

UNIVERSITATEA “POLITEHNICA” DIN BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

CONTROLUL UNUI SISTEM HOME CINEMA PRIN COMENZI
VOCALE

PROIECT DE DIPLOMĂ

prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul Inginerie Electronică și Telecomunicații
programul de studii *Microelectronică, Optoelectronică și Nanotehnologii*

Conducător științific:

Ș.L. Dr. Ing. Horia Cucu

Prof. Dr. Ing. Corneliu Burileanu

Student:

Mihai DOAGĂ

București
2016

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

Aprobat Director de Departament :

Prof. Dr. Ing. Gheorghe Ștefan

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Doagă Mihai, grupa 443E



1. Titlul temei: **Controlul unui sistem home cinema prin comenzi vocale.**
2. Contribuția practică, originală a studentului va consta în (*în afara* părții de documentare):
Proiectul are ca scop realizarea unui sistem de control pentru un sistem home cinema folosind comenzi vocale în limba română.
Sistemul va permite controlul conținutului media pe un ecran. Transmisia de date va fi digitală, iar controlul conținutului se va face cu ajutorul unei plăcuțe de dezvoltare pe baza unor librării de detecție de voce de la un microfon extern. Contribuția proprie constă în modulul care va rula pe plăcuța de dezvoltare împreună cu o interfață grafică și construirea sistemului.
Studentul va descrie conceptual sistemul mai sus menționat și va insista cu precădere asupra programării și interfeței grafice realizate.
3. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline: Arhitectura microprocesoarelor, Microcontrolere, Programare Obiect-Orientată, Structuri de Date și Algoritmi.
4. Proprietatea intelectuală asupra proiectului aparține: Laborator Cercetare Speed
5. Locul de desfășurare a activității: UPB
6. Realizarea practică rămâne în proprietatea: Laborator Cercetare Speed
7. Data eliberării temei: decembrie 2016

CONDUCĂTOR LUCRARE:

Ș.L. Dr. Ing. Horia Cucu

Prof. Dr. Ing. Corneliu Burileanu



STUDENT:

Doagă Mihai



Declarație de onestitate academică

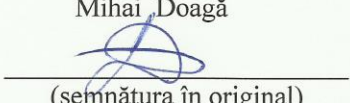
Prin prezenta declar că lucrarea cu titlul "*Controlul unui Sistem Home Cinema prin Comenzi Vocale*", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Inginerie Electronică și Telecomunicații*, programul de studii *Microelectronică, Optoelectronică și Nanotehnologii* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

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

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

București, 30.06.2016

Mihai Doagă



(semnătura în original)

CUPRINS

Cuprins	7
Lista Figurilor	9
Lista Tabelelor	11
Lista Acronimelor	13
Introducere	15
CAPITOLUL 1 Recunoașterea automată a vorbirii și filtrarea adaptivă	17
1.1 Mecanismele vorbirii.....	17
1.2 Arhitectura unui sistem de recunoaștere automată a vorbirii	18
1.2.1 Resursele fonetice, acustice și lingvistice	19
1.2.2 Modelarea acustică.....	20
1.2.3 Modelarea lingvistică.....	21
1.3 Evaluarea sistemelor de recunoaștere automată a vorbirii	22
1.4 Filtrarea adaptivă.....	23
CAPITOLUL 2 Tehnologii Software	25
2.1 Matlab.....	25
2.2 Python.....	26
2.3 XML	29
CAPITOLUL 3 Sistem de recunoaștere de cifre	31
3.1 Crearea resurselor și antrenarea modelelor	31
3.2 Evaluarea sistemului de recunoaștere	33

3.3	Optimizarea modelului acustic.....	34
3.4	Interpretarea rezultatelor	37
CAPITOLUL 4 Sistemul de control prin comenzi vocale		39
4.1	Descrierea aplicației și scenarii de utilizare	39
4.2	Schema bloc funcțională	42
4.3	Implementarea filtrării adaptive	43
4.4	Interfața grafică	49
Concluzii		53
Bibliografie		55
Anexa 1		57

LISTA FIGURILOR

Fig. 1.1 Arhitectura unui sistem de recunoaștere a vorbirii [4]	19
Fig. 1.2 Clasele de ambiguitate în fonetizarea limbii române [4].....	21
Fig. 1.3 Extragerea parametrilor MFCC [4]	21
Fig. 1.4 Schema bloc a unui filtru adaptiv [7]	24
Fig. 2.1 Interpretorul [10]	27
Fig. 2.2 Compilatorul [10]	27
Fig. 2.3 Model cod XML	29
Fig. 2.4 Model attribute XML.....	29
Fig. 3.1 Baza de date din arhitectura RAV	32
Fig. 3.2 Gramatica cu stări finite pentru recunoașterea cifrelor	33
Fig. 3.3 Implementarea gramaticii pentru recunoașterea cifrelor	33
Fig. 3.4 WER al vorbitorilor pentru sistemul independent de vorbitor	37
Fig. 4.1 Diagrama de funcționare a aplicației	40
Fig. 4.2 Gramatica cu stări finite utilizată în sistemul RAV	41
Fig. 4.3 Schema bloc a aplicației	42
Fig. 4.4 Schema bloc a filtrului adaptiv [7]	44
Fig. 4.5 Comparație dintre semnalul util și semnalul filtrat	45
Fig. 4.6 Variația în timp a coeficienților filtrului adaptiv.....	45
Fig. 4.7 Eroarea în funcție de constanta de adaptare	46

Fig. 4.8 Eroarea în funcție de ordinul filtrului	46
Fig. 4.9 Timpul de convergență în funcție de constanta de adaptare.....	47
Fig. 4.10 Timpul de convergență în funcție de ordinul filtrului	48
Fig. 4.11 Eroarea în funcție de constanta de adaptare și ordinul filtrului.....	48
Fig. 4.12 Timpul de convergență în funcție de constanta de adaptare și ordinul filtrului	49
Fig. 4.13 Interfața grafică – redare imagine.....	50
Fig. 4.14 Funcția de indexare a fișierelor și directoarelor	50
Fig. 4.15 Funcția de generare a panoului de navigare	51

LISTA TABELELOR

Tabel 3.1 Transcriere fonetică a cifrelor	32
Tabel 3.2 Rezultatele sistemului dependent de vorbitor	35
Tabel 3.3 Optimizarea sistemului independent de vorbitor	35
Tabel 3.4 Rezultatele sistemului independent de vorbitor	36

LISTA ACRONIMELOR

RAV - Recunoașterea Automată a Vorbirii
RVC - Recunoașterea vorbirii continue
FSG - Finite State Grammar
WER - Word Error Rate
SER - Sentence Error Rate
RSZ - Raport Semnal-Zgomot
XML - Extensible Markup Language
JSGF - Java Speech Grammar Format
GUI - Graphical User Interface
NLMS - Normalised Least Mean Squares
MFCC - Mel Frequency Cepstral Coefficients
HMM - Hidden Markov Model

INTRODUCERE

Motivația temei

În ultima perioadă s-a pus foarte mult accent pe noi soluții de interacțiune cu mașinile care să optimizeze modul de utilizare a acestora. O dată cu creșterea foarte mare a puterii de calcul interacțiunea cu dispozitivele inteligente devine din ce în ce mai rapidă, în anumite aplicații depășind capacitatea omului de a comunica cu acestea prin metodele clasice (tastatură, monitor, mouse, etc). Astfel, noi metode de comunicare au fost propuse și se lucrează în mod continuu la îmbunătățirea acestora, metode precum folosirea gesturilor, informațiilor fiziologice (ritm cardiac, respirație, etc), mișcarea ochilor, dar mai ales vorbirea și auzul. Cele din urmă sunt cele mai eficiente în majoritatea aplicațiilor tocmai pentru faptul că aceasta este metoda cea mai utilizată de către om în comunicarea de zi cu zi.

Pe măsură ce omul a evoluat și-a dezvoltat mecanismele optime de a comunica. Din acest motiv ne vom folosi de miile de ani de evoluție care s-au concretizat în aceste mecanisme ale vorbirii și vom încerca să le imităm prin dispozitivele din ziua de azi cu ajutorul unor sisteme specializate de recunoaștere a vorbirii.

Omenirea conturează o eră a internetului în acest moment, iar următorul pas este să interconectăm toate dispozitivele cu care interacționăm în viața de zi cu zi. Acest fenomen este cunoscut sub numele de „Internet-Of-Things”. Capacitatea de interacționa cu acestea și în alte moduri decât cele clasice (tactile) oferă omului facilitatea de a lucra cu mai multe astfel de dispozitive în paralel. Acest lucru se concretizează în creșterea eficienței omului, lucru care se urmărește în special în ultima vreme din cauza unei aglomerări continue a programului de zi cu zi. Pe măsura ce evoluăm descoperim noi limite ale capacității noastre de înțelegere, de lucru, de învățare, iar toate acestea împing la limită toate tehnologiile disponibile. Tocmai aceste tehnologii ne ajută să ne depășim limitele de eficiență, iar dezvoltarea lor este un punct esențial în evoluția umană.

Domeniul recunoașterii automate a vorbirii a fost studiat încă din anul 1932 de către cercetători precum Harvey Fletcher din laboratoarele Bell. A fost nevoie de aproximativ 30 de ani până la apariția primelor sisteme de recunoaștere continuă a vorbirii funcționale (Raj Reddy, absolvent al Universității Stanford), având ca primă aplicație comenzi vocale pentru un joc de șah. [1]

Până astăzi s-au format mari comunități științifice la nivel internațional care dezvoltă astfel de sisteme pentru un număr din ce în ce mai mare de limbi. În particular, pentru limba română comunitatea de cercetare nu este foarte dezvoltată, dar sistemele actuale fac față unui număr relativ mare de aplicații. Aspectele particulare ale limbii introduc dificultăți în dezvoltarea sistemelor, dar există și avantaje la această limbă, precum transcrierea fonetică a

celor mai multe cuvinte care nu prezintă mai multe variante de pronunție, simplificând astfel modelul fonetic.

Obiective

În cadrul acestui proiect s-a stabilit ca scop crearea unui sistem de recunoașterea automată a vorbirii care să fie independent de vorbitor, iar cu ajutorul transcrierii audio-text să poată fi controlat un sistem home cinema, mai exact conținutul multimedia disponibil pe acesta, prin intermediul unei aplicații care să ruleze local. O astfel de aplicație implică mai multe subsisteme, pe baza cărora putem contura obiectivele proiectului:

- Înțelegerea modului de funcționare a unui sistem de recunoaștere automată a vorbirii în vederea optimizării acestuia pentru sarcinile specifice proiectului;
- Implementarea unui filtru adaptiv la achiziția semnalului vocal pentru posibilitatea utilizării sistemului și în condiții de zgomot;
- Dezvoltarea unei aplicații de interpretare a comenzilor și control al fișierelor multimedia împreună cu o interfață grafică.

Sumarizarea lucrării de diplomă

Lucrarea a fost împărțită în patru capitole a căror structură reflectă chiar iterațiile prin care s-a trecut pentru realizarea aplicației de control a conținutului multimedia prin comenzi vocale.

În Capitolul 1 sunt prezentate conceptele ce stau la baza sistemelor de recunoaștere automată a vorbirii, subliniind tehnicile și soluțiile folosite în acest proiect. Subcapitolele detaliază principalele aspecte avute în vedere pe parcursul dezvoltării, dar și evaluarea sistemului. Tot în cadrul acestui capitol a fost abordată și tema filtrării adaptive pentru a explica la nivel teoretic funcționarea acestui tip de filtru, dar și limitările acestuia.

Capitolul 2 introduce tehnologiile utilizate pentru a putea crea modulele acestei aplicații, discutând performanțele, avantajele și dezavantajele acelor tehnologii cu scopul de a motiva utilizarea lor.

În Capitolul 3 este prezentat un proiect desfășurat cu scopul familiarizării conceptelor de recunoaștere automată a vorbirii și metodelor prin care acestea pot fi îmbunătățite. Pentru a concentra atenția asupra proceselor de antrenare și decodare, vocabulatură pe care s-a lucrat a fost unul redus, și anume cifrele de la zero la nouă. În acest capitol sunt prezentate iterațiile care au condus la construirea versiunii optime a sistemului.

Capitolul 4 prezintă aplicația în sine, cu fiecare modul dezvoltat pentru îndeplinirea cu succes a obiectivelor propuse. Pe parcursul acestuia vor fi prezentate atât modul de utilizare a aplicației, cât și componentele acesteia cu scopul evidențierii funcțiilor individuale.

În ultimul capitol sunt prezentate concluziile din urma dezvoltării proiectului de diplomă, sunt evidențiate contribuțiile personale, iar la final sunt discutate aspectele din cadrul proiectului care pot avea parte de îmbunătățiri ce ar permite sistemului să fie ușor de integrat în viața posivilor utilizatori.

CAPITOLUL 1

RECUNOAȘTEREA AUTOMATĂ A VORBIRII ȘI FILTRAREA ADAPTIVĂ

1.1 MECANISMELE VORBIRII

Vorbirea este una dintre cele mai mai complexe operații efectuate de organism. Aceasta este controlată de către creier, mai exact de către centri vorbirii unde cuvintele sunt decodificate pentru a transmite semnale la toți mușchii care vor contribui la producerea sunetelor. Printre aceștia se numără: plămâni, laringe, gură, limbă, abdomenul, nasul și palatul moale.

Producerea sunetelor este datorată fluxului de aer care pornește din plămâni sau diafragmă, urmând să parcurgă laringele. În această etapă fluxul de aer întâlnește corzile vocale care vor modula sunetul produs de aer la trecerea prin ele, determinând înălțimea și tonul semnalului vocal (cunoscută sub denumirea de „fonație”). [2]

Sunetele produse stau la baza vorbirii așa cum o știm, iar toate sunetele sunt împărțite în două categorii:

- Vocale – aceste tipuri de sunete sunt produse prin articulație;

- Consoane – spre deosebire de vocale, aceste tipuri de sunete necesită formarea unor obstacole în cavitatea bucală, mai exact cu ajutorul dinților și limbii prin care trece fluxul de aer.

Toate sunetele produse prin vorbire au un set de caracteristici prin care se pot clasifica în subcategorii. Aceste caracteristici sunt frecvența fundamentală, înălțimea și timbrul vocii (conținutul seriei de armonici). [3]

Astfel, *vocalele* sunt la rândul lor împărțite după următoarele aspecte fiziologice:

- După unghiul de deschidere a maxilarelor (apertura):
 - Vocale deschide: a;
 - Vocale medii (semideschise): e, ă, o;
 - Vocale închide: i, î(â), u;
- După locul de articulare (determinat de poziția mușchiului lingual):
 - Vocale anterioare: e, i;
 - Vocale centrale (neutre): a, ă, î(â);
 - Vocale posterioare: o, u;
- După rotunjirea buzelor:
 - Vocale rotunjite: o, u;
 - Vocale nerotunjite: a, e, i, ă, î(â);

În cazul *consoanelor* împărțirea pe subcategorii se poate face după apropierea de vocale:

- Consoane nesonante - specifice prin faptul că nu au caracteristici asemănătoare cu cele ale vocalelor: p, b, f, v, t, d, s, z, ț, ș, j, c, g, h;
- Consoane sonante – specifice prin faptul că au caracteristici comune cu vocalele, mai exact la articularea lor predomină tonurile muzicale: m, n, l, r.

1.2 ARHITECTURA UNUI SISTEM DE RECUNOAȘTERE AUTOMATĂ A VORBIRII

Procesul de recunoaștere automată a vorbirii (RAV) presupune transformarea unui semnal audio care conține vorbire într-o succesiune de cuvinte corespunzătoare mesajului vocal. Pentru un astfel de proces este necesară modelarea statistică a unor cantități mari de date. O astfel de modelare a fost creată ca o platformă statistică vreme de aproximativ 20 de ani de către mai multe echipe de cercetători în anii 1970-1990 printre care se numără: Baker (1975), o echipă din partea companiei IBM (1976) și o echipă din partea companiei AT&T (1983). [4]

La baza modelului statistic creat a stat ideea de a identifica cea mai probabilă secvență de cuvinte W în limba L , parte a mesajului vorbit X . Acest obiectiv poate fi exprimat matematic prin ecuația (1.1).

$$W^* = \arg \max_W p(W|X) = \arg \max_W \frac{p(X|W)p(W)}{p(X)} = \arg \max_W p(X|W)p(W) \quad (1.1)$$

În Ecuația 1.1 se ține cont de faptul că probabilitatea mesajului vorbit $p(X)$ este independentă de secvența de cuvinte W . Pentru identificarea secvenței de cuvinte X în context mesajului vocal dat va fi nevoie de identificarea termenilor din Ecuația 1.1 și anume:

- $p(W)$ – probabilitatea a priori a secvenței de cuvinte;
- $p(X|W)$ – probabilitatea mesajului vorbit dată fiind secvența de cuvinte pronunțată.

Estimarea celor doi factori se poate face în următorul mod: pentru primul se va utiliza un model de limbă, iar pentru cel de-al doilea se vor face estimări cu ajutorul unui model acustic.

Având în vedere modelele de care avem nevoie pentru a determina cât mai corect secvența de cuvinte corectă putem figura arhitectura RAV (Figura 2.1). Aceasta trebuie să asigure:

- recunoașterea prin utilizarea unei serii de parametri extrași din semnalul vocal;
- recunoașterea pe baza modelelor acustice, fonetice și lingvistice.

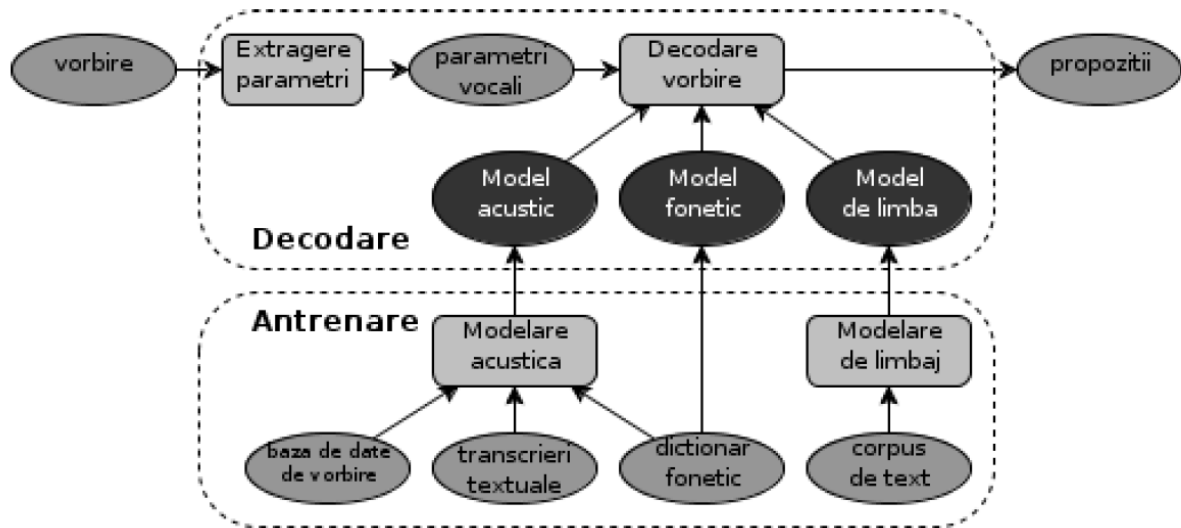


Fig. 1.1 Arhitectura unui sistem de recunoaștere a vorbirii [4]

Modelul acustic are rolul de a estima probabilitatea unei succesiuni de cuvinte din cadrul unui mesaj. Acest model este creat pe baza unităților acustice de bază precum *fonemele* sau *senonele*. Cuvintele nu pot fi folosite ca unități de bază pentru faptul că nu există astfel de modele antrenate sau date de antrenare, dar mai ales din cauza faptului că numărul de cuvinte dintr-o limbă este foarte mare.

Modelul de limbă are un rol complementar celui acustic, în sensul în care acesta va lua în calcul probabilitățile secvențelor de cuvinte din spațiul de căutare astfel încât propoziția formată să fie una validă.

Modelul fonetic conectează modelul acustic cu cel de limbă, acesta conținând diferitele pronunții pentru fiecare cuvânt. În condițiile în care cuvintele pot fi pronunțate diferit în funcție de context, acest model asociază fiecărui cuvânt din vocabular secvențele de foneme corespunzătoare.

1.2.1 Resursele fonetice, acustice și lingvistice

Un sistemul de recunoaștere automată a vorbirii procesează semnalul vocal cu ajutorul modelelor acustice, fonetice și lingvistice, iar acestea sunt antrenate cu o serie de tehnici care au la bază un set de fișiere. În Fig. 1.1 Arhitectura unui sistem de recunoaștere a vorbirii [4] avem resursele menționate resursele necesare, iar în continuare le vom analiza pe fiecare în parte pentru a defini modul prin care putem extrage informațiile în procesul numit „Antrenare”:

a) Baza de date de vorbire și transcrierile textuale – acestea sunt formate din următoarele componente:

- Un set de clipuri audio;
- Un set de fișiere text ce conțin transcrierile cuvintelor rostite în clipurile audio, dar și marcare temporale pentru cuvinte/foneme;

- Informații suplimentare ce vizează domeniul vorbirii, identitatea vorbitorului;

Baza de date este cel mai important element din antrenarea întregului sistem RAV, având cea mai mare influență asupra performanțelor sistemului ce vor fi discutate în subcapitolul următor. Însă putem enumera principalele aspecte care au impact major asupra performanțelor:

- Dimensiunea bazei de date: aceasta se reflectă prin durata totală a clipurilor audio și prin numărul total de vorbitori;
- Stilul vorbirii: Modul în care s-au pronunțat cuvintele/propozițiile din textele aferente (rostirea cuvintelor separată, continuă sau la nivel conversațional);
- Variabilitatea: prin aceasta se înțelege calitatea înregistrărilor, nivelul de zgomot și variabilitatea vorbitorilor.

- b) Dicționarul fonetic** – utilizarea acestuia se face atât la antrenarea modelului acustic, cât și cel fonetic. Această resursă specifică modul în care se pronunță cuvintele unei limbi, astfel va face legătura dintre forma scrisă și cea fonetică a cuvintelor.

Forma fonetică a fiecărui cuvânt este reprezentată de o succesiune de foneme. Pentru a construi un dicționar fonetic este necesară stabilirea unor reguli de fonetizare, dar și a unor excepții de la aceste reguli. Pentru un vocabular redus această sarcină poate dura un timp relativ scurt în cazul fonetizării manuale, dar pentru un vocabular extins sarcina efectuată în varianta manuală nu mai este o soluție.

În cazul limbii române corespondența între forma scrisă și pronunția cuvintelor este preponderent bijectivă, adică unei litere îi corespunde același fonem. Există doar câteva excepții, iar acestea sunt prezentate în tabelul de mai jos sub forma unor clase de ambiguitate.

- c) Corpusul de text** – această resursă este utilizată în antrenarea modelului lingvistic pentru sisteme care vor fi folosite la un vocabular extins. Dacă pentru sistemele de RAV de comandă și control avem un vocabular cu o dimensiune redusă, în cazul sistemelor proiectate pentru o arie mai largă de utilizare va fi necesar un corpus de text care vor determina capacitatea de predicție. Aceasta va depinde de numărul de cuvinte, deoarece afectează acuratețea probabilității de apariție a fiecărui cuvânt sau secvență de cuvinte.

1.2.2 Modelarea acustică

Sistemele de recunoaștere automată a vorbirii state-of-the-art estimează probabilitățile unor unități de vorbire de bază, precum foneme sau senone. Cu ajutorul acestora sunt create modele de foneme, astfel încât conectate mai multe astfel de modele ne permite să creăm modele pentru cuvinte, iar ulterior pentru secvențe de cuvinte. Toate acestea formează modelul acustic și sunt implementate cel mai bine utilizând modele Markov ascunse (hidden Markov models – HMM). [4]

Vorbirea este un semnal nestăționar, motiv pentru care analiza acestuia va fi făcută pe cadre scurte (20ms – 30ms) care vor deveni cvasistaționare. Pe aceste ferestre se va putea face o analiză spectrală, astfel vor fi folosite pentru a extrage parametri spectrali sau cepstrali. Parametri folosiți cel mai frecvent sunt de tip cepstral-perceptual, iar cel mai bun exemplu sunt *parametri Mel-cepstrali (Mel-Frequency Cepstrum Coefficients – MFCCs)*.

Litera	Fonemul	Exemple
e	e	cer, rest, rece
	e1	deal, te-am, ceainic, gheorghe
	i3 e	eu, el, eram, erai
i	i	fir, citit, prins
	i1	tari, rupi, deci
	i3	iarbă, doi, fier
o	o	gol, potrivit
	o1	voal, soare
u	u	sud, furtun
	w	băcăuan, nouă
c	k	casă, sac
	k1	cer, circ
	k2	chiar, ochelari
g	g	gară, olog
	g1	girafă, george
	g2	unghi, ghem
h	h	hartă, rahat
	mut	ochi, unghi
x	k s	pix, fix
	g z	exemplu, examen

Fig. 1.2 Clasele de ambiguitate în fonetizarea limbii române [4]

În Fig. 1.3 avem reprezentată o schemă de extragere a parametrilor MFCC. Putem evidenția existența unui bloc care extrage parametrii calculând vectorii de coeficienți, urmând ca apoi să fie modelați de către modelul acustic.

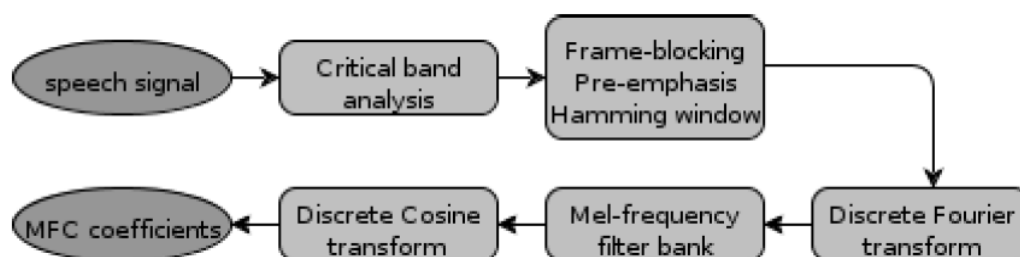


Fig. 1.3 Extragerea parametrilor MFCC [4]

1.2.3 Modelarea lingvistică

Modelarea lingvistică are o mare contribuție asupra corectitudinii recunoașterii vorbirii prin impunerea de reguli suplimentare, dar plecând de la premiza că orice secvență de cuvinte este posibilă. Cu cât impunem mai multe reguli gramaticale sau din informațiile statistice cu atât vom obține rezultate mai bune datorită faptului că reducem numărul de combinații posibile a secvențelor de cuvinte.

Principalul obiectiv al modelului este de a estima cât mai apropiat de realitate probabilitatea ca o anumită secvență de cuvinte să reprezinte o propoziție în limba respectivă. Astfel modelul de limbă va fi folosit în procesul de decodare a secvenței acustice.

Realizarea unui astfel de model, mai exact crearea probabilităților pentru posibilele secvențe de cuvinte, poate fi făcută prin numărarea evenimentelor de apariție a secvenței W (Ecuația

2.1). Această metodă are dezavantajul unei resurse mari de calcul necesară. O altă modalitate este bazată pe determinarea probabilității unei secvențe de cuvinte prin calculul probabilității fiecărui cuvânt, luând în considerare o serie de cuvinte anterioare. Această soluție poartă numele de Modelul N-gram. Probabilitatea secvenței W poate fi exprimată conform formulei 2.1.

$$P(W) = P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1, \dots, w_{n-1}) \quad (2.1)$$

Din relația (2.1) reiese faptul că estimarea probabilității unei secvențe de cuvinte este împărțită în sarcini mai mici care estimează probabilitatea pentru fiecare cuvânt în parte pe baza secvenței de cuvinte anterioară acestuia. Parametrul N din denumirea modelului, N-gram, face referire la numărul cuvintelor anterioare care contribuie la calculul probabilității cuvântului curent. Dimensiunea acestui parametru se va reflecta în precizia detecției sistemului. Cu cât este mai mare volumul de date, cu atât mai mare va fi și modelul lingvistic.

Modelul N-gram prezintă totuși un dezavantaj, și anume faptul că indiferent de dimensiunea volumului de date folosit pentru antrenarea modelului vor exista n-grame în textele de evaluare pe care nu le avem și în modelul lingvistic. Pentru aceste situații vor trebui ajustate probabilitățile calculate anterior pe baza secvențelor de cuvinte, mai exact pe baza numărului de apariție a n-gramelor din cadrul corpusculilor folosiți pentru antrenare. Acest proces de ajustări face parte dintr-un set de metode cunoscute sub numele de metode de netezire. [5]

1.3 EVALUAREA SISTEMELOR DE RECUNOAȘTERE AUTOMATĂ A VORBIRII

Evaluarea sistemului de recunoaștere automată a vorbiri este una din cele mai importante etape în optimizarea unui astfel de sistem. Prin metodele de evaluare se va determina gradul de corectitudine a transcrierilor, determinând astfel cât de potrivit este sistemul pentru diferite sarcini de transcriere a vorbirii. Testele efectuate în cadrul evaluării necesită un set de clipuri audio cu înregistrări care să fie cât mai asemănătoare cu sarcina reală la care va fi utilizat sistemul. Acest set de înregistrări vor forma baza de date de evaluare.

În cadrul evaluării întregului sistem de recunoaștere vocală se iau în calcul și performanțele modelului de limbă. Pentru aceasta, există anumite criterii de evaluare specifice pentru acest model, mai exact pentru modelul N-gram, care să determine performanțele acestuia. Printre acestea se numără:

- Accesările N-gram

La construirea unui model N-gram trebuie ținut cont de apariția unei probleme dată de împrăștierea datelor. Metoda Back-off este folosită pentru a rezolva acest inconvenient, având la bază mai multe modele de limbă și creând o versiune interpolată a tuturor acelor modele. Dacă avem la dispoziție modele unigram, bigram sau trigram putem crea modele de limbă interpolate ca o combinație liniară a modelelor de limbă cunoscute. [5]

Accesările trigramelor sunt calculate ca fiind raportul dintre numărul de cazuri în care modelul a putut folosi ambele cuvinte anterioare și numărul de cazuri în care modelul a folosit modelul bigram sau unigram cu scopul de a identifica probabilitatea n-gram curentă:

$$\text{Accesări trigram } [\%] = \frac{\text{număr accesări trigrame}}{\text{număr de cuvinte}} \times 100 \quad (2.2)$$

- Perplexitatea

Puterea de predicție a unui model de limbă poate fi determinat prin măsurarea probabilităților pe care modelul le atribuie secvențelor de cuvinte de test. Această măsurare poartă numele de perplexitate și reprezintă media geometrică a probabilităților corespunzătoare fiecărui cuvânt din setul W din modelul de limbă

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1, \dots, w_{i-1})}} \quad (2.3)$$

În acest caz, pentru a avea un sistem performant urmărim să avem o valoare redusă a perplexității, asta traducându-se ca un număr mai mic de confuzii făcute în procesul de recunoaștere a vorbirii.

- OOV (Out Of Vocabulary words – cuvinte în afara vocabularului)

Indiferent de dimensiunea vocabularului folosit, implicit a bazei de date pentru modelul de limbă, în realitate vor apărea cuvinte care nu se regăsesc printre cele cunoscute de sistem, iar acest lucru duce la incapacitatea modelului de a identifica corect cuvântul pe baza unor probabilități de apariție. Parametrul OOV indică un procent de cuvinte care nu se află în vocabular și se calculează după formula (2.4).

$$OOV[\%] = \frac{\text{numărul de OOV}}{\text{numărul de cuvinte}} \times 100 \quad (2.4)$$

Procesul de evaluare are la bază compararea unei secvențe de cuvinte de referință cu cea returnată de sistem pe baza vorbirii de la intrare. Principalul indicator pentru o astfel de comparație este cel de la nivel de cuvânt, numit WER (Word Error Rate – rata de eroare la nivel de cuvânt). Acesta se calculează făcând comparația descrisă anterior între două cuvinte. Erorile care pot apărea sunt de trei tipuri:

- Inserție – în acest caz în secvența de cuvinte a fost introdus un cuvânt eronat pe o anumită poziție;
- Ștergeri – din secvența de cuvinte transcrisă de către sistem pe o anumită poziție lipsește cuvântul corespunzător;
- Substituții – în secvența de cuvinte pe o poziție anume a fost tradus un cuvânt corespunzător celui de referință, dar eronat.

După alinierea celor două secvențe de cuvinte și identificarea celor trei tipuri de erori poate fi calculată eroarea la nivel de cuvânt cu formula (2.5).

$$WER[\%] = \frac{\text{Număr de inserții} + \text{Număr de ștergeri} + \text{Număr de substituții}}{\text{Număr de cuvinte în textul de referință}} \times 100 \quad (2.5)$$

Un alt indicator al performanței sistemului de recunoaștere automată a vorbirii este SER (Sentence Error Rate – rata de eroare la nivel de propoziție). Acesta poate fi de asemenea util, depinzând și de aplicația la care va fi folosit sistemul. SER se calculează conform formulei (2.6) având la bază eroarea la nivel de cuvânt, dar luând în considerare apariția acesteia în cadrul unei propoziții.

$$SER[\%] = \frac{\text{Număr de propoziții cu cel puțin o eroare la nivel de cuvânt}}{\text{Număr de propoziții în textul de referință}} \times 100 \quad (2.6)$$

1.4 FILTRAREA ADAPTIVĂ

Conceptul de filtrare adaptivă a fost pentru prima dată introdus de către John Kelly și Ben Logan de la laboratoarele Bell Telephone în anul 1965. Aceștia l-au creat cu scopul de a anula ecoul printr-un filtru adaptiv transversal prin folosirea semnalului util în realizarea adaptării. [7]

Utilitatea unor astfel de filtre apare în situațiile în care se dorește eliminarea unui zgomot suprapus peste semnalul util. Un exemplu ar fi zgomotul produs de motoarele unui avion care afectează semnalul util reprezentat de vocea piloților în timpul comunicării cu turnul de control.

Soluția aparent simplă de extragere directă a semnalului zgomotos din semnalul achiziționat nu este posibilă, deoarece zgomotul achiziționat de la senzorul (microfonul) de referință nu este același cu zgomotul suprapus peste semnalul util. Acesta este în general filtrat și întârziat în comparație cu cel de la sursa de zgomot, iar extragerea lui directă ar conduce chiar la un raport semnal-zgomot mai mare.

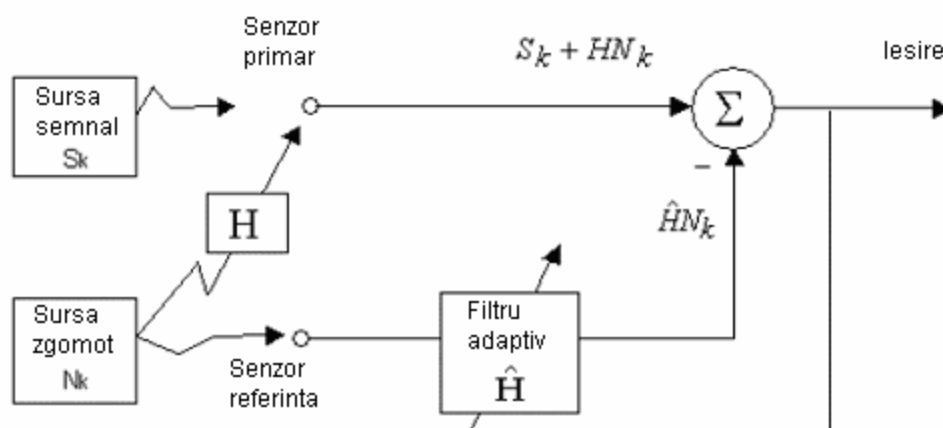


Fig. 1.4 Schema bloc a unui filtru adaptiv [7]

Conform Fig. 1.4, senzorul primar achiziționează semnalul util (provenit de la sursa de semnal S_k) și zgomotul (provenit de la sursa N_k), dar întârziat și/sau filtrat. Această modificare a semnalului zgomot este echivalentă cu trecerea printr-un filtru H . Astfel, semnalul înregistrat la senzorul primar este $S(n) + H * N_k(n)$.

Filtrul adaptiv prelucrează semnalul zgomotos cu scopul ca la ieșirea acestuia să rezulte un semnal cât mai apropiat de zgomotul suprapus peste semnalul util, și anume $H * N(n)$, urmând ca ulterior acesta să fie scăzut din semnalul înregistrat de senzorul (microfonul) principal. Acest lucru se poate obține plecând de la premisa că semnalul util și zgomotul sunt două semnale necorelate. În același timp zgomotul achiziționat de senzorul principal este corelat cu cel de la senzorul de referință, iar această proprietate este folosită în generarea coeficienților filtrului adaptiv, care este unul de tip FIR.

Trebuie menționat că acest tip de filtru presupune în plus ca semnalul util să nu fie prezent la senzorul de referință pentru a nu se produce o corelație și cu acesta în timpul generării dinamice a coeficienților filtrului adaptiv.

CAPITOLUL 2

TEHNOLOGII SOFTWARE

2.1 MATLAB

Matlab (Matrix Laboratory) este un sistem software interactiv creat pentru calcule numerice și grafice. După cum sugerează și numele, software-ul are ca principal obiectiv calculul cu matrici, și anume: sisteme de ecuații liniare, calcul de valori proprii și vectori proprii (pentru matrici). La aceste capacități de calcul se adună generarea de grafice cu o putere mare de extensie prin programe scrise în propriul limbaj de programare, astfel oferind o mare diversitate în reprezentarea valorilor cu care se lucrează. [9]

Acest software dedicat este proiectat să facă calcule numerice cu o precizie finită, astfel el va genera în majoritatea cazurilor rezultate aproximative.

Structura programelor MatLab

În cadrul mediului Matlab există două variante de a rula comenzile:

- Linia de comandă: în acest caz fiecare linie de comandă introdusă este prelucrată imediat după introducerea acesteia, iar rezultatul este afișat imediat după acestea;
- Fișiere cu programe scrise: aceste fișiere conțin un set de instrucțiuni din Matlab și au capacitatea de a apela și alte fișiere cu programe Matlab. Aceste fișiere sunt recunoscute prin extensia “.m”.

Programele create în Matlab pot fi scrise în două moduri:

- Sub formă de fișiere “script”: aceste fișiere externe conțin un set de comenzi, iar la apelarea acestora se va executa secvența de comenzi din cadrul programului. Dezavantajul acestui mod de scriere a programului este reprezentat de faptul că nu poate fi integrat în alte programe mai mari, cu alte cuvinte să fie folosit la rândul lui ca modul component al unui proiect de dimensiuni mai mari. Astfel, ele pot fi folosite pentru rezolvarea unor probleme care pot fi scrise și în linia de comandă, dar pentru faptul că este un număr mai mare de comenzi succesive se preferă executarea lor ca un întreg.

Variabilele folosite în cadrul unui program de tip script vor rămâne salvate în zona de memorie a aplicației după ce scriptul a fost executat complet;

- Sub formă de fișiere “funcție”: în aceste tipuri de fișiere conțin cuvântul cheie „function” în prima linie. Prin acest mod de scriere a programului apare avantajul de a lucra cu argumente.

Din punct de vedere al variabilelor, cele declarate în interiorul fișierului funcției vor fi localizate doar la nivelul acesteia, iar după ce funcția a fost executată vor rămâne în memorie doar variabilele de la ieșirea acesteia.

2.2 PYTHON

Python este un limbaj de programare de nivel înalt, alăturându-se altor limbaje din această categorie precum C++, PHP sau Java. Fiind un limbaj de scripting, Python este limbaj interpretat. Lipsa etapei de compilare implică un timp mai scurt în procesele de dezvoltare și debugging. [10]

La capătul opus acestei categorii de limbaje de nivel înalt, există limbajele de nivel inferior (limbaj de asamblare sau cod mașină). Codul executat direct de către procesor este cunoscut ca limbajul mașină, adică setul de instrucțiuni în binar. Acesta este singurul limbaj care poate fi înțeles de către un procesor. Din acest motiv programele scrise într-un limbaj de nivel înalt trebuie procesate înainte de a putea fi executate, iar acest proces este consumator de timp și resurse. În acest moment putem pune în balanță avantajele și dezavantajele limbajelor de nivel înalt.

De partea limbajelor de nivel inferior putem evidenția avantajele de a rula mai rapid și de a fi mai puternice în cazul unor aplicații specializate. Pe de altă parte, limbajele de nivel înalt vin cu o serie de avantaje care le justifică popularitatea din ziua de azi:

- Timpul de scrierea este redus substanțial;
- Volumul de cod este mult mai mic;
- Sunt mult mai ușor de citit și înțeles;
- Corectitudinea lor este mai ridicată datorită sintaxei clare;
- Sunt **portabile** – pot rula pe mai multe tipuri de calculatoare și platforme

Revenind la principalul dezavantaj al limbajelor de nivel înalt, și anume necesitatea procesării codului pentru transformarea în cod mașină, acesta este rezolvat prin două metode:

- **Interpretare**
- **Compilare**

Prima soluție se folosește de un interpretor care citește codul scris în nivel înalt și îl execută linie cu linie, executând comenzile pe măsură ce parcurge programul. Această tehnică aduce

marele avantaj al unei viteze mari la execuție și la timpul de programare, oferind cel mai scurt timp pentru etapa de iterații din dezvoltarea unei aplicații.



Fig. 2.1 Interpretorul [10]

La nivel schematic această soluție este simplă: preia codul sursă, îl execută puțin câte puțin pe măsură ce îl parcurge, iar răspunsul este dat în timp real la ieșire.

A doua soluție presupune o etapă intermediară care creează codul obiect sau executabil. Compilatorul citește programul și îl traduce complet în cod obiect înainte ca acesta să fie rulat. Avantajul este că odată compilat, codul poate fi executat în repetate rânduri fără alți pași suplimentari. Tot în etapa de compilare este verificată și corectitudinea sintaxei codului, reprezentând un alt avantaj al acestei etape intermediare de creare a codului obiect.



Fig. 2.2 Compilatorul [10]

Majoritatea limbajelor moderne se folosesc de ambele soluții. Într-o primă etapă sunt compilate în limbaj de un nivel inferior, numit **byte code**, și apoi sunt interpretate de o mașină virtuală. În aceeași manieră Python se folosește de ambele procese, dar este cunoscut ca fiind un limbaj interpretat datorită modului de interacție cu acesta.

Interpretorul din Python poate fi folosit în două moduri:

- Din linia de comandă: codul poate fi scris în linia de comandă Python, iar interpretorul returnează în timp real rezultatul;

Această soluție este utilă pentru situațiile în care dorim să testăm porțiuni mai mici de cod, ca un fel de ciornă pentru proiectul la care se lucrează.

- Printr-un program complet scris într-un fișier, având terminația specifică „*.py”.

Cu această variantă interpretorul va executa conținutul fișierului ca un întreg (cunoscut sub numele de *cod sursă*).

Python vs. Matlab

În cadrul proiectului de față a fost folosit Matlab pentru unul din principalele module, și anume „Filtrarea adaptivă”, în întreg procesul de testare și optimizare. Însă pentru a răspunde cerințelor proiectului întreg, Matlab nu a putut fi folosit și a fost utilizată o altă soluție software cu capacități mai extinse – Python. În continuare o să analizăm diferențele dintre cele două medii de programare.

Ecosistemul științific al limbajului de programare Python este într-o continuă dezvoltare, în principal datorită faptului că Python este gratuit, open-source și devine din ce în ce mai puternic.

Python este prin definiție un limbaj de programare. Pe lângă limbajul de programare și interpretorul pe care le-am descris mai sus, Python are acces la o librărie dezvoltată. Această

librărie este orientată către programare în general și conține module pentru diverse sarcini specifice, sisteme de operare, procese simultane, rețea, baze de date și multe altele.

Matlab este un mediu de programare și limbaj de programare comercial. Librăria standard a acestuia nu este foarte dezvoltată pentru funcționalitatea de programare, dar conține funcționalități dezvoltate de lucru cu matrici și de procesare de informații și reprezentare grafică.

Pentru a beneficia de aceleași funcționalități și în Python este necesară instalarea unor pachete dedicate precum: Numpy, Scipy, Matplotlib.

Mai mult decât atât, există librării dedicate creării de interfețe grafice pentru aplicațiile dezvoltate, librării care să ofere acces la resursele mașinii pe care rulează Python, dar și librării precum Cython create pentru îmbunătățirea vitezei algoritmilor prin convertirea codului Python în cod C.

Pentru o imagine mai clară asupra acestei comparații putem sumariza diferențele dintre cele 2 soluții evidențiind avantajele și dezavantajele fiecăreia. [11]

MATLAB:

- **Dezavantaje**

- Nu există acces la codul sursă al funcțiilor folosite, iar asta implică imposibilitatea verificării corectitudinii acestora;
- Cost ridicat;
- Mathworks – producătorul software-ului – impune restricții de **portabilitate**;
- Fiind dezvoltat de o companie privată, funcționalitățile nu pot fi extinse de către comunitate.

- **Avantaje**

- Librărie dezvoltată de funcții matematice;
- Comunitate științifică dezvoltată, în special în mediul academic;
- Pachetul standard include în general majoritatea funcțiilor necesare, pe când în Python este necesară instalarea pachetelor separat.

PYTHON

- **Avantaje**

- Gratuit și open-source;
- Sintaxa clară;
- Având un design bine gândit, facilitează transpunerea ideilor în cod funcțional. În plus, are o varietate de tipuri de date precum liste, seturi și dicționare.
- Lucrul cu șiruri de caractere este facil;
- Este open-source și **portabil** (funcționează pe Windows, Linux și OS X);
- Clase și funcții: acestea pot fi definite oriunde (linia de comandă sau în fișierele, fie modul, fie script).
- Librării pentru interfețe grafice.

Împreună cu avantajul că este gratuit și este accesibil pentru diverși dezvoltatori să creeze noi pachete sau alte utilitare care să extindă puterea și aplicabilitatea limbajului, Python este o alegere mai bună pentru un număr mai mare de aplicații.

2.3 XML

XML (eXtensible Markup Language) este folosit pentru descrierea și transmiterea datelor. Cu ajutorul acestui standard sunt implementate structuri de date menite să transfere informația prin internet, dar și prin rețele private în cel mai eficient și organizat mod.

Un exemplu minimal de conținut al unui fișier XML este prezentat mai jos.

```
<note>
  <to>Giani</to>
  <from>Alexander</from>
  <heading>Notification</heading>
  <body>Body text as a Lorem Ipsum for content filling!</body>
</note>
```

Fig. 2.3 Model cod XML

XML este strâns legat de limbajul HTML, deoarece ambele provin din SGML (Standard Generalized Markup Language) care este limbajul imperativ din care au fost produse atât HTML, cât și XML. SGML nu este un limbaj în adevăratul sens al cuvântului, ci mai degrabă o serie de comenzi interpretate de către alt program. [12]

Limbajul XML permite definirea atributelor pentru elemente și caracteristicile descriptibile ale acestora. Aceste atribute sunt poziționate în eticheta de început a unui element, precum în exemplul de mai jos.

```
<?xml version="1.0" standalone="yes"?>
<conversatie>
  <salut>Hei, Marius</salut>
  <reactie>Salutare!</reactie>
</conversatie>
```

Fig. 2.4 Model attribute XML

Beneficiile aduse de XML au fost revoluționare, motiv pentru a fost adoptat foarte rapid de mulți dezvoltatori, iar în prezent este folosit în multe aplicații. Acesta s-a remarcat prin simplitate și prin capacitatea de a face față unor cantități mari de informații structurate și organizate eficient.

În cadrul acestui proiect cu ajutorul XML este implementată comunicația cu serverul de transcriere automată a vorbirii.

CAPITOLUL 3

SISTEM DE RECUNOAȘTERE DE CIFRE

3.1 CREAREA RESURSELOR ȘI ANTRENAREA MODELELOR

Sistemul de recunoaștere de cifre utilizează arhitectura unui sistem de recunoaștere automată a vorbirii (RAV) cu scopul transformării unui semnal audio în textul corespunzător, acesta fiind reprezentat de o succesiune de cuvinte dintr-un vocabular cu dimensiune mică. Acest vocabular este format din cifrele de la 0 la 9.

În prima etapă sistemul va fi antrenat pe un singur vorbitor, urmând ca după evaluarea rezultatelor intermediare să fie antrenat pe un număr mai mare de vorbitori pentru a fi transformat într-un sistem independent de vorbitor. După această etapă sistemul este din nou evaluat, iar rezultatele vor fi interpretate, așteptându-ne ca acestea să fie mai slabe decât în cazul sistemului antrenat cu un singur vorbitor.

Înregistrarea clipurilor audio

Pentru antrenarea sistemului dependent de vorbitor este nevoie de o bază de date cu clipuri audio și transcriere textuale aferente. Clipurile audio au un format standard, o secvență de 12 cifre rostite continuu. Baza de date folosită are o dimensiune de 100 de astfel de clipuri audio. În Fig. 3.1 este evidențiată secțiunea pe care o creăm în această etapă din cadrul unei arhitecturi de sistem RAV.

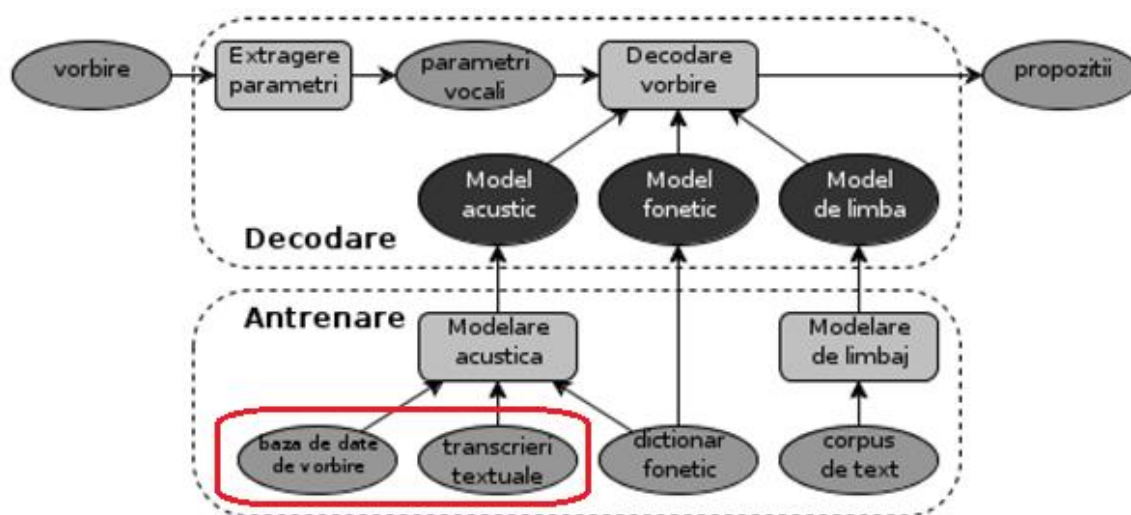


Fig. 3.1 Baza de date din arhitectura RAV

Dicționarul fonetic

Această resursă stă la baza modelului fonetic, dar și a modelului acustic. Dicționarul fonetic face corespondența dintre forma scrisă și forma fonetică a cuvintelor, reprezentate în acest caz de cifre. Forma fonetică a cuvintelor este dată de o succesiune de unități de bază ale limbii numite foneme.

Dicționarul fonetic trebuie să fie compus din toate cuvintele pe care dorim să le decodăm, dar și de transcrierile fonetice ale acestora pentru a avea rezultate bune în timpul decodării. Acest lucru este necesar deoarece acesta face legătura dintre modelul de limbă (modelul care indică cum se succed cuvintele în limba utilizată) și modelul acustic (modelul care indică cum se produc sunetele specifice limbii). [4]

La crearea dicționarului au fost scrise toate cuvintele necesare vocabularului utilizat alături de transcrierile fonetice ale acestora. Fiecare fonem este codificat sub forma unor caractere pe care le-am folosit pentru a crea transcrierea fonetică. Aceste codificări sunt preponderent reprezentate chiar de literele în cauză, iar excepțiile apar la literele sau grupurile de litere care au mai multe sunete corespondente, în funcție de context. Un exemplu de codare este cel de mai jos.

Tabel 3.1 Transcriere fonetică a cifrelor

Cuvânt	Transcriere fonetică
zero	z e r o
unu	u n u
nouă	n o l w a l

În punctul acesta avem toate resursele necesare pentru *antrenarea modelului acustic*. Din totalul de 100 de fișiere audio împreună cu transcrierile lor textuale vom utiliza un număr de 80 pentru antrenarea modelului acustic, iar restul de 20 de fișiere audio vor fi utilizate în etapa de testare a sistemului.

Modelul de limbă

Recunoașterea vorbirii continue având un vocabular cu cifre, implică un vocabular redus, face ca soluția optimă la crearea modelului de limbă să fie folosirea unui model de limbă de tip gramatică cu stări finite (FSG – Finite State Grammar). Acesta este mai performant pentru

sistemul de față în comparație cu modelele de limbă statistice, deoarece cuvintele (cifrele) apar în cadrul secvențelor din cuvinte cu probabilități aproximativ egale.

Modelul de limbă de tip FSG pe care îl utilizăm poate fi reprezentat sub forma unui graf figurând cuvintele ca nodurile grafului, iar tranzițiile între cuvinte ca arcele grafului. În plus, arcelor le poate fi atribuit un cost care reprezintă probabilitatea ca un cuvânt să fie precedat de un altul.

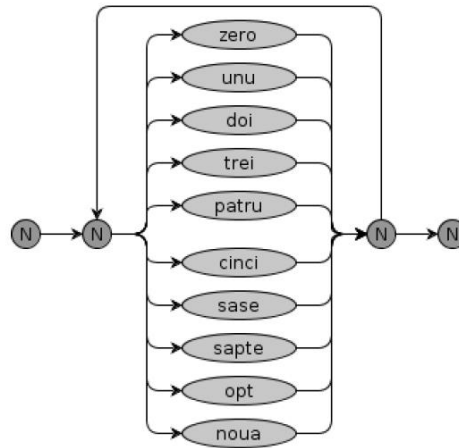


Fig. 3.2 Gramatica cu stări finite pentru recunoașterea cifrelor

În Fig. 3.2 observăm cum poate fi parcursă o astfel de gramatică, urmărind tranzițiile. Această gramatică a fost implementată în proiectul de recunoaștere de cifre cu ajutorul formatului Java Speech Grammar (JSGF). Urmând sintaxa acestui format, gramatica a fost scrisă precum în Fig. 3.3.

```
#JSGF V1.0;
grammar rodigits;
public<numbers> = (zero | unu | doi | trei | patru | cinci | șase | șapte | opt | nouă) * ;
```

Fig. 3.3 Implementarea gramaticii pentru recunoașterea cifrelor

După acești pași avem pregătite toate resursele necesare generării celor trei modele: modelul acustic, modelul de limbă și modelul fonetic. În continuare vom evalua această primă variantă de sistem de recunoaștere a cifrelor dependent de vorbitor, apoi și independent de vorbitor după ce se face o nouă antrenare a modelului acustic cu un număr mare de vorbitori.

3.2 EVALUAREA SISTEMULUI DE RECUNOAȘTERE

Sistem dependent de vorbitor

Pentru sistemul dependent de vorbitor care a fost antrenat cu 80 de clipuri audio a câte 12 cifre evaluarea se va face cu alte 20 de clipuri audio similare realizate cu vocea aceluiași vorbitor.

În procesul de evaluare se urmărește în principal criteriul de evaluare numit rata de eroare la nivel de cuvânt (WER – Word Error Rate), dar se vor lua în calcul și alte criterii precum rata de eroare la nivel de propoziție (SER – Sentence Error Rate) sau erorile din componența WER (erori de inserție, erori de substituție și erori de ștergere) cu scopul de a monitoriza posibilele tipare. Acestea pot indica anumite cauze de apariție a erorilor prin faptul că apar într-un procentaj mult mai mare decât restul tipurilor de erori. Totodată se vor vizualiza și rezultatele evaluării la fiecare clip audio pentru a ne verifica dacă anumite fișiere pot avea

probleme la nivel de corespondență cu transcrierile textuale aferente sau probleme legate de calitatea semnalului.

Configurația sistemului inițială pentru antrenarea modelului acustic a fost făcută cu un număr de 200 de senone și 8 densități Gaussiene de probabilitate.

În urma evaluării prin alinierea transcrierilor textuale de referință cu transcrierile textuale ipotetice (cele rezultate în urma procesului de decodare) am obținut următoarele rezultate:

- WER = 29,6% din care:
 - Erori de inserție = 1,3%
 - Erori de substituție = 10,4%
 - Erori de ștergere = 17,9%
- SER = 80%

Sistem independent de vorbitor

Pentru ca sistemul de recunoaștere a cifrelor să devină independent de vorbitor este nevoie ca la antrenarea modelului acustic să fie folosit un set de clipuri audio cu mai multe voci diferite, care să acopere cât mai multe tipare de voci. Astfel sistemul va recunoaște un număr mult mai mare de sunete care să corespundă aceluiași fonem datorită faptului că acum sistemul a fost antrenat cu mai multe voci pentru aceleași cuvinte.

Antrenarea modelului acustic a fost făcută cu clipuri audio de la 80 de vorbitori dintr-o bază de date, fiecare cu câte 80 de clipuri audio cu aceeași transcriere textuală aferentă precum cele folosite la antrenarea sistemului dependent de vorbitor. De asemenea, configurația sistemului este aceeași precum la sistemul dependent de vorbitor, și anume cu 200 de senone și 8 densități Gaussiene de probabilitate.

Evaluarea sistemului independent de vorbitor s-a făcut în 2 etape. În prima etapă, fiind una intermediară, s-au folosit alte clipuri audio de la aceeași vorbitori. Rezultatele sunt exprimate ca o medie a fiecărui parametru de evaluare între toți vorbitorii și au următoarele valori:

- WER = 7,3% din care:
 - Erori de substituție = 3,2%
 - Erori de ștergere = 0,6%
 - Erori de inserție = 3,5%
- SER = 43%

Observăm că antrenarea cu mai multe voci aduce rezultate mai bune. În acest punct putem trage o primă concluzie și anume că rezultatele pot avea variații foarte mari în momentul în care aducem modificări la antrenarea modelului acustic. De aceea dorim să optimizăm sistemul de recunoaștere de cifre, iar această etapă este elaborată în continuare în subcapitolul următor.

3.3 OPTIMIZAREA MODELULUI ACUSTIC

Optimizările pe care le putem face pentru modelul acustic țin de parametrii pe care îi setăm la configurația sistemului. Aceștia specifică numărul de senone utilizate și numărul de densități Gaussiene de probabilitate. Valorile celor doi parametrii vor fi variate și apoi evaluate performanțele, urmărind ca prin aceste iterații să identificăm configurația optimă.

Sistem dependent de vorbitor

Pentru sistemul dependent de vorbitor vom pleca de la configurația inițială și în prima fază vom varia numărul de densități Gaussiene de probabilitate, trecând prin valorile 8, 4, 2 și 1. Apoi modificăm numărul de senone la 100 și repetăm trecerea prin valorile anterioare pentru numărul de densități Gaussiene de probabilitate.

În urma efectuării tuturor iterațiilor menționate anterior, având ca parametru de evaluare principal WER, am centralizat rezultate în Tabelul 4.2 pentru a observa mai clar evoluția performanțelor.

Tabel 3.2 Rezultatele sistemului dependent de vorbitor

WER [%]	Număr senone	
Densități Gaussiene / senone	200	100
1	26,3	13,3
2	25	2,1
4	27,5	7,5
8	29,6	0,4

În urma procesului de optimizare observăm o îmbunătățire majoră a performanțelor sistemului. Dacă inițial cu configurația de 200 de senone și 8 densități Gaussiene per senone aveam o eroare la nivel de cuvânt de 29,6%, am ajuns doar prin schimbarea numărului de senone la 100 să obținem un rezultat mult mai bun, de doar 0,4%. Mai mult decât atât, observăm o diferență mare între rezultatele în cazul utilizării a 100 de senone în comparație cu 200, indiferent de numărul de densități Gaussiene. Acest lucru ne permite ca în optimizările viitoare să variem doar numărul de densități Gaussiene per senone, unde performanțele nu diferă foarte mult, iar numărul de senone să rămână 100 în toate testările, dat fiind faptul că rezultatele sunt mult superioare.

Sistem independent de vorbitor

Procesul de optimizare în cazul sistemului antrenat pentru a fi independent de vorbitor este similar celui pentru sistemul dependent de vorbitor. Pentru acesta au fost folosite aceleași fișiere cu care s-a făcut antrenarea, evaluând sistemul pentru diverse configurații. Rezultatele sunt trecute în Tabelul 3.3.

Tabel 3.3 Optimizarea sistemului independent de vorbitor

WER [%]	Număr senone	
Densități Gaussiene / senone	100	200
1	13,4	18
2	8,2	11,3
4	4,8	7,1
8	3,1	7,2
16	2,6	
32	2,2	
64	1,8	
128	1,9	
256	1,6	

De remarcat a fost faptul că îmbunătățirile performanțelor la ultima iterație (la trecerea de la 4 la 8 densități Gaussiene per senone) au fost relativ mari, motiv pentru care am continuat aceste iterații, crescând numărul de densități Gaussiene per senone până în punctul în care am observat doi pași consecutivi fără o îmbunătățire relativă semnificativă. Astfel, am determinat

configurația optimă a sistemului independent de vorbitor ca fiind cu 100 de senone și 64 de densități Gaussiene per senone.

Pentru evaluarea sistemului în condițiile optime determinate am efectuat o testare similară. Diferența constă în faptul că pentru evaluare au fost folosite fișiere cu aceeași transcriere textuală de referință, dar de la alți vorbitori în număr de 28. De la fiecare vorbitor am utilizat un număr de 20 de clipuri audio, deci avem un total de 560 de clipuri audio. Rezultatele sunt trecute în Tabelul 3.4.

Tabel 3.4 Rezultatele sistemului independent de vorbitor

WER [%]	Număr senone
Densități Gaussiene / senone	100
1	17,6
2	9,2
4	7,4
8	4,7
16	4,9
32	4,7
64	4,3
128	4,5
256	4,5

În continuare analizăm în detaliu performanțele pentru configurația optimă. Pentru detalierea erorilor vom face media dintre rezultatele fișierelor de la toți vorbitorii.

- WER = 4,3% din care:
 - Erori de substituție = 2,1%
 - Erori de ștergere = 1,5%
 - Erori de inserție = 0,6%
- SER = 24,1%

La analiza erorilor de substituție am observat în raportul evaluării că cea mai mare confuzie făcută de sistem este între cifrele șase și șapte. Acest lucru se poate datora faptului că cele două cuvinte au o transcriere fonetică asemănătoare, implicit pronunția este similară.

Următorul pas în analiza performanțelor este să cercetăm rezultatele obținute pentru fiecare vorbitor în parte. Pentru a evidenția mai bine fluctuațiile ratei de eroare la nivel de cuvânt am reprezentat grafic WER în funcție de vorbitor în Fig. 4.3. De aici observăm că în unele cazuri WER este mult peste medie, iar acest lucru trebuie analizat în continuare pentru ca reprezintă o sursă aparte de eroare, iar identificarea și eliminarea ei va contribui la optimizarea sistemului de recunoaștere de cifre.

Principalul vorbitor care contribuie la creșterea ratei de eroare la nivel de cuvânt are un WER de aproape 50%. Pentru descoperirea posibilelor cauze au fost analizate clipurile audio utilizate la testare. La ascultarea acestora s-a constatat că vorbitorul rostea o singură cifră în întregul fișier audio în loc de 12 câte sunt în fiecare transcriere textuală de referință.

După eliminarea clipurilor audio care introduceau erori foarte mari (cu WER de peste 5%), media ratei de eroare la nivel de cuvânt a scăzut aproape de valoarea mediană, și anume 1,7%.

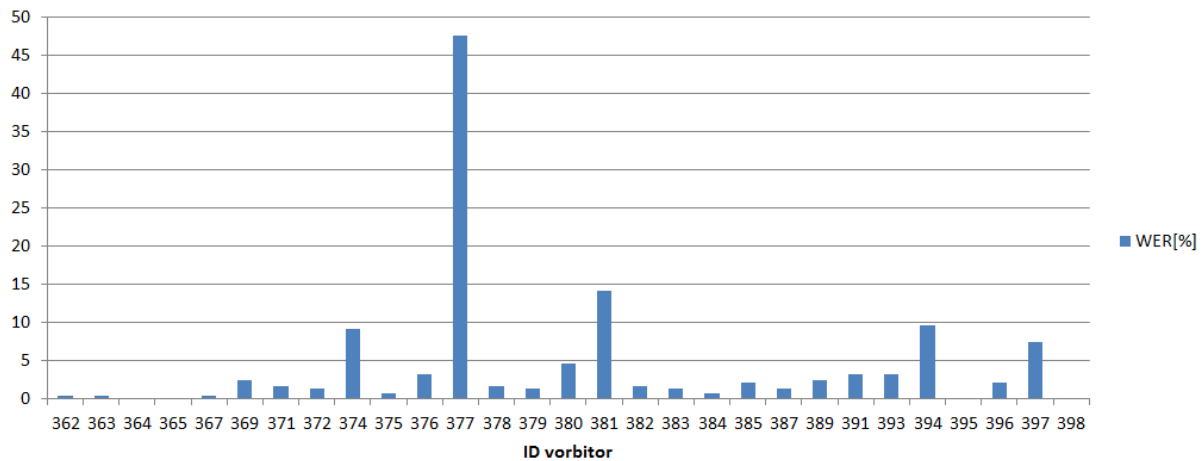


Fig. 3.4 WER al vorbitorilor pentru sistemul independent de vorbitor

3.4 INTERPRETAREA REZULTATELOR

În urma procesului de optimizare am îmbunătățit sistemele până la un nivel la care poate fi folosit în diverse aplicații practice. Este de remarcat faptul că sistemul independent de vorbitor are o rată a erorii la nivel de cuvânt mai mare decât sistemul dependent de vorbitor, ceea ce era de așteptat, dar nu cu o diferență foarte mare care să împiedice utilizarea lui într-o aplicație reală. Acest aspect este unul important deoarece majoritatea aplicațiilor practice presupun ca sistemul să primească semnalul audio provenit de la mai mulți vorbitori.

Un aspect care determină performanța sistemului este volumul și diversitatea fișierelor audio. Cu cât sunt folosite mai multe clipuri audio în procesul de antrenare provenite de la cât mai mulți vorbitori care să acopere o plajă largă de pronunții și timbre vocale, cu atât sistemul va fi capabil să recunoască cu succes semnalul audio.

Din analiza clipurilor audio folosite la evaluarea sistemului independent de vorbitor, care au avut rezultate mai slabe (WER peste 5%) au putut fi identificate cauze precum:

- Rostirea rapidă, care implicit provoca o eroare de ștergere în momentul rostirii a două cuvinte fără nicio pauză;
- Rostirea fără dicție, provocând erori de substituție (în special confuzii între cifrele șase și șapte);
- Defecte de vorbire (sâsâit sau rââit);
- Calitatea slabă a semnalului și/sau RSZ (raport semnal-zgomot) scăzut.

O parte din aceste cauze pot fi eliminate în cazul folosirii la antrenarea modelului acustic a unei baze de date mai dezvoltată, iar altele țin de condițiile de înregistrare a clipurilor audio folosite la evaluare.

CAPITOLUL 4

SISTEMUL DE CONTROL PRIN COMENZI VOCALE

4.1 DESCRIEREA APLICAȚIEI ȘI SCENARIUL DE UTILIZARE

Implementarea proiectului este realizată printr-o aplicație scrisă în Python care permite utilizatorului să folosească un sistem home cinema cu ajutorul comenzilor vocale. Aplicația recunoaște un set de comenzi vocale pentru a naviga prin directoarele disponibile și pentru a controla redarea fișierelor multimedia disponibile.

Resursele necesare aplicației sunt reprezentate de următoarele:

- Două microfoane;
- Server de recunoaștere automată a vorbirii;
- Display/TV;
- Sistem audio.

Microfoanele sunt utilizate pentru achiziția semnalului audio. Unul dintre acestea este poziționat în apropierea vorbitorului, iar cel de-al doilea în apropierea sursei de zgomot (reprezentată de difuzoarele sistemului audio). Cu ajutorul celor două semnale se va prelucra semnalul printr-o filtrare adaptivă pentru a elimina o parte cât mai mare din zgomot.

Cu ajutorul *serverului de Recunoaștere Automată a Vorbirii* se transcrie textual semnalul audio care conține vorbire, mai exact comenzile date de către utilizator. După ce serverul primește semnalul audio va răspunde cu un fișier text care conține textul transcris din semnalul audio. Acesta va fi mai departe interpretat de către aplicație.

Aplicația are o interfață grafică afișată cu ajutorul unui *display* sau *TV*, iar la redarea unor fișiere de tip audio sau video va fi utilizat și un *sistem audio*.

Diagrama de funcționare

Pentru prezentarea modului de funcționare a aplicației a fost realizată o diagramă de funcționare. În cadrul acesteia sunt evidențiate principalele etape care duc la îndeplinirea funcționalității aplicației.

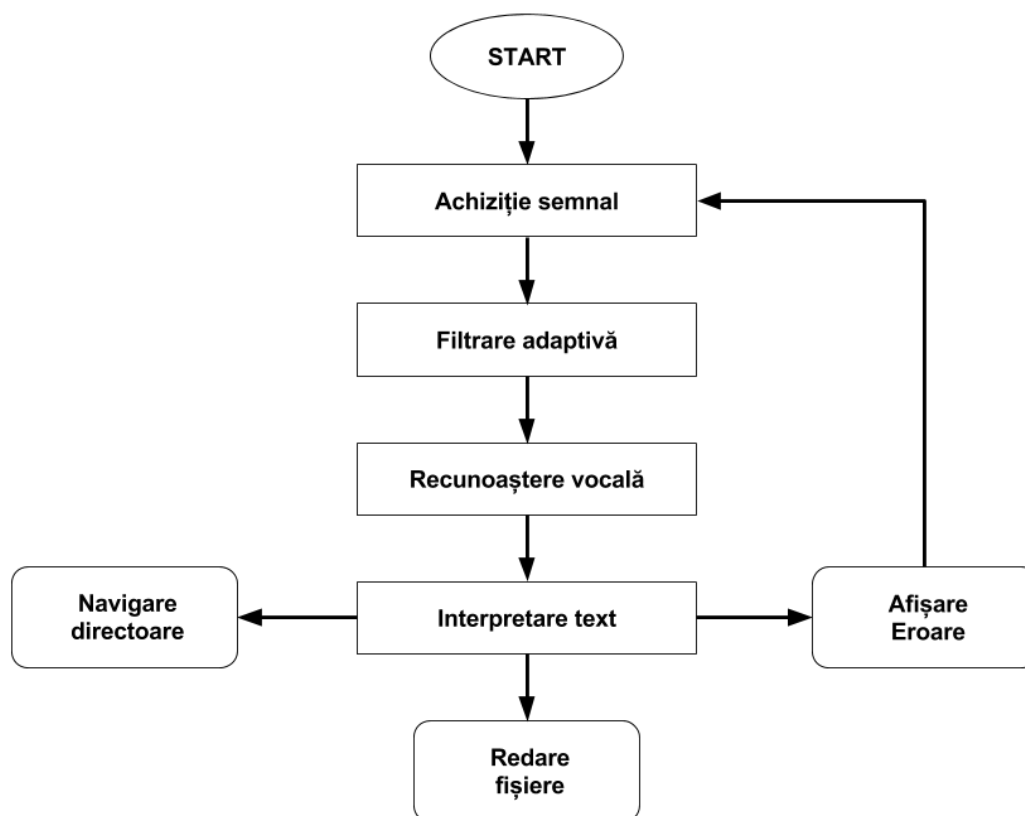


Fig. 4.1 Diagrama de funcționare a aplicației

La pornirea aplicației se face achiziția semnalului cu ajutorul a două microfoane. Acest semnal este filtrat cu ajutorul unui algoritm de filtrare adaptivă, iar apoi este trimis la sistemul de recunoaștere automată a vorbirii. Răspunsul sistemului RAV este parsat, iar comenzile sub formă text sunt interpretate. Interpretarea se face ținând cont de un set de comenzi pe care aplicația le recunoaște, iar prin comparație cu textului recepționat pot fi luate trei tipuri de decizii:

- Navigare prin directoarele disponibile
- Redarea și controlul fișierelor multimedia
- Afișarea unei erori în cazul primirii unei comenzi necunoscute sau imposibil de efectuat (de exemplu: deschiderea unui fișier inexistent)

Setul de comenzi

Aplicația permite primirea unui anumit set de comenzi pentru care va răspunde cu o acțiune în funcție de starea în care se află. Aceste comenzi sunt împărțite în două categorii, iar aceste categorii conțin un număr de comenzi după cum urmează:

- Comenzi de navigare prin directoare:
 - „deschide directorul <index director>”, unde <index director> reprezintă indicele atribuit directorului, așa cum este vizualizat în interfața grafică;
 - „deschide fișierul <index fișier>”, unde <index fișier> reprezintă indicele atribuit fișierului, precum în cazul anterior pentru directoare;
 - „înapoi”, comandă care revine la directorul anterior;
- Comenzi de control al redării fișierelor multimedia:
 - „următoarea” / „următorul”, comandă care deschide următorul fișier din directorul curent;
 - „înapoi”, comandă care deschide fișierul anterior din directorul curent;
 - „[dă] mai tare / încet”, comandă care modifică volumul audio;
 - „pauză”, comandă care pune pauză redării fișierelor audio sau video;
 - „continuă”, comandă care reia redarea fișierelor audio sau video.

Aceste comenzi au fost utilizate în momentul creării gramaticii utilizate la antrenarea sistemului de recunoaștere automată a vorbirii. Având o vocabular redus am putut folosi acest o gramatică cu stări finite care conduce la performanțe mai bune. În Fig. 4.2 este prezentat modul de scriere a gramaticii cu stări finite utilizată pentru această aplicație.

```
grammar functii;
//-----

<navigare> = (<deschidere_director> | <deschidere_fisier> | inapoi);
<deschidere_director> = deschide directorul (<numar_cifra> | <numar_ac>);
<deschidere_fisier> = deschide fișierul (<numar_cifra> | <numar_ac>);

//-----

<control> = (inchide | [dă] mai (tare|încet) | (următoarea | următorul) | inapoi | pauză | continuă);

//-----

<numar_cifra> = zero | unu | una | doi | două | trei | patru | cinci | șase | șapte | opt | nouă;
<numar_ac> = zece | unsprezece | doisprezece | douăsprezece | treisprezece | paisprezece | cincisprezece
| șaisprezece | șaptesprezece | optsprezece | nouăsprezece | douăzeci | douăzeci_și_unu |
douăzeci_și_doi | douăzeci_și_trei | douăzeci_și_patru | douăzeci_și_cinci | douăzeci_și_șase |
douăzeci_și_șapte | douăzeci_și_opt | douăzeci_și_nouă | treizeci | treizeci_și_unu | treizeci_și_doi |
treizeci_și_trei | treizeci_și_patru | treizeci_și_cinci | treizeci_și_șase | treizeci_și_șapte |
treizeci_și_opt | treizeci_și_nouă | patruzeci;

//-----

public <comandaCasa> = <navigare> | <control>;
```

Fig. 4.2 Gramatica cu stări finite utilizată în sistemul RAV

Pe lângă comenzile enumerate mai sus, în gramatica folosită la antrenarea sistemului RAV au fost introduse și un set de cuvinte cheie care reprezintă numerele de la 0 la 40. Aceste cuvinte apar în setul de comenzi pe pozițiile <index director> și <index fișier>.

Managementul erorilor

Având în vedere faptul că aplicația are un set de comenzi pe care le recunoaște, aceasta trebuie să aibă o acțiune cu care să răspundă pentru utilizator și în situația în care comanda nu face parte din setul predefinit. Pentru astfel de situații aplicația afișează un mesaj de avertizare în cadrul interfeței grafice care conține indicații legate de sursa erorii, iar apoi așteaptă o altă comandă de la utilizator.

Un alt tip de erori sunt cele generate de comenzi recunoscute de sistem, dar nu pot fi efectuate din cauza stării actuale a aplicației, precum: aplicația este în curs de redare a unui fișier, aplicația este în etapa de navigare prin directoare, etc. Astfel am identificat următoarele scenarii în care pot apărea erori, în care utilizatorii vor fi notificați cu privire la tipul erorii:

- Sistemul este în orice stare (de navigare sau redare a fișierelor), iar utilizatorul dă o comandă de accesare a unui director sau fișier inexistent;
- Sistemul este în etapa de navigare în directorul de bază, iar utilizatorul dă o comandă de revenire în directorul anterior;

4.2 SCHEMA BLOC FUNCȚIONALĂ

Întreaga aplicație poate fi reprezentată la nivel schematic pentru a pune în evidență resursele externe necesare, dar și modulele din componența aplicației împreună cu modalitatea prin care acestea comunică. În Fig. 4.3 este figurată schema bloc funcțională a sistemului multimedia controlat prin comenzi vocale.

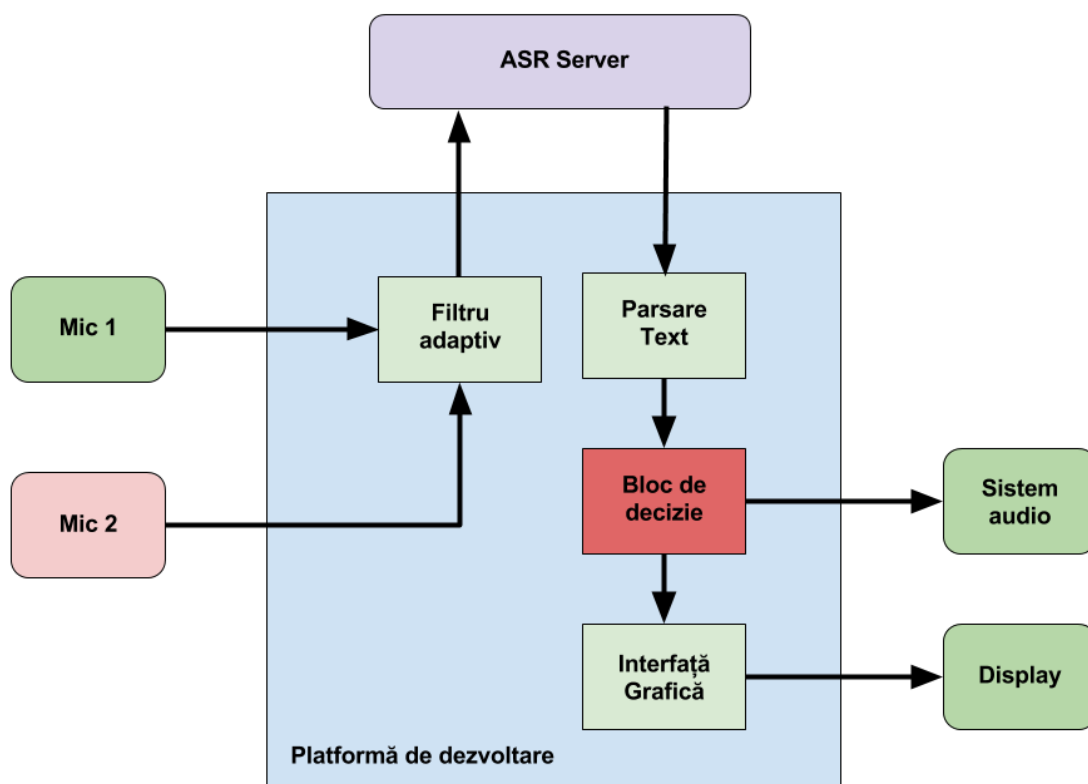


Fig. 4.3 Schema bloc a aplicației

Sistemul este compus dintr-o serie de componente externe hardware care comunică cu aplicația, dar și de un server care asigură transcrierea textuală a semnalului audio care conține vorbirea. Aplicația la rândul ei este formată dintr-o serie de module.

Platforma de dezvoltare poate fi reprezentată de un laptop sau o placă de dezvoltare (precum Raspberry Pi) datorită faptului că aplicația a fost scrisă în limbajul de programare Python, limbaj de programare independent de platformă (poate să ruleze pe sisteme de operare precum Linux, Windows, Mac OS), fiind necesară doar o serie de configurări care să permită accesul la resursele hardware.

Resursele hardware sunt următoarele:

- Un *microfon* utilizat pentru achiziția semnalului vocal, poziționat în apropierea utilizatorului;
- Un *microfon* utilizat pentru achiziția zgomotului. Zgomotul este reprezentat de sunetul produs de sistemul audio, de aceea microfonul este poziționat în apropierea difuzoarelor;
- *Sistemul audio* utilizat pentru redarea conținutului media (audio și video);
- *Display-ul* (poate fi atât display-ul laptopului, cât și unul extern precum un televizor sau monitor) utilizat pentru afișarea interfeței grafice a aplicației.

Serverul ASR (Automatic Speech Recognition) va comunica cu aplicația pentru a asigura decodarea semnalului audio care conține vorbire. Comunicarea cu acesta se face prin două socket-uri: unul pentru a transfera fișiere text, iar cel de-al doilea pentru a transfera semnalul audio. Fișiere text sunt în formatul XML, motiv pentru care toate informațiile legate de transcriere, dar și transcrierea textuală efectivă, sunt parsate pentru a putea fi mai departe prelucrate. Din același motiv cererile și informațiile pe care le trimitem sunt codate în formatul XML recunoscut de aplicația care rulează pe server.

Modulele care compun aplicația sunt următoarele:

- *Filtru adaptiv* – acest modul implementează un algoritm de filtrare adaptivă numit NLMS (Normalised Least Mean Squares) care pe baza celor două semnale audio achiziționate de la microfoane îmbunătățește RSZ-ul (raportul semnal-zgomot) semnalului care va fi trimis către server pentru decodare;
- *Parsare text* – acest modul va primi răspunsul serverului ASR și va decoda din acesta transcrierea textuală a semnalului audio trimis serverului;
- *Bloc de decizie* – în cadrul acestui modul se va analiza textul obținut în urma parsării. Textul este analizat pe baza setului de comenzi disponibil și va lua decizii de navigare, control al conținutului multimedia sau generare de erori (conform diagramei de funcționare din Fig. 4.1 Diagrama de funcționare a aplicației);
- *Interfața grafică* – acest modul asigură comunicarea cu utilizatorul prin afișarea stării în care se află aplicația împreună cu conținutul multimedia pe care utilizatorul îl comandă. În toate cazurile pe care le poate genera blocul de decizie este necesară o actualizare a interfeței grafice. Decizia luată de blocul de decizie este transmisă sub forma unei variabile interfeței grafice, iar aceasta va fi actualizată conform comenzii.

4.3 IMPLEMENTAREA FILTRĂRII ADAPTIVE

Filtrarea adaptivă este o tehnică ce a fost implementată cu ajutorul mai multor algoritmi, făcându-se continuu încercări de îmbunătățire a performanțelor acestora. Introdus încă din anul 1960, algoritmul LMS (Least Mean Squares), este cel mai cunoscut algoritm de anulare adaptivă a zgomotului datorită performanțelor sale care l-au adus să devină algoritmul de referință în domeniul filtrării adaptive. Acesta excelează atât prin rezultate bune, dar și printr-un volum mic de calcule necesar. [7] În Fig. 4.4 avem reprezentată schema de bază a algoritmului LMS, cu următoarele semnale:

- $x[n]$ – semnalul de intrare;
- $y[n]$ – semnalul de ieșire;
- $d[n]$ – semnalul de referință;
- $\varepsilon[n] = d[n] - y[n]$

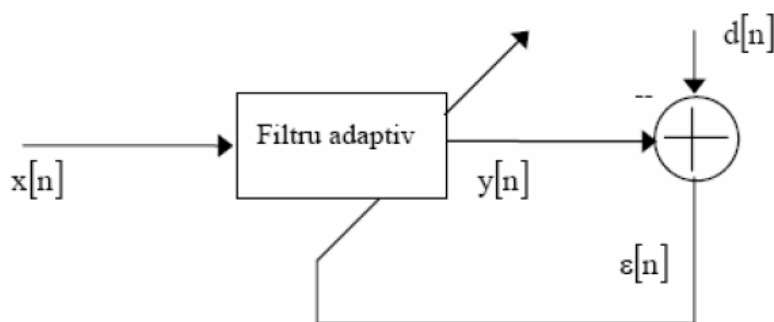


Fig. 4.4 Schema bloc a filtrului adaptiv [7]

Cu ajutorul semnalului $\varepsilon[n]$ determinăm deosebirea dintre semnalele $d[n]$ și $y[n]$ calculând eroarea medie pătratică $E\{\varepsilon^2[n]\}$, unde E este operatorul de mediere statistică. Astfel filtru își poate adapta coeficienții pe baza acestei erori pătratice medii.

Cu toate acestea, algoritmul LMS prezintă un dezavantaj prin faptul că este sensibil la variațiile mari ale intrării, iar acest lucru poate genera o instabilitate a algoritmului indiferent de constanta de adaptare aleasă, μ . O soluție pentru această problemă este implementată într-o variantă adaptată a algoritmului LMS, numită NLMS (Normalised Least Mean Squares), prin normalizare cu puterea semnalului de intrare. [8]

Coeficienții filtrului adaptiv au inițial valoarea zero, iar pe măsură ce sunt primite eșantioane ale semnalelor, valorile acestora vor fi actualizate conform Ecuației 5.1.

$$h(n+1) = h(n) + \frac{\mu e^*(n)x(n)}{x^H(n)x(n)} \quad (5.1)$$

Parametrul cel mai important al algoritmului este μ , reprezentând pasul algoritmului. Acesta controlează stabilitatea și convergența algoritmului. Pentru a obține cele mai bune rezultate cu acest tip de filtru am efectuat o serie de simulări în scopul optimizării performanțelor filtrului. Trebuie ținut cont că pentru valori foarte mici ale lui μ vom avea un timp de convergență al coeficienților ridicat (lucru nedorit), iar pentru o valoare foarte mare a acestuia vom crea un filtru cu o stabilitate foarte scăzută, implicit performanțe ale filtrării scăzute [13].

Alături de constanta de adaptare (sau pasul filtrului) vom testa și mai multe valori pentru ordinul filtrului, L . Pe parcursul simulărilor vom încerca să găsim un echilibru între timpul de convergență și calitatea filtrării.

Înregistrarea semnalului vocal s-a înregistrat într-o cameră de bloc de aproximativ 12 m² având un zgomot foarte mic de fundal. Sensorul folosit a fost microfonul integrat dintr-un laptop, iar software-ul utilizat a fost MATLAB R2015a. Semnalul a fost eșantionat la o frecvență de 48kHz.

Semnalul înregistrat, numit în continuare *semnalUtil*, a fost mixat cu un semnal aleator (*zgomotSursa*) trecut printr-un filtru trece-jos, numit *zgomotCaptat*. Semnalul *semnalUtil* împreună cu *zgomotSursa* au fost utilizate ca intrări în sistemul de filtrare adaptivă care la ieșire returnează un semnal $y[n]$ reprezentând o aproximare a semnalului *zgomotCaptat*. Acest semnal a fost scăzut din semnalul *semnalUtil*, pentru a obține semnalul filtrat, numit în continuare *semnalFiltrat*.

Evaluarea performanțelor filtrului adaptiv

Pentru a determina calitatea filtrării am utilizat o metodă grafică prin care am suprapus eșantioanele din același interval de timp al semnalelor *semnalUtil* și *semnalFiltrat*. Astfel în

Fig. 4.5 putem vizualiza diferențele dintre cele două semnale de pe un eșantion de timp scurt (10ms).

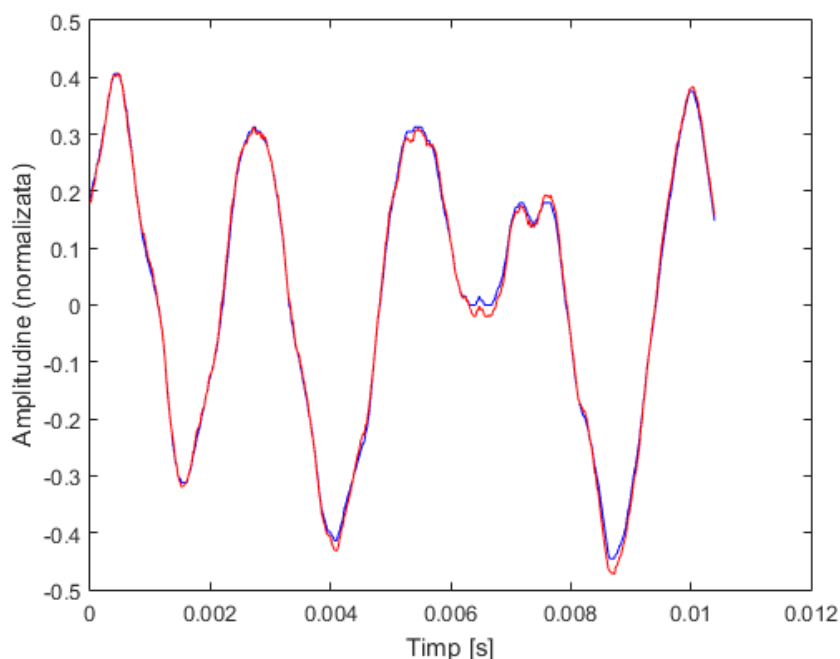


Fig. 4.5 Comparație dintre semnalul util și semnalul filtrat

Un indicator foarte important al filtrării adaptive cu ajutorul algoritmului NLMS este timpul de convergență. Pentru a face acest lucru vom măsura timpul după care coeficienții filtrului variază sub un anumit prag. Vom evidenția grafic această variație a coeficienților în Fig. 4.6.

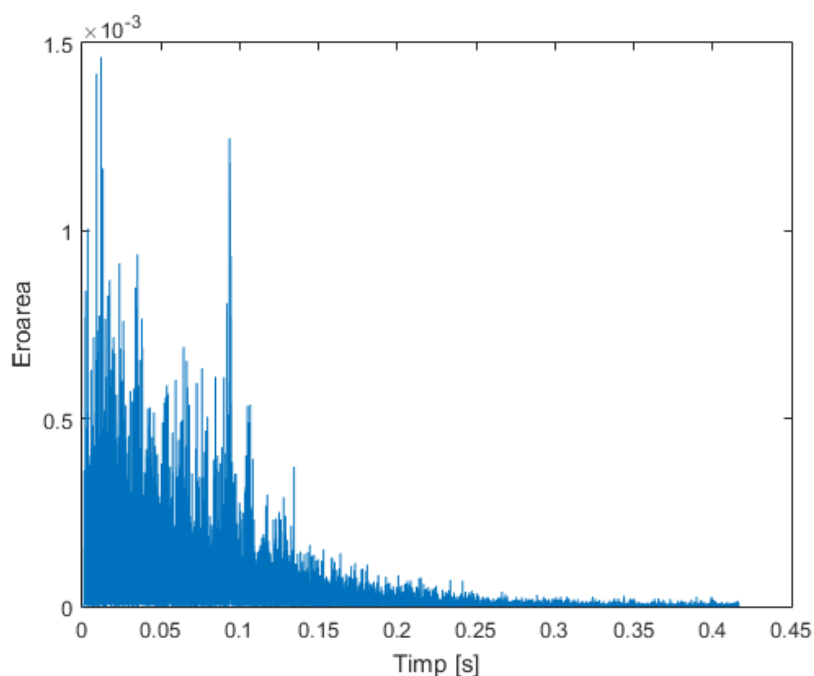


Fig. 4.6 Variația în timp a coeficienților filtrului adaptiv

Valorile inițiale utilizate pentru parametri de bază ai filtrului sunt:

- Coeficientul de adaptare $\mu=0,01$;
- Ordinul filtrului $L=60$.

Din grafic observăm că timpul de convergență este de aproximativ 0,2 secunde în condițiile în care considerăm un prag pentru axa de variație în jurul valorii de 0,1.

Optimizarea performanțelor filtrului adaptiv

Pentru evaluarea calității filtrării adaptive am creat un algoritm care să calculeze diferența dintre suprafața creată de axa timpului cu forma de undă a semnalului *semnalUtil*, respectiv cu cea a semnalului *semnalFiltrat*. Prin această metodă obținem o valoare numerică asociată gradului de abatere a semnalului obținut în urma filtrării în raport cu cel original.

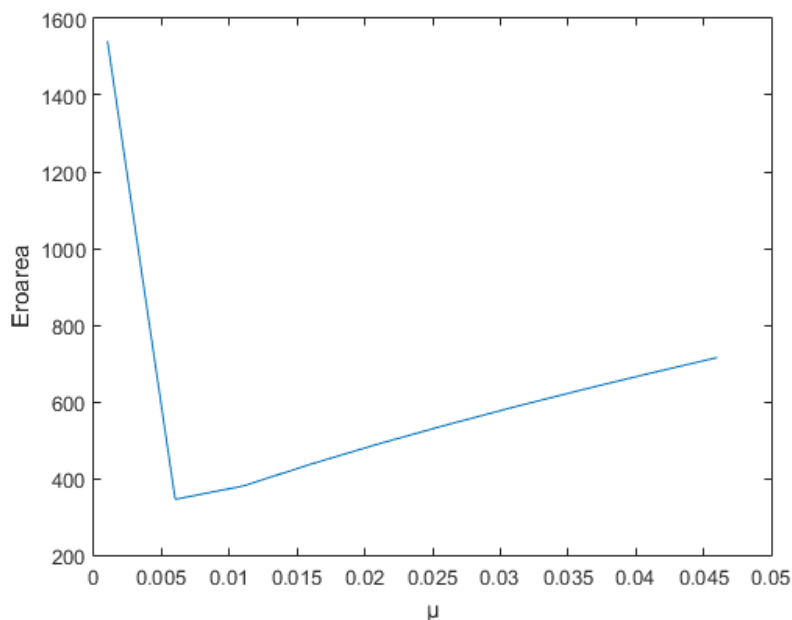


Fig. 4.7 Eroarea în funcție de constanta de adaptare

În Fig. 4.7 este evidențiată variația erorii calculată după metoda descrisă anterior pentru mai multe valori ale lui μ , iar ordinul filtrului menținut constant la valoarea de 60. Pe baza acestui grafic concluzionăm că valoarea optimă pentru constanta de adaptare este de 0.006.

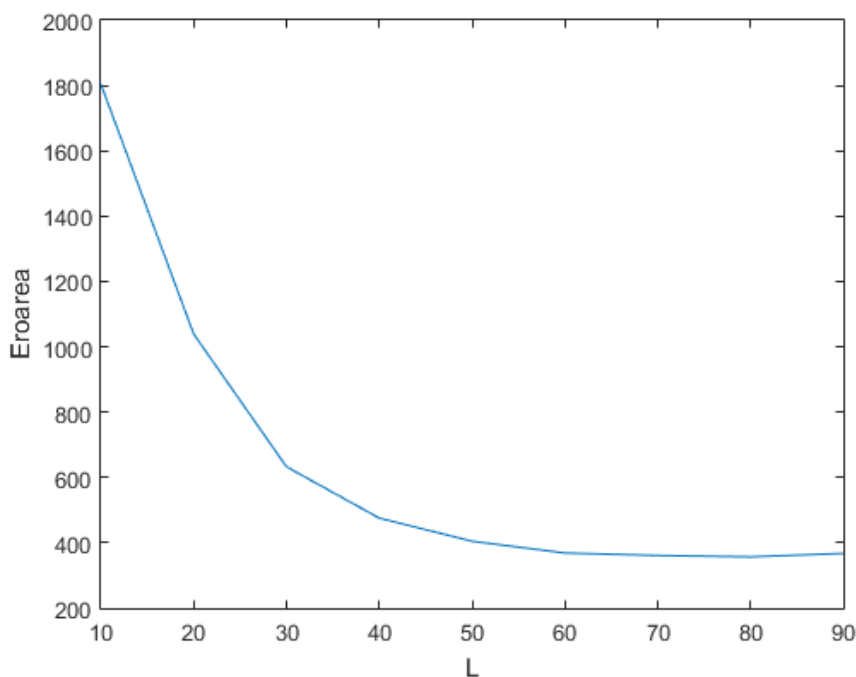


Fig. 4.8 Eroarea în funcție de ordinul filtrului

În Fig. 4.8 am simulat eroarea produsă de filtrul adaptiv în funcție de ordinul filtrului, în condițiile unui pas de adaptare constant, la valoarea de 0,01. Din grafic reiese faptul că ordinul filtrului influențează puternic performanțele algoritmului NLMS, dar până la valoarea de 60, urmând ca după această valoare eroarea să rămână constantă. Ținând cont că ne dorim să folosim o putere de calcul cât mai mică, un ordin al filtrului de 60 este valoarea optimă.

În continuare dorim să măsurăm performanțele filtrului adaptiv și după un al doilea criteriu foarte important, și anume timpul de convergență. Pentru aceasta, vom folosi un algoritm care să detecteze automat momentul în care variația coeficienților filtrului este mai mică decât o valoare de prag setată la 10% din cea mai mare valoare a variației.

Pentru această simulare observăm din Fig. 4.9 valori foarte mari ale timpului de convergență sub valoarea de 0,03 a constantei de adaptare. Trebuie ținut cont că valori mari ale constantei de adaptare pot provoca o instabilitate mare a algoritmului NLMS, iar în condițiile în care peste valoare de 0,1 nu mai avem mari îmbunătățiri ale timpului de convergență o putem considera o valoare optimă.

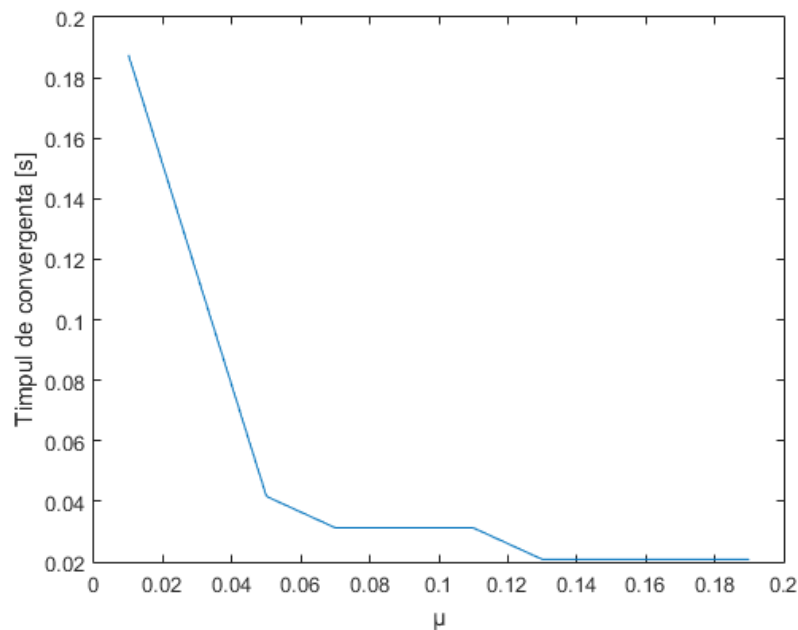


Fig. 4.9 Timpul de convergență în funcție de constanta de adaptare

De asemenea, dorim să identificăm o valoare optimă și pentru ordinul filtrului care să asigure un timp de convergență cât mai bun. În Fig. 4.10 observăm că avem rezultate bune ale timpului de convergență la valori mai mici ale ordinului filtrului, dar în același timp aceste valori mici aduc instabilitate, motiv pentru care vom prefera să lucrăm cu valori ale ordinului filtrului din zona în care este aproximativ constant timpul de convergență.

Simulările efectuate au evidențiat valori optime ale constantei de adaptare și a ordinului filtrului, dar fiind variate pe rând. Acest lucru nu este suficient, motiv pentru care pentru a identifica valorile optime trebuie să facem experimente în care să variem ambii parametrii.

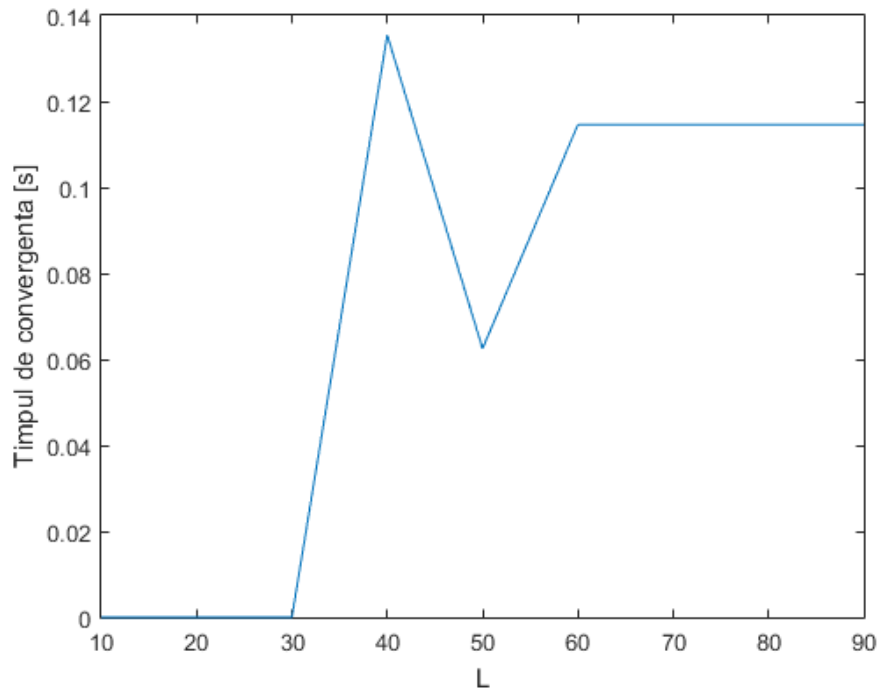


Fig. 4.10 Timpul de convergență în funcție de ordinul filtrului

După o simulare de variație simultană a celor doi parametri am reprezentat rezultatele sub forma unui grafic 3D (Fig. 4.11). Acesta apare sub forma unei suprafețe curbe, iar punctul cel mai de jos al acesteia ne indică coordonatele corespunzătoare valorilor optime. Acestea sunt:

- Coeficientul de adaptare $\mu=0,005$;
- Ordinul filtrului $L=60$.

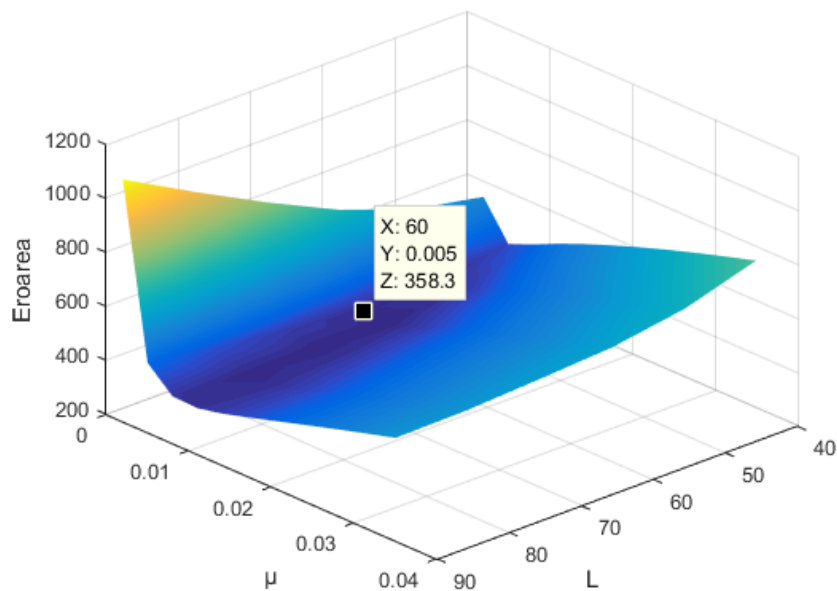


Fig. 4.11 Eroarea în funcție de constanta de adaptare și ordinul filtrului

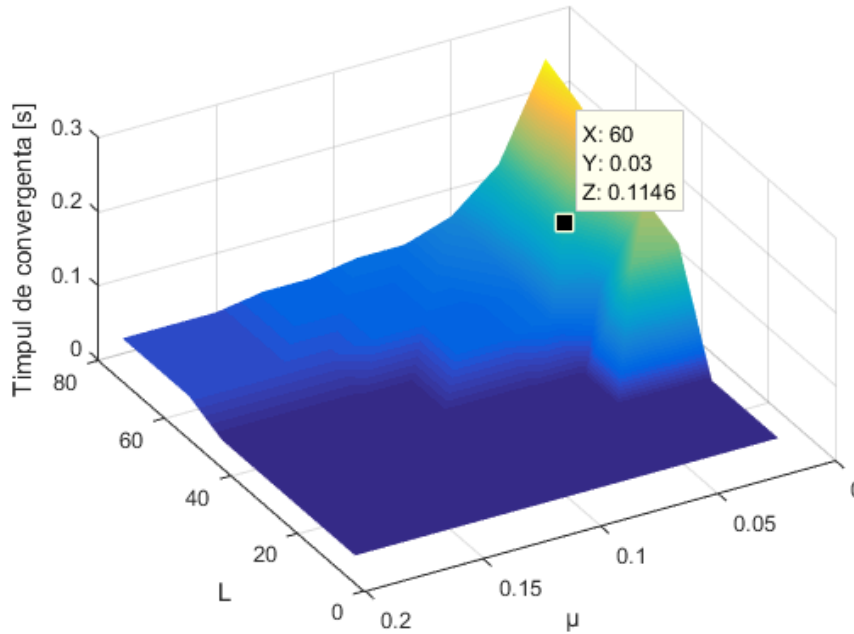


Fig. 4.12 Timpul de convergență în funcție de constanta de adaptare și ordinul filtrului

Un experiment similar a fost efectuat și pentru a determina dependența timpului de convergență (Fig. 4.12), iar valorile optime au fost alese ținând cont de eroare filtrării adaptive din Fig. 4.11.

Se observă că valorile optime alese din ultimele două simulări nu sunt apropiate pentru pasul de adaptare, motiv pentru care trebuie să identificăm o valoare care să nu afecteze prea mult filtrarea adaptivă după niciunul din cei doi indicatori. Astfel, suntem nevoiți să facem un compromis și să alegem următoarele două valori ca fiind echilibrul cel mai bun pentru performanțele filtrării:

- Coeficientul de adaptare $\mu=0,03$;
- Ordinul filtrului $L=60$.

Valoarea de 0,03 pentru coeficientul de adaptare a fost aleasă deoarece nu introduce o eroare mult mai mare față de valori mai mici, însă creșterea acesteia ar fi produs un timp de convergență mult prea mare pentru ca filtrul adaptiv să poată fi folosit într-o aplicație practică.

4.4 INTERFAȚA GRAFICĂ

Un dispozitiv Home Cinema presupune prezența unui terminal de afișare care poate fi reprezentat de un display, monitor sau TV. Acest dispozitiv este folosit pentru a reda fișierele grafice (imagini, video). Cu toate acestea, înainte ca fișierele să fie deschise este necesar ca utilizatorul să cunoască care sunt opțiunile pe care le are la dispoziție în vederea comandării sistemului. Pentru aceasta a fost implementată o interfață grafică care să afișeze în mod constant starea sistemului. Prin asta înțelegem vizualizarea directorului curent, fișierelor și directoarelor disponibile pentru accesare, dar mai ales a fișierelor în curs de redare.

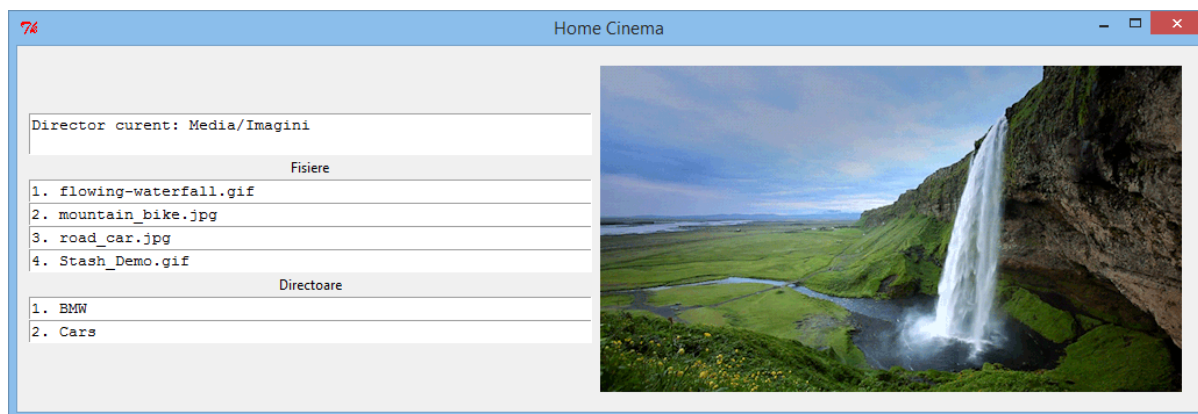


Fig. 4.13 Interfața grafică – redare imagine

În Fig. 4.13 este o captură a interfeței grafice în momentul afișării unei imagini. În panoul din stânga este afișată navigarea prin directoarele disponibile. Această secțiune cuprinde următoarele:

- Directorul curent, cu rădăcina la directorul de bază, denumit „Media”;
- Lista cu fișierele aflate în directorul curent și disponibile pentru redare;
- Lista cu directoarele disponibile pentru accesare.

În partea dreaptă se află secțiunea de redare a fișierelor grafice (video și imagini).

Această interfață grafică este creată dinamic pe baza comenzilor primite de modul. Librăria utilizată la crearea acestui GUI (Graphical User Interface) se numește *Tkinter* și este una dintre cele mai utilizate librării GUI pentru Python.

Indexarea fișierelor este făcută cu ajutorul unei funcții de citire precum o citire cu crawlere de la nivelul directorului de bază, numit „Media”. Listele de fișiere și directoare sunt generate de funcția din Fig. 4.14. Această funcție indexează toate fișierele și directoarele din directorul curent.

```
def getFiles(dirName):
    fileList = []
    dirList = []
    dirCount = 0
    fileCount = 0

    for item in os.listdir(dirName):
        if os.path.isfile(os.path.join(dirName, item)):
            fileList.append(item)
        else:
            dirList.append(item)
    return dirList, fileList
```

Fig. 4.14 Funcția de indexare a fișierelor și directoarelor

Funcția care generează panoul de navigare este prezentată în Fig. 4.15. În această funcție se observă modul în care au fost create elementele componente ale acestui panou. Calea directorului curent și listarea fișierelor s-a făcut cu ajutorul widget-ului de tip *Text*, iar etichetele „Fișiere” și „Directoare” cu widget-ul de tip *Label*. Această funcție primește ca parametri:

- directorul curent ca o variabilă de tip *șir de caractere*;
- lista de fișiere din directorul curent ca o variabilă de tip *vector*;
- lista de directoare din directorul curent ca o variabilă de tip *vector*.

```

def listare(dirCr, fileList, dirList):
    titleText = Text(frameList, width=60, height=2)
    titleText.insert(INSERT, 'Director curent: ' + dirCr)
    titleText.pack()

    label_files = Label(frameList, text = 'Fisiere')
    label_files.pack()

    counter = 1
    print '\n File list: '
    for f in fileList:
        print '\n', f
        text = Text(frameList, width=60, height=1)
        text.insert(INSERT, str(counter) + '. ' + f)
        text.pack()
        counter += 1

    label_dirs = Label(frameList, text = 'Directoare')
    label_dirs.pack()

    counter = 1
    print '\n Dir list: '
    for d in dirList:
        print '\n', d
        text = Text(frameList, width=60, height=1)
        text.insert(INSERT, str(counter) + '. ' + d)
        text.pack()
        counter += 1

    frameMedia.pack(side=RIGHT, padx=5, pady=5)

```

Fig. 4.15 Funcția de generare a panoului de navigare

CONCLUZII

Concluzii generale

Principalul obiectiv al acestei lucrări a fost de a crea un sistem de redare a conținutului multimedia care să poate fi controlat prin comenzi vocale, eliminându-se necesitatea unui dispozitiv de comandă de la distanță, precum o telecomandă. Această facilitate îi conferă utilizatorului libertatea de a folosi mâinile pentru alte activități, dar și de a își folosi privirea pentru altă acțiune în condițiile în care pentru unele comenzi recunoscute de aplicație nu este nevoie ca utilizatorul să verifice starea sistemului afișată cu ajutorul interfeței grafice.

Pentru a atinge acest obiectiv a trebuit să se aibă în vedere toate scenariile care pot apărea în timpul utilizării unui astfel de sistem. Astfel a fost identificată problema zgomotului suprapus peste semnalul util transmis prin voce de către utilizator. Acest zgomot este puternic în condițiile în care sistemul audio redă un fișier audio sau video, iar acel semnal zgomot are o amplitudine mare în momentul achiziției vocii. Soluția propusă pentru această problemă a fost descrisă în subcapitolul Implementarea filtrării adaptive, cu care s-a putut reduce nivelul de zgomot în vederea unei recunoașteri automate a vorbirii mai precisă.

În această lucrare au fost prezentate etapele dezvoltării aplicației. Prima etapă a fost familiarizarea cu sistemul de recunoaștere automată a vorbirii dezvoltat în prealabil în cadrul laboratorului Speed. Acest lucru a fost făcut prin dezvoltarea unui sistem de recunoaștere de cifre prezentat în Sistem de recunoaștere de cifre. Aplicația a fost construită dezvoltând o serie de module esențiale funcționării acesteia, module prezentate schematic în Fig. 4.3 Schema bloc a aplicației.

Contribuții personale

Contribuțiile autorului acestei lucrări sunt evidențiate în Sistem de recunoaștere de cifre și Sistemul de control prin comenzi vocale. În prima etapă a proiectului a fost construit un sistem de recunoaștere automată a vorbirii cu ajutorul unor modele de limbă și acustice dezvoltate deja în cadrul laboratorului Speed, dar cu contribuții personale la dezvoltarea modelului acustic. Acest sistem de RAV a fost antrenat și optimizat, proces prin care au fost înțelese conceptele și tehnicile utilizate pentru decodarea fișierelor audio, prezentate la nivel teoretic în Recunoașterea automată a vorbirii și filtrarea adaptivă.

După crearea unui sistem RAV cu ajutorul unei gramatici dedicate setului de comenzi al aplicației a fost implementat filtrul adaptiv prin algoritmul NLMS, algoritm preluat din librăriile Python. Contribuțiile personale se regăsesc în crearea modului de achiziție a semnalelor audio de la microfoane și filtrarea acestora. Pentru optimizarea algoritmului s-au efectuat o serie de simulări în software-ul Matlab, prezentate în Implementarea filtrării adaptive. [13]

Pentru comunicarea cu serverul de RAV a fost creat un modul în Python dedicat protocolului de comunicare impus de server. Acest modul presupune deschiderea conexiunii pe două socket-uri (unul pentru text, altul pentru stream audio), crearea mesajelor XML, dar și parsarea mesajelor primite ca răspuns de la server.

Un modul important în funcționarea aplicației este blocul de decizie care împreună cu interfața grafică redau utilizatorului rezultatul comenzii. Interfața grafică a fost realizată cu ajutorul librăriei Tkinter. [14]

Îmbunătățiri viitoare

Există o serie de aspecte care pot fi îmbunătățite teoretic, atât pe partea de experiență a utilizatorului, cât și referitor la extinderea funcționalităților, dar păstrând funcționalitatea de bază a aplicației.

Achiziția semnalului s-a făcut cu ajutorul unui microfon extern din componența unui headset. Acest lucru implică ca utilizatorul să se afle la o distanță mică de microfon, neavând libertatea totală de mișcare în spațiul în care se află. Acest lucru ar putea fi rezolvat prin poziționarea unui microfon special pentru achiziția semnalului la distanță într-o poziție centrală a camerei. Trebuie totuși ținut cont că o astfel de poziționare implică achiziția unui zgomot mai puternic, iar acest lucru ne duce la o altă posibilă îmbunătățire, și anume o filtrare adaptivă mai performantă.

O îmbunătățire a performanțelor filtrării adaptive poate fi obținută prin testarea în condițiile de utilizare normală a mai multor implementări de filtre adaptive, altele în afară de algoritmul NLMS utilizat în acest proiect.

Aplicația pune la dispoziția utilizatorului o serie de fișiere organizate în directoare aflate într-un director de bază la o locație predefinită. Pentru extinderea mai ușoară a fișierelor disponibile se poate implementa accesarea unei memorii de stocare externe, precum un stick USB.

Accesarea fișierelor indexate de aplicație în interfața grafică se face pe baza indexului din fața denumirii fișierului, respectiv directorului. A fost preferat acest mod de accesare datorită faptului că un vocabular redus asigură o rată de eroare mai mică. Însă accesarea ar putea fi făcută și cu denumirea efectivă a fișierului, iar acest lucru ar putea reprezenta o îmbunătățire.

BIBLIOGRAFIE

- [1] Speech Recognition, https://en.wikipedia.org/wiki/Speech_recognition, accesat la data: 15.6.2016
- [2] Radu Ana Maria, „Sistem Complet de Automatizare”, Proiect Diplomă, Universitatea Politehnica din București, România, 2015
- [3] Burileanu, D., Contribuții la sinteza vorbirii din text pentru limba română. Teză doctorat, 1999
- [4] Cucu Horia, „Proiect de cercetare-dezvoltare în Tehnologia Vorbirii”, Îndrumar de proiect
- [5] Cucu Horia, „Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian”, Teză de doctorat, Universitatea Politehnica din București, România, 2011
- [6] Șandru Elena Diana, „Automation System based on Microcontrollers”, Proiect Diplomă, Universitatea Politehnica din București, România, 2014
- [7] Bucos Constantin Marina, Betej Bogdan Ilie, Cater Gheorghe, „Anularea adaptivă a zgomotului”, Universitatea „Politehnica” Timișoara, Timișoara, Ianuarie 2004
- [8] Least mean squares filter, https://en.wikipedia.org/wiki/Least_mean_squares_filter, accesat la data: 18.6.2016
- [9] Introduction in Matlab, <http://www.math.mtu.edu/~msgocken/intro/node2.html>, accesat la data: 10.6.2016
- [10] The Python Programming Language, [\[link\]](#), accesat la data: 10.6.2016
- [11] Python vs Matlab, http://www.pyzo.org/python_vs_matlab.html, accesat la data: 11.6.2016
- [12] XML (Extensible Markup Language), <http://searchsoa.techtarget.com/definition/XML>, accesat la data: 11.6.2016
- [13] Dragoș Burileanu, Note de curs „Tehnici Avansate de Prelucrare Digitală a Semnalelor”, Universitatea Politehnica din București, România
- [14] An Introduction to Tkinter, <http://effbot.org/tkinterbook>, accesat la data: 17.6.2016

ANEXA 1

Achiziție și filtrare semnal

```
import pyaudio
import time
import wave
import numpy as np
import adaptfilt as adf
from scipy.io.wavfile import write

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
RECORD_SECONDS = 3
WAVE_OUTPUT_FILENAME = "output.wav"
WIDTH = 2

p = pyaudio.PyAudio()

frames = []
semnalCaptat = []
zgomot = []

frames_semnal = []
frames_zgomot = []

def callback_semnal(in_data, frame_count, time_info, status):
    frames_semnal.append(in_data)
    return (in_data, pyaudio.paContinue)

def callback_zgomot(in_data, frame_count, time_info, status):
    frames_zgomot.append(in_data)
    return (in_data, pyaudio.paContinue)

stream_semnal = p.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        # output=True,
                        input_device_index=1,
                        stream_callback=callback_semnal)

stream_zgomot = p.open(format=FORMAT,
                       channels=CHANNELS,
                       rate=RATE,
```

```

        input=True,
        input_device_index=2,
        stream_callback=callback_zgomot)

stream_semnal.start_stream()
stream_zgomot.start_stream()

for i in range (0, 30):
    time.sleep(0.1)

stream_semnal.stop_stream()
stream_zgomot.stop_stream()

stream_semnal.close()
stream_zgomot.close()

p.terminate()

semnalCaptatData = np.fromstring(b''.join(frames_semnal), 'Int16');
zgomotData = np.fromstring(b''.join(frames_zgomot), 'Int16');

semnalCaptat = np.array(semnalCaptatData, dtype='int64')
zgomot = np.array(zgomotData, dtype='int64')

# Filtering
M = 61 # No. of taps to estimate
mu = 0.0008 # Step size
beta = 0.08

# y, e, coeficienti = adf.lms(zgomot, semnalCaptat, M, mu,
returnCoeffs=True)
y, e, coeficienti = adf.nlms(zgomot, semnalCaptat, M, beta,
returnCoeffs=True)
semnalFiltrat = semnalCaptat[len(semnalCaptat)-len(y):len(semnalCaptat)] -
y

semnalFiltrat_int16 = np.int16(semnalFiltrat)
semnalCaptat_int16 = np.int16(semnalCaptat)
zgomot_int16 = np.int16(zgomot)
write('semnalFiltrat.wav', 44100, semnalFiltrat_int16)
write('semnalCaptat.wav', 44100, semnalCaptat_int16)
write('semnalZgomot.wav', 44100, zgomot_int16)

```

Interfața grafică

```

from Tkinter import *
import os

##### Indexare fisiere #####

directorCurent = 'Media/Imagini'
adresaDirectorCurent = ''

def navInainte():
    print 'Inainte'
    text = Text(frame, width=30, height=1)
    text.insert(INSERT, 'Demo data')
    text.pack()

def navInapoi():
    print 'Inapoi'

```

```

def getFiles(dirName):
    fileList = []
    dirList = []
    dirCount = 0
    fileCount = 0

    for item in os.listdir(dirName):
        if os.path.isfile(os.path.join(dirName, item)):
            fileList.append(item)
        else:
            dirList.append(item)
    return dirList, fileList

dirList, fileList = getFiles(directorCurent)

##### Creare GUI #####

master = Tk()
master.title('Home Cinema')

frame = Frame(
    master,
    padx='10',
    pady='10')
frame.pack()

frameMedia = Frame(
    frame,
    height=480,
    width=640,
    bg="blue",
    colormap="new"
)

frameList = Frame(
    frame
)
frameList.pack(side=LEFT)

def redareMedia():
    photo = PhotoImage(file="Media/Imagini/flowing-waterfall.gif")
    w = Label(frameMedia, image=photo)
    w.photo = photo
    w.pack()

def listare(dirCr, fileList, dirList):
    titleText = Text(frameList, width=60, height=2)
    titleText.insert(INSERT, 'Director curent: ' + dirCr)
    titleText.pack()

    label_files = Label(frameList, text = 'Fisiere')
    label_files.pack()

    counter = 1
    for f in fileList:
        print '\n', f
        text = Text(frameList, width=60, height=1)
        text.insert(INSERT, str(counter) + '. ' + f)
        text.pack()
        counter += 1

    label_dirs = Label(frameList, text = 'Directoare')

```

```

label_dirs.pack()

counter = 1
print '\n Dir list: '
for d in dirList:
    text = Text(frameList, width=60, height=1)
    text.insert(INSERT, str(counter) + '. ' + d)
    text.pack()
    counter += 1

frameMedia.pack(side=RIGHT, padx=5, pady=5)

def errorMessage():
    w = Message(master, text="this is a message")
    w.pack()

listare(directorCurent, fileList, dirList)
redareMedia()

master.mainloop()

```

Simulări MATLAB

```

close all
clear all

[semnalUtil,Fs] = audioread('voce.wav'); %citire semnal vocal
disp('Semnal util');
sound(semnalUtil,Fs);
T=length(semnalUtil)/Fs; % durata semnalului audio
zgomotMotor=0.1*randn(length(semnalUtil),1); %generare semnal aleator
[b,a]=butter(12,20000/(Fs/2));
zgomotFiltrat=filter(b,a,zgomotMotor); %trecerea prin filtru trece jos

semnalCabina=semnalUtil + zgomotFiltrat;
pause
disp('Semnal cabina');
sound(semnalCabina,Fs);

%% Filtrare voce

n=0:1/Fs:(T-1/Fs);
L = 60;
mu = 0.01;
alpha = 0.001;
b = zeros(1,L+1);
px = 0;
sigma = norm(semnalCabina(1:length(semnalCabina)))^2/length(semnalCabina);
% Puterea semnalului initial

[y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

%% 1. Masurare calitatii semnalului

semnalFiltrat=semnalCabina-y';

N = length(semnalUtil);
viz=500;
n1=n([1:viz]);
semnalUtil1=semnalUtil([N/2:N/2+viz-1]);
semnalFiltrat1=semnalFiltrat([N/2:N/2+499]);

```

```

pause
disp('Semnal filtrat');
sound(semnalFiltrat,Fs);
figure
plot(n1,semnalUtil1,'b',n1,semnalFiltrat1,'r');
title('Comparatie semnale');
xlabel('Timp [s]');
ylabel('Amplitudine (normalizata)');

%% 2. Masurare timp de convergenta

viz=20000;

b1=b(:,L);
for i=1:length(b1)-1
    delta(i)=abs(b1(i+1)-b1(i));
end
delta(i+1)=delta(i);
figure
plot(n([1:viz]),delta([1:viz]));
xlabel('Timp [s]');
ylabel('Eroarea');

%% 3. Optimizare filtru. Calitatea filtrarii

%%%%%%%%%%%%%% Calculul calitatii in functie de mu
%calitatea = 1/err
k=0;
mu_min=0.001;
mu_max=0.05;
pas=0.005;
for j=mu_min:pas:mu_max
    mu = j;
    b = zeros(1,L+1);
    px = 0;
    [y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

    semnalFiltrat=semnalCabina-y';
    dif=semnalFiltrat-semnalUtil;

    err=0;
    for i=10000:length(dif)
        %10000 corespunde 0.2s = timpul aprox de convergenta
        err=err+abs(dif(i)); % parametru care indica calitatea filtrarii
    end
    k=k+1;
    err_array(k)=err;
end
mu_array=mu_min:pas:mu_max;
figure
plot(mu_array,err_array);
%title('Eroarea filtrarii in functie de mu')
xlabel('μ')
ylabel('Eroarea')
%din grafic rezulta cel mai bun mu : 0.01
mu=0.01;

%%%%%%%%%%%%%% Calculul calitatii in functie de L

k=0;

```

Controlul unui sistem home cinema prin comenzi vocale

```

L_min=10;
L_max=90;
pas=10;
for j=L_min:pas:L_max
    L = j;
    b = zeros(1,L+1);
    px = 0;
    [y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

    semnalFiltrat=semnalCabina-y';
    dif=semnalFiltrat-semnalUtil;

    err=0;
    for i=10000:length(dif)
        err=err+abs(dif(i)); % parametru care indica calitatea filtrarii
    end
    k=k+1;
    err_array_L(k)=err;
end
L_array=L_min:pas:L_max;
figure
plot(L_array,err_array_L);
title('Eroarea filtrarii in functie de L');
xlabel('L');
ylabel('Eroarea');
% din grafic rezulta cel mai bun L : 60
L=60;

%% 4. Optimizare filtru. Timpul de convergenta

k=1;
mu_min=0.01;
mu_max=0.2;
pas=0.02;
for j=mu_min:pas:mu_max
    mu = j;
    b = zeros(1,L+1);
    px = 0;
    [y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

    b1=b(:,L);
    for i=1:length(b1)-1
        delta(i)=abs(b1(i+1)-b1(i));
    end
    delta(i+1)=delta(i);

    for i=0:(viz/500-1); %calculam media pe intervale de cate 500 esatioane
        suma=0;
        for p=(i*500+1):(i+2)*500
            suma=suma+delta(p);
        end
        med(i+1)=suma/500;
        if med(i+1)<(med(1)/10)
            tc_mu(k)=i*500/Fs; %timpul de convergenta in secunde
            break
        end
    end
    k=k+1;
end
mu_array=mu_min:pas:mu_max;
figure
plot(mu_array,tc_mu);

```

```

    title('Timpul de convergenta in functie de mu')
    xlabel('μ')
    ylabel('Timpul de convergenta [s]')
% din grafic rezulta cel mai bun mu : 0.15
mu=0.03;

%%%%%%%%%%%%%% Calculul timpului de convergenta in functie de L

k=1;
L_min=10;
L_max=90;
pas=10;
for j=L_min:pas:L_max
    L = j;
    b = zeros(1,L+1);
    px = 0;
    [y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

    b1=b(:,L);
    for i=1:length(b1)-1
        delta(i)=abs(b1(i+1)-b1(i));
    end
    delta(i+1)=delta(i);

    for i=0:(viz/500-1); %calculam media pe intervale de cate 500 esatioane
        suma=0;
        for p=(i*500+1):(i+2)*500
            suma=suma+delta(p);
        end
        med(i+1)=suma/500;
        if med(i+1)<(med(1)/10)
            tc_L(k)=i*500/Fs; %timpul de convergenta in secunde
            break
        end
    end
    k=k+1;
end
L_array=L_min:pas:L_max;
figure
plot(L_array,tc_L);
title('Timpul de convergenta in functie de L')
xlabel('L')
ylabel('Timpul de convergenta [s]')
% din grafic rezulta cel mai bun L este intre 10 si 30
L=60;

%% 6. Optimizarea filtrului adaptiv. Calitatea filtrarii

k=1;
mu_min=0.002;
mu_max=0.035;
pas_mu=0.003;

L_min=40;
L_max=90;
pas_L=10;

for j=mu_min:pas_mu:mu_max
    mu = j;
    w=1;
    for j=L_min:pas_L:L_max

```

```

        L = j;
        b = zeros(1,L+1);
        px = 0;
        [y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

        semnalFiltrat=semnalCabina-y';
        dif=semnalFiltrat-semnalUtil;

        err=0;
        for i=10000:length(dif)
            err=err+abs(dif(i));
            % parametru care indica calitatea filtrarii
        end
        err_matrix(k,w)=err;
        w=w+1;
    end
    k=k+1;
end
mu_array=mu_min:pas_mu:mu_max;
L_array=L_min:pas_L:L_max;
figure
surf(L_array,mu_array,err_matrix);
title('Eroarea filtrarii in functie de mu si L')
xlabel('L')
ylabel('μ')
zlabel('Eroarea')
shading interp
L=60;
mu=0.01;

```

%% 7. Optimizarea filtrului adaptiv. Timpul de convergenta

```

k=1;
mu_min=0.01;
mu_max=0.2;
pas_mu=0.02;

L_min=10;
L_max=80;
pas_L=10;

for h=mu_min:pas_mu:mu_max
    mu = h;
    w=1;
    for j=L_min:pas_L:L_max
        L = j;
        b = zeros(1,L+1);
        px = 0;
        [y,b,px] = nlms2(zgomotMotor,semnalCabina,b,mu,sigma,alpha,px);

        b1=b(:,L);
        for i=1:length(b1)-1
            delta(i)=abs(b1(i+1)-b1(i));
        end
        delta(i+1)=delta(i);

        for i=0:(viz/500-1);
            %calculam media pe intervale de cate 500 esatioane
            suma=0;
            for p=(i*500+1):(i+2)*500

```

```
        suma=suma+delta(p);
    end
    med(i+1)=suma/500;
    if med(i+1)<(med(1)/10)
        tc_matrix(k,w)=i*500/Fs; %timpul de convergenta in secunde
        break
    end
end
    end
    w=w+1;
end
    k=k+1;
end
mu_array=mu_min:pas_mu:mu_max;
L_array=L_min:pas_L:L_max;
figure
    surf(L_array,mu_array,tc_matrix);
    title('Timpul de convergenta in functie de mu si L')
    xlabel('L')
    ylabel('μ')
    zlabel('Timpul de convergenta [s]')
    shading interp
L=60;
mu=0.01;
```