Universitatea “Politehnica” din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Sistem de urmărire în timp real a obiectelor în secvențe video captate de drone

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de
Inginer în domeniul *Calculatoare şi Tehnologia Informaţiei*
programul de studii de licenţă *Ingineria Informaţiei*

Conducători ştiinţifici:
Ş. L. Dr. Ing. Anamaria RĂDOI
Prof. Dr. Ing. Corneliu BURILEANU

Absolvent
Marian-Valentin BĂNICĂ

București
2019

Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Departamentul **EAII**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **BĂNICĂ D.M. Marian-Valentin , 441A**

1. Titlul temei: Sistem de urmărire în timp real a obiectelor în secvențe video captate de drone

2. Descrierea contribuției originale a studentului (în afara părții de documentare) și specificații de proiectare:
Lucrarea de diplomă are drept scop realizarea unui algoritm de urmărire a obiectelor în timp real, prin obiecte înțelegându-se orice tip de ținte aflate în mișcare (de exemplu, mașini, oameni, bicicliști) și de orice dimensiuni. Înregistrările video vor fi realizate din drone. Acest algoritm este deosebit de util în cazul monitorizării în timp real a traficului, dar și în cazul altor aplicații ce presupun supravegherea unei zone de interes. Aplicația realizată în cadrul proiectului va permite urmărirea unui număr variabil de obiecte, pe care utilizatorul le va selecta în prealabil. Urmărirea propriu-zisă va porni de la o comparație a diverșilor algoritmi prezentați în literatura de specialitate, precum cei bazați pe tehnica Adaboost de selecție a trăsăturilor, tehnica învățării instanțelor multiple, sau metode bazate pe filtre adaptive de corelație. Se vor lua în considerare indicatori de performanță precum rata de succes și timpul necesar prelucrării fiecărui cadru. De asemenea, analiza va consta și în determinarea acelor algoritmi capabili să accepte un anumit grad de ocluzie a obiectelor. Pornind de la analiza comparativă a algoritmilor, studentul va dezvolta o aplicație care va permite urmărirea mai multor obiecte în timp real și va îmbunătăți acuratețea urmăririi prin combinarea celor mai performanți algoritmi. Acesta aplicație va fi implementată în Python.

3. Resurse folosite la dezvoltarea proiectului:
Limbaje de programare: Python Biblioteci: Python: numpy, opencv, scipy, imutils Dispozitive: laptop / PC / dronă

4. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline:
PI, POO, RFIA, DEPI

5. Proprietatea intelectuală asupra proiectului aparține: U.P.B.

6. Data înregistrării temei: 2018-11-28 22:14:33

Conducător(i) lucrare,
Ș. L. Dr. Ing. Anamaria RĂDOI
semnătura:

Prof. Dr. Ing. Corneliu BURILEANU
semnătura:

Student,
semnătura:

Director departament,
Prof. dr. ing Sever PAȘCA
semnătura:

Decan,
Prof. dr. ing. Cristian NEGRESCU
semnătura:

Cod Validare: **d48e69bb8d**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *"Sistem de urmărire în timp real a obiectelor în secvențe video captate de drone"*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Calculatoare și Tehnologia Informației*, programul de studii de licență *Ingineria Informației* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țara sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 11/06/2019

Absolvent Marian-Valentin BĂNICĂ

(semnătura în original)

Listă de figuri

Listă tabele

Listă de abrevieri

Adaboost – Adaptive Boosting
Wi-Fi - Wireless Fidelity
TFR - Transformata Fourier Rapidă
TFI - Transformata Fourier Inversă
GPS - Global Positioning System
IMU – Inertial Measuring Unit
LiPo - Litiu-Polymer
ARM - Advanced RISC Machine
GCS - Ground Control Station(Sistemul de control de la sol)
MOSSE – minimul sumei erorilor pătratice de la ieșire
PSR - Peak to Sidelobe Ratio(Raportul intre valoarea maxima si valoarea din jurul acestuia)
ASEF - Average of Synthetic Exact Filters
NOR - Noisy-OR
HOG – Histogram of Oriented Gradients
KCF - Kernelized Correlation Filtering
DCF - Dual Correlation Filter
MP - Mission Planner
MavLink - Micro Aerial Vehicle Link
MEMS - micro-electro-mecanice
UAV- Unmanned Aerial Vehicle
OBR- Object Detection Rate
OpenCV - Open Source Computer Vision Library
SITL - Software în the Loop
ARM - Architecture (Ashton Raggatt McDougall)
RISC - Reduced Instruction Set Computer

Cuprins

Introducere

Urmărirea de obiecte este una din temele de cercetare des întâlnite în domeniul Computer Vision. Aceasta poate fi inclusă într-o serie largă de aplicații din domeniul amintit, cum ar fi: supraveghere video, interacțiune om-mașină, și chiar aplicații în imagistica medicală. Urmărirea de obiecte este procesul prin care un utilizator poate selecta un obiect (de exemplu, desenând un chenar sau furnizând coordonatele acelui obiect în interiorul primului cadru în care acesta apare), după obiectul este identificat în restul cadrelor din fluxul video în mod automat. Așadar, în mod ideal, un algoritm de urmărire va primi o singură dată informații despre obiectul ce va fi urmărit și va fi suficient de rapid pentru a putea identifica obiectul în restul cadrelor inclusiv atunci când locația acestuia se modifică în mod substanțial de la un cadru la altul. În plus, dacă obiectul dispare din aria de interes (prin ocluzie cauzată de alte obiecte sau prin ieșirea în afara cadrului captat de obiectivul camerei de filmat), algoritmul trebuie să fie capabil să reidentifice obiectul urmărit. De asemenea, se dorește ca algoritmul de urmărire să funcționeze în timp real.

Chiar dacă problema urmării de obiect este pusă de chiar câteva decenii, iar progresul nu a întârziat să apară în anii recenți, acest domeniu încă este supus multor provocări. O întreagă serie de factori pot afecta performanțele unui algoritm de urmărire, iar dintre aceștia putem aminti: variația luminii, ocluziunile ce pot apărea în timpul urmăririi ș.a., până la momentul actual nu există un algoritm construit astfel încât să poată manevra cu succes toate problemele prezentate.

Pentru o evaluare completă a performanței, este esențial să se colecteze un set de date reprezentativ. În cazul nostru, obiectele vizate în mod normal sunt oameni sau mașini, iar fundalul în care acestea sunt prezente este de obicei static. Pentru a se stabili performanțele algoritmilor de urmărire în general se folosesc seturi de date ce au deja corect făcute încadrările pe obiectul urmărit. Pentru testul ce trebuie efectuat doar se compară pentru fiecare cadru încadrarea curentă a obiectului cu cea corectă și se face un raport de unde se estimează performanța.

Pentru început am studiat modelul pe care îl au la baza o serie de algoritmi de urmărire, Ulterior am realizat o comparație a performanțelor acestora în diferite condiții de interes, iar în final am încercat integrarea unui sistem automat de urmărire care ne poate ajută in misiunile de cercetare și salvare ce sunt solicitante deoarece implică survolarea zonelor largi, cu teren complex, într-un timp limitat. Misiunile obișnuite includ salvarea oamenilor răniți sau găsirea vehiculelor avariate în relief variat. Accidentele de pe autostrăzi sau navale impun de asemenea o zonă mare de cercetare, iar timpul este critic pentru probabilitatea de a găsi victima. În cazul furtunilor, cutremurelor, alunecărilor de teren și inundațiilor evaluarea imediată a pagubelor și găsirea supraviețuitorilor sunt condiții critice pentru a construi un plan de salvare. Imaginile de la aeronave fără pilot pot juca un rol important în misiunile de salvare la scară mare.

Odată cu dezvoltarea senzorilor bazați pe sisteme micro-electro-mecanice, folosirea dronelor mici a devenit o alternativă bună pentru cercetare, salvare și supravegherea mediului. UAV-urile pot fi echipate cu diverse sisteme de senzori, de exemplu pentru observarea dezastrelor, inclusiv

aprecierea daunelor pentru inundații și cutremure de pământ. Astăzi, datorită prețului scăzut al dronelor, oamenii pot dezvolta UAV-uri de dimensiuni reduse care au următoarele avantaje:

- pot survola o zonă pe durate îndelungate și la înălțimea dorită;
- produc imagini de rezoluție mai mare decât sateliții;
- pot zbura sub nivelul de zbor obișnuit al traficului aerian;
- se pot apropia de zonele de interes.

Folosirea tehnologiei aeronavelor fără pilot și a senzorilor pentru căutare, salvare și supraveghere nu este o idee nouă. Trebuie să luăm în calcul numărul de operatori necesar pentru un sistem UAV. Avem nevoie de un pilot pentru a controla, planifica și monitoriza drona și un copilot pentru a opera cu senzorii și fluxul de informații. Pentru că un om se poate concentra pe un număr limitat de sarcini, se încearcă optimizarea prezentării informației și automatizarea achiziției de date.

Pentru automatizarea achiziției de informații, se încearcă integrarea sistemelor de detecție video automate pentru oameni, mașini și nave. Datele produse în zonele afectate de dezastru sunt georeferențiate pentru a susține prezentarea informației și acțiunea umanitară. Folosind coordonatele locului unde s-a făcut o poză ce conține o țintă, înălțimea de zbor și poziția țintei în poză, se pot calcula coordonatele țintei.

În aceasta lucrare este propus un sistem în care este realizată urmărirea obiectului prin încadrarea acestuia în centrul imaginii pe secvența video captată de dronă. Aceasta se deplasează pentru a menține obiectul în cadru. Urmărirea începe când un operator uman încadrează obiectul pe care dorește să-l urmărească într-un chenar.

Urmărirea obiectului va fi realizată cu ajutorul unei drone ce are atașat un dispozitiv computațional încorporat, eficient din punct de vedere al consumului și cu o putere de calcul ridicată. Programul rulează la nivelul dronei, fără să fie nevoie de procesarea datelor pe un alt dispozitiv. Camera este atașată de dronă folosind un gimbal ce menține un unghi fix de filmare.

Capitolul 1 Instrumente utilizate

1.1 Aeronava fără pilot

Pentru realizarea experimentelor au fost făcute filmări din dronă la diferite altitudini și unghiuri de înclinare a camerei. Am ales o dronă de tip quad-copter (Figura 1.1) datorită agilității în zbor și a simplității mecanice, capabilă să se mențină în zbor deasupra țintei și stabilă în timp ce face poze. Quad-copter-ul e fabricat în întregime din fibră de carbon cu diametrul brațelor de 16mm și distanța între tuburi este de 650mm. Tuburile sunt goale pe interior pentru a reduce greutatea. În Figura 1.1 se pot observa principalele caracteristici ale aeronavei folosite și structura acesteia.

Figură 1.1 - Quad-copter(dronă)

Echiparea este împărțită pe trei nivele, și găzduiește întregul sistem. Pe nivelul superior avem unitatea de procesare a imaginii, NVIDIA Jetson TX2. Pe nivelul al doilea se găsește autopilotul Pixhawk, unitatea GPS cu magnetometru, antenele Wi-Fi și radio modemul pentru telemetrie. Nivelul 3, fabricat la CNC din duraluminiu găzduiește bateriile. Pe nivelul de jos avem gimbal-ul (sistemul

de orientarea al camerei) și camera Sony QX10. Gimbal-ul poate să orienteze camera la $\pm 45^{\circ}$ de la orizontală în ruliu și tangaj. Gimbal-ul este comandat de două motoare electrice fără perii controlate de un sistem de stabilizare și comanda prevăzut cu un sistem de măsură al orientării propriu (IMU – inertial measuring unit). Gimbal-ul este fabricat integral din sticlotextolit. Greutatea părților componente se regăsește în Tabelul 1.1.

Un lucru important de menționat din Tabelul 1.1 este dat de masă totală a aeronavei, deși ocupă un volum destul de ridicat, are o greutate de doar 2.39 *kg*.

Piesa	Masa (kg)	Piesa	Masa (kg)
Cadru	0.61	Camera	0.29
Motoare	0.39/buc.	Autopilot	0.02
Baterie	0.80	Jetson TX2	0.10
Gimbal	0.17		
		Total	2.39

Tabel 1.1 - Greutate componente

1.2 Sistemul de propulsie

Sistemul de propulsie are o serie de cerințe pe care trebuie să le îndeplinească:

- să producă suficientă putere pentru a finaliza misiunea,
- să maximizeze viteza de croazieră la survolarea zonei, toate în timp ce păstrăm o greutate redusă a sistemului.

Alegerea motoarelor este importantă pentru performanța quad-copterului. Criteriul principal este eficiența motorului dar și greutatea a fost luată în calcul. Cu ajutorul programului EPROP-calc pentru ansamblul compus din motor, elice, ESC (electronic speed controller) – controlul electronic al vitezei și acumulator a fost obținută o autonomie de 40 de minute. Abilitatea de a avea un consum ridicat de curent pentru un interval îndelungat de timp, în timp ce menține tensiunea nominală e o condiție importată în proiectarea dronei. Compoziția chimică a bateriei trebuie să ofere o densitate mare, cicluri de încărcare-descărcare repetabile și o cădere de tensiune mică la intensitate mare a curentului electric consumat. Un acumulator LiPo (Litiu-Polymer) de 11000 *mAh* a fost ales pentru a maximiza puterea electrică disponibilă, micșorând în același timp greutatea. Se realizează și un compromis al capacității bateriei ținând cont de surplusul de greutate adăugat dronei având ca și consecință imediată o scădere a autonomiei în loc de o creștere a acesteia.

Un motor electric 48-22-490 *KV*, combinat cu o elice de 16x5.5 *mm* din fibra de carbon a îndeplinit condiția de autonomie conform programului eCalc.

1.3 Arhitectura sistemului

Sistemul constă din autopilotul Pixhawk, computerul îmbarcat, Jetson TX2 și camera Sony QX10. Pixhawk este un autopilot foarte performant, potrivit pentru proiectul nostru; suportă atât zbor cu pilot uman, cât și complet autonom, incluzând navigație după coordonate GPS, controlul camerei, decolare și aterizare automată. Computerul pentru procesarea video instalat pe dronă de test este modelul Jetson TX2, de la NVIDIA, prezent în Figura 1.3.

Figură 1.2 - Arhitectura sistemului

Pentru a testa în siguranță funcționalitatea codului scris, a fost nevoie să folosim un întreg chit de dezvoltare aparținând NVIDIA, ce este capabil să simuleze arhitectura dronei. Sistemul de operare folosit de acest procesor de tip ARM este Linux. Dimensiunile scăzute și performanțele ridicate au dus la alegerea acestuia.

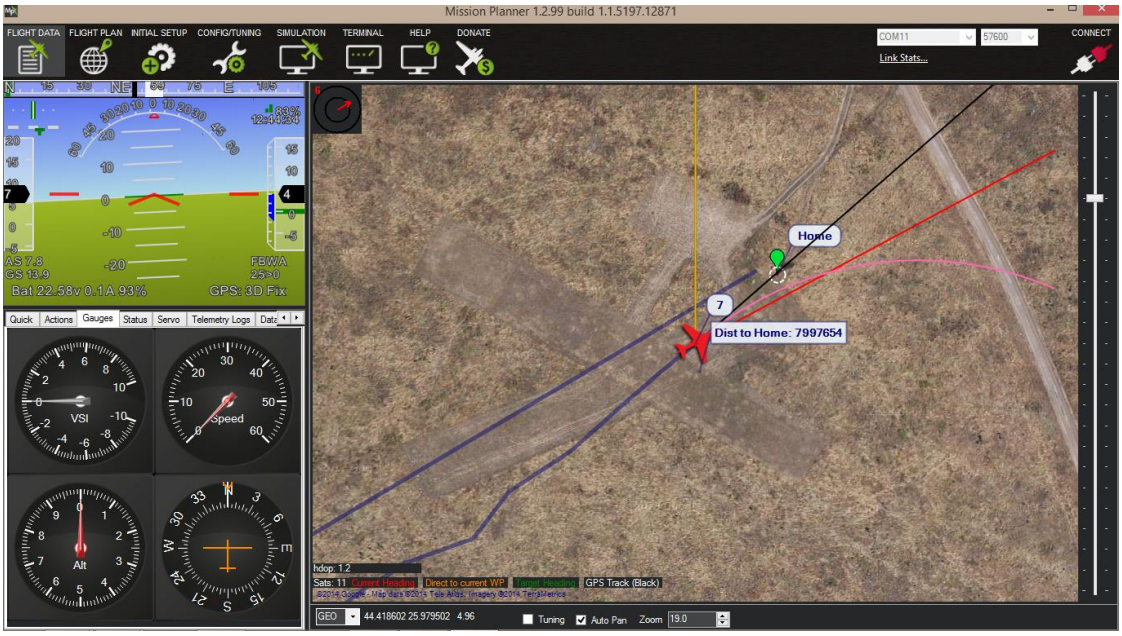
Specificațiile acestuia sunt următoarele:

- GPU: NVIDIA Pascal, 256 CUDA cores
- CPU: HMP Dual Denver 2/2 MB L2 + Quad ARM A57/2 MB L2
- Video: 4K x 2K 60 Hz codare (HEVC), 4K x 2K 60 Hz decodare (suport 12 biți)
- Memorie: 8 GB, 128 biți, LPDDR4, 58.3 GB/s
- Display: 2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
- CSI: Pana la 6 Camere (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)
- Stocare date: 32 GB eMMC, SDIO, SATA
- Conexiuni: CAN, UART, SPI, I2C, I2S, GPIOs
- USB: USB 3.0 + USB 2.0
- Conectivitate: 1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
- Putere: 7.5 W / 15 W
- Dimensiune: 50 mm x 87 mm (Conector placa-la-placa, compatibil 400 pini compatibil)[1]

Figură 1.3 - Jetson TX2 chit pentru dezvoltatori[1]

1.4 Sistemul de control de la sol (GCS)

Sistemul de control la sol e format dintr-un calculator pentru comandă, control și monitorizare a aeronavei fără pilot. Mission Planner e o aplicație open-source de control la sol pentru autopiloții bazați pe protocolul MAVlink și poate fi rulată pe Windows, Mac OSX și Linux. Mission Planner ne permite să configurăm un avion, copter sau rover să folosească un autopilot, să planifice, să salveze misiuni și să vizualizeze informația live în zbor.



Figură 1.4 - Planificatorul de misiuni

Vom folosi un laptop de pe care observatorul o să selecteze ținta pe care ulterior drona trebuie să o urmărească. Observatorul are și opțiunea de a schimbă ținta dorită, realizând o nouă selecție pentru a putea să actualizeze datele pe care algoritmul le utilizează sau în cazul destul de improbabil în care se pierde obiectul selectat inițial, să reactualizeze poziția acestuia.

La acestes se adaugă o antena Wi-Fi de mare putere folosită pentru a realiza conexiunea între laptopul la care are acces operatorul și drona care va survola în zona de interes.

Capitolul 2 Algoritmi de urmărire

2.1 Algoritmi de urmărire bazați pe filtre MOSSE

2.1.1 Urmărirea bazată pe filtre de corelație

Filtrele de urmărire modelează felul în care percep obiectele folosind filtre antrenate pe imagini date drept exemplu. Obiectul pe care dorim să-l urmărim este încadrat într-o fereastră centrată pe acesta în primul cadru din secvența video. Din acest punct, urmărirea și antrenarea filtrului vor lucra concomitent. Ținta va fi urmărită în urma operației de corelație între filtru și imaginea din cadrul următor; locul în care corelația are valoarea maximă indică noua poziție a țintei. Se va realiza și o reactualizare a încadrării bazată pe noua locație.[2]

Pentru a crea un algoritm de urmărire rapid, calculul pentru operația de corelație este efectuat în spațiul Fourier, mai exact se aplică Transformata Fourier Rapidă (TFR). Inițial, se pleacă de la transformata Fourier bidimensională a imaginii primite ca intrare: $F = \mathcal{F}(f)$, și tot în acest timp se realizează calculul și pentru transformata în spațiul Fourier a filtrului: $H = \mathcal{F}(h)$. După ce se realizează aceste transformări putem aplica teorema Convoluției, iar operația inițială de corelație este transformată într-un proces mult mai simplu de înmulțire element cu element în domeniul Fourier. Folosim simbolul "$\odot$" în continuarea capitolului pentru a specifica o înmulțire de tip element cu element, iar "$*$" pentru a specifica conjugatul complex, în cazul nostru operația se reduce la următoarea formă:

$$G = F \odot H^*$$

Pentru a efectua procesul invers, astfel încât final să obținem ieșirea în domeniul spațial trebuie realizată o operație inversă procesului inițial. Acest lucru se realizează folosind Transformarea Fourier Inversă. Limita vitezei de procesare în acest caz este dată de complexitatea de calcul a celor două procese folosite, Transformata Rapidă Fourier și Transformata Inversă corespunzătoare acesteia, așadar limita superioară de timp a procesului este $O(P\log(P))$, unde P reprezintă numărul de pixeli total dintr-un cadru al secvenței video. [2]

În următoarea parte va fi prezentat modul în care se efectuează preprocesarea datelor din cadrele secvenței video. [2]

O problemă a algoritmului de convoluție TFR este dată de faptul că imaginea și filtrul sunt mapate după structura topologică a unui tor. Mai exact, există o conexiune între marginea stânga a imaginii și marginea dreapta a acesteia, cât și pentru marginea superioară și inferioară. În timpul convoluției, imaginea se rotește în spațiul toroidal comparativ cu translația ce are loc în domeniul

spațial. Introducerea unei legături artificiale între marginile imaginii duce la apariția unor erori de aproximare a rezultatului obținut în urmă corelației. [2]

Pentru a reduce acest efect, valorile fiecărui pixel din cadru sunt transformate folosind o funcție logaritmică, ajuta și în cazul situațiilor de contrastare slabă. Valorile luate de pixeli sunt normalizate cu o valoare medie de 0.0 și cu o normă de 1.0. La sfârșit, imaginea este înmulțită cu fereastră de tip cosinus care are rolul de a reduce valoarea pixelilor din apropierea zonei de zero. Această tehnică are și scopul de a pune mai mult accent pe zona apropiată de centrul obiectului pe care dorim să-l urmărim. [2]

2.1.2 Filtre de tip MOSSE

MOSSE este un algoritm folosit pentru a produce filtre de tip ASEF ce utilizează un set redus de imagini de antrenare. Pentru început, avem nevoie de un set de imagini de antrenament f_i, și un set de rezultate de forma g_i. În general, setul de date g_i, poate lua orice forma. În acest caz, g_i, este generat pornind de la imaginile de antrenament, f_i, asupra cărora o sa se aplice o distribuție gaussiană cu o valoarea a variației egala cu 2. Antrenarea se face în domeniul Fourier pentru a profita de avantajul relației mult mai simple de înmulțire element cu element realizată între intrare și ieșire. Ca și în secțiunea prezentată anterior, o să folosim variabile notate cu majuscule pentru a reprezenta transformatele Fourier. În cazul nostru, F_i, G_i, și filtrul vor fi transformatele Fourier ale reprezentărilor acestora cu caractere mici . [2]

$$H_i^* = \frac{G_i}{F_i}$$

Pentru a calcula filtrul *H* de care avem nevoie pentru a realiza o mapare corectă între intrările folosite pentru antrenare și ieșirile dorite, algoritmul MOSSE o să folosească un filtru, notat *H*, în care o să se caute suma minimă a erorilor pătratice dintre ieșirea actuală și convoluția realizată între intrare și filtrul *H*. Formula utilizată pentru a rezolva această problema o să fie de formă:

$$\min_{H_i^*} \sum_i \left\| F_i \odot H^* - G_i \right\|^2$$

Ideea de a pleca de la minimizarea sumei erorilor pătratice la ieșire nu este ceva nou, chiar este folosită și pentru alte probleme de optimizare. În acest caz, diferența este că se presupune tot timpul faptul că țintă este centrată cu atenție în încadrarea realizată inițial, f_i, la fel și cu ieșirea g_i, ce a fost fixată pentru setul de date pentru antrenament. Aceasta este o idee fundamentală ce stă la baza algoritmilor de urmărire MOSSE și ASEF. Problemele în cazul algoritmilor de urmărire apar deoarece ținta nu este tot timpul centrată, și vârful ce apare pentru g_i se mută pentru a urmări ținta în încadrarea f_i. Într-un caz general, g_i poate să ia orice formă. Putem pleca de la exemplul în care avem mai multe obiecte pe care dorim să le urmărim, iar pentru fiecare va există corespunzător o

casetă de încadrare g_i, de unde rezultă mai multe valori de maxim pentru fiecare chenar de încadrare. [2]

Rezolvarea acestei probleme de optimizare nu este dificilă în mod special, dar trebuie acordată atenție deoarece funcția pe care dorim să o optimizăm va avea ca rezultat o variabilă reală plecând de la o variabilă complexă. În primul rând, fiecare element al filtrului *H*(ce o să fie indexat atât după ω cât și υ) poate fi tratat independent deoarece toate operațiile sunt realizate în domeniul Fourier și fiecare operație se poate realiza element cu element. Această condiție implică rescrierea funcției $H_{\omega\upsilon}$ și $H^*_{\omega\upsilon}$ după ambii termeni. Apoi, va urmă o derivare parțială după $H^*_{\omega\upsilon}$ ce trebuie egalată cu 0 pentru a obține punctul de minim, în timp ce variabilă $H_{\omega\upsilon}$ este tratată independent. [2]

$$0 = \frac{\partial}{\partial H^*_{\omega\upsilon}} \sum_i \left| F_{i\omega\upsilon} \odot H^*_{\omega\upsilon} - G_{i\omega\upsilon} \right|^2$$

Rezolvând pentru H^* o sa obținem o noua expresie specifica filtrului de tip MOSSE:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

În acest caz numărătorul este corelația realizată între intrare și ieșirea dorită , iar la numitor apare spectrul energetic de la intrare. [2]

Pentru a se demonstra faptul că algoritmul de tip MOSSE produce filtre mai performante decât filtrele de tip ASEF, s-a condus un experiment în care a fost modificat numărul inițial de imagini ce intră în procesul de antrenare a filtrului. Filtrele au fost inițializate aplicând mici perturbații aleatoare pe ferestrele de urmărire selectate in primul cadru al secvenței video. Parametrul folosit pentru a compara cele două filtre din punct de vedere calitativ a fost raportul între valoarea maximă și valoarea punctelor din jurul acestuia(PSR). [2]

În urma experimentelor s-a constatat ca algoritmul de urmărire de tip MOSSE produce filtre mai bune când sunt antrenate pe un număr mic de imagini. Analizând filtrul produs de acest algoritm care are la numitor suma energiilor pe mai multe imagini, ce conduc la o suma depărtată de 0, va rezulta în majoritatea cazurilor un filtru mai stabil. [2]

Plecând de la dorința de a obține un algoritm de urmărire performant ce poate funcționa sub condiții de luminozitate diferită sau chiar ocluzie parțială, filtrul trebuie să se adapteze rapid pentru a putea urmării în continuare obiectul. O soluție pentru îmbunătățirea performanțelor acestor algoritmi de urmărire a fost utilizarea unor parametrii de regularizare. [2]

Formulele de calcul reactualizate pentru algoritmul de urmărire MOSSE luând în calcul și parametrii de regularizare:

$$H_i^* = \frac{A_i}{B_i}$$

$$A_i = \eta G_i * F_i^* + (1-\eta) A_{i-1}$$

$$B_i = \eta F_i * F_i^* + (1-\eta) B_{i-1}$$

Parametrul η reprezintă rata de învățare. Acesta are rolul de a pune mai mult accent pe cadrele recente decât pe cadrele ce au fost cu mult timp în urmă. În practică, se folosește un $\eta = 0.125$ ce ajută ca filtrul să fie mult mai robust și totodată să-și păstreze adaptabilitatea în momentul în care intervin schimbări. [2]

2.1.3 Detecție eronata si parametrul PSR

După cum am menționat și în paragrafele anterioare, pentru a putea măsură puterea vârfului dat de algoritmul de urmărire o să folosim o metodă denumită PSR(vârf raportat la regiune laterală). Pentru a calcula valoarea PSR-ului ieșirea corelației g_i este împărțită în două părți: vârf, care reprezintă valoarea maximă, și regiunea laterală, care este compusă din restul pixelilor din fereastră de urmărire, mai puțin o regiune de 11 * 11 pixeli ce înconjoară punctul de maxim. [2]

Formula de calcul este:

$$PSR = \frac{g_{\max} - \mu_{s1}}{\tau_{s1}}$$

$g_{\max}$ - valoarea maximă (vârful)

μ_{s1} - media regiunii laterale

σ_{s1} - varianța regiunii laterale

2.2 Algoritmi de urmărire bazați pe tehnica Adaboost de selecție a trăsăturilor

În acest subcapitol o să fie descris algoritmul de urmărire bazat pe tehnica Adaboost, denumire ce provine de la expresia Adaptive Boosting, și presupune că urmărirea unui obiect poate fi privită ca o problemă de clasificare binară unde clasa pozitivă o să fie obiectul urmărit(un set de date în care obiectul apare în condiții diferite), iar clasa negativă este compusă din background sau orice alt obiect în afară celui urmărit.[3]

Pentru o înțelegere mai bună o să definim trei termeni ce stau la baza algoritmului prezentat:

- **clasificator slab**: un clasificator care are performanțe scăzute are în general o probabilitate de reușită puțin mai mare decât o alegerea aleatoare(ex.: în cazul unei clasificări binare, eroarea acestuia trebuie să fie mai mică decât 50%); notația acestuia este h^{weak} și corespunde pentru o caracteristică obținută după aplicarea unui algoritm de învățare; [3]
- **selector**: dintr-un set de M clasificatori slabi $h^{weak} = \left\{ h_1^{weak}, h_2^{weak}, ..., h_M^{weak} \right\}$, cu un selector va alege un sigur element dintre acestea; [3]

$$h^{sel}(\mathbf{x}) = h_m^{weak}(\mathbf{x})$$

- **clasificator puternic**: plecând de la un set de *N* clasificatori slabi, un clasificator puternic este calculat pornind de la o combinație liniară a selectorilor. [3]

Se dorește obținerea unui clasificator „puternic", iar asta se face plecând de la o serie de clasificatori „slabi" și a selectorilor corespunzători. Algoritmi utilizați în general pentru clasificatori „slabi" sunt arborii binari de decizie sau Nearest Neighbour. Pornind de la *M* clasificatori slabi $\left\{h_1^{weak}, h_2^{weak}, ..., h_M^{weak}\right\}$, se folosește un selector h^{sel} cu ajutorul căruia se va selecta un clasificator „slab" ce minimizează o funcție de cost, cum a fost prezentată și mai devreme:

$$h_n^{sel}(\mathbf{x}) = h_{m^*}^{weak}(\mathbf{x})$$

unde m^* reprezintă indexul clasificatorului „slab", unde există un cost minim. Întreg sistemul are în componență un număr total de *N* selectori. Ideea de bază a acestui algoritm de urmărire consta în introducerea *selectorilor*. Rolul unui selector este de a determina care este cel mai bun clasificator „slab" pentru fiecare tip de trăsătura extrasă. Acesta este și motivul pentru care se poate spune că algoritmul de boosting poate fi considerat un algoritm de selecție de trăsături. [3]

Într-un final, rezultatul unui clasificator „puternic" pentru un pachet *x* oarecare este dat de următoarea formula de calcul:

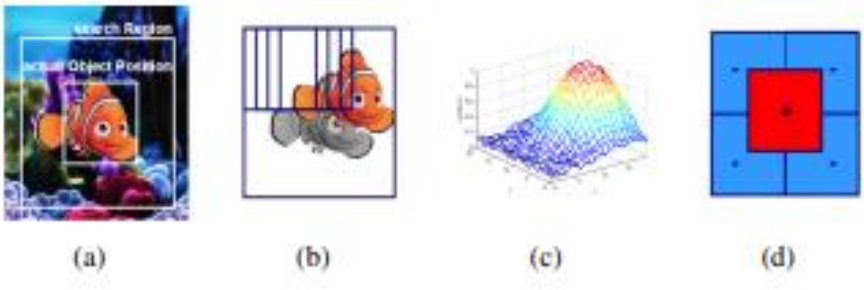
$$h^{strong}(x) = sign\left(\sum_{n=1}^{N} \alpha_n h_n^{sel}(x)\right) = \begin{cases} 1, pentru \sum_{n=1}^{N} \alpha_n h_n^{sel}(x) \geq 0 \\ -1, pentru \sum_{n=1}^{N} \alpha_n h_n^{sel}(x) < 0 \end{cases}$$

unde 1 reprezintă clasa în care se află obiectul pe care trebuie să-l urmărim, iar pentru valoarea de -1 o să corespundă clasei restului de obiecte din imagine. Parametrul α_n este pondera asociată termenului de pe poziția *n* din suma și se calculează cu următoarea formulă:

$$\alpha_n = \frac{1}{2}\log\frac{1-\varepsilon_n}{\varepsilon_n}$$

ε_n - costul selectorului *n*[3]

În Figura 2.1 este o reprezentare mult mai simplistă a modului in care algoritmul funcționează.



Figură 2.1 - Principiul de urmarire[3]

În această figura se pot observă cei 4 pași importanți din cadrul unui algoritm de urmărire ce are la baza clasificatori. Pornind de la selecția inițială a obiectului (a) la momentul de timp *t*, clasificatorul este evaluat pe o regiune vecină cât mai mare pentru a găsi obiectul selectat în cadrul imediat următor, *t + 1*. În urmă acestei evaluări se obține o harta de încredere (c) ce este analizată cu scopul de a obține cea mai probabilă poziție în care se află obiectul urmărit, iar apoi algoritmul este reactualizat. [3]

Acest algoritm de urmărire poate face urmărire de mai multe obiecte concomitent, cu condiția că trebuie inițializat un clasificator separat pentru fiecare obiect ce se dorește să fie urmărit. [3]

Antrenarea unui astfel de sistem se face în timp real, utilizatorul trebuind doar să marcheze obiectul ce se dorește a fi urmărit în primul cadru secvența video. [3]

2.3 Algoritmi de urmărire bazați pe tehnica Multiple Instance Learning (MIL)

2.3.1 Urmărirea realizată utilizând tehnica MIL

În acest subcapitol este introdus algoritmul de urmărire în timp real de tip MIL. O să fie prezentată o privire de ansamblu asupra acestui sistem de urmărire ce va include și o descriere a modelului de mișcare utilizat, iar după va fi discutat și modul de funcționare al algoritmului MIL. [4]

2.3.2 Descrierea sistemului și modelului de mișcare

Urmărirea obiectelor ce au la baza un algoritm ce utilizează o tehnică MIL are trei componente fundamentale:

- reprezentarea imaginii;
- modelul de clasificare;
- modelul de mișcare. [4]

Trăsăturile extrase din imagini sunt coeficienți calculați prin aplicarea unor filtre Haar asupra imaginilor. Aceste trăsături au avantajul unei complexități de calcul reduse, deci a unui timp de rulare foarte mic. În plus, este acceptabilă o rotire moderată a obiectelor în raport cu poziția inițială. [4]

Model discriminativ de clasificare unde sunt calculate probabilitățile $p(y=1|x)$ și $p(y=0|x)$ unde $y = 1$ și $y = 0$ reprezintă prezența, respectiv absența obiectului urmărit din pachetul analizat, *x*. La fiecare moment de timp *t*, algoritmul memorează locația obiectului l_t^*. Funcția $l(x)$ o să returneze locația pachetului de date *x*. Pentru fiecare nou cadru se determină un set de pachete $X^s = \left\{x \mid s > \left\| l(x) - l_{t-1}^* \right\|\right\}$ în apropierea locației , notate cu *s*, a obiectului. [4]

Nu este menținută o distribuție pentru locația țintei la fiecare cadru, în schimb se folosește un model de mișcare unde locația obiectului urmărit la momentul de timp *t* este la fel de probabilă să apară într-o anumită rază față de poziția la momentul de timp anterior, *t-1*:

$$p(l_t^* \mid l_{t-1}^*) = \begin{cases} 1, daca \left\| l_t^* - l_{t-1}^* \right\| < s \\ 0, altfel \end{cases}$$

Odată ce locația algoritmului este actualizată, se va realiza și actualizarea modelului. Pentru acesta este nevoie să utilizăm o nouă rază *r*, unde $r < s$ iar noul set de pachete o să fie de formă $X^r = \left\{x \mid r > \left\| l(x) - l_t^* \right\|\right\}$ și etichetat ca un bag de date pozitive. Pentru a obține un bag de date cu exemple negative vom decupa o regiune cuprinsă între raza *r*, folosită pentru a obține exemple pozitive și scalarul β. Din moment ce există pericolul de a obține un set mare de date, se lucrează doar cu un subset de imagini ales aleator din setul generat. Forma matematică pentru generarea exemplelor negative este următoarea: $X^{r,\beta} = \left\{x \mid \beta > \left\| l(x) - l_t^* \right\| > r\right\}$. [4]

2.3.3 Algoritmul MIL

Algoritmii discriminativi de învățare clasici folosiți pentru antrenarea clasificatorilor binari ce utilizează o estimare a lui $p(y|x)$ necesită inițial un set de date de antrenament de formă $\left\{\left(x_1, y_1\right), ..., \left(x_n, y_n\right)\right\}$ unde x_i reprezintă o instanță a vectorului de caracteristici pentru un pachet de imagini, iar y_i care reprezintă o etichetă binară ce spre exemplu poate lua valori de 1 și 0. Pentru algoritmul bazat pe tehnica MIL setul de date de antrenament este de forma $\left\{\left(X_1, y_1\right), ..., \left(X_n, y_n\right)\right\}$ unde $X_i = \left\{x_{i1}, ..., x_{im}\right\}$ reprezintă un bag, iar y_i va fi eticheta acestuia. Formula de calcul pentru y_i are următoarea formă de calcul:

$$y_i = \max_j \left(y_{ij}\right)$$

unde y_{ij} sunt instanțe pentru toate etichetele, care se presupun că există, dar nu sunt cunoscute în timpul antrenării. Cu alte cuvinte, un bag este considerat pozitiv atunci când există cel puțin o valoare pozitivă a unei instanțe din mulțimea de etichete y_{ij}. Numeroși algoritmi au fost propuși pentru a rezolva problemele pe care la are tehnica MIL. O formă a ecuației pentru rezolvarea anumitor problemelor pe care le are această tehnica folosește o antrenare a clasificatorului care maximizează probabilitatea logaritmului pentru fiecare bag:

$$\log L = \sum_i \left(\log p\left(y_i \mid X_i \right) \right)$$

De observat este și faptul că probabilitatea se calculează pe fiecare bag și nu pe fiecare instanță a unei etichete, deoarece în timpul antrenamentului instanțele sunt necunoscute, și totodată scopul principal este antrenarea unui clasificator cu ajutorul căruia se poate estima probabilitatea $p(y \mid x)$ a unei instanțe. Prin urmare, trebuie exprimată probabilitatea $p\left(y_i \mid x_{ij} \right)$ ca un bag să fie pozitiv. Pentru acesta a fost folosit modelul Noisy-OR(NOR) ce are următoarea formă:

$$p\left(y_i \mid X_i \right) = 1 - \prod_j \left(1 - p\left(y_i \mid x_{ij} \right) \right)$$

unde probabilitățile de forma $p(y \mid x)$ sunt date de ecuația:

$$p(y \mid x) = \sigma\left(h^{strong}\left(x \right) \right) = \frac{1}{1 + e^{-h^{strong}(x)}}$$

$h^{strong}\left(x \right)$ - este un clasificator „puternic” format folosind tehnica de selecție a trăsăturilor Adaboost prezentată anterior. [4]

Ecuația anterioară are proprietatea următoare: dacă una din instanțele dintr-un bag are o probabilitate ridicată, atunci probabilitatea bag-ului respectiv va fi și ea ridicată, în concordanța cu cea a instanței. [4]

2.3.4 Pașii algoritmului bazat pe tehnica MIL

Algoritmul MIL poate fi rezumat în urma a ceea ce a fost prezentat anterior cu o serie de pași după cum urmează:

1. Determinarea unui set de pachete X_s în vecinătatea unei locații a obiectului determinate la pasul anterior;
2. Estimare $p(y = 1 \mid x)$ pentru fiecare $x \in X_s$;
3. Actualizarea locației prin găsirea pachetului $x \in X_s$ care maximizează $p(y = 1 \mid x)$;
4. Determinarea a două subseturi de pachete, unul cu pachete aflate în vecinătatea noii locații (acest subset va fi etichetat ca fiind pozitiv), iar altul cu pachete aflate în exteriorul vecinătății (acest subset va fi etichetat ca fiind negativ);
5. Actualizarea modelului de clasificare folosind subseturile determinate la punctul anterior.

2.4 Algoritmi de urmărire bazați pe filtre de corelație și metoda kernel

Majoritatea algoritmilor de urmărire tradiționali au fereastră fixă folosită pentru căutarea în regiunea de interes și în mod obișnuit au la bază un prag fixat cu o valoare constantă pe toată durata urmăririi și nu sunt luate în calcul și informațiile despre starea actuală a obiectului, ce poate diferi de

la un cadru la altul din cauza condițiilor în care se află obiectul. În consecință, în aceste cazuri este mult mai probabil să avem o eroare în încercarea algoritmului de a urmări cu exactitate obiectul urmărit. [5]

În cazul folosirii unor modele discriminative precum Adaboost o să antrenăm în timpul rulării algoritmului o serie de clasificatori binari. Problema principală apare din cauza redundanței existente între seturile de antrenare unde sunt foarte mulți pixeli/pachete ce vor fi la fel. Tehnică MIL are o rezolvare pentru această problema, aici este ales un număr de exemple negative redus prin alegerea aleatoare a unui număr limitat de exemple. [5]

Tehnica KCF (Kernelized Correlation Filtering) își propune să învețe modele în mod eficient fără a reduce numărul de exemple. De asemenea, este folosită transformata Fourier care transformă o operație de convoluție între două semnale într-o înmulțire a transformatelor Fourier corespunzătoare semnalelor. [5]

Scopul antrenării unui model liniar este să determine o funcție $f(x)=\mathbf{w}^T\mathbf{x}$ care să minimizeze eroarea medie pătratică între $f(x_i)$ și valorile țintă y_i:

$$\min_{\mathbf{w}} \sum_i \left(\left(f(x_i) - y_i \right)^2 + \lambda \|\mathbf{w}\|^2 \right)$$

în care al doilea termen al sumei este folosit pentru regularizare și controlul unei învățări pe de rost. Soluția unei probleme ca cea de mai sus este:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1} \cdot \mathbf{X}^T\mathbf{y}$$

unde $\mathbf{X}$ este o matrice formată din vectorii $\mathbf{x}_1, \mathbf{x}_2, \ldots$ pe linii, $\mathbf{I}$ este matricea identitate, iar $\mathbf{y}$ este un vector format din valorile țintă y_i. Dacă elementele sunt numere complexe, $\mathbf{w}$ se calculează ca:

$$\mathbf{w} = (\mathbf{X}^H\mathbf{X} + \lambda\mathbf{I})^{-1} \cdot \mathbf{X}^H\mathbf{y}$$

unde $\boldsymbol{X}^H = (\boldsymbol{X}^*)^T$ este conjugat-transpusa matricii $\boldsymbol{X}$. [5]

Pentru a extrage parametrii pentru $\mathbf{w}$ se va folosi o transformată Fourier discretă inversă după modelul: $\mathbf{w} = IDFT\left(DFT\left(\mathbf{w}\right)\right)$.[5]

Evaluarea pentru un pachet de test, $\mathbf{z}$, se face folosind funcția f:

$$f(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i \kappa(z, \mathbf{x}_i)$$

în care $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ sunt pachete din setul de antrenare, $\kappa(\mathbf{z}, \mathbf{x}_i) = \varphi^T(\mathbf{z}) \cdot \varphi(\mathbf{x}_i)$ se numește funcție kernel, $\varphi(\mathbf{x})$ este vectorul de trăsături corespunzător pachetului $\mathbf{x}$.[5]

În cazul în care datele de antrenare sunt permutări circulare ale vectorului $\boldsymbol{x}$, atunci coeficienții $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_N]$ se determină, în domeniul frecvență ca:

$$DFT(\alpha) = \frac{DFT(\mathbf{y})}{DFT\left(\mathbf{k}^{\mathbf{xx}}\right) + \lambda} \quad (36)$$

unde $\boldsymbol{k^{xx}}$ este prima linie din matricea kernel formată din elemente $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j), i, j = 1 \dots N$, $\boldsymbol{x}_i$ **și** $\boldsymbol{x}_j$ fiind permutări circulare ale lui $\boldsymbol{x}$. [5]

În cazul tehnicii KFC sunt 2 proceduri de baza:

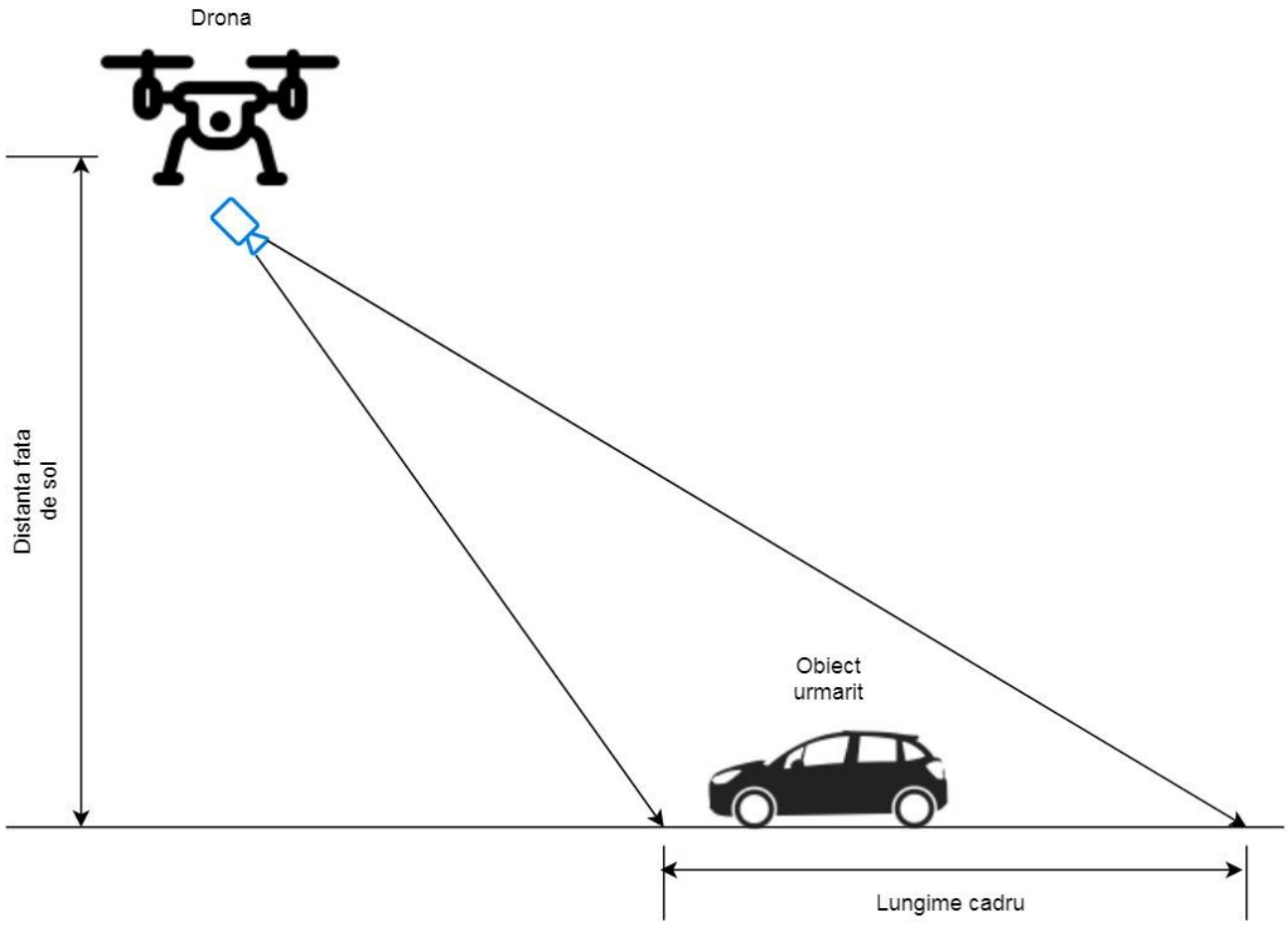
- Antrenare
- Testare

Cele două proceduri sunt rapid de efectuat și permit urmărirea în timp real. Detecția constă în aplicarea unui prag valorilor calculate pentru fiecare pachet de test în parte. [5]

Capitolul 3 Urmărire pe dronă

Pentru a realiza urmărirea cu ajutorul dronei au fost implementate două funcții folosite pentru deplasarea, iar drona va executa comenzile de mișcare primite. În primul caz, în funcția dezvoltată am setat vitezele pe care o să se deplaseze drona pe cele 2 axe în urma poziției obiectului urmărit în cadrul curent folosind o viteză constantă de deplasare, iar in al doilea caz, am folosit coordonatele GPS al obiectului urmărit față de poziția curentă a dronei.

Vom porni prin a prezenta situația de ansamblu, mai exact modul în care se realizează captarea secvențelor video cu ajutorul dronei. După cum se poate observa și în Figura 3.1 drona este lansată până la o anumită înălțime față de sol iar cu ajutorul camerei înclinate sub un anumit unghi față de axul dronei se va realiza captarea secvențelor video.



Figură 3.1 - Captarea secvențelor video din dronă

Pentru a explică cum lucrează funcțiile vom vorbi puțin și despre modul în care este realizată comunicația între dronă și aplicație. Comunicare între cele două se face prin intermediul protocolului

de comunicație MavLink. Mesajul transmis prin protocolul MavLink este la baza un buffer de 17 octeți codați de Mission Planner. Are loc și o codare ce presupune doar aranjarea datelor într-o structură deja stabilită și adăugarea unor octeți pentru corecția de erori. Cei 17 octeți ce compun mesajul sunt împărțiți in 3 categorii: 6 octeți header + 9 octeți mesaj de interes + 2 octeți de control. [6]

Octeții din header sunt împărțiți după cum urmează:

- octet 0 – header mesaj
- octet 1 – lungimea mesajului
- octet 2 – numărul secvenței
- octet 3 – System ID
- octet 4 – Componet ID
- octet 5 – Message ID

Folosind octeți din mesajul de interes transmitem comanda pe care dorim să o efectuăm, dacă dorim ca informația pe care o transmitem să nu aibă erori este indicat să folosim o rată de bit cât mai mică, spre exemplu este bine să utilizăm rata de 57 000 *bps* în defavoarea celei de 115200 *bps*, chiar dacă în al doilea caz informația transmisă este mai rapid recepționată, există posibilitatea mai mare de a transmite date greșite. [6]

În cele ce urmează o să intrăm în amănunt cu prezentarea mai exactă a modului în care se realizează deplasarea în cele 2 cazuri.

3.1 Urmărirea prin reglarea vitezelor

Acest tip de urmărire este realizat într-un mod ce are la baza o teorie destul de simplă, dar totodată o putem considera eficientă deoarece urmărirea țintei este realizată corect, având la bază datele obținute în simulator.

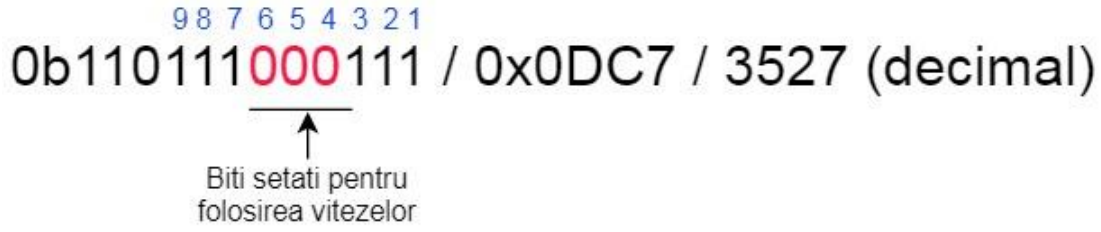
Urmărirea în acest caz o să o facem cu ajutorul funcției *set_velocity_body()*, în care apelăm funcția de deplasare din cadrul biblioteci *pymvalink* folosită și de pilotul automat, cum am amintit și într-un paragraf anterior. Funcția de control a vitezelor în zbor, ce apare cu următorul nume, *set_position_target_local_ned_encode()*, folosește o serie de parametrii pentru a seta corect modul în care vrem să utilizăm drona. Mai jos se pot observa parametrii de care funcția are nevoie:

```
msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,
        0, 0,
        mavutil.mavlink.MAV_FRAME_BODY_NED,
        0b0000111111000111, #-- masca de biti --> setata doar pentru viteze
        0, 0, 0,        #-- DISTANTE
        vx, vy, vz,     #-- VITEZE
        0, 0, 0,        #-- ACCELERATII
        0, 0)
```

Figură 3.2 - Modul de apelare a funcției *set_position_target_local_ned_encode()*

Biți de interes din masca noastră sunt cei notați de la 1 la 9 în Figura 3.3, în cazul de față, se lucrează cu logică negativă, iar setarea biților 0 implică utilizare lor, așadar, setarea biților 4, 5 și 6

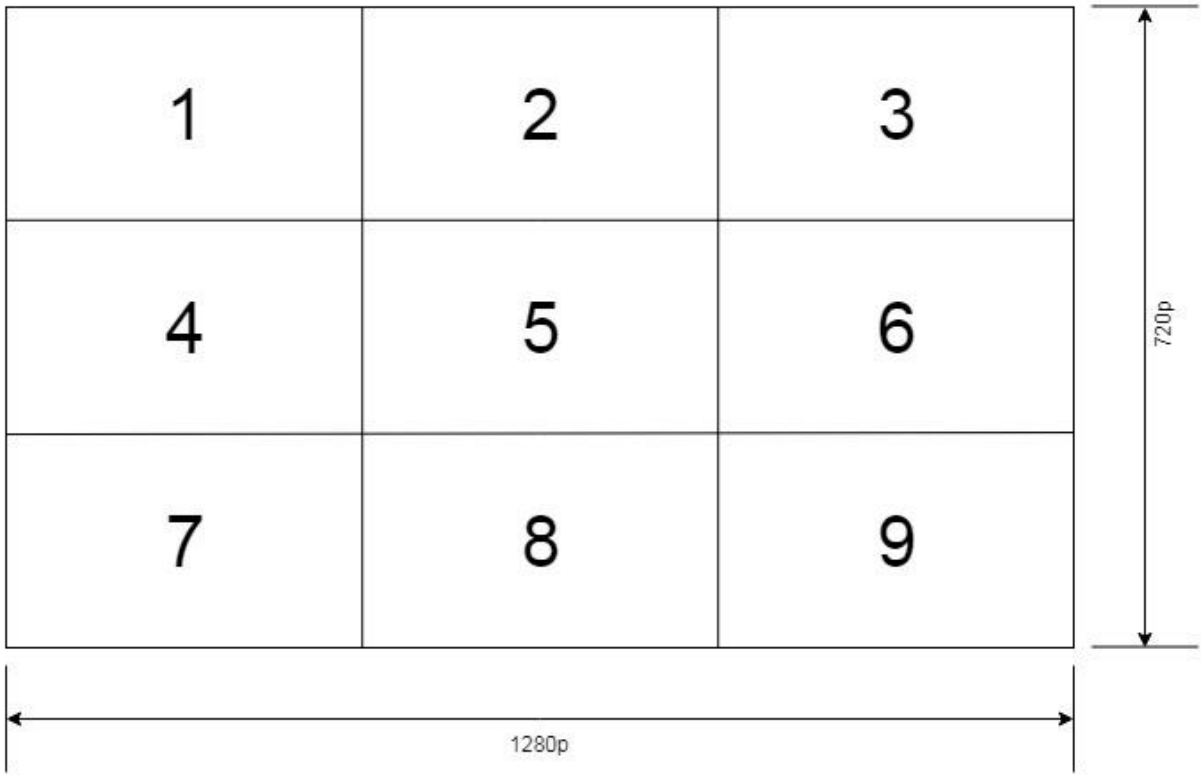
cu valoarea 0 implică utilizarea modului în care vom comunica viteze pentru deplasarea dronei. Tot în Figura 3.3 mai sunt trecute și valorile în hexazecimal și în zecimal pentru a seta valoarea dorita în biți astfel încât modul de deplasare să fie cel care utilizează viteze.



Figură 3.3 - Masca de biți

Setare biților de pe pozițiile 1, 2 și 3 va duce la o utilizare a dronei cu deplasare pe distanțe, iar setarea biților 7, 8 și 9 pentru o utilizare folosind accelerații. Folosirea unui mod nu implică o exclusivitate a acestuia, pot fi folosite și combinații, cum ar fi distanță și viteză.

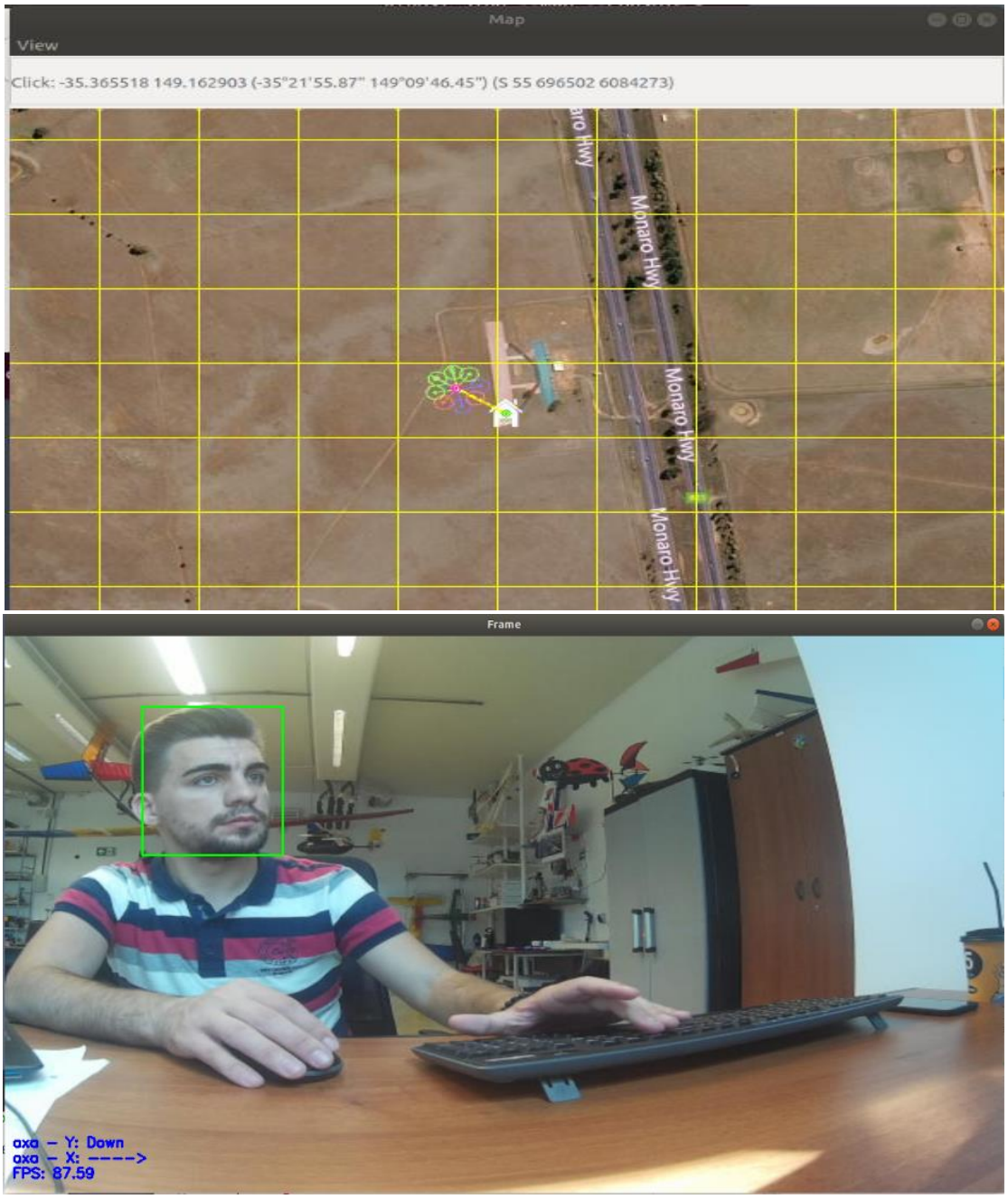
Pentru efectuarea unei urmăriri corecte am plecat de la o idee simplistă în care cadrul imaginii a fost împărțit în 9 regiuni ca în figura următoare:



Figură 3.4 - Împărțirea pe regiuni a cadrului imaginii

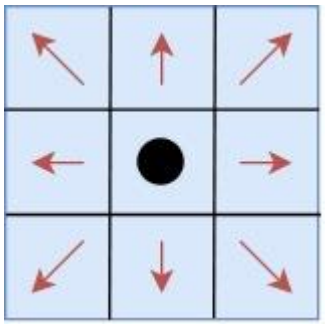
În cazul nostru lățimea și înălțimea imaginii sunt de 1280 pixeli, respectiv 720 pixeli . În continuare o să vorbim despre modul de funcționare, iar cel mai ușor de explicat este prezentând un exemplu.

Pentru exemplificare am atașat Figura 3.5 care reprezintă o captură de ecran din momentul în care este rulată aplicația. Obiectul urmărit se află cu centrul de greutate în prima regiune, așadar va exista o mișcare a dronei pe direcția corespunzătoare din Figura 3.6. Urmând aceeași idee prezentată anterior, mișcarea realizată de dronă o să fie corespunzătoare hărții de direcții din Figura 3.6, singurul caz special fiind în regiunea numărul 5, aici drona va staționa considerând că obiectul a fost centrat cu succes.



Figură 3.5 - Obiect urmărit în regiunea 1 și direcția pe care circulă drona

După cum se poate observa în Figura 3.6, față de poziția inițială, reprezentată cu simbolul unei case pe simulator, drona se va deplasa pe harta in direcția corespunzătoare regiunii 1.



Figură 3.6 - Direcțiile corespunzătoare fiecărei regiuni

Această modalitate de deplasare a dronei este utilizabilă și în cadru real dar are un dezavantaj, și anume fixarea vitezei. Se poate lucra și la un mecanism de implementare a unei viteze adaptive în funcție de numărul de pixeli pe care drona trebuie să-l parcurgă pentru a efectua centrarea corectă a obiectului urmărit dar necesită încă o operație de calcul realizată pentru fiecare cadru, și reapelarea funcției cu ajutorul căreia se realizează setarea vitezei.

3.2 Urmărirea prin setarea coordonatelor GPS

Pentru o mai bună înțelegere a modului ce are la baza urmărirea prin coordonate GPS vom vorbi puțin despre ce este GPS-ul și ce se află la baza acestuia.

Principiul de funcționare este destul : GPS-ul are la bază un satelit special proiectat pentru îmbunătățirea sistemelor de navigare radio. Există în prezent un număr de 24 de astfel de sateliți care orbitează în jurul Pământului şi care transmit semnale radio în mod continuu. Sistemul funcţionează pe baza unui algoritm de calcul al timpului parcurs de către un semnal radio emis de satelit care parcurge distanţa până la un obiect predeterminat, aflat într-o locaţie pe Pământ şi înapoi. Receptoarele GPS primesc un semnal pe care mai apoi îl decodează printr-o serie de calcule în cifre ce reprezintă cele trei coordonate: latitudinea, longitudinea şi altitudinea. GPS a fost creat de către departamentul american al Apărării în scopuri militare, dar se află şi la dispoziţia utilizatorilor civili din toată lumea în mod gratuit. Sistemul GPS determină o locaţie prin calcularea diferenţei între timpul la care un semnal este transmis de către satelit şi timpul la care este recepţionat de către receptorul de pe Pământ. Sateliţii GPS sunt echipaţi cu ceasuri atomice care asigură o măsurare a timpului extrem de precisă. Informaţia despre timp este plasată într-un cod care este transmis de către satelit în așa fel încât receptorul să poată determina în mod continuu timpul în care semnalul s-a propagat. Semnalul respectiv conţine datele de care un receptor are nevoie pentru a calcula locaţia satelitului şi a face ajustările necesare pentru o poziţionarea precisă a acestuia. Receptorul foloseşte diferenţa de timp dintre recepţia semnalului şi timpul de emisie pentru a calcula distanţa dintre acesta şi satelit. Receptorul trebuie să țină cont si de întârzierile de propagare sau scăderile de viteză ale semnalului, cauzate de mediul prin care acesta trec, mai exact prin ionosferă şi troposferă. Dacă are informaţii în legătură cu distanţele faţă de trei sateliţi şi cu localizarea satelitului în momentul în care acesta trimite semnalul, receptorul îşi poate calcula astfel propria poziţie tridimensională. Un ceas

atomic sincronizat este necesar pentru a putea calcula distanţele faţă de sateliţi din aceste trei semnale, oricum, luând în considerare măsurătorile de la un al patrulea satelit, receptorul evită folosirea unui ceas atomic la receptor (care ar face acest sistem prohibitiv de scump, un asemenea echipament având costuri ridicate). În concluzie, receptorul are nevoie de patru sateliţi pentru a-şi calcula longitudinea, latitudinea, altitudinea şi timpul.).[12]

Ideea urmăririi prin setarea coordonatelor GPS păstrează și unele elemente din metoda folosită anterior. Vom folosi în continuare împărțirea în 9 regiuni a imaginii doar că acum trebuie ținut cont de faptul că drona nu mai primește viteze pentru a realiza urmărirea obiectului, doar distanțe ce sunt relative la sistemul de referință al dronei.

De această dată urmărirea va fi realizată cu funcția *goto_position_target_local_ned(nord, est, altitudine)*, în care parametrii trimiși acesteia vor fi distanțele pe nord și est, dar și altitudinea la care dorim să se facă urmărirea. Se observă în Figura 3.7 că este nevoie de transmiterea ultimilor trei biți cu valoare de 0 pentru setarea acestui mod.

```
msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,
        0, 0,
        mavutil.mavlink.MAV_FRAME_LOCAL_NED,
        0b0000111111111000, #-- maca de biti - setata pentru distante
        north, east, down,
        0, 0, 0, #-- x, y, z VITEZE
        0, 0, 0, #-- x, y, z ACCELERATII
        0, 0)
```

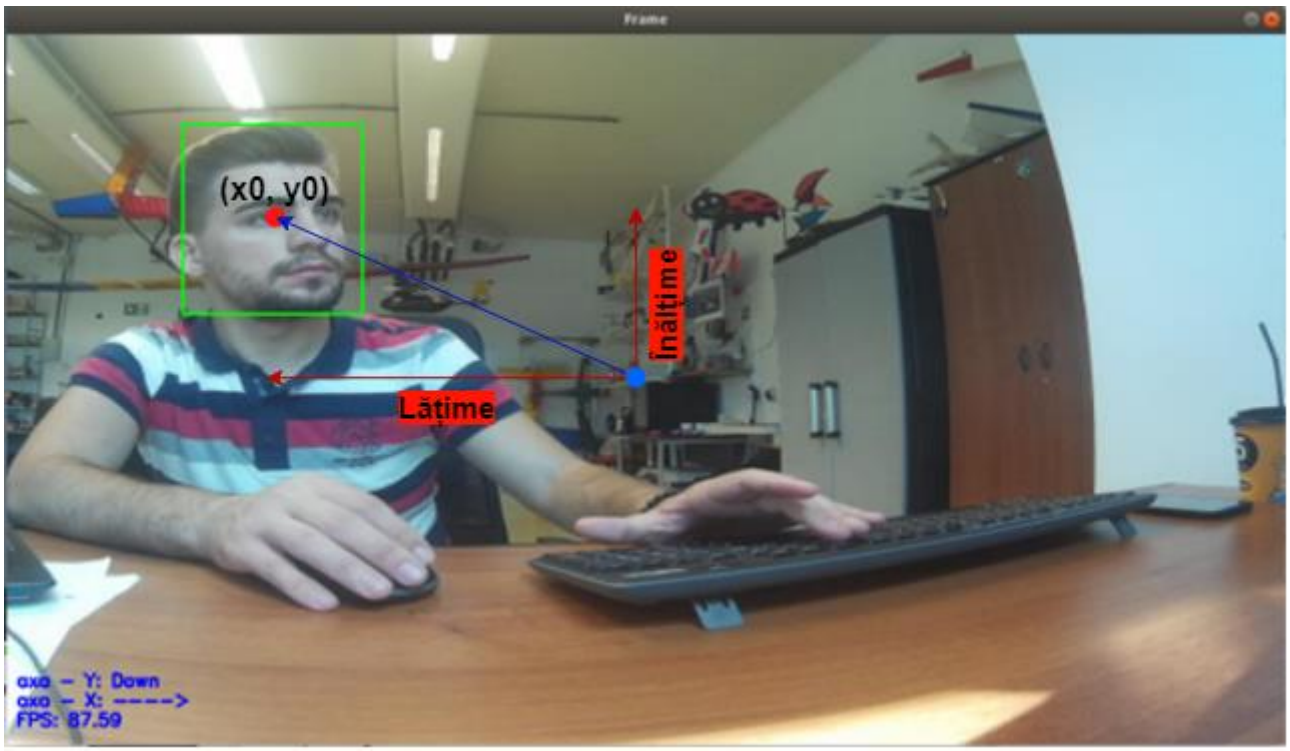
Figură 3.7 - Setarea măști de biți pentru folosirea distanțelor

Drona își memorează poziția GPS inițială la punerea în funcțiune, deci poate fi folosit un sistem de referință al navigației(local).

Luând în calcul că distanțele ce trebuie transmise către drona trebuie să fie relative la sistemul de referință al acesteia a fost creat un algoritm care să îndeplinească această condiție. Pentru acesta s-au folosit două variabile, distanță parcursă și raporta la nord alături de distanță parcursă și raportată la est. Pentru fiecare cadru distanțele obiectului urmărit față de poziția inițială sunt actualizate ținând cont și de poziția GPS actuală a dronei.

În Figura 3.8 putem observa că apar doi vectori ce stau la baza acestui mod de deplasarea. Ei reprezintă valorile în pixeli cu ajutorul cărora se vor calcula distanțele pe care trebuie să le transmitem funcției folosite pentru a realiza urmărirea.

Pentru a determina poziția absolută a țintei, avem nevoie de poziția dronei, pe care o obținem prin comunicația serială dintre pilotul automat (dotat cu receptor GPS, accelerometre, barometru și giroscoape) și computerul îmbarcat. Mai folosim și unghiurile de atitudine ale dronei și poziția relativă a țintei fața de dronă.



Figură 3.8 - Vectorii pentru calculul distanțelor

Pentru a determina poziția obiectului relativ față de dronă, atunci când camera este îndreptată la orizontală fată de axa pământului ne folosim de câmpul vizual al camerei, FOV, înălțimea la care se află aeronava, h și poziția acesteia în poză, (l_p, L_p), raportată la lățimea sau lungimea pozei l_{imag}, respectiv L_{imag}.

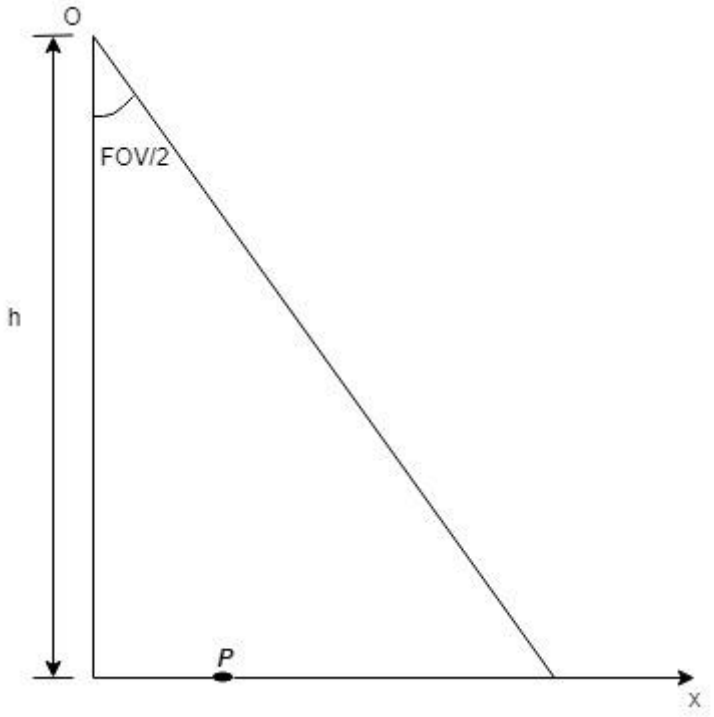
Obținem astfel:

$$\begin{cases} x = h \cdot \tan\frac{FOV}{2} \dfrac{L_p - \frac{L_{imag}}{2}}{\frac{L_{imag}}{2}} \\ y = h \cdot \tan\frac{FOV}{2} \dfrac{l_p - \frac{l_{imag}}{2}}{\frac{l_{imag}}{2}} \end{cases}$$

unde $\dfrac{L_p - \frac{L_{imag}}{2}}{\frac{L_{imag}}{2}}$ este factorul de scară.

În Figura 3.9 există și o reprezentare grafică pentru a vedea concret modul în care se aplică formula obținută anterior, dar și ce reprezintă fiecare parametru într-un cadru real.

În cele ce urmează vom discuta despre modul în care se realizează urmărirea în cazul prezentat. Variabilele de interes le putem observa in Figura 3.8, *Lățime* și *Înălțime*, care vor fi corespunzătoare notațiilor de L_{imag} și l_{imag} din formula prezentată anterior. Punctul de culoare albastră ce reprezintă centrul imaginii și corespunde unor valorilor $x = 0$ și $y = 0$ deoarece există egalitate între L_p și $\frac{L_{imag}}{2}$, respectiv l_p și $\frac{l_{imag}}{2}$.



Figură 3.9 - Determinarea poziției relative a țintei

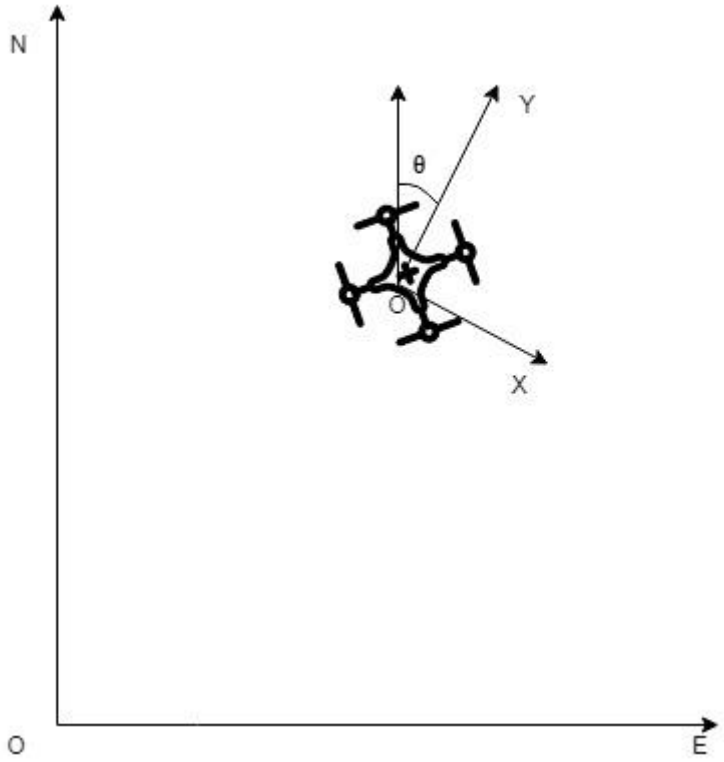
Presupunem că obiectul urmărit ajunge în coordonatele x_0 și y_0, în acest moment drona va începe deplasare spre poziția respectivă pentru a încerca să centreze obiectul urmărit, conform teoriei prezentate și în metoda anterioară.

Mișcare se va efectua pe diagonala realizată de cele două distanțe, deoarece este mult mai practică această abordare decât o parcurgere secvențială pentru fiecare din cele două distanțe. După cum am amintit vom păstra în continuare și hartă cu împărțirea în regiuni a imaginii, deoarece nu dorim să existe o deplasare continuă a dronei când obiectul urmărit nu-și schimbă cu mult poziția față de centrul imaginii. Această condiție ne ajută să prelungim autonomia bateriei și avem și o marja de eroare cu care putem urmărim obiectul.

Trebuie ținut cont și de faptul că drona are un sistem local de referință, iar pentru fiecare punct în care se deplasează în încercarea de a urmări obiectul, sistemul propriu de referință se schimbă conform direcției dată de cele două distanțe. Acest lucru se poate vedea și în Figura 3.9 în care se observă că sistemul de referință al dronei are un unghi de girație, θ, față de sistemul global de referință.

Luând în calcul acest detaliu, urmărirea obiectului cu ajutorul dronei se poate realiza în 2 moduri: folosim locația unde se deplasează ținta să calculăm noile coordonate GPS ale acesteia

raportate la sistemul global de referință sau folosim datele poziției curente ale dronei pentru a realiza deplasarea spre obiect față de poziția GPS actuală a dronei, precum si orientarea acesteia.



Figură 3.10 - Determinarea poziției țintei cunoscând coordonatele dronei și poziția țintei față de aceasta

Când folosim pentru fiecare cadru o estimare a coordonatelor GPS, există posibilitatea să apară o eroare în estimarea acestor coordonate în timp, iar urmărirea obiectului poate da greș din această cauză. Așadar, a fost preferată metoda de urmărire ce ține cont de poziția GPS actuală a dronei și estimarea coordonatelor GPS în funcție de sistemul actual de referință.

Capitolul 4 Experimente și aplicația finală

4.1 Alegerea algoritmului

Alegerea algoritmului a fost realizată având la bază o întreagă serie de teste ce au fost realizate în prealabil. În aceste teste s-au urmărit o serie de factori prezentați în introducere, dintre care cei mai importanți au fost performanță și precizia. Performanță a fost un criteriu deosebit de important deoarece am vrut să realizăm urmărirea obiectelor în timp real.

Evaluarea performanțelor s-a realizat pe un laptop cu configurația din Tabelul 4.1 și trebuie menționat faptul că nu am avut implementată partea de mișcare a dronei ce scade din viteza de execuție a algoritmului. A fost luat în calcul și faptul că procesorul pe care s-au realizat inițial testele nu era unul specializat pe partea de prelucrare de imagine, deoarece era un procesor de uz general, nu specializat.

Componente	Specificații
Procesor	Intel Core i3, 2.4Ghz(7th Gen)
RAM	8 GB DDR4
Placa video	NVIDIA 940MX 2 GB
HDD	5400 RPM

Tabel 4.1 - Configurație laptop

După mai multe rulări pe o serie de videoclipuri s-au obținut performanțele din Tabelul 4.2. Algoritmi de urmărire au fost testați pe mai multe videoclipuri ce prezentau diverse scenarii din viața reală și cu filmări realizate din dronă pentru a avea un rezultat mult mai relevant asupra performanței algoritmului.

Precizia algoritmilor se poate determina folosind o serie de metode de evaluare ce iau în calcul diferite aspecte. Pentru lucrarea prezentă parametrul de interes este Object Detection Rate, rata cu care obiectul a fost detectat corect obiectul urmărire. Acesta se calculează raportat la eticheta de referință a obiectului, acesta rată variază între 0 și 1, iar pentru cazul în care acesta este apropiata de 0 se va face o detecție proastă, pe când o valoare apropiată de 1 înseamnă că obiectul pe care algoritmul nostru de urmărire îl detectează se apropie de forma inițială.[7]

În urmă studiului mai multor lucrări de specialitate care sunt despre algoritmii online de urmărire, am observat că în cazul algoritmi studiați există o rată de precizie aproximativ egală, cu mici excepții. Alegerea finală a fost redusă doar la studierea performanțelor pe care aceștia le pot obține în diferite situații.

Totuși, pentru a fi siguri că aceștia sunt stabili, am mai rulat algoritmii de urmărire și în cazul în care avem un zgomot de tip gaussian aplicat pe obiectul urmărit. Și în acest caz algoritmii au fost robuști, urmărirea fiind în continuare una de calitate.

Metoda	Rezoluție video	FPS
Boosting	1280x720(HD)	10
MIL	1280x720(HD)	8
KCF	1280x720(HD)	30
MOSSE	1280x720(HD)	40

Tabel 4.2 - Comparația numărului de FPS-uri pentru algoritmi

Privind performanțele pe care algoritmii le-au obținut în Tabelul 4.2 si luând în calcul afirmațiile anterioare am ajuns la concluzia că tehnica cea mai bună de folosit în cazul nostru implică utilizarea unui algoritm ce are la bază tehnica de tip MOSSE.

Explicațiile modului de funcționare al acesteia sunt prezentate în capitolul 2. Un punct slab al acestui algoritm apare în momentul în care un alt obiect se suprapune cu obiectul urmărit și asemănarea dintre cele două este una ridicată(ex.: culoarea, formă) există o mică posibilitatea, după ce a dispărut suprapunerea, ca algoritmul să încadreze obiectul greșit când cele două se despart. În practică este destul de greu să se întâmple așa ceva deoarece există o probabilitate scăzută ca două obiecte care sunt asemănătoare și din punct de vedere al culorii dar și al formei să fie suprapuse mai mult timp într-un cadru video.

4.2 OpenCV și wxPython

OpenCV (Open Source Computer Vision Library) este o bibliotecă la care are acces toată lumea și ține de domeniul Computer Vision, ce are în componență o mulțime de algoritmi. OpenCV a fost realizat cu gândul de a avea o platforma comună pentru aplicațiile ce țin de domeniul Computer Vision și pentru a accelera folosirea dispozitivelor ce pot reacționa la diverși stimuli vizuali.[8]

Biblioteca conține la momentul actual mai mult de 2500 de algoritmi optimizați, în care sunt incluși un set cuprinzător din cei mai recenți algoritmi din domeniul Computer Vision. Acești algoritmi pot fi folosiți pentru recunoaștere facială, identificarea de obiecte, clasificarea acțiunilor pe care le fac oamenii, camere ce urmăresc mișcarea, urmărirea de obiecte și multe alte aplicații.[8]

Are suport pentru limbajele de programare C++, Python, Java și Matlab, dar și următoarele sisteme de operare: Windows, Linux, Android și Mac OS.[8]

Cu ajutorul bibliotecii wxPython am realizat interfața cu care operatorul trebuie să interacționeze pentru a utiliza algoritmul creat. wxPython are la baza un set de instrumente de tip GUI pentru limbajul de programare Python. Acesta permite programatorilor în Python să creeze programe cu o interfață grafică robustă, ușor utilizabilă și cu foarte multe funcții. .[9]

A fost implementat ca un set de extensii pentru limbajul Python care este un wrapper peste biblioteca wxWidgets, foarte populară în limbajul de programare C++. La fel ca și OpenCV-ul și această aplicație este open source, iar fiecare utilizator poate să-și configureze și să îmbunătățească codul după bunul plac și în funcție de nevoile pe care le are.[9]

Biblioteca wxPython are un set de instrumente de tip cross-platform, asta înseamnă că același program creat într-un anumit sistem de operare va funcționa în și alte sistem de operare, fără să fie nevoie de modificări asupra programului.[9]

4.3 Implementarea si limbajul de programare

Algoritmii folosiți atât pentru testare, cât și cel folosit pentru partea practică sunt implementați în biblioteca de tip open source de la OpenCV, singura constrângere pe care o avem fiind versiunea de OpenCV pe care o folosim. Spre exemplu, CSRT acceptă o versiune de OpenCV mai nouă de 3.4, în timp ce restul algoritmilor funcționează cu OpenCV 3.2.

Implementarea a fost realizată în limbajul Python. Python este un limbaj de programare dinamic, de nivel înalt, ce pune accent pe expresivitatea și înțelegerea ușoară a codului. Sintaxa sa permite implementări echivalente cu alte limbaje în mai puține linii de cod. Datorită acestui fapt, Python este foarte răspândit atât în programarea de aplicații, cât și în zona de scripting.

Limbajul facilitează mai multe paradigme de programare, în special paradigmă imperativă (C) și pe cea orientată pe obiecte (Java). Spre deosebire de C, Python nu este un limbaj compilat, ci interpretat. Acest fapt are atât avantaje, cât și dezavantaje. Pe de-o parte, Python este mai lent decât C. Pe de altă parte, aplicațiile Python sunt foarte ușor de depanat, codul putând fi ușor inspectat în timpul rulării. De asemenea, este foarte ușor de experimentat cu mici fragmente de cod folosind interpretorul Python.

Limbajul interpretat, nu compilat presupune că programele sunt transformate într-un limbaj intermediar. Acest lucru permite codului să fie ușor de mutat pe diverse sisteme de operare și arhitecturi hardware. Codul este executat linie cu linie. Astfel, dacă - de exemplu - apelăm o funcție care nu există, vom primi un mesaj de eroare abia când se încearcă executarea liniei respective. Erorile de sintaxă sunt raportate însă înainte de rularea programului.

Vom începe prezentarea aplicației și a modului cum aceasta funcționează în subcapitolele următoare.

Prima oară vom vorbi despre interfața grafică cu care are de interacționează operatorul ce se află la sol pentru a identifica obiectul urmărit, o consecință a faptului că acești algoritmi de urmărire sunt online. Pentru acesta am implementat o interfață grafică cu ajutorul bibliotecii, wxPython, recunoscută pentru ușurința cu care se lucrează cu acesta.

4.4 Interfața aplicației

În cele ce urmează vom descrie funcționalitatea interfeței prezența în Figura 4.1.

Figură 4.1 - Interfața aplicației

După cum se poate observa, interfața este una simplă și intuitivă pentru operator, astfel încât să nu apară probleme în momentul în care și un utilizator neexperimentat este pus în situația de a folosi această aplicație.

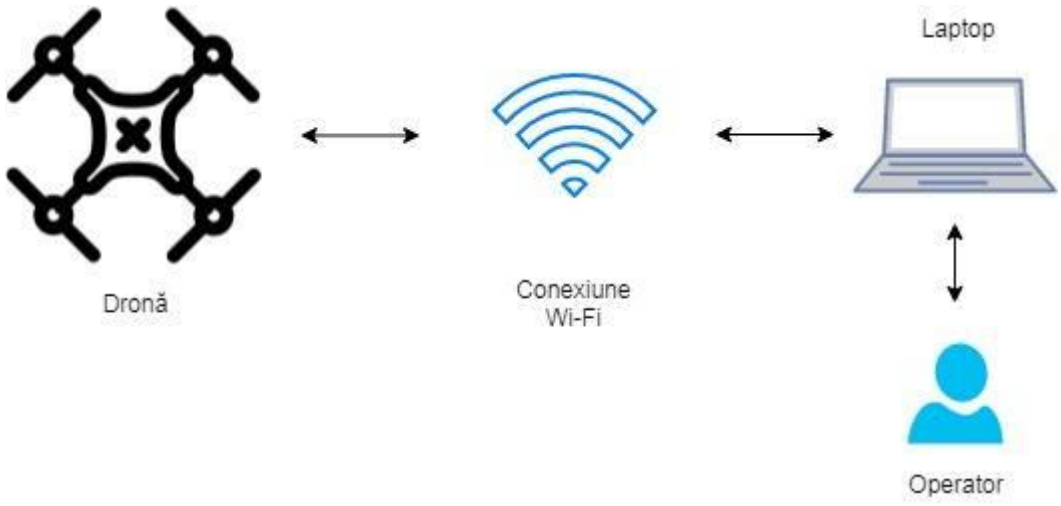
Avem un număr de 6 butoane, fiecare cu un scop bine definit. Vom începe prin a discuta funcționalitatea fiecăruia în cele ce urmează:

- butonul *Camera* – în momentul în care acesta este selectat, sursa video pe care rulează aplicația o să fie camera atașată dronei/echipamentului de testare;
- butonul *Videoclip* – în momentul în care acesta este selectat, sursa video pe care rulează aplicația o să fie un videoclip aflat la o adresă prestabilită;
- butonul *Start* – când este apăsat o să înceapă stream-ul/rularea secvențelor video;
- butonul *Select Object* – apăsarea acestui buton corespunde cu selectarea obiectului urmărit în secvența video;
- butonul *Stop* – oprește rularea aplicației.

4.5 Modul de funcționare al aplicației

Aplicația este rulată la nivelul dronei, dar luând în considerare faptul că ea nu este complet autonomă mai avem nevoie și de o stație pe care operatorul trebuie să facă selecția obiectului pe care dorim să-l urmărim și să se asigure că drona face urmărirea cum trebuie.

Mai jos avem o figură în care este prezentat modul de funcționare al aplicației:



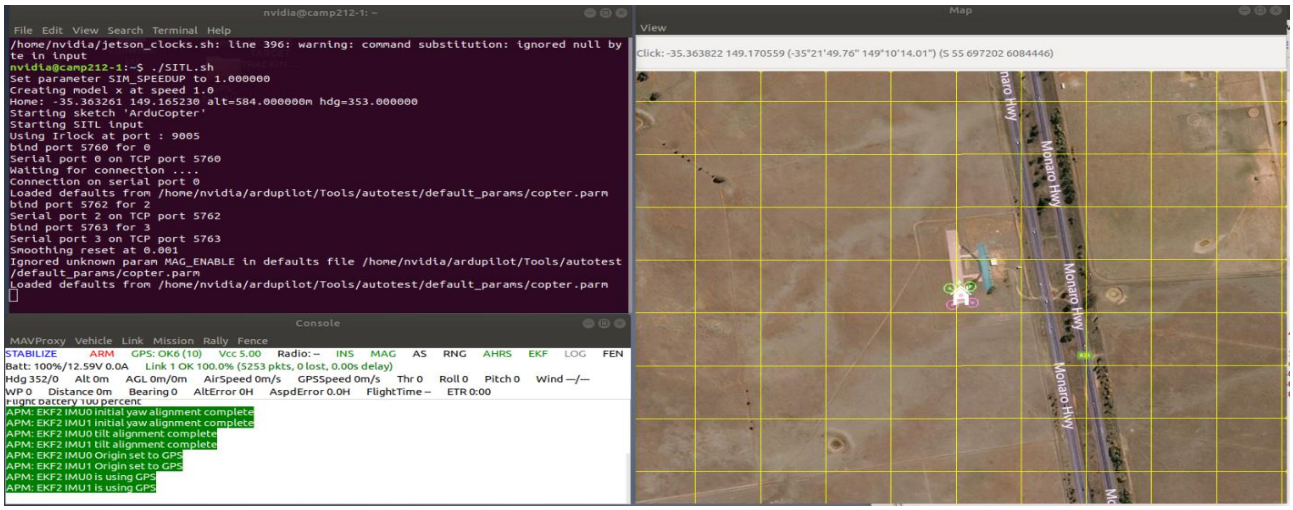
Figură 4.2 - Diagrama modului de funcționare

Important de menționat este faptul că aplicația rulează pe dronă și nu pe laptop, acesta fiind folosit doar pentru a accesa interfața acesteia și pentru a se putea face selecția obiectului. Acest lucru ne ajută deoarece în cazul în care este pierdută conexiunea între dronă și laptop, dar obiectul urmărit este încă localizat cu succes, drona poate rula într-un mod autonom până când se reface conexiunea sau până când obiectul a fost pierdut de sub supravegherea acesteia.

4.6 Testarea aplicației

Deoarece manevrarea dronelor poate fi periculoasă, mai ales a celor de dimensiuni ridicate, iar de cele mai multe ori un cod nu este funcțional de la prima rulare, se preferă testarea lui cu ajutorul unor aplicații specializate ce pot imita și răspunde în concordanță cu hardware-ul folosit.

În cazul nostru s-a preferat folosirea unui simulator de tip SITL(Software în the Loop) ce oferă acces la informații esențiale, după cum se poate observa și în Figura 4.3.

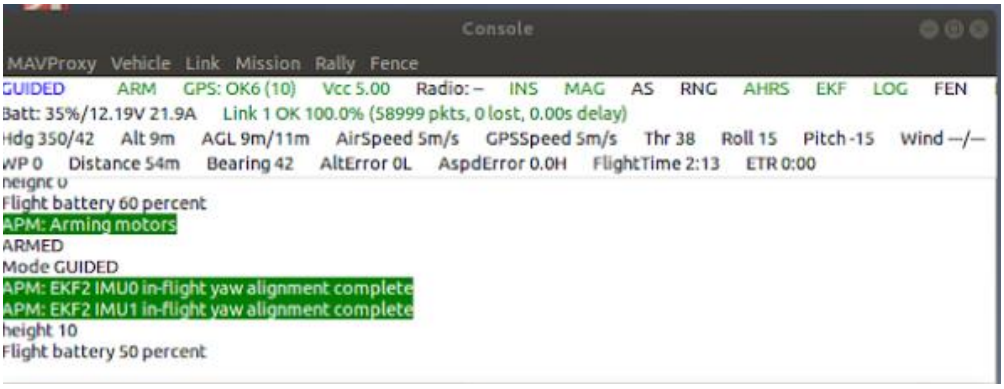


Figură 4.3 - Simulatorul SITL

Simulatorul folosit este unul ce poate reproduce cu fidelitate modul în care este controlată drona de aplicație și ne oferă detalii despre informațiile de interes. În Figura 4.4 putem observa drona urmând o direcție de zbor pe harta ce aparține simulatorului, important este faptul că în simulator putem vedea direcția pe care drona o s-o urmeze și putem vedea dacă aplicația funcționează conform așteptărilor sau au apărut erori.

Figură 4.4 - Harta simulatorului

Tot acest simulator are în componență și o consola prezentată în Figura 4.5 în care sunt afișate o serie întreagă de informații despre dronă. Acestea au o importanță foarte mare deoarece sunt date ce indică bună funcționare sau proastă funcționare a aparatului de zbor în timpul testelor. Dintre parametrii prezentați se pot remarca următorii: altitudinea, viteza în timpul zborului și distanța parcursă.



Figură 4.5 - Consola simulatorului

Testarea aplicației a fost realizat în laborator pe sistemul prezentat în capitolul 1. Un mare avantaj a fost dat de faptul că atât pentru testare cât și într-un scenariu real se folosește același procesor de la Nvidia, JetsonTX2. Acest procesor încorporează 2 procesoare de tip ARM, un procesor cu 4 nuclee, la o frecvența de 2.0 *Ghz* pe 64 biți, și un procesor superscalar cu 2 nuclee la o frecvența de tot 2.0 *Ghz*.[10]

Există și o mulțime de motive pentru care a fost folosit un procesor de tip ARM. Arhitectura ARM (Advanced RISC Machine) inițial cunoscută și sub numele de Acorn RISC Machine, este o arhitectură ce face parte din categoria de microprocesoare de tip RISC (Reduced Instruction Set Computer).[11]

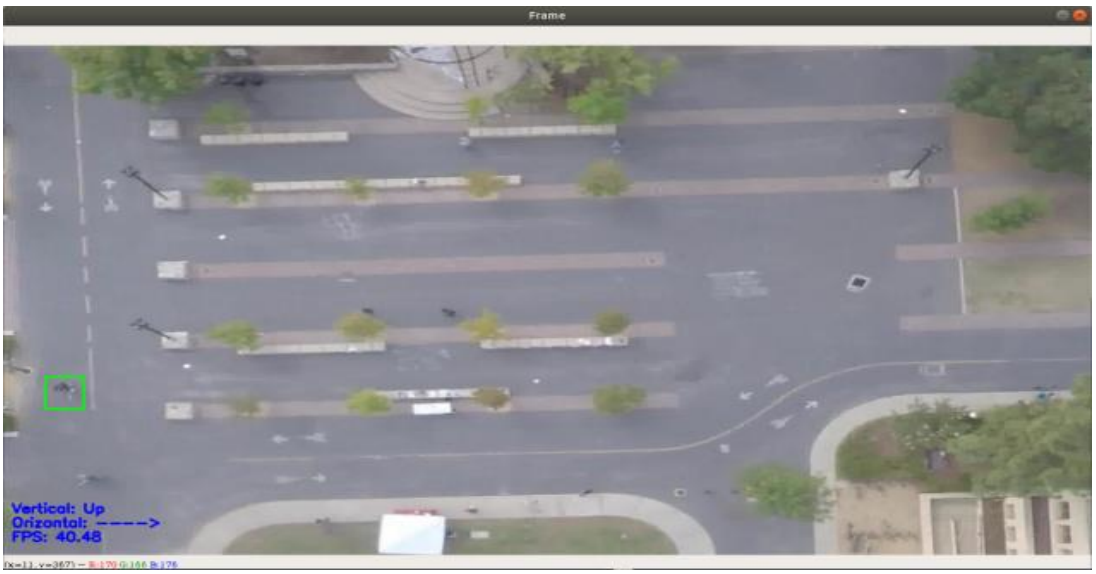
Diferența a fost observată în momentul când aplicația a fost rulată pe hardware-ul folosit pentru testare. Față de rezultatele obținute pe laptop-ul ce avea un procesor de uz general de la Intel și scotea un număr aproximativ de 40 de cadre pe secundă în timp ce făcea urmărirea pentru un singur obiect, pe hardware-ul de testare reușim să obținem undeva în jurul a 70-80 de cadre pe secundă când algoritmul realizează urmărirea unui obiect. Așadar, avem o performanță aproximativă de două ori mai bună față de cazul în care a fost rulat pe laptop.

Ne interesează foarte mult acest lucru deoarece aplicația noastră are ca scop urmărirea în timp real pentru orice obiect ce este selectat inițial de utilizator.

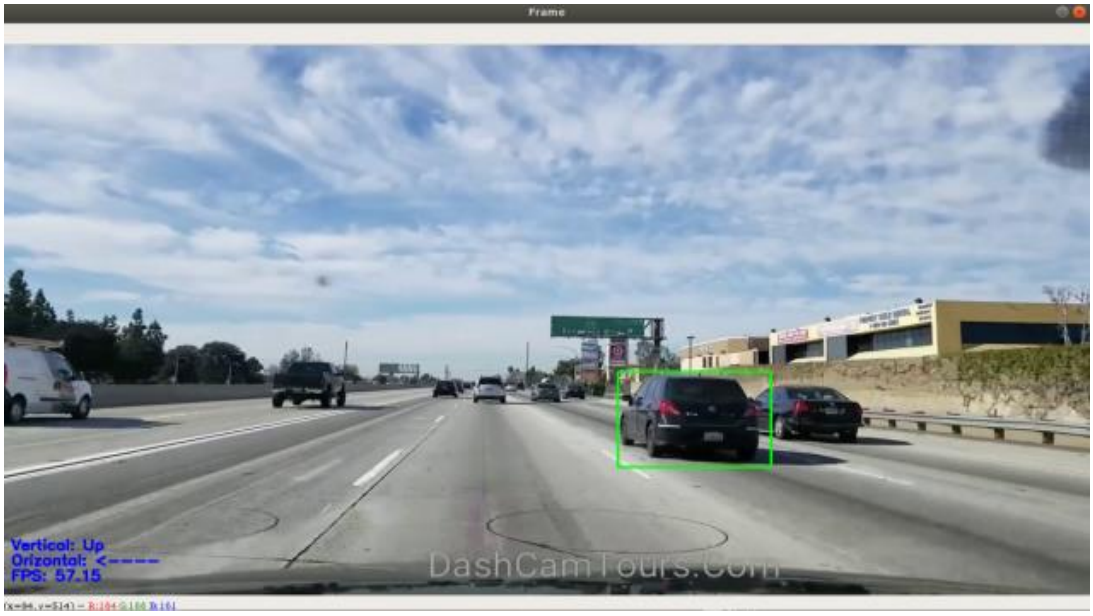
Un alt lucru foarte important pentru aplicația noastră pe care nu l-am amintit este faptul că poate funcționa indiferent de obiectul selectat de utilizator la începutul rulării acesteia. Comparația în acest caz se poate face cu algoritmii de ML care în anumite cazuri pot scoate performanțe mai bune, dar având în vedere că este mult mai dificil de lucrat cu aceștia deoarece au nevoie și de o antrenare inițială pentru fiecare tip de obiect pe care dorim să-l urmărim am ales să nu lucrăm cu aceștia.

Antrenarea este costisitoare deoarece necesită o baza de date cu mai multe instanțe din obiectul pe care dorim să-l urmărim, la care se adaugă timpul ce trebuie așteptat pentru a realiza această antrenare. Ca o soluție de mijloc ar putea să se utilizeze o aplicație ce utilizează cele două metode, iar pentru obiectele pe care le considerăm comune să se facă o urmărire ce are la baza algoritmi din categoria ML, iar pentru obiectele pentru care nu au fost antrenați încă algoritmii de ML să se utilizeze metodele bazate pe algoritmii prezentați în această lucrare.

Pentru a demonstra că nu avem nevoie de o antrenare înainte selecției, am atașat capturi de ecran din momentul rulării aplicației în care au fost selectate diverse obiecte pentru a se realiza urmărirea.



Figură 4.6 – Urmărire pieton

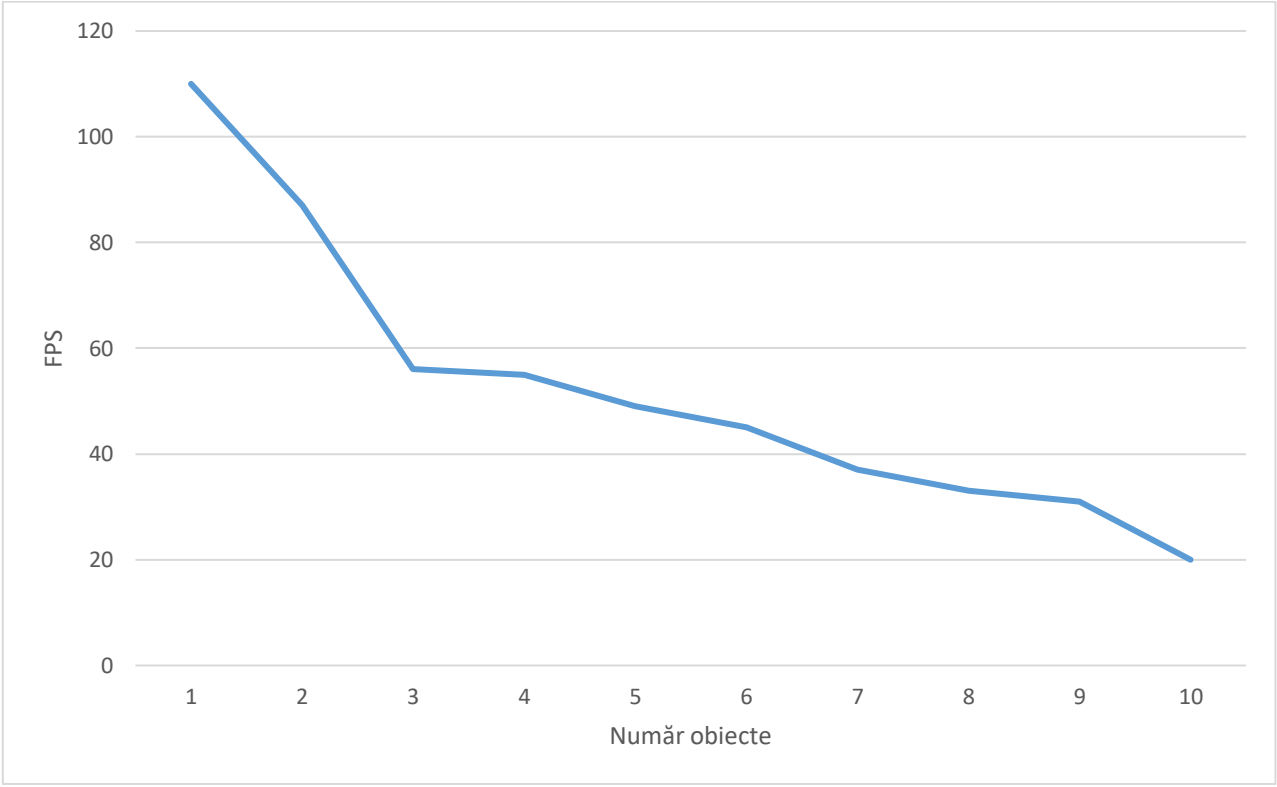


Figură 4.7 - Urmărire autoturism

Figură 4.8 - Urmărire persoană

4.7 Urmărirea multi-obiect

În încheierea experimentelor realizate s-au efectuate și câteva teste pentru a observa modul în care urmărirea multi-obiect afectează performanțele totale ale sistemului. A fost pus un accent ridicat ca și în testările anterioare pe numărul de cadre pe secundă pe care sistemul l-a scos când în aplicație au fost selectate mai multe obiecte pentru urmărire.



Figură 4.9 – Număr cadre pe secunda / Număr obiecte

Făcând o analiză graficului putem observa după cum era și de așteptat că odată cu creșterea numărului de obiecte pe care dorim să le urmărim scade și performanța sistemului nostru. Testarea aceasta a fost realizată pe procesorul Jetson TX2 cu algoritmul bazat pe tehnica de tip MOSSE.

Concluzii

Lucrarea de față urmărește crearea unei aplicații intuitivă, dar si stabilă, pentru urmărirea în timp real a obiectelor, cu ajutorul unei drone ce are atașată o camera ce poate capta secvențe video.

Scopul lucrării este de a realiza o urmărire cât mai exactă indiferent de forma obiectului selectat inițial, cu ajutorul unui UAV. Așadar, în mod ideal, un algoritm de urmărire ce primește o singură dată informațiile despre forma obiectului trebuie să fie și suficient de rapid pentru a identificat obiectul în restul cadrelor din secvența video captată cu ajutorul dronei, chiar dacă acesta își modifică substanțial locația de la un cadru la altul. Trebuie ținut cont și de faptul când în secvență video apare ocluzia obiectului urmărit, iar algoritmul trebuie să fie capabil de reidentificarea acestuia.

În încercarea de a realiza lucrarea au fost consultate o serie întreagă de surse ale literaturii de specialitate în care se prezentau avantaje și dezavantaje ale tehnicilor actuale de urmărire de obiect. Alegerea a fost făcută după ce s-au analizat și cele două categorii importate cu care se efectuează urmărirea de obiect în ziua de azi. Cele două categorii comparate au fost ale algoritmilor online de urmărire și a algoritmilor ce au la baza tehnici de ML. Principalul avantaj al algoritmilor online de urmărire, și motivul pentru care aceștia au fost aleși în lucrarea de față în detrimentul celor ce folosesc algoritmi de tip ML, este dat de faptul că aceștia pot face urmărirea indiferent de obiectul selectat și nu pentru un set redus de obiecte pentru care ar fi nevoie și de o antrenare în prealabil pentru obiectul pe care dorim să-l urmărim.

După ce am stabilit categoria algoritmilor pe care să-i folosim a urmat o etapă în care s-au studiat diverse metode disponibile în cadrul algoritmilor online de urmărire. Aceștia au fost testați în mai multe condiții de interes pentru lucrarea noastră, cu videoclipuri filmate cu ajutorul dronei pe care a fost implementată ulterior și aplicația.

Un alt aspect de care am ținut cont a fost și hardware-ul disponibil pe UAV, deoarece s-au testat și diferite metode prin care a fost comandată mișcarea dronei în încercarea acesteia de a urmări obiectul de interes cu o precizie cât mai ridicată.

Testele efectuate după implementările realizate au fost rulate pe placa de dezvoltare de la NVIDIA pentru a respecta măsurile de siguranță și nu a produce daune echipamentelor finale pe care trebuie folosită aplicația.

Contribuțiile personale pentru acest proiect au fost:

- realizarea interfeței pentru utilizarea aplicației;
- implementarea a doua modalități diferite de mișcare ale dronei în încercarea de a realiza o urmărire corectă;
- studiul performanței algoritmilor in diferite situații.

Utilitatea acestor sisteme este indiscutabilă, dar trebuie avut în vedere atingerea unor parametrii calitativi care să permită folosirea lor în diferite medii. Astfel, trebuie considerate toate aspectele ce ar putea duce la îmbunătățirea performanțelor și calității.

Pentru proiectul de față pot fi luate în considerare anumite direcții pentru perfecționarea acestuia, folosirea mai multor algoritmi de urmărire concomitent pentru a realiza o urmărirea corectă în momentul în care unul dintre algoritmi pierde ținta este una din modalitățile prin care se poate îmbunătății acest proiect.

Toate îmbunătățirile care se poate aduce acestui sistem trebuie să vizeze scopul final ce presupune folosirea aplicației pe un vehicul tip dronă și urmărirea de obiect.

Complexitatea și posibilitatea dezvoltării ulterioare a proiectului au constituit principala motivație a alegerii temei. Această lucrare reprezintă doar un prim pas în ansamblul final al proiectului, aprofundarea fiind necesară pentru dezvoltările ulterioare.

Bibliografie

[1] -, „Harness AI at the Edge with the Jetson TX2 Developer Kit”, disponibil online la adresa https://developer.nvidia.com/embedded/jetson-tx2-developer-kit, accesat la data de 15.06.2019

[2] David S. Bolme J. Ross Beveridge Bruce A. Draper Yui Man Lui, „Visual Object Tracking using Adaptive Correlation Filters”, Computer Science Department, Colorado State University, Fort Collins, CO 80521, USA

[3] Helmut Grabner, Michael Grabner, Horst Bischof, „Real-Time Tracking via On-line Boosting”, Institute for Computer Graphics and Vision, Graz University of Technology

[4] Boris Babenko Ming-Hsuan Yang Serge Belongie, „Visual Tracking with Online Multiple Instance Learning”, 2009 IEEE Conference on Computer Vision and Pattern Recognition

[5] Fengfa Yue and Xingfei Li, „Improved kernelized correlation filteralgorithm and application in theoptoelectronic tracking syst”, International Journal of AdvancedRobotic System, 2010, DOI:10.1177/1729881418776582

[6] -, „MavLink Tutorial for Absolute Dummies (Part –I)”, disponibil online la adresa https://api.ning.com/files/i*tFWQTF2R*7Mmw7hksAU-u9IABKNDO9apguOiSOCfvi2znk1tXhur0Bt00jTOldFvob-Sczg3*lDcgChG26QaHZpzEcISM5/MAVLINK_FOR_DUMMIESPart1_v.1.1.pdf accesat la data de 15.06.2019

[7] J. Popoola and A. Amer, "Performance evaluation for tracking algorithms using object labels," 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, Las Vegas, NV, 2008, pp. 733-736. doi: 10.1109/ICASSP.2008.4517714

[8] -, „About”, disponibil online la adresa https://opencv.org/about/, accesat la data de 15.06.2019

[9] -, „Overview of wxPython”, disponibil online la adresa https://wxpython.org/pages/overview/index.html, accesat la data de 15.06.2019

[10] Tanya Amert, Nathan Otterness, Ming Yang, James H. Anderson, and F. Donelson Smith, „GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed”, Department of Computer Science, University of North Carolina at Chapel Hill

[11] -, „Arhitectură ARM”, disponibil online la adresa https://ro.wikipedia.org/wiki/Arhitectur%C4%83_ARM, accesat la data de 15.06.2019

[12] Mr.instr.şef Nicolae MORO, „Sistemul de poziţionare global – avantajul tehnologiei în lucrările topogeodezi”

Anexa 1 Cod

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 13 16:37:12 2019

@author: marian
"""

# biblioteci necesare
from imutils.video import VideoStream
import time
import cv2
import datetime
from dronekit import connect, VehicleMode
from pymavlink import mavutil
import wx

key_s = False
key_q = False

class app_gui(wx.Frame):

    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, "Licenta",
size=(400,400))
        panel = wx.Panel(self)
        button_1 = wx.Button(panel, label = "Start", pos =
(150,110), size=(100,50))
        button_2 = wx.Button(panel, label = "Select object", pos =
(150,180), size=(100,50))
        button_3 = wx.Button(panel, label = "Stop", pos =
(150,250), size=(100,50))
        button_4 = wx.Button(panel, label = "Camera", pos =
(95,30), size=(100,50))
        button_5 = wx.Button(panel, label = "Videoclip", pos =
(205,30), size=(100,50))
        wx.StaticText(panel, label="Selectati sursa",
pos=(150,5),size=(100,20), style=0)
        self.Bind(wx.EVT_BUTTON, self.app_start_btn, button_1)
        self.Bind(wx.EVT_BUTTON, self.select_btn, button_2)
        self.Bind(wx.EVT_BUTTON, self.quit_btn, button_3)
        self.Bind(wx.EVT_BUTTON, self.source_true, button_4)
        self.Bind(wx.EVT_BUTTON, self.source_false, button_5)
```

```
        self.Bind(wx.EVT_CLOSE, self.closewindow)

    def app_start_btn(self, event):
        app_start()

    def select_btn(self, event):
        global key_s
        key_s = True

    def quit_btn(self, event):
        global key_q
        key_q = True

    def source_false(self, event):
        global source
        video_source(False)
        source = False

    def source_true(self, event):
        global source
        video_source(True)
        source = True

    def closewindow(self, event):
        self.Destroy()

class FPS:
    # contructor implicit pentru clasa FPS
    def __init__(self):
        self.timp_start = None
        self.timp_final = None
        self.numar_cadre = 0

    # functie pentru pornirea timer-ului
    def start(self):
        self.timp_start = datetime.datetime.now()
        return self

    # functie pentru oprirea timer-ului
    def stop(self):
        self.timp_final = datetime.datetime.now()

    # incrementarea numarului de cadre
    def update(self):
        self.numar_cadre += 1
```

```
    # diferenta intre timul de final ssss cel de start
    def elapsed(self):
        return (self.timp_final - self.timp_start).total_seconds()

    # calculeaza numarul de cadre aproximativ
    def fps(self):
        return self.numar_cadre / self.elapsed()


# specificarea
connection_string = ('tcp:127.0.0.1:5762')

# Conecatarea la drona
print('Conectarea cu drona se face pe adresa: %s' %
connection_string)
vehicle = connect(connection_string, wait_ready=True)
gnd_speed = 5 # [m/s]

# functie pentru armarea dronei si ridicarea acesteaia la o
anumita inaltime
def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print("Basic pre-arm checks")
    # Don't try to arm until autopilot is ready
    while not vehicle.is_armable:
        print(" Waiting for vehicle to initialise...")
        time.sleep(1)

    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    # Confirm vehicle armed before attempting to take off
    while not vehicle.armed:
        print(" Waiting for arming...")
        time.sleep(1)

    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)  # Take off to target
altitude
```

```
    # Wait until the vehicle reaches a safe height before
processing the goto
    #  (otherwise the command after Vehicle.simple_takeoff will
execute
    #   immediately).
    while True:
        print(" Altitude: ",
vehicle.location.global_relative_frame.alt)
        # Break and return from function just below target
altitude.
        if vehicle.location.global_relative_frame.alt >=
aTargetAltitude * 0.95:
            print("Reached target altitude")
            break
        time.sleep(1)


def set_velocity_body(vehicle, vx, vy, vz):
    """ Remember: vz is positive downward!!!
    http://ardupilot.org/dev/docs/copter-commands-in-guided-
mode.html

    Bitmask to indicate which dimensions should be ignored by the
vehicle
    (a value of 0b0000000000000000 or 0b0000001000000000 indicates
that
    none of the setpoint dimensions should be ignored). Mapping:
    bit 1: x,  bit 2: y,  bit 3: z,
    bit 4: vx, bit 5: vy, bit 6: vz,
    bit 7: ax, bit 8: ay, bit 9:


    """
    msg =
vehicle.message_factory.set_position_target_local_ned_encode(
            0,
            0, 0,
            mavutil.mavlink.MAV_FRAME_BODY_NED,
            0b0000111111000111, #-- BITMASK -> Consider only the
velocities
            0, 0, 0,        #-- POSITION
            vx, vy, vz,     #-- VELOCITY
            0, 0, 0,        #-- ACCELERATIONS
            0, 0)
    vehicle.send_mavlink(msg)
    vehicle.flush()
```

```
#-- Key event function
def actual_key(event):
    if event.char == event.keysym: #-- standard keys
        if event.keysym == 'r':
            print("r pressed >> Set the vehicle to RTL")
            vehicle.mode = VehicleMode("RTL")

    else: #-- non standard keys
        if event.keysym == 'Up':
            set_velocity_body(vehicle, gnd_speed, 0, 0)
        elif event.keysym == 'Down':
            set_velocity_body(vehicle,-gnd_speed, 0, 0)
        elif event.keysym == 'Left':
            set_velocity_body(vehicle, 0, -gnd_speed, 0)
        elif event.keysym == 'Right':
            set_velocity_body(vehicle, 0, gnd_speed, 0)

arm_and_takeoff(10)

print("Setarea vitezei dorite!()")
vehicle.airspeed = 3

def get_location_metres(original_location, dNorth, dEast):
    """
    Returns a LocationGlobal object containing the
latitude/longitude `dNorth` and `dEast` metres from the
    specified `original_location`. The returned LocationGlobal has
the same `alt` value
    as `original_location`.

    The function is useful when you want to move the vehicle
around specifying locations relative to
    the current vehicle position.

    The algorithm is relatively accurate over small distances (10m
within 1km) except close to the poles.

    For more information see:
    http://gis.stackexchange.com/questions/2951/algorithm-for-
offsetting-a-latitude-longitude-by-some-amount-of-meters
    """
    earth_radius = 6378137.0 #Radius of "spherical" earth
    #Coordinate offsets in radians
    dLat = dNorth/earth_radius
```

```
    dLon =
dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

    #New position in decimal degrees
    newlat = original_location.lat + (dLat * 180/math.pi)
    newlon = original_location.lon + (dLon * 180/math.pi)
    if type(original_location) is LocationGlobal:
        targetlocation=LocationGlobal(newlat,
newlon,original_location.alt)
    elif type(original_location) is LocationGlobalRelative:
        targetlocation=LocationGlobalRelative(newlat,
newlon,original_location.alt)
    else:
        raise Exception("Invalid Location object passed")

    return targetlocation;

def get_distance_metres(aLocation1, aLocation2):
    """
    Returns the ground distance in metres between two
LocationGlobal objects.

    This method is an approximation, and will not be accurate over
large distances and close to the
    earth's poles. It comes from the ArduPilot test code:

https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/
common.py
    """
    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5


def goto_position_target_local_ned(north, east, down, heading):
    """
    Send SET_POSITION_TARGET_LOCAL_NED command to request the
vehicle fly to a specified
    location in the North, East, Down frame.
    """
#    currentLocation = vehicle.location.global_relative_frame
#    targetLocation = get_location_metres(currentLocation, north,
east)
#    targetDistance = get_distance_metres(currentLocation,
targetLocation)
#
```

```
    msg =
vehicle.message_factory.set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_BODY_NED, # frame
        0b0000000111111000, # type_mask (only positions enabled)
        north, east, down,
        0, 0, 0, # x, y, z velocity in m/s  (not used)
        0, 0, 0, # x, y, z acceleration (not supported yet,
ignored in GCS_Mavlink)
        heading, 0)    # yaw, yaw_rate (not supported yet, ignored
in GCS_Mavlink)
    # send command to vehicle
    vehicle.send_mavlink(msg)
#    DURATION = targetDistance #Set duration for each segment.
#    time.sleep(DURATION)
#    while vehicle.mode.name=="GUIDED":
#        #Stop action if we are no longer in guided mode.
#        #print "DEBUG: mode: %s" % vehicle.mode.name
#        remainingDistance =
get_distance_metres(vehicle.location.global_relative_frame,
targetLocation)
#        print("Distance to target: ", remainingDistance)
#        if remainingDistance <= 2.0: #targetDistance*0.01:
#            #Just below target, in case of undershoot.
#            print("Reached target")
#            break;
#        time.sleep(1)

# $$$ variabila folosita pentru a selecta sursa
vs = None
source = None

def video_source(source):
    global vs
    # $$$ daca source=True, pornim camera
    # citrea este realizata cu biblioteca imutils pentru o mai
buna rata de procesare(FPS)
    if source:
        print("Se porneste streamul video!")
        vs = VideoStream(src=1).start()
        time.sleep(1.0)

    # daca setam source = False, utilizam un video salvat local
    else:
        vs = cv2.VideoCapture("/home/nvidia/DJI_0017.MP4")
```

```
        # ne asiguram ca videoclipul a fost pornit
        if(vs.isOpened() == False):
            print("Videoclipul nu poate fi pornit!")

trackers = cv2.MultiTracker_create()
def app_start():
    global key_s
    global key_q
    # intializarea obiectului de tip tracker pentru urmarirea mai
multor obiecte
    #trackers = cv2.MultiTracker_create()
    fps = None

    # pornim timerul pentru FPSs
    fps = FPS().start()

    # $$$ contor pentru numarul de cadre
    frame_number = 0

    #dimensiune streamului video
    width = 1280
    height = 720
    frame_center = [1280 / 2, 720 / 2]
    vertical = None
    orizontal = None
    direction = None

    # bucla peste frameurile din stream
    while True:
        # $$$ afisarea numarului de frameuri in consola
        #print("Nr. frame: ", frame_number)

        # comuta in functie de frameul curent pe ce sursa ne
aflam, VideoStream sau VideoCapture
        frame = vs.read()
        frame = frame[1] if not source else frame

     # verificare pana la ultimul frame
        if frame is None:
            break

     # redimensionare frame, cautarea unui optim pentru calitatea
camerei utilizate
        frame = cv2.resize(frame, (width, height))
        # dimensions of frame
        (H, W) = frame.shape[:2]
```

```
    # grab the updated bounding box coordinates (if any) for each
    # object that is being tracked
       (success, boxes) = trackers.update(frame)

    # bucla peste toate chenarele si desenarea acestora
       for box in boxes:
           (x, y, w, h) = [int(v) for v in box]
           cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255,
0), 2)
           if ((2*box[0] + box[2])/2 < frame_center[0]):
               #print("Left")
               orizontal = 0
           else:
               #print("Right")
               orizontal = 1
           if ((2*box[1] + box[3])/2 < frame_center[1]):
               #print("Down")
               vertical = 0
           else:
               vertical = 1
               #print("Up")

           if ((2*box[0] + box[2])/2 < frame_center[0] - 160 and
(2*box[1] + box[3])/2 < frame_center[1] - 60 ):
               direction = 0
           if ((2*box[0] + box[2])/2 > frame_center[0] - 160 and
(2*box[1] + box[3])/2 < frame_center[1] - 60 and (2*box[0] +
box[2])/2 < frame_center[0] + 160 ):
               direction = 1
           if ((2*box[0] + box[2])/2 > frame_center[0] + 160 and
(2*box[1] + box[3])/2 < frame_center[1] - 60):
               direction = 2
           if ((2*box[0] + box[2])/2 < frame_center[0] - 160 and
(2*box[1] + box[3])/2 > frame_center[1] - 60 and (2*box[1] +
box[3])/2 < frame_center[1] + 60 ):
               direction = 3
           if ((2*box[0] + box[2])/2 > frame_center[0] - 160 and
(2*box[1] + box[3])/2 > frame_center[1] - 60 and (2*box[1] +
box[3])/2 < frame_center[1] + 60 and (2*box[0] + box[2])/2 <
frame_center[0] + 160):
               direction = 4
           if ((2*box[0] + box[2])/2 > frame_center[0] + 160 and
(2*box[1] + box[3])/2 > frame_center[1] - 60 and (2*box[1] +
box[3])/2 < frame_center[1] + 60 ):
```

```
                direction = 5
            if ((2*box[0] + box[2])/2 < frame_center[0] - 160 and
(2*box[1] + box[3])/2 > frame_center[1] + 60 ):
                direction = 6
            if ((2*box[0] + box[2])/2 > frame_center[0] - 160 and
(2*box[1] + box[3])/2 > frame_center[1] + 60 and (2*box[0] +
box[2])/2 < frame_center[0] + 160):
                direction = 7
            if ((2*box[0] + box[2])/2 > frame_center[0] + 160 and
(2*box[1] + box[3])/2 > frame_center[1] + 60 ):
                direction = 8

        # update the FPS counter
        fps.update()
        fps.stop()

        # informatia afisata pe video
        info = [
                ("FPS", "{:.2f}".format(fps.fps())),
                ("axa - X", "<----" if orizontal else "---->"),
                ("axa - Y", "Up" if vertical else "Down"),


        # bucla peste enumerate  q
        for (i, (k, v)) in enumerate(info):
            text = "{}: {}".format(k, v)
            cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)


     # afisare frame dupa prelucrare
        cv2.imshow("Frame", frame)
        # daca dorim sa schimbam numarul de FPS uris
        key = cv2.waitKey(1) & 0xFF

        if key == ord("r"):
                print("r(renunta) >> Seteaza mod -> RTL")
                time.sleep(0.5)
                vehicle.mode = VehicleMode("RTL")
        else:
            if direction == 0:
                    set_velocity_body(vehicle, gnd_speed, -
gnd_speed, 0)
            if direction == 1:
                    set_velocity_body(vehicle, gnd_speed, 0, 0)
            if direction == 2:
```

```
                    set_velocity_body(vehicle, gnd_speed,
gnd_speed, 0)
            if direction == 3:
                    set_velocity_body(vehicle, 0, -gnd_speed, 0)
            if direction == 4:
                    set_velocity_body(vehicle, 0, 0, 0)
            if direction == 5:
                    set_velocity_body(vehicle, 0, gnd_speed, 0)
            if direction == 6:
                    set_velocity_body(vehicle, -gnd_speed, -
gnd_speed, 0)
            if direction == 7:
                    set_velocity_body(vehicle, -gnd_speed, 0, 0)
            if direction == 8:
                    set_velocity_body(vehicle, -gnd_speed,
gnd_speed, 0)

        if key == ord("g"):
            print("g(reincepe) >> Seteaza mod -> GUIDED")
            time.sleep(0.5)
            vehicle.mode = VehicleMode("GUIDED")

        if key_s == True:
            #trackers = cv2.MultiTracker_create()
          # selecteaza obiectul, dupa care se apsa "ENTER" sau
"SPACE"
            box = cv2.selectROI("Frame", frame, fromCenter=False,
showCrosshair=True)
            # $$$ afiseaza coordonatele pentru selectie
            print("Coordonate chenar: ", box)
          # se adauga noul obiect care o sa fie urmaritr(MOSSE
tracker)
            trackers.add(cv2.TrackerMOSSE_create(), frame, box)
            # pentru reactualizarea timpului, implicit si numarul
de cadre
            fps = FPS().start()
            key_s = False

        # daca tasta "s" este apasata, se va selecta astepta
selectarea obiectului
        if key == ord("s"):
            #trackers = cv2.MultiTracker_create()
          # selecteaza obiectul, dupa care se apsa "ENTER" sau
"SPACE"
            box = cv2.selectROI("Frame", frame, fromCenter=False,
showCrosshair=True)
```

```
            # $$$ afiseaza coordonatele pentru selectie
            print("Coordonate chenar: ", box)
          # se adauga noul obiect care o sa fie urmaritr(MOSSE
tracker)
            trackers.add(cv2.TrackerMOSSE_create(), frame, box)
            # pentru reactualizarea timpului, implicit si numarul
de cadre
            fps = FPS().start()
        # daca "q" este apasata, termina programul
        elif key == ord("q") or key_q == True:
            print("q(quit) selectat >> Seteaza mod -> RTL")
            vehicle.mode = VehicleMode("RTL")
            time.sleep(1)
            key_q == False
            vehicle.close()
            cv2.destroyAllWindows()
            break

        # $$$ incrementare contor
        frame_number += 1

if __name__=='__main__':
    app = wx.App()
    frame = app_gui(None, id = -1)
    frame.Show()
    app.MainLoop()

# daca se foloseste camera web, se va opri transmisia
if source:
     vs.stop()

# altfel, elibereaza calea catre fisier
else:
     vs.release()

# inchiderea ferestrelor
vehicle.close()
cv2.destroyAllWindows()
```