EE421/521 Image Processing

Lecture 8
CORNER DETECTION
EDGE DETECTION
LINE IDENTIFICATION

1

- Today
  - Edge Detection
  - Segmentation
- Next Tuesday 14:40-16:30
  - More topics on edge detection and segmentstion
  - Classroom will be announced later
- Next Thursday
  - Midterm 2

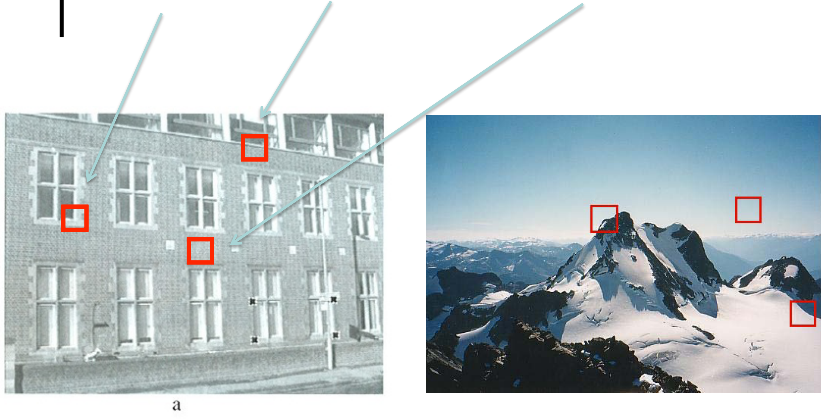
2

Edge Detection Applications

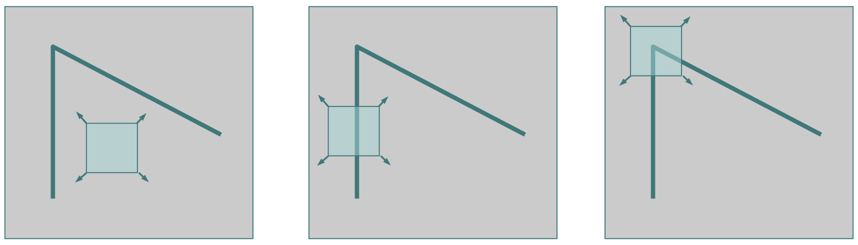
- Image alignment
- Object tracking
- Boundary identification
- Region segmentation

Corner Detection

Corners, Edges, Smooth Areas



Corners

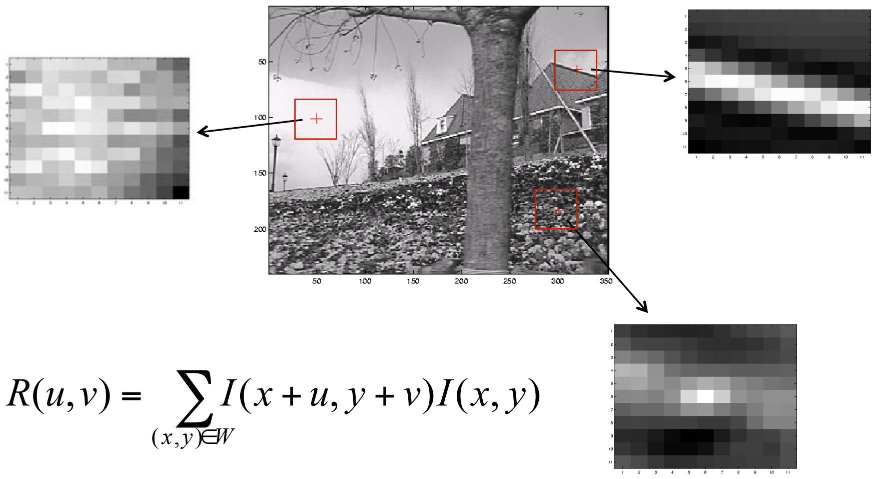


"flat" region:
no change in all
directions

"edge":
no change along
the edge direction

"corner":
significant change
in all directions

Autocorrelation: Indicator of Corners



$$R(u,v) = \sum_{(x,y)\in W} I(x+u, y+v) I(x,y)$$

Autocorrelation Calculation

$$R(u,v) = \sum_{(x,y)\in W} I(x+u, y+v) I(x,y)$$

Autocorrelation can be approximated by sum-squared-difference (SSD):

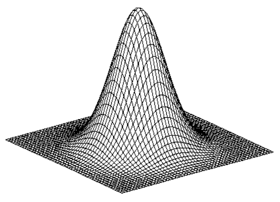
$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

SSD Calculation

Let $A = \begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix}$ and $u = \begin{bmatrix} u \\ v \end{bmatrix}$

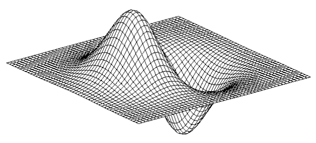
then $E(u,v) = u^T A u$

Use Sobel Operator for Gradient Computation



Gaussian

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

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

Horizontal derivative

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

Vertical derivative

Eigenvalues and Eigenvectors of the Auto-correlation Matrix

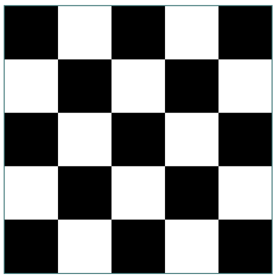
$$\overset{\text{lower limit}}{\lambda_-} \le E(u,v) = u^T A u \le \overset{\text{upper limit}}{\lambda_+}$$

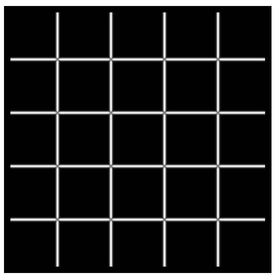
where λ_+ and λ_- are the two eigenvalues of A.

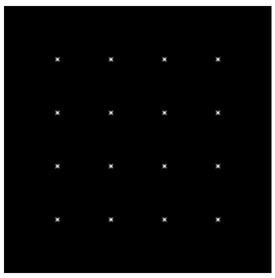
The eigenvector e_+ corresponding to λ_+ gives the direction of **largest** increase ***E***,

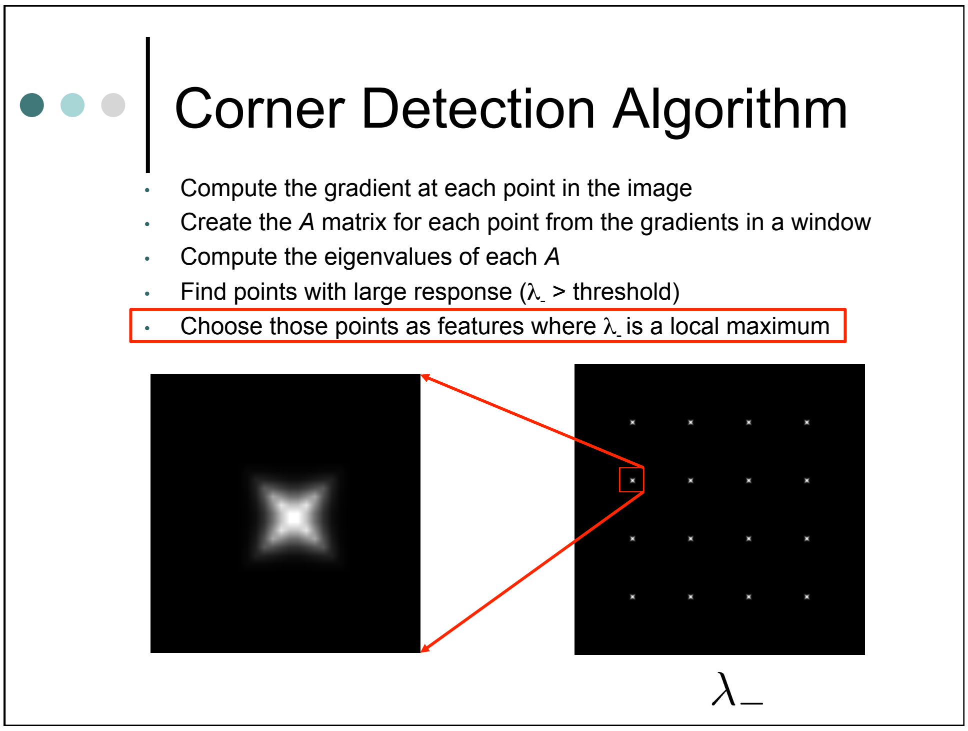
while the eigenvector e_- corresponding to λ_- gives the direction of **smallest** increase in ***E***.

λ_+ for Edges, λ_- for Corners


I

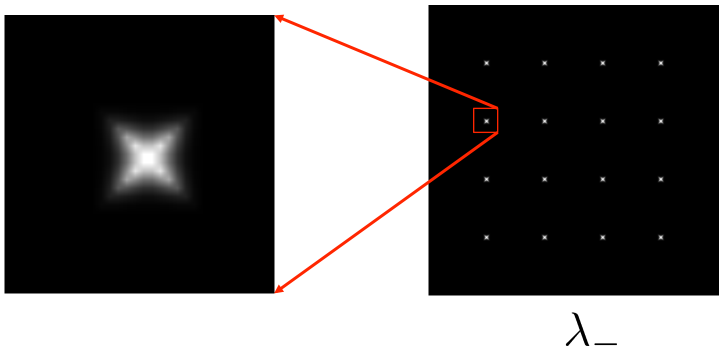

λ_+

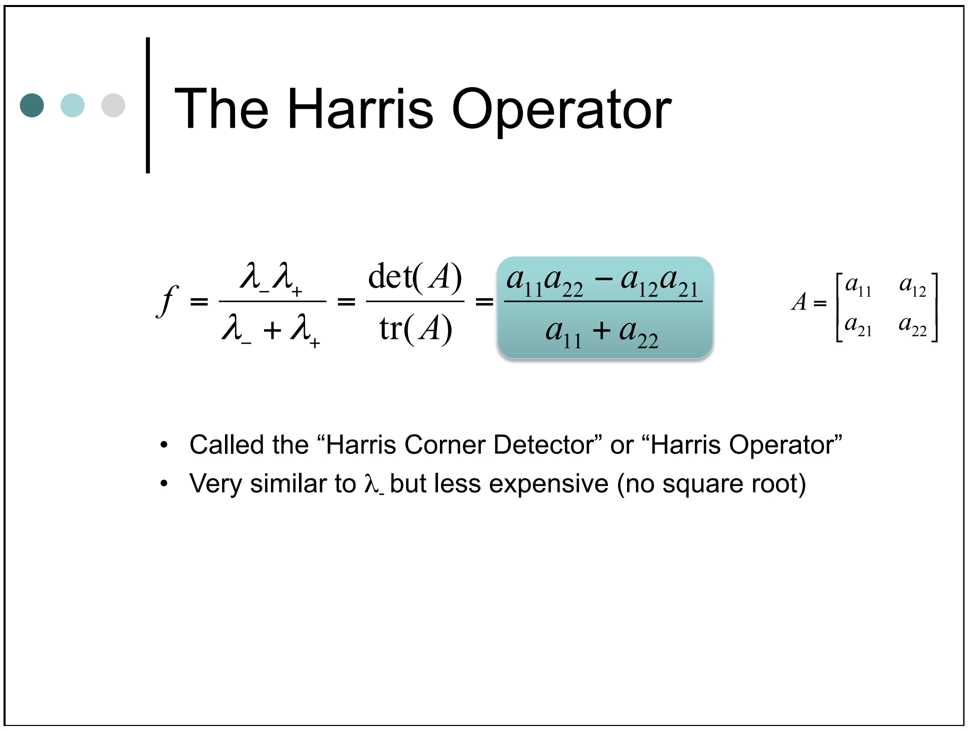

λ_-



Corner Detection Algorithm

- Compute the gradient at each point in the image
- Create the *A* matrix for each point from the gradients in a window
- Compute the eigenvalues of each *A*
- Find points with large response (λ_- > threshold)
- Choose those points as features where λ_- is a local maximum

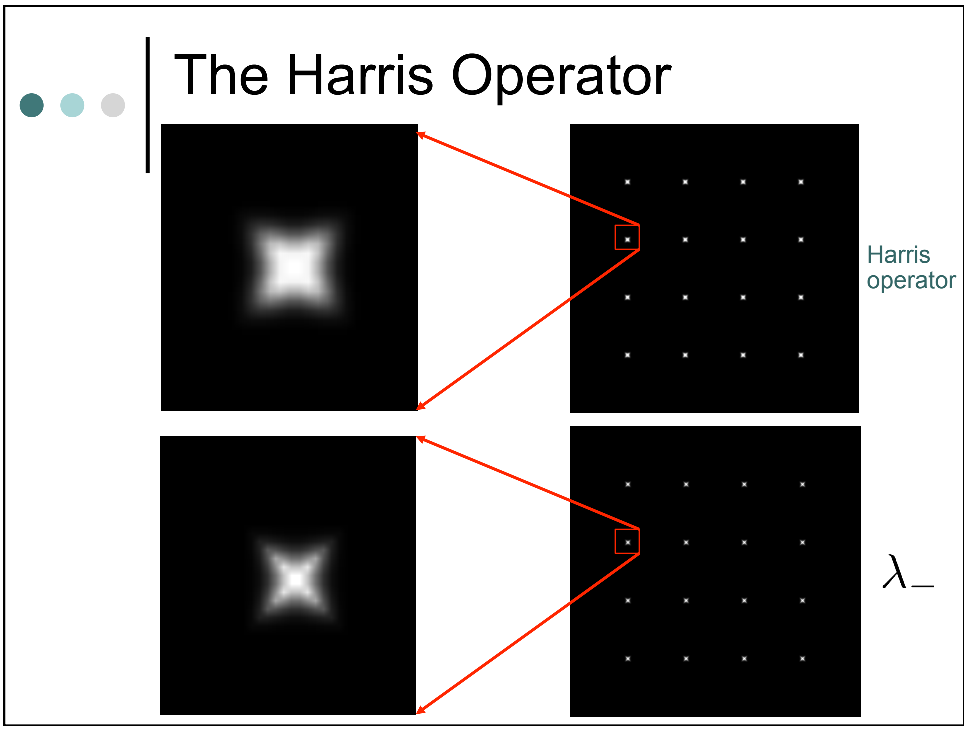




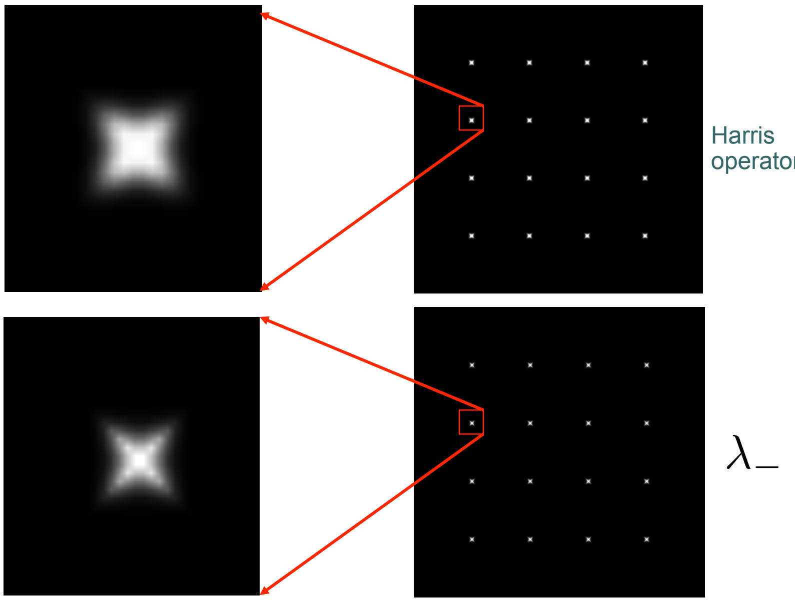
The Harris Operator

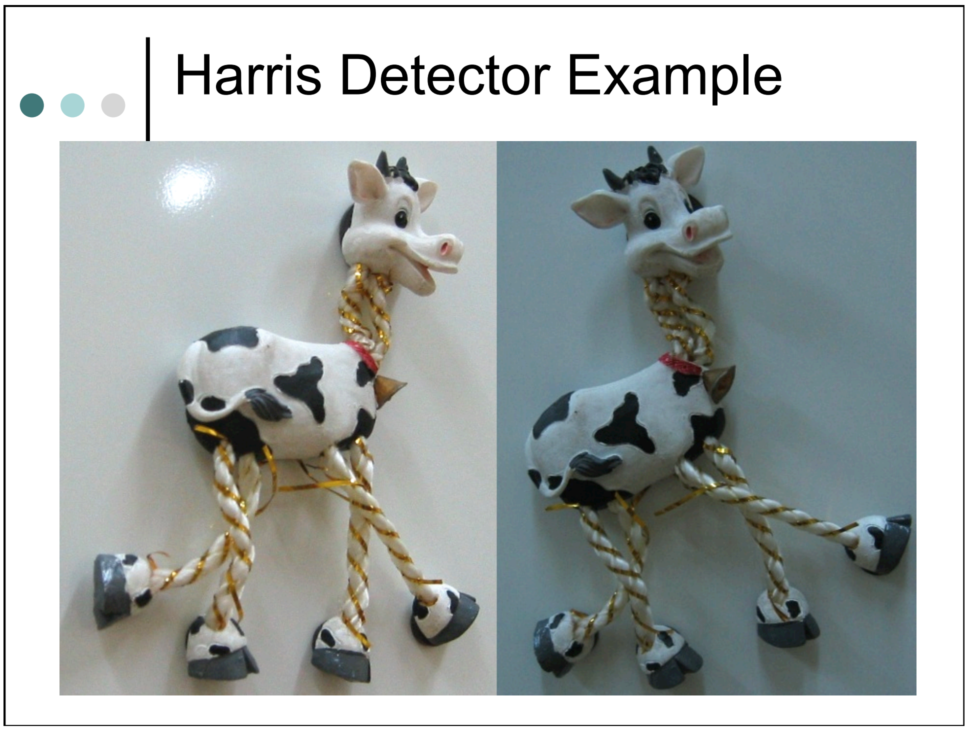
$$f = \frac{\lambda_- \lambda_+}{\lambda_- + \lambda_+} = \frac{\det(A)}{\operatorname{tr}(A)} = \boxed{\frac{a_{11}a_{22} - a_{12}a_{21}}{a_{11} + a_{22}}} \qquad A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

- Called the “Harris Corner Detector” or “Harris Operator”
- Very similar to λ_- but less expensive (no square root)



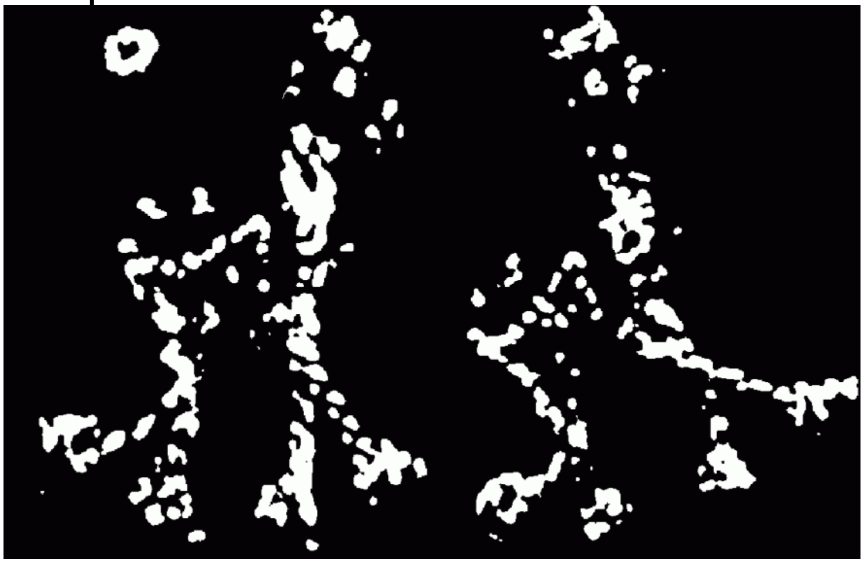
The Harris Operator



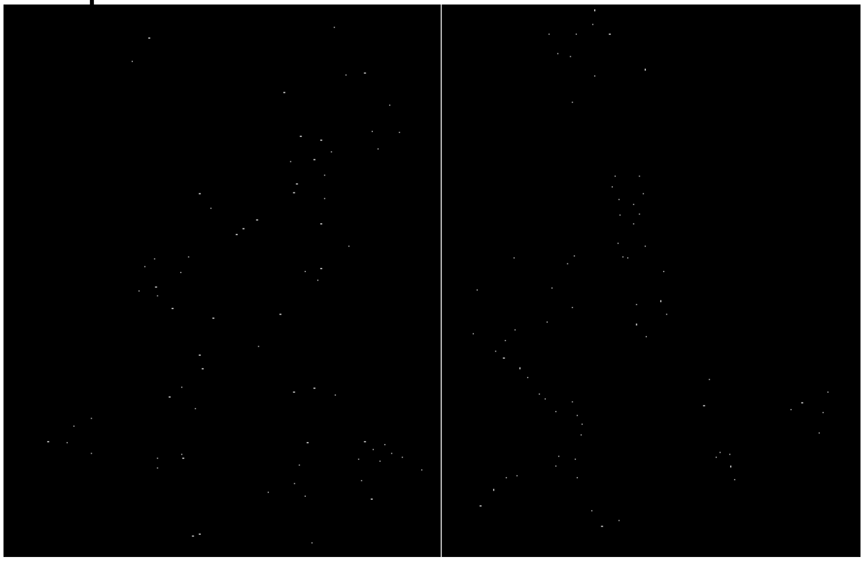


Harris Detector Example

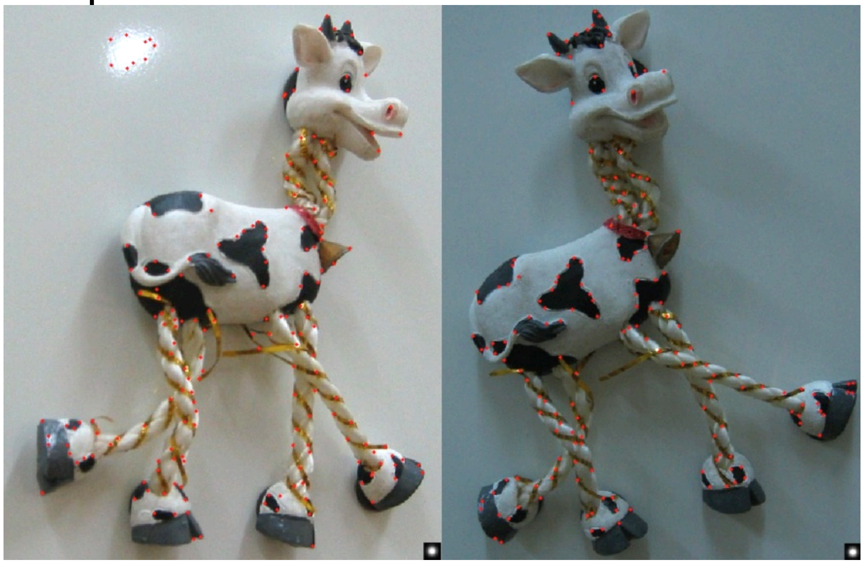
Threshold (*f* > value)



Find Local Maxima of *f*



Harris Features (in red)



Edge Detection

20

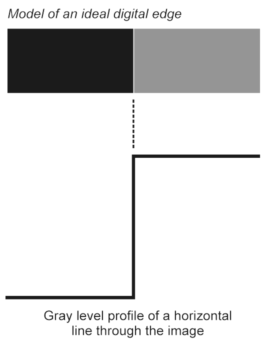
Slides mostly from Oge Marques, Copyright © 2011 by John Wiley & Sons, Inc..

What is Edge Detection?

- There is compelling evidence that the very early stages of the human visual system (HVS) contain **edge-sensitive** cells
- Goal of edge detection algorithms is to find the most relevant edges in an image.
- These edges could then be **connected** into meaningful lines and boundaries, resulting in a **segmented** image containing two or more regions.

Basic Concepts

- Edge: a boundary between two image regions having distinct characteristics according to some feature (e.g., gray level, color, or texture).
- In grayscale 2D images: a sharp variation of the intensity function across a portion of the image.


Model of an ideal digital edge
Gray level profile of a horizontal line through the image

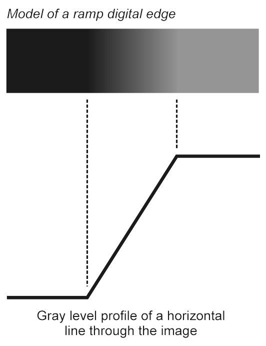
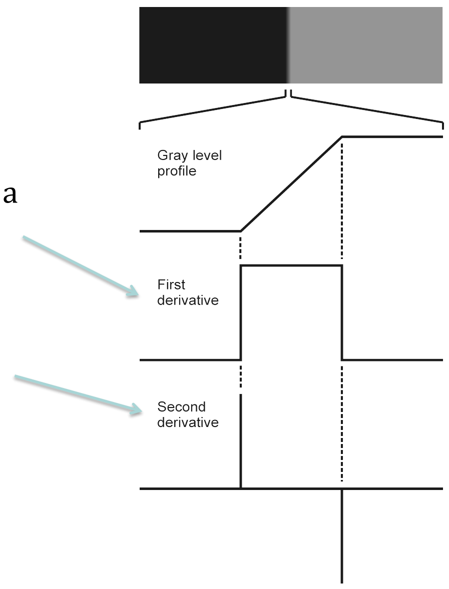

Model of a ramp digital edge
Gray level profile of a horizontal line through the image

Image Derivatives and Edges

- The magnitude of the **first derivative** can be used to detect the presence of an edge at a certain point in the image.
- The sign of **second derivative** can be used to determine whether a pixel lies on the dark or bright side of an edge.
  - Moreover, the **zero crossing** between the positive and negative peaks of the second derivative can be used to locate the **center** of thick edges.

Ramp Edge

- The **first derivative** has a peak at the center of the luminance edge.
- The **second derivative** has two peaks, with a positive value on the left and a negative value on the right.



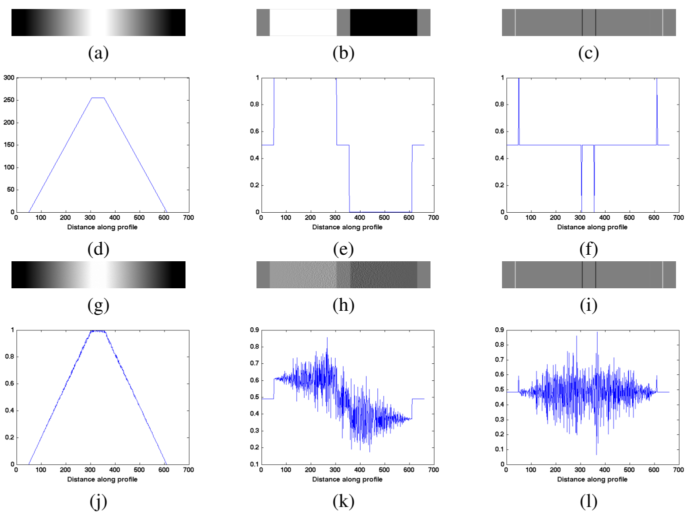
The Influence of Noise

(a),(b),(c)
Original image, first derivative, second derivative

(d), (e), (f)
Horizontal profiles

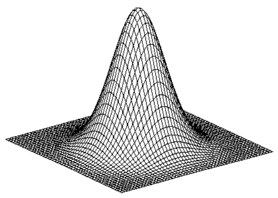
(g), (h), (i)
Noisy versions of images

(j), (k), (l)
Horizontal profiles



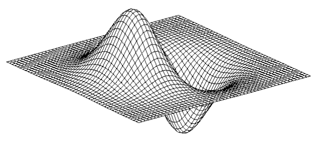
Even modest levels of noise can make the second order derivative useless.

2D Edge Detection Filters



Gaussian for Smoothing

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

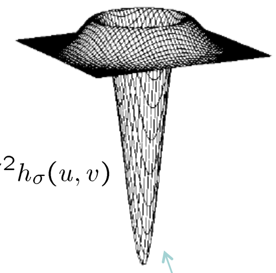


Derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

look for peaks

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

look for zero-crossings

Laplacian operator:

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

Prewit and Sobel Edge Detectors

- Compute derivatives in *x* and *y* directions
- Find gradient magnitude
- Threshold gradient magnitude

- Prewitt operator
  - Averaging to suppress noise

−1	0	1
−1	0	1
−1	0	1

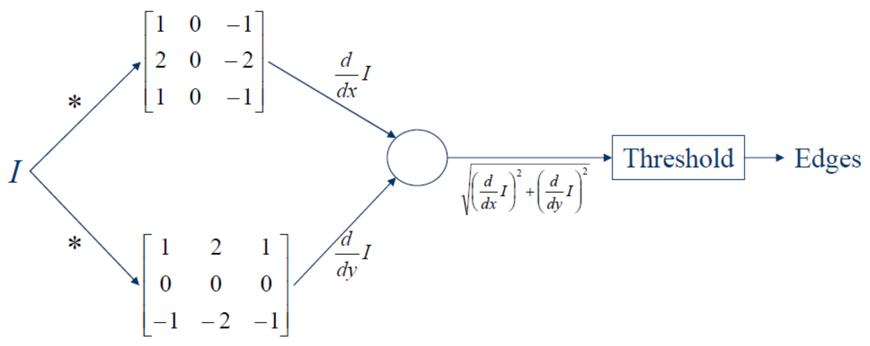
1	1	1
0	0	0
−1	−1	−1

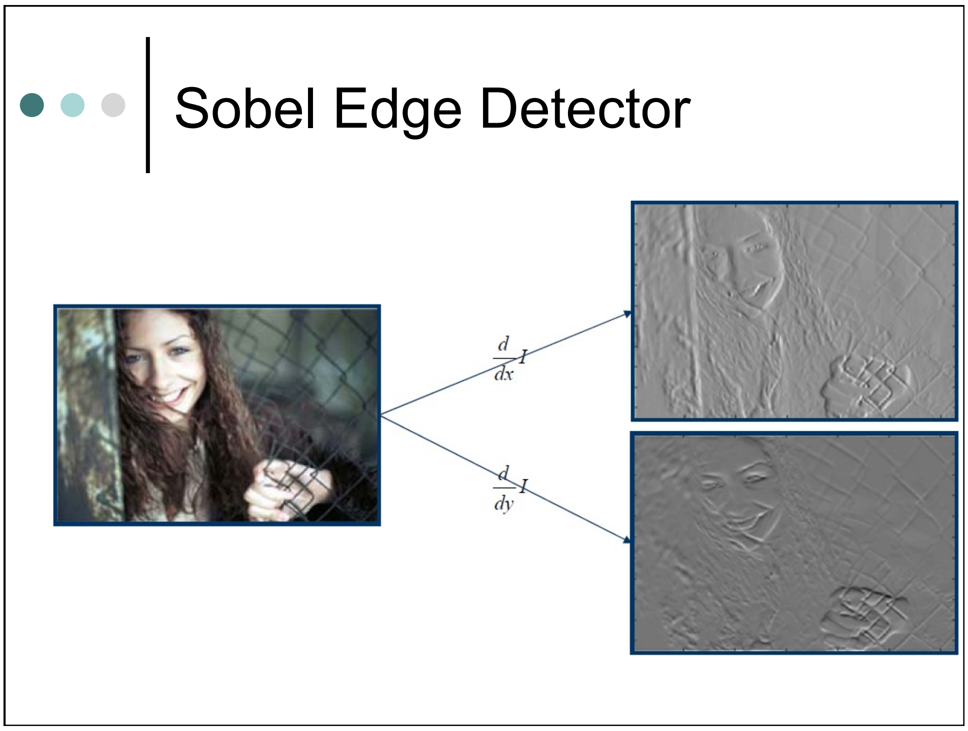
- Sobel operator
  - Averaging with emphasis to center pixels

−1	0	1
−2	0	2
−1	0	1

1	2	1
0	0	0
−1	−2	−1

Sobel Edge Detector

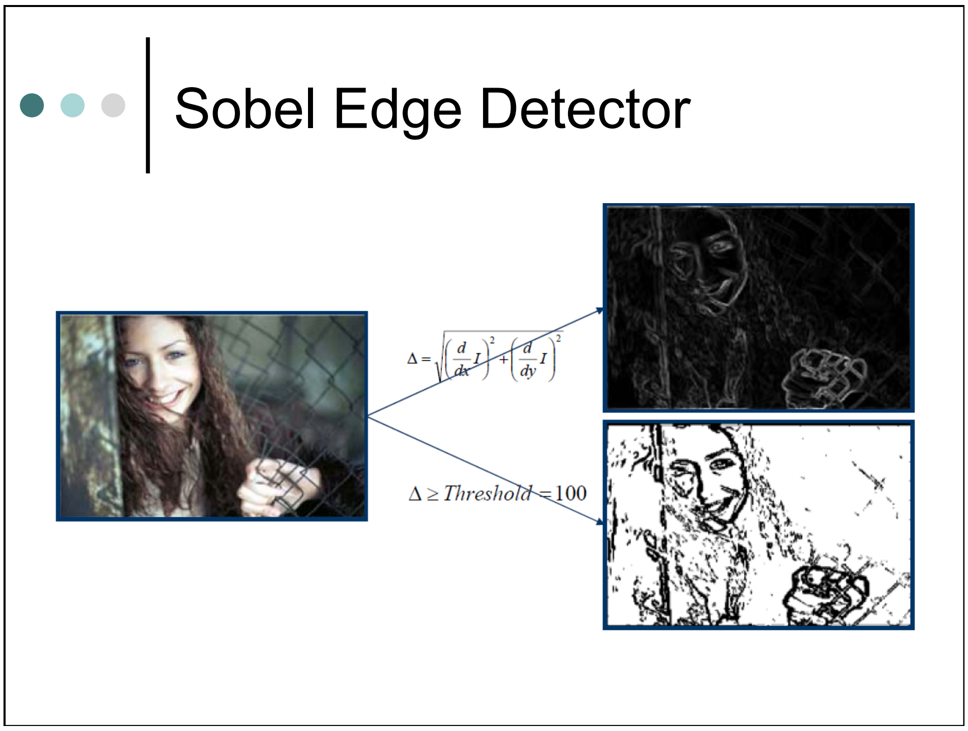

I
$*$
$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$
$\frac{d}{dx}I$
$*$
$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
$\frac{d}{dy}I$
$\sqrt{\left(\frac{d}{dx}I\right)^2+\left(\frac{d}{dy}I\right)^2}$
Threshold
Edges



Sobel Edge Detector

$\frac{d}{dx}I$

$\frac{d}{dy}I$

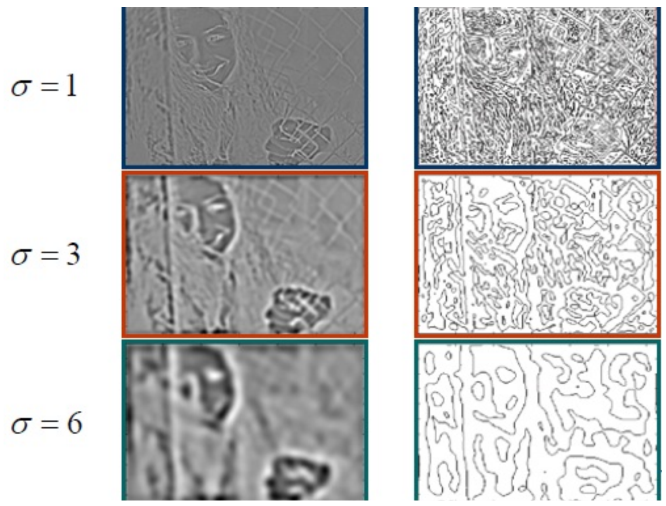


Sobel Edge Detector

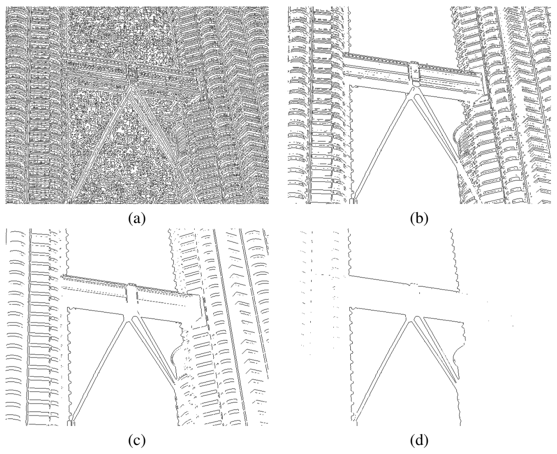
$\Delta = \sqrt{\left(\frac{d}{dx}I\right)^2 + \left(\frac{d}{dy}I\right)^2}$

$\Delta \geq Threshold = 100$

Effect of Smoothing (Noise Suppression)



Effect of Threshold



- (a) threshold of zero, too many spurious pixels
- (b) threshold of 0.05
- (c) threshold of 0.1
- (d) threshold of 0.2

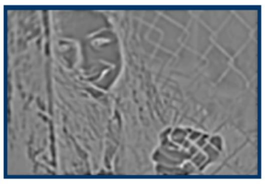
Zero Crossings of the Second Order Derivative

- Four cases of zero-crossings :
  - {+,-}
  - {+,0,-}
  - {-,+}
  - {-,0,+}
- Slope of zero-crossing {a, -b} is |a+b|.
- To mark an edge
  - compute slope of zero-crossing
  - Apply a threshold to slope

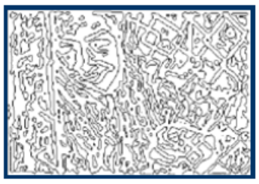
Laplacian Based Edge Detection

I

$I * \left(\Delta^2 g\right)$

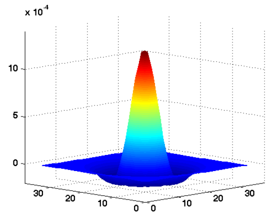


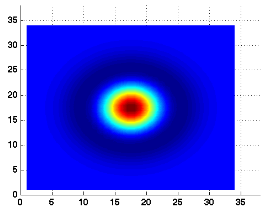
Zero crossings of $\Delta^2 S$



Laplacian of Gaussian

- Laplacian is rarely used in isolation because it is extremely sensitive to noise.
- **Laplacian of Gaussian** (LoG) works by smoothing the image with a Gaussian low-pass filter, and then applying a Laplacian to the result.
- Edge detection is achieved by LoG followed by zero-crossing detection





Laplacian of Gaussian $\nabla^2 G$

- Second derivative of the Gaussian function
- Gaussian part smooths the image:
  - Reduces noise
  - **Reduces image structures at scales much smaller than sigma.**
  - Gaussian is smooth in both spatial and frequency domains and does not have ringing artifact.
- Laplacian part:
  - Isotropic operator (invariant to rotation)
  - Corresponds to characteristics of human visual system
  - Responds equally to changes in intensity in any mask direction
  - No need to use multiple masks

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

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

$$= \frac{\partial}{\partial x}\left(\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right) + \frac{\partial}{\partial y}\left(\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right)$$

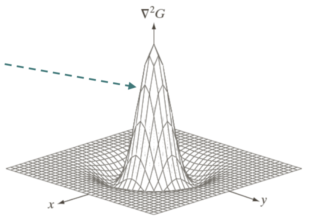
$$= \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} + \left[\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

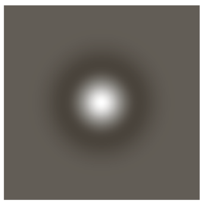
$$\boxed{\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}}$$

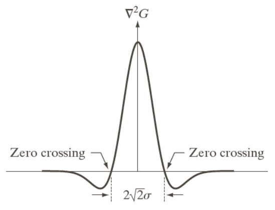
$$\begin{tikzcd} {} \arrow[r] & \boxed{\text{Gaussian}} \arrow[r] & \boxed{\text{Laplacian}} \arrow[r] & {} \arrow[r, dashed, "\text{is equivalent to}"'] & {} \arrow[r] & \boxed{\text{LoG}} \arrow[r] & {} \end{tikzcd}$$

Laplacian of Gaussian

- Negative of the LoG
- Zeros define a circle centered at origin.
- Also called as the **Mexican Hat** operator because of its shape.
- Can be approximated by a (5 x 5) mask
  - Not unique.
  - Capture the general shape (positive central term surrounded by negative terms, and zeros at the outer region)
  - Coefficients must sum to zero so that the response is zero at constant intensities.
  - Masks of arbitrary size can be generated by sampling the LOG function.







0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

zeros : $x^2 + y^2 - 2\sigma^2 = 0$ (define a circle of radius $\sqrt{2}\sigma$)

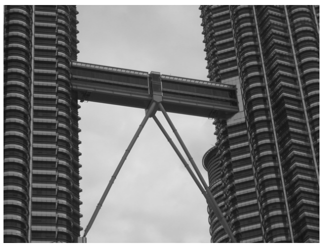
Laplacian Edge Detection Example

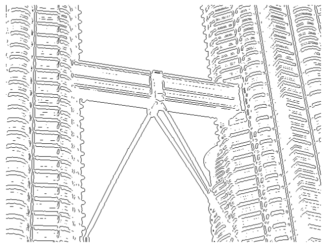
(a) Original image

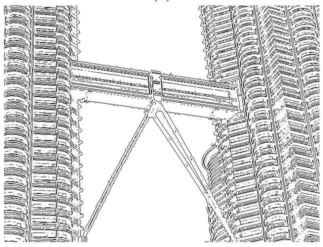
(b) sigma=2

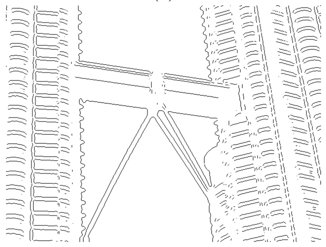
(c) sigma=1

(d) sigma=3


(a)

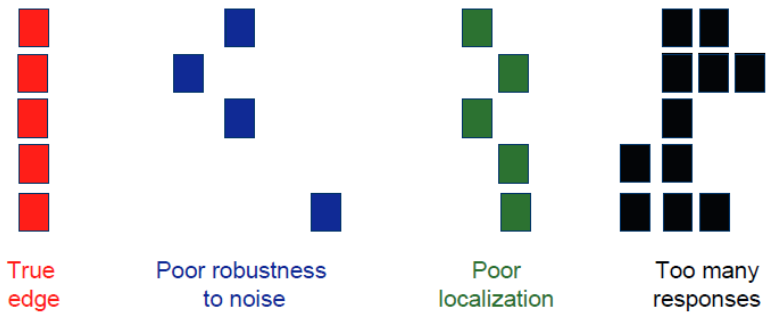

(b)


(c)


(d)

Canny Edge Detector

Quality of an Edge



Canny Edge Detector

- Criterion 1: Good Detection: The optimal detector must minimize the probability of false positives as well as false negatives.
- Criterion 2: Good Localization: The edges detected must be as close as possible to the true edges.
- Single Response Constraint: The detector must return one point only for each edge point.

Canny Edge Detector Steps

1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. Apply “Non-maximum Suppression”
5. Apply “Hysteresis Threshold”

Canny Edge Detector First Two Steps

- Smoothing

$$S = I * g(x,y) = g(x,y) * I$$

$$g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

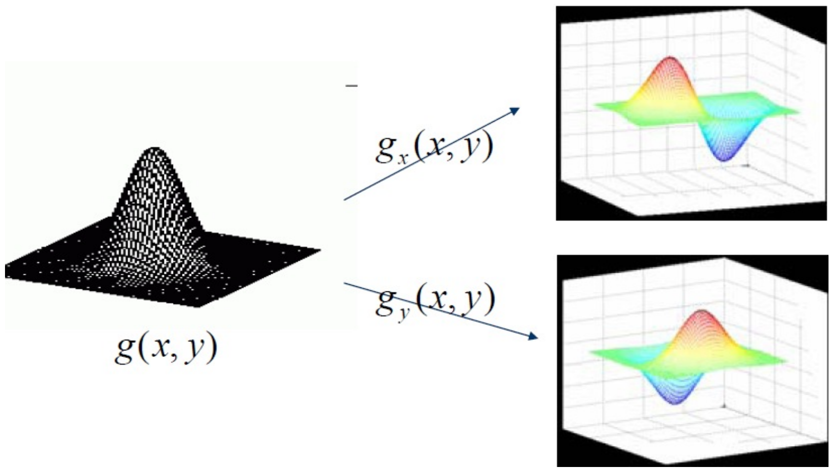
- Derivative

$$\nabla S = \nabla(g * I) = (\nabla g) * I$$

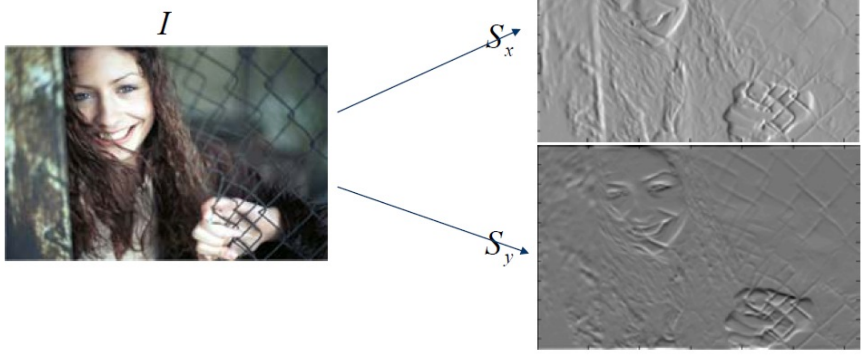
$$\nabla S = \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix}$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

Canny Edge Detector Derivative of Gaussian



Canny Edge Detector: Steps 1 & 2



Canny Edge Detector: Step 3

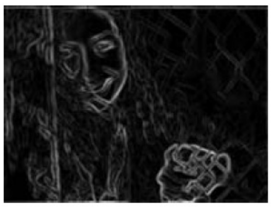
- Gradient magnitude and gradient direction

(S_x, S_y) Gradient Vector

magnitude $= \sqrt{(S_x^2 + S_y^2)}$

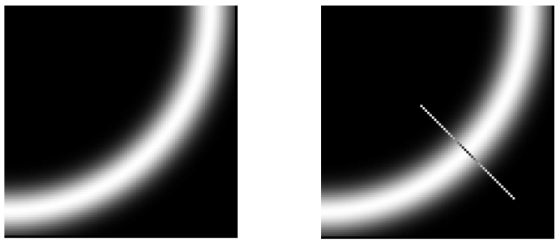
direction $= \theta = \tan^{-1} \frac{S_y}{S_x}$

image


gradient magnitude

Canny Edge Detector: Step 4

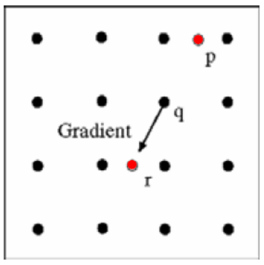
- Non maximum suppression

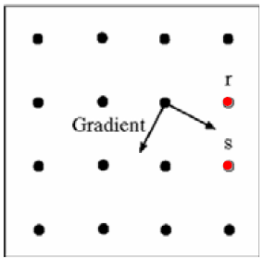


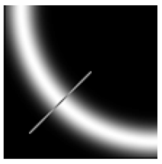
We wish to mark points along the curve where the **magnitude is biggest**. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

Canny Edge Detector Non-maximum Supression

- which point is the maximum?
  - along the gradient direction, q should be larger than p & r (both p & r are interpolated)
- where is the next maximum?
  - next one should be around the line perpendicular to the gradient vector, i.e. r or s

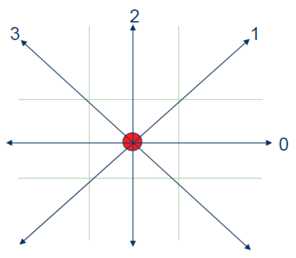


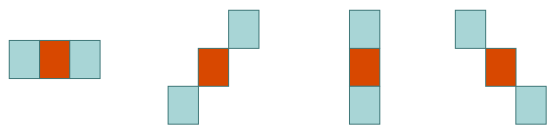




Canny Edge Detector Nonmaxima Supression

- Canny does edge thinning by nonmaxima suppression:
  - Classify gradient angle into one of 4 sectors:
    - 0: -22.5 to 22.5, 180-22.5 to 180+22.5
    - 1: 22.5 to 67.5, 180+22.5 to 180+67.5
    - 2: 67.5 to 112.5, 180+67.5 to 180+112.5
    - 3: 112.5 to 157.5, 180+112.5 to 180+157.5
  - Compare center with the 2 neighbors, set to 0 if not greater than both




Sector 0 Sector 1 Sector 2 Sector 3

$$\tan\theta = \frac{S_y}{S_x}$$

Canny Edge Detector Nonmaxima Supression

$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

M

For visualization

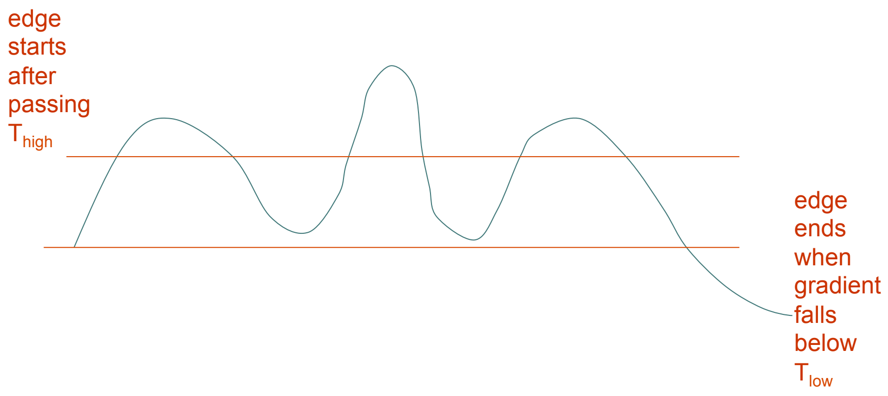
$M \geq Threshold = 25$

Canny Edge Detector Hysteresis Thresholding

- Check that maximum value of gradient is sufficiently large
  - **Hysteresis Thresholding**
    - Use a “High” threshold to start edge curves and a “Low” threshold to continue them.
- If the gradient at a pixel is
  - above “**High**”, declare it an ‘**edge pixel**’
  - below “**Low**”, declare it a “**non-edge-pixel**”
  - **between** “low” and “high”
    - Consider its neighbors iteratively then declare it an “edge pixel” if it is **connected** to an ‘edge pixel’ **directly** or via pixels **between** “low” and “high”.

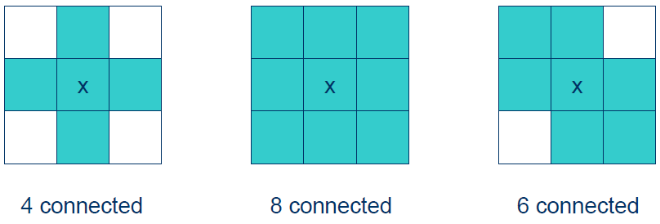
Canny Edge Detector Hysteresis Thresholding

- Double Thresholding

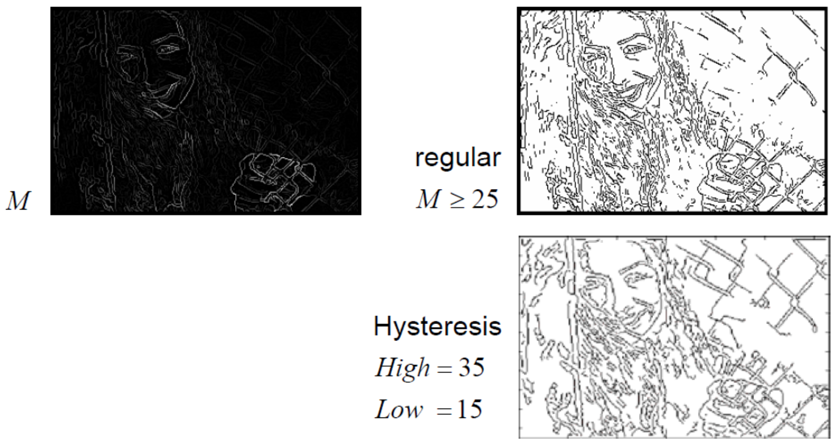

edge starts after passing T_{high}
edge ends when gradient falls below T_{low}

Hysteresis Thresholding

- Scan the image from left to right, top-bottom.
  - The gradient magnitude at a pixel is above a high threshold declare that as an edge point
  - Then recursively consider the *neighbors* of this pixel.
    - If the gradient magnitude is above the low threshold declare that as an edge pixel.



Canny Edge Detector Hysteresis Thresholding

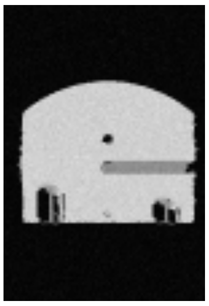


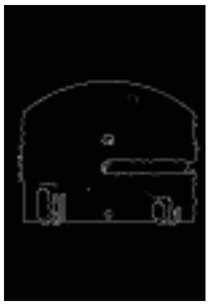
The Canny Edge Detector

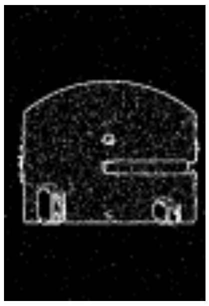
1. Compute smoothed gradient (intensity and direction) for each point in the image.
2. Thin edges, leaving only the pixels at the top of each ridge (*non-maximal suppression*).
3. Threshold ridge pixels using two thresholds, T_{low} and T_{high} (*hysteresis thresholding)*.
   1. Ridge pixels with value greater than T_{high} are considered **strong** edge pixels
   2. Ridge pixels with values between T_{low} and T_{high} are said to be **weak** edge pixels
4. Perform edge linking, aggregating weak pixels that are 8-connected to the strong pixels.

In MATLAB: `J = edge(I, 'canny', T, sigma);`
`% T contains two thresholds`

Canny Edge Detector


Noisy original


Canny


Sobel

Line Identification

57

Edge Identification

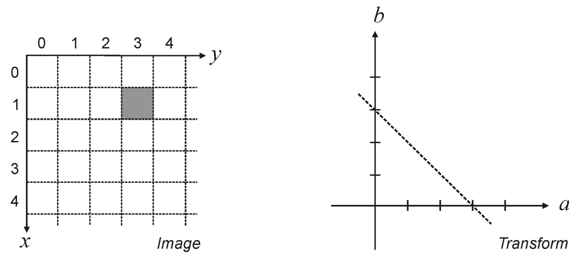
- Due to many technical challenges (noise, shadows, occlusion, etc,), most edge detection algorithms will output an image containing fragmented edges.
- Additional processing is needed to turn fragmented edge segments into useful lines and object boundaries.

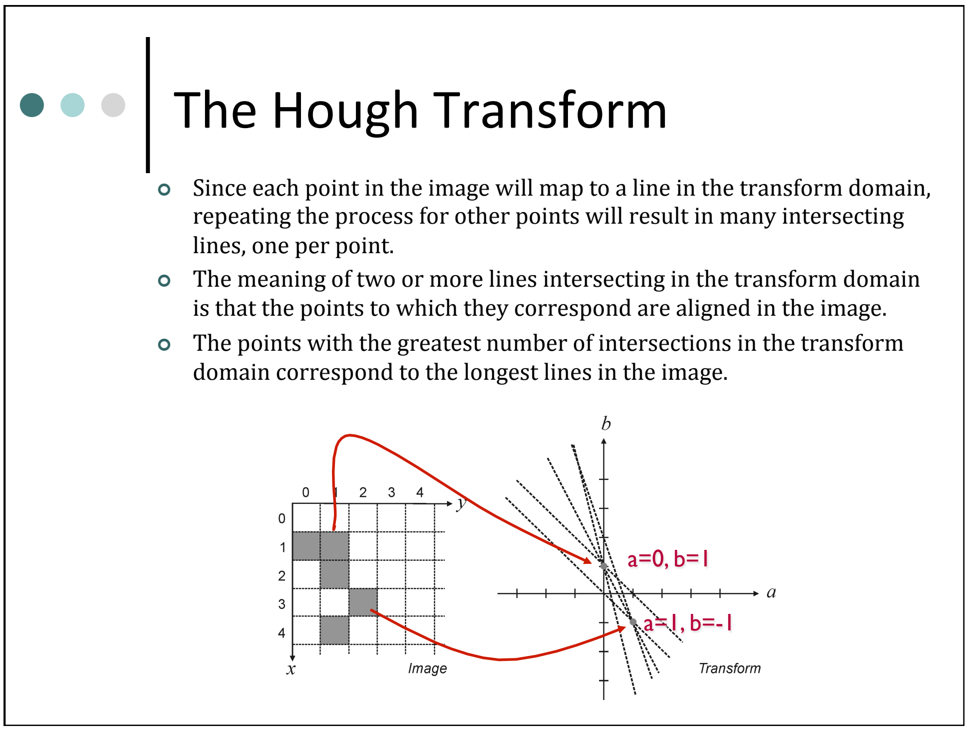
The Hough Transform

- A mathematical method designed to find lines in images.
- It can be used for linking the results of edge detection, turning potentially sparse, broken, or isolated edges into useful lines that correspond to the actual edges in the image.

The Hough transform

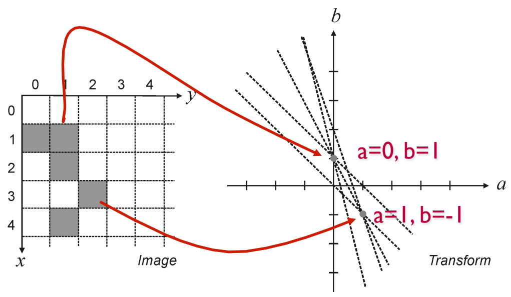
- Let *(x,y)* be the coordinates of a point in a binary image (containing thresholded edge detection results).
- The Hough transform stores in an *accumulator array* all pairs *(a,b)* that satisfy the equation *y = ax+ b*. The *(a,b)* array is called the *transform array*.
  - Example:, the point *(x,y)* = (1,3) in the input image will result in the equation *b = -a + 3*, which can be plotted as a line that represents all pairs *(a,b)* that satisfy this equation.

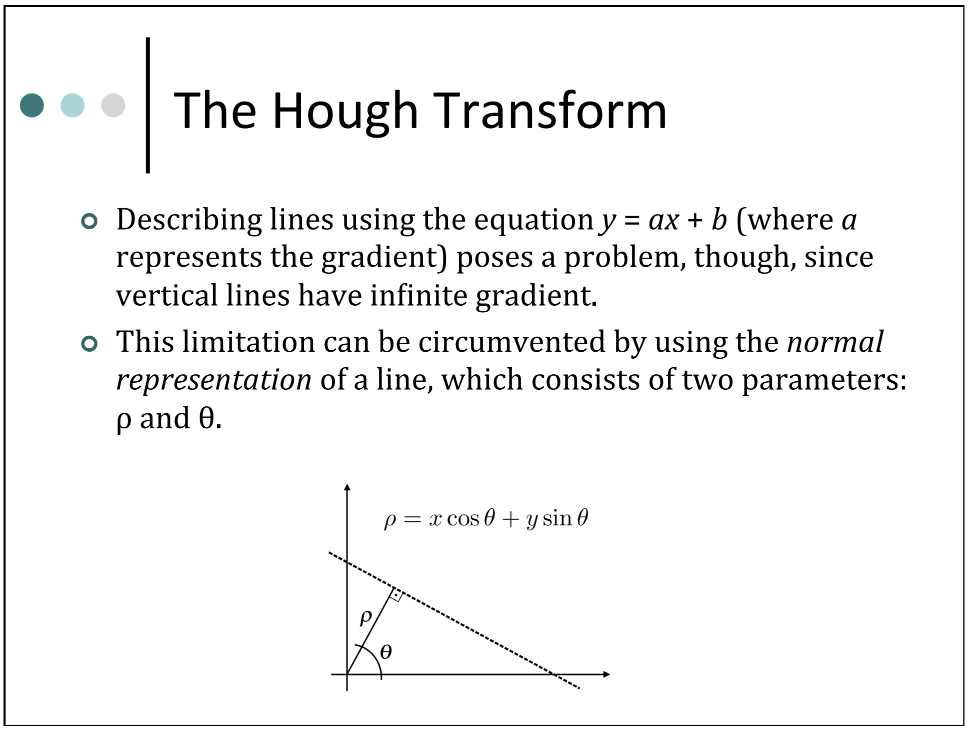




The Hough Transform

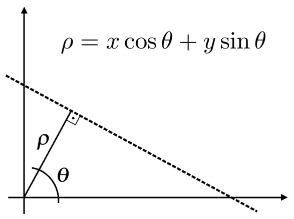
- Since each point in the image will map to a line in the transform domain, repeating the process for other points will result in many intersecting lines, one per point.
- The meaning of two or more lines intersecting in the transform domain is that the points to which they correspond are aligned in the image.
- The points with the greatest number of intersections in the transform domain correspond to the longest lines in the image.



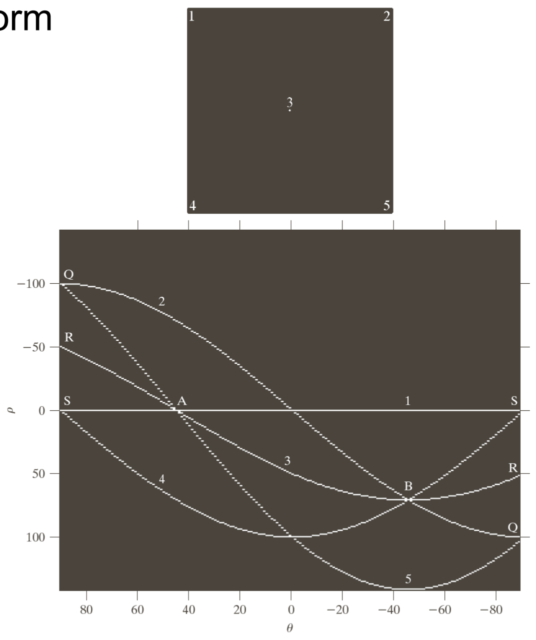


The Hough Transform

- Describing lines using the equation *y* = *ax* + *b* (where *a* represents the gradient) poses a problem, though, since vertical lines have infinite gradient.
- This limitation can be circumvented by using the *normal representation* of a line, which consists of two parameters: ρ and θ.



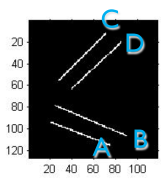
Hough Transform Example

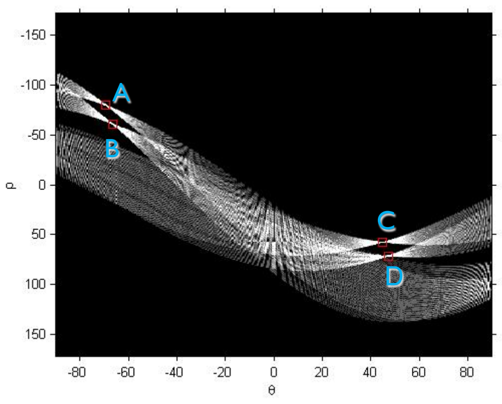


a
b

FIGURE 10.33
(a) Image of size 101 × 101 pixels, containing five points.
(b) Corresponding parameter space. (The points in (a) were enlarged to make them easier to see.)

Example: Hough Transform

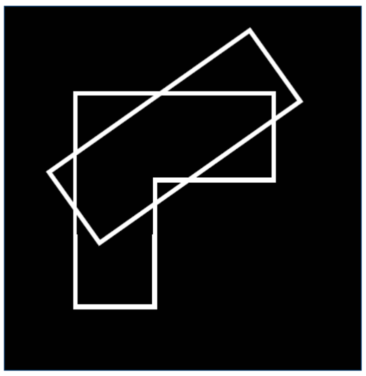


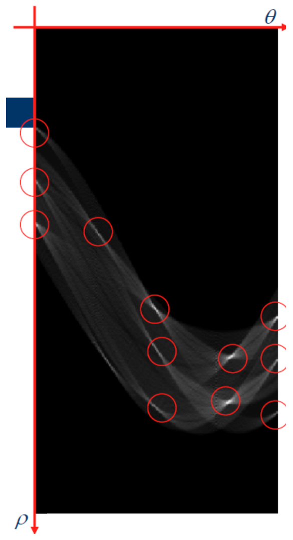


The Hough Transform Algorithm

1. Create a 2D array corresponding to a discrete set of values for ρ and θ. Each element in this array is referred to as an *accumulator cell.*
   1. Increments too big: May not distinguish different lines
   2. Increments oo small: Noise may cause lines to be missed
2. For each pixel (*x,y*) in the image and for each chosen value of θ, compute $x \cos \theta + y \sin \theta$ and write the result in the corresponding position (ρ, θ) in the accumulator array.
3. The highest values in the (ρ, θ) array will correspond to the most relevant lines in the image.

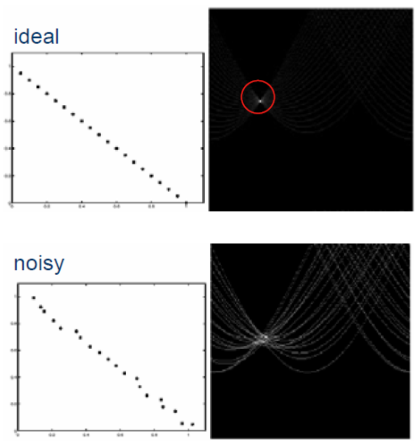
Line FittingHough Transform





Noise vs. Increments

- ρ and θ increments too big: May not distinguish different lines
- ρ and θ increments too small: Noise may cause lines to be missed



Project 1.8

Edge Detection

Due 08.12.2013

68

Problem 1.8

1. Select an image with a dominant edge in it. Display it.
2. Obtain gradient magnitude of its luminance channel. Use Sobel operator for calculating the derivatives. Display the horizontal and vertical gradient images.
3. Apply a threshold to the gradient magnitude image to detect edge pixels. Display the gradient magnitude image and its thresholded version. Pick an appropriate threshold using trial and error.
4. Use Hough transform to identify the parameters of the dominant edge. Display the Hough transformed image.
5. Comment on the performance of the above algorithm on finding the dominant edge in the image.

Next Lecture

IMAGE SEGMENTATION