Arpad GELLERT
Remus BRAD

Procesarea Imaginilor

Aplicaţii

Cuprins

1. Transformări simple ale imaginilor

1.1. Transformarea imaginilor din RGB în nivele de gri

Nivelul de gri corespunzător culorii unui pixel se obţine prin aducerea la aceeaşi intensitate a celor trei componente ale culorii pixelului respectiv (R-roşu, G-verde, B-albastru). Procedeul se aplică pentru toţi pixelii din imagine. Figura următoare prezintă cele 256 de nivele de gri reprezentate pe 24 biţi/pixel:

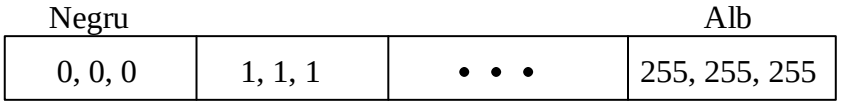

Figura 1. Nivelele de gri

Intensitatea comună ale celor trei componente RGB, poate fi obţinută prin următoarea regulă:

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

unde intensitatea comună I este o medie ponderată a componentelor de culoare (R, G, B). Se poate aplica și o medie aritmetică a componentelor de culoare, ca în pseudocodul de mai jos:

```
for r:= 0 to H-1 do
        for c := 0 to W-1 do
                R := getr(r, c)
                G := getg(r, c)
                B := getb(r, c)
                seti((R+G+B)/3, r, c)
```

1.2. Negativarea imaginilor

Negativarea unei imagini poate fi efectuată transformând componentele culorii fiecărui pixel folosind următoarea regulă:

$$\begin{cases} R = 255 - R \\ G = 255 - G \\ B = 255 - B \end{cases}$$

1.3. Modificarea luminozităţii imaginilor

Luminozitatea unei imagini se poate modifica prin adunarea unei valori (Δ) la componentele culorii fiecărui pixel din imaginea respectivă, având însă grijă să nu se iasă în afara intervalului [0, 255]. Folosind o valoare pozitivă ($\Delta>0$), se obţine o luminozitate mai deschisă, în timp ce o valoare negativă ($\Delta<0$) determină o luminozitate mai închisă a imaginii. Modificarea luminozităţii poate fi efectuată folosind următoarea regulă:

$$R = \begin{cases} 255, \text{ dacă } R + \Delta > 255 \\ 0, \text{ dacă } R + \Delta < 0 \\ R + \Delta, \text{ în rest} \end{cases}$$

$$G = \begin{cases} 255, \text{ dacă } G + \Delta > 255 \\ 0, \text{ dacă } G + \Delta < 0 \\ G + \Delta, \text{ în rest} \end{cases}$$

$$B = \begin{cases} 255, \text{ dacă } B + \Delta > 255 \\ 0, \text{ dacă } B + \Delta < 0 \\ B + \Delta, \text{ în rest} \end{cases}$$

Aplicaţii

Să se implementeze tehnicile prezentate în lucrare. Pentru modificarea luminozităţii să se folosească un slider prin care să se poată seta nivelul dorit de luminozitate.

2. Ajustarea contrastului

Histograma unei imagini reprezintă numărul de pixeli (frecvenţa) pentru fiecare intensitate din imaginea respectivă. De exemplu, în figura următoare 84 de pixeli din imagine au intensitatea 5:

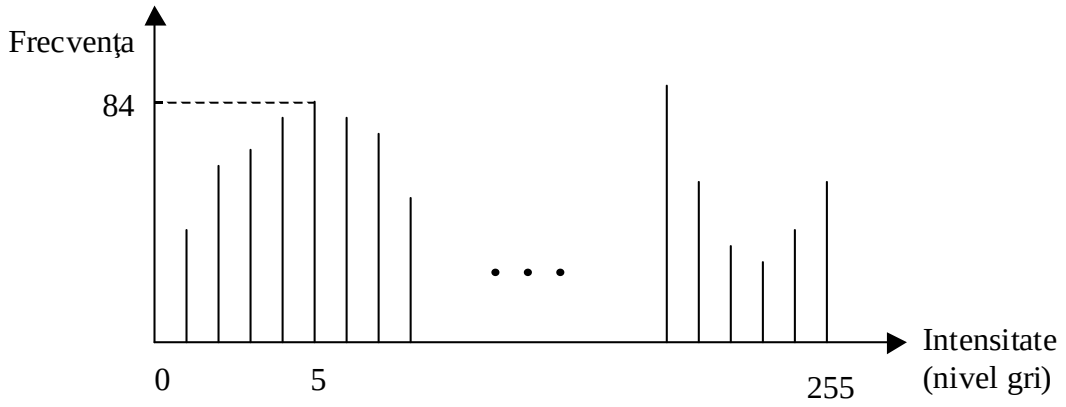

Figura 2. Histograma

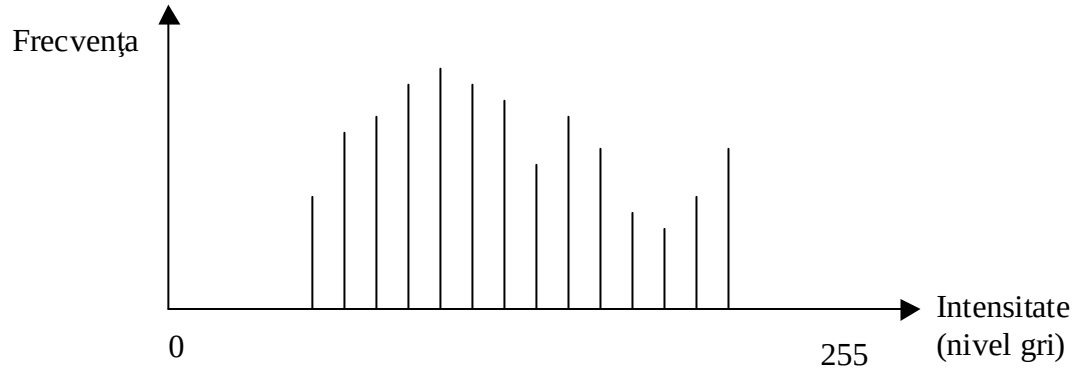

Figura 3. Histograma înaintea ajustării contrastului

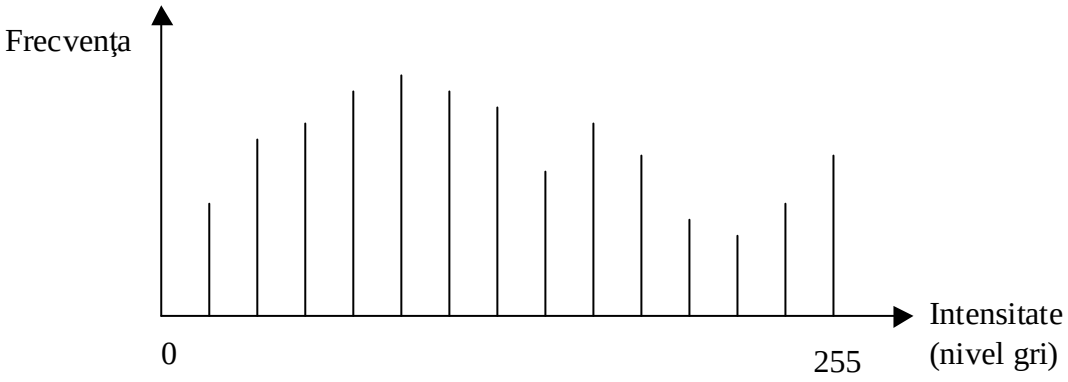

Figura 4. Histograma după ajustarea contrastului

Ajustarea contrastului unei imagini poate fi realizată prin rescalarea intensităţii fiecărui pixel folosind formula de mai jos:

$$I_N = \frac{b-a}{\max - \min} \cdot (I_O - \min) + a$$

- I_N – noua intensitate;
- I_O – intensitatea originală;
- min – intensitatea minimă în imagine (cea mai închisă);
- max – intensitatea maximă în imagine (cea mai deschisă);
- (max-min) – intervalul original de intensităţi;
- a – noua intensitate minimă;
- b – noua intensitate maximă;
- (b-a) – noul interval de intensităţi.

Aplicaţii

Să se implementeze algoritmul de ajustare a contrastului prin metoda prezentată.

3. Egalizarea histogramei

Egalizarea histogramei este o metodă de modificare neadaptivă a histogramei imaginilor şi are rolul de a scoate în evidenţă informaţii care pot fi greu identificate în imaginea originală. Etapele algoritmului de rescalare sunt următoarele:

1. Se determină histograma imaginii;
2. Se construieşte histograma cumulativă;
3. Se determină vectorul de transformare a histogramei;
4. Se modifică imaginea conform vectorului de transformare.

Efectul procesului de egalizare a histogramei poate fi observat în figura următoare:

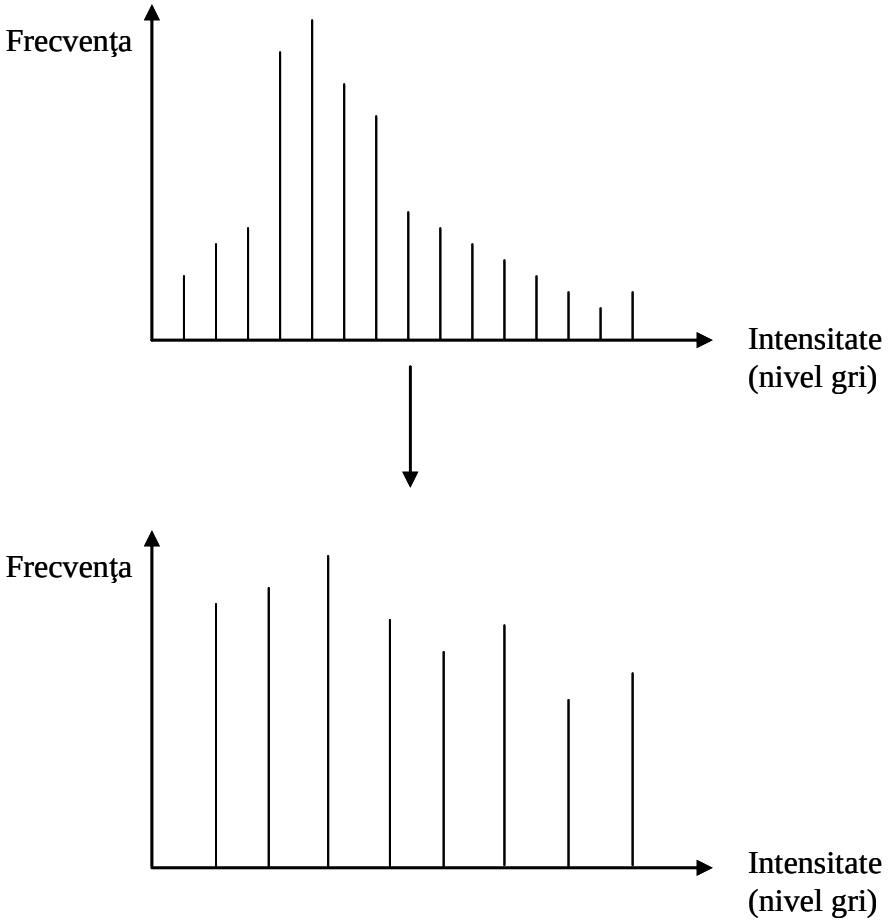

Figura 5. Egalizarea histogramei

Dacă numărul de niveluri de rescalare a imaginii originale e mare, e posibilă modificarea nivelelor de gri în aşa fel încât histograma imaginii îmbunăţite să fie aproape constantă.

Algoritmul de egalizare a histogramei este următorul:

```
for r:= 0 to H-1 do
        for c := 0 to W-1 do
                i := geti(r, c)
                hist[i] := hist[i] + 1
```

```
histc[0] := hist[0]

for i := 1 to 255 do
        histc[i]:=histc[i-1]+hist[i]

for i := 0 to 255 do
        transf[i]:=(histc[i]*255)/(W*H)

for r := 0 to H-1 do
        for c := 0 to W-1 do
                i := geti(r, c)
                seti(transf[i], r, c)
```

unde *hist* este histograma imaginii originale, *histc* este histograma cumulativa, *transf* este vectorul care păstrează modificările intensităţilor, *geti* şi *seti* preia respectiv setează intensitatea pixelului de pe o anumită poziţie din imagine.

Aplicaţie

Să se implementeze algoritmul de egalizare a histogramei.

4. Operaţii geometrice

Transformările geometrice preiau informaţiile despre pixelii din imaginea sursă şi le mapează, în noile locaţii din imaginea destinaţie. Forma generală a unei transformări geometrice poate fi exprimată matricial astfel:

$$\begin{vmatrix} x_2 \\ y_2 \end{vmatrix} = A \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} + B$$

unde (x_2,, y_2) reprezintă coordonatele pixelului din imaginea destinaţie, (x_1, y_1) reprezintă coordonatele pixelului din imaginea sursă, iar *A* şi *B* sunt matricile prin care se particularizează transformarea dorită. Translaţia poate fi implementată prin specificarea valorilor pentru matricea B, în timp ce scalarea, rotaţia si reflexia pot fi obţinute prin setarea matricii A (vezi paragraful 4.5). Pentru corectarea distorsiunilor geometrice introduse în imagini din cauza unor iregularităţi de perspectivă, se combină aceste transformări elementare, utilizând ambele matrici.

4.1. Scalarea

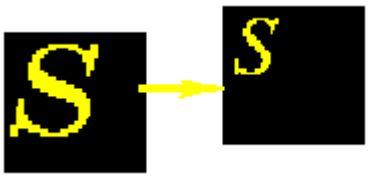

Figura 6. Scalarea

Scalarea este o operaţie geometrica care măreşte sau micşorează o imagine sau o parte a unei imagini. Micşorarea imaginii se realizează prin înlocuirea unui grup de pixeli din imaginea originală (mărimea grupului este determinată de factorul de scalare), cu un singur pixel în imaginea destinaţie. Acest pixel va avea intensitatea unui anumit pixel din grup, sau intensitatea obţinută prin *interpolarea* intensităţilor pixelilor din grup (de exemplu intensitatea medie).

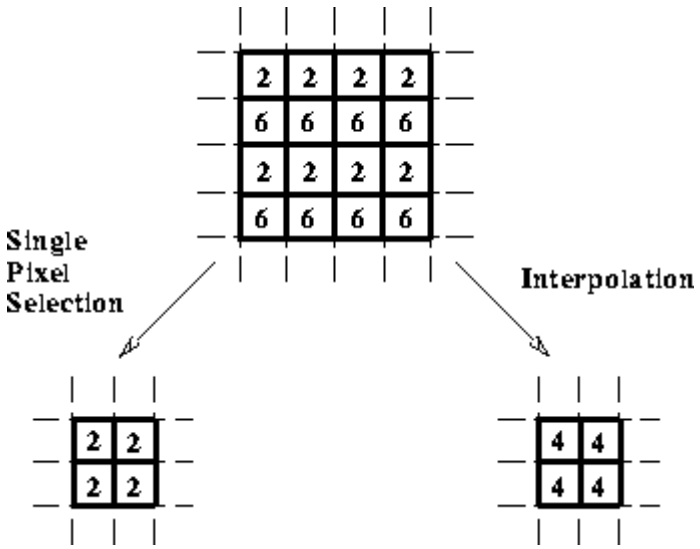

Figura 7. Metode de micşorare. a) Inlocuirea cu pixelul din dreapta sus. b) Interpolarea folosind valoarea medie.

Mărirea unei imagini este realizată prin replicare sau prin interpolare. Astfel, unui pixel din imaginea originală îi corespunde un grup de pixeli în imaginea destinaţie, care vor avea fie intensitatea pixelului original (din imaginea sursă), fie o interpolare a acesteia cu intensităţile pixelilor invecinaţi. Mărimea grupului este determinată de factorul de scalare. Operaţiile de interpolare, deşi sunt mai mari consumatoare de resurse de procesare (ceea ce se traduce prin creşterea timpului de procesare), oferă rezultate superioare din punct de vedere calitativ.

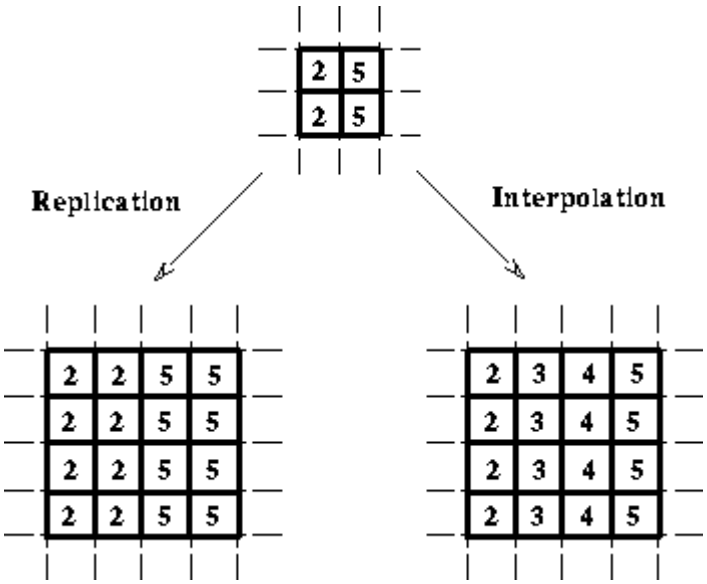

Figura 8. Metode de mărire. a) Replicarea unui singur pixel. b) Interpolarea.

4.2. Rotaţia

Figura 9. Rotaţia

Rotaţia unei imagini se realizează prin maparea unui pixel de intrare (x_1, y_1) în poziţia de ieşire (x_2, y_2), determinată în urma rotirii cu un unghi θ, în jurul unei origini (x_0, y_0). Transformarea aplicată asupra coordonatelor pixelilor este:

$$x_2 = \cos(\theta) \cdot (x_1 - x_0) - \sin(\theta) \cdot (y_1 - y_0) + x_0$$
$$y_2 = \sin(\theta) \cdot (x_1 - x_0) + \cos(\theta) \cdot (y_1 - y_0) + y_0$$

Rotaţia este folosită ca procedeu de ajustare/corectare a aspectului unei imagini, sau în cadrul operaţiilor de procesare ce acţionează direcţional. In majoritatea implementărilor coordonatele de ieşire (x_2, y_2) care se află în afara imaginii sunt ignorate. Algoritmul de rotaţie poate produce coordonate (x_2, y_2) care nu sunt întregi. Pentru generarea intensităţilor pixelilor cu coordonate întregi, cele mai importante metode folosite sunt următoarele:

- un pixel cu coordonate întregi poate primi intensitatea celui mai apropiat pixel cu coordonate ne-întregi;
- se calculează intensităţile pixelilor cu coordonate întregi făcându-se media intensităţilor pixelilor cu coordonate ne-întregi. Această metodă produce rezultate mai bune, dar necesită o putere de procesare mai mare.

4.3. Reflexia

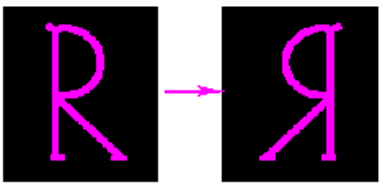

Figura 10. Reflexia

Reflexia geometrică (oglindirea) transformă o imagine sursă astfel încât pixelii (x_1, y_1) sunt reflectaţi faţă de o axă specificată în noua poziţie (x_2, y_2) în imaginea destinaţie. Reflexia faţă de o axă orizontală de ordonată y_0 se realizează prin transformările:

$$x_2 = x_1$$
$$y_2 = -y_1 + (2 \cdot y_0)$$

În mod similar, reflexia faţă de o axă verticală de abscisă x_0 se realizează astfel:

$$x_2 = -x_1 + (2 \cdot x_0)$$
$$y_2 = y_1$$

Reflexia dupa o axă orientată într-o direcţie arbitrara θ, şi care trece prin punctul (x_0, y_0):

$$x_2 = x_1 - 2 \cdot \Delta \cdot \sin(\theta)$$
$$y_2 = y_1 + 2 \cdot \Delta \cdot \cos(\theta)$$

unde $\Delta = (x_1 - x_0) \cdot \sin(\theta) - (y_1 - y_0) \cdot \cos(\theta)$. În caz că ($x_0$, y_0) nu este în centrul imaginii, o parte din imagine va fi reflectată în afara spaţiului vizibil al imaginii.

4.4. Translaţia

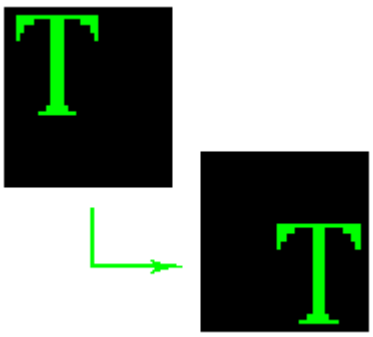

Figura 11. Translaţia

Translaţia presupune deplasarea pixelilor (x1, y1) din imaginea originală, cu o valoare (β_1, β_2) specificată de utilizator, în noile poziţii (x_2, y_2) determinate prin transformările:

$$x_2 = x_1 + \beta_x$$
$$y_2 = y_1 + \beta_y$$

În cazul in care noile coordonate (x_2, y_2) sunt in afara imaginii, operatorul de translaţie le ignoră.

4.5. Transformarea afină

Figura 12. Transformarea afină

În multe imagini sunt detectate o serie de distorsiuni introduse de iregularităţile de perspectivă. Dacă aceste distorsiuni sunt uniforme, ele pot fi corectate aplicând transformările afine, care mapează pixelii (x1, y1) dintr-o imagine de intrare, în noile poziţii (x2, y2) ale imaginii de ieşire aplicând o combinaţie liniară a *translaţiei*, *rotaţiei* şi a *scălării*. O transformare afină poate fi descrisă prin

$$\begin{vmatrix} x_2 \\ y_2 \end{vmatrix} = A \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} + B$$

Translaţia pură se obţine prin:

$$A = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix},\ B = \begin{vmatrix} b_1 \\ b_2 \end{vmatrix}$$

Rotaţia pură foloseşte doar matricea A:

$$A = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix},\ B = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

Similar, scalarea pură se obţine prin:

$$A = \begin{vmatrix} a_{11} & 0 \\ 0 & a_{22} \end{vmatrix},\ B = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

Aplicaţii

Să se implementeze operaţiile geometrice prezentate în lucrare.

5. Îndepărtarea zgomotului

Imaginile digitale sunt adesea afectate de diferite tipuri de zgomot. Există două mari categorii de zgomot: de tip Gaussian şi de tip impuls. Cel Gaussian este unul statistic cu distribuţie Gaussiană. Cel de tip impuls este independent de pixelii din imagine, având o distribuţie aleatoare. Imaginile digitale pot fi afectate de zgomot de tip impuls în timpul achiziţionării sau în timpul transmisiei. Zgomotul salt-and-pepper este unul de tip impuls compus din pixeli cu valori minime şi maxime în cadrul imaginii afectate. Principalul obiectiv al filtrelor de zgomot de tip impuls este păstrarea valorilor pixelilor neafectaţi şi reconstruirea celor afectaţi.

5.1. Filtrul Trece-Jos (FTJ)

Filtrarea trece-jos se poate efectua prin convoluţia imaginii afectate de zgomot cu o matrice:

$$H_1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H_2 = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad ..., \quad H_n = \frac{1}{(n+2)^2}\begin{bmatrix} 1 & n & 1 \\ n & n^2 & n \\ 1 & n & 1 \end{bmatrix}.$$

Aceste matrici, numite măşti de îndepărtare a zgomotului, sunt normalizate la unitate, astfel procesul de îndepărtare a zgomotului nu introduce o schimbare de amplitudine în imaginea rezultată. Convoluția imaginii cu matricea *H* se realizează după cum urmează:

```
for r := 1 to H-2 do
        for c := 1 to W-2 do
                s := 0
                for row := r-1 to r+1 do
                        for col := c-1 to c+1 do
                                i := geti(r, c)
                                s := s + i*H[row-r+1, col-c+1]
                s := s/((n+2)(n+2))
                seti(s, r, c)
```

5.2. Tehnica Outlier

O tehnică simplă de îndepărtare a zgomotului, numită *outlier*, compară fiecare pixel cu media a opt dintre vecinii săi. Dacă magnitudinea diferenţei e mai mare decât un prag, pixelul este considerat zgomot şi e înlocuit cu media pixelilor vecini.

$$\text{dacă } \left| I - \frac{1}{8}\sum_{i=1}^{8} p_i \right| > \varepsilon \text{, atunci } I = \frac{1}{8}\sum_{i=1}^{8} p_i \text{ .}$$

Media poate fi obţinută prin convoluţia imaginii cu următoarea mască:

$$H = \frac{1}{8}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

5.3. Filtrul median

Filtrul median e format dintr-o fereastră ce cuprinde un număr impar de pixeli. Pixelul central din fereastră e înlocuit cu medianul pixelilor din fereastră:

$$med(a,b,c,d,e) = \text{maximin}(\ a,b,c,d,e) = \max\{\min(\ a,b,c), \min(\ a,b,d), \min(\ a,b,e), \min(\ a,c,d), \min(\ a,c,e), \min(\ a,d,e), \min(\ b,c,d), \min(\ b,c,e), \min(\ b,d,e), \min(\ c,d,e)\}.$$

5.4. Filtrul pseudomedian

Filtrarea mediană necesită un volum mare de calcule, iar numărul operaţiilor creşte exponenţial cu dimensiunea ferestrei. Un operator mai simplu este filtrul pseudomedian:

$$pmed(a,b,c,d,e) = \frac{1}{2}\text{maximin}(\ a,b,c,d,e) + \frac{1}{2}\text{minimax}(\ a,b,c,d,e) =$$

$$= \frac{1}{2}\max\{\min(\ a,b,c), \min(\ b,c,d), \min(\ c,d,e)\} + \frac{1}{2}\min\{\max(\ a,b,c), \max(b,c,d), \max(c,d,e)\}.$$

În cazul unui filtru pseudomedian în formă de "+" calculul funcţiilor *min* şi *max* se face în ferestre, după cum urmează:

$$\begin{array}{ccccc} & & y_1 & & \\ & & \vdots & & \\ x_1 & \dots & x_M & \dots & x_C \\ & & \vdots & & \\ & & y_R & & \end{array}$$

unde şirurile $\{X_C\}$ şi $\{X_R\}$ conţin pixelii aflaţi pe orizontală şi respectiv verticală din fereastră. Pseudomedianul poate fi definit ca:

$$pmed = \frac{1}{2}\max[\ \text{maximin}\ \{X_C\},\ \text{maximin}\ \{Y_R\}] + \frac{1}{2}\min[\ \text{minimax}\ \{X_C\},\ \text{minimax}\ \{Y_R\}].$$

5.5. Filtrul Markov

În continuare este prezentată o metodă contextuală care aplică lanţurile Markov pentru înlocuirea pixelului afectat de zgomot cu pixelul care a apărut de cele mai multe ori în acelaşi context, într-o anumită vecinătate [2]. S-a adaptat lanţul Markov clasic care lucrează cu secvenţe de valori 1D astfel încât să poată folosi secvenţe de valori 2D. În imagini, considerăm contextul unui pixel ca fiind pixelii din jur. Pentru imagini în nivele de gri, stările lanţului Markov sunt pixeli cu valori cuprinse între 0 şi 255. Astfel, filtrul Markov se aplică în felul următor:

$$P[q_{x,y} \mid q_{x+i,y+j}, i, j = -SR, ..., SR,\ 0 \le x+i < W,\ 0 \le y+j < H, \textit{exceptând } i = j = 0] =$$

$$= P\left[q_{x,y} \,\middle|\, q_{x+i,y+j},\ i, j = -\frac{CS}{2}, ..., \frac{CS}{2},\ 0 \le x+i < W,\ 0 \le y+j < H, \textit{exceptând } i = j = 0 \right]$$

unde CS este dimensiunea contextului, iar SR este raza de căutare în jurul pixelului afectat de zgomot.

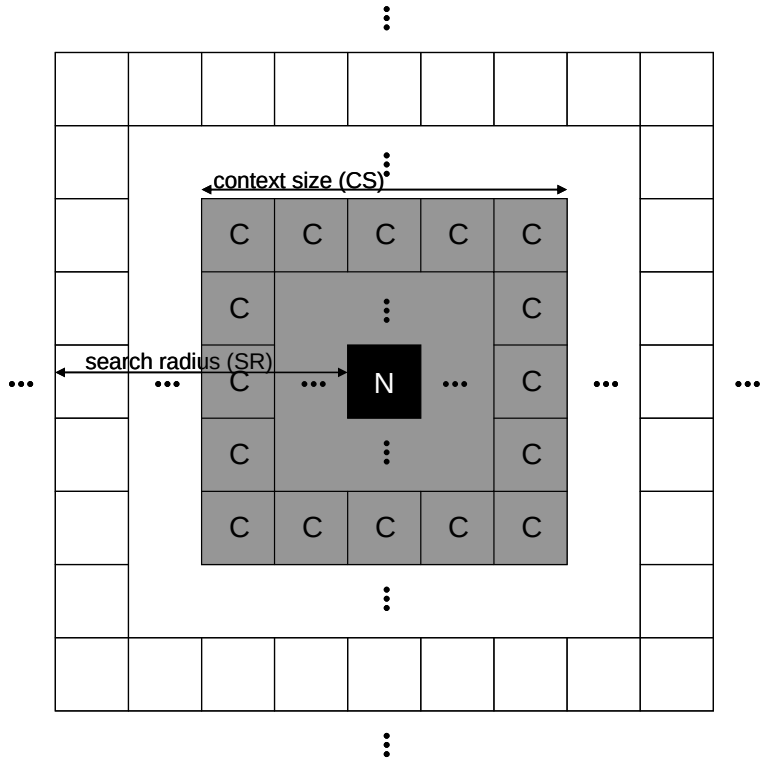

Figura 13. Filtrul Markov

În figura de mai sus am marcat cu negru pixelul afectat de zgomot (N) și cu gri pixelii care aparțin contextului (C). În continuare este prezentat pseudocodul filtrului Markov:

```
CBP (x, y, CS, SR, T)
      For i:=x-SR to x+SR, 0≤i<W
            For j:=y-SR to y+SR, 0≤j<H
                  If i=x AND j=y then
                        Continue
                  If SAD(x, y, i, j, CS)<T AND NOT Salt_Pepper(i, j) then
                        Q[Color(i, j)]:=Q[Color(i, j)]+1
      Return Max(Q)


SAD (x1, y1, x2, y2, CS)
      S:=0
      For i:= -CS/2 to CS/2, 0≤i+x1<W, 0≤i+x2<W, do
            For j:= -CS/2 to CS/2, 0≤j+y1<H, 0≤j+y2<H do
                  If i=0 AND j=0 then
                        Continue
                  S:=S + |Color(i+x1, j+y1)-Color(i+x2, j+y2)|
      Return S


CBPF(CS, SR, T)
      For i:=0 to W-1 do
            For j:=0 to H-1 do
                  If Salt_Pepper(i, j) then
                        Set_Color(i, j, CBP(i, j, CS, SR, T))
```

5.6. Filtrul Markov îmbunătățit

În continuare este prezentat filtrul Markov îmbunătățit [3] care folosește un context in formă de + în loc de contextul complet și aplică o căutare în forma de * în loc de căutarea completă a metodei anterioare:

$$P[q_{x,y}\left|q_{x,y+j}, j=-SR,...,SR,\ 0\le y+j<H, exceptând\ \ j=0;\right.$$
$$q_{x+i,y}, i=-SR,...,SR,\ 0\le x+i<W, exceptând\ i=0;$$
$$q_{x+k,y+k}, k=-SR,...,SR,\ 0\le x+k<W, 0\le y+k<H, exceptând\ k=0;$$
$$q_{x-l,y+l}, l=-SR,...,SR,\ 0\le x-l<W, 0\le y+l<H, exceptând\ l=0] =$$
$$= P\left[q_{x,y}\left|q_{x,y+j},\ j=-\frac{CS}{2},...,\frac{CS}{2},\ 0\le y+j<H, exeptând\ \ j=0;\right.\right.$$
$$\left.\left. q_{x+i,y},\ i=-\frac{CS}{2},...,\frac{CS}{2},\ 0\le x+i<W,\ exceptând\ i=0\right.\right]$$

unde, la fel ca la metoda anterioară, CS este dimensiunea contextului, iar SR este raza de căutare în jurul pixelului afectat de zgomot.

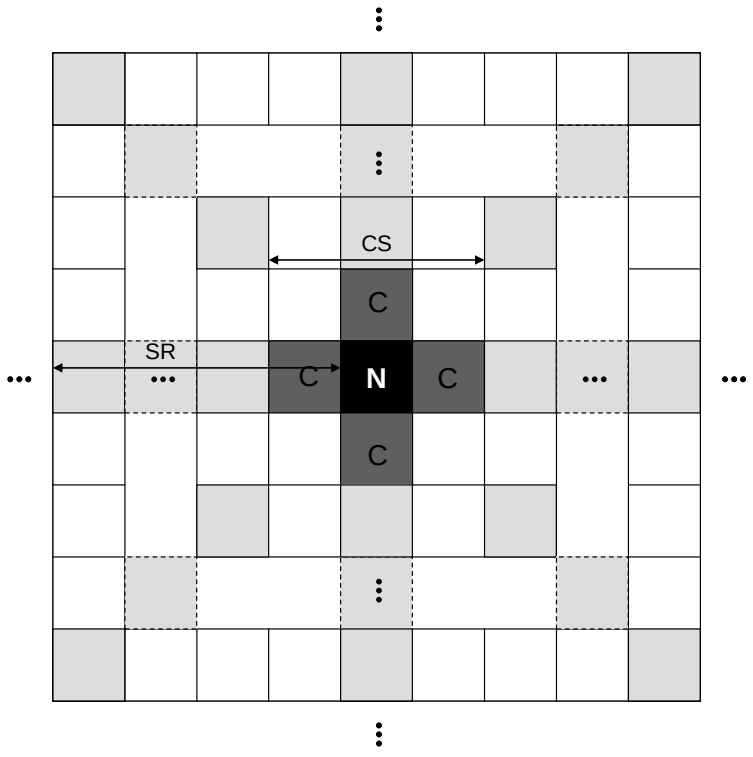

Figura 14. Filtrul Markov îmbunătățit

În figura de mai sus am marcat cu negru pixelul afectat de zgomot (N), cu gri închis pixelii care aparțin contextului (C) și cu gri deschis zonele în care se caută contextul. Pseudocodul filtrului Markov îmbunătățit:

```
Markov(x, y, CS, SR, T)
  For j:=y-SR to y+SR, 0≤j<H
    If j=y then Continue
    If SAD(x, y, x, j, CS)<T AND NOT Salt_Pepper(x, j) then
        Q[Color(x, j)]:=Q[Color(x, j)]+1
  For i:=x-SR to x+SR, 0≤i<W
    If i=x then Continue
    If SAD(x, y, i, y, CS)<T AND NOT Salt_Pepper(i, y) then
        Q[Color(i, y)]:=Q[Color(i, y)]+1
  For k:=-SR to SR, 0≤x+k<W, 0≤y+k<H
    If k=0 then Continue
    i:=x+k
    j:=y+k
    If SAD(x, y, i, j, CS)<T AND NOT Salt_Pepper(i, j) then
        Q[Color(i, j)]:=Q[Color(i, j)]+1
  For k:=-SR to SR, 0≤x-k<W, 0≤y+k<H
    If k=0 then Continue
    i:=x-k
    j:=y+k
    If SAD(x, y, i, j, CS)<T AND NOT Salt_Pepper(i, j) then
        Q[Color(i, j)]:=Q[Color(i, j)]+1
  If Q[Max(Q)]=0 then Return Color(x, y)
  Return Max(Q)


SAD(x1, y1, x2, y2, CS)
  S:=0
  For j:= -CS/2 to CS/2, 0≤j+y1<H, 0≤j+y2<H do
    If j=0 then Continue
    S:=S + |Color(x1, j+y1)-Color(x2, j+y2)|
  For i:= -CS/2 to CS/2, 0≤i+x1<W, 0≤i+x2<W, do
    If i=0 then Continue
    S:=S + |Color(i+x1, y1)-Color(i+x2, y2)|
  Return S


Markov_Filter(CS, SR, T)
  For i:=0 to W-1 do
    For j:=0 to H-1 do
      If Salt_Pepper(i, j) then
        Set_Color(i, j, Markov(i, j, CS, SR, T))
```

Aplicaţii

1. Să se implementeze filtrele de zgomot prezentate;
2. Să se implementeze filtrul de zgomot care înlocuieşte intensitatea fiecărui pixel din imagine cu elementul central al secvenţei care conţine intensitaţile sortate ale pixelilor vecini.

6. Accentuarea contururilor

6.1. Filtrul Trece-Sus (FTS)

Accentuarea contururilor se poate realiza prin utilizarea filtrelor trece-sus. Cele mai utilizate măşti FTS de mărime 3x3 sunt următoarele:

$$H_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \qquad H_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \qquad H_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

Aşa cum se poate observa, măştile utilizate pentru accentuarea contururilor au proprietatea că suma elementelor este 1, evitând în felul acesta modificări ale intensităţilor în imaginea procesată. Algoritmul constă în înlocuirea intensităţilor cu sumele obţinute prin convoluţia imaginii cu una din măştile FTS. Evident, este necesară şi o normalizare a noilor intensităţi la intervalul [0, 255].

6.2. Tehnica ‘unsharp masking’

Noua intensitate a unui pixel reprezintă diferenţa ponderată dintre intensitatea originală (din imaginea cu rezoluţie normală) şi intensitatea pixelului corespunzător din imaginea cu rezoluţie scăzută. Imaginea cu rezoluţie scăzută poate fi obţinută prin filtrare trece-jos (vezi paragraful 5.1).

$$G(j, k) = \frac{c}{2c-1} \cdot F(j, k) - \frac{1-c}{2c-1} \cdot F_L(j, k)$$

unde

- $c \in \left[\frac{3}{5}, \frac{5}{6}\right]$, deci $c \in [0.6, 0.8]$;
- $F(j, k)$ – intensitatea originală (intensitatea pixelului din imaginea cu rezoluţie normală);
- $F_L(j, k)$ – intensitatea obţinută cu un FTJ (intensitatea pixelului din imaginea cu rezoluţie scăzută);
- L – dimensiunea măştii FTJ.

Aplicaţii

Să se implementeze cele două tehnici de accentuare a contururilor prezentate în lucrare: filtrul trece-sus şi respectiv tehnica ‘unsharp masking’. Algoritmul ‘unsharp masking’ se va implementa printr-o singura parcurgere a imaginii procesate!

Observaţie

La convoluţia imaginii pentru accentuarea contururilor, se va lucra cu intensităţile originale şi nu cu cele modificate.

7. Detecţia contururilor

Modificările sau discontinuităţile de intensitate (amplitudine) dintr-o imagine constituie caracteristici fundamentale care pot indica prezenţa unor obiecte într-o imagine. Aceste discontinuităţi sunt denumite contururi. La toţi operatorii prezentaţi în lucrare, intensităţile se înlocuiesc cu sumele obţinute prin convoluţia imaginii procesate cu măştile corespunzătoare.

7.1. Operatorul Kirsch

1) $H_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad H_2 = [1 \ -1], \qquad H_3 = [1 \ \ 0 \ -1];$

unde H_1 este utilizat pentru detecţia contururilor orizontale, în timp ce măştile H_2 şi H_3 se folosesc pentru contururile verticale din imagine.

2) $H_1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \quad H_3 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \quad H_4 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$

$G = \max\{suma(H_1),\ suma(H_2),\ suma(H_3),\ suma(H_4)\}$

7.2. Operatorul Laplace

$$H_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad H_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix},$$

unde H_2 este operatorul Laplace propus de Prewitt.

7.3. Operatorul Roberts

$$P = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad Q = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},$$

$R = k \cdot \sqrt{suma(P)^2 + suma(Q)^2}$ (ex. k = 7).

7.4. Operatorul Prewitt

$$P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \qquad Q = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},$$

$R = \sqrt{suma(P)^2 + suma(Q)^2}$.

7.5. Operatorul Sobel

$$P = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad Q = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},$$

$$R = \sqrt{suma(P)^2 + suma(Q)^2}\,.$$

7.6. Operatorul Frei-Chen

$$F_1 = \begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{bmatrix},\ F_2 = \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix},\ F_3 = \begin{bmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{bmatrix},\ F_4 = \begin{bmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{bmatrix}$$

$$F_5 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix},\ F_6 = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix},\ F_7 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix},\ F_8 = \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix},\ F_9 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$R = \sqrt{\frac{\sum_{i=1}^{4} suma(F_i)^2}{\sum_{i=1}^{9} suma(F_i)^2}} \cdot 255\,.$$

Aplicaţii

Să se implementeze tehnicile de detecţie a contururilor prezentate în lucrare. Notaţia *suma(M)* reprezintă suma obţinută prin convoluţia imaginii procesate cu masca *M*.

8. Detecţia de contur bazată pe filtrul Gabor

Diferenţa importantă faţă de metodele de detecţie aconturului prezentate în lucrarea precedentă, este că filtrul Gabor determină dinamic matricea pentru fiecare fereastră de 3x3 din imagine. Cu alte cuvinte, se lucrează cu o matrice care se adaptează dinamic la context, şi nu cu una prestabilită. Filtrul lui Gabor constă în modulaţia unei funcţii gaussiene şi a unui semnal sinusoidal [1]. În algoritm, *suma(P)* şi *suma(Q)* reprezintă sumele obţinute prin convoluţia imaginii cu măştile *P* respectiv *Q*:

$$P = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \qquad Q = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Algoritmul:

pentru fiecare fereastră de 3x3 din imagine (unde (row, col) este pixelul din mijlocul ferestrei)

- se calculează cele două sume: suma(P) şi suma(Q);

  dacă suma(Q) = 0

  dacă suma(P) ≥ 0, atunci $u = \frac{\pi}{2}$;

  dacă suma(P) < 0, atunci $u = -\frac{\pi}{2}$;

  altfel

  $u = arctg\left(\frac{suma(P)}{suma(Q)}\right)$;

  dacă suma(Q) < 0, atunci $u = u + \pi$;

- $u = u + \frac{\pi}{2}$;
- suma = 0;

  pentru fiecare pixel (r, c) din fereastră (r-linia, c-coloana)

  - $scale = \left(e^{-\frac{poz(r)^2 + poz(c)^2}{2\sigma^2}} \right) \cdot \sin\left(\omega \cdot \left(poz(r) \cdot \cos u + poz(c) \cdot \sin u\right)\right)$

  //scale – elementele noii matrici (3x3).

  - $suma = suma + scale \cdot Intensitate(r, c)$;

  //poz(r) – poziţia pe verticală faţă de primul element al ferestrei, $poz(r) \in [0, 2]$

  //poz(c) – poziţia pe orizontală faţă de primul element al ferestrei, $poz(c) \in [0, 2]$

- se setează pe poziţia (row, col) intensitatea *suma*.

Observaţie: $\pi = 3.14$, $\sigma = 0.66$, $\omega = 1.5$.

Aplicaţie: să se implementeze filtrul Gabor.

9. Segmentarea imaginilor

Segmentarea este este unul din cei mai importanţi paşi în analiza unei imagini [1]. Scopul este acela de a împărţi imaginea în regiuni ce au o corelaţie puternică cu obiectele sau suprafeţele din imagine. Există două tipuri de segmentare [1]: completă şi parţială. Segmetarea completă generează un set de regiuni disjuncte ce corespund în mod unic cu obiectele din imagine. Pentru realizarea unei segmentări complete , este necesară cooperarea cu nivelele superioare de procesare , care utilizează cunoştinţe specifice din domeniu. În cazul segmentării parţiale, imaginea este împărţită în regiuni disjuncte care sunt omogene relative la o anumită proprietate cum ar fi luminozitatea, culoarea, contextul reflectivitatea, etc.

Ambiguitatea prezentă în imagini este principala problemă a segmentării şi ea este deseori însoţită de zgomot. Tehnicile de segmentare pot fi clasificate în patru categorii:

- filtrare locală şi fixare de praguri (*threshold*);
- metode bazate pe *snake* şi de tip balon, algoritmul *watershed*;
- tehnici de *region growing* şi *split and merge*;
- metode de optimizare globală bazate pe funcţii de energie, funcţii bayesiene sau pe criteriul MDL (Minimum Description Length).

Metoda filtrării foloseşte informaţia locală şi nu poate garanta contururi continue. Metodele de tip balon sau cele bazate pe *snake* folosesc doar informaţia existentă de-a lungul conturului şi necesită o bună estimare a poziţiei de iniţializare pentru obţinerea unei convergenţe corecte. Avantajul metodelor de *region growing* este acela că se bazează pe statistici realizate în interiorul fiecărei regiuni. Cele trei tehnici nu folosesc informaţia existentă la nivelul întregii imagini. Spre deosebire de acestea, metodele bazate pe energie, bayes sau MDL, utilizează criterii globale, dar fiind bazate pe o minimizare, miinimul este de multe ori greu de atins.

Tehnica *region growing* detectează o regiune identificând acei pixeli conectaţi la punctul de pornire care au intensitatăţi "similare". Algoritmul recursiv asigură extinderea regiunii în patru direcţii (spre cei patru pixeli vecini), dacă intensităţile acestora se încadrează în intervalul *[m-t, m+t]*, unde *m* este intensitatea medie a regiunii, iar *t* este o valoare prag introdusă de utilizator (parametru). Un alt parametru care trebuie introdus de utilizator este punctul de pornire care poate fi introdus printr-un click al mouse-ului pe imagine sau prin precizarea directă a coordonatelor acestuia. Ieşirea programului constă într-o copie binară a imaginii procesate, în care, un 1 reprezintă un pixel al obiectului detectat, iar un 0 indică un pixel care aparţine fundalului.

Un alt algoritm *region growing* împarte imaginea în blocuri de aceeaşi dimensiune. Se recomandă divizarea în blocuri de 2x2 în cazul în care se aplică direct algoritmul *region growing*, respectiv de 16x16 dacă se efectuează şi etapa *merge-split*.

În etapa opţională *merge-split* se foloseşte un prag (threshold) introdus de utilizator. Valoarea acestui prag determină care blocuri pot fi combinate (unite) într-un singur bloc şi care blocuri pot fi divizate în blocuri mai mici, în funcţie de diferenţa dintre intensitatea maximă şi minimă din fiecare bloc. Dacă diferenţa *max-min* a unui bloc este apropiată de diferenţa *max-min* a unui bloc vecin (diferenţa *max-min* aferentă celor două blocuri este sub valoarea pragului), atunci blocurile se unesc într-unul singur. Un bloc este divizat (1/2 sau 1/4) dacă diferenţa *min-max* este peste prag. Procesul *merge-split* se aplică recursive până când nici un bloc nu satisface criteriul de a fi divizat sau unit. Astfel, un bloc a cărui diferenţă *max-min* depăşeşte valoarea pragului, va fi divizat până când diferenţele max-min aferente blocurilor obţinute sunt sub prag,

sau dimensiunile acestora ajung la un pixel, în acest caz diferenţa *max-min* fiind zero. Pentru evitarea divizării imaginii într-un număr prea mare de regiuni foarte mici, se poate folosi un parametru (introdus de utilizator) care precizează dimensiunea minimă a blocurilor obţinute prin procesul de divizare. Astfel, utilizatorul poate forţa terminarea algoritmului de segmentare cu un număr mic de regiuni, deoarece blocurile generate nu vor fi mai mici decât dimensiunea specificată.

Etapa *region growing* analizează fiecare bloc unindu-l cu blocuri adiacente care satisfac un anumit criteriu. Un criteriu constă în diferenţa *max-min* şi determină aderarea acelor blocuri adiacente la o regiune, ale căror diferenţă max-min se încadrează cu o anumită toleranţă (specificată de utilizator) în diferenţa *max-min* a regiunii. Această toleranţă nu trebuie să fie identică cu pragul folosit în etapa *merge-split*. Un alt criteriu care determină care blocuri pot fi unite, constă în intensitatea medie a blocurilor. În acest caz, algoritmul asigură aderarea unui bloc la o regiune, dacă intensitatea medie a blocului se încadrează în intervalul *[m-t, m+t]*, unde *m* este intensitatea medie a regiunii, iar *t* este o valoare prag introdusă de utilizator (parametru).

Etapa *dissolve* combină regiunile care au o dimensiune mai mică decât un prag (stabilit de utilizator) cu regiunea adiacentă având cea mai apropiată medie a intensităţilor. În felul acesta se reduce numărul de regiuni, cele mai puţin semnificative fiind eliminate.

Aplicaţii

Să se implementeze, la alegere, unul din cei doi algoritmi de segmentare *region growing*, prezentaţi în lucrare.

10. Corelaţia imaginilor

Coeficientul de corelaţie este o metrică care exprimă similaritatea (nivelul de potrivire) dintre două semnale, de aceea, este foarte des folosit pentru căutarea şabloanelor (template matching). Coeficientul de corelaţie *r* al perechii *(x, y)* se calculează folosind următoarea formulă:

$$r_{xy} = \frac{\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \frac{\sum_{i=1}^{n} y_i}{n}}{\sqrt{\left(\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}\right)\left(\sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}\right)}}$$

Următorul pseudo-cod prezintă algoritmul care determină coeficientul maxim de corelaţie (cea mai bună potrivire) aferent unei perechi de imagine/şablon:

```
template_size = M * M;
sum = 0;
sqrsum = 0;
for j = 0 to M-1
{
   for i = 0 to M-1
   {
      sum += template[j, i];
      sqrsum += template[j, i] * template[j, i];
   }
}
mean_template = sum / template_size;
var2template = sqrsum – (sum * sum) / template_size;

for y = 0 to N-M                    //image rows
{
   for x = 0 to N-M                 //image columns
   {
      sum = 0;
      sqrsum = 0;
      prodsum = 0;
      for j = 0 to M-1              //template rows
      {
         for i = 0 to M-1           //template columns
         {
            sum += image[y+j, x+i];
            sqrsum += image[y+j, x+i] * image[y+j, x+i];
            prodsum += image[y+j, x+i] * template[j, i];
         }
      }
      var2image = sqrsum – (sum * sum) / template_size;
      corr_coeff[y,x] = (prodsum – (mean_template * sum)) / sqrt(var2image * var2template);
   }
}
```

unde *image* şi *template* sunt imagini bidimensionale de mărime *N*N* respectiv *M*M*, iar *var2image* este suma pătratelor distanţelor dintre intensităţile pixelilor din imagine acoperite de şablon. Aşa cum se poate observa, valorile *mean_template, var2template* respectiv *template_size* pot fi precalculate şi folosite pentru toţi coeficienţii de corelaţie dintr-o pereche imagine/şablon. Algoritmul poate fi folosit şi pentru estimarea locală a mişcării (vezi următoarea lucrare de laborator).

Aplicaţii

Să se implementeze algoritmul prezentat în lucrare. Pentru testare se va alege un set de imagini şi o imagine şablon. Pentru fiecare imagine se va determina coeficientul maxim de corelaţie cu imaginea şablon.

11. Algoritmi *block matching*

Algoritmii *block matching* [1, 4] sunt cele mai cunoscute metode de măsurare a similarităţii dintre două imagini şi sunt folosite pentru estimarea mişcării. Aceste metode au fost adoptate de diverse standarde de codificare a imaginilor video. Metodele *block matching* presupun faptul că un bloc de pixeli are aceeaşi mişcare de translaţie de la un cadru la altul. Cele mai utilizate metode de comparare a blocurilor (măsuri de similaritate) au la bază suma pătratelor diferenţelor SSD (Sum of Squared Differences), respectiv, suma diferenţelor absolute SAD (Sum of Absolute Differences). De multe ori se preferă utilizarea metodei SAD datorită nivelului mic de complexitate (mai ales în implementările hardware).

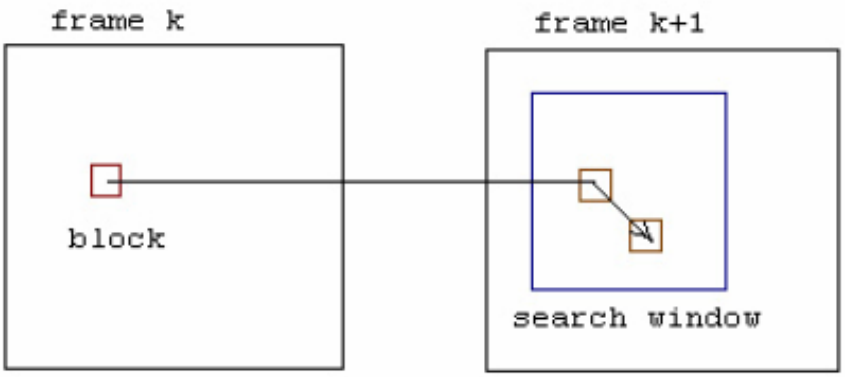

Figura 15. Algoritmul *block matching*

Considerând blocuri de dimensiune *NxN*, şi un vector de deplasament *(u, v)* al unui bloc candidat relativ la blocul şablon, cele două metrici de similaritate pot fi exprimate în felul următor:

$$SAD_{(u,v)} = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1} \left| f_t(i,j) - f_{t-1}(i+u, j+v) \right|$$

$$SSD_{(u,v)} = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1} \left(f_t(i,j) - f_{t-1}(i+u, j+v) \right)^2$$

Algoritmul *Full-Search* (FS) compară un anumit bloc *B* cu toate blocurile din fereastra de căutare şi găseşte vectorul de mişcare optim dintre toţi vectorii de mişcare din interiorul ferestrei. De aceea, acest algoritm necesită un număr mare de calcule. Din cauza complexităţii ridicate, algoritmul nu poate fi folosit în aplicaţii de timp real.

Algoritmul de căutare în trei paşi (*Three Step Search*), propusă de Koga în 1981, se bazează pe o abordare a căutării de la grosier la fin, ceea ce duce la o scădere logaritmică la fiecare pas [1]. Aşa cum se poate observa în Figura 16, la fiecare pas se verifică nouă puncte, iar punctul de minim al unei măsuri devine centrul de plecare al pasului următor. Căutarea se începe cu punctele marcate cu 0 (centrul de căutare) iar apoi cu cele marcate cu 1. Dacă în punctul 0 se obţine similaritate maximă, atunci blocul este considerat static. Dacă într-unul din punctele marcate cu 1 se obţine similaritate maximă, atunci acel punct devine noul centru, iar căutarea va continua în punctele marcate cu 2, şi aşa mai departe. Distanţa dintre punctele evaluate se înjumătăţeşte la fiecare pas al algoritmului.

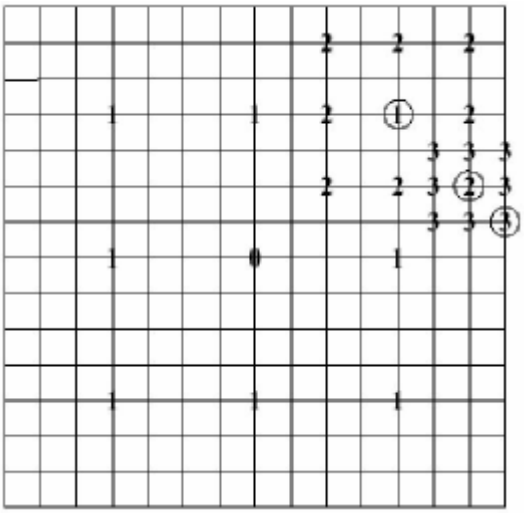

Figura 16. Algoritmul de căutare în trei paşi

Alţi algoritmi care reduc gradul de complexitate sunt: căutarea în patru paşi, căutarea logaritmică 2-D, căutarea ortogonală, căutarea în cruce, căutarea *gradient descent*.

Aplicaţii

Să se implementeze şi să se compare cei doi algoritmi de block matching prezentaţi în lucrare: *Full Search* respectiv *căutarea în trei paşi*.

Bibliografie

[1] Brad R., *Procesarea Imaginilor şi Elemente de Computer Vision*, Editura Universităţii "Lucian Blaga" din Sibiu, 2003.

[2] Gellert A., Brad R., *Context-Based Prediction Filtering of Impulse Noise Images*, IET Image Processing, Vol. 10, Issue 6, pag. 429-437, Stevenage, United Kingdom, June 2016.

[3] Gellert A., Brad R., *Studying the influence of search rule and context shape in filtering impulse noise images with Markov chains*, Signal, Image and Video Processing, Springer London, Vol. 12, Issue 2, pag. 315-322, February 2018.

[4] Ho H., *2D-3D Block Matching*, MSc Thesis, University of Auckland.

[5] http://www.cee.hw.ac.uk/hipr/

[6] http://www.ittc.ku.edu/~jgauch/research/kuim/html/00.00.html.