**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)

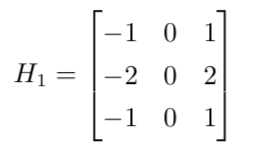
# 实现细节

1. 边缘检测
2. sobel算子
3. 简介

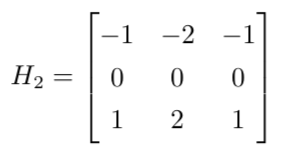
在边缘检测中，sobel边缘检测是一种常见的方法，sobel算子分为检测x方向边缘的水平算子和检测y方向边缘的竖直算子，主要思想为：不同近邻对梯度的贡献有所不同，所以采取一种加权的方式，可以降低边缘模糊程度

1. 算法思路
2. 定义两个方向的算子

x方向算子:



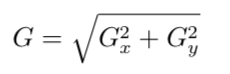
y方向算子:



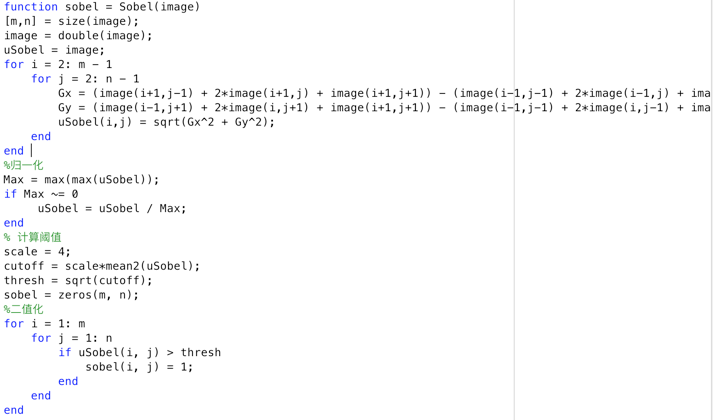
1. Gx与Gy是横向及纵向边缘检测的图像：

Gx = H1\*A, Gy = H2\*A

每一个像素的横向纵向梯度值：



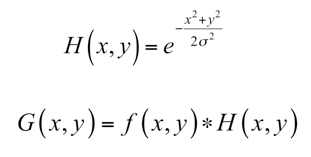
1. 设置阈值，对梯度值进行二值化，将大于阈值的梯度位置设为1；小于阈值的部分设为0
2. 代码实现



1. Marr算子
2. 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

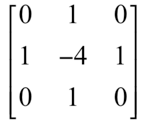
1. 算法思路
2. 使用高斯滤波平滑图像



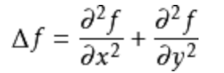
可以使用matlab自带的imfilter进行高斯平滑，图像会模糊，减少噪声的灰度

1. 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）



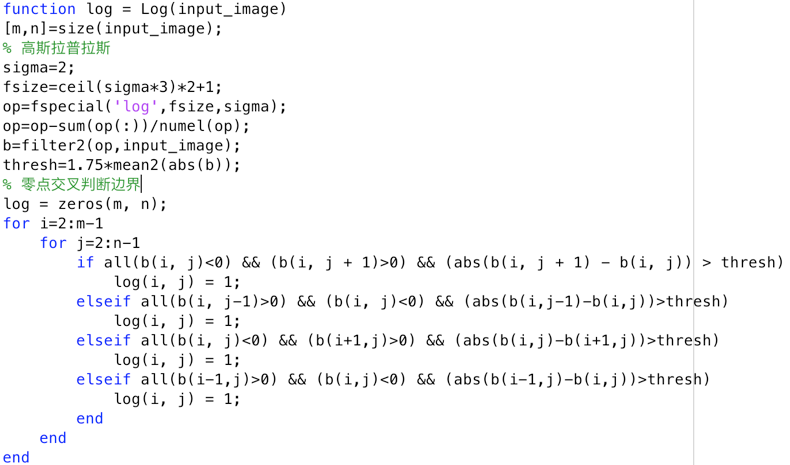
连续函数f(i j)在位置(i, j)的拉普拉斯算子表达式：



1. 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。

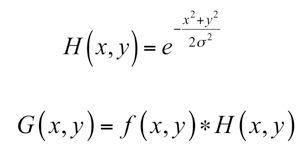
1. 代码实现：



1. canny算子
2. 简介

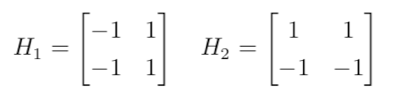
canny算子是最优的阶梯型边缘检测算法，能有效的抑制噪声，较好的精确边缘的位置

1. 算法思路：
2. 用高斯滤波平滑图像



1. 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：



x向的一阶偏导矩阵：

../../../Desktop/屏幕快照%202018-04-28%2016.35.05.png

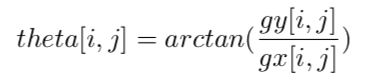
y方向的一阶偏导矩阵：

../../../Desktop/屏幕快照%202018-04-28%2016.36.30.png

梯度幅值：

../../../Desktop/屏幕快照%202018-04-28%2016.37.53.png

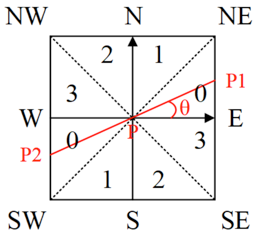
梯度方向：



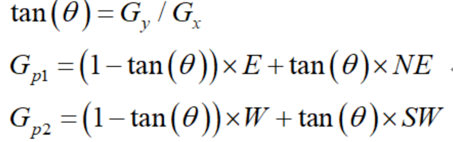
1. 对梯度幅值进行非极大值抑制

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：

（图源自网络）

1. 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

1. 代码实现：详情见Canny.m函数
2. 边缘链接&边缘追踪
3. 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

1. 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

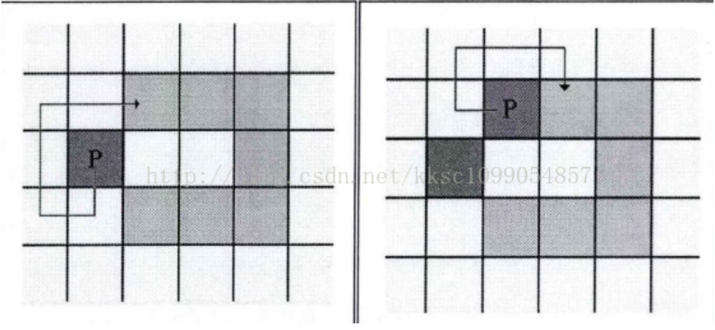
（3）确定下一个查找的像素点

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

# 结果

## 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

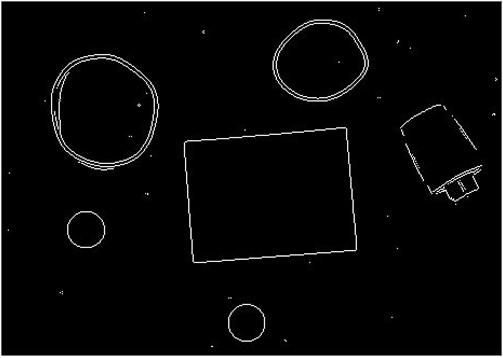
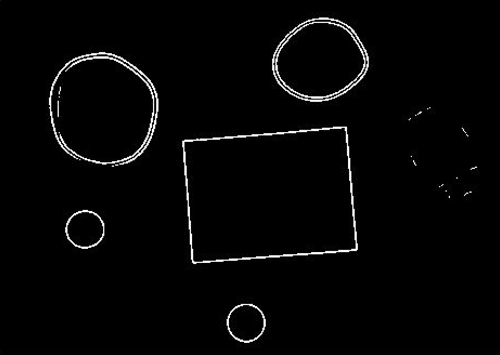
## 实验结果

1. 边缘检测

（1）Sobel算子：

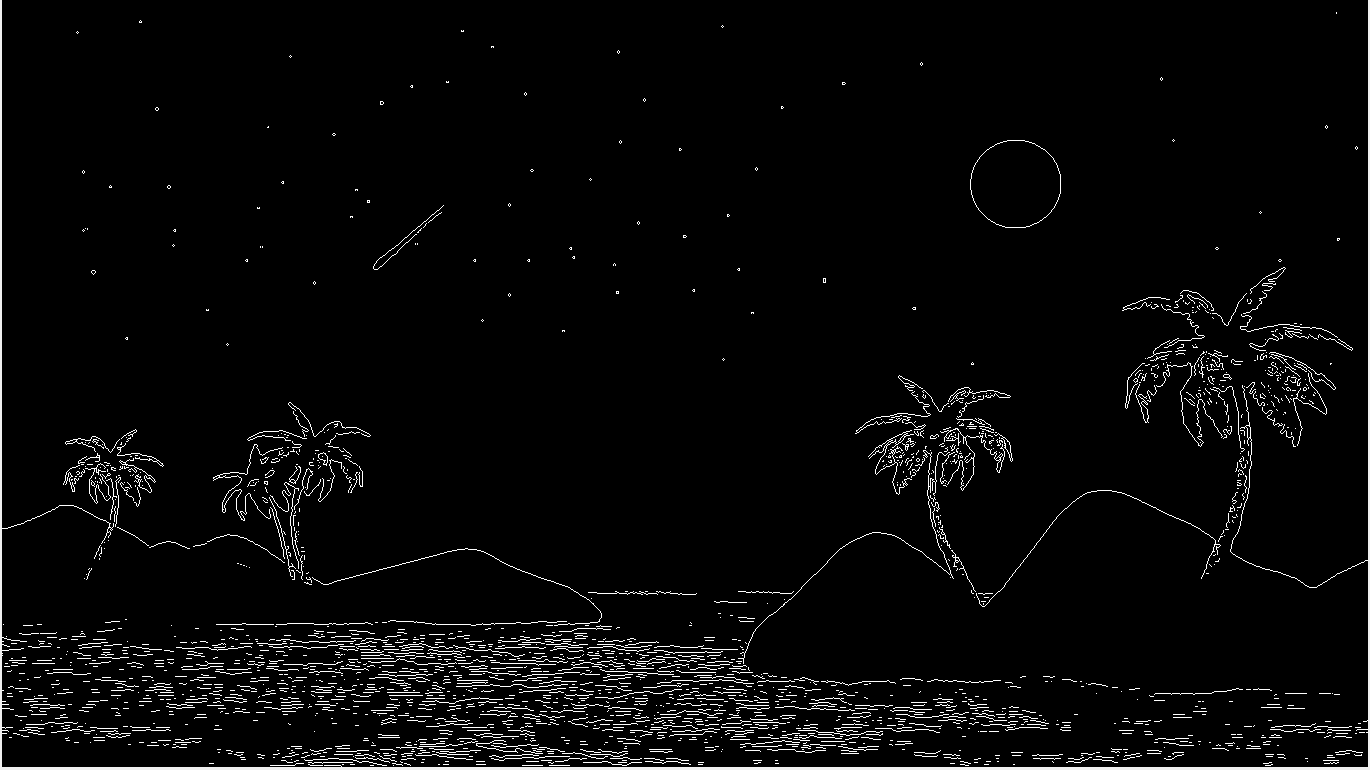
系统自带sobel： mySobel：

a.rubberband\_cap.png：

b. moon.jpg：

系统自带sobel：



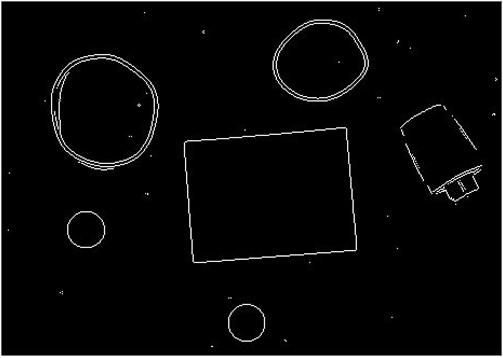
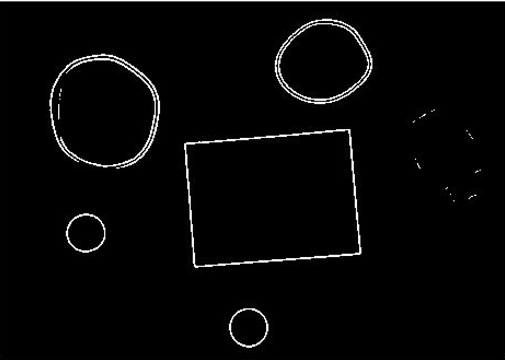
mySobel：



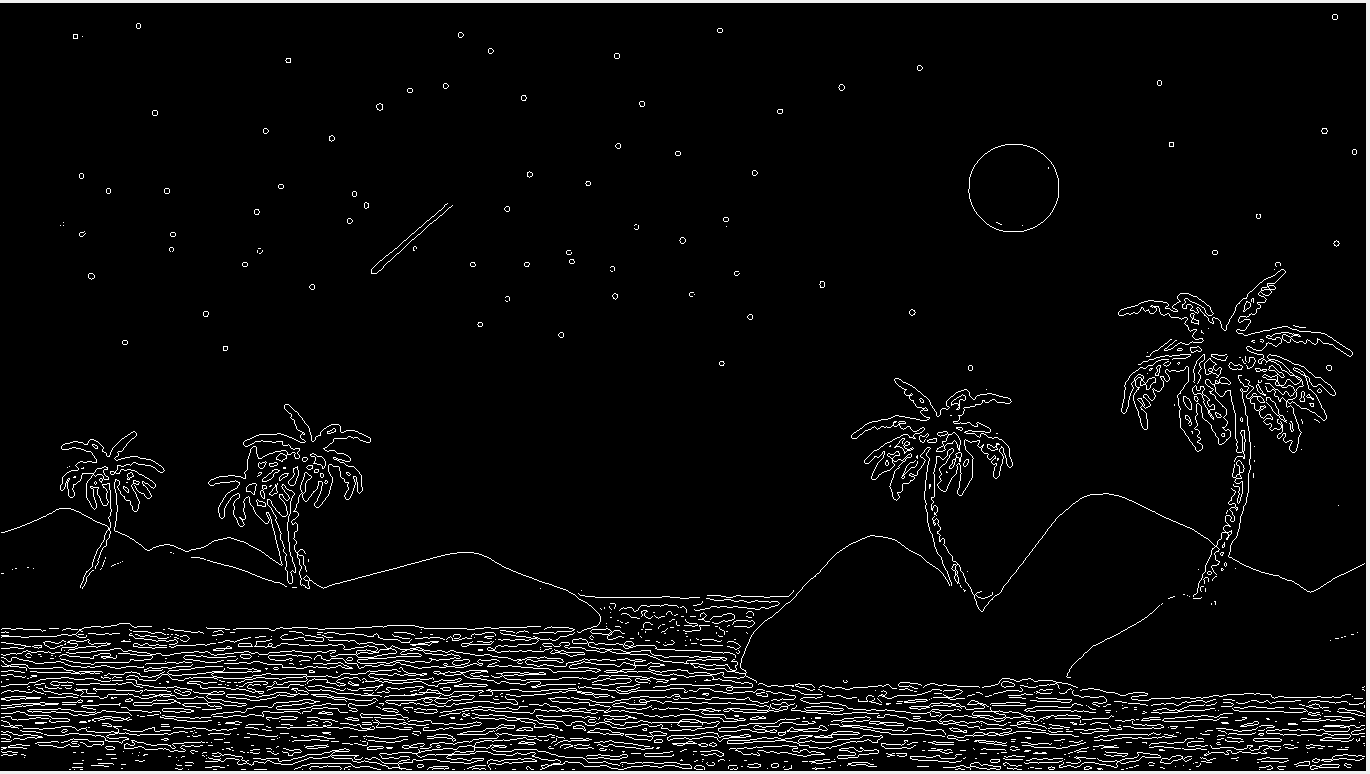
（2）Log算子：

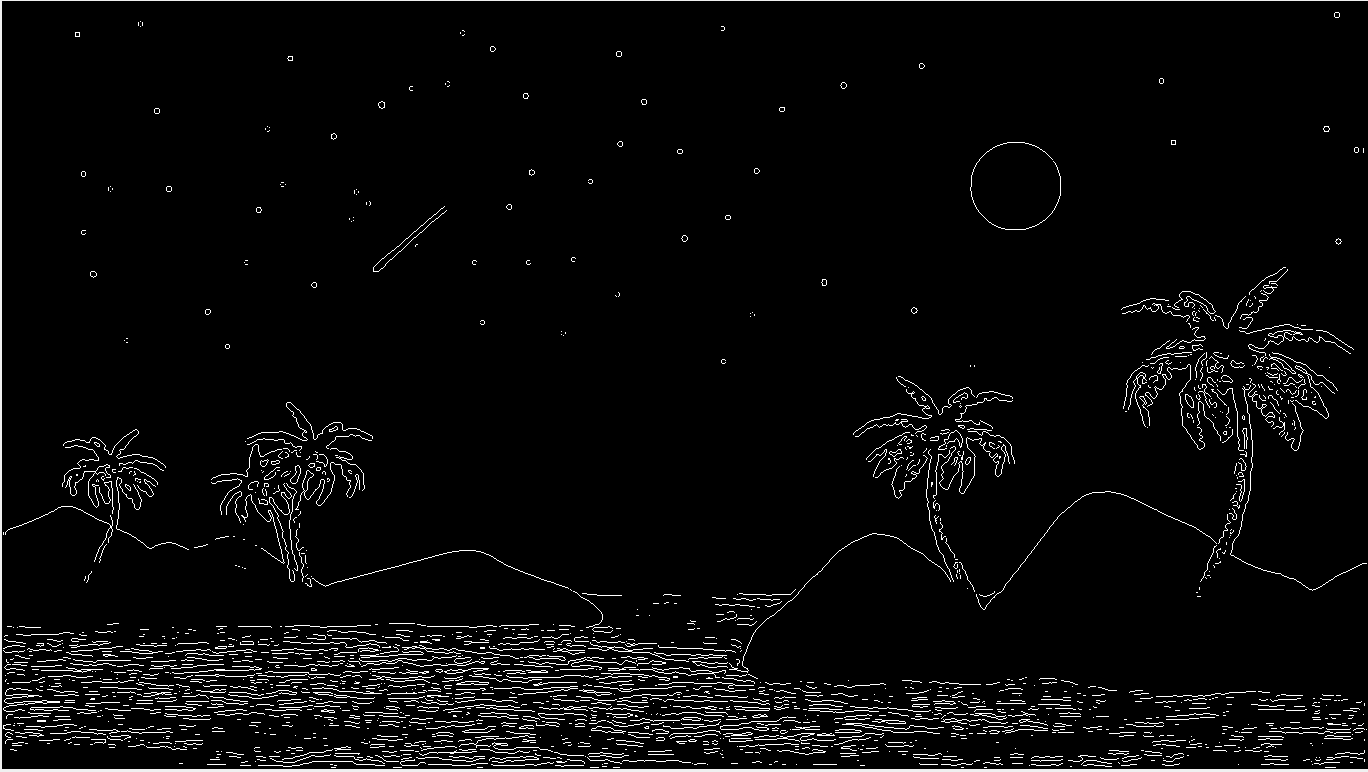
系统自带log： myLog：

a.rubberband\_cap.png：

b. moon.jpg：

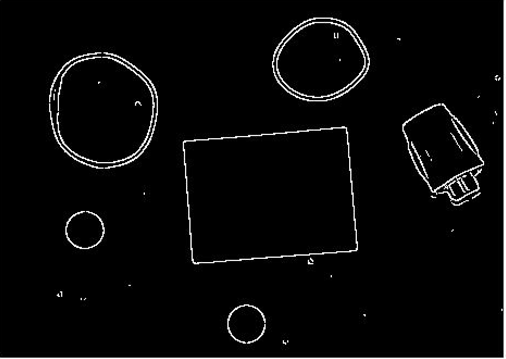
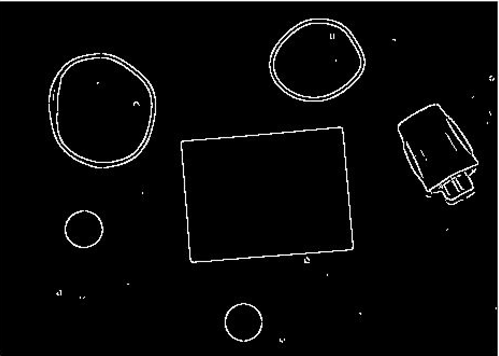




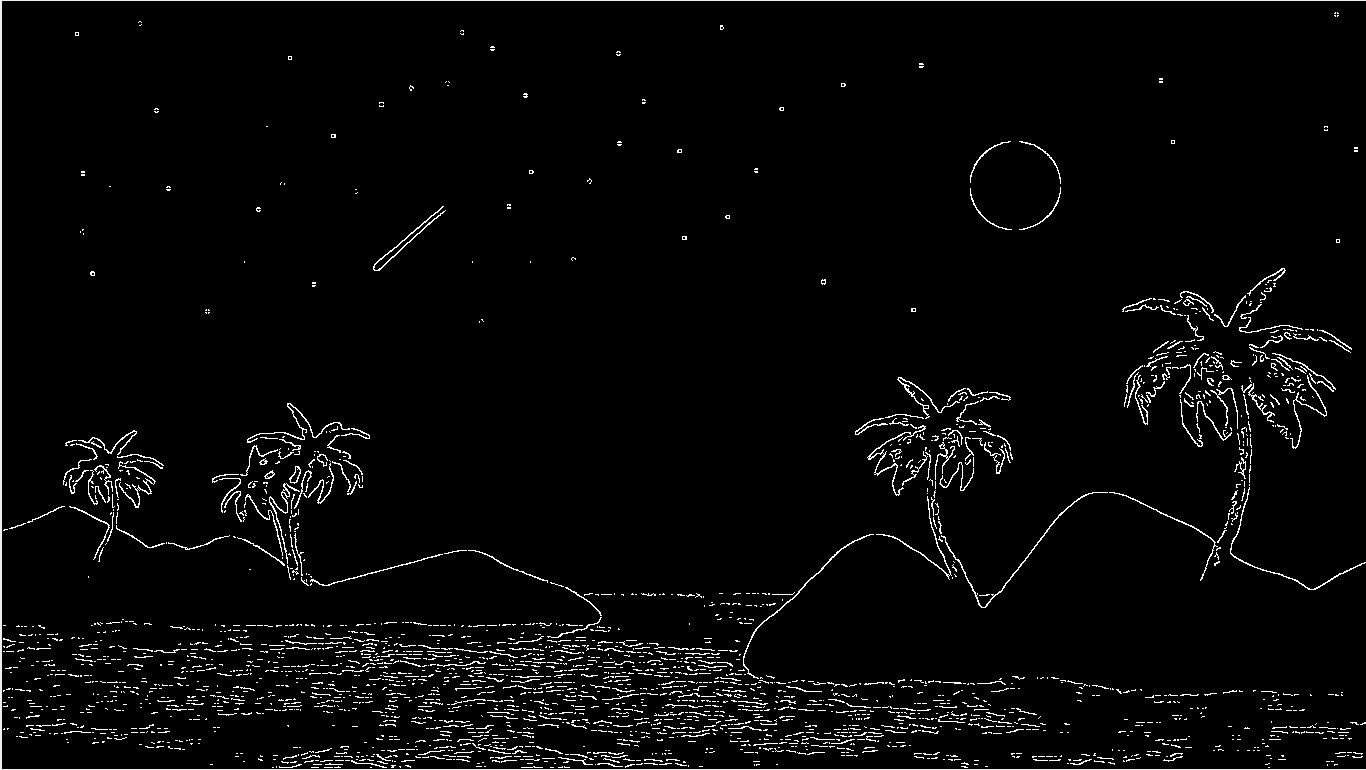
（3）Canny算子：

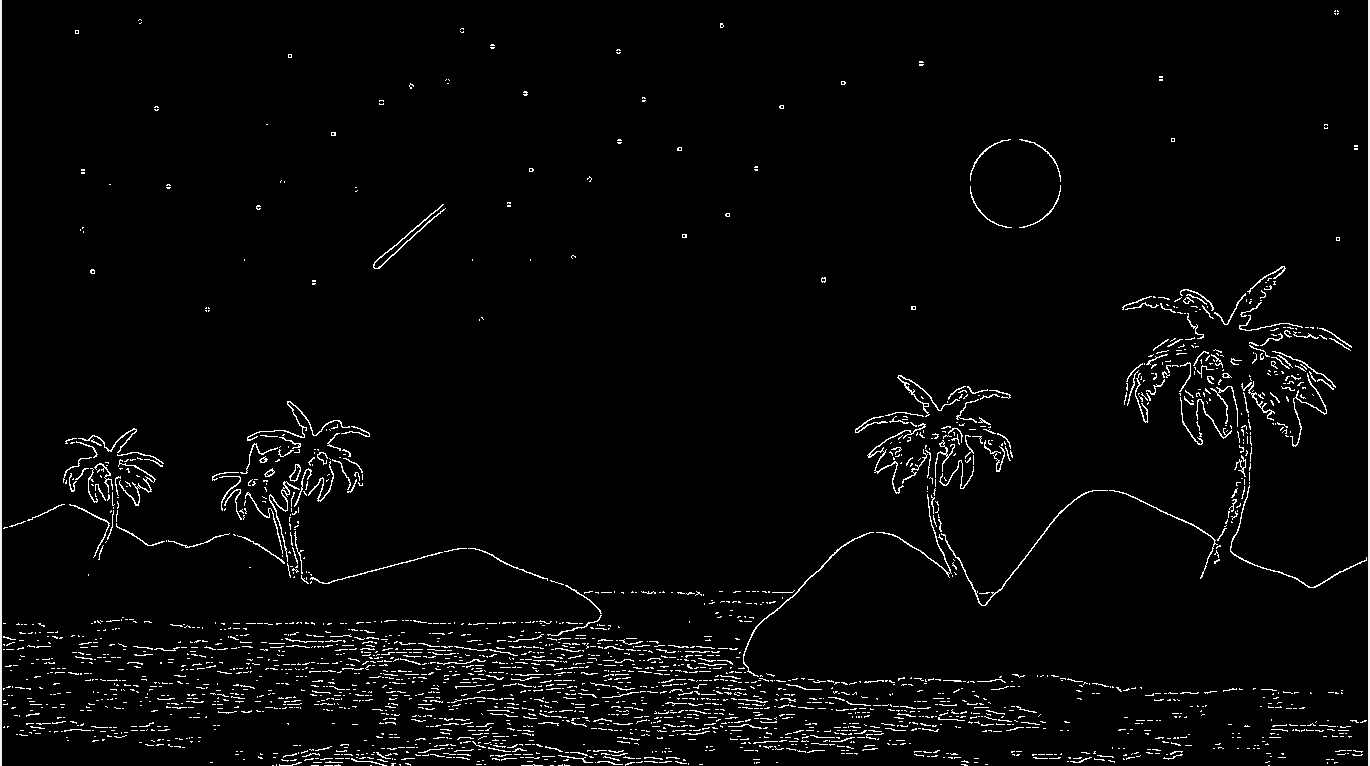
系统自带canny： myCanny：

a.rubberband\_cap.png：

b. moon.jpg：

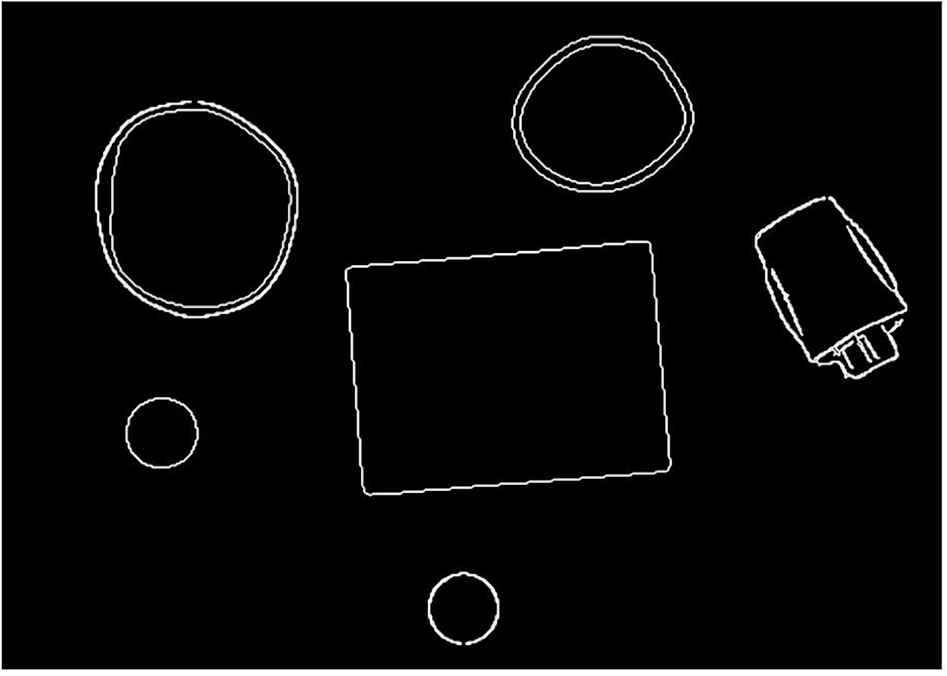




可以看出Canny算子的效果最好，自己实现的sobel算子不够好，与系统自带的函数有较大出入，log算子也有缺陷，可能在阈值参数选择上不够优。

1. 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。



# 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