计算机视觉与模式识别

苏远歧

新型计算机研究所

Canny Edge Detection

目标：获取图像的边缘

Original image, I

标记像素点是否是边缘的二值图

Edge map image, B

Canny Edge Detection

目标：获取图像的边缘

Original image, I

标记像素点是否是边缘的二值图

Edge map image, B

$$B(i,j)=\begin{cases}1 & I(i,j)\text{是边缘}\\ 0 & \text{否则}\end{cases}$$

Canny Edge Detection

1. 利用高斯梯度算子对图像进行滤波

2. 根据滤波结果计算每一像素点的边缘强度

3. 计算边缘方向

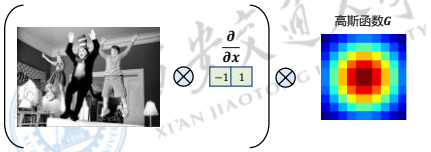
4. 检测局部最大值

5. 连接生成边缘

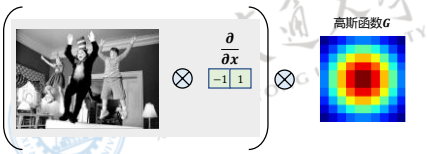
1）计算边缘梯度

计算图像I在x和y方向的边缘梯度

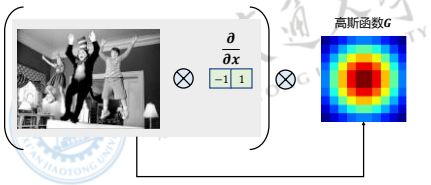
Step 1, 计算梯度并对梯度进行平滑:



Step 1, 计算梯度并对梯度进行平滑:

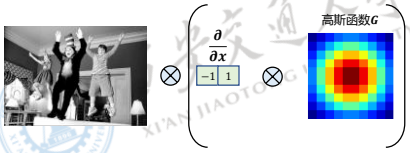


Step 1, 计算梯度并对梯度进行平滑:

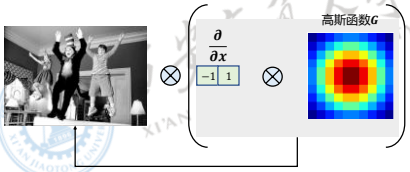


Step 1, 计算梯度并对梯度进行平滑:

$$(\mathrm{I} \otimes h) \otimes g = \mathrm{I} \otimes (h \otimes g)$$

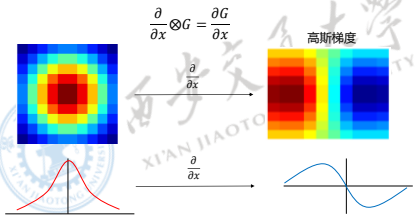


Step 1, 计算梯度并对梯度进行平滑:

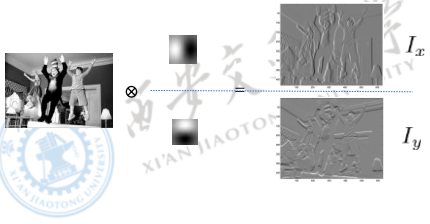


Step 1, 计算梯度并对梯度进行平滑:

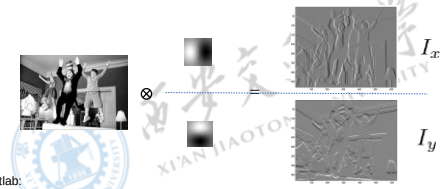
$$\frac{\partial}{\partial x} \otimes G = \frac{\partial G}{\partial x}$$



Step 1, 计算梯度并对梯度进行平滑:



Step 1, 计算梯度并对梯度进行平滑:

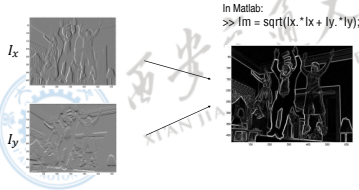


In matlab:
>> [dx,dy] = gradient(G); % G is a 2D gaussain
>> Ix = conv2(I,dx,'same'); Iy = conv2(I,dy,'same');

2) 计算边缘强度

综合图像I在x和y方向的边缘梯度计算一个强度

Step 2, 计算梯度强度:



In Matlab:
>> Im = sqrt(Ix.*Ix + Iy.*Iy);

通过边缘强度，我们大致知晓边缘的位置，但是仍需要进一步准确定位

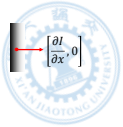
3) 计算边缘方向

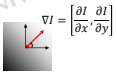
根据图像I在x和y方向的梯度计算方向

Step 3, 计算梯度方向:

- 图像梯度 $\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$

The gradient points in the direction of most rapid change in intensity!

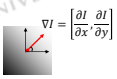


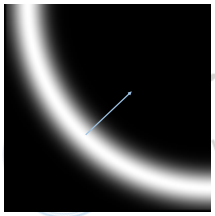


Step 3, 计算梯度方向:

- 图像梯度 $\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$

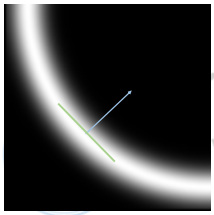
$\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} \Big/ \frac{\partial I}{\partial x}\right)$





- 梯度方向 $\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} \Big/ \frac{\partial I}{\partial x}\right)$

(Forsyth & Ponce)



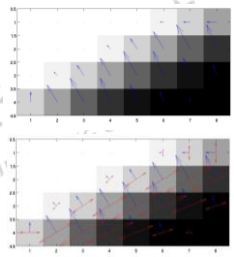
- 梯度方向 $\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} \Big/ \frac{\partial I}{\partial x}\right)$
- 边缘方向 $e = \tan^{-1}\left(-\frac{\partial I}{\partial x} \Big/ \frac{\partial I}{\partial y}\right)$

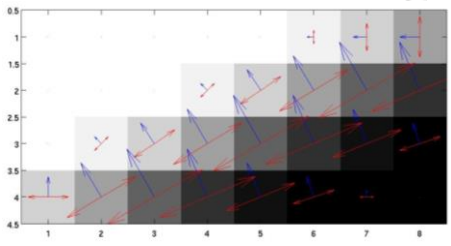
(Forsyth & Ponce)

如何观察梯度方向

善用Matlab的quiver函数!

```
%% define image gradient operator
dy = [1;-1];
dx = [1,-1];

%% compute image gradient in x and y
Iy = conv2(I,dy,'same');
Ix = conv2(I,dx,'same');


%% display the image gradient flow
figure(3);clf;imagesc(J);colormap(gray);axis image;
hold on;
quiver(Jx,Jy);
quiver(-Jy,Jx,'r');
quiver(Jy,-Jx,'r');
```





```
[gx,gy] = gradient(J);
mag = sqrt(gx.*gx+gy.*gy);
imagesc(mag);colorbar
```

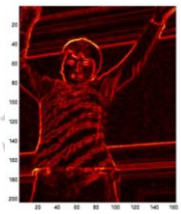
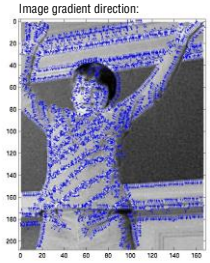
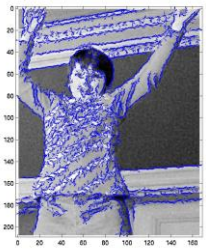


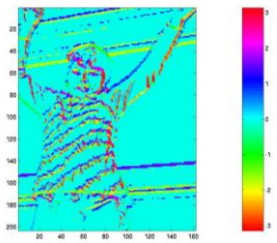
Image gradient direction:



Edge orientation direction:



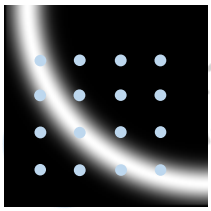
```
[gx,gy] = gradient(J);
% or you can use:[th,mag] = cart2pol(gx,gy);
th = atan2(gy,gx);
imagesc(th.*(mag>20));
colormap(hsv); colorbar
```



4) 检测局部最大值

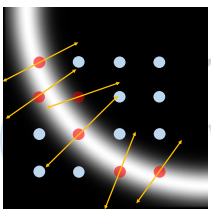
沿梯度方向搜索最大值

Step 4, 局部最大值检测:



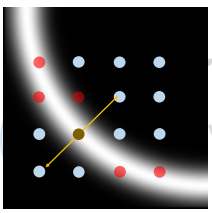
(Forsyth & Ponce)

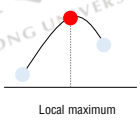
阈值处理:



(Forsyth & Ponce)

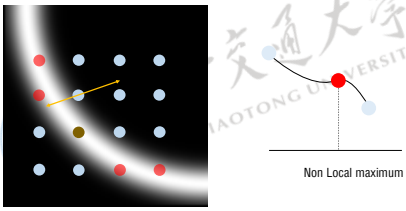
阈值处理:





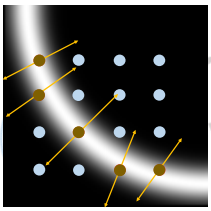
(Forsyth & Ponce)

阈值处理:



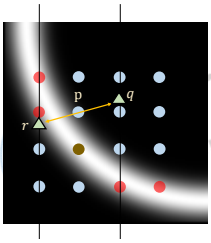
(Forsyth & Ponce)

Step 4, 局部最大值检测:



(Forsyth & Ponce)

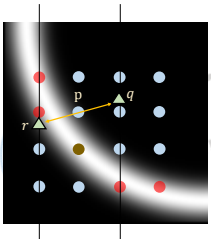
沿着梯度方向的搜索



q，r不在网格点上?

(Forsyth & Ponce)

沿着梯度方向的搜索



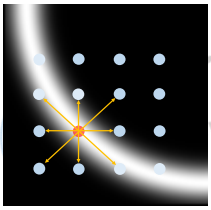
q，r不在网格点上?

- 对q，r点的强度值进行插值
- 限制边缘的方向

(Forsyth & Ponce)

阈值处理:

限制边缘的方向



限制在8邻域中搜索

$[0,\pi] \rightarrow \left(0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\right)$

θ和$\theta+\pi$是同一个搜索方向

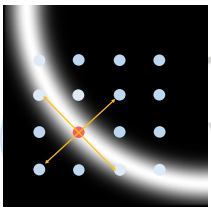
(Forsyth & Ponce)

5) 进行边缘的链接

沿边缘方向搜索下一个边缘点

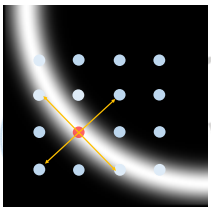
Step 5, 边缘连接:

已知一个边缘点，下一个边缘点在哪儿?



Step 5, 边缘连接:

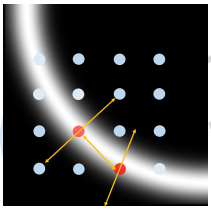
已知一个边缘点，下一个边缘点在哪儿?



- 估计边缘曲线的切线方向(which is normal to the gradient at that point)
- 据此去估计边缘上的下一个点

Step 5, 边缘连接:

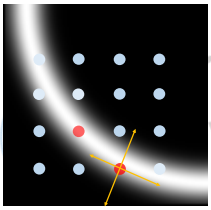
已知一个边缘点，下一个边缘点在哪儿?



- 估计边缘曲线的切线方向(which is normal to the gradient at that point)
- 据此去估计边缘上的下一个点

Step 5, 边缘连接:

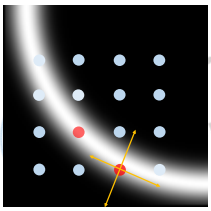
已知一个边缘点，下一个边缘点在哪儿?



- 估计边缘曲线的切线方向(which is normal to the gradient at that point)
- 据此去估计边缘上的下一个点

Step 5, 边缘连接:

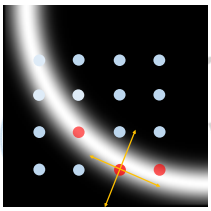
已知一个边缘点，下一个边缘点在哪儿?



- 估计边缘曲线的切线方向(which is normal to the gradient at that point)
- 据此去估计边缘上的下一个点

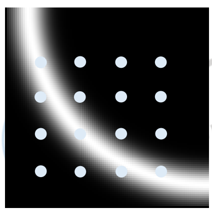
Step 5, 边缘连接:

已知一个边缘点，下一个边缘点在哪儿?



- 估计边缘曲线的切线方向(which is normal to the gradient at that point)
- 据此去估计边缘上的下一个点

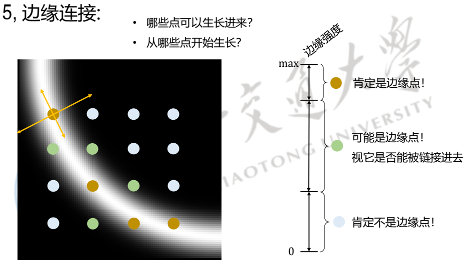
Step 5, 边缘连接:



- 哪些点可以生长进来?
- 从哪些点开始生长?

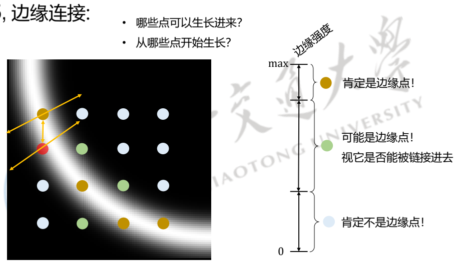
Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



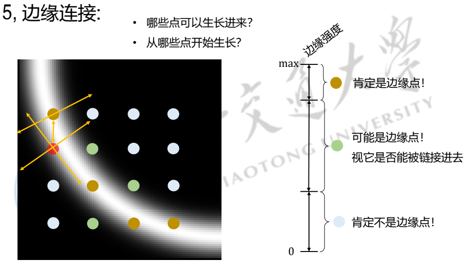
Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



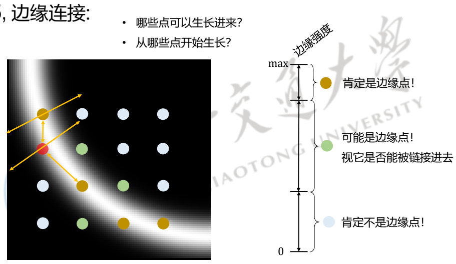
Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



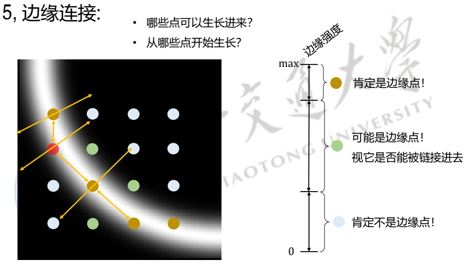
Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



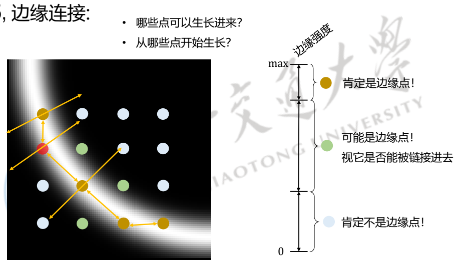
Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



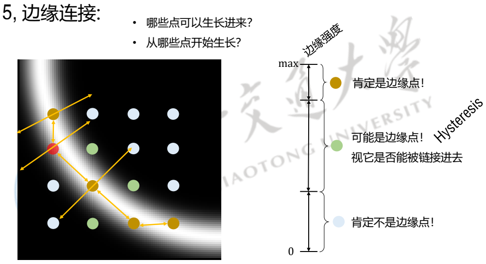
Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



Step 5, 边缘连接:

- 哪些点可以生长进来?
- 从哪些点开始生长?



Edge Linking: Hysteresis

- 边缘的链接
  - 从哪些点开始?
  - 可以将哪些点纳入到边缘中?

Canny边缘检测的Matlab实现


1. 利用高斯梯度算子对图像进行滤波


2. 根据滤波结果计算每一像素点的边缘强度


3. 计算边缘方向

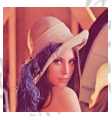

4. 检测局部最大值

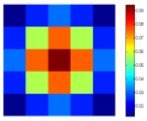

5. 连接生成边缘

1) 计算边缘梯度

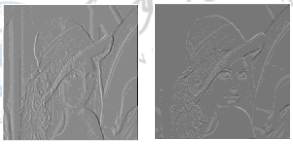
计算图像I在x和y方向的边缘梯度

```
img = imread ('Lenna.png');
img = rgb2gray(img);
img = double (img);

% Value for high and low thresholding
threshold_low = 0.035;
threshold_high = 0.175;

%% Gaussian filter definition (https://en.wikipedia.org/wiki/Canny_edge_detector)
G = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2];
G = 1/159.* G;

%Filter for horizontal and vertical direction
dx = [1 0 -1];
dy = [1; 0; -1];
```





```
% % Convolution of image with Gaussian
Gx = conv2(G, dx, 'same');
Gy = conv2(G, dy, 'same');

% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```



Ix Iy

2）计算边缘强度

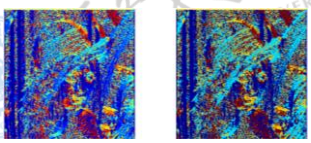
综合图像I在x和y方向的边缘梯度计算一个强度

3）计算边缘方向

根据图像I在x和y方向的梯度计算方向

```
%Calculate magnitude and angle
magnitude = sqrt(Ix.*Ix+Iy.*Iy);
angle = atan2(Iy, Ix);

%% Edge angle conditioning
angle(angle<0) = pi+angle(angle<0);
angle(angle>7*pi/8) = pi-angle(angle>7*pi/8);
```

```
% Edge angle discretization into 0, pi/4, pi/2, 3*pi/4
angle(angle>=0&angle<pi/8) = 0;
angle(angle>=pi/8&angle<3*pi/8) = pi/4;
angle(angle>=3*pi/8&angle<5*pi/8) = pi/2;
angle(angle>=5*pi/8&angle<=7*pi/8) = 3*pi/4;
```

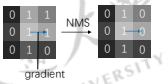


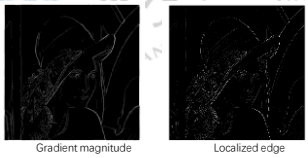
Continuous angle Discretized angle

4）检测局部最大值

沿梯度方向搜索最大值

```
edge = zeros(nr, nc);

%% Non-Maximum Supression
edge = non_maximum_suppression(magnitude, angle, edge);

edge = edge.*magnitude;
```

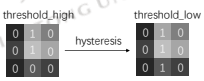




5) 进行边缘的链接

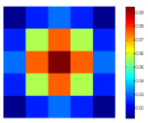
沿边缘方向搜索下一个边缘点

```
%% Hysteresis thresholding
% for weak edge
threshold_low = threshold_low * max(edge(:));
% for strong edge
threshold_high = threshold_high * max(edge(:));
linked_edge = zeros(nr, nc);
linked_edge = hysteresis_thresholding(threshold_low, threshold_high, linked_edge, edge);
```

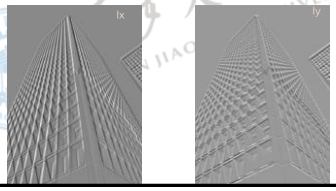


Canny Edge Implementation

```
img = imread ('image.png');
img = rgb2gray(img);
img = double (img);

% Value for high and low thresholding
threshold_low = 0.035;
threshold_high = 0.175;

%% Gaussian filter (https://en.wikipedia.org/wiki/Canny_edge_detector)
G = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2];
G = 1/159 * G;

%Filter for horizontal and vertical direction
dx = [1 -1];
dy = [1; -1];
```

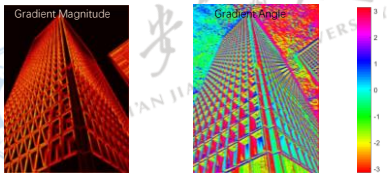


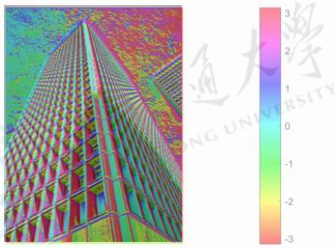
```
% % Convolution of image with Gaussian
Gx = conv2(G, dx, 'full');
Gy = conv2(G, dy, 'full');

% Convolution of image with Gx and Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```



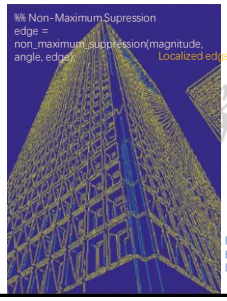
Canny Edge Implementation

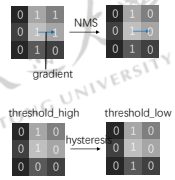
```
angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
```





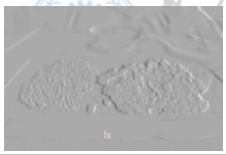
Canny Edge Implementation

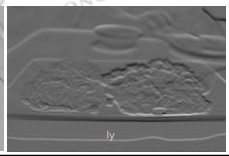





```
low = threshold_low * max(edge(:));
high = threshold_high * max(edge(:));
linked_edge = hysteresis_thresholding(low, high);
```

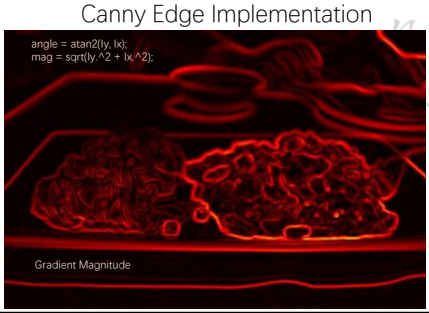
```
%% Convolution of image with
Gaussian
Gx = conv2(G, dx, 'full');
Gy = conv2(G, dy, 'full');

% Convolution of image with Gx and
Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```

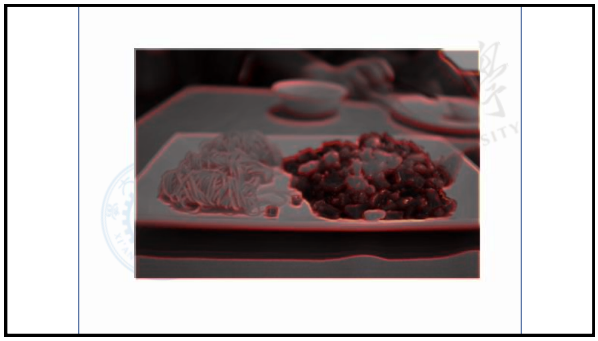




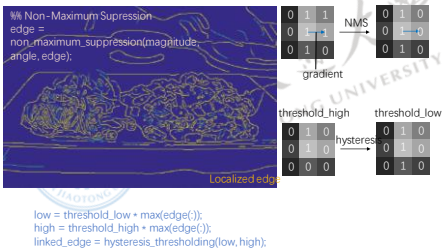
Canny Edge Implementation

```
angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
```





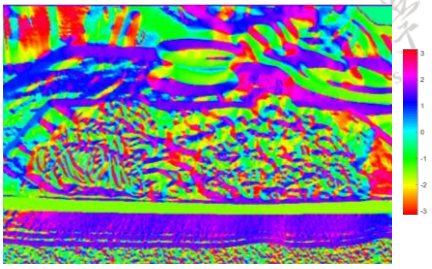
Canny Edge Implementation

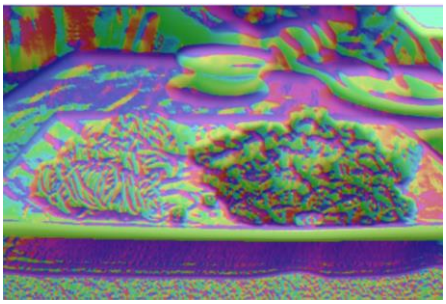


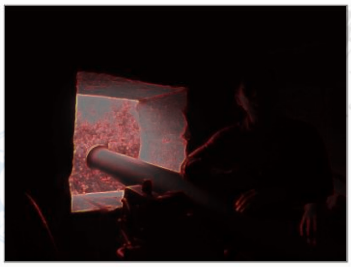

```
low = threshold_low * max(edge(:));
high = threshold_high * max(edge(:));
linked_edge = hysteresis_thresholding(low, high);
```

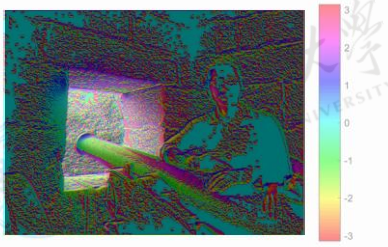
Canny Edge Implementation

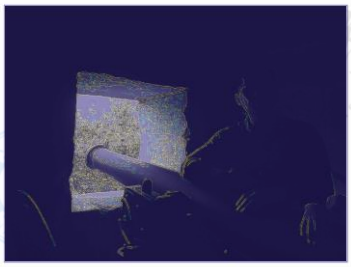
```
angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
```

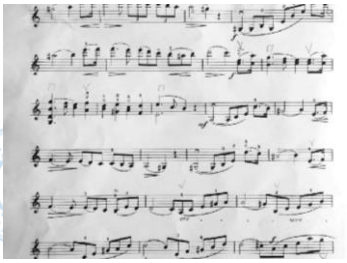




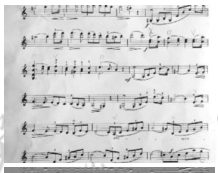


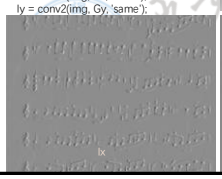

3
2
1
0
-1
-2
-3

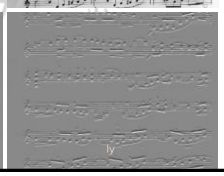


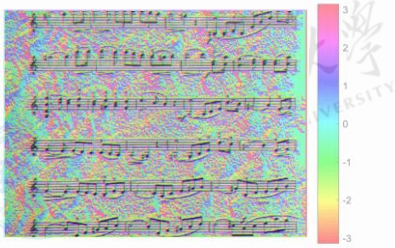


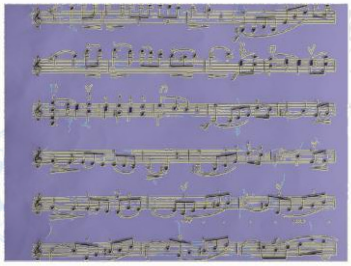
```
% % Convolution of image with
Gaussian
Gx = conv2(G, dx, 'full');
Gy = conv2(G, dy, 'full');

% Convolution of image with Gx and
Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```



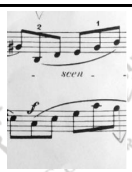


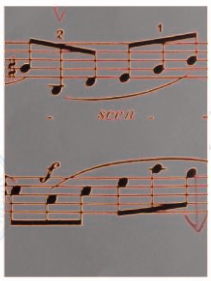


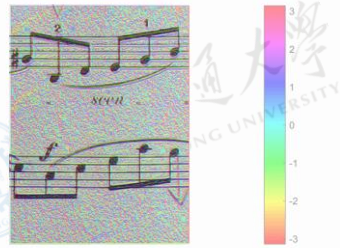


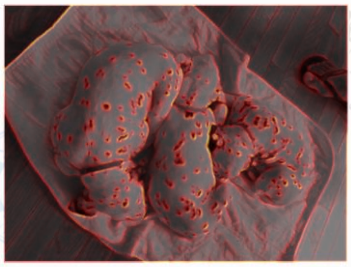


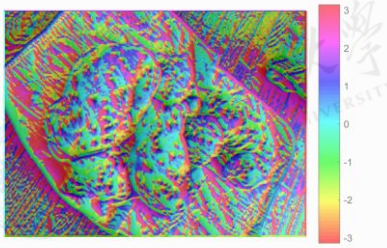
```
% % Convolution of image with
Gaussian
Gx = conv2(G, dx, 'full');
Gy = conv2(G, dy, 'full');

% Convolution of image with Gx and
Gy
Ix = conv2(img, Gx, 'same');
Iy = conv2(img, Gy, 'same');
```






3
2
1
0
-1
-2
-3




3
2
1
0
-1
-2
-3