UNIVERSITATEA DIN BUCUREȘTI

Facultatea de Matematică și Informatică

Specializarea Informatică

LUCRARE DE LICENȚĂ

Urmărirea mișcării obiectelor

Coordonator

Lect. Dr. Păduraru Ciprian

Absolvent

Enică Diana - Maria

București, 2018-2019

UNIVERSITATEA DIN BUCUREȘTI

Facultatea de Matematică și Informatică

Specializarea Informatică

LUCRARE DE LICENȚĂ

Urmărirea mișcării obiectelor

Coordonator

Lect. Dr. Păduraru Ciprian

Absolvent

Enică Diana - Maria

București, 2018-2019

Cuprins

Capitolul 1. Introducere

1.1 Introducere

Scopul algoritmilor de supraveghere este de a găsi un obiect, în acest caz o persoană într-un cadru ales dintr-un video. Această supraveghere a avut loc cu succes în momentul în care am putut identifica persoana atât în cadrele anterioare ale video-ului cât și în cele de după momentul selectat. Astfel, cu un algoritm eficient, se poate determina cu ce viteză și în ce direcție se deplasează obiectul ales. Cunoscând aceste lucruri, se poate calcula cu ușurință unde se va afla modelul în cadrele viitoare ale video-ului.

În viața de zi cu zi, algoritmii de supraveghere au o vastă utilizare, aceasta fiind și motivația alegerii acestei teme. Am observat prezența acestora la scară largă în recuperarea înregistrărilor video, controlul roboților și supravegherea traficului. Astfel, consider necesară o bună înțelegere a acestor algoritmi de urmărire atât din punct de vedere al mecanismului de funcționare cât și al performanței.

Rezultatele generate de algoritmii de supraveghere pot fi destul de greu de interpretat fără o bază de date vastă a unor video-uri de test. Totuși, având la dispoziție aceste resurse, se poate realiza o paralelă interesantă între diverși algoritmi de urmărire și scenariile în care aceștia pot fi folosiți, evidențiidu-se momentele cheie pentru studiul realizat: ocluziile dintre modelele urmărite și alte modele din fundal, ieșirea obiectului din cadru, mărirea sau micșorarea modelului observat, schimbarea bruscă a direcției de deplasare a acestuia etc.

1.2 Tehnologii folosite

1.2.1 Python 3.6

Python 3.6 este un limbaj de programare de nivel înalt ce a apărut în anul 2016. Acesta este folosit în inteligența artificială fiind un limbaj multi-paradigmă (programarea orientată pe obitecte, imperativă, funcțională, procedurală, reflectivă).

Am ales această tehnologie datorită bibliotecii de metode de care dispune. Trei dintre aceste biblioteci sunt OpenCV-Python, Numpy și MatplotLib.

1.2.2 OpenCV

OpenCV este o tehnologie ce a debutat în anul 1999. „Aceasta a fost folosită pe o mașină autonomă, poreclită „Stanley", ce a câștigat în anul 2005 DARPA Grand Challenge." [8] Suportă o varietate de limbaje de programare ca și C++, Python etc., si este disponibil atât pe Windows cât și pe Linux, OS X, Android și IOS.

În prezent, OpenCV conține o multitudine de algoritmi relativi la Computer Vision și Machine Learning.

1.2.3 OpenCV-Python

OpenCV-Python este o librărie dedicată acestui limbaj de programare la baza căreia stă Numpy, o librărie optimizată pentru operațiile numerice asemănătoare sintaxei MATLAB. Astfel, toate structurile de tip matrice sau vector din OpenCV sunt convertite în vectori Numpy și invers. Prin acest proces, librăria pentru învățarea mecanică (machine learning) poate fi accesată direct prin intermediul limbajului de programare Python.

1.3 Tracking vs Detection

Pentru „Object Tracking” (urmărirea obiectelor) există mai multe clase de algoritmi care conduc la același rezultat.

1. Dense Optical Flow (flux optic dens) sunt acei algoritmi care ajută la estimarea vectorului de mișcare pentru fiecare pixel dintr-un frame extras din video.
2. Sparse Optical Flow (flux optic redus) sunt algoritmii care urmăresc locația câtorva puncte din imagine. Din această categorie face parte și algoritmul Kanade-Lucas-Tomashi (KLT).
3. Kalman Filtering (filtrarea Kalman) reprezintă un algoritm de procesare a semnalului utilizat care prezice locația unui obiect în mișcare bazându-se pe informațiilor anterioare privind deplasarea sa. „Una dintre aplicațiile acestui algoritm a fost ghidarea rachetelor. În particular, calculatorul de bord care a condus coborârea modulului lunar Apollo 11 pe lună avea un filtru Kalman.” **[10]**
4. Meanshift și Camshift sunt algoritmi folosiți la localizarea maximului unei functți dense. De asemenea, se folosesc și pentru tracking.
5. Single Object Trackers (tracker de obiecte unice) reprezintă o clasă de algoritmi unde primul cadru în care apare modelul este marcat cu un dreptunghi pentru a indica locația sa. Obiectul este apoi urmărit de-a lungul secvențelor folosind acest algoritm. În practică, acești algoritmi se folosesc împreună cu un detector de obiecte.
6. Multiple Object Track Finding Algorithms (algoritmi de gasire a obiectelor multiple) sunt utilizați în cazul în care avem o detectare rapidă a obiectelor. Logica este detectarea mai multor obiecte în fiecare cadru și apoi rularea algoritmului de găsire care identifică ce dreptunghi dintr-un cadru corespunde cu alt dreptunghi din urmatorul cadru.

„Object detection” (detectarea obiectelor) reprezintă scanarea și căutarea unui obiect într-o imagine sau într-un video, spre deosebire de „object tracking” care nu doar găsește un obiect, ci și îl urmărește, returnând pe langă poziția sa, viteza și locul în care se deplasează. Astfel, apar mai multe motive pentru care tracking-ul unui obiect este mai eficient decat detectarea lui.

În primul rand, tracking-ul este mai rapid față de simpla detectare. Motivul este simplu. Atunci cand se urmărește un obiect care a fost detectat deja în cadrele anterioare din video, se știe deja cum arată obiectul, locațiile anterioare, direcția de deplasare și viteza mișcării. În urmatoarele

cadre, folosind toate aceste informații, se poate preciza locația obiectului printr-o căutare simplă în jurul locului în care ar trebui să se afle. Localizarea va dobândi astfel eficiență maximă din punct de vedere al timpului. Un algoritm de urmărire se va folosi mereu de toate datele de care dispune până în acel moment, în timp ce unul de detectare va începe mereu căutarea de la zero. Totuși, poți pierde urma unui obiect dacă acesta trece prin spatele unui obstacol pentru o perioadă mai lungă de timp sau dacă se mișcă mai repede față de viteza de urmărire a algoritmului. O altă problemă comună a algoritmilor de tracking este faptul că dreptunghiul ce urmărește obiectul se îndepărtează ușor de el, generând erori. Această problemă se rezolvă prin rularea algoritmului de detectare. Aceștia din urmă au mai multe informații legate de clasa obiectului, fiind antrenați pe un număr mai mare de exemple de obiecte. Totuși, cei de tracking au mai multe informații specifice fiecărui obiect urmărit. În primul frame, dreptunghiul de pe poziția 16 poate fi asociat, în cea de-a doua matrice, cu cel de pe poziția 23. Astfel, folosind detectarea, nu avem nicio idee despre ce dreptunghi ii corespunde cărui obiect. Trackig-ul oferă, pe de altă parte, o modalitate de a conecta literalmente punctele.

În al doilea rând, urmărirea păstrează identitatea fiecărui model. Rezultatul detectării obiectelor este o matrice de dreptunghiuri în jurul modelului. Cu toate acestea, nu există nicio identitate atașată obiectului. De exemplu, un detector ce găsește puncte roșii, va afișa dreptunghiuri corespunzătoare tuturor punctelor pe care le-a detectat într-un cadru. În cadrul următor, va fi generată o altă matrice de dreptunghiuri

De asemenea, tracking-ul (urmărirea) poate ajuta atunci când detectarea eșuează. Un bun exemplu este momentul în care fața unei persoane urmărite este acoperită parțial. Pentru acest scenariu, un algoritm eficient este „Multiple Instance Learning” (MIL) tracker.

Capitolul 2. Algoritmi de tip „Tracking”

2.1. Object Algorithms Tracking (algoritmi de urmărire a obiectelor)

Scopul algoritmilor de urmărire nu este de a avea o înțelegere teoretică a fiecărui tracker în parte, ci de a le înțelege din punct de vedere practic.

În urmărire, obiectivul principal este găsirea unui obiect în cadrul actual, având deja obiectul urmărit cu succes în cadrele anterioare. Din moment ce a fost urmărit obiectul până în cadrul actual, se cunoaște viteza, direcția de deplasare și locația acestuia. Fără alte cunoștințe, se poate prezice noua locație pe baza modelului de mișcare actual și aceasta poate fi destul de aproape de locul în care se află noua locație a obiectului. Dar, se cunosc mai multe informații nu doar despre mișcarea obiectului. Se știe cum arată obiectul în fiecare din cadrele anterioare, deci, se poate construi un „appearance model" (model de aparență) ce poate fi folosit pentru a căuta într-o zonă și mai restrânsă a locației prezise de modelul de mișcare. Astfel se va realiza o prezicere mai exactă a locației obiectului folosindu-se cele doua modele: cel de mișcare și cel de aspect.

Dacă obiectul a fost foarte simplu și nu s-a schimbat foarte mult, se poate folosi un șablon ca model de aspect și să se efectueze căutarea după acel șablon. Cu toate acestea, în viața reală nu este atât de simplu. Aspectul unui obiect se poate modifica dramatic de la un moment la altul. Pentru abordarea acestei probleme, multe tracker-e folosesc modelul de aspect ca pe un clasificator care e antrenat într-o manieră online. Clasificatorul are misiunea de a clasifica un dreptunghi ca fiind parte din background sau obiect returnând o valoare între 0 si 1 care indică probabilitatea ca pachetul de imagini să conțină obiectul. Scorul este 0 atunci când este absolut sigur că dreptunghiul face parte din fundal și 1 atunci cand face parte din obiect.

În procesul de învățare (machine learning) se folosește cuvântul „online" pentru a sugera algoritmii care sunt antrenați la momentul rulării. Acest timp de învățare este definită ca fiind învățarea nesupervizată. Pe de alta parte, cea supervizată (offline) are nevoie de mii de exemple pentru a se antrena, comparativ cu cel online care are nevoie doar de câteva exemple.

Astfel, pentru a antrena un clasificator să detecteze persoane și mașini, acesta trebuie antrenat cu mii de imagini ce conțin oameni și vehicule și imagini ce nu conțin nici una nici alta pentru a putea diferenția ce este om sau mașina de ceea este fundal. Pe de altă parte, dacă se folosește un clasificator online, nu vor exista aceste mii de exemple de clase pozitive și negative.

2.2 Tracker-ul BOOSTING

Acest tracker este bazat pe versiunea online a algoritmului AdaBoost. „Algoritmul AdaBoost al lui Freund și Schapire a fost primul algoritm folosit în practică și rămâne unul dintre

cele mai folosite la scară largă și cele mai studiate cu aplicații în multe domenii științifice." [11] Cu toate astea, algoritmul fiind vechi, are o performanță de rulare mediocră. De asemenea, nu se poate determina momentul în care tracking-ul a eșuat.

Acest clasificator trebuie să fie instruit în timpul rulării cu exemple pozitive și negative ale obiectului. Caseta de margine inițială furnizată de utilizator este considerată ca exemplu pozitiv pentru obiect și toate celelalte pachete de cadre din exterior sunt considerate parte din fundal. Având un nou cadru, clasificatorul este rulat pe fiecare pixel din vecinătatea locului anterior și se înregistrează scorul clasificatorului. Noua locație a obiectului este cea în care punctajul este maxim. Pe măsură ce video-ul rulează, sunt furnizate noi cadre, iar clasificatorul este actualizat pe baza acestora.

Matematic vorbind, algoritmul AdaBoost parcurge un număr de T iterații. La fiecare iterație alege clasificatorul slab cel mai bun $h^t(P)$ pe care îl folosește la construcția unui clasificator f(P) mai puternic. Această funcție [11] este de forma:

$$f(P) = \sum_{t=1}^{T} a_t h^t(P) \quad [11]$$

Algoritmul BOOSTING (pseudocod) [17]

Intrare: Distribuția comună Q cu $Z = X \times \{1, -1\}$, predicția inițială $H^{(0)} \in S[H]$. În practică, Q este probabilitatea empirică a datelor de antrenare.

Pentru $m = 1, \dots, M$ se execută

1. Găsește $h^{(m)} \in H$ astfel încât: $h^{(m)} = \arg\min_{h \in H} \int_z Q(dz)\, L'\left(-yH^{(m-1)}(x)\right) I(y \neq h(x))$. Minimizarea nu trebuie sa fie exactă.
2. Găsește un coeficient $\alpha^{(m)} \in \mathbb{R}$ astfel încât: $\alpha^{(m)} = \arg\min_{\alpha \in \mathbb{R}} R_L\left(Q, H^{(m-1)} + \alpha h^{(m)}\right)$. Metoda căutării liniare sau metoda lui Newton poate fi aplicată pentru a rezolva problema de optimizare a unei singure dimensiuni.
3. Actualizarea predicției: $H^{(m)} = H^{(m-1)} + \alpha^{(m)} h^{(m)}$.

Ieșire: $H^{(M)}$predicție estimată

2.3 Tracker-ul MIL

Acest tracker este similar cu tracker-ul BOOSTING descris anterior. Diferența mare este faptul că în loc să se considere doar locația curentă a obiectului ca un exemplu pozitiv, se caută în vecinătatea exemplului pentru a genera câteva potențiale exemple pozitive. Totuși, această tehnică pare să dea greș deoarece în majoritatea acestor exemple pozitive, obiectul nu este centrat.

Aici intervine modulul de învățare a instanțelor multiple (Multiple Instance Learning - MIL) pentru salvare. În MIL, nu se specifică exemple pozitive și negative, ci grupuri pozitive și negative. Colecția de imagini acumulată în grupul celor pozitive nu este întotdeauna formată din exemple pozitive. În schimb, este suficient ca doar o imagine din această grupare să fie pozitivă. În exemplul dat, un grup pozitiv conține un patch centrat pe locația curentă a obiectului și, de asemenea, patch-uri din vecinătatea acestuia. Chiar dacă locația curentă a obiectului urmărit nu este exactă, atunci când eșantioanele din vecinătatea locului curent sunt puse în grupul pozitiv, există o bună șansă ca în această colecție să fie cel puțin o imagine în care obiectul este bine centrat.

Performanța acestui tracker este destul de bună. Aceasta nu se deplasează la fel de mult ca tracker-ul BOOSTING, dar face o muncă rezonabilă. Totuși, în versiunea OpenCV 4.0, acest tracker nu este cel mai bun din colecție.

Legat de eroarea de urmărire, aceasta nu este raportată în mod fiabil.

Algoritmul MIL (pseudocod) **[13]**

Intrare: Datele de antrenare $\{(x_1, y_1), \cdots, (x_{N_B}, y_{N_B})\}$.

Antrenare: Învățarea $C = \{c_1, \dots, c_k\}$ bazată pe setul instanțelor $x_{\cdot j}$ din toate datele de intrare.

Pentru $i = 1, \dots, N_B$ se execută:

Alege noi vectori pentru funcția de mapare

$v_i \leftarrow M_v(X_i, C)$

$v_{i\cdot 1} \leftarrow sign(v_{i\cdot 1})\sqrt{|v_{i\cdot 1}|}$

$v_i \leftarrow v_i / \|v_i\|_2$

Folosește noul set de date $\{(v_1, y_1), \cdots, (v_{N_B}, y_{N_B})\}$ pentru a învăța clasificatorul F.

Testare: Pentru toate $X_i'(i' \in \{1,2, \ldots, N_B'\})$ execută:

Alege noi vectori pentru funcția de mapare

$v_i' \leftarrow M_v(X_{i'}, C)$

$v_{i\cdot 1}' \leftarrow sign(v_{i'\cdot 1})\sqrt{|v_{i'\cdot 1}|}$

$v_i' \leftarrow v_i'/\|v_i'\|_2$

Ieșire: Predicția $F(v_i')$

2.4 Tracker-ul KCF

Acest tracker a aparut în versiunea OpenCV 3.1 și este cel mai bun chiar și în versiunea OpenCv 4.0. Precizia, viteza acestuia și raportarea eșecurilor de urmărire sunt cele mai bune.

Tracker-ul a fost propus din necesitatea unei viteze mai bune de antrenare și învățare. Acesta poate rula pe orice calculator la o viteză de 300-450 fps deoarece nu necesită o antrenare offline ci, din contră, rețeaua învață online, parcurgând fiecare cadru din video.

Prescurtarea KCF (Kernelized Correlation Filter) înseamnă filtre de corelație kernel-izate. Acest tracker se bazează pe ideile prezentate anterior în celelalte două tracker-e. KCF se folosește de faptul că mostrele multiple pozitive folosite în cadrul tracker-ului MIL au suprafețe mari comune. Aceste date care se suprapun conduc la câteva proprietăți matematice frumoase care sunt exploatate pentru a realiza un tracking mai rapid și mai precis.

Algoritmul KCF (pseudocod) [3]

Intrare: Dreptunghiul inițial de urmărire (x_0, y_0, s_0), conturul cadrului f_C, ciclul complet al filtrului de scalare $n = 5$

Ieșire: Dacă $f_C \ \% \ n = 0$ (Condiția 1) atunci:

Starea Estimată a Țintei (x_t, y_t, s_t), Modelul Filtrul de Scalare R_s

Altfel dacă $f_C \,\%\, n > 0 \; and \; f_C \,\%\, n \leq n/2$ (Condiția 2) atunci:

Starea Estimată a Țintei $(x_t, y_t, s_t = s_{t-1})$, Modelul Filtrul de Translatare a unei zone mari R_t^L

Altfel (Condiția 3):

Starea Estimată a Țintei $(x_t, y_t, s_t = s_{t-1})$, Modelul Filtrul de Translatare a unei zone mici R_t^S

Funcția track $(x_{t-1}, y_{t-1}, s_{t-1})$

Tranzitează Filtrului de Particule către cadrul t și calculul mediei (x_t, y_t, s_{t-1})

Decupează ROI pentru R_t^L (Condiția 2), sau R_t^S (Condiția 3) fiind dat (x_t, y_t) și estimează o nouă poziție $(x_t, y_t) = \max(yR_t)$

Scalarea pentru R_s: $S = \{1.25, 1.0, 1/1.05\}$

Decupează ROI pentru R_s^i (Condiția 1), estimează factorului de scalare $\alpha = \arg\max_{i \in S} \left(PSR\left(y_{R_S^i} \right) \right)$, calculează o nouă scalare $s_t = \alpha * s_{t-1}$

Dacă este necesar, creează noi eșantioane și calculează media (x_t, y_t)

Actualizează R_t^S (Condiția 3)

Actualizează R_t^L (Condiția 2)

Dacă $PSR(y_{R_s}) \geq T_{R_s}$ atunci actualizează R_s (Condiția 1)

Întoarce (x_t, y_t, s_t)

2.5 Tracker-ul TLD

Prescurtarea TLD (Tracking, Learning and Detection) înseamnă urmărire, învățare și detectare. După cum sugerează și numele, acest tracker descompune sarcina de urmărire pe termen lung în trei componente pe termen scurt: urmărire, învățare și detectare. „Tracker-ul urmărește obiectul din cadru în cadru. Detectorul localizează toate aparițiile care au fost observate până în

acel moment și corectează tracker-ul dacă este necesar. Învățarea evaluează erorile detectorului și îl actualizează pentru a evita producerea acestora în viitor." [5] Datele de ieșire are tracker-ului tind spre haotic. De exemplu, dacă este urmărit un pieton și în scenă există și alți pietoni, acest tracker poate uneori să urmărească temporar un pieton diferit de cel ales inițial. Pe de o parte, această abordare pare să urmărească un obiect pe o scară mai mare. Dacă s-a ales o secvență video în care obiectul este ascuns în spatele unui alt obiect, acest tracker poate fi o alegere bună.

Tracker-ul funcționează cel mai bine pe mai multe cadre și urmărește cel mai bine modificările la scară largă. Cu toate astea, generează o mulțime de fals pozitiv care îl fac aproape inutilizabil.

Algoritmul TLD (pseudocod) – acesta este format din 5 algoritmi mai mici, fiecare având rolul lui bine definit.

Funcția „track" urmărește obiectul ales, funcția „learn" ajută la învățarea rețelei, funcția „fuse" fuzionează rezultatele, funcția „detect" găsește modelele pe toata suprafața cadrului și funcția „valid" validează următorul pas al algoritmului. Toate aceste funcții sunt descrise în Figurile 2.5.1, 2.5.2, 2.5.3, 2.5.4, 2.5.5 și în Figura 2.5.6 este funcția principală care le leagă alcătuind tracker-ul TLD.

Input: B_I, I, J
$p_1 \dots p_n \leftarrow \text{generatePoints}(B_I)$
for all p_i **do**
$p_i' \leftarrow LK(p_i)$
$p_i'' \leftarrow LK(p_i')$
$\varepsilon_i \leftarrow |p_i - p_i''|$
$\eta_i \leftarrow NCC(W(p_i), W(p_i'))$
end for
$med_{NCC} \leftarrow \text{median}(\eta_1 \dots \eta_n)$
$med_{FB} \leftarrow \text{median}(\varepsilon_1 \dots \varepsilon_n)$
if $med_{FB} > \theta_{FB}$ **then**
$B_J = \emptyset$
else
$C \leftarrow \{(p_i, p_i') \mid p_i' \neq \emptyset, \varepsilon_i \leq med_{FB}, \eta_i \geq med_{ncc}\}$
$B_J \leftarrow \text{transform}(B_I, C)$
end if

Figura 2.5.1 – Funcția *track* [7]

Input: I
Output: F
$F \leftarrow 0$
for $i = 1 \dots S$ **do**
$F \leftarrow 2 \times F$
if $I(d_{i,1}) < I(d_{i,2})$ **then**
$F \leftarrow F + 1$
end if
end for

Figura 2.5.2 – Funcția *valid* [7]

```
1: D_t ← ∅
2: F ← foreground(I);
3: I' ← integralImage(I);
4: I'' ← integralImage(I^2);
5: for all B ∈ R do
6:    if isInside(B, F); then
7:       if calcVariance(I'(B), I''(B)) > σ²_min then
8:          if classifyPatch(I(B)) > 0.5 then
9:             P ← resize(I(B), 15, 15)
10:            if matchTemplate(I(B)) > θ⁺ then
11:               D_t ← D_t ⋃ B
12:            end if
13:         end if
14:      end if
15:   end if
16: end for
17: D_t ← cluster(D_t)
```

Figura 2.5.3 – Funcția *detection* [7]

```
Input: R_t, D_t
Output: B_t
1:  B_t ← ∅
2:  valid(B_t) ← false
3:  if R_t ≠ ∅ then
4:     if |D_t| = 1 ∧ p⁺_{D_t} > p⁺_{R_t} then
5:        B_t ← D_t
6:     else
7:        B_t ← R_t
8:        if p⁺_{R_t} > θ⁺ then
9:           valid(B_t) ← true
10:       else if valid(B_{t-1}) ∧ p⁺_{R_t} > θ⁻ then
11:          valid(B_t) ← true
12:       end if
13:    end if
14: else if |D_t| = 1 then
15:    B_t ← D_t
16: end if
```

Figura 2.5.4 – Funcția *fuse* [7]

Input: I, B_t

for all $B \in \mathcal{R}$ **do**
  if overlap$(B, B_t) > 0.6$ **and** classifyPatch$(I(B)) < 0.5$ **then**
    for $k = 1 \ldots M$ **do**
      $F \leftarrow$ calcFernFeatures$(I(B), k)$
      $p_{F_k}[F] \leftarrow p_{F_k}[F] + 1$
    end for
  else if overlap$(B, B_t) < 0.2$ **and** classifyPatch$(I(B)) > 0.5$ **then**
    for $k = 1 \ldots M$ **do**
      $F \leftarrow$ calcFernFeatures$(I(B), k)$
      $n_{F_k}[F] \leftarrow n_{F_k}[F] + 1$
    end for
    if $p^+_{B_t} > \theta^-$ **then**
      $\mathcal{P}^- \leftarrow \mathcal{P}^- \cup I(B)$
    end if
  end if
end for
if $p^+_{B_t} < \theta^+$ **then**
  $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup I(B_t)$
end if

Figura 2.5.5 – Funcția *learn* [7]

Input: $I_1 \ldots I_n, B_1$

learn(I_1, B_1)
for $t = 2 \ldots n$ **do**
  $R_t \leftarrow$ track(I_{t-1}, I_t, B_{t-1})
  $D_t \leftarrow$ detect(I_t)
  $B_t \leftarrow$ fuse(R_t, D_t)
  if valid(B_t) **then**
    learn(I_t, B_t)
  end if
  print(B_t)
end for

Figura 2.5.6 – Funcția *principală* (main) [7]

2.6 Tracker-ul MEDIANFLOW

Acest tracker urmărește obiectul atât în direcția înainte cât și înapoi și măsoară discrepanțele dintre aceste două traiectorii. Minimizarea acestei erori de tip „ForwardBackward” permite detectarea în mod fiabil a eșecurilor de urmărire și selectează traiectoriile de încredere în secvențele video.

În testele efectuate, s-a descoperit că acest tracker funcționează cel mai bine atunci când micșorarea este predictibilă și mică. Spre deosebire de alte trackere care continuă să funcționeze chiar și atunci când urmărirea a eșuat în mod clar, tracker-ul MEDIANFLOW știe cu exactitate când urmărirea a eșuat.

Un dezavantaj al acesui tracker este eșuarea în cazul mișcărilor ample. Totuși, raportează cu exactitate eșecurile de urmărire și funcționează foarte bine atunci când mișcarea este previzibilă și fără a exista nicio anomalie.

Algoritmul MEDIANFLOW (pseudocod) – o nouă variantă a algoritmului Forward – Backward – este prezentat în Figura 2.5.7.

input: sentence $\bar{\mathbf{x}}$, parameters θ
Forward pass: Compute the forward probabilities
Initialization
for $y_l \in \mathbf{Y}$ **do**
  $\alpha_1(y_l) = \phi_1(y_l)$
end for
for $i = 2$ **to** N **do**
  for $y_l \in \mathbf{Y}$ **do**
    $\alpha_i(y_l) = \left[\sum_{m \in \mathbf{Y}} \phi_{(i-1)}(m, l)\alpha_{i-1}(y_m)\right] \phi_i(l)$
  end for
end for
Backward pass: Compute the backward probabilities
Initialization
for $y_l \in \mathbf{Y}$ **do**
  $\beta_N(y_l) = 1$
end for
for $i = N - 1$ **to** 1 **do**
  $\beta_i(y_l) = \sum_{y_m \in \mathbf{Y}} \phi_i(l, m)\phi_{i+1}(m)\beta_{i+1}(y_m).$
end for
output: The forward and backward probabilities α and β

Figura 2.6.1 – Pseudocodul algoritmul *MEDIANFLOW* [16]

2.7 Tracker-ul GOTURN

Denumirea GOTURN (Generic Object Tracking Using Regression Networks) sugerează exact procesul descris de acest algoritm: urmărirea modelelor folosind rețele de regresie.

Dintre toți algoritmii de urmărire, acest tracker este singurul bazat pe Rețeaua Neuronală Convoluțională (CNN). GOTURN preia toate avantajele unei astfel de rețele și le îmbunătățește

performanța deoarece se bazazează pe o învățare pasivă (offline) folosind un set de date static. Acesta este foarte bun în urmărirea modificărilor punctului de vedere, schimbărilor de lumină și deformărilor, totuși nu se descurcă bine cu eventualele anomalii care apar.

Algoritmul GOTURN are la baza metoda coborârii pe gradient, lucru ilustrat și prin pseudocodul din Figura 2.7.1.

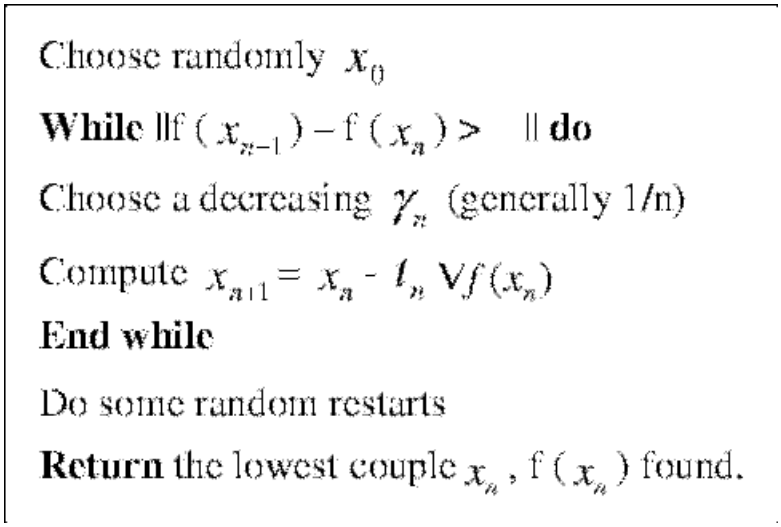


Figura 2.7.1 – Pseudocodul algoritmului GOTURN [6]

2.8 Tracker-ul MOSSE

Prescurtarea MOSSE (Minimum Output Sum of Squared Error) înseamnă suma minimă a erorii pătratice. Acest tracker utilizează corelația adaptivă pentru urmărirea obiectului, ceea ce produce filtre de corelare stabile atunci când este inițializat folosind un singur cadru. MOOSE este foarte bun în variațiile de deformare ale iluminării, dimensiunii, poziției și rigidității. De asemenea, detectează anomalii bazate pe raportul de la vârf la maxim local care permite urmăririi să se întrerupă și să se reia de unde s-a oprit atunci când obiectul reapare în cadru. Altfel spus, în momenul în care un obiect blochează obiectul urmărit (anomalie), tracker-ul nu va da greș.

MOOSE se descurcă bine și la o viteză mai mare (450 fps și chiar mai mult), este ușor de implementat și totuși la fel de precis ca și alte tracker-e. Pe o scară de performanță, acesta este la finalul clasamentului fiind depășit de tracker-e bazate pe „deep learning” (învățarea profundă).

Algoritmul MOSSE (implementare)

Inputs
- x: training image patch, $m \times n \times c$
- y: regression target, Gaussian-shaped, $m \times n$
- z: test image patch, $m \times n \times c$

Output
- responses: detection score for each location, $m \times n$

```
function alphaf = train(x, y, sigma, lambda)
  k = kernel_correlation(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detect(alphaf, x, z, sigma)
  k = kernel_correlation(z, x, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = kernel_correlation(x1, x2, sigma)
  c = ifft2(sum(conj(fft2(x1)) .* fft2(x2), 3));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2 * c;
  k = exp(-1 / sigma^2 * abs(d) / numel(d));
end
```

Figura 2.8.1 – Implementare MATLAB a algoritmului *MOSSE* [9]

2.9 Tracker-ul CSRT

Algoritmul CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability Tracker) a pornit de la urmărirea pe termen scurt a modelelor la care filtrele de corelare distrimantivă (DCF) au demonstrat performanțe incredibile. La acestea au fost adaugate concepte precum canal și fiabilitate spațială rezultând astfel algoritmul CSRT.

Acest tracker este mai rapid față de tracker-ul KCF, deși viteza video-ului asupra căruia se face urmărirea (FPS) va fi foarte mică.

Require:

Image $\mathbf{I}_t$, object position on previous frame $\mathbf{p}_{t-1}$, scale s_{t-1}, filter $\mathbf{h}_{t-1}$, color histograms $\mathbf{c}_{t-1}$, channel reliability $\mathbf{w}_{t-1}$.

Ensure:

Position $\mathbf{p}_t$, scale s_t and updated models.

Localization and scale estimation:

1: New target location $\mathbf{p}_t$: position of the maximum in correlation between $\mathbf{h}_{t-1}$ and image patch features $\mathbf{f}$ extracted on position $\mathbf{p}_{t-1}$ and weighted by the channel reliability scores

2: Using location $\mathbf{p}_t$, estimate new scale s_t.

Update:

3: Extract foreground and background histograms $\tilde{\mathbf{c}}^f$, $\tilde{\mathbf{c}}^b$.

4: Update foreground and background histograms $\mathbf{c}_t^f = (1-\eta_c)\mathbf{c}_{t-1}^f + \eta_c\tilde{\mathbf{c}}^f$, $\mathbf{c}_t^b = (1-\eta_c)\mathbf{c}_{t-1}^b + \eta_c\tilde{\mathbf{c}}^b$.

5: Estimate reliability map $\mathbf{m}$

6: Estimate a new filter $\tilde{\mathbf{h}}$ using $\mathbf{m}$

7: Estimate channel reliability $\tilde{\mathbf{w}}$ from $\mathbf{h}$

8: Update filter $\mathbf{h}_t = (1-\eta)\mathbf{h}_{t-1} + \eta\tilde{\mathbf{h}}$.

9: Update channel reliability $\mathbf{w}_t = (1-\eta)\mathbf{w}_{t-1} + \eta\tilde{\mathbf{w}}$.

Figura 2.9.1 – Pseudocodul algorimului *CSRT* [1]

Capitolul 3. Evaluarea

3.1 Baza de date folosită și modul de evaluare al algoritmilor

A fost ales setul de date oferit de Universitatea Stanford numit Stanford Drone Dataset[12] (SDD). Această bază de date constă în 8 scene diferite, fiecare având un număr diferit de video-uri. Aceste scene sunt: „gates”, „little”, „nexus”, „coupa”, „bookstore”, „deathCircle”, „quad” și „hyang”.

3.2 Evaluarea performanței algoritmilor

Considerând scena „bookstore”, în urma analizei algoritmilor de urmărire, s-au constatat următoarele rezultate și nereguli. Pentru o analiză corectă au fost alese aceleași modele din același cadru.

Tracker-ul BOOSTING este bazat pe N exemple de antrenare $\{X_i, y_i\}_1^N$ unde $x_i \in R^k$ și $y_i \in \{-1, +1\}$. Rezultatul returnat de algoritmul BOOSTING este codat ca fiind -1 sau +1. Scopul acestui algoritm este de a minimiza pierderea exponențială: $\mathcal{L} = \Sigma_i \exp\{-y_i H(x_i)\}$ [11]. Într-un mod iteal problema s-ar rezolva pentru toți parametri simultan. Totuși o astfel de optimizare nu ar fi practică. Astfel, majoritatea algoritmilor de tip BOOSTING caută parametri optimi adăugând câte un learner pe rând. Fie $H_{m-i} = \sum_{t=1}^{m-1} \alpha_t h(\cdot\,;\, a_t)$ [11] cel mai puternic model pentru primii m-1 cei mai slabi learneri. Regula de actualizare pentru minimizarea pierderii rezultatului celui mai puternic model va fi: $a_m = \underset{a}{\operatorname{argmin}}\, \mathcal{L}(H_{m-1} + \alpha_m h(\cdot;\, a) | \{x_i, y_i\}_1^N)$ [11].

Algoritmul BOOSTING are o viteză mică, se descurca la urmărirea modelelor precum se poate observa în Figura 3.2.1, dar nu și în condiții mai dificile precum ocluziile.

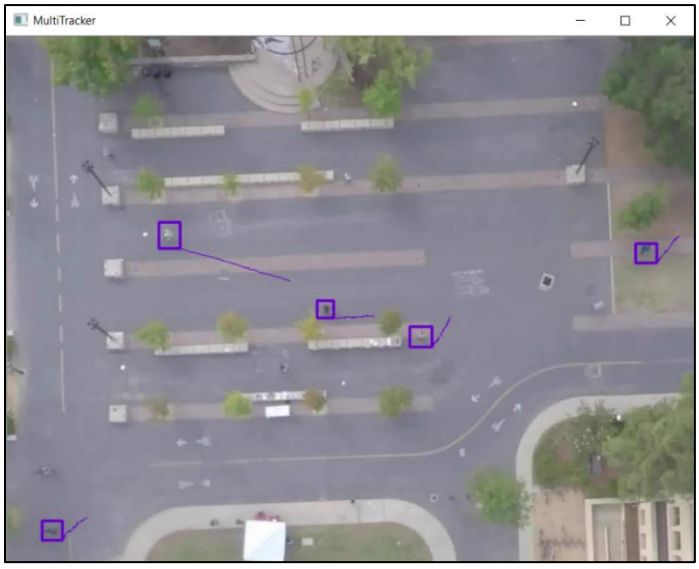


Figura 3.2.1 – Algoritmul BOOSTING la începutul urmăririi

Legat de rata de detectare a eșecului, algoritmul are o performanță slabă nefiind capabil să returneze eșuarea în detectare.

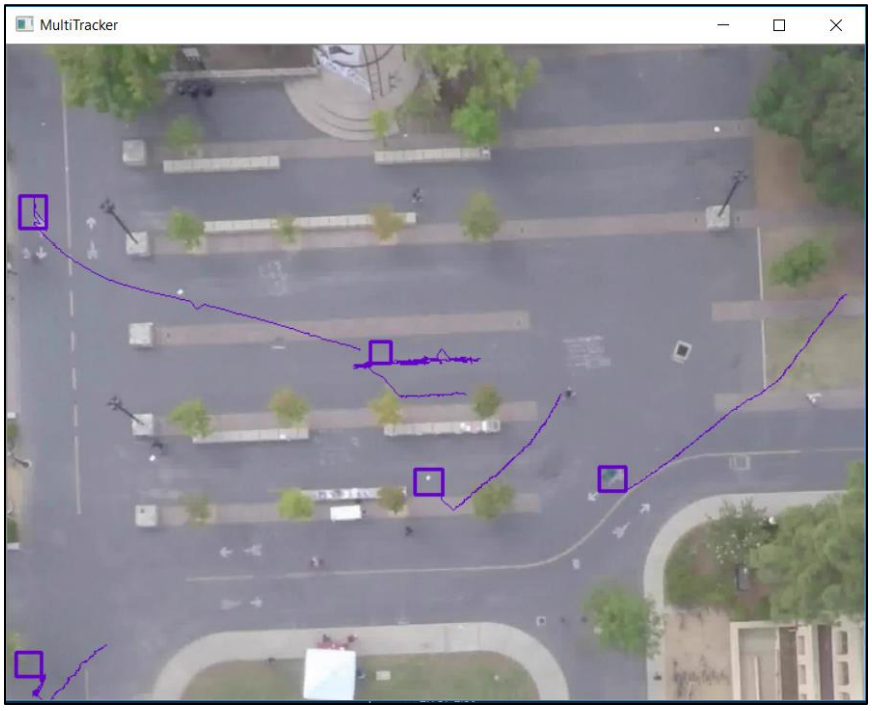


Figura 3.2.2 – Algoritmul BOOSTING

În Figura 3.2.2 se poate observa pierderea modelelor urmărite de către algoritmul BOOSTING, iar detectarea eșecului survenit în urma acesteia lipsește.

Formal, analizând algoritmul MIL, se oferă un set de date de antrenare $\{X_i, y_i\}_1^N$, unde $X_i = (x_{i1}, x_{i2}, \dots)$ este a i-a instanță și $y_i \in \{0,1\}$ este eticheta instanței. Această problemă poate beneficia de o abordare online, deoarece seturile de date de tip MIL tind să fie foarte mari. În comparație cu algoritmul BOOSTING, MIL va returna rezultatul sub forma a două valori 0 sau +1.

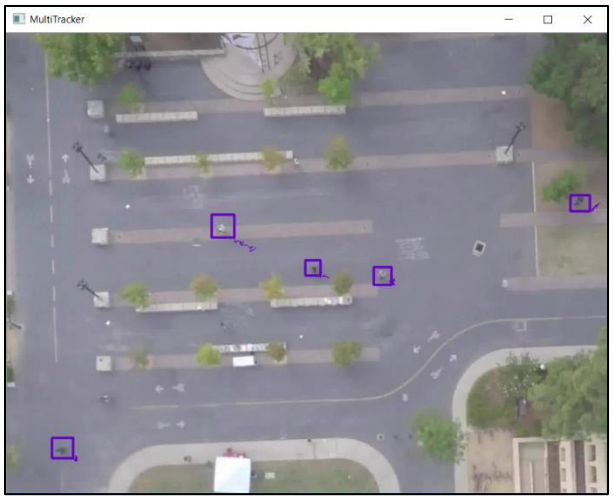
Probabilitatea ca o instanță să fie pozitivă este $p_i(H) = 1 - \prod_j \left(1 - \sigma\left(H(x_{ij})\right)\right)$ [2], unde $\sigma(.)$ reprezintă funcția sigmoidă. Această ecuație este bazată pe formula Noisy-OR. Funcția de pierdere [2] este dată de răspunsurile negative probabil pentru fiecare instanță:

$$\mathcal{L}(\mathrm{H}|\{x_i, y_i\}_1^N) = -\sum_{i=1}^{N} y_i \log p_i(H) + (1 - y_i) \log\left(1 - p_i(H)\right)$$

Pentru varianta online algoritmul primește câte un exemplu pe rând folosind regula de actualizate: $a_m \leftarrow a_m - \eta_i \frac{p_i - y_i}{p_i} \sum_j p_{ij} \frac{\partial h(x_{ij}; a)}{\partial a} |_{a_m}$ [2].

Observând comportamentul algoritmului asupra locatiei „bookstore”, se remarcă faptul că dreptunghiul generat de algoritmul MIL nu ramâne fixat pe model. Acesta încurcă modelul cu fundalul destul de des, iar dacă obiectul urmărit se apropie de alt obiect, algoritmul va continua urmărirea celui de-al doilea obiect.

Comparativ cu algoritmul BOOSTING, tracker-ul MIL are o viteză la fel de mică, pierde modelele în timpul urmăririi și de asemenea, lipsește detectarea eșecului apărut.



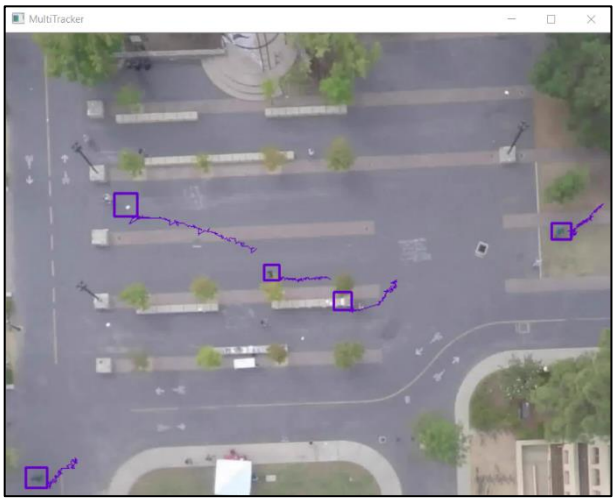


Figura 3.2.3 și Figura 3.2.4 – Algoritmul MIL

În Figura 3.2.3 se observă debutul urmăririi modelelor de către tracker-ul MIL. Inițial urmărirea are loc cu succes.

Pe parcursul urmăririi, se observă în Figura 3.2.4 pierderea modelelor și confundarea acestora cu fundalul, cel din urmă fiind urmărit în continuare. Dintr-un total de cinci modele urmărire simultan, tracker-ul rămâne în final cu trei obiecte observate în mod corect.

Totuși, situația se înrăutățește în momentul în care pe lângă unul dintre modelele urmărite, trece o persoana care face parte din fundal. În această situație comportamentul dorit ar fi păstrarea obiectului inițial urmărit și continuarea observării acestuia. Algoritmul MIL nu va face acest lucru, din contră, va începe observarea celuilalt obiect. Acest lucru se poate observa inițial în Figura 3.2.4 când în urmărire se află persoana cu o vestimentație verde din dreapta, iar în Figura 3.2.5, în chenarul de observație, se află altă persoană îmbrăcată în alb.

Algoritmul MIL ajunge în final la eșec prin faptul că dintr-un total de cinci modele urmărite, nu se păstrează niciunul, lucru care nu va fi detectat cu succes.

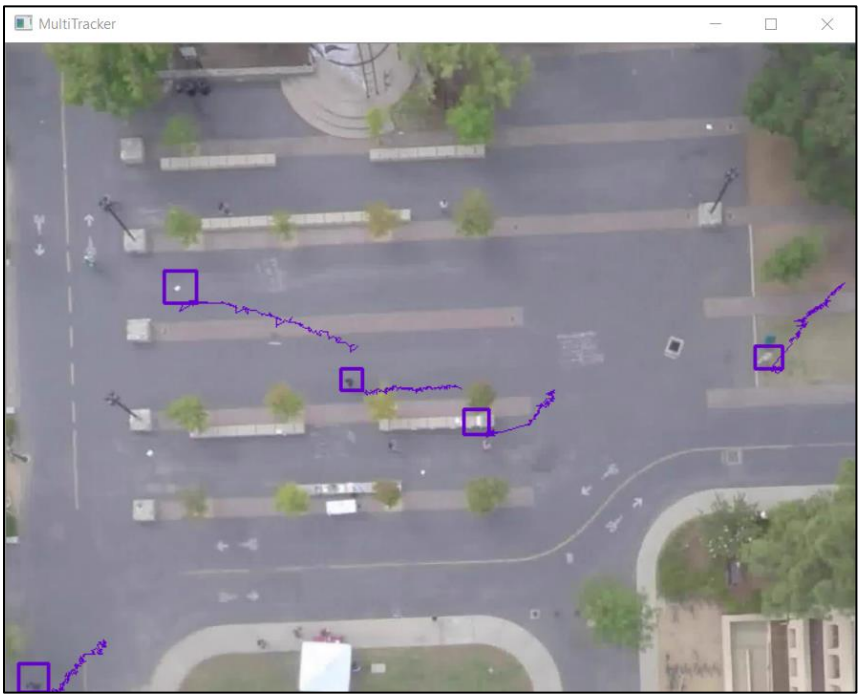


Figura 3.2.5 – Algoritmul MIL – schimbarea modelului de urmărire

Tracker-ul KCF este foarte rapid, dar poate pierde foarte ușor modelele, acesta urmărind în final fundalul. De asemenea, precum algoritmii BOOSTING și MIL, nu detectează eșecul din urmărire.

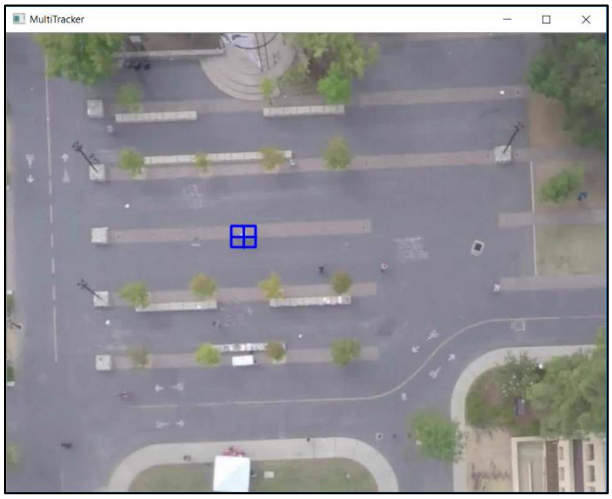
Formal, KCF antrenează rețeaua neuronală cu scopul de a găsi funcția: $f(z) = w^T z$ ce minimizează eroarea pătratică asupra datelor x_i și regresiei y_i: $\min_w \sum_i (f(x_i) - y_i)^2 + \lambda \|w\|^2$[9]. Parametrul λ este unul de regularizare ce controlează supraînvățarea. Minimizarea este de forma

$w = (X^T X + \lambda I)^{-1} X^T y$ [9] unde matricea X are pe fiecare rând datele de antrenare x_i și y reprezintă regresiile corespunzătoare. I este matricea identitate.

Din testele efectuate s-a remarcat faptul că încă din primele secunde de supraveghere multiplă a obiectelor, algoritmul KCF eșuează. Rapiditatea sa, deși pare să fie un punct forte, se dovedește în final a fi un mare dezavantaj. Urmărirea va eșua drastic în toate cazurile testate. Algoritmul în final va avea un număr foarte mare de ecuații liniare de rezolvat pentru a returna soluția optimă, iar acest lucru poate fi copleșitor într-un mediu de test bazat pe exemple reale.

S-a continuat testarea algoritmului mărindu-se chenarele de supraveghere, dar fără succes. Totuși, se remarcă reușita pe termen scurt a tracker-ului asupra unui singur model. Ca exemplu de urmărire a fost vizat biciclistul din Figura 3.2.6.

Acesta a fost urmărit cu succes precum se poate vedea in Figura 3.2.7, dar se observă în mod clar faptul că acest model a fost confundat cu fundalul, lucru care nu a fost semnalat de algoritm ca fiind un eșec.



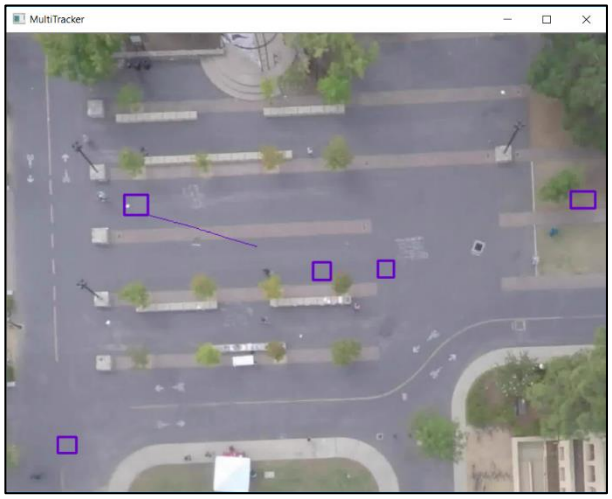


Figura 3.2.6 și Figura 3.2.7 – Algoritmul KCF

În concluzie, algoritmul KCF aplicat pe mai multe obiecte simultan nu este întotdeauna eficient, dar dacă totuși se dorește utilizarea acestuia, este de preferat alegerea unui cadru prielnic testării.

În comparație cu algoritmii prezentați anterior, tracker-ul TLD are o viteză de urmărire medie și este predispus la detectare de fals-pozitiv. Totuși, se descurcă mai bine decât algoritmul

KCF care ratează detectarea încă din primele momente.

Denumirea algoritmului TLD (Tracking-Learning-Detection) reprezintă exact modul de lucru al acestuia: urmărire, învățare și detectare. Tracker-ul estimează urmărirea mișcării obiectelor de la un frame la altul, și Detectarea tratează fiecare cadru din video în mod independent realizând o scanare a întregii imagini și , Învățarea reprezintă procesul realizat de rețeaua neuronală bazat pe totalitatea datelor (urmărire și detectare).

Formal, fie x un model din spațiul modelelor X și y o etichetă din mulțimea $y = \{-1,1\}$. Scopul rețelei este de a învăța un clasificator $f: X \to Y$ pe baza seturilor de date (X_i, Y_i) și să-i crească performanța pentru fiecare X_u neetichetat. La o anumită iterație k a algoritmului, clasificatorul a etichetat toate celelalte date anterioare: $y_u^k = f(x_u|\theta^{k-1})$ pentru toate $x_u \in X_u$. [15]

Erorile de tip fals-pozitiv și fals-negativ sunt calculate pe baza următoarelor doua formule:

$$\alpha(k+1) = \alpha(k) - n_c^-(k) + n_f^+(k) \quad [15] \qquad \beta(k+1) = \beta(k) - n_c^+(k) + n_f^-(k) \quad [15]$$

Tracker-ul TLD este predispus la detectare de fals-pozitiv încă din primele momente de supraveghere. Viteza de urmărire este medie, dar aceasta nu mai contează în momentul în care dintr-un total de cinci cadre, acesta reușește să urmărească un singur obiect, pe care în final îl va confunda cu fundalul. Se poate observa în Figura 3.2.8 faptul că au fost selectate inițial cinci obiecte pentru a fi urmărite, aceleași luate ca exemplu și pentru tracker-ele anterioare.

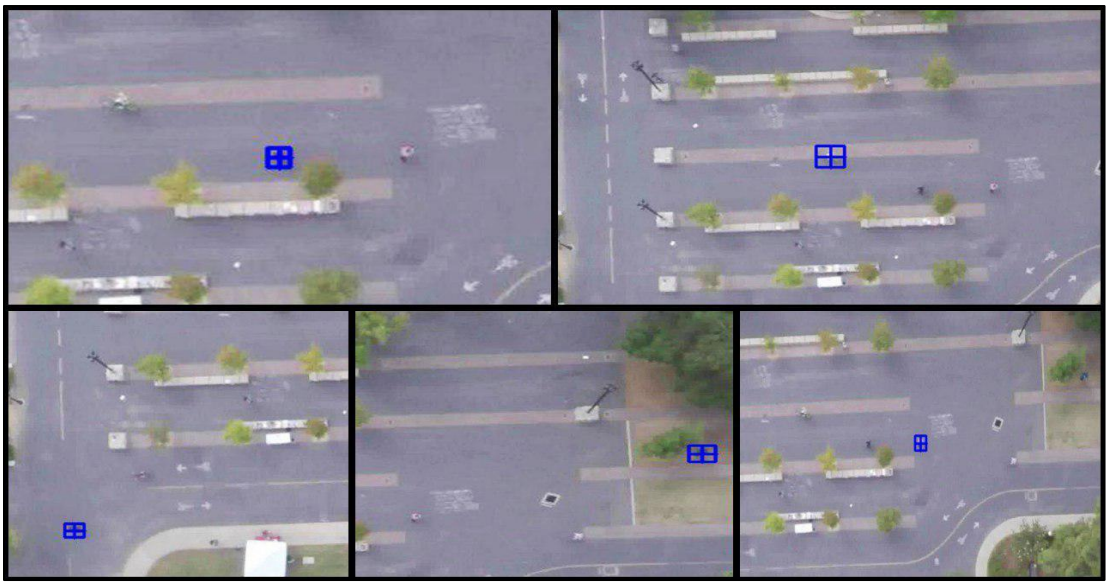


Figura 3.2.8 – Modelele alese pentru urmărirea cu algoritmul TLD

În momentul începerii urmăririi celor cinci modele, tracker-ul va urmări doar patru dintre acestea și chiar și în acest caz va eșua drastic.

În Figura 3.2.9 este evidențiat comportamentul algoritmului în cel mai clar mod. Acesta reușește să urmărească obiectele alese, totuși consideră fundalul ca fiind unul dintre modele. Astfel, tracker-ul TLD are cea mai mare rată de detectare a falsului pozitiv.

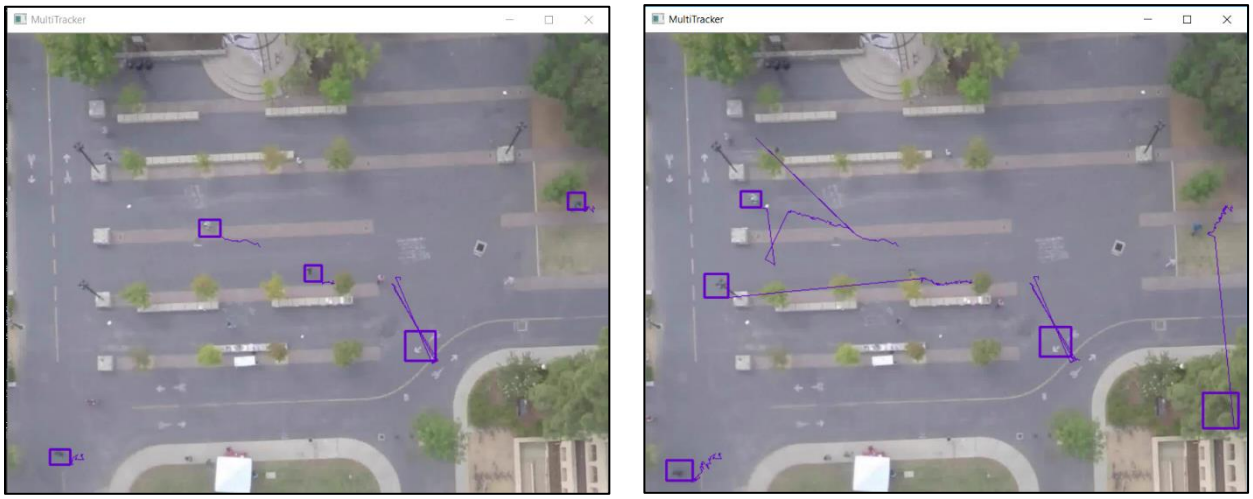


Figura 3.2.9 și Figura 3.2.10 – Algoritmul TLD

În Figura 3.2.10 se observă în mod foarte clar, felul în care algoritmul TLD acționează. Acesta urmărește cu succes modelele alese, iar în clipa în care a pierdut obiectul, acesta îl caută folosind detectarea, lucru care duce la o traiectorie neuniformă. Obietul din stânga imaginii este cel mai bun exemplu în acest sens. În cazul în care devierea este mult mai mare, traiectoria devine foarte bruscă, iar fundalul ajunge detectat ca fiind obiectul urmărit inițial. Algoritmul ajunge la eșec, deoarece predomină fals pozitivul.

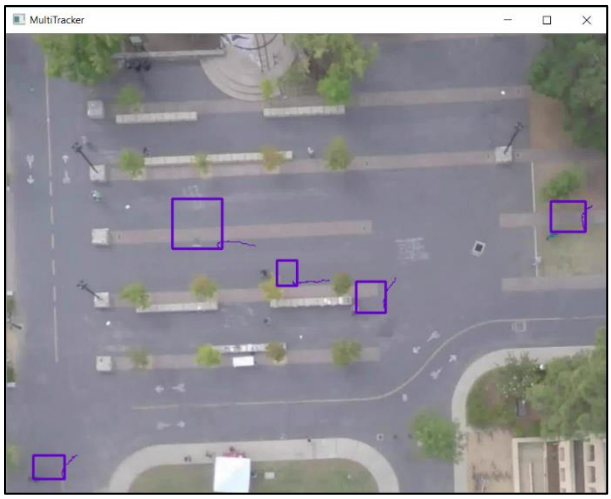
În comparație cu celelalte tracker-e, algoritmul TLD se află la coada clasamentului din toate punctele de vedere: viteză, detectarea eșecului de urmărire și urmărirea cu succes a modelului pe mai multe cadre.

Tracker-ul MEDIANFLOW are o viteză impresionantă în urmărire și reușește să supravegheze cu succes modelele în primele momente, dar la un moment dat, eșuează dramatic în

toate cele cinci cazuri de urmărire. Algoritmul rămâne blocat pe fundal având rata de eșec destul de mare, dar nu mai mare decât tracker-ul TLD sau tracker-ul KCF.

Un alt algoritm de urmărire este MEDIANFLOW, la baza căruia stă algoritmul FB(Forward-Backward). Fie $S = (I_t, I_{t+1}, \dots, I_{t+k})$ o secvență de imagini și x_t un punct localizat în timpul t. Dacă se folosește un tracker oarecare, punctul x_t va fi urmărit pe durata a k pași. Traiectoria rezultată va fi: $T_f^k = (x_t, x_{t+1}, \dots, x_{t+k})$ [14], unde f semnifică deplasarea „înainte" și k reprezintă lungimea. Scopul este de a estima eroarea traiectoriei T_f^k pe setul de imagini dat. Pentru acest lucru, este construită traiectoria deplasării „înapoi": $T_b^k = (\hat{x}_t, \hat{x}_{t+1}, \dots, \hat{x}_{t+k})$ [14]. Eroarea algoritmului Forward-Backward va fi calculată conform: $FB(T_f^k|S) = distance\left(T_f^k, T_b^k\right)$ [14]. Pentru simplitate, se va folosi distanța Euclidiană dintre punctul inițial și punctul final: $distance(T_f^k, T_b^k) = \|x_t - \hat{x}_t\|$. [14]

Se observă în Figura 3.2.11 faptul că algoritmul debutează cu o urmărire foarte bună a obiectelor, însă pe parcurs se remarcă un comportament diferit față de celelalte tracker-e: aproximativ toate chenarele își vor schimba dimensiunea și chiar vor pierde din urmărire modelele alese. Cel mai bun exemplu pentru redimensionarea cadrelor de urmărire este paralela dintre Figura 3.2.11 și Figura 3.2.12 de mai jos. De asemenea, imaginile ilustrează și faptul că algoritmul MEDIANFLOW eșuează în urmărirea obiectelor.



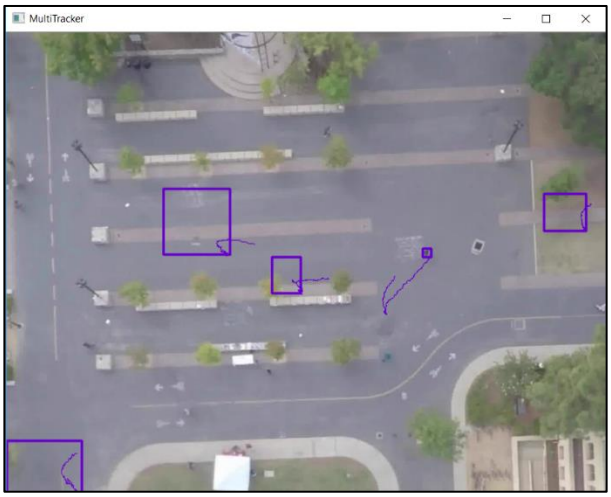


Figura 3.2.11 și Figura 3.2.12 – Algoritmul MEDIANFLOW

Redimensionarea cadrelor apare ca urmare a implementării algoritmului. Calculul erorii Forward-Backward nu este optimă pentru teste reale.

Analizând algoritmul de supraveghere MEDIANFLOW se poate spune că are o viteză destul de bună, rata de returnare a eșecului este foarte slabă. Asemenea tuturor algoritmilor analizați până acum, s-a remarcat faptul că niciunul nu se descurcă foarte bine la acest aspect al testării.

Algoritmul GOTURN este unic în această paralelă deoarece necesită o antrenare offline a rețelei ți folosește o rețea neuronală convoluțională. Aceasta trebuie să aibă o învățare lină; mișcările mici fiind preferate. Pentru această idee, se alege ca model centrul chenarului din imaginea prezentă (c'_x, c'_y) relativ la centrul dreptunghiului din viitoarele cadre (c_x, c_y) folosindu-se următoarele formule:

$$c'_x = c_x + w \cdot \Delta x \text{ [3]} \qquad c'_y = c_y + h \cdot \Delta y \text{ [3]}$$

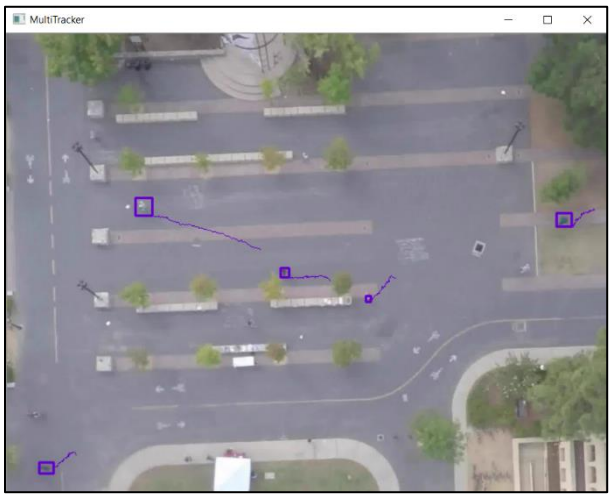
unde w și h sunt lățimea și înălțimea dreptunghiului de urmărire din cadrul anterior. Termenii Δx și Δy iau valori aleatoare care capturează schimbările poziției chenarului de urmărire de la un cadru la altul. În antrenarea acestei rețele, atât Δx cât și Δy pot fi reprezentate cu ajutorul unei distribuții Laplace cu media zero [3]. În mod similar, shimbările create de redimensionarea obiectului urmărit sunt tratate astfel:

$$w' = w \cdot \gamma_w \text{ [3]} \qquad h' = h \cdot \gamma_h \text{ [3]}$$

unde w' și h' sunt lățimea și înălțimea dreptunghiului de urmărire din cadrul prezent și w și h sunt lățimea și înălțimea dreptunghiului de urmărire din cadrul anterior. Variabilele γ_w și γ_h sunt modelate de distribuția Laplace cu media egală cu unu [3]. O astfel de distribuție oferă o probabilitate mai mare de a pastra marimea inițială a chenarului de urmărire de la un cadru la altul.

În practică, s-au efectuat teste asupra tracker-ului GOTURN și s-a observat un comportament de urmărire mult mai liniar față de toți ceilalți algoritmi prezentați până în acest moment. În timp ce algoritmii BOOSTING și KCF au fost predispuși la a urmări modelele fals pozitive, algoritmul GOTURN a afișat un comportament extrem de diferit, ajung foarte greu la cazurile de detectare a fals pozitivului. Totuși, precum algoritmul MEDIANFLOW, acesta are o problemă de redimensionare a cadrelor de urmărire, însă nu debutează la fel de repede.

Urmărindu-se folosirea unei mișcări line, continue, algoritmul GOTURN depășește la nivel de performață algoritmul TLD, cel din urmă având o traiectorie de surpraveghere a modelelor neregulată.



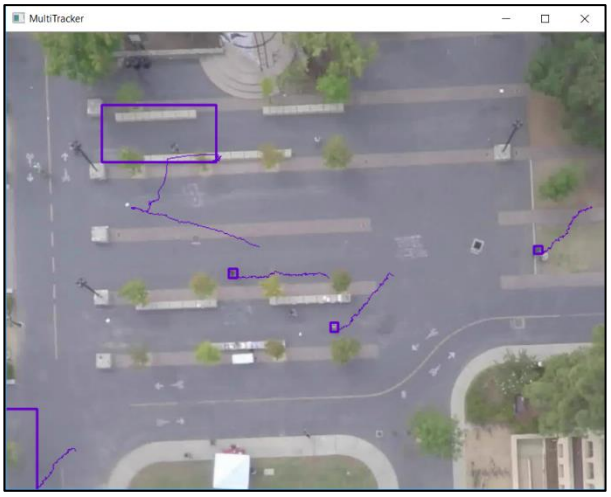


Figura 3.2.13 și Figura 3.2.14 – Algoritmul GOTURN

Se observă în Figura 3.2.13 debutul urmăririi tracker-ului GOTURN. Aceasta este una liniară aproape ideală, însă în Figura 3.2.14 situația se schimbă. În momentul în care modelul din josul imaginii a dispărut din cadru, rețeaua începe o urmărire haotică, iar în clipa când modelul urmărit de sus s-a apropiat de un alt obiect care semăna cu el, supravegherea de asemenea a devenit neuniforma.

Fiecare cadru de urmărire are altă dimensiunea față de cum era inițial, lucru care indică faptul că algoritmul, în ciuda formulelor și grijei sporite oferite în implementare, manifestă probleme de redimensionare. Acestea, combinate cu pierderea modelului de urmărire se pot transforma într-un adevărat eșec de supraveghere, lucru care se poate observa în Figura 3.2.14.

În ansamblu, algoritmul GOTURN se descurcă mult mai bine în condițiile reale de test spre deosebire de ceilalți algoritmi prezentați.

Algoritmul MOSSE este cel mai rapit algoritm, însă eficiența este foarte slabă. Acesta are o rată de urmărire de 669 fps, spre deosebire de algoritmul MIL care are 25 fps[5]. Ca și în cazul algoritmului KCF, se remarcă faptul că o viteză prea mare în urmărire duce în final la pierderea

modelului, rezultatul fiind detectarea falsului pozitiv. Și mai interesant este faptul că supravegherea celor cinci modele alese va fi identică cu cea de la tracker-ul KCF.

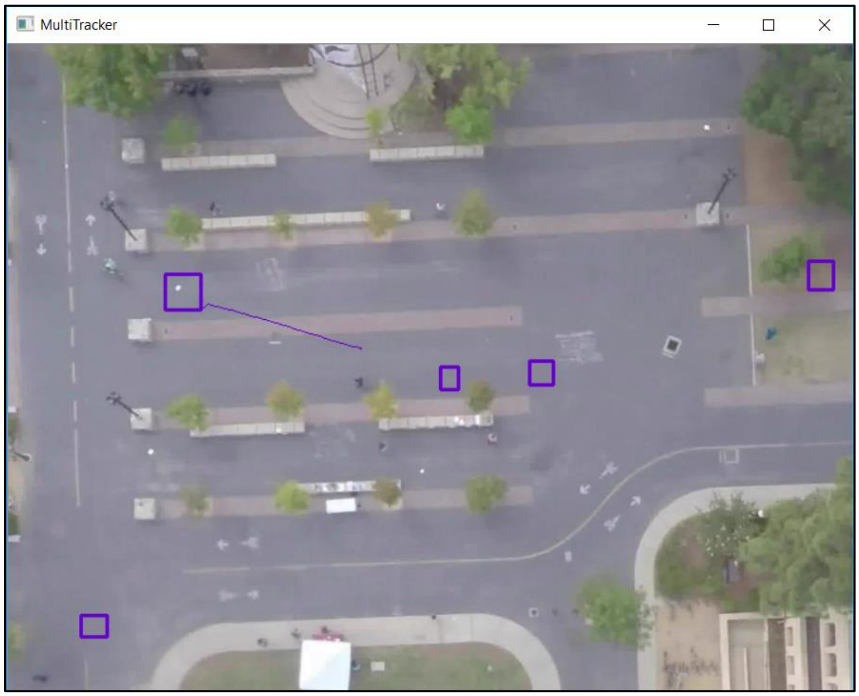


Figura 3.2.15 – Algoritmul MOSSE

S-a testat algoritmul mărindu-se chenarele de supraveghere, schimbându-se modelele de urmărire sau numărul obiectelor alese, dar fără succes. Totuși, se remarcă reușita pe termen scurt a tracker-ului asupra unui singur model. Ca exemplu de urmărire a fost vizat biciclistul din Figura 3.2.15. Acest rezultat o fost obținut și în cazul algoritmului KCF, dar traiectoria biciclistului diferă.

Pentru început, algoritmul MOSSE are nevoie de date de antrenare ce constau într-un set de imagini f_i și returnează datele antrenate g_i. În cazul acesta, datele antrenate vor avea o formă compactă 2D ($\sigma = 2.0$) Gaussiană cu vârful în centrat pe ținta urmărită în imaginile de antrenare f_i. Se definesc variabilele F_i, G_i și filtrul H asupra cărora se va aplica transformarea Fourier [5]:

$$H_i^* = \frac{G_i}{F_i} \quad [5]$$

Pentru a găsi un filtru ce mapează datele de antrenare, algoritmul MOSSE găsește filtrul H care minimizează suma erorii pătratice dintre datele de ieșire actuale și datele de ieșire dorite. Astfel, problema de minimizare va lua următoarea formă:

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \quad [5]$$

Fiecare element H (indexat după ω și ν) poate fi rezolvat independent deoarece fiecare operație dintr-o transformare Fourier este de sine stătătoare. Asta implică rescrierea funcției folosindu-se $H_{\omega\nu}$ și $H^*_{\omega\nu}$.

$$0 = \frac{\partial}{\partial H^*_{\omega\nu}} \sum_i |F_{i\omega\nu} H^*_{\omega\nu} - G_{i\omega\nu}|^2 \quad [5]$$

Prin rezolvarea ecuației pentru H^* se găsește o expresie pentru filtrul MOSSEE:

$$H^* = \frac{\Sigma_i G_i \odot F_i^*}{\Sigma_i F_i \odot F_i^*} \quad [5]$$

Filtrele trebuie sa aibă o învățare rapidă. Prin urmare, filtrul MOSSE devine:

$$H_i^* = \frac{A_i}{B_i} \quad [5]$$

$$A_i = \eta G_i \odot F_i^* + (1-\eta) A_{i-1} \quad [5]$$

$$B_i = \eta F_i \odot F_i^* + (1-\eta) B_{i-1} \quad [5]$$

unde η reprezintă rata de învățare. În practică, $\eta = 0.125$ îi permite filtrului MOSSE să se adapteze schimbărilor.

Algoritmul CSRT, pe baza testelor efectuate, se poate remarca faptul că se descurcă extraordinar de bine. Linia de urmărire este lină, nu detecteaza fals pozitiv-ul, are o viteză destul de bună și reușește să depășească orice alt tracker analizat până acum. Algoritmul GOTURN are o urmărire la fel de exactă asemenea algoritmului CSRT, însă pierde modelele de urmărire dacă acestea se apropie de zone din fundal care seamană cu obiectele inițiale.

Dat un set N_d de date, canale $f = \{f_d\}_{d=1:Nd}$ și filtrele corespunzătoare $h = \{h_d\}_{d=1:Nd}$, unde $f_d \in R^{d_w \times d_h}$ și $h_d \in R^{d_w \times d_h}$, iar poziția obiectului x este estimată prin maximizarea probabilității: $P(x|h) = \Sigma_{d=1}^{N_d} p(x|f_d) p(f_d)$ [1]. Densitatea $p(x|f_d) = [f_d * h_d](x)$ [1] este o convoluție a unei viitoare hărți cu o învățare evaluată în x și $p(f_d)$. Filtrele optime sunt obținute pe baza următoarei formule [1]:

$$\arg\min_{h} \sum_{d=1}^{N_d} \|f_d * h_d - g\|^2 + \lambda \sum_{d=1}^{N_d} \|h_d\|^2 = \arg\min_{h} \sum_{d=1}^{Nd} \left(\left\| \hat{h}_d^H \operatorname{diag}(\hat{f}_d) - \hat{g}_d \right\|^2 + \lambda \left\| \hat{h}_d \right\|^2 \right)$$

În egalitatea anterioara, bazată de teorema lui Parsevall, operatorul $\hat{a} = \text{vec}(F[a])$[1] este o transformare Fourier. Minimizarea formulei anterioare duce la ecuația gradientului unde fiecare canal este egal cu zero.

Harta fiabilității spațiale $m \in [0,1]^{d_w \times d_h}$ cu elementele $m \in \{0,1\}$ indică cât de fezabilă este învățarea fiecărui pixel din imagine. Probabilitatea ca pixelul x să fie fiabil este condiționată de y astfel: $p(m = 1|y, x) \propto p(y|m = 1, x)p(x|m = 1)p(m = 1)$[1]. Această hartă a fiabilității identifică pixelii ce trebuie ignorați în procesul de învățare utilizându-se constrângerea: $h \equiv m \odot h$ [1].

Tracker-ul CSRT va localiza obiectul vizat adunând răspunsurile primite de la filtere de învățare corelate h_{t-1}. Se vor distinge modelele de fundal prin două histograme împărțite conform chenarului de urmărire. Se va construi apoi harta fiabilității spațiale și va fi calculat un filtru optim $\tilde{h}$ și va fi estimat canalul de învățare fiabil $\widetilde{w} = [\widetilde{w}_1, \dots, \widetilde{w}_{Nd}]^T$ [1].

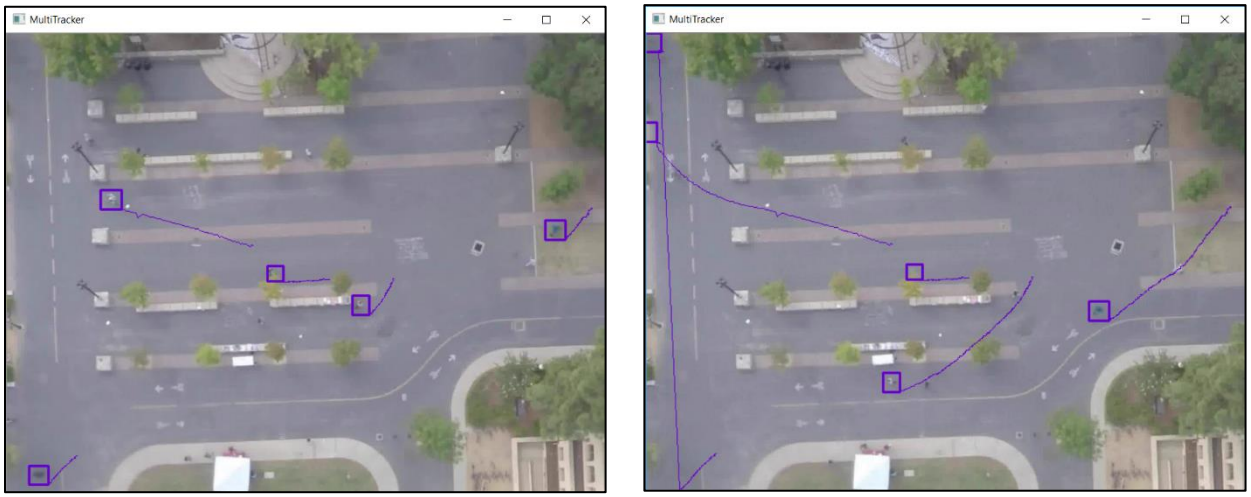


Figura 3.2.16 și Figura 3.2.17 – Algoritmul CSRT

În Figura 3.2.16 se poate observa parcursul urmăririi într-un mediu de test aproape de realitate. Datorită separării foarte exacte a modelelor de fundal, tracker-ul CSRT are o urmărire

foarte exactă și punctuală. Acesta reușește ceea ce niciun alt algoritm, din aceasta paralelă, nu a putut finaliza în cel mai dorit mod.

În momentul în care are loc o ocluzie mai greu de interceptat precum ieșirea din cadru a modelului urmărit, tracker-ul încearcă să continue supravegherea, dar desigur eșuează. Se remarcă în Figura 3.2.17 faptul că obiectul din josul imaginii părăsește cadrul și tracker-ul încearcă sa-l urmărească într-un loc total greșit, în partea de sus a figurii.

În comparație cu ceilalți algoritmi, urmărirea se desfășoară în mod liniar cu o viteză destul de mare.

3.3 Evaluarea algoritmilor în mod calitativ (occlusions)

Atunci când este dezvoltat un sistem ce urmărește modelele selectate, inevitabil apar ocluziile. Dacă este folosit un cadru fix, atunci majoritatea ocluziilor vor apărea din cauza faptului că obiectele ies din raza de acțiune a camerei de supraveghere. Dacă aceasta nu poate filma la o calitate foarte mare sau pe o rază care acoperă cât mai multe unghiuri moarte, implicit nici urmărirea obiectelor nu va avea randamentul dorit.

Pierderea obiectului urmărit din cauze externe, diferite de algoritmul implementat, pot fi cauza a mai multor scenarii: ieșirea modelului din cadru, mascarea obiectului de un alt model urmărit, ascunderea modelului în spatele unui alt obiect ce aparține fundalului etc. O ocluzie este reprezentată și de dispariția obiectului din cadru, schimbarea culorii sau a formei acestuia, sau chiar și redimensionarea modelului vizat.

Momentul în care un obiect își schimbă în mod brusc viteza sau direcția în care se deplasează, reprezintă o ocluzie, iar algoritmul de supraveghere poate să eșueze. Un alt exemplu de ocluzie este dat de o aglomerație sau o mulțime de obiecte cu caracteristici asemănătoare.

Tracker-ele alese pentru realizarea acestei paralele vor fi testate pentru fiecare tip de ocluzie exemplificat în paragraful de mai sus. Pentru acest lucru s-au folosit alte video-uri din setul de date oferit de Universitatea Stanford numit Stanford Drone Dataset[12].

Inițial s-au testat algoritmii pe video-ul numărul trei din setul de date „coupa” pentru a observa comportamentul acestora în momentul în care modelul urmărit părăsește cadrul. Pentru ca testele să fie corecte, a fost selectat același model pentru fiecare algoritm în parte.

Figura 3.3.1 – Modelul ales pentru testarea ieșirii din cadru

Testând algoritmul BOOSTING, s-a remarcat faptul că tracker-ul a pierdut modelul atunci când acesta a părăsit cadrul. Se observă în Figura 3.3.2 urmărirea liniară până în momentul părăsirii cadrului de către model.

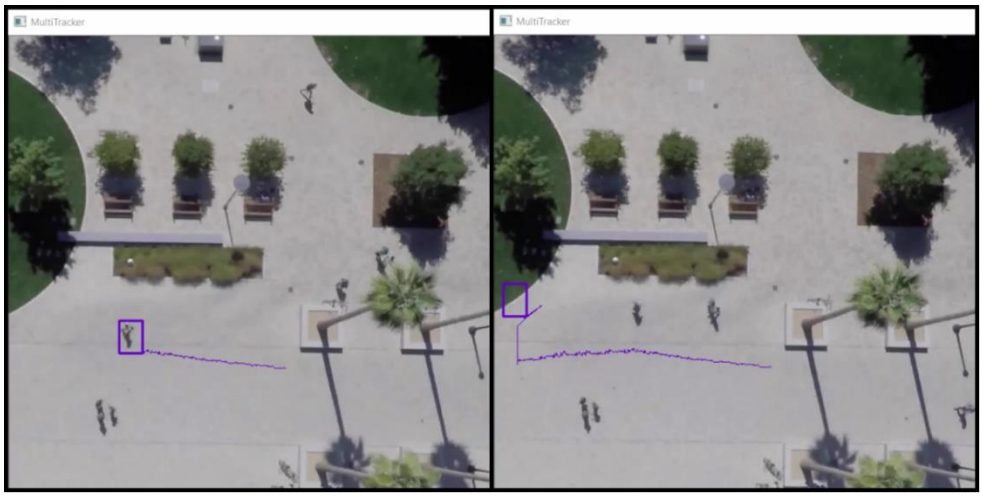


Figura 3.3.2 – Comportamentul algoritmului BOOSTING atunci când modelul părăsește cadrul camerei de supraveghere

Deși traiectoria descrisă de algoritmul MIL diferă de cea a algoritmului BOOSTING, rezultatul ocluziei este același. În Figura 3.3.3 se remarcă faptul că tracker-ul MIL a ajuns aproximativ în aceeași zonă din fundal ca și tracker-ul BOOSTING.

Figura 3.3.3 – Comportamentul algoritmului MIL atunci când modelul părăsește cadrul camerei de supraveghere

În comparație cu BOOSTING și MIL, algoritmul MOSSE are, de asemenea, o urmărire liniară, iar cadrul de urmărire se oprește în locul în care tracker-ul a pierdut modelul. Acest lucru este favorabil momentului în care se poate întoarce în cadru modelul urmărit.

Figura 3.3.4 – Comportamentul algoritmului MOSSE atunci când modelul părăsește cadrul camerei de supraveghere

Algoritmul CSRT are o traiectorie aproape identică cu algoritmul MOSSE în acest caz. Modelul este urmărit cu succes până în momentul în care acesta părăsește cadrul camerei de supraveghere. Acest lucru se poate remarca în Figura 3.3.5.

Figura 3.3.5 – Comportamentul algoritmului CSRT atunci când modelul părăsește cadrul camerei de supraveghere

Un comportament asemănător, dar diferit îl afișează tracker-ul GOTURN. Acesta, urmărește obiectul aproape liniar, însă atunci când apare ocluzia, se pierde și mărește chenarul de căutare pe întreaga suprafață a video-ului.

Figura 3.3.6 – Comportamentul algoritmului GOTURN atunci când modelul părăsește cadrul camerei de supraveghere

Se remarcă în Figura 3.3.6 momentul pierderii modelului urmărit. Tracker-ul încearcă să continue urmărirea însă fără succes. Totuși ajunge în final să părăsească cadrul video-ului.

Tracker-ul KCF nu poate fi testat pentru acest tip de ocluzie deoarece pierde modelul ajungând la detectarea fals pozitivului. S-a încercat testarea cu diferite dimensiuni ale dreptunghiului de urmărirea însă fără succes. Se vede în Figura 3.3.7 acest fapt.

Figura 3.3.7 – Comportamentul algoritmului KCF

Asemenea algoritmului KCF, nici algoritmul MEDIANFLOW nu a putut finaliza testarea. În ambele cazuri, ambele tracker-e au pierdut modelul și au ajuns în final să urmărească fundalul, deci să fie predispuse la fals pozitiv.

În cazul tracker-ului MEDIANFLOW, dreptunghiul de urmărire se va redimensiona așa cum se observă în Figura 3.3.8. În urma acestui fapt, algoritmul va pierde obiectul urmărit și testarea ocluziei nu va mai putea fi posibilă.

Figura 3.3.8 – Comportamentul algoritmului MEDIANFLOW atunci când modelul părăsește cadrul camerei de supraveghere

Algoritmul TLD are o urmărire neuniformă, iar în momentul în care modelul părăsește cadrul camerei de supraveghere, aceasta devine și mai haotică. Tracker-ul pierde modelul încă din primele clipe de urmărire. Încearcă să se redreseze asupra modelului urmărit inițial dar eșuează și continuă să urmărească alte obiecte asemănătoare din fundal. Tracker-ul sfârșește prin a supraveghea fundalul. În Figura 3.3.9 se observă urmărirea algoritlui TLD, supraveghere care este agresivă, neuniformă.

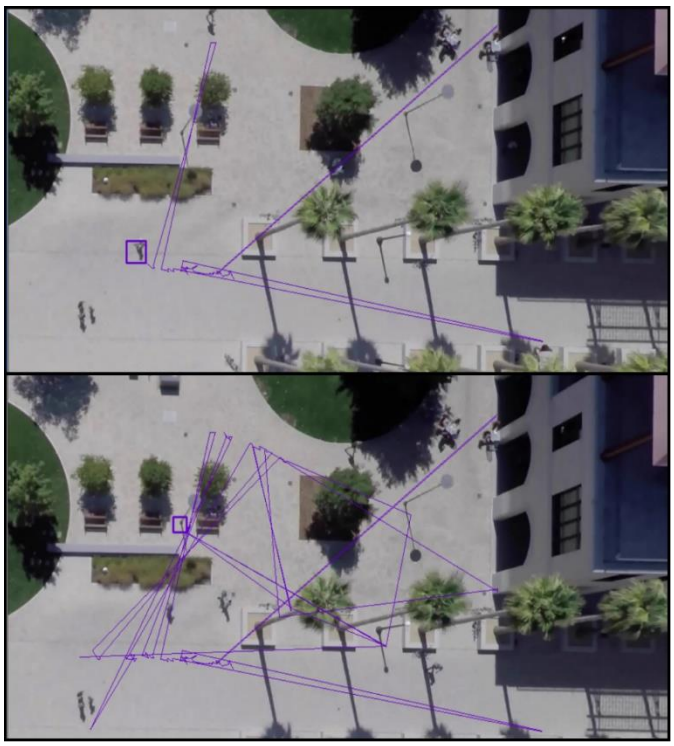


Figura 3.3.9 – Comportamentul algoritmului TLD atunci când modelul părăsește cadrul camerei de supraveghere

Algoritmii de supraveghere au fost testați în vederea ocluziei de suprapunere a modelului urmărit cu fundalul. Pentru acest lucru s-a folosit video-ul numărul zece din setul de date „nexus”. Pentru ca testele să fie corecte, a fost ales același model pentru fiecare algoritm în parte.

Figura 3.3.10 – Modelul ales pentru testarea suprapunerii cu alte obiecte din fundal

Inițial, modelul va trece parțial pe sub un copac, apoi pe sub al doilea copac unde va dispărea complet și va reapărea în fundal când iese de sub acesta. În urma acestui test, se va determina algoritmul cel mai optim din punctul de vedere al ocluziei cu un alt obiect din fundal.

Algoritmul BOOSTING, așa cum se poate observa și în Figura 3.3.11, trece cu brio ambele teste reușind să localizeze obiecte chiar și atunci când acesta dispare complet și reapare în altă parte. De asemenea, reușește să localizeze foarte exact și momentul în care obiectul este ascuns parțial de un alt obiect din fundal.

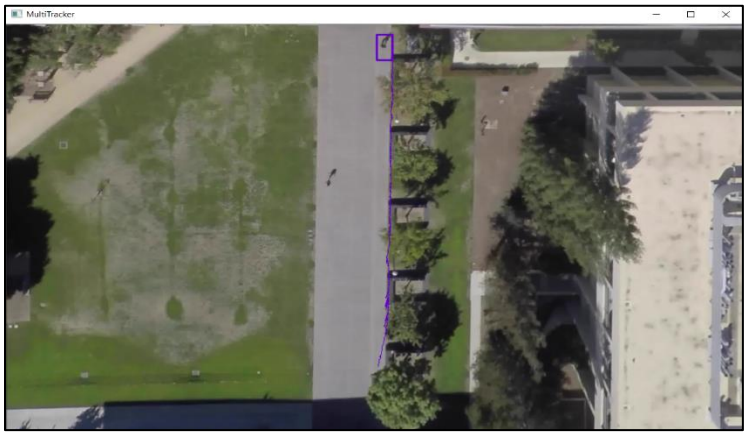


Figura 3.3.11 – Comportamentul algoritmului BOOSTING atunci când modelul este ascuns de alte obiecte din fundal

Figura 3.3.12 – Comportamentul algoritmului MIL atunci când modelul este ascuns de alte obiecte din fundal

În Figura 3.3.12 este surprins comportamentul tracker-ului MIL în cadrul descris anterior. Acesta depășește ocluzia parțială însă nu reușește să supravegheze obiectul la ieșirea de sub al doilea copac. Are loc o ocluzie totală între model și fundal, iar algoritmul nu este capabil să detecteze noua locație a obiectului în momentul reapariției acestuia în cadru.

Dacă până în acest momentul BOOSTING a reușit să treacă testul complet, iar MIL parțial, algoritmul KCF nu depășește nici măcar primul obstacol. Figura 3.3.13 reprezintă eșecul acestui tracker încă de la prima ocluzie.

Figura 3.3.13 – Comportamentul algoritmului KCF atunci când modelul este ascuns de alte obiecte din fundal

Figura 3.3.14 – Comportamentul algoritmului TLD atunci când modelul este ascuns de alte obiecte din fundal

Agoritmul TLD surprinde și de această dată într-un mod neplăcut. Acesta nu doar că nu reușește să urmărească obiectul în timpul ocluziilor, dar mai mult decat atât, găsește alte modele în fundal pe care le va supraveghea în locul obiectului ales inițial. Se poate observa în Figura 3.3.14 acest comportament.

Asemenea algotirmului MIL, tracker-ul MEDIANFLOW pierde obiectul încă de la prima ocluzie. Totuși, spre deosebire de MIL, încearcă o redimensionare a chenarului de urmărire, dar fără succes. Obiectul va rămâne pierdut, lucru ce poate fi văzut în Figura 3.3.15.

Figura 3.3.15 – Comportamentul algoritmului MEDIANFLOW atunci când modelul este ascuns de alte obiecte din fundal

Figura 3.3.16 – Comportamentul algoritmului GOTURN atunci când modelul este ascuns de alte obiecte din fundal

În Figura 3.3.16 este descris comportamentul algoritmului GOTURN. Acesta urmărește în primă instanță obiectul ales. La apropierea de o zonă din fundal care pare că seamănă cu obiectul inițial, tracker-ul va începe supravegherea fundalului, iar testarea ocluziilor va eșua drastic. La capitolul ocluzii, acest algoritm este la fel de ineficient ca și algoritmul TLD.

Figura 3.3.17 – Comportamentul algoritmului CSRT atunci când modelul este ascuns de alte obiecte din fundal

Figura 3.3.18 – Comportamentul algoritmului MOSSE atunci când modelul este ascuns de alte obiecte din fundal

Algoritmii CSRT și MOSSE au un comportament asemănător cu cel al algoritmului MIL. Atât Figura 3.3.17 cât și Figurile 3.3.18 și 3.3.12 arată dreptunghiul de urmărire blocat pe al doilea pom din fundal. Toate cele trei tracker-e depășesc obstacolul primei ocluzii și eșuează la al doilea test.

În concluzie, pentru ocluzii parțiale, CSRT, MOSSE și MIL sunt cei mai buni din toți cei analizați, însă pentru toate tipurile de ocluzii, algoritmul BOOSTING rămâne cea mai bună alegere.

Capitolul 4. Concluzie

4.1 Concluzie

În urma tuturor analizelor efectuate asupra algoritmilor de urmărire s-au remarcat câteva rezultate surprinzătoare. Fiecare algoritm are puncte forte și puncte slabe lucru care face ca unele teste efectuate să fie favorabile unor tracker-e și altora nu.

Dacă se dorește o viteză foarte mare de urmărire, este recomandată folosirea algoritmilor MOSSE și KCF, însă nu sunt o alegere tocmai potrivită pentru urmărirea mai multor obiecte simultan. Algoritmul BOOSTING supraveghează cu exactitate traiectoria obiectelor și modelele în sine foarte bine. Despre acesta, în urma tuturor testelor efectuate, se poate spune că este cel mai bun în urmărirea obiectelor.

Dacă obiectul este redimensionat (se apropie sau se îndepărtează de cameră) este indicată folosirea algoritmului MEDIANFLOW datorită comportamentului acestuia. Pentru cazurile în care obiectul este parțial ascuns de un alt obiect, se pot folosi tracker-ele MIL, CSRT și MOSSE fiind foarte exacte în astfel de cazuri.

Bibliografie

[1] Alan Lukezic, Tomas Vojır, Luka Cehovin Zajc, Jiri Matas and Matej Kristan, articolul „*Discriminative Correlation Filter with Cahnnel and Spatial Reliability*”, publicat în „*International Journal of Computer Vision*”, 2016

[2] Boris Babenko, Ming-Hsuan Yang , Serge Belongie, articolul „*A Family of Online Boosting Algorithms*”, publicat în „*IEEE 12th International Conference on Computer Vision Workshops*”, 2009

[3] Burak Uzkent, Young Woo Seo, articolul „*EnKCF: Ensemble of Kernelized Correlation Filters for High-Speed Object Tracking*”, publicat în „*IEEE Winter Conference on Applications of Computer Vision(WACV)*”, 2018

[4] David Held, Sebastian Thrun, Silvio Savarese, articolul „*Learning to Track at 100 FPS with Deep Regression Networks*”, publicat în „*European Conference on Computer Vision (ECCV)*”, 2016

[5] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, Yui Man Lui, articolul *„Visual Object Tracking using Adaptive Correlation Filters”*, publicat în „*IEEE Computer Society Conference on Computer Vision and Pattern Recognition*” 2010

[6] Farhad Soleimanian Gharehchopogh, Isa Maleki, Seyyed Reza Khaze, „*A new approach in dynamic traveling salesman problem: a hybrid of ant colony optimization and descending gradient*”, publicat în „*International Journal of Managing Public Sector Information and Communication Technologies*”, volumul 3, 2012

[7] Georg Nebehay, teză de doctorat cu titlul „*Robust Object Tracking Based on Tracking-Learning-Detection*”, 2012

[8] Gary Bradski, Adrian Kaehler, *„Learning OpenCV”*, ed. O’Reilly, sept 2008, p. XVI

[9] Joao F. Henriques, Rui Caseiro, Pedro Martins and Jorge Batista, articolul „*High-Speed Tracking with Kernelized Correlation Filters*”, publicat în „*IEEE Transactions on Pattern Analysis and Machine Intelligence*”, 2014

[10] Mohinder S. Grewal și Angus P. Andrews, „Applications of Kalman Filtering in Aerospace 1960 to the Present”, publicat în „IEEE Xplore”, mai 2010

[11] Robert E. Schapire, „*Expaining AdaBoost*”, publicată de Springer, Berlin, Heidelberg, 2013

[12] A. Robicquet, A. Sadeghian, A. Alahi, S. Savarese, „*Learning Social Etiquette: Human Trajectory Prediction In Crowded Scenes in European Conference on Computer Vision (ECCV)*”, 2016

[13] Xiu-Shen Wei, Jianxin Wu, Zhi-Hua Zhou, articolul „*Scalable Algorithms for Multi-Instance Learning*”, publicat în „*IEEE Transactions on Neural Networks and Learning Systems*”, volumul 28, 2007

[14] Zdenek Kalal, Krystian Mikolajczyk, Jiri Matas, articolul „*Forward-Backward Error: Automatic Detection of Tracking Failures*”, publicat în „*International Conference on Pattern Recognition*”, august 2010

[15] Zdenek Kalai, Krystian Mikolajczyk and Kiri Matas, articolul „*Tracking-Learning-Detection*”, publicat în „*IEEE Transactions on Pattern Analysis and Machine Intelligence*”, volumul 6, ianuarie 2010

[16] http://lxmls.it.pt/2011/guide_day2.pdf

[17] https://www.researchgate.net/figure/Pseudocode-for-generic-boosting-algorithms_fig1_6265168

Anexe