《计算机视觉（1）》实验报告

实验五 利用边缘改进全局阈值处理

实验小组成员 （学号+班级+姓名）	分工及主要完成任务	成绩
201800830004 18 数科 叶家辉	全部	

山东大学

2021 年 4 月

完成《数字图像处理》P485页例10.17和10.18的编程实验，编程语言可以选择Matlab，C，C++，OpenCV，Python等。设计方案可参照教科书中的分析，也可以自行设计新的方案。

（1） 图10.42(a)为一幅包含均值为零、标准差为10个灰度级的加性高斯噪声的图像，其中相对于背景，目标所占像素比例极小。采用以梯度为基础的边缘信息结合全局阈值处理方法，实现小目标的分割。

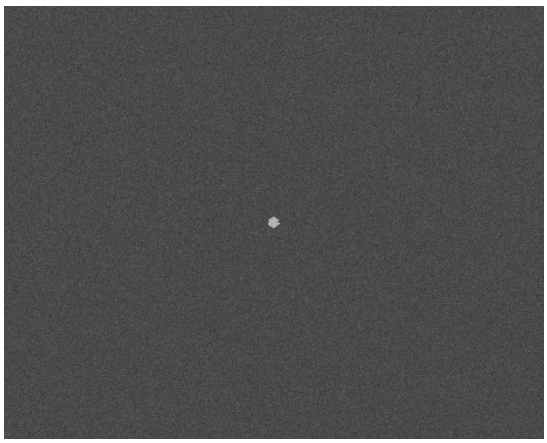


图10.42(a)

（2）图10.43(a)显示了一幅酵母细胞的8比特图像，采用以拉普拉斯为基础的边缘信息结合全局阈值处理方法，实现亮点目标的分割。

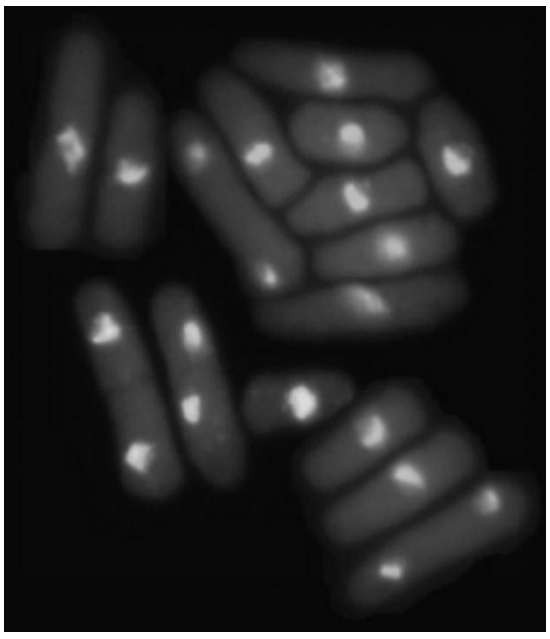


图10.43(a)

原始图像的电子版图像在 Images 文件夹中。实验报告写在如下空白处，页数不限。

一、实验目的与原理

1.1 梯度幅度

在前面的实验中已经涉及过图像梯度幅度的数学含义和实际应用，在此做一个简单的复习。为了在一幅图像的(x,y)处寻找边缘的强度，就需要借助梯度。梯度由下式来进行定义：

$$\nabla f \equiv grad(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

梯度向量指出了图像在(x,y)处灰度的最大变化率的幅度和方向。具体地，幅度$M(x,y)$通过以下式子得到：

$$M(x,y) = mag(\nabla f) = \sqrt{g_x^2 + g_y^2} \qquad (1)$$

为了计算在每个像素点处的偏导数$\frac{\partial f}{\partial x}$和$\frac{\partial f}{\partial y}$，我们选择用于计算梯度偏导数的滤波器模版，通常称为梯度算子。本实验中选用的是 Sobel 算子，其 x 和 y 方向的算子分别为以下形式：

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

其近似 x 和 y 方向的一阶偏微分分别为：

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

我们使用检测水平方向的 Sobel 算子和检测垂直方向的 Sobel 算子就可以分别得到 x 和 y 方向上的梯度分量$|g_x|$和$|g_y|$。计算梯度可用上面式(1)，或用$M(x,y) = |g_x| + |g_y|$近似代替。

1.2 绝对拉普拉斯

在本课程前面的章节中我们讨论过，对于点的检测应该以二阶导数为基础，即使用拉普拉斯算子。

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\nabla^2 f(x,y) = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

如果考虑八邻域，则拉普拉斯模版有下图的形式：

1	1	1
1	-8	1
1	1	1

对于线的检测，二阶导数也将导致更强的响应，并产生比一阶导数更细的线。所以对于线检测，也可以使用拉普拉斯模版。由于拉普拉斯算子是一种微分算子，所以其应用强调的是图像中灰度的突变，并不强调灰度缓慢变化的区域。在计算中，结果可能出现负值的情况，所以需要对最后的结果取绝对值，即得到绝对拉普拉斯图像。这也是边缘检测的一种有力手段。

1.3 全局阈值处理

在前面的实验中也已经涉及过阈值处理，这里的思路和之前类似。一般来说，从背景中提取物体的一种明显方法是选择一个将这些模式分开的阈值T，然后将每个$f(x,y)>T$的点作为对象点，反之则为背景点。即由下式生成一幅二值图像：

$$g(x,y)=\begin{cases}1, f(x,y)>T\\0, f(x,y)\le T\end{cases}$$

当T是一个适用于整个图像的常数时，该公式给出的处理称为全局阈值处理。

1.4 Otsu 算法

Otsu 算法包括以下 7 个步骤：

1、计算输入图像的归一化直方图，用p_i表示该直方图的各个分量；

2、用式$P_1(k)=\sum_{i=1}^{k}p_i$计算累积和$P_1(k)$；

3、用式$m(k)=\sum_{i=1}^{k}ip_i$计算累积均值$m(k)$；

4、用式$m_G=\sum_{i=1}^{L}ip_i$计算全局均值m_G；

5、用式$\sigma_B^2(k)=\frac{[m_GP_1(k)-m(k)]^2}{P_1(k)[1-P_1(k)]}$计算类间方差$\sigma_B^2(k)$；

6、寻找使得$\sigma_B^2(k)$取得最大值的k^*即为得到的 Otsu 阈值，如果最大值不唯一，则用得到最大值的各个k取平均得到k^*；

7、在$k=k^*$处用式$\eta(k)=\frac{\sigma_B^2(k)}{\sigma_G^2}$计算可分性度量$\eta^*$，其中$\sigma_G^2=\sum_{i=1}^{L}(i-m_G)^2p_i$。

1.5 利用边缘改进全局阈值处理的算法

一般来说，如果直方图的波峰是高、窄、对称的，且被深的波谷分开，则选取一个较好的阈值的可能性是比较大的。所以可以仅考虑那些位于或靠近物体和背景间边缘的像素，来改进直方图的形状，因为此时直方图不再依赖物体和背景的相对大小，波峰高度近似相同，直方图模式的对称性也会有所改进，同时边缘像素有加深直方图波峰间波谷的倾向。一个图像是否位于边缘上可通过计算其梯度或者绝对拉普拉斯来确定，本实验的两个部分会分别选择这两种路径。

利用边缘改进全局阈值处理的算法包括以下 4 个步骤：

1、计算$f(x,y)$的边缘图像（梯度或拉普拉斯）；

2、设定一个阈值T（该阈值一般以百分比的形式来表示，通常设置得较高，以便在计算中使用边缘图像中的较少像素），对边缘图像进行阈值处理产生二值边缘图像$g_T(x,y)$，这是强边缘像素的模版图像（mask）；

3、仅对$g_T(x,y)$中像素值为 1 的对应位置计算$f(x,y)$中的直方图；

4、用上述直方图对$f(x,y)$使用 Otsu 方法进行全局分割。

二、实验内容与步骤

注：本实验分为两个部分（“以梯度为基础的边缘信息改进全局阈值处理”【下称“实验 A”】和“以拉普拉斯为基础的边缘信息改进全局阈值处理”【下称“实验 B”】），且每个部分均需要六个显示区域，所以在设计上进行了一定的考量。第一行第一列、第二列的显示区为两个实验相同的部分，包括原图象的显示和直方图的显示。其余四个显示区（即第一行第三列和第二行）根据按钮的不同执行不同的功能，其中实验 A 的功能在上面一行，实验 B 的功能在下面一行。

2.1 界面总体设计与图像导入

本实验的 GUI 界面延续之前一贯的简洁明快的设计风格。同时为了集成计算机视觉（1）的所有实验项目，在 GUI 的顶部特别设计了导航栏，可以在同一窗口定位到不同的实验项目。点击“5-利用边缘改进全局阈值处理”后即可得到本实验的整体操作界面。

和前几个实验一样，左上角有“打开文件”和“保存图像”两个图标。点击“打开文件”图标选择相应图片，即可在坐标图区 axes1（第一行第一列图像显示区）导入这张图片。“打开文件”按钮的回调函数如下图所示。前四行获取了所选定图片的文件名和路径名，再根据得到的路径名和文件名读取文件。后两行则把读取到的文件显示到坐标图区，并设定顶部的文本框显示图片路径名/文件名。

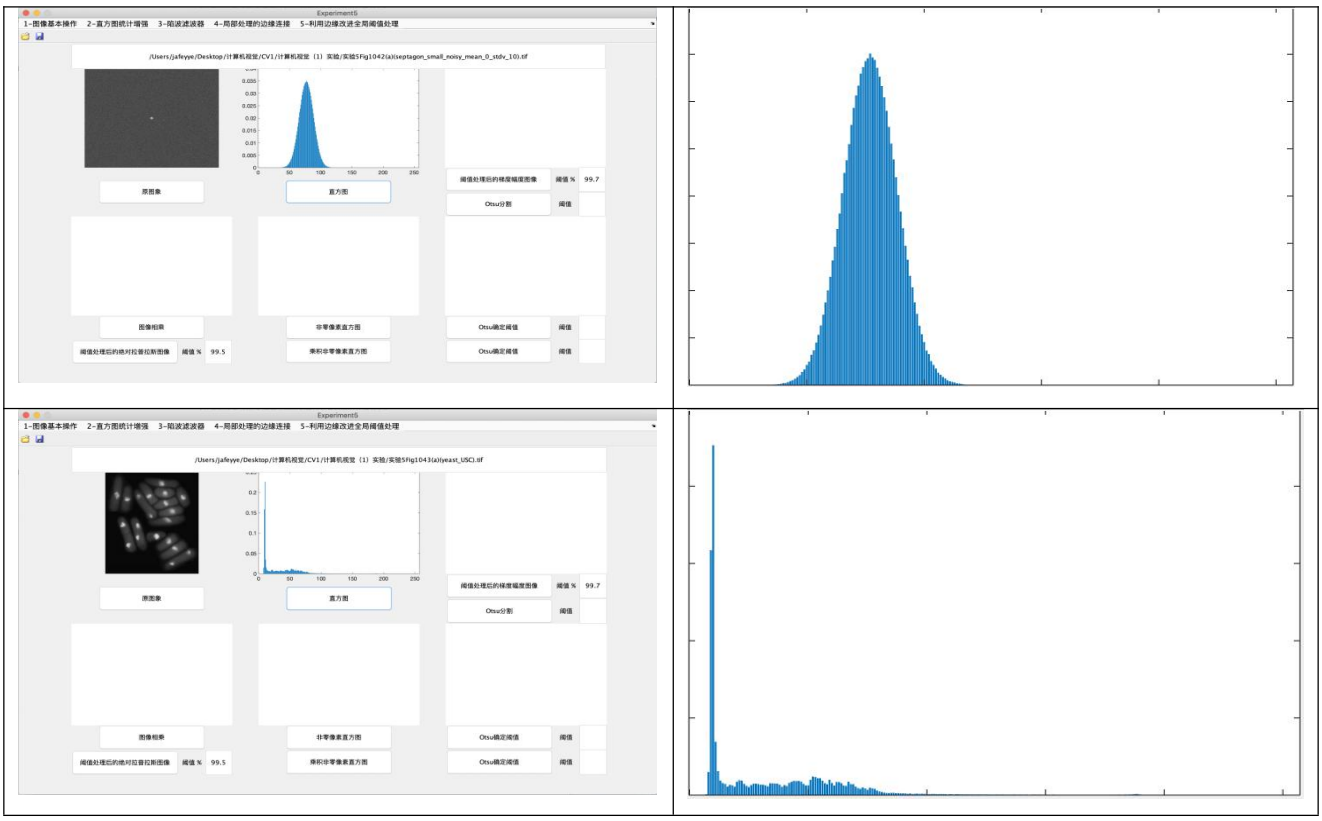

```
% --------------------------------------------------------------------
function uipushtool1_ClickedCallback(hObject, eventdata, handles)
% hObject    handle to uipushtool1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename,pathname]=uigetfile({'*.bmp;*.jpg;*.png;*.jpeg;*.tif'},'请选择图片');
str=[pathname filename];
OriginalPic = imread(str);
axes(handles.axes1);

imshow(OriginalPic);
set(handles.edit1,'String',str);
```

2.2 图像直方图的绘制

在前面的实验中涉及过直方图的绘制，在此结合本实验做一个简单的回顾。通过点击“直方图”按键实现直方图的绘制，并将结果返回至第一行第二列的显示区域中。“直方图”按键的回调函数如下图所示。首先从顶部文本框处获取图像路径名和文件名，并由此读入图像信息，然后用每个灰度像素出现的个数除以整张图的像素个数来计算这张图的直方图（注意 Matlab 中的范围是 1 到 256），并绘制在坐标图区 axes2（第一行第二列的显示区域）中。

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
img = imread(get(handles.edit1,'string'));
h = imhist(img)/numel(img);
h1 = h(1:1:256);
horz=1:1:256;
axes(handles.axes2);
bar(horz,h);
```



A. 以梯度为基础的边缘信息改进全局阈值处理

2.3 阈值处理后的梯度幅度图像

这个部分主要分为两个步骤，首先是得到梯度幅度图像，然后是根据设定的阈值得到一幅二值图像。

为了计算一幅图像的梯度大小，首先需要计算在每一个像素点上分别沿 x 轴和 y 轴方向上的一阶导数，然后利用式(1)即可得到梯度的幅度。梯度的方向则与边缘的方向相垂直。一幅图像的边缘处的梯度幅度比其它处更大，所以梯度幅度的图像反映了边缘的情况。下图 210 ~ 213 行展示的是得到梯度幅度的回调函数。读入图像后，首先得到图像每一点处的灰度值，然后进行归一化，再通过 imgradientxy( )函数计算图像f在(x,y)坐标处沿 x 轴和 y 轴方向的导数（这里选择 Sobel 算子）。然后根据式子(1)计算结果得到梯度幅度矩阵$M(x,y)$。

阈值处理的基本思想是当且仅当像素(x,y)处的某些特征（此处是梯度幅度值）满足一定条件时，将该像素置为 1，否则置为 0。落实到操作上，下图中代码的其余部分展示了这一过程。首先从“阈值处理后的梯度幅度图像”按键右侧的文本框中读取设定的阈值。该值通常由百分数n表示，指的是大于给定集合内所有数字的$n\%$的最小数字。所以接下来通过这个百分数计算出该阈值在整个集合中所排的位置，并且找到这个位置所代表的具体值，这一过程由下图中第 214 ~ 216 行代码来实现。然后创建一个与原图象同维的零矩阵，并对$M(x,y)$中的每个元素做判断：如果该元素大于阈值，就将零矩阵对应位置的元素置 1，否则保持为 0。最后把如是生成的二值图像显示在坐标图区 axes3（第一行第三列的显示区域）中。

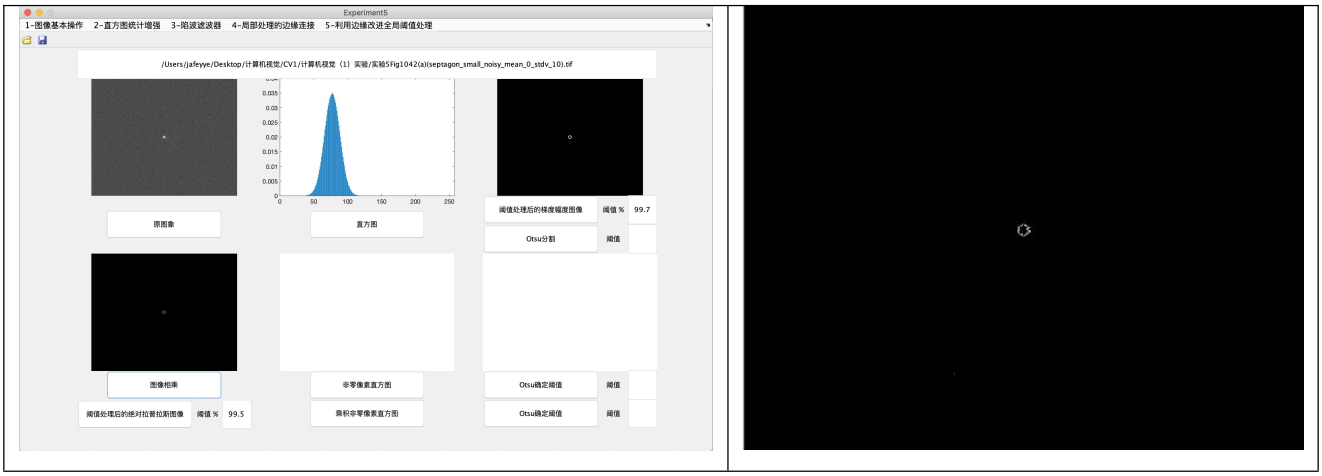
```
204      % --- Executes on button press in pushbutton3.
205      function pushbutton3_Callback(hObject, eventdata, handles)
206      % hObject    handle to pushbutton3 (see GCBO)
207      % eventdata  reserved - to be defined in a future version of MATLAB
208      % handles    structure with handles and user data (see GUIDATA)
209 -    T = str2num(get(handles.edit2,'string'))/100;
210 -    file = get(handles.edit1,'string');
211 -    f = imread(file)/256;
212 -    [gx gy] = imgradientxy(f, 'sobel');
213 -    M = (gx.^2+gy.^2).^0.5;
214 -    num = int64(numel(M) * T);
215 -    sort_M = sort(M(:));
216 -    threshold = sort_M(num);
217 -    C = zeros(size(f));
218 -    [H,W] = size(f);
219 -    for i = 1:H
220 -        for j = 1:W
221 -            if M(i,j)>threshold
222 -                C(i,j)=1;
223 -            end
224 -        end
225 -    end
226 -    axes(handles.axes3);
227 -    imshow(C);
```

2.4 将二值图像与原图象相乘

这一步骤其实是为了方便后续操作，把对$g_T(x,y)$中像素值为 1 的对应位置计算$f(x,y)$中的直方图，转化为计算$g_T(x,y)$与$f(x,y)$乘积图像的直方图。

“图像相乘”的回调函数如下图所示。首先读取顶部文本框的路径名/文件名来输入原图象，然后读入第一行第三列的二值边缘图像$g_T(x,y)$，因为 imread( )读入的格式为 uint8（1～256），为了保证阵列相乘两项的数据格式相同，这里把二值图像也转化为 uint8 的格式。最后把相乘的结果显示在坐标图区 axes4（第二行第一列的显示区域）中。

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = get(handles.edit1,'string');
f = imread(file);
img_C = getimage(handles.axes3);
img_D = f .* uint8(img_C);
axes(handles.axes4);
imshow(img_D);
```

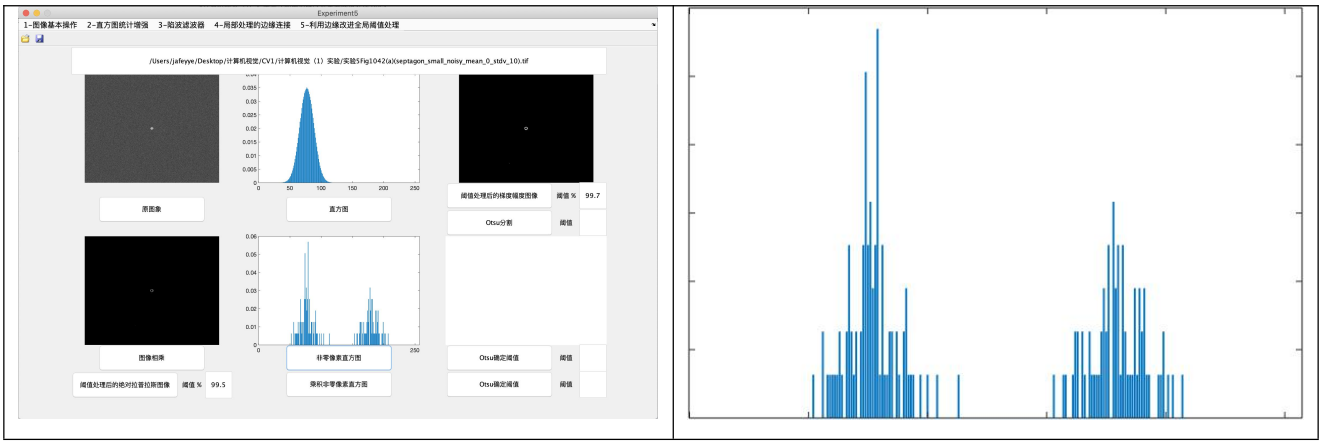


2.5 计算非零像素直方图

得到了$g_T(x,y)$与$f(x,y)$乘积的图像，接下来只要计算该图像中除 0 灰度处外其余区域的直方图即可。“非零像素直方图”回调函数如下图所示。首先读入第二行第一列的乘积图像并计算其归一化直方图，然后把灰度直方图第一个灰度值的频率设为 0，再用剩余各个灰度的频率分别除以剩余灰度的总和，以进行归一化，使得剩下的灰度频率总和还是 1，这一步非常关键，不可或缺。最后将直方图显示在坐标图区 axes5（第二行第二列的显示区域）中即可。

```
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
img_D = getimage(handles.axes4);
h = imhist(img_D) / numel(img_D);
h(1) = 0;
h = h / sum(h);
h1 = h(1:1:256);
horz=1:1:256;
axes(handles.axes5);
bar(horz,h);
```

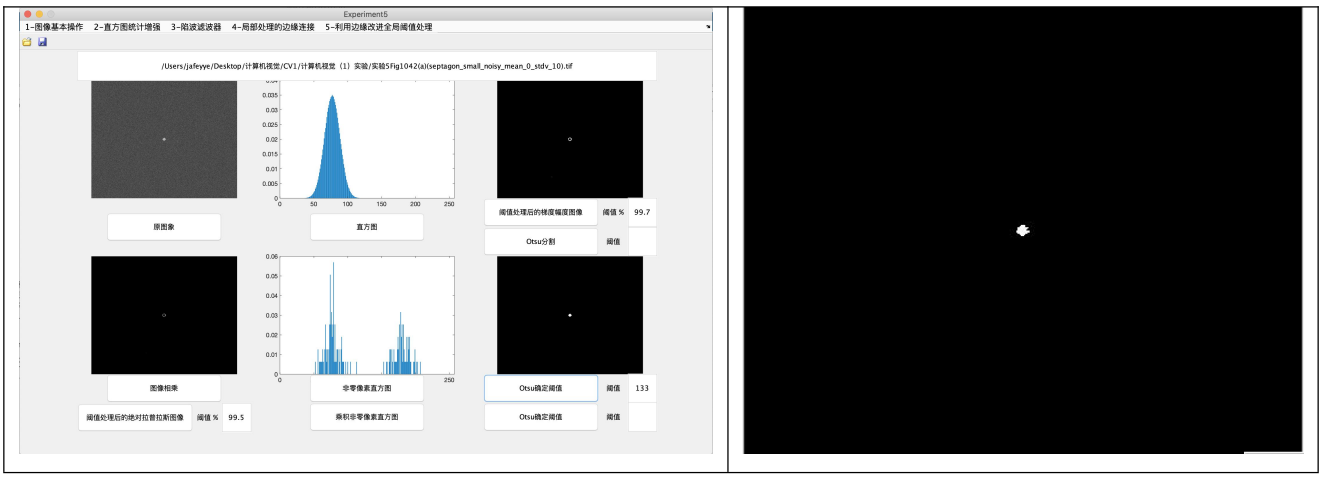
从直方图的形状中可以看出，两组波峰近似对称，同时被明显的波谷所分割，所以选到一个好的阈值的希望很大。



2.6 用 Otsu 法确定阈值

首先还是根据顶部文本框来读入原图象，因为之后的全局分割需要在原图象上进行。接下来仿照 2.5 重新生成一遍直方图（277 ~ 281 行）。再按照 1.4 中的步骤依次计算各个值（282 ~ 294 行），然后将得到的阈值显示在按钮旁的文本框中，并根据阈值用 im2bw( )函数对原图进行分割，将分割的结果显示在坐标图区 axes6（第二行第三列的显示区域）中。

```
270      % --- Executes on button press in pushbutton6.
271      function pushbutton6_Callback(hObject, eventdata, handles)
272      % hObject    handle to pushbutton6 (see GCBO)
273      % eventdata  reserved - to be defined in a future version of MATLAB
274      % handles    structure with handles and user data (see GUIDATA)
275 -    file = get(handles.edit1,'string');
276 -    f = imread(file);
277 -    img_D = getimage(handles.axes4);
278 -    h = imhist(img_D) / numel(img_D);
279 -    h(1) = 0;
280 -    h = h / sum(h);
281 -    h1 = h(1:1:256);
282 -    P1(1) = h1(1);
283 -    m(1) = h1(1);
284 -    for k = 2:length(h1)
285 -        P1(k) = P1(k-1) + h1(k);
286 -        m(k) = m(k-1) + k*h1(k);
287 -    end
288 -    mG = m(length(h1));
289 -    for k = 1:length(h1)
290 -        sigmaB2(k) = (mG*P1(k)-m(k))^2/(P1(k)*(1-P1(k)));
291 -    end
292 -    maximum = max(sigmaB2);
293 -    K = find(sigmaB2==maximum);
294 -    k_star = mean(K);
295 -    set(handles.edit3,'string',num2str(uint8(k_star)));
296 -    axes(handles.axes6);
297 -    final = im2bw(f,k_star/256);
298 -    imshow(final);
```



不难看出，最后的结果还是比较完美的。

B. 以拉普拉斯为基础的边缘信息改进全局阈值处理

2.7 Otsu 法分割

得到原图的直方图后，直接把 Otsu 方法用于所示的直方图上，观察所能得到的结果。整体思路与实现方法均与 2.6 类似，在此不再赘述。最后将结果显示在坐标图区 axes3（第一行第三列的显示区域）中。

不难发现，Otsu 方法未能实现检测亮点这一原始目标，同时，尽管该方法能隔离出某些细胞区域本身，但右侧几个分割后的区域依旧没有分开。

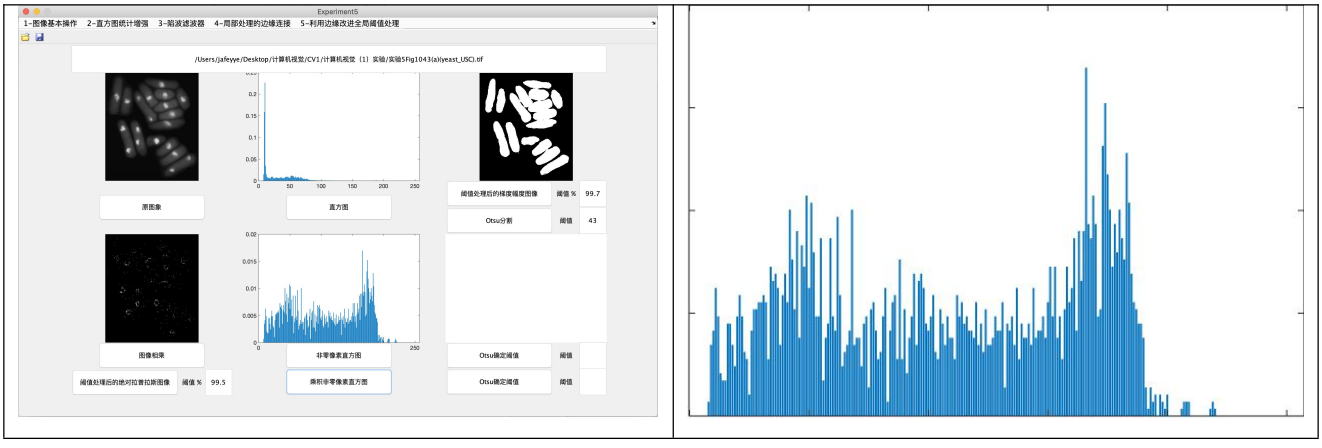
2.8 阈值处理后的绝对拉普拉斯图像

这个部分与 2.3 类似，区别只在与生成边缘图像的方式（445 ~ 447 行）。2.3 中生成边缘图像采用的是梯度幅度，这里选择绝对拉普拉斯。首先构造一个拉普拉斯算子，然后运用该算子对图像进行滤波，就可以得到锐利的边缘。由于可能出现负值，所以对结果取绝对值，即可得到绝对拉普拉斯图像。其他部分均可参照 2.3 的相应内容，包括计算给定阈值百分数所对应的具体灰度值，然后根据该灰度值得到一幅二值图像。最后将结果显示在坐标图区 axes4（第二行第一列的显示区域）中。

```
439 -   function pushbutton8_Callback(hObject, eventdata, handles)
440 -   % hObject    handle to pushbutton8 (see GCBO)
441 -   % eventdata  reserved - to be defined in a future version of MATLAB
442 -   % handles    structure with handles and user data (see GUIDATA)
443 -   T = str2num(get(handles.edit5,'string'))/100;
444 -   file = get(handles.edit1,'string');
445 -   f = im2double(imread(file));
446 -   w = fspecial('laplacian',0);
447 -   g = abs(imfilter(f,w,'replicate'));
448 -   num = int64(numel(g) * T);
449 -   sort_g = sort(g(:));
450 -   threshold = sort_g(num);
451 -   D = zeros(size(f));
452 -   [H,W] = size(f);
453 -   for i = 1:H
454 -       for j = 1:W
455 -           if g(i,j)>threshold
456 -               D(i,j)=1;
457 -           end
458 -       end
459 -   end
460 -   axes(handles.axes4);
461 -   imshow(D);
```

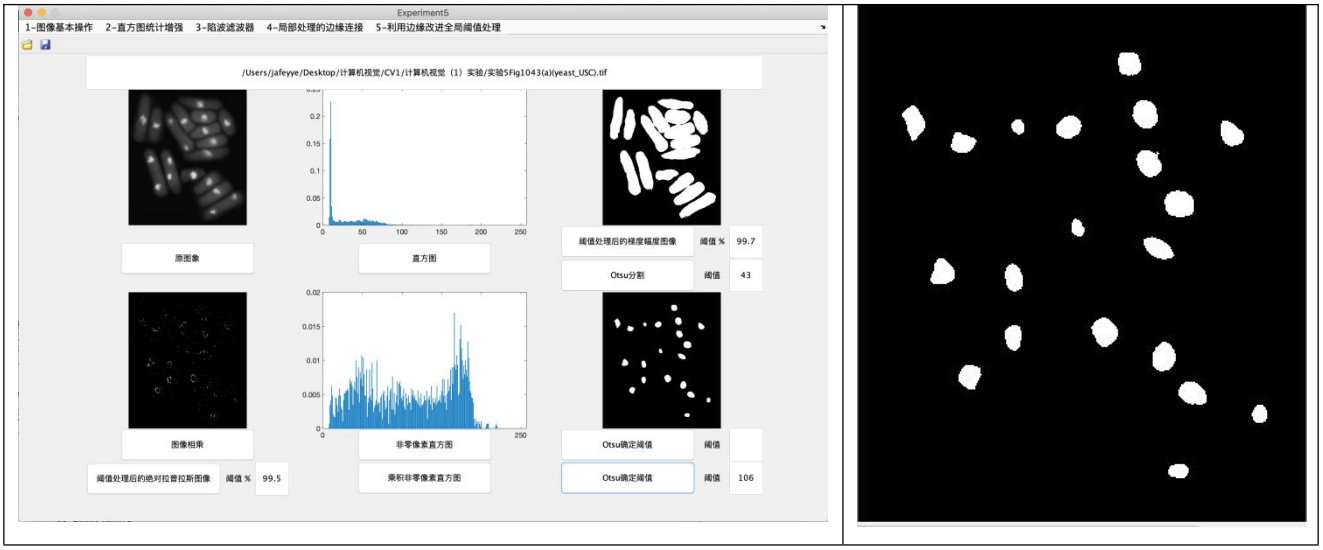
2.9 计算二值图像与原图象乘积的非零像素直方图

该部分相当于 2.4 和 2.5 的整合。先计算 2.8 中得到的二值图像与原图象的乘积，然后对乘积后得到的图像中除 0 灰度外的其它灰度值归一化后做直方图。在此不再赘述。



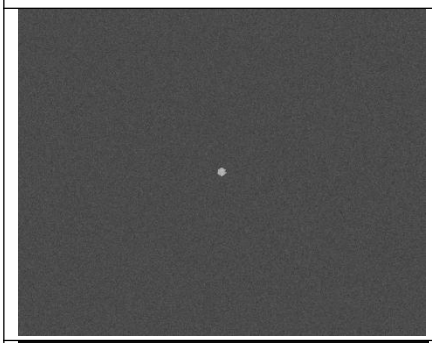
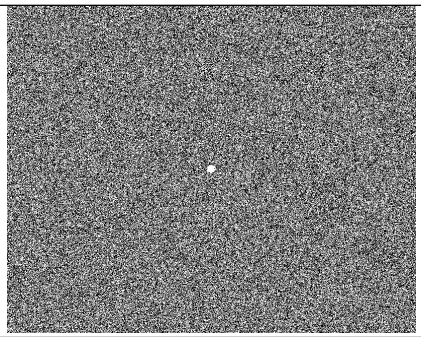
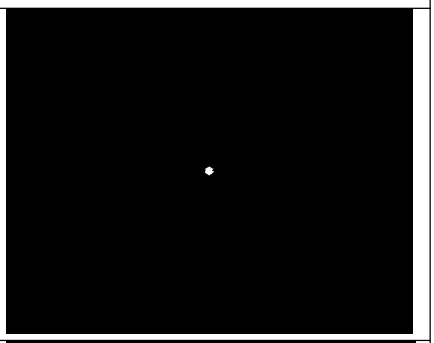
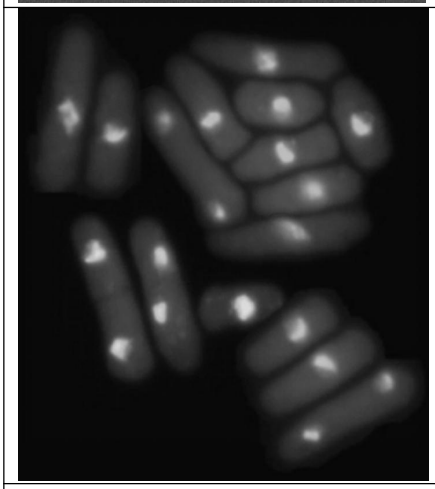
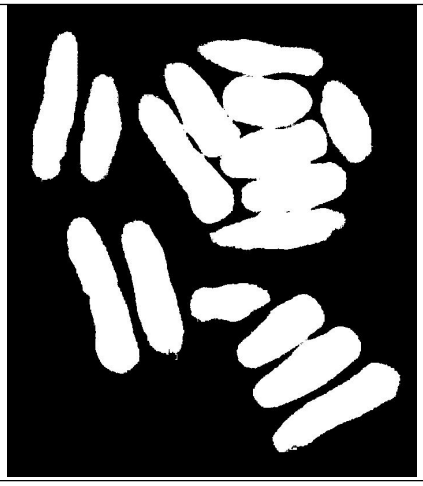
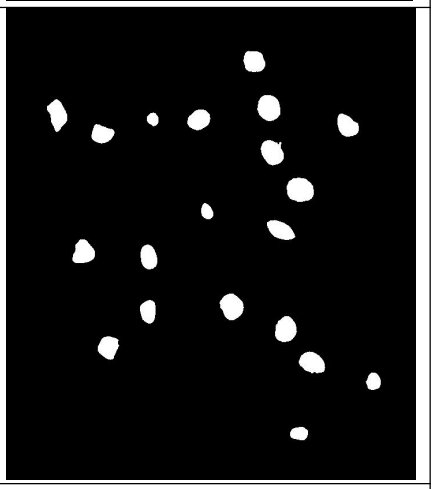
2.10 用 Otsu 法确定阈值

该部分仿照 2.6，先读入原图象，再生成 2.9 中的直方图。按照 1.4 中的式子依次计算各个值，然后将得到的阈值显示在按钮旁的文本框中，并根据阈值用 im2bw( )函数对原图进行分割，将分割的结果显示在坐标图区 axes6（第二行第三列的显示区域）中。



很明显，最后的结果有了很大的提升。

三、实验结果与总结

原图象	直接使用 Otsu 方法	利用边缘改进的 Otsu 方法
		
		

可以看出，利用边缘改进全局阈值处理能够获得比较大的提升。