3. Segmentarea prin partiţionare

Histograma unei imagini gri
Segmentarea regiunilor de interes prin partiţionare histogramei
Extragerea contururilor dintr-o imagine binară

În acest laborator va fi prezentată o metodă de analiză a unei imagini gri, prin partiţionarea histogramei. După ce operaţia de segmentare a fost aplicată, obiectele din imaginea binară vor fi extrase folosindu-se analiza contururilor.

3.1 Baze teoretice

3.1.1 Histograma unei imagini

Considerându-se o imagine gri, unde L reprezintă numărul de niveluri de intensitate (pentru o imagine cu 8 biţi, L are valoarea 255), histograma nivelelor de intensitate se defineşte ca o funcţie $h(g)$ ce are ca valoare numărul de pixeli din imagine (sau dintr-o regiune) cu o intensitate $g \in [0, 1, \ldots, L]$. Un exemplu de histograma a unei imagini este prezentată în Fig. 3.1.

3.1.2 Segmentarea prin partiţionare

Tehnicile care au la bază segmentarea regiunilor din imagini au ca obiectiv principal gruparea pixelilor în funcţie de proprietăţi comune ale imaginii, cum este cazul valorilor de intensitate, textură, sau profile spectrale.

Cea mai populară metodă de segmentare a unei regiuni dintr-o imagine este *partiţionarea*[1] histogramei. Dacă valorile pixelilor dintr-o histogramă pot fi separate printr-o *valoare globală* T_G, atunci pixelii de fundal din imaginea binară de ieşire sunt reprezentaţi de acei pixeli din imaginea de intrare care au o valoare mai mică decât T_G. Pixelii obiect vor fi reprezentaţi de aceia care au o valoare mai mare sau egală cu T_G. În aceast îndrumar, valoarea de separaţie a histogramei va fi denumită *threshold*, termen consacrat din limba engleză. Matematic, operaţia de separaţie poate fi descrisă astfel:

$$t_G(x, y) = \begin{cases} 1, & \text{if } f(x, y) \geq T_G, \\ 0, & \text{if } f(x, y) < T_G, \end{cases} \tag{3.1}$$

unde $t_G(x, y)$ reprezintă imaginea binară de ieşire. În Fig. 3.1 sunt exemplificaţi paşii operaţiei de partiţionare prin regiuni a unei imagini gri.

O cerinţă în segmentarea imaginilor folosind doar un singur threshold este ca imaginea de intrare să conţină un singur obiect vizualizat, pe fondul unui fundal uniform. Această

[1] Eng. *Thresholding*

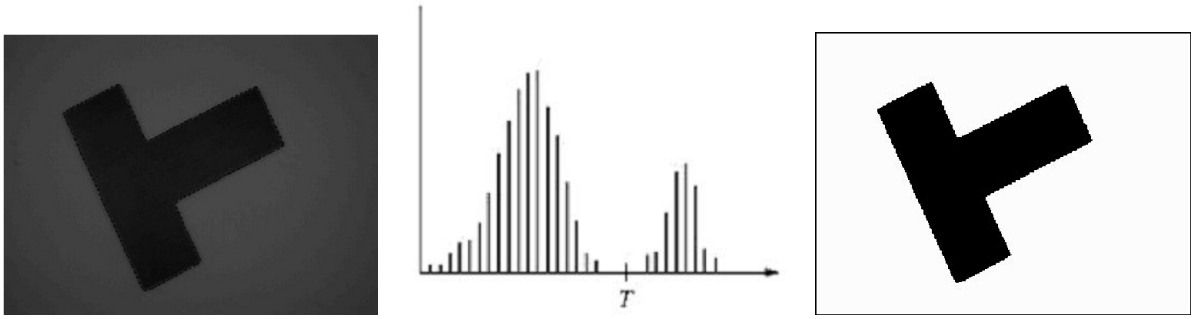

Fig. 3.1 Segmentarea unei imagini gri. (a) Imaginea de intrare. (b) Histograma imaginii de intrare. (c) Rezultatul partiţionării imaginii de intrare prin valoarea de threshold $T_G = 42$.

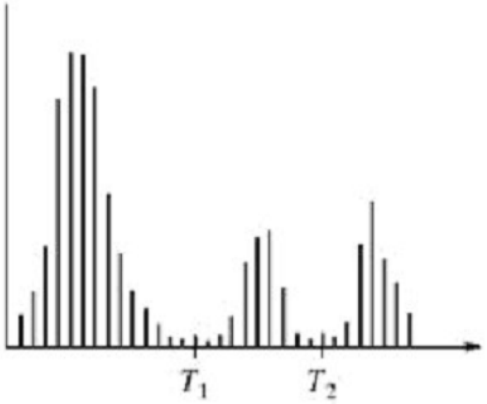

Fig. 3.2 Partiţionarea unei histograme printr-un interval de segmentare $T = [T_{low}, T_{high}]$.

problemă poate fi soluţionată prin definirea unui interval de partiţionare $T = [T_{low}, T_{high}]$:

$$t(x,y) = \begin{cases} 1, & \text{if } f(x,y) \in T, \\ 0, & \text{if } f(x,y) \notin T, \end{cases} \tag{3.2}$$

unde $f(x,y)$ reprezintă valoarea unui pixel la coordonatele (x,y) din imagine. T_{low} şi T_{high} sunt valorile minime şi maxime ale intervalului de partiţionare aplicat histogramei imaginii $f(x,y)$. Un exemplu de partiţionare a unei histograme printr-un interval de partiţionare este ilustrat în Fig. 3.2.

O metodă automată de partiţionare a histogramei este aşa numitul *threshold adaptiv*, care partiţionează imaginea având în vedere o fereastră glisantă, sau mască [**?**]. Valoarea optimală de threshold T_{opt} este calculată în funcţie de media aritmetica a valorilor pixelilor din mască.

3.1.3 Extragerea de contururi

În funcţie de tipul segmentării, de regiune (partiționare) sau de detectare a cantelor, obiectele dintr-o imagine binară pot fi reprezentate de *mulţimi grupate*[2] de pixeli obiect, în cazul segmentării regiunilor, sau de pixeli de cante conectaţi, în cazul segmentării cantelor. Principiul de bază al extragerii de contururi este ordonarea pixelilor de pe marginea obiectului segmentat şi numerotarea lor în sens orar, sau anti-orar. Procedura mai este întâlnită şi sub denumirea de *urmărirea marginilor*[3] [**?**].

[2]Eng. *Blobs*
[3]Eng. *Boundary* (*Border*) *Following*

O metoda des întâlnită în procesele de extragere a contururilor este aşa numita metodă de *codare a lanţurilor*[4]. Prin codarea lanţurilor marginea unui obiect este descrisă de o secvenţă conectată de segmente de linii drepte care au o direcţie şi o lungime specifică. De obicei, metoda are la bază conectivitatea segmentelor dintre pixelii cu 4 sau 8 vecini. În această reprezentare, denumita şi *codare a lanţurilor tip Freeman*[5], direcţia fiecărui segment este codată sub forma unei secvenţe de numere direcţionale, de la un pixel la următorul [**?**]. Un exemplu de codare a direcţiei bazată pe 8 vecini a obiectului sintetic din Fig. 3.3 este:

0 0 0 0 6 0 6 6 7 7 6 4 5 6 6 4 4 4 4 4 4 2 4 2 2 2 2 2 0 2 2 0 2

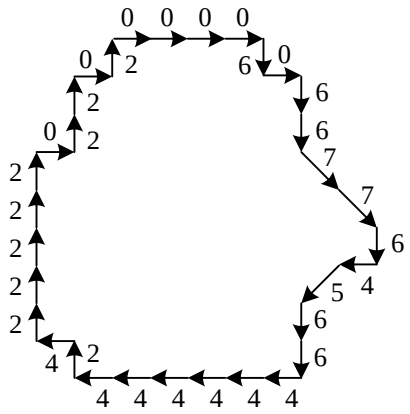

Fig. 3.3 Codarea directiei bazata pe 8 vecini a marginii unui obiect segmentat.

O asemenea margine digitală poate fi ulterior aproximată printr-un poligon. Obiectivul aproximării poligonale este acela de a se transforma marginea codată extrasă într-o formă care să descrie esenţa marginii obiectului cu cel mai mic număr de segmente posibil. O metodă utilizată cu precădere în aproximarea poligonală presupune descrierea marginilor unui obiect prin *poligonul de perimetru minim*[6] [**?**]. Din poligonul obţinut, un număr de atribute pot fi extrase, ca de exemplu *aria*, *perimetrul*, *diametrul*, *axa majoră* şi *axa minoră* împreună cu *excentricitatea* (raportul dintre axele majoră şi minoră), *curbura*, etc.

3.2 Cerinţe

1. Să se încarce şi să se convertească o imagine color într-una gri;
2. Să se calculeze histograma imaginii;
3. Să se segmenteze imaginea utilizându-se datele din histogramă;
4. Să se extragă contururile din imaginea segmentată;
5. Să se calculeze aria şi perimetrul contururilor detectate.

3.3 Codul sursă al aplicaţiei

```
1 #include <iostream>
2 #include <opencv2\core\core.hpp>
3 #include <opencv2\highgui\highgui.hpp>
4 #include <opencv2\imgproc\imgproc.hpp>
5
6 using namespace cv;
```

[4]Eng. *Chain Code*
[5]Eng. *Freeman Chain Code*
[6]Eng. *Minimum-perimeter Polygon*

```
7  using namespace std;
8
9  int main(int argc, char ** argv)
10 {
11   cout << "SVA Laborator 03: Utilizarea histogramelor si extragerea ↩
       de contururi" << endl;
12
13   Mat matImg_in, matImg_out, matImgGray;
14   string strImgPath;
15
16   RNG rng(12345);
17   vector< vector<Point> > contours;
18   vector<Vec4i> hierarchy;
19
20   // Incarcarea imaginii de pe disk
21   if (argc >= 2)
22     strImgPath = argv[1];
23   else
24     strImgPath = "Th.png";
25
26   matImg_in =  imread(strImgPath.c_str(), CV_LOAD_IMAGE_UNCHANGED);
27
28   // Verificarea incarcarii corecte a imaginii
29   if( matImg_in.empty() )
30   {
31     cout << "Imaginea " << strImgPath << " nu a putut fi incarcata" ↩
         << endl;
32     return -1;
33   }
34
35   // Conversia imaginii color intr-una gri
36   cvtColor(matImg_in, matImgGray, CV_RGB2GRAY);
37   imshow("GrayImage", matImgGray);
38   waitKey();
39
40   // Vector pentru contorizarea pixelilor
41   vector <int> vPixelsVector;
42
43   // Initializare vector
44   for (int i = 0; i < 256; i++)
45     vPixelsVector.push_back(0);
46
47   // Determinarea numarului de pixeli avand diferite nivele de gri
48   for (int i = 0; i < matImgGray.rows; i++)
49   {
50     for (int j = 0; j < matImgGray.cols; j++)
51     {
52       unsigned char ucPixelValue = matImgGray.at<unsigned char>(i,j);
53       vPixelsVector.at((int)ucPixelValue)++;
54     }
55   }
56
```

```
57  // Imaginea de afisare a histogramei imaginii
58  Mat matHistImg =  Mat::zeros(200, 255, CV_8UC1);
59
60  // Desenarea componentelor histogramei
61  for (int i = 0; i < vPixelsVector.size(); i++)
62    line(matHistImg,
63    Point(i, matHistImg.rows),
64    Point(i, 200 - (int)(vPixelsVector.at(i))/100),
65    Scalar(255, 100, 10));
66
67  // Afisarea histogramei
68  imshow("HistImg", matHistImg);
69  waitKey();
70
71  // Aplicare threshold
72  Mat matThreshold =  Mat::zeros(matImg_in.size(), CV_8UC1);
73  threshold(matImgGray, matThreshold,42,255, CV_THRESH_BINARY_INV);
74
75  // Afisarea imaginii segmentate
76  imshow("ThImage", matThreshold);
77  waitKey(0);
78
79  /*
80   * Cautare contururi.
81   * Fiecare contur va fi salvat intr-un vecor, numit contours.
82   * Fiecare contur repezinta, de fapt, un vector de puncte.
83   */
84  findContours( matThreshold, contours, hierarchy, CV_RETR_TREE, ↩
       CV_CHAIN_APPROX_SIMPLE,  Point(0, 0) );
85
86  // Desenarea fiecarui contur, folosind o culoare generata aleator
87  for ( int i = 0; i< contours.size(); i++ )
88  {
89     Scalar color = Scalar( 100, 10, 255 );
90     drawContours( matImg_in, contours, i, color, 2, 8, hierarchy );
91     imshow("Contur", matImg_in);
92     waitKey();
93  }
94
95  // Calculul ariei fiecarui contur
96  for( int i = 0; i< contours.size(); i++ )
97  {
98    double area =  contourArea(contours.at(i));
99    cout << "Aria conturului " << i << " = " << area << endl;
100 }
101
102 // Calculul perimetrul fiecarui contur
103 for( int i = 0; i< contours.size(); i++ )
104 {
105   double perimeter =  arcLength(contours.at(i), true);
106   cout << "Perimetrul conturului " << i << " = " << perimeter << ↩
        endl;
```

```
107     }
108
109     imshow("Imagine Contur", matImg_in);
110     waitKey(0);
111 }
```

3.4 Descrierea funcţiilor principale

```
62 void line(Mat& img, Point pt1, Point pt2, const Scalar& color, int ↩
       thickness=1)
```

Desenează o linie care conectează două puncte.

- `img`: imaginea pe care va fi desenată linia;
- `pt1`: punctul de start;
- `pt2`: punctul de oprire;
- `color`: culoarea liniei;
- `thickness`: grosimea liniei.

```
73 double threshold(InputArray src, OutputArray dst, double thresh, ↩
       double maxval, int type)
```

Aplică un threshold cu valoare fixă asupra tuturor pixelilor din imagine

- `src`: imaginea de intrare;
- `dst`: imaginea rezultată;
- `thresh`: valoarea de threshold;
- `maxval`: valoarea maximă care urmează a fi utilizată cu parametrul `type` de tipul THRESH_BINARY sau THRESH_BINARY_INV;
- `type`: poate fi utilizat unul dintre următoarele tipuri de threshold:
  - `THRESH_BINARY` pentru:

$$t(x,y) = \begin{cases} \text{maxval}, & \text{daca src}(x,y) > \text{thresh}, \\ 0, & \text{altfel}, \end{cases} \tag{3.3}$$

  - `THRESH_BINARY_INV` pentru:

$$t(x,y) = \begin{cases} 0, & \text{daca src}(x,y) > \text{thresh}, \\ \text{maxval}, & \text{altfel}, \end{cases} \tag{3.4}$$

  - `THRESH_TRUNC` pentru:

$$t(x,y) = \begin{cases} \text{threshold}, & \text{daca src}(x,y) > \text{thresh}, \\ \text{src}(x,y), & \text{altfel}, \end{cases} \tag{3.5}$$

  - `THRESH_TOZERO` pentru:

$$t(x,y) = \begin{cases} \text{src}(x,y), & \text{daca src}(x,y) > \text{thresh}, \\ 0, & \text{altfel}, \end{cases} \tag{3.6}$$

  - `THRESH_TOZERO_INV` pentru:

$$t(x,y) = \begin{cases} 0, & \text{daca src}(x,y) > \text{thresh}, \\ \text{src}(x,y), & \text{altfel}, \end{cases} \tag{3.7}$$

```
84 void findContours(InputOutputArray image, OutputArrayOfArrays ↩
       contours, OutputArray hierarchy, int mode, int method, Point ↩
       offset=Point())
```

Caută contururile într-o imagine binară.

- `image`: imaginea de intrare, în format de 8 biţi cu 1 canal;
- `contours`: vectorul care conţine contururile detectate. Fiecare contur este stocat sub forma unui vector de puncte;
- `hierarchy`: este un parametru opţional care conţine informaţii despre topologia imaginii;
- `mode`: modalitatea sub care se va face extragerea contururilor. Următoarele variante sunt posibile:
  - `CV_RETR_EXTERNAL`: extrage doar contururile exterioare;
  - `CV_RETR_LIST`: extrage toate contururile fără a se stabili relaţii ierarhice între acestea;
  - `CV_RETR_CCOMP`: extrage toate contururile şi le organizează sub forma unei ierarhii cu două niveluri;
  - `CV_RETR_TREE`: extrage toate contururile şi reconstruieşte o ierarhie completă a contururilor sub formă de cascadă.
- `method`: reprezintă modalitatea de reprezentare a contururilor. Poate fi aleasă una dintre următoarele combinaţii:
  - `CV_CHAIN_APPROX_NONE`: salvează toate punctele conturului considerat;
  - `CV_CHAIN_APPROX_SIMPLE`: realizează compresia segmentelor pe orizontală, verticală şi pe diagonală, lăsând nemodificate dor punctele lor terminale;
  - `CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS` aplică una dintre metodele algoritmului de aproximare Teh-Chin.
- `offset`: parametru opţional prin care fiecare punct din contur poate fi deplasat.

```
90 void drawContours(InputOutputArray image, InputArrayOfArrays contours↩
       , int contourIdx, const Scalar& color, int thickness=1, int ↩
       lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, ↩
       Point offset=Point() )
```

Desenează contururile detectate într-o imagine.

- `image`: imaginea unde vor fi desenate contururile detectate;
- `contours`: vectorul care conţine contururile detectate. Fiecare contur este salvat sub forma unui vector de puncte;
- `contourIdx`: parametru care specifică indexul conturului ce urmează a fi desenat. O valoare negativă a acestui parametru conduce la desenarea tuturor contururilor detectate;
- `color`: culoarea cu care urmează a fi desenat conturul specificat;
- `thickness`: grosimea liniei care defineşte conturul considerat;
- `lineType`: tipul liniei cu care este desenat conturul;
- `hierarchy`: parametru opţional, necesar doar atunci când se doreşte desenarea anumitor contururi;

```
98 double contourArea(InputArray contour, bool oriented=false )
```

Calculează aria unui contur.

- `contour`: vectorul de puncte care defineşte conturul;
- `oriented`: indicator care permite determinarea orientării unui contur.

```
105 double arcLength(InputArray curve, bool closed)
```

Calculează perimetrul unui contur.

- `curve`: vectorul de puncte care defineşte conturul;
- `closed`: indică dacă un contur este închis sau nu.