

Universitatea Politehnica București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Inteligența artificială în robotul NAO

Lucrare de Licență

prezentată ca cerință parțială pentru obținerea *Titlului de Inginer* în
domeniul *Electronică, Telecomunicații și Tehnologia Informației*,
programul de studii *Microoptoelectronică și Nanotehnologii*

Conducători științifici
**Prof. Dr. Ing. Corneliu
BURILEANU**

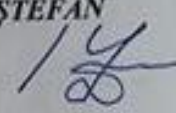
**Ș. L. Dr Ing Horia
CUCU**

Absolvent

**Vlad-Constantin
FURTUNĂ**

Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Departamentul Dispozitive, Circuite și Arhitecturi Electronice

Aprobat Director de Departament:
Prof. Dr. Ing. Gheorghe M. ȘTEFAN



TEMA PROIECTULUI DE DIPLOMĂ
a studentului FURTUNĂ B. VLAD-CONSTANTIN

1. Titlul temei: **Inteligența artificială în robotul NAO**

2. Contribuția practică, originală a studentului va consta în:

Acest proiect de diplomă își propune implementarea unui modul de inteligență artificială folosind robotul umanoid NAO, fiind construit din următoarele elemente:

- Portarea, optimizarea și testarea unui sistem de inteligență artificială folosind sistemul de operare prevăzut pe robot (NAOqi).
- Dezvoltarea unui modul de prelucrare digitală a imaginilor folosind bibliotecile deja existente pe robot și construirea bibliotecilor dedicate aplicației de inteligență artificială.
- Crearea unui modul de realizare a răspunsurilor la stimuli externi, folosind senzorii optici, auditivi și tactili ai robotului, împreună cu motoarele existente pe robot și antrenarea sistemului de inteligență artificială pentru execuția corectă a răspunsurilor la stimuli.

Pentru realizarea proiectului se vor folosi următoarele module software: Python GUI, Choreographe, NAOqi, C++ SDK. Studentul va interconecta modulele create și optimizate, în vederea realizării unei aplicații practice a proiectului de diplomă.

3. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline: *Programarea Calculatoarelor, Structuri de date și algoritmi, Programare Obiect Orientată, Arhitectura Microprocesoarelor, Microcontrolere.*

4. Proprietatea intelectuală asupra proiectului aparține: *Studentului, Conducătorilor de lucrare.*

5. Locul de desfășurare a activității: *UPB-ETTI, Sala 3 de Calculatoare.*

6. Realizarea practică rămâne în proprietatea: *Studentului, UPB.*

7. Data eliberării temei: *01.06.2015*

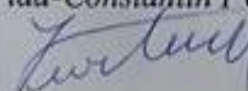
CONDUCĂTOR LUCRARE:

STUDENT:

Prof. Dr. Ing Corneliu BURILEANU

Vlad-Constantin FURTUNĂ

Ș. L. Horia CUCU



Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul “*Sistem de inteligență artificială implementată pe robot umanoid, 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, Telecomunicații și Tehnologia Informației*, programul de studii *Microoptoelectronică și Nanotehnologii* 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 măsurărilor 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, 25. 06. 2016

Absolvent *Vlad-Constantin Furtuna*

(semnătura în original)

Cuprins

LISTA DE FIGURI	9
LISTA DE TABELE.....	10
LISTA DE ACRONIME	11
INTRODUCERE	13
CAPITOLUL I.....	17
NAO-ROBOTUL UMANOID AUTONOM	17
1.1 CARACTERISTICI GENERALE	18
1.2 CARACTERISTICI HARDWARE	19
1.2.1 Construcția.....	19
1.2.2 LED-uri	19
1.2.3 Alimentare.....	20
1.2.4 Conectivitate	20
1.2.5 Audio.....	20
1.2.6 Camere Video.....	20
1.2.7 Dispozitive Infra-Rosii si sonare.....	20
1.2.8 Girometrul si Accelerometrul.....	20
1.2.9 Senzori de presiune, pozitie si contact	21
1.2.10 Unitățile de comanda si control	21
1.2.11 Actuatore si grade de libertate.....	22
1.3 CARACTERISTICI SOFTWARE.....	24
CAPITOLUL II.....	27
MEDII DE PROGRAMARE ȘI DEZVOLTARE SOFTWARE UTILIZATE.....	27
2.1 CHOREOGRAFHE.....	27
2.2 PROGRAMAREA ORIENTATĂ PE OBIECTE.....	31
2.3 LIMBAJUL PYTHON.....	33
2.4 NUMPY.....	34
2.5 OPENCV.....	35
2.6 NAOQI	36
CAPITOLUL III.....	41
JOCUL DE X ȘI 0 ȘI IMPLEMENTAREA ACESTUIA	41
3.1 CONCEPTUL APLICATIEI.....	41
3.2 MODULUL MAIN	42
3.3 MODULUL DE PROCESARE AL IMAGINILOR	42
3.4 MODULUL DE RECUNOASTERE VOCALA.....	46
3.5 MODULELE DE STRATEGIE	47
3.6 MODULUL DE MIȘCARE ȘI ANALIZĂ A STĂRII JOCULUI.....	49
CAPITOLUL IV	57
CONCLUZII	57
BIBLIOGRAFIE.....	59

LISTA DE FIGURI

Figura [1] 0.1 Functionarea generica a unui agent inteligent..... 14

Figura [2] 1.1 Robotul NAO..... 18

Figura [3] 1.2 Dimensiunile robotului NAO..... 19

Figura [4] 1.3 Schema bloc functionala a procesorului Intel Atom Z350..... 21

Figura [5] 1.4 Arhitectura electronică a robotului NAO..... 22

Figura [6] 1.5 Motoarele Robotului.. 23

Figura [7] 1.6 Legatura intre nivelul software inferior si cel superior prin DCM 25

Figura [8] 2.1 Librăriile predefinite ale robotului NAO 27

Figura [9] 2.2 Folosirea unei comenzi predefinite 28

Figura [10] 2.3 NAO executând Tai Chi Chuan vizualizat prin Robot Pose 30

Figura [11] 2.4 Utilitarul Robot Applications..... 30

Figura [12] 2.5 Aplicatia Video Monitor implementata in Choreographe..... 31

Figura [13] 2.6 Clasele utilizate în aplicația curentă..... 32

Figura [14] 2.7 Modularitatea Naoqi 37

Figura [15] 3.1 Conceptul aplicatiei software..... 41

Figura [16] 3.2 Împărțirea imaginii HSV în două imagini distincte 45

Figura [17] 3.3 Fereastra GAME 45

Figura [18] 3.4 Diagrama de stări a algoritmului de decizie..... 49

Figura [19] 3.5 Numerotarea căsuțelor pe tabla de joc 54

Figura [20] 3.6 Indicarea pozitiei 1 folosind un robot simulat..... 54

Figura [21] 3.7 Valorile actuatorilor în momentul execuției unei mișcări. 55

LISTA DE TABELE

Tabela 1.2-1 LED-uri prezente pe robot19

Tabela 1.2-2 Gradele de libertate ale robotului NAO.....22

LISTA DE ACRONIME

A

ARM = Advanced RISC Machine 21

C

CISC = Complex Instruction Set Computing 21

H

HSV = Hue Saturation Value 38

I

IEEE = Institute of Electrical and Electronics Engineers 20

IP = Internet Protocol 20

G

GUI = Graphical User Interface 20

N

Numpy = Numerical Python 34

O

OpenCV = Open Source Computer Vision 34

R

RISC = Reduced Instruction Set Computing 21

RGB = Red Green Blue 19

RJ = Registered jack 20

S

SSH = Secure Shell 26

Y

YUV = Luminance (Y), blue–luminance (U), red–luminance (V) 38

INTRODUCERE

Încă din timpuri străvechi, inteligența umană a fost un domeniu pe care oamenii au încercat să îl înțeleagă și să dezvolte teorii despre el. Omul s-a denumit pe șine însuși homo-sapiens, omul intelligent, întrucât capabilitățile noastre mentale sunt importante pentru viețile noastre. Inteligența artificială încearcă să înțeleagă mai bine funcționarea creierului uman, un organ atât de mic, comparativ cu lumea pe care o înțelege. Pe lângă înțelegerea funcționării corpului uman, inteligența artificială încearcă să construiască entități inteligente și să le folosească în scopul ușurării vieții umane.

Inteligența artificială acoperă o varietate de domenii, comparabile cu posibilitățile creierului uman, fiind folosită la momentul actual pentru roboți inteligenți ce reușesc să depășească posibilitățile omului în domenii precum șahul, unde roboții au depășit încă din anii 1990 marii maeștri ai jocului. Acest câmp de cercetare se prezintă ca fiind unul universal, capabil să integreze în șine orice tip de algoritm sau senzație cognitivă.

Inteligența artificială reprezintă inteligența pe care roboții sau mașinile sau instrumentele software o arată și de asemenea, termenul ce definește studiul și dezvoltarea mașinilor inteligente de calcul și achiziție în scopul maximizării șanselor de succes. Termenul a fost introdus în 1955 de către John McCarthy. Problemele centrale pe care acesta încearcă să le rezolve sunt următoarele: percepția, planificarea, învățarea și cunoașterea.

Motivul dezvoltării acestui câmp de cercetare a fost încercarea dezvoltatorilor de a recrea cu ajutorul circuitelor logice, ce stau la bază procesoarelor de orice tip, un creier uman capabil să înțeleagă și să învețe să răspundă la stimuli exteriori. La momentul actual, posibilitățile de simulare a creierului uman sunt cu mult sub potențialul dorit, puterea de procesare fiind inferioară creierului uman.

Istoria inteligenței artificiale începe în anul 1943 când Warren McCulloch și Walter Pitts au îmbinat trei domenii distincte: psihologia, logica și neuroștiința propunând un model în care fiecare neuron simulat are două stări: 1 și 0 iar fiecare “neuron” este influențat de numărul de electroni activați în vecinătatea sa. Aceștia au sugerat că acest model ar putea fi folosit pentru învățare. Munca acestora a fost continuată în anii 1950 de către Alan Turing și Claude Shannon ce dezvoltau motoare de șah pentru computere de tip von-Neumann. În anul 1955 John McCarthy introduce termenul de inteligență artificială. Deși inițial s-a crezut într-o dezvoltare mai rapidă a domeniului, Herbert Simon spunând în anul 1957 că în următorii 10 ani un computer va surclasa oamenii la șah, acest lucru s-a întâmplat abia cu 30 de ani mai târziu, în 1997 când Gary Kasparov a fost învins de către DeepBlue într-un meci de șah organizat la Philadelphia.

Fără îndoială, pentru succesul inteligenței artificiale, este nevoie de două lucruri: inteligență și unelte de dezvoltare. Procesoarele reprezintă unealtă necesară umanității pentru atingerea a culmi cât mai înalte în dezvoltarea de agenți inteligenți. Încă din anii 1940, începând cu prima mașină Turing, folosită de armata engleză pentru decodificarea mesajelor criptate de Germania Nazistă cu ajutorul mașinii Enigma. Tendința actuală este reprezentată cel mai bine de legea lui Moore, postulată în anul

1965 ce prezice dublarea numărului de tranzistoare pe aria chipului la fiecare doi ani, iar costurile se reduc cu o rată asemănătoare. [1]

În domeniul inteligenței artificiale se definește termenul de agent inteligent ca fiind o entitate ce observă mediul în care există prin intermediul senzorilor și își îndreaptă activitatea către îndeplinirea unui scop. Aceștia sunt capabili să învețe și să folosească o varietate de cunoștințe în vederea îndeplinirii unui scop. Un agent inteligent poate fi considerat și unul dintre cele mai simple dispozitive, un termostat, spre exemplu, sau dintre cei mai complecși, precum un robot umanoid cu trăsături faciale complexe și putere mare de procesare.

Generic, senzorii sunt cei ce leagă roboții de acest tip de lumea exterioară, făcându-i capabili să înțeleagă mediul înconjurător. După achiziționarea datelor legate de mediu, aceștia decid pe baza unor modele comportamentale de tipul dacă-atunci ce acțiuni trebuie să întreprindă. Îndeplinirea acestora se face cu ajutorul actuatorilor, acestea fiind cele care duc la modificarea mediului din care robotul face parte pentru îndeplinirea scopului inițial. În figura următoare se descrie schema generic de funcționare a unui agent inteligent.[2]

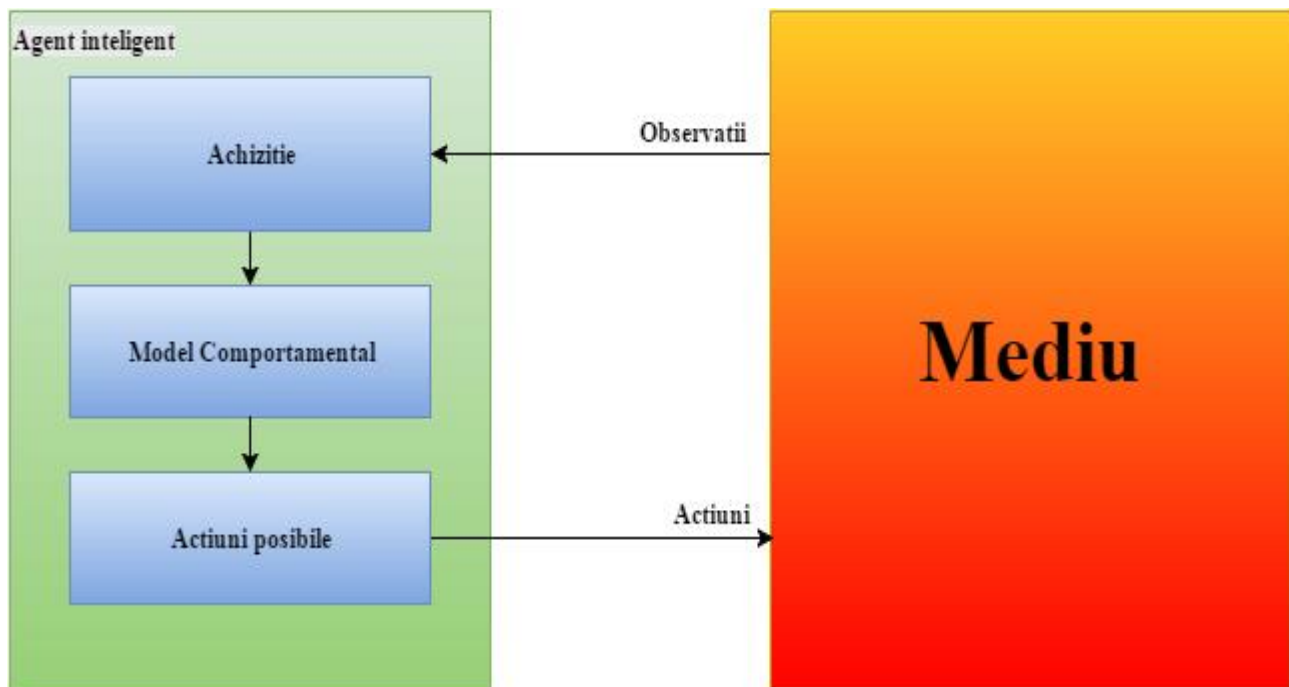


Figura [1] 0.1 Functionarea generica a unui agent inteligent

Întrucât la momentul actual acest domeniu este încă la început și departe de a-și fi arătat adevăratul potențial, ce pare a fi infinit, poetic vorbind, poate fi definit ca “domeniul în care vreau să lucrez”, ca locul în care noul Einstein se poate afirma, întrucât celelalte domenii sunt la stadii avansate de dezvoltare și necesită cunoștințe vaste pentru progres.

Necesitatea explorării modului în care agenții inteligenți funcționează survine din nevoia de progress a umanității și din încercarea de a moderniza prin orice mijloace modul în care aceștia influențează existența umană, aceștia devenind din ce în ce mai abili și cu o putere din ce în ce mai mare de procesare. La momentul actual, aceștia încep să acapareze piețele, fascinând prin apropierea lor de creatorii lor umani. În lucrarea curentă

se va explicita modul în care agentul inteligent, robotul NAO interacționează cu mediul înconjurător în vederea obținerii unor rezultate cât mai bune în momentul în care acesta este supus unor stimuli de tip viziune, vorbire sau mișcare.

CAPITOLUL I

NAO-ROBOTUL UMANOID AUTONOM

Un robot umanoid este un robot ce seamănă din punct de vedere constructiv cu corpul uman, folosit în principal pentru experimente ce presupun mișcări de locomoție și alte tipuri de mișcări complexe. În construcția acestora se încearcă apropierea cât mai bună de corpul uman, aceștia prezentând, în general, un cap, mâini, picioare, abdomen. Aceștia sunt folosiți pentru cercetarea locomoției și biomecanicii corpului uman. O parte dintre aceștia se utilizează pe anumite particularități ale corpului uman precum față, mână, piciorul fiind construiți și estetic pentru a replica înfățișarea umană detaliat, cât și acțiunile umane.[3]

Termenul de autonomie se referă la capacitatea acestora de a îndeplini anumite funcții cu un anumit grad de autonomie fiind de dorit în domenii precum: tratarea autismului, în care NAO este folosit intens, explorarea spațiului, curățarea caselor sau livrarea de bunuri și servicii. Există roboți autonomi precum cei utilizați în fabrici care sunt autonomi având doar câteva grade de libertate, spre exemplu, un robot care mută cutii, nu va avea libertatea de a-și alege poziția din care va apuca cutia pentru a o apuca mai ușor. Un robot complet autonom trebuie să îndeplinească următoarele condiții: să culeagă informație despre mediul în care se situează, să muncească un timp îndelungat fără a fi nevoiea unei intervenții umane, să se poată muta prin mediul în care operează fără a fi nevoie de intervenție exterioară și să evite situații care ar putea fi dăunătoare pentru oameni sau mediul înconjurător. O altă caracteristică a roboților autonomi este aceea că pot comunica cu oamenii, pot înțelege mediul înconjurător sau pot anunța probleme pe care le întâmpină, precum un status scăzut al bateriei sau supraîncălzirea robotului.[4]

Luând în considerare toate cele menționate anterior, se poate afirma că NAO este un robot *umanoid autonom*, de dimensiuni reduse, constituit din cap, cutie toracică, două mâini și două picioare, în cele ce urmează se vor descrie pe larg caracteristicile constructive ale acestuia.

1.1 CARACTERISTICI GENERALE

Construit de Aldebaran Robotics, robotul NAO este un robot conceput cu scopul de a deveni un companion de nădejde în viața de zi cu zi, având caracteristicile unui robot de dimensiuni reduse și putere de calcul sporită. Este capabil să parcurgă distanțe rezonabile, să vorbească, să asculte, să învețe obiecte, să învețe fețe și să le recunoască ulterior. A fost lansat în anul 2006 și a evoluat în mod constant de la o generație la alta, devenind astăzi unul dintre cei mai utilizați roboți în educație și cercetare, fiind folosit în peste 5000 de locații de către universități și centre de cercetare. Este un robot interactiv și personalizabil ce reușește să devină un companion de nădejde pentru orice programator. Aplicațiile concepute pentru acesta sunt limitate doar de nevoile și imaginația programatorului.



Figura [2] 1.1 Robotul NAO

După cum se poate observa în figura 1.1, robotul Nao este unic datorită design-ului, având o formă umanoidă și o mobilitate ridicată reușind să replice cu ușurință mișcările corpului uman, având motoare de tip DC, ce îi permit acestuia să efectueze mișcări de o complexitate ridicată.

Nao este utilizat în mai bine de 70 de țări ca support didactic pentru cursurile de robotică, programare obiect-orientată, știința calculatoarelor din școlile primare până la nivel de universitate. Cu ajutorul lui, elevii și studenții pot aprofunda limbaje de programare la un nivel avansat, robotul oferind o manieră simplă și distractivă pentru învățare. Robotul NAO reușește să atragă atenția dezvoltatorilor întrucât deține o putere impresionantă de calcul și este un mediu potrivit dezvoltării aplicațiilor software.[5]

1.2 CARACTERISTICI HARDWARE

1.2.1 Construcția

În cele ce urmează vor fi prezentate detaliile tehnice ale robotului conform cu foile sale de catalog. Robotul NAO este un robot de dimensiuni medii și greutate medie. Înălțimea acestuia este de 574 mm, anvergura mâinilor este de 311 mm iar lățimea acestuia este de 275 mm. Greutatea acestuia este de 5.4 kilograme, fiind construit din două tipuri de plastic rezistent la căldură, ABS-PC/PA-66, și un tip de plastic rezistent la șocuri mecanice, XCF-30, ce are în componența și fibră de carbon în proporție de 30%[6].

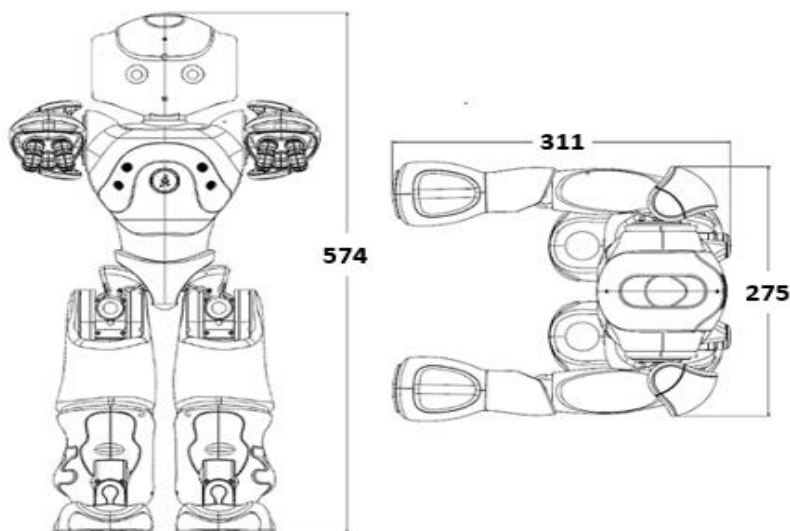


Figura [3] 1.2 Dimensiunile robotului NAO

1.2.2 LED-uri

Robotul are instalat o multitudine de LED-uri pentru notificarea alertelor, animarea acestuia și răspunsul la anumite scripturi rulate pe acesta, spre exemplu, LED-urile de pe ochi devenind verzi în cazul în care robotul memorează o față cu succes.

Localizare	Cantitate	Detalii
Cap	12	LED-uri cu 16 nivele de albastru
Urechi	2x10	LED-uri cu 16 nivele de albastru
Ochi	2x8	LED-uri pe tot spectrul RGB
Piept	1	LED-uri pe tot spectrul RGB
Picioare	2x1	LED-uri pe tot spectrul RGB

Tabela 1.2-1 LED-uri prezente pe robot

1.2.3 Alimentare

NAO este echipat cu o baterie realizată în tehnologie Litiu-Ion ce îi oferă o energie disponibilă de 48,6 Wh, fiind echivalentul a 60 de minute de utilizare activă sau a 90 de minute de utilizare în moduri pasive. Acesta poate fi utilizat și dacă este alimentat de la rețeaua de curent electric, însă acesta ne-încărcând bateria pe durata utilizării în acest mod.

1.2.4 Conectivitate

Robotul NAO dispune de două tipuri de conexiuni, prima fiind de tip Ethernet iar cealaltă prin WiFi. Robotul folosește o mufă de tipul RJ-45 compatibilă cu trei tipuri de adaptări 10/100/1000base T (viteze de 10Mbps, 100 Mbps sau 1Gbps). Aceasta se utilizează în principal pentru stabilirea conexiunii robotului prin WiFi. După conectarea cablului, robotului i se alocă o adresă IP locală, ce poate fi accesată printr-un browser, iar după autentificare se va putea conecta robotul la rețeaua wireless aceasta oferindu-i mai multă mobilitate și primind o nouă adresă IP. După crearea conexiunii, aceasta va rămâne setată iar robotul va rămâne conectat la rețeaua wireless până la restartarea routerului păstrând aceeași adresă IP menționată anterior.

Conexiunea prin WiFi este utilizată în programarea robotului cu ajutorul programelor dedicate precum Coreographe sau Python GUI fiind compatibilă cu standardul IEEE 802.11 b/g/n.

1.2.5 Audio

Nao este echipat cu un sistem de difuzoare stereo prezente în urechile sale, precum și de patru microfoane prezente pe cap, ce pot capta sunete cu o frecvență între 150 Hz și 12 kHz.

1.2.6 Camere Video

Alte două dispozitive ce îi conferă robotului autonomie și o mai bună înțelegere a mediului înconjurător sunt cele două camere video amplasate pe bărbie și pe frunte, acestea având o rezoluție de 1280x960 pixeli și o rată de achiziție de până la 30 de cadre pe secundă, fiind folosite în aplicații precum găsirea de obiecte într-o cameră sau identificarea facială.

1.2.7 Dispozitive Infra-Roșii si sonare

În ceea ce privește dispozitivele infra-roșii, ele sunt prezente în ochii robotului, prezentând un emițător și un receptor, funcționând pe o lungime de undă de 940 nm, fiind utilizate în aplicații de comunicații inter-roboți, comunicarea fiind asigurată astfel. Sonarele ajută robotul să detecteze obstacole într-un domeniu de detecție între 0.25 m și 2.55 m, funcționând la o frecvență de 40 kHz. Dacă distanța până la un obiect este sub 0.25 m, robotul nu poate returna niciun fel de informație legată de distanță, știind doar de existența obiectului.

1.2.8 Girometrul si Accelerometrul

Aceste două dispozitive constituie unitatea inercială a robotului, fiind necesari menținerii echilibrului în mers sau în diverse acțiuni efectuate de robot. Girometrul măsoară orientarea robotului pe o anumită direcție, iar accelerometrul măsoară forța de apăsare

1.2.9 Senzori de presiune, pozitie si contact

Robotul vine echipat cu senzori de presiune prezenți pe talpa robotului, aceștia fiind constituiți din rezistențe sensibile la forță ce pot returna valori între 0 și 11 kilograme cu exactitate. Senzorii de poziție se bazează pe efectul Hall și sunt denumiți generic ERM, encoder de rotație magnetică. Mișcările efectuate se sunt resimțite prin străpungerea unui câmp magnetic, ce determină o diferență de potențial prelucrată ulterior astfel încât să se obțină informații despre localizarea în spațiu. Senzorii de contact și tactili descriu dacă cineva sau ceva atinge robotul. Aceștia pot fi enumerați astfel: 3 senzori tactili pe cap, un senzor tactil pe piept, doi senzori tactili pe mâini și doi senzori pe picioare pe post de protecție la contactul cu obiecte străine.

1.2.10 Unitățile de comandă si control

Nao vine echipat cu două unități de comandă și control, una localizată la nivelul capului și cea de-a doua situată la nivelul toracelui. Primul controller este un processor Intel Z530 ce deține o memorie cache de 512 kb, cu o frecvență de ceas de 1.6GHz. Setul de instrucțiuni este pe 32 de biți iar procesorul este realizat în tehnologie de 45nm. Este un procesor cu un singur nucleu ce are încorporat la nivelul său tehnologia Intel Hyper-Threading. Este un procesor destinat în principal telefoanelor mobile, întrucât consumul de putere este de 2 W, iar acesta suportă virtualizarea aplicațiilor. De asemenea, acesta este bazat pe microarhitectură Bonnell, fiind capabil să execute două instrucțiuni pe ciclu de ceas. Acesta face translația operațiilor complexe de tip CISC în operațiuni simple, de tip RISC. În figura următoare se prezintă schema bloc funcțională a procesorului, conform cu foaia de catalog.[7]

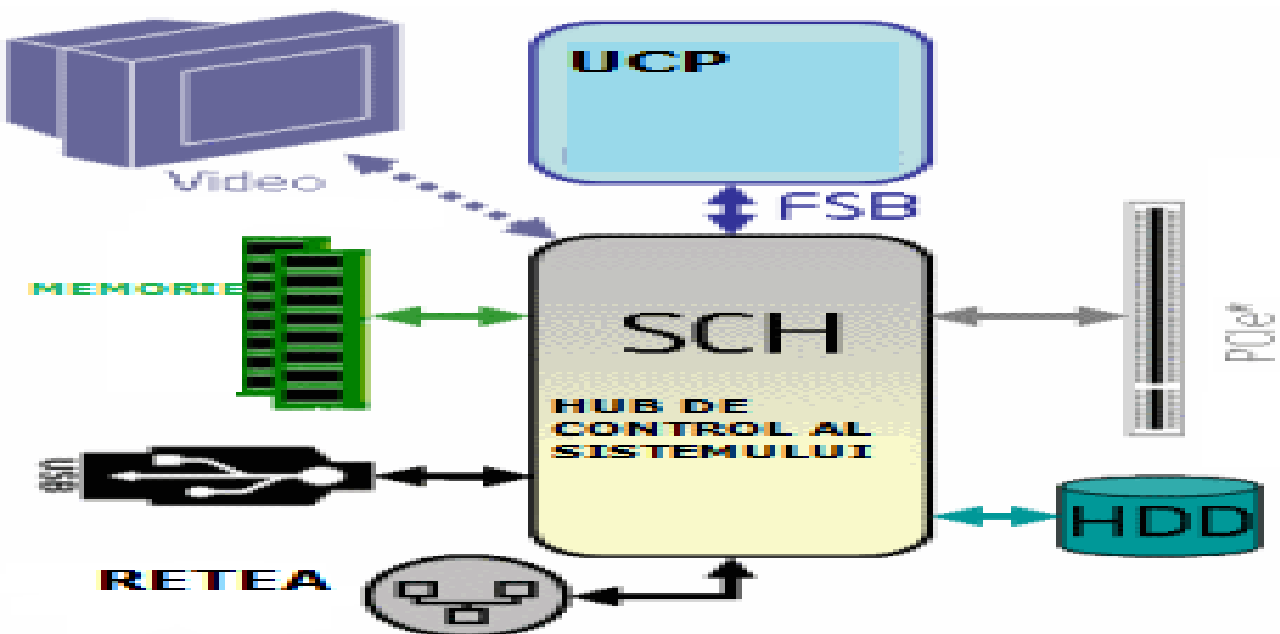


Figura [4] 1.3 Schema bloc funcțională a procesorului Intel Atom Z350

A doua unitate de comandă și control este constituită dintr-un microprocesor de tip ARM7TDMI ce controlează actuatorii, distribuind informația de control tuturor microcontrolerelor de la nivelul actuatorilor, fiind un microcontroller cu setul de instrucțiuni pe 32 de biți, de tipul RISC,

oferind performanțe ridicate și un consum mic de putere. Microcontrolerele de la nivelul actuatorilor sunt de tipul Microchip 16-biți dsPIC, comunicația între acestea și unitatea de control menționată anterior făcându-se prin intermediul a două magistrale de tipul RS485 cu o viteză de 460 Kbps. În schema de mai jos se prezintă arhitectura hardware din punct de vedere electronic a robotului NAO.

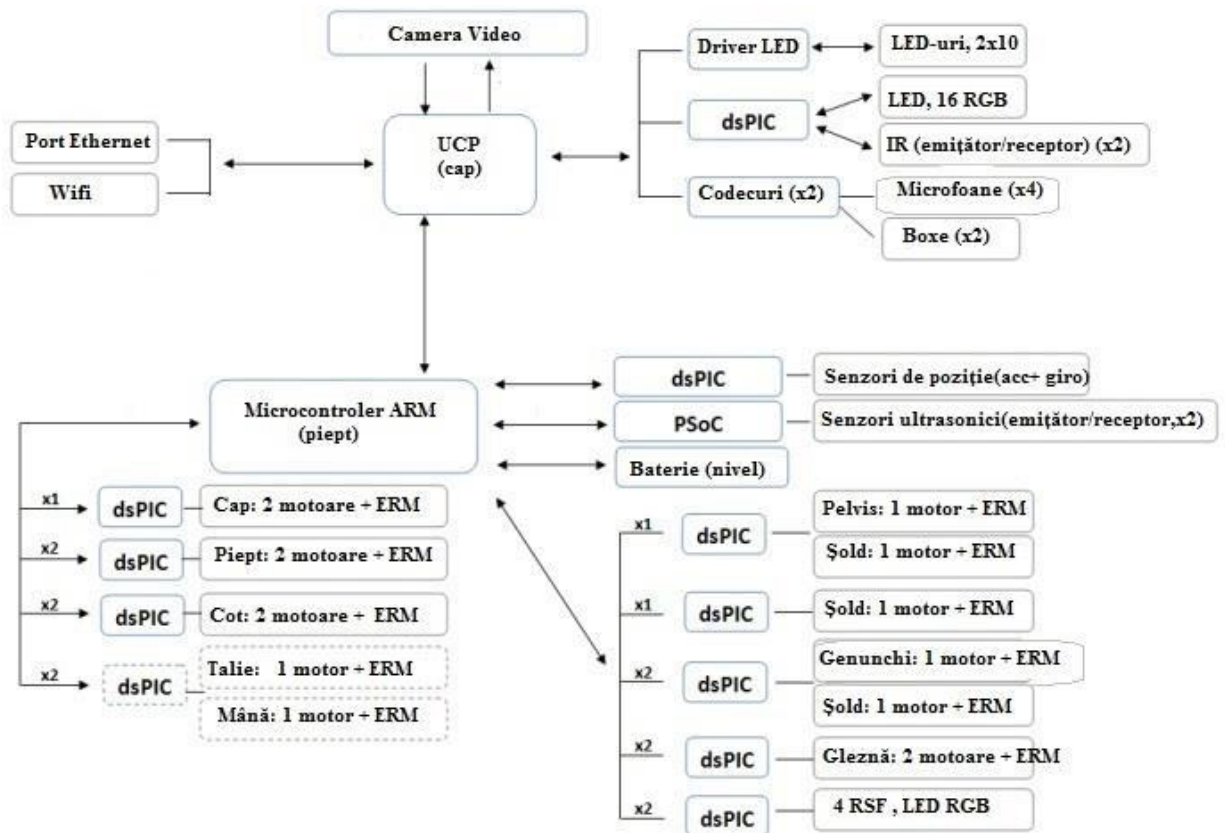


Figura [5] Figura 1.4 Arhitectura electronică a robotului NAO

1.2.11 Actuatore si grade de libertate

Actuatoarele sunt motoarele ce asigură robotului libertatea de mișcare. Robotul vine echipat cu 25 astfel de motoare, acestea fiind poziționate în încheieturi pentru a simula cât mai bine mișcările corpului uman, având totuși o structură diferită de mușchi și articulații. Robotul NAO deține motoare cu perii de carbon întrucât acestea au un cost redus și viteză simplă de controlat. Motoarele de acest tip generează cuplu direct din sursa de alimentare folosind comutație internă, magneți staționari și electromagneți rotativi.

În funcție de viteză de rotație și cuplul dezvoltat când motorul este blocat, se împart în trei categorii:

1. 8300 rotații pe minut și 68 miliNewtoni metru
2. 8400 rotații pe minut și 9.4 miliNewtoni metru
3. 10700 rotații pe minut și 14,3 miliNewtoni metru

Motoarele de tipul 1 sunt puternice, fiind folosite la picioare, celelalte două tipuri fiind utilizate în mâini și articulațiile degetelor și a gâtului, pentru mișcări mai precise și mai lente. În figura următoare se prezintă motoarele robotului NAO conform cu numele din documentația oficială.

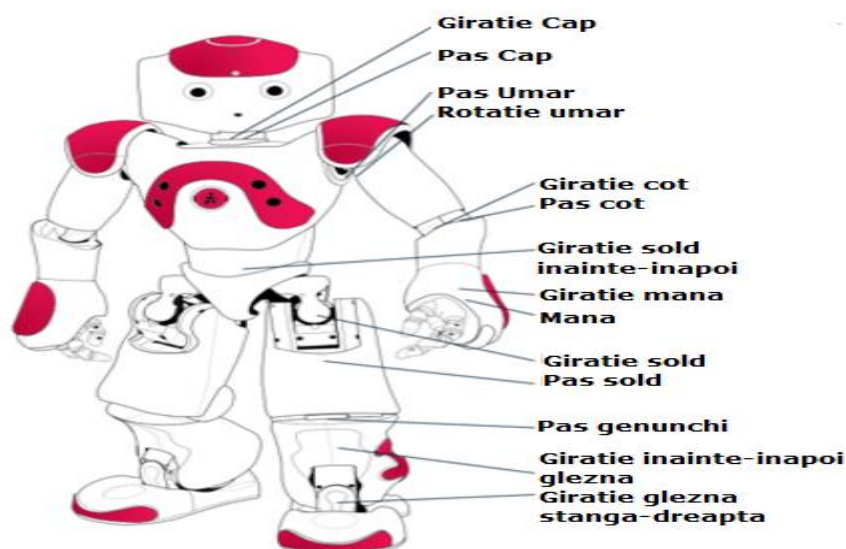


Figura [6] 1.5 Motoarele Robotului

Gradele de libertate exprimă fiecare dintre mărimile scalare independente necesare pentru determinarea stării unui sistem. Numărul minim de parametri prin care se poate preciza starea sistemului reprezintă numărul de grade de libertate al acestuia. Spre exemplu, pentru determinarea poziției unui punct material, este nevoie de trei coordonate carteziane: x , y și z . Fiecare legătură scade numărul gradelor de libertate cu o unitate.

Nao are 25 de grade de libertate, 11 pentru partea inferioară, ce includ pelvisul și picioarele și 14 pentru partea superioară a trunchiului, inclusiv capul. În tabelul următor se prezintă componentele robotului ce dețin grade de libertate și numărul acestora.

Componentă	Grade de libertate
Cap	2
Braț (x2)	5
Pelvis	1
Picior (x2)	5
Mână (x2)	1

Tabela 1.2-2 Gradele de libertate ale robotului NAO

1.3 CARACTERISTICI SOFTWARE

Sistemul de operare se numește OpenNAO, ce rulează pe un nucleu de Linux (Gentoo) și face parte din arhitectura software NAOqi, cea care dă viață robotului. Acesta este o versiune integrată, dezvoltată special pentru roboții concepuți de firma Aldebaran. Oferă un număr mare de librării și programe, fiind un sistem modular capabil să lucreze cu un număr mare de fișiere executabile, având o arhitectură bazată pe evenimente. Acesta reușește să anime robotul și dă utilizatorului posibilitatea de a crea diferite aplicații fără ca acesta să dețină un număr mare de cunoștințe în domeniul programării.

Dintre avantajele arhitecturii, demne de menționat sunt următoarele:

- utilizatorul are posibilitatea de a rula comportamente ale robotului de la distanță sau local;
- funcționalitățile robotului pot fi testate independent, spre exemplu: vorbirea, mersul, vederea, etc;
- dezvoltarea aplicațiilor în acest mediu este mai simplă întrucât codul poate fi compilat pe platforme diferite;
- sistemul permite utilizatorului să aibă acces la toate variabilele sale în orice moment al execuției.

Funcționalitățile arhitecturii NAOqi sunt următoarele:

- Aplicațiile pot fi scrise în diferite limbaje de programare, C++; Python sau Java.
- Procesele pot fi manageriate ușor.
- Memoria partajată este simplu de manageriat.
- Managementul intefetelor pentru programarea aplicațiilor (cunoscute ca API).
- Execuția metodelor se poate face paralel sau secvențial, în funcție de nevoile programatorului.
- Suportă mai multe platforme precum: Linux, Windows sau MAC OS.

Din punct de vedere structural, arhitectura NAOqi poate fi divizată în trei părți:

- Sistemul de operare NAOqi sau OpenNAO, menționat anterior
- Bibliotecile NAOqi ce conțin diferite acțiuni pe care robotul le poate face. Avantajul acestora fiind dat de nivelul de abstractizare, utilizatorul putând programa robotul fără a fi nevoie să fie în apropierea acestuia.
- Managementul de comunicații al dispozitivelor (DCM-Device Communication Manager)- este un modul ce are rolul de a controla comunicația cu dispozitivele electronice din robot cu excepția senzorilor de tip audio și video. Acesta face legătura între nivelele superioare ale arhitecturii și cele inferioare, constituite din module electronice dedicate (senzori, motoare). Acesta preia informații de la dispozitivele electronice prin intermediul unității de prelucrare de la nivelul toracelui. [8]

În figura următoare se prezintă funcționarea modulului DCM între nivelele software:

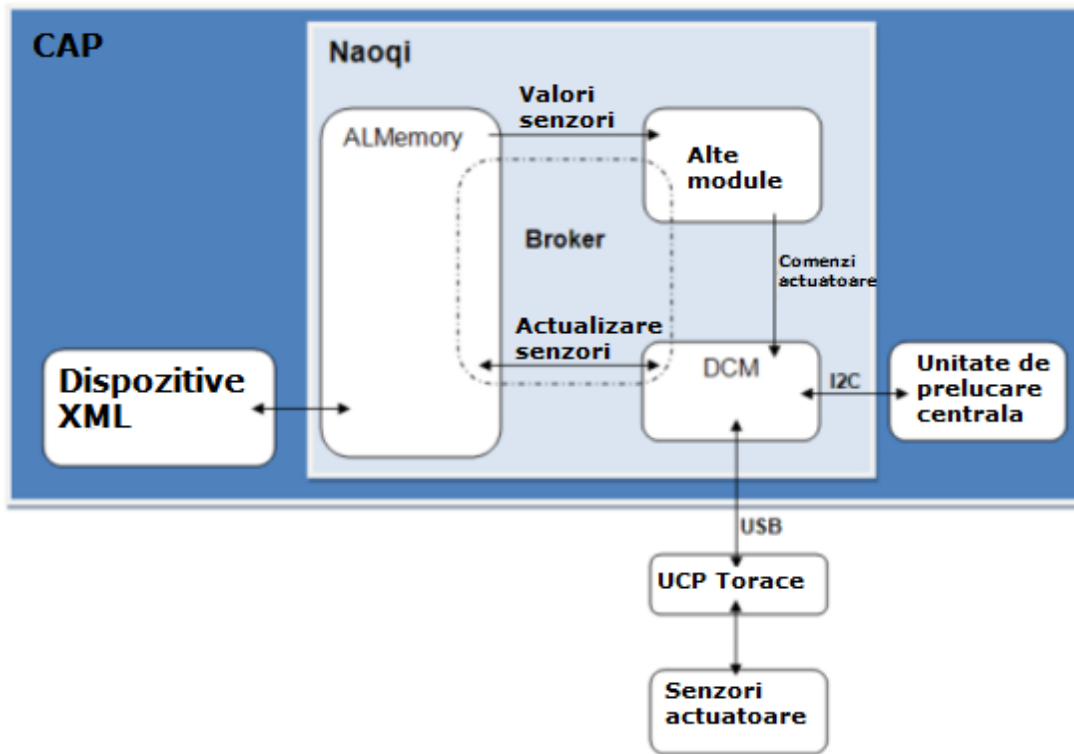


Figura [7] 1.6 Legatura intre nivelul software inferior si cel superior prin DCM

Arhitectura NAOqi este una orientată pe obiecte, având trei tipuri principale de obiecte:

Broker (Agent): este un obiect ce asigură căutarea modulelor sau metodelor și acces la rețea, dând voie modulelor să acceseze metode din afara modulului care rulează. Un broker poate fi copil pentru un altul, formând astfel un arbore distribuit;

Modul: conține metode definite de utilizator;

Proxy: este un obiect ce se comportă ca modulul pe care îl reprezintă, fiind o copie a acestuia, de exemplu, crearea unui proxy către modulul ALMemory va crea o copie fidelă a acestuia pentru a fi accesată de utilizator.

La crearea unui modul, dezvoltatorul instanțiază un broker de care leagă apoi modulul său. Imediat ce proxy-ul este pregătit, metodele din interiorul acestuia pot fi apelate, indiferent dacă modulul este local sau la distanță sau scris în Python, C++ sau Java. Nucleul NAOqi este compus din trei module:

- *ALMemory:* Reprezintă o memorie centralizată folosită pentru stocarea informației legată de partea hardware a robotului. Aceasta reține în interiorul ei informații despre senzori și actuatori și poate fi folosită pentru valori specifice, având o structură de hub de stocare. Prin intermediul acesteia, utilizatorul poate accesa senzorii, spre exemplu: distanță față de obiectele din stânga și dreapta robotului prin intermediul sonarului.[9]

- *ALLogger*: Acest modul permite utilizatorului să vizualizeze erori, avertismente și informații pe un dispozitiv conectat la robot (de tip PC, laptop, tabletă, etc). Acest modul permite utilizatorului să decidă ce erori poate ignora, spre exemplu, o supraîncălzire temporară a motoarelor.[10]
- *ALPreferencesManager*: Acest modul este utilizat pentru stocarea setărilor și preferințelor utilizatorului, având o copie în Cloud. Printre utilitățile acestui modul se număra și comportamentul de început al robotului și instantierea fișierelor de tip XML. [11]

Conectarea la robot în vederea dezvoltării aplicațiilor se poate face prin două metode: utilizarea Coreographe prin intermediul protocolului de comunicație SSH, fie printr-un program ce simulează un terminal-client pentru SSH.

SSH este un protocol care permite crearea unei sesiuni de lucru la distanță, transferul de fișiere și crearea unor canale de comunicație pentru alte aplicații, toată transmisia fiind sigură împotriva atacurilor intrușilor. Securitatea și confidențialitatea transmisiei este asigurată prin criptare. Integritatea se bazează pe trimiterea unor sume de control criptografice. Autentificarea serverului se face prin criptografie asimetrică, serverul având o cheie secretă, iar clienții dispun de cheia publică corespunzătoare. Autentificarea clientului se face fie prin criptografie asimetrică, ca și în cazul autentificării serverului (dar bineînțeles folosind altă pereche de chei), fie cu parola clasică, dată de client după autentificarea serverului [12]. În cazul acestei lucrări, clientul care inițiază o conexiune prin SSH este calculatorul utilizatorului, iar serverul este robotul NAO. Pentru a suporta asemenea conexiune, robotul trebuie să aibă instalat pe sistemul său de operare un server SSH. Orice sistem de operare bazat pe Linux are implicit instalat un asemenea server.

CAPITOLUL II

Medii de programare și dezvoltare software utilizate

În dezvoltarea acestei lucrări au fost folosite două instrumente de dezvoltare, Choreographe, oferit de firma Aldebaran și IDLE (Python GUI), o interfață dezvoltată de Python Software Foundation.

2.1 Choreographe

Choreographe este mediul de lucru ales pentru dezvoltarea comportamentului robotului, fiind o platformă de tipul cross, ce permite integrarea și utilizarea unei multitudini de limbaje de programare, precum C++, Java, Python, cel din urmă fiind cel ales pentru dezvoltarea aplicației. Choreographe este o aplicație desktop ce permite crearea de comportamente, animații și ferestre de dialog. Programele pot fi testate înainte de instalarea lor pe robot cu ajutorul unui robot virtual, pentru a nu afecta integritatea fizică a robotului NAO.

Aplicația permite monitorizarea și controlul complet al robotului prin multitudinea de setări pe care le pune la dispoziție. Există funcții predefinite în cadrul programului, fiind grupate în funcție de tipul acestora, așa cum se poate observa în figura următoare.

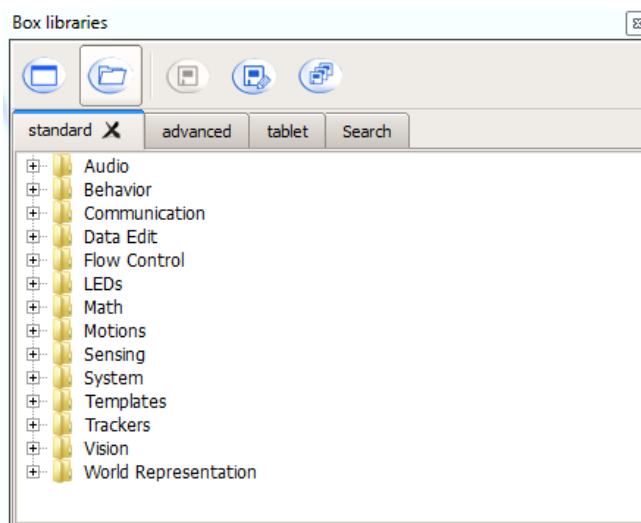


Figura [8] 2.1 Librăriile predefinite ale robotului NAO

Limbajul de programare utilizat în descrierea funcțiilor din Coreographe este Python. Acest limbaj de programare a fost creat în 1989 de programatorul olandez Guido van Rossum. Python pune accentul pe curățenia și simplitatea codului fiind un limbaj orientat pe obiecte, ce permite și programarea imperativă, funcțională sau procedurală. Sintaxa acestuia este una simplă, mai clară și mai concisă decât în alte limbaje de programare precum C++. Principalul avantaj al limbajului este faptul

că acesta reprezintă o platformă de dezvoltare pentru numeroase aplicații, utilizatorului fiindu-i oferite o multitudine de biblioteci standard.

Folosirea metodelor predefinite se poate face prin fereastra de lucru iar acestea sunt reprezentate printr-o multitudine de simboluri. La pornire se trimite un semnal de la butonul “play” iar acesta activează modulele conectate la acesta, pornind, spre exemplu, un modul denumit *Stand Up* (Stai în picioare), exemplificat în figură.

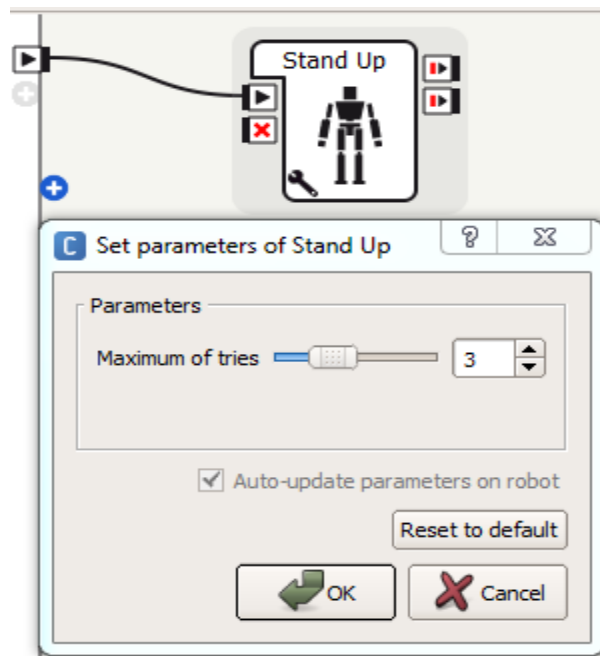


Figura [9] Folosirea unei comenzi predefinite

Utilizatorul poate alege să modifice sau nu parametrii predefiniți ai funcției, în acest caz fiind vorba de numărul de încercări de ridicare ale robotului. Funcția poate fi vizualizată și ca un cod Python, prin accesarea utilizarea editorului de scripturi. Un script este un program software de care se ocupă un alt program, altul decât procesorul calculatorului (precum în cazul unui program compilat). În biblioteca standard Choregraphe, utilizatorul poate crea module folosind modulul denumit Python Script ce conține structură standard pentru crearea unui script. Acesta are următoarea alcătuire:

```
MyClass(GeneratedClass):  
  
    def __init__(self):  
        GeneratedClass.__init__(self)  
  
    def onLoad(self):  
        #put initialization code here  
        pass  
  
    def onUnload(self):  
        #put clean-up code here  
        pass
```

```

def onInput_onStart(self):
    #self.onStopped() #activate the output of the box
    pass

def onInput_onStop(self):
    self.onUnload() #it is recommended to reuse the clean-up as the box is
stopped
    self.onStopped() #activate the output of the box

```

Clasa *MyClass* reprezintă modulul creat. Aceasta moștenește clasă *GeneratedClass*, o clasă ce se generează automat la execuția unui modul. Aceasta din urmă include toată informația necesară execuției unui modul (intrări, parametri, ieșiri). În codul funcției utilizatorul poate adăuga alte funcții în clasă, folosind instrucțiunea *def*.

Metodele din modul trebuie definite folosind sintaxa de mai sus, acesta fiind un cod minimal ce trebuie implementat. Metodele sunt apelate pe rând, *onLoad* este apelată la încărcarea modulului pe robot. Metoda *onUnload* este apelată la descărcarea unui modul de pe robot, fiind recomandat să se apeleze în metodă *onStop* astfel încât modulul să se reinitializeze la oprire. Intrările în modulele *Choreographe* sunt de mai multe tipuri:

- *onStart*, când intrarea e stimulată, modulul începe;
- *onStop*, când intrarea e stimulată, modulul se oprește;
- *onEvent*, nu are niciun effect asupra modulului cu excepția momentului când intrarea este stimulată, activându-se metodă *onInput*;
- *onLoad*, stimulată când rularea programului ajunge la nivelul respectiv;
- *AlMemory Input*, stimulată atunci când valoarea unei variabile, spre exemplu *TactileRightArm*, ce este activată de atingerea senzorului tactil de la mână dreaptă se modifică.

Ieșirile unui modul *Choreographe* sunt de două tipuri:

- *onStopped*, când modulul este oprit, această ieșire este stimulată;
- *punctual*, având rolul de a transmite parametri mai departe către un alt modul, fără a afecta în vreun fel comportamentul modulului.

Intrările și ieșirile pot fi de mai multe tipuri, având corespondenți în limbajele de programare astfel:

- *Bang* este echivalentul unei variabile de tip Boolean în limbajul C++;
- *Number* este echivalentul unei variabile de tip Float sau a unui vector de acest tip în limbajul C++;
- *String* este corespondentul unui șir de caracter sau a unui vector de șiruri, de tipul *char*
- *Dynamic* poate reprezenta un eveniment simplu, precum *Bang* sau unul ce conține date, de tip *Number* sau *String*.

Choreographe oferă o multitudine de alte modalități de supraveghere a robotului și de debugging utile programatorului. În figura următoare se prezintă utilitarul Robot View ce permite vizualizarea permanentă a stării robotului pe calculatorul la care este conectat. În figură se prezintă un cadru din execuția modului Tai Chi Chuan dezvoltat de Aldebaran ca demonstrație a echilibrului de care NAO este capabil.

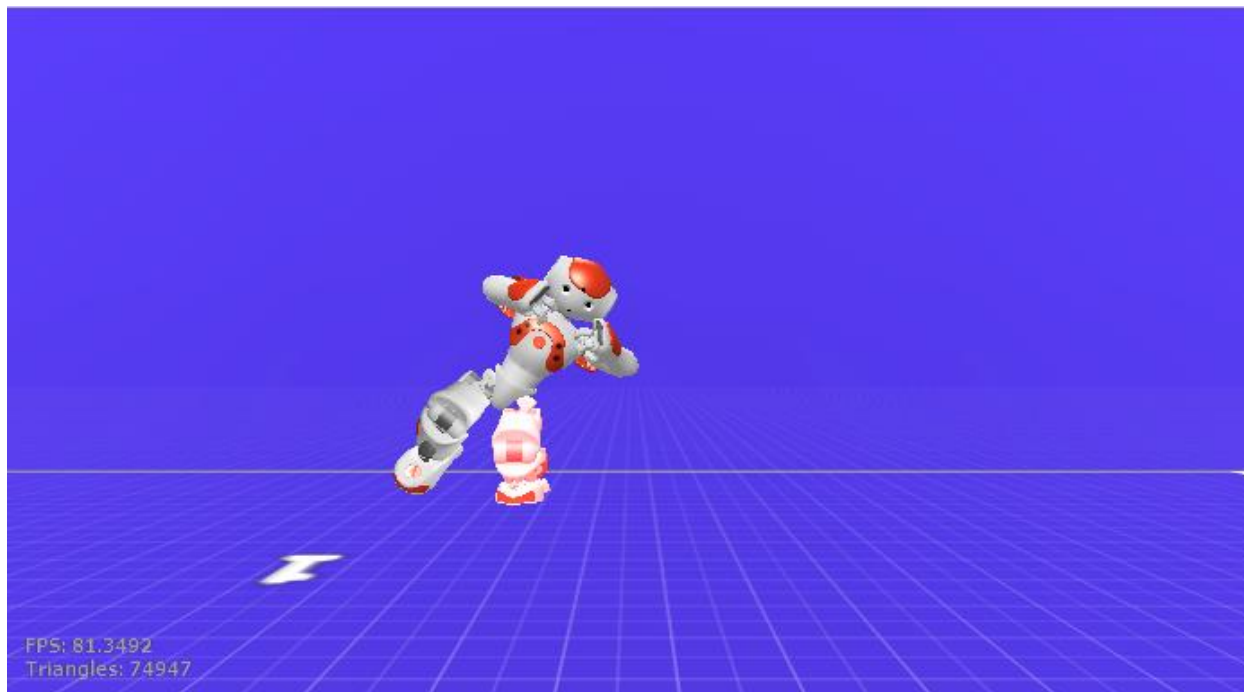


Figura [10] 2.3 NAO executând Tai Chi Chuan vizualizat prin Robot Pose

Robotul poate avea o multitudine de comportamente predefinite și încărcate pe acesta, pentru a-i ușura munca programatorului și a nu fi nevoie de reîncărcarea unor programe pe robot. Această parte a memoriei se poate accesa cu ajutorul utilitarului Robot Applications ce face ușoară programarea la repornire.

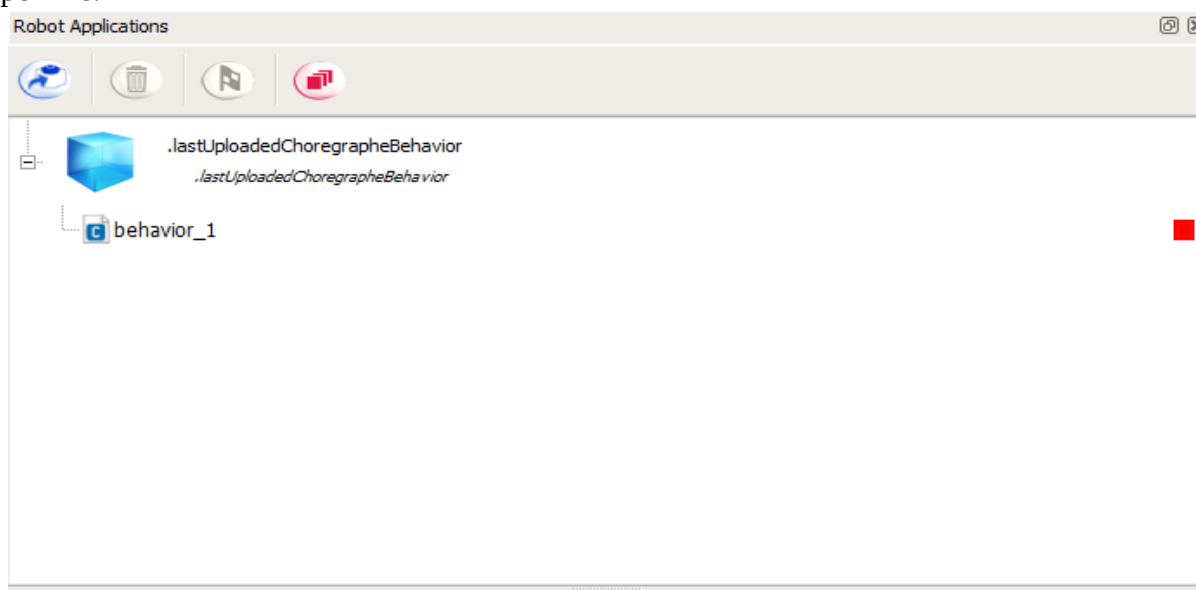


Figura [11] 2.4 Utilitarul Robot Applications

De departe cel mai utilă aplicație integrată în Choreographe este Monitor, ce permite utilizatorului să vizualizeze imaginile capturate de robot în timp real pe ecranul calculatorului la o rată de achiziție de 30 de cadre pe secundă și o rezoluție 1920x1080 pixeli. Astfel, programarea robotului de la distanță devine simplă, întrucât programatorul știe la orice moment de timp starea în care robotul se află.

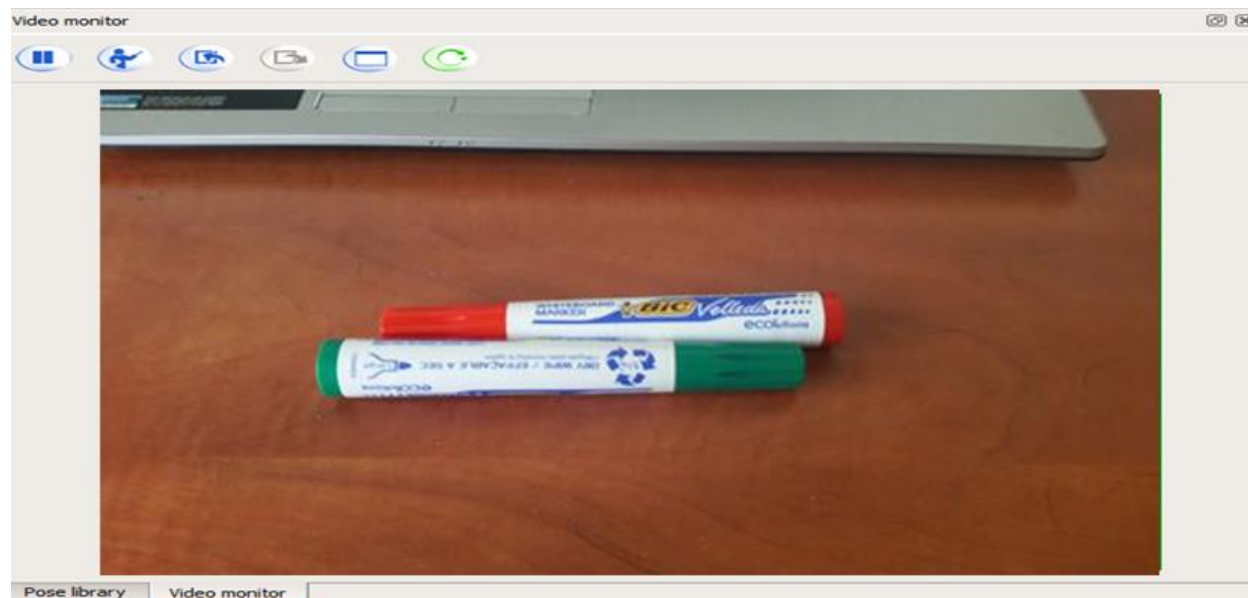


Figura [12] Aplicația Video Monitor implementată în Choreographe

De asemenea, Monitor permite și vizualizarea variabilelor din memorie, mai exact din modulul *ALMemory*, separat, pentru un control mai bun asupra buclelor de decizie în timp real. Spre exemplu, se poate vizualiza variația distanței față de peretele din stânga accesând variabila de stare denumită *Value2* din directorul aparținător *US/Left/Sensor*, aceasta stocând valoarea până la cel mai apropiat obstacol obținută printr-un procedeu de detecție cu unde radio.

2.2 Programarea Orientată pe Obiecte

Programarea orientată pe obiecte reprezintă o paradigmă de programare, ce se axează pe ideea încapsulării, a grupării datelor și a codului ce rulează într-o singură structură. Un alt concept ce merită menționat este polimorfismul care permite abstractizarea pentru o descriere conceptuală mai simplă a soluției implementate de programator. Aceasta a apărut din necesitatea exprimării instrucțiunilor și a structurilor utilizate într-un mod mai natural și accesibil omului. Astfel se realizează o corelare mai bună între modul în care oamenii gândesc programele și procesoarele le interpretează. Înaintea apariției programarea se făcea în limbaje de programare procedurale, precum C sau Pascal sau în limbaje ce nu ofereau posibilitatea implementării funcțiilor, spre exemplu, limbajul Assembler.[13]

Principală problemă a limbajelor procedurale este separarea datelor de funcții ceea ce duce la o întreținere greoaie a programelor. Astfel, s-a pus problema ca datele și subrutinele să fie grupate într-un

anume mod, astfel încât subrutinele să știe ce date prelucrează și să formeze un modul, pentru reutilizarea codului. În ciuda faptului că paradigma este denumită “orientată pe obiecte”, conceptul care stă la baza ei este clasa, aceasta fiind o descriere generală a unei multitudini de obiecte ce au caracteristici comune, spre exemplu, clasă Electrocasnic are în componența sa toate entitățile pe care noi le definim ca fiind electrocasnice, programatorul lucrând cu o instanță a clasei, televizor, mașină de spălat, cuptor cu microunde, etc. Trecerea de la o structură generală, clasa, la un obiect, se face prin atribuirea unor proprietăți clasei până la momentul în care aceasta devine particularizată complet.

Diferența majoră dintre un program scris într-un limbaj procedural, ce conține o serie de date și funcții, și unul scris într-un limbaj orientat pe obiecte, este faptul că al doilea reprezintă o multitudine de obiecte ce comunică între ele, ceea ce duce la abordare diferită a rezolvării problemelor apărute și a implementării programelor. În exemplul următor, sunt explicate cele două clase folosite în lucrarea curentă.

```
class NaoXO():
    """
    Class used for playing the game with the robot
    """
class ImgProcessingXO():
    """
    ###Nao uses this class to do the image processing
    """
```

Figura [13] 2.6 Clasele utilizate în aplicația curentă

Cele două clase sunt principalele instrumente folosite în lucrarea de față, prima dintre ele fiind folosită pentru funcțiile de mișcare, iar cea de-a doua pentru funcțiile de procesare de imagine. Aceste două clase au în componența lor o multitudine de metode ce permit robotului să proceseze imagini efectueze mișcările necesare. Aceste clase comunica între ele prin intermediul funcțiilor (metodelor) implementate în interiorul acestora. O clasă reprezintă o generalizare a unui obiect. Obiectul este constituit din date și comportamente asociate. Metodele sunt definite într-o clasă iar obiectele reprezintă instanțe ale claselor și reprezintă modul prin care proprietățile obiectelor să fie modificate. Definirea unei metode se face într-un program Python folosind instrucțiunea **def** urmată de numele parametrilor funcției. Codul fiecărei funcții pornește pe următoarea linie, este indentat cu un spațiu după declararea funcției. Acestea se apelează în interiorul programelor cu ajutorul unor apeluri de *rezultat = nume_clasa.nume_metoda (parametri)* limbajul Python. .

O altă proprietate ce deosebește limbajul Python de limbajul C este reprezentat de posibilitatea de a avea parametri de început în funcție, prin simpla atribuire în declararea funcției, spre exemplu: *def funcție (parametru1 = 0, parametru2)*. La folosirea acestui tip de declarare, primul parametru, dacă nu este dat de utilizator, va avea valoarea 0. Fiecare funcție poate returna, cu ajutorul instrucțiunii *return*, un set de parametri utilizatorului ce pot fi utilizați în continuare, după apelul funcției. În programarea obiect orientată există două tipuri de variabile: globale sau locale, primele pot fi accesate în tot programul, iar cele din urmă pot fi accesate doar în anumite zone ale programului, mai exact, în anumite metode din interiorul claselor.

Principalele diferențe între programarea procedurală și programarea obiect orientată, deși ambele folosite în prezent, fiecare cu avantaje și dezavantaje, putem enumera:

- În programarea procedurală, datele tind să fie distincte față de funcțiile care operează folosindu-le, în timp ce într-o paradigmă de programare de tipul obiect orientat datele tind să aibă în spate o multitudine de funcții
- Programarea obiect orientată aduce împreună datele și funcțiile făcând înțelegerea programului mai simplă pentru utilizator
- Lungimea codului într-o aplicație scrisă într-un limbaj orientat pe obiecte este semnificabil mai scurtă decât într-un limbaj procedural.
- În limbajele procedurale, scopul principal este scrierea de funcții noi, în timp ce în limbajele de tip obiect-orientat scopul este reprezentat de crearea de noi obiecte[14]

2.3 Limbajul Python

În continuare se va prezenta limbajul Python conform documentației oferite de site-ul www.tutorialspoint.com. Python reprezintă un limbaj de nivel înalt interactiv și orientat pe obiecte folosit în special pentru scrierea scripturilor definite anterior. Acest limbaj se vrea a fi unul ce facilitează citirea de către utilizator și are mai puține construcții sintactice decât alte limbaje de programare. Acesta este un limbaj ce facilitează interactivitatea, acesta putând să interacționeze cu programul doar prin scrierea de comenzi în fereastră de comandă.

A fost dezvoltat de către Guido van Rossum spre sfârșitul anilor 1980 și începutul anilor 1990 în Olanda. Acesta este derivat din multe alte limbaje precum ABC, C, C++ și Unix shell. Acesta acceptă în structura sa programarea structurală și programarea orientată pe obiecte și poate fi folosit ca limbaj pentru scrierea scripturilor. Acesta oferă o gamă largă de tipuri de date și poate fi integrat ușor ca liant între diferite limbaje de programare dintre cele menționate anterior. În această lucrare s-a folosit limbajul Python ca liant între comportamentele robotului și librăriile deja existente pe robot.

O altă caracteristică importantă a limbajului este faptul că nu acceptă în numele obiectelor caractere speciale precum @, \$, %, !, acestea fiind rezervate pentru anumite operații în interiorul programelor. Dintre convențiile importante ce merită menționate pentru limbajul Python se pot menționa următoarele: numele claselor încep întotdeauna cu literă mare, începerea unui identificador cu ajutorul caracterului „_” denotă faptul că acesta este privat iar unele variabile nu pot fi denumite cu nume predefinite precum *and*, *if*, *not*, *pass*, *print*, *global*, *class*.

Pentru a facilita citirea, blocurile subordonate unei instrucțiuni se indentază cu același număr de spații sau tab-uri. Comentariile se scriu având caracterul “#” înaintea textului, sau, în cazul în care sunt scrise pe mai multe linii, încep și se termină cu o serie de caractere de tip apostrof ```.

Variabilele în limbajul Python sunt de mai multe feluri, pentru declararea acestora fiind necesară doar atribuirea unei valori, fără folosirea unei instrucțiuni de definire a tipului variabilei, precum în limbajul C, spre exemplu. Tipurile de variabile acceptate de limbajul de programare sunt: număr întreg,

număr întreg lung, număr real, număr complex, șiruri de caractere, organizate în liste, tuple, dicționare și matrici. La fel ca limbajul C, acesta conține o multitudine de bucle de decizie și de bucle repetitive, precum *while*, *for*, *if*, *else*. În ceea ce privește organizarea funcțiilor, acestea sunt porțiuni de cod menite să fie accesate repetitiv, ce sunt folosite pentru a efectua una sau mai multe operații aducând modularitate codului și ajutând la reutilizarea codului.

Limbajul poate fi extins cu ajutorul unor librării predefinite, precum openCV, numpy, naoqi, ce vor fi descrise în următoarele pagini ale lucrării. Avantajul principal al mediului de programare este că permite integrarea simplă a librăriilor, prin utilizarea instrucțiunii *import*, extindere duce la o viteză de creare a aplicațiilor ce nu cer viteze de procesare ridicate.

Se recomandă modularizarea codului pentru o mai bună efectuare a operațiunilor de îmbunătățire sau curățare de erori a codului. Simpla instrucțiune *import* duce la aducerea tuturor funcțiilor sau claselor prezente într-un modul în modulul în care se lucrează, acestea fiind accesibile celui din urmă doar prin scrierea numelui funcției sau prin instantierea unui obiect al unei clase. Se recomandă ca modulul ce se utilizează ca modul importat să fie în același folder cu modulul care importă pentru o mai simplă curățare a codului în cazul erorilor.

```
import time
import speech_reco
import vision_definitions
from determining_the_winner import determining_the_winner
from speech_reco import *
from tic_tac_toe_strategy import strategy_x
from tic_tac_toe_strategy import strategy_o
```

În secvența anterioară au fost prezentate cele trei tipuri de instrucțiuni de aducere a unui modul în modulul curent pentru folosirea metodelor și claselor în modulul de mișcare al aplicației curente.

Mediul de programare utilizat a fost IDLE (Python GUI), oferit de dezvoltatorii limbajului de programare ce se vrea să fie o unealtă de dezvoltare a programelor în limbajul Python ce facilitează citirea programelor fiind un editor de text. O dovadă a versatilității limbajului de programare este faptul că mediul de programare este dezvoltat în Python având posibilitatea de integrare a punctelor de întrerupere în rularea programelor și o vizualizare bună a variabilelor și a metodelor prin simpla utilizare a funcției *print*. Acesta ajută programatorul în folosirea spațiilor pentru delimitarea blocurilor subordonate instrucțiunilor prin punerea automată a spațiilor pe următorul rând după instrucțiuni de tip *if*, *else*, *for*, și revenirea la alinierea anterioară după instrucțiuni de tip *return*.

2.4 NumPy

NumPy reprezintă abrevierea de la Numerical Python, pitonul numeric, și este un set de extensii a limbajului Python ce permite programatorului să facă o multitudine de operații cu seturi mari de date

cu o viteză îmbunătățită. În continuare se va descrie rațiunea utilizării aceste extensii conform cu documentația oficială. Seturile de date cu care NumPy lucrează sunt mulțimi de numere, organizate vectorial, matriceal sau ca vectori n-dimensionali, totuși acestea diferind de definițiile algebrice ale vectorilor n-dimensionali.

Rațiunea utilizării acestei extensii este reprezentată de problema utilizării unor seturi mari de date în limbajul Python, ce, în mod normal ocupă mult spațiu și pentru procesarea acestuia timpii ar putea ajunge la ordinul sutelor de zile. În general, operațiile care se fac cu seturi mari de date pot fi împărțite în operații standard, acesta fiind secretul extensiei. Ea reprezintă aplicarea experiențelor utilizatorilor la nivel de optimizare ducând la folosirea ei într-o varietate de contexte. A fost dezvoltată în limbaje de tip MATLAB, FORTRAN, S și S++. NumPy este construit, în principal pentru procesarea imaginilor, conținând algoritmi ce duc la o viteză mai mare de procesare a acestora.

Compunerea extensiei se face cu o serie de submodule dintre care cele mai importante sunt *Numeric.py*, scrisă în limbajul Python, ce definește o serie de proceduri pentru manipularea datelor și punerea lor în mulțimi, *LinearAlgebra.py*, ce deține o multitudine de operații de algebra liniară optimizate. Obiectele cu care extensia lucrează sunt reprezentate de mulțimi ce reprezintă colecții de dimensiuni mari și omogene de numere.

Principala funcție folosită în procesarea de imagine în lucrarea curentă este reprezentată de funcția *asarray(parametru1,parametru2,parametru3)* ce întoarce utilizatorului o multime de numere cu care operează, parametru1 fiind reprezentat de un vector de numere ce construiește o matrice. [15]

2.5 OpenCV

OpenCV, *Open Source Computer Vision Library*, reprezintă o librărie ce are în componența sa sute de algoritmi de viziune optimizați pentru computere și procesoare. În continuare se va prezenta funcționarea acestei librării conform documentației oficiale oferită de dezvoltator. Are o structură modulară, folosind NumPy ca extensie pentru procesare rapidă, iar principalele module sunt:

- *Core*, un modul compact ce definește structuri de date și funcții de operații cu mulțimi de numere mari;
- *Imgproc*, folosit pentru filtrarea imaginilor, transformarea acestora, redimensionare, împărțire, conversie de spațiu de culoare, histograma;
- *Video*, constituit de o multitudine de algoritmi de estimare a mișcărilor, urmărire de obiecte sau detectare de fundal;
- *Calib3d*, ce are în componența sa algoritmi de calibrare pentru camera video, poziții ale obiectelor și elemente pentru reconstrucția tri-dimensională a obiectelor;
- *Features2d*, conține funcții pentru detecția trăsăturilor unui obiect într-un plan bidimensional;
- *Objdetect*, ce are în componența sa funcții de identificare a unor obiecte predefinite precum fețe, mașini, ochi, copaci;

- *Highgui*, o interfață pentru captură de imagini și operații simple cu acestea;
- *Gpu*, algoritmi optimizați pentru diferitele module constitutive ale librăriei.

Extensia se ocupă integral de gestionarea memoriei și conține o serie de destructori pentru curățarea zonelor tampon pentru procesare. La inițializarea unei matrici sau copierea acesteia, se va modifica doar variabila ce indică utilizarea acelei zone de memorie. Un alt avantaj al acestei librării este faptul că ea permite utilizarea concomitentă de către două fire de execuție a aceleași funcții ducând la o viteză mai mare de lucru.

Semnarea erorilor se folosește cu ajutorul excepțiilor. Spre exemplu, când un algoritm de optimizare nu a ajuns la convergență, funcțiile returnează o variabilă de tip boolean. Excepțiile pot fi instanțe ale clasei `cv::Exception` sau derivate ale sale. [16]

Această librărie dorește să fie o platformă comună în dezvoltarea aplicațiilor de viziune pentru aplicații pe calculator, gratis, încurajând utilizatorii să aducă îmbunătățiri algoritmilor deja existenți. În realizarea acestei lucrări au fost folosite tipuri diferite de algoritmi de viziune și detecție ce vor fi explicați pe larg în capitolul următor.

2.6 NAOqi

NAOqi reprezintă programul principal ce controlează robotul și rulează pe acesta. Este folosit pentru programarea tuturor roboților produși de firma Aldebaran, satisfăcând nevoi precum paralelismul, sincronizare, reacții la evenimente și mișcări complexe, având avantajul de a oferi o metodă de comunicare între diferite module (mișcare, audio, video, posturi), fiind capabil de a crea aplicații distribuite ce pot fi încărcate simultan pe mai mulți roboți. Acesta poate fi folosit pe toate sistemele de operare importante, Windows, Linux, MacOS folosind limbaje de programare precum Python, C++ sau Java. O dovadă a versatilității platformei este faptul că aceasta deține nume identice de funcții integrate pentru toate limbajele menționate anterior.

Principalul avantaj al acestei platforme este faptul că, în cazul apelării unei funcții, robotul și calculatorul de pe care se încarcă secvențele de cod, pot ști dacă funcția a fost executată, pot pune pauze în desfășurarea acesteia sau pot vedea în timp real modul în care funcția se desfășoară. Executabilul NAOqi ce rulează pe robot este un agent, acesta încărcând un fișier `autoload.ini` ce definește ce librării trebuie încărcate pe robot. Fiecare librărie conține una sau mai multe metode ce folosesc brokerul pentru apelarea lor. În figura următoare se explică modularitatea robotului NAO.

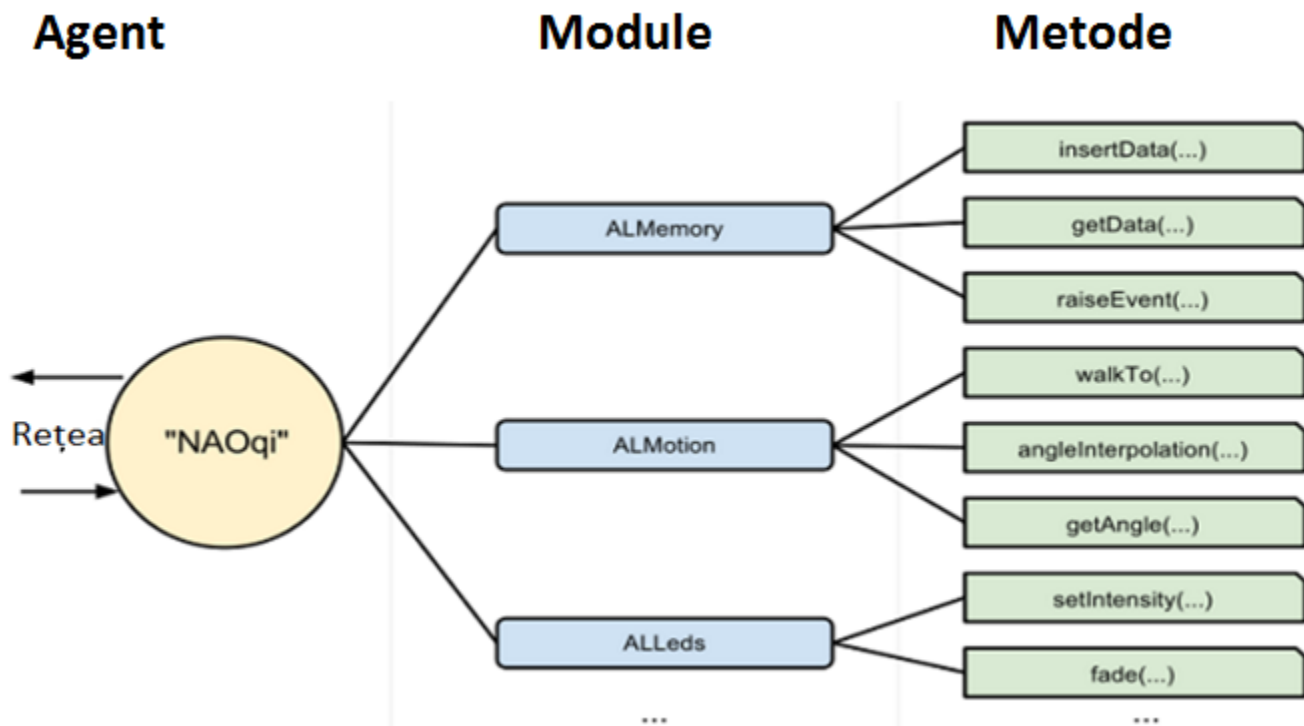


Figura [14] Figura 2.7 Modularitatea Naoqi

În principal, în această lucrare au fost folosite modulele *ALProxy*, *ALMotion*, *ALMemory*, *ALTextToSpeech*, *ALBehaviorManager*, *ALVideoDevice*, *ALBasicAwareness*, *ALSpeechRecognition* și *ALRobotPosture*, ce au ajutat robotul să arate un comportament inteligent la stimuli exteriori și la evenimentele din timpul aplicației practice. În continuare se vor prezenta aceste module conform cu documentația oficială oferită de constructorii robotului.

ALModule reprezintă un modul de stare al celorlalți agenți ce rulează pe robot și poate controla rularea acestora. Acesta poate fi accesat de utilizator și deține în componenta sa o multitudine de funcții ce pot transmite mai departe utilizarea procentuală a unui modul, închiderea acestuia sau o verificare a funcționalității.

ALProxy este un modul utilizat pentru conectarea a doi agenți și copierea lor în memoria robotului. Conectarea a două module se face prin conectarea agenților lor. Conectarea modulelor se face prin specificarea adresei IP și a portului agentului principal. După crearea acestui tip de conexiune, apelarea funcțiilor conținute în agentul importat se va face prin simpla scriere a numelui funcției, fără a fi nevoie de specificarea continuă a adresei de rețea și a portului utilizat. Se poate utiliza și o conexiune de tip proxy către agent. În cazul în care robotul utilizează o astfel de conexiune și dorește conectarea modului B la modulul C, modulul B poate accesa funcțiile din modulul C în timp ce reciproca nu este valabilă.

ALMemory reprezintă o memorie centralizată utilizată pentru stocarea tuturor informațiilor cheie legate de componentele fizice ale robotului. Acesta este un modul ce stochează fără a prelucra sau altera datele. Pe lângă accesarea în timp real a evenimentelor acesta poate stoca și istoricul evenimentului. Utilizatorul poate adăuga la alegere diferite evenimente și le poate accesa prin variabile

de adrese (pointeri) sau prin accesarea directă cu numele variabilei. Accesul cu pointeri are o viteză de până la trei ori mai mare fiind de preferat celei menționate anterior pentru aplicațiile în timp real.

ALMotion este un agent ce conține funcții pentru controlul complet al articulațiilor robotului și al motoarelor. Acesta moștenește metode din modulul *ALModule* prezentat anterior. Dintre principalele funcții îndeplinite de acesta, amintim: controlul tăriei motoarelor, controlul articulațiilor, implementarea funcțiilor de mișcare, funcții de detectare și evitare de obstacole, controlul echilibrului, controlul robotului în cazul în care robotul cade și controlul cartezian pe șase direcții de orientare.

ALRobotPosture controlează poziția robotului pentru ca acesta să își păstreze echilibrul, având o multitudine de funcții predefinite, ce ajută la o mai simplă dezvoltare a aplicațiilor ce presupun mișcări complexe. Dintre utilitățile importante ale acestui modul se pot menționa controlul vitezei și al numărului maxim de încercări pe care robotul le are la dispoziție pentru ajungerea într-o anumite poziție.

ALBasicAwareness ajută robotul să perceapă stimuli proveniți din exterior, aceștia putând fi de mai multe tipuri: zgomot, mișcare sau atingere. Urmărirea stimulilor se face cu ajutorul modulului de memorie la detecția de stimuli de tip atingere sau voce.

ALBehaviorManager ajută robotul să aibă anumite comportamente definite de utilizator sau a celor predefinite. Un comportament reprezintă o sumă de reacții pe care robotul le are la întâlnirea unor stimuli externi sau la încărcarea unui cod pe acesta.

ALVideoDevice reprezintă cel mai important modul utilizat în aplicația curentă, aducând utilizatorului posibilitatea de a se conecta la robot cu ajutorul creării unui agent dedicat prelucrării de imagine. Acesta aduce utilizatorului posibilitatea de a accesa în timp real camera video prezentă pe frunte sau pe bărbia robotului și să le utilizeze paralel pentru o mai bună prelucrare a imaginii. O altă funcție îndeplinită de modul este conversia imaginii din formate de tip RGB în formate de tip HSV sau YUV. Programatorul poate alege ce spațiu de culoare dorește să utilizeze în funcție de nevoile sale, putând să aducă în memoria robotului imagini cu o rezoluție de până la 1280* 960 pixeli cu o rată de achiziție de treizeci de cadre pe secundă.

ALSpeechRecognition definește în interiorul său o multitudine de metode și clase ce ajută robotul să recunoască secvențele vocale primite prin intermediul microfoanelor localizate în zonele laterale ale capului. Programatorul are posibilitatea de a adăuga un număr mare de cuvinte ce pot fi recunoscute de robot și de a implementa diferite secvențe de cod în funcție de nevoile sale. În continuare se prezintă o secvență de cod ce face recunoașterea vocală pentru recunoașterea a două cuvinte în limba engleză: “easy” și “hard”.

```
def ask_for_difficulty (player):  
  
    ROBOT_IP = "10.90.2.78"  
    # Creates a proxy on the speech-recognition module  
    asr = ALProxy("ALSpeechRecognition", ROBOT_IP, 9559)  
    memProxy = ALProxy("ALMemory", ROBOT_IP, 9559)  
    asr.setLanguage("English")
```

```

    # Adds "easy", "hard" and "please" to the vocabulary (without
wordspotting)
    vocabulary = ["easy", "hard"]
    asr.setVocabulary(vocabulary, True)
    player.tts.say ('Do you want hard mode or easy mode?')
    time.sleep(4)

    # Start the speech recognition engine with user Test_ASR
    asr.subscribe("Test_ASR")
    print 'Speech recognition engine started'
    word = memProxy.getData("WordRecognized")[0]

    if (word == '<...> easy <...>'):
        print 'easy has been understood'
        result = 'easy'
    else:
        print 'hard has been understood'
        result = 'hard'

    asr.unsubscribe("Test_ASR")

return result

```

În secvența anterioară de cod se face implementarea la nivel de cod Python a modului existent deja în Choreographie ce recunoaște cuvinte sau secvențe și returnează un rezultat utilizatorului în funcție de bucla de decizie if-else.

ALTextToSpeech permite robotului să reproducă orice secvență de text și să își modifice tonul vocii respectiv volumul la care vorbirea este făcută. Rezultatul sintezei este reprodus de difuzoarele robotului. NAO își poate modifica în mijlocul propoziției tonul sau volumul vocii pentru a avea un efect similar cu vocea umană, folosind funcția *ALTextToSpeech.setParameter()*. [17]

CAPITOLUL III

Jocul de X și 0 și implementarea acestuia

Jocul de X și 0 este un joc cunoscut încă din vremuri străvechi, mai exact din perioada antichității, în jurul primului secol de dinaintea erei noastre. A fost denumit *Terni Lapilli* și în locul unui joc care se termină, fiecare jucător deținea trei pietre pe care trebuia să le ordoneze într-o linie, acestea fiind puse într-un pătrat de dimensiuni 3x3. În ciuda simplității aparente a jocului, el necesită o analiză detaliată pentru a determina numărul de combinații posibile pentru joc.

Implementarea acestuia a fost făcută cu două module principale, unul dintre ele fiind modulul de procesare în timp real al imaginii iar cel de-al doilea este modulul de mișcare. În ajutorul acestora au fost construite alte module adiționale ce vor fi explicate în continuare, fiind apelate de către funcția *main* din modulul *main_function*.

3.1 Conceptul aplicației

Aplicația software a fost gândită pe modelul unui agent inteligent, prezentat în introducerea lucrării, fiind adaptată pentru situația curentă, astfel, agentul inteligent a fost împărțit în cele cinci părți componente în interiorul cărora este descris modelul comportamental. Mediul, în acest caz este compus din adversar și mediul propriu-zis de joc. Împărțirea pe module a fost făcută din rațiuni de respectare a paradigmei de programare, algoritmi de joc fiind cei de decizie asupra acțiunilor următoare. Modulul de procesare de imagini se ocupă de procesarea stimulilor externi, împreună cu modulul de recunoaștere vocală, iar modulul de mișcare și comportament, este cel care efectuează acțiunea, conform cu figura următoare.

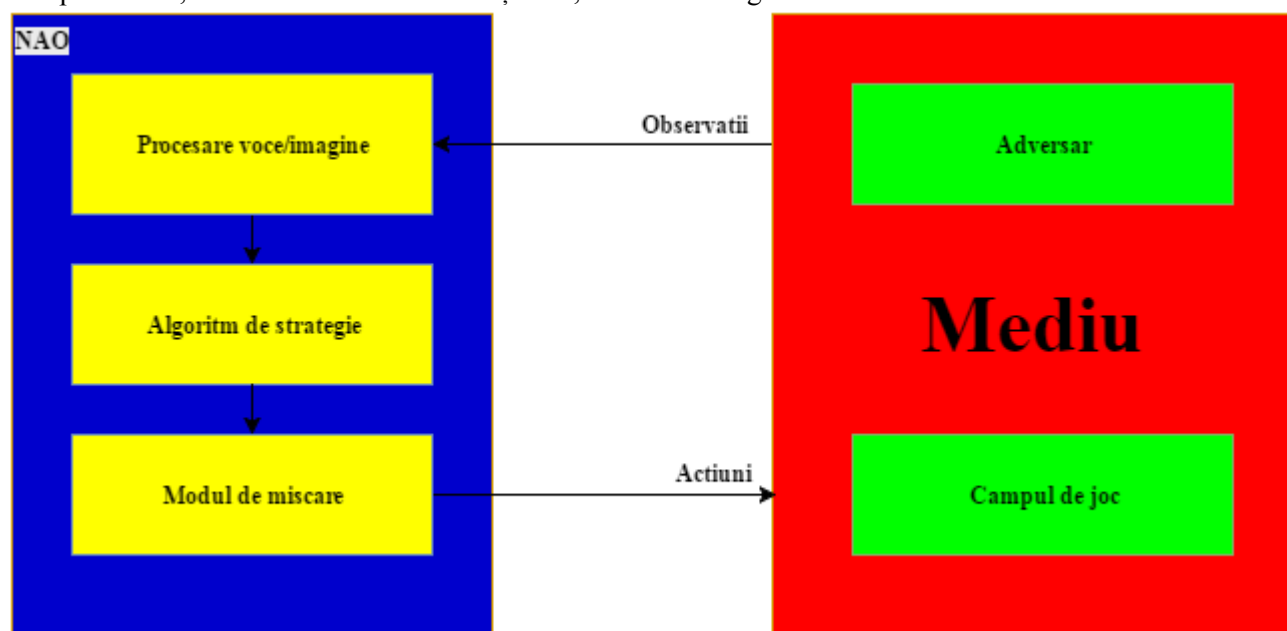


Figura [15] 3.1 Conceptul aplicației software

3.2 Modulul main

Acest modul este cel în care utilizatorul se conectează la robot scriind adresa de rețea și portul prin care se vor încarca modulele componente. Acesta va instanția în interiorul său clasă *NaoXO* ce are în componența sa funcții ce vor fi explicitate în capitolul 3.3. Conectarea la robot se face printr-o conexiune de tip SSH în rețeaua locală. Folosind instrucțiunea `import` se aduc la dispoziția utilizatorului diferite funcții de sistem. După conectare, acesta va poziționa robotul în poziția predefinită *standInit*, aceasta fiind favorabilă poziționării câmpului de joc în rază vizuală a robotului. În cazul în care robotul nu reușește să înceapă jocul, pe fereastra utilitarului *IDLE* va apărea un mesaj “game not Initialized” împreună cu un mesaj de eroare în cazul în care acesta există. Robotul va apela metoda de curățare a variabilei, acesta fiind destructorul clasei *NaoXO* și va afișa eroarea. Robotul va întreba utilizatorul la finalul jocului dacă dorește o rejucare a jocului folosind modulul de vorbire deja existent pe acesta, adus la dispoziția utilizatorului folosind instrucțiunea *ALProxy*.

3.3 Modulul de procesare al imaginilor

Acest modul este responsabil cu găsirea câmpului și procesarea de imagine, fiind folosite cu precădere funcții predefinite din platforma *openCV*. Acesta are în componența sa o multitudine de funcții de procesare a imaginilor, ce oferă un răspuns în timp real la stimuli exteriori. În interiorul acestui modul este definită clasă *ImgprocessingXO()* ce va fi folosită ulterior. Aceasta preia de la camera video a robotului secvențe de imagini pe care le procesează și oferă celorlalte module componente informațiile necesare pentru starea jocului și modul în care este poziționată tabla de joc.

Prima funcție utilizată în apelarea acesteia, pe langa constructorul clasei este funcția de găsim a liniilor din imagine. Funcția este sugestiv denumită *preprocessLines* și va fi explicată ulterior. Funcția convertește imaginile primite de la camera într-un format de imagine alb-negru și o afișează utilizatorului folosind funcția *imshow*. Următorul pas este găsirea liniilor prin aplicarea unui algoritm de detecție denumit *HoughLines* ce va găsi liniile din imagine prin diferențierea culorilor și le va afișa pe imagine cu roșu. În cazul în care robotul nu găsește liniile, acesta va afișa pe ecran mesajul *I have found no lines*, ajutând utilizatorul să optimizeze funcțiile utilizate.

```
def preprocessLines(self, image):  
    '''  
    Here we convert the image to grayscale  
    extract the edges,  
    and apply hough to obtain the lines in the field  
    '''  
    self.img_grayscale = openCV.cvtColor(image,openCV.cv.CV_BGR2GRAY)  
    openCV.imshow("Black_And_White", self.img_grayscale)  
    ## show edges  
    edges = openCV.Canny(self.img_grayscale, self.lCanny, self.uCanny)  
    openCV.imshow("Edges", edges)  
    openCV.waitKey(1)  
    ## imshow does not work without waitKey
```

```

    ## use try-catch method for error detection
    try:
        lines = openCV.HoughLines(edges, self.rhoRes, self.thetaRes,
self.houghThreshold) [0]
        image_lines = self.draw_lines(image.copy(), lines)
        openCV.imshow("Lines", image_lines)
        openCV.waitKey(1)
    except:
        print("I have found no lines")
        #blank return
        return []
    return lines

```

După găsirea liniilor, robotul va găsi intersecțiile dintre acestea și va aplica un algoritm de detecție asupra intersecțiilor pentru verificarea lor cu ajutorul funcției *checkIntersections*. Această funcție găsește o aglomerare de puncte negre în jurul unui punct iar dacă orientarea aglomerărilor va fi de 90 de grade va semnaliza acest fapt printr-o variabilă booleană. Următorul pas este reprezentat de găsirea punctelor de colț din imagine, presupunând că fiecare dintre șabloanele utilizate pentru tablă arată identic, fiind matrici de dimensiuni 3x3.

Principala funcție implementată în acest modul, ce reușește să coreleze celelalte funcții este reprezentată de funcția *getGameState()* ce primește ca parametri *self* (obiectul asupra căruia funcția acționează), *img* (imaginea provenită de la camera video și *margin_lines* ce reprezintă vectori cu marginile tablei. Aceasta returnează utilizatorului doi parametri, *state* și *board*, doi vectori de dimensiune 9x1 care stochează starea actuală a jocului.

```

def getGameState(self, img, margin_lines):

    ### function used to calculate the state of the game and store it
    ## by analyzing the image from the camera
    ## create vector for storing the game state
    state=['-']*9
    board = [[-1,-1,-1],
              [-1,-1,-1],
              [-1,-1,-1]]
    ## create board
    if margin_lines:
        ## if there are the lines accordingly to the pattern
        if len(margin_lines)==4:
            noughts, crosses = self.getCentroids(img)
            #used getCentroids for getting the state of the game
            if noughts:
                ## check positions for zeros and put them in the field
                for nought in noughts:

```

```

        (ind_x, ind_y)=self.getBoxInField(nought,
margin_lines)

        ## save in state
        state[ind_x*3+ind_y]='o'
        #replace 'o' with zero
        board[ind_x][ind_y] = 0
    if crosses:
        ## get crosses and put them in the right position
        for cross in crosses:
            (ind_x, ind_y)=self.getBoxInField(cross,
margin_lines)

            ## save in state
            state[ind_x*3+ind_y]='x'
            # replace 'x' with ones
            board[ind_x][ind_y] = 1

    ## here we return the params in which were saved the positions
    return state, board

```

Funcția apelează în interiorul său funcția *getCentroids(img)*, o funcție folosită pentru găsirea formelor în interiorul imaginii. Aceasta folosește o limită pentru detecția formelor ignorând posibilele semne mici și greu de distins ce ar putea fi doar erori de scriere ale robotului. Folosind acest tip de detecție și folosind parametri liniilor de margine, robotul găsește semne de X sau de 0 și le introduce în doi vectori, o și x folosind instrucțiunea *append*.

Detecția semnelor de X și 0 se face folosind funcțiile *getContours_X(self, imgHSV)* respectiv *getContoursO(self, imgHSV)* ce împarte imaginea într-un spațiu de culori. Funcția face găsirea contururilor de X și 0 diferit, însă singură diferență remarcabilă este reprezentată de schimbarea parametrilor limită..

```

def getContours_O(self, imgHSV):
    '''
    Finds O's written with green
    '''
    lower_green= np.array([35,74,50])
    upper_green= np.array([70,255,200])
    ## segment the image
    binImg = openCV.inRange(imgHSV, lower_green, upper_green)
    openCV.imshow('zerouri', binaryImg)
    ## dilate and erode using morphologyEx
    binImg = openCV.morphologyEx(binaryImg, openCV.MORPH_OPEN,
np.ones((3,3),np.uint8))
    ## contour find and append
    contours_o = openCV.findContours(binImg, openCV.cv.CV_RETR_TREE,
openCV.cv.CV_CHAIN_APPROX_SIMPLE) [0]

```

```
## return
return contours_o
```

Rezultatul împărțirii este afișat utilizatorului pe ecran în două imagini separate, una pentru formele de X iar cealaltă pentru formele de O.



Figura [16] 3.2 Împărțirea imaginii HSV în două imagini distincte

După găsirea formelor de X și de O robotul va returna utilizatorului cei doi vectori amintiți anterior ce vor fi introduși în imaginea Game folosind instrucțiunea *openCV.imshow*. Aceasta reprezintă un răspuns la stimuli exteriori în mod continuu, reușind să avertizeze utilizatorul în cazul în care detecția se face incorect prin simpla comparație dintre imaginea de pe ecran și cea din fereastra *Game*.

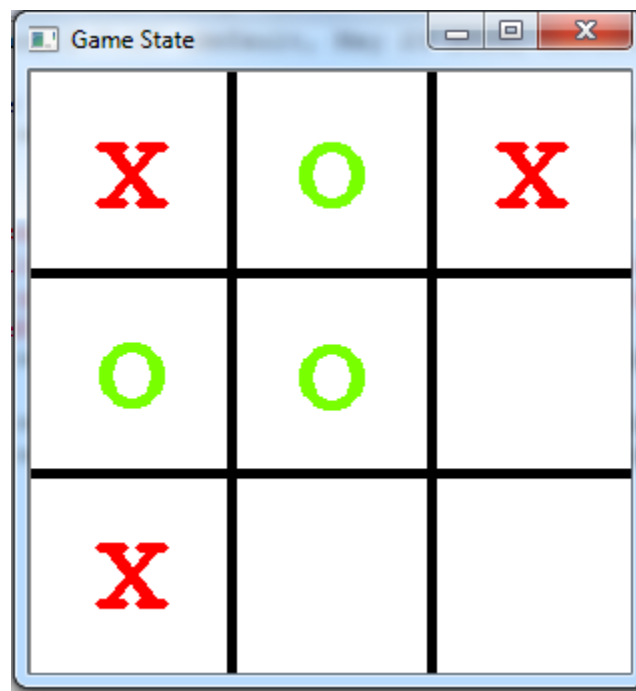


Figura [17] Figura 3.3 Fereastra GAME

3.4 Modulul de recunoaștere vocală

Modulul de recunoaștere vocală reprezintă o implementare și o adaptare a exemplului oferit de constructorul robotului ca exemplu. Este un modul additional ce aduce robotului posibilitatea de a comunica într-o formă verbală cu utilizatorul. Scopul principal al acestuia este de a returna modulului *motionlib.py* explicat anterior și modulului *tic_tac_toe_strategy.py* dificultatea cu care robotul va juca jocul de X și 0.[18]

```
import time
from naoqi import ALProxy

def ask_for_difficulty (player):

    ROBOT_IP = "IP address here"
    # Creates a proxy on the speech-recognition module
    asr = ALProxy("ALSpeechRecognition", ROBOT_IP, 9559)
    memProxy = ALProxy("ALMemory", ROBOT_IP, 9559)
    asr.setLanguage("English")

    # Adds "easy", "hard" and "please" to the vocabulary (without
    wordspotting)
    vocabulary = ["easy", "hard"]
    asr.setVocabulary(vocabulary, True)
    player.motion.setStiffnesses("Body", 1.0)
    player.tts.say ('Do you want hard mode or easy mode?')
    time.sleep(8)
    asr.subscribe("NAOXO ")

    print 'Speech recognition engine started'
    time.sleep(10)
    word = memProxy.getData("WordRecognized")[0]

    if (word == '<...> easy <...>'):
        print 'easy has been understood'
        result = 'easy'
    else:
        print 'hard has been understood'
        result = 'hard'

    asr.unsubscribe("NAOXO")
    return result
```

Acest modul utilizează patru module deja existente pe robot, *ALProxy*, *ALMemory*, *ALSpeechRecognition* și *ALTextToSpeech* pentru a comunica mesaje utilizatorului. Pe lângă returnarea rezultatului analizei vocale, robotul va afișa și în fereastra de comanda cu ajutorul instrucțiunii *print*

cuvântul înțeles. Secvența returnată va fi preluată apoi de către modulele de strategie și modulul de mișcare al robotului. Funcția primește ca parametru variabila *player* ce reprezintă o instanță a clasei NaoXO, se conectează la robot folosind adresa de rețea și adaugă în vocabularul robotului două cuvinte, *easy* și *hard* ce sunt echivalentele nivelului mic respectiv mare de dificultate al jocului..

3.5 Modulele de strategie

Modulele de strategie sunt împărțite în două părți, o parte pentru determinarea câștigătorului, iar cea două parte componentă este algoritmul de decizie al următoarei mișcări. Algoritmul de determinare al câștigătorului este prezentat în secvența următoare de cod pentru determinarea câștigătorului pe linii, respectiv coloane.

În secvența următoare de cod, robotul primește ca parametru o matrice de dimensiuni 3x3, în care semnele de X sunt înlocuite cu valori de 1 iar semnele de 0 sunt înlocuite cu valori de 0, spațiile goale fiind desemnate ca fiind înlocuite cu -1 din funcțiile anterioare. Dacă rezultatul analizei pe linii la numărătoarea semnelor de X sau 0 este 3 acesta va returna utilizatorului o secvența pentru a fi procesată ulterior. În caz contrar, rezultatul va fi *still_playing*, semnificând continuarea jocului. Secvența de cod este repetată pentru numărătoarea pe coloane, respectiv pentru numărătoarea pe diagonale. Aceasta este un modul important în modulul de strategie întrucât indică robotului dacă poate efectua o mișcare ce îi poate aduce victoria.

```
def determining_the_winner (panel):
    result="draw"

    #lines
    for i in range(3):
        number_of_xs=0
        number_of_os=0
        for j in range(3):
            if panel[i][j]==1:
                number_of_xs=number_of_xs+1
            elif panel[i][j]==0:
                number_of_os=number_of_os+1
        difference=3-number_of_xs-number_of_os
        if (difference>0) and (result=="draw"):
            result="still playing"
        elif number_of_xs==3:
            result="winner x"
        elif number_of_os==3:
            result="winner o"
```

A două parte componentă a modulelor de strategie este modulul *tic_tac_toe_strategy* ce are în componența sa mai multe funcții. Robotul utilizează pentru a juca jocul două metode, în funcție de

simbolul ce îi este alocat, presupunând tot timpul că simbolul de X este cel care începe jocul. Algoritmul de joc folosește câteva funcții adiționale pentru determinarea locurilor disponibile pentru inserarea simbolurilor. În continuare vor fi prezentate aceste funcții. Aceste două funcții returnează pozițiile libere respectiv pozițiile cu X-uri de pe tablă, cea de-a doua funcție, având corespondentul ei pentru simbolurile de O. Acestea vor introduce în câte o matrice, folosind instrucțiunea `append` valorile pentru a fi returnate.

```
def free_places_on_board (board):

    free_places=[]
    for i in range (3):
        for j in range (3):
            if board [i][j]==-1:
                free_places.append([i,j])
    return free_places

def x_locations (board):
x=[]
    for i in range (3):
        for j in range (3):
            if board [i][j]==1:
                x.append ([i,j])
    return x
```

Funcțiile `strategy_x`, respectiv `strategy_o` sunt cele două funcții vizibile în exteriorul modulului, fiind împărțite în două părți, în funcție de secvența primită de la modulul de voce. În cazul în care robotul a primit secvența 'easy' acesta va lua decizia de a pune aleator în spațiile libere de pe matricea de joc simbolul.

```
def strategy_x (board,difficulty):
    if(difficulty == 'easy'):
        temporary_copy=copy(board)
        free_places=free_places_on_board(temporary_copy)
        field = free_places[randint(0,len(free_places)-1)]
```

În cazul în care secvența este modul greu de joc, acesta va lua o decizie ce respectă următoarea diagrama de stări, ce îi aduce acestuia minim un joc egal cu utilizatorul. Diagrama logică din figura următoare presupune ca pozițiile în care robotul poate pune simboluri sunt calculate la fiecare trecere.

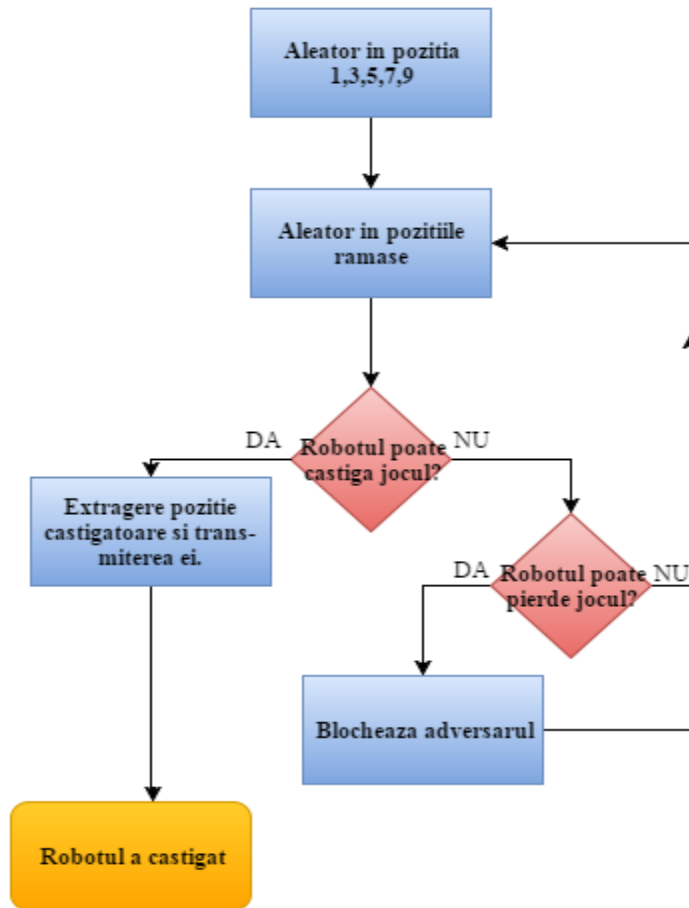


Figura [18] 3.4 Diagrama de stări a algoritmului de decizie

3.6 Modulul de mișcare și analiză a stării jocului

Modulul de mișcare și analiză a stării jocului, denumit sugestiv *motionlib.py* este unealta software principală prin care robotul interacționează cu programatorul. Acest modul este apelat de către funcția main prin instantierea clasei *NaoXO* în corpul funcției principale. Ca orice clasă, aceasta are nevoie de un constructor, denumit conform paradigmei de programare `__init__` ce face conexiunea către modulele explicate anterior, modulul de comportamente, modulul de memorie, modulul video, modulul text către vorbire și modulul de procesare de imagine cu ajutorul instrucțiunii `self.imgproc = img.imgProcessingXO(self.size)` unde parametrul `self.size` reprezintă dimensiunile imaginilor procesate de către camera video.

Următoarea funcție apelată este `cameraInit`, funcție ce primește ca parametri 5 variabile, f_x , f_y , c_x , c_y și s ce reprezintă distanță focală pe axa x și y, centrul imaginii pe axa x și y iar s este factorul de corecție al înclinației camerei.

Cea mai importantă funcție din modul este reprezentată de funcția `gameInit` ce conține în interiorul său o buclă infinită de tip `while True` ce utilizează modulul de procesare de imagine. Metoda va căuta în primă instanță câmpul de joc, folosind modulul de procesare explicat anterior și va încerca

sa găsească diferențe între starea anterioară și starea curentă a jocului, comparând doi vectori de stare. La prima apelare, acesta va număra obiectele de pe tablă.

```
## if there are 2 or more objects on the playing field, robot does
not know how to start
    if count_x_on_the_field + count_o_on_the_field >= 2:
        self.tts.say("Too many objects on the playing field, please
clean the board and restart")
        return False

    elif count_x_on_the_field == 1 and count_o_on_the_field == 0:
        self.tts.say("You started first. Ok, I will be playing with
zeros, please draw them with a green pen")
        self.mode = 'o'
        self.difficulty = ask_for_difficulty(self)
        #self.difficulty == 'hard'
        return True

## clean board, robot plays first
elif count_x_on_the_field == 0 and count_o_on_the_field == 0:
    self.tts.say("Ok, I will play first using crosses, please draw
them with a red pen")
    self.mode = 'x'
    #self.difficulty == 'hard'
    self.difficulty = ask_for_difficulty(self)
    return True
```

În timpul execuției acestei secvențe de cod, robotul va avea trei posibilități. În cazul în care pe tabla vor fi mai mult de două semne de X sau 0 acesta va abandona jocul și va cere utilizatorului restartarea programului. În cazul în care numărul de X-uri este unu, acesta va ști faptul ca utilizatorul a început jocul și va juca utilizând simbolul 0, în caz contrar, tabla fiind goală, va începe jocul. Următorul pas este apelarea modulului de recunoaștere vocală, ce va face robotul să întrebe nivelul de dificultate la care se dorește a se juca jocul, stocând în memorie valoarea returnată, *easy* sau *hard*.

Metoda *setFieldSize* este o funcție ce primește ca parametru două obiecte date de utilizator: *self*, instanță a clasei asupra căreia se fac operații și *dimension* reprezentând dimensiunea câmpului de joc. Vor fi create în memoria robotului trei matrici de coordonate folosind centrul imaginii ca origine a axelor, acestea fiind:

- Colțurile pătratului central;
- Centrele pătratelor;
- Colțurile plăcii de joc.

O metodă importantă în dezvoltarea acestei aplicații este *getImage(self)* ce aduce memoria robotului imaginea de la camera video, astfel :

```

def getImage(self):
    img_params = self.videoProxy.getImageRemote(self.video)
    openCV.cv.SetData(self.imgheader, img_params[6])
    self.image = np.asarray(self.imgheader[:, :])
    return self.image

```

Pentru a nu utiliza o singură metodă și a folosi valori fixe, robotul dispune de posibilitatea aducerii parametrilor camerei video prin metoda *getImageRemote* care returnează într-un vector cu șase poziții lărgimea și lățimea imaginii, numărul de spații de culoare utilizate, spațiul de culoare, ștampila de timp cu secunda capturii și cea cu miliseconda capturii. Fiecare pixel al imaginii este convertit într-un vector de valori conținând trei parametri, luminuoșitate, nuanță și saturație, conform standardului HSV, fiind procesat mai departe de către algoritmi de viziune.

Alte două funcții importante implementate în cadrul acestui modul sunt funcțiile *checkStates* și *updateState* acestea reprezentând modul în care robotul prelucrează informația din mediul exterior. Prima dintre acestea două verifică dacă starea anterioară este aceeași cu starea curentă, în cazul unei inegalități, aceasta va apela funcția care face remprospatarea stării jocului prin modulul de procesare de imagine, făcând o actualizare în modulul de strategie al robotului..

Desenarea utilitarului *Game State*, menționat anterior, se face utilizând metoda *draw_board* ce va returna într-o imagine pe ecranul calculatorului conectat la robot starea curentă a jocului, fiind cel mai bun mod de verificare a corectitudinii codului și al funcționării modulului de procesare de imagine.

```

def draw_board(self, state):
    '''
    here we draw the state of the game according to the game
    '''
    #white background
    game_state = np.ones((300,300,3), np.uint8)
    game_state[:]=(255,255,255)
    #black lines
    openCV.line(game_state, (0,100), (300,100), (0,0,0),3,8)
    openCV.line(game_state, (0,200), (300,200), (0,0,0),3,8)
    openCV.line(game_state, (100,0), (100,300), (0,0,0),3,8)
    openCV.line(game_state, (200,0), (200,300), (0,0,0),3,8)
    #here we create the boxes
    fields
    [(30,65), (130,65), (230,65), (30,165), (130,165), (230,165), (30,265),
    (130,265), (230,265)]

    for counter_i in range(9):
        if state[counter_i]=='x':

```

```

openCV.putText(game_state, state[counter_i], fields[counter_i], openCV.cv
.CV_FONT_HERSHEY_COMPLEX, 2, (0, 0, 255), 3, 8)
    #red x-s
    if state[counter_i]=='o':
openCV.putText(game_state, state[counter_i], fields[counter_i], openCV.cv
.CV_FONT_HERSHEY_COMPLEX, 2, (0, 255, 120), 3, 8)
    #green zeros
    return game_state

```

Acesta va desena pe ecran un pătrat având dimensiunea de 300 x 300 pixeli, alb, și va crea în interiorul acestuia nouă căsuțe. În acestea, folosind metoda predefinită *putText*, utilizatorul va observa faptul că în momentul în care robotul “scrie” un simbol pe tablă, acesta va apărea în utilitar, fie că este X, fie că este 0, acest fapt întâmplându-se și în cazul în care utilizatorul va desena un simbol pe tablă.

Un alt utilitar folosit pentru găsirea câmpului și vizionarea modului în care robotul vede câmpul de joc este funcția *drawstuff*.

```

def drawstuff(self, flag)
    ## open two windows for debugging
    openCV.namedWindow("Initial Image")
    openCV.namedWindow("Game State")

    ## if the flag is set to false, do nothing
    if flag:
        #draw valid intersections
        for i in range(4):
            intersection_one =
(openCV.cv.Round(self.imgproc.corners[0][i]),
openCV.cv.Round(self.imgproc.corners[1][i]))
            intersection_two =
(openCV.cv.Round(self.imgproc.corners[0][(i-1)%4]),
openCV.cv.Round(self.imgproc.corners[1][(i-1)%4]))
            openCV.line(self.img, intersection_one,
intersection_two, (0,255,0), 3, 8)
            #draw lines using red
            openCV.circle(self.img, intersection_one, 5,
(0,0,255), openCV.cv.CV_FILLED)
            #here we put points in the places of the intersections
using blue (0, 0, 255)= params rgb
            for point in self.intersections:
                openCV.circle(self.img, point, 5, (255,0,0),
openCV.cv.CV_FILLED)

        ## show image
        game_state = self.draw_board(self.state)

```

```

openCV.imshow("Initial Image", self.img)
openCV.imshow("Game State", game_state)
## waitKey used for being able to use imshow function, does
not work without one
openCV.waitKey(1)

```

Pe imaginea inițială, denumită sugestiv cu traducerea în limba engleză, *Initial Image*, robotul va desena utilizând culoarea roșie liniile găsite în urma aplicării algoritmilor de detecție al intersecțiilor. Intersecțiile vor fi desenate cu culoarea albastră, conform standardului BGR, acestea având o valoare a culorii albastre de 255/255, iar valorile culorilor componente verde și roșu vor fi setate la valoarea 0. Aceasta va introduce în interiorul căsuței de vizualizare *Game State* starea jocului la momentul curent, în cazul în care valoarea variabilei flag este diferită de zero.

Funcția *checkResult* reprezintă modul în care robotul interacționează cu modulul de determinare a câștigătorului, folosindu-se de modulele de viziune și de comportamentele dezvoltate în programul Choreographe. În ceea ce privește rezultatele returnate de robot, acestea sunt patru numere, ce vor fi prelucrate ulterior, -1, 0, 1, 2. În cazul în care robotul a câștigat, rezultatul va fi 1, în cazul în care robotul a pierdut va fi 2, în cazul în care jocul a fost terminat la egalitate metoda va returna 0 iar în cazul în care jocul este încă în desfășurare, returnând -1, robotul va apela modulele de strategie și viziune.

CalculateGoalPos(self) este o metodă prin care robotul îi va comunica utilizatorului diferite decizii de punere a valorilor într-un anume pătrat folosind comunicarea cu modulul de strategie explicitat anterior. În cazul în care acesta ia decizia ca un simbol să fie poziționat pe poziția Y, unde Y este un număr întreg între unu și nouă, acesta va transmite utilizatorului vocal, folosind instrucțiunea *tts.say* unde dorește să pună simbolul pentru a ușura modalitatea prin care cei doi interacționează. Câmpurile în de X 0 numerotate de la unu la conform următoarei figuri.

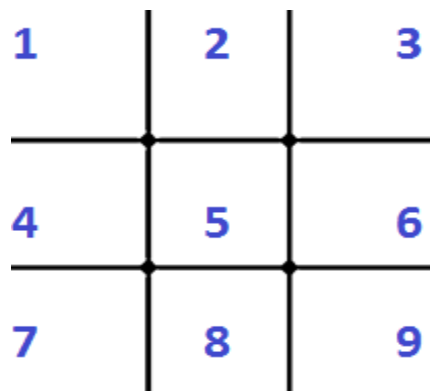


Figura [19] 3.5 Numerotarea căsuțelor pe tabla de joc

Funcția *play(self)* este folosită de robot pentru a lega componentele centrale ale modulului de mișcare. Acesta este folosit de robot pentru a găsi câmpul și pentru a repositiona robotul în cazul în care acesta nu găsește cele patru intersecții. Următorul pas este verificarea stării jocului, iar în cazul în

care robotul a pierdut sau observa o egalitate, va activa comportamentele predefinite în utilitarul *Choreographe*. În cazul în care rezultatul verificării este “*inca se joaca*” acesta va verifica validitatea și apariția unor noi semne pe tablă, transmițând utilizatorului “*este rândul tau*”. În cazul în care robotul va decide că a câștigat jocul, pe baza algoritmului de decizie din modulul *determining_the_winner*, acesta va activa comportamentul ce va fi explicat ulterior..

Metoda *playMove* este cea care animă robotul prin apelarea comportamentelor pentru indicarea pozițiilor în care robotul dorește să pună un semn de X sau O. Acestea au fost încărcate și construite pe robot cu ajutorul funcției *Timeline* din utilitarul *Choreographe*, ce permite utilizatorului să înregistreze mișcări complexe, că mai apoi robotul să le poată reproduce integral, fără a fi nevoie de un control utilizând funcții implementate în limbajul Python. Fiecare poziție are un comportament diferit, robotul indicând poziția în funcție de mâna care este cel mai aproape de câmp. În cazul în care câmpul dorit este 1, 4 sau 7, Nao va indica poziția cu ajutorul mâinii stângi, iar în caz contrar, va folosi mâna dreaptă. Apelarea din modulul de mișcare se face cu ajutorul comenzii *behavior.runBehavior*, funcție definită în modulul comportamental amintit în capitolul 3.2.

```
if (temp_goal == 1):  
    self.behavior.runBehavior('tic_tac_toe_animations-  
62c8f2/point_towards_1')
```

Apelarea funcției se face prin transformarea unui parametru primit de funcție într-o variabilă de tip întreg și prelucrarea ei cu ajutorul unei bucle de decizie. În figura următoare sunt prezentate trei cadre din modul în care robotul indică poziția 1, implementată pe un robot simulat.

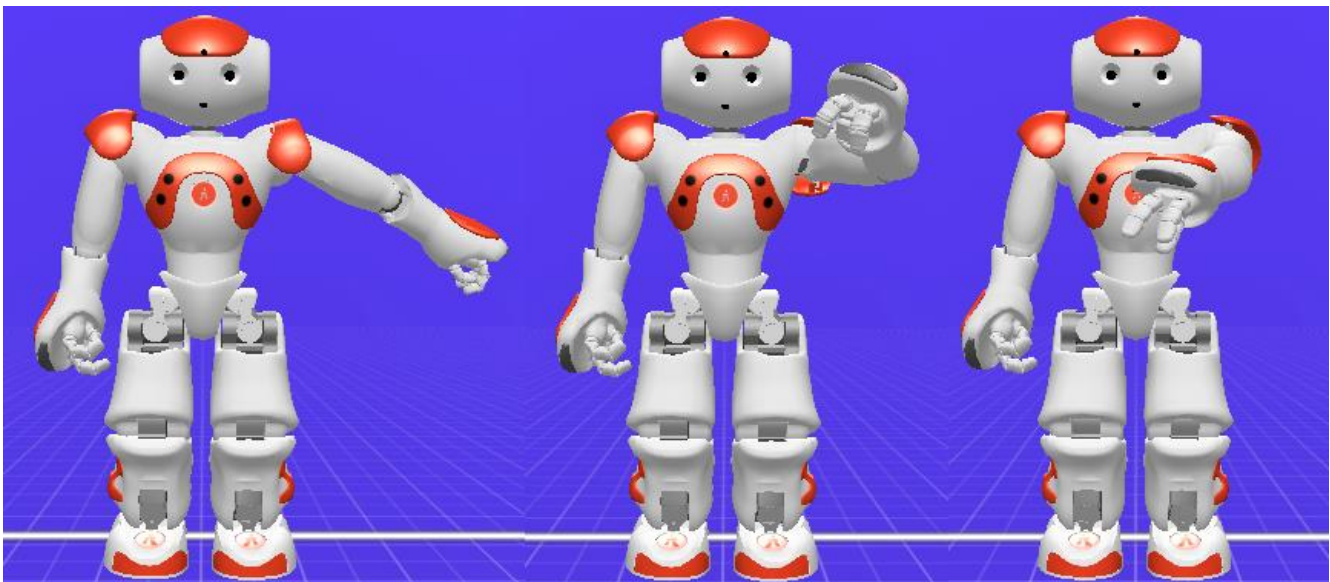


Figura [20] 3.6 Indicarea pozitiei 1 folosind un robot simulat

După executarea acestei mișcări, robotul va reveni în poziția inițial folosind funcții predefinite din librăria de mișcare și librăria de posturi posibile, ajungând în poziția *standInit* executând

următoarea secvență de cod ce setează tăria motoarelor din incheituri și viteza cu care robotul ajunge în poziție:

```
self.motion.setStiffnesses("Body", 1.0)
self.motion.setStiffnesses("LArm", 0.0)
self.motion.setStiffnesses("RArm", 0.0)
self.posture.goToPosture("StandInit", 0.5)
self.torso_default = self.motion.getPosition("Torso", 2, True)
self.motion.wbEnableEffectorControl(nameEffector, False)
```

Animațiile au fost construite într-un proiect Choreographe cu ajutorul utilitarului Timeline, ce reușește să înregistreze valorile actuatorilor robotului în fișiere scrise în limbajul C++ sau Python sau ce pot fi vizualizate imediat în utilitarul ante-menționat. Motoarele, cele douăzeci și cinci de grade de libertate pe care robotul le are sunt salvate ca valori reale, putând fi modificate în orice moment de utilizator prin accesarea imaginii de pe ecran a robotului și modificarea valorilor actuatorilor.

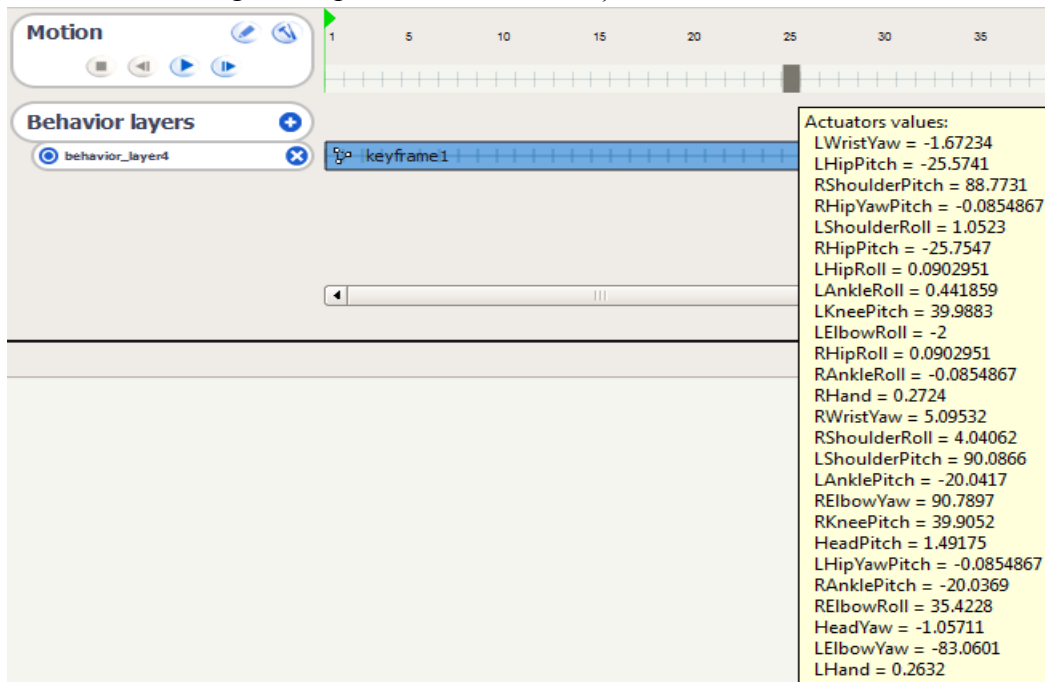


Figura [19] 3.7 Valorile actuatorilor în momentul execuției unei mișcări.

În cazul în care în execuția programului erori fatale, determinate de imposibilitatea conectării, pierderea conexiunii către utilizator, imposibilitatea conectării la un modul, robotul sechiunea de cod denumita sugestiv cleanup ce primește parametri obiectul, si executa următoarea secvență de cod pentru a deconecta robotul de la modulul video a pentru închide comunicarea cu modulul openCV, astfel:

```
def cleanup(self):
    self.videoProxy.unsubscribe(self.video)
    openCV.destroyAllWindows()
```


CAPITOLUL IV

Concluzii

Obiectivul principal al acestei lucrări a fost dezvoltarea și implementarea unui joc de X și 0 pe robotul umanoid NAO, exemplificarea practică a acestuia făcându-se prin implementarea la nivel vocal și motric al modulelor de strategie. În esență, robotul trebuie să răspundă stimulilor proveniți de la mediul exterior și să comunice cu programatorul, în scopul unei mai bune înțelegeri a modului în care robotul operează și procesează diferiți stimuli de tip mișcare sau viziune. Pentru atingerea acestui scop au fost stabilite câteva ținte principale: înțelegerea algoritmilor de viziune, înțelegerea algoritmilor de mișcare și a programelor software utilizate și mai cu seamă, implementarea unui sistem ce dorește să fie utilizat în continuare în vederea optimizării și al implementării unor funcții mai stabile.

Primul obiectiv a constat în implementarea unor algoritmi de viziune pe robot, fiind implementat cu ajutorul librăriei predefinite openCV ce se dorește a fi o platformă comună de dezvoltare a aplicațiilor de viziune, de detectare a obiectelor și de găsimă a câmpului de joc. În dezvoltarea acestei părți a aplicației software au fost luate în considerare diferiți factori de zgomot ce ar putea destabiliza sistemul, acesta având o multitudine de sisteme de asigurare a funcționării corecte în caz de eroare și de detecție a acestora.

Această teză prezintă etapele parcurse de autor în vederea obținerii unei aplicații ce este stabilă și reușește să integreze în structura sa diferite module existente pe robot cu cele noi aduse în structura acestuia. Capitolul 1 prezintă o scurtă descriere a caracteristicilor importante ale robotului NAO din punct de vedere hardware cât și software, conform cu documentația oficială. O altă parte importantă a primului capitol este explicarea modului în care robotul efectuează mișcări complexe cu ajutorul microcontrolerelor principale și ale celor secundare, prezente în articulații, cât și particularitățile de funcționare a acestora. Autorul a fost nevoit să înțeleagă modul în care robotul lucrează și acționează la comenzile ce îi vin din exterior.

Capitolul al doilea reprezintă o prezentare succintă a limbajelor de programare utilizate precum și a librăriilor predefinite ce au fost utilizate, având o caracteristică personalizată aplicației software. Au fost explicitate cele două librării principale utilizate, sistemul de operare și paradigma de programare, împreună cu limbajul de programare Python.

Capitolul 3 este capitolul în care autorul a expus contribuția personală, scrierea codului, implementarea anumitor algoritmi de viziune pe robot, mai exact crearea unor module în vederea găsimii modului optim de implementare a aplicației pe robotul umanoid. Autorul a luat în considerare limitările robotului și ale algoritmilor de viziune ce nu reușesc să funcționeze de cele mai multe ori la parametri doriți datorită multiplelor probleme ale robotului, precum blocarea firelor de execuție sau funcționarea incorectă a unor module existente pe robot. Feedback-ul primit de autor în cadrul aplicației practice se realizează cu ajutorul modului vocal și al modulelor comportamentale, fiind creat special pentru această aplicație de inteligență artificială.

Etaplele parcurse de autor în realizarea practică au avut ca obiectiv crearea unui sistem de inteligență artificială ce ar putea să fie utilizat de orice programator în vederea înțelegerii mai bune a componentelor software ale robotului și a modului în care acesta face diferite feluri de procesare a imaginii.

Rezumatul contribuțiilor presonale ale autorului poate fi următorul:

- Conceperea aplicației software;
- Stabilirea unui mod de generare a feedback-ului în timp real;
- Implementarea modulelor de strategie;
- Implementarea modulelor de viziune pe robotul NAO;
- Crearea unui modul de recunoaștere vocală;
- Crearea unor comportamente specifice pentru aplicația de față.

Soluția sistemului propusă de autor este realizabilă practic, după cum se poate dovedi în aplicația practică, ce a fost realizată în scopul înțelegerii modului în care robotul acționează și lucrează sub acțiunea diferiților stimuli proveniți din mediul exterior. Sistemul descris în aceasta teză poate veni ca un punct de plecare în realizarea unor algoritmi mai complecși de joc pentru robotul NAO, putând fi adaptat și pentru alte tipuri de jocuri de tip “board”. Proiectele viitoare ce pot fi implementate pe robot, pornind de la aplicația de față, ar putea fi:

Implementarea unor jocuri mai complexe: GO, Connect-4 sau Șah;

- Îmbunătățirea sistemelor de viziune și dezvoltarea lor pentru o stabilitate ridicată și o detecție mai bună;
- Îmbunătățirea modului de mișcare în vederea eliminării codării fixe a comportamentelor și animării robotului;
- Sintetizarea de voce pentru limbă română în vederea eliminării acestui neajuns și implementarea modulelor de recunoaștere și vorbire pe robot.

Bibliografie

- [1] Russell, Stuart J. (Stuart Jonathan) Artificial intelligence: a modern approach/ Stuart Russell, Peter Norvig, Prentice-Hall, Inc. A Simon & Schuster Company Englewood Cliffs, New Jersey 07632, 1995
- [2] https://en.wikipedia.org/wiki/Intelligent_agent
- [3] https://en.wikipedia.org/wiki/Humanoid_robot
- [4] https://en.wikipedia.org/wiki/Autonomous_robot
- [5] <https://www.aldebaran.com/en/humanoid-robot/nao-robot>
- [6] <http://plastics.ulprospector.com/datasheet/e147224/luvocom-1-xcf-30>
- [7] http://ark.intel.com/products/35463/Intel-Atom-Processor-Z530-512K-Cache-1_60-GHz-533-MHz-FSB
- [8] <http://doc.aldebaran.com/2-1/dev/tools/opennao.html>
- [9] <http://doc.aldebaran.com/2-1/naoqi/core/almemory.html>
- [10] <http://doc.aldebaran.com/2-1/naoqi/core/allogger.html>
- [11] <http://doc.aldebaran.com/2-1/naoqi/core/alpreferencemanager.html#alpreferencemanager>
- [12] <http://www.cs.ubbcluj.ro/~rlupsa/edu/retele-2003/c5.html>
- [13] [https://en.wikipedia.org/wiki/Method_\(computer_programming\)](https://en.wikipedia.org/wiki/Method_(computer_programming))
- [14] <http://raganwald.com/2013/04/08/functional-vs-OOP.html>
- [15] Numerical Python, David Ascher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin, Travis Oliphant, The Regents of the University of California, 1999.
- [16] <http://docs.opencv.org/2.4.13/modules/core/doc/intro.html>
- [17] <http://doc.aldebaran.com/2-1/naoqi/audio/altexttospeech.html>
- [18] <http://doc.aldebaran.com/2-1/naoqi/audio/alspeechrecognition.html>
- [19] github.com/larics

