

 **Facultatea de Automatică și Calculatoare**

**Programul de Master: SIAPS**

Aplicație software de recunoaștere a stărilor emoționale PRIN INTERPRETAREA trăsăturilor faciale

**Teză de disertație**

Florentina-Mihaela BLIDARIU

Conducători științifici:

Ș.l.dr.ing. Loredana STANCIU (Universitatea Politehnica Timişoara)

**Timișoara**

**2016**

**Cuprins**

[**1. Introducere** 5](#_Toc453542978)

[***1.1Definirea problemei și motivarea alegerii temei*** 5](#_Toc453542979)

[***1.2Scopul final al proiectului*** 6](#_Toc453542980)

[***1.3Prezentarea pe scurt a capitolelor existente în lucrare*** 6](#_Toc453542981)

[**2. Fundamentare teoretică** 8](#_Toc453542982)

[***2.1 Scurtă prezentare a mecanismelor folosite în implementare*** 8](#_Toc453542983)

[***2.2 Domenii studiate*** 9](#_Toc453542984)

[***2.3 Biblioteca OpenCV*** 13](#_Toc453542985)

[***2.3.1 Modulele din cadrul OpenCV*** 14](#_Toc453542986)

[***2.3.2 Inspecție optică, segmentare și detectare*** 16](#_Toc453542987)

[***2.3.3 Clasificarea obiectelor*** 19](#_Toc453542988)

[***2.3.4 Haarcascade training*** 21](#_Toc453542989)

[***2.4 Limbajul Python*** 26](#_Toc453542990)

[***2.4.1 Structuri de control în Python*** 26](#_Toc453542991)

[***2.5 Limbajul XML*** 29](#_Toc453542992)

[**3. Specificațiile aplicației** 32](#_Toc453542993)

[***3.1 Schema bloc a sistemului. Scurtă descriere a aplicației*** 32](#_Toc453542994)

[***3.2 Funcțiile sistemului*** 36](#_Toc453542995)

[***3.3 Interfața cu utilizatorul*** 38](#_Toc453542996)

[***3.4 Structuri de date și fișiere*** 39](#_Toc453542997)

[***3.4.1 Fișiere*** 39](#_Toc453542998)

[**4. Proiectarea de detaliu** 42](#_Toc453542999)

[***4.1 Arhitectura programului*** 42](#_Toc453543000)

[***4.2 Descrierea componentelor*** 42](#_Toc453543001)

[**5. Utilizarea sistemului** 44](#_Toc453543002)

[***5.1 Scurtă prezentare*** 45](#_Toc453543003)

[***5.2 Instalare*** 45](#_Toc453543004)

[***5.3 Utilizare*** 45](#_Toc453543005)

[***6. Realizarea, punerea în funcțiune și rezultate experimentale*** 49](#_Toc453543006)

[**7. Concluzii** 51](#_Toc453543007)

[**8.Bibliografie** 52](#_Toc453543008)

# **1. Introducere**

## ***1.1Definirea problemei și motivarea alegerii temei***

Prezenta lucrare reprezintă o aplicație software de recunoaștere a stărilor emoționale pe bazat răsăturilor faciale, fiind destinată utilizatorilor Windows. Această aplicație are capacitatea de a recunoaște stările emoționale în urma unor capturi efectuate de către utilizator( camera web). În funcție de trăsăturile faciale ale persoanei din poză, aplicația poate descrie starea emoțională a respectivei persoane.

În ultima perioadă, s-a căutat găsirea unor soluții prin care omul să interacționeze cu calculatoarele, iar calculatoarele la rândul lor să interacționeze cu oamenii. Domeniul în cadrul căruia aceste lucruri sunt posibile se numește *affective computing*. Acest domeniu presupune recunoașterea, interpretarea, procesarea și simularea emoțiilor umane. De la descoperirea acestui domeniu, s-au realizat mai multe aplicații prin care sistemul sa poată interacționa cu omul. Ducerea la bun sfârșit a acestor aplicații a fost posibilă datorită omului, dar folosindu-se de unele dintre cele mai importante științe cum ar fi: *Computer Science*, *Psychology* și *Cognitive Science*.

Aplicațiile realizate pot răspunde emoțiilor omenești, dar și altor stimuli externi. Un dispozitiv cu asemenea capacități, folosește o varietate de surse pentru determinarea emoțiilor. Gesturile, expresiile faciale, vocea sunt doar câteva dintre schimbările semnificative care pot afecta starea emoțională a unei persoane, acestea pot fi detectate și interpretare de către un calculator.

Se poate spune că recunoașterea facială este o tehnică barometrică fiind folosită pentru identificarea unei persoane dar și pentru identificarea stărilor emoționale ale acelei persoane. Sistemul de recunoaștere facială se bazează pe imaginea statică a feței unei persoane(o fotografie) care este de fapt un set de pixeli ordonați după un anumit model (pixelul fiind unitatea funcțională fundamentală a unei imagini digitale).

Sistemul de recunoaștere facială percepe chipul unui individ ca pe o mulțime de pixeli alăturați. Abilitatea sistemului de localizare a feței individului şi nu a imaginilor de fond este esențială în procesul de recunoaștere facială.

Înainte de procesul propriu-zis de recunoaștere este necesară crearea unei galerii de imagini, galeria fiind un set de modele biometrice care servește drept referință în procesul de comparare. Crearea galeriei de imagini presupune parcurgerea următoarelor etape: captarea imaginii, detectarea feței, standardizarea, extragerea trăsăturilor şi crearea șablonului pentru fiecare imagine.

## ***1.2Scopul final al proiectului***

Acest subiect nu este unul foarte abordat datorită faptului că nu este întocmai ușor de realiza și nu prea există variante cu care ar putea fi comparat. Are o mare importanță pentru că realizarea acestei aplicații constituie cunoașterea stărilor emoționale ale unui individ doar din poze, fără a fi nevoie de o altă analiză.

Scopul final al proiectului este acela de recunoaștere ale stărilor emoționale, de percepere și interpretare a trăsăturilor umane pe baza unor poze. Această recunoaștere se va putea face atât din pozele realizate cu camera .

## ***1.3Prezentarea pe scurt a capitolelor existente în lucrare***

În continuare, vor fi prezentate capitolele care vor face parte din această lucrare și prin care voi încerca să lămuresc anumite probleme referitoare la acest subiect. Lucrarea va fi împărțită, așadar, în mai multe capitole și subcapitole. Pentru început voi spune câteva cuvinte despre fiecare în parte, urmând ca descrierea în detaliu să fie făcută în cadrul fiecărui capitol.

Capitolele și subcapitolele prezente în lucrare:

**− Studiul bibliografic și Fundamentare teoretică** − în care va fi expus suportul teoretic care va sta la baza aplicației pe care doresc să o realizez; vor fi descrise limbajele de programare necesare implementării. Vor fi prezentate, în detaliu limbajele și mediile de programare OpenCV, Python, motivele pentru care am ales sa folosesc aceste limbaje și nu altele, comparații între limbaje(asemănări și deosebiri).

− **Specificațiile aplicației** – în acest capitol se va prezenta schema bloc care stă la baza realizării acestei aplicații. Vor fi prezentate componentele acestei scheme, precum și relațiile dintre acestea. Tot în acest capitol, voi vorbi despre modul în care am gândit aplicația, despre interfața aplicației, care trebuie să fie una funcționala, simplă, pe care utilizatorul să o poată folosi fără prea multe probleme.

Capitolul **Specificațiile aplicației** va avea mai multe subcapitole:

− **Funcțiile sistemului** − reprezintă de fapt o descriere a aplicației, o prezentare a ceea ce este ea și ce face; despre funcțiile generale ale sistemului.

− **Interfața cu utilizatorul −** explicarea pe scurt a folosirii interfeței de către utilizator.

− **Structura de baze de date si fișiere −** prezentarea fișierelor folosite în cadrul acestei aplicații.

După capitolul în cadrul căruia sunt prezentate specificațiile, urmează capitolele:

− **Proiectarea de detaliu −** reluarea capitolului anterior și explicarea detaliată a acestuia; se reia fiecare subcapitol și se insistă mai mult pe anumite părți.

− **Utilizarea sistemului −** va fi prezentat un manual de utilizare aplicației pentru cei care vor folosi aplicația, se prezintă detaliat modul de utilizare pas cu pas al aplicației prin expunerea unor capturi de ecran.

− **Realizarea, punerea în funcțiune și rezultatele experimentale −** prezentarea rezultatului final, compararea rezultatului cu ipoteza, prezentarea impactului asupra utilizatorului, problemele întâmpinate la proiectare sau la utilizarea aplicației, rezolvarea acestor probleme.

− **Concluzii –** rezultate obținute, atingerea sau nu a scopului propus.

− **Referințe bibliografice** − menționarea bibliografiei utilizate pentru realizarea lucrării.

# **2. Fundamentare teoretică**

În cadrul acestui capitol voi prezenta fundamentul teoretic care va sta la baza realizării acestei lucrări. Voi prezenta mecanismele pe care le voi folosi în implementare, dar și explicații legate de acestea, voi argumenta de ce am ales aceste mecanisme și nu altele folosindu-mă de comparații între aceste mecanisme și alte mecanisme nefolosite din anumite motive, care desigur le voi preciza si explica în detaliu.

## ***2.1 Scurtă prezentare a mecanismelor folosite în implementare***

Mecanismele pe care le voi folosi sunt următoarele:

* OpenCV este o biblioteca de Python creată pentru a rezolva problemele de *computer vision*. Unul dintre principalele avantaje ale OpenCV este faptul că este foarte optimizată și disponibilă pe aproape toate platformele.
* Python este un limbaj ușor de folosit, oferă posibilitatea unei mai bune structurări, este un limbaj de nivel foarte înalt, dispune de tipuri de date de nivel înalt cum ar ﬁ tablouri sau dicționare foarte ﬂexibile, a căror implementare în C ar lua programatorului zile[2], îl voi folosi tocmai din aceste motive.
* *XML* (*Extensible Markup Language*) este un limbaj simplu care definește reguli de codificare ale documentelor într-un format lizibil omului
* LXML – bibliotecă Python pentru citire și scriere XML-uri

## 

## ***2.2 Domenii studiate***

Aplicația realizată de mine se încadrează în domeniul numit *affective computing* , care permite dezvoltarea de sisteme și dispozitive care pot recunoaște, interpreta, procesa, și simula emoțiile umane . Acest domeniu este unul interdisciplinar care acoperă mai multe științe cum ar fi: știința calculatoarelor, psihologie și științele cognitive. [9]

O motivație pentru cercetarea acestui domeniu o reprezintă abilitatea de a simula empatie. Mașina trebuie să interpreteze starea emoțională a oamenilor și să se adapteze comportamentului lor, dând un răspuns adecvat pentru acele emoții[10].

Pentru detectarea informației emoționale avem nevoie de senzori care captează date despre starea fizică a utilizatorului sau despre comportamentul acestuia fără a interpreta și datele de intrare. Un astfel de exemplu este camera video care poate capta expresii faciale, poziția corpului și gesturi. Alți senzori pot detecta semnalele emoționale cum ar fi temperatura pielii.

Identificarea informațiilor emoționale necesită extragerea modelelor semnificative din datele colectate. Acest lucru se face folosind tehnici de învățare care procesează diferite modalități de: recunoaștere a vocii, procesare a limbajului natural sau de detectare a expresiilor faciale.

În științele cognitive există două modele de conducere care descriu modul în care oamenii percep și clasifică emoțiile: modelul continuu și modelul categoric. Modelul continuu definește fiecare expresie facială ca un vector caracteristic într-un spațiu al feței.[9].

Modelele: categoric și continuu nu pot identifica mai multe emoții. Majoritatea sistemelor actuale sunt dependente de date, iar pentru detectarea emoțiilor bazate pe vorbire, sau detectarea celor bazate pe trăsăturile faciale este nevoie de baze de date adecvate folosite pentru a instrui clasificatorul[9].

Pentru detectarea și prelucrarea expresiilor faciale se folosesc diverse metode , cum ar fi: fluxul optic , modelul Markov ascuns și procesarea de rețele neuronale.

[Paul Ekman](https://translate.googleusercontent.com/translate_c?depth=1&hl=ro&prev=search&rurl=translate.google.ro&sl=en&u=https://en.wikipedia.org/wiki/Paul_Ekman&usg=ALkJrhhS9FkezyJr49nBNh2zIdFYMWQ7Xg)  a definit șase emoții de bază: furie, dezgust, frică, fericire, tristețe, surprindere. Pe baza acestora, a fost conceput un sistem pentru clasificarea și exprimarea fizică a emoțiilor. Acest sistem poartă numele de *FACS* ( creat de Paul Ekman și Wallace V. Friesen în 1978).

Pentru detectarea și prelucrarea facială, este necesară depășirea obstacolelor pentru evidențierea algoritmului și a metodei folosite. Pentru  aceasta se folosește curba lui Gauss , filtrare mediană, sau metode mai noi.

Este cunoscut faptul că gradul de acuratețe în recunoașterea facială nu a fost adus la un nivel suficient de ridicat pentru a permite utilizarea lor pe scară largă( au existat multe încercări, dar nu s-a reușit identificarea cu succes a infractorilor). Fără a îmbunătăți precizia hardware și software folosită pentru a scana fețele, progresul este foarte mult încetinit[9].

Gesturile pot fi utilizate în mod eficient pentru detectarea unei anumite stări emoționale a utilizatorului, în special atunci când este utilizat în conjuncție cu vorbirea și recunoașterea feței. În funcție de acțiunea specifică, gesturile ar putea fi simple reflexe ( ridicarea umerilor atunci când nu se cunoaște răspunsul la o întrebare), sau ar putea fi complexe și pline de înțeles atunci când comunică cu limbajul semnelor. În cazul în care se folosesc obiecte, care pot fi atinse sau mutate, acestea ar trebui să fie și recunoscute, analizate și utilizate eficient de către un computer.

Detectarea stării emoționale a unui utilizator se poate face prin monitorizarea și analiza semnelor fiziologice. Cele trei semne fiziologice principale care pot fi analizate sunt: ​​pulsul volumului sanguin , răspunsul galvanic al pielii , electromiografie facială[9].

Acest domeniu de cercetare este încă în fază incipientă, dar care prinde avânt. Affective computing folosește diverse modalități pentru analizarea emoțiilor uman. Una dintre cele mai importante modalități este *Computer Vision*.

*Computer Vision este un* domeniu care include metode pentru dobândirea , prelucrarea, analizarea și înțelegerea imaginilor și, în general, date high-dimensionale din lumea reală , în scopul de a produce informații numerice sau simbolice. O temă dezvoltată în acest domeniu a fost dublarea capacitățile vederii umane , prin perceperea și înțelegerea unei imagini în format electronic, transformarea imaginilor vizuale în descrieri ale lumii și interfațarea cu alte procese. Această înțelegere a imaginilor poate fi văzută ca despletirea informației simbolice în *date image* folosind modele construite cu ajutorul geometriei, fizicii, statistici, și teoria învățării. *Computer Vison* a fost de asemenea descris ca automatizarea și integrarea unei largi game de procese și reprezentări pentru percepția vederii[11].

*Computer Vison* studiază teoria din spatele sistemelor artificiale care extrag informațiile din imagini. *Image date* poate lua diverse forme: secvențe video, date multi-dimensionale de la un scanner medical. Această disciplină, caută să aplice teoriile și modelele sale în construirea de aplicații *Computer Vision*.

Inteligența artificială și *Computer Vision* împărtășesc aceeași viziune despre subiecte cum ar fi:  recunoașterea formelor și a tehnicilor de învățare. În consecință, *Computer Vison* este uneori văzut ca o parte a domeniului inteligenței artificial, dar și a domeniului *Affective Computing*[11].

*Computer Vision* este opusul graficii pe calculator. În timp ce grafica computerizată produce *image date* de la modele 3D, *Computer Vision* produce modele 3D de la *image date*.

O problemă des întâlnită în Computer Vision este aceea de a procesa imaginile, dar și de a determina dacă imaginile conțin anumite obiecte. Diferite varietăți ale problemei recunoașterii sunt descrise în literatura de specialitate:

* **Recunoașterea obiectului** (**clasificarea obiectelor)** - unul sau mai multe obiecte sau clase de obiecte pot fi recunoscute, de obicei după forma lor. *Blippar*, *LikeThat* furnizează programe independente care ilustrează această funcționalitate.
* **Identificarea** - o instanță individuală a unui obiect este recunoscută. Exemplele includ identificarea feței sau a amprentelor digitale al unei anumite persoane, identificarea cifrelor scrise de mână , sau identificarea unui anumit vehicul.
* **Detecție** –*image date* sunt scanate pentru o anumită afecțiune. Exemplele includ detectarea posibilelor celule sau țesuturi, sau detectarea unui vehicul într-un sistem automat de taxare rutieră. Detecția bazată pe calcule relativ simple și rapide este uneori folosită pentru identificarea unei regiuni mai mici de date a unei imagini[11].

Cei mai buni algoritmi pentru astfel de sarcini sunt cei bazați pe rețele neuronale convoluționale . Performanța acestor rețele pe testele ImageNet, este aproape de cea a oamenilor.

*Computer Vison* este extrem de dependent de aplicație. Unele sisteme sunt stand-alone, altele constituie un sub-sistem al modelului mai mare, care poate conține alte sub-sisteme pentru controlul dispozitivelor de acționare mecanice, planificare, etc.

O imagine digitală din cadrul unei aplicații din acest domeniu este produsă de unul sau mai mulți senzori de imagine, care, pe lângă diferite tipuri de camere sensibile la lumină, includ senzori cu rază, dispozitive de tomografie, radar, camere video ultra-sonic, etc., în funcție de tipul de senzor, rezultând o imagine obișnuită 2D, un volum 3D, sau o secvență de imagini. Valorile pixelilor corespund uneia sau mai multor benzi spectrale (imagini gri sau imagini color)[11]. Este necesară pre-procesarea imaginilor/datelor în scopul îndeplinirii anumitor ipoteze implicite. Câteva exemple ar putea fi:

* + Re-eșantionare pentru a se asigura că sistemul de coordonate al imaginii este corectă.
  + Reducerea zgomotului pentru a se asigura că zgomotul senzorului nu introduce informații false.
  + Îmbunătățirea contrastului pentru a se asigura că informațiile relevante pot fi detectate.

În procesul de prelucrare se iau o serie de decizii cu privire la punctele si regiunile cele mai relevante pentru o prelucrare ulterioară. Exemple:

* + Selecția unui anumit set de puncte de interes.
  + Segmentarea uneia sau a mai multor regiuni de imagini care conțin un anumit obiect de interes.
  + Segmentarea imaginii în scene formate din grupuri de obiecte, obiecte singulare sau părți ale obiectelor importante[11].

După pre-procesare, urmează procesarea la nivel înalt. În cadrul acesteia se folosește un set mic de date de intrare, un set de puncte sau o regiune a unei imagini care se presupune că, conține un anumit obiect. Exemple:

* + Verificarea că datele satisfac modelele de bază.
  + Estimarea parametrilor specifici de aplicare, mărimea obiectului.
  + De recunoaștere a imaginilor - clasificarea unui obiect detectat în diferite categorii.
  + Înregistrarea imaginii - compararea și combinarea a două puncte de vedere diferite ale aceluiași obiect.

Luarea deciziei finale, de exemplu:

* + Pass / fail privind cererile de inspecție automate.
  + Potrivire / nepotrivire în aplicații de recunoaștere.
  + Pavilion pentru examinare umană și mai mult în aplicații medicale, militare, de securitate și de recunoaștereă[11].

## ***2.3 Biblioteca OpenCV***

Pentru a înțelege emoțiile și sentimentele celor din jur, trebuie să le analizăm mai întâi expresiile faciale. Este mult mai complicat ca o „mașină” să facă asta în locul omului. O mașină trebuie instruită și învățată tehnica recunoașterii. Pentru a face acest lucru, sistemul a fost împărțit în 3 module, de pre-procesare, selecție caracteristică și extracție și de clasificare. În această lucrare, sistemul de recunoaștere a expresiilor faciale utilizează *Haar Cascade Clasifier*. Detectarea rapidă a caracteristicilor faciale se realizează mai mult prin concentrarea atenției asupra ochilor și a gurii care reduc foarte mult din calcul. Metoda propusă a fost implementată folosind *OpenCV* .

Aplicația realizată( ca și restul aplicațiilor de tip *Computer Vision*) este interesantă și utilă, dar algoritmii care stau la baza ei sunt intensivi din punct de vedere computațional.

Biblioteca *OpenCV* permite rularea algoritmilor *Computer Vision* în timp real. *OpenCV* a devenit biblioteca standard în acest domeniu. Unul dintre principalele avantaje ale bibliotecii OpenCV este faptul că este foarte optimizată și disponibilă pe aproape toate platformele. OpenCV este una dintre cele mai populare biblioteci utilizate care dezvoltă aplicații de tipul *Vision Computer*. Ea permițând rularea mai multor algoritmi *Computer Vision* în timp real[12].

Unul dintre cele mai bune lucruri pe care le face *OpenCV* este că oferă o mulțime de primitive pentru construirea operațiunilor legate de procesarea imaginii și *Computer Vision.* Definirea unei imagini, a unui punct, a unui dreptunghi sunt fundamentale pentru aproape orice algoritm din cadrul *Computer Vision*. OpenCV conține toate structurile enumerate în modulul de bază. Aceste structuri fiind deja optimizate pentru viteză și memorie.

### ***2.3.1 Modulele din cadrul OpenCV***

Modulul *imgcodecs* se ocupă de scrierea și citirea imaginilor. Dacă se prelucrează o imagine de intrare, imagina rezultată( cea de ieșire) se poate salva ca un jpg. În cadrul prezentei aplicații, se lucrează cu o mulțime de fișiere video.

 Modulul *videoio* se ocupă de tot ce este legat de fișierele video, fie că sunt de intrare, fie că sunt de ieșire. Se poate captura cu ușurință un video de la un webcam sau se poate citi un fișier video în formate diferite. Există posibilitatea să se salveze chiar și mai multe cadre ca un fișier video prin setarea proprietăților, cum ar fi cadre pe secundă, dimensiunea cadrului, și așa mai departe[12].

Operațiunile de procesare se regăsesc în funcțiile din modulul i*mgproc*. Lucrurile care se pot face folosind funcțiile din acest modul sunt: filtrarea imaginilor, operații morfologice, transformări geometrice, conversii de culoare, histograme, analiza formei, analiza de mișcare, de detectare caracteristică, și așa mai departe[12].Mai există însă și un modul numit *ximgproc* care conține algoritmi avansați de procesare a imaginii.

Un alt modul oferit de *OpenCV* este *highgui* care se ocupă de toate operațiile de interfață de utilizator de nivel înalt.  Acest modul are funcții care pot fi folosite pentru a crearea de ferestre pentru a afișajul imaginilor. Există și o funcție de așteptare, care va aștepta până când apăsarea unei taste pentru a trece la pasul următor. O altă funcție utilă folosită pentru dezvoltarea de aplicații interactive și pentru desenarea de dreptunghiuri pe o anumită regiune, este cea care poate detecta un eveniment la apăsarea *mouse*-ului.

Analiza video include sarcini, cum ar fi analiza mișcării între cadre succesive într-un film, de urmărire diferite obiecte într-un film, crearea de modele de supraveghere video, și așa mai departe. OpenCV oferă un modul și un model numit *video* care se ocupă de analiza mișcării intre cadre succesive sau de crearea de modele de supraveghere video. De stabilizarea video se ocupă *videostab*.

Reconstrucția 3D este un subiect important în Computer Vision. Având în vedere un set de imagini 2D, se reconstitui scena 3D folosind algoritmi relevanți. OpenCV prevede algoritmi care pot găsi relația dintre diferitele obiecte din aceste imagini 2D pentru a calcula pozițiile lor 3D. Există un modul numit *calib3d* care se poate ocupa de toate astea. Acest modul se poate ocupa de asemenea, de calibrarea aparatului foto, care este esențială pentru a estima parametrii camerei[12].

Sistemul vizual uman are tendința de a extrage caracteristicile de bază dintr-o anumită scenă, astfel încât să poată fi recuperate ulterior. Pentru imitarea acestui lucru, oamenii au început proiectarea diferitelor extractoare de caracteristici care pot extrage aceste puncte importante dintr-o anumită imagine. Cei mai populari algoritmi includ SIFT, SURF, FAST și așa mai departe. Există un modul numit *features2d* care oferă funcții pentru a detecta și a extrage toate aceste caracteristici. Un alt modul numit *xfeatures2d* oferă câteva extractoare, unele dintre funcții care sunt încă în faza experimentală.

Detectarea obiectelor se referă de fapt la detectarea locației unui obiect într-o anumită imagine. Acest proces nu este influențat de tipul obiectului. Detectarea localizării obiectelor este un pas foarte important în multe sisteme de *Computer Vision*.

Detectarea obiectelor folosite necesită calcul intensiv, din cauza numărului de calcule necesare pentru a efectua detectarea la diferite scări. Pentru a rezolva acest lucru, Paul Viola si Michael Jones au venit cu un mare algoritm, oferind o modalitate rapidă de a proiecta un detector de obiect pentru orice obiect. *OpenCV* are module numite *objdetect* și *xobjdetect* care oferă cadrul pentru a proiecta un detector de obiect[12].

*Computer Vision* utilizează diverși algoritmi de învățare automată pentru a realiza lucruri diferite. *OpenCV* oferă un modul numit *ml*, care are mulți algoritmi de învățare incluși în ea. Unii dintre algoritmi include: *Bayes clasificator*, *Support Vector Machines*, arbori de decizie, rețele neuronale, și așa mai departe. Există și un modul numit *Flann* care conține algoritmi pentru căutări rapide( cel mai apropiat vecin în seturi mari de date). Algoritmii de învățare mașină sunt utilizați pe scară largă pentru a construi sisteme de recunoaștere a obiectului, clasificarea imaginii, de detectare a feței, căutări vizuale.

Noțiunea de *shape* este crucială în Computer Vision. Noi analizăm datele vizuale prin recunoașterea diferitelor forme dintr-o imagine. Acesta este de fapt un pas important în mulți algoritmi. Sunt oferiți toți algoritmii necesari pentru a extrage diferite forme.

Algoritmii *optical flow* sunt utilizați în video pentru urmărirea funcțiilor din cadre succesive. Să presupunem că doriți să urmăriți un anumit obiect dintr-un film. Este nevoie de extragerea caracteristicilor din cadrul curent și apoi de urmărirea acestor caracteristici în cadre succesive.

Recunoașterea feței se referă la identificarea persoanei dintr-o imagine. Acest lucru nu este același lucru cu detectorul feței. Dacă se dorește construirea unui sistem practic biometric care poate recunoaște persoana din fața camerei, trebuie mai întâi rulat detectorul de față care poate identifica locația feței, după care trebuie executat un *recognizer* al feței care poate recunoaște acea persoană. Există un modul numit *face* care se ocupă de recunoașterea feței. Așa cum sa discutat mai devreme, *Computer Vision* încearcă să modeleze algoritmi pe baza modului in care oamenii percep datele vizuale. Pentru a găsi regiuni și obiecte importante în imagini, cum ar fi recunoașterea obiectelor, detectarea obiectelor este utilă folosirea modului numit *saliency* care este proiectat în acest scop. Acesta oferă algoritmi care pot detecta regiunile importante în imagini și videoclipuri statice[12].

Anumite aplicații includ recunoașterea plăcuței de identificare, recunoaștere a semnelor de circulație pentru mașinile, scanarea cărților în format digital. Pentru aceste aplicații există un modul numit *text* care conține diverși algoritmi pentru a detecta și a recunoaște un text.

### ***2.3.2 Inspecție optică, segmentare și detectare***

Sectorul industrial utilizează sisteme complexe Vision și hardware. *Computer Vision* încearcă să detecteze problemele și să minimizeze erorile produse în procesul de producție.

În zilele noastre, utilizarea de inspecție optică, folosind diferite tipuri de camere, cum ar fi camere video în infraroșu, 3D depinde de obiectele de măsură, de detectarea efectelor de suprafață.

Se extrag atât caracteristicile fiecărui obiect segmentat precum și algoritmul pentru a identifica fiecare obiect de clasă pentru a permite clasificarea obiectele noastre.

Pașii care trebuie urmați pentru preprocesare sunt:

• eliminarea zgomotului de ștergere

• Iluminare

• Binarizare

În etapa de segmentare, se folosesc doi algoritmi diferiți:

• Algoritmul de detecție a conturului

• conectata de extragere a componentei (etichetare)[12]

Pentru crearea unei aplicații, avem nevoie de câțiva parametri de intrare atunci când utilizatorul le execută:

• O imagine de intrare pentru a fi procesată

• Modelul imaginii de lumină

• Operațiunea de lumină

Preprocesarea este prima modificare pe care o facem într-o nouă imagine.

În mod normal, în etapa de preprocesare, se reduce la minimum zgomotul imaginii, condițiile de iluminare, sau deformările imaginii datorita lentilelor camerei. Acești pași minimizează erorile atunci când se încearcă să se detecteze obiecte.

Dacă nu se elimină zgomotul, se detectează mai multe obiecte decât ne așteptăm, deoarece în mod normal, zgomotul este reprezentat ca un punct mic din imagine și poate fi segmentat ca un obiect. Circuitul senzorului și în mod normal, scanerul produce acest zgomot. Această variație de luminozitate sau culoare poate fi reprezentată în diferite tipuri, cum ar fi zgomotul gaussian, Spike zgomot, și zgomotul împușcat. Există diferite tehnici care pot fi folosite pentru a elimina zgomotul[12].

Cum putem elimina lumina din imaginea noastră? Este foarte simplu; avem nevoie doar de o imagine, fără nici un obiect, care este luată de la exact aceeași poziție de unde au fost luate alte imagini, și să aibă aceleași condiții de lumină.

Apoi, cu o operație matematică simplă, putem elimina acest model de lumină. Există două opțiuni pentru a le elimina:

• Diferență

• Diviziunea imaginilor reprezintă cea mai simplă abordare.

Dacă avem modelul L de lumină și de imagine I, rezultatul de îndepărtare R este diferența dintre ele: R = L-I . Această diviziune este un pic mai complexă, dar simplă, în același timp. Dacă avem o matrice a luminii de model L și matricea de imagine I, rezultatul de îndepărtare R este după cum urmează: R = 255 \* (1- (I / l)).

În cazul în care modelul nostru de lumină este de culoare albă, iar obiectele sunt mai întunecate decât banda purtătoare de fond, atunci valorile pixelilor imaginii vor rămâne întotdeauna la fel sau vor fi mai mici decât valorile pixelilor luminoși. Apoi, rezultatul pe care îl obținem de la I / L este între 0 și 1. În cele din urmă, am inversat rezultatul acestei diviziuni pentru a obține aceeași gamă de direcție de culoare și îl înmulțim cu 255 pentru a obține valori între intervalul 0-255. În codul nostru, vom crea o nouă funcție numită *removeLight* cu următorii parametri:

• O imagine de intrare pentru a elimina lumina / fundal

• model de lumină mat

• Metoda: 0 este diferența, 1 divizia, ieșirea este o matrice nouă imagine fără lumină /fundal[12].

Când nu avem un model de lumină, cu ajutorul unui filtru, putem crea unul care poate fi utilizat, dar există algoritmi mai buni de la care se poate afla fundalul de la câteva imagini, în cazul în care apar piesele în diferite domenii. Pentru a estima imaginea de fundal, vom folosi o estompare cu o dimensiune miez mare care este aplicată la imaginea noastră de intrare. Aceasta este o tehnică comună utilizată în OCR în cazul în care literele sunt relativ subțiri și mici pentru întregul document, și ne permite să efectuăm o aproximare a modelelor de lumină din imagine.

După scoaterea fundalului, trebuie doar să se binarizeze imaginea pentru segmentarea viitoare. Acum, vom aplica funcția de prag cu ajutorul a două valori de prag diferite: o valoare foarte scăzută atunci când eliminăm lumina / fundal, deoarece toate regiunile de bază non-dobândă sunt valori negre sau foarte mici, și o valoare medie, atunci când nu folosim o îndepărtare ușoară.

Două tehnici utilizate pentru segmentarea imaginii noastre thresholded:

• Componentele conectate

• Funcția findContours

Cu aceste două tehnici, se poate extrage fiecare regiune de interes a imaginii noastre în cazul în care apar obiectele noastre țintă[12].

Componenta conectată este un algoritm foarte comun folosit pentru a identifica segmente și părți în imagini binare. O componentă conectată este un algoritm iterativ folosit în scopul etichetării unei imagini cu ajutorul unui pixel cu 8 sau 4-conectivitate. Doi pixeli sunt conectași în cazul în care au aceeași valoare și sunt vecini.

### ***2.3.3 Clasificarea obiectelor***

Pentru clasificarea fiecărui obiect, trebuie să instruim sistemul, să-l facem să fie capabil să învețe parametrii necesari pentru a decide care etichetă specifică ar trebui să fie atribuită obiectului detectat.

Pentru fiecare obiect, vom extrage diferitele caracteristici pe care sistemul trebuie să le analizeze folosind un algoritm de învățare. Cu ajutorul unui algoritm de învățare, suntem capabili de a demonstra( folosind interfața noastră de utilizator), etichetele fiecărui obiect detectat în imaginea de intrare.

Mașina de învățare implică o recunoaștere a modelului și teoria învățării în domeniul inteligenței artificiale și este legată de statisticile de calcul. Este utilizată în sute de aplicații, cum ar fi OCR (recunoaștere optică a caracterelor), filtrarea spam-ului, motoarele de căutare, și mii de aplicații informatice.

În funcție de modul în care mașina de algoritmi ML trebuie să învețe din date sau mostre, există 3 moduri de învățare:

• învățare supervizată: Calculatorul învață dintr-un set de date etichetate. Scopul este de a învăța parametrii modelului și regulile care permit computerelor să mapeze relația dintre date și rezultate de etichete de ieșire.

 • învățare nesupravegheată: Nu există etichete, sunt date, iar calculatorul încearcă să descopere structura de intrare a datelor de intrare.

• învățare Armătură: Calculatorul interacționează cu un mediu dinamic care realizează scopul și învață din greșelile sale.

În funcție de rezultatele dorite pe care le obținem de la algoritmul nostru de învățare automată, le putem clasifica în următoarele:

• Clasificare: În clasificare, spațiul intrărilor pot fi împărțite în clase N. Aceasta este una dintre categoriile cele mai utilizate. Un exemplu tipic este o filtrare de spam de e-mail în cazul în care există doar două clase: de spam și de non spam sau OCR, în cazul în care sunt disponibile numai N caractere, iar fiecare caracter este o singură clasă.

 • Regresie: Rezultatul este o valoare continuă în locul unei valori discrete, cum ar fi un rezultat de clasificare. Un exemplu de regresie poate fi de predicție a prețului casei furnizând mărimea casei, numărul de ani, și locația.

 • clustering: Intrările sunt împărțite în N grupuri de formare folosind nesupravegheată.

• Estimarea Densitate: Aceasta găsește (probabilitate) distribuția de intrări.

În procesul de învățare mașină, există mai multe abordări și metode, iar unele dintre ele folosite sunt SVM (mașini vector suport), ANNs (rețele neuronale artificiale), grupare cum ar fi K-apropiați vecini, arbori de decizie, sau învățare profundă, care este abordarea rețele neuronale utilizate. Toate aceste metode și abordări sunt susținute, puse în aplicare, și bine documentate în OpenCV[12].

În procesul de învățare mașină, cea mai mare consumatoare de timp parte este metoda de formare. Aceasta poate dura de la câteva secunde până la câteva săptămâni sau luni pentru seturi mari de date și structuri complexe de învățare automată. Acest lucru înseamnă că nu ne putem antrena algoritmul nostru de fiecare dată când vom rula aplicația, și este recomandat să salvam modelul după ce a fost antrenat pentru că toți parametrii de formare / predicție *machine learning* să fie salvați.[img1]

Metoda de training este responsabilă pentru învățarea parametrilor modelului de la un set de date de formare.

Aproape orice aplicație Computer Vision începe cu o etapă de preprocesare, care se aplică imaginii de intrare. Preprocesarea implică condiții de îndepărtare a luminii și a zgomotului, Thresholding. După ce aplicăm toate etapele de preprocesare necesare pentru imaginea de intrare, al doilea pas este segmentarea. În etapa de segmentare, avem nevoie pentru a extrage regiunile de interes ale unei imagini și de a le izola pentru a găsi obiectele de interes. De exemplu, într-un sistem de detectare a feței, avem nevoie pentru a separa fețele de restul pieselor din scenă. După obținerea obiectelor din interiorul imaginii, vom continua cu pasul următor. Avem nevoie pentru a extrage toate caracteristicile fiecăruia dintre obiectele detectate; o caracteristică este un vector de caracteristici ale obiectelor. O caracteristică care descrie obiectele noastre poate fi suprafața obiectului, conturul, modelul. [img2]

Acum, avem descriptorul obiectului nostru; un descriptor este o caracteristică care descrie un obiect, acești descriptori folosindu-se pentru instruire. Pentru a face acest lucru, trebuie creat un set mare de date de caracteristici, în cazul în care sunt preprocesate sute, mii și milioane de imagini.[img3]

### ***2.3.4 Haarcascade training***

*Haar cascades* sunt clasificatori în cascadă, care se bazează pe caracteristicile Haar. Ce este un clasificator cascadă? Este pur și simplu o înlănțuire a unui set de clasificatori slabi, care pot fi folosiți pentru a crea un clasificator puternic. Ce se înțelege prin clasificatori slabi și puternici? clasificatori slabi sunt clasificatori ale căror performanțe sunt limitate. Ei nu au capacitatea de a clasifica totul corect. Clasificatorii puternici sunt foarte buni la clasificarea corectă a datelor noastre.  O altă parte importantă a *Haar cascades* sunt caracteristicile Haar[12]. Aceste caracteristici sunt simple dreptunghiuri dintr-o imagine. Să considerăm următoarea figură:



Fig 2.3.4.1

Dacă dorim să calculăm caracteristicile Haar ale regiunii ABCD, trebuie să calculăm diferența dintre pixelii albi și pixelii de culoare din acea regiune. Vom folosi diferite modele pentru a construi caracteristici Haar.

Algoritmul utilizează o cascadă amplificată de clasificatori slabi. Acest sistem este folosit pentru a construi un clasificator puternic, care se poate realiza foarte bine. Pentru a construi un astfel de sistem, trebuie colectate imagini care pot fi folosite pentru instruirea acestuia. Avem nevoie de imagini pozitive, dar și de imagini negative. Sistemul va ingera aceste date și apoi va învăța să facă diferența între acestea.

Pentru a asigura calitatea clasificatorilor, Viola și Jones au descris o tehnică *nifty* în etapa cascadă. Primul a fost extragerea caracteristicilor din toate imaginile. Ei au folosit caracteristici Haar în lucrarea lor pentru a construi vectori de trăsături. Odată ce aceste caracteristici sunt extrase, vor fi trecute printr-o cascadă de clasificatori. Urmează verificarea tuturor subregiunilor dreptunghiulare și păstrarea celor care nu conțin fețe.

Pentru a extrage caracteristicile Haar, trebuie să calculăm suma valorilor pixelilor închiși în multe regiuni dreptunghiulare ale imaginii. Pentru a face această scală invariantă, trebuie să calculăm aceste zone la scări multiple (adică, pentru diferite dimensiuni ale dreptunghiului). Dacă va fi implementat în mod naiv, acest lucru ar fi un proces intensiv foarte computațional. Trebuie să evităm iterarea acelorași pixeli de mai multe ori. Pentru a evita acest lucru, putem folosi imaginile integrale. Aceste imagini pot fi inițializate într-un timp liniar. Pentru a înțelege mai bine, să aruncăm o privire la figura de mai jos:



*Fig 2.3.3.2*

Dacă dorim să calculăm suprafața oricărui dreptunghi din imagine, nu trebuie să iterăm toți pixelii din acea regiune. Putem să considerăm un dreptunghi format dintr-un punct din stânga-sus și orice punct P ca și colț opus. AP denotă zona acestui dreptunghi. De exemplu, în figura precedentă, AB denotă zona 5 x 2 dreptunghiul fiind format prin preluarea punctului din stânga sus și B ca și colțuri opuse. Hai să aruncăm o privire la figura următoare în scopul clarității[12]:



*Fig 2.3.3.3*

Pixelii colorați indică zona dintre pixel sus-stânga și punctul A. Aceasta este notat cu AA. Diagramele rămase sunt notate cu: AB, AC și AD. Pentru calcularea ariei dreptunghiului ABCD folosim formula: aria dreptunghiului ABCD = AC - (AB + AD - AA)

Lucrul cu un clasificator cascadă include două etape majore: de formare și de detecție.

Există două aplicații în OpenCV pentru „instruirea” cascadei clasificator: *opencv\_haartraining* și *opencv\_traincascade*. *Opencv\_traincascade* este o versiune mai nouă, scrisă în C ++ conform OpenCV 2.x API. Principala diferență dintre aceste două aplicații este că *opencv\_traincascade* suportă ambele tipuri de caracteristici: Haar [Viola2001] și LBP [Liao2007] (modele binare locale).  În ceea ce privește LBP calitatea de detectare depinde de calitatea setului de date de formare și de parametrii de formare. *Opencv\_traincascade* și *opencv\_haartraining* stochează clasificatorul în diferite formate de fișiere. Noua variantă de *opencv\_traincascade* suportă ambele formate. *Opencv\_traincascade* poate salva o cascadă formată în format mai vechi. Există unele utilitare auxiliare legate de formare. *Opencv\_createsamples* este utilizat pentru a prepara un set de date de formare de probe pozitive și de testare. *Opencv\_createsamples* produce set de date probe pozitive într-un format care este acceptat atât de *opencv\_haartraining* și *opencv\_traincascade*. Ieșirea este un fișier cu extensia \* .vec, acesta este un format binar care conține imagini. *Opencv\_performance* pentru evaluarea calității clasificatorilor, dar numai antrenat de *opencv\_haartraining*. Pentru întocmirea raportului de performanță  avem nevoie de o colecție de imagini, de numărul de obiecte găsite, numărul de obiecte pierdute, numărul de alarme false și alte informații. Din moment ce *opencv\_haartraining* este o aplicație învechită care nu se mai prea folosește în momentul de față, doar *opencv\_traincascade* va fi descrisă în continuare[13].

Un utilitar *opencv\_createsamples* oferă funcționalități pentru generarea setului de date. Pentru formare avem nevoie de un set de mostre. Există două tipuri de probe: negative și pozitive. Eșantioanele negative corespund imaginilor non-obiect. Probele pozitive corespund imaginilor cu obiecte detectate. Setul de eșantioane negative, trebuie să fie pregătite manual, în timp ce un set de probe pozitive este creat folosind utilitarul *opencv\_createsamples*.

Probele negative sunt luate din imagini arbitrare. Aceste imagini nu trebuie să conțină obiecte detectate. Eșantioanele negative sunt enumerate într-un fișier special. Este un fișier text în care fiecare linie conține un nume de fișier imagine (relativ la directorul fișierului de descriere) a imaginii eșantionului negativ. Acest fișier trebuie creat manual.

Probele pozitive sunt create de utilitarul *opencv\_createsamples*. Acestea pot fi create dintr-o singură imagine cu un obiect sau dintr-o colecție de imagini marcate anterior. Cantitatea și gama de dezordine pot fi controlate prin argumente din linia de comandă. Exemple de argumente în linia de comandă:

*-vec* numele fișierului de ieșire ce conține probele pozitive pentru formare

*-img* Imagine obiect sursă

 -*bg*fișier descriere fundal; conține o listă de imagini, care sunt folosite ca fundal pentru versiunile distorsionate în mod aleatoriu ale obiectului

-*num* Numărul de eșantioane pozitive pentru a genera

-*bgcolor* Culoare de fundal (în prezent, imaginile în tonuri de gri sunt asumate). Din moment ce ar putea exista artefacte de compresie, cantitatea de toleranță de culoare poate fi specificată de *-bgthresh*. Toți pixelii *withing bgcolor-bgthresh* și *bgcolor + bgthresh* gama sunt interpretate ca transparente.

 -*bgthresh -inv* În cazul în care sunt specificate, culorile vor fi inversate*.*

*-randinv* În cazul în care sunt specificate, culorile vor fi inversate în mod aleatoriu.

 -*maxidev* deviație de intensitate maximă a pixelilor în probe de prim-plan.

-*maxxangle*

-*maxyangle*

-*maxzangle*

Unghiurile maxime de rotație trebuie să fie date în radiani. În cazul în care se specifică, fiecare probă va fi afișată. Apăsarea butonului Esc va continua procesul de creare a eșantioanelor.

 -*w*Lățime (în pixeli) a probelor de ieșire.

-*h* Înălțimea (în pixeli) a probelor de ieșire.

-*pngoutput* Cu această opțiune activată opencv\_createsamples generează o colecție de eșantioane PNG și un număr de fișiere de adnotări asociate, în loc de un singur fișier vec. Utilitarul poate opencv\_createsamples lucra într-un număr de moduri, și anume:

-Crearea setului de formare dintr-o singură imagine și o colecție de medii:

-cu un singur fișier vec ca o ieșire;

-cu o colecție de imagini JPG și un fișier cu listă adnotări ca o ieșire;

-cu o colecție de imagini PNG și fișiere asociate cu adnotări ca o ieșire;

- Convertirea colecției marcate a probelor într-un format vec;

- Se afișează conținutul fișierului VEC[13].

Pentru a obține un astfel de comportament: -img, -bg, -info și ar trebui să fie specificate. Numele fișierului specificat cu -info ar trebui să includă cel puțin un nivel de ierarhie director, acel director va fi folosit ca directorul de nivel superior pentru setul de formare.

Probele pozitive, de asemenea, pot fi obținute dintr-o colecție de imagini marcate anterior. Această colecție este descrisă printr-un fișier text similar cu descrierea de fundal fișier. Fiecare linie a acestui fișier corespunde unei imagini. Primul element al liniei este numele fișierului. Acesta este urmat de numărul de instanțe obiect. Următoarele numere sunt coordonatele obiectelor (x, y, lățime, înălțime).

## ***2.4 Limbajul Python***

Acest limbaj este cel mai potrivit pentru elaborarea de *script*-uri, precum și pentru dezvoltarea de aplicații în mai multe domenii, pe majoritatea platformelor.

Este un limbaj des utilizat deoarece este ușor de învățat, dar este în același timp și un limbaj de programare puternic. Are nenumărate avantaje: oferă posibilitatea unei mai bune structurări, reprezintă un bun suport pentru programe mai mari.

Pe de altă parte dispune de o mult mai bună verificare a erorilor decât C-ul, și fiind un limbaj de nivel foarte înalt, dispune de tipuri de date de nivel înalt cum ar fi tablouri sau dicționare foarte flexibile, a căror implementare în C ar lua programatorului zile. Datorită caracterului general al tipurilor de date Python poate fi aplicat într-o arie mult mai largă de probleme decât AWK sau PERL. Cu toate astea, multe lucruri se realizează la fel de ușor în Python ca și în aceste limbaje[1].

Este foarte util deoarece permite divizarea programelor în module care pot fi folosite apoi în alte programe Python.

Fiind un limbaj interpretat, acesta accelerează munca în timpul procesului de dezvoltare, nu este nevoie ca de fiecare dată să se facă o compilare a codului.

Limbajul *Python* permite scrierea unor programe foarte compacte¸ ușor de citit. Programele scrise în Python sunt mult mai mici decât echivalentele în C sau C++ pentru că:

− **tipurile de date de nivel înalt** permit realizarea operațiilor complexe într-o singură instrucțiune;

−**gruparea instrucțiunilor** este dată de paragrafare în loc de blocuri begin/end sau de acolade;

−**nu este necesară declararea variabilelor**.

Python este un limbaj extensibil: dacă aveți cunoștințe de C puteți foarte ușor sa adăugați o nouă funcție sau un modul, fie pentru a efectua operații la viteză maximă, fie pentru a lega programe Python la biblioteci disponibile numai în formă binară (biblioteci grafice).[1]

### ***2.4.1 Structuri de control în Python***

Python dispune de structurile de control obișnuite, întâlnite și în celelalte limbaje. Acestea sunt:

− **instrucțiuni *if*** (instrucțiunea prezentă în toate limbajele de programare).

Fără această instrucțiune, nu s-ar putea face verificarea uneia sau mai multor condiții. În cadrul unui program pot exista mai multe secțiuni *elif* sau niciuna, iar secțiunea *else* este opțională. Cuvântul cheie “*elif”*  fiind prescurtarea de la ”*elseif”*, acesta este folositor pentru a evita tabularea excesivă. O secvență *if..elif..elif* funcționează ca un bloc case sau *switch*, secvențe proprii altor limbaje de programare [1].

− **instrucțiuni *for***

Instrucțiunile *for* din *Python* sunt diferite. Iterațiile instrucțiunii *for* din *Python* funcționează după elementele listei sau a unui șir, pe când în alte limbaje de programare, de exemplu în limbajul *C* iterația este una flexibilă, programatorul putând defini atât pasul cât și condiția de oprire.

− **funcția *range()***

Această funcție este folosită pentru generarea unor liste ce conțin progresii aritmetice. Parametrul furnizat funcției nu va fi niciodată un membru al secvenței. Secvența generată de această funcție să nu înceapă de la 0. Rația progresiei aritmetice poate fi modificată de fiecare dată.

**− instrucțiuni *break***

Instrucțiunea din cadrul acestui limbaj are aceeași semnificație ca și în limbajul de programare *C*, adică terminarea forțată a unei bucle *while* sau *for.*

Instrucțiunile din cadrul unei clauze else sunt executate atunci când bucla se termină odată cu terminarea listei (*for*) sau atunci când condiția buclei devine falsă (pentru *while*) aceste instrucțiuni nu sunt executate dacă bucla este terminată printr-o instrucțiune *break.*[1]

**− instrucțiuni *continue***

Această instrucțiune trece necondiționat la următoarea iterație.

**− instrucțiuni *pass***

Instrucțiunea “*pass*” poate fi folosită atunci când este necesară prezența sintactică a unei instrucțiuni, fără ca aceasta să execute ceva.[1]

**− instrucțiuni *funcții***

Cuvântul *def* este un cuvânt cheie în Python, este primul cuvânt folosit în definiția unei funcții. Acesta este urmat de numele funcției și de o listă de parametrii ai respectivei funcții.

Instrucțiunile care formează funcția încep pe linia următoare, și sunt plasate la o distanță diferită de margine față de definiția funcției.

Prima instrucțiune din corpul funcției poate fi un șir decaractere (opțional), acest șir reprezentând documentația funcției. În momentul în care se produce execuția unei funcții, se creează o nouă tabelă de simboluri, care se folosește pentru variabilele locale. Toate atribuirile din cadrul unei funcții vor stoca valorile în tabela de simboluri locală. Atunci când interpretorul găsește un nume de variabilă, întâi caută în tabela de simboluri locală, apoi în cea globală, și în final în cea predefinită. Cu toate acestea variabilelor globale nu li se pot atribui valori în cadrul unei funcții, decât dacă se folosește instrucțiunea global.

Parametrii actuali ai unei funcții apelate sunt introduși în tabela de simboluri locală a acelei funcții în momentul apelării ei. Transferul argumentelor făcându-se prin utilizarea unui apel prin valoare(unde valoarea este întotdeauna un obiect referință, nu valoarea obiectului).Când o funcție apelează o alta funcție, este creată o nouă tabelă de simboluri pentru funcția apelată.

Definirea unei funcții trece numele funcției în tabela de simboluri curentă.[1]

**− Liste în *Python***

Acest tip de date dispune de câteva metode. Acestea sunt:

**•** *append(x) −*Adăugarea unui element la sfârșitul listei.

**•** *extend(L) −*Adăugarea elementelor furnizate ca și parametru la sfârșitul listei.

**•** *insert(i, x) −*Inserarea unui element într-o anumită poziție. Primul argument reprezintă indicele elementului din listă înaintea căruia se va face inserția.

**•** *remove(x)* −Ștergerea din listă a primului element găsit cu valoarea ’x’. Dacă nu există un astfel de element apare o eroare.

**•***pop([i])* − Ștergerea din listă a elementului de pe poziția ’i’, și returnarea valorii acestuia.

**•** *index(x) −* Întoarce indicele primului parametru din listă care are valoarea ’x’.

**•***count(x) −* Întoarce numărul de apariții ale valorii ’x’ între elementele listei.

**•***sort() −* Sortează elementele listei.

**•** *reverse() −* Schimbă ordinea elementelor din listă[1].

Din cauza organizării elementelor în cadrul fișierelor *Xml* pe care doresc să le convertesc într-un format ușor de vizualizat, voi folosi atât liste cât și dicționare; așadar parcurgerea se va face în funcție de tipul de date prezent în fișiere. În momentul conversiei se va ține cont de aceste structuri de date, conversiile fiind diferite, în sensul că, dacă un element este de tip listă va arăta diferit față de cel care inițial era de tip dicționar.

**−Dicționare**

La fel ca și lista, dicționarul reprezintă un tip de date predefinit, pe care îl voi folosi în implementare, fiind foarte folositor. Se pot întâlni și în alte limbaje de programare dicționare, dar sub o altă denumire: “memorii asociative” sau “ valori asociative”.

Dicționarele sunt indexate de *chei*, care pot fi definite de oricare din tipurile de date invariabile(nemodificabile). Perechile pot fi folosite drept chei ale unui dicționar numai dacă conțin obiecte invariabile. Nu se pot folosi drept chei, listele deoarece acestea pot fi modificate folosind metode ca *append()* sau *extend()*.Este indicat să ne gândim la un dicționar ca la o mulțime neordonată de perechi cheie-valoare, cu observația că o cheie trebuie să fie unică într-un dicționar. O pereche de acolade creează un dicționar gol[1].

Operațiile realizate de un dicționar sunt:

**•** stocarea unei valori cu anumită cheie

**•**  extragerea unei valori cunoscându-se o anumită cheie.

O pereche cheie-valoare poate fi ștearsă folosind instrucțiunea *del*. Dacă se adaugă o valoare în dicționare pentru o cheie care există deja, valoarea veche asociată acelei chei este pierdută.

Metoda *keys()* a unui obiect dicționar întoarce o listă cu toate cheile existente în respectivul dicționar. Lista returnată este sortată, însă pentru a o sorta se poate folosi metoda sort() a obiectului listă returnat de funcție.

## ***2.5 Limbajul XML***

Fișierele folosite in detectarea trăsăturilor umane sunt reprezentate de o serie de coordonate și au formatul XML, din această cauză voi încerca să descriu pe scurt acest limbaj, pentru ca oricine va vedea fișierele în acest format să poată să înțeleagă elementele de bază, să poată să-și facă o idee referitor la ce se află în acel fișier, chiar dacă până acum nu a avut de-a face cu acest limbaj.

Limbajul XML, acronimul de la *eXtensible Markup Language*, a fost definit de consorțiul W3C în anul 1996.

Voi prezenta în câteva cuvinte XML-ul:

* la fel ca şi HTML se bazează pe *tag*-uri.
* XML poate fi folosit de autorii de pagini Web ale căror nevoi depășesc limitele HTML-ului.
* XML a fost creat pentru descrierea structurilor de date
* Tag-urile XML nu sunt predefinite; altfel spus, XML permite crearea tag-urilor personalizate, ceea ce este nemaipomenit.
* Pentru descrierea structurilor de date, documentele XML pot utiliza un DTD (Document Type Definition).
* Un document care conține un DTD este auto descriptiv.

*Xml* -ul este un metalimbaj utilizat pentru descrierea datelor.[3]

Fie că se dorește crearea unui document *Xml* sau doar studierea și parcurgerea unui document *Xml* trebuie ținut cont de faptul că acest metalimbaj are niște reguli care trebuie respectate. Acestea reguli pot fi: de sintaxă *Xml* și de validitate a documentelor *Xml*.

Reguli de sintaxă *Xml:*

**− Închiderea tuturor elementelor −** toate elementele care se găsesc în cadrul unui document *Xml* trebuie să aibă un tag de deschidere și unul de închidere.

**− Închiderea tuturor elementelor vide −** dacă în cadrul unui document *Xml* există unul sau mai multe elemente vide, acestea nu pot fi lăsate deschise, trebuie închise.

**− Utilizarea ghilimelelor pentru a delimita valorile unor atribute −** dacă se folosesc atribute, valorile acestora trebuie neapărat puse între ghilimele.

**− Toate atributele folosite într-un document *Xml* trebuie să conțină valori −** fiecare atribut trebuie să aibă atribuită o valoare.

**− Elementele *Xml* sunt *case-sensitive* −** spre deosebire de alte limbaje, limbajul *Xml* este unul *case-sensitive*, literele mici fiind interpretate diferit față de cele mari. Așadar elementele dar și atributele trebuie utilizate așa cum au fost ele descrise în *DTD* –ul (*Document Type Definition*) asociat. În cazul în care nu se folosește un *DTD*, trebuie avut grijă ca fiecare *tag* de deschidere să aibă un *tag* de închidere.

**− Imbricarea corectă a elementelor −** un element poate conține și alte elemente, asta înseamnă că elementele se pot imbrica. Există și niște reguli referitoare la modul de imbricare și anume:

− Fiecărui tag de deschidere trebuie să-i corespundă un tag de închidere. Numele celor două tag-uri trebuie să fie identice.

− Tag-ul de închidere al unui element “fiu” trebuie să preceadă tag-ul de închidere al „părintelui” său. Altfel spus, un element „fiu” trebuie să fie închis înaintea „părintelui” său.

− Elementele imbricate sunt „fii ai elementului părinte”[3].

Entitățile analizate recursiv sunt interzise, adică un element nu se poate referi la el însuși.

Un document *Xml* este un document “bine format” dacă respectă anumite reguli structurale și notații *Xml*.

Reguli de validitate a documentelor *Xml* :

**− Utilizarea elementului rădăcină −** orice document *Xml* trebuie să conțină cel puțin un element rădăcină, iar acest element trebuie să le conțină pe celelalte elemente ale documentului. Un element rădăcină este delimitat printr-un tag de deschidere plasat la începutul documentului şi un tag de închidere plasat la finele documentului[3].

**− Un document*Xml* este considerat valid dacă conține un *DTD (Document Type Definition)* − a**șadar un document *Xml* este format din două părți:

**− Prolog (declarația documentului)**

**− Elementul rădăcină**

După această scurtă introducere în *Xml* și însușirea unor cunoștințe de bază în acest limbaj, putem să privim altfel fișierele cu care avem de-a face.

# **3. Specificațiile aplicației**

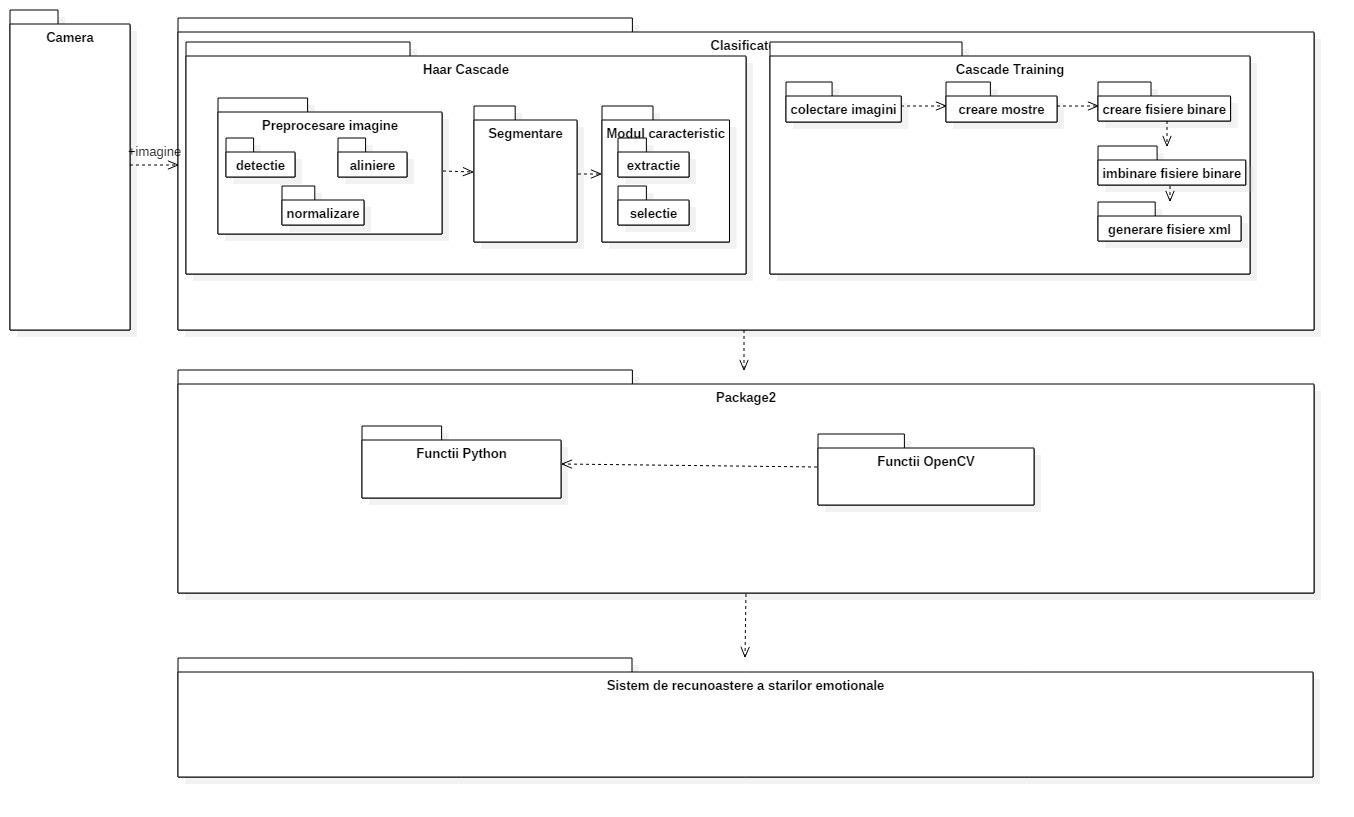
## ***3.1 Schema bloc a sistemului. Scurtă descriere a aplicației***

Aplicația trebuie să poată să recunoască stările emoționale ale unei persoane în funcție de trăsăturile faciale ale persoanei respective. Cu ajutorul camerei web, sunt realizate niște capturi ale fețelor persoanelor, iar aceste capturi/poze sunt prelucrate pentru a obține în final rezultatul dorit și anume aflarea stării emoționale a persoanei care folosește aplicația în acel moment.

Pentru prelucrarea acestor poze avem nevoie de un set de informații suplimentare:

* O bază de date cu poze ale fețelor unor persoane de diferite rase; poze care surprind diferite stări emoționale ale acestor persoane
* Diverse coordonate obținute pe baza acestor poze; coordonate care apar sub forma unor fișiere cu extensia xml
* Diverși algoritmi care stau în spatele funcțiilor de *OpenCV*, necesari în procesul de prelucrare și obținere a rezultatelor finale

La baza oricărei aplicații stă o schemă logică, în care sunt prezentate componentele de bază ale aplicației, precum și legăturile dintre ele. Voi prezenta schema care stă la baza aplicației mele, dar și schema care stă la baza întregului sistem de recunoaștere, încercând să evidențiez prin părțile realizate de mine, prezentând și explicând fiecare modul în parte, legătura dintre ele, importanța fiecărui modul din cadrul acestui uimitor sistem și mai ales importanța modulelor/aplicației mele. Schema generală a proiectului este prezentată în Fig3.1.1.



*Fig 3.1.1 Schema generală a sistemului*

Voi încerca să explic pe larg toate modulele, legăturile dintre ele și cum în final, acestea puse cap la cap o să producă un anumit rezultat, rezultatul trebuind să fie desigur cel așteptat.

Așadar, întreg proiectul se împarte in 5 module, unele mai complexe, altele mai puțin complexe si anume:

Modulul ***Camera*** reprezintă *input*-ul acestui sistem complex. Pentru ca această aplicație să poată fi folosită, existența unei camere web este absolut necesară.

Camera web reprezintă „motorul” acestei aplicații. Așa cum o mașină nu poate înainta dacă motorul ei este oprit, tot așa nici această aplicație nu poate fi folosită în cazul în care calculatorul nu este dotat cu o cameră web.

Rolul camerei web este acela de a capta imagini cu fața utilizatorului. În urma folosirii acestei camere rezultă o serie de imagini ce urmează a fi prelucrate în scopul descoperirii emoțiilor utilizatorului.

Imaginile produse de camera web trebuie pre-procesate pentru îndeplinirea anumitor ipoteze implicite: re-eșantionare (ne asiguram că sistemul de coordonate al imaginii este corect), îmbunătățirea contrastului pentru a se asigura că informațiile relevante pot fi detectate.

Următorul modul denumit ***Preprocesare*** reprezintă un modul complex, în cadrul căruia sunt îmbinate o serie de funcții din cadrul *OpenCV* împreună cu o serie de algoritmi codați în Python pentru:

* Detecția obiectelor( în acest caz detecția fețelor persoanelor, dar și detecția altor simboluri necesare în recunoașterea emoțiilor)
* Alinierea este localizarea componentelor faciale, cum ar fi ochii, sprâncenele, nasul si buzele.
* Normalizarea geometrică se realizează în cazul în care fața este rotită, decupată, redimensionată sau chiar deformată[14].

În etapa de preprocesare, se reduce la minimum zgomotul imaginii, condițiile de iluminare, sau deformările imaginii datorită lentilelor camerei. Acești pași minimizează erorile atunci când se încearcă să se detecteze obiectele.

Dacă nu se elimină zgomotul, se detectează mai multe obiecte decât ne așteptăm, [12].

După ce aplicăm toate etapele de preprocesare necesare pentru imaginea de intrare, al doilea pas este ***Segmentarea***. În cadrul acestui modul se realizează extragerea regiunilor de interes ale unei imagini și izolarea lor pentru a găsi obiectele de interes. Pentru detectarea feței, trebuie separate fețele de restul pieselor din scenă. După obținerea obiectelor din interiorul imaginii, vom continua cu pasul următor. Avem nevoie de extragerea tuturor caracteristicilor pentru fiecare obiect detectat.

Pentru fiecare obiect, vom extrage diferite caracteristici pe care sistemul trebuie să le analizeze folosind un algoritm de învățare. Cu ajutorul unui algoritm de învățare, suntem capabili de a demonstra etichetele fiecărui obiect detectat în imaginea de intrare. Toate acestea se realizează în cadrul ***Modulului Caracteristic*** în cadrul căruia se realizează cele două operații:

* Extracție
* Selecție

*Pixel data* este convertit într-o reprezentare de nivel superior dacă forma, mișcarea, culoarea, textura, și configurația feței sau a caracteristicilor faciale obținute prin extracție vor putea fi utilizate pentru clasificare ulterioară.

Așadar, avem descriptorul obiectului nostru; un descriptor este o caracteristică care descrie un obiect, acești descriptori folosindu-se pentru instruire. Pentru a face acest lucru, trebuie creat un set mare de date de caracteristici, în cazul în care sunt preprocesate sute, mii și milioane de imagini.

În această lucrare, sistemul de recunoaștere a expresiilor faciale utilizează Haar Cascade Clasifier. ***Haar Cascade*** este un modul complex în cadrul căruia se execută operațiile enumerate mai sus.

După realizarea acestor module și sub module, se va trece la următorul modul în care se va face clasificarea propriu-zisă. Acest modul poartă numele de ***Cascade Training***.

În cadrul acestui modul se face instruirea întregului sistem, sistemul devine capabil de a învăța parametrii necesari pentru a decide care etichetă specifică ar trebui să fie atribuită obiectului detectat.

În această lucrare, sistemul de recunoaștere a expresiilor faciale utilizează *Haar Cascade Clasifier. Haar Cascade* este un modul complex în cadrul căruia se execută operațiile enumerate mai sus.

După realizarea acestor module și sub module, se va trece la următorul modul în care se va face clasificarea propriu-zisă. Acest modul poartă numele de ***Cascade Training.***

În cadrul acestui modul se face instruirea întregului sistem, sistemul devine capabil de a învăța parametrii necesari pentru a decide care etichetă specifică ar trebui să fie atribuită obiectului detectat.

Pentru fiecare obiect, se extrag diferitele caracteristici pe care sistemul trebuie să le analizeze folosind un algoritm de învățare.

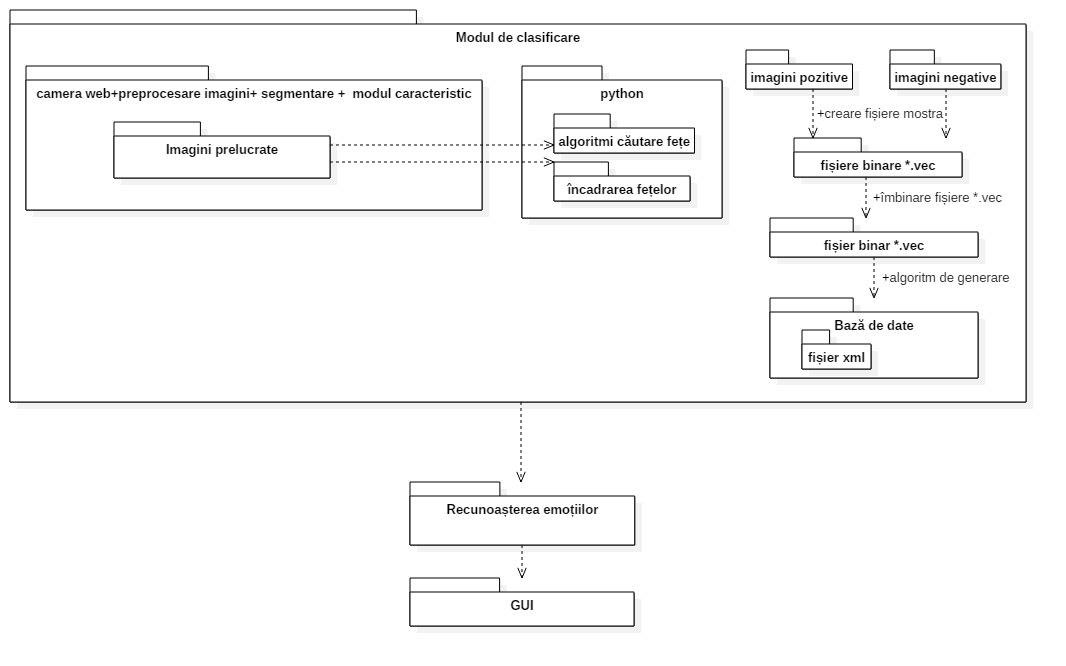
Se colectează un set de imagini pozitive și negative(în cadrul modulului ***Colectare Imagini***). *Opencv\_createsamples* produce un set de date( probe pozitive) într-un format care este acceptat atât de *opencv\_haartraining* și *opencv\_traincascade(*modul ***Creare Mostre****)*. Ieșirea este un fișier cu extensia *\* .vec(* modul ***Creare Fișiere binare****)*, acesta este un format binar care conține imagini o*pencv\_performance* pentru evaluarea calității clasificatorilor, dar numai antrenat de *opencv\_haartraining*. Pentru întocmirea raportului de performanță  avem nevoie de o colecție de imagini, de numărul de obiecte găsite( modul ***Îmbinare fișiere***), numărul de obiecte pierdute, numărul de alarme false și alte informații[13].

În modulul ***Generarea fișierelor XML*** avem o bază de date care conține mai mulți parametrii ai datelor inițiale. Acești parametrii sunt puși în fișiere cu extensia *xml*.

Informațiile din această bază de date sunt folosite în modulele denumite ***Creare funcții*** ***Python*** și ***Creare Funcții OpenCV***. În cadrul acestui modul sunt folosite funcții din celelalte module , iar rezultatele finale ajung în modulul Sistem de recunoaștere a stărilor emoționale.

Am adus o contribuție în cadrul tuturor modulelor, în special în cadrul ultimelor module.

După prezentarea schemei generale a sistemului, urmează prezentarea schemei modulelor realizate de mine:



*Fig 3.1.2 Aplicație recunoaștere stări emoționale*

Așa cum reiese și din schema de mai sus(Fig 3.1.2) părțile componente ale modului ***Cascade Training*** sunt: colecție imagini, creare mostre, mapare fișiere binare, îmbinare fișiere binare precum și generarea fișierelor xml.

## 

## ***3.2 Funcțiile sistemului***

Modulele care prezintă o mai mare atenție vor fi prezentate. Ele sunt împărțite în sub module (amintite în paragraful de mai sus). În continuare sunt expuse sub modulele celor mai complexe module.

Primul element este reprezentat de **imaginile** capturate de camera web, care reprezintă intrările în sistem, sunt considerate intrări deoarece acestea reprezintă punctul de plecare şi prin prelucrarea acestora, precum şi prin adăugarea altor module o să poată fi realizate ieșirile/rezultatul, rezultat pe care la început am presupus ca îl voi obține.

Aceste imagini sunt supuse mai multor operații: preprocesare, segmentare, extracție și selecție pentru a ne asigura că imaginile nu sunt deformate, decupate și dacă conțin obiectul căutat.

După toate aceste operații, am folosit niște algoritmi de învățare, acești algoritmi fiind o altă componentă importantă a aceste aplicații. Prin care calculatorul este instruit în căutarea și găsirea anumitor obiecte( în cazul de față este vorba în primul rând de față iar mai apoi de restul elementelor care au un rol important în detecția stărilor emoționale).

Operațiile de preprocesare, segmentare, extracție și selecție sunt realizate în modulul ***Haar Cascade*,** un modul de bază din cadrul *OpenCV*.

Un alt modul important este ***Cascade Training*** asupra că voi insista pentru faptul că este unul foarte complex care conține la rândul său multe alte elemente sau sub module.

Pentru că am încercat să creez propriul clasificator am avut nevoie de mostre, ceea ce înseamnă că am avut nevoie și de o mulțime de imagini care conțin obiectul dorit (probă pozitivă) și mai multe imagini care nu conțin obiectul dorit (proba negativă). De câte imagini am avut nevoie? Și de câte imagini este nevoie pentru crearea unui astfel de clasificator indiferent de obiectul dorit? Numerele depind de o grămadă de factori, inclusiv de calitatea imaginilor, de obiectul dorit de recunoscut, de metoda de generare probelor precum și de puterea procesorului. Pentru crearea unui clasificator foarte precis este nevoie de mult timp și un număr foarte mare de probe. Clasificatorii făcuți pentru recunoașterea feței au fost creați de către cercetători cu mii de imagini pozitive.

După ce am colectat destule imagini, le-am decupat astfel încât numai obiectul nostru dorit să fie vizibil.  Cele mai bune rezultate provin din ***imagini pozitive***, care arată exact ca cele pe care le-am dori să le detecteze obiectul, cu excepția faptului că acestea sunt decupate astfel încât numai obiectul este vizibil.

După aceste operații, am colectat și ***imaginile negative***. Pentru combinarea imaginilor pozitive și negative, am scris niște funcții în c++ despre care voi da mai multe detalii în capitolul de proiectare în detaliu.

Asta înseamnă că fișierele cu extensia \*.txt servesc ca o listă de eșantioane negative. Am folosit *OpenCV*: *opencv\_createsamples*. Acest program oferă mai multe opțiuni pentru generarea de eșantioane formate din imagini de intrare și scoate la ieșire un fișier cu extensia *\*.vec*, fișier pe care îl voi folosi la instruirea clasificatorului.

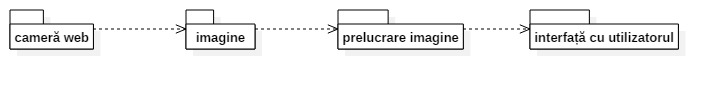
Am continuat cu generarea fișierului ***xml*** care reprezintă o bază de date pentru această aplicație din cauza faptului că el conține o serie de caracteristici folosite pentru compararea caracteristicilor din setul mare de poze cu poza realizată cu camera web.

Acest fișier/fișiere ***.xml***( pentru că pot fi mai multe în funcție de numărul obiectelor pe care dorim să le detectăm) este importat într-un fișier cu extensia ***\*.py***

## ***3.3 Interfața cu utilizatorul***

Cunoscând intrările și ieșirile din sistem, se poate proiecta o schemă bloc a interfeței cu utilizatorul. Așadar intrarea în blocul de interfață va fi reprezentat de capturile realizate de camera web. În momentul în care utilizatorul va rula fișierul cu extensia *.py*, se va porni camera web iar mai apoi se va deschide o fereastră în cadrul căruia se va realiza captura. Interfața are niște butoane cu ajutorul cărora se poate mări sau micșora imaginea, sau se poate închide. În poza realizată de cameră va apărea un dreptunghi care marchează fața, iar dacă utilizatorul este fericit, va mai apărea un dreptunghi care va marca zâmbetul și se va afișa mesajul „happy”, iar dacă utilizatorul nu este fericit, nu va mai apărea dreptunghiul care marchează zâmbetul și va apărea mesajul “not happy”.

Schema bloc pentru interfață este următoarea:



*Fig 3.3.1 Schema bloc pentru interfață*

Interfața cu utilizatorul este una foarte simplă, pentru ca orice utilizator care nu are cunoștințe legate de programele în care s-a realizat aceasta să poată să o folosească cu un minim de cunoștințe, desigur voi prezenta și un manual de utilizare, ca să nu existe dubii legate de folosirea interfeței, iar imaginile care vor trebui modificate să poată fi accesate și vizualizate, totul pentru o bună desfășurare a lucrurilor. Manualul de utilizare precum și alte explicații, capturi cu interfața vor fi prezentate într-un alt capitol.

Pentru dezvoltarea interfeței cu utilizatorul folosesc niște funcții pe care le-am scris în *Python*. Pentru crearea acestor funcții m-am folosit de o serie de biblioteci existente deja în *Python*, pe care le-am importat. Utilizatorul trebuie să facă un singur lucru, să ruleze un fișier, iar fereastra se va deschide automat, fără ca acesta să mai dea vreun click. În această fereastră vor fi prezentate exact rezultatele pe care le dorește, fără a fi nevoie ca acesta să mai deschidă alte ferestre. După vizionarea rezultatelor, fereastra se închide ușor, apăsând butonul x din dreapta sus sau prin apăsarea combinaților de taste ctrl+q.

## ***3.4 Structuri de date și fișiere***

### ***3.4.1 Fișiere***

Lucrul cu clasificatori cascadă include două etape majore: de **formare** și de **detecție**. Etapa de detecție este descrisă în documentația modulului din cadrul *OpenCV*, de care am amintit în capitolele precedente.  Pentru învățarea clasificatorului cascadă, în *OpenCV* există două aplicații: *opencv\_haartraining* and *opencv\_traincascade*. În etapa de formare avem nevoie de un set de probe pozitive și negative. Acestea se folosesc pentru crearea fișierelor finale, care au extensia .*xml*. Pentru a genera acesta fișiere am avut nevoie de alte fișiere cu diferite extensii pe care le-am obținut din probele pozitive și negative:

* Fișiere cu extensia *.txt*
* Fișiere binare cu extensia *.vec*

După colectarea imaginilor pozitive, acestea se decupează pentru a avea aceeași lungime și aceeași lățime și se plasează în directorul *positive\_images*, imaginile având de asemenea și același format și nume *.jpg*.

Pentru a genera fișierul cu același nume ca și directorul și cu extensia .txt, am rulat scriptul *positives.sh*.

Acest fișier conține numele directorului de bază concatenat cu numele fiecărei poze în parte; astfel, numărul de linii al acestui fișier este egal cu numărul de poze pozitive care se găsesc în directorul *positive\_images*.

Același procedeu l-am folosit și pentru a genera fișierul cu mostre negative, cu aceeași extensie *.text.* Acest fișier se generează prin rularea scriptului *negatives.sh* conține informații pentru seturile de imagini negative.

Mai jos este expus modelul de fișier cu extensia *.text* pentru imaginile pozitive folosite în cadrul acestei aplicații.

./positive\_images/0001.jpg

./positive\_images/0002.jpg

./positive\_images/0022.jpg

...

Fișierul cu extensia *.text* pentru imaginile pozitive arată ca în exemplul de mai jos:

./negative\_images/0001.jpg

./negative\_images/0002.jpg

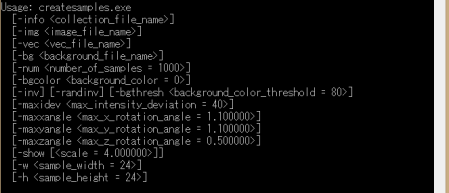
./negative\_images/0003.jpg

…

După generarea fișierelor cu extensia *.txt*, am generat fișiere binare cu extensia \**.vec*.

Prin rularea scriptului *make\_samples.sh* se generează din imaginile pozitive și negative în directorul *samples* un fișier binar cu extensia \**.vec*.

Comenzile folosite pentru rularea scriptului dar și exemple care conțin fișiere \*.*vec* sunt prezentate Fig 3.4.1.1.



*Fig 3.4.1.1 Comenzi și argumente pentru generare* fișiere \**.vec*

Argumentele din *Fig 3.4.1.1*, dar și semnificația acestora, vor fi atașate la finalul lucrării, în cadrul unei anexe.

După rularea scriptului *make\_samples.sh* și generarea în directorul *samples* a fișierelor \*.vec, urmează unirea tuturor fișierelor *\*.vec* într-unul singur, acțiune care se realizează prin rularea scriptului *mergevec\_py\mergevec.bat*. Rezultatul se copiază în directorul rădăcină.

Pentru generarea fișierelor cu extensia .*xml*( fișiere utilizate în cadrul modulelor ***funcții Python*** și ***funcții OpenCV*** pentru detectarea obiectelor) se ruleazăscriptul *generate\_cascade.sh* .

În *Fig 3.4.1.2* este prezentat un exemplu de fișier cu extensia *.xml* (o parte a fișierului, pentru că un fișier întreg poate să conțină zeci de mii de linii).



*Fig 3.4.1.2 Fișier XML pentru detectarea diverselor obiecte*

# **4. Proiectarea de detaliu**

## ***4.1 Arhitectura programului***

În cadrul unei arhitecturi software este descrisă structura unui sistem de programe la nivel abstract. Arhitectura programului meu, constă dintr-un set de componente, conectori şi configurații care sunt caracterizate de un anumit șablon(tip arhitectural). Tipul arhitectural trebuie ales în așa fel încât să fie unul potrivit, pentru ca arhitectura să fie în conformitate cu specificațiile. Trebuie așadar găsită o metodă prin care un astfel de program complex să poată fi evaluat din punct de vedere al fiabilității. Această metodă oferă un mijloc prin care proiectanții să poată proiecta arhitectura care se potrivește cel mai bine cu specificațiile aplicației. În același mod am procedat şi eu, atunci când am creat arhitectura aplicației mele, am ținut cont de specificațiile primite de la client.

În cadrul schemei prezentate în Fig 3.1.2 sunt ilustrate componentele aplicației mele, precum şi conectorii. Voi descrie detaliat fiecare componentă în parte din cadrul acestei scheme, explicând relația dintre ele, precum şi modul în care au fost create.

Pentru crearea acestor componente am folosit ca şi medii de dezvoltare următoarele medii: *Notepad++, OpenCV,* iar limbajul pe care l-am folosit este *Python(* împreună cu o serie de biblioteci și funcții din OpenCV).

## ***4.2 Descrierea componentelor***

De-a lungul timpului au existat numeroase tipuri de abordări, ca metode, tehnici şi algoritmi, pentru găsirea unor soluții optime, fiabile, performante şi eficiente, de tratare a problemei detectării figurilor umane în imagini (digitale). Primul sistem de detectare a fețelor a fost dezvoltat în anii '70 . Dezvoltarea tehnicilor de recunoaștere a fețelor a făcut necesară şi dezvoltarea unor algoritmi mai performanți pentru detectarea acestora, ca prim pas în recunoașterea automată[16].

Așa cum am mai menționat și în alte capitole, detectarea automată a fețelor este o problemă dificilă, practic, imposibil de definit, fiind o varietate de moduri în care pot apărea figurile umane în imagini. Şi asta nu doar din cauza trăsăturilor şi particularităților fizionomice, cât mai ales din cauza varietății în percepția lor în 2-D, datorate poziției (orientării / înclinării, scalării) .

Putem enumera următorii factori principali care pot induce Problemele cu care m-am luptat în crearea acestei aplicații au avut ca principali factori:

* + poziția şi orientarea obiectelor în imagine (frontal, profil, sub un unghi etc.) - anumite caracteristici faciale (ochi, nas) putând fi parțial sau total ascunse;
  + prezența / absența unor componente structurale - unele caracteristici faciale precum barbă, mustața, ochelari putând fi, sau nu, prezente şi existând o mare variabilitate a acestora din punct de vedere al formei, culorii sau dimensiunilor;
  + expresia facială - geometria feței fiind afectată de aceasta;
  + obturarea - fețelor putând fi parțial mascate (acoperite) de alte obiecte (inclusiv alte fețe);
  + condițiile în care a fost realizată fotografia - iluminarea (spectrul, poziția şi/sau distribuția sursei / surselor de lumină, intensitatea) şi caracteristicile aparatului foto (lentilele, senzorul) afectând foarte puternic felul în care o figură apare în imagine[16].

Metodele utilizate, indiferent de tipul acestora, presupun numeroase iterații, analize pe diferite criterii, scalări, filtrări, comparații. Aceste sisteme se bazează pe antrenări anterioare ale unor clasificatoare (în general cu două clase: față şi non-față) de diferite tipuri, utilizând rețele neuronale, *support vector machines* (SVM), modele ascunse (*hidden*) Markov etc.

Așa cum am mai menționat, algoritmii de detecție ai fețelor pot fi împărțiți în patru mari categorii:

* Metoda bazată pe cunoștințe
* Abordarea bazată pe caracteristici invariante
* Metoda bazată pe modele / șabloane (templates)
* Metoda bazată pe apariție / înfățișare (appearence)

În cadrul aplicației mele *detecția* este una dintre cele mai importante componente. Detecția obiectului se realizează în etapa de preprocesare a imaginilor, care face parte din modulul de Haar Cascade. Biblioteca *OpencCV* vine cu o serie de module în cadrul cărora sunt implementați o serie de algoritmi. Algoritmul care stă în spatele modului de *OpenCV* este cunoscut drept metoda Viola-Jones şi este unul dintre cei mai robuști, performanți. Acest algoritm nu presupune analizarea imaginii în sine, ci doar anumite „caracteristici” (*features*) dreptunghiulare în aceasta. Aceste caracteristici, sunt cunoscute sub denumirea de „caracteristici de tip Haar”.

În practică se poate utiliza pentru fiecare pixel (x, y) al imaginii o relație de forma:

i(x, y) = 0,299 R(x, y) + 0,587 G(x, y) + 0,114 B(x, y),

unde valorile R, G, B reprezintă respectiv componentele de roșu, verde şi albastru ale valorii respectivului pixel (x, y) în spațiul RGB utilizat frecvent în reprezentarea digitală a imaginilor color, pe 3 sau 4 octeți (24 sau 32 de biți). Valorile sunt cuprinse între 0 şi 255 (cât poate fi reprezentat pe un octet).

Pe imaginea în 256 de nivele de gri astfel obținută pot fi analizate anumite caracteristici dreptunghiulare prin adunarea valorilor intensităților pixelilor în diferite blocuri dreptunghiulare. Astfel, pot fi detectate în imagine caracteristici formate din blocuri mai întunecate, adiacente unor blocuri mai luminoase, suma pixelilor din primele fiind mai mică decât cea a pixelilor din cele din urmă. Trebuie precizat faptul că aceste caracteristici nu reprezintă totuși în niciun fel anumite caracteristici / trăsături faciale specifice. Însă pentru îmbunătățirea vitezei, în spatele modulelor OpenCV stă următorul algoritm: imaginea originală de analizat se generează inițial o așa numită „imagine integrală”, mapată 1 la 1 peste imaginea originală, în care fiecare punct capătă valoarea sumei tuturor pixelilor din imaginea originală situați la stânga şi deasupra coordonatelor punctului respectiv, inclusiv.

Valorile tuturor punctelor ii(x, y) din imaginea integrală pot fi calculate cu o singură trecere prin imaginea inițială, pornind din colțul din stânga-sus (x = 0 şi y = 0), pixel cu pixel, linie după linie[16].

Componenta de *normalizare* după cum îi spune și numele realizează normalizarea în cazul în care fața este rotită, decupată, redimensionată sau chiar deformată[14].

În cadrul componentei denumită *aliniere* se realizează localizarea componentelor faciale, cum ar fi ochii, sprâncenele, nasul si buzele.

După primul modul denumit *Preprocesare* urmează modulul de *Segmentare*

# **5. Utilizarea sistemului**

Orice program, fie el *software* sau de altă natură, trebuie sa aibă un manual de utilizare care să fie ca un ghid pentru cel ce-l răsfoiește și urmează să folosească produsul respectiv. În acest manual trebuie să fie precizate instrucțiuni referitoare la punerea în funcțiune și folosirea programului sau aparatului.

Dacă ar fi să fac un scurt ghid utilizare al aplicației mele, pe care utilizatorul ar fi recomandat să-l parcurgă înainte de folosire, acesta ar conține următoarele capitole:

## ***5.1 Scurtă prezentare***

Prezentul Ghid de Utilizare se referă la “Aplicație software de recunoaștere a stărilor emoționale bazate pe trăsăturile facile”, și se recomandă parcurgerea lui înainte de utilizarea aplicației.

Aplicația este folosită pentru a detecta o stare emoțională în funcție de trăsăturile feței, mai exact, în funcție de față, de poziția și forma gurii, nasului și gurii. Datorită acestei aplicații, datorită acestei aplicații, starea acestuia poate fi detectată.

Aceasta aplicație poate fi așadar folosită gratuit, iar prin simpla generare a unui nou fișier xml, utilizatorul poate detecta orice obiect dorește, aplicația putând fi folosită în extreme de multe scopuri.

Pentru ca aplicația să poată fi utilizată este necesară instalarea pe calculatorul fiecărui utilizator a unor programe.

## ***5.2 Instalare***

Fiecare utilizator va trebuie să instaleze următoarele programe:

- mediul de dezvoltare Eclipse *Kepler* împreună cu *plugin*-ul *PyDev* pentru a putea vedea codul scris și pentru a adăuga în script-ul de *Python* calea spre fișierul pe care dorește să-l convertească, dar și pentru a rula aplicația.

-mediul Qt 5.1 pentru a putea vizualiza fișierele convertite.

## ***5.3 Utilizare***

Odată instalate programele de care este atâta nevoie, este timpul ca aplicația să fie utilizată în scopul pentru care a fost creată. Așadar fiecare utilizator după ce a instalat aceste programe și după ce a copiat pe calculatorul personal *folder*-ul care conține aplicația trebuie să efectueze următorii pași:

− Să deschidă mediul de dezvoltare Eclipse și din meniu să aleagă File->Import, prin selectarea acestei opțiuni proiectul realizat va fi adus din locul în care este salvat în acest mediu pentru a putea fi vizualizat conținutul fișierului care s-a creat în momentul în care s-a creat aplicația. Utilizatorul trebuie să fie atent în momentul în care trebuie să introducă calea spre proiectul salvat. Așadar, după introducerea corectă a căii, trebuie să selecteze proiectul, acest lucru este necesar, deoarece există probabilitatea ca să mai existe mai multe proiecte care pot fi aduse din locul unde este și proiectul necesar a fi importat. După aceasta urmează ca utilizatorul să dea *click* pe butonul *Next,* pentru a putea fi important proiectul deja existent.

În momentul acesta, proiectul este adus din locul selectat, iar pe *desktop* ar trebui să vadă următoarea fereastră:

*Fig 5.3.1 Mediul de dezvoltare Eclipse*

Se poate observa faptul că proiectul importat conține mai multe fișiere. Utilizatorul nu are dreptul de a modifica aceste fișiere care conțin cod de *Python*. Trebuie să deschidă fișierul cu denumirea *src*, care conține mai multe fișiere dintre care unul cu numele *templates* (acesta trebuie deschis). În momentul în care acest fișier este deschis se poate observa că el conține mai multe *script*-uri de *Python,* dintre acestea trebuie deschis *script*-ul cu numele *pxa.py.* În cadrul acestuia, singurul lucru care trebuie să-l facă este acela de a modifica calea spre fișierul care dorește să fie convertit pentru a putea fi vizualizat în final.

În codul din acest *script* o să trebuiască adăugată calea spre fișierul care urmează să fie convertit. În Fig 5.3.2 este specificat printr-un chenar roșu exact linia de cod care trebuie modificată, desigur dacă utilizatorul va dori să poată converti mai târziu și alte fișiere, poate să introducă și alte căi, însă în linia pentru convertire trebuie introdusă numele unei singure variabile în care s-a salvat calea spre un anumit fișier. Iar în momentul rulării va avea loc conversia propriu zisă, iar rezultatul va fi afișat așa cum apare în Fig 5.3.3

*Fig 5.3.2 Introducerea căii fișierului*

Mai jos va fi expus rezultatul obținut în momentul în care are loc conversia.

*Fig 5.3.3 Rezultatul conversiei*

Acest rezultat nu ar trebui să sperie utilizatorul, aceasta este dovada că fișierul pe care el dorea să-l convertească a fost convertit, acesta fiind codul de *qml*, noul cod care va apărea dacă fișierele vor fi deschise doar pentru a vizualiza codul din interior. Codul este de *qml* pentru că fișierele convertite vor avea extensia *qml.* Utilizatorul nu trebuie să înțeleagă acest cod, trebuie doar să fie convins că fișierul pe care va dori să-l vadă a fost convertit și în scurt timp va putea chiar să vizualizeze conținutul acelui fișier.

După conversia fișierului, utilizatorul trebuie să mai facă un ultim pas și anume:

− Trebuie deschis *Qml Viewer* care se poate găsi într-un singur mod și anume:

− Se urmează calea: Qt->5.1.1->mingw48\_32->bin->qml Viewer.

− Se deschide fereastra de Qml Viewer care trebuie să arate ca în Fig 5.3.4.

*Fig 5.3.4 Fereastra Qml Viewer*

Pentru a putea vizualiza imaginile din cadrul fișierului tocmai convertit, este necesar ca utilizatorul să acceseze *File*->*Open* și să adauge calea spre fișierul convertit. Fișierele convertite se află în: fișierul *PXAPlayer*, mai exact în locația: *PXAPlayer->pxa->assets->applications*, selectându-se și unul dintre modulele existente, în cadrul cărora se află fișierele convertite.

În urma acestei acțiuni o să apară o interfață foarte simplă cu utilizatorul în cadrul căreia o să apară conținutul fișierului (imaginea mult așteptată).

Rezultatul pe care îl va vedea în final utilizatorul, va fi cel așteptat, iar în Fig 5.3.5 va fi prezentat rezultatul unui fișier *pxa*, transformat în *qml*.

*Fig 5.3.5 Interfața cu utilizatorul*

Utilizatorul poate mări sau micșora imaginea, după cum dorește și poate reveni oricând la varianta inițială a imaginii. Un exemplu de mărire a imaginii este prezentat în Fig 5.3.6.

Fig 5.3.6 Mărirea imaginii

# ***6. Realizarea, punerea în funcțiune și rezultate experimentale***

Momentul în care are loc punerea în funcțiune precum și aflarea rezultatelor experimentale, reprezintă un moment important dar și tensionat în orice domeniu. În acest capitol o să prezint punerea în funcțiune a aplicației mele pentru a urmări și testa funcționalitatea sistemului prin date experimentale.

Voi lua un exemplu concret și anume: în momentul în care se dorește să se afișeze pe *display* o anumită imagine sau un grup de imagini, sunt foarte importante specificațiile, din acest motiv fără specificațiile grafice nu se poate face nimic, chiar dacă sunt oferite toate detaliile și condițiile care trebuie să fie îndeplinite în momentul în care trebuie afișat pe display, fără imaginile care trebuie să fie afișate nu se poate face nimic. Așa cum am mai spus imaginile trebuie prelucrate și trebuie făcute niște legături pentru ca ele să apară pe *display*.

Fișierele primite de la client, pe care le-am convertit sunt împărțite în mai multe module. În mod normal, în bordul unei mașini, sunt afișate multe imagini, grupuri de imagini, poate chiar text sau filmulețe, depinde de mașină. Noi însă trebuie să respectăm întru-totul cerințele impuse de client.

Aceste module sunt: alert, drive, gauge, gearbox, generic, hmi, navigation, obc, onoff, park, settings.

Fiecare modul în parte are o anumită specificație, dar indiferent de specificația acestuia, trebuie testată funcționalitatea pentru fiecare modul în parte.

Daca se dorește ca pe *display* să fie afișată o poză din modulul *alert* prin care să se semnalizeze faptul că o ușă este deschisă, trebuie făcute o grămadă de lucruri și trebuie scris foarte mult cod ca să se ajungă aici. Unul dintre primii pași care trebuie făcuți este acela de a vedea ce poză din cadrul acestui modul este cea potrivită și care îndeplinește anumite condiții. Pentru asta, în primă fază se convertesc fișierele din cadrul modului *alert*, după care, ar trebui să se verifice fiecare fișier convertit pentru a se vedea ce poză conține. Trebuie verificat dacă poza este cea potrivită însă fișierele au nume sugestive, din această cauză se poate observa cu ușurință care ar trebui să fie fișierul care ar trebui să conțină o poză.

Pentru a vizualiza fișierul deschidem *Qml Viewer,* și cu ajutorul File->Open deschidem fișierul cu numele sugestiv *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants* și vom observa faptul ca apare poza din Fig 6.1.

Se mai poate observa și faptul că poza poate fi mărită, micșorată sau se poate reveni la starea inițială în cazul în care se dorește acest lucru. În Fig 6.2, Fig 6.3 se observă faptul că sunt posibile și aceste opțiuni menționate.

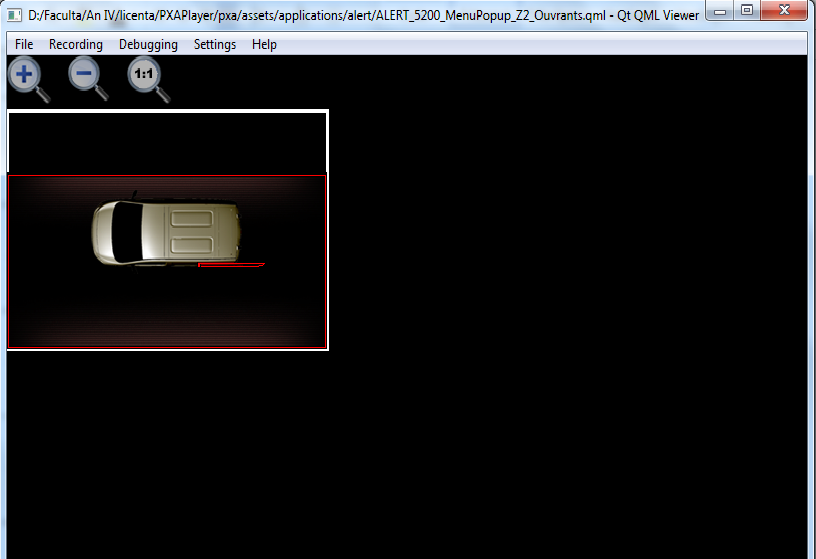


Fig 6.1 Fișierul *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants*

Fig 6.2 Fișierul *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants*(*rezultat mărit)*

Fig 6.3 Fișierul *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants*(rezultat micșorat)

# **7. Concluzii**

În această lucrare s-a urmărit implementarea unui convertor pentru limbajul descriptiv *HMI.* Acesta aplicație realizată de mine este o parte a unui proiect din cadrul companiei Yazaki Component Technology.

Am ales să implementez și desigur am implementat acest convertor, pentru că nu există unul, iar fișierele primite de la client nu puteau fi vizualizate, acest fapt reprezentând o piedică în calea inginerilor de HMI care au ca sarcină realizarea părții grafice. Sarcina acestora a fost ușurată pentru că de acum înainte, nu vor fi nevoiți să piardă timpul pentru a căuta soluții greu de găsit, vor putea vizualiza conținutul fișierelor (pozele din interior) și le vor putea modifica și prelucra în programul folosit pentru acest lucru, numit *Populus*. Imaginile prelucrate ajung ulterior pe *LCD*-ul din cadrul unui bord de mașină.

Modulele implementate sunt toate funcționale, aplicația întreagă este funcțională, aceasta fiind testată de inginerii de *HMI* de nenumărate ori, chiar din momentul finalizării aplicației. Nu au apărut probleme în timpul utilizării. Modul de utilizare nu a ridicat probleme nimănui, aceasta datorându-se interfeței care este una simplă și prietenoasă cu utilizatorul.

Deși am realizat această aplicației pentru un anumit proiect, în cadrul căruia a fost neapărată nevoie de ea, aceasta poate fi folosită și după terminarea acestuia, în cadrul altui proiect. Oricare ar fi limbajul descriptiv din viitoarele proiecte, bazat pe *Xml* și concepte asemănătoare *Qml*-ului convertorul poate fi folosit de fiecare dată prin simpla modificare a regulilor dicționarelor.

Unele dintre cele mai importante proprietăți ale unei aplicații software sunt: flexibilitatea, reutilizabilitatea și faptul dacă aceasta este generică sau nu. Aplicația realizată de mine este generică, flexibilă deoarece prin simple modificări făcute în cadrul acesteia devine funcțională pentru orice proiect și reutilizabilă deoarece se poate reutiliza de fiecare dată.

# **8.Bibliografie**

[1]<http://python.net/~gherman/propaganda/tut-ro/tut-ro.pdf>

[2]<http://www.w3schools.com/xml/xml_whatis.asp>

[3] <http://vechi.upg-ploiesti.ro/col/ldumitrascu/pdf/xml/XML.pdf>

[4]<http://www.slideshare.net/dragosacostachioaie/mediul-de-programare-multiplatforma-qt>

[5] <http://ro.wikipedia.org/wiki/Structur%C4%83_de_date>

[6] <http://qt-project.org/doc/>

[7]<http://qt-project.org/doc/qt-4.8/qdeclarativeintroduction.html>

[8] <http://zetcode.com/gui/pyqt4/introduction/>

[9] <https://en.wikipedia.org/wiki/Affective_computing>

[10] Affective Computing by [Rosalind W. Picard](http://www.goodreads.com/author/show/802927.Rosalind_W_Picard)

[11] <http://www.journals.elsevier.com/computer-vision-and-image-understanding/>

[12] OpenCV by Example

[13] <http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html>

[14] <http://portal.fke.utm.my/fkelibrary/files/chaijialiang/2015/1300_CHAIJIALIANG2015.pdf>

[15] <http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html>

[16] <http://rria.ici.ro/ria2013_2/art03.pdf>