4. Detectarea cantelor

Calculul gradientului într-o imagine
Detectorul de cante Canny
Transformata Hough

În această lucrare vor fi studiate metode de detectare a cantelor prin evaluarea gradientului intensității pixelilor dintr-o imagine și segmentarea muchiilor obiectelor. Liniile din imagine vor fi determinate utilizându-se evaluarea proprietății de coliniaritate a pixelilor obiect folosind transformata Hough.

4.1 Baze teoretice

4.1.1 Calculul gradientului într-o imagine

Gradientul unei imagini reprezintă o schimbare direcțională a intensității pixelilor unei imagini. Acesta este utilizat cu precădere la segmentarea cantelor prin extragerea marginilor obiectelor vizualizate. Operația de determinare a gradientului este în mod normal efectuată prin detectarea tranzițiilor locale bruște ale intensităților pixelilor imaginii. Uzual, operația mai este cunoscută și sub numele de *extragerea cantelor*. Aceasta are ca principal rezultat o imagine binara având ca pixeli de fundal marginile obiectelor din imagine, sau locul unde intensitatea își schimbă valoarea brusc.

Principiul extragerii cantelor este bazat pe calculul local al gradientului imaginii prin derivate parțiale de ordinul unu sau doi. Ilustrarea grafică și numerică a gradientului imaginii de-a lungul unei direcții (x) poate fi observată în Fig. 4.1.

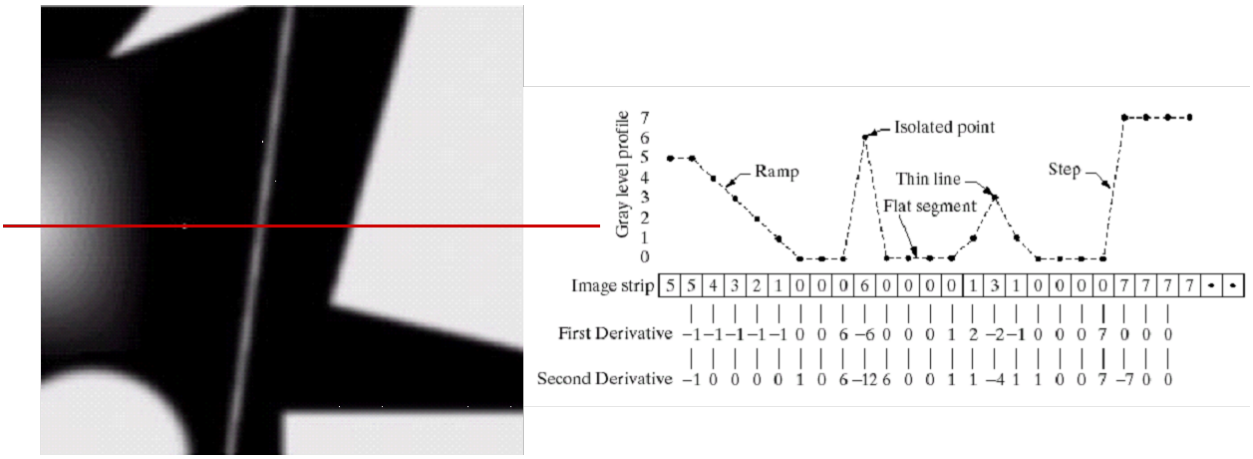

Fig. 4.1 Detectarea tranzițiilor bruște într-o imagine gri (sursa [**?**]).

Matematic, gradientul unei imagini $f(x, y)$, poate fi definit astfel:

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \end{bmatrix}. \tag{4.1}$$

unde (x, y) sunt coordonatele pixelilor din imagine.

Calculul gradientului imaginii este efectuat folosind o fereastră glisantă translatată pe imaginea de intrare. Imaginea gradient obținută este partiționată folosind relația **??**, prezentată în lucrarea de laborator numărul **??**. O problemă majoră în folosirea acestui detector de cante este dificultatea alegerii unei valori optimale de partiționareEng: *threshold*. Dacă valoarea de partiționare este prea mică, atunci imaginea binara de ieșire va conține cante false cunoscute și sub numele de *pozitive false*. Pe de alta parte, dacă valoarea de prag este prea mare, cantele reale vor fi suprimate, în cazul acesta denumite și *negative false*.

4.1.2 Detectorul de cante Canny

Detectorul de cante *Canny* are la bază partiționarea globală a imaginii gradient [**?**]. Acesta este foarte des folosit în procesarea de imagini datorită timpului de execuție scăzut și a calității foarte bune a cantelor obținute. Detectarea optimală a cantelor implică atingerea a trei obiective:

- *detectarea de cante optimală*: toate marginile din imagine trebuiesc detectate cat mai aprope de marginile reale;
- *localizarea optimală de puncte a cantelor*: poziția marginilor obținute trebuie să fie cât mai aproape posibil de marginile reale;
- *răspuns optimal al punctelor marginilor*: cantele calculate trebuie să fie cât mai subțiri posibil, adică detectorul nu trebuie să identifice două cante acolo unde există doar una singură.

Detectorul de cante Canny este bazat pe trei pași secvențiali. La început, este folosită o operație de convoluție între imaginea de intrare $f(x, y)$ și un *filtru Gaussian de netezire*:

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{4.2}$$

unde $G(x, y)$ este o funcție Gaussiană cu deviația standard σ. Acest tip de filtraj suprimă zgomotele din imaginea de intrare, din moment ce prima derivată a unei Gaussiene folosită în calculul gradientului imaginii este susceptibila la zgomot prezent în imaginea neprocesată.

Al doilea pas al algoritmului Canny este calculul magnitudinii imaginii gradient $M(x, y)$ și al direcției (unghiului) $\alpha(x, y)$:

$$M(x, y) = \sqrt{g_x^2 + g_y^2}, \tag{4.3}$$

$$\alpha(x, y) = tan^{-1}\left[g_x/g_y\right], \tag{4.4}$$

unde g_x și g_y sunt direcțiile orizontale și verticale ale imaginii gradient.

Cantele obținute sunt subțiate folosind *supresia non-maxima*, adică patru ferestre filtru sunt utilizate în specificarea unui număr de orientări discrete ale normalei cantei: orizontală, verticală, $+45°$ și $-45°$.

În final, imaginea gri obținută este binarizată folosind așa numita tehnică de *partiționare prin histerezis*, ce utilizează doua valori de partiționare: nivel jos T_L și nivel sus T_H. Pixelii cu o valoare peste T_H sunt considerați pixeli aparținând cantelor "*puternici*", iar cei cu o valoare sub T_L sunt considerați cante false. Acei pixeli ce aparțin intervalului $[T_L, T_H]$, denumiți și

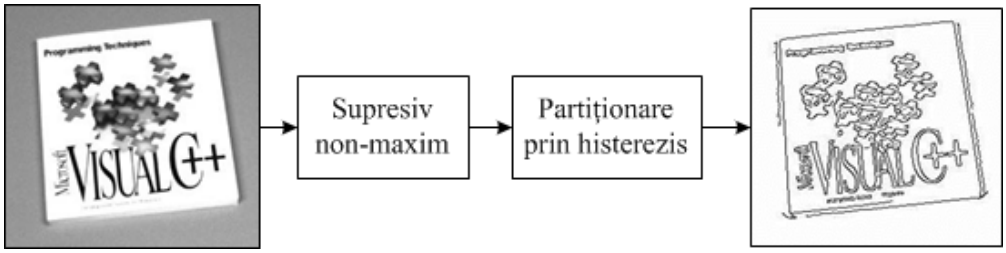

Fig. 4.2 Schema-bloc a operației de segmentare a cantelor prin metoda Canny.

pixeli "*slabi*", sunt considerați margini dacă sunt conectați deja la pixeli "*puternici*".

Potrivit [**?**], valoarea joasă de prag poate fi exprimată ca o funcție a valorii superioare:

$$T_L = 0.4 \cdot T_H, \tag{4.5}$$

Schema-bloc a metodei de segmentare a cantelor Canny este reprezentată în Fig. 4.2.

4.1.3 Transformata Hough

O problemă des întâlnita în segmentarea cantelor este aceea că de foarte multe ori cantele obținute nu sunt continue, adică între pixelii marginilor exista mici întreruperi. Acest fenomen se datorează zgomotului prezent în imaginea de intrare, iluminării neuniforme, cât și datorită unor efecte ce introduc discontinuități în imaginea de intensitate.

Transformata Hough [**?**], folosită la conectarea pixelilor cantelor, este o metodă bazată pe forma obiectului. Cu toate că orice formă de obiect poate fi reprezentată prin așa numita *transformata Hough generalizată*, în practică, datorita limitării resurselor de calcul, reprezentarea formelor se realizează prin linii, cercuri sau elipse. Transformata Hough poate fi folosită în combinație cu detectoarele de cante pentru localizarea cu precizie a marginilor obiectelor.

Principiul transformatei Hough pentru detecția de linii, reprezentat în Fig. 4.3, este bazat pe ecuația generalizată a unei linii drepte:

$$y_i = ax_i + b, \tag{4.6}$$

unde (x_i, y_i) este un punct al liniei.

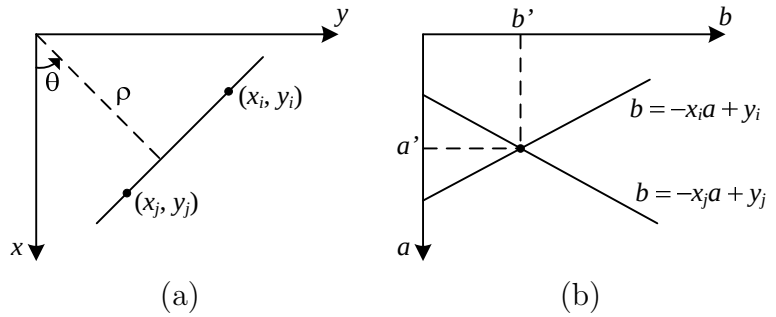

Fig. 4.3 Principiul transformatei Hough. (a) Planul xy al imaginii. (b) Spațiul parametric ab.

Prin punctul (x_i, y_i) trece un număr infinit de linii, toate satisfăcând ecuația 4.6 cu diferite valori ale coeficienților a și b. Dacă în locul planului imaginii xy ecuația liniei este reprezentată în funcție de planul ab din Fig. 4.3(b), plan numit și *spațiu parametric*, atunci ecuația unei singure linii pentru o pereche fixă (x_i, y_i) poate fi descrisă după cum urmează:

$$b = -x_i a + y_i. \tag{4.7}$$

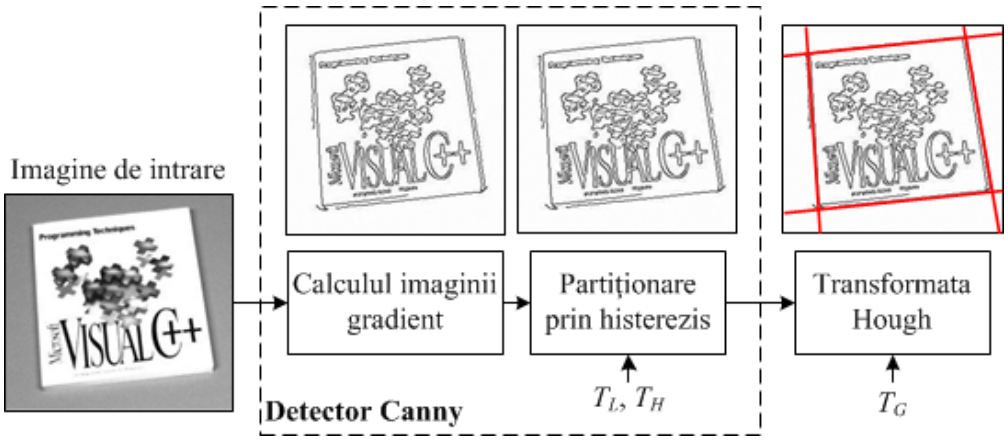

Fig. 4.4 Schema-bloc a unui sistem de extragere a liniilor dintr-o imagine utilizând detectorul Canny și transformata Hough.

După cum poate fi văzut în Fig. 4.3, dacă un al doilea punct (x_j, y_j) este colinear cu punctul (x_i, y_i), atunci, în spațiul parametric, cele doua linii corespunzătoare se intersectează într-un anumit punct (a', b').

Schema bloc completă a unui sistem de detectare a liniilor, bazat pe detectorul de cante Canny și transformata Hough, poate fi văzută în Fig. 4.4.

4.2 Cerințe

1. Să se încarce şi să se convertească o imagine către o reprezentare cu niveluri de gri;
2. Să se calculeze gradientului imaginii utilizând filtrul SOBEL;
3. Să se detecteze cantele din imagine utilizând filtrul Laplace
4. Să se detecteze cantele din imagine utilizand filtrul Canny.

4.3 Codul sursă al aplicației

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <opencv2/core/core.hpp>
4  #include <opencv2/highgui/highgui.hpp>
5  #include <opencv2/imgproc/imgproc.hpp>
6
7  using namespace cv;
8  using namespace std;
9
10 int main(int argc, char ** argv)
11 {
12   cout << "SVA Laborator 04: Calculul Gradientului" << endl;
13
14   int rezolutie_imagine = CV_16S;
15   int ordin_derivata = 0;
16   int marime_filtru = 0;
17   int raport = 3;
18   int threshold_canny_high = 100;
19   int threshold_hough = 155;
20   vector<Vec4i> linii_hough;
21   Mat img_in, img_out;
22   Mat img_filtrare_Sobel;
```

```
23  Mat img_filtrare_Laplace;
24  Mat img_segmentare_Canny;
25  string strCaleImagine;
26
27  // Incarcarea imaginii de pe disk
28  if (argc != 4)
29  {
30    cout<<"Apelare: aplicatie <cale-imagine> <ordin-derivata> <↩
        dimensiune-fereastra-glisanta>"<<endl;
31    exit(0);
32  }
33  else
34  {
35    strCaleImagine = argv[1];
36  }
37
38  img_in = imread(strCaleImagine.c_str(), CV_LOAD_IMAGE_UNCHANGED);
39
40  // Verificarea incarcarii corecte a imaginii
41  if(!img_in.data)
42  {
43    cout << "Imaginea " << strCaleImagine << " nu a putut fi ↩
        incarcata" << endl;
44    return -1;
45  }
46
47  // Conversia imaginii de intrare color in imagine gri
48  cvtColor(img_in, img_out, CV_RGB2GRAY);
49  imshow("Imagine Gri", img_out);
50  waitKey();
51  destroyAllWindows();
52
53  // Selectia ordinului derivatei
54  sscanf(argv[2], "%d", &ordin_derivata);
55
56  // Selectia marimii filtrului
57  sscanf(argv[3], "%d", &marime_filtru);
58
59  // Calculul gradientului imaginii utilizand filtrul SOBEL
60  Sobel(img_out, img_filtrare_Sobel, rezolutie_imagine, ↩
      ordin_derivata, ordin_derivata, marime_filtru);
61
62  // Conversie scala
63  convertScaleAbs(img_filtrare_Sobel, img_filtrare_Sobel);
64  namedWindow("SOBEL", CV_WINDOW_AUTOSIZE);
65  imshow("SOBEL", img_filtrare_Sobel);
66  waitKey();
67  destroyAllWindows();
68
69  // Detectarea cantelor utilizand filtrul Laplace
70  Laplacian(img_out,img_filtrare_Laplace, rezolutie_imagine, ↩
      marime_filtru, 1, 0 , BORDER_DEFAULT);
```

```
71
72    // Conversie scala
73    convertScaleAbs(img_filtrare_Laplace, img_filtrare_Laplace);
74    namedWindow("LAPLACE", CV_WINDOW_AUTOSIZE);
75    imshow("LAPLACE", img_filtrare_Laplace);
76    waitKey();
77    destroyAllWindows();
78
79    // Detectarea cantelor utilizand filtrul Canny
80    Canny(img_out, img_segmentare_Canny, threshold_canny_high*0.7, ↩
          threshold_canny_high, marime_filtru);
81    namedWindow("Filtru Canny", CV_WINDOW_AUTOSIZE);
82    imshow("Filtru Canny", img_segmentare_Canny);
83    waitKey();
84    destroyAllWindows();
85
86    // Calculul liniilor prin transformata Hough
87    HoughLinesP(img_segmentare_Canny, linii_hough, 1, CV_PI/180, ↩
          threshold_hough, 100, 20);
88
89    // Desenarea liniilor Hough pe imaginea de intrare
90    for( size_t i = 0; i < linii_hough.size(); i++ )
91    {
92      Vec4i l = linii_hough[i];
93      line(img_in, Point(l[0], l[1]), Point(l[2], l[3]), Scalar↩
            (0,0,255), 3, CV_AA);
94    }
95
96    // Afisarea liniilor Hough
97    imshow("Linii Hough", img_in);
98    waitKey();
99
100   return 0;
101 }
```

4.4 Descrierea funcțiilor principale

```
60 void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int ↩
       dy, int ksize=3, double scale=1)
```

Calculează prima, a doua sau a treia derivată a unei imagini, utilizând operatorul Sobel.

- `src`: imaginea de intrare;
- `dst`: imaginea de ieșire;
- `ddepth`: adâncimea imaginii de ieșire (`CV_8U, CV_16U, CV_16S, CV_32F, CV_64F`);
- `xorder`: ordinul derivatei pe axa x;
- `yorder`: ordinul derivatei pe axa y;
- `ksize`: mărimea ferestrei Sobel (trebuie să fie un număr impar: 1, 3, 5 sau 7);
- `scale`: factor de scalare opțional.

```
63 void convertScaleAbs(InputArray src, OutputArray dst, double alpha=1)
```

Scalează, calculează valori absolute și convertește rezultatul pe o rezoluție de 8-biți.

- `src`: imaginea de intrare;
- `dst`: imaginea de ieșire;

- `alpha`: factor de scalare opțional.

```
70 void Laplacian(InputArray src, OutputArray dst, int ddepth, int ksize↩
   =1, double scale=1)
```

Calculează Laplacianul unei imagini. Aceiași parametrii ca și la filtrul Sobel.

```
80 void Canny(InputArray image, OutputArray edges, double threshold1, ↩
   double threshold2, int apertureSize=3)
```

Detector de cante Canny.

- `image`: imagine gri pe 8-biți;
- `edges`: imagine conținând cante;
- `threshold1`: primul prag al metodei de partiționare prin histerezis;
- `threshold2`: al doilea prag al metodei de partiționare prin histerezis;
- `apertureSize`: mărimea ferestrei mască Sobel.

```
87 void HoughLinesP(InputArray image, OutputArray lines, double rho, ↩
   double theta, int threshold, double minLineLength=0, double ↩
   maxLineGap=0 )
```

Calculează segmente de linii într-o imagine binară utilizând transformata Hough probabilistică.

- `image`: imagine gri (binară) pe 8-biți;
- `lines`: liniile Hough detectate. Fiecare linie este reprezentată de un vector cu 4 elemente (x_1, y_1, x_2, y_2), unde (x_1, y_1) și (x_2, y_2) sunt coordonatele capetelor liniilor;
- `rho`: rezoluția, exprimată în pixeli, a matricei acumulator;
- `theta`: rezoluția, exprimată în radiani, a unghiului dintre punctele din matricea acumulator;
- `threshold`: valoarea de threshold aplicată matricei acumulator;
- `minLineLength`: lungimea minimă a liniilor;
- `maxLineGap`: distanța maximă admisă dintre 2 pixeli ai aceleiași linii.