实验二：边缘检测和边缘链接

银琦 141220132 141220132@smail.nju.edu.cn 18260066573

(南京大学 计算机科学与技术系，南京 210093)

1 实现细节

1.1 边缘检测

1.1.1 Roberts 算子

是一种利用局部差分算子寻找边缘的算子，它在 2x2 领域上计算对角导数，代码中采用了简化后的计算，用梯度的绝对值近似：$g(i,j)=|f(i,j)-f(i+1,j+1)|+|f(i+1,j)-f(i,j+1)|$。

如果 $g(i,j)\geq \theta$ ，则为边缘，否则不为边缘。

代码中首先得到原矩阵 input_image 经过 Roberts 算子处理后的矩阵 roberts，然后调用 graythresh 函数计算出该图像合适的阈值 robertThreshold，最后遍历矩阵 roberts，判断出边界后存入输出矩阵 outputRoberts。

1.1.2 Prewitt 算子

采用了 3x3 领域进行计算，公式如下：

$f'_i=f(i-1,j+1)+f(i,j+1)+f(i+1,j+1)-f(i-1,j-1)-f(i,j-1)-f(i+1,j-1)$

$f'_j=f(i+1,j-1)+f(i+,j)+f(i+1,j+1)-f(i-1,j-1)-f(i-1,j)-f(i-1,j+1)$

如果 $|f'_i|+|f'_j|\geq \theta$ ，则为边缘，否则不为边缘。

代码中首先得到原矩阵 input_image 经过 Prewitt 算子处理后的矩阵 Prewitt，然后调用 graythresh 函数计算出该图像合适的阈值 PrewittThreshold，最后遍历矩阵 Prewitt，判断出边界后存入输出矩阵 outputPrewitt。

1.1.3 Sobel 算子

不同近邻对梯度的贡献应该有所不同，所以采用了一种加权的方式。

$f'_i=f(i-1,j+1)+2*f(i,j+1)+f(i+1,j+1)-f(i-1,j-1)-2*f(i,j-1)-f(i+1,j-1)$

$f'_j=f(i+1,j-1)+2*f(i+1,j)+f(i+1,j+1)-f(i-1,j-1)-2*f(i-1,j)-f(i-1,j+1)$

如果 $|f'_i|+|f'_j|\geq \theta$ ，则为边缘，否则不为边缘。

代码中首先得到原矩阵 input_image 经过 Sobel 算子处理后的矩阵 sobel，然后调用 graythresh 函数计算出该图像合适的阈值 sobelThreshold，最后遍历矩阵 sobel，判断出边界后存入输出矩阵 outputSobel。

1.1.4 Laplacian 算子

前三种算子均为一阶算子，而 Laplacian 算子为二阶算子，通过求图像的二阶导数的零交叉点找到边缘点。

$\nabla^2 f=4*f(i,j)-f(i+1,j)-f(i-1,j)-f(i,j+1)-f(i,j-1)$

如果 $\nabla^2 f\geq \theta$ ，则为边缘，否则不为边缘。

代码中首先得到原矩阵 input_image 经过 Laplacian 算子处理后的矩阵 Laplacian，然后调用 graythresh 函数计算出该图像合适的阈值 LaplacianThreshold，最后遍历矩阵 Laplacian，判断出边界后存入输出矩阵

outputLaplacian。

1.1.5 Marr 算子

首先对图像进行平滑高斯滤波，然后对平滑后的图像采用 Laplacian 算子，接着通过零交叉点判断边缘，最后采用线性插值的方法估计边缘的位置。

代码中有两个对 Marr 的实现，第一个实现是从网络上找的代码，用于参考，第二个实现是自己实现的，在自己实现的代码中首先建立一个滤波器对原图像 input_image 进行高斯滤波，得到矩阵 Marr，然后调用 graythresh 函数计算出该图像合适的阈值 MarrThreshold，然后进行边界提取，如果某点与任一邻点符号相反，并且差值大于阈值，那么该点是边界，将结果存入矩阵 outputMarr 中。

1.1.6 Canny 算子

首先用高斯滤波器平滑图像，然后用一阶偏导的有限差分类计算梯度的幅值和方向，接着对梯度幅值进行非极大值抑制，最后用双阈值算法检测和连接边缘。

代码中首先对原图像 input_image 进行高斯滤波得到矩阵 canny，然后对 canny 求梯度，通过求出的梯度进行非极大值抑制，保留可能是边缘的点，接着定义了双阈值，大于高阈值的一定为边缘，在高低阈值之间的可能为边缘，在高低阈值之间的点的 8 邻点位置寻找可以连接到轮廓上的边缘，直到将边缘连接。此处我选择的低阈值为 0.02，高阈值为低阈值的 2.5 倍。

1.2 边缘连接

在边缘连接算法中，首先调用库函数 bwmorph 对边缘进行细化，然后使用一个队列，从传入的点开始对当前点的 8 邻域进行遍历寻找边缘，若找到了下一个点认为是边缘，则点进入队列，直到回到起点或者找不到边缘点。

2 结果

2.1 实验设置

实验中选用了 5 张图进行测试，在测试时只需要对 edge_test.m 中 10-14 行进行注释和取消注释即可，运行程序后会输出 9 个对话框，第 1 个对话框中绘制的是原图的灰度图像，第 2-8 个对话框分别是 Roberts 算子，Prewitt 算子，Sobel 算子，Laplacian 算子，非自己实现的 Marr 算子，自己实现的 Marr 算子，Canny 算子处理后的边缘图，第 9 个对话框是前八张图的对比图。

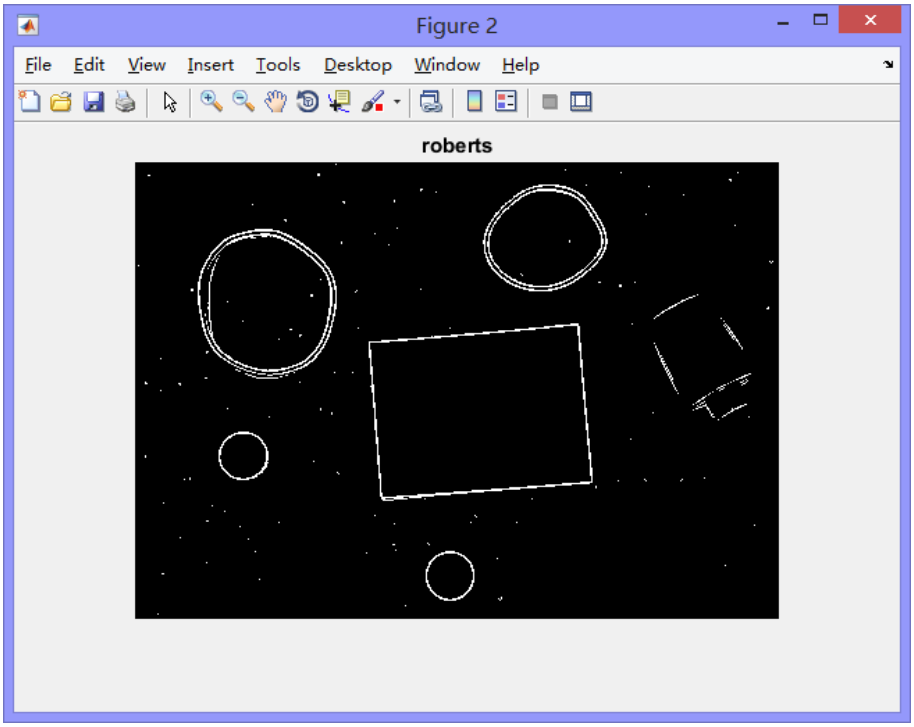
返回 edge_test.m 的输出图像是 Roberts 算子产生的结果，可以在 my_edge.m 的 479-485 行进行修改，但是如果测试图像时第一张图，修改之后可能会导致传入边缘连接函数的位置处为 0，使得边缘连接函数无法运行，其余图像修改之后无影响。

程序会自动判断，如果是第一张图“rubberband_cap.png”，那么会对传入的点进行边缘连接，则会输出 11 个对话框，前九个与其它相同，第 10 个为边缘连接的输出图像，第 11 个为 imtool 函数输出的用来定位的图像。实验对图像一中除了笔帽以外的物体都进行了边缘连接，但是左上角的橡皮圈的效果不太好，其余物体都能正确连接，最后一起在 background 上输出。

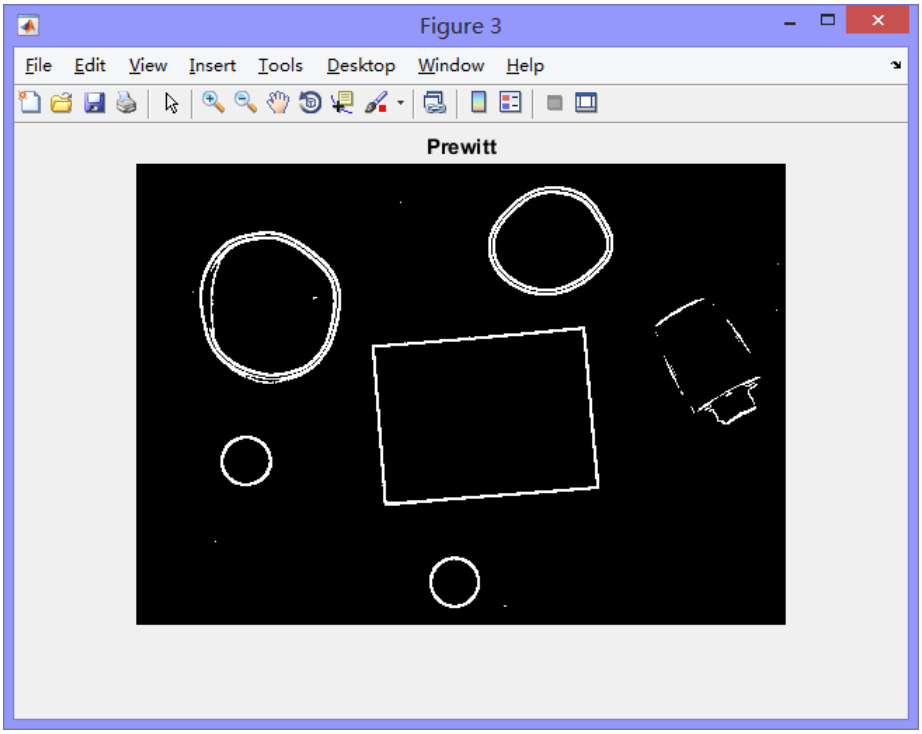
2.2 实验结果

图一：rubberband_cap.png

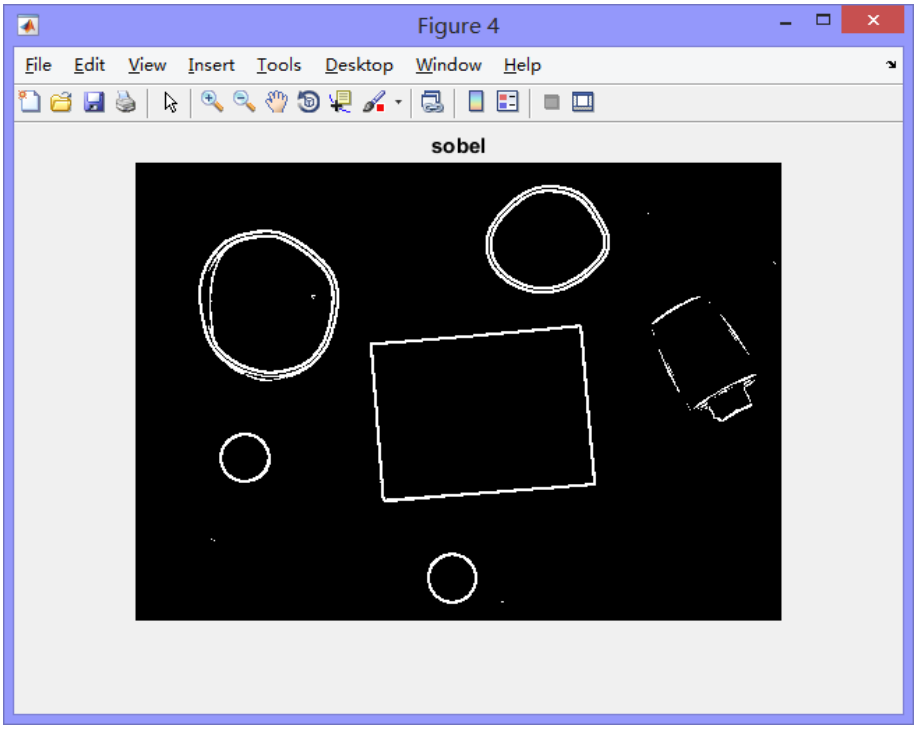
Roberts 算子



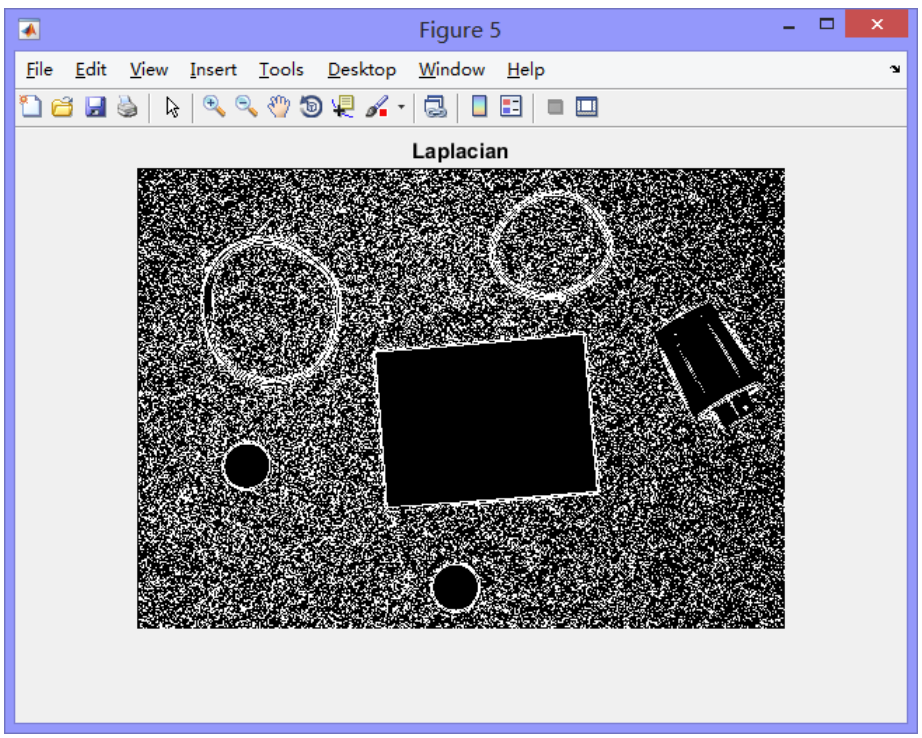
Prewitt 算子



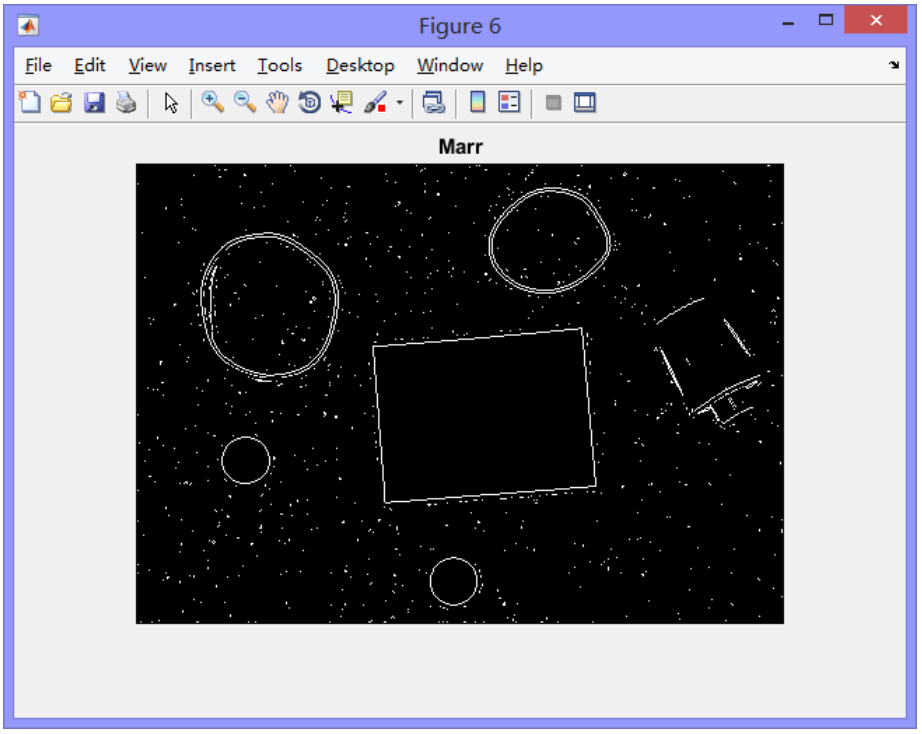
Sobel 算子



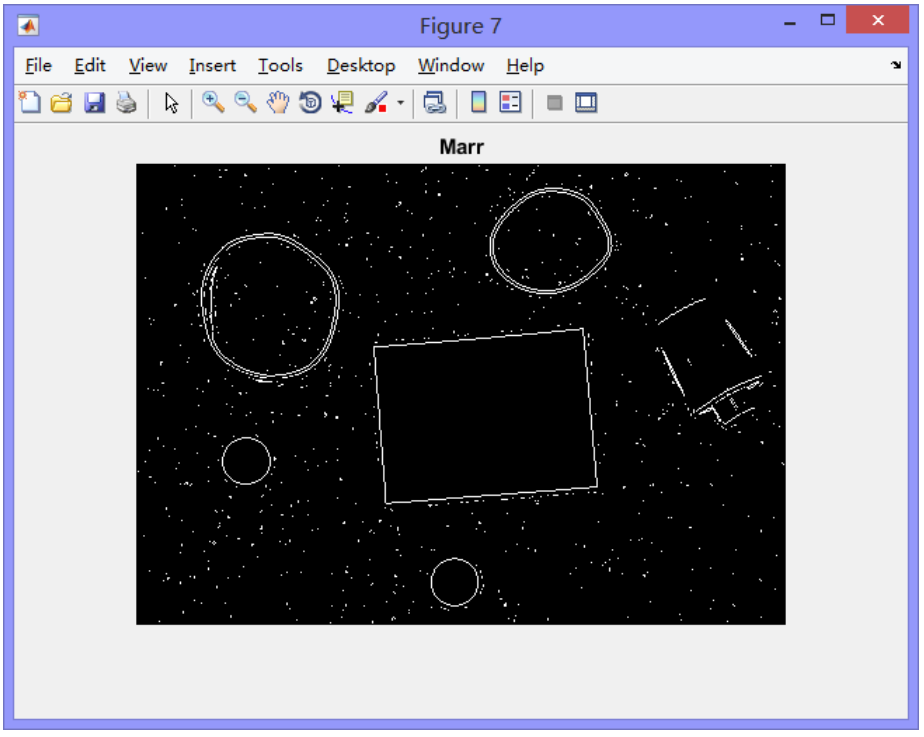
Laplacian 算子



非自己实现的 Marr 算子



自己实现的 Marr 算子



Canny 算子

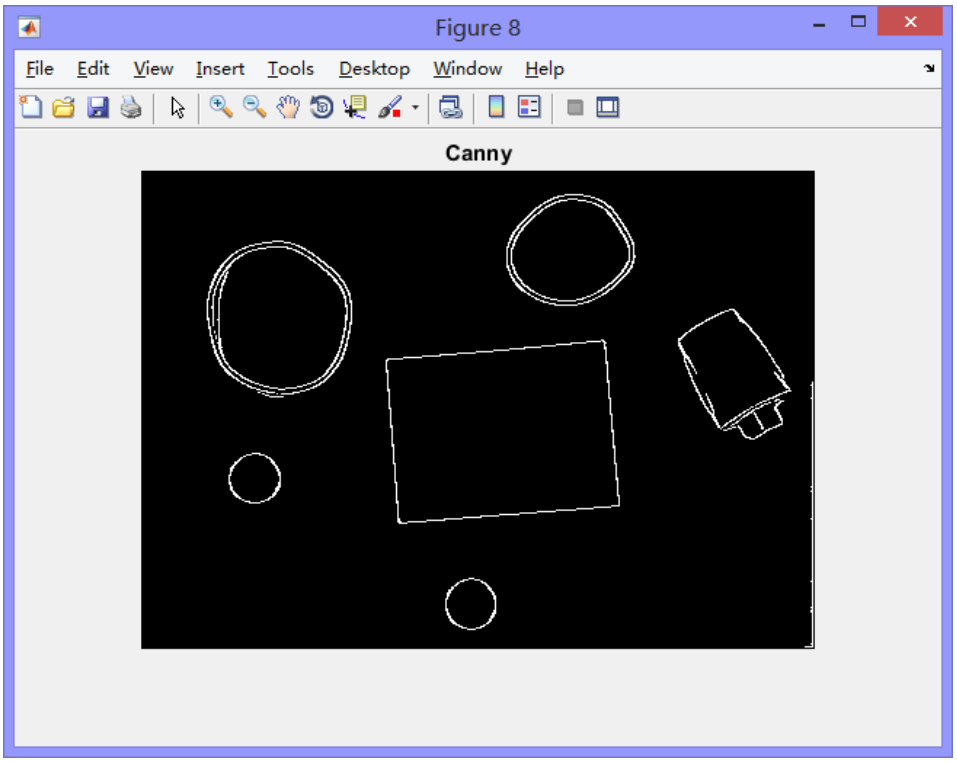


Figure 8
File Edit View Insert Tools Desktop Window Help
Canny

对比图

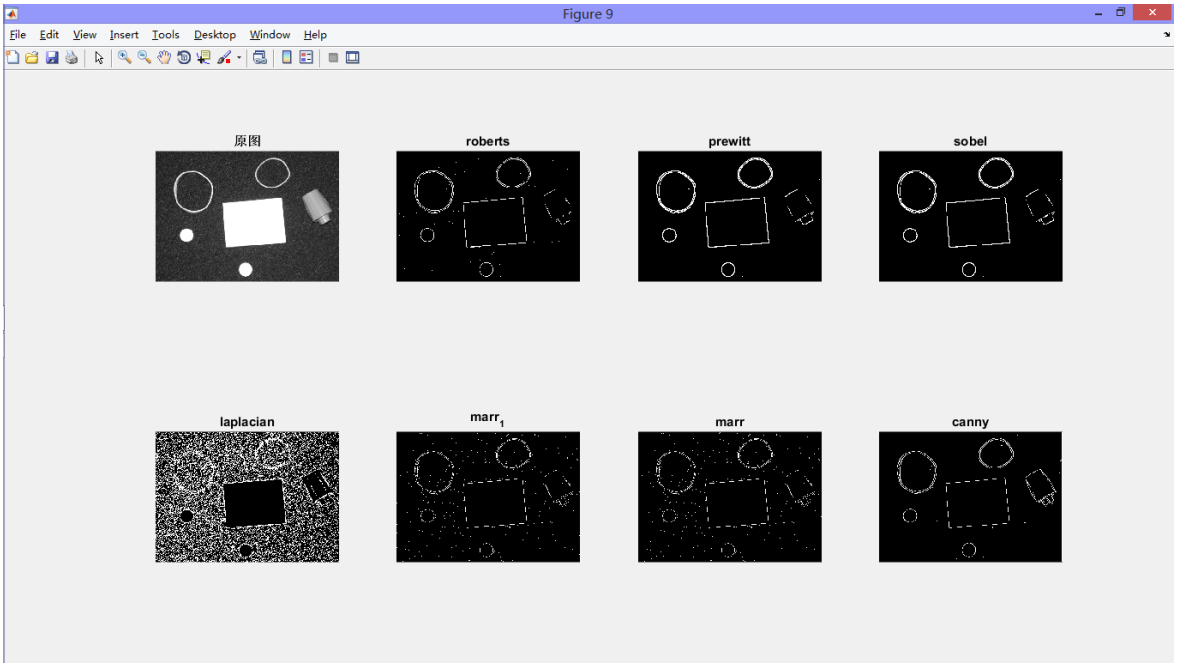
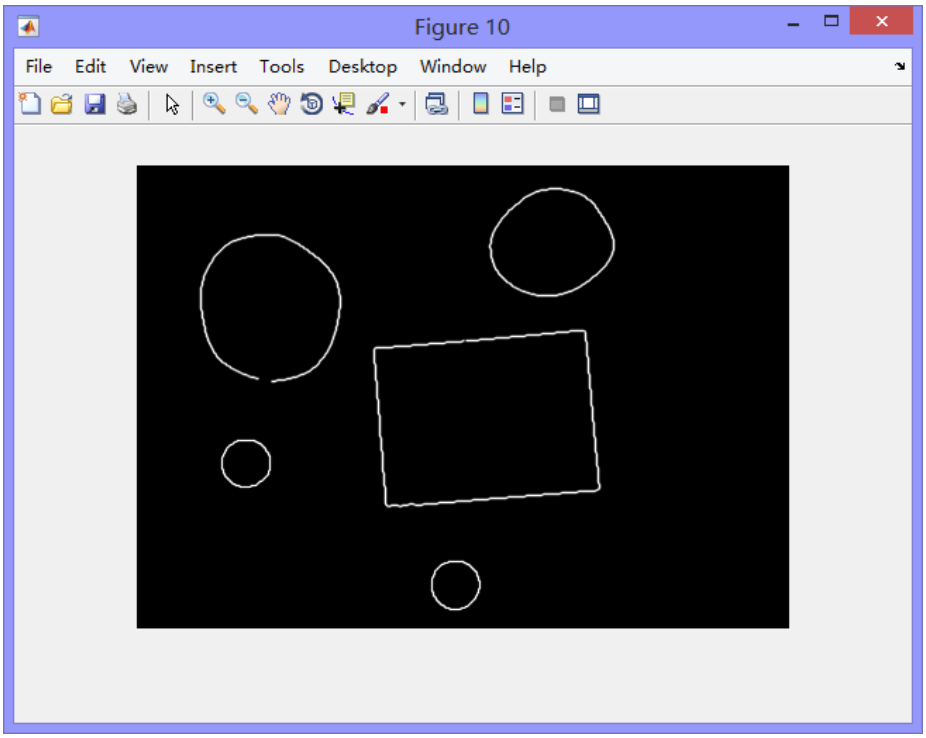


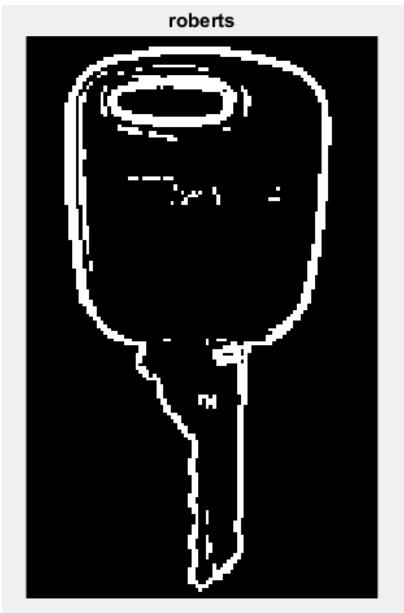
Figure 9
File Edit View Insert Tools Desktop Window Help
原图
roberts
prewitt
sobel
laplacian
marr1
marr
canny

边缘连接



图二：single_key.png

Roberts 算子



Prewitt 算子

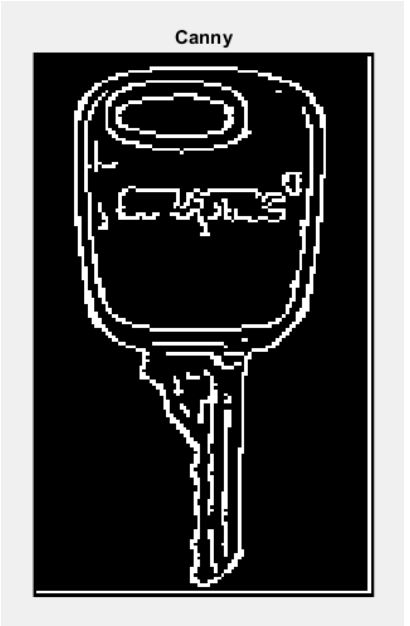
Sobel 算子

Laplacian 算子

非自己实现的 Marr 算子

自己实现的 Marr 算子

Canny 算子


Canny

对比图

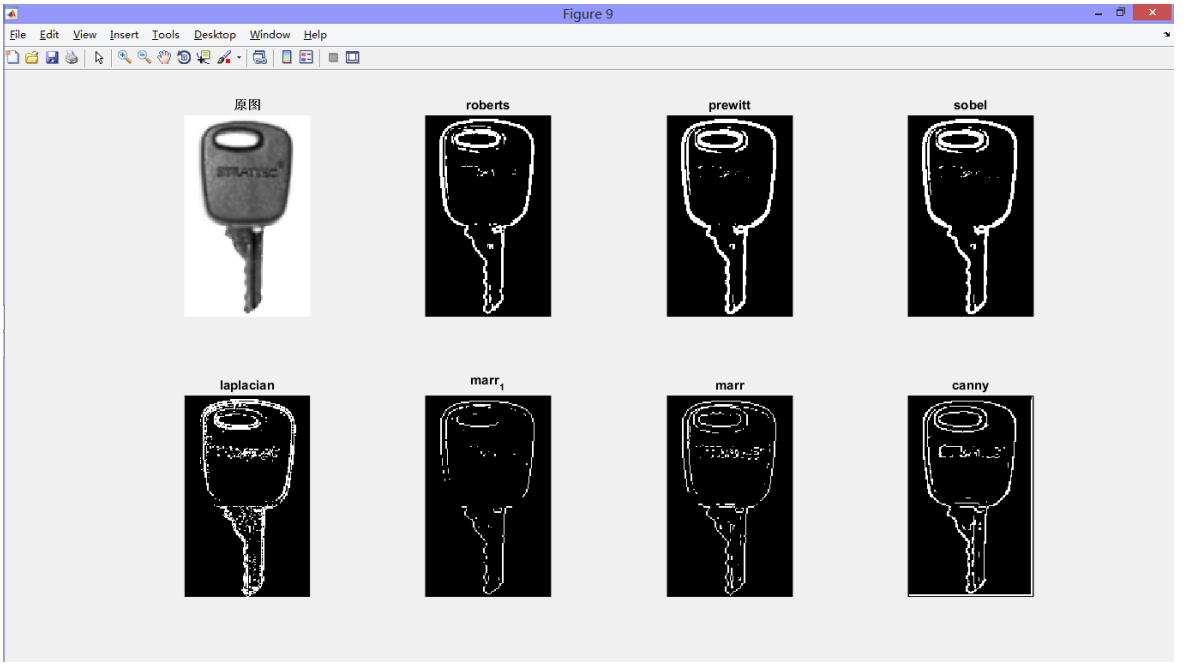
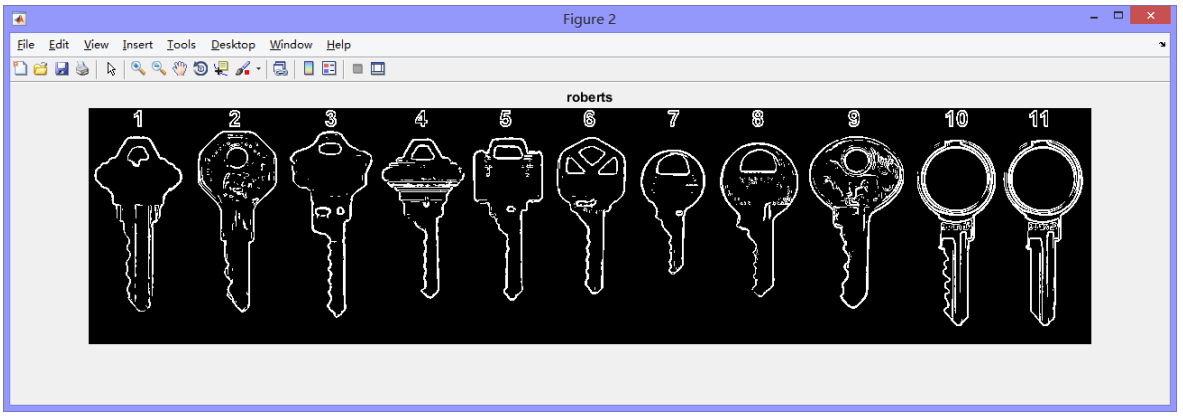


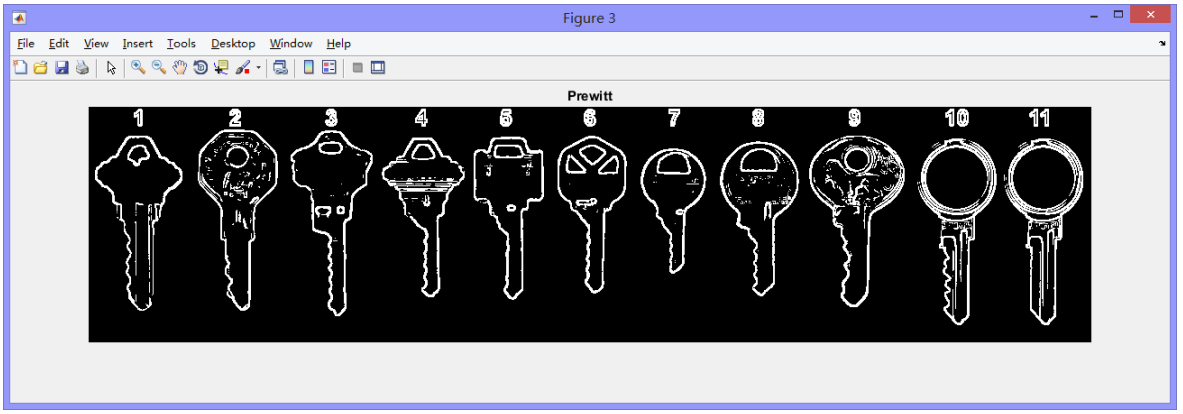
Figure 9
File Edit View Insert Tools Desktop Window Help
原图
roberts
prewitt
sobel
laplacian
marr1
marr
canny

图三：keys_set.png

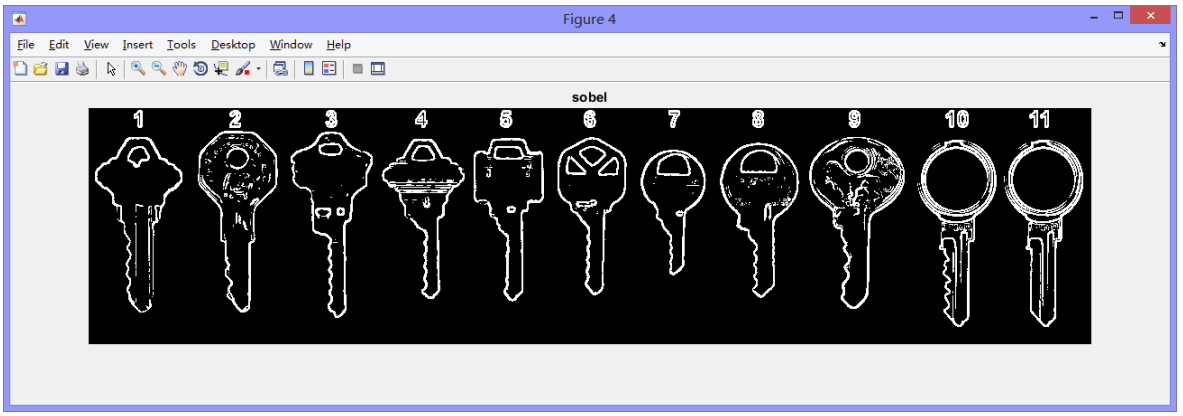
Roberts 算子



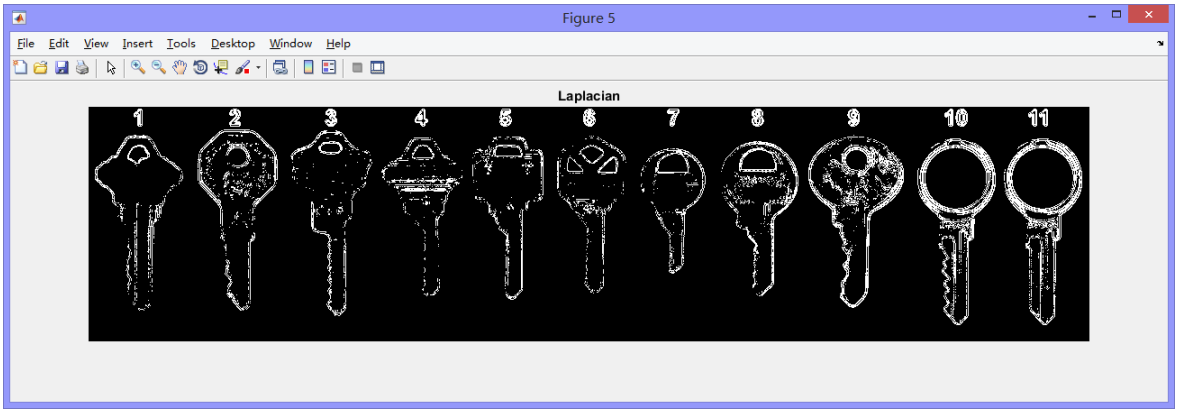
Prewitt 算子



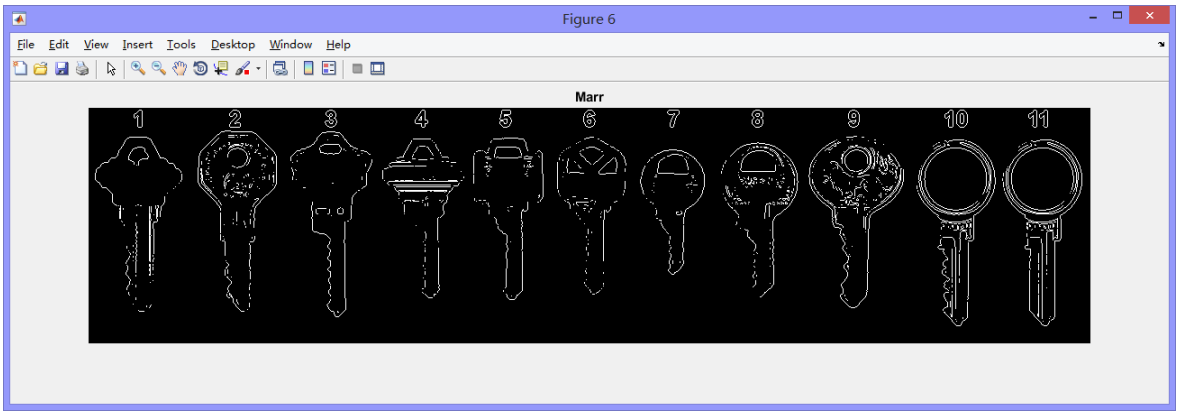
Sobel 算子



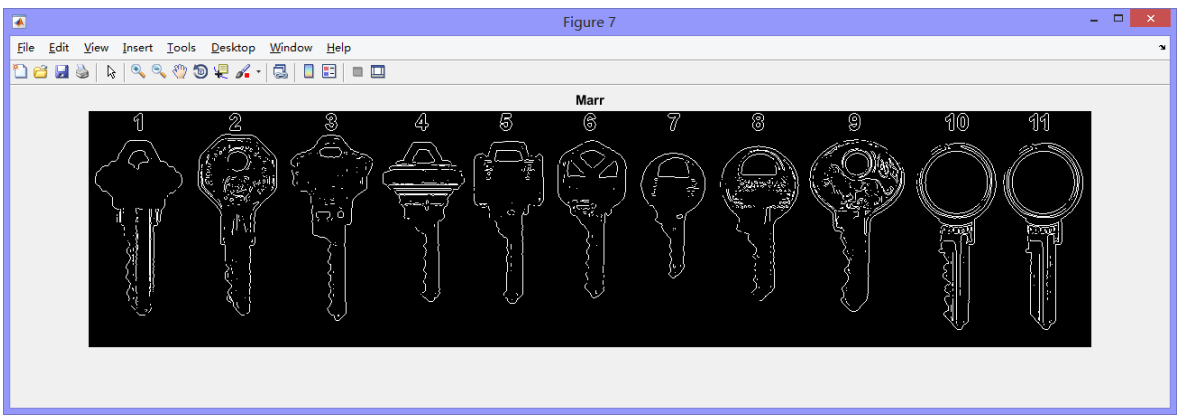
Laplacian 算子



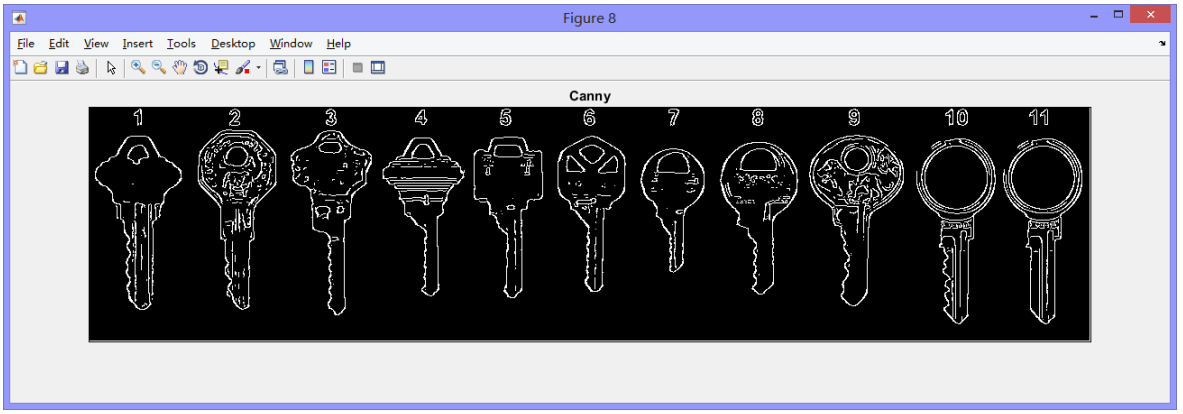
非自己实现的 Marr 算子



自己实现的 Marr 算子

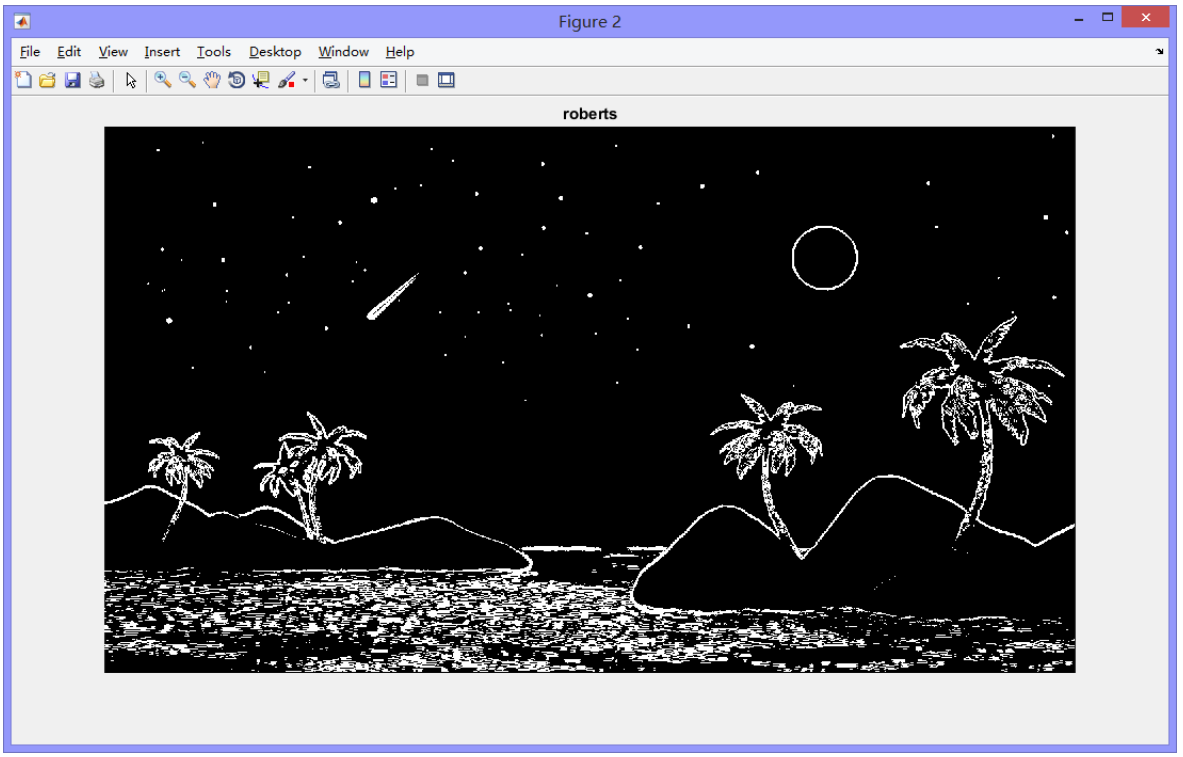


Canny 算子

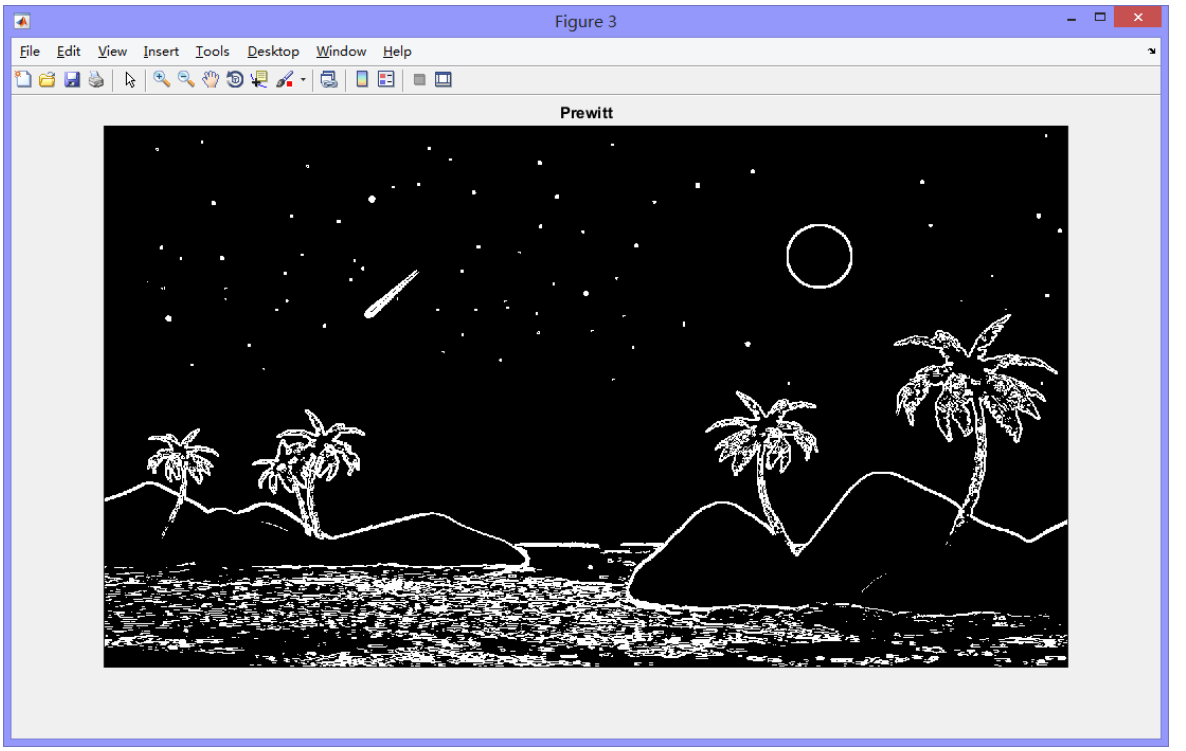


图四：moon.jpg

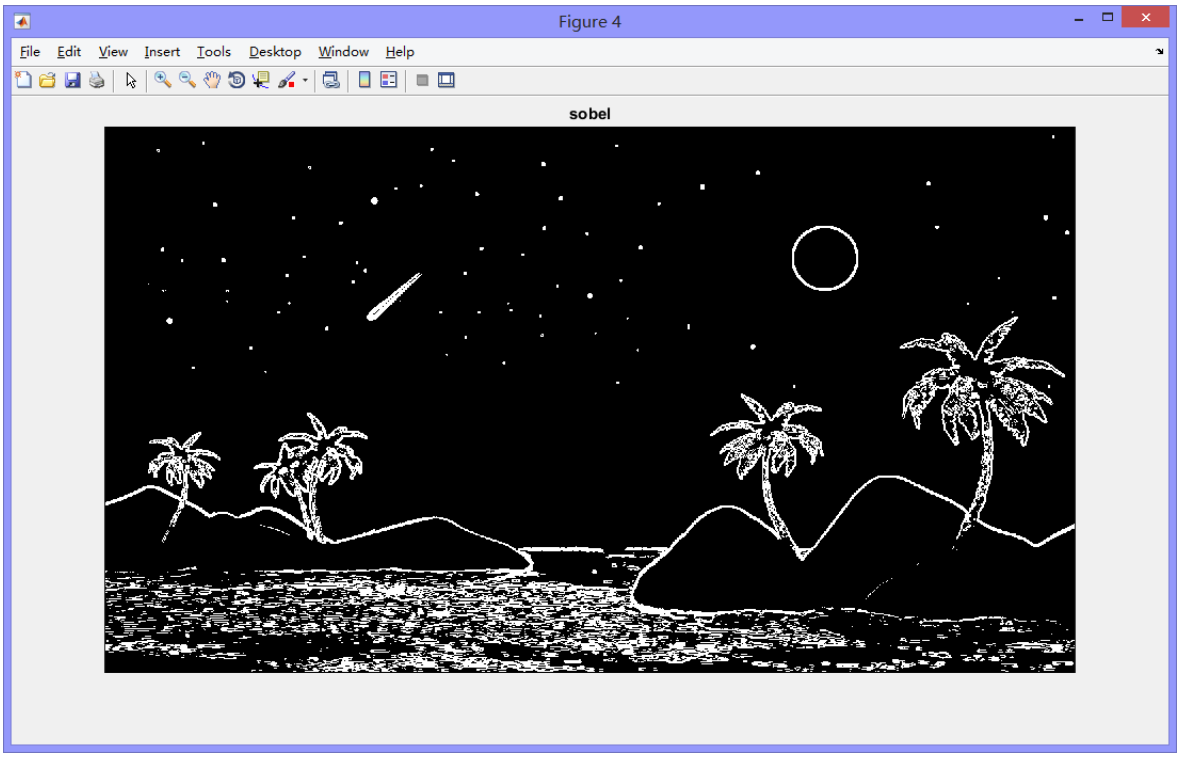
Roberts 算子



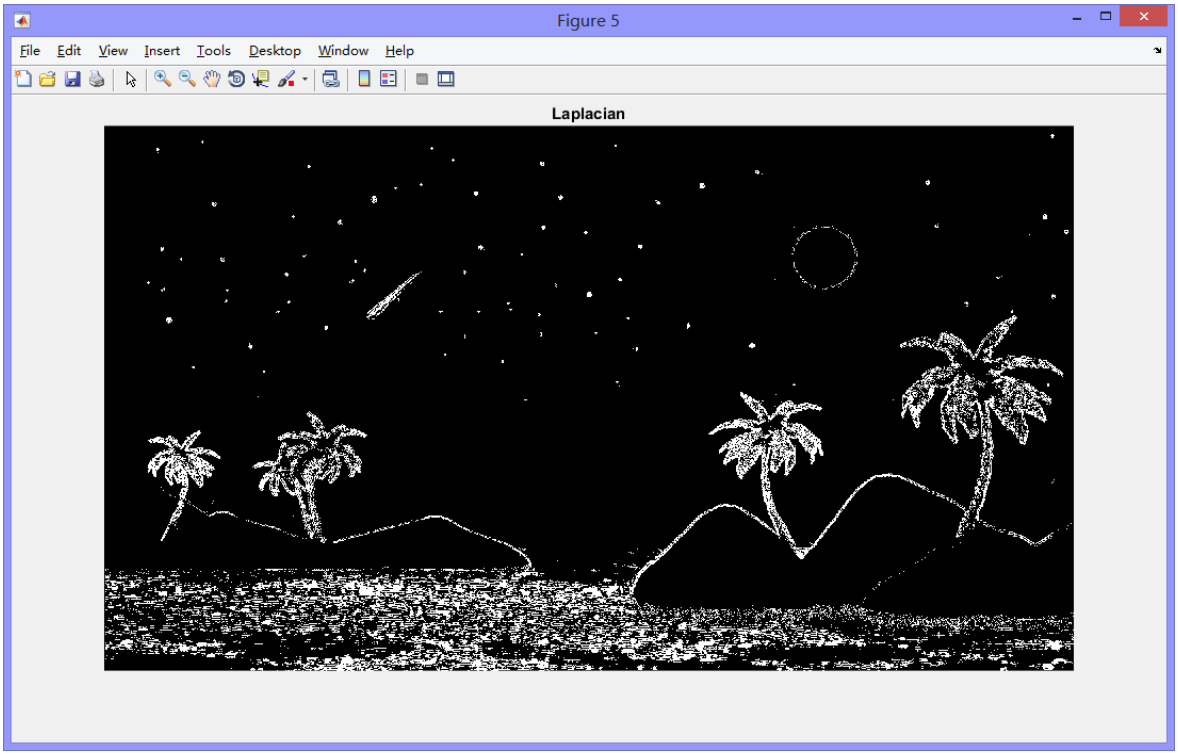
Prewitt 算子



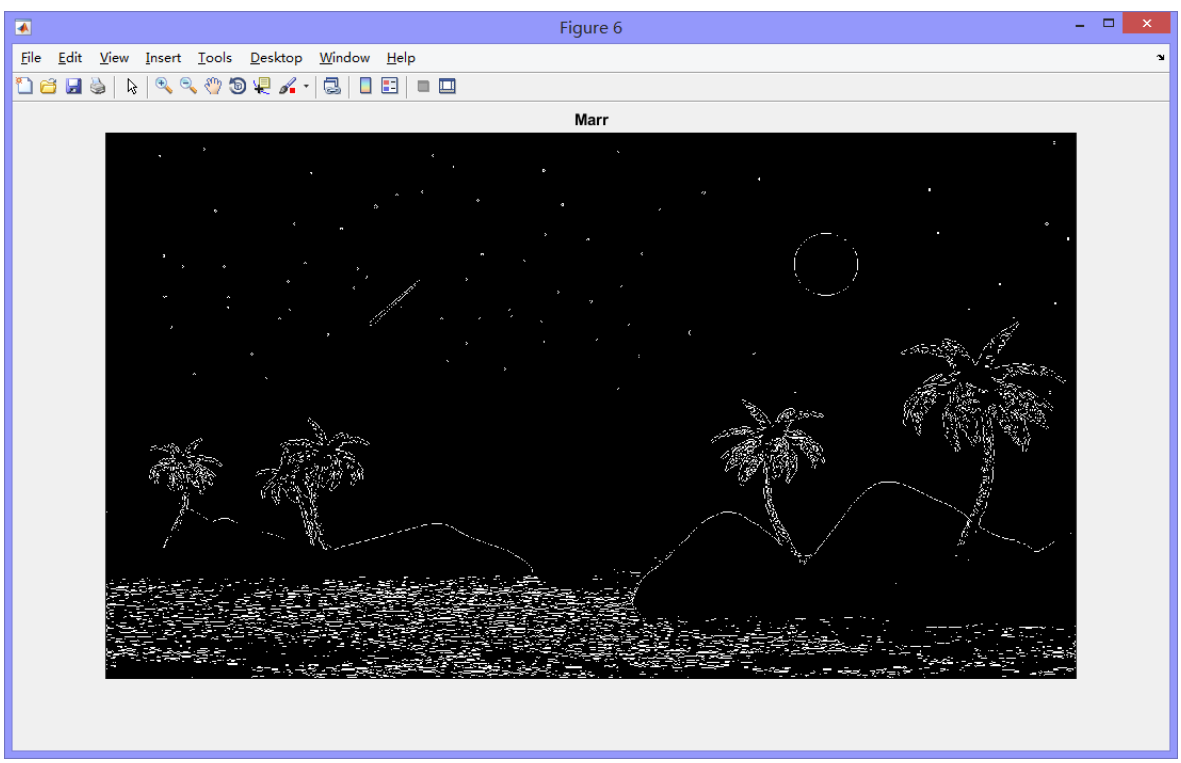
Sobel 算子



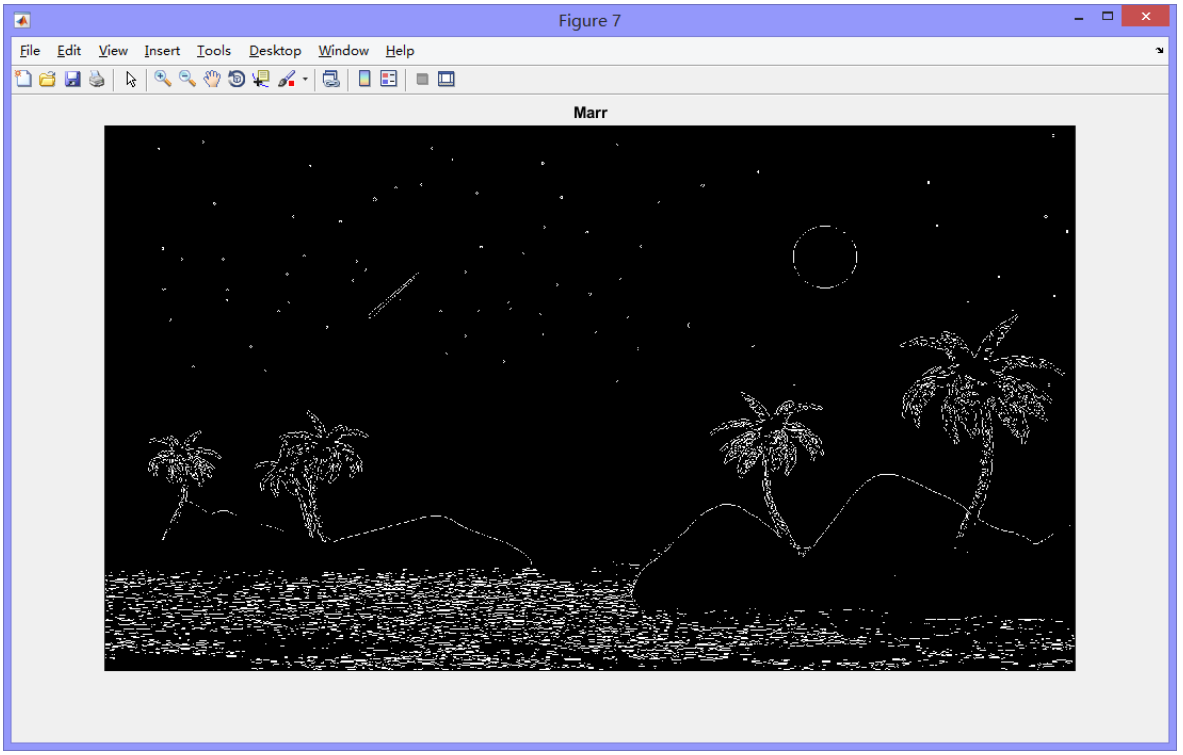
Laplacian 算子



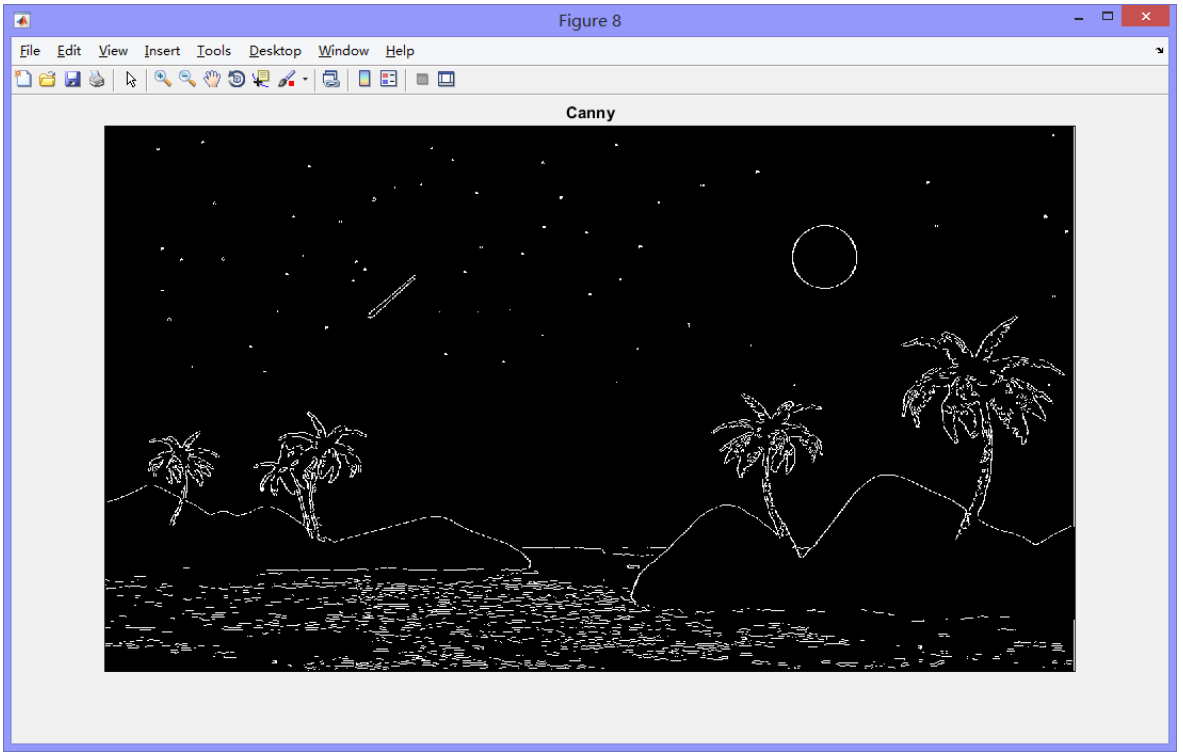
非自己实现的 Marr 算子



自己实现的 Marr 算子

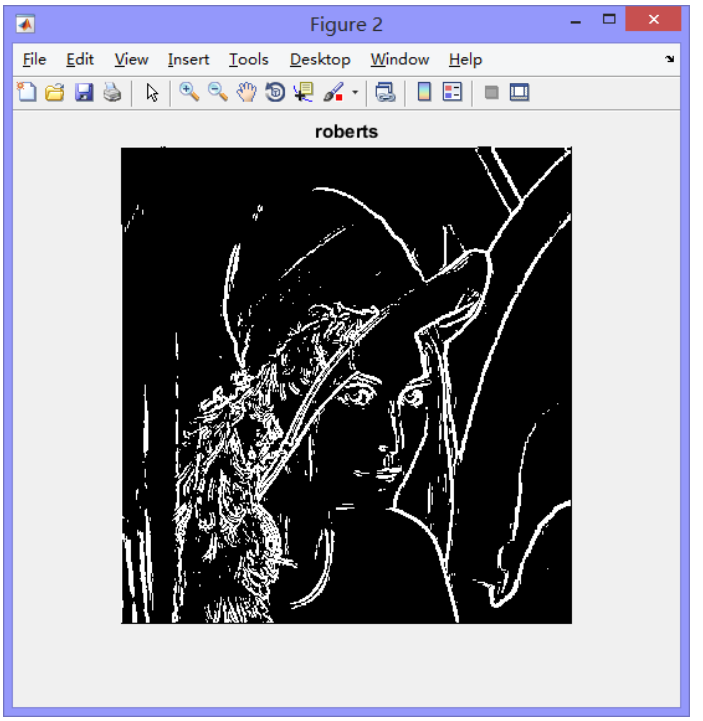


Canny 算子

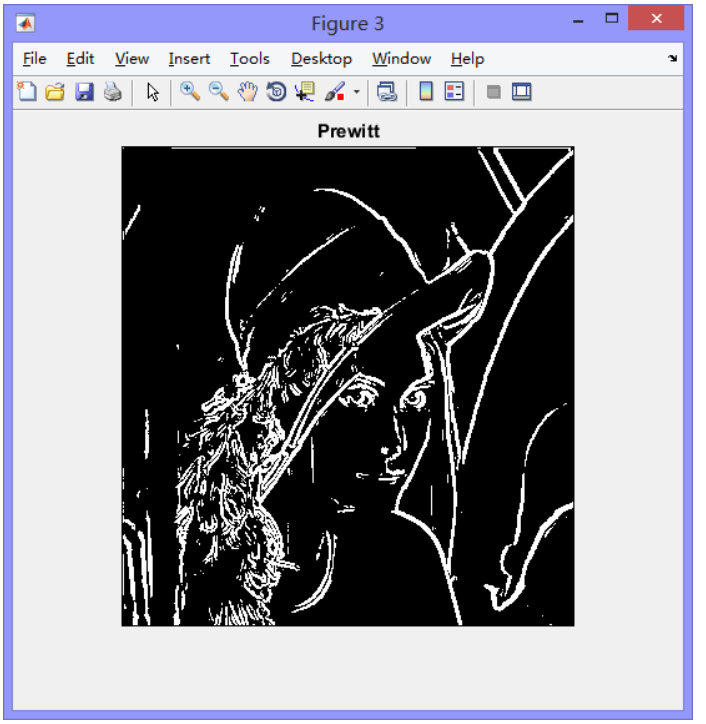


图五：test5.jpg

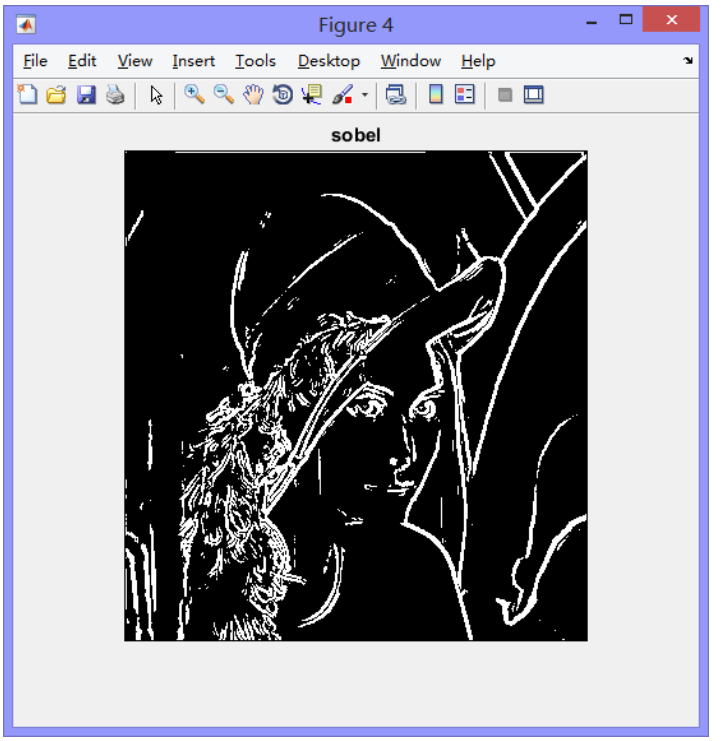
Roberts 算子



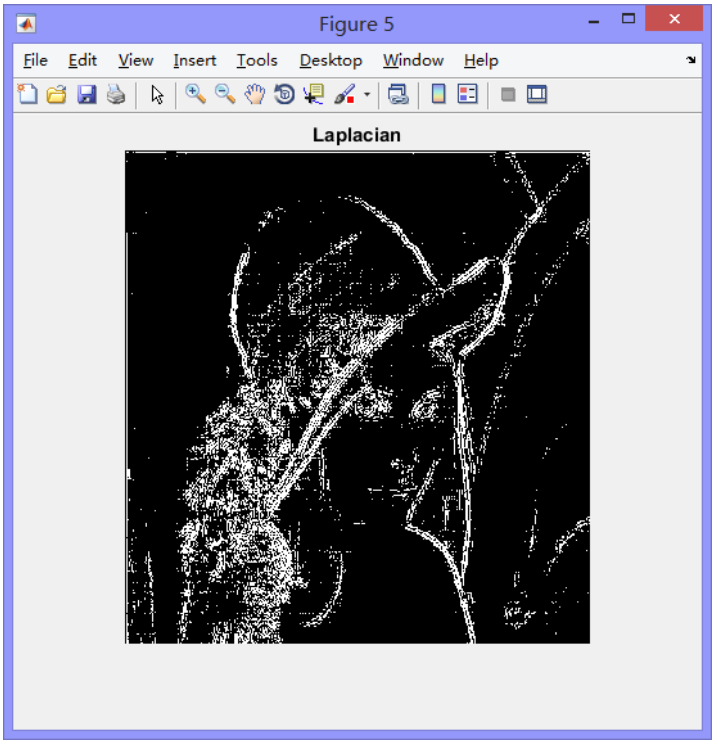
Prewitt 算子



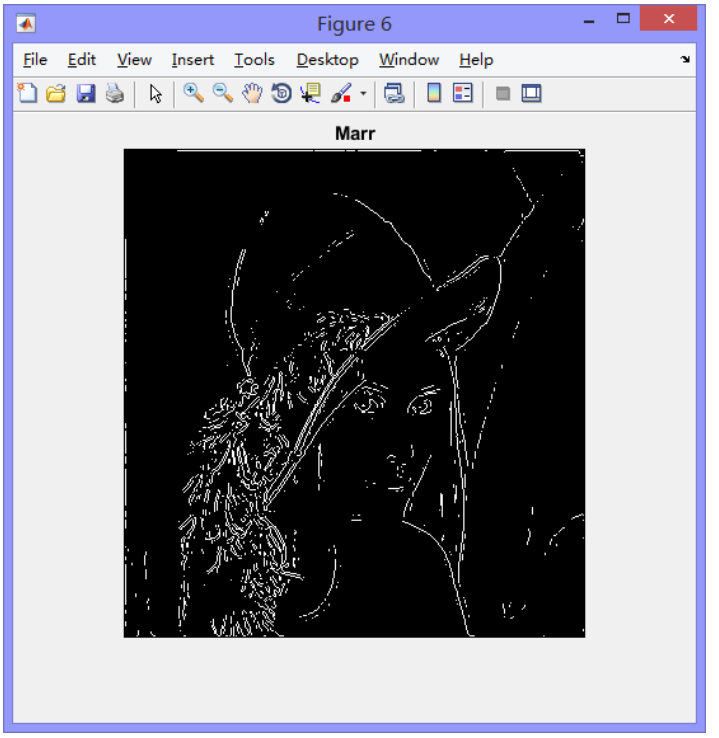
Sobel 算子



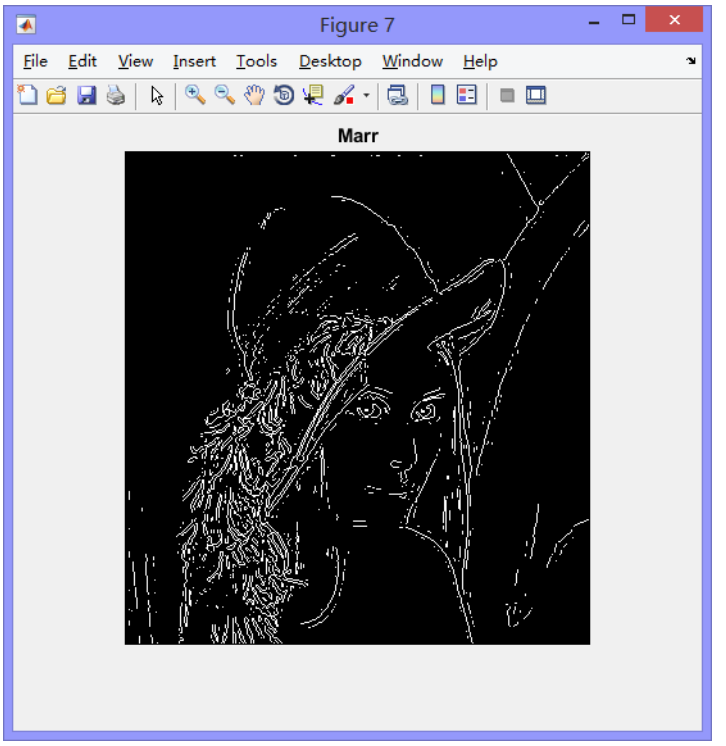
Laplacian 算子



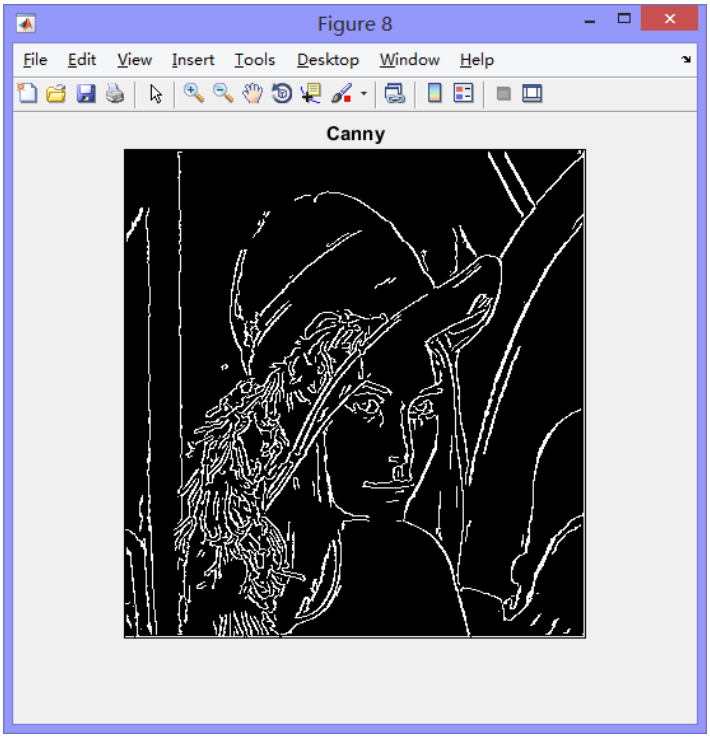
非自己实现的 Marr 算子



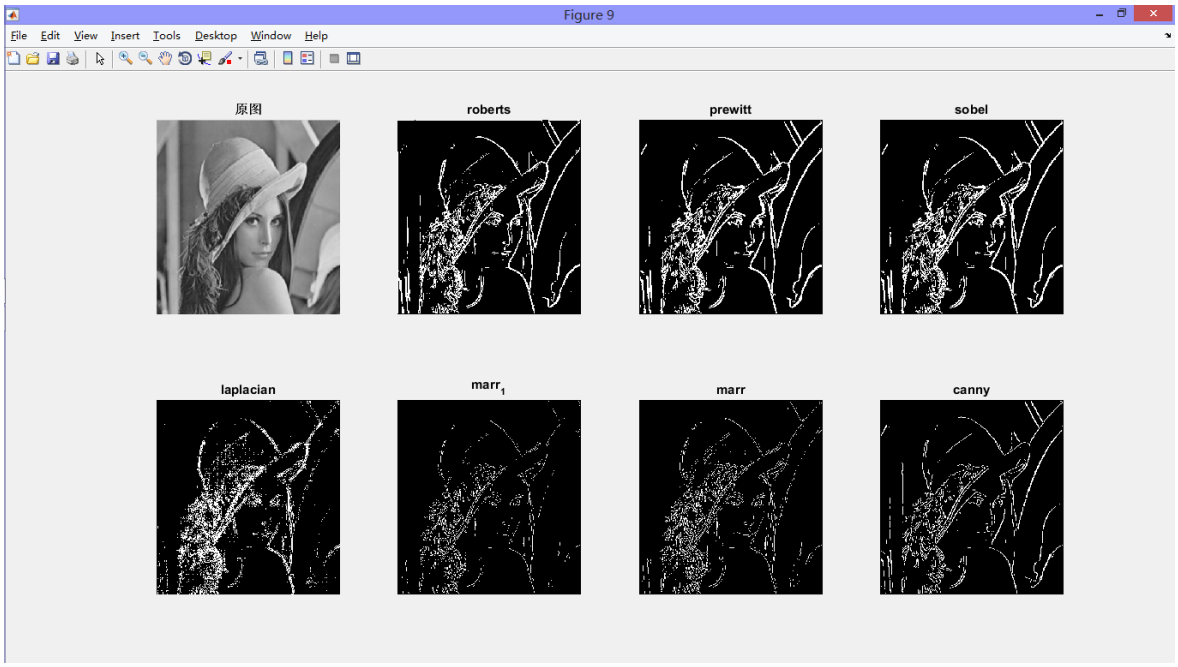
自己实现的 Marr 算子



Canny 算子



对比图



注：图 3 和图 4 对比图太小，无法看清，所以没有展示。

2.3 结果分析

从结果图中，可以看出来：

1) Roberts 算子对于边界陡峭且噪声比较小的图像检测效果比较好，对噪声比较敏感。
2) Perwitt 算子考虑更多领域，对噪声有抑制作用，较 Roberts 算子减少了对噪声的影响。
3) Sobel 算子较前两种算子更为精细，但 Sobel 算子并没有将图像的主题与背景严格地区分开来。
4) 直观的看，前三种算法在测试的 5 张图中基本差别不大，效果差不太多。
5) Laplacian 算子不依赖边缘方向，具有旋转不变性，对噪声十分敏感。
6) Marr 算子将高斯滤波和 Laplacian 算子结合在一起，减小了噪声的影响，但是参数的选取十分重要，不合适的参数可能因为过度平滑而丢失边缘。
7) Canny 算子效果基本是上述算子中最优的，有效的抑制了噪声及假边缘，但是对于低阈值的选择也需要多次尝试才能得出较优结果，否则容易丢失边缘。
8) 前四种算子得出的边缘较粗，对边缘定位的精度不是很高，后三种算子得出的边缘较细。
9) Marr 算子能比 Canny 算子检测出更多的细节，但 Canny 算子检测出的真正边缘的效果更好。
10) Canny 算子较其它算子运行时间要长很多。

3 实验中遇到的问题和心得体会

1) 前四种算法的实现均较为简单，可以直接根据课件上的公式实现。
2) 一开始虽然理解了 Marr 算子和 Canny 算子的原理与流程，但是不太能独立实现，一方面的原因是对 matlab 还不够熟悉，另一方面的原因是自身编程能力有所欠缺，所以在网上查找了不少资料，其中代码中第一个 Marr 算子的实现便是来自网络，经过理解，加上自己的认知对 Marr 算法进行了实现，在图 2 和图 3 中的效果比从网络上获得的代码效果要好一点。
3) Canny 算子比 Marr 算子更为复杂，所以代码基本是边参考网络上的代码边自己实现的，进行了多次尝试最后选择了效果最好的一种实现。
4) 在边缘连接函数中，不是很明白具体要做什么，所以调用了库函数对边缘进行细化然后简单的进行了边缘跟踪，理论上我认为还应该对断开的边缘进行连接，但是尝试后没有成功，也没有想到比较好的算法，所以未能实现。
5) 一开始未注意到边缘连接函数输出结果实质上不是一个矩阵而是若干点对，所以一直无法在 background 上输出结果。
6) 可以输出结果后没有在寻找完一个方向后写 continue，所以在数据结构为队列的情况下，存储点的顺序并不是顺时针或者逆时针，导致最终输出结果为实心的图形，经过检查发现 plot 函数的参数是画线而不是画点，加入 continue 后能保证是按照逆时针追踪边界，所以可以按顺序输出。
7) 一开始没有调用库函数进行边缘细化，但是在对各个圆进行边缘连接时总会在半中间走向反方向，没有想到别的解决办法，所以调用了库函数进行边缘细化。
8) 在实现功能之前应该多理解代码框架，弄清接口的类型结构等，否则会走不少弯路。

4 参考文献

1) https://wenku.baidu.com/view/ae860924bcd126fff7050b0a.html
2) http://blog.csdn.net/u012808193/article/details/45722283
3) http://blog.csdn.net/humanking7/article/details/46606791
4) http://blog.csdn.net/lk274857347/article/details/51919357
5) 课件内容