UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

UTILIZAREA UNEI APLICAȚII DE RECUNOAȘTERE VOCALĂ PENTRU CONTROLAREA UNUI ROBOT MOBIL

LUCRARE DE DILPOMĂ

prezentată ca cerință parțială pentru obținerea titlului de Inginer în domeniul Electronică, Telecomunicații și Tehnologia Informației

Programul de studii: Microelectronică, Optoelectronică și Nanotehnologii

Conducători lucrare:
Ș.l. Dr. Ing. Horia Cucu
Prof. Dr. Ing. Corneliu Burileanu

Student:
Ariton Adela-Cristina

București
2016

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Dispozitive, Circuite şi Arhitecturi Electronice

Aprobat Director de Departament :

Prof. Dr. Ing. Gheorghe Ştefan

TEMA PROIECTULUI DE DIPLOMĂ

1. Titlul temei: **Utilizarea unei aplicatii de recunoaştere vocală pentru controlarea unui robot mobil.**

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare):
Proiectul presupune captarea semnalului vocal folosind un microfon, urmată de transmiterea acestuia către o aplicaţie de conversie a vorbirii in text. Textul rezultat va fi transmis către o aplicaţie realizată pe computerul gazdă, ce il va interpreta ca fiind o comandă introdusă de utilizator. Această comandă va fi transmisă, printr-un modul de comunicare, către microcontroller-ul robotului, care va acţiona motoarele sau va prelua date de la senzori.

3. Proiectul se bazează pe cunoştinţe dobândite în principal la următoarele 3-4 discipline: **Microcontrolere, Tehnici Avansate de Procesare Digitală a Semnalelor, Programare Obiect-Orientată.**

4. Proprietatea intelectuală asupra proiectului aparţine: : **Laboratorului de cercetare Speech and Dialogue (SpeeD) şi studentei Ariton Adela Cristina.**

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

6. Realizarea practică rămâne în proprietatea: **Laboratorului de cercetare Speech and Dialogue (SpeeD) şi studentei Ariton Adela Cristina.**

7. Data eliberării temei: **decembrie 2015**

CONDUCĂTOR LUCRARE:

Ş.L. Dr. Ing. Horia Cucu

Prof. Dr. Ing. Corneliu Burileanu

STUDENT:

Ariton Adela Cristina

DECLARAŢIE DE ONESTITATE ACADEMICĂ

Prin prezenta declar că lucrarea cu titlul "Utilizarea unei aplicații de recunoștere vocală pentru contolarea unui robot mobil", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii „Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* in domeniul Inginerie Electronică şi Telecomunicaţii, programul de studii *Microelectronică, Optoelectronică şi Nanotehnologii* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

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

Bucureşti, *2016* Ariton Adela-Cristina

CUPRINS

LISTĂ DE FIGURI

LISTĂ DE TABELE

LISTĂ DE ACRONIME

B

BSD – Berkeley Software Distribution

C

ChER – Character Error Rate

CPR – Count Per Revolution

D

DCM – Direction Cosine Matrix

F

FSG – Finite State Grammar

G

GMM – Gaussian Mixture Model

GPS – Global Positioning System

H

HMM – Hidden Markov Models

I

I/O – Input/Output

I2C – Inter-Integrated Circuit

IMU – Inertial Measurement Unit

IP – Internet Protocol

L

LAN – Local Area Network

LDA – Linear discriminant analysis

LED – Light Emitting Diode

M

MDI – Multiple Document Interface

MFCC – Mel-Frecquency Cepstrum Coefficients

MLLR – Maximum likelihood linear regression

MLLT – Maximum likelihood linear transform

P

PLP – Perceptual Linear Prediction

PPR – Pulses Per Revolution

R

RAV – Recunoașterea Automată a Vorbirii

RVC – Recunoașterea Vorbirii Continue

S

SER – Sentence Error Rate

T

TCP – Transmission Control Protocol

U

UART – Universal Asynchronous Receiver Transmitter

V

VTLN – Vocal tract length normalization

W

WER – Word Error Rate

Wi-Fi – Wireless Fidelity

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIA ALEGERII TEMEI

Vorbirea este, dintotdeauna, cel mai natural și eficient mijloc de comunicare al ființelor umane. De mii de ani oamenii transmit și recepționează informații prin intermediul vorbirii. Datorită eficienței și rapidității acestui mijloc de comunicare, oamenii au simțit nevoia de a comunica în acest mod și cu calculatorul sau cu alte dispozitive cu care interacționează. Mai mult decât atât, există numeroase aplicații în care oamenii au nevoie să-și folosească mâinile și ochii pentru altceva, vorbirea rămânând singura lor opțiune de a fi eficienți în dialogul cu calculatorul.

Astfel, a luat naștere domeniul de recunoaștere automată a vorbirii (RAV), care face parte din domeniul mai larg de recunoaștere a formelor. În prezent, acesta este un domeniu de mare interes, întrucât reprezintă un pas important în evoluția tehnologiei și implicit, în simplificarea și eficientizarea sarcinilor oamenilor.

Un alt domeniu important care a contribuit la evoluția tehnologiei îl reprezintă robotica. Roboții au în principiu, același rol de a facilita sau chiar de a reduce anumite sarcini îndeplinite până acum de către oameni. Mai mult decât atât, roboții pot ajuta persoanele cu dizabilități să aibă o viață

mult mai apropiată de cea „normală” și pot reduce chiar riscul pierderii de vieți omenești , fiind trimiși în locul oamenilor, în misiuni de explorare specifice domeniului militar. Aceștia au adus îmbunătățiri semnificative și în domeniul industrial, medical și cel aero-spațial, reprezentând o forță de muncă ce nu „obosește” niciodată și un înlocuitor în situații ce prezintă riscuri asupra vieții sau sănătății unei persoane.

Cele două domenii, cel al recunoașterii automate a vorbirii și cel al roboticii, se pot îmbina maximizând eficiența și extinzând domeniile de aplicabilitate.

Toate afirmațiile de mai sus, reprezintă motive obiective care, alături de motive subiective precum orientarea către o profesie în acest domeniu și preferințele în materie de discipline studiate pentru “Arhitectura Microprocesoarelor” și “Microcontrolere”, constituie motivația alegerii ca temă pentru această lucrare “Utilizarea unei aplicații de recunoaștere vocală pentru controlarea unei platforme robotice mobile”.

1.2 Obiectivele Lucrării

Lucrarea de față și-a propus atingerea următoarelor obiective:

- familiarizarea cu structura și modul de funcționare ale robotului Jaguar 4x4 descrise în detaliu în capitolul 2, care conține atât o descriere generală a robotului și a domeniilor lui de aplicabilitate, cât și o descriere a structurii hardware și a programului cu ajutorul căruia sunt transmise și recepționate datele.
- înțelegerea și adaptarea aplicației C# cu care a venit robotul, la cerințele lucrării.

  Pentru îndeplinirea acestui obiectiv am considerat utile realizarea unei interfețe de comunicare cu robotul mai prietenoasă și înlăturarea acelor secvențe de cod, irelevante pentru demonstrația practică (de exemplu: recepționarea datelor de la modulul GPS care nu pot fi transmise în interiorul clădirilor, ci doar în exterior) .
- realizarea unui sistem de recunoaștere de vorbire independent de vorbitor detaliat in capitolele 3 și 4. Am creat mai întâi un sistem simplu de recunoaștere de vorbire dependent de vorbitor, la care am obținut rezultate foarte bune, ce vor fi prezentate de asemenea, în capitolul 4.
- controlul robotului prin comenzi vocale preluate de la un microfon conectat la calculatorul gazdă. Semnalul audio ce conține vorbire preluat de microfon este transmis către serverul de dezvoltare pe care am realizat sistemul de recunoaștere, semnalul fiind analizat de sistemul care va trimite ca raspuns în aplicația C#, transcrierea textuală a comenzii rostite. Aceasta va fi interpretată tot de aplicația C#, care va afișa datele primite de la senzori sau va acționa motoarele, în funcție de comanda primită.
- ultimul capitol, capitolul 6, prezintă concluzii și obiective de atins în continuarea acestei lucrări.

CAPITOLUL 2

PLATFORMA ROBOTICĂ JAGUAR 4X4

2.1 DESCRIERE GENERALĂ

Platforma robotică mobilă Jaguar 4x4 Wheel cu braț mobil este concepută atât pentru aplicații de interior, cât și pentru operațiuni de exterior ce necesită suprafețe mari de explorare și manevrare mai rapidă. Brațul este unul robust, cu consum redus de putere, mărime compactă și greutate mică. Acesta are 3 grade de libertate (3 "încheieturi") și un clește de prindere ce poate ridica un corp ce cântărește până la 4 kg. Camera color montată pe braț este foarte utilă în aplicații de detecție și prindere de obiecte.

Caracteristici ale robotului Jaguar 4x4:

- are 4 roți controlate individual de câte un motor puternic (105W)
- este construit să meargă pe teren accidentat
- este capabil să urce trepte cu înălțimea de până la 11 cm
- poate urca rampe de până la $45^{0.}$
- cântărește aproximativ 33 kg
- poate atinge o viteză de până la 7 km/h

- are o strucură compactă, rezistentă la apă și condiții de temperatură extreme (-30^0 C , $+40^0$ C).
- conexiune wireless
- poate merge pe nisip, piatră, beton, pietriș, iarbă, sol și altele
- are 2 camere de înaltă rezoluție
- scaner laser pentru detecția de obiecte

Platforma robotică dispune de un senzor GPS pentru exterior și de un modul senzorial IMU (Giroscop/Accelerometru/Compas) pentru navigare autonomă. [1]

În figurile urmatoare este ilustrat robotul Jaguar 4x4 cu brat mobil, împreună cu componentele lui principale.

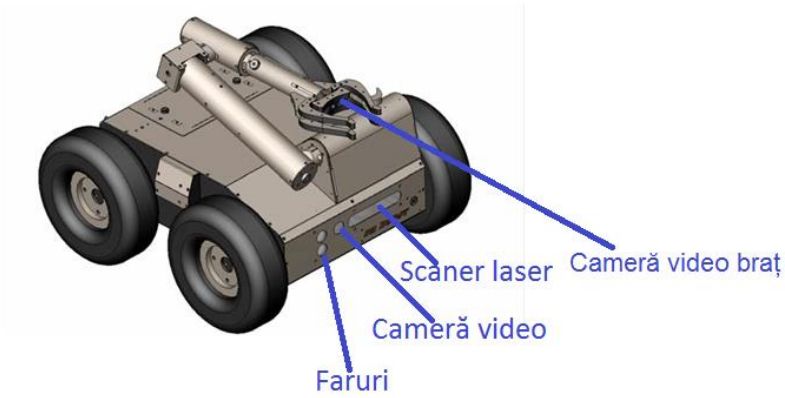


Figura 1. Robotul Jaguar 4x4 cu braț mobil (vedere lateral – față)

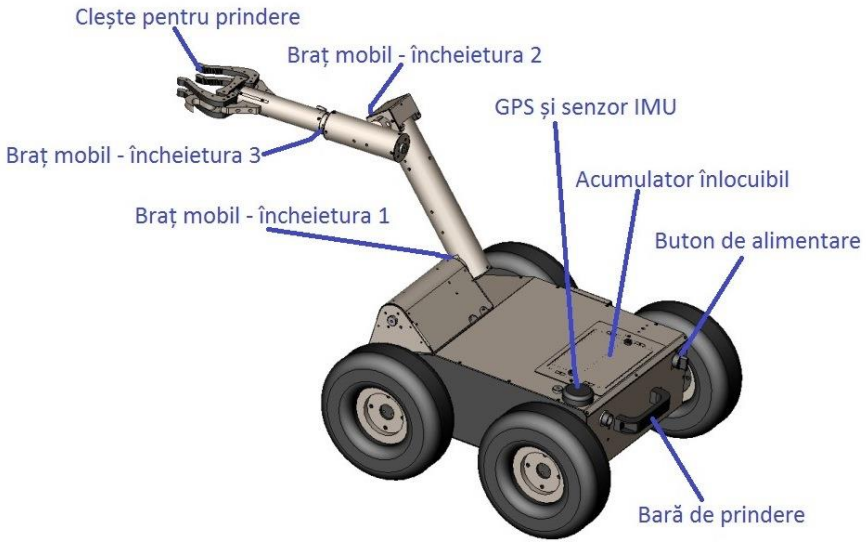


Figura 2. Robotul Jaguar 4x4 cu braț mobil (vedere lateral – spate)

2.2 COMPONENTE HARDWARE

Platforma robotică mobilă Jaguar 4x4 deține o gamă largă de componente hardware care ajută la crearea mai multor tipuri de aplicații. Structura hardware a robotului este destul de complexă, înglobând mai multe tipuri de senzori (scaner laser, modul senzorial IMU, encoder, etc.) , actuatori și module de rețea pentru a facilita capacitatea de comunicare.

În tabelul următor sunt menționate principalele componente hardware ale robotului Jaguar 4x4.

Tabelul 1. Componentele de bază ale robotului Jaguar 4x4 cu braț mobil [1]

JAGUAR 4x4W-ME	Carcasă Jaguar 4x4 Wheel (include motoarele și encoder-ele)	1
JAGUAR-ARM	Braț mobil Jaguar	1
PMS5006	Controler de mișcare și senzori	1
WFS802	Modul de rețea	2
DMD1202	Driver de motoare DC cu canal dual 10A	4
OGPS501	GPS de exterior cu rată de update de 5Hz	1
IMU9002	Senzor IMU (Accelerometru/Giroscop/Compas)	1
WRT802G	Router wireless 802.11N	1
BPN-LP-10	Acumulator LiPo de 22.2 V	1
LPBC5000	Încărcător LiPo de 2A	2
GPC0010	Gamepad Controler	1
PMCHR12	Placă de alimentare DC-DC	1

Diagrama prezentată în continuare reprezintă arhitectura hardware a sistemului și ilustrează inter-conexiunea între modulele și componentele de bază ale platformei robotice mobile Jaguar 4x4.

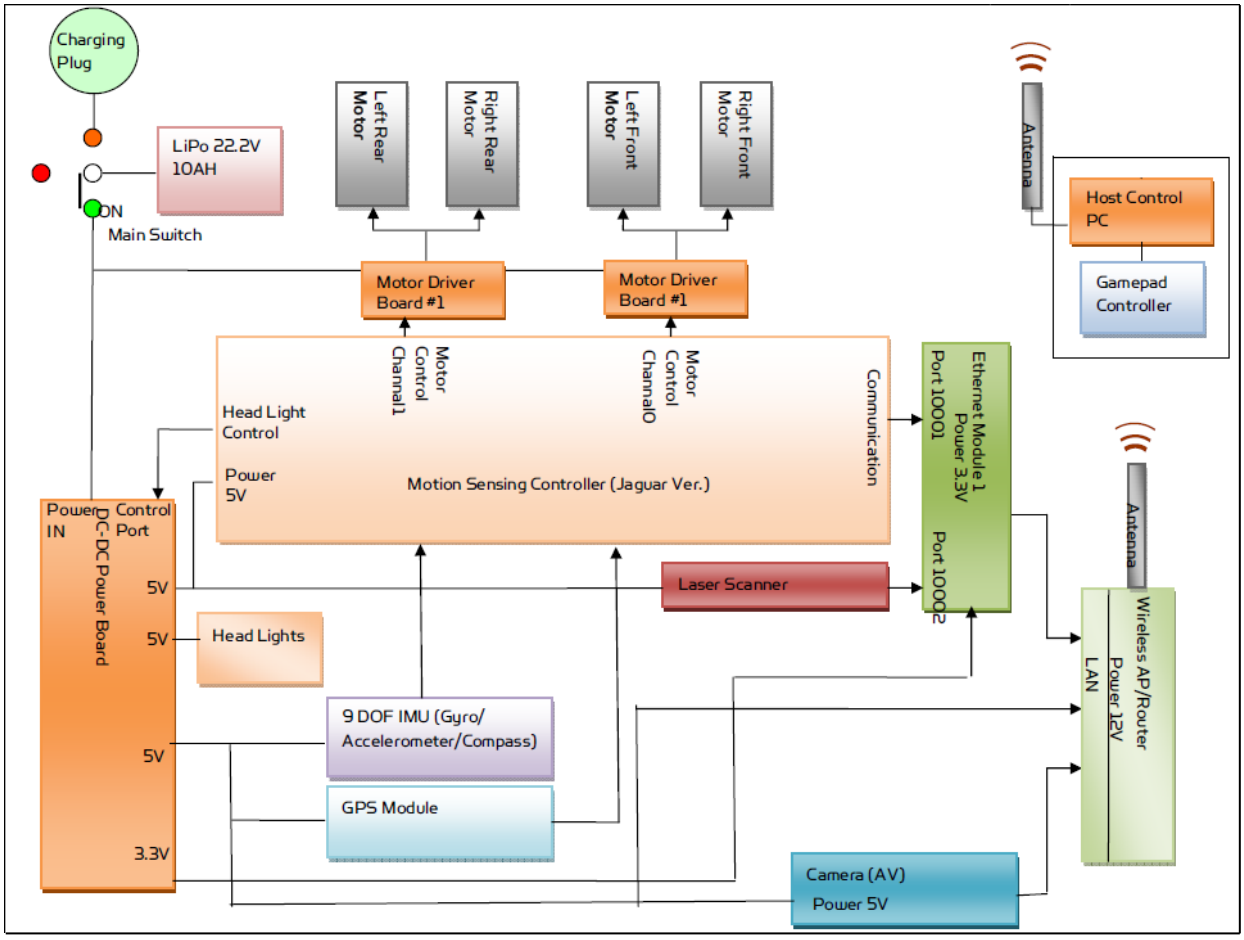


Figura 3. „Arhitectura hardware a robotului Jaguar 4x4" [1]

Se observă în diagramă modulele de funcționare și componentele principale ale robotului: Modulul Ethernet cu cele două porturi la care se conectează controler-ul de mișcare, respectiv scaner-ul laser. Modulul Ethernet se conectează la router-ul wireless care deține o antenă pentru o arie mai mare de acoperire.

Controler-ul de mișcare și senzori conține două canale (canalul 0 și canalul 1) la care sunt conectate cele două drivere de motoare ce cotrolează motoarele astfel: primul controlează motoarele stânga-spate și dreapta-spate, iar celălalt driver controlează motoarele stânga-față și dreapta-față. Acest controler mai conține și un modul de control al farurilor, care este conectat la portul de control al plăcii de alimentare.

Placa de alimentare primește la intrare 22.2V de la baterie, când comutatorul se află pe poziția "ON" și oferă la ieșire o tensiune de 5V de la care se alimentează controler-ul, scanner-ul laser, farurile, camera, router-ul wireless și, de asemenea, senzorul IMU și GPS-ul care sunt conectate la controller, și o tensiune de 3.3V care alimentează modulul Ethernet.

Informațiile se transmit wireless către calculatorul gazdă sau către gamepad controller.

O altă componentă hardware esențială pentru funcționarea sistemului este **encoder-ul optic cu incrementare.** Acestea sunt module care ajută la măsurarea distanței parcurse pentru un motor și a vitezei unui motor. Encoder-ele absolute returnează un număr pe mai mulți biți (în

funcție de rezoluție), pe când cele cu incrementare trimit pulsuri când se rotesc. Numărul de revoluții care au avut loc la mișcarea motoarelor se poate transmite numărând aceste pulsuri. După numărul de pulsuri dintr-o perioadă de timp sau din intervalul de timp scurs între două pulsuri poate fi determinată viteza de rotație. Deoarece encoder-ele cu incrementare sunt dispozitive digitale, ele vor măsura viteza și distanţa cu foarte mare acuratețe. Este necesar însă să diferențiem modul de numărare a pulsurilor, întrucât motoarele se pot mişca atât în spate, cât şi în faţă, astfel că acestea vor decrementa sau incrementa un numărător din aplicație.

Encoder-ele în cuadratură au două canale, A și B, ca în figura de mai jos. Aceste canale sunt defazate electric cu 90^0. În acest mod, direcția de rotație se poate afla prin monitorizarea relației de fază dintre canalele A şi B.

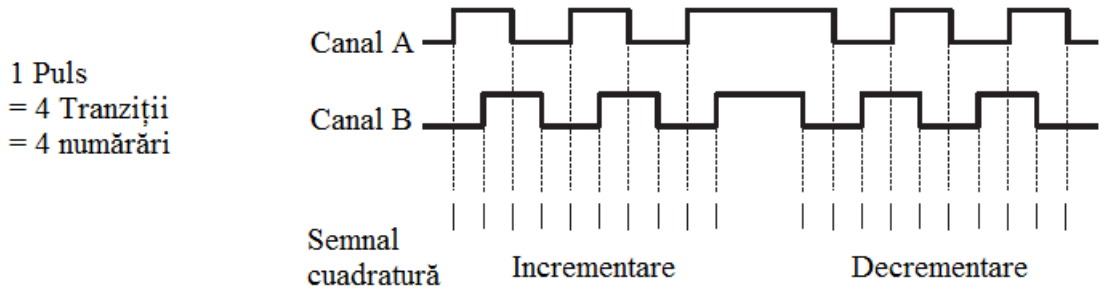


Figura 4. Semnalele de la ieșirea encoder-ului în cuadratură

În imaginea de mai jos este prezentat modul tipic în care se construiește un encoder în cuadratură. Cât timp discul se rotește în fața unei măști staționare, acesta blochează lumina provenită de la un LED. Lumina care trece prin mască este recepționată de un foto-detector. Pentru ca pulsurile să fie defazate cu 90^0, se plasează două foto-detectoare, unul lângă celălalt, pentru ca lumina care trece prin mască să lovească un detector și apoi pe celălalt, astfel producându-se defazajul.

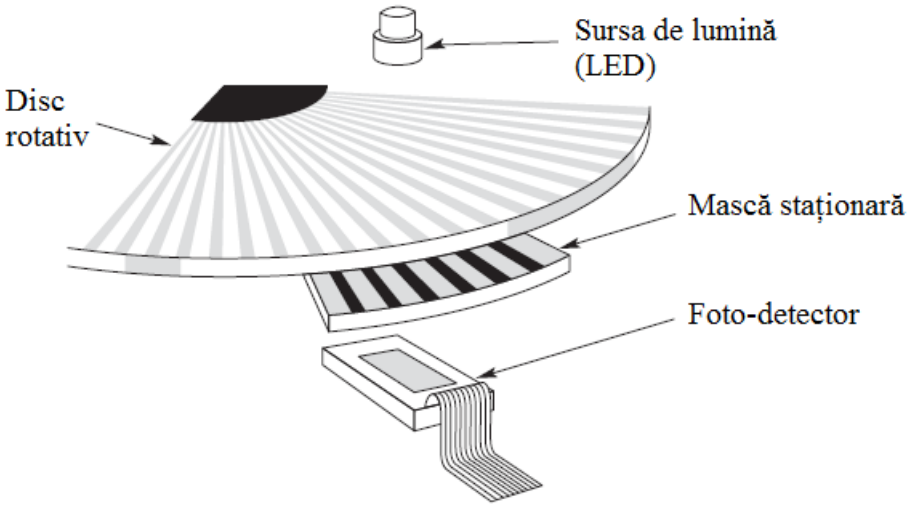


Figura 5. Modul tipic de construcție a encoder-ului în cuadratură

O altă componentă hardware importantă a robotului Jaguar 4x4 și poate cea mai complexă, este **controler-ul PMS5006**, care funcționează ca centru principal de control al robotului. Acesta comunică cu controler-ul master printr-un port serial. Pentru a comunica cu acest controler, se poate stabili un socket TCP de la calculatorul gazdă folosind IP-ul : 192.168.0.XX și portul 10001. (XX- este diferit de la un robot la altul. În cazul meu, XX a fost 60).

Acest controler poate controla până la 4 drivere de motoare și poate citi curentul de la motoare, temperatura motoarelor, tensiunea, puterea de ieșire, encoder-ele motoarelor și status-ul driver-ului.

Pentru robotul Jaguar 4x4, driver-ul de motoare 1 este presetat să comande motoarele stânga - față și dreapta - față și driver-ul de motoare 2 să comande motoarele stânga - spate și dreapta – spate, ca în tabelul următor: [4]

Tabelul 2. Controlul motoarelor la robotul Jaguar 4x4

Driver de motoare 1	Canalul 0	Motor stânga - față	Encoder 1
	Canalul 1	Motor dreapta - față	Encoder 2
Driver de motoare 2	Canalul 0	Motor stânga - spate	Encoder 1
	Canalul 1	Motor dreapta - spate	Encoder 2

Controler-ul va citi toate datele de la senzorul IMU prin portul I2C.

Firmware-ul (program special, de mici dimensiuni, care asigură comanda și controlul unor aparate și dispozitive de o anumită complexitate) de pe placă va estima orientarea robotului folosind un algoritm DCM și datele preluate de la senzorii IMU și GPS.

Mesajele recepționate de la senzori sunt:

- Mesajele GPS

Mesajele senzorului GPS vor fi trimise de controler-ul PMS5006 către controler-ul master.

Exemplu de mesaj transmis de senzorul GPS:

$GPRMC, 220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W*70

1 2 3 4 5 6 7 8 9 10 11 12

Tabelul 3. Exemplu de mesaj transmis de senzorul GPS

1	220516	Amprenta temporală
2	A	Validitatea: A-Valid, V-Nevalid
3	5133.82	Latitudine curentă
4	N	Nord/Sud
5	00042.24	Longitudine curentă
6	W	Est/Vest
7	173.8	Viteză in Nod-uri
8	231.8	Cursul real
9	130694	Data
10	004.2	Variația
11	W	Est/Vest
12	*70	Checksum

- Mesaje recepționate de la senzorii și driver-ele motoarelor

  Controler-ul PMS5006 colectează datele de la driver-ul de motoare și le transmite controler-ului master. Așa cu am mai precizat, controler-ul nostru poate controla până la 4 drivere de motoare, ceea ce înseamnă că mesajele vor începe cu:

  - "MM0" –> datele sunt primite de la Driver-ul de motoare 1
  - "MM1" –> datele sunt primite de la Driver-ul de motoare 2
  - "MM2" –> datele sunt primite de la Driver-ul de motoare 3
  - "MM3" –> datele sunt primite de la Driver-ul de motoare 4

- Mesajele de date recepționate de la senzorul IMU

Controler-ul PMS5006 va transmite toate datele de la modulul senzorial IMU și valoarea estimată a orientării. Orice pachet de date care va începe cu "#" va fi asociat unui mesaj de date IMU.

- Comenzi de control:

  - Comanda System "PING"

  Formatul: PING

Controler-ul master trebuie să trimită acest mesaj către PMS5006 la o rată de aproximativ 4Hz, altfel că PMS5006 va opri toate motoarele – consideră că a pierdut comunicarea cu controler-ul master.

  - Comanda GPIO

  Formatul SYS GPO n:

  "n" este un număr pe 8 biți, iar această comandă va controla un port I/O.

- Comanda de control de putere

Formatul: SYS MMC n

”n” este un număr pe 8 biți. Această comandă va controla un canal de putere de pe controler-ul PMS5006. Doar bit-ul 7 este disponibil pentru robotul Jaguar (restul biților fiind rezervați), bit7 = 1 va aprinde luminile, iar bit7 = 0 va stinge luminile.

- Comanda de control GPS

Formatul: EGPS n

Această comandă va determina controler-ul să activeze sau să dezactiveze GPS-ul. În algoritmul DCM, pentru n=1=> enable GPS, pentru n=0 => disable GPS.

- Setare valoare inițială DCM

Formatul: DCM n

Această comandă va seta valoarea inițială pentru DCM. “n” este valoarea direcției setată între -180 ~ +180.

- Comenzi de control pentru motoare

Întrucât PMS5006 poate controla până la 4 drivere de motoare, există 5 tipuri de comenzi:

Tabelul 4. Tipuri de comenzi pentru motoare

Comandă motor	Driver-ul care primește comanda de la controler-ul PMS5006
MM0	1
MM1	2
MM2	3 (nu este folosit în cazul nostru)
MM3	4 (nu este folosit)
MMW	1 și 2 în același timp.

Comanda ”MMW” va fi folosită dacă este nevoie de control pentru toate motoarele simultan.

Se pot trimite toate comenzile de control și configurare listate în manualul SDC2130, spre exemplu:

“MMW !M 200 -200” va controla toate cele 4 motoare în același timp și va determina robotul să se miște în față.

“MM0 !G 0 200” va controla doar motorul stânga - față.

“MM0 !G 1 -200” va controla doar motorul dreapta - față.

“MM1 !G 0 200” va controla doar motorul stânga - spate.

“MM1 !G 1 -200” va controla doar motorul dreapta - spate.

“MMW !EX” va opri de urgență toate motoarele robotului Jaguar 4x4.

“MMW !MG” va reda accesul la controlul tuturor motoarelor. [4]

2.3 COMPONENTE SOFTWARE

Aplicația de control al platformei robotice Jaguar 4x4 rulează pe un calculator gazdă și trimite comenzi prin modulele de comunicații Wi-Fi către controler-ul PMS5006 al robotului, pe care l-am prezentat în capitolul 2.2. Controler-ul le interpretează și acționează motoarele corespunzător. Aplicația inițială demonstrativă de control creată de Dr. Robot Inc. [] este o aplicație de tip Windows, scrisă în limbajul de programare C#, folosind Microsoft Visual Studio 2010. Din acest motiv, am ales ca aplicația finală pentru acest proiect să aibă la bază același limbaj și mediu de programare ca și aplicația demonstrativă (C#).

Schema funcțională a sistemului este prezentată în figura de mai jos:

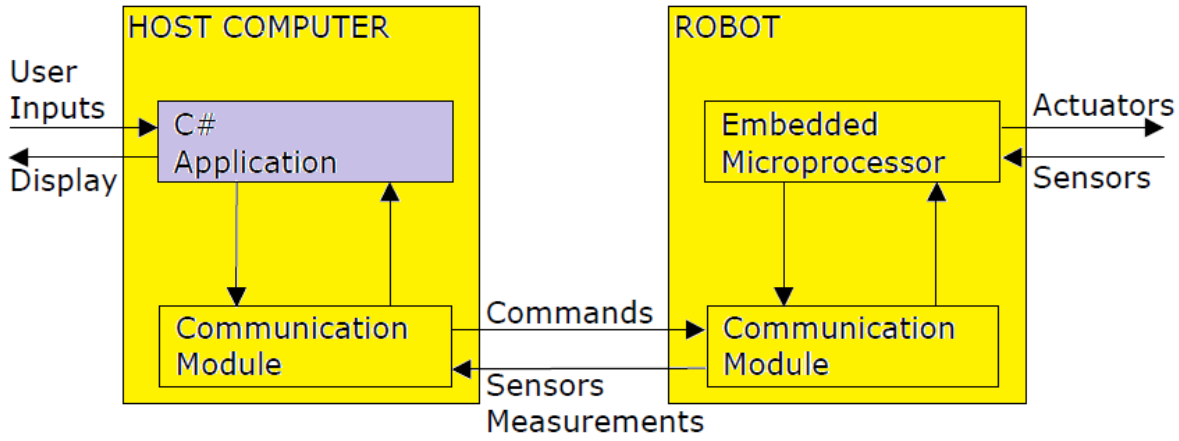


Figura 6. Schema bloc a sistemului

Pe calculatorul gazdă rulează o aplicație C# care transmite semnale de comandă către microprocesorul embedded al robotului folosind o conexiune wireless (între computer și router-ul wireless al robotului). Recepționând aceste semnale, microprocesorul va acționa motoarele. De asemenea, microprocesorul primește și date de la senzori, pe care le transmite mai departe către aplicația C# (prin aceeași conexiune wireless). Aplicația decodează mesajele primite și folosește datele de la senzori fie pentru a le afișa, fie pentru a lua decizii ce influențează controlul actuatorilor.

În subcapitolele următoare voi prezenta diferitele tehnologii utilizate în dezvoltarea acestei aplicații.

2.3.1. Framework-ul .NET

Deși se bazează pe limbaje de programare și medii de dezvoltare care au fost create cu mai mulți ani în urmă, Framework-ul .NET este mai recent, iar succesul său pe piață este în continuă ascensiune. Limbajul de programare Java a avut mult succes în medii de tip UNIX sau server web, și este limbajul ales pentru multe aplicații web în prezent.

Microsoft a încercat sa revoluționeze modul în care gândim dezvoltarea unei aplicații, rezultatul fiind Framework-ul .NET și limbajul de programare C#.

Framework-ul .NET este o interfață de programare și un mediu de execuție pentru sisteme de operare Windows, și o mare parte din framework-ul propriu-zis a fost scris în C#.

Limbajul de programare C# a fost reconstruit complet cu ideea de a păstra flexibilitatea limbajelor C și C++. Multe erori comune de tip ”runtime” în C++ sunt în C# erori de compilare.

Alte caracteristici includ tipul de date string ca fiind built-in , lipsa variabilelor globale, și integrarea erorilor critice de sistem și de aplicație într-un model comun de excepții.

Framework-ul .NET include aproape toate tehnologiile și mediile de dezvoltare care au evoluat în timp, de la COM la XML și ASP la Visual Studio. Aceste tehnologii sunt recreate și reinventate sub un singur framework. Deși compatibilitatea inversă nu s-a pierdut complet, framework-ul .NET redefinește clasele și metodele pentru aceste tehnologii și produsele care le utilizează. În particular, framework-ul include un nou suport pentru dezvoltarea aplicațiilor Windows, acces la site-uri web, comunicații remote pentru programe, interacțiuni cu baze de date, securitate și alte tehnologii. [12]

2.3.2 Formulare Windows

După cum am menționat mai devreme, toate obiectele din .NET Framework sunt scrise în C# și sunt organizate în *namespace-uri.* În dezvoltarea aplicațiilor de tip Windows se va folosi namespace-ul `System.Windows.Forms`.

Înainte de a discuta despre clase specifice, este foarte important să ne familiarizăm cu trei concepte importante în framework-ul .NET dar și în namespace-ul Windows Forms: *componente, containere și control*.

O *componentă* este un obiect care permite distribuirea între aplicații. Clasa `Component` încapsulează această noțiune, și este baza pentru majoritatea membrilor din namespace-ul Windows Forms.

Un *container* este un obiect care poate îngloba una sau mai multe componente. Un container este pur și simplu un mecanism de grupare ce asigură că seturi de componente sunt încapsulate și manipulate în moduri similare. Containerele sunt folosite în namespace-ul Windows Forms de fiecare dată când este nevoie de un grup de obiecte. Clasa `Container` încapsulează conceptul de container.

Un *control* este o componentă cu aspect vizual. În namespace-ul Windows Forms, un control este o componentă care prezintă o interfață grafică pe Windows desktop. Clasa `Control` stă la baza aplicațiilor Windows Forms.

În general, orice interfață grafică vedem pe un Windows desktop este un *control*, și orice obiect ”din spate” este o *componentă*.

Majoritatea elementelor vizuale ale unei interfețe grafice, cum ar fi butoanele, casete de dialog, casete de text, sunt reprezentate de clase de control. [12]

Namespace-uri și clase

Folosim cuvântul cheie `namespace` în program pentru a declara un nou namespace, spre exemplu `namespace JaguarControl`.

Un namespace conține unul sau mai multe tipuri, spre exemplu o clasă din program. O clasă definește o nouă abstractizare de date, astfel ea definind un nume de clasă, precum și o

colecție de membri pentru a reprezenta și operarea în clasă. O clasă este unul din tipurile posibile dintr-un namespace.

În C#, clasele suportă moștenirea singulară, ceea ce înseamnă că o clasă moștenește de la o altă clasă. În program putem defini o clasă care să moștenească din clasa `Form` ce se găsește în namespace-ul `System.Windows.Forms`.

Clasa `Form` stă la baza aplicațiilor Windows în farmework-ul .NET. Aceasta reprezintă orice tip de fereastră dintr-o aplicație, de la casete de dialog, la interfețe de documente multiple (MDI –Multiple Document Interface). Clasa `Form` introduce posibilitatea de a afișa sau așeza diferite tipuri de controale și interacționează cu fereastra de aplicație.

Clasele în .NET conțin unul sau mai mulți membri care definesc comportamentul și caracteristicile clasei. Membri unei clase pot fi niște constante, câmpuri, metode, proprietăți, evenimente, operatori, constructori și declarații de tip nested.

Constructori și metode

Pentru o anumită clasă putem defini *constructori* și *metode*. Acestea pot fi declarate cu un anumit tip de accesibilitate, C# aducând mai multe tipuri de nivele de accesibilitate ca: `public`, `protected` și `private`.

Primul membru se numește *constructor,* și funcționează la fel ca și un constructor din C++. Dacă acesta inițializează o nouă instanță a unei clase se numește *instance constructor*. Un constructor de instanță fără parametri se numește *default constructor*. C# suportă de asemenea constructori statici pentru a inițializa clasa.

Un constructor de instanță este invocat când un obiect al clasei respective este creat. În mod normal, obiectele de orice tip sunt intițializate folosind cuvântul cheie *new*. O metodă trebuie invocată explicit în program.

Un alt membru al clasei poate fi o *metodă*. O metodă este un membru care face o operație în clasa respectivă. Metodele din C# funcționează, în mare parte, ca în C++.

Clasificarea tipurilor C#

Pentru a inițializa orice tip în C# este utilizat cuvântul cheie *new*. Acesta include atât clase și structuri cât și tipuri simple ca int, sau enumerări.

Există două clasificări pentru tipuri în C#, fiecare având un anumit comportament de inițializare.

1. *Value types* conține respectivele date pentru tip. Acesta include tipuri *built-in* cum ar fi `int, char, bool` dar și structuri create folosind cuvântul cheie `struct`. Tipurile de valori sunt de obicei mici, ceea ce face ușoară stocarea lor în stivă sau în obiectul care le conține.
2. *Tipurile de referință* conțin o referință la datele tipului. Toate clasele din C# sunt tipuri de referință, ca și tipurile *built-in* `object` și `string`.

Compilatorul convertește automat tipurile de valori în tipuri de referință folosind un proces numit *boxing*. [12]

Aria de memorie rezervată pentru referințe de date se numește *heap*. Memoria alocată în heap, este recăpătată folosind *garbage collection.* Fără a intra prea mult în detalii despre acest subiect, vom considera acest garbage collector ca o metodă prin care scăpăm de pointeri și utilizări ineficiente de memorie.

2.3.2 CMU Sphinx

CMU Sphinx, este un termen general folosit pentru a descrie un grup de sisteme de recunoaștere a vorbirii dezvoltate la universitatea Carnegie Mellon. Acestea includ serii de module de recunoaștere de vorbire (Sphinx 2 - 4) și un modul de antrenare al modelului acustic (SphinxTrain).

În anul 2000, grupul Sphinx de la Carnegie Mellon a făcut publice câteva componente ale modulului de recunoaștere, incluzând Sphinx 2 și mai târziu Sphinx 3 (în 2001). Decodificatoarele de vorbire vin cu modele acustice și exemple de aplicații. Resursele disponibile includ și softaware-ul pentru antrenarea modelului acustic, compilarea modelului de limbă și un dicționar fonetic, „cmudict".

Sphinx cuprinde un număr de sisteme software, descrise în continuare:

- **Sphinx**

Sphinx este un sistem de recunoaștere a vorbirii continue, independent de vorbitor ce utilizează modele acustice Markov ascunse (HMM) și un model de limbă statistic de tip n-gram. Mai multe detalii despre acestea vor fi discutate în capitolul 3s.

- **Sphinx 2**

Este un modul de recunoaștere rapid, orientat către performanță. Sphinx 2 se focusează pe recunoașterea în timp specifică aplicațiilor de limbă vorbită. Acesta este folosit în sisteme de dialog și sisteme de învățare a limbii.

- **Sphinx 3**

Sphinx 2 folosea o reprezentare semi-continuă pentru modelul acustic (exemplu: pentru toate modelele se folosește un singur set de Gaussiene). Sphinx 3 a adoptat reprezentarea continuă HMM și a fost utilizat la început pentru recunoașterea cu mare acuratețe, dar nu în timp real. Cercetări recente (în algoritmi și hardware) au făcut Sphinx 3 să fie "aproape" în timp real. Acesta este încă în proces de dezvoltare și împreună cu SphinxTrain face accesibile numeroase tehnici de modelare moderne, precum LDA/MLLT, MLLR and VTLN, care îmbunătățesc acuratețea recunoașterii.

- **Sphinx 4**

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.

Direcțiile curente de dezvoltare includ:

- dezvoltarea unui nou model acustic
- implementarea adaptării la vorbitor (exemplu: MLLR)
- îmbunătățirea managementului configurației

- **PocketSphinx**

Este o veriune de Sphinx ce poate fi utilizată în sisteme embedded (exemplu: bazate pe un procesor ARM). PocketSphinx se află și el în plin proces de dezvoltare. [11]

2.3.3 Aplicaţii client-server

Internetul functionează pe un sistem de protocoale numit TCP/IP (Transmission Control Protocol/Internet protocol). Acestea stabilesc regulile cu ajutorul cărora două calculatoare comunicî între ele. Java implementează protocoalele de nivel superior al stivei de protocoale TCP/IP, acest lucru facilitând utilizarea protocoalelor HTTP (HyperText Transfer Protocol) și FTP(File Transfer Protocol). Astfel, programatorul va utiliza niște clase și interfețe predefinite, fără a fi necesar să cunoască detaliile de implementare a acestora.

Serverul este o aplicație ce oferă servicii clienților sosiți prin rețea. Serverele oferă o gamă variată de servicii. Cel mai cunoscut server este serverul Web, care furnizează documentele cerute de către clienți. Arhitectura client-server este instrumentul de bază în dezvoltarea aplicațiilor de rețea.

Clientul este o aplicație care utilizează serviciile oferite de către un **server**. Pentru a putea realiza acest lucru, clientul trebuie să cunoască unde se află serverul în rețea, cum trebuie comunicat cu acesta și ce servicii oferă. Cu alte cuvinte, dacă un client dorește o comunicare cu serverul, trebuie să cunoscă trei lucruri:

- adresa server
- portul server utilizat pentru comunicare
- protocolul de comunicație utilizat de server

Dacă aceste date sunt disponibile, se poate realiza comunicația cu serverul.

Porturi și socluri

Porturile și soclurile reprezintă mecanismul prin care se realizează legătura cu un server. La aceeași adresă se pot oferi diferite servicii, acestea fiind oferite la porturi diferite. Același calculator (cu o singură adresă IP) poate să ofere oricâte servicii dorește. Clienții care apelează la serviciile acestui calculator vor utiliza aceeași adresă, indiferent la care serviciu apelează, și toți clienții care doresc utilizarea unui anumit serviciu vor utiliza același port. Un număr de port este un număr înreg din intervalul 1-9999.

Un **soclu** este de fapt un nod abstract de comunicație. Soclurile reprezintă o interfață de nivel scăzut pentru comunicarea în rețea. Acestea permit comunicarea între procese aflate pe același calculator sau pe calculatoare diferite din rețea.

Mecanismul de socluri a fost definit prima dată în BSD UNIX (Berkeley Software Distribution Unix). Java suportă trei tipuri de socluri. În lucrarea de față am utilizat clasa Socket care folosește un protocol orientat pe conexiune (TCP). Soclurile utilizează fluxuri de date (streamuri) pentru a trimite și a receptiona mesaje. [6]

Modelul de comunicație orientat pe conexiune.Clasele Socket și ServerSocket

Acest model are la bază protocolul TCP. Într-o aplicație rețea întotdeauna avem două părți: partea *client* care inițializează conversația și trimite cereri, și partea *server* care primește cererile și răspunde la acestea. Clientul întotdeauna crează un soclu pentru a iniția „conversația” și trebuie să cunoască serverul căruia adresează cererea, iar serverul trebuie să fie pregătit pentru a recepționa aceste cereri. În momentul recepționării mesajului crează un soclu pe partea serverului, soclu care va facilita deservirea clientului. Atât pe partea de client cât și pe cea de server se utilizează câte un obiect de tip Socket pentru comunicare. Pe partea de server mai trebuie să creăm un obiect de tip ServerSocket, care are sarcina primirii conexiunilor și acceptarea acestora.[6]

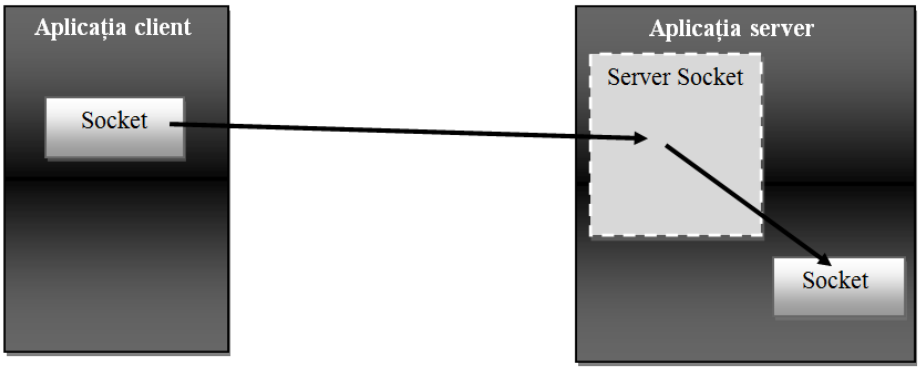


Figura 7. Comunicarea client-server folosită în lucrarea curentă

CAPITOLUL 3

SISTEM DE RECUNOAȘTERE AUTOMATĂ A VORBIRII

Procesul de recunoaștere automată a vorbirii are drept scop fundamental transformarea unui semnal audio ce conține vorbire într-un cuvânt sau o succesiune de cuvinte. Acest proces are o gamă largă de aplicabilitate, mai ales în domeniul interfețelor ”hands-free” și ”eyes-free”, unde utilizatorii au nevoie sa-și folosească ochii și mâinile pentru altceva, cea mai bună și mai eficientă soluție în interacțiunea cu calculatorul fiind vorbirea. Mai mult decât atât, procesul recunoașterii automate a vorbirii poate fi extins încercând să producă informații de natură semantică (propoziții cu sens).

Domenii de utilizare a sistemelor de recunoaștere automată a vorbirii:

- ✓ sisteme de dialog pentru call-centere
- ✓ sisteme de traducere speech-to-speech
- ✓ dictare (sisteme de transcriere speech-to-text)

  Sistemele de transcriere pot fi utilizate transcierea discuțiilor din timpul proceselor de judecată, pentru editarea documentelor de către jurnaliști, scriitori, etc.
- ✓ sisteme de autentificare (identificare de vorbitor)

Procesul pe care se bazează funcționarea acestor sisteme are la bază două etape:

1. Diarizarea – etapă ce încearcă să determine *cine a vorbit* și *momentul în care a început să vorbească*.
2. Transcrierea – etapă ce încearcă să determine *ce a spus* vorbitorul identificat.

- ✓ sisteme de securizare, etc.

Există însă o mulțime de factori ce influențează în mod negativ sarcina de recunoaștere. Acestea ar putea fi clasificate în surse de variabilitate acustică, cum ar fi: vorbitor necunoscut, calitatea microfonului, caracteristicile vorbitorului, mediu zgomotos sau/și reverberant, vorbire spontană și surse de incertitudine lingvistică: complexitatea morfologiei, dimensiunea vocabularului, spontaneitatea vorbirii, etc.

În tabelul următor sunt prezentate procentele de cuvinte incorecte obținute pentru diverse sarcini de recunoaștere.

Tabelul 5. Rezultate WER obținute pentru diverse sarcini de RAV [Jurafsky, 2009]

Sarcina de RAV	Dimensiunea vocabularului	WER[%]
TI Digits	11 cuvinte	0.55
Wall Street Journal read speech	5 000 cuvinte	3.0
Wall Street Journal read speech	20 000 cuvinte	<6.6
Broadcast News	64 000+ cuvinte	9.9
Conversational Telephone Speech	64 000+ cuvinte	20.7

Datele se referă la sisteme pentru limba Engleză.

Rata de cuvinte eronate (word error rate – WER) este criteriul de performanță standard utilizat pentru evaluarea sistemelor de RAV la nivel de cuvânt.

$$\text{WER [\%]} = \frac{\text{Erori de inserție + Erori de substituție + Erori de ștergere}}{\text{Cuvinte în transcrierea de referință}} \text{ X } 100 \qquad (1)$$

[3]

Mai multe detalii despre criteriile de evaluare a sistemului vor fi prezentate în ultimul subcapitol al acestui capitol.

3.1 ARHITECTURA UNUI SISTEM DE RECUNOAŞTERE AUTOMATĂ A VORBIRII (RAV)

Procesul de recunoaștere automată a vorbirii se bazează pe două lucruri fundamentale:

1. Recunoașterea se face utilizând o serie de parametric vocali extrași din semnalul vocal.
2. Recunoașterea se face pe baza unor modele (acustic, fonetic, lingvistic) dezvoltate în prealabil.

În figura următoare este prezentată arhitectura generală a unui sistem de RAV:

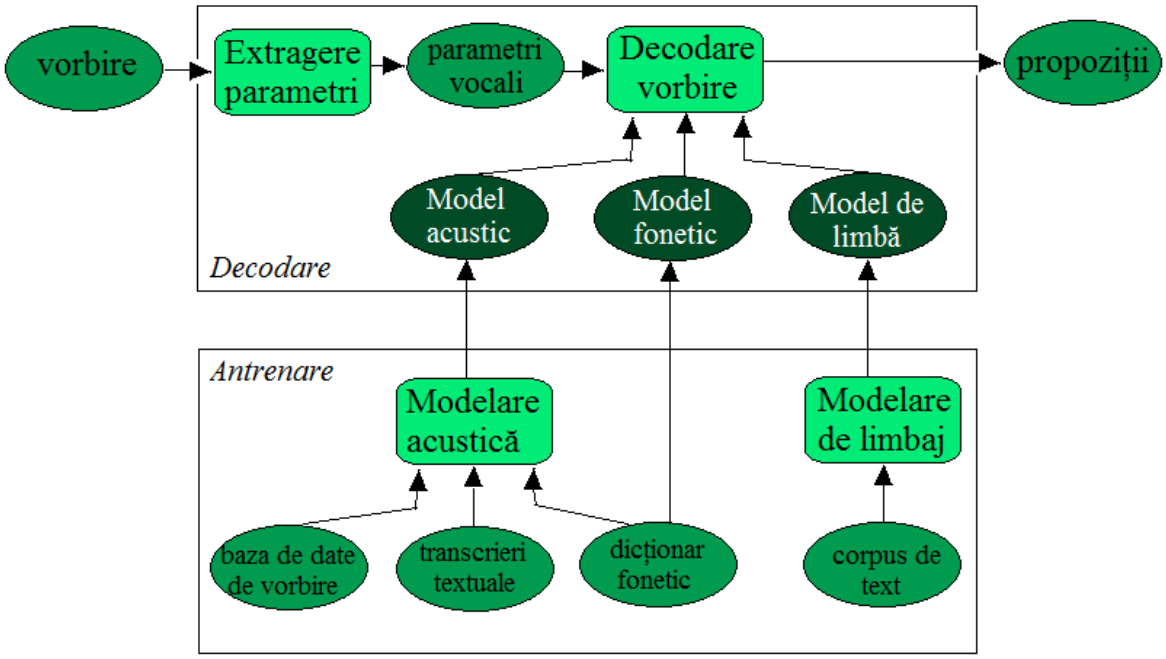


Figura 8. Arhitectura generală a unui sistem de RAV

În figura de mai sus sunt reprezentate atât procesele implicate în dezvoltarea unui sistem de recunoaștere automata a vorbirii, cât și resursele necesare creării modelelor acustice, fonetice și lingvistice.

Modelul acustic are rolul de a estima probabilitatea unui mesaj vorbit, fiind dată o succesiune de cuvinte. Întrucât numărul de cuvinte diferite dintr-o limbă este foarte mare și fiecare sarcină de recunoaștere automată a vorbirii are un vocabular de cuvinte diferit pentru care nu există modele deja antrenate și nici date de antrenare disponibile, în sistemele actuale de recunoaștere a vorbirii continue (RVC) modelul acustic nu folosește cuvintele ca unități acustice de bază. În locul cuvintelor, se folosesc unități acustice de bază sub-lexicale precum fonemele sau, mai recent, senonele (unități sub-fonemice).

Deci, în timpul procesului de decodare, modelele pentru cuvinte și apoi modelele pentru succesiuni de cuvinte se formează prin conectarea unui set de modele pentru foneme/senone care formează modelul acustic. Ele sunt utilizate apoi pentru a estima probabilitatea ca mesajul rostit să fie format dintr-o succesiune sau alta de cuvinte. Într-o manieră statistică, putem formula procesul de transformare a vorbirii în text astfel: ”Care este cea mai probabilă secvență de cuvinte W* în limba L, dat fiind mesajul vorbit X?” [3]

O reprezentare formală ar fi următoarea ecuație:

$$W^* = \arg\max_W p(W \mid X) = \arg\max_W \frac{p(X \mid W)\, p(W)}{p(X)} = \arg\max_W p(X \mid W)\, p(W) \quad (2)$$

unde funcția **argmax** selectează argumentul ce maximizează probabilitatea secvenței de cuvinte.

Aceasta este defapt o dezvoltare ce are la bază regula lui Bayes și s-a facut ținând cont de faptul că probabilitatea mesajului vorbit p(X) este independent de secvența de cuvinte W.

Modelul de limbă este utilizat tot în procesul de decodare, pentru a estima probabilitățile tuturor secvențelor de cuvinte din spațiul de căutare. Rolul modelului de limbă este, în general, de a estima probabilitatea ca o secvență de cuvinte $W=w_1, w_2, \ldots, w_n$, să fie o propoziție cu sens a limbii. Aceste probabilități sunt foarte utile în procesul de decizie al modelului acustic.

Modelul fonetic are rolul de a conecta cele două modele (modelul acustic și modelul de limbă).Acesta este, în general, un dictionar de pronunție care asociază fiecărui cuvânt din vocabular una sau mai multe secvențe de foneme corespunzătoare, ce arată modul în care se poate pronunța cuvântul repectiv.

În figura 8 observăm și resursele de care are nevoie fiecare dintre cele trei modele (acustic, fonetic și lingvistic). Astfel, deducem faptul că modelul acustic se construiește strict pe baza unui set de clipuri audio înregistrate în prealabil, pe baza transcrierilor textuale corespunzătoare a mesajului rostit și pe baza unui dictionar fonetic ce conține toate cuvintele din transcrierea textuală.

În cazul proceselor de recunoaștere a vorbirii continue cu vocabular extins se utilizează modele de limbă statistice care se construiesc, așa cum se vede și în figură, folosind corpusuri de text de dimensiuni cât mai mari și cât mai specific domeniului din care fac parte mesajele vorbite ce se doresc a fi decodate. [3]

3.2 RESURSE FONETICE, ACUSTICE ŞI LINGVISTICE NECESARE ÎN CONSTRUCŢIA UNUI SISTEM DE RECUNOAŞTERE A VORBIRII CONTINUE

În acest capitol ne vom concentra pe resursele de care avem nevoie pentru construcția unui sistem de RVC, și anume: baza de date de vorbire, transcrierile textuale, dicționarul fonetic și corpusul de text, evidențiate în figura următoare.

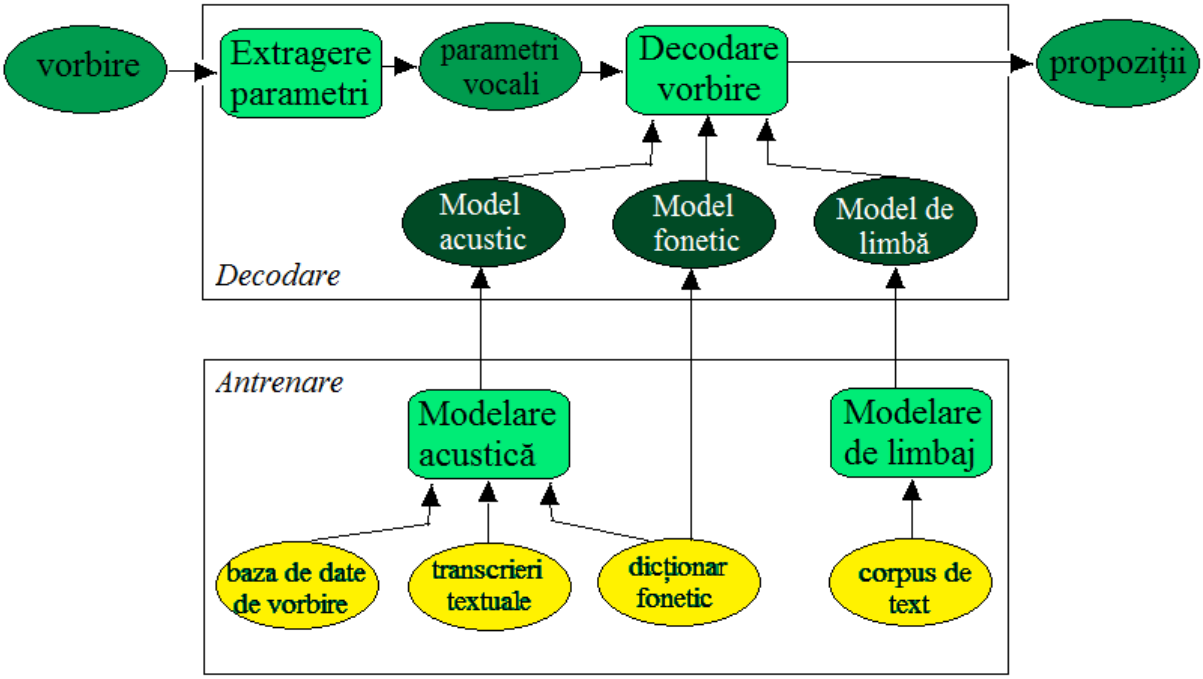


Figura 9. Resursele necesare pentru a construi un sistem de RVC

Baza de date de vorbire este construită pe baza unui set de clipuri audio înregistrate apriori.

Transcrierile textuale reprezintă o listă de cuvinte corespunzătoare tuturor cuvintelor rostite în clipurile audio.

Acestea două, împreună cu dicționarul fonetic care specifică modul în care se pronunță fiecare cuvânt din lista de transcrieri textuale, definesc procesul de modelare acustică ce dezvoltă modelele acustice necesare procesului de decodare a vorbirii. Dicționarul fonetic determină dezvoltarea modelului fonetic, de asemenea necesar decodării.

Procesul de modelare a limbajului necesită ca resursă crearea corpusului de text, și împreună cu procesul de modelare acustică și resursele aferente ale acestuia, constituie partea de **antrenare** a sistemului.

Procesul de modelare de limbaj dezvoltă modelul de limbă utilizat și el în procesul de decodare a vorbirii, alături de modelul acustic și cel fonetic. Acest proces, împreună cu parametrii vocali rezultați în urma procesului de extragere de parametri alcătuiesc partea de **decodare** a sistemului.

Astfel, după ce semnalul ce conține vorbire este trecut prin partea de decodare ce implică și partea de antrenare, acesta este transformat în propoziții.

Dicționrul fonetic

Fonemul este unitatea de sunet fundamentală a limbii vorbite, care ajută la diferențierea cuvintelor și morfemelor (prefixe, sufixe, desinențe). Prin modificarea unui fonem se poate genera un cuvânt inexistent sau un cuvânt cu alt sens.

Așa cum am mai spus, dicționarul fonetic specifică modul în care se pronunță cuvintele, cu alte cuvinte este instrumentul care face corespondența între forma fonetică și transcrierea textuală a unui cuvânt. În sistemul de recunoaștere a vorbirii continue, dicționarul fonetic face legătura între modelul acustic și modelul de limbă. Fonemele corespunzătoare limbii române sunt prezentate în tabelul de mai jos.

Tabelul 6. Fonemele limbii române [3]

Fonemul			Exemple de cuvinte	
Tip	**Simbol IPA**	**Simbol intern**	**Forma scrisă**	**Forma fonetică**
vocale	a	a	**sat**	s a t
	e	e	**mare**	m a r e
	i	i	**lift**	l i f t
	o	o	**loc**	l o c
	u	u	**șut**	s1 u t
	ə	a1	**gură**	g u r a1
	i̯	i2	**între**	i2 n t r e
vocale împrumutate	y	y	**ecru**	e c r y
	ø	o2	**bleu**	b l o2
semivocale	e̯	e1	**deal**	d e1 a l
	j	i3	**fiară**	f i3 a r a1
	o̯	o1	**oase**	o1 a s e
	w	w	**sau**	s a w
consoane	c	k2	**che**m	k2 e m
	b	b	**bar**	b a r
	p	p	**par**	p a r
	k	k	**acum**	a k u m
	ʧ	k1	**cenușă**	k1 e n u s1 a1
	g	g	**g**alben	g a l b e n
	ʤ	g1	**gi**rafă	g1 i r a f a1
	ʒ	g2	**unghi**	u n g2
	d	d	**d**ar	d a r
	t	t	**t**ot	t o t
	f	f	**f**ața	f a t1 a
	v	v	**v**apor	v a p o r
	h	h	**h**arta	h a r t a
	ʒ	j	**j**utor	a j u t o r
	ʃ	s1	**coș**	k o s1
	l	l	**l**ac	l a c
	m	m	**măr**	m a1 r
	n	n	**n**as	n a s
	s	s	**s**are	s a r e
	z	z	**z**ar	z a r
	r	r	**r**isc	r i s k
	ʦ	t1	**ț**ăran	t1 a1 r a n
consoană palatalizată	ʲ	i1	**ti**ri	t a r i1

Construcția dicționarului fonetic se poate face automat, semi-automat sau manual. În cazul utilizării unui vocabular redus (zeci-sute de cuvinte) construcția dicționarului fonetic se poate poate face manual, într-un timp suficient de scurt.

Dicționarul fonetic este utilizat, așa cum se vede și în figura 9, pentru antrenarea modelului fonetic.

Baza de date de vorbire

O bază de date de vorbire completă trebuie să conțină trei componente:

-un set de clipuri audio ce conțin vorbire

-un set de fișiere text cu transcrierea textuală a cuvintelor din clipurile audio

-informații suplimentare privind identitatea vorbitorului, domeniul vorbirii, etc.

Aspectele principale ale bazei de date de vorbire, care determină calitatea acesteia sunt:

-dimensiunea bazei de date (numărul de vorbitori, numărul de ore de vorbire)

-variabilitatea (calitatea înregistrărilor, variabilitatea vorbitorilor, zgomotul de fundal, reverberația)

-stilul vorbirii (vorbire spontană, vorbire continuă citită, cuvinte izolate).

În tabelul următor sunt sugestiile date de CMU Sphinx pentru dimensiunea bazei de date de vorbire.

Tabelul 7. Dimensiunile sugerate de CMU Sphinx pentru bazele de date de vorbire

Sarcina de RAV	Sistem dependent de vorbitor	Sistem independent de vorbitor
Comandă și control (vocabular redus)	1 oră de înregistrări, 1 vorbitor	5 ore de înregistrări, 200 de vorbitori
Dictare (vocabular extins)	10 ore de înregistrări, 1 vorbitor	50 ore de înregistrări, 200 de vorbitori

Baza de date de vorbire este utilizată, așa cum se poate deduce din figura 9, pentru antrenarea modelului acustic. [3]

3.3 PRINCIPIILE DE BAZĂ ALE MODELĂRII ACUSTICE

Sistemele de recunoaștere a vorbirii continue nu estimează direct probabilitatea mesajului vorbit, ci probabilitatea unor unități de vorbire mai mici, în general foneme. S-a demonstrat că această abordare poate fi implementată cu succes utilizând modele Markov ascunse (HMM – Hidden Markov Models).

Modelele Markov ascunse sunt automate probabilistice cu un număr finit de stări care pot fi combinate în mod ierarhic în scopul contruirii unor modele pentru cuvinte din modele pentru unități de vorbire mai scurte. [3]

Parametri acustici

Întrucât semnalele ce conțin vorbire sunt nestaționare, Modelele Markov ascunse nu modelează semnalul vocal, ci niște parametrii extrași prin ferestruirea semnalului inițial în domeniul timp. Din fiecare fereastră se pot extrage parametri de tip spectral sau de tip cepstral. Cei mai utilizați sunt parametri de tip cepstral-perceptual. Exemple de parametri de acest tip:

- parametri Mel-cepstrali (Mel-Frecquency Cepstrum Coefficients – MFCCs)
- parametri perceptuali obținuți prin predicție liniară (Perceptual Linear Prediction – PLP)

Se preferă reprezentarea cepstrală datorită decorelației coeficienților, spre deosebire de gradul de corelație ridicat al coeficienților spectrali vecini.

Figura următoare ilustrează procesul de extragere a parametrilor MFCC.

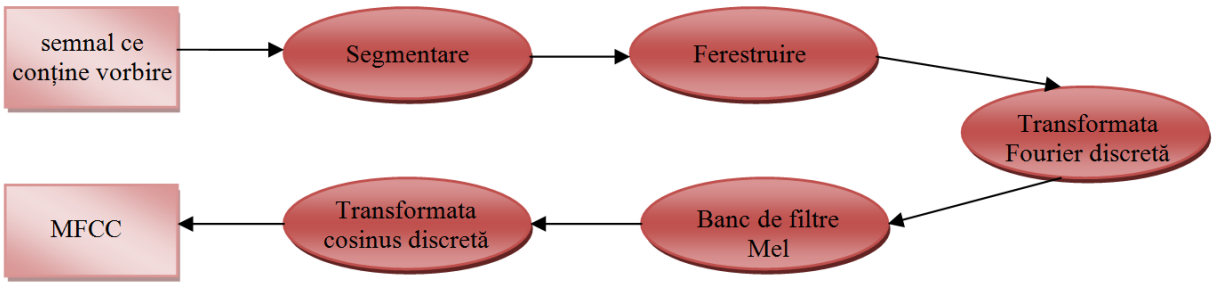


Figura 10. Procesul de extragere a parametrilor MFCC

3.4 PRINCIPIILE DE BAZĂ ALE MODELĂRII LIMBAJULUI NATURAL

Rolul modelului de limbă este acela de a estima probabilitatea ca o anumită secvență de cuvinte să fie o propoziție cu sens pentru sarcina de recunoaștere respectivă. Cu alte cuvinte, din două propoziții similare din punct de vedere acustic precum *"Florile vor înflori mai devreme anul acesta."* și *"Flori le vor în flori a nul acest a."* atribuie o probabilitate mult mai mare primei propoziții, întrucât aceasta are sens.

Modele de limbă statistice

Cele mai noi sisteme de RVC cu vocabular mare utilizează modele statistice de tip n-gram. Acestea se construiesc pe baza unor corpusuri mari de text, fiind utilizate apoi în procesul de decodare pentru a calcula probabilitățile secvențelor de cuvinte propuse de modelul acustic.

Problema estimării probabilităţii secvenţei de cuvinte W este împărţită în mai multe probleme de estimare a probabilităţii unui singur cuvânt, fiind dată secvenţa de cuvinte anterioare lui. Acest lucru se traduce prin următoarea relaţie:

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

Din motive de eficienţă, se alege un număr finit „m" de cuvinte anterioare de care depinde cuvântul curent, adică se presupune că doar un număr limitat de cuvinte afectează probabilitatea următorului cuvânt. Aceasta conduce la modelul de limbă convenţional n-gram. Uzual, se foloseşte modelul de limbă trigram, care ia în considerare doar ultimele două cuvinte pentru a-l prezice pe al treilea. De asemenea, pot fi folosite şi modele de limbă de tip bigram sau de ordine superioare.

În cazul unui model de limbă bigram, spre exemplu, se estimează probabilităţile $p(w_j \mid w_i)$ pentru fiecare pereche de cuvinte(w_i, w_j).Aceste probabilităţi se calculează folosind principiul „likelihood", adică se înumără de câte ori cuvântul w_i este urmat de cuvântul w_j, în comparaţie cu alte cuvinte.

$$p(w_j \mid w_i) = \frac{count(w_i, w_j)}{\sum_w count(w_i, w)} \qquad (3)$$

Estimarea acestor probabilităţi este cu atât mai corectă cu cât baza de date de antrenare este mai mare.

3.5 EVALUAREA SISTEMELOR DE RECUNOAŞTERE A VORBIRII CONTINUE

Pentru ca un sistem să aibă rezultate cât mai bune, adică să transcrie cât mai corect mesajul vorbit, este esenţial ca baza de date de evaluare ce conţine clipuri audio (folosite doar pentru evaluare în cazul unui sistem independent de vorbitor) să reprezinte cât mai fidel sarcina de recunoaştere la care a fost supus sistemul. Acest lucru înseamnă că următoarele carecteristici trebuie să fie aceleaşi pentru sarcina de recunoaştere şi pentru baza de date de evaluare:

-Mediul acustic: zgomotos, nezgomotos

-Caracteristicile vorbitorului: nativ/non-nativ, tânăr/în vârstă, un singur vorbitor/mai mulţi vorbitori

-Domeniul vorbirii: general (vorbire liberă pe orice temă) sau particular (pe teme istorice/culturale/ştiinţifice)

-Tipologia vorbirii: vorbire continuă spontană, vorbire continuă citită sau cuvinte izolate. [3]

Evaluarea completă a sistemului de recunoaştere a vorbirii

Evaluarea completă a sistemului de recunoaştere a vorbirii se face prin compararea în mod automat a transcrierilor textuale ale clipurilor audio de evaluare (transcrieri de referinţă) cu transcrierile rezultate în urma procesului de decodare a clipurilor audio de evaluare (transcieri ipotetice).

3.5.1 Criterii de performanţă utilizate în sistemele de recunoaştere a vorbirii

În cazul în care transcrierea ipotetică a unei propoziţii devine inutilă atunci când conţine un cuvânt greşit (exemplu: comenzi scurte, recenunoaşterea unor cifre ce reprezintă conturi bancare sau un CNP), ceea ce înseamnă că avem nevoie de o precizie foarte mare la nivelul întregii propoziţii, criteriul de evaluare folosit este **rata de eroare la nivel de propoziţie – SER** (Sentence Error Rate). Aceasta se calculează ca raport între numărul de propoziţii eronate şi numărul total de cuvinte din transcrierea de referinţă, astfel:

$$\text{SER}[\%] = \frac{\text{Număr propoziţii eronate}}{\text{Număr propoziţii în transcrierea de referinţă}} \times 100 \qquad (4)$$

Acest criteriu este unul puţin cam drastic, deoarece în majoritatea cazurilor utilizatorul final poate înţelege mesajul transmis, chiar dacă transcrierea lui conţine 10% cuvinte greşite. Din acest motiv, se preferă utilizarea unui alt criteriu de performanţă pentru eveluarea sistemelor de recunoaştere a vorbirii, şi anume **rata de eroare la nivel de cuvânt – WER** (Word Error Rate).

Rata de eraoare la nivel de cuvânt se calculează în funcţie de erorile de recunoaştere la nivel de cuvînt, după ce fraza ipotetică şi fraza de referinţă au fost aliniate cu ajutorul unui algoritm.

Erorile ce pot apărea la nivel de cuvânt sunt: cuvinte inserate, cuvinte substituite şi cuvinte şterse. Prin urmare, rata de eroare la nivel de cuvânt se calculează pe baza următoarei formule:

$$\text{WER}[\%] = \frac{\text{Număr erori de inserţie+Număr erori de substituţie+Număr erori de ştergere}}{\text{Număr de cuvinte în transcrierea de referinţă}} \times 100 \qquad (5)$$

Acest criteriu este mult mai adecvat pentru măsurarea performanţelor unui sistem de recunoaştere a vorbirii, deşi acesta nu ţine cont de faptul că o eroare de substituţie spre exemplu, poate fi reparată mult mai uşor dacă procentul de caractere eronate din cuvântul ipotetic este mic, decât dacă acest procent ar fi mare. Din acest motiv, în unele situaţii se utilizează **rata de eroare la nivel de caracter – ChER** (Character Error Rate) ce are ca unică diferenţă faţă de WER faptul că unitatea de bază utilizată în comparaţie este caracterul, nu cuvântul.

$$\text{ChER}[\%] = \frac{\text{Nr. erori inşerţie de caract. + Nr. erori substituţie de caract. + Nr. erori ştergere de caract.}}{\text{Nr. caractere în transcierea de referinţă}} \times 100 \qquad (6)$$

Cel mai utilizat criteriu de măsurare a performanţelor unui sistem de recunoaştere de vorbire rămâne totuşi rata de eroare la nivel de cuvânt (WER – ul). [3]

În figura următoare este ilustrat procedeul prin care se face o evaluare completă a sistemului de recunoaştere.

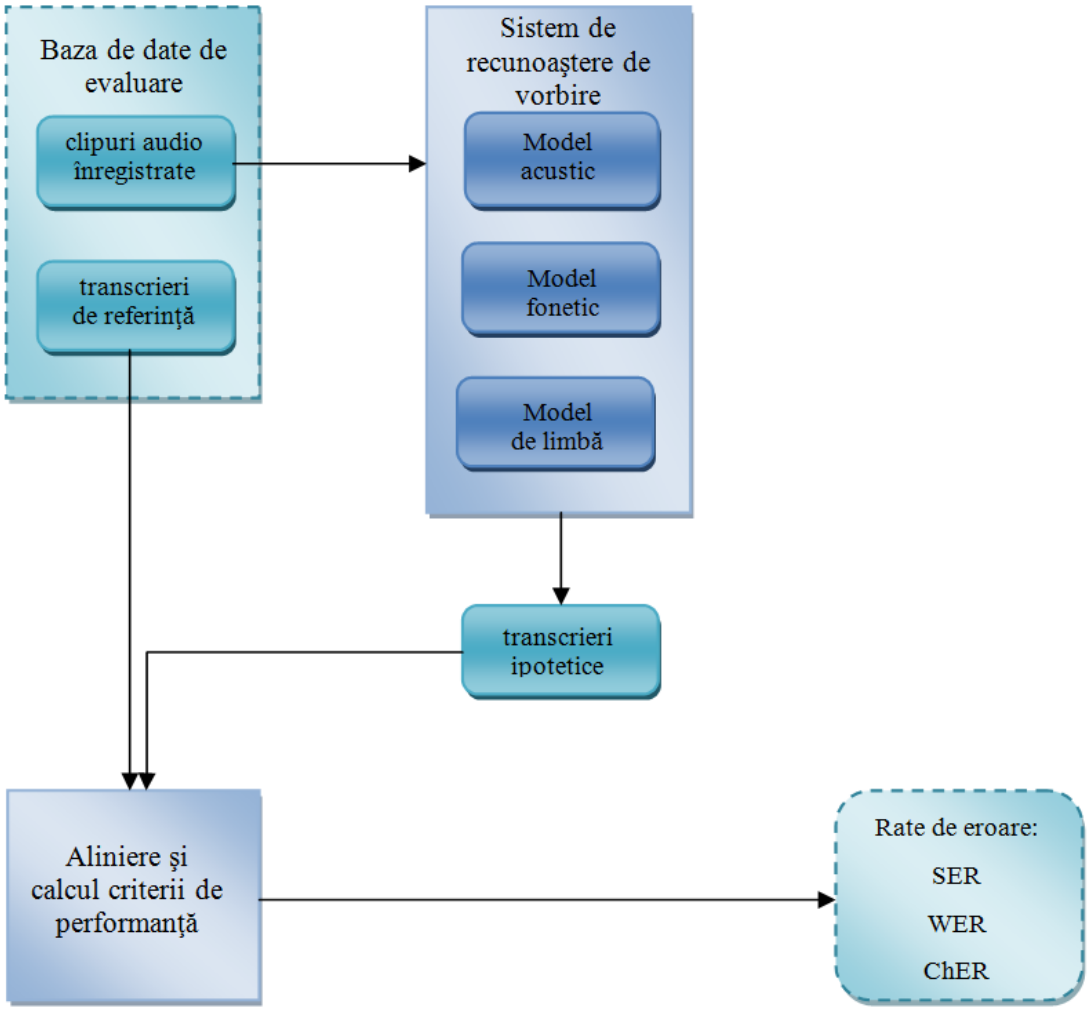


Figura 11. Evaluarea completă a sistemului de recunoaştere

CAPITOLUL 4

CONSTRUCȚIA UNUI SISTEM SIMPLU DE RECUNOAȘTERE AUTOMATĂ A VORBIRII

4.1 INTRODUCERE

Așa cum am menționat și în capitolul 3 variabilitatea inter-vorbitori reprezintă o problemă importantă în sarcina de recunoaștere de vorbire. Aceasta ar putea fi rezolvată prin crearea unui sistem dependent de vorbitor, la care atât antrenarea cât și evaluarea sistemului se fac pe aceeași voce. Dezavantajul acestui tip de sistem de recunoaștere este acela că, pentru fiecare vorbitor nou trebuie antrenat un nou model acustic. Acest lucru face ca sistemele de recunoaștere a vorbirii dependente de vorbitor să fie mai complexe și mai problematice din punctul de vedere al datelor de antrenare necesare. Din acest punct de vedere un sistem independent de vorbitor este mult mai simplu și mai flexibil, însă nu are performanțe la fel de bune.

4.2 ETAPE DE CONSTRUIRE ȘI OPTIMIZARE A SISTEMULUI

Pentru realizarea acestui proiect (”Utilizarea unei aplicații de recunoaștere vocală pentru controlul unui robot mobil”) am început prin a construi un sistem simplu de recunoaștere automată a vorbirii, și anume un sistem ce are ca sarcină **recunoașterea unei secvențe audio ce conțin cifre**. Ceea ce face ca acest sistem să fie mai simplu este faptul că are un vocabular redus

alcătuit doar din cele zece cifre ale sistemului zecimal: zero, unu, ..., nouă.

În prima fază am creat sistemul de recunoaștere dependent de vorbitor (dependent de vocea mea). Pentru realizarea acestui lucru am parcurs următoarele etape:

1) Înregistrarea unei baze de date de clipuri audio

Așa cum am văzut în figura 8 din capitolul 3, pentru a construi modelul acustic este nevoie de o bază de date de clipuri audio. În acest scop, am înregistrat 100 de clipuri audio a câte 12 cifre fiecare (exemplu: 7130 9562 1788).

Pentru realizarea înregistrărilor am conectat un microfon la calculator și am intrat pe serverul de înregistrări al laboratorului SpeeD (Speech and Dialogue), unde am creat un nou vorbitor, după care am început înregistrările, încercând să pronunț cât mai natural frazele afișate în mod aleator de către aplicația de înregistrări de pe server.

2) Crearea dicționarului fonetic și a listei de foneme

În tabelul 7 din capitolul 3 am prezentat fonemele limbii române și exemple de pronunție a unor cuvinte. În proiectul meu, dicționarul fonetic conține cele zece cifre ale sistemului zecimal: zero, unu, doi, trei, patru, cinci, șase, șapte, opt și nouă.

Următorul pas în realizarea proiectului a fost logarea la serverul de dezvoltare și crearea unui director de lucru temporar în care am creat și editat fișierul ce constituie dicționarul fonetic.

Formatul dicționarelor fonetice utilizate de CMU Sphinx impun scrierea unui singur cuvânt (însoțit de transcrierea lui fonetică) pe fiecare linie a fișierului. Pentru creșterea performanțelor unui sistem de recunoaștere de acest tip, manualul CMU Sphinx recomandă utilizarea de foneme specifice fiecărui cuvânt, adică modelarea diferită a fiecărui fonem în funcție de poziția lui în cuvânt. Spre exemplu, pentru cifra unu, vom avea: u_unu1 n_unu u_unu2. Observăm că același fonem ”u” este scris în funcție de poziția în care apare în cuvânt și astfel va fi modelat diferit.

La finalul editării fișierul arată astfel:

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

șase s1_șase a_șase s_șase e_șase

șapte s1_șapte a_șapte p_șapte t_șapte e_șapte

opt o_opt p_opt t_opt

nouă n_nouă o1_nouă w_nouă a1_nouă

Am creat apoi încă un fișier ce conține toate fonemele ce vor fi modelate de modelul acustic, respectând formatul impus (câte un fonem pe fiecare linie). Am creat acest fișier automat, cu ajutorul câtorva instrucțiuni specifice atașate în anexă, după care am adăugat ”fonemul” SIL pentru a modela zonele de liniște și am sortat fonemele în ordine alfabetică, tot cu ajutorul instrucțiunilor corespunzătoare.

3) Antrenarea modelului acustic

Primul pas al acestei etape a fost crearea unui proiect CMU Sphinx. Am creat un nou director numit ”rodigits” pe care l-am configurat corespunzător pentru a putea fi utilizat ca proiect CMU Sphinx, după care am copiat în acest director cele 100 de clipuri audio pe care le-am înregistrat. Apoi, într-un alt director numit ”etc” am copiat transcierea textuală a cuvintelor pronunțate în clipurile audio, precum și dicționarul fonetic, lista de foneme, dicționarul cu elemente acustice care nu sunt foneme și scripturile ”createFileIds.sh” și ”createTranscriptions.sh” cu ajutorul cărora se creează lista tuturor clipurilor audio înregistrate și, respectiv transcrierile lor textuale, în formatul necesar pentru procesul de antrenare.

Următorul pas a fost crearea listei cu clipuri audio ce vor fi folosite pentru antrenare, selectând primele 50 și ultimele 30 de clipuri audio. Cu celelalte 20 de clipuri audio am creat o listă ce va fi folosită pentru evaluare. În același mod am creat și listele cu transcrierile textuale corespunzătoare.

4) Configurarea procesului de antrenare și crearea fișierelor cu coeficienți MFCC corespunzătoare clipurilor audio de antrenare

În cadrul acestei etape am înlocuit în fișierul de configurare, valorile implicite ale parametrilor acustici specificând locația resurselor utilizate în procesul de antrenare. Apoi, în urma execuției unei comenzi simple, atașată de asemenea în anexă, am creat cele 80 de fișiere cu coeficienși cepstrali corespunzătoare celor 80 de clipuri audio de antrenare.

5) Antrenarea modelului acustic

Executând comanda : sphinxtrain_biosinf run am pornit procesul de antrenare.

6) Crearea modelului de limbă

În această etapă am implementat o gmatică cu stări finite (FSG – Finite State Grammar) utilizând formatul Java Speech Grammar (JSGF).

7) Evaluarea Sistemului

Primul pas a fost configurarea procesului de decodare, care este similară cu cea a procesului de antrenare, doar că de această dată, parametrii modificați au fost cei care specificau locația resurselor necesare în procesul de decodare și denumirea aplicației cu care se va face alinierea textului rezultat în urma recunoașterii cu textul de referință.

Apoi, la fel ca și la antrenare, am creat fișierele cu coeficienți MFCC corespunzătoare de această dată clipurilor de eveluare, am pornit procesul de decodare, iar la finalul procesului de decodare am vizualizat rezultatele evaluării, adică WER – ul și SER – ul despre care am discutat în capitolul 3. Mesajul obținut pe ecran este următorul:

MODULE: DECODE Decoding using models previously trained (decode script:

psdecode_fsg.pl)

Decoding 20 segments starting at 0 (part 1 of 1)

0%

Aligning results to find error rate

SENTENCE ERROR: 15.0% (3/20) WORD ERROR RATE: 1.7% (4/240)

Observăm că, la nivel de cuvânt, am obţinut un WER = 1.7%, mai exact, din cele 240 de cuvinte evaluate sistemul a recunoscut corect 236 de cuvinte şi a recunoscut greşit doar 4 cuvinte, ceea ce înseamnă că sistemul are performanţe destul de bune.

8) Optimizarea modelului acustic

Modelul acustic creat de CMU Sphinx este construit în mod implicit cu unități de vorbire dependente de context de tip trifonem (fonem căruia i se precizează vecinii stânga-dreapta). Tot în mod implicit, aceste trifoneme sunt implementate ca modele Markov ascunse cu trei stări emisive, fiecare având o distribuție de probabilitate de ieșire implementată cu un GMM (Gaussian Mixture Model). Fiecare astfel de stare-model se numește **senone** și ea poate fi comună mai multor trifoneme. În mod implicit, numărul de densități Gaussiene de probabilitate este configurat la 8, iar numărul de senone este configurat la 200.

Pentru optimizarea modelului acustic, am realizat o serie de teste ce au constat în decodarea acelorași fișiere, însă folosind modele fonetice dependente de context cu diferite densități Gaussiene per senone. În tabelul următor voi prezenta rezultatele obținute:

Tabelul 8. Evaluarea sistemului pentru număr variabil de densități Gaussiene și 200 de senone

Numărul de densități Gaussiene	WER[%]	SER[%]
1	15.4	70.0
2	15.4	75.0
4	17.1	85.0
8	1.7	15.0

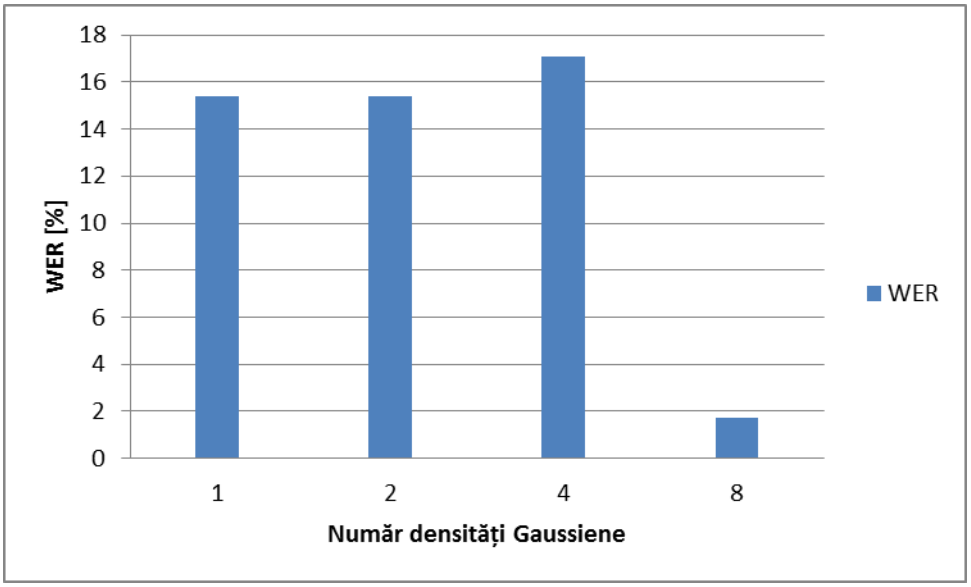


Figura 12. Reprezentarea grafică a WER-ului pentru număr variabil de densități Gaussiene și 200 senone

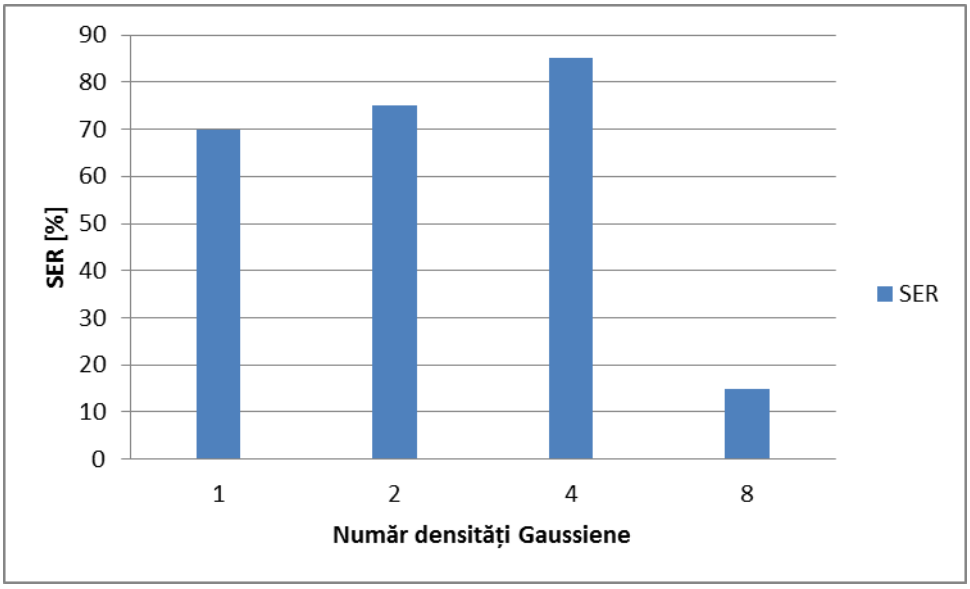


Figura 13. Reprezentarea grafică a SER-ului pentru număr variabil de densități Gaussiene și 200 senone

Observăm că cel mai bun WER s-a obținut 8 densități Gaussiene, iar cel mai bun SER s-a obținut de asemenea tot pentru 8 densități Gaussiene (WER = 1.7% și SER = 15%), valoari ce au fost, așa cum am mai spus, setate implicit.

În continuare, tot cu scopul optimizării sistemului, am realizat aceleași teste, dar modificând de această dată numărul de senone, de la 200 la 100. Rezultatele obținute sunt prezentate mai jos:

Tabelul 9. Evaluarea sistemului pentru 100 de senone și număr variabil de densități Gaussiene

Numărul de densități Gaussiene	WER[%]	SER[%]
1	1.7	15
2	2.1	10
4	2.1	10
8	2.1	10

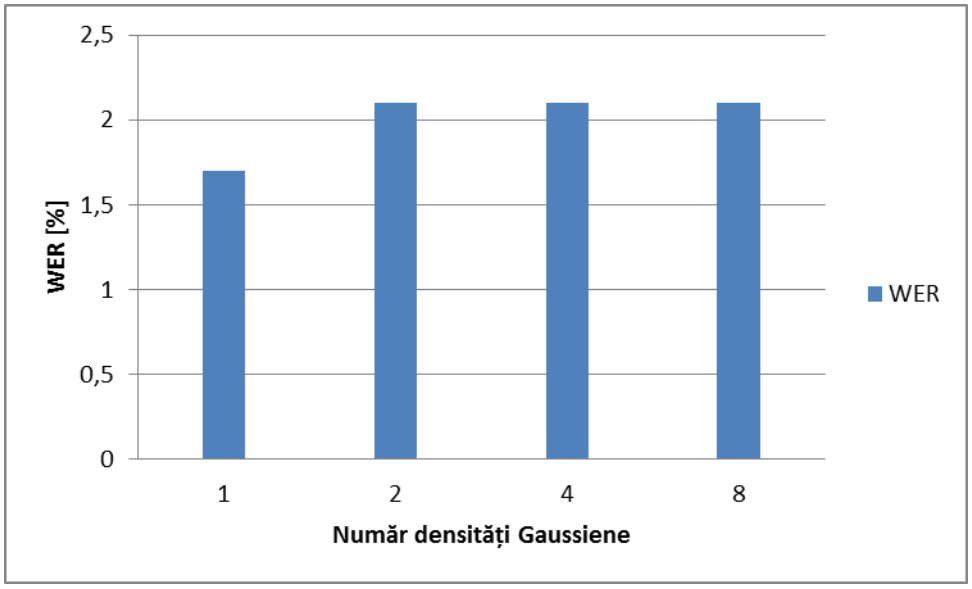


Figura 14. Reprezentarea grafică a WER-ului pentru 100 de senone și număr variabil de densități Gaussiene

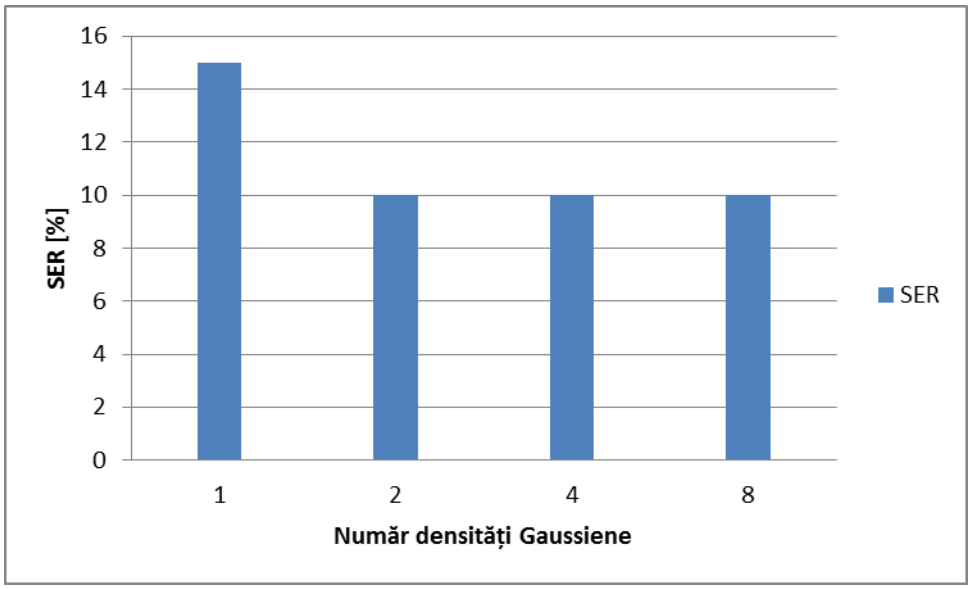


Figura 15. Reprezentarea grafică a SER-ului pentru 100 de senone și număr variabil de densități Gaussiene

Din ultimele două grafice, observăm că cel mai bun WER s-a obținut, la 100 de senone, pentru 1 densitate Gaussiană , iar cel mai bun SER pentru 2, 4 și 8 densități Gaussiene.

Recapitulând, cel mai bun WER rezultat pentru 100 de senone s-a obținut pentru 1 densitate Gaussiană și este egal cu cel mai bun WER rezultat pentru 200 de senone (1.7%) la 8 densități Gaussiene, ceea ce înseamnă că, strict din punctul de vedere al ratei de eroare la nivel de cuvânt, sistemul optim poate fi obținut în două moduri, prin următoarele configurări:

- 100 de senone și 1 densitate Gaussiană
- 200 de senone și 8 densități Gaussiene

Aceste sisteme pot fi utilizate cu success pentru recunoașterea unor comenzi simple, formate din 1-2 cuvinte.

Pe de altă parte, strict din punctul de vedere al ratei de eroare la nivel de propoziție, cele mai bune rezultate s-au obținut pentru 100 de senone și 2,4, respectiv 8 densități Gaussiene.

Deci, luând în considerare doar acest aspect, un sistem optim se poate obține din următoarele configurări:

- 100 de senone și 2 densități Gaussiene
- 100 de senone și 4 densități Gaussiene
- 100 de senone și 8 densități Gaussiene

Însă un sistem optim trebuie să ia în calcul ambele aspecte (WER și SER) și pentru că, așa cum observăm în următoarele două grafice, rezultatele sunt incomparabil mai bune pentru 100 de senone putem stabili că sistemul optimizat va trebui să aibă configurat un număr de 100 de senone.

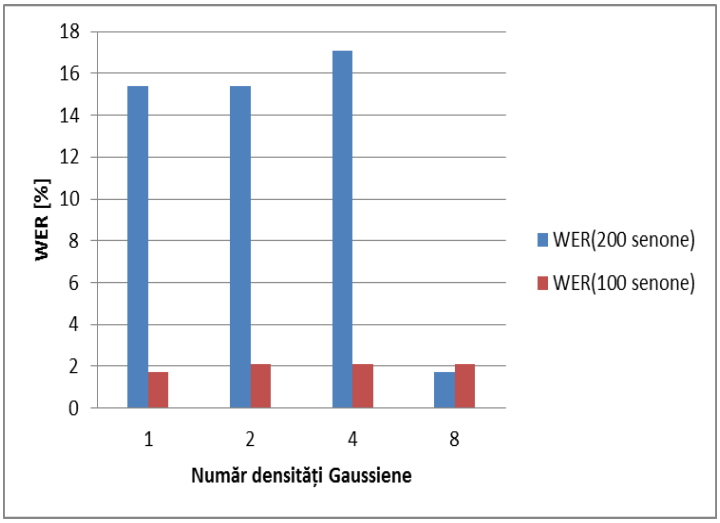


Figura 16. Comparație WER pentru 100 și 200 de senone

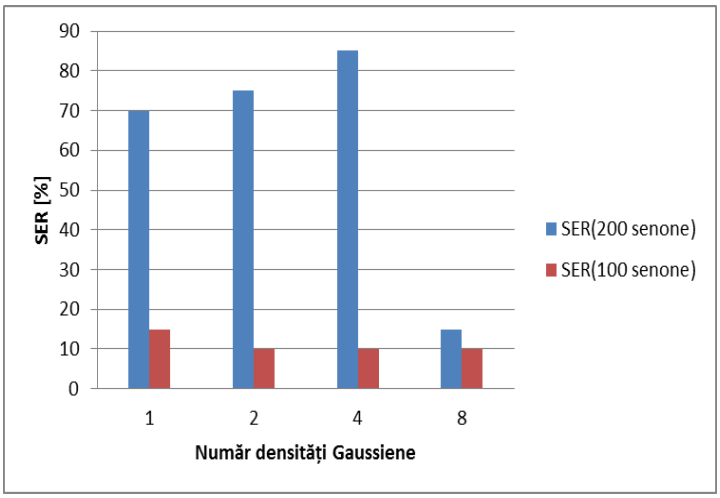


Figura 17. Comparație SER pentru 100 și 200 de senone

În ceea ce privește numărul de densități Gaussiene, cel mai bun WER este la 1 densitate Gaussiană, iar cel mai bun SER, la 2,4 și 8 densități Gaussiene, ceea ce determină necesitatea unui compromis. Acest compromis se va face în funcție de sarcina de recunoaștere, astfel că dacă sistemul trebuie să recunoască secvențe simple de 1-2 cuvinte, se va stabili drept optim sistemul ce lucrează cu 1 densitate Gaussiană și 100 de senone.

În proiectul meu, sarcina sistemului este de a recunoaște secvențe de câte 12 cuvinte, motiv pentru care sistemul pe care l-am considerat optim va fi cel ce lucrează cu 8 densități Gaussiene și 100 de senone.

CAPITOLUL 5

CONTROLUL ROBOTULUI JAGUAR 4X4

5.1 INTERFAȚA GRAFICĂ

Robotul a venit cu un program de control ce rulează ca aplicație Windows și permite controlarea și navigarea robotului. După ce programul este pornit, pe ecranul calculatorului apare fereastra de logare din figura următoare:

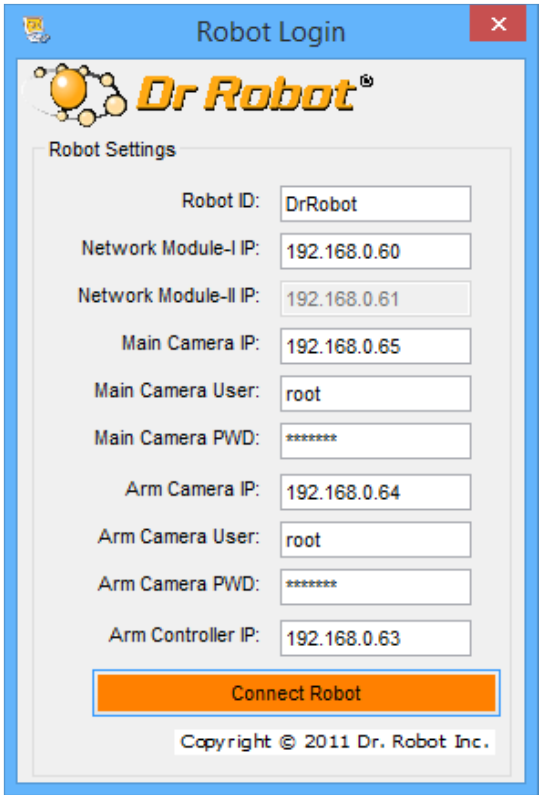


Figura 18. Fereastra de logare [1]

Aplicația de control se va conecta la platforma robotică folosind o conexiune Wi-Fi. Robotul este echipat cu un router Wi-Fi care asignează fiecărei componentă principală conectată la acesta un anumit IP. IP-urile fiecărui modul vor fi folosite pentru a trimite sau primi date.

Primul pas, pentru conectarea la router-ul robotului a fost schimbarea adresei IP a modulului de rețea Wireless al calculatorului gazdă. IP-ul acestuia va trebui să fie static și să se afle în gama de IP-uri corectă. IP-ul modulului de rețea prezent pe robotul cu care am lucrat este 192.168.0.60 și restul componentelor primesc IP-uri până la 192.168.0.65. Astfel vom seta IP-ul modulului de rețea Wireless la 192.168.0.66.

Pornirea robotului va determina pornirea router-ului intern și astfel va fi vizibil în lista de conexiuni posibile Wi-Fi (fiecare din roboți are asignat un ID unic, robotul folosit în acest proiect are asignat ID-ul: DriJaguar) așa cum arată și figura de mai jos.

Figura 19. Conectarea la robotul Jaguar prin Wi-Fi

Clasa folosită pentru realizarea inițializării câmpurilor de IP-uri ale robotului este clasa DrRobotRobotConnection.cs. În această clasă sunt folosite metode de citire și scriere în fișiere de tip XML. Pentru setarea acestor IP-uri se citește un fișier de configurare numit OutDoorRobotConfig_4x4.xml.

După conectarea la robot, aplicația de control pornește automat aplicația *Google Earth*, iar pe ecran vor apărea următoarele ferestre:

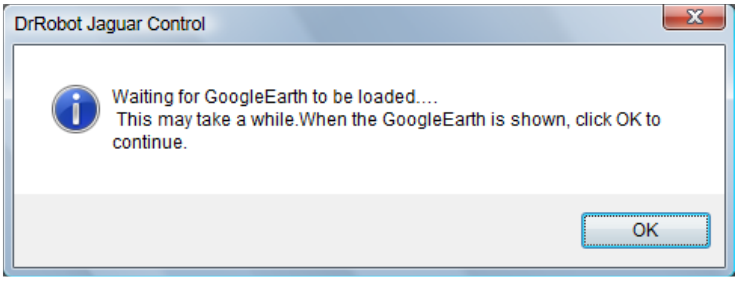


Figura 20. Mesaj încărcare Google Earth [1]

Așa cum putem deduce și din figura 20, procesul de încărcare a aplicației Google Earth durează destul de mult. Mai mult decât atât, aceasta nu este relevantă pentru atingerea obiectivelor acestui proiect, motiv pentru care am decis să o înlătur din aplicație și implicit, din interfața grafică, ce arăta astfel la început:

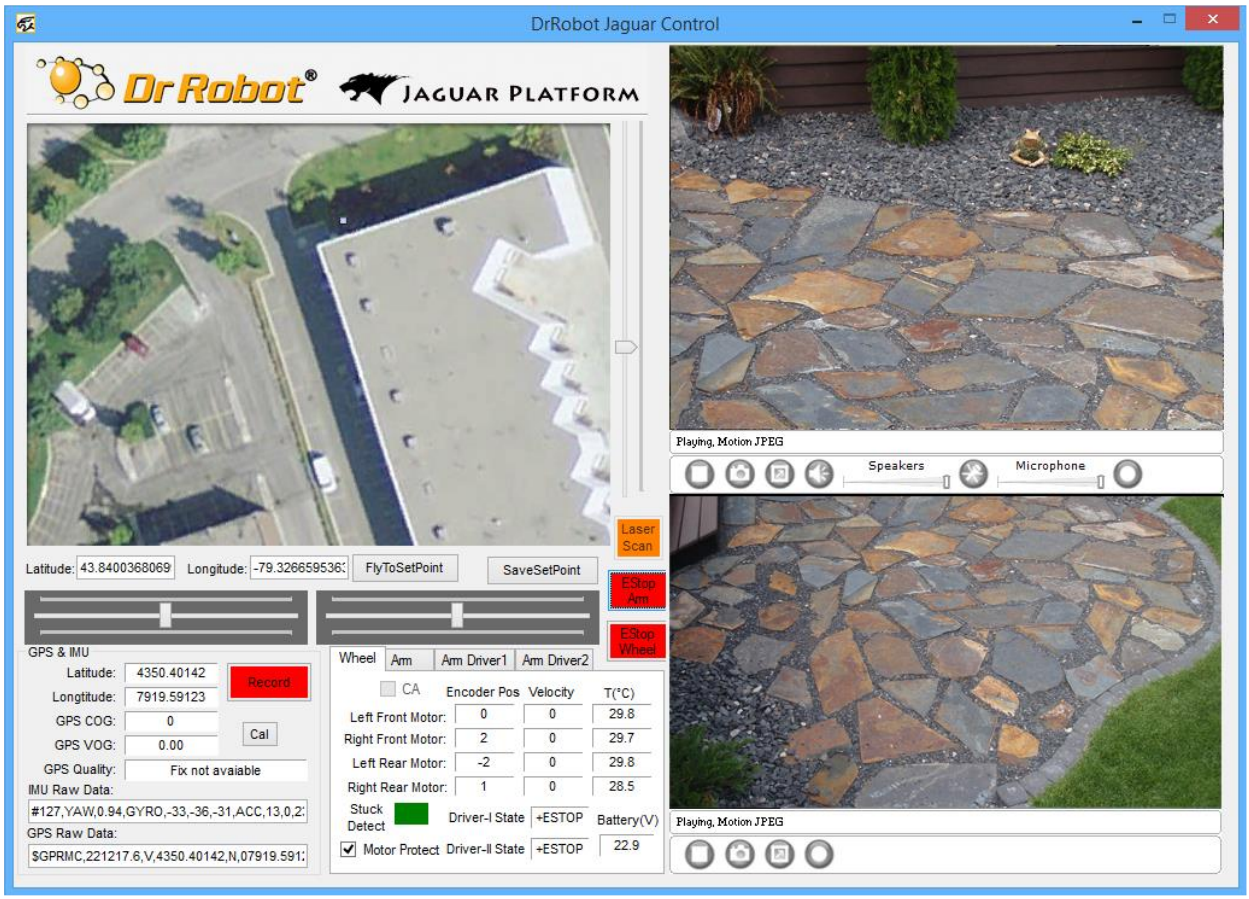


Figura 21. Interfața grafică [1]

În jumătatea dreaptă a interfeței se observă stream-urile video preluate de cele două camera ilustrate în figura 1. De asemenea, observăm existența celor două track bar-uri ce permit acționarea manuală a motoarelor robotului (stânga, dreapta, față, spate), direct din aplicație, precum și prezența casetelor de text în care sunt afișate datele primite de la senzori și a celor două butoane de oprire de urgență pentru motoare, respectiv pentru braț.

5.2 CONTROLUL MOTOARELOR

După utilizarea și înțelegerea codului aplicației demonstrative, folosind limbajul de programare C# și mediul de dezvoltare Microsoft Visual Studio 2010 am încercat construirea propriei aplicații de control. Folosind informații despre comenzile necesare pentru acționarea motoarelor (prezentate în capitolul 2.2) putem crea în interfața grafică butoane de control. În aplicația originală, motoarele sunt controlate folosind track-baruri. Aceste trackbar-uri trimit valorile într-o funcție care asigură faptul că aceste valori nu sunt eronate (prea mari sau prea mici) pentru a nu afecta în mod negativ funcționarea motoarelor. Am păstrat aceste funcții din motive de siguranță.

Astfel, avem două posibilități de a controla motoarele, fie trimițând comenzi prin folosirea clasei de comunicații DrRobotMainComm și funcția SendCommand(string) care trimite comenzi direct către controler-ul motoarelor, fie modificând valoarea trackbar-urilor trimițând astfel aceeași comadă indirect, folosind de asemenea și funcția de verificare.

```
forward = trackBarForwardPower.Value;
turn = trackBarTurnPower.Value;
```

Comanda trimisă de trackbar-uri arată în felul următor:

```
string strCmd = "MMW " + "!M " + cmd1.ToString() + " "
+(cmd2).ToString();
drrobotMainComm.SendCommand(strCmd);
```

Am realizat de asemenea și un buton de oprire , comanda aferentă acestui buton fiind:

```
string strCmd = "MMW !M 0 0";
drrobotMainComm.SendCommand(strCmd);
```

Se trimite ambelor controlere de motare valoarea 0.

Utilizând aceste informații am realizat o interfață grafică simplă care va avea ca scop doar controloarea motoarelor folosind butoane. În această variantă a aplicației nu am utilizat informațiile de la senzorul de distanță pentru a ne asigura că robotul va evita coliziunea cu obstacolele, astfel că aplicația este destul de nesigură, motiv pentru care testele au fost realizate folosind un suport pentru suspendarea robotului.

Figura 22. Interfața grafică simplă pentru control al motoarelor.

Codul pentru aceste butoane a fost scris folosind trackbar-urile, dar pentru butonul față am folosit și funcția de trimitere a comenzii de oprire, schimbând numele butonului (prin apăsarea lui) din ”față” în ”stop”.

```
private void button1_Click(object sender, EventArgs e)
        {
            trackBarTurnPower.Value = -350;
        }
```

```
private void button2_Click(object sender, EventArgs e)
{
    trackBarTurnPower.Value = 350;
}

private void button3_Click(object sender, EventArgs e)
{

    if (button3.Text == "fata")
    {
        trackBarForwardPower.Value = 90;
        button3.Text = "stop";
    }
    else if (button3.Text == "stop")
    {
        string strCmd = "MMW !M 0 0";
       drrobotMainComm.SendCommand(strCmd);
        button3.Text = "fata";
    }

}

private void button4_Click(object sender, EventArgs e)
{
    trackBarForwardPower.Value = -90;
}
```

5.3 Controlul Robotului prin Comenzi Vocale în Limba Engleză

Schema de principiu a controlului platformei robotice Jaguar utilizând comenzi vocale în limba engleză este prezentată în figura următoare:

Figura 23. Schemă de principiu

Pe scurt, principiul de funcționare este următorul:

Semnalul vocal este preluat cu ajutorul unui microfon conectat la calculator. Pe calculator rulează o aplicație C# care preia semnalul de la microfon, începe procesul de recunoaștere și oferă transcrierea textuală a cuvântului sau secvenței de cuvinte rostite. Cuvintele rostite sunt asociate unor comenzi ce sunt transmise prin Wi-Fi către microprocesorul embedded al robotului. Microprocesorul decodează aceste comenzi și în funcție de acestea, acționează

motoarele sau ”citește” informațiile de la senzori, pe care le trimite înapoi în aplicația C# de pe calculator, tot prin intermediul modulelor de comunicații Wi-Fi.

5.3.1 Namespace-ul System.Speech.Recognition

În ideea că aplicația va rula în modul offline, laptopul fiind deja conectat wireless la robot, neavând astfel acces la server-ul care folosește CMU Sphinx pentru transcrierea textuală a comenzilor vocale, am decis folosirea namespace-ului System.Speech.Recognition. Integrarea acestui namespace nu a fost dificlă datorită faptului ca în crearea aplicației mele utilizez framework-ul .NET și mediul Microsoft Visual Studio.

Astfel, am creat o aplicație de recunoaștere a vorbirii. Singurul deficient în utilizarea funcțiilor puse la dispoziție de acest namespace este acela că putem crea comenzi doar pentru limba engleză, întrucât gramatica și dicționarul au fost create pentru limba engleză.

Aplicația de recunoaștere a vorbirii realizează următoarele operații de bază:

- Inițializează un sistem de recunoaștere de vorbire.
- Crează gramatica pentru recunoașterea vorbirii.
- Încarcă dicționarul în sistemul de recunoaștere.
- Crează un *handler* pentru evenimentul de recunoaștere a vorbirii.

În acest proiect am folosit clasa SpeechRecognizer, această clasă controlând recunoașterea de vorbire în Windows.

Pentru a inițializa sistemul de recunoaștere am folosit:

```
SpeechRecognizer sr = new SpeechRecognizer();
```

Se va crea astfel o instanță nouă pentru SpeechRecognizer.

Pentru crearea gramaticii folosite în recunoașterea de vorbire este necesară utilizarea constructorilor și a metodelor din clasele GrammarBuilder și Choices. Cuvintele sunt adăugate folosind un obiect Choices. Pentru a se realiza recunoașterea, utilizatorul trebuie să rostească exact unul din elementele adăugate de instanța Choices. Cuvintele se vor adăuga ca șir de caractere și vor fi argumente pentru metoda Add([ ]).

După crearea instanței Choices și setarea acesteia, se va crea o instanță pentru GrammarBuilder. Folosind metoda Append(Choices), se vor adăuga obiectele create în instanța GrammarBuilder.

```
Choices lista = new Choices();
            lista.Add(new string[] { "start laser" });
            lista.Add(new string[] {"forward"});
            lista.Add(new string[] { "turn around" });
            lista.Add(new string[] { "right" });
            lista.Add(new string[] { "start" });
            lista.Add(new string[] { "left" });
            lista.Add(new string[] { "stop" });
            lista.Add(new string[] { "lights on" });
            lista.Add(new string[] { "turn lights off" });
            lista.Add(new string[] { "blower on" });
            lista.Add(new string[] { "blower off" });
```

```
            Grammar dictionar = new Grammar(new
GrammarBuilder(lista));
```

Apoi se va crea o instanță Grammar și se va inițializa cu instanța GrammarBuilder.

Vom încărca gramatica în sistemul de recunoaștere:

```
speech_rec.LoadGrammar(dictionar);
```

Sistemul de recunoaștere va declanșa mai multe evenimente în timpul execuției, inclusiv evenimentul SpeechRecognized. Acest eveniment este declanșat când a fost recunoscut un cuvânt din gramatică.

```
speech_rec.SpeechRecognized += recunoscut;
```

În cazul de față, dacă se va declanșa evenimentul SpeechRecognized, vom executa evenimentul *recunoscut.*

Pentru a transcrie în text comanda vocală vom folosi argumentul evenimentului declanșat:

```
private void recunoscut(object sender, SpeechRecognizedEventArgs
e)
        {
            int i=0;


    if (e.Result.Text == "lights on")
            {
                powerIO = powerIO | 0x80;
                string strCmd = "SYS MMC " + powerIO.ToString();
                drrobotMainComm.SendCommand(strCmd);
            }
```

În codul prezentat mai sus, după declanșarea evenimentului recunoscut – adică după ce a fost recunoscută una din comenzile inițializate în gramatică se va executa o anumită acțiune. Dacă comanda recunoscută a fost *lights on* , atunci se va trimite o comandă către controler-ul robotului pentru a aprinde luminile.

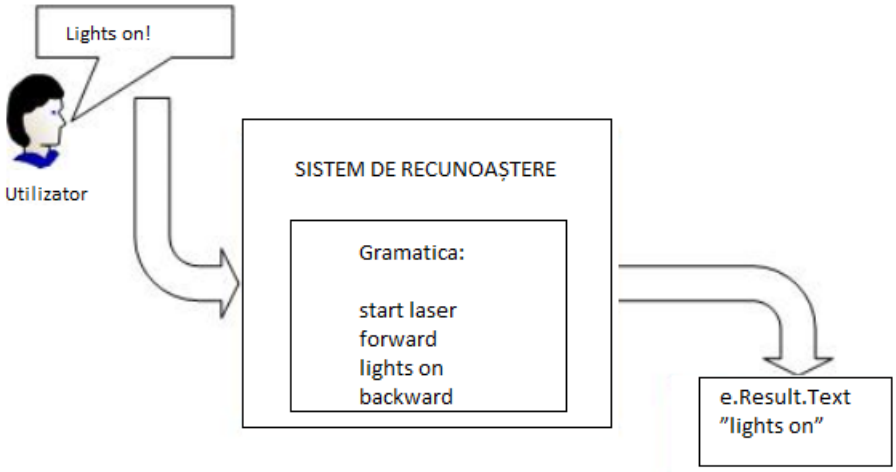


Figura 24. Mod de funcționare al sistemului de recunoaștere

Aceste comenzi au fost implementate pentru a comanda robotul Jaguar folosind aplicația C#. În aplicație au fost păstrate butoanele create anterior pentru a exista de asemenea și posibilitatea de control manual.

În continuare, am mărit siguranța execuției aplicațiilor folosind informațiile de la senzorul laser.

5.4 Utilizarea Senzorului Laser

Robotul Jaguar 4x4 este dotat cu un senzor de distanță laser foarte performant, Hokuyo Laser Scanner UTM-30LX, caracteristicile acestui senzor ajutându-ne foarte mult în preluarea de informații utile din mediul exterior. Senzorul acționează pe o rază de 30 de metri și are un unghi de detecție 270^0.

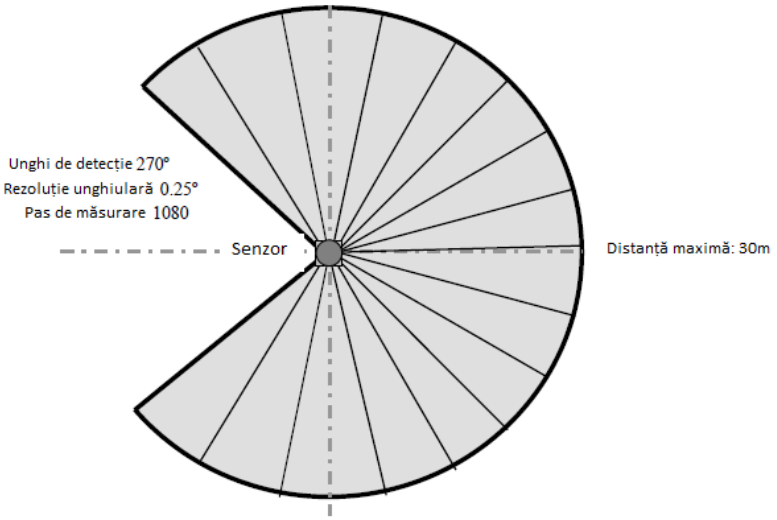


Figura 25. Aria de acoperire a senzorului laser

După cum se poate observa în imaginea de mai sus, aria de acoperire este foarte mare, făcând astfel realizarea aplicațiilor cu roboți autonomi o sarcină mult mai ușoară în ceea ce privește detecția obstacolelor. Senzorul are un pas de măsurare de 1080, acest lucru înseamnă că avem 1080 de raze laser trimise pe direcții diferite, distanța se va măsura calculând durata de timp în care raza laser se întoarce înapoi la senzor și faza acestei raze –reflectată de un obstacol.

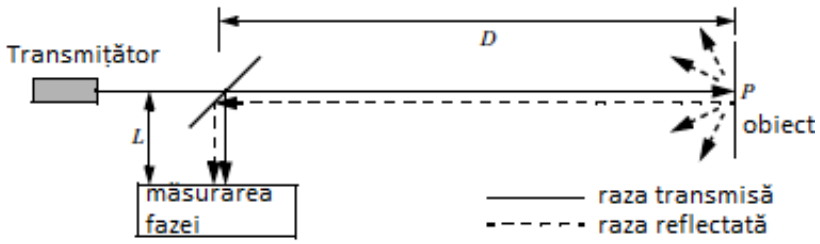


Figura 26. Modul de măsurare al distanței și fazei

Folosind aceste date, informațiile prezente în manualul de utilizare (protocol de comunicare), și informațiile prezentate în codul aplicației demonstrative, am putut înțelege

suficient de bine modul de utilizare pentru a integra informațiile de la acest senzor în aplicația de control.

Conectarea la senzor se face folosind o aplicație de tip client –server. Vom deschide un socket de comunicație de tip TCP, vom transmite un stream de date (comenzi) și vom primi un alt stream de date (măsurători) din care vom extrage informațiile utile.

```
clientSocketLaser = new TcpClient();
cmdStreamLaser = clientSocketLaser.GetStream();
readerLaser = new BinaryReader(cmdStreamLaser);
writerLaser = new BinaryWriter(cmdStreamLaser);
```

Mesajul primit de la senzor va fi convertit în codare de tip ASCII și apoi va fi interpretat, datele fiind puse în vectorul dataArray[ ]:

```
while (lineCnt < DATALINE - 1)
                {
                    temp = dataRead.ReadLine();
temp = temp.Remove(64, 1);
                    tempData =
System.Text.Encoding.ASCII.GetBytes(temp);
                    Array.Copy(tempData, 0, dataArray, lineCnt *
64, tempData.Length);
                    lineCnt++;
                }
```

Se realizează apoi o conversie matematică pentru a transforma datele din acest vector în date ce vor reprezenta distanța în milimetri. Datele convertite se vor stoca în senzorul disData[ ]:

```
for (int i = 0; i < DISDATALEN; i++)
                {
                    disData[i] = ((long)(dataArray[3 * i] -
0x30) << 12) + (((long)(dataArray[3 * i + 1] - 0x30)) << 6) +
(long)(dataArray[3 * i + 2] - 0x30);
                    if (disData[i] < Math.Abs(LASER_OFFSET_X *
1000)) disData[i] = (long)(MAX_LASER_DIS * 1000);
                    laserSensorData.DisArrayData[i] =
((double)disData[i]) / 1000;
                }
```

Astfel, putem folosi informațiile din acest vector pentru a afla distanța până la un obiect pe o anumită direcție. După cum am menționat mai devreme, există un număr de 1080 de măsurători, astfel vectorul nostru va avea exact aceeași dimensiune. Citind spre exemplu valoarea din elementul 0 al vectorului disData[ ] vom afla distanța până la cel mai apropiat obiect pe direcția corespunzătoare razei cu unghiul 0 (prima rază din partea dreaptă) , disData[540] va reprezenta raza din mijloc, etc.

În aplicația mea, voi scana partea din față pentru a afla minimul din acea parte a vectorului. Dacă acest minim este mai mic decât o valoare setată în program, înseamnă că robotul este aproape de a lovi un obstacol, motiv pentru care am luat decizia de oprire de urgență a motoarelor.

Funcția care va realiza această operație se numește check_laser( ), aceasta va fi apelată cu o frecvență de 12,5 Hz.

```
    private void check_laser(object sender, EventArgs e)
```

```
        {
            float fata = disData[512];
            float minim_fata = 50000;
            int i;
            for (i = 490; i < 530; i++)
            {
                if (disData[i] < minim_fata)
                {
                    minim_fata = disData[i];
                }
            }
            if (minim_fata < 1500)
            {
                safe = false;
                trackBarForwardPower.Value = 0;
                trackBarTurnPower.Value = 0;
            }
            else
            {
                safe = true;
            }
        }
```

Implementarea senzorului în aplicație a ridicat foarte mult gradul de siguranță în ceea ce privește controlul vocal, astfel chiar dacă o comandă nu va fi recunoscută (din motive de zgomot, etc), robotul va evita coliziunile cu obstacole în mod autonom. Din acest motiv, prima comandă care trebuie dată robotului este aceea de activare a laserului. Dacă laserul nu este activat, nu va fi disponibil controlul motoarelor.

```
if (e.Result.Text == "start laser")
            {
                sendCommandLaser("BM");
                laser_started = true;
                sendCommandLaser(SCANCOMMAND);
                textBox2.Text = "Laser started";
            }
```

Astfel, aplicația în această versiune, permite controlul manual al robotului, control vocal folosind comenzi în limba engleză și evitarea coliziunii cu obstacole folosind senzorul de distanță laser. Testarea aplicației în această formă a fost realizată în exterior, în condiții de zgomot ușor, și a fost demostrată o funcționalitate destul de bună, robotul reușind să răspundă la aproximativ 90% din toate comenzile care i-au fost date.

5.5 CONTROLUL ROBOTULUI PRIN COMENZI VOCALE ÎN LIMBA ROMÂNĂ

Schema de principiu a controlului platformei robotice Jaguar 4x4 prin comenzi vocale în limba română este ilustrată în figura următoare:

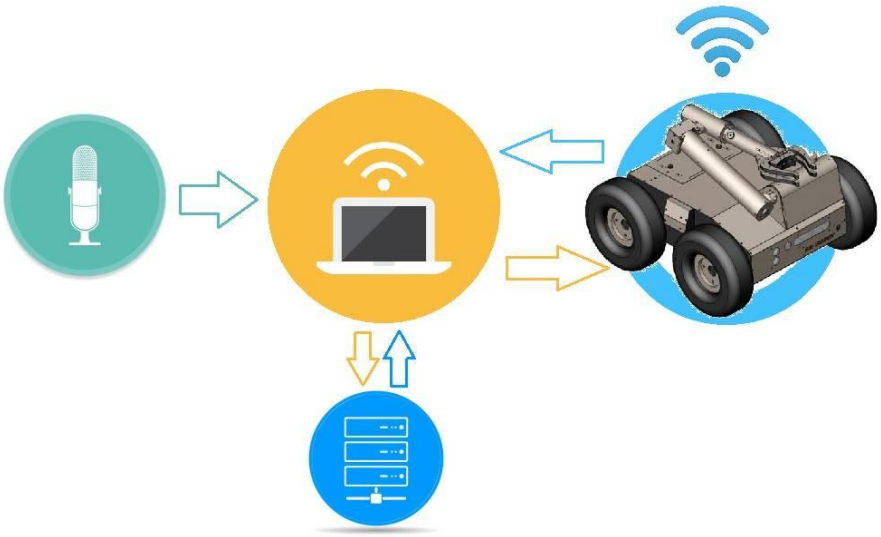


Figura 27. Schemă de principiu

Principiul este asemănător cu cel al controlului cu comenzi în limba engleză, prezentat în capitolul 5.3, cu excepția faptului că pentru sarcina de recunoaștere nu am mai folosit namespace-ul System.Speech.Recognition, ci sistemul de recunoaștere cu modelul creat de mine, utilizând toolkit-ul CMU Sphinx.

Pe scurt, principiul de funționare este următorul:

Achiziția semnalului audio se face utilizând librăria Naudio, după care fișierul wav rezultat este apoi convertit într-un șir de bytes care va fi trimis la server. În aplicația server, folosind toolkit-ul CMU Sphinx se va obține transcrierea textuală a cifrelor rostite. Aceasta va fi transmisă sub forma unui șir de bytes către calculatorul gazdă (aplicația de tip client) și se va interpreta textul primit care va fi asociat unei comenzi. Comanda rezultată va fi trimisă către microprocesorul embedded al robotului care va acționa motoarele sau va preluata datele de la senzori, în funcție de comanda trimisă.

5.5.1 Utilizarea toolkit-ului CMU Sphinx

În aplicația de recunoaștere automată a vorbirii pentru cifre, realizată de mine, am folosit toolkit-ul CMU Sphinx. Acesta a fost pus la dispoziție pe serverul SpeeD. Astfel, în scopul realizării unui sistem de comandă vocală în limba română pentru aplicația de control al robotului a fost necesară folosirea acestui toolkit.

Integrarea funcțiilor CMU Sphinx într-o aplicație de tip C# nu a fost deloc ușoară datorită faptului că este un toolkit în totalitate pentru Java. Singura soluție existentă astfel este crearea unei aplicații client-server, unde aplicația client va fi scrisă în totalitate folosind limbajul de programare C#, iar aplicația server va fi scrisă în totalitate în limbajul de programare Java.

Scenariul de funcționare (ilustrat în figura 28.):

- Se va prelua stream-ul audio de la microfonul calculatorului
- Se va converti semnalul audio într-un șir de bytes
- Se va deschide un socket TCP din aplicația client
- Se va trimite șirul de bytes către aplicația server
- Aplicația server va avea un socket TCP deschis și va avea, de asemenea integrate funcțiile CMU Sphinx
- Se va transmite șirul de bytes la sistemul de recunoaștere bazat pe modelele create de mine
- Se va transmite transcrierea textuală a voribirii înapoi către aplicația client
- În funcție de comenzile recunoscute se vor acționa motoarele

5.4.1 Aplicația Client (C#)

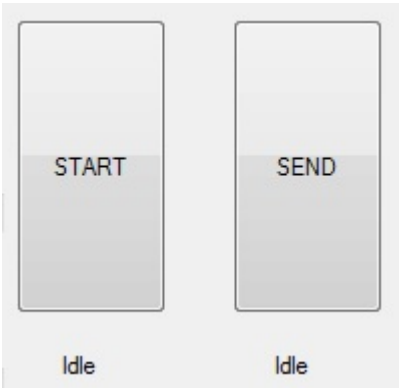
După cum am menționat mai devreme, aplicația client va trebui scrisă în totalitate în C# fiind de asemenea aceeași aplicație care controlează robotul. Aplicația client va trebui să realizeze captura semnalului audio să îl transforme în șir de bytes și să trimită acest șir către server. Pentru a captura semnalul audio de la microfon și pentru ca acesta să aibe codarea necesară aplicației server Java, am folosit o librăria NAudio. Această librările pune la dispoziție o multitudine de funcții pentru prelucrarea semnalelor audio. Am folosit funcțiile de captură a semnalului pentru o frecevență setată de noi. Toolkit-ul CMU Sphinx necesită fișiere .wav înregistrate cu o frecvență de eșantionare de 16kHz folosind un singur canal (mono). Toate aceste setări se pot face în C# folosind librăria audio menționată mai devreme.

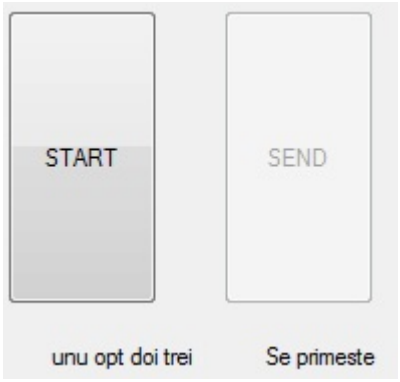
Pentru a nu bloca Thread-ul principal de execuție în timpul capturii semnalului audio a trebuit să deschid un al thread în care să se execute această operație. În varianta inițială a aplicației captura semnalului și trimiterea la server se face folosind butoane de control (pentru simplificare). Astfel, apăsarea butonului ”START” va crea o nouă instanță a thread-ului workerThread și va porni execuția funcțiilor de captură audio.

```
static working.Worker workerObject = new working.Worker();

private void button6_Click(object sender, EventArgs e)
        {
            button6.Enabled = false;
            button7.Enabled = true;
            started = true;
Thread workerThread = new Thread(workerObject.DoWork);
            textBox2.Text = "Recording";
            working._shouldStop = false;
            workerThread.Start();
        }
```

În timpul capturii butonul de ”START” va deveni indisponibil, va putea fi folosit însă butonul de ”SEND”, butonul SEND va deveni indisponibil după apăsare, dar butonul start va deveni din nou disponibil (după cum se poate observa în figura de mai jos). Acest buton va opri thread-ul nou creat și va face posibilă execuția funcției SendFile( ).





În funcția SendFile( ) vom trimite datele sub formă de bytes prin socketul TCP nou creat, către serverul Java.

```
try
            {
                Socket socket = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);


                IPAddress ip = IPAddress.Parse("192.168.0.78");
                IPEndPoint end = new IPEndPoint(ip, 5000);
                MesajCurent = "Conectare server";
                socket.Connect(end);
                MesajCurent = "Conectat server";
                byte[] wav =
File.ReadAllBytes(@"C:\ExempluAdela\Test0001.wav");
                socket.Send(wav);
                MesajCurent = "Date trimise";
                socket.Close();
          }
```

După ce trimitem datele, vom aștepta răspunsul:

```
byte[] rcvBytes = new byte[1024];
                socket.Receive(rcvBytes);
                MesajCurent = "S-a primit";
                String rcv =
System.Text.Encoding.ASCII.GetString(rcvBytes);
                MesajCurent = rcv;
```

Vom converti datele primite în codare ASCII și vom folosi apoi aceste date pentru a trimite comenzi robotului.

Spre exemplu vom putea atașa un timer unei etichete (label1)și vom actualiza textul afișat de această etichetă cu textul reținut în `MesajCurent`. Vom folosi apoi textul rezultat pentru a controla valoarea unui trackbar și implicit motoarele robotului.

```
private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Text = Receiver.MesajCurent;
            label2.Text = Sender.MesajCurent;
        }


if (label1.Text == "unu doi trei")
            {
                trackBarForwardPower.Value = 300;
            }
```

Se pot observa etichetele din imaginile de mai sus –inițial acestea sunt Idle, după transmiterea semnalului li recepția transcrierii textuale observăm cum se schimbă și textul etichetelor.

5.5.2 Aplicația Server (Java)

Pentru a folosi toolkit-ul CMU Sphinx aplicația server trebuie realizată în limbajul Java. Este dorită utilizarea acestui toolkit pentru a putea folosi modelul realizat de mine, antrenat pe baza de date personală de clipuri audio, rezultând astfel un sistem de recunoaștere dependent de vorbitor.

CMU Sphinx creează un set de fișiere care vor fi folosite în aplicația server:

- Dicționarul fonetic.
- Modelul acustic
- Gramatica

```
import java.net.*;

import edu.cmu.sphinx.api.Configuration;
import edu.cmu.sphinx.api.SpeechResult;
import edu.cmu.sphinx.api.StreamSpeechRecognizer;
```

Aplicația server va primi stream-ul de date de la aplicația client folosind un socket de tip ServerSocket:

```
private ServerSocket  serverSocket;
```

Se vor inițializa apoi variabilele folosite:

```
private Configuration configuration;
private StreamSpeechRecognizer recognizer;
private SpeechResult result;

private static String ACOUSTIC_MODEL_PATH="/models/acustic";
private static String DICTIONARY_PATH=
"/models/dictionar/rodigits.dic";
private static String GRAMMAR_PATH= "/models/gramatica";
private static String GRAMMAR_NAME="rodigits";
```

Se va porni serverul, se va ține într-o buclă infinită în care se vo primi datele de la aplicația client și se va trimite înapoi răspunsul text.

```
while(true)
```

```
{ System.out.println("server alive");
   try
   {
         Socket server = serverSocket.accept();
       Socket server2 = serverSocket2.accept();
   OutputStream os = server2.getOutputStream();
     // Socket server = serverSocket.accept();
    byte[] toSendBytes = new byte[1024];

      if(server.getInputStream() != null ){
      recognizer.startRecognition(server.getInputStream());
            while ((result = recognizer.getResult()) != null)
            {

                number = result.getHypothesis();
                //System.out.println("rezultat=: "+number);
                toSendBytes = number.getBytes();
                os.write(toSendBytes);
                System.out.println("rezultat=: "+number);
                os.flush();
                System.out.println("flushed");

            }
```

5.5 FUNCȚIONAREA APLICAȚIEI FINALE

Folosind toate aceste informații prezentate în subcapitolele anterioare am reușit realizarea unei aplicații de control pentru platforma robotică Jaguar 4x4, scrisă complet în C#. Aplicația pune la dispoziție posibilitatea de control manual, posibilitatea de control prin comenzi vocale în limba engleză (folosind System.Speech.Recognition) dar și posibilitatea de control folosind comenzi în limba română (folosind conexiunea la aplicația server scrisă în Java).

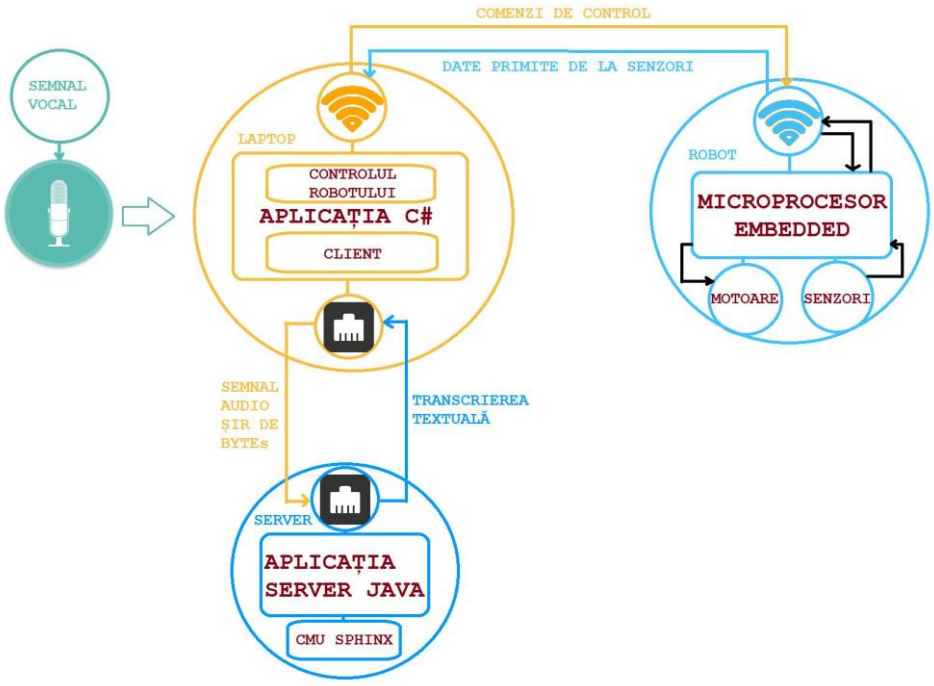


Figura 28. Arhitectura sistemului în versiunea finală

Pentru a scăpa de utilizarea butoanelor din interfața grafică, am folosit namespace-ul System.Speech.Recognition, pentru a crea un set de comenzi care realizează aceleași funcții ca și butoanele START și SEND prezentate mai devreme.

Sistemul funcționează folosind în permanență date de la senzorul Laser, pentru a oferi un grad mai mare siguranță. În interfața grafică sunt de asemenea afișate și datele de la senzorii motoarelor, pentru ca utilizatorul să se asigure de buna funcționare a robotului.

Transcrierea textuală a cifrelor rostite de utilizator este realizată pe serverul Java și transmisă înapoi către aplicație. Se afișează răspunsul text sub butoanele START și SEND. Interfața, în starea ei actuală este prezentată în imaginea de mai jos:

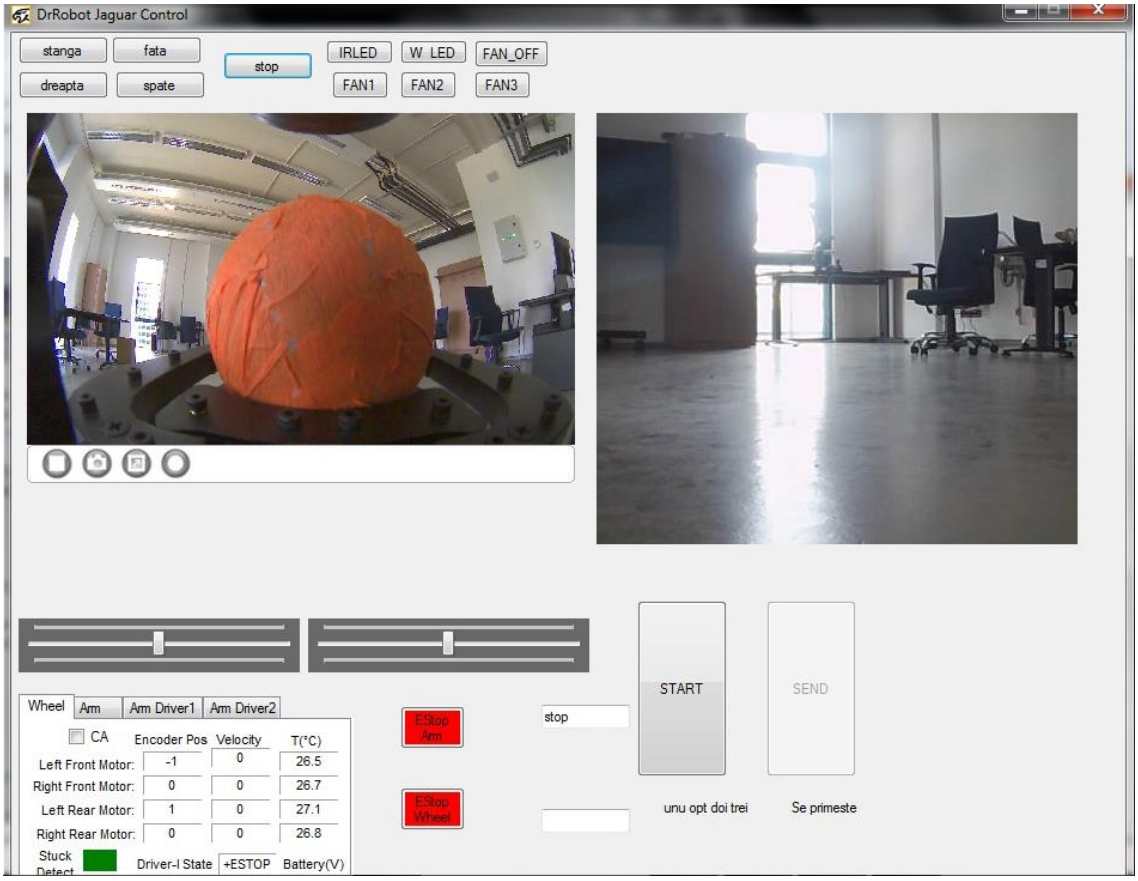


Figura 29. Interfața grafică a aplicației de control

În figura de mai jos observăm acțiunile realizate de aplicația server (afișaj în consolă):

Figura 30. Aplicația server

CAPITOLUL 6

CONCLUZII

6.1 CONCLUZII GENERALE

Lucrarea de față a avut ca scop controlarea unei platforme robotice mobile utilizând o aplicație de recunoaștere vocală. Obiectivul a fost atins în două moduri: cu comenzi în limba engleză, utilizând namespace-ul System.Speech.Recognition și cu comenzi în limba română, utizând toolkit-ul CMU Sphinx și sistemul propriu de recunoaștere automată a vorbirii dependent de vorbitor. Aplicația finală este funcțională, dar poate fi îmbunătățită.

Această lucrare îmbină cunoștințe din domenii precum: prelucrarea digitală a semnalelor, programare orientată pe obiecte (C#, Java), fonetica limbii române, sisteme embedded, aplicații client-server.

În concluzie, consider acest proiect o etapă importantă în dezvoltarea personală și un bun început pentru proiecte mai complexe.

6.2 CONTRIBUȚII PERSONALE

Contribuțiile personale aduse în această lucrare sunt prezentate în capitolele 4 și 5 și pot fi rezumate în:

a) Înreagistrarea unei baze de date de clipuri audio ce vor putea fi folosite și de către alți utilizatori care au acces la server, pentru realizarea altor proiecte. (Secțiunea 4.2).

b) Cunstrucția unui sistem de recunoaștere a sevențelor audio ce conțin cifre (Capitolul 4).

c) Cercetarea și dezvoltarea modului de funcționare a robotului Jaguar 4x4 (Capitolul 5).

d) Realizarea unei interfețe grafice pentru controlul robotului (Capitolul 5)

6.3 PROIECTE VIITOARE

- Sistem inteligent bazat pe recunoaștere vocală
- Implementarea unei aplicații de navigare autonomă
- Prelucrare de imagini utilizând cele două camere
- Utilizarea senzorului IMU al robotului pentru misiuni de explorare

REFERINȚE

[1] Dr Robot, 2014 "All – Terrain Autonomous Navigation Robot with GPS-IMU, *Jaguar 4x4 Wheel with Manipulator Arm, User Guide"*

[2] [Cucu, 2011] Cucu, H., "Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian," Teză de doctorat, Universitatea Politehnica din Bucureşti, România, 2011, http://speed.pub.ro/academic/research-and-development-project-in-spoken-language-technology.

[3] Proiect de cercetare-dezvoltare în Tehnologia Vorbirii – SpeeD

speed.pub.ro/speed3/wp-content/uploads/.../Indrumar-de-proiect-PCDTV-v11.pdf

[4] Documentație Jaguar-PMS5006-Protocol

[5] Advanced Digital Motor Controllers User Manual - RobotShop

www.robotshop.com/content/PDF/user-manual-vdc2450.pdf

[6] Richard Blum , Tcp sockets csharp pdf - P(2) - Docs-Engine.com /C# Network Programming.pdf

www.docs-engine.com/pdf/2/tcp-sockets-csharp.html

[7] Get Started with Speech Recognition Tutorial

https://msdn.microsoft.com/en-us/library/office/hh361683(v=office.14).aspx

[8] E190Q-Lab01-IntroToTheJaguar

https://www.hmc.edu/.../E190Q/E190Q-Lab01-IntroToTheJ

[9] System.Speech.Recognition Namespace

https://msdn.microsoft.com/en-us/library/office/system.speech.recognition.aspx

[10] [Buzo, 2011] Buzo, A., "Automatic Speech Recognition over Mobile Communication Networks," Teză de doctorat, Universitatea Politehnica din Bucureşti, România, 2011, http://speed.pub.ro/academic/research-anddevelopment-project-in-spoken-language-technology.

[11] CMUSphinx Tutorial For Developers [CMUSphinx Wiki]

[12] Erik Brown, "Windows Forms Programming with C#"

ANEXĂ

Construcția unui Sistem Simplu de Recunoaștere Automată a Vorbirii

Etape:

- Crearea dicţionarului fonetic şi a listei de foneme

mkdir phonetics // crearea unui director de lucru temporar

cd phonetics // accesarea directorului

vi rodigits.dic // Crearea şi editarea unui nou fişier numit rodigits.dic utilizând editorul vim

sed 's/ /\n/g' rodigits.dic > rodigits.phones.temp // Din această listă am selectat numai fonemele (profitând de faptul că ele conţin caracterul "_")

grep "_" rodigits.phones.temp > rodigits.phones.temp.onlyPhones // listarea fonemelor într-un nou fişier numit rodigits.phones.temp.onlyPhones

echo "SIL" >> rodigits.phones.temp.onlyPhones // Adăugarea la sfârşitul fişierului a "fonemului" SIL (acest "fonem" este utilizat pentru a modela zonele de linişte):

sort -u rodigits.phones.temp.onlyPhones > rodigits.phones.temp.onlyPhones.sorted // Sortarea fonemelor în ordine alfabetică

mv rodigits.phones.temp.onlyPhones.sorted rodigits.phones // redenumirea fişierului rezultat

rm rodigits.phones.temp* // ștergerea tuturor celorlalte fişiere temporare

- Antrenarea modelului acustic

cd ~ mkdir projects // crearea unui director numit projects în care am creat ulterior toate proiectele

cd ~/projects mkdir rodigits cd ~/projects/rodigits // crearea unui nou director numit rodigits

sphinxtrain_biosinf -t rodigits setup mkdir logdir // configurarea directorului rodigits pentru a putea fi utilizat ca proiect CMU Sphinx

cd ~/projects/rodigits

mkdir wav

cp -r /home/biosinfShare/resources/wav/SPEAKER_ID/ ~/projects/rodigits/wav/ // copierea clipurilor audio înregistrate în prealabil în directorul wav

cp /home/biosinfShare/resources/transcription/rodigits.data ~/projects/rodigits/etc/ // copierea transcrierii textuale a cuvintelor pronunţate în clipurile audio în directorul etc

cp ~/phonetics/rodigits.dic ~/projects/rodigits/etc/ // copierea dicţionarului fonetic creat în prealabil în directorul etc

cp ~/phonetics/rodigits.phones ~/projects/rodigits/etc/ //copierea listei de foneme create în prealabil în directorul etc

cp /home/biosinfShare/resources/phonetic/rodigits.filler ~/projects/rodigits/etc/ // copierea dicţionarului cu elemente acustice care nu sunt foneme în directorul etc

cp -r /home/biosinfShare/scripts/createFileIds.sh ~/projects/rodigits/etc/ // copierea scriptului createFileIds.sh în directorul etc

cp -r /home/biosinfShare/scripts/createTranscriptions.sh ~/projects/rodigits/etc/ // copierea scriptului createTranscriptions.sh în directorul etc

vi createFileIds.sh // vizualizarea scriptului createFileIds.sh utilizând editorul vim

./createFileIds.sh > rodigits.fileids.all // crearea listei tuturor clipurilor audio înregistrate

Crearea listei clipurilor audio utilizate pentru antrenare selectând primele 50 şi ultimele 30 de clipuri audio din lista tuturor clipurilor (rodigits.fileids.all)

head -50 rodigits.fileids.all > rodigits.fileids.train

tail -30 rodigits.fileids.all >> rodigits.fileids.train

diff rodigits.fileids.train rodigits.fileids.all | grep '>' | sed 's/> //' > rodigits.fileids.test // crearea unui fişier cu lista celorlalte clipuri audio ce vor fi utilizate pentru evaluare

./createTranscriptions.sh > rodigits.transcription.all // crearea transcrierii textuale (în formatul corespunzător cerut de CMU Sphinx) pentru toate clipurile audio

head -50 rodigits.transcription.all > rodigits.transcription.train tail -30 rodigits.transcription.all >> rodigits.transcription.train // crearea unui fişier care să conţină numai transcrierea textuală pentru clipurile audio ce vor utilizate pentru antrenare selectând primele 50 şi ultimele 30 de linii din transcrierea textuală

diff rodigits.transcription.train rodigits.transcription.all | grep '>' | sed 's/> //' > rodigits.transcription.test // crearea unui fişier cu lista celorlalte linii din transcrierea textuală ce vor fi utilizate pentru evaluare

cd ~/projects/rodigits/etc/ vi sphinx_train.cfg // vizualizarea fişierului de configurare al proiectului

Identificarea în acest fişier a liniileâor care specifică locaţia resurselor utilizate în procesul de antrenare (dicţionarul fonetic, lista de foneme, dicţionarul de sunete care nu sunt foneme, lista de clipuri audio de antrenare şi transcrierea textuală a acestor clipuri audio)

$CFG_DICTIONARY = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";

$CFG_RAWPHONEFILE = "$CFG_LIST_DIR/$CFG_DB_NAME.phone";

$CFG_FILLERDICT = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";

$CFG_LISTOFFILES = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.fileids";

$CFG_TRANSCRIPTFILE = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.transcription";

Modificarea acestor linii astfel:

$CFG_DICTIONARY = "$CFG_LIST_DIR/rodigits.dic";

$CFG_RAWPHONEFILE = "$CFG_LIST_DIR/rodigits.phones";

$CFG_FILLERDICT = "$CFG_LIST_DIR/rodigits.filler";

$CFG_LISTOFFILES = "$CFG_LIST_DIR/rodigits.fileids.train";

$CFG_TRANSCRIPTFILE = "$CFG_LIST_DIR/rodigits.transcription.train";

Crearea fișierelor cu coeficienți MFCC corespunzătoare clipurilor audio de antrenare cd ~/projects/rodigits/ /usr/local/sphinx/lib/sphinxtrain/scripts/000.comp_feat/slave_feat.pl

ls feat/SPEAKER_ID/*

ls feat/SPEAKER_ID/* | wc -l

ls wav/SPEAKER_ID/* | wc -l

cd ~/projects/rodigits/ sphinxtrain_biosinf run // antrenarea modelului acustic

- Crearea modelului de limbă

cd ~/projects/rodigits/etc/ vi rodigits.jsgf

#JSGF V1.0;

grammar rodigits;

public <numbers> = (zero | unu | doi | trei | patru | cinci | șase | șapte | opt | nouă) * ;

sphinx_jsgf2fsg -jsgf

rodigits.jsgf -fsg rodigits.fsg // transformarea gramaticii sarcinii de recunoaștere din format JSGF în formatul intern FSG utilizat de CMU Sphinx

vi rodigits.fsg

- Evaluarea sistemului de recunoaștere și interpretarea rezultatelor

cd ~/projects/rodigits/etc/

vi sphinx_train.cfg

$DEC_CFG_SCRIPT = 'psdecode.pl';

$DEC_CFG_SCRIPT = 'psdecode_fsg.pl'; // linia anterioară modificată

// Modificarea liniilor

$DEC_CFG_DICTIONARY = "$CFG_BASE_DIR/etc/$CFG_DB_NAME.dic";

$DEC_CFG_FILLERDICT = "$CFG_BASE_DIR/etc/$CFG_DB_NAME.filler";

$DEC_CFG_LISTOFFILES = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}_test.fileids";

$DEC_CFG_TRANSCRIPTFILE = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}_test.transcription";

$DEC_CFG_RESULT_DIR = "$CFG_BASE_DIR/result";

astfel:

$DEC_CFG_DICTIONARY = "$CFG_BASE_DIR/etc/rodigits.dic";

$DEC_CFG_FILLERDICT = "$CFG_BASE_DIR/etc/rodigits.filler";

$DEC_CFG_LISTOFFILES = "$CFG_BASE_DIR/etc/rodigits.fileids.test";

$DEC_CFG_TRANSCRIPTFILE = "$CFG_BASE_DIR/etc/rodigits.transcription.test";

$DEC_CFG_RESULT_DIR = "$CFG_BASE_DIR/result.cd_cont_200_8";

Modificarea liniei

$DEC_CFG_LANGUAGEMODEL="$CFG_BASE_DIR/etc/${CFG_DB_NAME}.lm.DMP";

ce specifică numele modelului de limbă ce va fi utilizat la decodare, astfel:

$DEC_CFG_LANGUAGEMODEL = "$CFG_BASE_DIR/etc/rodigits.fsg";

Modificarea liniei

$DEC_CFG_ALIGN = "builtin";

ce specifică numele aplicației folosite pentru alinierea textului

$DEC_CFG_ALIGN = "sclite";

Crearea fişierelor cu coeficienţi MFCC corespunzătoare clipurilor audio de evaluare:

cd ~/projects/rodigits/

/usr/local/sphinx/lib/sphinxtrain/scripts/000.comp_feat/slave_feat.pl

Execuţia acestui script are ca efect crearea a 100 de fişiere cu coeficienţi cepstralicorepunzătoare tuturor clipurilor audio înregistrate

ls feat/SPEAKER_ID/*

ls feat/SPEAKER_ID/* | wc -l

ls wav/SPEAKER_ID/* | wc -l

cd ~/projects/rodigits/

/usr/local/sphinx/lib/sphinxtrain/scripts/decode/slave.pl // începerea procesului de decodare

cd ~/projects/rodigits/

vi result.cd_cont_200_8/rodigits.align // vizualizarea rezultatelor

Decodarea clipurilor audio de evaluare folosind modele fonetice independente de context

cd ~/projects/rodigits/etc/ vi sphinx_train.cfg

Models to use.

$DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";

Modificarea liniei

$DEC_CFG_RESULT_DIR = "$CFG_BASE_DIR/result.cd_cont_200_8";

cu:

$DEC_CFG_RESULT_DIR = "$CFG_BASE_DIR/result.ci_cont";

cd ~/projects/rodigits /usr/local/sphinx/lib/sphinxtrain/scripts/decode/slave.pl // pornirea procesului de evaluare

Decodarea clipurilor audio de evaluare folosind modele fonetice dependente de context cu mai puţine densităţi Gaussiene per senone:

cd ~/projects/rodigits/etc/

vi sphinx_train.cfg

Models to use.

```
#$DEC_CFG_MODEL_NAME                                    =
"$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";
 $DEC_CFG_MODEL_NAME                                    =
"$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}_1";


$DEC_CFG_RESULT_DIR     = "$CFG_BASE_DIR/result.cd_cont_200_1";


cd ~/projects/rodigits
cp model_parameters/rodigits.cd_cont_200/mdef model_parameters/rodigits.cd_cont_200_1/
        /usr/local/sphinx/lib/sphinxtrain/scripts/decode/slave.pl
```

Antrenarea unui model acustic având în total 100 de senone

```
cd ~/projects/rodigits/etc/
vi sphinx_train.cfg
# Number of tied states (senones) to create in decision-tree clustering $CFG_N_TIED_STATES
= 100;
cd ~/projects/rodigits
sphinxtrain_biosinf run
cd ~/projects/rodigits
 ls -all model_parameters
```

Decodarea clipurilor audio de evaluare folosind modele acustice cu 100 de senone

```
cd ~/projects/rodigits/etc/
vi sphinx_train.cfg
# Models to use.
$DEC_CFG_MODEL_NAME                                     =
"$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";
$DEC_CFG_RESULT_DIR     = "$CFG_BASE_DIR/result.cd_cont_100_8";
cd ~/projects/rodigits
/usr/local/sphinx/lib/sphinxtrain/scripts/decode/slave.pl
vi logdir/decode/rodigits-1-1.log
```

Clasele folosite în aplicația finală de control al robotului

Clasa JaguarCtrl.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using EARTHLib;
using System.Drawing.Drawing2D;
using System.IO;
using Microsoft.DirectX.DirectInput;
using System.Threading;
using System.Speech.Recognition;
using System.Timers;
using Timer = System.Timers.Timer;
using NAudio.Wave;
using System.Net;
using System.Net.Sockets;

namespace DrRobot.JaguarControl
{
    public partial class JaguarCtrl : Form
    {
        static working.Worker workerObject = new working.Worker();


        DrRobotRobotConnection drRobotConnect = null;
        RobotConfig robotCfg = null;
        RobotConfig.RobotConfigTableRow jaguarSetting = null;
        private const string configFile =
"c:\\DrRobotAppFile\\OutDoorRobotConfig_4x4.xml";
        public WaveIn waveSource = null;
        public WaveFileWriter waveFile = null;

        public int contor = 0;
        public bool laser_started = false;
        public bool safe = false;
        bool started = false;
        public System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();
…

…

…
#region form funtion
        public JaguarCtrl(DrRobotRobotConnection form, RobotConfig robotConfig)
        {


           drRobotConnect = form;
           robotCfg = robotConfig;
           jaguarSetting =
(RobotConfig.RobotConfigTableRow)robotCfg.RobotConfigTable.Rows[0];
            InitializeComponent();
            timer.Tick+=new EventHandler(check_laser);
            timer.Interval = 80;
            timer.Start();
```

```
            Choices lista = new Choices();
            lista.Add(new string[] { "start laser" });
            lista.Add(new string[] {"forward"});
            lista.Add(new string[] { "turn around" });
            lista.Add(new string[] { "right" });
            lista.Add(new string[] { "robot" });
            lista.Add(new string[] { "left" });
            lista.Add(new string[] { "stop" });
            lista.Add(new string[] { "lights on" });
            lista.Add(new string[] { "turn lights off" });
            lista.Add(new string[] { "blower on" });
            lista.Add(new string[] { "blower off" });
            lista.Add(new string[] { "send" });

            Grammar dictionar = new Grammar(new GrammarBuilder(lista));
            try {
                speech_rec.RequestRecognizerUpdate();
                speech_rec.LoadGrammar(dictionar);
                speech_rec.SpeechRecognized += recunoscut;
                speech_rec.SetInputToDefaultAudioDevice();
                speech_rec.RecognizeAsync(RecognizeMode.Multiple);
            }
            catch {}


        }
private void JaguarCtrl_Shown(object sender, EventArgs e)
        {
            drRobotConnect.Hide ();
            this.Focus();
        }
private void tmrPing_Tick(object sender, EventArgs e)
        {
             try
            {
                if (drrobotMainComm != null)
                {
                    drrobotMainComm.SendCommand("PING");
                }
            }
            catch
            {
            }

        }

        private void btnEStopWheel_Click(object sender, EventArgs e)
        {
            if (btnEStopWheel.Text == "EStop Wheel")
            {
                btnEStopWheel.Text = "Resume Wheel";
                sendEStopCmd();
            }
            else
            {
                btnEStopWheel.Text = "EStop Wheel";
                sendEStopRelease();
            }
        }

        private void btnCalIMU_Click(object sender, EventArgs e)
        {
            drrobotMainComm.SendCommand("SYS CAL");
```

```
}

private void btnIRLED_Click(object sender, EventArgs e)
{
    if (btnIRLED.Text == "IRLED")
    {
        btnIRLED.Text = "IROFF";
        powerIO = powerIO | 0x40;

    }
    else
    {
        btnIRLED.Text = "IRLED";
        powerIO = powerIO & 0xbf;
    }

    string strCmd = "SYS MMC " + powerIO.ToString();
    drrobotMainComm.SendCommand(strCmd);
}

private void btnLight_Click(object sender, EventArgs e)
{
    if (btnLight.Text == "W_LED")
    {
        btnLight.Text = "W_OFF";
        powerIO = powerIO | 0x80;
    }
    else
    {
        btnLight.Text = "W_LED";
        powerIO = powerIO & 0x7f;
    }
    string strCmd = "SYS MMC " + powerIO.ToString();
    drrobotMainComm.SendCommand(strCmd);
}

private void btnFanOff_Click(object sender, EventArgs e)
{
    int cmd = 5;
    string sendServo = "SER 0 " + cmd.ToString();
    drrobotMainComm.SendCommand(sendServo);
}

private void btnFan1_Click(object sender, EventArgs e)
{
    int cmd = 6;
    string sendServo = "SER 0 " + cmd.ToString();
    drrobotMainComm.SendCommand(sendServo);
}

private void btnFan2_Click(object sender, EventArgs e)
{
    int cmd = 7;
    string sendServo = "SER 0 " + cmd.ToString();
    drrobotMainComm.SendCommand(sendServo);
}

private void btnFan3_Click(object sender, EventArgs e)
{
    int cmd = 8;
    string sendServo = "SER 0 " + cmd.ToString();
    drrobotMainComm.SendCommand(sendServo);
}
```

```
        private void myAMC_OnError(object sender,
AxAXISMEDIACONTROLLib._IAxisMediaControlEvents_OnErrorEvent e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            trackBarTurnPower.Value = -350;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            trackBarTurnPower.Value = 350;
        }

        private void button3_Click(object sender, EventArgs e)
        {

            if (button3.Text == "fata")
            {
                trackBarForwardPower.Value = 90;
                button3.Text = "stop";
            }
            else if (button3.Text == "stop")
            {
                trackBarForwardPower.Value = 0;
                button3.Text = "fata";
            }

        }

        private void button4_Click(object sender, EventArgs e)
        {
            trackBarForwardPower.Value = -90;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            trackBarForwardPower.Value = 0;
            trackBarTurnPower.Value = 0;
        }
        private void recunoscut(object sender, SpeechRecognizedEventArgs e)
        {
            int i=0;


            if (e.Result.Text == "start laser")
            {
                sendCommandLaser("BM");
                laser_started = true;
                sendCommandLaser(SCANCOMMAND);
                textBox2.Text = "Laser started";
            }

            if (e.Result.Text == "lights on")
            {
                powerIO = powerIO | 0x80;
                string strCmd = "SYS MMC " + powerIO.ToString();
                drrobotMainComm.SendCommand(strCmd);
            }
            if (e.Result.Text == "turn lights off")
            {
```

```
        powerIO = powerIO & 0x7f;
        string strCmd = "SYS MMC " + powerIO.ToString();
        drrobotMainComm.SendCommand(strCmd);

    }
    if (e.Result.Text == "forward")
    {
        if (laser_started==true && safe==true )
        {
            textBox1.Text = "Forward";
            trackBarForwardPower.Value = 200;
            sendEStopRelease();
        }
        else
        {
            MessageBox.Show("Laser not started or unsafe distance !!!");
            trackBarForwardPower.Value = 0;
            trackBarTurnPower.Value = 0;

        }
    }
    if (e.Result.Text == "turn around")
    {
        if (laser_started == true && safe == true)
        {
            trackBarTurnPower.Value = 480;
            for (i = 0; i < 1340; i++)
            {
                Thread.Sleep(1);
            }
            trackBarTurnPower.Value = 0;
            sendEStopRelease();
        }
        else
        {
            trackBarForwardPower.Value = 0;
            trackBarTurnPower.Value = 0;
        }

    }
        if (e.Result.Text == "left")
        {
            textBox1.Text = "left";
            if (laser_started == true && safe == true)
            {
                trackBarTurnPower.Value = -480;
                for (i = 0; i < 650; i++)
                {
                    Thread.Sleep(1);
                }
                trackBarTurnPower.Value = 0;
                sendEStopRelease();
            }
            else
            {
                trackBarForwardPower.Value = 0;
                trackBarTurnPower.Value = 0;
            }
        }

         if (e.Result.Text == "right")
    {
        textBox1.Text = "right";
```

```
            if (laser_started == true && safe == true)
            {
                trackBarTurnPower.Value = 480;
                for (i = 0; i < 650; i++)
                {
                    Thread.Sleep(1);
                }
                trackBarTurnPower.Value = 0;
                sendEStopRelease();
            }
            else
            {
                trackBarForwardPower.Value = 0;
                trackBarTurnPower.Value = 0;
            }

        }


            if (e.Result.Text == "robot" && started == false)
             {
                 button6.Enabled = false;
                 button7.Enabled = true;
                 started = true;
                 Thread workerThread = new Thread(workerObject.DoWork);
                 textBox2.Text = "Recording";
                 working._shouldStop = false;
                 workerThread.Start();


             }

       if (e.Result.Text == "send" && started == true)
        {
            button7.Enabled = false;
            button6.Enabled = true;
            started = false;

            textBox2.Text = "stop";
            workerObject.RequestStop();


        }
        if (e.Result.Text == "stop")
        {
            sendEStopCmd();
            textBox2.Text = "ESTOPPED";
        }
    }


        //StartBtn.Enabled = true;


    private void check_laser(object sender, EventArgs e)
    {
        float fata = disData[512];
        float minim_fata = 50000;
        int i;
        for (i = 490; i < 530; i++)
        {
            if (disData[i] < minim_fata)
            {
                minim_fata = disData[i];
```

```
            }
        }
        if (minim_fata < 1500)
        {
            safe = false;
            trackBarForwardPower.Value = 0;
            trackBarTurnPower.Value = 0;
        }
        else
        {
            safe = true;
        }
        if (label1.Text == "unu doi trei")
        {
            trackBarForwardPower.Value = 300;
        }
        if (label1.Text == "cinci patru trei")
        {
            trackBarForwardPower.Value = 0;
        }
        if (label1.Text == "zero zero trei")
        {

            sendCommandLaser("BM");
            laser_started = true;
            sendCommandLaser(SCANCOMMAND);
            textBox2.Text = "Laser started";
        }
    }

    private void button6_Click(object sender, EventArgs e)
    {
        button6.Enabled = false;
        button7.Enabled = true;
        started = true;
        Thread workerThread = new Thread(workerObject.DoWork);
        textBox2.Text = "Recording";
        working._shouldStop = false;
        workerThread.Start();
    }

    private void button7_Click(object sender, EventArgs e)
    {
        button7.Enabled = false;
        button6.Enabled = true;
        started = false;

        textBox2.Text = "stop";
        workerObject.RequestStop();


    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        label1.Text = Receiver.MesajCurent;
        label2.Text = Sender.MesajCurent;
    }


}
```

Clasa WorkerThread.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NAudio.Wave;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Windows.Forms;

namespace DrRobot.JaguarControl
{
    public partial class JaguarCtrl : Form
    {
        public class working
        {
            public static bool _shouldStop = false;
            public class Worker
            {


                public WaveInEvent waveSource = null;
                public WaveFileWriter waveFile = null;
                // This method will be called when the thread is started.
                public void DoWork()
                {
                    if (!_shouldStop)
                    {
                        //textBox2.Text = "Recording"
                        waveSource = new WaveInEvent();
                        waveSource.WaveFormat = new WaveFormat(16000, 16, 1);

                        waveSource.DataAvailable += new
EventHandler<WaveInEventArgs>(waveSource_DataAvailable);
                        waveSource.RecordingStopped += new
EventHandler<StoppedEventArgs>(waveSource_RecordingStopped);

                        waveFile = new WaveFileWriter(@"C:\ExempluAdela\Test0001.wav",
waveSource.WaveFormat);

                        waveSource.StartRecording();
                    }



                }
                void waveSource_DataAvailable(object sender, WaveInEventArgs e)
                {
                    if (waveFile != null)
                    {
                        waveFile.Write(e.Buffer, 0, e.BytesRecorded);
                        waveFile.Flush();

                    }
                }

                void waveSource_RecordingStopped(object sender, StoppedEventArgs e)
                {

                    if (waveSource != null)
                    {
                        waveSource.Dispose();
                        waveSource = null;
                    }
```

```
                    if (waveFile != null)
                    {
                        waveFile.Dispose();
                        waveFile = null;
                        Sender.SendFile();
                    }
                }
                public void RequestStop()
                {
                    waveSource.StopRecording();
                    _shouldStop = true;

                }
                // Volatile is used as hint to the compiler that this data
                // member will be accessed by multiple threads.

            }


        }
    }
}
```

Clasa Receiver.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace DrRobot.JaguarControl
{
    class Receiver
    {
        public static string MesajCurent = "Idle";


        public static void RecFile()
        {
            try
            {
                Socket socket2 = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                IPAddress ip = IPAddress.Parse("192.168.0.78");
                IPEndPoint end = new IPEndPoint(ip, 6000);
                MesajCurent = "Conectare server";

                socket2.Connect(end);

                MesajCurent = "Conectat";


                byte[] rcvBytes = new byte[1024];
                socket2.Receive(rcvBytes);
                MesajCurent = "S-a primit";
                String rcv = System.Text.Encoding.ASCII.GetString(rcvBytes);
                MesajCurent = rcv;
                //socket.Receive(buffer1);
```

```
                // MesajCurent = "S-a PRIMIT CEVA";
                //  MesajCurent = Encoding.UTF8.GetString(buffer1);
                // MesajCurent = "rcv";


                MesajCurent = rcv;

                socket2.Close();


                // socket2.Close();


            }
            catch { }

        }



    }
}
```

Clasa Sender.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NAudio.Wave;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace DrRobot.JaguarControl
{
    class Sender
    {
        public static string MesajCurent = "Idle";

        static bool connected = false;
        public static void SendFile()
        {



            try
            {
                Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);


                IPAddress ip = IPAddress.Parse("192.168.0.78");
                IPEndPoint end = new IPEndPoint(ip, 5000);
                MesajCurent = "Buffering...";
                MesajCurent = "Conectare server";


                socket.Connect(end);

                MesajCurent = "Conectat server";

                byte[] wav = File.ReadAllBytes(@"C:\ExempluAdela\Test0001.wav");
```

```
                socket.Send(wav);
                MesajCurent = "Date trimise";



                socket.Close();
                Receiver.RecFile();


                MesajCurent = "Se primeste";
            }
            catch { }

        }



    }
}
```

Clasa DrRobotComm, folosită pentru conectarea la robot:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO.Ports;
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;
using System.Globalization;
using System.IO;
namespace DrRobot.JaguarControl
{
    class DrRobotComm
    {

        private DrRobotConnectionMethod commMethod = DrRobotConnectionMethod.WiFiComm;
//1--for wifi 2 for serial

        private StreamReader readerReply = null ;
        private StreamWriter writer = null;
        private NetworkStream replyStream = null;


        private static byte[] recBuffer;

        private Thread receiveThread = null;

        SerialPort serialPort = new System.IO.Ports.SerialPort();

        //static
        private  Socket clientSocket;
        private  int localPort = 0;

        private  string lastRecvErrorMsg = String.Empty;
        private  int lastRecvError = 0;
        private  string lastSendErrorMsg = String.Empty;
        private  int socketErrorCount = 0;
```

```
//static


private string processStr = "";
private string recStr = "";

private JaguarCtrl _form;
private string comID = "MOT1";

public DrRobotComm(JaguarCtrl form, string id)
{
    _form = form;
    comID = id;
}

//here is the WiFi connecting start
internal bool StartClient(string addr, int portNum)
{
    // Connect to remote server
    commMethod = DrRobotConnectionMethod.WiFiComm;
    try
    {
        // Establish the remote endpoint for the socket
        // GetHostEntry was an attempt to do a hostname lookup
        // so that you did not have to type an IP address.
        // However, it takes a LONG time for an IP address
        // (around 20 seconds).
        //                IPHostEntry ipHostInfo = Dns.GetHostEntry(addr);
        //                IPAddress ipAddress = ipHostInfo.AddressList[0];
        IPAddress ipAddress;
        int remotePort = portNum;
        try
        {
            ipAddress = IPAddress.Parse(addr);
        }
        catch (Exception e)
        {
            lastSendErrorMsg = e.Message;
            return false;
        }

        IPEndPoint remoteEP = new IPEndPoint(ipAddress, remotePort);

        // Create a TCP socket

        clientSocket = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);


        //udp
       // clientSocket = new Socket(AddressFamily.InterNetwork,
        //   SocketType.Dgram, ProtocolType.Udp);
        //need send data to active the communication

        // Connect to the remote endpoint
        // We will wait for this to complete rather than do it
        // asynchronously

        receiveThread = new Thread((dataReceive));
        receiveThread.CurrentCulture = new CultureInfo("en-US");

        clientSocket.Connect(remoteEP);

        localPort = ((IPEndPoint)clientSocket.LocalEndPoint).Port;
```

```
                //for communication need to send a package first
                SendCommand("?A\r");



                receiveThread.Start();

                return true;

            }
            catch (SocketException e)
            {
                socketErrorCount++;
                lastRecvError = e.ErrorCode;
                lastRecvErrorMsg = e.ToString();
                return false;
            }
        }

        /// <summary>
        /// Open a serial port if in config we choose serial communication.
        /// </summary>
        /// <param name="comPort"></param>
        /// <param name="baudRate"></param>
        /// <returns>A Ccr Port for receiving serial port data</returns>
        internal bool SerialOpen(int comPort, int baudRate)
        {
            commMethod = DrRobotConnectionMethod.SerialComm;
            if (recBuffer == null)
                recBuffer = new byte[4095];

            if (baudRate < 1200)
                baudRate = 115200;

            serialPort.Close();

            string portName = "COM" +
comPort.ToString(System.Globalization.NumberFormatInfo.InvariantInfo);
            if (serialPort == null)
            {
                serialPort = new SerialPort(portName, baudRate);
            }
            else
            {
                serialPort.PortName = portName;
                serialPort.BaudRate = baudRate;
            }
            serialPort.Encoding = Encoding.Default;
            serialPort.Parity = Parity.None;
            serialPort.DataBits = 8;
            serialPort.StopBits = StopBits.One;
            serialPort.Handshake = Handshake.None;      // Handshake.RequestToSend;
//hardware flow control
            //   serialPort.WriteTimeout = 2000;
            serialPort.NewLine = "\r";
            serialPort.ReadTimeout = 0;
            serialPort.ReceivedBytesThreshold = 2;             //the shortest package is
ping package
            serialPort.ReadBufferSize = 4096;
```

```
            //serialPort.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);
            //serialPort.ErrorReceived += new SerialErrorReceivedEventHandler(portError);
            try
            {
                serialPort.Open();

                //start a thread to receive
                //       if (receiveThread == null)
                {
                    receiveThread = new Thread((dataReceive));
                    receiveThread.CurrentCulture = new CultureInfo("en-US");
                    receiveThread.Start();
                }

            }
            catch
            {

                return false;
            }
            return true;
        }

        /// <summary>
        /// to decode receive package here
        /// </summary>
        private void dataReceive()
        {
            if (commMethod == DrRobotConnectionMethod.WiFiComm)
            {
                if (readerReply != null)
                    readerReply.Close();
                if (replyStream != null)
                    readerReply.Close();
                replyStream = new NetworkStream (clientSocket );

                readerReply = new StreamReader(replyStream);

            }

            //receive data here
            bool StayConnected = true;

            byte[] recs = new byte[4095];
            int scount = 0;

            while (StayConnected)       //keep running
            {
                try
                {
                    // *serial
                    if (commMethod == DrRobotConnectionMethod.SerialComm)
                    {
                        //recStr = serialPort.ReadExisting();
                        if (serialPort.BytesToRead > 0)
                        {
                            recStr = serialPort.ReadLine();
                            processData();
                        }
                        else
                        {
                            Thread.Sleep(20);
                        }
```

```
            }
            else if (commMethod == DrRobotConnectionMethod.WiFiComm)
            {
                // wifi
                if (!replyStream.DataAvailable)
                {
                    Thread.Sleep(20);
                }
                else
                {
                    recStr = readerReply.ReadLine();
                    processData();
                }
            }
        }
        catch (Exception ed)
        {
            //need to handle some error here

        }

    }
}


void processData()
{
    if (recStr.Length == 1)
    {
        //here is the ack message, "+", "-"
        if (recStr == "+")
            _form.UpdateACkMsg(comID + "VALID");
        else if (recStr == "-")
            _form.UpdateACkMsg(comID + "INVALID");
    }
    else if(recStr.Length > 0)
    {
        _form.UpdateRcvDataMsg(comID + recStr);
    }

}


/// <summary>
/// DO NOT USE THIS COMMAND DIRECTLY.
///
/// </summary>
/// <param name="cmd"></param>
/// <returns></returns>
internal bool SendCommand(string Cmd)
{

    /// <summary>

    byte[] cmdData = ASCIIEncoding.UTF8.GetBytes(Cmd);

    if (commMethod == DrRobotConnectionMethod.SerialComm)
    {
        if (cmdData != null && serialPort != null && serialPort.IsOpen)
        {
            int packetLength = cmdData.Length;// (2 + (cmdData.Length));
            try
            {
```

```
                    serialPort.Write(cmdData, 0, packetLength );
                }
                catch
                {
                    return false;
                }
                return true;
            }
            else
                return false;
        }

        else if (commMethod == DrRobotConnectionMethod.WiFiComm)
        {
            if ((cmdData != null) && (clientSocket != null))  //&&
(clientSocket.Connected)
            {
                try
                {
                    if (clientSocket.Send(cmdData) == cmdData.Length)
                    {

                    }
                    else
                    {
                        return false;
                    }
                }
                catch
                {
                    return false;
                }


                return true;
            }
            else
                return false;
        }
        else
            return false;
    }

    /// <summary>
    /// Close the connection to a serial port.
    /// </summary>
    public void Close()
    {
        try
        {
            if (readerReply != null)
                readerReply.Close();
            if (writer != null)
                writer.Close();
        }
        catch
        {
        }


        if (serialPort != null)
        {
            if (serialPort.IsOpen)
            {
```

```
                serialPort.Close();
            }
        }
        if ((clientSocket != null))
        {
            //if (clientSocket.Connected)
                clientSocket.Close();
        }
        if (receiveThread != null)
        {
            if (receiveThread.IsAlive)
                receiveThread.Abort();      //terminate the receive thread
        }
    }

    /// <summary>
    /// True when the serial port connectionis open.
    /// </summary>
    public bool IsOpen
    {
        get { return serialPort != null && serialPort.IsOpen; }
    }
 }
 public enum DrRobotConnectionMethod
 {
     SerialComm = 0x01,
     WiFiComm = 0x02,
 }
}
```

Aplicația server scrisă în limbajul Java:

```
package app;

import java.net.*;

import edu.cmu.sphinx.api.Configuration;
import edu.cmu.sphinx.api.SpeechResult;
import edu.cmu.sphinx.api.StreamSpeechRecognizer;

import java.io.*;

public class Server extends Thread
{
   private ServerSocket serverSocket, serverSocket2;

    private Configuration configuration;
      private StreamSpeechRecognizer recognizer;
      private SpeechResult result;

      private static String ACOUSTIC_MODEL_PATH="/models/acustic";
      private static String DICTIONARY_PATH= "/models/dictionar/rodigits.dic";
      private static String GRAMMAR_PATH= "/models/gramatica";
      private static String GRAMMAR_NAME="rodigits";




   public Server(int port) throws IOException
   {
      serverSocket = new ServerSocket(port);
```

```
    serverSocket2 = new ServerSocket(6000);

      configuration = new Configuration();
           configuration.setAcousticModelPath(ACOUSTIC_MODEL_PATH);
           configuration.setDictionaryPath(DICTIONARY_PATH);
           configuration.setGrammarPath(GRAMMAR_PATH);
           configuration.setUseGrammar(true);
           configuration.setGrammarName(GRAMMAR_NAME);
           recognizer = new StreamSpeechRecognizer(configuration);


}

public void run()
{

       String number = " ";
   while(true)
   { System.out.println("server alive");
      try
      {
       Socket server = serverSocket.accept();
     Socket server2 = serverSocket2.accept();
       OutputStream os = server2.getOutputStream();
        // Socket server = serverSocket.accept();
       byte[] toSendBytes = new byte[1024];

         if(server.getInputStream() != null ){
        recognizer.startRecognition(server.getInputStream());
               while ((result = recognizer.getResult()) != null)
               {

                   number = result.getHypothesis();
                   //System.out.println("rezultat=: "+number);
                   toSendBytes = number.getBytes();
                   os.write(toSendBytes);
                   System.out.println("rezultat=: "+number);
                   os.flush();
                   System.out.println("flushed");

               }
         recognizer.stopRecognition();
               server.close();
               server2.close();

               //recognizer.stopRecognition();
               System.out.println("INCHEIAT");}
       // toSend = number;


        //server.close();
        //server2.close();
      }catch(IOException e)
      {
         e.printStackTrace();
         break;
      }
   }
}
```

```
   public static void main(String [] args)
   {
      int port = 5000;
      try
      {
         Thread t = new Server(port);
         t.start();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```