Algorytmy Przetwarzania Obrazów

Operacje na obrazach (II)

WYKŁAD 2
Dla studiów niestacjonarnych 2025/2026

Dr hab. Anna Korzyńska, prof. IBIB PAN

Operacje na obrazach

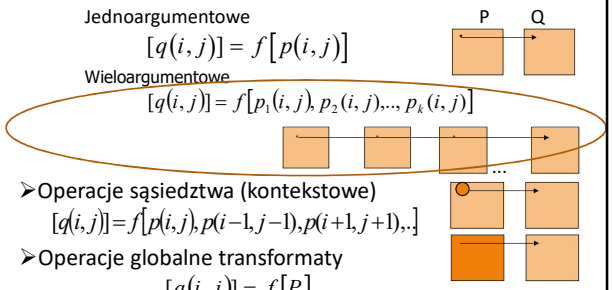
- Operacje punktowe (jednopunktowe):

Jednoargumentowe

$$[q(i,j)] = f[p(i,j)]$$

Wieloargumentowe

$$[q(i,j)] = f[p_1(i,j), p_2(i,j),.., p_k(i,j)]$$



- Operacje sąsiedztwa (kontekstowe)

$$[q(i,j)] = f[p(i,j), p(i-1,j-1), p(i+1,j+1),..]$$

- Operacje globalne transformaty

$$[q(i,j)] = f[P]$$

Laboratorium 2

Zadanie 2
Opracowanie algorytmu i uruchomienie funkcjonalności realizującej operacje logiczne na obrazach monochromatycznych i binarnych:

- not
- and
- or
- Xor

Uwagi:

- Proszę pamiętać o sprawdzeniu zgodności typów i rozmiarów obrazów stanowiących operandy.
- Proszę pamiętać: w operacjach jednopunktowych dwuargumentowych logicznych na obrazach działania prowadzone są na odpowiednich pikselach obrazów stanowiących argumenty danej operacji. W szczególności działania prowadzone są na bitach o tej samej wadze.
- Proszę o przygotowanie własnych monochromatycznych i binarnych obrazów testowych.
- Dodatkowo proszę umożliwić użytkownikowi zamianę obrazów z maski binarnej na maskę zapisaną na 8 bitach i na odwrót (jeśli w wybranym środowisku i języku jest to możliwe).

Operacje logiczne

Operacje logiczne

jednopunktowe dwuargumentowe

Poziom jasności n jest zapisany w kodzie dwójkowym jako kombinacja ośmiu 0 i 1:

Czerń 00000000
Biel 11111111
127 01000001

Operacje logiczne:

NOT NOT(1)=0; NOT(0)=1
AND 1 AND 1=1; 0 AND 0=0; 1 AND 0=0; 0 AND 1=0
OR 1 OR 1=1; 0 OR 0=0; 1 OR 0=1; 0 OR 1=1
XOR 1 XOR 1=0; 0 XOR 0=0; 1 XOR 0=1; 0 XOR 1=1

Algorytm powinien zawierać następujące kroki:

1. Sprawdzenie operandów
   1. Czy są obrazami jednokanałowymi
   2. Czy mają identyczny rozmiar w pionie i poziomie
2. Test wyniku obu powyższych warunków:
   1. W przypadku niezgodności komunikat o błędzie i koniec operacji
   2. W przypadku zgodności warunków
      1. przygotowanie tablicy na obraz wynikowy
      2. pętla przeszukujące oba operandy i obraz wynikowy (jako struktur2D lub jako wektor 1D - jeśli język na to pozwala)
         1. Wewnątrz **pętli po bajtach/wartościach** tablicy wybieramy **w pętli bit po bicie** lub działamy na całych 8-bitowych konstrukcjach - jeśli język na to pozwala
         2. Koniec operacji
      3. Zapamiętanie obrazu wynikowego
3. Wyświetlenie wyniku operacji jako komunikatu o błędzie lub w postaci obrazu

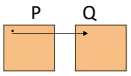
W logicznych dwuargumentowych operacjach jednopunktowych na obrazach działania prowadzone są na odpowiednich pikselach obrazów stanowiących argumenty danej operacji na bitach o tej samej wadze.

Operacje na obrazach

- Operacje punktowe (jednopunktowe):

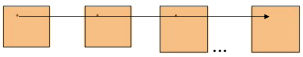
Jednoargumentowe

$$[q(i,j)] = f[p(i,j)]$$



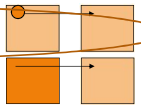
Wieloargumentowe

$$[q(i,j)] = f[p_1(i,j), p_2(i,j),.., p_k(i,j)]$$



- Operacje sąsiedztwa (kontekstowe)

$$[q(i,j)] = f[p(i,j), p(i-1,j-1), p(i+1,j+1),..]$$



- Operacje globalne transformaty

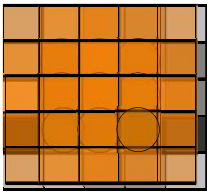
$$[q(i,j)] = f[P]$$

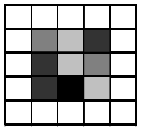
OPERACJE SĄSIEDZTWA

Są to operacje, w których na wartość zadanego piksela obrazu wynikowego ***q*** o współrzędnych *(i, j)* mają wpływ wartości pikseli **pewnego otoczenia** piksela obrazu pierwotnego ***p*** o współrzędnych *(i, j)*

8

Proces liczenia operacji sąsiedztwa





Do operacji sąsiedztwa F dla maski 3x3

Wynik operacji zależy od wielkości maski, ale głównie od funkcji zdefiniowanej na punkcie i jego otoczeniu.

```
For  i=2 to X-2 do:
        Begin.
         For  j=2 to X-2 do
                Begin.
                f new(i,j):= F( f(i-1, j-1), f(i-1, j),
                f(i-1, j+1), f(i, j-1), f(i,j) f(i, j+1),
                f(i+1, j-1), f(i+1, j), f(i+1, j+1))
                End.
        End.
```

Laboratorium 2

Operacja sąsiedztwa

Proszę dołączyć bibliotekę OpenCV i korzystać z niej przygotowując poszczególne funkcjonalności.

Zadanie 3 (4,5 p)

Opracowanie algorytmu i uruchomienie funkcjonalności realizującej operacje:

- wygładzania liniowego oparte na typowych maskach wygładzania (uśrednienie, uśrednienie z wagami, filtr gaussowski – przedstawione na wykładzie) przestawionych użytkownikowi jako maski do wyboru,
- wyostrzania liniowego oparte na 3 maskach laplasjanowych (podanych w wykładzie) przestawionych użytkownikowi maski do wyboru,
- kierunkowej detekcji krawędzi w oparciu o maski 8 kierunkowych masek Prewitta (podstawowe 8 kierunków) przestawionych użytkownikowi do wyboru,
- detekcji krawędzi operatorami opartymi na dwóch prostopadłych maskach Sobela.

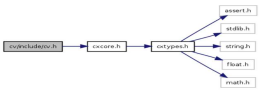
Biblioteka OpenCV

- **OpenCV** (*Open source computer vision*) to biblioteka funkcji przystosowanych do zastosowań w komputerowej wizji, czyli do wykorzystania przy tworzeniu oprogramowania pracującego w trybie czasu rzeczywistego.
- Powstała dla firmy **Intel**, przeszła w ręce firmy **Itseez,** którą następnie wchłonął **Intel**.
- Jest biblioteką darmową pod licencją **open-source BSD license**
- Zawiera ponad 500 algorytmów i około 5000 funkcji – bardzo efektywnych i szybkich
- Jest napisana w C++
- Obsługuje wiele języków programowania pod różnymi systemami operacyjnymi (C++, C#, Python, Java and MATLAB oraz może pracować pod Windows, Linux, Android i Mac OS.).

OpenCV

Implementacja liczenia histogramu w OpenC

```
#include <cv.h>
#include <highgui.h>

#pragma comment (lib, "cv210")
#pragma comment (lib, "highgui210")

IplImage* image= 0;
IplImage* imgHistogram = 0;
IplImage* gray= 0;

CvHistogram* hist;

int main(){

      image = cvLoadImage("xxxxx\a.jpg",1);

    //okreslenie rozmiaru histogramu (ilości koszy)
    int size[] = {256};

     //wartość kosza i jego znormalizowana wartość
    float val;
    int norm;

    //zakres jasności
    float range[] = { 0, 256 };
    float* ranges[] = { range };

       //wartość minimalna i maksymalna histogramu
       float min_val = 0, max_val = 0;

    //konwersja wejsciowego obrazu kolorowego do 8 bitowego w skali szarości
    gray = cvCreateImage( cvGetSize(image), 8, 1 );
    cvCvtColor( image, gray, CV_BGR2GRAY );

    //Tworzenie okien do pokazania rezultatu
```



Intel Processing Library

```
    cvNamedWindow("original",1);
    cvNamedWindow("gray",1);
    cvNamedWindow("histogram",1);

    //obrazy z których zostanie obliczony histogram
    IplImage* images[] = { gray };
                                    //Tworzenie okien do pokazania rezultatu
    // przygotowanie pustego histogramu
    hist = cvCreateHist( 1, size, CV_HIST_ARRAY, ranges,1);
       // obliczenie histogramu z obrazów
    cvCalcHist( images, hist, 0, NULL);
       // znalezienie minimalnej i maksymalnej wartości histo
    cvGetMinMaxHistValue( hist, &min_val, &max_val);

    // tworzenie 8bitowego obrazu do pokazania histogramu
    imgHistogram = cvCreateImage(cvSize(256, 50),8,1);
       // wypełnienie go białym kolorem
    cvRectangle(imgHistogram, cvPoint(0,0), cvPoint(256,50),
CV_RGB(255,255,255),CV_FILLED);

    //rysowanie histogramu jasności pikseli znormalizowanego od 0 do 50
    for(int i=0; i < 256; i++){
            val = cvQueryHistValue_1D( hist, i);
            norm = cvRound(val*50/max_val);
            cvLine(imgHistogram,cvPoint(i,50), cvPoint(i,50-norm),CV_RGB(0,0,0));
    }

    //pokazanie okien z rezultatem
    cvShowImage( "original", image );
    cvShowImage( "gray", gray );
    cvShowImage( "histogram", imgHistogram );
       // zapis obrazu histogramu do pliku
       cvSaveImage("histogray.jpg",gray);
    cvWaitKey();

    return 0;
}
```

```
 * @function EqualizeHist_Demo.cpp
 * @brief Demo code for equalizeHist function
 * @author OpenCV team

#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;
/**
 * @function main
 */
int main( int argc, char** argv )
{
  //! [Load image]
  CommandLineParser parser( argc, argv, "{@input | xxxxx| input image}" );
  Mat src = imread( samples::findFile( parser.get<String>( "@input" ) ),
     IMREAD_COLOR );
  if( src.empty() )
  {
    cout << "Could not open or find the image!\n" << endl;
    cout << "Usage: " << argv[0] << " <Input image>" << endl;
    return -1;
  }
```

```
//! [Load image]
  //! [Convert to grayscale]
  cvtColor( src, src, COLOR_BGR2GRAY );
  //! [Convert to grayscale]
  //! [Apply Histogram Equalization]
  Mat dst;
  equalizeHist( src, dst );
  //! [Apply Histogram Equalization]
  //! [Display results]
  imshow( "Source image", src );
  imshow( "Equalized Image", dst );
  //! [Display results]
  //! [Wait until user exits the program]
  waitKey();
  //! [Wait until user exits the program]
  return 0;
```

OpenCV has a function to do this, **cv2.equalizeHist()**

OpenCV dla Python

- Instalacja:
  - pip install **numpy** (niezbędne do działania openCV ze względu na obliczenia macierzowe obrazów)
  - pip install **opencv-python**
- Wczytanie:
  - import cv2 as cv
- Test:
  - print(cv.__version__)

OpenCV dla C++

- Opcja 1:
  - #include *"opencv2/core/core.hpp"*
  - cv::Mat H = cv::findHomography(points1, points2, CV_RANSAC, 5);
- Opcja 2:
  - #include *"opencv2/core/core.hpp"*
  - **using namespace** cv;
  - Mat H = findHomography(points1, points2, CV_RANSAC, 5);

OpenCV dla Java

- import org.opencv.core.Core;
- import org.opencv.core.CvType;
- import org.opencv.core.Mat;

- ulokowanie pliku opencv-300.jar w katalogu \opencv\build\java
- a biblioteki opencv_java3xx.dll library w katalgu: \opencv\build\java\x64 (64-bitowy system) lub \opencv\build\java\x86 (32-bitowy system).

OpenCV.js

- Ładowanie biblioteki:

```
<script src="opencv.js"
type="text/javascript"></script>
```

- Po załadowaniu jest gotowy do użycia:

```
imgElement.onload = function() {
      let mat = cv.imread(imgElement);
      cv.imshow('canvasOutput', mat);
      mat.delete();
};
```

Laboratorium 4

Zadanie 1

Opracowanie algorytmu i uruchomienie funkcjonalności realizującej operacje:

- wygładzania liniowego oparte na typowych maskach wygładzania (uśrednienie, uśrednienie z wagami, filtr gaussowski – przedstawione na wykładzie) przestawionych użytkownikowi jako maski do wyboru,

0	1	0
1	1	1
0	1	0

a	b	c
d	e	f
g	h	i

1	2	1
2	5	2
1	2	1

- wyostrzania liniowego oparte na 3 maskach laplasjanowych (podanych w wykładzie) przestawionych użytkownikowi maski do wyboru,

0	1	0
1	-4	1
0	1	0

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

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

- kierunkowej detekcji krawędzi w oparciu o maski 8 kierunkowych masek Prewitta (podstawowe 8 kierunków) przestawionych użytkownikowi do wyboru,
- detekcji krawędzi operatorami opartymi na maskach Sobela

Wybór sposobu uzupełnienie marginesów/brzegów w operacjach sąsiedztwa według zasady wybranej spośród następujących zasad:
wypełnienie ramki wybraną wartością stałą n narzuconą przez użytkownika: BORDER_CONSTANT
wyliczenie ramki według BORDER_REFLECT
wyliczenie ramki według BORDER_WRAP

Operacje wygładzania

w_1 x-1,y-1	w_2 x-1,y	w_3 x-1,y+1
w_4 x,y-1	w_5 x,y	w_6 x,y+1
w_7 x+1,y-1	w_8 x+1,y	w_9 x+1,y+1

y

x

Filtracja liniowa (metody *konwolucyjne*, tzn. uwzględniające pewne otoczenie przetwarzanego piksela):

$$g(x,y) = \sum_{k=1}^{n} w_k f_k(x,y)$$

n - liczba punktów (pikseli) otoczenia wraz z pikselem przetwarzanym
f(x,y) - wartość piksela o współrzędnych x,y obrazu pierwotnego
g(x,y) - wartość piksela o współrzędnych x,y obrazu wynikowego
w_k - waga k-tego piksela otoczenia

20

Wygładzanie

$$K = \frac{1}{ksize.width * ksize.height} \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}$$

Filtr jedynkowy (o równych wagach)
W openCV - normalized box filter

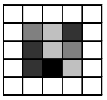
- Python:
  cv2.blur(src, ksize[, dst[, anchor[, borderType]]]) → dst
- C++
  void blur(InputArray **src**, OutputArray **dst**, Size **ksize**, Point **anchor**=Point(-1,-1), int **borderType**=BORDER_DEFAULT)
- JS
  cv.blur(src, dst, ksize, anchor, cv.BORDER_DEFAULT);

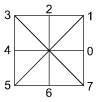
Argumenty

Argumenty wejściowe:

- src – obraz wejściowy
- ksize – jądro przekształcenia
- anchor – punkt zaczepienia jądra przekształcenia
- dst – obraz wyjściowy
- borderType – określa postępowanie z pikselami brzegowymi

Metody operacji na pikselach wchodzących w skład skrajnych kolumn i wierszy





1. Wartości pikseli są nieokreślone (xxxxxxxxxx)
2. Pozostawienie wartości pikseli bez zmian
3. Nadanie pikselom wartości arbitralnie zadanych przez użytkownika (np. same wartości „0”, „15”, „10” itd.
4. Operacje z zastosowaniem kolumn i wierszy pomocniczych (zdublowanie (powielenie) -skrajnych wierszy i kolumn)
5. Operacje z wykorzystaniem pikseli z **istniejącego** sąsiedztwa.
   - Lewa skrajna kolumna (oprócz pikseli górnego i dolnego rogu) – kierunki 0,1,2,6,7,
   - Lewa skrajna kolumna piksel w górnym rogu – kierunki 0, 6,7,
   - Lewa skrajna kolumna (piksel w dolnym rogu) – kierunki 0,1,2,
   - Prawa skrajna kolumna (oprócz pikseli górnego i dolnego rogu) – kierunki 2,3,4,5,6,
   - Prawa skrajna kolumna piksel w górnym rogu – kierunki 4,5,6,
   - Prawa skrajna kolumna (piksel w dolnym rogu) – kierunki 2,3,4,
   - Górny skrajny wiersz (oprócz pikseli z lewego i prawego rogu) – kierunki 4,5,6,7,0
   - Dolny skrajny wiersz (oprócz pikseli z lewego i prawego rogu) – kierunki 0,1,2,3,4.

Wartości brzegowe

- Wiele funkcji (klas) w OpenCV przyjmuje jako argument wejściowy zmienną określającą jak należy postępować z pikselami brzegowymi
- Typowo jest to parametr **borderType**

```
/* Various border types,
            image boundaries are denoted with '|'

* BORDER_REPLICATE: aaaaaa|abcdefgh|hhhhhhh
* BORDER_REFLECT: fedcba|abcdefgh|hgfedcb
* BORDER_REFLECT_101: gfedcb|abcdefgh|gfedcba
* BORDER_WRAP: cdefgh|abcdefgh|abcdefg
* BORDER_CONSTANT: iiiiii|abcdefgh|iiiiiii
                        with some specified 'i' */
```

Uniwersalny filtr konwolucyjny w OpenCV: filter2D()

```
void cv::filter2D      (InputArray   src,
                        OutputArray  dst,
                        int          ddepth,
                        InputArray   kernel,
                        Point        anchor = Point(-1,-1),
                        double       delta = 0,
                        int          borderType = BORDER_DEFAULT
                        )
Python:
dst  =  cv.filter2D(src, ddepth, kernel[, dst[, anchor[,
                    delta[, borderType]]]]
                   )
```

Operacje wyostrzania i detekcji krawędzi

Metoda: maska *filtracji górnoprzepustowej(FG)*.

W wyostrzaniu stosuje się metody numeryczne aproksymujące pochodną.

Zadanie wyostrzania:

- podkreślenie na obrazie konturów obiektów
- podkreślenie na obrazie punktów informatywnych (np. wierzchołki dla wielokątów, zakończenia, skrzyżowania, rozgałęzienia linii dla rysunków technicznych, wykresów lub pisma).

wyostrzania: wydobycie i uwypuklenie krawędzi obiektu.

26

Detekcja (wykrywanie) krawędzi (edge detection)
Jest to technika segmentacji obrazu, polegająca na znajdowaniu pikseli krawędziowych przez sprawdzanie ich sąsiedztwa.

Krawędź
Zbiór pikseli na krzywej mający taką właściwość, że piksele w ich sąsiedztwie, lecz po przeciwnych stronach krzywej mają różne poziomy jasności.

Cel detekcji
znalezienie lokalnych nieciągłości w poziomach jasności obrazu oraz granic obiektów zawartych w obrazie.

27

Cyfrowa wersja gradientu

Pochodna pionowa G_x funkcji f(x,y)

$$G_x \overset{def}{=} [f(x+1,y-1)+2f(x+1,y)+f(x+1,y+1)] - [f(x-1,y-1)+2f(x-1,y)+f(x-1,y+1)]$$

maska:

	y-1	*y*	*y+1*
x-1	-1	-2	-1
x	0	0	0
x+1	1	2	1

Pochodna pozioma G_y funkcji f(x,y)

$$G_y \overset{def}{=} [f(x-1,y+1)+2f(x,y+1)+f(x+1,y+1)] - [f(x-1,y-1)+2f(x,y-1)+f(x+1,y-1)]$$

$$G(x,y) = \sqrt{G_x^2 + G_y^2}$$

maska:

	y-1	*y*	*y+1*
x-1	-1	0	1
x	-2	0	2
x+1	-1	0	1

28

Cyfrowa wersja laplasjanu

$L(x, y) = [f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y)]$

maska:

	y-1	*y*	*y+1*
x-1	0	1	0
x	1	-4	1
x+1	0	1	0

Własności:
Gradient: wrażliwy na intensywność zmiany; używany tylko do detekcji krawędzi;
Laplasjan: podaje dodatkową informację o położeniu piksela względem krawędzi (po jasnej czy po ciemnej stronie).
Uwaga: Dla operacji wyostrzania współczynnik maski **K=1**

29

Laplasjany

Dyskretna forma drugiej pochodnej

0	-1	0		-1	-1	-1		1	-2	1		-1	-1	-1		0	-1	0
-1	4	-1		-1	8	-1		-2	4	-2		-1	9	-1		-1	5	-1
0	-1	0		-1	-1	-1		1	-2	1		-1	-1	-1		0	-1	0

30

Laplasjan/ogólny filtr $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

- Python:
  cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]]) → dst
- C++
  void Laplacian(InputArray **src**, OutputArray **dst**, int **ddepth**, int **ksize**=1, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)
- JS
  cv2.Laplacian(src, dst, cv.CV_8U, ksize, scale, 0, cv.BORDER_DEFAULT);

---

cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) → dst

Metody skalowania tablic obrazów wynikowych

Cel skalowania: sprowadzanie wartości pikseli do zakresu ***[0, (M-1)]***

Metoda proporcjonalna

$$g'(x,y) = \frac{g(x,y) - g(x,y)_{\min}}{g(x,y)_{\max} - g(x,y)_{\min}} \cdot (M-1)$$

Własność:
Równomierne przeskalowanie wszystkich pikseli obrazu.
Końcowy efekt: obraz z zakresu [0, (*M-1*)]

32

Metoda trójwartościowa

$$g'(x,y) = \begin{cases} 0 \text{ dla } g(x,y) < 0 \\ E[(M-1)/2] \text{ dla } g(x,y) = 0 \\ M-1 \text{ dla } g(x,y) > 0 \end{cases}$$

Zastosowanie
obrazy o jednolitym tle i dobrze widocznych obiektach - np. obrazy binarne. Efekt: czarno-biała krawędź na szarym tle.

Metoda obcinająca

$$g'(x,y) = \begin{cases} 0 \text{ dla } g(x,y) < 0 \\ g(x,y) \text{ dla } 0 \le g(x,y) \le M-1 \\ M-1 \text{ dla } g(x,y) > M-1 \end{cases}$$

33

Uniwersalny filtr konwolucyjny w OpenCV: filter2D()

```
void cv::filter2D   (InputArray   src,
                     OutputArray  dst,
                     int          ddepth,
                     InputArray   kernel,
                     Point        anchor = Point(-1,-1),
                     double       delta = 0,
                     int          borderType = BORDER_DEFAULT
                     )
```

Python:

```
dst  =  cv.filter2D(src, ddepth, kernel[, dst[, anchor[,
                    delta[, borderType]]]]
                  )
```

Ogólnie – filtr 2D i laplasian

Argumenty wejściowe: obraz, rozmiar otoczenia

- Python:
  cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) → dst
- C++
  void filter2D(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernel**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)
- JS
  cv.filter2D(src, dst, cv.CV_8U, cv.Mat.eye(3, 3, cv.CV_32FC1), anchor, 0, cv.BORDER_DEFAULT);

Argumenty – filtr 2D i laplasian

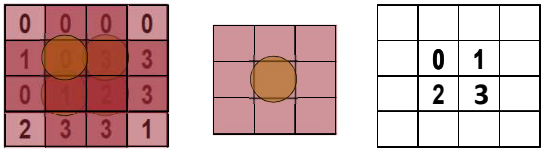
- Argumenty wejściowe:
  - src – obraz wejściowy
  - ddepth – głębokość obrazu wyjściowego (ddepth = -1 dla zachowania wartości z obrazu wejściowego)
  - kernel – jądro przekształcenia
  - anchor – punkt zaczepienia jądra przekształcenia
  - delta – (opcjonalnie) stała dodana do wartości obrazu
  - dst – obraz wyjściowy
  - borderType – określa postępowanie z pikselami brzegowymi
  - scale – (opcjonalne) określenie liczby przez którą mnożymy w celu przeskalownaia

Laboratorium 4

Zadanie 4 (1 p)

Opracowanie algorytmu i uruchomienie aplikacji realizującej uniwersalną operację medianową opartą na otoczeniu 3x3, 5x5, 7x7, (9x9?) zadawanym w sposób interaktywny (wybór z list, przesuwanie baru lub wpisanie w przygotowane pole). Zastosować opcjonalnie trzy metod operacji na brzegowych pikselach obrazu jak w zadaniu 3.

Mediana i pozostałe filtry statystyczne



1, 2, 2, 4, 5, 6, 7, 8, 9

0, 0, 0, 0, 0, 1, 1, 2, 3 mediana=0, min=0, max=3, najbardziej prawdopodobna=0

0, 0, 0, 0, 1, 2, 3, 3, 3 mediana=1, min=0, max=3, najbardziej prawdopodobna=0

0, 0, 1, 1, 2, 2, 3, 3, 3 mediana=2, min=0, max=3, najbardziej prawdopodobna=3

0, 1, 1, 2, 3, 3, 3, 3, 3 mediana=3, min=0, max=3, najbardziej prawdopodobna=3

38

Filtracja medianowa

scr – obraz źródłowy;
ksize – rozmiar otoczenia mediany
dst – obraz wynikowy

- Python:
  cv2.medianBlur(src, ksize[, dst]) → dst
- C++
  void medianBlur(InputArray **src**, OutputArray **dst**, int **ksize**)
- JS
  Cv.MedianBlur(src, dst, ksize);
- Java
  Imgproc.medianBlur(src, dst, 5);

Funkcja działa na obrazach monochromatycznych i kolorowych

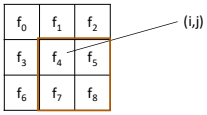
Metoda specjalnego gradientu

Stosowana w przypadkach, gdy metody filtracji górnoprzepustowej (FG) powodują wzmocnienie zakłóceń w obszarach leżących wewnątrz konturu.

Zasada

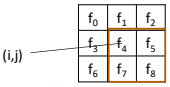
Krawędź uznana jest za istniejącą, jeśli wartość gradientu intensywności w pewnych punktach przekracza ustalony próg. Metody aproksymacji: **Robertsa**, **Sobela**, **Prewitta, Canny**.

Oznaczenia pikseli:



40

Metoda Robertsa



$$R(i,j) = \sqrt{(f_4 - f_8)^2 + (f_7 - f_5)^2};$$

$$\alpha = -\frac{\pi}{4} + tg^{-1}\left(\frac{f_7 - f_5}{f_4 - f_8}\right)$$

gdzie:
R(i,j) - specjalny gradient w punkcie (i,j)
α - kierunek gradientu intensywności.

Metoda Sobela

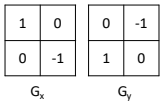
dwie składowe gradientu:

$$S_x = (f_2 + 2f_5 + f_8) - (f_0 + 2f_3 + f_6)$$
$$S_y = (f_6 + 2f_7 + f_8) - (f_0 + 2f_1 + f_2)$$
$$S(x,y) = \sqrt{S_x^2 + S_y^2}$$

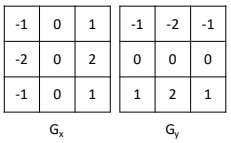
41

Maski konwolucyjne do konstrukcji operatorów krawędziowych

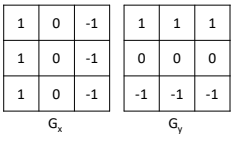
Roberts



Sobel:



Prewitt:



42

Operatory krawędziowania

Konstruowane przez nieliniową kombinacje dwóch prostopadłych kierunków gradientu (liniowych transformacji)

dokładnej $G = \sqrt{Gx^2 + Gy^2}$

przybliżonej $G = |Gx| + |Gy|$

Robertsa

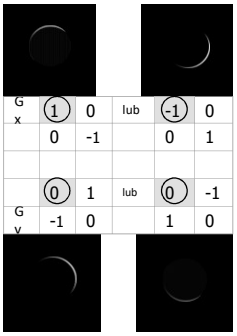
Gx	1	0	lub	-1	0
	0	-1		0	1
	0	1	lub	0	-1
Gy	-1	0		1	0

Sobela

	1	0	-1		-1	0	1
Gx	2	0	-2	lub	-2	0	2
	1	0	-1		-1	0	1
	1	2	1		-1	-2	-1
Gy	0	0	0	lub	0	0	0
	-1	-2	-1		1	2	1

43

Filtry i operatory Robertsa



Gx	1	0	lub	-1	0
	0	-1		0	1
	0	1	lub	0	-1
Gy	-1	0		1	0

$G = \sqrt{Gx^2 + Gy^2}$

44

Operator oparty na maskach Prewitta

Filtry kierunkowe Prewitta

Wschód			Południowy wschód			Południe			Południowy zachód		
-1	0	1	-1	-1	0	-1	-1	-1	0	-1	-1
-1	0	1	-1	0	1	0	0	0	1	0	-1
-1	0	1	0	1	1	1	1	1	1	1	0
E			SE			S			SW		

Dokładny $G = \sqrt{Gx^2 + Gy^2}$

Przybliżony $G = |Gx| + |Gy|$

W			NW			N			NE		
1	0	-1	1	1	0	1	1	1	0	1	1
1	0	-1	1	0	-1	0	0	0	-1	0	1
1	0	-1	0	-1	-1	-1	-1	-1	-1	-1	0
Zachód			Północny zachód			Północ (N)			Północny wschód		

Ten operator należy opracować indywidualnie

45

Filtry kierunkowe Sobela

Wschód			Południowy wschód			Południe			Południowy zachód		
-1	0	1	-2	-1	0	-1	-2	-1	0	-2	-1
-2	0	2	-1	0	1	0	0	0	1	0	-1
-1	0	1	0	1	2	1	2	1	1	2	0
E			SE			S			SW		

W			NW			N			NE		
1	0	-1	2	1	0	1	2	1	0	1	2
2	0	-2	1	0	-1	0	0	0	-1	0	1
1	0	-1	0	-1	-2	-1	-2	-1	-2	-1	0
Zachód			Północny zachód			Północ (N)			Północny wschód		

46

Filtry kierunkowe rzeźbiące

Wschód			Południowy wschód			Południe			Południowy zachód		
-1	0	1	-1	-1	0	-1	-1	-1	0	-1	-1
-1	1	1	-1	1	1	0	1	0	1	1	-1
-1	0	1	0	1	1	1	1	1	1	1	0
E			SE			S			SW		

W			NW			N			NE		
1	0	-1	1	1	0	1	1	1	0	1	1
1	1	-1	1	1	-1	0	1	0	-1	1	1
1	0	-1	0	-1	-1	-1	-1	-1	-1	-1	0
Zachód			Północny zachód			Północ (N)			Północny wschód		

47

Operator Sobela

```
void cv::Sobel    (InputArray    src,
                   OutputArray   dst,
                   int           ddepth,
                   int           dx,
                   int           dy,
                   int           ksize = 3,
                   double        scale = 1,
                   double        delta = 0,
                   int           borderType = BORDER_DEFAULT
                   )
```

Python:

```
dst=    cv.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[,
                 borderType]]]]]
                 )
```

Operator Sobela

dx – rząd pochodne po x
dy – rząd pochodne po y

Python:
```
dst=    cv.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[,
                 borderType]]]]]
                 )
```

C++
```
void cv::Sobel    (InputArray     src,
                   OutputArray    dst,
                   int        ddepth,
                   int        dx,
                   int        dy,
                   int        ksize = 3,
                   double     scale = 1,
                   double     delta = 0,
                   int        borderType = BORDER_DEFAULT
                   )
```

Laboratorium 2

Zadanie 4 (1p)

Opracowanie algorytmu i uruchomienie aplikacji realizującej uniwersalną operację medianową opartą na otoczeniu 3x3, 5x5, 7x7, (9x9) zadawanym w sposób interaktywny (wybór z list, przesuwanie baru). Zastosować powyższych metod uzupełniania brzegowych pikselach obrazu, dając użytkownikowi możliwość wyboru, jak w zadaniu 1.

Zadanie 5 (0,5)

Implementacji detekcji krawędzi operatorem Cannyego.

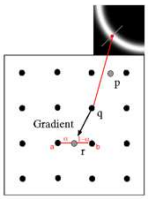
Operator Canny-ego

Wielostopniowy algorytm detekcji krawędzi zaproponowany przez twórcę teorii *Computational theory of edge detection* **John F. Canny** w 1986 r. Jego działanie oparte jest o znane detektory krawędzi (Robertsa, Sobela / Prewitta) i **progowanie i śledzenia z histerezą**, które optymalizuje wynik wyrzucając krawędzie rozmyte, nachylone pod kątami niewiele odchylonymi od wcześniej wykrytych, zapobiega przerwaniu krawędzi w miejscach utraty kontrastu, itp..

1986. *A computational approach to edge detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, 1986, pp. 679–698

Etapy działania operatora Canny-ego

- Zamiana obrazu kolorowego do monochromatycznego w odcieniach szarości
- Rozmycie filtrem gaussowskim o zadanym parametrze: sigma (δ)
- Obliczenie gradientu Sobele/Prewitta i kierunku gradientu według Robertsa
- Wytłumienie lokalnych nie maksymalnych gradientów w obrazie gradientu celem otrzymania cienkiej linii
  - Dokonanie podwójnego progowania według zadanych parametrów Tmin (i Tmax):
  - <Tmin - brak krawędzi,
  - >Tmax – silne krawędzie
  - >=Tmin i <=Tmax – krawędzie słabe
  - Wykonanie śledzenia krawędzi opartego na histerezie w celu ich uciąglenia: słabe krawędzi będące przedłużeniem silnych są dołączane, pozostałe są oznaczane do czyszczenia
- Wyczyszczenie krawędzi



Implementacje

Język C
```
void cv::Canny(InputArray  image,
               OutputArray edges,
               double      threshold1,
               double      threshold2,
               int         apertureSize = 3,
               bool        L2gradient = false
               )
```

- blur(src_gray, detected_edges, Size(3,3));
- Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size);
- argumenty:
  - *detected_edges*: Source image, grayscale
  - *detected_edges*: Output of the detector (can be the same as the input)
  - *lowThreshold*: The value entered by the user moving the Trackbar
  - *highThreshold*: Set in the program as three times the lower threshold (following Canny's recommendation)
  - *kernel_size*: We defined it to be 3 (the size of the Sobel kernel to be used internally)

Python
```
Edges = cv.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]    )
Edges = cv.Canny(dx, dy, threshold1, threshold2[, edges[, L2gradient]]            )
```

Java
```
cv.Canny(src, dst, 50, 100, 3, false);
```

Laboratorium 3

Zadanie 2.

Opracować algorytm i uruchomić funkcjonalność wykonywania podstawowych operacji morfologii matematycznej: erozji, dylacji, otwarcia i zamknięcia wykorzystując następujące elementy strukturalne 3x3 (tam gdzie możliwe jest rozróżnienie kształtu oddzielne dla krzyża i prostokąta).

Zadanie 3

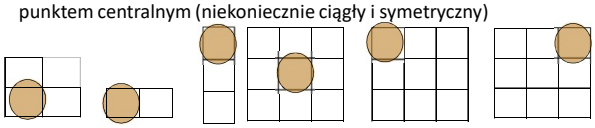
Opracować algorytm i uruchomić funkcjonalność wykonywania szkieletyzacji obiektu na mapie binarnej.

Operacje morfologii matematycznej na obrazach

Operacje pozwalające na budowanie złożonych operacji, pozwalających na analizę kształtu i wzajemnego położenia obiektów.

Fundamentalne pojęcie: **element strukturalnym (strukturujący)**

– podzbiór obrazu z wyróżnionym punktem, zwanym często punktem centralnym (niekoniecznie ciągły i symetryczny)



55

Operacje morfologii matematycznej na obrazach

– w elemencie strukturalnym występują następujące symbole:
- **1** element wskazuje piksel zapalony tzn. wartość obiektu w masce binarnej
- **0** element wskazuje piksel wytłumiony tzn. wartość tła w masce binarnej
- **X** element wskazuje dowolną wartość tzn. wartość tła lub obiektu w masce binarnej

Przekształcenia polegają na zmianie intensywności lub pozostawieniu intensywności punktu przykrytego przez punkt centralny elementu strukturalnego w zależności od spełnienia warunków logicznych.

Operacje morfologiczne przekształcają tylko część punktów obrazu

Morfologia matematyczna wymaga zdefiniowania konwencji zapisu obrazu binarnego:
- Biała kartka papieru
- Czarna tablica

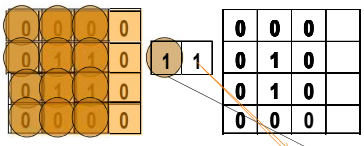
56

Operacje morfologiczne

1. Element strukturalny jest przemieszczany po wszystkich punktach obrazu tak, że punkt centralny elementu strukturalnego jest nakładany na kolejne punkty w kolejnych wierszach,
2. W każdym położeniu elementu sprawdza się, czy rzeczywista konfiguracja punktów jest zgodna (koincydentna) ze wzorcem zawartym w elemencie strukturalnym zakodowanym symbolami 1, 0, X
3. W przypadku wykrycia zgodności jest wykonywana operacja związana z filtrem, a w przeciwnym przypadku wartość występująca w obrazie pierwotnym jest przepisywana.

57

Operacje morfologiczne np. dla erozji



Jeśli punkt analizowanego otoczenia punktu centralnego jest wygaszony (równy wartości tła - 0) przy zapalonym (większym od tła - 1) elemencie centralnym, wynik zostaje wygaszany,

a w przeciwnym wypadku (tzn. punkt analizowanego otoczenia jest zapalony – 1) wygaszamy go

58

Podstawowe operacje morfologii matematycznej

0-zgaszony; 1-zapalony; X-o dowolnej wartości.

- Erozja

$$q(i,j) = \min_{i_n, j_m \in B(i,j)} (p(i_n, j_m))$$

1	1	1
1	1	1
1	1	1

- Dylatacja (dylacja) dualna do erozji

$$q(i,j) = \max_{i_n, j_m \in B(i,j)} (p(i_n, j_m))$$

X	X	X
X	0	X
X	X	X

B(i,j) element strukturalny z punktem centralnym o współrzędnych (i,j)

Dylatacja jest operacją dualną do erozji i na odwrót

59

Przykłady operacji erozji

O małym (2x2)/dużym(6x6) kwadratowym elemencie strukturalnym

O wertykalnym/horyzontalnym elemencie strukturalnym

60

Praca dyplomowa Szymona Mireckiego

OpenCV

erode() erode3x3()

```
GMat cv::gapi::erode  (
        const GMat & src,
        const Mat & kernel,
        const Point & anchor = Point(-1,-
                                     1),
        int iterations = 1,
        int borderType = BORDER_CONSTANT,

        const Scalar & borderValue = morphology
    DefaultBorderValue()
                    )
```

Przykłady operacji dylacji na obrazach w skali szarości

O małym (2x2)/dużym(6x6) kwadratowym elemencie strukturalnym

O wertykalnym/horyzontalnym elemencie strukturalnym

Maksimum / Rozjaśnianie

OpenCV

dilate() dilate3x3()

```
GMat cv::gapi::dilate(
        const GMat & src,
        const Mat & kernel,
        const Point & anchor = Point(-1,-1),
        int iterations = 1,
        int borderType = BORDER_CONSTANT,
        const Scalar & borderValue = morphologyDefaultBorderValue()

                )
```

Operacji erozji i dylacji działających na dowolnie zdefiniowanym elemencie centralnym

morphologyEx()

```
void cv::morphologyEx (
InputArray src,
OutputArray dst,
int op,
InputArray kernel,
Point anchor = Point(-1,-1),
int iterations = 1,
int borderType = BORDER_CONSTANT,
const Scalar & borderValue = morphologyDefaultBorderValue()
                )
```

Python:dst=cv.morphologyEx(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])

getStructuringElement()

```
Mat cv::getStructuringElement (
int shape,
Size ksize,
Point anchor = Point(-1,-1)
                )
```

Python:retval=cv.getStructuringElement(shape, ksize[, anchor])

```
enum  cv::MorphShapes {
  cv::MORPH_RECT = 0,
  cv::MORPH_CROSS = 1,
  cv::MORPH_ELLIPSE = 2
 }
```

shape of the structuring element More...

```
enum  cv::MorphTypes {
    cv::MORPH_ERODE = 0,
    cv::MORPH_DILATE = 1,
    cv::MORPH_OPEN = 2,
    cv::MORPH_CLOSE = 3,
    cv::MORPH_GRADIENT = 4,
    cv::MORPH_TOPHAT = 5,
    cv::MORPH_BLACKHAT = 6,
    cv::MORPH_HITMISS = 7
 }
```

type of morphological operation More...

Typy operacji

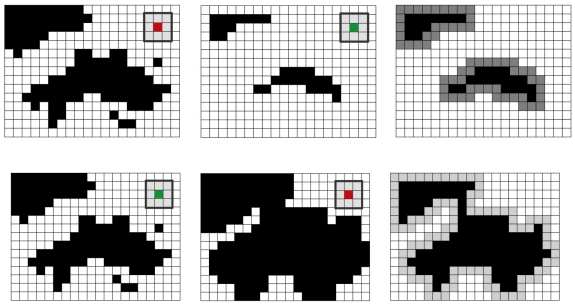
MORPH_ERODE Python: cv.MORPH_ERODE	see erode
MORPH_DILATE Python: cv.MORPH_DILATE	see dilate
MORPH_OPEN Python: cv.MORPH_OPEN	an opening operation dst=open(src,element)=dilate(erode(src,element))
MORPH_CLOSE Python: cv.MORPH_CLOSE	a closing operation dst=close(src,element)=erode(dilate(src,element))
MORPH_GRADIENT Python: cv.MORPH_GRADIENT	a morphological gradient dst=morph_grad(src,element)=dilate(src,element)−erode(src,element)
MORPH_TOPHAT Python: cv.MORPH_TOPHAT	"top hat" dst=tophat(src,element)=src−open(src,element)
MORPH_BLACKHAT Python: cv.MORPH_BLACKHAT	"black hat" dst=blackhat(src,element)=close(src,element)−src
MORPH_HITMISS Python: cv.MORPH_HITMISS	"hit or miss" .- Only supported for CV_8UC1 binary images. A tutorial can be found in the documentation

Inne operacje morfologii matematycznej

- **Otwarcie (=Erozja+Dylacja)**
- **Zamknięcie[domknięcie] (=Dylacja+Erozja)**
- Detekcja ekstremów Top Hat (=Zamknięcie-Obraz=Obraz-Otwarcie)
- Gradient morfologiczny (= Otwarcie+Zamknięcie)
- Wygładzanie morfologiczne (=Dylacja–Erozja)
- Pocienianie
- Pogrubianie
- Szkieletyzacja (znalezienie szkieletu czyli punktów obiektu równoodległych od jej brzegów)
- Odcinanie gałęzi (artefaktów z nieregularności obiektów szkeletyzowanych)
- Detekcja centroidów (punktów centralnych obiektu)
- Dylatacja bez styków (SKIZ ang. Skeleton by influece zone)
- Erozja warunkowa
- Rekonstrukcja (wygłodzanie obszru, czyszczenie brzegów, zalewanie dziur)
- Automediana

67

Otwarcie i zamknięcie

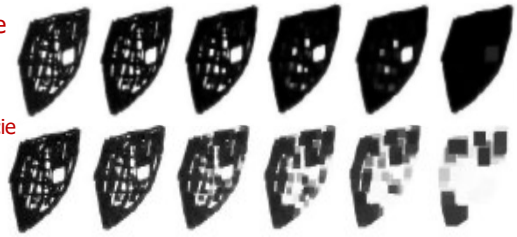


Odwrócona mapa – nie pokazywać

Operacje otwarcia i zamknięcia na obrazach szaroodcieniowych

Otwarcie

Zamknięcie



Element strukturalny: 3x3 5x5 9x9 13x13 21x21

Komplementarność operacji

Praca dyplomowa Szymona Mireckiego

69

Operacje morfologiczne w projektach

- Operacji erozji i dylacji działających na dowolnie zdefiniowanym elemencie centralnym
- Operacje erozji warunkowej i dylacji warunkowej
- Rekonstrukcja morfologiczne przez erozję/dylację
- Operacji wyliczenia transformaty odległościowej
- Ekstrakcja linii pionowych i poziomych za pomocą operacji morfologicznych
- Otoczenia wypukłe

Działamy na obrazach binarnych chyba że ktoś sam rozszerzy na obrazy monochromatyczne w szarych odcieniach

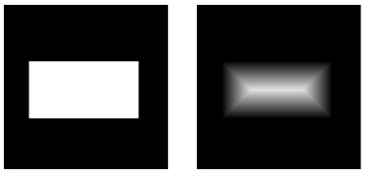
Operacje erozji warunkowej i dylacji warunkowej

- **warunek erozji** – obiekt nie może zniknąć;
  Jeśli erozja prowadzi do zniknięcia obiektu to jej wynikiem jest obraz wejściowy.

- **warunek dylacji** - obiekt nie może połączyć się z innym obiektem;
  Jeśli dylacjia prowadzi do połączenia z innym obiektem to wynikiem dylacji jest obraz wejściowy

Konieczność śledzenia liczby obiektów w poszczególnych etapach realizacji operacji

Operacji wyliczenia euklidesowej transformaty odległościowej

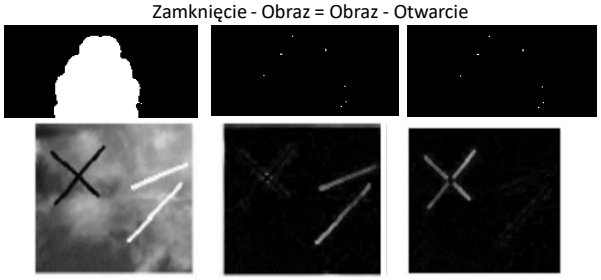


Obraz transformaty odległościowej obiektu, to obraz, w którym każdy piksel zawiera odległość euklidesową do najbliższego piksela brzegu obiektu

Wykonywać kolejne erozje obiektu, aż do jego zniknięcia, zapisując różnicę między obrazem wejściowym i wyjściowym dla każdego etapu. Rozliczyć poziomy jasności na kolejne etapy i wygenerować obraz wynikowy

Top Hat

Zamknięcie - Obraz = Obraz - Otwarcie



White Top Hat

Black Top Hat

cv::MORPH_TOPHAT = 5,

73

Rekonstrukcja przez otwarcie

Otwarcie (maska)

Rekonstrukcja przez otwarcie

Rekonstrukcja polega na cyklicznym dokonywaniu dylatacji maski i wyznaczeniu części wspólnej z obrazu uzyskanego po dylatacji i obrazu wyjściowego przekształcenia (wykonuje się operację logiczną AND z obrazem wyjściowym. W ten sposób usuwa się te fragmenty, które zostały dodane podczas dylatacji a faktycznie będące poza odtwarzaną figurą).

74

Ekstrakcja linii pionowych i poziomych za pomocą operacji morfologicznych

Operacje morfologiczne do detekcji pionowych i poziomych linii powinny mieć dobrane odpowiednie elementy strukturalne umożliwiające erozję i dylację pozwalająca wyodrębnić takie linie:

1	1	1	1	1

1
1
1
1
1

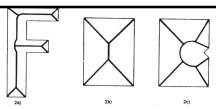
Otoczenie wypukłe

Wypukłe otoczenie figury to najmniejsza figura wypukłą zawierającą daną figurę.

1	x	x
1	x	1
1	1	x

x	1	x
1	x	0
x	1	x

Algorytmy szkeletyzacji



Umożliwiają upraszczanie obiektów na obrazach prowadząc do zastąpienia obiektu jego szkieletem, który odzwierciedla podstawowe topologiczne własności obiektów.

Jego analiza może zostać wykorzystana do:

- klasyfikacji figur ze względu na kształt,
- wyznaczania orientacji figur podłużnych,
- określania linii środkowej szerszych linii,
- rozdzielanie złączonych obiektów.

Ścienianie jest potrzebne, aby odtworzyć liniową strukturę obrazu wejściowego nie niszcząc jego spójności.

Matematyczna definicja szkieletu obiektu na płaszczyźnie ciągłej:

DEFINICJA 1: Niech R będzie zbiorem punktów na płaszczyźnie, B jego brzegiem, a P punktem należącym do R. Najbliższym sąsiadem punktu P na brzegu B jest punkt M należący do B taki, że nie istnieje inny punkt należący do B, którego odległość od punktu P jest mniejsza od odległości PM. Jeżeli punkt P ma więcej niż jednego najbliższego sąsiada, to P nazywamy punktem szkieletowym zbioru R. Zbiór wszystkich punktów szkieletowych jest szkieletem lub osią środkową zbioru R.

Szkielet figury, to zbiór wszystkich punktów równoodległych od co najmniej dwóch brzegów

DEFINICJA 2.

Szkieletem zbioru R elementów obrazu cyfrowego jest zbiór wyznaczony w następujący sposób.

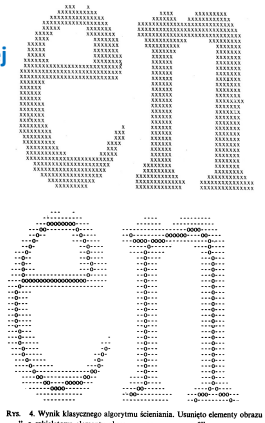
W zbiorze R określa się:

- potencjalnie szkieletowe (otoczone punktami obiektu) lub szkieletowe (stanowiące krzywą lub prostą ciągnącą się w dowolnym kierunku)

oraz

- konturowe (w otoczeniu jest poziom jasności tła „0") elementy obrazu.

Następnie usuwa się wszystkie konturowe elementy obrazu, które nie są szkieletowymi i z tak otrzymanym zbiorem R rekurencyjnie powtarzamy procedurę aż do uzyskania zbioru zawierającego jedynie szkieletowe elementy obrazu.



Rys. 4. Wynik klasycznego algorytmu ścieniania. Usunięto elementy obrazu oznaczone „–", a szkieletowe elementy obrazu oznaczono przez „0"

Wyznaczanie szkieletu binarnego polega na wielokrotnym stosowaniu (często naprzemiennych, z różnymi elementami strukturalnymi) operacji pocieniania – do momentu, aż kolejne operacje nie wpływają na wygląd obrazu wynikowego. W tym celu można stosować różne zestawy elementów strukturalnych. Przykładem adekwatnego zestawu jest 8 elementów otrzymanych w wyniku obrotów następujących elementów strukturalnych:

$$\begin{bmatrix} 0 & 0 & 0 \\ z & 1 & z \\ 1 & 1 & 1 \end{bmatrix} \text{ oraz } \begin{bmatrix} z & 0 & 0 \\ 1 & 1 & 0 \\ z & 1 & z \end{bmatrix} \text{ o kąty } 0^\circ, 90^\circ, 180^\circ \text{ i } 270^\circ.$$

Algorytm 1 (klasyczny algorytm ścieniania):

Oznaczenia: I oznacza obraz wejściowy. P oznacza zbiór wzorców sąsiedztwa szkieletowych elementów obrazu wraz z obróconym o 90° pierwszym wzorcem i obróconym o 90°, 180° i 270° drugim wzorcem. Znacznik *remain* z wartością *true* wskazuje, że nieszkieletowe elementy obrazu mogą pozostać. Znacznik *skel* z wartością *true* wskazuje, że sąsiedztwo elementu obrazu odpowiada jednemu ze wzorców zbioru P. Jedynka/zero we wzorcu odpowiada niezerowemu/zerowemu elementowi w sąsiedztwie.

1. Podstaw *true* jako wartość znacznika *remain* .
2. **While** *remain* = *true* **do** kroki 3-12.
   Begin
   3. Podstaw *false* jako wartość znacznika *remain* . {nie dokonano żadnej zmiany}
   4. **For** j = 0,2,4 i 6 **do** kroki 5-12.
      Begin
      5. **For** dla wszystkich elementów p obrazu I **do** kroki 6-10.
         Begin
         6. **If** p = 1 **and if** jego j-sąsiad = 0 **then do** kroki 7-10.
            Begin
            7. Podstaw *false* jako wartość znacznika *skel.*
            8. **For** wszystkich wzorców P **do** krok 9.
               Begin
               9. **If** *sąsiedztwo* p odpowiada wzorcowi P **then** podstaw *true* jako wartość *skel* i wyjdź z pętli
                  End.
               10. **If** *skel* =*true* **then** podstaw 2 jako wartość p {szkieletowy element obrazu) **else** podstaw 3 jako wartość p {usuwamy element obrazu}i podstaw *true* jako wartość *remain.*
                  End.
            End.
         11. **For** wszystkich elementów p obrazu I **do** krok 12.
            Begin
            12. **If** p = 3, **then** podstaw jako p wartość 0.
            End.
         End.
      End.
13. Koniec algorytmu.

$$\begin{bmatrix} 0 & 0 & 0 \\ z & 1 & z \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} z & 0 & 0 \\ 1 & 1 & 0 \\ z & 1 & z \end{bmatrix}$$

$0^\circ, 90^\circ, 180^\circ$ i 270°

a)

	A	A	A		A	A	A	
	0	P	0		A	P	0	
	B	B	B		A	0	2	

b)

A	A	C
0	2	2+
B	B	C

Rys. 3. Wzorce sąsiedztwa powtarzalnych elementów obrazu. **a) Co najmniej jeden z każdej grupy elementów obrazu oznaczonych** przez **A lub B musi być niezerowy. b) Co najmniej jeden z elementów obrazu oznaczony przez C musi być niezerowy. Jeżeli obydwa elementy obrazu oznaczone przez C są niezerowe, to wartości elementów obrazu oznaczonych przez A i B mogą być dowolne. W przeciwnym przypadku co najmniej jeden z elementów każdej pary oznaczonej przez A lub B musi być niezerowy**

Mini-projekty egzaminacyjne

nr.	Tytuł projektu dla APOZ 2025/2026
1	Udoskonalenie oprogramowania przygotowanego na zajęciach przez dołączanie oprogramowania do segmentacji obiektów na podstawie wskazanego na wszystkich kanałach zakresu intensywności oraz zliczania wysegmentowanych obiektów z powstałej mapy binarnej.
2	Udoskonalenie oprogramowania przygotowanego na zajęciach przez implementacje funkcji detekcja całej twarzy i oczu za pomocą kaskadowego klasyfikatora opartego na cechach Haara
3	Udoskonalenie oprogramowania przygotowanego na zajęciach przez: dołożenia operacji erozji i dylacji działających na dowolnie zdefiniowanym elemencie struturyzującym – proszę zaprojektować i wykonać edytor elementu struturyzyjącego.
4	Udoskonalenie oprogramowania przygotowanego na zajęciach przez przygotowanie funkcji liczenie FFT i iFFT oraz edytora manipulacji widmem amplitudowym.
5	Segmentacja obrazu monochromatycznego/binarnego zawierającego nuty i rozpoznawanie nut.
6	Program prezentacji zasad przebiegu procesu wprowadzania i korekcji zniekształceń radiometrycznych z wykorzystaniem obrazów szaro odcieniowych oraz ich fragmentów w postaci obrazowej a także tablic liczb.
7	Udoskonalenie oprogramowania przygotowanego na zajęciach przez przygotowanie funkcji wyznaczającej kontur obiektu w binarnym obrazie: porównanie rozwiązań opartych na metodach morfologii matematycznej i konwolucyjnych operacji konturowania.
8	Udoskonalenie oprogramowania przygotowanego na zajęciach przez przygotowanie funkcji zmieniającej obraz według przekształcenia opisanego przemieszczeniem trzech nie współliniowych punktów.
9	Udoskonalenie oprogramowania przygotowanego na zajęciach przez przygotowanie funkcji przycięcia (kadrowanie) obrazu według prostokąta niekoniecznie równoległego do osi X i Y wyznaczonego odpowiednim narzędziem wskazywania położenia.
10	Przeniesienie oprogramowania stworzonego na zajęciach tak, aby działał dla systemu operacyjnego iOS.

11	**Oprogramowanie to wykonania transformaty Hough'a**
12	Oprogramowanie do nakładania zniekształceń geometrycznych na obrazy monochromatyczne i edytora doboru funkcji korygujących takie zniekształcenie.
13	Udoskonalenie oprogramowania przygotowanego na zajęciach przez: dołożenie operacji wyliczenia transformaty odległościowej dla obrazu binarnego oraz implementację rozdzielenia obiektów dotykających się z wykorzystaniem tej transformaty.
14	Segmentacja obrazu monochromatycznego/binarnego zawierającego znaki specjalne i symbole oraz wyszukiwanie znaków o kształtach wklęsłych np.: ☏, Ξ, ★, ♠, ♨, £, itp..
15	Segmentacja z obrazów monochromatycznych/binarnych zawierających wybrane 5 typów symboli o różnym rozmiarze, ich rozpoznawania metodami klasycznymi lub opartymi o sieci neuronowe.
16	Segmentacja obrazów z wykorzystaniem klasteryzacji szarych odcieni.
17	Udoskonalenie oprogramowania przygotowanego na zajęciach przez implementację progowanie obrazu prawdopodobieństwa przypasowania do zadanej tekstury.
18	Udoskonalenie oprogramowania przygotowanego na zajęciach przez implementację nowego narzędzie do tworzenia panoramy na postawie serii zdjęć.
19	Implementacja linii profilu i wycinanie nie będących prostokątem fragmentów z istniejących obrazów monochromatycznych.
20	Udoskonalenie oprogramowania przygotowanego na zajęciach przez wykonanie narzędzia do ekstrakcja linii pionowych i poziomych za pomocą operacji morfologicznych.
21	Udoskonalenie oprogramowania przygotowanego na zajęciach przez przygotowanie funkcji wykonywania wyrównania histogramu obrazu kolorowego zapisanego z wykorzystaniem modelu L*a*b*.
22	Udoskonalenie oprogramowania przygotowanego na zajęciach przez: -implementacja operacji filtracji logicznych na obrazach binarnych; -rozwinięcie możliwości wyświetlania i zapisywania jako obraz fragmentów obrazu powstałych na podstawie wskazanych, niekoniecznie spójnych, fragmentów histogramu.

23	**Segmentacja obrazu monochromatycznego zawierających drobne obiekty metalowe o różnych kształtach zarejestrowane na taśmie produkcyjnej przez kamerę monochromatyczną.**
24	Program prezentacji sposobu działania metody α-NN (α najbliższych sąsiadów) z wizualizacją przestrzeni cech przed i po fazie uczenia.
25	Udoskonalenie oprogramowania przygotowanego na zajęciach przez implementację operacji przenikania dwóch obrazów monochromatycznych.
26	Udoskonalenie oprogramowania przygotowanego na zajęciach przez wykonanie narzędzia do rozciągania i zawężania histogramy z możliwym zastosowaniem funkcji liniowej lub funkcji gamma (analogicznie jak w Corelu PhotoPaint-cie).
27	Udoskonalenie oprogramowania przygotowanego na zajęciach przez dołożenie operacji otoczki wypukłej obiektu – operacja morfologii matematycznej
28	Segmentacja obrazu monochromatycznego/binarnego zawierającego znaki specjalne i symbole oraz wyszukiwanie znaków o kształtach wypukłych np.:Θ, θ, ◎, ◘, ◒, ◑, itp..
29	Udoskonalenie oprogramowania przygotowanego na zajęciach przez dołożenie operacji tworzenia histogramu dwuwymiarowego z obrazu monochromatycznego i jego matematycznego przekształcenia. Implementacja narzędzia do segmentacji na podstawie histogramu dwuwymiarowego.
30	Udoskonalenie oprogramowania przygotowanego na zajęciach przez: zaimplementowanie operacji rekonstrukcji morfologicznej.
31	Udoskonalenie oprogramowania przygotowanego na zajęciach przez dołożenie operacji zmniejszenia udziału szumu przez uśredniania kilku obrazów zebranych w takich samych warunkach oraz operacji logicznych na zaszumionych obrazach binarnych.

Pomoc - przykłady

- https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html
- https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.htmlhttps://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html
- https://docs.opencv.org/3.4/d4/dbd/tutorial_filter_2d.htmlhttps://docs.opencv.org/3.4/d4/dbd/tutorial_filter_2d.html
- https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.htmlhttps://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
- https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.htmlhttps://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html
- https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.htmlhttps://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

Materiał:

- Materiały wykładowe POBD z zeszłego roku na UBIKu
- T.Pavlidis, Grafika i Przetwarzanie Obrazów, WNT Warszawa 1987.
- **I.Pitas,** Digital image processing, algorithms and applications, John Wiley &Sons, Inc. 2000, (UBI **w katalogu *APOD\Pitas***).