作业 2：边缘检测和边缘链接:从你的图像中检测对象

王琛然 151220104 17721502736@163.com 17721502736

(南京大学 计算机科学与技术系, 南京 210093)

1 实现细节

A. 边缘检测

1. sobel 算子

（1） 简介

在边缘检测中，sobel 边缘检测是一种常见的方法，sobel 算子分为检测 x 方向边缘的水平算子和检测 y 方向边缘的竖直算子，主要思想为：不同近邻对梯度的贡献有所不同，所以采取一种加权的方式，可以降低边缘模糊程度

（2） 算法思路

a. 定义两个方向的算子

x 方向算子:

$$H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

y 方向算子:

$$H_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

b. Gx 与 Gy 是横向及纵向边缘检测的图像：

Gx = H1*A, Gy = H2*A

每一个像素的横向纵向梯度值：

$$G = \sqrt{G_x^2 + G_y^2}$$

c. 设置阈值，对梯度值进行二值化，将大于阈值的梯度位置设为 1；小于阈值的部分设为 0

（3） 代码实现

```
function sobel = Sobel(image)
[m,n] = size(image);
image = double(image);
uSobel = image;
for i = 2: m - 1
    for j = 2: n - 1
        Gx = (image(i+1,j-1) + 2*image(i+1,j) + image(i+1,j+1)) - (image(i-1,j-1) + 2*image(i-1,j) + ima
        Gy = (image(i-1,j+1) + 2*image(i,j+1) + image(i+1,j+1)) - (image(i-1,j-1) + 2*image(i,j-1) + ima
        uSobel(i,j) = sqrt(Gx^2 + Gy^2);
    end
end
%归一化
Max = max(max(uSobel));
if Max ~= 0
    uSobel = uSobel / Max;
end
% 计算阈值
scale = 4;
cutoff = scale*mean2(uSobel);
thresh = sqrt(cutoff);
sobel = zeros(m, n);
%二值化
for i = 1: m
    for j = 1: n
        if uSobel(i, j) > thresh
            sobel(i, j) = 1;
        end
    end
end
```

2. Marr 算子

（1） 简介

拉普拉斯算子是不依赖边缘方向的二阶微分算子，具有各向同性的特点，对任何模版方向的灰度变换有相同响应，减少计算，但是由于拉普拉斯算子对噪声非常敏感，所以在进行拉普拉斯算法之前，可以先采用高斯滤波平滑噪声

（2） 算法思路

a. 使用高斯滤波平滑图像

$$H(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$G(x,y) = f(x,y) * H(x,y)$$

可以使用 matlab 自带的 imfilter 进行高斯平滑，图像会模糊，减少噪声的灰度

b. 对平滑后的图像采用拉普拉斯算子

拉普拉斯算子模版：（一种）

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

连续函数 f(i j)在位置(i, j)的拉普拉斯算子表达式：

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

c. 零交叉点判断边缘

零交叉点的确定：现有以 P 为中心的 3*3 邻域，查找相对方向，左/右、上/下和两个对角，当两个对角的二阶微分正负不同且二者之差大于设定的阈值，则 P 点是边缘。

（3） 代码实现：

```
function log = Log(input_image)
[m,n]=size(input_image);
% 高斯拉普拉斯
sigma=2;
fsize=ceil(sigma*3)*2+1;
op=fspecial('log',fsize,sigma);
op=op-sum(op(:))/numel(op);
b=filter2(op,input_image);
thresh=1.75*mean2(abs(b));
% 零点交叉判断边界
log = zeros(m, n);
for i=2:m-1
    for j=2:n-1
        if all(b(i, j)<0) && (b(i, j + 1)>0) && (abs(b(i, j + 1) - b(i, j)) > thresh)
            log(i, j) = 1;
        elseif all(b(i, j-1)>0) && (b(i, j)<0) && (abs(b(i,j-1)-b(i,j))>thresh)
            log(i, j) = 1;
        elseif all(b(i, j)<0) && (b(i+1,j)>0) && (abs(b(i,j)-b(i+1,j))>thresh)
            log(i, j) = 1;
        elseif all(b(i-1,j)>0) && (b(i,j)<0) && (abs(b(i-1,j)-b(i,j))>thresh)
            log(i, j) = 1;
        end
    end
end
```

3. canny 算子

（1） 简介

canny 算子是最优的阶梯型边缘检测算法，能有效的抑制噪声，较好的精确边缘的位置

（2） 算法思路：

a. 用高斯滤波平滑图像

$$H(x,y)=e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$G(x,y)=f(x,y)*H(x,y)$$

b. 用一阶偏导的有限差分计算梯度的幅值和方向

选择比较简单的 canny 算子模版：

$$H_1=\begin{bmatrix}-1 & 1\\ -1 & 1\end{bmatrix} \quad H_2=\begin{bmatrix}1 & 1\\ -1 & -1\end{bmatrix}$$

x 向的一阶偏导矩阵：

$$gx[i,j]=(f[i,j+1]-f[i,j]+f[i+1,j+1]-f[i+1,j])$$

y 方向的一阶偏导矩阵：

$$gy[i,j]=(f[i,j]-f[i+1,j]+f[i,j+1]-f[i+1,j+1])$$

梯度幅值：

$$m[i,j]=sqrt(gx[i,j]^2+gy[i,j]^2)$$

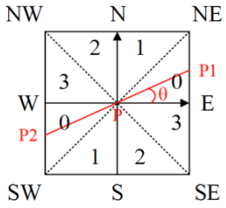
梯度方向：

$$theta[i,j]=arctan(\frac{gy[i,j]}{gx[i,j]})$$

c. 对梯度幅值进行非极大值抑制

非极大值抑制：寻找像素点局部最大值，将非极大值点所对应的灰度值置为 0

利用梯度方向，在 3*3 邻域范围内，将邻域的中心像素 P 与沿着梯度线的两个像素相比较，若 P 的梯度幅值是最大的，则 P 是边界。



（图源自网络）

利用线性插值确定 Gp1 和 Gp2 的坐标：

$$\tan(\theta) = G_y / G_x$$

$$G_{p1} = (1 - \tan(\theta)) \times E + \tan(\theta) \times NE$$

$$G_{p2} = (1 - \tan(\theta)) \times W + \tan(\theta) \times SW$$
（图源自网络）

d. 双阈值检测和链接边缘

经过非极大值抑制的图像仍会由于噪声等引起的边缘像素，所以为了减少噪声对图片的影响，需要用弱梯度值过滤边缘像素，保留具有高梯度值的边缘像素，即可设置高低阈值：当边缘像素的梯度值高于高阈值，则被标记为强边缘像素；如果边缘像素的梯度值低于低阈值，则标记为 0；如果边缘像素的梯度值在高阈值与低阈值中间，则标记为弱边缘像素。

对于弱边缘像素，为了确定其是真正的边缘像素还是由于噪声引起的，可以对弱边缘像素周围的 3*3 邻域查找是否存在强边缘像素，若存在强边缘像素，则可以认为该弱边缘像素是真正的边缘像素

（3） 代码实现：详情见 Canny.m 函数

B. 边缘链接&边缘追踪

1. 边缘链接

（1）简介

上述边缘检测的方法得到的图像会出现边缘断裂的情况，在边缘追踪的时候会有部分无法追踪，所以需要对断裂的边缘进行链接。连接的方法有很多，我实现的算法是依据 8 邻域和 16 邻域对空白像素点进行更新。

（2）算法思路：

a. 遍历整个图片，对每一个是边缘的像素点 P 的 3*3 邻域进行判断，计算边缘节点的个数

b. 若边缘节点的个数为 1，极大概率是边缘的断裂处，所以寻找 P 的 16 邻域

c. 若 P 的 16 邻域中有边缘像素且与 P 中间的像素点为 0，则置中间点为 1（在这次实验中，由于 rubberband_cap.png 这张图片橡皮筋的两个边缘的像素点挨得很近，所以在本次实验中只判断了水平和竖直方向上的 4 个 16 邻域节点）

（3）代码实现：详情可见 my_edgelinking.m 函数

2. 边缘追踪

（1）简介

本次实验的边缘追踪算法采用摩尔邻域追踪算法，基本思想为：搜索以边界起始点为中心的邻域，找到下一个边界像素，再对所选的边界像素查找邻域中的边界像素，直到回到起始点。

（2）算法：

a. 将起始点 s 放进边界点集 B

b. 设查找像素点 p 为起始点，p=s

c. 查找像素点 p 的摩尔邻域 M(p)

d. 设像素点 c 是在 M(p)中按顺时针方向的下一个节点，当 c != s

d.i 如果 c 为边缘像素，将 c 放入 B，查找像素点 p=c，原路返回到上一个空白像素（即在新

的 M(p)中，进入 p 像素的像素的下一个）

d.ii 如果 c 不是边缘像素，查找 M(p)中的下一个像素点

e. 重复 d 知道 c = s

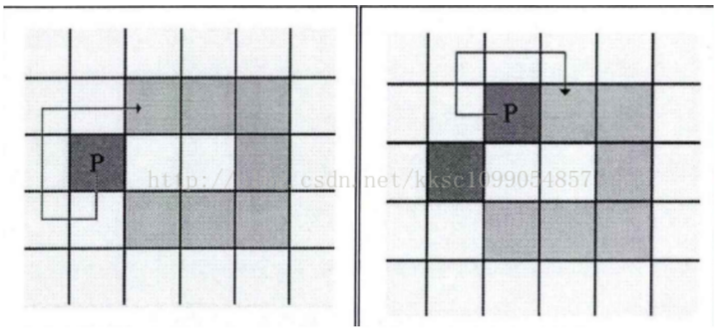
需要注意的是，c 和 s 判断相等不仅 c 和 s 是同一坐标，且梯度方向也相同

（3）确定下一个查找的像素点

摩尔邻域为：

2	3	4
1	X	5
8	7	6

以该图为例：



（图片源自网络）

从 p1 开始，搜索 p 的摩尔邻域，在 4 方向上找到了黑色像素 p2，进入 p2 的像素是从左边进入的，而对于 p2 来说，下一个查找的点是 M(p2)的 2 方向，为了使 p2 下一个查找的像素点为 M(p2)的 2 方向，设一个退出方向 exit_direction = [7 7 1 1 3 3 5 5]，对于 1，2 方向上的点，先前的白色像素是从下面进入的，所以二者都会回退到下面的节点，而在摩尔邻域中，下面的像素是 7 方向，所以设置它的退出方向为 7；其他 3、4；5、6；7、8 同理，分别从左、上、右进入，所以退出方向为 1、3、5；而在新节点 p2 上，若从哪个方向进入则从那个方向开始，所以可以利用 matlab 中的循环移位函数找到下一个方向。

（4）代码实现：详情可见 my_edgetracing.m 函数

2 结果

2.1 实验设置

用 matlab2017 版， 代码在 code 文件夹中，包括：

edge_test.m：程序入口

my_edge.m：边缘检测

my_edgelinking.m：边缘链接

my_edgetracing.m：边缘追踪

Canny.m：canny 算子

Sobel.m：sobel 算子

Log.m：log 算子

result 文件夹存放 rubberband_cap.png 的边缘检测和边缘追踪结果

运行时只需运行 edge_test.m 文件

2.2 实验结果

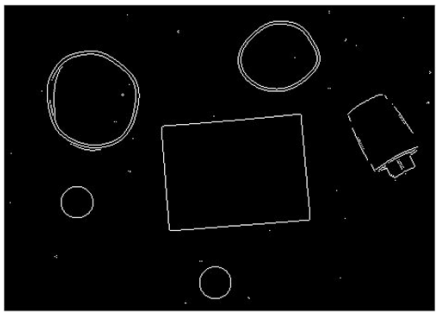
1. 边缘检测

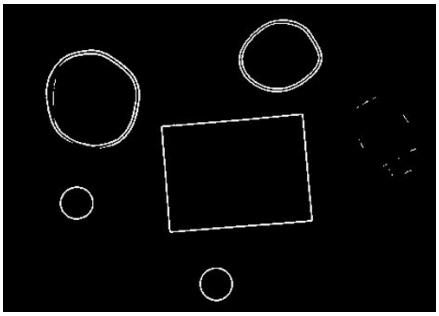
（1）Sobel 算子：

系统自带 sobel：

mySobel：

a.rubberband_cap.png：





b. moon.jpg：

系统自带 sobel：

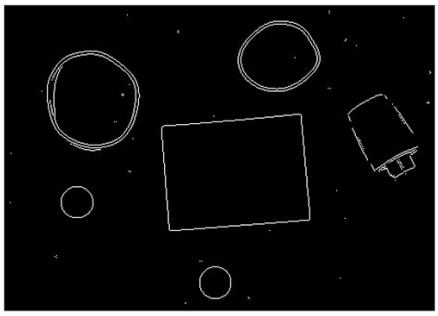
mySobel：

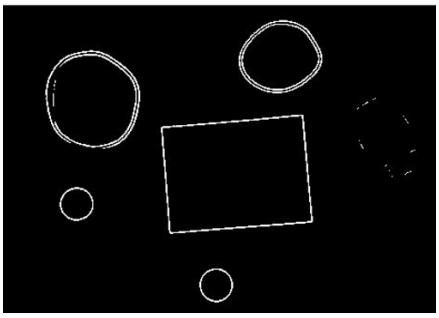
（2）Log 算子：

系统自带 log：

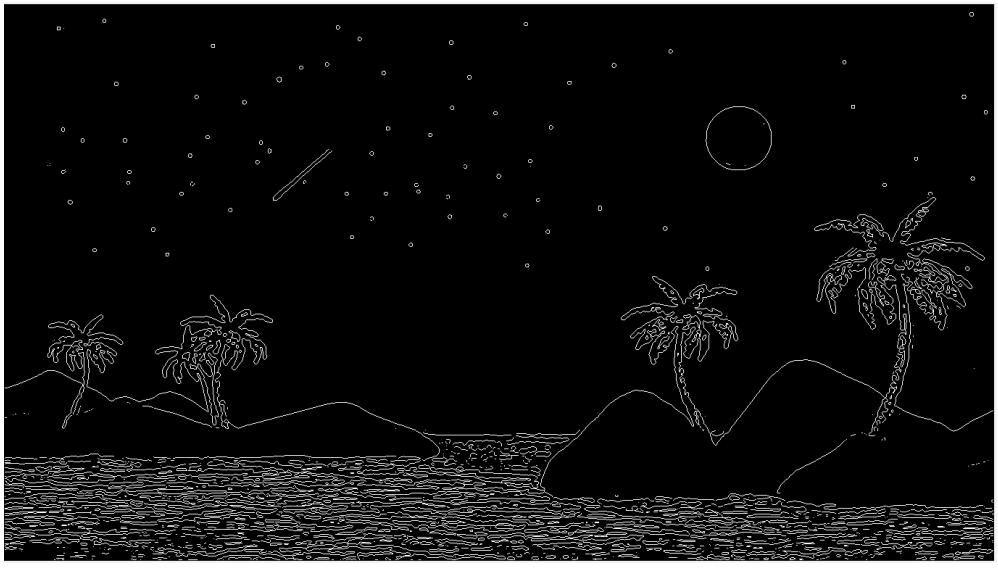
myLog：

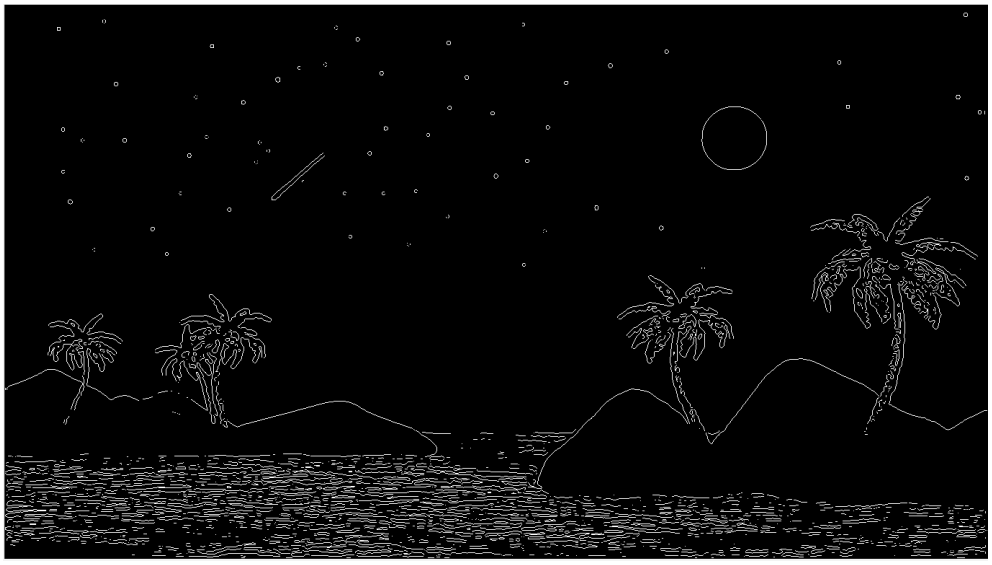
a.rubberband_cap.png：





b. moon.jpg：



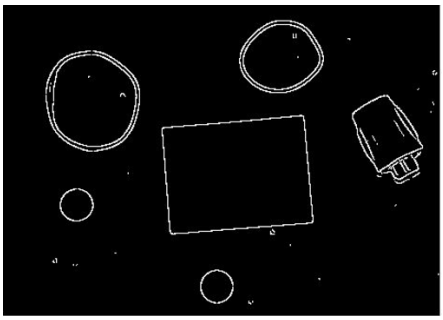


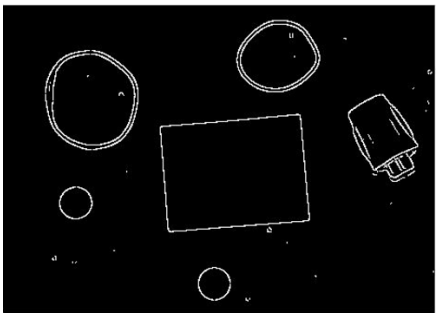
（3）Canny 算子：

系统自带 canny：

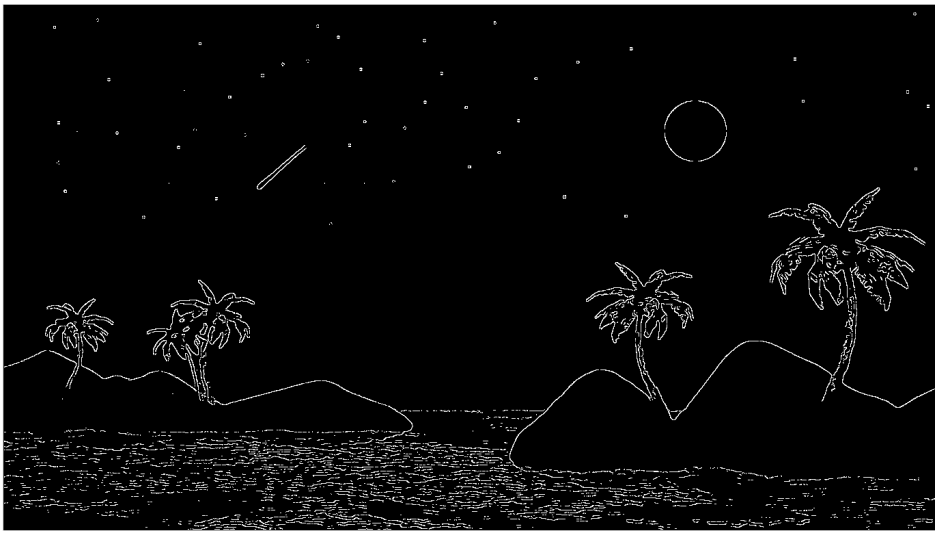
myCanny：

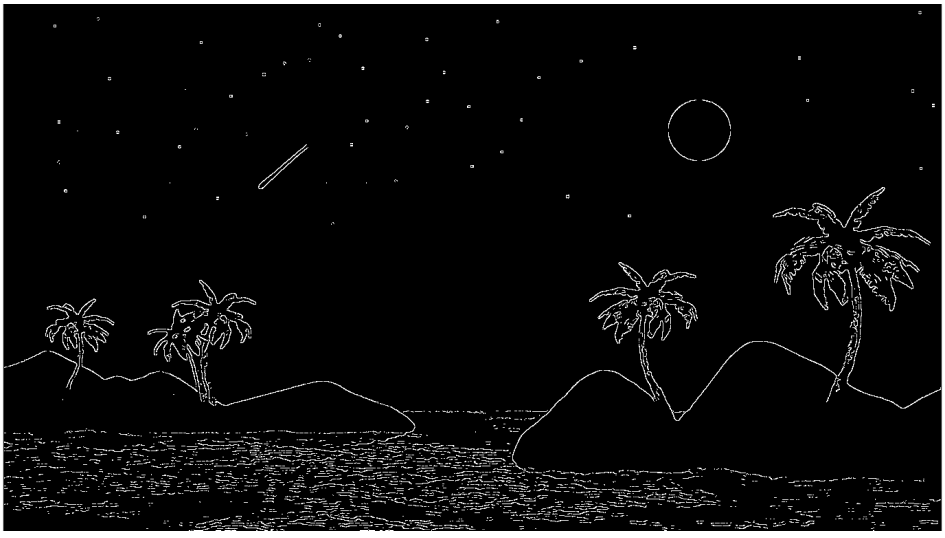
a.rubberband_cap.png：





b. moon.jpg：

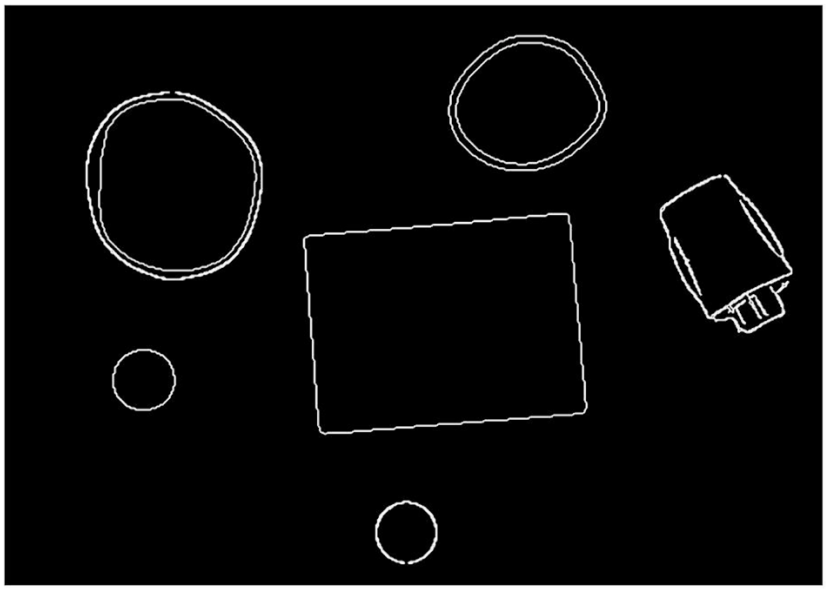




可以看出 Canny 算子的效果最好，自己实现的 sobel 算子不够好，与系统自带的函数有较大出入，log 算子也有缺陷，可能在阈值参数选择上不够优。

2. 边缘链接&边缘追踪：

在每个图形上取一个点，分别进行追踪，并一起显示，但是对绿色笔盖的边缘链接还不够好，有些细碎的地方没有连在一起，导致输出的部分不全。



3 参考文献

[1] 数字图像处理 ppt

[2] https://blog.csdn.net/mmmmmttttff/article/details/51271763 Canny

[3] http://www.cnblogs.com/tiandsp/archive/2012/12/13/2817240.html matlab 练习程序（Canny 边缘检测）

[4] https://blog.csdn.net/goodshot/article/details/10051309 基于 matlab 边缘提取的几种方法的比较

[5] https://blog.csdn.net/bettyshasha/article/details/51757185 LOG 边缘检测--Marr-Hildreth 边缘检测算法

[6] https://www.cnblogs.com/netilu/p/4285542.html 图像特征提取：Sobel 边缘检测

[7] https://blog.csdn.net/xddwz/article/details/78006655 边缘检测：Sobel 算子

[8] https://www.cnblogs.com/techyan1990/p/7291771.html 边缘检测之 Canny

[9] https://blog.csdn.net/cxf7394373/article/details/8790844 边缘断裂处理算法-边缘连接算法

[10] https://blog.csdn.net/kksc1099054857/article/details/74937848 边界跟踪算法（二）—摩尔邻域跟踪算法

[11]http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/index.html contour tracing

[12] https://ww2.mathworks.cn/matlabcentral/fileexchange/42144-moore-neighbor-boundary-trace Moore Neighbor Boundary Trace