Edge Detection

Cristina P Santos
cristina@dei.uminho.pt

14/11/2019

Contents

This lecture will cover:

- **Edges**
- **Edge Detection**
  - Sharpening filters using gradients
    - 1^{st} Derivative
    - 2^{nd} Derivative
  - Canny edge detector

Points Detection

Points

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9$$
$$= \sum_{i=1}^{9} w_i z_i$$

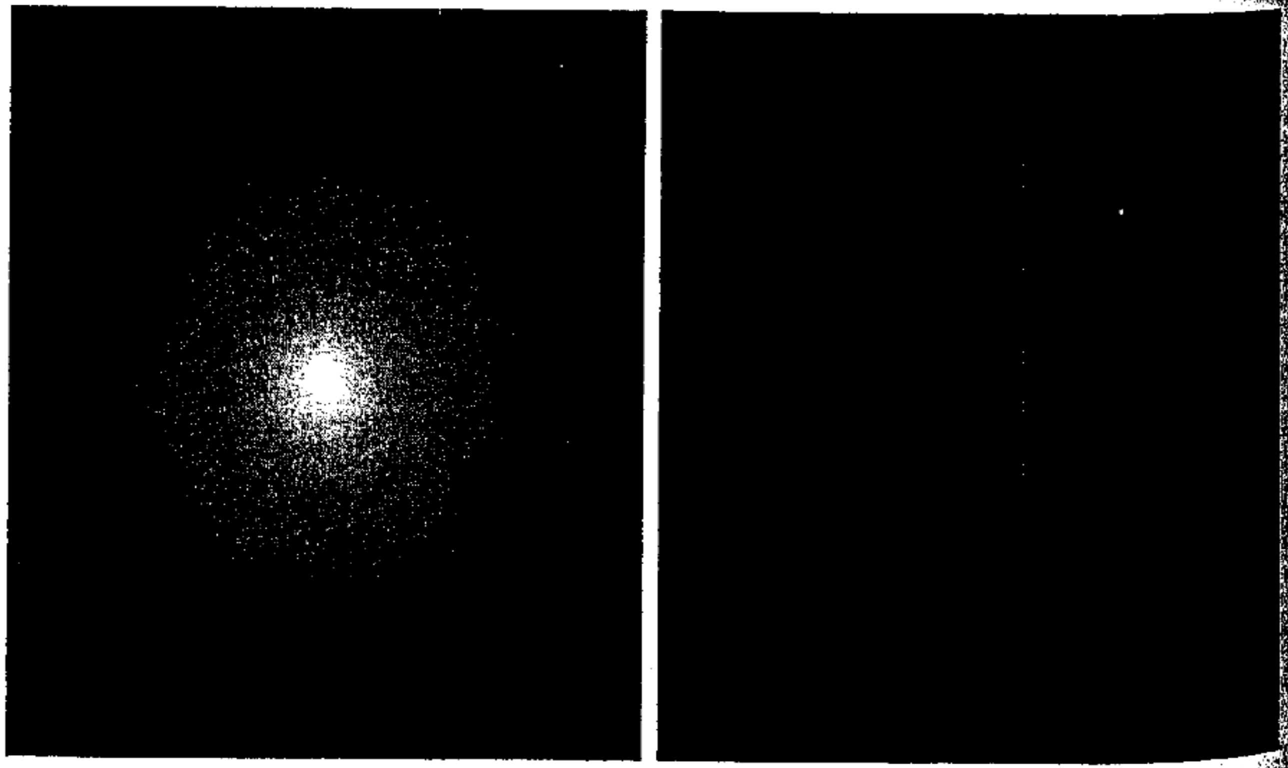
$$|R| \geq T$$

−1	−1	−1
−1	8	−1
−1	−1	−1

EXAMPLE 10.1: Point detection.

■ Figure 10.2(a) shows an image with a nearly invisible black point in the dark gray area of the northeast quadrant. Letting f denote this image, we find the location of the point as follows:

```
>> w = [-1 -1 -1; -1 8 -1; -1 -1 -1];
>> g = abs(imfilter(double(f), w));
>> T = max(g(:));
>> g = g >= T;
>> imshow(g)
```



a b

FIGURE 10.2 (a) Gray-scale image with a nearly invisible isolated black point in the dark gray area of the northeast quadrant. (b) Image showing the detected point. (The point was enlarged to make it easier to see.)

Line Detection

Line Detection

- Mask 1– responds more strongly to line (one pixel thick) oriented horizontally (highest response in the midle)
- Preferred direction weighted by a larger coefficient
- Coefficients of each mask sum to zero, indicating a zero response in areas of constant intensity

−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

−1	−1	2
−1	2	−1
2	−1	−1

+45°

−1	2	−1
−1	2	−1
−1	2	−1

Vertical

2	−1	−1
−1	2	−1
−1	−1	2

−45°

Line Detection

- If we are interested in detecting lines in one particular direction we use the mask associated with the direction and threshold its output.
- If we want to detect all the lines in an image in the direction defined by a mask we run the mas through the all image and threshold the absolute value of the result
- For lines one pixel thick, corresponds to the direction of the mask.

Lets see next

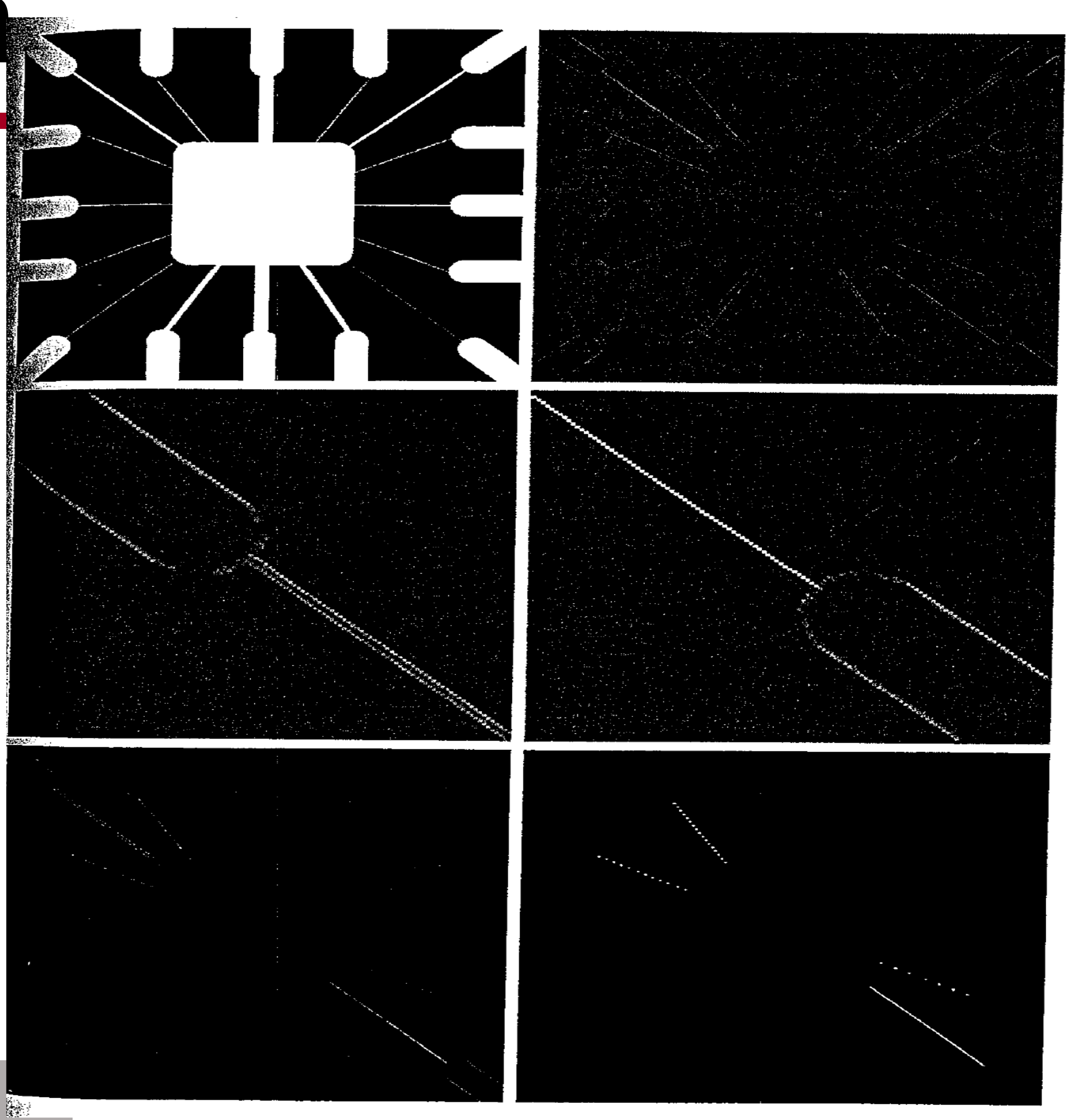
```
1 2];

o)

Fig. 10.4(c)
0:end);
```

Line Detection

- Suppose you are interested in finding all the lines that are one pixel width oriented at -45ª in the next image. Use mask 3 an the following code

```
>> w = [2 -1 -1 ; -1 2 -1; -1 -1 2];
>> g = imfilter(double(f), w);
>> imshow(g, [ ])  % Fig. 10.4(b)
>> gtop = g(1:120, 1:120);
>> gtop = pixeldup(gtop, 4);
>> figure, imshow(gtop, [ ]) % Fig. 10.4(c)
>> gbot = g(end-119:end, end-119:end);
>> gbot = pixeldup(gbot, 4);
>> figure, imshow(gbot, [ ]) % Fig. 10.4(d)
>> g = abs(g);
>> figure, imshow(g, [ ]) % Fig. 10.4(e)
>> T = max(g(:));
>> g = g >= T;
>> figure, imshow(g)  % Fig. 10.4(f)
```

Line Detection



a b
c d
e f

FIGURE 10.4 (a) Image of a wire-bond mask. (b) Result of processing with the $-45°$ detector in Fig. 10.3. (c) Zoomed view of the top, left region of (b). (d) Zoomed view of the bottom, right section of (b). (e) Absolute value of (b). (f) All points (in white) whose values satisfied the condition $g >= T$, where g is the image in (e). (The points in (f) were enlarged slightly to make them easier to see.)

Edges & Edge Detection

Edges

- **Edges** are significant **local changes in the image intensity** and are important features for analyzing images. Edges typically occur on the **boundary between two different regions in an image.**

Edge is usually associated with a **discontinuity** in either the **image intensity or the first derivative of the image intensity.**

It is also possible for an edge to have different types of discontinuity.

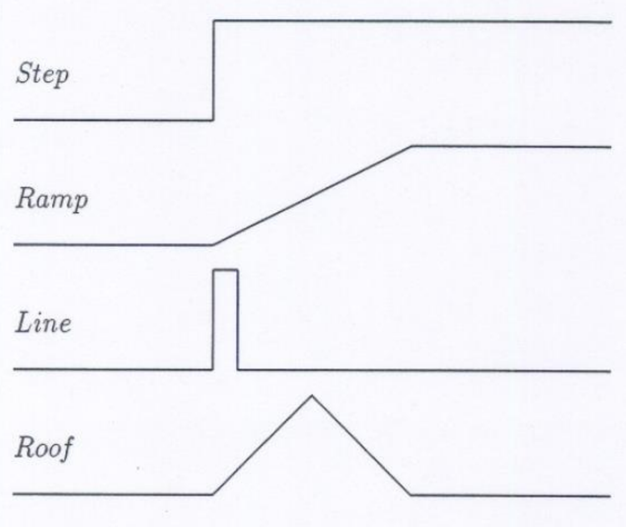


Figure 5.1: One-dimensional edge profiles.

Edge Detection

- **Edge detection** is essentially the operation of **detecting significant local changes in an image**. It is frequently the first step in recovering information from images.

- **Main idea**: look for strong gradients, post-process

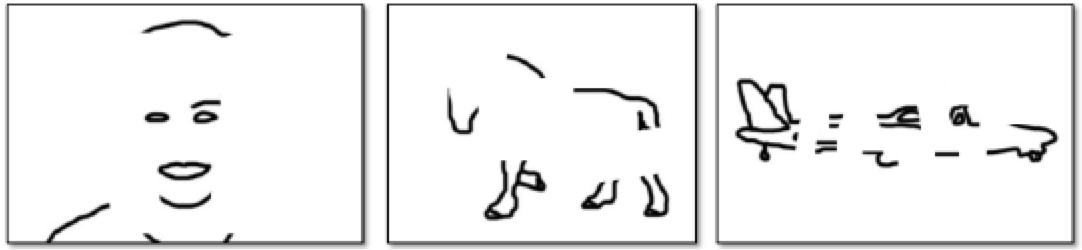
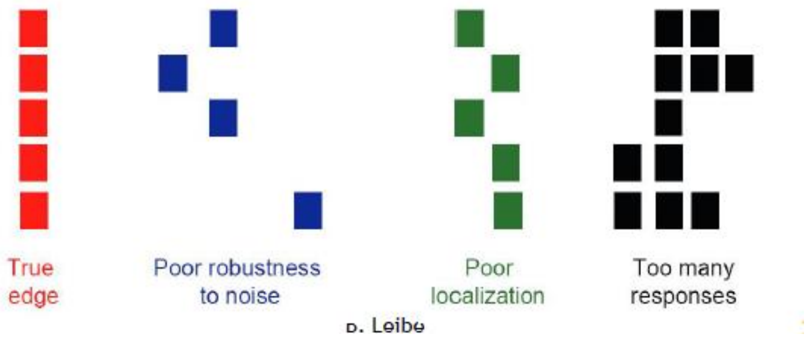


Figure from J. Shotton et al., PAMI 2007

Edge Detection

Criteria for an "optimal" edge detector:

- **Good detection:** the optimal detector should minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges);
- **Good localization:** the edges detected should be as close as possible to the true edges;
- **Single response:** the detector should return one point only for each true edge point; that is minimize the number of local maxima around the true edge.



Edge Detection: Procedure

Primary edge detection steps:

1. **Filtering => Smoothing**: to improve the performance of an edge detector with respect to **noise.**
2. **Edge enhancement:** In order to facilitate the detection of edges, it is essential to **determine changes in intensity** in the neighborhood of a point (gradient)
3. **Detection:** Many points in an image have a **nonzero value for the gradient**, and not all of these points are edges for a particular application. **Some method should be used to determine which points are edge points.**

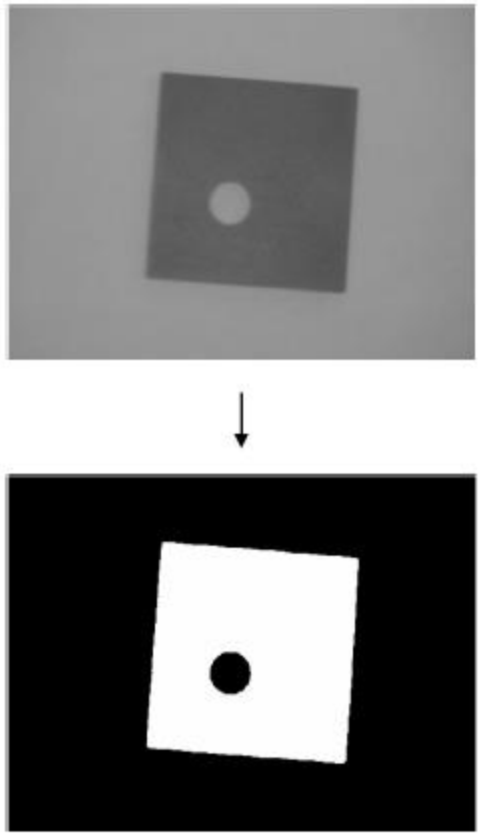
Frequently, **thresholding** provides the criterion used for detection.

4. **Edge localization**

Edge Detection: Recall thresholding

- Choose a threshold *t*
- Set any pixels less than *t* to zero (off)
- Set any pixels greater than or equal *t* to one (on)

$$F_T[i,j] = \begin{cases} 1, & \text{if } F[i,j] \geq t \\ 0, & \text{otherwise} \end{cases}$$



Edge Detection: Procedure

Two issues

- At what scale do we want to extract structures?
- How sensitive should the edge extractor be?

Edge Detection: Example

Original

Gradient magnitude Image

Thresholding with a Lower Threshold

Thresholding with a Higher Threshold

Edge Detection:
Sharpening filters using gradients

Image Gradients

Gradients: can be computed through the derivative function

For the 2D function $f(x,y)$, the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

For discrete data, we can approximate this using finite differences:

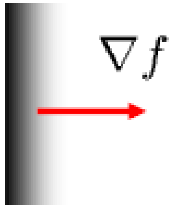
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

Image Gradient

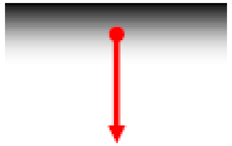
The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

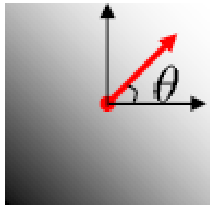
The gradient points in the direction of most rapid intensity change



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient direction (orientation of edge normal) is given by:

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

The edge strength is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

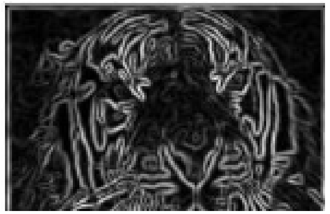
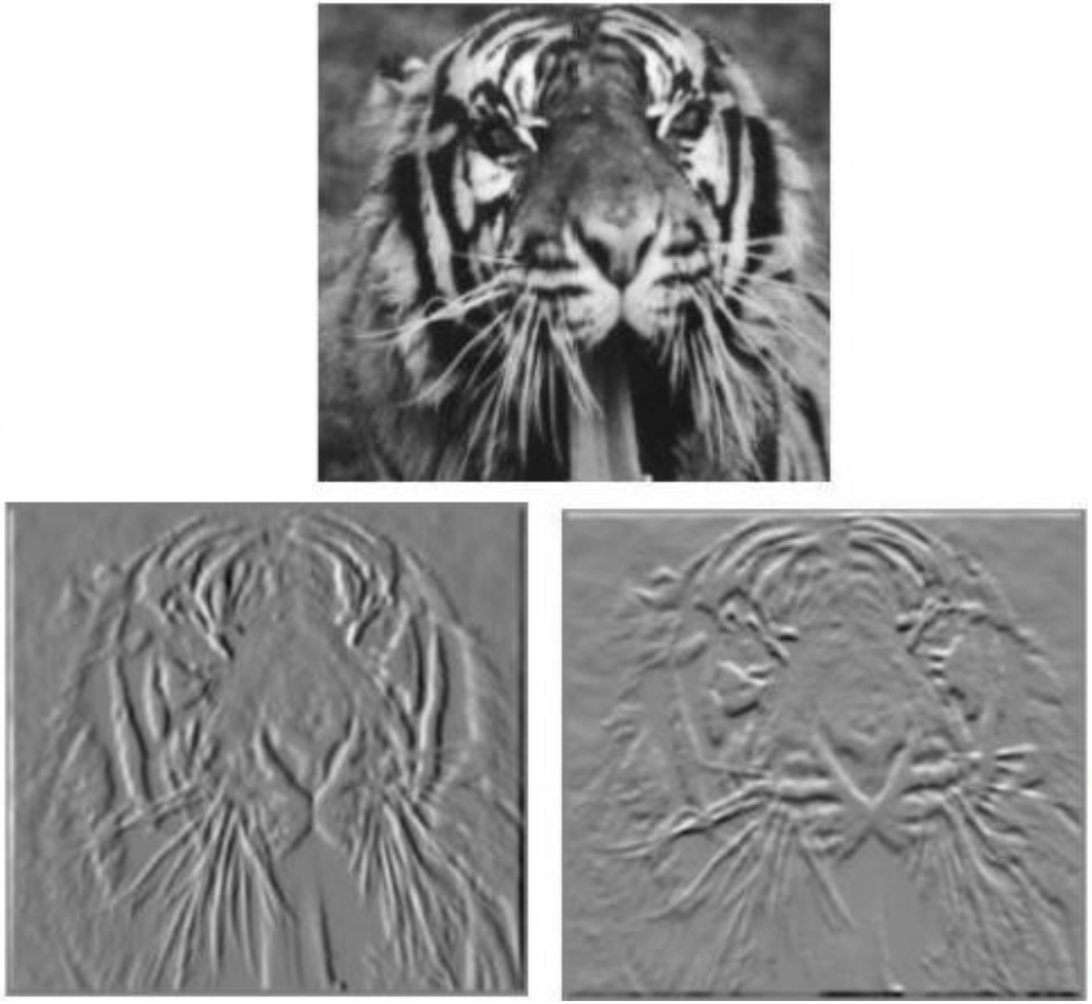


Image Gradients: Partial Derivatives



$$\frac{\partial f(x,y)}{\partial x}$$

$$\frac{\partial f(x,y)}{\partial y}$$

Sharpening Spatial filters

Sharpening goal:

Highlight fine detail in an image or to enhance detail that has been blurred, either as an error or as a natural effect of a particular method of acquisition

Sharpening spatial filters seek to highlight fine detail

- Remove blurring from images
- Highlight edges
- Highlights the **small details** or enhances **details**

Sharpening filters are based on **spatial differentiation =>**

Sharpening ~ differentiation

Sharpening Spatial filters

- 1st Derivative: $$\frac{\partial f}{\partial x} = f(x+1) - f(x).$$

- 2nd Derivative: $$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x).$$

- **1st Derivative:** thicker edges in an image; stronger responses to step

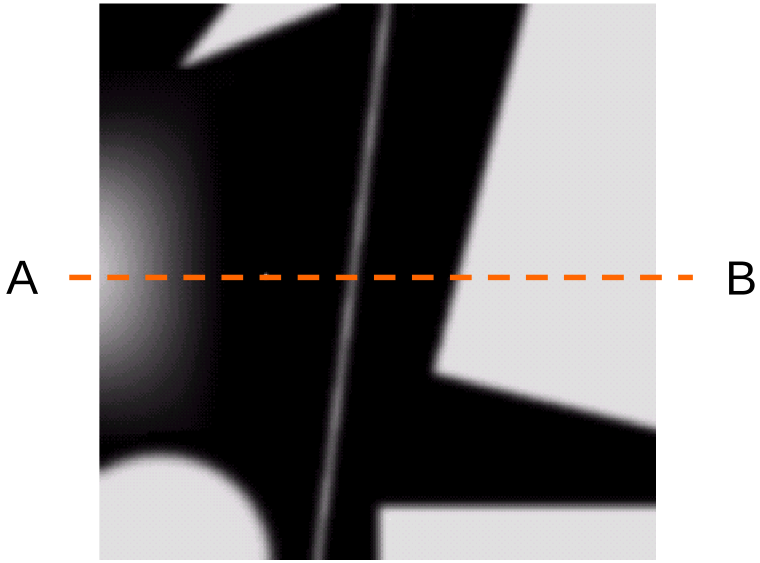
- **2nd Derivative: stronger response to fine detail** (thin lines and isolated points).

Edge Detection

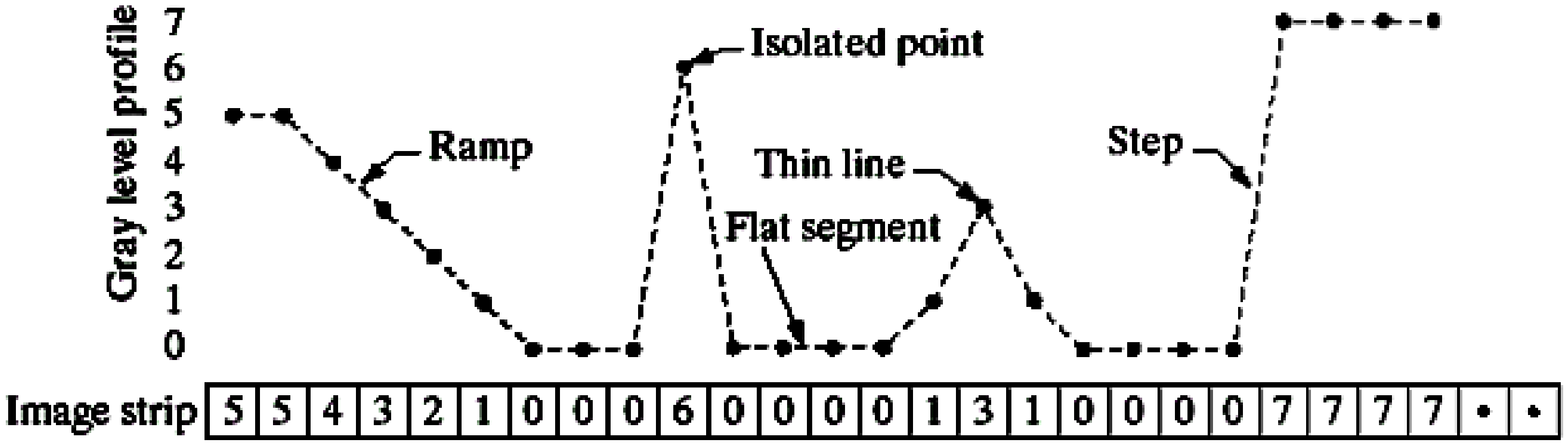
- Basic idea behind edge detection is to find places in an image where the intensity changes rapidly, using one or two general criteria:
- Find place where the first derivative is greater in magnitude than a threshold
- Find place where the second derivative has a zro crossing

```
[g, t] = edge(f, 'method', parameters)
```

Sharpening Spatial filters



Differentiation measures the rate of change of a function



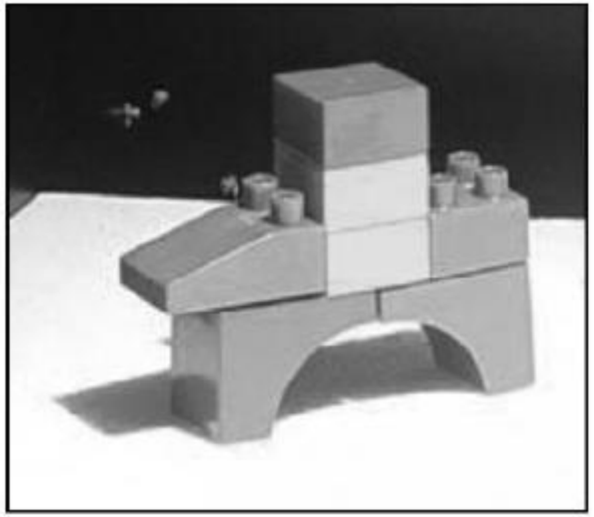
1st Derivative: Roberts' Kernel

Gradient Filtering using Roberts' Kernel

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- Due to approaching the gradient with a reduced number of points is very **susceptible to noise**.



(a)

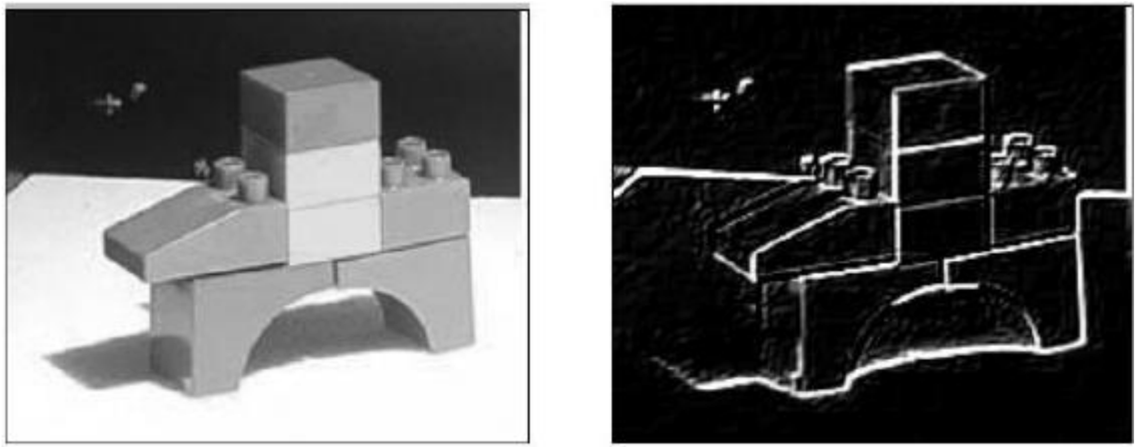
(b)

1^{st} Derivative: Sobel's Kernel

Gradient Filtering using Sobel's Kernel: Applications of 2 kernels

$$Z_h = \begin{bmatrix} -1 & \boxed{-2} & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{aligned} g &= [G_x^2 + G_y^2]^{1/2} \\ &= \{[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 \\ &\quad + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2\}^{1/2} \end{aligned}$$



(a) (b)

$$Z_v = \begin{bmatrix} -1 & 0 & 1 \\ \boxed{-2 \quad 0 \quad 2} \\ -1 & 0 & 1 \end{bmatrix}$$

A pixel at location (x,y) is an edge pixel if g>=T at that location

Note that this operator places an emphasis on pixels that are closer to the center of the *mask.*

1^{st} Derivative: Sobel's Kernel

Gradient Filtering using Sobel's Kernel

$$Z_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$Z_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

[g,t] = edge (f,'sobel',T,dir)

1st Derivative: Prewitt's Kernel

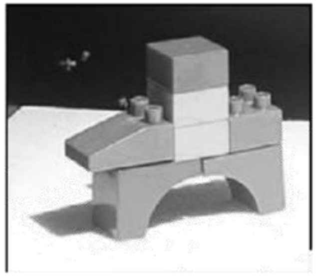
Gradient Filtering using Prewitt's Kernel

$$G_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

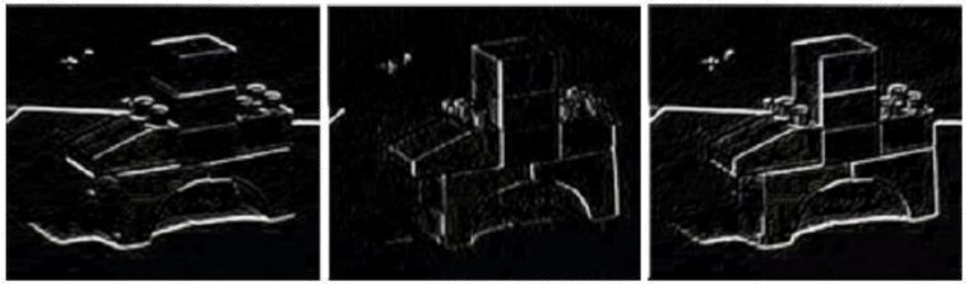
Note that, unlike the Sobel operator, this **operator does not place any emphasis** on pixels that are closer to the **center of the masks**.

More easy to implement computationally

More prone to noise



(a)



(b) (c) (d)

Figura 5.50 - Aplicação do operador gradiente na imagem Blocos original (a), com detecção de contorno no sentido horizontal (b), no sentido vertical (c) e o resultado da soma dos sentidos vertical e horizontal (d) .

[g,t] = edge (f,'Prewitt',T,dir)

1^{st} Derivative: Comparison

Gradient: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Good Localization
Noise Sensitive
Poor Detection

Roberts (2 x 2): $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Prewitt (3 x 3): $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & 1 \end{bmatrix}$

Sobel (3 x 3): $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Poor Localization
Less Noise Sensitive
Good Detection

2^{nd} Derivative

- **The 2^{nd} derivative is more useful for image enhancement** than the 1^{st} derivative
  - Stronger response to fine detail

- The first sharpening filter we will look at is the **Laplacian**
  - **Isotropic** – uniform in all directions
  - One of the simplest sharpening filters
  - We will look at a digital implementation
  - Linear operator

Rotation invariant => Independent of the direction of the discontinuities in the image to which the filter is applied

Rotating the image + filter = **filter + rotating the image**

2^{nd} Derivative: Laplacian Filter

- **Approach:** Construct a discrete formulation of the second order derivative and then constructing a filter mask based on that formulation

- **Laplacian is defined as follows**: $$\nabla^2 f = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y}$$

Where the partial 2^{nd} order derivative in the *x* direction is defined as follows:

$$\frac{\partial^2 f}{\partial^2 x} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

and in the *y* direction as follows:

$$\frac{\partial^2 f}{\partial^2 y} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

2nd Derivative: Laplacian Filter

- **Laplacian is defined as follows**:

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

$$\begin{aligned}\nabla^2 f = & [f(x+1,y) + f(x-1,y) \\ & + f(x,y+1) + f(x,y-1)] \\ & - 4f(x,y)\end{aligned}$$

- **Laplacian using masks**:

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

=

0	1	0
1	-4	1
0	1	0

Isotropic results for rotations in increments of 90º

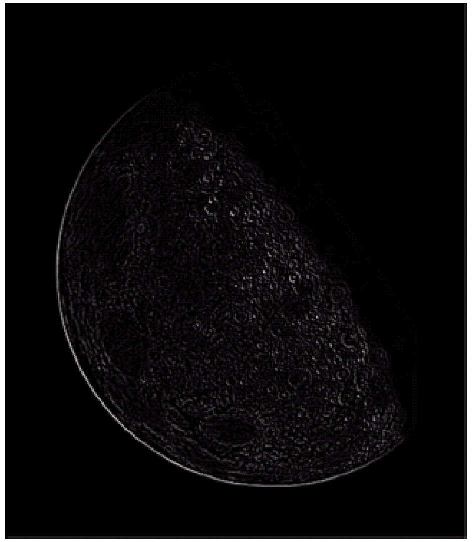
1	1	1
1	−8	1
1	1	1

Isotropic results for rotations in increments of 45º => Variant of Laplacian

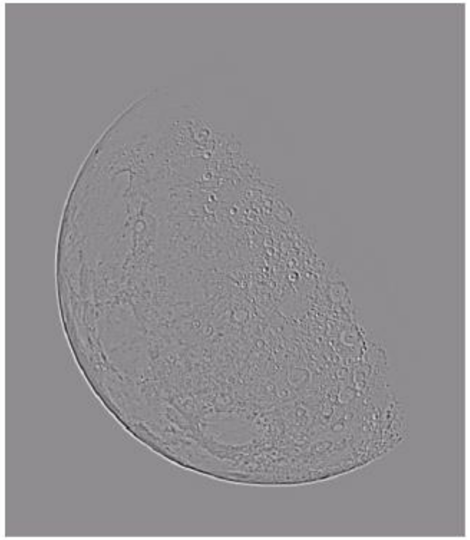
2nd Derivative: Laplacian Filter

- Applying the Laplacian to an image we get a new image that highlights edges and other discontinuities

Original Image



Laplacian Filtered Image

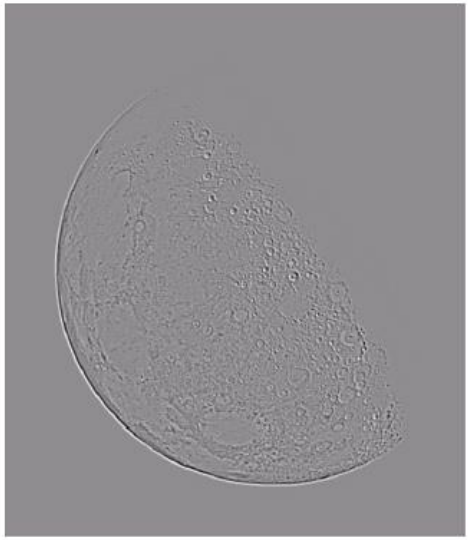


Laplacian Filtered Image Scaled for Display

2nd Derivative: Laplacian Filter

- **The result of a Laplacian filtering is not an enhanced image**
- We have to do more work in order to get our final image => How???
- Subtract the Laplacian result from the original image to generate our final sharpened enhanced image

$$g(x,y) = f(x,y) - \nabla^2 f$$



Laplacian
Filtered Image
Scaled for Display

2nd Derivative: Laplacian Filter

Original Image

-

Laplacian Filtered Image

=

Sharpened Image

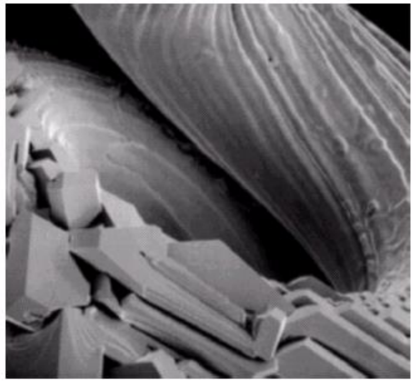
In the final sharpened image edges and fine detail are much more obvious

Simplified Image Enhancement

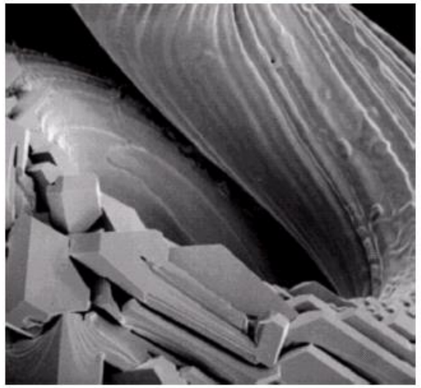
- The entire enhancement can be combined into a single filtering operation

$$g(x,y) = f(x,y) - \nabla^2 f$$
$$= f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)]$$
$$= 5f(x,y) - f(x+1,y) - f(x-1,y) - f(x,y+1) - f(x,y-1)$$

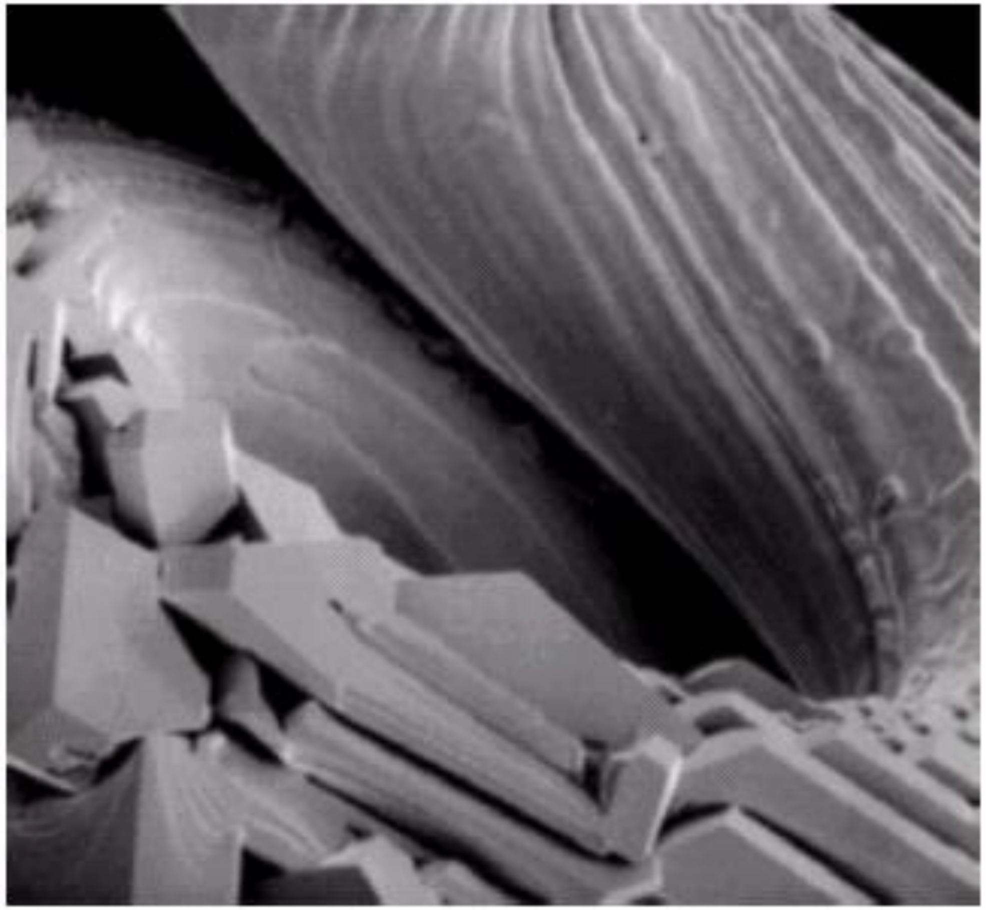
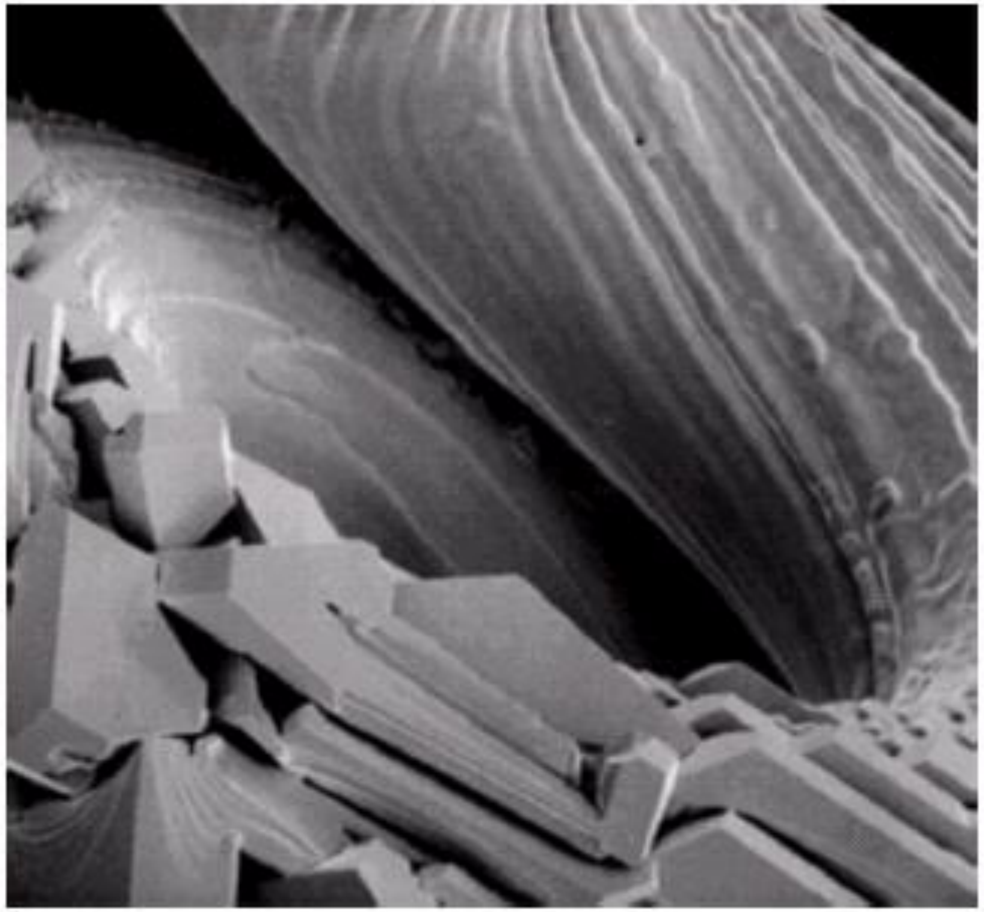
- This gives us a new filter which does the whole job for us in one step



0	-1	0
-1	5	-1
0	-1	0



Simplified Image Enhancement

Edge Detection:
Canny edge detector

Canny edge detector

- Probably the **most widely used edge detector** in computer vision
- **Theoretical model**: step-edges corrupted by **additive Gaussian noise**
- **Canny** has shown that the **1st derivative of the Gaussian** closely approximates the operator that **optimizes the product of signal-to-noise ratio and localization**

Canny edge detector: Procedure

1. **Filter image with derivative of Gaussian**
2. Compute the **Magnitude and Orientation** of the gradient using finite-difference approximations for the partial derivatives
3. Apply **non-maximum suppression** to the gradient magnitude

4. Use the **double thresholding** algorithm to detect and link edges (hysteresis)
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

MATLAB:

```
>> edge(image, 'canny');
>> help edge
```

Canny edge detector: Smoothing

- Let $I[i,j]$ denote the image and $G[i,j,\sigma]$ be a Gaussian smoothing filter σ is the spread of the Gaussian and controls the degree of smoothing.

- The result of convolution of $I[i,j]$ with $G[i,j,\sigma]$ gives an array of smoothed data as:

$$s[i,j] = G[i,j,\sigma] * I[i,j]$$

Canny edge detector: Gradient

Sobel's operator is used to obtain a 2-D spatial gradient

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

Gx

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

Gy

$$\frac{\partial S}{\partial x} = S[i, j] * Gx$$

$$\frac{\partial S}{\partial y} = S[i, j] * Gy$$

$$M[i, j] = \sqrt{\frac{\partial S}{\partial x}^2 + \frac{\partial S}{\partial y}^2}$$

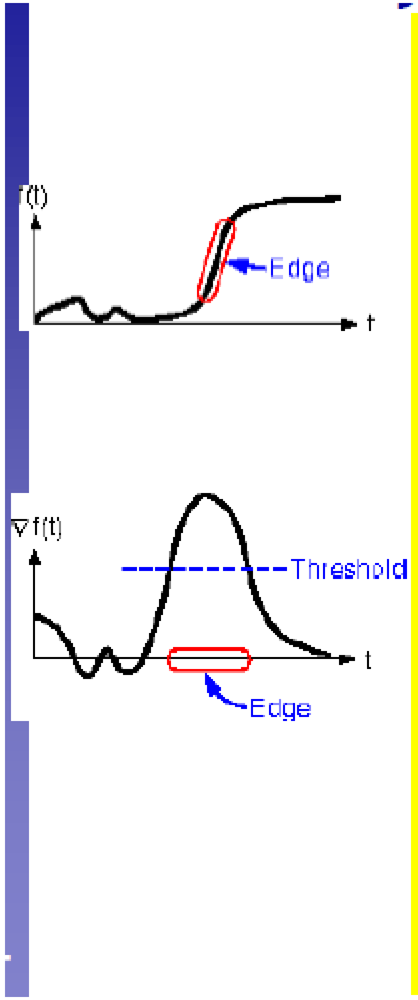
$$\theta[i, j] = \arctan\left(\frac{\partial S / \partial y}{\partial S / \partial x}\right)$$

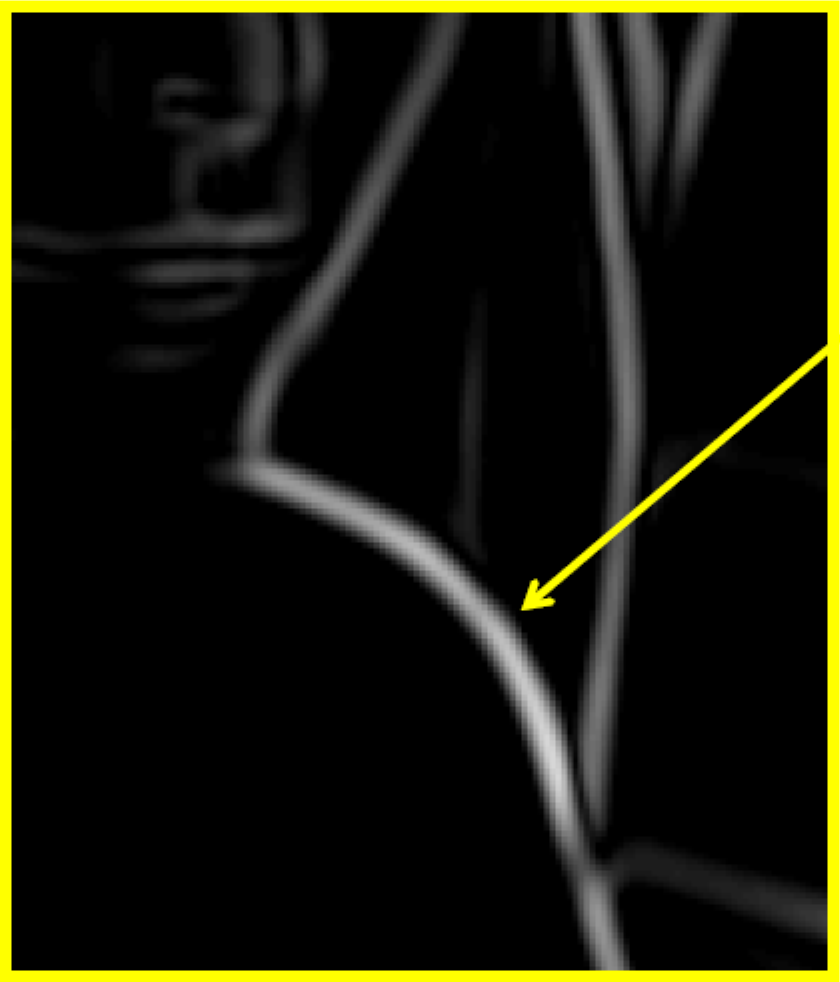
Canny edge detector

Original image (Lena)

Gradient magnitude

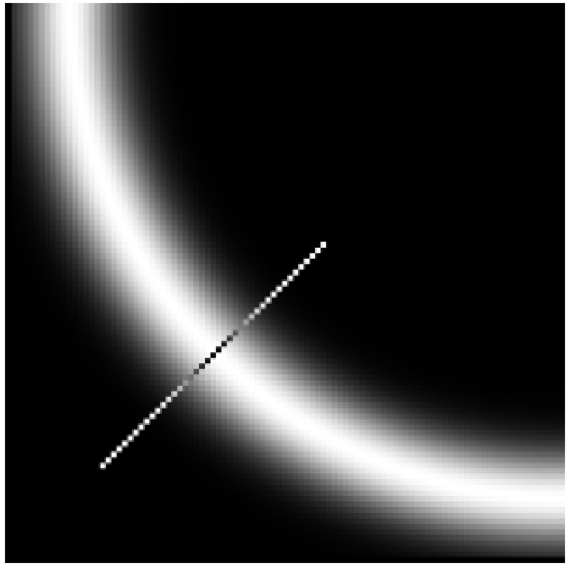
Canny edge detector

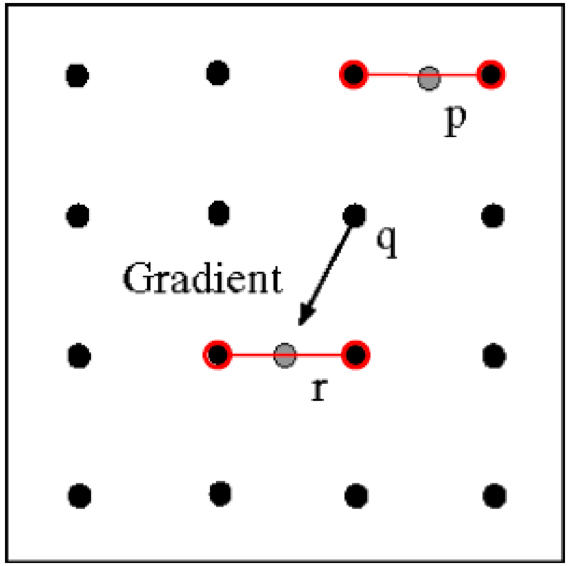




How to turn these thick regions of the gradient into curves?

Canny edge detector: Non-Maximum Suppression





- Non-maxima suppression: to “thin” the edge responses and refine the localization
- It **preserves all local maxima** in the gradient image and **deleting everything** else.

Check if pixel is local maximum along gradient direction, select single max across width of the edge

- Requires checking interpolated pixels p and r

⇒ Linear interpolation based on gradient direction

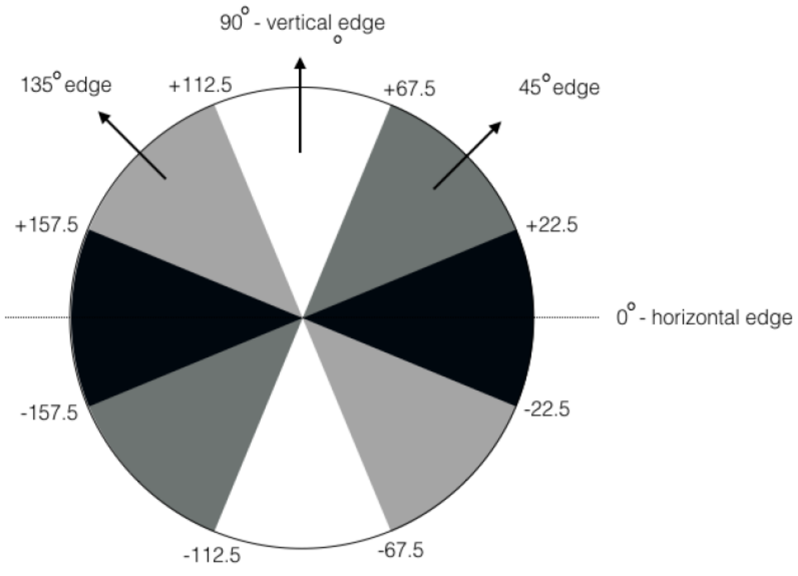
Canny edge detector: Non-Maximum Suppression

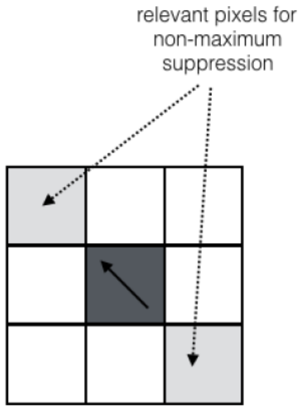
For each pixel in the gradient image, the algorithm is as follows:

1. Know the **edge direction**
2. Specify a **number of discrete orientations for the gradient vector**.

E.g., in a 3 × 3 region, we can define four orientations d_k for an edge passing through the center point of the region.

- 0 degrees (in the horizontal direction),
- 45 degrees (along the positive diagonal),
- 90 degrees (in the vertical direction), or
- 135 degrees (along the negative diagonal).





Canny edge detector: Non-Maximum Suppression

3. Compare the **edge strength of the current pixel** with the **edge strength of the neighbor pixels** in **two opposite sides** along the quantized direction.

4. If the **edge strength** of the current pixel is **greater than that of the neighboring pixels** in the gradient direction, **preserve the value of the edge strength**. If not, suppress the value, i.e., set the value to zero.

$$E_m(i,j) = \begin{cases} 0 & \text{if } E_s(i,j) \text{ is smaller than at least one of its two neighbors on } d_k, \\ E_s(i,j) & \text{else.} \end{cases}$$

Canny edge detector

Gradient magnitude

Non-Maximum Suppression

Thinning
(non-maximum suppression)

Problem: pixels along this edge didn't survive the thresholding.

Canny edge detector

The image output by NONMAX- SUPPRESSION I[i,j] still contains the **local maxima created by noise**.

- If we set a **low threshold** in the attempt of capturing true but **weak edges**, **some noisy maxima will be accepted too** (false contours);
- The **values of true maximum may fluctuate above and below** the threshold, fragmenting the result edge.

A solution is thresholding

Canny edge detector: Hysteresis Thresholding

Solution => Hysteresis Thresholding

Hysteresis: A lag or momentum factor

Idea: Maintain two thresholds k_{high} and k_{low}

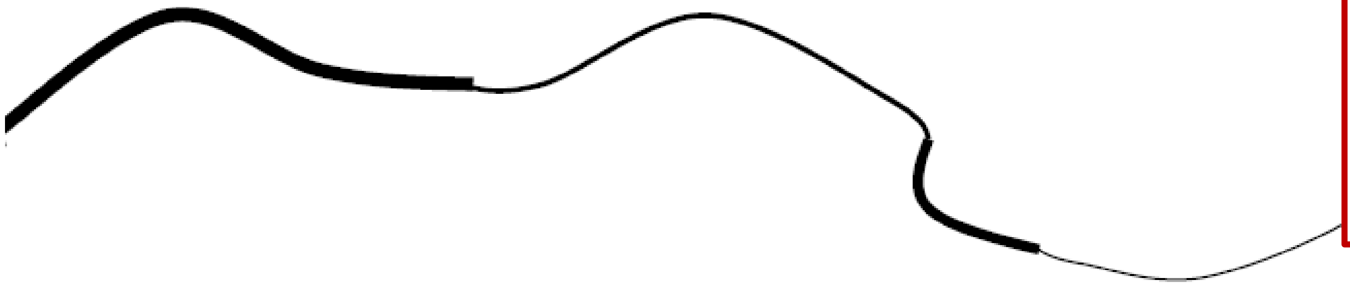
- Use k_{high} to find strong edges to start edge chain
- Use k_{low} to find weak edges which continue edge chain

Typical ratio of thresholds is roughly

$$k_{high} \,/\, k_{low} = 2$$

Apply this threshold for all the edge points in image

Usually a **weak edge pixel** caused from true edges will be **connected to a strong edge** pixel while **noise responses are unconnected.**



Canny edge detector: Hysteresis Thresholding

Solution => Hysteresis Thresholding

The **edge pixels** are divided into connected blobs **using 8-connected neighborhood**. Blobs **containing at least one strong edge pixel are preserved**, while blobs containing only weak edges are suppressed.

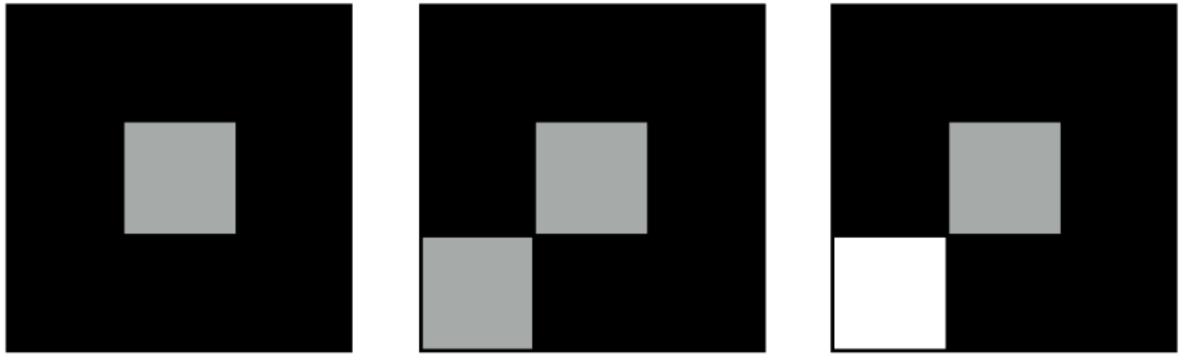


Figure 1.2: Left, middle: A weak edge (gray) with no surrounding strong edges is removed. Right: A weak edge connected to at least one strong edge (white) is kept.

The output is a set of list, each describing the position of a connected contour in the image.

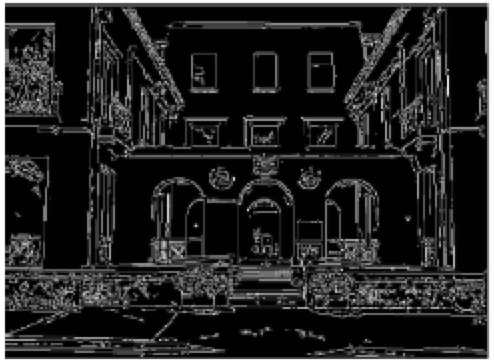
Canny edge detector

Solution => Hysteresis

Original image

High threshold
(strong edges)

Low threshold
(weak edges)



courtesy of G. Loy

Hysteresis threshold

Canny edge detector: Observation

original $\sigma = 1$ Canny with $\sigma = 2$

- The choice of σ depends on desired behavior
  - large σ detects large scale edges
  - small σ detects fine features

Summary

This lecture covered:

- **Edges**
- **Edge Detection**
  - Sharpening filters using gradients
    - 1^{st} Derivative
    - 2^{nd} Derivative
  - Canny edge detector