UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

SISTEM DE INTERACȚIUNE PRIN VOCE CU ROBOTUL NAO ÎN LIMBA ROMÂNĂ

LUCRARE DE DIPLOMĂ

Prezentată ca cerință parțială pentru obținerea titlului de Inginer
În domeniul Electronică, Telecomunicații si Tehnologia Informației
Programul de studii Tehnologii şi sisteme de telecomunicaţii

Conducători științifici:
Prof. Dr. Ing. Dragoș Burileanu
Ș.L. Dr. Ing. Horia Cucu

Student:
Georgian Nicolae

București
2016

Anexa 1

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Telecomunicaţii

Aprobat Director de Departament :
Prof. Dr. Ing. Silviu Ciochină

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Nicolae E. Georgian, grupa 441 C

1. Titlul temei: *Sistem de interacţiune prin voce cu robotul NAO în limba română*

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

Acest proiect de diplomă îşi propune realizarea unui sistem complet de interacţiune prin voce cu robotul NAO in limba română ceea ce presupune implementarea pe robot a unui sistem de recunoaştere vocală simultan cu un sistem de sinteză vocală in limba română cât şi înţelegerea întrebărilor de către robot şi generarea unui răspuns corespunzător la aceste întrebări. Realizarea acestui proiect constă in mai multe etape:

- *Implementarea sistemelor de recunoaştere şi sinteză vocală pe calculator*
- *Portarea acestor sisteme pe robot*
- *Implementarea pe robot a unei aplicaţii care să includă aceste sisteme recunoscând în primul rând întrebarea, căutând un răspuns pentru aceasta şi apoi generând clipului audio corespunzător răspunsului*

3. Proiectul se bazează pe cunoştinţe dobândite în principal la urmatoarele 3-4 discipline: *Programarea Calculatoarelor, Structuri de Date şi Algoritmi, Programare Obiect-Orientată*

4. Proprietatea intelectuală asupra proiectului aparţine: *Laboratorului de cercetare Speech and Dialogue (SpeeD) şi studentului Nicolae Georgian*

5. Locul de desfăşurare a activităţii: *Laboratorul de cercetare Speech and Dialogue (SpeeD)*

6. Realizarea practică/ proiectul ramân în proprietatea*: Laboratorului de cercetare Speech and Dialogue (SpeeD) şi studentului Nicolae Georgian*

7. Data eliberării temei: *1.08.2015*

CONDUCATOR LUCRARE:

Prof. Dr. Ing. Dragoş Burileanu

S.L. Horia Cucu

STUDENT:

Nicolae Georgian

DECLARAȚIE DE ONESTITATE ACADEMICĂ

Prin prezenta declar că lucrarea cu titlul ”Sistem de interacțiune prin voce cu robotul NAO în limba română” , prezentată in cadrul Facultații de Electronică, Telecomunicații și Tehnologia Informației a Universitații „Politehnica” din București ca cerință parțială pentru obținerea titlului de Inginer in domeniul Inginerie Electronică și Telecomunicații, programul de studii Tehnologii și sisteme de telecomunicații este scrisă de mine și nu a mai fost prezentată niciodata la o facultate sau instituție de invațământ 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 alta limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alti autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor in vigoare.

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

București, 2016 Georgian Nicolae

CUPRINS

LISTĂ DE FIGURI

LISTĂ DE TABELE

LISTĂ DE ABREVIERI

CISC – Complex Instruction Set Computer

RISC – Reduced Instruction Set Computer

IEEE – Institute of Electrical and Electronics Engineers

WEP – Wired Equivalent Privacy

WPA – Wireless Protected Access

USB – Universal Serial Bus

LED – Light Emitting Diode

FSR – Force Sensitive Resistors

RAV – Recunoastere Automata a Vorbirii

SRV – Sistem de Recunoastere a Vorbirii

HMM – Hidden Markov Model

PLP – Perceptual Linear Prediction

LPC – Linear Predictive Coefficients

MFCCs – Mel Frequency Cepstral Coefficients

WER – Word Error Rate

SER – Sentence Error Rate

ChER – Character Error Rate

GMM – Gaussian Mixture Model

FSG – Finite State Grammar

SPTK – Speech signal Processing Toolkit

FFT – Fast Fourrier Transform

TTS – Text to Speech

ASCII – American Standard Code for Information Interchange

IPA – International Phonetic Alphabet

SAMPA – Speech Assessment Methods Phonetic Alphabet

X-SAMPA – Extended SAMPA

HTS – HMM-based Speech Synthesis System

HTK – The Hidden Markov Model Toolkit

PSOLA – Pitch Synchronous Overlap and Add

RSZ – Raportul Semnal Zgomot

NLP – Natural Language Processing

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIA LUCRĂRII

Formele de comunicare au o istorie la fel de veche ca și apariția vieții pe Pământ, pornind de la primele forme de viață care realizau un schimb de informații prin mijloace chimice ajungând in zilele noastre la forme complexe de comunicație ce se desfășoară prin diferite medii de propagare reușind astfel un transfer al informației dintr-o parte în cealaltă a globului într-un interval de timp de multe ori insesizabil de către om.

Dintre aceste multe mijloace de comunicare cel mai important, după părerea mea, este cel ce se realizeaza prin unde sonore deoarece omul dispune de propriile aparate: aparatul auditiv și aparatul vocal cu ajutorul cărora poate realiza o transmitere bidirecțională a informației. La fel ca orice alte aparate și aparatele de care dispune corpul uman au anumite specificații, de exemplu urechea umană poate auzi sunetele aflate în intervalul de frecvențe de la 20 Hz la 20 KHz, iar aparatul ce se ocupă de prducerea vocii poate produce sunete între 50 Hz și 7.5 kHz .

Crearea unor sisteme care să funcționeze precum aceste aparate ale corpului uman nu este deloc simplă și implică foarte multe studii asupra modelului biologic de funcționare pe baza cărora se dorește crearea unui model matematic care poate aproxima sau, intr-un caz ideal, reproduce aceeași funcție.

Implementarea unor astfel de sisteme conduce la o interacțiune om-mașină mult mai accesibilă pentru utilizatorul de rând iar cea mai importantă problemă în acest caz fiind faptul că

omul sau sistemul nu dispun de o limbă cunoscută comună. Transpunând acest sistem pe un robot cu o înfățișare umană putem obține rezultate remarcabile în majoritatea domeniilor. Mai multe informații despre utilitatea și funcționalitatea unor roboți de tip umandoid se vor detalia în Capitolul 1.

1.2 OBIECTIVUL LUCRĂRII

În această lucrare am urmărit obținerea unui sistem de interacțiune prin voce cu robotul umanoid NAO care să funcționeze pentru limba română. Acest sistem presupune recunoașterea vocală a unor înlănțuiri a unor cuvinte cheie pe care robotul să le interpreteze și mai apoi să execute anumite proceduri asociate fiecărei astfel de secvențe în parte. Acest proiect, față de cele ale colegelor mele Diana Elena Șandru și Ana-Maria Radu, implică o recunoaștere continuă a vorbirii cât și portarea pe robot a unui tool de recunoaștere vocală, în acest caz fiind vorba despre o procesare locală care are un avantaj enorm față de o procesare care are loc pe un server dedicat deoarece se elimină necesitatea unei conexiuni la internet absolut indispensabilă pentru trimiterea fișierelor audio pe un server. Desigur că o procesare locală prezintă și o serie de dezavantaje cum ar fi resursele disponibile, care în acest caz se limitează la cele de care dispune robotul, care impun anumite constrângeri în ceea ce priveste numărul de cuvinte care pot fi recunoscute și tipul de gramatică folosit deoarece ne dorim ca prelucrarea sa fie în timp real, timpul de răspuns să fie cât mai redus. Scopul unei procesări locale este obținerea unor rezultate într-un timp mult mai scurt decât dacă am trimite datele pentru prelucrare pe un server, urmând ca apoi să ne fie returnat rezultatul. Partea de sinteză vocală în limba româna va fi detaliată într-un capitol dedicat.

În acest proiect au fost vizate următoarele obiective:

- Înțelegerea fundamentelor unui sistem automat de recunoaștere vocală și modul în care este creat un astfel de sistem pentru un singur vorbitor, optimizarea acestuia și evaluarea acestuia cu un set de date de la mai mulți vorbitori
- Crearea unui sistem automat de recunoaștere vocală independent de vorbitor și evaluarea acestuia
- Verificarea compatibilității tool-kit-ului pocketsphinx cu sistemul de operare al robotului NAO și investigarea asupra unei metode de portare acestuia pe robot
- Crearea unui dicționar fonetic și a unei gramatici cu stări finite care să permită recunoașterea în mod continuu a vorbirii astfel încât să aibă loc recunoașterea secvențelor cheie și înlăturarea sunetelor și cuvintelor care nu fac parte din acestea
- Crearea unei aplicații pe robot care să interpreteze datele recunoscute de pocketsphinx și executarea anumitor comenzi in funcție de aceste date
- Investigarea metodei de reducere a zgomotului folosită de pocketsphinx, metodă care se bazează existența unui singur semnal audio
- Realizarea unui sistem de sinteză vocală pentru limba română folosind HTS și portarea acestuia pe robot

1.3 SUMAR

În Capitolul 2 vor fi prezentate sumar specificatii ale robotului umanoid NAO incluzând resursele necesare rulării tool-ului și dezvoltării aplicației, Capitolul 3 va aborda problema recunoașterii automate a vorbirii. Vor fi prezentate succint câteva principii ale modelării semnalului vocal, arhitectura unui sistem de recunoaștere automate a vorbirii precum și câteva criterii de evaluare a performanțelor unui SRV. Acest capitol va include și prezentarea grupului de intrumente CMU Sphinx utilizat pentru antrenarea modelului acustic și pentru decodarea care are loc pe robot. Capitolul 4 va detalia modul de realizare practică a unui SRV pornind de la construirea unui sistem pentru un singur vorbitor și ajungând la un sistem independent.

Ulterior va fi tratat subiectul sintezei vocale pentru limba română folosind HMM în Capitolul 5, enumerând concluziile la care am ajuns în urma cercetării efectuate pe marginea acestui subiect. Implementarea aplicației va fi dezvoltată în Capitolul 6, ce conține etapele pe care le-am urmat în construirea aplicației. Ultimul capitol va fi rezervat observațiilor și concluziilor.

CAPITOLUL 2

ROBOȚII UMANOIZI

2.1 ASPECTE FUNDAMENTALE

Cercetările din ultimii ani în domeniul roboticii și mecatronicii au urmărit, prin dezvoltarea roboților androizi și roboților umanoizi, construirea unui robot care să fie prin caracteristicile sale compatibil cu cerințele tehnologiei și ale societății, cerințe ce evoluează neîntrerupt. Astfel, robotul umanoid autonom este acela care are ca model omul, în toată complexitatea sa, ajungând să întâlnească, fizic, prin înfățișare, cât și prin automie, modelul uman și abilitățile umane.

Un dispozitiv care să poată înlocui fizic o persoană, robotul umanoid, apare în mai multe versiuni. Unii dintre roboți pot fi specializați numai în anumite părți ale corpului uman, imitând înfăţişarea acestor părţi, însă alţii pot chiar să înlocuiască fizic o persoană, replicând până şi caracteristicile faciale.

Prin autonomia sa, robotul umanoid se dorește a fi compatibil cu omul prin realizarea unor activităţi cât mai complexe cu putinţă, aproape de imitarea comportamentului uman, însă are în plus autonomia, capacitatea de învăţare, de adaptare, de comunicare şi de interacţiune.

2.2 APLICAȚII ALE ROBOȚILOR UMANOIZI

Utilitatea sa este incontestabilă, atât în economie, robotul umanoid fiind capabil să presteze servicii prin interacțiunea sa cu obiectele, cât și în plan experimental, contribuind în studiul asupra evoluției umane, mişcarea bipedă fiind unul dintre exemplele de studiu.

Serviciile prestate de un robot umanoid, de la relationarea cu clientii, citirea emoţiilor sau folosirea acestuia de către organizaţiile care tratează bolnavii de autism, până la a face sport, a practica yoga şi a se juca cu copiii, sunt toate dezvoltate prin cercetarea în plan experimental. Dacă omul se naşte cu abilitatea de a vedea, de a recunoaşte, de a analiza, robotul este un dispozitiv ce trebuie programat pentru a dezvolta toate aceste abilităţi. În plan experimental, adăugarea unor funcţii pe care robotul umanoid urmează să le realizeze este ca un studiu asupra funcţionării fiinţei umane, ca un model de cercetare în complexitatea propriei naturi.

2.3 REZULTATE OBȚINUTE ȘI PERSPECTIVE DE VIITOR

Interesul în a dezvolta roboții umanoizi crește pe măsură ce aceștia își confirmă capacitatea de a interacționa cu oamenii și de a comunica cu aceștia, capacitatea cognitivă de a își analiza singur mediul și de a acționa în conformitate cu cerințele impuse, acestea fiind caracteristici care reies din autonomia robotului.

In urma evolutiei lor, se consideră că în următorii 20 -30 de ani roboţii umanoizi vor ajunge o prezenţă obişnuită în lumea noastră. In prezent, mersul biped al robotului umanoid prezintă un avantaj, robotul fiind mobil pe terenuri accidentate sau în mediul construit de oameni, poziţionează sistemul vizual la un nivel mai înalt, iar membrele anterioare pot fi folosite pentru manipularea obiectelor. Astfel, roboţii umanoizi pot acţiona butoane, pot ridica obiecte, pot urmări anumite mişcări, pot lovi o minge, deschide uşi sau acţiona vehicule cu ajutorul senzorilor interni şi externi, sau actuatori.

2.4 ROBOTUL NAO

NAO, cel mai vândut robot umanoid din lume, a dovedit prin prezența sa în toate domeniile că activitatea robotică este utilă, fiind îndrăgit pentru dinamismul său şi pesonalizat după dorinţele, imaginaţia şi nevoile fiecăruia. Este creat cu următorul motto, dar deopotrivă şi scop:"să contribuim la bunăstarea omenirii ", aşa cum spune Bruno Maisonnier, fondatorul companiei Alderabaran Robotics. [1]

2.1.1 Caracteristici generale

NAO, robotul umanoid, autonom şi biped este proiectul Aldebaran Robotics folosit în scop de cercetare, educaţie şi entertainment. Proiectul a început în anul 2004, odată cu construirea robotului. Un număr de 6 prototipuri NAO au fost construite între 2005 şi 2007 însă prima versiune a robotului, Nao RoboCup Edition, a fost lansată în martie 2008, urmatoarea versiune, Nao Academics Edition, urmând să fie lansată la sfârşitul aceluiaşi an în universităţi şi laboratoare de cercetare.

În 2010 NAO a devenit un robot umanoid celebru prin prezentarea unui dans sincronizat la Shanghai Expo in China, ceea ce a avut drept rezultat achiziţionarea a 30 de modele de către Universitatea din Tokyo pentru a deveni asistenţi de laborator. În ceea ce a urmat, s-a dorit lansarea unei noi versiuni NAO cu motoare mai performante. Nao Next Gen, ca şi versiune, a fost caracterizată de îmbunătăţiri software constând într-o mai bună calitate a camerelor video, o construcţie mai rezistenta, sisteme anti-coliziune şi creşterea vitezei de deplasare.

NAO Evolution Robot include durabilitate sporită, sinteza vocală în mai multe limbi, recunoaşterea feţelor şi a formelor folosind noi algoritmi, îmbunătăţirea sistemelor de cunoaştere a sursei unui sunet folosind patru microfoane direcţionale. Modelul a fost lansat în iunie 2014.

Începând cu 2011, peste 200 de instituţii educaţionale utilizează robotul umanoid, iar la sfârşitul lui 2014 peste 5000 de roboţi NAO erau deja în uz în scopuri educaţionale şi experimentale în peste 70 de ţări.

În 2015 NAO a fost folosit pentru a demonstra într-un experiment filosofic o formă de bază a conştiinţei de sine, iar într-un alt proiect a fost testat un sistem robotic de”memorie autobiografică” pentru a îi ajuta pe membri Staţiei Internationale Spaţiale sau pe pacienţii în vârstă.

Noua generaţie NAO are un procesor puternic, de 1.6GHz, şi două camere video HD, are 58 de centimetri înălţime, iar după cum se poate observa în Figura 2.1 are o forma umanoidă. Este dotat cu 4 microfoane cu recunoaştere vocală, poate dialoga în 8 limbi şi este dotat cu un senzor de distanţă, două receptoare şi emiţătoare de unde infraroşii, 9 senzori tactili şi 8 senzori de presiune.

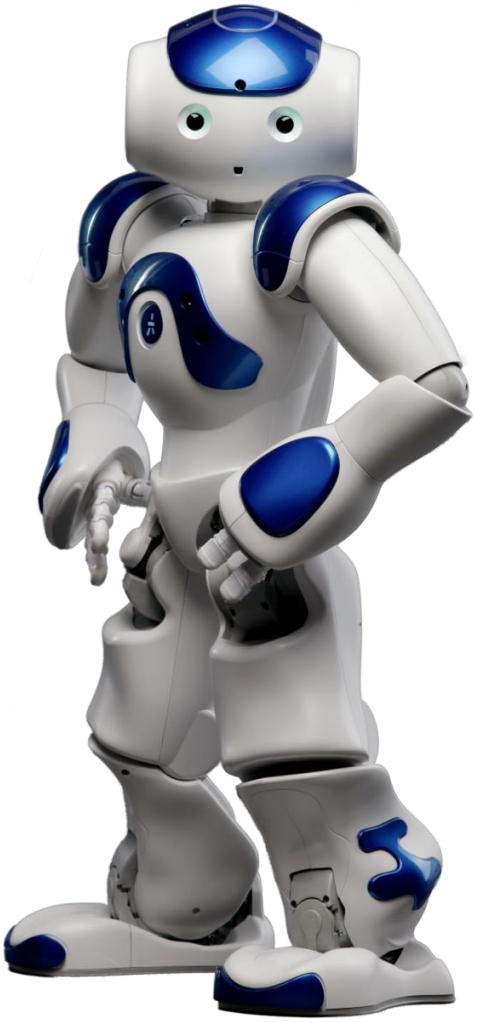

Figura 2.1: Aspect exterior al robotului NAO [1]

Forma sa, alături de capacitatea de a se mişca la 25 de grade şi inerţia construcţiei sale, îi permit să se miste, să îşi menţină echilibrul şi să ştie în ce poziţie se află, dacă este în picioare sau culcat.

2.1.2 Resurse disponibile la nivelul robotului

Cei 574 mm - inaltime, 275 mm - lăţime, 311mm - grosime, aceste dimensiuni susţin o greutate de 4,5kg şi influenţează în mod direct performanţele de mişcare ale robotului, construit din materialele ABS-PC/PA-66/XCF-30.

Este dotat cu o baterie Lithium-Ion ce poate fi încărcată cu un voltaj maxim de 25.2 V, are o energie de 48.6 Wh şi poate fi încărcată în mai puţin de 3 ore.

Versiunea 4 a plăcii de bază prezintă două procesoare principale, unul localizat la nivelul capului, celălalt la nivelul toracelui.

La nivelul capului, este constituit dintr-un procesor Intel x86, de tipul ATOM Z530 și cu memorie cache de 512KB. Viteza de ceas este de 1,6 GHz, pentru setul de instrucțiuni de 32 de biți, microcontrolerul fiind compatibil cu Intel System Controller Hub – Intel® SCH, componentă ce combină funcționalitățile plăcii grafice, ale interfeței procesorului și ale controlerului de memorie cu extensiile de intrare/ieșire, în vederea unui consum cât mai mic de energie.

ATOM Z530 este un procesor cu un singur nucleu, destinat în principal dispozitivelor mobile. Avantajele acestuia sunt consumul de energie redus și faptul că suportă virtualizarea aplicațiilor. De asemenea, acesta este bazat pe microarhitectura Bonnell, fiind astfel capabil să execute până la maxim două instrucțiuni pe ciclul de ceas. Acesta face translația instrucțiunilor complexe (de tip CISC) în operații interne, simple (de tip RISC) înainte de execuție.

Procesorul de la nivelul toracelui este de tipul ARM7TDMI și controlează actuatoarele, distribuind biţii de control ai tuturor microcontrolerelor locale de la nivelul modulelor actuatoare.

ARM7TDMI este un microcontroler cu setul de instrucțiuni pe 32 de biți, de tipul RISC, oferind performanțe ridicate pentru un consum mic de putere. Microcontrolerele locale de la nivelul modulelor actuatoare sunt Microchip 16-biți dsPIC, iar comunicarea între acestea și cel de-al doilea procesor se face prin intermediul a două magistrale de tipul RS-485 la o viteza de 460Kbps.

Procesorul ATOM Z530 1.6 GHz are la dispoziţie o memorie RAM de 1 GB, 2 GB de memorie Flash şi posibilitatea inserării unui card de maximum 8 GB Micro SDHC.

Din punct de vedere al conectivităţii, robotul suportă conexiuni prin două protocoale de comunicație, Ethernet și Wi-Fi. Pentru Ethernet, robotul are la nivelul capului, o mufă de tipul RJ-45, compatibilă cu 3 tipuri de adaptări: 10/100/1000 base T (viteze de 10Mbps, 1000Mbps sau 1 Gbps). Conexiunea prin cablu este cel mai des utilizată pentru setarea conexiunii prin WiFi.[1]

1×RJ45 - 10/100/1000 base T
NAO V5 - IEE 802.11 a/b/g/n
NAO V4 si V3.x - IEEE 802.11 b/g/n
Security: 64/128 bit: WEP, WPA/WPA2.

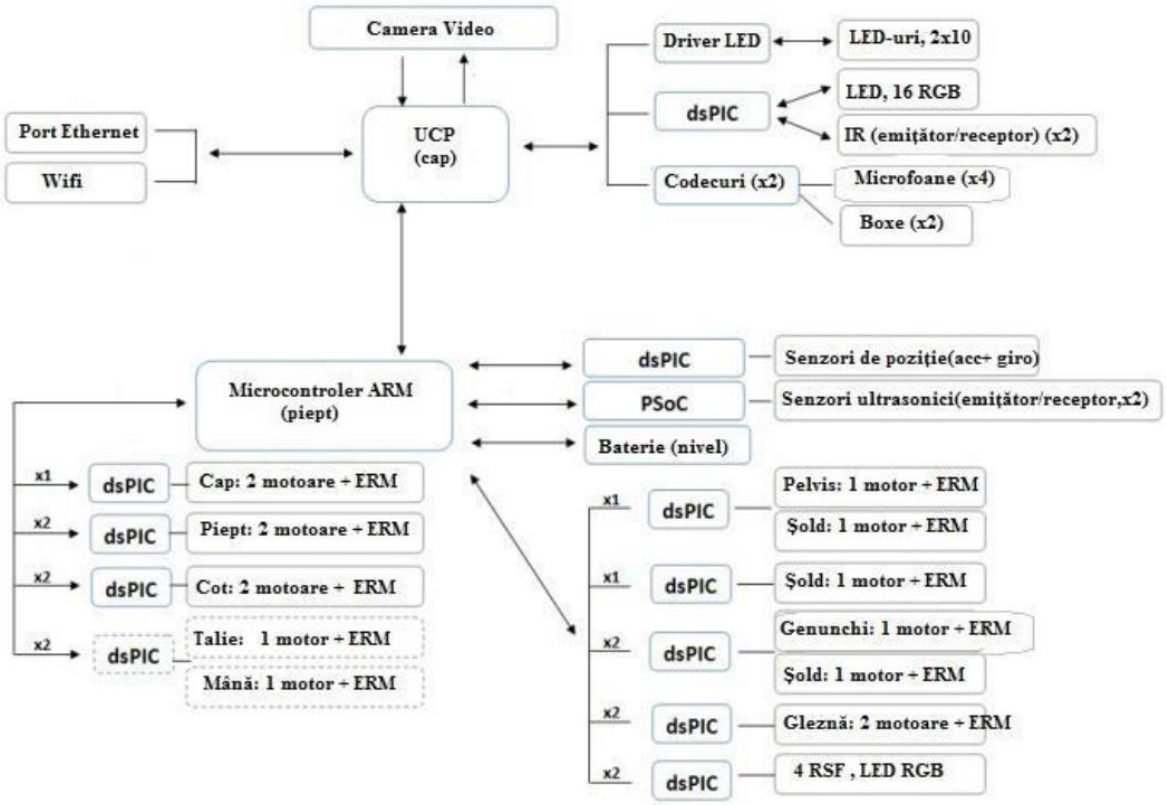

Figura 1.2: Schema de intercomunicare a dispozitivelor de control şi procesare [1]

De obicei, pentru a actualiza sistemul robotului, cel mai des este folosit portul USB la care se pot conecta dispozitive precum Kinect, Asus 3D Sensor sau Arduino.

Pentru ca interacţiunea să fie realizabilă şi la nivel audio, NAO este dotat cu un sistem stereo de difuzare format din doua difuzoare plasate de o parte şi de alta a capului, şi cu 4 microfoane plasate tot pe cap, cu o senzitivitate de la 20mV/Pa +/-3dB până la 1 KHz şi cu o gamă de frecvenţă de la 300 Hz până la 8 KHz. [1]

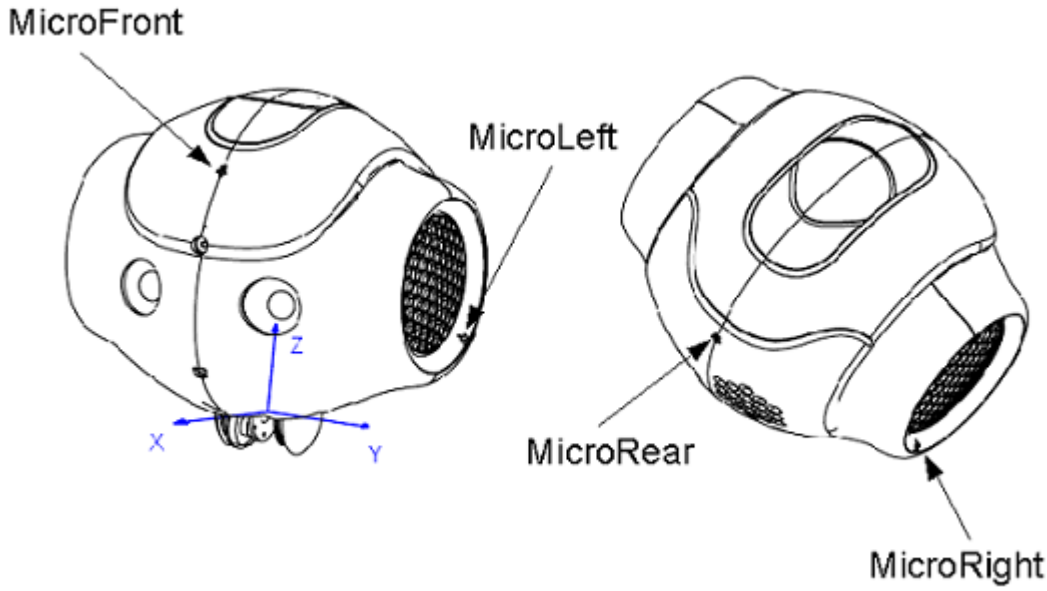

Figura 2.3: Poziţia microfoanelor[1]

În partea din față a capului robotului, sunt localizate două camere video identice, care furnizează imagini cu o rezoluţie de 1280x960 la 30 de cadre pe secundă. Ele sunt necesare pentru identificarea obiectelor ce se află în faţa lui NAO, iar cea de jos este necesară pentru identificarea spaţiului în care robotul se deplasează.

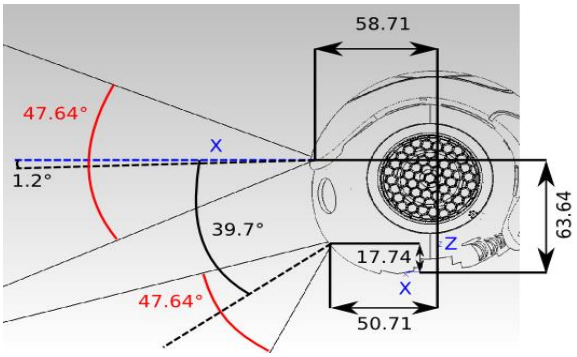

Figura 2.4: Apertura camerelor frontale [1]

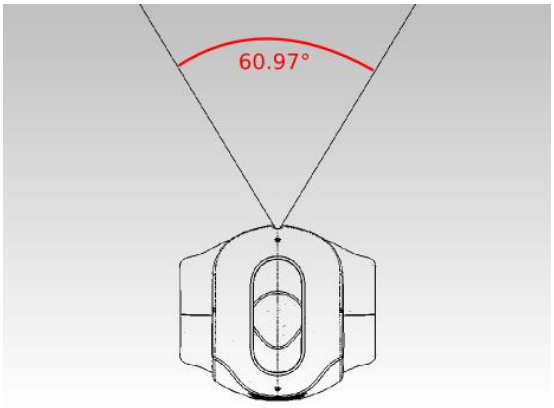

Figura 2.5: Apertura camerei utilizată pentru deplasarea fizică a robotului [1]

Camerele sunt SOC Image Sensor, modelul MT9M114, au rezoluţia de 1.22 Mp şi formatul optic 1/6 inch, mărimea unui pixel este 1.9µm*1.9µm, gama dinamică- 70 dB, semnalul - 37dB(max), responsivitatea - 2.24V/Lux-sec (550 nm), şi un output de 1280*960@30fps. Tipul de obturator este Electronic Rolling shutter (ERS) câmpul vizual - 72.6°DFOV (60.9°HFOV,47.6°VFOV), raza de focalizare de la 30cm până la infinit, iar tipul de focalizare - focus fix. [1]

Robotul prezintă la nivelul său şi un număr de LED-uri localizate conform tabelului de mai jos:

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

Tabelul 2.1 : Inventarul LED-urilor de la nivelul robotului [1]

Figura 2.6: Locația LED-urilor de care dispune robotul NAO [1]

Din punct de vedere al senzorilor mai sus menţionati, cei rezistivi sunt de tip FSR – Force Sensitive Resistors şi măsoară rezistenţa în funcţie de presiunea aplicată. Sunt localizaţi în picioare şi au o rază de acţiune de la 0N la 25N. [1]

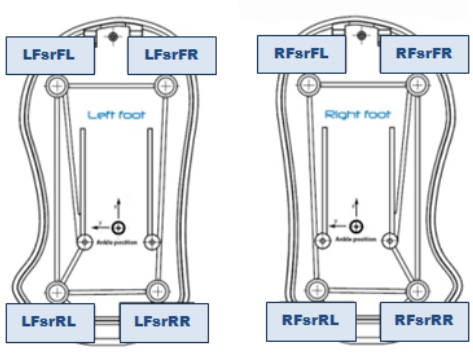

Figura 2.7: Localizarea senzorilor FSR [1]

Girometrul şi accelerometrul reprezintă unitatea inerţială a robotului, girometrul masurând orientarea bazată pe aflarea axei unghiulare dată de 3 axe spaţiale, iar accelerometrul măsoară acceleraţia proprie sau forţă.

Sonarele sunt alcătuite din două emiţătoare şi doua receptoare. Contribuie la estimarea distanţei faţă de obiectele din mediul înconjurător cu o frecvenţă de lucru de 40kHz, o rezoluţie de 1cm-4cm, în funcţie de distanţă, obtinând în final o rază de detectare de 0.20m-0.80m.

Sub 20 cm nu există o informaţie despre distanţă, se ştie doar că un obiect este prezent, iar peste 80 cm valoarea returnată este o estimare.

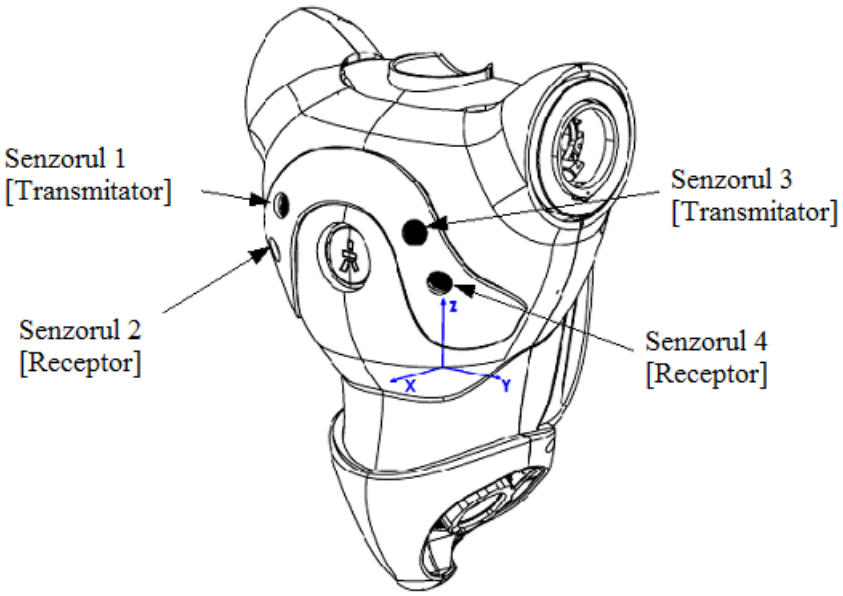

Figura 2.8: Senzorii de tip sonar [1]

Senzori tactili şi de contact sunt localizaţi astfel: trei sezori tactili pe cap, un senzor tactil pe piept, doi senzori tactili pe mâini şi doi pe picioare. Aceştia au rol în a sesiza atingerile asupra robotului. Cei doi senzori de pe mâini şi cei de pe picioare au drept scop protecţia la contactul cu obiectele străine.

Motoarele NAO sunt în numar de 25, plasate la nivelul articulaţiilor, acestea garantând mobilitatea robotului şi emularea cât mai bună a mişcărilor corpului uman. Actuatoarele cu perii de carbon au un cost redus, viteză reconfigurabilă de către utilizator şi generează un cuplu direct de la sursa de alimentare. Acestea sunt posibile datorită comutației interne, a magneților staționari și a electromagneților rotativi.

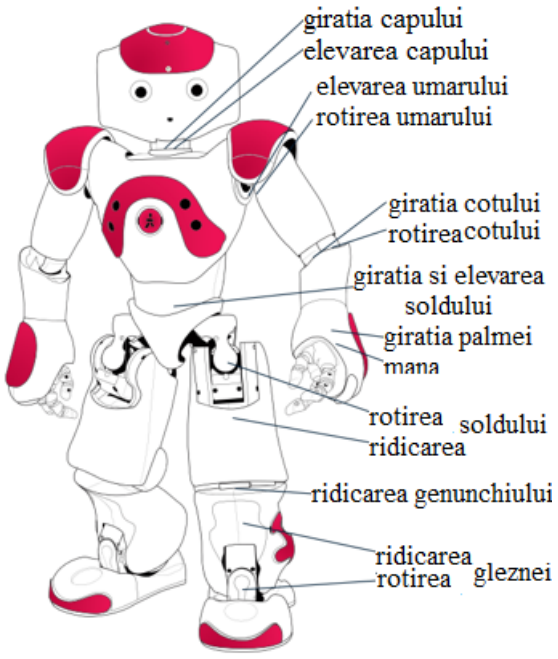

Figura 2.9: Articulațiile active [1]

	Motor tip 1	**Motor tip 2**	**Motor tip 3**
Model	22NT82213P	17N88208E	16GT83210E
Viteza	8 300 rpm ±10%	8 400 rpm ±12%	10 700 rpm ±10%
Cuplu	68 mNm ±8%	9.4 mNm ±8%	14.3 mNm ±8%
Cuplu nominal	16.1 mNm	4.9 mNm	6.2 mNm

Tabelul 2.2: Caracteristicile tipurilor de motoare [1]

Motoarele de tipul 1 sunt puternice şi din acest motiv se folosesc la picioare, celelalte două tipuri fiind utilizate la mâini, articulațiile falangelor și a gâtului, obţinând mişcare precisă şi lentă.

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 este egal cu numărul de grade de libertate al acestuia. Pentru determinarea poziției unui punct material, este nevoie de coordonatele 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.

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

Tabelul 2.3: Dispunerea gradelor de libertate de la nivelul robotului NAO [1]

CAPITOLUL 3

RECUNOAȘTEREA AUTOMATĂ A VORBIRII

Procesul de recunoaştere automată a vorbirii (RAV) poate fi definit ca transpunerea automată şi independentă a unui semnal vocal provenit de la unul sau mai mulţi vorbitori, într-o succesiune de cuvinte în timp real. În cazul în care semnalul procesat conţine informaţii provine de la mai mulţi vorbitori, procesul de recunoaştere automată a vorbirii se împarte în două etape: etapa de diarizare, ce încearcă să distingă numărul de vorbitori independenți şi momentul în care a vorbit fiecare. O altă etapă este cea de trascriere, ce transpune în text mesajul rostit de fiecare vorbitor. În ultimii 50 de ani, cercetarea în acest domeniu s-a concentrat pe dezvoltarea unui sistem capabil să înteleagă vorbirea fluentă; deşi tehnologia actuală este departe de a înţelege pe deplin mecanismul vorbirii, au fost făcute progrese impresionante.

Scopul final al unui SRV este acela de a permite unei maşini să recunoască în timp real, independent de zgomot, de dimensiunea vocabularului, de accentul vorbitorului, cu o acurateţe de 100% toate cuvintele rostite de o persoană. Bazele acestui tip de sistem au fost puse de către Baker, în 1975 prin sistemul DRAGON, ce folosea modele Markov ascunse (HMM). Principalii factori care pot influenţa acurateţea răspunsului sistemlui sunt:

- Zgomotul de fundal;
- Tipul de vorbitor: cunoscut sau necunoscut pentru sistem;
- Caracteristicile mesajului vorbit: accent, dialect, rapiditatea pronunţiei, claritate etc;

- Complexitatea limbii;
- Dimensiunea vocabularului;
- Incertitudinea lingvistică a discursului.

Recunoaşterea automată a vorbirii are o gamă foarte largă de aplicabilitate, întrucât vorbirea reprezintă cea mai facilă şi eficientă metodă de interacţiune umană. Cel mai important dintre acestea este probabil domeniul interfeţelor eyes-free şi hands-free, în cadrul aplicaţiilor ce folosesc comanda vocală. De asemenea, se poate implementa şi în cadrul call center-urilor, prin antrenarea unui sistem care să răspundă automat unor întrebări frecvente, dar şi în traducerile speech-to-speech.

3.1 PRINCIPIILE MODELĂRII SEMNALULUI VOCAL

Limba vorbită este folosită pentru a transmite o anumită informaţie de la un vorbitor la un ascultător. Studiile de specialitate au arătat ca doar pronunţia nu este suficientă, percepţia vorbirii reprezentând un alt element cheie în comunicarea mesajului dorit.

3.1.1 Mecanismul vorbirii

Vorbirea este rezultatul acustic al mişcărilor voluntare ale aparatului respirator şi masticator, reprezentând principala formă de comunicare între fiinţele umane, dar şi dintre om şi maşina. Semnalul vocal este compus din unităţi fundamentale numite foneme, ce constituie baza limbajului vorbit formând silabe şi implicit cuvinte.

Din punct de vedere al conţinutului, un mesaj vocal poate fi caracterizat prin sens conotativ sau strict. Sensul strict conţine informaţia mesajului în timp ce conotaţia constă în intonaţie, astfel obţinându-se un mesaj vocal mai puternic decât cel scris.

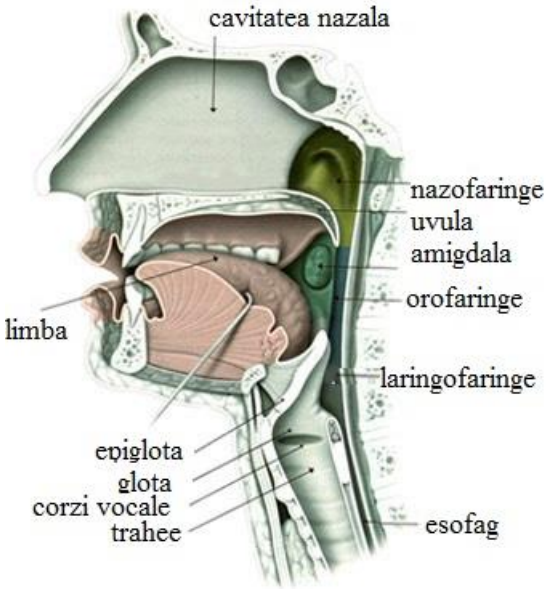

Figura 3.1: Aparatul de vorbire anatomic [16]

Aparatul anatomic de vorbire se situează între plămâni şi buze. Sistemul respirator produce energia necesară vorbirii sub formă de unde de aer la o anumită presiune, iar fluxul rezultat este împins de plămâni spre exterior, traversând traheea, de unde este modulat de laringe înainte să pătrundă în tractul vocal.

Laringele reprezintă un ansamblu de muşchi şi cartilaje mobile care înconjoară o cavitate situată în partea de sus a traheei. Plasate de-alungul laringelui, cele două perechi de corzi vocale manipulează acuitatea şi volumul sunetului generat. Oscilaţia corzilor vocale se poate face în tandem sau individual, ceea ce produce un sunet vocal sau nevocal. Glota este punctul în care corzile vocale converg şi dimensiunea ei este controlată de contracţia acestora, astfel, modulându-se fluxul de aer care tranzitează aria.

Tractul vocal este compus din tractul oral, în compunerea căruia intră gura şi faringele, dar şi din cel nazal. În acest punct aerul este supus unor schimbări, ca în final să ajungă la nivelul gurii. Sunetele sunt de diferite tipuri şi se pot genera doar prin controlul mecanismului de vorbire unde, în esenţă, are loc o succedare de tuburi şi cavităţi rezonante. [6]

3.1.1 Caracteristici fonetice

Mesajul vocal poate fi descompus în sunete scurte şi distincte numite foneme. Fonemele sunt unităţile elementare ale limbajului fonetic şi reprezintă deopotriva unitatea de bază a semnalului de vorbire în care două cuvinte pot fi diferite fizic şi conceptual. Pentru detalierea celor de mai sus, luăm exemplul celor două cuvinte paronime din limba română “glacial” şi “glaciar” unde diferenţa fizică se afla la nivelul celor două litere “l”, respectiv “r”de la sfârşitul cuvintelor menţionate.

Dificultatea în învăţarea unui limbaj sau crearea unui model acustic survine datorită faptului că nu se poate stabili cu rigurozitate o corespondenţă literă-fonem. Totodată, o fonem poate fi reprezentată prin mai multe litere, în timp ce o literă poate corespunde mai multor foneme.

O clasificare standardizată a fonemelor limbii române, dar şi din alte limbaje, nu există; de aceea, în cele ce urmează, va fi descrisă o metodă posibilă. Fonemele se pot diviza în 3 categorii, în funcţie de modul lor de producere pe tractul vocal:

Consoana – definită ca şi sunetul format de închiderea parţial sau totală a tractului vocal uman şi poate fi de mai multe tipuri. Primul tip este consoana nazală, care pentru a fi pronunţată se folosesc deopotrivă atât cavitatea orală, cât şi cea nazală simultan. Al doilea tip, denumit consoana de fricţiune, ce poate fi sonoră sau nesonoră. Cel de-al treilea tip, aşanumitele consoane ‘lichide’, combină sunetele vibrante şi laterale. Ultimul tip este reprezentat de consoana de obstrucţie, pentru a cărei pronunţie aerul este obturat în amonte de laringe în timpul “duratei de retenţie”, după care este eliberat brusc. Şi această categorie se împarte în consoane sonore şi nesonore, unde cele vocale implică concomitent şi procesul de vibrare al corzilor.[13]

Semivocala – sună similar vocalei din punct de vedere fonetic, dar are un caracter nonsilabic, ceea ce o face să fie în general extremitatea silabei şi nu nucleul acesteia.

Vocala – prezintă caracteristici unice de identificare: înălțimea, care depinde e frecvența fundamentală de vibrație a coardelor vocale și timbru, care depinde de conținutul de armonici ce se alătură frecvenței fundamentale. [12]

3.2 ARHITECTURA UNUI SRV

Realizarea unui sistem de recunoaştere automată a vorbirii poate avea diferite niveluri de complexitate în funcţie de scopul final al aplicaţiei (ce tip de vorbire se doreşte a fi recunoscut). Un sistem rudimentar poate recunoaşte doar cuvinte izolate, pe când unul complex poate recunoaşte vorbirea spontană şi continuă.

Sarcina unui sistem de recunoaştere a vorbirii este aceea de a returna o secvenţă de cuvinte când primeşte ca input un semnal acustic. Transformarea vorbirii în text, privită din punct de vedere

statistic se reduce la găsirea secvenţei de cuvinte W* dintr-o limbă L, dacă la input se dă semnalul acustic X.

Semnalul X reprezintă o secvenţă de eşantionare consecutive, în care fiecare index xi este un interval de timp:

$$X= x1,x2, …., xt \quad (3.1)$$

Secvenţa W reprezintă o secvenţă de cuvinte, wi fiind un cuvânt individual:

$$W = w1, w2, …. ,wn \quad (3.2)$$

Pentru a găsi cea mai mare probabilitate, folosim funcţia arg max, ce selectează argumentul care maximizează probabilitatea secvenţei de cuvinte. Deci, cea mai probabilă secvenţă ($\hat{W}$) este cea care are cea mai mare probabilitate aposteriori (P(W|X)), ştiind secvenţa acustică de intrare X.

$$\hat{W} = \arg\max_w P(W|X) \quad (3.3)$$

Folosind criteriul Bayes pentru a calcula probabilitatea aposteriori, ecuația devine:

$$\hat{W}= \arg\max_w (P(X|W)P(W))/(P(X)) \quad (3.4)$$

În ecuația (2.4), termenul P(X) este probabilitatea unei secvenţe acustice de intrare independent de secvenţa de cuvinte W, prin urmare poate fi neglijată. Ecuaţia devine:

$$\hat{W} = \arg\max_w P(W|X)P(W) \quad (3.5)$$

Problema recunoaşterii de vorbire devine mai simplă acum: în loc să estimăm secvenţa de cuvinte, ştiind secvenţa acustică de la intrare, putem estima două mărimi: probabilitatea apriori a secvenţei de cuvinte W (P(W)) şi probabilitatea informaţiei acustice date de secvenţa (P(X|W)).

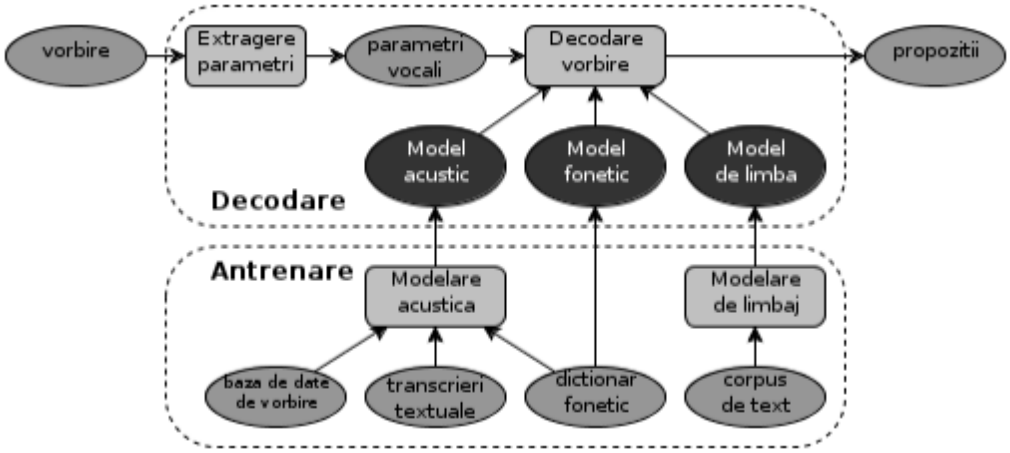

Figura 3.2: Arhitectura unui SRV [2]

Schema unui sistem de recunoaştere automată a vorbirii este prezentată în figura de mai sus. Modelul de limbă estimează probabilitatea ca secvenţa de cuvinte să fie o propoziţie a limbii, iar această probabilitate este folosită de modelul acustic în procesul de decizie. Modelul acustic estimează probabilitatea ca mesajul rosit să fie format dintr-o succesiune sau alta de foneme. Fonemul reprezintă unitatea de sunet fundamentală în limbile vorbite ce diferenţiaza cuvintele între ele. Acest model introduce nişte constrângeri datorită faptului că în orice limbă unele secvenţe de foneme apar mai des decât altele.[2]

Aceste modele pot fi construite separat, după cum se poate vedea în figura de mai sus, dar pot funcţiona şi împreună pentru a decoda o anumită secvenţă acustică de intrare în baza ecuaţiei (2.5).

Pentru a face legătura între modelul acustic şi modelul de limbă, se foloseşte un alt treilea tip de model, fonetic, ce reprezintă un dicţionar de pronunţie ce atribuie fiecărui cuvânt secvenţa adecvată de foneme.

3.2.1 Modelarea Acustică

Modelul acustic (MA) poate fi considerat nucleul unui SRV, fiind cel mai important element în sarcina de recunoaştere. Acesta foloseşte modele acustice antrenate în prealabil şi parametrii secvenţei acustice de intrare pentru a decide ce fonem a fost rostit de către vorbitor.

MA este compus dintr-un set de modele fonetice care în tipmul procesului de decodare acestea sunt tranformate în cuvinte, iar ulterior cuvintele se leagă formând o secvenţă de cuvinte. Acest tip de abordare generativă poate fi implementată cu succes folosind modelele Markov ascunse (HMM). HMM-urile sunt automate probabilistice cu număr finit de stări care pot fi combinate în mod ierarhic pentru a construi modele pentru secvenţe de cuvinte din modele pentru unităţi de vorbire mai scurte. [1]

HMM-urile nu modelează semnalul acustic de intrare; după cum se poate oberva în figura 11, este necear un bloc de extragere a parametrilor pentru a calcula o serie de vectori de parametrii pe baza cărora se realizează procesul de modelare Acustică.

Cei mai folosiţi parametri din domeniul de recunoaştere vocală sunt:

- Coeficienţii Liniari Predictivi (Linear Predictive Coefficients -LPC) – bazaţi pe un model liniar de generare a semnalului vocal [2]
- Coeficientii Mel Cepstrali (MFC- Mel Frequency Coefficients)
- Parametri Perceptuali obţinuţi prin Predicţie Liniară (PLP – Perceptual Linear Prediction) - proprietățile sistemului auditiv sunt simulate prin diferite aproximări. Spectrul semnalului vocal care rezultă din aceste aproximări este modelat cu un model “numai poli” autoregresiv. [3]

Coeficienții MFC au fost introduşi încă din anii ’80 şi prezintă un avantaj major faţă de coeficienţii spectrali, acela de a fi necorelaţi unul de celălalt.

Sistemul auditiv al omului percepe sunetul într-un mod logaritmic, parametrii acustici nefiind distribuiţi liniar pe scala frecvenţelor. Din acest motiv, MFCC-urile se bazează pe tranformata cosinus liniară a logaritmului spectrului de putere pe scala de frecvență Mel. În primul rând, cepstrul este calculat pe baza Transformatei Fourier Inversă a logaritmului spectrului semnalului acustic. Cepstrul este o tranformare omomorfă, deci convoluţia celor două semnale în domeniul timp este egal cu suma lor în domeniul cepstral.[4]

Principala diferenţă dintre coeficienţii cepstrali şi coeficienţii Mel este dată de relaţia de mai jos:

$$\mathrm{f}_{\mathrm{Mel}} = 2595 \log_{10}(1 + \frac{f}{700}) \tag{3.6}$$

În figura de mai jos, este prezentat modelul în care coeficenţii MFCC sunt calculaţi; Semnalul acustic este trecut printr-o fereastră de tip Hamming, după care este aplicată Transformata Fourier Discretă pentru a obţine spectrul. Scala Mel este împărţită în două benzi de frecvenţă egale, iar fiecărei benzi îi corespunde un filtru trece bandă triunghiular. Spectrul de puteri se poate mapa acum pe scala Mel şi logaritmul puterii este calculat la fiecare frecvenţă Mel. Ultimul pas este aplicarea Transformatei Cosinus Discrete, fiecărei puteri logaritmate. MFC reprezintă amplitudinile spectrului final.

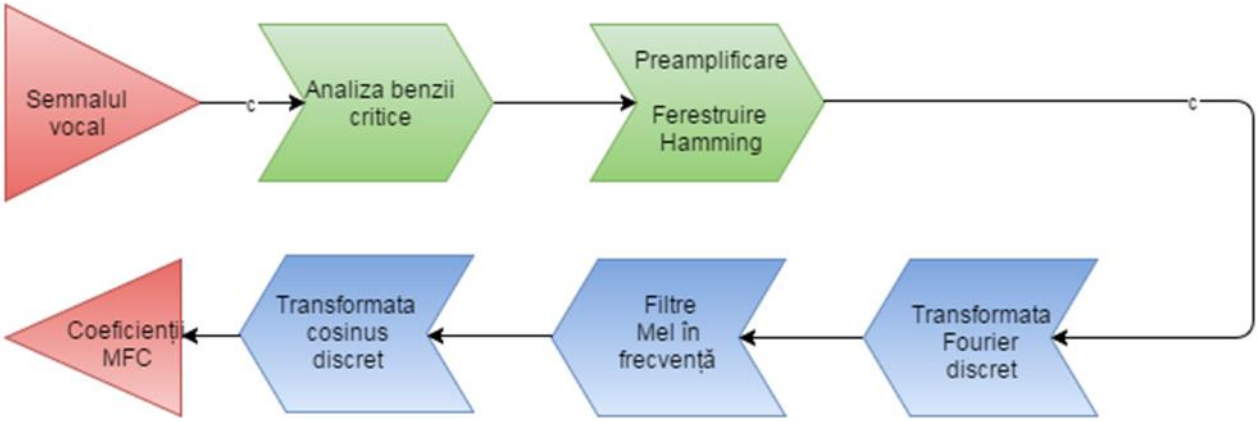

Figura 3.3: Obținerea parametrilor MFCC

De obicei, după finalizarea calculelor, se extrag între 12 şi 20 de coeficienţi. Un SRV foloseşte în mod normal un vector de parametri cu 39 de dimensiuni, compus din cei 12 coeficienţi MFC, energia lor şi derviatele lor de ordinul întâi şi doi.

3.2.2 Modelarea lingvistică

Modelul de limbă este o parte foarte importantă a sistemului de recunoaştere vocală deoarece introduce restricţii suplimentare pornind de la premisa că nu toate secvenţele de cuvinte sunt propoziţii posibile ale limbii. Aceste restricţii pot include un set de reguli de gramatică sau informaţii statistice privind secvenţele de cuvinte, reducând astfel numărul de combinaţii posibile.

Acest model este folosit în timpul procesului de decodare a secvenţei acustice de intrare, având rolul de a estima probabilităţile tuturor secvenţelor de cuvinte din spaţiul de cutare. În esenţă, scopul este de a estima cât mai precis posibil dacă o anumită secvenţă de cuvinte este o propoziţie validă a limbii. Mai mult decât atât, acest proces se traduce în a lua decizii în ceea ce priveşte ordinea cuvintelor, anexând probabilităţi astfel încât succesiunea firească a cuvintelor să fie mai uşor recunoscută.

Una dintre posibilele abordări este să calculăm probabilitatea P(W) pe baza frecvenţei cu care o secvenţă de cuvinte W apare, proces dezavantajos din punct de vedere al calculelor. O altă abordare posibilă, pe care am şi folosit-o în aceasta teză, constă în determinarea probabilităţii identificării unei secvenţe de cuvinte, calculând probabilitatea fiecărui cuvânt, pe baza unui număr de cuvinte anterioare, de unde şi nomenclatura sugestivă de Model N-gram.

Probabilitatea unei secvenţe $W = w_1, w_2,.., w_n$ poate fi exprimată astfel:

$$P(W) = P(w_1, w_2,.., w_n) = P(w_1)P(w_2 \mid w_1)...P(w_n \mid w_1, w_2,.., w_{n-1}) \quad (3.7)$$

Conform ecuaţiei de mai sus, sarcina estimării probabilităţii de cuvinte W este împărţită în sarcini mai mici care estimează probabilitatea fiecărui cuvânt, cunoscându-se probabilitatea secvenţei anterioare. Termenul N din denumirea N-gram indică numărul de cuvinte anterioare ce afectează probabilitatea cuvântului curent. Acest număr nu este ales aleator, fiind corelat cu cantitatea de date utilizată pentru antrenare. Cu cât avem o cantitate mai consistentă de informaţie de antrenare, cu atât gramul N al modelului de limbă creşte.

Cel mai frecvent folosit N-gram este trigram-ul, ceea ce presupune că probabilitatea cuvântului curent să depindă de două cuvinte anterioare. În vederea obţinerii unui model N-gram viabil, este necesară stabilirea numărului de apariţii a fiecărei combinaţii de N-cuvinte în cadrul unui corpus de text suficient de mare. De exemplu, estimarea probabilităţii unei secvenţe wi, wk, wj se exprimă matematic astfel:

$$P(w_i | w_k, w_j) = \frac{numarul\ (wi, wk, wj)}{\sum_w numarul\ (wi, wk, wj)} \quad (3.8)$$

3.3 EVALUAREA PERFORMANȚELOR UNUI SISTEM DE RECUNOAȘTERE AUTOMATĂ A VORBIRII

Indicatorul cel mai reprezentativ în evaluarea performanţelor unui sistem dedicat recunoaşterii vocale constă în *rata de eroare la nivel de cuvânt*. La acest parametru se adaugă şi puterea de predicţie a unui Model de Limbă prin măsurarea probabilistică a asocierii de cuvinte de test, în consecinţă, un Model de Limbă bine definit ar trebui să atribuie o probabilitate mare în cazul unei predicţii corecte, respectiv, o probabilitate mică în caz contrar. Aşanumita *perplexitate* evaluează pe o secvenţă de cuvinte predefinită asociind o valoare mare în cazul unui grad de corectitudine mare şi se poate corela facil cu rata de eroare la nivel de cuvânt amintită anterior.[2]

La antrenarea sistemului se folosește un set de cuvinte sau fraze cumulate într-un corpus de antrenare. Tot ceea ce nu este inclus în acest cumul, reprezintă *cuvinte în afara vocabularului* şi nu vor putea fi prezise. Aceste cuvinte îngreunează evaluarea întrucât produc o perplexitate infinită ce nu poate fi adunată la perplexităţile celelalte pentru a nota întreaga secvenţă. Pentru a diminua acest dezavantaj, se predefineşte un procent de cuvinte în afara Modelului de Limbă pentru a nu se îngreuna calculul matematic din spatele logicii implementate. În această logică survin erori de tip I si II care corespund prin rejectarea unei ipoteze false, respectiv unei ipoteze fals pozitive.

Criteriile WER şi SER realizează analiza mesajelor transcrise textual. Rata de eroare la nivel de propoziţie se calculează ca raport între numărul de propoziţii eronate (propoziţii ce conţin cel putin o greşeală) şi numărul total de propoziţii, pe când rata de eroare la nivel de cuvânt se calculează ca raportul dintre suma tuturor erorilor şi cuvintele din transcrierea de referinţă. De asemenea există şi Cher (rata de eroare la nivel de caracter) care este similară cu WER, numai că acesta raportează eroarea la nivel de caracter şi nu de cuvânt.

Rata de eroare la nivel de propoziţie (SER - sentence error rate): raportul între numărul de propoziţii eronate (propoziţii ce conţin cel puţin o greşeală) şi numărul total de propoziţii din transcrierea de referinţă:

$$SER[\%] = \frac{\#\ Propoziţii\ eronate}{\#\ Propoziţii\ in\ transcrierea\ de\ referinta} * 100 \quad (3.9)$$

Rata de eroare la nivel de cuvânt (WER – word error rate):

$$WER\ [\%] = \frac{\#Erori\ de\ inserţie + \#\ erori\ de\ substituţie + \#erori\ de\ ştergere}{\#\ cuvinte\ în\ transcrierea\ de\ referinţă} * 100 \quad (3.10)$$

Rata de eroare la nivel de caracter (ChER- Character error rate):

$$ChER[\%] = \frac{\#\ erori\ de\ inserţie\ de\ caracter + \#erori\ de\ substituţie\ de\ caracter + \#erori\ de\ ştergere\ de\ caracter}{\#caractere\ în\ transcrierea\ de\ referinţă} * 100$$

(3.11)

În concluzie, evaluarea unui sistem cu aplicabilitate în domeniul recunoaşterii informaţiei vorbite este necesară din moment ce pentru antrenare se folosesc corpusuri, limitate, în esenţă. Măsurarea performanţelor ajută şi la clasificarea comparativă a sistemelor cu acelaşi scop.[4]

3.4 CMUSPHINX

CMUSphinx reprezintă un grup de instrumente software ce cuprinde mai multe sisteme de recunoaștere a vorbirii, dezvoltate de Universitatea Carnegie Mellon. Acestea includ serii de module de recunoaștere de vorbire (Sphinx 2-4), dar includ și un modul de antrenare al modelului acustic (SphinxTrain).

Cele mai reprezentative module din pachetul Sphinx sunt următoarele:

- Sphinx- este primul sistem de recunoastere a vorbirii continue de recunoastere a vorbirii continue dezvoltat, ce folosește Modele acustice Markov Ascunse(HMM) și un model statistic de tip N-gram, lansat in 1986. Totuși, nu mai este folosit în ziua de azi, fiind înlocuit de versiunile ulterioare, ce oferă rezulate mult mai bune.
- Sphinx 2 -este un modul de recunoaștere rapid, orientat către performanță, lansat în anul 2000. Sphinx 2 se focusează pe recunoașterea în timp real, specifică aplicațiilor de limbă vorbită, principala lui utilizare fiind în sistemele de dialog. Acesta folosește reprezentarea semi-continuă pentru modelarea acustică, adica un singur set de Gaussiene pentru toate modelele.
- Sphinx 3 - este utilizat cu precădere pentru recunoașterea de mare acuratețe, dar nu în timp real. Acest model folosește reprezentarea continuă a HMM
- Sphinx 4- este o rescriere completă a Sphinx cu scopul de a aduce un framework mai flexibil pentru cercetarea în recunoașterea vorbirii, scrisă în întregime în limbajul de programare Java.

  Principalele direcții de dezvoltare la momentul actual sunt:

  - Dezvoltarea unui nou modul de antrenare al modelului acustic
  - Implementarea adaptării la vorbitor
  - Îmbunătățirea managementului de configurație
- PoketSphinx - Este o versiune de Sphinx ce poate fi utilizată în sisteme embedded. PocketSphinx se află și el în plin proces de dezvoltare, intrucât poate fi implementat cu ușurință pe dispozitive mobile. [17]

CAPITOLUL 4

REALIZAREA UNUI SISTEM DE RECUNOAŞTERE AUTOMATĂ A VORBIRII

În prezent, recunoaşterea vocală este o carcateristică disponibilă pe din ce în ce mai multe dispozitive: telefoanele, tabletele cât şi alte dispozitive răspund la comenzile rostite de către utilizator. Aceasta câştigă popularitate şi specialiştii preconizează că în viitorul apropiat, toate interacţiunile dintre om şi maşina se vor realiza prin comandă vocală.

Din păcate, un SRV necesită o putere de calcul ridicată şi din acest motiv încă nu poate fi dezvoltat pe orice tip de dispozitiv. Robotul NAO are implementat un SRV pentru limba engleză, dar sarcina de a recunoaşte comenzi şi în limba română este una destul de dificilă, întrucât necesită anumite trăsături pe care Nao nu le are.

Nao a fost construit iniţial ca un robot destinat cercetării, dar ultimele sale versiuni au fost îmbunătăţite pentru a putea fi folosit şi cu alte scopuri; motto-ul companiei este "un NAO în fiecare casă". Robotul va fi o interfaţă foarte plăcută între oameni şi case inteligente, iar următorul pas ar fi conectarea sa la sistemul de operare ce controlează dispozitivele din casă.

Bineînţeles, primul pas este programarea lui Nao pentru a executa anumite instrucţiuni în urma unei comenzi vocale în limba română.

4.1 CONSTRUIREA UNUI SISTEM PENTRU UN SINGUR VORBITOR

În realizarea unui SRV, varibilitatea între vorbitori reprezintă o problemă importantă. De aceea, în faza incipientă a dezvoltării proiectului de diplomă am realizat un sistem de recunoaştere dependent de vorbitor: antrenarea şi evaluarea s-au realizat folosind fişiere audio înregistrate cu vocea mea. Dezavantajul unui astfel de SRV este că pentru fiecare nou vorbitor trebuie antrenat un nou model acustic, ceea ce îl face extrem de complex din punct de vedere al datelor necesare pentru antrenare.

4.1.1 Etape de realizare şi optimizare a proiectului

Primul pas în realizarea unui sistem de recunoaştere a vorbirii pentru limba română s-a constituit în construirea unui SRV simplu, ce are ca sarcină recunoaşterea unor secvenţe de cifre. Vocabularul necesar creării unui astfel de sistem este redus, întrucât acesta conţine doar cifre: zero, unu,..., nouă.

În vederea realizării acestui sistem , s-au succedat următoarele etape:

1) Înregistrarea unei baze de date de clipuri audio

Pentru realizarea unui model acustic al unui sistem de recunoaştere automată a vorbirii, este necesară în primul rând o bază de date de clipuri audio înregistrare. Această bază de date a constat în înregistrarea a 100 de clipuri audio, fiecare conţinând o secvenţă de 12 cifre aleatoare.

Înregistrarea propriu-zisă a fişierelor audio am realizat-o cu ajutorul unui microfon conectat la calculator, pe server-ul de înregistrări al laboratorului SpeeD (Speech and Dialogue), unde au fost generate automat cele 100 de secvenţe a câte 12 cifre.

2) Crearea dicţionarului fonetic şi a listei de foneme

Cum am menţionat şi mai sus, dicţionarul fonetic are rolul de a conecta modelul acustic cu modelul de limbă. Acesta trebuie să conţină toate cuvintele necesare realizării sarcinii de recunoaştere pe care ne-o propunem, în cazul meu, recunoaşterea cifrelor din limba română, împreună cu transcrierile fonetice ale acestor cuvinte.

Dicţionarul fonetic pentru acest prim proiect conţine doar cele zece cifre ale sistemului zecimal: zero, unu, doi, trei, patru, cinci, şase, şapte, opt şi nouă. Sistemul de recunoaştere a vorbirii continue CMU Sphinx utilizează dicţionare fonetice în format text (.txt). Readactarea acestuia trebuie să respecte anumite regului: pe ficare rând va fi scris un singur cuvânt, alături de descrierea sa fonetică, fonemele fiind separate prin spaţiu. Pentru a creşte şi mai mult performanţele sistemului, este recomandată utilizarea de foneme specifice fiecărui cuvânt. Spre exemplu, pentru cifra cinci, care conţine de două ori fonemul "i", linia corespunzătoare din dicţionarul fonetic va arăta astfel:

cinci c_cinci1 i_cinci1 n_cinci c_cinci2 i_cinci2

Se poate observa că fonemul "i" va fi modelat diferit în funcţie de poziţia pe care o are în cuvânt.

Pentru realizarea dicţionarului propriu-zis, m-am conectat la server-ul de dezvoltare al laboratorului SpeeD şi am creat un director de lucru temporar în care am creat şi am editat un fişier text ţinând cont de regulile impuse de CMU Sphnix.

Fişierul rezultat este următorul:

zero z_zero e_zero r_zero o_zero

unu u_unu1 n_unu u_unu2

doi d_doi o1_doi i3_doi

trei t_trei r_trei e1_trei i3_trei

patru p_patru a_patru t_patru r_patru u_patru

cinci c_cinci1 i_cinci1 n_cinci c_cinci2 i_cinci2

şase s1_şase a_şase s_şase e_şase

şapte s1_şapte a_şapte p_şapte t_şapte e_şapte

opt o_opt p_opt t_opt

nouă n_nouă o1_nouă w_nouă a1_nouă

Următorul pas a constat în crearea unui fişier care să conţină toate fonemele ce urmează să fie modelate, respectând, bineînţeles, formatul impus de CMUSphinx de a avea un singur fonem pe fiecare linie. Am adăugat, pe lângă fonemele utilizate pentru transcrierea cifrelor, şi fonemul "SIL" ce are ca scop modelarea zonelor de linişte.

3) Configurarea procesului de antrenare

Am început prin a crea un nou proect CMU Sphnix, numit "rodigits", în care am adăugat resursele necesare: cele 100 de clipuri audio înregistrate în prealabil. În alt director, numit "etc", am copiat dicţionarul fonetic, împreună cu lista de foneme, transcrierea textuală a cifrelor pronunţate în cele 100 de fişiere audio şi dicţionarul cu elemente acustice ce nu reprezintă foneme. Ulterior am adăugat şi scripturile "createFileIds.sh" şi "createTranscriptions.sh" ce transpun lista tuturor clipurilor audio înregistrate, precum şi transcrierile lor textuale în formatul necesar procesului de antrenare.

Pentru antrenarea modelului am utilizat 80 dintre clipurile audio, creând o listă ce conţine primele 50 şi ultimele 30 de clipuri audio, păstrând resul 20 pentru evaluarea sistemului. În acelaşi mod am împărţit şi transcrierile textuale corespunzătoare.

Am accesat fişierul de configurare, modificând locaţia resurselor necesare în procesul de antrenare, dar şi valorile implicite ale parametrilor modelului acustic. Apoi am creat 80 de fişiere cu coeficienţi cepstrali, corespunzătoare celor 80 de clipuri audio de antrenare.

4) Antrenarea modelului acustic

Pentru a porni procesul de antrenare, am rulat comanda "sphinxtrain_biosinf run" în terminal.

5) Crearea modelului de limbă

Sistemele de recunoaştere de vorbire cu vocabular mare utlizează în principiu modele de limbă statistice de tip N-gram. Totuşi, sarcina de recunoaştere a cifrelor are un vocabular redus pentru care nu se pretează modelele de limbă statistice.

Pentru antrenarea acestui model, am ales să implementez o gramatică cu stări finite, ce reprezintă un model de tip graf în care nodurile reprezintă cuvinte ale limbii, iar tranziţiile între cuvinte sunt transpuse în arcele grafului. Cifrele din cadrul clipurilor audio înregistrate apar cu probabilităţi egale, iar modele de tip gramatică cu stări finite (FSG) sunt mai potrivite.

În figura de mai jos este prezentată grafic gramatica cu stări finite folosită pentru sarcina de recunoaştere a cifrelor. Modelul conţine 14 noduri dintre care 10 reprezintă cuvinte ale limbii, cifele de la zero la nouă.

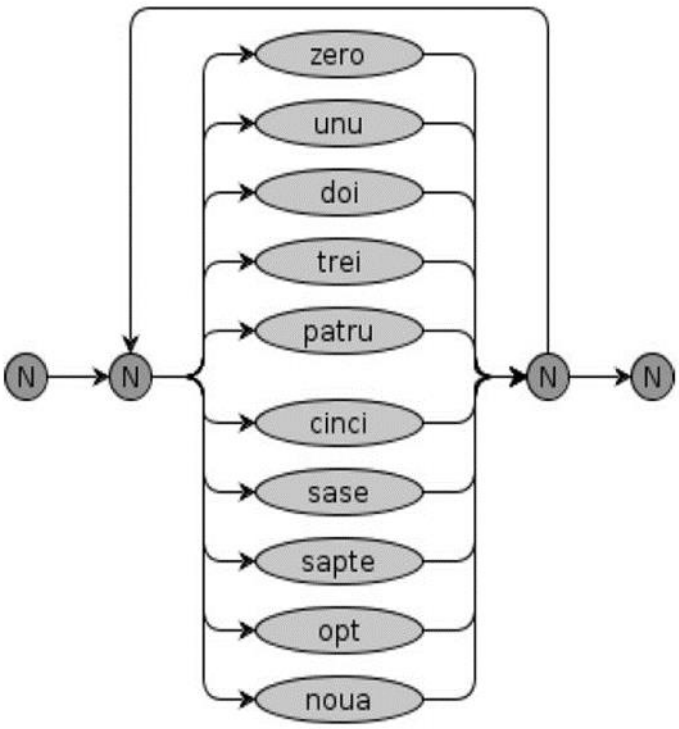

Figura 4.1: Gramatica cu stări finite a sarcinii de recunoaştere rodigits [2]

6) Evaluarea Sistemului

Pentru a porni procesul de evaluare, se impune în primă fază configurarea procesului de decodare, proces care se realizează în mod asemănător cu cel antrenare: am accesat fişierul de configurare, doar că în acest caz am modificat parametrii ce specificau locaţia resurselor necesare evaluării.

Ulterior, am creat listele cu coficienți MFC specifice clipurilor de evaluare şi am pornit procesul rulând comanda:

/usr/local/sphinx/lib/sphinxtrain/scripts/decode/slave.pl

În urma execuţiei, am obţinut pentru acest model o valoare a erorii la nivel de cuvânt WER=3.7%, ceea ce înseamnă că dintr-un total de 240 de cuvinte evaluate, sistemul antrenat a recunoscut corect 231.

7) Optimizarea Sistemului

După decodarea sistemului, rezultatul obţinut, cu o eroare de 3.7% nu este unul foarte bun, aşa că în mod natural, următorul pas a fost încercarea de a îmbunătăți acest rezultat, diminuând eroarea pe cât posibil.

Modelul acustic creat de CMUSphinx este construit în mod implicit cu unităţi de vorbire dependente de fonem de tip trifonem (fonem căruia i se precizează vecinii din stânga şi din dreapta); fiecare trifonem este implementat că un HMM cu trei stări emisive, având o distribuţie de probabilitate de ieşire implementată cu un GMM. Un astfel de model de stare poartă denumirea de senonă şi poate fi comună mai multor trifoneme. În mod implicit, într-un proiect Cmu Sphinx, numărul de senone este configurat la 200, iar cel de densități gaussiene de probabilitate la 8.

Pentru a îmbunătăţi modelul acustic, am rulat o serie de teste, modificând numărul de GMM per senone, folosind 1,2 sau 4 densităţi Gaussiene. Rezultatele obţinute sunt prezentate mai jos.

Nr densităţi	WER [%]
1	15
2	9,1
4	6,2
8	3,7

Tabelul 4.1: WER pentru 200 de senone pentru un singur vorbitor

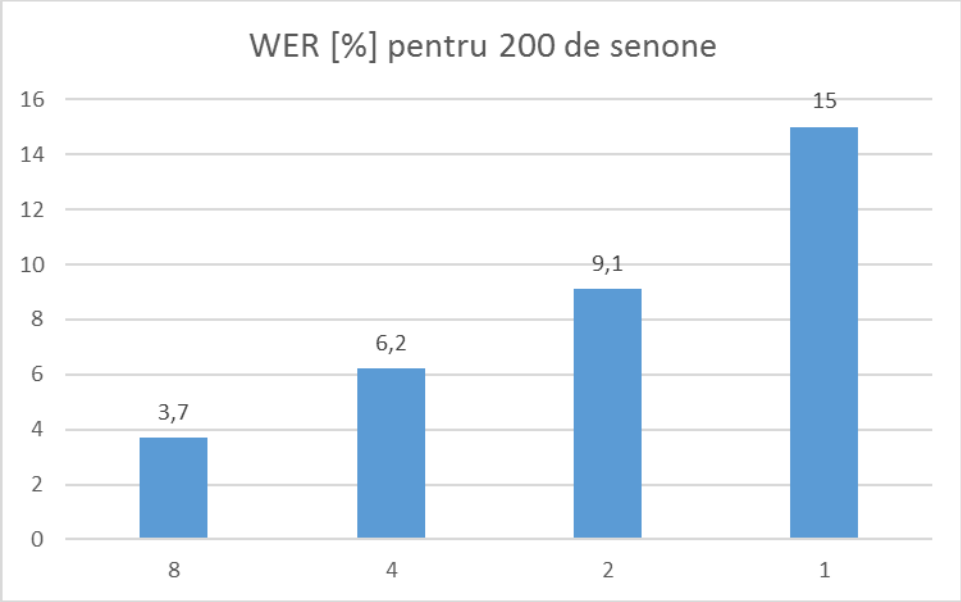

Figura 4.2: WER pentru 200 de senone (un singur vorbitor)

Din grafic se poate observa cu uşurinţă că dacă scădem numărul de densităţi gaussiene per senone, rata de eroare creşte, deci nu are loc o îmbunătăţire a rezultatelor obţinute.

Ulterior, am rulat aceleaşi teste, modificând numărul de senone de la 200 la 100. Rezultatele obţinute sunt figurate în cele ce urmează.

Nr densităţi	WER [%]
8	0
4	0,4
2	0,4
1	1,2

Tabelul 4.2: WER pentru 100 de senone pentru un sigur vorbitor

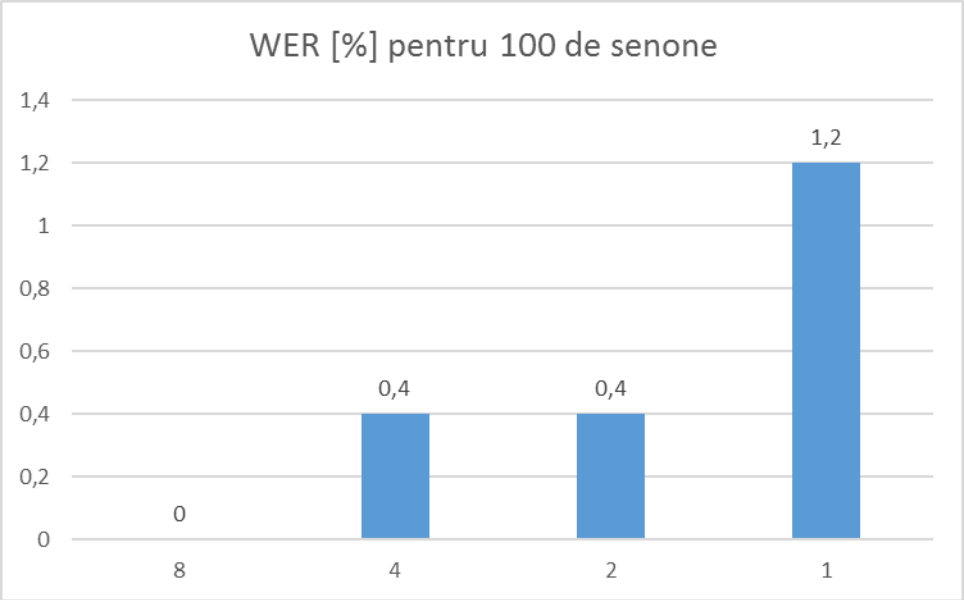

Figura 4.3: WER pentru 100 de senone (un singur vorbitor)

Se poate observa că pentru 100 de senone, rezultatele obţinute s-au îmbunătăţit semnificativ faţă de cele obţinute anterior. În cazul în care decodarea se face folosind 8 densităţi gausiene şi 100 de senone, sarcină de recunoaştere a unei secvenţe de 12 cifre se realizează cu o acurateţe de 100%, sistemul recunoscând toate cele 240 de cuvinte destinate evaluării.

Următorul pas a fost testarea sistemul folosind clipuri audio înregistrate de către alţi vorbitori. Rezultatele sunt prezentate în tabelul de mai jos.

Vorbitor	WER [%]
vorbitor nr 50	100%
vorbitor nr 51	98,30%
vorbitor nr 52	97,10%
vorbitor nr 53	99,60%
vorbitor nr 54	91,50%
Medie	97,50%

Tabelul 4.3: WER pentru vorbitori necunoscuţi folosind un sistem antrenat pe un singur vorbitor

Se observă că eroarea este foarte mare, sistemul recunoaşte doar câteva cuvinte în funcţie de diferenţele dintre voci, ceea ce este absolut normal având în vedere că modelul a fost antrenat pentru un singur vorbitor. Testarea s-a realizat folosind sistemul cu cele mai bune performanţe : cu 8 densităţi gaussiane per senone şi 100 de senone.

4.3 Construirea unui sistem independent de vorbitor

Deşi rezultatele obţinute anterior au fost foarte bune pentru vocea mea, atunci când l-am testat pe alţi vorbitori, rata de eroare a fost foarte mare, depăşind 90%. Întrucât doresc să dezvolt un sistem pe care să-l poată folosi oricine, fără să fie nevoie să antrenez un nou model acustic pentru fiecare nou utilizator, următoarea sarcină abordată s-a constituit în crearea unui sistem de recunoaştere automată a vorbirii independente de vorbitor.

Pentru a crea un sistem independent de vorbitor, am creat un nou proiect CMU Sphinx, numit rodigitsIndep, după care am urmat aceaşi paşi ca şi la sistemul anterior cu o singură diferenţă majoră.

În loc să folosesc doar baza de date formată din clipurile audio înregistrate de mine, am utilizat toată baza de date disponibilă pe serverul de dezvoltare de la SpeeD, ce conţinea câte 100 de clipuri audio înregistrate de utilizatori diferiţi. Am antrenat modelul acustic folosind 80 de fişiere de la 74 de vorbitori, restul de 20 de fişiere le-am păstrat pentru a fi folosite în procesul de decodare.

Am antrenat modelul acustic până la 128 de densităţi gaussiene per senone mai întâi pentru 200, apoi pentru 100 de senone în vederea optimizării acestuia. Rezultatele obţinute sunt prezentate mai jos.

Nr. densităti Gaussiene per senone	WER [%] 200 senone	WER [%] 100 senone	Scădere relativă WER [%]
1	22,20	16,90	
2	14,80	11,50	31,95%
4	11,40	7,10	38,26%
8	8,90	5,30	25,35%
16	8,90	4,40	16,98%
32	8,40	3,80	13,64%
64	7,90	3,40	10,53%
128	7,70	3,10	8,82%

Tabelul 4.4: WER pentru un sistem indepenent de vorbitor evaluat pentru vorbitori cunoscuţi

În tabelul de mai sus se poate observa scăderea relativă a ratei de eroare la nivel de cuvânt cu creşterea numărului de densităţi gaussiene per senone. Rezultatul optim este obţinut atunci când sunt folosite 128 de densităti de probabilitate per senone şi 100 de senone.

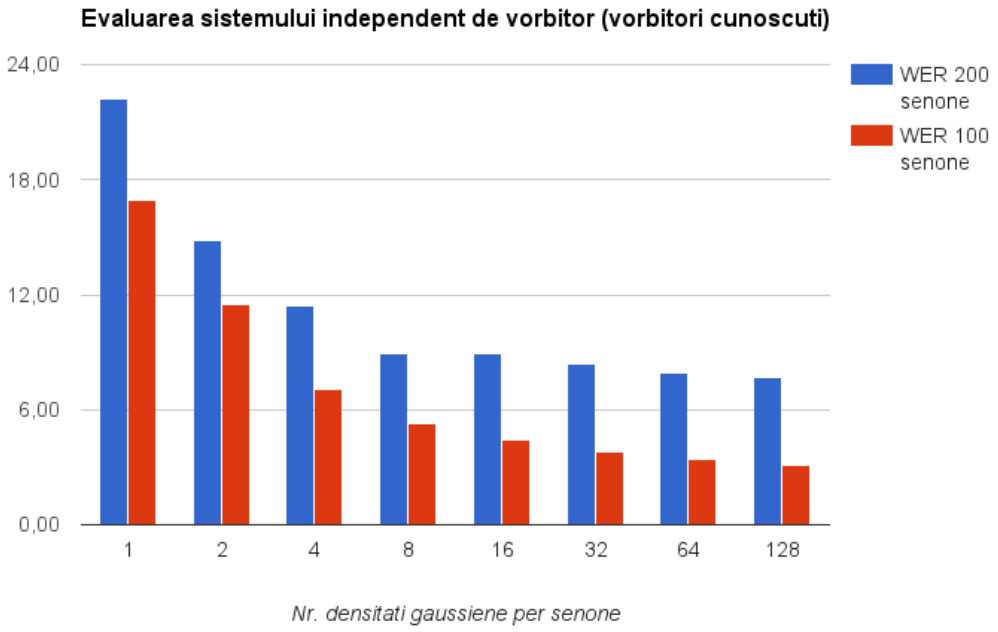

Figura 4.4: Evaluarea sistemului independent de vorbitor (pentru vorbitori cunoscuţi)

În cazul numărului de densităţi Gaussiene observăm o tendinţă descrescătoare a WER care începe să se satureze, scăderile relative calculate în tabelul 6 confirmând acest aspect. Din acest motiv, consider că o scădere relativă de sub 8 % a WER nu ar justifica o creştere a densităţilor gaussiene până la 256.

Vorbitor	WER [%]
370	91,70
385	4,60
374	4,60
226	4,60
88	4,60
369	4,20
389	3,80
118	3,80
48	3,80
381	3,30
380	3,30
394	2,90

Tabelul 4.5: Evaluarea sistemului independent de vorbitor (pentru vorbitori cunoscuţi)

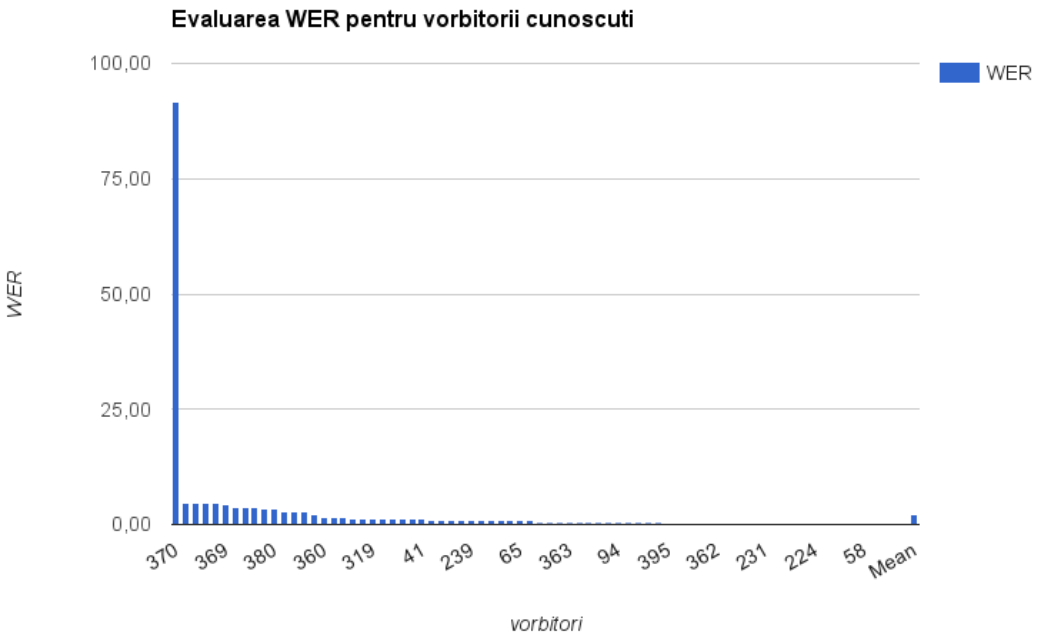

Figura 4.5: Evaluarea WER a sistemului independent de vorbitor per vorbitor (pentru vorbitori cunoscuţi)

Din graficul şi tabelul de mai sus, se observă că WER se încadrează în limitele normale, pentru toţi vorbitorii în afara celui cu nr. 370, având în vedere că sistemul a fost antrenat pentru aceşti utilizatori.

Vorbitorul nr. 370 are un WER neobişnuit de ridicat şi ulterior am investigat cazul acestuia pentru a descoperi neregulile ce au condus la acest rezultat. Am ascultat clipurile audio înregistrate de acesta şi am descoperit că rosteşte o singură cifră într-un fişier audio. În concluzie vom renunţa la acest vorbitor şi vom antrena din nou modelul acustic. Am şters din fişierele rodigits.fileids.train şi rodigits.transcription.train liniile corespunzătoare vorbitorului nr. 370 apoi am rulat încă odată procesul de antrenare. După terminarea antrenării am şters şi liniile corespunzătoare vorbitorului nr. 370 şi din fişierele de test rodigits.fileids.test şi rodigits.transcription.test pentru a obţine o medie a WER corectă. După rularea procesului de decodificare am obţinut un WER de 2%, deci o scădere relativă cu 35 % a erorii

Următoarea etapă de optimizare a constat în testarea acestui model acustic folosind pentru decodare clipuri audio provenind de la vorbitori necunoscuţi, adică alţi vorbitori faţă de cei folosiţi în procesul de antrenare.

Vorbitor	251	252	253	256	259	260	261	262	263	264	265	266	267	268	269
WER[%]	2,1	4,2	0	0,8	0	0,4	0,4	0,4	0	0,4	1,3	0	2,1	2,5	0

273	274	275	276	300	108	227	45	377	211	223	210	387	55	374	382	43
0	0,4	1,7	3,3	2,9	17,9	14,2	11,7	11,3	6,30	7,10	7,50	6,30	5,80	5,40	6,70	5,80

Tabelul 4.5: Evaluarea WER a sistemului independent de vorbitor (pentru vorbitori necunoscuţi)

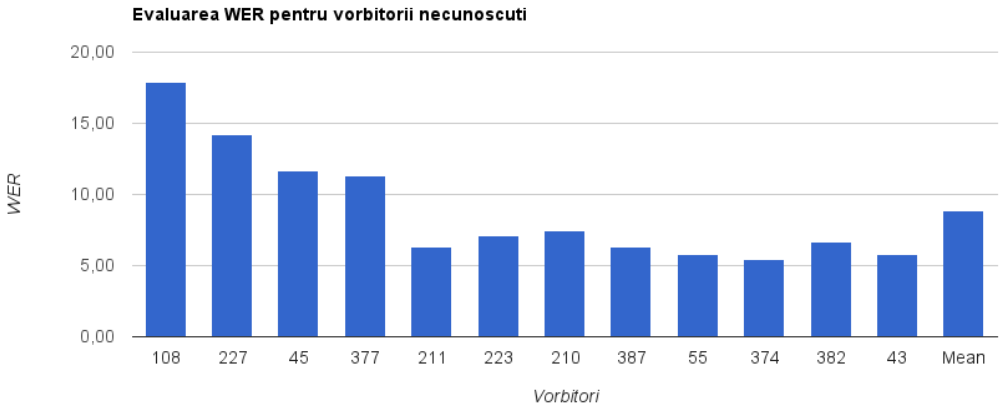

Figura 4.6: Evaluarea a sistemului independent de vorbitor WER pentru vorbitori necunoscuţi

În cazul în care decodarea s-a făcut cu înregistrări audio de la vorbitori necunoscuţi, s-a înregistrat o creştere a WER pentru unii vorbitori, iar pentru alţii sarcină de recunoaştere s-a executat fără erori. După ce am testat modulul pe 32 de vorbitori necunoscuţi, am obţinut o medie a ratei erorii la nivel de cuvânt de 8,83%, rezultat pe care îl consider mulţumitor.

4.4 Rezultate şi Concluzii

Am implementat cu succes un sistem de recunoaştere automată a vorbirii optimizat la maxim pentru baza de date pe care am avut-o la dispoziţie, capabil să recunoască o secvenţă de cifre cu o eroare relativ scazută.

Rezultate mai bune se pot obţine mărind numărul de vorbitori pentru care facem antrenarea (la un moment dat se va ajunge la saturaţie că şi în cazul numărului de densităţi gaussiene).

Pentru a obţine rezultate semnificative calitatea fişierelor audio trebuie să fie bună - să nu prezinte zgomot, pronunţia trebuie să fie cât mai exactă atât în cazul fişierelor de antrenare cât şi în cazul fişierelor de evaluare.

Cunoscând modul în care se antrenează și se optimizează un sistem, timpul care trebuie acordat pentru colectarea unei baze de date de fisiere audio pentru ca sistemul sa functioneze cu un minim de eroare pe cât mai multe tipuri de voci, și timpul necesar realizării unui număr suficient de antrenări, eliminând pe rând fișiere corupte din baza de date pentru a obține un rezultat satisfăcător am optat pentru folosirea unui model acustic pentru limba română antrenat în prealabil ce mi-a fost pus la dispoziție de catre Laboratorul SpeeD.

CAPITOLUL 5

ASPECTE ASUPRA DEZVOLTĂRII UNUI SISTEM DE SINTEZĂ VOCALĂ ÎN LIMBA ROMÂNĂ

5.1 PERSPECTIVE

Pentru realizarea unei interacțiuni cât mai naturale între om și mașină, este necesar ca sistemele de sinteză vocală să aibă abilitatea de a genera un semnal vocal cu o voce arbitrară și diferite stiluri de vorbire: vesel, trist, cu diferite accente și transmițând diferite emoții.

În acest scop, în ultimele decenii, mai multe metode de sinteză au fost propuse. Înainte de anii ’90, metode bazate pe un anumit format și set implicit de reguli erau cel mai des întâlnite. Aceste metode foloseau fonemele ca unități fundamentale și modelau după anumite reguli fiecare fonem în parte. După anii ’90, au fost abordate metode de sinteză bazate pe concatenarea segmentelor de vorbire extrase dintr-un corpus. Corpusul reprezintă o bază de date de clipuri audio ce conțin cuvinte înregistrate. Aceste metode generau un semnal vocal concatenând unități de vorbire din baza de date.[8]

După 1995, a fost propus modelul statistic, ce poate fi antrenat automat folosind date ce contin vorbire. Modelul permite și personalizarea vocii prin modificarea anumitor parametri. Din acest motiv, sinteza vocală ce folosește HMM ca model statistic devine populară la jumătatea anilor ’00.

Deşi transformarea unui text scris în vorbe rostite stârneşte, de multă vreme, interesul programatorilor, traducerea procedurii în aplicaţii practice a fost relativ restrânsă. Metoda în sine este folosită pentru învăţarea limbilor străine şi permite oamenilor care nu pot vorbi să comunice prin telefon. La ora actuală, se pare că a fost brevetat chiar un sintetizator care traduce gândurile în vorbe, conform publicației online New Scientist[4], în articolul intitulat Invention: Thought-controlled voice synthesizer, al autorului Barry Fox, în iulie 2005.

Firma Call Center Technology oferă, printre altele, un program de transformare a cuvintelor rostite în date ce pot fi interpretate de către calculator printr-un program de recunoaștere a vocii interlocutorului și un program de sinteză vocală pe baza unui text deja existent. Dezvoltarea explozivă din ultima vreme a acestor sisteme a cunoscut acest proces datorită evoluției mecanismelor de marketing bazate pe comunicații și pe tehnologia informației, alături de însuși domeniul de call center, așadar în aceste servicii se tinde la înlocuirea factorului uman cu roboți capabili de sinteză și recunoaștere vocală. Astfel, o bază de date bine pusă la punct, un corpus, poate reduce personalul unei astfel de companii și pentru câteva mii de euro se poate achiziționa un sistem funcțional 24/7.

Sintezele se clasifică în două tipuri: HW și SW. Cele HW nu transformă textul în formatul de date .mp3 sau .wav ci îl citesc direct, fiind niște dispozitive independente. În fază incipientă au apărut proiecte disponibile publicului larg, precum: WinTalker, TextAloud, Narrator, Jaws, Eloquence Software, sau chiar Expresivo care deja avea să reprezinte un pas evolutiv și nu era orientat doar pe limba engleză. În prezent, sinteza vocală a ajuns la un statut atât de dezvoltat încât se dorește emularea de accente și calitatea constă în naturaleţea redării.[5]

Analiza lingvistică necesară conversiei unui text în limbaj vorbit presupune separarea acestuia în unități lexicale: cuvinte, silabe, foneme, ca mai apoi să se recurgă la conversia lor în semnale sonore combinate ulterior în vorbire. Metodele cele mai utilizate se bazează pe principiul PSOLA, în care procesarea vorbirii se realizează în domeniul timp. Câștigă teren și metodele pe bază de corpus, care folosesc cantități mari de date achiziționate în urma rostirii naturale a unui text.

Instrumentele utilizate în analiza și prelucrarea semnalului vocal sunt: Festival împreună cu suita de instrumente din pachetul demo disponibil online gratuit. Am ales acest mediu de dezvoltare întrucât este unul de uz general orientat pe construcția sistemelor de sinteză vocală și vine insoțit de module de exemple variate. Ca și întreg, pune la dispoziție API-uri complexe TTS precum: interpretor de comandă și schemă sub formă de librărie C++, extensii JAVA și interfață EMACS. Festival este multilingvistic, deși modelul de limbă engleză este cel mai complet.

Extensia FESTVOX face construcția unei voci sintetice în mod sistematic și bine documentat, facil de utilizat de către orice utilizator. Se bazează pe două tehnici noi de sinteză: MULTISYN și HTS. Pentru utilizare necesită o mașină virtuală UNIX. Este compatibil și oferă suport CMUSphinx și SPHINXTrain-ului în construcția modelelor acustice și etichetarea lor.

Un alt instrument ultizat este SPTK care vine și el insoțit de o suită de instrumente dedicate mediilor UNIX, precum analizoare LPC, PARCOR, LSP și filtre de sinteză. Acestea, alături de tehnicile de cuantizare au fost atent selecționate de către profesorii universităților științifice din Tokyo. Codurile sursă inițiale au fost redactate de un grup de participanți la activitățile de cercetare asupra grafurilor, procesării de date, FFT și eșantionare.

Alte instrumente avansate de lucru sunt: SpTool, folosit pentru proiectarea de filtre digitale, Dsp Tool, utilizat pentru prelucrarea digitală de semnal, inclusiv semnal audio, Neural Network Tool – pentru proiectarea şi testarea sistemelor bazate pe reţele neuronale, Statistic Tool – pentru calcule statistice, etc.

Aplicaţia dispune de un limbaj de nivel foarte înalt ce respectă principiile programării structurate, cu o sintaxă asemănătoare limbajului C. Operaţiile de bază ale limbajului sunt operaţii cu matrici, specificate prin variabile de tip vector ce dispun de redimensionare automată. Operaţiile asupra vectorilor se realizează folosind funcţiile ce sunt puse la dispoziţia utilizatorului prin intermediul unei biblioteci matematice extinse.

Prelucrarea numerică a semnalului vocal cuprinde toate metodele de operare directă asupra semnalului, în cadrul proceselor de filtrare, codificare și compresie. A analiza un semnal vocal presupune identificarea parametrilor semnalului pe baza eșantioanelor de vorbire înregistrate de la vorbitor, care vor fi utilizați ulterior în aplicația dedicată de sinteză sau recunoaștere.

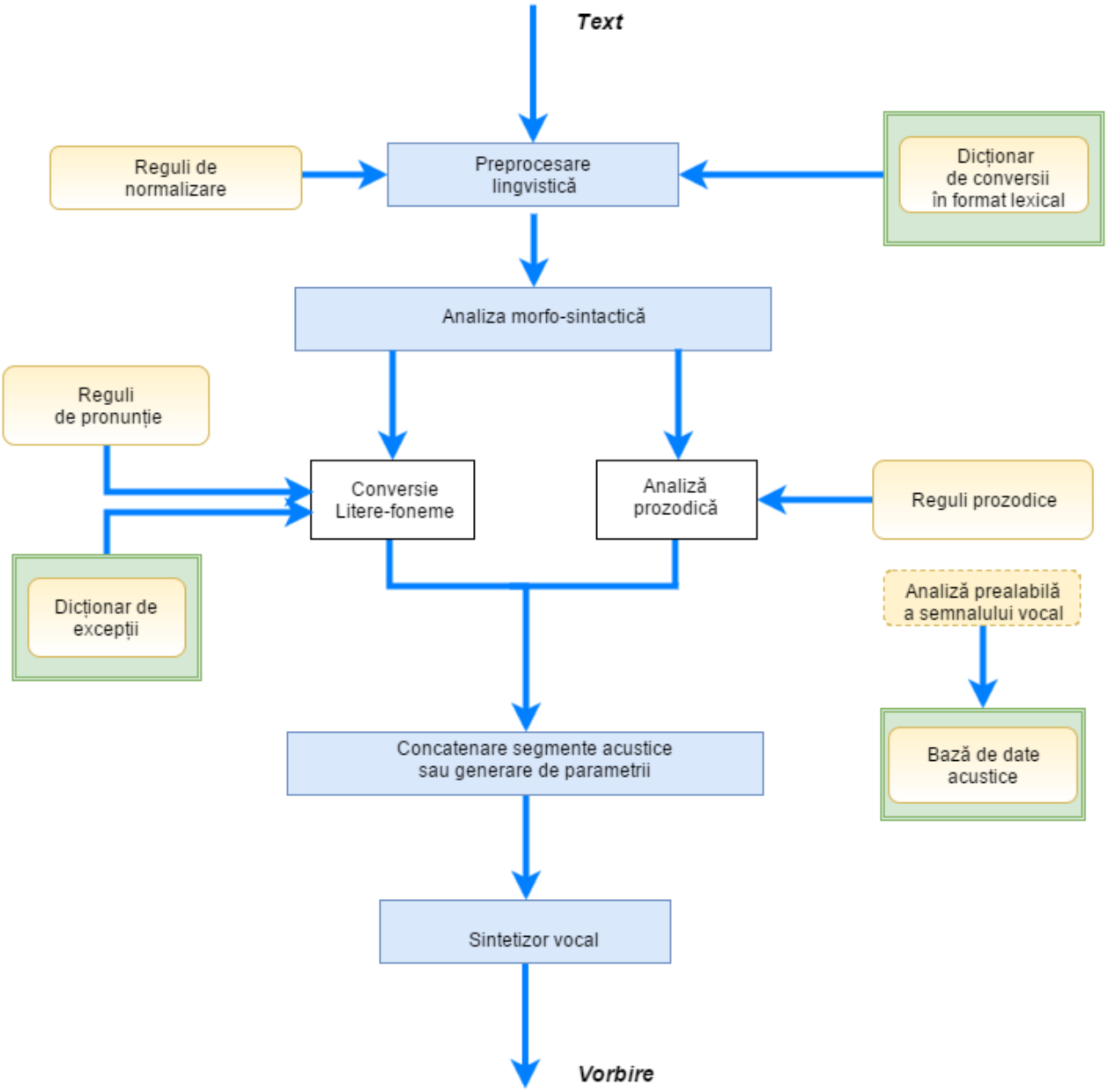

Figura 5.1: Scehma bloc a unui sistem TTS [12]

Scopul unui sistem de sinteză vocală este de a transforma un mesaj scris într-o anumită limbă într-un semnal audio care apoi poate fi redat prin intermediul unei aplicații. Un astfel de sistem poarta denumirea de sistem “text to speech” (TTS) a căruischemă bloc este figurată mai sus.

Preprocesarea lingvistcă – se referă la transcrierea sub formă de text a informațiilor inițiale care nu sunt scrise cu foneme, ci folosind caractere de tip cifră sau simbol. Ele nu formează în mod direct unități lexicale, ci abrevieri ale acestora. Din categoria carcaterelor tip cifră sau simbol fac parte și carcaterele speciale precum: „+”, „-”, „:”, „/”, „<”, „>”, „=”.

Abrevierile sunt și ele împărțite pe categorii: unități de măsură (m, g, l, Hz, V), pefixe reprezentând subunitați de măsură (n, m ,c ,d, da, K, M), grade universitare (asist., ș.l, conf., prof.),

grade militare (serg., plut., mai., căp., col., gen.), dar și alte abrevieri exprimând localitate, scară, bloc etc.

Cele mai importante și cele mai des folosite sunt cele numerice, iar explicitarea sub formă fonematică, necesită supunerea unor reguli lexicale speciale, ce țin cont de poziția unei cifre în cadrul numărului pentru a o denomina unei categorii verbale.

Analiza morfo-sintactică – reprezintă un criteriu de calitate în sinteza vorbirii, întrucât cea mai mică greșală de sintaxă în textul inițial va duce la o sinteză eronată, ceea ce pretinde un efort din partea auditoriului de recreare mentală a cuvântului corect, cu prețul atenției față de cuvintele ulterioare, sau chiar pierderea sensului întregii propoziții.

Pentru a evita acest scenariu, se recurge la proiectarea unui astfel de analizor prin două metode de bază: construirea unui vocabular complet al limbii, sau folosirea unor reguli sintactice de specificare a unor condiții excepționale. Prima metodă este una exhaustivă cu dezavantajul efortului de construire a vocabularului, care trebuie să conțină și forme flexionare ale cuvintelor (rădăcină, terminații, declinare, conjugare). A doua metodă, oroentată pe regulile de identificare a formei de bază a cuvintelor, verifică corectitudinea lor printr-un dicționar, cu dezavantajul gradului de necompletitudine.

Conversia litere-foneme – are drept scop transcrierea fonetică standardizată a fonemelor corespondente textului numite grafeme, pe baza unui alfabet de simboluri fonetice. De exemplu, pentru limba română în dicționarele fonetice putem găsi trascrierea pe baza alfabetului IPA, alfabet ce nu este întocmai potrivit pentru a facilita o prelucrare automată facuta de calculator deoarece multe dintre simbolurile folosite nu un corespondent în tabelul ASCII . Astfel, s-au dezvoltat alfabete fonetice care să încurajeze prelucrarea de către sistemele de calcul, de examplu: SAMPA și X-SAMPA.

Analiză prozodică – reprezintă determinarea accentului, intonației și ritmul adecvate unității fonetice. Regulile prozodice se aplică la nivel segmental (în interiorul cuvintelor), dar și la nivel suprasegmental (în propoziție).

Aceste reguli oferă corecția la nivelul parametrilor: amplitudine, durată, frecvență, în funcție de tipul propoziției stabilită în etapa de procesare lingvistică, de topica cuvintelor, dar și cea a silabei sau fonemului.

Tendințele de actualitate, țin cont tot mai mult de prozodia emoțională pentru a emula starea vorbitorului (încântare, melancolie etc).

Generarea parametrilor acustici – este etapa care preia simbolurile fonetice obținute în urma transcrierii fonetice și a modificărilor prozodice și le generază într-o secvență de parametri corespunzători textului de sintetizat.

Cunoștințele din baza de date vocală se modifică în urma alterărilor prozodice și se efectuează o concatenare în vederea obținerii unui întreg contextual.

Sintetizatorul vocal – efectuează transformarea succesiunii de parametri în formă de undă ce poate fi redată audio prin intermediul unui dispozitiv dedicat. Fonemele de undă din secvența de parametri se obțin prin aplicare de reguli de sinteză. [12]

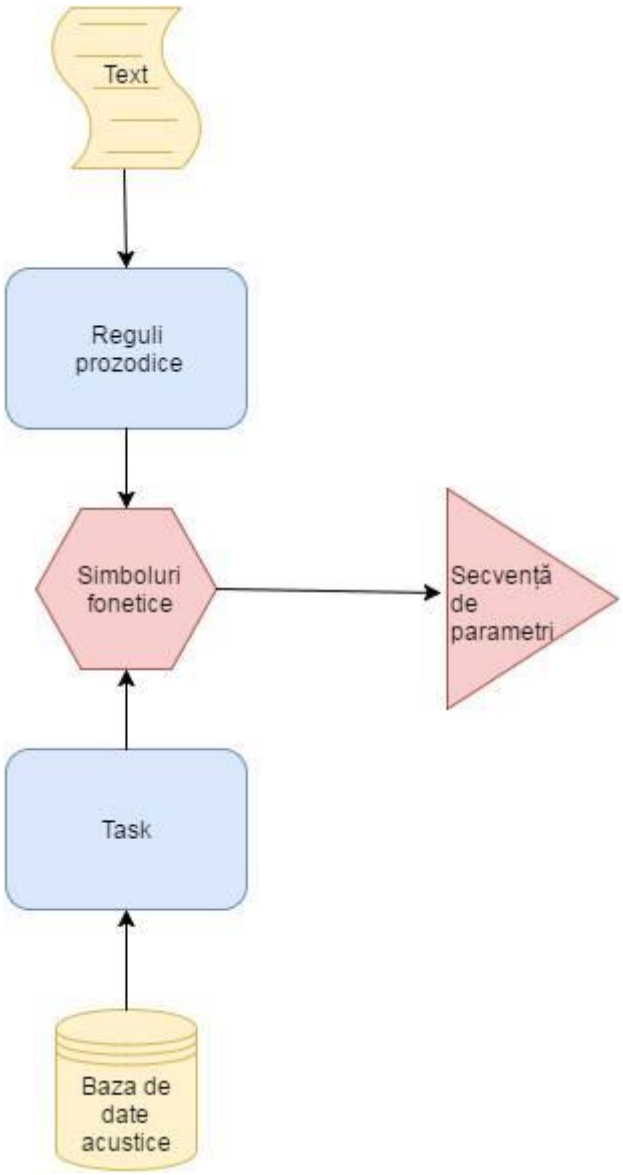

Figura 5.2: Generarea parametrilor acustici

5.2 SINTEZĂ VOCALĂ BAZATĂ PE HMM

HMM este un model statistic de tip Markov în care sistemul modelat se presupune a fi un proces Markov cu stări neobservalibe (ascunse). Poate fi prezentat drept cel mai simplu model dinamic Bayesian. Matematica din spatele HMM, dezvlotate de L. E. Baum este relaționtă cu problema de filtrare non-lineară folosită inițial la descrierea procedurii de tip *înainte-înapoi.*

Modelele Markov simple precum lanțul Markov au stările direct vizibile observatorului, de aceea probabilitățile tranzițiilor constiuie singurii parametri. In HMM, starea nu este direct vizibilă, dar ieșirea, dependentă de stare este. Fiecare dintre aceste stări are asociată o distribuție de probabilitate în raport cu ieșirile posibile, de aceea, secvența de ieșire oferă ea însăși informații despre stările modelului. Atributul de „ascuns” se referă la o secvență de stări prin care modelul trece în timpul funcționării și nu la prametrii modelului.

HMM este conoscut în mod deosebit datorită apilcabilității în recunoașterea structurilor temporale precum vorbirea, scrisul cursiv de mână, recunoaștereea gesturilor și bioinformaticii. Este considerat a fi o generalizare a unui model mixt unde variabilele latente sunt relaționate cu procesul Markov și nu sunt considerate a fi independente una de cealaltă. Recent, modelele HMM

au fost clasificate în modele de tip triplet Markov și perechi Markov care abordează structuri de date mai complexe și modelarea de date dinamice.

Sinteza vocală bazata pe HMM mai este cunoscută și sub denumirea de sinteză statistică de parametri. În acest sistem, spectrul de frecvență asociat tractului vocal, frecvența fundamentală asociată sursei vocale și durata (prozodia) discursului sunt modelate simultan. Formele de undă ale vorbirii sunt genreate din înseși HMM-urile bazate pe criteriul probabilității maxime.[15]

Cu acest model Markov se pot modela procese complexe unde stările emit semnale în concordanță cu o distribuție de probabilitate precum cea Gaussiană. Mai mult decât atât, ar putea reprezenta un comportament mai complex atunci când ieșirea succesiunii de stări este reprezentată sub forma unei combinații de două sau mai multe Gaussiene în care probabilitatea de a genera o ieșire este produsul tutror probabilităților de a selecta una dintre Gaussiene si ce de a genera o obervație din respectiva Gaussiană.

Sistemele TTS au început să folosească HMM pentru a genera „părţi de vorbire”, ceea ce ajută la dezambiguizarea omografelor. Această tehnică este una de succes în cele mai multe cazuri. De exemplu, cuvantul „haină” care poate lua sensul de îmbrăcăminte sau femininul de la „hain”.

Rata de erori tipică în urma utilizării acestor modele este de sub 5%, implicit pentru limbile europene, desi este mai dificilă antrenarea unui corpus asociat.

O altă problemă pregnantă a sistemelor TTS este cea a numerelor compuse. De exemplu, „1989” trebuie pronunţat în limba română „o mie nouă sute opt zeci şi nouă”, în unele contexte, dar există şi situaţia în care trebuie citită ca atare „unu-nouă-opt-nouă”. Această problemă se exinde şi la numerele romane care se citesc diferit în funcţie de context : „clasa a II-a”, sau „Capitolul II”; aici „II” poate fi citit ca „a doua” sau „doi”.

Problema abrevierilor este una ambiguă; de exemplu „asist.”, prescurtarea de la „asistent” şi „asist”, forma verbului „a asista” la prezent.

O posibilă soluţie a acestor probleme o reprezintă sistemele TTS inteligente, care ghicesc pe baza antrenării ambiguităţile sau care, pentru toate cazurile, oferă aceeaşi soluţie mai probabilă. De exemplu, este mult mai probabil ca într-un text dat cuvântul „Ca” de la „calciu” sa fie asociat prepoziţiei comparative „ca” pe baza apariţiilor mai frecvente în limba română.

Imaginea de ansamblu a unui sistem de sinteză vocală bazat pe principiul HMM este divizat în două zone de prelucrare a datelor: o zonă de antrenare și o zonă de sinteză. În zona de antrenare se regăsesc baza de date de vorbire, parametrii spectrali și de excitație în conformitate cu care se face antrenarea. În cea de-a doua zonă se efectuează o analiză a textului și se genrează parametrii modelului Markov.

Zona de antrenare a modelululi HTS este similară celei bazată pe HMM. Diferența esențială este aceea că vectorul stărilor de ieșire include nu numai parametri spectrali, precum cepstrul Mel, dar și sursa excitației, parametrii F0. [14]

Partea de sinteză face operația inversă recunoașterii vocale, respectiv concatenează fonemele HMM în asociere cu textul de sintetizat. O secvență de parametri de vorbire este ulterior determinată în așa fel încât probabilitatea sa de apariție devine maximă. În final, semnalul audio este sintetizat prin filtrare.

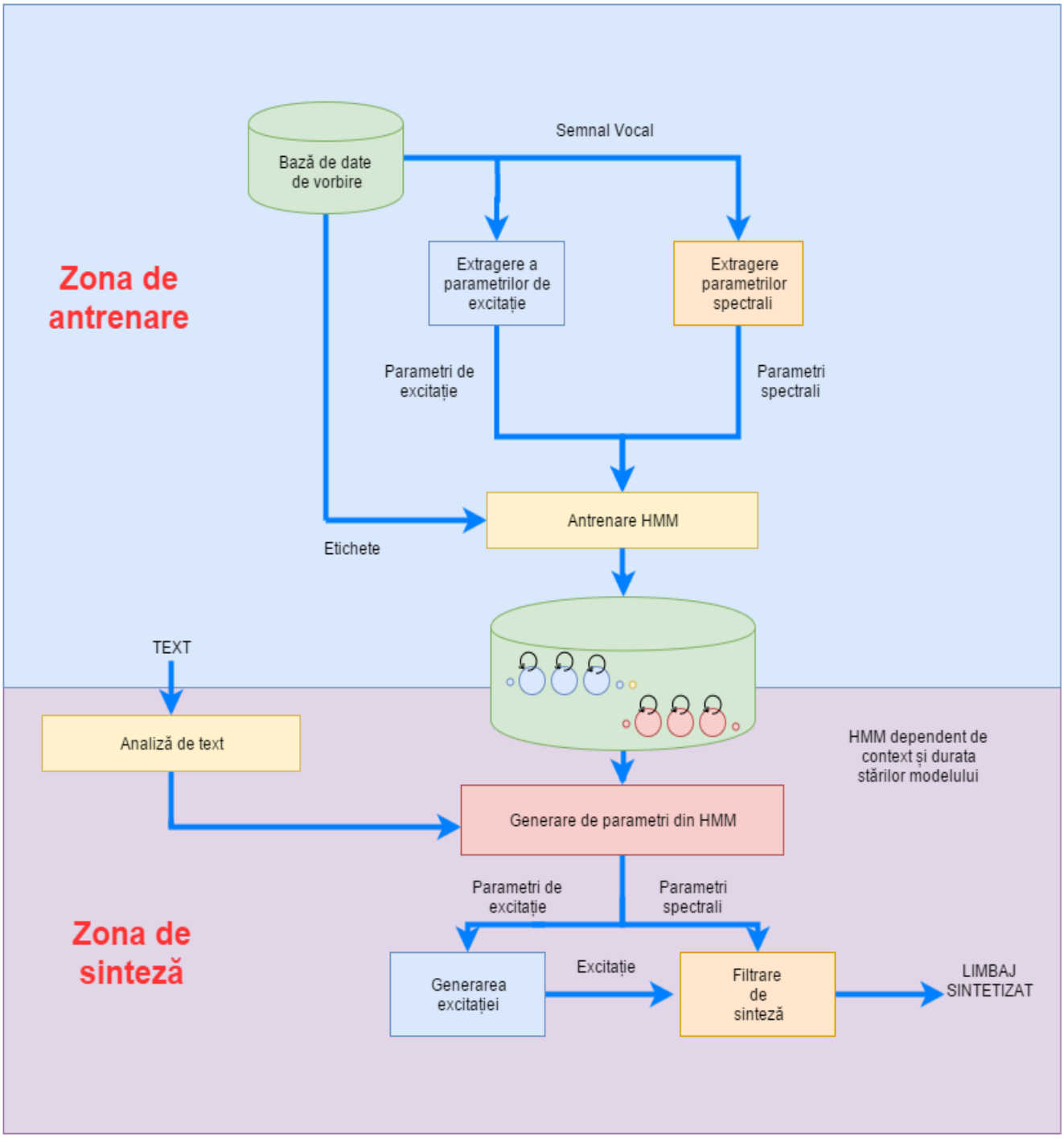

Figura 5.3: Schema bloc a sistemului HTS [14]

5.3 STUDIU ASUPRA DEZVOLTĂRII UNUI SISTEM DE SINTEZĂ PENTRU LIMBA ROMÂNĂ

Sistemul de sinteză vocală bazat pe HMM, HTS, este foarte bine documentat din prisma caracteristicilor teoretice ce țin de modul în care acesta funcționează dar, în schimb, nu există o documentație despre felul în care ar trebui modelată limba țintă a sistemului. Acesta este un foarte mare neajuns deoarece, pentru acest lucru sunt puse la dispoziție, de către cei care au dezvoltat acest instrument, doar câteva exemple pentru câteva limbi cum am fi: engleză, portugheză, japoneză etc. Aceste exemple sunt prezentate sub forma unor demonstații în care sunt incluse toate

resursele necesare antrenării și evaluării unui astfel de sistem, incluzând și o bază de date de clipuri audio cu fișierele de descriere corespunzătoare.

Primul pas pe care l-am facut în sensul înțelegerii modului în care caracteristicile lingvistice sunt pasate grupului de instrumente din HTS a fost să instalez unul dintre aceste demo-uri oferite de cei care au creat acest toolkit. De menționat este și faptul că HTS este contruit pronind de la baza unui alt grup de instrumente care a ca destinație recunoașterea vocală: HTK (Toolkit pentru Modele Markov Ascunse) care include funcții necesare creării și dezvoltării unei aplicații care are la bază Modelele Markov Ascunse.

Am ales ca punct de plecare demo-ul pentru limba engleză doarece este cel mai bine documentat dintre alternativele existente, dar un alt important criteriu de alegere a fost și faptul că era singura limbă pe care o cunoșteam dintre cele oferite spre demonstrație.

Rulând procesul de antrenare și evaluare pentru limba engleză am observat că, deși baza de date este relativ una de mici dimensiuni, rezultatele sunt foare promițătoare. Clipurile audio generate la evaluare au fost peste nivelul așteptărilor. Antrenarea s-a facut cu 1132 de clipuri audio având transcrierile textuale asociate, iar evaluarea s-a facut pentru 40 de transcrieri texuale conform cărora au fost generate clipurile audio corespunzătoare de către sistemul antrenat. Aceste transcrieri textuale oferă toate informațiile necesare generării unui clip audio, pornind de la transcrierea fonetică, durata estimată a fonemelor până la modelul de context.

Urmărind codul sursă am observat că transcrierile textuale sunt generate din textul propriu-zis de către un instrument extern care face parte din grupul *Festival,* care pune la dispoziție o serie de unelte necesare și suficiente pentru crearea de noi voci. Căutând în funcțiile din *Festival* care au fost folosite pentru a generea aceste transcrieri am observat că acestea se foloseau de un model de limbă pentru limba engleză deja existent. Verificând ce fișiere sunt oferite în exemplele pentru alte limbi, am descoperit că nu era oferit textul propriu zis pentru fiecare clip audio în parte, ci erau oferite doar transcrierile textuale. Un alt lucru pe care l-am observat, a fost faptul că formatul fișierelor care conțineau transcrierea textuală este diferit de la o limbă la alta, în funcție de caracteristicile limbii respective.

Studiind foarte amănunțit toate tipurile de fișiere existente într-un demo, am fost capabil să realizez un scurt sumar al tuturor informațiilor necesare pentru a pune în funcțiune un astfel de sistem. În figura 5.4 se ilustrează tipurile de fișiere cu informațiile conținute de acestea.

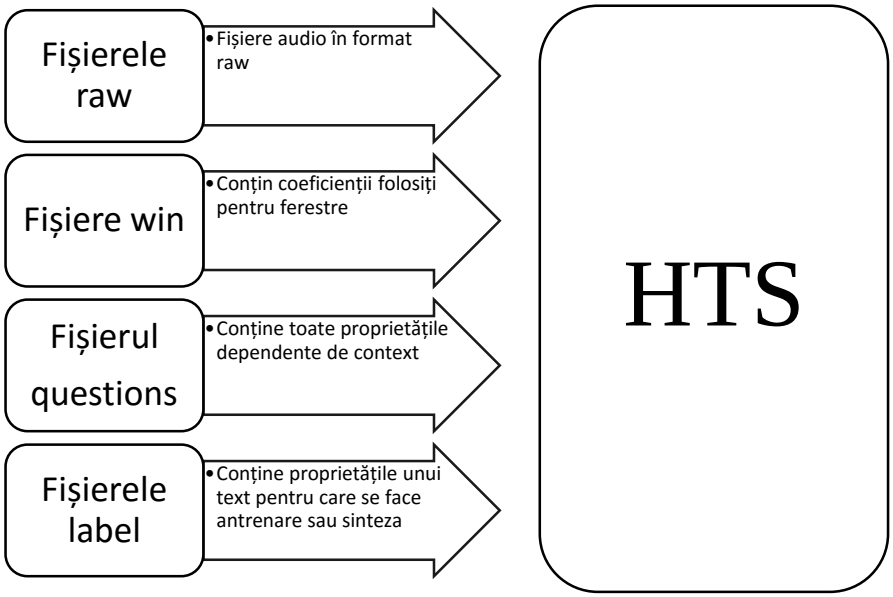

Figura 5.4: Datele de intrare în HTS

Fișierele *raw* au drept conținut semnalul vocal pe baza căruia se face antrenarea sistemului. Aceste fișiere pot fi obținute foarte simplu din fișiere audio, în care semnalul vocal este comprimat cu ajutorul diverselor mijloace de compresie, în formatul *raw* datele audio fiind necomprimate. Pentru conversie se pot folosi numeroase instrumente, dar în demo-ul oferit se recomandă folosirea unei funcții conținută de toolkit-ul SPTK pentru această conversie. Mai rămâne însă problema modului de selecție a acestor clipuri audio, deoarece dorim sa avem o acoperire cât mai largă asupra situațiilor de vorbire pentru limba antrenată și in același timp dorim și ca numărul situațiilor alese să permită modelului o antrenare îndeajuns de bună pentru ca sistemul să fie capabil mai apoi de o generalizare.

Fișierele *win* conțin coeficienții pentru ferestrele folosite de HTS pentru a extrage proprietățile dinamice din semnalul vocal. Modul în care este definită o astfel de fereastra este prezentat mai jos.

```
/* HTS_Window: window coefficients to calculate dynamic features. */
   94 typedef struct _HTS_Window {
   95    size_t size;                /* # of windows (static + deltas) */
   96    int *l_width;               /* left width of windows */
   97    int *r_width;               /* right width of windows */
   98    double **coefficient;       /* window coefficient */
   99    size_t max_width;           /* maximum width of windows */
   100 } HTS_Window;
```

Fișierul *questions* explicitează o listă a tuturor proprietăților pe care le poate avea un anumit fonem. Există o corespondență biunivocă între acest fișier *questions* și formatul în care trebuie livrate datele de transcriere textuală. Presupunând că pentru un fonem avem 7 proprietăți care trebuie livrate în fișierul de transcriere textuală, atunci fișierul *questions* va fi structurat sub forma a șapte liste de posibile proprietăți, fonemul respectiv; în acest caz, având prima proprietate aleasă din prima listă din fișierul *label*, a 2-a din a 2-a listă și așa mai departe. Aceste liste de proprietăți depind de caracteristicile limbii pentru care se face antrenarea. Listele de proprietăți din fișierul *questions* sunt reprezentate sub forma unor înlănțuiri de comenzi (întrebări). Un exemplu de astfel de comandă este:

QS "LL_Vowel" {a^*,E^*,O^*,e^*,i^*,o^*,u^*,a~^*,e~^*,i~^*,o~^*,u~^*}

QS este un cuvânt cheie care inidică faptul că urmează o declarație a unei comenzi, “LL_Vowel” reprezintă numele proprietății iar enumerația dintre paranteze desemnează fonemele care pot fi încadrate ca având această proprietate. Din păcate, nu am putut găsi nici un fel de documentație cu privire la cum se poate realiza acest tip de fișier.

Fișierele *label* sunt fișierele care conțin transcrierea textuală asociată fiecărui clip audio. În aceste fișiere se află informațiile de durată și cele dependente de context ale fiecărui fonem.

5.4 Concluzii

Dezvoltarea și optimizarea unui astfel de sistem implică, în primul rând, un studiu amănunțit asupra caracteristicilor limbii române, evaluarea impactului pe care îl au aceste caracteristici asupra vorbirii și selectarea acelora absolut necesare pentru ca sistemul de sinteză să poată genera la ieșire un semnal audio cât mai asemănător cu acela înregistrat de la un vorbitor.

În al 2-lea rând, se pune problema de o codificare a acestor proprietăți pentru a realiza o transcriere textuală automată pornind de la un simplu text, ceea ce se traduce mai exact prin crearea unui modul de NLP care să ofere la ieșire informațiile despre text necesare HTS-ului.

În al 3-lea rând, se pune problema obținerii unei baze de date de clipuri audio care să acopere cât mai multe din situațiile întâlnite în limba vorbită. Ținând cont de aceste lucruri am considerat că obiectivul de a realiza un sistem de această anvergură într-un timp atât de scurt nu este posibilă, cel putin fără neglijarea celorlalte obiective și chiar și în acest fel șansele să reușesc ar fi minime.

CAPITOLUL 6

DEZVOLTAREA APLICAȚIEI DE INTERACȚIUNE PRIN VOCE

Pe durata dezvoltării am realizat un cumul de instrumente ce rezidă în logica indusă robotului NAO, precum *text-to-speech, speech-to-text,* sau *fetching* de obiecte, care își pot găsi utilitatea în viața oamenilor cu deficiențe de vorbire, auz, sau motorii. În cele ce urmează, va fi descris procesul de realizare practică al proiectului ce constituie obiectul acestei teze. Pentru o descriere facilă, se pornește de la organigrama algoritmului, atașată mai jos.

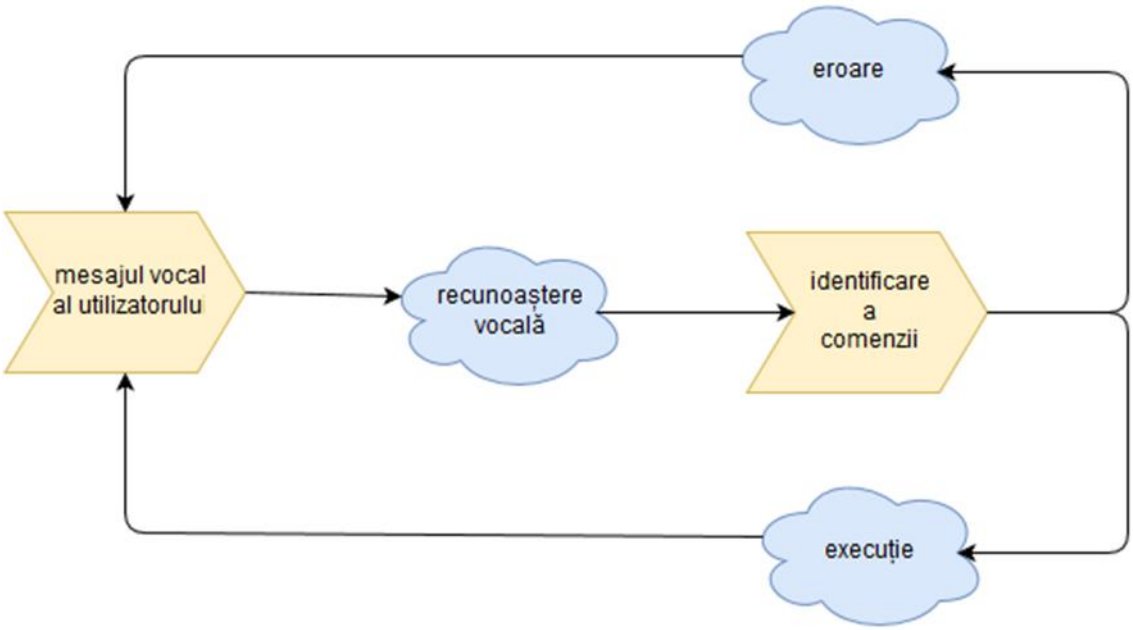

Figura 6.1: Organigrama algoritmului functional

Am început prin a căuta o metodă de a elimina necesitatea unei conexiuni la internet, deoarece până acum aplicațiile de recunoaștere vocală în limba română dezvoltate pentru robotul NAO realizau o conexiune la un server pe care rula procesul propriu-zis de recunoaștere vocală. Eliminarea acestei conexiuni conduce la reducerea notabilă a timpului de răspuns în cazul în care procesul de recunoaștere nu ar necesita foarte multe resurse, cum este cazul unei recunoașteri de comenzi. De asemenea, un alt obiectiv este recunoașterea comenzilor în mod continuu; în acest fel interacțiunea cu robotul realizându-se doar prin voce, fapt care până acum nu a fost realizat deoarece aplicațiile necesitau apăsarea senzorilor tactili de pe capul robotului pentru a înregistra un fișier audio pentru a fi trimis spre serverul de recunoaștere vocală.

Având în vedere că sistemul de operare al robotului nu este echipat cu un compilator de C se pune problema compatibilității aplicației *PocketSphinx* și compilarea acestei aplicații astfel încât să poată rula ca un proces de sine stătător pe robot. Studiind modalitățile pe care le sugera producătorul robotului pentru obținerea unui software compilat special pentru a respecta arhitectura sistemului de calcul al robotului, am aflat cum se poate inițializa și configura un mediu special care sa permită o compilare de această natură.

Această soluție necesită foarte mult timp dedicat și ar fi fost fezabilă doar în cazul în care se dorea dezvoltarea unei aplicații pentru arhitectura robotului, acest volum de muncă fiind nejustificat doar pentru compilarea unei aplicații o singură dată.

Având în vedere aceste aspecte, am luat în considerare ca soluție optimă compilarea aplicației de recunoaștere vocală *PocketSphinx* pe o mașină virtuală care sa respecte întocmai caracteristicile sistemului de operare aflat pe robotul NAO cât și resursele pe care acesta le are la dispoziție, urmând ca apoi, această aplicație să fie portată de pe mașina virtuală pe robot incluzând atât executabilele cât și librăriile rezultate în urma compilării.

Urmând această metodă, am realizat compilarea instrumentului *PocketSphinx* și am reușit portarea pe robot. Așa cum am detaliat în Capitolul 3, am folosit modelul acustic oferit de Laboratorul SpeeD și am continuat prin crearea unui dicționar fonetic și a unei gramatici care sa se ridice la nivelul obiectivelor prezentate în capitolul de introducere.

Crearea dicționarului fonetic, care s-a desfășurat urmând pașii descriși în capitolul 3, versiunea finală conținând și cuvintele de tip garbage care au fost introduse pentru a face posibilă

recunoașterea continuă a vorbirii. În figura următoare se ilustrează o secțiune din dicționarul fonetic în varianta finală.

```
un u n
verzi v e r z i1
vorbeşte v o r b e s1 t e
vremea v r e m e1 a
garbage a
garbage(2) e
garbage(3) a1
garbage(4) o
```

Figura 6.2: Secțiune din dicționarul fonetic

După cum se poate observa, cuvintele de tip garbage sunt de fapt toate fonemele existente în modelul acustic care printr-o gramatică special concepută vor facilita recunoașterea comenzilor indiferent de încadrarea lor într-un context. În dicționarul fonetic avem pe fiecare linie cuvântul urmat de transcrierea lui fonetică. Un dicționar fonetic permite asocierea mai multor transcrieri fonetice ale aceluiași cuvânt acest lucru realizându-se prin adăugarea unui indice care să le diferențieze. În exemplul din figură, în dreptul cuvântului *garbage* apare prima sa variantă de transcriere fonetică iar în dreptul cuvântului *garbage(2*) apare a doua variantă de transcriere fonetică a acestuia.

Gramatica folosită pentru acest sistem este de tipul gramatică cu stări finite în formatul FSG. Așa cum am detaliat în Capitolul 3, formatul FSG al gramaticii se obține prin prelucrarea unui fișier JSGF cu utilitarul *sphinx_jsgf2fsg*. Spre deosebire de gramatica folosită pentru realizarea sistemului de recunoaștere de cifre, în acest caz a fost necesară și o implementare a probabilităților de succesiune ale comenzilor astfel cuvintele *garbage* au o probabilitate mult mai scazută de apariție față de comenzi. Un exemplu de folosire al acestui utilitar poate fi observat în figura următoare.

```
C:\Users\ngeony\Desktop\pocketsphinx\bin\Release\x64>sphinx_jsgf2fsg -jsgf nao.jsgf -fsg nao.fsg
Current configuration:
[NAME]          [DEFLT] [VALUE]
-compile        no      no
-fsg                    nao.fsg
-fsm
-help           no      no
-jsgf                   nao.jsgf
-symtab
-toprule

INFO: jsgf.c(706): Defined rule: <nao.g00000>
INFO: jsgf.c(706): Defined rule: <nao.g00001>
INFO: jsgf.c(706): Defined rule: <nao.all>
INFO: jsgf.c(706): Defined rule: <nao.g00003>
INFO: jsgf.c(706): Defined rule: <nao.g00004>
INFO: jsgf.c(706): Defined rule: <nao.g00005>
INFO: jsgf.c(706): Defined rule: <nao.g00006>
INFO: jsgf.c(706): Defined rule: <nao.g00007>
INFO: jsgf.c(706): Defined rule: <nao.g00008>
INFO: jsgf.c(706): Defined rule: <nao.g00009>
INFO: jsgf.c(706): Defined rule: PUBLIC <nao.commands>
INFO: main.c(118): No -toprule was given; grabbing the first public rule: '<nao.commands>' of the grammar 'nao'.
INFO: jsgf.c(365): Right recursion <nao.g00009> 4 => 0
INFO: fsg_model.c(843): Writing FSG file 'nao.fsg'
```

Figura 6.3: Utilitarul sphinx_jsgf2fsg (Versiunea pentru windows)

Această gramatică reprezintă un set de reguli prestabilite, sub formă de graf, unde putem imagina nodurile ca fiind cuvintele vocabularului, iar arcele grafului reprezintă tranzițiile între cuvinte. Pe baza acestuia se construiește Modelul de Limbă. Inițial se definesc nodurile de intrare și cele de ieșire, care nu corespund afișării vreunui cuvânt ci au rol de infrastructură. Ulterior, se crează nodurile specifice fiecărui cuvânt în parte și se face interconectarea lor la intrare, respectiv ieșire cu ajutorul arcelor direcționate la care se mai adaugă și tranziția înapoi.

În corpusul gramaticii sunt înregistrate comenzi vocale precum: “Mergi un metru”, “Fă-ți ochii verzi/roșii/albaștri”, “Mișcă-ți mâna stângă/dreaptă”, “Vorbește mai tare/încet”, “Stai jos!”,

"Ridică-te în picioare", dar și cupluri întrebare-răspuns precum: "Cât este ceasul?", "Cum e vremea?".

```
INFO: cmn_prior.c(131): cmn_prior_update: from < 12.08 -0.13 -0.37 -0.02 -0.71 -0.5
5 -0.37 -0.31  0.12 -0.20 -0.13 -0.05 -0.28 >
INFO: cmn_prior.c(149): cmn_prior_update: to   < 11.95 -0.17 -0.35 -0.02 -0.71 -0.5
4 -0.36 -0.30  0.13 -0.20 -0.13 -0.05 -0.28 >
INFO: fsg_search.c(859): 266 frames, 1780 HMMs (6/fr), 6004 senones (22/fr), 265 hi
story entries (0/fr)

INFO: fsg_search.c(869): fsg 0.36 CPU 0.135 xRT
INFO: fsg_search.c(871): fsg 3.71 wall 1.389 xRT
vorbeşte mai tare
INFO: continuous.c(275): Ready....
INFO: continuous.c(261): Listening...
INFO: cmn_prior.c(131): cmn_prior_update: from < 11.95 -0.17 -0.35 -0.02 -0.71 -0.5
4 -0.36 -0.30  0.13 -0.20 -0.13 -0.05 -0.28 >
INFO: cmn_prior.c(149): cmn_prior_update: to   < 12.00 -0.15 -0.34 -0.02 -0.72 -0.5
4 -0.34 -0.31  0.11 -0.19 -0.12 -0.03 -0.27 >
INFO: fsg_search.c(859): 183 frames, 1905 HMMs (10/fr), 5946 senones (32/fr), 293 h
istory entries (1/fr)

INFO: fsg_search.c(869): fsg 0.65 CPU 0.352 xRT
INFO: fsg_search.c(871): fsg 19.33 wall 10.507 xRT
mişcă mâna stângă
INFO: continuous.c(275): Ready....
```

Figura 6.4: Rezultat returnat în cadrul *speech-to-text*

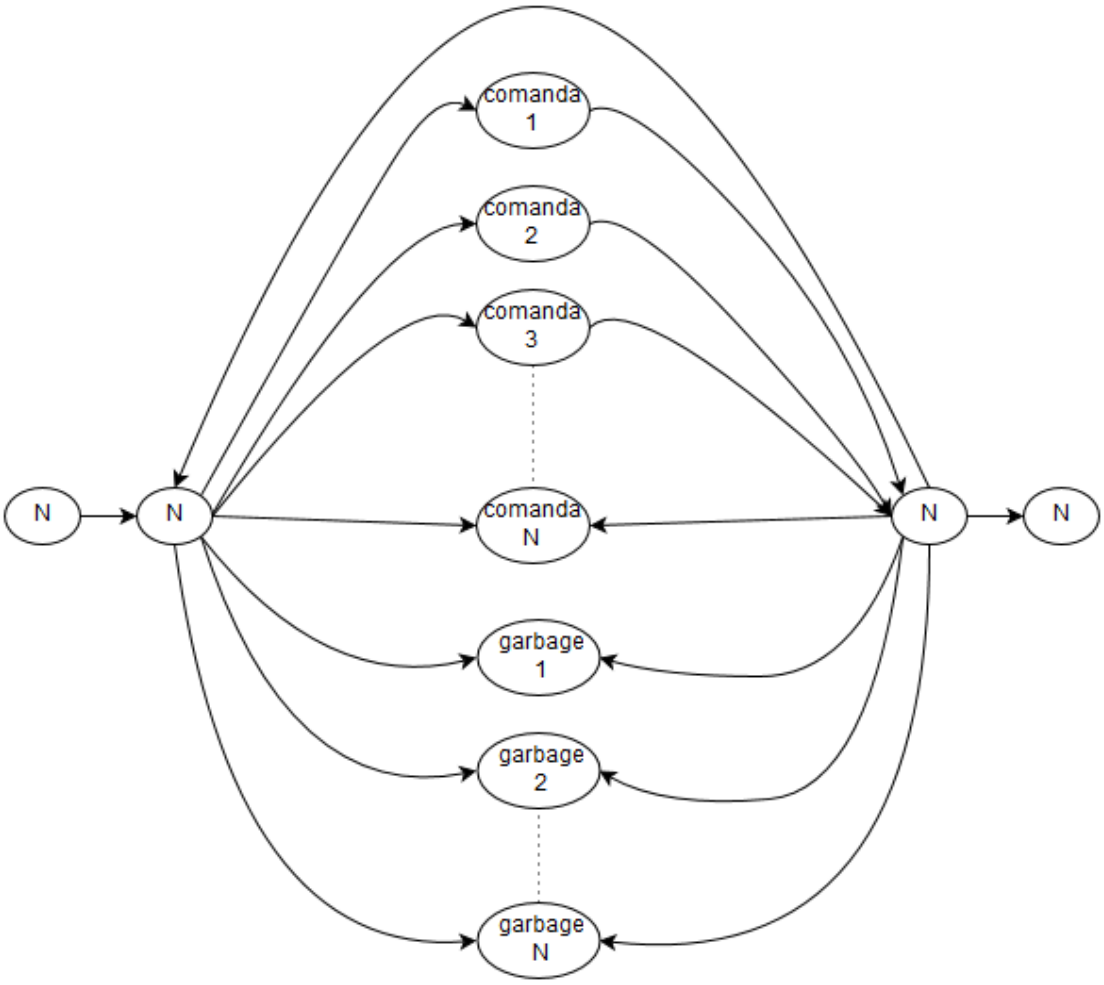

Figura 6.5: Graful gramaticii implementate

Succedarea evenimentelor constructive presupun inițierea prin mesaj vocal din partea utilizatorului, adresat robotului NAO asigurându-se că microfonul recepționează inteligibil semnalul sonor. Aici au fost întâmpinate dificultăți, zgomotul, dar acest impediment a fost soluționat prin compilarea unei noi versiuni a instrumentului *PocketSphinx* care are incorporat un modul de reducere a zgomotului. Acest modul se bazează pe urmărirea minimului spectral în subbenzi, presupunând astfel ca acest minim este chiar zgomotrul ce se dorește eliminat. Potrivit rezultatelor obtinute de cei care au creat acest modul, aceasta metoda necesită între 14% și 23 % din resursele unui procesor de semnal aflat în acest moment pe piață. [10][11]

Odată achiziționat mesajul audio, este trecut prin sistemul de recunoaștere vocală, în vederea transpunerii acestuia în instrucțiuni cu rol de comandă de execuție. Transpunerea se face prin pornirea *PocketSphinx* -ului care scrie în fișier ce a recunoscut pe fiecare linie de ieșire. Scrierea se face pe baza unui script, atașat în **Anexa**. Un alt script în Python preia fișierul rezultat și prelucrează aceste ieșiri, aducându-le sub formă de comandă.

Pentru etapa de execuție, este redactat un al treilea script în Python, care face legătura între comanda primită și răspunsul fizic al robotului. Legătura dintre comanda procesata în blocul de identificare a comenzii și efectuarea propriu-zisă a unei acțiuni în urma procesării acesteia, este realizată prin intermediul librăriei ALProxy dezvoltată de producător pentru instrumentele Python. Astfel, setul de comenzi NAO, puse la dispoziție de pachetul Aldebaran Robotics, s-a corelat cu instrucțiunile create prin parcurgerea scripturilor premergătoare acestei faze.[1] De exemplu, pentru a aduce robotul în anumite poziții ar trebui să avem următoarele instanțieri:

```
posture = ALProxy("ALRobotPosture", self.robotIp, self.robotPort)
posture.goToPosture(self.mapping[position], 0.5)
```

După cum se poate observa prima funcție crează o conexiune cu programul principal care se ocupă de managementul robotului. A doua a linie de cod reprezintă instrucțiunea propriu-zisă, *self.mapping[position*] fiind ori “Sit” care indică robotului că trebuie să stea jos, ori “Stand” care indică statul în picioare.

La sfârșitul execuției unei instrucțiuni se intră într-o stare de expectativă a unui nou mesaj care preia ciclul descris mai sus.

Python este un limbaj de uz general, ceea ce înseamnă că poate fi integrat în majoritatea proiectelor ce presupun cod, pe care îl face să devină mai fluent prin intermediul librăriilor și instrumentelor sale. Din punct de vedere profesional, Python este util în partea de dezvoltare backend, analiza de texte date, inteligența artificială și calcul avansat. Mediul de dezvoltare este unul prietenos cu utilizatorul. În perimetrul gramaticii construite, NAO poate redacta în terminal textul mesajului vocal recepționat. Mai jos este un exemplu al acestui instrument creat. Un eventual scop al extensiei, ar fi acela de a dicta un text robotului, pe care il translateaza în corpul unui e-mail, sau își poate găsi utilitatea în activitatea unei persoane cu deficiență de auz.

În imaginea de mai sus, s-a obținut această ieșire după ce i s-a rostit lui NAO mesajul vocal: “vorbește mai tare”. Acesta a scris în fișier mesajul, după care a exprimat audio că și-a setat volumul la un nou nivel, vorbind efectiv, mai tare.

Pentru a obține răspuns despre vreme a fost conectat prin protocolul RestClient la un URL în format HTTP la o pagină cu informații meteo.

Pentru o descriere explicită a execuției unei comenzi, luăm exemplul următor. La procesarea cuvântului “fă-ți ochii”, NAO se așteaptă la mai mult decât o posibilitate. Așadar, discutăm despre un proces decizional pe baza criteriului culoare. Secvența se va executa ramificat.

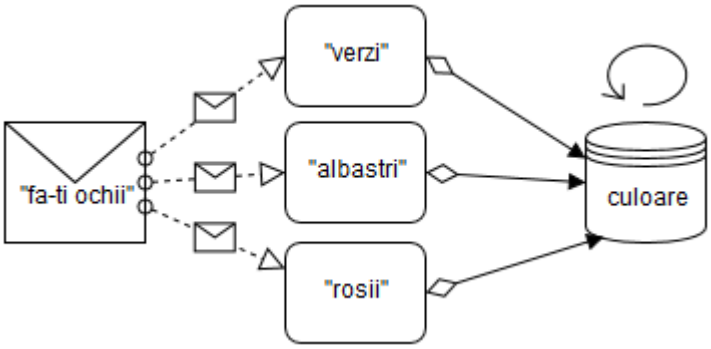

Figura 6.6: Proces decizional în logica implementată

CAPITOLUL 7

CONCLUZII

Principalul obiectiv al acestei lucrări a fost construirea unui sistem de interacțiune prin voce cu robotul NAO în limba română, în esență robotul să poată recunoaște anumite comenzi predefinite în limba română dar în același timp să permită și o recunoaștere continuă, mai exact să poată selecta secvențele cu comenzi din cadrul unui context și să renunțe la datele care sunt redundante. Apoi robotul să interpreteze aceste comenzi și să genereze un răspuns corespunzător atât din punct de vedere vocal cât și din punct de vedere al mișcării.

Pe parcursul realizării tezei am înțeles fundamentele unui sistem automat de recunoaștere vocală și modul în care este creat un astfel de sistem pentru un singur vorbitor, optimizarea acestuia și observând rezultatele obținute la evaluarea acestuia pe un set de date de la mai mulți vorbitori, am decis că este absolut necesară crearea unui sistem automat de recunoaștere vocală independent de vorbitor.

Am reușit portarea aplicației de recunoaștere vocală *PocketSphinx* pe robot și prin adăugarea resurselor necesare am obținut un rezultat foarte mulțumitor din punct de vedere al comenzilor recunoscute. Vorbirea continuă am impementat-o folosind o gramatică de tip *garbage* și astfel am considerat atinse obiectivele pe care mi le-am propus din punct de vedere al recunoaștrii vocale.

Am abordat subiectul sintezei vocale, dar după un studiu mai amănunțit al modului în care aceasta se realizează folosind HMM am decis că o continuare pe aceasta direcție nu mi-ar mai fi permis atingerea celorlalte obiective în timp util.

Pentru crearea aplicației de comandă și control al robotului am folosit limbajul *Python* deoarece pentru acest limbaj producătorul robotului oferă o gamă largă de materiale și de exemple care deși sunt destul de dificile au fost o documentație bună.

Ca dezvoltări ulterioare ale acestei aplicații consider absoult necesar realizarea unui sistem de sinteză vocală pentru limba română deoarece până acum interacțiunea în limba română se realizează doar dinspre utilizator spre robot cât și realizarea unui studiu care să aibă un rezultat concludent asupra faptului dacă robotul dispune de resursele necesare pentru a rula aceste 2 sisteme fără a avea întârzieri foarte mari.

BIBLIOGRAFIE

[1] http://doc.aldebaran.com/2-1/

[2] H. Cucu, *Proiect de cercetare-dezvoltare în Tehnologia Vorbirii* - îndrumar de proiect

[3] http://telecom.etc.tuiasi.ro/pns/cc/lab_cc/L07_TCSM_CompresiaAudio.pdf accesat 26.08.2016

[4] A. Buzo, *Recunoaşterea Limbajului Vorbit în Reţele de Telecomunicaţii Mobile* , PhD Thesis

[5] https://www.newscientist.com/article/dn7664-invention-thought-controlled-voice-synthesizer/ accesat 29.08.2016

[6] G. Toderean, Al. Caruntu, *Metode de recunoaştere a vorbirii*, Editura Risoprint, Cluj-Napoca, 2005

[7] Moller, Aage R. (2006). *Hearing: Anatomy, Physiology, and Disorders of the Auditory System (2 ed.).* Academic Press. p. 217. ISBN 9780080463841.

[8] Titze, I.R. (1994). *Principles of Voice Production*, Prentice Hall (currently published by NCVS.org) (pp. 188), ISBN 978-0-13-717893-3.

[9] G. Doblinger *Computationally efficient speech enhancement by spectral minima tracking in subbands*, Institut fur Nachrichtentechnik und Hochfrequenztechnik, Technische Universitat Wien, Gusshausstr. 25, A-1040 Vienna, Austria

[10] R. Martin, *Spectral subtraction based on minimum statistics*, in Proc. Seventh European Signal Processing Conference, pp. 11821185, Sept. 1994.

[11] Y. Ephraim, D. Malah, *Speech enhancement using a minimum mean-square error short-time spectral amplitude estimator*, IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-32, pp. 1109{1121, Dec. 1984.

[12] D. Burileanu, *Contributions on Text-to-Speech Synthesis in Romanian Language* , PhD Thesis

[13] C. Burileanu, editor, *Speech Technology and Human-Computer Dialogue*, the Publishing House of the Romanian Academy, Bucharest, 2003 (ISBN 973-27-0963-4).

[14] HTS Working Group, *HTS Slides ver. 2.3* , http://hts.sp.nitech.ac.jp/, Copyright (c) 1999 – 2015 ,Nagoya Institute of Technology, Department of Computer Science

[15] Dr. Ing A. C. STAN, *Sinteza text-vorbire în limba română bazată pe modele Markov şi optimizarea interactivă a intonaţiei*, Conducător ştiinţific Prof.dr.ing. Mircea GIURGIU, 2011

[16] http://thebrain.mcgill.ca/flash/capsules/outil_bleu21.html accesat la 25.08.2016

[17] http://cmusphinx.sourceforge.net/wiki/ accesat la 29.08.2016

ANEXA 1

DICȚIONAR FONETIC

albaştrii a l b a s1 t r i i3
azi a z i1
cât k i2 t
ce k1 e
ceasul k1 e1 a s u l
cine k1 i n e
cum k u m
dată d a t a1
dreaptă d r e1 a p t a1
e i3 e
este i3 e s t e
eu e w
fă-ţi f a1 t1 i1
în i2 n
încet i2 n k1 e t
jos j o s
mai m a i3
mâna m i2 n a
mergi m e r g1
metru m e t r u
mişcă m i s1 k a1
ochii o k2 i i3
picioare p i k1 o1 a r e
ridică-te r i d i k a1 t e
roşii r o s1 i i3
stai s t a i3
stângă s t i2 n g a1

sunt s u n t

tare t a r e

un u n

verzi v e r z i1

vorbeşte v o r b e s1 t e

vremea v r e m e1 a

garbage a

garbage(2) e

garbage(3) a1

garbage(4) o

garbage(5) b

garbage(6) k

garbage(7) k2

garbage(8) k1

garbage(9) s

garbage(10) s1

garbage(11) d

garbage(12) e1

garbage(13) i3

garbage(14) f

garbage(15) g

garbage(16) g2

garbage(17) g1

garbage(18) h

garbage(19) i

garbage(20) i2

garbage(21) j

garbage(22) l

garbage(23) m

garbage(24) n

garbage(25) w

garbage(26) o1

garbage(27) p

garbage(28) r

garbage(29) z

garbage(30) t

garbage(31) t1

garbage(32) u

garbage(33) v

garbage(34) y

garbage(35) i1

garbage(36) o2

GRAMATICA ÎN FORMAT JSGF

```
#JSGF V1.0;

/**
 * JSGF Grammar for the FSG Speech-to-Text Online Demo
 */

grammar nao;

/**
 * Commands for the NAO robot
 */
<all> = (a | e | a1 | o | b | k | k2 | k1 | s | s1 | d | e1 | i3 | f | g | g2 | g1 | h | i | i2 | j | l | m | n | w | o1 | p | r | z | t | t1 | u | v | y | i1 | o2 )*;
 public <commands> = (/0.0001/ garbage |/0.999/ mergi un metru |/0.999/ mişcă mâna (dreaptă | stângă) |/0.999/ ridică-te în picioare |/0.999/ stai jos |/0.999/ cât (e | este) ceasul |/0.999/ cum (e | este) vremea |/0.999/ fă-ţi ochii (roşii | albaştrii | verzi) |/0.999/ vorbeşte mai (tare | încet))*;
```

GRAMATICA ÎN FORMAT FSG

```
FSG_BEGIN <nao.commands>
NUM_STATES 18
START_STATE 0
FINAL_STATE 1
TRANSITION 0 2 0.999001 vorbeşte
TRANSITION 0 4 0.000100 garbage
TRANSITION 0 5 0.999001 fă-ţi
TRANSITION 0 7 0.999001 cum
```

```
TRANSITION 0 9 0.999001 cât
TRANSITION 0 11 0.999001 stai
TRANSITION 0 12 0.999001 ridică-te
TRANSITION 0 14 0.999001 mişcă
TRANSITION 0 16 0.999001 mergi
TRANSITION 0 1 1.000000
TRANSITION 2 3 1.000000 mai
TRANSITION 3 4 1.000000 tare
TRANSITION 3 4 1.000000 încet
TRANSITION 4 0 1.000000
TRANSITION 5 6 1.000000 ochii
TRANSITION 6 4 1.000000 roşii
TRANSITION 6 4 1.000000 albaştrii
TRANSITION 6 4 1.000000 verzi
TRANSITION 7 8 1.000000 e
TRANSITION 7 8 1.000000 este
TRANSITION 8 4 1.000000 vremea
TRANSITION 9 10 1.000000 e
TRANSITION 9 10 1.000000 este
TRANSITION 10 4 1.000000 ceasul
TRANSITION 11 4 1.000000 jos
TRANSITION 12 13 1.000000 în
TRANSITION 13 4 1.000000 picioare
TRANSITION 14 15 1.000000 mâna
TRANSITION 15 4 1.000000 dreaptă
TRANSITION 15 4 1.000000 stângă
TRANSITION 16 17 1.000000 un
TRANSITION 17 4 1.000000 metru
FSG_END
```

SCRIPTUL DE INTEROGARE AL FIȘIERULUI DE IESIRE AL POCKETSPHINX

```
import argparse
from Nao import Nao

def nao_module(naoInstance):
     executedCommand = False
     with open("./log", "r+") as f:
          for line in f:
               executedCommand                                    =
naoInstance.interpretCommand(line)
               if executedCommand:
                    break
          f.close()
     if executedCommand is True:
          with open("./log", "w") as f: pass

if __name__ == "__main__":
     naoRobot = Nao()
```

```
     while True:
          nao_module(naoRobot)
```

CODUL SURSĂ AL APLICAȚIEI ÎN PYTHON

```
#!/usr/bin/python
#coding=UTF-8

import os
from naoqi import ALProxy
import motion
import datetime
import subprocess
from random import randint
import time
import urllib2

class Nao:

    def __init__(self, name = "Jhonny", robotIP = "127.0.0.1",
robotPort = 9559):
        self.name = name
        self._presentMySelf()
        self.robotIp = robotIP
        self.robotPort = robotPort
        self._createEyeColors()
        self._startListening()

        self.commands = {
            'mişcă mâna': self.moveHand,
            'fă-ţi ochii': self.setEyeColor,
            'vorbeşte mai': self.setVolume,
            'ceasul': self.sayHour,
            'ridică-te în picioare': self.bodyPosition,
            'stai jos': self.bodyPosition,
            'vremea': self.sayWeather,
            'mergi': self.walk,
            'înapoi': self.walkBack,
            'ora exactă': self.sayHour,
        }

        self.mapping = {
            'stângă': 'L',
            'dreaptă': 'R',
            'verzi': 'green',
            'roşii': 'red',
            'albaştrii': 'blue',
            'ridică-te în picioare': 'Stand',
            'stai jos': 'Sit',
            'un metru': 1,
```

```
            'un metru jumate': 1.5,
            'tare': 1,
            'încet': 0.5
        }

        self.jointMapping = {
            'stângă':     ["LShoulderPitch",     "LShoulderRoll",
"LElbowYaw", "LElbowRoll"],
            'dreaptă':      ["RShoulderPitch",   "RShoulderRoll",
"RElbowYaw", "RElbowRoll"]
        }

    def stiffnessOn(self, proxy):
        # We use the "Body" name to signify the collection of all
joints
        pNames = "Body"
        pStiffnessLists = 1.0
        pTimeLists = 1.0
        proxy.stiffnessInterpolation(pNames,     pStiffnessLists,
pTimeLists)

    def _presentMySelf(self):
        os.system("say \"initialized module NAO\"")
        os.system("say \"my name is " + self.name + "\"")

    def _startListening(self):
        output = open("log", "w+")
        os.environ['LD_LIBRARY_PATH'] = '/home/nao/rv2/local/lib'
        os.environ['PKG_CONFIG_PATH']                              =
'/home/nao/rv2/local/lib/pkgconfig'

#os.system("/home/nao/rv2/local/bin/pocketsphinx_continuous  -hmm
/home/nao/rv/share/pocketsphinx/model/ASR/mixModels_2KS.cd_cont_2
000_64/                                                    -dict
/home/nao/rv/share/pocketsphinx/model/ASR/europarl9amHotnews10k.a
ugmented.dic.full.plus.FSGv4.sortUniq.dic                   -fsg
/home/nao/rv/bin/modelnqo.fsg -inmic yes 1> log")

subprocess.Popen("/home/nao/rv2/local/bin/pocketsphinx_continuous
-hmm
/home/nao/rv/share/pocketsphinx/model/ASR/mixModels_2KS.cd_cont_2
000_64/    -dict    /home/nao/rv2/local/bin/modelnqo.dic    -fsg
/home/nao/rv2/local/bin/nao.fsg    -inmic    yes    -vad_threshold
3".split(), stdout=output, shell=False)

    def bodyPosition(self, position, *args):
        posture    =    ALProxy("ALRobotPosture",    self.robotIp,
self.robotPort)
        posture.goToPosture(self.mapping[position], 0.5)

    def setVolume(self,ca,direction):
        direction = self.mapping[direction]
```

```
        print self
        print direction
        print ca
        try:
            tts    =    ALProxy("ALTextToSpeech",    self.robotIp,
self.robotPort)
        except Exception,e:
            print "Could not create proxy to ALTextToSpeech"
            print "Error was: ",e
            sys.exit(1)

        #Changes the volume
        tts.setVolume(direction)
        tts.say("Volume set to "+str(direction*100)+"%")

    def walk(self, originalCommand, distance):
        distance = self.mapping[distance]
        motionProxy       =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        postureProxy  =  ALProxy("ALRobotPosture",  self.robotIp,
self.robotPort)

        motionProxy.wakeUp()
        postureProxy.goToPosture("StandInit", 0.5)
        motionProxy.setMoveArmsEnabled(True, True)

motionProxy.setMotionConfig([["ENABLE_FOOT_CONTACT_PROTECTION",
True]])
        # TARGET VELOCITY
        X         = 1.0
        Y         = 0.0
        Theta     = 0.0
        Frequency = 1.0

        # Speed walk  (MaxStepX = 0.06 m)
        # Could be faster: see walk documentation
        motionProxy.moveToward(X,    Y,    Theta,    [["Frequency",
Frequency],
                                                ["MaxStepX", 0.06]])
        time.sleep(distance * 8.0)

        # stop walk on the next double support
        motionProxy.stopMove()
        # Go to rest position
        motionProxy.rest()

    def rotate(self, angle, direction):
        motionProxy       =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        postureProxy  =  ALProxy("ALRobotPosture",  self.robotIp,
self.robotPort)

        motionProxy.wakeUp()
```

```
        postureProxy.goToPosture("StandInit", 0.5)
        motionProxy.setMoveArmsEnabled(True, True)

motionProxy.setMotionConfig([["ENABLE_FOOT_CONTACT_PROTECTION",
True]])
        # TARGET VELOCITY
        X = 1.0
        Y = 0.0
        if direction == 'R':
            Theta     = 1.0
        else:
           Theta     = -1.0

        Frequency = 1.0


        # Speed walk  (MaxStepX = 0.06 m)
        # Could be faster: see walk documentation
        motionProxy.moveToward(X,    Y,    Theta,    [["Frequency",
Frequency],
                                                ["MaxStepX", 0.06]])
        time.sleep(angle * 0.008)
        # stop walk on the next double support
        motionProxy.stopMove()

    def walkBack(self, originalCommand, distance):
        self.rotate(180, 'L')
        self.walk(originalCommand, distance)

    def walkWithBack(self, originalCommand, distance):
        distance = self.mapping[distance]
        motionProxy       =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        postureProxy  =  ALProxy("ALRobotPosture",  self.robotIp,
self.robotPort)

        motionProxy.wakeUp()
        postureProxy.goToPosture("StandInit", 0.5)
        motionProxy.setMoveArmsEnabled(True, True)

motionProxy.setMotionConfig([["ENABLE_FOOT_CONTACT_PROTECTION",
True]])
        # TARGET VELOCITY
        X         = -1.0
        Y         = 0.0
        Theta     = 0.0
        Frequency = 1.0


        # Speed walk  (MaxStepX = 0.06 m)
        # Could be faster: see walk documentation
```

```
        motionProxy.moveToward(X,    Y,    Theta,    [["Frequency",
Frequency],
                                                    ["MaxStepX", 0.06]])
        time.sleep(distance * 8.0)
        # stop walk on the next double support
        motionProxy.stopMove()
        # Go to rest position
        motionProxy.rest()

    def _createEyeColors(self):
        leds = ALProxy("ALLeds", self.robotIp, self.robotPort)
        redEyes = [
            "Face/Led/Red/Left/0Deg/Actuator/Value",
            "Face/Led/Red/Left/45Deg/Actuator/Value",
            "Face/Led/Red/Left/90Deg/Actuator/Value",
            "Face/Led/Red/Left/135Deg/Actuator/Value",
            "Face/Led/Red/Left/180Deg/Actuator/Value",
            "Face/Led/Red/Left/225Deg/Actuator/Value",
            "Face/Led/Red/Left/270Deg/Actuator/Value",
            "Face/Led/Red/Left/315Deg/Actuator/Value",
            "Face/Led/Red/Right/0Deg/Actuator/Value",
            "Face/Led/Red/Right/45Deg/Actuator/Value",
            "Face/Led/Red/Right/90Deg/Actuator/Value",
            "Face/Led/Red/Right/135Deg/Actuator/Value",
            "Face/Led/Red/Right/180Deg/Actuator/Value",
            "Face/Led/Red/Right/225Deg/Actuator/Value",
            "Face/Led/Red/Right/270Deg/Actuator/Value",
            "Face/Led/Red/Right/315Deg/Actuator/Value"
        ]
        leds.createGroup("red",redEyes)
        greenEyes = [
            "Face/Led/Green/Left/0Deg/Actuator/Value",
            "Face/Led/Green/Left/45Deg/Actuator/Value",
            "Face/Led/Green/Left/90Deg/Actuator/Value",
            "Face/Led/Green/Left/135Deg/Actuator/Value",
            "Face/Led/Green/Left/180Deg/Actuator/Value",
            "Face/Led/Green/Left/225Deg/Actuator/Value",
            "Face/Led/Green/Left/270Deg/Actuator/Value",
            "Face/Led/Green/Left/315Deg/Actuator/Value",
            "Face/Led/Green/Right/0Deg/Actuator/Value",
            "Face/Led/Green/Right/45Deg/Actuator/Value",
            "Face/Led/Green/Right/90Deg/Actuator/Value",
            "Face/Led/Green/Right/135Deg/Actuator/Value",
            "Face/Led/Green/Right/180Deg/Actuator/Value",
            "Face/Led/Green/Right/225Deg/Actuator/Value",
            "Face/Led/Green/Right/270Deg/Actuator/Value",
            "Face/Led/Green/Right/315Deg/Actuator/Value"
        ]
        leds.createGroup("green", greenEyes)
        blueEyes = [
            "Face/Led/Blue/Left/0Deg/Actuator/Value",
            "Face/Led/Blue/Left/45Deg/Actuator/Value",
            "Face/Led/Blue/Left/90Deg/Actuator/Value",
```

```
            "Face/Led/Blue/Left/135Deg/Actuator/Value",
            "Face/Led/Blue/Left/180Deg/Actuator/Value",
            "Face/Led/Blue/Left/225Deg/Actuator/Value",
            "Face/Led/Blue/Left/270Deg/Actuator/Value",
            "Face/Led/Blue/Left/315Deg/Actuator/Value",
            "Face/Led/Blue/Right/0Deg/Actuator/Value",
            "Face/Led/Blue/Right/45Deg/Actuator/Value",
            "Face/Led/Blue/Right/90Deg/Actuator/Value",
            "Face/Led/Blue/Right/135Deg/Actuator/Value",
            "Face/Led/Blue/Right/180Deg/Actuator/Value",
            "Face/Led/Blue/Right/225Deg/Actuator/Value",
            "Face/Led/Blue/Right/270Deg/Actuator/Value",
            "Face/Led/Blue/Right/315Deg/Actuator/Value"
        ]
        leds.createGroup("blue", blueEyes)

    def interpretCommand(self, command):
        command = command.replace('\x00', '').strip()
     command = command.replace('garbage ','')^M
     command = command.replace(' garbage','')
     time.sleep(2)
        for key in self.commands:
            if command.find(key) > -1:
                if key != 'ceasul' and key != 'vremea' and key !=
'stai jos' and key != 'ridică-te în picioare':
                    indexOfSubCommand   =   command.find(key)   +
len(key)
                    subcommand                                    =
command[indexOfSubCommand:].replace('\x00', '').strip()
                    print subcommand.strip()
                    for subKey in self.mapping:
                        if subcommand.find(subKey) > -1:
                            self.commands[key](command, subKey)
                            print "executed"
                            return True
                else:
                    self.commands[key](command)
                    return True
        return False

    def sayWeather(self, *args):
        #create weather api url
        url                                                          =
"http://api.openweathermap.org/data/2.5/weather?q=Bucharest&APPID
=06b5c9d7218daf2582bdbedff58420dd"
        try:
            # open weather api url
            f = urllib2.urlopen(url)
        except:
            # if there was an error opening the url, return
            return "Error opening url"
        # read contents to a string
```

```
        s = f.read()

        # extract weather condition data from xml string
        weather = s.split('description":"')[-1].split('","')[0]

        os.system("say \"" + weather + "\"")

    def moveHand(self, orignalPhrase, hand):
        motionProxy    =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        self.stiffnessOn(motionProxy)
        JointNames = self.jointMapping[hand]
        self.openHand(orignalPhrase, self.mapping[hand])
        self.closeHand(orignalPhrase, self.mapping[hand])

        Arm1 = [randint(-45, 45),  randint(-45, 45), randint(-45,
45), randint(-45, 45)]
        Arm1 = [ x * motion.TO_RAD for x in Arm1]
        Arm2 = [randint(-45, 45),  randint(-45, 45), randint(-45,
45), randint(-45, 45)]
        Arm2 = [ x * motion.TO_RAD for x in Arm2]

        pFractionMaxSpeed = 0.4
        motionProxy.angleInterpolationWithSpeed(JointNames, Arm1,
pFractionMaxSpeed)
        motionProxy.angleInterpolationWithSpeed(JointNames, Arm2,
pFractionMaxSpeed)
        motionProxy.angleInterpolationWithSpeed(JointNames, Arm1,
pFractionMaxSpeed)


    def sayHour(self, *args):
        now=datetime.datetime.now().strftime("%I:%M%p")
        os.system("say \"" + now + "\"")

    def openHand(self, orignalPhrase, hand):
        motionProxy    =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        if hand == 'R':
            motionProxy.openHand("RHand")
        elif hand == 'L':
            motionProxy.openHand("LHand")

    def closeHand(self, originalPhrase, hand):
        motionProxy    =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        if hand == 'R':
            motionProxy.closeHand("RHand")
        elif hand == 'L':
            motionProxy.closeHand("LHand")

    def setEyeColor(self, orignalPhrase, eyeColor):
        leds = ALProxy("ALLeds", self.robotIp, self.robotPort)
```

```
        leds.off('green')
        leds.off('red')
        leds.off('blue')
        leds.on(self.mapping[eyeColor])


    def moveHandsRandom(self, *arg):
        motionProxy    =    ALProxy("ALMotion",    self.robotIp,
self.robotPort)
        self.stiffnessOn(motionProxy)
        JointNames    =    ["LShoulderPitch",    "LShoulderRoll",
"LElbowYaw", "LElbowRoll"]
        JointOtherNames   =   ["RShoulderPitch",   "RShoulderRoll",
"RElbowYaw", "RElbowRoll"]
        Arm1 = [randint(-45, 45),  randint(-45, 45), randint(-45,
45), randint(-45, 45)]
        Arm1 = [ x * motion.TO_RAD for x in Arm1]

        Arm2 = [randint(-45, 45),  randint(-45, 45), randint(-45,
45), randint(-45, 45)]
        Arm2 = [ x * motion.TO_RAD for x in Arm2]

        pFractionMaxSpeed = 0.4

        motionProxy.angleInterpolationWithSpeed(JointNames, Arm1,
pFractionMaxSpeed)
        motionProxy.angleInterpolationWithSpeed(JointOtherNames,
Arm1, pFractionMaxSpeed)
        motionProxy.angleInterpolationWithSpeed(JointNames, Arm2,
pFractionMaxSpeed)
        motionProxy.angleInterpolationWithSpeed(JointOtherNames,
Arm2, pFractionMaxSpeed)
```