9. Validarea detectiei feţei si a componentelor faciale pe sevente de imagini

9.1. Scop

Scopul acestei lucrari este de a integra metoda Viola&Jones de detectie a feţelor si a componentelor faciale (in principal ochii) cu o validare bazata pe analiza unei secvente de imagini. Genul de utilizare a unei astfel de abordari este in aplicatii de tip „liviness detection” in sistemele biometrice bazate pe recunoasterea de fete. Astfel de aplicatii inceraca sa valideze suplimentar o un algoritm de recunoastere faciala printr-o analiza suplimentara bazata pe miscare pentru a preveni incercarile de fraudare a sistemului prin prezentarea unui imagini a personei in loc de prezenta efectiva a persoanei.

La modul concret se va face o validare bazata pe *detectia clipitului* persoanei respective. Din punct de vedere practic se va realiza o integrare a implementarilor din lucrarile L5 (Segmentarea obiectelor in miscare prin modelarea si eliminarea fundalului - “Background Subtraction”) si L8 (Detectia feţelor si a componentelor faciale) si procesari de imagini suplimentare.

9.2. Mersul lucrarii

Se va citi secventa video de test (`test_msv1_short.avi`). Procesarea se va face cadru cu cadru, avansul intre cadre facandu-se prin apasarea unei taste (setati la ‚0’ parametrul functiei `waitKey(0)`). Se va converti cadrul-ul curent (de analizat) din color (cu trei canale) in grayscale (matrice cu un canal). Pentru fiecare frame/cadru (incepand cu al doilea) se vor realiza urmatoarele operatii:

1. Se va apela metoda de detectie a fetelor si a componentelor faciale pe frame-ul curent. Se va memora pozitia (dreptunghiul care incadreaza) prima faţă gasita in frame-ul curent (`Rect faceROI=faces[0]`). Acest dreptunghi se va folosi ulterior ca si masca de procesare (Regions Of Interest – ROI).

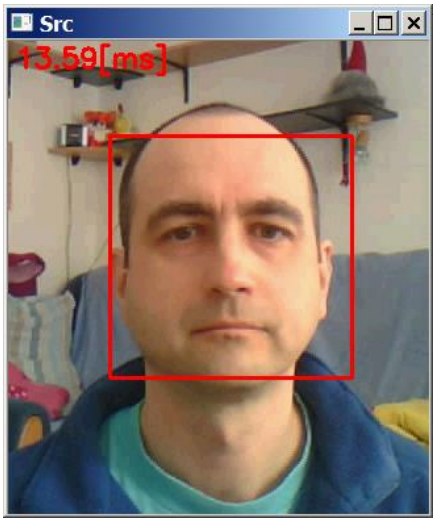
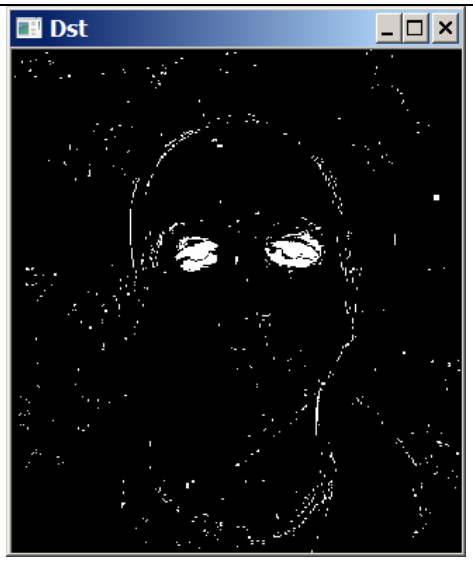


Fig. 9.1. Incadrarea primei fete gasite in imaginea curenta.

2. Se va implementa operatia de „Background Subtraction” prin metoda de diferentiere simpla intre cadrul curent si cel precedent. In momentul in care persoana clipeste in dreptul ochilor vor aparea zone cu pixeli albi la nivelul ochilor:



Fig, 9.2. Imaginea diferenta dintre cadrul curent si cadrul precedent.

3. Pe imaginea diferenta se vor aplica operatii morfologice (o eroziune urmata de o dilatare) pentru eliminarea zgomotelor datorate eventualelor miscarii ale capului sau zgomotului din imagine sau artefactelor introduse de compresia secventei video.

Fig. 9.3. Imagine diferenta dupa aplicarea filterlor morfologice
(o eroziune urmata de o dilatare cu nucleu de dimensiune 3x3)

4. Se va masca imaginea diferenta finala (*dst*) de la pasul 3 cu ROI-ul feței obtinut la pasul 1):

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

5. Se realizeaza etichetarea obiectelor ramase in zona ROI aferenta fetei (din matricea *temp*). Se calculeaza proprietatile geometrice ale componentelor conexe detectate: aria si centrul de masa. Pentru etichetarea obiectelor si calculul proprietatilor geometrice (arie si centre de masa) folositi codul dat ca si exemplu in functia `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, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
Moments m;
if (contours.size() > 0)
{
	// iterate through 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. 9.4. Rezultatul aplicarii operatiei de etichetare pe imaginea binatra a ROI-lui fetei.

6. Se vor adauga ariile si centrele de masa ale obiectelor detectate intr-o lista (vector) de candidati Folositi o structura pentru a memora elementele vectorului:

```
	// 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 abore de decizie simplu bazat pe cateva reguli simple:

- Se retin pozitiile centrelor de masa ale celor doua obiecte (din lista de candidati) care au aria cea mai mare (ca sa eliminati eventualele zgomote ramase in imgine chiar si dupa aplicarea filtrelor morfologice). Se vor desena in fereastra aferenta ROI (matricea *roi*) centrele de masa folosind functia `DrawCross` (definita modulul *Functions*): cu rosu pentru ochiul stang si cu albastru pentru ochiul drept (imaginea pe care o observam este in oglinda).
- Pozitiile centrelor de masa (CM) ale acestor obiecte trebuie sa fie aproximativ pe o linie orizontala (ex: diferentele pe y ale CM sa fie mai mici decat o valoare ky*ROI.height (ex: ky ≈ 0.1) ŞI sa fie situate in jumatatea superioara a ROI)
- Diferentele pe orizontala (distanta pupilara DP) dintre centrele de masa (CM) sa fie un procent din latimea fetei: kx1*ROI.width < DP < kx2*ROI.width (ex: kx1 ≈ 0.3, kx2 ≈ 0.5)
- Pozitiile centrelor de masa aale celor mai mari 2 richete trebuie sa apara in cele 2 jumatati verticale diferiteale feţei: xc_ochi_ochi stang > ROI.width/2 (ochiul stang il vedem in imagine in dreapta) si xc_ochi_ochi_drept < ROI.width/2 (ochiul drept il vedem in imagine in stanga)

Observatie: va trebui sa ajustati eventualele constante (kx, ky la valori optime)!

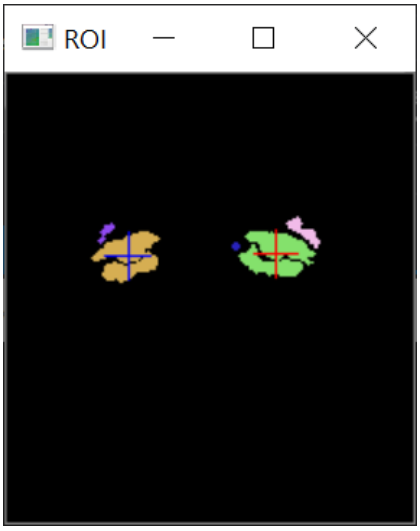


Fig. 9.5. Marcarea centrelor celor mai mari 2 obiecte cu o cruce rosie pentru ochiul stang (in jumatatea stanga a fetei) si cu o cruce albastra pentru ochiul drept (in jumatatea dreapta a fetei) - imaginea pe care o observam este in oglinda !

8. Se va afisa in imaginea sursa faţa detectata cu culoarea verde daca sa trecut cu succes prin procesul de validare (s-a detectat clipitul – ochi inchisi) si cu rosu in caz contrar (ochii deschisi). Se va afisa si timpul de procesare.

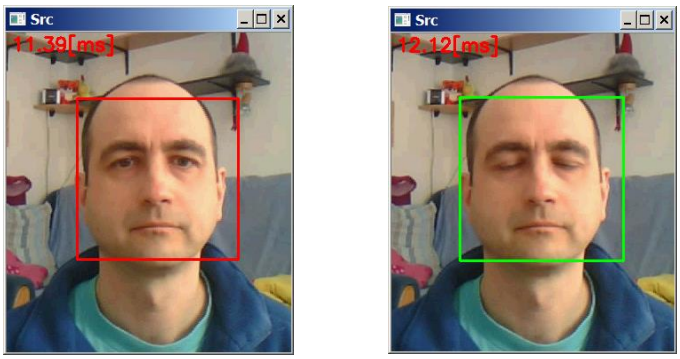


Fig. 9.5. Afisarea momentului in care se detecteaza clipitul peste imaginea sursa prin schimbarea culorii dreptunghiului are incadreaza fata.

Pentru desenarea unui dreptunghi aveti la dispozitie in OpenCV functia `rectangle(frame, faceROI, color, 2, 8, 0)`, unde *color* este o structura de tip *Scalar(B,G,R)*

Pentru a afisa timpul de procesare *t* peste matricea sursa puteti folosi functia `putText`:

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

9.3. Activitati practice

1. Implmenetati pasii de procesare descrisi in capitolul 9.2 intr-o singura functie de procesare. Testati implementarea pe secventa video `test_msv1_short.avi`.
2. Testati implementarea si pe streamul video live captat cu camera web proprie, inlocuind modul de intializare al obiectului *VideoCapture*:
   ```
   VideoCapture cap(0);  // video from webcam
   ```
   Daca este necesar ajustati parametrii folositi in algoritm pentru o functionare optima.