Problem 9

Digital Image Processing

肖逸群 116033910112

Image segmentation

问题描述

> (a). Develop a program to implement the Roberts, Prewitt, Sobel, the Marr-Hildreth and the Canny edge detectors. Use the image 'building.tif' to test your detectors. (For technique details of Marr-Hildreth and Canny, please refer to pp.736-747 (3rd edition, Gonzalez DIP) or MH-Canny.pdf at the same address of the slides.)
> (b). Develop a program to implement the Otsu's method of thresholding segmentation, and compare the results with the global thresholding method using test image 'polymersomes.tif. (For technique details, please refer to pp.763-770 (3rd edition, Gonzalez DIP), or Otsu.pdf at the same ftp address of slides.)

算法思想

本题要求我们以 Roberts, Prewitt, Sobel, Marr - Hildreth, Canny 算子来对图像进行边缘检测。
各个算法的定义如下：

- Roberts 算子：
  Roberts 算子的近似偏导定义如下：

$$g_x = z_9 - z_5$$
$$g_y = z_8 - z_6$$

- Prewitt 算子：
  Prewitt 算子的近似偏导定义如下：

$$g_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$
$$g_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

- Sobel 算子：
  Sobel 算子的近似偏导定义如下：

$$g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$
$$g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

  在本题中，我采用了直接以偏导 gx, gy 相加的方式来得到近似的边缘图片。
- Marr - Hildreth 算子：
  对于 MH 算子，需要先进行高斯变换和拉普拉斯变换，LoG 的定义如下：

$$LoG(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

在对原始图像进行 LoG 变换后，需要先对得到的变换矩阵进行归一化。
然后对原始图片与变换后的矩阵进行卷积，求其差值，得到差值图像。
接着求出差值图像通过 0 的点，在本题中我的思路是将该点与四周四个方向的点分别求和，假如该点为 0，则必然存在一个方向上的和大于 0，且还同时有另一个方向上的和小于 0。
找出点后我进行标记，把为 0 的点当做边界。

- Canny 算子：
  Canny 算子跟之前介绍的 MH 算子一样，也属于先平滑后求导数的方法。
  Canny 算法的步骤如下：

  a. 去噪：对原始图像与高斯掩模做卷积，得到模糊图像。这样做可以达到去噪的效果。
  b. 用一阶偏导的有限差分来计算梯度的幅值和方向。
  c. 对梯度幅值进行非极大值抑制，因为仅仅得到全局的梯度还是没办法确定边缘，故需要保留局部梯度最大的点，并且抑制非极大值。
  d. 用双阈值算法检测和连接边缘，双阈值法的定义为：对非极大值抑制图象作用两个阈值τ1和τ2，且2τ1≈τ2，从而可以得到两个阈值边缘图象N1［i, j］和N2［i，j］。由于N2［i，j］使用高阈值得到，因而含有很少的假边缘，但有间断(不闭合)。双阈值法要在N2［i，j］中把边缘连接成轮廓，当到达轮廓的端点时，该算法就在N1［i, j］的8邻点位置寻找可以连接到轮廓上的边缘，这样，算法不断地在N1［i,j］中收集边缘，直到将N2［i, j］连接起来为止。(在连接边缘的时候，用数组模拟队列的实现。以进行8-连通域搜索)

- 全局阈值法
  全局阈值法的思路如下：

  a. 为全局阈值选择一个初始估计值T(图像的平均灰度)。
  b. 用T分割图像。产生两组像素：G1有灰度值大于T的像素组成，G2有小于等于T像素组成。
  c. 计算G1和G2像素的平均灰度值m1和m2；
  d. 计算一个新的阈值:T = (m1 + m2) / 2;
  e. 重复步骤2和4,直到连续迭代中的T值间的差为零。

- Ostu 阈值法
  在本次实验中我使用了 Ostu 大律法，算法思路如下：
  一幅有depth个灰度级，根据每个灰度级t，可以将一幅图分为前景和背景。前景指所有灰度级低于等于t的像素点，背景指大于t的像素点。
  w0指前景像素个数; w1指背景像素个数; u0指前景加权平均，即

```
temp = 0;
for i = 1 : t
  temp = temp + i * hist(i)
end
u0 = temp / w0;
u = u0 * w0 + u1 * w1
```

  其中w0=hist(0)+hist(1)+……+hist(t)
  hist指各个灰度级上的像素个数; u1对应是背景加权平均; u对应整幅图的加权平均
  大津法的结果指使得g最大的t值：
  g = w0 *(u0 - u)* ^ *2* + *w1* (u1 - u) ^ 2 = w0 *w1* (u1 - u0) ^ 2
  再以 g 为阈值进行边缘检测。

源码分析

- 实验过程如下：
  a. 读取原始图片并将其转为灰度图像。

b. 对原始图片分别使用 Roberts, Prewitt, Sobel 算子进行边缘检测，得到边缘图像，显示各种方法处理得到的边缘图像。
c. 对原始图片使用 MH 算子进行边缘检测，实验过程在算法思想里有涉及，得到边缘图像并展示。
d. 对原始图片使用 Canny 算子进行边缘检测，流程同算法思想，得到边缘图像并展示。
e. 分别使用全局阈值和 Ostu 阈值进行边缘检测，得到边缘图像并展示。

```
def main():
    originalImage = np.array(Image.open('./resource/building.tif'))

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    robertsGxImage, robertsGyImage, robertsImage = segmentAlgorithm.Roberts(originalImag
e)
    plt.subplot(2, 2, 2)
    plt.imshow(robertsGxImage, cmap = plt.get_cmap('gray'))
    plt.title('Roberts gx')

    plt.subplot(2, 2, 3)
    plt.imshow(robertsGyImage, cmap = plt.get_cmap('gray'))
    plt.title('Roberts gy')

    plt.subplot(2, 2, 4)
    plt.imshow(robertsImage, cmap = plt.get_cmap('gray'))
    plt.title('Roberts')

    plt.show()

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    prewittGxImage, prewittGyImage, prewittImage = segmentAlgorithm.Prewitt(originalImag
e)
    plt.subplot(2, 2, 2)
    plt.imshow(prewittGxImage, cmap = plt.get_cmap('gray'))
    plt.title('Prewitt gx')

    plt.subplot(2, 2, 3)
    plt.imshow(prewittGyImage, cmap = plt.get_cmap('gray'))
    plt.title('Prewitt gy')

    plt.subplot(2, 2, 4)
    plt.imshow(prewittImage, cmap = plt.get_cmap('gray'))
    plt.title('Prewitt')

    plt.show()

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    sobelGxImage, sobelGyImage, sobelImage = segmentAlgorithm.Sobel(originalImage)
    plt.subplot(2, 2, 2)
    plt.imshow(sobelGxImage, cmap = plt.get_cmap('gray'))
    plt.title('Sobel gx')

    plt.subplot(2, 2, 3)
```

```
    plt.imshow(sobelGyImage, cmap = plt.get_cmap('gray'))
    plt.title('Sobel gy')

    plt.subplot(2, 2, 4)
    plt.imshow(sobelImage, cmap = plt.get_cmap('gray'))
    plt.title('Sobel')

    plt.show()

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    plt.subplot(2, 2, 2)
    LoGImage, marrHildrethImage1, marrHildrethImage2 = marrHildrethAlgorithm.MarrHildreth(originalImage)
    plt.imshow(LoGImage, cmap = plt.get_cmap('gray'))
    plt.title('LoG Without Threshold')

    plt.subplot(2, 2, 3)
    plt.imshow(marrHildrethImage1, cmap = plt.get_cmap('gray'))
    plt.title('Marr-Hildreth Threshold = 0')

    plt.subplot(2, 2, 4)
    plt.imshow(marrHildrethImage2, cmap = plt.get_cmap('gray'))
    plt.title('Marr-Hildreth Threshold = 15% Max(LoGImage)')

    plt.show()

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    plt.subplot(2, 2, 2)
    smoothImage, cannyImage = cannyAlgorithm.canny(Image.open('./resource/building.tif').convert('L'))
    plt.imshow(smoothImage, cmap = plt.get_cmap('gray'))
    plt.title('Smooth')

    plt.subplot(2, 2, 3)
    plt.imshow(marrHildrethImage2, cmap = plt.get_cmap('gray'))
    plt.title('Marr-Hildreth Threshold = 15% Max(LoGImage)')

    plt.subplot(2, 2, 4)
    plt.imshow(cannyImage, cmap = plt.get_cmap('gray'))
    plt.title('Canny')

    plt.show()

    originalImage = np.array(Image.open('./resource/polymersomes.tif'))

    plt.subplot(1, 3, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    plt.subplot(1, 3, 2)
    globalSegmentImage = thresholdSegmentAlgorithm.globalThresholdSegment(originalImage)
    plt.imshow(globalSegmentImage, cmap = plt.get_cmap('gray'))
    plt.title('Global Threshold Segment')

    plt.subplot(1, 3, 3)
```

```
    otsuSegmentImage = thresholdSegmentAlgorithm.otsuThresholdSegment(originalImage)
    plt.imshow(otsuSegmentImage, cmap = plt.get_cmap('gray'))
    plt.title('Otsu Threshold Segment')

    plt.show()
```

- 图像预处理，添加一层边缘

```
def imagePreprocessing(input):
    row, col = input.shape
    output = np.zeros((row + 2, col + 2))
    for i in range(1, row + 1):
        for j in range(1, col + 1):
            output[i, j] = input[i - 1, j - 1]
    for i in range(row + 2):
        output[i, 0] = output[i, 1]
        output[i, col + 1] = output[i, col]
    for j in range(col + 2):
        output[0, j] = output[1, j]
        output[row + 1, j] = output[row, j]
    return output
```

- 图像各个方向平移，以便进行求偏导的计算

```
def upperShift(originalImage):
    U = np.zeros(originalImage.shape)
    U[: U.shape[0] - 1, :] = originalImage[1 : originalImage.shape[0], :]
    U[U.shape[0] - 1, :] = U[U.shape[0] - 2, :]
    return U

def downShift(originalImage):
    D = np.zeros(originalImage.shape)
    D[1 :, :] = originalImage[: originalImage.shape[0] - 1, :]
    D[0, :] = D[1, :]
    return D

def leftShift(originalImage):
    L = np.zeros(originalImage.shape)
    L[:, : L.shape[1] - 1] = originalImage[:, 1 : originalImage.shape[1]]
    L[:, L.shape[1] - 1] = L[:, L.shape[1] - 2]
    return L

def rightShift(originalImage):
    R = np.zeros(originalImage.shape)
    R[:, 1 :] = originalImage[:, : originalImage.shape[1] - 1]
    R[:, 0] = R[:, 1]
    return R
```

- Roberts, Prewitt, Sobel 算法

```
def Roberts(originalImage):
    U = upperShift(originalImage)
    L = leftShift(originalImage)
    UL = leftShift(U)

    gx = np.abs(UL - originalImage)
    gy = np.abs(U - L)
    M = gx + gy

    return gx, gy, M

```

```
def Prewitt(originalImage):
    U = upperShift(originalImage)
    D = downShift(originalImage)
    L = leftShift(originalImage)
    R = rightShift(originalImage)

    UL = leftShift(U)
    UR = rightShift(U)
    DL = leftShift(D)
    DR = rightShift(D)

    gx = np.abs(UR + U + UL - DR - D - DL)
    gy = np.abs(DL + L + UL - DR - R - UR)
    M = gx + gy

    return gx, gy, M

def Sobel(originalImage):
    U = upperShift(originalImage)
    D = downShift(originalImage)
    L = leftShift(originalImage)
    R = rightShift(originalImage)

    UL = leftShift(U)
    UR = rightShift(U)
    DL = leftShift(D)
    DR = rightShift(D)

    gx = np.abs(UR + 2 * U + UL - DR - 2 * D - DL)
    gy = np.abs(DL + 2 * L + UL - DR - 2 * R - UR)
    M = gx + gy

    return gx, gy, M
```

- MH 算法

```
def LoG(x, y, sigma):
    t = np.power(x, 2) + np.power(y, 2)
    res = (t - np.power(sigma, 2) * 1.0) / np.power(sigma, 4) * np.exp(- t / (2 * np.pow
er(sigma, 2)))
    return res

def LoGThreshold(inputImage, threshold):
    m, n = inputImage.shape
    outputImage = inputImage.copy()
    for i in range(1, m - 1):
        for j in range(1, n - 1):
            tmpList = []
            tmpList.append(np.sum(inputImage[i - 1 : i, j - 1 : j]))
            tmpList.append(np.sum(inputImage[i - 1 : i, j : j + 1]))
            tmpList.append(np.sum(inputImage[i : i + 1, j - 1 : j]))
            tmpList.append(np.sum(inputImage[i : i + 1, j : j + 1]))
            maxValue = max(tmpList)
            minValue = min(tmpList)
            if maxValue > threshold and minValue < threshold:
                outputImage[i, j] = 255
    return outputImage

def MarrHildreth(originalImage):
    maxPix = np.max(originalImage)
    minPix = np.min(originalImage)
```

```
    diff = maxPix - minPix
    originalImage = (originalImage - minPix) * 1.0 / diff
    n = 25
    sigma = 4
    w = np.zeros((n, n))
    center = (n - 1) / 2
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            y = i - center
            x = j - center
            w[i - 1, j - 1] = LoG(x, y, sigma)
    w = w * 1.0 / np.sum(w)
    LoGImage = originalImage - signal.convolve2d(originalImage, w, mode = 'same')

    marrHildrethImage1 = LoGThreshold(LoGImage, 0)
    threshold = 0.15 * np.max(LoGImage)
    marrHildrethImage2 = LoGThreshold(LoGImage, threshold)
    return LoGImage, marrHildrethImage1, marrHildrethImage2
```

- Canny 算法

```
def canny(img):
    sigma = 4
    imgdata = numpy.array(img, dtype = float)
    gaussianImage = ndi.filters.gaussian_filter(imgdata, sigma)

    emptyImage = Image.new('L', img.size)
    gradientX = numpy.array(emptyImage, dtype = float)
    gradientY = numpy.array(emptyImage, dtype = float)

    sobelX = [[-1,0,1],
              [-2,0,2],
              [-1,0,1]]
    sobelY = [[-1,-2,-1],
              [0,0,0],
              [1,2,1]]

    m = img.size[1]
    n = img.size[0]

    for x in range(1, m - 1):
        for y in range(1, n - 1):
            px = (sobelX[0][0] * gaussianImage[x - 1][y - 1]) + (sobelX[0][1] * gaussianImage[x][y - 1]) + \
                 (sobelX[0][2] * gaussianImage[x + 1][y - 1]) + (sobelX[1][0] * gaussianImage[x - 1][y]) + \
                 (sobelX[1][1] * gaussianImage[x][y]) + (sobelX[1][2] * gaussianImage[x + 1][y]) + \
                 (sobelX[2][0] * gaussianImage[x - 1][y + 1]) + (sobelX[2][1] * gaussianImage[x][y + 1]) + \
                 (sobelX[2][2] * gaussianImage[x + 1][y + 1])

            py = (sobelY[0][0] * gaussianImage[x - 1][y - 1]) + (sobelY[0][1] * gaussianImage[x][y - 1]) + \
                 (sobelY[0][2] * gaussianImage[x + 1][y - 1]) + (sobelY[1][0] * gaussianImage[x - 1][y]) + \
                 (sobelY[1][1] * gaussianImage[x][y]) + (sobelY[1][2] * gaussianImage[x + 1][y]) + \
                 (sobelY[2][0] * gaussianImage[x - 1][y + 1]) + (sobelY[2][1] * gaussianImage[x][y + 1]) + \
                 (sobelY[2][2] * gaussianImage[x + 1][y + 1])
```

```
            gradientX[x][y] = px
            gradientY[x][y] = py

    gradientImageMag = sp.hypot(gradientX, gradientY)
    gradientImageDir = sp.arctan2(gradientY, gradientX)

    for x in range(m):
        for y in range(n):
            if (gradientImageDir[x][y] < 22.5 and gradientImageDir[x][y] >= 0) or \
            (gradientImageDir[x][y] >= 157.5 and gradientImageDir[x][y] < 202.5) or \
            (gradientImageDir[x][y] >= 337.5 and gradientImageDir[x][y] <= 360):
                gradientImageDir[x][y] = 0
            elif (gradientImageDir[x][y] >= 22.5 and gradientImageDir[x][y] < 67.5) or \
                (gradientImageDir[x][y] >= 202.5 and gradientImageDir[x][y] < 247.5):
                gradientImageDir[x][y] = 45
            elif (gradientImageDir[x][y] >= 67.5 and gradientImageDir[x][y] < 112.5)or \
                (gradientImageDir[x][y] >= 247.5 and gradientImageDir[x][y] < 292.5):
                gradientImageDir[x][y] = 90
            else:
                gradientImageDir[x][y] = 135


    magSup = gradientImageMag.copy()

    for x in range(1, m - 1):
        for y in range(1, n - 1):
            if gradientImageDir[x][y] == 0:
                if (gradientImageMag[x][y] <= gradientImageMag[x][y + 1]) or \
                    (gradientImageMag[x][y] <= gradientImageMag[x][y - 1]):
                    magSup[x][y] = 0
            elif gradientImageDir[x][y] == 45:
                if (gradientImageMag[x][y] <= gradientImageMag[x - 1][y + 1]) or \
                    (gradientImageMag[x][y] <= gradientImageMag[x + 1][y - 1]):
                    magSup[x][y] = 0
            elif gradientImageDir[x][y] == 90:
                if (gradientImageMag[x][y] <= gradientImageMag[x + 1][y]) or \
                    (gradientImageMag[x][y] <= gradientImageMag[x - 1][y]):
                    magSup[x][y] = 0
            else:
                if (gradientImageMag[x][y] <= gradientImageMag[x + 1][y + 1]) or \
                    (gradientImageMag[x][y] <= gradientImageMag[x - 1][y - 1]):
                    magSup[x][y] = 0

    maxGrayValue = numpy.max(magSup)
    th = 0.1 * maxGrayValue
    tl = 0.04 * maxGrayValue


    gnh = numpy.zeros((m, n))
    gnl = numpy.zeros((m, n))

    for x in range(m):
        for y in range(n):
            if magSup[x][y] >= th:
                gnh[x][y] = magSup[x][y]
            if magSup[x][y] >= tl:
                gnl[x][y] = magSup[x][y]
    gnl = gnl - gnh


    def traverse(i, j):
```

```
        x = [-1, 0, 1, -1, 1, -1, 0, 1]
        y = [-1, -1, -1, 0, 0, 1, 1, 1]
        for k in range(8):
            if gnh[i + x[k]][j + y[k]] == 0 and gnl[i + x[k]][j + y[k]] != 0:
                gnh[i + x[k]][j + y[k]] = 1
                traverse(i + x[k], j + y[k])

    for i in range(1, m - 1):
        for j in range(1, n - 1):
            if gnh[i][j]:
                gnh[i][j] = 1
                traverse(i, j)

    return gaussianImage, gnh
```

- 全局阈值算法

```
def globalThreshold(originalImage, deltLimit = 0):
    threshold = np.mean(originalImage)
    deltT = threshold
    while(deltT > deltLimit):
        g1 = originalImage[originalImage > threshold]
        g2 = originalImage[originalImage <= threshold]

        t1 = np.mean(g1)
        t2 = np.mean(g2)

        newThreshold = (t1 + t2) / 2
        deltT = np.abs(newThreshold - threshold)
        threshold = newThreshold

    return threshold

def globalThresholdSegment(originalImage):
    threshold = globalThreshold(originalImage)
    globalSegmentImage = originalImage.copy()
    globalSegmentImage[globalSegmentImage > threshold] = 255
    globalSegmentImage[globalSegmentImage <= threshold] = 0
    return globalSegmentImage
```

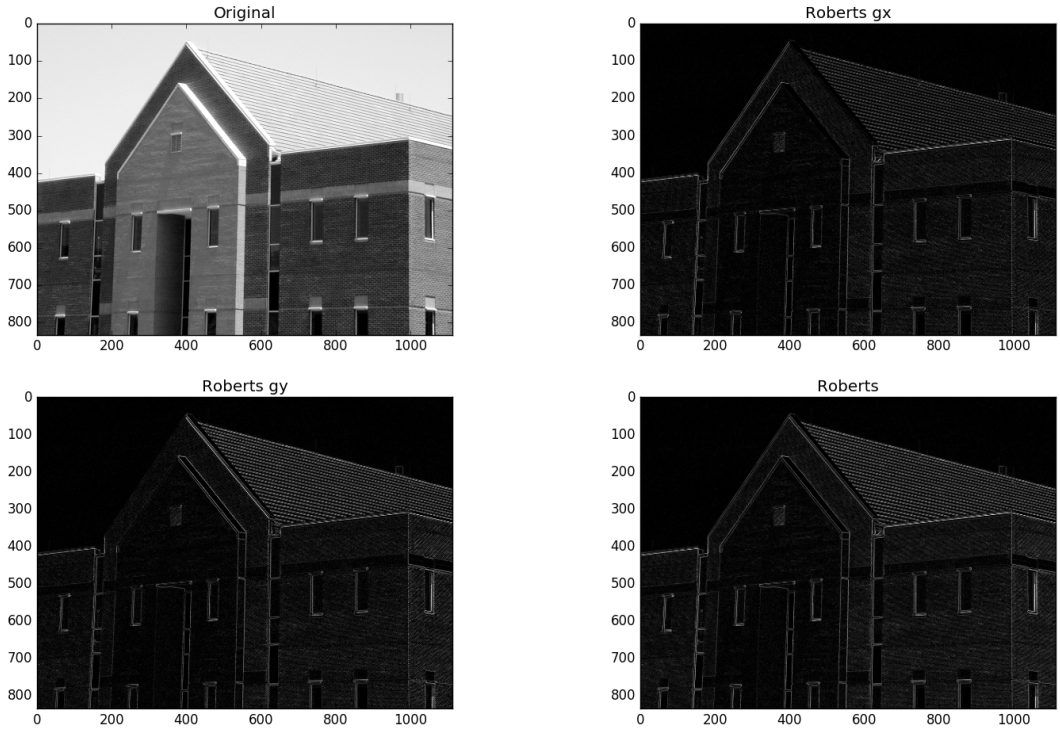
- Ostu 阈值法

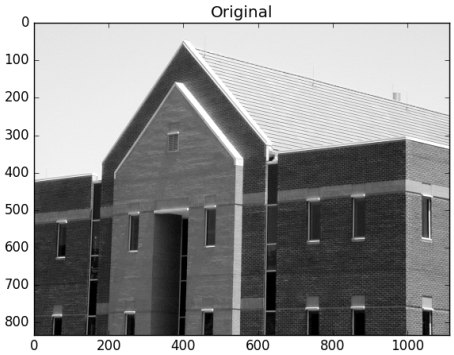
```
def getSigma(image, i):
    pNumH = np.sum(image.histogram()[: i + 1])
    pNumL = np.sum(image.histogram()[i + 1 :])

    mSumH = 0
    for j in range(1, i + 1):
        mSumH += j * image.histogram()[j]

    mSumL = 0
    for j in range(i + 1, 255):
        mSumL += j * image.histogram()[j]
    try:
        u0 = 1.0 * mSumH / pNumH
        u1 = 1.0 * mSumL / pNumL
        w0 = 1.0 * pNumH / (pNumH + pNumL)
    except:
        return 0

    w1 = 1.0 - w0
```

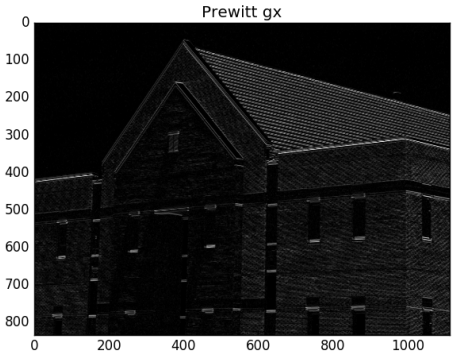
```
    u = (u0 - u1) ** 2
    sigma = w0 * w1 * u

    return sigma

def otsuThreshold(inputImage):
  threshold = 0
  gSigma = 0
  for i in range(1,255):
    sigma = getSigma(inputImage,i)
    if gSigma < sigma:
      gSigma = sigma
      threshold = i
  return threshold

def otsuThresholdSegment(originalImage):
    inputImage = Image.open('./resource/polymersomes.tif').convert("L")
    threshold = otsuThreshold(inputImage)
    otsuSegmentImage = originalImage.copy()
    otsuSegmentImage[otsuSegmentImage > threshold] = 255
    otsuSegmentImage[otsuSegmentImage <= threshold] = 0
    return otsuSegmentImage
```

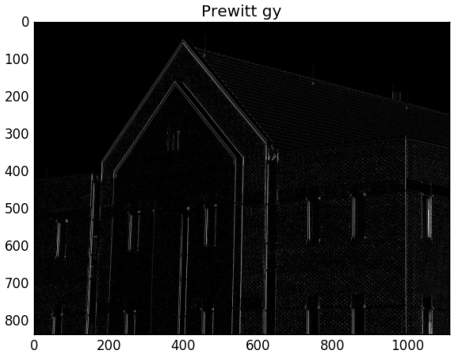
实验结果

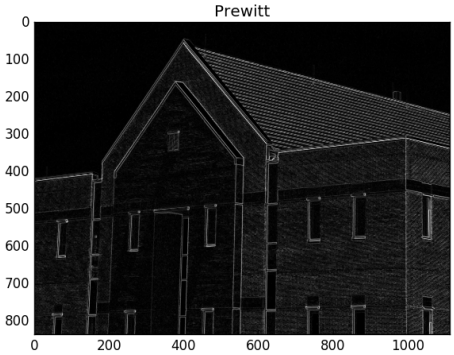
下列图片为不同边缘检测算法下的实验结果：

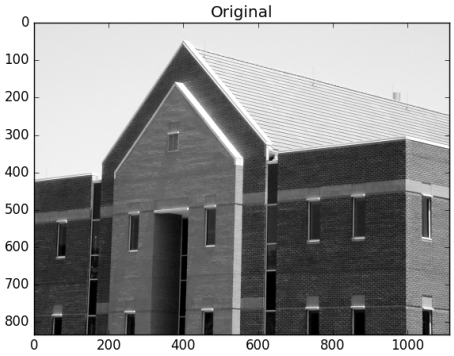


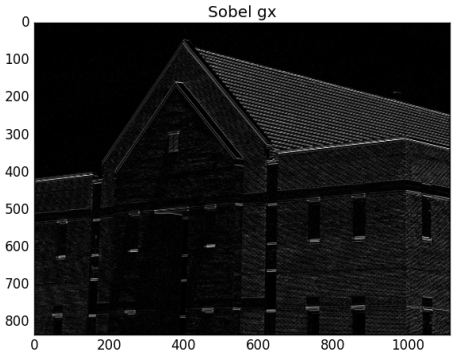

Original
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

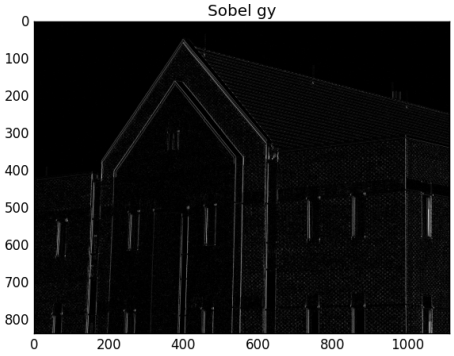

Prewitt gx
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

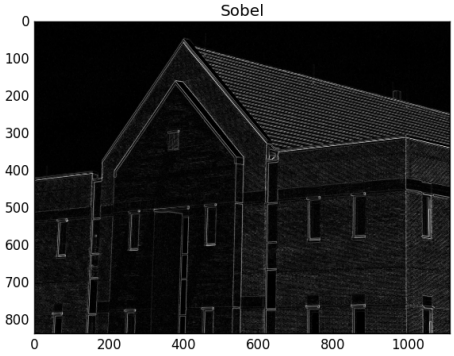

Prewitt gy
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

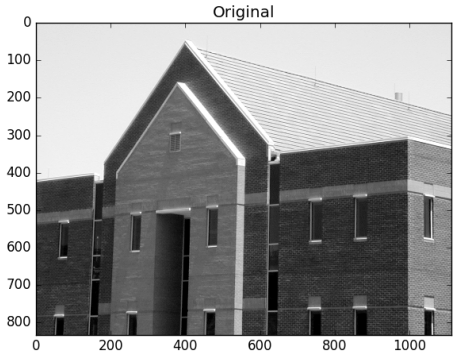

Prewitt
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

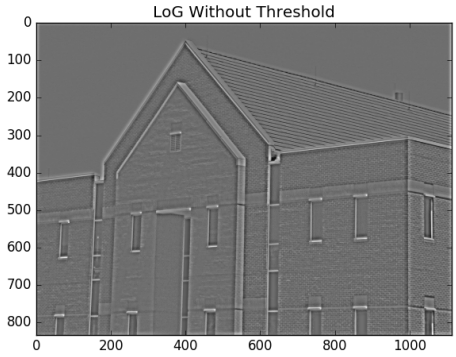

Original
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

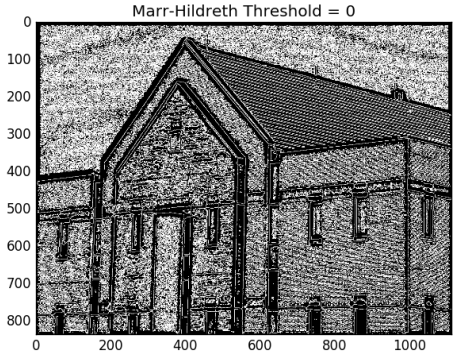

Sobel gx
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

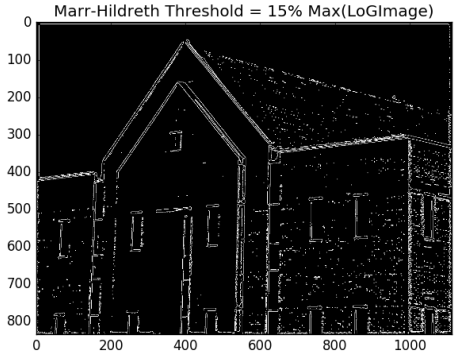

Sobel gy
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

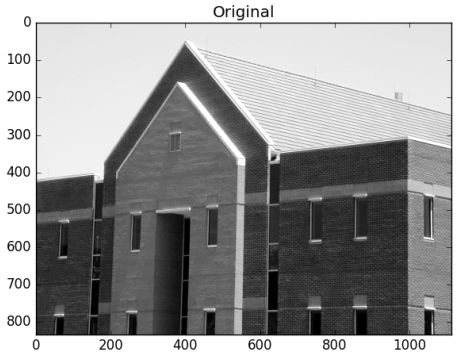

Sobel
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

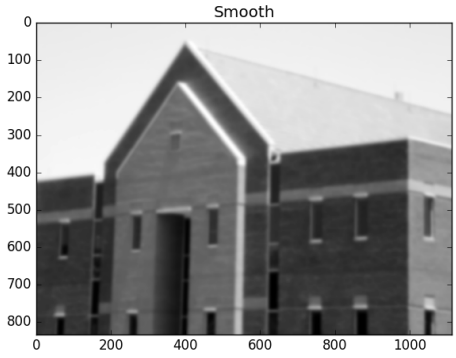

Original

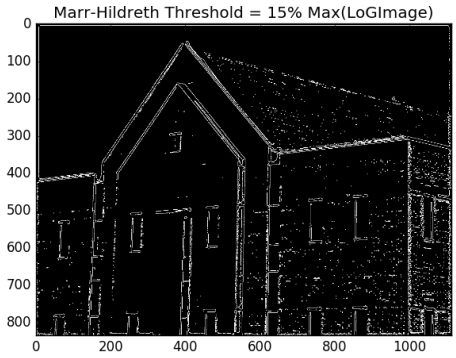

LoG Without Threshold

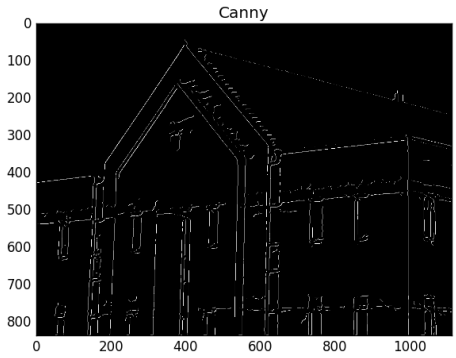

Marr-Hildreth Threshold = 0

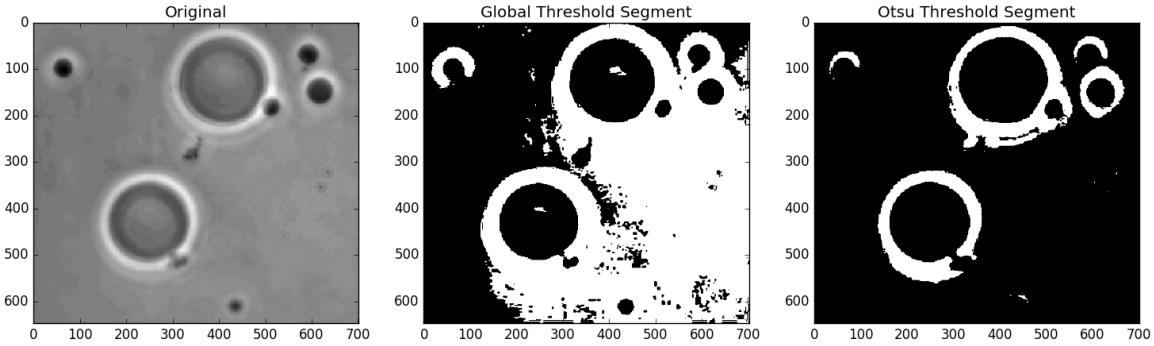

Marr-Hildreth Threshold = 15% Max(LoGImage)


Original


Smooth


Marr-Hildreth Threshold = 15% Max(LoGImage)


Canny


Original
Global Threshold Segment
Otsu Threshold Segment
0
100
200
300
400
500
600
700