FACOLTÀ DI INGEGNERIA

TESI DI LAUREA

SIMULAZIONE NUMERICA DEL CONTRIBUTO ACUSTICO DELLA TESTA NELL'ASCOLTO SPAZIALE: ANALISI DEI DATI

Laureando: Francesco Foscarin

Relatore: Federico Avanzini
Correlatore: Simone Spagnol

Corso di Laurea Triennale in Ingegneria dell'Informazione

Data di laurea 26/9/2013

Anno Accademico 2012/2013

Prefazione

Diversi sono i fattori che permettono la spazializzazione del suono mediante audio binaurale (ossia basato sulla riproduzione tramite cuffie), ma la maggior parte delle tecniche di rendering binaurale attualmente utilizzate in ricerca fanno affidamento sull'uso delle cosiddette Head-Related Transfer Functions (HRTFs), ovvero particolari filtri che catturano le trasformazioni subite da un'onda sonora nel proprio percorso dalla sorgente al timpano, generalmente dovute a effetti di riflessione e diffrazione sul torso, sulla testa, sulle spalle e sui padiglioni auricolari dell'ascoltatore.

Su questo campo è incentrato anche il lavoro svolto in questa tesi. In particolare si studierà il contributo acustico della testa del manichino KEMAR: saranno simulate delle HRTFs con *AcouSTO*, un software di simulazione numerica, tramite *Boundary Element Method*. Saranno poi confrontate con altre HRTFs registrate, per valutare quanto la simulazione possa essere attendibile.

Sommario

Il concetto di audio 3d è sconosciuto ai più. Ma si tratta di un fattore di primaria importanza nel nostro modo di rapportarci con l'ambiente che ci circonda. Facciamo un semplice esempio: immaginiamo di essere in un treno ad occhi chiusi e di provare a percepire quello che accade attorno a noi solamente ascoltando. Sentiremmo non solo quello che stanno dicendo le persone, ma riusciremmo a capire anche in che direzione parlano e quanto distanti sono da noi. Inoltre potremmo sentire altri suoni e capire se giungono dall'interno o dall'esterno del nostro vagone. Tutte queste informazioni vanno ben oltre il puro suono prodotto dalla fonte e ci permettono di ricostruire la realtà tridimcquisire tutti questi dati e di simulare la spazialità dei suoni attraverso cuffie o altoparlanti. Le applicazioni di una tale tecnologia si possono facilmente immaginare in campo cinematografico e musicale, ma anche, ad esempio, a scopo medico e riabilitativo.

I primi studi in questo campo risalgono ai primi anni del Novecento quando John Strutt, meglio noto come Lord Rayleigh, aveva tra i primi cominciato a studiare in modo sistematico la percezione del suono nello spazio. Negli anni successivi sono proseguiti, conseguendo numerosi successi, gli esperimenti e gli studi in questo ambito, fino a giungere ai giorni nostri, quando la disponibilità di strumenti hardware e software sempre più potenti e raffinati ha aperto nuove strade alla ricerca.

Lo scritto si divide quindi in tre capitoli: il primo contestualizza il problema della spazializzazione del suono, il secondo verte sul software *AcouSTO* e sulla simulazione e il terzo tratta l'analisi dei dati.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il suono nello spazio

1.1 Percezione del suono nello spazio

È noto dalla letteratura [2] che le parti anatomiche dell'uomo interferiscano con le onde sonore emesse da una sorgente acustica principalmente a causa dei fenomeni di riflessione e diffrazione. In particolare gli elementi che maggiormente influenzano la nostra percezione del suono sono: la testa, l'orecchio esterno e il torso.

La nostre orecchie non sono oggetti isolati nello spazio, ma si trovano ai lati della testa: un corpo che funge da ostacolo alla libera propagazione dei suoni. Le conseguenze principali sono due:

- presenza di una *Interaural Time Diffference (ITD)*, cioè un ritardo nella ricezione del suono da parte di un orecchio rispetto all'altro, a causa della finita velocità del suono e della distanza non nulla che separa le due orecchie;

- presenza di una *Interaural Level Difference (ILD)*, cioè una differenza di intensità del suono percepita da un orecchio rispetto all'altro, dato che la testa agisce come un ostacolo che oscura parte del suono.

Per orecchio esterno si intende la pinna e il canale uditivo, fino al timpano (figura 1.1). La particolare forma dell'orecchio esterno comporta una trasformazione del suono: le sue cavità risonanti provocano l'amplificazione di alcune frequenze e alcune caratteristiche geometriche portano fenomeni di interferenza che attenuano altre frequenze. Inoltre questi comportamenti dipendono dalla direzione da cui provengono i suoni (vedi figura 1.2). L'orecchio esterno si comporta quindi come un filtro, la cui funzione di trasferimento varia in base alla direzione da cui provengono le onde acustiche [7].

Il busto e le spalle invece contribuiscono alla spazializzazione del suono in due modi principali: aggiungono ulteriori effetti di riflessione e oscurano parte del suono quando le onde acustiche provengono dal basso. Nonostante la geometria del torso sia abbastanza complicata, può essere approssimata considerando il torso ellissoidale e la testa sferica (snowman models). La figura 1.3

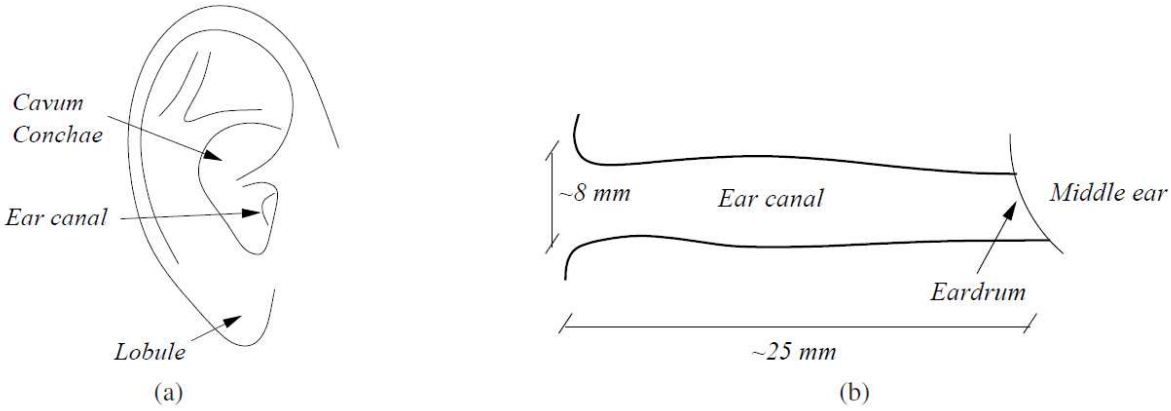


Figura 1.1: *Orecchio esterno: (a) pinna, (b) canale uditivo*

mostra i due effetti del torso; spostando verticalmente dall'alto verso il basso un'ipotetica sorgente, si può constatare come al di sotto di una certa altezza le riflessioni scompaiano e cominci a manifestarsi l'effetto di mascheramento da parte del busto (figura 1.3).

Il contributo di busto e spalle non è rilevante quanto quello introdotto dall'orecchio esterno; tuttavia non va trascurato perchè diventa significativo alle basse frequenze, dove molti suoni hanno la maggior parte della loro energia e dove la risposta dell'orecchio esterno è praticamente piatta . I due contributi si possono quindi considerare in prima approssimazione complementari tra loro.

1.2 Head Related Transfer Functions

Tutti gli effetti esaminati nel precedente paragrafo godono di un'importante proprietà che permette di proseguire nel loro studio: sono lineari. Questo implica due cose: possono essere descritti da una funzione di trasferimento e si possono combinare attraverso una semplice somma.

Le HRTF mettono in relazione i suoni misurati in una posizione all'interno del canale uditivo con i suoni che sarebbero misurati nella stessa posizione se non fosse presente la testa. In termini più tecnici: se la testa è centrata nel punto *P* e la sorgente ha elevazione φ, azimut θ e distanza *r*, in un sistema di cordinate sferiche centrato nella testa, allora la HRTF $H(r, \theta, \varphi, f)$ è il rapporto tra la trasformata di Fourier del segnale all'orecchio $F_e(f)$, con il segnale che sarebbe stato ricevuto al punto P in campo libero $F_p(f)$:

$$H(r, \theta, \varphi, f) = \frac{F_e(f)}{F_p(f)}$$

La figura 1.4 mostra il grafico di una HRTF.

Ci sono due modi per ottenere le HRTFs[5]: diretto e indiretto. Il primo consiste nel far muovere una sorgente (speaker) e nel misurare il segnale con un microfono posizionato all'ingresso del canale uditivo. Nel secondo invece la sorgente ad essere posizionata all'ingresso del

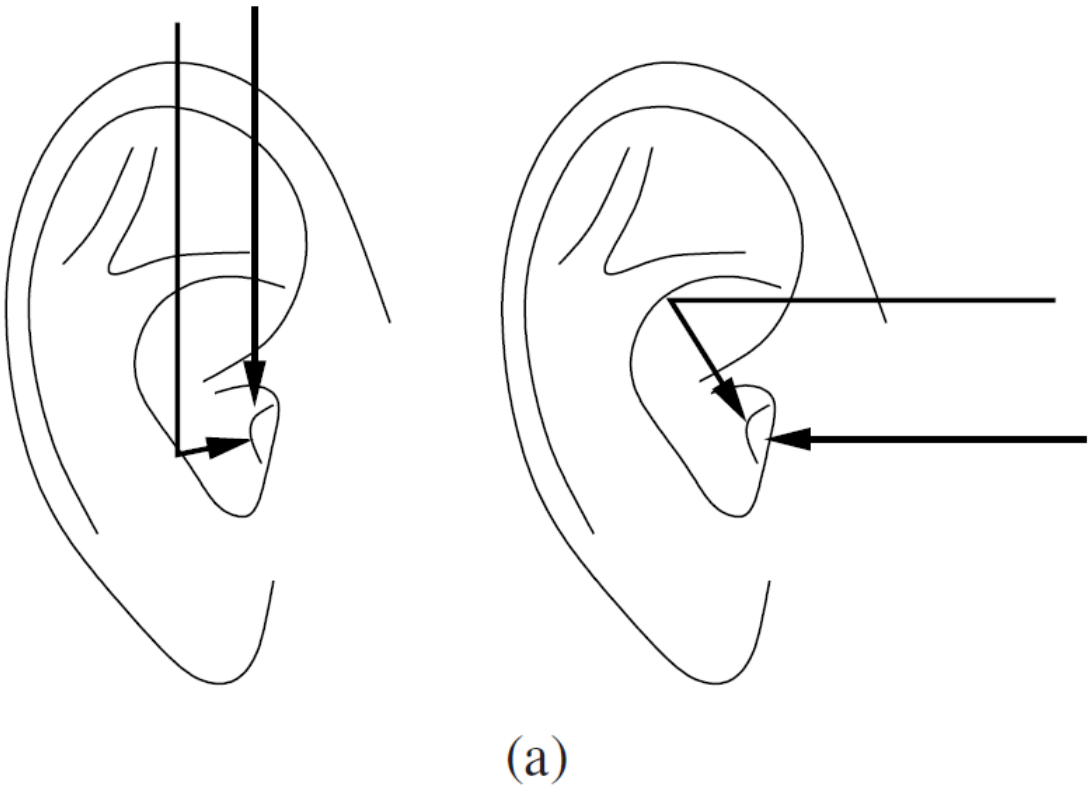


Figura 1.2: *Dipendenza dalla posizione della sorgente*

canale uditivo, i microfoni sono in vari punti dello spazio e le HRTFs sono determinate usando il principio di reciprocità di Helmholtz (un approfondimento sulla questione si trova al paragrafo 2.4).

1.3 Perchè simulare le HRTFs?

Come abbiamo visto, queste funzioni dipendono dalla geometria di testa, spalle e orecchie, la loro affidabilità viene quindi a mancare se esse sono utilizzate su un soggetto diverso da quello su cui sono state misurate. Ma la misurazione delle HRTFs e' molto dispendiosa, in termini di tempo e di attrezzature necessarie. Inoltre una persona dovrebbe rimanere pressochè immobile per tutto il processo, situazione quasi irrealizzabile. La soluzione a questi problemi si trova quindi nella simulazione numerica di queste funzioni [3], che permette di ottenere dei risultati senza dover dipendere da un soggetto fisico e permette di poter variare la geometria operarando facilmente cambiamenti anche drastici, ad esempio rimuovere completamente un orecchio. Diventa quindi più facile anche in fase di studio capire la dipendenza delle HRTFs dalle diverse caratteristiche strutturali della testa e della pinna.

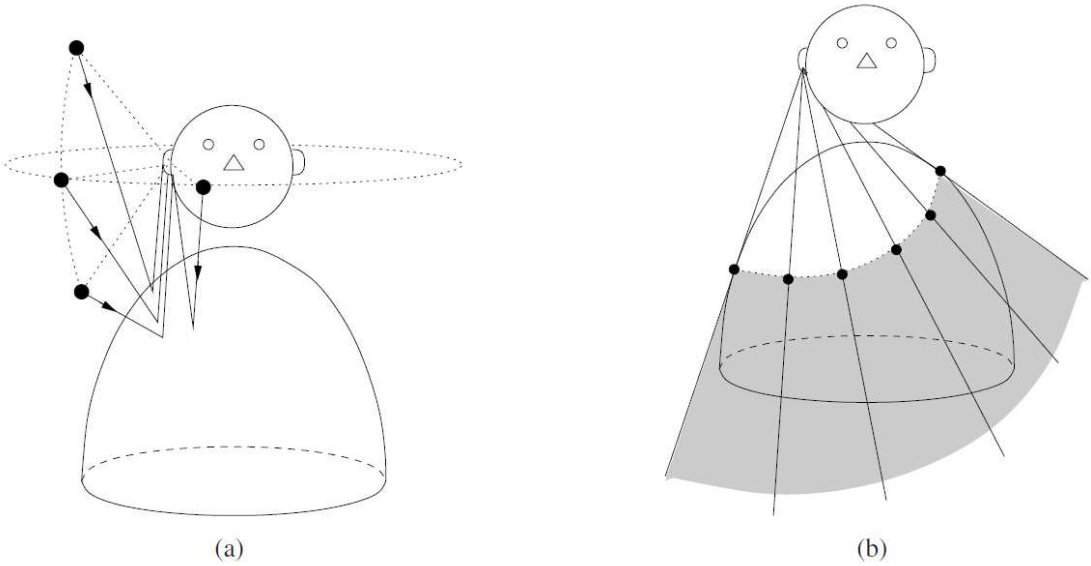


Figura 1.3: *Contributo del torso: (a) riflessione (b) mascheramento*

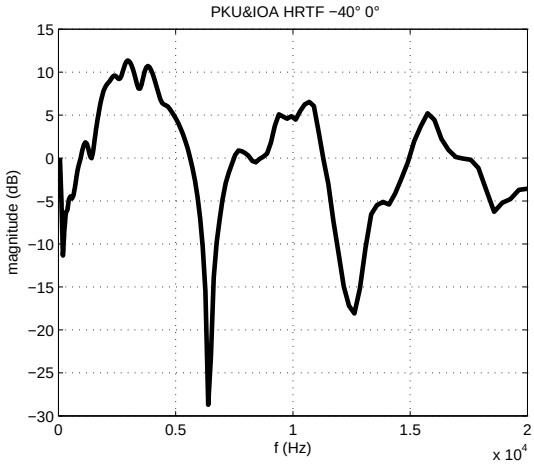


Figura 1.4: *Grafico di HRTF*

Capitolo 2

Boundary Element Method

2.1 Simulazioni di HRTFs

Per eseguire simulazioni di HRTFs esistono numerosi metodi [3], che variano in complessità e in costo computazionale. Il metodo preso in considerazione in questa tesi prende il nome di Boundary Elementh Method (BEM), considera la testa come una mesh di elementi discreti. Fondamentale è l'ipotesi semplificativa che solo la superfice della testa sia da tenere in considerazione; inoltre tale superficie viene considerata come se avesse un'impedenza acustica infinita, quindi tutte le propagazioni attraverso la testa sono ignorate.

2.2 Problema matematico

Il problema acustico [8] è scritto in termini della funzione potenziale di velocità φ nel dominio di Laplace

$$\nabla^2\widetilde{\varphi}(x) - k^2\widetilde{q}(x) = \widetilde{q}, \qquad x \in \Omega \tag{2.1}$$

dove $\widetilde{q}$ rappresenta la sorgente acustica e $k = s/c_0$ è il numero d'onda complesso, essendo $s = \alpha + j\omega$ la variabile di Laplace e c_0 la velocità del suono nelle condizioni di riferimento. Il problema è completato dalle condizioni al contorno per $x \in \partial\Omega$. L'equazione 2.1 vale sia per il potenziale di velocità che per la perturbazione di pressione. Il significato fisico della soluzione è lasciato come scelta all'utilizzatore, che deve completare con le adeguate condizioni al contorno. Assumendo $\widetilde{\varphi}$ come potenziale di velocità e ponendo $\widetilde{q} = 0$, la formulazione integrale sulla frontiera per $\widetilde{\varphi}$ si può scrivere come

$$E(y)\widetilde{\varphi}(y) = \oint_S \left(G\frac{\partial\widetilde{\varphi}}{\partial n} - \widetilde{\varphi}\frac{\partial G}{\partial n} \right) dS(x) \tag{2.2}$$

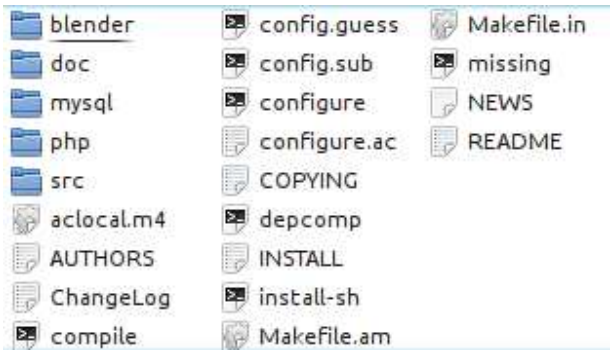


Figura 2.1: *Gerarchia dei file sorgente di AcouSTO.*

dove la funzione di dominio $E(y)$ è

$$\begin{cases} 1 & if \quad y \in \Omega \\ 1/2 & if \quad y \in \partial\Omega \\ 0 & if \quad y \notin \Omega \end{cases} \tag{2.3}$$

e $S = \partial\Omega$ per i problemi interni e $S = \partial\Omega/S_\infty$ per i problemi esterni. Richiamando l'espressione 2.1, la 2.2 diventa

$$E(y)\widetilde{\varphi}(y) = \oint_S \left(G\frac{\partial\widetilde{\varphi}}{\partial n} - \widetilde{\varphi}\frac{\partial G_0}{\partial n} + s\widetilde{\varphi}G_0\frac{\partial\theta}{\partial n} \right) e^{-s\theta} dS(x) \tag{2.4}$$

L'eq 2.4 è usata come una rappresentazione integrale al contorno per $\widetilde{\varphi}$ in un punto arbitrario nello spazio (i microfoni) come funzione della distribuzione di $\widetilde{\varphi}$ sulla frontiera.

L'equazione 2.4 è risolta per via numerica attraverso il Boundary Element Method. La frontiera del dominio è partizionata in N quadrilateri e tutte le quantità sono considerate costanti all'interno di ogni pannello. L'integrale di superficie nell'equazione 2.4 è approssimato con una somma di N pannelli.

2.3 AcouSTO

AcouSTO (Acoustic Simulation TOol) è un simulatore acustico che utilizza la tenica Boundary Element Method (BEM) per risolvere l'equazione integrale di Kirchhoff-Helmholtz. È opensource e può essere scaricato gratuitamente all'indirizzo `http://acousto.sourceforge.net/`.

Per le simulazioni si è utilizzata la versione 1.5 del programma.

La figura 2.1 mostra la struttura del filesystem di *AcouSTO*. Oltre ai file di configurazione degli *autotools* e ad altri file di testo sono presenti le seguenti cartelle:

1. **blender/** — contenente lo script di esportazione delle mesh dal programma di modellazione 3D *Blender*.

2. **doc/** — contenente il manuale utente e la documentazione *doxygen/*.
3. **mysql/** — contenente uno script *SQL* per impostare le tabelle in caso di utilizzo di backend *DBMS MySQL* per il salvataggio dei risultati delle simulazioni.
4. **php/** — contenente un insieme di script *PHP* che permettono di interfacciarsi con i dati salvati in *MySQL* tramite browser.
5. **src/** — contenente il sorgente vero e proprio di *AcouSTO*.

AcouSTO è suddiviso in moduli, ciascuno racchiuso in un file `.c` diverso. I più importanti sono:

- `ac_coef_body.c` — file di implementazione del precalcolo dei coefficienti integrali della superficie.
- `ac_coef_mics.c` — file di implementazione del precalcolo dei coefficienti integrali dei microfoni.
- `ac_gmres.c` — file di implementazione del risolutore iterativo di sistemi lineari.
- `geom.c` — modulo geometrico.
- `geom_utils.c` — file di implementazione delle funzioni relative alla generazione delle geometrie definite nel file di configurazione.
- `main.c` — core di *AcouSTO*. Si occupa di inizializzare la libreria *MPI*, leggere il file di configurazione, richiamare i moduli necessari e terminare correttamente l'esecuzione.
- `mysqlsave.c` — modulo di interfacciamento con il database *MySQL*.
- `nrwash.c` — file di implementazione delle routine che calcolano il campo incidente e le condizioni al contorno.
- `solution.c` — modulo che si occupa della risoluzione del problema.
- `vtkout.c` — modulo di output della soluzione in formato *Visualization ToolKit* (vtk) e della geometria in formato VRML2. I file generati sono molto utili per la visualizzazione dei risultati in programmi come *paraview*.

2.3.1 Caratteristiche

AcouSTO presenta molte funzionalità interessanti, tra le quali:

1. Gestione delle simmetrie per ridurre il tempo di calcolo
2. Esecuzione parallela su cluster MPI2
3. Sistema di gestione della memoria che consente di controllare la quantità di RAM allocata

4. Gestione di modelli 3D di geometria arbitraria
5. Supporto del DBMS MySQL
6. Nessun limite software sul numero di onde incidenti o sorgenti puntiformi
7. Possibilità di eseguire simulazioni si all'interno che all'esterno di un modello 3D.

I problemi che esso può risolvere sono di due tipi:

1. Scattering di onde piane o sferiche dovuto a molteplici corpi di forma arbitraria
2. Irraggiamento di superfici chiuse vibranti

Per svolgere le simulazioni esso si avvale di librerie standard e ampiamente testate quali *ScaLAPACK*, *BLAS* e *MPI-2*.

2.3.2 Installazione in ambiente Ubuntu Linux

Come prerequisito è necessario scaricare il sorgente dall'indirizzo `http://sourceforge.net/projects/acousto/files/acousto/1.5/acousto-1.5.tar.gz/download`.

A causa del build system personalizzato utilizzato in *AcouSTO*, prima di procedere alla compilazione dei sorgenti è necessario eseguire alcuni passi:

1. Installazione dei pacchetti necessari:

```
> sudo apt-get install gfortran
> sudo apt-get install libconfig8 libconfig8-dev
> sudo apt-get install openmpi-bin
> sudo apt-get install libblacs-mpi1 libblacs-mpi1-dev
> sudo apt-get install libscalapack-mpi1
libscalapack-mpi1-dev
```

2. Creazione dei symlink necessari affinchè *AcouSTO* rilevi la libreria `scalapack`:

```
> cd /usr/lib
> sudo ln -s libscalapack-openmpi.a libscalapack.a
```

   In caso di sistemi diversi da Ubuntu o di distribuzioni *ScaLAPACK* con nomenclatura differente i comandi precedenti vanno modificati adeguatamente.

3. Compilazione:

```
> ./configure
> make
> sudo make install
```

In caso di errori derivanti da versioni datate degli `autotools` è utile eseguire il comando `autoreconf` prima del passo precedente. Questo programma esegue gli *autotools* ripetutamente per aggiornare il *build system* nella directory corrente e nelle sotto-directory in maniera ricorsiva.

2.3.3 Complessità e costo computazionale

La simulazione mediante BEM è molto potente, ma allo stesso tempo è molto costosa dal punto di vista computazionale. Come abbiamo visto il principio alla base è la risoluzione di una serie di equazioni. Ogni equazione contiene un termine per ogni elemento della mesh e c'è un'equazione per ogni elemento. Con il crescere degli elementi della mesh il costo computazionale cresce dunque in modo quadratico; nel caso particolare di *AcouSTO*, per problemi di tipo implementativo, la complessità diventa $O(n^3)$. Con costo computazionale si intendono più precisamente due elementi: il primo è ovviamente il tempo necessario ad eseguire il calcolo, il secondo è la quantità di memoria di storage richiesta al computer, dato che le equazioni del problema devono essere memorizzate in vettori e matrici. Esistono diverse varianti della BEM che ottimizzano il primo o il secondo aspetto.[3]

Un possibile compromesso tra tempo di calcolo e memoria richiesta è dato dal parametro `pre_calculated_coefs`: se viene impostato a 0 i coefficienti integrali vengono calcolati e scartati di volta in volta, mentre se impostato a 1 essi vengono precalcolati e tenuti in memoria.

Ne risulta che nel caso in cui un coefficiente venga riutilizzato più di una volta il fatto di averlo precalcolato porta ad un leggero miglioramento delle prestazioni. Tuttavia, come indicato in [8, par. 4.5.2], questa eventualità non è molto frequente ed è quindi consigliabile lasciare `pre_calculated_coefs` a 0.

È anche importante notare che nelle versioni più recenti di *AcouSTO* sono presenti due risolutori. Il primo, chiamato *PSEUDOINV*, è il risolutore di sistemi lineari integrato nella libreria *ScaLAPACK* e invocato dal file `linsys.c`, mentre il secondo, *GMRES*, è un risolutore iterativo sviluppato da zero dagli autori del programma ed implementato nel file `ac_gmres.c`. Sebbene *GMRES* non sia stato testato sufficientemente, questo tipo di risolutori fornisce delle alte prestazioni di calcolo parallelo con matrici dense, che sono proprio il tipo di matrici con cui lavora *AcouSTO*.

Un miglioramento delle prestazioni si può ottenere anche con un'attenta impostazione della simulazione. In quella esaminata sono state valutati principalmente due aspetti. Il prima riguarda le simmetrie: con un'approssimazione accettabile nel caso del manichino (meno verosimile se si fosse utilizzata una vera testa umana) si può considerare la testa simmetrica rispetto ad un piano verticale. Questo di fatto dimezza le equazioni e i termini da considerare; inoltre in questo modo la simulazione viene eseguita per un solo orecchio, considerando i risultati uguali per le due orecchie. La seconda ottimizzazione riguarda il cosiddetto principio di reciprocità ed è trattata nel paragrafo che segue.

2.4 Principio di reciprocità e inversione del problema acustico

Idealmente, per effettuare le misurazioni desiderate, sarebbe necessario predisporre un microfono in ciascun orecchio del manichino. Dopodiché, una singola sorgente sonora andrebbe spostata nello spazio per osservare le modifiche di ampiezza e fase al variare di distanza, azimut ed elevazione.

C'è però un altro modo per ottenere gli stessi risultati riducendo il costo computazionale: usando un teorema analogo a quello di Maxwell-Betti per l'acustica possiamo scrivere

$$p_{x_1}(x_2) = \frac{-qp_{x_2}(x_1)}{i\omega\rho A_0 v_{n_0}}$$

dove A_0 è l'area dell'elemento vibrante, v_{n_0} è la velocità normale al punto medio di x_1, ρ è la densità del mezzo e q è l'intensità della sorgente posizionata a x_2. Cioè esiste una semplice relazione algebrica tra la pressione sonora $p_{x_1}(x2)$ causata da un'eccitazione al punto x_1 con la pressione sonora p_{x_2} causata da un'eccitazione al punto x_2.

Al posto di effettuare una simulazione per ogni punto di interesse è sufficiente una singola simulazione invertendo la sorgente con il ricevitore: nel nostro caso metteremo quindi la sorgente (altoparlante) nell'orecchio e il ricevitore (microfono) nei diversi punti di interesse dello spazio.

2.5 Scenari

Per la simulazione della risposta in frequenza di una testa umana (HRTF) è stato usato il modello 3D di un manichino KEMAR per test acustici appartenente al gruppo di ricerca *SMC* (*Sound Music Computing*) e utilizzato dal team di tecnologie binaurali.

Il modello è stato ottenuto tramite scansione laser di un manichino reale.

L'acquisizione della testa del manichino *KEMAR* è stata fatta con lo scanner 3D *NextEngine 3D Scanner HD*[1] visibile in figura 2.2.

Figura 2.2: *Scanner 3D.*

La simulazione prevede la presenza di una sorgente puntiforme nell'orecchio destro (2.3) e di una griglia di 793 microfoni disposti a 160cm dal centro della testa. Il tutto è rappresentato in figura 2.4.

La sorgente ha coordinate, in metri, $(-0.06698, 0.000288, 0.0)$ e la sua distanza minima dalla testa è di circa $1.024\,\mathrm{mm}$.

Figura 2.3: *Sorgente puntiforme nell'orecchio destro del manichino.*

Il sistema di riferimento di azimut ed elevazione differisce da quello usato in [6] (figura 2.5) ed è mostrato in figura 2.6. Guardando la testa dall'alto, l'azimut si riferisce all'angolo antiorario tra l'orecchio sinistro (visto dal manichino). Esso varia da 0 a 360.

Per quanto riguarda l'elevazione, si riferisce al piano orizzontale. Viene fatta variare tra -40 e 90 a passi di 10 gradi.

Si tratta quindi di un sistema polare-verticale con $\theta = [0, 360)$ e $\phi = [-40, 90]$.

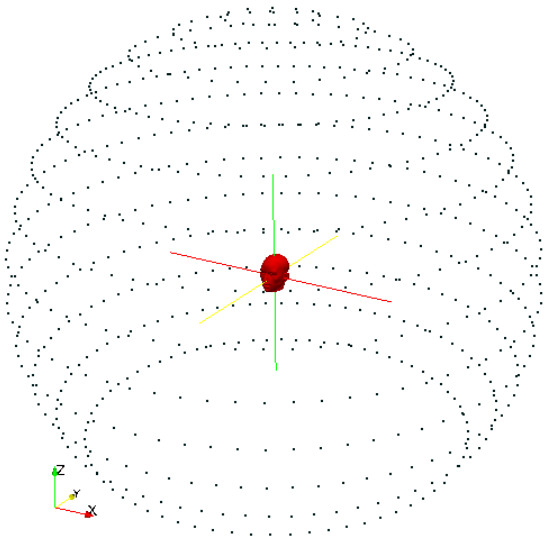


Figura 2.4: *Griglia di 793 microfoni.*

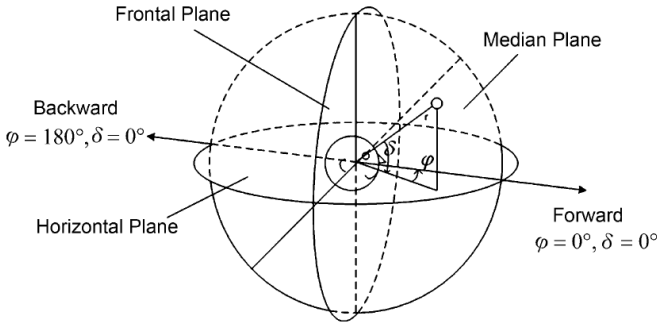


Figura 2.5: *Coordinate sferiche verticali polari usate da [6] per i microfoni. Azimut ϕ ed elevazione δ.*

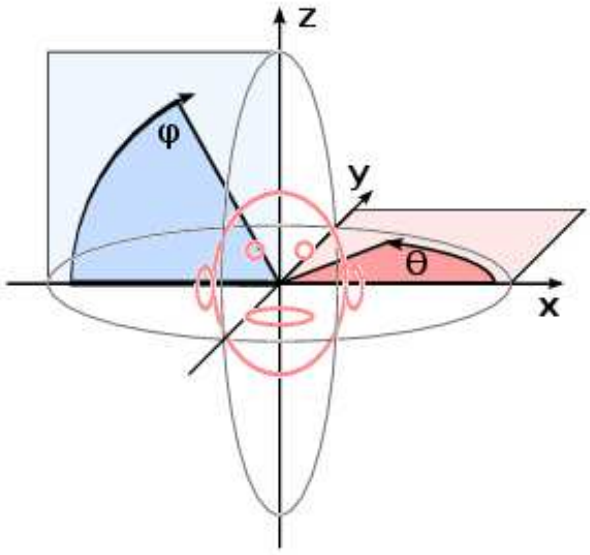


Figura 2.6: *Coordinate sferiche verticali polari usate per i microfoni. Azimut ϕ ed elevazione θ.*

Capitolo 3

Analisi dei Dati

L'analisi dei dati è stata effettuata con matlab. In particolare sono stati scritti alcuni script per l'acquisizione e la regolarizzazione dei dati e per la stampa dei grafici. I dati simulati sono stati poi confrontati con il database di HRTF *PKU&IOA* (`www.cis.pku.edu.cn/auditory/Staff/Dr.Qu.files/Qu-HRTF-Database.html`)

3.1 Script

Gli script presentati sono 6:

- *clean_data* (A.2.1): esegue la 'pulizia' dei dati (maggiori informazioni si trovano di seguito nel capitolo);
- *display_data* (A.3.1): crea i grafici delle HRTFs, per una frequenza scelta oppure tutte insieme;
- *HRTFsPlotByElevation* (A.4.2), *HRTFsPlotByAzimuth* (A.4.1): creano dei grafici 3d delle HRTFs;
- *readhrir* (A.3.2): legge i dati dal database *PKU&IOA* e stampa i grafici delle HRTFs.
- *QuHRTFsplotByAzimuth* (A.4.3): legge i dati dal database *PKU&IOA* e stampa dei grafici 3d delle HRTFs

In particolare questi script sono stati usati (con qualche modifica) prima sulle HRTFs della sfera e successivamente su quelle della testa del manichino KEMAR.

Altri script sono usati per l'analisi dei dati, ma non presentano particolari elementi di interesse. Sono comunque presente nell'appendice.

Gli script *HRTFsPlotByElevation* e *HRTFsPlotByAzimuth* sono stati pensati basandosi su altri script già esistenti, ma a causa del diverso formato dei dati di input sono stati completamente riscritti.

Lo script *Readhrir* è stato preso dal database PKU&IOA e poi editato in modo da permettere il passaggio da risposta all'impulso a risposta il frequenza e il plot dei dati.

3.2 Unità di misura

Arrivati a questo punto è necessario fare una piccola precisazione sulle unità di misura, in modo da capire a cosa si riferiscono i valori numerici che saranno riportati più avanti nella simulazione. Parliamo innanzitutto di segnali discreti $x[n]$ la cui energia ϵ_x e potenza media P_x sono definite rispettivamente come segue:

$$\epsilon_x = \sum_{n=-\infty}^{\infty} |x[n]|^2 \tag{3.1}$$

$$P_x = \frac{\epsilon_x}{N} = \frac{1}{N} \sum_{N-1}^{n=0} |x[n]|^2 \tag{3.2}$$

Una misura comune per descrivere un segnale è il suo *root mean square (RMS)*, defininito semplicemente come $\sqrt{P_x}$.

Nel caso di segnali audio $x[n]$ denota generalmente una pressione acustica; inoltre, trattandosi di onde che viaggiano in un mezzo, si può considerare la potenza RMS distribuita su tutto il fronte d'onda. La forza dell'onda si misura quindi come potenza su unità di area ed è nota come *intensità* (I).

Ad ogni modo la definizione comune di *sound pressure level (SPL)* è espressa in termini logaritmici e si ricava dall'intensità secondo la formula seguente:

$$SPL = 10 \log_{10}(I/I_0) \quad (dB) \tag{3.3}$$

dove I_0 è un'intensità di rifermento, solitamente scelta come la più piccola intensità sonora che può essere udita.

Poichè l'intensità è direttamente proporzionale al quadrato pressione RMS è possibile scrivere l'equazione in termini di rapporti di pressione:

$$SPL_2 - SPL_1 = 10 \log_{10}(p_2^2/p_1^2) = 20 \log_{10}(p_2/p_1) \qquad (dB) \tag{3.4}$$

L'output di *AcouSTO* è il valore di pressione al ricevitore alle diverse frequenze. Riprendiamo per chiarezza la definizione di HRTFs di [2]: le HRTFs sono il rapporto tra l'SPL al timpano e l'SPL al centro della testa in campo aperto. Ponendo $p_1 = 1$ e considerando la SPL in campo aperto uguale ad uno, dato che ci troviamo in condizioni ideali, possiamo quindi considerare l'output di *AcouSTO* come HRTFs.

3.3 Problema di Dirichlet e pulizia dei dati

Questo metodo di simulazione per una superficie chiusa S è però affetto dal cosiddetto *fictious eigenfrequencies problem*. Il campo acustico nel dominio aperto che circonda S, mostra delle risonanze non-fisiche, che sono collegate alle autosoluzioni del problema interno definito nella cavità acustica delimitata da S. Più precisamente la soluzione del problema esterno definito dalle condizione al contorno di Neumann ha una singolarità alle autofrequenze del problema interno

di Dirichlet, mentre le autofrequenze del problema interno di Neumann sono collegate con le soluzioni del problema esterno di Dirichlet.

In breve, ci sono delle determinate frequenze per cui il calcolo delle HRTFs fallisce per cui il valore simulato non è corretto. Questi punti sono facilmente individuabili quando la funzione di trasferimento varia molto lentamente; più difficile è invece trovarli quando le variazioni sono veloci. Al fine di correggere questi errori si è quindi proceduto in due diversi modi nel caso della sfera e della testa.

3.3.1 Sfera

I test sulla sfera sono stati eseguiti a solo scopo di impratichirsi con il linguaggio matlab e con il formato dei dati, lavorando su un problema semplificato e già risolto e potendo facilmente confrontare le soluzioni con i risultati presenti in letteratura [4]. I dati utilizzati sono stati generati da *AcouSTO* in una simulazione fatta da un gruppo di ricerca dell'Universit di Roma 3.

Nella sfera, specialmente alle basse frequenze, le funzioni di trasferimento variavano molto lentamente. Si è quindi proceduto individuando i punti che si discostavano di molto rispetto all'andamento medio della curva, con un algoritmo molto semplice:

```
for bin = 2:(numel(data{i})-1)
      prima= data{i}(bin-1).Abs;
      mezzo= data{i}(bin).Abs;
      fine= data{i}(bin+1).Abs;
      dist= abs(fine-prima);
      if abs(prima-mezzo)>dist
           data{i}(bin).Abs= (fine+prima)/2;
end
end
}
```

Le figure 3.1 e 3.2 mostrano alcune funzioni di trasferimento della sfera prima e dopo l'operazione di correzione. Anche solo con questo metodo base si può notare come gli errori alle basse frequenze siano stati risolti. Restano però alcuni problemi alle alte frequenze.

3.3.2 Testa KEMAR

Le HRTFs della testa del manichino KEMAR sono molto più articolate e un metodo come quello usato nel punto precedente non produceva risultati accettabili. E' stato scelto quindi di usare un filtro a media mobile per effettuare lo smoothing della curva e correggere gli errori di simulazione

Le figure 3.3 e 3.4 mostrano le HRTFs della testa del manichino KEMAR prima e dopo il filtraggio.

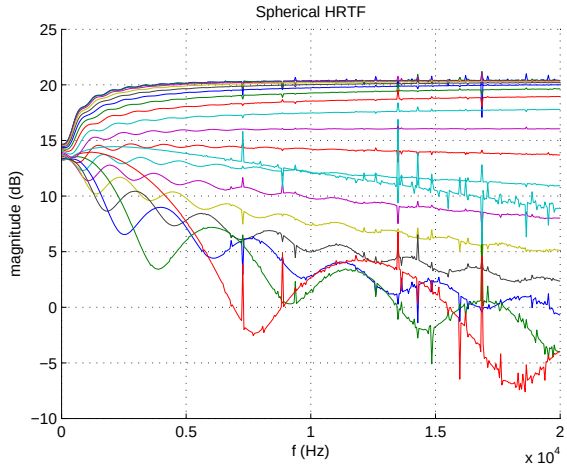


Figura 3.1: *HRTFs della sfera*

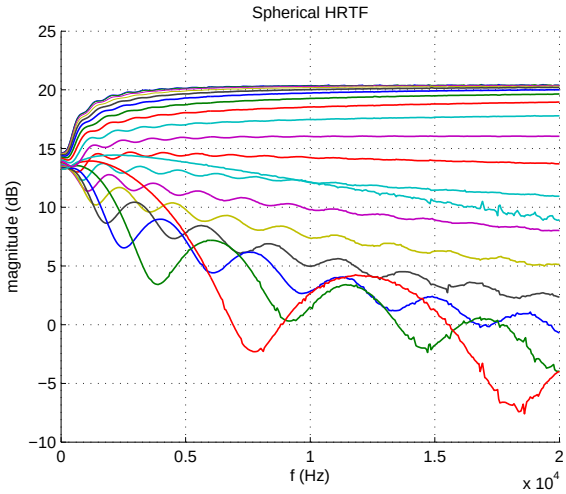


Figura 3.2: *HRTFs della sfera 'pulite'*

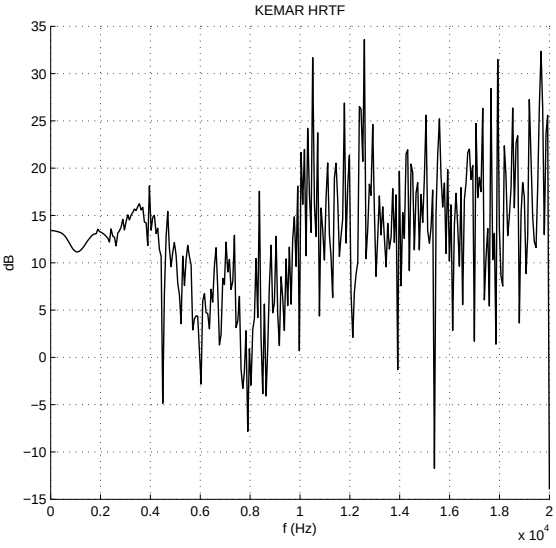


Figura 3.3: *HRTF della testa*

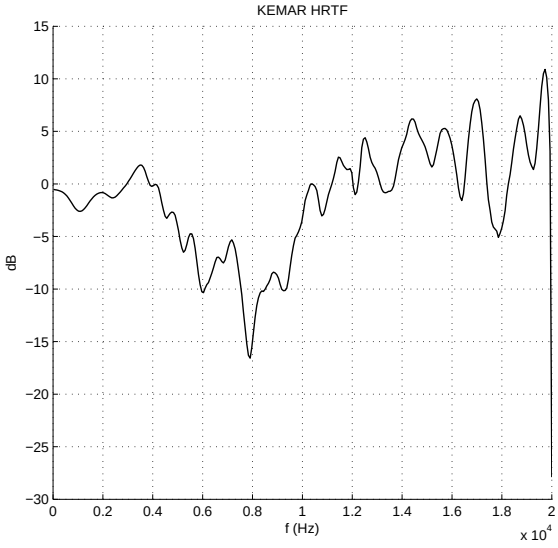


Figura 3.4: *HRTF della testa 'pulite'*

3.4 Altre modifiche dei dati

3.4.1 Normalizzazione

Una volta risolto il problema di Dirichlet si è dovuti procedere a sistemare ulteriormente i dati in modo da poterli confrontare con un database di HRTFs misurate. Le HRTFs dovrebbero a livello teorico partire dal valore 0 (dB) dato che la frequenza nulla non viene in alcun modo modificata. È stato quindi necessario traslare sull'asse Y i le funzioni in modo da ottenere $H(0) = 0$. Dato che per errori di simulazione le HRTFs non partivano tutte dallo stesso punto si è scelto di fare una media dei valori iniziali e di traslare tutte le curve in modo che la media venisse spostata sullo 0. L'operazione eseguita per effettuare la traslazione è stata diversa nei vari casi: nel database *PKU&IOA* i dati erano già forniti in forma logaritmica e si è quindi proceduto sottraendo la media dei valori iniziali. Nella simulazione con *AcouSTO*, invece, i dati non erano in forma logaritmica; si è quindi diviso tutti i valori per la media dei valori iniziali. L'uguaglianza dei due metodi è garantita dalla proprietà dei logaritmi

$$\log_a(x) - \log_a(y) = log_a\left(\frac{x}{y}\right)$$

3.4.2 Inversione e traslazione

Il database scelto per il confronto è quello di PKU&IOA. Questo database è uguale al nostro per il posizionamento e numero dei microfoni, entrambi hanno 14 differenti valori di elevazione e un numero di microfoni per ogni elevazione che varia tra 72 e 1 man mano che l'elevazione cresce. La tabella 3.1 mostra il numero di microfoni presenti per ogni elevazione e la differenza (in gradi) tra una posizione e l'altra.

Ci sono tuttavia due differenze che è stato necessario sistemare:

- I microfoni del database sono indicizzati in senso orario, mentre i nostri in senso antiorario;
- La posizione di azimuth 0 corrisponde nel database al naso e nella nostra simulazione, all'orecchio sinistro.

Dato che ogni microfono è salvato nel workspace come un puntatore in un cell array è stato necessario quindi invertire l'ordine dei dati all'interno dei blocchi di altezza (per passare da verso antiorario a verso orario) e poi traslare circolarmente questi stessi blocchi (per cambiare la posizione dello 0).

Il fatto che il numero di microfoni variasse senza una precisa relazione matematica in funzione dell'elevazione ha complicato lo svolgimento di queste operazioni in un'unico ciclo for. Alla fine si è scelto per comodità e chiarezza di scrivere singolarmente tutte le 14 istruzioni di inversione e shift.

La tabella 3.1 mostra il numero di microfoni presenti per ogni elevazione e la differenza (in gradi) tra una posizione e l'altra.

Elev (deg)	-40	-30	-20	-10	0	10	20	30	40	50	60	70	80	90
Step (deg)	5	5	5	5	5	5	5	5	5	5	10	15	30	360
Numero	72	72	72	72	72	72	72	72	72	72	36	24	12	1

Tabella 3.1: *Posizionamento dei microfoni*

3.5 Grafici 3d

La visualizzazione dei grafici posizione per posizione è molto precisa, ma dà un idea molto ristretta dei dati generali. È stato scelto perciò di scrivere due script che stampino dei grafici 3D delle HRTFs:

- *HRTFsPlotByElevation*: stampa i grafici in funzione di frequenza e elevazione, dato in ingresso un certo valore di azimut. Questo script considera i microfoni solo fino ad elevazione 50 gradi, dato che per elevazioni maggiori il numero di microfoni varia in funzione dell'elevazione.

- *HRTFsPlotByAzimuth*: stampa i grafici in funzione di frequenza e azimut, dato in ingresso un certo valore di elevazione

3.6 Confronto

La figura 3.5 mostra un grafico 3d delle HRTFs ad elevazione 0 gradi. Si può notare come ci sia una zona di alta pressione acustica attorno ai 90 gradi di azimut e una corrispettiva zona di bassa pressione acustica attorno ai 270 gradi. Ciò avviene perchè la sorgente è posta nell'orecchio destro; quando i microfoni considerati si avvicinano alla sorgente, chiaramente registrano un aumento del segnale; quando invece si trovano in prossimità dell'orecchio sinistro sono schermati dalla testa e registrano una diminuzione del segnale.

Le figure 3.6, 3.7, 3.8 permettono un rapido confronto tra le HRTFs simulate e le HRTFs del database. È necessario innanzitutto precisare che la simulazione è stata fatta con un dettaglio della mesh che non assicura l'affidabilità dei risultati oltre i 7000 Hz. Inoltre i dati del database, sono stati registrati con un manichino con testa e busto, mentre la nostra simulazione considera solamente la testa. Nonostante questo si può comunque fare un'analisi approssimata notando che i principali notch coincidono tra le due figure.

Le figure 3.9 e 3.10 mostrano dei grafici 3d delle HRTFs simulate e del database ad azimuth 0 gradi, con diversi valori di elevazione. Anche qui si può notare come, a meno di un fattore di normalizzazione, i due grafici coincidano fino ad una frequenza di circa 8000 Hz e si differenzino per frequenze superiori.

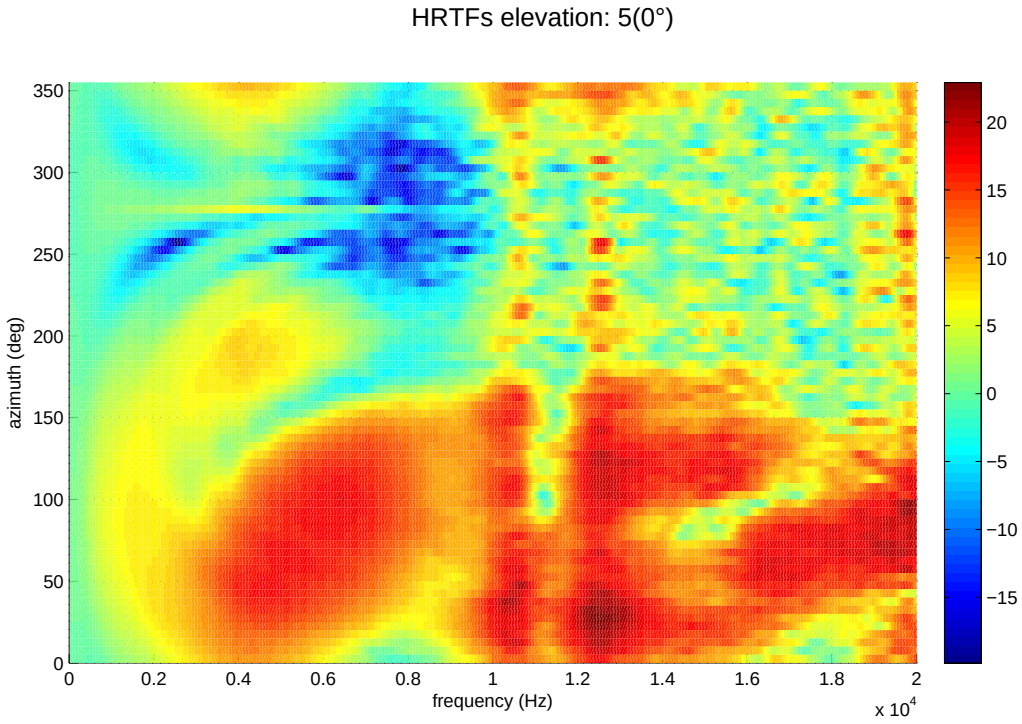
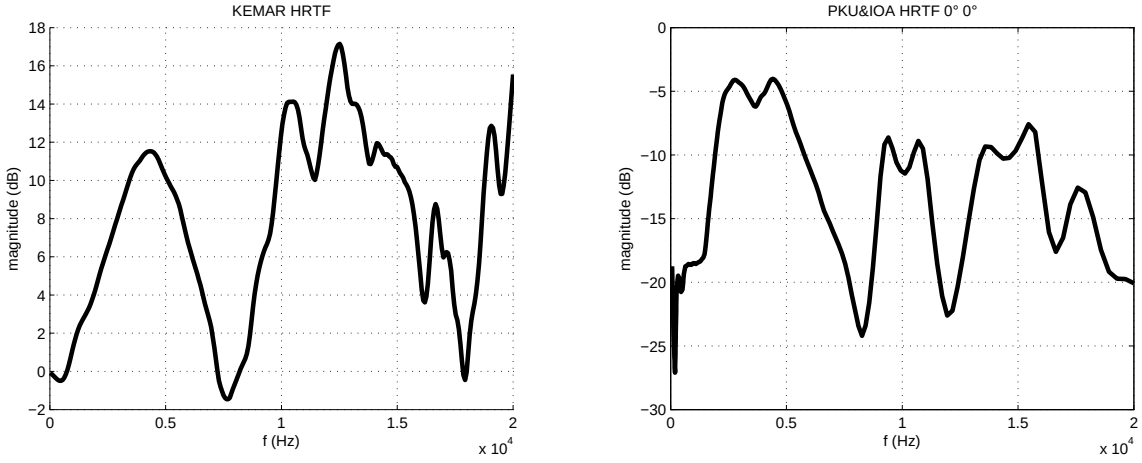
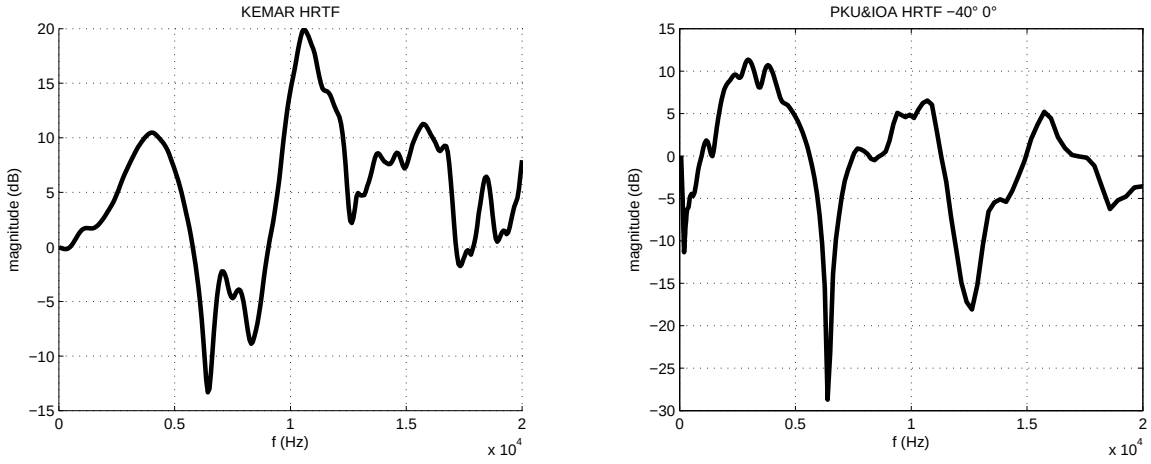


Figura 3.5: *3D-HRTFs*



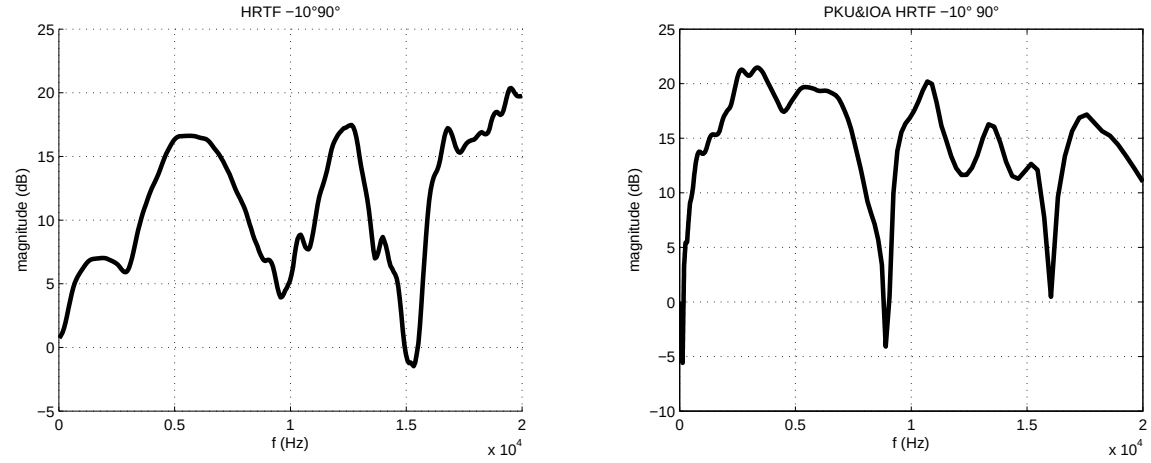
(a) HRTF simulata (b) HRTF PKU&IOA

Figura 3.6: *Elevazione 0, Azimut 0*



(a) HRTF simulata (b) HRTF PKU&IOA

Figura 3.7: *Elevazione -40, Azimut 0*



(a) HRTF simulata (b) HRTF PKU&IOA

Figura 3.8: *Elevazione -10, Azimut 90*

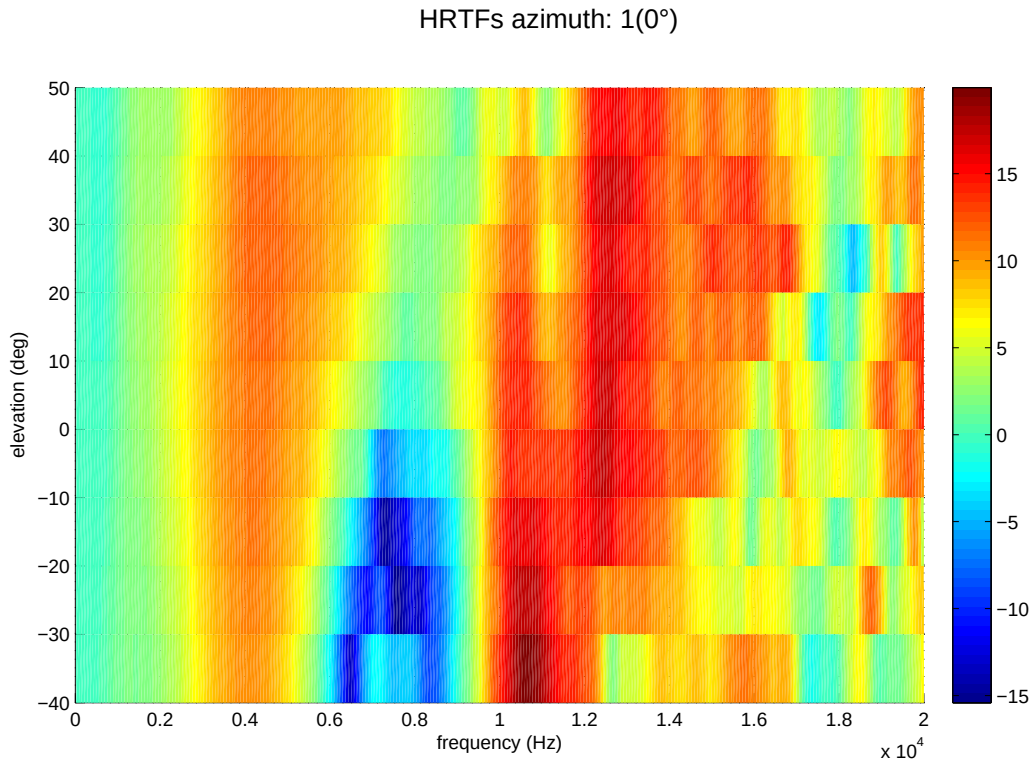


Figura 3.9: *3D-HRTF simulate Azimut 0*

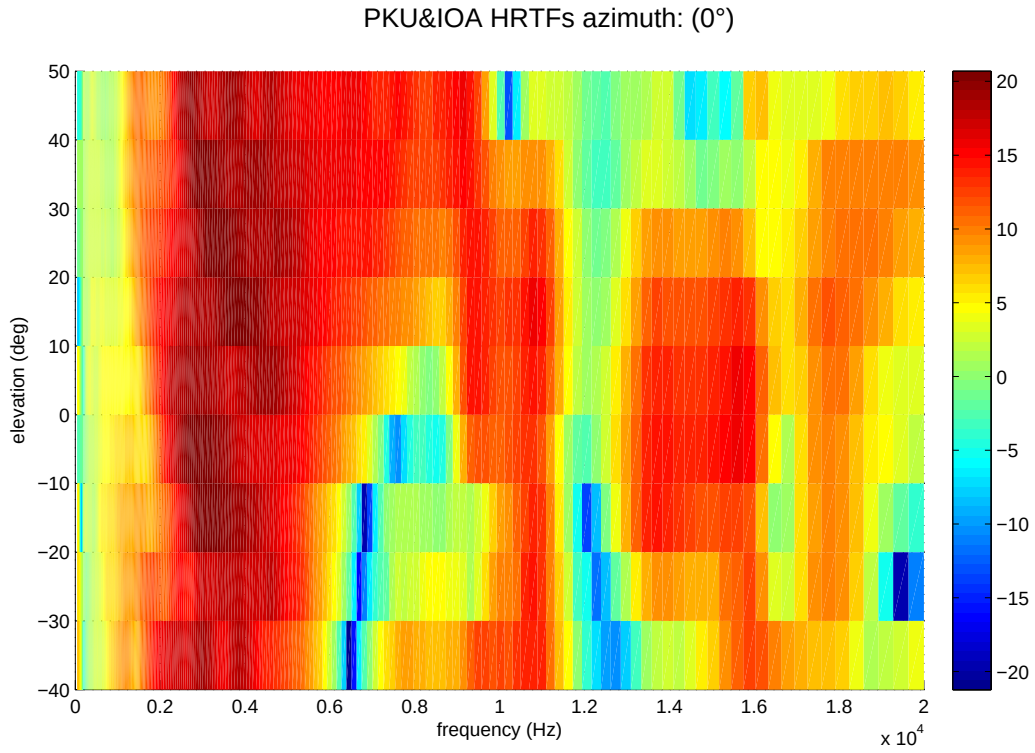


Figura 3.10: *3D-HRTFs PKU&IOA Azimut 0*

3.7 Conclusioni

Il notevole tempo richiesto per eseguire e portare a termine simulazioni non banali, come quella del manichino *KEMAR*, e i problemi di compatibilità con *Power7* hanno reso al momento impraticabile l'utilizzo di *AcouSTO* per ottenere risultati significativi su tutto l'intervallo di frequenze udibili da un essere umano all'interno del lavoro di tesi. Infatti il principale vincolo alla simulazione del manichino è stato il tempo impiegato per la soluzione di una singola frequenza. La memoria RAM utilizzata è solo una piccola parte di quella disponibile in quanto, aumentando la risoluzione della mesh per utilizzarla nella sua totalità, si sono ottenuti tempi di esecuzione dell'ordine dei mesi.

I risultati ottenuti non sono quindi fedeli alla letteratura, ma si possono definire incoraggianti nel continuare la ricerca nello sviluppo e nella sperimentazione con questo software.

3.8 Sviluppi Futuri

Tra i numerosi sviluppi futuri vengono riportati di seguito quelli di più immediata realizzazione:

- simulazione con una mesh più dettagliata;
- miglioramento dello script di regolarizzazione dei dati;
- confronto numerico tra i risultati della simulazione e la letteratura.

Un migliore sistema di regolarizzazione e un aumento del dettaglio della mesh permetterebbero infatti di diminuire le approssimazioni dovute agli errori di simulazione.

A questo punto sarebbe possibile confrontare i dati con la letteratura usando misure come la distorsione spettrale

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(20\log_{10}\frac{|H(f_i)|}{|\widetilde{H}(f_i)|}\right)^2} \qquad (dB) \tag{3.5}$$

per valutare formalmente il grado di precisione delle simulazioni.

Appendici

Appendice A

Codice

A.1 Caricamento dati

A.1.1 export_data

```
%
%  IN:
%    folder path contente tutti e SOLO i file da esportare a parita' di
%    distanza
%    nome del workspace dove salvare i dati caricati


function export_data(folder, ws_name)

% Cerca tutti i file con estensione out
dirName = folder;                           % folder path
files = dir( fullfile(dirName,'*.out') );   % list all *.xyz files
files = {files.name}';                      % file names

data = cell(numel(files),1);                %# store file contents
data_clean = cell(numel(files),1);
for i=1:numel(files)
    fname = fullfile(dirName,files{i});     %# full path to file
    data{i} = load_data(fname);             %# load file
end

% (salvataggio in file mat)
file_ws = ['./ws/' ws_name '.mat'];
%matfile(file_ws,'Writable',true, '-append');
save(file_ws);

end %end function
```

A.1.2 load_data

```
%
%  IN: file da caricare in una struct

function data = load_data(file)
    fid = fopen(file,'r');
    header = {'ifreq', 'omega', 'f', 'Re', 'Im', 'Abs'};

    %read first line (comment)
    %line = fgetl(fid);
    %read second line
    line = fgetl(fid);

    ID=1;
    while ischar(line)

       l_string = strread(line,'%s','delimiter',' ');
       l = str2double(l_string);

       for i=1:numel(l)
           data(ID).(header{i}) = l(i);
       end

       ID = ID+1;
       line = fgetl(fid);
    end

    fclose(fid);
end %end function
```

A.2 Modifica dati

A.2.1 clean_data

```
%
%  IN: file contenente il workspace con i dati
%  OUT: un file workspace_clean con i dati modificati.
%
%   In particolare lo script:
%   -Normalizza i valori portando le H(0)=0 (db)
%   -Toglie alcuni valori dovuti ad errori di simulazione
%   -filtra i dati  con un filtro a media mobile
%   -riordina i dati in modo da farli coincidere con l'ordine deciso da Qu
%   nel suo paper


function clean_data(ws_name)

% caricamento data da workspace
```

```
file = ['./ws/' ws_name];
load(file);

%%Normalizza le trasformate sull'asse y in modo che i valori delle
%trasformate a 0 Hz siano circa uguali a 0 db
%
%NOTA: Siccome i valori iniziali non sono tutti uguali, fa una media e
%divide tutti i valori per la media

somma=0;
for i=1:numel(data)  %somma
    somma=somma+data{i}(1).Abs;
end
media= somma/numel(data); %media
for i=1:numel(data)  %abbassa tutti i dati
    for bin = 1:numel(data{i})
        data{i}(bin).Abs=data{i}(bin).Abs /media;
    end
end

%%Corregge gli errori di simulazione trovando i punti molto "distanti" e
%%sostituendoli con la media dei due ai lati

for i=1:numel(data)
    for bin = 2:(numel(data{i})-1)
        prima= data{i}(bin-1).Abs;
        mezzo= data{i}(bin).Abs;
        fine= data{i}(bin+1).Abs;
        dist= abs(fine-prima);
        if abs(prima-mezzo)>dist     %controlla se il valore  paragonabile
            data{i}(bin).Abs= (fine+prima)/2;
        end

    end
end

%%Filtra i dati con un filtro a media mobile
vect1= []; %vettori di appoggio
vect2= [];
b=ones(1,5)/5;
for i=1:numel(data)  %per ogni posizione microfonica
    for bin = 1:numel(data{i})   %trasferisce nel vettore di appoggio
        vect1(bin) =data{i}(bin).Abs;
    end
    vect2 = abs(filtfilt(b,1,vect1))+eps; %filtra

    for bin = 1:numel(data{i})  %ritrasferisce in data
        data{i}(bin).Abs =vect2(bin);
    end
end
```

```
%%Riordina i dati in modo da farli coincidere con quelli di Q
%In particolare:
%-inverte i dati (devo passare da senso antiorario a orario)
%-li trasla (devono partire dal naso al posto che dall'orecchio sx)

data(2:1:72)=data(72:-1:2);        %inverte i dati
data(74:1:144)=data(144:-1:74);
data(146:1:216)=data(216:-1:146);
data(218:1:288)=data(288:-1:218);
data(290:1:360)=data(360:-1:290);
data(362:1:432)=data(432:-1:362);
data(434:1:504)=data(504:-1:434);
data(506:1:576)=data(576:-1:506);
data(578:1:648)=data(648:-1:578);
data(650:1:720)=data(720:-1:650);
data(722:1:756)=data(756:-1:722);
data(758:1:780)=data(780:-1:758);
data(782:1:792)=data(792:-1:782);
%Il microfono 793  l'unico e non deve essere messo a posto

data(1:1:72)=circshift(data(1:1:72),-18);  %trasla i dati
data(73:1:144)=circshift(data(73:1:144),-18);
data(145:1:216)=circshift(data(145:1:216),-18);
data(217:1:288)=circshift(data(217:1:288),-18);
data(289:1:360)=circshift(data(289:1:360),-18);
data(361:1:432)=circshift(data(361:1:432),-18);
data(433:1:504)=circshift(data(433:1:504),-18);
data(505:1:576)=circshift(data(505:1:576),-18);
data(577:1:648)=circshift(data(577:1:648),-18);
data(649:1:720)=circshift(data(649:1:720),-18);
data(721:1:756)=circshift(data(721:1:756),-9);
data(757:1:780)=circshift(data(757:1:780),-6);
data(781:1:792)=circshift(data(781:1:792),-3);
%Il microfono 793  l'unico e non deve essere messo a posto

% (aggiornamento file .mat in forma della struttura data_clean seguendo
% lo stesso formato data
file_ws = ['./ws/' ws_name '_clean.mat'];
%matfile(file_ws,'Writable',true, '-append');
save(file_ws);

end %end function
```

A.3 Plot 2D

A.3.1 display_data

```
%
%  input: ws_name: file contente il workspace da visualizzare
%         plot_name: nome del grafico
```

```
%         index: indice del microfono da visualizzare. index = -1 se voglio visualizzarl
%  tutti
%
% output: plot dei dati
%
% @author        Francesco Foscarin
% @email        foscarin.francesco@gmail.com


function display_data(ws_name, plot_name, index)

% caricamento data da workspace
file = ['./ws/' ws_name];
load(file);
p = figure();
colori = lines(numel(data)); %assegnazione colori dei diversi grafici
if index == -1
    inizio=1;
    fine=numel(data);
elseif index>=0 && index<=793
    inizio=index;
    fine=index;
else disp('Wrong index');
    return;
end

for i=index:fine
    %Data columns:
    % header = {'ifreq', 'omega', 'f', 'Re', 'Im', 'Abs'};
    freq_resp = data{i};             % load freq resp data

    % plot
    Fr = []; %frequency
    A_HRTF = []; %amplitude
    for bin = 1:(numel(freq_resp))
        Fr = [Fr freq_resp(bin).f];
        A_HRTF = [A_HRTF freq_resp(bin).Abs];
    end

    hold on;
    semilogx(Fr, 20*log10(A_HRTF),'Color', colori(i,:));
    grid on; zoom on;
    %axis([Fr(1) Fr(end)]);
    title(['KEMAR HRTF']);
    xlabel('f (Hz)');
    ylabel('dB');
    hold off;

    % salvataggio plot
end
```

```
saveas(p,['plot\' plot_name '.fig']);

end %end function
```

A.3.2 readhrir

```
function hrir = readhrir(filepath, dist, elev, azi, lr, plot_name)
% writen by Zhen Xiao, edited by Francesco Foscarin
% read Qu-HRTF database
% filepath is the path of the database
% dist is the distanceincluding 20 30 40 50 75 100 130 160
% elev is the elevation, value from -40 to 90 in step of 10
% azi is the azimulth, value form 0 to 355 in step of 5 (elev <= 50), from 0 to 350 in s
% lr represent the data form right ear or left ear. lr == l means left, and lr == r mean
% the length of each HRIR is 1024 points.
%
%output: data plotted

if nargin ~= 6
    error('Wrong number of input arguments')
end
filename = [filepath, 'dist', int2str(dist), '\elev', int2str(elev), '\azi', int2str(azi
p = fopen(filename,'r');
if lr == 'l'
    hrir = fread(p, 1024, 'double');
else if lr == 'r'
        fseek(p,1024*8,'bof');
        hrir = fread(p, 1024, 'double');
    else
        hrir = fread(p, 'double');
        hrir = reshape(hrir, 1024, 2);
    end
end

fclose(p);

%Trasforma la HRIR->HRTF

[DFT_r dB_r phi_r sf_r]= freq_resp_tot(hrir,0,20000,Inf,1,1024,65536);
%Normalizza i risultati
Val=dB_r(1);
for i=1:numel(dB_r)
    dB_r(i)=dB_r(i)-Val;
end
%Plot
p=figure();
plot(sf_r, dB_r);
grid on; zoom on;
title(['Spherical HRTF - Qu' ]);
xlabel('f (Hz)');
ylabel('dB');
```

```
xlim([0 20000]);

saveas(p,['plot\' plot_name '_Qu.fig']);
end
```

A.4 Plot 3D

A.4.1 HRTFsplotByAzimuth

```
function HRTFsplotByAzimuth( ws_name, azimuth_value )
    %% HRTFsplotByAzimuth( ws_name, azimuth_value )
    % This function receive the value of azimuth and plot the HRTFs at
    % different elevation
    % This script assume that the number of the sampled frequences is 300
    % This script plot mics only
    %
    % Input:
    %       ws_name: name of workspace
    %       azimuth value: value of azimuth (from 1 to 72)1-> 0, 2-> 5, ..., 72->355
    % Output:
    %       plotted data
    %
    % @author        Francesco Foscarin
    % @email        foscarin.francesco@gmail.com

    %% Initialization
    file = ['./ws/' ws_name];
    load(file);

    %% Get data to plot

    %calcola qual  l'angolo
    real_azimuth=(azimuth_value-1)*5;

    %trova la matrice con le ampiezze e il vettore con gli angoli
    A_HRTF = zeros(300,10); %amplitude
    Elevation_values = [ -40 -30 -20 -10 0 10 20 30 40 50]; %angle
    bin=azimuth_value;
    i=1;
    while bin<720 %fa il ciclo solo sui microfoni che interessano
        freq_resp = data{bin};
        for k=1:300 %ciclo su tutte le frequenze
            A_HRTF(k,i) = freq_resp(k).Abs;
        end
        i=i+1;
        bin=bin+72;
    end
    Val_min=min(min(A_HRTF)); %Min di tutte le ampiezze
    Val_max=max(max(A_HRTF)); %Max di tutte le ampiezze
```

```
    %Trova il vettore delle frequenze (dato che  uguale per tutte le
    %posizioni)
    Fr = []; %frequency
    for ind=1:300
        Fr = [Fr data{1}(ind).f];
    end

    % plot
    xwidth = 600;
    ywidth = 700;
    figure(1);
    hFig = figure(1);
    set(gcf,'PaperPositionMode','auto');
    set(hFig, 'Position', [100 100 xwidth ywidth]);
    suptitle({sprintf('HRTFs azimuth: %g(%g) \n', azimuth_value, real_azimuth)});
    axis tight;
    title('Left channel','FontSize', 11, 'FontWeight','bold');
    logHRTF=20*log10(A_HRTF);
    surf(Elevation_values,Fr,logHRTF);

    %set(gca, 'YScale', 'log');
    %caxis([-25 30]);
    shading flat;

    % set color
    colormap('jet');
    colorbar;

    % set view
    view(2);%view([0 90]);

    % set axis
    xlabel('elevation (deg)');
    ylabel('frequency (Hz)');
    zlabel('magnitude (dB)');
    xlim([-40 50]);
    ylim([0 20000]);
    %zlim([-25 30]);

end
```

A.4.2 HRTFsplotByElevation

```
function HRTFsplotByElevation( ws_name, elevation_value )
    %% HRTFsplotByElevation( ws_name, elevation_value )
    % This function receive the value of elevation and plot the HRTFs at
    % different azimuth
    % This script assume that the number of the sampled frequences is 300
    %
    % Input:
    %       ws_name: name of workspace
```

```
%       elevation_value: value of elevation (from 1 to 14)
% Output:
%       data_l/r: plotted data
%
% @author        Francesco Foscarin
% @email        foscarin.francesco@gmail.com

%% Initialization
file = ['./ws/' ws_name];
load(file);
inizio=0;
fine=0;

%% Get data to plot
%calcola qual  l'indice di partenza dei microfoni da plottare
%1-> -40, 2-> -30, ..., 14->90

switch elevation_value
    case {1,2,3,4,5,6,7,8,9,10}
        inizio=((elevation_value-1)*72)+1;
        fine=(inizio-1)+72;
    case 11
        inizio=721;
        fine=756;
    case 12
        inizio=757;
        fine=780;
    case 13
        inizio=781;
        fine=792;
    case 14
        inizio=793;
        fine=793;

otherwise
    warning('Unexpected elevation_value. No plot created.');
end

%calcola qual  l'angolo
real_elevation=(elevation_value*10)-50;

%trova la matrice con le ampiezze e il vettore con gli angoli
A_HRTF = zeros(300,fine+1-inizio); %amplitude
Azimuth_values = []; %angle
i=0;
incr_azimuth=360/(fine+1-inizio); %calcola ogni quanti gradi si trova un mic

for bin=inizio:fine %fa il ciclo solo sui microfoni che interessano
    freq_resp = data{bin};
    for k=1:300 %ciclo su tutte le frequenze
        A_HRTF(k,bin+1-inizio) = freq_resp(k).Abs;
```

```
        end
        Azimuth_values= [Azimuth_values i] ;
        i=i+incr_azimuth;
    end
    Val_min=min(min(A_HRTF)); %Min di tutte le ampiezze
    Val_max=max(max(A_HRTF)); %Max di tutte le ampiezze

    %Trova il vettore delle frequenze (dato che  uguale per tutte le
    %posizioni)
    Fr = []; %frequency
    for ind=1:300
        Fr = [Fr data{1}(ind).f];
    end

    % plot
    xwidth = 600;
    ywidth = 700;
    figure(1);
    hFig = figure(1);
    set(gcf,'PaperPositionMode','auto');
    set(hFig, 'Position', [100 100 xwidth ywidth]);
    suptitle({sprintf('HRTFs elevation: %g(%g) \n', elevation_value, real_elevation)});
    axis tight;
    title('Left channel','FontSize', 11, 'FontWeight','bold');
    logHRTF=20*log10(A_HRTF);
    surf(Azimuth_values,Fr,logHRTF);

    %set(gca, 'YScale', 'log');
    %caxis([Val_min Val_max]);
    shading flat;

    % set color
    colormap('jet');
    colorbar;

    % set view
    view(2);%view([0 90]);

    % set axis
    xlabel('azimuth (deg)');
    ylabel('frequency (Hz)');
    zlabel('magnitude (dB)');
    xlim([0 360]);
    ylim([0 20000]);
    %zlim([Val_min Val_max]);

end
```

A.4.3 QuHRTFsplotByAzimuth

```
function QuHRTFsplotByAzimuth( azimuth_value )
    %% QuHRTFsplotByAzimuth( azimuth_value )
    % This function receive the value of azimuth and plot the HRTFs at
    % different azimuth
    % This script assume that the number of the sampled frequences is 300
    %
    %
    % Input:
    %       azimuth value: value of azimuth (from 1 to 72)1-> 0, 2-> 5, ..., 72->355
    % Output:
    %       plotted data
    %
    % @author        Francesco Foscarin
    % @email        foscarin.francesco@gmail.com

    %% Initialization

    filepath='Hrir\';
    dist=160;
    real_azimuth=azimuth_value;
    lr='r';
    sf_r=[];

    %% Get data to plot

    %trova la matrice con le ampiezze e il vettore con gli angoli
    A_HRTF = zeros(313,10); %amplitude
    Elevation_values = [ -40 -30 -20 -10 0 10 20 30 40 50]; %angle
    bin=-40;
    i=1;
    while bin<=50 %fa il ciclo solo sui microfoni che interessano
        %carica i dati dalla cartella
        filename = [filepath, 'dist', int2str(dist), '\elev', int2str(bin), '\azi', int2
        p = fopen(filename,'r');
        if lr == 'l'
            hrir = fread(p, 1024, 'double');
        else if lr == 'r'
                fseek(p,1024*8,'bof');
                hrir = fread(p, 1024, 'double');
            else
                hrir = fread(p, 'double');
                hrir = reshape(hrir, 1024, 2);
            end
        end

        fclose(p);

        %Trasforma la HRIR->HRTF

        [DFT_r dB_r phi_r sf_r]= freq_resp_tot(hrir,0,20000,Inf,1,1024,65536);
```

```
    %Normalizza i risultati
    Val=dB_r(1);
    for x=1:numel(dB_r)
        dB_r(x)=dB_r(x)-Val;
    end
    %Memorizza i dati nella matrice
    for k=1:313 %ciclo su tutte le frequenze
        A_HRTF(k,i) = dB_r(k);
    end
    i=i+1;
    bin=bin+10;
end
Val_min=min(min(A_HRTF)); %Min di tutte le ampiezze
Val_max=max(max(A_HRTF)); %Max di tutte le ampiezze

%Trova il vettore delle frequenze (dato che  uguale per tutte le
%posizioni)
Fr = []; %frequency
Fr=sf_r;

% plot
xwidth = 600;
ywidth = 700;
figure(1);
hFig = figure(1);
set(gcf,'PaperPositionMode','auto');
set(hFig, 'Position', [100 100 xwidth ywidth]);

suptitle({sprintf('Qu HRTFs azimuth: (%g) \n', real_azimuth)});
axis tight;
title('Right channel','FontSize', 11, 'FontWeight','bold');
surf(Elevation_values,Fr,A_HRTF);

%set(gca, 'YScale', 'log');

%caxis([-25 30]);
shading flat;

% set color
colormap('jet');
colorbar;

% set view
view(2);%view([0 90]);

% set axis
xlabel('elevation (deg)');
ylabel('frequency (Hz)');
zlabel('magnitude (dB)');
xlim([-40 50]);
ylim([0 20000]);
```

```
    %zlim([-25 30]);
end
```

Bibliografia

[1] *NextEngine 3D Scanner HD TechSpecs*. `http://www.nextengine.com/assets/pdf/scanner-techspecs.pdf`.

[2] G. De Poli F. Avanzini. *Algorithms for Sound and Music Computing*. 2009.

[3] Brian F. G. Katz. Boundary element method calculation of individual head-related transfer function. rigid model calculation. *Acoustical Society of America*, 2001.

[4] Duda R. O. & Martens W. L. Range dependence of the response of a spherical head model. pages 3048–3058, 1998.

[5] Ramani Duraiswami Nail A. Gumerov, Adam E. ODonovan and Dmitry N. Zotkin. Computation of the head-related transfer function via the fast multipole accelerated boundary element method and its spherical harmonic representation. *Acoustical Society of America*, 2009.

[6] Tianshu Qu. Distance-dependent head-related transfer functions measured with high spatial resolution using a spark gap. 2009.

[7] Geronazzo M. e Avanzini F. Spagnol S. On the relation between pinna reflection patterns and head-related transfer function features. pages 508–519, 2013.

[8] Vincenzo Marchese Umberto Iemma. Acousto, acoustic simulation tool, February 2013.