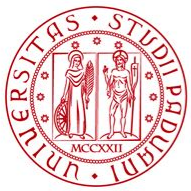

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Elettronica

ALGORITMI ADATTIVI PER LA CANCELLAZIONE ATTIVA DEL RUMORE

Relatore:

Prof. Mattia Zorzi

Laureando:

Nicola Viale

Mat. 2032405

Data di Laurea 17/07/24

Anno Accademico 2023/2024

Indice

Sommario

Il presente lavoro si propone di analizzare l'utilizzo di filtri adattivi per l'applicazione di tecniche di cancellazione attiva del rumore per le cuffiette bluetooth. Viene inizialmente fornita un'introduzione esaustiva alle tematiche trattate, delineando il contesto e l'importanza dell'argomento preso in esame. In seguito si propone una breve trattazione teorica volta a facilitare la comprensione dell'analisi degli algoritmi ricorsivi PEM e LMS, utilizzati per l'aggiornamento dei coefficienti dei filtri implementati. Viene infine descritta l'implementazione pratica di tali algoritmi in ambiente MATLAB, seguita da un confronto delle prestazioni ottenute.

Capitolo 1

Introduzione

1.1 Contestualizzazione del problema

La cancellazione del rumore è una tecnologia progettata per ridurre o eliminare i suoni indesiderati, noti come "rumore", da un ambiente audio. A causa del costante incremento dell' inquinamento acustico, ovvero dell'eccesso di suono indesiderato che può interferire con le attività umane, la ricerca di una soluzione efficiente a questo problema ha assunto un'importanza via via crescente e sta riscontrando ai giorni nostri un impiego sempre maggiore. In particolare l'inquinamento acustico viene quantificato rispetto all'intensità delle onde di pressione sonore e misurato usando una scala in *decibel* (dB). Viene riportata di seguito una tabella esemplificativa. È interessante notare come la soglia del dolore

dB SPL	Sorgente
140	Colpo di pistola a 1 m
130	Soglia del dolore
125	Aereo al decollo a 50 m
120	Sirena
110	Motosega a 1 m
100	Discoteca, concerto rock vicino al palco
90	Urlo
80	Camion pesante a 1 m
70	Aspirapolvere a 1 m
60	Ufficio rumoroso, radio, conversazione
50	Ambiente domestico
40	Quartiere abitato, di notte
30	Sussurri a 1 m
20	Respiro umano a 20 cm

Figura 1.1: Percezione umana della scala in dB.

umano, posta a 130dB, sia molto maggiore rispetto alle soglie presenti nelle normative vigenti. Una delle normative europee di riferimento per la gestione del rumore ambientale è la direttiva *2002/49/CE*, questa richiede agli stati membri non solo di adottare delle misure per ridurlo, ma anche di mapparlo in modo da tenerlo il più possibile monitorato

su larga scala. Questa direttiva stabilisce inoltre delle soglie di esposizione per alcune delle fonti più comuni di rumore, tra queste ci sono le attività industriali ed il traffico stradale, aereo e ferroviario. Prendendo l'esempio delle aree residenziali nelle ore diurne viene raccomandata un'esposizione non superiore a 55-65 dB, per le aree industriali invece si raccomanda un limite di 70-75 dB. Risulta ora evidente come la cancellazione del rumore stia ricoprendo un ruolo sempre più rilevante in una vasta gamma di contesti, tra questi spiccano per importanza:

- Industria automotive
- Auricolari bluetooth
- Edilizia
- Aviazione
- Apparecchiature mediche.

Il fatto che queste applicazioni siano tra le più svariate per ambito di competenza sottolineano che da una parte un'evoluzione così rapida sia dovuta all'esposizione delle persone ad ambienti sempre più rumorosi, al punto da poter nuocere alla propria salute; dall'altra alla sempre maggiore richiesta di comfort e di una migliore esperienza acustica, la quale ad oggi muove la macchina economica globale. Tra gli auricolari bluetooth ad oggi più venduti che implementano una tecnologia di *Active Noise Cancellation* (ANC), troviamo sicuramente le *AirPods Pro* di *Apple Inc.* . Queste, ad esempio, implementano una tecnologia di cancellazione che utilizza due microfoni esterni e un microfono interno per rilevare il suono ambientale e neutralizzarlo con onde sonore in opposizione di fase. L'effetto ottenuto è una riduzione del rumore fino a 20-25 decibel, creando un'esperienza sonora più immersiva e priva di distrazioni per i propri utenti. D'altra parte, le cuffie *Sony WH-1000XM4* sono anch'esse riconosciute per la loro efficacia, queste utilizzano la tecnologia proprietaria *Adaptive Sound Control*, la quale monitora costantemente l'ambiente circostante e regola automaticamente le impostazioni di cancellazione in base alla posizione e alle attività dell'utente. Queste cuffie offrono una riduzione del rumore fino a 30 decibel su una vasta gamma di frequenze, garantendo un'esperienza acustica superiore anche alla precedente. Tuttavia, nonostante i notevoli progressi tecnologici ottenuti da prodotti di alta qualità come questi ultimi e l'elevata attenuazione che si è riusciti a raggiungere, rimangono ad oggi da perfezionare alcuni aspetti legati alla distorsione del segnale, difatti queste tecniche non presentano una cancellazione uniforme in banda. In conclusione, la cancellazione del rumore emerge come una risposta critica all'aumento dell'inquinamento acustico, un fenomeno che minaccia in primis il benessere umano e la qualità della vita, è

quindi per questo motivo che si stanno investendo tempo e risorse e si stanno esplorando tecniche sempre più all'avanguardia per realizzare una cancellazione con attenuazione più elevata possibile e con minima distorsione.

1.2 Scopo della ricerca

Il presente studio si propone di esplorare e implementare un sistema per la cancellazione attiva del rumore nell'ambito delle cuffiette bluetooth. L'obiettivo primario è sviluppare un codice in *MATLAB* che realizzi un filtro adattivo, i cui coefficienti vengano aggiornati in tempo reale mediante l'algoritmo noto come *recursive prediction error method* (*RPEM*), la cui implementazione dettagliata sarà presentata in seguito. Tale filtro è essenziale per la previsione e la riduzione del rumore futuro. Il fine ultimo è realizzare un codice efficiente e compatto, minimizzando l'utilizzo di risorse di memoria e potenza di calcolo. Ciò consentirebbe una migliore integrazione del sistema di cancellazione del rumore all'interno delle cuffiette bluetooth, garantendo prestazioni audio di alta qualità e un'esperienza d'ascolto ottimale per gli utenti. Un aspetto cruciale di questa ricerca è il confronto tra due metodi di aggiornamento dei coefficienti del filtro di predizione del rumore: l'algoritmo *least mean squares* (*LMS*), comunemente utilizzato per la sua minore complessità computazionale, e l'*RPEM* proposto in questo studio, il quale, sebbene più complesso, si prevede possa garantire una migliore precisione nella predizione del rumore. Una volta completate la realizzazione e la verifica della funzionalità del codice, verrà condotto un confronto tra questi due approcci al fine di determinare se la maggiore complessità computazionale dell'*RPEM* sia realmente giustificata dalle sue prestazioni superiori nella cancellazione del rumore. Si intende inoltre fornire un'idea di massima riguardo l'estensione dei concetti proposti all'implementazione in un microcontrollore, in modo da valutare non solo l'efficienza delle soluzioni proposte, ma anche la praticità.

1.3 Concetti fondamentali sulla cancellazione passiva del rumore

La cancellazione passiva del rumore è una tecnica che si basa sull'impiego di barriere antirumore, tra gli ambiti di utilizzo più comuni ci sono sorgenti quali il traffico stradale e ferroviario. Tra le caratteristiche principali di una barriera antirumore possiamo trovare:

- Assorbimento acustico: le barriere antirumore sono progettate per assorbire parte dell'energia sonora incidente. Questo avviene attraverso l'utilizzo di materiali porosi

ad alta densità, come la fibra di vetro o la lana minerale, che sono noti per le loro proprietà di assorbimento acustico. Lo spessore dei materiali può variare in base alla frequenza del rumore da attenuare, gli spessori tipici sono compresi tra 5 e 10 cm per una buona performance a frequenze medie;

- Isolamento acustico: una caratteristica chiave delle barriere antirumore è il loro effetto di isolamento acustico. Questo implica l'utilizzo di materiali densi e spessi, come il cemento armato o il metallo, che sono in grado di bloccare efficacemente la trasmissione del suono. Lo spessore in particolare può influenzare significativamente le prestazioni di isolamento acustico della barriera;

- Riflessione acustica: per riflettere il suono indesiderato lontano dall'area di interesse, contribuendo così a ridurre l'esposizione al rumore, come il vetro o l'alluminio possono essere utilizzati per massimizzare l'effetto. La forma e l'inclinazione della barriera in particolare dovranno essere ottimizzate per indirizzare il suono riflesso lontano dal punto di interesse;

- Diffrazione: quando il suono incontra un oggetto, come ad esempio una barriera antirumore, parte dell'energia sonora viene deviata attorno all'ostacolo anziché essere riflesso indietro. Le barriere antirumore possono essere progettate con aperture o fessure strategicamente posizionate per sfruttare la diffrazione e ridurre l'impatto del rumore.

Vediamo qui sotto riportato in (Fig. 1.2) un esempio del principio di funzionamento.

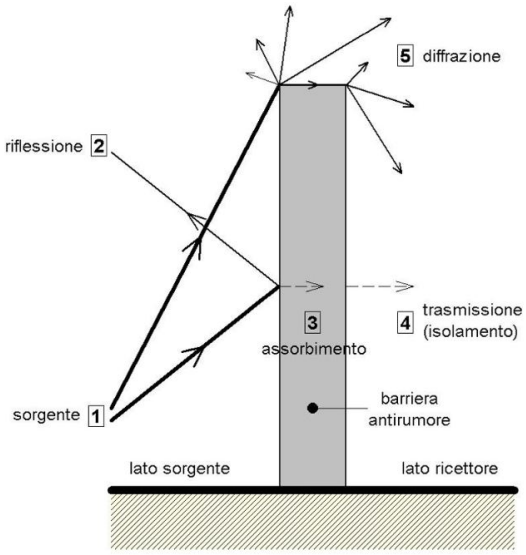

Figura 1.2: Principali fenomeni delle onde di pressione sonora.

Portando l'esempio delle barriere acustiche che spesso si trovano lungo le autostrade, queste spesso vengono realizzate in vetroresina e basano il proprio funzionamento sulla riflessione acustica, altri motivi per i quali questo è uno dei materiali più utilizzati sono la resistenza meccanica, la leggerezza, la resistenza agli agenti atmosferici e la trasparenza che assicura un'esperienza di guida più gradevole. Uno dei modelli più utilizzati e semplici che descrivono il funzionamento di una barriera acustica porosa prevede che la loro frequenza di risonanza intrinseca sia fissata ad un valore pari a:

$$f_R = \frac{c}{4 \cdot d},$$

dove rispettivamente:

- c è la velocità di propagazione del suono in aria;
- d è lo spessore del materiale poroso.

Da un'analisi al primo ordine è facile dedurre che per una barriera di spessore fissato il rumore ambientale verrà attenuato di un valore accettabile solamente per frequenze sufficientemente elevate, in particolare molto maggiori della frequenza di risonanza intrinseca della barriera. É proprio per questo motivo e per l'evidente impossibilità di intervenire sulle dimensioni che vengono ora introdotti metodi di cancellazione attiva del rumore.

1.4 Concetti fondamentali sulla cancellazione attiva del rumore

Il crescente impiego di attrezzature industriali, tra le quali alcuni esempi: motori, ventilatori, trasformatori e compressori, ha portato ad una crescente consapevolezza dei problemi legati al rumore acustico nell'ambiente circostante. Tradizionalmente, come visto in precedenza, per attenuare il rumore indesiderato, sono state utilizzate tecniche di cancellazione passiva come barriere e silenziatori. Tuttavia, queste soluzioni si rivelano spesso ingombranti, costose ed inefficienti alle basse frequenze. L'approccio innovativo dell' *Active Noise Control* (ANC) coinvolge sistemi elettroacustici o elettromeccanici che annullano il rumore primario mediante il principio della sovrapposizione. Tale metodo permette di generare un'onda acustica di uguale ampiezza ed in controfase rispetto al rumore indesiderato, ne consegue così la cancellazione di entrambi i rumori attraverso il fenomeno che prende il nome di *interferenza distruttiva*. É interessante notare che l'energia presente in ambedue le onde sonore non viene distrutta, ma si trasforma in energia meccanica, la quale fa vibrare le particelle del mezzo trasmissivo, ed in energia termica. L'ANC, come

citato nel capitolo precedente, si dimostra particolarmente efficace alle basse frequenze, dove le soluzioni passive possono risultare di dimensioni non consone o eccessivamente costose. Il motivo principale che ne determina l'efficienza alle basse frequenze è dovuto alla possibilità di creare un modello di predizione del rumore residuo, che nell'esempio degli auricolari bluetooth è il rumore residuo all'interno della scocca, che abbia una altissima frequenza di campionamento del rumore esterno rispetto a quella del rumore da predire. Per la ricostruzione efficiente di un segnale di rumore acquisito in ingresso è infatti necessario avere una frequenza di campionamento che sia adeguatamente maggiore della frequenza massima del segnale; in particolare come afferma il *Teorema di Shannon-Nyquist*, per poter ottenere una ricostruzione totale del segnale acquisito dovrà essere rispettata la seguente condizione:

$$f_C >= \ 2 \cdot f_S,$$

in cui:

- fc è la frequenza di campionamento
- fs è la frequenza massima del segnale da campionare.

Nel caso in cui questa condizione non venga soddisfatta si può incorrere nel fenomeno dell'aliasing, ovvero della comparsa di componenti spuria nello spettro del segnale ricostruito come mostrato nella figura sottostante.

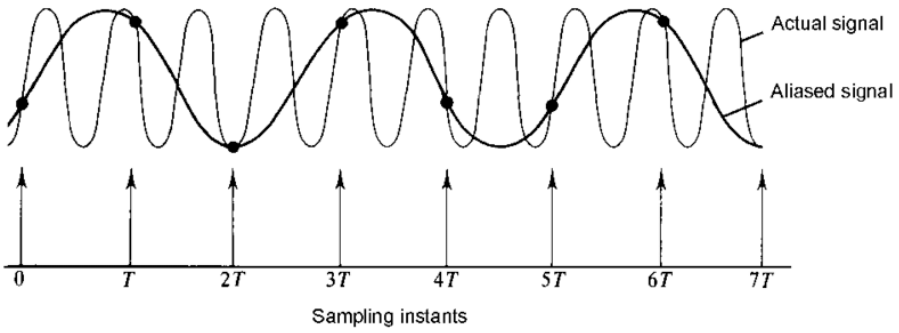

Figura 1.3: Fenomeno dell'aliasing dovuto al sottocampionamento.

In figura 1.3 l'asse orizzontale rappresenta gli istanti di campionamento, l'onda sottile rappresenta il segnale che si vuole campionare, l'onda in grassetto invece rappresenta una componente a bassa frequenza non voluta, ma comunque rilevata. È ora evidente come l'effettiva realizzabilità di un filtro predittivo si appoggi alla necessità di un sistema fisico che presenti una frequenza di acquisizione di valore piuttosto elevato. È ora importante

notare che i rumori più comuni citati in precedenza, come quelli di natura meccanica, non siano propriamente a frequenze basse o comunque note. Tuttavia, nella trattazione seguente, incentrata sullo studio dell'applicazione di questi concetti alle cuffiette bluetooth, queste ipotesi possono essere rilassate in quanto in primis le frequenze udibili dall'orecchio umano sono limitate ad una banda intorno ai 20 KHz, ed inoltre la scocca delle cuffiette in sé può essere modellata come un filtro passa basso, che attenua quindi le componenti del rumore ad alta frequenza in modo analogo ad un filtro di cancellazione passiva. Nella trattazione seguente, oltre alla creazione di un algoritmo che sia in grado di prevedere il rumore futuro e adattarsi ad esso in modo da poterlo rimuovere, ci si focalizza anche sulle caratteristiche del rumore attuale, passato e sull'azione filtrante realizzata intrinsecamente dalla scocca. Per potersi adattare dinamicamente alle variazioni delle sorgenti di rumore, gli algoritmi adattivi impiegati nell'ANC regolano i propri coefficienti attraverso un algoritmo ricorsivo, consentendo una rapida convergenza ed una maggiore affidabilità ed attenuazione. Tra gli algoritmi più comuni che realizzano l'aggiornamento dei coefficienti in tempo reale ci sono l'algoritmo *LMS*, basato sullo scarto quadratico medio, e la sua variante normalizzata rispetto alla potenza istantanea del segnale di ingresso, l'algoritmo *NMLS*. Altri approcci più sofisticati, come tecniche di machine learning nell'ANC, stanno pian piano prendendo piede, queste ultime possono essere addestrate su ampi set di dati per identificare pattern e correlazioni nel rumore ambientale, consentendo una cancellazione più efficace e mirata, in particolare possono essere utilizzate per fornire pesi differenti per la variazione dei coefficienti basati sul contesto del rumore identificato.

Capitolo 2

Revisione della letteratura

2.1 Identificazione di sistemi Lineari Tempo Varianti

Il modello di un sistema dinamico lineare tempo variante (LTV) viene comunemente definito come segue:

$$M(\theta_t) : y(t) = F_{\theta_t}(z)u(t) + G_{\theta_t}(z)e(t).$$

Il modello $M(\theta_t)$ varia in funzione del parametro θ_t ed è descritto dalla formula sopra riportata in cui l'uscita $y(t)$ è uguale all'ingresso $u(t)$ moltiplicato per una funzione di trasferimento $F_{\theta_t}(z)$, che dipende da θ_t, posta a tempo discreto poiché l'acquisizione che permette l'elaborazione dei dati non può avvenire a tempo continuo, sommato ad un termine di errore denominato $e(t)$, moltiplicato per la relativa funzione di trasferimento $G_{\theta_t}(z)$, che dipende da θ_t. La principale caratteristica che identifica questa classe di modelli è la variabilità di $F_{\theta_t}(z)$ e $G_{\theta_t}(z)$ al trascorrere del tempo. In particolare, in questa trattazione si considera l'ipotesi per la quale i parametri cambino lentamente rispetto alla variabile temporale, sarebbe altrimenti impossibile, se il modello mutasse troppo velocemente, predire in modo accurato i coefficienti come proposto nei capitoli a seguire. In particolar modo la struttura del modello adottato in questo documento è di tipo *Auto-Regressive with eXogenous inputs* (ARX). I modelli ARX, sono modelli basati sull'idea che l'uscita di un sistema dipenda linearmente dai suoi valori passati, da ingressi esterni e da un termine di rumore. Nello specifico la formulazione matematica che li definisce lega l'uscita del sistema ai suoi valori passati, ad ingressi esterni, e ad un termine di rumore, mediante dei coefficienti che non siano costanti ma lentamente variabili nel tempo. La struttura che definisce queste classi di modelli è la seguente:

$$y(t) = a_1(t)y(t-1) + .. + a_{n_A}(t)y(t-n_A) + b_1(t)u(t-1) + .. + b_{n_B}(t)u(t-n_B) + e(t). \quad (2.1)$$

$$= \phi(t-1)^T \hat{\theta}(t-1) + e(t) \tag{2.2}$$

dove i termini $a_1(t), ..., a_{n_A}(t)$ sono i coefficienti auto regressivi, mentre i termini $b_1(t), ..., b_{n_B}(t)$ sono i coefficienti degli ingressi, nA è l'ordine della regressione ed nB è il numero degli ingressi. Tutti questi termini caratterizzano θ_t.

2.2 Derivazione dell'algoritmo RPEM

Viene ora presentata una breve trattazione teorica finalizzata a fornire l'intuizione per ricavare le formule per l'aggiornamento ricorsivo dei coefficienti del filtro adattivo presentato nella sezione precedente. Partendo dalla formula di un generico modello ARX all'istante t disponiamo delle seguenti informazioni:

$$y_t = [y(1), ..., y(t)], u_t = [u(1), ..., u(t)] \tag{2.3}$$

che rappresentano rispettivamente i vettori riga delle uscite passate e degli ingressi passati forniti al modello. Si definiscono ora, al fine di utilizzare una funzione più compatta:

- $\phi(t)$: vettore colonna contenente i valori passati delle n_A uscite e degli n_B ingressi disposti come segue:

$$\phi(t) = [-y(t-1), ..., -y(t-nA), u(t-1), ..., u(t-nB)]^T. \tag{2.4}$$

- $\theta(t)$: vettore colonna contenente i coefficienti dei valori passati, disposti in modo corrispondente rispetto a quelli di $\phi(t)$, come segue:

$$\theta(t) = [a_1, ..., a_{nA}, b_0, ..., b_{nB-1}]^T. \tag{2.5}$$

Volendo ora ottenere una stima dei coefficienti del filtro che sia la più affidabile possibile, si vuole di fatto minimizzare lo scarto quadratico medio dell'errore di predizione definito come:

$$\epsilon_\theta(s) = y(s) - \phi(s)^T \theta.$$

Di conseguenza la stima ottima dei coefficienti si ottiene minimizzando la funzione:

$$\frac{1}{t} \sum_{s=1}^{t} (\epsilon_\theta(s))^2 = \frac{1}{t} \sum_{s=1}^{t} (y(s) - \phi(s)^T \theta)^2.$$

Tuttavia, seguendo questo tipo di approccio, viene dato lo stesso peso ai campioni più vecchi, che potrebbero rappresentare un modello passato anche molto differente, rispetto a quelli attuali, i quali, rispettando le ipotesi fatte in precedenza di un modello lentamente variabile nel tempo rispetto all'aggiornamento dei coefficienti, dovrebbero disporre di un peso più significativo. Proprio per questo motivo viene ora introdotta una funzione peso $\beta(t,s) \geq 0$, in cui l'ordine delle operazioni nelle formule proposte è posto in modo da avere pesi più alti associati a valori acquisiti più recentemente. La funzione da minimizzare per ricavare la stima ottima dei coefficienti voluti diventa ora:

$$\hat{\theta}_t = \underset{\theta}{argmin} \quad ((\sum_{s=1}^{t} \beta(t,s)) \quad ^{-1} \sum_{s=1}^{t} \beta(t,s)(y(s) - \phi(s)^T\theta)^2). \tag{2.6}$$

Al fine di abbreviare maggiormente la formula, passaggio necessario al fine di poter applicare in seguito alcune proprietà fondamentali delle matrici per poter continuare la dimostrazione, definiamo ora:

- R_t che è definita come:

$$R_t^{-1} = \begin{bmatrix} \beta(t,1) & 0 & ... & 0 \\ 0 & \beta(t,2) & ... & ... \\ ... & ... & ... & 0 \\ 0 & ... & 0 & \beta(t,t) \end{bmatrix}$$

- Φ_t che è definita come:

$$\Phi_t = \begin{bmatrix} \phi(1)^T \\ \phi(2)^T \\ ... \\ \phi(t)^T \end{bmatrix}.$$

Idealmente il vettore $\theta(t)$ che minimizza (2.6) soddisfa la condizione:

$$y_t = \Phi_t\theta$$

da cui, moltiplicando ambo i membri per ${\Phi_t}^T {R_t}^{-1}$ e utilizzando le proprietà delle matrici per isolare theta si ottiene:

$$\hat{\theta}_t = ({\Phi_t}^T {R_t}^{-1}\Phi_t)^{-1}{\Phi_t}^T {R_t}^{-1} y_t.$$

Si può dimostrare che tale $\hat{\theta}_t$ è la soluzione di (2.6). É importante notare che le dimensioni delle matrici contenute nella formula crescono linearmente col numero di campioni, ciò

non è in accordo con le necessità di un utilizzo di memoria finito e di una complessità computazionale che non aumenti in funzione del tempo. Diviene ora necessario adottare delle ipotesi aggiuntive in modo da poter ricavare un algoritmo ricorsivo che rispetti questi requisiti e che permetta di calcolare il vettore $\hat{\theta}_t$ in funzione dei valori passati, ovvero in maniera ricorsiva. Si sceglie allora una particolare struttura per la funzione β dei pesi:

$$\beta(t,s) = \begin{cases} \prod_{k=s+1}^{t} \lambda(k) & 1 \leq t \\ 1 & s = t \end{cases}.$$

Nella formula soprastante viene introdotto il coefficiente d'oblio λ, un valore tale che $0 \leq \lambda \leq 1$. Questo coefficiente è stato introdotto in modo da poter definire β in funzione della sua versione all'istante di campionamento precedente:

$$\beta(t,s) = \begin{cases} \lambda(t)\beta(t-1,s) & 1 \leq t \\ 1 & s = t. \end{cases} \tag{2.7}$$

Si definisce una matrice S_t, con lo scopo di alleggerire ulteriormente la notazione:

$$S_t = \sum_{s=1}^{t} \beta(t,s)\phi(s)\phi(s)^T. \tag{2.8}$$

É ora possibile riscrivere S_t rispetto a quella dell'istante precedente. Sostituendo nuovamente quest'ultima nella precedente formula di $\hat{\theta}_t$, si ottengono dunque le formule ricorsive di aggiornamento dei coefficienti ottimali stimati:

$$\begin{cases} S_t = \lambda(t)S_{t-1} + \phi(t)\phi(t)^T \\ \hat{\theta}_t = \hat{\theta_{t-1}} + S_{t-1}\phi(t)[y(t) - \phi(t)^T\hat{\theta_{t-1}}]. \end{cases} \tag{2.9}$$

È infine possibile notare come la memoria e la capacità computazionale richiesta non aumentino più in funzione della variabile temporale, ma rimangano costanti.

2.3 Estensione all'algoritmo LMS

In questa sezione, la trattazione teorica dell'algoritmo *RPEM* presentata nel capitolo precedente viene brevemente ripresa ed estesa all'algoritmo *LMS*, in modo da disporre di una panoramica che permetta di comprendere l'analisi che viene proposta in seguito. A

partire dalle ipotesi effettuate nella sezione precedente si era ottenuto il seguente risultato:

$$\hat{\theta}_t = \underset{\theta}{argmin} \quad ((\sum_{s=1}^{t} \beta(t,s)) \quad ^{-1} \sum_{s=1}^{t} \beta(t,s)(y(s) - \phi(s)^T\theta)^2).$$

A partire dalla formula soprastante possono essere definiti:

$$\gamma(t) = (\sum_{s=1}^{t} \beta(t,s))^{-1} \tag{2.10}$$

$$V_t(\theta) = \gamma(t) \sum_{s=1}^{t} \beta(t,s)(y(s) - \phi(s)^T\theta)^2).$$

L'idea ora è quella di ricavare il vettore dei nuovi coefficienti a partire dal vettore di quelli trovati al passo di campionamento precedente, per farlo viene utilizzato il passo dell'*Algoritmo del gradiente*, definito come segue:

$$x_f = x_i - \mu \nabla f(x_i) \tag{2.11}$$

in cui:

- x_f è il vettore che rappresenta il punto finale
- x_i è il vettore che rappresenta il punto iniziale
- $\nabla f(x_i)$ è il gradiente della funzione f calcolata nel punto iniziale
- μ è un parametro scalare che regola la velocità di apprendimento.

Ci si vuole quindi spostare in modo graduale lungo la direzione del gradiente, in modo da ridurre progressivamente il valore della funzione fino a raggiungere un minimo locale o globale. Questo si traduce nell'aggiornare iterativamente i parametri del modello, muovendosi nella direzione opposta al gradiente, il quale viene continuamente aggiornato ad ogni passo iterativo dato che, essendo il modello tempo variante, il minimo della funzione di errore assume posizioni differenti ad ogni ciclo. Per garantire un corretto avanzamento, si utilizza un tasso di apprendimento μ che regola la lunghezza del passo lungo il gradiente. Se il tasso di apprendimento fosse troppo elevato si rischierebbe di oltrepassare il punto di minimo desiderato, mentre se fosse troppo basso potrebbe non riuscire ad avvicinarsi abbastanza prima del nuovo ciclo di lettura e aggiornamento del minimo. Trovare il giusto equilibrio è quindi fondamentale per massimizzare l'efficienza complessiva dell'algoritmo, in particolare viene introdotto un coefficiente μ, che ne regola

la velocità, ed è un valore compreso tra 0 e 1. La costruzione della funzione di cui si vuole trovare il minimo si può immaginare come una forma tridimensionale ricavata dalla storia passata degli ingressi forniti al modello ed alle corrispettive uscite. Il punto di minimo di questa forma può quindi essere pensato come un punto in continuo movimento che si sta cercando di inseguire con un vettore il cui verso è diretto nel verso di minima crescita ed il cui modulo dipende dal coefficiente μ, che viene sommato vettorialmente una volta per passo di campionamento. É ora possibile minimizzare la funzione presentata in precedenza con l'algoritmo descritto, il risultato è il seguente:

$$\hat{\theta}_t = \hat{\theta}_{t-1} - \mu V_t^{'}(\hat{\theta}_{t-1})$$

dove con $V_t^{'}(\hat{\theta}_{t-1})$ viene indicato il gradiente di $V_t(\theta)$ per $\theta = \hat{\theta}_t$. Imponendo ora le condizioni seguenti:

$$V_{t-1}{}^{'}(\hat{\theta}_{t-1}) = 0$$

$$\beta(t,s) = \begin{cases} \lambda(t)\beta(t-1,s) & 1 \leq t \\ 1 & s = t \end{cases}.$$

e assumendo che i coefficienti del passo precedente $\hat{\theta}_{t-1}$ minimizzano la funzione al passo precedente $V_{t-1}^{'}(\theta)$. Svolgendo la derivata, imponendola pari a zero ed applicando le definizioni delle variabili definite in precedenza si giunge alla seguente formula finale di aggiornamento dei coefficienti:

$$\hat{\theta}_t = \hat{\theta}_{t-1} + 2\mu\gamma(t)\phi(t)(y(t) - \phi(t)^T\hat{\theta}_{t-1}). \tag{2.12}$$

Ciò che quindi rende questo algoritmo meno accurato rispetto al precedente è l'utilizzo dell'ipotesi che $\hat{\theta}_{t-1}$ minimizza $V_{t-1}(\theta)$. D'altro canto, l'algoritmo LMS occupa un quantitativo minore di memoria e richiede una minore capacità computazionale, ma necessita anche di un parametro μ che deve essere definito a priori e può non assicurare la convergenza se scelto erroneamente.

Capitolo 3

Metodologia di implementazione

3.1 Descrizione del modello di filtro adattivo basato su RPEM

Ora che la trattazione teorica è completa e si ha una panoramica migliore di quali algoritmi sono stati implementati si procede alla descrizione dello schema a blocchi e del principio di funzionamento della cancellazione attiva del rumore nell'ambito delle cuffiette bluetooth. Il prototipo di auricolare che viene preso in esame in questa trattazione è dotato di due microfoni, rispettivamente uno nella parte interna della scocca, uno nella parte esterna e di un altoparlante. Viene di seguito presentata la legenda dei nomi scelti per le variabili che rappresentano i segnali trattati:

- r è il rumore esterno alla scocca della cuffietta
- s è il suono/musica che si desidera riprodurre ed ascoltare
- d è il disturbo interno percepito, ed è quindi il rumore esterno filtrato dalla scocca, la quale agisce da filtro di cancellazione passiva di tipo passa basso, come descritto nei capitoli precedenti, ed arriva all'interno dell'auricolare, descrive quindi ciò che si vuole rimuovere dato che viene percepito in aggiunta alla musica
- $d_{stimato}$ è il disturbo generato dal filtro predittivo, idealmente equivale alla versione in opposizione di fase del disturbo da cancellare d, viene riprodotto dall'altoparlante insieme alla musica
- v è ciò che l'utente sente effettivamente se non viene attivata la cancellazione, di conseguenza è la sovrapposizione di musica e disturbo d

- y è ciò che l'utente sente effettivamente se viene attivata la cancellazione, ovvero se viene prodotto $d_{stimato}$, di conseguenza è la sovrapposizione di musica, disturbo e disturbo in controfase.

Nella figura sottostante vengono presentate le principali correlazioni che legano i segnali in gioco ed uno schema atto a fornire un' overview del ragionamento logico che viene adottato:

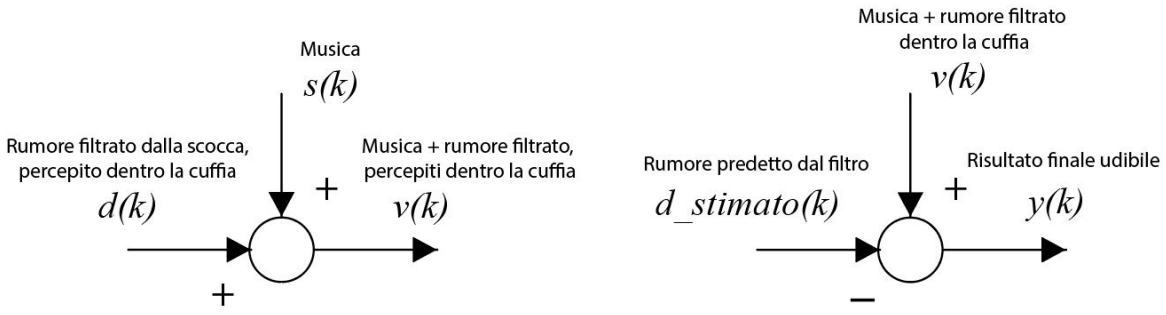

Figura 3.1: Relazioni principali tra i segnali.

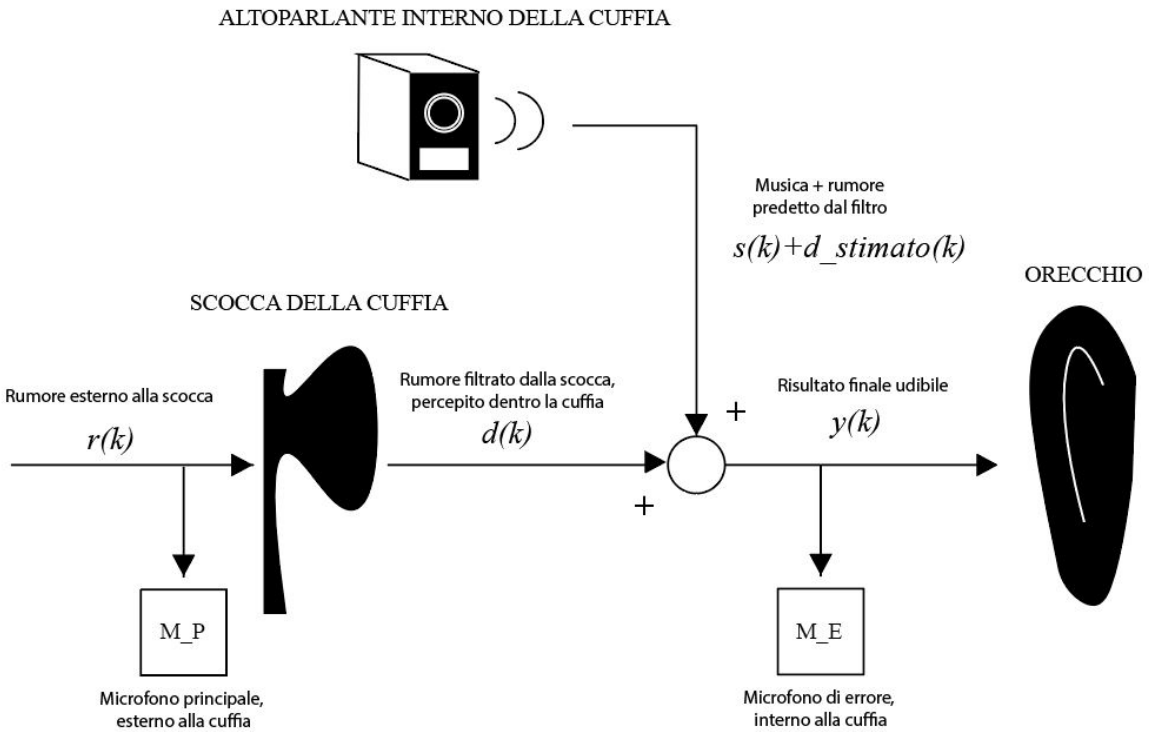

Figura 3.2: Schema visivo dei segnali.

Il sistema di cancellazione del rumore in cuffia proposto è costituito da diverse componenti, si veda (Fig. 3.2), tra queste il microfono principale M_P posizionato esternamente alla cuffia, il microfono di errore M_E, posizionato all'interno della cuffia vicino all'orecchio dell'utente ed il filtro ANC che elabora i segnali provenienti da entrambi i microfoni per stimare il rumore che raggiunge l'orecchio dell'utente all'istante di campionamento successivo. Un altoparlante interno emette quindi un suono uguale e opposto al rumore stimato, creando così un'onda di cancellazione che riduca il rumore percepito dall'utente. Il funzionamento del sistema segue il principio seguente: il microfono principale rileva il suono ambientale, mentre il microfono di errore rileva il suono che raggiunge l'orecchio dell'utente, comprendente il rumore esterno filtrato dalla scocca della cuffia $d(k)$, la musica $s(k)$, ed il rumore calcolato al passo precedente per eliminarlo $d_{stimato}(k)$. Il microcontrollore integrato nella cuffia elabora i segnali dei due microfoni per ricavarne il rumore filtrato $d(k)$, sapendo ciò che sta riproducendo, il rumore letto viene quindi processato per aggiornare i valori del filtro ed infine preparato per essere sommato al segnale del microfono principale all'istante successivo. In questo modo, l'algoritmo di predizione aggiorna i propri coefficienti e l'altoparlante interno al passo successivo emetterà un suono che ridurrà il rumore percepito dall'utente, consentendo un ascolto con un rumore attenuato. Di seguito viene ora presentato il diagramma a blocchi completo:

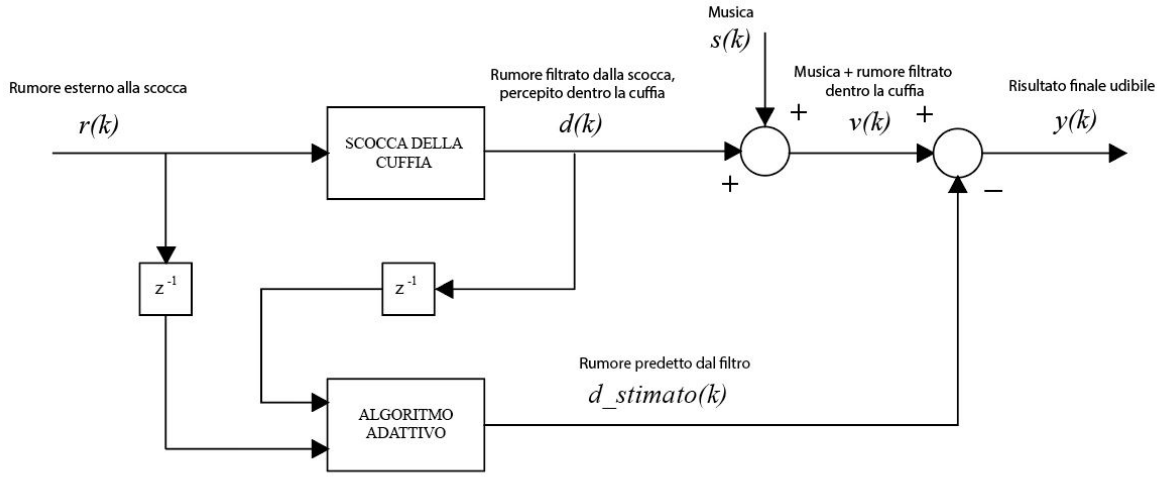

Figura 3.3: Schema a blocchi dei segnali.

Nello schema a blocchi soprastante si suppone di essere al passo di campionamento k, i blocchi raffiguranti z^{-1} indicano l'operazione di ritardo ad un passo. Il blocco del filtro adattivo ha in ingresso il rumore esterno ed interno alla cuffietta all'istante precedente e li utilizza per aggiornare i propri coefficienti e generare una predizione per il rumore interno

successivo, che in questo caso è quello attuale. Questo si somma poi al rumore interno attuale ed alla musica, generando il suono percepito dall'utente.

3.2 Progettazione ed implementazione in MATLAB

In questa nuova sezione viene ora presentata e commentata l'implementazione dell' algoritmo di controllo attivo del rumore utilizzando l'ambiente di sviluppo *Matlab* in modo da poter fornire al lettore una rapida guida alla realizzazione di un codice simile a quello proposto. Ogni sottosezione è dedicata all'approfondimento delle singole funzioni. Nello specifico si è scelto di riportare unicamente le funzioni più importanti al fine di collegare i concetti teorici presentati con la realizzazione pratica proposta. In particolare al termine di ogni sezione di codice viene riportata una rapida spiegazione di ciò che è stato effettuato. I risultati ottenuti verranno infine mostrati e commentati nel capitolo successivo.

3.2.1 Main

Il codice main è il cuore del processo implementativo del programma realizzato, esso illustra le scelte progettuali effettuate di architettura, valutazione delle prestazioni ed inizializzazione delle variabili. Il codice è impostato nel modo seguente:

- c'è una fase di *training* dove si ricercano nA, nB e λ, citati nelle (2.1) (2.7), da utilizzare nelle simulazioni successive al fine di ottenere una efficienza di cancellazione più elevata possibile per un dataset fissato utilizzando *RPEM*
- creazione di funzioni di calcolo dell'energia media dei segnali acustici, visualizzazione dei risultati ottenuti e di ascolto per la distinzione del rumore e della musica prima e dopo l'utilizzo del filtro
- c'è poi una fase di *validazione* dove si considera un nuovo dataset, con rumori differenti, al fine di testare l'efficacia dell'algoritmo ricorsivo RPEM in contesti differenti da quelli di *training*
- introduzione di un rumore nella lettura del microfono interno alle cuffie in modo da analizzarne la loro non idealità
- implementazione dell'algoritmo ricorsivo LMS e confronto finale delle prestazioni tra i due metodi proposti.

`main`

```
1     secondi = 3;
2     fs = 44100;
3     inizi_scale = [1, 1, 0.95];
4     passi_scale = [1, 1, 0.025];
5     lati_ipercubo = [12,12,3];
6     secondi_val = 19;
7     p = 0.03;
8     mu = 0.05;
```

Il codice inizia con la definizione di alcuni parametri fondamentali ai fini della simulazione, *secondi* rappresenta il tempo di riproduzione musicale che si desidera simulare, fs rappresenta la *Sampling frequency* o frequenza di campionamento. In questo progetto si è deciso di utilizzare dei file musicali *.mp3*, si è quindi indicata la frequenza standard di lavoro di un file di questo tipo, 44100 Hz . Si definiscono ora i valori iniziali del numero dei diversi coefficienti nA, nB e λ, che si vogliono usare per la simulazione dell'algoritmo RPEM. Tali valori sono riportati nella variabile *lati_ipercubo*. Il primo valore di questi tre vettori rappresenta quindi per quanti valori di nA viene testato l'algoritmo, il valore nella seconda posizione rappresenta quello di nB ed il terzo quello di λ. La variabile *secondi_val* rappresenta il tempo massimo simulabile in secondi di ogni audio, ovvero la durata dell'audio più breve usato nella simulazione. Viene poi definito p, ovvero la percentuale di errore che viene introdotta nella lettura del microfono interno ai fini di simulare la non idealità del microfono utilizzato in un contesto reale, in questo caso il valore viene impostato al 3% . Per concludere viene definita la variabile mu, la quale rappresenta il coefficiente μ citato nella (2.11).

```
9     campioni = secondi * fs;
10    s = leggi_mp3('musica.mp3',campioni);
11    s = s*4;
12    r = 4*leggi_rumore(campioni);
13    [d,rumore_extra] = filtra_rumore_filter(s,r,p);
```

Si prosegue con il calcolo del numero *campioni*, dei campioni che verranno presi in esame da ogni file audio, dato quindi dal prodotto della frequenza di campionamento per il numero di secondi che si desiderano simulare. Viene poi definito s, ovvero il vettore contenente il

sound / musica, attraverso una funzione costruita appositamente, *leggi_mp3*. Tale segnale viene amplificato di un fattore quattro, analogamente anche per il rumore successivo. Si definisce quindi r come il rumore esterno alla cuffietta bluetooth, anch'esso utilizza una funzione, *leggi_rumore*, per la propria creazione che viene presentata nelle sezioni a seguire, fornendo un vettore contenente i propri valori generati. Viene infine simulato l'effetto filtrante della scocca della cuffietta mediante la funzione *filtra_rumore_filter*, quest'ultima in particolare prende come argomenti sia il rumore, che la musica e la percentuale di errore in lettura in quanto, come detto in precedenza, si vuole introdurre un errore per simulare la non idealità del microfono, che viene quindi applicata mediante questa funzione.

```
14      energia_media_rimasta = zeros(lati_ipercubo);
15      energia_media_d = energia_media_rimasta;
16      tempo_di_esecuzione = energia_media_rimasta;
17      memoria_occupata = energia_media_rimasta;
18      percentuale_di_tempo_impiegato = energia_media_rimasta;
19      index_nA = 1;
20      for nA = inizi_scale(1):passi_scale(1):passi_scale(1)*...
21              (lati_ipercubo(1)-1)+inizi_scale(1)
22          index_nB = 1;
23          for nB = inizi_scale(2):passi_scale(2):passi_scale(2)*...
24                  (lati_ipercubo(2)-1)+inizi_scale(2)
25              index_lambda = 1;
26              for lambda = inizi_scale(3):passi_scale(3):...
27                      passi_scale(3)*(lati_ipercubo(3)-1)+inizi_scale(3)
28                  tic;
29                  [y,~,d_stimato,memoria_occupata(index_nA,index_nB,index_lambda)]=...
30                      calcolo_predittivo(lambda,nA,nB,r,d,s,campioni);
31                  [energia_media_d(index_nA,index_nB,index_lambda), ...
32                      energia_media_rimasta(index_nA,index_nB,index_lambda)] =...
33                      calcola_energia_media_errore_di_predizione(d,d_stimato);
34                  tempo_di_esecuzione(index_nA,index_nB,index_lambda)=toc/(campioni);
35                  percentuale_di_tempo_impiegato(index_nA,index_nB,index_lambda) = ...
36                      tempo_di_esecuzione(index_nA,index_nB,index_lambda)*fs;
37                  index_lambda = index_lambda +1;
38                  fprintf('Indici: [nA = %d | nB = %d | lambda = %d] \n', ...
39                      index_nA,index_nB,index_lambda-1);
40              end
41              index_nB = index_nB + 1;
```

```
42          end
43          index_nA = index_nA +1;
44      end
```

Dopo aver importato i file audio, creato il rumore esterno ed averlo filtrato, si passa ora all'inizializzazione delle matrici per il calcolo di energia media degli errori, tempo di esecuzione del programma e percentuale di memoria occupata rispetto al tempo massimo che si ha per terminare il calcolo prima di incorrere nel nuovo istante di campionamento. Ognuna di queste matrici ha la dimensione dell'ipercubo di partenza, questo sta a significare che per ogni terna di valori nA, nB, λ che vengono simulati, viene fornito un valore a queste matrici che può essere utilizzato in seguito per il confronto dei risultati ottenuti per coefficienti differenti. Attraverso la funzione *calcolo_predittivo* si passa ora alla previsione del rumore futuro percepito all'interno della cuffietta al nuovo passo di campionamento, utilizzando l'algoritmo di *RPEM*, ai fini di generare un'onda in controfase che sia in grado di cancellarla. Dato che la simulazione viene effettuata per diversi valori di nA, nB e λ, sono stati utilizzati tre cicli *for* annidati al fine di scorrere tra i valori del numero di coefficienti e di λ che si desiderano confrontare. La funzione di calcolo del rumore futuro, che viene presentata nella sezione successiva, restituisce y, ovvero il risultato finale udibile, dato dalla sovrapposizione di musica, rumore e dall'opposto del rumore previsto; *d_stimato*, che costituisce il rumore predetto dal filtro, ed infine la memoria occupata per quel set di coefficienti. Si procede poi al calcolo delle energie del rumore interno alla cuffietta e di quello residuo dopo la previsione, le quali saranno fondamentali per ricavare il fit del filtro per ogni insieme di valori. Nel mentre, mediante le funzioni *tic* e *toc*, viene misurato il tempo di esecuzione, in modo da permettere poi la stima della percentuale del tempo a disposizione che viene utilizzato per quel determinato numero di coefficienti. Si passa infine ad un *print* degli indici per i quali la funzione ha completato la simulazione, in modo da poter stimare visivamente il tempo di completamento necessario, dato che, se si desiderano tempi di simulazione molto elevati e filtri che utilizzano un numero di coefficienti molto alto, il tempo di esecuzione di questa parte di codice diviene tutt'altro che trascurabile.

```
45      grafica_errori_di_predizione(energia_media_d,energia_media_rimasta,...
46      inizi_scale,passi_scale);
```

La funzione soprastante, *grafica_errori_di_predizione*, si occupa della visualizzazione dei valori ottenuti di energia per ogni insieme di coefficienti nA, nB e λ. Questa funzione è

necessaria al fine di visualizzare in modo rapido per quali parametri il calcolo predittivo effettuato dall'algoritmo di *RPEM* avviene in modo più efficace; quindi con un fit maggiore e di conseguenza rappresenta anche un modello migliore del sistema preso in esame. Al fine di calcolare il fit citato in precedenza si è proceduto come segue:

- calcolo dell' energia media dei vettori del disturbo d e del disturbo residuo $d - d\_stimato$ utilizzando la funzione *calcola_energia_media_errore_di_predizione* ed in particolare le formule:

  $$E_d = \frac{1}{N}\sum_{i=1}^{N}(d[i])^2,$$

  dove E_d è l'energia media del disturbo e

  $$E_{d-d_{stimato}} = \frac{1}{N}\sum_{i=1}^{N}(d[i] - d[i]_{stimato})^2,$$

  dove $E_{d-d_{stimato}}$ è l'energia media del disturbo residuo

- calcolo del fit utilizzando la formula:

  $$\text{fit} = 1 - \frac{E_{d-d_{stimato}}}{E_d} \tag{3.1}$$

  all'interno della funzione *grafica_errori_di_predizione*.

Come mostrato nel capitolo successivo, che presenta i risultati ottenuti, questo metodo realizza un ipercubo, o nel caso preso in esame un parallelepipedo, dato che il numero di indici che si è deciso di variare è stato stabilito a tre, nA, nB, λ, e che il numero di passi di questi ultimi può essere anche differente uno dall'altro. Per avere una prima intuizione sul tipo di rappresentazione del risultato si può immaginare che questo parallelepipedo sia composto da punti posti ad intervalli regolari ed equidistanti, e che ognuno di questi rappresenti un insieme nA, nB, λ, e venga colorato in funzione della percentuale del fit ottenuta a partire dai calcoli precedenti.

```
47      X_in = 4;Y_in = 4;Z_in = 1;
48      X_f = 7;Y_f = 7;Z_f = 3;
49      sottomatrice_energie_medie_rimaste = ...
50          energia_media_rimasta(X_in:X_f,Y_in:Y_f,Z_in:Z_f);
51      sottomatrice_energie_medie_d = ...
52          energia_media_d(X_in:X_f,Y_in:Y_f,Z_in:Z_f);
53      grafica_errori_di_predizione(sottomatrice_energie_medie_d,...
```

```
54          sottomatrice_energie_medie_rimaste,inizi_scale+...
55          [X_in-1,Y_in-1,Z_in-1],passi_scale);
```

Questa parte del codice è stata dedicata allo zoom sui punti importanti del parallelepipedo ottenuto in precedenza, dato che spesso il fit assume valori molto più bassi in alcune aree della figura rispetto ad altre, non permettendo una chiara visualizzazione del colore del risultato ottenuto, si utilizza quindi nuovamente la funzione di visualizzazione, ma inizializzando per quali intervalli degli indici si desidera visualizzarlo.

```
56      grafica_percentuale_di_tempo_impiegato(percentuale_di_tempo_impiegato);
57      grafica_memoria_occupata(memoria_occupata);
```

Si procede quindi con il display del tempo impiegato e della memoria occupata. Per quanto riguarda il tempo impiegato si è ragionato come segue, dato che un file *.mp3* è composto da campioni riprodotti ad una frequenza di 44100 Hz, ne consegue che le fasi di campionamento dei microfoni e di calcolo debba avvenire in un tempo massimo:

$$\Delta t_{max} = \frac{1}{fs}.$$

Si è quindi calcolata la percentuale di tempo utilizzata dall'algoritmo di predizione, rispetto al Δt_{max}.

```
58      secondi = 0.5; aspetta_secondi = 2.3;
59      campioni = round(secondi*fs);
60      salto = round(aspetta_secondi*fs);
61      y_max = 0.03 + max(max(abs(s(salto:salto+campioni))),...
62          max(abs(d(salto:salto+campioni))));
63      tempo = linspace(0, secondi, campioni+1);
64      grafica_audio(tempo,s(salto:salto+campioni),"s");
65      grafica_audio(tempo,d(salto:salto+campioni),"d");
66      grafica_audio(tempo,d_stimato(salto:salto+campioni),"d_stimato");
67      grafica_audio(tempo,y(salto:salto+campioni),"y");
68      ylim([-y_max, y_max]);
```

A partire dalle righe di codice soprastanti viene ora presentato il display di una breve parte della simulazione, di durata *secondi*, vengono inoltre saltati i primi secondi attraverso la variabile *aspetta_secondi*, dato che lo scopo di questa parte della visualizzazione è quella di verificare che dopo una prima fase di assestamento dei coefficienti i grafici di *musica* e *uscita* e quelli di *rumore* e *rumore_stimato* siano quasi sovrapposti, ad indicare che la stima stia effettivamente avvenendo nel modo corretto. Viene quindi creata una base dei tempi con la funzione *linspace*, definito il limite massimo verticale del grafico, *y_max*, in modo da avere lo zoom desiderato, ed eventualmente modificarlo comodamente rieseguendo solamente codesta parte di codice.

```
69      secondi = 3; campioni = round(secondi*fs); aspetta_secondi = 0.5;
70      salto = round(aspetta_secondi*fs);
71      y_max = max(abs(d(salto:salto+campioni))+0.1);
72      grafica_scostamenti(y_max,d(salto:salto+campioni),...
73          d_stimato(salto:salto+campioni),rumore_extra(salto:salto+campioni),fs);
```

Si passa infine ad un confronto grafico diretto tra il rumore effettivo percepito con e senza l'aggiunta della predizione.

```
74      secondi = 6; campioni = round(secondi*fs); pausa = 1;
75      ascolta(d(salto:salto+campioni),fs,campioni,pausa);
```

Terminata ora la parte relativa al display, inizia quella relativa all'ascolto dei rumori, nelle righe soprastanti viene richiamata la funzione *ascolta*, la quale riproduce il rumore filtrato percepito all'interno dell'auricolare a musica spenta per mezzo dell'altoparlante del PC. Viene quindi data enfasi al disturbo d, che si desidera cancellare.

```
76      secondi = 6; campioni = round(secondi*fs);
77      ascolta(d(salto:salto+campioni)-d_stimato(salto:salto+campioni),...
78          fs,campioni,pausa);
```

Si passa successivamente all'ascolto del rumore residuo all'interno dell'auricolare, a musica spenta, ovvero di ciò che non si è riusciti a cancellare. In questo modo, ascoltando questo

audio ed il precedente si può avere un'altra idea del funzionamento effettivo del metodo implementativo scelto, o in pratica della differenza del rumore con e senza l'utilizzo della cancellazione.

```
79      secondi = 6; campioni = round(secondi*fs); pausa = 1;
80      ascolta(y(salto:salto+campioni)+d_stimato(salto:salto+campioni),...
81          fs,campioni,pausa);
```

Conclusa ora la parte relativa ai rumori, inizia quella dedicata alla musica, in particolare nel codice soprastante viene proposto l'ascolto della musica che si desidera ascoltare, con sovrapposto il disturbo d, rappresentante il rumore filtrato dalla scocca della cuffietta, senza l'utilizzo della cancellazione attiva.

```
82      secondi = 6; campioni = round(secondi*fs); pausa = 1;
83      ascolta(y(salto:salto+campioni),fs,campioni,pausa);
```

L'ultimo ascolto è dedicato alla sovrapposizione di musica, rumore filtrato e rumore previsto dal filtro attivo. Ne consegue che, se il filtro è stato implementato correttamente, ci si aspetta di sentire la musica desiderata in assenza quasi totale di rumore. Lo scopo di questi ultimi due ascolti è quello ancora una volta di apprezzare la differenza tra il non utilizzo del filtro di cancellazione e l'utilizzo di quest'ultimo.

```
84      [energia_minima, indice_lineare] = min(energia_media_rimasta(:));
85      [indice_x, indice_y, indice_z] = ind2sub(size(energia_media_rimasta),...
86          indice_lineare);
87      nA_migliore = inizi_scale(1) + passi_scale(1)*(indice_x-1);
88      nB_migliore = inizi_scale(2) + passi_scale(2)*(indice_y-1);
89      lambda_migliore = inizi_scale(3) + passi_scale(3)*(indice_z-1);
90      nA_migliore,nB_migliore,lambda_migliore,
```

Si passa poi alla ricerca del miglior fit ottenuto dai calcoli di predizione effettuati in precedenza, ovvero dei migliori valori di nA, nB e λ che minimizzino l'errore quadratico medio di predizione. Questo viene fatto attraverso le righe *84* e *85*. La prima di queste

dispone la matrice delle energie in colonna, in modo da poterne estrarre facilmente l'indice lineare della posizione del valore minimo con la funzione *min*. Si utilizza poi la funzione *ind_2sub* per ricavare i tre indici della matrice a partire da quello della matrice disposta in colonna e conoscendone le dimensioni.

```
91      [b,a,rumore_extra] = crea_filtro_butter(s);
```

Ora che le simulazioni con la *RPEM*, le verifiche di funzionamento e la relativa ricerca dei migliori nA, nB e λ sono conclusi si passa ad un nuovo set di dati. Nello specifico ciò che è stato fatto è utilizzare rumori differenti, ovvero quelli di un phon, di un martello pneumatico e di un tagliaerba per verificare che l'utilizzo di questi valori ottimali ottenuti siano validi anche per classi di rumori differenti e di conseguenza non siano limitati al caso isolato del *training dataset*. Oltre a questo si è deciso di complicare il modello che simula l'effetto filtrante della scocca, come viene presentato nelle sezioni successive.

```
92      r_phon = 5*leggi_mp3('rumore_phon.mp3',secondi_val*fs)';
93      [d_phon,rumore_extra] = filtra_rumore_butter(b,a,rumore_extra, r_phon,s,fs,p);
94      campioni_phon = min(size(r_phon,2),size(s,2));
95      [~,~,d_stimato_phon] = calcolo_predittivo(lambda_migliore,nA_migliore,...
96          nB_migliore,r_phon,d_phon,s,campioni_phon);
97      [energia_media_d_phon, energia_media_rimasta_phon] =...
98          calcola_energia_media_errore_di_predizione(d_phon,d_stimato_phon);
99      fit_phon_PEM = 1-energia_media_rimasta_phon/energia_media_d_phon;
100     fprintf('Fit del modello: %4f',fit_phon_PEM);
101     aspetta_secondi = 0.1; secondi = secondi_val-aspetta_secondi;
102     y_max = max(abs(d_phon)+0.1); campioni = round(secondi*fs);
103     salto = round(aspetta_secondi*fs);
104     grafica_scostamenti(y_max,d_phon(salto:salto+campioni),...
105         d_stimato_phon(salto:salto+campioni),...
106         rumore_extra(salto:salto+campioni),fs);
```

Si procede importando il nuovo file audio del rumore generato, in questo caso da un phon, e lo si riscala adeguatamente, vi si applica poi il filtro creato in precedenza allo scopo di ricavare il rumore residuo interno alla cuffietta, *d_phon*. Si effettua poi nuovamente il calcolo predittivo dei rumori previsti per passi successivi servendosi questa volta unicamente

dei migliori na, nB, λ, i quali hanno garantito il miglior fit nel set di allenamento. Si procede poi con il calcolo del fit e con le funzioni di display analogamente a quanto fatto in precedenza. Ai fini di mantenere una certa scorrevolezza nella trattazione vengono ora tralasciati i casi in cui i rumori esterni importati sono quelli di un martello pneumatico e di un tagliaerba, in quanto ricavati a partire da righe di codice analoghe alle soprastanti appena commentate.

```
107     campioni_te = min(size(r_te,2),size(s,2));
108     [theta,d_stimato_te] = ...
109         calcolo_lms_predittivo(mu,lambda_migliore,...
110         nA_migliore,nB_migliore,r_te,d_te,s,campioni_te);
111     [energia_media_d_te, energia_media_rimasta_te] = ...
112         calcola_energia_media_errore_di_predizione(d_te,d_stimato_te);
113     fit_te_LMS = 1-energia_media_rimasta_te/energia_media_d_te;
114     fprintf('Fit del modello: %4f',fit_te_LMS);
115     aspetta_secondi = 0.1; secondi = ...
116         secondi_val-aspetta_secondi; y_max = max(abs(d_te)+0.1);
117     campioni = round(secondi*fs); salto = round(aspetta_secondi*fs);
118     grafica_scostamenti(y_max,d_te(salto:salto+campioni), ...
119         d_stimato_te(salto:salto+campioni),...
120         rumore_extra(salto:salto+campioni),fs);
```

Si procede infine con il confronto degli algoritmi di *RPEM* ed *LMS*. Nello specifico si è deciso di adottare l'esempio del tagliaerba, ci si serve quindi della funzione *calcolo_lms_predittivo*, la quale ingloba il miglior numero di coefficienti ottenuti per la *RPEM*, ovvero nA ed nB, ma non utilizzando più λ, in quanto non previsto da questo algoritmo, ed introducendo invece il coefficiente μ, definito all'inizio del codice. Analogamente a quanto fatto in precedenza, anche in questo caso vengono proposti dei grafici ai fini di una rapida interpretazione visiva di quanto ottenuto.

```
121     fprintf(['Confronto delle simulazioni col rumore del tagliaerba:...' ...
122         ' \nFit del modello RPEM: %4f\nFit del modello LMS: %4f'],...
123         fit_te_PEM,fit_te_LMS);
```

Per concludere viene mostrato tramite *print* il risultato del confronto delle simulazioni ed in particolare del fit ottenuto mediante i due algoritmi, *RPEM* ed *LMS*.

3.2.2 Funzione di calcolo predittivo RPEM

Nella sezione presente viene illustrata la metodologia di implementazione dell'algoritmo *RPEM*, applicando la teoria richiamata nei capitoli precedenti. Ancora una volta agli spezzoni di codice proposti segue una rapida spiegazione.

`calcolo_predittivo`

```
function [y,theta,d_stimato,memoria_occupata] = ...
    calcolo_predittivo(lambda,nA,nB,r,d,s,campioni)

    y = zeros(1,campioni); d_stimato = y; M_P = y; M_E = y; out = y;
    memoria_iniziale = sum([whos().bytes]);
    fiT = zeros(1,nA+nB);
    fi = (fiT');
    St = eye(nA+nB);
    theta = 0.1*ones(nA+nB,1);
    for i = 1:1:campioni-2
```

Le variabili adottate in questa funzione sono λ, nA ed nB, il rumore esterno r, il rumore interno d, il segnale musica s ed il numero dei campioni da simulare. Vengono quindi inizializzate le variabili, in particolare il vettore ϕ dei valori passati di ingresso e uscita, definito nella (2.4), la matrice St, definita nella (2.8) ed il vettore dei coefficienti θ definito nella (2.5). Si inizia infine ad iterare tra gli istanti di campionamento al fine di simulare l'acquisizione dei dati.

```
        %M_P(1,i) = r(1,i); %lettura del microfono esterno, principale
        %M_E(1,i) = s(1,i)+d(1,i)-d_stimato(1,i); %lettura del...
                    %microfono interno, di errore
        %d(1,i) = M_E(1,i)-s(1,i)+d_stimato(1,i);
```

Viene ora proposta una parte di codice con lo scopo di verificare l'adattabilità di questa funzione ad un contesto reale, ma che viene bypassata al fine di ridurre la complessità computazione del codice e ridurne di conseguenza il tempo di esecuzione. In riferimento allo schema del principio di funzionamento proposto in (Fig. 3.2), non disponendo

direttamente dei rumori esterno ed interno alla cuffia d, vi è la necessità di ricavarli a partire dai soli segnali noti. Per fare questo si utilizzano rispettivamente i microfoni principale, M_P, esterno alla scocca e di errore, M_E, interno all'auricolare. In particolare quindi, il rumore esterno r viene letto dal microfono M_P principale esterno, mentre il disturbo interno d viene dedotto dal microfono M_E di errore interno sapendo che questo ad ogni istante acquisisce il segnale s, di valore noto, il disturbo voluto d ed il disturbo stimato al passo di campionamento precedente *d_stimato*, anch'esso di valore noto. Risulterebbe quindi solamente ripetitivo ed inutile in questo esempio ricavarsi il valore letto dai microfoni per poi ottenere nuovamente i valori già noti dei rumori r e d, mentre sarebbe indispensabile nel contesto di una applicazione in *real time*.

```
15          % fiT_theta = fi' * theta;
16          % StT = St + lambda * (fi * fi');
17          % L = chol(StT);
18          % inv_StT = (L') \ ((L) \ eye(size(StT)));
19          % theta = theta + inv_StT * fi * (d(1,i) - fiT_theta);
20
21          fiT_theta = fi' * theta;
22          StT = St + lambda * (fi * fi');
23          theta = theta + StT \ fi * (d(1,i) - fiT_theta);
```

Qui sopra vengono ora utilizzate le formule ricorsive della PEM come ricavato nei capitoli precedenti nella (2.9). Nella parte commentata viene inoltre proposta una metodologia differente per l'implementazione dell'algoritmo, più conforme ad invertire rapidamente matrici di dimensioni più elevate, quest'ultima si basa sulla *decomposizione di Cholesky*.

```
24          fi = aggiornafi(fi,- d(1,i),r(1,i),nA);
25          fiT = fi';
```

Si procede con l'aggiornamento del vettore dei campioni, in particolare del rumore r e del disturbo d. Questa procedura avviene tramite la funzione *aggiornafi*, la quale aggiorna il vettore ad ogni passo di campionamento introducendo i valori di rumore e disturbo, dedotti dalle letture effettuate dai microfoni.

```
26          d_stimato(1,i+1) = fiT*theta;
```

Viene quindi effettuata la stima del rumore previsto per il passo di campionamento successivo mediante il prodotto del vettore dei valori per quello dei coefficienti del filtro.

```
27          y(1,i) = s(1,i) - d_stimato(1,i) + d(1,i);
28      end
29      memoria_occupata = sum([whos().bytes])-memoria_iniziale;
30  end
```

Ci si preoccupa infine di estrarre il risultato finale y, che si ricorda rappresentare il contenuto musicale che viene poi effettivamente udito dall'utente, questo è dato dalla sovrapposizione dei contributi di musica, rumore e rumore di cancellazione.

3.2.3 Funzione di calcolo predittivo LMS

Si riportano ora unicamente le tecniche di aggiornamento dell'algoritmo proposto in precedenza in modo da esaminare brevemente il caso specifico dell'implementazione dell'algoritmo LMS, realizzato mediante la medesima tecnica.

`calcolo_lms_predittivo`

```
9       nuovo_val=0;
10      for s=1:nA+nB
11          nuovo_val = nuovo_val + lambda^(s-1);
12      end
13      gamma = nuovo_val^(-1);
```

Nelle righe soprastanti si definisce il parametro *gamma* come riportato nella (2.10) il quale, sostituendo S_t, rende non più necessario il calcolo di inversione matriciale.

```
16          fiT_theta = fi' * theta;
17          theta = theta + (2*mu*gamma*eye(nA+nB,nA+nB))*fi * (d(1,i) - fiT_theta);
```

All'interno del ciclo di aggiornamento dei coefficienti viene inoltre sostituita la formula con quella riportata nella (2.12). Con le seguenti modifiche la trattazione effettuata in precedenza del caso generico *RPEM*, si riduce al caso più specifico *LMS*.

3.2.4 Funzioni di filtraggio del rumore esterno

Nella presente sottosezione viene ora proposta una breve trattazione della metodologia di simulazione del filtraggio del rumore attuato dalla scocca dell'auricolare. Nello specifico si parte da un modello semplice dell'auricolare, utilizzato per testare il funzionamento del programma ed allenare l'algoritmo sul *training dataset* per estrarre il numero più adatto di coefficienti nA ed nB per predire il risultato del filtraggio; per poi passare ad un modello più complicato per vedere se i risultati ottenuti rimangono validi anche per filtri più complessi e rumori differenti. A partire allora dal rumore esterno r, si vuole quindi ottenere il disturbo d, risultante dal filtraggio di r per mezzo della presenza fisica della scocca degli auricolari tra sorgente di rumore e timpano dell'utente. Come visto nei capitoli precedenti un modello semplificato di questa azione filtrante è dato da un filtro passa basso del primo ordine, il quale descrive quindi il comportamento fisico della scocca. Per i primi test si è allora deciso di utilizzare per a e b, che rappresentano rispettivamente i coefficienti di denominatore e numeratore del filtro discreto. I valori sono riportati sotto:

`filtra_rumore_filter`

```
2         b = [0,0.1255,0.1255];
3         a = [1,-0.75];
```

I valori di a e b, rappresentanti i coefficienti del filtro che si desidera predire in modo da poter in seguito cancellare il disturbo, sono stati scelti in modo da realizzare un filtro che:

- descriva il comportamento fisico passa basso della scocca dell'auricolare;
- descriva un filtro predittivo, imponendo quindi il primo termine dell'array b pari a zero e simulandone conseguentemente il ritardo presente inevitabilmente nei processi di lettura del caso reale.

Si riporta ora la forma della funzione di trasferimento del filtro discreto implementato ai fini di una più diretta visualizzazione delle considerazioni citate in precedenza:

$$d = z^{-1}\frac{b_1 + b_2 z^{-1}}{a_0 + a_1 z^{-1}1} r + e.$$

Ora che si dispone dei coefficienti del filtro e del rumore esterno r è possibile simulare il disturbo che sarà presente all'interno dell'auricolare mediante la funzione *filter*. A questo disturbo viene aggiunto del rumore Gaussiano a media non nulla e. Si procede quindi con

la costruzione di un filtro più complicato, dedito alla verifica della validità dei risultati ottenuti. Ai fini di realizzare un filtraggio il più aleatorio possibile si è utilizzata la metodologia seguente:

crea_filtro_butter

```
    while true
        b = randn(1, 7)/7;
        a = randn(1, 9)/9;
        [h,w] = freqz(b, a);
        if max(abs(h)) < 1
            h=h/max(abs(h));
            for i=1:length(h)
                if(abs(h(i))<0.1)
                    h(i)=0.1;
                end
            end
            [b,a]=invfreqz(h,w,3,3);
            break;
        end
    end
```

Nel codice soprastante si utilizza un ciclo *while* ai fini di produrre coefficienti random fintanto che uno di questi rispetti determinate caratteristiche e permetta di uscire dal ciclo. In particolare si utilizza la funzione *freqz* per ricavare la risposta in frequenza del filtro dato dai coefficienti a e b generati casualmente. Si passa poi alla verifica della stabilità del filtro, controllando che il valore assoluto del valore massimo del modulo non superi il valore uno, e conseguentemente tutto il diagramma di Bode del modulo sia inferiore all'asse orizzontale in ogni punto. Si prosegue poi riscalando il modulo in modo tale da portare il modulo del valore massimo ad uno, alzando quindi il diagramma di Bode del modulo fintanto che il punto di massimo valga uno. Si limita inoltre il modulo ad un valore minimo pari a 0.1, in modo da evitare che vi sia un'attenuazione troppo consistente dell'ingresso, questo, oltre ad essere non verosimile al caso reale, comporterebbe la generazione di uscite di valore tendente a zero, risultando quindi nella non invertibilità della matrice St dell'algoritmo

PEM. L'ultimo passaggio consiste nell'estrarre dei nuovi coefficienti che descrivano questo tipo di comportamento, per farlo si è utilizzata la funzione *invfreqz*, la quale, a partire dalla risposta in frequenza modellata come desiderato, ne ricava una stima producendone in uscita i relativi valori dei coefficienti a e b che la realizzano. Terminata l'applicazione di questo primo filtro si aggiungono poi degli errori di origine Gaussiana, come fatto in precedenza e si applica un ulteriore filtro ai fini di simulare nuovamente il comportamento fisico della scocca, questa volta non più con un filtro del primo ordine ma con la funzione *butter*, molto più selettiva, come proposto in seguito:

`filtra_filtro_butter`

```
3     ordine = 6; frequenza_di_taglio = 2000;
4     [b2, a2] = butter(ordine, frequenza_di_taglio / (fs / 2), 'low');
```

É stata quindi presentata in maniera molto discorsiva la metodologia di implementazione dei due filtri realizzati: il primo, semplice e dedicato al *dataset* di allenamento; il secondo, più complicato e dedito alla verifica dell'effettiva efficacia su un più ampio spettro dei risultati ottenuti su un modello più semplice.

3.3 Verifica di correttezza

Ora ci si pone il problema di verificare che gli algoritmi descritti in precedenza funzionino correttamente prima di iniziare le simulazioni. Come prima cosa si è scelto di rimuovere gli errori aggiuntivi Gaussiani introdotti per emulare la non idealità dei microfoni al fine di evitare che questi ultimi diano un contributo troppo consistente al filtraggio del rumore in ingresso. Si è inoltre scelto di utilizzare un filtro con i valori dei coefficienti a e b ben definiti, come mostrato nella sottosezione precedente, in modo da poter verificare se al termine del programma i coefficienti stimati convergessero o meno a quelli scelti: $b = [0, 0.1255, 0.1255]$; $a = [1, -0.75]$. Di conseguenza si è innanzitutto verificato che scegliendo nA ed nB pari al valore vero le stime di a e b convergessero al valore vero dei coefficienti a e b scelti per il filtro discreto. Al termine di questa breve simulazione si sono poi introdotti e tarati i rumori Gaussiani in modo tale che non fossero né irrisori e né inadeguatamente elevati relativamente al volume del rumore esterno, indicativamente si è scelto di mantenerli al di sotto di valori che non superassero il 50% del volume massimo del rumore esterno. Tramite le opportune simulazioni effettuate è stato possibile osservare come i coefficienti convergessero ai valori iniziali anche in presenza di errore e partendo

da valori iniziali differenti. Si è poi passato a simulare il funzionamento anche con un numero di coefficienti differenti da quelli inizialmente scelti, in codesto caso si è osservato che i coefficienti tendessero a valori differenti in presenza degli errori Gaussiani introdotti; tuttavia, mediante questi ultimi, lo scarto quadratico medio dopo un certo intervallo di tempo convergeva comunque ad un valore nullo. Ascoltando inoltre il risultato finale della sovrapposizione di musica, disturbo e disturbo predetto si può chiaramente sentire come questo venga rimosso quasi interamente, accertandone quindi la correttezza.

Capitolo 4

Risultati sperimentali

In questo capitolo vengono presentati i risultati ottenuti mediante simulazioni, in particolare il focus viene posto sull'interpretazione.

4.1 Analisi dei dati di input: rumori registrati e variabili

I dati di input che sono stati forniti all'algoritmo per la simulazione che viene presentata in seguito sono i seguenti:

- tempo di simulazione di 3 secondi: si è osservato che prendendo un tempo di simulazione molto superiore al tempo di convergenza dell'algoritmo, deducibile graficamente, lo scarto quadratico medio tende a zero per qualunque coppia di numeri di coefficienti nA ed nB superiori ad un minimo di 2/3, rendendo di fatto inutile lo studio in quanto dipendente unicamente dagli errori gaussiani;

- frequenza di campionamento di 44100Hz: si è scelto di utilizzare file multimediali di tipo *.mp3*, la cui frequenza intrinseca è quella indicata, la scelta implementativa è stata fatta ai fini di testare il funzionamento in un caso di comune applicazione;

- i parametri nA, nB e λ per cui effettuare il confronto sul *training dataset*: per l'algoritmo *RPEM* si è scelto di prendere le combinazioni di nA ed nB con valori compresi tra 1 e 12 e di λ per valori compresi tra 0.95 e 1 con passo 0.25 ;

- percentuale di errore di lettura del microfono esterno: si è introdotto un errore di lettura per rendere la previsione dei coefficienti dipendente non solo dal rapporto tra rumore esterno r e disturbo interno filtrato d, ma anche dal rumore esterno stesso. Si è quindi simulata la presenza di un errore di lettura aggiungendo un offset

di valore pari al 3% di s a d, in aggiunta agli errori Gaussiani, come mostrato qui sotto:

  - definizione di media e deviazione standard degli errori Gaussiani introdotti:

$$\begin{aligned} m_1 &= 0, & \sigma_1 &= 0.02 \\ m_2 &= -0.001, & \sigma_2 &= 0.03 \\ m_3 &= 0.0002, & \sigma_3 &= 0.04 \end{aligned}$$

  - generazione degli errori Gaussiani rispetto alla normale standard:

$$\begin{aligned} r_1 &= \sigma_1 \cdot \mathcal{N}(0,1) + m_1 \\ r_2 &= \sigma_2 \cdot \mathcal{N}(0,1) + m_2 \\ r_3 &= \sigma_3 \cdot \mathcal{N}(0,1) + m_3 \end{aligned}$$

  - aggiunta dell'offset del 3% del segnale s:

$$r_4 = 0.03 \cdot s$$

  - calcolo del rumore totale aggiunto al disturbo interno d:

$$\text{rumore\_extra} = r_1 + r_2 + r_3 + r_4$$

  - aggiunta dei rumori al disturbo interno d:

$$d = d + \text{rumore\_extra}.$$

- coefficiente μ dell'algoritmo LMS: sperimentalmente è stato possibile verificare che per valori intorno a 0.05 del tasso di apprendimento, che regola la velocità di aggiornamento dei coefficienti, quest'ultimo aveva il minimo scarto quadratico medio. Come suggerisce la teoria, inoltre, un valore troppo elevato potrebbe portare la predizione all'instabilità o a delle oscillazioni non desiderate, mentre dei valori troppo bassi potrebbero portare a una convergenza e adattabilità a nuove sorgenti esterne troppo lenta;

- tempo di validazione del *testing dataset*: si è osservato sperimentalmente che prendendo un tempo di validazione troppo basso non è sempre possibile osservare con buona accuratezza la convergenza dell'algoritmo, volendo inoltre simulare una casistica il più possibile analoga al caso reale si è scelto un compromesso tra un tempo

di validazione troppo basso, ed uno troppo elevato che aumenterebbe i tempi di simulazione;

- audio scelti per le simulazioni: al fine di *allenare* l'algoritmo si è utilizzata la concatenazione di tre rumori esterni differenti, quello di un treno percepito all'interno di una carrozza, quello di auto in corsa e quello di una forte folata di vento. Al fine di confrontare le performance degli algoritmi si è invece scelto di utilizzare come rumori esterni di test il rumore prodotto rispettivamente da: un phon, un martello pneumatico ed un tagliaerba.

Diviene ora possibile presentare le performance ottenute dalla simulazione mediante i dati di ingresso presentati nell'elenco soprastante.

4.2 Performance del filtro adattivo RPEM, training

A partire dalle ipotesi effettuate nella sezione precedente si è quindi inizialmente utilizzato l'algoritmo RPEM presentato in precedenza ai fini di valutare per quale terna di valori di nA, nB, λ, fosse possibile minimizzare lo scarto quadratico medio tra il disturbo percepito all'interno delle cuffiette e quello stimato mediante algoritmo al passo di campionamento antecedente, a partire dal *testing dataset*. Di conseguenza in corrispondenza di ogni punto della figura sottostante il colore rappresenta, a seconda dei valori riportati in legenda, la percentuale di energia del segnale che è stata correttamente predetta dall'algoritmo in corrispondenza della terna indicata dalle coordinate del punto. In particolare, per ogni punto avviene una simulazione indipendente dalle altre, la quale utilizza l'algoritmo RPEM per predire il disturbo al successivo passo di campionamento, questo per ogni valore del file *.mp3* di input.

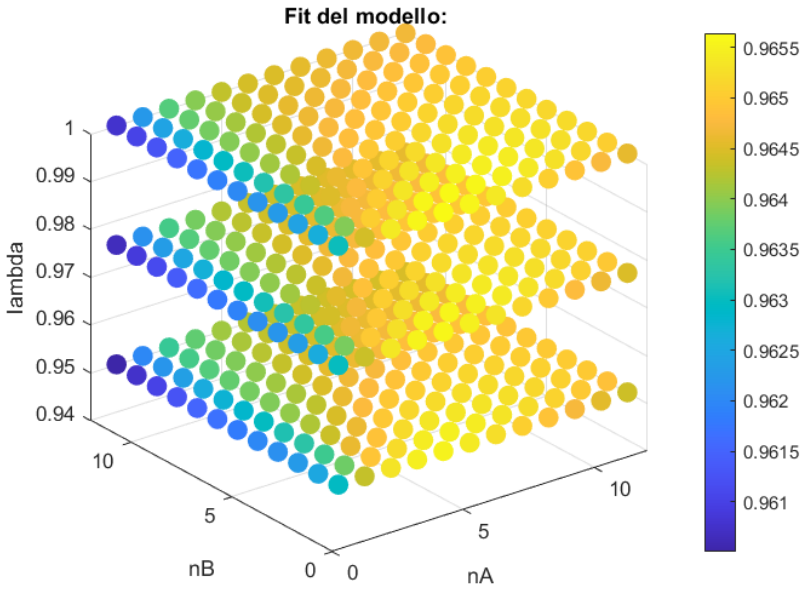

Figura 4.1: Fit del modello in funzione di nA, nB, λ.

Dall'immagine soprastante si evince che la performance dell'algoritmo RPEM è buona per qualsiasi terna scelta, presentando un fit minimo del 96%. Tuttavia, per valori bassi di nA, la performance è di fatto inferiore alle altre combinazioni possibili, mentre per valori di lambda compresi tra 0.95 ed 1 non vi è una sostanziale differenza nelle previsioni, questo è probabilmente dovuto al fatto che essendo il file di test scelto di dimensioni non troppo elevate il numero di campioni per il breve intervallo di tempo scelto è anch'esso basso e ne risulta quindi una scarsa dipendenza dal coefficiente d'oblio. Inoltre, rispetto alla definizione di fit (3.1) è possibile notare che nonostante siano stati introdotti degli

errori aggiuntivi ed il tempo di simulazione sia relativamente breve, 3 secondi, l'algoritmo converga abbastanza velocemente all'andamento futuro del disturbo. Da questi risultati si evince che la miglior terna di valori possibili ai fini di predire l'errore nella casistica presa in esame è la seguente:

- nA_migliore =1
- nB_migliore = 5
- lambda_migliore = 1.

Ora che la fase di test è stata completata e si dispone della terna di numeri rappresentanti il numero ottimale di coefficienti nA ed nB ed il miglior coefficiente λ da utilizzare per ottenere il minor scarto quadratico medio, è possibile vedere la corrispondente performance (Fig. 4.2).

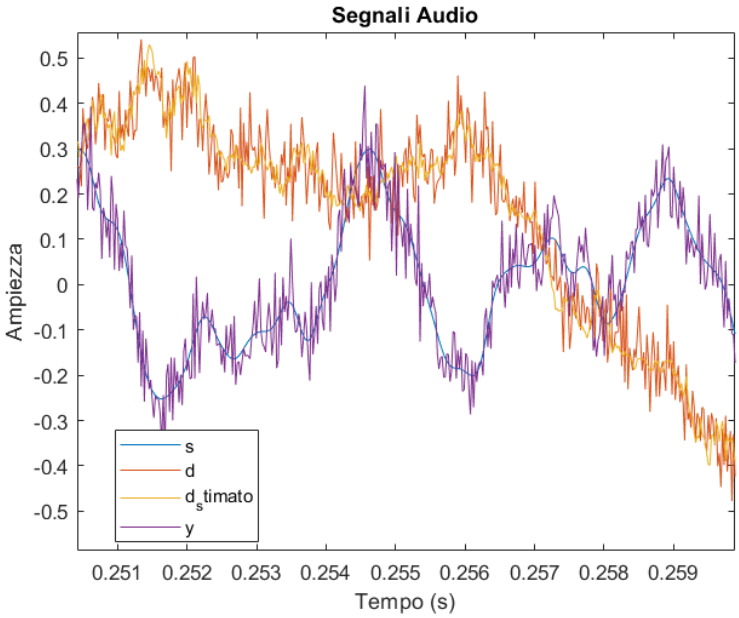

Figura 4.2: Segnali audio del testing dataset.

Nell'immagine soprastante vengono proiettati i principali segnali presentati in precedenza nello schema a blocchi (Fig. 3.1). Nello specifico si ricorda:

- s è il segnale musica che l'utente desidera ascoltare
- d è il disturbo interno o rumore esterno filtrato che arriva all'interno dell'auricolare
- d_stimato è il disturbo stimato dall'algoritmo RPEM per il passo di campionamento futuro, idealmente pari a d

- y è il segnale udito dall'utente.

A partire dal grafico, rappresentante un lasso di tempo per il quale l'algoritmo abbia superato una fase iniziale di assestamento, è possibile dedurre che sia i grafici di d e d_stimato, e conseguentemente quelli di musica s e risultato y siano quasi sovrapposti, a meno di un errore a prima vista relativamente tollerabile. Successivamente sono stati effettuati gli ascolti relativi alle quattro forme d'onda a partire dai quali è stato possibile confermare che il rumore venisse cancellato nella sua quasi totalità, validandone il corretto funzionamento. Vengono ora presentate le differenze tra:

- d che è il disturbo interno o rumore esterno filtrato che arriva all'interno dell'auricolare
- d - d_stimato che è l' errore di predizione ottenuto
- rumore_extra è il rumore che descrive la non idealità del microfono.

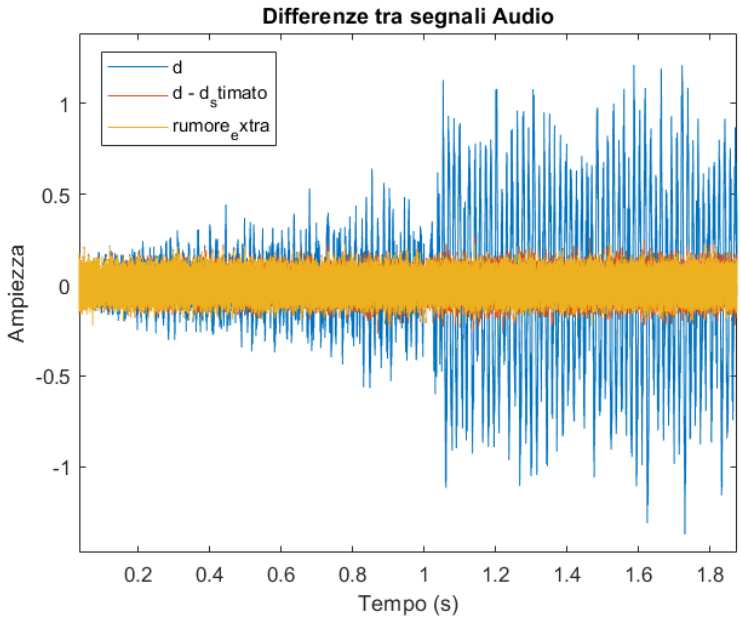

Figura 4.3: Legame tra rumore extra introdotto e previsione del disturbo.

A partire dal grafico è possibile cogliere come l'errore di predizione dipenda nella sua quasi totalità dalla componente di errore non prevista che viene aggiunta in seguito. Questo è possibile notarlo anche dal fatto che il cambio di rumore esterno effettuato al tempo di un secondo non influenzi in modo troppo marcato la predizione. Questo fatto viene reso più evidente anche nell'immagine successiva, la quale non è altro che uno zoom del grafico mostrato in precedenza.

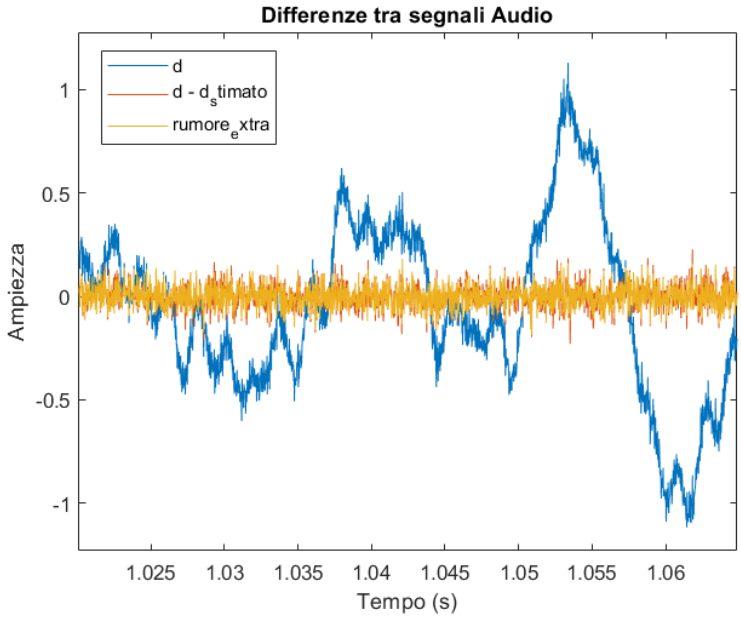

Figura 4.4: Ingrandimento dell'immagine precedente.

Per mezzo di ulteriori simulazioni è stato inoltre possibile accertare che rimuovendo completamente il rumore extra, l'algoritmo convergesse a valori prossimi allo zero dell'errore di predizione.

4.3 Performance del filtro adattivo RPEM, testing

Ora che la fase di training condotta sul dataset composto dalla concatenazione rispettivamente dei rumori esterni elencati in precedenza è terminata ed i valori ottimali di nA, nB e λ sono stati estratti è possibile utilizzarli per verificare se questi continuino a funzionare efficacemente anche per rumori esterni differenti e scorrelati dal dataset di partenza.

4.3.1 Simulazione del rumore di un phon

Nella presente simulazione si è dunque utilizzata la registrazione di un phon ai fini di estendere la validità dei valori estratti in precedenza.

Dalle immagini (Fig. 4.5) e (Fig. 4.6) è evidente come la stima dell'errore sia elevata all'istante di partenza, questo comportamento ancora una volta è dovuto al transitorio necessario affinché i coefficienti del filtro stimati emulino quelli reali della cuffietta. Tuttavia, è interessante notare come per questa tipologia di filtro continui a funzionare

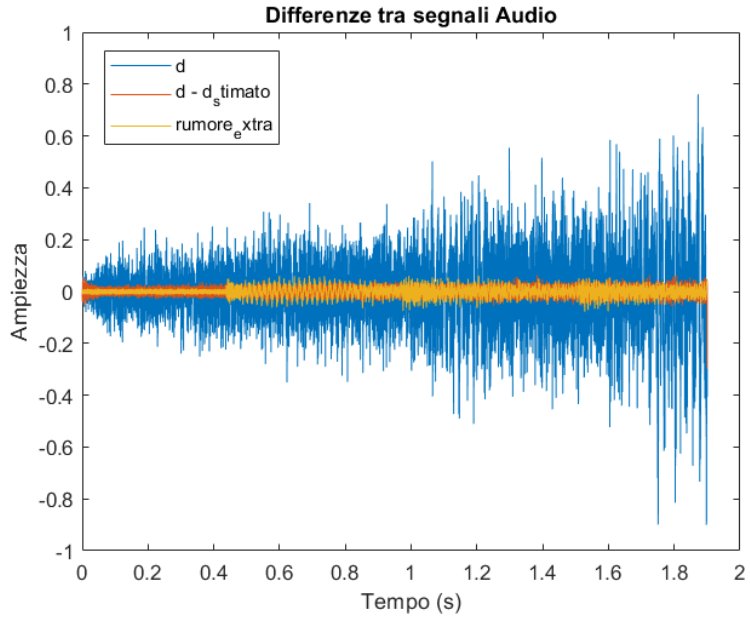

Figura 4.5: Risultato della cancellazione del rumore del phon.

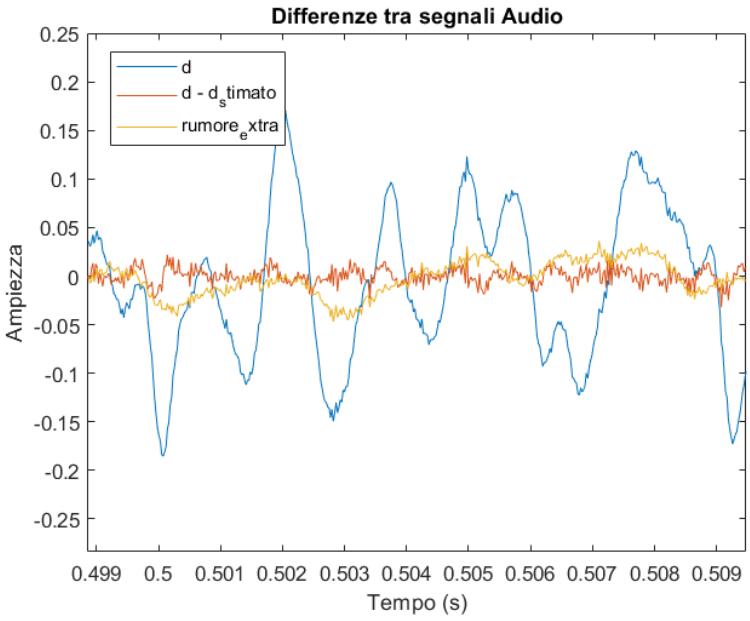

Figura 4.6: Ingrandimento dell'immagine precedente.

correttamente. Difatti, a partire dai calcoli svolti dal software, risulta una fit quasi totale pari a 0.991905, questo primo risultato sembra confermare l'affidabilità dei risultati fin qui ottenuti in relazione ai coefficienti ricavati precedentemente. Malgrado il valore sia molto elevato, anche in relazione al fit del training dataset, è anche vero che l'energia del rumore extra introdotto è molto minore in relazione all'energia del rumore del phon

rispetto a quanto lo fosse in precedenza, questa considerazione giustifica quindi un valore così elevato per il fit.

4.3.2 Simulazione del rumore di un martello pneumatico

Analogamente a quanto fatto in precedenza si estende ora lo studio al caso di un rumore esterno dovuto ad un martello pneumatico, i risultati sono riportati nella figura seguente:

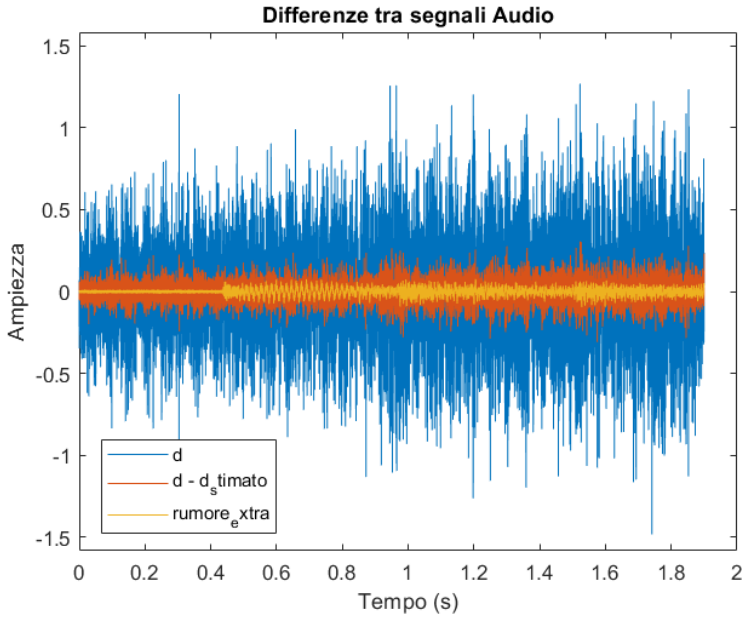

Figura 4.7: Risultato della cancellazione del rumore del phon.

Appare subito evidente il fatto che, a differenza di quanto osservato in precedenza, questa volta la stima del disturbo futuro avviene con una precisione di gran lunga inferiore. Le conclusioni che possono essere tratte partendo da questa nuova simulazione sono che il fit dipende dalla natura del rumore esterno percepito e di conseguenza non dipende unicamente dal filtro. Nel caso proposto si ottiene questa volta un fit di 0.967685, valore di molto inferiore a quello della simulazione precedente, ma comunque in linea con le aspettative dovute ai risultati ottenuti nel training set, intorno a 0.96. Si ricorda inoltre che prendendo un tempo di simulazione più elevato e non introducendo errori aggiuntivi è possibile aumentare di moltissimo le prestazioni, questo ad evidenziare il fatto che molte delle limitazioni dell'algoritmo si basino su questi fattori e non unicamente sul rumore in ingresso.

4.4 Analisi comparativa con approccio LMS

Si passa ora al confronto tra *RPEM* e *LMS*.

4.4.1 Simulazione del rumore di un tagliaerba con algoritmo RPEM

In maniera analoga a quanto fatto in precedenza viene ora presentata la simulazione effettuata rispetto al rumore generato da un tagliaerba utilizzando l'algoritmo predittivo *RPEM*.

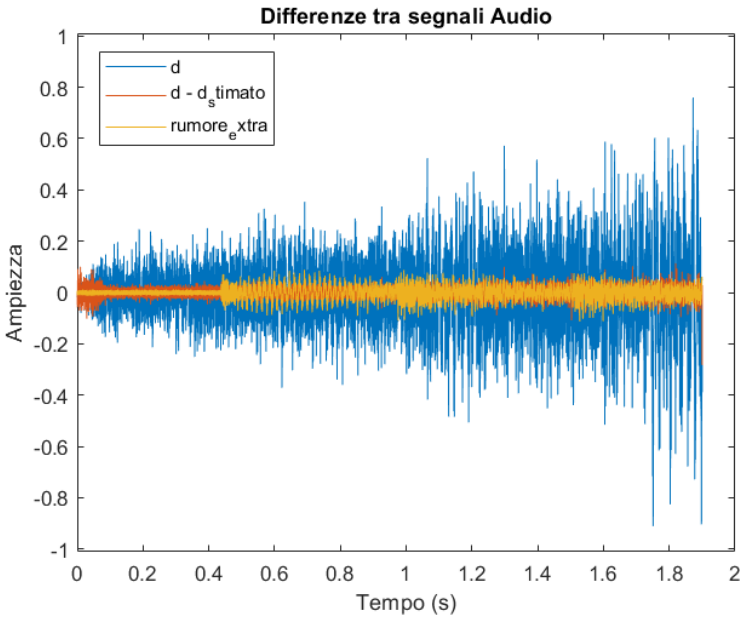

Figura 4.8: Risultato della cancellazione RPEM del rumore del tagliaerba.

Dal grafico sopra riportato è evidente come l'errore ottenuto sulla stima del disturbo, evidenziato in rosso, non sia istantaneamente trascurabile, ma abbia bisogno di un certo tempo per assestarsi, questo fenomeno è più evidente in questa simulazione rispetto alle precedenti in quanto inizialmente il rumore extra prevale sul disturbo causando una mancata convergenza dei coefficienti del filtro, per poi assestarsi qualche istante a seguire, anche con l'aumentare dell'intensità del disturbo, vediamo qui sotto un ingrandimento della fase iniziale della simulazione:
É quindi possibile osservare come l'errore commesso dall'algoritmo, evidenziato in rosso, si assesti a valori ben inferiori ai picchi del disturbo, evidenziato in blu, dopo un tempo di circa un decimo di secondo.

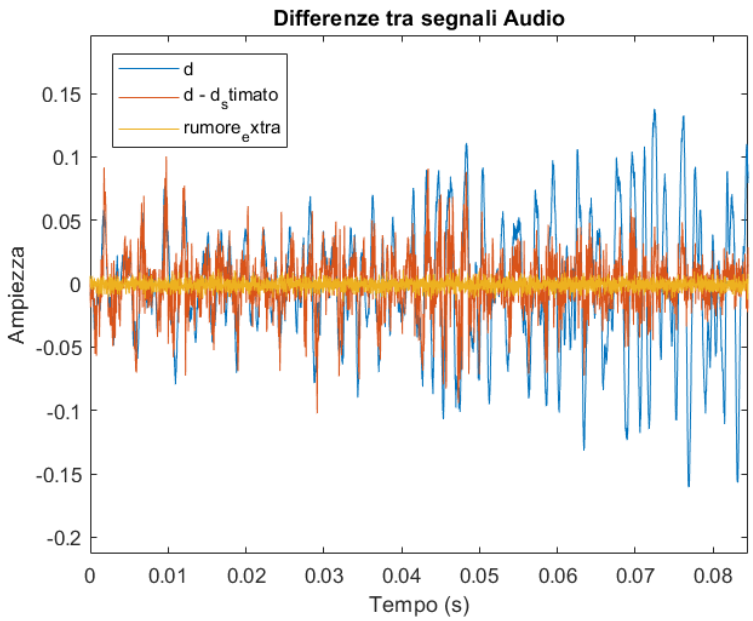

Figura 4.9: Fase iniziale della simulazione RPEM del tagliaerba.

Per questa simulazione si è ottenuto un fit di 0.989726, anch'esso in linea con i risultati ottenuti in precedenza.

4.4.2 Simulazione del rumore di un tagliaerba con algoritmo LMS

Si passa infine alla medesima simulazione, ricavata a partire dal rumore del tagliaerba, ma questa volta utilizzando il filtro predittivo *LMS*. I risultati ottenuti sono riportati in Fig. 4.10. Risulta possibile vedere dall'immagine qui riportata che la predizione inizia ad essere efficace dopo un intervallo di tempo più lungo. Si riporta anche in questo caso un ingrandimento della fase di assestamento dei coefficienti. É possibile notare che il transitorio questa volta è ben superiore al caso visto in precedenza, difatti quest'ultimo non è più dell'ordine di un decimo di secondo, ma di almeno 4 o 5 decimi. Il fit risultante da questa simulazione è pari a 0.96792, valore inferiore a quello ottenuto in precedenza con l'algoritmo *RPEM*. Dalle seguenti analisi si è quindi visto come adottare un approccio dalla maggiore complessità computazionale possa, in generale, portare ad una più rapida convergenza e adattabilità a nuove sorgenti di rumore, nonché ad un errore quadratico medio di predizione considerevolmente inferiore.

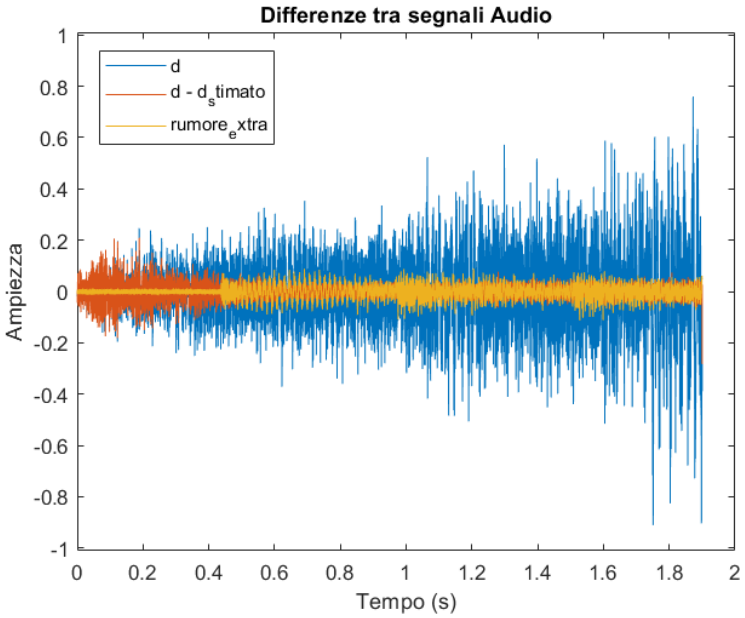

Figura 4.10: Risultato della cancellazione LMS del rumore del tagliaerba.

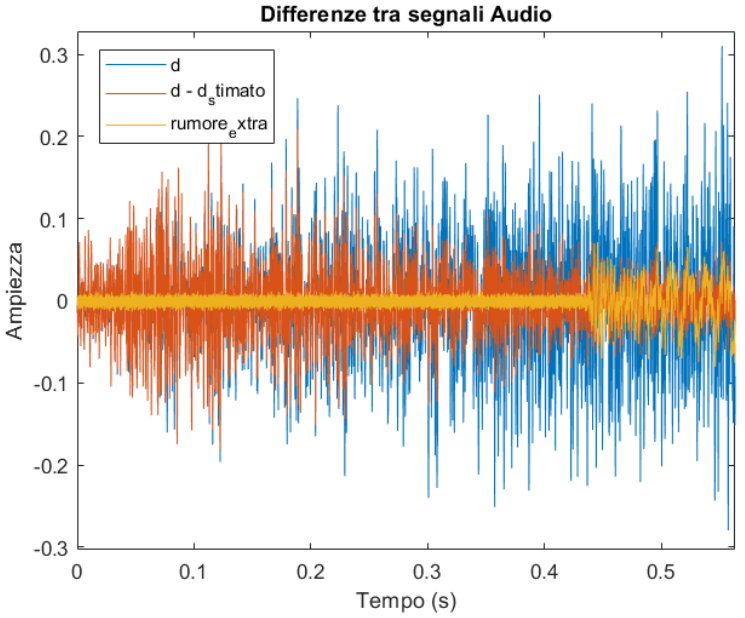

Figura 4.11: Fase iniziale della simulazione LMS del tagliaerba.

4.5 Valutazione di memoria occupata e velocità di calcolo

Alla analisi effettuate in precedenza sull'algoritmo *RPEM* si è deciso di analizzare anche la memoria occupata e la velocità di calcolo in funzione di λ, nA ed nB. I risultati otte-

nuti sono riportati in Fig. 4.12:

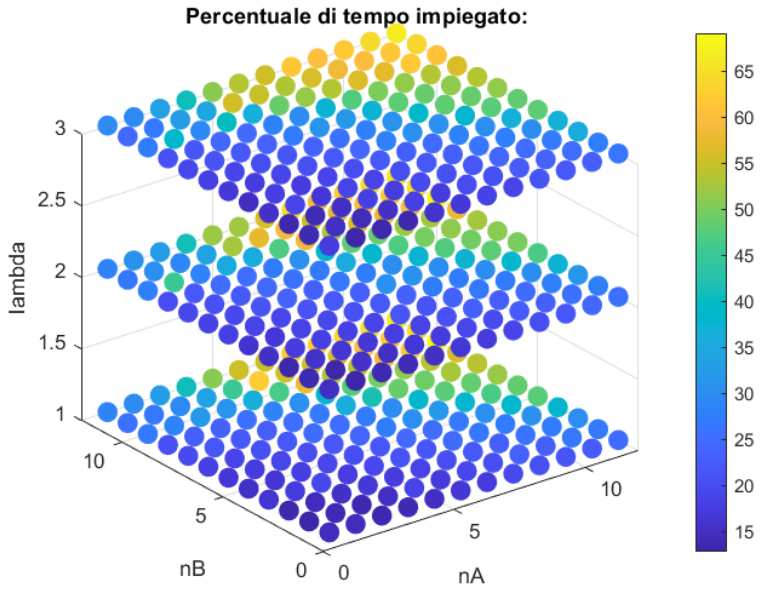

Figura 4.12: Percentuale di tempo impiegato per eseguire i calcoli della RPEM.

In virtù delle ipotesi effettuate all'inizio della trattazione si è scelto di lavorare con file *.mp3*, ne consegue che tra due istanti di campionamento successivi intercorrano circa $t_{max} = 22.7us$, dovuti al fatto che la frequenza di campionamento è fissata a 44100 Hz. Si è quindi deciso di misurare il tempo medio di calcolo impiegato da $RPEM$ e confrontarlo con $t_{max} = 22.7us$ al variare dei coefficienti, come detto in precedenza. Dal grafico è possibile notare, come di fatto era intuibile, che all'aumentare del numero di coefficienti che compongono il filtro, nA ed nB, aumenti conseguentemente anche il tempo di calcolo, questo è dovuto alle operazioni tra vettori, matrici, e soprattutto all'inversione della matrice S_t, che nel caso preso in esame assumerebbero dimensioni più elevate. É inoltre importante notare come questi numeri siano solo indicativi in quanto non ci si deve scordare che i calcoli in un contesto reale non verrebbero svolti da un PC, ma da un microprocessore di potenza di calcolo e dimensioni molto inferiori, impiegando quindi un tempo più lungo. In aggiunta a questo le acquisizioni non avvengono istantaneamente e potrebbero influire notevolmente sul tempo disponibile rimasto, nonché occupare il microprocessore per task talvolta differenti dal solo calcolo qui previsto. In ogni caso, a discapito delle ipotesi molto generali che si sono adottate, si è voluto avere una vaga idea sulle percentuali ottenute. Oltre a questo, si è inoltre analizzata la memoria occupata dalle variabili utilizzate da questo algoritmo, (Fig. 4.13).
Come è possibile osservare la memoria aumenta all'aumentare di nA ed nB, come nel caso visto del tempo impiegato, questo ancora una volta è dovuto all'incremento nelle

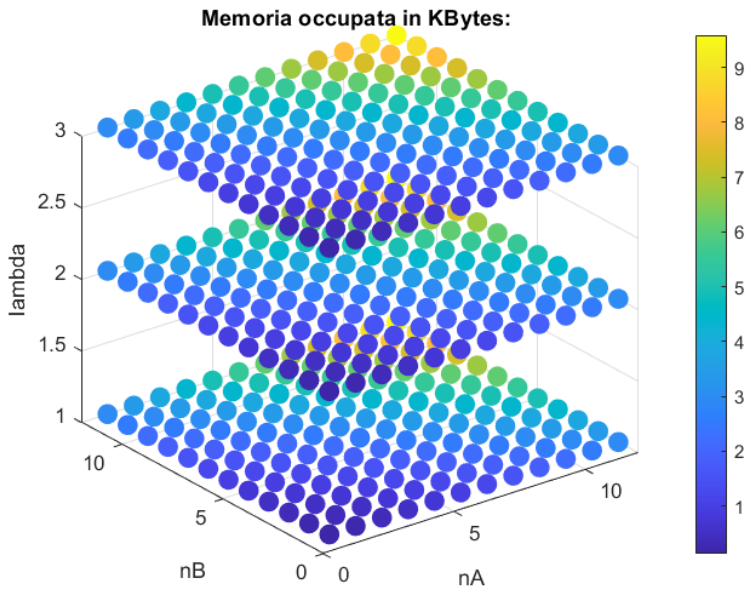

Figura 4.13: Memoria impiegata per svolgere i calcoli della RPEM.

dimensioni di vettori e matrici. Svolgendo un paio di simulazioni ci si accorge che la memoria, in questi approcci di tipo ricorsivo, non aumenta all'aumentare del tempo simulato, valido motivo per il quale aver esteso la trattazione teorica nei capitoli precedenti al fine di ottenere algoritmi ricorsivi, non saturando la memoria e rendendo di conseguenza questi concetti applicabili al caso reale. I risultati ottenuti dimostrano come la memoria sia confinata ad un massimo di una decina di KBytes, requisito soddisfabile anche da dispositivi di dimensioni ridotte.

Conclusioni

Attraverso l'indagine condotta in questa ricerca ed il confronto proposto tra gli algoritmi predittivi *RPEM* ed *LMS* è stato possibile visualizzare mediante le simulazioni effettuate come ognuna di esse presenti pregi e difetti. Nel caso dell'algoritmo *LMS* si è notato come questo, utilizzando delle ipotesi semplificative, si presti maggiormente ad applicazioni che dispongano di una limitata potenza di calcolo e necessitino quindi di un tempo di elaborazione minore, questo è per lo più dovuto al fatto che non sono previste inversioni matriciali nell'algoritmo. Oltre a questo si è notato come questo tipo di approccio necessiti di una scelta a priori del parametro μ: per valori troppo elevati di questo coefficiente non si assicura la convergenza, mentre, per valori troppo bassi si rischia di compromettere l'efficienza dell'algoritmo. Perciò è complicato rendere questo algoritmo adattabile a variazioni tra rumori di intensità differenti. Nel caso dell'algoritmo *RPEM* invece si è notato come la necessità di una mole più ampia di calcoli porti ad una maggiore precisione e rapidità nella convergenza, assicurando prestazioni più efficaci nella cancellazione attiva del rumore. Un altro aspetto che si è riscontrato è la non diretta correlazione tra un aumento del numero di coefficienti che descrivono il filtro adattivo ed un miglioramento del fit. Questo fenomeno è dovuto al fatto che all'aumentare del numero di coefficienti ne corrisponde un *overfitting* del dataset di allenamento, perdendo la capacità di generalizzare su dati di natura simile. Questo potrebbe quindi comportarne un miglior adattamento agli errori del *training dataset* ed una conseguente più scarsa precisione nel *testing dataset*. Al contrario, un numero troppo basso di coefficienti non sarebbe in grado di descrivere in modo completo la dinamica del filtro, sfociando anche in questo caso in un fit non ottimale, questo fenomeno passa sotto il nome di *underfitting*. In conclusione, i risultati più rilevanti ottenuti dalle simulazioni sono i seguenti riportati nella seguente tabella:

FIT	Rumore simulato	RPEM	LMS
Training dataset	Macchina + Vento + Treno	96.55%	X
Testing dataset	Phon	99.19%	X
	Martello pneumatico	96.77%	X
Confronto	Tagliaerba	98.97%	96.79%

Figura 4.14: Fit ottenuti dalle simulazioni.

Risulta quindi evidente che a livello teorico e ora anche pratico, che la soluzione più adatta alla cancellazione attiva del rumore implementata per le cuffiette bluetooth è quella che impiega l'algoritmo *RPEM*. Tuttavia in un contesto reale, data la minor potenza di calcolo richiesta ed il conseguente minor consumo di energia potrebbe comunque essere necessario o preferibile in alcuni casi utilizzare l'algoritmo *LMS*.

Bibliografia

[1] Mattia Zorzi. *Lectures on System Identification*. Independently published, 2020.

[2] Augusto Ferrante. *Appunti di Automatica per Ingegneria Biomedica con esercizi e temi d'esame risolti*. Edizioni Progetto, Padova, 2023.

[3] Alessandro Carlotto. Modellizzazione e controllo per la cancellazione attiva del rumore. URL `https://thesis.unipd.it/handle/20.500.12608/52555`.

[4] S. Mitra S. M. Kuo and Woon-Seng Gan. Active noise control system for headphone applications. URL `https://ieeexplore.ieee.org/document/1597204`.

[5] MathWorks. Adaptive filter convergence for headphone applications. URL `https://it.mathworks.com/help/dsp/ug/adaptive-filter-convergence.html`.
[1] [2] [3] [4] [5]