


Universitatea Politehnica Bucureşti
Facultatea de Automatică si Calculatoare
Departamentul de Automatică şi Ingineria Sistemelor

LUCRARE DE DIPLOMĂ

Anularea zgomotului de fundal cu un robot umanoid

Absolvent
Cristian GRĂDINARU

Coordonator
Ș. L. dr. ing. Cristian FLUTUR

Bucureşti, 2022

Cuprins

Glosar

API Application Programming Interface. 6

CPU Central Processing Unit. 4

FIR Finite-Impulse Response. 10, 11

FTB Filtru Trece-Banda. 10, 22, 23

FTJ Filtru Trece-Jos. 9

IIR Infinite-Impuse Response. 11

LED Light-Emitting Diode. 7

MCMMP Metoda Celor Mai Mici Patrate. 2, 11, 12, 17, 24

OS Operating System. 5

SDK Software Development Kit. 4, 5, 8, 9, 14

SNR Signal-Noise Ratio. 10

Listă de figuri

Listă de tabele

1. Introducere

1.1. Motivația cercetării

Odată cu evoluția tehnicilor de automatizare și cu miniaturizarea elementelor componente ale unui calculator, s-a dezvoltat ideea creării unor sisteme autonome care să îndeplinească sarcini efectuate în mod normal de către un om. Acele sarcini monotone, care nu necesită creativitate sau al căror procedeu de execuție nu variază de la o iterație la alta se pretează a fi executate de un robot autonom.

Pentru a funcționa autonom un robot trebuie să interacționeze cu mediul înconjurător și să reacționeze la stimulii proveniți de la elementele externe. Împreună cu simțul tactil și vizual, simțul auditiv are o importanță deosebită în interacțiunea omului cu mediul, iar această interacțiune are rareori loc în condiții optime. Din acest motiv robotul trebuie să poată izola semnalul util de zgomotul de fundal pentru a putea comunica eficient cu un interlocutor uman. În cazul auzului uman, această "procesare" este făcută de către creier, însă în cazul microfoanelor rejecția zgomotelor trebuie facută de către programator.

În domeniul rejecției perturbațiilor asupra microfoanelor robotilor umanoizi s-au făcut cercetări în ceea ce privește rejecția zgomotelor produse de către motoarele interne ale robotului pe timpul deplasării [9], precum și identificarea unui sunet de tip fluier într-un spațiu zgomotos [15], dar și studii asupra modalităților de procesare a datelor provenite de la multiplele microfoane prezente pe robotul NAO. [11]

Deși aceste studii și experimente denotă un interes crescut pentru acest tip de roboți autonomi și pentru modalități de îmbunătățire a performanțelor de funcționare ale acestora, niciuna dintre aceste lucrări nu abordează modulul de recunoaștere vocală a robotului și nu se concentrează pe abilitatea unui robot de a înțelege și răspunde cerințelor provenite de la operatori umani.

1.2. Scopul și obiectivele lucrării

Această lucrare își propune să îmbunătățească performanțele modulului de recunoaștere audio de pe robotul umanoid NAO printr-o procedură de anulare a zgomotului de fundal prezent atunci când un vorbitor se adresează robotului. Se remarcă două posibilități de a realiza acest obiectiv: putem să creăm un modul de recunoaștere vocală proprietar, controlând astfel toate aspectele funcționalității acestuia, sau să încercăm să îmbunătățim calitatea semnalului audio ce ajunge la robot prin prelucrare de semnal, ușurând în acest fel sarcina modulului de recunoaștere vocală încorporat. Dată fiind complexitatea crearii unui algoritm de recunoaștere vocală, am decis să folosesc cunostințele de transmisii de date și prelucrarea semnalelor pentru a realiza acest obiectiv.

În primul rând, având în vedere dificultatea implementării unui algoritm de anulare a zgomotului în timp real, în această lucrare se va încerca anularea zgomotului într-un fișier audio de tip .wav înregistrat în prealabil de către robot, și prelucrarea fișierului în mod automat din locația corespunzătoare aflată în spațiul de stocare al robotului.

Deoarece modulul de recunoaștere vocală încorporat pe robotul umanoid NAO este tehnologie proprietară de tip closed-source, ne vom baza de asemenea pe abilitatea robotului de a reda semnal audio cu ajutorul unor difuzoare montate pe capul robotului.

Vom face o serie de comparații ale spectogramelor semnalelor înainte și după prelucrare, precum și o comparație a rezultatelor returnate de modulul de recunoaștere vocală Nuance încorporat pe robot.

1.3. Tehnici de anulare a zgomotului

Domeniul prelucrării semnalelor are ca obiectiv manipularea semnalelor reale în scopul obținerii rezultatului dorit. Translația din mediul teoretic în cel real atrage după sine consecințele imposibilității de reprezentare totală a complexității fenomenelor electrice, sonore sau vizuale. În ceea ce privește mediul fonic, zgomotul a fost un factor omniprezent încă de la primele încercări de captare și redare ale sunetelor.

Conceptul de anulare activă a zgomotului a luat naștere in 1933, cand Paul Lueg a creat principiile anularii active a zgomotului.[8] Aceste principii nu au putut fi aplicate în practică decat mult mai tarziu, odată cu creșterea în popularitate și disponibilitate a calculatoarelor personale. De la acea primă iterație și până în prezent, domeniul a fost într-o continuă dezvoltare, iar mentalitățile și tehnicile abordate s-au modificat odată cu timpurile.

Primele încercări de anulare a zgomotului s-au concentrat pe reducerea zgomotelor pe o anumită bandă de frecvente [2]. Ținând cont de faptul că zgomotele unui avion sau motor au un răspuns în frecvență relativ bine definit, este ușor de înteles de ce filtrele FIR de tip Trece-Jos, Trece-Sus sau Trece-Bandă sunt opțiuni populare de început în cadrul studiilor academice și de specialitate. Aceste filtre sunt însă limitate din punctul de vedere al răspunsului în frecventă. Depășirea acestor limitari a condus la crearea filtrelor adaptive[13], ale căror primă implementare în domeniul anulării zgomotului a fost realizată de Burgess.[5] Filtrele adaptive functionează prin varierea ponderilor cu care se calculează ieșirea filtrului, cel mai popular astfel de algoritm de calcul al ponderilor fiind Metoda Celor Mai Mici Patrate (MCMMP).

Filtrele Weiner [25] sunt de asemenea o variantă ce ar putea fi implementată în scopul rejectiei perturbației fonice. Ele functionează similar cu cele ce implementează MCMMP, adică calculează eroarea medie patratică între semnalul real și cel estimat. Filtrele Weiner au ca dezavantaj necesitatea ca spectrele semnalului vocal și al zgomotului să aibă valori stabile, fapt rar întâlnit în realitate. [2]

Scăderea spectrala este o altă metodă populara ce poate fi utilizată în scopul anulării active a zgomotului. Această tehnică se bazează pe calcularea spectrului zgomotului în micile pauze de vorbire dintr-o conversație și scăderea acestui spectru calculat din spectrul semnalului zgomotos [4]. Rezultatele timpurii ale acestei tehnici au prezentat însă distorsiuni la nivel de timbru vocal, obținându-se un semnal lipsit de zgomot, dar cu un timbru robotic.[26]

1.4. Capitole

În capitolul 2 vom face o descriere sumară a robotului, precum și a specificațiilor tehnice ale acestuia, prezentând pe scurt echipamentul hardware încorporat în robot, și mai pe larg particularitățile software ale sistemului de operare al lui NAO, precum și a modalității de funcționare a librăriilor, modulelor și metodelor care se află la îndemâna programatorului pentru a fi utilizate în cadrul programului dezvoltat de acesta.

Capitolul 3 prezintă modalitățile de realizare ale obiectivelor anterior-menționate (mediile și opțiunile de programare disponibile), precum și o descriere teoretică a modalităților de anulare a zgomotului pentru care am optat în această lucrare.

Capitolul 4 descrie în detaliu implementarea programelor de interfațare cu robotul umanoid Nao, fluxul operațional al acestui proiect, precum și o prezentare a algoritmilor de anulare a zgomotului, însoțite de secțiuni de cod acolo unde este necesar.

Capitolul 5 acoperă rezultatele experimentelor efectuate în cadrul proiectului: parametri operaționali inițiali, datele rezultate după aplicarea algoritmilor de prelucrare, precum și o comparație cu algoritmi de nivel înalt disponibili pe site-urile si forum-urile de specialitate.

În ultimul capitol, al 6-lea, se oferă o viziune de ansamblu asupra rezultatelor întregului proiect, concluzii asupra echipamentului și tehnicilor folosite, și o evidențiere a contribuțiilor personale și a oportunităților viitoare de cercetare.

2. Robotul NAO și modulele sale

2.1. Robotul NAO

Alegerea implementării algoritmilor pe acest robot a fost motivată de faptul că oportunitatea de a folosi un astfel de echipament este una de care se poate profita doar într-un mediu academic, costurile asociate fiind în afara bugetului oricăror cercetari individuale.

NAO este un robot umanoid produs de către compania japoneză SoftBank Robotics, cunoscută și sub numele de Aldebaran robotics, cu scopul de a fi utilizat de către persoane din orice mediu academic, cu diferite pregătiri tehnice în domeniul programării. Astfel robotul NAO, ajuns la a 6-a iterație de dezvoltare, este recunoscut ca o resursă extrem de utilă în domeniul educației și al cercetării, putând fi folosit atât la dezvoltarea interesului în robotică pentru cei mai tineri dintre elevi, cât și pentru cercetări în domenii precum roboți autonomi, vedere artificială sau prelucrare de semnale.

Robotul NAO poate fi programat în funcție de datele furnizate de către multitudinea de senzori pe care îi are în dotare. Sunt disponibili 7 senzori tactili, localizați pe cap, picioare și mâini, senzori de tip sonar și inerțiali pentru orientarea în spatiu, 4 microfoane direcționale montate pe capul robotului, precum și două camere 2-D pentru recunoașterea elementelor din spațiul vizual al acestuia (forme, obiecte, oameni). [19]

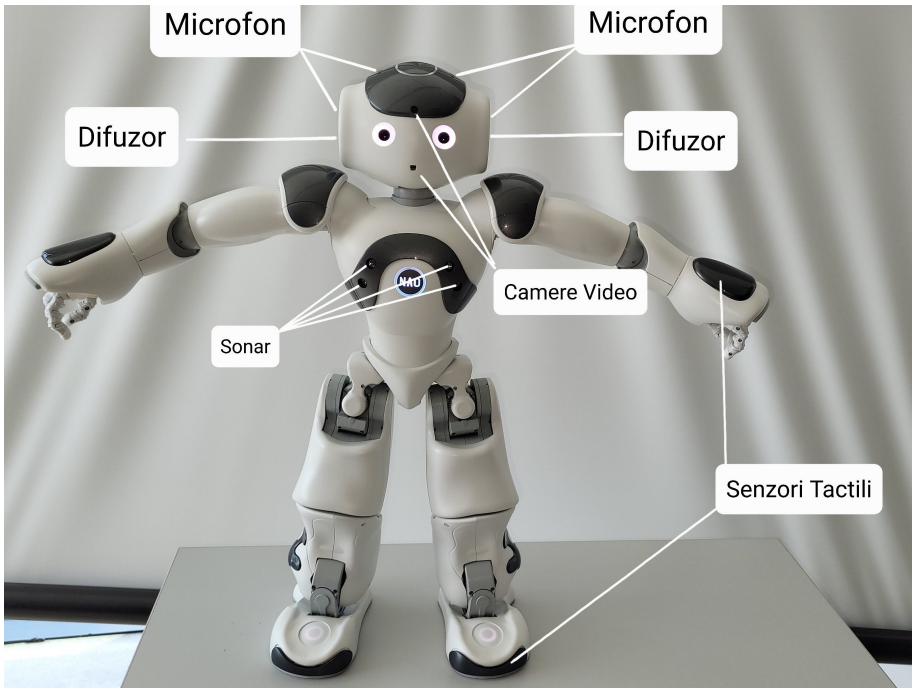

Figura 2.1.: Poză descriptivă a hardware-ului NAO

Ca unitate centrală de procesare Central Processing Unit (CPU) NAO v6 are un procesor Intel Atom E3845 [22] ce poate fi utilizat pentru a programa direct pe robot utilizând fie C++, fie Python, cu ajutorul Software Development Kit (SDK)-ului specializat furnizat de către producător, care conține modulele necesare interacțiunii cu robotul, și răspunsurile preprogramate

ale acestuia. Un exemplu al unui astfel de modul ar fi cel de mișcare intitulat ALMotion; programatorul nu trebuie să dea comandă fiecarui motor în parte pentru a executa o deplasare în linie dreaptă a robotului, ci pur și simplu să apeleze metoda intitulată moveTo, ce aparține modulului anterior menționat. Aceste utilitare vor fi descrise mai amanunțit în secțiunile ce urmează.

În plus față de SDK-urile descrise mai sus, Aldebaran oferă Choregraphe, o modalitate de programare mai prietenoasă cu utilizatorii neexperimentați. Această aplicație este bazată pe blocuri funcționale ce pot fi plasate și interconectate de către utilizator, oferind o libertate mare utilizatorului, care nu mai este nevoit să scrie linii de cod propriu-zise.

2.2. NAOqi

Pentru a putea detalia modalitatea de utilizare a robotului trebuie descrisă în prealabil structura software a robotului programabil. Acesta funcționează cu ajutorul unui sistem de operare denumit NAOqi, derivat dintr-o distribuție de Linux (Gentoo) [21]. NAOqi este un sistem de operare integrat creat de Aldebaran special pentru nevoile robotului programabil NAO, și este responsabil de rularea numărului mare de programe și biblioteci necesare funcționării robotului. Un avantaj al acestei particularități este că programatorul poate trata robotul ca pe un calculator încorporat într-un robot: transferul de fișiere se poate face atât în cod (cu ajutorul bibliotecilor specializate), cât și prin utilitare precum Putty sau WinSCP. Astfel, instalarea de librării noi pe care robotul le poate folosi este relativ facilă pentru utilizatorii experimentați. În plus, se pot folosi mașini virtuale utilizând NAOqi Operating System (OS), creând astfel un mediu virtualizat NAO.

Structura sistemului de operare și a sistemului de fișiere NAO sunt descrise în mai mult detaliu in [21] si [17]. Aceste referințe pot fi consultate dacă se dorește o aprofundare a celor descrise mai sus. Pentru realizarea obiectivelor acestei lucrări, nu va fi nevoie să utilizam abilitatea de virtualizare a sistemului de operare NAOqi OS și nici să interacționam la nivel de linie de comandă cu sistemul de operare bazat pe Gentoo, așadar ne vom folosi doar de cunoștințele referitoare la bibliotecile NAOqi și la sistemul de fișiere.

2.3. Framework-ul NAOqi

Framework-ul (sau Cadrul) NAOqi este un cadru de programare folosit pentru programarea lui NAO, care îndeplinește nevoile uzuale necesare în general în robotică și programare: resurse, paralelism, sincronizare și evenimente. Programele create pentru acesta pot fi dezvoltate în C++ sau Python, utilizatorul nefiind restricționat la un singur limbaj de programare, aceleași metode fiind disponibile indiferent de alegerea programatorului.

2.3.1. Procesul NAOqi

Executabilul NAOqi este un proces ce rulează pe robot la inițializare și care ca scop încărcarea unui fișier de preferințe care definește ce biblioteci vor fi încărcate în sistem, în acest mod îndeplinind rolul de Broker. Fiecare librărie conține cel puțin un modul, iar fiecare modul conține o serie de metode ce pot fi folosite pentru a face robotul să execute comanda dorită. [17]

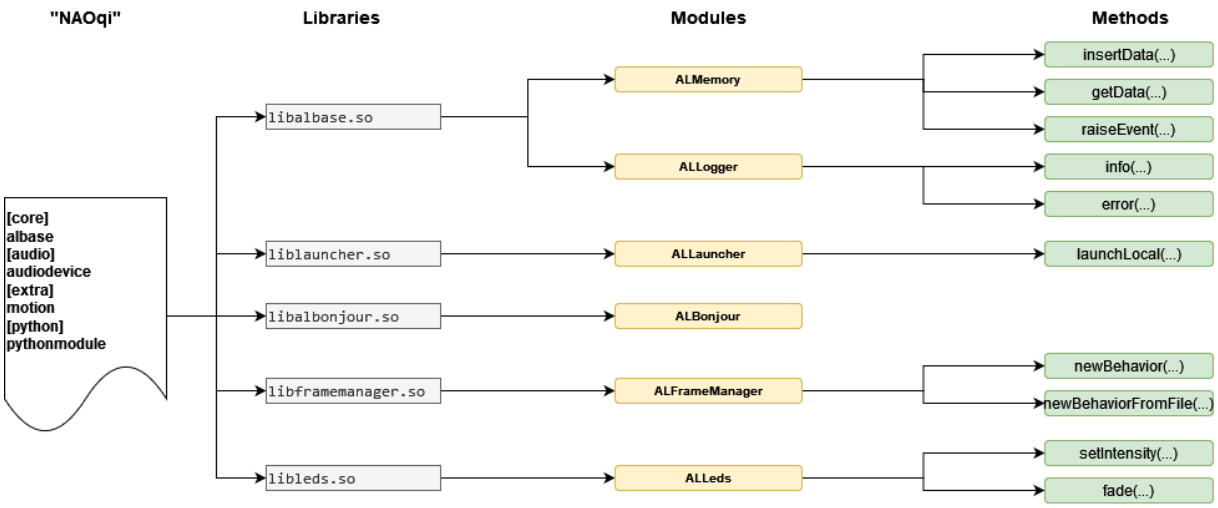

Figura 2.2.: Diagrama framework-ului NAOqi

2.3.2. API-urile NAOqi

Un Application Programming Interface (API) este un set de funcții ce permit aplicațiilor să interacționeze cu elemente externe, sisteme de operare sau servicii. În cazul NAOqi, acestea se pot împărți în mai multe categorii definite în concordanță cu funcția îndeplinită [20]:

- **NAOqi Core**

  Conține module de bază ce sunt mereu disponibile. Fiecare modul are o serie de metode de bază ce pot fi apelate. Câteva dintre modulele importante cuprinse în această categorie sunt: ALBehaviorManager(folositor pentru pornirea și oprirea diverselor metode), ALMemory (folositor pentru a interacționa cu bancurile de memorie ale robotului), ALConnectionManager (pentru gestionarea conexiunii către o rețea) și multe altele.

- **NAOqi Motion**

  Conține toate modulele asociate mobilității robotului precum deplasare, animație și postură. Câteva exemple importante de module din această categorie ar putea fi: ALRobotPosture (folosită pentru a face robotul să execute o postură predefinită (ex. să se ridice sau să se așeze pe podea)), ALNavigation (pentru a comanda robotul să se deplaseze către un anumit set de coordonate, sau o distanță dată de către utilizator), precum și ALTracker (care permite robotului să urmărească diferite obiecte de interes, prin rotații ale capului sau ale corpului).

- **NAOqi Audio**

  Aceasta este categoria de API-uri pe care o vom utiliza cu precădere. Conține module pentru înregistrare audio cu ajutorul microfoanelor, redarea de fișiere audio din spațiul de stocare intern al robotului, precum și un modul proprietar de recunoaștere vocală. Așadar, modulele de interes pentru această lucrare sunt: ALAudioDevice (pentru managerierea inputurilor și outputurilor audio), ALAudioPlayer (pentru redarea fișierelor audio de pe robot), ALSpeechRecognition (pentru recunoașterea cuvintelor spuse de un interlocutor uman), și ALTextToSpeech (pentru a permite textului scris de către programator să fie redat ca semnal audio).

- **NAOqi Vision**

  În această categorie se pot găsi toate metodele care implementează funcționarea celor două camere video 2D montate pe robotul umanoid. Câteva module de importanță sporită

ar fi: ALVisionRecognition (folosit pentru a face robotul să recunoască obiecte sau poze) ALLocalization (cu ajutorul căruia robotul poate să învețe să se reîntoarcă la o ”Casă” definită ca imagine).

- **NAOqi Sensors & :LEDs**

  Categorie responsabilă pentru interacțiunea cu senzorii montați pe robot și cu Light-Emitting Diode (LED)-urile acestuia. Ca exemple de module existente avem: ALLeds (pentru controlul LED-urilor de pe robot) ALTactileGestures (pentru definirea și managerierea gesturilor tacticle de pe capul robotului), precum și module pentru managerierea hardware-ului de pe robot (ALBattery pentru bateria încorporată în robot, ALBodyTemperature pentru atenționarea la trecerea peste un anumit prag de siguranță din punct de vedere al temperaturii componentelor interne).

- Alte categorii: People Perception, Interaction Engines, DCM

API-ul NAOqi poate fi extins de către utilizator, dacă acesta cunoaște procesul de creare a unor module noi.

3. Soluția propusă

3.1. Implementarea în Python versus Choregraphe

Robotul umanoid NAO poate fi programat în mai multe moduri, cele cu compatibilitatea cea mai extinsă fiind aplicația Choregraphe și SDK-urile pentru C++ si Python.

3.1.1. Choregraphe: Descriere și funcționare

Choregraphe este o aplicație de programare vizuală dezvoltată de către SoftBank Robotics, creată cu un accent pentru usurință în utilizare și dezvoltare. Aplicația implementează o structură pe bază de module funcționale, reprezentate prin blocuri, cu o interfață de tip Drag-And-Drop, cu ajutorul căreia se pot realiza programe de complexitate medie și înaltă. Mai mult de atât, fiecare modul functional poate fi customizat, codul Python ce ruleaza în spatele interfeței grafice fiind vizibil pentru utilizator.

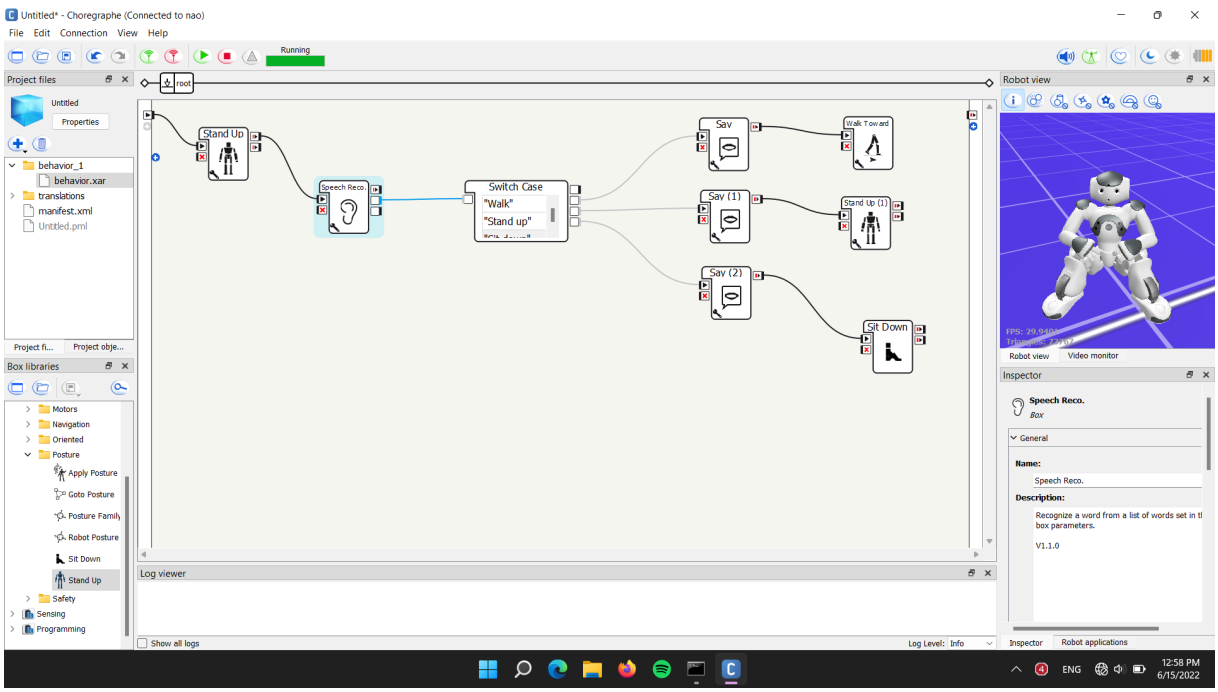

Figura 3.1.: Mediul de dezvoltare Choregraphe

Folosirea acestei aplicații mi s-a părut o modalitate excelentă de a mă familiariza cu capabilitățile robotului. Nu a fost însă soluția optimă pentru proiectul pe care mi l-am propus, deoarece acesta necesită o multitudine de biblioteci noi și modificări ale blocurilor funcționale, iar modul în care este structurată aplicația și în care sunt construite blocurile funcționale nu sunt usor de înțeles. Din acest motiv, am optat să comand comportamentul robotului cu ajutorul SDK-ului Python, pe care îl voi descrie succint în continuare.

3.1.2. Python SDK: Descriere și funcționare

Cea de-a doua modalitate principală de programare a robotului NAO este prin intermediul unui editor de cod, cu limbaj C++ sau Python. Aceste două limbaje sunt cele cu cea mai mare compatibilitate la nivel de sistem de operare cu NAOqi, și totodată variantele recomandate de către producator în documentația oficială de dezvoltare.[18]

Este important de menționat că programul scris în Choregraphe sau Python SDK nu rulează în mod integrat pe robot, ci pe mașina de calcul pe care se scrie programul, aceasta interactionând cu robotul doar atunci când se cheamă o metodă specifică NAOqi. Există și modalități de rulare a codului direct pe mașină, însă sunt necesare tehnici de programare care să țină cont de puterea redusă de calcul pe care o deține procesorul Intel Atom E3845, precum și o instalare în prealabil a tuturor bibliotecilor ce vor fi folosite în codul ce trebuie rulat, instalare care se poate face cu ajutorul unui utilitar precum Putty sau WinSCP.

3.2. Mediul și modalitatea de programare

Pentru această lucrare am scris codul în editorul de cod standard al Python, intitulat Idle. Editarea programului principal în această manieră face necesară utilizarea unui obiect denumit ALProxy prin intermediul căruia vom avea acces la toate metodele și modulele pe care le vom folosi pe robot.

Totodată, datorită modalității în care robotul realizează înregistrările cu ajutorul celor 4 microfoane montate pe capul acestuia, este posibil să realizăm un transfer automat al fișierului de înregistrare. Astfel, luând în considerare lipsa de experiență a studentului cu limbajul de programare Python, contrabalansată de cunostințele semnificativ mai avansate de Matlab, am optat pentru o implementare de algoritmi în ambele limbaje, în functie de beneficiile aduse de fiecare dintre acestea.

3.3. Modalități de anulare a zgomotului

3.3.1. Anularea zgomotului prin filtre de tip medie alunecătoare

Filtrul de tip medie alunecătoare este foarte des utilizat în domeniul prelucrării semnalelor datorită ușurintei în construcție și înțelegere. Acesta este un tip de Filtru Trece-Jos (FTJ) al cărui sarcină principală este să netezească un semnal deja perturbat de zgomot printr-o tehnică de mediere a N valori numerice din semnalul de input și echivalarea lor cu un singur punct de ieșire.

Acest tip de filtru este extrem de bine structurat pentru a netezi semnale, însă extrem de nepotrivit în a separa frecvențele una de cealaltă [23], deoarece un filtru teoretic perfect nu există, și performanțe bune într-un domeniu sunt mereu contrabalansate de performanțe deficitare în celălalt, acest fenomen fiind numit principiul de incertitudine Gabor-Heisenberg.[27]

Filtrul de tip medie alunecătoare este definit de următoarea ecuație:

$$y[n] = \frac{1}{N} \sum_{i=n}^{N+n-1} x[i] \tag{3.1}$$

unde y[n] este outputul la momentul n, N este lungimea filtrului, iar x[i] este inputul la momentul i.

Pentru implementare, este important să aflăm lungimea N a filtrului, iar relația dintre aceasta și frecvența de tăiere poate fi exprimată cu urmatoarea formulă [12]:

$$N = \frac{\sqrt{0.196202 + fc^2}}{fc} \tag{3.2}$$

unde fc este frecvența de tăiere.

3.3.2. Anularea zgomotului prin filtre de tip FIR Trece-Banda

Printre cele mai la îndemână metode de anulare a zgomotului se regăsește utilizarea filtrelor cu răspuns finit la impuls. Acestea au o construcție simplă și sunt usor de înțeles, fiind foarte des utilizate în aplicații de anulare a zgomotului. Filtrele cu răspuns finit la impuls Finite-Impulse Response (FIR) au o structură gândită pentru a manipula semnalul de intrare astfel încât la ieșire să se înregistreze tipul de răspuns dorit. În cazul concret al semnalelor audio, rareori acestea sunt înregistrate într-un mediu lipsit de perturbații. Așadar, cu ajutorul filtrelor și coeficienților acestora, se poate reduce amplitudinea zgomotului, mentinând semnalul util la aceleași amplitudini, mărind astfel Signal-Noise Ratio (SNR) al înregistrării. Filtrele FIR au avantajul că sunt ușor de implementat din punct de vedere al complexității și sunt de fază liniară.

În prima fază am încercat realizarea unui filtru trece banda în forma lui cea mai simplă, fără să apelez la metoda ferestrei. Astfel, dat fiind că funcția fir1 ce implementează filtre de acest tip folosește fereastra Hamming dacă argumentul lipsește, prima iterație de filtru FIR are următorul răspuns în frecvență:

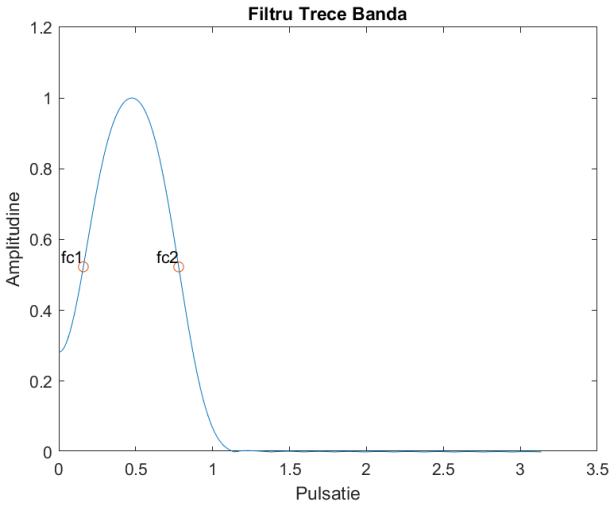

Figura 3.2.: Raspunsul în frecvență al FTB cu fereastră Hamming

Se observă că răspunsul acestuia nu este ideal, și din acest motiv am încercat să aplic un filtru de tip trece banda, dar de această dată cu o fereastră cu o performanță crescută. De asemenea, se observă punctele fc1 si fc2 care sunt pulsațiile de tăiere normalizate ale Filtru Trece-Banda (FTB); între acestea două se regasește banda de trecere. În continuare, bazându-mă pe experiența dobândită în domeniul prelucrării semnalelor am ajuns la concluzia că rezultate mai bune ar putea fi obținute folosind o fereastră Kaiser.[14]

Fereastra Kaiser se poate defini folosind formula [1]:

$$\omega(n) = \frac{I_0(\beta\sqrt{1-(\frac{2n}{M}-1)^2}}{I_0(\beta)} \tag{3.3}$$

unde M este durata ferestrei, I_0 este funcția Bessel de ordinul zero modificată,iar β este înălțimea relativă (în dB) a primului lob parazit (în intervalul [2, 10]). Această formulă poate fi implementată în Matlab folosind funcția Kaiser, care primește ca argumente ordinul și valoarea lui Beta. Pentru $\beta = 0$ se obține o implementare cu fereastră dreptunghiulară.

Răspunsul în frecvență al unui filtru construit cu această fereastră poate fi observat în figura de mai jos:

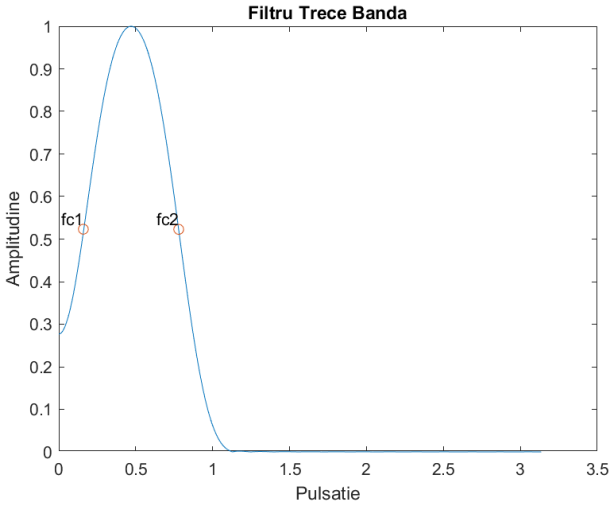

Figura 3.3.: Răspunsul în frecvență al FTB cu fereastră Kaiser

Se observă o foarte mică îmbunătățire în ceea ce privește micșorarea zonei tranzitorii. Pentru această figură s-a ales în continuare ordinul filtrului ca fiind 31, și $\beta = 5$. Scăderea ordinului ar aduce după sine o degradare a performanțelor, iar creșterea acestuia ar îmbunătăți performanțele filtrului, probabil contrabalansată de un cost crescut din punct de vedere al procesării.

3.3.3. Anularea zgomotului prin Filtre Adaptive

O altă clasificare pentru tipurile de filtre, în afară de filtre cu răspuns finit la impus (FIR) și filtre cu răspuns infinit la impuls (Infinite-Impuse Response (IIR)) este cea bazată pe flexibilitatea filtrelor; se remarcă două astfel de categorii: filtre fixe și filtre adaptive. În cazul filtrelor fixe, coeficienții acestora sunt calculați la început și nu își schimbă valoarea pe timpul rulării, spre deosebire de filtrele adaptive ai căror coeficienți se schimbă continuu pe parcursul rulării, prin intermediul unui algoritm de optimizare.

Dat fiind că rezultatele variantei anterioare nu au fost extrem de reușite, am considerat că performanțele se pot îmbunătăți și am decis să caut o altă metodă care ar putea să favorizeze o rejectare a zgomotului mai eficientă. Astfel am ajuns la Filtre adaptive folosind MCMMP.

Această metodă se alătură utilizarii filtrelor Kalman și Weiner, în soluționarea problematicii unui anulator de zgomot adaptiv [10] [6]. MCMMP este cea mai folosită dintre acestea datorită simplicității și robusteții sale.

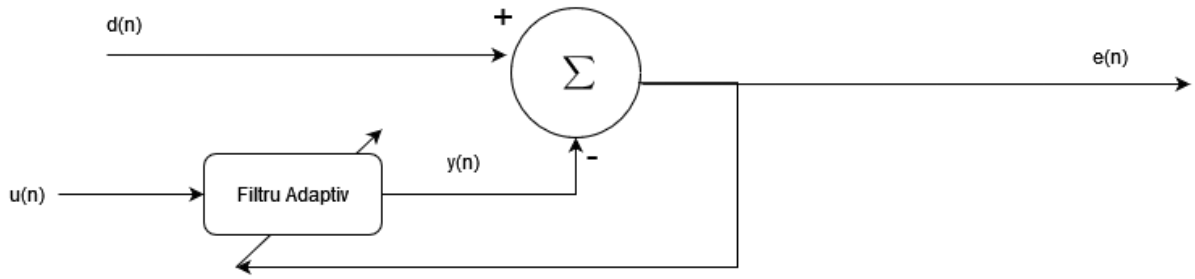

Figura 3.4.: Un sistem de anulare a zgomotului cu filtru adaptiv

Acest tip de anulare a zgomotului numit și anulare adaptivă (Adaptive Noise Cancellation)[16] constă într-o implementare care folosește două semnale distincte: unul pentru zgomot și altul pentru semnalul pe care se încearcă rejectia zgomotului. Vom încearca folosirea unui semnal de referință care este trecut prin filtrul adaptiv pentru a-l face egal cu zgomotul care este prezent în semnalul audio ce conține informația. După aceea, vom scadea din semnalul zgomotos semnalul obtinut la ieșirea din filtru, pentru a obține un semnal neafectat de zgomot[16].

Filtrul MCMMP adaptiv poate fi descris cu ajutorul următoarelor ecuații:

$$y(n) = w(n)u(n), \tag{3.4}$$

unde $\omega(n)$ este coeficientul filtrului la a n-a iterație, u este semnalul de intrare al filtrului, iar y este semnalul de ieșire al filtrului

$$e(n) = d(n) - y(n), \tag{3.5}$$

unde d este răspunsul dorit, iar e este discrepanța dintre răspunsul dorit și ieșirea din filtru

$$\omega(n+1) = \omega(n) + \mu * u(n) * e(n), \tag{3.6}$$

unde μ este dimensiunea "pasului". Dacă μ este mare, atunci algoritmul va converge mai rapid, dar poate duce la instabilitatea sistemului ce folosește algoritmul [3]. Dacă, în schimb, μ este mic, rezultă o convergența lentă care deseori nu poate fi folosită in aplicații practice. Din acest motiv, se poate opta pentru un algoritm MCMMP cu pas variabil, care se bazează pe ideea că se vor folosi valori mari pentru μ atunci când rezultatul este departe de cel căutat și valori mai mici atunci când ne apropiem de rezultatul optim.

4. Detalii despre implementare

4.1. Implementarea pe robotul NAO

Prima provocare în realizarea acestei lucrări a fost înțelegerea fluxului de lucru pe un robot. Așa cum am stabilit în introducere, scopul acestei lucrări este de a îmbunătăți performanțele modulului de recunoaștere vocală cu ajutorul prelucrării de semnal audio înregistrat de către robot. Pentru a realiza acest deziderat trebuie mai întâi detaliată modalitatea de funcționare a acestui modul.

4.1.1. Modulul de recunoaștere vocală

Robotul umanoid NAO are încorporat un sistem de recunoaștere vocală provenit de la compania Nuance, care funcționează prin identificarea cuvintelor captate de către microfoanele robotului și compararea lor cu o bază de date internă. Din acest motiv, funcționarea acestui modul este clar limitată de baza de date deja instalată pe robot. Mai mult decât atât, acest modul este closed-source, nu permite intervenția programatorului asupra vocabularului robotului sau modificarea algoritmilor de detecție; de aceea, în cadrul proiectului am ales să identificăm cuvinte în limba engleza, deoarece presupunem că această limbă va avea performanțele cele mai bune.

În plus, modulul de recunoaștere vocală permite identificarea cuvintelor doar din informația captată de către microfoane odată ce modulul a fost activat, mai exact nu se poate încerca recunoașterea vocală pe un fișier deja existent pe robot. Pentru a depăși această limitare trebuie să recurgem la o metodă de captare a semnalului audio și de redare a acestuia cu ajutorul difuzoarelor montate pe robot.

Din păcate, în urma acestui proces introducem proprietățile constructive ale difuzoarelor robotului în procesul de identificare, deoarece reproducerea semnalului de către acestea nu se poate compara cu complexitatea semnalului audio real.

Pentru a se depăși cel puțin limitările introduse de difuzoare și microfoane se pot implementa mai multe soluții diferite, cum ar fi folosirea unui modul de recunoaștere vocală creat de către utilizator sau deja existent dar care permite recunoaștere pe fișiere, precum și folosirea unor difuzoare sau microfoane mai performante, care să reducă impactul introdus de necesitatea miniaturizării componentelor.

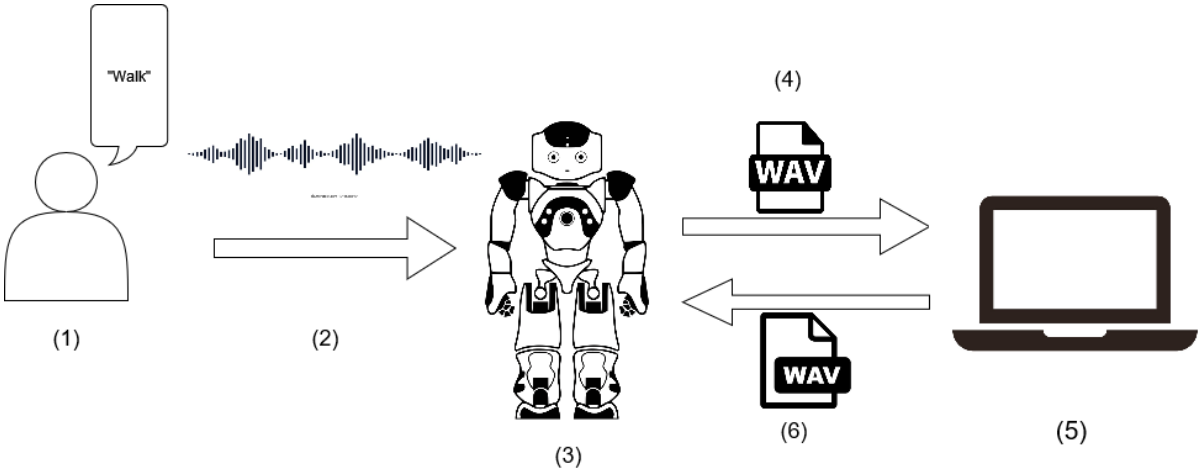

Figura 4.1.: Fluxul de lucru cu Robotul NAO

unde (1) este comanda vocală dată de operatorul uman, (2) este transmiterea undelor prin aer, (3) este procesul de înregistrare al robotului, (4) este transmiterea fișierului .wav către mediul de procesare, (5) este procesarea propriu zisă, iar (6) este întoarcerea fișierului .wav modificat către robot pentru a reîncerca recunoașterea cuvântului.

4.1.2. Algoritmul ce rulează pe robot

Așa cum am stabilit anterior, vom folosi SDK-ul Python pentru a accesa modulele funcționale ale robotului. Această decizie atrage după sine o serie de mici precizări.

În primul rând, interfațarea cu robotul NAO prin Python SDK presupune precizarea IP-ului și portului de comunicație prin care se face conexiunea la robot. În plus, este necesar un modul de interfațare numit ALProxy, al cărui scop este să fie legatura dintre codul Python și modulele încorporate pe NAO; prin intermediul acestuia se va face apelarea fiecărui modul necesar programului nostru.

Aceste particularități sunt specifice acestui mediu de dezvoltare și nu se regăsesc atunci când programatorul lucrează cu Choregraphe, unde aplicația în sine este responsabilă cu interfațarea cu robotul, iar singura sarcină a utilizatorului este să comande acțiunile dorite.

Înregistrarea semnalului

Pentru început, trebuie să obținem o înregistrare a semnalului audio ce se dorește a fi procesat, pentru a putea executa etapele ulterioare. Această înregistrare se poate face folosind mai multe metode: fie cu modulul ALAudioRecorder, fie cu ALAudioDevice. În ambele cazuri utilizatorul are de ales între a înregistra un canal audio, sau pe toate 4, în format .wav sau .ogg. În cazul acestei lucrări, am optat pentru înregistrarea pe 4 canale, la 48000Hz, în format .wav. Am ales aceste setari în principal pentru a asigura o compatibilitate mare cu metodele de procesare disponibile în Matlab sau Python.

Totodată, am ales să lucrăm cu o înregistrare de 5 secunde, care va conține o parte de început în care va fi prezent doar zgomotul de fundal, urmată de o intervenție vocală din partea operatorului uman.

Pașii în care se va realiza procedura sunt următorii: Modulul de interfațare ALProxy se va ”abona” modului ALAudioDevice, iar acesta va declanșa metoda startMicrophonesRecording, urmată de o așteptare de 5 secunde și o apelare a metodei stopMicrophonesRecording. Astfel, fișierul rezultat va fi intitulat record.wav și se va afla în bancurile de stocare internă ale robotului.

Redarea înregistrării pe difuzoarele robotului

Înainte de redarea înregistrării trebuie să pregătim modulul de recunoaștere vocală, ca acesta să fie activ atunci când înregistrarea pornește. Prin urmare, trebuie să ne abonăm la modulul ALSpeechRecognition, și să executăm setările necesare pentru funcționarea corectă a acestuia: setarea limbii și a vocabularului ce va fi recunoscut.

Urmatorul pas este să redăm înregistrarea anterior produsă, iar acest lucru se poate face cu ajutorul modulului ALAudioPlayer. Acest modul poate apela metoda "play", ce primește ca parametru locația fișierului. În urma experimentelor, s-a dovedit faptul că această metodă de redare, deși din punct de vedere teoretic pare că ar funcționa, în practică nu funcționează corespunzator din cauza interferențelor induse de montarea difuzoarelor în imediata proximitate a microfoanelor. Drept urmare am optat pentru redarea semnalului cu ajutorul unui difuzor extern.

Un dezavantaj al acestui modul de recunoaștere vocală este că vocabularul trebuie setat în prealabil, adică robotul nu poate folosi întreaga bază de date pentru a recunoaște în mod dinamic cuvintele auzite, ci ele trebuie precizate anterior de către programator. Acest lucru este un dezavantaj masiv în ceea ce privește programarea robotului pentru recunoaștere independentă de subiectul de discuție, programatorul fiind nevoit să programeze vocabularul de recunoaștere în mod specific pentru subiectul pentru care se încearcă recunoașterea.

Odata ce am făcut setările necesare modulului de recunoaștere vocală, suntem pregătiți pentru a rula această secvență. Însă, pentru a putea spori interactivitatea cu robotul pentru utilizator, mai dorim să reținem într-o variabilă cuvântul identificat de către robot, pentru a putea programa răspunsuri specifice în funcție de cuvântul identificat. Cuvintele identificate de către modulul de recunoaștere sunt automat salvate în memoria robotului, așadar trebuie să ne abonăm și la un modul care ne permite citirea acestor valori din memorie. Modulul cu care putem realiza acest obiectiv este ALMemory, și cu ajutorul metodei getData este posibil să reținem într-o variabilă cuvântul identificat de robot.

Pregătirea fișierului pentru etapa de prelucrare

În această etapă avem fișierul în spațiul de stocare al robotului, însă ținând cont de faptul că prelucrarea va fi facută pe calculatorul conectat la robot (cel puțin atât timp cât programul nu rulează în mod integrat pe robot), ne dorim să îl transmitem în mod automat către calculator. Modalitatea de transfer cu ajutorul Putty nu este de dorit, deoarece este lentă și nu poate fi folosită în mod sustenabil în cadrul fiecărei rulari din cauza dificultății de realizare.

Pentru a îndeplini acest obiectiv am ales să folosesc biblioteca *paramiko*, o bibliotecă bine-cunoscută de transfer de fișiere, ce implementează protocolul SSHv2. Astfel, cu ajutorul acestei biblioteci, și furnizând informații precum username-ul și parola de conectare la robotul NAO, precum și locația din care se dorește preluarea fișierului însoțită de locația în care se dorește depunerea fișierului, se poate realiza un transfer de fișiere în manieră repetabilă, fiabilă și integrată în programul principal.

Modalități de prelucrare

În ceea ce privește prelucrarea propriu-zisă a semnalului, aceasta s-a făcut prin două metode diferite:

- În Python prin scrierea codului inline cu programul principal.
- În Matlab prin rularea unui script cât timp programul principal este în așteptare.

Transmiterea fișierului înapoi pe robot și ultimii pași

După ce fișierul a fost procesat, acesta trebuie transferat înapoi către robot, pentru ca să fie redat prin difuzoarele robotului și să se reîncerce recunoașterea vocală. Pașii acestei etape sunt:

1. Transferul fișierului de pe mașina de calcul către robot cu ajutorul bibliotecii paramiko
2. Pornirea motorului de recunoaștere vocală
3. Redarea fișierului audio cu o compensare de volum pentru a contrabalansa efectul nedorit de reducere a volumului prin prelucrare de semnal
4. Extragerea cuvântului recunoscut de modul și executarea comenzii recunoscute

4.2. Anularea zgomotului folosind filtru de medie alunecătoare

După cum am arătat anterior, pentru anularea zgomotului am utilizat formula de definire a filtrului de medie alunecătoare (3.1). Pentru a implementa aceasta funcționalitate în Python am considerat că cea mai eficientă metodă este folosirea funcției cumsum din biblioteca numpy (returnează suma cumulativă a elementelor pe o anumită axă). În cazul nostru, ne dorim ca această sumă să se realizeze pe intervale de lungime egală cu lungimea filtrului (ex: 3 elemente pentru filtru de lungime 3). De asemenea, ne dorim ca poziția elementelelor adăugate să fie incrementate până când se ajunge la capăt de vector.

Pentru a realiza acest obiectiv, este nevoie de o scădere a două sume cumulative, una corespunzătoare pentru introducerea tuturor termenilor de la primul la [n], iar alta pentru scăderea termenilor până la [n-windowLength]. În plus, va fi nevoie să adăugăm elementul 0 înainte de vectorul de date, în caz contrar rezultatul este corect, însă de dimensiune mai mică decât cea necesară (dacă la intrare se introduce un vector de 5 elemente, la ieșire va rezulta un vector de 2 elemente, în loc să rezulte un vector de 3 elemente). Prin adăugarea unui element nul la început, deplasăm indicii vectorilor astfel încât rezultatul să fie corect.

```
1 def running_mean(x, windowSize):
2   cumsum = np.cumsum(np.insert(x, 0, 0))
3   return (cumsum[windowSize:] - cumsum[:-windowSize]) / windowSize
```

Listarea 4.1: Implementarea funcției de medie alunecătoare

Astfel, linia 2 implementează adăugarea elementului 0 peste vectorul de date x, iar formula de la linia 3 are scopul stabilirii formulei corecte (3.1) pentru fiecare dintre elementele vectorului returnat: adunarea unui număr de elemente egal cu lungimea filtrului, începând cu indici din ce în ce mai mari, rezultat raportat în final la lungimea filtrului.

Această lungime a filtrului nu este impusă de către programator în mod static, ci este calculată conform formulei (3.2). Astfel, formula poate fi implementată cu urmatoarea secvență de cod:

```
1     freqRatio = (cutOffFrequency/sampleRate)
2     N = int(math.sqrt(0.196196 + freqRatio**2)/freqRatio)
```

Listarea 4.2: Determinarea lungimii filtrului de medie alunecătoare

unde freqRatio este frecvența de tăiere normalizată, iar N este lungimea filtrului conform formulei (3.2).

4.3. Anularea zgomotului folosind filtre FIR fixe de tip Trece-Banda

Implementarea acestor filtre a fost facută în Matlab, așadar în prima fază este necesar să citim fișierul .wav transmis de către robot la mașina de calcul în etapa ”Pregătirea fișierului pentru etapa de prelucrare” din secțiunea 4.1.2. Odată ce se realizează acest obiectiv, se definește frecvența Nyquist, precum și frecvențele de tăiere normalizate.

```
[y, Fs]= audioread('record.wav');
Fn = Fs/2;                                  % Nyquist Frequency
fc1 = 1200/Fn;                               % Normalised Frequency
fc2 = 6000/Fn;                              % Normalised Frequency
```

Listarea 4.3: Citire .wav si parametri filtru trece-banda

Urmează etapa de creare a filtrului propriu zis cu ajutorul funcției fir1, precum și trasarea răspunsului în frecvența acestui filtru.

```
B= fir1(31, [fc1 fc2], 'bandpass');
% se foloseste fereastra Hamming ca standard
[H1, om1]=freqz(B);        %raspunsul in frecventa
```

Listarea 4.4: Crearea filtrului și trasarea răspunsului în frecvență

În final, se filtrează datele din vectorul y (vectorul de date) cu ajutorul funcției filtfilt și se scrie informația în fișierul .wav prelucrat cu ajutorul funcției audiowrite.

```
y_Filtered= filtfilt(B, 1, y);
figure(2)
plot(y_Filtered)

audiowrite('Matlab-bandpass-with-Hamming.wav',y_Filtered,Fs);
```

Listarea 4.5: Filtrarea datelor și scrierea fișierului prelucrat

Suplimentar, am făcut mici modificări pentru aplicarea unei ferestre Kaiser asupra unor date identice, modificările având loc în declararea ferestrei și a funcției fir1 pentru filtru după cum urmează:

```
beta=3;
win= kaiser(32, beta);
B= fir1(31, [fc1 fc2], 'bandpass', win);
```

Listarea 4.6: Definirea unui FTB cu fereastră Kaiser

S-au ales parametrii ferestrei Kaiser conform cu observațiile și experiența dobândită în domeniul prelucrării semnalelor: Beta cuprins între 2 și 10, și un ordin suficient de mare, cu dimensiunea ferestrei cu o unitate mai mare decât dimensiunea filtrului.

4.4. Anularea zgomotului folosind filtre adaptive

În cadrul algoritmului de anulare a zgomotului cu ajutorul unui filtru de tip MCMMP adaptiv se vor folosi două semnale diferite pentru calcularea ieșirii: u(n) ce va conține doar zgomotul, și d(n) ce va conține semnalul asupra căruia ne dorim să facem anularea zgomotului. Aceste date vor fi folosite pentru implementarea ecuațiilor (3.4), (3.5), si (3.6) după cum urmează:

```
for n = filterLength: length(signal)
      input = noise(n:-1:n-filterLength+1);
      output(n) = weights' * input;  %Iesirea filtrului
      err(n)  = signal(n) - output(n); %eroarea
      weights = weights + step_size * err(n) * input; %ponderile
end
```

Listarea 4.7: Ecuațiile ce definesc filtrul adaptiv

unde input este u, output este y, err este e, weights este ω și stepsize este μ.

Astfel, la fiecare pas se face o nouă calculare a ieșirii filtrului, iar rezultatul este folosit în calcularea erorii dintre semnalul dorit și cel obținut. Cu ajutorul erorii se updatează apoi ponderile folosite pentru filtrul adaptiv.

5. Rezultatele experimentului

Mediul de testare în care s-au obținut rezultatele a fost laboratorul de sisteme cyberfizice al Facultății de Automatică și Calculatoare din clădirea Precis. Pentru perturbarea semnalului a fost aleasă o poziționare a robotului lângă o fereastră cu vedere la bulevardul Iuliu Maniu (bulevard cu circulație intensă pietonală și motorizată), zgomotul perceptat fiind unul relativ constant, previzibil și repetabil în multiple instanțe din zile diferite. Robotul este staționar pe durata testelor, cu modulul de Autonomous Life oprit (acest modul este responsabil de mișcarea robotului similar unui om: simularea respirației, a micromișcărilor de membre etc).

5.1. Rezultatele fără modificări asupra semnalului

Semnalul nealterat are următoarea formă:

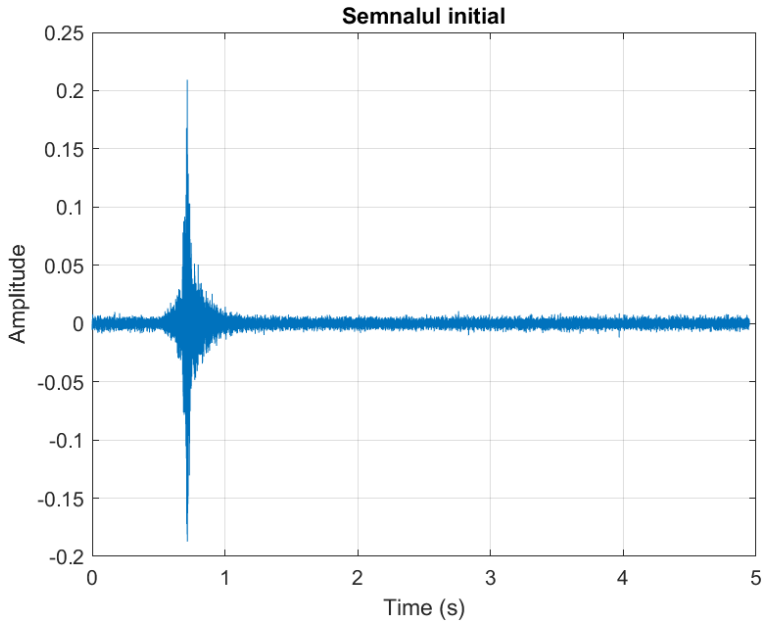

Figura 5.1.: Semnalul inițial în timp

Se remarcă un cuvânt în proximitatea momentului timp de o secunda, iar în rest doar perturbațiile produse de zgomotul orașului.

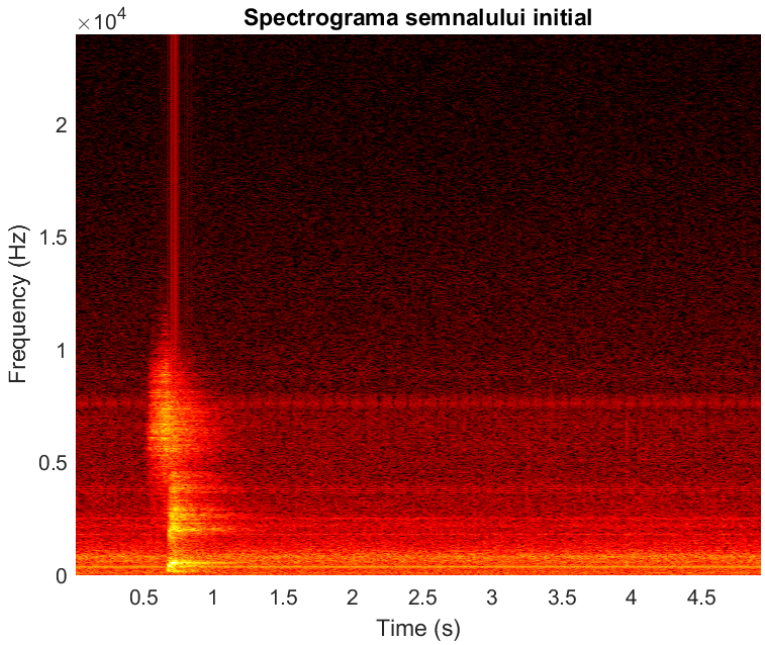

Figura 5.2.: Spectrograma semnalului inițial

Se observă că majoritatea zgomotului de putere mare este în zona frecvențelor joase, intensitatea diminuându-se pe măsură ce frecvența crește. În concordanță cu graficul precedent, se observă creșterea de putere spectrală în intervalul 0.5-1s, această zonă fiind cea pe care ne dorim să o păstrăm.

În continuare ne vom ghida după spectrogramele semnalelor, deoarece acestea ne oferă o indicație mai bună asupra performanțelor algoritmilor nostri decât putem vedea în graficele amplitudine-timp.

5.2. Rezultatele după prelucarea semnalului audio

5.2.1. Cu filtru de medie alunecătoare

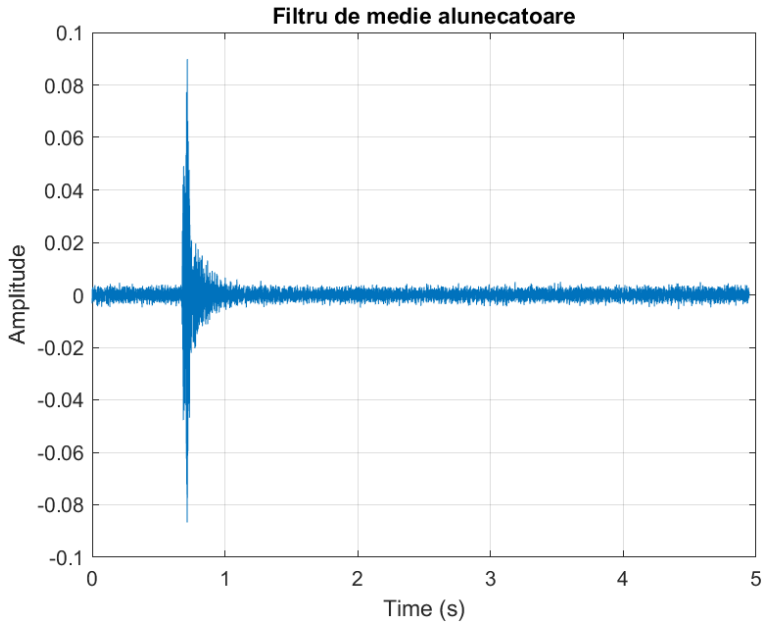

Figura 5.3.: Semnalul la ieșirea din filtrul cu medie alunecătoare

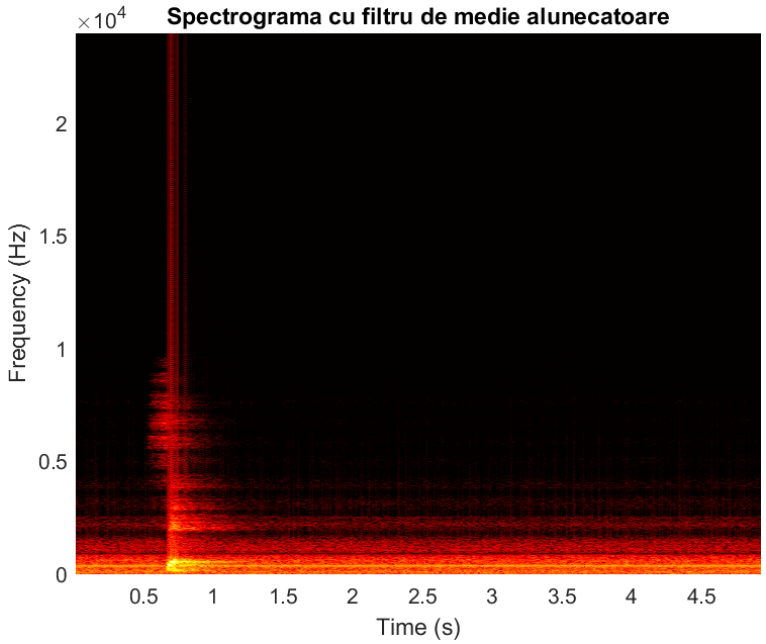

Figura 5.4.: Spectrograma semnalului filtrat cu filtrul cu medie alunecătoare

În cazul filtrului cu medie alunecătoare, acesta rejectează toate perturbațiile de frevențe medii și înalte, dar și o bună parte de perturbații în zona de joasă frecvență, excepție făcând primii 1000Hz. Semnalul util este prezervat într-o bună masură, iar aceste observații sunt susținute de o audiție a fișierului rezultat.

5.2.2. Cu filtru FIR Trece-Banda

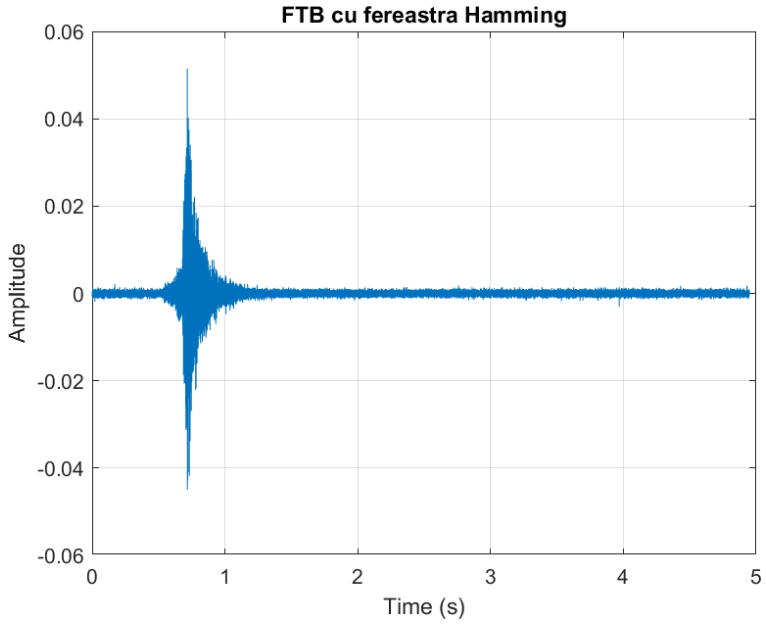

Figura 5.5.: Semnalul la ieșirea FTB cu fereastră Hamming

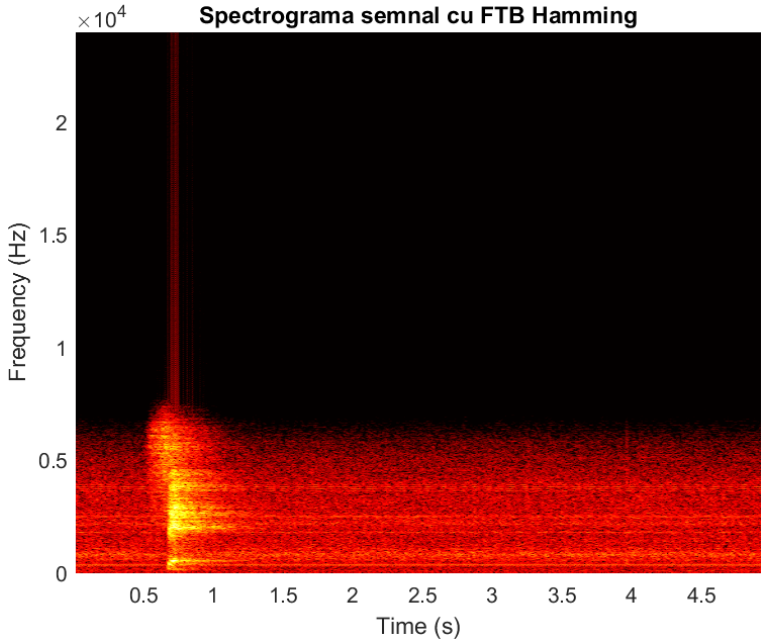

Figura 5.6.: Spectrograma semnalului filtrat cu FTB cu fereastră Hamming

În cazul realizărilor unor FTB, se observă o mică îmbunătățire față de semnalul inițial în rejecția perturbațiilor la frecvențe foarte joase, însă banda de stop dintre 0hz și frecvența de tăiere inferioară nu poate fi largă deoarece semnalul util se află în acest interval. În ceea ce privește frecvențele de deasupra frecvenței de tăiere superioară, acestea sunt în totalitate rejectate.

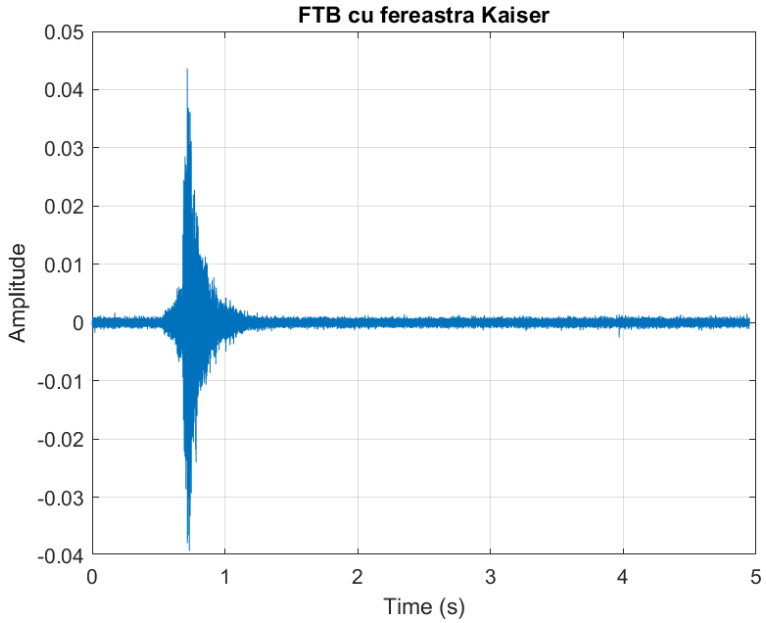

Figura 5.7.: Semnalul la ieșirea FTB cu fereastră Hamming

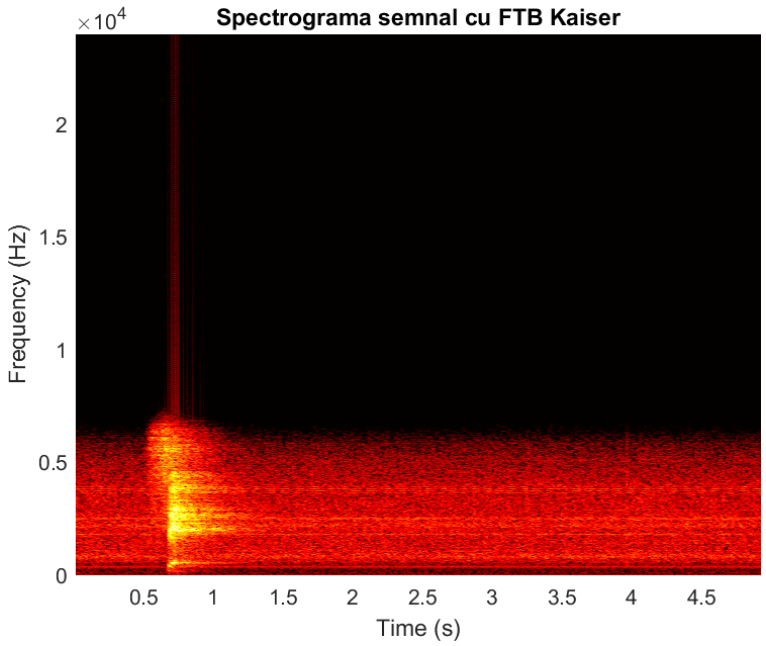

Figura 5.8.: Spectrograma semnalului filtrat cu FTB cu fereastră Kaiser

Nu se remarcă diferențe ușor vizibile între realizarea unui FTB cu fereastră Hamming față de cel realizat cu fereastră Kaiser, ceea ce este parțial de așteptat, deoarece caracteristicile acestor două ferestre sunt suficient de asemănătoare încât să poată fi confundate.

5.2.3. Cu filtre adaptive

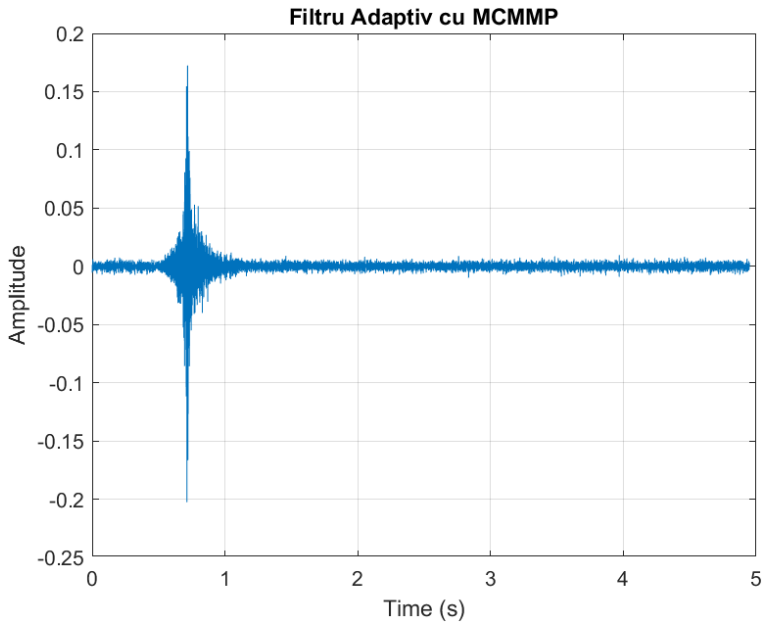

Figura 5.9.: Semnalul la ieșirea filtrului adaptiv cu MCMMP

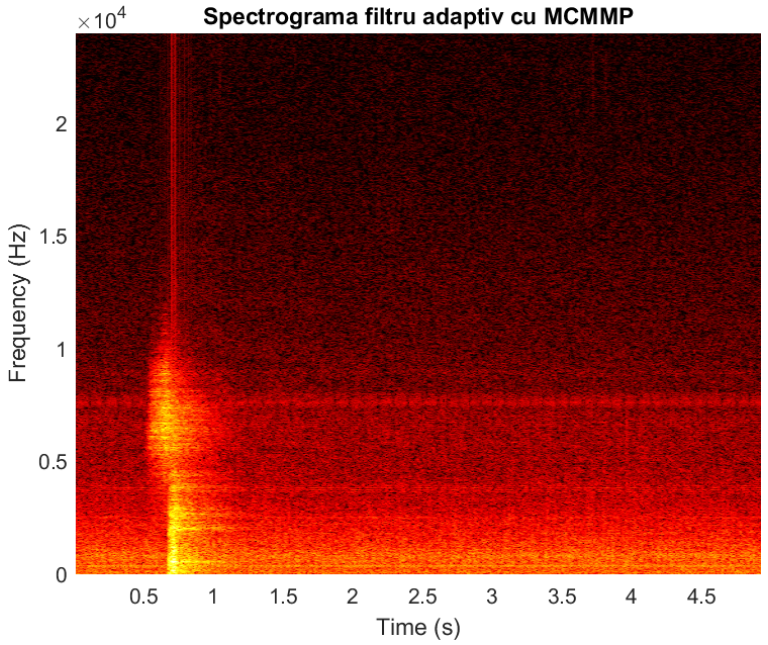

Figura 5.10.: Spectrograma semnalului filtrat cu filtrul adaptiv cu MCMMP

În cazul filtrului adaptiv realizat cu MCMMP, se observă că modalitatea de construcție a acestuia nu pare să fie performantă în cazul unui zgomot precum cel prezent în înregistrarea noastră, care este similar cu un zgomot alb. Caracterul aleator al zgomotului se răsfrânge asupra calculului ieșirii filtrului ca d-y și face ca micile variații să fie foarte dificil de atenuat.

5.2.4. Comparație cu metode deja existente

Versus Vuvuzela

Acest algoritm pare să fie cel mai bun din punctul de vedere al rejecției perturbațiilor. Algoritmul [24] este o implementare a algoritmului de anulare a zgomotului cu scădere spectrală propus de către Stephen F. Boll în [4]. Algoritmul a fost inițial realizat pentru o versiune anterioară a Matlab, astfel că a fost nevoie de o înlocuire a funcțiilor acolo unde acestea nu mai erau incluse în versiunile ulterioare ale programului dezvoltat de Mathworks.

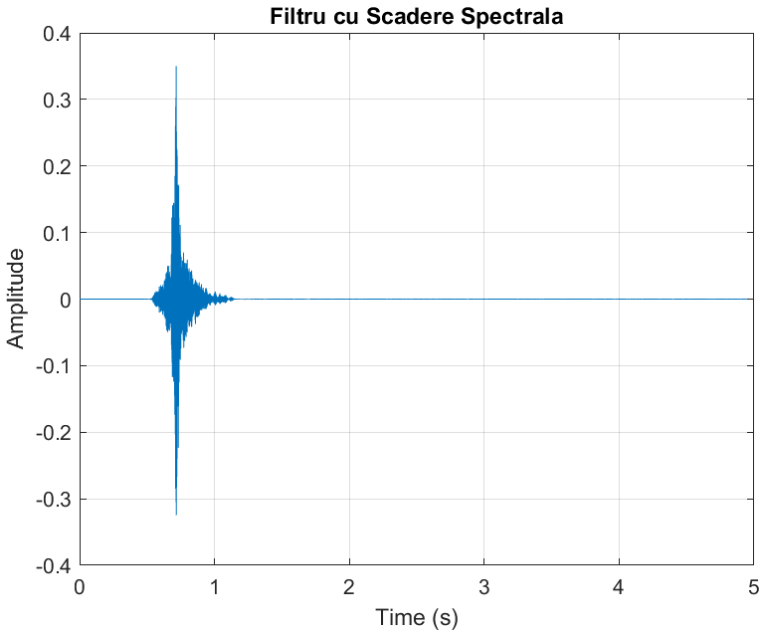

Figura 5.11.: Semnalul la ieșirea din filtrul cu scădere spectrală

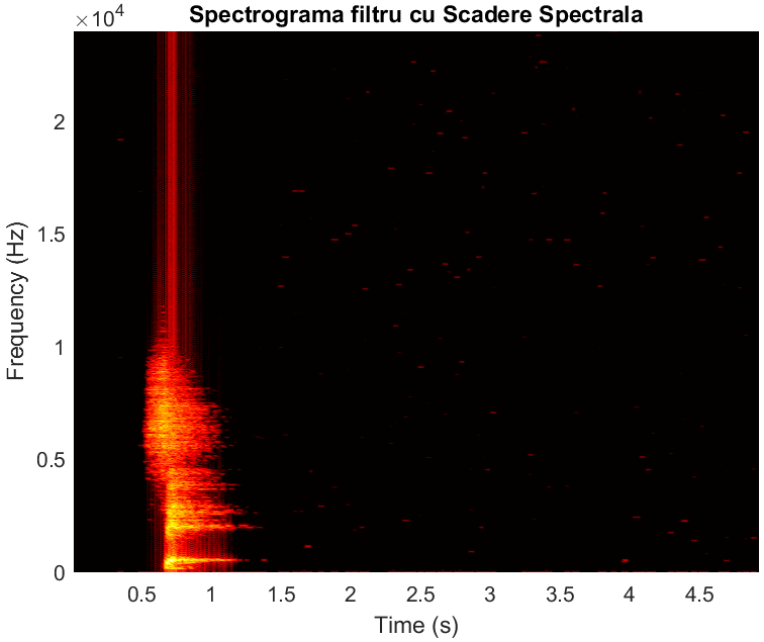

Figura 5.12.: Spectrograma semnalului la ieșirea din filtrul cu scădere spectrală

Se observă că acesta păstrează aproape în totalitate semnalul util, și rejectează perturbațiile eficient pe toată plaja de frecvențe. Acest proces se face cu ajutorul unui extras de zgomot

de la începutul fișierului audio, care se folosește apoi pentru a realiza spectrul acestui zgomot, și scăderea lui sistematică din fișierul semnalului util cu ajutorul transformatei Fourier de timp scurt (Short-time Fourier Transform- STFT). [4] [7]

5.2.5. Încorporarea soluției cu algoritmul de recunoaștere vocală

Succesul de implementare al tuturor acestor metode de prelucrare trebuie să se reflecte în încrederea cu care robotul NAO recunoaște cuvintele rostite de către vorbitor. Dacă robotul nu reușește să recunoască cuvintele cu o încredere mare, sau chiar deloc înainte de etapa de prelucrare, concluzia după rejectarea zgomotului cu ajutorul algoritmilor anterior-mentionați trebuie să fie că această peformanță de recunoaștere a fost îmbunătățită.

În practică, rezultatele returnate de către robot se află sub formă de vector, cu primul element conținând cuvântul identificat încadrat între apostrofuri, sau None dacă niciun cuvânt nu a fost recunoscut, urmat de un număr între 0 și 1 care reprezintă încrederea algoritmului în identificarea făcută, așa cum se poate observa în captura de ecran următoare:

```
data: ['sit', 0.4406999945640564]
```

Figura 5.13.: Forma în care se returnează datele în urma rulării algoritmului de recunoaștere vocală

Rezultatele comparației încrederii de identificare a cuvintelor de către robotul umanoid sunt încurajatoare. În majoritatea cazurilor, identificarea aposteriori prelucrării returnează fie rezultate mai bune, fie rezultate ce se află in marja de eroare de 5 procente, în care nu se poate spune cu certitudine că identificarea a fost îmbunatițită datorită algoritmului de rejecție a zgomotelor, sau datorită șansei.

```
========== RESTART: C:\Python27\Experiment.py ==========

Warning (from warnings module):
  File "C:\Python27\lib\site-packages\paramiko\transport.py", line 33
    from cryptography.hazmat.backends import default_backend
CryptographyDeprecationWarning: Python 2 is no longer supported by the Python co
re team. Support for it is now deprecated in cryptography, and will be removed i
n the next release.
data: ['stand', 0.3321000039577484]
data: ['stand', 0.3714999854564667]
>>>
```

Figura 5.14.: Rezultatul identificării în urma filtrării cu o metodă bazată pe un filtru de medie alunecătoare

```
============== RESTART: C:\Python27\Python_without_Playback.py ==============

Warning (from warnings module):
  File "C:\Python27\lib\site-packages\paramiko\transport.py", line 33
    from cryptography.hazmat.backends import default_backend
CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team.
pygame 2.0.3 (SDL 2.0.16, Python 2.7.18)
Hello from the pygame community. https://www.pygame.org/contribute.html
data: ['sit', 0.23309999704360962]
data: ['stand', 0.3686000108718872]
>>>
```

Figura 5.15.: Rezultatul identificării în urma aplicării metodei de rejecție a zgomotului bazată pe scădere spectrală

S-au observat totuși câteva ocazii în care identificarea a fost deficitară după aplicarea algoritmilor de filtrare, deși conditiile de testare au fost păstrate cât mai unitar posibil.

```
>>>
============== RESTART: C:\Python27\Python_without_Playback.py ============

Warning (from warnings module):
  File "C:\Python27\lib\site-packages\paramiko\transport.py", line 33
    from cryptography.hazmat.backends import default_backend
CryptographyDeprecationWarning: Python 2 is no longer supported by the Pyth
pygame 2.0.3 (SDL 2.0.16, Python 2.7.18)
Hello from the pygame community. https://www.pygame.org/contribute.html
data: ['sit', 0.27869999408721924]
data: ['sit', 0.14509999752044678]
>>>
```

Figura 5.16.: Exemplu variație al rezultatului identificării

6. Concluzii și dezvoltări viitoare

6.1. Concluzii generale

În concluzie, elaborarea acestei lucrări mi-a oferit oportunitatea de a pune în practică cunoștințe dobândite pe parcursul facultății și de a lua contact cu un domeniu aflat în plină dezvoltare.

În ceea ce privește robotul NAO, consider că este o platformă didactică capabilă, însă nu perfectă. Ca puncte forte se remarcă abilitatea de a programa în multiple modalități, croite pe niveluri de complexitate pentru programatori din orice etapă a pregătirii lor, multitudinea de elemente cu care robotul poate interacționa cu mediul (senzori tactili, camere, microfoane) precum și construcția robustă a robotului.

Ca dezavantaje, aș putea enumera dificultatea de a găsi modalitatea corectă de programare și lipsa de versatilitate a multora dintre module.

Au constituit provocări alegerea mediului de programare și găsirea modalităților de interfațare cu modulele și metodele NAOqi. De asemenea, documentația disponibilă online este suficientă, însă nu este structurată într-o manieră accesibilă, și deseori la nivel de suport modalitatea de depășire a unui obstacol este aceea de încercare-și-eșec.

Din punctul de vedere al prelucrării de semnale, am reușit să înțeleg mai în amănunt conceptele învățate pe parcursul facultății, dar mai presus de orice, am observat cât de vast și complex este acest domeniu. Înainte de a începe proiectul știam la nivel macro în ce constă prelucrarea de semnale audio, însă această experiență în care a trebuit să cercetez prelucrarea de semnal la nivel micro mi-a oferit un nou avânt în ceea ce privește entuziasmul meu de a afla mai multe despre domeniu și de a intelege și mai multe despre tehnicile cele mai eficiente.

6.2. Contribuții personale

Consider ca importanța disponibilității de a ne confrunta cu probleme noi este deseori subapreciată, și din acest motiv cred ca la nivel de contribuții personale probabil cel mai semnificativ succes a fost faptul că am reușit să înțeleg cum funcționează elementele complexe ce alcătuiesc un robot umanoid performant, și cum se pot folosi funcțiile și metodele încorporate pe NAO pentru a opera într-o manieră originală. În plus, am reușit să folosesc un mediu de dezvoltare avansat bazat pe limbajul Python, cu care nu aveam foarte multă experiență anterioară, și cu ajutorul documentației prezente pe site-ul SoftBank robotics [20], am înțeles cum se apelează metodele în cadrul unui program Python ce interfațează cu NAOqi.

La nivel de prelucrare de semnale, mi-am dezvoltat competențele de implementare a algoritmilor în formă matematică, precum și abilitatea de a analiza documente de specialitate și de a concluziona asupra fezabilității implementarii acestora în cazul acestei lucrări.

6.3. Dezvoltari viitoare

În urma dificultăților pe care le-am întâmpinat în timpul dezvoltării, sunt de părere că dezvoltarea unui modul de recunoaștere vocală propriu ar putea fi un domeniu de cercetare fascinant și totodata util în folosirea NAO în scopuri de prelucrare de semnal. Recunoașterea vocală încorporată la momentul actual nu este suficient de robustă pentru a comunica în mod fluent cu NAO, programatorul fiind nevoit să definească în prealabil vocabularul în baza căruia se va face detecția, ceea ce înseamnă că de fiecare dată când se dorește o schimbare de subiect cu robotul, vocabularul acestuia trebuie redefinit, și programul trebuie rescris pe robot. În plus, recunoașterea în sine returnează o încredere de detecție de aproximativ 60 de procente, deși mediul în care se face detecția este aproape de cel ideal.

O altă direcție de dezvoltare ar putea fi integrarea lui NAO cu un algoritm de anulare a zgomotului specializat și eficient care să poată rula direct pe NAO, astfel încât acesta să poată fi folosit oricând se solicită recunoașterea vocală din partea robotului umanoid, extinzând astfel domeniile de utilizare ale robotului.

Din armonizarea celor două sugestii ar putea rezulta obținerea unui sistem de recunoaștere vocală capabil să recunoască comenzi din mai multe domenii, dotat cu abilitatea de a identifica comenzile și în medii nefavorabile (ex. sală de clasă zgomotoasă, șantier de construcții).

Anularea zgomotului de fundal cu un robot umanoid Cristian Gradinaru Cristian Flutur		
	Activitate	Durată [zile]
1	Cercetare concentrata pe capabilitatile robotului NAO	5
2	Cercetare concentrata pe modalitatile de programare NAO	6
3	Cercetare concentrata pe modalitatile de anulare a zgomotului disponibile	7
4	Implementarea programului principal NAO	4
5	Implementarea algoritmilor de anulare a zgomotului in Python si Matlab	5
6	Testarea solutiilor, rezolvarea defectelor si analiza rezultatelor	2.5
7	Redactarea lucrarii scrise	11
	Total	40.5

Tabelul 6.1.: Lista contribuțiilor personale

Anexe

A. Fișiere sursă

```
#importing necesary libraries
import naoqi
import numpy
from naoqi import ALProxy
import time
import scipy
import scipy.io
from scipy.io import wavfile
from scipy.signal import butter, lfilter
import paramiko
import matplotlib.pyplot as plt
import numpy as np
import wave
import sys
import math
import contextlib
from pygame import mixer



naoIP= "169.254.72.52"
port=9559
word=[]

#We will be using nao's text to speech functionality to let the user know
# the robot has successfully initialized

tts= ALProxy("ALTextToSpeech", naoIP, port) #initializing the text to speech module
tts.say("The robot is starting to record")

time.sleep(1)   #waiting to give the user time to interact

# we can choose whether to use ALAudioDevice or ALAudioRecorder
# for this particular subject, we will be using ALAudioDevice
AD= ALProxy("ALAudioDevice", naoIP, port)
AD.startMicrophonesRecording("/data/home/nao/recorings/microphones/record.wav")
time.sleep(5) # we want to record for 5 seconds
AD.stopMicrophonesRecording()

tts.say("The robot has stopped recording")
time.sleep(1)

#automatic file transfer protocol
NAO_IP = naoIP
NAO_USERNAME = "nao"
NAO_PASSWORD = "Quetzal_3"
word=[]
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(NAO_IP, username=NAO_USERNAME, password=NAO_PASSWORD)
sftp = ssh.open_sftp()
localpath = 'D:\\record.wav'
remotepath = '//data//home//nao//recorings//microphones//record.wav'
sftp.get(remotepath, localpath)
sftp.close()

```

```
57 #now we should try a recognition on the sounds, played by nao's internal speakers
58
59 #speech recognition module initializations
60 asr= ALProxy("ALSpeechRecognition", naoIP, port)
61 asr.pause(True)
62 asr.setLanguage("English")
63 vocabulary= ["stand", "sit"]
64 asr.setVocabulary(vocabulary, False)
65
66
67 #subscribing to the speech recognition module
68 tts.say("The robot is starting the speech recognition service")
69 asr.subscribe(naoIP)   #subscribing to the speech recognition module
70 mem= ALProxy("ALMemory", naoIP, port)
71 mem.subscribeToEvent('WordRecognized', naoIP, 'wordRecognized')
72
73
74 #playing the file on the host machine
75 asr.pause(False)
76 time.sleep(2)
77 mixer.init()
78 mixer.music.load('D:\\record.wav')
79 mixer.music.play()
80
81 #the alloted time slot has expired
82 time.sleep(7)
83 asr.unsubscribe(naoIP)
84 word= mem.getData("WordRecognized")
85 print( "data: %s" % word )
86 tts.say(" The recognized word is %s" %word[0])
87 time.sleep(2)
88 tts.say("Robot finished the speech recognition procedure")
89 mem.removeData("WordRecognized")
90 #process it
91 tts.say("Robot is starting processing")
92
93
94
95 fname = 'D:\\record.wav'
96 outname = 'D:\\off_plus_noise_filtered.wav'
97
98 fc = 600.0
99
100 #calculating the running mean
101 def running_mean(x, windowSize):
102   cumsum = np.cumsum(np.insert(x, 0, 0))
103   return (cumsum[windowSize:] - cumsum[:-windowSize]) / windowSize
104
105 #default recording mode is interleaved ( otherwise channels.shape = (n_channels,
      n_frames))
106 def interpret_wav(raw_bytes, n_frames, n_channels, sample_width):
107     dtype = np.int16 # signed 2-byte short
108
109     channels = np.fromstring(raw_bytes, dtype=dtype)
110
111     channels.shape = (n_frames, n_channels)
112     channels = channels.T
113
114     return channels
115
116 with contextlib.closing(wave.open(fname,'rb')) as winfo:
117     sampleRate = winfo.getframerate()
118     ampWidth = winfo.getsampwidth()
119     nChannels = winfo.getnchannels()
120     nFrames = winfo.getnframes()
121
```

```
    #get Filter Length
    normFreq = (fc/sampleRate)
    N = int(math.sqrt(0.196196 + normFreq**2)/normFreq)

    # Extract audio
    signal = winfo.readframes(nFrames*nChannels)
    winfo.close()

    channels = interpret_wav(signal, nFrames, nChannels, ampWidth)

    # Use moviung average (only on first channel)
    filtered = running_mean(channels[0], N).astype(channels.dtype)

    wav_file = wave.open(outname, "w")
    wav_file.setparams((1, ampWidth, sampleRate, nFrames, winfo.getcomptype(), winfo
    .getcompname()))
    wav_file.writeframes(filtered.tobytes('C'))
    wav_file.close()


sftp = ssh.open_sftp()
localpath = 'D:\\off_plus_noise_filtered.wav'
remotepath = '/data/home/nao/recorings/microphones/off_plus_noise_filtered.wav'
sftp.put(localpath, remotepath)
sftp.close()
ssh.close()
#retry recognition
tts.say("The robot is playing audio clip")
asr.subscribe(naoIP)
mem.subscribeToEvent('WordRecognized', naoIP, 'wordRecognized')
time.sleep(2)
mixer.init()
mixer.music.load('D:\\anti_vuvuzela.wav')
mixer.music.play()
time.sleep(7)
asr.unsubscribe(naoIP)
word= mem.getData("WordRecognized")
if word[0]== 'sit':
  pos= ALProxy("ALRobotPosture", naoIP, port)
  pos.goToPosture('Sit', 0.5)
if word[0]== 'stand':
  pos= ALProxy("ALRobotPosture", naoIP, port)
  pos.goToPosture('Stand', 0.5)
print( "data: %s" % word )
tts.say(" The recognized word is %s" %word[0])
time.sleep(2)
tts.say("Robot finished the speech recognition procedure")
```

Listarea A.1: Cod Python – Codul care ruleaza pe robot

```
%Bandpass Filter on data
[y, Fs]= audioread('record.wav');
Fn = Fs/2;                              % Nyquist Frequency
fc1 = 1200/Fn;                           % Normalised Frequency
fc2 = 6000/Fn;                          % Normalised Frequency

beta=3;
win= kaiser(32, beta);
B= fir1(31, [fc1 fc2], 'bandpass', win);
[H1, om1]=freqz(B);        %raspunsul in frecventa
figure(1)
fc1= [  0.159534 0.779262];
fc2= [  0.5232  0.522845];
labels= {'fc1', 'fc2'};
plot(om1, abs(H1));
% hold on
```

```
plot(fc1, fc2, 'o')
title('Filtru Trece Banda');
xlabel('Pulsatie')
ylabel('Amplitudine')
text(fc1, fc2, labels, 'VerticalAlignment','bottom', 'HorizontalAlignment', 'right')
hold off

y_Filtered= filtfilt(B, 1, y);
figure(2)
plot(y_Filtered)

audiowrite('Matlab-bandpass-withspectogram.wav',y_Filtered,Fs);
```

Listarea A.2: Cod Matlab – Codul pentru realizarea unui filtru trece banda cu fereastra Kaiser

```
%Filtru Adaptiv cu MCMMP

[signal1, Fs]= audioread('record.wav');
[noise1, Fsn]= audioread('justNoise.wav');
signal=signal1(:, 1)*10;
noise= noise1(:, 1)*10;


% Parametrii filtrului Adaptiv
step_size= 5;
filterLength = 28;

% initializarea parametrilor cu care vom lucra
weights = zeros(filterLength, 1);
output = zeros(1,length(signal));
err = zeros(1,length(signal));
input = zeros(1,filterLength);

% Bucla functionala
for n = filterLength: length(signal)
      input = noise(n:-1:n-filterLength+1);
      output(n) = weights' * input;  %Iesirea filtrului
      err(n)  = signal(n) - output(n); %eroarea
      weights = weights + step_size * err(n) * input; %ponderile
end

yClean = err/10;

audiowrite('AdaptiveFilterExp.wav',yClean,Fs);
```

Listarea A.3: Cod Matlab – Codul pentru realizarea unui filtru adaptiv cu MCMMP

```
%Incarcarea datelor si masurarea lungimii inregistrarii in esantioane
[y,Fe]=audioread('record.wav');
x=y(1:end,1).';  %remove the beginning of the sample
Nx=length(x);
fprintf('Audioread s-a executat cu succes');

%Parametrii algoritmului
apriori_SNR=1;  %A se selecta 0 pentru estimare aposteriori a SNR, 1 pentru apriori
    (see [2])
alpha=0.05;      %only used if apriori_SNR=1
beta1=0.5;
beta2=1;
lambda=3;

%parametrii STFT
NFFT=2048;   %numarul de puncte in care se face calculul TF discrete
window_length=round(0.031*Fe); %lungimea ferestrei, are un mare impact asupra
    roboticitatii vocii, tried to match it
window=kaiser(window_length, 3);
```

```
window = window(:);
overlap=floor(0.3*window_length); %numarul de ferestre folosite fara overlap, are impact asupra amplitudinii semnalului, matched it to the input

%Parametrii semnalului ce va fi folosit pentru zgomot
t_min=0.01;    % [t_min t_max] intervalul in care se va izola zgomotul
t_max=0.5;

%constructia spectogramelor
[S,F,T] = spectrogram(x+1i*eps,window,window_length-overlap,NFFT,Fe); %s-a adaugat un 1i*eps pentru a adauga o foarte mica parte imaginara si pt a obtine o spectograma cu doua parti(?)
[Nf,Nw]=size(S);

%S= contine o estimare de termen scurt a lui X
%F= vector de frecvente iar T este un vector de timp, ambele specifica
%momentele in care spectograma este calculata


%Extragerea spectrului zgomotului
t_index=find(T>t_min & T<t_max);  %indexii de timp in care T este cuprins intre cele doua intervale, specifica cat din spectograma este zgomotul de la inceput
absS_vuvuzela=abs(S(:,t_index)).^2;
vuvuzela_spectrum=mean(absS_vuvuzela,2); %spectrul vuvuzelei mediu
vuvuzela_specgram=repmat(vuvuzela_spectrum,1,Nw);
fprintf('Zgomotul s-a izolat cu succes');

%Calcularea SNR
absS=abs(S).^2;
SNR_est=max((absS./vuvuzela_specgram)-1,0); % a posteriori SNR = Semnal supra zgomot totusi
if apriori_SNR==1
    SNR_est=filter((1-alpha),[1 -alpha],SNR_est);  %a priori SNR: [2] eq 53
end

%----------------------------%
%  Compute attenuation map  %
%----------------------------%
fprintf('-> Step 4/5: Compute TF attenuation map -');
an_lk=max((1-lambda*((1./(SNR_est+1))).^beta1)).^beta2,0);
STFT=an_lk.*S;
fprintf(' OK\n');

%---------------------------%
%   Compute Inverse STFT   %
%---------------------------%
fprintf('-> Step 5/5: Compute Inverse STFT:');
ind=mod((1:window_length)-1,Nf)+1;
output_signal=zeros((Nw-1)*overlap+window_length,1);

for indice=1:Nw %Overlapp add technique
    left_index=((indice-1)*overlap) ;
    index=left_index+[1:window_length];
    temp_ifft=real(ifft(STFT(:,indice),NFFT));
    output_signal(index)= output_signal(index)+temp_ifft(ind).*window;
end
fprintf(' OK\n');


%-----------------    Display Figure   -------------------------------------

%show temporal signals
figure
subplot(2,1,1);
t_index=find(T>t_min & T<t_max);
plot([1:length(x)]/Fe,x);
```

```
xlabel('Time (s)');
ylabel('Amplitude');
hold on;
noise_interval=floor([T(t_index(1))*Fe:T(t_index(end))*Fe]);
plot(noise_interval/Fe,x(noise_interval),'r');
hold off;
legend('Original signal','Noise Only');
title('Original Sound');
%show denoised signal
subplot(2,1,2);
plot([1:length(output_signal)]/Fe,output_signal );
xlabel('Time (s)');
ylabel('Amplitude');
title('Sound without noise');

%show spectrogram
t_epsilon=0.001;
figure
S_one_sided=max(S(1:length(F)/2,:),t_epsilon); %keep only the positive frequency
pcolor(T,F(1:end/2),10*log10(abs(S_one_sided)));
shading interp;
colormap('hot');
title('Spectrogram: speech + noise');
xlabel('Time (s)');
ylabel('Frequency (Hz)');

figure
S_one_sided=max(STFT(1:length(F)/2,:),t_epsilon); %keep only the positive frequency
pcolor(T,F(1:end/2),10*log10(abs(S_one_sided)));
shading interp;
colormap('hot');
title('Spectrogram: speech only');
xlabel('Time (s)');
ylabel('Frequency (Hz)');



%-----------------    Listen results   -----------------------------------


fprintf('OK\n');
fprintf('Write anti_vuvuzela.wa:');
audiowrite('anti_vuvuzela.wav', output_signal,Fe);
fprintf('OK\n');
```

Listarea A.4: Cod Matlab – Codul pentru realizarea anularii de zgomot cu scadere spectrala

Bibliografie

[1] Suhaib Ahmed, Mudasir Bashir și Ashish Suri. „Low Pass FIR Filter Design and Analysis Using Hamming, Blackman and Kaiser Windows“. În: *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* 3 (2014), pag. 8676–8683.

[2] Ruth Bentler și Li-Kuei Chiou. „Digital Noise Reduction: An Overview“. În: *Trends in Amplification* 10.2 (2006), pag. 67–82.

[3] Dariusz Bismor, Krysztof Czyz și Zbigniew Ogonowski. „Review and Comparison of Variable Step-Size LMS Algorithms“. În: *International Journal of Acoustics and Vibration* 21.1 (2016), pag. 24–39.

[4] S Boll. „Suppression of acoustic noise in speech using spectral subtraction“. În: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 27.2 (1979), pag. 130–120.

[5] J. C. Burgess. „Active adaptive sound control in a duct: A computer simulation“. În: *The Journal of the Acoustical Society of America* 70.3 (1981), pag. 715–726.

[6] Huijun Ding și alții. „A spectral filtering method based on hybrid wiener filters for speech enhancement“. În: *Speech Communication* 51.3 (2009), pag. 259–267.

[7] Yariv Ephraim și David Malah. „Speech Enhancement Using a Minimum Mean-Square Error Short-Time Spectral Amplitude Estimator“. În: *IEEE Transactions on Acoustics, speech,and signal processing* 32.6 (1984), pag. 1109–1121.

[8] D. Guicking. „ On the invention of active noise control by Paul Lueg“. În: *The Journal of the Acoustical Society of America* 87.5 (1990).

[9] G. Ince, K. Nakadai și T. et al Rodemann. „Ego noise cancellation of a robot using missing feature masks“. În: *Applied Intelligence (2011)* 34.3 (2011), pag. 360–371.

[10] R. Kazemi și alții. „Detection and extraction of periodic noises in audio and biomedical signals using Kalman filter“. În: *Signal Processing* 88 (2008), pag. 2114–2121.

[11] Heinrich Löllmann și alții. „Microphone Array Signal Processing for Robot Audition“. În: *Hands-free Speech Communications and Microphone Arrays (HSCMA)*. Vol. 4. IEEE. 2017, pag. 51–55.

[12] Massimo. *What is the cut-off frequency of a moving-average filter*. URL: https://dsp.stackexchange.com/questions/9966/what-is-the-cut-off-frequency-of-a-moving-average-filter/14648#14648.

[13] Dubravko Miljkovic. „Active noise control: From analog to digital — Last 80 years“. În: *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics*. MIPRO. 2016, pag. 1358–1363.

[14] R. P. Narwaria și M. Kumar. „Comparison of FIR Filter Using Different Window Functions“. În: *International Research Journal of Engineering and Technology (IRJET)* 05.10 (2018), pag. 634–637.

[15] Kyle Poore și alții. „Single- and Multi-channel Whistle Recognition with NAO Robots“. În: *Lecture Notes in Artificial Intelligence*. Vol. 8992. Springer, Cham. 2015, pag. 245–257.

[16] Anam Rafique și Syed Sohail Ahmen. „Performance Analysis of a Series of Adaptive Filters in Non-Stationary Environment for Noise Cancelling Setup“. În: *International Journal of Electronics and Communication Engineering* 7.2 (2013), pag. 149–152.

[17] Aldebaran Robotics. *NAOqi Framework*. URL: http://doc.aldebaran.com/1-14/dev/naoqi/index.html. accessed 13.06.2022.

[18] Aldebaran Robotics. *SDKs*. URL: http://doc.aldebaran.com/1-14/dev/sdk.html#sdk-index. accessed 10.06.2022.

[19] SoftBank Robotics. *Meet Nao*. URL: https://www.softbankrobotics.com/emea/en/support/nao-6/1-meet-nao. (accessed: 7.06.2022).

[20] SoftBank Robotics. *NAOqi APIs*. URL: https://developer.softbankrobotics.com/nao6/naoqi-developer-guide/naoqi-apis. accessed 08.06.2022.

[21] SoftBank Robotics. *NAOqi OS - Getting started*. URL: http://doc.aldebaran.com/2-1/dev/tools/opennao.html. accessed 05.06.2022.

[22] Generation Robots. *Programmable Humanoid Robot NAO v6*. URL: https://www.generationrobots.com/en/403100-programmable-humanoid-robot-nao-v6.html. accessed: 07.06.2022.

[23] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub, 1997.

[24] Choqueuse Vincent. *Vuvuzela sound denoising algorithm*. URL: https://www.mathworks.com/matlabcentral/fileexchange/27912-vuvuzela-sound-denoising-algorithm.

[25] N. Weiner. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. Cambridge : Technology Press of the Massachusetts Institute of Technology ; New York : J. Wiley & Sons ; London : Chapman & Hall, 1949.

[26] Mark Weiss și Ernest Aschkenasy. *Automatic Detection and Enhancement of Speech Signals*. Grifiss Air Force Base, Digital Noise Reduction / Bentler, Chiou 81 NY: Rome Air Development Cente 1975. Report No. RADC-TR-75-77.

[27] William Williams, Mark Brown și Alfred Hero. „Uncertainty, information, and time-frequency distributions“. În: *Proceedings Volume 1566, Advanced Signal Processing Algorithms, Architectures, and Implemenations II* 1566 (1991), pag. 144–156.