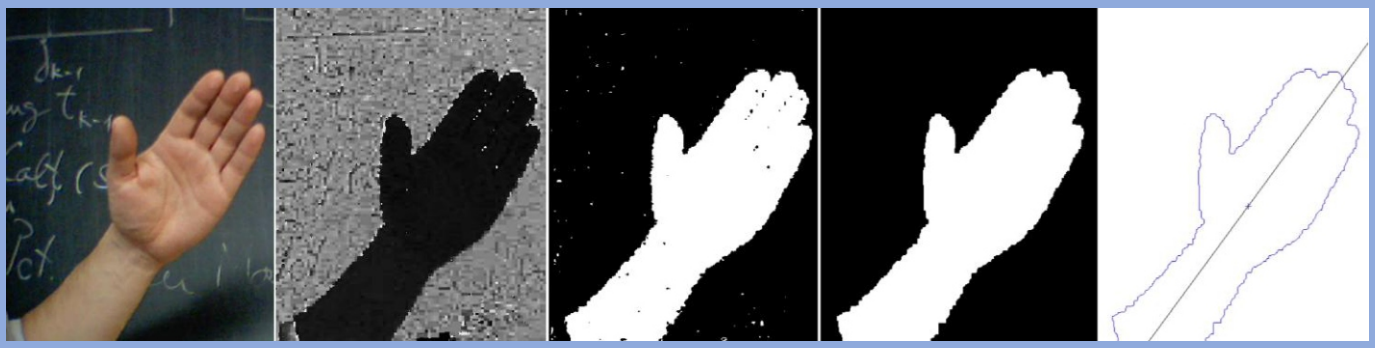
Tiberiu MARIȚA

INTERACȚIUNE OM-CALCULATOR

Îndrumător de laborator



U.T.PRESS
Cluj-Napoca, 2026
ISBN 978-606-737-841-2

Tiberiu MARIȚA

INTERACȚIUNE OM-CALCULATOR

Îndrumător de laborator

U.T.PRESS
Cluj - Napoca, 2026
ISBN 978-606-737-841-2

Editura U.T.PRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.: 0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Recenzia: Prof. dr. ing. Radu Gabriel Dănescu
Conf. dr. ing. Ion-Augustin Giosan

Pregătire format electronic on-line: Gabriela Groza

ISBN 978-606-737-841-2

Prefață

Îndrumătorul de laborator reprezintă materialul de bază pentru pregătirea lucrărilor practice la disciplina Interacțiune Om-Calculator și este destinat studenților de anul IV, ciclul de licență, specializarea Tehnologia Informației, domeniul Calculatoare și Tehnologia Informației din cadrul Facultății de Automatică și Calculatoare.

De asemenea îndrumătorul poate fi util oricui dorește să se familiarizeze cu tehnici elementare folosite în interacțiunea om-calculator bazată pe viziune computerizată și poate fi folosit ca și punct de plecare pentru implementarea proiectelor de semestru aferente disciplinei sau la implementarea unor proiecte mai complexe în cadrul lucrărilor de licență sau disertație. Îndrumătorul poate fi util și studenților de la specializările de master Interfață Om-Calculator și Human-Computer Interfaces, îndeosebi celor care nu au urmat în ciclul de licență disciplina Interacțiune Om-Calculator pentru a se familiariza cu conceptele și tehnicile de bază specifice domeniului.

Capitolele 1-3 ale îndrumătorului tratează problema segmentării imaginilor color având ca și finalitate segmentarea zonelor de piele din imagine pentru detecția mâinilor prin modele statistice sau creșterea de regiuni. Capitolul 4 este o introducere practică în detecția punctelor de interes de tip colț, care reprezintă trăsături cu localizare precisă pretabile la potrivirea și urmărirea trăsăturilor vizuale în imagini succesive. Capitolele 5-7 tratează problema detecției mișcării în secvențe de imagini, de la segmentarea obiectelor în mișcare surprinse de o cameră statică prin tehnicile de „Backgrund Subtraction" la detecția fluxului optic discret sau dens și la estimarea câmpului de mișcare prin urmărirea în imagini succesive a trăsăturilor de tip colț. Capitolele 8-9 tratează problema detecției fețelor și componentelor faciale și a detecției clipitului îmbunătățite prin validări antropomorfice. Ultimele 3 capitole abordează detecția persoanelor și părților corporale folosind trăsături Haar si HOG însoțite de modalități de corecție/îmbunătățire rezultatelor imperfecte ale implementărilor de bază prin procesare de imagini, validări bazate pe trăsături antropomorfice și algoritmi de clustering.

La fel ca și în cazul îndrumătorului de la disciplina „Procesarea Imaginilor", implementările de la activitățile practice vor avea ca și element de plecare un proiect software bazat pe biblioteca Open Computer Vision Library (OpenCV). Însă pe lângă facilitățile oferite de aceasta bibliotecă pentru citirea, stocarea și afișarea imaginilor, interacțiunea cu muse-ul sau tastatura, pentru lucrările din prezentul îndrumător se vor folosi și funcții de procesare de nivel înalt disponibile în OpenCV care implementează deja algoritmi mai complecși, de la operații morfologice, etichetare, calculul momentelor spațiale, detecția de colțuri, estimarea fluxului optic, detecția de fețe, detecția de persoane sau părți ale acestora folosind clasificatori cascadă și trăsături Haar sau clasificatori SVM și trăsături HOG.

În descrierea lucrărilor s-a urmărit o succesiune constructivă a informațiilor prezentate. Fiecare capitol începe cu prezentarea obiectivele urmărite continuată de o scurtă prezentare a conceptelor teoretice necesare, care stau la baza aspectelor practice ale implementării. În ceea ce privește implementarea activităților practice, sunt detaliate modalitățile de folosire ale funcțiilor de nivel înalt implementate în OpenCV care pot fi folosite la implementare, descrierea semnificației parametrilor de intrare/ieșire ale acestora precum și prezentarea unor șabloane de procesare cu exemple de utilizare a lor ca și puncte de plecarea pentru implementările proprii ale cerințelor lucrărilor. Majoritatea capitolelor conțin la sfârșit anexe cu implementările unor funcții ajutătoare sau șabloane de cod care pot fi folosite la implementare.

Autorul vă dorește lectură plăcută!

Cuprins

1. Segmentarea imagini color (1): conversii intre modele de culoare și construirea histogramelor de culoare

Scop: transformarea componentelor de culoare ale unei imagini color din modelul RGB24 (Red Green Blue) în modelul HSVValue (Hue Saturation Value), model invariant la variații de luminozitate; calculul și afișarea histogramelor de culoare ale componentelor H, S și V; tratarea evenimentelor de mouse pentru afișarea de informații din imaginile color.

1.1. Transformarea RGB ⇒ HSV

Modelul de culoare RGB nu este adecvat pentru a fi folosit în operațiile de segmentare a imaginilor color deoarece nu este invariant la variațiile de iluminare ale scenei (canalele RGB includ atât componenta cromatică cât si componenta de intensitate. În cazul modelului HSV componenta de intensitate (V – Value) este separată de componenta cromatică (H - Hue) și de componenta de saturație (S – Saturation)

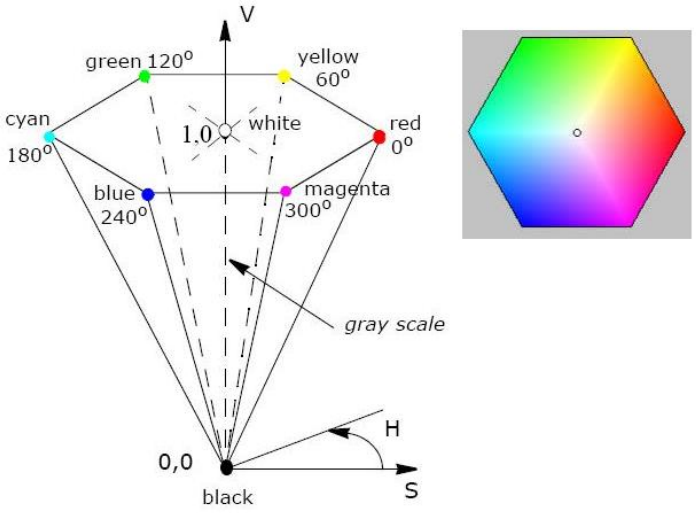


Fig. 1.1. Reprezentarea modelului (spațiului de culoare) HSV.

Ecuațiile de transformare din componentele RGB în HSV sunt [2]:

r = R/255; // r : componenta R normalizata
g = G/255; // g : componenta G normalizata
b = B/255; // b : componenta B normalizata
// Atentie declarati toate variabilele pe care le folositi de tip float
// Daca ati declarat *R* de tip *uchar*, trebuie sa faceti cast: *r = (float)R/255* !!!
M = max (r, g, b);
m = min (r, g, b);
C = M - m;
Value:
    V = M;
Saturation:
    If (C)
        S = C / V;
    Else // grayscale
        S = 0;

Hue:

If (C) {
*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 pentru 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 componenta 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

1.2. Transformarea unei imagini color din modelul RGB în HSV cu ajutorul funcției cvtColor și afișarea componentelor H, S și V

Conversia între cele 2 modele de culoare se poate face în OpenCV (Open Computer Vision Library) cu ajutorul funcției `cvtColor(src, dest, tip_concersie)` [2]. Imaginea rezultată (modelul HSV) este o matrice cu elemente având 24 biți/pixel (3 canale, fiecare cu 8 biți/pixel). Separarea imaginii cu 3 canale în 3 imagini / matrice distincte, fiecare cu cate 8 biți/pixel se poate face cu ajutorul funcției `split` din OpenCV:

```
void myBGR2HSV()
{
	char fname[MAX_PATH];
	while (openFileDlg(fname))
	{
		Mat rgb = imread(fname);
		Mat hsv;
		Mat channels[3];
		cvtColor(rgb, hsv, COLOR_BGR2HSV);
		split(hsv, channels);
		// Componentele de culoare ale modelului HSV
		Mat H = channels[0]*255/180;
		Mat S = channels[1];
		Mat V = channels[2];
		imshow("input rgb image", rgb);
		imshow("input hsv image", hsv); //  imagine fara relevanta !!
		imshow("H", H);
		imshow("S", S);
		imshow("V", V);
		waitKey();
	}
}
```

Observație: în OpenCV valorile celor trei canale ale unei imagini color RGB cu 24 biți/pixel sunt stocate în ordinea Blue, Green, Red (BGR). Din acest motiv, argumentul care specifica tipul conversiei din funcția cvtColor are prefixul Color_BGR2xxx.

1.3. Activități practice

1. Înțelegeți și implementați exemplul de la punctul 1.2 din descrierea lucrării.

2. Plecând de la implementarea de la punctul 1.2, se vor calcula histogramele (vezi capitolul 8 [4]) componentelor de culoare normalizate (0 .. 255) ale canalelor H, S, V și se vor afișa aceste histograme. Pentru afișare există în proiectul OpenCVApplication, funcția predefinita `showHistogram` (definită in modulul *Functions.cpp*)

```
showHistogram (nume_fereastra, hist, hist_cols, hist_height, true)
```

unde:

hist - vector de tip int in care se calculeaza valorile histogramei
hist_cols – lungimea vectorului histograma (256)
hist_height – inaltimea ferestrei in care se afiseaza histograma

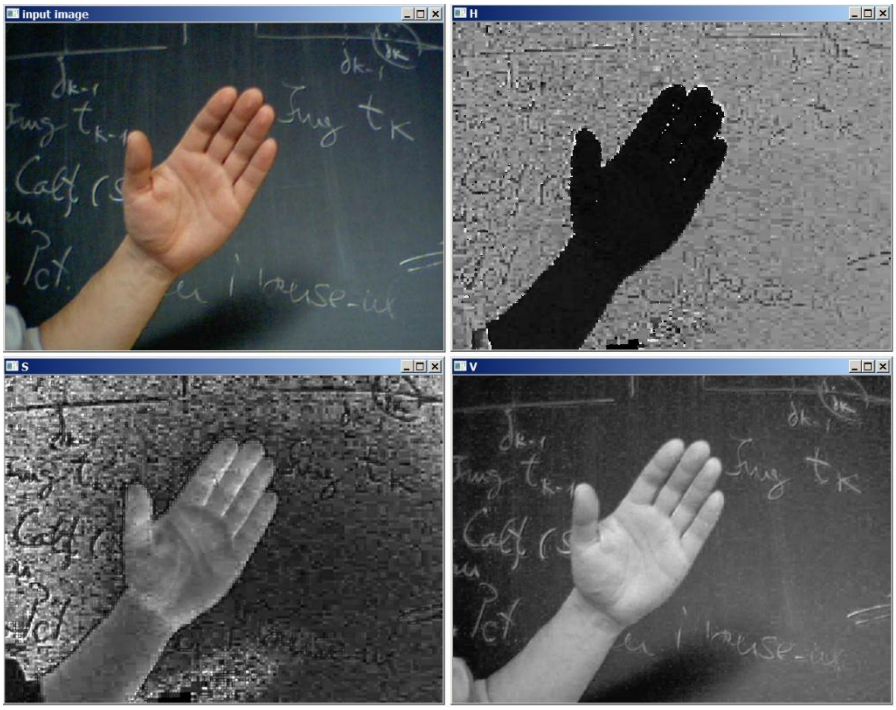


Fig. 1.2. Exemplu cu rezultatele care ar trebui obținute la punctul 2 (imaginea sursa, canalul H, canalul S, canalul V).

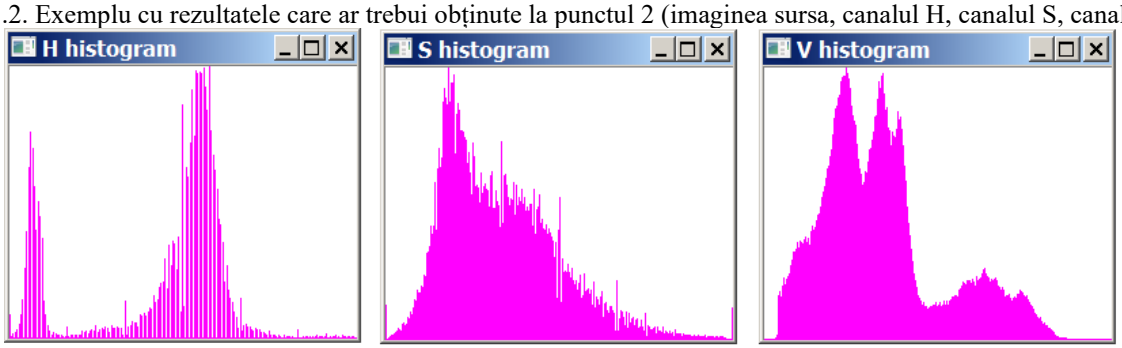


Fig. 1.3. Exemplu cu rezultatele care ar trebui obținute la punctul 2 (histograma canalului H, histograma canalului S, histograma canalului V).

3. Se va face o analiza a histogramelor H, S, V pentru toate imaginile de test ce conțin o mână și se va decide care componenta de culoare este cea mai potrivită pentru segmentarea mâinii (separarea pixelilor de mână de cei de fundal)

4. Se va binariza matricea componentei de culoare alese cu un prag arbitrar ales astfel încât sa se separe cat mai bine pixelii de mana față de cei de fundal și se va afișa rezultatul (fundalul cu negru, mana cu alb).

5. Calculați un prag de binarizare automat pentru componenta de culoare aleasa folosind implementarea algoritmului din capitolul 8.5 [4] și binarități componenta respectivă cu acest prag. Comparați rezultatul cu cel de la pct. 3. Testați binarizarea pe toate imaginile de test din arhiva *img.zip.*

6. Adăugați o funcție care afișează la linia de comanda coordonatele pixelului peste care ați făcut click și valorile componentelor de culoare H,S,V ale sale pe imaginea sursa(rgb) (vezi `myBGR2HSV()`, `testMouseClick()`, `MyCallBackFunc()`). Verificați corectitudinea implementării folosind utilitarul `ImageWatch`.

7. Adăugați o funcție care afișează pe imaginea sursa (rgb) folosind funcția `putText` valorile componentelor de culoare H,S,V ale pixelului peste care se deplasează mouse-ul (`EVENT_MOUSEMOVE`) și valorile coordonatelor curente (vezi `myBGR2HSV()`, `testMouseClick()`, `MyCallBackFunc()`).Verificați corectitudinea implementării folosind utilitarul `ImageWatch`.

Observații:
- deoarece funcției de callback puteți sa ii transmiteți un singur parametru (matricea hsv), va trebui să declarați matricea sursa (rgb) global (în afara corpului funcției);
- ca sa nu alterați sursa, la afișare (în cadrul instrucțiunii *if* aferent evenimentului *mouse-move* din funcția de callback) faceți o copie/clona a imaginii sursă (rgb) într-o alta matrice *temp* peste care veți afișa textul generat:

```
Mat temp = rgb.clone();
char msg[100];
sprintf(msg, "string fotmatare",valori);
putText(temp, msg, Point(5, 20), FONT_HERSHEY_SIMPLEX, 0.5, CV_RGB(0, 255, 0), 1, 8);
imshow("rgb", temp);
```

8. Salvați-vă ceea ce ați lucrat. Utilizați aceeași aplicație în laboratoarele viitoare. La sfârșitul laboratorului va trebui să prezentați propria aplicație cu algoritmii implementați!!!

1.4. Bibliografie

[1] Intel, Color models, https://software.intel.com/en-us/node/503873.
[2[Open Computer vision Library, Reference guide, cvtColor() function, http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor, https://docs.opencv.org/4.9.0/d8/d01/group__imgproc__color__conversions.html
[3] Open Computer vision Library, Reference guide, split() function https://docs.opencv.org/4.x/d2/de8/group__core__array.html#ga0547c7fed86152d7e9d0096029c8518a
[4] S. Nedevschi, T. Marita, R. Danescu, F. Oniga, R. Brehar, I. Giosan, C. Vancea, R. Varga, Procesarea Imaginilor - Îndrumător de laborator, editia a 2-a, Editura U.T. Press, Cluj-Napoca, https://biblioteca.utcluj.ro/carti-online-cu-coperta.html, 2023.

[5] Image Watch, Image Watch - Visual Studio 2015 | Microsoft Learn, Image Watch for Visual Studio 2022 - Visual Studio Marketplace

1.5. Anexe

Folosirea "plug-in"-ului "Image Watch"

Image Watch "plug-in" pentru Visual Studiu permite vizualizarea imaginilor (structuri de tip *Mat*) din memorie în timp ce depanați o aplicație. Folosirea plug-in-ului poate fi utila la găsirea erorilor sau la înțelegerea unei anumite parți de cod prin vizualizarea conținutului intermediar / final al imaginilor procesate.

Pentru deschiderea ferestrei "Image Watch" accesați din Visual Studio meniul "View" -> "Other Windows" -> "Image Watch":

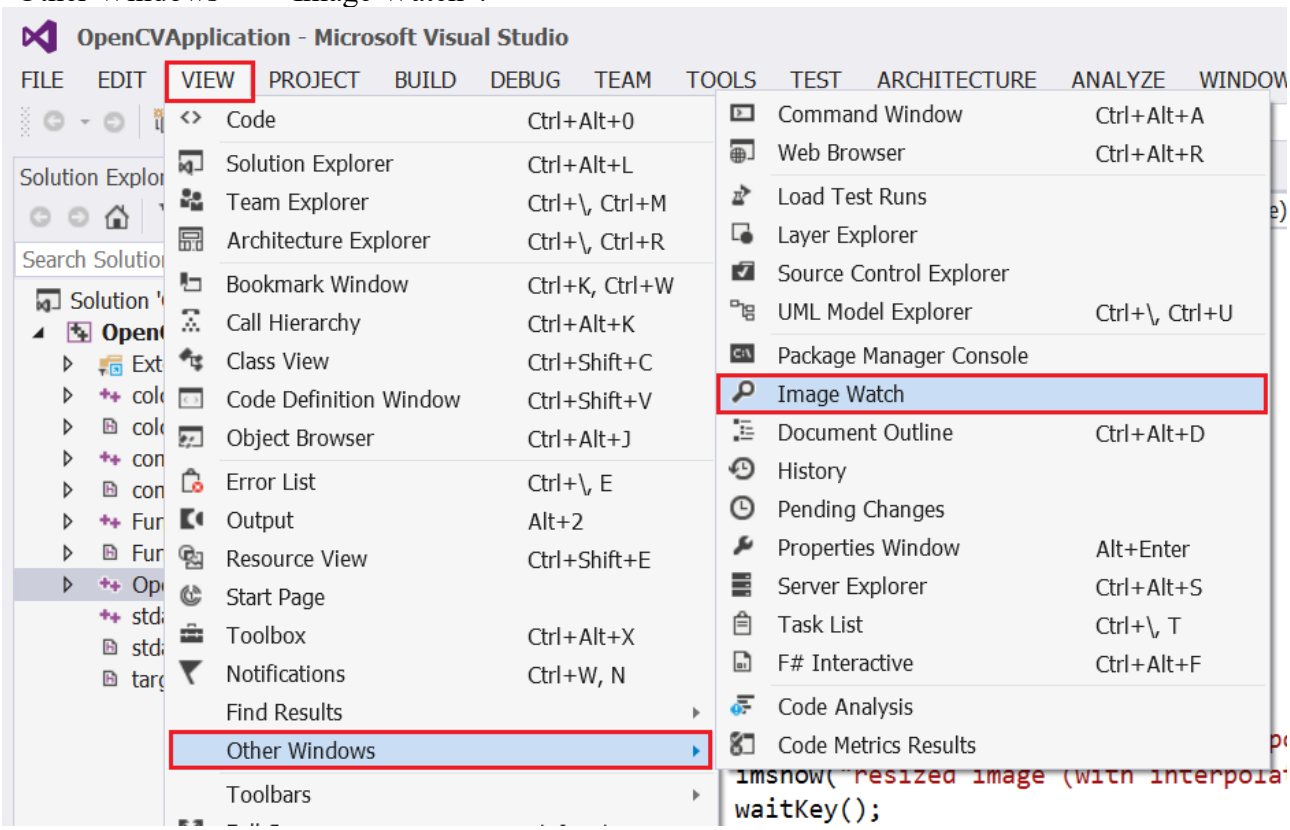


Fig. 1.4. Accesul la fereastra "Image Watch" in Visual Studio.

Pentru utilizare, inserați un break-point în funcția pe care doriți sa o depanați. În fereastra "ImageWatch" veți putea vizualiza conținutul imaginilor (variabilelor de tip Mat) aferente contextului local al funcției depanate.

Puteți face zoom-in/out pe o anumita zona din imagine (cu mouse-scroll) și puteți vizualiza conținutul canalelor imaginii la o anumita locație (pixel): x y | c0 c1 … cN (canalele ci sunt afișate în ordinea în care apar în memorie). Mai multe informații despre utilizarea Image Watch găsiți in [5].

2. Segmentarea imaginilor color (2): crearea unui model de culoare și clasificarea pixelilor pe baza modelului

Scop: construirea unui model de culoare pentru pielea umana (în particular culoarea medie a pielii mâinilor unui set de persoane), segmentarea imaginilor prin clasificarea pixelilor pe baza modelului construit, detecția mâinii sub forma unui obiect unitar (se vor aplica post-procesari) și calculul proprietăților geometrice simple ale obiectului obținut.

2.1. Construirea modelului de culoare al mâinii

Modelul de culoare pentru piele va fi o histograma globală (o distribuție de probabilitate de forma cvasi-gaussiană) calculată din multiple regiuni ale mâinilor vizibile în setul de imagini de antrenare. Parametrii modelului vor fi media si deviația standard a acestei distribuții [1].

```
#define MAX_HUE 256
// variabile globale
int histc_hue[MAX_HUE]; // histograma cumulativa (in care se aduna valorile histogramelor locale)
```

2.1.1. Inițializarea modelului

Se va adaugă o funcție pentru inițializarea modului (histogramei globale) prin setarea elementelor vectorului histogramei globale la 0:

```
void L2_ColorModel_Init()
{
	memset(histc_hue, 0, sizeof(unsigned int) * MAX_HUE);
	printf("Initialize / reset the Global Hue histogram\n");
	waitKey(2000);
}
```

2.1.2. Construirea modelului de culoare al mâinii

Se va adaugă o funcție pentru construirea modului (histogramei globale). Se va considera un set de imagini relevante (folosiți imaginile cu mana folosite și in laboratorul precedent). Pentru fiecare imagine se vor considera/selecta regiuni de interes, în care se va calcula o histograma locala a componentei de culoare Hue (normalizată în intervalul 0 ..255):

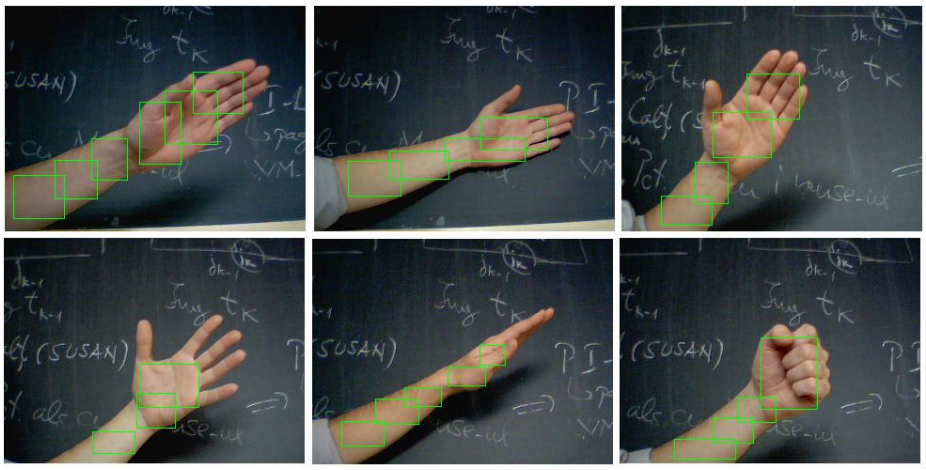


Fig. 2.1. Setul de imagini de antrenare folosit pentru construirea modelului.

Selecția regiunilor de interes se va face manual cu ajutorul mouse-ului prin tratarea evenimentelor `EVENT_LBUTTONDOWN` pentru selecția punctului de start al unei regiunii de interes rectangulare (`Pstart`), `EVENT_MOUSEMOVE` pentru selectarea dinamică a regiunii selectate (pentru implementarea acțiunii/efectului de mouse-dragging [2] și `EVENT_LBUTTONUP` pentru selecția punctului final (`Pend`) al regiunii de interes rectangulare.

```
Point Pstart, Pend; // Punctele/colturile aferente selectiei ROI curente (declarate
global)
bool draw = false;  // Variabila globala care arata starea actiune de mouse dragging:
true - in derulare

//Functia CallBack care se apeleaza la declansarea evenimentelor de mouse
void CallBackFuncL2(int event, int x, int y, int flags, void* userdata)
{
	Mat* H = (Mat*)userdata;
	Rect roi; // regiunea de interes curenta (ROI)

	if (event == EVENT_LBUTTONDOWN)
	{
		// punctul de start al ROI
		Pstart.x = x;
		Pstart.y = y;
		draw = true;
		printf("Pstart: (%d, %d)  ", Pstart.x, Pstart.y);
	}
	else if (event == EVENT_MOUSEMOVE)
	{
		if (draw == true) // actiune de mouse dragging in derulare (activ)
		{
			// desenarea  se face intr-o copie a matricii sursa
			Mat temp = srcg.clone();
			rectangle(temp, Pstart, Point(x, y), Scalar(0, 255, 0), 1, 8, 0);
			imshow("src", temp);
		}
	}
	else if (event == EVENT_LBUTTONUP)
	{
		// punctul de final (diametral opus) al ROI rectangulare
		draw = false; // actiune de mouse dragging s-a terminat (inactiva)
		Pend.x = x;
		Pend.y = y;
		printf("Pend: (%d, %d)  ", Pend.x, Pend.y);
		// sortare crescatoare a celor doua puncta selectate dupa x si y
		roi.x = min(Pstart.x, Pend.x);
		roi.y = min(Pstart.y, Pend.y);
		roi.width = abs(Pstart.x - Pend.x);
		roi.height = abs(Pstart.y - Pend.y);
		printf("Local ROI: (%d, %d), (%d, %d)\n", roi.x, roi.y,
				roi.x + roi.width, roi.y + roi.height);
		rectangle(srcg, roi, Scalar(0, 255, 0), 1, 8, 0);
		// desenarea selectiei rectangulare se face peste imaginea sursa
		imshow("src", srcg);

		int hist_hue[MAX_HUE];    // histograma locala a lui Hue
		memset(hist_hue, 0, MAX_HUE * sizeof(int));
		// Din toata imaginea H se selecteaza o subimagine (Hroi) aferenta ROI
		Mat Hroi = (*H)(roi);
		uchar hue;
		//construieste histograma locala aferente ROI
		for (int y = 0; y < roi.height; y++)
			for (int x = 0; x < roi.width; x++)
```

```
                {
                        hue = Hroi.at<uchar>(y, x);
                        hist_hue[hue]++;
                }

        int countg = 0, countl = 0;
        //acumuleaza histograma locala in cea globala
        for (int i = 0; i < MAX_HUE; i++) {
                histc_hue[i] += hist_hue[i];
                countg += histc_hue[i];
                countl += hist_hue[i];
        }
        printf("Histogram count (global / local) = %d / %d \n", countg, countl);
        // afiseaza histohrama locala
        showHistogram("H local histogram", hist_hue, MAX_HUE, 200, true);
        // afiseaza histohrama globala
        showHistogram("H global histogram", histc_hue, MAX_HUE, 200, true);
    }
}

// Functia principala in care se citesc imaginile sursa si se face legatura intre
// functia de mouse CallBack si fereastra in caare este afisata imaginea sursa
void L2_ColorModel_Build()
{
    //Mat srcg; declarata global
    Mat hsv;
    // Citestea imaginea din fisier
    char fname[MAX_PATH];
    while (openFileDlg(fname))
    {
        srcg = imread(fname);
        int height = srcg.rows;
        int width = srcg.cols;

        // Aplicare FTJ gaussian pt. eliminare zgomote / netezire imagine
    // http://opencvexamples.blogspot.com/2013/10/applying-gaussian-filter.html
        GaussianBlur(srcg, srcg, Size(5, 5), 0, 0);

        //Creare fereastra pt. afisare
        namedWindow("src", 1);

        // Componenta de culoare Hue a modelului HSV
        Mat channels[3];

        cvtColor(srcg, hsv, COLOR_BGR2HSV); // conversie RGB -> HSV

        split(hsv, channels);

        // Componentele de culoare ale modelului HSV
        Mat H = channels[0] * 255 / 180;
        Mat S = channels[1];
        Mat V = channels[2];

        // asociere functie de tratare a evenimentelor MOUSE cu ferestra "src"
        // Ultimul parametru al functiei este adresa lamatricea H
        setMouseCallback("src", CallBackFuncL2, &H);

        imshow("src", srcg);

        // Wait until user press some key
        waitKey(0);
    }
}
```

2.1.3. Postprocesarea modelului de culoare al mâinii și calcularea parametrilor modelului de culoare al mâinii

Se va adaugă o funcție pentru postprocesarea modelului, calculul parametrilor modelului (media și deviația standard) și salvarea datelor într-un fișier text.

```
void L2_ColorModel_Save()
{
	int hue, sat, i, j;
	int histF_hue[MAX_HUE]; // histograma filtrata cu FTJ
	memset(histF_hue, 0, MAX_HUE*sizeof(unsigned int));
```

Opțional, se pot filtra elementele histogramei cumulative (modelului) cu un filtru trece jos (FTJ) de tip medie aritmetica sau gaussian.

```
//Filtrare histograma Hue (optional)
#define FILTER_HISTOGRAM 1

#if FILTER_HISTOGRAM == 1
	// filtrare histograma cu filtru gaussian 1D de dimensiune w=7
	float gauss[7];
	float sqrt2pi = sqrtf(2 * PI);
	float sigma = 1.5;
	float e = 2.718;
	float sum = 0;
	// Construire gaussian
	for (i = 0; i<7; i++) {
		gauss[i] = 1.0 / (sqrt2pi*sigma)* powf(e, -(float)(i - 3)*(i - 3)
				/ (2 * sigma*sigma));
		sum += gauss[i];
	}
	// Filtrare cu gaussian
	for (j = 3; j<MAX_HUE - 3; j++)
	{
		for (i = 0; i<7; i++)
			histF_hue[j] += (float)histc_hue[j + i - 3] * gauss[i];
	}
#elif
	for (j = 0; j<MAX_HUE; j++)
		histF_hue[j] = histc_hue[j];

#endif // End of "Filtrare Gaussiana Histograma Hue"
```

De asemenea, **este recomandat** sa filtrați valorile din histograma model care sunt mai mici decât x% (x% = 1 ..10 %) din valoarea maxima a histogramei (pentru eliminarea eventualelor „zgomote” datorate selecției imprecise a regiunilor (dacă se selectează și zone de fundal din afara perimetrului mâinii, umbre etc.).

Forma histogramei cumulative obținute va fi asemănătoare unei distribuții gaussiene. Pentru aceasta distribuție se vor calcula media și deviația standard (parametrii modelului) – vezi [1]:

$$\bar{g} = \mu = \int_{-\infty}^{+\infty} g \cdot p(g) dg = \sum_{g=0}^{L} g \cdot p(g) = \frac{1}{M} \sum_{g=0}^{L} g \cdot h(g) \tag{2.1}$$

$$\sigma = \sqrt{\sum_{g=0}^{L} (g - \mu)^2 \cdot p(g)} \tag{2.2}$$

Unde:

g = hue
L = `MAX_HUE`
M = numarul de elemente din histograma model: `M += histF_hue[j]; j= 0.. MAX_HUE`

Afișarea histogramelor globale (nefiltrată și filtrată):

```
        showHistogram("H global histogram", histc_hue, MAX_HUE, 200, true);
        showHistogram("H global filtered histogram", histF_hue, MAX_HUE, 200, true);

        // Wait until user press some key
        waitKey(0);
} //end of L2_ColorModel_Save()
```

Pentru setul de imagini de antrenare (Fig. 2.1) modelul arată ca și în figura de mai jos, cu parametri aproximativi:

```
hue_mean = 17
hue_std = 5
```

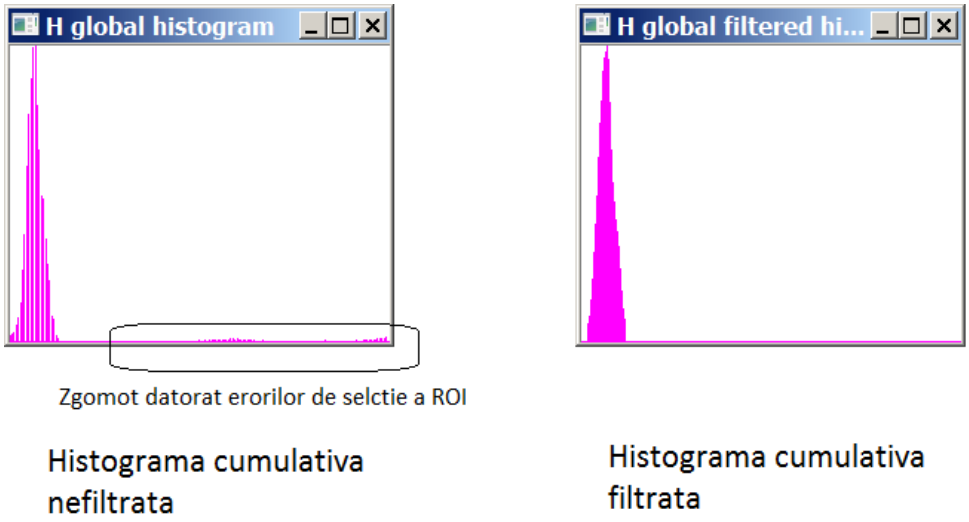


Fig. 2.2. Modelului de culoare al mâinii: histograma globala nefiltrată (stânga) si filtrată (dreapta).

Parametrii modelului (histograma modelului, media și deviația standard) se pot salva opțional într-un fișier text (pentru verificare/depanare):

```
// pregatire pt. scriere valoari model in fisier
        FILE *fp;

        // Hue
        fp = fopen("D:\\Hue.txt", "wt");

        fprintf(fp, "H=[\n");

        for (hue = 0; hue<MAX_HUE; hue++){
                fprintf(fp, "%d\n", histF_hue[hue]);
        }

        fprintf(fp, "];\n");
        fprintf(fp, "Hmean = %.0f ;\n", hue_mean);
        fprintf(fp, "Hstd = %.0f ;\n", hue_std);

        fclose(fp);
```

2.2. Segmentarea imaginii / mâinii

2.2.1. Clasificarea pixelilor din imagine

Aceasta etapă constă în clasificarea pixelilor din imagine pe baza valorii curente a componentei *hue*:

- Daca valoarea lui *hue* este în intervalul: `[hue_mean-k*hue_std .. hue_mean+k*hue_std]`, atunci pixelul respectiv se clasifică/etichetează ca fiind pixel de tip OBIECT.
- Altfel se clasifică/etichetează ca fiind de tip FUNDAL.

Unde: k este o valoare cuprinsă între 2 .. 3 (deoarece lățimea unei distribuții gaussiene este de aproximativ `6*hue_std`)

Pentru implementarea operației de segmentare, se poate folosi șablonul prezentat în `L3_ColorModel_Build()` din care se elimină apelul `setMouseCallback`. Aplicați operația de clasificare a pixelilor pe matricea H (componenta Hue). Puneți rezultatul (imaginea alb/negru) într-o matrice destinație (8biti/pixel). Afișați imaginea destinație într-o fereastra nouă.

Observație: verificați ca intervalul `[hue_mean-k*hue_std .. hue_mean+k*hue_std]`, să nu depășească intervalul pe care ați definit / scalat valorile lui hue: `[0 .. MAX_HUE]`. Adăugați o condiție de limitare.

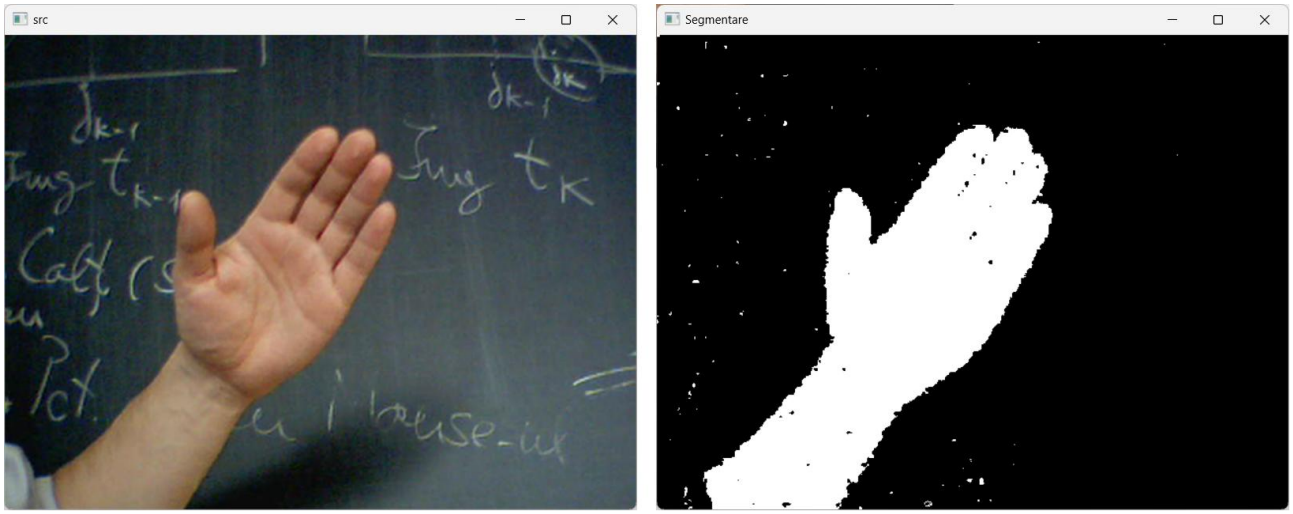


Fig. 2.3. Rezultatul segmentării pt. hue_mean = 17, hue_std = 5, k=2.5. Pixelii OBIECT sunt desenați cu alb iar cei de FUNDAL cu negru (convenția din OpenCV).

2.2.2. Postprocesarea imaginii segmentate

Imaginea segmentată (Fig. 2.3) poate prezenta erori de segmentare: pixeli de obiect clasificați greșit ca fiind pixeli de fundal și invers. O metodă simplă de postprocesare constă în eliminarea acestor zgomote (pixeli albi în interiorul obiectului și pixeli negrii în zona de fundal) prin operații morfologice (dilatare, eroziune) aplicate în mod repetat / succesiv.

Pentru operațiile morfologice se vor folosi funcțiile implementate în OpenCV. **Atenție:** operațiile sunt implementate în logica inversa (față de cum ați fost obișnuiți la laboratorul de Procesarea Imaginilor): dilatarea si eroziunea în OpenCV se fac folosind convenția: pixel OBIECT =„alb”, respectiv: pixel FUNDAL=”negru”.

Exemplu de utilizare al operațiilor morfologice din OpenCV:

```
// crearea unui element structural de dimensiune 5x5 de tip cruce
Mat element1 = getStructuringElement(MORPH_CROSS, Size(5, 5));
//eroziune cu acest element structural (aplicata 1x)
erode(dst, dst, element1, Point(-1, -1), 1);

// crearea unui element structural de dimensiune 3x3 de tip patrat (V8)
Mat element2 = getStructuringElement(MORPH_RECT, Size(3, 3));
// dilatare cu acest element structural (aplicata 2x)
dilate(dst, dst, element2, Point(-1, -1), 2);
```

Observație: pentru eliminarea zgomotului definiți/alegeți un singur tip element structural (ca formă și dimensiune). Eliminați întâi zgomotul din zonele de fundal (prin operații de eroziune). Dacă în obiectul de interes au mai rămas goluri sau acestea s-au mărit, umpleți aceste goluri prin dilatări. Ca să nu modificați aria obiectului de interes trebuie ca numărul total de eroziuni aplicate să fie egal cu numărul total de dilatări (cu același tip de element structural) - vezi exemplul din figura de mai jos:

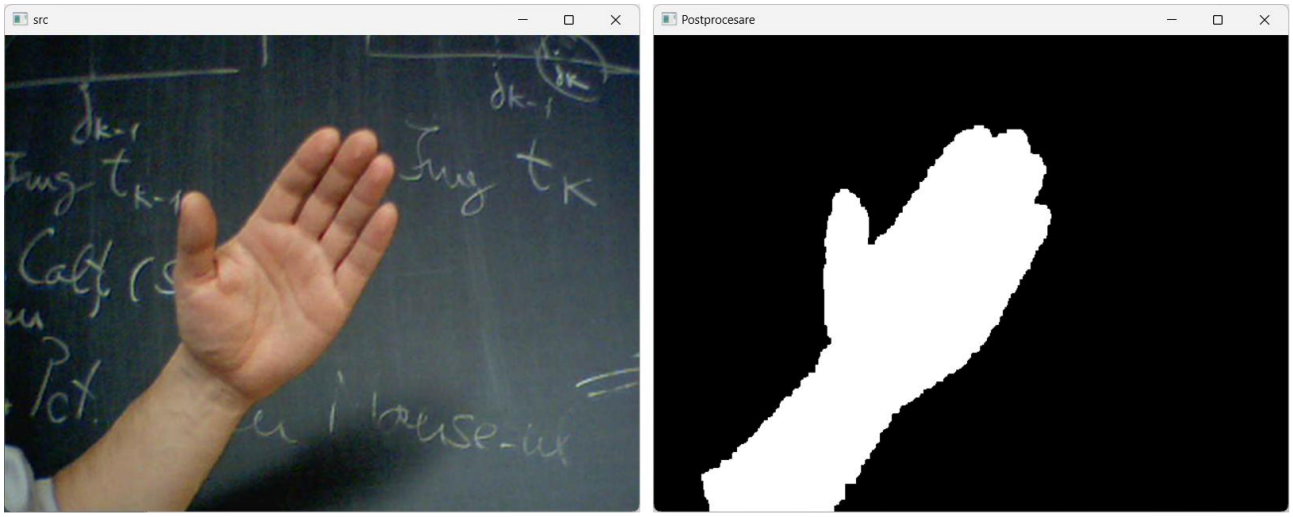


Fig. 2.4. Rezultatul după postprocesarea cu operații morfologice cu un element structural pătrat de dimensiune 3x3, aplicat în mod repetat: 2 x eroziuni + 4x dilatări + 2x eroziuni.

2.2.3. Calculul proprietăților geometrice și extragerea conturului obiectului segmentat

Dacă nu s-au putut elimina toate erorile de segmentare prin aplicarea post-procesărilor morfologice (au rămas zone mici reziduale de pixeli obiect în zona de fundal) se poate face o filtrare suplimentară a acestora pe baza de arie în funcția `Labeling` (definita în modulul *Functions.cpp*), înlocuind constanta 0 cu o valoare mai mare decât aria celui mai mare obiect segmentat eronat:

```
void Labeling(const string& name, const Mat& src, bool output_format)
{
. . .
	if (arie > 0)
	{
	double xc = m.m10 / m.m00; // coordonata x a CM al componentei conexe idx
	double yc = m.m01 / m.m00; // coordonata y a CM al componentei conexe idx
. . .
}
```

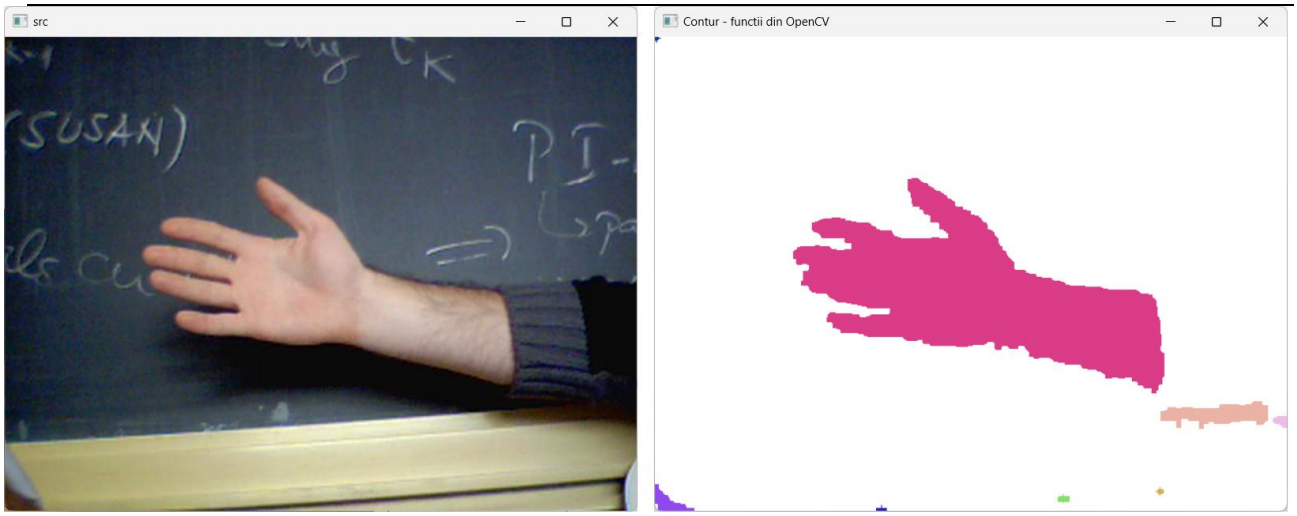


Fig. 2.5. Rezultatul după etichetarea obiectelor folosind funcții din OpenCV (funcția Labeling, cu parametrul `output_format = true` – afișare obiecte/etichete pline) pentru cazul unei segmentări imperfecte în care au rămas zone mici reziduale de pixeli obiect în zona de fundal.

După eliminarea zgomotelor (se presupune că în imagine a rămas un singur obiect) se pot extrage anumite proprietăți geometrice ale obiectului / mâinii (aria, CM, axa de alungire) și se pot extrage pixelii de contur [1]. Ca și alternativă la operațiile de calcul a proprietăților geometrice și detecție a conturului puteți folosi funcțiile din OpenCV. Un exemplu îl găsiți în funcția `Labeling` definită în modulul *Functions.cpp*. Pentru a utiliza funcțiile din OpenCV trebuie să utilizați convenția *„pixelii OBIECT desenati cu alb si pixeli de FOND cu negru”* atunci când apelați funcții de procesare pe imagini alb-negru implementate in OpenCV.

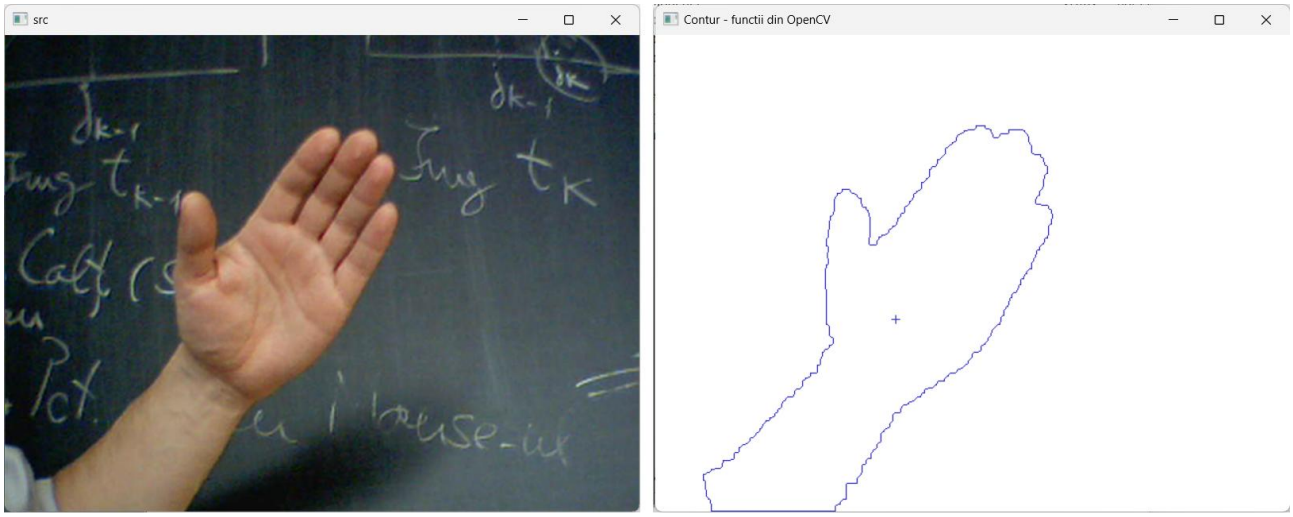


Fig. 2.6. Rezultatul după extragerea conturului și detecția centrului de masa folosind funcții din OpenCV (funcția Labeling, cu parametrul `output_format = false` – afișare obiecte prin contur)

2.2.4. Desenarea axei de alungire a mâinii

Pentru a desena axa de alungire a mâinii puteți folosi funcția din OpenCV: `line(img, P1, P2, color, thickness, 8)`. Trebuie doar să calculați 2 puncte între care desenați linia. Pentru cazul de față aceste 2 puncte ar trebui sa fie cele 2 puncte de intersecție ale axei de alungire cu marginile imaginii (*intercepts*) care se afla in intervalele:

x= [0 .. width-1] si y = [0 .. height-1] (exista doar 2 astfel de puncte !!!).

Pentru a calcula punctele *intercepts* scrieți ecuația axei de alungire obținute sub forma:

$$y\text{-}yc = slope * (x\text{-}xc) \tag{2.3}$$

unde: *(xc,yc)* - coordonatele centrului de masă al obiectului
slope = tan(teta), teta este unghiul axei de alungire [radiani]

Algoritm:

1. Înlocuiți *x* cu *0* respectiv *Width-1* în ecuația (2.3) și calculați valorile corespunzătoare ale lui y; înlocuiți *y* cu *0* respectiv *Height-1* in ecuația (2.3) și calculați valorile corespunzătoare ale lui x; cele 4 perechi de valori obținute vor fi coordonatele celor 4 puncte *intercepts*.
2. Selectați acele puncte (*intercepts*) care se află în intervalele: *x= [0 .. width-1]* si *y = [0 .. height-1]* (există doar 2 astfel de puncte).
3. Desenați linia între aceste 2 puncte.

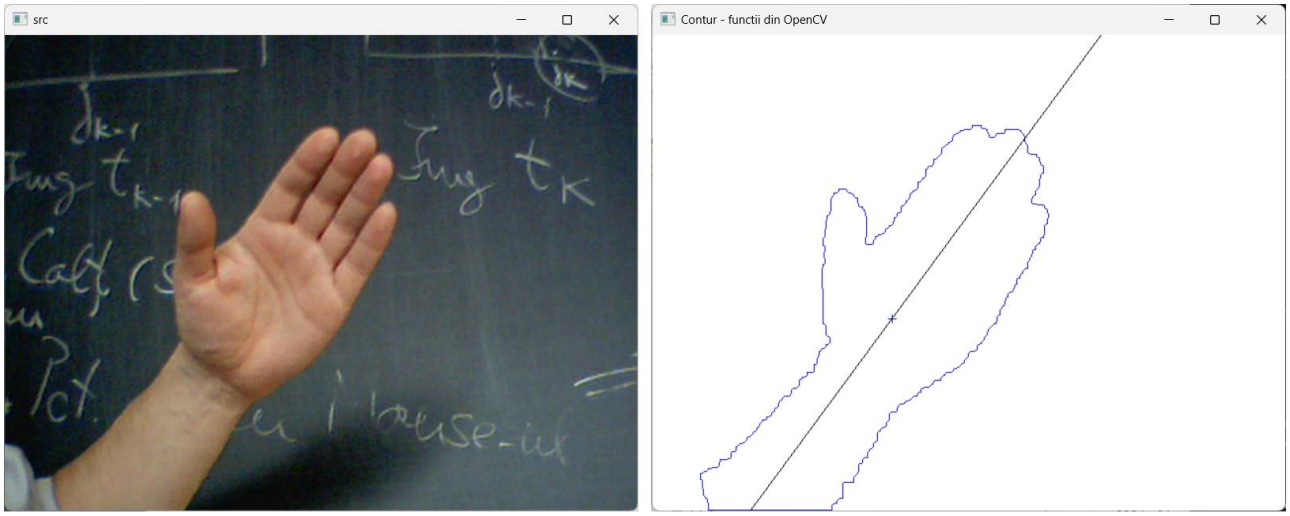


Fig. 2.7. Rezultatul după extragerea conturului, afișarea centrului de masă și a axai de alungire calculate.

2.3. Activități practice

Presupunând ca modelul de culoare este deja construit pentru setul de imagini dat:

```
hue_mean = 17
hue_std = 5
```

se va adăuga o funcție de procesare care să integreze următorii pași de procesare descriși în capitolul 2.2:

a. Segmentarea imaginilor prin clasificarea la nivel de pixel (cap. 2.2.1). Se va alege o valoare optimă pentru k (k = 2...3), pentru fiecare imagine în parte din setul dat.
b. Postprocesarea imaginii segmentate pentru eliminarea zgomotelor folosind filtre morfologice (cap. 2.2.2).
c. Extragerea conturului mâinii segmentate și calculul proprietăților geometrice ale obiectului segmentat (integrați o copie a funcției `Labeling` din modulul *Functions* (cap. 2.2.3).
d. Desenarea axei de alungire a mâinii peste imaginea finala obținută la punctul c. (capitolul 2.2.4), Inserați codul aferent acestui calcul în varianta proprie/modificată a funcției `Labeling`.

2.4. Bibliografie

[1] S. Nedevschi, T. Marita, R. Danescu, F. Oniga, R. Brehar, I. Giosan, C. Vancea, R. Varga, Procesarea Imaginilor - Îndrumător de laborator, editia a 2-a, Editura U.T. Press, Cluj-Napoca, https://biblioteca.utcluj.ro/carti-online-cu-coperta.html, 2023.

[2[Open Computer vision Library, Reference guide, Mouse as a Paint-Brush, https://docs.opencv.org/4.x/db/d5b/tutorial_py_mouse_handling.html

[3] OpenCV, On-line reference manual and tutorials: Erosion & Dilation, https://docs.opencv.org/4.9.0/db/df6/tutorial_erosion_dilatation.html

[4] OpenCV, On-line reference manual and tutorials: Find Contours and Draw Contours https://docs.opencv.org/4.9.0/df/d0d/tutorial_find_contours.html, https://docs.opencv.org/4.9.0/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0

https://docs.opencv.org/4.9.0/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc

[5] OpenCV, On-line reference manual and tutorials: Moments, https://docs.opencv.org/4.9.0/d8/d23/classcv_1_1Moments.html, https://docs.opencv.org/4.9.0/d0/d49/tutorial_moments.html

3. Segmentarea imaginilor color (3): segmentarea bazată pe regiuni

Scop: implementarea unui algoritm de segmentare bazat pe regiuni, de tip „region-growing" folosind parametrii modelului de culoare construit anterior (lucrarea 2).

3.1. Considerații teoretice

O altă categorie de metode de segmentare a imaginilor constă în identifica de regiuni *(componente conexe*) care satisfac anumite criterii de omogenitate, bazate pe trăsături derivate din componentele spectrale. Aceste componente sunt definite în spațiul de culoare considerat.

Regiune:= setul maximal de pixeli pentru care este satisfăcută o condiție de uniformitate (predicat de omogenitate). Sunt două categorii de metode prin care se poate împărții imaginea in regiuni uniforme:
(a) *Region growing*: Regiunile uniforme sunt obținute prin creșterea unui bloc/seed prin unirea cu alți pixeli sau blocuri de pixeli similari
(b) *Region splitting*: Regiunile uniforme sunt obținute prin împărțirea unor regiuni mai mari care nu sunt omogene în regiuni mai mici care sunt.

În această lucrare se va implementa o metoda de tip „region growing". Metoda *region growing* are la bază un proces iterativ prin care regiuni ale imaginii sunt fuzionate/crescute începând de la regiuni primare (care pot fi pixeli sau alte regiuni mici – celule de bază). Iterațiile de creștere se opresc atunci când nu mai sunt pixeli de procesat!

Algoritm:

1. Se segmentează imaginea în celule de bază (dimensiune >= 1 pixel).
2. Fiecare celulă este comparată cu vecinii ei folosind o măsura de similaritate. În caz afirmativ (valoarea metricii de similaritate < prag) celulele sunt fuzionate într-un fragment mai mare și se actualizează trăsăturile fragmentului fuzionat (de obicei prin mediere ponderata).
3. Se continuă procesul de creștere al fragmentului prin examinarea tuturor vecinilor pană când nu se mai pot realiza fuziuni.
4. Se trece la următoarea celula rămasă nemarcata (neetichetată) și se repetă pașii 2-3. Algoritmul se oprește atunci când nu au mai rămas celule nemarcate.

3.2. Detalii de implementare

3.2.1. Șablonul de procesare recomandat

1. Se alege un punct de start (seed point) ales cu ajutorul mouse-ului (prin tratarea mesajului corespunzător - vezi L1 sau L2). Puteți folosi ca și exemplu / punct de plecare funcțiile `L2_ColorModel_Build` si `CallBackFuncL2` prezentate in L2.
2. În funcția de procesare principală pe care o apelați din meniul principal (vezi `L2_ColorModel_Build`) este recomandat sa faceți următoarele:
   - Sa filtrați imaginea sursă cu un filtru trece jos Gaussian (ex. de dimensiune 5) pentru eliminarea zgomotelor
   - Sa convertiți imaginea sursă din formatul BGR în HSV și să transmiteți pointerul matricei H (Hue) funcției CallBack de tratare a evenimentului mouse-ului (vezi `L2_ColorModel_Build`).
3. Codul aferent algoritmului de region growing să îl adăugați în funcția CallBack de tratare a evenimentului mouse-ului.

3.2.2. Algoritmul de creștere a regiunilor

Codul aferent algoritmului de region growing este recomandat să îl adăugați în funcția CallBack de tratare a evenimentului mouse-ului. Componenta de culoare pe care se va lucra este H (Hue).

Se va aloca o matrice de etichete *labels* de dimensiunea imaginii. Se inițializează cu 0 fiecare element al ei. De fiecare dată când un punct se va adăuga la regiunea curentă, se va marca poziția corespondentă din matricea *labels* cu valoarea etichetei curente procesate (1 în cazul implementării de față). Matricea l*abels* va fi folosită și ca un indicator de parcurgere al pixelilor (0 daca pixelul încă nu a fost parcurs, 1 daca pixelul a fost parcurs).

```
Mat labels = Mat::zeros(H.size(),CV_16UC1);
```

1. Dacă nu ați aplicat în prealabil un filtru trece jos (ex. Gaussian) pe imaginea sursă după citirea ei de pe disc, este recomandat să calculați o valoare medie (*Hue_avg*) a lui Hue într-o vecinătate de dimensiune w x w (w= 3, 5, 7 ...) în jurul punctului de start (*x,y*) ca și o măsura de precauție suplimentară pentru a nu lua în considerare în mod accidental zgomote.
2. Se adaugă coordonatele pixelului de start într-o lista FIFO. Pentru implementarea listei FIFO puteți folosi clasa container QUEUE[1] (vezi anexa). Ați mai folosit-o si la laboratorul de Procesarea Imaginilor la implementarea algoritmului de etichetare prin metoda BFS (Breadth First Search) [2]. Se setează eticheta curenta k = 1 (în final toți pixeli din regiune vor trebui sa aibă eticheta 1) si N = 1 (numărul de pixeli din regiune)
3. Algoritmul de region growing:
   while (coada nu este goala)
   - pentru fiecare vecin (i,j) al pixelului din poziția „bottom” a listei:
     - daca *labels(i,j) =0* (pixel neprocesat încă) și *abs(Hue(i,j)-Hue_avg) < T*:
       - adaugă coordonatele (i,j) în lista FIFO la poziția top (la regiunea curentă)
       - pixelul (i,j) primește eticheta k: `labels(i,j)=k`
       - se actualizează valoarea medie a lui Hue:

$$Hue\_avg = \frac{N * Hue\_avg + Hue(i,j)}{N+1}$$

       - incrementează N
   - șterge elementul de pe poziția bottom (cel mai vechi element din lista FIFO)

   end while
4. Se vor afișa pixelii din regiunea curentă (labels(i,j) = 1) în imaginea destinație cu alb (cei de fundal vor fi negri) - declarați matricea destinație ca si o matrice cu un canal in funcția de CallBack)

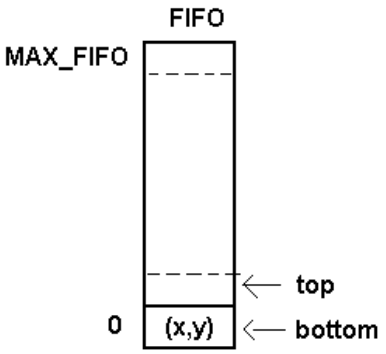


Fig. 3.1. Reprezentarea unei liste FIFO

5. Imaginea segmentată poate prezenta erori de segmentare (goluri în interiorul mâinii segmentate). O metoda simplă de postprocesare constă în eliminarea acestor zgomote (pixeli

negri din interiorul obiectului alb) prin operații morfologice (dilatare, eroziune) aplicate repetat / succesiv (vezi L2). Găsiți cea mai buna varianta de combinare a acestor operații morfologice, fără sa schimbați aria obiectului sau sa deteriorați excesiv conturul obiectului.

6. Afișați matricea destinație cu funcția `imshow` în funcția de CallBack (în instrucțiunea *if* în care tratați evenimentul de mouse, la sfârșit). În felul acesta veți putea testa rezultatul obținut pentru click-uri succesive în diverse puncte ale mâinii în cadrul funcției de procesare curente.

Observație
Valoarea pragului *T* se va alege în funcție de parametrii modelului de culoare construit anterior (lucrarea 2) *T= konst*hue_std*, unde *konst* = 2 .. 3;

3.3. Activități practice

1. Adăugați/implementați funcțiile de procesare necesare pentru a implementa algoritmul de creștere a regiunilor aplicat pe canalul de culoare H (hue) urmărind indicațiile din mersul lucrării (cap. 3.2.1 și 3.2.2)
2. Adăugați o noua funcție de procesare care sa aplice algoritmul de creștere a regiunilor pe canalul V (Value – canalul de intensitate). Conținutul funcției de CallBack nu este nevoie să îl modificați, doar în funcția de procesare principală sa-i transmiteți ca si parametru canalul V (Value) – canalul de intensitate grayscale. Comparați rezultatul cu segmentarea obținută pe canalul H (Hue).

3.4. Temă de casa / proiect

Algoritmul se poate aplica și automat (fără a selecta punctul de start manual) în felul următor:

1. Se considera primul pixel din imagine (neetichetat / neparcurs încă) ca seed point (k=1)
2. Sa aplică pașii 2 – 3 (din cap. 3.2.2) (cu diferența ca pixelii din regiunea curentă primesc eticheta k)
3. Se incrementează eticheta curentă (k++) și se repetă pașii 1 – 3 până când se epuizează toți pixelii din imagine

Implementarea va fi asemănătoare cu cea a algoritmului de etichetare prin metoda BFS (Breadth First Search) [2], cu diferența că se aplica pe o matrice care conține valori în intervalul 0 .. 255, în loc de o imagine/matrice binara (alb/negru).

3.5. Bibliografie

[1] Clasa Container Queue, https://learn.microsoft.com/en-us/cpp/standard-library/queue-class?view=msvc-170

[2] S. Nedevschi, T. Marita, R. Danescu, F. Oniga, R. Brehar, I. Giosan, C. Vancea, R. Varga, Procesarea Imaginilor - Îndrumător de laborator, editia a 2-a, Editura U.T. Press, Cluj-Napoca, https://biblioteca.utcluj.ro/carti-online-cu-coperta.html, 2023.

3.6. Anexe

Pentru implementarea listei FIFO puteți folosi clasa container QUEUE

```
#include <queue>
using namespace std;
```

Șablonul de procesare care se va insera în funcția de CallBack:

```
queue <Point> que;

k=1; //ethiceta curenta
N=1; // numarul de pixeli din regiune
que.push(Point(x,y)); // adauga element (seed point) in coada
// acesta primeste eticheta k

while (!que.empty())
{      // Retine poz. celui mai vechi element din coada
       Point oldest=que.front();
       que.pop();   // scoate element din coada

       int xx=oldest.x; // coordonatele lui
       int yy=oldest.y;

       // Pentru fiecare vecin al pixelului (xx, yy) ale carui coordonate
       // sunt in interiorul imaginii
              // Daca abs(hue(vecin) – Hue_avg)<T si labels(vecin) == 0
                     // Aduga vecin la regiunea curenta
                     // labels(vecin) = k
                     // Actualizeaza Hue_avg (medie ponderata)
                     // Incrementeza N

}
```

4. Detecția punctelor de interes de tip colț

Scop: Detecția punctelor de interes de tip colț pe baza implementării din OpenCV a metodei Harris de detecție a lor.

4.1. Metoda Harris de detecție a colturilor

Un punct de interes (*interest point*) este un punct care are o poziție bine definită în imagine și poate fi detectat într-un mod robust. El poate fi un minim sau maxim local de intensitate, o terminație a unei linii, un punct de pe un contur în care curbura are un maxim local etc. Pentru ca un punct de interes sa fie colț sunt necesare analize și validări suplimentare.

Un colț se definește ca fiind un punct în care se intersectează cel puțin 2 muchii cu orientări diferite respectiv în a cărui vecinătate exista cel puțin 2 direcții dominante (diferite) ale gradientului. Punctele de colt au o localizare bine definita în imagine iar detecția lor necesita folosirea unor metode robuste care să permită detecția aceluiași set de colțuri în condiții de iluminare diferită, sau dacă imaginea obiectului suferă anumite transformări geometrice (rotație, scalare, transformare de perspectivă).

Există numeroase metode de detecție a punctelor de tip colț. Dintre acestea se va prezenta metoda Harris. Ea a fost propusă de Moravec [1] și a stat la baza a numeroase variante derivate: Harris & Stephens sau Shi/Kande & Tomasi [1] care se regăsesc și în implementările din OpenCV.

Pentru localizarea punctelor de colț metoda Harris propune utilizarea matricei de auto-corelație (covarianța a derivatelor) *M* care conține toți operatorii diferențiali care descriu geometria "suprafeței" imaginii în punctul (x,y) (din punctul de vedere al variațiilor locale de intensitate) [7]:

$$\boldsymbol{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{4.1}$$

unde:

I_x, I_y sunt componentele orizontale respectiv verticale ale gradientului [2] care în ecuația (4.1) sunt ridicate la pătrat sau înmulțite intre ele și sunt ponderate cu o fereastra *w* de ponderi gaussiene.

Matricea de auto-corelație fiind simetrică fața de diagonala principală, poate fi diagonalizată prin rotația axelor de coordonate la forma:

$$\boldsymbol{M} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \tag{4.2}$$

unde:

λ_1, λ_2 sunt valorile proprii ale matricei de auto-corelație

Pe baza matricei de auto-corelație se poate calcula matricea de răspuns ***R**(x,y)* care este o măsura a „tăriei" colțului în fiecare pixel *p(x,y)*:

$$\boldsymbol{R}(x,y) = \det(\boldsymbol{M}) - k * trace(\boldsymbol{M}) \tag{4.3}$$

unde:

$det(\boldsymbol{M}) = \lambda_1 * \lambda_2 = A*B - C^2$ este determinantul matricei **M**
$trace(\boldsymbol{M}) = \lambda_1 + \lambda_2 = A + B$ este urma matricei **M**
$k = 0.04 \ldots 0.15$ (este o constantă)

Punctele de colț vor corespunde unor locații în care matricea de răspuns are o valoare pozitivă mare, respectiv valorile proprii λ_1, λ_2 au și ele o valoare mare. Punctele în care funcția de răspuns are

o valoare negativă iar între valorile proprii este o diferența semnificativa de valoare corespund unor puncte de muchie. Zonele din imagine în care funcția de răspuns respectiv valorile proprii au o valoare mica corespund unor zone netede din imagine (fără textură, fără muchii sau colțuri) – vezi Fig 4.1.

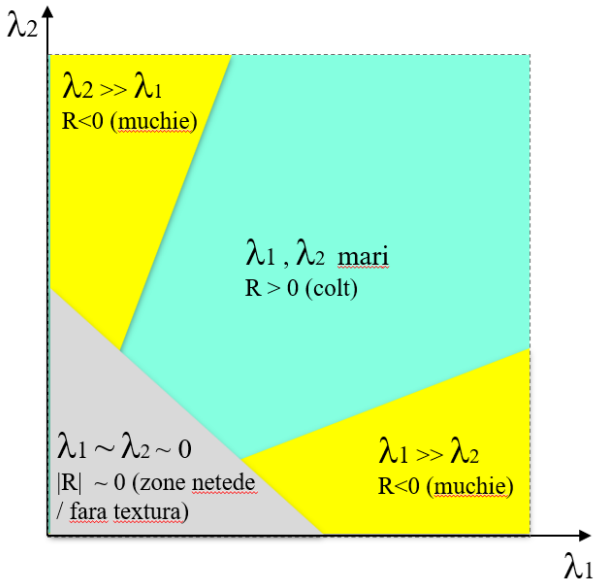


Fig. 4.1. Reprezentare vizuala a semnificației valorilor proprii ale matricei de auto-corelație **M** si ale matricei de răspuns **R**.

Metoda Harris de detecție a colțurilor poate fi sintetizată prin algoritmul următor [3]:

Metoda Haris:

1. Pentru fiecare pixel din imagine ***p****(x, y)* se calculează matricea de auto-corelație **M**.
2. Se calculează "harta" (matricea) funcției de răspuns ***R****(x, y)* (în fiecare pixel ***p****(x, y)*).
3. Se filtrează ***R****(x, y)* pe baza unui prag *T*: daca ***R****(x,y)* $< T \Rightarrow$ ***R****(x,y)=0*. Pragul *T* se poate alege prin analiza histogramei valorilor matricei ***R****(x,y)*.
4. Se aplică algoritmul de supresie a non-maximelor pe matricea **R** $\Rightarrow$ se reține maximele locale din fiecare vecinătate de dimensiune *wxw* a fiecărei locații (x,y) (valorile din ***R*** care nu corespund unor maxime locale se elimina: ***R****(x,y)=0*).
5. Toate punctele ramase (***R****(x,y)>0*) vor fi colturile detectate.
6. Opțional se poate limita numărul maxim de colțuri raportate.

4.2. Detecția de colțuri folosind funcția OpenCV *goodFeaturesToTrack()*

Implementarea a doua variante ale metodei Harris se regăsește în funcția OpenCV *goodFeaturesToTrack*. În funcție de valoarea parametrului de intrare *useHarrisDetector* este apelată metoda Harris (true) sau Shi-Tomasi (false). Semnificațiile parametrilor funcției sunt descrise mai jos:

```
// Lista/vector de iesire care va contine coordonatele (x,y) ale colturilor detectate
(output)
	vector<Point2f> corners;

// Nr. maxim de colturi luate in considerare. Daca nr. de colturi > maxCorners se vor
considera un numar de maxCorners colturi care ay cele mai mari valori ale functiei de
raspuns
```

```
	int maxCorners = 100;

// Factor cu care se multiplica masura de calitate a celui mai bun colt (val. proprie
minima) pt. metoda Shi-Tomasi respectiv valoarea functiei de raspuns R (Harris)
ex: qualityMeasure = 1500, qualityLevel = 0.01 ⇒ colturile cu  valoarea mai mica de
1500*0.01 sunt rejectate:
	double qualityLevel = 0.01;
```

Corner quality:

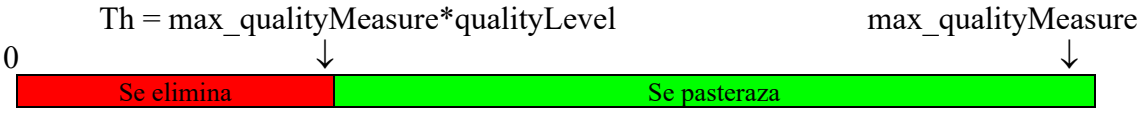


Fig. 4.2. Reprezentare vizuala a modulului de calcul al pragului *Th* folosit la discriminarea valorilor matricei de răspuns R în puncte de colț / non-colț.

```
// Raza vecinatatii pe care se aplica etapa de supresie a non-maximelor
	double minDistance = 10;

// Dimensiunea ferestrei de integrare w folosita in calculul matricei de autocorelatie M
(covarianta a derivatelor)
	int blockSize = 3; // 2,3, ...

// Selectia metodei de detectie: Harris (true) sau Shi-Tomasi (false).
	bool useHarrisDetector = true;

// Factorul k (vezi descrierea teoretica a metodei)
	double k = 0.04;

// Apel functie
	goodFeaturesToTrack( Src_image_gray,
		corners,
		maxCorners,
		qualityLevel,
		minDistance,
		Mat(), //masca pt. ROI - optional
		blockSize,
		useHarrisDetector,
		k );
```

Funcția *goodFeaturesToTrack* detectează cele mai proeminente (“de calitate” = care au funcția de răspuns cea mai mare) colțuri din imagine

1. Calculează măsura de calitatea a colțurilor în fiecare pixel al imaginii folosind funcțiile `cornerMinEigenVal()` dacă `useHarrisDetector = false` sau `cornerHarris()` dacă `useHarrisDetector = true`.
2. Colțurile cu valoare *quality_measure* > *Th* se păstrează (zona verde), celelalte (zona roșie) se elimină (vezi Fig. 4.2). Colțurile rămase se sortează descrescător.
3. Se aplică *non-maxium suppression*: funcția elimină orice colț pentru care există într-o vecinătate cu raza maxDistance un alt colț mai “puternic” (cu *quality_measure* mai mare).

Algoritmul Harris este prezentat în detaliu în cursul nr. 4 al disciplinei Interacțiune Om-Calculator [4] iar metoda Shi-Tomasi in [5].

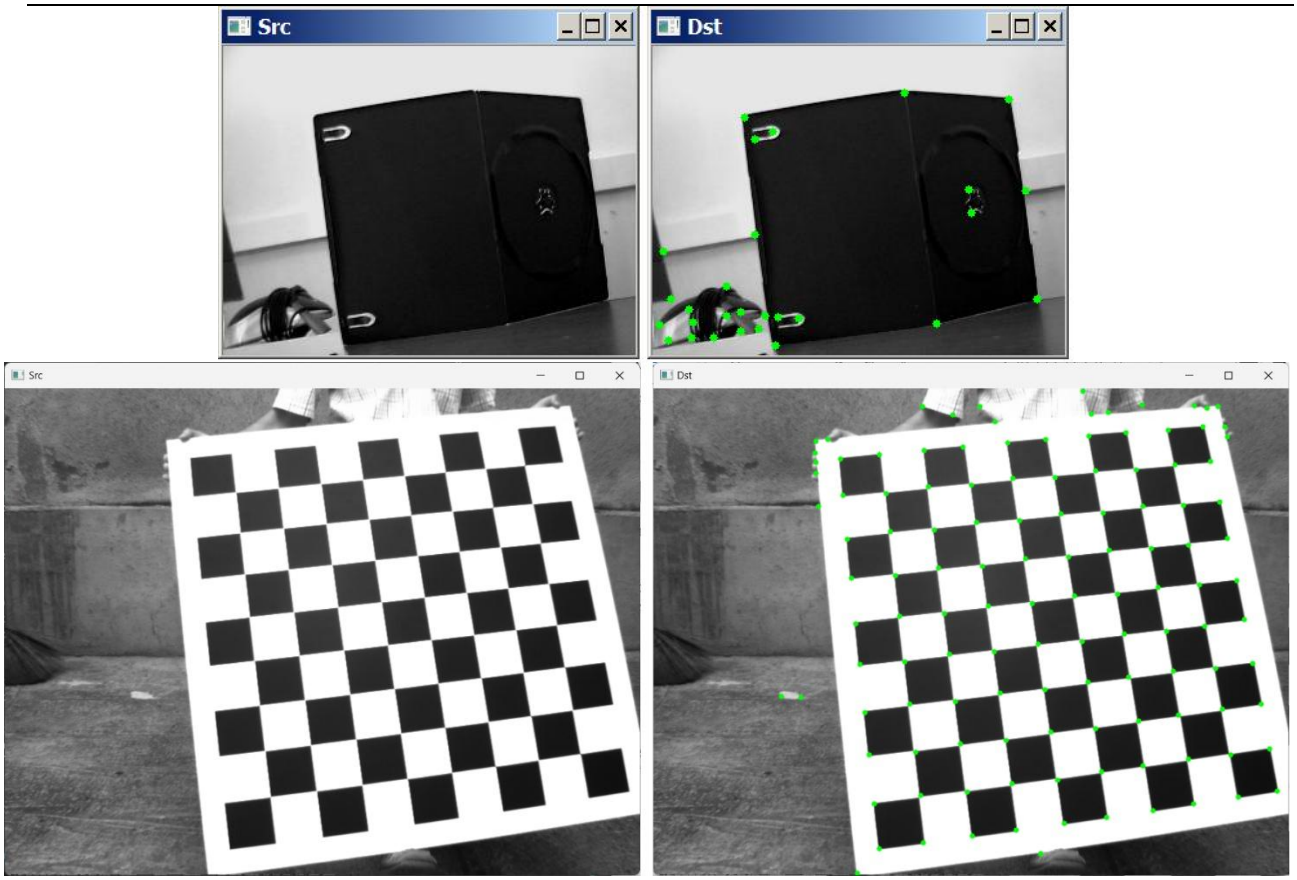


Fig. 4.3. Ilustrarea rezultatului detecției de colturi folosind funcția *goodFeaturesToTrack* pe diverse imagini (`useHarrisDetector = true`.).

4.3. Activități practice

1. Se va adaugă o funcție de procesare pentru detecția colțurilor prin apelul metodei *goodFeaturesToTrack*.

Observație: Pentru a putea detecta colțurile indiferent de tipul imaginii sursă (grayscale sau RGB24) și pentru a putea marca colțurile detectate în imaginea destinație (ex. cercuri de culoare verde) vă recomandăm să faceți următoarele:

- Să citiți imaginea sursă astfel: `Src_image = imread(fname, IMREAD_COLOR);`
- Să clonați sursa în destinație: `Dst_image = Src_image.clone();` la final, peste aceasta imagine veți desena colțurile detectate.
- Să convertiți imaginea sursă într-o imagine grayscale (`cvtColor`) și apoi să aplicați un filtru trece-jos gaussians (`GaussianBlur`) înainte de a apela metoda de detecție a colțurilor.
- Să vizualizați colțurile detectate în imaginea destinație prin parcurgerea vectorului `corners` și desenarea unor cercuri cu raza 3..5 și de culoare verde (`Scalar(0, 255, 0)`) în jurul punctelor de colț.

2. Se va vizualiza rezultatul metodelor de detecție a colturilor din *goodFeaturesToTrack* pentru diferite valori ale parametrilor de intrare: `maxCorners, qualityLevel, minDistance, blockSize, useHarrisDetector = true/false, k.`

3. Se va adapta funcția implementată la punctul 1 pentru a calcula coordonatele rafinate ale colțurilor detectate cu precizie de sub-pixel și se vor afișa într-un fișier text sau la linia de comandă. Metoda de

calcul a coordonatelor colțurilor cu precizie de sub-pixel este descrisă în tutorialul [6]. Pentru aceasta se inițializează parametrii necesari:

```
Size winSize = Size( 5, 5 );
Size zeroZone = Size( -1, -1 );
TermCriteria criteria = TermCriteria( TermCriteria::EPS + TermCriteria::COUNT, 40, 0.001 );
```

Se calculează locațiile rafinate ale colturilor astfel:

```
cornerSubPix(Src_image_gray, corners, winSize, zeroZone, criteria );
```

și se vor scrie într-un fișier text sau la linia de comandă coordonatele (corners[i].x, corners[i].y) cu precizie de 2 zecimale.

4. Se va integrarea metoda Harris (vezi funcția *cornerHarris_demo*) [7] ca o funcție noua de procesare în OpenCVApplication. Această funcție calculează funcția de răspuns **R**, îi normalizează valorile în intervalul (0 .. 255) și apoi afișează colțurile a căror valoare normalizata a răspunsului este peste un prag.
   Cum interpretăm rezultatul (vezi Fig. 4.2):
   - `dst = Mat::zeros( src.size(), CV_32FC1 )-` va conține funcția de răspuns R(x,y)
   - această valoare se normalizează în intervalul 0..255 și se pune într-o imagine grayscale (1 canal): `dst_norm_scaled`
   - Zonele netede din imagine ($R(x,y) \approx 0$) sunt mapate în nuanțe de gri
   - Punctele de muchie ($R(x,y) < 0$ sunt mapate la nuanțe închise (negru)
   - Punctele de colț ($R(x,y) > 0$ sunt mapate la nuanțe deschise (alb)

   Observație: se poate observa ca unele colțuri lipsesc ($R < thresh$) iar altele sunt încercuite de mai multe ori (puncte vecine care au funcția de răspuns peste prag). Pentru a elimina răspunsurile multiple pentru un colț puteți aplica „metoda de supresie a non-maximelor” [4].

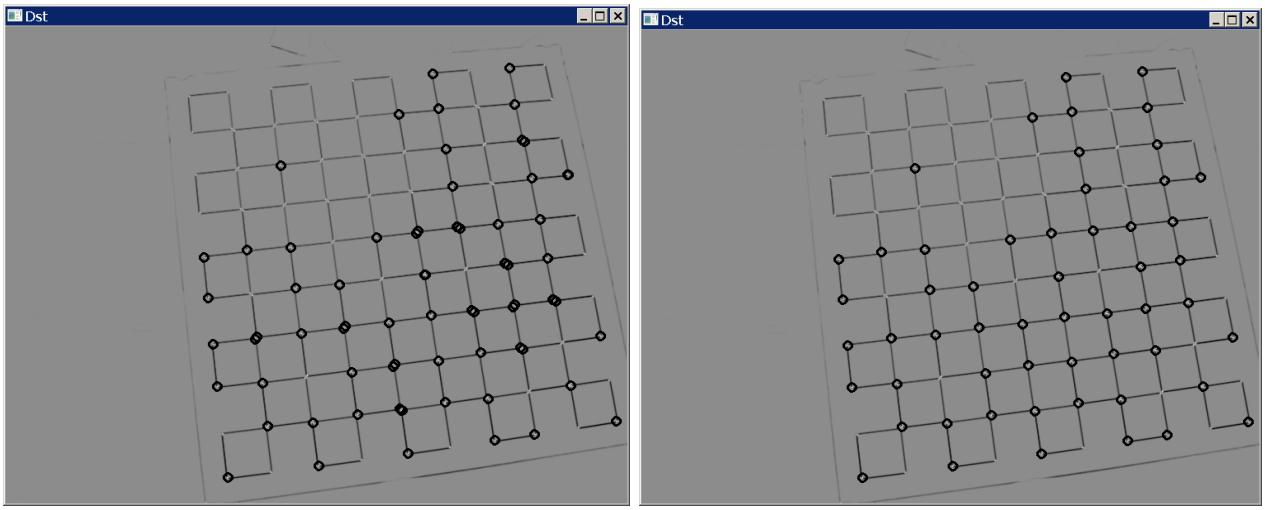


Fig. 4.4. Ilustrarea rezultatului implementării *cornerHarris_demo* (*thresh=200*): fără supresia non-maximelor (stânga) și cu supresia non-maximelor pe o vecinătate de dimensiune 11x11 (dreapta).

5. Adăugați o funcție de procesare care să realizeze detecția de colțuri pe secvențe video sau secvențe live captate de la webcam. Folosiți ca și șablon de procesare funcția `testVideoSequence()` din exemplele existente în proiectul *OpenCVApplication* și integrați în ea procesarea făcută la punctul 1 din mersul lucrării.

4.4. Bibliografie

[1] Metode de detecție a punctelor de tip colt, http://en.wikipedia.org/wiki/Corner_detection, citat 2024.
[2] S. Nedevschi, T. Marita, R. Danescu, F. Oniga, R. Brehar, I. Giosan, C. Vancea, R. Varga, Procesarea Imaginilor - Îndrumător de laborator, editia a 2-a, Editura U.T. Press, Cluj-Napoca, https://biblioteca.utcluj.ro/carti-online-cu-coperta.html, 2023.
[3] A. Koschan, M. Abidi, Digital Color Image Processing, Wiley & Sons, 2008, cap 6, pag 143 - 144.
[4] Interactiune Om-Calculator, Note de curs, Curs 4: http://users.utcluj.ro/~tmarita/HCI/C4.pdf .
[5] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.
[6] The OpenCV Tutorials, (\doc\opencv_tutorials.pdf), 6.7 Detecting corners location in subpixeles. https://docs.opencv.org/4.9.0/dd/d92/tutorial_corner_subpixels.html
[7] The OpenCV Tutorials, Release (\doc\opencv_tutorials.pdf), 6.2 Harris corner detector. https://docs.opencv.org/4.9.0/d4/d7d/tutorial_harris_detector.html
[8] The OpenCV Tutorials, Release (\doc\opencv_tutorials.pdf), 6.5 Shi-Tomasi corner detector, https://docs.opencv.org/4.9.0/d8/dd8/tutorial_good_features_to_track.html

5. Segmentarea obiectelor în mișcare prin modelarea și eliminarea fundalului ("Background Subtraction")

Scop: Scopul acestui laborator este de a experimenta metode de segmentare a obiectelor aflate în mișcare captate cu o cameră fixă. Scena fiind statică, pixelii scenei vor fi clasificați ca și fundal (background) în timp ce pixelii obiectelor în mișcare (cu prezență tranzitorie în scena) vor fi clasificați ca și obiect (foreground). Ca realizare practica se propune implementarea primelor doua metode de segmentare prezentate în materialul suport pentru curs al disciplinei: prin diferență simplă între imagini succesive și prin modelarea fundalului (background), simplă și selectivă.

5.1. Segmentarea pixelilor obiect (foreground)

Cerința fundamentală a acestei metode este de a segmenta pixelii care aparțin obiectelor (foreground) în mișcare, în timp ce fundalul/camera sunt fixe. Pentru aceasta se va construi un model (imagine) a fundalului (background) și pentru fiecare cadru (frame) de imagine *i* se va testa următoarea condiție la nivelul fiecărui pixel *(x,y)*:

$$diff = |frame_i(x,y) - backgnd_i(x,y)| > Th \quad (5.1)$$

Unde: *frame* este imaginea curenta *i*
backgnd este modelul fundalului la momentul curent
Th – este o valoare de prag aleasă arbitrar

Operația respectiva se poate aplica atât pe imagini color (multi-canal) sau grayscale (un singur canal). Pentru simplitate în lucrarea de față se va lucra cu imagini grayscale (un canal) iar ecuația de mai sus se poate scrie:

$$diff = |gray_i(x,y) - backgnd_i(x,y)| > Th \quad (5.2)$$

Unde: $gray_i$ este imaginea curentă grayscale obținută prin următoarea conversie:

```
cvtColor(frame, gray, COLOR_BGR2GRAY);
```

Aceste imagini se pot declara în OpenCv de tipul *Mat*:

```
Mat frame, gray, backgnd, diff;
```

iar diferența în modul dintre imaginea curenta *gray* și modelul *backgnd* se poate calcula folosind funcția OpenCV *absdiff* cu rezultatul plasat în matricea *diff*:

```
absdiff(gray, backgnd, diff);
```

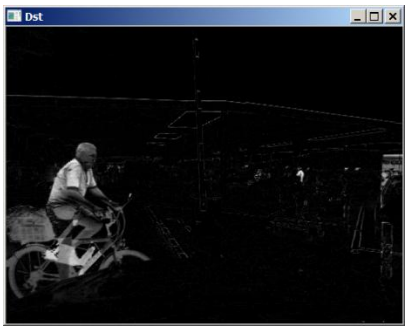


Fig. 5.1. Exemplu cu rezultatul imaginii diferență *diff*

Pixelii care satisfac condiția din ecuația (2) se pot reprezenta cu o culoare specifică (de ex. 255 – alb). Pentru aceasta trebuie parcursă imaginea diferență *diff* și testată condiția din ecuația (2) la nivel de pixel. Pentru o imagine cu 1 canal (grayscale) această condiție se poate testa prin următoarea secvență:

```
if (diff.at<uchar>(i,j) > Th)
       dst.at<uchar>(i,j) = 255;
```

Unde: *dst* este imaginea de ieșire pentru vizualizarea rezultatului.

Imaginea de ieșire trebuie inițializată la nivelul fiecărui cadru achiziționat! Dacă se dorește suprapunerea pixelilor obiect colorați cu 255 (white) peste imaginea originală (grayscale) atunci inițializarea ei se poate face astfel:

```
dst=gray.clone();
```

Dacă se dorește ca imaginea fundal peste care se afișează pixelii obiect sa fie o imagine goală (neagră) atunci inițializarea se poate face în felul următor:

```
dst = Mat::zeros( gray.size(), gray.type() );
```

a. b.

Fig. 5.2. Vizualizarea pixelilor marcați ca și obiect (alb): a – peste imaginea sursă grayscale; b- peste un fundal/imagine neagră.

5.2. Modelarea pixelilor de fundal (background)

Aceasta modelare se va face în funcție de algoritmul de Background Subtraction (BS) ales pentru implementare.

5.2.1. Metoda 1: diferența dintre cadre

În acest caz $backgnd_{i+1} = gray_i$ (cadrul grayscale curent, în momentul *i*, va fi model al fundalului în cadrul următor: momentul *i+1*). Aceasta se poate face în bucla de achiziție/prelucrare a imaginilor/cadrelor prin apelul funcției *gray.clone()* plasat după calculul diferenței absolute dintre cadrul curent și modelul curent al fundalului, ca și în secvența de mai jos:

```
absdiff(gray, backgnd, diff);
backgnd = gray.clone();
```

5.2.2. Metoda 2: modelarea fundalului prin mediere ponderată (running average)

În acest caz $backgnd_{i+1}$ (modelul fundalului în momentul $i+1$) se va calcula prin medierea ponderată a imaginii grayscale curente și modelul fundalului de la momentul prezent:

$$backgnd_{i+1}(x,y) = \alpha * gray_i(x,y) + (1-\alpha)backgnd_i(x,y) \quad (5.3)$$

Unde α este un coeficient de ponderare sub-unitar ($\alpha << 1-\alpha$, cu valori tipice între 0.01 .. 0.05).

Ecuația (5.3) se poate implementa prin apelul funcției OpenCV *addWeighted* plasat în bucla de achiziție/prelucrare a imaginilor/cadrelor după calculul diferenței absolute dintre cadrul curent și modelul curent al fundalului ca și în secvența de mai jos:

```
absdiff(gray, backgnd, diff);
addWeighted(gray, alpha, backgnd, 1.0-alpha, 0, backgnd);
```

5.2.3. Metoda 3: modelarea fundalului prin mediere ponderată selectivă (selective running average)

Diferența față de metoda 2 este că actualizarea modelului pentru pixelii de fundal se va face selectiv:

- dacă pixelul curent este clasificat ca și obiect, nu se schimbă modelul fundalului pentru pixelul respectiv
- dacă pixelul curent este clasificat ca și fundal atunci el contribuie la actualizarea modelului fundalului

```
// current pixel is forground (object) -> color in white
if (diff.at<uchar>(i,j) > Th)
	dst.at<uchar>(i,j) = 255; // current pixel is forground (object) -> color in
white
	// selective running average -> no change to the background pixel model value
	// backgnd.at<uchar>(i,j) = backgnd.at<uchar>(i,j);

else // current pixel is background -> update background model for the current pixel
	backgnd.at<uchar>(i,j) =
			alpha*gray.at<uchar>(i,j) + (1.0-alpha)*backgnd.at<uchar>(i,j);
```

5.3. Activități practice

1. Pentru implementarea metodelor se va crea o funcție de procesare nouă în *OpenCV Application*. Fișierul video se va citi de pe disk (vezi exemplul existent `testVideoSequence()` din proiectul OpenCV Application). La rulare avansarea în fișierul video se face cadru cu cadru prin apăsarea oricărei taste. Tasta ESC permite oprirea vizualizării.

2. Se vor face următoarele inițializări:

```
Mat frame, gray; //current frame: original and gray
Mat backgnd;   // background model
Mat diff;             //difference image: |frame_gray - bacgnd|
Mat dst;              //output image/frame
char c;
int frameNum = -1;    //current frame counter
const int method = 0;
// method =
```

```
	//                    1 - frame difference
	//                    2 - running average
	//                    3 - running average with selectivity
	const unsigned char Th = 15;
	const double alpha = 0.05;
```

3. Se va implementa metoda de segmentare a pixelilor de fundal (5.1) folosind cele trei variante expuse pentru modelarea fundalului (5.2.1 … 5.2.3).

Observație: Operațiile aferente segmentării nu se vor aplica asupra primului cadru din secvență, ci doar pentru următoarele. La primul cadru se recomandă inițializarea fundalului cu cadrul grayscale curent, conform șablonului de mai jos:

```
for(;;){
	cap >> frame; // achizitie frame nou
	if( frame.empty()  )
	 {
		printf("End of video file\n"));
		break;
	 }
	++frameNum;
	if (frameNum == 0)
		imshow( “sursa”, frame);     // daca este primul cadru se afiseaza doar sursa
	cvtColor(frame, gray, COLOR_BGR2GRAY);
	//Optional/recomandabil puteti aplica si un FTJ Gaussian
	GaussianBlur(gray, gray, Size(5, 5), 0.8, 0.8);

	//Se initializeaza matricea / imaginea destinatie pentru fiecare frame
	//	dst=gray.clone();
	// sau
	// preferabil cu o imagine neagra
	//	dst = Mat::zeros( gray.size(), gray.type() );
	const int channels_gray = gray.channels();
	//restrictionam utilizarea metodei doar pt. imagini grayscale cu un canal (8 bit / pixel)
	if (channels_gray > 1)
		return;
	if (frameNum > 0 ) // daca nu este primul cadru
	{
		//------ SABLON DE PRELUCRARI PT. METODELE BACKGROUND SUBTRACTION -------
		// Calcul imagine diferenta dintre cadrul current (gray) si fundal (backgnd)
		// Rezultatul se pune in matricea/imaginea diff
		// Se actualizeaza matricea corespunzatoare modelului fundalului (backgnd)
		// conform celor 3 metode:
		// met 1: backgnd = gray.clone();
		// met 2: addWeighted(gray, alpha, backgnd, 1.0-alpha, 0, backgnd);

		// Binarizarea matricii diferenta (pt. toate metodele):
		//  Se parcurge sistematic matricea diff
			//daca valoarea pt. pixelul current diff.at<uchar>(i,j) > Th
				// marcheaza  pixelul din imaginea destinatie ca obiect:
				dst.at<uchar>(i,j) = 255 // pentru toate metodele
			// altfel
				// actualizeaza model background (doar pt. met 3), pt. fiecare pixel
		//-------------------------------------------------------------------------

		// Afiseaza imaginea sursa si destinatie
		imshow( “sursa”, frame);     // show source
		imshow(“dest”, dst);         // show destination

		// Plasati aici codul pt. vizualizarea oricaror rezultate intermediare
		// Ex: afisarea intr-o fereastra noua a imaginii diff
	}
	else    // daca este primul cadru, modelul de fundal este chiar el
		backgnd = gray.clone();
```

```
        // Conditia de avansare/terminare in cilului for(;;) de procesare
        c = waitKey(0);  // press any key to advance between frames
        //for continous play use cvWaitKey( delay > 0)
        if (c == 27) {
                // press ESC to exit
                printf("ESC pressed - playback finished\n"));
                break;  //ESC pressed
                }
}
```

4. Se vor vizualiza în imaginea de ieșire (*dst*) atât rezultatele intermediare (imaginea *diff* – Fig. 1) cât și rezultatul final al segmentării (Fig. 2. a sau b) (matricea *dst*). Pentru teste se vor putea folosi filmele din fișierele *BS.avi*, *taxi.avi, walkcircle.avi, walkstraight.avi, campus.avi, laboratory.avi*). Puteți să le descărcați de aici: http://users.utcluj.ro/~tmarita/HCI/Media/Video/ și le copiați în directorul *Videos* din proiectul *OpenCVApplication*.

5. Pe rezultatul final al segmentării (Fig 2.b – pixelii de fundal colorați în negru) se pot aplica operații morfologice (pixelii de obiect în operațiile morfologice din OpenCV sunt considerați cei albi), pentru eliminarea de zgomote (pixeli singulari).
Un exemplu de cod care realizează 2 eroziuni succesive urmate de 2 dilatări succesive cu un element structural de tip cruce de dimensiune 3x3, cu elementul de referința central, este prezentat mai jos:

```
Mat element = getStructuringElement( MORPH_CROSS, Size( 3, 3 ) );
erode ( dst, dst, element, Point(-1,-1), 2 );
dilate( dst, dst, element, Point(-1,-1), 2 );
```

6. Pt. fiecare cadru/frame se va măsura timpul de procesare [ms] și se va afișa la linia de comanda (vezi Anexa 2)

5.4. Bibliografie

[1] http://docs.opencv.org/index.html
[2] The OpenCV Tutorials, Release (`doc\opencv_tutorials.pdf`), 2.2. How to scan images, lookup tables and time measurement with OpenCV.
http://docs.opencv.org/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html#howtoscanimagesopencv, `doc\opencv_tutorials.pdf`
[3] OpenCV_Tutorials (`doc\opencv_tutorials.pdf`), 3.2 Eroding and Dilating, 3.3 More Morphology Transformations:
http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html#morphology-1

5.5. Anexe

Măsurarea timpului de execuție se poate face astfel:

```
double t = (double)getTickCount(); // Get the current time [s]

// . . . insert here the processing functions / code

// Get the current time again and compute the time difference [s]
t = ((double)getTickCount() - t) / getTickFrequency();
// Print (in the console window) the processing time in [ms]
printf("%d - %.3f [ms]\n", frameNum, t*1000);
```

Metoda de acces la pixelii individuali dintr-o imagine în OpenCV folosită într-un exemplu care implementează negativul unei imagini. Pentru accesul cel mai rapid din punctul de vedere al timpului de parcurgere al imaginii consultați si celelalte exemple prezentate in [2].

```
Mat src, dst; // source and destination images
src = imread(opened_file_path , IMREAD _UNCHANGED);    // Read the file
/*-------------------------------------------------------------------------------
The second argument of imread specifies the format in what we want the image. This
may be:
IMREAD__LOAD_IMAGE_UNCHANGED (<0) loads the image as is
IMREAD_GRAYSCALE ( 0) loads the image as an intensity one
IMREAD_COLOR (>0) loads the image in the RGB format
-------------------------------------------------------------------------------*/
dst=src.clone(); // copy source image in destination
// accept only char type matrices
CV_Assert(src.depth() != sizeof(uchar));
const int channels = src.channels();
switch(channels)
	{
	case 1:  //  8bit image (1 channel image)
		{
		for( int i = 0; i < src.rows; ++i)
			for( int j = 0; j < src.cols; ++j )
				//performs image negative
				dst.at<uchar>(i,j) = 255 - src.at<uchar>(i,j);
	break;
		}
	case 3:  // 24 bit (3 channels image)
		{
		Mat_<Vec3b> _src=src;
		Mat_<Vec3b> _dst=dst;
		for( int i = 0; i < src.rows; ++i)
			for( int j = 0; j < src.cols; ++j )
			{
				_dst(i,j)[0] = 255 - _src(i,j)[0];
				_dst(i,j)[1] = 255 - _src(i,j)[1];
				_dst(i,j)[2] = 255 - _src(i,j)[2];
			}
		dst = _dst;
		break;
		}
	}
```

6. Estimarea fluxului optic și urmărirea punctelor de interes în secvențe de imagini

Scop: Scopul acestei lucrări este de a studia și integra metode de detecție a mișcării obiectelor din imagine succesive. Mișcarea poate fi detectata prin estimarea unor parametrii ai mișcării:

Câmp de mișcare := setul vectorilor (vitezelor) de mișcare ale punctelor din imagine (2D) induse de mișcarea relativă dintre scenă (obiecte ale scenei) și cameră

- nu este măsurabil direct din imagine, dar se poate estima prin urmărirea în imagini succesive a unor trăsături relevante (ca de exemplu colțuri)

Fluxul optic := mișcarea aparentă a șabloanelor de intensitate din imagine

- se poate măsura direct din imagine
- este o aproximare a câmpului de mișcare cu o rată de eroare mică în puncte cu gradient mare (daca direcția gradientului și direcția mișcării coincid)

6.1. Metode de măsurare a fluxului optic

Toate metodele au la bază ecuația constanței intensității imaginii (Image Brightness Constancy Equation) care folosește asumpția că intensitatea luminoasă a unui punct din scenă (chiar daca acest punct își schimbă poziția de la o imagine la alta) rămâne constantă:

$$(\nabla E)^T v + E_t = 0 \tag{6.1}$$

Unde:

$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{bmatrix} = \begin{bmatrix} E_x \\ E_y \end{bmatrix}$ - este gradientul imaginii în punctul curent studiat

$E_t = E(t) - E(t-1)$ - este derivata temporală a intensității imaginii în punctul curent studiat

E – este intensitatea imaginii în punctul curent studiat

$v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$ - este vectorul de deplasament al poziției punctului curent studiat (între cele două imagini succesive)

Majoritatea metodelor de estimare a fluxului optic se bazează pe algoritmi iterativi care încearcă să găsească pentru fiecare pixel din imagine un vector de deplasament care minimizează următoarea funcție reziduală:

$$e(\boldsymbol{v}) = e(v_x, v_x = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=v_y-w_y}^{y=v_y+w_y} (I(x,y) - J(x+vx, y+vy)) \tag{6.2}$$

unde *I(x,y)* – este un punct în imaginea *I* (t) iar *J(x+v_x , y+v_y)* este noua locație a aceluiași punct (pixel) în imaginea J de la momentul (t+Δt).

6.1.1. Calculul fluxului optic prin algoritmul Horn-Schunk (iterativ)

Algoritmul Horn-Schunk este o metoda iterativă de estimare a fluxului opticcare folosește ca parametrii de intrare: n_0 – numărul maxim de iterații și λ – ponderea de corecție.

1. Se face o primă parcurgere a imaginii. Pentru fiecare pixel p se calculează $E_x(p)$, $E_y(p)$, $E_t(p)$ și se inițializează v_x și v_y cu 0 (este necesară alocarea unor matrice de dimensiunea imaginii pentru stocarea acestor valori – câte o matrice pentru fiecare variabilă).
2. Se aleg valorile pentru λ (ex. $\lambda = 10$) și n_0 ($n_0 = 8$).
3. Pentru $n = 1 .. n_0$:
   Se parcurg matricele v_x și v_y. Pentru fiecare pixel *p* se calculează valorile medii ale v_x și v_y (din vecinii de pe direcțiile cardinale:

$$\bar{v}_x = \frac{1}{4}\left[v_x(i-1,j) + v_x(i+1,j) + v_x(i,j-1) + v_x(i,j+1)\right] \tag{6.3}$$

$$\bar{v}_y = \frac{1}{4}\left[v_y(i-1,j) + v_y(i+1,j) + v_y(i,j-1) + v_y(i,j+1)\right] \tag{6.4}$$

   Se calculează coeficientul de corecție α:

$$\alpha = \lambda \frac{E_x \bar{v}_x + E_y \bar{v}_y + E_t}{1 + \lambda \cdot (E_x^2 + E_y^2)} \tag{6.5}$$

   Se actualizează (corectează) valorile v_x si v_y:

$$v_x = \bar{v}_x - \alpha \cdot E_x \tag{6.6}$$

$$v_y = \bar{v}_y - \alpha \cdot E_y \tag{6.7}$$

6.1.2. Calculul fluxului optic prin algoritmul Lukas-Kanade (iterativ) [1]

Se inițializează vectorii de flux optic cu 0:

$$\bar{\boldsymbol{v}}^{\boldsymbol{0}} = [0 \quad 0]^{\boldsymbol{T}} \tag{6.8}$$

Ca și condiție de terminare a iterațiilor se poate limita numărul de pași (ex. K=20) sau se poate impune condiția ca norma factorului de corecție $\| \eta^k \| < th$ (ex. $th = 0.03$):

For *k=1* **to** *K* (step 1) (sau $\| \eta^k \| < th$)

Se calculează imaginea diferență:

$$\delta I_k(x,y) = I^L(x,y) - J^L\left(x + v_x^{k-1}, y + v_y^{k-1}\right) \tag{6.9}$$

Se calculează vectorul de eroare al imaginii diferență:

$$\overline{\boldsymbol{b}_k} = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{y=p_y+w_y} \begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix} \tag{6.10}$$

Se calculează corecția pentru fluxul optic la pasul k:

$$\bar{\boldsymbol{\eta}}^{\boldsymbol{k}} = \boldsymbol{G}^{-1}\overline{\boldsymbol{b}_k} \tag{6.11}$$

Se actualizează fluxul optic la pasul k:

$$\bar{\boldsymbol{v}}^{\boldsymbol{k}} = \bar{\boldsymbol{v}}^{\boldsymbol{k-1}}\bar{\boldsymbol{\eta}}^{\boldsymbol{k}} \tag{6.12}$$

End

Vectorii de flux final vor fi $\boldsymbol{v}^K$.

6.1.3. Urmărirea de trăsături în imagini succesive

Estimarea câmpului de mișcare se poate face doar pentru trăsături discrete prin urmărirea unor astfel de trăsături relevante (ca de exemplu colțuri) în imagini succesive. O astfel de abordare este metoda Lukas-Kanade în varianta piramidală [1].

Acest algoritm aplică metoda iterativă de calcul a fluxului optic Lukas-Kanade pe o piramidă de imagini care conține imaginea la rezoluția originală (nivel L= 0) și sub-imagini obținute din imaginea originala prin decimare (nivel L = 1, 2, ... 4 - de obicei se considera L=3). Vectorii de flux optic se estimează doar pe un set de trăsături relevante (cu putere de discriminare mare) care au cel putini doua direcții distincte ale gradientului, cum sunt colțurile.

Pentru astfel de trăsături fluxul optic și câmpul de mișcare coincid, rezultatul obținut la ieșire fiind câmpul de mișcare al acestor trăsături (se mai numește și flux optic „rar" – sparse optical flow).

Metoda este descrisă sumar în cursul 6 (http://users.utcluj.ro/~tmarita/HCI/C6.pdf) și prezentata in detaliu in [1] – se va studia ca tema de casa!

6.2. Detalii de implementare

6.2.1. Șablonul de procesări ale secvențelor de imagini (bitmaps)

Implementarea se va face plecând de la șablonul `testOpenImagesFld()` din *OpenCVApplication.cpp*, în care este exemplificată parcurgerea unei secvențe de imagini conținute într-un folder selectat prin intermediul unui controlul de tip dialog box. De asemenea se va folosi și șablonul de prelucrare a unei secvențe de imagini introdusă în laboratorul 5: se procesează imaginea curentă și imaginea precedentă. Procesările pentru estimarea vectorilor de mișcare se vor face pe imagini grayscale

```
Mat crnt;     // current frame red as grayscale (crnt)
Mat prev;    // previous frame (grayscale)
Mat flow;    // flow - matrix containing the optical flow vectors/pixel
char folderName[MAX_PATH];
char fname[MAX_PATH];
if (openFolderDlg(folderName) == 0)
      return;
FileGetter fg(folderName, "bmp");
```

Șablonul de procesare pentru bucla principală (în care se citesc succesiv imaginile din secvența de fișiere bitmap) ar trebui să arate în felul următor:

```
int frameNum = -1;  //current frame counter
while (fg.getNextAbsFile(fname))// citeste in fname numele caii complete
                                // la cate un fisier bitmap din secventa
{
      crnt = imread(fname, IMREAD_GRAYSCALE);
      GaussianBlur(crnt, crnt, Size(5, 5), 0.8, 0.8);
      ++frameNum;
      if (frameNum > 0 ) // not the first frame
      {
      . . . .
      // functii de procesare (calcul flux optic) si afisare
      . . .
      }
      // store crntent frame as previos for the next cycle
      prev = crnt.clone();
```

```
            c = cvWaitKey(0);  // press any key to advance between frames
            //for continous play use cvWaitKey( delay > 0)
            if (c == 27) {
                  // press ESC to exit
                  printf("ESC pressed - playback finished\n\n");
                  break;  //ESC pressed
            }
      }
```

Ca preprocesare este utilă aplicarea unui filtru gaussian pe imaginea sursă convertită în grayscale pentru eliminarea eventualelor zgomote (ex. un gaussian de 5x5 cu sigma=0.8).

6.2.2. Calculul fluxului optic folosind metoda Horn-Schunk

Se va implementa algoritmul de calcul al fluxului optic Horn-Schunk sub forma unei funcții care se va apela în șablonul de procesare prezentat mai sus:

```
void calcOpticalFlowHS(const Mat& prev, const Mat& crnt, float lambda, int n0, Mat&
flow)
{
      Mat vx = Mat::zeros(crnt.size(), CV_32FC1); // matricea comp. x a fluxului optic
      Mat vy = Mat::zeros(crnt.size(), CV_32FC1); // matricea comp. y a fluxului optic
      Mat Et = Mat::zeros(crnt.size(), CV_32FC1); // derivatele temporale
      Mat Ex, Ey; // Matricele derivatelor spatiale (gradient)

      // Calcul componenta orizontala a gradientului
      Sobel(crnt, Ex, CV_32F, 1, 0);
      // Calcul componenta verticala a gradientului
      Sobel(crnt, Ey, CV_32F, 0, 1);
      // Calcul derivata temporala
      Mat prev_float, crnt_float; // matricile crnt si prev convertite in float
      prev.convertTo(prev_float, CV_32FC1);
      crnt.convertTo(crnt_float, CV_32FC1);
      Et = crnt_float - prev_float;

      // Insercati codul aferent algoritmului Horn-Schunk
      // . . .

      // Compune comp. vx si vy ale fluxului intr-o matrice cu elemente de tip Point2f
      flow = convert2flow(vx, vy);

      // Vizualizare rezultate intermediare:
      // gradient,derivata temporala si comp. vectorilor de miscare sub forma unor
      // imagini grayscale obtinute din matricile de tip float prin normalizare
      Mat Ex_gray, Ey_gray, Et_gray, vx_gray, vy_gray;
      normalize(Ex, Ex_gray, 0, 255, NORM_MINMAX, CV_8UC1, Mat());
      normalize(Ey, Ey_gray, 0, 255, NORM_MINMAX, CV_8UC1, Mat());
      normalize(Et, Et_gray, 0, 255, NORM_MINMAX, CV_8UC1, Mat());
      normalize(vx, vx_gray, 0, 255, NORM_MINMAX, CV_8UC1, Mat());
      normalize(vy, vy_gray, 0, 255, NORM_MINMAX, CV_8UC1, Mat());
      imshow("Ex", Ex_gray);
      imshow("Ey", Ey_gray);
      imshow("Et", Et_gray);
      imshow("vx", vx_gray);
      imshow("vy", vy_gray);
}
```

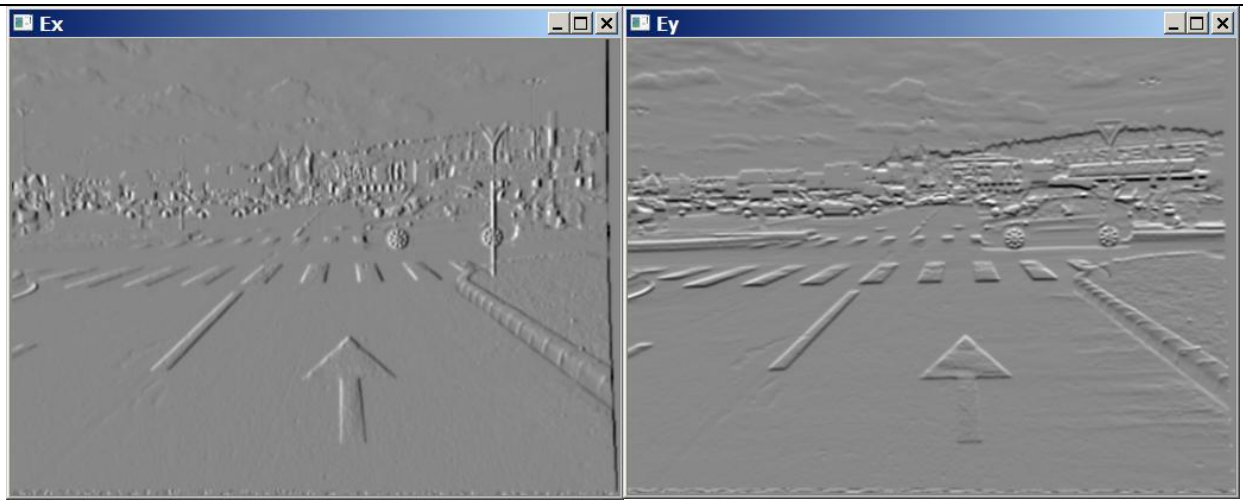


Fig. 6.1 Matricele derivatelor parțiale normalizate la imagini grayscale (gri – derivata nulă; negru - derivata negativă; alb derivata pozitivă)

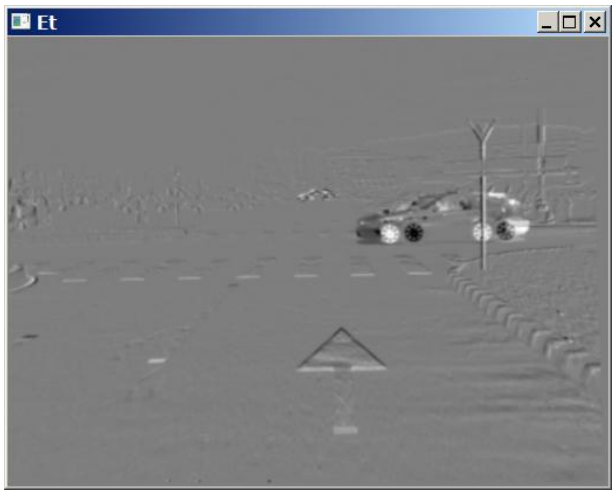


Fig. 6.2. Matricea derivatei temporale normalizata la grayscale (gri – derivata nulă; negru - derivata negativă; alb derivata pozitivă)

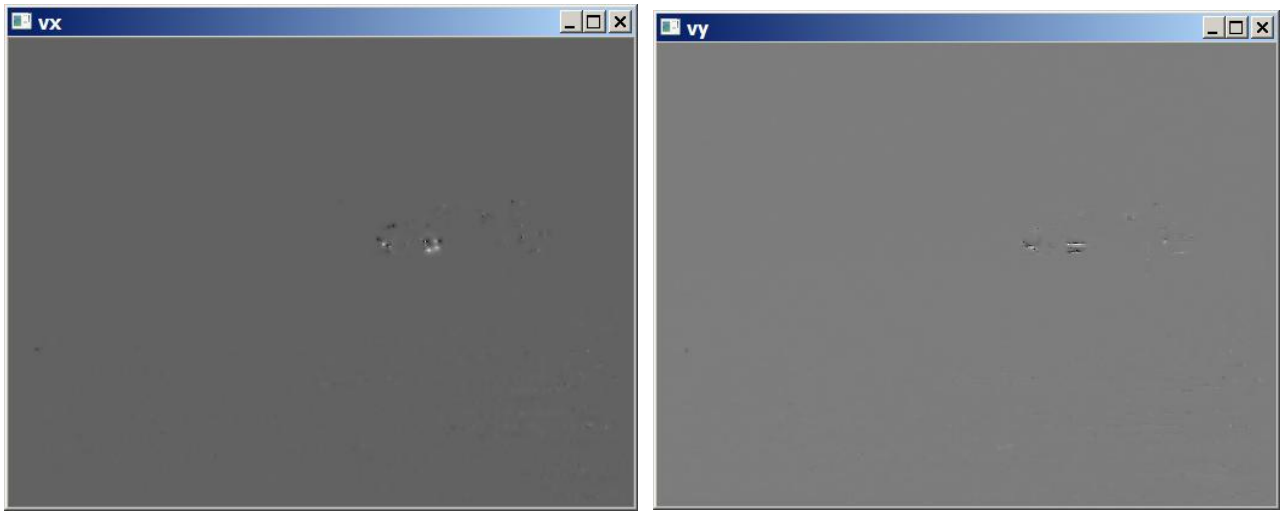


Fig. 6.3. Matricele componentelor orizontale și verticale ale vectorilor de flux optic normalizate la imagini grayscale.

Inserați apelurile de mai jos ale funcției Horn-Schunk de calcul al fluxului optic și a funcției de afișare a fluxului optic în șablonul de procesare al secvenței de imagini, adăugând și codul aferent calculului timpului de procesare:

```
// Horn-Shunk
double t = (double)getTickCount();
//calcOpticalFlowHS(prev, crnt, 0, 0.1, TermCriteria(TermCriteria::MAX_ITER, 16, 0),
flow);
calcOpticalFlowHS(prev, crnt, lambda, n0, flow);
// Stop the proccesing time measure
t = ((double)getTickCount() - t) / getTickFrequency();
printf("%d - %.3f [ms]\n", frameNum, t * 1000);
showFlow(WIN_DST, prev, flow, 1, 1, true, true, false);
```

6.2.3. Funcția de afișarea vectorilor de flux optic

Afișarea se poate realiza prin apelul funcției *showFlow* care desenează vectorii de flux optic, sub forma unor segmente de dreaptă. Funcția permite operația de *zoom* pe imaginea de ieșire (factor de scalare *mult*), filtrarea vectorilor pe baza unui prag *minVel* aplicat modulului/lungimii lor, afișarea selectivă a vectorilor și/sau a originii acestora. Funcția este integrată în modulul *Functions*: *Functions.h* / *Functions.cpp* si *NU mai este nevoie sa o copiați in funcția de procesare*.

Funcția *showFlow* afișează automat rezultatul în fereastra specificata ca și prim parametru (`const string& name`). Deci NU mai este nevoie sa adăugați în bucla de procesare a un apel explicit pentru afișarea ferestrei destinație (de ex. `imshow(WIN_DST, dst)`).
Exemplu apel:

```
showFlow (“Dst”, prev, flow, 1, 1.5, true, true, false);
```

Observație importantă legată de accesul la pixeli în structura Mat: accesul la pixeli sau date structurate în forma matriceală (ca de ex. matricea *flow*) se face prin perechi de puncte de tip (y,x) unde y este linia si x coloana: `Point2f f = flow.at<Point2f>(y, x);`

Fig. 6.4. Afișarea rezultatelor estimării fluxului optic pentru metoda Horn-Schunk.

6.2.4. Urmărirea trăsăturilor prin metoda Lukas-Kanade piramidală

Aceasta metodă necesită detecția prealabilă a unor trăsături de interes (colțuri) - vezi lucrarea L5, metoda `goodFeaturesToTrack` care furnizează un set de punte (colțuri) din imaginea precedentă `prev_pts` care vor fi folosite ca și intrări la metoda de urmărire a trăsăturilor Lukas-Kanade (LK) piramidală (`calcOpticalFlowPyrLK`). Metoda furnizează la ieșire setul de puncte/colțuri corespondente din imaginea curentă `crnt_pts`, un vector `status` care indică dacă s-a găsit potrivirea pentru fiecare punct și un vector de eroare `error` :

```
// Apply corner detection
	goodFeaturesToTrack(prev,prev_pts, ... )
	calcOpticalFlowPyrLK( prev, crnt, prev_pts, crnt_pts, status, error,
```

```
                                                winSize, maxLevel, criteria );
```

Inițializarea parametrilor pentru detectorul de colțuri (`goodFeaturesToTrack`) se va face ca și în lucrarea L4. Secvența de inițializare se va adăuga undeva la începutul funcției de procesare. Pentru metoda de calcul a fluxului optic LK piramidală sunt necesare inițializări asemănătoarea cu cele din exemplul de mai jos (se vor adăuga tot la începutul funcției de procesare) :

```
// parameters for calcOpticalFlowPyrLK
vector<Point2f> prev_pts; // vector of 2D points with previous image features
vector<Point2f> crnt_pts; // vector of 2D points with current image (matched) features
vector<uchar> status;     // output status vector: 1 if the flow for the corresponding
feature was found. 0 otherwise
vector<float> error;      // output vector of errors; each element of the vector is set
to an error for the corresponding feature
Size winSize=Size(21,21); // size of the search window at each pyramid level - deafult
(21,21)
int maxLevel=3; // maximum pyramid level number - deafult 3
//parameter, specifying the termination criteria of the iterative search algorithm
// (after the specified maximum number of iterations criteria.maxCount or when the
search window moves by less than criteria.epsilon
// deafult 30, 0.01
TermCriteria criteria=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 20, 0.03);
int flags=0;
double minEigThreshold=1e-4;
```

Afișarea rezultatului se poate face prin apelul funcției `showFlowSparse` (vezi Anexele). Funcția este integrată în modulul *Functions*: *Functions.h* / *Functions.cpp* si *NU mai este nevoie sa o copiați in functia de procesare*:

```
showFlowSparse ("Dst", prev, prev_pts, crnt_pts, status, error, 2, true, true, true);
```

Funcția *showFlowSparse* afișează automat rezultatul în fereastra cu numele specificat ca și prim parametru (`const string& name`) și NU mai este nevoie sa adăugați în bucla de procesarea un apel explicit pentru afișarea ferestrei destinație (de ex. `imshow("Dst", dst)`).

Funcția *showFlowSparse* permite afișarea vectorilor da mișcare a trusturilor (colțurilor) detectate în imagini. Funcția permite operația de *zoom* pe imaginea de ieșire (factor de scalare *mult*), filtrarea vectorilor pe baza valorilor din vectorul *status*, afișarea selectivă a vectorilor și/sau a originilor acestora (cerc roșu de raza 4) si a terminaților (cerc albastru de raza 2) – Fig. 6.5

Fig. 6.5. Afișarea rezultatelor estimării câmpului de mișcare obținut prin metoda LK piramidală pentru un scenariu cu un vehicul care traversează o intersecție de la dreapta spre stânga.

6.3. Activități practice

1. Pentru test se vor folosi secvențe de bitmaps (*polus.zip*, *sigma.zip*): pe care le puteți descarcă din foderul https://users.utcluj.ro/~tmarita/HCI/Media/Video/ sau de pe Team-ul disciplinei.
2. Se va implementa algoritmul Horn-Schunk de calcul al fluxului optic și se va integra in șablonul de procesare furnizat.
3. Se va integra metoda de calcul a câmpului de mișcare LK piramidala). Se vor afișa grafic și valorile erorilor (valoarea *error[i]* pentru colțurile la care s-a detectat corespondentul *status[i]==1*. Această eroare se poate reprezenta vizual desenând cercul de origine (roșu) al vectorilor de mișcare cu o raza variabila proporționala cu eroarea (este necesara găsirea unui factor de scalare adecvat). Modificarea trebuie făcută în codul funcției `showFlowSparse()` la apelul funcție `circle` de desenare a originii vectorului.

6.4. Bibliografie

[1] Pyramidal Implementation of the Lucas Kanade Feature Tracker - Description of the algorithm, Jean-Yves Bouguet, Intel Corporation Microprocessor Research Labs, jean-yves.bouguet@intel.com, http://robots.stanford.edu/cs223b04/algo_tracking.pdf
[2] Simon Baker, Daniel Scharstein, J.P. Lewis, Stefan Roth, Michael J. Black, Richard Szeliski, A Database and Evaluation Methodology for Optical Flow, http://vision.middlebury.edu/flow/.

6.5. Anexe

Deoarece implementarea propisă metodei Horn-Schunk furnizează fluxul optic în forma a 2 matrice de vectori `velx, vely` se furnizează o funcție (integrată în modulul *Functions*: *Functions.h* / *Functions.cpp*) de conversie a acestora într-un format matriceal compatibil cu rezultatul furnizat de alte metode de calcul a fluxului optic:

```
Mat convert2flow(const Mat& velx, const Mat& vely)
// converts the optical flow vectors velx and vely in a single matrix of flow
// in wich each element encodes the 2 components of the optical flow
{
    Mat flow(velx.size(), CV_32FC2);
    for(int y = 0 ; y < flow.rows; ++y)
        for(int x = 0 ; x < flow.cols; ++x)
            flow.at<Point2f>(y, x) = Point2f(velx.at<float>(y, x), vely.at<float>(y,
x));
    return flow;
}
```

Definiția funcției `showFlow` pentru afișarea (vectorilor) fluxului optic compatibilă cu metodele Legacy implementate în versiuni mai vechi ale OpeCV:

```
/*-----------------------------------------------------------
Function used to display to display the optical flow vectors (LEGACY methods) filtered
out by a length (modulus) threshold
Input:
      name - destination (output) window name
      gray -  background image to be displayed (usually the prev image)
      flow - optical flow as a matrix of (x,y)
      mult - scaling factor of the displayed image/window and of the optical flow
vectors
```

```
       minVel - threshold value (for modulus) for filtering out the displayed vectors
Call example:
       showFlow ("Dst", prev, flow, 1, 4, true, true, false);
-----------------------------------------------------------------*/
void showFlow (const string& name, const Mat& gray, const Mat& flow, int mult, float
minVel,
                            bool showImages , bool showVectors, bool showCircles)
{
    if (showImages)
    {
        Mat tmp, cflow;
        resize(gray, tmp, gray.size() * mult, 0, 0, INTER_NEAREST);
        cvtColor(tmp, cflow, CV_GRAY2BGR);
// gain factor for the flow vectors display (usefull if the vectors are very small
m2>1)
        const float m2 = 1.0f;

        for(int y = 0; y < flow.rows; ++y)
            for(int x = 0; x < flow.cols; ++x)
            {
                Point2f f = flow.at<Point2f>(y, x);
                if (f.x * f.x + f.y * f.y > minVel * minVel)
                {
                    Point p1 = Point(x, y) * mult;
                    Point p2 = Point(cvRound((x + f.x*m2) * mult),
                                     cvRound((y + f.y*m2) * mult));
                          if (showVectors) // display flow vectors as green lines
                                line(cflow, p1, p2, CV_RGB(0, 255, 0));
                                if (showCircles) // mark ve origins by a red circle
                                circle(cflow, Point(x, y)*mult, 2, CV_RGB(255, 0, 0));
                    }
             }

        imshow(name, cflow);
    }
}
```

Definiția funcției `showFlowSparse` pentru afișarea (vectorilor) fluxului pentru metoda LK piramidală:

```
/*-----------------------------------------------------------------
Function used to display to display the SPARSE optical dense vectors filtered out by
their status (1 or 0)
Input:
       name - destination (output) window name
       gray -  background image to be displayed (usually the prev image)
       vector<Point2f> prev_pts
       vector<Point2f> crnt_pts
       vector<uchar> status;
       vector<float> error;
       mult - scaling factor of the displayed image/window and of the optical flow
vectors
Call example:
       showFlowSparse ("Dst", prev, prev_pts, crnt_pts, status, error, 2, true, true,
true);
-----------------------------------------------------------------*/
void showFlowSparse (const string& name, const Mat& gray, const vector<Point2f>&
prev_pts, const vector<Point2f>& crnt_pts, const vector<uchar>& status, const
vector<float>& error, int mult, bool showImages, bool showVectors, bool showCircles)
{
```

```
    if (showImages)
    {
       Mat tmp, cflow;
       resize(gray, tmp, gray.size() * mult, 0, 0, INTER_NEAREST);
       cvtColor(tmp, cflow, CV_GRAY2BGR);
      for(int i = 0; i < prev_pts.size(); ++i)
      {
            if (showCircles) // mark flow vectors' origins by a red ircle
                  circle(cflow, prev_pts[i] * mult, 4, CV_RGB(255, 0, 0));
            if ( status[i] ) //flow for crntent point i exists
            {
                  Point2f p1 = prev_pts[i] * mult;
                  //Point2f p2 = crnt_pts[i] * mult;
                  Point2f p2 = Point(cvRound( crnt_pts[i].x * mult),
                                    cvRound( crnt_pts[i].y * mult));
                        if (showVectors) // display flow vectors as green segments
                              line(cflow, p1, p2, CV_RGB(0, 255, 0));
                        if (showCircles) // mark flow vectors' end by a blue circle
                              circle(cflow, crnt_pts[i]*mult, 2, CV_RGB(0, 0, 255));
            }
      }
      imshow(name, cflow);
    }
}
```

7. Analiza mișcării pe baza fluxului optic dens

Scop: Scopul acestei lucrări este de a integra o metoda de detecție a fluxului optic dens și de analiză a mișcării relative dintre camera și scena pe baza vectorilor de flux optic. Pentru aceasta se va folosi metoda propusă de Gunnar Farneback [1] și se va vizualiza fluxul optic dens al vectorilor de mișcare folosind codificarea Middleburry [2]. Pentru vectorii de mișcare se va construi histograma direcțiilor lor și se va face analiza formei acestei histograme, inferând-se mișcarea relativă dintre cameră și scenă sau dintre camera și obiectele din scenă (când camera este fixă).

7.1. Estimarea fluxului optic dens

Pentru estimarea fluxului optic dens se va integra metoda propusă de Gunnar Farneback [1], implementată în OpenCV prin funcția: `calcOpticalFlowFarneback`. Apelul funcției se realizează în felul următor:

```
calcOpticalFlowFarneback ( prev, crnt, flow, pyr_scale, levels, winSize, iterations,
                                                        poly_n, poly_sigma, flags);
```

Unde:

Mat *prev, crnt* – reprezintă imaginea precedentă, respectiv curentă (grayscale, 8 biti /pixel)
Mat *flow* – este o matrice de aceeași dimensiune cu imaginile dar de tip CV_32FC2 – conține vectorii de mișcare
pyr_scale – este factorul de scalare (0...1) folosit la construcția piramidei de imagini (ex 0.5 – înjumătățire a rezoluției între 2 nivele succesive din piramidă)
levels – este numărul de nivele din piramidă (*levels* = 1 – nu se folosesc piramide de imagini; valoarea recomandată este 3).
winsize – este dimensiunea ferestrei de mediere: valori ridicate măresc robustețea la zgomot și permit detecția vectorilor de mișcare cu amplitudine mai mare dar câmpul de mișcare rezultat va avea un aspect netezit (blurred)
iterations – este numărul de iterații ale algoritmului pentru fiecare nivel al piramidei (o valoare ridicată va duce la creșterea semnificativa a timpului de procesare!)
poly_n – este dimensiunea vecinătății în care se caută expansiunea polinomiala pentru fiecare pixel (pentru valori mari imaginea va fi aproximată cu suprafețe mai netede → algoritm de estimare mai robust și un câmp de mișcare mai netezit (blurred)); valori tipice = 5 ... 7.
poly_sigma – este deviația standard a filtrului gaussian-ului cu care se filtrează derivatele imagini folosite ca și baza a expansiunii polinomiale (valori tipice: *poly_n=5 / poly_sigma=1.1* sau *poly_n=7 / poly_sigma=1.5*
flags – este o constanta care poate lua una dintre următoarele valori: OPTFLOW_USE_INITIAL_FLOW sau OPTFLOW_FARNEBACK_GAUSSIAN. Dacă se alege prima constantă se folosește valoarea din *flow* ca aproximare inițiala a fluxului iar dacă se setează cel de-a doua constantă se folosește un Gaussian de dimensiune *winsize x winsize* în locul unui box-filter pentru estimarea fluxului optic (folosind gaussianul se obțin rezultate mai precise dar scade viteza de procesare).

Exemple de apel:

```
winSize=11;
calcOpticalFlowFarneback( prev, crnt, flow, 0.5, 3, winSize, 10, 6, 1.5, 0);
```

sau

```
winSize=15;
calcOpticalFlowFarneback( prev, crnt, flow, 0.5, 3, winSize, 10, 7, 1.5,
                    OPTFLOW_FARNEBACK_GAUSSIAN); //slower but more accurate
```

Funcția *calcOpticalFlowFarneback* estimează fluxul optic pentru fiecare pixel din imaginea precedenta (*prev*):

$$prev(y,x) = crnt(y + flow(y,x)[1], x + florw(y,x)[0]) \quad (7.1)$$

În consecință vectorii din *flow* au orientarea dinspre pixelul din imaginea curentă înspre corespondentul lui din imaginea precedentă. Dacă se dorește reprezentarea lor în sens invers trebuie considerată valoarea – *flow(y,x)* !!!

Șablonul de procesare pentru bucla principală (în care se citesc succesiv imaginile din secvența de fișiere bitmap) ar trebui să arate așa (identic ca și la lucrarea 6):

```
Mat crnt;    // current frame red as grayscale (crnt)
Mat prev;    // previous frame (grayscale)
Mat flow;    // flow - matrix containing the optical flow vectors/pixel

char folderName[MAX_PATH];
char fname[MAX_PATH];
if (openFolderDlg(folderName) == 0)
      return;
FileGetter fg(folderName, "bmp");


int frameNum = -1;  //current frame counter

while (fg.getNextAbsFile(fname)) // citeste in fname numele caii complete
                                 // la cate un fisier bitmap din secventa
{
      crnt = imread(fname, CV_LOAD_IMAGE_GRAYSCALE);
      GaussianBlur(crnt, crnt, Size(5, 5), 0.8, 0.8);

      ++frameNum;

      if (frameNum > 0 ) // not the first frame
      {
      . . . .
      // functii de procesare (calcul flux optic) si afisare
      // calcul histogrrama directii si afisare
      . . .
      }

      // store crntent frame as previos for the next cycle
      prev = crnt.clone();

      c = cvWaitKey(0);  // press any key to advance between frames
      //for continous play use cvWaitKey( delay > 0)
      if (c == 27) {
            // press ESC to exit
            printf("ESC pressed - playback finished\n\n");
            break;  //ESC pressed
      }
}
```

7.2. Afișarea hărții dense a fluxului optic folosind codificarea de culoare Middleburry

Pentru afișarea hărții dense a fluxului optic folosind codificarea de culoare Middleburry [2] aveți nevoie de modulul *colorcode* (inclus deja in proiect prin fișierele: *colorcode.h* si *colorcode.cpp*).

Pentru afișarea hărții dense a fluxului optic în fereastra destinație puteți folosi funcția `showFlowDense` (vezi ANEXA) integrată în modulul cu funcții ajutătoare (*Functions h, Functions.cpp*) din proiect. Apelul funcției `showFlowDense` se va face după apelul funcției de calcul a fluxului optic!

Funcția *showFlowDense* afișează automat rezultatul în fereastra specificată ca și prim parametru (`const string& name`). Nu este nevoie sa adăugați în bucla de procesarea un apel explicit pentru afișarea ferestrei destinație (de ex. `imshow("dst", dst)`).

Important: Funcția *showFlowDense* apelează funcția *computeColor* care folosește un LookUp Table (LUT) in care sunt codificate culorile de afișat și care trebuie inițializat. Inițializarea trebuie făcută doar o singura data în aplicație prin apelul `makeColorwheel()`. De asemenea adăugați apelul funcției `make_HSI2RGB_LUT` necesară pentru afișarea histogramei direcțiilor de mișcare în codul de culori folosit. Inserați cele doua apeluri la începutul funcției de procesare aferente laboratorului curent.

```
makeColorwheel(); // initaializes the colorwhel for the colorcode module
make_HSI2RGB_LUT();
```

Prin aceasta codificare fluxul optic dens în fiecare punct este reprezentat printr-un pixel a cărui culoare codifică direcția și amplitudinea vectorului de flux optic (similar reprezentării culorilor în modelul HSV, valoarea Hue (culoarea) codificând direcția vectorului de fux optic iar saturația codificând amplitudinea acestuia (fig. 7.1).

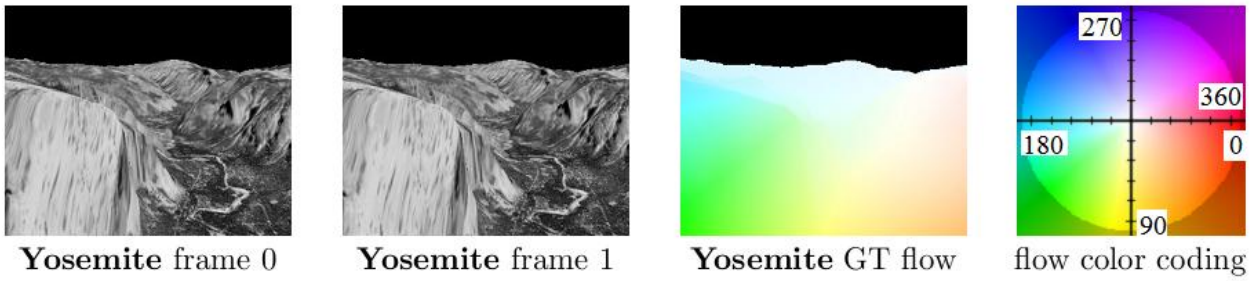
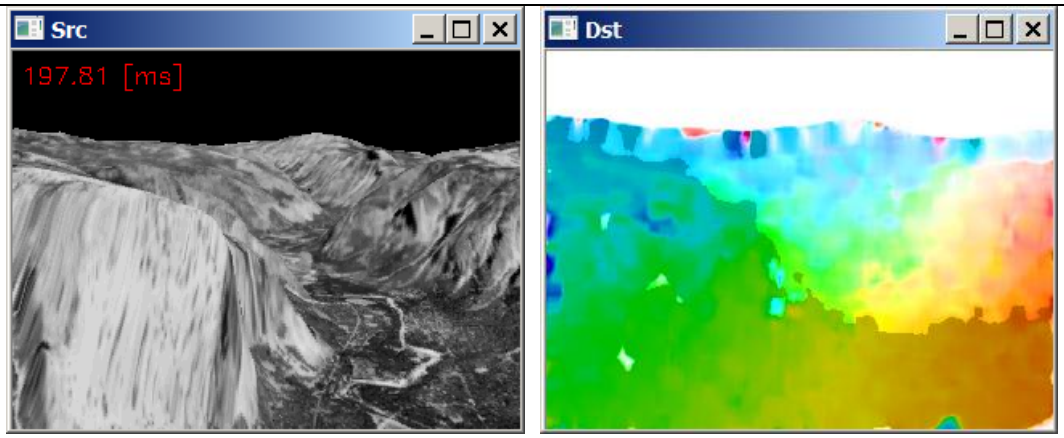


Fig. 7.1. Exemplificare a rezultatului codificării fluxului de mișcare in conformitate cu convenția de culoare Middleburry [2].

7.3. Măsurarea si afișarea timpului de procesare

Pentru a măsura timpul de procesare aferent cadrului curent puteți folosi funcțiile din OpenCV:

```
double t = (double)getTickCount(); // Get the crntent time [s]
// . . . insert here the processing functions / code
// Get the crntent time again and compute the time difference [s]
t = ((double)getTickCount() - t) / getTickFrequency();
// Print (in the console window) the processing time in [ms]
printf("%d - %.3f [ms]\n", frameNum, t*1000);
```



a. b.

Fig. 7.2. a. Imaginea sursa); b. imaginea destinație în care s-au afișat valorile vectorilor fluxului optic prin codificarea de culoare Middleburry.

7.4. Calculul histogramei direcțiilor vectorilor de mișcare

Calculul histogramei direcțiilor vectorilor de mișcare poate fi utilă în analiza mișcării din imagine. Pentru a calcula direcția vectorului de mișcare asociat fiecărui pixel trebuie calculat unghiul făcut de vector cu axa Ox (orizontală). Unghiul se va exprima in grade:

```
Point2f f = flow.at<Point2f>(r, c); // vectorul de miscare in punctual (r,c)
// vectorul de miscare al punctului este calculat se va afisa originea in imaginea
precedenta (prev)  si varful in imaginea curenta (crnt) -> pentru aceasta se iau
valorile lui din vectorul flow cu minus !
float dir_rad = pi + atan2(-f.y, -f.x); //directia vectorului in radiani
int dir_deg = dir_rad*180/pi; //folositi aceasta valuare la construirea histogramei
directiilor vectorilor de miscare
```

Histograma poate fi reprezentată sub forma unui vector de întregi, care va trebui reinițializat cu 0 in fiecare cadru/frame al secvenței de imagini prelucrate (deci în bucla de procesare a secvenței):

```
int hist_dir[360]={0};
```

Observație: Dacă la afișarea fluxului optic dens (*showFlowDense*) se face filtrarea vectorilor pe baza pragului *minVel* (*minVel* > 0) impus magnitudinii vectorilor de flux optic, aceeași condiție trebuie integrată și la construirea histogramei (se vor acumula în histogramă doar vectorii de flux optic cu magnitudinea > *minVel*).

7.5. Afișarea histogramei direcțiilor vectorilor de mișcare

Pentru fiecare cadru din secvența video se va calcula histograma direcțiilor si se va afișa într-o fereastră nouă. Afișarea histogramei direcțiilor se va putea realiza prin apelul funcțiilor *showHistogram* sau *showHistogramDir* integrate în modulul *Functions*.

Funcția *showHistogram* (vezi Anexele) este dedicată pentru afișarea unei histograme generice (Fig. 7.3.a) – vezi și laboratorul de Procesarea Imaginilor. Afișarea se face prin trasarea de segmente de dreaptă verticale de la baza histogramei pana la valoare curenta a histogramei, pentru fiecare intrare din vectorul histogramă („bin” – engleza).

Exemplu de apel:

```
showHistogram ("Hist", hist_dir, 360, 200, true);
// 200 [pixeli] = inaltimea ferestrei de afisare a histogramei
```

Funcția *showHistogramDir* este dedicată pentru afișarea histogramei direcțiilor vectorilor de flux optic (fig. 7.3.b). Afișarea se face prin trasarea de segmente de dreapta verticale de la baza histogramei pana la valoare curenta a histogramei, pentru fiecare intrare din vectorul histogramă („bin” – engleza). Fiecare „bin” al histogramei va avea culoarea codificată a direcției (vezi modulul *colorcode* și apelurile funcției `makeColorwheel()`, respectiv `make_HSI2RGB_LUT()` care inițializează un LUT necesar pentru generarea culorilor “bin”-urilor din histograma și este definită în *Function.cpp* și pe care le-ați integrat deja la începutul funcției de procesare).

Exemplu apel:

```
showHistogramDir ("Hist", hist_dir, 360, 200, true);
```

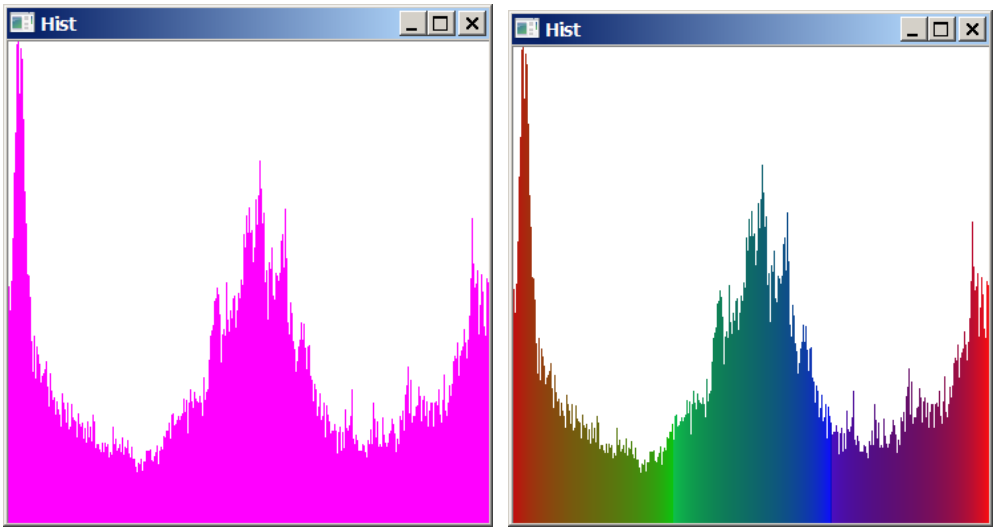


Fig. 7.3.a. Afișarea histogramei direcțiilor în forma generica (*ShowHistogram*); b. afișarea “bin”-urilor histogramei prin codurile de culoare ale direcțiilor (*ShowHistogramDir*)

7.6. Activități practice

4. Se va integra funcția de calcul a fluxului optic `calcOpticalFlowFarneback` și se va afișa harta densă a vectorilor de flux folosind codificarea *colorcode*. Se va afișa la linia de comandă și timpul de procesare pentru fiecare cadru.
5. Se vor vizualiza rezultate pentru diverse setări ale parametrilor de intrare ai funcției `calcOpticalFlowFarneback` și pentru parametrul de filtrare *minVel* (0.1 .. 1) al magnitudinii vectorilor de mișcare din funcția de afișare `showFlowDense`.
6. Se va calcula histograma direcțiilor vectorilor de mișcare și se va afișa. Se va folosi o condiție de filtrare bazată pe pragul *minVel* impus magnitudinii vectorilor de miscare la calulul histogramei direcțiilor (folosiți aceeași valoare ca și la pragul folosit la afișarea fluxului optic).
7. După ce s-a construit histograma direcțiilor, se va filtra vectorul histogramă obținut cu un filtru trece jos 1D (medie aritmetică sau gausian de dimensiune 5 sau 7). Sa va afișa histograma filetată cu ajutorul funcției *showHistogramDir*. Se va calcula poziția (unghiul) care corespunde maximului din histograma filtrată și se va afișa la linia de comandă.
8. Se vor vizualiza rezultatele (în varianta cu filtrarea modulului vectorilor) pe secvențe de imagini bitmap relevante (ex: *Polus.zip*).

9. Modificați partea de citire a secvențelor de imagini bitmap din funcția de procesare astfel încât sa puteți rula întregul flux de procesare (punctele 2 si 3 de la activitățile practice) pe secvențe video și/sau pe secvențe live de la camera web. Testați rezultatele pe câteva secvențe video cu camera statica (*taxi.avi, walkcircle.avi, walkstraight.avi, campus.avi, laboratory.avi* - http://users.utcluj.ro/~tmarita/HCI/Media/Video/). Folosiți un prag de filtrare *minVel* mai mare de 0.7 pentru a filtra zgomotele/artefactele introduse de compresia video foloistă în secvențele video respective.

Fig. 7.4. Rezultate experimentale pentru secvența video *Taxi.avi*.

10. **Tema de casa / proiect de semestru:** Folosind histograma direcțiilor de mișcare filetată cu un filtru trece jos, se vor detecta vârfurile (maximele locale) ale histogramei (vezi lucrarea de laborator L3 de la Procesarea Imaginilor [3]). Se vor afișa unghiurile (direcțiile) corespunzătoare acestor maxime locale în imaginea sursa (de ex. sub timpul de procesare, în grade) sau la linia de comandă. Se va experimenta implementarea pe un scenariu relevant (scenariu cu camera statica și obiecte în mișcare: *Taxi.avi* sau *walkstraight.avi*) în varianta cu filtrare a modulelor direcțiilor (ex. *minVel* = 0.7). Se va implementa un algoritm de segmentare (region growing + filtre morfologice) aplicat pe valorile direcțiilor de mișcare asociate fiecărui pixel

7.7. Bibliografie:

[1] Gunnar Farneback, Two-frame motion estimation based on polynomial expansion, Lecture Notes in Computer Science, 2003, (2749), 363-370,

[2] Simon Baker, Daniel Scharstein, J.P. Lewis, Stefan Roth, Michael J. Black, Richard Szeliski, A Database and Evaluation Methodology for Optical Flow, http://vision.middlebury.edu/flow/, Int J Comput Vis (2011) 92: 1–31.

[3] S. Nedevschi, T. Marita, R. Danescu, F. Oniga, R. Brehar, I. Giosan, C. Vancea, R. Varga, Procesarea Imaginilor - Îndrumător de laborator, editia a 2-a, Editura U.T. Press, Cluj-Napoca, https://biblioteca.utcluj.ro/carti-online-cu-coperta.html, 2023.

7.8. Anexe

Definiția funcției de afișare a fluxului optic dens: `showFlowDense`

```
/*-----------------------------------------------------------
Function used to display the DENSE optical flow vectors values using the Middlebury color
encoding
Input:
	name - destination (output) window name
	gray -  background image to be displayed (usually the prev image)
```

```
       flow - optical flow as a matrix of (x,y)
       minVel - threshold value (for vectors' modulus) for filtering out the displayed
vectors
Call example:
       showFlowDense (WIN_DST, crnt, flow, minVel, true)

-----------------------------------------------------------------*/
void showFlowDense (const string& name, const Mat& gray, const Mat& flow, float minVel, bool
showImages = true)
{
    if (showImages)
    {
        Mat cflow;
              cvtColor(gray, cflow, CV_GRAY2BGR);
              unsigned char color[3];
              Mat_<Vec3b> _cflow=cflow;

        for(int y = 0; y < flow.rows; ++y)
            for(int x = 0; x < flow.cols; ++x)
            {
              Point2f f = flow.at<Point2f>(y, x);

              //Middlebury color convetion
              computeColor(f.x, f.y, color);
              // gnerates a color that encodes the orientation and modulus of the OF vector
              float mod = sqrt(f.x * f.x + f.y * f.y); // modulus of the curent OV vector

              if (cflow.channels() == 3 && mod >= minVel )
              {
                     _cflow(y,x)[0] = color[0];
                     _cflow(y,x)[1] = color[1];
                     _cflow(y,x)[2] = color[2];
              }
            }
       cflow = _cflow;
       imshow(name, cflow);
    }
}
```

Definiția funcției de afișare a histogramei într-o singura culoare predefinita: `showHistogram`

```
/* Histogram display function - display a histogram using single color bars (simlilar to L3
/ PI)
Input:
       name - destination (output) window name
       hist - pointer to the vector containing the histogram values
       hist_cols - no. of bins (elements) in the histogram = histogram image width
       hist_height - height of the histogram image
Call example:
       showHistogram (WIN_HIST, hist_dir, 360, 200, true);
*/
void showHistogram (const string& name, int* hist, const int  hist_cols, const int
hist_height, bool showImages = true)
{
    if (showImages)
    {
              Mat imgHist(hist_height, hist_cols, CV_8UC3, CV_RGB(255, 255, 255));

              //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));
            }

            imshow(name, imgHist);
    }
}
```

Definiția funcției de afișare a histogramei direcțiilor cu coduri de culoare: `showHistogramDir`

```
/* Optical flow directions histogram display function - display a histogram using bars
colored in Middlebury color coding
Input:
      name - destination (output) window name
      hist - pointer to the vector containing the histogram values
      hist_cols - no. of bins (elements) in the histogram = histogram image width
      hist_height - height of the histogram image
Call example:
      showHistogramDir (WIN_HIST, hist_dir, 360, 200, true);
*/
void showHistogramDir (const string& name, int* hist, const int  hist_cols, const int
hist_height, bool showImages = true)
{
      unsigned char r, g, b;
    if (showImages)
    {
            Mat imgHist(hist_height, hist_cols, CV_8UC3, CV_RGB(255, 255, 255));

            //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));
                  r=HSI2RGB[x][R];
                  g=HSI2RGB[x][G];
                  b=HSI2RGB[x][B];
                  line(imgHist, p1, p2, CV_RGB(r, g, b));
            }

            imshow(name, imgHist);
    }
}
```

8. Detecția fețelor și a componentelor faciale

Scop: Scopul acestei lucrări este de a integra metoda Viola&Jones [1] de detecție a fețelor și a componentelor faciale. Metoda se bazează pe detecția de trăsături de tip Haar calculate pe sub-regiuni din imagine folosind imaginea integrală (vezi cursul 7-8) [2,3], și identificarea prezentei fețelor umane cu ajutorul unui clasificator de tip cascadă. Abordarea propusă de autori metodei oferă capabilități de timp real și versatilitate pentru detecția oricăror tipuri de obiecte pentru care s-a făcut o antrenare prealabilă a clasificatorului cascadă.

8.1. Modele de clasificare de obiecte bazate pe trăsături Haar

În OpenCV sunt disponibile modelele obținute prin antrenarea clasificatorilor bazați pe trăsături Haar sub forma unor fișiere *.xml. pentru diverse tipuri de obiecte, dar lista nu este exhaustivă putând fi extinsă de către utilizatori. Modelele clasificatorilor cascadă bazați pe trăsături Haar sunt disponibile în instalarea OpenCV la locația *OpenCV\sources\data\haarcascades* și sunt următoarele:

haarcascade_eye.xml
haarcascade_eye_tree_eyeglasses.xml
haarcascade_frontalface_alt.xml
haarcascade_frontalface_alt_tree.xml
haarcascade_frontalface_alt2.xml
haarcascade_frontalface_default.xml
haarcascade_fullbody.xml
haarcascade_lefteye_2splits.xml
haarcascade_lowerbody.xml
haarcascade_mcs_eyepair_big.xml
haarcascade_mcs_eyepair_small.xml
haarcascade_mcs_leftear.xml
haarcascade_mcs_lefteye.xml
haarcascade_mcs_mouth.xml
haarcascade_mcs_nose.xml
haarcascade_mcs_rightear.xml
haarcascade_mcs_righteye.xml
haarcascade_mcs_upperbody.xml
haarcascade_profileface.xml
haarcascade_righteye_2splits.xml
haarcascade_upperbody.xml

Există și modele pentru clasificatori cascadă care folosesc și alte tipuri de trăsături:
- Local Binary Patterns (LBP): *lbpcascade_frontalface.xml (OpenCV\sources\data\lbpcascades)*
- Histogram of Oriented Gradients (HOG): *hogcascade_pedestrians.xml, (OpenCV\sources\data\hogcascades)*

8.2. Detecția obiectelor folosind clasificatori cascadă în OpenCV

Detecția obiectelor se poate realiza apelând funcția:

void `CascadeClassifier::detectMultiScale`(const Mat& **image**, vector<Rect>& **objects**, double **scaleFactor**=1.1, int **minNeighbors**=3, int **flags**=0, Size **minSize**=Size(), Size **maxSize**=Size())

Unde:

- **CascadeClisifier** – este clasa din care se instanțiază obiectul (modelul) pentru clasificatorul cascadă. Modelul poate fi încărcat dintr-un fișier XML sau YAML folosind funcția Load().
- **image** – este o matrice de tip CV_8UC1 conținând imaginea sursă
- **objects** – este un vector de ieșire conținând regiunile de interes (dreptunghiuri) care conțin obiectele detectate
- **scaleFactor** – este un parametru care specifică factorul de scalare folosit în detecția multi-rezoluție.
- **minNeighbors** – este un parametru care specifică câți vecini ai fiecărui candidat (zone rectangulare) se rețin
- **flags** – acest parametru nu se folosește în modelele cascadă de tip nou (doar în cele vechi – cvHaarDetectObjects)
- **minSize** – este dimensiunea rectangulară minimă a obiectului detectabil. Obiecte mai mici sunt ignorate
- **maxSize** – este dimensiunea rectangulară maximă a obiectului detectabil. Obiecte mai mari sunt ignorate

Un tutorial care exemplifică folosirea clasificatorului cascadă pentru detecția fețelor este prezentat în [4]. Se indică modul în care se poate realiza detecția tuturor fețelor dintr-o imagine. Pentru fiecare față detectată (în regiunea rectangulară care încadrează fiecare față) se poate face detecția componentelor faciale (ochi, gura, nas). Un șablon derivat din acest tutorial este dat în exemplul de mai jos:

```
/* ----------------------------------------------------------------------------
 Detects all the faces and eyes in the input image
 window_name - name of the destination window in which the detection results are
displayed
 frame - source image
 minFaceSize - minimum size of the ROI in which a Face is searched
 minEyeSize - minimum size of the ROI in which an Eye is searched
      acording to the antropomorphic features of a face, minEyeSize = minFaceSize / 5
Usage: FaceDetectandDisplay( "Dst", dst, minFaceSize, minEyeSize );
---------------------------------------------------------------------------- */
void FaceDetectandDisplay( const string& window_name, Mat frame,
                                        int minFaceSize, int minEyeSize )
{
  std::vector<Rect> faces;
  Mat frame_gray;

  cvtColor( frame, frame_gray, CV_BGR2GRAY );
  equalizeHist( frame_gray, frame_gray );

  //-- Detect faces
  face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0,
                                                 Size(minFaceSize, minFaceSize) );
  for( int i = 0; i < faces.size(); i++ )
  {
    // get the center of the face
    Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
    // draw circle around the face
    ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5), 0, 0,
                                                 360, Scalar( 255, 0, 255 ), 4, 8, 0 );
    Mat faceROI = frame_gray( faces[i] );
```

```
    std::vector<Rect> eyes;
    //-- In each face (rectangular ROI), detect the eyes
    eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0,
                                                    Size(minEyeSize, minEyeSize) );
    for( int j = 0; j < eyes.size(); j++ )
     {
       // get the center of the eye
       //atentie la modul in care se calculeaza pozitia absoluta a centrului ochiului
       // relativa la coltul stanga-sus al imaginii:
       Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5,
                                  faces[i].y + eyes[j].y + eyes[j].height*0.5 );
       int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
       // draw circle around the eye
       circle( frame, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
     }
  }
  imshow( window_name, frame ); //-- Show what you got
 }
```

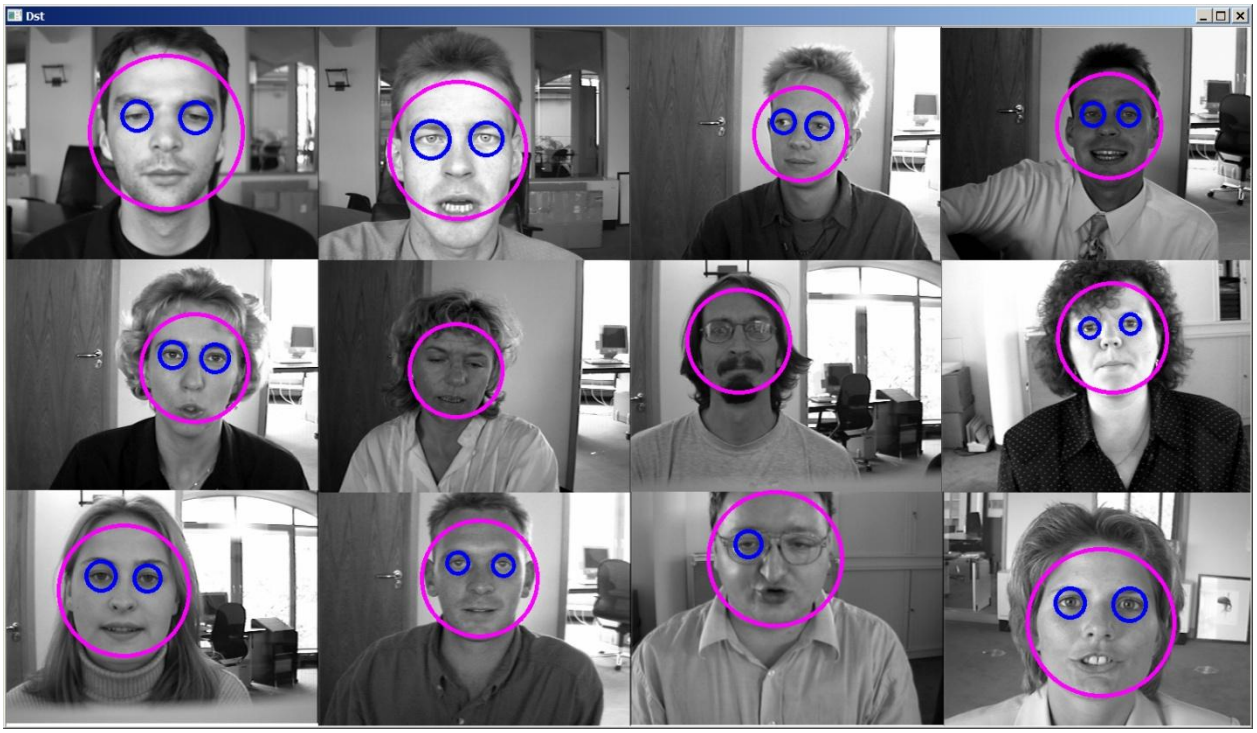


Fig. 8.1. Rezultatele apelului funcției `FaceDetectandDisplay` pe imagini din setul de date BioID[5].

8.3. Detalii de implementare

Pentru detecția obiectelor, în cadrul funcției *OpenCVApplication.cpp* trebuie sa mai faceți următoarele:

- sa declarați următoarele variabile globale:

```
CascadeClassifier face_cascade; // cascade clasifier object for face
CascadeClassifier eyes_cascade; // cascade clasifier object for eyes
CascadeClassifier mouth_cascade; // cascade classifier object for face
CascadeClassifier nose_cascade; // cascade classifier object for eyes
```

- sa vă asigurați că modelele folosite ale clasificatorilor cascadă (fișierele *.xml)

haarcascade_eye_tree_eyeglasses.xml
haarcascade_frontalface_alt.xml
haarcascade_mcs_mouth.xml
haarcascade_mcs_nose.xml

există în directorul de lucru curent (directorul rădăcină) al proiectului OpenCVApplication.

În funcția de procesare aferentă laboratorului curent, înainte de apelul funcției `FaceDetectandDisplay`, este necesară încărcarea modelelor clasificatorilor, ca și în exemplul de mai jos.

```
String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
String mouth_cascade_name = "haarcascade_mcs_mouth.xml";
String nose_cascade_name = "haarcascade_mcs_nose.xml";

// Load the cascades
if (!face_cascade.load(face_cascade_name))
{
	printf("Error loading face cascades !\n");
	return;
}
if (!eyes_cascade.load(eyes_cascade_name))
{
	printf("Error loading eyes cascades !\n");
	return;
}
.........
```

Exemplu de apel al funcției `FaceDetectandDisplay` (în funcția de procesare aferenta laboratorului curent):

```
...
src = imread("Images/Facesx12.bmp", CV_LOAD_IMAGE_COLOR);
dst=src.clone();
int minFaceSize = 30;
int minEyeSize = minFaceSize /5; // conform proprietatilor antropomorfice ale
fetei (idem pt. gura si nas)
FaceDetectandDisplay( WIN_DST, dst, minFaceSize, minEyeSize );
...
```

8.4. Restricționarea detecției în regiuni de interes (ROI)

Pentru detecția altor componente faciale (ex. gura, nas), modelul cascada *haarcascade_mcs_mouth.xml* nu este foarte precis. În consecință se poate restricționa regiunea de căutare la o subregiune a feței pentru a evita detecții fals-pozitive. Pentru căutarea componentelor faciale într-o Regiune de Interes (ROI) care sa fie o subregiune a feței se va proceda astfel:

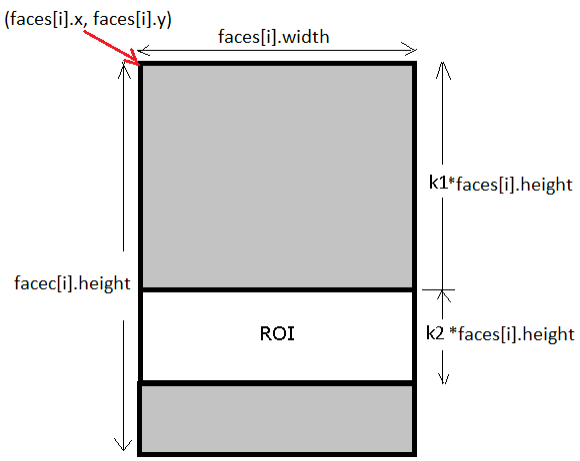


Fig. 8.2. Definirea regiunilor de interes (ROI) din cadrul unei fețe pentru detecția anumitor componente faciale specifice.

Se definește o noua ROI rectangulară pentru o subregiune a feței care corespunde unei anumite componente faciale):

```
Rect comp_rect;
comp_rect.x=faces[i].x;
comp_rect.y=faces[i].y + k1*faces[i].height;
comp_rect.width=faces[i].width;
comp_rect.height= k2*faces[i].height;
```

unde: *k1, k2* sunt valori subunitare și *k1 + k2 <1* (vedeți valorile specificate la punctul 2 din mersul lucrării !!!)

Se extrage din imaginea sursă matricea de intensități corespunzătoare ROI definite:

```
Mat comp_ROI = frame_gray(comp_rect);
```

Se apelează detectorul de obiecte. De exemplu, pentru gură. procedați astfel:

```
Rect mouth_ROI;
std::vector<Rect> mouth;
//-- In each face (rectangular ROI), detect the mouth
mouth_cascade.detectMultiScale(moth_ROI, mouth, 1.1, 1, 0,
                                        Size(minMouthSize, minMouthSize) );
```

8.5. Activități practice

1. Se va integra într-o funcție de procesare nouă metoda de detecție a fețelor (pentru detecția tuturor instanțelor dintr-o imagine) și a celorlalte componente faciale (ochi, nas, gură) din fiecare față detectată (funcția `FaceDetectandDisplay`). Imagini de test puteți descarcă de la următoarea adresă: http://users.utcluj.ro/~tmarita/HCI/Media/Images/Faces.zip

Observație:

- Cel mai simplu este sa afișați componentele faciale detectate sub forma unor dreptunghiuri, folosind funcția de desenare *rectangle*: `rectangle(frame, rect, culoare, grosime, 8, 0);`

2. Se va extinde procesarea de la pasul 1 pentru detecția și a altor componente faciale (gura, nas). Optimizați însă căutarea componentelor faciale doar în anumite zone din ROI-ul corespondent feței ca și în exemplul de mai jos:

```
Rect eyes_rect; //ochii ocupa o fisie intre 20%-55% din inaltimea fetei
eyes_rect.x = faces[i].x;
eyes_rect.y = faces[i].y + 0.2*faces[i].height;
eyes_rect.width = faces[i]. width;
eyes_rect.height = 0.35*faces[i].height;
Mat eyes_ROI = frame_gray(eyes_rect);
Rect nose_rect; //nasul ocupa o fisie intre 40%-75% din inaltimea fetei
…
Rect mouth_rect; // gura ocupa o fisie intre 70%-99% din inaltimea fetei
…
```

Observație:

- atenție la modul în care se calculează poziția absolută a componentei faciale relative la colțul stânga-sus al imaginii: componentele detectate au coordonate relative la regiunea de interes (ROI) în care le-ați căutat ⇒ mai departe trebuie sa le translatați relativ la coltul stânga-sus al imaginii destinație ca sa le afișați corect (vezi exemplul dat in `FaceDetectandDisplay`)

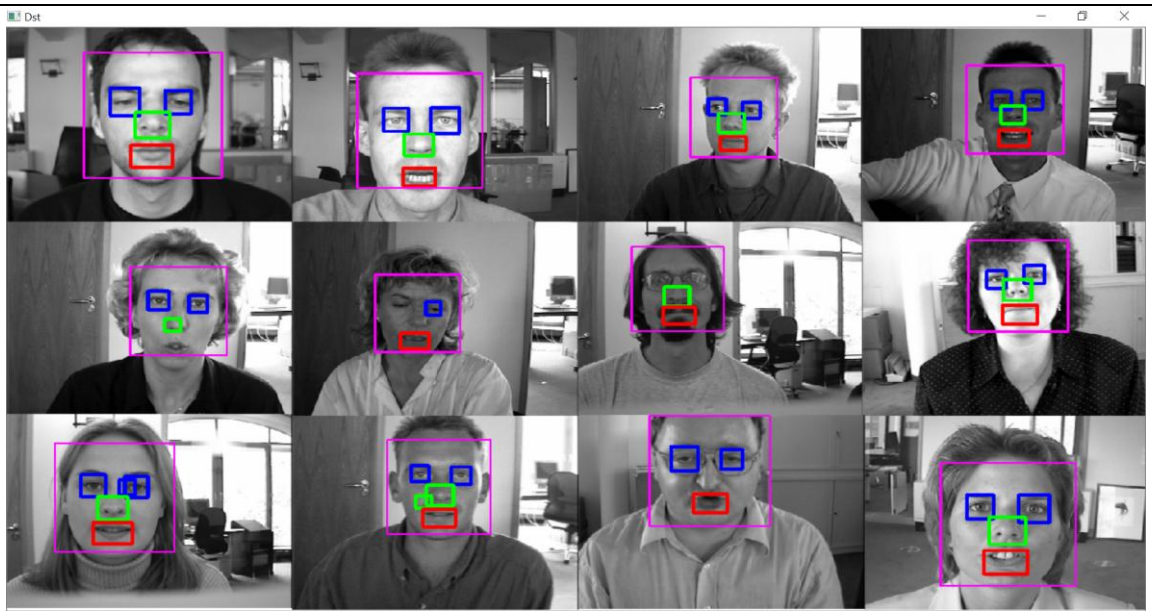


Fig. 8.3. Rezultatele apelului funcției FaceDetectandDisplay modificată pentru detecția fețelor și a componentelor faciale (varianta optimizată prin restricționarea zonelor în care se caută componentele faciale) pe imagini din setul de date BioID[5].

3. Se va crea o noua funcție de procesare pentru detecția fețelor din secvențe video (off-line / on-line). Pentru acesta cerință creați o versiune simplificată a funcției `FaceDetectandDisplay` care sa detecteze și afișeze doar fețele (renunțați la căutarea celorlalte componente faciale). Se va măsura și afișa (la linia de comandă) timpul de procesare pentru apelul funcției `FaceDetectandDisplay`.

4. Se va testa detecția fețelor (fără componente faciale) folosind modelul de clasificator bazat pe trăsături LBP (Local Binary Pattern) în locul celui bazat pe trăsături Haar. Pentru aceasta se va încarcă modelul clasificatorului LBP *lbpcascade_frontalface.xml* în locul *haarcascade_frontalface_alt.xml*

8.6. Bibliografie

[1] Viola, P., Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, Vol. 1, pp. I-511-I518, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=990517
[2] T. Marita, Interactiune Om-Calculator, note de curs, http://users.utcluj.ro/~tmarita/HCI/C7-8.pdf.
[3] OpenCV Face Detection: Visualized, http://vimeo.com/12774628
[4] OpenCV Tutorials, *objdetect* module. Object Detection, http://docs.opencv.org/2.4.11/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html#cascade-classifier
[5] BioID Face Database, https://www.bioid.com/face-database/

9. Validarea detecției feței și a ochilor pe secvențe de imagini

Scop: Scopul acestei lucrări este de a combina metoda Viola&Jones de detecție a fețelor și a ochilor cu o validare bazată pe analiza unei secvențe de imagini. Un exemplu tipic de utilizare a unei astfel de abordări este în aplicații de tip „liviness detection" în sistemele biometrie bazate pe recunoaștere facială. Astfel de aplicații încearcă sa valideze suplimentar un algoritm de recunoaștere facială printr-o analiza suplimentara pentru a preveni încercările de fraudare a sistemului biometric prin prezentarea unui imagini statice cu poze ale persoanei în loc de prezența efectivă a persoanei. La modul concret se va face o validare bazată pe *detecția clipitului* persoanei respective. Din punct de vedere practic se va realiza o integrare a implementărilor din lucrările L5 (Segmentarea obiectelor în mișcare prin modelarea și eliminarea fundalului - "Background Subtraction") și L8 (Detecția fețelor și a componentelor faciale) cu procesări de imagini suplimentare.

9.1. Detalii de implementare

Se va citi secvența video de test (`test_msv1_short.avi`). Procesarea se va face cadru cu cadru, avansul între cadre făcându-se prin apăsarea unei taste (setați la ‚0' parametrul funcției `waitKey(0)`). Se va converti cadrul curent (de analizat) din color (cu trei canale) în grayscale (matrice cu un canal). Pentru fiecare cadru (începând cu al doilea) se vor realiza următoarele operații:

1. Se va apela metoda de detecție a fețelor și a componentelor faciale pe cadrul curent. Se va memora poziția (dreptunghiul care încadrează) prima față găsita în cadrul curent (`Rect faceROI=faces[0]`). Acest dreptunghi se va folosi ulterior ca și mască de procesare (Region Of Interest – ROI).

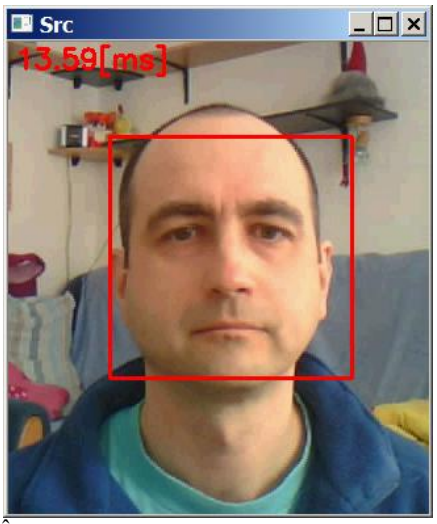


Fig. 9.1. Încadrarea primei fete găsite in imaginea curentă.

2. Se va implementa operația de „Background Subtraction" prin metoda de diferențiere simplă între cadrul curent și cel precedent. În momentul în care persoana clipește (momentul de tranziție de la pleoape deschise la pleoape închise) în zona ochilor vor apare pixeli albi în imaginea diferență binarizată:

3. Pe imaginea diferență se vor aplica operații morfologice (o eroziune urmata de o dilatare) pentru eliminarea zgomotelor datorate eventualelor mișcării ale capului, zgomotului din imagine sau artefactelor introduse de compresia secvenței video.

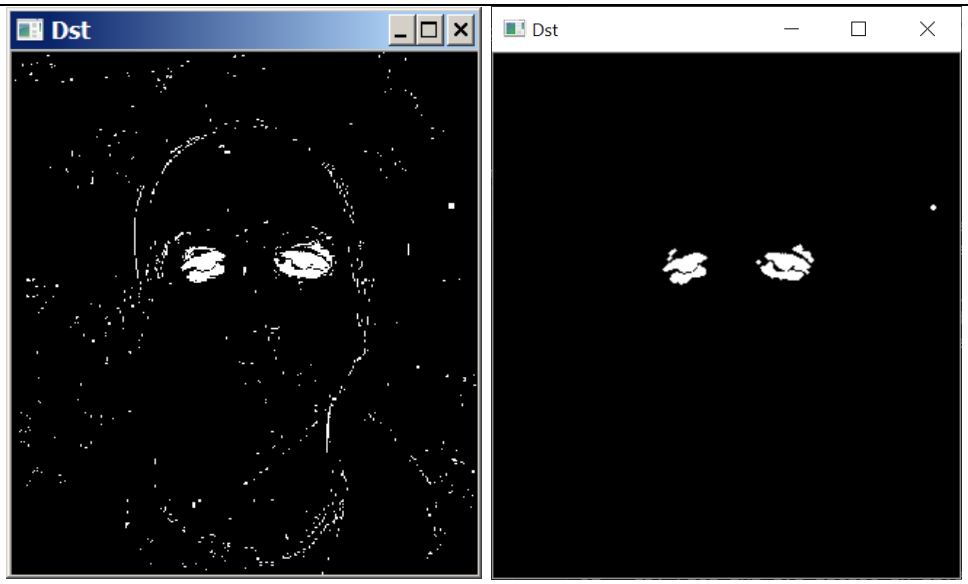


Fig. 9.2. Stânga: imaginea diferență dintre cadrul curent și cadrul precedent; Dreapta: - imaginea diferență după aplicarea filtrelor morfologice (o eroziune urmată de o dilatare cu nucleu pătrat de dimensiune 3x3)

4. Se va masca imaginea diferență finală (*dst*) de la pasul 3 cu ROI-ul feței obținut la pasul 1):

```
temp = dst(faceROI); // mat grayscale pt. procesarea ROI
```

5. Se realizează etichetarea obiectelor rămase în zona ROI aferentă feței (din matricea *temp*). Se calculează proprietățile geometrice ale componentelor conexe (etichetelor) detectate: aria și centrul de masa. Pentru etichetarea obiectelor și calculul proprietăților geometrice (arie si centre de masa) folosiți codul dat ca si exemplu in funcția `Labeling` din modulul *Functions*.

```
// Labeling -----------------------------------------
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
Mat roi = Mat::zeros(temp.rows, temp.cols, CV_8UC3); // matrice (3 canale) folositea
pentru afisarea (color) a obiectelor detectate din regiunea de interes
findContours(temp, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_NONE);
Moments m;
if (contours.size() > 0)
{
	// iterate among all the top-level contours,
	// draw each connected component with its own random color
	int idx = 0;
	for (; idx >= 0; idx = hierarchy[idx][0])
	{
		const vector<Point>& c = contours[idx];

		m = moments(c); // calcul momente
		double arie = m.m00; // aria componentei conexe idx
		double xc = m.m10 / m.m00; // coordonata x a CM al componentei conexe idx
		double yc = m.m01 / m.m00; // coordonata y a CM al componentei conexe idx

		Scalar color(rand() & 255, rand() & 255, rand() & 255);
		drawContours(roi, contours, idx, color, FILLED, 8, hierarchy);
	}
}
```

Fig. 9.3. Rezultatul aplicării operației de etichetare pe imaginea binară a ROI-lui feței.

6. Se vor adăuga ariile și centrele de masa ale obiectelor detectate într-o lista (vector) de candidați Folosiți o structura pentru a memora elementele vectorului, ca și în exemplul de mai jos:

```
// Declararea structurii in care se vor tine datele aferente fiecarei etichete
// Puteti insera declaratiile de mai jos la inceputul functiei de procesare
typedef struct {
	double arie;
	double xc;
	double yc;
} mylist;
	....
vector<mylist> candidates;
candidates.clear(); // se apeleaza pt. fiecare cadru
// Labeling code …
	// Pt. fiecare obiect
		// adauga proprietatile geometrice in lista candidates
			mylist elem;
			elem.arie = arie;
			elem.xc = xc;
			elem.yc = yc;
			candidates.push_back(elem);
```

7. Se va implementa un arbore de decizie bazat pe câteva reguli simple deduse din trăsăturile antopomorfice ale feței:

- Se rețin pozițiile centrelor de masa ale celor doua obiecte (din lista de candidați) care au aria cea mai mare (ca să eliminați eventualele zgomote ramase în imagine chiar și după aplicarea filtrelor morfologice).
- Pozițiile centrelor de masa (CM) ale acestor obiecte trebuie sa fie aproximativ pe o linie orizontală (exemplu: diferențele pe y ale CM sa fie mai mici decât o valoare *ky*faceROI.height* (ex. *ky* ≈ 0.1) ȘI sa fie situate în jumătatea superioară a ROI)
- Diferențele pe orizontală (distanța pupilara DP) dintre centrele de masă (CM) să fie un procent din lățimea feței: *kx1*faceROI.width* < DP < *kx2*faceROI.width* (ex: *kx1* ≈ 0.3, *kx2* ≈ 0.5)
- Pozițiile centrelor de masă ale celor mai mari două obiecte trebuie sa apară în cele 2 jumătăți verticale diferite ale feței: *xc_ochi_ochi_stang* > *faceROI.width/2* (ochiul stâng îl vedem în imagine în dreapta) și *xc_ochi_ochi_drept* < *faceROI.width/2* (ochiul drept îl vedem în imagine în stânga)
- Se vor desena în fereastra aferenta ROI (matricea *roi*) centrele de masă ale obiectelor validate folosind funcția `DrawCross` (definită în modulul *Functions*): cu culoare roșie pentru ochiul stâng și cu culoare albastră pentru ochiul drept (imaginea pe care o observam este în oglindă).

Observație: va trebui sa ajustați constante (*kx*, *ky* la valori optime)!

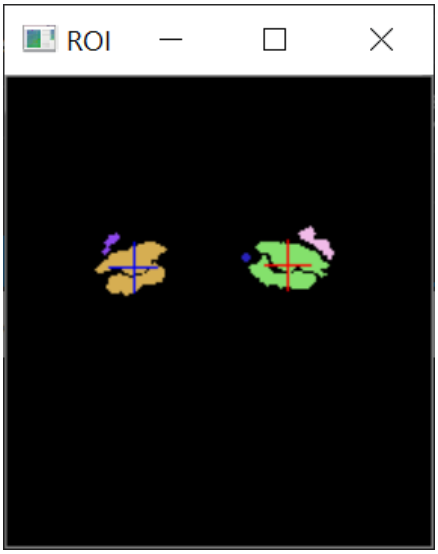


Fig. 9.4. Marcarea centrelor de masă ale celor mai mari 2 obiecte cu o cruce roșie pentru ochiul stâng (în jumătatea stânga a feței) si cu o cruce albastra pentru ochiul drept (in jumătatea dreapta a fetei) - imaginea pe care o observăm este în oglindă !

8. Se va afișa în imaginea sursă fața detectată cu culoarea verde dacă sa trecut cu succes prin procesul de validare (s-a detectat clipitul: tranziția de la ochi deschiși la ochi închiși sau invers) și cu roșu în caz contrar (ochii deschiși). Se va afișa si timpul de procesare.

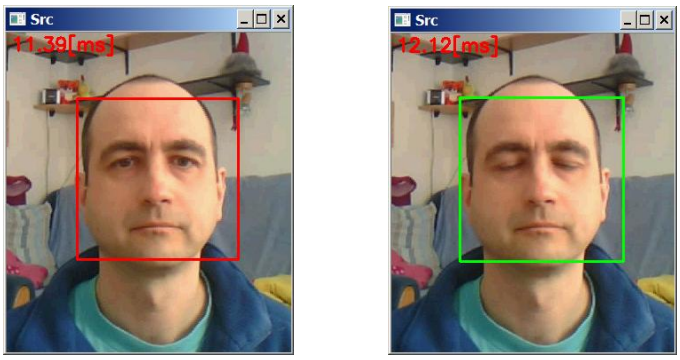


Fig. 9.5. Afișarea momentului în care se detectează clipitul prin schimbarea culorii dreptunghiului care încadrează fața.

Pentru desenarea unui dreptunghi aveți la dispoziție în OpenCV funcția `rectangle(frame, faceROI, color, 2, 8, 0)`, unde *color* este o structura de tip *Scalar(B,G,R)*

Pentru a afișa timpul de procesare *t* peste imagine sursa folosi funcția `putText`:

```
char msg[100];
sprintf(msg, "%.2f[ms]", t * 1000);
putText(frame, msg, Point(5, 20), FONT_HERSHEY_SIMPLEX, 0.7, color 2, 8);
```

9.2. Activități practice

1. Implementați pașii de procesare descriși in capitolul 9.1 într-o singură funcție de procesare. Testați implementarea pe secvența video `test_msv1_short.avi`.
2. Testați implementarea și pe stream-ul video live captat cu camera web proprie, înlocuind modul de inițializare al obiectului *VideoCapture*:
   ```
   VideoCapture cap(0);  // video from webcam
   ```
   Daca este necesar ajustați parametrii folosiți în algoritm pentru o funcționare optimă.

10. Detecția de persoane folosind trăsături HAAR și validări antropomorfice

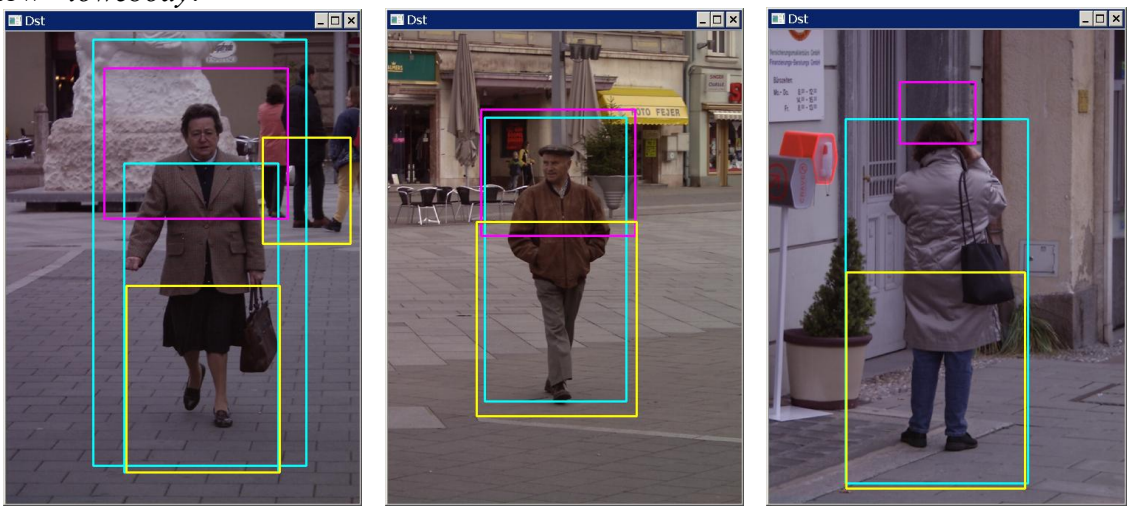
Scop: Scopul acestei lucrări este de a implementa o metodă de detecție de persoane sau părți corporale. Metoda se bazează pe detecția de trăsături de tip Haar calculate pe sub-regiuni din imagine folosind imaginea integrală (vezi cursurile 7-8) și identificarea prezenței persoanelor sau a unor parți ale corpului uman cu un clasificator de tip cascadă, urmată de validări suplimentare aplicate pentru gruparea și fuziunea părților corporale detectate care aparțin aceleiași persoane prin reguli bazate pe trăsăturile antropomorfice.

Modelele clasificatorilor cascadă bazați pe trăsături Haar pentru detecția de persoane sunt disponibile la locația: *%OPENCV_DIR%\data\haarcascades* din instalarea OpenCV și sunt următoarele:

haarcascade_fullbody.xml
haarcascade_lowerbody.xml
haarcascade_upperbody.xml
haarcascade_mcs_upperbody.xml

10.1. Detalii de implementare

1. Se va crea o funcție de procesare folosind șablonul de la metoda de detecție a fețelor pe imagini statice (L8). Se va modifica aceasta funcție pentru a încarcă modelele de clasificatori pentru persoane (*fullbody, lowerbody, upperbody*). Se va apela funcția de detecție a obiectelor `detectMultiScale`, succesiv, pentru fiecare dintre cele 3 modele.
2. Se va parcurge lista de obiecte obținută pentru fiecare tip de obiect și se vor afișa dreptunghiurile care încadrează fiecare parte corporala în culori distincte: cyan - *fullbody*, magenta - *upperbody*, yellow - *lowebody*.



a. b. c.

Fig. 10.1. Rezultate detecție (detecția pentru fiecare model în parte *fullbody, lowerbody, upperbody* încadrată cu câte un dreptunghi de culoare diferită)

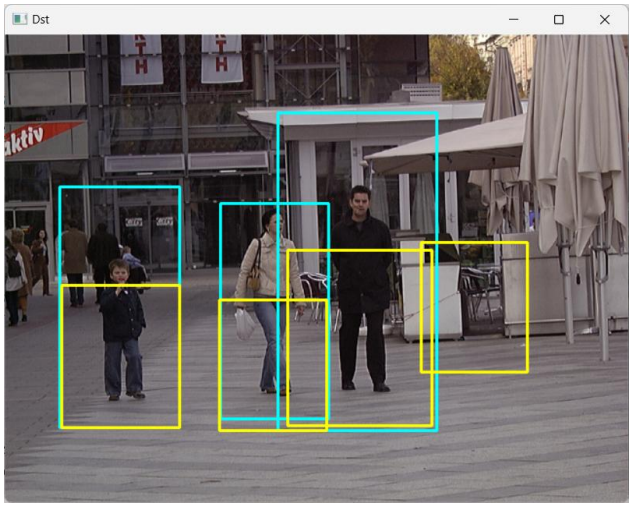
Observație: ultimii 2 parametri ai funcției `detectMultiScale` specifică dimensiunea minimă inițială (*width, height*) a zonei rectangulare în care se caută părțile corporale. Pentru fiecare tip de obiect se vor respecta proporțiile antropomorfice:

(fullbody, lowerbody, upperbody) → width ~ 0.4 .. 0.5 * `minBodyHeight`;
(lowerbody, upperbody) → height ~ 0.5* `minBodyHeight`;

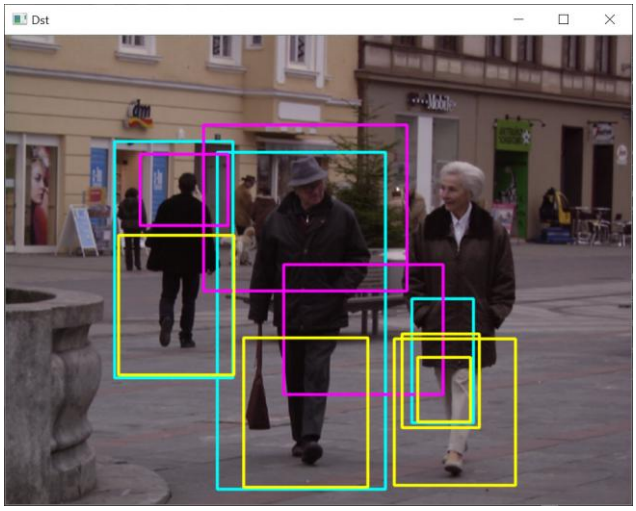
fullbody → height = `minBodyHeight` (aproimativ 100 ... 150 pt. imaginile de test din setul de date de test)

Rezultate detecțiilor, cu parametrii de apel de mai jos se pot observa în Fig. 10.2:

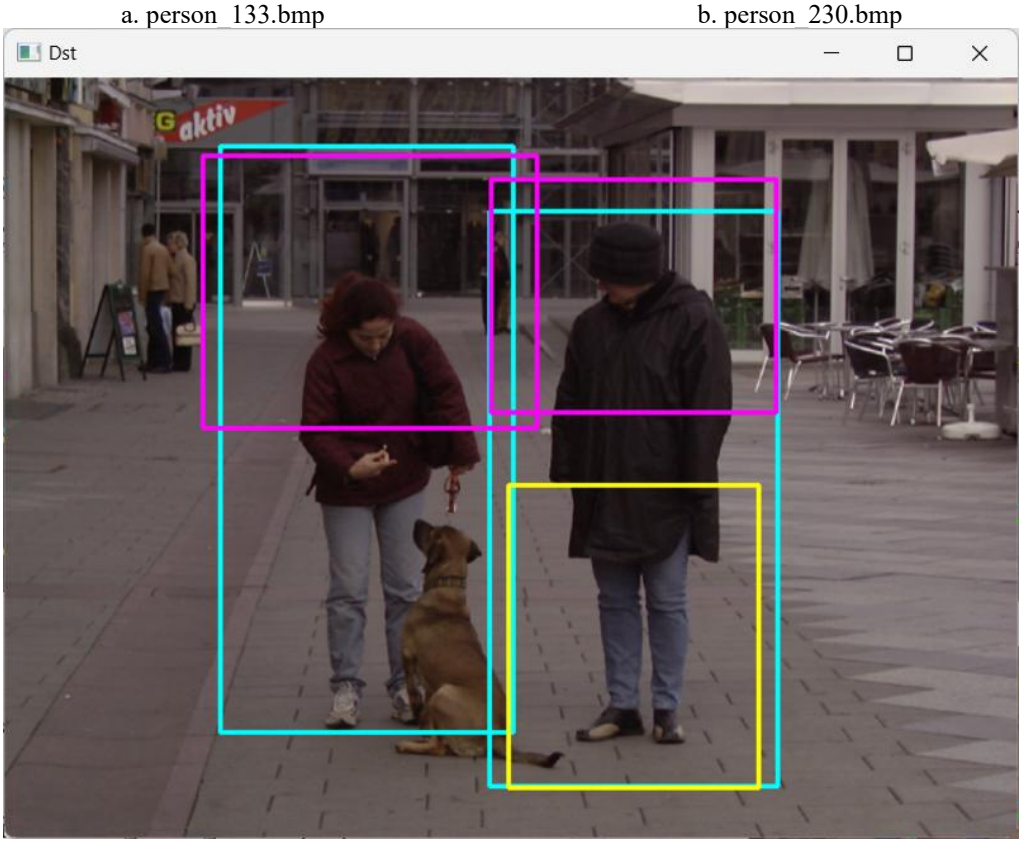
```
int minBodyHeight = 150;
//-- Detect bodies
fullbody_cascade.detectMultiScale(frame_gray, fbodies, 1.1, 2, 0, Size(minBodyHeight *
0.5f, minBodyHeight));
upperbody_cascade.detectMultiScale(frame_gray, ubodies, 1.1, 2, 0, Size(minBodyHeight *
0.5f, minBodyHeight * 0.5));
lowerbody_cascade.detectMultiScale(frame_gray, lbodies, 1.1, 2, 0, Size(minBodyHeight *
0.5f, minBodyHeight * 0.5));
```



a. person_133.bmp



b. person_230.bmp



c. person_238.bmp

Fig. 10.2. Rezultatele detecțiilor pe imagini cu persoane multiple.

3. Se va face o validare suplimentară pe baza stabilirii unui set de reguli legate de proporțiile obiectelor detectate și de poziția lor relativă. Pentru fiecare persoană detectată se va afișa un scor de încredere/confidență (CF).

   Exemplu:

   a. Dacă se detectează *3 obiecte* (*fullbody, lowerbody, upperbody*) cu centrele aproximativ pe accesași verticală (diferența pe orizontală între centrele acestor dreptunghiuri sa fie mai mică decât 0.5 * `minBodyHeight`) **ŞI** centrul lor au o valoare crescătoare a coordonatei y în ordinea *upperbody, fullbody,lowerbody,* **ŞI** ariile lor respectă următoarea relație: 0.7 < aria(*fullbody*) / aria(*lowerbody* ∪ *upperbody*) < 1.3) **atunci** scorul CF este de este **0.99** și se reține un dreptunghi (rectangle) obținut prin *fullbody* ∩ (*lowerbody* ∪ *upperbody*) (vezi exemplele din figura 10.3).

   b. Dacă se detectează *2 obiecte (lowerbody, upperbody)* cu centrele aproximativ pe accesași verticală **ŞI** *upperbody* este detectat deasupra lui *lowerbody* **ŞI** distanța pe verticală între centrele lor este mai mică decât 2.5 * `minBodyHeight` **atunci** scorul CF este **0.66** și se reține un dreptunghi (*lowerbody* ∪ *upperbody*).

   c. Dacă se detectează *2 obiecte (upperbody, fullbody)* cu centrele aproximativ pe accesași verticală **ŞI** centrul lui *upperbody* este detectat deasupra centrului lui *fullbody* **ŞI** aria(*upperbody* ∩ *fullbody)* / aria(*uperbody)* > 0.5 **atunci** scorul CF este **0.66** și se reține un dreptunghi (*upperbody* ∪ *fullbody).*

   d. Daca se detectează *2 obiecte (lowerbody, fullbody)* cu centrele aproximativ pe accesași verticală **ŞI** centrul lui *lowerbody* este detectat sub centrul lui *fullbody* **ŞI** aria(*lowerbody* ∩ *fullbody)* / aria(*lowerbody)* > 0.5, **atunci** scorul CF este **0.66** și se reține un dreptunghi (*lowerbody* ∪ *fullbody).*

   e. Obiectele neprocesate se raportează ca și persoane distincte cu scorul CF de **0.33**, dacă nu se suprapun cu obiecte validate la criteriile a .. d.

Sugestii pentru implementare:

- Creați-vă un vector pentru a stoca dreptunghiurile care circumscriu persoanele validate și unul pentru scorurile de confidență asociate/calculate (exemplu: `persons si pesronsCF`). Odată ce ați validat o persoana conform criteriilor a ... e inserați dreptunghiul și scorul CF în vectorii creați.
- Utilizați niște vectori (`fproc, lproc, uproc`, având accesași lungime ca și vectorii `fbodies, ubodies, lbodies`) și îi inițializați cu 0 pentru a marca părțile corporale pe care le-ați procesat. Odată ce ați procesat o parte corporală, schimbați valoarea din vector din 0 in 1 pentru a evita sa îl reprocesați/considerați ulterior.
- Pentru criteriul e:
  - parcurgeți lista/vectorul cu persoane validate (`persons`) și pentru fiecare element *i* din listele/vectorii `fbodies, ubodies, lbodies` (care este marcat ca neprocesat în vectorii `fproc, lproc, uproc`), verificați dacă se suprapune peste vreun element din lista de persoane validate (`persons`), pe baza unui test de arie de tipul: *aria(Xbodies[i]) ∩ persons[ii])) / aria(Xbodies[i]) > 0.5*. În caz afirmativ îl marcați ca și procesat (fără sa îl mai inserați în vectorul `persons`).
  - Parcurgeți din nou, pe rând elementele din vectorii `fbodies, ubodies, lbodies`, și dacă sunt marcate ca neprocesate în `fproc, lproc, uproc`, le schimbați starea din 0 (neprocesat) în 1(procesat) și le inserați în lista `persons` cu un factor de confidență CF = **0.33**.
- Pentru a calcula centrul unui structuri de tip *Rect* și aria acestuia aveți funcțiile definite în modulul Functions: `RectCenter` si `RectArea` (vezi si anexa).

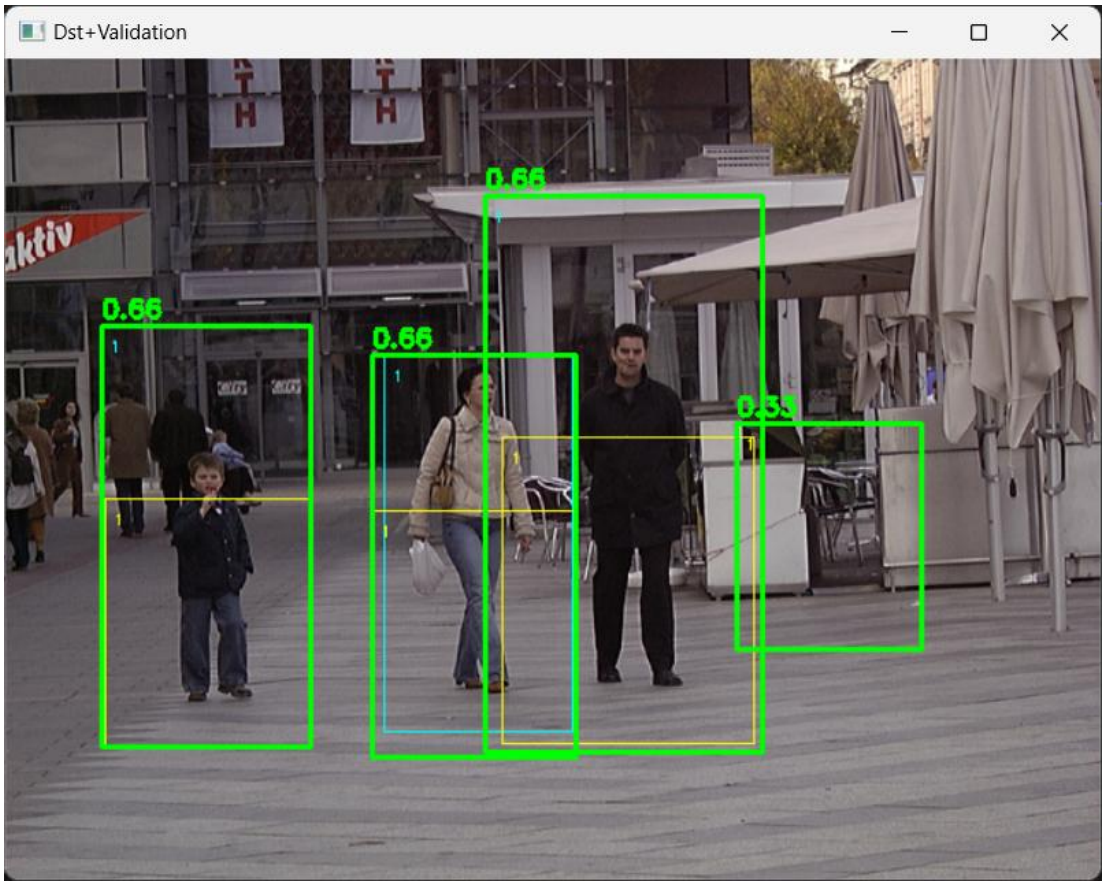
- Pentru a realiza intersecția respectiv reuniunea dintre două structuri de tip *Rect,* aveți operatorii & respectiv | din OpenCV (vezi anexa).

10.2. Activități practice

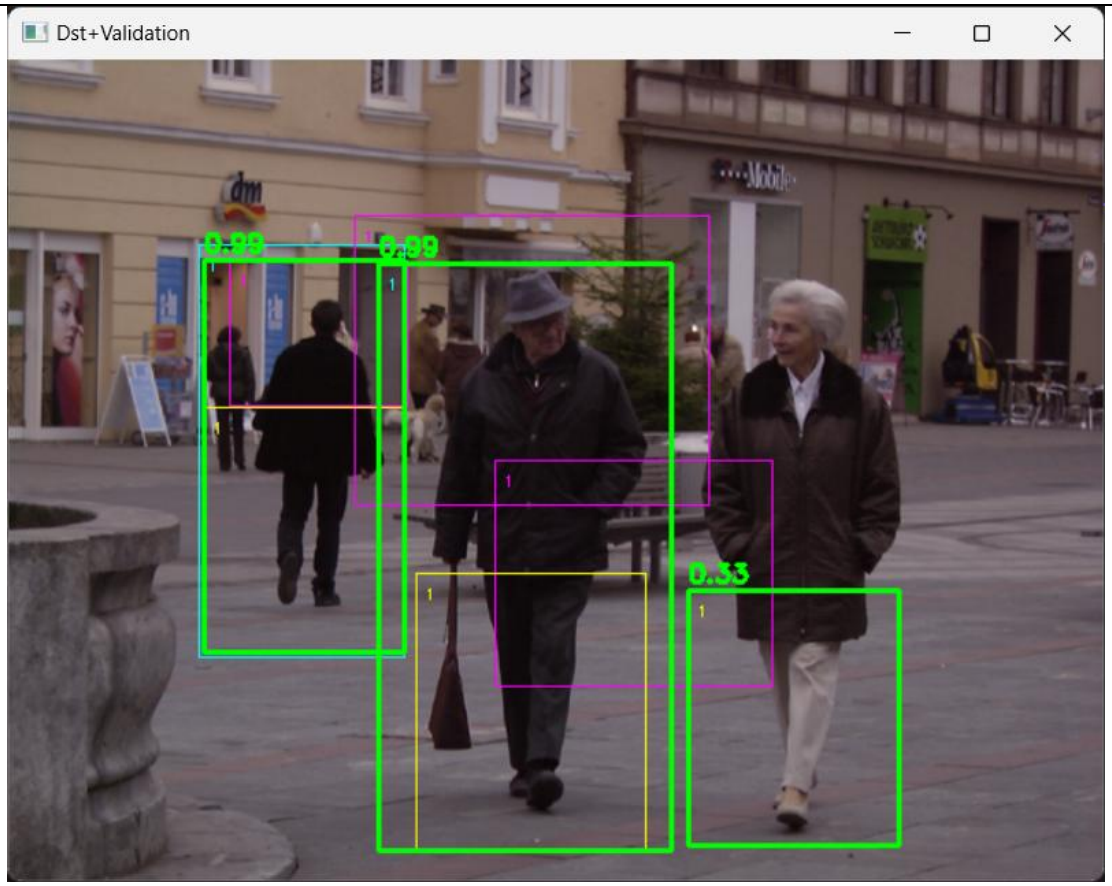
Implementați cerințele 1-3 într-o singură funcție de procesare.

Rezultatul final al validării puteți sa îl desenați peste imaginea color în care afișați rezultatele detecțiilor sub forma unui chenare verzi și a scorurilor de confidența asociat (utilizați funcția *putText*).

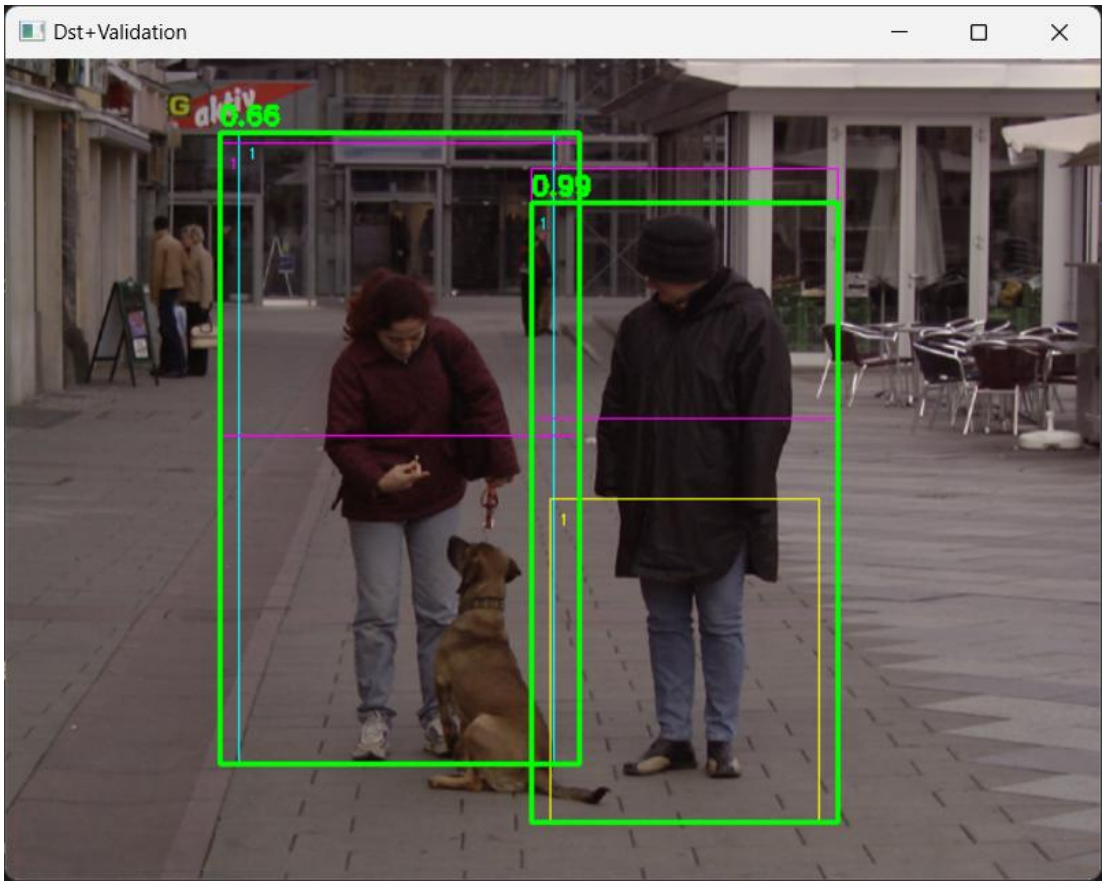
Exemplele cu persoane (imaginile de test) sunt selectate din setul de date: http://www.emt.tugraz.at/~pinz/data/GRAZ_01/) și se găsesc la locația: http://users.utcluj.ro/~tmarita/HCI/Media/Images/Persons.zip sau pe Team-ul disciplinei. Pentru a testa corectitudinea implementării este recomandat sa lucrați atât pe imagini cu o persoana (011, 090, 096) cât si pe imagini cu persoane multiple (138, 230,238)



a. Rezultatul validării pentru imaginea person_133.bmp



b. Rezultatul validării pentru imaginea person_230.bmp



c. Rezultatul validării pentru imaginea person_238.bmp

Fig. 10.3. Rezultatele detecțiilor cu validare pe imagini cu persoane multiple.

10.3. Anexe

1. Calculul intersecției dintre două dreptunghiuri
OpenCV (Rect): http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#rect

```
rect = rect1 & rect2  //rectangle intersection
```

2. Calculul reuniunii dintre două dreptunghiuri (dreptunghiul de arie minima care conține cele doua dreptunghiuri componente)
OpenCV (Rect): http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#rect

```
rect = rect1 | rect2  //rectangle union
```

3. Calcul centru dreptunghi – funcție definită in modulul *Functions*

```
Point RectCenter(Rect R)
{
	Point P;
	P.x = R.x + R.width / 2;
	P.y = R.y + R.height / 2;
	return P;
}
```

4. Calcul arie dreptunghi – funcție definită in modulul *Functions*

```
int RectArea(Rect R)
{
	return R.width*R.height;
}
```

11. Detecția de persoane folosind trăsături Haar și clustering

Scop: Scopul acestei lucrări este de a implementa o metodă de detecție de persoane bazată pe detecțiile clasificatorului HAAR de parți corporale și rafinarea detecției prin gruparea parților corporale care aparțin aceleiași persoane prin aplicarea unui algoritm de grupare (clustering) pe centrele regiunilor rectangulare corespunzătoare părților corporale detectate. Abordarea curentă își propune înlocuirea sistemului de reguli bazate pe trăsăturile antopomorfice folosite în validarea detecțiilor din lucrarea L11 cu algoritmul de grupare/clustering Modified Basic Sequential Algorithmic Scheme (MBSAS)[1].

Modelele clasificatorilor cascadă bazați pe trăsături Haar pentru detecția de părți corporale care se vor folosi sunt disponibile la locația: *%OPENCV_DIR%\data\haarcascades* din instalarea OpenCV și sunt următoarele:

haarcascade_fullbody.xml
haarcascade_lowerbody.xml
haarcascade_upperbody.xml

11.1. Considerații teoretice

Metodele de clustering sunt metode destul de simple și rapide folosite în gruparea vectorilor de trăsături. În majoritatea lor, toți vectorii de caracteristici sunt prezentați algoritmului o dată sau de câteva ori. Rezultatul final depinde, de obicei, de ordinea în care vectorii sunt prezentați algoritmului. Aceste scheme tind să producă clustere compacte și de formă hiper-sferică sau hiper-elipsoidală, în funcție de metrica de distanță utilizată.

Vom considera $d(X,C)$ distanță (sau disimilaritatea) dintre un vector de trăsături X și un cluster C. Pentru un cluster se ia de obicei un vector reprezentativ al acestuia, care poate fi centrul de greutate / vectorul mediu m_C al tuturor vectorilor de trăsături care intră în componența clusterului: $d(X,C) = d(X,m_c)$. Parametrii definiți de utilizator, necesari în schema algoritmică, sunt:

- metrica de distanță d folosită pentru calculul disimilaritatăţii
- pragul de disimilaritate θ
- numărul maxim admis de clustere q (opțional)

Dacă numărul maxim admis de clustere q nu este constrâns putem lăsa algoritmul să „decidă” cu privire la numărul adecvat de clustere. Constrângerea lui q devine necesară atunci când se lucrează cu implementări în care resursele de calcul disponibile sunt limitate. În lucrarea de față se va implementa o abordare în care valoarea q se va specifica de utilizator (ex. numărul maxim de peroane prezente în scenă).

Vectorul mediu al clusterului se poate calcula iterativ pe măsura ce se adaugă un nou vector de trăsături la cluster, folosind formula mediei ponderate:

$$m_{C_k}^{new} = \frac{n_{C_k^{old}} * m_{C_k}^{old} + X}{n_{C_k^{old}} + 1}$$

Unde:

- $n_{C_k^{old}}$ - este cardinalul clusterului K înainte de adăugarea vectorului de trăsături X
- $m_{C_k}^{new}$ ($m_{C_k}^{old}$) – este valoarea vectorului mediu al clusterului K după (înaintea) adăugării vectorului de trăsături X
- metrica de distanță d poate fi o distanță de tip L1 (city-block) sau L2 (euclidiană).

Algoritmul MSBAS[1] poate fi rezumat prin următorul pseudocod:

Faza 1: Determinarea Clusterelor

1.1 • $m = 1$
1.2 • $C_m = \{X_1\}$
1.3 • ***For*** $i = 1$ ***to*** *N-1*
1.4 – Găsește C_k astfel încât $d(X_i,C_k) = \min_{1\le j\le m} d(X_i,C_j)$
1.5 – ***If*** $(d(X_i,C_k) > \theta)$ ***AND*** $(m < q)$ ***then***
1.6 * $m = m + 1$
1.7 * $C_m = \{Xi\}$
1.8 – ***End***
1.9 • ***End***

Faza 2: Clasificarea Șabloanelor (vectorilor de trăsături)

2.1 • ***For*** $i = 1$ ***to*** *N-1*
2.2 – ***If*** X_i nu a fost atribuit la un cluster, ***then***
2.3 * Găsește C_k astfel încât $d(X_i,C_k) = \min_{1\le j\le m} d(X_i,C_j)$
2.4 * $C_k = C_k \, U \, \{X_i\}$
2.5 * Actualizează vectorul mediu
2.6 – ***End***
2.7 • ***End***

11.2. Detalii de implementare

Datele de intrare pentru algoritmul de clustering vor fi extrase din vectorii ce conțin regiunile de interes (ROI dreptunghiulare) detectate pentru fiecare parte corporală prin aplicarea metodei `detectMultiScale`, succesiv, pentru fiecare dintre cele 3 modele de clasificare (*fullbody, lowerbody, upperbody* - vezi L10). Sugerăm ca datele de intre pentru algoritmul de clustering să fie chiar mulțimea centrelor tuturor dreptunghiurilor care încadrează părțile corporale detectate, pe care le putem stoca într-un vector cu elemente individuale având structura de mai jos:

```
typedef struct {
	Point center; // coordonatele punctului central al unei parti corporale
	byte clasa; // 1 - fbody, 2 - ubody, 4 - lbody
	short cluster; // clusterul la care este atribuit; -1 = neatribuit
}element; // strcuctura pentru memorarea datelor de intrare pentru clustering
vector <element> X; // setul de date de intrare pentru clustering
```

Pe lângă coordonatele centrului părții corporale (*center.x, center.y*) care constitui și vectorul de trăsături principal care caracterizează datele de intrare pentru algoritmul de clustering ar fi util să memorăm în vectorul de intrare *X* si *clasa* părții corporale (1 - fullbody, 2 - upperbody, 4 – lowerbody; s-au considerat codificările clasei ca si puteri ale lui 2 pentru a putea calcula mai ușor scorul de confidență al detecției persoanei) și *cluster*-ul căruia îi va fi atribuit vectorul de trăsături.

Pentru construirea vectorului *X* parcurgeți cei trei vectori care conțin cele 3 parți corporale detectate (vezi L10) și adăugați centrele acestora în vectorul X. Simultan puteți realiza și afișarea acestora (Fig. 11.1):

```
	dx_cross = 10;
	//parcurge vectorul fullbodies
	for (int i = 0; i < fbodies.size(); i++)
	{
		Point centru = RectCenter(fbodies[i]);
		X.push_back({ centru,1,-1 });
		Scalar color = Scalar(255, 255, 0); //cyan
		rectangle(frame, fbodies[i], color, 1, 8, 0);
		DrawCross(frame, centru, dx_cross, color, 1);
```

```
        }
        ......
```

Pentru memorarea clusterelor detectate am avea nevoie de coordonatele vectorului mediu (centroidul) clusterului *(Xc,Yc)* și a cardinalului acestuia (numărul centrelor părților corporale care sunt atribuite clusterului respectiv) ca și în exemplul de mai jos. Ca și observație, primul cluster va avea indicele 0 (*Clusters[0]*), iar ultimul va avea indicele *Clusters.size()-1*.

```
typedef struct {
        int Xc, Yc; // coordonatele centroidului clusterului
        int Nc; // cardinalul clusterului
}cluster; // strcuctura pentru memorarea datelor aferente unui cluster
vector <cluster> Clusters; // setul de date de intrare pentru clustering
```

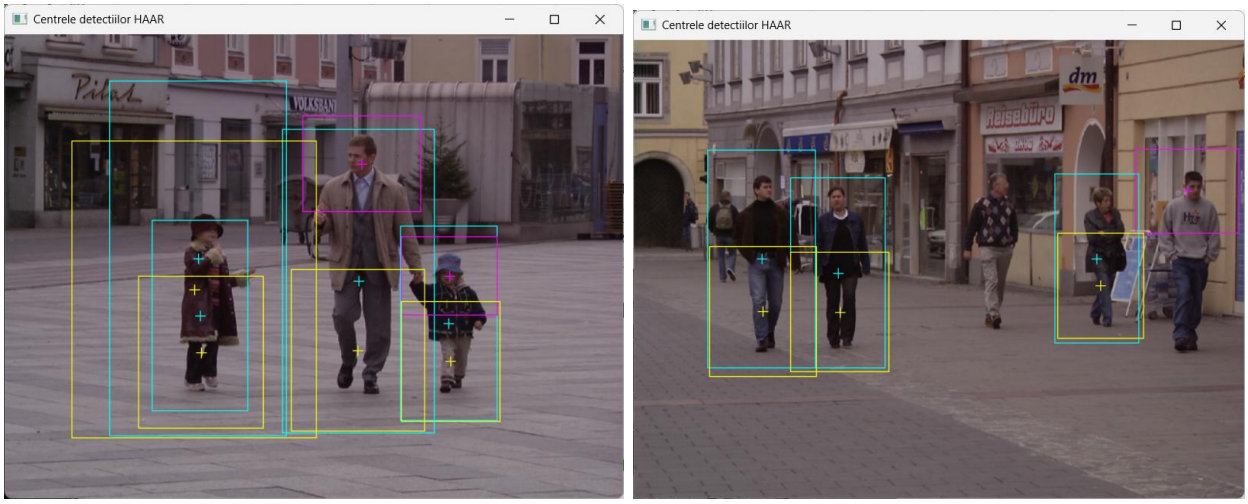


Fig. 11.1. Ilustrarea centrelor părților corporale detectate care vor fi datele de intrare pentru algoritmul de clustering.

Primul cluster se va inițializa cu primul centru al primei părți corporale detectate. Acest cluster va avea indicele 0 in vectorul *Clusters*:

```
//creeaza primul cluster cu primul element din setul de date (centrele partilor corp.)
Clusters.push_back({ X[0].center.x, X[0].center.y, 1 });
int m = 0; // indicele clusterului curent; numarul curent de clustere = m+1
X[0].cluster = m;
```

Pentru aplicarea algoritmului de clustering la problema actuală (forma clusterelor trebuie să fie mai alungită pe verticală conform trăsăturilor antropomorfice ale unei persoane) se va adopta a varianta a metricii L1 descompusă pe cele doua direcții ortogonale ale sistemului de coordonate al imaginii (x,y) prin distanțele d_x și d_y, și se vor folosi două praguri θ_x și θ_y. Metrica descompusă pe cele 2 direcții ortogonale ale coordonatelor imaginii va fi implementă prin aplicarea succesivă a două instrucțiuni if-else.

Faza 1: Determinarea Clusterelor – adaptată la structura antropomorfica a unei persoane

1.1 • $m = 0$
1.2 • $C_m = \{X_0\}$
1.3 • ***For*** $i = 1$ ***to*** *X.size()-1*
1.4 – Găsește C_k astfel încât $d_x(X_i, C_k) = \min_{1 \le j \le m} d_x(X_i, C_j)$,
1.5 – Calculează $d_y(X_i, C_k)$ aferent
1.6 – ***If*** $(d_x(X_i, C_k) > \theta_x)$ ***AND*** $(m < q)$ ***then***
1.7 * $m = m + 1$
1.8 * $C_m = \{X_i\}$

1.9 – ***ElseIf*** $(d_y(X_i,C_k) > \theta_y)$ ***AND*** $(m < q)$ ***then***
(daca X_i si C_k sunt aliniate pe orizontală dar distanța pe verticala > prag)
1.10 * $m = m + 1$
1.11 * $C_m = \{X_i\}$
1.12 – ***End***
1.13 • ***End*** {For}

Dacă se folosește ca și parametru de intrare pentru detecția părților corporale (prin funcția *detectMultiScale*) constanta *minBodyHeight* (care se poate considera cca. 150 pentru imaginile din setul de date *Persons* – vezi L10), se pot seta următoarele valori experimentale pentru θ_x si θ_y : θ_x = *minBodyHeight*0.5* şi θ_y = *minBodyHeight*3*.

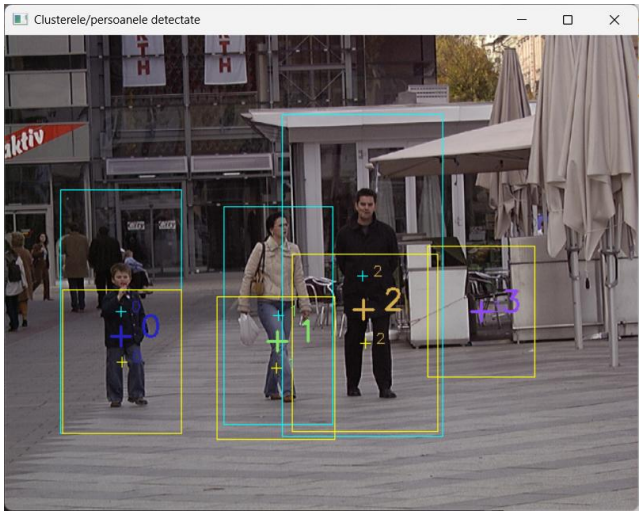
Faza 2: Clasificarea Șabloanelor – adaptată la structura antropomorfica a unei persoane

2.1 • ***For*** $i = 1$ ***to*** *X.size()-1*
2.2 – ***If*** X_i nu a fost atribuit la un cluster, ***then***
2.3 * Găsește C_k astfel încât $d_x(X_i,C_k) = \min_{1\le j\le m} d_x(X_i,C_j)$
2.4 * Calculează $d_y(X_i,C_k)$ aferent
2.5 * ***If*** $(d_x(X_i,C_k) < \theta_x)$ **AND** $(d_y(X_i,C_k) < \theta_y)$ **then**
2.6 - $C_k = C_k \, U \, \{X_i\}$
2.7 - Actualizează vectorul mediu
2.8 – ***End***
2.9 • ***End***

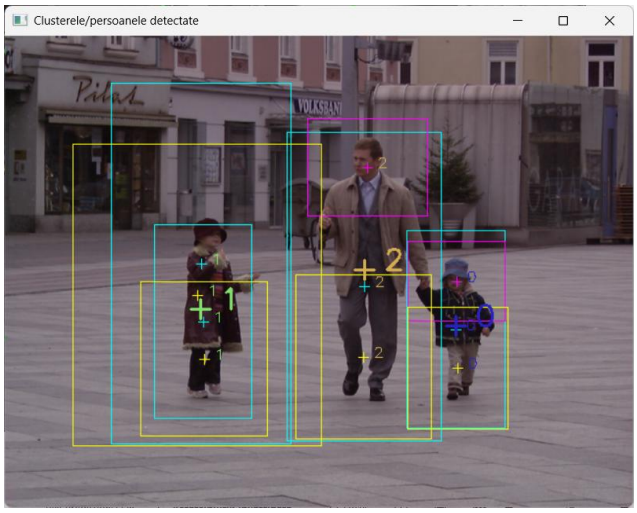
Pentru verificarea vizuală a corectitudinii implementării, se poate genera o paleta de culori cu valori aleatorii (vezi L5 din Procesarea Imaginilor - Îndrumătorul de laborator[2[) și se poate afișa peste imaginea destinație centroidul (vectorul mediu) al fiecărui cluster printr-o cruce cu o astfel de culoare. De asemenea se pot marca centrele părților corporale (*X[i].centru*) cu numărul clusterului de care aparțin tot în culoarea aleatorie specifică clusterului (vezi secvențele de cod de mai jos și exemplele din figura 11.2).

```
//generare paleta de culori
Scalar* colorLUT = new Scalar[Clusters.size()];
Scalar color;
for (int i = 0; i < Clusters.size(); i++)
{
	Scalar color(rand() & 255, rand() & 255, rand() & 255);
	colorLUT[i] = color;
}
// Deseneaza centrele partilor corporale
// in culoarea aleatorie asociata fiecrui cluster
dx_cross = 10;
for (int i = 0; i < X.size(); i++)
{
	if (X[i].cluster >= 0)
	{
	char msg[2];
	sprintf(msg, "%d", X[i].cluster);
	// deseneaza numarul clasei de care apartine centrul partii corporale
	putText(frame, msg, Point(X[i].center.x + dx_cross, X[i].center.y + 0),
FONT_HERSHEY_SIMPLEX, 0.5, colorLUT[X[i].cluster], 1, 8);
	}
}
// Deseneaza centroidul clusterului in culoarea aleatorie asociata lui
dx_cross = 20;
for (int j = 0; j < Clusters.size(); j++)
{
```

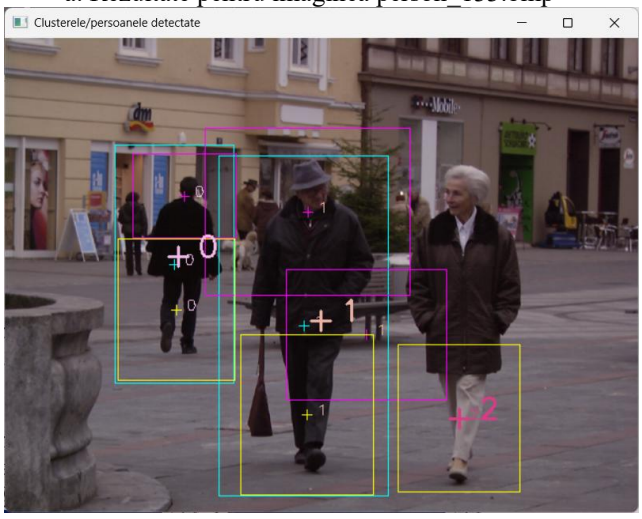
```
        // Deseneaza cu o cruce centroidul clasei si o coloreaza cu culoarea asociata
        DrawCross(frame,Point(Clusters[j].Xc,Clusters[j].Yc), dx_cross, colorLUT[j], 2);
        // Deseneaza si numarul clasei alaturi
        char msg[2];
        sprintf(msg, "%d", j);
        putText(frame, msg, Point(Clusters[j].Xc + dx_cross, Clusters[j].Yc + 0),
FONT_HERSHEY_SIMPLEX, 1, colorLUT[j], 2, 8);
}
```



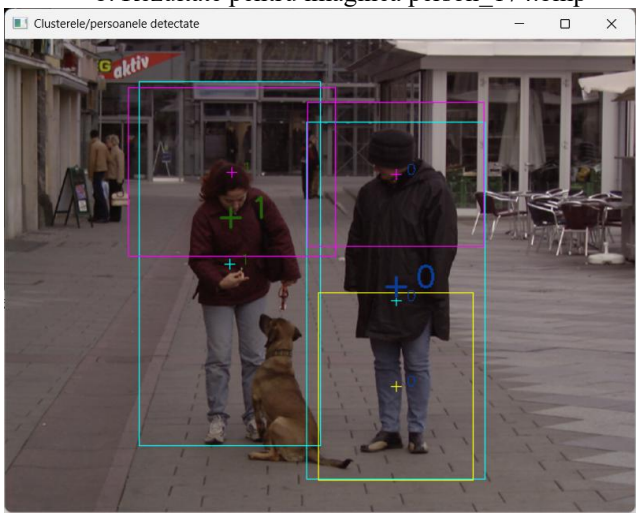
a. Rezultate pentru imaginea person_133.bmp



b. Rezultate pentru imaginea person_174.bmp



c. Rezultate pentru imaginea person_230.bmp



d. Rezultate pentru imaginea person_238.bmp

Fig. 11.2. Rezultatele detecțiilor pe imagini cu persoane multiple. Atât centrul clusterului cât și centrele părților corporale care îi aparțin sunt marcate cu numărul clusterului de care aparțin în culoarea aleatorie specifica clusterului.

11.3. Activități practice

1. Implementați algoritmul de clustering în conformitate cu indicațiile din capitolele 11.1 și 11.2. Afișați rezultatele conform indicațiilor sau puteți implementa orice altă variantă care să poată evidenția vizual gruparea centrelor părților corporale detectate în clustere. Exemplele ilustrate (imaginile de test) sunt selectate din setul de date: http://www.emt.tugraz.at/~pinz/data/GRAZ_01/) și se găsesc la locația: http://users.utcluj.ro/~tmarita/HCI/Media/Images/Persons.zip sau pe Team-ul disciplinei (MS-Teams).
2. Implementați o metoda de calculare și afișare a gradului de confidență (CF) pentru fiecare cluster/persoană detectată folosind o codificare similară cu cea din lucrarea L10:
   - CF = 0.33 dacă dintr-un cluster face parte o singur tip de parte corporală;
   - CF = 0.66 dacă dintr-un cluster fac parte două tipuri diferite de părți corporale

- CF = 0.99 dacă dintr-un cluster fac parte trei tipuri diferite de părți corporale

Pentru a implementa metoda, mai adăugați un atribut *byte CF_voting* (confidence factor voting) la structura *cluster* și urmați următorii pași:.

- Parcurgeți vectorul *X* (care conține câmpul părții corporale (*clasa*) inițializat la construirea lui cu valori posibile: 1 – fullbody (0000 0001binar); 2 – uppbody (0000 0010 binar); 4 – lowerbody (0000 0100 binar) si aplicați următoarea schema de votare:

```
short cluster_curent = X[i].cluster;
Clusters[cluster_curent].CF_voting |= X[i].clasa;
```

- Definiți un LUT pentru conversia câmpului CF_voting în valoare CF (factor de confidență):

```
// Definire LUT pentri conversia valoare_votare-> CF
float votingLUT[8] = { 0.0f,  // 0 - nedefinit
                       0.33f, // 1 = 001b
                       0.33f, // 2 = 010b
                       0.66f, // 3 = 011b
                       0.33f, // 4 = 100b
                       0.66f, // 5 = 101b
                       0.66f, // 6 = 110b
                       0.99f  // 7 = 111b
};
```

- La parcurgerea vectorul *Clusters* pentru desenarea centrului și a numărului fiecărui cluster, afișați alături de numărul clusterului și scorul de confidență folosind *voting_LUT*:

```
uchar voting_result = Clusters[j].CF_voting;
sprintf(msg, "%d-%.2f", j, votingLUT[voting_result]);
putText(frame, msg, Point(Clusters[j].Xc + dx_cross, Clusters[j].Yc + 0),
     FONT_HERSHEY_SIMPLEX, 0.5, colorLUT[j], 2, 8);
```

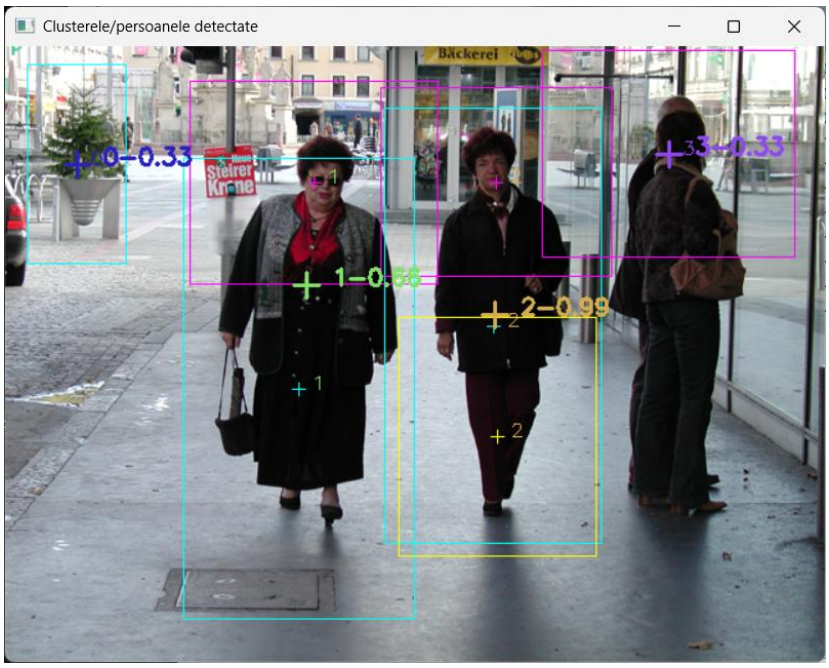


Fig. 11.3. Rezultatele detecțiilor clusterelor si a factorului de confidență CF pentru imaginea Person_138.

11.4. Bibliografie

[1] S. Theodoridis, K. Koutroumbas, Pattern recognition 2-md edition, Academic Press, 2003

[2] S. Nedevschi, T. Marita, R. Danescu, F. Oniga, R. Brehar, I. Giosan, C. Vancea, R. Varga, Procesarea Imaginilor - Îndrumător de laborator, ediția a 2-a, Editura U.T. Press, Cluj-Napoca, https://biblioteca.utcluj.ro/carti-online-cu-coperta.html, 2023.

12. Detecția de persoane folosind trăsături HOG

Scop: Scopul acestei lucrări este de a implementa o metodă de detecție a persoanelor bazată pe trăsături HOG (Histogram of Oriented Gradients) și de a îmbunătăți rezultatele detecției prin ajustarea regiunii de interes (ROI) astfel încât acestea să delimiteze cât mai precis marginea de jos a ROI la nivelul picioarelor. O astfel de abordare ar putea fi folosită în aplicații în care, cunoscând parametrii camerei cu care se face achiziția imaginii, s-ar putea estima poziția persoanei conform modelului matematic prezentat în cursul de Procesarea Imaginilor [1].

12.1. Considerații teoretice

Metoda de detecție a persoanelor bazată pe trăsături HOG (Histogram of Oriented Gradients) a fost propusă de Dalal și Triggs [2]. Metoda presupune parcurgerea imaginii de intrare cu ferestre glisante cu ajutorul cărora se decupează din imagine regiunii de interes (ROI) cu o proporție fixă între lățime și înălțime de 1:2. Deoarece nu se poate ști dinainte dimensiunea persoanelor din imagine, scanarea imagini cu aceste ferestre glisante se va face la rezoluții diferite (multi-rezoluție).

Fiecare ROI curentă se scalează la o dimensiune fixa de 64x128 pixeli și se convertește în grayscale (fig. 12.1). În fiecare ROI se calculează componentele orizontale și verticale ale gradientului (folosind de exemplu filtrele Sobel – ec. 12.1) și din acestea se calculează magnitudinea (12.2) si orientarea (12.3) gradientului în fiecare pixel al ROI. Valorile orientării gradientului sunt scalate la valori întregi și pozitive in intervalul 0 .. 180°, deoarece s-a demonstrat empiric că gradienții fără semn funcționează mai bine decât gradienții cu semn pentru detectarea persoanelor.

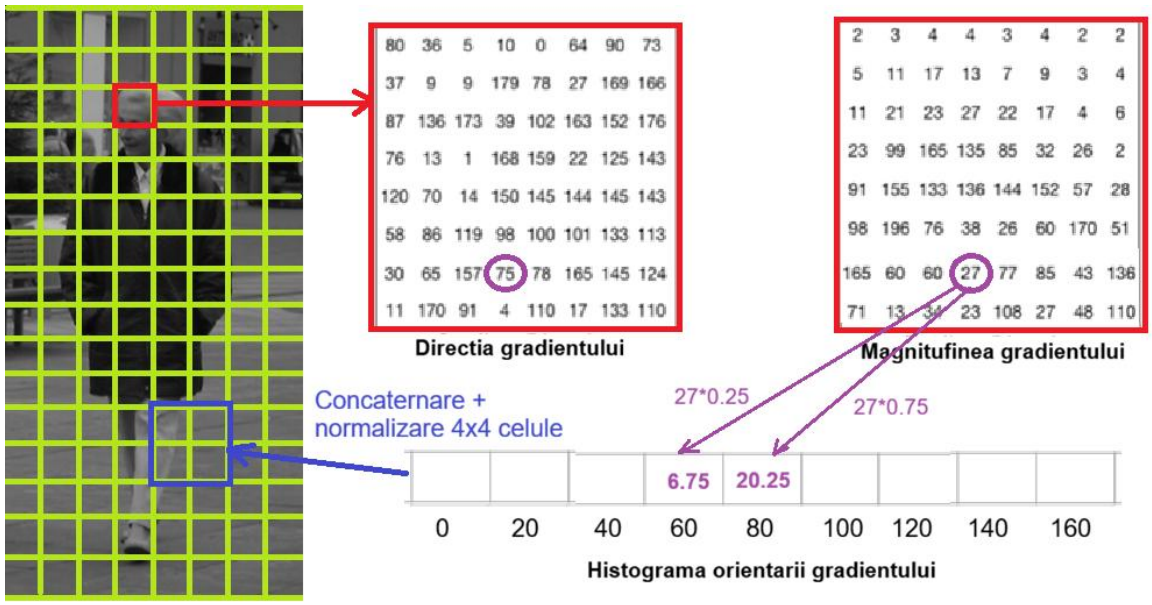
Fig. 12.1. Ilustrare a procesului de extragerea regiunilor de interes ROI cu o fereastră glisantă, scalarea și conversia acestora în grayscale.

$$\nabla f_x = f(x,y) * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
$$\nabla f_y = f(x,y) * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (12.1)$$

$$|\nabla f(x,y)| = \sqrt{(\nabla f_x(x,y))^2 + (\nabla f_y(x,y))^2} \tag{12.2}$$

$$\theta(x,y) = arctg\left(\frac{\nabla f_y(x,y)}{\nabla f_x(x,y)}\right) \tag{12.3}$$

Regiunile de interes ROI scalate la 64x128 pixeli se împart mai departe în celule de 8x8 pixeli (fig. 12.2.a) rezultând un umăr de 8x16 celule. În fiecare celulă se calculează câte o histogramă a orientărilor gradientului, cuantizată la 9 direcții principale (0, 20, 40, 60, 80, 100, 120, 140, 160 grade) folosind o schemă de votare ponderata care ține seama de amplitudinea gradientului și de distanța orientării gradientului față de cele mai apropiate direcții principale ale histogramei [3]. De exemplu, dacă într-un pixel direcția gradientului este o valoare multiplu de 20 grade se va vota/adăuga în histogramă la direcția respectiva întreaga valoare a magnitudinii gradientului. Dacă într-un pixel cu magnitudinea gradientului *m*, direcția gradientului este un număr *d* situat între 2 direcții principale d_i și d_{i+1} ($d_{i+1}=d_i+20°$), atunci se va aduna în histogramă la direcția d_i valoarea $m*(d_{i+1}-d)/20$, iar la direcția d_{i+1} se va aduna valoarea $m*(d-d_i)/20$, ca și in exemplul din figura 12.2.b. Pentru direcțiile gradientului între 160 si 179 de grade se va ține seama ca 180 grade este echivalent cu 0 grade și se va vota ponderat între direcțiile 160 si 0 grade.



a. b.

Fig. 12.2. a. Împărțirea ROI în celule de dimensiune 8x8; b. Ilustrarea schemei de votare folosite în construirea histogramei orientărilor gradientului în fiecare celulă de 8x8 pixeli.

Deoarece gradienții imaginii sunt sensibili la variațiile de iluminare ale scenei, se aplică o schemă de normalizare suplimentara astfel:

- Se parcurge imaginea cu o fereastră glisantă de dimensiune 2x2 celule (16 x 16 pixeli) (vezi fereastra albastră din figura 12.2.a).
- Pentru fiecare fereastră/bloc de 16x16 pixeli (2x2 celule) se concatenează cele patru histograme ale orientărilor gradientului calculate în fiecare celulă componentă, rezultând un vector cu 4x9=36 de elemente. Valorile acestui vector se normalizează împărțind fiecare element cu norma L2 a vectorului.
- Se deplasează fereastra de 2x2 blocuri cu o celulă spre stânga (8 pixeli) și se calculează un nou vector normalizat, iar când se ajunge la capătul din dreapta al ROI se repeta procesul cu o celula (8 pixeli mai jos) rezultând în final un număr de 7x15 poziții ale ferestrei glisante de

dimensiune 2x2 celule (16x16 pixeli), în fiecare dintre acestea fiind calculat un vector de 36 elemente/valori.

Rezultă astfel un vector de trăsături (descriptor HOG) de dimensiune 36x7x15=3780 elemente, care poate fi folosit pentru antrenarea unui model declasificare al persoanelor. In [2] autorii au optat pentru antrenarea unui clasificator SVM liniar, model care va fi folosit și în lucrarea curentă prin intermediul implementării clasei `HOGDescriptor` din OpenCV[4].

12.2. Utilizarea implementării din OpenCV a metodei de detecție a persoanelor bazată pe trăsături HOG

Șablonul de procesare pentru detecția persoanelor și afișarea rezultatelor este prezentat mai jos:

```
/* ----------------------------------------------------------------------------
window_name - name of the detsination window in which the detection results are
displayed
frame - source image (1 channel grayscale or 3 channel color image
---------------------------------------------------------------------------- */
void BodyDetectandDisplayHOG(const string& window_name, Mat frame)
{
	Scalar color_green(0, 255, 0);
	Scalar color_cyan(255, 255, 0);

	// Initializarea clasei HOGDescriptor
	HOGDescriptor hog;
	hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

	// Detectia persoanelor în imagine
	vector<Rect> persons;    // vector cu regiunile aferente persoanelor detectate
	vector<double> weights; // scor de confidenta (mai mare = detective mai buna)
	hog.detectMultiScale(frame, persons, weights);

	// Afisarea rezultatelor detectiei si a scorului de confidenta asociat
	for (int i = 0; i < persons.size(); i++)
	{
		rectangle(frame, persons[i], color_cyan, 1, 8, 0);
		char msg[10];
		sprintf(msg, "%.3f", weights[i]);
		putText(frame, msg, Point(persons[i].x + 5, persons[i].y + 15),
				FONT_HERSHEY_SIMPLEX, 0.5, color_red, 2, 8);
	}
}
```

În șablon se instanțială un obiect `hog` din clasa `HOGDescriptor` și se setează modelul de clasificare folosit: SVM pentru detecția de persoane. Apoi se apelează metoda `detectMultiScale` care furnizează un rezultat similar ca și la detecția de fețe sau de persoane cu trăsături HAAR, rezultând la ieșire un vector de regiuni rectangulare corespunzătoare detectaților pozitive însoțit de un vector ce conține scorurile de confidență pentru fiecare detecție.

12.3. Activități practice

11. Se va crea o funcție de procesare în care se va insera șablonul de procesare de mai sus.
12. Implementați o logică de fuziune a suprapunerilor cvasi-totale dintre detecții pe baza relației:
    If *aria(persons[i] ∩ persons [j]) / min(aria(persons [i], arie(persons [j]) > 0.85 AND (persons [i], persons [j]) sunt aproximativ aliniate pe accesași fâșie verticală* (dacă cele doua boxuri sunt cvasi-suprapuse și diferența în valoare absolută pe orizontală a coordonatelor *x* ale centrelor lor este mai mică decât lățimea detecției cu arie minimă)

Then *păstrează detecția cu aria cea mai mare si elimină detecția cu aria cea mai mica* (se recomandă să folosiți un vector `validPerson` de lungime `persons.size()` inițializat cu valori de 1 iar pentru detecțiile care nu sunt valide în urma acestui test sa schimbați valoarea din vectorul `validPersons` din 1 in 0).

unde: indicii *i* și *j* sunt indicii cu care se parcurge vectorul `persons` într-un for dublu pentru a putea compara detecțiile două cate două (evitați sa comparați de doua ori aceeași pereche: *[i,j] ~ [j,i]* sau detecții cu același indice *[i=j]*).

Fig. 12.3. Cazuri de detecții multiple pentru aceeași persoană.

Fig. 12.4. Rezolvarea cazurilor de detecție multiplă prin fuziunea detecțiilor conform metodei propuse la pasul 2.

3. Pentru fiecare persoană detectată extrageți câte o sub-imagine aferenta ROI (Region Of Interest) validate și ajustați marginea inferioară a ROI astfel încât sa delimitați cât mai precis marginea de jos (tangenta cu picioarele), folosindu-vă de cunoștințele acumulate la materiile de Procesarea Imaginilor ș Interacțiune Om-Calculator

   Exemplu:
   - conversie ROI în greyscale
   - aplicare filtru gaussian și eventual egalizare histogramă
   - binarizarea ROI cu prag adaptiv calculat automat astfel încât pixelii care aparțin persoanelor sa fie albi (obiectele de interes sa fie albe)
   - aplicare operații morfologice pentru eliminarea erorilor de segmentare
   - calculul proiecției orizontale și parcurgerea vectorului proiecție orizontală (de la dreapta la stânga) și găsirea primei valori diferite de 0 care ar trebui să indice limita de jos a picioarelor în imagine

   La nevoie afișați și rezultatele intermediare aferente etapelor de procesare pentru a putea depana eventualele erori

a. b.

Fig. 12.5. Exemplu cu rezultate intermediare al procesărilor efectuate pentru corecția limitei de jos a detecției (dreptunghiului care încadrează persoana): a. Regiunea aferenta persoanei după binarizare si filtrare morfologică; b. Reprezentare vizuala a vectorului proiecției orizontale.

4. Redesenați peste imaginea originală regiunile ajustate care corespund detecțiilor corectate.

Fig. 12.6. Ajustarea limitei inferioare a detecțiilor prin conversia ROI în grayscale, binarizare cu prag automat, filtrare morfologică și ajustarea limitei inferioare a detecțiilor prin analiza proiecției orizontale.

12.4. Bibliografie

[1] T. Marița, Procesarea Imaginilor, note de curs, https://users.utcluj.ro/~tmarita/IPL/IPCurs/IPCurs.htm

[2] Dalal, N., Triggs, B. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Vol. 1, pp. 886-893.

[3] Satya Mallick, Histogram of Oriented Gradients explained using OpenCV, Learn OpenCV, 6 Decembrie, 2016, Histogram of Oriented Gradients explained using OpenCV.

[4] OpenCV, Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. https://docs.opencv.org/4.9.0/d5/d33/structcv_1_1HOGDescriptor.html