2. Image processing

1. Internal image representation

1) Bitmap data

Monochromatic or color image:

- object of some template class Matrix<T> (type T: one field for intensity image, 3 fields for color image)
- number of intensity (or color component) levels (e.g. 256)

A depth-map - represents the depth of scene elements:

- object of some class Matrix<T> (pixel type T : distance value)
- number of distance levels,

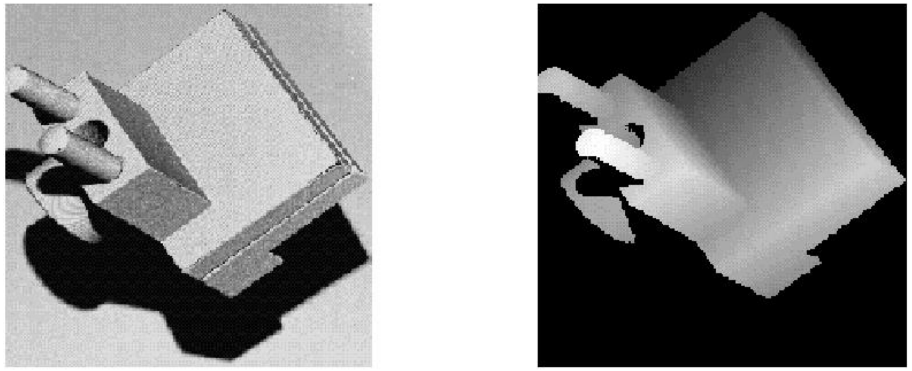


Fig. 1. Intensity image (left image) and corresponding depth image (right).

2) OpenCV (C++)

The OpenCV C++ reference manual is here:
http://docs.opencv.org

Key OpenCV Classes

Point_	Template 2D point class
Point3_	Template 3D point class
Size_	Template size (width, height) class
Vec	Template short vector class
Matx	Template small matrix class
Scalar	4-elementary vector
Rect	Rectangle
Range	Integer value range

Mat	2D or multi-dimensional dense array (can be used to store matrices, images, histograms, feature descriptors, voxel volumes etc.)
SparseMat	Multi-dimensional sparse array
Ptr	Template smart pointer class

Matrix Basics

- Create a matrix

```
Mat image(240, 320, CV_8UC3);
```

- [Re]allocate a pre-declared matrix

```
image.create(480, 640, CV_8UC3);
```

- Create a matrix initialized with a constant

```
Mat A33(3, 3, CV_32F, Scalar(5));
Mat B33(3, 3, CV_32F); B33 = Scalar(5);
Mat C33 = Mat::ones(3, 3, CV_32F)*5.;
Mat D33 = Mat::zeros(3, 3, CV_32F) + 5.;
```

- <u>Create a matrix initialized with specified values</u>

```
double a = CV_PI/3;
Mat A22 = (Mat_<float>(2, 2) << cos(a), -sin(a), sin(a), cos(a));
float B22data[] = {cos(a), -sin(a), sin(a), cos(a)};
Mat B22 = Mat(2, 2, CV_32F, B22data).clone();
```

- <u>Initialize a random matrix</u>

```
randu(image, Scalar(0), Scalar(256)); // Uniform distribution
randn(image, Scalar(128), Scalar(10)); // Gaussian distribution
```

- Convert matrix to/from other structures

(without copying the data)

```
Mat image_alias = image;
float* Idata=new float[480*640*3];
Mat I(480, 640, CV_32FC3, Idata);
vector<Point> iptvec(10);
Mat Ip(iptvec); // Ip: a 10 × 1 CV_32SC2 matrix
```

... (with copying the data)

```
Mat newI = I.clone();
```

Note: Mat in itself is a smart pointer to its underlying array buffer – multiple Mat objects can reference single buffer. The buffer is destroyed when the last associated Mat object is dereferenced

- <u>Access matrix elements</u>

Use at operator:

```
A33.at<float>(i,j) = A33.at<float>(j,i)+1;
```

Or directly access underlying buffer (faster):

```
Mat dyImage(image.size(), image.type());
for(int y = 1; y < image.rows-1; y++) {
    Vec3b* prevRow = image.ptr<Vec3b>(y-1);
    Vec3b* nextRow = image.ptr<Vec3b>(y+1);
    for(int x = 0; x < image.cols; x++)
        for(int c = 0; c < 3; c++)
            dyImage.at<Vec3b>(y,x)[c] =
                saturate_cast<uchar>(nextRow[x][c] - prevRow[x][c]);
}
Mat_<Vec3b>::iterator it = image.begin<Vec3b>(),
itEnd = image.end<Vec3b>();
for(; it != itEnd; ++it)        (*it)[1] ^= 255;
```

Reference assignment vs. memory allocation

- reference assignment only

```
Mat A, C; // reference variables only
A = imread(argv[1], CV_LOAD_IMAGE_COLOR);
Mat B(A);  // use the copy constructor
C = A; // assignment operator;
```

- new image allocation

```
Mat F = A.clone();
Mat G;
A.copyTo(G);
```

Region of interest

```
Mat D(A, Rect(10, 10, 100, 100) ); // using a rectangle
Mat E = A(Range:all(), Range(1,3)); // using row and column boundaries
```

Image filtering

Filtering

Smooth (also called *blur*) the image with one of the linear or non-linear filters

- boxFilter() - smoothing by a mean filter
- GaussianBlur() – smoothing by a Gaussian filter
- medianBlur() – smoothing by the median filter
- bilateralFilter() - edge preserving smoothing with weight modification,

Compute the spatial image derivatives

- Sobel() - calculate the derivatives from an image
- Scharr() - calculate a more accurate derivative for a kernel of size 3×3

Compute the Laplacian: $I = \partial^2 I / \partial x^2 + \partial^2 I / \partial y^2$

- Laplacian()

Morphological operations

erode() – perform the Erosion operation
dilate() – perform the Dilation operation.

General filter function: filter2D()

Example.

Filter image in-place with a 3x3 high-pass kernel (preserve negative responses by shifting the result by 128):

```
filter2D(image, image, image.depth(),
        (Mat_<float>(3,3) << -1, -1, -1, -1, 9, -1, -1, -1, -1),
        Point(1,1), 128);
```

3) Typical image types

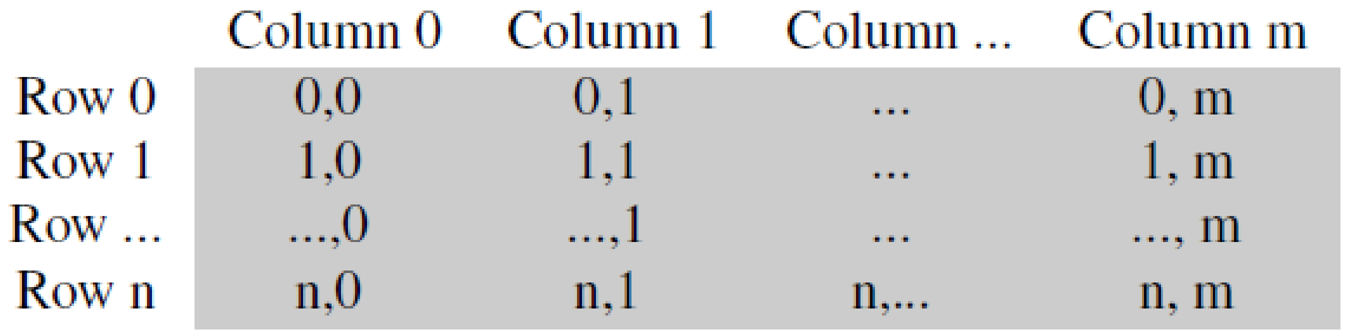
Binary images

- Only two pixel values (e.g. corresponding to foreground objects and the background).
- A binarization threshold is provided.

This image requires a simple analysis (sufficient in many applications).

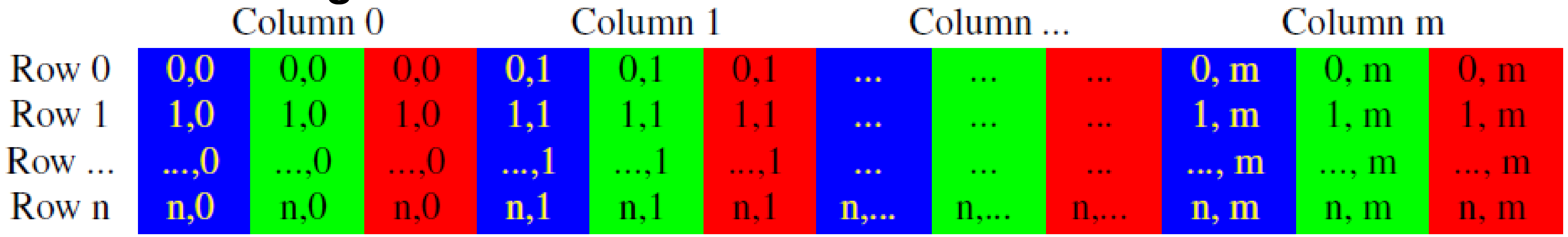
Internal storage of bitmaps in OpenCV

Gray scale image



	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

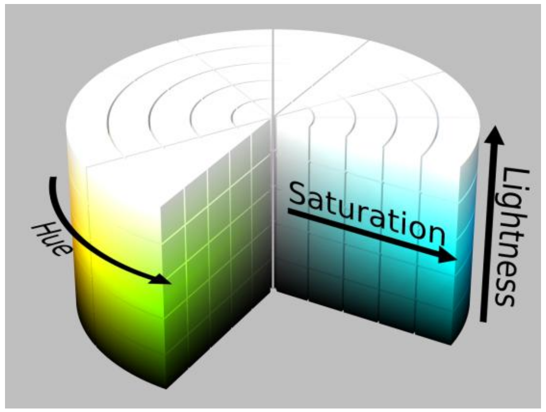
RGB color image

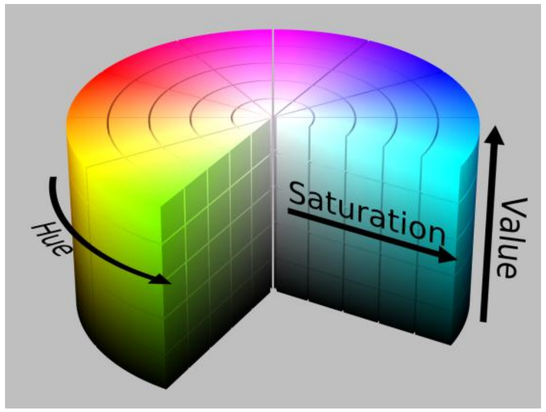


	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

Other color spaces

- **HSL** (Hue, Saturation, Lightness) or **HSV** (Hue, Saturation, Value) - based on psychological observations on human **perception**.
- In the cylinder, the angle around the central vertical axis corresponds to "hue", the distance from the axis corresponds to "saturation", and the distance along the axis corresponds to "value" (or "brightness").





Źródło: https://en.wikipedia.org/wiki/HSL_and_HSV

The transformation RGB ←→ HSV is a nonlinear one.
Technical systems represent color as a mixture of three basic vectors in the color space:

- A typical color system for computer monitors - contributions of three basic colors: red, green, blue (the **RGB** image) - the additive positive-oriented system - "white" is the sum of its components.

- Typical color printings on printers - the **CMY** (Cyan, Magenta, Yellow) system - an additive "negative" model ("black" is the sum of components) :

  $$[C, M, Y] = [1, 1, 1] - [R, G, B].$$

  and CMYK (Cyan, Magenta, Yellow, Black).

- Typical television broadcast - **YIQ** (Y - luminance, I, Q - chromatic components - differences from red and red-blue) and **YUV**.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1471 & -0.2889 & 0.436 \\ 0.6149 & -0.515 & -0.100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- In digital media - **Y Cb Cr**
  where Cb, Cr - distances of the color from "gray" along the blue and red axis.

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Note: The R, G, B values are from [0, 1]. Y also takes values from [0, 1], but C_b, C_r (or U, V) take values from [-0.5, 0.5]. Luminance is encoded with greater bandwidth – since human eye is more sensitive to it

The inverse transformations:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.14 \\ 1.0 & -0.3947 & -0.5806 \\ 1.0 & 2.032 & 0.0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.34414 & -0.71414 \\ 1.0 & 1.772 & 0.0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix}$$

4) Color pixel resolution

In external image representation (compressed form), the color components are usually of lower resolution than the intensity:

- RGB is transformed to the Y Cb Cr (or YUV) - space
- Chromatic components Cb, Cr (or U, V) are stored for several pixels (2-4):

Number of bytes per 4-pixel block			
RGB 4:4:4	4:2:2 (Rec. 601)	4:2:0 (JPEG, H.261, MPEG-1)	4:2:0 (MPEG-2)
R_0 R_1 R_2 R_3	Y_0 Y_1 Y_2 Y	Y_0 Y_1 Y_2 Y	Y_0 Y_1 Y_2 Y_3
G_0 G_1 G_2 G_3	C_{B0} C_{B2}	C_B	C_B
B_0 B_1 B_2 B_3	C_{R0} C_{R2}	C_R	C_R

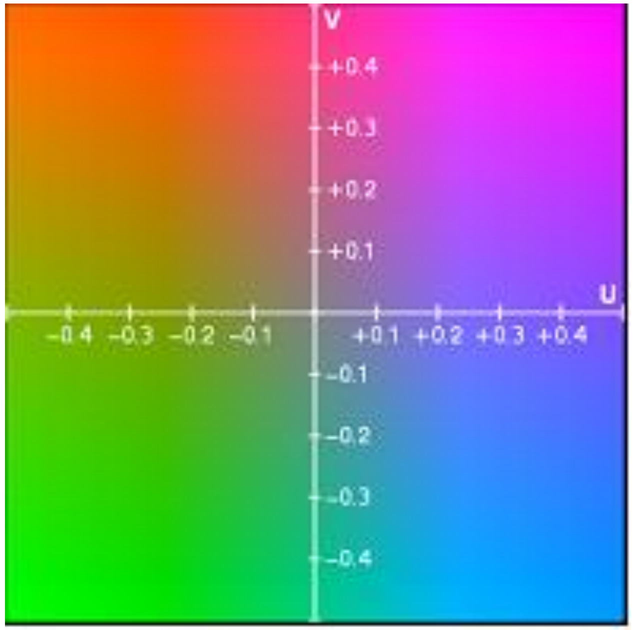
5) Color calibration

Ideal colours (blue, green, red, yellow, magenta, white, dark), specific "skin" colours ("dark skin" and "light skin") , etc. are defined in the YUV space. They are applied to make a colour calibration of the image.

```
const unsigned char kIdealMacbethYUV[4][6][3] = {
        {
                {88, 114, 146},         // Dark Skin
                {161, 110, 151}, // Light Skin
...
        },
        {
                {140, 69, 182}, // Orange,
        {
                {66, 176, 113}, // Blue
             {117, 100, 95}, // Green
             {85, 111, 193}, // Red
             {191, 35, 161}, // Yellow
             {120, 143, 175}, // Magenta
             {98, 166, 57} // Cyan
      },

         {
                {241, 128, 128}, // White
                {201, 128, 128}, // Neutral 0.8
                {161, 128, 128}, // Neutral 0.65
                {121, 128, 128}, // Neutral 0.5
                {83, 128, 128}, // Neutral 0.35
                {49, 128, 128} // Black
         }
};
```

Fig. 2 The UV subspace for center-normalized Y (Y=0.5 for continuous variable, or 128 in digital form)



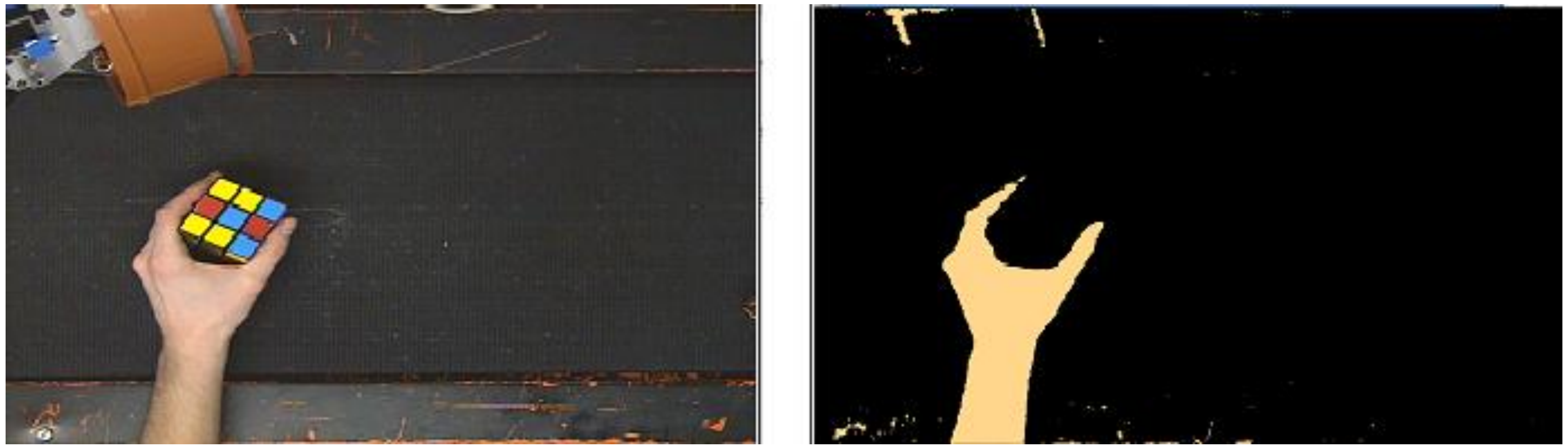


Fig. 3 Illustration of skin colour detection in a colour image.

2. Histogram equalization

Images with a low contrast can be enhanced using the technique called „**histogram equalization**”. The „equalization” of histogram *H* means such a transformation of pixel values in the original image, that the histogram of transformed image is more “flat” that the original one. In other words the purpose of equalization is to make every pixel value (nearly) equally probable.

The histogram equalization transformation is defined as:

$$z = f(w) = round[D_m \frac{1}{PixNum} \sum_{i=0}^{w} H(i)] \qquad (1)$$

where H is the original histogram, $H(i)$ is the histogram value for pixel value i, $PixNum$ – total pixel number, $[0, 1, ..., D_m]$ is the interval of digital levels of variable w in the original image, $round[]$ – specifies the rounding operation to nearest digital level.

Remark: perfectly flat histogram results are **seldom** obtained when working with **digital** variables. Equalization usually leads to a reduction of pixel value levels in the transformed image if compared to the original one.

Explanation of equation (1)

The cumulative probability distribution of w is :

$$F(w) = \frac{1}{PixNum} \sum_{i=0}^{w} H(i) \tag{2}$$

$0 \le F(w) \le 1$, but $0 \le z \le D_m$.

Hence, equation (1) can be rewritten as:

$$z = f(w) = round[D_m F(w)] \tag{3}$$

For a **continuous** variable w using a transformation function equal to the cumulative distribution of w produces a uniform density.

Let us assume, that w is a continuous variable and $w \in [0, 1]$. Let the function T ($z = T(w)$) produces a value a for every value of w and $z \in [0, 1]$. We want the density of original and transformed variable be the same in a small int.:

$$p_z(z) = p_w(w)\frac{dw}{dz} \qquad (4)$$

Assume, the function T is equal to the cumulative distribution of w, i.e.

$$z = T(w) = \int_o^w p_w(t)dt \text{ , } 0 \le w \le 1 \qquad (5)$$

Hence, the derivative of z with respect to w in this case is:

$$\frac{dz}{dw} = p_w(w) \qquad (6)$$

Substituting (1.6) into eq. (1.4) yields:

$$p_z(z) = \left[p_w(w)\frac{1}{p_w(w)} \right] = 1 \text{ , } 0 \le z \le 1 \qquad (7)$$

Hence, using a transformation function equal to the cumulative distribution of w produces an uniform density of the output variable - all equal to 1.

Remark

For a digital variable z only D_m equally spaced levels are allowed. Each of the transformed values according to eq. (1.1) is assigned to its closest level.

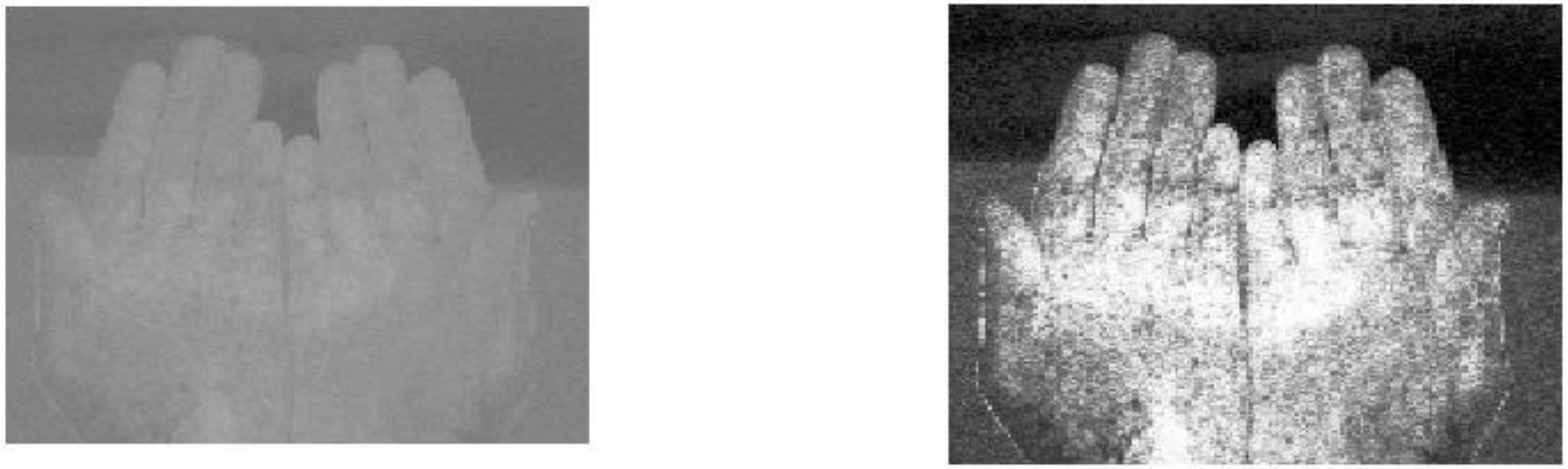


Fig. 4. Example of results of histogram equalization: (a) input image, (b) output image after histogram equalization.

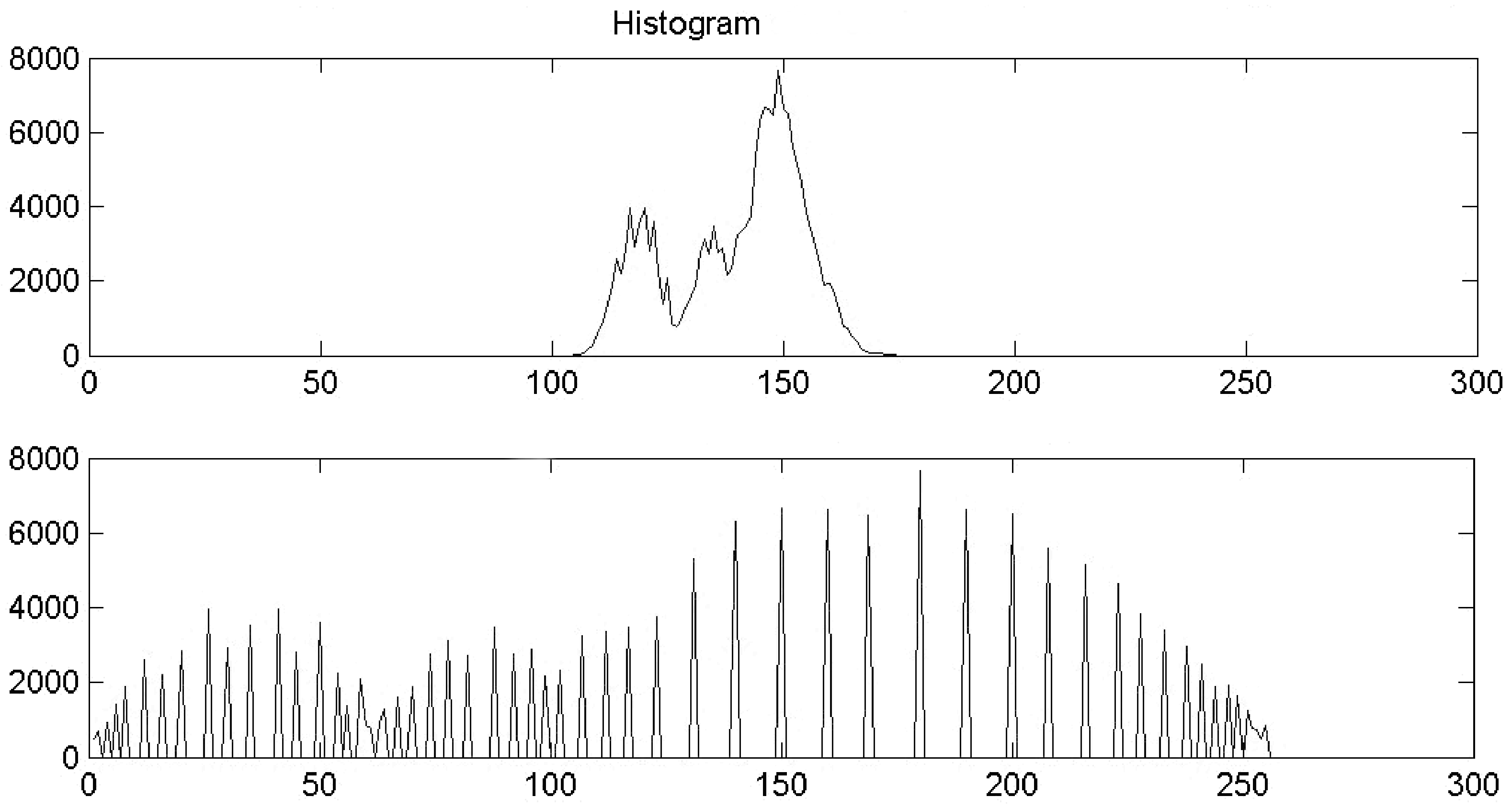


Fig. 4 (cont.) Original histogram (top), histogram after histogram equalization-based image transformation (bottom).

3. Image thresholding (binary image)

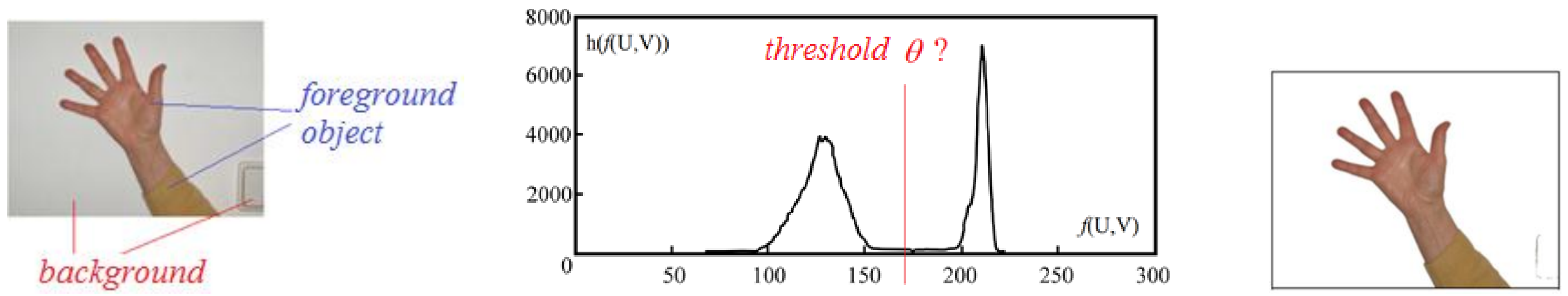


Fig. 5 The idea of image binarization: (a) foreground object and background, (b) the pixel value distribution (histogram), c) extracted foreground.

Problem: how properly to set the threshold for image binarization?

Solution idea: the image histogram is approximated by a sum of two normal (Gaussian) distributions. The crossing point of both distribution curves determines the threshold point.

A pdf mixture with scaling coefficient α :

$$p(l) = \alpha N(l, m_1, \sigma_1) + (1-\alpha) N(l, m_2, \sigma_2) \quad (8)$$

$$p(l) = \frac{\alpha}{\sigma_1\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{l-m_1}{\sigma_1}\right)^2\right] + \frac{1-\alpha}{\sigma_2\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{l-m_2}{\sigma_2}\right)^2\right]$$

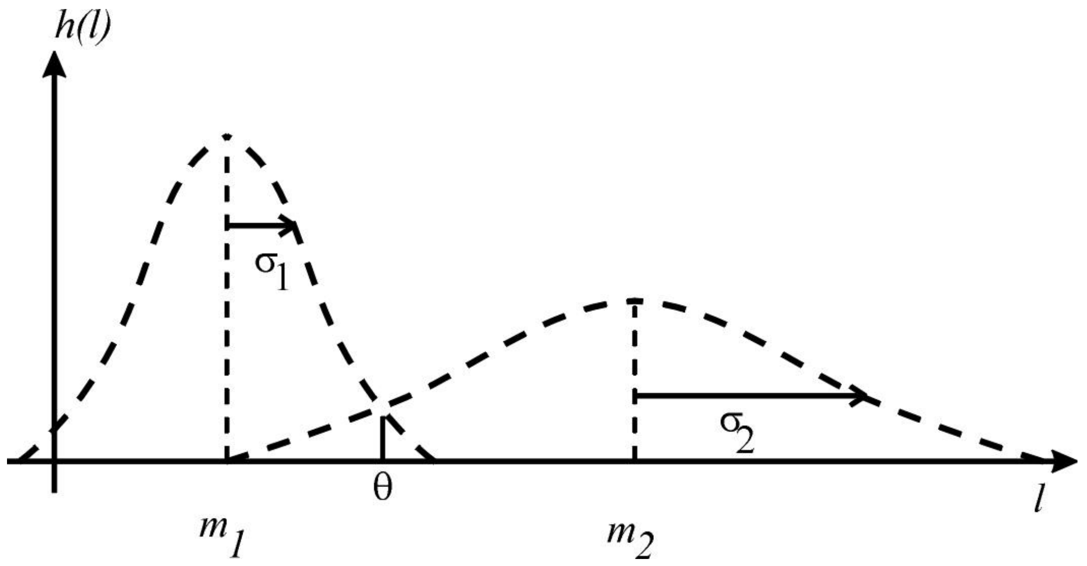


Fig. 6. Image histogram approximation by a mixture of 2 Gaussian distributions.

Task: to estimate the parameters α, m_1, m_2, σ_1, σ_2 of the mixture distribution that approximates the histogram in a best way and next – to compute the crossing point of the two component Gaussian distributions.

Full solution: Expectation-Maximization algorithm

The method of Otsu (1979)

This is a further simplification of the mixed Gaussian distribution search. Only first-order moments need to be computed. A threshold is found that guarantees the largest possible between-classes variance:

1. FOR all possible thresholds θ;
   obtain the between-class variance as:

$$\sigma(\theta)_B^2 = \sigma(\theta)^2 - \sigma(\theta)_W^2 = P_1(\theta)(1 - P_1(\theta))(m_1(\theta) - m_2(\theta))^2 \, , \tag{9}$$

where $P_1(\theta) = \dfrac{\sum_{l=1}^{\theta} h(l)}{\sum_{l=1}^{L} h(l)}$.

2. Select θ such that $\sigma(\theta)_B{}^2$ is of maximum value.
3. Computation can be performed recursively for subsequent threshold values for efficiency

4. Image filtering

1) Convolution

Image filters are mainly used to suppress either
- the high frequencies in the image, *i.e.* smoothing the image, or
- the low frequencies, *i.e.* enhancing or detecting edges in the image.

Spatial filter

The filtering process in the spatial domain means to convolve the input image $f(i,j)$ with some filter function $h(i,j)$ (called kernel):

$$g(i,j) = h(i,j) * f(i,j) \tag{10}$$

The **discrete convolution** in time (spatial) domain is a `*shift and multiply*' operation, where we shift the kernel over the image and multiply its value with the corresponding pixel values of the image.

For a square kernel with size $(M+1)\times(M+1)$ the discrete convolution is:

$$g(i,j) = \sum_{m=-M/2}^{M/2} \sum_{n=-M/2}^{M/2} h(m,n) \cdot f(i-m, j-n) \tag{11}$$

Main image filters:

1. **Mean filter**- noise reduction (NR) using mean of neighborhood
2. **Median filter**- NR using median of neighborhood
3. **Gaussian smoothing** - NR with a Gaussian smoothing kernel
4. Various **gradient-based** edge detection filters
5. **Laplace filter**- second derivation-based edge detection filter.
6. **Morphology transformation**

2) Image smoothing (blurring, noise reduction)

The goal of a **mean filter** is to replace each pixel value in an image with the mean (`average') value of its neighbors, including itself. Mean filtering is a simple method for noise reduction:

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

Fig. 7 The convolution kernel of a 3x3 mean filter.

An example of a non-linear operator is the **median** filter – applied for noise reduction. It replaces the pixel value with the *median* of neighbor values.

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

3x3 neighborhood values:
115, 119, 120, 123,
124,
125, 127, 127, 150

Median value: 124.

Fig. 8. The central pixel value of 150 is replaced by the median value.

The median filter has two main advantages over the mean filter:

- The median is a more robust average than the mean - a single neighbor will not affect the median value significantly.
- It does not create new unrealistic pixel values when the filter includes an edge – it is better preserving sharp edges than the mean filter.

The **Gaussian smoothing** operator is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a kernel that represents the Gaussian shape (i.e. `bell-shaped').

In 2-D, an isotropic (*i.e.* circularly symmetric) Gaussian takes the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2} \right) \tag{12}$$

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Fig. 9 Discrete approximation to Gaussian function with $\sigma = 1.0$.

The 2-D isotropic Gaussian is separable into *x* and *y* components, that apply the same 1-D convolution kernel to rows and columns.

.006	.061	.242	.383	.242	.061	.006

Fig. 10 A 1-D Gaussian convolution kernel.

A Gaussian filter provides gentler smoothing and preserves edges better than a similarly sized mean filter.

3) Edge detection

An **edge** – the border between two homogeneous image regions having different illumination intensities or colors.

This definition implies that an edge is a local variation of illumination. Hence the main approach to edge detection: to localize discontinuities of the intensity function.
Usually a **pre-processing step** performs a **smoothing operation** in order to remove noise (spiky-like variations) from the image.

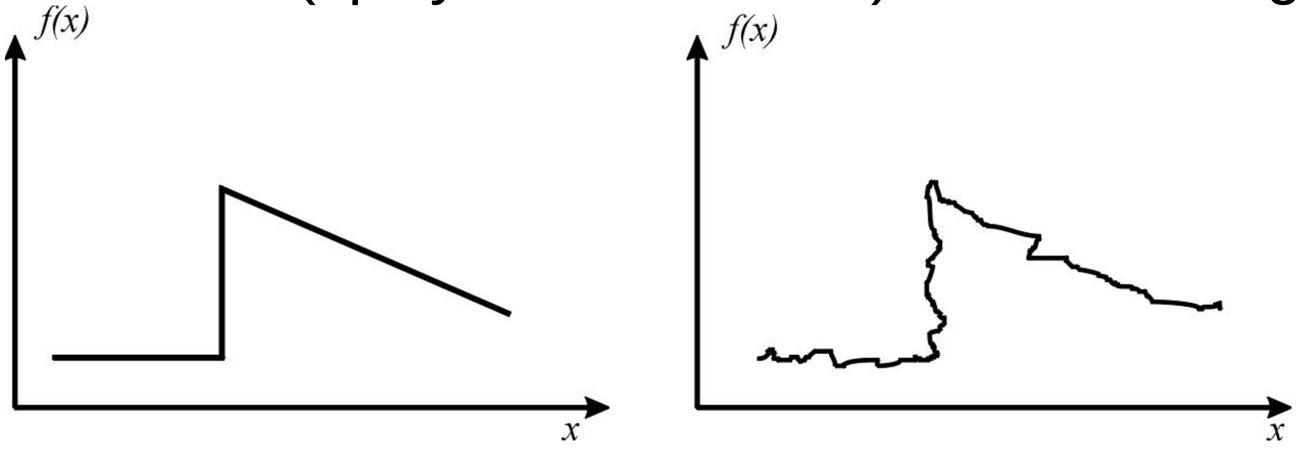


Fig 11. 1-D discontinuities: ideal step-like, a "noisy" step discontinuity in real images.

Discrete gradients of intensity function

Rapid changes and discontinuities of the intensity function are detected as:

- Local maximum peak of the first derivative, or
- A zero of the second derivative.

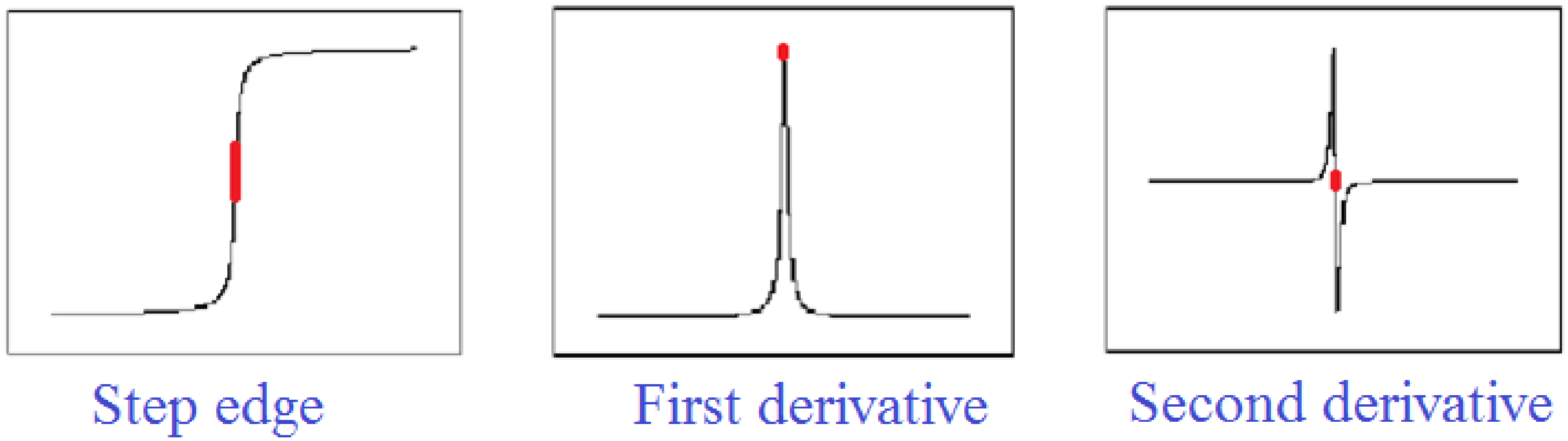
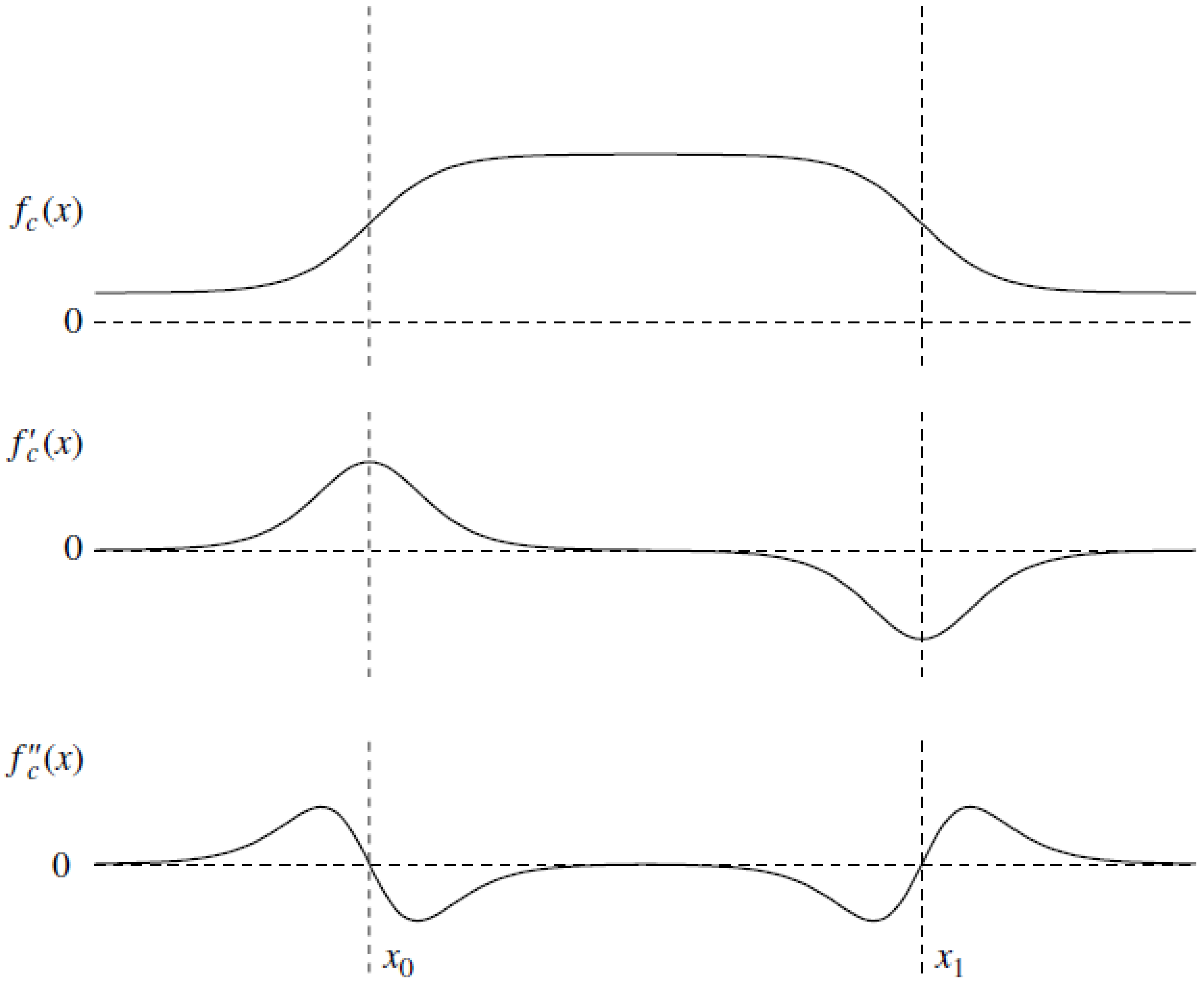


Fig. 12. Examples of rapid changes and discontinuities of a 1-D function.



Source: Milsna, Rodrigues: Gradient and Laplacian Edge Detection

The gradient of a 2-D continuous function:

$$\nabla f(x,y) = \begin{pmatrix} f_x(x,y) \\ f_y(x,y) \end{pmatrix} = \begin{pmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{pmatrix} \tag{13}$$

Discrete differences in case of a 2-D discrete image function $\hat{f}(i,j)$, that approximates the continuous function $f(x,y)$:

$$\hat{f}_x(i,j) = \hat{f}(i+1,j) - \hat{f}(i,j) \;;$$

$$\hat{f}_y(i,j) = \hat{f}(i,j+1) - \hat{f}(i,j) \tag{14}$$

$$\hat{f}_x(x,y) = \lim_{h\to 0} \frac{f(x+h,y) - f(x,y)}{h} \tag{15}$$

As a result of edge detection two maps (images) are computed:

(1) the magnitude (**strength**) ***s*** or the "absolute" strength ***s'*** (for computational simplicity)

$$s = \sqrt{f_x^2 + f_y^2}$$

or

$$s' = | f_x | + | f_y | \qquad (16)$$

(2) and the **direction** ***r*** of the normal vector to edge :

$$r = \arctan(f_y / f_x)$$

Note:

r stores the direction of the **normal vector** of the detected edge line.

Sobel operator

It combines smoothing (triangular filter) and differentiation.

The Sobel operator uses two kernels of odd size, e.g. for size 3×3:

k_x

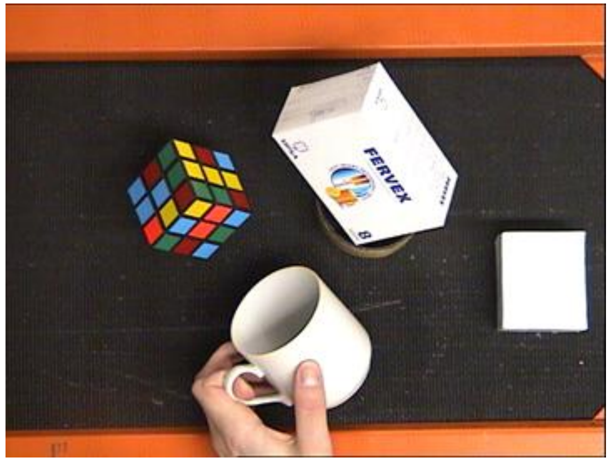
-1	0	1
-2	**0**	2
-1	0	1

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

k_y

Fig. 13 The two Sobel operator kernels for size 3×3.

(a)



(b)

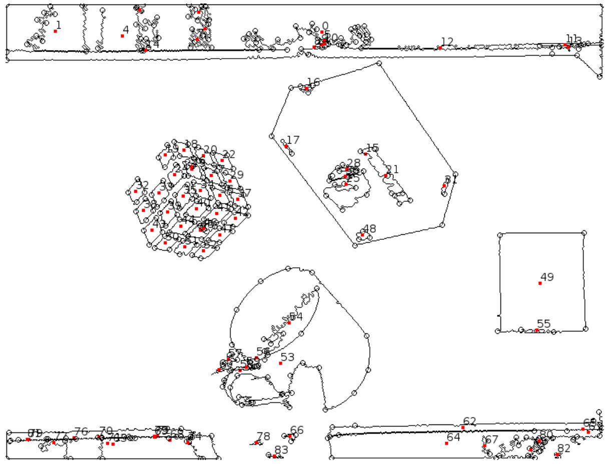


Fig. 14 Illustration of an edge operator: input image, edge chains in the image

Discrete directions of an edge element (remark: the image matrix co-ordinate system has the Y axis oriented top-down).

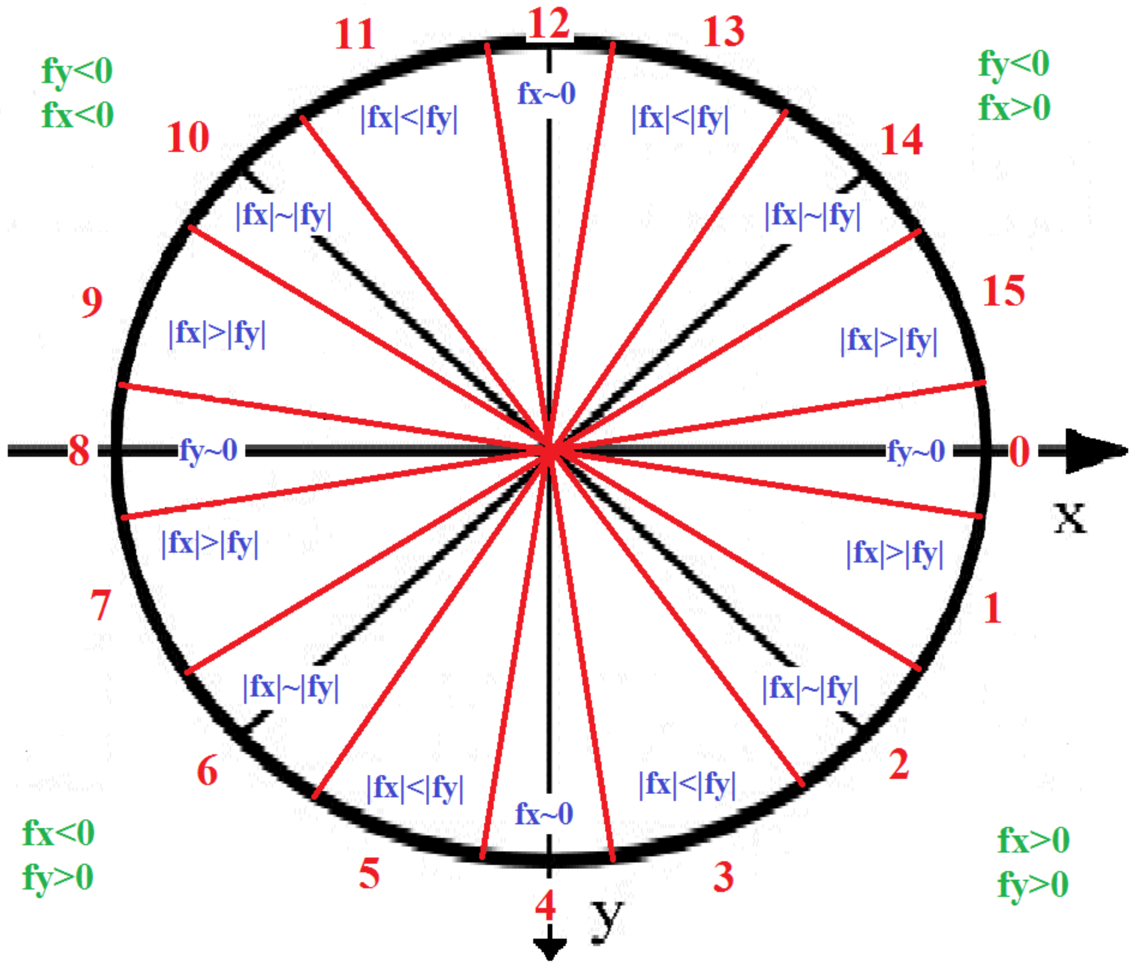


Fig. 15 Example of 16 discrete image directions computed in an efficient way (by comparison operations).

The Scharr operator (OpenCV)

For the kernel size is 3x3, in OpenCV the Scharr function is defined, which implements the following kernels:

$$G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

Laplacian operator (second-order derivation)

For a continuous 2-D function the Laplacian operator is defined as:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = f_{xx} + f_{yy} \qquad (17)$$

This operator is independent from any direction.

For a discrete function the Laplacian operator is approximated by:

$$\nabla^2 f(i,j) = \Delta_i \Delta_i f(i,j) + \Delta_j \Delta_j f(i,j) =$$
$$= 4 f(i,j) - f(i-1,j) - f(i+1,j) - f(i,j-1) - f(i,j+1) \qquad (18)$$

Using a second derivative makes the Laplacian highly sensitive to noise → use Gaussian smoothing as a preprocessing step.

Laplacian of the Gaussian (LOG filter):

$$L(x,y) = \nabla^2 (G(x,y) * I(x,y))$$

Due to linearity (of discrete realizations):

$$\boxed{L(x,y) = \nabla^2 G(x,y) * I(x,y)}$$

Derivation of the LOG filter (1D case):

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{x^2}{2\sigma^2})$$

- first derivative:

$$G'(x) = -\frac{x}{\sigma^3\sqrt{2\pi}} \exp(-\frac{x^2}{2\sigma^2})$$

- second derivative:

$$G''(x) = \frac{1}{\sigma\sqrt{2\pi}} (\frac{x^2}{\sigma^3} - \frac{1}{\sigma}) \exp(-\frac{x^2}{2\sigma^2})$$

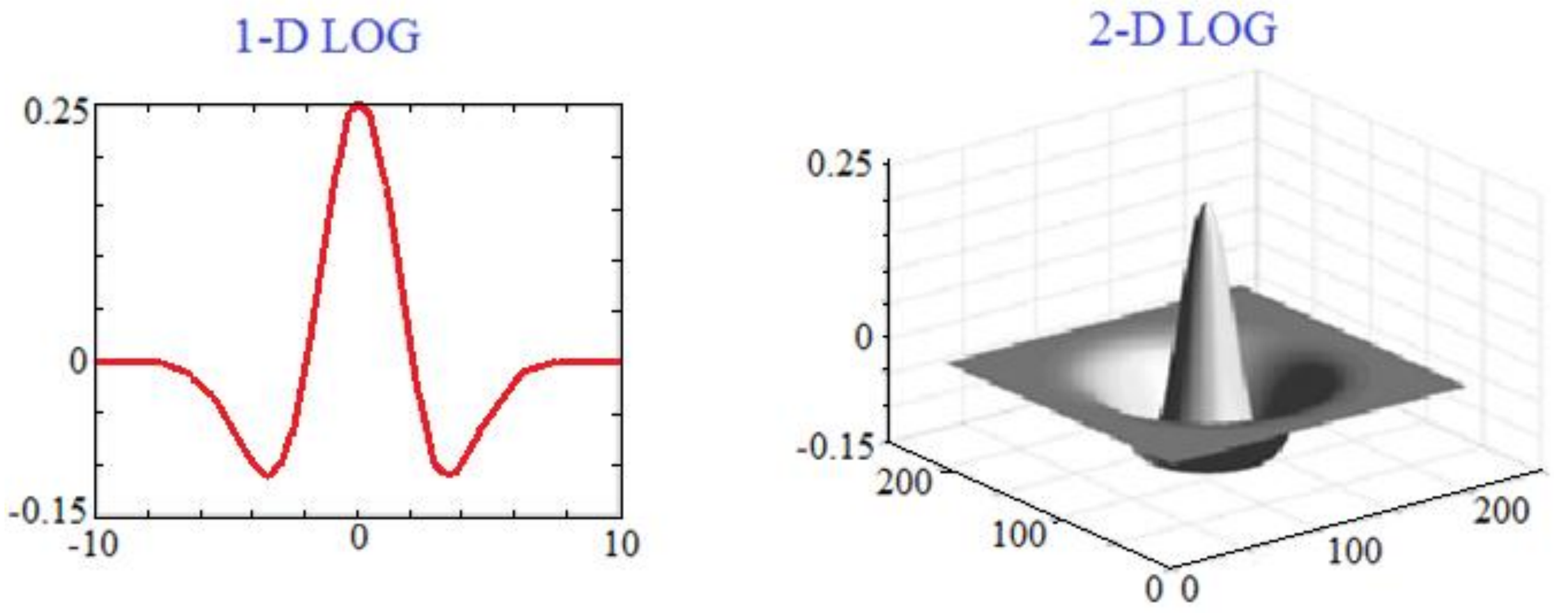


Fig. 16 Illustration of the LOG filter

Discrete Laplace operators in kernel mask form:

0	-1	0
-1	4	-1
0	-1	0

(a)

-1	-1	-1
-1	8	-1
-1	-1	-1

(b)

1	-2	1
-2	4	-2
1	-2	1

(c)

Fig. 17 Laplace kernels: (a) the basic Laplacian, (b,c) some LOG (Laplacian of Gaussian) filter mask.

Note.

The Laplace operator gives a "strong" answer, i.e. a zero value, for both real edges as well as homogeneous regions in the image.

Due to this fact use is in a combination with some other operator, for example with a gradient-based operator.

4) Morphological filtering

In OpenCV there are basic morphological operators defined as:

erode – perform the **Erosion** operation,

dilate – perform the **Dilation** operation.

Morphological operations apply a **structuring element** to an input image and generate an output image.

Purpose:

- Removing noise;
- Isolation of individual elements or joining disparate elements in an image.
– Finding of intensity bumps or holes in an image.

Structuring element (filter kernel):

- Rectangular box: MORPH_RECT
- Cross: MORPH_CROSS
- Ellipse: MORPH_ELLIPSE

Dilation

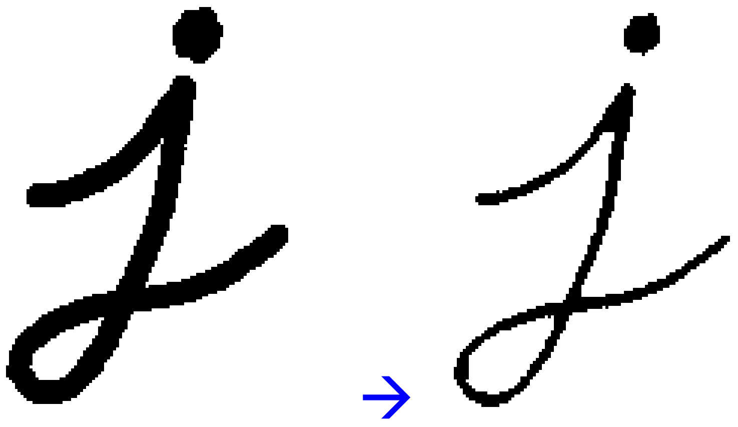
dilate(src, dst, element); or

dilate(src, dst); // default kernel is a 3x3 matrix

As the filter kernel K is scanned over the image, the MAXIMAL pixel value overlapped by K is detected and the image pixel in the anchor point position is replaced with that maximal value.

This maximizing operation causes bright regions within an image to “grow”.

Example



Erosion

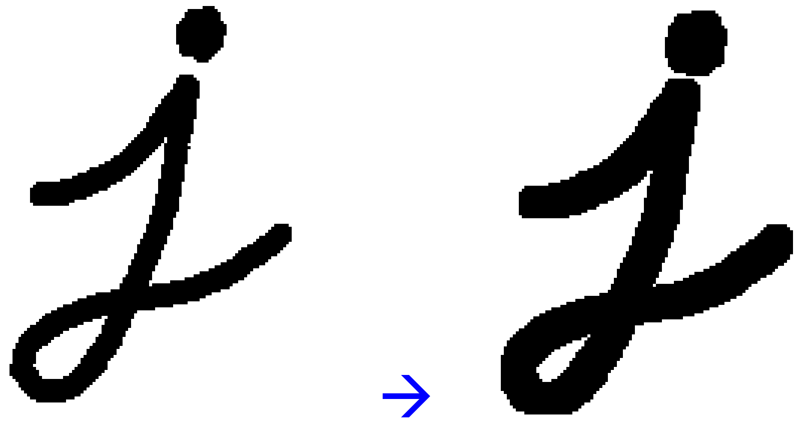
erode(src, dst, element); or

erode(src, dst); // default kernel is a 3x3 matrix

As the filter kernel K is scanned over the image, the MINIMAL pixel value overlapped by K is detected and the image pixel under the anchor point is replaced with that minimal value.

Thus, the bright areas of the image (the background, apparently), get thinner, whereas the dark zones (the "object") gets bigger.

Example



Morphological transformations [OpenCV]

Morphological transformations like **Opening** and **Closing**, are based on the Erosion and Dilation operations.

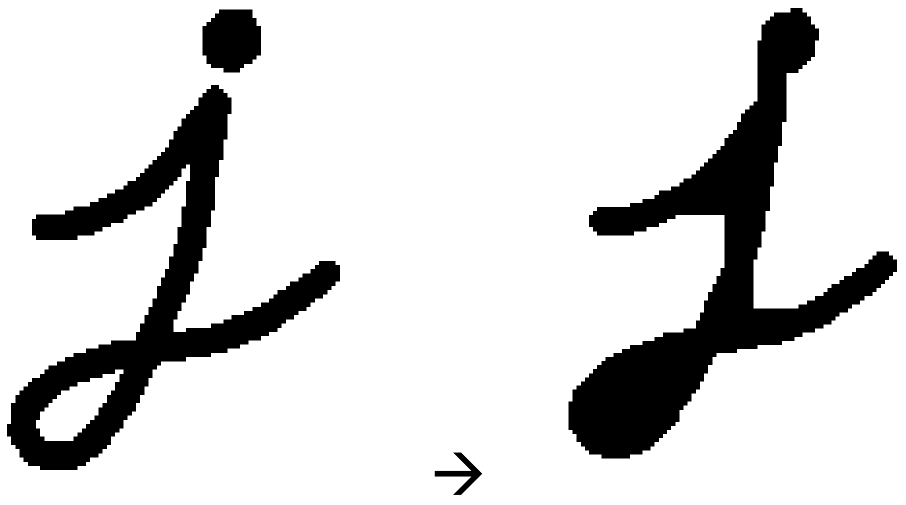
Opening

It is obtained by the erosion of an image followed by a dilation.

$$\mathbf{dst = open(src, element) = dilate(erode(src; element))}$$

Useful for removing small image patches or corners.

Example



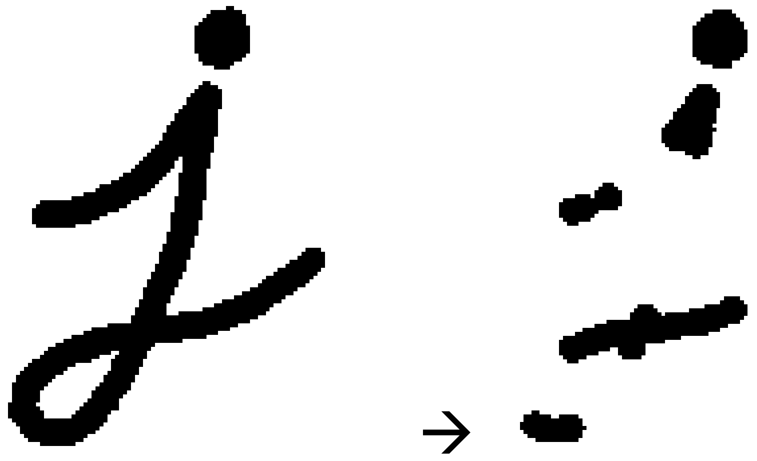
Closing

It is obtained by the dilation of an image followed by an erosion.

dst = close(src, element) = erode(dilate(src; element))

Useful to remove discontinuities in objects.

Example



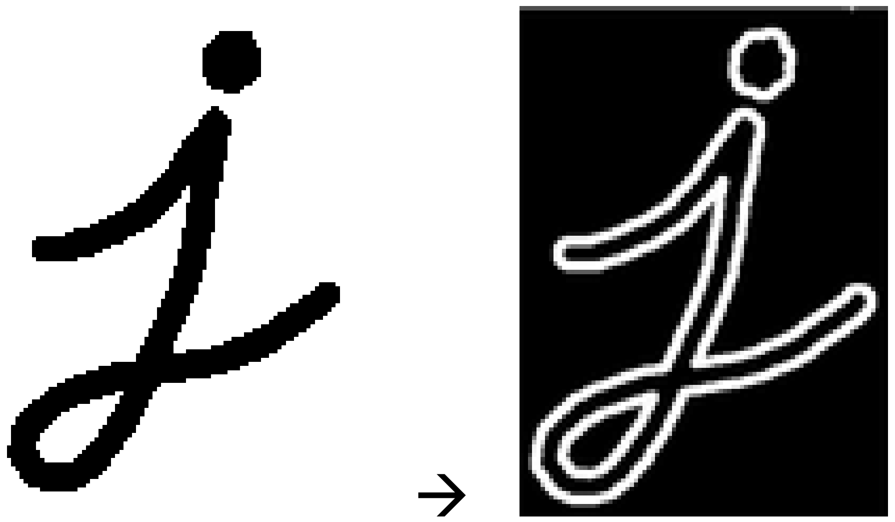
Morphological gradient

It is the **difference** between the dilation and the erosion of an image.

dst = morphgrad(src; element) = dilate(src; element) - erode(src; element)

It is useful for finding the outline of an object.

Example



Other operations (useful for leaving only small objects):

dst = tophat(src; element) = src - open(src; element)

dst = blackhat(src; element) = close(src; element) – src

General function for morphology operations:

morphologyEx(src, dst, operation, element);

where **operation**=

- Opening: MORPH_OPEN : 2
- Closing: MORPH_CLOSE: 3
- Gradient: MORPH_GRADIENT: 4
- Top Hat: MORPH_TOPHAT: 5
- Black Hat: MORPH_BLACKHAT: 6

5) Edge operator for color images

The extension of the Sobel operator for color images can be provided while exchanging the difference of elements of a single discrete function ***f*** by a difference measure in color space.

Let pairs of color pixels be given: $f_1 = (r_1, g_1, b_1) , f_2 = (r_2, g_2, b_2)$.
For a Sobel operator there exists three such pairs for the "vertical edge" mask and 3 pairs for the "horizontal edge" mask.

Possible distance measures:

$$D_1 (f_1 ; f_2) = \{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2 \}^{1/2}$$

$$D_2 (f_1, f_2) = | r_1 - r_2 | + | g_1 - g_2 | + | b_1 - b_2 |$$

$$D_3 (f_1, f_2) = \max \{ | r_1 - r_2 |, \ | g_1 - g_2 |, \ | b_1 - b_2 | \} \quad (19)$$

$$D_4 (f_1, f_2) = \omega_r | r_1 - r_2 | + \omega_g | g_1 - g_2 | + \omega_b | b_1 - b_2 |$$

5. „Edge thinning”

1) Method 1

The simplest **edge thinning** method means an **edge suppression** operator working with a **minimum threshold θ**. The threshold is either fixed or set adaptively for current edge image (i.e. $\theta = \gamma S_{\max}$, where $\gamma \in (0,1)$). Due to this operator "weak" edge elements are eliminated from the edge image (i.e. they are reset to "zero"):

$$s_{thin} = \begin{cases} s(P), if \ s(P) > \theta \\ 0, otherwise \end{cases} \tag{20}$$

The above operator requires only a single run (no iteration) for the edge image.

2) Method 2

In the next edge thinning method, that we call **non maximum suppression operator**, a check in the **local neighborhood** of given pixel P is made. Pixels remain which have maximum strength among its potentially competing neighbors, i.e.:

$$\begin{aligned} & IF\ ((s(P) \geq s(N_L)\ OR\ |r(P) - r(N_L)| \geq 30^o) \\ & \quad AND\ (s(P) \geq s(N_R)\ OR\ |r(P) - r(N_R)| \geq 30^o)\ ) \\ & THEN\ s(P)_{thin} = s(P); \quad ELSE\ s(P)_{thin} = 0; \end{aligned} \tag{21}$$

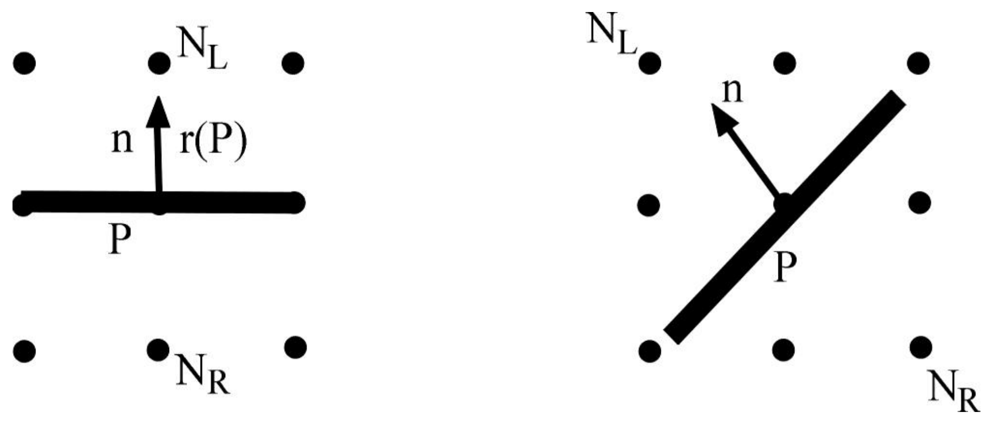


Fig. 18 The localization of competing neighbors N_L and N_R for a candidate (tested) pixel P (for "horizontal" lines and "diagonal" lines).

3) Canny thinning

Edge suppression with a hysteresis threshold.

- It works with two edge strength thresholds: the upper θ_H and the lower θ_L.
- In the first run over the whole image only theses edge pixels are individually marked as „good" that show strengths higher than the upper threshold.
- In the next run these "good" pixels are tried to be "extended" along a potential contour line in both directions ("positive" and "negative" line direction). For extension, the neighbor pixels need to show strengths that are higher than the lower threshold.

Remark: now the neighbors are not competing with each other and they are searched along the contour line (not across it).

6. The Canny operator

The Canny operator is a multi-stage edge detection and thinning procedure that is best suitable for images corrupted by Gaussian noise. Basically it requires 3 parameters:

- the variance σ for the Gaussian mask,
- edge strength thresholds θ_H, θ_L, where $\theta_H > \theta_L$, for hysteresis-based thinning.

Input to the Canny operator: a grey-scale image;

Result of Canny operator: a “thinned” edge image.

Steps of the Canny operator

1. Image smoothing by discrete convolution with a Gaussian mask.
2. Edge image detection by a simple first derivative operator in a 3x3 neighborhood.
3. Edge thinning by the no maximum suppression operator.
4. Edge suppression with a hysteresis threshold.

EXERCISES 2

Example 2.1 Color transformation

Assume that in a computer program we represent all the colour coefficients in terms of **8 bits** – using unsigned **integer** values from the interval [0, 255].

1. Perform transformations between RGB and YUV colour spaces: for blue, green and red colors.

Example 2.2

Suppose that a 64×64, *8*-level image has the intensity-value distribution as shown in the table below.

Pixel value w	Histogram value $h(w)$	pdf value $p_w(w)$
0	790	0.19
1	1023	0.25
2	850	0.21
3	656	0.16
4	329	0.08
5	245	0.06
6	122	0.03
7	81	0.02

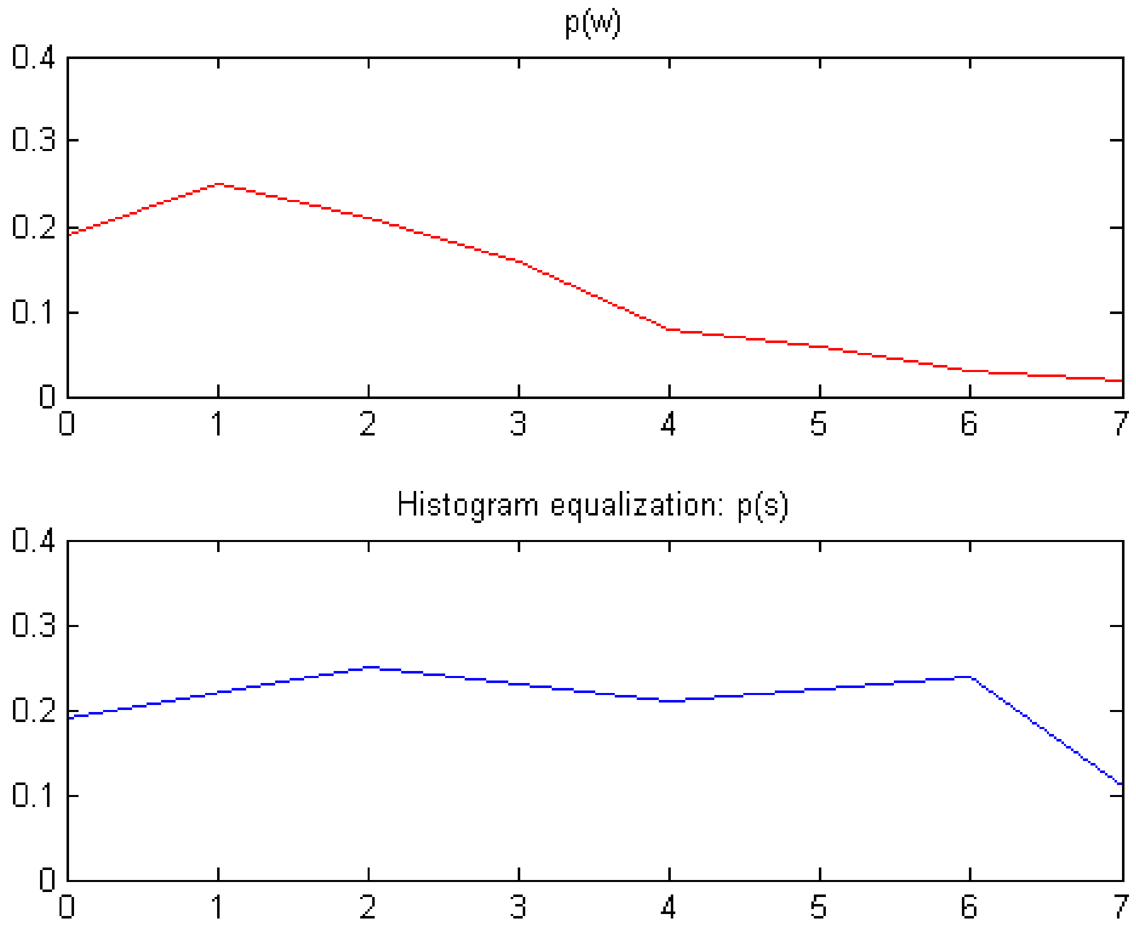
1. Perform *histogram equalization*.
2. Give graphical illustration of both histograms.
3. Determine the number of distinct levels in the output image.
4. Specify the pdf for values of the final output image.

Solution 2.2

1.

w	$p_w(w)$	$F_w(w)$	$z = T(w)$	5. new pmf $p_z(z)$
0	0.19	0.19	$0.19 \cdot 7 = 1.33 \rightarrow 1$	0.19
1	0.25	0.44	$3.08 \rightarrow 3$	0.25
2	0.21	0.65	$4.55 \rightarrow 5$	0.21
3	0.16	0.81	$5.67 \rightarrow 6$	0.16 + 0.08 = 0.24
4	0.08	0.89	$6.23 \rightarrow 6$	
5	0.06	0.95	$6.65 \rightarrow 7$	0.06+0.03+0.02 = 0.11
6	0.03	0.98	$6.86 \rightarrow 7$	
7	0.02	1.00	7	

2.



2. Reduction of the level number from 8 to 5.

Example 2.3

Let the following image be given.

3	2	8	2
5	1	7	1
9	1	6	1
8	0	5	2

1. Determine the threshold for image **binarization** using the **Otsu method**. Show intermediate results.
2. Perform **image thresholding**.

Example 2.4

Apply the Sobel and Scharr *edge operators* to the two images given below. Present the edge strengths and orientations. Provide comments to these results.

1)

1	3	5	4
1	2	5	5
2	3	5	5
1	2	4	5
0	3	5	5

2)

1	2	1	2
2	2	3	6
3	3	6	6
3	6	6	5
5	6	6	6

Example 2.5

Simulate 3 different *edge thinning* procedures as applied to the results of the Sobel operator for two above images (given in example 2.4).