9. Detectia feţelor si a componentelor faciale

9.1. Introducere

Scopul acestei lucrari este de a integrara metoda Viola&Jones [1] de detectie a feţelor. Metoda se bazaeaza pe detectia de trasaturi de tip Haar calculate pe sub-regiuni din imagine folosind imaginea integrala (vezi cursul 7-8), si identificarea prezentei feţelor umane cu ajutorul unui clasificator de tip cascada. Abordarea propusa de autori ofera capabilitati de timp real si versatilitate pentru detectia oricaror tipuri de obiect pentru care s-a facut o antrenare prealabila a clasificatorului cascada.

In OpenCV sunt disponibile modelele obtinute prin antrenarea clasificatorului in format *xml. pt. diverse tipuri de obiecte, dar lista nu este exhaustiva putand fi extinsa de catre utilizatori.

Modelele clasificatorilor cascada bazati pe trasaturi Haar sunt disponibile in instalarea curenta a OpenCV in locatia *OpenCV\sources\data\haarcascades* si sunt urmatoarele:

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

Exista si modele pt. clasificatori cascada care folosesc si alte tipuri de trasaturi:
- Local Binary Patterns (LBP): *lbpcascade_frontalface.xml (OpenCV\sources\data\lbpcascades)*
- Histogram of Oriented Gradients (HOG): *hogcascade_pedestrians.xml, (OpenCV\sources\data\hogcascades)*

9.2. Detectia obiectelor folosind clasificatori cascada in OpenCV

Detectia obiectelor se poate realiza apaland functia:

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** – clasa din care se instantiata obiectul (modelul) pt. clasificatorul cascada. Modelul poate fi incarcat dintr-un fisier XML sau YAML folosinf functia Load().
- **image** – Matrice de tip CV_8U continand imaginea sursa.
- **objects** – Vector de iesire continand ROI (dreptunghiuri) care contin obiectele detectate.
- **scaleFactor** – Parametru care specifica factorul de scalare folosit in detectia multirezolutie.
- **minNeighbors** – Parametru care specifica cati vecini ai fiecarui candidat (zone rectangulare) se retin.
- **flags** – Nu se foloseste in modelele cascada de tip nou (doar in cele vechi – cvHaarDetectObjects).
- **minSize** – Dimensiunea rectangulara minima a obiectului detectabil. Obiecte mai mici sunt ignorate.
- **maxSize** – Dimensiunea rectangulara maxima a obiectului detectabil. Obiecte mai mari sunt ignorate.

Un tutorial care exemplifica folosira clasificatorului cascada pentru detectia fetelor din imagini este prezentat in [4]. Se indica modul in care se poate realiza detectia tuturor feţelor dintr-o imagine. Pentru fiecare faţa detectata (in regiunea rectangulara care incadreaza fiecare faţa) se poate face detectia componentelor faciale (ochi, gura, nas). Un sablon derivat din acest tutorial este dat in forma functiei este dat 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|CV_HAAR_SCALE_IMAGE,
                                                 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 |CV_HAAR_SCALE_IMAGE,
                                                 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
}
```

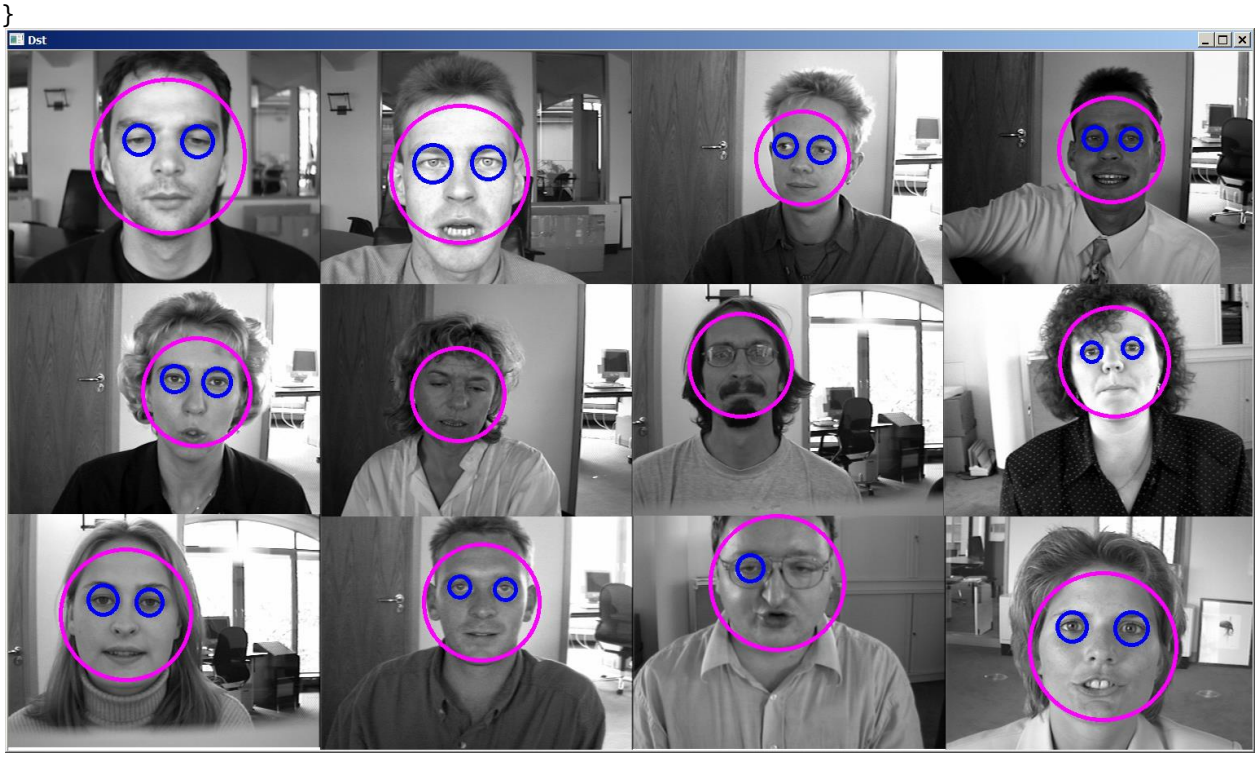


Fig. 9.1. Rezultatele apelului functiei `FaceDetectandDisplay`.

9.3. Detalii de implementare

In modulul *OpenCVApplication.cpp* trebuie sa mai faceti urmatoarele:

- sa includeti in fisierul *common.h* directiva:

```
#include "opencv2/objdetect/objdetect.hpp"
```

- sa declarati urmatoarele variabile globale:

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

Modelele folosite ale clasificatorilor cascada (fisierele *.xml)

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

se vor copia in directorul de lucru current (directorul radacina) al aplicatiei OpenCVApplication.

In functia de procesare aferenta laboratorului curent, inainte de apelul functiei `FaceDetectandDisplay`, este necesara incarcarea modelelor clasificatorilor, ca si in exemplul de mai jos.

```
String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";

// Load the cascades
if (!face_cascade.load(face_cascade_name))
```

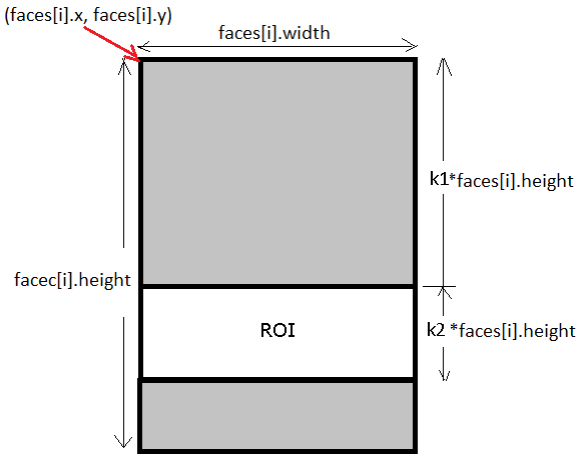
```
	{
		printf("Error loading face cascades !\n");
		return;
	}
	if (!eyes_cascade.load(eyes_cascade_name))
	{
		printf("Error loading eyes cascades !\n");
		return;
	}
```

Exemplu de apel al functiei `FaceDetectandDisplay` (in functia de procesare aferenta laboratorului curent):

```
	...
	src = imread("Images/BioID_1139.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 );
	...
```

9.4. Restrictionarea detectiei in regiuni de interes (ROI)

Pt. detectia altor componente faciale (ex. gura, nas), modelul cascada *haarcascade_mcs_mouth.xml* nu este foarte precis! In consecinta se poate restrictiona regiunea de cautare (ROI) intr-o subregiune a fetei pt. a evita detectii fals-pozitive. Pt. cautarea componentelor faciale intr-o ROI care sa fie o subregiune a fetei se va proceda astfel:



Se defineste o noua ROI rectangulara pentru o subregiune a fetei:

```
	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 si *k1 + k2 <1* (vedeti valorile specificate la punctul 2 din mersul lucrarii !!!)

Se extrage din imaginea sursa matricea de intensitati corespunzatoare ROI definite:

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

Se apeleaza detectorul de obiecte (nu uitati sa incarcati in prealabil modelul clasificatorului cascada). De exemplu pt. gura procedati 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 |CV_HAAR_SCALE_IMAGE,
                                        Size(minMouthSize, minMouthSize) );
```

9.5. Mersul lucrării

1. Se va integra intr-o functie de procesare noua metoda de detectie a fetelor pentru detectia tuturor fetelor dintr-o imagine si a ochilor din fiecare fata (functia `FaceDetectandDisplay`). Imagini de test puteti descarca de la urmatoarea adresa: http://users.utcluj.ro/~tmarita/HCI/Media/Images/Faces.zip
2. Se va extinde procesarea de la pasul 1 pentru pentru detectia si a altor componente faciale (gura, nas). Optimizati cautarea componentelor faciale doar in anumite zone din ROI-ul corespondent fetei ca si in exempluul de mai jos:

```
Rect eyes_rect; //eyes are in the 20% ... 55% height of the face
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; //nose is the 40% ... 75% height of the face
…
Rect mouth_rect; //mouth is in the 70% ... 99% height of the face
…
```

Observatii:

- Cel mai simplu este sa afisati componentele faciale ca si dreptunghiuri (folositi functia rectangle: `rectangle(frame, rect, culoare, grosime, 8, 0);`
- atentie la modul in care se calculeaza pozitia absoluta a componentei faciale relative la coltul stanga-sus al imaginii: componentele detectate sunt detectate relativ la regiunea de interes (ROI) in care le-ati cautat ⇒ mai departe trebuie sa le translatati relativ la coltul stanga-sus al imaginii ca sa le afisati correct in imaginea destinatie (vezi exemplul dat in `FaceDetectandDisplay`)

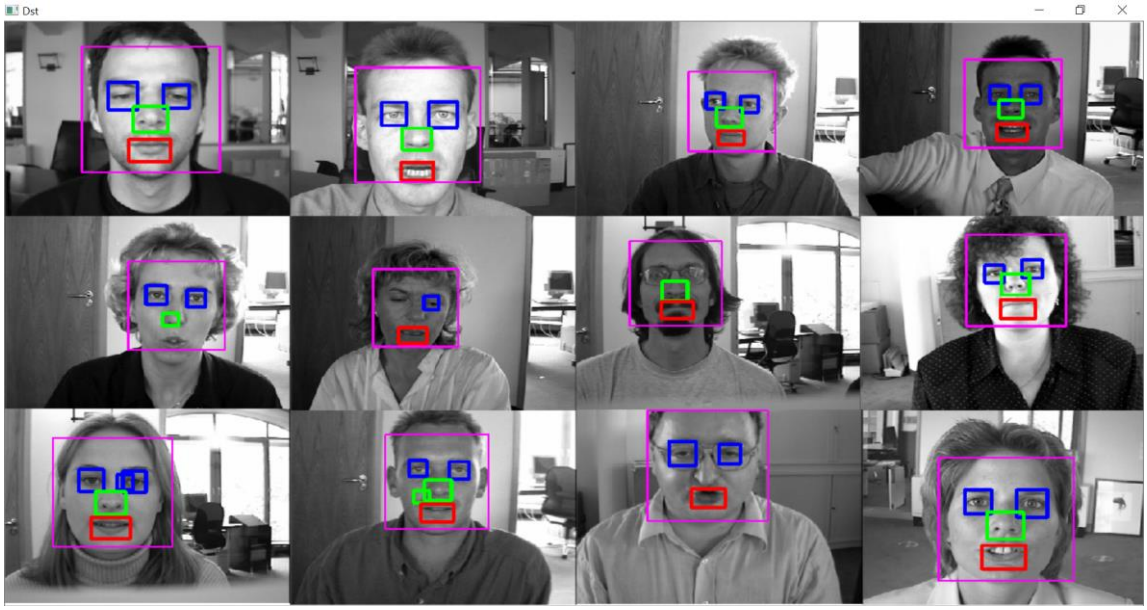


Fig. 9.2. Rezultatele apelului functiei FaceDetectandDisplay modificate pentru detectia feţelor si componentelor faciale (varianta optimizata prin restictionarea zonelor in care se cauta componentele faciale

3. Se va crea o noua functie de procesare pentru detectia fetelor din secvente video (off-line / on-line). Pentru acest task creati o versiune simplificata a functiei `FaceDetectandDisplay` care sa detecteze si afiseze doar fetele (renuntati la cautarea celorlalte componente faciale). Se va masura si afisa (la linia de comanda) timpul de procesare pentru apelul functiei `FaceDetectandDisplay` .
4. Se va testa detectia fetelor (doar pt. fete, fara componente faciale) folosind modelul de clasificator bazat pe trasaturi LBP (Local Binary Pattern) in locul celui bazat pe trasaturi Haar. Pt. aceasta se va incarca modelul clasificatorului LBP *lbpcascade_frontalface.xml* in locul *haarcascade_frontalface_alt.xml*

Bibliografie

[1] PaulViola, Michael Jones, Robust Real-time Object Detection, SECOND INTERNATIONAL WORKSHOP ON STATISTICAL AND COMPUTATIONAL THEORIES OF VISION– MODELING, LEARNING, COMPUTING, ANDSAMPLING VANCOUVER, CANADA, JULY 13, 2001, https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf http://cs.nyu.edu/~eugenew/publications/viola-facedet04-talk.pdf
[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