UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Urmarirea fetei in imagini succesive

-proiect-

Nume: Péntek Tamás

Grupa: 30642

Data: 05.01.2021

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Introducere

In cadrul acestui proiect, a fost implementat un program, care urmareste fata unei persoane in imagini successive. Pentru algoritmul de tracking a fost folosit Tracking API, implementat in OpenCV. Pe langa acesta a fost utilizat algoritmul Viola-Jones pentru detectarea fetei.

Consideratii teoretice

Acest proiect se bazeaza pe doi algoritmi: algoritmul Viola-Jones care a fost utilizat pentru detectarea fetei si algoritmul de tracking, folosit pentru urmarirea fetei pe imagini succesive.

Algoritmul Viola-Jones

Algoritmul Viola-Jones este unul dintre cel mai performant si eficient algoritmn utilizat pentru detectia fetelor, care functioneaza in timp real. Aceasta metoda ofera o viteza remarcabila si o rata foarte inalta de acuratete. Metoda completa foloseste pana la 38 de filtre sau clasificatoare. Acest algoritm nu analizeaza direct imaginea completa, ci cauta anumite caracteristici (features) dreptunghiulare in imagine completa. Aceste caracteristici sunt cunoscute sub denumirea de caracterisitici de tip Haar, dupa matematicianul Alfred Haar.

Prima data transforma imaginea color intr-o imagine greyscale, iar pe aceasta imagine pot fi analizate anumite caracteristici dreptunghiulare prin insumarea valorilor intensitatilor pixelilor in diferite blocuri dreptunghiulare. La implementarea acestui algoritm au folosit un sistem de invatare automata (machine learning) denumit AdaBoost (Adaptive Boosting) pentru a construi un clasificator. In cele din urma, sunt folosite 38 de niveluri pentru cascada de clasificatori. Ulterior, pentru a face mai bine fata situatiilor in care fetele nu apar frontal in imagine, au introdus niste filtre diagonale si niste caracterisitici rotite cu 45 de grade.

Acest algoritm este implementat in biblioteca OpenCV, astfel algoritmul poate fi utilizat direct, prin includerea functiilor si structurilor in programul nostru.

Algoritmul de tracking

Daca avem deja un algoritm de detectare a fetelor, de ce nu putem folosi acest algoritm pe fiecare imagine fara sa mai implementam un algoritm de tracking? Primul argument este ca un algoritm de tracking este mult mai rapid, decat un algoritm de detection. Cand urmarim un obiect care a fost detectat in frame-ul anterior, deja stim foarte multe lucruri despre acest obiect: locatia, directia si viteza miscarii. In acest fel, putem utiliza toate aceste informatii pentru a localiza cu exactitate si mult mai rapid obiectul pe frame-ul urmator. Un alt argument este ca un algoritm bun de tracking gaseste fata pe frame-ul urmator chiar si in conditii mai slabe, in care un algoritm de detection esueaza.

La implementarea acestui proiect a fost utilizat un algoritm de tracking, care este implementat in biblioteca OpenCV. In aceasta biblioteca este implementat un API de tracking care contine 8 diferite algoritmi de urmarire: BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE, and CSRT. La inceput, a fost testat fiecare algoritm, iar algoritmul care are cea mai buna performanta si acuratete pentru a urmari fete, este algoritmul MOSSE. De aceea, in cadrul acestui proiect am folosit acest algoritm.

Un algoritm de tracking are ca si scop sa gaseasca un obiect pe frame-ul curent, dat fiind ca am urmarit obiectul cu succes pe toate frame-urile anterioare. Deoarece am urmarit obiectul pana la cadrul curent, stim cum s-a deplasat. Cu alte cuvinte, cunoastem parametrii modelului de miscare. Dar avem mult mai multe informatii decat doar miscarea obiectului: stim cum arata obiectul in fiecare dintre cadrele anterioare, astfel incat putem sa construi un model de aspect (appearance model) care codifica aspectul obiectului. Acest model de aspect poate fi folosit pentru a cauta in vecinatatea locatiei prezise de modelul de miscare pentru a prezice mai corect locatia obiectului.

Modelul de miscare prezice locatia aproximativa a obiectului, iar modelul de aspect ajusteaza fin aceasta estimare pentru a oferi o estimare mai precisa pe baza aspectului. Problema este ca aspectul unui obiect se poate schimba dramatic de la un frame la altul. Pentru a rezolva aceasta problema, in multi algoritmi de tracking modelul de aspect este un clasificator construit si antrenat folosind machine learning.

Algoritmul de tracking folosit este algoritmul MOSSE (Minimum Output Sum of Squared Error). Acest algoritm utilizeaza o corelatie adaptiva (adaptive correlation) pentru urmarirea obiectelor, care produce filtre de corelatie stabile atunci cand algoritmul este initializat cu un singur frame. Corelatia este procesul de deplasare a unei masti de filtrare, a unui kernel peste imagine si calcularea sumei de produse in fiecare locatie. Trackerul MOSSE este robust la variatiile de iluminare, de scalare, la pozitie si la deformari non-rigide. De asemenea, detecteaza ocluzia, care permite trackerului sa faca o pauza si sa reia locul unde a ramas cand obiectul reapare pe frame. Acest tracker functioneaza cu un fps (frame per second) foarte mare, ajunge la 450 fps sau chiar si mai mult. Pe langa acesta, algoritmul este foarte usor de implementat, este la fel de precis ca si alte trackere complexe si este mult mai rapid. Totusi, pe o scara de performanta ramane in urma algoritmilor care folosesc deep learning in implementarea lor. (Nu am gasit detalii de implementare despre acest algoritm.)

Implementare

Acest proiect a fost scris in limbajul de programare C++ si a fost folosit biblioteca OpenCV, versiunea 3.4.11. Atat pentru face detection, cat si pentru tracking a fost folosit niste algoritmi din OpenCV.

Proiectul incepe prin incarcarea fisierului care contine caracterisiticile de tip Haar, iar dupa aceea este apelata metoda process_video() in care este analizat video-ul frame dupa frame.

```
void face_tracking() {
String face_cascade_name = "haarcascade_frontalface_alt.xml";
if (!face_cascade.load(face_cascade_name)) {
printf("Error loading face cascades !\n");
return;
}
process_video();
}
```

In metoda process_video() avem un ciclu de tip while, care citeste fiecare frame din video, pana la sfarsitul video-ului.

```
void process_video() {

    string trackerTypes[8] = { "BOOSTING", "MIL", "MOSSE", "CSRT" };
    string trackerType = trackerTypes[2];
    std::vector<myTracker> trackers;
    VideoCapture video("vid3.mp4");

    if (!video.isOpened())
    {
        cout << "Could not read video file" << endl;
        return;
    }

    int frameNr = 0;
    std::vector<bBox> bBoxes;
    Mat frame;

    while (video.read(frame))
    {

        frameNr++;
        double t = (double)getTickCount();

        if (frameNr > 1) {

            for (int i = 0; i < trackers.size(); i++) {
                bool ok = trackers[i].tracker->update(frame, bBoxes[i].boundingBox);
                if (ok)
                {
                    rectangle(frame, bBoxes[i].boundingBox, Scalar(0, 255, 0), 2, 8, 0);
                    String id = "ID: " + to_string(i + 1);
                    putText(frame, id, Point(bBoxes[i].boundingBox.x + 5,
bBoxes[i].boundingBox.y + 20), FONT_HERSHEY_DUPLEX, 0.75, Scalar(92, 0, 230), 2);
                }
                else
                {
                    std::vector<bBox> bBoxes2;
                    bBoxes2 = detect_face_index("Tracking", frame, true, bBoxes);
                    if (bBoxes2[0].boxID != -1 && bBoxes2[0].boundingBox != Rect2d(0, 0,
0, 0)) {
                    for (int j = 0; j < bBoxes2.size(); j++) {
                            if (bBoxes2[j].boxID == trackers[i].trackerID) {
                                trackers[i].tracker = TrackerMOSSE::create();
```

```
                                trackers[i].tracker->init(frame, bBoxes2[j].boundingBox);
                            }}}}

                if (frameNr % 75 == 0) {
                    std::vector<bBox> bBoxes2 = detect_face_index("Tracking", frame,
false, bBoxes);
                    if (bBoxes.size() == bBoxes2.size()) {
                        if (bBoxes2[0].boxID != -1 && bBoxes2[0].boundingBox != Rect2d(0,
0, 0, 0)) {
                        for (int j = 0; j < bBoxes.size(); j++) {
                            for (int k = 0; k < bBoxes2.size(); k++) {
                                if (bBoxes[j].boxID == bBoxes2[k].boxID) {
                                    int eucDist = getEuclideanDistance(bBoxes[k],
bBoxes2[k]);
                                    if (eucDist > 40) {
                                        for (int i = 0; i < trackers.size(); i++) {
                                            if (trackers[i].trackerID ==
bBoxes2[k].boxID) {
                                                trackers[i].tracker =
TrackerMOSSE::create();
                                                trackers[i].tracker->init(frame,
bBoxes2[k].boundingBox);
                                            }}}}}}}}}}}}
        else {
            bBoxes = detect_face("Tracking", frame);
            if (bBoxes[0].boxID != -1 && bBoxes[0].boundingBox != Rect2d(0, 0, 0, 0)) {
            for (int i = 0; i < bBoxes.size(); i++) {
                    myTracker tracker;
                    tracker.tracker = TrackerMOSSE::create();
                    tracker.tracker->init(frame, bBoxes[i].boundingBox);
                    tracker.trackerID = bBoxes[i].boxID;
                    trackers.push_back(tracker);
                }
            }
        }

        t = ((double)getTickCount() - t) / getTickFrequency();
        printf("Execution time = %.3f [ms]\n", t * 1000);

        imshow("Tracking", frame);
        int k = waitKey(30);
        if (k == 27)
            break;
    }

    video.release();
}
```

Daca suntem la primul frame, este apelata metoda detect_face, care parcurge frame-ul curent si cu ajutorul algoritmului Viola-Jones gaseste fetele prezente pe acest frame.

```
std::vector<bBox> detect_face(const string& window_name, Mat frame) {

    int minFaceSize = 150;
    std::vector<Rect> faces;
```

```
    std::vector<bBox> detectedFaces;
    Mat frame_gray;
    cvtColor(frame, frame_gray, CV_BGR2GRAY);
    equalizeHist(frame_gray, frame_gray);
    face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
Size(minFaceSize, minFaceSize));
    if (!faces.empty()) {
        for (int i = 0; i < faces.size(); i++)
        {
            bBox box;
            box.boundingBox = Rect2d(faces[i].x, faces[i].y, faces[i].width,
faces[i].height);
            box.boxID = i + 1;
            detectedFaces.push_back(box);
            rectangle(frame, faces[i], Scalar(0, 255, 0), 2, 8, 0);
        }
    }
    else {
        bBox wrongBox;
        wrongBox.boundingBox = Rect2d();
        wrongBox.boxID = -1;
        detectedFaces.push_back(wrongBox);
    }
    return detectedFaces;
}
```

Dupa ce avem toate fetele gasite, creez un tracker pentru fiecare fata cu metoda TrackerMOSSE::create(). Dupa aceea initializez acest tracker cu un bounding box care contine fata detectata pe frame-ul curent.

Incepand cu al doilea frame este apelata metoda update a obiectului de tracker, acesta metoda efectueaza procesul de urmarire si transmite rezultatul in al doilea parametru a acestei metoda. Daca tracker-ul gaseste fata pe frame-ul curent atunci afisez rezultatul pe video, iar daca nu gaseste fata, este apelata metoda detect_face_index care pe baza index-ului actualizeaza lista de bounding box-uri cu noile fete gasite de metoda Viola-Jones. Daca aceasta metoda gaseste fata (care nu a fost gasita de algoritmul tracking), atunci tracker-ul corespunzator acestei fete este initializat cu noul bounding box.

```
std::vector<bBox> detect_face_index(const string& window_name, Mat frame, bool
applyIf,std::vector<bBox> currentBBoxes) {

    int minFaceSize = 150;
    std::vector<Rect> faces;
    std::vector<bBox> detectedFaces;
    Mat frame_gray;
    cvtColor(frame, frame_gray, CV_BGR2GRAY);
    equalizeHist(frame_gray, frame_gray);
    face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
Size(minFaceSize, minFaceSize));
    bool condition = applyIf || (currentBBoxes.size() == faces.size());
    if (!faces.empty() && condition) {
            for (int i = 0; i < currentBBoxes.size(); i++) {
```

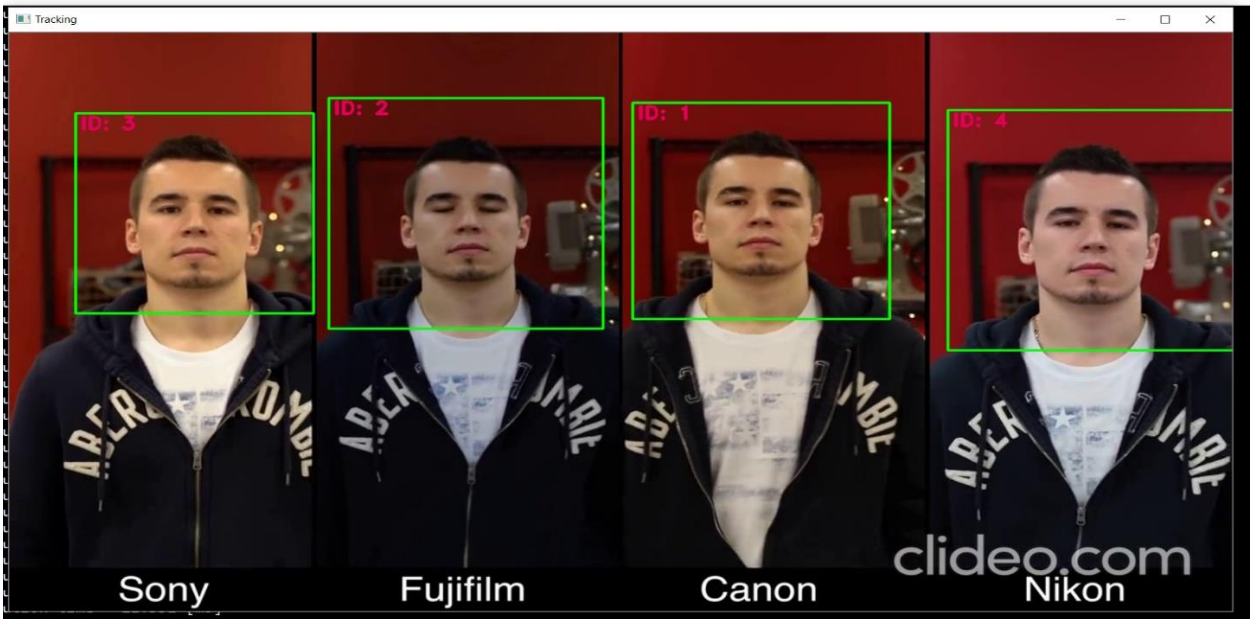
```
                int minimDist = 10000;
                int index = -1;
                for (int j = 0; j < faces.size(); j++) {
                    int diffX = abs(currentBBoxes[i].boundingBox.x - faces[j].x);
                    int diffY = abs(currentBBoxes[i].boundingBox.y - faces[j].y);
                    int eucDist = sqrt(diffX * diffX + diffY * diffY);
                    if (eucDist < minimDist) {
                        minimDist = eucDist;
                        index = j;
                    }
                }
                if (minimDist != 10000 && index != -1) {
                    bBox box;
                    box.boundingBox = faces.at(index);
                    box.boxID = currentBBoxes.at(i).boxID;
                    detectedFaces.push_back(box);
                }
            }
    }
    else {
        bBox wrongBox;
        wrongBox.boundingBox = Rect2d();
        wrongBox.boxID = -1;
        detectedFaces.push_back(wrongBox);
    }
    return detectedFaces;
}
```

La fiecare 75-lea frame am introdus o logica care actualizeaza bounding box-urile, ca sa fie aliniate corect, sa nu fie decalate. In acest caz este apelata metoda detect_face_index() care actualizeaza bounding boxurile dupa indexul (ID-ul) fetelor. Daca distanta euclidiana intre coltul stanga sus a bounding box-ului vechi si nou a aceeasi fete este mai mare decat 40, atunci bounding box-ul fetei este actualizat. Altfel nu are rost sa actualizam, deoarece diferenta ar fi prea mica si in acest fel ar creste timpul de executie a programului.

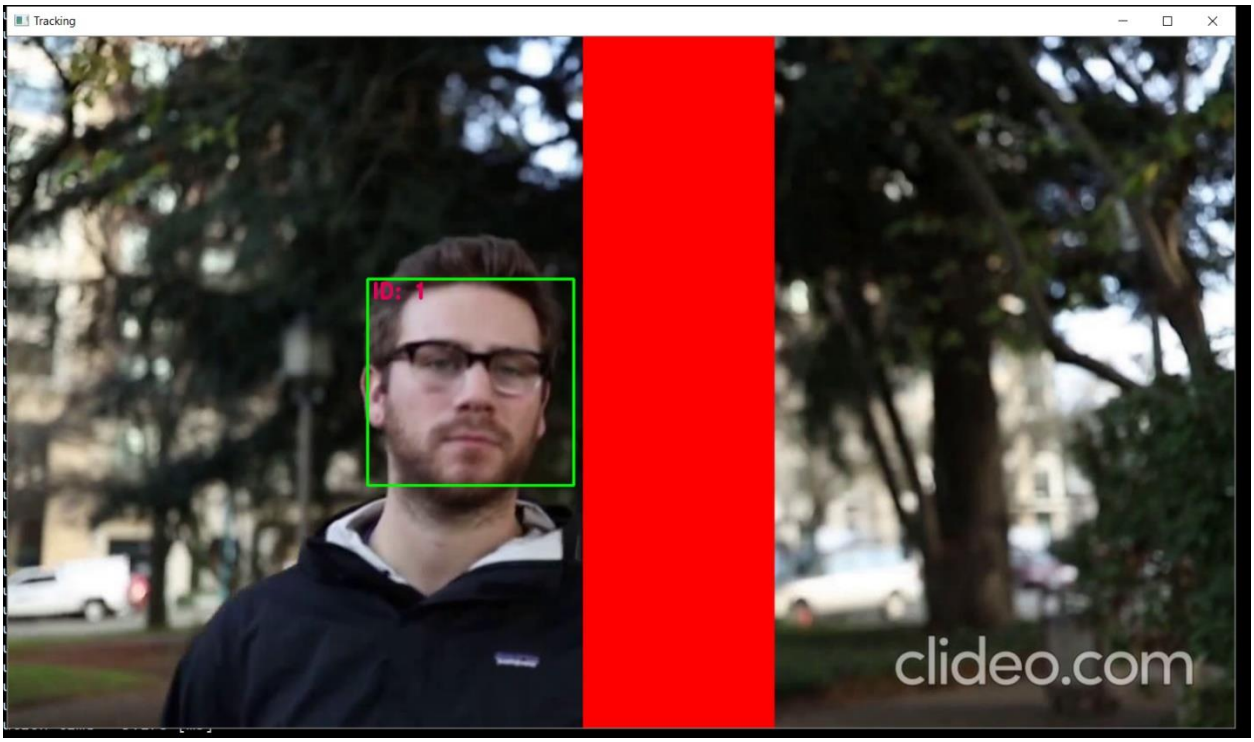
Rezultate

Acest proiect a fost testat pe mai multe video-uri, am incercat sa testez functionarea in mai multe situatii: video cu o singura fata, cu mai multe fete, situatia cand apare un obstacol in fata unei fete, etc.

Programul functioneaza in cazul mai multor fete:

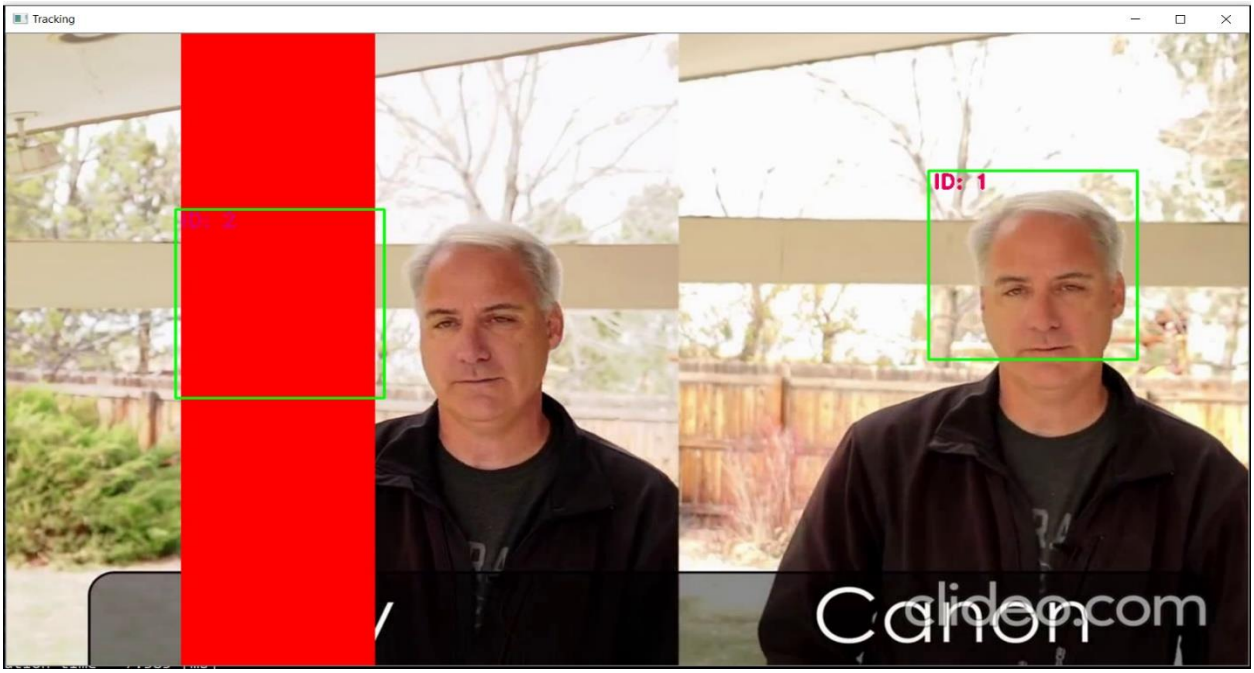


Programul gaseste fata atunci cand trece peste un obstacol:

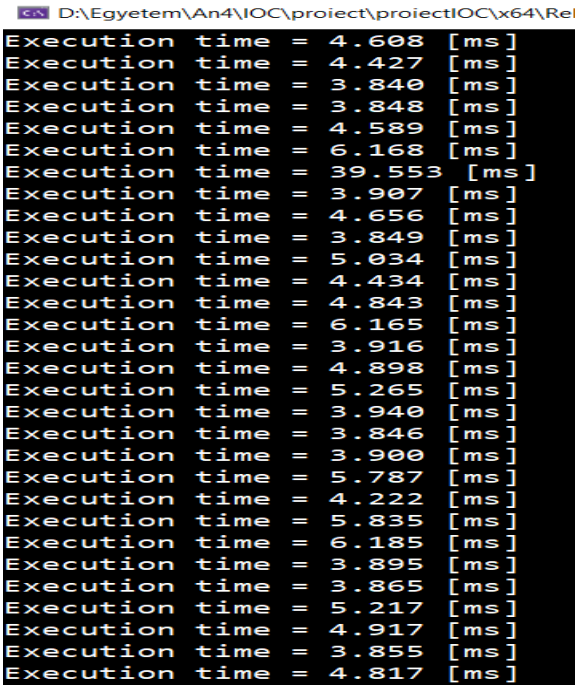


In cazurile in care exista un obstacol si functioneaza doar algoritmul de tracking si nu a ajuns la al 75-lea frame ca sa actualizeze fata, algoritmul de tracking pierde fata, nu resuseste sa urmareasca. Acest caz a fost testat si cu celelalte tipuri de algoritmi, iar rezultatele sunt:

- **Algoritmul BOOSTING**: acest algoritm nu pierde fata, reuseste sa urmareasca si dupa ce iese fata din obstacolului rosu, dar timpul de executie este mult mai mare (peste 150 de ms)
- **Algoritmul MIL**: acest algoritm nu pierde fata, urmareste si cand fata este sub obstacolul rosu, dar precizia bounding box-ului este mai slaba, iar timpul de executie este foarte mare (peste 250 de ms)
- **Algoritmul CSRT:** acest algoritm pierde fata, cand fata dispare, dispare si bounding box-ul, (nu pastreaza un boundix box ca si la cazul algoritmului MOSSE, ceea ce e prezentat pe imaginea de mai jos) iar timpul de executie este destul de mare (peste 150 de ms)



Din punct de vedere al performantei, timpul de executie este mult mai mic atunci cand functioneaza doar algoritmul de tracking, iar cand este apelata metoda Viola-Jones timpul de executie creste, din care rezulta ca trackingul este mult mai performant decat algoritmul de detection. Acest lucru se poate observa si pe imaginea urmatoare. Timpul de executie depinde de foarte multe lucruri, de exemplu daca avem un video care are un fps mai mare, timpul creste, dar diferenta de timp intre cei doi algoritmi ramane mereu.



Concluzii

In concluzie, algoritmul de tracking MOSSE, combinat cu algoritmul de detection Viola-Jones are o performanta destul de buna in timp real si impreuna resusesc sa urmareasca fata in aproapre orice situatie. Nici in acest fel nu putem sa vorbim despre un algoritm perfect, dar ajungem la un program performant, cu o acuratete buna.

Referinte

[1] - https://rria.ici.ro/wp-content/uploads/2013/06/05-VrejoiuHotaran-OK-4.pdf

[2] - https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/

[3] - https://towardsdatascience.com/convolution-vs-correlation-af868b6b4fb5