# 7. Estimarea fluxului optic si urmrairea punctelor de interes in secvente de imagini

## 7.1. Introducere

Scopul acestei lucrari este de a studia si integra metode de detectie a miscarii obiectelor din imagine succesive. Miscarea poate fi detectata prin estimarea unor parametrii ai miscarii:

**Câmp de mişcare** := setul vectorilor (vitezelor) de mişcare ale punctelor din imagine (2D) induse de mişcarea relative dintre scena (obiecte ale scenei) si camera

- nu este măsurabil direct din imagine, dar se poate estima prn urmarirea in imagini succesive a unor trasaturi relevante (colturi)

**Fluxul optic** := mişcarea aparenta a „patern-urilor" de intensitate din imagine

- se poate măsura direct din imagine
- este o aproximare a câmpului de mişcare cu o rată de eroare mică în puncte cu gradient mare (daca direcţia gradientului si direcţia mişcării coincid)

## 7.2. Metode de măsurare a fluxului optic

Toate metodele au la baza ecuaţia constantei intensităţii imaginii (Image Brightness Constancy Equation) care se bazează pe asumpţia ca intensitatea luminoasă a unui punct din scenă (chiar daca acest punct îşi schimbă poziţia de la o imagine la alta) rămâne constantă:

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

Unde:

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

$E_t = E(t) - E(t-1)$ - derivata temporala a intensităţii imaginii în punctul curent studiat

E – intensitatea imaginii în punctul curent studiat

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

Majoritatea metodelor de estimare a fluxului optic se bazeaza pe algoritmi iterativi care inceraca sa gaseasca pentru fiecare pixel din imagine un vector de deplasamanet care minimizeaza urmatoarea functie reziduala:

$$\epsilon(\mathbf{v}) = \epsilon(v_x, v_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x,y) - J(x+v_x, y+v_y))^2 . \tag{6.2}$$

unde I*(x,y)* – punct in imaginea I (t) iar J*(x+*$v_x$ *, y+*$v_y$*)* este noua locatie a aceluiasi punct (pixel) in imaginea J de la mometul (t+Δt).

**Calculul fluxului optic prin algoritmul Horn-Schunk (iterativ)**

- metodă iterativă
- Parametrii de intrare: $n_0$ – numărul maxim de iteraţii; $\lambda$ – ponderea de corecţie

1. Se face o primă parcurgere a imaginii. Pentru fiecare pixel p se calculează $E_x(p)$, $E_y(p)$, $E_t(p)$ şi se iniţializează $v_x$ şi $v_y$ cu 0 (este necesară alocarea unor matrice de dimensiunea imaginii pentru stocarea acestor valori).
2. Se aleg valorile pt. $\lambda$ (ex. $\lambda = 10$) şi n0 ($n_0 = 8$).
3. Pentru n = 1 .. $n_0$ :
   Se parcurge imaginea. Pentru fiecare pixel *p* se calculează valorile medii ale $v_x$ si $v_y$ (din vecinii de pe direcţiile cardinale:

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

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

   Se calculează coeficientul $\alpha$:

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

   Se actualizează (corectează) valorile $v_x$ si $v_y$:

$$v_x = \bar{v}_x - \alpha \cdot E_x \quad (6.6)$$

$$v_y = \bar{v}_y - \alpha \cdot E_y \quad (6.7)$$

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

Se initializeaza vectoruii de flux optic cu 0:

$$\overline{\nu}^0 = [0 \;\; 0]^T$$

Ca si conditie de terminare a iteratiilor se poate limita numarul de pasi ex: K=20 sau se poate impune conditia ca norma factorului de corectie $\| \eta^k \| < th$ (ex. th = 0.03):

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

Se calculeaza imaginea diferenta:

$$\delta I_k(x,y) = I^L(x,y) - J^L(x + \nu_x^{k-1}, y + \nu_y^{k-1})$$

Se calculeaza vectorul de eroare al imaginii diferenta:

$$\overline{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k(x,y)\, I_x(x,y) \\ \delta I_k(x,y)\, I_y(x,y) \end{bmatrix}$$

Se calculeaza corectia pt. fluxul optic la pasul k:

$$\overline{\eta}^k = G^{-1}\, \overline{b}_k$$

Se actualizeaza fluxuul optic la pasul k:

$$\overline{\nu}^k = \overline{\nu}^{k-1} + \overline{\eta}^k$$

**End**

Vectorii de flux final vor fi $\nu^K$.

### 7.3. Urmarirea de trasaturi in imagini succesive

Estimarea campului de miscare se poate face doar pentru trasaturi discrete prin urmarirea unor astfel de trasaturi relevante (ca de exemplu colturi) in imagini succesive. O astfel de abordare este metoda Lukas-Kanade in varianta piramidala [1].

Acest algoritm aplica metoda iterativa de calcul a fluxului optic Lukas-Kanade pe o piramida de imagini (imagine la rezolutia originala – nivel L= 0) si subimagini obtinute din imaginea originala prin decimare (nivel L = 1, 2, ... 4 - de obicei se considera L=3). Vectorii de flux optic se estimeaza doar pe un set de trasaturi relevante (cu putere de discriminare mare) care au cel putini doua directii distincte ale gradientului, cum sunt colturile.

Pentru astfel de trasaturi fluxul optic si campul de miscare coincid, rezultatul obtinut la iesire fiind campul de miscare al acestor trasaturi (se mai numeste si flux optic „rar" – sparse optical flow).

Metoda este descrisa sumar in cursul 6 ( http://users.utcluj.ro/~tmarita/HCI/C6.pdf ) si prezentata in detaliu (intr-o maniera inteligibila) in [1] – se va studia ca tema de casa!

### 7.4. Detalii de implementare

#### Sablonul de procesari al secventelor de imagini (bitmaps)

Implementarea se va face folosind ca sablon/model exemplul `testOpenImagesFld()` din *OpenCVApplication.cpp*. in care este exemplificata parcurgerea unei secvente de imagini continute untr-un folder selectat prin controlul dialog box. De asemenea se va folosi si sablonul de prelucrare a unei secvente de imagini intrdusa in laboratorul 6: se proceseaza imagine curenta si imagine trecuta (salvata intr-un buffer). Procesarile pentru estimarea vectorilor de miscare 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");
```

Sablonul de procesare pentru bucla principala (in care se sitesc succesiv imaginile din secventa de bitmaps) ar trebui sa arate asa:

```
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
            }
      }
```

Ca preprocesare este utila aplicarea unui filtru gaussin pe imaginea sursa convertita in grayscale pentru eliminarea eventualelor zgomote ( ex. un gaussian de 5x5 cu sigma=0.8).

**Calculul fluxului optic folosind metoda Horn-Schunk**

Se va implementa algoritmul de calcul al fluxului optic Horn-Schunk sub forma unei functii care se va apela in sablonul 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 imaginii crnt sip rev se convertesc 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. x si y ale fluxului optic intr-o matrice cu elemente de tip Point2f
      flow = convert2flow(vx, vy);

      // Vizualizare rezultate intermediare:
      // gradient,derivata temporala si componentele 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);
}
```

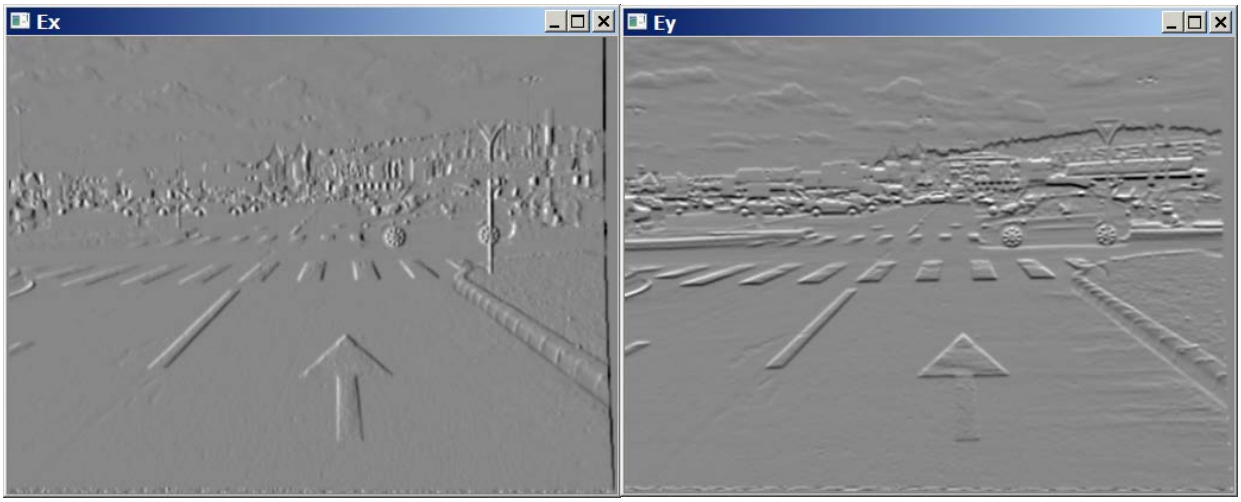

Fig. 7. 1 Matricile derivatelor partiale normalizate la imagini grayscale (gri – derivata nula; negru - derivata negativa; alb derivata pozitiva)

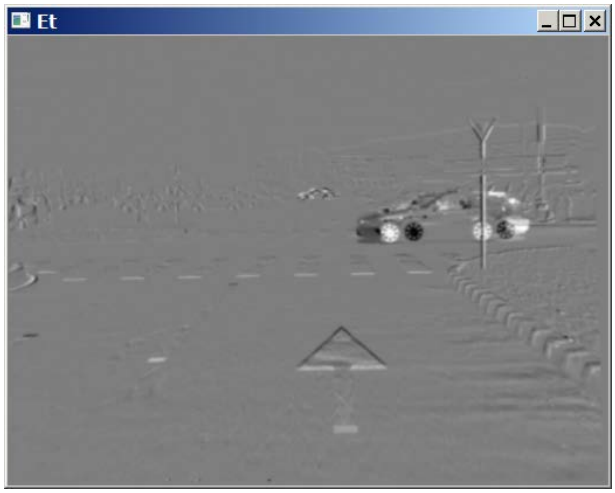

Fig. 7.2. Matricea derivatei temporale normalizata la grayscale (gri – derivata nula; negru - derivata negativa; alb derivata pozitiva)

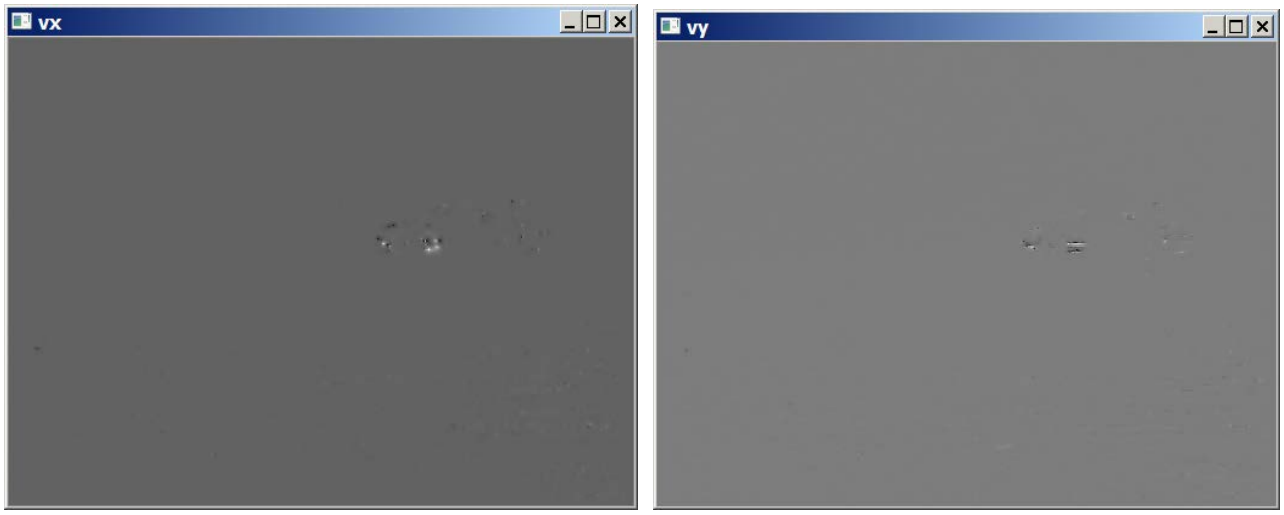

Fig. 7.3. Matricele componentelor orizontale si verticale ale vectorilor de flux optic normalizate la imagini grayscale.

Apelul functiei Horn Schunk de calcul al fluxului optic si a functiei de afisare (inserati cele 2 apeluri de mai jos in sablonul de procesare al secventei de imagini:

```
// 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);
```

**Functia de afisarea vectorilor de flux optic**

Afisarea se poate realiza prin apelul functiei *showFlow* care deseneaza vectorii de flux optic (segmente de dreapta). Functia permite operatia de zoom pe imaginea de iesire (factor de scalare *mult*), filtrarea vectorilor pe baza unui prag *minVel* aplicat modulului lor, afisarea selectiva a vectorilor si/sau a originii acestora. Functia este integrata in modulul *Functions*: *Functions.h* / *Functions.cpp* si *NU mai este nevoie sa o copiati in functia de procesare*.

Functia *showFlow* afiseaza automat rezultatul in fereastra specificata ca si prim parametru (`const string& name`). Deci NU mai este nevoie sa adaugati in bucla de procesarea un apel explicit pt. afisarea ferestrei destinatie (de ex. `imshow(WIN_DST, dst)`).
Exemplu apel:

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

**Observatie importanta legata de accesul la pixeli in structura Mat:** - accesul la pixeli sau date structurate in forma matriceala (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);`

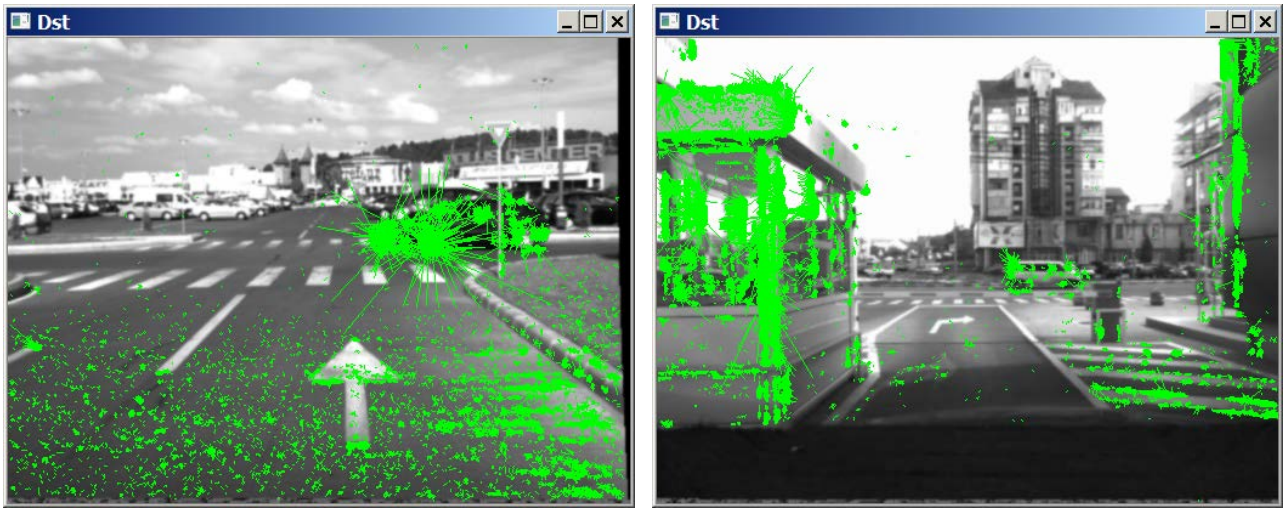


Fig. 7.1. Afisarea rezultatelor estimarii fluxului optic pentru metoda Horn-Schunk.

**Urmarirea trasaturilor prin metoda LK piramidala**

Aceasta metoda necesita detectia prealabila a unor trasaturi de interes (colturi) vezi lucrarea L5 – met. `goodFeaturesToTrack` care furnizeaza un set de punte (colturi) din imaginea tercuta `prev_pts` care vor fi foloiste ca si intrari la metoda de urmarite a trasaturilor LK piramidala (`calcOpticalFlowPyrLK`). Metoda furnizeaza la iesire setul de puncte/colturi corespondente din imaginea curenta `crnt_pts`, un vector care indica daca s-a gasit potrivirea pt. fiecare punct `status` si un vector de eroare `error` :

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

Pentru a folosi functia LK piramidala trebuie sa aveti inclusa in sectiunea `#include` din *common.h* dierctiva:

```
#include <opencv2/video/tracking.hpp>
```

Initializarea parametrilor pentru detectorul de colturi (`goodFeaturesToTrack`) se va face ca si in lucrarea L4. Secventa de initializare se va adauga undeva la inceputul functiei de procesare. Pentru metoda de calcul a fluxului optic LK piramidala sunt necesare initializari asemanatoare cu cele din exemplul de mai jos (se vor adauga tot la inceputul functiei 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 wlow 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; // maximal 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;
```

Afisarea rezultatului se poate face prin apelul functiei `showFlowSparse` (vezi ANEXA). Functia este integrata in modulul *Functions*: *Functions.h* / *Functions.cpp* si *NU mai este nevoie sa o copiati in functia de procesare*:

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

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

Functia *showFlowSparse* permite afisarea vectorilor da miscare ale trasturilor (colturilor) detectate in imagini. Functie permite operatia de zoom pe imaginea de iesire (factor de scalare *mult*), filtrarea vectorilor pe baza valorilor din vectorul *status*, afisarea selectiva a vectorilor si/sau a originilor acestora (cerc rosu de raza 4) si a terminatilor (cerc albastru de raza 2) – fig. 2.

Fig. 7.2. Afisarea rezultatelor estimarii campului de miscare obtinut prin metoda LK piramidala pentru un scenariu cu un vehicul care traverseaza o intersectie de la dreapta la stanga.

### 7.5. Mersul lucrării

1. Pentru test se vor folosi secvente de bitmaps (*polus.zip*, *sigma.zip*): pe care le puteti descarca din folderul http://users.utcluj.ro/~tmarita/HCI/L7/

2. Se vor implementa algoritmul Horn-Schunk de calcul al fluxului optic si se va integra in sablonul de procesare furnizat.
3. Se va integra metoda de calcul a campului de miscare LK piramidala). Se vor afisa grafic si valorile erorilor (valoarea *error[i]* pentru colturile la care s-a detectat corespondentul *status[i]==1*. Aceatsa eroare se poate reprezenta desenand cercul de origine (rosu) al vectorilor de miscare cu o raza variabila proportionala cu eroarea (este necesara gasirea unui factor de scalare adecvat). Modificarea trebuie facuta in codul functiei `showFlowSparse()` la apelul functie `circle` de desenare a originii vectorului.
4. Optional sau tema de casa: se vor codifica culorile de desenare ale vectorilor de miscare folosind/integrand functia de codificare Middleburry [2] (vezi codul din modulul *colorcode*). Functia de colorare va colora vectorii dupa un cod de culoare care codifica directia vectorilor cu o culoare corespunzatoare valorii/unghiului Hue (H) din modelul HSI si o Saturatie (S) proportionala cu directia vectorilor.

Exemplu de integrare (modificati functia showFlow dupa urmatorul sablon):

```
unsigned char color[3];
for(int y = 0; y < flow.rows; ++y)
    for(int x = 0; x < flow.cols; ++x)
    {
        Point2f f = flow.at<Point2f>(y, x);
        computeColor(f.x, -f.y, color); // gnerates a color that encodes the
        orientation and modulus of the OF vector in (x,y)
        . . .
      }
```

Pentru desenarea compunentelor de culoare (R, G, B) se vor folosi valorile: (color[0], color[1],color[2]).

Mai este necesar sa initializati un LUT in care sunt generate culorile prin apelul: `makeColorwheel()` la inceputul functiei de procesare.

**Bibliografie**

[1] Pyramidal Implementation of the Lucas Kanade Feature Tracker - Description of the algorithm, Jean-Yves Bouguet, Intel Corporation Microprocessor Research Labs, jean-ves.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/.

---

**ANEXA**

Deorece metoda Horn-Schunk furnizeaza fluxul optic in forma a 2 vectori `velx, vely` se furnizeaza o functie (integrata in modulul *Functions*: *Functions.h* / *Functions.cpp*) de conversie a acestora in format matriceal, pentru compatibilitatea 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 matrix flow
//in wich each element encodes the 2 components of the optical flow vor each pixel
{
    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;
}
```

---

Definitia functiei `showFlow` pentru afisarea (vectorilor) fluxului optic (pentru metodele Legacy):

```
/*-----------------------------------------------------------------
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 line segments
                                line(cflow, p1, p2, CV_RGB(0, 255, 0));
                            if (showCircles) // mark flow vectors' origins by a red circle
                                circle(cflow, Point(x, y) * mult, 2, CV_RGB(255, 0, 0));
                     }
              }

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

---

Definitia functiei `showFlowSparse` pentru afisarea (vectorilor) fluxului pentru metoda LK piramidala:

```
/*-----------------------------------------------------------------
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 line 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);
    }
}
```