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

***Sistem minimal de autentificare facială***

# Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de
*Inginer* în domeniul *Inginerie electronică și telecomunicații*
program de studii de licență *Electronică aplicată*

Conducători științifici
*Ș.l. Dr. Ing. Anamaria RĂDOI*
*Prof. Dr. Ing. Corneliu BURILEANU*

Absolvent
*Sorin-Valentin GEANĂ*

2017

Anexa 1

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
**Departamentul * Electronică Aplicată și Ingineria Informației**

**Aprobat Director de Departament *:**
***Prof. Dr. Ing. Sever PAȘCA***

**TEMA PROIECTULUI DE DIPLOMĂ**
**a studentului GEANĂ C. Sorin-Valentin 441B**

**1.** Titlul temei: **Sistem minimal de autentificare facială**

**2.** Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare): *se vor descrie în 15..20 rînduri următoarele elemente:*
*Proiectul de diplomă va consta în realizarea unui sistem de autentificare prin recunoaștere facială, ce ar putea fi folosit în incintele unor companii pentru a permite accesul persoanelor autorizate. Sistemul va consta în două module și anume, unul de detecție a feței și unul de recunoaștere a feței. În acest sens, vor fi testați mai mulți algoritmi din literatură (Eigenfaces, Fisherfaces, LBPH) și se va folosi algorimul care obține cel mai bun scor de recunoaștere pe mai multe baze de date publice. În urma măsurătorilor efectuate, cel mai bun algoritm va fi portat pe o platformă mobilă echipată cu o cameră foto. Se vor efectua teste pentru a valida implementarea algoritmului pe platforma mobilă în mai multe scenarii. Antrenarea algoritmului de recunoaștere se va face pe o serie de imagini preluate înainte de a porta algoritmul pe platforma mobilă. Setul de antrenare va conține mai multe persoane care pot avea acces în incinta companiei.*
*Implementările vor folosi librării OpenCV ce vor fi modificate de student pentru a fi portate pe platforma mobilă, dar și pentru a îmbunătăți rata de recunoaștere facială. Ca platformă mobilă, va fi folosit un robot KUKA pe care va fi fixată senzorul de captare a imaginilor faciale.*

**3.** Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline: Microcontrolere, Prelucrarea digitală a semnalelor, Robotică, Imagistică medicală.

**4.** Proprietatea intelectuală asupra proiectului aparţine: *UPB*

**5.** Locul de desfăşurare a activităţii: *UPB*

**6.** Data eliberării temei: 18.10.2016

**CONDUCĂTOR LUCRARE:**

*S.l. Dr. Ing. Anamaria RĂDOI*
*Prof. Dr. Ing. Corneliu BURILEANU*

**STUDENT:**

*Sorin-Valentin GEANĂ*

Anexa 5

# Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Sistem minimal de autentificare facială*", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Inginerie electronică și telecomunicații*, programul de studii *Electronică aplicată* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

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

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

București, *21.06.2017*

Absolvent *Sorin-Valentin GEANĂ*

(semnătura în original)

# Cuprins

Introducere ..... 15
Motivația lucrării ..... 15
Obiectivul lucrării ..... 15
Sumar ..... 16
Capitolul 1. Detecție și recunoaștere facială ..... 17
1.1 Detecție facială ..... 17
1.2 Recunoașterea facială ..... 18
Capitolul 2. Algoritmi de detecție facială ..... 19
2.1 Detecția facială folosind clasificatori de tip cascadă bazați pe caracteristici Haar ..... 19
2.2 Detecția facială folosind clasificatori de tip cascadă bazați pe caracteristici LBP ..... 20
Capitolul 3. Algoritmi de recunoaștere facială ..... 23
3.1 Analiza Componentelor Principale ..... 23
3.1.1 Eigenfaces ..... 24
3.2 Recunoaștere facială bazată pe algoritmul Eigenfaces ..... 26
3.3 Analiza Discriminatorie Liniară (LDA) ..... 32
3.4 Recunoaștere facială bazată pe algoritmul Fisherfaces ..... 34
3.5 Recunoaștere facială bazată pe algoritmul LBPH ..... 36
Capitolul 4. Limbajele de programare și tehnologiile software și hardware folosite în proiect. ..... 39
4.1 Limbajele de programare folosite ..... 39
4.2 Tehnologii software: ..... 39
4.2.1 OpenCV ..... 39
4.2.2 EXtensible Markup Language ..... 39
4.3 Tehnologii hardware: Kuka Youbot și Asus Xtion ..... 40
4.3.1 Arhitectura Software și Hardware a robotului KUKA Youbot ..... 40
Capitolul 5. Rezultate experimentale ..... 45
5.1 Detecție facială ..... 45
5.2 Recunoaștere facială ..... 47
Capitolul 6. Prezentarea aplicației ..... 49
6.1 Funcționarea sistemului ..... 49
Concluzii ..... 53
Bibliografie ..... 55

# Listă figuri

Figura 1.1 Detecție eronată vs. Detecție corectă ..... 17
Figura 2.1 Caracteristici Haar folosite de Viola Jones ..... 19
Figura 2.2 Imagine Integrală ..... 20
Figura 2.3 Imagine Originală ..... 20
Figura 2.4 Proces extragere trăsături LBPH ..... 21
Figura 2.5 Histograma LBP ..... 21
Figura 2.6 Imagine negativă ..... 22
Figura 2.7 Histograma LBP Imagine negativă ..... 22
Figura 3.1 Proiecția punctelor pe prima componentă principală ..... 23
Figura 3.2 Imagine originală ..... 24
Figura 3.3 Reconstrucția imagini ..... 24
Figura 3.4 Transformare imagine de 7x7 într-un vector 49-dimensional ..... 25
Figura 3.5 Setul de date ..... 27
Figura 3.6 „Mean face” ..... 27
Figura 3.7 Imagini normalizate ..... 28
Figura 3.8 Matricea de covarianță pentru setul de date (25 imagini) ..... 29
Figura 3.9 Eigenfaces-urile setului de date ..... 29
Figura 3.10 Imagine de test comparativ cu Imagine din baza de date ..... 30
Figura 3.11 Matricea de covarianță ..... 31
Figura 3.12 Comparație între PCA și FLD ..... 33
Figura 3.13 Medii locale ..... 35
Figura 3.14 „Fisherfaces” ..... 36
Figura 3.15 Binarizare Imagine ..... 37
Figura 4.1 Exemplu XML ..... 40
Figura 4.2 Arhitectura robotului KUKA [11] ..... 40
Figura 4.3 Stările Manipulatorului [11] ..... 42
Figura 4.4 Platformă mobilă [12] ..... 43
Figura 4.5 Structura cinematică a brațului robotic [12] ..... 43
Figura 4.6 Asus Xtion Pro LIVE [13] ..... 44
Figura 5.1 Durata perioadei de detecție Haar Cascade comparativ cu LBP Cascade ..... 45
Figura 5.2 Gradul de suprapunere a ferestrelor de detecție ..... 46
Figura 5.3 Gradul de suprapunere a ferestrelor de detecție în condiții de lumină nefavorabile ..... 46
Figura 5.4 Baza de date proprie ..... 47
Figura 5.5 Rata de recunoaștere ..... 47
Figura 6.1 Organigramă sistem ..... 49
Figura 6.2 Modul detecție față și ochi ..... 50
Figura 6.3 Modul detecție (vedere din aplicație) ..... 51
Figura 6.4 Modul de recunoaștere ..... 51
Figura 6.5 Aplicație de configurare a brațului robotic ..... 52
Figura 6.6 Verificare persoană ..... 52

# Listă tabele

Tabel 5.1. Rezultate LBPH ..... 48
Tabel 5.2. Rezultate Eigenfaces ..... 48
Tabel 5.3. Rezultate Fisherfaces ..... 48

## Lista Acronimelor

| | |
|---|---|
| ANOVA | Analysis of Variance |
| API | Application Programming Interface |
| FERET | The Facial Recognition Technology |
| FLD | Fisher’s linear discriminant |
| GB | GigaByte |
| LBPH | Local Binary Pattern Histogram |
| LDA | Linear Discriminant Analysis |
| PCA | Principal Component Analysis |
| SSD | Solid State Drive |
| VGA | Video Graphics Array |
| XML | EXtensible Markup Language |

# Introducere

## Motivația lucrării

Securitatea a fost dintotdeauna un domeniu vital, fie că este vorba despre securitatea fizică, sau securitatea virtuală, omul a avut nevoie de metode prin care să își protejeze datele computerizate, compania sau propria casă.

Securitatea datelor poate fi obținută prin folosirea parolelor, însă acestea, pot fi "sparte" cu ajutorul unor programe care generează câteva zeci de mii de combinații posibile pe secundă. Datorită acestei probleme cercetătorii au venit cu următoarea soluție, sistemele biometrice de securitate. Sistemele biometrice pot fi sisteme bazate pe amprente, detecție iris sau detecție facială.

Un sistem de recunoaștere faciala este o aplicație computerizată capabilă să detecteze și să verifice identitatea unei persoane dintr-o imagine digitală sau un cadru de la o cameră video. Acest lucru se poate face comparând caracteristicile faciale ale persoanei din imagine cu o baza de date deja existentă.

Aceste aplicații sunt folosite în sisteme de securitate și pot fi văzute ca alte verificări biometrice cum ar fi cele bazate pe amprente sau detecția irisului unei persoane.
Comparativ cu alte tehnici biometrice, recunoașterea facială nu este cea mai eficientă și de încredere metodă, dar are totuși avantajul de a fi o metodă neinvazivă astfel informațiile sunt luate fără contact direct cu subiectul.

Asemenea sisteme amplasate în aeroporturi, locuri publice, corporații private pot identifica indivizi în mulțime fără ca aceștia să fie conștienți de acest lucru. Alte sisteme de verificare biometrice, amprentele, scanarea irisului sau recunoașterea vorbitorului nu pot realiza acest tip de identificare în masă a populației. Există sistem de acest tip chiar și în calculatorul personal, de exemplu Windows 10 are o opțiune de autentificare folosind recunoașterea facială.

Recunoașterea facială este făcută ușor de către oameni. Experimentele realizate de profesorul Anthony Norcia au arătat că încă de la patru luni creierul unui bebeluș procesează fețele aproape la fel de bine ca un adult. Deci dacă un copil poate face asta, cât de greu să fie acest lucru pentru o mașină de calcul?

Recunoașterea facială bazată pe caracteristici geometrice ale feței este cea mai potrivită abordare. Primul sistem automat de recunoaștere facială a fost descris în 1973 de către Kanade Takeo unde afirma că punctele de interes (poziția ochilor, urechilor, nasului) erau folosite pentru a construi un vector caracteristic (distanța dintre punctele de interes, unghiul dintre acestea). Recunoașterea se efectua prin calcularea distanței euclidiene dintre vectorii caracteristici alea unei imagini de test și o imagine de referință. Această metodă este eficientă la schimbarea iluminării, însă dezavantajul principal este că înregistrarea exactă a punctelor de interes este complicată.

## Obiectivul lucrării

Se dorește implementarea unui algoritm de recunoaștere facială pe un sistem cu resurse limitate pus la dispoziție de către laboratorul CAMPUS - Center for Advanced Research on New

Materials, Products and Innovative Processes - Robots – Autonomous and Adaptive Systems Lab. Sistemul este alcătuit dintr-o platformă mobilă ce are încorporată un procesor Intel AtomTM Dual Core D510 (1M cache, 2x1.66 GHz), o memorie RAM de 2GB cu frecvența de lucru de 667MHz și un dispozitiv de stocare SSD de 32GB.

Pe lângă platforma mobilă sistemul are un braț robotic cu 6 grade de libertate și un senzor de mișcare **Asus Xtion Pro LIVE,** care vine cu 2 camere încorporate, o cameră video, capabilă să filmeze la rezoluție de 1280x102, 640x480 și o cameră de adâncime cu rezoluția de 320x240.

## Sumar

În capitolul 1 se va explica conceptul de detecție și recunoaștere facială.

În partea teoretică vor fi explicate mai pe larg începând cu capitolul 2 metodele de detecție bazate pe cascade Haar și cascade LBP  cu principalele avantaje și dezavantaje ale fiecărei metode în parte.

În capitolul 3 vom discuta despre metodele statistice ce stau la baza algoritmilor de recunoaștere facială contemporani și despre principalii algoritmi de recunoaștere facială (cu exemple) ce ne sunt puși la dispoziție de către OpenCV.

În capitolul 4 se va face o prezentare a echipamentului hardware reprezentat de robotul KUKA YouBot, împreună cu aplicația programabilă integrată și un alt periferic, o cameră web de la ASUS.

În capitolul 5 va fi prezentată o serie de experimente în cadrul cărora vor fi testate cele două metode de detecție puse la dispoziție de OpenCV: Haar Cascade și LBP cascade, din punct de vedere al acurateței și gradul de centrare a detectorului pe fața subiectului, apoi vor fi testate comparativ metodele de recunoaștere facială sub diferite condiții (luminozitate redusă, luminozitate ridicată, rotirea imaginii).

În capitolul 6 se va prezenta mai pe larg aplicația și posibile îmbunătățiri ce pot fi aduse acestui proiect într-o versiune viitoare. Apoi o să închei cu capitolul de concluzii.

# Capitolul 1. Detecție și recunoaștere facială

**Detecția facială** este o tehnologie folosită într-o varietate de aplicații folosite la identificarea fețelor în imagini digitale sau cadre provenite de la o cameră video.

Detecția facială poate fi privită ca un caz particular al detecției de obiecte. Algoritmi de detecție facială se concentrează pe imagini cu fețe care privesc înainte, deoarece acelea sunt mai ușor detectabile.

Sistemele de **recunoaștere facială** sunt aplicații capabile să identifice și să verifice identitatea unei persoane dintr-o imagine digitală sau cadre video. Metoda de recunoaștere facială folosită în această lucrare constă în compararea anumitor caracteristici faciale între o imagine de test și o imagine stocată într-o bază de date.

## 1.1 Detecție facială

Pentru a putea face recunoaștere facială într-un mod optim, trebuie mai întâi să facem o detecție facială cât mai precisă. Mai jos avem un exemplu de detecție eronată deoarece gradul de încadrare a feței în dreptunghiul roșu este foarte scăzut (imagine stângă) comparativ cu imaginea din dreapta în dreptunghiul este centrat pe față perfect.

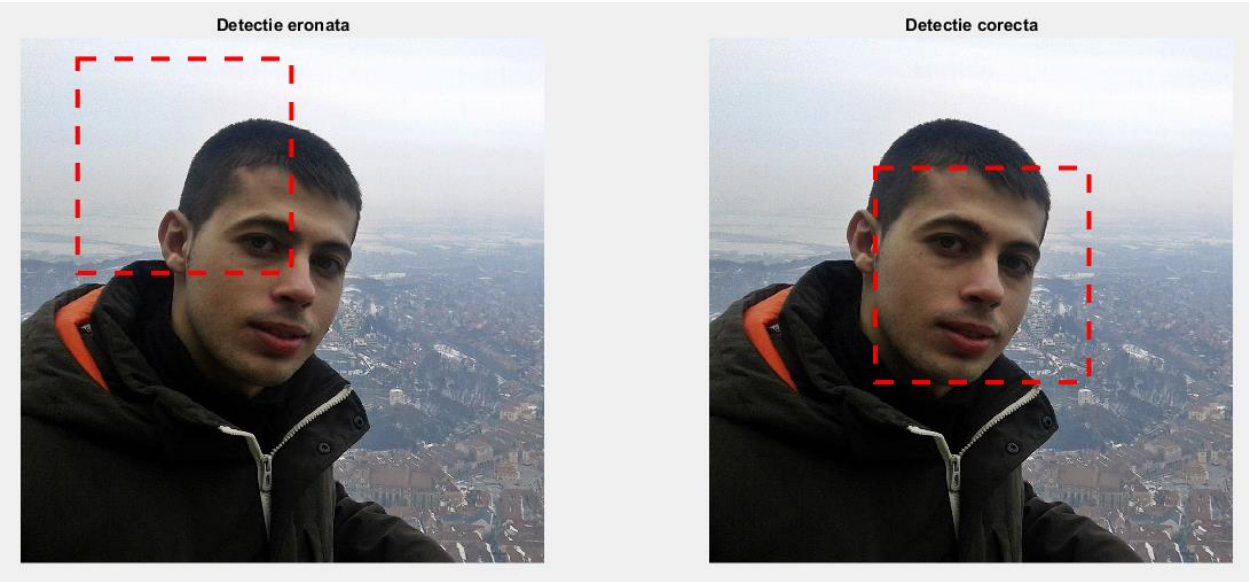



Figura 1.1 Detecție eronată vs. Detecție corectă

Există două probleme majore legate de detecția facială: detecția în imaginile statice și detecția în timp real. Detecția în timp real presupune detecția feței dintr-o serie de cadre preluate de la o cameră video. Deși volumul de calcul este mai mare pentru detecția în timp real, aceasta este mai simplă, decât în imaginile statice, deoarece persoanele sunt într-o continuă mișcare, în timp ce mediul înconjurător este static, astfel folosind metode prin care eliminăm componentele statice din cadrele obținute putem obține o detecție cu un grad ridicat de acuratețe. Astfel folosind aceste etape de preprocesare a imagini detecția se poate face și în medii deschise, necontrolate apriori.

## 1.2 Recunoașterea facială

În momentul actual recunoașterea facială este folosită de Facebook, de exemplu atunci când încarci o poză Facebook îți sugerează care ar putea fi numele persoanei din imagine folosind un sistem inteligent de recunoaștere facială antrenat cu rețele neuronale multistrat. O altă aplicație de divertisment folosită care folosește recunoașterea facială este Snapchat-ul, acea aplicație detectează fața unei persoane și permite diferite modificări ale imagini (schimbare fețe între 2 persoane, adăugare elemente de amuzament).

# Capitolul 2. Algoritmi de detecție facială

## 2.1 Detecția facială folosind clasificatori de tip cascadă bazați pe caracteristici Haar

Cascadele Haar (engl. *Haar Cascade*) reprezintă un algoritm de detecție de obiecte folosit în numeroase aplicații de detecția a persoanelor angajate dintr-o firmă, numerele de înmatriculare ale mașinilor ce circulă pe o autostradă , expresii faciale în imagini și cea mai mare utilizare a acestui algoritm este în detecția facială.

Trăsăturile se obțin astfel, fiecare trăsătura reprezintă o valoare calculată ca diferența dintre suma pixelilor aflați sub dreptunghiul alb și suma pixelilor aflați sub dreptunghiul negru (vezi figura 2.1).

Figura 2.1 Caracteristici Haar folosite de Viola Jones

În prima etapă a algoritmului de mai sus, se încarcă în sistem un set de poze ce reprezintă imagini faciale ale mai multor persoane (diferite poziții ale feței, fundal diferit, lumină mai puternică, lumină mai slabă), care vor fi privite ca imagini pozitive pentru detecția facială și imagini care nu seamănă cu o față de exemplu, imagini cu paturi, scaune, mese, pereți, ceasuri, animale, plante, acestea vor fi privite ca imagini negative pentru detecția facială, iar extragerea de trăsături se face odată cu antrenarea clasificatorului folosind imaginile integrale (figura 2.2) și algoritmul Adaboost. [1]

Imaginile integrale, acest concept a fost introdus în domeniul procesării de imagini pentru prima dată de Paul Viola și Michael Jones în lucrarea "Rapid object detection using a boosted cascade of simple features" [2] în anul 2001. Imaginea integrala de la coordonatele (x,y) conține suma pixelilor situații la stânga și deasupra coordonatelor imaginii curente. Astfel imaginea integrală poate fi reprezentată conform următoarei formule:

$$intX_{(x,y)} = \sum_{\substack{x' \le x \\ y' \le y}} X_{(x',y')}$$

Unde intX reprezintă imaginea integrală și X este imaginea originală. Imaginea integrală reduce volumul de operații efectuate asupra imaginilor, în general în procesarea de imagini este nevoie de extragerea trăsăturilor dintr-o anumite regiune, nu din toată imaginea, numită în literatura de specialitate (ROI – *Region Of Interest*), operație care, folosind imaginea integrală se face mult mai simplu, folosind pentru o singură regiune dreptunghiulară doar patru puncte de referință. [1]

Mai jos se poate observa o imagine integrală și imaginea originală din care a fost reprodusă cea integrală.

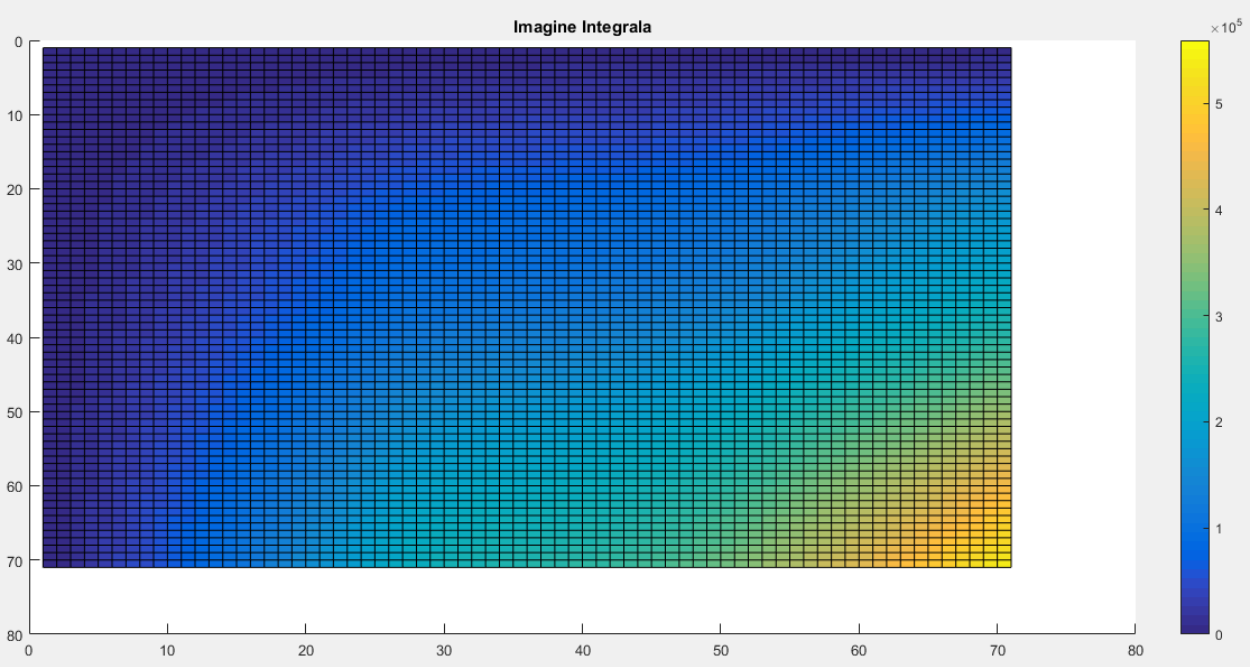



Figura 2.2 Imagine Integrală

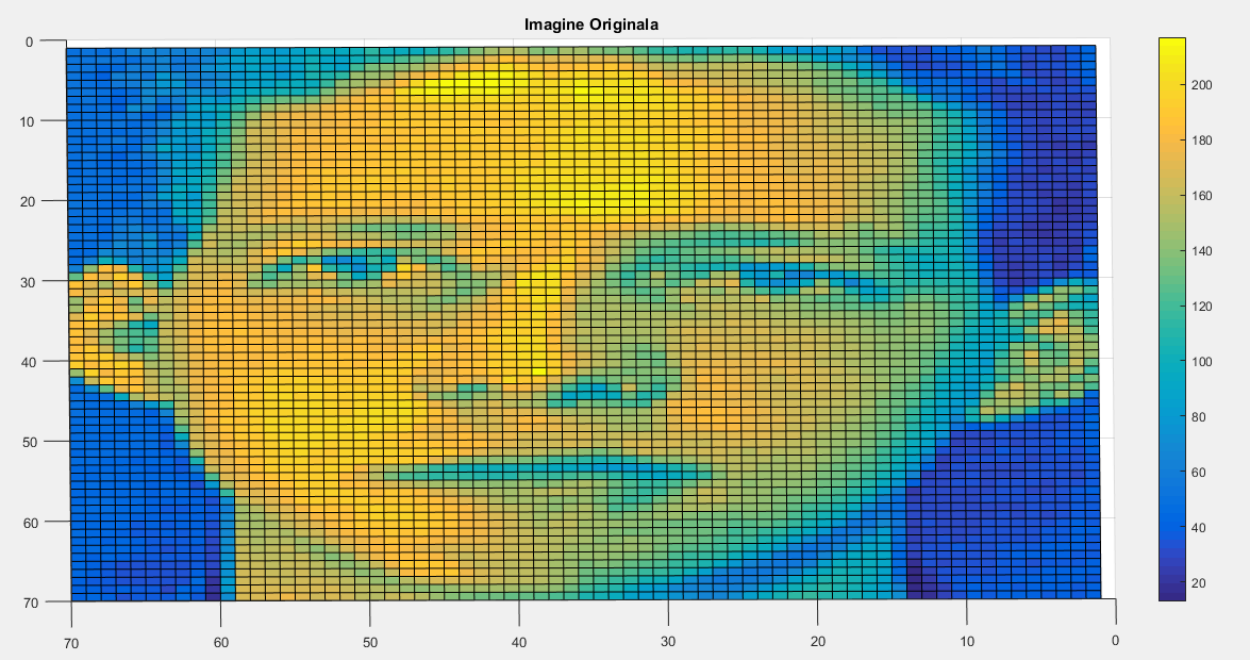



Figura 2.3 Imagine Originală

## 2.2 Detecția facială folosind clasificatori de tip cascadă bazați pe caracteristici LBP

Cascadele LBP (engl. LBP Cascade), la fel ca și cele Haar, reprezintă un algoritm de detecție a obiectelor inclus în librăria OpenCV.

Pentru a extrage trăsăturile LBP, mai întâi separăm imaginea în mai multe secțiuni, iar pentru fiecare secțiune facem o parcurgere a tuturor pixelilor din secțiunea respectivă pentru a obține valoarea acestor pixeli. Pentru fiecare pixel se verifică valorile pixelilor vecini, dacă valoarea pixelului vecin este mai mare acel pixel ia valoarea 1, dacă valoarea este mai mică acel pixel ia valoarea 0. Pixeli vecini sunt luați într-o ordine stabilită, ori în sensul acelor de ceasornic, ori invers acelor de ceasornic, rezultând un șir de 8 elemente care mai apoi este convertit într-un număr zecimal, care va fi noua valoare a pixelului central. După ce s-a verificat o secțiune se generează histograma pentru aceea secțiune, apoi se trece la următoarea. După ce au fost parcurse toate secțiunile se concatenează histogramele rezultate din fiecare și obținem o histogramă care conține informația necesară unei viitoare clasificării, fie că este vorba de detecție de obiecte sau recunoaștere de obiecte.

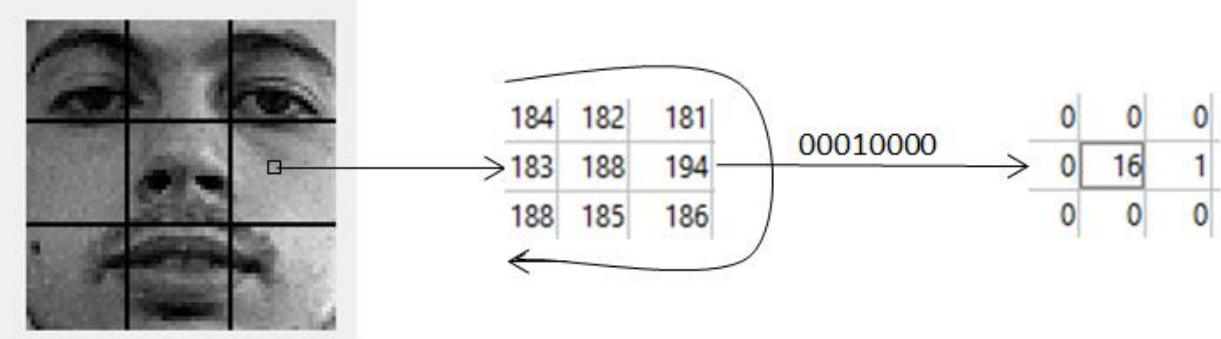



Figura 2.4 Proces extragere trăsături LBPH

După ce au fost parcurse toate secțiunile imaginii se generează histograma LBP, mai jos avem un exemplu de astfel de histogramă generată pentru imaginea din Figura 2.4

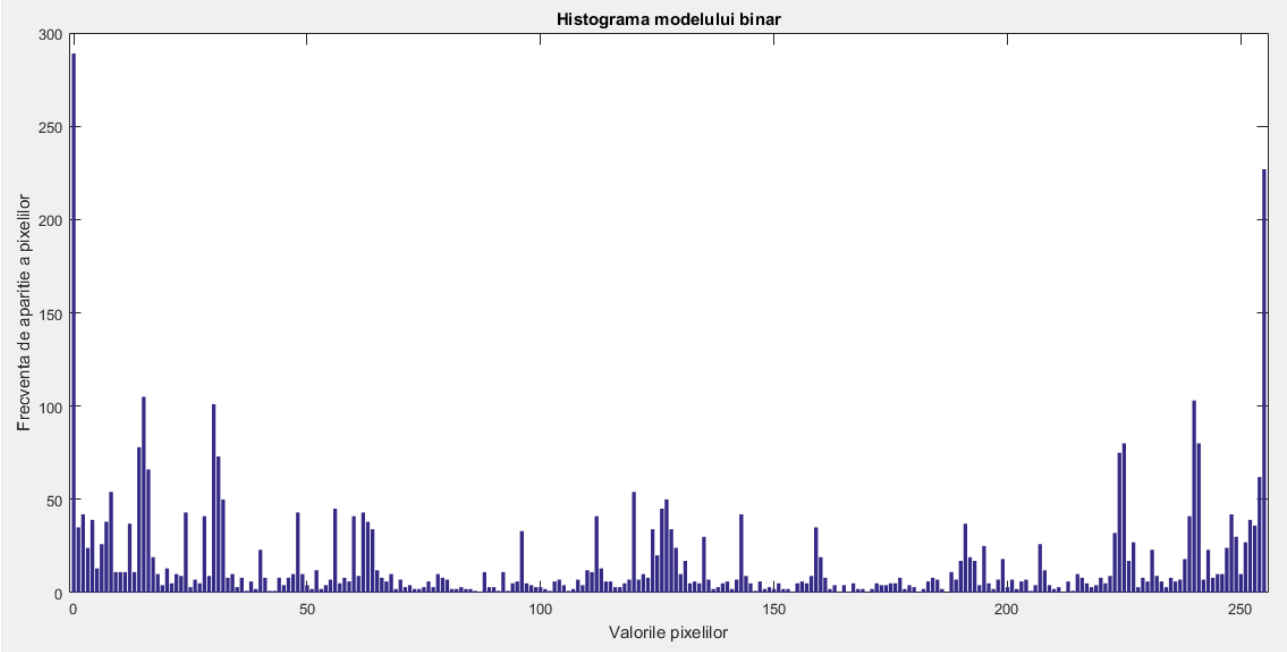



Figura 2.5 Histograma LBP

Folosind cât mai multe imagini în procesul de antrenare, se poate învăța un model care recunoaște o imagine facială în funcție de distribuția caracteristicilor LBP. Pentru a antrena un

model, la fel ca la cascadele Haar avem nevoie și de imagini care nu se aseamănă cu o față, un exemplu de astfel de imagine se poate observa în Figura 2.6 și histograma LBP corespondentă acesteia în Figura 2.7

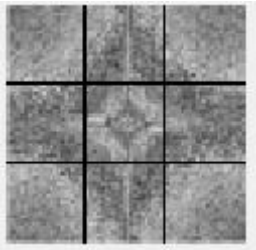


Figura 2.6 Imagine negativă

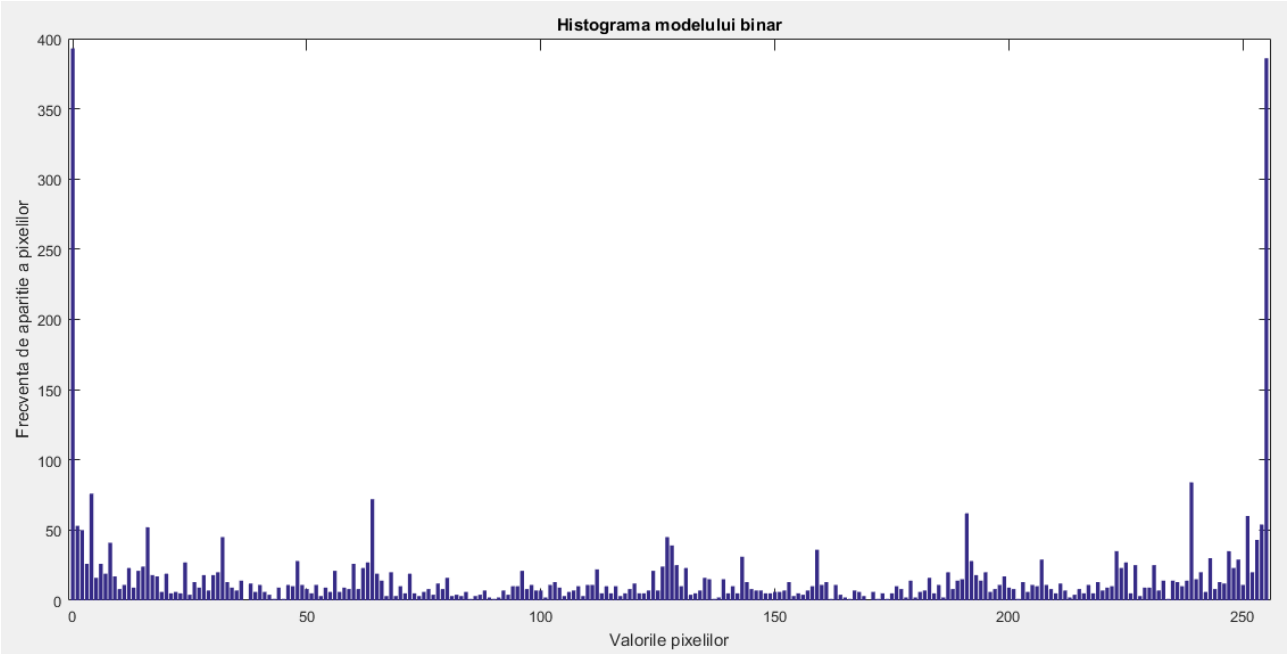



Figura 2.7 Histograma LBP Imagine negativă

Această metodă are avantaje și dezavantaje față de metoda cascadelor Haar, unul dintre majorele dezavantaje este acuratețea cu care se face detecția, în schimb această metodă este foarte rapidă și necesită un nivel de calcul scăzut.

Însă acuratețea poate fi îmbunătățită folosind diverse filtre, de exemplu căutarea celui mai mare obiect din imagine, acest filtru te ajută să obții un grad de acuratețe dar poate fi folosit pentru detecția unei singure persoane la un moment dat.

# Capitolul 3. Algoritmi de recunoaștere facială

## 3.1 Analiza Componentelor Principale

Analiza componentelor principale ( engl. Principal Component Analysis – PCA) este o metodă des utilizată pentru a realiza compresia datelor. Este o metodă eficientă de extragere a trăsăturilor dintr-un set de date şi în decursul timpului a devenit o metodă de referință în recunoașterea imaginilor. [3] [4]

Mai jos avem un exemplu PCA efectuat pe 20 de puncte bidimensionale (caracteristica 1 reprezintă axa X și caracteristica 2 reprezintă axa Y), care au fost proiectate pe componenta principală, obținând astfel reducerea dimensionalității.

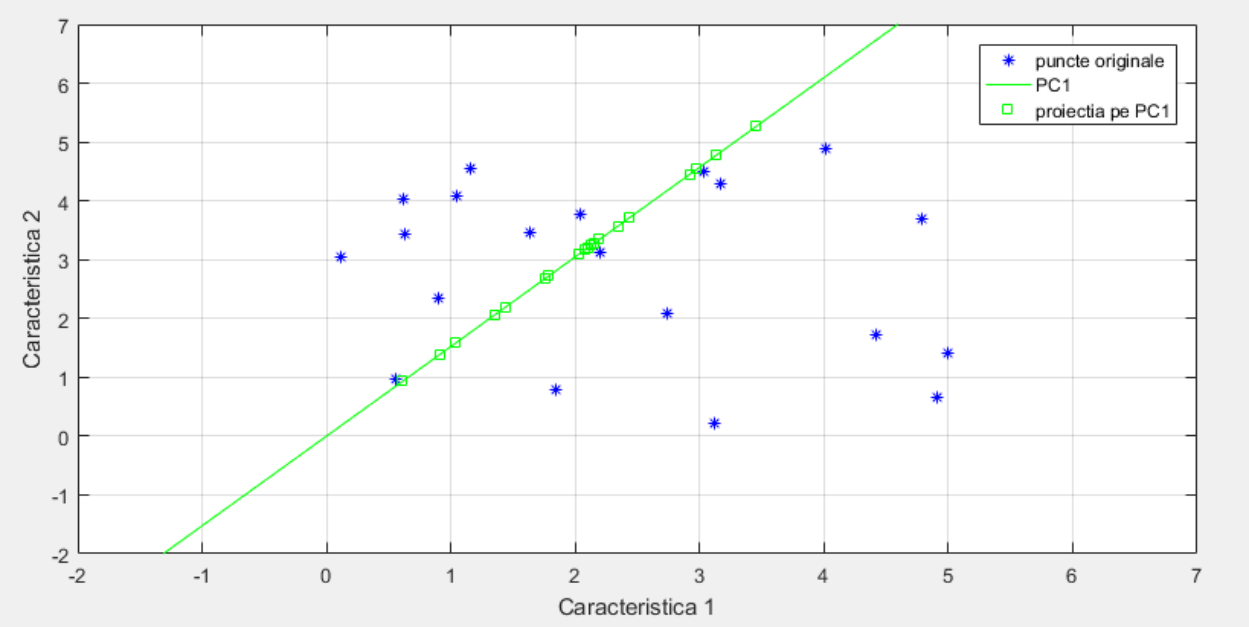



Figura 3.1 Proiecția punctelor pe prima componentă principală

Analiza Componentelor Principale (PCA) este o metodă statistică simplă pentru reducerea dimensionalităţii (compresie) care este deseori utilizată pentru realizarea recunoașterii modelelor.

Această metodă nu folosește similarități geometrice, ca distanța dintre ochi sau dimensiunea gurii pentru a clasifica o față. În schimb, un set de fețe umane este analizat folosind PCA pentru a determina contribuția ridicată a unei variabile la variația dintre imaginile din setul respectiv. În domeniul recunoașterii faciale, numim aceste variabile *eigenfaces*, deoarece atunci când sunt reprezentate, acestea se aseamănă într-o proporție foarte mare cu chipul uman.

Cea mai importantă calitate a PCA-ului este comprimarea și proiectarea datelor într-un spațiu mai mic. De exemplu daca avem o imagine de 66x66 pixeli (Figura 3.2) ce conține o față poate fi reprezentată cu o eroare medie foarte mică folosind doar 15 componente. Fiecare componentă conține informația care arată cât de mult influențează aceea componentă imaginea originală.

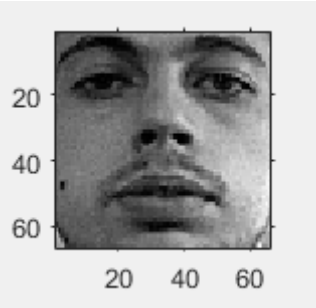



Figura 3.2 Imagine originală

Figura 3.3 Reconstrucția imagini

Astfel de exemplu pentru operația de recunoaștere facială avem nevoie doar de 15 eigenfaces pentru reconstrucția imagini în loc să folosim combinații liniare între cele 4356 de valori obținute din fiecare pixel al imaginii originale. Acest lucru este foarte util atunci când se vrea implementarea unei aplicații pe un sistem mobil, sistem care nu beneficiază de o putere de calcul sporită și care trebuie să facă recunoașterea facială în timp real.

### 3.1.1 Eigenfaces

Orice imagine în tonuri de gri (un singur plan de culoare) este formată dintr-o matrice de NxN elemente ce corespund valorilor intensității luminoase se poate reprezenta ca un vector linie de dimensiune $N^2$. Conform exemplului de mai sus o imagine de 66x66 a fost transformată într-un vector 4356-dimensional. Mai jos este prezentat un exemplu generic în care se folosește o imagine de 7x7, care este transpusă într-un vector multidimensional.

Figura 3.4 Transformare imagine de 7x7 într-un vector 49-dimensional

Pe de altă parte, folosind toate cele 4356 de dimensiuni pentru recunoașterea facială pentru antrenarea unui model nu ar ajuta la nimic, din contră acel detector ar avea erori foarte mari, deoarece fețele umane sunt asemănătoare, astfel vectorii ar fi foarte apropiați.

Vectorii originali care descriu fața subiectului sunt puternic corelați, astfel prin analiza componentei principale, cercetătorii au încercat să obțină o reprezentare mai bună a fețelor căutând vectorii care influențează cel mai mult distribuția imaginii. Acești vectorii creează spațiul facial, care oferă o reprezentare mai bună decât spațiul imaginilor care conține fiecare combinație posibilă de fețe. [5]

Vectorii proprii ar putea fi priviți ca fiind un set de caracteristici generale ale variațiilor imaginilor din baza de date. Odată ce portretele sunt normalizate (aduse la o dimensiune standard), ele pot fi tratate ca vectori unidimensionali de valori de pixeli. Fiecare imagine are o reprezentare exactă printr-o combinație liniară a acestor vectori proprii.

Algoritmul PCA pentru determinarea componentelor principale:

- Încărcăm datele inițiale sub formă matricială (m x n)
- Calculăm media pentru fiecare set de date
- Scădem media din matricea datelor inițiale
- Calculăm matricea de covarianță
- Se determină cele n valori proprii ale matricei Σ ca soluții ale ecuației :

$$\left|\sum - \lambda I\right| = 0$$

- Pentru fiecare valoare proprie $\lambda_j$ se determină vectorul propriu atașat

$\sum \alpha^j = \lambda_j \alpha^j$ cu condiția ca $\left\|\alpha^j\right\| = 1$

- Se determină forma noilor variabile $w_1,\ldots\ldots w_n$ conform formulelor:

$$w_j = (\alpha^j)^T X \quad j = 1,\ldots n$$

- Dintre $w_1,\ldots\ldots w_n$ se reține k componente principale

Această metodă este sensibilă la variații ca scalare, iluminare, rotație sau translație şi necesită o reînvățare completă a datelor de antrenare pentru a adăuga noi subiecți în baza de date. [3]

O astfel de reducere a dimensionalității poate fi foarte utilă în vizualizarea și procesarea seturilor de date multi-dimensionale, păstrând pe cât posibil datele cu varianța cea mai mare.

PCA are ca efect concentrarea a cât mai multor date în primele componente principale, ce pot fi folosite la reducerea dimensionalității, în timp ce componentele principale de ordin superior pot fi dominate de zgomot, astfel pot fi eliminate fără a se pierde foarte multă informație.

## 3.2 Recunoaștere facială bazată pe algoritmul Eigenfaces

În continuare va fi realizată o introducere în domeniul detecției și identificării fețelor umane și se va descrie un sistem de detecție facială care procesează informația cu un timp de așteptare foarte mic față de timpul real, acest lucru depinzând și de performanțele sistemului de calcul pe care se lucrează.

Dezvoltarea unui model de calcul pentru detecția facială este destul de dificilă, deoarece fețele au trăsături complexe și prezintă caracteristici diferite de la o față la alta, cum ar fi poziția și unghiul de cădere al luminii, îmbătrânirea și expresia facială.

O mare parte din activitatea de recunoaștere facială asistată de calculator s-a axat pe detecția trăsăturilor unice faciale: ochi, nas, frunte, gura și definirea unui model facial bazat pe poziția, mărimea și relația dintre aceste trăsături.

În prezent, există modele de recunoaștere care sunt capabile să învețe din informațiile primite din mediul înconjurător.

Algoritmii folosiți în fază incipientă pentru recunoaștere facială se bazau doar pe distanțele dintre elementele principale ale feței. În această teză se dorește implementarea unui sistem care extrage informații relevante din imaginile captate și să le comparăm cu imagini dintr-o baza de date obținute prin aceleași mijloace ca imaginea de test.

În termeni matematici vrem să identificăm componentele principale din distribuția fețelor, sau vectorii proprii ai matricei de covarianță din setul de imagini stocat în baza noastră de date. Acești vectori proprii pot fi priviți ca un set de caracteristici prin care putem face diferența între imagini. Fiecare pixel din imagini are o contribuție mai mare sau mai mică la fiecare vector propriu. *Eigenfaces* reprezintă un set de vectori proprii folosiți pentru recunoaștere facială.

Fiecare imagine din setul de antrenare poate fi reprezentată ca o combinație liniară de *eigenfaces.*

Numărul maxim de *eigenfaces* este egal cu numărul imaginilor din setul de antrenare. Dar în general pentru o detecție cu o acuratețe sporită se folosesc doar cele mai bune *eigenfaces,* adică acelea care au cele mai mari valori proprii și care automat sunt responsabile de variațiile majore între imaginile din setul de antrenare. Cele mai bune X *eigenfaces* creează un subspațiu X-dimensional numit, spațiul al fețelor, care cuprinde toate imaginile din setul de antrenare.

Pentru a calcula *eigenfaces* selectăm o imagine din setul de antrenare. Transformăm imaginea bidimensională într-un vector de NxN-dimensional. Repetăm această operație pentru toate imaginile din setul nostru, astfel obținem un ansamblu de imagini apoi asociem colecția de puncte într-un spațiu multidimensional pe care îl numim spațiul facial. Imaginile faciale, fiind asemănătoare ca și structură , nu o să fie distribuite aleator în spațiu, ele putând fi descrise într-un spațiu relativ mic comparativ cu întregul spațiu. Fiecare vector format reprezintă o combinație liniară din imaginile originale. Deoarece acești vectori reprezintă exact vectorii proprii ai matricei de covarianță (Figura 3.8), ce corespunde imaginilor faciale originale, și ei sunt de fapt o imagine facială, putem să îi numim *eigenfaces*. Exemplu de eigenfaces puteți vedea în Figura 3.9. [4]

Mai jos este prezentat procesul de recunoaștere facială folosind algoritmul eigenfaces, în exemplu este folosită baza de date cu imagini oferită de AT&T Laboratories Cambridge.

Imaginile sunt de dimensiune fixă 112x92 pixeli, organizate în 40 de subdirectoare cu câte 10 imagini pentru fiecare subiect.

Procesul se desfășoară după următoarele etape:

- Încărcarea unui set de antrenare cu imagini faciale:

Figura 3.5 Setul de date

- Obținerea mediei imaginilor din setul de date (“Mean face”)

Figura 3.6 „Mean face”

- Se extrage media din fiecare imagine din setul de date

Figura 3.7 Imagini normalizate

- Se calculează matricea de covarianță pentru întreg setul de date, după extracția mediei

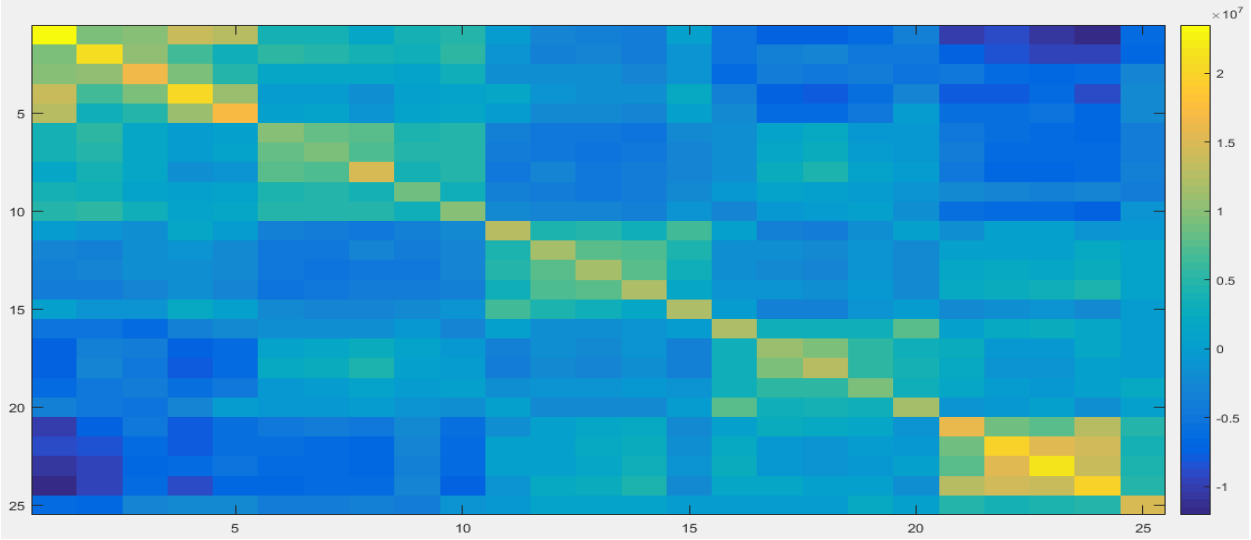



Figura 3.8 Matricea de covarianță pentru setul de date (25 imagini)

- Generăm vectori proprii (*eigenfaces-urile)*

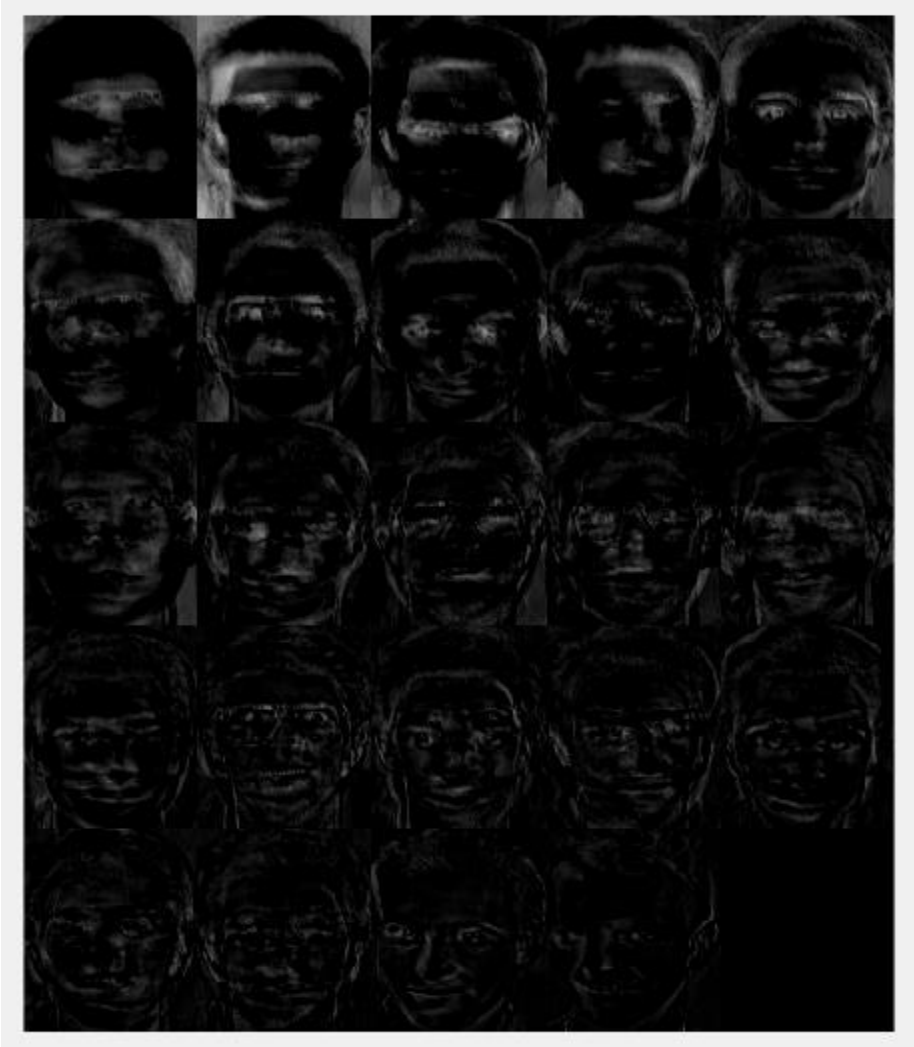


Figura 3.9 Eigenfaces-urile setului de date

- Se determină proiecția fiecărui vector din spațiul facial pe vectorii proprii (*eigenfaces*)
- Pentru o imagine nouă se determină proiecția imaginii de test pe eigenfaces și se caută

distanța minimă dintre proiecția imaginii de test și imaginile din baza de date.

La înregistrarea unei noi imagini faciale se calculează un grad de încredere al imaginii bazat pe imaginea captată și pe primele X *eigenfaces* proiectând imaginea peste fiecare *eigenfaces.*

Se determină dacă imaginea este o fața cunoscută (din baza de date) sau necunoscută, verificând cât de aproape de spațiul fețelor se află imaginea respectivă.


Figura 3.10 Imagine de test comparativ cu Imagine din baza de date

Dacă este o imagine facială se verifică gradul de încredere pentru a putea spune dacă este o față cunoscută sau nu.

Opțional se poate face *update* la modelul de antrenare, înregistrând fața necunoscută în baza de date.

Mai jos avem reprezentată matricea de covarianță obținută din 8 imagini de la 8 persoane diferite. Contrar exemplului de mai sus unde au fost folosite 5 imagini pentru fiecare subiect se observă că diagonala matricei are valori mai mari comparativ cu restul elementelor ceea ce arată că influența imaginii x asupra imaginii y este foarte mică.

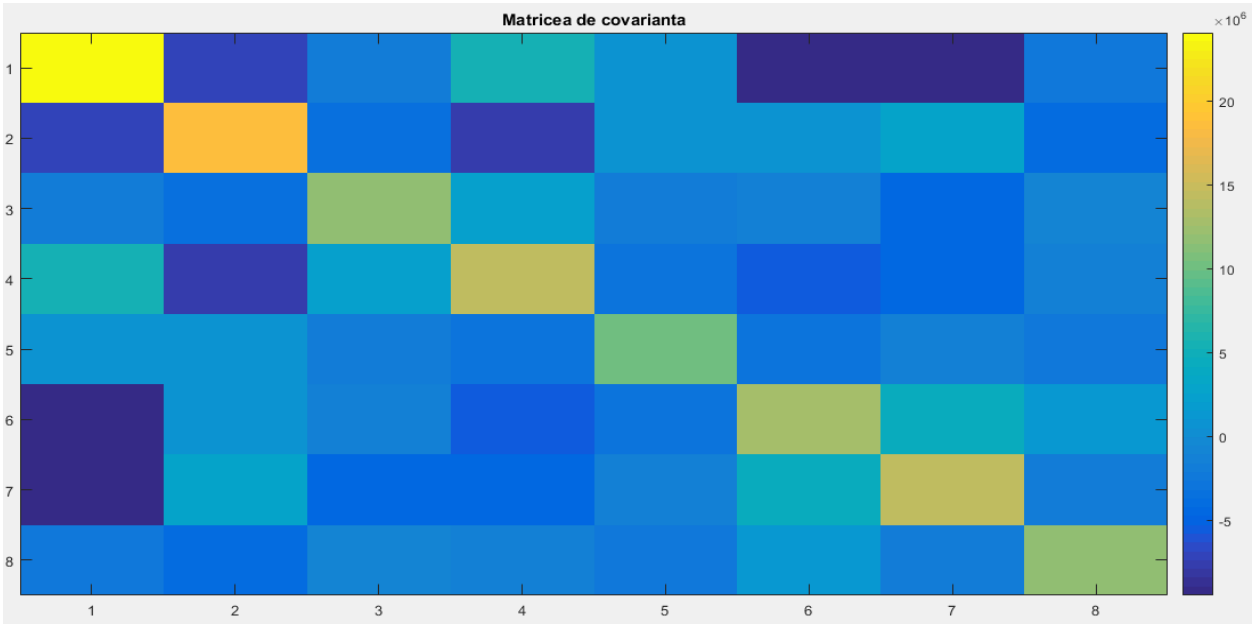


Figura 3.11 Matricea de covarianță

Mai exact, se consideră un set de N imagini $\{x_1,x_2,\ldots x_N\}$ ce sunt cuprinse într-un spațiu de imagini multidimensional, și să presupunem că fiecare imagine aparține unei din clasele C $\{X_1,X_2,\ldots X_C\}$. Să luăm în considerare de asemenea o transformare liniară de mapare a spațiului original n-dimensional într-un spațiu m-dimensional, unde m<n (reducerea dimensionalității). Noi vectori caracteristici $y_i \in \mathbb{R}^m$ sunt definiți de transformarea liniară:

$$y_k = W^T x_k \qquad k=1,2,\ldots N$$

Unde $W \in \mathbb{R}^{m\,x\,n}$ este o matrice cu coloane ortonormate.

Dacă matricea totală de dispersie $S_T$ definită ca:

$$S_T = \sum_{k=1}^{N} (x_k - \mu)(x_k - \mu)^T$$

unde n reprezintă numărul de imagini de probă, și $\mu \in \mathbb{R}^n$ este media tuturor imaginilor de probă, atunci după aplicarea transformatei liniare $W^T$, dispersia transformatei vectorilor caracteristici $\{y_1,y_2,\ldots y_N\}$ este $W^T \cdot S_T \cdot W$. În PCA, proiecția optimală $W_{opt}$ este aleasă astfel încât să maximizeze determinantul matricei de dispersie a eșantioanelor proiectate.

$$W_{opt} = arg \max_W |W^T S_T W| = [w_1\ w_2 \ldots w_m]$$

Unde $\{w_i|\ i=1,2,\ldots m\}$ reprezintă vectorii proprii ai matricei $S_T$ din setul n-dimensional corespunzători celor mai mari ***m*** valori proprii. Cum acești vectorii proprii au aceeași dimensiune ca imaginile originale, ei sunt menționați ca *eigenfaces* (figura 3.8 ) în lucrarea "Face Recognition Using Eigenfaces" scrisă de M. Turk și A. Pentland în 1991. [6]

Această metodă este folosită pentru reducerea dimensionalității de la NxN la m unde m<<NxN. Un dezavantaj al acestui algoritm este acela că maximizează nu doar dispersia între clase, ceea ce este folositor pentru clasificare, ci și dispersia în interiorul clasei, ceea ce este nedorit deoarece variația de la o imagine la alta este dată de schimbarea iluminării. Astfel în cazul în care folosim PCA pentru imagini cu iluminare variată matricea de proiecție $W_{opt}$ va conține componentele principale (eigenfaces), care rețin în proiecția spațiului caracteristic variațiile de

iluminare. În consecință punctele proiectate în spațiu nu vor fi foarte bine grupate, sau mai rău se vor amesteca cu punctele celorlalte clase. [4]

S-a constatat că dacă eliminăm primele trei cele mai importante componente principale, variația luminii este redusă. Deși, este puțin probabil ca primele componente principale să corespundă numai variației de iluminare, rezultând astfel o posibilă pierdere de informație utilă. [4]

## 3.3 Analiza Discriminatorie Liniară (LDA)

Analiza discriminatorie liniară (Linear Discriminant Analysis - LDA)) este o formă generală a FLD-ului (Fisher's Linear Discriminant) care, combină avantajele ambelor metode, în sensul că folosește mai întâi PCA-ul pentru a reduce dimensionalitate apoi folosește LDA pentru a dispersa cât mai mult clasele una de cealaltă. Spre deosebire de PCA, unde se urmărește o proiecție în sensul maximizării matricei totale de covariație, aici se caută o proiecție în sensul maximizării matricei de covariație inter-clase $S_B$ şi minimizării matricei de covariaţie cumulată din interiorul claselor $S_W$. Mai exact LDA încearcă să găsească cea mai bună direcție de proiecție în care vectorii de antrenare aparținând în clase diferite sunt cel mai bine separați.

Analiza discriminatorie liniară este folosită în statistică, recunoaștere de obiecte și *machine learning* (un subdomeniu al științei calculatoarelor care oferă sistemului de calcul abilitatea de a învăța fără să fie programate în mod explicit) pentru a găsi combinații liniare între proprietățile ce caracterizează sau separă două sau mai multe clase de obiecte sau evenimente. Rezultatul combinației poate fi folosit ca și clasificator liniar, sau, ca mijloc de reducere a dimensionalității.

LDA este strâns legată cu analiza variației (ANOVA) și analiza regresiei, care încearcă de asemenea să își exprime o variabilă dependentă ca o combinație liniară de alte caracteristici. [7]

LDA este de asemenea strâns legată de analiza componentelor principale (PCA) și analiza factorială în care ambele caută combinații liniare de variabile care interpretează cel mai bine datele. [8]

Analiza discriminatorie este totuși diferită de analiza factorială prin faptul că nu este o tehnică de interdependență: trebuie făcută o separare între variabilele independente și variabilele dependente.

LDA funcționează atunci când măsurătorile efectuate asupra variabilelor independente pentru fiecare observație sunt cantități continue. În cazul în care se ocupă cu variabile independente categorice, tehnica echivalentă este analiza corespondenței discriminantă. [9]

Algoritmul LDA pentru realizarea analizei discriminatorie:

Fie N persoane fotografiate(clase) cu câte $V_i$ vectori n-dimensionali (în acest caz numărul de pixeli ) în fiecare clasă.

Primul pas după stabilirea spațiului n-dimensional format din vectorii fiecărei clase, trebuie calculată matricea de covariație între clase, $S_B$ (*scatter between*) și matricea de covariație din interiorul clasei, $S_W$ (*scatter within*), conform următoarelor formule:

$$S_B = \sum_{i=1}^{N} P(\omega_i)(\mu_i - \mu)(\mu_i - \mu)^T$$

- $P(\omega_i)$ = probabilități apriori

- $\mu$ este media celor N clase, $\mu_i$ este media fiecarei clase.

$$S_W = \sum_{i=1}^{N} P(\omega_i) \frac{1}{V_i} \sum_{k=1}^{V_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

După calculul matricelor $S_B$ și $S_W$ se determină valorile proprii ale matricei $S_W^{-1}S_B$ și apoi se ordonează descrescător valorile proprii obținute.

$$|S_W^{-1}S_B - \lambda I_n| = 0$$

Se determină vectorii proprii corespunzători valorilor proprii($\Phi_i$):

$$S_W^{-1}S_B\Phi_i = \lambda_i\Phi_i$$

Matricea **W** de transformare LDA va fi formată din vectorii proprii determinați anteriori

$$W = [\Phi_1, \Phi_2, \dots, \Phi_n]^T$$

Fiecare vector de intrare **V** se va proiecta în spațiul LDA, astfel:

$$Y = WV$$

Se vor reține numai m<n componente ale lu Y, restul de n-m fiind înlocuite de zerouri. Se va obține astfel vectorul $\tilde{Y}$

Pentru a evidenția beneficiile proiecției liniare pe clase individuale , construim un exemplu minidimensional în care eșantioanele din fiecare clasă fac parte dintr-un subspațiu linear. În figura 4.12 este făcută o comparație între PCA și FLD într-o problemă cu două clase în care eșantioanele fiecărei clase sunt aleator distribuite într-o direcție perpendiculară pe subspațiul liniar. În acest exemplu N=20 (puncte),n=2 (clase) și m=1 (vectori proprii păstrați). Ambele metode au fost folosite pentru a proiecta puncte bidimensionale pe o dimensiune. Comparând cele două proiecții se observă că pentru metoda PCA clasele se amestecă între ele, nemaifiind separate în spațiul proiectat(spațiul unidimensional). Dar se vede totuși că sunt foarte separate una de alta. Prin metoda FLD punctele sunt separate pe clase , astfel clasificare este simplificată. [4]

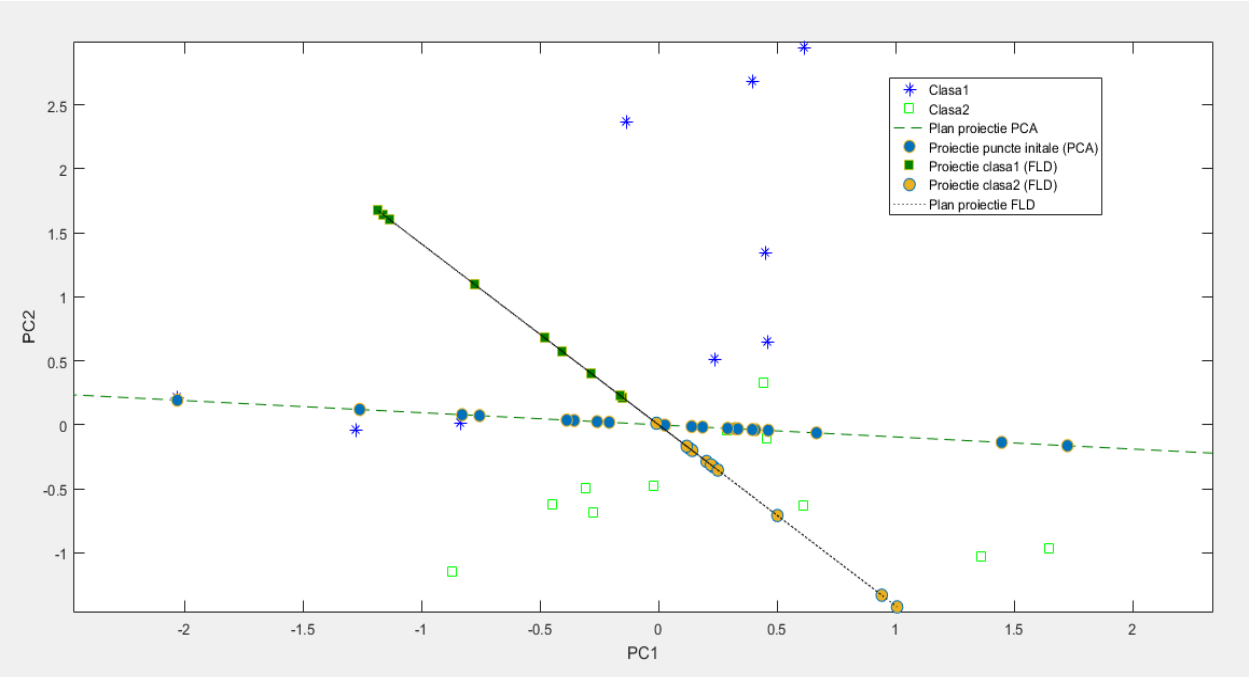



Figura 3.12 Comparație între PCA și FLD

Dacă aplicăm algoritmul PCA pentru reducerea dimensionalității, vectorii proprii sunt importanți deoarece ei vor forma noile axe ale noului subspațiu de caracteristici; valorile proprii asociate vectorilor ne spun ce cantitate de informație ne oferă fiecare axa nou creată.

## 3.4 Recunoaștere facială bazată pe algoritmul Fisherfaces

S-a dezvoltat un algoritm de recunoaștere facială care este independent de variațiile direcției luminii sau expresiei faciale. Într-o analiză a modelului de clasificare considerăm fiecare pixel al imaginii ca fiind o coordonată într-un spațiu multidimensional. Profităm de faptul că imaginile unei anumite fețe, sub diferite unghiuri de iluminare dar într-o poziție fixă , se găsește într-un subspațiu liniar tridimensional aflat în spațiul de fețe multidimensional, dacă fața respectivă este o suprafața Lambertiană (luminanța unei suprafețe Lambertiene este aceeași indiferent de unghiul de vizualizare al observatorului) fără umbre, reflectanța fiind maximă. Dar, din moment ce fețele nu sunt cu adevărat suprafețe Lambertiene, ele având umbre, imaginile vor fi deviate de la subspațiul liniar. Decât să explicităm aceste deviații, preferăm să proiectăm liniar imaginea într-un subspațiu astfel încât să neglijăm acele regiuni ale feței cu deviație mare. Metoda de proiecție este bazată pe discriminantul liniar al lui Fisher (LDA), despre care am vorbit în capitolul anterior, care produce clase separate într-un subspațiu de dimensiuni mai mici, chiar și atunci când există variații puternice ale luminii și expresiei faciale. Metoda Eigenfaces se asemăna metodei Fisherfaces însă rezultatele au demonstrat că metoda Fisherfaces are o rată de eroare mai mică decât Eigenfaces.

Pentru a calcula *fisherfaces*, presupunem că datele din fiecare clasă sunt distribuite normal (este eliminată caracteristica comună dintre clase, media). Notăm distribuția normală cu $N_i(\mu_i, \sum_i)$, unde $\mu_i$ reprezintă media și $\sum_i$ reprezintă matricea de covarianță și densitatea de probabilitate $f_i(x|\mu_i, \sum_i)$.

*Fisherfaces* este o metodă care încearcă să contureze variația, astfel încât să fie mai potrivită pentru clasificare. Metoda selectează W (matricea cu coloane ortonormate) în așa fel încât raportul dintre $S_B$ (matricea de covariație între clase) și $S_W$ (matricea de covariație în interiorul unei clase) , este maximizat.

Considerăm matricele de covariație $S_B$ și $S_W$ definite astfel:

$$S_B = \sum_{i=1}^{C} N_i(\mu_i - \mu)(\mu_i - \mu)^T$$

$$S_W = \sum_{i=1}^{C} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

Unde $\mu_i$ este media imaginilor din clasa $X_i$, iar $N_i$ reprezintă numărul de imagini dintr-o clasă $X_i$. Dacă determinantul matricei $S_w$ este diferit de 0 atunci proiecția optimală $W_{opt}$ este aleasă ca matrice cu coloane ortonormate care maximizează raportul dintre determinantul matricei de dispersie între clase a eșantioanelor proiectate și matricea de dispersie în interiorul clasei a eșantioanelor proiectate.

$$W_{opt} = arg\max_{W} \frac{|W^T S_B W|}{W^T S_W W} = [w_1\ w_2 \dots w_m]$$

unde {$w_i$| i=1,2,…m} este un set de vectori proprii ai matricelor $S_W$ și $S_B$ care corespund celor mai importante **m** valori proprii {$\lambda_i$| i=1,2,…m}

$S_B w_i = \lambda_i S_W w_i$ i=1,2,.....m

O problemă cu recunoașterea facială este aceea că matricea de dispersie din interiorul clasei are mereu determinantul 0. Acest lucru rezultă din faptul că rang-ul matricei $S_W$ este maxim N-n, iar de multe ori numărul imaginilor de test este mult mai mic decât numărul pixelilor fiecărei imagini. De aceea este foarte posibil să alegem o matrice W care să aibă variația în interiorul clasei exact 0. Metoda Fisherfaces rezolvă această problemă proiectând setul de imagini într-un spațiu minidimensional , astfel matricea de variație în interiorul clasei este diferită de 0. Acest lucru este posibil utilizând PCA pentru a reduce dimensionalitatea la N-n, apoi aplicăm metoda FLD pentru a reduce și mai mult dimensiunea la n-1.

Din punct de vedere matematic, $W_{opt}$ este definit ca:

$$W_{opt}^T = W_{fld}^T W_{pca}^T$$

Unde,

$$W_{pca} = arg\max_{W} |W^T S_T W|$$

$$W_{fld} = arg\max_{W} \frac{\left|W^T W_{pca}^T S_B W_{pca} W\right|}{\left|W^T W_{pca}^T S_W W_{pca} W\right|}$$

O altă metodă ar fi alegerea unei matrice W astfel încât să maximizeze dispersia între clase între proiecțiile eșantioanelor după ce matricea din interiorul clasei a fost redusă. Dusă la extrem, această metodă maximizează numai dispersia între clasele care au dispersia în interiorul clasei egală cu zero.

Procesul de recunoaștere facială folosind algoritmul *Fisherfaces* se desfășoară astfel:

- Se încarcă setul de date, același ca pentru exemplul cu eigenfaces (Figura 3.5 Setul de date)
- Se extrage media locală, media pentru fiecare clasă în parte.

Figura 3.13 Medii locale

- Se extrage media globală a tuturor claselor din setul de date (Figura 3.6 „Mean face”)
- Se determină matricea de covariație între imaginile din aceeași clasă ($S_W$) și matricea de covariație între imaginile din clase diferite ($S_B$).

- Se extrag vectorii proprii (fisherfaces) din matricea rezultată din înmulțirea matricei inverse $S_W$ și matricea $S_B$.

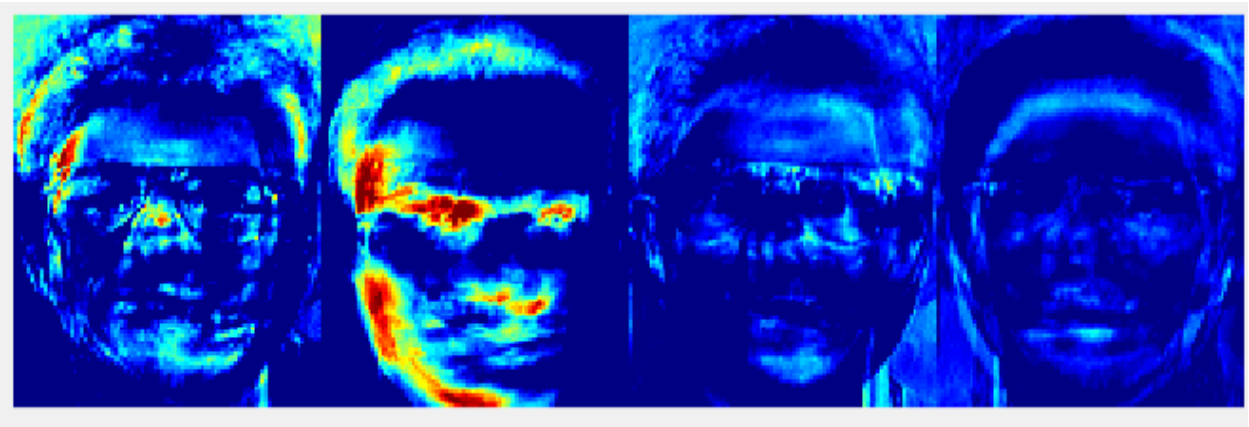


Figura 3.14 „Fisherfaces”

- Se determină proiecția tuturor imaginilor pe vectorii proprii (fisherfaces) obținuți
- Pentru a face recunoașterea, se ia o imagini de test se determină proiecția pe vectorii proprii și se compară cu proiecțiile celorlalte imagini din setul de antrenare.

## 3.5 Recunoaștere facială bazată pe algoritmul LBPH

În acest subcapitol o să prezint o metodă de recunoaștere facială care folosește atât forma cât și textura imaginilor pentru a reprezenta imaginile faciale. Suprafața faciala este segmentată în regiuni mai mici (șabloane) de unde se extrage histograma modelelor binare și sunt concatenate într-o singură histogramă spațială cu o caracteristică mai precisă. Recunoașterea se face pe baza vecinătăților spațiului compus din totalitatea modelelor. Experimentele au demonstrat ca această metodă este mult mai bună decât celelalte metode (PCA, LDA) . Testele au pus accent pe expresia facială, îmbătrânirea și luminarea subiectului. Folosind această metoda putem extrage foarte ușor informațiile dintr-o anume imagine.

Existența numeroaselor sisteme de recunoaștere facială arată că această știință a progresat semnificativ. În ciuda realizărilor acest subiect este încă activ.  Acest lucru se întâmplă deoarece sistemele bazate pe analiza componentelor principale par să funcționeze foarte bine în spații controlate din punct de vedere al poziției, iluminării dar atunci când sunt supuse acestor schimbări acuratețea lor scade. De aceea principala sarcină a  dezvoltatorilor acestui domeniu a fost să găsească o metodă care să crească acuratețea în aceste condiții. Astfel a fost creată o bază de date FERET  pentru a compara rata de detecție corecta de către diverși algoritmi de recunoaștere facială împotriva acestor factorilor enunțați mai sus.

Prima variantă de model binar local a fost introdusă de Ojala și colaboratorii. în anul 1996, și este un mijloc eficient de descriere texturală. Această metodă etichetează fiecare pixel al unei imagini folosind un prag reprezentat de pixelul central dintr-o vecinătate de 3x3,astfel,daca valoare unui pixel vecin este mai mare decât pragul setat valoare este suprascrisă cu 1, iar dacă valoare este mai mică decât prag se suprascrie valoare pixelului vecin cu 0 astfel se obține ca rezultat un număr

binar. Apoi histograma etichetelor poate fi utilizată ca descriptor textural. Pentru a înțelege mai bine urmăriți figura 2.4

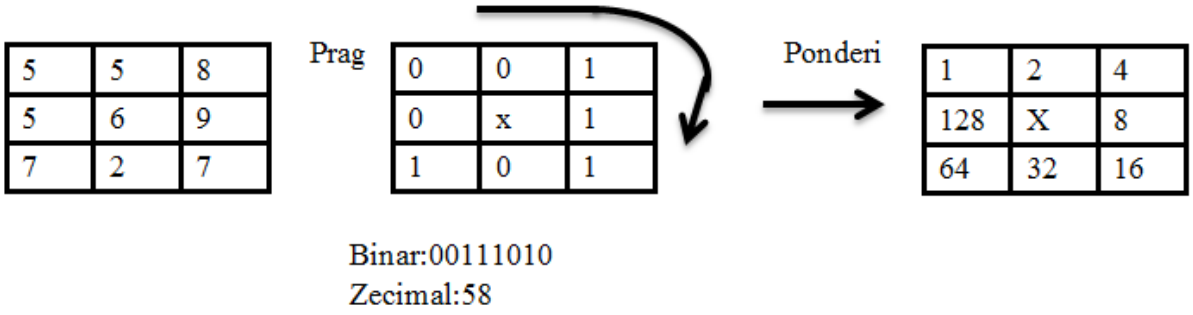



Figura 3.15 Binarizare Imagine

Pentru a clasifica o vecinătate se folosește un cod LBP obținut ca produs între valorile vecinătății după aplicarea pragului și ponderile date fiecărui pixel apoi la final însumăm produsele.

Astfel în cazul de mai sus **codul LBP** este : 4+8+16+64= **92**

Deoarece modelele binare sunt prin definiție monotone la schimbările alb-negru, s-a introdus o mărime ortogonală pentru caracterizarea contrastului local(C). Această mărime se calculează ca media dintre nivelele de gri mai mari decât nivelul central din care se scădea media nivelelor de gri mai mici sau egale decât nivelul central. Distribuția bidimensională a LBP-ului și măsurătorile contrastului local erau folosite ca trăsături. Acest operator purta numele de LBP/C

**Contrastul(C)** pentru exemplul de mai sus este : (6+7+7+9+8)/5-(5+3+2+2)/4 = **4.4**

O a doua variantă LBP, derivată din prima a fost introdusă tot de Ojala și colaboratorii în 2002, aceasta utiliza vecinătăți de mărimi diferite. Folosind vecinătăți circulare puteam folosii orice număr de pixeli situații la orice distanța de centrul vecinătății(R).

Să luăm exemplu textura T ca funcție de distribuție a valorilor nivelelor de gri ale pixelilor texturii respective . Astfel $T=f(g_c, g_1,\dots g_{p-1})$ unde $g_c$ este valoarea nivelului de gri al pixelului central iar celelalte sunt valorile nivelelor de gri ale pixelilor din vecinătatea respectivă (P). Astfel valoarea codului LBP a unui pixel P reprezentând centrul unei vecinătăți de coordonate $x_c$ și $y_c$ este dată de următoarea expresie:

$$LBP(P,R) = \sum_{i=0}^{P-1} s * (g_p - g_c) * 2^i$$

Unde s este funcția de prag, care este 1 atunci când o valoare este mai mare sau egala cu valoarea de prag și 0 în rest.

O abordare ce are la bază descrierea unei imagini pe baza caracteristicilor LBP afirmă că imaginea este împărțită în regiuni și sunt extrași descriptorii texturii din fiecare regiune individual. Apoi descriptorii sunt concatenați pentru a obține descrierea globală a imaginii. Această histogramă conține o descriere a imaginii pe 3 nivele : informațiile de pe primul nivel sunt la nivel de pixel apoi acestea sunt însumate la nivel de regiune obținând informația regională corespunzătoare nivelului 2, apoi histogramele pe regiuni sunt concatenate obținând o informație globală asupra imaginii corespunzătoare nivelul 3. [10]

# Capitolul 4. Limbajele de programare și tehnologiile software și hardware folosite în proiect.

## 4.1 Limbajele de programare folosite

În acest proiect am ales să folosesc limbajul de programare C++, limbaj dezvoltat de Bjarane Stroustrup în anii 1980, acesta a introdus clasele de obiecte , suprascrierea operatorilor, excepțiile și funcțiile virtuale. Fiind un limbaj orientat pe obiecte are următoarele proprietăți:

- **Polimorfism** reprezintă posibilitatea de folosii obiectele în mod diferit , de exemplu avem clasa poligon ce are o metodă arie, trebuie să existe un mod ca poligoanele derivate din această clasa (pătrat, cerc, triunghi) să poată avea arii diferite.
- **Încapsularea** reprezintă un mecanism de restricționare a accesului direct de către utilizator la datele componente ale obiectului. Acele componente pot fi accesate prin metode puse la dispoziție de obiect, numite *getters* și *setters*.
- **Moștenire** reprezintă proprietatea unui obiect de a folosii caracteristicile și funcțiile altui obiect.
- **Modularitate** reprezintă proprietatea obiectelor de a fi folosite independent. Altfel spus modulele nu sunt altceva decât niște fișiere ce pot fi compilate separat.

## 4.2 Tehnologii software:

### 4.2.1 OpenCV

OpenCV este o bibliotecă open-source (pune la dispoziție anumite produse finite, lăsând utilizatorii să o modifice și să o îmbunătățească fără nici un fel de obligație) pentru domeniul prelucrării de imagini și nu numai, dezvoltată inițial de către Intel. Biblioteca este are interfețe pentru C/C++, Python, Java, Matlab și rulează pe Windows, Linux și Mac OS X. Această tehnologie se bazează în mod special pe procesarea imaginilor în timp continuu. În acest moment biblioteca OpenCV suportă capturi în timp continuu, detecția obiectelor , aplicarea filtrelor simple pe imagini.
În proiectul meu am folosit biblioteca OpenCV pentru detecția facială utilizând cascadele Haar și LBP, recunoașterea facială utilizând algoritmi Fisherfaces, Eigenfaces și LBPH.

### 4.2.2 EXtensible Markup Language

XML este un meta-limbaj de marcare folosit la dezvoltarea altor limbaje cum ar fi XHTML, RSS. Acesta este folosit și ca model de stocare a informației și permite utilizatorului să își creeze propriile tag-uri, care să fie adaptate fix la aplicația acestuia. În OpenCV modelele de detecție facială și modele de recunoaștere facială generate de algoritmii specifici, Eigenfaces, Fisherfaces și LBPH pot fi stocate sub formă de fișiere XML, un astfel de fișier este structurat ca în figura de mai jos:

```xml
<mail>
    <Expeditor>Sorin</Expeditor>
    <Destinatar>Alina</Destinatar>
    <Mesaj>Am adus cartea!</Mesaj>
</mail>
```

Figura 4.1 Exemplu XML

## 4.3 Tehnologii hardware: Kuka Youbot și Asus Xtion

### 4.3.1 Arhitectura Software și Hardware a robotului KUKA Youbot.

Arhitectura Robotului KUKA este formată din două module, driver-ul EtherCAT și interfața programabilă KUKA YouBot.

**Driver-ul EtherCAT** reprezintă nivelul hardware care controlează toate actuatoarele robotului prin protocolul EtherCAT. Protocolul folosește fire de rețea și adaptoare pentru conectarea la rețeaua locală de internet, adaptoare de tip RJ45 pentru a conecta componentele slave cu cea master şi a stabili conexiunea între acestea. KUKA folosește o implementare de driver open-source SOEM pentru componenta master.

**Interfața KUKA YouBot** reprezintă nivelul software , acesta a fost dezvoltat de Autonomous Systems Group at Bonn-Rhein-Sieg University of Applied Sciences din Germania. Acesta este bazat pe driverele EtherCAT SOEM și oferă funcții pentru cinematica platformei și pentru comandarea articulațiilor manipulatorului.

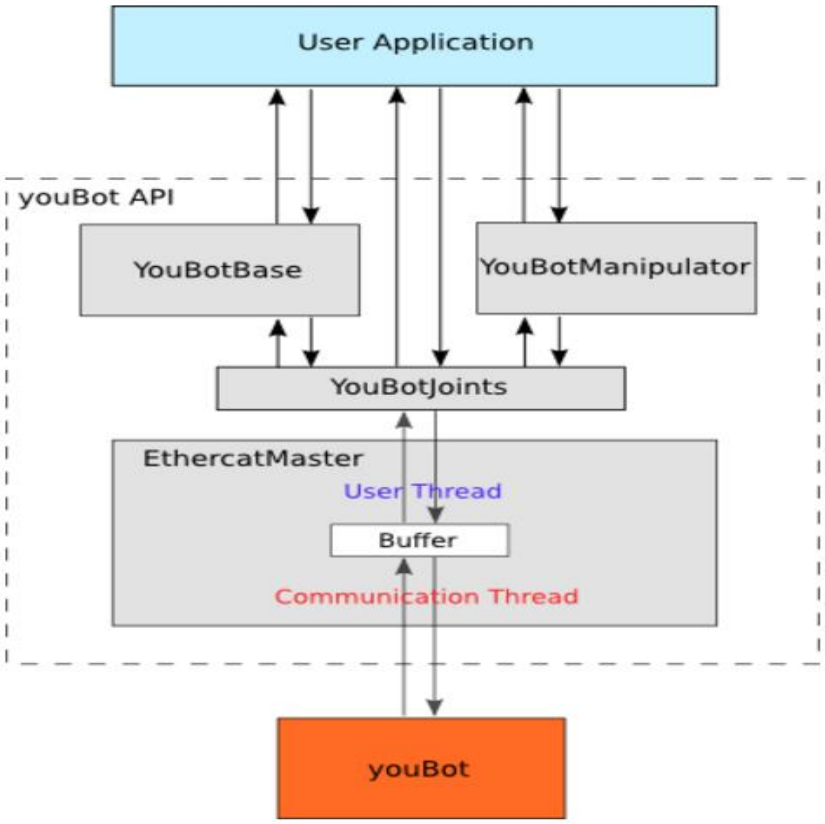



Figura 4.2 Arhitectura robotului KUKA [11]

În interfața programabilă KUKA YouBot robotul este văzut ca o combinație de sub-sisteme funcționale decuplate.
În API-ul KUKA YouBot există 3 clase principale:

- youBotManipulator aceasta reprezintă brațul robotului ca un sistem unitar format din articulații și un clește de prindere.
- youBotBase acesta reprezintă platforma mobilă.
- youBotJoint acesta reprezintă articulațiile care stau la baza celor doua clase de mai sus.

Mai jos este prezentată schema manipulatorului robotului din punct de vedere al funcționalității software:

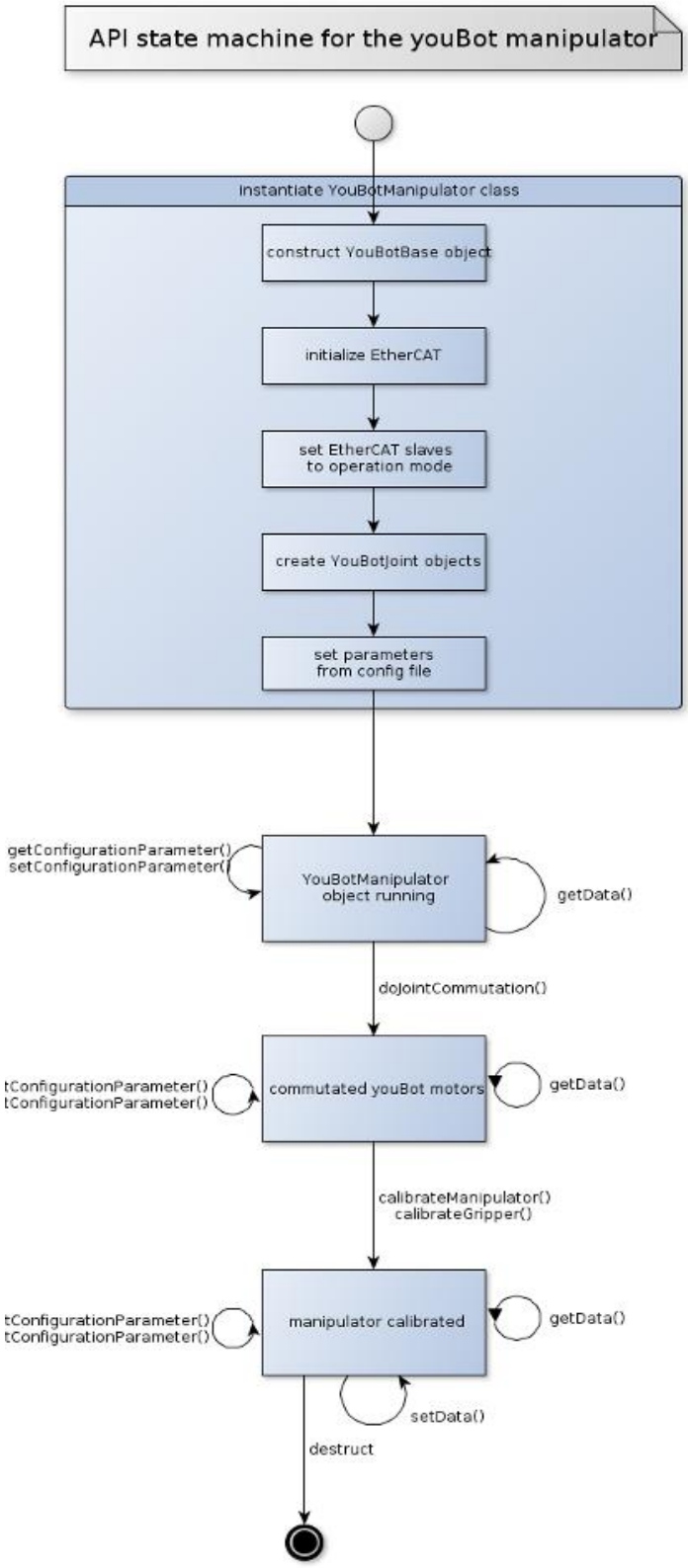



Figura 4.3 Stările Manipulatorului [11]

Kuka Youbot este robotul folosit în cadrul proiectului meu. Acesta este un manipulator mobil ce a fost dezvoltat inițial în scop educativ și pentru cercetare. Kuka Youbot vine cu o interfața care permite utilizatorilor să acceseze sistemul la toate nivele de control al hardware-ului.

Kuka Youbot are două componente principale:

- O platformă mobilă (Figura 4.4) ce formează șasiul robotului, 4 roți mecanice, motoare, alimentare și o unitate de procesare. Utilizatorul poate rula programele direct de pe bordul integrat sau controla de la distanța folosind un calculator.

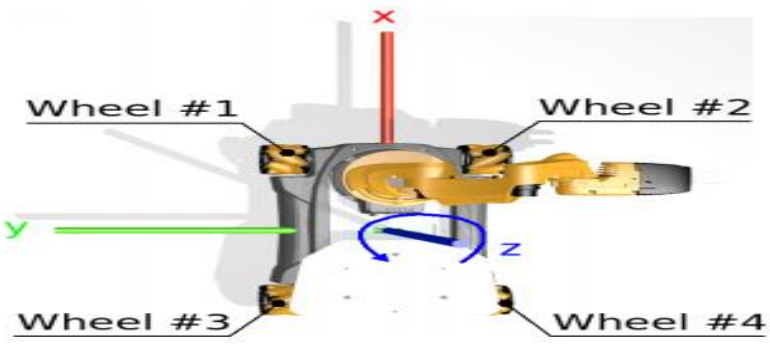



Figura 4.4 Platformă mobilă [12]

- Brațul robotic care are 5 grade de libertate și ca efector, un clește de prindere. Brațul poate fi controlat atât de pe placa dedicată, atunci când este conectat la platforma mobilă, cât și de la un alt calculator prin cablul Ethernet. În Figura 4.5 este prezentată structura cinematică a brațului robotic.

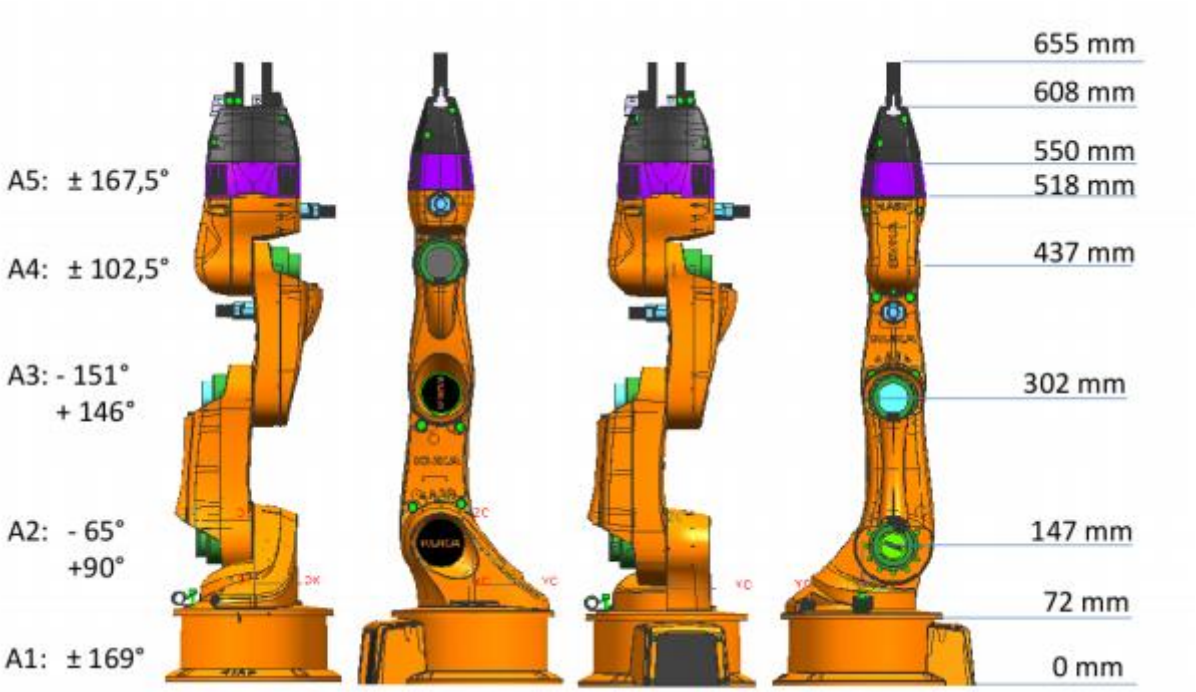



Figura 4.5 Structura cinematică a brațului robotic [12]

Unitatea de procesare integrată este o placă Mini-ITX. Acesta are un procesor Intel AtomTM Dual Core D510 (1M cache, 2x1.66 GHz), o memorie RAM de 2GB cu frecvența de lucru de 667MHz, și un spațiu de stocare SSD de 32GB. [12]

Acest robot permite conectare prin USB a unei tastaturi și a unui mouse , iar pentru a conecta un monitor avem nevoie de un cablu VGA. Acestea sunt foarte utile atunci când apar probleme de conexiune sau în procesul de depanare.

**Asus Xtion Pro LIVE** (Figura 4.6) este o cameră cu senzor de mișcare. Aceasta folosește senzori cu infraroșu, tehnologie adaptivă de detecție a adâncimii, senzori color și un sistem audio în timp real pentru a obține o detecție a utilizatorului cât mai precisă.


Figura 4.6 Asus Xtion Pro LIVE [13]

Această cameră dispune de 2 microfoane, o camera RGB ce poate funcționa la o rezoluție de 1280x1024 , 640x480 sau 320x240 și o cameră de adâncime care funcționează la rezoluția de 320x240. Distanța la care poate fi utilizată este între 0.8m și 3.5m, și are un consum de putere de maxim 2.5W, dispune de o interfața USB 2.0 și este compatibilă cu Windows(32/64),Linux și Android. Ca și kit software de dezvoltare folosește framework-ul OpenNI (Open Natural Interaction) și poate fi programată în C++ pe platforma Linux sau Java.

În proiectul meu folosesc această cameră RGB la rezoluția maximă (1280x1024) pentru a face capturi foto pe baza cărora se realizează detecția și recunoașterea facială.

# Capitolul 5. Rezultate experimentale

## 5.1 Detecție facială

Rezultatele experimentale de mai jos au fost făcute folosind o bază de date de la AT&T. O să încep această secțiune cu experimentele efectuate asupra funcției de detecție facială din punct de vedere atât al timpului de detecție, unde se va face o comparație între detecția obiectelor folosind *Haar Cascade*, metodă propusă de Paul Viola și Michael Jones în lucrarea : "Rapid Object Detection using a Boosted Cascade of Simple Features" publicată în 2001 [2], și *LBP Cascade*, metodă propusă de Jo Chang-Yeon într-o lucrare publicată în 2008. [14]

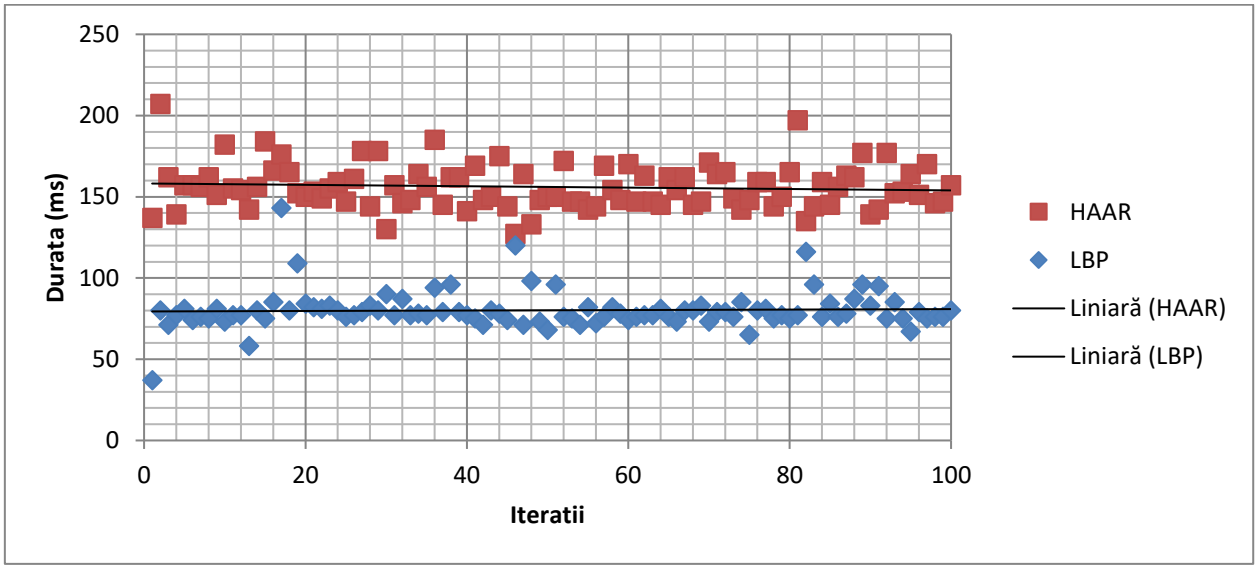



Figura 5.1 Durata perioadei de detecție Haar Cascade comparativ cu LBP Cascade

Testul a fost efectuat de 100 de ori și au fost salvate duratele perioadei de detecție pentru cele două metode și s-a obținut o durată medie de **160 milisecunde** pentru metoda Haar și în jur de **80 de milisecunde** pentru metoda LBP. Prin urmare pe un sistem cum este Kuka Youbot, care lucrează în timp real este recomandată implementarea metodei LBP deoarece timpul de procesare este mai scurt (toate calculele asupra imaginii se fac folosind numere întregi, comparativ cu Haar, care folosește numere reale în operațiile matematice de calcul al ferestrei, fapt ce solicită procesorul sistemului KUKA foarte mult), caracteristică esențială acestui tip de aplicație. Totuși LBP are o acuratețe mai slabă față de Haar. Modelul de detecție este salvat sub formă XML pentru a putea fi ușor manipulat, de exemplu dacă avem un model LBP care detectează expresii faciale rotite la $30^o$ putem să îi modificăm trăsăturile pentru a obține un model care detectează expresii faciale rotite cu un unghi de $-30^o$.

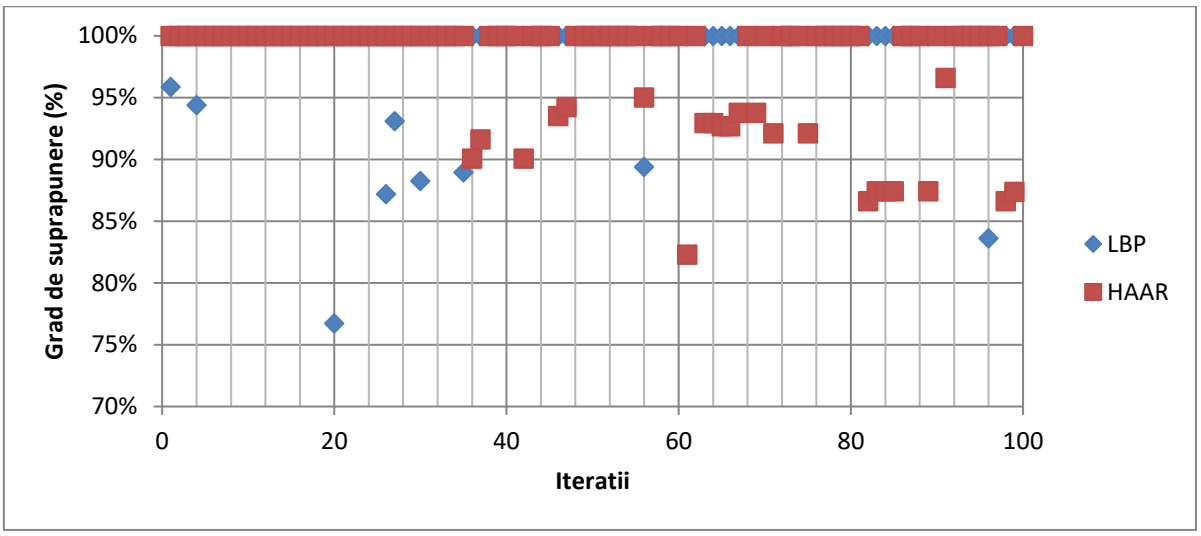



Figura 5.2 Gradul de suprapunere a ferestrelor de detecție

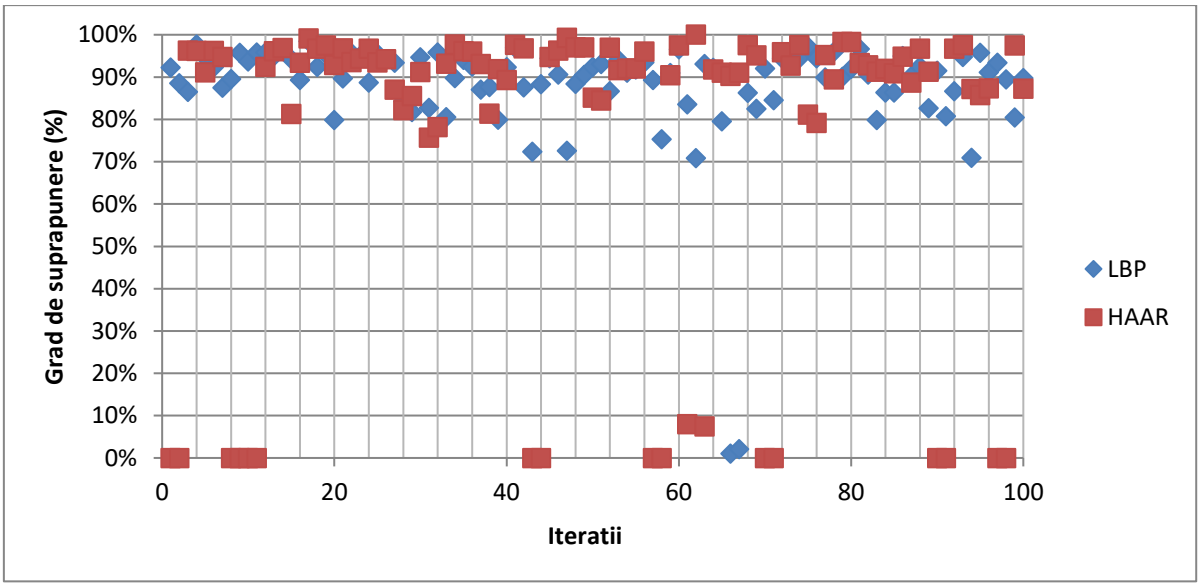


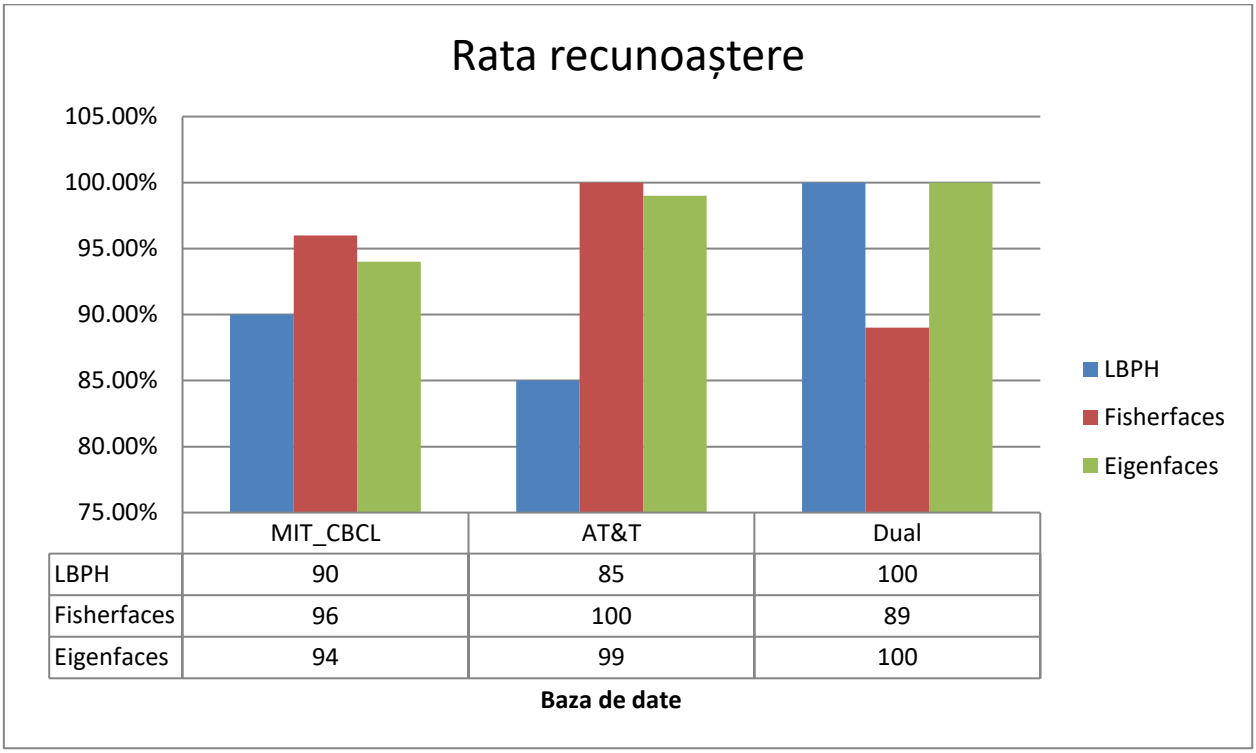

Figura 5.3 Gradul de suprapunere a ferestrelor de detecție în condiții de lumină nefavorabile

Pentru gradul de suprapunere a ferestrelor a fost mai întâi determinată manual fereastra optima de detecție, apoi au fost realizate 100 de cadre cu un detector bazat pe cascade LBP si Haar, automat, după aceea au fost extrase coordonatele ferestrelor de încadrare a feței provenite de la detectorul automat și suprapuse peste fereastra determinată manual, determinându-se astfel procentajul de suprapunere a ferestrei automate peste cea manuală. S-a constata că în condiții de luminozitate normală, gradul de suprapunere atât al ferestrelor cu cascade Haar, cât și al celor cu cascade LBP a fost între 70 și 100 %, preponderent procentajul fiind 100%, în timp ce în cazul în care condițiile de lumină erau nefavorabile (lumină intensa pe o parte a feței, unghiul luminii diferit) au existat rezultate cu 0% suprapunere preponderent pentru detecția Haar, în jur de 15%, iar pentru detecția LBP 2%.

## 5.2 Recunoaștere facială

La partea de experimente pe recunoașterea facială s-a testat acuratețea cu care se face recunoașterea facială, experimentul a fost realizat pe o bază de date de la AT&T, una de la MIT-CBC și o bază de date proprie cu 4 persoane, iar pentru fiecare persoană s-au ales 20 de poze de dimensiune 112x92.

Figura 5.4 Baza de date proprie




| | MIT_CBCL | AT&T | Dual |
|---|---|---|---|
| LBPH | 90 | 85 | 100 |
| Fisherfaces | 96 | 100 | 89 |
| Eigenfaces | 94 | 99 | 100 |

Figura 5.5 Rata de recunoaștere

După cum se observă din graficul de mai sus rezultate mai bune au fost pentru baza de date AT&T, deoarece aceea conține imagini în care nu există diferențe de lumină, astfel conform suportului teoretic care spune că atunci când nu există variații mari de luminozitate sau de poziție a feței atât algoritmul Eigenfaces cât și Fisherfaces au acurateță foarte mare, atunci când s-a folosit baza de date de la MIT metodele și-au păstrat eficiența , pe când atunci când s-au concatenat cele două baze de date într-una singură nu s-au mai obținut rezultate bune cu fisherfaces însă rezultatele algoritmului bazat pe modele binare au crescut foarte mult. Totuși pentru recunoașterea în timp real

cel mai eficient algoritm s-a dovedit a fi LBPH-ul, fiind un mediu necontrolat, celelalte metode au un nivel scăzut de acuratețe fapt ce se poate vedea din următoarele tabele:

Tabel 5.1. Rezultate LBPH

| Subiecți | Confidența | | | Acuratețe |
|---|---|---|---|---|
| | Minimă | Maximă | Medie | |
| Sorin | 28 | 52 | 40 | 95% |
| Florin | 35 | 57 | 46 | 86% |
| Georgiana | 43 | 55 | 49 | 85% |
| Andreea | 41 | 53 | 47 | 79% |

Tabel 5.2. Rezultate Eigenfaces

| Subiecți | Confidența | | | Acuratețe |
|---|---|---|---|---|
| | Minimă | Maximă | Medie | |
| Sorin | 228 | 343 | 285.5 | 40% |
| Florin | 270 | 350 | 310 | 34% |
| Georgiana | 240 | 320 | 280 | 20% |
| Andreea | 210 | 380 | 295 | 15% |

Tabel 5.3. Rezultate Fisherfaces

| Subiecți | Confidența | | | Acuratețe |
|---|---|---|---|---|
| | Minimă | Maximă | Medie | |
| Sorin | 150 | 650 | 400 | 78% |
| Florin | 200 | 800 | 500 | 65% |
| Georgiana | 340 | 700 | 520 | 54% |
| Andreea | 300 | 690 | 495 | 43% |

Unde confidența arată distanța dintre imaginea de test și cea mai apropiată imagine din baza de date de aceasta, cu cât distanța este mai mică cu atât certitudinea că persoana a fost recunoscută corect este mai mare.

Iar pentru a elimina situațiile în care o persoană este recunoscută ca fiind altă persoană s-a setat un nivel de prag determinat prin medierea nivelelor de confidență obținute, de la care se acceptă persoana din imaginea de test ca fiind cea corectă.

# Capitolul 6. Prezentarea aplicației

## 6.1 Funcționarea sistemului

Modul de funcționare a sistemului este exemplificat în organigrama de mai jos:

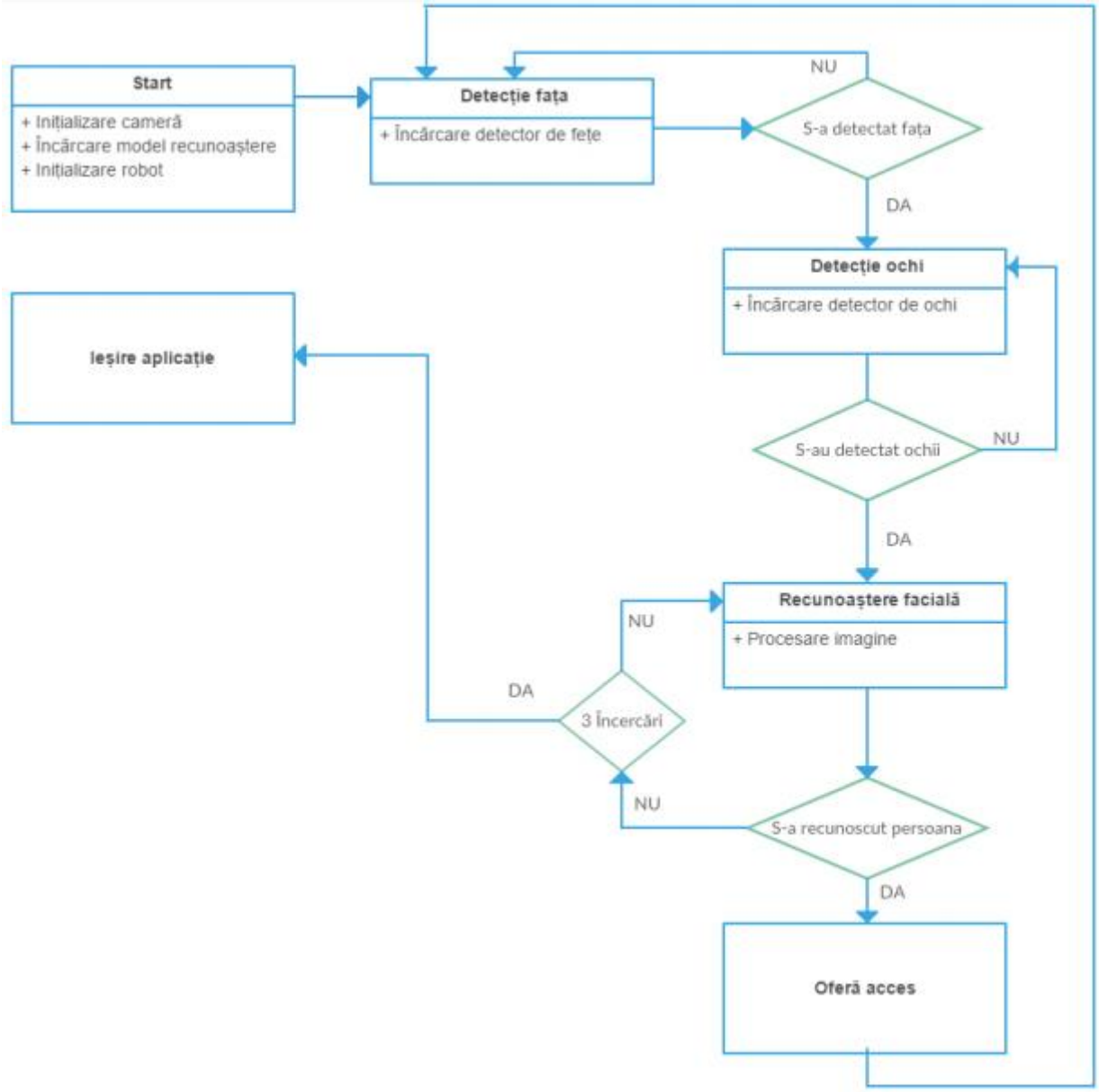



Figura 6.1 Organigramă sistem

Sistemul se pornește, făcând inițializările necesare, pornește camera, calibrează brațul robotic (Vezi ANEXA 1), se încarcă etichetele și numele persoanelor din baza de date, extrăgând informațiile dintr-un fișier folosind metoda `read_csv` (Vezi ANEXA 2), se încarcă un model pe care îl vom folosii în modulul de recunoaștere și inițializează articulațiile robotului. Modelul a fost creat înainte de execuția programului, acesta a fost format din 100 de imagini de test extrase dintr-o bază de date alcătuită din 4 subiecți (25 de imagini pentru fiecare subiect). Imaginile au fost redimensionate la o dimensiune fixa (100x100), dimensiune ce va fi folosită ulterior și pentru

imaginile obținute de la cameră. Apoi imaginile sunt procesate pentru a elimina variațiile de lumină din imagine.

După etapa de inițializare urmează etapa de detecție în care se încarcă detectorul de față și se caută fața unei persoane în câmpul vizual al camerei, după ce s-a detectat o față se mai face o căutare după ochii persoanei pentru a elimina detecțiile eronate.

```
//Modul detectie
static void FaceDetect(Mat& sample,vector< Rect_<int> > &faces)
{
    CascadeClassifier face_cascade;

    face_cascade.load(facePath);
    face_cascade.detectMultiScale(sample, faces,1.1,4,CASCADE_FIND_BIGGEST_OBJECT);
    return;
}
static void EyesDetect(Mat& sample,vector< Rect_<int> > &eyes)
{   CascadeClassifier eye_cascade;
    eye_cascade.load(eyePath1);
    eye_cascade.detectMultiScale(sample,eyes, 1.1 , 2, 0|CV_HAAR_SCALE_IMAGE,Size(8,20));
    return;
}
```

Figura 6.2 Modul detecție față și ochi

Pe lângă detecția de ochii, ca o îmbunătățire adusă sistemul s-a introdus căutarea celui mai mare obiect din imagine, prin adăugarea unui parametru funcției `detectMultiScale`, `CASCADE_FIND_BIGGEST_OBJECT`, acest lucru eliminând și mai mult posibilitatea unei false detecții.

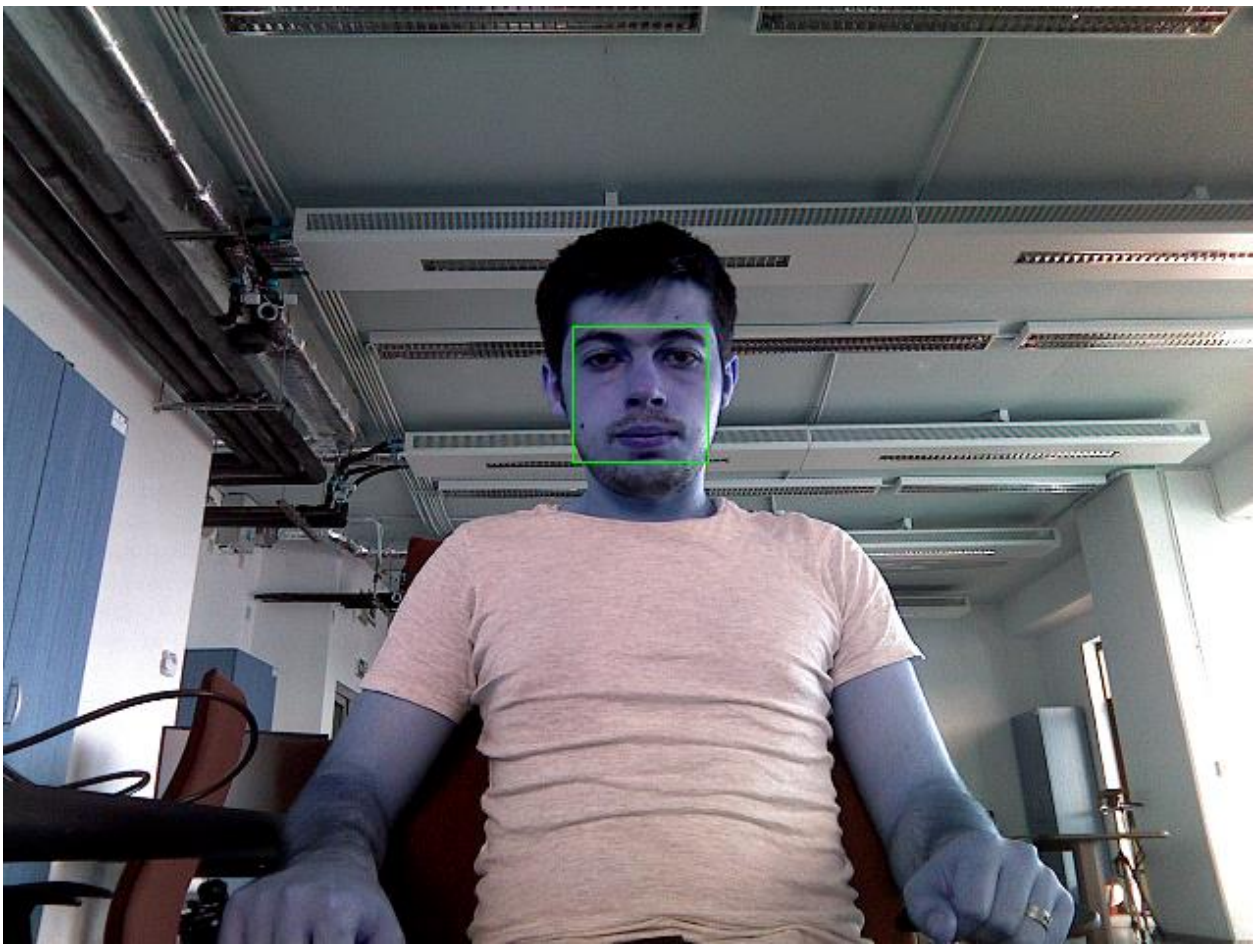


Figura 6.3 Modul detecție (vedere din aplicație)

Dacă s-au detectat și ochii se preia  imaginea, care intră mai apoi într-o etapă de procesare pentru a corespunde cu imaginile din faza de antrenare care au stat la baza modelului încărcat în etapa de inițializare, apoi aceea imagine este folosită ca intrare în modulul de recunoaștere

```
//Modul recunoastere
struct pattern recognize(Mat test,Ptr<FaceRecognizer> model)
{
    struct pattern v;
    int prediction=-1;
    double confidence;
    model->predict(test, prediction, confidence);
    v.id=prediction;
    v.confidence=confidence;
 return v;
}
```

Figura 6.4 Modul de recunoaștere

care decide dacă persoana din imagine face parte din baza de date sau este o persoană străina, dacă este o persoană străină se reia procesul, se introduce o nouă captură și se verifică din nou dacă persoana este din baza de date sau nu, acest lucru se repetă de 3 ori, dacă după a treia încercare persoana nu este recunoscută, sistemul se oprește. Dacă însă persoana se află în baza de date, se va afișa pe ecran un mesaj prin care robotul înștiințează persoana că a fost recunoscută cu succes, apoi acesta va primi o cartelă de acces pe care robotul o va lua folosind brațul robotic de care este atașat

un cleşte, toate unghiurile articulațiilor robotului inclusiv cleștele sunt programate folosind un API propriu reprezentat în figura de mai jos.

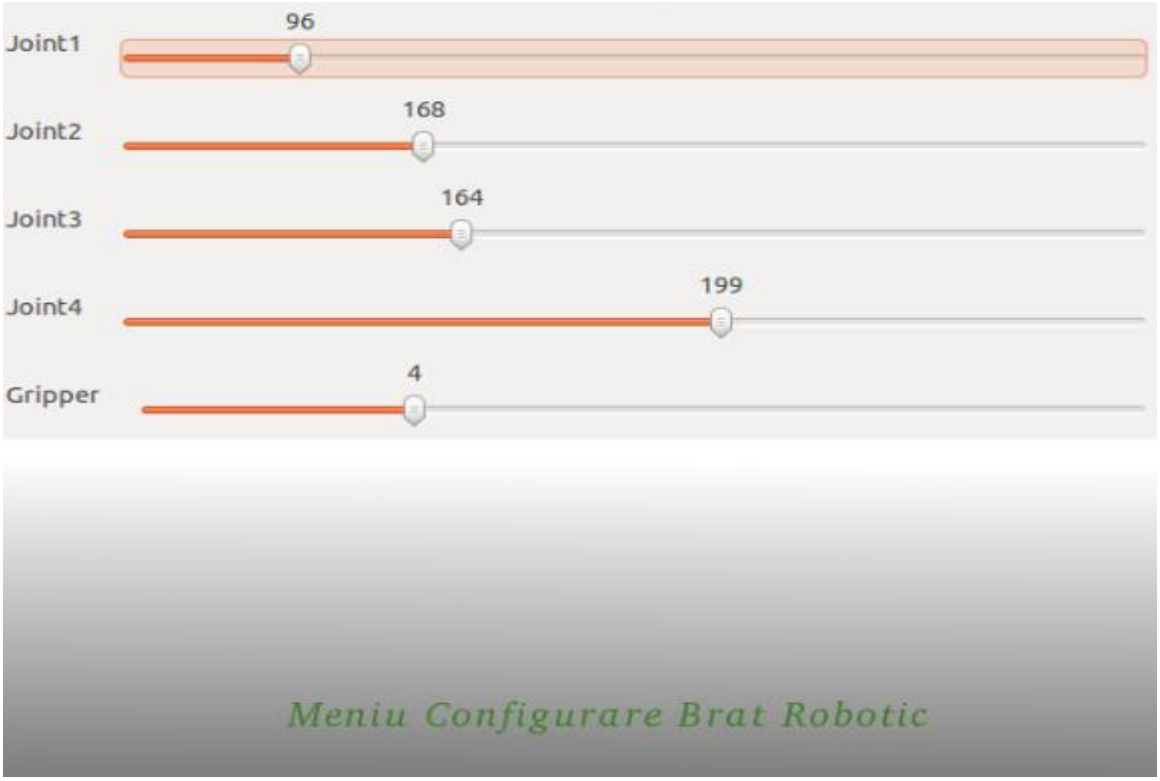



Figura 6.5 Aplicație de configurare a brațului robotic

Pentru a avea un grad de siguranța ridicat, că persoana a fost recunoscută cu succes se efectuează 6 teste înainte ca sistemul să dea un rezultat.

```
if (persoana.size()==6)
{
    for(int j=0;j<5;j++)
     {
        if(persoana[j+1].id==persoana[j].id) grd++;
        suma=suma+persoana[j].confidence;}
        medie=suma/5;
        persoana.clear();
        suma=0;
        //cout<<"GRD"<<grd<<endl;
    if (grd>=3 && medie<45)
            { cout<<"Confidenta"<<medie<<"Persoana"<<v.id<<endl;
              rejected=false;}
            else
            { cout<<"Confidenta"<<medie<<"Persoana"<<v.id<<endl;
              rejected=true;
              trial++;
              if (trial==3) exit(1);}
```

Figura 6.6 Verificare persoană

Dacă sistemul a recunoscut persoana ca fiind aceeași în 4 cazuri din 6 (aproximativ 67%) cu o confidență mai mică de 45, atunci se admite că persoana a fost recunoscută corect.

# Concluzii

La momentul actual sistemul este capabil să detecteze fața unei persoane aflată într-o mulțime și să recunoască aceea persoană cu un grad de acuratețe de până la 95% . După ce persoana este recunoscută ca făcând parte din baza de date încărcată în calculatorul integrat pe sistemul robotic, robotul este capabil să ofere persoanei o cartelă de acces prin rotația articulațiilor la un unghi setat apriori folosind interfața de configurare.

Avantajul major al acestui ansamblu este faptul că a putut fi portat pe un sistem cu resurse și putere de calcul limitată. Pentru realizarea sistemului am avut la dispoziție o cameră VGA de la un senzor de mișcare, un braț robotic și o platformă mobilă cu o unitate centrală integrată oferită de către laboratorul **CAMPUS - Center for Advanced Research on New Materials, Products and Innovative Processes - Robots – Autonomous and Adaptive Systems Lab** pe perioada desfășurării activității de pregătire a proiectului de licență.

Principala îmbunătățire a fost folosirea proprietății de căutare a celui mai mare obiect în imagine, astfel chiar dacă avem o sală plină de oameni, sistemul va detecta doar o singură persoană la un moment dat.

Posibile perspective pe viitor cu privire la acest sistem ar putea fi configurarea adaptivă a brațului robotic, astfel să poată identifica un obiect după culoare sau formă și folosirea rețelelor neuronale pentru detecția și recunoașterea facială.

# Bibliografie


[1] P. Viola și M. J. Jones, „Robust Real-Time Face Detection," *International Journal of Computer Vision,* 2004.

[2] P. Viola și M. Jones, „Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001,* vol. 1, pp. 511-518, 2001.

[3] J. Shlens, „A TUTORIAL ON PRINCIPAL COMPONENT ANALYSIS," 2003.

[4] P. N. Belhumeur, J. P. Hespanha și D. J. Kriegman, „Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 19, nr. 7, pp. 711-720, 1997.

[5] K. Deffenbacher, T. Vetter, J. Johanson și A. & O'Toole, „Facial aging, attractiveness, and distinctiveness," *Perception,* vol. 27, nr. 10, pp. 1233-1243, 1998.

[6] M. A. Turk și A. P. Pentland, „Face Recognition Using Eigenfaces," 1991.

[7] „THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS," *Annals of Eugenics,* 1936.

[8] M. Martinez și A. C. Kak, „PCA versus LDA," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 23, nr. 2, pp. 228-223, 2001.

[9] M. Welling, „Fisher Linear Discriminant Analysis," Toronto.

[10] T. Ojala, M. Pietikäinen și T. Mäenpää, „Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 24, nr. 7, pp. 971-987, 2002.

[11] y. wiki, „API architecture," [Online]. Available: http://www.youbot-store.com/wiki/index.php/API_architecture.

[12] *KUKA youBot User Manual,* Locomotec, 2012.

[13] A. Global, „Xtion PRO LIVE | 3D Sensor," [Online]. Available: https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/.

[14] J. Chang-yeon, „Face Detection using LBP features," 2008.

[15] T. Ojala, M. Pietikäinen și D. Harwood, „Pattern Recognition," în *A comparative study of texture measures with classification based on featured distribution.*, 1996, pp. 51-59.

ANEXA 1

```
#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include <OpenNI.h>
#include "./headers/Parsing.h"
#include "./headers/Image_processing.h"
#include <sstream>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <fstream>
#include "youbot_driver/youbot/YouBotBase.hpp"
#include "youbot_driver/youbot/YouBotManipulator.hpp"
#include <list>
#include <time.h>
using namespace cv;
using namespace std;
using namespace youbot;
// Camera streams
openni::VideoStream color;
//Calea catre modelele de detectie
const char *facePath="./Cascade/lbp_frontalface_alt2.xml";
const char *eyePath1="./Cascade/haarcascade_eye.xml";
//Model pentru recunoasterea faciala
Ptr<FaceRecognizer> model = createLBPHFaceRecognizer();
//Obiect care stocheaza identitatea si gradul de apartententa
struct pattern {
                              int id;
                              double confidence;
};
//Incarcare model de recunoastere
void learnLBPH(Ptr<FaceRecognizer> model, vector<Mat> img, vector<int> tagID)
{
                              model->load("./YML/LBPH_4pers_12.06.2017.yml");
}
//Module detectie
static void FaceDetect(Mat&,vector< Rect_<int> >&);
static void EyesDetect(Mat&,vector< Rect_<int> >&);
//Modul recunoastere
struct pattern recognize(Mat test,Ptr<FaceRecognizer> model)
{
                              struct pattern v;
                              int prediction=-1;
                              double confidence;
                              model->predict(test, prediction, confidence);
                              v.id=prediction;
                              v.confidence=confidence;
 return v;
}
//========================================================================
void init_youbot(bool& youBotHasBase, bool& youBotHasArm, YouBotBase* myYouBotBase,
                                                YouBotManipulator* myYouBotManipulator)
{
                              try
                              {
                               myYouBotBase->doJointCommutation();
                               youBotHasBase = true;
                              }
                              catch (std::exception& e)
                              {
                               LOG(warning) << e.what();
                               youBotHasBase = false;
```

```
                            }

                            try
                            {
                             myYouBotManipulator->doJointCommutation();
                             myYouBotManipulator->calibrateManipulator();
                             myYouBotManipulator->calibrateGripper();

                             youBotHasArm = true;
                            }
                            catch (std::exception& e)
                            {
                             LOG(warning) << e.what();
                             youBotHasArm = false;
                            }
                            GripperBarSpacingSetPoint gripperSetPoint;
                            gripperSetPoint.barSpacing=0.022*meter;
                            myYouBotManipulator->getArmGripper().setData(gripperSetPoint);
                            JointAngleSetpoint desiredJointAngle;
                            desiredJointAngle.angle = 3.03 * radian;//56244 * radian;
                            myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);

                            desiredJointAngle.angle = 1.04883 * radian;
                            myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);

                            desiredJointAngle.angle = -2.43523 * radian;
                            myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);

                            desiredJointAngle.angle = 3.0192 * radian;//1.73184 * radian;
                            myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
                            desiredJointAngle.angle = 2.96 * radian;//1.73184 * radian;
                            myYouBotManipulator->getArmJoint(5).setData(desiredJointAngle);
                            LOG(info) << "unfold arm";
                            SLEEP_MILLISEC(4000);
}
//======================================================================
void grab_key(YouBotManipulator *myYouBotManipulator,float arr[])
{
                            JointAngleSetpoint currentAngleOX;
                            GripperBarSpacingSetPoint gripperSetPoint;
                            gripperSetPoint.barSpacing=0.005*meter;
                            //currentAngleOX.angle =1.26221 * radians;
                            currentAngleOX.angle = arr[0]*radians;
                            myYouBotManipulator->getArmJoint(1).setData(currentAngleOX);
                            SLEEP_MILLISEC(2000)
                            //currentAngleOX.angle =1.65491 * radians;
                            currentAngleOX.angle = arr[1]*radians;
                            myYouBotManipulator->getArmJoint(2).setData(currentAngleOX);

                            SLEEP_MILLISEC(2000)
                            //currentAngleOX.angle =2.01049 * radians;
                            currentAngleOX.angle = arr[2]*radians;
                            myYouBotManipulator->getArmJoint(4).setData(currentAngleOX);
                            SLEEP_MILLISEC(2000)
                            //currentAngleOX.angle =-1.67229 * radians;
                            currentAngleOX.angle = arr[3]*radians;
                            myYouBotManipulator->getArmJoint(3).setData(currentAngleOX);
                            SLEEP_MILLISEC(2000)
                            //currentAngleOX.angle =2.95967* radians;
                            currentAngleOX.angle = arr[4]*radians;
                            myYouBotManipulator->getArmJoint(5).setData(currentAngleOX);

                            SLEEP_MILLISEC(2000)
                            myYouBotManipulator->getArmGripper().setData(gripperSetPoint);
}
//======================================================================
void give_key(YouBotManipulator *myYouBotManipulator)
```

```
{
                            JointAngleSetpoint desiredJointAngle;
    desiredJointAngle.angle = 2.03029 * radian;//1.73184 * radian;
                            myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
                            SLEEP_MILLISEC(1000)
                            desiredJointAngle.angle = 1.04883 * radian;
                            myYouBotManipulator->getArmJoint(2).setData(desiredJointAngle);
                            SLEEP_MILLISEC(1000)
                            desiredJointAngle.angle = -2.43523 * radian;
                            myYouBotManipulator->getArmJoint(3).setData(desiredJointAngle);
                            SLEEP_MILLISEC(1000)
                            desiredJointAngle.angle = 3.0192 * radian;//1.73184 * radian;
                            myYouBotManipulator->getArmJoint(4).setData(desiredJointAngle);
                            SLEEP_MILLISEC(1000)
                            desiredJointAngle.angle = 2.96 * radian;//1.73184 * radian;
                            myYouBotManipulator->getArmJoint(5).setData(desiredJointAngle);
                            desiredJointAngle.angle = 3.03 * radian;//56244 * radian;
                            myYouBotManipulator->getArmJoint(1).setData(desiredJointAngle);
                            SLEEP_MILLISEC(4000)
    LOG(info) << "Poftim Cartela";

}
//=================================================================================
int main(int argc, const char *argv[])
{
    string fn_csv = "./Csv_FINI.csv";
    //Cascade folosite in modulele de detectie
CascadeClassifier face_cascade;
CascadeClassifier eye_cascade;
                            bool youBotHasBase = false, youBotHasArm = false;
                            struct pattern v;
                            vector <pattern> persoana;
float joints[5]={1.26,1.7,2.01049,-1.67229,2.95967};
float joints1[5]={0.98,1.7,2.01049,-1.67229,2.95967};
                            YouBotBase* myYouBotBase = 0;
                            YouBotManipulator* myYouBotManipulator = 0;
                            myYouBotManipulator = new YouBotManipulator("youbot-manipulator",
"/opt/ros/hydro/share/youbot_driver/config");
                            myYouBotBase = new YouBotBase("youbot-base",
"/opt/ros/hydro/share/youbot_driver/config");
                            init_youbot(youBotHasBase,youBotHasArm, myYouBotBase,
myYouBotManipulator);

    vector<Mat> images;
    vector<int> labels;
    vector<label2string> names;

    try
    {
        read_csv(fn_csv, images, labels, names);
    }
    catch (cv::Exception& e)
    {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg << endl;
        exit(1);
    }

GripperBarSpacingSetPoint gripperSetPoint;
                            //Incarcare model de recunoastere faciala
                            learnLBPH(model,images,labels);

                            //initializare camera
                            openni::Device device;
                            openni::OpenNI::initialize();
                            device.open(openni::ANY_DEVICE);
                            color.create(device, openni::SENSOR_COLOR);
                            openni::VideoMode vm=color.getVideoMode();
```

```
                                //SLEEP_MILLISEC(20);
                                //getting camera resolution
                                int cols,rows;
                                //vm.setResolution(640,480);
                                cols=vm.getResolutionX();
                                rows=vm.getResolutionY();
                                int fps=vm.getFps();
                                cout<<cols << rows <<fps<< endl;
                                openni::VideoFrameRef vidFrame;
                                color.start();
                                //Initializare variabile;
                                double medie;
                                double suma = 0.0;
                                JointAngleSetpoint currentAngleOX, currentAngleOY;
                                int trial = 0;

                                vector< Rect_<int> > faces;
                                vector< Rect_<int> > eyes;
                                bool gotFaceandEyes=0;
                                Mat asteptare =
imread("/home/youbot/Projects/Databases/Validating/Loading.png");
                                Mat face;
                                Rect face_i;
                                int prediction;
                                double score;
                                static int nr = 1;
                                int grd=0;
                                Mat
persoana_necunoscuta=imread("/home/youbot/Projects/Databases/Validating/Unknown.jpg");
                                bool rejected;
                                for(;;)
                                {       char key = (char) waitKey(3);
                                 if(key==char(27))
                                 {
                                 destroyAllWindows();
                                 openni::OpenNI::shutdown();
    break;
                                 }

                                persoana_necunoscuta=imread("/home/youbot/Projects/Databases/Validatin
g/Unknown.jpg",1);
                                 openni::VideoFrameRef vidFrame;
                                 color.readFrame(&vidFrame);
                                 openni::RGB888Pixel* colorData =
(openni::RGB888Pixel*)vidFrame.getData();
                                 if (!colorData) continue;
                                 Mat color_frame(rows, cols, CV_8UC3, colorData);
                                 cv::resize(color_frame, color_frame, Size(640, 480), 1.0, 1.0,
INTER_CUBIC);
                                 Mat original = color_frame.clone();
                                 cvtColor(color_frame, color_frame, CV_BGR2GRAY);
//////////Detect face & eyes
                                 FaceDetect(color_frame,faces);
face_i = faces[0];
if (faces.size()==1)
                                EyesDetect(color_frame,eyes);
                                if (faces.size() && eyes.size()){
                                face = color_frame(face_i);
                                rectangle(original, face_i, CV_RGB(0, 255,0), 1);
        cv::resize(face, face, Size(100, 100), 1.0, 1.0, INTER_CUBIC);
                                face=tan_triggs_preprocessing(face);
                                 v=recognize(face,model);
                                 persoana.push_back(v);
                                 if (persoana.size()==6)
                                 {
                                        for(int j=0;j<5;j++)
                                         {
```

```
                                        if(persoana[j+1].id==persoana[j].id) grd++;
                                        suma=suma+persoana[j].confidence;}
                                        medie=suma/5;
                                        persoana.clear();
                                        suma=0;
                                        //cout<<"GRD"<<grd<<endl;
                                if (grd>=3 && medie<45)
                                                {
cout<<"Confidenta"<<medie<<"Persoana"<<v.id<<endl;
                                                        rejected=false;}
                                                else
                                                {
cout<<"Confidenta"<<medie<<"Persoana"<<v.id<<endl;
                                                        rejected=true;
                                                        trial++;
                                                        if (trial==3) exit(1);}

                                string box_text;
                                if (rejected==true)
                                                {
                                        int pos_x = std::max(face_i.tl().x - 10, 0);
                                        int pos_y = std::max(face_i.tl().y - 10, 0);
                                        box_text = format("Unknown user");
                                        imshow("face_recognizer",persoana_necunoscuta);
                                        waitKey(5);
                                        }
                                else
                                        {
                                                string wanted;
                                                for(int t = 0; t < names.size(); t++)
                                                        if(names[t].id == v.id)
                                                        {
                                                                wanted = names[t].name;
                                                                break;
                                                        }
                                int pos_x = std::max(face_i.tl().x - 10, 0);
                                int pos_y = std::max(face_i.tl().y - 10, 0);
                                box_text = format("Welcome, %s, please wait for your
keyCard!", wanted.c_str());
                                string path_to_user =
"/home/youbot/Projects/ProiectLicenta/prj/Camera/Validare/";
                                path_to_user = path_to_user + wanted + ".pgm";
                                Mat user = imread(path_to_user.c_str());
                                resize(user,user,Size(640,480),0,0,INTER_CUBIC);
                                putText(user, box_text, Point(100, 100), FONT_HERSHEY_PLAIN,
1.0, CV_RGB(0,255,0), 2.0);
                                imshow("face_recognizer", user);
                                waitKey(5);
                                if(wanted=="sorin")
                                {
                                //cout << "Soriin"<<endl;
                                grab_key(myYouBotManipulator,joints);
                                give_key(myYouBotManipulator);
                                }
                                else
                                {grab_key(myYouBotManipulator,joints1);
                                give_key(myYouBotManipulator);
                                }
                                gripperSetPoint.barSpacing=(double((15+3.5))/1000)*meter;
                                cout << "Voi elibera cartela in 3 secunde:"<<endl;
                                char key = (char) waitKey(3);
                                if(key==char(27))
                                        {
                                        destroyAllWindows();
                                        openni::OpenNI::shutdown();
                                break;}
                                myYouBotManipulator->getArmGripper().setData(gripperSetPoint);
```

```
                                        continue;
                                                }
                                        }
                                grd=0;
}
                                //show the video stream
                                imshow("face_recognizer", original);


                                if((char)27==waitKey(1))
            break;

    }

                                destroyAllWindows();
                                openni::OpenNI::shutdown();
    return 0;
}
//////////////////////////////
//Modul detectie
static void FaceDetect(Mat& sample,vector< Rect_<int> > &faces)
{
                                CascadeClassifier face_cascade;

                                face_cascade.load(facePath);
                                face_cascade.detectMultiScale(sample, faces,
1.1,4,CASCADE_FIND_BIGGEST_OBJECT);
                                return;
}
static void EyesDetect(Mat& sample,vector< Rect_<int> > &eyes)
{                               CascadeClassifier eye_cascade;
                                eye_cascade.load(eyePath1);
                                eye_cascade.detectMultiScale(sample,eyes, 1.1 , 2,
0|CV_HAAR_SCALE_IMAGE,Size(8,20));
                                return;
}
```

ANEXA 2

```
#include <fstream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sstream>
using namespace std;
using namespace cv;
struct label2string
{
        string name;
        int id;
};
//function for parsing a csv file and creating the image and label vectors
void read_csv(const string& filename, vector<Mat>& images, vector<int>& labels,
vector<label2string>& names, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please check the given filename.";
        CV_Error(CV_StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
                        if(labels[labels.size() - 1] != labels[labels.size() - 2])
                        {
                                int final = path.find_last_of('/');
                                int first = path.substr(0, final).find_last_of('/');
                                label2string aux;
                                aux.name = path.substr(first + 1, final - first - 1);
                                aux.id = atoi(classlabel.c_str());
                                names.push_back(aux);
                        }
                }
    }
}
```