UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura

Corso di Laurea in Ingegneria Informatica, Elettronica e delle Telecomunicazioni

Correzione del tono adattativa basata su tecniche di phase vocoding

Adaptive pitch correction based on phase vocoding techniques

Relatore:

Prof. Riccardo Raheli

Tesi di Laurea di:

Riccardo Straccia

ANNO ACCADEMICO 2016-2017

Agli amici,
che mi sospingono fortissimo,
e non mi lasciano cadere mai.

Indice

Introduzione

L'industria musicale è cambiata profondamente negli ultimi decenni grazie all'avvento dell'elaborazione numerica dei segnali. Non solo questa ha migliorato la qualità nella registrazione, modifica e riproduzione musicale, ma ha anche facilitato l'accesso al settore grazie ai costi sempre più contenuti delle attrezzature e alla potenza sempre crescente dei moderni calcolatori. L'espansione del settore e le novità introdotte dall'elaborazione numerica hanno permesso inoltre a nuovi effetti audio di prendere piede nel mercato. Tra quelli di maggior successo si trovano sicuramente i sistemi di correzione del tono, in grado di "aggiustare" l'intonazione di una traccia vocale in modo da perfezionare la produzione di un brano senza dover ripetere numerose volte la registrazione.

Il funzionamento di un sistema di correzione del tono è necessariamente legato alla tipologia di segnali elaborati, ovvero segnali audio formati da componenti sinusoidali a frequenze armoniche (tipicamente tutti i segnali audio prodotti da strumenti musicali melodici, inclusa la voce umana, rientrano in questa categoria). Il sistema deve infatti estrarre le informazioni frequenziali da questi segnali e riconoscere una precisa frequenza (tipicamente la frequenza fondamentale) per poterli elaborare correttamente. In assenza di polifonia (quindi nel caso della riproduzione di una singola voce), i segnali di questo tipo sono caratterizzati da una distribuzione della densità spettrale di potenza concentrata in picchi nell'intorno della frequenza fondamentale e delle sue varie armoniche.
Dal momento che i segnali nell'ambito musicale sono generalmente non-stazionari, il loro spettro varia nel tempo, ed è importante quindi specificare che la distribuzione della densità spettrale di potenza indicata si riferisce allo **spettro a tempo breve** del segnale, che verrà meglio definito nel capitolo 1.

Un sistema di correzione del tono rientra nella categoria degli **effetti audio adattativi** (così definiti secondo la classificazione riportata in [1]) poiché ottenuto com-

binando un effetto audio con uno stadio di controllo adattativo. In particolare, poiché questo stadio di controllo è basato su informazioni ottenute parzialmente dall'esterno del sistema di elaborazione (operando sulla base di un insieme di tonalità a frequenze predefinite indipendenti dal segnale), un sistema di correzione del tono rientra nella categoria degli effetti **adattativi esterni**.

L'obiettivo di questa tesi consiste quindi nella realizzazione di un sistema automatico di correzione del tono, il cui utilizzo sia possibile sia su segnali pre-registrati, sia in tempo reale.
Nel capitolo 1 verranno gettate le basi teoriche del sistema di correzione, nel capitolo 2 verrà presentato un algoritmo sviluppato in linguaggio MATLAB, e nel capitolo 3 verranno riportati i risultati delle prove effettuate sull'algoritmo, le eventuali problematiche legate all'utilizzo di questo sistema, e verranno infine tratte le relative conclusioni.

Capitolo 1

Correzione adattativa del tono tramite phase vocoder

In questo capitolo verrà descritto l'intero processo di progettazione del sistema di correzione adattativa del tono. Nello specifico, verranno inizialmente presentate le basi teoriche sulle quali poggia il sistema di elaborazione del segnale, per poi spiegare nel dettaglio l'approccio con il quale viene effettuata la correzione del tono.

1.1 Il Phase Vocoder

1.1.1 Origine e basi del modello

Il *phase vocoder* è un modello di elaborazione del segnale presentato originariamente in [2] e notevolmente aggiornato e migliorato in [3].
Tale modello deve il suo nome al fatto di essere l'evoluzione del *vocoder*, nato come sistema di codifica vocale ma in seguito utilizzato anch'esso ampiamente in ambito musicale, come riportato in [4].
Il vocoder originale consiste in un parallelo di filtri passabanda distribuiti lungo la banda frequenziale d'interesse del segnale modulante. Questi filtri estraggono il contenuto di ciascuna sottobanda analizzata e permettono di sintetizzare un nuovo segnale a partire da un segnale portante filtrato in modo analogo al modulante. Originariamente questa tecnica conservava solo informazioni sull'energia, e quindi sul modulo dello spettro del segnale modulante, cancellando completamente le informazioni di fase. Un modello tecnicamente dettagliato è descritto dall'inventore di questa tecnologia in [5]. In ambito musicale, il tradizionale vocoder rappresenta

un effetto tipico, utilizzato in modo pionieristico anche nelle prime composizioni di musica elettronica. Tipicamente viene utilizzato per modulare suoni di sintetizzatore tramite la voce, in modo da ottenere un segnale che ricalca quello della voce, ma dal timbro alterato, ottenendo un effetto di "robotizzazione".
Il phase vocoder differisce dall'originale vocoder poiché conserva le informazioni sulla fase del segnale modulante, garantendo quindi una ricostruzione più accurata. Il suo utilizzo comporta tre stadi operativi, rappresentati in figura 1.1:

- **Analisi**: il segnale viene analizzato e ne vengono estratte le informazioni che lo caratterizzano in entrambi i domini di tempo e frequenza;
- **Modifica**: vengono alterate le informazioni estratte in modo da ottenere l'effetto desiderato sul segnale;
- **Sintesi**: viene ricostruito un nuovo segnale, ottenuto modificando il segnale di partenza.

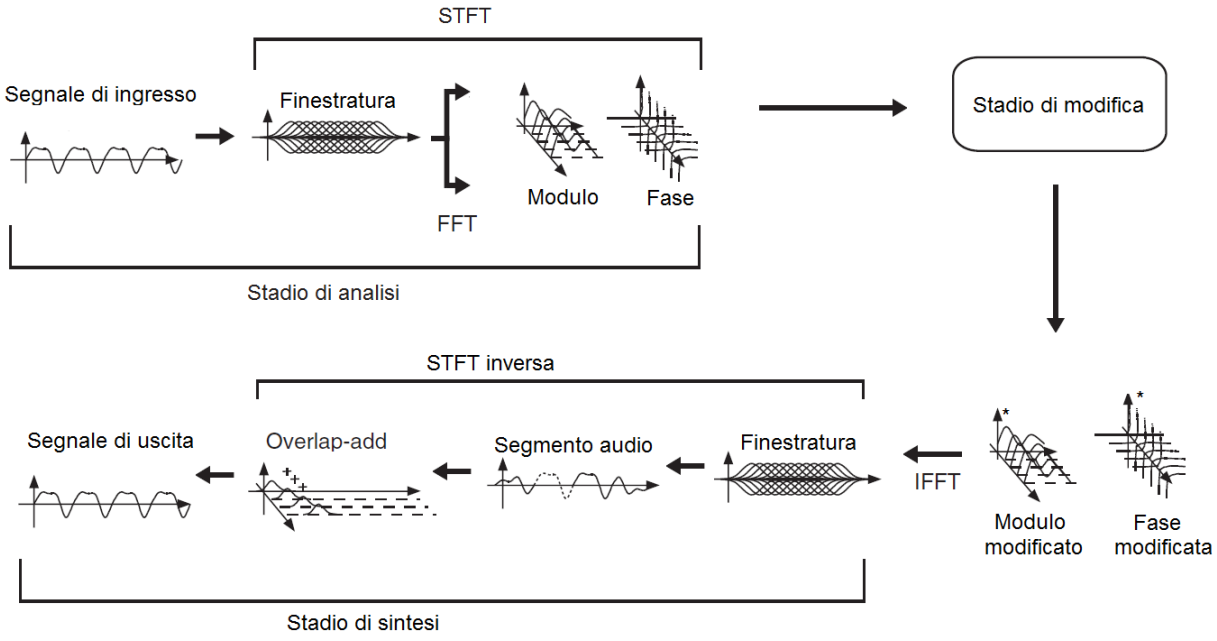

Figura 1.1: Rappresentazione grafica del principio di funzionamento del phase vocoder.

L'utilità del phase vocoder risiede nella capacità di questo modello operativo di elaborare segnali basandosi su una contemporanea rappresentazione tempo-frequenza. La sua popolarità è notevolmente aumentata grazie alle migliorie introdotte in [3]: l'utilizzo del phase vocoder non più attraverso un banco di filtri passabanda, ma

con un equivalente modello basato sulla trasformata di Fourier ha reso incredibilmente conveniente l'applicazione di questo algoritmo per l'elaborazione digitale di segnali audio in svariati ambiti, introducendo convenienti metodologie per l'applicazione di particolari effetti, molte di queste riportate in [1].

Nello stadio di analisi viene fatto utilizzo di una trasformazione in frequenza denominata **Short Time Fourier Transform** (abbreviata *STFT*). Operando su segnali digitali, ci si riferirà d'ora in poi con il termine STFT alla sua variante a tempo discreto.

1.1.2 Stadio di Analisi: STFT

L'STFT è una variante della tradizionale trasformata di Fourier in grado di estrarre una rappresentazione spettrale dinamica di un segnale, ovvero l'evoluzione temporale dello spettro di una successione non periodica di campioni, quindi dal contenuto frequenziale non costante nel tempo.
Se si indicano con m l'indice temporale della STFT e con k l'indice frequenziale (come nella **Discrete Fourier Transform**, abbreviata DFT), si può formalizzare la STFT come:

$$X[m,k] = \sum_{n=-\infty}^{\infty} x[n]h[n-m]W_N^{-nk} \tag{1.1}$$
$$W_N = e^{j2\pi/N}$$

in cui $h[n]$ denota la finestra temporale utilizzata, che viene fatta scorrere da $-\infty$ a ∞.
Dal momento che la finestra temporale ha una durata finita N (che supponiamo da $n = 0$ a $n = N-1$), per un valore fissato $m = m_0$ la STFT assume la forma di una DFT a meno di un fattore di traslazione temporale, corrispondente a una componente lineare di fase nel dominio della frequenza:

$$X[m_0,k] = \sum_{n=m_0}^{m_0+N-1} x[n]h[n-m_0]W_N^{-nk}. \tag{1.2}$$

Ponendo $n' = n - m_0$,

$$\begin{aligned} X[m_0,k] &= \sum_{n'=0}^{N-1} x[n'+m_0]h[n']W_N^{-n'k} \cdot W_N^{-m_0k} \\ &\equiv DFT\{x[n'+m_0]h[n']\} \cdot W_N^{-m_0k} \end{aligned} \tag{1.3}$$

dove il primo termine indica la DFT della sequenza di N campioni moltiplicati per la finestra, in un sistema di riferimento che li pone all'origine. Il termine $W_N^{-m_0k}$ comporta, nel dominio della frequenza, una rotazione lineare di fase, che nel tempo corrisponde a una rotazione circolare dei campioni della DFT. Questo termine verrà tuttavia ignorato nelle prossime considerazioni riguardanti questo particolare sistema, e si valuteranno solamente gli aspetti relativi alla DFT, poiché questo termine è ininfluente ai fini dell'elaborazione; le motivazioni sono meglio specificate nel paragrafo 1.1.3 e in [7].
Il risultato di una STFT può essere visto quindi come una matrice contenente nei

vettori colonna il risultato delle singole DFT applicate ciascuna alla finestratura del segnale corrispondente all'istante m indicato da quella determinata colonna. I vettori riga contengono invece i valori assunti da un singolo campione della trasformata nella sua evoluzione temporale.
La corretta impostazione della STFT rappresenta un punto cardine nell'analisi effettuata dal phase vocoder. Un'analisi con parametri ben dimensionati permette infatti di ottenere una modifica efficace e una ricostruzione valida del segnale finale. Essendo N il numero di campioni utilizzati nella DFT e definendo con T la durata del segnale $x[n]$, quindi:

$$X[m,k] = \begin{Bmatrix} X_{11} & X_{12} & \dots & X_{1(T-N)} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N1} & X_{N2} & \dots & X_{N(T-N)} \end{Bmatrix}$$

Dal momento che le effettive implementazioni della DFT prevedono l'utilizzo dell'algoritmo computazionalmente efficiente di **Fast Fourier Transform** (abbreviata FFT), è preferibile scegliere la dimensione N della finestra come potenza di 2. Così facendo, la quantità di campioni di $x[n]$ in ingresso ad ogni singola trasformata, N, è sempre una quantità per cui è possibile calcolare la FFT.
È necessario fissare anche un valore di intervallo della funzione finestra, ossia la quantità di campioni di cui questa scorre da una DFT all'altra. Questo valore viene chiamato **fattore di salto** (denotato con R). Fissato R, è possibile riformulare l'espressione della STFT in questo modo, a partire dall'espressione contenuta nella (1.1):

$$X[sR,k] = \sum_{n=-\infty}^{\infty} x[n]h[n-sR]W_N^{-nk} \tag{1.4}$$

a cui segue, dalla (1.3), scegliendo $n' = n - sR$, la forma analoga:

$$X[sR,k] = DFT\{x[n'+sR]h[n']\} \cdot W_N^{-sRk}. \tag{1.5}$$

Il termine s è un intero arbitrario che rappresenta l'indice della colonna corrispondente alla DFT effettuata nell'istante sR. In questo modo R indica uno specifico "tasso di campionamento" con il quale viene effettuata la STFT sul segnale. Il valore R deve sottostare a delle regole precise per garantire alcune importanti proprietà. Innanzitutto, R non può superare la durata della finestra N, altrimenti

questa perde parte dell'informazione tra una DFT e l'altra. Deve essere quindi scelto in modo che:

$$R \leq N.$$

Utilizzando un valore di R minore di N si ottiene una sovrapposizione parziale (*overlap*) delle finestre.
Nel caso di una finestra rettangolare è possibile scegliere $R = N$. L'utilizzo di tale finestra, tuttavia, causa l'insorgenza di un fenomeno denominato *frequency leakage*, che consiste in una tipica "sbavatura" (*smearing*) delle informazioni frequenziali estratte dalla trasformata, spesso intollerabile per un'analisi qualitativamente valida. Più precisamente, la finestra rettangolare nel tempo fa sì che ogni contributo spettrale del segnale originario venga espanso secondo l'andamento dello spettro della finestra stessa.
Utilizzando altre tipologie di finestre per ridurre l'insorgenza di questo fenomeno è d'altro canto necessario sfruttare la sovrapposizione, per evitare che l'attenuazione apportata dai bordi di esse comporti una perdita di informazione.
Per questo motivo la scelta di R è fortemente legata ad h (e alla sua lunghezza N). Per assicurarsi una ricostruzione valida, è necessario infatti scegliere un valore di R che soddisfi la seguente proprietà:

$$\sum_{s=-\infty}^{\infty} h[n - sR] = 1, \forall n \in \mathbb{Z}. \tag{1.6}$$

Se sussiste questa proprietà, a cui ci si riferisce con la dicitura **Constant Overlap-Add** (o con l'acronimo COLA), allora è possibile ricostruire un segnale $x[n]$ dalla sovrapposizione dei suoi segmenti, idealmente sommandoli nelle loro posizioni originali nel tempo, come riportato in [6].
Definendo

$$x_s[n] \triangleq x[n]h[n - sR]$$

e supponendo che la (1.6) sia verificata, si può scrivere quindi:

$$\begin{aligned} \sum_{s=-\infty}^{\infty} x_s[n] &= \sum_{s=-\infty}^{\infty} x[n]h[n - sR] \\ &= x[n] \sum_{s=-\infty}^{\infty} h[n - sR] \\ &\equiv x[n]. \end{aligned} \tag{1.7}$$

In [8] è presente un'analisi completa di numerose funzioni di finestratura e dei relativi valori di **sovrapposizione raccomandata** (*recommended overlap*, con acronimo ROV), ovvero valori per cui viene rispettata la proprietà. Una finestra che rappresenti un buon candidato nel nostro caso è rappresentato dalla finestra di **Hann** (anche nota come finestra di Hanning o finestra a coseno rialzato), per la quale è riportato un valore di ROV del 50% (grafico in figura 1.2), e la cui espressione è:

$$w_{hann}[n] = 0.5 \cdot (1 - \cos(2\pi \tfrac{n}{N-1}))$$

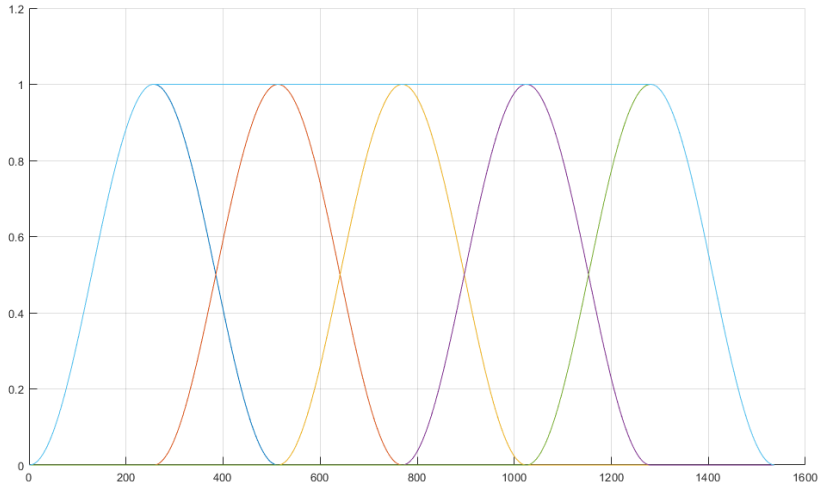

Figura 1.2: Finestre di Hann con 50% di sovrapposizione e somma verificante la proprietà 1.6.

La proprietà COLA ha tuttavia un'analoga variante nel caso in cui oltre ad utilizzare una finestra di analisi, si utilizzi una finestra anche per la ricostruzione del segnale nella fase di sintesi, chiamata proprietà di **Weighted Overlap-Add** (abbreviata WOLA, presentata in [7]), che verrà meglio discussa nel prossimo paragrafo. Per lo stesso valore di R sulla stessa finestra h, in generale, non sussistono entrambe le proprietà. Per valori differenti di R, tuttavia, per la stessa tipologia di finestra può sussistere l'una o l'altra proprietà (a meno di un fattore di scala). In questa realizzazione del phase vocoder verrà scelta una finestra che soddisfi la proprietà WOLA in modo da poter usare sia una finestra in fase di analisi che una in fase di sintesi.

Infine, è necessario imporre che il segnale preso in esame abbia una durata di T

campioni, con

$$T = N + aR \tag{1.8}$$

per un generico valore a intero positivo, per garantire che la finestra attraversi totalmente il segnale in un numero intero di traslazioni. Si tratta di un requisito banale, in quanto effettuando opportunamente un riempimento con degli zeri (*zero padding*) al termine della sequenza originale, qualunque segnale di lunghezza finita può essere "allungato" in un segnale di durata T, con $\frac{T-N}{R} = a$.

Ridefinendo per semplicità la notazione della STFT:

$$X_s[k] \triangleq X[sR, k]$$

l'espressione della STFT si può scrivere esplicitando modulo e fase dalla (1.4):

$$\begin{aligned} X_s[k] &= \sum_{n=-\infty}^{\infty} x[n]h[n - sR]W_N^{-nk} \\ &= |X_s[k]| \cdot e^{j\phi_s[k]} \end{aligned} \tag{1.9}$$

$X_s[k]$ può essere chiamato lo **spettro a tempo breve** (*short-time spectrum*) del segnale $x[n]$ all'istante sR.

1.1.3 Stadio di Sintesi: ricostruzione del segnale

Dal momento che la ricostruzione avviene dopo la modifica, lo spettro del segnale utilizzato per la ricostruzione verrà indicato con la notazione $\hat{X}_s[k]$ poiché generalmente diverso da quello di analisi.
La ricostruzione del segnale in questa realizzazione del modello del phase vocoder si basa sul principio di sovrapposizione e somma pesata (*WOLA*), e fa utilizzo del medesimo R utilizzato nell'analisi. Per ricostruire il segnale a partire dal suo spettro short-time $\hat{X}_s[k]$, si effettua la trasformata inversa dello spettro moltiplicato per una rotazione di fase opposta a quella presente nello stadio di analisi, ottenendo un segmento, o granulo (*grain*), del segnale nel tempo corrispondente alla posizione sR analoga a quella da cui è stato estratto. A questo si applica una seconda finestratura prima della sovrapposizione e somma. Questa seconda finestra, chiamata finestra di sintesi, attenua le discontinuità presenti agli estremi del segmento audio, eliminando eventuali artefatti legati alle modifiche che introducono variazioni non-lineari, come lo spostamento di tono o la compressione/espansione temporale (*pitch shifting* e *time stretching*).
La scelta della finestra di sintesi è tipicamente coincidente con quella della finestra di analisi. Utilizzando due volte la finestra temporale, la proprietà *WOLA* risulta essere:

$$\sum_{s=-\infty}^{\infty} h^2[n - sR] = 1, \forall n \in \mathbb{Z}. \tag{1.10}$$

Dopo aver applicato la finestra di sintesi, il segmento viene sommato agli altri segmenti ottenuti precedentemente per costruire il segnale sintetizzato dopo la modifica.
Si denota quindi con $\hat{x}_s[n]$ il segmento risintetizzato, traslato nella sua posizione temporale originaria e rimoltiplicato per la finestra $h[n - sR]$:

$$\hat{x}_s[n] \triangleq IDFT\{\hat{X}_s[k] \cdot W_N^{sRk}\} \cdot h[n] = h[n] \cdot \sum_{k=0}^{N-1} \hat{X}_s[k] W_N^{nk} W_N^{sRk}. \tag{1.11}$$

Se non è stata apportata alcuna modifica tra lo stadio di analisi e quello di sintesi nello schema del phase vocoder, allora:

$$\hat{X}_s[k] = X_s[k].$$

Quindi:

$$\hat{x}_s[n] = IDFT\{X_s[k] \cdot W_N^{sRk}\} \cdot h[n - sR].$$

Sostituendo $X_s[k]$ con la (1.5) (sostituendo la variabile n quindi con $n' = n - sR$ e tenendo conto del cambio di notazione), segue:

$$\begin{aligned}\hat{x}_s[n' + sR] &= IDFT\{DFT\{x[n' + sR]h[n']\} \cdot W_N^{-sRk} \cdot W_N^{sRk}\} \cdot h[n'] \\ &= IDFT\{DFT\{x[n' + sR]h[n']\}\} \cdot h[n'] \\ &= x[n' + sR]h^2[n']\end{aligned} \tag{1.12}$$

e quindi, sapendo che $n = n' + sR$, si può scrivere:

$$\hat{x}_s[n] = x[n]h^2[n - sR]. \tag{1.13}$$

Notare come il fattore W_N^{sRk} annulli la rotazione di fase che compare nella formula di analisi (1.5), dal momento che si usa il medesimo valore di R. In una realizzazione pratica di un algoritmo di correzione del tono non è dunque necessario introdurre questi fattori a patto di traslare correttamente il segmento del segnale alla posizione originale. Una costruzione più generale del phase vocoder è presente in [7]. Riferendoci ad a come al valore considerato nella (1.8), e chiamando $y[n]$ l'intero segnale ricostruito, questo risulta infine essere:

$$y[n] = \sum_{s=0}^{a} \hat{x}_s[n]$$

e quindi,

$$\begin{aligned}y[n] &= \sum_{s=0}^{a} x[n] \cdot h^2[n - sR] \\ &= x[n] \sum_{s=0}^{a} h^2[n - sR]\end{aligned} \tag{1.14}$$

Utilizzando una finestra che rispetti la (1.10), allora

$$\sum_{s=0}^{a} h^2[n - sR] = 1, \forall n \in [c, d]$$

Se il nostro segnale $x[n]$ racchiude tutti i campioni utili nell'intervallo $[c, d]$ (per ottenere questo basta effettuare un semplice inserimento di zeri all'inizio e alla fine del segnale), allora gli intervalli $[0, c)$ e $(d, T]$ (dove il valore della somma dei quadrati delle finestre non è unitario) non influiscono sull'elaborazione del segnale.

Sotto queste prerogative, il phase vocoder privato dello stadio di modifica risulta quindi una funzione identità:

$$y[n] \equiv x[n].$$

È importante specificare la scelta dello stesso R per entrambe le finestre poiché non si operano modifiche temporali nella ricostruzione del segnale. La scelta di un valore $R_s \neq R$ implica infatti che i termini di fase non si annullino, ma diano luogo nel dominio del tempo a una rotazione circolare dei campioni legata alla differenza dei due termini R_s e R. Con questa tecnica si realizzano tipicamente operazioni di time stretching.

In questa particolare realizzazione verrà utilizzata la **finestra di Hann** moltiplicata per un fattore di ampiezza $A = \sqrt{\frac{2}{3}}$ e con una sovrapposizione del 75% ($R = \frac{N}{4}$), la quale verifica la proprietà della (1.10) come è possibile vedere nel grafico 1.3. Si è scelto quindi:

$$h[n] = \sqrt{\frac{2}{3}} \cdot w_{hann}[n]. \tag{1.15}$$

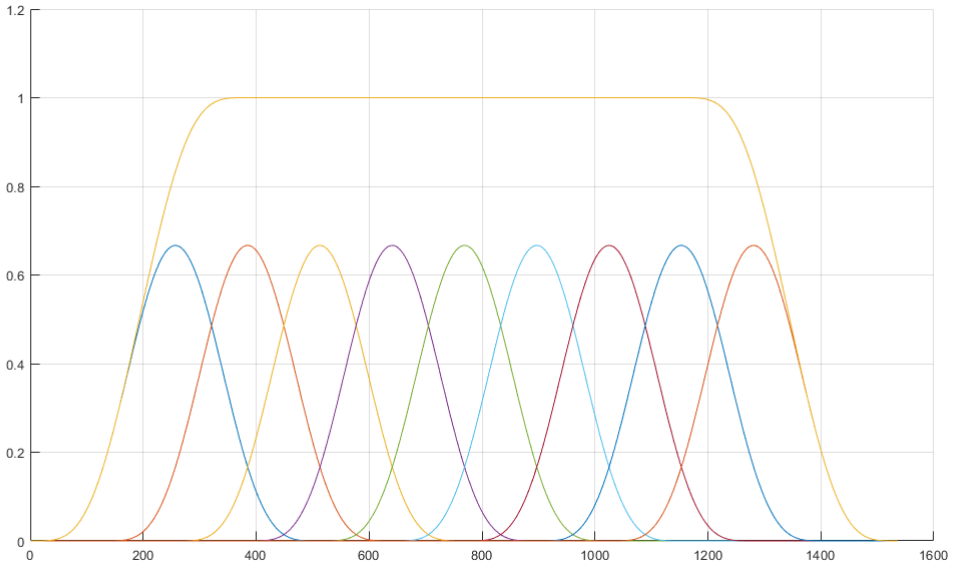

Figura 1.3: Quadrati delle finestre descritte da (1.15) con sovrapposizione del 75% e relativa somma verificante la proprietà 1.10.

La scelta di una finestra con elevato fattore di sovrapposizione può risultare più impegnativa nella potenza di calcolo richiesta, dal momento che viene effettuata l'elaborazione di più segmenti, ma può fornire un risultato qualitativo più elevato.

1.2 Estrazione del tono

Nell'introduzione si è stabilito che un sistema di correzione del tono è un effetto audio adattativo basato sull'estrazione del contenuto frequenziale del segnale, più precisamente nel riconoscimento della tonalità della nota riprodotta. In questo paragrafo verranno dettagliatamente illustrate le modalità con cui è possibile estrarre con adeguata precisione la frequenza istantanea di un segnale nel tempo tramite la sua STFT.
È importante sottolineare, in questo frangente, come il metodo riportato non sia l'unico con il quale è possibile effettuare l'estrazione del tono. Esistono diverse tipologie di approccio a questo problema, e un'esaustiva analisi e comparazione delle loro caratteristiche viene effettuata in [9].

1.2.1 Stima sul modulo

Si parte dal presupposto che il segnale preso in esame rispetti le considerazioni effettuate nell'introduzione, ovvero che sia formato da componenti sinusoidali a frequenze armoniche. In tal caso una prima stima della tonalità della nota deriva direttamente dal modulo della STFT. Indicato con H il numero di armoniche, il modulo $|X_s[k]|$ del segnale analizzato per un generico valore di s presenta un andamento del tipo mostrato in figura 1.4, e può essere approssimato da una sommatoria di funzioni delta di Kronecker.

Definita la funzione delta di Kronecker:

$$\delta[k] \triangleq \begin{cases} 1, & \text{se } k = 0 \\ 0, & \text{se } k \neq 0 \end{cases}$$

si può scrivere la sommatoria che approssima $|X_s[k]|$:

$$|X_s[k]| \simeq \sum_{i=1}^{H} A_i \cdot \delta[k - k_i].$$

In questa formula A_i è l'ampiezza della relativa armonica (A_1 indica l'ampiezza della fondamentale) e k_i indica il campione della DFT in cui è localizzata l'armonica (k_1 indica il campione in cui è localizzata la fondamentale).

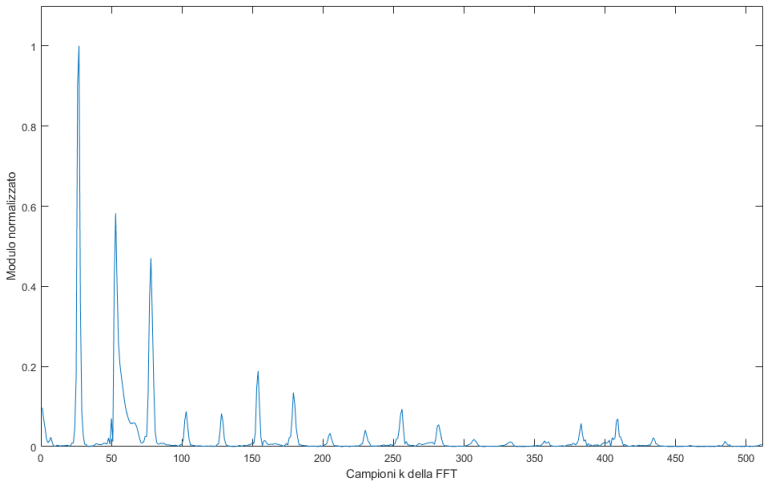

Figura 1.4: Modulo di un segmento di $N = 1024$ campioni della parte vocale della canzone *Tom's Diner*. Si riporta metà del modulo, essendo lo spettro simmetrico.

Individuare l'indice di questi campioni della DFT fornisce una prima stima della tonalità della nota, dal momento che un campione k_i corrisponde alla frequenza $f_i = \frac{F_c \cdot k_i}{N}$, con F_c frequenza di campionamento del segnale $x[n]$.
L'individuazione di questi campioni si riduce a un problema di ricerca di punti di massimo locale, per il quale esistono numerosi algoritmi di risoluzione numerica. Un metodo approssimativo ma estremamente semplice e valido in questo contesto viene fornito in [10]; esso consiste nel considerare massimo locale un campione della DFT il cui modulo sia maggiore di quello misurato nei 4 punti ad esso più vicini, e che superi un certo valore di soglia.

L'individuazione di un massimo assoluto, in generale, non corrisponde all'individuazione della frequenza fondamentale: succede spesso, infatti, che una frequenza armonica possieda più potenza della frequenza fondamentale. La fondamentale consiste invece nel primo picco significativo, generalmente localizzato nelle basse frequenze.
In questa realizzazione, per motivi meglio illustrati in seguito, non sarà necessaria l'individuazione della frequenza fondamentale, ma sarà sufficiente individuare uno qualunque dei picchi armonici, per semplicità quello di valore massimo. Individuare solamente un massimo assoluto rispetto al cercare tutti i punti di massimo locale permette di semplificare l'operazione di ricerca della frequenza e impedisce inoltre la possibilità che venga individuata una struttura armonica errata, dovuta ad eventuali massimi locali non legati alle armoniche del segnale, ma individuati a

causa di un elevato rumore o di un valore di soglia inadeguato.

Il problema più grande riguardante la stima effettuata sul modulo risiede nel fatto che i campioni della trasformata sono equispaziati di un valore corrispondente alla risoluzione frequenziale della DFT eseguita, ovvero $\triangle f = \frac{F_c}{N}$. È quindi necessario stabilire il margine di errore massimo $\frac{\triangle f}{2}$ sulla stima. Questo errore, per valori realistici di F_c e N, è tipicamente troppo grande per poter considerare questa stima sufficiente a distinguere le tonalità delle note ed effettuare la correzione. Si ricorre dunque a una tecnica resa possibile dalla struttura intrinseca della STFT e dalle proprietà della trasformata di Fourier, per l'estrazione della frequenza istantanea in modo estremamente più preciso.

1.2.2 Phase unwrapping e perfezionamento della stima

Una procedura di *phase unwrapping* (letteralmente: svolgimento di fase) permette di visualizzare la risposta di fase di un segnale nell'intervallo $[0; \infty)$, piuttosto che nell'intervallo $[-\pi; +\pi)$ come tipicamente ottenuto calcolando l'argomento di una generica DFT; tale procedura è di importanza fondamentale nella realizzazione dell'intero sistema di correzione del tono poiché costituisce la base sia per una precisa estrazione della tonalità delle note, sia per la correzione stessa. I metodi di phase unwrapping vengono descritti in [1], [9] e [11]. Per poter effettuare il phase unwrapping è necessario calcolare progressivamente la fase del segnale nella sua evoluzione temporale. Per questo motivo un algoritmo di phase unwrapping è particolarmente conveniente se si utilizza la STFT, poiché questa fornisce le informazioni di fase utilizzate dall'algoritmo nella forma necessaria.

Prima di illustrare l'algoritmo, viene definita una funzione che estrae l'argomento principale, definita come $p(\phi)$:

$$p(\phi) \triangleq [(\phi + \pi) \mod (2\pi)] - \pi.$$

Tale funzione fa utilizzo dell'operatore modulo, definito, per un generico valore b, come:

$$x \mod b \triangleq x - \left\lfloor \frac{x}{b} \right\rfloor \cdot b$$

in cui $\lfloor \cdot \rfloor$ denota la funzione parte intera.
Il risultato della funzione p applicata a un generico angolo ϕ riporta il valore di

questo angolo nell'intervallo $[-\pi; +\pi)$; di fatto la funzione p costituisce la procedura inversa al phase unwrapping.

Si supponga di avere un segnale $x[n]$ costituito da una sinusoide a frequenza $\tilde{f}_0$, e se ne calcoli la STFT con fattore di salto R. Sia k_0 il campione individuato da una stima iniziale sul modulo (corrispondente quindi all'arrotondamento all'intero più vicino del valore $\frac{\tilde{f}_0 \cdot N}{F_c}$).

La fase calcolata in corrispondenza dell'istante sR e sul campione k_0 della trasformata viene indicata con:

$$\phi_s[k_0] = \arg(X_s[k_0])$$

Basandoci su questo valore di fase calcolato e sulla corrispondenza del campione k_0 con la frequenza $f_0 = \frac{F_c \cdot k_0}{N}$, è possibile calcolare il valore di fase obiettivo previsto al successivo istante $(s+1)R$:

$$\hat{\phi}_{(s+1)}[k_0] = \phi_s[k_0] + 2\pi \frac{f_0}{F_c} R$$

Dal momento che la frequenza f_0 non corrisponde in realtà alla frequenza $\tilde{f}_0$ della sinusoide che compone il segnale originale, ma riporta l'errore di misura riportato precedentemente e di valore al più $\frac{\triangle f}{2}$, il valore di fase misurato realmente all'istante $(s+1)R$ sarà:

$$\phi_{s+1}[k_0] = \arg(X_{s+1}[k_0])$$

per cui, generalmente:

$$\phi_{s+1}[k_0] \neq \hat{\phi}_{(s+1)}[k_0].$$

La misura di $\phi_{s+1}[k_0]$ permette di calcolare la deviazione causata dall'errore tra la fase target e quella realmente calcolata, ristretta nell'intervallo $[-\pi; +\pi)$:

$$\phi_D[k_0] = p(\phi_{s+1}[k_0] - \hat{\phi}_{(s+1)}[k_0]).$$

A questo punto si può calcolare la fase *unwrapped* come la fase obiettivo più la deviazione ϕ_D.

$$\begin{aligned} \tilde{\phi}_{(s+1)}[k_0] &= \hat{\phi}_{(s+1)}[k_0] + \phi_D[k_0] \\ &= \phi_s[k_0] + 2\pi \frac{f_0}{F_c} R + p(\phi_{s+1}[k_0] - \hat{\phi}_{(s+1)}[k_0]) \\ &= \phi_s[k_0] + 2\pi \frac{f_0}{F_c} R + p(\phi_{s+1}[k_0] - \phi_s[k_0] - 2\pi \frac{f_0}{F_c} R). \end{aligned} \tag{1.16}$$

Sapendo che la frequenza angolare di una sinusoide si può individuare come derivata della fase nel tempo, è possibile ora trovare con una precisione estremamente maggiore la frequenza $\tilde{f}_0$. Si costruisce la variazione di fase come differenza tra la fase calcolata in sR e quella calcolata in $(s+1)R$:

$$\begin{aligned}\triangle\phi_{s+1}[k_0] &= \tilde{\phi}_{(s+1)}[k_0] - \phi_s[k_0] \\ &= \phi_s[k_0] + 2\pi\frac{f_0}{F_c}R + p(\phi_{s+1}[k_0] - \phi_s[k_0] - 2\pi\frac{f_0}{F_c}R) - \phi_s[k_0] \\ &= 2\pi\frac{f_0}{F_c}R + p(\phi_{s+1}[k_0] - \phi_s[k_0] - 2\pi\frac{f_0}{F_c}R).\end{aligned} \tag{1.17}$$

A questo punto la stima di $\tilde{f}_0$ diventa, in forma di rapporto incrementale della variazione di fase:

$$\tilde{f}_0 = \frac{1}{2\pi}\frac{\triangle\phi_{s+1}[k_0]}{R}F_c. \tag{1.18}$$

In una prova di questo sistema, utilizzando un tono puro con $F_c = 48$kHz e $N = 1024$, è stato possibile perfezionare la stima delle corrispondenti frequenze discretizzate passando da uno scarto di errore di circa 23 Hz ad uno di circa 0.02 Hz, cioè tre ordini di grandezza inferiore.

1.3 Correzione del tono

Nel precedente paragrafo è stato illustrato come la tonalità (frequenza) della nota riprodotta nel segmento audio di lunghezza N acquisito all'istante sR venga estratta dalla variazione del contenuto di fase della sua STFT.
Il metodo descritto in questo paragrafo descriverà dunque come effettuare una modifica di tale contenuto in modo da ottenere l'effetto di spostamento di tono del segnale, comunemente denominato *pitch shifting*.
Il riferimento del sistema per effettuare la correzione del tono è costituito da delle frequenze predefinite. Queste vengono calcolate secondo lo standard moderno per l'accordatura degli strumenti, l'**ISO 16**. Questo prevede che la frequenza di riferimento del **La** centrale di un pianoforte (nella notazione inglese **A4**) sia di 440 Hz, con un'accuratezza dell'accordatura di ± 0.5 Hz.
Quando la frequenza raddoppia, si ripete la stessa nota ad un'ottava superiore (la frequenza della nota A5, ad esempio, si trova a 880 Hz). Da questi presupposti si ricavano, per intervalli di semitono, tutte le note nell'intervallo di interesse per la correzione.
Un intervallo di semitono è il più piccolo intervallo tonale nel sistema musicale moderno, corrispondente al rapporto tra la frequenza di una nota e la frequenza della nota appena precedente, ed il suo valore è $2^{\frac{1}{12}}$. Un intervallo doppio rispetto a quello di semitono, cioè di $2^{\frac{1}{6}}$ è invece un intervallo di tono.
La scala composta da 12 note intervallate l'una dall'altra da un semitono prende il nome di **scala cromatica**. Le altre due scale più conosciute ed utilizzate nella musica moderna sono la **scala maggiore** e la **scala minore**. Sono entrambe scale composte da sette suoni (otto considerando l'estensione di un'ottava), e i loro intervalli sono così strutturati a partire dalla tonica:
Scala maggiore : Intervalli di semitono tra la 3° e la 4° nota e la 7° e l'8° nota della scala, intervalli di tono altrove.
Scala minore : Intervalli di semitono tra la 2° e la 3° nota e la 6° e la 7° nota della scala, intervalli di tono altrove.

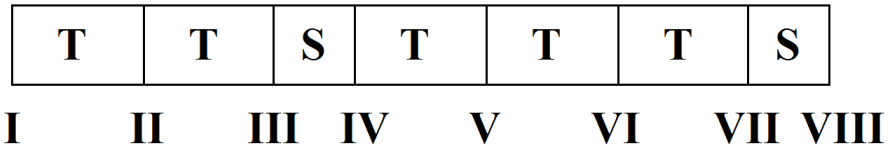

Figura 1.5: Scala maggiore

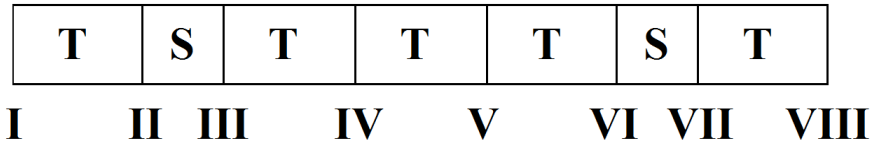

Figura 1.6: Scala minore

L'effetto di spostamento del tono desiderato consiste in una discretizzazione della tonalità della nota verso la più vicina tonalità precalcolata. La funzione di discretizzazione viene quindi definita sulla base di un vettore precalcolato $V(j)$ contenente le frequenze delle note standard in un determinato intervallo di interesse.
La scelta di questo intervallo dipende dalle tipologie di segnali su cui si lavora: operando sulla voce umana è realistico supporre che la frequenza fondamentale non scenda sotto i 100 Hz e che le armoniche di potenza più alta non superino i 3000 Hz (valori riportati da [12]).
Definendo con f_s la frequenza individuata dall'algoritmo di stima nel segmento di indice s del segnale analizzato, e con $\hat{f}_s(f_s)$ la sua più vicina frequenza "corretta" definita all'interno del vettore $V(j)$, la ricerca della frequenza $\hat{f}_s(f_s)$ a partire da f_s consiste in un banale problema di individuazione di un punto di minimo:

$$\hat{f}_s(f_s) \triangleq \arg\min_{V(j)}\{|f_s - V(j)|\} \tag{1.19}$$

Una rappresentazione grafica di questa funzione è contenuta in figura 1.7.
L'idea fondamentale dietro lo spostamento del tono per un segnale analizzato con STFT è quella di risintetizzare ciascun segmento audio $x_s[n]$ analizzato con la trasformata, utilizzando un contenuto di fase differente da quello del segnale originale.

$$X_s[k] = |X_s[k]| \cdot e^{j\phi_s[k]}$$

$$\hat{X}_s[k] = |X_s[k]| \cdot e^{j\psi_s[k]}.$$

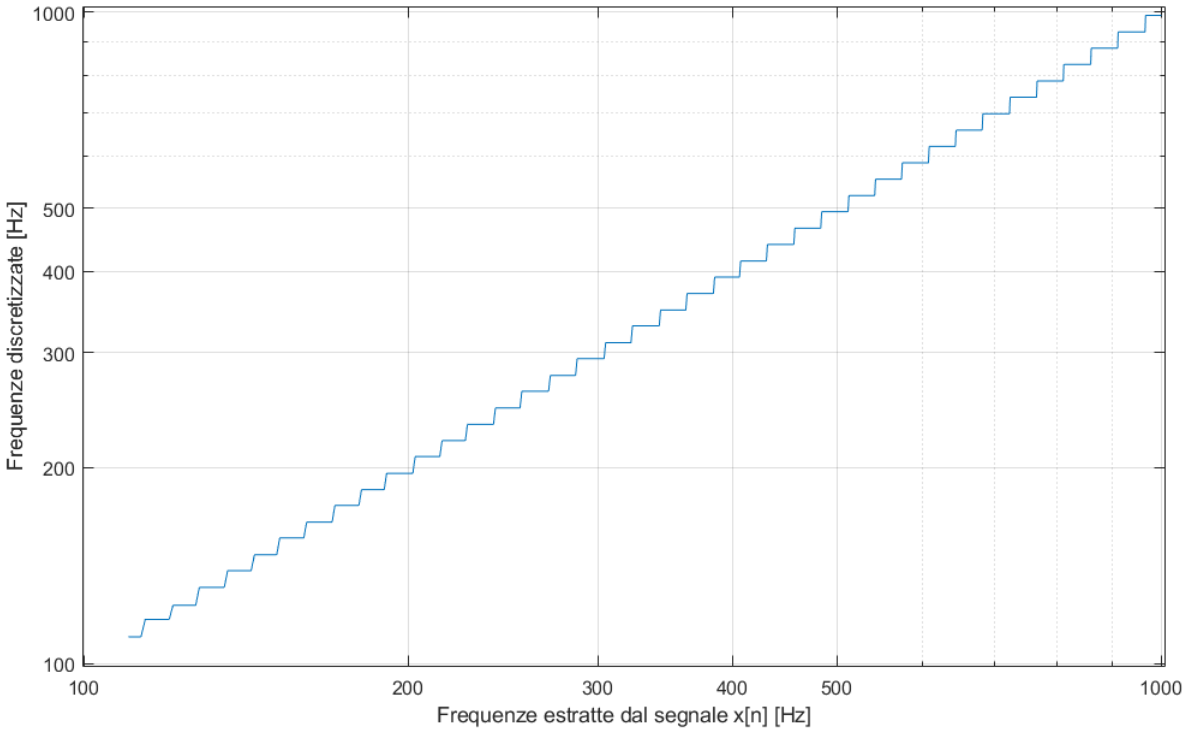

Figura 1.7: Discretizzazione delle frequenze tra 110 e 1000 Hz basata sulla scala cromatica con rappresentazione in scala logaritmica.

Come già descritto nei precedenti paragrafi, la variazione di fase per ogni campione k tra sR e $(s+1)R$ corrisponde ai valori $\triangle\phi_{s+1}[k]$, che per il campione k_0 in corrispondenza del punto di massimo del modulo identifica la frequenza fondamentale o una delle armoniche, secondo la (1.18). Esplicitando la variazione di fase, si ottiene la seguente formula inversa:

$$\triangle\phi_{s+1}[k_0] = \frac{2\pi f_s R}{F_c}$$

Si vuole ora modificare questa variazione di fase in modo che la tonalità si sposti da f_s a $\hat{f}_s(f_s)$. Individuando il rapporto $\alpha_s = \dfrac{\hat{f}_s}{f_s}$, si può costruire una variazione di fase $\triangle\psi_{s+1}$ in questo modo:

$$\begin{aligned} \triangle\psi_{s+1}[k_0] &\triangleq \frac{2\pi \hat{f}_s R}{F_c} \\ &= \alpha_s \cdot \frac{2\pi f_s R}{F_c} \\ &= \alpha_s \cdot \triangle\phi_{s+1}[k_0]. \end{aligned} \tag{1.20}$$

Sapendo che:

$$\phi_{s+1}[k] = p(\phi_s[k] + \triangle\phi_{s+1}[k])$$

allora, conseguentemente:

$$\psi_{s+1}[k] = p(\psi_s[k] + \alpha_s \cdot \triangle\phi_{s+1}[k]). \tag{1.21}$$

L'effetto ottenuto è quello di una rotazione di fase $\triangle\psi_{s+1}[k]$ "modificata" di un fattore α_s rispetto alla rotazione $\triangle\phi_{s+1}[k]$.
Questo fattore è tale da trasformare $\triangle\phi_{s+1}[k_0]$ da una rotazione corrispondente alla frequenza frequenza f_s a una rotazione corrispondente alla frequenza discretizzata $\hat{f}_s$. Similmente, le rotazioni delle armoniche di f_s vengono trasformate in rotazioni corrispondenti alle rispettive armoniche della frequenza discretizzata $\hat{f}_s$. Il risultato equivalente è quello di avere nel vettore $\psi_{s+1}[k]$ la fase estratta dallo spettro di un ipotetico segnale con la medesima struttura armonica di $X_s[k]$, ma relativa ad una frequenza fondamentale $\hat{f}_s$ piuttosto che f_s.
La possibilità di operare un corretto spostamento per tutte le frequenze basandosi sul calcolo del valore α_s è il motivo per cui nella precedente fase di estrazione del tono è stato sufficiente individuare una sola delle armoniche, piuttosto che l'intera struttura armonica del segnale $x_{s+1}[n]$.

Capitolo 2

Programmazione dell'algoritmo in MATLAB

Per la realizzazione dell'algoritmo, è stato scelto il linguaggio MATLAB per le seguenti motivazioni:

- Estrema facilità di gestione di dati in forma di vettore e file audio, supporto nativo al calcolo di numeri complessi;
- Implementazione già presente dell'algoritmo di **FFT**;
- Integrazione semplificata di dispositivi multimediali di acquisizione e riproduzione con supporto ai driver **ASIO**;
- Possibilità di esportazione del codice in C/C++ per supporto a differenti piattaforme;
- Presenza di implementazioni pre-esistenti di altri algoritmi basati sul phase vocoder in codice MATLAB in [1] e [11].

2.1 Algoritmo di correzione

L'algoritmo sviluppato si presenta sotto forma di funzione MATLAB, denominata *autotune.m*. Il contenuto dell'algoritmo verrà illustrato in questo paragrafo.

Codice 2.1: Funzione *autotune()*

```
1 function [grain_out, grain_phi, grain_psi] =  autotune(
    grain, grain_old_phi, grain_old_psi, comparisonPitches,
     FS)
```

Nel codice 2.1 viene dichiarata la funzione *autotune.m*, la quale può essere richiamata da un programma esterno ricevendo in ingresso dei parametri:

- *grain* è un vettore di campioni provenienti da un flusso audio in ingresso, di dimensione N (in riferimento al paragrafo 1.1.2);
- *grain_old_phi* e *grain_old_psi* sono due differenti vettori contenenti rispettivamente il contenuto di fase del precedente *grain* di ingresso e del precedente *grain* di uscita;
- *comparisonPitches* è un vettore contenente le frequenze discretizzate di riferimento (vedere le considerazioni nel paragrafo 1.3 riguardo le scale musicali);
- *FS* contiene il valore della frequenza di campionamento di *grain*.

Codice 2.2: Parametri del phase vocoder

```
% PHASE VOCODER
s_win    = length(grain);
hs_win   = s_win/2;
R        = s_win/4;
w1       = 0.5 * hanning(s_win, 'periodic');
w2       = w1;
omega    = 2*pi*R*[0:hs_win-1]'/s_win;
```

Nel codice 2.2 vengono impostati i parametri del phase vocoder. La variabile *s_ win* assume il valore N acquisito direttamente da *grain*. Viene memorizzata la metà di questo valore in *hs_ win* per poter utilizzare una tecnica che permette di dimezzare il tempo di elaborazione di questo algoritmo. R contiene il valore del fattore di salto (in riferimento al paragrafo 1.1.2).
w1 e *w2* sono le finestre di analisi e sintesi. Viene utilizzata la medesima finestra per entrambi i casi, rispettando la proprietà (1.10) come riportato nel paragrafo 1.1.3.
Viene inoltre definito *omega*, vettore contenente i valori $2\pi\frac{f_i}{F_c}R$ nella forma equivalente $2\pi\frac{k}{N}R$, $\forall k \in [0, N-1]$.
Operando solo su metà dello spettro si utilizzano $k \in [0, \frac{N}{2} - 1]$.

Codice 2.3: Elaborazione di *grain*

```
% GRAIN PROCESSING
grain = grain .* w1;
f     = fft(grain);
f     = f(1:hs_win);
m     = abs(f);
phi   = angle(f);
```

Nel codice 2.3 viene moltiplicato il vettore *grain* per la finestra *w1*, e ne viene calcolata la *FFT*. Essendo *grain* un segnale reale, la sua trasformata gode di simmetria hermitiana. È quindi possibile operare su metà del vettore risultante dalla *FFT* senza perdere informazione utile, in modo da dimezzare la quantità di dati su cui si opera e migliorare l'efficienza dell'algoritmo. Dalla trasformata vengono quindi estratti modulo e argomento.

Codice 2.4: Estrazione del tono

```
% TONE EXTRACTION
delta_phi = omega + princarg(phi-grain_old_phi-omega);
[m_max, mMaxIndex] = max(m);
estimFrequency = 1/(2*pi)*delta_phi(mMaxIndex)*FS/R;
estimDifference = abs(comparisonPitches - estimFrequency);
[min_val, estimNearIndex] = min(estimDifference);
```

Nel codice 2.4 viene svolto il calcolo di *delta_phi*, come descritto dalla (1.17). Si procede quindi all'individuazione del campione *mMaxIndex* dove il modulo assume valore massimo, e si utilizza la (1.18) per ottenere una stima opportuna del tono. Si calcola infine l'indice *estimNearIndex* che rappresenta la frequenza nel vettore *comparisonPitches* che minimizza la distanza dalla frequenza stimata, come considerato nella (1.19).

Codice 2.5: Correzione del tono

```
% FREQ. SHIFTING
shiftRatio = comparisonPitches(estimNearIndex) /
   estimFrequency;
if isinf(shiftRatio) || isnan(shiftRatio)
    shiftRatio = 1;
end
psi   = princarg(grain_old_psi+delta_phi*shiftRatio);
```

Nel codice 2.5 viene effettuato il calcolo di un valore *shiftRatio* (corrispondente ad α nel paragrafo 1.3). Nel caso il valore assunto da *shiftRatio* dovesse essere inaccettabile a causa di un'eventuale stima errata o dell'elaborazione di un segnale inappropriato, lo si forza ad assumere un valore unitario, per cui non si opera alcuna modifica del segnale. Si calcola quindi il vettore *psi* contenente i valori di fase assunti dal segnale modificato.

Codice 2.6: Resintesi

```
% RESYNTHESIS
ft    = (m.* exp(1i*psi));
ft = [ ft(1:hs_win) ; conj(ft(hs_win:-1:1))];
grain_out = real(ifft(ft)).*w2;
grain_phi = phi;
grain_psi = psi;
```

Nel codice 2.6 viene infine ricostruito un vettore complesso a partire dai valori di modulo e fase modificata, e si applica una ricostruzione dell'altra metà dello spettro per simmetria hermitiana.
Si calcola quindi la trasformata inversa, moltiplicata per la finestra *w2*, e si costruiscono i parametri di uscita della funzione *autotune.m*.

2.2 Utilizzo in tempo reale

È possibile applicare l'algoritmo precedentemente descritto a qualunque flusso audio, sotto la fondamentale condizione di fornire in ingresso dei segmenti audio di lunghezza N a intervalli di $R = \frac{N}{4}$, per fare sì che ogni ciclo di elaborazione ottenga dal ciclo di acquisizione una segmento distante R campioni dal precedente, come previsto dall'algoritmo.
A questo proposito, è stato realizzato uno script MATLAB facente uso di dispositivi hardware di ingresso e uscita audio per acquisire e riprodurre in tempo reale (con un certo ritardo introdotto dall'acquisizione ed elaborazione dei campioni) un segnale modificato dall'algoritmo di correzione del tono.

Codice 2.7: Impostazioni *realtime.m*

```
1 TIME = 10;
2 SIZE = 1024;
3 FS = 48000;
4 KEY ='A'; % not used
5 OCTAVES = 8;
6 HOP = SIZE/4;
```

Nel codice 2.7 vengono definite alcune impostazioni del programma.

- *TIME* è una variabile contenente la durata dell'acquisizione in secondi;
- *SIZE* indica la dimensione N dei segmenti audio da mandare in ingresso all'algoritmo;
- *FS* indica la frequenza di campionamento con cui si acquisisce l'audio;
- *KEY*, seppure non utilizzata in questo specifico script, contiene la tonalità della scala con cui si desidera effettuare la correzione, nel caso non si voglia utilizzare la scala cromatica;
- *OCTAVE* indica l'estensione in ottave delle frequenze precalcolate;
- *HOP* corrisponde al valore R riportato più volte nel capitolo 2.

Codice 2.8: Inizializzazione variabili

```
1  [pitchesHz, majHz, minHz] = precalculateNotes(OCTAVES,
       KEY);
2
3  phi = zeros(SIZE/2,1);
4  psi = zeros(SIZE/2,1);
5  buffer_in_1 = zeros(HOP,1);
6  buffer_in_2 = zeros(HOP,1);
7  buffer_in_3 = zeros(HOP,1);
8  buffer_in_4 = zeros(HOP,1);
9  buffer_out_1 = zeros(HOP,1);
10 buffer_out_2 = zeros(HOP,1);
11 buffer_out_3 = zeros(HOP,1);
12 buffer_out_4 = zeros(HOP,1);
```

Nel codice 2.8 vengono inizializzate delle variabili.
La funzione *precalculateNotes* restituisce tre vettori contenenti le frequenze discretizzate con cui si opera la correzione.
I due vettori di fase con cui l'algoritmo in *autotune.m* opera vengono inizializzati a zero.
Vengono inoltre preparati 4 vettori di memoria di ingresso e 4 vettori di memoria di uscita (denotati con il termine *buffer*). Questi vettori, di dimensione pari a $R = \frac{N}{4}$ vengono utilizzati per poter acquisire a intervalli di R ma poter operare nell'algoritmo con un vettore di dimensione N.

Codice 2.9: Inizializzazione dispositivi hardware

```
1 deviceReader = audioDeviceReader(FS, HOP, 'Driver', '
    ASIO', 'Device', 'Focusrite USB ASIO');
2 deviceWriter = audioDeviceWriter('SampleRate', FS, '
    BufferSize', HOP, 'Driver', 'ASIO', 'Device', '
    Focusrite USB ASIO');
3 fileWriter = dsp.AudioFileWriter('SampleRate',
    deviceReader.SampleRate);
```

Nel codice 2.9 vengono inizializzati i dispositivi di acquisizione e riproduzione. Per una resa corretta di questo script è necessario l'utilizzo di driver **ASIO** (vedi [13]), che rappresentano una tecnologia disponibile per una vastissima gamma di schede audio, anche di fascia amatoriale, o nella versione di supporto universale *ASIO4ALL* (in questo caso il sistema è stato provato con una scheda *Focusrite Scarlett 6i6* dotata di driver ASIO dedicati). Il mancato utilizzo di driver di questo tipo o di un'adeguata attrezzatura per l'acquisizione comporta in molti casi un pessimo funzionamento del sistema, con l'insorgenza di ritardi e artefatti dovuti ad un'inadeguatezza prestazionale di natura hardware o software.

Nei parametri viene riportata la frequenza di campionamento e la dimensione della memoria di acquisizione viene impostata al valore R, così che ad ogni ciclo il dispositivo di acquisizione restituisca un vettore di R campioni e il dispositivo di riproduzione ne riproduca R.

Viene infine predisposto un gestore per la scrittura su file dei dati elaborati. Quest'ultima funzione è inclusa nel pacchetto **DSP System Toolbox** di MATLAB.

Codice 2.10: Acquisizione, elaborazione, riproduzione

```
tic
while toc < TIME
    buffer_in_1 = buffer_in_2;
    buffer_in_2 = buffer_in_3;
    buffer_in_3 = buffer_in_4;
    buffer_in_4 = deviceReader();

    toProcSignal = [buffer_in_1;buffer_in_2;
       buffer_in_3;buffer_in_4];

    [procSignal,phi,psi] = autotune(toProcSignal, phi,
        psi, pitchesHz, FS);

    reproducSignal = buffer_out_1;
    deviceWriter(reproducSignal);
    fileWriter(reproducSignal);

    buffer_out_1 = buffer_out_2 + procSignal(1:HOP);
    buffer_out_2 = buffer_out_3 + procSignal(HOP+1:2*
       HOP);
    buffer_out_3 = buffer_out_4 + procSignal(2*HOP
       +1:3*HOP);
    buffer_out_4 = procSignal(3*HOP+1:SIZE);
end
```

Il cuore del sistema in tempo reale, è descritto nel codice 2.10 e consiste nello scorrimento dei campioni acquisiti lungo le memorie di ingresso e uscita. All'inizio del ciclo il contenuto delle memorie di ingresso scorre: il contenuto della seconda memoria viene spostato nella prima, e così via fino alla quarta, che invece acquisisce R campioni dal dispositivo di ingresso. Le memorie vengono quindi concatenate in modo da formare un vettore *toProcSignal* di N campioni, che rappresenta il segnale da processare con il nostro algoritmo. L'algoritmo in *autotune.m* viene quindi richiamato e il segmento audio di uscita viene assegnato al vettore *procSignal*. Viene quindi riprodotto e scritto su file il segmento audio contenuto nella prima memoria di uscita.

Infine, viene effettuato lo scorrimento delle memorie di uscita in modo da rispettare la tecnica di *overlap and add* prevista dal phase vocoder. Nella prima memoria di uscita viene scritto il contenuto della seconda e vi vengono sommati i primi R campioni provenienti dall'elaborazione effettuata con l'algoritmo. Nella seconda viene copiato il contenuto della terza più i successivi R campioni, e così via fino all'ultima, in cui viene copiata l'ultima sequenza di R campioni in uscita dall'algoritmo.
L'esecuzione dell'algoritmo raggiunge il termine quando un timer indipendente raggiunge il valore di *TIME*.

Codice 2.11: Chiusura gestore

```
1 release(deviceReader);
2 release(deviceWriter);
3 release(fileWriter);
```

Nel codice 2.11 vengono infine chiusi i gestori che regolavano il flusso I/O di acquisizione, riproduzione e scrittura su file, e il programma può terminare.

Capitolo 3

Prova dell'algoritmo e risultati

3.1 Prove dell'algoritmo

L'algoritmo sviluppato è stato provato su tre differenti segnali:

- Un segnale composto da una singola sinusoide (tono puro), con frequenza di valore arbitrario diverso da quello di un tono già "corretto" (in questo caso $f_0 = 215$ Hz) e frequenza di campionamento $F_c = 48$kHz.
- Un segnale di *chirp* con andamento esponenziale (anche chiamato *sine sweep*) in un intervallo di frequenze dai 200 a 2000 Hz e frequenza di campionamento $F_c = 48$ kHz.
- Un segnale vocale cantato contenente le prime due strofe del brano *Tom's Diner* e frequenza di campionamento $F_c = 44.1$ kHz.

Tutte e tre le prove sono state effettuate con tre diversi valori di N, ovvero 512, 1024 e 2048, fissando sempre $R = \frac{N}{4}$. Differenze sostanziali sono state rilevate sia nella rappresentazione grafica sia nell'ascolto soggettivo solamente nel caso di *Tom's Diner*.
Una rappresentazione grafica dell'effetto della correzione viene realizzata applicando un algoritmo di stima della frequenza fondamentale nel tempo (contenuto in [1]) molto simile a quello utilizzato nel nostro sistema, e utilizzando MATLAB per tracciare il relativo grafico. Questo sistema di rappresentazione grafica interpone degli spazi vuoti laddove non sia in grado di stimare adeguatamente una frequenza fondamentale, ad esempio dove il segnale non rientri nella modellizzazione sinusoidale descritta nell'introduzione (tale è il caso delle finestre di segnale contenenti alcuni suoni consonantici, i cui campioni in frequenza hanno un andamento più simile a quello del rumore che non a quello di picchi a frequenze armoniche).

3.1.1 Prova effettuata su segnale sinusoidale

La sinusoide viene generata numericamente all'interno di MATLAB ad una frequenza definita come $f_0 = 215$ Hz. Prevedibilmente, la frequenza fondamentale viene riportata in figura 3.1 come una linea retta di altezza f_0.

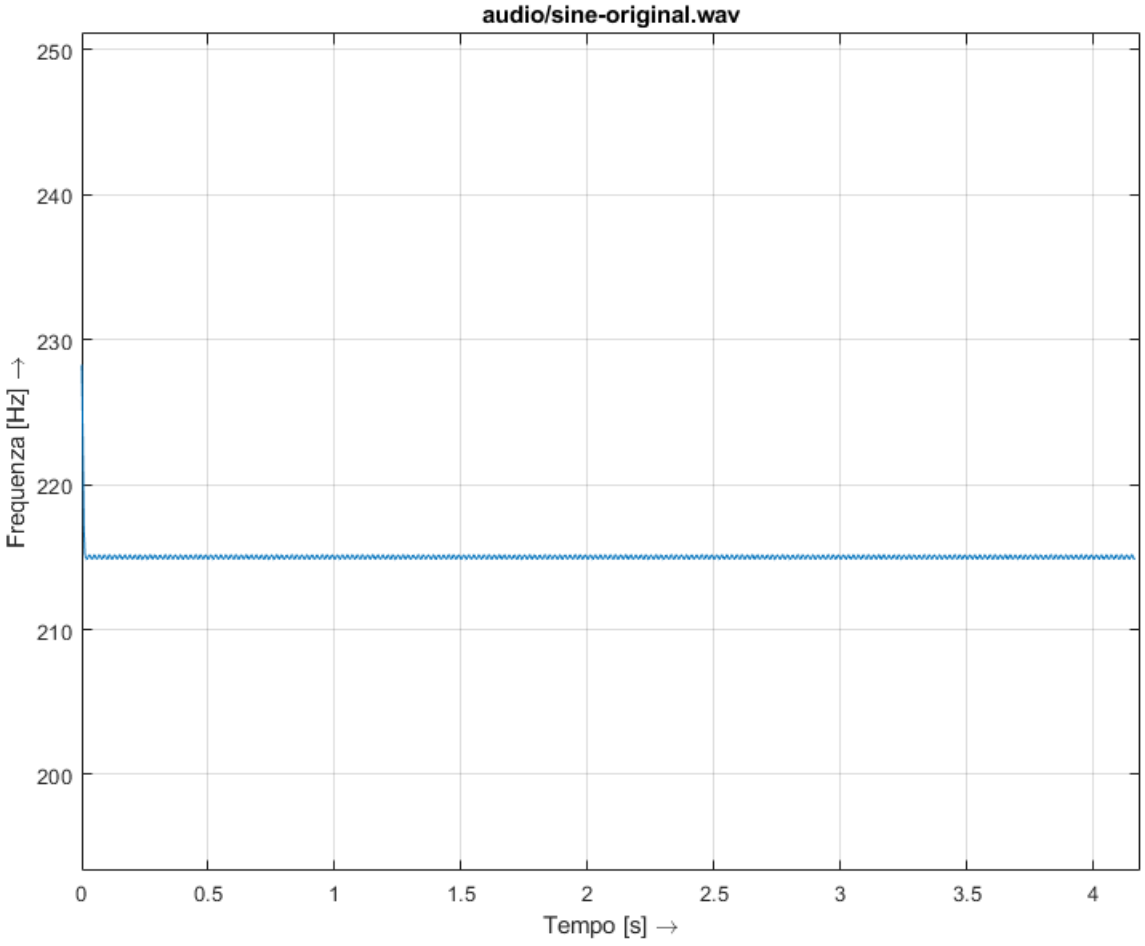

Figura 3.1: Sinusoide a 215 Hz.

La più vicina frequenza contenuta nel vettore di tonalità standard è $A3 = 220$ Hz, e la prova di correzione effettuata, la cui rappresentazione è riportata in figura 3.4, conferma i risultati previsti dalla teoria.

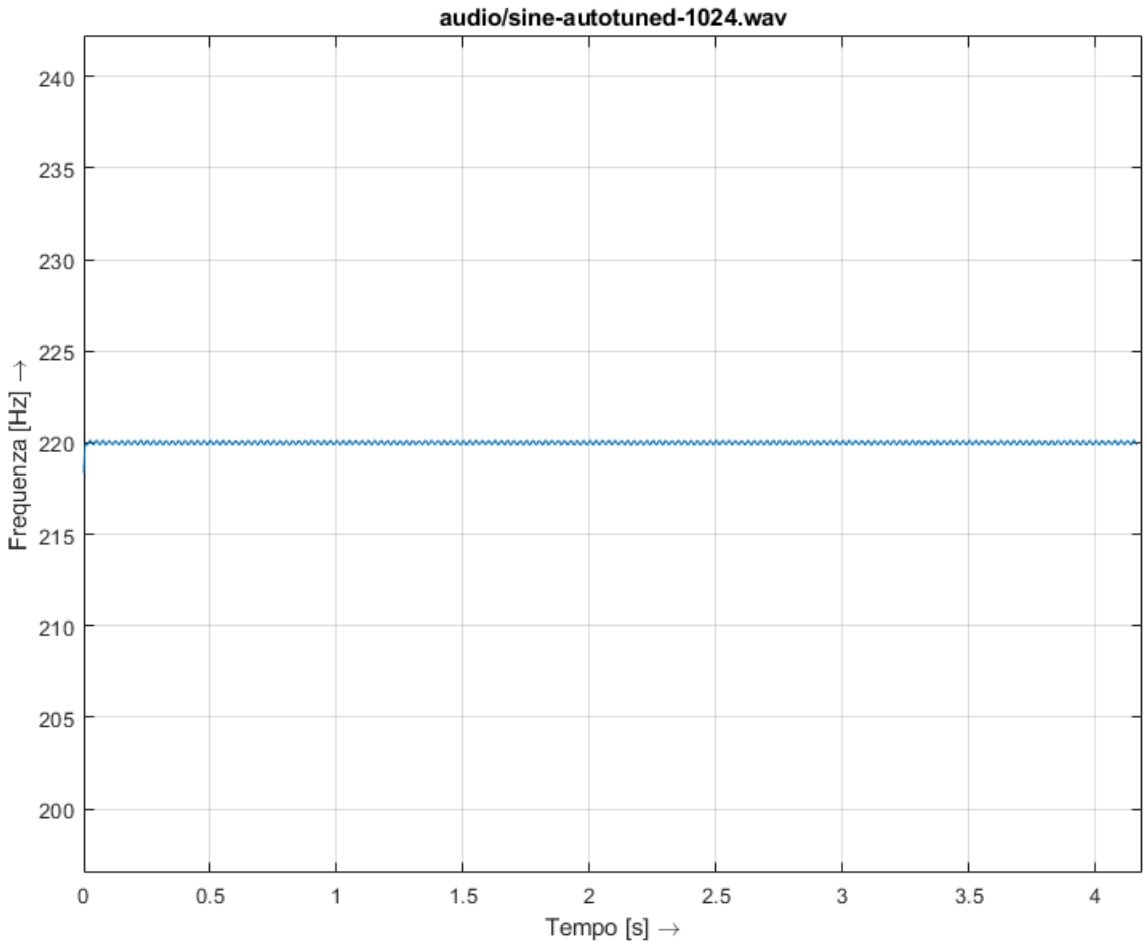

Figura 3.2: La sinusoide "intonata" a 220 Hz.

3.1.2 Prova effettuata su segnale di sine sweep

Il segnale di sine sweep, riportato in figura 3.3 viene generato numericamente all'interno di MATLAB impostando come parametri una frequenza di partenza $f_1 = 200$ Hz e una di arrivo $f_2 = 2000$ Hz. Il suo andamento è riportato graficamente in figura 3.3.

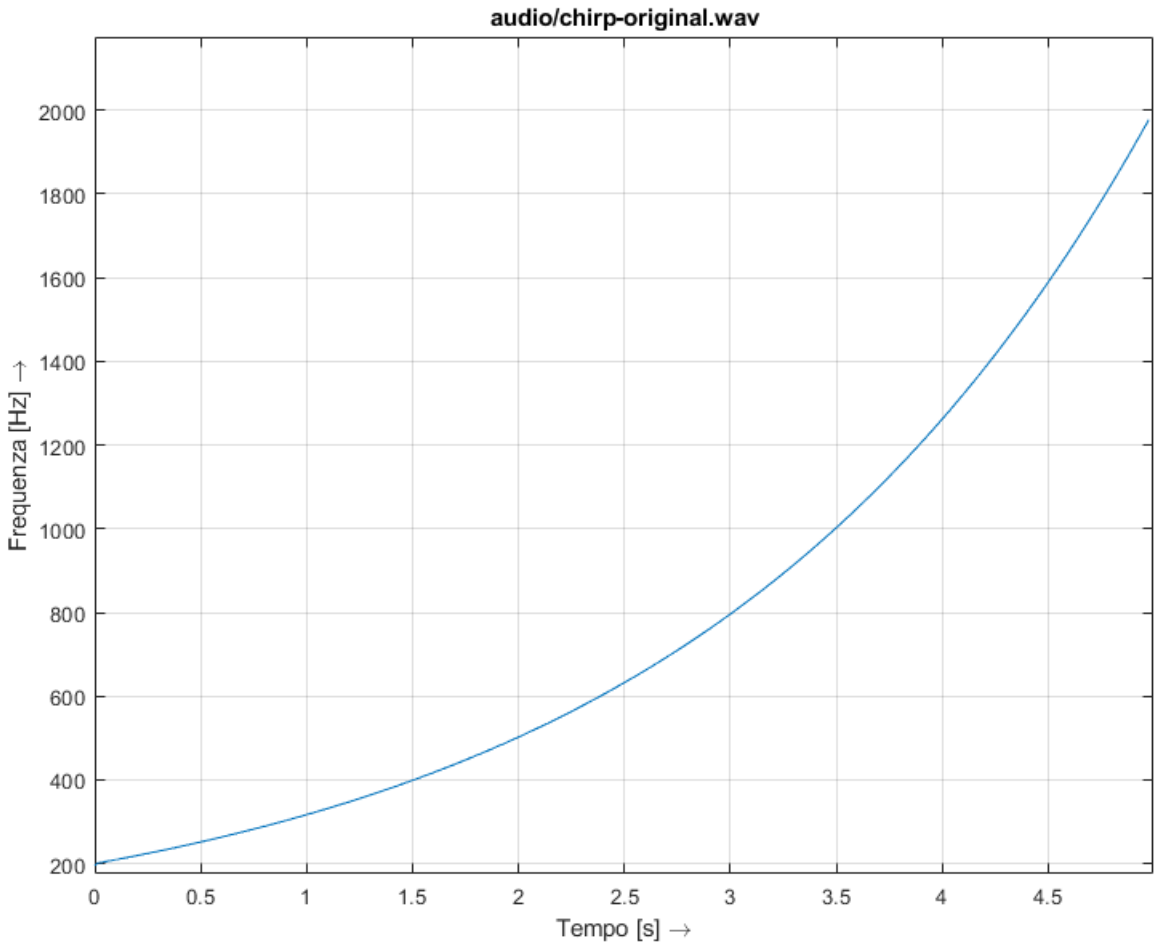

Figura 3.3: Il segnale di sine sweep originale.

L'andamento del segnale elaborato, riportato in figura 3.4, è quello di una scala cromatica composta da note di durata costante, a causa dell'andamento del segnale di sine sweep e della natura esponenziale della scala.

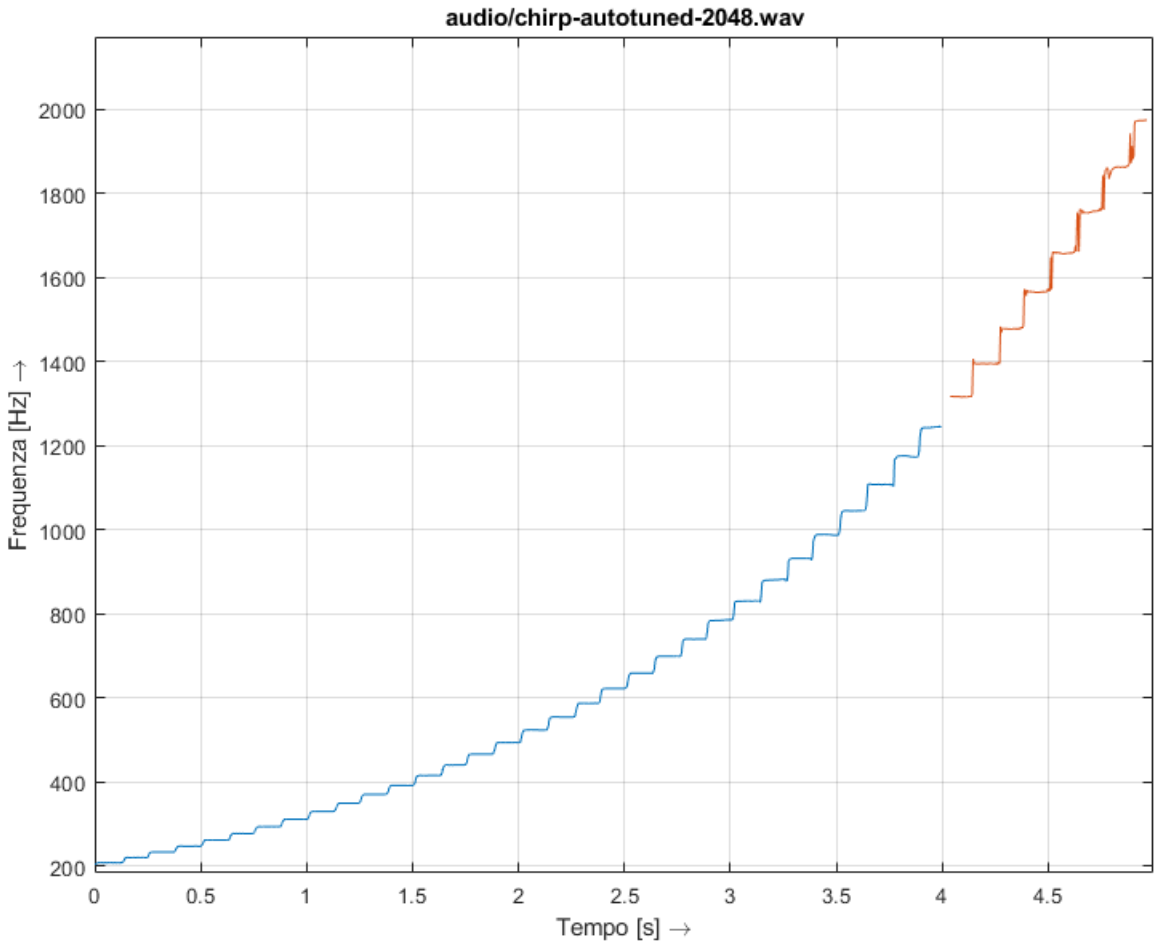

Figura 3.4: Il segnale di sine sweep "gradinato" in una scala cromatica.

3.1.3 Prova effettuata su segnale vocale

La prova effettuata su *Tom's Diner* rappresenta un esempio di utilizzo reale, ed è quella che risponde maggiormente alle differenze sulla scelta di N. La prova è stata effettuata utilizzando come riferimento nelle impostazioni dell'algoritmo, la scala di **Mi maggiore** (*E major*), tonalità della canzone, per garantire una correzione del tono più accurata.
Nel caso del segnale originale, riportato in figura 3.5, l'algoritmo fa difficoltà a tenere traccia della frequenza fondamentale; è comunque possibile localizzare le zone in corrispondenza delle note, ignorando i tratti di transizione da una nota all'altra, in cui la stima della frequenza oscilla fortemente.

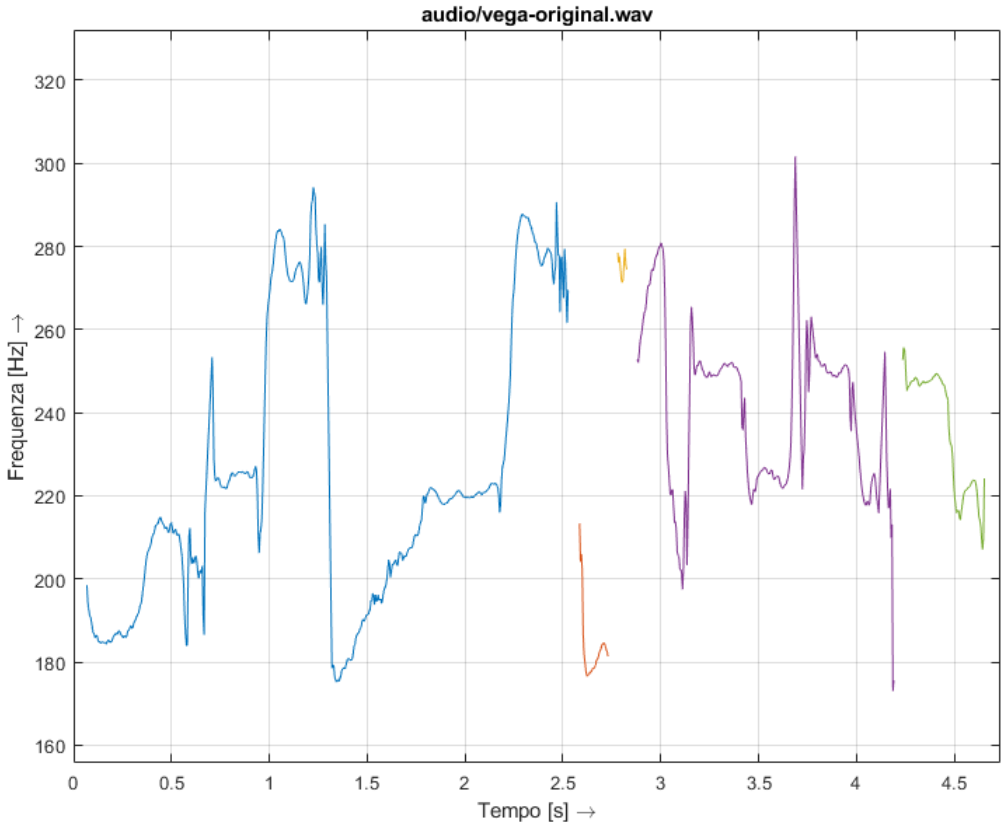

Figura 3.5: *Tom's Diner*, versione originale.

La versione della canzone elaborata dall'algoritmo con $N = 512$, riportata in figura 3.6, presenta un andamento visibilmente diverso rispetto all'originale: seppure siano presenti i medesimi picchi "spuri", il tracciamento della frequenza fondamentale risulta molto più stabile ad eccezione delle transizioni da una nota all'altra, segno che l'algoritmo di correzione del tono è stato effettivamente in grado di correggere la tonalità delle note in accordo con la scala utilizzata. Alla prova d'ascolto questa versione risulta essere quella di qualità più elevata tra le tre.

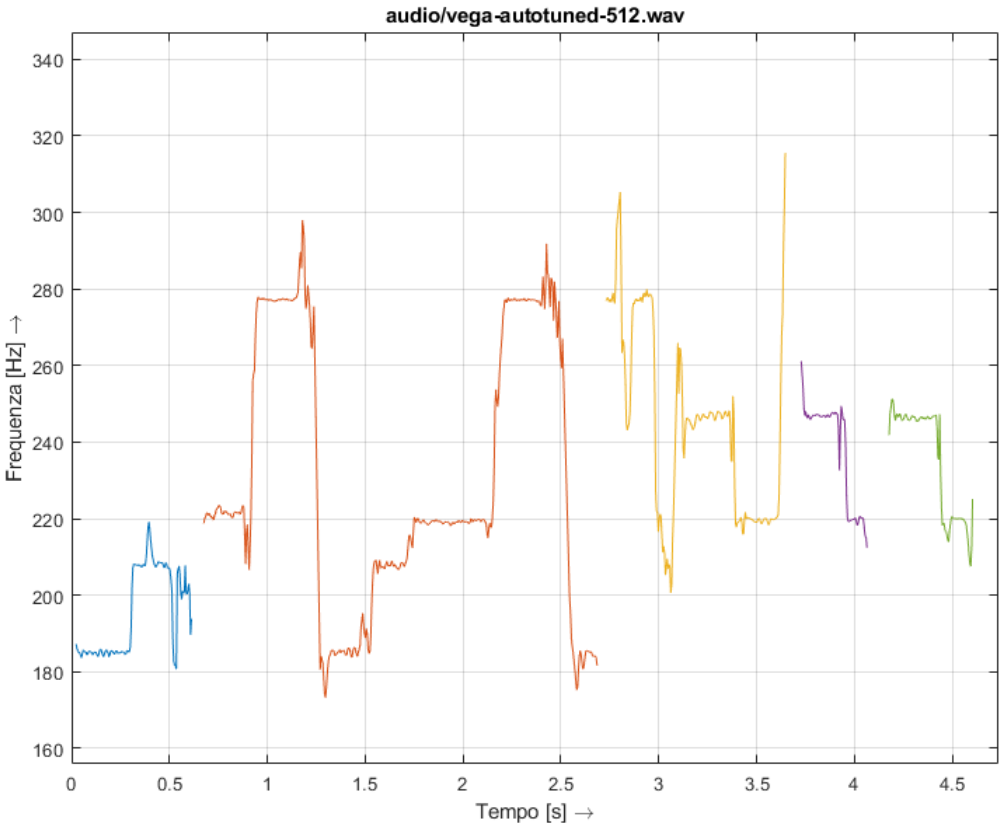

Figura 3.6: *Tom's Diner*, versione con correzione del tono con $N = 512$.

La versione con $N = 1024$, riportata in figura 3.7, presenta alla prova d'ascolto un livello qualitativo solo leggermente inferiore rispetto a quello della versione con $N = 512$. Le differenze con la versione precedente sono minime, seppure apparentemente l'algoritmo di tracciamento della frequenza fondamentale riveli una minore presenza di oscillazioni nella transizione tra le note.

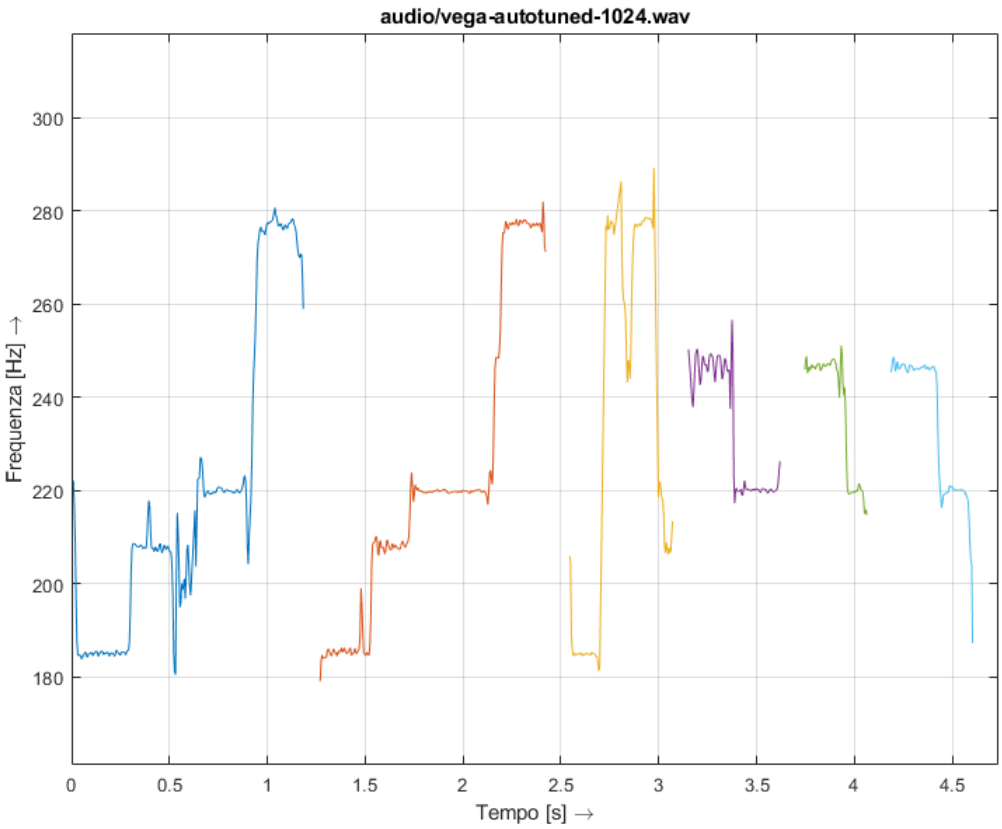

Figura 3.7: *Tom's Diner*, versione con correzione del tono con $N = 1024$.

La versione con $N = 2048$ è rappresentata in figura 3.8 e riporta la peggiore qualità audio, sebbene graficamente l'algoritmo di tracciamento sia riuscito a localizzare la frequenza fondamentale con meno discontinuità rispetto agli altri due casi. Nello specifico, all'ascolto è possibile notare la presenza di "oscillazioni" indesiderate nella voce (una sorta di effetto di vibrato ad alta frequenza) che ne rovinano la qualità generale. Il motivo della scarsa resa è molto probabilmente spiegato dall'eccessivo valore di $R = 512$. Se la distanza tra due segmenti è eccessiva, infatti, è possibile che la correzione del tono effettuata non tenga il passo con la variazione naturale del tono della voce della cantante, risultando quindi in un'oscillazione udibile dal momento che la correzione viene ri-calibrata solo dopo R campioni.

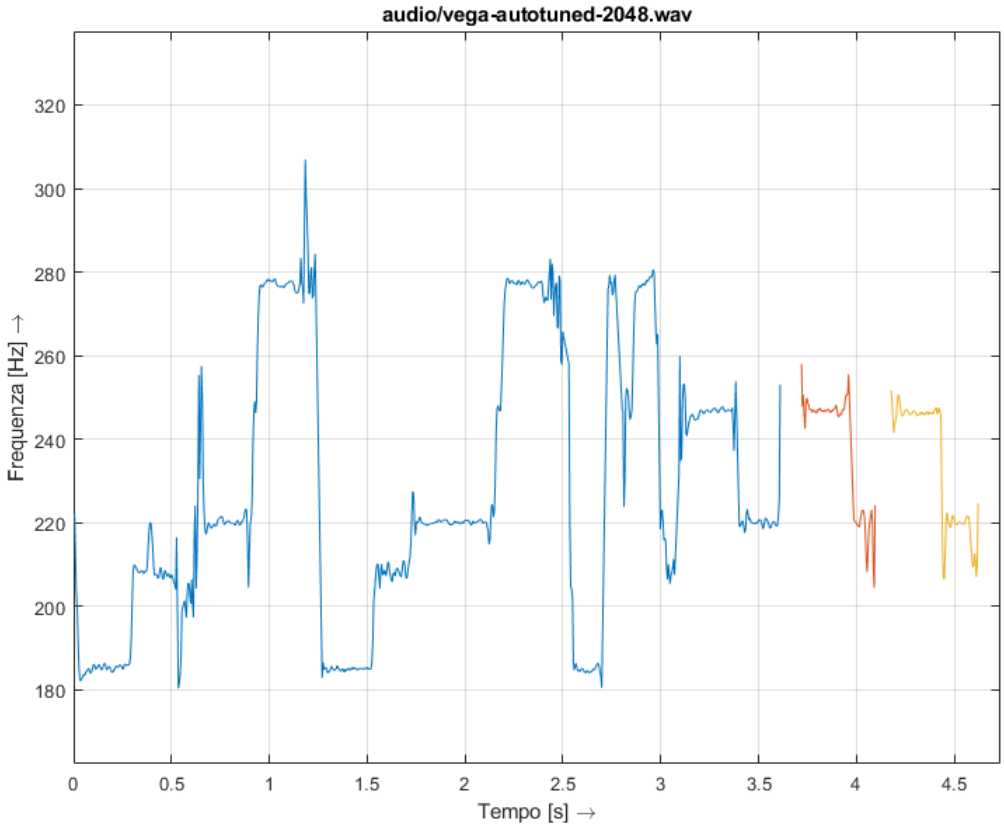

Figura 3.8: *Tom's Diner*, versione con correzione del tono con $N = 2048$.

3.2 Funzionamento in tempo reale del sistema

Le verifiche riportate sono stati effettuati sull'effetto qualitativo della funzione di correzione del tono descritta nel paragrafo 2.1.
È stato tuttavia effettuato un'ulteriore prova riguardante i tempi di calcolo dell'algoritmo e la latenza nel caso di utilizzo in tempo reale.
L'ambiente di sviluppo MATLAB è dotato di funzionalità integrate per la misurazione dei tempi di calcolo degli algoritmi sviluppati: effettuando un'analisi su questi tempi è risultato che il tempo di elaborazione totale sull'audio di *Tom's Diner* con $N = 1024$ è di $0.460s$.
Il tempo di calcolo impiegato solamente dalla sezione di elaborazione dei segmenti audio (escludendo quindi le operazioni preliminari di impostazione delle variabili e di scrittura su file) è di 0.325 secondi.
Dal momento che questa durata corrisponde al tempo impiegato da 1636 cicli (cioè $a = 1636$) in cui la funzione di correzione del tono è stata richiamata, il tempo medio per la correzione del tono di un singolo segmento di 1024 campioni è di circa 0.2ms, un tempo di latenza molto basso nel campo audio, seppure siano da considerare le prestazioni del calcolatore utilizzato per le misurazioni (CPU i5-3570k @ 3.4 GHz con 8 GB di RAM DDR3).

Nell'utilizzo dell'algoritmo in tempo reale, dunque, la maggiore latenza è quella procurata dall'acquisizione di N campioni. Nel caso di una frequenza di campionamento di $F_c = 48$ kHz, l'acquisizione di $N = 512$ campioni comporta una latenza di circa $\frac{F_c}{N} \simeq$ 10ms, quindi considerando 4 chiamate dell'algoritmo che avvengono prima di ottenere in uscita i primi R campioni acquisiti, la latenza totale è di circa 11ms.
La percezione della latenza in musica diventa oggettivamente evidente al di sopra dei 15ms, dunque l'unico modo di utilizzare un valore di N maggiore di 512 è quello di alzare la frequenza di campionamento F_c. Dalle osservazioni tratte nella prova con *Tom's Diner* del nostro algoritmo, tuttavia, si può ipotizzare che non sia qualitativamente desiderabile raggiungere un valore superiore a $N = 512$.

3.3 Obiettivi futuri

Il sistema di correzione del tono realizzato, seppure adempia allo scopo proposto, consiste, similmente ad altre soluzioni presenti nel mercato musicale, in un effetto che non si limita a correggere il tono vocale ma vi imprime anche un caratteristico timbro dalle sonorità artificiali, seppure molto differente dall'effetto del vocoder tradizionale. Questo è dovuto principalmente a due fattori:

- La correzione del tono in questo algoritmo avviene *"istantaneamente"*, o meglio, nell'arco di due soli segmenti a distanza R. Per questo motivo qualunque naturale effetto di **legato** o di **vibrato** viene cancellato similmente a quanto avvenuto nella prova sul segnale di *sine sweep*.

- Come riportato in [14], l'utilizzo del phase vocoder per *pitch shifting* e *time stretching* comporta l'insorgere di artefatti caratteristici legati alla rotazione di fase effettuata nella modifica. Sempre in [14] gli autori proseguono proponendo il metodo del *Peak phase locking* per ovviare a questo problema. Lo stesso metodo viene descritto dagli stessi autori anche in [10], proponendo un'alternativa al tradizionale metodo di spostamento del tono adottato in questo algoritmo.

Seppure l'algoritmo sia già funzionante ed utilizzabile, tra gli obiettivi futuri rientrano quindi un' del sistema di *phase locking*, che comporta l'utilizzo di un diverso metodo di individuazione delle frequenze e conseguentemente di un diverso metodo di rotazione di fase, e l'introduzione di un parametro che permetta di regolare la velocità con cui l'algoritmo corregge il tono nel tempo.
Una volta perfezionato il sistema, sarà possibile eventualmente la realizzazione di un software, ad esempio un **plugin VST** (vedi [13]) basato su di esso, o anche la realizzazione di un dispositivo *hardware embedded* in grado di replicare l'effetto nell'ambito di spettacoli dal vivo.

Conclusioni

In questo documento è stato riportato il processo di realizzazione di un sistema di correzione del tono, a partire dalle sue basi concettuali, fino alla sua programmazione, prova e valutazione.
Il sistema è stato utilizzato per la correzione del tono su segnali di prova per verificarne il funzionamento corretto e le ottimali impostazioni dei parametri. Il sistema è stato inoltre provato utilizzando come sorgente il flusso di campioni audio ottenuti da un scheda di acquisizione in tempo reale collegata ad un microfono, e ne sono stati valutati i tempi di latenza media. L'impiego del sistema di correzione in tempo reale è risultato possibile, e dal risultato equivalente a quello ottenuto in post-elaborazione su brani registrati.
In generale, il sistema di correzione del tono è risultato funzionante se utilizzato in accordo con le prerogative enunciate inizialmente nell'introduzione e dal risultato qualitativo estremamente simile alle soluzioni commerciali, presentando tuttavia le problematiche riportate nel paragrafo 3.3. Nello stesso paragrafo sono stati descritti dei possibili approcci per la correzione di queste problematiche e lo sviluppo futuro di una versione migliorata di questo algoritmo di qualità superiore rispetto a quella attuale.

Bibliografia

[1] D. Arfib, F. Keiler, U. Zölzer, V. Verfaille, J. Bonada. *DAFX: Digital Audio Effects*, second edition. U. Zölzer, UK: John Wiley & Sons, 2011.

[2] J.L. Flanagan and R.M. Golden. “Phase vocoder,” *Bell Syst. Tech. J.*, vol. 45, pp. 1493–1509, 1966.

[3] M. R. Portnoff. "Implementation of the Digital Phase Vocoder Using The Fast Fourier Transform" in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, p. 243, 1976.

[4] R. Hoffmann. "On the development of early vocoders" in *Second IEEE Region 8 Conference on the History of Telecommunications Conference (HISTELCON)*, 2010.

[5] Homer Dudley. "The carrier nature of speech" in *The Bell System technical journal"*, vol. 19, October 1940.

[6] J. O. Smith III. "Overlap-Add (OLA) STFT Processing" in *Spectral Audio Signal Processing*, http://www.dsprelated.com/freebooks/sasp/, 2011, accessed 20/11/2017.

[7] R. Crochiere. "A weighted overlap-add method of short-time Fourier analysis/-Synthesis" in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, pp. 99-102, 1980.

[8] G. Heinzel, A. Rüdiger and R. Schilling. *Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.455.7976&rep=rep1&type=pdf, 2002, accessed 1/12/2017.

[9] A. v. d. Knesebeck, U. Zölzer. "Comparison of pitch trackers for real-time guitar effects" in *Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx-10)*, 2010.

[10] J. Laroche, M. Dolson. "New Phase Vocoder techniques for Pitch-shifting, harmonizing and other exotic effects" in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 91-94, 1999.

[11] A. De Götzen, N. Bernadini, D. Arfib. "Traditional (?) implementations of a phase vocoder: The tricks of the trade." in *Proc. DAFX-00 Conference on Digital Audio Effects*, 2000.

[12] E. Joliveau, J. Smith, J. Wolfe. "Vocal tract resonances in singing: the soprano voice." in *The Journal of the Acoustical Society of America*, 2004.

[13] Steinberg Media Technologies GmbH. "Technologies", https://www.steinberg.net/en/company/technologies.html, 2017, accessed 4/12/2017.

[14] J. Laroche, M. Dolson, "About this phasiness business" in *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*, 1997.

Ringraziamenti

È doveroso dedicare questa sezione a chi mi è stato di grande aiuto in questo percorso impegnativo.

Ringrazio i miei genitori per il loro grande affetto, ma soprattutto per avermi formato, insegnato a pormi delle domande ed avermi supportato in ogni mia scelta. Ringrazio mia sorella Alessandra per essere stata sempre la mia sicurezza in ogni momento di difficoltà ed aver contribuito a rendermi una persona migliore.

Ringrazio gli amici con i quali ho legato nel corso del tempo ed i compagni di studi, gioie e dolori: senza di loro questo traguardo non sarebbe stato possibile. Il loro costante supporto e la loro fiducia nelle mie capacità sono componenti fondamentali per permettere la riuscita dei miei progetti.

Ringrazio inoltre tutti i docenti, in particolare il prof. Raheli, relatore di questa tesi, per la fiducia riposta nel mio operato, per il supporto fornitomi nella realizzazione di questo progetto e per avermi spinto a indagare a fondo e conoscere nel dettaglio gli argomenti presi in esame.

A tutti coloro che nel tempo mi hanno sostenuto ed insegnato qualcosa di nuovo e che tuttora continuano a farlo, io rivolgo i miei più sentiti ringraziamenti.