Non-maximum suppression along the line of the gradient

(Forsyth & Ponce)
Gradient direction

(Forsyth & Ponce)
%% 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');
quiver(Jy,-Jx,'r');
In python:

```python
ind = np.arange(10)
I = np.arange(10).reshape(1,10) + np.arange(4).reshape(4,1)
I = I[:-1,:-1]

dy = np.array([[1,-1]]).reshape(2,1)
dx = np.array([[1,-1]]).reshape(1,2)
ly = signal.convolve2d(I,dy,'same', boundary = 'symm')
lx = signal.convolve2d(I,dx,'same', boundary = 'symm')

ly = ly * -1

plt.imshow(I)
plt.quiver(lx, ly, color = 'b')
plt.quiver(-1 * ly, lx, color = 'r')
plt.quiver(ly, -1 * lx, color = 'r')
plt.show()
```

Efficiently created 2d array by broadcasting trick.

\[ \text{Ary1} \text{ (N1,1) + Ary2(1,N2) = AryOUT (N1, N2)} \]

Flip the image, pixel value ascending from left to right, bottom to up.

In quiver, the coordinates order of Y axis is different from the image coordinates.
No intensity values at r and p: Interpolate these intensities using neighbor pixels.

Where is next edge point?
Python interp2 code

Vq = Interp2(V, Xq, Yq)

V: The 2d data array, has shape of H*W
Xq: The X coordinates of quiver point, 1D or 2D array
Yq: The Y coordinates of quiver point, 1D or 2D array
Vq: The query point value, the same shape as Xq, Yq

Sample code:

```
V = np.arange(4).reshape(2,2) + 10

V:

[[10 11]
 [12 13]]

Xq, Yq = np.meshgrid(np.arange(2,step = 0.5), np.arange(2,step = 0.5))

Xq:

[[ 0.  0.5  1.  1.5]
 [ 0.  0.5  1.  1.5]
 [ 0.  0.5  1.  1.5]
 [ 0.  0.5  1.  1.5]]

Yq:

[[ 0.  0.  0.  0.]
 [ 0.5 0.5 0.5 0.5]
 [ 1.  1.  1.  1.]
 [ 1.5 1.5 1.5 1.5]]

Vq = interp2(V,Xq,Yq)

Vq:

[[ 10.  10.5  11.  11. ]
 [ 11.  11.5  12.  12. ]
 [ 12.  12.5  13.  13. ]
 [ 12.  12.5  13.  13. ]]
```
Edge Linking: Hysteresis

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
  - use a high threshold to start edge curves and a low threshold to continue them.
Where is next edge point?

we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points
Where is next edge point?

we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points
Edge Linking: Hysteresis

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
  - use a high threshold to start edge curves and a low threshold to continue them.
Edge Linking: Hysteresis

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.
Edge Linking: Hysteresis

threshold_high     threshold_low
0 1 0 0 0           0 1 0 0 0
0 1 0 1             0 1 0 1
0 0 0 0             0 1 0 0
0 1 0 0             0 1 0 1
0 1 0 0             0 1 0 1
Edge Linking: Hysteresis

threshold_high  threshold_low  hysteresis

0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1

0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1

arrow
Canny Edge Detection

1. Filter image by derivatives of Gaussian
2. Compute magnitude of gradient
3. Compute edge orientation
4. Detect local maximum
5. Edge linking
Canny Edge Implementation

```matlab
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, -1];
dy = [1; -1];
```
Canny Edge Implementation

% % 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');
ly = conv2(img, Gy, 'same');

Ix  ly
Canny Edge Implementation

\[ \text{angle} = \text{atan2}(I_y, I_x); \]

%%% Edge angle conditioning
\[
\text{angle}(\text{angle}<0) = \pi + \text{angle}(\text{angle}<0);
\]
\[
\text{angle}(\text{angle}>7\pi/8) = \pi - \text{angle}(\text{angle}>7\pi/8);
\]

% Edge angle discretization into 0, \(\pi/4\), \(\pi/2\), \(3\pi/4\)
\[
\text{angle}(\text{angle}>=0&\text{angle}<\pi/8) = 0;
\]
\[
\text{angle}(\text{angle}>=\pi/8&\text{angle}<3\pi/8) = \pi/4;
\]
\[
\text{angle}(\text{angle}>=3\pi/8&\text{angle}<5\pi/8) = \pi/2;
\]
\[
\text{angle}(\text{angle}>=5\pi/8&\text{angle}<=7\pi/8) = 3\pi/4;
\]

Continuous angle  Discretized angle
Canny Edge Implementation

angle = np.arctan2(Iy, lx);

%% Edge angle conditioning
angle[angle<0] = np.pi+angle[angle<0];
angle[angle>7*np.pi/8] = np.pi-angle[angle>7*np.pi/8];

% Edge angle discretization into 0, pi/4, pi/2, 3*pi/4
angle[np.logical_and(angle>=0,angle<np.pi/8)] = 0;
angle[np.logical_and(angle>=np.pi/8,angle<3*np.pi/8)] = np.pi/4;
angle[np.logical_and(angle>=3*np.pi/8,angle<5*np.pi/8)] = np.pi/2;
angle[np.logical_and(angle>=5*np.pi/8,angle<=7*np.pi/8)] = 3*np.pi/4;
Canny Edge Implementation

% Calculate magnitude
magnitude = sqrt(lx.*lx+ly.*ly);
edge = zeros(nr, nc);

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

edge = edge.*magnitude;

Gradient magnitude
Localized edge
Canny Edge Implementation

```matlab
%% 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);
```

![Hysteresis Thresholding Diagram](image)
Canny Edge Implementation

```matlab
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 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 -1];
dy = [1; -1];
```
Canny Edge Implementation

% % 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');
Canny Edge Implementation

angle = atan2(Iy, Ix);
mag = sqrt(Iy.^2 + Ix.^2);
Canny Edge Implementation

```matlab
%% Non-Maximum Suppression
edge = non_maximum_suppression(magnitude, angle, edge);

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, 'same');
Gy = conv2(G, dy, 'same');

% 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);

Gradient Magnitude
Canny Edge Implementation

\[
\text{angle} = \text{atan2}(\text{ly}, \text{lx})
\]

\[
\text{mag} = \sqrt{\text{ly}^2 + \text{lx}^2}
\]
Canny Edge Implementation

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

low = threshold_low * max(edge(:));
high = threshold_high * max(edge(:));
linked_edge = hysteresis_thresholding(low, high);
Convolution of image with Gaussian:

\[ G_x = \text{conv2}(G, dx, 'same'); \]
\[ G_y = \text{conv2}(G, dy, 'same'); \]

Convolution of image with \( G_x \) and \( G_y \):

\[ I_x = \text{conv2}(\text{img}, G_x, 'same'); \]
\[ I_y = \text{conv2}(\text{img}, G_y, 'same'); \]
Canny Edge Implementation

```matlab
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];
```
Canny Edge Implementation

% % 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

\[
\begin{align*}
\text{angle} &= \text{atan2}(\text{l}_y, \text{l}_x); \\
\text{mag} &= \sqrt{\text{l}_y^2 + \text{l}_x^2};
\end{align*}
\]
Canny Edge Implementation

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

Localized edge

gradient

threshold_high = threshold_low * max(edge(:));
threshold_low = 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');
The rows of black and white squares are all parallel.
The vertical zigzag patterns disrupt our horizontal perception.
Different scale of image encodes different edge response.
Image Pyramids

Known as a Gaussian Pyramid [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*