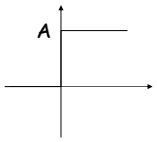
Image Features

Jana Kosecka, GMU

Edges

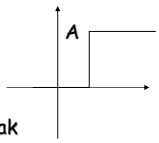
Edge Model (1D)

- An ideal edge can be modeled as an step

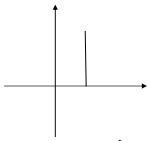


$$G(x) = \begin{cases} 0 \text{ if } x < 0 \\ A \text{ if } x \leq 0 \end{cases}$$

1-D edge detection



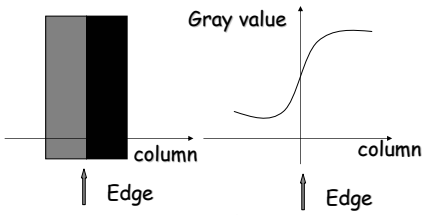
Derivative – ideal peak



In reality – image edges are not sharp and regions around edges have noise

Edges

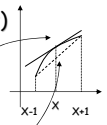
- They happen at places where the image values exhibit sharp variation



Digital Approximation

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$\frac{df(x)}{dx} \cong \frac{f(x+1) - f(x-1)}{2}$$



⇒ Convolve with:

-1	0	1

Simple way to compute derivatives

- How to compute derivatives in discrete images
- Approximate it with differences

$$\Delta_x I = I(x,y) - I(x-1,y)$$
$$\Delta_y I = I(x,y) - I(x,y-1)$$

- Corresponds to convolution with a filter (mask) [1, -1]
- Alternatively we can use [-1, 1]
- In order to obtain value at the center pixel convolve with [1, 0, -1]
- Another option Robert's mask

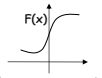
  0 1 1 0
  -1 0 0 -1

Computing Derivatives in 2D

- commonly used in feature extraction stage
- Computing derivatives of continuous functions – well established
- Think of an image as function $I(x,y)$
- Computing derivative of 2D function $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$
- Derivative – gradient – 2D vector with components of partial derivatives $\nabla I = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]$
- Magnitude $M = \sqrt{(\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2}$
- Orientation $\theta = \tan^{-1}(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$

Edge Detection (2D)

1D

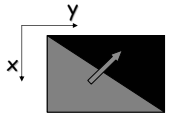


I(x)

$\frac{dI(x)}{dx}$

$\frac{dI(x)}{dx}$ > Th

2D



I(x,y)

$\nabla I(x,y) = [\partial I(x,y)/\partial x \; \partial I(x,y)/\partial y]^\dagger = [I_x(x,y) \; I_y(x,y)]^\dagger$

$|\nabla I(x,y)| = (I_x^2(x,y) + I_y^2(x,y))^{1/2} > Th$

$\tan\theta = I_x(x,y) / I_y(x,y)$

Edge Detection (2D)

Vertical Edges:

Convolve with:

-1	0	1

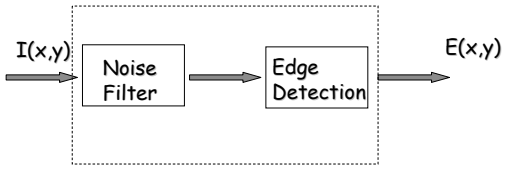
Horizontal Edges:

Convolve with:

-1
0
1

Noise cleaning and Edge Detection

Taking differences – increases noise



Combine Linear Filters

Noise Smoothing & Edge Detection

Convolve with:

-1	0	1
-1	0	1
-1	0	1

Noise Smoothing

Vertical Edge Detection

This mask is called the (vertical) Prewitt Edge Detector

Noise Smoothing & Edge Detection

Convolve with:

-1	-1	-1
0	0	0
1	1	1

Horizontal Edge Detection

Noise Smoothing

This mask is called the (horizontal) Prewitt Edge Detector

Sobel Edge Detector

Convolve with:

-1	0	1
-2	0	2
-1	0	1

and

-1	-2	-1
0	0	0
1	2	1

Gives more weight to the 4-neighbors

Smoothing and derivatives

- Derivatives emphasize noise – smoothing images before computing derivatives
- Instead of smoothing with box filter – averages everything equally – used Gaussian
- Another motivation for Gaussian – serves as interpolating function to compensate for discrete sampling effects

Computing Derivatives - Theory

- commonly used in feature extraction stage
- Computing derivatives of continuous functions – well established
- continuous function related to discrete by sampling
- If Nyquist condition is met the continuous signal can be reconstructed exactly by conv. (ideal sync)

$$f(x) = f[x] * h(x), \quad x \in \Re \qquad h(x) = \frac{\sin(\pi x/T)}{\pi x/T}, \quad x \in \Re$$

Strategy

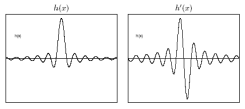
1. reconstruct continuous function
2. take a derivative
3. sample the result

Computing derivatives

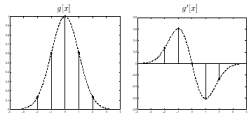
$$D\{f(x)\} = D\{f[x] * h(x)\}$$

$$D\{f(x)\} = f[x] * D\{h(x)\}$$ differentiation

$$S\{f'(x)\} = S\{f[x] * D\{h(x)\}\} = f[x] * S\{h'(x)\} = f[x] * h'[x]$$ sampling



Infinite extent



Approximation with Gaussian

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}, \quad g'(x) = -\frac{x}{\sigma^2\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}.$$

Computing Derivatives

- Images should be smoothed prior computing derivatives
- Convolution with a smoothing filter, followed by derivative filter
- Can be accomplished in one step, convolution with the derivative of the smoothing filter

$$f'[x] = f[x] * g'[x] = \sum_{k=-\frac{w}{2}}^{k=\frac{w}{2}} f[x]g'[x-k]\}$$

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}, \quad g'(x) = -\frac{x}{\sigma^2\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}.$$

Gaussian derivatives

- recipe for computing derivatives in x and y
- if using derivatives of Gaussians

- smooth in y and compute derivative in x
- smooth in x and compute derivative in y

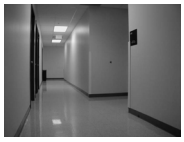
$$I_x[x,y] = I[x,y] * g'[x] * g[y] = \sum_{k=-\frac{w}{2}}^{\frac{w}{2}} \sum_{l=-\frac{w}{2}}^{\frac{w}{2}} I[k,l]g'[x-k]g[y-l],$$

$$I_y[x,y] = I[x,y] * g[x] * g'[y] = \sum_{k=-\frac{w}{2}}^{\frac{w}{2}} \sum_{l=-\frac{w}{2}}^{\frac{w}{2}} I[k,l]g[x-k]g'[y-l].$$

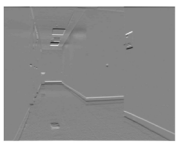
- This all can be realized by 1-D convolutions
- Gaussian is a separable filter

$$g_\sigma(x,y) = g_\sigma(x)g_\sigma(y)$$

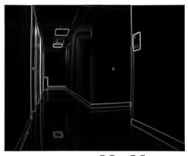
Computing derivatives



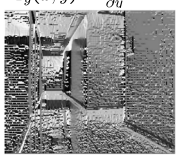
$I(x,y)$



$I_y(x,y) = \frac{\partial I}{\partial y}$



$\nabla I = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]$



$\theta = \tan^{-1}(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$

Formal Design of an Optimal Edge Detector

- Edge detection involves 3 steps:
  - Noise smoothing
  - Edge enhancement
  - Edge localization
- J. Canny formalized these steps to design an *optimal* edge detector

Canny Edge Detector

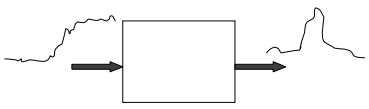
- Experiments consistently show that it performs very well
- Probably, the most used by C.V. practitioners

Canny Edge Detector

- Uses a mathematical model of the edge and the noise
- Formalizes a performance criteria
- Synthesizes the best filter

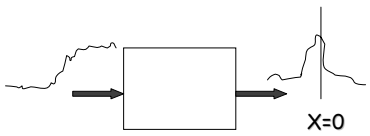
Performance Criteria (1)

- Good detection
  - The filter must have a stronger response at the edge location (x=0) than to noise



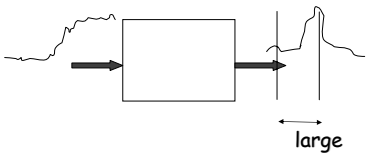
Performance Criteria (2)

- Good Localization
  - The filter response must be maximum very close to x=0



Performance Criteria (3)

- Low False Positives
  - There should be only one maximum in a reasonable neighborhood of x=0



Canny Edge Detector

- Canny found a linear, continuous filter that maximized the three given criteria.
- There is no close-form solution for the optimal filter.
- However, it looks VERY SIMILAR to the derivative of a Gaussian.

Algorithm CANNY_ENHANCER

- The input is image I; G is a zero mean Gaussian filter (std = σ)

  1. J = I * G (smoothing)
  2. For each pixel (i,j): (edge enhancement)
     - Compute the image gradient
       - » $\nabla J(i,j) = (J_x(i,j),J_y(i,j))'$
     - Estimate edge strength
       - » $e_s(i,j) = (J_x^2(i,j)+ J_y^2(i,j))^{1/2}$
     - Estimate edge orientation
       - » $e_o(i,j) = \arctan(J_x(i,j)/J_y(i,j))$
- The output are images E_s and E_o

CANNY_ENHANCER

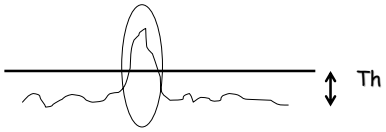
Edge strength – gradient magnitude

- The output image E_s has the magnitudes of the smoothed gradient.
- Sigma determines the amount of smoothing.
- E_s has large values at edges

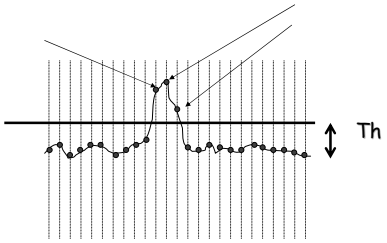
→ Edge *ENHANCER*

How do we "detect" edges?

- E_s has large values at edges:
  - Find local maxima



- … but it also may have wide ridges around the local maxima (large values *around* the edges)

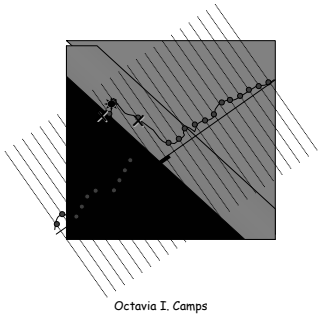


NONMAX_SUPRESSION

Edge orientation

- The inputs are E_s & E_o (outputs of CANNY_ENHANCER)
- Consider 4 directions D={ 0,45,90,135} wrt x
- For each pixel (i,j) do:
  1. Find the direction $d \in D$ s.t. $d \equiv E_o(i,j)$ (normal to the edge)
  2. If {$E_s(i,j)$ is smaller than at least one of its neigh. along d}
     - $I_N(i,j)=0$
     - Otherwise, $I_N(i,j)= E_s(i,j)$
- The output is the thinned edge image I_N

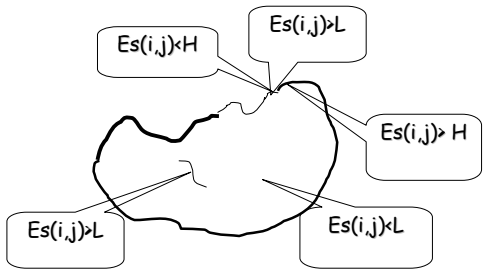
Graphical Interpretation



Thresholding

- Edges are found by thresholding the output of NONMAX_SUPRESSION
- If the threshold is too high:
  - Very few (none) edges
    - High MISDETECTIONS, many gaps
- If the threshold is too low:
  - Too many (all pixels) edges
    - High FALSE POSITIVES, many extra edges

SOLUTION: Hysteresis Thresholding



Strong edges reinforce adjacent weak edges

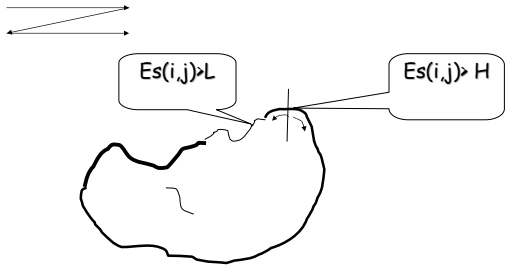
HYSTERESIS_THRESH

Inputs:
- I_N (output of NONMAX_SUPRESSION),
- E_o (output of CANNY_ENHANCER),
- thresholds L and H.

- For all pixels in I_N and scanning in a fixed order:
  1. Locate the next *unvisited* pixel s.t. $I_N(i,j)>H$
  2. Starting from $I_N(i,j)$, follow the chains of connected local maxima, in both directions perpendicular to the edge normal, as long as $I_N>L$.
     - Mark all visited points, and save the location of the contour points.

Output: a set of lists describing the contours.

Hysteresis Thresholding



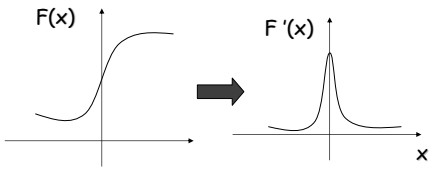
Canny Edge detector - Threshold

Other Edge Detectors

(2nd order derivative filters)

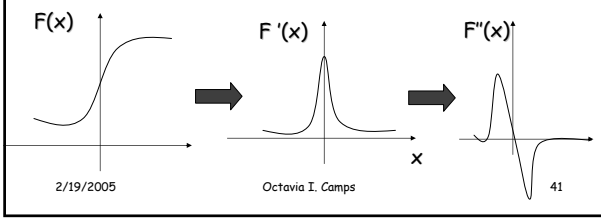
First-order derivative filters (1D)

- Sharp changes in gray level of the input image correspond to "peaks" of the first-derivative of the input signal.



Second-order derivative filters (1D)

- Peaks of the first-derivative of the input signal, correspond to "zero-crossings" of the second-derivative of the input signal.



NOTE:

- F''(x)=0 is not enough!
  - F'(x) = c has F''(x) = 0, but there is no edge
- The second-derivative must change sign, -- i.e. from (+) to (-) or from (-) to (+)
- The sign transition depends on the intensity change of the image – i.e. from dark to bright or vice versa.

Edge Detection (2D)

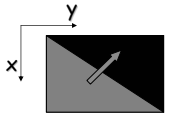
1D

I(x)

$\left|\frac{dI(x)}{dx}\right| > Th$

$\frac{d^2I(x)}{dx^2} = 0$

2D



I(x,y)

$|\nabla I(x,y)| = (I_x^2(x,y) + I_y^2(x,y))^{1/2} > Th$

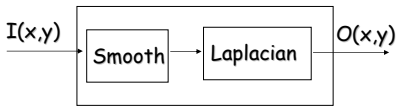
$\tan\theta = I_x(x,y)/ I_y(x,y)$

$\nabla^2 I(x,y) = I_{xx}(x,y) + I_{yy}(x,y) = 0$

Laplacian

Notes about the Laplacian:

- $\nabla^2 I(x,y)$ is a SCALAR
  - ↑ Can be found using a SINGLE mask
  - ↓ Orientation information is lost
- $\nabla^2 I(x,y)$ is the sum of SECOND-order derivatives
  - But taking derivatives increases noise
  - Very noise sensitive!
- It is always combined with a smoothing operation:

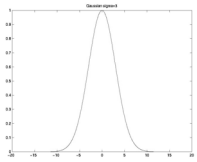


LOG Filter

- First smooth (Gaussian filter),
- Then, find zero-crossings (Laplacian filter):
  - $O(x,y) = \nabla^2(I(x,y) * G(x,y))$
- Using linearity:
  - $O(x,y) = \nabla^2 G(x,y) * I(x,y)$
  - This filter is called: "Laplacian of the Gaussian" (LOG)

1D Gaussian

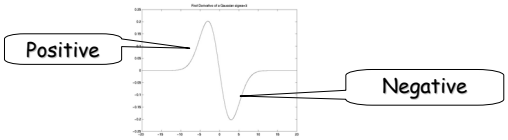
$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

First Derivative of a Gaussian

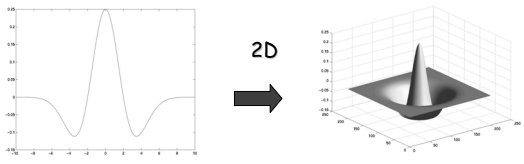
$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$



As a mask, it is also computing a difference (derivative)

Second Derivative of a Gaussian

$$g''(x) = (\frac{x^2}{\sigma^3} - \frac{1}{\sigma}) e^{-\frac{x^2}{2\sigma^2}}$$



"Mexican Hat"