Visione artificiale
(a.a. 2006/07)

Edge detection

Goals of edge detection:

Primary

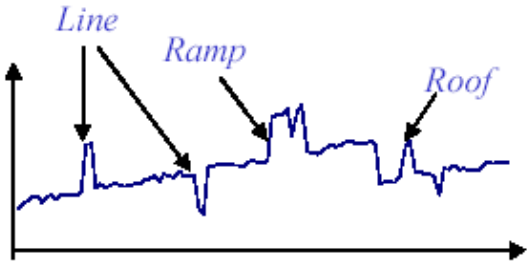
To extract information about the two-dimensional projection of a 3D scene.

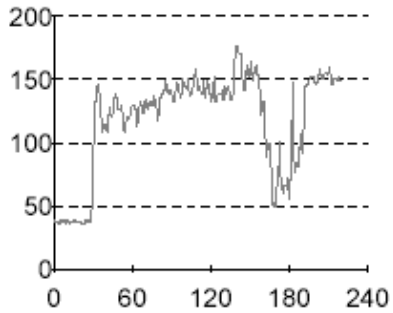
Secondary

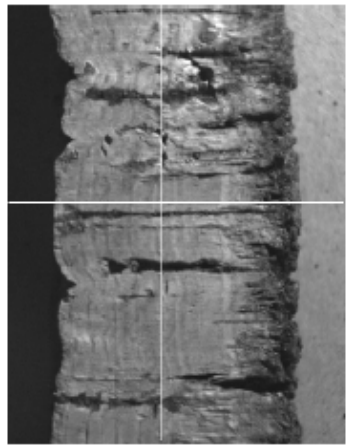
Image segmentation, region separation, object description and recognition, hand/eye tasks, ...

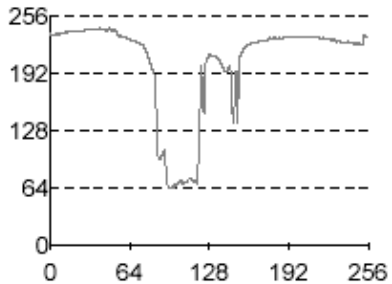
Types of Edges

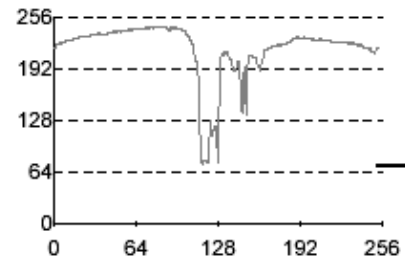
Types of Edges

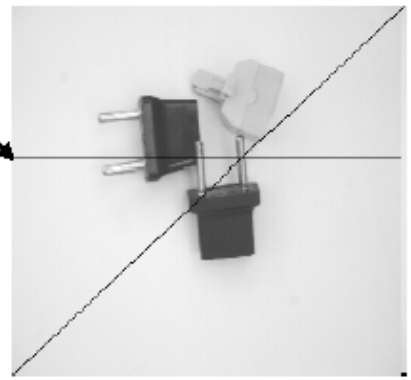


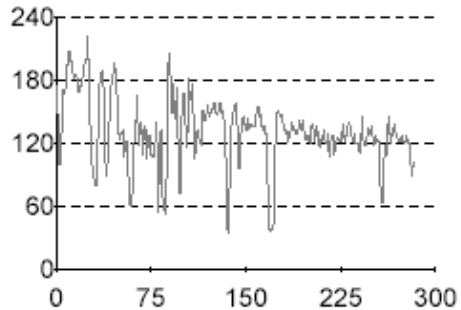


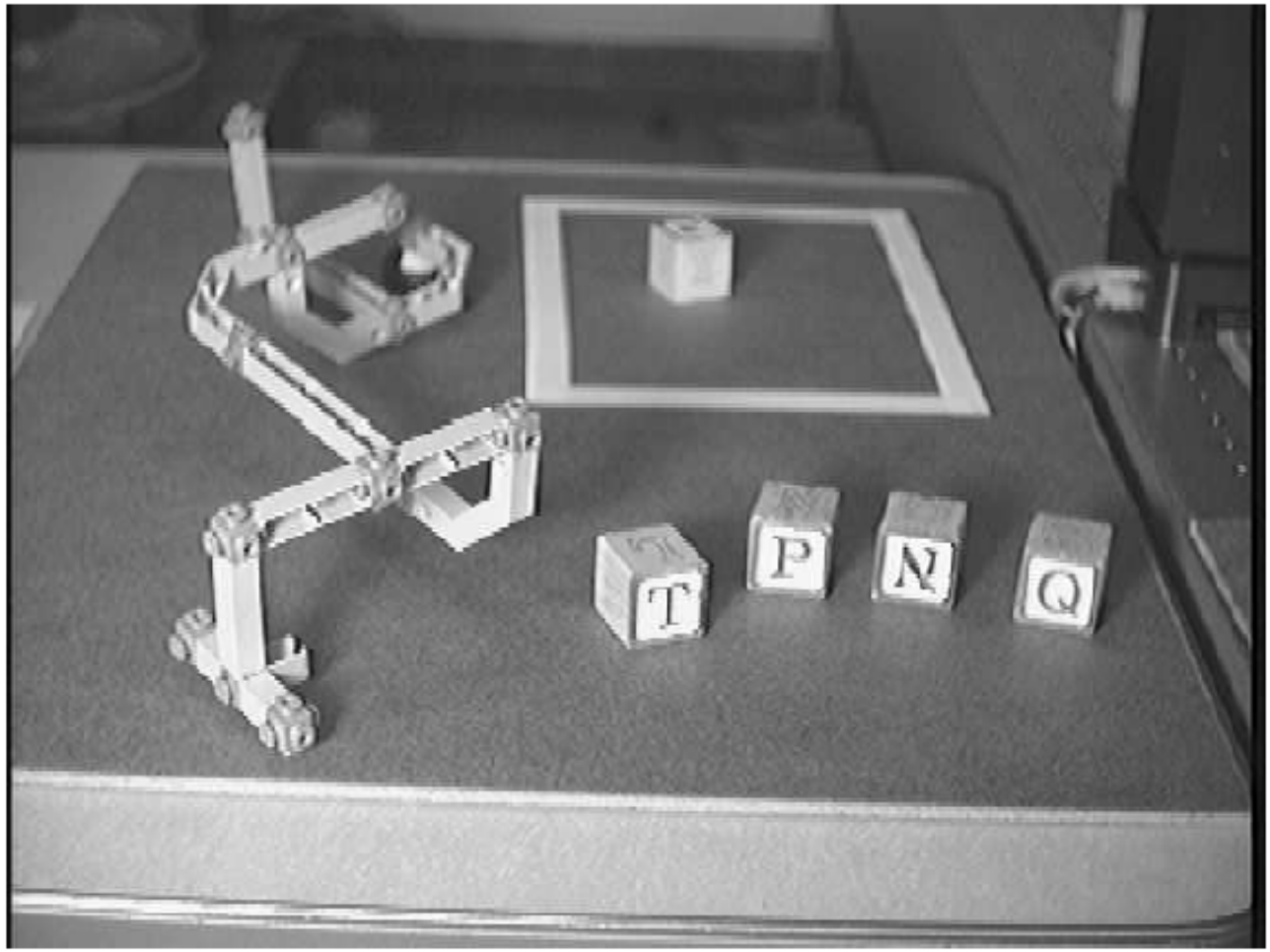


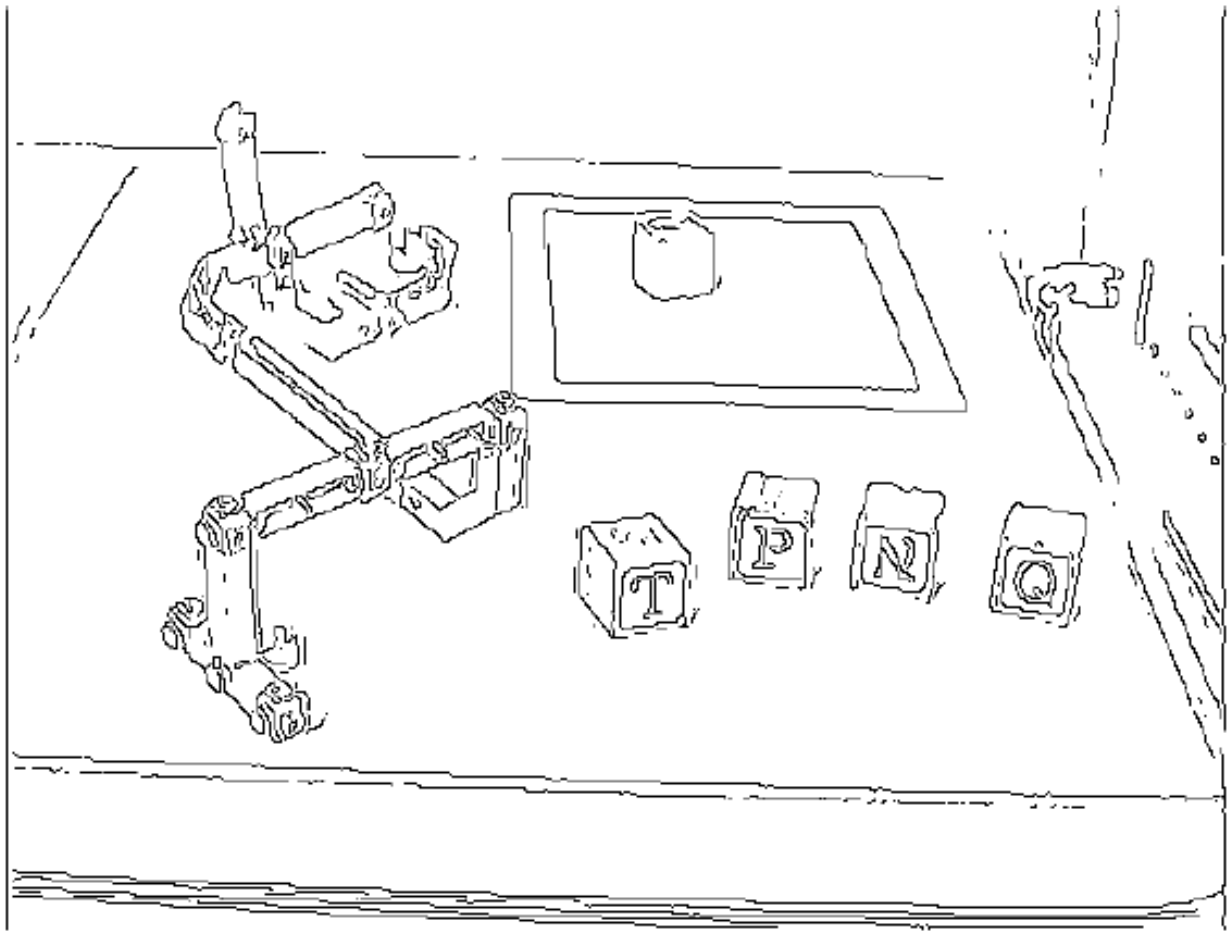


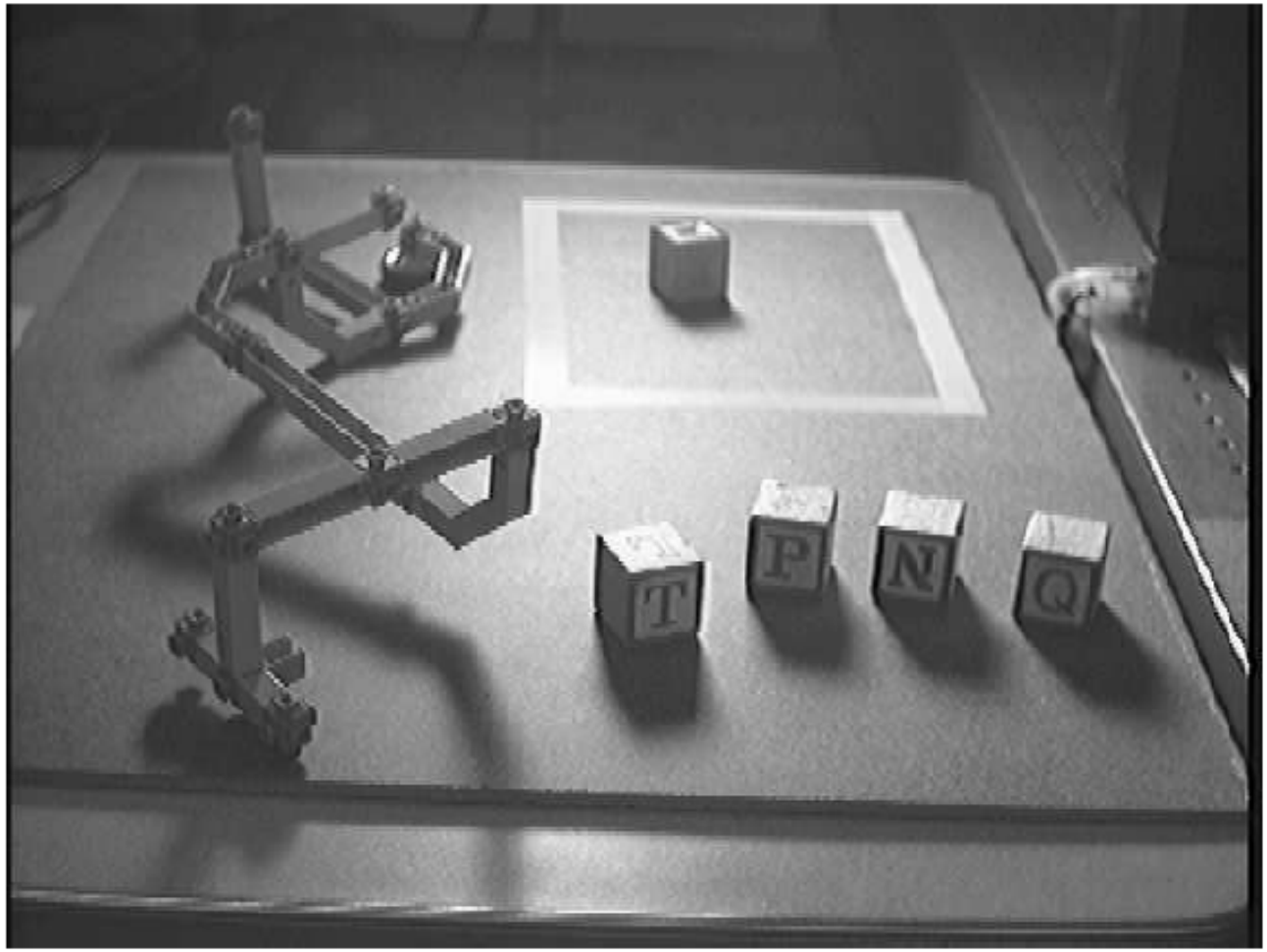


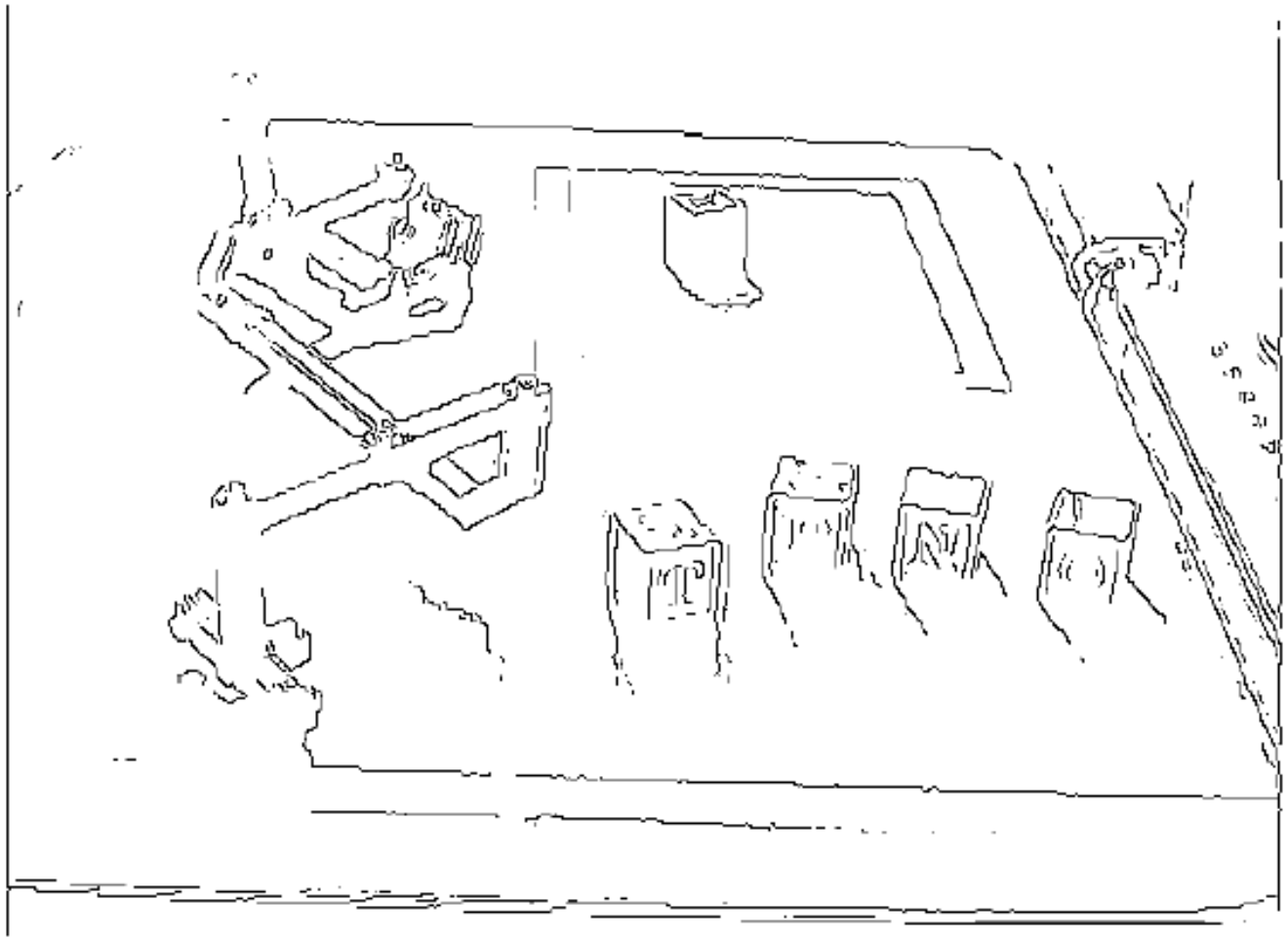





T
P
N
Q


T
P
N
Q


T
P
N
Q



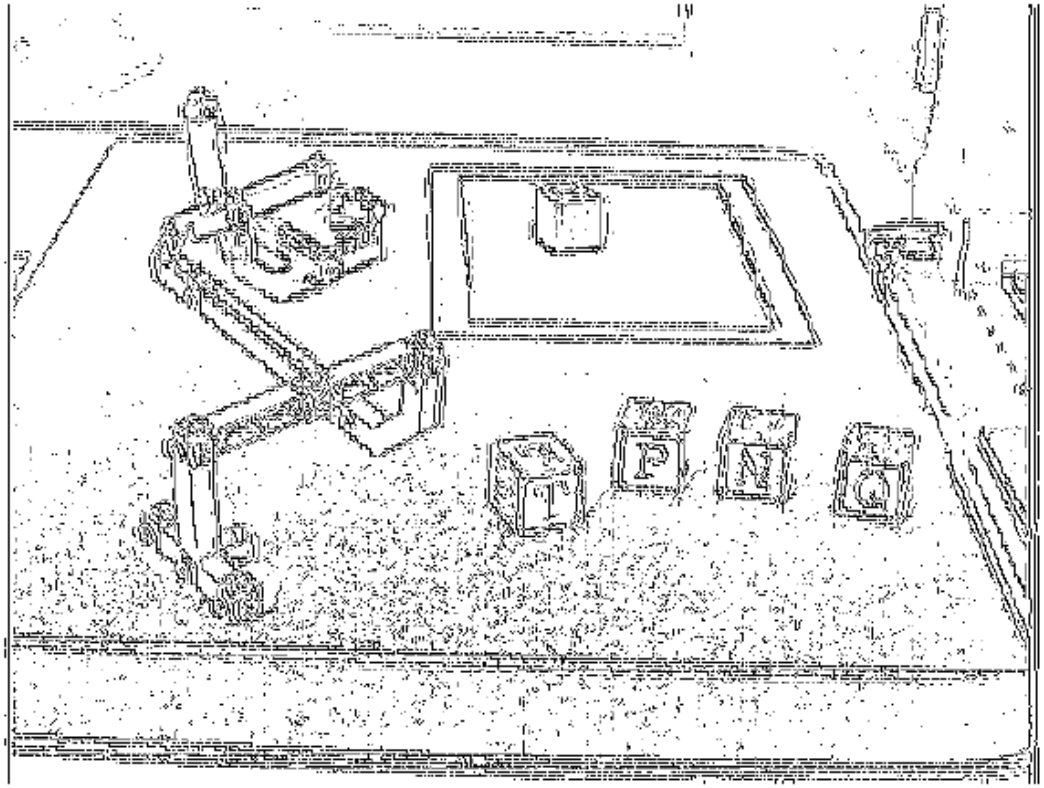
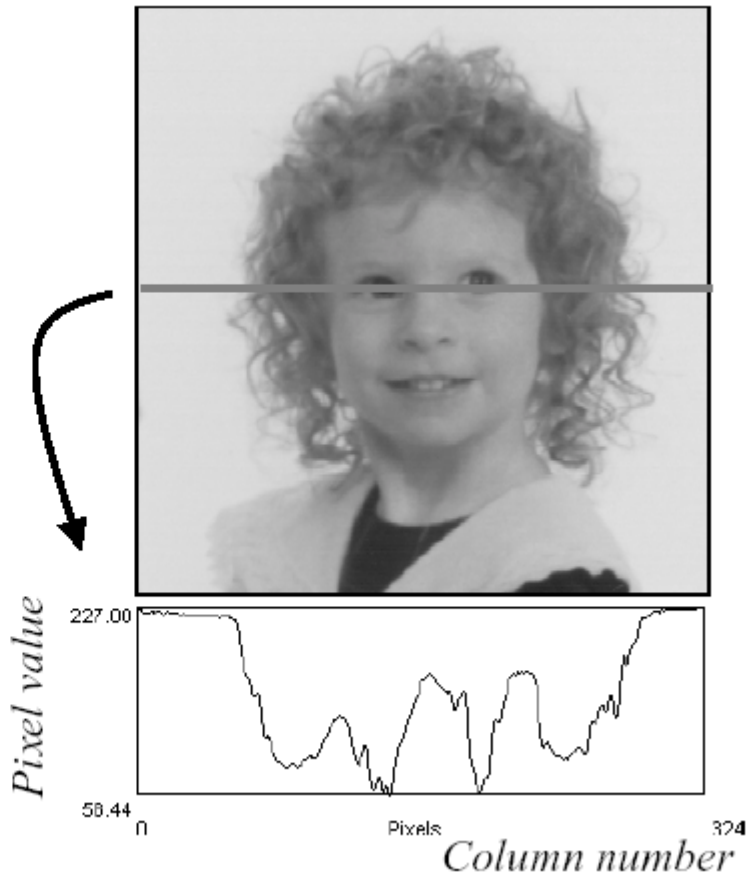
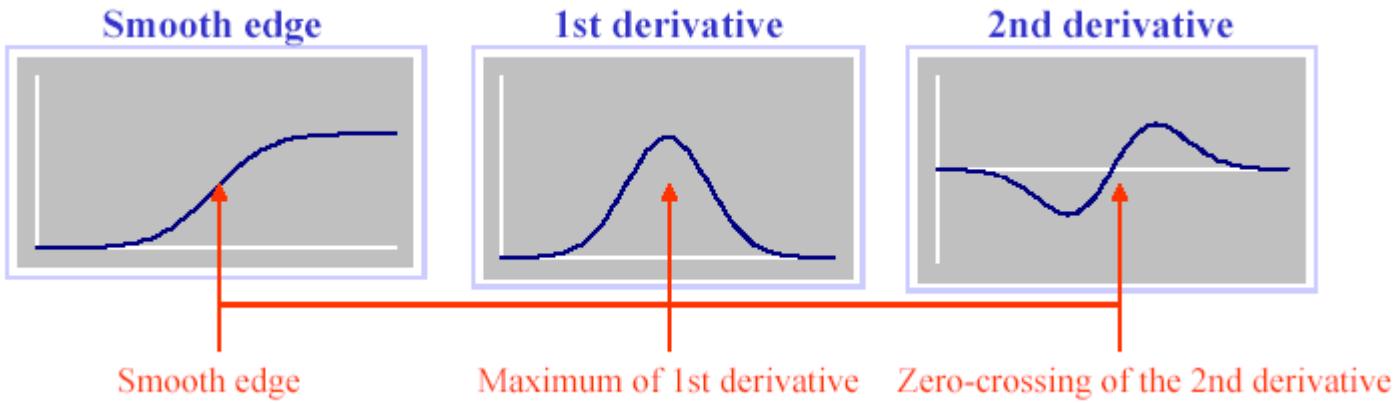


Figure 3: Local edges extracted from the 'good' image of the simple scene.

Figure 5: Lowpass filtering (smoothing) an image with a Gaussian filter.

- Consider a plot of intensity values, taken approximately from the row indicated
  - This is sometimes called a "profile" of image intensity values
  - Notice that now we are considering only one dimension
- We can let x represent the horizontal direction, and $f(x)$ represent the image intensity
- How do you normally detect *change* in some function $f(x)$?




Smooth edge
1st derivative
2nd derivative
Smooth edge
Maximum of 1st derivative
Zero-crossing of the 2nd derivative

The answer: estimate the **derivative** of image intensity

- *Remember:* for a 1-dimensional signal $f(x)$, the derivative is defined as

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

- A discrete approximation of the derivative is therefore

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

- **Now extend this idea to a 2-dimensional image $I(x, y)$, and let Δx be related to the pixel size**

- Discrete approximations to the partial derivatives:

$$\frac{\partial I}{\partial x} \approx \frac{I(x+\Delta x, y) - I(x, y)}{\Delta x}$$

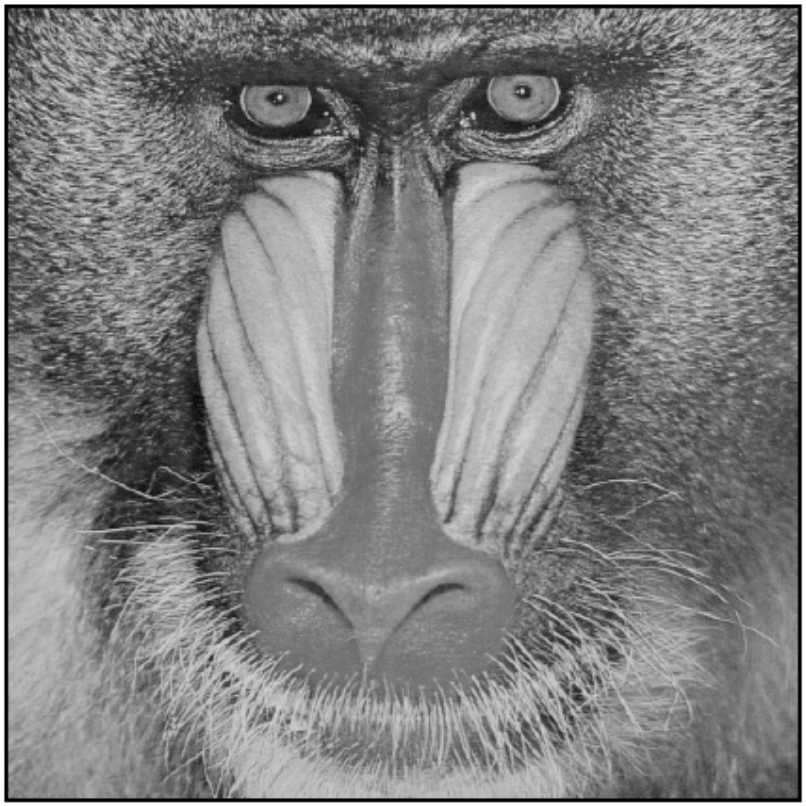
$$\frac{\partial I}{\partial y} \approx \frac{I(x, y+\Delta y) - I(x, y)}{\Delta y}$$

- The first expression above represents a simple edge detector in the horizontal (x) direction
- Now let's consider the discrete image array $I(r, c)$, and define Δx to be the width of one pixel:

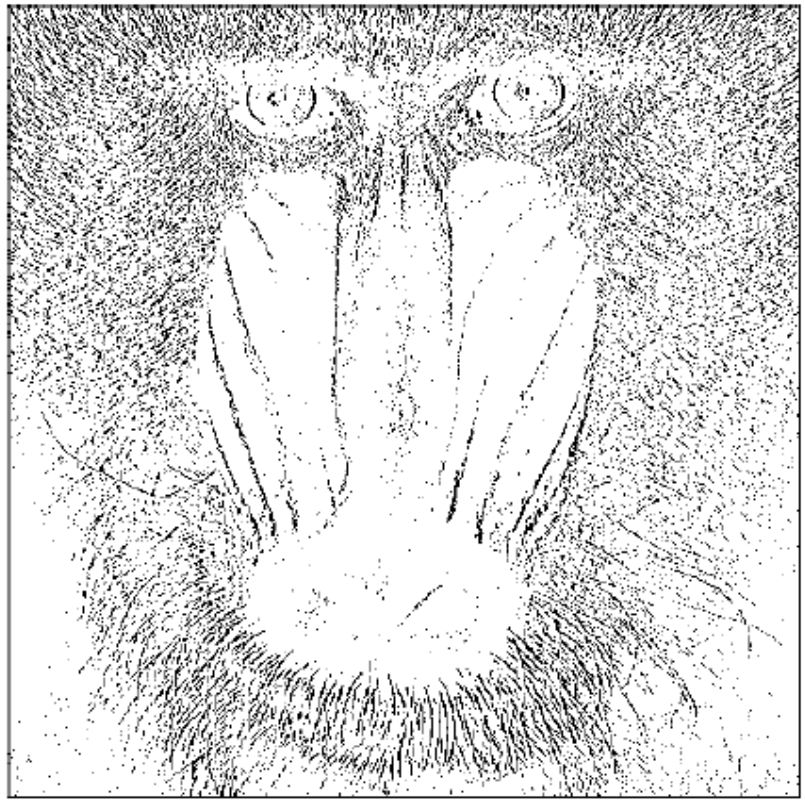
$$\frac{\partial I}{\partial x} \approx \frac{I(x+\Delta x, y) - I(x, y)}{\Delta x} \quad \rightarrow \quad I_x(r,c) = I(r, c+1) - I(r, c)$$

Compute $I_x(r,c) = I(r, c+1) - I(r, c)$
then take the absolute value,
and then threshold ($T = 20$)

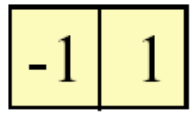
Original image $I(r, c)$ (size 512x512)

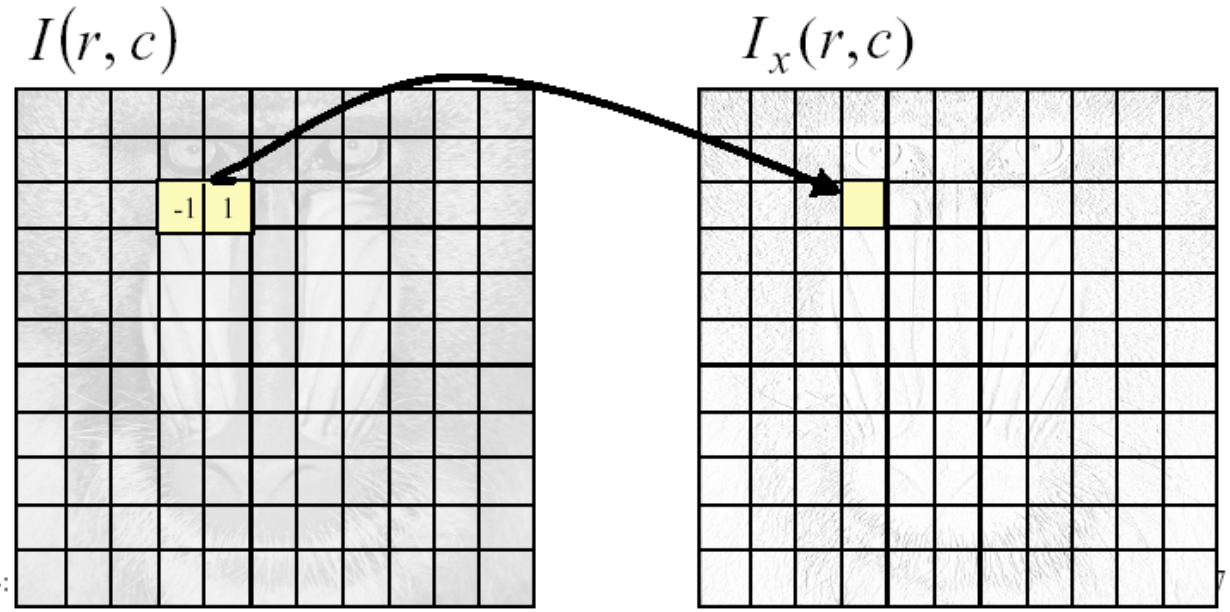


Result

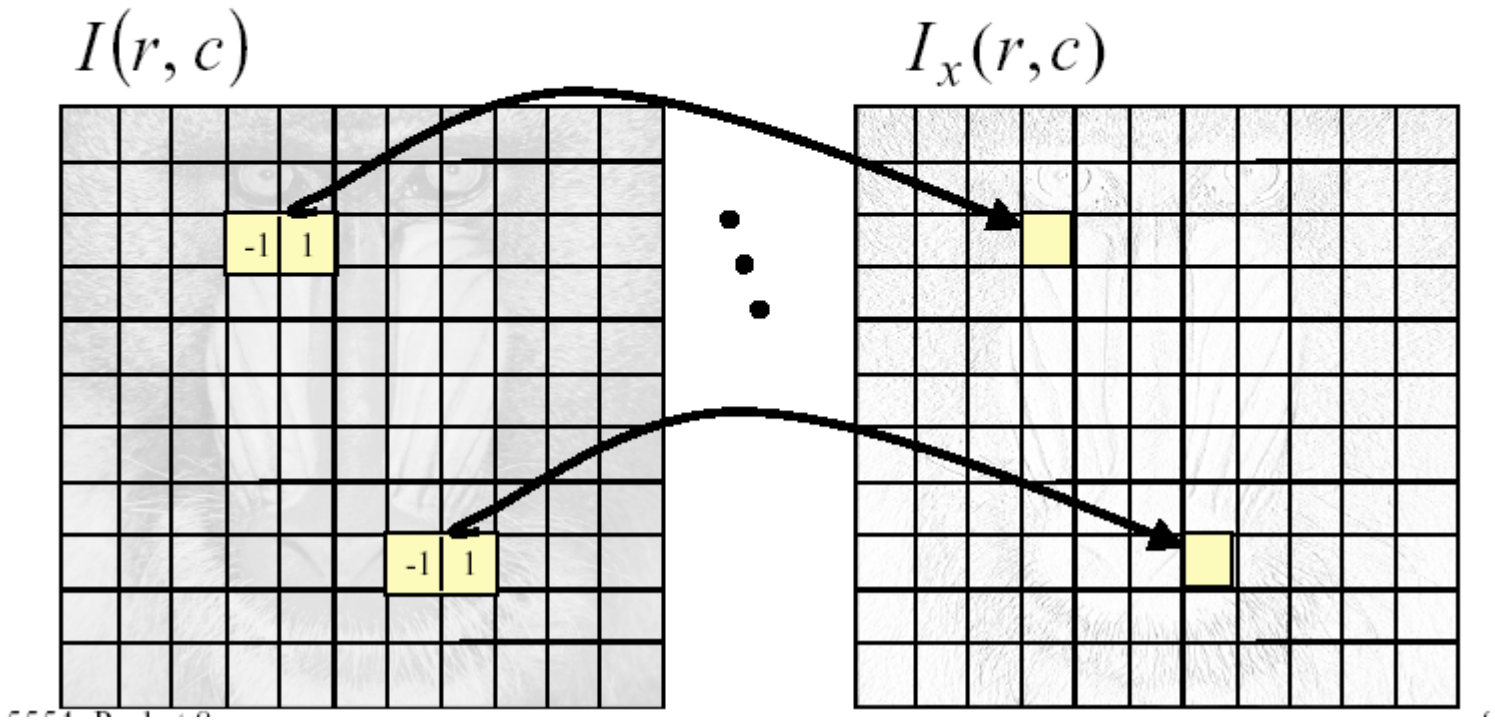


$$I_x(r,c) = I(r, c+1) - I(r, c)$$
$$= I(r, c)\cdot(-1) + I(r, c+1)\cdot(1)$$





- The small grid is called a **template** or **operator** or **mask** or **filter**
- We imagine moving the template over the original image to create the new image



Many other templates are possible for approximating $\frac{\partial I}{\partial x}$

-1	1

$$I_x(r,c) = I(r, c+1) - I(r, c)$$

-1	0	1

$$I_x(r,c) = I(r, c+1) - I(r, c-1)$$

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

$\bullet \frac{1}{3}$

$$\begin{aligned} I_x(r,c) = [\, & I(r-1,\ c+1) - I(r-1,\ c-1) \\ & + I(r,\ c+1) - I(r,\ c-1) \\ & + I(r+1,\ c+1) - I(r+1,\ c-1)\,] \bullet \tfrac{1}{3} \end{aligned}$$

Think of combining 3 separate approximations of $\frac{\partial I}{\partial x}$ into a single estimate

Some pixels can be weighted more heavily than others

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

$\bullet \frac{1}{4}$

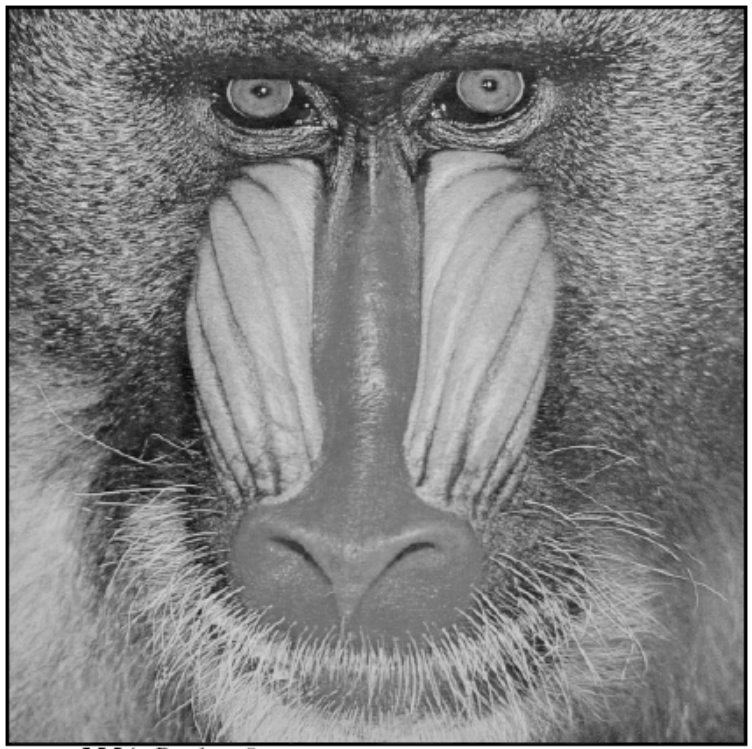
$$I_x(r,c) = [I(r-1,\ c+1) - I(r-1,\ c-1) + 2\ I(r,\ c+1) - 2\ I(r,\ c-1) + I(r+1,\ c+1) - I(r+1,\ c-1)] \bullet \frac{1}{4}$$

This template is very common;
it is called the (horizontal) **Sobel operator**

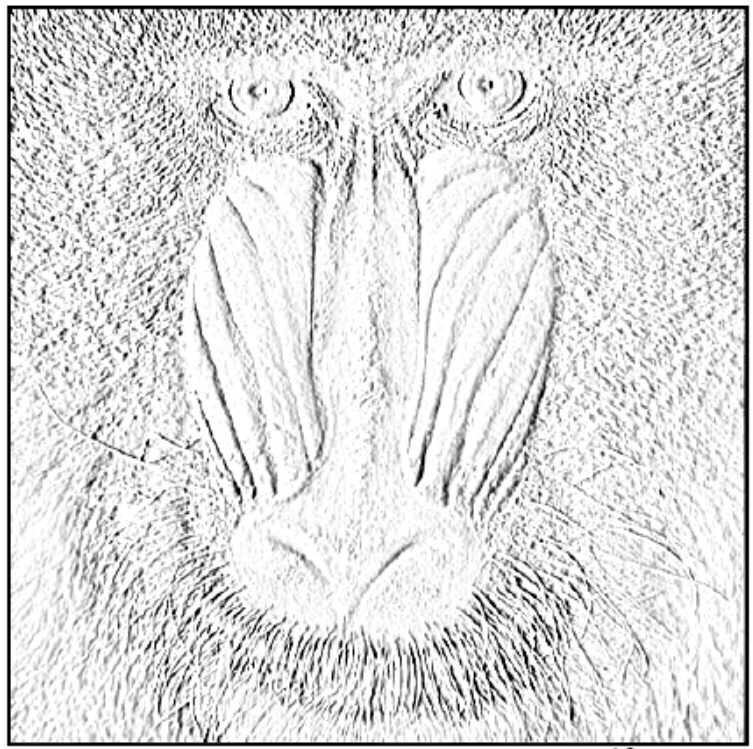
We often drop the fraction at the end

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

Original image $I(r, c)$



Result (without thresholding)



Now consider the vertical direction: $\frac{\partial I}{\partial y}$

1
-1

$$I_y(r,c) = I(r-1,c) - I(r,c)$$

1
0
-1

$$I_y(r,c) = I(r-1,c) - I(r+1,c)$$

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

$\bullet \frac{1}{4}$

$$\begin{aligned} I_y(r,c) = \; & [I(r-1,\, c-1) - I(r+1,\, c-1) \\ & +2\; I(r-1,\, c) - 2\; I(r+1,\, c) \\ & +I(r-1,\, c+1) - I(r+1,\, c+1)] \bullet \tfrac{1}{4} \end{aligned}$$

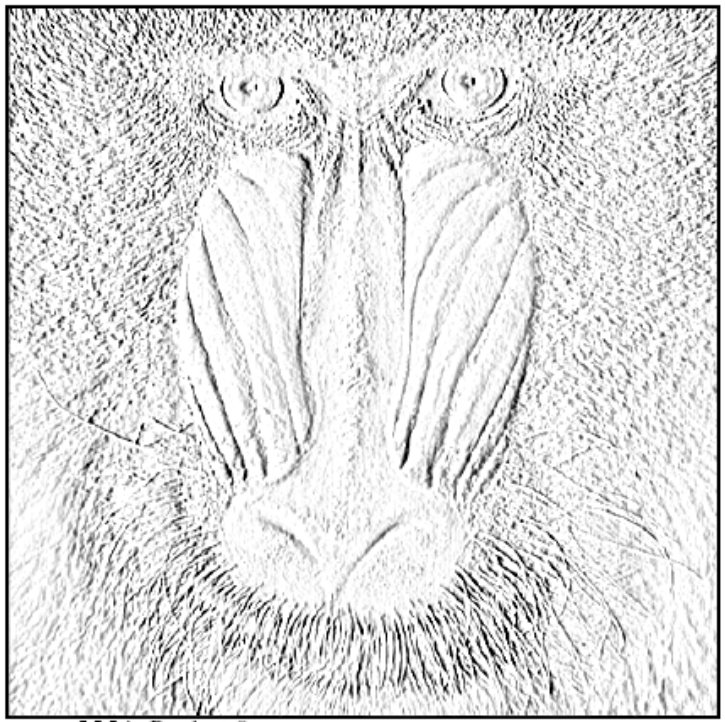
This is called the (vertical) **Sobel operator**

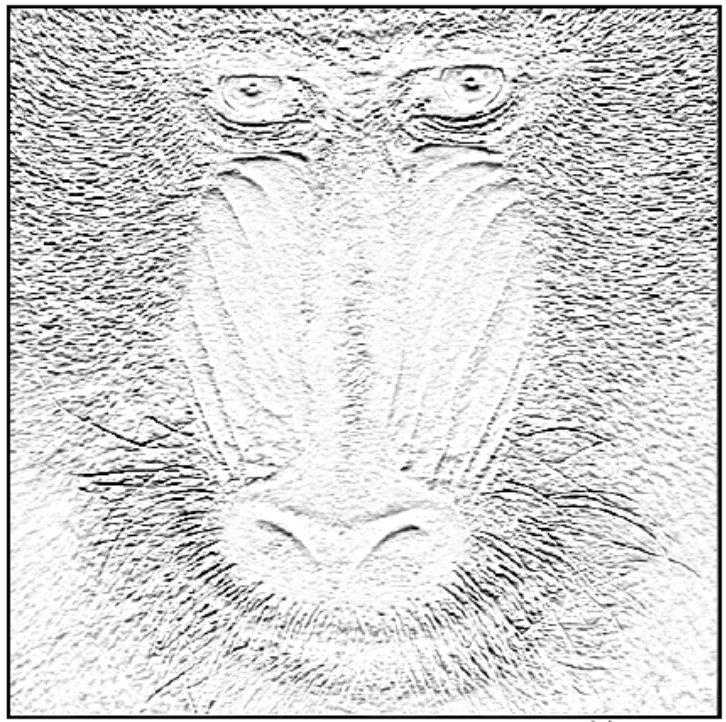
I_x

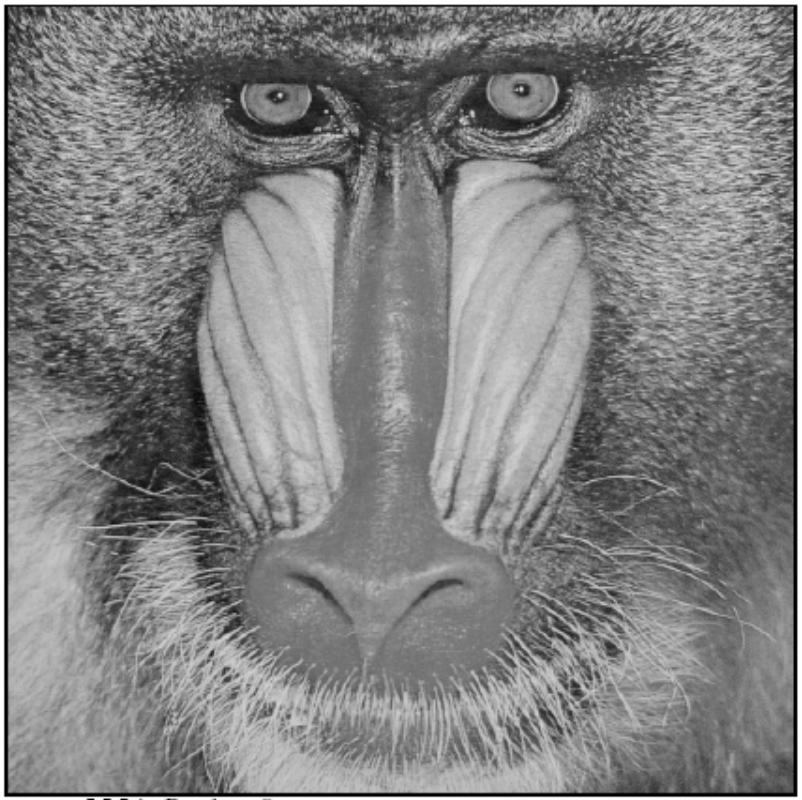
-1	0	1
-2	0	2
-1	0	1

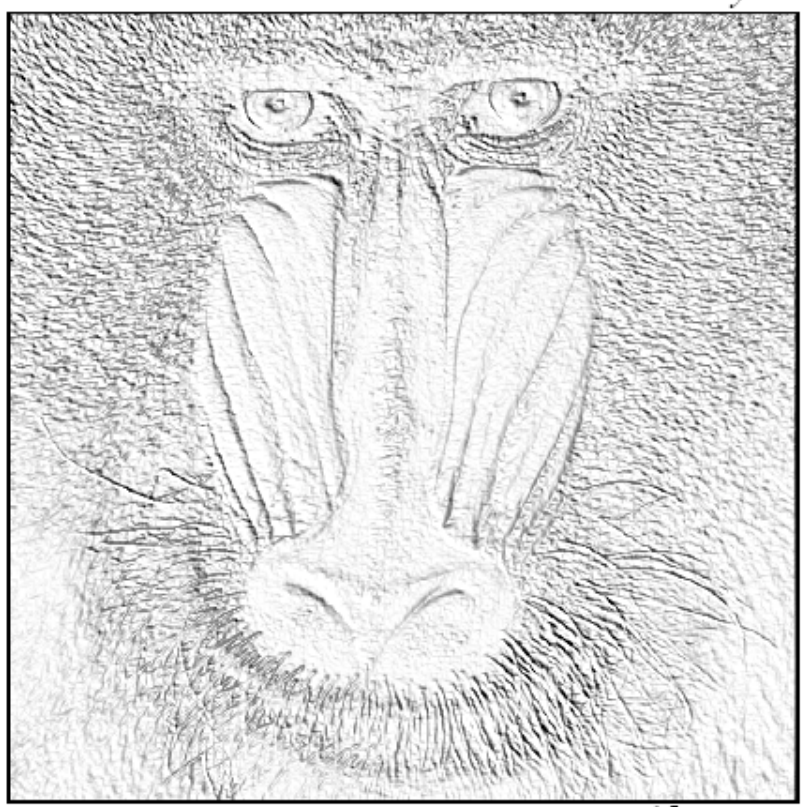
I_y

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






Original image: I


Combine the 2 Sobel results: $|I_x| + |I_y|$

Roberts Operator

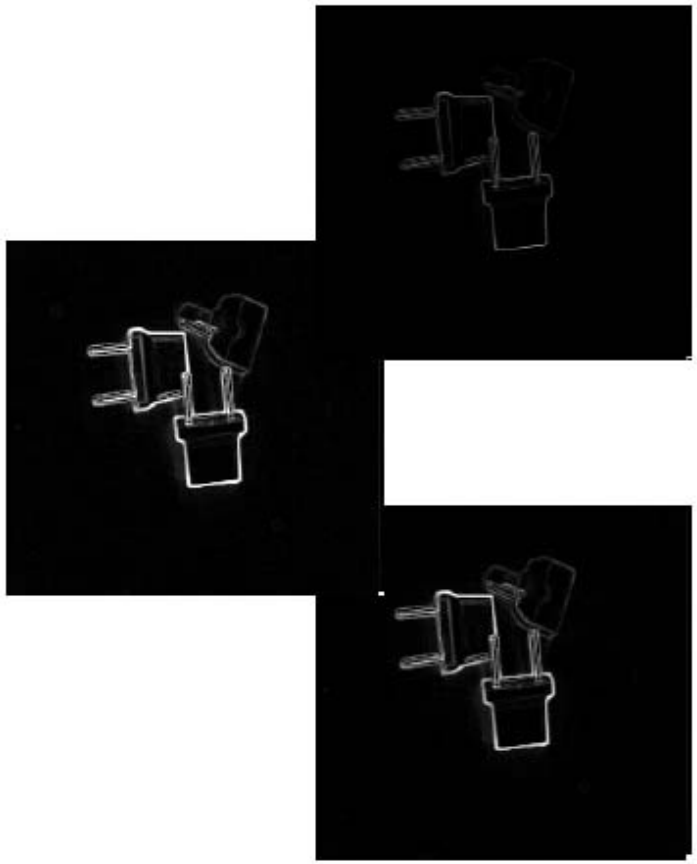
$$G_x \approx \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad G_y \approx \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

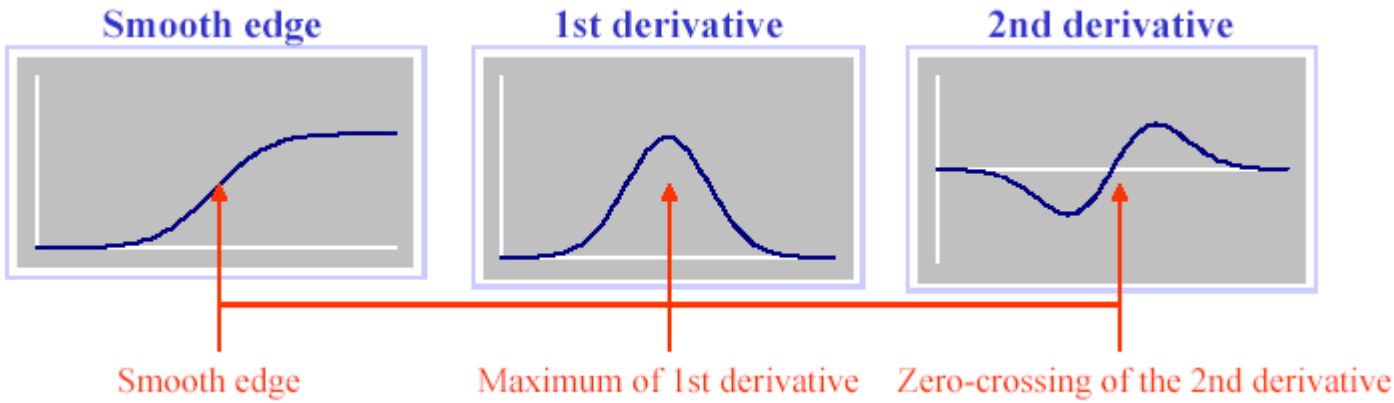
Sobel Operator

$$G_x \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Prewitt Operator

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




Smooth edge
1st derivative
2nd derivative
Smooth edge
Maximum of 1st derivative
Zero-crossing of the 2nd derivative

Metodi basati sulla derivata prima

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$$

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

Metodi basati sulla derivata seconda: Il Laplaciano

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

Finite approximations: 1-D

$$\frac{dF}{dx} \approx F[j+1] - F[j]$$

$$\frac{d^2F}{dx^2} \approx F[j-1] - 2F[j] + F[j+1]$$

Finite approximations: 2-D

j, k	j+1, k
j,k+1	j+1, k+1

$$\frac{\partial I}{\partial \hat{x}} \approx I[j+1,k+1] - I[j,k]$$

$$\frac{\partial I}{\partial \hat{y}} \approx I[j,k+1] - I[j+1,k]$$

$$\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2 \approx (I[j+1,k+1] - I[j,k])^2 + (I[j,k+1] - I[j+1,k])^2$$

```
for i from 0 to imax − 1
  do for j from 0 to jmax − 1
   do M[j, k] = (I[j + 1, k + 1] − I[j, k])^2 + (I[j, k + 1] − I[j + 1, k])^2
```

Approximating the Laplacian

j-1, k+1	j, k+1	j+1, k+1
j-1, k	j,k	j+1, k
j-1, k-1	j, k-1	j+1, k-1

$$\frac{\partial^2 I}{\partial x^2} \approx I[j-1,k] - 2I[j,k] + I[j+1,k]$$

$$\frac{\partial^2 I}{\partial y^2} \approx I[j,k-1] - 2I[j,k] + I[j,k+1]$$

$$\nabla^2 I \approx I[j-1,k] + I[j,k-1] + I[j+1,k] + I[j,k+1] - 4I[j,k]$$

	1	
1	-4	1
	1	

Note that on a rectangular grid its hard to come up with an approximation to ∇^2 that is rotationally symmetric (even though the continuous operator is symmetric). Certainly the computation specified by equation (2) is not rotationally symmetric — it depends critically on the orientation of the axes. For example, if we consider a 45° rotated coordinate system, then we get

1		1
	-4	
1		1

A particularly accurate approximation to the Laplacian is given by a weighted sum of the above two approximations, where the x-y-oriented term is weighted approximately twice as much as the diagonally oriented term. This results in a mask of

1	4	1
4	-20	4
1	4	1

Algoritmo di Marr-Hildreth

The Marr-Hildreth edge detector [5] is based on computing the zero crossings of the Laplacian of the Gaussian smoothed image,

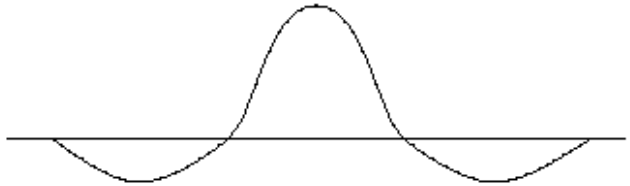
$$\nabla^2(G_\sigma \otimes I) \,.$$

We know from the previous section that both the Gaussian smoothed image and the Laplacian can be implemented by convolution, and are hence linear operators. Thus, by associativity we can equivalently express the computation as,

$$(\nabla^2 G_\sigma) \otimes I \,.$$

The Marr-Hildreth edge detector is thus often referred to as a Laplacian of Gaussian operator, because $\nabla^2 G_\sigma$ (the Laplacian of Gaussian) is convolved with the image, I.

LoG & DoG



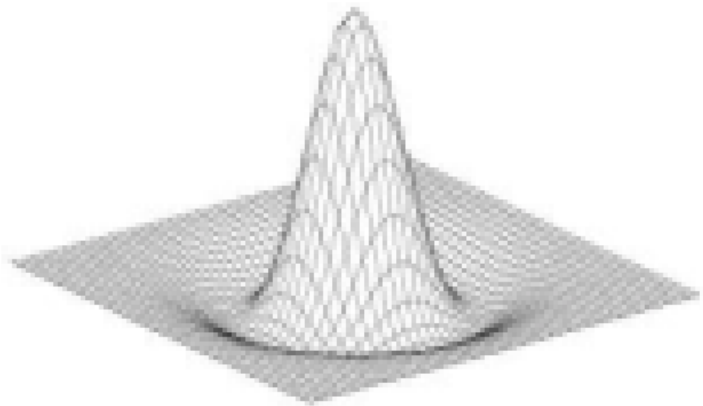


Figure 6: The Laplacian of Gaussian.

The Laplacian of Gaussian can be approximated by a difference of two Gaussians. In the one-dimensional case this is

$$DOG(\sigma_e, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_e} e^{\frac{-x^2}{2\sigma_e^2}} - \frac{1}{\sqrt{2\pi}\sigma_i} e^{\frac{-x^2}{2\sigma_i^2}},$$

which approximates a second derivative operation where $\sigma_e < \sigma_i$ (generally $\sigma_i/\sigma_e \approx 1.6$).

1. Smooth the image by convolution with G_σ (use the one-dimensional decomposition from Section 3).

2. Apply the Laplacian to the result of the previous step (using convolution with the mask from Section 2).

3. Identify edge pixels at the boundaries of regions of constant sign in the result of the previous step.

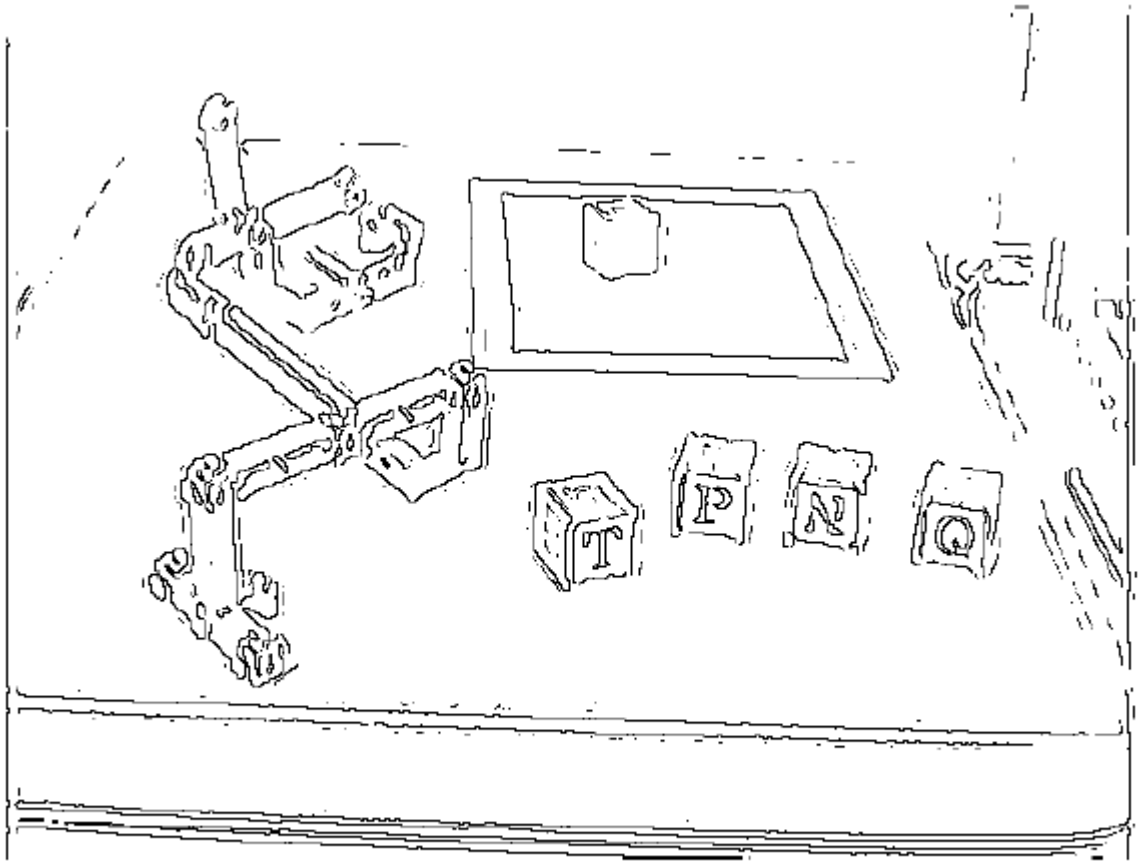
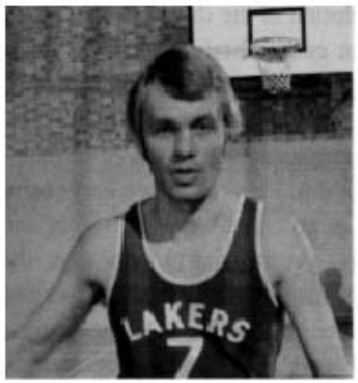
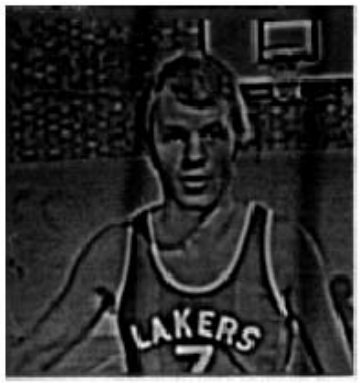


Figure 7: Laplacian of Gaussian edges for an image.



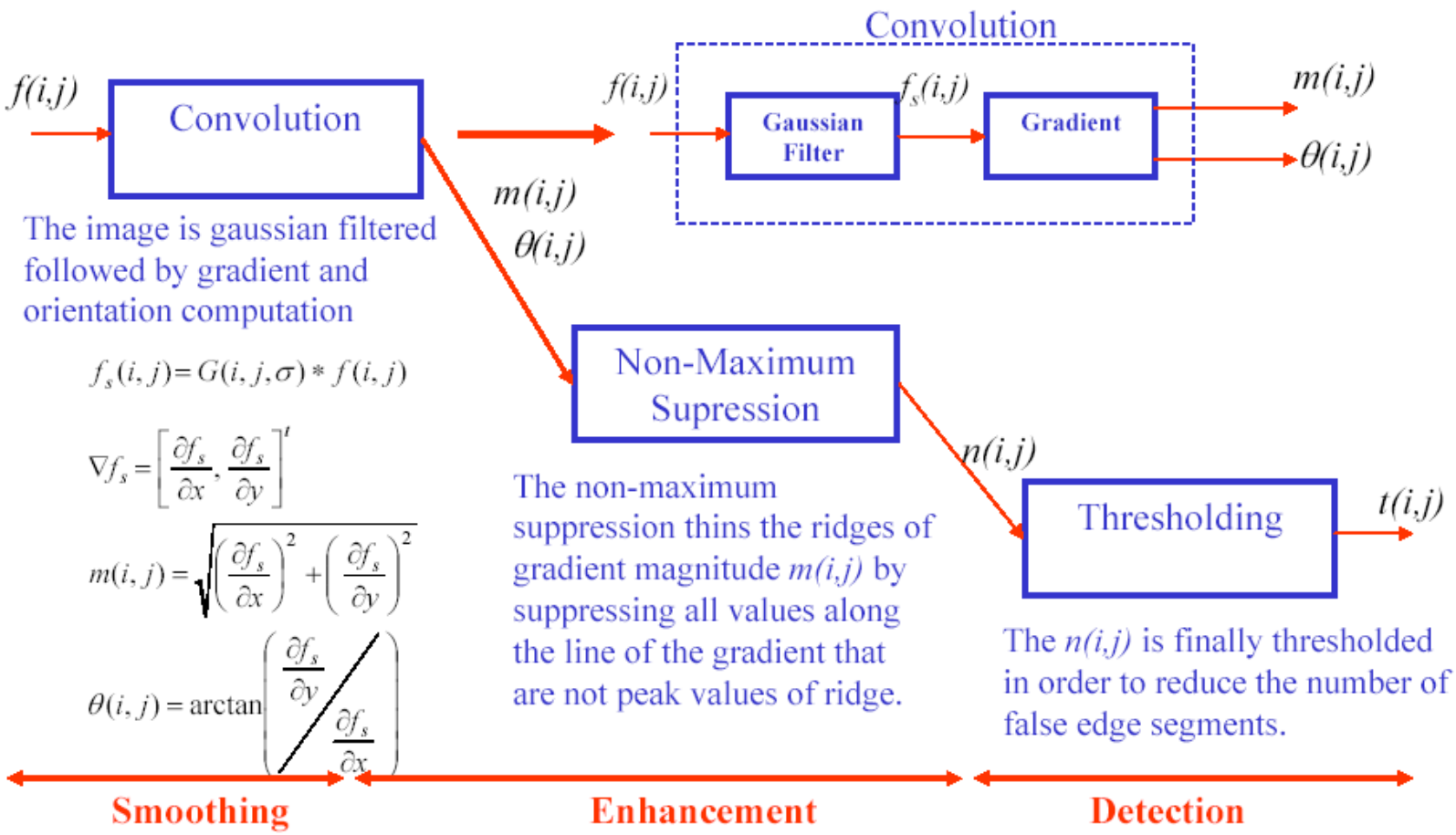
Original Image

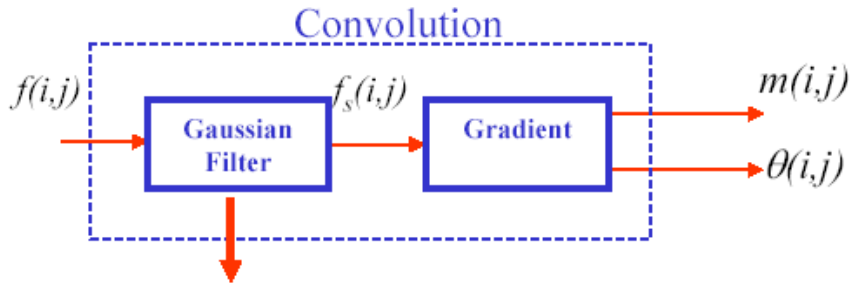


LoG Filter

Zero Crossings

Scale (σ)


Convolution
f(i,j)
Convolution
f(i,j)
Gaussian Filter
$f_s(i,j)$
Gradient
m(i,j)
θ(i,j)
m(i,j)
θ(i,j)
The image is gaussian filtered followed by gradient and orientation computation
$f_s(i,j) = G(i,j,\sigma) * f(i,j)$
$\nabla f_s = \left[\frac{\partial f_s}{\partial x}, \frac{\partial f_s}{\partial y}\right]^t$
$m(i,j) = \sqrt{\left(\frac{\partial f_s}{\partial x}\right)^2 + \left(\frac{\partial f_s}{\partial y}\right)^2}$
$\theta(i,j) = \arctan\left(\frac{\partial f_s}{\partial y} \Big/ \frac{\partial f_s}{\partial x}\right)$
Non-Maximum Supression
The non-maximum suppression thins the ridges of gradient magnitude m(i,j) by suppressing all values along the line of the gradient that are not peak values of ridge.
n(i,j)
Thresholding
t(i,j)
The n(i,j) is finally thresholded in order to reduce the number of false edge segments.
Smoothing
Enhancement
Detection



$$f_s(i,j) = G(i,j,\sigma) * f(i,j) = \left[G_h(i,\sigma) * f(i,j)\right] * G_v(j,\sigma) = \left(\left[a_1\, a_2 \cdots a_n\right] * f(i,j)\right) * \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

$\sigma = 1$

$\sigma = 2$

$\sigma = 3$

Kernels (h1)

[4 60 272 450 27 2 60 4]

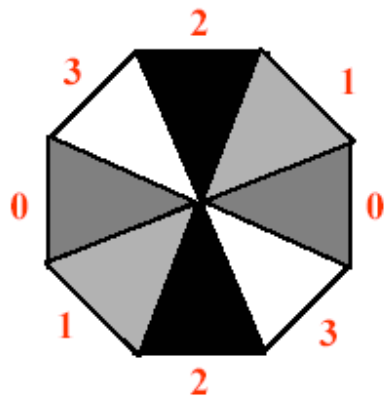
[4 19 60 146 272 397 450
397 272 146 60 19 4]

[1 4 12 29 60 112 185 272 360 425 450
425 360 272 185 112 60 29 12 4 1]

L'algoritmo di Canny

Step 1: magnitude and orientation computation $m(i,j) = \sqrt{\left(\frac{\partial f_s}{\partial x}\right)^2 + \left(\frac{\partial f_s}{\partial y}\right)^2}$ $\theta(i,j) = \arctan\left(\frac{\partial f_s}{\partial y} \Big/ \frac{\partial f_s}{\partial x}\right)$

Step 2: Partition of angle orientations $\vartheta(i,j) = sector(\theta(i,j))$



Step 3: At each pixel DO:

n(i,j) = m(i,j);

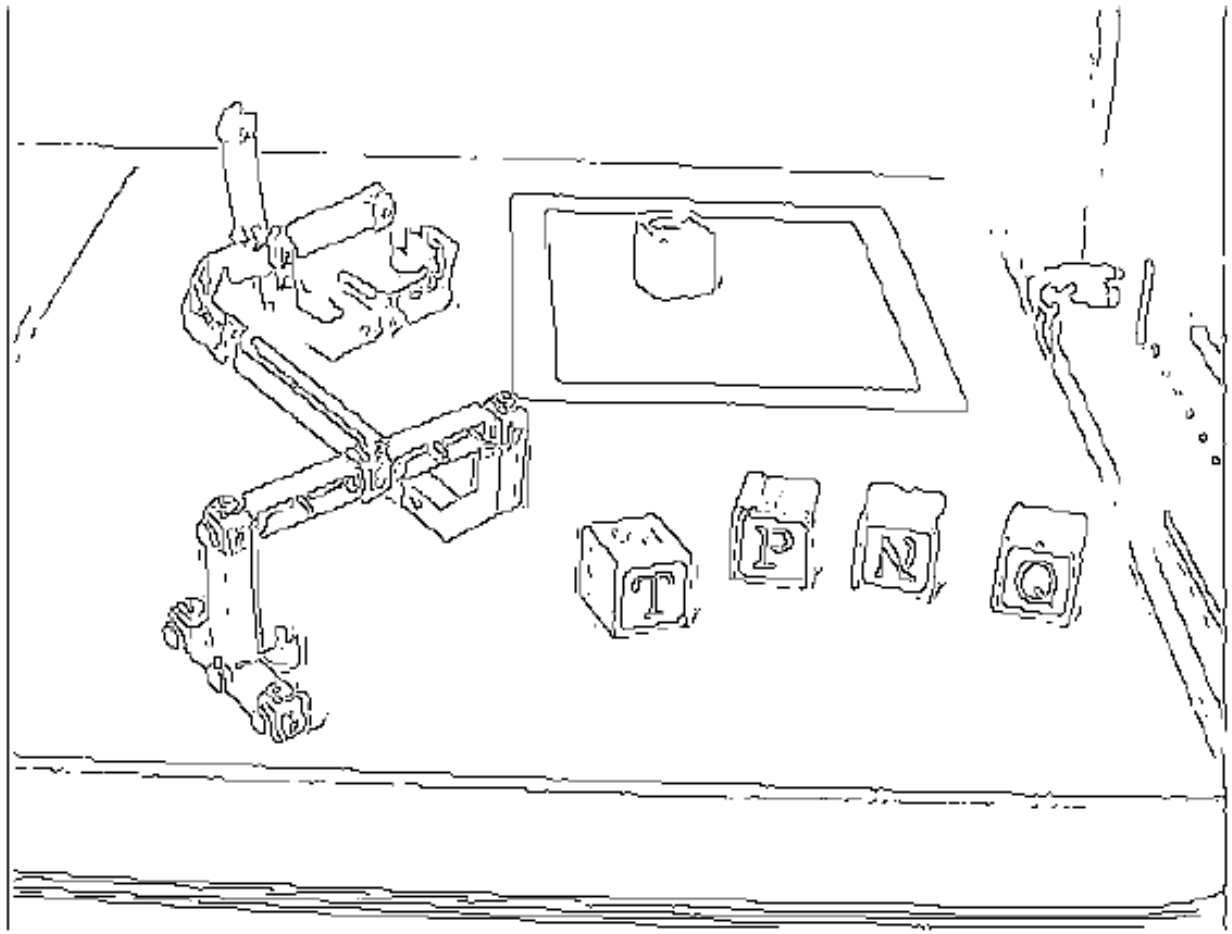
IF *m(i,j)* ≤ the neighbors along the gradient sector

THEN n(i,j)=0;

Step 4: Double Thresholding:

Create two thresholded images $t_1(i,j)$ and $t_2(i,j)$, *using two thresholds* T_1 *and* T_2, *with* $T_1 \approx 0{,}4\ T_2$.

This double threshold method allow to add weaker edges (those above T_1) if they are neighbors of stronger edges (those above T_2). So the threshold image is formed by $t_2(i,j)$ including some of the edges in $t_1(i,j)$


T
P
N
Q

Results