阈值。

threshold2:迟滞过程的第二个阈值。

apertureSize : Sobel算子的孔径大小。

L2gradient:一个标志，指示是否应用更精确的方式计算图像梯度幅值.。

四、代码示例：

cv::Mat src;

src = cv::imread("D:\\QtProject\\Opencv\_Example\\canny\\canny.jpg", cv::IMREAD\_GRAYSCALE);

if (src.empty()) {

cout << "matTemplate Cannot load image" << endl;

return;

}

cv::imshow("src", src);

cv::Mat matCanny;

const int lowThreshold = 10;

const int maxThreshold = 200;

const int kernel\_size = 3;

cv::Canny(src, matCanny, lowThreshold, maxThreshold, kernel\_size);

cv::imshow("matCanny", matCanny);

程序运行效果：

