Lesson 6: Contours

1. Introduction

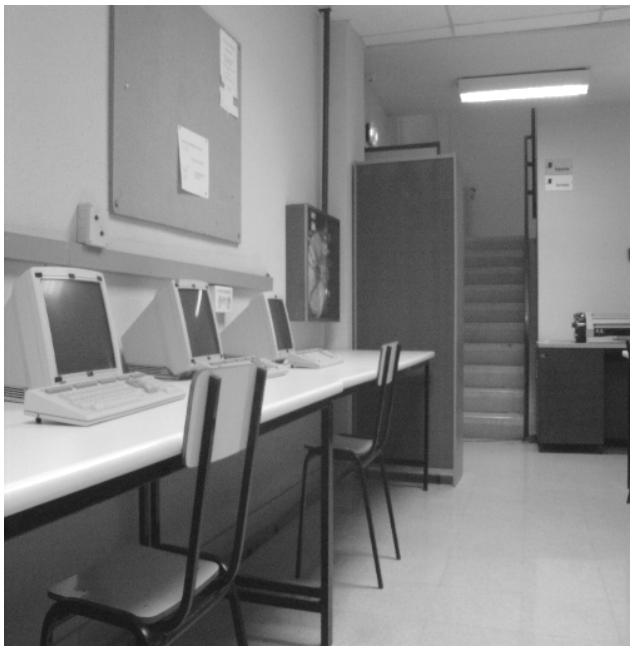
2. Image filtering: Convolution

3. Edge Detection
   - Gradient detectors: Sobel, Canny, ...
   - Zero crossings: Marr-Hildreth

4. Contour segmentation
   - Local tracking
   - Hough transform

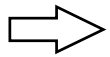
Contour Segmentation

- Obtain meaningful elements of the objects' surface
- Contours summarize most of the image information



Contour Segmentation

- Edge detection
  - Edges: points with significant brightness change
    - » Maxima of the image Gradient
      (1st derivative of brightness)
    - » Zero crossings of the image Laplacian
      (2nd derivative of brightness)



Convolution Masks

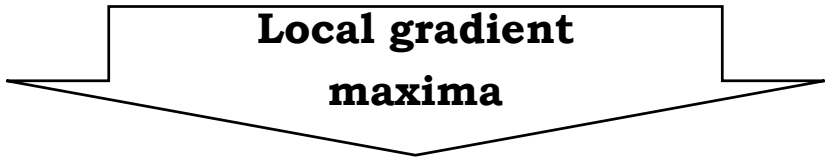
- Contour Segmentation
  - Edge grouping
  - Noise removal

- Line or curve fitting

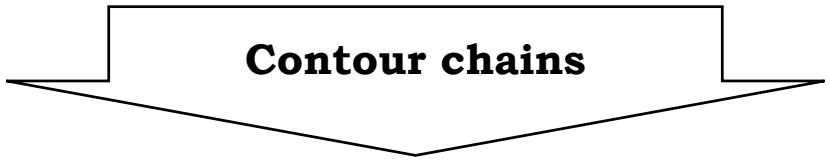
Example: Canny's method

- Canny operator: derivative of a Gaussian

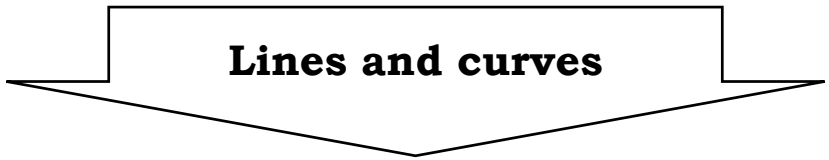
- Maxima detection



- Contour segmentation
  - Local tracking + hysteresis thresholding



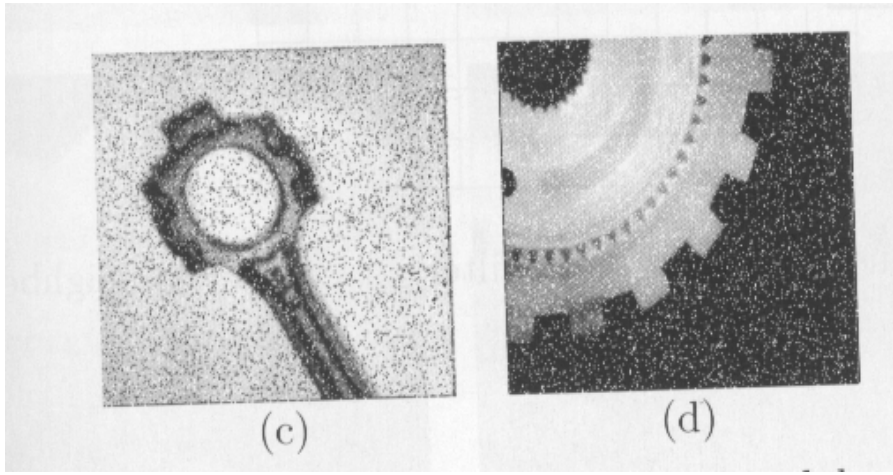
- Line or curve fitting



2. Image filtering. Convolution

- **Typical image noise:**
  - **Salt and pepper noise:** some random pixels that are completely white or black
  - **Gaussian noise:** the grey level of all pixels is perturbed by a random gaussian value

$$I[i,j] = I_{\text{ideal}}[i,j] + p \qquad p \approx N(0,\sigma^2)$$



Linear filters: Convolution

- Every pixel in the output image is a linear combination of several neighbouring pixels from the input image
- Convolution Mask (or Kernel):

$$K[m,n] = \begin{bmatrix} k_{-a,-c} & \cdots & k_{-a,0} & \cdots & k_{-a,d} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ k_{0,-c} & \cdots & k_{0,0} & \cdots & k_{0,d} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ k_{b,-c} & \cdots & k_{b,0} & \cdots & k_{b,d} \end{bmatrix}$$

$$g[i,j] = K[m,n] * f[i,j] = \sum_{m=-a}^{b} \sum_{n-c}^{d} K[m,n] f[i+m, j+n]$$

  - Masks are usually centred
  - If not, mark the central pixel

Convolution: Examples

- Uncentred mask
  - Forward difference

$$K = \boxed{-1 \;\; 1}$$

$$g(i,j) = f(i,j+1) - f(i,j)$$

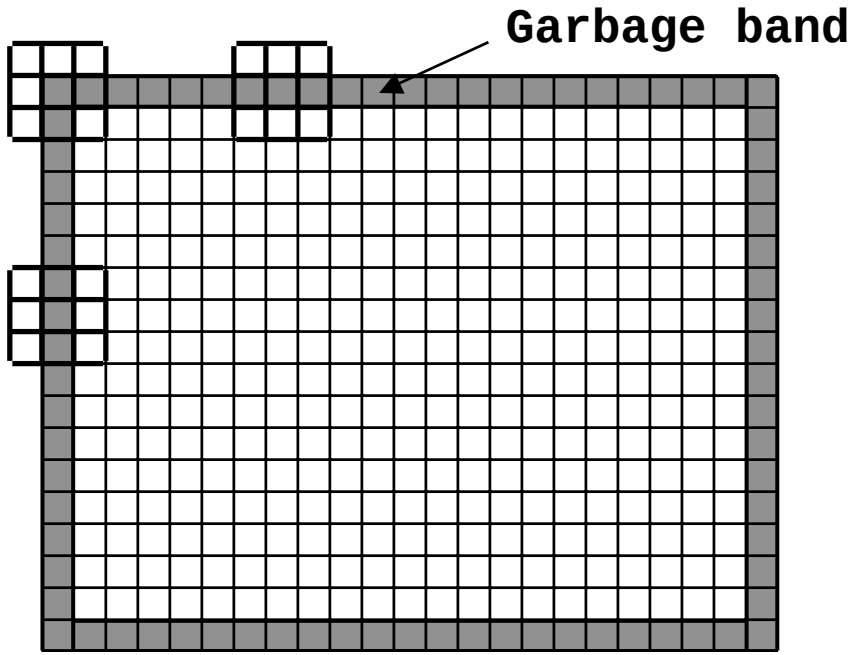
- Centred mask
  - Binomial filter 3x3

$$B_{3x3} = \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

$$\begin{aligned} g(i,j) = \frac{1}{16} \; [& f(i-1,j-1) + 2f(i-1,j) + f(i-1,j+1) + \\ & 2f(i,j-1) + 4f(i,j) + 2f(1,j+1) + \\ & f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1)] \end{aligned}$$

Convolution: Garbage Band

- Near the image borders, the mask does not fit inside the actual image



- For a n x n mask (with n odd) the size of the garbage bands is: (n-1)/2
  - Either let the garbage enter the garbage band
  - Or better, fill the band with zeros

Image Smoothing (noise removal)

- Box filter
  - Arithmetic mean:

$$K_{3x3} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Binomial Filter
  - Values from the Pascal triangle:

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array}$$

  - Weight decrease far from the centre:

$$B_{3x3} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Dividing by the sum of the mask weights, the mean image brightness remains unchanged

Gaussian Smoothing

- Gaussian Filter

$$G_\sigma(i,j) = c \cdot e^{-\frac{(i^2+j^2)}{2\sigma^2}}$$

  - c is computed such that the weights sum up to 1
  - For bigger σ, bigger smoothing effect
  - Mask size: 5σ - 7σ
- Example: Gaussian mask 5 x 5 for $\sigma = 1$

0.0183	0.0821	0.1353	0.0821	0.0183
0.0821	0.3679	0.6065	0.3679	0.0821
0.1353	0.6065	1.0000	0.6065	0.1353
0.0821	0.3679	0.6065	0.3679	0.0821
0.0183	0.0821	0.1353	0.0821	0.0183

c = 1/6.1689

Gaussian Smoothing

- Smoothing with σ = 1, 2, 3 and 4




Separable Masks

- Separability condition: rank(K) = 1

$$K(i,j) = K_y(i) \cdot K_x(j)$$

$$g(i,j) = K(i,j) * f(i,j) = K_y(i) * [K_x(j) * f(i,j)] = K_x(j) * [K_y(i) * f(i,j)]$$

Mask nxn ⇒ **Mask nx1 + Mask 1xn** — **more efficient**

– Binomial Filter:

$$B_{3x3} = \frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1\\2\\1\end{bmatrix} \cdot \begin{bmatrix}1&2&1\end{bmatrix}$$

– Gaussian Filter:

$$G_{\sigma=1} = \frac{1}{6.1689}\begin{bmatrix}0.1353\\0.6065\\1.0000\\0.6065\\0.1353\end{bmatrix} \cdot \begin{bmatrix}0.1353&0.6065&1.0000&0.6065&0.1353\end{bmatrix}$$

Separable Masks: Implementation

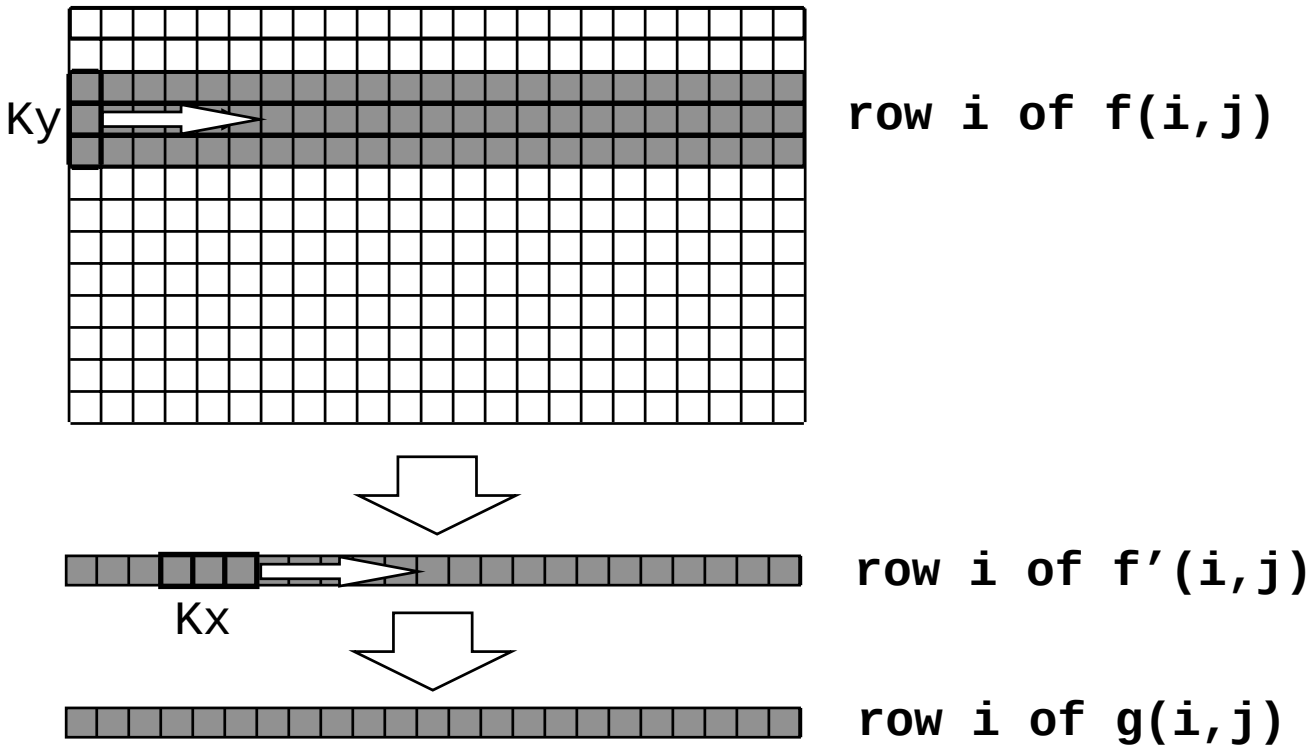
- Efficient programming:

$$g(i,j) = K(i,j) * f(i,j) = K_x(j) * [K_y(i) * f(i,j)]$$

$$f'(i,j) = K_y(i) * f(i,j)$$

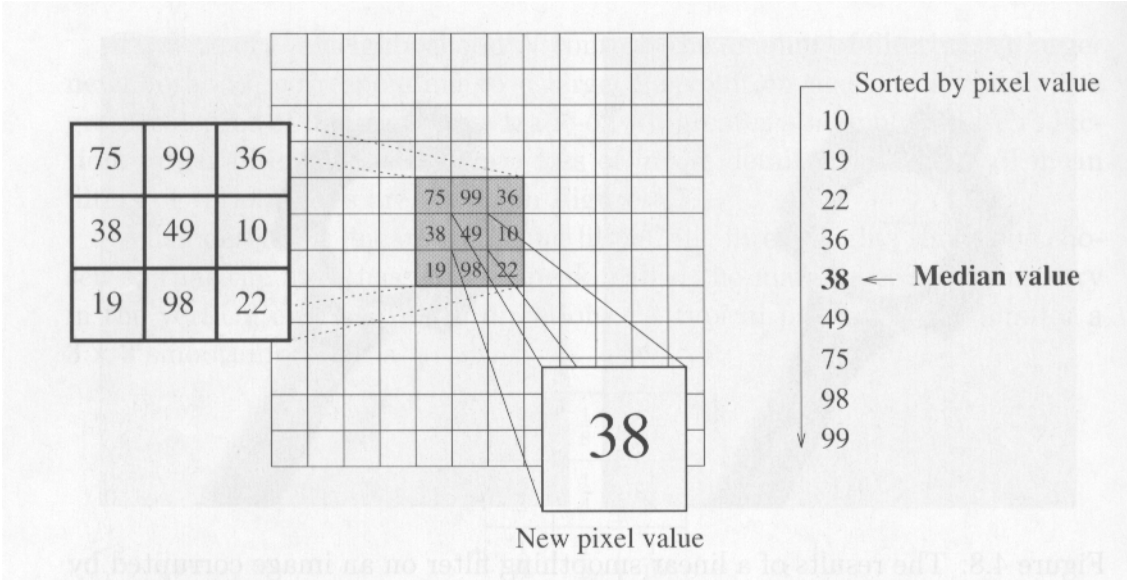
$$g(i,j) = K_x(j) * f'(i,j)$$

- Only an intermediate image row is needed



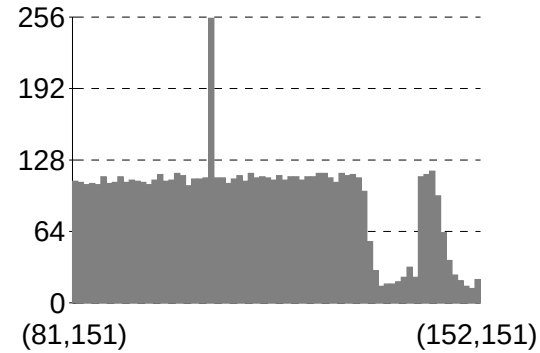
Median Filter

- It is a non-linear filter
  - Sort the nxn neighbours of pixel (i,j) by grey level
  - Assign to the pixel (i,j) in the output image the central value (the median)
  - Good for salt and pepper noise and for removing defects on analogic slides or negatives

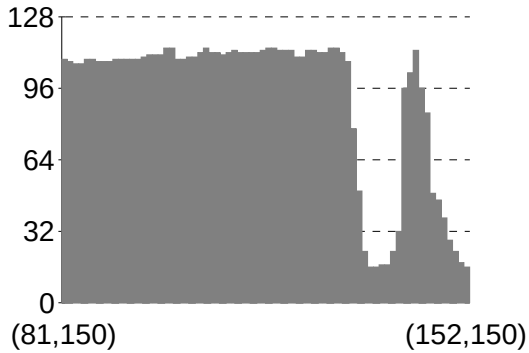


» The outliers do not influence the output value

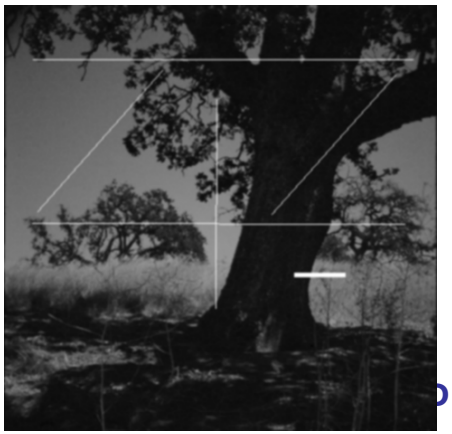
Median Filter

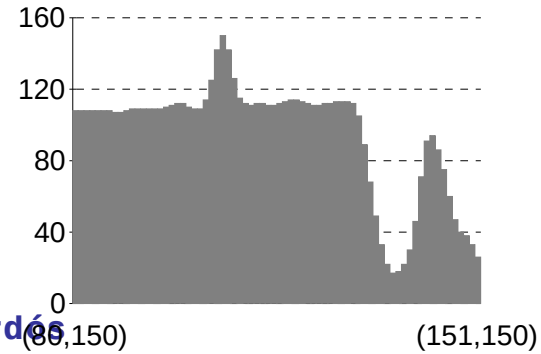


Original image



Median filter

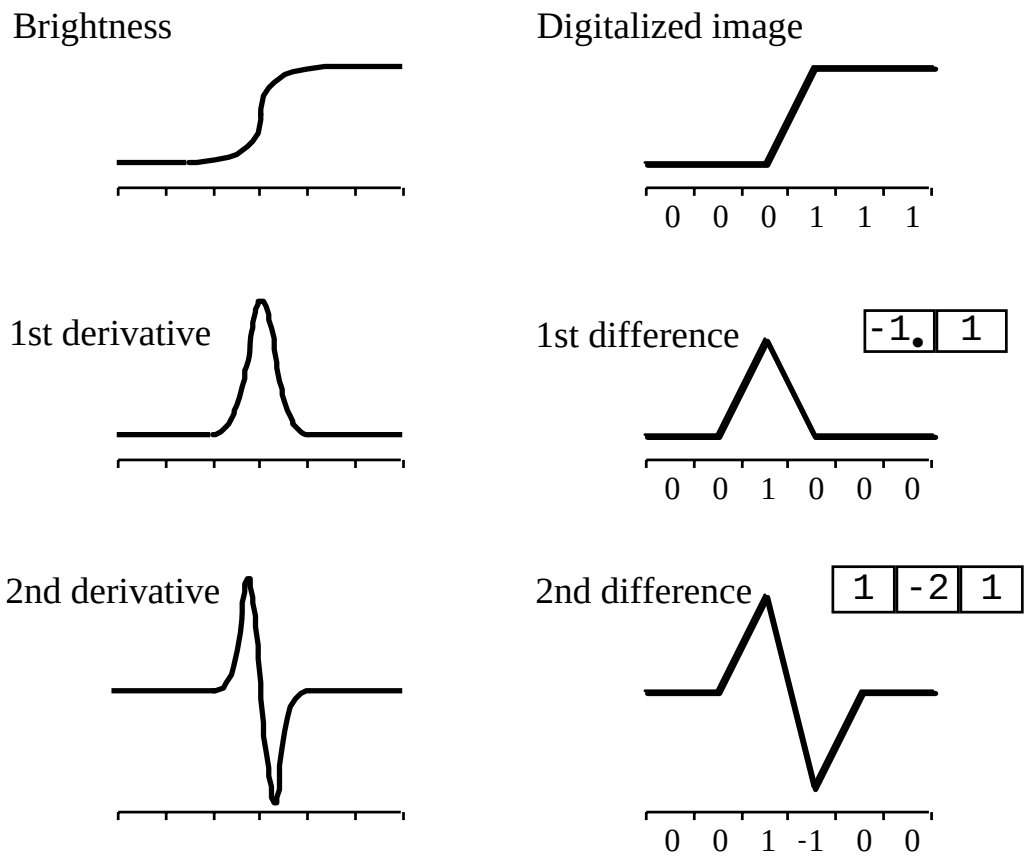




Linear filter

3. Edge Detection

- Maxima of the Gradient (1st derivative)
- Zero crossings of the Laplacian (2nd derivative)



Gradient-Based Edge Detectors

- Gradient:

Gradient $\vec{\nabla} f(x,y) = (\nabla_x, \nabla_y) = \left(\frac{\partial f}{\partial x} , \frac{\partial f}{\partial y} \right)$

Magnitude $|\nabla f| = \sqrt{(\nabla_x)^2 + (\nabla_y)^2} \approx |\nabla_x| + |\nabla_y|$

Orientation $\theta = \text{atan } 2(\nabla_x, \nabla_y)$

- Operators:

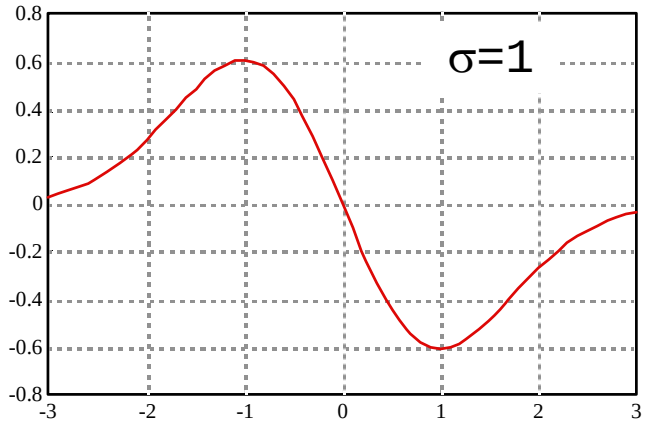
	∇_x	∇_y
1st *difference*	$\begin{bmatrix} -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$
Roberts	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Prewitt	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Divide by the sum of the positive mask coefficients

Canny Edge Detector

- Was designed to optimize:
  - Detection robustness against noise
  - Precision in edge location
  - Single response
- Can be approximated by the derivative of a Gaussian:

$$G'_\sigma(x) = \frac{-x}{\sigma^2}\exp\left(\frac{-x^2}{2\sigma^2}\right)$$



- Mask size: 5σ–7σ
- Example, for σ = 1

0.2707	0.6065	0.0	-0.6065	-0.2707

Divide by the sum of the positive mask coefficients

Canny Edge Detector

- Gradient in x and y axis:
  - It is a separable operator

$$\nabla_x = G'_\sigma(x) * G_\sigma(y) * f(i,j)$$
$$\nabla_y = G_\sigma(x) * G'_\sigma(y) * f(i,j)$$

  - For $\sigma = 1$ and n = 5

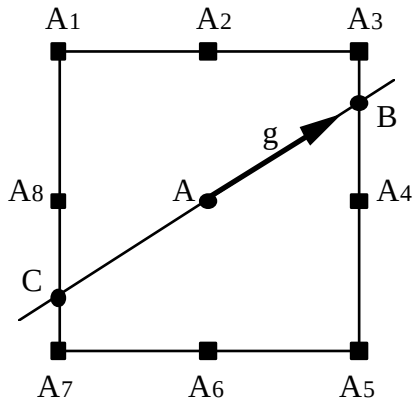
$$\nabla_x = \frac{1}{K} \begin{bmatrix} -0.2707 & -0.6065 & 0.0 & 0.6065 & 0.2707 \end{bmatrix} * \begin{bmatrix} 0.1353 \\ 0.6065 \\ 1.0 \\ 0.6065 \\ 0.1353 \end{bmatrix} * f(i,j)$$

$$\nabla_y = \frac{1}{K} \begin{bmatrix} 0.1353 & 0.6065 & 1.0 & 0.6065 & 0.1353 \end{bmatrix} * \begin{bmatrix} 0.2707 \\ 0.6065 \\ 0.0 \\ -0.6065 \\ -0.2707 \end{bmatrix} * f(i,j)$$

Non-Maximum Suppression

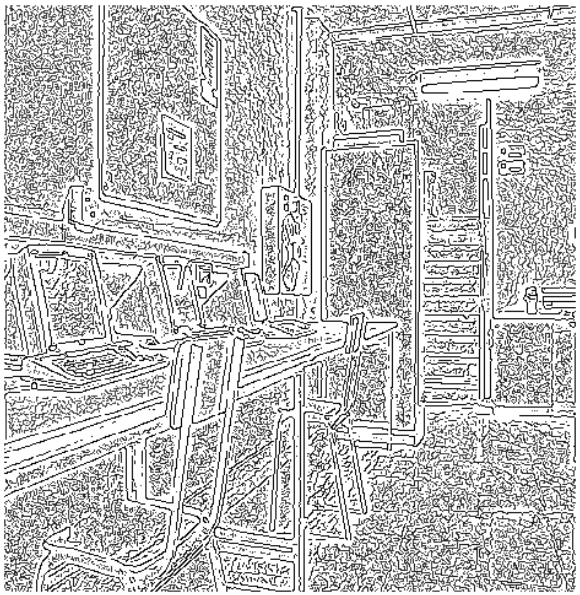
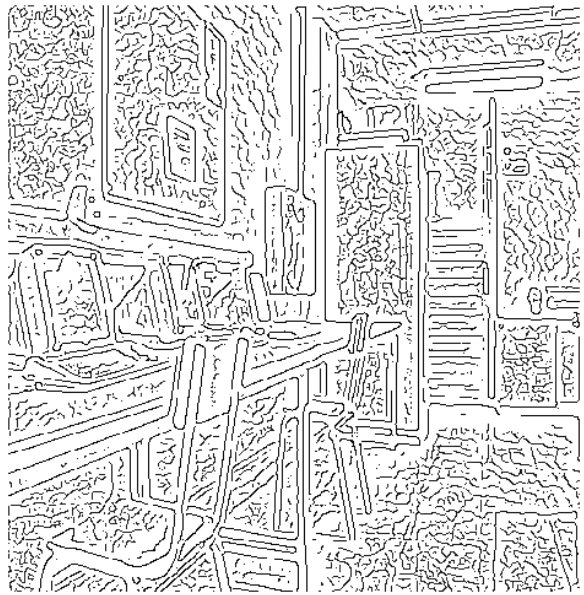
Find point where the gradient magnitude is maximal in the direction of the gradient vector

- For every pixel with magnitude bigger that a threshold, verify the maximum using a 3x3 window:

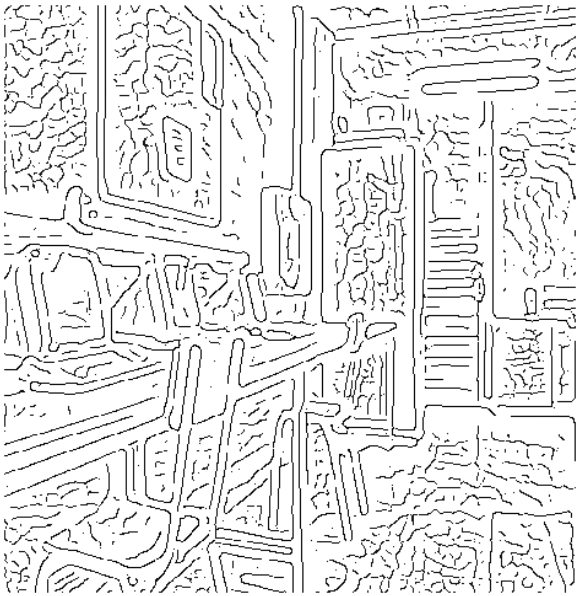
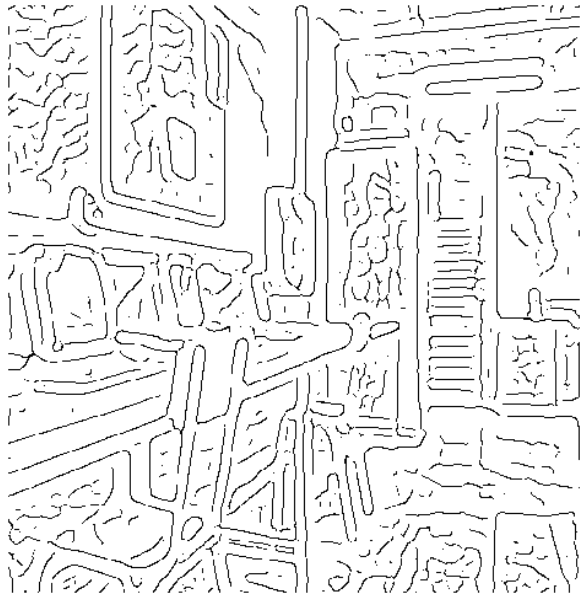


  - Compute the gradient magnitude at points B and C by linear interpolation using the magnitudes at A3-A4 and A7-A8 and the distance between the points
  - Point A is a maximum if Magnitude(A) > Magnitude (B) and Magnitude(A) >= Magnitude (C)

Gradient Maxima

$\sigma = 1, 2, 3, 4$

Edge Detection using the Laplacian

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

	$\nabla^2 x$	$\nabla^2 y$
2^a *diferencia*	1 −2 1	1 / −2 / 1

- Typical Laplacian operators:

0	1	0
1	−4	1
0	1	0

1	4	1
4	−20	4
1	4	1

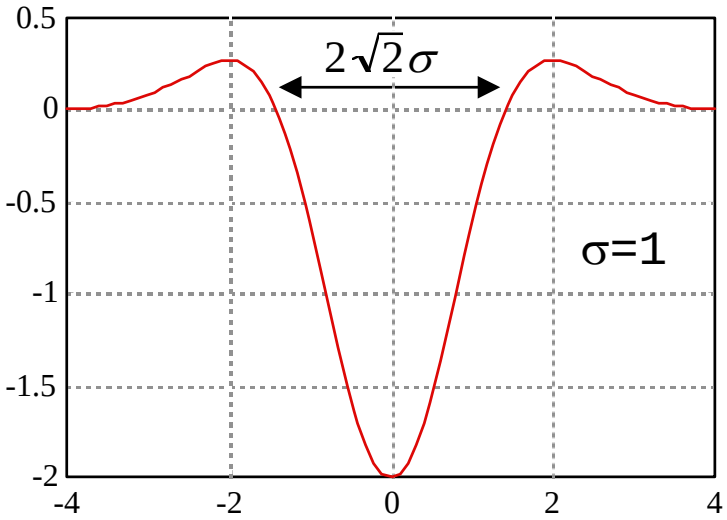
  - Zero crossings give continuous and closed contours (why?)
  - The 2nd derivative amplifies the noise twice: smoothing is mandatory

Marr-Hildreth operator

- Gaussian Filter + Laplacian

$$\nabla^2 * G_\sigma * f(i,j) = \left(\nabla^2 G_\sigma\right) * f(i,j)$$

$$\nabla^2 G_\sigma(r) = \frac{r^2 - 2\sigma^2}{\sigma^4} \exp\left(\frac{-r^2}{2\sigma^2}\right)$$



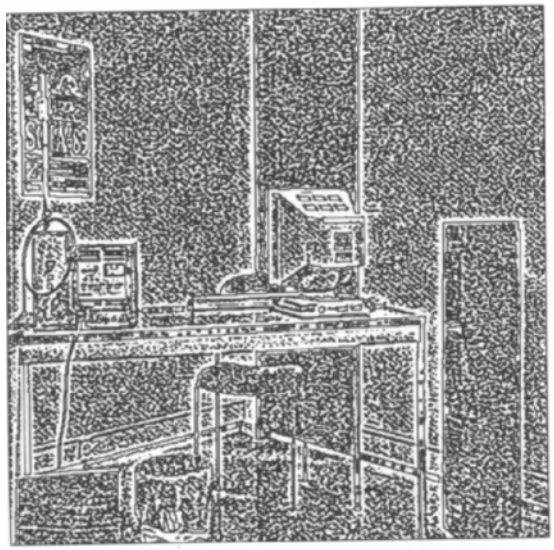
Inverted Mexican Hat

Finding Zero-Crossings

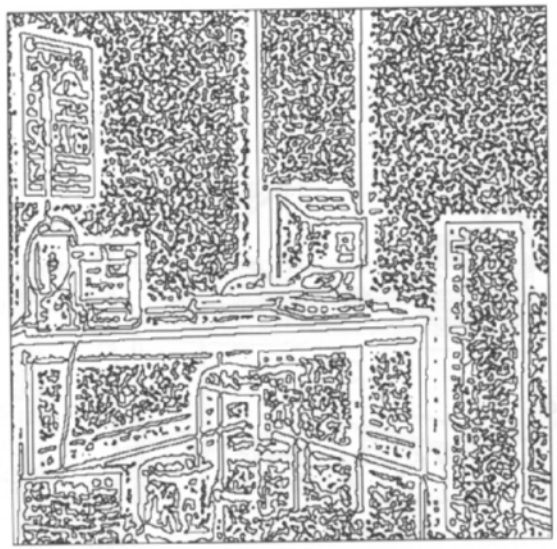
- Only keep crossings bigger than a threshold

```
mag_y:= 0
mag_x:= 0
if g(i,j) >= 0 then
  if g(i+1,j) < 0 then
    mag_y:= g(i,j) - g(i+1,j)
  endif
  if g(i,j+1) < 0 then
    mag_x:= g(i,j) - g(i,j+1)
  endif
else
  if g(i+1,j) >= 0 then
    mag_y:= g(i+1,j) - g(i,j)
  endif
  if g(i,j+1) >= 0 then
    mag_x:= g(i,j+1) - g(i,j)
  endif
endif
magnitude(i,j):= max(mag_y,mag_x)
```

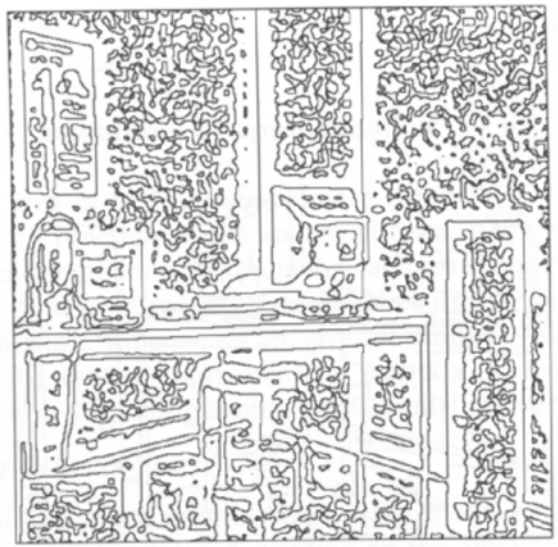
Edges Obtained at Zero-Crossings



Zero crossings of $\nabla^2 G$ with σ=1.



Zero crossings of $\nabla^2 G$ with σ=2.



Zero crossings of $\nabla^2 G$ with σ=3.

Figure 4.10 The zero-crossings of the Marr-Hildreth operator applied to the image in figure 4.9.

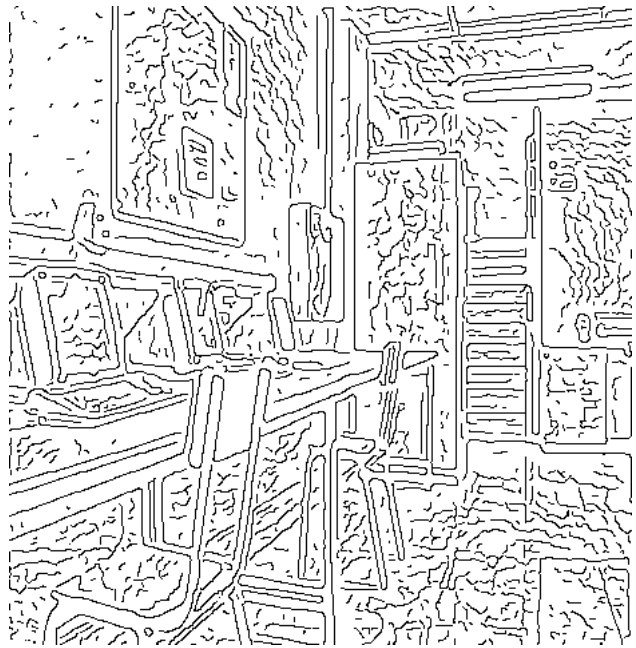
4. Contour segmentation

- Goal:
  - Obtain straight lines or curves
  - Remove noisy contours
- Main techniques:
  - Local analysis:
    - » Contour following
      - Hysteresis thhresholding
    - » Line or curve fitting
      - Split and merge
      - Total least squares
  - Hough Transform

Hysteresis thresholding

- Goal: obtain chains of edge points with a given intensity (magnitude of gradient or zero-crossing)
- Finding a good threshold may be difficult

th = 8
broken contours



th = 4
too much noise

Hysteresis thresholding

- An contour chain is required to have:
  - All pixels with magnitude >= th_inf
  - At least one pixel with magnitude >= th_sup
- Remove short contour chains

Th_inf=4, Th_sup=8

Th_inf=4, Th_sup=8, Length>30

Contour Following

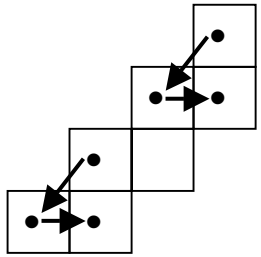
```
procedure get_chains
  for i:= 1 to num_rows
    for j:= 1 to num_cols
      start:=(i,j)
      if magnitude(start) >= th_sup then
        n:= 0
        follow(start,chain,n)
        reverse(chain, n)
        follow(next(start),chain,n))
        if n >= minimal_length then
          store(chain,n)
        endif
      endif
    endfor
  endfor
end get_chains

procedure follow(current, in out chain, in out n)
  while current <> nil
    n:=n+1
    chain(n):= current
    magnitude(current):= 0
    current:= next(current)
  endwhile
end follow

function next(current) return pixel
  for next in neighbours(current)
    if magnitude(next) >= th_inf then
      return next
    endif
  endfor
  return nil
end next
```

In which order?

Contour Following: which order?

- Reading order

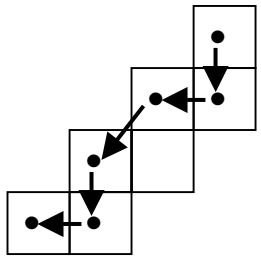
1	2	3
4		5
6	7	8



Contour Broken

- 4-neighbours first

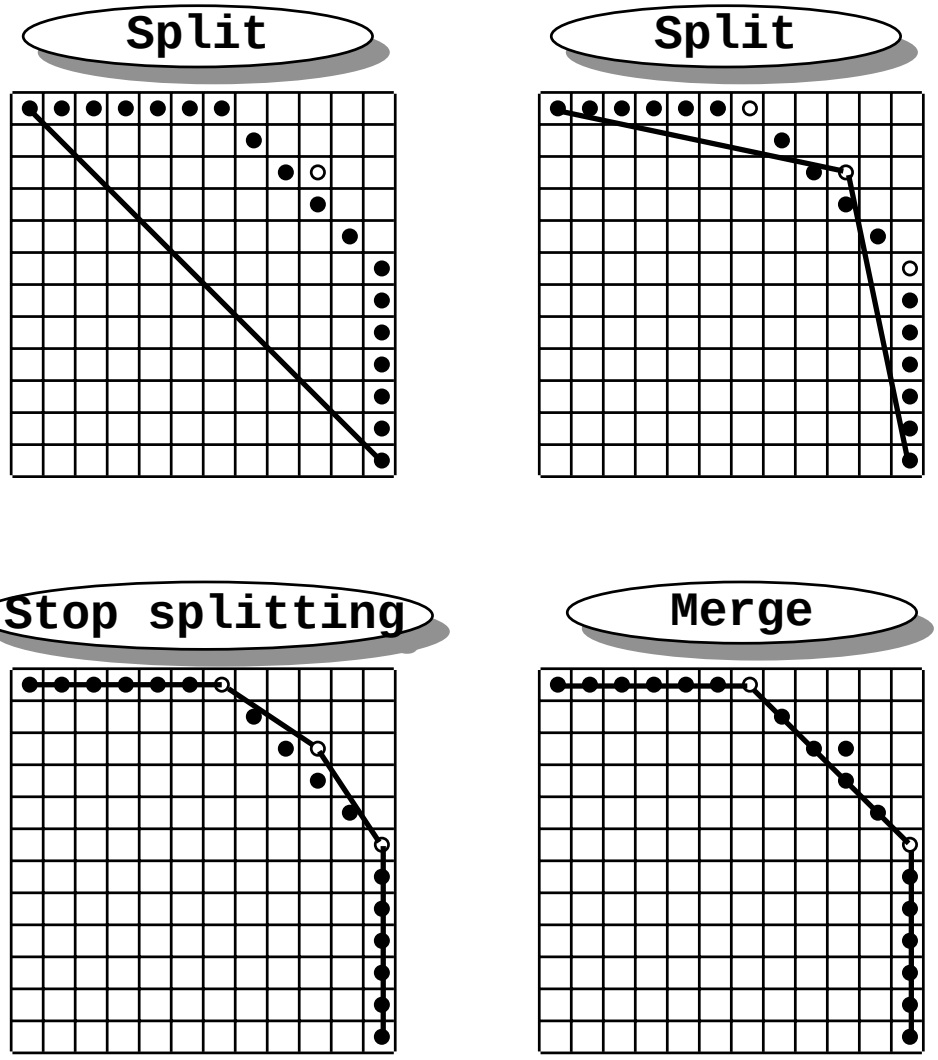
5	1	6
2		3
7	4	8



Split-and-Merge

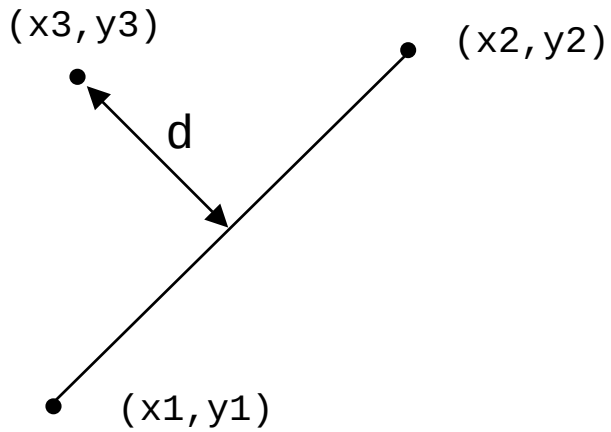
1. Split (recursive):
   - Obtain the line passing thru both chain endpoints
   - Find the pixel further away from the line
   - If distance > threshold (e.g. 2 pixels), split at this point
   - Recursion for both chain sides
2. Merge:
   - Merge two neighbouring lines if the distances of all the intermediate points to the merged line is smaller that the threshold
3. Remove short lines and lines with small gradient magnitude
4. Obtain the line passing thru the chain endpoints or perform a total least-squares line fitting

Split-and-Merge: Example



Computing distances

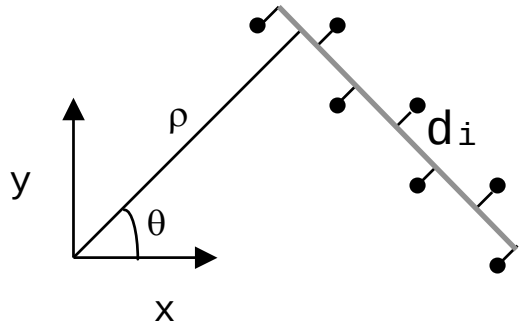
- Distance from point 3 to the line passing through points 1 and 2:



$$d = \frac{\left| x_3(y_1 - y_2) - y_3(x_1 - x_2) + y_2 x_1 - y_1 x_2 \right|}{\sqrt{(y_1 - y_2)^2 + (x_1 - x_2)^2}}$$

Least-Squares Line Fitting

- Total least-Squares (total regression)
  - Find the line that minimizes the squared sum of perpendicular distances from all the points to the line:



$$\text{Line Equation:} \quad x\cos\theta + y\sin\theta - \rho = 0$$

$$\text{Distance of pixel i:} \quad d_i = x_i\cos\theta + y_i\sin\theta - \rho$$

$$\text{Minimize:} \quad D^2 = \sum_i \left(x_i\cos\theta + y_i\sin\theta - \rho\right)^2$$

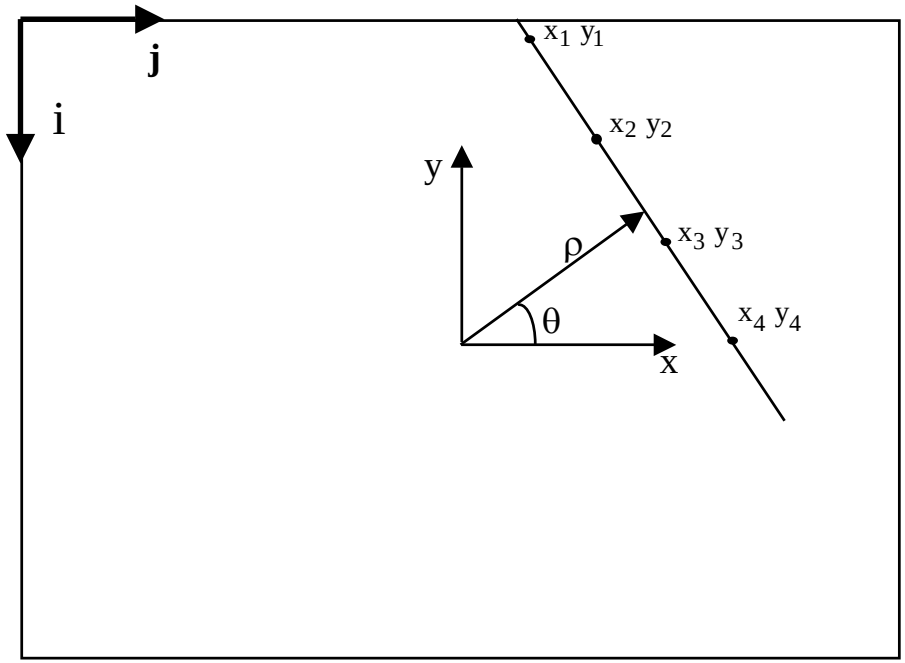
- Solution: same that finding the axis of minimal inertia
  - Compute the position and orientation of the “blob”
- Endpoints: compute the perpendicular projection of the first and last pixels on the line (how?)

Contour chains and lines obtained

4.2 Hough Transform

- Global detection of lines or curves
- Using parametric representation
- Parameters defining a line: ρ y θ

$$\rho = x \cos\theta + y \sin\theta$$



$$x = j - num\_cols/2$$

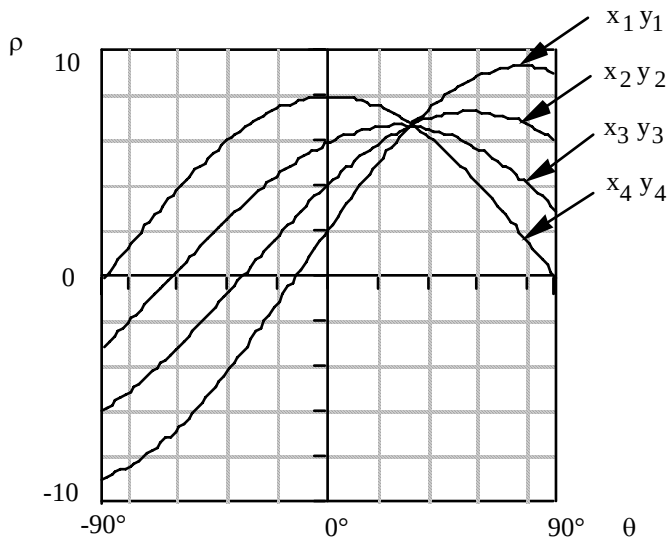
$$y = num\_rows/2 - i$$

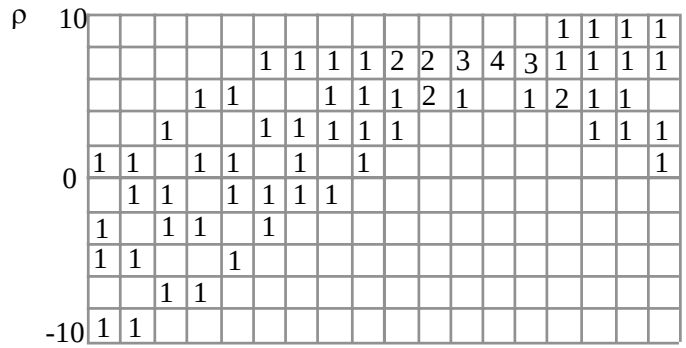
Hough Transform

- A pixel (x_i, y_i) belongs to all lines with equation:

$$\rho = x_i \cos\theta + y_i \sin\theta$$

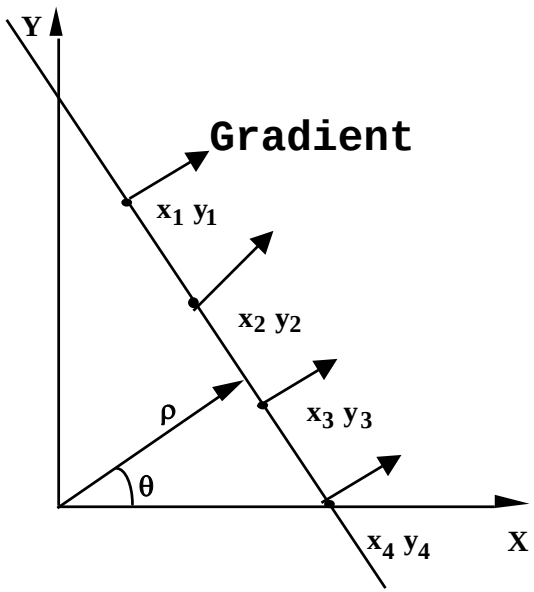
- The space (ρ,θ) is divided in cells
- Each pixel votes for the lines passing through itself
- Find the most voted lines

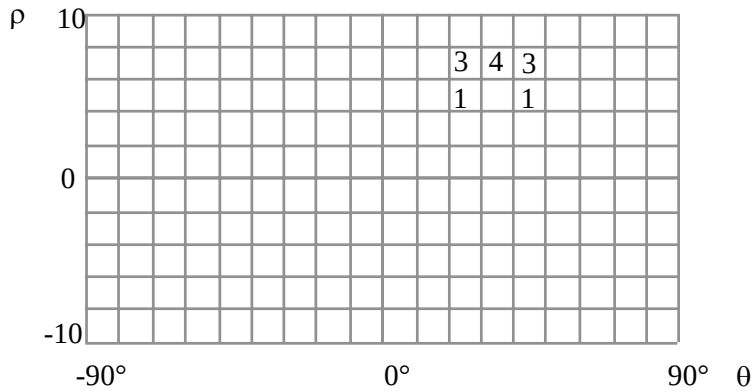




Hough Tr.: Using the Orientation

- Pixels only vote for lines with orientation similar to its gradient orientation
  - Improves the computing time





Hough Tr.: Using the Orientation

Without Orientación

```
for i:= 1 to num_rows
  for j:= 1 to num_cols
    if magnitude(i,j)>=th
      x:= j - num_cols /2
      y:= num_rows/2 - i
      for θ:=θmin to θmax
        ρ:= x*cos(θ) + y*sin(θ)
        vote_to_line(i,j,ρ,θ)
      endfor
    endif
 endfor
endfor
```

Using Orientación

```
for i:= 1 to num_rows
  for j:= 1 to num_cols
    if magnitude(i,j)>=th
      x:= j - num_cols /2
      y:= num_rows/2 - i
      θ:= orientation(i,j)
      ρ:= x*cos(θ) + y*sin(θ)
      vote_to_line(i,j,ρ,θ)
    endif
  endfor
endfor
```

Hough Transform for Circles

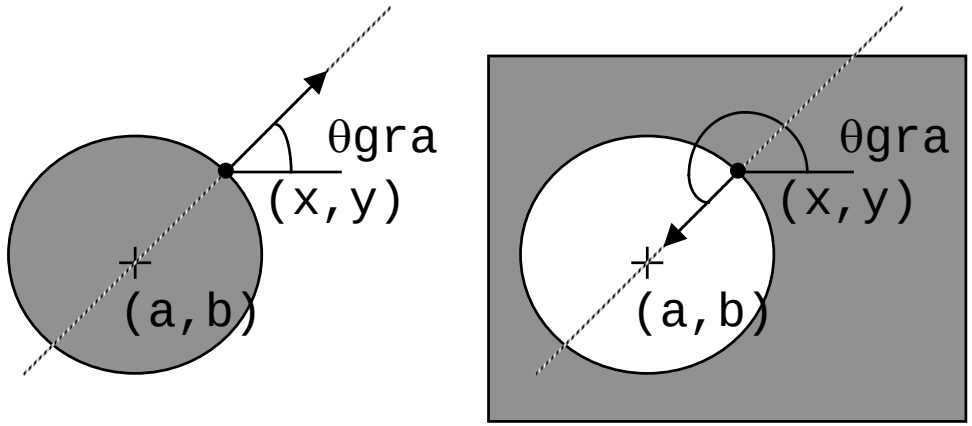
- Valid for curves, but the number of parameters increases
- Example: circle
  - Radius and centre (r, a, b)
  - Parametric form (polar)

$$r^2 = (x-a)^2 + (y-b)^2$$

$$x = a + r\cos\theta$$

$$y = b + r\sin\theta$$

  - Using gradient orientation:



$$a = x - r\cos\theta \qquad a = x + r\cos\theta$$

$$b = y - r\sin\theta \qquad b = y + r\sin\theta$$

Hough Transform for Circles

- 3D voting table: (r,a,b)

Using Orientation

```
for i:= 1 to num_rows
  for j:= 1 to num_cols
    if magnitude(i,j)>=th
      x:= j - num_cols /2
      y:= num_rows/2 - i
      θ:= orientation(i,j)
      for r:= rmin to rmax
        a:= x - r*cos(θ)
        b:= y - r*sin(θ)
        vote_circle(i,j,r,a,b)
      endfor
    endif
  endfor
endfor
```

Finding the centre of a corridor

- Edge points vote for a point of the horizon (all?)