DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Progettazione ed Implementazione di un Plugin VST per la Simulazione di Differenti Profili Uditivi di Cuffie "

Relatore: Prof. / Dott Sergio Canazza

Laureando/a: Risi Emilio

Correlatore: Prof,/Dott Fiordelmondo Alessandro

ANNO ACCADEMICO 2022 – 2023

Data di laurea 18/07/2023

Abstract

Questa tesi presenta il design e l'implementazione di un plugin VST per la simulazione di diversi profili uditivi di cuffie. Il plugin permette agli utenti di valutare le caratteristiche uditive di diverse cuffie ad alta fedeltà, consentendo loro di prendere decisioni informate nell'acquisto di nuovi modelli. L'implementazione si basa su un modello matematico che considera le cuffie come sistemi lineari e invarianti nel tempo. La simulazione viene ottenuta attraverso operazioni di convoluzione e deconvoluzione delle risposte impulsive delle cuffie. L'implementazione del plugin viene realizzata utilizzando la libreria JUCE, un framework ampiamente utilizzato per lo sviluppo di plugin audio. L'architettura del plugin è progettata per includere un'interfaccia utente intuitiva per il controllo dei parametri.

Indice dei contenuti

Contents

1 Introduzione **2**

2 Fondamenti Teorici **3**
- 2.1 Profili Uditivi di Cuffie . . . 3
- 2.2 Operazioni di Convoluzione . . . 4
- 2.3 Sistemi Lineari e Invarianti nel Tempo . . . 4
- 2.4 Algoritmi di Convoluzione . . . 5
  - 2.4.1 Convoluzione tramite FFT . . . 5
  - 2.4.2 Metodologia di Convoluzione tramite Overlap e Add . . . 5

3 Modello adottato per la simulazione **6**
- 3.1 Ottenimento delle misurazioni di Risposta Impulsiva utilizzate nel progetto . . . 8

4 VST, DAWS e JUCE **10**
- 4.1 Formato di un Plugin Audio: VST (Virtual Studio Technology) . 10
- 4.2 La framework JUCE . . . 11

5 HP Simulator **13**
- 5.1 Interfaccia utente . . . 13
- 5.2 Generazione dei filtri $h_s[N]$ . . . 14
- 5.3 Codice Del plugin . . . 15
  - 5.3.1 Caricamento della risposta impulsiva . . . 16
  - 5.3.2 Processazione audio in tempo reale del Plugin . . . 19

6 Conclusioni **21**

7 Bibliografia **22**

1 Introduzione

La continua evoluzione della tecnologia audio e la crescente domanda di esperienze sonore di alta qualità hanno reso le cuffie un dispositivo sempre più diffuso, presentando anche differenze significative nelle loro caratteristiche uditive.

Se Infatti una cuffia che enfatizza le frequenze basse, fornendo un suono potente e avvolgente, e un'altra cuffia potrebbe offrire una risposta in frequenza bilanciata, riproducendo fedelmente tutti gli spettri sonori. Queste differenze nella risposta in frequenza possono influenzare notevolmente l'esperienza uditiva dell'utente, portando a preferenze personali diverse.

La capacità di valutare in modo accurato e oggettivo queste differenze è fondamentale per gli utenti che desiderano scegliere le cuffie che meglio si adattano alle loro preferenze personali e alle esigenze specifiche. Esigenze derivate da una vasta gamma di applicazioni professionali, come la produzione musicale e l'audio mastering, dove una riproduzione fedele e accurata del suono è essenziale per una corretta riproduzione sui dispositivi dell'utente finale.

In questo contesto, la presente ricerca propone il design e l'implementazione di un plugin VST (Virtual Studio Technology) per la simulazione del profilo uditivo di diverse cuffie.

La ricerca si basa su un modello matematico sviluppato dal Centro di Sonologia Computazionale (CSC) in collaborazione con Audio Innova. Il CSC, appartenente al Dipartimento di Ingegneria dell'Informazione (DEI) dell'Università di Padova, è riconosciuto a livello internazionale come un centro di ricerca e sperimentazione di rilevanza nel campo della musica computerizzata.

Alcune possibili applicazioni di un simile progetto sono:

- Fornire agli utenti uno strumento pratico e affidabile per valutare e confrontare le caratteristiche uditive di diverse cuffie.
- Consentire agli utenti di prendere decisioni informate nell'acquisto di cuffie, migliorando l'esperienza di ascolto e soddisfacendo le loro preferenze personali.
- Offrire una soluzione efficace per professionisti del settore audio, come produttori musicali e ingegneri del suono, che richiedono una riproduzione accurata del suono attraverso le cuffie.

2 Fondamenti Teorici

Questa sezione presenta i fondamenti teorici necessari per comprendere il funzionamento del plugin.

2.1 Profili Uditivi di Cuffie

Quando si parla di profilo uditivo di una cuffia ci si riferisce a alle proprietà e caratteristiche che influenzano l'ascolto e la percezione del suono riprodotto da questi strumenti Alcuni di questi sono :

- La Risposta in frequenza $H(f)$: La risposta in frequenza indica come una cuffia riproduce i diversi suoni nelle diverse frequenze. La risposta in frequenza ideale è piatta, il che significa che la cuffia riprodurrà in modo accurato tutte le frequenze nell'intero spettro udibile. Tuttavia, molte cuffie possono avere una risposta in frequenza che presenta lievi variazioni, come un picco o un calo in determinate frequenze, influenzando la resa del suono.

- Risposta agli impulsi $h(t)$: La risposta agli impulsi si riferisce alla capacità di una cuffia di rispondere rapidamente ai segnali di breve durata, come i transienti, comuni nelle registrazioni musicali. Una buona risposta agli impulsi garantisce che i suoni transitori siano riprodotti in modo preciso e dettagliato, consentendo di percepire meglio i piccoli dettagli musicali e le sfumature nell'audio. Questo dato sarà fondametale per lo sviluppo di questa tesi dato che ,tramite la convoluzione, permette di simulare le caratteristiche uditive di un suono riprodotto attraverso un dispositivo di riproduzione , solamente possedendo la risposta impulsiva di quest'ultimo (se approssimabile ad un sistema lineare tempo invariante)

- Larghezza di banda B : La larghezza di banda si riferisce all'intervallo di frequenze che una cuffia può riprodurre in modo accurato. Una cuffia con una larghezza di banda ampia è in grado di riprodurre sia le frequenze più basse che quelle più alte con precisione. Una larghezza di banda limitata può portare a una mancanza di dettagli nelle alte o basse frequenze, riducendo l'esperienza di ascolto complessiva.

- Distorsione armonica THD : La distorsione armonica è una misura della fedeltà con cui una cuffia riproduce un segnale audio senza introdurre distorsioni indesiderate. Una distorsione armonica bassa è preferibile poiché indica che la cuffia riproduce il suono in modo fedele all'originale. Al contrario, una distorsione armonica elevata può alterare la qualità del suono, aggiungendo rumore o alterando la chiarezza e la purezza delle note.

2.2 Operazioni di Convoluzione

La convoluzione tra due segnali $f(t)$ e $g(t)$ è definita come:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) \, d\tau$$

L'operazione di convoluzione combina due segnali per produrre un terzo segnale che rappresenta la loro interazione. Nel contesto della simulazione dei profili uditivi di cuffie, l'operazione di convoluzione viene utilizzata per combinare il segnale audio di ingresso con la risposta impulsiva del profilo uditivo della cuffia. Questo permette di replicare in modo realistico il modo in cui la cuffia modifica il segnale audio in termini di risposta in frequenza, larghezza di banda, risposta agli impulsi e altre proprietà uditive.

2.3 Sistemi Lineari e Invarianti nel Tempo

Le cuffie possono essere modellate come sistemi lineari e invarianti nel tempo (LTI), semplificando così l'analisi delle loro caratteristiche uditive. Questa modellazione matematica consente di utilizzare strumenti come la convoluzione, per comprendere e valutare la resa sonora delle cuffie. Un sistema LTI deve possedere due caratteristiche principali:

- **Proprietà di linearità:** La linearità delle cuffie significa che la loro risposta a una combinazione lineare di segnali, come una somma ponderata di varie frequenze o intensità, può essere calcolata come la somma ponderata delle risposte a ciascun segnale preso singolarmente. Questo permette di analizzare le caratteristiche uditive delle cuffie in modo coerente, isolando gli effetti specifici di ciascun componente del segnale audio.
- **Proprietà di invarianza nel tempo:** L'invarianza nel tempo delle cuffie implica che la loro risposta alle variazioni del segnale audio non cambia nel tempo. Ciò consente di applicare le stesse analisi e modelli matematici a diversi istanti nel tempo, rendendo possibile una valutazione costante e affidabile delle prestazioni uditive delle cuffie.

E' possibile rappresentare un sistema LTI utilizzando la convoluzione. Data una risposta all'impulso del sistema $h(t)$ e un segnale di ingresso $x(t)$, l'uscita $y(t)$ del sistema può essere calcolata come:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t - \tau) \, d\tau$$

dove $x(\tau)$ rappresenta il segnale di ingresso ritardato di τ.

2.4 Algoritmi di Convoluzione

Essendo una operazione della quale è necessario avere una buona efficenza sua dal punto del vista di precisione che dal punto di vista di latenza del risultato, nel tempo sono nati piu algoritmi di Convoluzione.

2.4.1 Convoluzione tramite FFT

La convoluzione tramite FFT (Fast Fourier Transform) è una tecnica che sfrutta le proprietà delle trasformate di Fourier per effettuare la convoluzione in modo efficiente. Questo approccio è particolarmente utile quando si desidera convolvere segnali di lunghezza significativa.

1. Esegue la trasformata di Fourier del segnale di ingresso $x(t)$ e della risposta impulsiva del sistema $h(t)$ utilizzando la FFT. Otteniamo le rispettive trasformate di Fourier $X(f)$ e $H(f)$.

2. Moltiplica le due trasformate di Fourier: $Y(f) = X(f) \cdot H(f)$. Questa operazione corrisponde alla convoluzione nel dominio delle frequenze.

3. Applica la trasformata di Fourier inversa (IFFT) a $Y(f)$ per ottenere il segnale convoluto $y(t)$ nel dominio del tempo.

La convoluzione tramite FFT è vantaggiosa quando la lunghezza del segnale di ingresso e della risposta impulsiva è grande, in quanto sfrutta l'efficienza computazionale della FFT per ridurre i tempi di calcolo.

2.4.2 Metodologia di Convoluzione tramite Overlap e Add

Un'altra metodologia di convoluzione tramite la tecnica dell'"overlap e add" è un approccio utilizzato quando si desidera convolvere segnali di lunghezza superiore alla capacità di elaborazione disponibile.

1. Divide il segnale di ingresso $x(t)$ e la risposta impulsiva del sistema $h(t)$ in segmenti più piccoli (finestre). I segmenti solitamente si sovrappongono tra loro.

2. Esegue la convoluzione tra ciascun segmento del segnale di ingresso e la risposta impulsiva utilizzando la moltiplicazione nel dominio del tempo.

3. Somma i segmenti convoluti sovrapposti, in genere tramite l'operazione di "addizione" o "overlap e add". Questa operazione combina i contributi dei segmenti sovrapposti per ottenere il segnale convoluto finale.

La metodologia di convoluzione tramite "overlap e add" è utile quando la capacità di elaborazione è limitata o quando si desidera ottenere risultati in tempo reale, sacrificando la precisione in alcuni punti.

In questa tesi si è preferito utilizzare la metodologia FFT principalmente per il motivo che "overlap and add" non sarebbe dovuta essere implementata

da zero non essendo supportata dal JUCE framework e avrebbe causato un aumento eccessivo della complessità del codice, sebbene il suo utilizzo avrebbe garantito un minore carico di lavoro e una migliore efficenza del plugin.

3 Modello adottato per la simulazione

L'assunzione fondamentale alla base del funzionamento del simulatore di cuffie richiede di ipotizzare che le cuffie Hi-Fi possano essere considerate sistemi lineari, invarianti nel tempo (LTI) rispetto al segnale audio fornito in ingresso. In particolare, in questa ipotesi, sia le cuffie monitor(indossate) che le cuffie target(simulate) devono soddisfare il requisito. Se assumiamo che le cuffie siano sistemi LTI, teoricamente è possibile simulare le caratteristiche uditive di qualsiasi cuffia utilizzando solo la sua funzione di trasferimento.

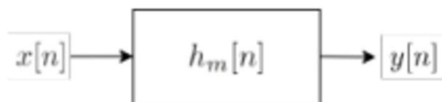

Figure 1: Rappresentazione generica di un sistema LTI

In generale, come mostrato nella Figura 1, le cuffie monitor con risposta impulsiva $h_m[n]$ modificheranno il contenuto spettrale del segnale di ingresso $x[n]$, generando il segnale audio in uscita $y[n]$ con uno spettro nuovo:

$$Y(f) = H_m(f) \cdot X(f)$$

Si noti che le funzioni indicate tra parentesi quadre rappresentano segnali discreti nel tempo, mentre le quantità tra parentesi tonde sono continue. Ad esempio, $H_m(f)$ rappresenta la trasformata di Fourier di $h_m(t)$, la controparte continua di $h_m[n]$. La simulazione delle cuffie di destinazione può essere ottenuta inserendo un filtro di simulazione specifico prima dell'azione delle cuffie da monitoraggio sul segnale di ingresso.

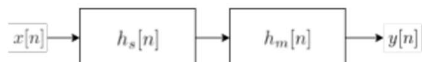

Figure 2: Posizionamento del filtro nel sistema

Questo filtro $h_s[n]$ dovrebbe rimuovere l'effetto della convoluzione applicata dalle cuffie da monitoraggio al segnale di ingresso, ossia la colorazione spettrale dell'audio originale, e allo stesso tempo replicare il risultato che si otterrebbe utilizzando le cuffie di destinazione desiderate, ottenendo una risposta in frequenza finale ovvero:

$$Y(f) = H_t(f) \cdot X(f)$$

dove $H_t(f)$ rappresenta la trasformata di Fourier della risposta impulsiva delle cuffie di target $h_t(t)$. Di conseguenza, la risposta in frequenza di questo filtro di simulazione sarà il rapporto tra la funzione di trasferimento delle cuffie da simulare e la funzione di trasferimento delle cuffie da monitoraggio:

$$H_s(f) = \frac{H_t(f)}{H_m(f)}$$

In questo modo, quando si utilizzano le cuffie da monitoraggio per ascoltare l'audio filtrato $y[n]$, il segnale di uscita risultante avrà la colorazione spettrale che sarebbe data dalle cuffie di destinazione desiderate:

$$Y(f) = H_s(f) \cdot H_m(f) \cdot X(f) = \frac{H_t(f)}{H_m(f)} \cdot H_m(f) \cdot X(f) = H_t(f) \cdot X(f)$$

Analizzando quest'ultima equazione, è possibile dedurre che il filtro di simulazione nel dominio del tempo deve corrispondere a un sistema dato dalla cascata di due filtri, il primo con risposta impulsiva data da:

$$h_i(t) = \mathcal{F}^{-1}\left[\frac{1}{H_m(f)}\right]$$

e il secondo con la stessa risposta impulsiva delle cuffie di destinazione $h_t[n]$.

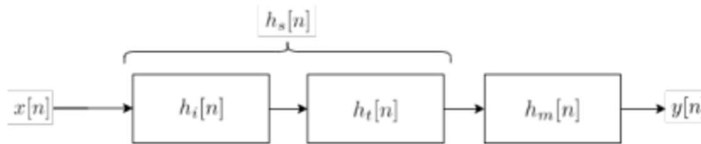

Figure 3: Schema completo del modello

Infine, lo schema generale che descrive il funzionamento del simulatore di cuffie è mostrato nella Figura 3.

3.1 Ottenimento delle misurazioni di Risposta Impulsiva utilizzate nel progetto

Le risposte impulsionali delle cuffie utilizzate in questo plugin sono state raccolte dal dataset Crinacle, una figura popolare nella comunità degli audiofili, che contiene più di 500 risposte impulsionali di cuffie diverse. Queste risposte impulsionali sono state misurate utilizzando il software Room Eq. Wizard (REW) e l'approccio swept-sine (tecnica coinvolge la generazione di un segnale sinusoidale che viene gradualmente spostato lungo una gamma di frequenze specifica).

Successivamente, le risposte impulsionali sono state elaborate utilizzando il software REW e il linguaggio di programmazione Python. Sono stati applicati algoritmi di elaborazione del segnale per ottimizzare le risposte e rimuovere eventuali artefatti indesiderati.

Infine, le risposte impulsionali elaborate sono state convertite in file audio nel formato Wav (Waveform Audio File Format) con un campionamento a 48 kHz. Questi file audio rappresentano le risposte impulsionali delle cuffie e vengono utilizzati nel processo di simulazione del profilo uditivo delle cuffie nel plugin.

In breve, la procedura comprende la raccolta delle risposte impulsionali dal dataset Crinacle, l'elaborazione delle risposte con software e algoritmi specifici, e la conversione in file audio Wav per essere utilizzate nel processo di simulazione delle cuffie.

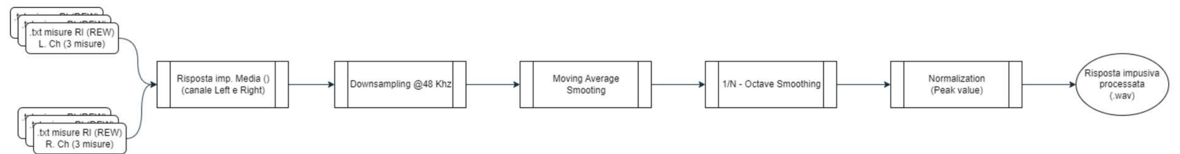

Figure 4: Pipeline estrazione risposta impulsiva dai file prodotti da REW

Il lavoro di ricerca ha utilizzato un totale di dieci diversi modelli di cuffie ad alta fedeltà, quattro delle quali sono state impiegate come monitor e dieci come target.

High Fidelity Headphones List and Characteristics					
Headphone Name	**Monitor**	**Target**	**Acoustic Design**	**Sensitivity**	**Impedance**
AKG K240	✓	✓	Semi-Open	98.67 dB/mW	78.32 Ω
Beyerdynamic T1	✓	✓	Open	99.37 dB/mW	819.01 Ω
Beyerdynamic DT990	✓	✓	Open	93.42 dB/mW	267.61 Ω
Sennheiser HD600	✓	✓	Open	100.54 dB/mW	352.12 Ω
AKG K701		✓	Open	90.64 dB/mW	66.11 Ω
Audio Technica ATH-AWKT		✓	Closed	102 dB/mW	48.01 Ω
Focal Stellia		✓	Closed	103.85 dB/mW	38.53 Ω
Sennheiser HD650		✓	Open	100.45 dB/mW	344.89 Ω
Sennheiser HD800S		✓	Open	100.09 dB/mW	300 Ω
Shure SRH1540		✓	Closed	99.14 dB/mW	45.81 Ω

Figure 5: Tabella delle cuffie coinvolte nell'esperimento

Figure 6: Immagini delle cuffie coinvolte nell'esperimento

4 VST, DAWS e JUCE

Figure 7: Plugin VST Fruity Convolver caricato sul DAW Fruity Loop Studio

4.1 Formato di un Plugin Audio: VST (Virtual Studio Technology)

VST (Virtual Studio Technology) è uno standard di interfaccia per plugin audio digitali sviluppato da Steinberg. Questo standard consente di estendere le funzionalità dei software audio digitali, come i DAW (Digital Audio Workstation), aggiungendo nuovi effetti sonori, strumenti virtuali e altre funzionalità di elaborazione del suono.

I plugin VST sono ampiamente utilizzati nell'industria musicale e audio professionale per la registrazione, la produzione musicale, il missaggio e il mastering. Offrono un'ampia gamma di effetti e strumenti che consentono agli utenti di modificare e migliorare il suono in modi creativi e tecnici.

I plugin VST utilizzano principalmente due formati di file. Il formato VST, con estensioni come .vst o .vst3, è l'estensione originale sviluppata da Steinberg. Questi file possono essere caricati e utilizzati all'interno di un software di produzione musicale compatibile con il formato VST. Il formato VSTi, con estensioni come .vst, .vsti o .dll, è specifico per gli strumenti virtuali, che sono plugin VST dedicati alla generazione di suoni e strumenti musicali.

I plugin VST vengono caricati e utilizzati all'interno di un software di produzione musicale per elaborare e modificare i segnali audio.

4.2 La framework JUCE

JUCE è un framework di sviluppo software C++ multi-piattaforma, creato da ROLI (ora parte di Raw Material Software Ltd.). È ampiamente utilizzato per la creazione di applicazioni audio professionali, tra cui plugin VST.

Figure 8: Logo JUCE

JUCE offre un'ampia gamma di funzionalità per lo sviluppo di plugin audio, compresi strumenti per il design dell'interfaccia utente, l'elaborazione del segnale audio, la gestione dei file audio e altro ancora. Questa libreria semplifica il processo di sviluppo dei plugin, offrendo una solida base di codice e una vasta gamma di componenti predefiniti.

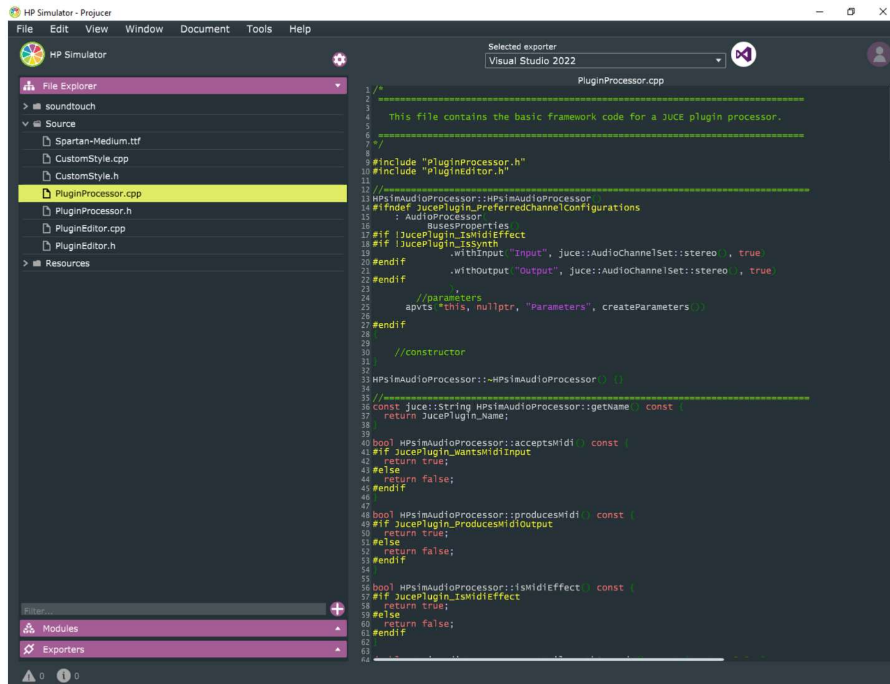

Figure 9: Projucer framework

L'utilizzo di JUCE nella progettazione del plugin VST per la simulazione dei profili uditivi di cuffie offre numerosi vantaggi. JUCE permette di creare interfacce utente intuitive e reattive, supporta la gestione delle risorse audio e semplifica l'implementazione delle funzionalità di elaborazione del segnale audio. Inoltre, JUCE offre la possibilità di creare plugin VST compatibili con diverse piattaforme, tra cui Windows, macOS e Linux.

Grazie alla sua flessibilità e alle sue potenti funzionalità, JUCE si è affermato come una scelta popolare tra gli sviluppatori di plugin audio professionali. È utilizzato da numerosi produttori di software audio e consente di creare plugin di alta qualità con un'efficace gestione delle risorse e delle prestazioni.

5 HP Simulator

Il plugin *HP Simulator* (Figura 10)è molto simile nella processazione ad un plugin di convoluzione (ad esempio fruity convolver di Image-line), ma con la

Figure 10: Interfaccia grafica del Plugin *HP Simulator*

differenza che non presenta i parametri di dryness e wetness. Si è scelto di evitare di includerli in quanto non sarebbero necessari allo scopo principe del progetto ovvero la simulazione accurata di dispositivi

5.1 Interfaccia utente

Il plugin include un'interfaccia utente intuitiva, realizzata utilizzando i componenti grafici offerti dalla libreria JUCE, che permette agli utenti di regolare i parametri e di visualizzare le informazioni sul profilo uditivo selezionato. Il plugin presenta una serie di parametri per agevolare l'applicazione del filtro, tra cui:

1. Pulsante per l'apertura del file .wav di risposta impulsiva
2. Grafico della risposta impulsiva (normalizzata)
3. Tre knob di input , output gain (applicati prima e dopo l'applicazione della convoluzione) e uno di mix che regola l'intensità dell'effetto (utilizzato principalmente per verificare che la risposta impulsiva non presenti difetti)
4. Un pulsante di bypass dell'effetto, per poter continuare l'ascolto dell'audio senza attivare il filtro

5.2 Generazione dei filtri $h_s[N]$

Come già spiegato per effettuare la simulazione è necessario costruire un filtro $h_s[N]$ tale che permetta di elimiare l'effetto delle cuffie monitor.

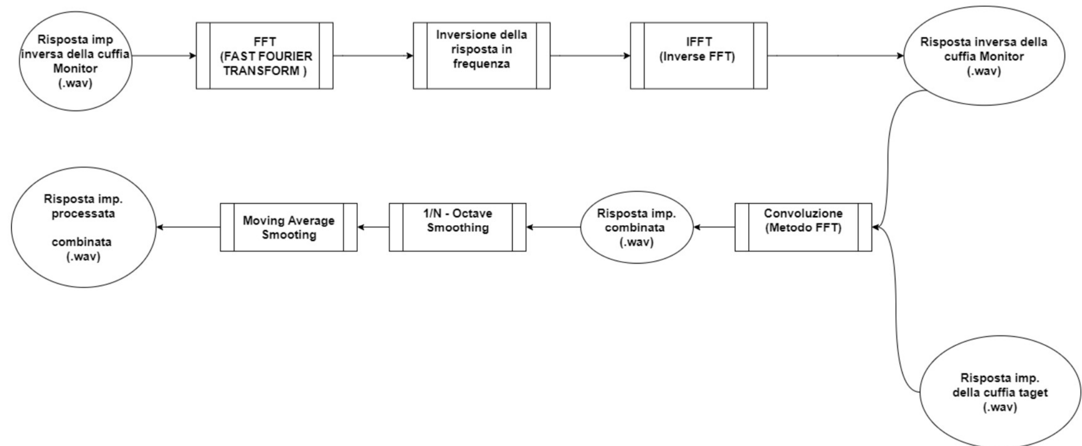

Figure 11: Pipeline di elaborazione per la generazione delle IR utilizzate da HP Sim.

Questa processazione (descritta dalla pipeline in figura 11) non è eseguita dal pligin ma è preprocessata tramite del codice python (in paricolare tramite le librerie Numpy e Scipy per convoluzione e FFT). Le risposte impulsive $h_s[N]$ create quindi per ogni combinazione di cuffia monitor e target sono memorizzate all'interno del plugin in formato .wav a 48Khz

5.3 Codice Del plugin

Un plugin JUCE tipico è composto da diversi componenti, file e metodi importanti che svolgono ruoli specifici. Ecco una breve panoramica dei componenti comuni in un plugin JUCE:

PluginProcessor.h:

- `PluginProcessor` (classe): Questa classe rappresenta il processore audio del plugin. Contiene i membri e i metodi per gestire l'elaborazione audio, la configurazione dei parametri, la gestione delle risorse e altro ancora.
- `prepareToPlay(double sampleRate, int samplesPerBlock)`: Questo metodo viene chiamato prima dell'inizio della riproduzione e consente di preparare il processore audio per la riproduzione. Viene tipicamente utilizzato per inizializzare vari componenti, come filtri, generatori di riverbero, ecc.
- `processBlock(juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midiMessages)`: Questo metodo è fondamentale ed è responsabile dell'elaborazione dei blocchi audio nel plugin. Prende in input un oggetto `juce::AudioBuffer` contenente i dati audio in ingresso e un oggetto `juce::MidiBuffer` contenente gli eventi MIDI. All'interno di questo metodo, vengono applicati gli effetti, i filtri, i riverberi, ecc., al segnale audio.

PluginProcessor.cpp:

- Implementazione dei metodi dichiarati in `PluginProcessor.h`: Questo file contiene l'implementazione dei metodi dichiarati nella classe `PluginProcessor`. Qui puoi trovare l'implementazione del metodo `prepareToPlay()`, `processBlock()`, `releaseResources()` e altri metodi specifici del tuo plugin.
- Inizializzazione e gestione dei parametri: Nel file `.cpp` del `PluginProcessor`, è possibile trovare il codice per l'inizializzazione dei parametri del plugin utilizzando l'oggetto `AudioProcessorValueTreeState`. Questo include l'aggiunta dei parametri, la definizione delle loro caratteristiche (nome, valore predefinito, intervallo, ecc.) e l'associazione ai controlli dell'interfaccia utente.
- Gestione dei bus audio: Nel `PluginProcessor.cpp`, è possibile trovare il codice per la configurazione dei bus audio utilizzati dal plugin, ad esempio specificando il numero di canali in ingresso e in uscita e le configurazioni di routing audio.

PluginEditor.h e **PluginEditor.cpp:**

- Questi file riguardano la parte dell'interfaccia utente del plugin. `PluginEditor.h` contiene la dichiarazione della classe `PluginEditor` e `PluginEditor.cpp` contiene l'implementazione dei relativi metodi. Questi file includono la creazione e la gestione degli elementi dell'interfaccia utente come pulsanti, slider, etichette, ecc., e l'associazione dei controlli ai parametri del `PluginProcessor`.

5.3.1 Caricamento della risposta impulsiva

Il caricamento e la formattazione della IR filtro viene effettuata principalmente da 3 metodi

- **void HPsimAudioProcessorEditor::openButtonClicked()** Il metodo `openButtonClicked()` è un listener attaccato al pulsante per caricare l'IR nell'editor del plugin. Quando il pulsante viene cliccato, questo metodo viene eseguito. All'interno di questo metodo, è possibile implementare la logica per il caricamento dell'IR da un file o da una risorsa esterna (vedi punto di codice 1). Una volta caricato l'IR, è possibile chiamare la funzione `loadImpulseResponse()` per gestire l'IR caricato e aggiornare l'effetto di riverbero del plugin.

  Listing 1: Metodo `openButtonClicked()`

```
void HPsimAudioProcessorEditor::openButtonClicked() {
    //....initial code to set the directory.....
   //Utility per aprire la cartella contenente le IR

  fileChooser = std::make_unique<juce::FileChooser>("Choose a
      support IR File (WAV, AIFF, OGG)...", initialDirectory,
      "*.wav;*.aif;*.aiff;*.ogg", true, false);

  auto chooserFlags = juce::FileBrowserComponent::openMode |
      juce::FileBrowserComponent::canSelectFiles;

  fileChooser->launchAsync(chooserFlags, [this](const
      juce::FileChooser& fc) {

     auto file = fc.getResult();

  //legge il file e chiama loadImpulseResponse
}
```

- **loadImpulseResponse()** La funzione `loadImpulseResponse()` è responsabile del caricamento dell'Impulse Response (IR), che rappresenta la risposta all'impulso di un sistema. L'IR è un file audio che viene utilizzato per simulare l'effetto di riverbero di un ambiente. In questa funzione, l'IR viene caricato in un oggetto `AudioBuffer<float>` chiamato `originalIRBuffer` (vedi punto di codice 3). Successivamente, viene normalizzato il segnale dell'IR utilizzando la tecnica di normalizzazione peak (vedi punto di codice 2). Questo assicura che l'IR abbia un volume adeguato e evita distorsioni indesiderate.

  Listing 2: Normalizzazione dell'Impulse Response (IR)

```
// Normalizzazione peak dell'IR
```

```
float globalMaxMagnitude = originalIRBuffer.getMagnitude(0,
    originalIRBuffer.getNumSamples());
originalIRBuffer.applyGain(1.0f / (globalMaxMagnitude + 0.01));
//...
```

Listing 3: Caricamento dell'Impulse Response (IR)

```
void HPsimAudioProcessor::loadImpulseResponse() {
  //.....
  // trim IR signal
  int numSamples = originalIRBuffer.getNumSamples();
  int blockSize =
      static_cast<int>(std::floor(this->getSampleRate()) / 100);
  int startBlockNum = 0;
  int endBlockNum = numSamples / blockSize;
  float localMaxMagnitude = 0.0f;
  while ((startBlockNum + 1) * blockSize < numSamples) {
    localMaxMagnitude =
        originalIRBuffer.getMagnitude(startBlockNum * blockSize,
            blockSize);
    // find the start position of IR
    if (localMaxMagnitude > 0.001) {
      break;
    }
    ++startBlockNum;
  }

  localMaxMagnitude = 0.0f;
  while ((endBlockNum - 1) * blockSize > 0) {
    --endBlockNum;
    localMaxMagnitude =
        originalIRBuffer.getMagnitude(endBlockNum * blockSize,
            blockSize);
    // find the time to decay by 60 dB (T60)
    if (localMaxMagnitude > 0.001) {
      break;
    }
  }

  int trimmedNumSamples;
  if (endBlockNum * blockSize < numSamples) {
    trimmedNumSamples = (endBlockNum - startBlockNum) * blockSize -
        1;
  } else {
    trimmedNumSamples = numSamples - startBlockNum * blockSize;
  }
  modifiedIRBuffer.setSize(originalIRBuffer.getNumChannels(),
      trimmedNumSamples,
                          false, true, false);
```

```
  for (int channel = 0; channel <
       originalIRBuffer.getNumChannels();
       ++channel) {
    for (int sample = 0; sample < trimmedNumSamples; ++sample) {
      modifiedIRBuffer.setSample(
          channel, sample,
          originalIRBuffer.getSample(channel,
                                     sample + startBlockNum *
                                         blockSize));
    }
  }



}
```

- **updateImpulseResponse()** La funzione `updateImpulseResponse()` viene chiamata dopo aver caricato l'IR per aggiornare l'effetto di convoluzione. Questa funzione prende in input l'IR modificato (`modifiedIRBuffer`) e lo passa al convolutore (`convolver`) (vedi punto di codice 4).

Listing 4: Aggiornamento del Convolutore

```
void
    HPsimAudioProcessor::updateImpulseResponse(juce::AudioBuffer<float>
    irBuffer) {
convolver.loadImpulseResponse(std::move(irBuffer),
    this->getSampleRate(),
juce::dsp::Convolution::Stereo::yes,
juce::dsp::Convolution::Trim::no,
juce::dsp::Convolution::Normalise::yes);
}
```

5.3.2 Processazione audio in tempo reale del Plugin

L'effettiva processazione Audio in real time effettuata dal plugin viene mostrata nella pipeline in Figura 11. Essa si compone esclusivamente dell'applicazione del filtraggio tramite convoluzione effettuato da convolver.process(context) (punto 1) al buffer di dati in arrivo dal DAW (rappresentato da juce::AudioBuffer¡float¿ buffer (punto 2)) in ingresso con l'inpulso creato precedentemente ad-hoc per quella combinazione di cuffia monitor e cuffia target necessitata dall'utente

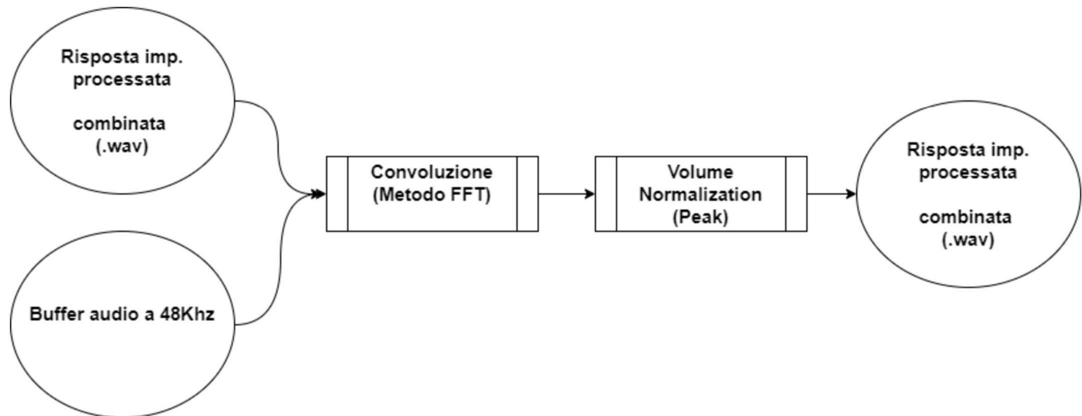

Figure 12: Pipeline della processazione in tempo reale del Plugin

Listing 5: Funzione `processBlock()`

```
void HPsimAudioProcessor::processBlock(juce::AudioBuffer<float> &buffer,
                                       juce::MidiBuffer &midiMessages) {
  juce::ScopedNoDenormals noDenormals;
  auto totalNumInputChannels = getTotalNumInputChannels();
  auto totalNumOutputChannels = getTotalNumOutputChannels();

  // Controlla il numero di canali
  for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
    buffer.clear(i, 0, buffer.getNumSamples());

  auto inputGainValue = apvts.getRawParameterValue("InputGain");
  auto outputGainValue = apvts.getRawParameterValue("OutputGain");
  auto dryWetMixValue = apvts.getRawParameterValue("DryWetMix");
  auto isBypassed = apvts.getRawParameterValue("Bypassed");

  if (isBypassed->load() == true) {
    return;
  }

  inputGainer.setGainDecibels(inputGainValue->load());
  outputGainer.setGainDecibels(outputGainValue->load());
```

```
    dryWetMixer.setWetMixProportion(dryWetMixValue->load() / 100.0f);
    auto block = juce::dsp::AudioBlock<float>(buffer);
    auto context = juce::dsp::ProcessContextReplacing<float>(block);

    % Guadagno del segnale in ingresso
    inputGainer.process(context);

    % Convoluzione (metodo FFT)
    convolver.process(context);

    % Normalizzazione a Peak Frequency
    float globalMaxMagnitude = buffer.getMagnitude(0,
        buffer.getNumChannels());
    buffer.applyGain(1.0f / (globalMaxMagnitude + 0.01));

    dryWetMixer.mixWetSamples(block);
    % Guadagno del segnale in uscita
    outputGainer.process(context);
}
```

Il segnale dopo essere stato convoluto viene normalizzato con una tecnica di normalizzazione basata sui picchi , comunemente chiamata "Peak Normalization" (nel codice al punto 3) Importante notare che dato che le IR sono state campionate con frequenza di 48 Khz è importante che anche le impostazioni del DAW siano concordi perchè fornisca buffer di campioni audio alla medesima frequenza

6 Conclusioni

In conclusione, questa tesi ha presentato il progetto e lo sviluppo di un plugin VST chiamato *HP Simulator* per la simulazione dei profili uditivi di cuffie HI-fi. La processazione audio in tempo reale del plugin avviene attraverso la convoluzione del segnale audio di ingresso con le risposte impulsive dei filtri, seguita dalla normalizzazione del segnale convoluto. In sintesi, il plugin *HP Simulator* rappresenta uno strumento utile per la simulazione dei profili uditivi degli headphones, offrendo un'interfaccia intuitiva e una buona qualità audio. Il suo sviluppo può contribuire al miglioramento delle simulazioni delle cuffie e all'ottimizzazione dell'esperienza di ascolto per gli utenti in ambito di produzione musicale o di selezione prodotto. Uno dei possibili aspetti di sviluppo per il futuro riguardano la valutazione soggettiva delle simulazioni effettuate dal plugin (e' importante che in queste si mantenga lo stesso volume durante i test per garantire una comparabilità accurata tra le diverse simulazioni). Dal punto di vista software , come giò sottolineato si potrebbe inserire all'interno del plugin la parte di processazione della risposta impulsiva , rendendo non necessaria la memorizzazione di un impulso per ogni coppia monitor/target, bensi rendendo necessaria solo la memorizzazione di una risposta impulsiva di ogni cuffia supportata. Inoltre come già detto l'utilizzo di un'implementazione di un altre tipologie di algoritmo di convoluzione (overlap and add) renderebbe il plugin piu efficiente dal punto di vista computazionale per la riproduzione istantanea.

7 Bibliografia

1. GORGHETTO, Luca. ”Design and development of a simulation system for the analysis of linear transformations in high fidelity headphones”, 2022.
2. JUCE documentation. Disponibile all’indirizzo: https://docs.juce.com/.
3. FL Studio. Sito ufficiale: https://www.image-line.com/flstudio/.
4. Repository di coneko su GitHub. Disponibile all’indirizzo: https://github.com/etosphere/coneko.