Sergiu NEDEVSCHI Tiberiu MARIȚA
Radu DĂNESCU Florin ONIGA
Raluca BREHAR Ion GIOSAN
Cristian VANCEA Robert VARGA

PROCESAREA IMAGINILOR

Îndrumător de laborator
Ediția a 2-a

UTPRESS
Cluj-Napoca, 2023
ISBN 978-606-737-625-8

Sergiu NEDEVSCHI **Tiberiu MARIȚA**
Radu DĂNESCU **Florin ONIGA**
Raluca BREHAR **Ion GIOSAN**
Cristian VANCEA **Robert VARGA**

PROCESAREA IMAGINILOR

Îndrumător de laborator

Ediția a 2-a

UTPRESS
Cluj-Napoca, 2023
ISBN 978-606-737-625-8

Editura UTPRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.: 0264-401.999
e-mail: utpress@biblio.utcluj.ro
www.utcluj.ro/editura

Director: ing. Dan COLȚEA

Recenzia: Prof.dr.ing. Dorian Gorgan
Prof.dr.ing. Vasile Dădârlat

Pregătire format electronic on-line: Gabriela Groza

ISBN 978-606-737-625-8

Cuprins

Prefață

Îndrumătorul de laborator reprezintă materialul de bază pentru pregătirea lucrărilor practice la disciplina Procesarea Imaginilor și este destinat studenților de anul III, ciclul de licență, domeniul Calculatoare și Tehnologia Informației din cadrul Facultății de Automatică și Calculatoare. De asemenea îndrumătorul poate fi util și oricui dorește să studieze o serie de metode tradiționale de procesare a imaginilor.

Capitolele sunt organizate didactic, pe subiecte prezentate tehnic, cu grad de dificultate gradual, având la bază un formalism teoretic cu rol de fundamentare a cunoștințelor în vederea unei finalități practice specifice fiecărui subiect. Se recomandă parcurgerea capitolelor în ordinea de prezentare în cadrul îndrumătorului, având în vedere faptul că unele capitole conțin elemente din cele anterioare.

În prezenta ediție sunt studiate o serie de tehnici uzuale în activitatea de cercetare întreprinsă în cadrul Centrului de Cercetare Procesare de Imagine și Recunoașterea Formelor (IPPRRC) din cadrul Departamentului de Calculatoare. Din experiența acumulată în cadrul mai multor teme de cercetare specifice domeniului procesării de imagini s-au selectat mai multe tematici de bază cu utilizare largă în ceea ce privește gradul de aplicabilitate și cu o abordare facilă atât din punct de vedere al formalismului teoretic cât și din punct de vedere practic printr-o fundamentare bazată pe utilizarea unei aplicații cadru dezvoltată folosind librăria OpenCV. Totodată, autorii le mulțumesc colegilor pentru observațiile constructive, care au dus la îmbunătățirea conținutului acestui îndrumător.

În prezentarea lucrărilor s-a urmărit o succesiune constructivă a informațiilor prezentate. Fiecare capitol conține o introducere în care se prezintă obiectivele urmărite continuate de o scurtă prezentare a conceptelor teoretice necesare, care stau la baza aspectelor practice ale implementării. Activitățile practice ce trebuie îndeplinite sunt prezentate la sfârșitul capitolelor. În repetate rânduri sunt expuse mai multe soluții pentru implementarea aceluiași obiectiv, iar varietatea acestora permite efectuarea unei analize comparative specifice activităților de cercetare. Pentru familiarizarea cu temele prezentate se recomandă parcurgerea capitolelor curente înainte de participarea la activitățile de laborator.

Autorii vă urează lectură plăcută!

1. Introducere în utilizarea bibliotecii OpenCV

1.1. Introducere

Scopul acestei lucrări de laborator este de a familiariza studenții cu aplicația cadru care va fi folosită în activitatea practică la disciplina Procesarea Imaginilor.

Cunoștințele necesare pentru a putea parcurge acest laborator sunt:

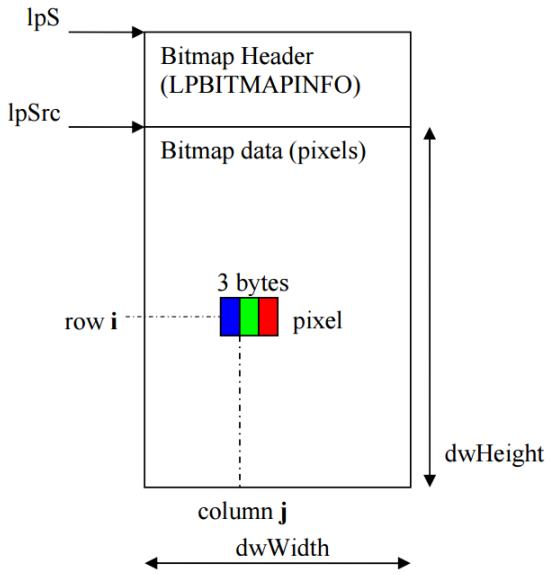
- Obligatorii: Programarea în C++, Structuri de date și algoritmi
- Opționale: *Metode orientate pe obiect, Algoritmi fundamentali, Algebră liniară, Matematici discrete, C++, Visual C++ (Visual Studio)*

1.2. Formatul de imagine Bitmap

Formatul "bmp" este folosit pentru a stoca imagini în formă necomprimată. Acest format stochează imaginile digitale în format rastru (matrice), independent de dispozitivul de afișare, și poate stoca imagini monocrome sau color cu mai multe nivele de adâncime a culorii. Nivelul de adâncime a culorii determină numărul de culori ale unei imagini și dimensiunea memoriei necesare pentru a stoca această imagine. Fișierul "bmp" are următoarea structură:

- Un antet (header) al fișierului bitmap – conține semnătura tipului bmp, dimensiunea fișierului și punctul de început al șirului de pixeli;
- Antetul DIB – conține informații precum dimensiunea imaginii (înălțime, lățime) și numărul de biți per pixel;
- Tabelul culorilor (sau tabelul look-up) – pentru imagini color care folosesc paletă;
- Șirul de pixeli – imaginea propriu zisă, ca un șir unidimensional de valori, ce poate include și valori de umplutură pentru alinierea datelor.

Următoarea imagine ilustrează formatul bitmap pentru o imagine de 24 de biți. Dimensiunile imaginii, înălțime și lățime, sunt notate cu dwHeight și dwWidth.



1.3. Vedere de ansamblu asupra mediului OpenCV

Mediul în care veți lucra conține biblioteca OpenCV împreună cu Visual Studio. Setările sunt preconfigurate și toate bibliotecile statice și dinamice sunt incluse în soluție.

Sarcina voastră este de a crea funcții noi care vor fi apelate din funcția principală (main). Aceste funcții trebuie grupate în funcție de ședințele de laborator la care veți participa și trebuie să primească nume sugestive. Toate exemplele de cod presupun că ați inclus namespace-ul cv (*using namespace cv*), altfel toate clasele și metodele OpenCV trebuie prefixate cu **`cv::`** . Următorul fragment de cod vă indică ce trebuie să faceți pentru a introduce noi funcții (textul pe fond gri indică ce trebuie să adăugați în plus):

```
void negative_image(){
	// implement function
}
int main(){
	int op;
	do{
		printf("Menu:\n");
		// ...
		printf(" 7 - L1 Negative Image \n");
		// ...
		printf(" 0 - Exit\n\n");
		printf("Option: ");
		scanf("%d",&op);
		switch (op)
		{
			//...
			case 7:
				negative_image();
				break;
		}
	}
	while (op!=0);
	return 0;
}
```

Trebuie să salvați ce ați lucrat la fiecare ședință de laborator. Dimensiunea proiectului poate fi redusă apelând scriptul clean.bat, care șterge toate fișierele generate de compilator. O altă metodă este de a salva doar fișierul cpp principal, deoarece restul fișierelor proiectului nu se schimbă.

1.4. Clasa Mat

În OpenCV imaginile sunt stocate în memorie ca obiecte ale clasei Mat. Această clasă este o clasă pentru manipularea matricelor generice bidimensionale sau a matricelor cu mai multe dimensiuni.

Câmpurile importante ale clasei Mat sunt:

- rows – numărul de rânduri ale matricei = înălțimea imaginii;
- cols – numărul de coloane ale matricei = lățimea imaginii;
- data – un pointer la locația din memorie a pixelilor imaginii

Cea mai simplă și curată metodă de a crea un obiect de tip Mat este de a apela constructorul cu trei parametri:

```
Mat img(rows, cols, type);
```

Ultimul parametru indică tipul de date stocat în matrice. Un exemplu de tip este CV_8UC1, care reprezintă: 8 biți, fără semn, un singur canal. În general primul număr după CV_ reprezintă numărul de biți, litera reprezintă tipul, iar Cx indică numărul de canale pentru un pixel.

Tipurile de date esențiale sunt prezentate în următorul tabel:

Codul type	**Tipul de date**	**Utilizare**
CV_8UC1	unsigned char	Imagine monocromă (8bits/pixel)
CV_8UC3	Vec3b	Imagine color (3x8bits/pixel)
CV_16SC1	short	Stocare date
CV_32FC1	float	Stocare date
CV_64FC1	double	Stocare date

Exemplu 1 – crearea unei imagini monocrome de dimensiune 256x256:

```
Mat img(256,256,CV_8UC1);
```

Exemplu 2 – crearea unei imagini color cu 720 rânduri și 1280 coloane:

```
Mat img(720,1280,CV_8UC3);
```

Exemplu 3 – crearea unei matrice cu numere reale de dimensiune 2x2, care conține valorile: [1 2; 3 4] și afișarea acesteia:

```
float vals[4] = {1, 2, 3, 4};
Mat M(2,2,CV_32FC1,vals); // constructor cu 4 parametri
std::cout << M << std::endl;
```

După cum se poate observa, un obiect de tip Mat se poate afișa folosind stream-ul de ieșire standard.

Pentru o descriere detaliată a clasei Mat se poate consulta documentația oficială: https://docs.opencv.org/4.5.1/d3/d63/classcv_1_1Mat.html

1.5. Citirea unei imagini din fișier

Pentru a deschide un fișier imagine și a stoca conținutul acestuia într-un obiect de tip Mat, se folosește funcția imread:

```
Mat img = imread("path_to_image", flag);
```

Primul parametru este calea relativă sau absolută spre fișierul imagine; al doilea parametru, flag, poate fi:

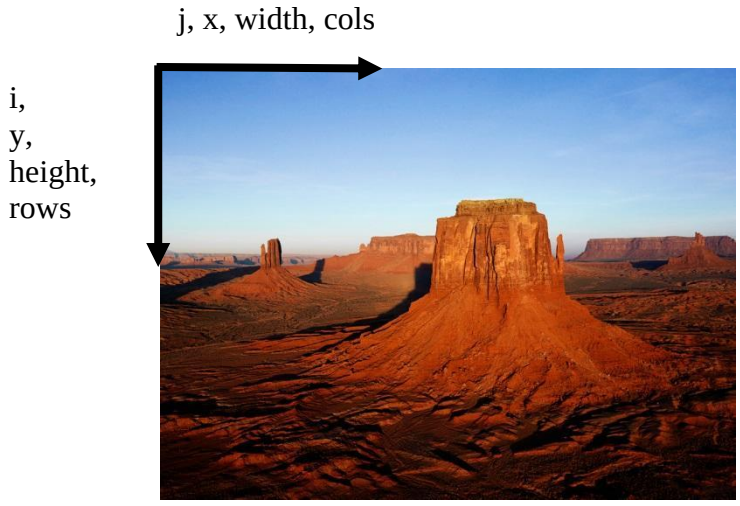
- IMREAD_UNCHANGED (-1) – deschide imaginea în același format în care este salvată pe disc;
- IMREAD_GRAYSCALE (0) – deschide imaginea ca imagine monocromă (grayscale); încărcarea face conversia la CV_8UC1 (1 canal, unsigned char);
- IMREAD_COLOR (1) – se încarcă imaginea și se convertește la CV_8UC3 (3 canale unsigned char);

Exemplu 1 – se deschide o imagine din directorul curent în formatul în care a fost salvată:

```
Mat img = imread("cameraman.bmp", -1);
```

1.6. Accesarea datelor din imagine

Elementele din matrice sunt indexate conform convenției standard din matematică. Acest lucru înseamnă că originea va fi în colțul stânga sus al imaginii, primul parametru va indica rândul (crescător în jos), iar al doilea parametru va indica coloana (crescător spre dreapta). Următoarea figură ilustrează schema de indexare:



Respectați această convenție întotdeauna, pentru a evita erorile. Când parcurgeți o imagine folosind două cicluri for, ciclul exterior va parcurge rândurile, iar cel interior coloanele.

Pentru a accesa pixelul de la coordonatele (i, j) dintr-o imagine monocromă img, se va folosi metoda *at*:

```
unsigned char pixel = img.at<unsigned char>(i,j);
```

După cum se poate observa, trebuie să specificați tipul de date care sunt stocate în matrice (unsigned char).

Pentru acces mai rapid se poate utiliza direct pointerul la datele matricei:

```
unsigned char pixel = img.data[i*img.step[0] + j];
```

Toți pixelii sunt stocați într-un șir unidimensional, rând după rând, și pixelii de pe rând ordonați de la stânga la dreapta, începând cu adresa indicată de pointerul `img.data`. Pe un rând pot exista mai mulți pixeli decât numărul de coloane, deci **accesarea pixelilor folosind `i*img.cols+j` poate duce la rezultate greșite**!

Se poate obține un pointer individual pentru rândul i:

```
unsigned char pixel = img.ptr(i)[j];
```

Pentru a accesa cele trei componente de culoare a unui pixel de pe rândul i și coloana j a unei imagini color, trebuie folosit tipul dedicat Vec3b:

```
Vec3b pixel = img.at< Vec3b>(i,j);
unsigned char B = pixel[0];
unsigned char G = pixel[1];
unsigned char R = pixel[2];
```

Vec3b este un vector cu trei componente de tip byte (unsigned char) și este tipul recomandat pentru manipularea imaginilor color.

Codul poate fi simplificat prin utilizarea clasei cu șablon Mat_<T> , subclasă a clasei Mat, care permite omiterea tipului pentru operațiile de acces. La crearea unui obiect Mat_<T> trebuie să specificați tipul care este stocat în matrice.

```
Mat_<uchar> img = imread("fname", IMREAD_GRAYSCALE);
uchar pixel = img(i,j);
```

În exemplul de mai sus am folosit specificatorul de tip uchar, care este echivalent cu unsigned char. Accesarea unei valori (pixel) de la o anumită poziție permite și citirea, și scrierea acesteia.

1.7. Vizualizarea unei imagini

Pentru a vizualiza o imagine se folosește funcția imshow, urmată de un apel al funcției waitKey:

```
imshow("image", img);
waitKey(0);
```

Aceste linii de cod vor afișa imaginea într-o fereastră nouă și apoi programul va aștepta ca utilizatorul să apese o tastă. Funcția waitKey are un singur parametru: cât de mult să aștepte după acțiunea utilizatorului (în milisecunde). Valoarea zero înseamnă că sistemul va aștepta un timp infinit.

Folosiți întotdeauna waitKey după imshow. Ferestrele de tip imagine pot fi mutate sau redimensionate, pentru a facilita procesul de analiză a rezultatelor procesării.

1.8. Salvarea/scrierea unei imagini pe disc

Pentru a salva o imagine, folosiți funcția imwrite:

```
imwrite("fname", img);
```

Numele fișierului trebuie să conțină calea (directorul), numele și extensia, care va determina formatul în care se va face scrierea.

1.9. Funcție exemplu

Următoarea funcție încarcă o imagine monocromă și o transformă în negativul ei:

```
void negative_image(){
	Mat img = imread("Images/cameraman.bmp",
				IMREAD_GRAYSCALE);
	for(int i=0; i<img.rows; i++){
		for(int j=0; j<img.cols; j++){
			img.at<uchar>(i,j) = 255 - img.at<uchar>(i,j);
		}
	}
	imshow("negative image",img);
	waitKey(0);
}
```

Fișierul imagine trebuie să fie localizat în directorul Images, inclus în directorul proiectului aplicației cadru.

1.10. Activități practice

1. Descărcați și compilați aplicația *OpenCVApplication*.
2. Testați funcția negative_image().
3. Implementați o funcție care schimbă nivelele de gri cu un factor aditiv.
4. Implementați o funcție care schimbă nivelele de gri cu un factor multiplicativ. Salvați imaginea rezultat.
5. Creați o imagine color de dimensiune 256 x 256. Împărțiți imaginea în 4 cadrane egale și colorați acestea, din stânga-sus până în dreapta-jos, astfel: alb, roșu, verde, galben.
6. Creați o matrice 3x3 de tip float, determinați inversa ei și tipăriți-o.
7. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

1.11. Bibliografie

https://docs.opencv.org/4.5.1/index.html

2. Spații de culoare

2.1. Introducere

Scopul acestui al doilea laborator este învățarea tehnicilor de bază în lucrul cu culorile imaginilor digitale în format bitmap.

2.2. Spațiul RGB

Culoarea fiecărui pixel (atât pentru echipamentele de achiziție – camere) cât și pentru afișare (TV, CRT, LCD) se obține prin combinația a trei culori primare: roșu, verde și albastru. (**R**ed, **G**reen și **B**lue) (spațiu de culoare aditiv – Fig. 2.1 și 2.2).

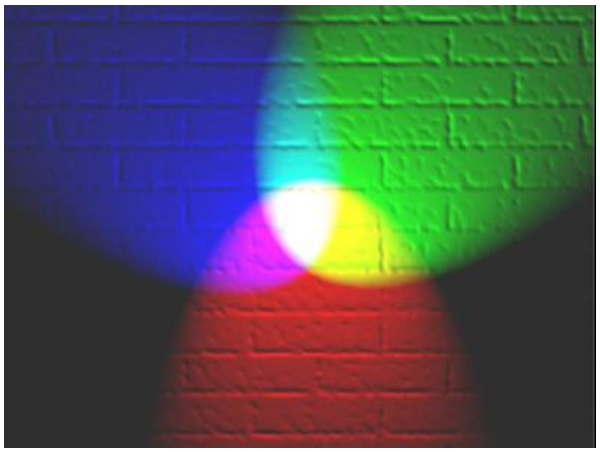


Fig. 2.1 Reprezentarea combinării aditive a culorilor. Acolo unde culorile primare se suprapun se observă apariția culorilor secundare. Acolo unde toate trei culorile se suprapun se observă apariția culorii albe [1]

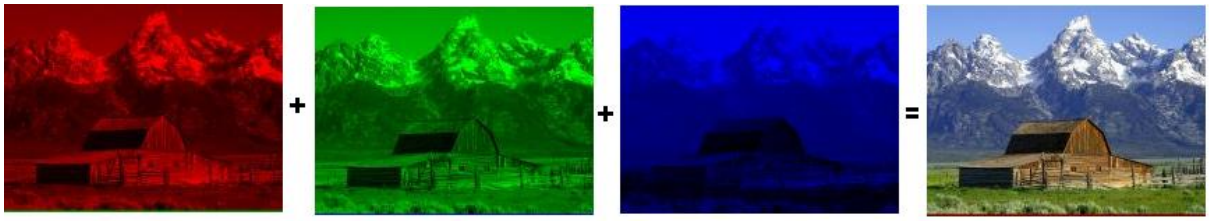


Fig. 2.2 Imaginea color se obține prin combinarea la nivel de pixel a celor trei culori primare (vezi cele trei canale)

Astfel, fiecare pixel din imagine va fi caracterizat prin câte o valoare pentru fiecare din cele trei componente de culoare primare. Culoarea sa reprezintă un punct în spațiul 3D al modelului de culoare RGB (Fig. 2.3). În acest cub al culorilor, originea axelor R, G și B corespunde *culorii negre* (0, 0, 0). Vârful opus al cubului corespunde *culorii albe* (255, 255, 255). Diagonala cubului, între negru și alb corespunde tonurilor de gri (grayscale) (R=G=B). Trei dintre vârfuri corespund culorilor primare roșu, verde și albastru. Celelalte 3 vârfuri corespund culorilor complementare: turcoaz, mov și galben (**C**yan, **M**agenta and **Y**ellow). Dacă translatăm originea sistemului de coordonate în punctul „alb” și redenumim cele 3 axe de coordonate ale sistemului în C, M, Y obținem spațiul de culoare complementar CMY (folosit la dispozitive de imprimare color).

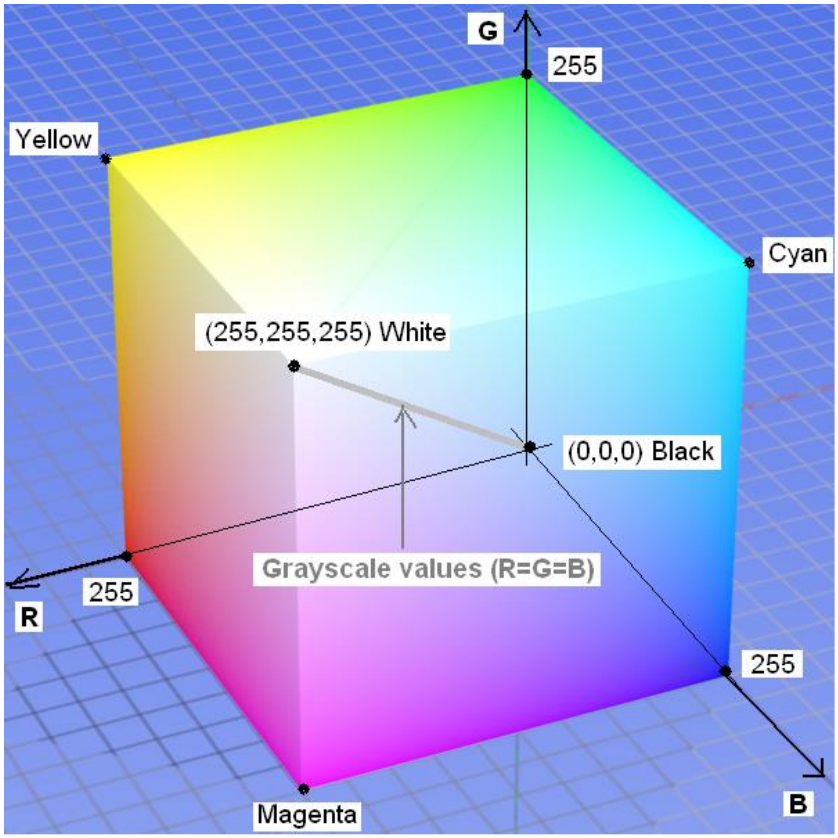


Fig. 2.3 Modelul de culoare RGB mapat pe un cub. În acest exemplu fiecare culoare este reprezentată pe câte 8 biți (256 de niveluri) rezultând imagini bitmap RGB24. Numărul total de culori este 2^8x2^8x2^8 = 2^{24} = 16.777.216

Pentru imagini RGB24 (24 biți/pixel) spațiul de culoare poate fi reprezentat complet. Într-o imagine indexată (cu paletă) poate fi reprezentat doar un anumit subspațiu al spațiului de culoare din Fig. 2.3. În acest context, numărul de biți/pixel (numărul de biți folosiți pentru codificarea unei culori) se numește „adâncime de culoare" (color depth) și variantele sun prezentate în Tabelul 2.1:

Tabel 2.1 Adâncimea și tipul imaginii

Adâncimea de culoare	Nr. Culori	Mod de culoare	Palette (LUT)
1 bit	2	Indexed Color	Yes
4 biți	16	Indexed Color	Yes
8 biți	256	Indexed Color	Yes
16 biți	65536	True Color	No
24 biți	16.777.216	True Color	No
32 biți	16.777.216	True Color	No

Există și alte modele de culoare [2], care nu vor fi discutate aici.

2.3. Conversia unei imagini color într-o imagine grayscale

Pentru a converti o imagine color într-o imagine grayscale, cele trei componente ale culorii fiecărui pixel trebuie egalizate. O metodă des folosită este medierea celor 3 componente:

$$R_{Dst} = G_{Dst} = B_{Dst} = \frac{R_{Src} + G_{Src} + B_{Src}}{3} \tag{2.1}$$

2.4. Conversia imaginilor grayscale în imagini binare (alb-negru)

O imagine binară (alb-negru) este o imagine care conține doar două culori: alb și negru. O imagine binară se obține dintr-o imagine grayscale printr-o operație simplă numită binarizare cu prag (thresholding). Binarizarea cu prag este cea mai simplă tehnică de segmentare a imaginilor, care permite separarea obiectelor de background (Fig. 2.4).

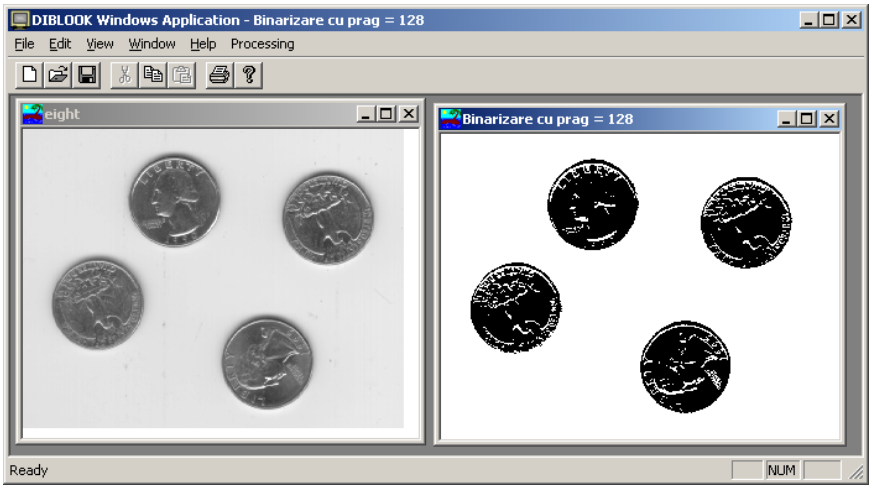


Fig. 2.4 Binarizarea cu prag

În acest laborator va fi discutată binarizarea cu prag fix (ales arbitrar) pentru imagini indexate (8 biți/pixel) de tip grayscale. Binarizarea poate fi aplicată prin parcurgerea valorilor pixelilor din imaginea sursă și înlocuirea lor în imaginea destinație cu valoarea dată de:

$$Dst(i,j) = \begin{cases} 0 \quad (black) & , \quad if \quad Src(i,j) < threshold \\ 255 \quad (white) & , \quad if \quad Src(i,j) \geq threshold \end{cases} \tag{2.2}$$

2.5. Spațiul de culoare HSV (Hue Saturation Value)

Este un spațiu / model de culoare invariant (componentele H (culoare) și S (saturație) sunt separate și cvasi-independente de iluminarea scenei caracterizată de V (intensitate)). Se reprezintă sub forma unei piramide cu baza hexagonală sau a unui con

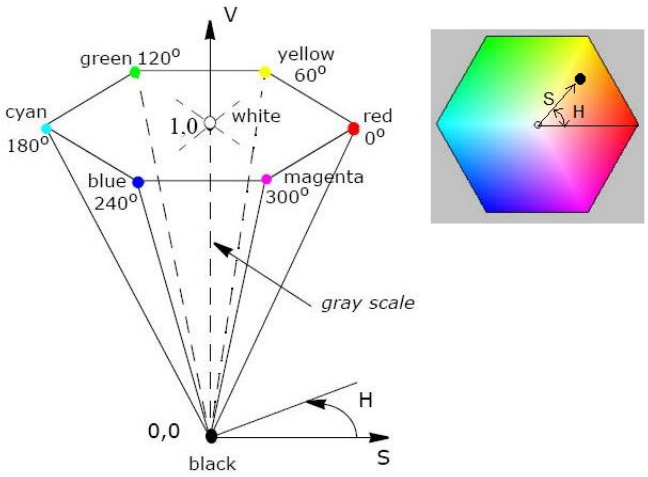


Fig. 2.5 Modelul (spațiul) de culoare HSV

unde:

H – reprezintă unghiul făcut de culoarea curentă cu raza corespunzătoare culorii Roșu
S – reprezintă distanta culorii curente fata de centrul bazei piramidei/conului
V – reprezintă înălțimea culorii curente în piramidă/con

2.6. Transformarea RGB → HSV

Ecuațiile de transformare din componentele RGB în HSV sunt [3]:

```
r = R/255;  // r : componenta R normalizată
g = G/255; // g : componenta G normalizată
b = B/255; // b : componenta B normalizată
// Atenție declarați toate variabilele pe care le folosiți de tip float
// Dacă ați declarat R  de tip uchar, trebuie să faceți cast: r = (float)R/255 !!!

M = max (r, g, b); // Atenție: în Visual C există predefinit un macro pentru min() și max()
m = min (r, g, b);  // care primește 2 parametri. Cu 3 parametri nu dă eroare la compilare,
                    // dar rezultatul este incorect. Rescrieți în forma corectă cu 2 parametri.
C = M - m;
```

Value:

```
V = M;
```

Saturation:

```
If (V!=0)
        S = C / V;
Else // negru
        S = 0;
```

Hue:

```
If (C!=0) {
        if (M == r) H = 60 * (g - b) / C;
        if (M == g) H = 120 + 60 * (b - r) / C;
        if (M == b) H = 240 + 60 * (r - g) / C;
        }
Else  // grayscale
        H = 0;
If (H < 0)
        H = H + 360;
```

Valorile pt. H, S și V calculate cu formulele de mai sus vor avea următoarele domenii de valori:

H = 0 .. 360
S = 0 .. 1
V = 0 .. 1

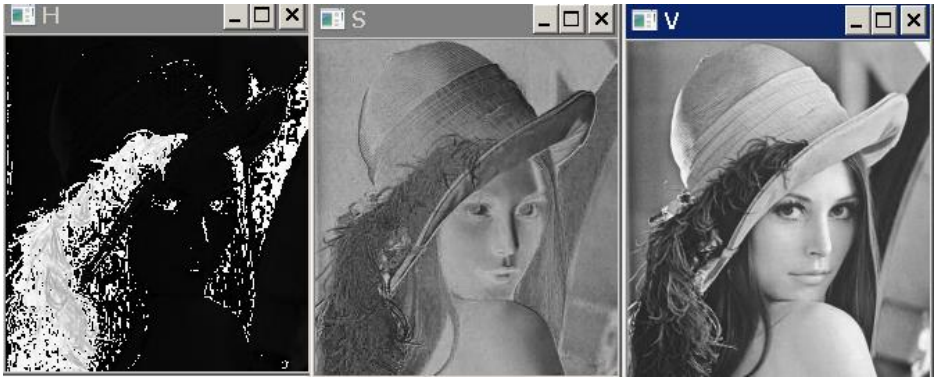
Aceste valori se normalizează (scalează) în intervalul 0 .. 255 pentru a reprezenta fiecare componentă de culoare ca și o imagine cu 8 biți/pixel (de tip CV_8UC1):

H_norm = H*255/360
S_norm = S*255
V_norm = V*255

2.7. Activități practice

1. Adăugați la framework o funcție care copiază canalele R, G, B ale unei imagini RGB24 (tip CV_8UC3) în trei matrice de tip CV_8UC1. Afișați aceste matrice în 3 ferestre diferite.
2. Adăugați la framework o funcție de conversie de la o imagine color RGB24 (tip CV_8UC3) la o imagine grayscale de tip (CV_8UC1) și afișați imaginea rezultat într-o fereastră destinație.
3. Adăugați la framework o funcție de procesare pentru conversia de la *grayscale* la *alb-negru* pentru imagini grayscale (CV_8UC1), folosind (2.2). Citiți valoarea pragului de la linia de comandă. Testați operația de binarizare folosind diverse imagini și diverse praguri.
4. Adăugați la framework o funcție care convertește canalele R, G, B ale unei imagini RGB24 (tip CV_8UC3) în componente H, S, V folosind ecuațiile din 2.6. Memorați componente H, S, V în câte o matrice de tip CV_8UC1 corespunzătoare canalelor H, S, V. Afișați aceste matrice în 3 ferestre diferite. Verificați corectitudinea implementării prin comparație vizuală cu rezultatele de mai jos.
5. Implementați o funcție *isInside(img, i, j)* care verifică dacă poziția indicată de perechea *(i,j)* (rând, coloană) este înăuntrul imaginii *img*.
6. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

a) Rezultate obținute pe imaginea *flowers_24bits.bmp* (24 bits/pixel)



b) Rezultate obținute pe imaginea *Lena_24bits.bmp* (24 bits/pixel)

Fig. 2.6 Exemple ale conversiei imaginilor din spațiul de culoare RGB în HSV

2.8. Bibliografie

[1] http://en.wikipedia.org/wiki/RGB_color_model
[2] http://en.wikipedia.org/wiki/Color_models
[3] Open Computer vision Library, Reference guide, cvtColor() function, https://docs.opencv.org/4.5.1/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab

3. Histograma nivelurilor de intensitate

3.1. Introducere

În această lucrare se vor prezenta conceptul de histogramă a nivelurilor de gri și un algoritm pentru a diviza această histogramă în mai multe zone în scopul reducerii numărului de niveluri de gri (cuantizare în spațiul culorilor).

3.2. Histograma nivelurilor de intensitate

Având o imagine în niveluri de gri cu nivelul maxim de intensitate L (pentru o imagine cu 8 biți/pixel $L=255$), histograma nivelurilor de intensitate (gri) se definește ca o funcție $h(g)$ care are ca valoare, numărul de pixeli din imagine (sau dintr-o regiune) care au intensitatea $g \in [0 \dots L]$:

$$h(g) = N_g \tag{3.1}$$

N_g - numărul de pixeli din imagine sau din regiunea de interes (ROI) care au intensitatea g.

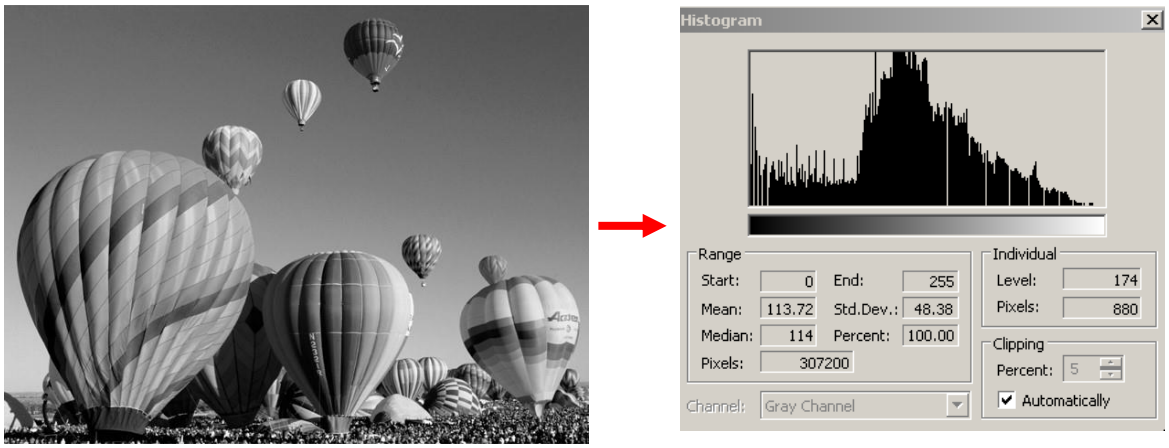


Fig. 3.1 Exemplu: Histograma unei imagini grayscale

Histograma normalizată cu numărul de pixeli din imagine (din ROI) se numește funcția densității de probabilitate[1] (FDP) a nivelurilor de intensitate:

$$p(g) = \frac{h(g)}{M} \tag{3.2}$$

unde:

$M = image\_height \times image\_width$

FDP are următoarele proprietăți:

$$\begin{cases} p(g) >= 0 \\ \int_{-\infty}^{\infty} p(g)dg = 1, \quad \sum_{g=0}^{L} \frac{h(g)}{M} = \frac{M}{M} = 1 \end{cases} \tag{3.3}$$

[1]Având domeniul de definiție discret în statistică se numește funcție masă de probabilitate. Totuși se folosește termenul de funcție densitate de probabilitate corespunzător domeniului continuu datorită utilizării frecvente în literatura de specialitate.

3.3. Aplicație: Determinarea pragurilor multiple

Acest algoritm determină praguri multiple în scopul reducerii numărului de niveluri de intensitate (gri) a unei imagini.

Primul pas constă în determinarea maximelor din histogramă. Apoi fiecare nivel de gri este asignat la cel mai apropiat maxim.

Pentru determinarea maximelor din histogramă se urmăresc pașii:

1. Se normalizează histograma (se transformă într-o FDP)
2. Se alege o fereastră de lățime *2*WH+1* (o valoare bună pentru WH este 5)
3. Se alege un prag *TH* (o valoare bună este 0.0003)
4. Se „suprapune" fereastra peste histogramă. Pentru fiecare poziție *k* (a mijlocului ferestrei) de la 0+*WH* până la *255-WH*
   - Se calculează media *v* a valorilor din histograma normalizată în intervalul [*k-WH, k+WH*]. Obs: valoarea *v* este media a *2*WH+1* valori
   - Dacă *FDP*[*k*]>*v+TH* și *FDP*[*k*] este mai mare sau egal ca toate valorile *FDP* din intervalul [*k-WH, k+WH*] atunci *k* corespunde la un maxim din histogramă. Se memorează și se continuă din poziția următoare.
5. Se inserează 0 la începutul listei de poziții de maxime și 255 la sfârșit. Aceasta permite culorilor negru, respectiv alb, să fie reprezentate exact.

Al doilea pas constă în determinarea efectivă a pragurilor. Acestea sunt situate la distanțe egale între maximele determinate anterior. Algoritmul de reducere a nivelurilor de gri este unul simplu: se asignează fiecare pixel la valoarea culorii celui mai apropiat maxim din histogramă.

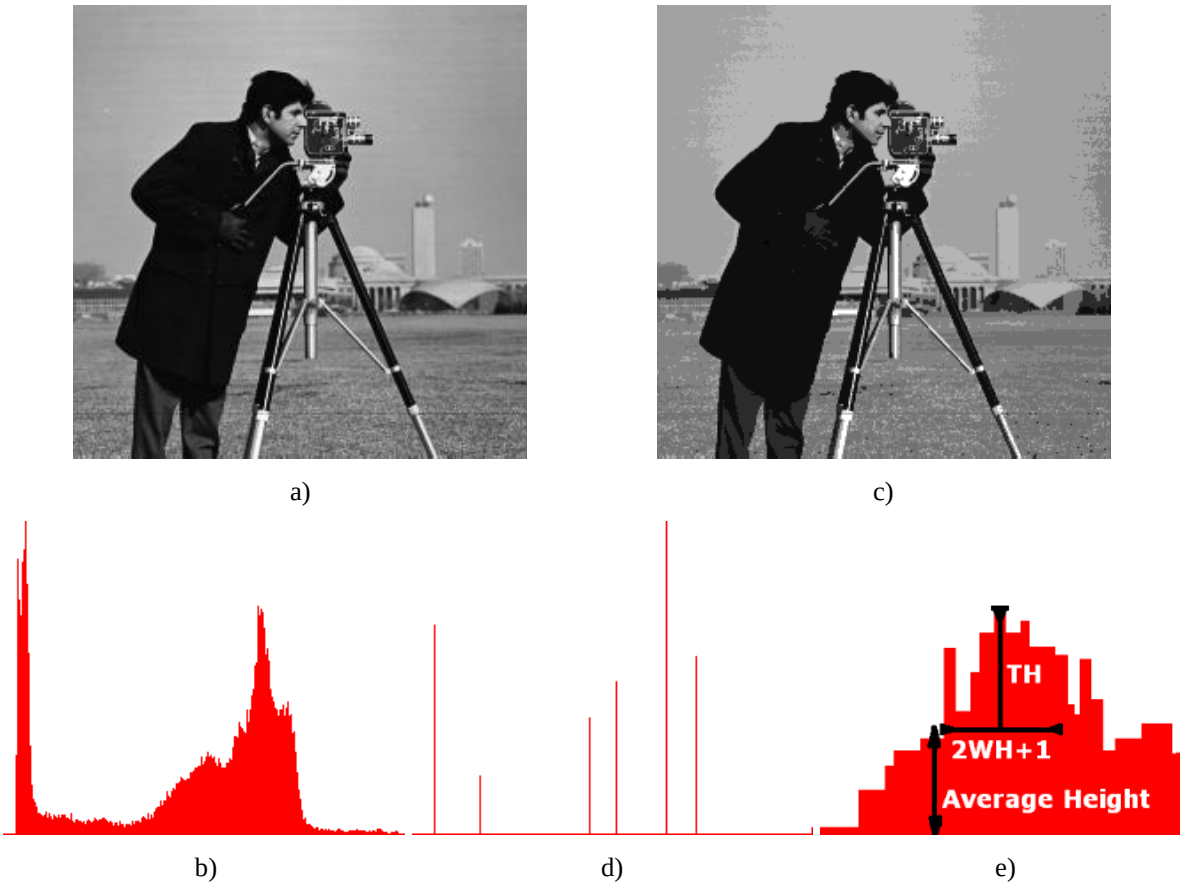


Fig. 3.2 a) Imaginea inițială; b) Histograma imaginii inițiale; c) Imaginea cuantizată obținută (praguri multiple); d) Histograma imaginii cuantizate; e) Algoritmul de calcul al maximelor din histogramă

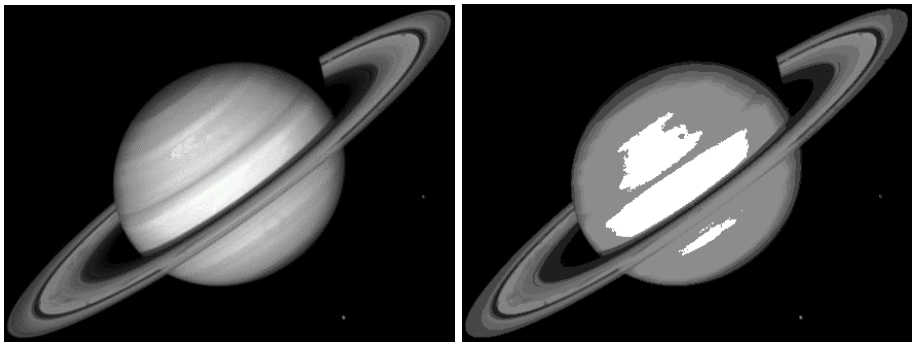
3.4. Algoritmul de corecție Floyd-Steinberg

După cum se observă în Fig. 3.3, rezultatele sunt vizual inacceptabile când numărul de niveluri de gri este scăzut. Pentru a obține rezultate vizuale acceptabile se poate aplica un algoritm de distribuire a erorii de cuantizare pe mai mulți pixeli (*dithering*). Un exemplu de astfel de algoritm este Floyd-Steinberg:

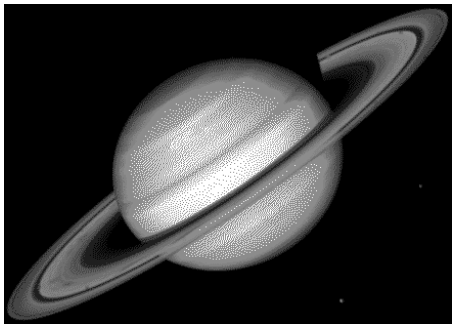
```
for i from [0, rows-1]
   for j from [0, cols-1]
      oldpixel := img(i,j)
      newpixel := find_closest_histogram_maximum(oldpixel)
      img(i,j) := newpixel
      error := oldpixel - newpixel
      if (i,j+1) inside img then
       img(i,j+1)   := img(i,j+1)   + 7*error/16
      if (i+1,j-1) inside img then
       img(i+1,j-1) := img(i+1,j-1) + 3*error/16
      if (i+1,j) inside img then
       img(i+1,j)   := img(i+1,j)   + 5*error/16
      if (i+1,j+1) inside img then
       img(i+1,j+1) := img(i+1,j+1) + error/16
```

Acest algoritm calculează eroarea de cuantificare și o distribuie la pixelii vecini. Fracțiunile de distribuție a erorii sunt prezentate în matricea următoare (**X** = locația pixelului curent):

0	0	0
0	**X**	7/16
3/16	5/16	1/16



a) b)



c)

Fig. 3.3 a) Imaginea inițială; b) Imaginea cuantizată obținută (praguri multiple); c) Împrăștierea erorii pragurilor multiple utilizând algoritmul Floyd-Steinberg

3.5. Detalii de implementare

3.5.1. Afișarea histogramei ca o imagine

Histograma poate fi afișată sub forma unui grafic cu bare verticale. Pentru fiecare nivel de gri se desenează o bară verticală proporțională cu numărul de apariții al acestuia în imagine. Funcția **showHistogram** de mai jos realizează această operație (disponibilă în aplicația **OpenCVApplication**). Trebuie să furnizați titlul ferestrei de afișare, histograma calculată, numărul de acumulatoare, dimensiunea dorită pentru înălțimea imaginii de ieșire. Barele sunt redimensionate automat astfel încât să încapă în imagine și totodată rămân proporționale cu valorile din histogramă.

```
void showHistogram(const string& name, int* hist, const int  hist_cols,
                        const int hist_height) {
      Mat imgHist(hist_height, hist_cols, CV_8UC3, CV_RGB(255, 255, 255));
                                                      // constructs a white image
      //computes histogram maximum
      int max_hist = 0;
      for (int i = 0; i<hist_cols; i++)
            if (hist[i] > max_hist)
                  max_hist = hist[i];
      double scale = 1.0;
      scale = (double)hist_height / max_hist;
      int baseline = hist_height - 1;
      for (int x = 0; x < hist_cols; x++) {
            Point p1 = Point(x, baseline);
            Point p2 = Point(x, baseline - cvRound(hist[x] * scale));
            line(imgHist, p1, p2, CV_RGB(255, 0, 255));   // histogram bins
                                                          // colored in magenta
      }
      imshow(name, imgHist);
}
```

3.5.2. Histograma cu un număr redus de acumulatoare (*bins*)

Histograma se poate calcula și dacă folosim un număr redus de acumulatoare $m \leq 256$. Pentru aceasta trebuie să împărțim intervalul [0,255] în m părți egale și apoi să numărăm pixelii care aparțin fiecărei părți. Această reprezentare este utilă fiindcă are dimensionalitate redusă.

3.6. Activități practice

1. Calculați histograma (într-un vector de întregi de dimensiune 256) pentru o imagine grayscale (cu 8 biți/pixel).
2. Calculați FDP (într-un vector de tip float de dimensiune 256).
3. Afișați histograma calculată utilizând funcția din laborator.
4. Calculați histograma folosind un număr redus de $m \leq 256$ acumulatoare.
5. Implementați algoritmul de reducere a nivelurilor de gri (praguri multiple) de la secțiunea 3.3.
6. Îmbunătățiți algoritmul de reducere a nivelurilor de gri (praguri multiple) utilizând distribuirea erorii cu algoritmul Floyd-Steinberg de la secțiunea 3.4.
7. Realizați algoritmul de reducere a nivelurilor de gri pe canalul Hue din spațiul de culoare HSV al unei imagini color. Modificați doar valorile din canalul H, păstrând canalele S și V neschimbate. O altă opțiune este setarea lor (S și V) la valoarea maximă permisă. Pentru vizualizare transformați înapoi în spațiul RGB.
8. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

3.7. Bibliografie

[1] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.
[2] Algoritmul Floyd-Steinberg, http://en.wikipedia.org/wiki/Floyd-Steinberg_dithering

4. Trăsături geometrice ale obiectelor binare

4.1. Introducere

În această lucrare sunt prezentate câteva trăsături importante ale obiectelor binare precum și modul de calcul al acestora. Trăsăturile prezentate sunt aria, centrul de masă, axa de alungire, perimetrul, factorul de subțiere al obiectului, elongația și proiecțiile obiectului binar.

4.2. Considerații teoretice

În urma operațiilor de segmentare și etichetare a obiectelor din imaginile digitale se obține o imagine cu obiecte care pot fi referite distinct.

Obiectul 'i' poate fi descris în imagine de următoarea funcție:

$$I_i(r,c) = \begin{cases} 1, & \text{dacă } I(r,c) \in \text{obiectului etichetat 'i'} \\ 0 & \text{în celelalte cazuri} \end{cases}$$

$$\text{unde } r \in [0...Height-1] \text{ și } c \in [0...Width-1]$$

Trăsăturile geometrice ale obiectelor se pot clasifica în următoarele două categorii:

- de poziționare și orientare: centrul de masă, aria, perimetrul, axa de alungire;
- de formă: factorul de aspect, factorul de subțiere, numărul Euler, proiecțiile, diametrele Feret ale obiectelor

4.2.1. Aria

$$A_i = \sum_{r=0}^{H-1}\sum_{c=0}^{W-1} I_i(r,c)$$

Aria A_i este măsurată în pixeli și indică mărimea relativă a obiectului.

4.2.2. Centrul de masă

$$\bar{r}_i = \frac{1}{A_i}\sum_{r=0}^{H-1}\sum_{c=0}^{W-1} r I_i(r,c)$$

$$\bar{c}_i = \frac{1}{A_i}\sum_{r=0}^{H-1}\sum_{c=0}^{W-1} c I_i(r,c)$$

Aceste formule corespund liniei și respectiv coloanei centrului obiectului. Această proprietate ajută la localizarea obiectului într-o imagine bidimensională.

4.2.3. Axa de alungire (axa de inerție minimă / momentul de inerție de ordin 2)

$$\tan(2\varphi_i) = \frac{2\sum_{r=0}^{H-1}\sum_{c=0}^{W-1}(r-\bar{r}_i)(c-\bar{c}_i)I_i(r,c)}{\sum_{r=0}^{H-1}\sum_{c=0}^{W-1}(c-\bar{c}_i)^2 I_i(r,c) - \sum_{r=0}^{H-1}\sum_{c=0}^{W-1}(r-\bar{r}_i)^2 I_i(r,c)}$$

Dacă atât numărătorul cât și numitorul fracției din formula de mai sus sunt zero atunci obiectul prezintă simetrie circulară, orice dreaptă ce trece prin centrul de greutate fiind axă de simetrie.

Astfel, pentru aflarea unghiului trebuie aplicată funcția arctangentă. Această funcție este definită pe domeniul (-∞, +∞) și are valori în domeniul (-π/2, π/2). Evaluarea acestei funcții devine instabilă atunci când numitorul fracției se apropie de 0. Totodată, semnele numitorului și numărătorului sunt importante pentru determinarea cadranului corect în care se află rezultatul, dar funcția arctangentă nu face diferența între direcții diametral opuse. Din acest motiv se recomandă folosirea funcției „atan2", funcție care ia ca argumente numărătorul și respectiv numitorul unei astfel de fracții și returnează un rezultat în domeniul (-π, π).

Această proprietate dă informații despre orientarea obiectului. Axa corespunde direcției în jurul căreia obiectul (văzut ca o suprafață plană de grosime constantă) se poate roti cel mai ușor (are momentul cinetic minim).

După ce unghiul φ_i a fost determinat, se verifică corectitudinea valorii găsite prin trasarea axei de alungire. Axa de alungire va corespunde liniei care trece prin centrul de masă al obiectului și care face unghiul φ_i cu axa Ox.

4.2.4. Perimetrul

Perimetrul obiectului ne ajută să determinăm poziția obiectului în spațiu și detalii despre forma lui. Perimetrul poate fi determinat prin numărarea pixelilor de pe contur (pixeli cu valoarea 1 care au cel puțin un pixel vecin de valoare 0).

O primă variantă de detecție a conturului ar fi scanarea imaginii, linie cu linie, și numărarea pixelilor din obiect, care îndeplinesc condiția mai sus menționată. Ca principal dezavantaj al acestei metode este faptul că nu putem face distincție între conturul exterior și eventualele contururi interioare (datorate găurilor din obiect). Deoarece pixelii din imaginile digitale sunt distribuiți după un rastru dreptunghiular, lungimile curbelor și ale dreptelor oblice nu pot fi estimate corect prin numărarea pixelilor. O primă corecție ar fi înmulțirea perimetrului rezultat la algoritmul precedent cu π/4. Există și alte metode de corecție a lungimilor care țin seama de tipul de vecinătate folosit (V4, V8 etc.).

Altă metodă de detecție a conturului unui obiect ar fi detecția muchiilor sale folosind un algoritm de detecție consacrat, subțierea acestora la grosime unitară și numărarea pixelilor de muchie rezultați.

Metodele de tip "chain-codes" sunt metode mai complexe de detecție a conturului și oferă acuratețe mai mare.

4.2.5. Factorul de subțiere al obiectului (thinness ratio)

$$T = 4\pi\left(\frac{A}{P^2}\right)$$

Această funcție are valoarea maximă 1, care corespunde cercului. Această proprietate poate fi folosită pentru a vedea cât de rotund este un obiect. Cu cât este mai aproape de 1 cu atât obiectul este mai rotund. Această valoare dă și informații despre regularitatea obiectului.

Obiectele cu contur regulat au o valoare a raportului mai mare decât obiectele cu contur neregulat. Valoarea 1/T se numește factor de neregularitate al obiectului (sau factor de compactitate).

4.2.6. Elongația (factorul de aspect) (“aspect ratio”/elongație/excentricitate)

Această mărime se determină prin scanarea imaginii și determinarea valorilor minime și maxime ale liniilor și coloanelor ce formează dreptunghiul circumscris obiectului.

$$R = \frac{c_{\max} - c_{\min} + 1}{r_{\max} - r_{\min} + 1}$$

4.2.7. Proiecțiile obiectului binar

Proiecțiile ne dau informații despre forma obiectului. Proiecția orizontală este egală cu suma pixelilor pe fiecare linie, iar proiecția verticală este egală cu suma pixelilor pe fiecare coloană.

$$h_i(r) = \sum_{c=0}^{W-1} I_i(r,c) \qquad v_i(c) = \sum_{r=0}^{H-1} I_i(r,c)$$

Proiecția este utilă în programele de recunoaștere a scrisului unde obiectul care ne interesează poate fi normalizat.

4.3. Detalii de implementare

Pentru a face distincție între diversele obiecte prezente în imagine vom considera că fiecare dintre ele este desenat cu o culoare diferită. Aceste culori pot fi rezultatul unei operații prealabile de etichetare sau pot fi generate manual (vezi Fig. 4.1).

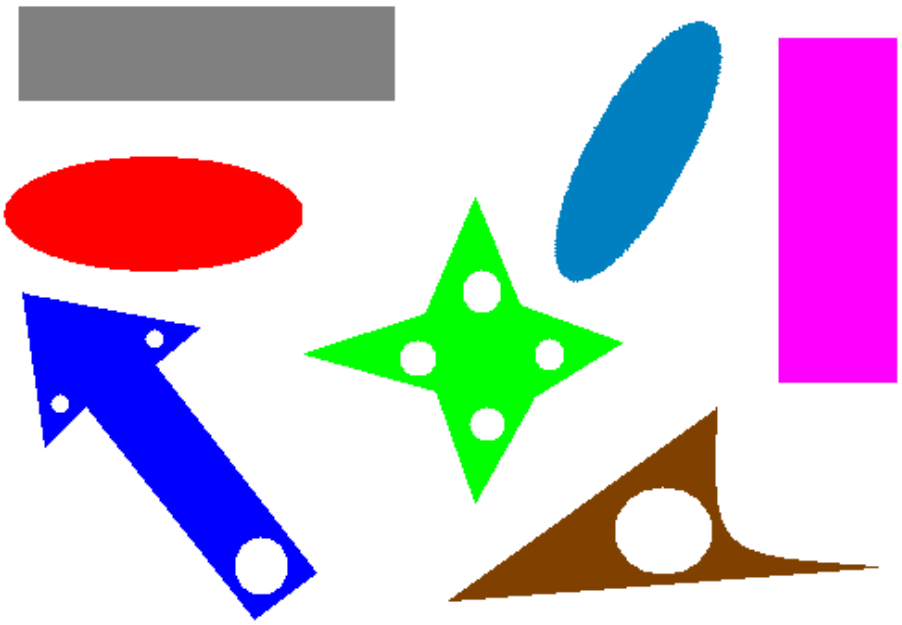


Fig. 4.1 Exemplu de imagine etichetată pe care pot fi testați algoritmii descriși

Există mai multe abordări pentru implementarea extragerii proprietăților geometrice:

4.3.1. Calcularea proprietăților geometrice pentru toate obiectele din imagine, ca o singură operație

Se vor identifica pixelii fiecărui obiect pe baza etichetei (culorii) unice și se vor calcula trăsăturile geometrice. Se aplică acest procedeu pentru fiecare obiect etichetat din imagine.

4.3.2. Calcularea proprietăților geometrice a unui anumit obiect, selectat în prealabil cu ajutorul mouse-ului

Utilizatorul va selecta obiectul dorit printr-un click pe suprafața lui. Ca răspuns la această acțiune se vor afișa la consolă trăsăturile geometrice dorite.

Pentru a adăuga o metodă handler pentru evenimentul de click se va folosi funcția `setMouseCallback` din OpenCV, care are rolul de a seta un *handler* pentru mouse la o anumită fereastră.

```
void setMouseCallback(const string& winname, MouseCallback onMouse, void* userdata=0)
```

`winname` – numele ferestrei,
`onMouse` – numele funcției *callback*, care va fi apelată în momentul când apare un eveniment de mouse in fereastra `winname`,
`userdata` – parametru opțional, care poate fi transmis funcției *callback*

Se va implementa și funcția **onMouse** în care se vor implementa operațiile de calcul ale trăsăturilor.

```
void onMouse (int event, int x, int y, int flags, void* param)
```

`event` - este evenimentul de mouse, care poate avea următoarele valori:

- EVENT_MOUSEMOVE
- EVENT_LBUTTONDOWN
- EVENT_RBUTTONDOWN
- EVENT_MBUTTONDOWN
- EVENT_LBUTTONUP
- EVENT_RBUTTONUP
- EVENT_MBUTTONUP
- EVENT_LBUTTONDBLCLK
- EVENT_RBUTTONDBLCLK
- EVENT_MBUTTONDBLCLK

`x, y` – sunt coordonatele x și y la care s-a produs evenimentul,
`flags` – corespunde unor condiții specifice atunci când evenimentul se produce,
`param` – sunt parametrii utilizator, care se transmit din funcția `setMouseCallback`.

În framework-ul OpenCVApplication este prezentat un exemplu concret de handler la evenimentele de mouse în funcția `testMouseClick()`.

Pentru trasarea axei de alungire, folosiți funcția `line` din OpenCV:

```
void line( Mat img,  Point pStart, Point pEnd, Scalar color, int thickness )
```

`img` – imagine în care se face desenarea
`pStart`, `pEnd` – cele două puncte între care se trasează linia
`color` – culoarea liniei
`thickness` – grosimea liniei

4.4. Activități practice

1. Pentru un anumit obiect, selectat printr-un *click* dintr-o imagine etichetată, calculați aria, centrul de masă, axa de alungire, perimetrul, factorul de subțiere și elongația (factorul de aspect).
    a. Afișați valorile obținute la consolă
    b. Într-o imagine separată (copie a imaginii sursă):
        - Desenați punctele de contur ale obiectului selectat
        - Desenați centrul de masă al obiectului selectat
        - Afișați axa de alungire a obiectului selectat folosind funcția *line* din OpenCV
    c. Calculați și afișați într-o imagine separată (copie a imaginii sursă) proiecțiile obiectului selectat
2. Creați o funcție nouă care, având ca și imagine sursă o imagine etichetată, să păstreze în imaginea destinație doar acele obiecte care:
    a. au *aria* < *TH_arie*
    b. și au o anumită orientare *phi*, unde *phi_LOW* < *phi* < *phi_HIGH*

   unde *TH_arie, phi_LOW, phi_HIGH* sunt citite de la tastatură.
3. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

4.5. Bibliografie

[1] Umbaugh Scot E., *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8.

5. Etichetarea componentelor conexe

5.1. Introducere

În această lucrare de laborator se vor prezenta algoritmi pentru etichetarea obiectelor distincte din imagini alb-negru. Ca rezultat, fiecare obiect va avea un număr unic. Acest număr, numit etichetă, va putea fi folosit pentru analiza fiecărui obiect în parte.

5.2. Fundamente teoretice

Vom prezenta doi algoritmi pentru etichetare. Intrarea acestor algoritmi este imaginea binară, iar ieșirea este o matrice de etichete, de aceeași dimensiune ca imaginea de intrare. Această matrice trebuie să aibă celula de un tip numeric capabil să rețină numărul total de etichete.

În imaginea binară de intrare, obiectele sunt reprezentate ca și componente conexe de culoare neagră (0), iar fundalul are culoarea albă (255). Pentru a defini noțiunea de componentă conexă, trebuie să definim tipurile de vecinătate.

Vecinătatea de 4 a unei poziții **(i,j)** este definită ca pozițiile:

$$N_4(i,j)=\{(i-1,j),\ (i,j-1),\ (i+1,j),\ (i,j+1)\},$$

i.e. vecinii de sus, stânga, jos și dreapta.

Vecinătatea de 8 conține toate pozițiile care diferă de poziția curentă cu maxim 1 unitate pe o una sau ambele coordonate:

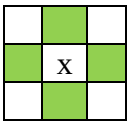
$$N_8(i,j) = \{(k,l) \mid |k-i|\leq 1,\ |l-j|\leq 1,\ (k,l)\neq(i,j)\},$$

Deci include vecinătatea de 4 și vecinii de pe diagonale.

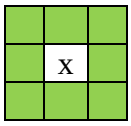
La parcurgerea imaginii într-o direcție oarecare, putem defini vecinii anteriori raportat la această direcție. Vecinii anteriori ai unei parcurgeri sus-jos, stânga-dreapta sunt:

$$Np(i,j)=\{(i,j-1),\ (i-1,j-1),\ (i-1,j),\ (i-1,j+1)\}.$$

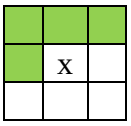
Aceste definiții sunt ilustrate în figura de mai jos.



a) Vecinătate de 4



b) Vecinătate de 8



c) Vecinii anteriori

Vom defini un graf format de imaginea binară. Vârfurile grafului sunt pozițiile pixelilor obiect, iar vecinătatea dintre pixeli reprezintă muchiile grafului. Două vârfuri sunt vecine dacă unul dintre ele aparține vecinătății celuilalt. Vom utiliza vecinătatea de 4 sau de 8, astfel că graful generat este neorientat. O componentă conexă este o mulțime de vârfuri pentru care există o cale între oricare două elemente ale acestei mulțimi.

5.2.1. Algoritm 1 – Traversarea în lățime

Vom începe cu o metodă directă pentru etichetare, care se bazează pe traversarea în lățime a grafului format de imaginea binară. Primul pas este inițializarea matricei de etichete cu valoarea zero pentru toți pixelii, indicând faptul că inițial totul este neetichetat. Apoi, algoritmul va căuta un pixel de tip obiect care este neetichetat. Dacă acest punct este găsit, el va primi o etichetă nouă, pe care o va propaga vecinilor lui. Vom repeta acest proces până când toți pixelii obiect vor primi o etichetă.

Algoritmul este descris de următorul pseudocod:

```
label = 0
labels = zeros(height, width) // matrice height x width, cu valori 0
for i = 0:height-1
      for j = 0:width-1
            if img(i,j)==0 and labels(i,j)==0
                  label++
                  Q = queue()
                  labels(i,j) = label
                  Q.push( (i,j) )
                  while Q not empty
                        q = Q.pop()
                        for each neighbor in N8(q)
                           if img(neighbor)==0 and labels(neighbor)==0
                              labels(neighbor) = label
                              Q.push( neighbor )
```

Algoritm 1. Traversare în lățime pentru etichetarea componentelor conexe

Structura de date de tip coadă menține lista punctelor care trebuie etichetate cu eticheta curentă ”label”. Deoarece este o structură FIFO, se va obține traversarea în lățime. Vom marca nodurile vizitate setând eticheta pentru poziția lor, în matricea de etichete. Dacă structura de date se schimbă într-o stivă, se va obține o traversare în adâncime.

5.2.2. Algoritm 2 – Două treceri cu clase de echivalență

Etichetarea poate fi realizată prin două parcurgeri liniare pe imagine, plus o procesare suplimentară pe un graf mult mai mic. Această abordare folosește mai puțină memorie. Deoarece în algoritmul anterior aveam nevoie de memorarea unei liste de puncte, în cazul în care o componentă conexă este foarte mare dimensiunea listei poate fi comparabilă cu dimensiunea imaginii.

Algoritmul al doilea va face o primă parcurgere și va genera etichete inițiale pentru pixelii obiect. Pentru fiecare pixel vom lua în considerare pixelii deja vizitați și etichetați, deci vom folosi vecinătatea pixelilor anteriori definită mai sus, N_p. După inspectarea etichetelor pixelilor deja vizitați, avem următoarele cazuri:

- Dacă nu avem vecin anterior etichetat, vom crea o etichetă nouă
- Dacă avem vecini etichetați, vom selecta minimul acestor etichete (notat cu *x*). După aceea, vom marca fiecare etichetă *y* din vecinătate, diferită de *x*, ca echivalentă cu *x*.

Vom asigna eticheta găsită în pasul anterior pentru poziția curentă. După prima trecere, există câte o etichetă pentru fiecare poziție din imagine. Totuși, vor exista etichete diferite pentru același obiect, care sunt etichete echivalente, deci va trebui să fie înlocuite cu o etichetă nouă, care reprezintă o clasă de echivalență.

Relațiile de echivalență definesc un graf neorientat, având ca noduri etichetele inițiale. Acest graf este de obicei mult mai mic decât graful inițial al pixelilor obiect, deoarece numărul de noduri este numărul de etichete generate la prima parcurgere. Muchiile grafului sunt relațiile de echivalență. Putem aplica algoritmul 1 pe acest graf mai mic, pentru a obține o nouă listă de etichete. Toate etichetele echivalente cu 1 se re-etichetează cu 1, următoarea componentă conexă ne-echivalentă cu 1 se re-etichetează cu 2, și așa mai departe. O nouă trecere prin matricea de etichete inițiale va realiza re-etichetarea.

```
label = 0
labels = zeros(height, width)
vector<vector<int>> edges(1000)
for i = 0:height-1
      for j = 0:width-1
            if img(i,j)==0 and labels(i,j)==0
                  L = vector()
                  for each neighbor in Np(i,j)
                        if labels(neighbor)>0
                              L.push_back(labels(neighbor))
                  if L.size() == 0        // asignează o nouă etichetă
                        label++
                        labels(i,j) = label
                  else                    // asignează vecinul minim
                        x = min(L)
                        labels(i,j) = x
                        for each y from L
                           if (y <> x)
                              edges[x].push_back(y)
                              edges[y].push_back(x)

newlabel = 0
newlabels = zeros(label+1)   // șir de zero, de dimensiune label+1
for i = 1:label
      if newlabels[i]==0
            newlabel++
            Q = queue()
            newlabels[i] = newlabel
            Q.push( i )
            while Q not empty
                  x = Q.pop()
                  for each y in edges[x]
                        if newlabels[y] == 0
                              newlabels[y] = newlabel
                              Q.push( y )


for i = 0:height-1
      for j = 0:width-1
            labels(i,j) = newlabels[labels(i,j)]
```

Algoritm 2. Etichetarea componentelor conexe prin două treceri

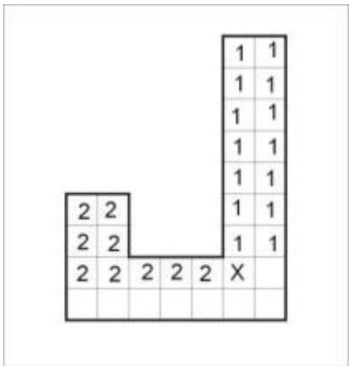


Fig. 5.1 Exemplu de caz unde vecinii anteriori au etichete diferite.
Etichetele 1 și 2 sunt marcate ca echivalente

5.3. Detalii de implementare

Codul următor ilustrează cum se poate parcurge vecinătatea de 4 a unui pixel. Acest cod se poate modifica ușor pentru a obține o vecinătate de 8 sau pentru a lua în considerare doar vecinii de sus și din stânga.

```
int di[4] = {-1,0,1,0};
int dj[4] = {0,-1,0,1};
uchar neighbors[4];
for(int k=0; k<4; k++)
    neighbors[k] = img.at<uchar>(i+di[k], j+dj[k]);
```

Atenție la limitele imaginii! Nu le depășiți!

Rețineți etichetele într-o matrice capabilă să reprezinte numărul maxim de etichete:

$2^8 = 256$ - **`uchar (CV_8UC1)`**
$2^{16} = 65536$ - **`short (CV_16SC1)`**
$2^{32} \sim 2.1e9$ - **`int (CV_32SC1)`**

Puteți utiliza containerele **`std::stack`** și **`std::queue`** pentru a reține punctele pentru Algoritmul 1: stivă pentru DFS, coadă pentru BFS. Punctele se pot stoca sub formă de structuri **`pair<int,int>`**. Un exemplu de cod pentru inițializarea unei cozi și efectuarea de operații pe aceasta:

```
#include <queue>
queue<pair<int,int>> Q;
Q.push( pair<int,int>(i,j) );
pair<int,int> p = Q.front(); Q.pop();
// se pot accesa coordonatele punctului p astfel
i = p.first; j = p.second;
```

Relațiile de echivalență care definesc muchiile grafului mai mic pot fi memorate folosind liste de adiacență, sub forma **`vector<vector<uchar>>`**. Exemplu de cod pentru inițializarea listei și inserarea muchiilor:

```
// ne asigurăm că vectorul are o dimensiune suficientă
vector<vector<int>> edges(1000);
// dacă u este echivalent cu v
edges[u].push_back(v);
edges[v].push_back(u);
```

Pentru a afișa matricea de etichete sub forma unei imagini color trebuie să generăm o culoare aleatoare pentru fiecare etichetă. Se poate utiliza generatorul implicit, care este mai bun decât apelul clasic **`rand()%256`**.

```
#include <random>
default_random_engine gen;
uniform_int_distribution<int> d(0,255);
uchar x = d(gen);
```

5.4. Exemple de etichetare

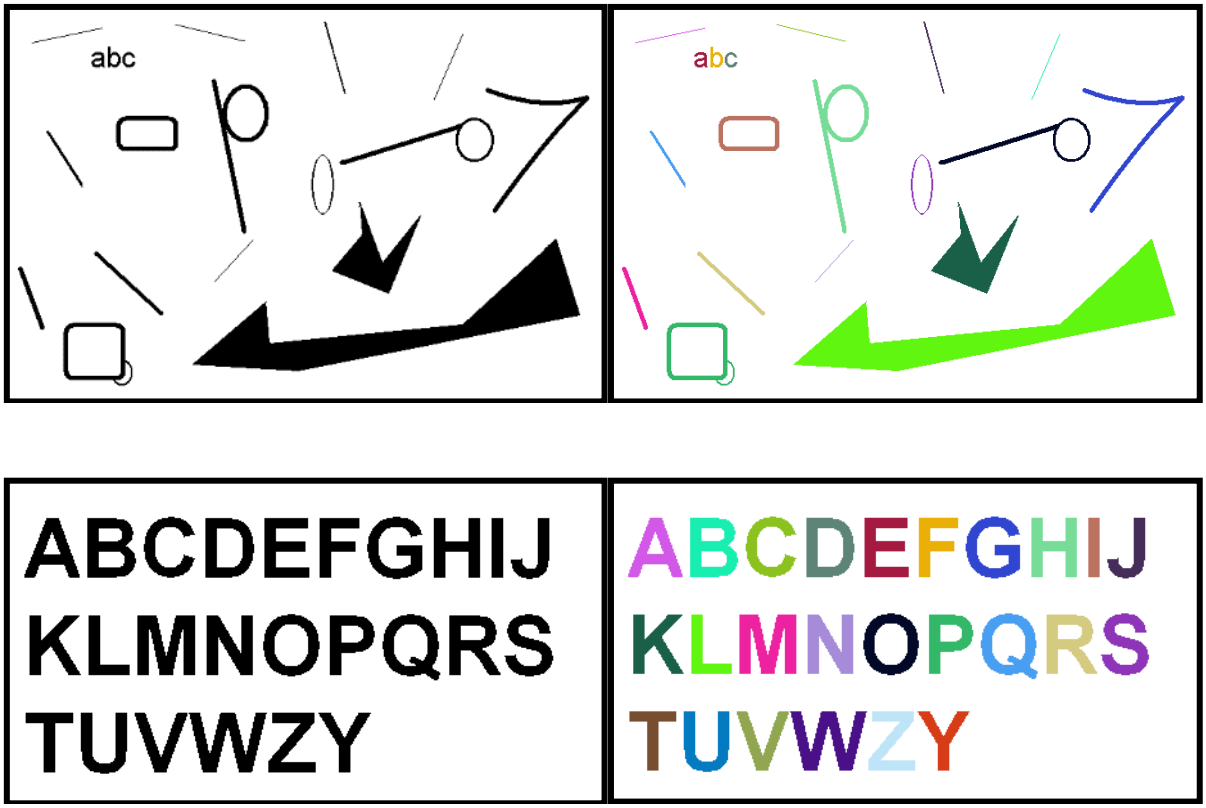


Fig. 5.2 Exemple de etichetare

5.5. Activități practice

1. Implementați algoritmul de traversare în lățime (Algoritmul 1). Implementați astfel încât să puteți schimba vecinătatea din tipul N4 în tipul N8 și invers.
2. Implementați o funcție care va genera o imagine color pornind de la matricea de etichete. Afișați rezultatele.
3. Implementați algoritmul de etichetare cu două treceri (Algoritmul 2). Afișați rezultatele intermediare după prima parcurgere. Comparați acest rezultat cu rezultatul final și cu rezultatul primului algoritm.
4. Opțional, vizualizați procesul de etichetare prin afișarea rezultatelor intermediare și făcând o pauză după fiecare pas pentru a ilustra ordinea de traversare a punctelor.
5. Opțional, schimbați coada în stivă, pentru a implementa traversarea DFS.
6. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

5.6. Bibliografie

[1] Umbaugh Scot E., *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8.
[2] Robert M. Haralick, Linda G. Shapiro, *Computer and Robot Vision*, Addison-Wesley Publishing Company, 1993.

6. Algoritmul de urmărire a conturului

6.1. Obiective

Obiectivele acestui laborator sunt următoarele:

- extragerea conturului obiectelor folosind algoritmul de urmărire a conturului;
- reprezentarea eficientă a conturului extras folosind coduri înlănțuite;
- exploatarea avantajelor utilizării codurilor înlănțuite în reprezentarea contururilor obiectelor (reconstrucția conturului, potrivire, unire, etc.).

6.2. Fundamente teoretice

6.2.1. Algoritmul de urmărire a conturului

Algoritmul de urmărire a conturului este folosit pentru extragerea conturului obiectelor dintr-o imagine. La aplicarea acestui algoritm presupunem că imaginea este binară sau că obiectele din imagine au fost etichetate în prealabil.

Pașii algoritmului:

1. Se scanează imaginea din colțul stânga sus până când se găsește un pixel care aparține unei regiuni; acest pixel P_0 reprezintă pixelul de start al conturului regiunii. Se definește o variabilă *dir* în care se reține direcția mutării anterioare de-a lungul conturului de la elementul anterior spre elementul curent. Se inițializează:
   (a) *dir* = 0 dacă conturul este detectat folosind vecinătate de 4 (Fig. 6.1a)
   (b) *dir* = 7 dacă conturul este detectat folosind vecinătate de 8 (Fig. 6.1b)
2. Se parcurge vecinătatea de 3x3 a pixelului curent în sens invers acelor de ceasornic, începând cu pixelul corespunzător poziției:
   (a) (*dir* + 3) *mod* 4 (Fig. 6.1c)
   (b) (*dir* + 7) *mod* 8 dacă *dir* este par (Fig. 6.1d)
   (*dir* + 6) *mod* 8 dacă *dir* este impar (Fig. 6.1e)
   Primul pixel găsit care are aceeași valoare ca și pixelul curent este noul element P_n al conturului. Se actualizează valoarea lui *dir*.
3. Dacă elementul curent P_n al conturului este egal cu al doilea element P_1 din contur și dacă elementul anterior P_{n-1} este egal cu primul element P_0, atunci algoritmul se încheie. Altfel se repetă pasul (2).
4. Conturul detectat este reprezentat de pixelii $P_0 \ldots P_{n-2}$.

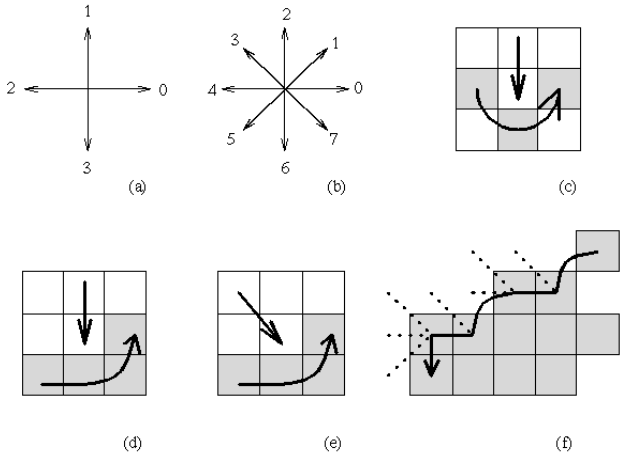


Fig. 6.1 (a) Reprezentarea direcției, vecinătate de 4, (b) vecinătate de 8, (c) secvența de căutare în cazul vecinătății de 4 pixeli, (d),(e) secvența de căutare în cazul vecinătății de 8 pixeli, (f) urmărirea conturului pentru vecinătate de 8 (liniile întrerupte arată pixelii care au fost testați în timpul algoritmului de urmărire a conturului) [3]

Observații:

- Algoritmul funcționează pentru toate regiunile care au suprafața mai mare de un pixel.
- Determinarea conturului unei regiuni formate dintr-un pixel este o problemă trivială.
- Algoritmul descris mai sus determină conturul exterior al regiunilor, dar nu găsește conturul găurilor din interiorul regiunilor.
- Pentru a determina și conturul găurilor care apar într-un obiect, conturul trebuie urmărit începând cu fiecare regiune sau element din conturul unei găuri dacă acest element nu face parte dintr-un contur parcurs deja.
- Dacă obiectele au lățimea egală cu un pixel trebuie adăugate condiții suplimentare.

6.2.2. Extragerea codurilor înlănțuite

Codul înlănțuit reprezintă o modalitate eficientă de reprezentare a conturului unui obiect dintr-o imagine alb negru. Codul înlănțuit încorporează informații despre lungimea conturului obiectului, despre aria sa și despre momente. Codurile înlănțuite sunt folosite la calcularea unor parametri pentru diferite tipuri de curbe. Codurile înlănțuite sunt reversibile, încât conturul unui obiect poate fi reconstruit având la dispoziție codul înlănțuit.

Ideea de bază la reprezentarea codului înlănțuit este că fiecare pixel de pe conturul unui obiect are un vecin adiacent care face parte din contur și direcția de trecere de la un pixel dat de pe contur la acest vecin poate fi specificată printr-un număr unic ce ia valori între 0 și 7 (vecinătate de 8). Codurile înlănțuite pot fi definite și folosind o vecinătate de 4. Un exemplu este dat în Fig. 6.4.

În explicațiile care urmează vom folosi doar vecinătatea de 8 pixeli. Fiind dat un pixel, se consideră cei 8 vecini ai săi. Fiecăruia dintre ei i se poate asocia un număr de la 0 la 7 care reprezintă una din cele 8 direcții posibile de trecere de la pixelul curent la unul din vecini (vezi Fig. 6.2). Această orientare se păstrează pentru toată imaginea.

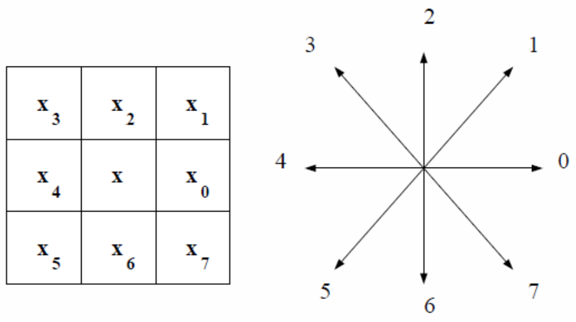


Fig. 6.2 Vecinătatea de 8 și cele 8 direcții asociate

Codul înlănțuit al conturului unei imagini binare este o secvență de numere întregi $c=\{c_0, c_1, \dots , c_{n-1}\}$, cu c_i aparținând mulțimii {0, 1, … , 7} pentru i=0, 1, … , n-1. Numărul de elemente din mulțimea c reprezintă lungimea codului înlănțuit. Elementul c_0 este punctul *inițial* și c_{n-1} este punctul *final* al codului. Pornind de la un punct de referință, conturul unui obiect dintr-o imagine alb-negru poate fi urmărit parcurgând codul înlănțuit.

Fig. 6.3 ilustrează procesul de urmărire a conturului unui triunghi folosind vectorii de direcție. Să presupunem că alegem pixelul cel mai de sus punct de referință (x=109, y=61) pentru codificarea conturului. Codul înlănțuit pentru conturul triunghiului va fi o secvență de 245 coduri: 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2.

Fiind dat punctul de referință și codul înlănțuit, conturul triunghiului poate fi reconstruit complet. Codul înlănțuit reprezintă o modalitate eficientă de a stoca informația de contur deoarece în reprezentarea lui sunt necesari 3 biți ($2^3 = 8$) pentru a determina oricare din cele 8 direcții de deplasare.

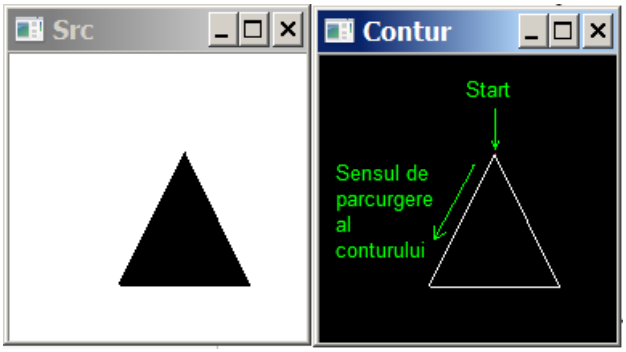


Fig. 6.3 Direcțiile codului înlănțuit cu numerele corespunzătoare

Codurile înlănțuite pot fi reprezentate independent de poziție prin ignorarea „punctului inițial” („punctul de început”). În cazul contururilor închise codurile înlănțuite pot fi normalizate în raport cu punctul de start prin alegerea acestuia astfel încât secvența rezultată la reprezentarea codului înlănțuit să formeze un număr întreg cu valoare absolută minimă.

„Derivata” codului înlănțuit este o reprezentare utilă deoarece este invariantă la rotația conturului. Derivata (o diferență *modulo* 4 sau 8) este o altă secvență de numere care indică direcția relativă a segmentelor codului înlănțuit; ele reprezintă numărul de rotații cu 90 de grade sau cu 45 de grade necesare pentru a obține direcția următorului segment din codul înlănțuit. O diferență *modulo* 4 sau 8 se numește derivata codului înlănțuit (vezi Fig. 6.4).

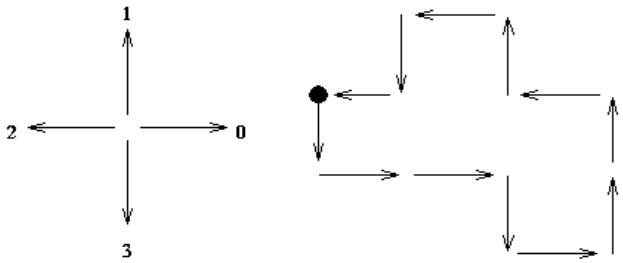


Fig. 6.4 Cod înlănțuit pentru vecinătate de 4 și derivata codului înlănțuit

Cod:	3, 0, 0, 3, 0, 1, 1, 2, 1, 2, 3, 2
Derivata:	1, 0, 3, 1, 1, 0, 1, 3, 1, 1, 3, 1

Proprietăți ale codului înlănțuit:

- Codurile înlănțuite descriu un obiect folosind o secvență de segmente de dimensiune unitate având orientări date (vecinătate de 4).
- Primul element al unei astfel de secvențe trebuie să conțină informații despre poziția primului pixel pentru ca regiunea să poată fi reconstruită.
- Codurile pare {0, 2, 4, 6} corespund direcțiilor verticale și orizontale; codurile impare {1, 3, 5, 7} corespund direcțiilor diagonale.
- Fiecare cod poate fi considerat ca fiind direcția unghiulară, în multipli de 45 de grade, în care trebuie să fie parcurși pixelii succesivi ai conturului.
- Coordonatele absolute ale primului pixel din contur (cel mai de sus, din stânga) împreună cu codul înlănțuit al conturului reprezintă informația completă despre conturul regiunii.
- O schimbare a două elemente consecutive din codul înlănțuit marchează o schimbare în direcția conturului. Punctul în care apare aceasta schimbare se numește *colț*.

6.3. Activități practice

Utilizând *OpenCVApplication* și fișierele adiționale laboratorului:

1. Implementați algoritmul de urmărire a conturului și desenați conturul obiectului dintr-o imagine având un singur obiect.
2. Folosind algoritmul de urmărire a conturului scrieți algoritmul care construiește codul înlănțuit și derivata codului înlănțuit al unui obiect. Calculați și afișați (la linia de comandă sau într-un fișier text) ambele coduri (codul înlănțuit și derivata) pentru o imagine având un singur obiect.
3. Implementați o funcție care reconstruiește (afișează) conturul unui obiect peste o imagine, având ca și date de intrare coordonatele punctului de start și secvența de cod înlănțuit utilizând vecinătate de 8 (*reconstruct.txt*). Încărcați imaginea *gray_background.bmp* și apelați funcția care reconstruiește conturul. Trebuie să obțineți conturul cuvântului „EXCELLENT” (având literele unite între ele).
4. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

Informații adiționale:

Imaginile de test care conțin un singur obiect au:

- 8 biți/pixel;
- indexul 0 pentru pixelii obiect (pixeli negri)
- altă valoare a indexului pentru pixelii de fundal (pixeli albi)

Fișierul *reconstruct.txt* este un fișier text care conține:

- pe prima linie coordonatele punctului de start (rând și coloană) separate printr-un spațiu;
- pe a doua linie numărul de coduri înlănțuite;
- pe a treia linie codurile înlănțuite (secvența de direcții pentru vecinătate de 8 pixeli) separate printr-un spațiu (nu este trecut codul implicit de start (7) aferent primului punct).

6.4. Bibliografie

[1] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.
[2] Umbaugh Scot E., *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8.
[3] M. Sonka, V. Hlavac, R. Boyle, *Segmentation*, In Image Processing, Analysis and Machine Vision. Springer, Boston, MA, 1993, pp. 112-191, DOI 10.1007/978-1-4899-3216-75.
[4] G.X. Ritter, J.N. Wilson, *Handbook of Computer Vision Algorithms*, In Image Algebra Second Edition – Chapter 10.4 Chain Code Extraction and Correlation, CRC Press, New York 2001.
[5] Representation of Two-Dimensional Geometric Structures, http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/LIB/bandb8_12.pdf

7. Operații morfologice pe imagini binare

7.1. Introducere

Operațiile morfologice pe imagini afectează forma sau structura unui obiect. Ele se aplică doar pe imagini binare (imagini cu doar două culori, alb și negru). Operațiile morfologice se folosesc de obicei ca etape de pre- sau post-procesare a imaginilor (filtrare, subțiere sau eliminare a protuberanțelor) sau pentru obținerea unei reprezentări sau descrieri a formei obiectelor sau regiunilor (contururi, schelete, înfășurători convexe).

7.2. Considerații teoretice

Operațiile morfologice principale sunt *dilatarea* și *eroziunea* [1]. Dilatarea mărește obiectele, permițând umplerea unor mici goluri și conectarea obiectelor disjuncte. Eroziunea micșorează obiectele prin erodarea marginilor obiectelor. Aceste operații pot fi adaptate diverselor aplicații prin selectarea elementului structural folosit, care determină modul în care vor fi dilatate sau erodate obiectele.

Notații:

pixeli obiect: mulțimea pixelilor de interes (asupra cărora se aplică operațiile morfologice)
pixeli fundal: complementul mulțimii pixelilor de obiect

7.2.1. Dilatarea

Dilatarea se face prin suprapunerea unui element structural **B** peste imaginea **A** și mutarea lui succesivă, începând de la stânga la dreapta, de sus în jos. Imaginea care se va obține după dilatare are aceeași dimensiune cu imaginea **A.** Toți pixelii acesteia sunt inițializați la valoarea de "fundal" înainte să înceapă procesul de dilatare. Operația aplicată este una neliniară și poate fi definită prin următoarea secvență de pași:

1. Dacă originea elementului structural se suprapune cu un punct de "fundal" din imaginea **A**, nu se efectuează nicio modificare în imaginea rezultat și se trece mai departe la următorul pixel.
2. Dacă originea elementului structural coincide cu un punct "obiect" în imagine, atunci toți pixelii acoperiți de elementul structural devin pixeli "obiect" în imaginea rezultat.

Notație: $A \oplus B$

Un element structural este alcătuit numai din pixeli marcați ca "obiect". Dispunerea acestora poate varia dând naștere la diverse forme. Câteva forme des întâlnite sunt:

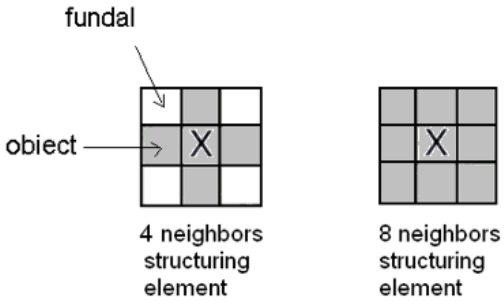


Fig. 7.1 Forme tipice pentru elementele structurale **B**

În Fig. 7.2 este prezentat un exemplu de dilatare. A se observa că în cazul de față toți pixelii "obiect" vor fi păstrați, marginile obiectelor vor fi extinse, iar micile goluri vor fi umplute.

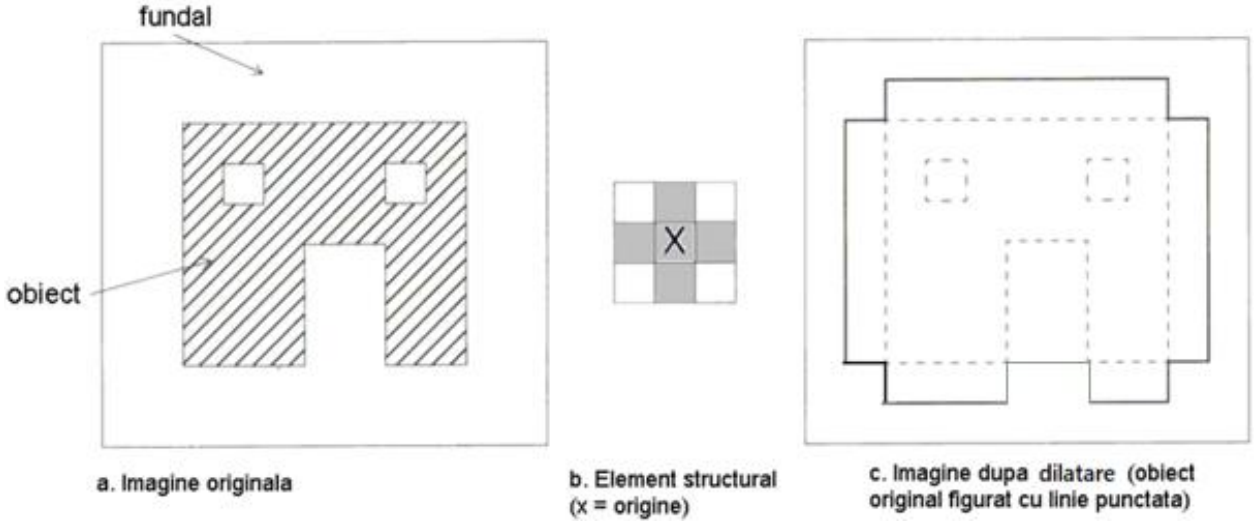


Fig. 7.2 Ilustrarea procesului de dilatare [1]

a) b)

Fig. 7.3 Exemplu de dilatare (obiect = negru / fundal = alb): a) Imaginea originală **A**;
b) Imaginea rezultată în urma operației: $\mathbf{A} \oplus \mathbf{B}$

7.2.2. Eroziunea

Operația de eroziune este similară cu cea de dilatare, dar efectul este oarecum invers. Imaginea rezultat, de aceeași dimensiune cu **A**, este inițializată integral cu pixeli de "fundal". Se utilizează aceeași tehnică și anume elementul structural este deplasat peste imaginea **A**, de la stânga la dreapta, de sus în jos. La fiecare poziție nouă se va folosi următoarea secvență de pași:

1. Dacă elementul structural acoperă doar puncte "obiect" în **A**, pixelul din imaginea destinație corespunzător poziției elementului structural devine pixel "obiect".
2. Dacă elementul structural acoperă cel puțin un punct de "fundal", pixelul din imaginea destinație va rămâne pixel de "fundal".

Notație: $A \ominus B$

În Fig. 7.4 au rămas doar pixelii "obiect" care coincid cu originea elementului structural, dacă acesta s-a suprapus în întregime peste pixelii unui obiect existent. Pentru că elementul structural are

3 pixeli lățime, partea din dreapta a obiectului a fost complet erodată, dar partea din stânga și-a menținut câțiva dintre pixeli, pentru că inițial avea 3 pixeli lățime.

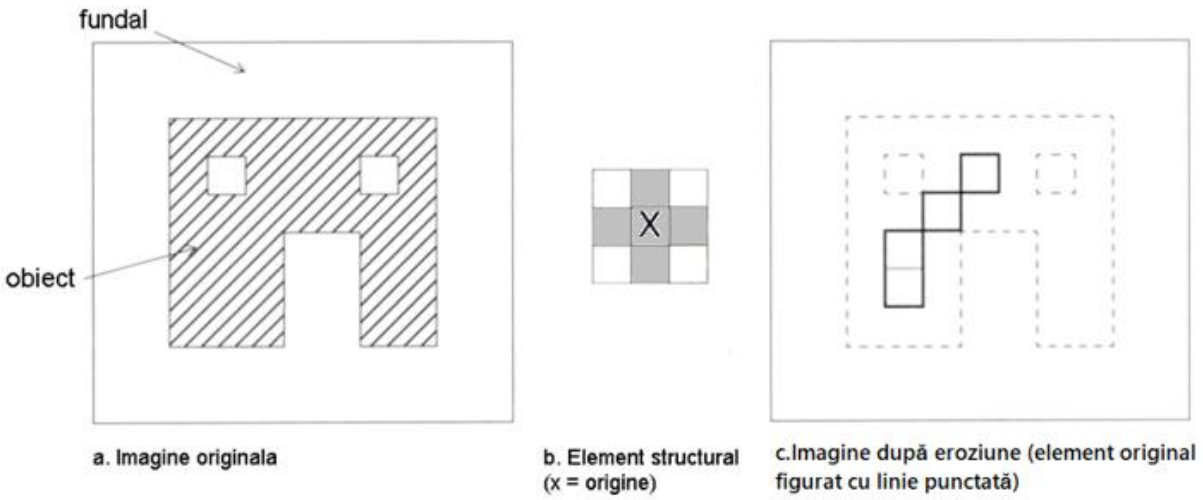
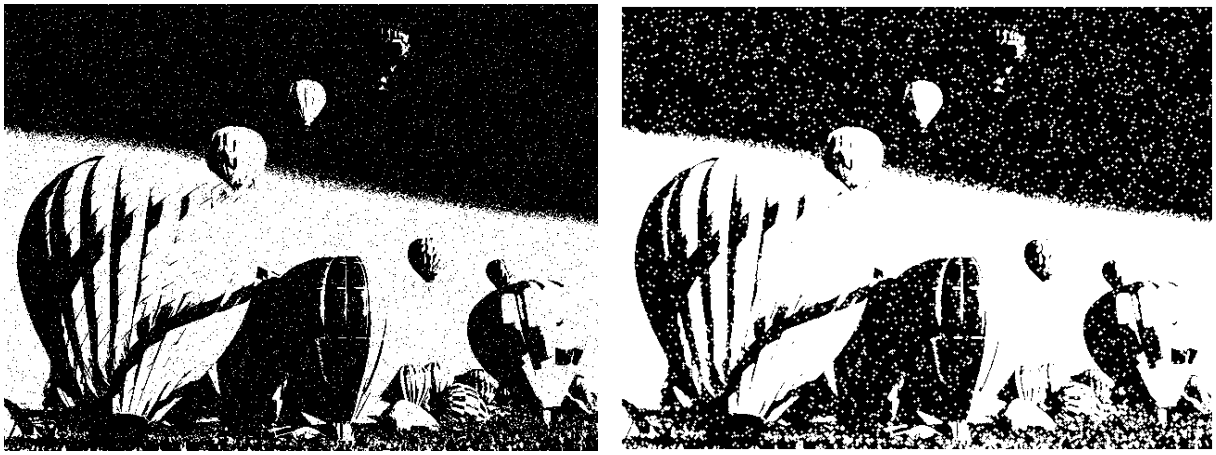


Fig. 7.4 Ilustrarea procesului de eroziune [1]



a) b)

Fig. 7.5 Exemplu de eroziune (obiect = negru / fundal = alb): a) Imaginea originală **A**;
b) Imaginea rezultat: $\mathbf{A} \ominus \mathbf{B}$

7.2.3. Deschidere și închidere

Cele două operații de bază, dilatarea și eroziunea pot fi combinate în secvențe de operații complexe. Cele mai utile operații de filtrare morfologică sunt deschiderea și închiderea [1]. *Deschiderea* constă într-o eroziune urmată de o dilatare și poate fi folosită pentru eliminarea pixelilor din regiunile care sunt prea mici pentru a conține elementul structural. În acest caz, elementul structural este adesea numit *sondă*, pentru că se caută obiectele prea mici și le filtrează. Vezi Fig. 7.6 ca exemplu pentru deschidere.

Notație: $A \circ B = (A \ominus B) \oplus B$

Închiderea constă într-o dilatare urmată de o eroziune și poate fi folosită pentru umplerea de goluri sau mici discontinuități. În Fig. 7.7 se poate vedea că operația de închidere are ca efect umplerea golurilor și a discontinuităților. Comparând imaginile din partea stângă și partea dreaptă a Fig. 7.8, putem observa că ordinea operațiilor de dilatare și eroziune este importantă. Închiderea și deschiderea au rezultate diferite, chiar dacă ambele constau dintr-o operație de eroziune și una de

dilatare.

Notație: $A \bullet B = (A \oplus B) \Theta B$

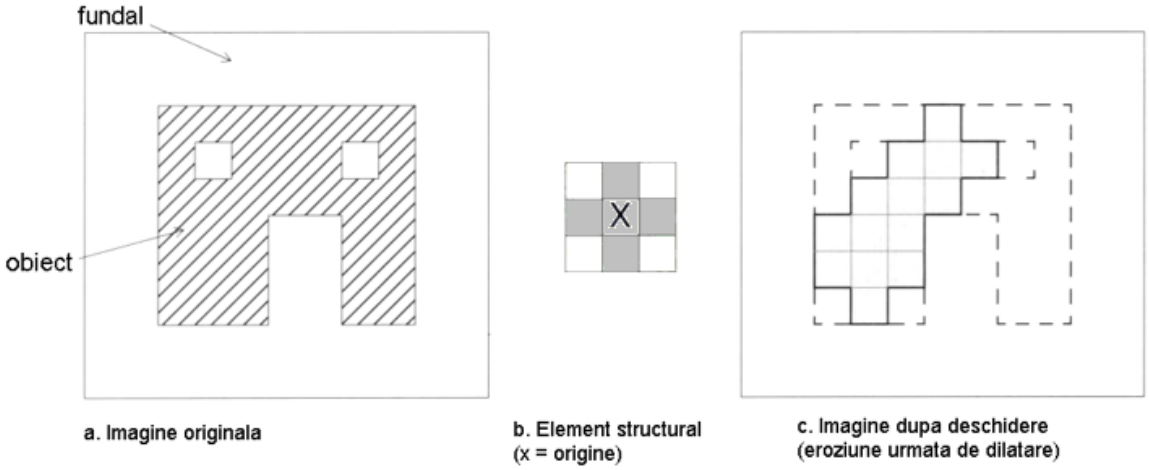


Fig. 7.6 Ilustrarea operației de deschidere [1]

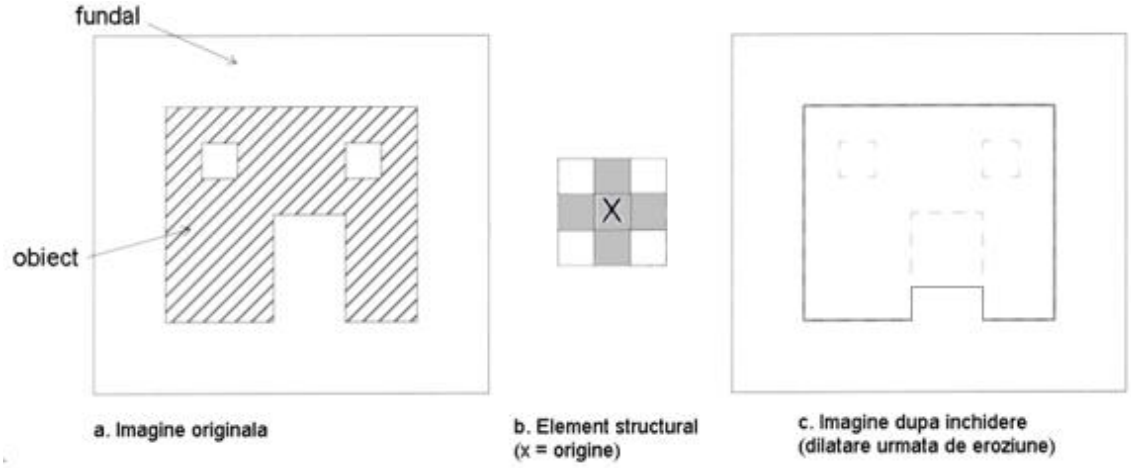


Fig. 7.7 Ilustrarea operației de închidere (obiect = negru / fundal = alb) [1]

a. b.

Fig. 7.8 Rezultatul deschiderii (a) și al închiderii (b) aplicate pe imaginea originală din Fig. 7.5a (obiect = negru / fundal = alb)

7.2.4. Prezentarea unor algoritmi morfologici de bază [2]

Extragerea conturului

Conturul unei mulțimi *A*, notat prin *β(A)*, poate fi obținut prin erodarea mulțimii *A* cu elementul structural *B*, urmată de efectuarea diferenței dintre *A* și rezultatul eroziunii ei, mai precis:

$$\beta(A) = A - (A \ominus B)$$

unde

B este un element structural.

'–' reprezintă operația de diferență a două mulțimi (ilustrată în Fig. 7.10)

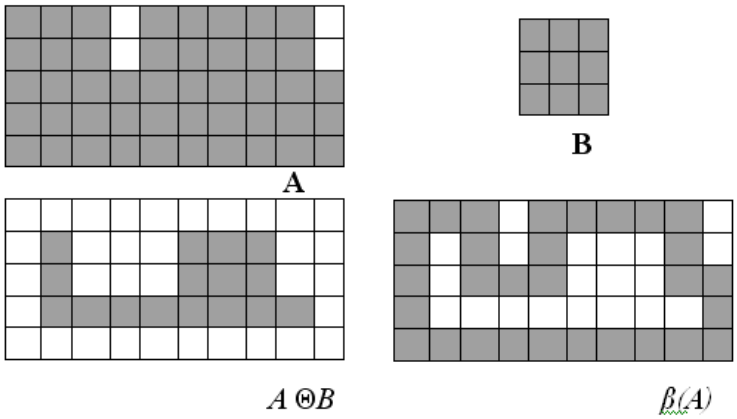


Fig. 7.9 Ilustrarea algoritmului de extragere a conturului

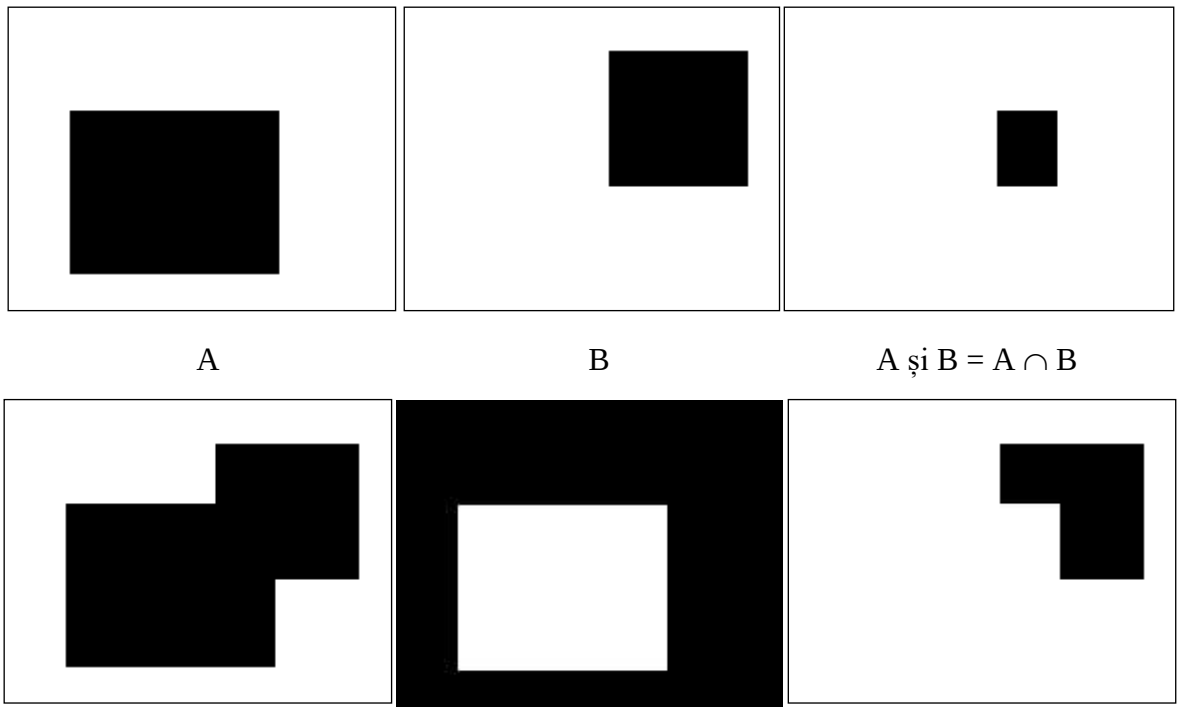


Fig. 7.10 Ilustrarea operații pe mulțimi (prin diagrame Venn-Euler)

Umplerea regiunilor

În continuare vom prezenta un algoritm simplu de umplere a regiunilor pentru care se știe conturul. Algoritmul este bazat pe dilatări, complement și intersecții.

Pornind de la un punct *p* aflat în interiorul unei regiuni (unui contur), obiectivul algoritmului este umplerea întregii regiuni cu pixeli "obiect". Dacă adoptăm convenția că toate punctele care nu aparțin conturului regiunii sunt de "fundal", atunci vom atribui valoarea "obiect" punctului *p* la începutul algoritmului. Prin folosirea următorului algoritm vom umple întreaga regiune cu pixeli "obiect":

$$X_k = (X_{k-1} \oplus B) \cap A^C \qquad k=1,2,3,\ldots$$

unde

$X_0=p$,

B – reprezintă elementul structural

$\cap$ – reprezintă operatorul de intersecție (vezi Fig. 7.10)

A^C – reprezintă complementul mulțimii A (vezi Fig. 7.10)

Algoritmul se termină atunci când la iterația k are loc egalitatea $X_k=X_{k-1}$. Reuniunea mulțimilor X_k și A conține atât interiorul mulțimii A cât și conturul acesteia, ambele conținând doar puncte "obiect".

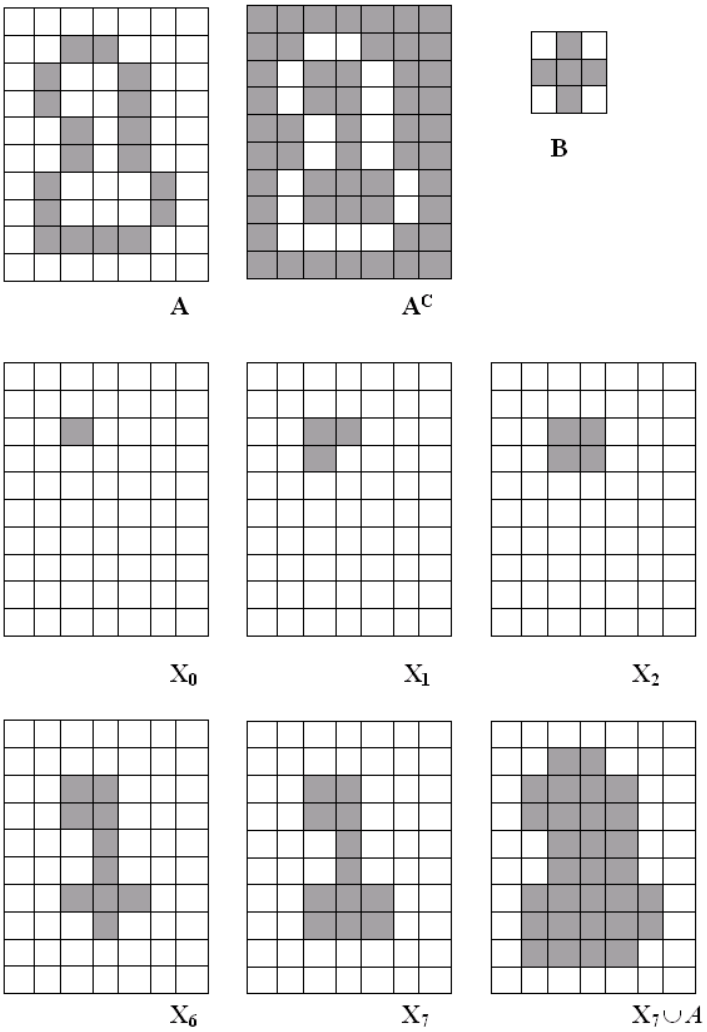


Fig. 7.11 Ilustrarea algoritmului de umplere a regiunilor

7.3. Detalii de implementare

7.3.1. Folosirea unui buffer suplimentar pentru procesări înlănțuite

Aplicarea operațiilor morfologice (dilatare și eroziune) trebuie făcută în felul următor:

Imagine destinație = *Imagine sursă* (operator) *Element structural*

Imaginea sursă nu trebuie să fie afectată în niciun fel!

Pentru implementarea operațiilor morfologice complexe (deschidere și închidere) sau a operațiilor repetate (de exemplu: *n* eroziuni consecutive) într-o singură funcție de procesare, este necesară crearea unui buffer de imagine suplimentar.

7.4. Activități practice

1. Adăugați la framework-ul *OpenCVApplication* funcții de procesare care să implementeze operațiile morfologice de bază (dilatare, eroziune, deschidere, închidere).
2. Adăugați facilități care să permită aplicarea în mod repetat (de *n* ori) a operațiilor morfologice. Introduceți numărul de repetiții de la linia de comandă. Remarcați proprietatea de idempotență a deschiderii și închiderii (vezi curs) care face ca aplicarea în mod repetat să fie inutilă în cazul lor.
3. Implementați algoritmul de extragere a conturului.
4. Implementați algoritmul de umplere a regiunilor.
5. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

7.5. Bibliografie

[1] Umbaugh Scot E., *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8.
[2] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.

8. Proprietăți statistice ale imaginilor de intensitate

8.1. Introducere

În această lucrare se vor prezenta principalele trăsături statistice care caracterizează distribuția nivelurilor de intensitate într-o imagine de intensitate (grayscale) sau dintr-o zonă/regiune de interes (ROI) a imaginii. Aceste mărimi statistice se pot aplica în mod analog și imaginilor color pe fiecare componentă de culoare în parte.

În cadrul acestei lucrări vom folosi următoarele notații:

- $L = 255$ nivelul maxim de intensitate al imaginii
- $h(g)$ histograma imaginii (numărul de pixeli având nivelul de gri g)
- $M = H$ x W, numărul de pixeli din imagine
- $p(g) = h(g)/M$ funcția de densitate de probabilitate a nivelurilor de gri (FDP).

8.2. Valoarea medie a nivelurilor de intensitate

Este o măsură a intensității medii a imaginii sau a regiunii de interes. O imagine întunecată va avea o medie scăzută (Fig. 8.1a), iar una luminoasă o medie ridicată (Fig. 8.1b).

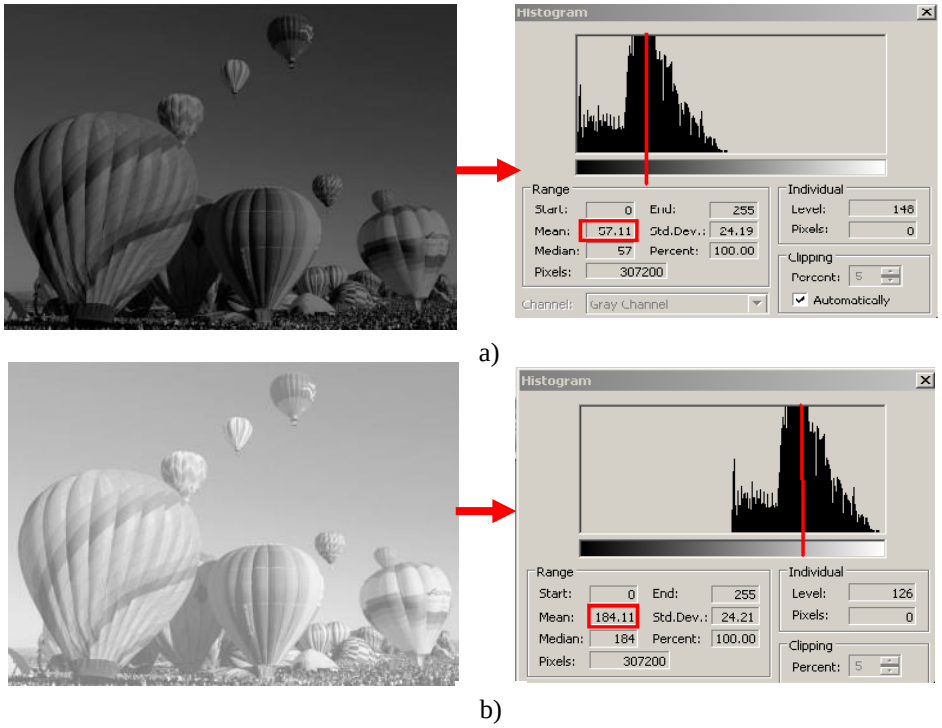


Fig. 8.1 Ilustrarea poziției histogramei și a valorii medii a nivelurilor de intensitate pentru o imagine întunecată (a) și una luminoasă (b)

Calculul valorii medii a intensităților se face folosind formulele:

$$\bar{g} = \mu = \int_{-\infty}^{+\infty} g \cdot p(g) dg = \sum_{g=0}^{L} g \cdot p(g) = \frac{1}{M} \sum_{g=0}^{L} g \cdot h(g) \quad (8.\ 1)$$

$$\bar{g} = \mu = \frac{1}{M} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} I(i,j) \quad (8.\ 2)$$

8.3. Deviația standard a nivelurilor de intensitate

Este o măsură a contrastului imaginii (regiunii de interes) și caracterizează gradul de împrăștiere al nivelurilor de intensitate față de valoarea medie. O imagine cu contrast ridicat va avea o deviație standard mare (Fig. 8.2a – histograma este împrăștiată pe întreaga plajă a nivelurilor de intensitate), iar o imagine cu contrast scăzut va avea o deviație standard mică (Fig. 8.2b – histograma este restrânsă la câteva niveluri de intensitate în jurul valorii medii).

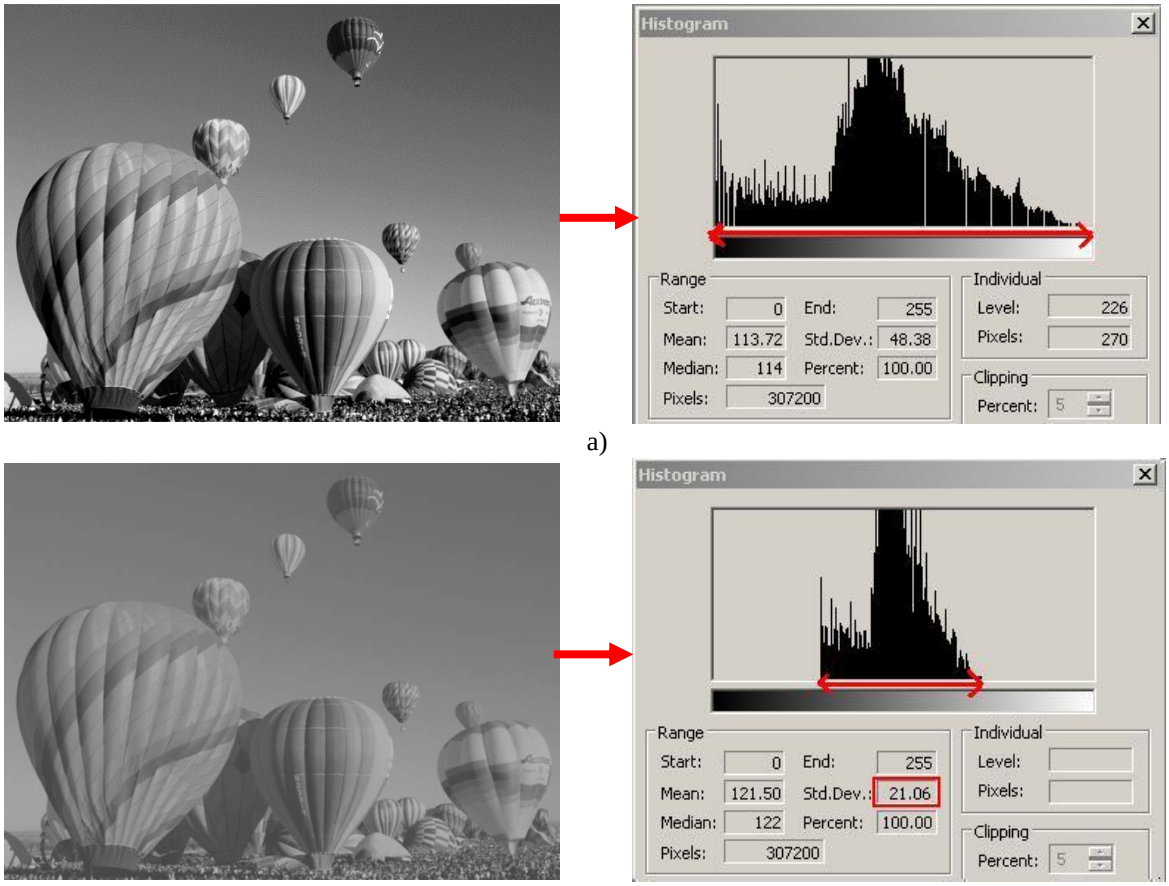


Fig. 8.2 Ilustrarea poziției histogramei și a deviației standard (2σ) a nivelurilor de intensitate pentru o imagine cu contrast ridicat (a) și una cu contrast scăzut (b)

Calculul deviației standard a intensităților:

$$\sigma = \sqrt{\sum_{g=0}^{L}(g-\mu)^2 \cdot p(g)} \tag{8. 3}$$

$$\sigma = \sqrt{\sum_{i=0}^{H-1}\sum_{j=0}^{W-1}(I(i,j)-\mu)^2} \tag{8. 4}$$

8.4. Histograma cumulativă

Histograma cumulativă C calculează numărul pixelilor din imagine a căror intensitate este mai mică sau egală cu o valoare g, unde g∈[0,255]. Histograma cumulativă se poate calcula prin acumularea valorilor din histogramă. Astfel:

$$C(g) = \sum_{j=0}^{g} h(j)$$

unde *h* este histograma nivelurilor de intensitate din imagine.

8.5. Binarizare automată globală

Acest algoritm de binarizare folosește imagini care au histograma bimodală (două vârfuri, obiecte și fundal). Având două vârfuri, este de ajuns un singur prag (*T*) pentru binarizare.

Algoritm

1. Inițializare:
   - Se calculează histograma *h*
   - Se găsește intensitatea maximă I_{max} și intensitatea minimă I_{min} în imagine
   - Se alege o valoare inițială pentru pragul *T*:

$$T = (I_{min} + I_{max}) / 2$$

2. Se segmentează imaginea pe baza pragului T și se calculează valorile medii de intensitate:
   - se calculează valoarea medie μ_{G_1} pentru pixelii care satisfac condiția G_1: $I(i,j) \leq T$
   - se calculează valoarea medie μ_{G_2} pentru pixelii care satisfac condiția G_2: $I(i,j) > T$

Implementare eficientă: se calculează mediile μ_{G_1} și μ_{G_2} folosind histograma inițială

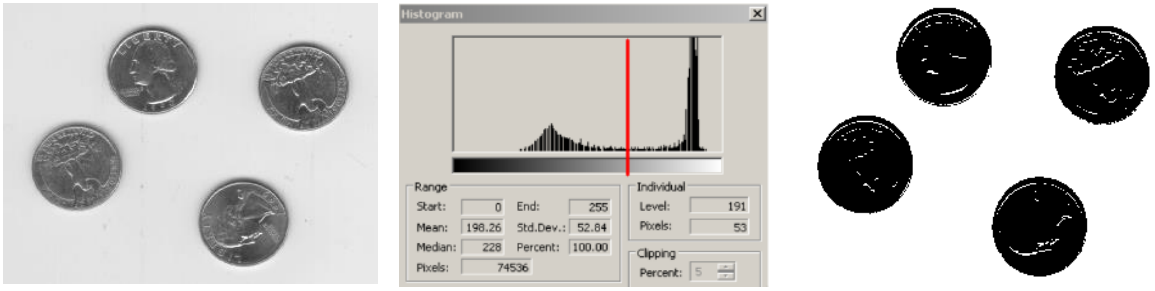
$$\mu_{G_1} = \frac{1}{N_1} \sum_{g=I_{min}}^{g=T} g \cdot h(g), \text{ unde } N_1 = \sum_{g=I_{min}}^{g=T} h(g)$$

$$\mu_{G_2} = \frac{1}{N_2} \sum_{g=T+1}^{g=I_{max}} g \cdot h(g), \text{ unde } N_2 = \sum_{g=T+1}^{g=I_{max}} h(g)$$

3. Se actualizează pragul de binarizare: $T = (\mu_{G_1} + \mu_{G_2})/2$

4. Se repetă pașii 2-3 până când $|T_k - T_{k-1}| < eroare$ (unde *eroare* este o valoare pozitivă)

5. Se binarizează imaginea folosind pragul *T*



a) Imaginea originală b) Histograma imaginii c) Imagine binară după segmentare cu pragul T = 165 (*eroare* = 0.1)

Fig. 8.3 Rezultatul binarizării cu pragul calculat

8.6. Funcții de transformare cu formă analitică

În Fig. 8.4 sunt ilustrate câteva funcții tipice de transformare a nivelurilor de intensitate, exprimate într-o formă analitică:

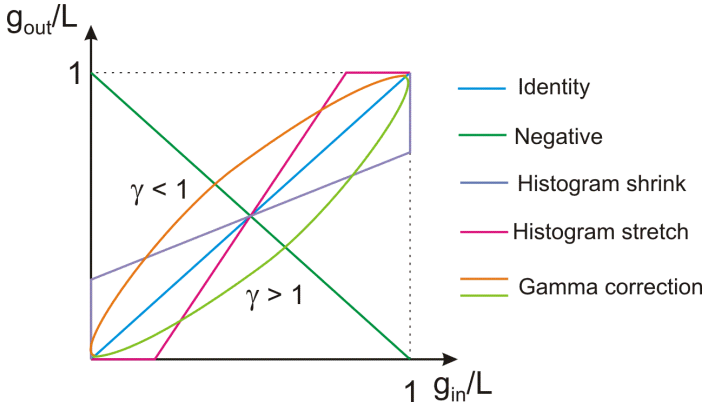


Fig. 8.4 Funcții tipice de transformare a nivelurilor de intensitate

8.6.1. Funcția identitate (fără efect)

$$g_{out} = g_{in} \tag{8. 5}$$

8.6.2. Negativul imaginii

$$g_{out} = L - g_{in} = 255 - g_{in} \tag{8. 6}$$

8.6.3. Modificarea luminozității

- Adunarea unei valori pozitive (*offset* > 0) rezultă în creșterea luminozității
- Adunarea unei valori negative (*offset* < 0) rezultă în scăderea luminozității

$$g_{out} = g_{in} + offset \tag{8. 7}$$

Atenție: se va face întotdeauna verificarea următoare: 0 <= g_{out} <=255, iar eventualele depășiri se vor rezolva prin saturare !!!

a) Imaginea inițială b) *Offset = 50* c) *Offset = -50*

Fig. 8.5 Modificarea luminozității

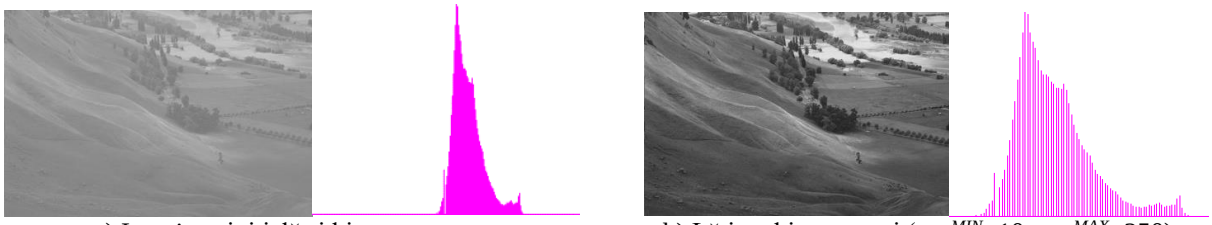
8.6.4. Modificarea contrastului (lățirea/îngustarea histogramei)

- Transformarea intensităților originale din intervalul [g_{in}^{MIN}, g_{in}^{MAX}] în intervalul [g_{out}^{MIN}, g_{out}^{MAX}]
- Lățirea histogramei determină creșterea contrastului
- Îngustarea histogramei determina scăderea contrastului

$$g_{out} = g_{out}^{MIN} + (g_{in} - g_{in}^{MIN}) \frac{g_{out}^{MAX} - g_{out}^{MIN}}{g_{in}^{MAX} - g_{in}^{MIN}} \quad (8.\ 8)$$

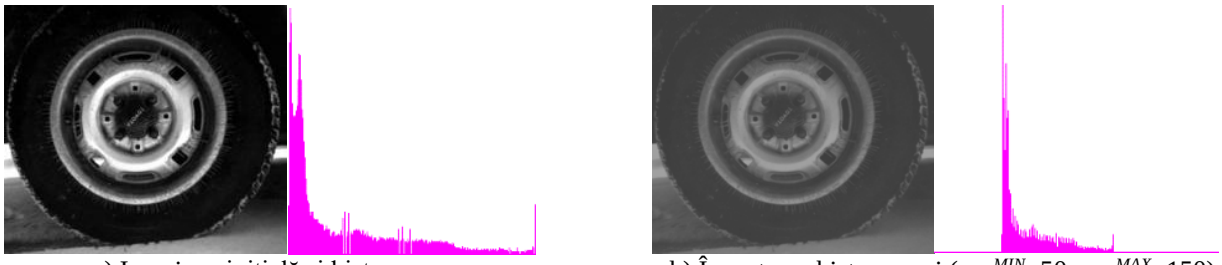
unde:

$$\frac{g_{out}^{MAX} - g_{out}^{MIN}}{g_{in}^{MAX} - g_{in}^{MIN}} = \begin{cases} > 1 & \Rightarrow \quad lățire \\ < 1 & \Rightarrow \quad îngustare \end{cases} \quad (8.\ 9)$$



a) Imaginea inițială și histograma b) Lățirea histogramei (g_{out}^{MIN}=10, g_{out}^{MAX}=250)

Fig. 8.6 Lățirea histogramei



a) Imaginea inițială și histograma b) Îngustarea histogramei (g_{out}^{MIN}=50, g_{out}^{MAX}=150)

Fig. 8.7 Îngustarea histogramei

8.6.5. Corecția gamma

$$g_{out} = L\left(\frac{g_{in}}{L}\right)^{\gamma} \quad (8.\ 10)$$

unde: γ este un coeficient pozitiv: subunitar (codificare/compresie gamma) sau supraunitar (decodificare/decompresie gamma)

Atenție: se va face întotdeauna verificarea următoare: 0 <= g_{out} <=255, iar eventualele depășiri se vor rezolva prin saturare !!!

Imaginea inițială

$\gamma < 1$: codificare/comprimare gamma

$\gamma > 1$: decodificare/decompresie gamma

Fig. 8.8 Ilustrarea rezultatelor operațiilor de corecție gamma

8.7. Egalizarea histogramei

Este o transformare care permite obținerea unei imagini cu histogramă/FDP cvasiuniformă, indiferent de forma histogramei/FDP imaginii de intrare. Pentru aceasta se va folosi următoarea transformare (vezi notele de curs pentru mai multe detalii):

$$s_k = p_c(k) = \sum_{g=0}^{k} \frac{h(g)}{M} g_{out}^{MIN}, \ k = 0 \dots L \tag{8. 11}$$

unde:
k – nivelul de intensitate al imaginii de intrare,
s_k – nivelul de intensitate normalizat al imaginii de ieșire,
$p_c(k)$ – funcția densității de probabilitate cumulative (FDPC) a imaginii de intrare.

8.7.1. Algoritmul de egalizare a histogramei

1. Se calculează histograma sau FDP a imaginii de intrare (vector de 256 elemente).
2. Se calculează FDPC, conform (8.11), sub forma unui vector p_c de 256 elemente.
3. Deoarece din relația (8.11) se obțin valori s_k normalizate ale intensităților de ieșire este necesară înmulțirea valorii s_k cu L (255):

$$g_{out} = Ls_k = \frac{L}{M} \sum_{g=0}^{g_{in}} h(g) \qquad , \qquad k = g_{in} \tag{8. 12}$$

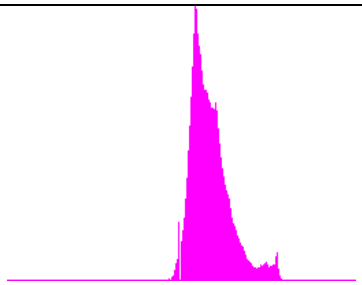
Această funcție de transformare se poate scrie sub forma unei tabele (vector) de echivalențe:

$$g_{out} = tab(g_{in}) = 255 \cdot p_C(g_{in}) \tag{8. 13}$$

4. Se calculează intensitățile pixelilor din imaginea de ieșire (egalizată) pe baza echivalențelor din tabelă:

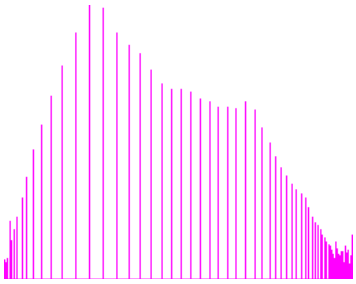
$$Dst(i,j) = tab(Src(i,j)) \tag{8. 14}$$

a) Imaginea inițială



b) Histograma inițială

c) După egalizarea histogramei



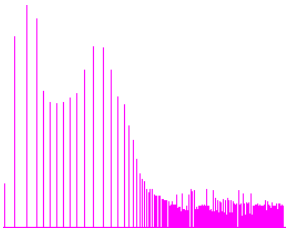
d) Histograma egalizată

Fig. 8.9 Egalizarea histogramei

e) Imaginea inițială

f) După egalizare

g) FDPC



h) Histograma egalizată

Fig. 8.10 Egalizarea histogramei

8.8. Activități practice

1. Calculați și afișați media, deviația standard, histograma și histograma cumulativă a nivelurilor de intensitate. Pentru histogramă folosiți funcția *ShowHistogram* din *OpenCVApplication* (vezi și Laborator 3).
2. Implementați metoda de determinare automată a pragului de binarizare (vezi secțiunea 8.5) și binarități imaginile folosind acest prag. Afișați pragul obținut.
3. Implementați funcțiile de transformare a histogramei (vezi secțiunea 8.6) pentru calculul negativului imaginii, lățirea/îngustarea histogramei, corecția gamma, modificarea luminozității. Introduceți limitele g_{out}^{MIN}, g_{out}^{MAX}, coeficientul gamma și valoarea de creștere a luminozității prin intermediul consolei. După fiecare procesare afișați histogramele imaginilor (sursă si destinație).
4. Implementați algoritmul de egalizare a histogramei (vezi secțiunea 8.7). Afișați histogramele imaginilor (sursă și destinație).
5. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

8.9. Bibliografie

[1] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.

9. Filtrarea imaginilor în domeniul spațial și frecvențial

9.1. Introducere

În această lucrare se va prezenta operatorul de convoluție. Acest operator stă la baza aplicării operațiilor liniare de filtrare a imaginilor aplicate în domeniul spațial (în planul imagine prin manipularea directă a pixelilor din imagine) sau în domeniul frecvențelor (aplicarea unei transformate Fourier, filtrare și apoi aplicarea transformatei Fourier inversă). Exemple de astfel de filtre sunt: filtre trece-jos (de netezire a imaginilor, de eliminare a zgomotelor), filtre trece-sus (de evidențiere a muchiilor) etc.

9.2. Operația de convoluție în domeniul spațial

Operația de convoluție implică folosirea unei măști/nucleu de convoluție H (de obicei de formă simetrică de dimensiune $w \times w$, cu $w=2k+1$), care se aplică peste imaginea sursă în conformitate cu (9.2).

$$I_D(i,j) = H * I_S \tag{9.1}$$

$$I_D(i,j) = \sum_{u=0}^{w-1} \sum_{v=0}^{w-1} H(u,v) \cdot I_S(i+u-k, j+v-k) \tag{9.2}$$

Aceasta implică parcurgerea imaginii sursă I_S, pixel cu pixel, ***ignorând primele și ultimele k linii și coloane*** (Fig. 9.1) și calcularea valorii intensității de la locația curentă (i, j) a imaginii de ieșire I_D în conformitate cu (9.2). Nucleul de convoluție se poziționează cu elementul central peste poziția curentă (i, j).

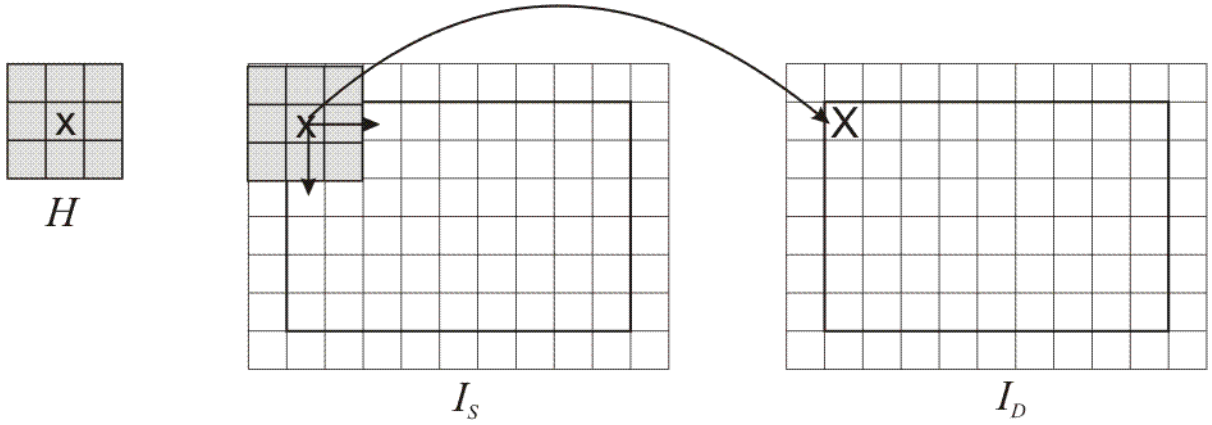


Fig. 9.1 Ilustrarea operației de convoluție

Nucleele de convoluție pot avea și forme ne-simetrice (elementul central / de referință nu mai este poziționat în centrul de simetrie). Modul de aplicare a operației de convoluție cu astfel de nuclee este similar, dar astfel de exemple nu vor fi prezentate în lucrarea de față.

9.2.1. Filtre de tip „trece-jos”

Aceste nuclee se folosesc pentru operații de netezire și/sau filtrare a zgomotelor (sunt filtre de tip „trece-jos”/”low-pass” care permit trecerea doar a frecvențelor joase – vezi notele de curs). Efectul lor este o mediere a pixelului curent cu valorile vecinilor săi, observabilă prin netezirea (“blur”) a imaginii de ieșire. Aceste nuclee au doar elemente pozitive. Din acest motiv, o practică curentă este

împărțirea rezultatului convoluției cu suma elementelor nucleului de convoluție cu scopul de a scala rezultatul în domeniul de valori al intensității pixelilor din imaginea de ieșire:

$$I_D(i,j) = \frac{1}{c}\sum_{u=0}^{w-1}\sum_{v=0}^{w-1} H(u,v)\cdot I_S(i+u-k, j+v-k) \tag{9.3}$$

unde:

$$c = \sum_{u=0}^{w-1}\sum_{v=0}^{w-1} H(u,v) \tag{9.4}$$

Exemple:

Filtrul medie aritmetică (3x3):

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{9.5}$$

Filtrul gaussian (3x3):

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \tag{9.6}$$

a) b) c)

Fig. 9.2 a) Imaginea originală; b) Rezultatul obținut în urma filtrării de tip medie aritmetică cu un nucleu de dimensiune 3x3; c) Rezultatul obținut în urma filtrării de tip medie aritmetică cu un nucleu de 5x5

9.2.2. Filtre de tip „trece-sus”

Operația de convoluție cu nuclee de acest tip are ca efect punerea în evidentă a zonelor din imagine în care există variații bruște ale intensității pixelilor (cum sunt de exemplu muchiile). Ele realizează o filtrare de tip „trece-sus” (vor permite doar trecerea frecvențelor înalte din imagine – vezi note de curs).

Nucleele folosite pentru detecția punctelor de muchii au suma elementelor componente egală cu 0:

Filtre Laplace (detecție de muchii) (3x3):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{9.7}$$

sau

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{9.8}$$

Filtre high-pass (trece-sus) (3x3):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{9.9}$$

sau

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{9.10}$$

a) b) c)

Fig. 9.3 a) Rezultatul aplicării filtrului Laplace de detecție a muchiilor (9.8) pe imaginea originală (Fig. 9.2a); b) Rezultatul aplicării filtrului Laplace de detecție a muchiilor (9.8) pe imaginea din Fig. 9.2b (filtrată în prealabil cu filtrul medie aritmetică); c) Rezultatul obținut în urma filtrării de tip high-pass cu nucleul (9.10)

9.3. Filtrarea imaginilor în domeniul frecvențial

Transformata Fourier discretă (DFT) unidimensională a unui șir format din N numere reale sau complexe este un șir de N numere complexe, date de:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi jkn}{N}}, \quad k = \overline{0 \ldots N-1} \tag{9.11}$$

Inversa transformatei Fourier discrete (IDFT) este dată de:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi jkn}{N}}, \quad n = \overline{0 \ldots N-1} \tag{9.12}$$

Transformata Fourier discretă bidimensională este obținută prin aplicarea DFT unidimensionale pe fiecare rând al imaginii de intrare și apoi pe fiecare coloană a rezultatului obținut la aplicarea pe linii.

Transformata inversă este obținută prin aplicarea IDFT unidimensionale pe fiecare coloană a imaginii DFT și apoi pe fiecare linie a rezultatului precedent. Setul de numere complexe rezultat după aplicarea DFT poate fi reprezentat și în coordonate polare (magnitudine și fază). Mulțimea magnitudinilor (numere reale) reprezintă spectrul de frecvență (frequency power spectrum) al șirului original.

DFT și inversa ei sunt realizate folosind o abordare recursivă a transformatei Fourier rapide (Fast Fourier Transform), care reduce timpul de calcul de la $O(n^2)$ la $O(n \ln n)$ fapt care reprezintă o creștere a vitezei de calcul, mai ales în cazul procesării imaginilor bidimensionale, la care o complexitate de $O(n^2m^2)$ ar fi foarte mare în raport cu complexitatea aproape liniară, în număr de pixeli, de $O(nm\ln(nm))$ dată de transformata Fourier rapidă (FFT).

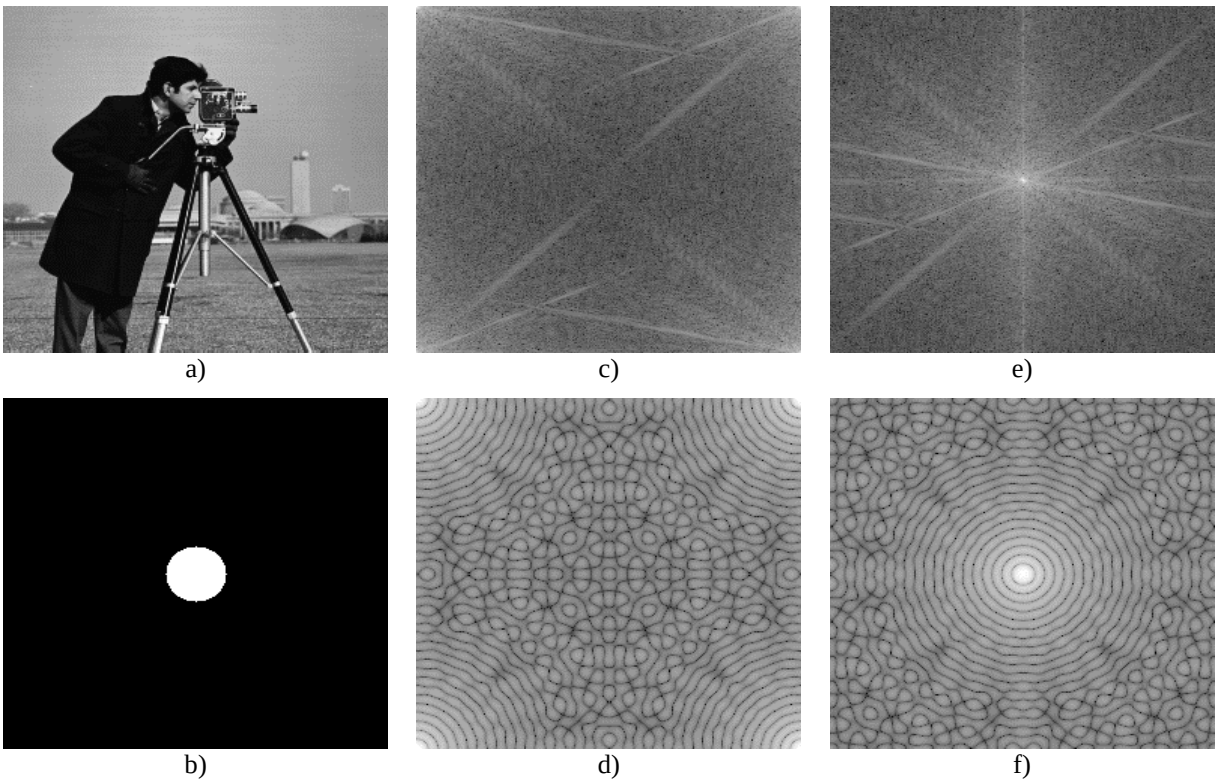


Fig. 9.4 a) și b) Imagini originale; c) și d) Logaritmul spectrului magnitudinii; e) și f) Logaritm centrat al spectrului magnitudinii

9.3.1. Aliasing

Fenomenul de aliasing este o consecință a limitei frecvenței Nyquist (un semnal eșantionat nu poate reprezenta frecvențe mai mari decât jumătate din frecvența de eșantionare). Astfel, jumătatea de sus a reprezentării în domeniul de frecvență este redundantă. Acest lucru poate fi observat din identitatea:

$$X_k = X_{N-k}^* \tag{9.13}$$

(unde * se referă la conjugata complexă) care este adevărată dacă numerele x_k din șirul de intrare sunt reale. Astfel, spectrul tipic Fourier 1D va conține componentele de frecvență joasă, atât în partea de jos cât și în partea de sus, iar frecvențele înalte sunt localizate simetric în raport cu centrul. În spațiul 2D componentele de frecvență joasă vor fi localizate lângă colțurile imaginii, iar componentele de frecvență înaltă vor fi în centru (vezi Fig. 9.4c, d). Acest lucru face ca spectrul să

fie destul de greu de citit și de interpretat. Pentru a centra componentele de frecvență joasă în mijlocul spectrului, ca prim pas, ar trebui realizată următoarea transformare a datelor de intrare:

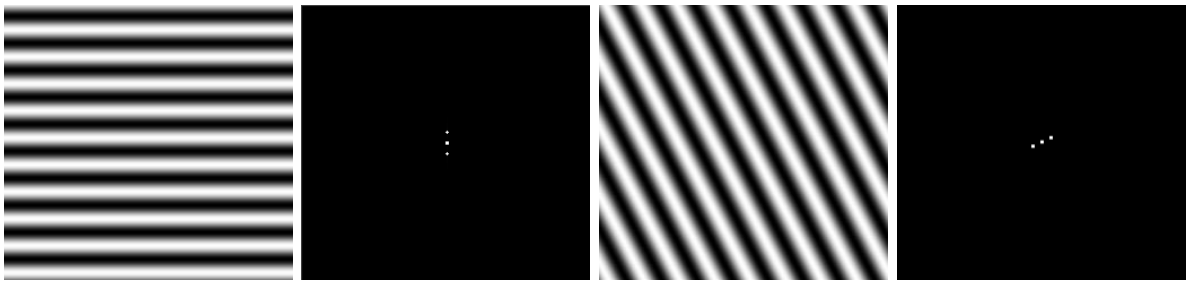
$$x_k \leftarrow (-1)^k x_k \tag{9.14}$$

În spațiul 2D transformarea de centrare devine:

$$x_{uv} \leftarrow (-1)^{u+v} x_{uv} \tag{9.15}$$

După aplicarea acestei transformări în spațiul 1D spectrul va conține în mijloc componentele de frecvență joasă, iar componentele de frecvență înaltă vor fi localizate simetric spre capetele stâng și drept ale spectrului. În 2D componentele de frecvență joasă vor fi localizate în mijlocul imaginii, în timp ce diversele componente de frecvență înaltă vor fi localizate spre muchii.

Magnitudinile localizate pe orice linie care trece prin centrul imaginii DFT reprezintă componentele 1D din spectrul de frecvență al imaginii originale, pe direcția liniei. Toate liniile de acest fel sunt simetrice în raport cu mijlocul (centrul imaginii).



a) b) c) d)

Fig. 9.5 Transformate Fourier ale imaginilor cu unde sinusoidale a) și c). Punctul de centru în b) și d) reprezintă componenta continuă, celelalte două puncte simetrice se datorează frecvenței undelor sinusoidale

9.3.2. Filtre ideale de tip „trece-jos” și „trece-sus”, în domeniul frecvențial

Operația de convoluție în domeniul spațial este echivalentă cu înmulțirea scalară în domeniul frecvențial. Astfel, pentru nuclee de convoluție mari, este mai convenabil din punct de vedere computațional să se realizeze operația de convoluție în domeniul frecvențial.

Algoritmul de filtrare în domeniul frecvențial este următorul:

a) Se realizează transformata de centrare a imaginii pe imaginea originală (9.15).
b) Se realizează transformata DFT.
c) Se schimbă coeficienții Fourier în funcție de filtrarea dorită.
d) Se realizează transformata IDFT.
e) Se realizează transformata de centrare a imaginii (anulează efectul primei centrări a imaginii).

Un filtru ideal de tip „trece-jos” va modifica toți coeficienții Fourier care sunt mai departe de centrul imaginii *(W/2, H/2)* decât o distanță R dată. Acești coeficienți vor primi valoarea 0 (*W* este lățimea imaginii și *H* este înălțimea imaginii):

$$X'_{uv} = \begin{cases} X_{uv}\,, \left(\frac{H}{2}-u\right)^2 + \left(\frac{W}{2}-v\right)^2 \le R^2 \\ 0\,, \quad \left(\frac{H}{2}-u\right)^2 + \left(\frac{W}{2}-v\right)^2 > R^2 \end{cases} \tag{9.16}$$

Un filtru ideal de tip „trece-sus” va schimba în 0 toți coeficienții Fourier aflați la o distanță mai mică decât R față de centrul imaginii (*W*/2, *H*/2):

$$X'_{uv} = \begin{cases} X_{uv}\,, & \left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2 > R^2 \\ 0\,, & \left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2 \le R^2 \end{cases} \tag{9.17}$$

Rezultatele aplicării unui filtru ideal de tip „trece-jos” și de tip „trece-sus” sunt prezentate în Fig. 9.6 b) și c). Din păcate, filtrele spațiale corespunzătoare din Fig. 9.6 e) și g) nu sunt FIR (au un suport infinit) și oscilează îndepărtându-se de centrele lor. Din această cauză imaginile rezultate după aplicarea celor două filtre (trece-sus și trece-jos) au un aspect de undă circulară. Pentru a corecta acest lucru, tăierea (suprimarea) în domeniul frecvențial trebuie să fie mai netedă, așa cum este prezentat în secțiunea următoare.

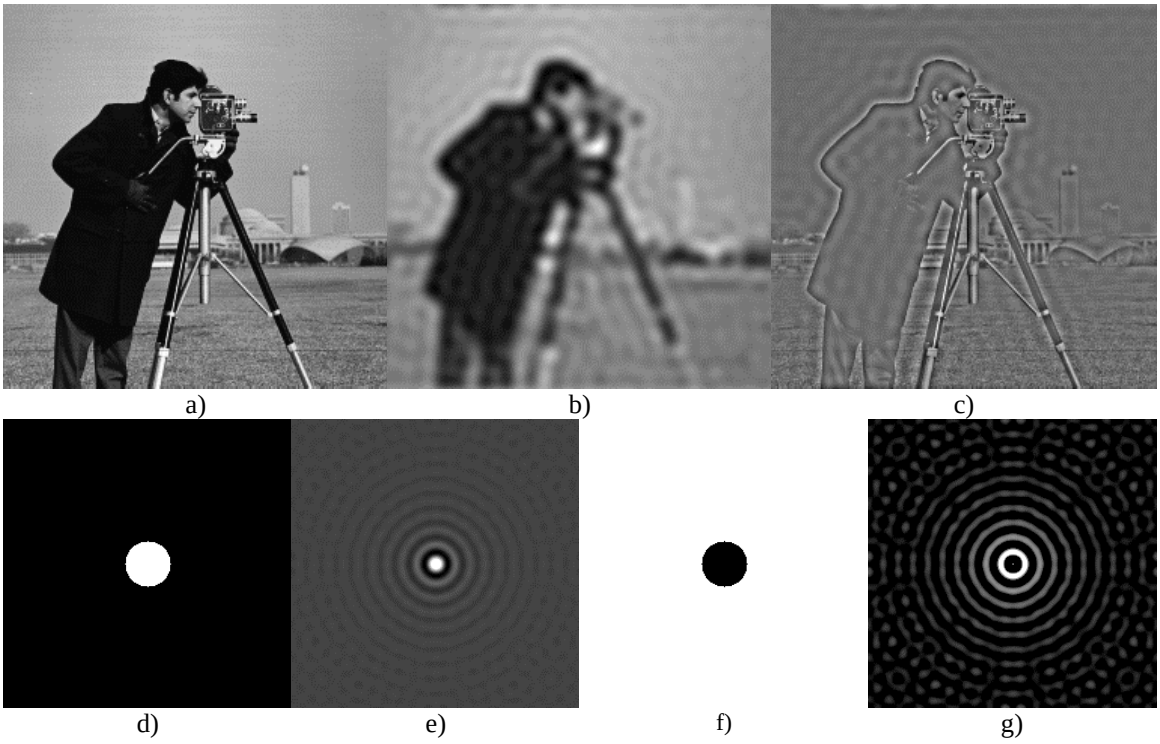


Fig. 9.6 a) Imaginea originală; b) Rezultatul aplicării filtrului ideal de tip „trece-jos”; c) Rezultatul aplicării filtrului ideal de tip „trece-sus”; d) Filtru ideal de tip „trece-jos” în domeniul frecvențial; e) Filtrul ideal de tip „trece-jos” corespunzător în domeniul spațial; f) Filtru ideal de tip „trece-sus” în domeniul frecvențial; g) Filtrul ideal de tip „trece-sus” corespunzător în domeniul spațial (R = 20)

9.3.3. Filtru Gaussian de tip „trece-jos” și „trece-sus” în domeniul frecvențial

În cazul filtrului de tip Gauss, coeficienții de frecvență nu sunt tăiați brusc, ci este folosit un proces de suprimare mai netedă. Acest proces ține cont și de faptul că DFT a unei funcții de tip Gauss este tot o funcție de tip Gauss. (Fig. 9.7d-g).

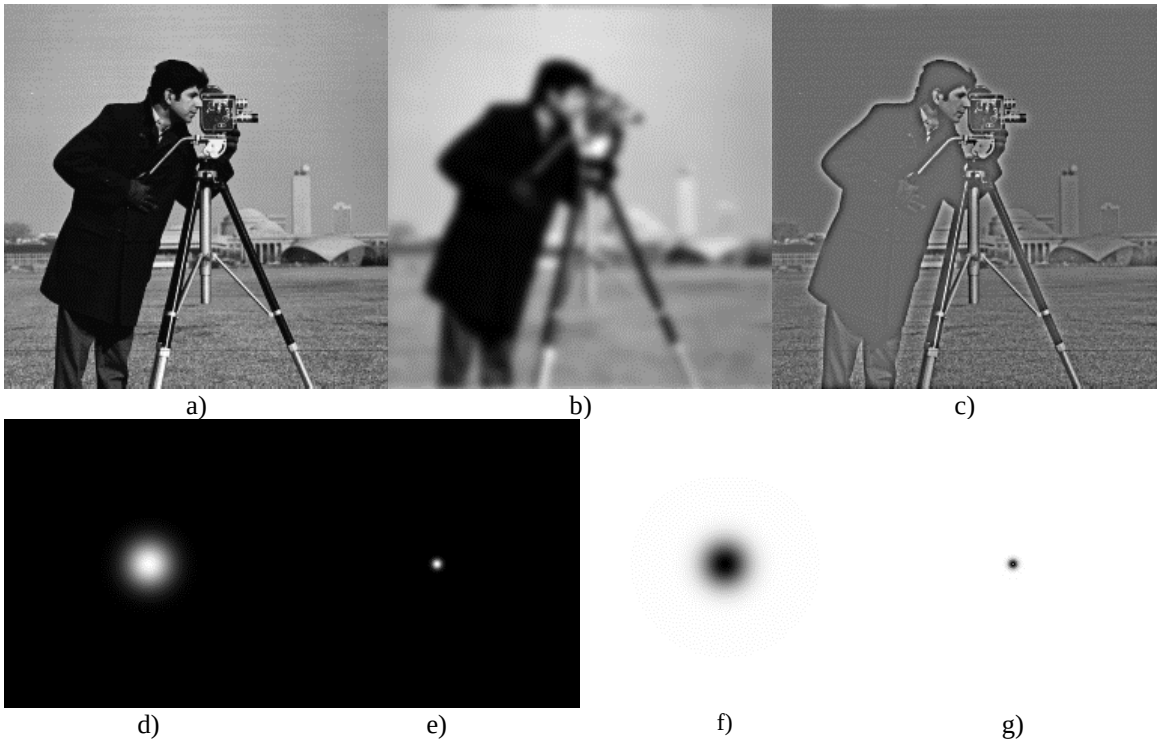
Filtrul Gaussian de tip „trece-jos” atenuează componentele din domeniul de frecvență care sunt mai îndepărtate față de centrul imaginii ($W/2$, $H/2$). $A \sim \frac{1}{\sigma}$ unde σ este deviația standard a filtrului Gaussian echivalent în domeniul spațial.

$$X'_{uv} = X_{uv} e^{-\frac{\left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2}{A^2}} \tag{9.18}$$

Filtrul Gaussian de tip „trece-sus” atenuează componentele de frecvență care sunt aproape de centrul imaginii (*W*/2, *H*/2):

$$X'_{uv} = X_{uv}\left(1 - e^{-\frac{\left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2}{A^2}}\right) \tag{9.19}$$

Fig. 9.7 arată rezultatele aplicării unui filtru de tip Gauss. A se remarca faptul că efectul de unde circulare vizibil în Fig. 9.6 a dispărut.



a) b) c)

d) e) f) g)

Fig. 9.7 a) Imaginea originală; b) Rezultatul aplicării unui filtru Gaussian de tip „trece-jos”; c) Rezultatul aplicării unui filtru Gaussian de tip „trece-sus”; d) Filtru Gaussian de tip „trece-jos” în domeniul frecvențial; e) Filtru Gaussian corespunzător de tip „trece-jos” în domeniul spațial; f) Filtru Gaussian de tip „trece-sus” în domeniul frecvențial; g) Filtru Gaussian corespunzător de tip „trece-sus” în domeniul spațial (*A* = 20)

9.4. Detalii de implementare

9.4.1. Filtre în domeniul spațial

Filtrele de tip „trece-jos” vor avea întotdeauna coeficienți pozitivi și astfel imaginea rezultată după aplicarea filtrului va conține valori pozitive. Trebuie să vă asigurați că imaginea rezultată are valori cuprinse în intervalul dorit (în cazul nostru 0-255). Pentru a realiza acest lucru, trebuie să vă asigurați că suma coeficienților din filtrul trece jos este 1. Dacă folosiți operații întregi să fiți atenți la ordinea operațiilor! În mod normal, împărțirea este ultima operație care ar trebui efectuată, pentru a minimiza erorile datorate rotunjirii.

Filtrele de tip „trece-sus” vor avea coeficienți pozitivi și negativi. Trebuie să vă asigurați că valorile din imaginea rezultat sunt numere întregi cuprinse în intervalul 0 și 255!

Există trei posibilități pentru a vă asigura că imaginea rezultat este în intervalul dorit. Prima metodă presupune calculul următor:

$$S_+ = \sum_{F_k>0} F_k, \quad S_- = \sum_{F_k<0} -F_k,$$
$$S = \frac{1}{2\max\{S_+, S_-\}} \tag{9.20}$$
$$I_D(u,v) = S(F * I_S)(u,v) + \left\lfloor \frac{L}{2} \right\rfloor$$

În formula de mai sus S_+ reprezintă suma coeficienților pozitivi din filtru și S_- reprezintă suma valorilor absolute a coeficienților negativi. Rezultatul operației $F*I_S$, adică al aplicării filtrului de tip „trece-sus" este întotdeauna în intervalul $[-LS_-, LS_+]$ unde L este nivelul maxim de gri din imagine (255). Înmulțirea cu S va plasa rezultatul în intervalul $[-L/2, L/2]$, iar adunarea cu $L/2$ va muta rezultatul în intervalul $[0, L]$.

O altă abordare constă în realizarea tuturor operațiilor folosind numere întregi cu semn (rezultă imaginea $F*I_S$), apoi determinarea valorii minime (min) și maxime (max) din rezultat (din $F*I_S$) urmată de o transformare liniară a valorilor rezultate folosind formula:

$$I_D(u,v) = L \cdot \frac{(F * I_S)(u,v) - \min}{\max - \min} \tag{9.21}$$

A treia abordare calculează magnitudinea rezultatului și saturează toate valorile care depășesc domeniul de reprezentare $[0, L]$.

9.4.2. Filtre în domeniul frecvențial

Pentru vizualizare și procesare în domeniul frecvențial este util să considerăm o reprezentare care conține coeficientul (0,0) în centrul imaginii. Acest lucru se poate realiza prin interschimbarea cadranelor opuse din transformata Fourier. În mod echivalent se poate preprocesa imaginea de intrare folosind (9.15). Filtrul prezentat mai jos folosește următoarea funcție pentru a realiza această operație de centrare atât pe imaginea de intrare cât și pe rezultat.

```
void centering_transform(Mat img){
        // imaginea trebuie să aibă elemente de tip float
        for (int i = 0; i < img.rows; i++){
             for (int j = 0; j < img.cols; j++){
                img.at<float>(i, j) = ((i + j) & 1) ? -img.at<float>(i, j) : img.at<float>(i, j);
             }
        }
}
```

Librăria OpenCV conține o funcție pentru calcularea Transformării Fourier Discrete. Următorul cod șablon ilustrează cum se face atât transformarea directă cât și cea inversă. Procesările în domeniul frecvențial se vor realiza pe canalul de magnitudine. Fiindcă DFT-ul lucrează cu imagini care au dimensiuni egale cu puteri ale lui doi, se va lucra cu imaginea *cameraman.bmp*.

```
Mat generic_frequency_domain_filter(Mat src){
        //imaginea trebuie să aibă elemente de tip float
        Mat srcf;
        src.convertTo(srcf, CV_32FC1);

        //transformarea de centrare
        centering_transform(srcf);

        //aplicarea transformatei Fourier, se obține o imagine cu valori numere complexe
        Mat fourier;
        dft(srcf, fourier, DFT_COMPLEX_OUTPUT);
        //divizare în două canale: partea reală și partea imaginară
```

```
        Mat channels[] = { Mat::zeros(src.size(), CV_32F), Mat::zeros(src.size(), CV_32F) };
        split(fourier, channels);  // channels[0] = Re(DFT(I)), channels[1] = Im(DFT(I))

        //calcularea magnitudinii și fazei în imaginile mag, respectiv phi, cu elemente de tip float
        Mat mag, phi;
        magnitude(channels[0], channels[1], mag);
        phase(channels[0], channels[1], phi);

        //aici afișați imaginile cu fazele și magnitudinile
        // ......

        //aici inserați operații de filtrare aplicate pe coeficienții Fourier
        // ......

        //memorați partea reală în channels[0] și partea imaginară în channels[1]
        // ......

        //aplicarea transformatei Fourier inversă și punerea rezultatului în dstf
        Mat dst, dstf;
        merge(channels, 2, fourier);
        dft(fourier, dstf, DFT_INVERSE | DFT_REAL_OUTPUT | DFT_SCALE);

        //transformarea de centrare inversă
        centering_transform(dstf);

        //normalizarea rezultatului în imaginea destinație
        normalize(dstf, dst, 0, 255, NORM_MINMAX, CV_8UC1);
        //Notă: normalizarea distorsionează rezultatul oferind o afișare îmbunătățită în intervalul
        //[0,255]. Dacă se dorește afișarea rezultatului cu exactitate (vezi Activitatea 3) se va
        //folosi în loc de normalizare conversia:
        //dstf.convertTo(dst, CV_8UC1);

        return dst;
}
```

9.5. Activități practice

1. Implementați un filtru general care realizează operația de convoluție cu un nucleu de convoluție oarecare. Coeficientul de scalare trebuie calculat automat. Acesta este inversul sumei coeficienților pentru filtre trece-jos, iar pentru filtrele trece-sus se calculează folosind ecuația (9.20).
2. Testați filtrul folosind nucleele din ecuațiile (9.5) … (9.10)
3. Studiați funcția șablon dată pentru un filtru în domeniul frecvențial. Realizați trecerea unei imagini sursă din domeniul spațial în domeniul frecvențial aplicând transformata Fourier directă (DFT), iar apoi aplicați transformarea Fourier inversă (IDFT) pe coeficienții spectrului Fourier obținuți și verificați că imaginea rezultată este identică cu imaginea sursă.
4. Adăugați o funcție de procesare care calculează și afișează logaritmul magnitudinii transformatei Fourier a unei imagini de intrare. Adăugați 1 la magnitudine pentru a evita *log*(0).
5. Adăugați funcții de procesare care aplică: un filtru ideal trece-jos, un filtru ideal trece-sus, un filtru Gaussian trece-jos și un filtru Gaussian trece-sus în domeniul frecvențial utilizând ecuațiile (9.16) … (9.19).
6. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

9.6. Bibliografie

[1] Umbaugh Scot E., *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8.
[2] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.

10. Modelarea și eliminarea zgomotelor din imaginile digitale

10.1. Introducere

Zgomotul este o informație nedorită care deteriorează calitatea unei imagini. El se definește ca orice proces (*n*) care afectează imaginea achiziționată în memorie (*f*) și nu face parte din scenă (semnalul inițial - *s*). În conformitate cu modelul de zgomot aditiv, acest proces se poate scrie:

$$f(i, j) = s(i, j) + n(i, j) \tag{10.1}$$

Zgomotul din imaginile digitale poate proveni dintr-o mulțime de surse. Procesul de achiziție al imaginii digitale, care convertește un semnal optic într-un semnal electric și apoi într-unul digital este un proces prin care zgomotele apar în imagini digitale. La fiecare pas din acest proces există fluctuații cauzate de fenomene naturale și acestea adaugă o valoare aleatoare la extragerea fiecărei valori a luminozității pentru un pixel dat.

10.2. Modelarea zgomotelor

Zgomotul (*n*) poate fi modelat fie printr-o histogramă sau o funcție a densității de probabilitate care se suprapune peste cea a imaginii originale (*s*). În continuare se vor prezenta modele pentru cele mai des întâlnite tipuri de zgomote: zgomotul de tip *sare-și-piper* (*salt&pepper*) și zgomotul de tip gaussian. În literatură mai există și alte modele cum ar fi modelul exponențial negativ, modelul gamma/Erlang, modelul Ralyeigh (vezi note de curs!).

10.2.1. Zgomotul sare-și-piper (salt&pepper)

În modelul de zgomot de tip *salt&pepper* există doar două valori posibile, *a* și *b*, și probabilitatea de apariție a fiecăruia este mai mică de 0.1 (la valori mai mari, zgomotul va domina imaginea). Pentru o imagine cu 8 biți/pixel, valoarea de intensitate tipică pentru *zgomotul pepper* este apropiată de 0 și pentru *zgomotul salt* este apropiată de 255.

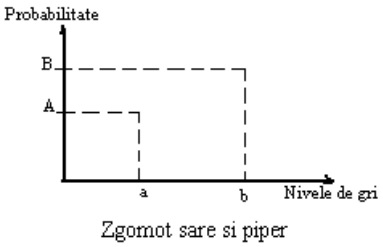


Fig. 10.1 Funcția densității de probabilitate a modelului de zgomot *salt&pepper*

$$FDP_{sare\&piper} = \begin{cases} A, \ pentru \ g = a \ ("piper") \\ B, \ pentru \ g = b \ ("sare") \end{cases} \tag{10.2}$$

Zgomotul salt&pepper este în general cauzat de funcționarea proastă a celulelor din senzorii camerelor sau de greșeli ale locațiilor de memorie sau de erori de sincronizare în procesul de digitizare sau de erori de transmisie.

10.2.2. Zgomotul gaussian

Este un zgomot a cărui funcție de densitate de probabilitate are o formă gaussiană:

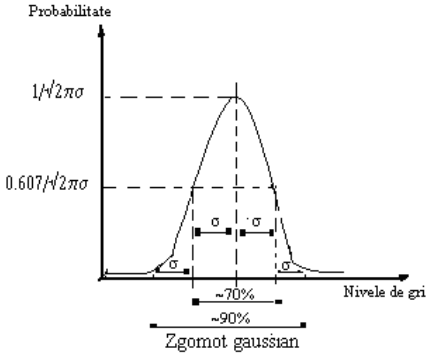


Fig. 10.2 Funcția densității de probabilitate a modelului de zgomot gaussian

$$FDP_{Gaussian} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(g-\mu)^2}{2\sigma^2}} \tag{10.3}$$

unde:

g = nivel de gri;
μ = media;
σ = deviația standard.

Aproximativ 70% din valori sunt încadrate între $\mu \pm \sigma$ iar 90% dintre valori sunt cuprinse între $\mu \pm 2\sigma$. Deși din punct de vedere teoretic ecuația definește valori cuprinse între -∞ și +∞, valorile FDP gaussiene se pot considera nule dincolo de intervalul $\mu \pm 3\sigma$.

Zgomotul gaussian este folosit pentru modelarea proceselor naturale care introduc zgomote (ex: zgomotul datorat naturii discrete a radiației și procesului de conversie a semnalului optic în semnal electric – detector/shot noise, zgomotul electric din timpul procesului de achiziție – amplificarea semnalului electric generat de senzori, etc.).

10.3. Eliminarea zgomotelor cu ajutorul filtrelor spațiale

10.3.1. Filtre ordonate (neliniare)

Filtrele ordonate sunt bazate pe un tip specific de statistică a imaginilor numită **statistică ordonată**. Ele se numesc neliniare pentru că nu se pot aplica printr-un operator liniar (așa cum este cazul operatorului de convoluție). Aceste filtre operează tot pe *ferestre* mici și înlocuiesc valoarea pixelului central (similar cu procesul de convoluție). Statistica ordonată este o tehnică care aranjează toți pixelii într-o ordine secvențială bazată pe valoarea nivelurilor de gri. Poziția unui element în cadrul acestei mulțimi ordonate va fi caracterizată de un *rang.* Dându-se o fereastră *W* de *N*x*N* pixeli, valorile pixelilor pot fi ordonate crescător după cum urmează:

$$I_1 \le I_2 \le I_3 \le \ldots \le I_{N^2} \tag{10.4}$$

unde:

$\{ I_1, I_2, I_3, \ldots, I_{N^2} \}$ reprezintă mulțimea valorilor intensităților corespunzătoare subsetului de pixeli din imagine, care sunt în fereastra *W* de *N*x*N* pixeli.

Exemplu: Dându-se o fereastră de dimensiune 3x3:

$$\begin{bmatrix} 110 & 110 & 114 \\ 100 & 106 & 104 \\ 95 & 88 & 85 \end{bmatrix}$$

rezultatul aplicării statisticii ordonate va fi:

{85, 88, 95, 100, 104, 106, 110, 110, 114}

Filtrul median: selectează valoarea de mijloc a unui pixel dintr-o mulțime ordonată și îl înlocuiește în poziția corespunzătoare din imaginea destinație. În exemplul de mai sus valoarea selectată ar fi 104. Filtrul median permite eliminarea *zgomotului de tip salt&pepper*.

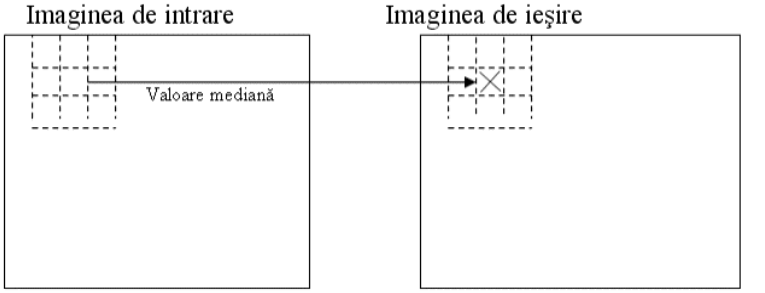


Fig. 10.3 Aplicarea filtrului median

Filtrul de maxim: selectează cea mai mare valoare dintr-o fereastră ordonată de valori ale pixelilor. În exemplul de mai sus valoarea selectată ar fi 114. Acest filtru poate fi folosit pentru eliminarea *zgomotului de tip pepper*, dar aplicat pentru imagini cu zgomot de tip *salt&pepper* amplifică zgomotul de tip *salt*.

Filtrul de minim: selectează cea mai mică valoare dintr-o fereastră ordonată de valori ale pixelilor. În exemplul de mai sus valoarea selectată ar fi 85. Acest filtru poate fi folosit pentru eliminarea *zgomotelor de tip salt* dar aplicat pentru imagini cu zgomot de tip *salt&pepper* amplifică zgomotul de tip *pepper*.

10.3.2. Filtre liniare

Aceste filtre se aplică prin operația de convoluție (operație liniară) cu un nucleu de convoluție/filtru de tip trece jos (vezi lucrarea 9!). În continuare se va prezenta modul de calcul al elementelor unui nucleu de convoluție folosit la eliminarea zgomotului de tip gaussian.

10.3.3. Proiectarea unui nucleu de convoluție gaussian de dimensiune variabilă

Eliminarea zgomotului gaussian trebuie făcută cu un filtru având o formă și o dimensiune adecvate, în concordanță cu deviația standard σ a zgomotului gaussian care afectează imaginea (vezi Fig. 10.2). Dimensiunea *w* unui astfel de filtru se alege de obicei de 6σ (exemplu: pentru un zgomot gaussian cu σ=0.8 $\Rightarrow$ *w* = 4.8 $\approx$ 5).

Construcția elementelor unui astfel de nucleu/filtru gaussian G se va face cu formula de mai jos:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}} \tag{10.5}$$

unde:

(x_0, y_0) – sunt coordonatele coloanei și liniei centrale a nucleului (vezi Fig. 10.4).

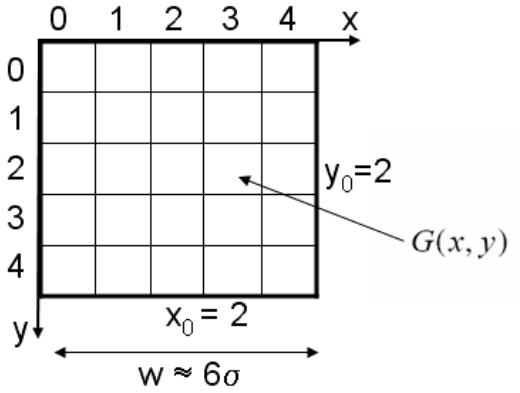


Fig. 10.4 Exemplu de construire al unui nucleu/filtru gaussian *G* de dimensiune 5x5

10.3.4. Filtrarea/restaurarea imaginii

Se realizează prin convoluția imaginii sursă cu nucleul/filtrul gaussian calculat anterior:

$$I_D = G * I_S \tag{10.6}$$

În cazul în care dimensiunea *w* a filtrului este mare, operația de convoluție poate fi costisitoare (*w* x *w* înmulțiri pentru fiecare pixel). În acest caz se poate folosi proprietatea de separabilitate a funcției gaussiene:

$$G(x,y) = G(x)G(y) \tag{10.7}$$

și înlocuirea convoluției cu un nucleu bidimensional cu 2 convoluții cu câte un nucleu unidimensional G_x și G_y (Fig. 10.5):

$$I_D = \left(G_x G_y\right) * I_S = G_x * \left(G_y * I_S\right) \tag{10.8}$$

unde:

G_x și G_y sunt vectorii unidimensionali (Fig. 10.5):

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-x_0)^2}{2\sigma^2}} \tag{10.9}$$

$$G(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-y_0)^2}{2\sigma^2}} \tag{10.10}$$

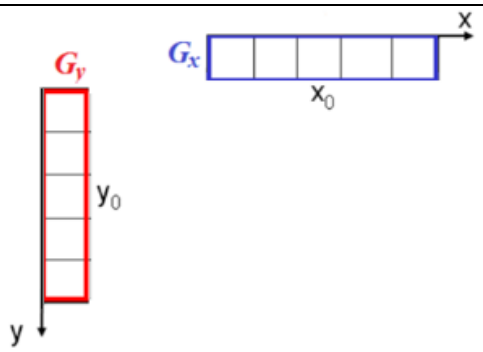


Fig. 10.5 Ilustrarea celor două componente vectoriale G_x și G_y în care poate fi separat un nucleu gaussian bidimensional

În acest caz numărul de înmulțiri efectuate pentru fiecare pixel este *w* pentru fiecare dintre cele două parcurgeri/convoluții ale imaginii.

10.4. Calcularea și afișarea timpului de procesare

```
double t = (double)getTickCount(); // Găsește timpul curent în [CPU cycles]
// … Procesarea propriu-zisă …
// Găsește timpul curent din nou în [CPU cycles] și calculează timpul scurs
în [sec]
t = ((double)getTickCount() - t) / getTickFrequency();
// Afișarea la consolă a timpului de procesare în [ms]
printf("Time = %.3f [ms]\n", t * 1000);
```

10.5. Activități practice

1. Se va implementa un filtru median de dimensiune *w* variabilă (*w* = 3, 5 sau 7), specificată de utilizator. Afișați timpul de procesare.
2. Se va implementa operația de filtrare cu un nucleu gaussian bidimensional cu dimensiunea *w* variabilă (*w* = 3, 5 sau 7), specificată de utilizator. Valorile componentelor nucleului gaussian se vor calcula automat în funcție de σ ($\sigma = w/6$), în conformitate cu (10.5). Afișați timpul de procesare. Comparați timpii de procesare obținuți pentru *w* variabil.
3. Se va implementa operația de filtrare cu un nucleu gaussian separat în cele două componente vectoriale G_x și G_y cu dimensiunea *w* variabilă (*w* = 3, 5 sau 7), specificată de utilizator. Valorile componentelor vectorilor G_x și G_y se vor calcula automat în funcție de σ ($\sigma = w/6$), în conformitate cu (10.9) și (10.10). Afișați timpul de procesare. Comparați timpii de procesare obținuți cu cele două metode de aplicare a filtrului gaussian: filtru bidimensional vs. filtru vectorial.
4. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

10.6. Bibliografie

[1] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.

11. Detecția punctelor de muchie

11.1. Introducere

În această lucrare se va aborda problematica detecției punctelor de muchie din imagini digitale. Punctele de muchie sunt puncte în jurul cărora intensitatea imaginii suferă un salt brusc de-a lungul unei anumite direcții ‚*x*' (Fig. 11.1). Această variație de intensitate poate fi identificată prin detecția punctelor de maxim ale derivatei de ordin 1 a imaginii (gradientului: $\nabla f = f'$) sau prin detectarea trecerilor prin ‚0' ale derivatei de ordin 2 a imaginii (laplacianului: $\nabla^2 f = f''$).

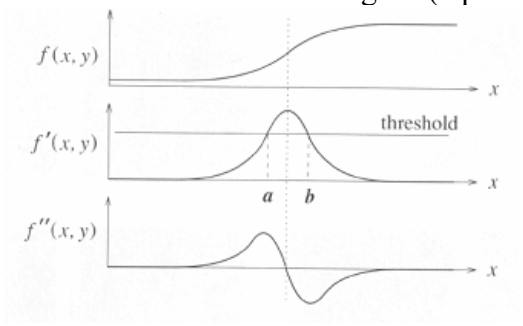


Fig. 11.1 Modalități de detecție a punctelor de muchie (zonele în care intensitatea variază brusc)

În această lucrare se vor prezenta metode bazate pe detecția punctelor de maxim ale derivatei de ordin 1 (gradientul imaginii : $\nabla f(x,y)$).

11.2. Calculul gradientului imaginii

Gradientul într-un punct al imaginii este un vector care indică direcția de variație a intensității imaginii (Fig. 11.2) în jurul acelui punct, modulul său fiind proporțional cu amplitudinea acestei variații raportată la unitatea de lungime (viteza acestei variații) (Fig. 11.1). Dacă punctele de muchie fac parte dintr-un contur (ca și în Fig. 11.2), gradientul într-un punct de muchie al acestui contur va fi perpendicular pe tangenta la acest contur în punctul respectiv.

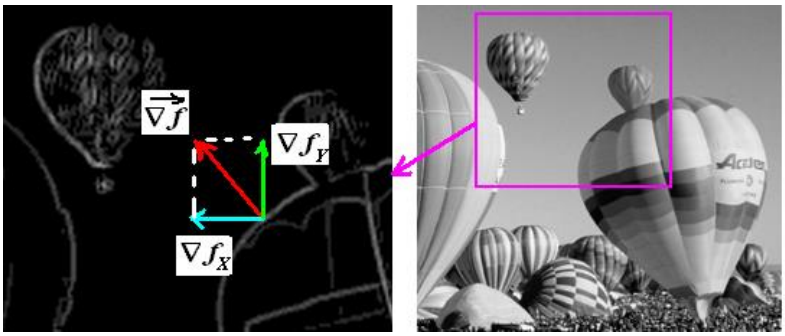


Fig. 11.2 Ilustrarea semnificației gradientului imaginii. În figura din stânga este reprezentată imaginea modulului gradientului

Gradientul unei funcții continue *f* de 2 variabile se definește ca:

$$\nabla f(x,y) = \begin{bmatrix} \nabla f_x \\ \nabla f_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} \lim\limits_{\Delta x \to 0} \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x} \\ \lim\limits_{\Delta y \to 0} \frac{f(x, y+\Delta y) - f(x,y)}{\Delta y} \end{bmatrix} \tag{11.1}$$

Pentru imagini digitale gradientul se poate aproxima înlocuind Δx și Δy cu 1 în (11.1):

$$\nabla f(x,y) = \begin{bmatrix} \nabla f_x \\ \nabla f_y \end{bmatrix} = \begin{bmatrix} f[x+1,y] - f[x,y] \\ f[x,y+1] - f[x,y] \end{bmatrix} \tag{11.2}$$

Alte aproximări ale celor două componente ale gradientului se pot obține prin convoluția imaginii cu următoarele nuclee:

Prewitt:

$$\begin{aligned} \nabla f_x &= f(x,y) * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \\ \nabla f_y &= f(x,y) * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \end{aligned} \tag{11.3}$$

Sobel:

$$\begin{aligned} \nabla f_x &= f(x,y) * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\ \nabla f_y &= f(x,y) * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \end{aligned} \tag{11.4}$$

Roberts (cross):

$$\begin{aligned} \nabla f_x &= f(x,y) * \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \nabla f_y &= f(x,y) * \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \end{aligned} \tag{11.5}$$

În cazul aplicării convoluției cu filtrul Roberts (cross) de dimensiune 2 x 2, rezultatul convoluției este stocat pe poziția stânga sus a ferestrei.

Ca orice mărime vectorială, gradientul este caracterizat de un modul și o direcție. Modulul gradientului se calculează cu formula (11.6), iar direcția cu formula (11.7).

$$\text{Modul: } |\nabla f(x,y)| = \sqrt{(\nabla f_x(x,y))^2 + (\nabla f_y(x,y))^2} \tag{11.6}$$

$$\text{Direcție: } \theta(x,y) = arctg\left(\frac{\nabla f_y(x,y)}{\nabla f_x(x,y)}\right) \tag{11.7}$$

Pentru Roberts (cross) se adaugă 135^0 ($3\pi/4$) la direcția obținută.

11.3. Metoda Canny de detecție a muchiilor

Metoda propusă de Canny se bazează pe calculul gradientului imaginii, dar în plus permite:

- maximizarea raportului semnal zgomot pentru o detecție corectă;
- o localizare bună a punctelor de muchie;
- minimizarea numărului de răspunsuri pozitive la o singură muchie singulară (eliminarea non-muchiilor).

Pașii metodei propuse de Canny sunt dați mai jos:

1. Filtrarea imaginii cu un filtru Gaussian pentru eliminarea zgomotelor.
2. Calculul modulului și direcției gradientului.
3. Suprimarea non-maximelor modulului gradientului.
4. Binarizarea adaptivă a punctelor de muchie și prelungirea muchiilor prin histereză.

11.3.1. Filtrarea imaginii cu un filtru Gaussian

Zgomotul din imagine este o informație de "frecvență înaltă" care se suprapune peste imaginea originală. Aceasta inducea apariția unor puncte de muchie false. Zgomotul inerent procesului de achiziție al imaginilor are un model Gaussian și se poate elimina cu un filtru omonim. Procedura de filtrare a zgomotului Gaussian a fost prezentată în detaliu în lucrarea 10.

11.3.2. Calculul modulului și direcției gradientului

Calcularea modulului și direcției gradientului presupune alocarea a câte unui buffer temporar de dimensiunea imaginii și inițializarea elementelor lor în conformitate cu (11.6) și respectiv (11.7), unde componentele orizontale $\nabla f_x(x,y)$ și respectiv verticale $\nabla f_y(x,y)$ se pot calcula prin convoluția imaginii în conformitate cu (11.3) sau (11.4).

11.3.3. Suprimarea non-maximelor modulului gradientului

Are drept scop subțierea muchiilor prin păstrarea doar a punctelor de muchie care au modulul maxim al gradientului de-a lungul direcției de variație a intensității (de-a lungul direcției gradientului).

Primul pas constă în cuantificarea direcțiilor gradientului (calculate folosind (11.7)) în 4 intervale de câte 45° în conformitate cu Fig. 11.3:

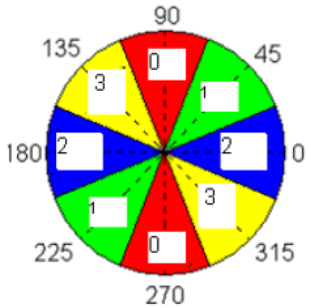


Fig. 11.3 Cuantificarea direcțiilor gradientului în procesul de eliminare a non-maximelor [3]

Presupunând, de exemplu, că direcția gradientului într-un anumit punct al imaginii este „1" (Fig. 11.4) modulul gradientului în punctul *P* este un maxim local dacă: $|\nabla P| > |\nabla I_1|$ și $|\nabla P| > |\nabla I_1'|$. Dacă aceste condiții sunt îndeplinite, punctul *P* se reține. Altfel, se elimină.

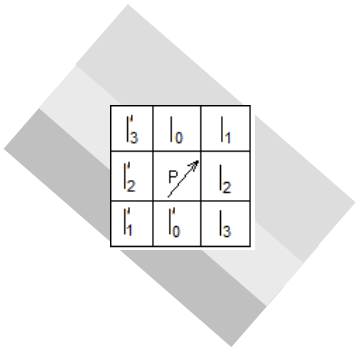


Fig. 11.4 Ilustrarea unui exemplu de calcul al punctului de muchie cu modulul maxim al gradientului

11.3.4. Binarizarea adaptivă a punctelor de muchie și prelungirea muchiilor prin histereză

După efectuarea cu succes a calculului gradientului și a operației de suprimare a non-maximelor modulului gradientului, **se obține o imagine în care intensitatea pixelilor este egală cu valoarea modulului gradientului în acel punct**. De asemenea, grosimea punctelor de muchie din această imagine (a căror modul este nenul) este (ideal) de un pixel. În continuare se descriu pașii pentru obținerea muchiilor finale.

11.3.4.1. Binarizarea adaptivă

Binarizarea adaptivă încearcă să extragă un număr relativ constant de puncte de muchie pentru o dimensiune dată a imaginii. În acest fel se compensează iluminarea și contrastul diferit între imagini (un prag fix sau va scoate prea multe muchii, sau nu va scoate muchii deloc).

Parametrul care se dă procedurii de detecție a pragului este raportul dintre numărul de puncte de muchie și numărul de puncte cu modulul gradientului diferit de zero din imagine.

$$NrPuncteMuchie = p \bullet (NumărPixeli - NumărPixeliCuModulGradientNul) \quad (11.8)$$

Parametrul p ia de obicei valori între 0.01 și 0.1.

Algoritmul este următorul:

1. Se calculează histograma de intensități a imaginii (a modulelor gradientului după suprimarea non-maximelor). Se vor normaliza aceste valori în intervalul [0..255] (prin împărțire la $4\sqrt{2}$ în cazul în care gradientul a fost calculat utilizând operatorul Sobel). Rezultă o histogramă $Hist[0..255]$,

$$Hist[i] = Numărul\ pixelilor\ având\ gradientul\ normalizat\ de\ modul\ i \quad (11.9)$$

2. Se calculează numărul de pixeli diferiți de zero care nu vor fi puncte de muchie:

$$NrNonMuchie = (1-p) \bullet (Height \bullet Width - Hist[0]) \quad (11.10)$$

3. Se pornește de la poziția 1 din histogramă și se însumează valorile histogramei pe măsură ce avansăm spre 255. Dacă la un moment dat suma este mai mare decât *NrNonMuchie*, atunci ne oprim și poziția la care am ajuns este pragul pe care l-am găsit. Această tehnică, intuitiv, ne găsește valoarea intensității (*PragAdaptiv*) sub care se găsesc *NrNonMuchie* pixeli.

Atenție la marginile imaginii (pentru care nu s-a calculat gradientul)! Ele trebuie să fie zero sau să nu fie luate în considerare, deoarece valorile nedefinite în aceste zone pot influența valoarea pragului găsit.

11.3.4.2. Extinderea muchiilor prin histereză

Binarizarea cu pragul adaptiv nu garantează completitudinea muchiilor (părți umbrite ale obiectului sau prezența zgomotelor pot afecta procesul de detecție). Rezultatul va fi o imagine cu foarte multe muchii fragmentate.

De aceea se impune o tehnică de prelungire a muchiilor. Muchiile obținute prin binarizarea inițială cu pragul găsit la punctul anterior, care sunt muchii definitive, se caută a se prelungi cu muchii mai puțin clare, care nu trec testul binarizării cu acest prag, dar ar putea să apară ca muchii la un prag mai scăzut.

Formal, se definesc două praguri:

$$Prag\_înalt = PragAdaptiv \quad (11.11)$$

$$Prag\_coborât = k \bullet Prag\_înalt \quad (11.12)$$

unde k<1 (de exemplu, $k = 0.4$).

Imaginea modulului gradientului se parcurge pixel cu pixel. În imaginea rezultat se marchează pixelii cu intensitatea mai mare decât *Prag_înalt* ca MUCHIE_TARE (ex. 255), iar pixelii cu intensitatea mai mare decât *Prag_coborât* dar mai mică decât *Prag_înalt* se marchează ca MUCHIE_SLABĂ (ex. 128). Pixelii cu intensitatea mai mică decât *Prag_coborât* sunt considerați NON_MUCHII și sunt eliminați. Rezultatul acestei marcări (negativat) se vede în Fig. 11.5 stânga.

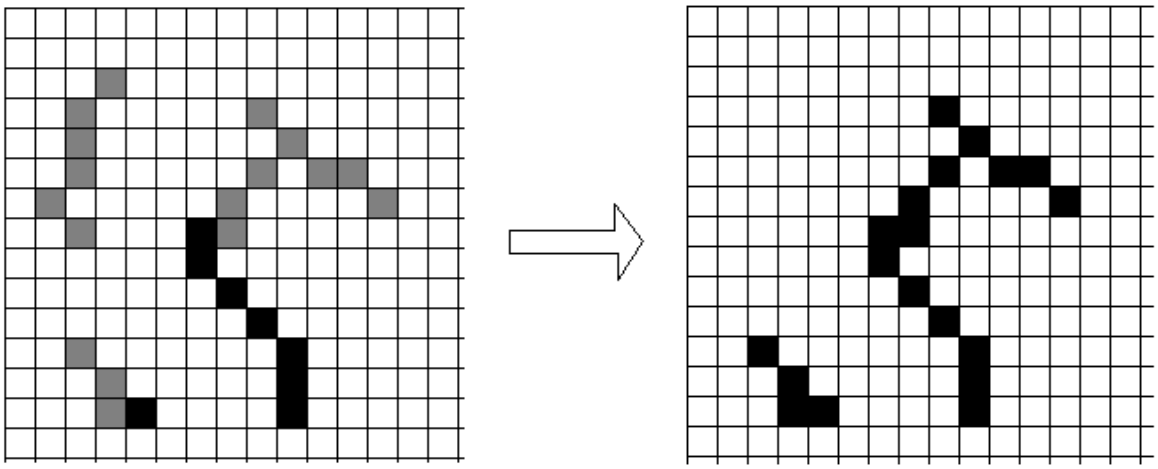


Fig. 11.5 Stânga: imaginea cu muchii tari și slabe; Dreapta: rezultatul prelungirii muchiilor tari și eliminarea celor slabe rămase

Următorul pas este de a se extinde muchiile tari cu pixelii muchiilor slabe învecinate, ca în Fig. 11.5. Dacă un punct de muchie tare se învecinează cu un punct de muchie slabă, punctul de muchie slabă este marcat ca muchie tare. Acest punct la rândul său devine sursă de extindere. Procesul se repetă până nu mai există puncte de muchie tare învecinate cu puncte de muchie slabă.

O implementare eficientă folosește o coadă pentru a ține coordonatele punctelor de muchie tare. Algoritmul rulează astfel:

1. Parcurge imaginea de la stânga la dreapta și de sus în jos și găsește primul punct de MUCHIE_TARE și pune coordonatele sale în coadă.
2. Cât timp coada nu este vidă:
    a) Extrage primul punct din coadă;
    b) Găsește toți vecinii de tip MUCHIE_SLABĂ ai acestui punct;
    c) Marchează în imagine vecinii acestui punct ca puncte de MUCHIE_TARE;
    d) Pune coordonatele acestor puncte în coadă;
    e) Continuă cu următorul punct din coadă.
3. Continuă de la pasul 1 cu următorul punct de MUCHIE_TARE.
4. Elimină toate punctele de MUCHIE_SLABĂ din imagine, dându-le valoarea NON_MUCHIE (0).

O considerație finală se impune la definiția "învecinării" unui punct de muchie tare cu un punct de muchie slabă. Se poate considera învecinarea directă (pixeli adiacenți), în vecinătate de 4 sau de 8, sau se poate tolera o distanță oarecare (de 1, 2 pixeli). Un argument pentru aceasta ar putea fi faptul că, din cauza zgomotului, muchiile pot să fie întrerupte pe distanțe mici.

11.4. Activități practice

Activitățile practice ale aceste lucrări se vor desfășura de-a lungul a două ședințe de laborator:

11.4.1. Laborator 1

1. Se vor calcula componentele orizontale ∇f_x și verticale ∇f_y ale gradientului prin convoluție cu măștile prezentate în (11.3) ... (11.5) și se vor vizualiza rezultatele (operația de convoluție a fost implementată în lucrarea 9).
2. Se vor calcula modulul (11.6) și direcția (11.7) gradientului în fiecare pixel al imaginii pentru cele trei aproximări ale gradientului (Sobel, Prewitt și Roberts) și se va vizualiza modulul gradientului într-o fereastră destinație.
3. Se va realiza binarizarea imaginii rezultat de la punctul 2 cu un prag fix, arbitrar ales.
4. Se vor implementa pașii 1-3 ai algoritmului Canny (pasul 1 – filtrarea gaussiană - a fost implementat în lucrarea 10, la pasul 2 se va aplica operatorul Sobel, iar la pasul 3 se va implementa operația de subțiere a muchiilor). Se va vizualiza într-o fereastră destinație rezultatul obținut după aplicarea pasului 3. Se va compara cu rezultatul obținut prin aplicarea simplă a operatorului Sobel (punctul 2).
5. **Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

11.4.2. Laborator 2

1. Se va implementa algoritmul de binarizare cu prag adaptiv și algoritmul de prelungire a muchiilor prin histereză. Se vor vizualiza rezultatele intermediare pentru muchii tari, muchii slabe și rezultatul final. Se va experimenta cu diferite criterii de învecinare și cu diferiți parametrii *p*.
2. Se vor testa și vizualiza rezultatele finale ale aplicării detectorului de muchii Canny pe diverse tipuri de imagini.
3. **Salvați-vă ceea ce ați lucrat. La sfârșitul laboratorului de procesare a imaginilor va trebui să prezentați propria aplicație cu algoritmii implementați!!!**

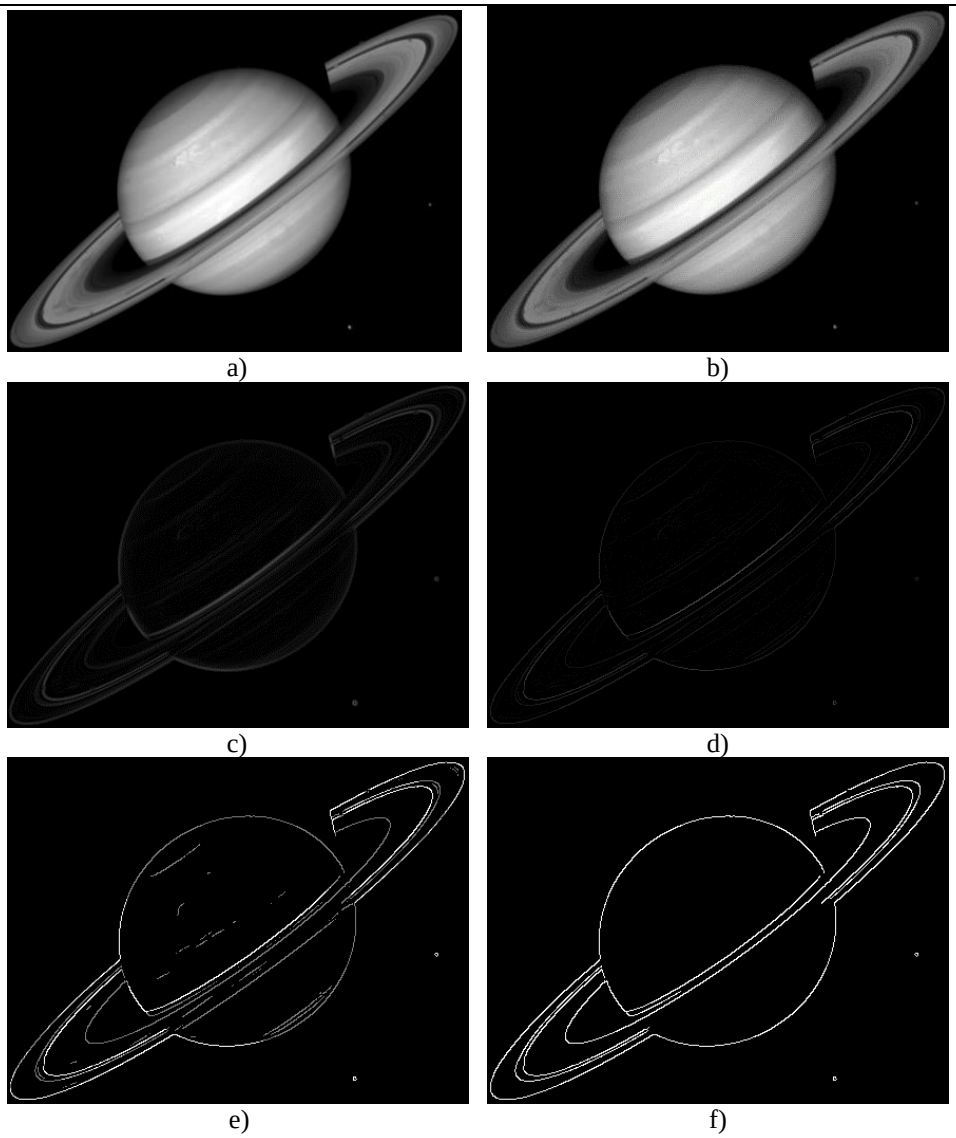


Fig. 11.6 Detecția punctelor de muchie: a) Imaginea inițială; b) Rezultatul filtrării Gaussiene ($\sigma = 0.5$); c) După calculul modulului și direcției gradientului (folosind Sobel); d) După suprimarea non-maximelor modulului gradientului; e) Binarizarea adaptivă a punctelor de muchie ($p = 0.1$); f) Prelungirea muchiilor prin histereză folosind 8 vecinătăți urmată de eliminarea punctelor de muchie slabă

11.5. Bibliografie

[1] E.Trucco, A.Verri, *Introductory Techniques for 3-D Computer Vision*, Prentice Hall, 2001.
[2] R.C.Gonzalez, R.E.Woods, *Digital Image Processing, 4-th Edition*, Pearson, 2017.
[3] S.Nedevschi, M.Vaida, G.Farkaș, T.Marița, D.Moga, R.Dănescu, D.Frențiu, C.Mariș, F.Oniga, C.Rotaru, C.Pocol, *Stereo-Camera Based Object Recongnition System for Vehicle Application – Technical Report*, Cluj-Napoca, 2002.