7. Analiza miscarii pe baza fluxului optic dens

Scop: Scopul acestei lucrari este de a integra o metoda de detectie a fluxului optic dens si de analiza a miscarii relative dintre camera si scena pe baza vectorilor de flux optic. Pentru aceasta se va folosi metoda propusa de Gunnar Farneback [1] si se va vizualiza fluxul optic dens al vectorilor de miscare folosind codificarea Middleburry [2]. Pentru vectorii de miscare se va construi histograma directiilor lor si se va face analiza formei acestei histograme, inferandu-se miscarea relativa dintre camera si scena sau dintre camera si obiectele din scena (cand camera este fixa).

7.1. Estimarea fluxului optic dens

Pentru estimarea fluxului optic dens se va integra metoda propusa de Gunnar Farneback [1], implementata in OpenCV prin functia: `calcOpticalFlowFarneback`. Apelul functiei se realizeaza in felul urmator:

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

Unde:

Mat *prev, crnt* – imaginea trecuta si curenta (grayscale, 8 biti /pixel)
Mat *flow* – matrice de aceeasi dimensiune cu imaginile dar de tip CV_32FC2 – contine vectorii de miscare
pyr_scale – factorul de scalare (0...1) folosit in constructia piramidelor de imagini (ex: 0.5 – injumatatire a rezolutiei intre 2 nivele succesive)
levels – numarul de nivele din piramida (*levels* = 1 – nu se folosesc piramide de imagini, valoare recomandata 3).
winsize – dimensiunea ferestrei de mediere: valori ridicate maresc robustetea la zgomot si permit detectia vectorilor de miscare de amplitudine mai mare dar campul de miscare rezultat va avea un aspect netezit (blurred)
iterations – numarul de iteratii ale algoritmului pt. fiecare nivel al piramidei (o valoare ridicata va duce la cresterea semnificativa a timpului de procesare!)
poly_n – dimensiunea vecinatatii in care se cauta expansiunea polinomiala pentru fiecare pixel (pt. valori mari imaginea va fi aproximata cu suprafete mai netede → algoritm de estimare mai robust si un camp de miscare mai netezit (blurred)); valori tipice = 5 ... 7.
poly_sigma – deviatia standard gaussianului cu care se filtreaza derivatele imagini folosite ca si baza a expansiunii polinomiale (valori tipice: *poly_n=5 / poly_sigma=1.1* sau *poly_n=7 / poly_sigma=1.5*
flags – OPTFLOW_USE_INITIAL_FLOW sau OPTFLOW_FARNEBACK_GAUSSIAN. Daca se seteaza primul flag se foloseste valoarea din *flow* ca aproximare initiala a fluxului iar daca se seteaza cel de-al doilea se foloseste un Gaussian de dimensiune *winsize x winsize* in locul unui box-filter pentru estimarea fluxului optic (folosind gaussianul se obtin 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
```

Functia calcOpticalFlowFarneback estimeaza fluxul optic pentru fiecare pixel din imaginea trecuta (prev):

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

In consecinta vectorii din *flow* au orientarea dinspre pixelul din imaginea curenta inspre corespondentul lui din imaginea trecuta. Daca se doreste reprezentarea lor in sens invers trebuie considerata valoarea – *flow(y,x)* !!!

Sablonul de procesare pentru bucla principala (in care se sitesc succesiv imaginile din secventa de bitmaps) ar trebui sa arate asa (identic ca si 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
	. . .
	}

	// 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. Afisarea hartii dense a fluxului optic folosind codificarea de culoare Middleburry [2]

Pentru afisarea hartii dense a fluxului optic folosind codificarea de culoare Middleburry aceasta aveti nevoie de modulul *colorcode* (inclus deja in proiect: *colorcode.h* si *colorcode.cpp*).

Pentru afisarea hartii dense a fluxului optic in fereastra destinatie puteti folosi functia `showFlowDense` (vezi ANEXA) integrata in modulul cu functii ajutatoare (*Functions h*, *Functions.cpp*) din proiect. Apelul functiei `showFlowDense` se va face dupa apelul functiei de calcul a fluxului optic!

Functia *showFlowDense* afiseaza automat rezultatul in fereastra specificata ca si prim parametru (`const string& name`). Nu este nevoie sa adaugati in bucla de procesarea un apel explicit pt. afisarea ferestrei destinatie (de ex. `imshow("dst", dst)`).

Important: Functia *showFlowDense* apeleaza functia *computeColor* care foloseste un LookUp Table (LUT) in care sunt codificate culorile de afisat si care trebuie initializat. Initializarea trebuie facuta doar o singura data in aplicatie prin apelul `makeColorwheel()`. De asemena adaugati apelul functiei `make_HSI2RGB_LUT` necesara pentru afisarea histogramei directiilor de miscare in codul de culorifolosit. Inserati cele doua apeluri la inceputul functiei de procesare aferente laboratorului curent.

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

Prin aceasta codificare fluxul optic dens in fiecare punct este reprezentat printr-un pixel a carui culoare codifica directia si amplitudinea vectorului de flux optic (similar reprezentarii culorilor in modelul HSI, valoarea Hue (culoarea) codificand directia vectorului de fux optic iar saturatia codificand amplitudinea acestuia (fig. 7.1).

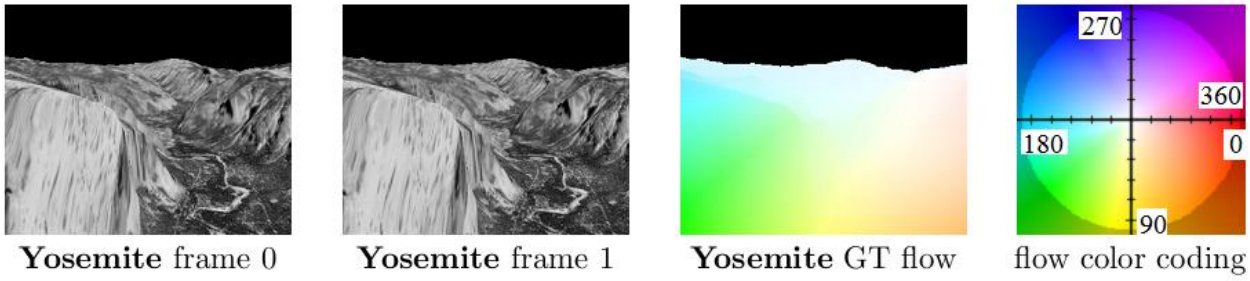
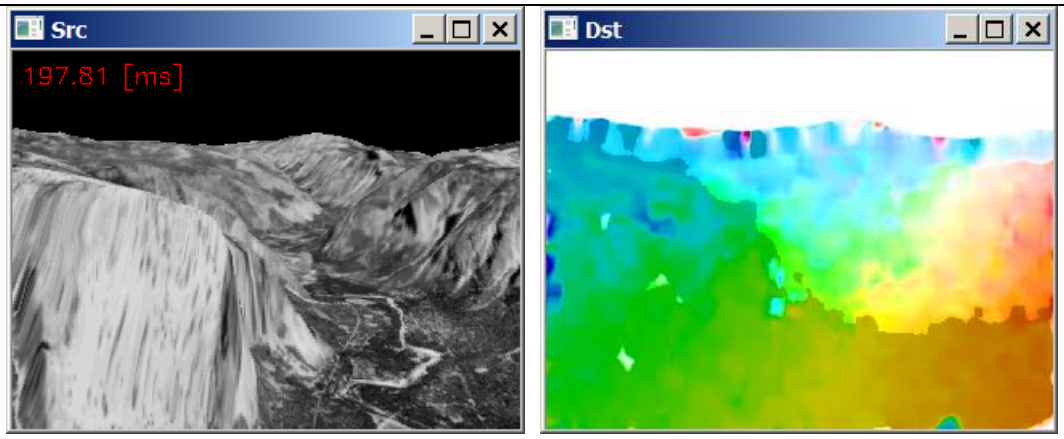


Fig. 8.1. Exemplificare a rezultatului codificarii fluxului de miscare in conformitate cu conventia de culoare Middleburry [2].

7.3. Masurarea si afisarea timpului de procesare

Pentru a masura timpul de procesare aferent cadrului curent puteti folosi functiile 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 destinatie in care s-au afisat valorile vectorilor fluxului optic prin codificarea de culoare Middleburry.

7.4. Calculul histogramei directiilor vectorilor de miscare

Calculul histogramei directiilor vectorilor de miscare poate fi utila in analiza miscarii din imagine. Pentru a calcula directia vectorului de miscare asociat fiecarui pixel trebuie calculat unghiul facut de vector cu axa Ox (orizonala). 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 se considera cu originea in imaginea trecuta (prev)
// si varful in imaginea curenta (crnt) -> 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 d emiscare
```

Histograma poate fi reprezentata sub forma unui vector de intregi, care va trebui reinitializat cu 0 in fiecare cadru/frame al secventei de imagini prelucrate (deci in bucla de procesare a secventei):

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

Observatie: Daca la afisarea fluxului optic dens (*showFlowDense*) se face filtrarea vectorilor pe baza pragului *minVel* (*minVel* > 0) impus magnitudinii vectorilor de flux optic, **aceeasi conditie trebuie integrata si la construirea histogramei (se vor acumula in histograma doar vectorii de flux optic cu magnitudinea > *minVel*).**

7.5. Afisarea histogramei directiilor vectorilor de miscare

Pentru fiecare cadru din secventa video se va calcula histograma directiilor si se va afisa intr-o fereastra noua. Afisarea histogramei directiilor se va putea realiza prin apelul functiilor *showHistogram* sau *showHistogramDir* integrate in modulul *Functions*.

Functia *showHistogram* (vezi ANEXA) este dedicata pentru afisarea unei histograme generice (fig. 7.3.a) – vezi si laboratorul de Procesarea Imaginilor. Afisarea se va face prin trasarea de segmente de dreapta verticale de la baza histogramei pana la valoare curenta a histogramei, pentru fiecare intare din vectorul histograma („bin” – engleza).

Exemplu apel:

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

Functia *showHistogramDir* este dedicata pentru afisarea histogramei directiilor vectorilor de flux optic (fig. 7.3.b). Afisarea se va face prin trasarea de segmente de dreapta verticale de la baza histogramei pana la valoare curenta a histogramei, pentru fiecare intrare din vectorul histograma („bin” – engleza). Fiecare „bin” al histogramei va avea culoarea codificata a directiei (vezi modulul *colorcode* si apelurile functiei `makeColorwheel()`, respectiv `make_HSI2RGB_LUT()` care initializeaza un LUT necesar pentru generarea culorilor “bin”-urilor din histograma si este definita in *Function.cpp* - pe care le-ati integrat deja la inceputul functiei de procesare).

Exemplu apel:

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

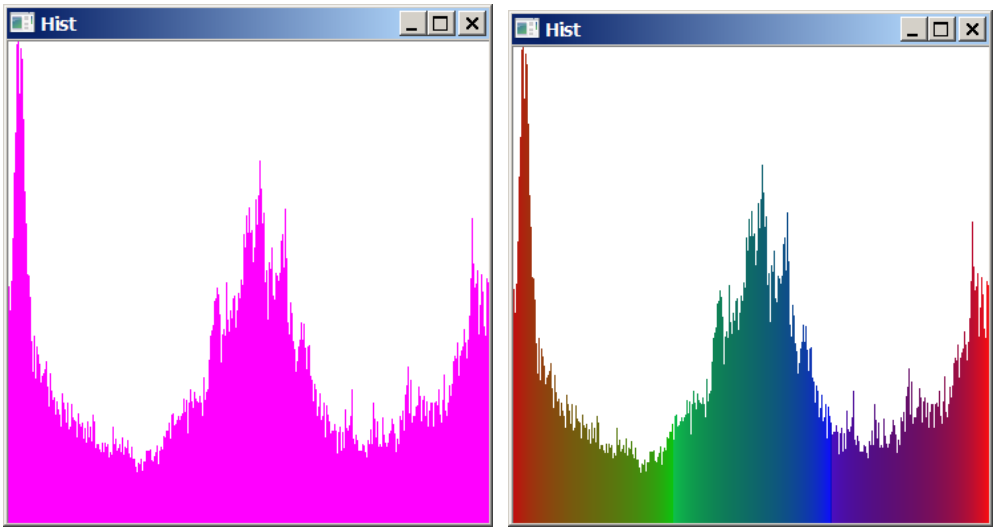


Fig. 7.3.a. Afisarea histogramei directiilor in forma generica (*ShowHistogram*); b. afisarea “bin”-urilor histogramei prin codurile de culoare ale directiilor(*ShowHistogramDir*)

7.6. Activitati practice

1. Se va integra functia de calcul a fluxului optic `calcOpticalFlowFarneback`, se va afisa harta densa a vectorilor de flux folosind codificarea *colorcode*. Se va afisa la linia de comanda si timpul de procesare pentru fiecare cadru.
2. Se vor vizualiza rezultate pentru diverse setari ale parametrilor de intrare ai functiei `calcOpticalFlowFarneback` si pt. parametrul de filtrare *minVel* al magnitudinii vectorilor de miscare din functia de afisare `showFlowDense`.
3. Se va calcula histograma directiilor vectorilor de miscare si se va afisa. Se va adauga o conditie de filtrare bazata pe pragul *minVel* impus magnitudinii vectorilor de miscare (*minVel* = 0.4 .. 1.0) si la calulul histogramei directiilor.
4. Se vor vizualiza rezultatele (in varianta cu fitrarea modulului vectorilor) pe secvente de bitmaps relevante (ex: *Polus.zip*). Se va face analiza miscarii prin vizualizarea histogramei si se vor identifica urmatoarele cazuri: scenariu cu camera statica si obiect/obiecte in miscare, scenariu cu camera si obiect/obiecte in miscare, etc:

Fig. 4. Rezultate experimentale pentru secventa *Taxi.avi*.

5. **Tema de casa / proiect de semestru:** Se va implementa o metoda de filtrare (filtru trece jos / de netezire - medie aritmetica sau gaussian) a histogramei directiilor si se vor detecta varfurile (maximele locale) ale histogramei (vezi http://users.utcluj.ro/~tmarita/IPL/IPLab/PI-L3r.pdf). Se vor afisa unghiurile (directiile) [rad] corespunzatoare acestor maxime locale in imaginea sursa (de ex. sub timpul de procesare). Se va experimenta implementarea pe un scenariu relevant (scenariu cu camera statica si obiect/obiecte in miscare: *Taxi.avi* sau *walkstraight.avi*) in varianta cu filtrare a modulelor directiilor (ex. *minVel* = 0.7).

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.

7.8. ANEXA

Definitia functiei de afisare a fluxului optic dens: `showFlowDense`

```
/*-----------------------------------------------------------------
Function used to display 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);
    }
}
```

Definitia functiei de afisare a histogramei intr-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);
    }
}
```

Definitia functiei de afisare a histogramei directiilor 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);
    }
}
```