Universitatea “Politehnica” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Sistem de recunoaștere și urmărire a unei ținte integrat în robotul Nao

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul *Inginerie Electronică și Telecomunicații*
programul de studii de licență *Electronică Aplicată*

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

Absolvent
Lisa-Marie SARU

2019

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

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **SARU C.D. Lisa-Marie , 443B**

1. Titlul temei: Sistem de recunoaștere și urmărire a unei ținte integrat în robotul Nao

2. Descrierea contribuției originale a studentului (în afara părții de documentare) și specificații de proiectare:
Scopul lucrării de diplomă este dezvoltarea unui sistem autonom de recunoaștere a unei ținte pre-definite (template) și urmărirea acesteia de către un robot Nao. Robotul Nao este caracterizat de o serie de constrângeri hardware și software (de exemplu, memorie limitată, capacitate pentru prelucrarea datelor relativ redusă). Prin aceasta lucrare, se dorește extinderea autonomiei robotului NAO pentru a fi ulterior utilizat în diverse aplicații de interacțiune în timp real cu utilizatorii. În cadrul proiectului de diplomă, vor fi propuși 2-3 algoritmi de identificare a unui template. Pentru a respecta constrângerile hardware și software impuse de utilizarea robotului NAO, studentul va realiza o serie de experimente astfel încât sistemul să funcționeze pe capturi video în timp real, realizate folosind camera video integrată în robotul NAO. Urmărirea obiectului presupune deplasarea robotului astfel încât ținta urmărită să fie păstrată în centrul cadrelor video. Astfel, robotul NAO va fi programat să efectueze anumite mișcări în funcție de poziția obiectului urmărit. Algoritmii de identificare a țintei și software-ul pentru deplasare vor fi implementați în Python și se va folosi librăria OpenCV.

3. Resurse folosite la dezvoltarea proiectului:
Limbaj de programare: Python 2.7 Biblioteci: OpenCV 3.3, numpy, matplotlib, naoqi Mediu de dezvoltare: Spyder, Choreograph

4. Proiectul se bazează pe cunoștințe dobîndite în principal la următoarele 3-4 discipline:
DEPI, PDS, IM, SDA

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

6. Data înregistrării temei: 2018-11-29 14:43:46

Conducător(i) lucrare,
Prof. dr. ing. Corneliu BURILEANU
semnătura: .....

Rădoi Anamaria
semnătura: .....

Student,
semnătura:.....

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

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

Cod Validare: **33ac404fec**

Anexa 5

Declarație de onestitate academică

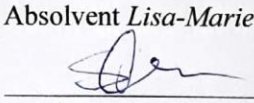
Prin prezenta declar că lucrarea cu titlul *"Sistem de recunoaștere și urmărire a unei ținte integrat în robotul Nao"*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *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, *23.06.2019*

Absolvent *Lisa-Marie SARU*



(semnătura în original)

Cuprins

LISTA FIGURILOR

LISTA TABELELOR

LISTA ACRONIMELOR

API – Application Programming Interface
RISC – Reduced Instruction Set Computer
RAM – Random Access Memory
SDHC – Secure Digital High Capacity
LED – Light Emitting Diode
USB – Universal Serial Bus
YUV – (Y) Luminanță, (U) Lățimea benzii, (V) Cromatică
GMM – Gaussian Mixture Model
MAP – Maximum A-Posteriori
EM – Expectation Maximization
RGB – Red Green Blue

CAPITOLUL 1

Introducere

1.1 MOTIVAȚIA

De-a lungul ultimilor ani, robotica medicală a avut parte de o dezvoltare accelerată, ajutând persoane atât cu probleme fizice, cât și cu probleme psihice. Pentru multe dintre acestea s-au găsit idei inovatoare de a introduce diverse ajutoare robotice care să contribuie la reabilitarea pacienților. Tulburările de spectru autist la copii se numără printre problemele vizate, astfel că s-au dezvoltat diverși roboți care pot fi introduși în ședințele de terapie pentru a facilita învățarea prin joacă, de a dezvolta procesele cognitive și de asemenea, pentru a-i ajuta pe copii să se orienteze în spațiu.

Principalul scop al proiectului meu este de altfel realizarea unei aplicații pe care să o atribui robotelului umanoid Nao și care să contribuie la creșterea gradului de autonomie împreună cu alte funcționalități. Robotul are deja câteva module implementate care facilitează un demers bun în ședințele de terapie sau chiar și în cadrul unor clase educative, însă acestea nu sunt destule pe un termen lung de timp. Modulele sale destul de limitate sunt suficiente cât să stârnească interesul copiilor, însă nu sunt de ajuns cât să permită libertatea de a însuși o gamă variată de idei.

Prin robotul Nao se dorește dezvoltarea unui sistem inteligent capabil să se integreze în lumea medicală și educațională și să se adapteze oricărei noi cerințe cu ușurință. Realizarea unui program de urmărire a unui obiect pe care îl înregistrăm pe loc este un prim pas care contribuie la dezvoltarea inteligenței artificiale pe Nao. El va avea în acest mod un comportament plin de viață care să îl facă și mai îndrăgit de către copii și care să le sporească atenția și abilitățile de integrare în societate.

1.2 ROBOȚI UMANOIZI ÎN DOMENIUL MEDICAL

1.2.1 Aspecte generale

Ne aflăm într-o etapă crucială în ceea ce privește domeniul roboticii. În fiecare zi, noi descoperiri și inovații aduse de oamenii de știință ne împing către un viitor în care din ce în ce mai multe sarcini atribuite nouă sunt preluate de către roboți. Procesul de automatizare nu este un concept nou, ci mai degrabă o dezvoltare la nivel de tehnologie care vine în ajutorul nostru și care este într-o continuă expansiune.

Printre domeniile vizate în care tehnologia ia din ce în ce mai multă amploare se numără și medicina, unde roboții autonomi se pot integra ușor printre personalul spitalelor sau pot prelua diverse sarcini care vin în ajutorul medicilor.

Un robot umanoid este o mașină care nu doar arată ca o ființă umană, ci este și capabilă să comunice atât cu noi, cât și cu alte sisteme, să interpreteze date preluate din mediul exterior și să realizeze diverse sarcini. [1][2]

1.2.2 Aplicații ale roboților

Aplicațiile unui robot umanoid sunt bine stabilite în cadrul educației și medicinei. Des întâlnite sunt cazurile în care aceștia sunt folosiți în tratarea copiilor care suferă de autism sau pot diminua nivelul de stres al celor care suferă de cancer sau paralizie cerebrală. Pe de altă parte, în educație, acești roboți motivează elevii nu doar să învețe, ci și să participe în mod activ la ore.

Roboții care servesc în domeniul medical sunt folosiți de pacienți fie acasă, fie în centrele medicale pentru a-și trata și îmbunătăți problema de sănătate. Astfel de roboți necesită intervenția unei persoane autorizate (medic, terapeut) pentru a putea fi controlați sau sunt deja programați de către ingineri îndrumați de specialiști pentru a efectua anumite programe de terapie.

Printre cele mai importante responsabilități ale unui robot umanoid într-o clinică se numără ameliorarea stării de tristețe, monitorizare de la distanță și interacțiunea cu pacienții. De exemplu, s-a observat că atunci când copiii care sunt expuși unei dureri pe durata unei proceduri medicale o tolerează mai bine dacă sunt asistați de un robot în cabinet deoarece atenția acestora este distrasă și se simt încurajați. În alte cazuri, scopul roboților a fost să reducă nivelul de anxietate, furie sau depresie atunci când pacienții trebuie să treacă prin procedura unui tratament.

Roboți în terapiile pentru tulburări de spectru autist

Una dintre caracteristicile tulburării de spectru autist este deficitul atenției atunci când un copil trebuie să urmărească unul sau mai multe obiecte care se pot mișca și pe una sau multe traiectorii. Această sarcină este realizată într-un mod mult mai lent sau deloc în unele cazuri, iar pentru a-i ajuta pe copii să fie capabili de un astfel de lucru este destul de dificil pentru terapeuți, iar metodele utilizate nu au tot timpul randament mare.

Atenția unui astfel de pacient este mai greu de atras, însă odată cu introducerea tehnologiei în cadrul terapiilor s-a observat că cei mici sunt mult mai atrași de un robot și mai motivați să îndeplinească sarcinile primite de către terapeuți. În unele cazuri, ei trebuie să imite diverse gesturi, așa că un robot ar fi mult mai stimulant decât o persoană și ar învăța mult mai repede.

O aplicație care vine în sprijinul problemei deficitului de atenție este cea prezentată în proiectul de față care va îndruma pacientul să urmărească diverse obiecte și să se apropie sau să se depărteze de acestea în funcție de distanța la care se află de ele. În acest mod, copilul se va putea orienta mult mai bine în spațiu și va învăța să se descurce singur în anumite situații din viața de zi cu zi. Tot cu ajutorul acestei aplicații, se vor putea dezvolta și abilitățile sale de cunoaștere. Terapiile se vor baza astfel pe învățare și dezvoltare prin joacă deoarece roboții pot fi percepuți ca jucării sau chiar și ca prieteni datorită interacțiunii cu aceștia.

Pe lângă această aplicație mai există și alte metode prin care roboții pot contribui la ședințele de terapie, precum redarea unor cântece, efectuarea unor mișcări de dans sau simple gesturi ale corpului, chiar și mici jocuri. Copii se vor putea dezvolta astfel din punct de vedere mintal și vor deveni mai sociabili.

Roboți în educație

Roboții umanoizi din domeniul educațional asistă în general la cursuri profesorii de care pot fi comandați. Aceștia stârnesc mai mult interesul elevilor și îi determină să fie mai receptivi li să participe activ pe durata cursurilor deoarece ei sunt curioși și atrași de orice este nou. [2][3]

1.2.3 Robotul Nao

Nao este un robot umanoid, autonom și programabil dezvoltat inițial de către firma franceză Aldebaran Robotics. Câteva versiuni ale acestui robot au fost scoase pe piață încă din anul 2008 pentru a servi în scopuri ce țin de cercetare și educație venind ca un ajutor pentru numeroase universități și laboratoare de cercetare din întreaga lume.

În vara anului 2010, Nao a devenit celebru la Shanghai Expo în China printr-o demonstrație de dans sincronizat, iar în toamna aceluiași an, Universitatea din Tokyo a achiziționat 30 de astfel de roboți cu scopul de a-i integra în domeniul educației pentru a activa în cadrul laboratoarelor.

În decembrie 2011, Aldebaran a scos un nou prototip de robot Nao, de data aceasta însă mai performat atât din punct de vedere hardware, cât și software. Printre aceste îmbunătățiri se numără o cameră video de mai bună calitate, un sistem anti-coliziune mai avansat și o construcție mult mai rezistentă.

Robotul Nao Evolution, care a apărut în anul 2014, a venit cu o sinteză vocală în mai multe limbi, un algoritm de detecție și recunoaștere a formelor și trăsăturilor faciale și un sistem de recunoaștere a unei surse de sunet cu ajutorul a patru microfoane.

Firma Aldebaran Robotics a fost achiziționată în 2015 de către cei de la SoftBank Group și redenumită ca SoftBank Robotics. Implicit proiectul a continuat să ia amploare și ulterior au fost creați și alți roboți.

Nao și-a continuat evoluția și în plan medical, fiind din ce în ce mai popular printre medici, terapeuți și ingineri datorită ajutorului acordat copiilor cu dizabilități psihice. Acesta este unealta ideală pentru a stimula capacitățile cognitive, orientarea și abilitățile sociale ale unor astfel de pacienți. [4]

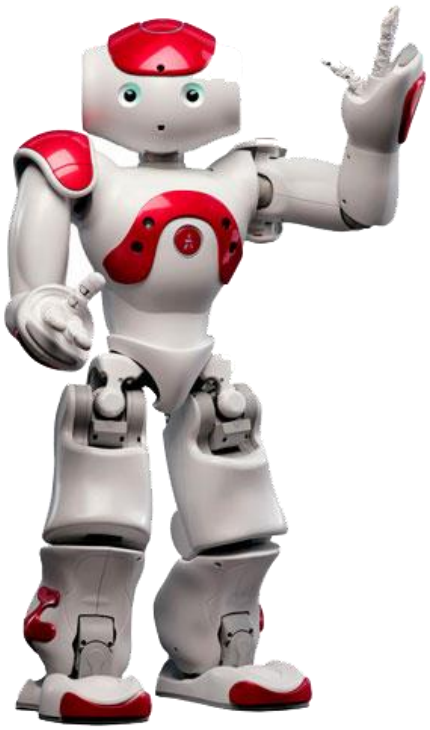


Figura 1.1 *– Aspectul exterior al robotului Nao [19]*

1.2.4 Rezultate obținute de-a lungul timpului

Terapiile cu astfel de roboți s-au dovedit a avea rezultate vizibile spre deosebire de cele în care nu s-a utilizat interacțiunea dintre pacient și robot. Unul dintre principalele lucruri care face diferența este că sesiunea de lucru cu persoanele în cadrul terapiilor poate deveni copleșitoare pentru un copil cu probleme, ceea ce duce la îngreunarea ședințelor, pe când folosirea unui robot este mai puțin sufocantă pentru ei datorită comportamentului mai rezervat și a elementului de tehnologie inclus.

Rezultatele după astfel de terapii pot fi vizibile după 20 de ședințe săptămânale, copiii îmbunătățindu-și deja o parte dintre abilități precum formularea întrebărilor și a unor cereri către apropiați. Făcând o comparație cu cei care nu au lucrat cu un robot pe parcursul terapiei, s-a observat o evoluție mai rapidă. [5]

1.3 METODA PROPUSĂ

Urmărirea unei ținte pare la prima vedere un lucru banal. Pentru noi ca oameni este o sarcina de rutină să localizăm un obiect sau o persoană, să o urmărim în timp ce se află în mișcare și să ne dăm seama cât de aproape sau departe se află de noi. Dorind să realizăm acest lucru cu ajutorul unui robot, ne lovim de câteva întrebări care ne ajută să ne dăm seama ce e de făcut pentru implementarea unui astfel de program, și anume: „Cum știm ceea ce vrem să urmărim?”, „Cum detectăm obiectul?”, „ Cum ne mișcăm?”.

În primul rând, pentru a stabili resursele necesare putem face o simplă analogie cu omul. Pentru a fi conștient de mediul înconjurător, o persoană are nevoie de simțuri (văz, aud, miros etc.). În cazul prezentat, robotul ar trebui să dispună de o camera video de la care să preia date și pe care să le prelucreze pentru a putea lua o decizie. De asemenea, dacă procesul urmăririi implică și anumite mișcări are robotului, trebuie avut în vedere capacitățile motorii și spațiul de desfășurare.

De exemplu, dacă vrem să fie urmărită o minge în mișcare, trebuie ca robotul să știe că este vorba despre o minge și nu altceva, așa că putem vorbi despre o captură a obiectului respectiv care poate fi folosită ca referință (caz particular) sau antrenarea unor clase de obiecte pe care robotul să le recunoască și ulterior să le urmărească, introducând astfel și concepte de inteligență artificială și rețele neuronale. După acest pas urmează detecția obiectului la fiecare cadru video realizat, urmând mișcarea robotului după țintă.

În al doilea rând, gama de roboți pe care s-ar putea implementa un astfel de algoritm este variată, motiv pentru care programatorul trebuie să aibă în vedere capacitățile și resursele sistemului cu care lucrează. În proiectul de față s-a folosit un robot umanoid Nao, răspândit și folosit în întreaga lume. Acesta este unealta ideală, scopul lui fiind întocmai să fie integrat în terapiile pentru copii cu probleme de sănătate psihică.

Proiectul în sine are în vedere crearea unei aplicații care să contribuie la implementarea unui întreg sistem inteligent pe Nao. Contribuția mea a fost să dezvolt câțiva algoritmi cu ajutorul cărora robotul să poată extrage un obiect din imagine și să știe că pe acela va trebui să îl urmărească. Pașii după care m-am ghidat au avut la bază ideile menționate mai sus în care am făcut analogie cu omul pentru a putea înlănțui logic ideile de care aveam nevoie pentru a duce la bun sfârșit scopul propus, ținând cont în același timp și de resursele puse la dispoziție.

CAPITOLUL 2

Resursele Nao

2.1 CARACTERISTICI GENERALE

Așa cum am menționat și mai sus, Nao este printre cei mai populari roboți umanoizi din lume și și totodată și cel mai folosit în educație și medicină pediatrică. El are un aspect prietenos, formele sale rotunde fiind gândite cu atenție pentru ca acesta să fie potrivit pentru copii. De asemenea nu este nici foarte mare, având o înălțime de 58 cm și o lățime de 28 cm cântărind aproximativ 4.5 kg.

Nao este o unealtă puternică în cele două domenii deoarece acesta oferă posibilitatea utilizatorilor să îl programeze și să îl controleze cu ușurință, putând fi astfel folosit în diverse activități medicale și educative. El a fost conceput să poată fi personalizat în funcție de aplicația dorită. Cu ajutorul resurselor hardware și software de care dispune, acesta se integrează foarte ușor în medii închise putând detecta și evita obstacole, luând decizii pe baza datelor de la senzori și realizând tot felul de mișcări, de la simpla acțiune de a merge și până la păstrarea echilibrului.

Lucrându-se în prezent cu cea de-a cincea versiune și fiind un robot cu caracteristici performante, Nao rămâne un sistem embedded, ceea ce înseamnă că nu are aceeași putere de procesare și resursele unui calculator. Aplicații avansate precum cele ce impun sisteme de recunoaștere a sunetelor sau detecție de obiecte necesită programe ample cu algoritmi complecși crescând astfel gradul de autonomie al robotului. Se dorește astfel crearea unui sistem inteligent care să valorifice inteligenta artificială.

2.2 RESURSE HARDWARE

Nao, deși este un robot umanoid de dimensiuni relativ mici, este capabil să realizeze o gamă variată de mișcări și este destul de rezistent cât să nu fie distrus în urma unei căderi produse din picioare la nivelul podelei. Toate acestea și multe altele sunt datorate numeroaselor resurse ale sale prezentate în *Figura 2.1*.

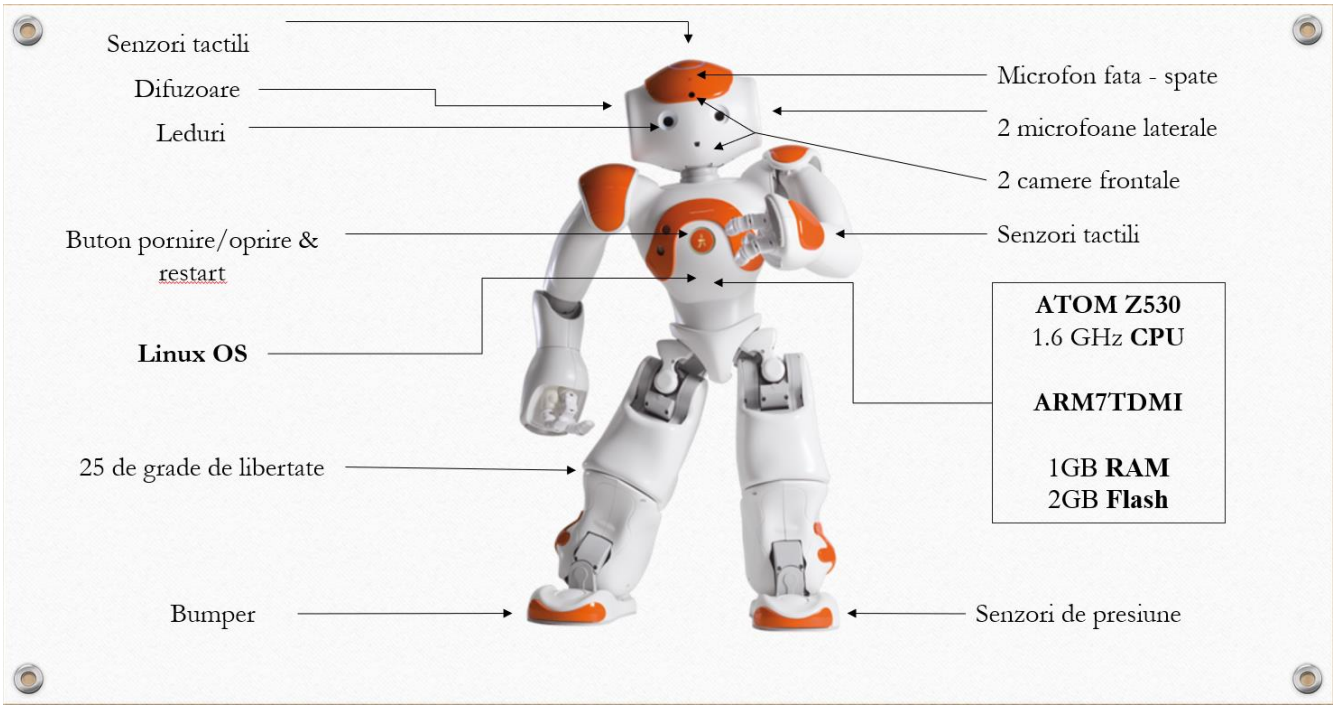


Figura 2.1 *– Aspectele hardware ale lui Nao*

Bateria

Bateria este din litiu-ion, având o tensiune nominală de 21.6 V, autonomie de 90 de minute în condiții de uz normal, iar durata de încărcare este de 3 ore. Aceasta poate fi încărcată la maximum 25.2V și are o energie de 48.6 Wh. Nao poate fi folosit și în timp ce este în priză. [6]

Procesoare

Cea de-a patra versiune a lui Nao (cel utilizat de mine) dispune de două procesoare, unul aflându-se la nivelul capului de tipul Intel x86 de tipul ATOM Z530 și unul situat la nivelul toracelui, ARM7TDMI, care este responsabil de controlul actuatorilor. Comunicarea dintre cele două procesare se face cu ajutorul a două magistrale de tipul RS-485.

Procesorul ATOM Z530

Fiind un procesor RISC, acesta prezintă un set de instrucțiuni pe 32 de biți. Prezintă o viteză de ceas de 1.6 GHz, reușind să execute două instrucțiuni într-un ciclu de ceas. De asemenea, poate face o translație de instrucțiuni complexe în operații simple înainte de a fi executate. Dispune de o memorie RAM de 1 GB și o memorie Flash de 2GB, având în același timp și posibilitatea adăugării unui card de memorie de maximum 8 GB Micro SDHC.

Procesorul are un singur nucleu, fiind cel mai des întâlnit la telefoanele mobile avantajul său fiind consumul redus de energie.[4]

Procesorul ARM7TDMI

Este tot un procesor de tip RISC cu un set de instrucțiuni de 32 de biți oferind aceleași performanțe pentru un consum cât mai mic de putere. El are sarcina de a transmite modulelor actuatorilor biții de control ce provin de la microcontrolerele locale. [4]

Motoare – Actuatori

Pentru a putea realiza mișcări cât mai apropiate de cele ale unui om, au fost necesare 25 de motoare care au fost plasate la nivelul articulațiilor. Ele au o viteză pe care utilizatorul o poate reconfigura ori de câte ori este nevoie.

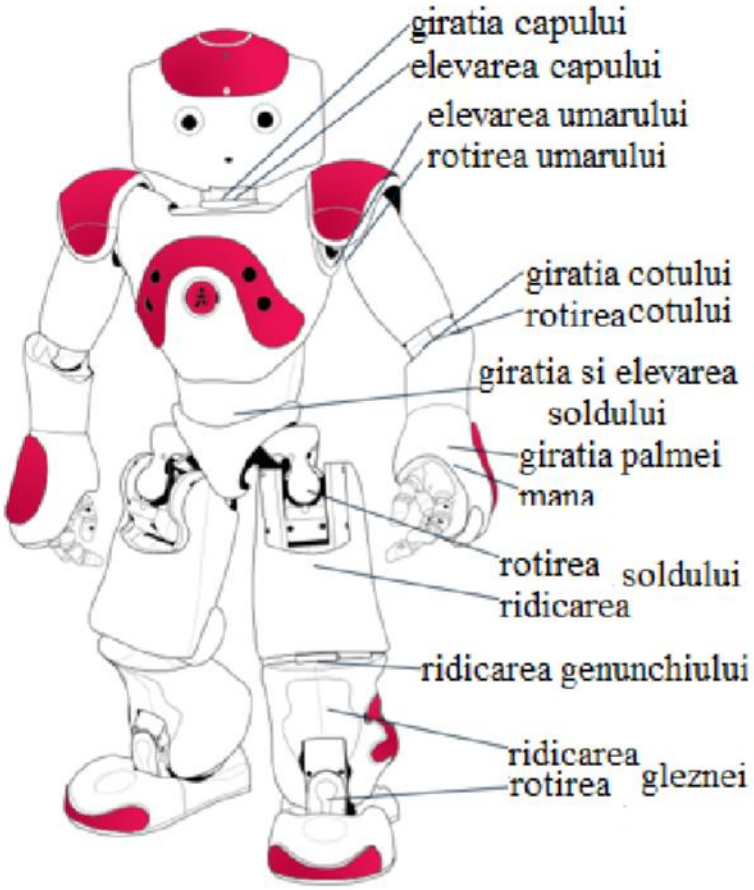


Figura 2.2 *– Articulațiile lui Nao [20]*

Robotul are integrate trei tipuri de motoare. Cele de tipul 1 sunt folosite pentru picioare fiind mai puternice, iar restul corespund mâinilor gâtului și articulațiilor.

Gradele de libertate determină starea unui sistem cu ajutorul unor mărimi scalare independente. Astfel, pentru a se putea preciza starea unui sistem este nevoie de cel puțin un număr egal cu numărul gradelor de libertate.

Nao prezintă 25 de grade de libertate, dintre care 14 sunt dedicați părții superioare trunchiului și restul de 11 corespund părții inferioare. [8]

Localizare	Grade de libertate	Observații
Cap	2	Capul se poate mișca stânga-dreapta, sus-jos
Brațe	10	Amplasare a câte 5 în fiecare mână
Mâini	2	Câte unul pentru fiecare mână pentru a apuca lucruri
Pelvis	1	-
Picioare	10	Amplasare a câte 5 în fiecare picior

Tabel 2.1 **-** Dispunerea gradelor de libertate ale lui Nao

Sistemul audio

Pe lângă multe alte facilități de care dispune, Nao este prevăzut și cu un sistem audio stereo format din două difuzoare plasate pe părțile laterale ale capului astfel încât să poată exista o interacțiune la nivel audio. Din același motiv, au fost integrate și 4 microfoane dispuse tot pe cap care au o frecvență cuprinsă în intervalul 300Hz-8KHz. [9][10]

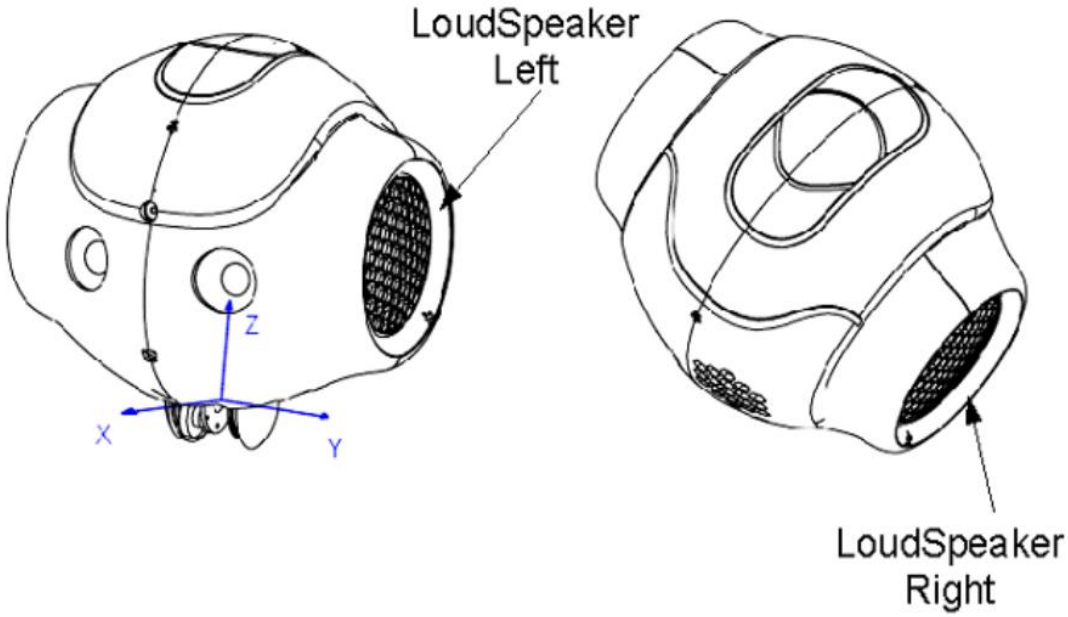


Figura 2.3 *– Amplasarea difuzoarelor [9]*

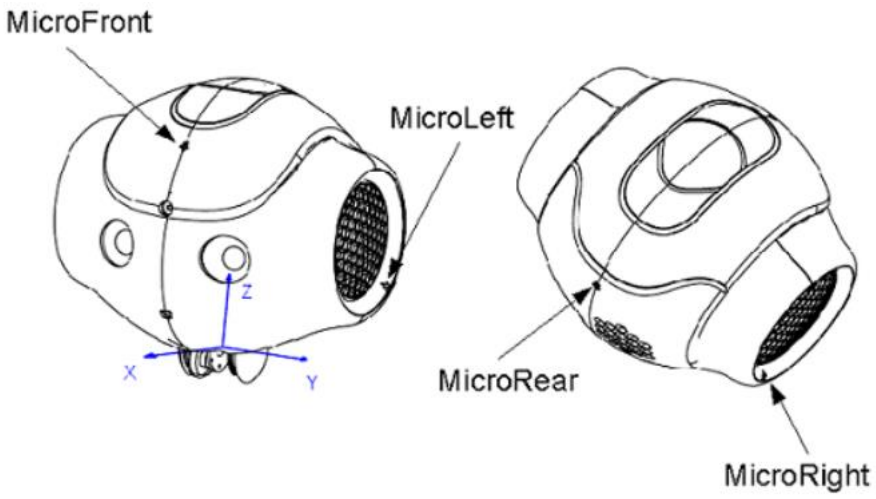


Figura 2.4 *– Amplasarea Microfoanelor [10]*

Sistemul video

La nivelul capului sunt dispuse 2 camere video frontale care au aceleași specificații și furnizează imagini cu rezoluția 1280x960 și care capturează 30 de cadre pe secundă. Acestea nu pot funcționa simultan, iar la începutul fiecărei aplicații se pornește cea de sus și poate fi schimbată dacă este nevoie. În general, a doua se folosește atunci când este necesar să fie identificat spațiul de deplasare al robotului. [11]

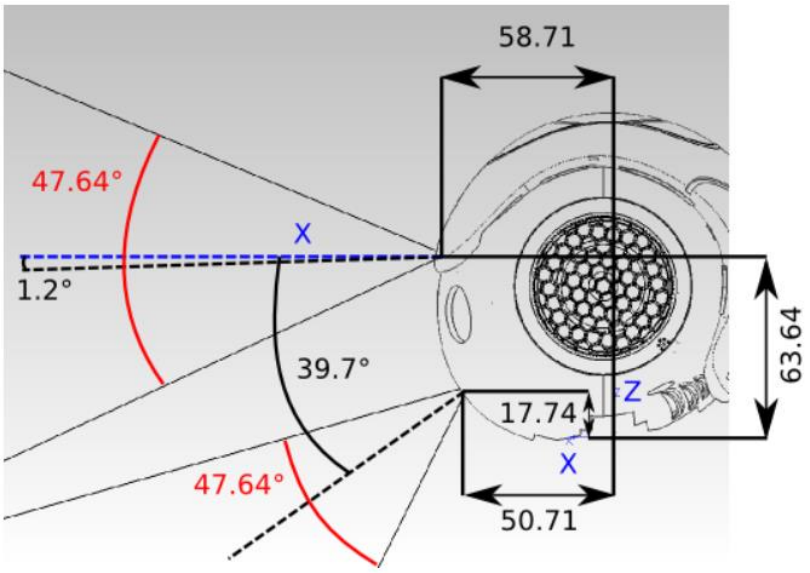


Figura 2.5 *– Apertura camerelor video în plan vertical [11]*

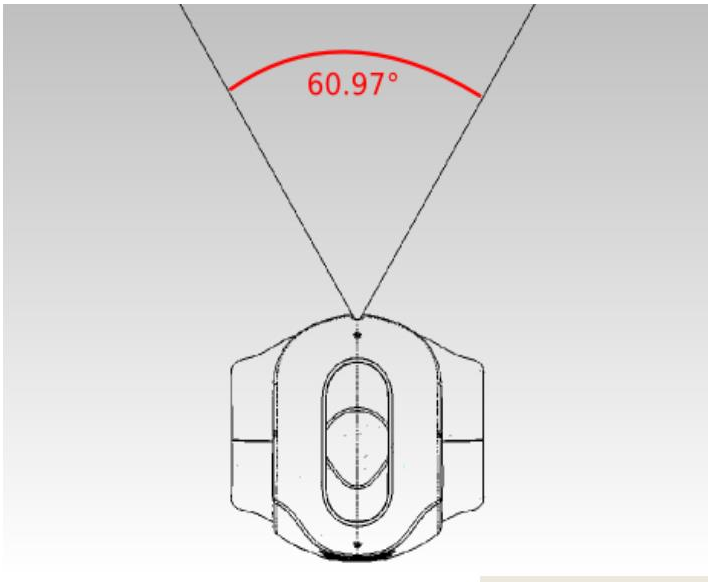


Figura 2.5 *– Apertura camerelor video în plan orizontal [11]*

LED-uri

Pentru ca orice activitate cu Nao să fie interactivă, pe corpul acestuia sunt plasate mai multe LED-uri conform *Tabelului 2.2* de mai jos:

Localizare	Număr de LED-uri	Detalii
A. Cap	12	16 nivele de albastru
B. Urechi	20	16 nivele de albastru
C. Ochi	16	Tot spectrul RGB
D. Piept	1	Tot spectrul RGB
E. Picioare	2	Tot spectrul RGB

Tabel 2.2 *- LED-urile integrate pe Nao [20]*

Figura 2.7 *– Dispunerea LED-urilor lui Nao [20]*

Senzori

În partea superioară a capului sunt prezenți 3 senzori tactili, iar pe fiecare mână se află plasat câte un senzor tactil. Pentru activarea lor, este necesară programarea robotului astfel încât aceștia pot fi personalizați în funcție de cerințe.

Senzorii tactili și cei de contact sunt amplasați după cum urmează:

- Trei senzori tactili la nivelul capului
- Un senzor tactil pe piept
- Doi senzori tactili pe mâini
- Doi pe picioare

Rolul lor este de a sesiza când robotul a fost atins și pentru a-l proteja de alte obiecte.

Conectivitate

Pe Nao vom găsi două protocoale de comunicație: Ethernet și Wi-Fi. Pentru primul, robotul este dotat cu o mufă la nivelul capului și este cel mai des utilizată atunci când vrem să setăm conexiunea prin Wi-Fi. De asemenea, Nao dispune și de un port USB la care se pot conecta diverse dispozitive.

2.2. RESURSE SOFTWARE

Nao dispune de o gamă explicită de resurse care îl fac pe acesta să funcționeze din punct de vedere software, cu dezavantajul că acestea sunt limitate din punct de vedere al dezvoltării unor aplicații cu un grad mare de complexitate. Există însă metode prin care se pot implementa resursele lipsă care vin în ajutorul realizării unui sistem inteligent.

2.3.1 Framework-ul NAOqi

Pentru a putea face posibil accesul la modulele robotului, audio, video sau cele de mișcare, cei de la Aldebaran au creat un framework pentru robot ținând cont de paralelism, sincronizare și declanșarea evenimentelor. În acest mod, utilizatorul poate lucra eficient în timp ce acționează mai multe resurse.

Pentru optimizarea sa, a fost integrată introspecția astfel încât să fie posibilă căutarea modulelor necesare unor procese într-o manieră eficientă. Framework-ul va ști ce module sunt disponibile și unde să le caute. Tot datorită framework-ului, modulele pot comunica între ele, pot fi folosite în programarea robotului și informația poate fi distribuită.

Executabilul (Broker-ul) pentru NAOqi de pe robot funcționează ca un mediator. La pornire, se încarcă un fișier autoload.ini ce definește bibliotecile care vor trebui încărcate, fiecare dintre ele având una sau două module care folosesc mediatorul pentru a-și lista metodele.

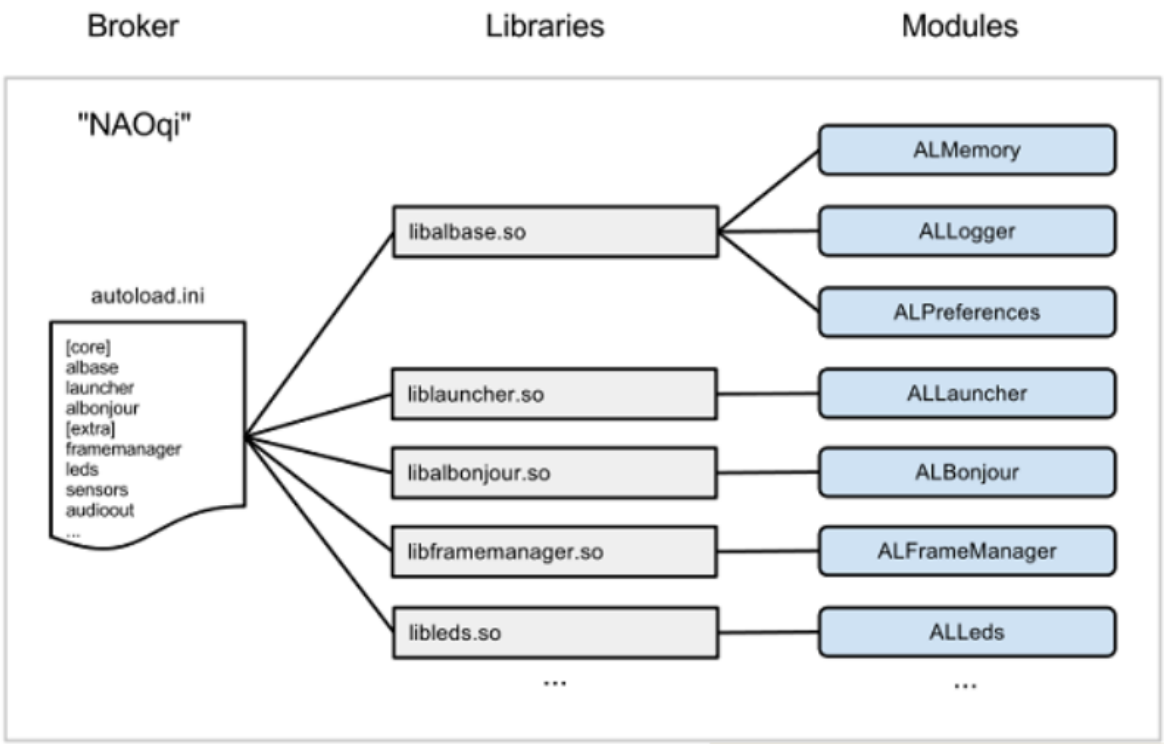


Figura 2.8 *– Relația dintre executabil, biblioteci și module [12]*

Scopul mediatorului este să încarce bibliotecile necesare și să asigure accesul la rețea pentru ca metodele să fie chemate. Toate acestea sunt realizate printr-un proces transparent în sensul că permit utilizatorului să programeze robotul într-o manieră obișnuită atunci când sunt apelate module locale.

Proxy-ul este un obiect care poate fi instanțiat cu un modul și care va putea apela metodele pe care le conține modulul respectiv. De exemplu, daca se creează un proxy pentru modulul ALMotion, vom avea un obiect care ne permite accesul și utilizarea tuturor metodelor implementate de mișcare și control al membrelor robotului.

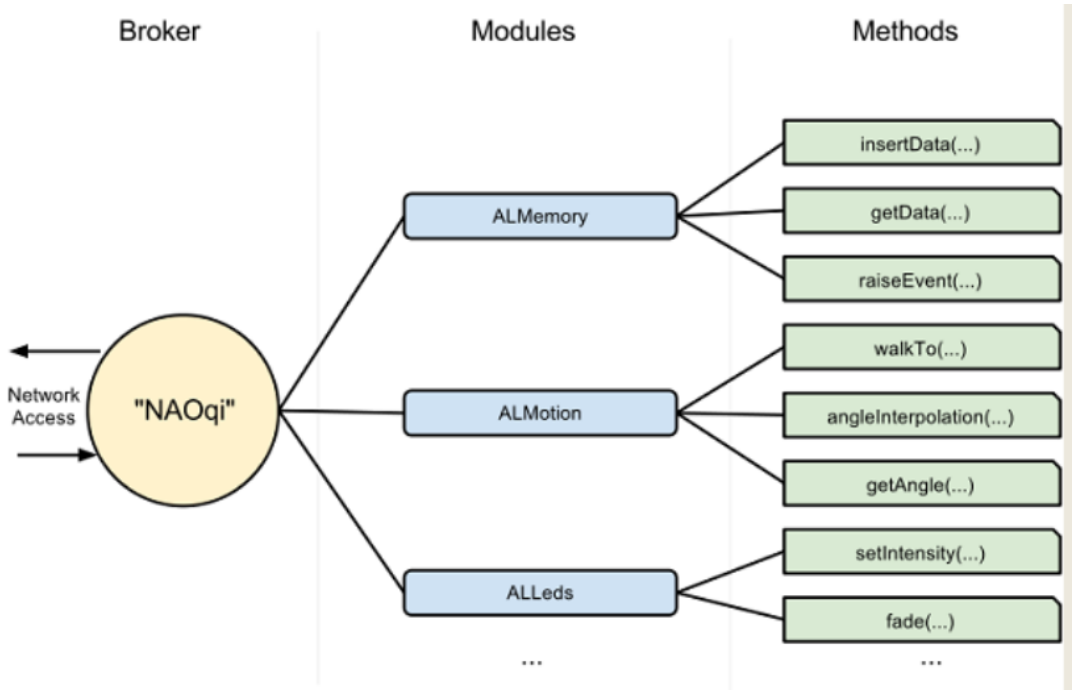


Figura 2.9 *– Relația dintre executabil, module și metode [12]*

Modulele sunt clase ce aparțin unei librării, iar când aceasta este încărcată, în mod automat modulele vor fi instanțiate. Fiecare are un anumit număr de metode, dintre care unele pot fi chemate și din afara modulului.

API-urile pentru modulele NAOqi:

- Modulul de mișcare conține metode care pot fi folosite cu scopul de a-l face pe Nao să își folosească toate membrele și încheieturile.
- Modulul audio oferă posibilitatea utilizării unor funcții ce permit redarea sau înregistrarea unor fișiere audio. În același timp Nao pune la dispoziție utilizatorului funcții de recunoaștere a vorbirii și de a transpune orice text în voce.
- Modulul ce face legătura cu senzorii are în vedere toate caracteristicile care țin de interacțiunea cu mediul înconjurător, și anume senzorii tactili, bateria, LED-urile și senzorii de presiune.
- Sistemul vizual conține module prin care cele două camere video pot fi accesate și le pot fi modificate caracteristicile. Totodată, Nao are funcții ce permit recunoașterea a unor fizionomii, imagini care pot fi înregistrate cu camera. [12]

2.3.2 NAOqi Vision

Scopul celor două camere video ale robotului este de a-i permite acestuia să obțină informații din mediul exterior și să ia decizii pe baza a ceea ce s-a găsit. Pentru a putea fi prelucrate datele, framework-ul conține câteva module care lucrează în concordanță cu cele două camere.

În tabelul de mai jos sunt enumerate toate modulele categoriei sistemului video:

Nume modul	Descriere
ALBacklightingDetection	Folosit pentru luminozitatea cadrului imaginii
ALDarknessDetection	Folosit pentru a detecta dacă mediul înconjurător este întunecat
ALFaceDetection	Abilitatea de a detecta și de a recunoaște fizionomii
ALLandmarkDetection	Abilitatea de a detecta niște puncte de reper specifice
ALMovementDetection	Abilitatea de a detecta mișcarea și de a spune de unde vine
ALPhotoCapture	Unealtă pentru a realiza capturi fotografice
ALRedBallDetection	Abilitatea de detecta obiecte roșii circulare
ALVideoDevice	Administrează intrările video
ALVideoRecorder	Permite înregistrarea video
ALVisionRecognition	Abilitatea de a învăța și de a recunoaște anumite șabloane vizuale
ALVisualCompass	Abilitatea de a folosi o imagine ca pe o busolă
ALVisionToolbox	Unealtă ce permite înregistrarea sau analizarea unor imagini

***Tabel 2.3** - Modulele aferente sistemului video [13]*

2.3.2.1 Modulul de urmărire al unei mingii roșii

ALRedBallDetection este un modul ce permite robotului să detecteze și să urmărească orice obiect rotund de culoare roșie. Metoda de funcționare se bazează pe detecția pixelilor roșii din imaginea capturată de camera sa. Aceștia sunt filtrați în funcție de distanța lor față de valoarea culorii roșu din gama YUV, folosindu-se de o referință calculată ce permite detecția chiar și în condițiile în care luminozitatea se modifică. Apoi din setul de pixeli roșii sunt păstrați doar cei care definesc o formă circulară.

Ca dezavantaj al lui Nao este faptul că detecția de obiecte se limitează doar la cele care sunt roșii și de formă circulară. Alte mingii, de exemplu, care sunt altă culoare nu pot fi depistate. [14]

2.3.2.2 Modulul de recunoaștere de obiecte

ALVisionRecognition oferă posibilitatea de a recunoaște diferite imagini, părți din anumite obiecte sau chiar și locuri învățate anterior.

Modulul funcționează pe principiul recunoașterii unor lucruri care deja au fost învățate, iar dacă cel care îl utilizează își dorește ca Nao să învețe un nou obiect, va trebui să își aloce câteva secunde pentru a înregistra obiectul în baza sa de date din memorie.

Din păcate, modulul dispune de o serie de limitări care nu facilitează un proces optim de recunoaștere a obiectelor, precizia fiind destul de mică. De asemenea, funcționarea se bazează mai mult pe recunoașterea unor puncte cheie și nu pe conturul exterior al obiectelor. Până în momentul de față, nu poate recunoaște clase de obiecte, cum ar fi o cutie, ci ar recunoaște o anumită cutie care a fost înregistrată în memorie. În același timp, Nao nu poate detecta obiectele învățate la mai multe scale. [15]

2.3.3 Choregraphe

Choregraphe este o aplicație software special concepută pentru Nao și care pune la dispoziție oricărui utilizator o interfață intuitivă cu scopul de a-i permite crearea diferitelor animații și comportamente pe care le însușească ulterior robotului și pe care le poate testa atât pe un Nao simulat, cât și pe unul real.

Aplicație dispune și de monitorizare video a robotului, fiecare mișcare a sa fiind ilustrată aproape instantaneu într-o fereastră, putând în același timp să vizualizăm și ce surprinde Nao pe una dintre camerele video integrate.

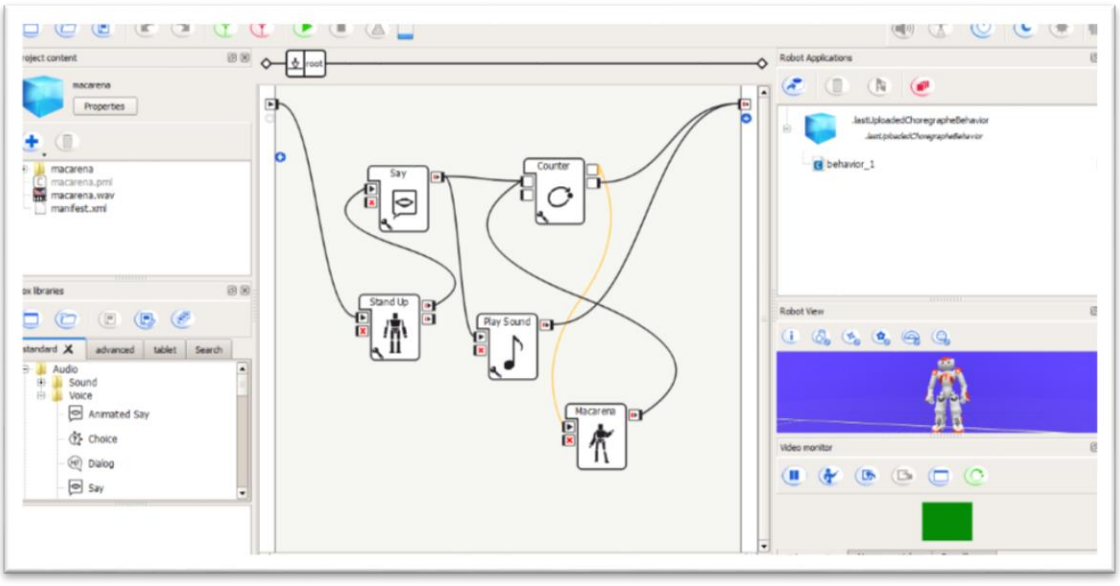


Figura 2.10 *– Exemplu de aplicație în Choregraphe*

Unul dintre avantajele acestei aplicații este ca Nao poate să fie programat și controlat fără a fi nevoie de vreo linie de cod. Tot ce trebuie să facă utilizatorul este să selecteze dintr-o listă predefinită o comandă (*Figura 2.12*) pe care vrea să o atribuie și să o integreze într-o rețea împreună cu alte

comenzi pentru a forma un comportament complet pentru robot. Singura restricție în ceea ce privește programarea acestuia este faptul că atunci când se adaugă o nouă comandă în rețea, trebuie să se țină cont de logica fluxului de mișcări, în sensul că dacă se dorește ca Nao să execute o mișcare, în care la sfârșit el va rămâne într-o poziție nenaturală, iar următoarea nu va începe cu o poziție de echilibru, se poate întâmpla ca acesta să se dezechilibreze și să cadă. Este important deci să se țină cont de logica fluxului de mișcări.

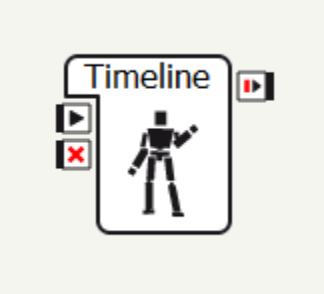


Figura 2.11 *– Exemplu de comandă prestabilită a programului Choregraphe*

Comportamentele create în Choregraphe sunt în spate scrise și procesate în limbajul grafic specific sistemului său. NAOqi are rolul de a interpreta aceste comenzi și de a le executa. Interacțiunea programului cu NAOqi permite utilizatorilor să folosească toate modulele de care dispune NAO.

Pe lângă programarea în Choregraphe, robotului i se pot adăuga fișiere scrise în Python sau C++ care pot să conțină diverse programe și algoritmi care să îl pună în funcțiune.

Avantajul pe care îl are Choregraphe față de utilizarea unui SDK este că dacă vrem să realizăm animații, o putem face mult mai ușor și ar dura mult mai puțin pentru ca avem la dispoziție o cronologie *(Figura 2.13*) în care putem înregistra orice mișcare a robotului cu ușurință.

Figura 2.12 *– Cronologia de înregistrare a mișcărilor în Choregraph*

Dezavantajul ar fi că pe lângă faptul că acele comportamente realizate în Choregraphe au un timp de execuție mai mare, există și multe restricții de implementare a programelor pe care ne dorim să le realizăm deoarece putem alege dintr-un număr limitat de comenzi. Astfel, folosind un SDK ne este oferită mult mai multă flexibilitate și libertate de implementare.

Modul de lucru optim pentru a realiza o aplicație cu un grad mare de complexitate este de a îmbina cele două soluții de programare. Odată realizată o animație, Choregraphe facilitează exportul acesteia sub formă de cod în Python sau C++ care se poate adăuga unui fișier în care este folosit unul dintre cele două SDK-uri.

2.3.4 Biblioteca OpenCV

Din cauza lipsei de resurse pentru prelucrarea datelor vizuale a sistemului lui Nao, a fost necesară instalarea bibliotecii OpenCV versiunea 2.4. Aceasta este disponibilă oricărui utilizator, fără a necesita deținerea unei licențe și se poate descărca online. A fost creată cu scopul prelucrării elementelor vizuale la nivel computațional pentru a eficientiza volumul de lucru și de a pune la dispoziție rezultate corecte.

Ultimele versiuni ale acestei biblioteci conține mai mult de 2500 de algoritmi optimizați care pot fi folosiți în aplicații precum:

- Identificarea unui obiect
- Detectarea și recunoașterea fizionomiilor
- Urmărirea anumitor mișcări sau obiecte
- Combinări de imagini
- Extragerea unor modele 3D ale obiectelor.

2.3.4.1 Căutarea unui șablon

În proiectul de față, principala funcție pusă la dispoziție de biblioteca Open CV pe care am folosit-o este “matchTemplate”, care caută anumite zone dintr-o imagine dată folosindu-se de un șablon.

Ceea ce ne trebuie sunt două imagini:

- Imaginea sursă – cea în care vrem să găsim modelul dorit;

- Imaginea șablon – obiectul pe care vrem să îl depistăm în poza dată.

Scopul funcției este să găsească aria imaginii sursă cu cea mai precisă potrivire. Acest lucru se realizează prin comparația șablonului dat cu fragmente din imaginea sursă. Aceste fragmente sunt de fapt arii din sursa noastră de aceeași dimensiune cu șablonul. Ele se selectează într-un mod glisant, în sensul că dacă primul fragment corespunde indicilor 1, 2, ..., n pe orizontală și n este o dimensiune a modelului, al doilea fragment va corespunde pentru 2, 3, ..., n+1, același lucru fiind valabil și pentru glisarea pe verticală. Cu alte cuvinte șablonul va parcurge imaginea sursă din pixel în pixel pentru a verifica toate posibilitățile. În mod evident, șablonul trebuie să aibă dimensiunea mai mică decât sursa pentru a fi posibilă comparație.

La fiecare mișcare, este calculat cât de bine se potrivește modelul nostru cu fragmentul respectiv, iar rezultatul va fi stocat într-o matrice (R), unde elementele acesteia conțin nivelul de asemănarea calculat pentru fiecare fragment în parte. Selectarea rezultatului cel mai bun se realizează cu ajutorul funcției minMaxLoc care găsește minimul și maximul global într-o matrice.

Funcția matchTemplate pune la dispoziție mai multe metode prin care rezultatul de similaritate este calculat și stocat in matricea R. Acestea sunt:

- CV_TM_SQDIFF – aceasta calculează diferența pătratică, apoi celei mai bune potriviri îi va fi atribuită valoarea 0.

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2 \quad (1)$$

- CV_TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \quad (2)$$

- CV_TM_CCORR – aceasta calculează corelația pentru a determina similitudinea, după care, cele mai bune rezultate vor avea valori de 100%

$$R(x,y) = \sum_{x',y'}(T(x',y') \cdot I(x+x',y+y')) \quad (3)$$

- CV_TM_CCORR_NORMED

$$\mathrm{R}(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \quad (4)$$

- CV_TM_CCOEFF – această metodă determină similitudinea cu ajutorul coeficienților de corelație, după care cele mai bune rezultate vor avea însușite valori de 100%

$$R(x,y) = \sum_{x',y'}(T'(x',y') \cdot I(x+x',y+y')) \quad (5)$$

- CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}} \quad (6)$$

Unde:

- w = lățimea
- h = înălțimea
- I = imaginea sursă
- T = imaginea obiectului țintă
- R = elementele matricei corespunzătoare rezultatelor obținute
- x’ = 0, …, w-1
- y’ = 0, …, h-1

Metodele normate vor normaliza variația luminii dintr-o imagine. Metodele vor genera rezultate cu valori în intervalul (0,1), unde 1 reprezintă cea mai bună potrivire pentru CCORR și COEFF, iar pentru SQDIFF valoarea este 0. [17][18]

2.4 LIMITĂRI

Nao, deși el este un robot destul de performant, este în continuare un sistem cu resurse limitate atât din punct de vedere hardware, cât și software.

Memoria sa de lucru este destul de mică (1 GB RAM și 2 GB Flash) și poate deveni un impediment major atunci când programatorul are nevoie să dezvolte o aplicație mai amplă, iar suprascrierea memoriei nu este întotdeauna o soluție utilă sau eficientă. Se pot adăuga în schimb până la maximum 8 GB prin inserarea unui card de memorie.

Tot ca limitare hardware, este și puterea de procesare mică pe care o are Nao deoarece se poate întâmpla ca atunci când trebuie efectuat un volum mare de calcule, acesta va înceta să își acționeze o parte dintre celelalte resurse existente.

Din cauza lipsei de module special concepute pentru prelucrări de date, este necesară instalarea unor librării suplimentare, însă chiar și așa este necesară instalarea unor versiuni mai vechi pentru ca sistemul lui Nao să fie compatibil cu ele și să le poată folosi. De exemplu, pentru prelucrarea imaginilor se poate instala OpenCV versiunea 2.4 care conține un număr mult mai mic de funcții față de versiunile mai avansate. Acest lucru duce la necesitatea reimplementării unor funcții care ar putea simplifica modul de lucru și la economisirea timpului.

Aceeași problemă este și pentru SDK-ul limbajului de programare Python. Versiunea compatibilă cu sistemul actual este 2.7, ceea ce poate cauza probleme elementare, chiar și la nivel de rotunjiri și aproximări și care într-un algoritm mai substanțial poate cauza erori majore.

Deși în mod normal modulele prezente în NAOqi ar trebui să poată funcționa pe principiul paralelismului, se poate întâmpla ca unele dintre ele să nu se poată sincroniza și să fie nevoie să se recurgă la alte metode de implementare sau chiar alte idei decât cele inițiale.

CAPITOLUL 3

Modele de mixturi Gaussiene

3.1 INTRODUCERE

Un model de mixtură este un model al densității care cuprinde un anumit număr de câteva componente, în general gaussiene. Acestea sunt folosite cu scopul de a crea o densitate multimodală. Pentru a înțelege mai bine acest concept, putem face referire la proiectul de față, în care ne dorim selectarea unei culori dintr-o imagine. Este nevoie deci de o prelucrare complexă a imaginii din care se vor extrage culorile dominante, fiind luate în considerare și variațiile acestora.

Astfel de modele pot crea constrângeri puternice în vederea asignării unui obiect la o clasă în funcție de culoare. De exemplu, se pot realiza aplicații ce netezesc spații provenite din mostrele de date de la sursă. O precizie bună este necesară întocmai pentru a obține cele mai reușite rezultate posibile dintr-o clasificare a pixelilor bazată pe culoare, cu scopul de a realiza o segmentare calitativă. Odată ce un model este generat, pot fi create probabilități condiționate pentru pixelii color ai imaginii.

Un model de mixtură gaussiană este practic o funcție parametrică a densității de probabilitate, reprezentată ca suma ponderată a componentelor gaussiene de densitate. GMM-urile sunt de obicei folosite ca modele de distribuție probabilistică a unor măsurători sau trăsături într-un sistem. Ele pot fi de asemenea interpretate ca o formă generalizată a unor funcții cu bază radială folosite într-o rețea neuronală. Parametrii pentru GMM sunt estimați prin antrenarea unui set de date fiind folosit fie algoritmul iterativ Exprectation-Maximization, fie estimarea Maximum A-Posteriori (MAP) care pornește de la un model bine antrenat.

Modelele de mixturi gaussiene sunt des întâlnite în aplicațiile ce au ca scop recunoașterea anumitor șabloane, machine learning sau analiză statistică. GMM-urile sunt metode parametrice, ele depinzând de un set de parametrii definit, ceea ce le transformă într-o alternativă destul de bună față de analiza pe baza unei histograme, care nu este parametrică. În multe aplicații, parametrii acestora sunt determinați cu ajutorul funcției de cost, folosind de obicei algoritmul de Expectation-Mazimization (EM).

3.2 ALGORITMUL DE CLASIFICARE K-MEANS

Vom începe prin a considera problema identificării unor grupuri de puncte dintr-un spațiu multidimensional. Presupunem că avem un set de date ce conține un număr N de elemente dintr-un spațiu euclidian D-dimensional. Scopul este de a separa datele într-un număr de K clase, unde K poate aparține mulțimii {3,4,5} și de cele mai multe ori este dat. Putem vedea un grup ca pe o mulțime de puncte ale căror distanțe unele față de altele sunt mult mai mici decât în cazul distanțelor față de punctele aflate în afara formațiunii respective. Putem introduce astfel un set de vectori D-dimensionali pe care îi vom nota cu μ_k, unde k = 1, . . ., K; μ_k va reprezenta un parametru asociat celui de-al k-lea grup de date. În proiectul propus, vom folosi această notație pentru a reprezenta centrii fiecărei formațiunii de pixeli grupați. Astfel, scopul nostru este de a reuși să grupăm pixelii unei imagini și de a determina setul de vectori $\{\mu_k\}$ ce va conține media fiecărui grup în parte.

Vom defini câteva notații pentru asocierea datelor cu grupurile de clasificare. Astfel, pentru fiecare punct x_n vom stabili un set de variabile binare $r_{nk} \in \{0,1\}$, unde k = 1, . . ., K pentru a ne spune cărui grup aparține punctul respectiv. Deci, dacă punctul x_n aparține celui de-al k-lea grup, atunci r_{nk} = 1 și r_{nj} = 0, j fiind diferit de k. Funcția pe care vrem să o definim poartă numele de funcție cost și este de forma:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 \quad (7)$$

și reprezintă suma pătratelor distanțelor fiecărui punct față de vectorul μ_k. Scopul nostru este să găsim valorile pentru $\{r_{nk}\}$ și $\{\mu_k\}$ astfel încât funcția J să fie minimă. Acest lucru se face cu ajutorul unui proces iterativ în care fiecare iterație implică doi pași corespunzători unei optimizări succesive în concordanță cu r_{nk} și μ_k. Mai întâi vom inițializa valorile μ_k, apoi, în primă fază, vom minimiza funcția cost în funcție de valorile r_{nk}, menținând μ_k fix, după care vom face același lucru însă de această dată vom minimiza în funcție de valoarea lui μ_k, cu valoarea lui r_{nk} fixă. Cele două etape se vor repeta până când vom obține o convergență. În cele ce urmează, vom vedea că modificările valorilor pentru r_{nk} și μ_k corespund celor două etape ale algoritmului EM, și anume E(expectaion) și M(maximization).

Algoritmul de atribuire a punctelor în două etape celor K grupuri și recalcularea centrilor sunt repetate până când niciun punct nu mai este asignat altei clase decât celei din care face parte sau până când numărul maxim de iterații a fost depășit. Convergența algoritmului este asigurată datorită faptului ca la fiecare iterație este minimizată funcția J din ce în ce mai mult.

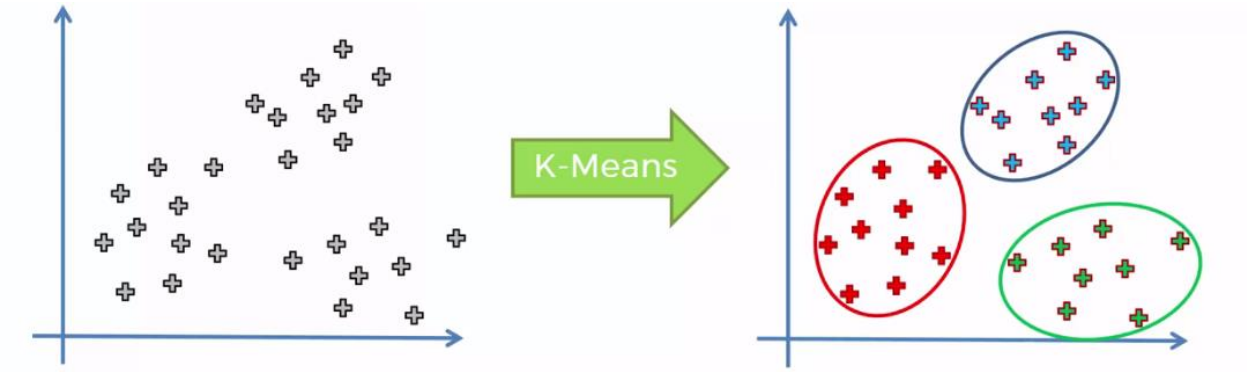


Figura 3.1 *– Model de grupare a datelor prin algoritmul K-means [21]*

Vom lua în considerare faptul că inițial sunt alese niște valori aleatorii pentru fiecare medie a grupurilor pentru ca algoritmul să poată trece prin câteva iterații înainte de convergență. De menționat faptul că metoda K-means este des utilizată în inițializarea parametrilor unui model de mixtură gaussiană, așa cum vom vedea mai încolo.

Din nefericire, implementarea acestui algoritm poate duce la o procesare destul de greoaie deoarece in fiecare etapă de E(expectaion) este necesară calcularea distanței euclidiene dintre toate valorile centrilor și punctele pe care vrem să le separăm.

3.3 SEGMENTAREA UNEI IMAGINI

Pentru a înțelege modul de aplicare al algoritmului K-means putem considera problema pe care vrem să o rezolvăm în proiectul de față, și anume segmentarea unei imagini pentru a extrage un anumit obiect. Scopul segmentării este de a separa imaginea în regiuni, fiecare având o anumită caracteristică, mai exact o medie generală. Fiecare pixel al imaginii reprezintă un punct care face parte dintr-un spațiu tridimensional ce ilustrează intensitatea canalelor de roșu, verde și albastru. Segmentarea va trata deci fiecare pixel în parte ca pe o dată de intrare separată.

Putem aplica deci algoritmul de K-means fără dificultăți și putem de asemenea să demonstrăm convergența algoritmului pentru orice număr de grupuri K, redesenând imaginea dată prin înlocuirea fiecărui vector de pixel RGB cu intensitatea tripletului dată de centrul μ_k căruia i-a fost asignat pixelul pe care dorim să îl prelucrăm. Este de menționat faptul că pentru o valoare dată a lui K, algoritmul va putea reprezenta imaginea folosind doar K culori. Această metodă nu este neapărat cea mai performantă abordare pentru a segmenta imaginea în special din cauză că nu ține cont de proximitatea spațială a pixelilor.

3.4 MIXTURI GAUSSIENE

Pentru o perspectivă mai detaliată în ceea ce privește distribuțiile, putem vorbi de o formulare a mixturilor gaussiene în termeni de variabile latente discrete. În acest fel, putem aduce în discuție algoritmul EM.

Distribuția mixturii poate fi scrisă sub forma:

$$p(x) = \sum_{k=1}^{K} \pi_k \, \mathrm{N}\left(\mathrm{x}\middle|\mu_{k,}, \Sigma_k\right). \tag{8}$$

Vom introduce o variabilă K-dimensională z binară și aleatoare în a cărei reprezentație există un anumit element z_k egal cu 1, iar restul fiind nuli. Valorile pe care le ia z_k pot fi 0 sau 1 și $\sum_k z_k = 1$, observând că din K posibilități ale vectorului z, doar un singur element poate lua valoarea 1 pentru a reține cărei distribuții îi aparține elementul.

Distribuția p(x, z) o vom împărți în distribuție marginală p(z) și distribuție condiționată p(x|z), unde p(x|z) va fi scrisă sub forma: $p(z_k = 1) = \pi_k$, unde $0 \leq \pi_k \leq 1$ și $\sum_{k=1}^{K} \pi_k = 1$ pentru ca probabilitățile să fie valide.

Similar, distribuția condiționată a unei valori z atribuite lui x este o funcție gaussiană:

$$p(x|z_k = 1) = N(x|\mu_k, \Sigma_k) \tag{9}$$

Distribuția p(x, z) este dată de relația p(z)p(x|z), iar distribuția marginală a lui x este obținută apoi prin însumarea produselor dintre p(x, z) cu toate stările posibile ale lui z, urmând să obținem relația:

$$\text{p(x)} = \sum_z p(z)\ \text{p(x|z)} = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k) \tag{10}$$

Dacă avem un număr de N valori ale lui x, atunci, vrând să reprezentăm distribuția marginală în $\text{p(x)} = \sum_z p(x, z)$, vom obține pentru fiecare dată x_n o variabilă latentă z_n corespunzătoare.

3.5 FUNCȚIA DE COST MAXIMĂ

Vom porni de la presupunerea ca avem un set de date $\{x_1, \dots, x_N\}$ și vrem să le modelăm folosind mixturi gaussiene. Putem reprezenta acest set de date sub forma unei matrice X de dimensiune N × D în care cel de-al n-lea rând este dat de x_n^T. În mod similar, variabilele latente corespunzătoare vor putea fi scrise ca pe o matrice Z unde rândul n este dat de z_n^T. Funcția cost este reprezentată cu ajutorul următoarei formule:

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k) \right\}. \tag{11}$$

Trebuie adus în discuție faptul că poate apărea o problemă semnificativă asociată cu aplicarea funcției cost asupra unui model de mixturi gaussiene din cauza existenței singularităților, lucru care nu are loc atunci când există o singură distribuție. Vom considera că avem o mixtură ale cărei elemente au matrici de covarianță date de $\Sigma_k = \sigma_k^2 I$, unde I este matricea unitate. Presupunând că elementul j al modelului de mixtură are media η_j egală cu unul dintre componentele de intrare x_n, atunci x_n va contribui la funcția de cost sub următoarea formă:

$$N(x_n|x_n, \sigma_k^2 I) = \frac{1}{(2\pi)^{1/2}} \frac{1}{\sigma_j}. \tag{12}$$

Dacă am considera că limita apare când $\sigma_j \to 0$, atunci termenul va tinde către infinit și la fel și funcția de cost logaritmică. În plus, maximizarea funcției de cost va fi întotdeauna problematică din cauză că singularitățile vor exista în permanență. Amintim că această problemă nu apare în cazul unei singure distribuții gaussiene. Pentru a înțelege diferența, dacă o singură gaussiană va avea erori pentru o singură dată de intrare, atunci la funcția de cost vor contribui mai mulți factori care provin de la celelalte date și care vor tinde exponențial către zero, lucru care determină costul să conveargă către zero, și nu infinit. Dacă în schimb am avea cel puțin două componente în mixtură, atunci una dintre componente poate avea o varianță finită, prin urmare, poate asigna o probabilitate finită datelor de intrare. În același timp, restul componentelor se axează pe câte o singură dată, ducând astfel la o creștere a funcției de cost. Acest lucru este ilustrat în *Figura 3.2*.

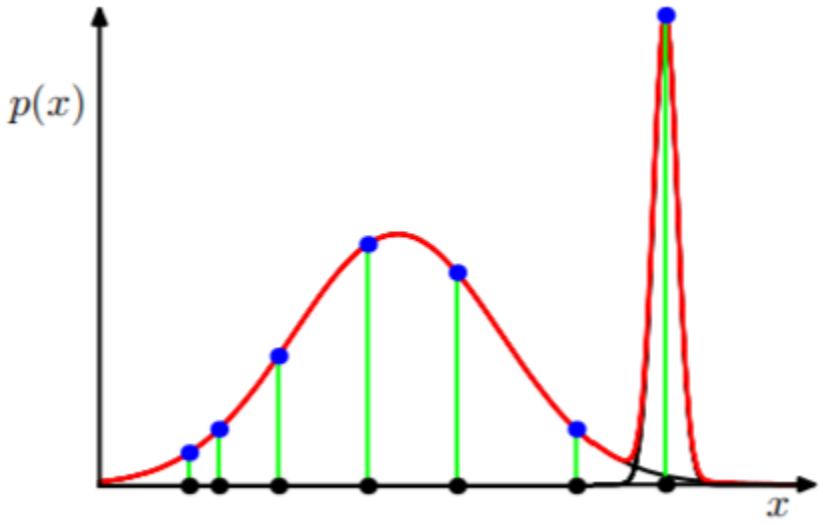


Figura 3.2 *– Ilustrarea grafică a modulului în care singularitățile afectează mixturile gaussiene [18]*

Folosind funcția maximă de cost în determinarea mixturilor gaussiene va trebui să evităm găsirea unor astfel de soluții patologice și să încercăm sa găsim un maxim local. Putem încerca să detectăm când obținem o eroare în calculul unei componente gaussiane și să resetăm media la o valoarea aleasă aleator, iar covariația să o înlocuim cu o valoare mare, continuând astfel să optimizăm procesul.

3.6 ALGORITMUL EXPECTATION MAXIMIZATION ÎN MIXTURILE GAUSSIENE

O soluție puternică pentru problema găsirii funcției maxime de cost este algoritmul Expectation Maximization (EM) deoarece acesta acoperă o gamă variată de aplicații și de asemenea este des întâlnit în diverse contexte.

Vom începe prin a scrie condițiile care trebuie îndeplinite pentru a satisface funcția maximă de cost. Egalând cu zero derivata lui $\ln(p(X|\pi,\mu,\Sigma)) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k N(x_n|\mu_k,\Sigma_k)\}$ fără a modifica valoarea mediei μ_k, vom obține:

$$0 = -\sum_{n=1}^{N} \frac{\pi_k N(x_n|\mu_k,\Sigma_k)}{\sum_j \pi_j N(x_n|\mu_j,\Sigma_j)} \Sigma_k (x_n - \mu_k) \tag{13}$$

unde $\frac{\pi_k N(x_n|\mu_k,\Sigma_k)}{\sum_j \pi_j N(x_n|\mu_j,\Sigma_j)}$ este de fapt $\gamma(\mu_{nk}) \equiv \mathrm{p}(z_k = 1|x)$

Multiplicând cu Σ_k^{-1}, pe care o presupunem a nu fi singulară, vom obține:

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n \tag{14}$$

unde $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$.

N_k = poate fi interpretat ca numărul efectiv de puncte asignate unui grup de clasificare k. Observăm că media celui de-al k-lea element gaussian μ_k este obținut prin media ponderată a tuturor punctelor din setul de date în care ponderile datelor de intrare sunt calculate prin intermediul probabilității posterioare $\gamma(z_{nk})$ în care componenta k generează x_n.

Dacă vom egala derivata lui $\ln(p(X|\pi,\mu,\Sigma)$ cu zero fără a modifica dispersia Σ_k și în mod analog față de funcția maximă de cost calculăm dispersia unei singure componente gaussiene, vom obține forma:

$$\Sigma_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})(x_n-\mu_k)(x_n-\mu_k)^T \tag{15}$$

care are aceeași formă ca și rezultatul pentru o singură gaussiană din setul de date, dar de asemenea fiecare dată este ponderată de probabilitatea posterioară și numitorul este numărul efectiv de puncte aferente fiecărei componente.

În final vom maximiza $\ln(p(X|\pi,\mu,\Sigma)$ păstrând valorile ponderilor mixturii π_k. Va trebui să ținem cont că $\sum_{k=1}^{K}\pi_k = 1$. Acest lucru poate fi obținut dacă folosim un coeficient Lagrange:

$$\ln(p(X|\pi,\mu,\Sigma) + \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right) \tag{16}$$

din care va rezulta

$$0 = \sum_{n=1}^{N}\frac{N(x_n|\mu_k,\Sigma_k)}{\sum_j \pi_j N\left(x_n|\mu_j,\Sigma_j\right)} + \lambda \tag{17}$$

Dacă vom multiplica ambele părți cu π_k și vom face sumă după k, vom găsi $\lambda = -N$. Eliminând termenul λ și reordonând ecuația vom avea $\pi_k = \frac{N_k}{N}$ (18), astfel că ponderea pentru componenta k este dată chiar de media ponderilor calculată pentru acea componentă.

Aceste rezultate vor duce la o metodă iterativă mai simplă pentru găsirea soluției la problema funcției maxime de cost, pe care o putem vedea ca pe o instanța a algoritmului EM în care este particularizat cazul modulului mixturilor gaussiene .

În primul rând, media, distribuția și ponderile mixturii se vor inițializa cu niște valori pe care le obținem prin aplicarea algoritmului K-means. Vom discuta în mod alternativ mai departe despre cei doi pași ai metodei EM, pe care ii vom numi pasul E (mediere a funcției de cost) și pasul M (maximizare). La pasul de mediere vom folosi valorile curente ale parametrilor pentru a evalua probabilitățile posterioare. Vom folosi apoi aceste probabilități în etapa de maximizare pentru a estima din nou media, dispersia și ponderile folosind rezultatele obținute la (14), (15) și (18). Mai întâi vom evalua noile medii folosind (14), apoi cu aceste valori vom calcula dispersiile folosind (15), scopul lor fiind determinarea unei sigure gaussiene.

Fiecare actualizare a parametrilor rezultați în urma celor două etape, de mediere și de maximizare va asigura creșterea funcției cost logaritmice. De obicei, în practică, acest algoritm se presupune că va converge atunci când funcția de cost logaritmică va depăși o anumită limită.

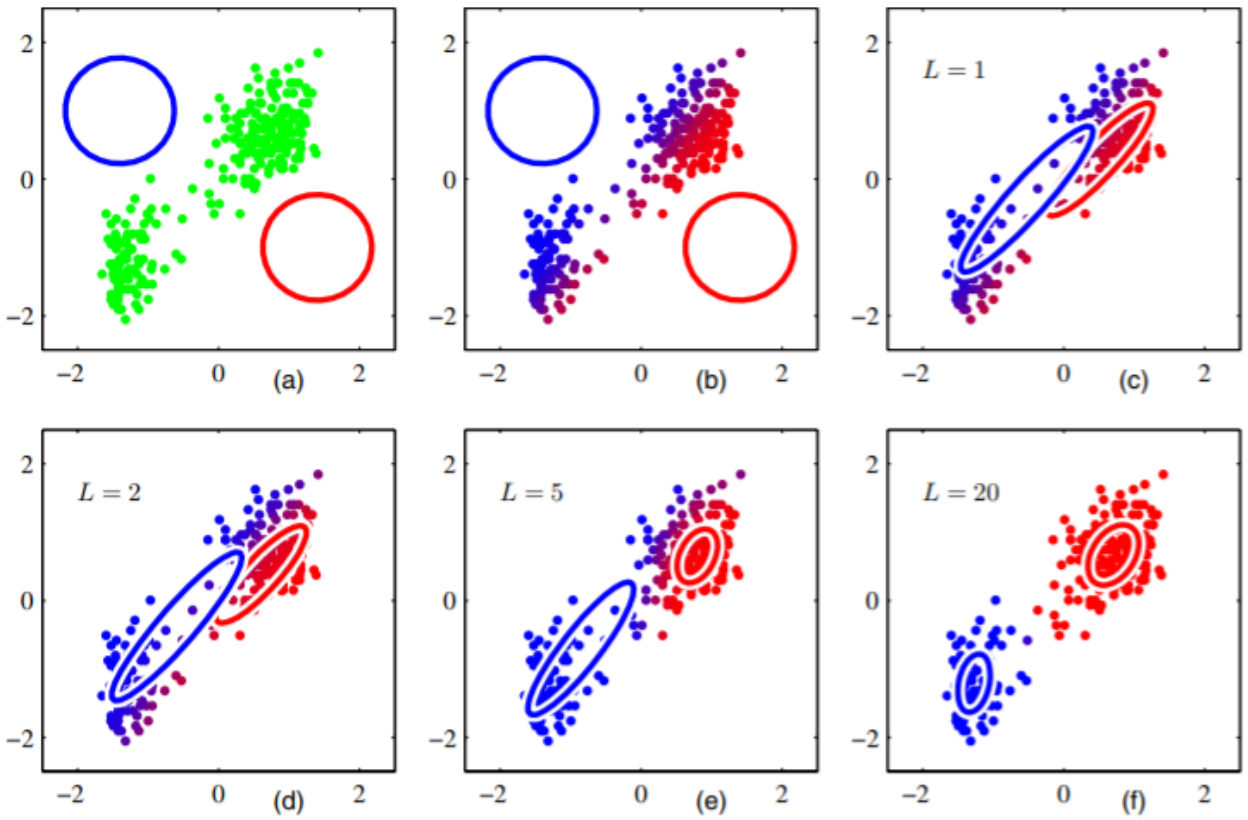


Figura 3.3 – *Ilustrarea paşilor de separare a datelor folosind algoritmul GMM [18]*

În *Figura 3.3,* este ilustrat algoritmul GMM aplicat pentru 2 distribuții. În figura (a) datele de intrare pe care se va lucra sunt reprezentate cu verde împreună cu cele două configurații inițiale ale componentelor gaussiene reprezentate cu albastru, respectiv roșu. Figura (b) conține rezultatul primului pas E (de mediere a funcției de cost), în care fiecare punct este reprezentat cu ajutorul unei proporții de albastru egală cu probabilitatea posterioară prin care a fost generată componenta albastră, similar obținându-se și pentru cea roșie. Astfel, punctele care au o probabilitate semnificativă de apartenență la ambele clase sunt reprezentate cu mov. În imaginea (c) este reprezentat momentul de după pasul M (de maximizare) în care media componentei gaussiene albastre și-a modificat valoarea astfel încât este egală cu media setului de date, ponderată de probabilitățile fiecărui punct care aparține grupului de culoare albastră. Se procedează similar pentru valoarea covariației gaussienei albastre pentru a o egala cu cea a setului de date albastre. Analog vom determina și rezultatele pentru componenta roșie. Restul imaginilor ilustrează ce s-a obținut după un număr L de iterații complete ale algoritmului EM.

Pe scurt, atunci când avem un model de mixturi gaussiene ne dorim să maximizăm funcția de cost parcurgând următorul algoritm:

1. Aplicarea algoritmului K-means pentru generarea valorilor de inițializare a mediei μ_k, a dispersiei Σ_k și ponderile π_k, evaluând ulterior valoarea inițială a funcției de cost logaritmice.
2. La pasul de mediere vom calcula funcția de cost cu ajutorul valorilor
$$\gamma(\mu_{nk}) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j N(x_n|\mu_j, \Sigma_j)} \quad (19)$$
3. Pasul de maximizare presupune reestimarea parametrilor

$$\mu_k^{nou} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})x_n \tag{20}$$

$$\Sigma_k{}^{nou} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})(x_n - \mu_k^{nou})(x_n - \mu_k^{nou})^T \tag{21}$$

$$\pi_k{}^{nou} = \frac{N_k}{N} \tag{22}$$

4. Evaluarea funcției de cost logaritmice

$$\ln p(X|\pi,\mu,\Sigma) = \sum_{n=1}^{N}\ln\{\sum_{k=1}^{K}\pi_k N(x_n|\mu_k,\Sigma_k)\} \tag{23}$$

și verificarea convergenței fiecărui parametru. În cazul în care condiția de convergență nu a fost îndeplinită, atunci algoritmul se va relua de la pasul 2.

3.7 CONCLUZII

Algoritmul GMM este unul dintre cele mai puternice unelte pentru aplicații privind imagini, sunete și statistici, făcând posibile clasificări ale datelor de prelucrare cu scopul căutării unor anumite trăsături. Extragerea obiectului principal dintr-o imagine reprezintă una dintre multele aplicații care pot fi rezolvate cu ajutorul algoritmului de GMM, în special dacă sunt folosite modele simple în imaginea de lucru. GMM-ul ajută, așa cum am explicat mai sus, la o clasificare a pixelilor în funcție de numărul de culori pe care îl stabilim prin K-means, iar mai departe, culorile sunt separate printr-un proces iterativ cu metoda EM.

CAPITOLUL 4

Dezvoltarea unui sistem de captură, detecție și urmărire a unui obiect

4.1 PRINCIPIUL DE FUNCȚIONARE

În proiectul de față, mi-am propus dezvoltarea unui sistem integrat pe Nao care să realizeze o captură a unui obiect, iar apoi robotul să îl urmărească în timp ce acesta este mișcat. Programul a fost scris în limbajul Python versiunea 2.7 cu ajutorul aplicației Spyder. Am ales limbajul Python în defavoarea lui C++ deoarece oferă mult mai multă flexibilitate de lucru.

În algoritmul propus se urmărește comanda robotului cu ajutorul unor senzori tactili, cei trei amplasați pe cap și cel de pe mâna dreaptă. Primul pas după lansarea aplicației care ar trebui făcut este să se realizeze o captură a unui obiect pe care vrem ca robotul să îl poată urmări. Din imaginea capturată, obiectul va fi extras și salvat în memoria lui Nao pentru a fi folosit ca model în detecția ce urmează a fi făcută.

Dacă vrem să pornim procesul de urmărire, este nevoie de acționarea butonului frontal de pe cap, ce va încărca obiectul țintă sub forma unui șablon. În continuare, au loc într-o structură repetitivă capturi fotografice cu ajutorul uneia dintre camerele video integrate, iar în fiecare imagine se încearcă depistarea țintei realizată la mai multe scale. Dacă s-a depistat obiectul în imagine, atunci se vor calcula coordonatele necesare pentru ca Nao să își miște capul astfel încât obiectul să se centralizeze în câmpul său vizual. Robotul se va apropia de obiect când va fi necesară o vizualizare mai de aproape a acestuia.

Dacă între timp vrem să ieșim din programul curent, se apasă cel de-al treilea buton tactil de pe cap. Astfel, opțiunile pe care le are utilizatorul sunt de a înregistra un nou obiect, de a opri aplicația folosind senzorul tactil de pe mâna dreaptă sau de a rula din nou programul de detecție și urmărire.

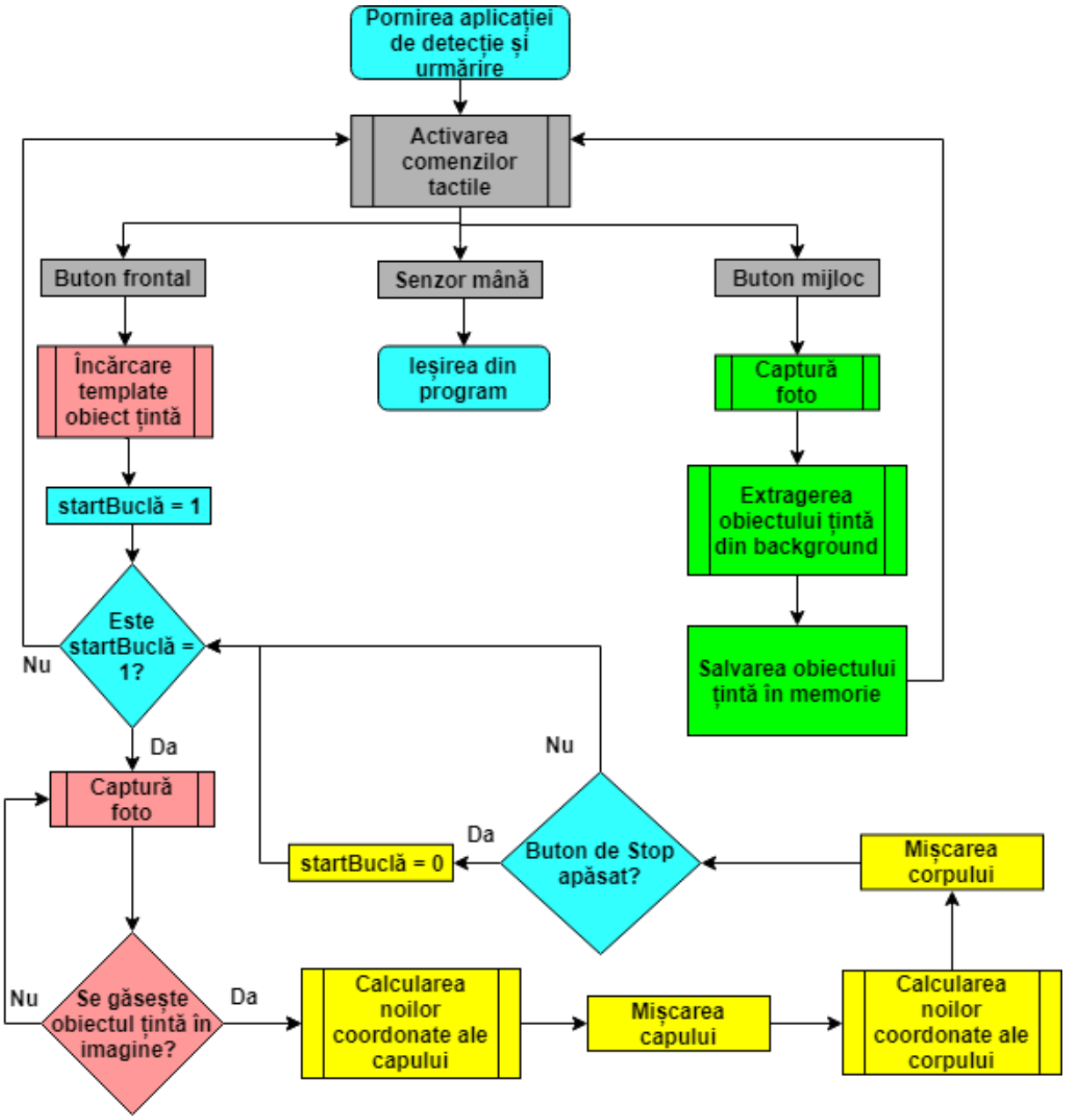


Figura 4.1 (a) *– Schema bloc a algoritmului dezvoltat*

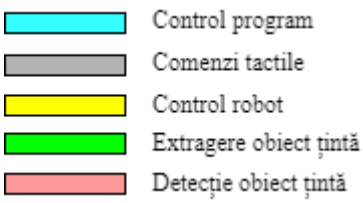


Figura 4.1 (b) *– Legenda schemei bloc*

4.2 DEZVOLTAREA ALGORITMULUI

4.2.1 Comenzile tactile

Senzorii plasați la nivelul capului

Nao are integrați sub formă de butoane trei senzorii tactili la nivelul capului. În proiectul de față, acestora le-au fost însușite comenzi de control cu scopul de a selecta programul dorit.

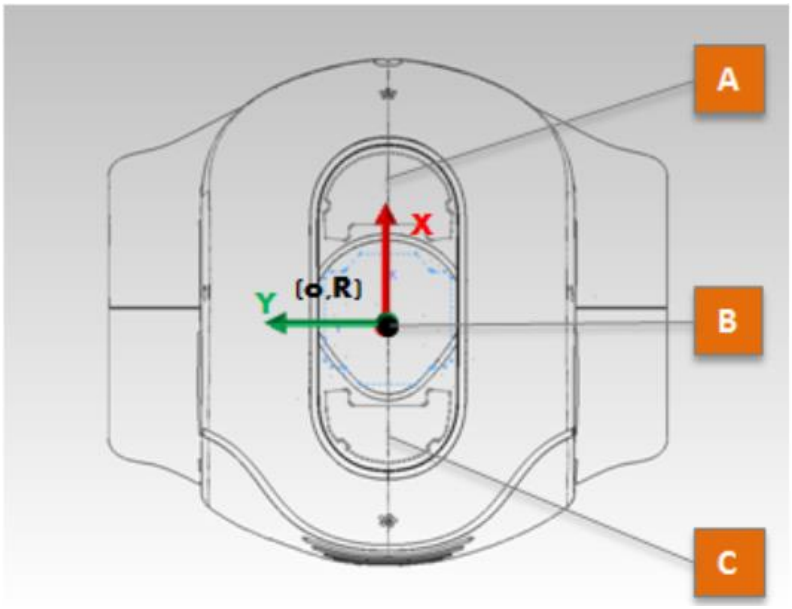


Figura 4.2 *– Amplasarea senzorilor de pe cap [22]*

Butonul A - este butonul frontal și este corespunzător declanșării algoritmului de urmărire a obiectului. După ce a fost apăsat, se va dezactiva și va rămâne în această stare până când se va părăsi bulca ce conține algoritmul.

Butonul B – este butonul mijlociu și odată apăsat se va realiza o captură a imaginii pe care o vede Nao, după care se va extrage obiectul principal și va fi salvat în memorie. Acesta este activ doar atunci când robotul este în așteptarea unei comenzi.

Butonul C – este butonul anterior și acesta se va activa doar după pornirea algoritmului de urmărire, iar scopul său este să iasă din bucla ce conține algoritmul. Odată apăsat, acesta va deveni din nou inactiv.

Senzorul plasat pe mâna dreaptă

Scopul acestui buton tactil este de a permite părăsirea programului atunci când se dorește. El este activ doar atunci când robotul este în așteptarea unei comenzi tactile.

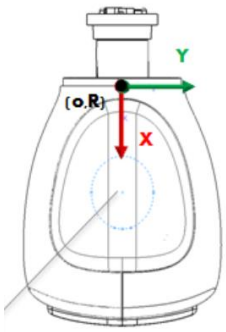


Figura 4.3 – Amplasarea senzorului tactil pe mâna dreaptă [22]

4.2.2 Extragerea obiectului țintă

În ceea ce privește metoda propusă, obiectul pe care ne dorim să îl urmărim nu face parte dintr-o clasă învățată cu ajutorul unei rețele neuronale, ci se va realiza o captură a sa cu ajutorul camerei video a lui Nao din care vor fi extrași doar pixelii ce aparțin țintei noastre. Pentru a face posibil acest lucru a fost nevoie de implementarea unui algoritm GMM de a clasifica pixelii după culoare, după care a necesitat extragerea celor potriviți după o referință stabilită.

4.2.2.1 Implementarea algoritmului GMM

Scopul algoritmului GMM în proiectul de față este de a împărți pixelii unei imagini surprinse de Nao în trei distribuții gaussiene, calculând pentru fiecare în parte media și dispersia. Cu alte cuvinte, vom încerca să realizăm astfel de distribuții orientându-ne după forma histogramei, mai exact a cocoașelor prezente în aceasta.

Declanșarea procesului de extracție al țintei este făcută prin acționarea butonului mijlociu situat la nivelul capului. Mai departe, pentru implementarea sa am realizat o funcție în care se pornește de la aplicarea metodei K-means prin care se vor calcula valorile inițiale ale mediei, covariației și a ponderii mixturii gaussiene. Am ales pentru K valoarea 3 deoarece trei distribuții sunt suficiente pentru a prelucra o imagine care nu conține multe detalii. În urma calculelor se obțin trei centroizi ale căror valori nu garantează acuratețea rezultatelor pe care ni le dorim.

În continuare, într-un mod iterativ, vom îmbunătăți valorile inițiale ale parametrilor GMM prin algoritmul de Expectation Maximization, iar în funcție de rezultatele obținute se calculează costul prin funcția maximă logaritmică. Iterațiile continuă până când s-au efectuat de 10 ori sau până când noul cost este mai mare decât cel anterior, scopul nostru fiind să maximizăm rezultatul funcției cost.

4.2.2.2 Extragerea pixelilor aferenți obiectului țintă

Imaginile pe care le capturează camera video a lui Nao sunt în format RGB, ceea ce înseamnă că un singur pixel este reprezentat de un set de trei valori ce corespund fiecărui canal în parte. Pentru a putea fi prelucrată cu ușurință o astfel de imagine, este necesar să separăm canalele de roșu, verde și albastru. Fiecare canal este transpus într-o matrice corespunzătoare pentru a putea fi prelucrate într-un mod accesibil. În *Figurile 4.4 (a), (b), (c)* și *(d)* de mai jos avem un model de separare a celor trei canale după un model folosit în dezvoltarea proiectului.

Figura 4.4 (a) *– Imaginea originală*

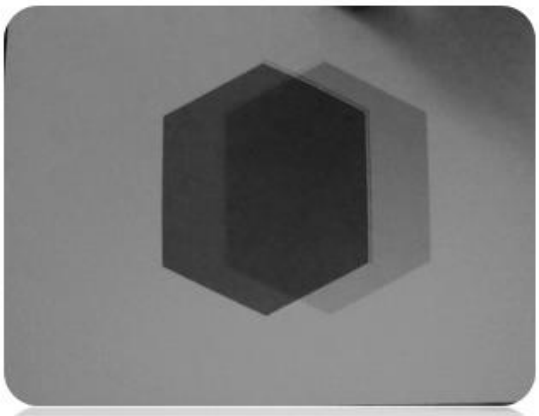


Figura 4.4 (b) *– Extragerea canalului de roșu*

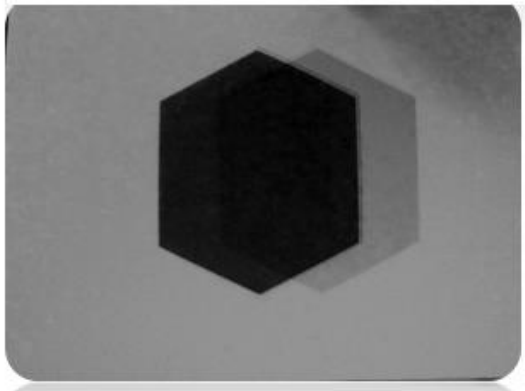


Figura 4.4 (c) *– Extragerea canalului de verde*

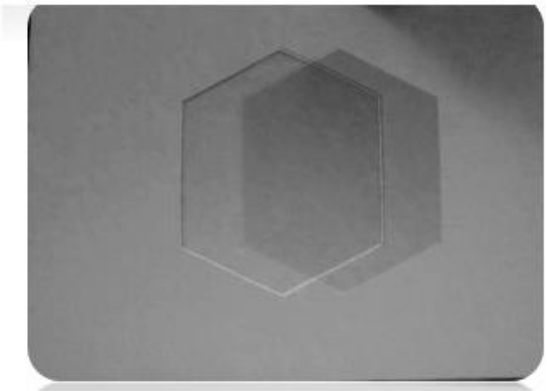


Figura 4.4 (b) *– Extragerea canalului de albastru*

În continuare, pentru fiecare canal se va aplica algoritmul de GMM pentru extragerea parametrilor: mediile și dispersiile. Obiectul țintă corespunde unei singure gaussiene din fiecare canal. Astfel că va trebui să o selectăm pe cea potrivită. Ca să putem face acest lucru este nevoie să alegem niște valori de referință care să fie orientative în privința mediilor, așa că impunem ca la realizarea capturii obiectului de interes, acesta să se afle în centrul câmpului vizual al robotului. Odată capturată imaginea, se va selecta o matrice pătratică din centrul ei și se va calcula media pixelilor corespunzători pe fiecare canal în mod separat.

Pentru un bun repertoriu, în *Figurile 4.5 (a), (b)* și *(c)* avem reprezentate histogramele celor trei canale în care putem observa pentru fiecare în parte câte trei formațiuni ce reprezintă contururile superioare ale unor gaussiene. Distribuțiile pe care le putem vedea sunt calculate cu funcția de GMM care va genera mediile dispersiile necesare.

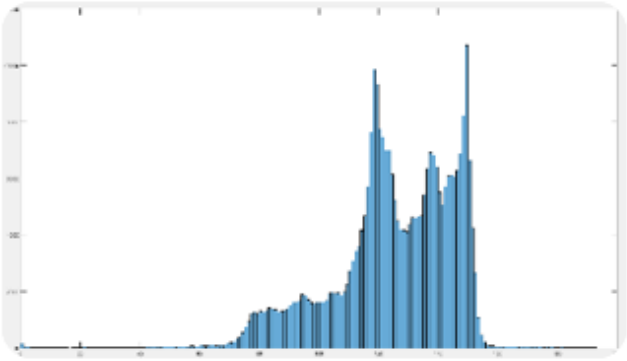


***Figura 4.5 (b)** – Histograma aferentă canalului de albastru*

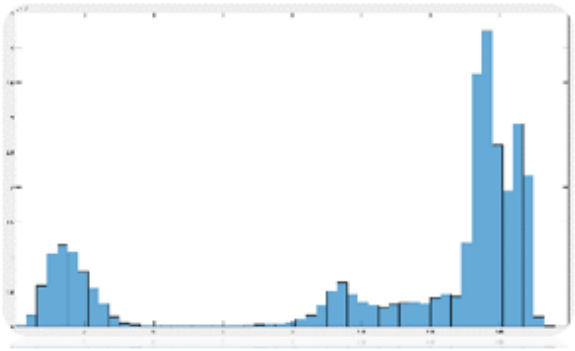


***Figura 4.5 (a)** – Histograma aferentă canalului de roșu*

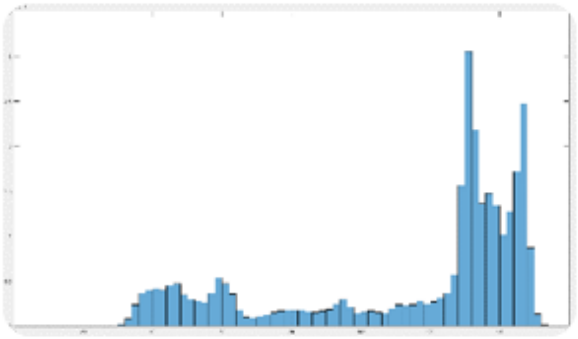


***Figura 4.5 (c)** – Histograma aferentă canalului de verde*

Distribuția gaussiană a unui canal se alege în urma determinării celei mai mici diferențe dintre mediile calculate prin GMM și referința orientativă. Se determină astfel valorile precise de care avem nevoie, urmând să se realizeze extragerea pixelilor din imaginea sursă.

Pixelii sunt aleși pe baza mediei și a dispersiei selectate, astfel ca valorile lor trebuie să se încadreze în jurul mediei la care se adaugă sau se scade dispersia corespunzătoare. Cei corespunzători își vor păstra valorile inițiale, iar restul vor fi modificați la valoarea 0 pentru toate canalele cu scopul de a obține culoarea negru în jurul obiectului. În acest mod, valorile pixelilor necorespunzători nu vor afecta calculele realizate în următorii pași. În decursul selecției, se vor reține și pozițiile limită ale obiectului (stânga, dreapta, sus, jos) pentru a se putea decupa obiectul de interes din imaginea inițială.

După extragerea țintei, aceasta se va salva în memoria lui Nao sub forma unei imagini și va fi folosită ca șablon în algoritmul de detecție și urmărire.

4.2.3 Detecția obiectului în imaginile capturate

Dacă acționăm butonul frontal de la nivelul capului, acesta se va dezactiva, cel anterior va deveni funcțional și în același timp se va declanșa și urmărirea țintei. Pentru ca algoritmul să se poată desfășura, este nevoie să existe un model după care Nao să se orienteze. Acest lucru se realizează prin încărcarea din memoria sa a obiectului extras la punctul anterior. În cazul în care acesta nu are vreun model salvat în memorie, robotul nu va începe procesul de urmărire, anunțând utilizatorul printr-un mesaj vocal că va trebui întâi să încarce în memorie un obiect înainte de execuție.

Dacă totuși există un model, atunci acesta va fi convertit din RGB într-o imagine cu nuanțe de gri și reținut într-o variabilă de tip matrice. Conversia la gri se realizează pentru a facilita ușurința în calcul, în sensul ca în loc să se lucreze pentru un pixel cu un set de trei date, se va lucra cu o singură valoare aferenta pixelului respectiv.

În continuare, în cadrul unei bucle se va realiza scopul propriu-zis până când va fi apăsat butonul anterior de pe cap. La începutul fiecărei iterații, se va face o captură fotografică pentru a analiza ceea ce vede Nao și de asemenea va fi stocată într-o matrice după conversia din RGB în nuanțe de gri.

Cu ajutorul funcției matchTemplate corespunzătoare bibliotecii OpenCV, se va căuta șablonul încărcat din memorie în poza surprinsă, aceasta fiind parcursă pas cu pas. Căutarea are loc pentru trei dimensiuni ale modelului obiectului, ca în *Figura 4.6*, iar scala optimă va fi corespondenta valorii maxime dintre cele generate de funcția matchTemplate.

Dacă obiectul a fost găsit în imagine, Nao își va aprinde înainte în jurul ochilor LED-urile verzi și le va menține așa până când obiectul nu va mai fi detectat, caz în care LED-urile vor fi stinse. De asemenea, dacă ținta a fost găsită va începe procesul de calculare a coordonatelor pentru mișcarea corpului, iar în caz contrar se va sări peste acești pași.

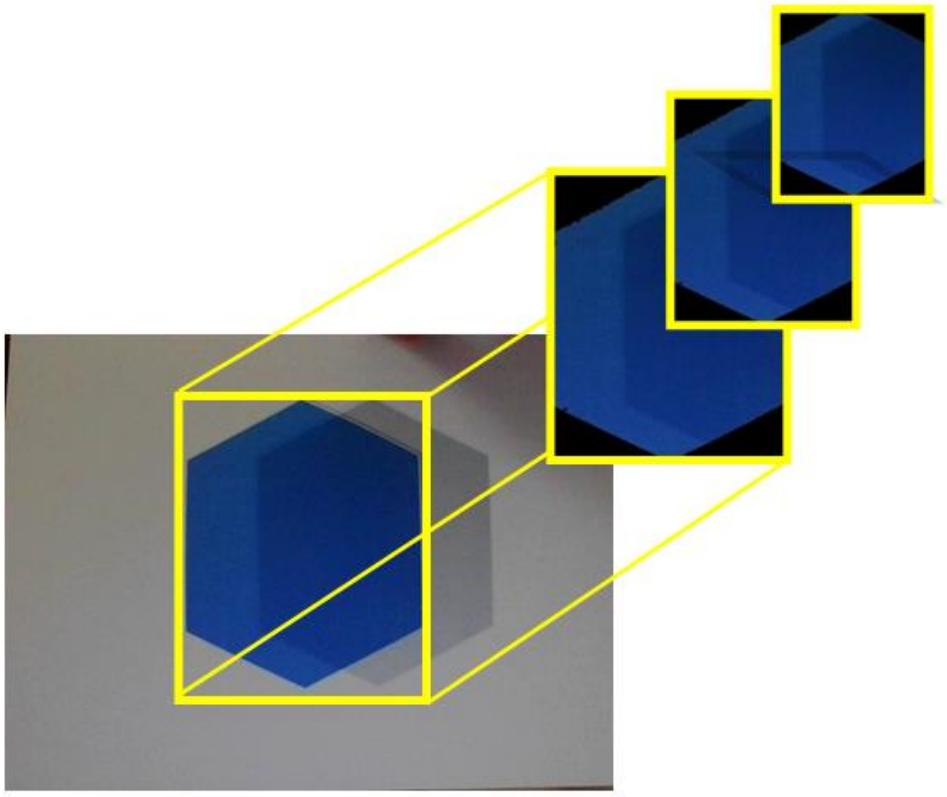


Figura 4.6 *– Căutarea șablonului în imaginea capturată folosindu-se mai multe scale*

4.2.4 Urmărirea obiectului

Ideea de bază a acestui algoritm este ca atunci când în imaginea captată de Nao a fost detectat obiectul, acesta să își miște capul pentru a centra obiectul în câmpul său vizual și de a se apropia de obiect în cazul în care se află mai departe.

Figura 4.7 (a) *– Detecția țintei în captură*

Figura 4.7 (b) *– Centrarea în câmpul vizual*

Algoritmul va porni de la calcularea coordonatelor poziției în care se află obiectul în imagine. Acestea sunt determinate prin rezultatul obținut în urma folosirii funcției de matchTemplate, în urmă căruia vom obține mai multe valori pe care le vom media astfel încât să obținem o valoare aproximativă și cu o precizie destul de bună. Rezultatul calculat reprezintă coordonatele colțului din stânga sus, însă ne vor trebui cele din centrul obiectului pe care le vom determina. Amintim că forma șablonului este dreptunghiulară.

Odată dedus unde se află centrului țintei, urmează calcularea noilor coordonate ale lui Nao pentru mișcarea capului. Va trebui ca robotul să își miște capul mai întâi în stânga și dreapta, apoi în sus și în jos. Mișcarea este limitată de motoarele lui Nao, iar în Figurile 4.8 (a) și (b) sunt reprezentate plajele de valori de mișcare ale capului.

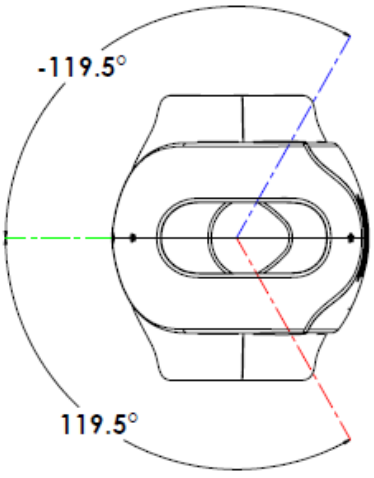


Figura 4.8 (a) *– Mișcarea capului stânga – dreapta [23]*

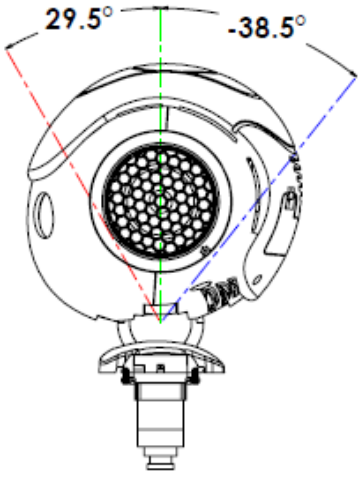


Figura 4.8 (b) *– Mișcarea capului sus – jos [23]*

Mișcarea de stânga – dreapta

Pentru a calcula cu câte grade este nevoie ca Nao să își miște capul pe orizontală, a fost nevoie de găsirea unor formule de calcul. Aceasta se va determina în funcție de o axă verticală ce separă captura în două părți egale. Scopul este de a stabili în ce sens își va roti robotul capul. Folosindu-mă de Choregraphe, am determinat câteva rezultate experimentale pentru a putea găsi formule următoare:

- Mișcare stânga

$$a = \frac{(320 - b) * 0.342}{222} \tag{24}$$

- Mișcare dreapta

$$a = \frac{(b - 320) * 0.342}{222} \tag{25}$$

unde a reprezintă numărul de radiani corespunzător valorii cu care ar trebui deplasat capul pe orizontală în stânga, respectiv dreapta axei și b este poziția pe verticală a centrului obiectului țintă, mai exact coloana pe care se situează, fiind valoarea aflată la pasul în care am calculat coordonatele centrului.

Valorile numerice din formule au fost deduse prin simularea procesului care ar trebui realizat în mod automat. Cu Choregraphe am înregistrat două poziții ale capului: una în care Nao are capul situat pe axa centrală, iar obiectul se află în stânga câmpului său vizual și una în care acesta are capul întors spre stânga cât să aibă obiectul centrat în imaginea pe care o vede. Cele două poziții, odată înregistrate în program, au fost exportate într-un script în limbajul Python, din care s-au extras coordonatele în radiani ale capului pe orizontală. Cu scopul de a afla câți radiani corespund unei anumite deplasări în funcție de un număr de pixeli stabilit, cele două valori extrase se vor scădea.

S-a obținut în acest mod că pentru o distanță de 222 de pixeli pe orizontală față de axa centrală, căreia îi corespunde coloana a 320-a de pixeli, sunt necesari 0.3420821 radiani. Prin regula de trei simplă se vor deduce astfel formulele de mai sus.

Mișcarea de sus-jos

În mod similar am determinat și formulele pentru mișcarea pe verticală, selectând de data aceasta o axă orizontală care să împartă imaginea în două părți egale. Față de aceasta am determinat experimental valorile pe care să le introduc în formulele calculate cu ajutorul regulii de trei simplă.

Am obținut în acest fel următoarele formule:

- Mișcare în sus

$$a = \frac{(240 - b) * 0.292}{123} \tag{26}$$

- Mișcare în jos

$$a = \frac{(b - 240) * 0.292}{123} \tag{27}$$

unde a reprezintă numărul de radiani corespunzător deplasării capului pe verticală deasupra sau dedesubtul axei, iar b este poziția pe orizontală a centrului obiectului vizat.

După obținerea valorilor corespunzătoare deplasărilor pe cele două direcții se vor folosi funcții ale robotului care vor realiza mișcarea efectivă a capului său, mai întâi pe verticală, apoi pe orizontală într-un timp suficient astfel încât mișcarea să pară cât mai naturală. În final, obiectul va fi centrat în câmpul vizual al robotului, sarcina fiind îndeplinită.

În continuare, Nao va trebui să se deplaseze către țintă în cazul în care aceasta a fost detectată și este prea mică. Ca să știe dacă este nevoie să se apropie de obiect, se va verifica dacă scala este cea mai mică, iar dacă aceasta îndeplinește condiția, atunci Nao se va mișca în față până când se va apropia de țintă.

Așa cum am menționat și mai devreme, etapele de detecție și urmărire se desfășoară în mod repetitiv până când este acționat butonul anterior de pe cap, care va declanșa ieșirea din buclă și reactivarea tuturor celorlalți senzori folosiți.

CAPITOLUL 5

Rezultate obținute

În urma implementării algoritmilor descriși în capitolele anterioare, am reușit realizarea unui sistem de extragere a obiectului țintă și de urmărire a acestuia în timp real care să funcționeze pe robotul Nao.

5.1 ACȚIONAREA BUTOANELOR

Comanda robotului se realizează cu ajutorul celor trei senzori tactili de la nivelul capului și de pe mâna dreaptă.

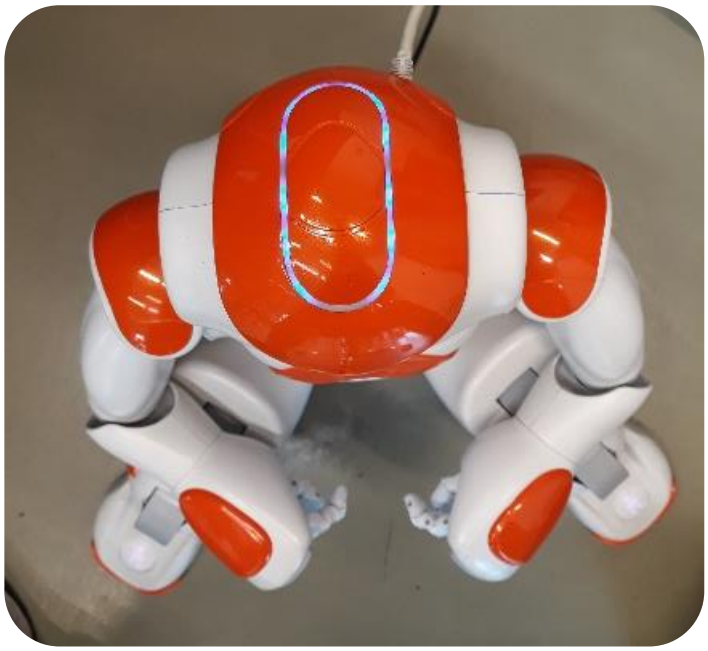


Figura 5.1 *– Butoanele tactile de la nivelul capului*

Figura 5.2 (a) *– Pornirea algoritmului de urmărire a obiectului*

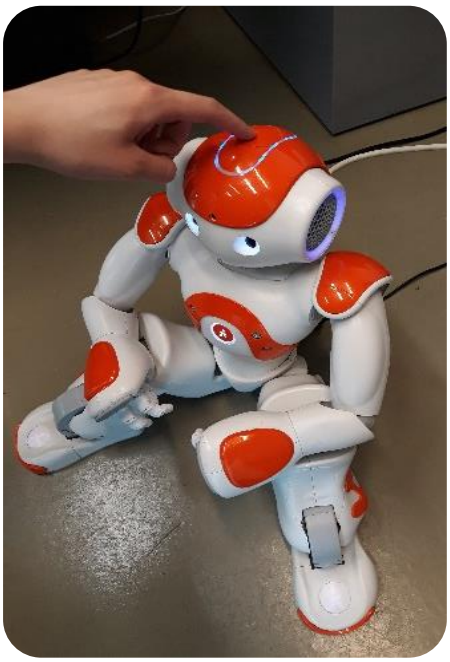


Figura 5.2 (b) *– Captura obiectului țintă*

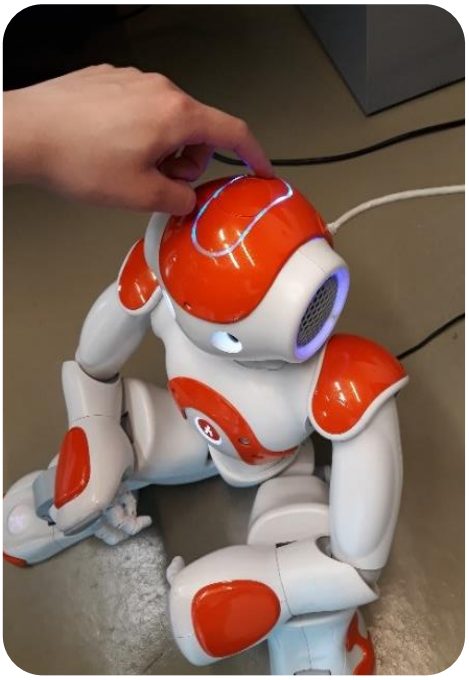


Figura 5.2 (c) *– Părăsirea programului de urmărire*

Figura 5.2 (d) *– Părăsirea aplicației*

5.2 PERFORMANȚELE ALGORITMULUI GMM

În urma unor capturi fotografice, am aplicat algoritmul de extragere al obiectului principal din fiecare imagine prin metoda mixturilor gaussiene. Rezultatele obținute pentru un hexagon albastru și pentru o stea roz sunt prezentate în ilustrațiile de mai jos conform pașilor după care au fost realizate:

1. Capturarea imaginilor

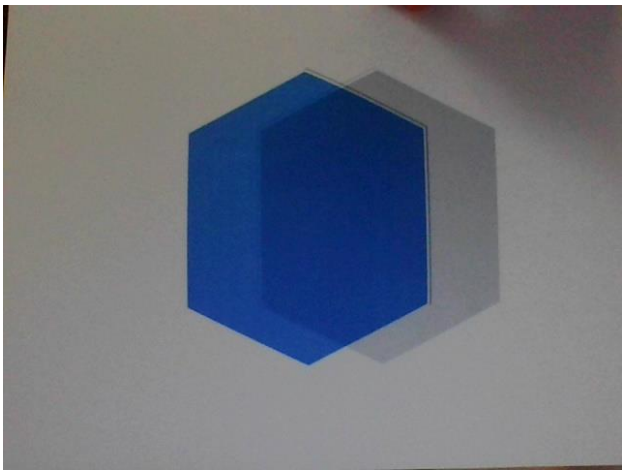


Figura 5.3 (a) *– Captură hexagon*

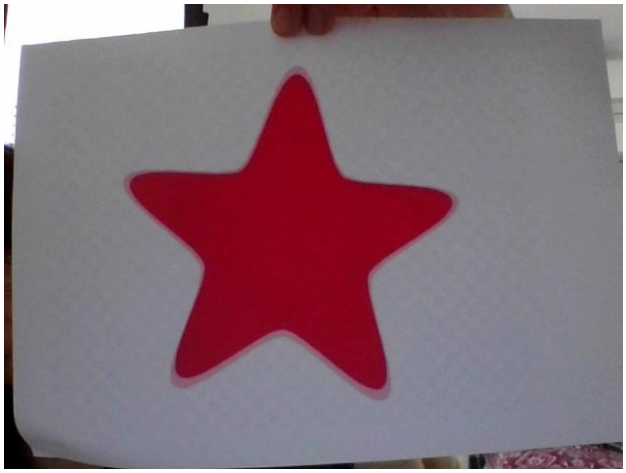


Figura 5.3 (b) *– Captură stea*

2. Extragerea pixelilor centrali imaginilor sursă

Figura 5.4 (a) *– Pixelii centrali ai imaginii cu hexagonul*

Figura 5.4 (b) *– Pixelii centrali ai imaginii cu steaua*

3. Extragerea obiectelor principale din imaginile sursă

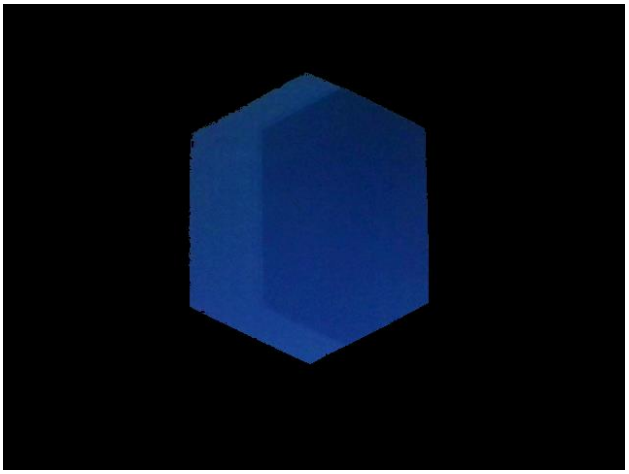


Figura 5.5 (a) *– Extragerea hexagonului*

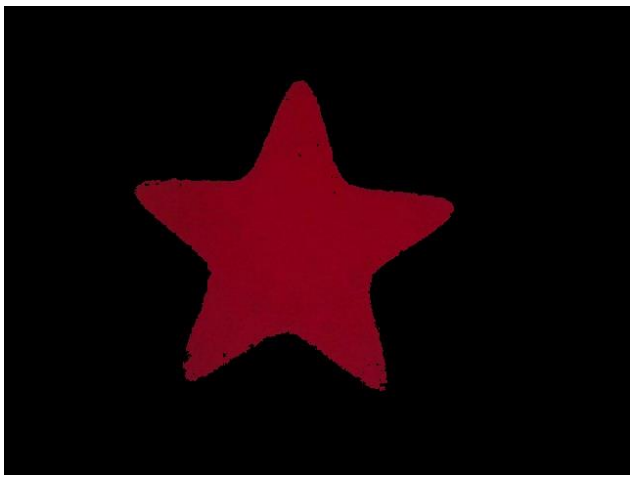


Figura 5.5 (b) *– Extragerea stelei*

4. Decuparea obiectelor principale și înregistrarea lor în memorie

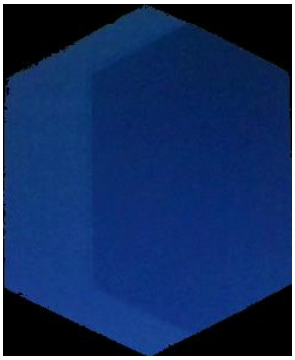


Figura 5.6 (a) *– Decuparea hexagonului*

Figura 5.6 (b) *– Decuparea stelei*

5.3 REZULTATELE DETECȚIEI OBIECTULUI ȘI ALE MIȘCĂRII ROBOTULUI

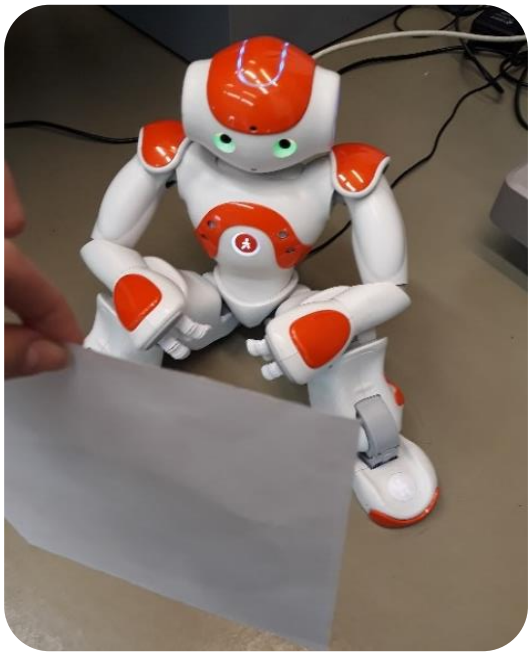


Figura 5.7 *– Detectarea obiectului*

Figura 5.8 (a) *– Urmărirea obiectului, perspectiva utilizatorului*

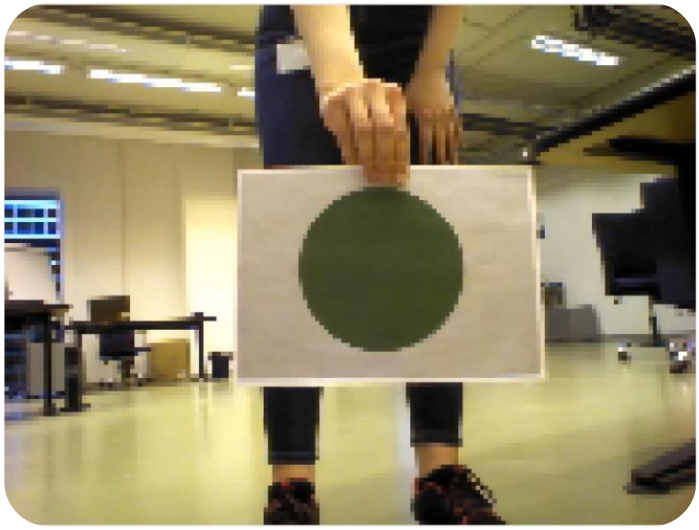


Figura 5.8 (b) *– Urmărirea obiectului, perspectiva lui Nao*

Figura 5.9 (a) *– Mișcarea lui Nao, prima ipostază*

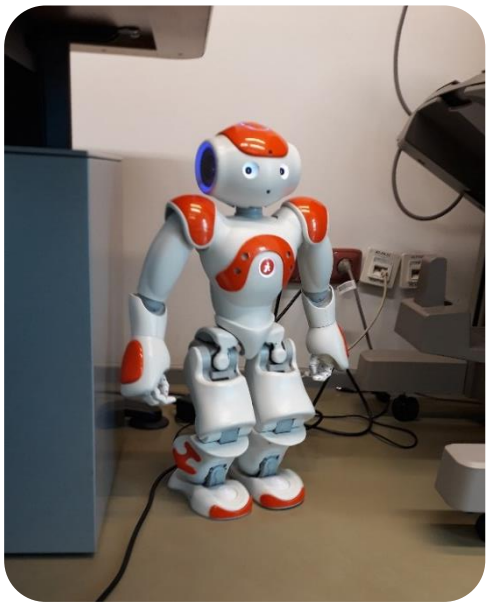


Figura 5.9 (b) *– Mișcarea lui Nao, a doua ipostază*

5.4 SOLUȚII IMPLEMENTATE

Aducem aminte ca Nao este un robot care are un sistem cu resurse limitate. Acest lucru impune o serie de probleme în momentul în care se dorește implementarea unor programe care să se încadreze în domeniul inteligenței artificiale.

În proiectul de față, am reușit să implementez o serie de soluții care să suplinească lacunele sistemului, reușind să ating în acest mod scopul pe care mi l-am propus. Printre soluțiile implementate se numără:

- Instalarea bibliotecii OpenCV pentru adăugarea unor resurse de prelucrare de imagine. A necesitat însă o versiune mai veche a acesteia care nu dispune de toate funcțiile necesare proiectului. Astfel, am implementat un algoritm suplinitor pentru modelele de mixturi gaussiene.
- Prelucrarea capturilor fotografice și nu a cadrelor video din cauza duratei mult mai mari a proceselor care se desfășoară pentru sarcina respectivă. Este de menționat și faptul că acest lucru nu a afectat în vreun fel desfășurarea normală a întregului proces.
- Rescrierea datelor ca urmare a limitării memoriei sistemului de lucru, folosindu-mă astfel într-un mod optim de spațiul pe care l-am avut la dispoziție.
- Din cauza puterii de procesare mică, am redus numărul de pixeli în prelucrarea imaginilor de extragere a obiectelor pe care doresc să le urmărească Nao, fără a influența rezultatele obținute și reduc astfel și timpul de procesare.
- Pentru activarea programelor aplicației, am înlocuit comenzile vocale cu acționarea senzorilor tactili pentru a evita desincronizarea capturilor fotografice față de procesul de urmărire al obiectului.

CONCLUZII

CONCLUZII GENERALE

Scopul principal al acestei teze a fost crearea unei aplicații care să îi permită lui Nao să urmărească diverse obiecte pe care le înregistrează în memoria sa, crescându-i astfel autonomia și dezvoltând o idee de bază de la care pot pleca module de inteligență artificială. În acest mod, șansele ca el să contribuie într-un mod semnificativ la o terapie pentru copii cu dizabilități sunt mult mai mari, iar randamentul va creste semnificativ.

Aplicația finală constă în două părți semnificative: înregistrarea unei ținte în memorie și urmărirea acesteia prin mișcări naturale ale corpului. În acest mod, Nao este capabil să fie atent la un obiect în mișcare și va putea stimula orientarea și interacțiunea unui copil care suferă de tulburări de spectru autist.

În momentul de față, robotul este comandat cu ajutorul unor senzori tactili pentru a putea lansa programele principale. Puțini fiind, aceștia însă sunt la îndemână oricui și de asemenea sunt și intuitivi. La baza programelor principale se află algoritmi de procesare de imagine care includ modele de mixturi gaussiene și potrivirea unor șabloane, dar și elemente de robotică pentru a putea coordona mișcările lui Nao în concordanță cu urmărirea unei ținte.

Obiectul pe care dorim să îl extragem dintr-o imagine capturată va trebui să fie un obiect simplu cu o singură culoare și cu un fundal în care detaliile să existe la un nivel minimalist. Acest lucru se întâmplă din cauza lipsei de resurse, a puterii de procesare destul de mică și a memoriei sale deoarece sistemul său nu permite instalarea unor module foarte puternice, iar resursele sale sunt limitate și insuficiente pentru anumite proiecte.

Cu toate acestea, Nao este un robot ideal pentru a fi integrat în ședințe de terapie, putând face față chiar și aplicația creată deoarece precizia și acuratețea cu care urmărește un obiect în timp real este suficient de bună, iar mișcările sale sunt naturale și nu lasă impresia că depinde de acțiunea unei persoane.

CONTRIBUȚII PERSONALE

Pentru a îmi atinge scopul propus prin acest proiect, contribuțiile mele asupra aplicației au fost următoarele:

- implementarea în limbajul de programare Python a modulelor puse la dispoziție de Naoqi
- implementarea algoritmului de extragere a unui obiect dintr-o imagine dată prin modele de mixturi gaussiene
- integrarea funcționalităților deja existente pe Nao cu cele importate din biblioteca OpenCv și codul realizat de mine
- implementarea unui algoritm de mișcare a corpului în mod autonom prin calcularea coordonatelor necesare

PLANURI DE VIITOR

Proiectul de față, așa cum am menționat și mai devreme, este un prim pas către crearea unui sistem autonom în întregime, iar datorită lucrărilor care au fost făcute de-a lungul timpului și care vor continua să se dezvolte în acest domeniu vor, se va putea duce la bun sfârșit scopul pe care ni-l dorim, și anume autonomia robotului.

Un pas care ar putea contribui la o astfel de aplicație este introducerea elementelor de machine learning, prin care Nao să poată fi capabil fie să învețe diverse obiecte, fie să i se integreze diferite clase deja învățate cu ajutorul unui alt sistem și care pot servi într-o terapie pentru copii. În acest fel, obiectele vizate nu ar mai fi limitate, ci vor fi recunoscute la nivel de clasă. Spre exemplu, dacă robotul știe că trebuie să urmărească o pisică, atunci el va căuta modelul acesteia în imaginile capturate, fără a căuta o pisică anume.

Pe lângă îmbunătățirile care îi pot fi aduse, robotul deja este capabil să poată fi introdus unor copii pentru a încerca să își îndeplinească scopul și pentru a vedea exact randamentul real al aplicației, după care se pot lua decizii și măsuri în privința dezvoltării și îmbunătățirii sale. Totodată, versiuni mai noi ale lui Nao ar putea funcționa mult mai bine și programarea acestuia s-ar realiza cu mai multă ușurință, iar aplicațiile ar fi concepute mult mai rapid.

Privind puțin în viitor, Nao deține potențialul unui bun asistent în terapiile pentru copii cu dizabilități pe care îi va putea ajuta să se dezvolte mintal, să învețe și să se bucure atunci când lucrează cu el. Astfel progresele celor mici se vor cunoaște, iar ajutorul pe care robotul îl acordă se realizează într-o manieră prietenoasă și atractivă pentru ei.

REFERINȚE

[1] Tomlinson Z., “15 Medical Robots That Are Changing the World”, disponibil online la adresa https://interestingengineering.com/15-medical-robots-that-are-changing-the-world, accesat la data: 12.06.2019

[2] Choudhury A., Li H., Greene C.M, "Humanoid Robot: Application and Influence", International Journal of Applied Science - Reserach and Review, No. 5, Vol. 17, 2018, DOI: 10.21767/2394-9988.100082, *CoRR, abs/1812.06090*.

[3] Koldewyn K., Weigelt S., Kanwisher N., Jiang Y., “Multiple Object Tracking in Autism Spectrum Disorders”, National Institutes of Health Public Access – Author Manuscript, No. 6, Vol. 43, 2013, DOI: :10.1007/s10803-012-1694-6

[4] Georgian Nicolae, „Sistem de interacțiune prin voce cu robotul Nao în limba română”, Lucrare de diploma, 2016

[5] Thompson D., “Meet Nao, the robot that helps treat kids with autism”, disponibil online la adresa https://medicalxpress.com/news/2018-05-nao-robot-kids-autism.html, accesat la data: 13.06.2019

[6] -, “Aldebaran documentation: *NAO* – Battery”, disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/battery_robot.html, accesat la data: 14.06.2019

[7] -, “Aldebaran documentation: Motherboard”, disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/motherboard_robot.html, accesat la data: 14.06.2019

[8] -, “Aldebaran documentation: Motors”, disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/motors_robot.html, accesat la data: 14.06.2019

[9] -, “Aldebaran documentation: Loudspeaker”, disponibil online la adresa http://doc.aldebaran.com/21/family/robots/loudspeaker_robot.html, accesat la data 15.06.2019

[10] -, “Aldebaran documentation: Microphones”, disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/microphone_robot.html, accesat la data: 15.06.2019

[11] -, “Aldebaran documentation: Nao – Video camera”, disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/video_robot.html, accesat la data: 15.06.2016

[12] -, “NAO Software 1.14.5 documentation: NAOqi Framework”, disponibil online la adresa http://doc.aldebaran.com/1-14/dev/naoqi/index.html, accesat la data:15.06.2019

[13] -, “NAO Software 1.14.5 documentation: NAOqi Vision”, disponibil online la adresa http://doc.aldebaran.com/1-14/naoqi/vision/index.html, accesat la data 16.06.2019

[14] -, “NAO Software 1.14.5 documentation: ALRedBallDetection”, disponibil online la adresa http://doc.aldebaran.com/1-14/naoqi/vision/alredballdetection.html#alredballdetection, accesat la data: 16.06.2019

[15] -, ”NAO Software 1.14.5 documentation: ALVisionRecognition”, disponibil online la adresa http://doc.aldebaran.com/1-14/naoqi/vision/alvisionrecognition.html#alvisionrecognition, accesat la data: 16.06.2019

[16] -, “OpenCV: Template Matching”, disponibil online la adresa https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html, accesat la data: 21.06.19

[17] -, “OpenCV: Manual calculation of template matching”, disponibil online la adresa https://answers.opencv.org/question/63587/manual-calculation-of-template-matching/, accesat la data: 21.06.19

[18] Christopher M. Bishop (2006), “Pattern Recognition and Machine Learning”, Editura Springer, ISBN-10: 0-387-31073-8

[19] -, “RobotLAB: Nao Power V6 Educator Pack”, disponibil online la adresa https://www.robotlab.com/store/nao-power-v6-educator-pack, accesat la data: 12.06.2019

[20] -, “Aldebaran documentation: LEDs: H25 LEDs”, disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/leds_robot.html, accesat la data: 23.06.2019

[21] Patil P., “K Means Clustering: Identifying F.R.I.E.N.D.S in the World of Strangers”, disponibil online la adresa https://towardsdatascience.com/k-means-clustering-identifying-f-r-i-e-n-d-s-in-the-world-of-strangers-695537505d, accesat la data: 23.06.2019

[22] -, “NAO Software 1.14.5 documentation: Contact and tactile sensors”, disponibil online la adresa http://doc.aldebaran.com/1-14/family/robots/contact-sensors_robot.html, accesat la data: 23.06.2019

[23] -, "NAO Documentation: Joints", disponibil online la adresa http://doc.aldebaran.com/2-1/family/robots/joints_robot.html, accesat la data: 23.06.2019

ANEXA 1

```
def initialize_centroids(points, k):
    """returns k centroids from the initial points"""
    centroids = points.copy()
    np.random.shuffle(centroids)
    return centroids[:k]

def closest_centroid(points, centroids):
    """returns an array containing the index to the nearest centroid for each
point"""
    distances = np.sqrt(((points - centroids[:, np.newaxis])**2))
    return np.argmin(distances, axis=0)

def move_centroids(points, closest, centroids):
    """returns the new centroids assigned from the points closest to them"""
    return np.array([points[closest==k].mean(axis=0) for k in
range(centroids.shape[0])])


def kMeans(X, K, maxIters = 10):
    centroids = initialize_centroids(X, K)
    for it in range(maxIters):
        closest = closest_centroid(X, centroids)
        centroids = move_centroids(X, closest, centroids)
    clusters = closest
    return np.array(centroids) , clusters

def GMM_me(X, K, maxIters = 3):
    # K-means init
    (mu, C) = kMeans(X, K)
    alpha = np.array([np.sum(C==k) for k in range(K)])
    alpha = alpha.astype(float)/np.sum(alpha).astype(float)
    sigma = np.array([X[C == k].std(axis = 0) for k in range(K)])
    it = 1
    L_new = log_likelihood(X, K, alpha, mu, sigma)
    while(it < maxIters):
        L_old = L_new
        # Expectation step
        w_ik = expectation_step(X, K, alpha, mu, sigma)
        # Maximization step
        (alpha, mu, sigma) = maximization_step(X, K, w_ik)
        # Log - likelihood
        L_new = log_likelihood(X, K, alpha, mu, sigma)
        print(it)
#        print(it, mu, sigma, '\n')
        if(np.abs(L_old-L_new)<1e-7):
            break
        it = it + 1
    return (alpha, mu, sigma)

def expectation_step(X, K, alpha, mu, sigma):
    N = X.shape[0]
    w_ik = np.zeros(shape=(K,N), dtype = float)
    for i in range(N):
        for k in range(K):
            w_ik[k,i] = alpha[k]*(1/(np.sqrt(2*np.pi*(sigma[k]**2))))*np.exp(-
((X[i]-mu[k])**2) / (2*(sigma[k]**2)))
```

```
    w_ik = w_ik / w_ik.sum(axis=0)[None,:]
    return w_ik

def maximization_step(X, K, w_ik):
    N = X.shape[0]
    N_k = np.zeros(shape=(K,1), dtype = float)
    alpha = np.zeros(shape=(K,1), dtype = float)
    for k in range(K):
        for i in range(N):
            N_k[k] = N_k[k] + w_ik[k,i]
        alpha[k] = N_k[k]/N
    mu = np.zeros(shape=(K,1), dtype = float)
    for k in range(K):
        for i in range(N):
            mu[k] = mu[k] + w_ik[k,i]*X[i]
        mu[k] = mu[k] / N_k[k]
    sigma = np.zeros(shape=(K,1), dtype = float)
    for k in range(K):
        for i in range(N):
            sigma[k] = sigma[k] + w_ik[k,i]*(X[i]-mu[k])**2
        sigma[k] = sigma[k] / N_k[k]
        sigma[k] = np.sqrt(sigma[k])
    return (alpha, mu, sigma)


def log_likelihood(X, K, alpha, mu, sigma):
    N = X.shape[0]
    L = 0
    for i in range(N):
        l = 0
        for k in range(K):
             l = l + alpha[k]*(1/(np.sqrt(2*np.pi)*sigma[k]))*np.exp(-((X[i]-
mu[k])**2) / (2*(sigma[k]**2)))
        if l != 0:
            L = L + np.log(l)
    return L
```

ANEXA 2

```
img = cv2.imread("poza.jpg")
obj = cv2.imread("poza_decupata.jpg")
w = 640
h = 480

img_b,img_g,img_r = cv2.split(img)
obj_b,obj_g,obj_r = cv2.split(obj)

#determinare gaussiene canale imagine
mu_img_b, sigma_img_b, mu_img_g, sigma_img_g,mu_img_r, sigma_img_r =
gaussiene_canale_img(img_b, img_g, img_r)

x_img_x = np.linspace(mu_img_b - 3*sigma_img_b, mu_img_b + 3*sigma_img_b, 100)

#determinare medie canale obiect
mu_obj_b, mu_obj_g, mu_obj_r = medie_canale(obj_b, obj_g, obj_r)
#<----- BLUE------>
mu_vect_b = np.ones([3])
for i in range(0,3):
    mu_vect_b[i] = (mu_img_b[i][0] - mu_obj_b)*(mu_img_b[i][0] - mu_obj_b)
#calcul valoare de referinta
index_b = min_mu(mu_vect_b)
mu_b = mu_img_b[index_b][0]
sigma_b = sigma_img_b[index_b][0]



#<----- GREEN------>
mu_vect_g = np.ones([3])
for i in range(0,3):
    mu_vect_g[i] = (mu_img_g[i][0] - mu_obj_g)*(mu_img_g[i][0] - mu_obj_g)
#calcul valoare de referinta
index_g = min_mu(mu_vect_g)
mu_g = mu_img_g[index_g][0]
sigma_g = sigma_img_g[index_g][0]



#<----- RED------>
mu_vect_r = np.ones([3])
for i in range(0,3):
    mu_vect_r[i] = (mu_img_r[i][0] - mu_obj_r)*(mu_img_r[i][0] - mu_obj_r)
#calcul valoare de referinta
index_r = min_mu(mu_vect_r)
mu_r = mu_img_r[index_r][0]
sigma_r = sigma_img_r[index_r][0]

#parcurgem imaginea captata pe cele 3 canale separate si comparam valoarea cate
unui pixel cu referintele alese mai sus
img_finala = cv2.imread("poza.jpg")

#cautam si limitele obiectului pentru a putea face apoi croparea pentru template
i_min = h
i_max = 0
```

```
j_min = w
j_max = 0

for i in range (0,h):
    for j in range (0,w):
        if(pixel_negru(img_b[i,j], img_g[i,j], img_r[i,j], mu_b, sigma_b, mu_g,
sigma_g, mu_r, sigma_r) == 1):
            img_finala[i,j] = [0,0,0]
        else:
            #calcul margini obiect
            if i < i_min:
                i_min = i
            if i > i_max:
                i_max = i
            if j < j_min:
                j_min = j
            if j > j_max:
                j_max = j
cv2.imwrite("poza_finala.jpg", img_finala)

#cropare imagine
print('i_min, i_max, j_min, j_max', i_min, i_max, j_min, j_max)
image_crop = img_finala[i_min:i_max, j_min:j_max]
cv2.imwrite("poza_finala_cropata.jpg", image_crop)
```

ANEXA 3

```
p = [255,255,255]
        green = [0,255,0]
        leds = ALProxy("ALLeds","127.0.0.1",9559)

        motion = ALProxy("ALMotion","127.0.0.1",9559)
        names_m  = 'Body'
        motion.setStiffnesses(names_m, 1)
        names = list()
        times = list()
        keys = list()

        names.append("HeadPitch")
        times.append([0.6])
        keys.append([[-0.0537319, [3, -0.2, 0], [3, 0, 0]]])

        names.append("HeadYaw")
        times.append([0.6])
        keys.append([[-0.0521979, [3, -0.2, 0], [3, 0, 0]]])
        motion.angleInterpolationBezier(names, times, keys)
        motion.setStiffnesses(names_m, 0)

        template = cv2.imread('greenCirc.jpg', 0)

        w, h = template.shape[::-1]

        avgX = 0
        avgY = 0

        photoCaptureProxy = ALProxy("ALPhotoCapture", "127.0.0.1", 9559)
        photoCaptureProxy.setResolution(2)
        photoCaptureProxy.setPictureFormat("jpg")

        pozPitch = -0.0537319
        pozYaw =  -0.0521979

        #connect to rear button
        self.id2 = self.touch2.signal.connect(functools.partial(self.onTouched2,
"RearTactilTouched"))
        ReactToTouch.startLoop = 1
        while ReactToTouch.startLoop == 1:
            avgX = 0
            sumX = 0
            avgY = 0
            sumY = 0
            nr = 0

            photoCaptureProxy.takePictures(1, "/home/nao/obj-det-track/", "image")
            gray = cv2.imread('image.jpg', 0)

            res = cv2.matchTemplate (gray, template, cv2.TM_CCOEFF_NORMED)
            threshold = 0.6
            loc = np.where(res >= threshold)

            for pt in zip(*loc[::-1]):
```

```
                #cv2.rectangle(frame, pt, (pt[0]+w, pt[1]+h), (0,255,255),
thickness = 1)
                nr = nr + 1
                sumX = sumX + pt[0]
                sumY = sumY + pt[1]

            if nr != 0:
                leds.post.fadeRGB("FaceLeds", 256*256*green[0] + 256*green[1] +
green[2],1)

                avgX = sumX/nr
                avgY = sumY/nr

                avgX = avgX + w/2
                avgY = avgY + h/2

                names = list()
                times = list()
                keys = list()

                motion.setStiffnesses(names_m, 1)

                if avgY < 240:
                    moveY = ((240 - avgY) * 0.292952) / 123
                    names.append("HeadPitch")
                    times.append([0.6])
                    keys.append([[pozPitch - moveY, [3, -0.2, 0], [3, 0, 0]]])
                    pozPitch = pozPitch - moveY
                else :
                     moveY = ((avgY - 240) * 0.292952) / 123
                     names.append("HeadPitch")
                     times.append([0.6])
                     keys.append([[pozPitch + moveY, [3, -0.2, 0], [3, 0, 0]]])
                     pozPitch = pozPitch + moveY

                if avgX < 320:
                    moveX = ((320 - avgX) * 0.3420821) / 222
                    names.append("HeadYaw")
                    times.append([0.6])
                    keys.append([[pozYaw + moveX, [3, -0.2, 0], [3, 0, 0]]])
                    pozYaw = pozYaw + moveX
                else :
                    moveX = ((avgX - 320) * 0.3420821) / 222
                    names.append("HeadYaw")
                    times.append([0.6])
                    keys.append([[pozYaw - moveX, [3, -0.2, 0], [3, 0, 0]]])
                    pozYaw = pozYaw - moveX
                if moveX > 0.015 or moveY > 0.015:
                    motion.angleInterpolationBezier(names, times, keys)
                motion.setStiffnesses(names_m, 0)
            else:
                leds.post.fadeRGB("FaceLeds", 256*256*p[0] + 256*p[1] + p[2],1)
```