Comparative analysis of contour detection algorithms in images in computer vision tasks. Part I

Dmytro Mishchuk[1], Yevhen Mishchuk[2], Dmytro Korzhevin[3]

[1,2] Kyiv National University of Construction and Architecture,
31 Air Force Avenue, Kyiv, 03037, Ukraine,
[3]Cisco Talos Intelligence,
8135 Maple Lawn Blvd, Fulton, MD 20759, USA
[1]mischuk.do@knuba.edu.ua, https://orcid.org/0000-0002-8263-9400,
[2]mischuk.ieo@knuba.edu.ua, https://orcid.org/0000-0002-1888-3687,
[3]dkorzhev@cisco.com, https://orcid.org/0009-0002-1916-5019

Received: 12.11.2025, accepted: 08.12.2025
https://doi.org/10.32347/st.2025.4.1208

Abstract. Identifying contours in images is a fundamental task in computer vision, underpinning many modern technologies. Their accurate detection enables the solution of a wide range of practical problems, including segmentation and object recognition, 3D reconstruction, medical diagnostics, and autonomous control of systems.

In the field of computer vision, significant progress has been made in developing image processing algorithms; however, many unsolved problems remain. The tasks of improving the speed of solving existing algorithms and developing algorithms for working with limited computational memory resources remain relevant.

This paper presents a description and analysis of several well-known image processing algorithms for detecting contours in an image.

It is known that an image in computer systems is a set of pixels of a given brightness and color. Such a representation can be represented in the form of matrices, where each of its elements is a pixel, and the position is the coordinates of each pixel. Thus, image processing from the position of mathematics is working with matrices and sets. The quality of the processing algorithm will depend on the chosen method of working with matrices.

Neural networks, although not a new approach, have gained widespread popularity relatively recently, mainly due to the development of convolutional neural networks, which specialize in processing photo and video data. In addition, the use of recurrent neural networks allows you to search for the optimal architecture for neural network systems. Different models have been adapted to specific constraints, such as the size of the model itself, computational resources, or ensuring the required accuracy.

Alternative approaches to neural networks are algorithms based on detectors and descriptors. In essence, these are algorithms that allow you to compare images based on some basic features. For example, to understand that two different photos show the same object, the computer does not compare every pixel, but looks for "special" places and describes them mathematically, which significantly speeds up image processing, but can lead to errors.

Dmytro Mishchuk
Ph.D. ,department of construction machinery

Yevhen Mishchuk
Ph.D., department of machinery and equipment of technological processes

Dmytro Korzhevin
Cisco

Keywords: Roberts operator, Prewitt operator, Laplassian, erosion, blurring, morphological processes, dilation, normalization.

INTRODUCTION

Contours are boundaries between objects or areas with different visual characteristics.

Today visual information processing is becoming increasingly automated; and contour detection methods are widely used in various industries. For example, in medicine, such image processing algorithms are used to analyze medical images (X-rays, MRI, CT scans), which allows doctors to diagnose diseases faster and more accurately. In the field of autonomous vehicles, contours help guidance systems detect obstacles, pedestrians, and other objects on the road. In remote sensing of the Earth, contour detection is used to process satellite images, which is important for monitoring environmental changes, urban planning, and resource management.

Thus, the development and improvement of contour detection methods is a relevant area of research, since the quality and reliability of many automated systems depend on their effectiveness.

PURPOSE AND OBJECTIVES

The main goal of this work is to analyze and compare classical and modern methods of contour detection.

The objectives of the work were to identify the advantages and disadvantages of each approach and assess the applicability of the methods depending on the type of images and tasks.

PRESENTATION OF THE MAIN MATERIAL

Image edge detection methods can be classified according to the approach used to detect the boundaries between objects. The main known detection categories include:

1. *Gradient methods* - These methods are based on the analysis of the intensity changes of pixels in an image. Contours are defined as areas with a high brightness gradient. The most common algorithms include the Sobel, Prewitt, Roberts operators, and the Canny operator, which combines noise filtering, gradient calculation, and hysteresis thresholding to obtain thin and continuous contours.
2. *Laplacian-based methods.* The Laplacian is a second-order differential operator that detects areas of rapid intensity change. This method is sensitive to noise, so it is often used in conjunction with pre-filtering (e.g., Gaussian blur). Laplacian-based methods allow for the detection of contours of varying thickness, but require additional processing to eliminate false detections.
3. *Methods based on morphological analysis.* Morphological processes (erosion, dilation, opening, closing) are used to extract contours by processing binary or gray images. These methods are especially effective for processing images with clear boundaries between objects, for example, in the tasks of texture segmentation or analysis of microscopic images.
4. *Methods based on active contours (Snakes, Level Set)* - these are methods that use deformed curves that adapt to the boundaries of objects under the influence of internal and external forces. Active contour methods allow you to obtain smooth and continuous contours, but require an initial approximation and are computationally more complex compared to gradient methods.
5. *Machine learning and deep learning methods* - these are modern approaches to edge detection that are built on the basis of fuzzy neural network logic, in particular convolutional neural networks (CNN) and U-Net-type architectures. These methods allow you to automatically learn to detect edges based on large data sets, which significantly increases accuracy and noise resistance. Examples: HED (Holistically-Nested Edge Detection) algorithms, RCF (Rich Convolutional Features).

The Roberts operator is one of the simplest methods for determining contours. It calculates the gradient using two 2×2 matrices (convolution kernels) that approximate derivatives along the diagonals:

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}; \quad G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \qquad (1)$$

Algorithm for applying the Roberts operator. Suppose that the input image is represented as a two-dimensional matrix of pixels I with size $M \times N$, where each pixel has an intensity from 0 to 255. At the output, this algorithm returns a new image matrix G with selected contours of size $M \times N$.

To implement this algorithm, you first need to create an empty matrix G with size $M \times N$ and specify Roberts kernels – matrices of size 2×2 (1). To process the original image, it is necessary for each pixel $I_{(i,\ j)}$ (where $1<i<(M-1)$, $1<j<(N-1)$) calculate the gradient value using the next formula:

$$g_x = I_{(i,\ j)} \cdot G_{x(0,\ 0)} + I_{(i,\ j+1)} \cdot G_{x(0,\ 1)} + I_{(i+1,\ j)} \cdot G_{x(1,\ 0)} + I_{(i+1,\ j+1)} \cdot G_{x(1,\ 1)} \quad (2)$$

$$g_y = I_{(i,\ j)} \cdot G_{y(0,\ 0)} + I_{(i,\ j+1)} \cdot G_{y(0,\ 1)} + I_{(i+1,\ j)} \cdot G_{y(1,\ 0)} + I_{(i+1,\ j+1)} \cdot G_{y(1,\ 1)} \quad (3)$$

Next, the gradient modulus is calculated, which will reflect the pixel value in the new image:

$$G_{(i,j)} = \sqrt{g_x^2 + g_y^2} \quad (4)$$

To visualize the gradient values, they need to be normalized. To do this, you need to find the minimum $G_{\min}$ and maximum $G_{\max}$ values in the gradient matrix and give each value $G_{i,j}$ to the range [0, 255] according to the following formula:

$$G_{norm\ (i,\ j)} = round\left[255 \cdot \frac{G_{(i,\ j)} - G_{\min}}{G_{\max} - G_{\min}} \right], \quad (5)$$

where $G_{norm\ (i,\ j)}$ - normalized pixel brightness value.

If $G_{norm(i,\ j)} < 0$ then $G_{norm\ (i,\ j)} = 0$, if $G_{norm(i,\ j)} > 255$ then $G_{norm\ (i,\ j)} = 255$, if $G_{\min} = G_{\max}$ then $G_{norm\ (i,\ j)} = 0$.

The advantages of this method of image processing are its simplicity of implementation, but for large image sizes it requires a large number of calculations and sorting of the gradient array. This method is also sensitive to noise and can detect contours only along diagonals, which can lead to missing some boundaries. It is mainly used for fast processing of small images or as a preliminary stage in more complex algorithms.

The Prewitt operator uses two kernel matrices of size 3×3 to calculate the horizontal gradient (G_x) and vertical (G_y):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}; \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}. \quad (6)$$

Algorithm for applying the Prewitt operator. For the image I in size M×N an empty matrix G created with size M×N. Next needs to specify the Prewitt kernels (6).

For each pixel $I_{(i,\ j)}$, where $1 < i < M-1$ and $1 < j < N-1$, need to calculate the gradient values:

$$g_x = I_{(i-1,\ j-1)} \cdot G_{x(0,\ 0)} + I_{(i-1,j)} \cdot G_{x(0,\ 1)} + I_{(i-1,\ j+1)} \times G_{x(0,\ 2)} + I_{(i,\ j-1)} \cdot G_{x(1,\ 0)} + I_{(i,\ j)} \cdot G_{x(1,\ 1)} + I_{(i,\ j+1)} \cdot G_{x(1,\ 2)} + I_{(i+1,\ j-1)} \cdot G_{x(2,\ 0)} + I_{(i+1,\ j)} \cdot G_{x(2,\ 1)} + I_{(i+1,\ j+1)} \cdot G_{x(2,\ 2)}; \quad (7)$$

$$g_y = I_{(i-1,\ j-1)} \cdot G_{y(0,\ 0)} + I_{(i-1,\ j)} \cdot G_{y(0,\ 1)} + I_{(i-1,\ j+1)} \times G_{y(0,\ 2)} + I_{(i,\ j-1)} \cdot G_{y(1,\ 0)} + I_{(i,\ j)} \cdot G_{y(1,\ 1)} + I_{(i-1,\ j+1)} \cdot G_{y(1,\ 2)} + I_{(i+1,\ j-1)} \cdot G_{y(2,\ 0)} + I_{(i+1,\ j)} \cdot G_{y(2,\ 1)} + I_{(i+1,\ j+1)} \cdot G_{y(2,\ 2)}. \quad (8)$$

This algorithm ignores the extreme rows and columns, since the kernel is 3×3 and does not fit on the edges. Next, the gradient modulus is determined using a similar formula (4), and normalization is performed.

This algorithm detects contours better than the Roberts algorithm, but is also sensitive to noise - small objects or individual pixels. It is often used for processing medium-quality images, where speed and simplicity are important.

The Sobel operator is the most common of these three. It also uses two 3×3 cores, but with additional weighting for the center row/column, which reduces sensitivity to noise:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; \quad (9)$$

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

The gradient for the Sobel operator is calculated similarly to that for the Prewitt operator. This method can suppress noise better than Prewitt and Roberts, which allows you to more accurately determine the direction of the contour. The disadvantages of this method of contour determination are that this algorithm can produce thick contours and requires additional processing for fine boundary selection. The Sobel operator using in computer vision systems, medical imaging, and satellite image processing.

The Laplacian calculates the change in the intensity of the image brightness at each point, indicating the boundaries of objects where the brightness changes sharply and is defined as the sum of the second derivatives in the coordinates x and y:

$$\nabla^2 = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}, \quad (10)$$

where: I — pixel intensity.

Brightness function I depends on the coordinates of the pixel in the image – x and y, that is $I(x, y)$ – is the brightness of the pixel at position (x, y). The derivative shows how quickly the brightness changes when moving from one pixel to the next. Since it is not possible to calculate the derivative directly in the image, the following approximation is used:
- the first derivative is approximated as the difference in brightness of neighboring pixels:

$$\frac{\partial I}{\partial x} \approx I_{(x+1,y)} - I_{(x,y)}; \quad (11)$$

- second derivative – difference of first derivatives:

$$\frac{\partial^2 I}{\partial x^2} \approx (I_{(x+1,y)} - I_{(x,y)}) - (I_{(x,y)} - I_{(x-1,y)}). \quad (12)$$

In practice, for discrete images, instead of searching for gradients, Laplacian approximation is used using a convolution kernel, for example:

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \quad (13)$$

Algorithm for applying the Laplacian. At the beginning of processing, the original image must be converted to grayscale (if it is color) and transformed into a matrix of pixels I with normalization to values from the range [0, 1] or [0, 255]. Next, a new matrix of the same size is created, where each pixel will be the result of applying the Laplacian to the corresponding pixel of the original image. To do this, a kernel (kernel convolution) is applied to each pixel of the original image (except for the boundaries) so that the center of the kernel coincides with the current pixel. Next, the new pixel value is calculated as the sum of the products of the image pixel values and the corresponding kernel values:

$$\begin{aligned} \text{new\_pixel}_{(x,y)} &= I_{(x-1,y-1)} \cdot K[0,0] + \\ &+ I_{(x-1,y)} \cdot K[0,1] + I_{(x-1,y+1)} \cdot K[0,2] + \\ &+ I_{(x,y-1)} \cdot K[1,0] + I_{(x,y)} \cdot K[1,1] + \\ &+ I_{(x,y+1)} \cdot K[1,2] + I_{(x+1,y-1)} \cdot K[2,0] + \\ &+ I_{(x+1,y)} * K[2,1] + I_{(x+1,y+1)} * K[2,2]. \end{aligned} \quad (14)$$

Pixels at the image borders can be ignored (left black), reflected, or border extended.

After convolution with the Laplacian kernel, pixel values can be both positive and negative, so normalization to the range [0, 255] is applied for visualization according to the algorithm using the formula (5).

Sometimes thresholding is also applied after normalization to retain only the strongest contours:

```
if abs(normalized_pixel) > threshold:
    edge_pixel = 255
else:
    edge_pixel = 0
```

This approach allows for contour detection because the Laplacian responds to abrupt changes in intensity.

The Kenny algorithm is a multi-stage algorithm that combines high accuracy in detecting significant edges with minimizing false positives. It is one of the most effective methods for detecting contours in computer vision. The Kenny algorithm is not limited to calculating the smoothed gradient of the image. Only the points of maximum gradient of the image remain in the contour, and the maxima near the boundary are removed. His method also uses information about the direction of change of the image contour boundary in order to remove noise near the boundary and not break the contour near local gradient maxima. Further, weak boundaries are removed using two thresholding processes. Fragments of the contour boundaries are processed as a single whole fragment. If the gradient value anywhere on the processed fragment exceeds the upper threshold, then such a fragment remains valid even in those places where the gradient value is lower than this threshold value, until the gradient value becomes lower than the lower threshold. If there is no point with a value greater than the upper threshold on the entire fragment, then such a fragment is deleted. This approach allows you to reduce the number of breaks in the original boundaries. The use of noise reduction in the Kenny algorithm increases the stability of the results, but increases the computational costs and can lead to distortion and loss of clarity of boundaries. For example, this algorithm rounds the corners of objects and destroys the boundaries at the connection points.

The algorithm for applying the Kenny method consists of sequentially executing the following image processing cycles.

1. Gaussian smoothing, which is essentially a blurring of the original image to remove noise (small scattered pixels). A two-dimensional Gaussian filter is used - a matrix (kernel) that processes the pixels of the original image. Each pixel of the kernel is a weighted average of its neighbors, and the weights are determined by the Gaussian function:

$$G_{(x,y)} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \qquad (15)$$

where σ – standard deviation, which determines the degree of "blur" (the larger this value, the stronger the blur and the better the noise suppression); x, y – pixel coordinates relative to the center of the filter kernel.

For a blur kernel of size 3×3 and $\sigma = 1$ were define the components of the matrix:

$$G_{(0,0)} = \frac{1}{2 \cdot 3{,}14 \cdot 1^2} 2{,}71^{-\frac{0^2+0^2}{2 \cdot 1^2}} = 0{,}159$$

$$G_{(-1,0)} = G_{(0,-1)} = G_{(0,1)} = G_{(1,0)} =$$

$$= \frac{1}{2 \cdot 3{,}14 \cdot 1^2} 2{,}71^{-\frac{1^2+0^2}{2 \cdot 1^2}} = 0{,}262$$

$$G_{(-1,1)} = G_{(1,-1)} = G_{(-1,-1)} = G_{(1,1)} =$$

$$= \frac{1}{2 \cdot 3{,}14 \cdot 1^2} 2{,}71^{-\frac{1^2+1^2}{2 \cdot 1^2}} = 0{,}432$$

Then the blur filter kernel will be like this:

$$g = \begin{bmatrix} 0{,}432 & 0{,}262 & 0{,}432 \\ 0{,}262 & 0{,}159 & 0{,}262 \\ 0{,}432 & 0{,}262 & 0{,}432 \end{bmatrix}. \qquad (16)$$

In practical applications, a simplified Gaussian formula is often used for each pixel:

$$GS_{(x,y)} = e^{-\frac{x^2+y^2}{2\sigma^2}}. \qquad (17)$$

In this case, the resulting kernel matrix is normalized. To do this, the sum of all elements of this matrix is determined, and each of its elements is divided by this sum. This allows you not change the brightness of the original image. To calculate the new brightness for a specific pixel, you need to superimpose the convolution kernel on the original image. To do this, superimpose the center of the convolution matrix on the selected pixel (usually starting from the upper left corner). Next, the brightness value of each neighboring pixel must be

multiplied by the corresponding coefficient of the kernel matrix, which is "underneath it". The resulting values added, and the result will be the new (smoothed) pixel value:

$$p_{new} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I_{(x+i,y+j)} \cdot g_{(i,j)} \ , \quad (18)$$

where p_{new} is pixel intensity value of the new image; $I_{(x+i,y+j)}$ the pixel intensity of the original image; $g_{(i,j)}$ is blur kernel value.

The resulting total value is written to a new image array at the same location as the central kernel pixel. Then the kernel matrix is shifted one pixel to the right in the original image, and the procedure is repeated. When the row ends, the matrix moves to the next row. To process the edges of the image, when the center of the matrix is at the outermost pixel of the photo, part of the kernel protrudes beyond the image. To solve this problem, either addition or ignoring is used. When adding, a frame of black pixels is added, or the outermost pixels are duplicated. Ignoring – the outermost strip with a width equal to the kernel radius is not processed. After this step, an image will be obtained where small details and noise are absent, and important contours remain, but slightly blurred.

2. Applying a gradient (usually Sobel), which allows you to determine the intensity (strength) of the gradient and its direction.

3. Non-Maximum Suppression – fine-tuning of boundaries. The essence of this stage is to “thin out” the edges. Only those pixels that are local intensity maxima in the gradient direction are left. The gradient direction (which can be at any angle from 0 to 360°) is rounded to one of four main directions: 0° (horizontal), 45° (diagonal), 90° (vertical), 135° (diagonal). For each pixel, it is compared with two neighboring pixels in the same direction. If the gradient is directed horizontally (0°), the current pixel is compared to the pixels to the left and right. If the current pixel has the highest intensity among these three, then it is left. If not, set its brightness to 0.

4. Double threshold filtering, which allows filtering out pixels that have a high gradient due to noise, rather than the real boundary of the object. The algorithm uses two thresholds: lower and upper. All pixels are divided into three types:

- strong – gradient above the upper threshold.
- weak – gradient between thresholds. These are “candidates” in the limit.
- irrelevant – gradient below the lower threshold. These pixels are removed (turned black).

5. Hysteresis edge tracing is the final step, where a weak pixel is recognized as part of a true boundary and remains if it touches a strong pixel. If a weak pixel stands alone and there are no strong pixels around it, then such a pixel is considered noise and is removed. Before using such a detector, the original image is usually converted to grayscale to reduce computational costs. This step is typical for many image processing methods.

Thus, first-order methods (gradient methods) determine contours by calculating the first derivative (change in brightness). To do this, small matrices are used that "slide" over the image. The Roberts operator calculates the difference between diagonal pixels, has a kernel with a matrix 2×2, very fast and finds sharp corners well. Disadvantage is sensitive to noise and intermittent contours. The Prewitt operator uses a mask to calculate the gradient horizontally and vertically. The size of the mask matrix 3×3, suppresses noise better than Roberts due to larger analysis area, but creates "blurred" and thick contours and does not take into account the central weight of the pixel. The Sobel operator is an improved version of the Prewitt operator. In the matrix, the central pixels have a greater weight, the kernel has a size 3×3, quickly solves simple problems, smooth’s noise well and gives a clear gradient direction. The disadvantage is that it leaves thick outlines.

Second-order operators (Laplacian) look for points where the first derivative has an extreme (crossing through zero of the second derivative). Responds equally to contours in all directions, finds very fine boundaries, but is extremely sensitive to noise. Even noise invisible to the eye can turn the result into a

"white noise". Usually used together with blurring (Laplacian of Gaussian).

Kenny algorithm is a multi-step method that includes a whole process, namely Gaussian filtering, Sobel filtering, non-maximal suppression and hysteresis. This algorithm has the best accuracy compared to the previous ones, gives thin, continuous lines. The disadvantage is that it is computationally complex (slower than Sobel). Requires manual adjustment of two thresholds.

In Table 1 shows a comparison of indicators for different image processing operators.

Table 1. Operator comparison

Operator	Core size	Sensitivity to noise	Detection accuracy	Speed
Roberts	2×2	High	Low	High
Prewitta	3×3	Wednesday	Medium	Medium
Sobel	3×3	Low	High	Medium
Kenny	3×3	Low	High	Low
Laplacian	3×3	High	High	Medium

Morphological analysis is a set of image processing methods based on set theory and topology, which allows you to change image parameters and subsequently accelerate processing for contour detection. The main functions of morphological analysis are erosion, dilation, opening and closing. Morphological analysis functions allow you to modify the shape of objects in an image, highlighting or suppressing certain structures, for example, blurring or enhancing the boundaries of foreground pixel regions. Structural elements of modifiers can be small matrices (usually 3×3 or 5×5), which define the shape of operations, in particular, the most commonly used are a rectangle and a circle (approximated on a discrete grid). For the rectangular modifier 3×3, the kernel will be this matrix:

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \quad (19)$$

The principle of the erosion modifier is to reduce the size of objects in an image by blurring their boundaries. The erosion operator takes two pieces of data as input. The first is the image to be eroded, the second is a structural element (the erosion kernel) that determines the exact effect of erosion on the input image. Typically, erosion is best applied to binary images, but there are versions that work with grayscale images.

Erosion for a binary image consists in superimposing an erosion kernel on a part of the pixel array of the output image that is equal in size to the pixel array of the mask. Provided that all the pixels of the mask coincide with the pixels of the output image, then the pixel of the output image that corresponds to the central position of the mask remains unchanged, otherwise such a pixel is converted to the background.

Let A be the original image represented as a function

$$A: Z^2 \rightarrow V, \quad (20)$$

where: V – a set of pixel values (e.g., {0,1} for binary images or [0,255] for grayscale).

If B is a structural element that can also be represented as a matrix B:

$$Z^2 \rightarrow \{0,1\}, \quad (21)$$

where: B defines the shape of the neighborhood (e.g., a 3×3 rectangle), then for binary images the erosion is defined as:

$$A \Theta B(x) = \min_{b \in B} A(x + b) = \min \begin{pmatrix} A_{(i-1,j-1)} & A_{(i-1,j)} & A_{(i-1,j+1)} \\ A_{(i,j-1)} & A_{(i,j)} & A_{(i,j+1)} \\ A_{(i+1,j-1)} & A_{(i+1,j)} & A_{(i+1,j+1)} \end{pmatrix}, \quad (22)$$

where: $\ominus$ – erosion operator, x – pixel coordinates, b – coordinates of non-zero elements of a structural element B.

On Fig. 1 shows a graphical algorithm for performing erosion for a binary image. Such an algorithm has the following description. For each pixel $A_{(i,j)}$ checks whether all pixels that

overlap the mask 3×3 equal 1. If at least one pixel in the neighborhood equals 0, then the result for this pixel is – 0. Only the central pixels of the object around which all pixels are the same and non-zero brightness remain 1.

The effect of applying erosion: removes small objects (noise); separates stuck objects; reduces the size of objects.

The 3×3 square is probably the most common structural element used in erosion operations, but others can be used. A larger structural element creates a more pronounced erosion effect, although very similar effects can usually be achieved by repeated erosions using a smaller structural element of similar shape. With larger structural elements, it is quite common to use a structural element that is roughly disc-shaped, as opposed to a square.

Erosions can be made directional by using less symmetrical structuring elements. For example, a structuring element that is 10 pixels wide and 1 pixel tall will erode only in the horizontal direction. Similarly, a 3×3 square structuring element with its origin in the middle of the top row rather than in the center will erode the bottom of the region more than the top.

Erosion of gray tones using a flat disk-shaped structural element usually darkens the image. Light areas surrounded by dark areas decrease in size, and dark areas surrounded by light areas increase in size. Small bright spots in the image will disappear as they fade to their surrounding intensity value, while small dark spots will become larger. The effect is most noticeable in areas of the image where the intensity changes rapidly, while areas of fairly uniform intensity will remain more or less unchanged except at their edges. Flat disk-shaped erosion cores cause small peaks in the image to disappear and valleys to widen.

REFERENCES

1. **Redmon, J., Divvala, S., Girshick, R., Farhadi, A**. (2015). You only look once: Unified, real-time object detection, [Online]. Available: https://arxiv.org/abs/1506.02640.
2. **Melin P., Gonzalez C., Castro J., Mendoza O., Castillo O**. (2014). "Edge-Detection Method for Image Processing Based on Generalized Type-2 Fuzzy Logic," IEEE Transactions on Fuzzy Systems, v. 22, 1515—1525.
3. **Tymchyshyn R., Volkov O., Gospodarchuk O., Bogachuk Yu.** (2018). Modern approaches to computer vision. Control Systems and Computers, No.6, 47-73. https://doi.org/10.15407/usim.2018.06.046.

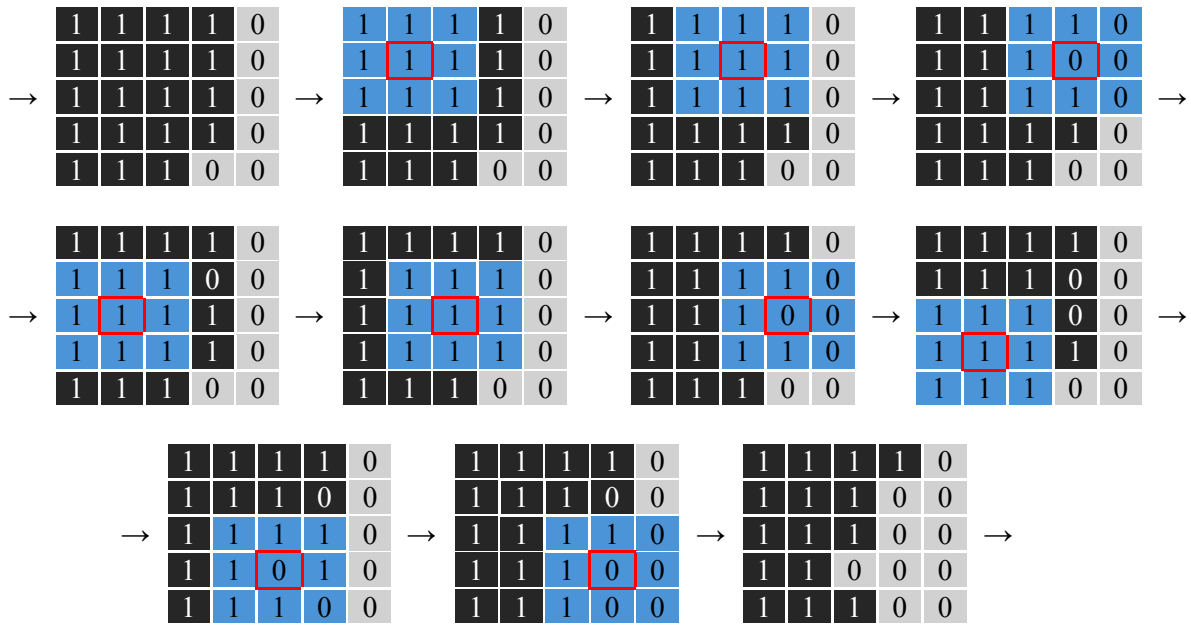

Fig. 1. Example of applying erosion the binary image

4. **Suriyababu V.K., Vuik C., Möller M.** (2023). Towards a High Quality Shrink Wrap Mesh Generation Algorithm Using Mathematical Morphology. Computer-Aided Design, Vol. 164, 103608, https://doi.org/10.1016/j.cad. 2023.103608.
5. **Courteaux M., Ramlot B., Lambert P., Van Wallendael G.** (2025). Lightweight Implicit Approximation of the Minkowski Sum of an N-Dimensional Ellipsoid and Hyperrectangle. Mathematics, 13(8), 1326. https://doi.org/10.3390/math13081326.
6. **Bernholt T., Eisenbrand F., Hofmeister T.** (2009). Constrained Minkowski sums: A geometric framework for solving interval problems in computational biology efficiently. Discret. Comput. Geom., 42, 22–36.
7. **Govindaraju N. K., Lin M. C., Manocha D.** (2004). Fast and reliable collision culling using graphics hardware. In Proceedings of the ACM Symposium on Virtual Reality Software and Technology, New York, NY, USA, 10–12 November, VRST'04, 2–9.
8. **Agarwal P. K., Flato E., Halperin D.** (2002). Polygon decomposition for efficient construction of Minkowski sums. Comput. Geom., 21, 39–61.
9. **Courteaux M., Mareen H., Ramlot B., Lambert P., Van Wallendael, G.** (2024). Dimensionality Reduction for the Real-Time Light-Field View Synthesis of Kernel-Based Models. Electronics, 13(20), 4062. https://doi.org/10.3390/ electronics13204062/
10. **Zerman E., Ozcinar C., Gao P., Smolic A.** (2020). Textured Mesh vs Coloured Point Cloud: A Subjective Study for Volumetric Video Compression. In Proceedings of the 2020 12th International Conference on Quality of Multimedia Experience, QoMEX, Athlone, Ireland, 26–28 May 2020.

Порівняльний аналіз алгоритмів визначення контурів на зображеннях в задачах комп'ютерного зору. Частина 1

Дмитро Міщук[1], *Євген Міщук*[2], *Дмитро Коржевін*[3]

[1,2]*Київський національний університет будівництва і архітектури*,
[3]*Cisco Talos Intelligence*

Анотація. Визначення контурів на зображеннях є однією з фундаментальних задач комп'ютерного зору, яка лежить в основі багатьох сучасних технологій. Їх точне виявлення дозволяє розв'язувати широкий спектр практичних завдань: від сегментації та розпізнавання об'єктів до 3D-реконструкції, медичної діагностики та автономного керування системами.

В галузі комп'ютерного зору досягнуто значного прогресу в створенні алгоритмів обробки зображень, проте залишається багато нерозв'язаних задач. Актуальними залишаються задачі з покращення швидкодії розв'язання існуючих алгоритмів, розробки алгоритмів для роботи з обмеженими обчислювальними ресурсами пам'яті.

В даній роботі пропонується опис і аналіз деяких відомих алгоритмів обробки зображень з визначення контурів на зображенні. Як відомо зображення, в комп'ютерних системах це набір пікселів заданої яскравості та кольору. Таке представлення можна відображати у виді матриць, де кожен її елемент – це піксель, а положення – це координати кожного пікселя. Таким чином обробка зображення з позиції математики – це робота з матрицями та множинами. Якість алгоритму обробки буде залежати від обраної методики роботи з матрицями.

Нейронні мережі, хоча і не є новим підходом, проте здобули широку популярність відносно нещодавно, головним чином завдяки розвитку згорткових нейронних мереж, які спеціалізуються на обробці фото- та відеоданих. Крім того, використання рекурентних нейронних мереж дозволяє здійснювати пошук оптимальної архітектури для нейромережевих систем. Різні моделі були адаптовані до специфічних обмежень, таких як розмір самої моделі, обчислювальні ресурси чи забезпечення необхідної точності.

Альтернативними підходами до нейронної мережі є алгоритми на основі детекторів та дескрипторів. По суті це алгоритми, які дозволяють порівняти зображення на основі деяких базових ознак. Наприклад, щоб зрозуміти, що на двох різних фото зображений один і той самий об'єкт, комп'ютер не порівнює кожен піксель, а шукає «особливі» місця і описує їх математично, що значно пришвидшує обробку зображення, проте може давати похибки.

Ключові слова: оператор Робертса, оператор Превітта, Лапласіан, ерозія, розмиття, морфологічні процеси, дилатація, нормалізація.