Segmentation
University of the Philippines
August 2006
Diane Lingrand
University of Nice – Sophia Antipolis, France
lingrand@polytech.unice.fr
http://www.polytech.unice.fr/~lingrand
Why edge detection?

- in order to detect an object in a scene
  - Example: a robot that wants to play soccer
- in order to do measures
  - objects verification in a factory
- in order to extract informations
- in order to compress image data
Which primitives?

- Edges
- Regions
- Point of interest or corners
- Patterns
Course's overview

• **Edge detection**
  – A simple example
  – Convolution approach
    • $1^{st}$ derivative (Sobel, Kirsch, Prewitt, ...)
    • $2^{nd}$ derivative (Laplacien, ...)
  – Optimal filtering
    • Canny, Deriche

• **Regions detection**
  – similarity
  – Hough transform
  – contour deformation
Where are the edges?
Where are the edges? (2)

- Let us begin with a very simple image:
Where are the edges? (3)

- Let us look closer to the pixels:
A first filter (1)

• Let us build an new image where each pixel \((i,j)\) is obtained from the difference between pixel at \((i,j)\) and pixel at \((i-1,j)\)
  – pixels are still between 0 and 255: we need to transform the results
  – positive and negative differences: we only need the absolute value
  – be careful of image borders!
A first filter (2)

- Result on our simple image
A first filter (3)

- Result on another image

Wrong!
Enhancing our first filter

- **We will compose:**
  - a vertical edge detector
    - difference between \((i,j)\) and \((i-1,j)\):
      - \(I_{cv}(i,j) = I(i,j) - I(i-1,j)\)
  - and an horizontal edge detector
    - difference between \((i,j)\) and \((i,j-1)\):
      - \(I_{ch}(i,j) = I(i,j) - I(i,j-1)\)
  - magnitude: \(I_c = \sqrt{I_{cv}(i,j)^2 + I_{ch}(i,j)^2}\)
Let us try on a single image

- Image without noise:
Let us try on a real image

$v$ filter

$h$ filter

results $\gamma = 0.3$
More in details
Other image:
Conclusion on our first filter

• **Drawbacks:**
  – sensitive to noise
  – too many edges detected
  – more sensitive to horizontal and vertical edges than the others

• **Advantages:**
  – fast
  – not difficult to implement
The ideal edge

- edge = where the intensity is not continuous
- maxima of the derivative
Intensity derivative

• Definition:

\[
\frac{df}{dx}(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

with \( h \) at a maximum of 1

• Approximation:

\[
\frac{\partial f}{\partial x} = I[x + 1] - I[x]
\]

\textbf{that is not really correct}
Convolution

- Convolution operator with a kernel $K$

$$I_2(i, j) = \sum_{k=0}^{2} \sum_{l=0}^{2} I_1(i + k - 1, j + l - 1)K(k, l)$$
Convolution in Java

```java
float [] filtre = {-1.0f, 0.0f, 1.0f,
                   -2.0f, 0.0f, 2.0f,
                   -1.0f, 0.0f, 1.0f};
Kernel kernel = new Kernel(3, 3, filtre);
ConvolveOp cop = new ConvolveOp(kernel,
                                 ConvolveOp.EDGE_NO_OP, null);
cop.filter(input, output);
```

or:

```java
ConvolveOp.EDGE_ZERO_FILL
```

Roberts filters (1965)

\[
\begin{align*}
\frac{dI}{dx} &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \\
\frac{dI}{dy} &= \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}
\end{align*}
\]

- **Magnitude** = edge strength
- **Direction of edge normal**

\[
\sqrt{\left(\frac{dI}{dx}\right)^2 + \left(\frac{dI}{dy}\right)^2} \quad \text{ou} \quad \max\left(\left|\frac{dI}{dx}\right|, \left|\frac{dI}{dy}\right|\right)
\]

\[
\arctan\left(\frac{\frac{dI}{dy}}{\frac{dI}{dx}}\right)
\]
Example

x filter

y filter

magnitude of result
Sensibility to noise

- Let us assume: \( I_{\text{obs}} = I_{\text{signal}} + \varepsilon \sin(\omega x) \)
- Derivative: \( I'_{\text{obs}} = I'_{\text{signal}} + \varepsilon \omega \cos(\omega x) \)
- \( \omega = 2\pi f \): high frequencies will disturb the signal derivative
- we need to eliminate these high frequencies: smoothing
  - mean smoothing
  - gaussian smoothing
  - exponential filtering, ...
Mean smoothing

- ex: filter 3x3

```
1 1 1
1 1 1
1 1 1
```

/ 9

![Graphs showing mean smoothing effect](image)
Gaussian smoothing
(Marr, 1980)

- $h(x, y) = \frac{1}{(2\pi \sigma^2)} \exp(-\frac{x^2 + y^2}{2\sigma^2})$
- truncated and discretised gaussian:

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & 4 & 1 \\
0 & 1 & 0 \\
\end{array} \quad /8
\]

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 8 & 1 \\
1 & 1 & 1 \\
\end{array} \quad /16
\]
Sobel filter (1970)

- Convolution:
  - smoothing $[1 \ 2 \ 1]$  
  - derivative $[1 \ 0 \ -1]$

- Mean of « derivatives » at $x$ and $x-1$

\[
\frac{I[x + 1] - I[x - 1]}{2}
\]

\[
\frac{dI}{dx} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} / 4
\]

\[
\frac{dI}{dy} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} / 4
\]
Sobel: results (1)

v filter

h filter

magnitude of result
Closer
Sobel : results (2)
Sobel : results (3)
Other filters: using more directions

- **Prewitt (1970)**

\[
\begin{pmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{pmatrix},
\begin{pmatrix}
1 & 1 & 0 \\
1 & 0 & -1 \\
0 & -1 & -1
\end{pmatrix},
\begin{pmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1
\end{pmatrix} / 3
\]

- **Kirsch (1971)**

\[
\begin{pmatrix}
5 & -3 & -3 \\
5 & 0 & -3 \\
5 & -3 & -3
\end{pmatrix},
\begin{pmatrix}
5 & 5 & -3 \\
5 & 0 & -3 \\
-3 & -3 & -3
\end{pmatrix},
\begin{pmatrix}
5 & 5 & 5 \\
-3 & 0 & -3 \\
-3 & -3 & -3
\end{pmatrix} / 15
\]
Enhancement : thresholding

- we eliminate all the pixels that have a value below a minimum threshold
How to choose the threshold?

- Low threshold: all edges are detected but we have false positives
- High threshold: all the pixels detected are edge pixels but we are missing some of them (false negatives)
Enhancement (2)

- Hysteresis thresholding

\[\text{high thres.} \quad \text{low thres.}\]

\[\text{belongs to the edge}\]
Laplacian

- Laplacian $\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$

- Smoothing and Laplacian: Laplacian of a gaussian

\[
\begin{pmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0
\end{pmatrix} / 4
\]

\[
\begin{pmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{pmatrix} / 8
\]
Laplace : results (1)

$\gamma = 0.3$
Laplace : results (2)

\[ \gamma = 0.3 \]
DOG Laplacian (80's) Marr and Hildreth

- Laplacian seen as the difference between two gaussian smoothing filters with different kernels
- Edge image: difference between a weakly smoothed image and a strongly smoothed image
  - try it on computers
Huertas-Médioni (1986)

- Laplacian of a gaussian:

\[
\Delta g(x, y) = \frac{1}{G_0} \left( 2 - \frac{x^2 + y^2}{2\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

- Separation: \(g_1(x).g_2(y) + g_1(y).g_2(x)\) with:

\[
g_1(x) = \frac{1}{\sqrt{2G_0}} \left( 1 - \frac{x^2}{2\sigma^2} \right) e^{-\frac{x^2}{2\sigma^2}}
\]

\[
g_2(x) = \frac{1}{\sqrt{2G_0}} e^{-\frac{x^2}{2\sigma^2}}
\]

Example: \(1/G_0 = 4232\) et \(\sigma^2 = 2\) :

\[g_1 = [-1 -6 -17 -17 18 46 18 -17 -17 -6 -1] \]

\[g_2 = [0 1 5 17 36 46 36 17 5 1 0] \]
What about color?

- Detection on each color component (R, G, B) separately and recomposition of the color after filtering?
- Max of the results on each component?
- Mean?
- Luminance (Y)?
Exercise

- In file ConvolutionFilter.java
  - implement the convolution function
    public BufferedImage filter(BufferedImage bin)
- Try the following filters :
  - Sobel, Prewitt, Kirsch, Laplace
- Try different gaussian filters
  - using different kernels
- Combine filters, smoothings et thresholding
  - keep the best results
public ConvolutionFilter(int type) {
    super();
    name = NAMES[type] + " filter";
    filters = new ArrayList();
    double [][] filter;
    switch(type) {
    case NAIVE_X:
        filter = new double [3][3];
        filter[0][0] = 0; filter[0][1] = 0; filter[0][2] = 0;
        filter[1][0] = -1; filter[1][1] = 1; filter[1][2] = 0;
        filter[2][0] = 0; filter[2][1] = 0; filter[2][2] = 0;
        filters.add(filter);
        break;
    case NAIVE_Y:
        filter = new double [3][3];
        filter[0][0] = 0; filter[0][1] = -1; filter[0][2] = 0;
        filter[1][0] = 0; filter[1][1] = 1; filter[1][2] = 0;
        filter[2][0] = 0; filter[2][1] = 0; filter[2][2] = 0;
        filters.add(filter);
        break;
    case NAIVE:
        filter = new double [3][3];
        filter[0][0] = 0; filter[0][1] = 0; filter[0][2] = 0;
        filter[1][0] = -1; filter[1][1] = 1; filter[1][2] = 0;
        filter[2][0] = 0; filter[2][1] = 0; filter[2][2] = 0;
        filters.add(filter);
        break;
    case GAUSSIAN3:
        filter = GAUSSIAN3_FILTER;
        filters.add(filter);
        break;
    case GAUSSIAN5:
        filter = GAUSSIAN5_FILTER;
        filters.add(filter);
        break;
    default:
    }
}

public BufferedImage filter(BufferedImage bin) {
    int w = bin.getWidth();
    int h = bin.getHeight();
    BufferedImage bout = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
    for(int i = 0; i < w; i++ )
    for(int j = 0; j < h; j++ )
    bout.setRGB(i,j,0);

    int nbFilters = filters.size();
    switch(nbFilters) {
    case 0:
        return bout;
        break;
    case 1:
    ....
Optimal filtering

- Canny (1986)
- impulsional response $h(x)$ such as:
  - good detection
  - good localisation
  - weak multiplicity of maxima due to noise
- we search for an edge
  - of magnitude $A$
  - with an additive gaussian noise of null mean and variance $n_0^2$
Quality 1: good detection

- The more smoothing, the better the quality of the detection
  - we want to maximize the ratio signal by noise:

\[
\Sigma = \frac{A \int_{-\infty}^{0} f(x) \, dx}{n_0 \int_{-\infty}^{\infty} f^2(x) \, dx}
\]
Quality 2: Localisation

- The less smoothing, the better the localisation.
  - we want to minimize the variance of the positions of roots. It is equivalent to maximize:

\[
\Lambda = \frac{A|f'(0)|}{n_0 \int_{-\infty}^{\infty} f'^2(x) \, dx}
\]
Quality 3: unique response

- We want an unique response for one edge: we limit the distance between 2 maxima

\[ x_{\text{max}} = 2\pi \left( \sqrt{\frac{\int_{-\infty}^{\infty} f'^2(x) \,dx}{\int_{-\infty}^{\infty} f''^2(x) \,dx}} \right) \]
General optimal filter

• Trade-off between $\Sigma$ and $\Lambda$
• Find $f$ maximising $\Sigma \Lambda$ under the constraint: $x_{\text{max}} = k$

which consist to solve the differential equation:

$$2f(x) - 2 \lambda_1 f''(x) + 2\lambda_2 f'''(x) + \lambda_3 = 0$$

of general solution:

$$f(x) = a_1 e^{\alpha x} \sin(\omega x) + a_2 e^{\alpha x} \cos(\omega x) + a_3 e^{-\alpha x} \sin(\omega x) + a_4 e^{-\alpha x} \cos(\omega x)$$
Optimal filtering : Canny

• finite impulse response filter [-M;+M]
• constraints:
  \( x_{\text{max}} = kM \)
  \( f(0) = 0; f(M) = 0; \)
  \( f'(0) = S; f'(M) = 0; \)
  \( x<0 : f(x) = -f(-x) \)
• numerical optimisation : \( \Sigma \Lambda = 1.12 \)
• approximation : derivative of a gaussian (\( \Sigma \Lambda = 0.92, k = 0.51, 20\% \) lower performance)
Optimal filtering : Deriche (1987)

- infinite impulse response filtering
- constraints:
  \[ f(0) = 0 ; f(+\infty) = 0 ; f'(0) = S ; f'(+\infty) = 0 ; \]
- where:

\[
    f(x) = \frac{S}{\omega} e^{-\alpha|x|} \sin(\omega x)
\]

\[
    \Lambda = \sqrt{2\alpha}, \quad \Sigma = \sqrt{\frac{2\alpha}{\alpha^2 + \omega^2}}
\]

\[
    \Sigma \Lambda = \frac{2\alpha}{\alpha^2 + \omega^2}, \quad k = \sqrt{\frac{\alpha^2 + \omega^2}{5\alpha^2 + \omega^2}}
\]
Optimal filtering: Deriche (2)

• Let us set $\alpha = m\omega$:

<table>
<thead>
<tr>
<th>Case</th>
<th>Condition</th>
<th>$\Lambda$</th>
<th>$\Sigma$</th>
<th>$\Sigma\Lambda$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1</td>
<td>$m \gg 1$</td>
<td>$\sqrt{2\alpha}$</td>
<td>$\sqrt{\frac{2}{\alpha}}$</td>
<td>2</td>
<td>0.44</td>
</tr>
<tr>
<td>case 2</td>
<td>$m = 1$</td>
<td>$\sqrt{2\alpha}$</td>
<td>$\sqrt{\frac{1}{\alpha}}$</td>
<td>$\sqrt{2}$</td>
<td>0.58</td>
</tr>
<tr>
<td>case 3</td>
<td>$m = \sqrt{3}$</td>
<td>$\sqrt{2\alpha}$</td>
<td>$\sqrt{\frac{3}{2\alpha}}$</td>
<td>$\sqrt{3}$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- case 1: the best, corresponds to $Sx e^{-\alpha|x|}$
- case 2: for same value of $k$, performance index of Deriche increase of 25 %
- case 3: for same value of $k$, performance index of Deriche increase of 90 %
Optimal filtering: Deriche (3)

- optimal integrator: $S(\alpha|x|+1)e^{-\alpha|x|}$
- bidimensionnal filters:
  
  $f_x(x,y) = k_1 m e^{-\alpha|m|} k_2 (\alpha |n|+1) e^{-\alpha|n|}$

  $f_y(x,y) = k_1 (\alpha |m|+1) e^{-\alpha|m|} k_2 n e^{-\alpha|n|}$

- recursive implementation
Recursive implementation

• Phase 1
  for m from 0 to (w-1) :
    for n from 0 to (h-1) :
      \[ y_1(m, n) = a_1I_1(m, n) + a_2I_1(m, n - 1) + b_1y_1(m, n - 1) + b_2y_1(m, n - 2) \]
    for n from (h-1) to 0 :
      \[ y_2(m, n) = a_3I_1(m, n + 1) + a_4I_1(m, n + 2) + b_1y_2(m, n + 1) + b_2y_2(m, n + 2) \]
    for n from 0 to (h-1) :
      \[ r(m, n) = c_1(y_1(m, n) + y_2(m, n)) \]

• Phase 2
  for n from 0 to (h-1) :
    for m from 0 to (w-1) :
      \[ y_1(m, n) = a_5r(m, n) + a_6r(m - 1, n) + b_1y_1(m - 1, n) + b_2y_1(m - 2, n) \]
    for m from (w-1) to 0 :
      \[ y_2(m, n) = a_7r(m + 1, n) + a_8r(m + 2, n) + b_1y_2(m + 1, n) + b_2y_2(m + 2, n) \]
  for m from 0 to (w-1) :
    \[ I_2(m, n) = c_2(y_1(m, n) + y_2(m, n)) \]
Coefficients for Deriche filters

<table>
<thead>
<tr>
<th>k</th>
<th>Smoothing</th>
<th>x-derivative</th>
<th>y-derivative</th>
<th>Laplace 1</th>
<th>Laplace 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \frac{(1-e^{-\alpha})^2}{1+2\alpha e^{-\alpha}-e^{-2\alpha}} )</td>
<td>( \frac{(1-e^{-\alpha})^2}{1+2\alpha e^{-\alpha}-e^{-2\alpha}} )</td>
<td>( \frac{(1-e^{-\alpha})^2}{1+2\alpha e^{-\alpha}-e^{-2\alpha}} )</td>
<td>( \frac{1-e^{-2\alpha}}{2\alpha e^{-\alpha}} )</td>
<td>( \frac{1-e^{-2\alpha}}{2\alpha e^{-\alpha}} )</td>
</tr>
<tr>
<td>a_1</td>
<td>( k )</td>
<td>0</td>
<td>( k )</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a_2</td>
<td>( ke^{-\alpha}(\alpha-1) )</td>
<td>1</td>
<td>( ke^{-\alpha}(\alpha-1) )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a_3</td>
<td>( ke^{-\alpha}(\alpha+1) )</td>
<td>-1</td>
<td>( ke^{-\alpha}(\alpha+1) )</td>
<td>( e^{-\alpha} )</td>
<td>1</td>
</tr>
<tr>
<td>a_4</td>
<td>( -ke^{-2\alpha} )</td>
<td>0</td>
<td>( -ke^{-2\alpha} )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a_5</td>
<td>( k )</td>
<td>( k )</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a_6</td>
<td>( ke^{-\alpha}(\alpha-1) )</td>
<td>( ke^{-\alpha}(\alpha-1) )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a_7</td>
<td>( ke^{-\alpha}(\alpha+1) )</td>
<td>( ke^{-\alpha}(\alpha+1) )</td>
<td>-1</td>
<td>( e^{-\alpha} )</td>
<td>1</td>
</tr>
<tr>
<td>a_8</td>
<td>( -ke^{-2\alpha} )</td>
<td>( -ke^{-2\alpha} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b_1</td>
<td>( 2e^{-\alpha} )</td>
<td>( 2e^{-\alpha} )</td>
<td>( 2e^{-\alpha} )</td>
<td>( e^{-\alpha} )</td>
<td>( 2e^{-\alpha} )</td>
</tr>
<tr>
<td>b_2</td>
<td>( -e^{-2\alpha} )</td>
<td>( -e^{-2\alpha} )</td>
<td>( -e^{-2\alpha} )</td>
<td>0</td>
<td>( -e^{-2\alpha} )</td>
</tr>
<tr>
<td>c_1</td>
<td>1</td>
<td>( -(1-e^{-\alpha})^2 )</td>
<td>1</td>
<td>1</td>
<td>( \frac{1-e^{-2\alpha}}{2} )</td>
</tr>
<tr>
<td>c_2</td>
<td>1</td>
<td>1</td>
<td>( -(1-e^{-\alpha})^2 )</td>
<td>1</td>
<td>( \frac{1-e^{-2\alpha}}{2} )</td>
</tr>
</tbody>
</table>
Deriche smoothing

0.25  0.5  0.75

1  2  3
Edges with first derivative by
Deriche

0.5
1
3
Laplace by Deriche: sign and zeros
An easy image

solEssi3212.jpg  after Sobel filtering
Difficult images
Regions detection

- Regions are bounded by edges
  - detection of closed curves
    - linking all edge points
    - iterative line fitting
    - finding lines or circles or ... with the Hough transform
- Classification
- Region growing algorithm
- Split and Merge algorithm
- Other approaches:
  - snakes, deformable contours, ...
Iterative endpoint fit
Hough transform

- Well known for lines search
- Based on parameterisation of geometrical objects
- Transformation to a parameters space
- Quantification of parameters
Hough transform of a line

- **Line parameterisation:**
  \[ \rho = x \cos \theta + y \sin \theta \]
  ou
  \[ y = a x + b \]
- **For each point** \((x,y)\) **labelled as edge pt:**
  - for each \(\theta\) value:
    - compute \(\rho = x \cos \theta + y \sin \theta\)
    - add a vote for \((\rho, \theta)\): \(\text{vote}[\rho][\theta]+\)
- **Quantification of the parameters space**
Why does it work? example of a line

- For each point that does not belong to the background, we look for all the lines that go through this point.
- Each founded line vote once.
- If the points are aligned, each point will vote for this line. This line will obtain more votes than any other.
Implementation (1)

- We need to determine the interval of parameter's values:
  - \( a \) in \([a_{\text{min}}, a_{\text{max}}]\)
  - \( b \) in \([b_{\text{min}}, b_{\text{max}}]\)

- We need to determine the quantification steps (uniform) : \( \Delta a, \Delta b \)

- loop : \( \text{for}(a=a_{\text{min}}; a <= a_{\text{max}}; a+= \Delta a) \{... \}

- uniform quantification of \( b \) values
Implementation (2)

• **What interval?**
  - related to the geometrical meaning of the parameters: we do not want to lose lines
  - case $y = ax + b$:
    - the slope varies from -infinite to +infinite
    - we consider values from $-h$ to $+h$
    - $b$ varies from $h(1-w)$ to $h(w+1)$
  - case $\rho = x \cos(\theta) + y \sin(\theta)$
    - $\theta$ varies from 0 to $\pi$
    - $\rho$ varies from $-\omega_0 + \rho_{\text{diag}}$ where $\rho_{\text{diag}}$: image diagonal
Implementation (3)

• Which step?
  – the smallest the step, the more accurate the solution but the biggest the image and the highest the computation time.
  – we can deal with 2 stages:
    • large interval, large step
    • small interval, small step

• Hough transform image:
  – of dimensions $(a_{\text{max}} - a_{\text{min}} + 1; b_{\text{max}} - b_{\text{min}} + 1)$
  – to be normalized between 0 and 255 before display
  – score array: indexes often result of an affine transform of real parameters values
Example of transformed images

parameters space \([a,b]\) t.q. \(y=ax+b\)

parameters space \([\rho,\theta]\) s.as \(\rho=x\cos(\theta)+y\sin(\theta)\)
Result

\[ y = a_0 x + b_0 \]

\[ \rho_0 = \cos(\theta_0)x + \sin(\theta_0)y \]
Another example with 5 lines
Lines detection in an image

- One line detection in an image:
  - finding the point of maximal vote in the Hough transform image

- Multiple lines detection in an image:
  - finding multiple local maxima in the Hough transform image
    - how many maxima?
    - all points below a threshold?

- The accuracy of the detection depends on the quantification of the parameters.
Hough transform of a curve

• Same idea as for a line
• Parameterisation:
  \[ ax^2 + bxy + cy^2 = 1 \]
• For each point \((x, y)\) in the image
  – for each value of \(a\) between \(a_{\text{min}}\) and \(a_{\text{max}}\)
    • for each value of \(b\) between \(b_{\text{min}}\) and \(b_{\text{max}}\)
      – compute \(c\) such as \(ax^2 + bxy + cy^2 = 1\)
      – ad a vote for \((a,b,c)\): \(\text{vote}[a][b][c]++\);
Just try it!

- Compute and display the Hough transform for a line:
  - \( y = ax + b \)
  - \( \rho = \cos(\theta)x + \sin(\theta)y \)
- Draw, in the original image, the line that corresponds to the maximal vote
- Find all lines in the image
- Determine the corresponding polygon
Connex components labeling

image after edge detection and thresholding

image of labels after the first browsing

image of labels after update

1 2 3 4 5 6 7 8 9 10

2 3 2 3 3 3 3

1 1 1