Edge Detection

Code

\[
J_b = \text{rgb2gray}(J);
\]

\[
\text{imagesc}(J_b); \text{axis image}; \text{colormap(gray)};
\]

\[
bw = \text{edge}(J_b,'canny');
\]
Numerical Image Filtering

Looping through all pixels

\[ [nr, nc] = \text{size}(Jb); \]
\[ J\_out = \text{zeros}(nr, nc); \]
\[ \text{for } i=1:nr, \]
\[ \quad \text{for } j=1:nc; \]
\[ \quad \quad \text{if } (i<nr) \&\& (i>1), \]
\[ \quad \quad \quad J\_out(i,j) = 2*Jb(i,j) - 0.8*Jb(i+1,j) - 0.8*Jb(i-1,j); \]
\[ \quad \quad \text{else} \]
\[ \quad \quad \quad J\_out(i,j) = Jb(i,j); \]
\[ \quad \text{end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{figure; imagesc}(J\_out); \text{colormap(gray)}; \]

Computation time: 0.050154 sec
Numerical Image Filtering
Convolution without Looping using meshgrid

>> [x,y] = meshgrid(1:5,1:3)

x =

1  2  3  4  5
1  2  3  4  5
1  2  3  4  5

y =

1  1  1  1  1
2  2  2  2  2
3  3  3  3  3
\[ [x,y] = \text{meshgrid}(1:nc,1:nr); \]

\text{figure}(1); \text{imagesc}(x); \text{axis image}; \text{colorbar}; \text{colormap(jet)};

\text{figure}(2); \text{imagesc}(y); \text{axis image}; \text{colorbar}; \text{colormap(jet)};
Convolution without Looping using meshgrid

```matlab
[x,y] = meshgrid(1:nc,1:nr);
y_up = y-1;
y_down = y+1;
y_up = min(nr,max(1,y_up));  % keep y_up index within legal range of [1,nr]
y_down = min(nr,max(1,y_down));

ind_up = sub2ind([nr,nc],y_up(:),x(:));  % create linear index
ind_down = sub2ind([nr,nc],y_down(:),x(:));

J_out = 2*Jb(:) - 0.8*Jb(ind_up) - 0.8*Jb(ind_down);
J_out = reshape(J_out, nr, nc);

figure; imagesc(J_out);colormap(gray)

Computation time: 0.024047 sec
```
Convolution without Looping using meshgrid

\[ [x, y] = \text{meshgrid}(1:\text{nc}, 1:\text{nr}); \]

\[ \]

\( x \) and \( y \) are subscript indice.

\[ J_{b_{xy}} \]

\[ J_{out} = 2*J_{b} - 0.8*J_{b_{up}} - 0.8*J_{b_{down}}; \]

\[ J_{out} = \text{reshape}(J_{out}, nr, nc); \]

\[ \text{figure;} \]

\[ \text{imagesc}(J_{out}); \text{colormap(gray)} \]
Convolution without Looping using meshgrid

\[ [x,y] = \text{meshgrid}(1:nc,1:nr); \]

\[ y\_up = y-1; \]
\[ y\_down = y+1; \]
Convolution without Looping using meshgrid

\[ y_{\text{up}} = y - 1; \]
\[ y_{\text{down}} = y + 1; \]

\[ y_{\text{up}} = \min(nr, \max(1, y_{\text{up}})); \quad \% \text{keep } y_{\text{up}} \text{ index within legal range of } [1, nr] \]
\[ y_{\text{down}} = \min(nr, \max(1, y_{\text{down}})); \]

\[ y_{\text{up}} = \scriptsize \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix} \quad y = \scriptsize \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix} \quad y_{\text{down}} = \scriptsize \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix} \]
Convolution without Looping using meshgrid

\[
y_{\text{up}} = \min(nr, \max(1, y_{\text{up}})); \quad \% \text{keep } y_{\text{up}} \text{ index within legal range of } [1, nr]
\]

\[
y_{\text{down}} = \min(nr, \max(1, y_{\text{down}}));
\]

\[
\text{ind}_{\text{up}} = \text{sub2ind}([nr, nc], y_{\text{up}}(:), x(:)); \quad \% \text{create linear index}
\]

\[
\text{ind}_{\text{down}} = \text{sub2ind}([nr, nc], y_{\text{down}}(:), x(:));
\]

\[
\text{linear}_{\text{index}} = \text{sub2ind}([n_{\text{row}}, n_{\text{col}}], \text{row}_{\text{subscript}}, \text{col}_{\text{subscript}})
\]
Convolution without Looping using meshgrid

\[ y_{\text{up}} = \min(nr, \max(1, y_{\text{up}})) \text{; } \% \text{ keep } y_{\text{up}} \text{ index within legal range of } [1, nr] \]

\[ y_{\text{down}} = \min(nr, \max(1, y_{\text{down}})) \]

\[ \text{ind}_{\text{up}} = \text{sub2ind}([nr, nc], y_{\text{up}}(:), x(:)) \% \text{ create linear index} \]

\[ \text{ind}_{\text{down}} = \text{sub2ind}([nr, nc], y_{\text{down}}(:), x(:)) \]

\[ \text{linear}_{\text{index}} = \text{sub2ind}([n_{\text{row}}, n_{\text{col}}], \text{row}_{\text{subscript}}, \text{col}_{\text{subscript}}) \]

\[ J_{\text{out}} = 2*J_b(:) - 0.8*J_b(\text{ind}_{\text{up}}) - 0.8*J_b(\text{ind}_{\text{down}}) \]

\[ J_{\text{out}} = \text{reshape}(J_{\text{out}}, nr, nc) \]

\[ \text{figure; } \text{imagesc}(J_{\text{out}}); \text{colormap(gray)} \]

\[ \text{A= 4} \]

\[ A_{32} = \text{A(12)} = \text{A(7)} \]

\[ 7 = \text{sub2ind([4 3], 3,2)} \]

\[ \text{ind}_{\text{up}} = \text{sub2ind}([nr, nc], y_{\text{up}}(:), x(:)) \]

\[ \text{Operation on vectors } J_{b_{xy}} = J_{b_{\text{idx}}} \]
Convolution without Looping using meshgrid

\[ J_{out} = 2 \cdot J_{b}() - 0.8 \cdot J_{b}(\text{ind\_up}) - 0.8 \cdot J_{b}(\text{ind\_down}); \]
\[ J_{out} = \text{reshape}(J_{out}, nr, nc); \]

\[ J_{out} = \text{imagesc}(J_{out}); \text{colormap(gray)} \]
With loop

Without loop

Computation time: 0.050154 sec

Computation time: 0.024047 sec
Canny Edge Detection

Objective: to localize edges given an image.

Binary image indicating edge pixels

\[ B(i,j) = \begin{cases} 
1 & \text{if } I(i,j) \text{ is edge} \\
0 & \text{if } I(i,j) \text{ is not edge} 
\end{cases} \]

Original image, I

Edge map image, B
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
1) **Compute Image Gradient**

the first order derivative of Image I in x, and in y direction
Edge Detection, Step 1,
Filter out noise and compute derivative:

\[
\left( \frac{\partial}{\partial x} \ast G \right)
\]

Gradient of Gaussian
Edge Detection, Step 1,
Filter out noise and compute derivative:

\[ \otimes \left( \frac{\delta}{\delta x} \otimes G \right) \]

Image

Smoothed Derivative

\[ I_x \]

\[ I_y \]
Edge Detection, Step 1, Filter out noise and compute derivative:

In matlab:

```matlab
>> [dx,dy] = gradient(G); % G is a 2D gaussian
>> Ix = conv2(I,dx,'same'); ly = conv2(I,dy,'same');
```
Edge Detection: Step 2
Compute the magnitude of the gradient

In Matlab:
$$\text{Im} = \sqrt{I_x \cdot I_x + I_y \cdot I_y};$$
We know roughly where are the edges, but we need their precise location.
Finding the orientation of the edge

- The gradient of an image:
  \[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \]

- The gradient points in the direction of most rapid change in intensity
  \[ \nabla f = [\frac{\partial f}{\partial x}, 0] \]
  \[ \nabla f = [0, \frac{\partial f}{\partial y}] \]

- The image gradient direction is given by:
  \[ \theta = \tan^{-1} \left( \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right) \]
  – how does this relate to the direction of the edge?
  \[ \theta_{edge} = \tan^{-1} \left( -\frac{\delta f}{\delta x} / \frac{\delta f}{\delta y} \right) \]
%% 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');
\[ [gx,gy] = \text{gradient}(J); \]
\[ \text{mag} = \sqrt{gx.\ast gx+gy.\ast gy}; \quad \text{imagesc(mag);colorbar} \]
image gradient direction:
Edge orientation direction:
[gx,gy] = gradient(J);

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

(Forsyth & Ponce)
Thesholding

(Forsyth & Ponce)
Non-maximum suppression along the line of the gradient.
Gradient direction

(Forsyth & Ponce)
Local maximum
No intensity values at r and p: Interpolate these intensities using neighbor pixels.

Where is next edge point?
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 1 0 1 0 1 0 1
0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 1

Diagram of edge linking with hysteresis.
Edge Linking: Hysteresis

threshold_high  threshold_low  hysteresis

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 0 0 0 0 1 0 0 0 1 0 1
0 1 0 0 0 1 0 1 0 1 0 0
0 1 0 0 0 1 0 0 0 1 0 0
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 0 -1];
dy = [1; 0; -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

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

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

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

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

edge = edge.*magnitude;

Gradient magnitude

Localized edge
%% 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);
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('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];
```

---

**Gaussian Filter Definition**

The Gaussian filter is defined as:

\[
G = \begin{bmatrix}
2 & 4 & 5 & 4 & 2 \\
4 & 9 & 12 & 9 & 4 \\
5 & 12 & 15 & 12 & 5 \\
4 & 9 & 12 & 9 & 4 \\
2 & 4 & 5 & 4 & 2
\end{bmatrix}
\]

The filter is normalized to have a total of 159 elements.

---

**Filter for Horizontal and Vertical Directions**

The filters for horizontal and vertical directions are defined as:

\[
dx = [1 -1] \\
dy = [1; -1]
\]
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

%% 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);

NMS
gradient
threshold_high
threshold_low
hysteresis
Localized edge
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

\[
\text{angle} = \text{atan2}(I_y, I_x);
\]
\[
\text{mag} = \sqrt{I_y^2 + I_x^2};
\]
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

\[
\text{angle} = \text{atan2}(I_y, I_x);
\]
\[
\text{mag} = \sqrt{I_y^2 + I_x^2};
\]
Canny Edge Implementation

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

Localized edge

```c
gradient
```

```c
threshold_high
threshold_low
```

```c
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');
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