UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Elettronica

Studio e sviluppo di algoritmi DSP per l'effettistica musicale su piattaforma embedded

Study and development of DSP algorithms for music effects on embedded platform

Relatore: Chiar.mo/a
Prof. **Stefano Squartini**

Tesi di Laurea di:
Francesco Ballarini

Correlatore:
Ing. **Leonardo Gabrielli**

A.A. **2019 / 2020**

ABSTRACT

La presente tesi ha come obiettivo lo studio e lo sviluppo di algoritmi di pitch shifting che possano essere implementati sulla piattaforma embedded “Audiolino Brick”. Gli algoritmi da ottenere devono rispettare la caratteristica hard real-time richiesta dall’utilizzo live della scheda.

Inizialmente si presenteranno gli algoritmi principalmente studiati e si mostrerà la loro implementazione nel software Octave. Successivamente si mostrerà come si è passati all’implementazione in linguaggio C degli algoritmi che permettono il loro utilizzo real-time sulla scheda.

Indice

CAPITOLO 1: INTRODUZIONE pag.5

1.1 Stato dell'arte pag.7

1.2 Percorso di sviluppo pag.8

CAPITOLO 2: PITCH SHIFTING BY DELAY-LINE MODULATION pag.9

2.1 Descrizione del metodo pag.9

2.2 Algoritmo Octave per il metodo a 2 delay-line pag.13

2.3 Algoritmo Octave per il metodo a 3 delay-line pag.23

CAPITOLO 3: PSOLA (PITCH-SYNCHRONOUS OVERLAP AND ADD) pag.29

3.1 Descrizione del metodo pag.29

3.2 Algoritmi Octave per il pitch detection e il pitch marking pag.33

3.3 Algoritmo Octave per l'elaborazione PSOLA pag.34

CAPITOLO 4: BLOCK-BY-BLOCK APPROACH (FFT/IFFT) pag.39

4.1 Descrizione del metodo pag.39

4.2 Algoritmo Octave per il metodo block-by-block (FFT/IFFT) pag.42

CAPITOLO 5: IMPLEMENTAZIONE IN LINGUAGGIO C pag.47

5.1 Scelta degli algoritmi da implementare pag.47

5.2 Algoritmo in linguaggio C per il metodo a 2 delay-line pag.48

5.3 Algoritmo in linguaggio C per il metodo block-by-block pag.51

5.4 Audiolino board pag.52

CAPITOLO 6: CONCLUSIONI pag.55

BIBLIOGRAFIA pag.56

.

Capitolo 1

Introduzione

Un suono può essere descritto in base ad una serie di attributi percettivi che lo caratterizzano e che permettono di classificare sia i suoni sia gli effetti audio in varie categorie. Le caratteristiche principali che descrivono un suono sono [1] [2]:

- L'intensità o loudness è l'attributo che permette di differenziare i suoni dal debole al forte in base alla pressione acustica generata dalle vibrazioni della sorgente sonora. Ha una relazione diretta con la dinamica del suono, quindi con i fraseggi (legato e pizzicato), le sfumature, gli accenti, il tremolo.
- Le caratteristiche temporali quali durata, tempo, ritmica.
- Gli aspetti inerenti allo spatial hearing, cioè localizzazione della sorgente, moto relativo, direttività e room effect.
- Pitch o altezza dei suoni armonici che dipende principalmente dalla frequenza e dall'ampiezza delle armoniche del suono, ma la sua percezione è influenzata anche dal timbro. Corrisponde, dal punto di vista fisico, alla frequenza fondamentale. Un suono musicale può essere perfettamente armonico (Es. strumenti a fiato), quasi armonico (Es. strumenti a corde) o inarmonico.
- Il timbro è la caratteristica più difficile da descrivere scientificamente, ma qualitativamente è quell'attributo mediante il quale è possibile distinguere suoni diversi anche quando sono di pari altezza. Dipende da caratteristiche a breve termine, come attacco e transienti, e da caratteristiche a lungo termine, come formanti e brillantezza.

Le tecniche di pitch shifting, come si può intuire dal nome, vanno a modificare principalmente il pitch del suono sottoposto all'elaborazione. A seconda del metodo utilizzato possono essere introdotte altre modifiche più o meno accentuate ad altre caratteristiche del suono, che possono essere sia effetti voluti che effetti indesiderati. Le tecniche che alterano il pitch si basano su algoritmi simili, ma adattati alle varie esigenze. Qui sotto è presente una tabella riassuntiva delle tecniche che hanno effetto principalmente sul pitch.

Effetto	Attributi percettivi principalmente modificati	Altri attributi percettivi modificati
Pitch shifting senza conservazione delle formanti	Pitch	Timbro
Pitch shifting con conservazione delle formanti	Pitch	
Pitch change	Pitch	
Pitch discretization (auto-tune)	Pitch	Timbro
Harmonizer	Pitch	
Inharmonizer	Pitch	

Tabella 1: Effetti digitali con obiettivo la modifica del pitch

1.1 Stato dell'arte

La trasposizione per un musicista è uno strumento base che consiste nell'eseguire una melodia in una differente tonalità, cioè, ragionando in termini di processamento del segnale, eseguire una melodia dopo aver effettuato un pitch shifting di un intervallo fisso. Ogni volta che effettua una trasposizione usa un diverso registro dello strumento, dunque non è solo il pitch a cambiare, ma anche il timbro. Nell'ambito degli effetti digitali quando si crea un algoritmo per il pitch shift bisogna scegliere se effettuare la trasposizione senza preoccuparsi di mantenere il timbro originale o se mantenere il timbro dello strumento invariato in tutti i registri. Nel primo caso si parla di "variable timbre transposition", nel secondo di "constant timbre transposition".
L'esempio più chiaro per capire il concetto di modifica o mantenimento del timbro è la voce. L'emissione di un suono vocale può essere suddivisa in due parti:

- Il pitch del segnale vocale è determinato dalla frequenza di vibrazione delle corde vocali e da questa frequenza dipendono le armoniche presenti nello spettro del segnale. Dalle corde vocali deriva la parte di eccitazione nel segnale vocale.
- Il timbro è determinato invece dall'apparato fonatorio caratteristico del cantante. L'effetto delle cavità fonatorie è quello di enfatizzare alcune parti dello spettro che prendono il nome di formanti. Dalle formanti derivano le risonanze del segnale vocale. L'effetto delle formanti è quello di un filtro che viene applicato alla parte di eccitazione.

Quando un cantante effettua una trasposizione può modificare le formanti in modo indipendente dal pitch. Modificando solamente le costituenti armoniche del suono si ha un'alterazione del pitch con mantenimento del timbro, modificando solamente le risonanze si ha un'alterazione del pitch mantenendo il timbro.
Dei metodi studiati in questa trattazione due realizzano un pitch shifting senza conservazione delle formanti mentre uno realizza un pitch shifting con conservazione delle formanti.

1.2 Percorso di sviluppo

Lo studio dei metodi da implementare, dopo aver analizzato a grandi linee varie tecniche di pitch shifting, si è concentrato su tre algoritmi:

- Pitch shifting by delay-line modulation;
- PSOLA (Pitch-Synchronous OverLap and Add);
- Block-by-block approach (FFT/IFFT);

Per prima cosa si è realizzato un algoritmo tramite il software Octave per studiare pregi e difetti dei risultati ottenibili con ogni algoritmo. Tenendo conto delle specifiche richieste dall'uso real-time sulla scheda Audiolino Brick, prima su tutte la necessità di un'elaborazione hard real-time e poi la capacità di funzionamento con più fonti sonore possibili, sono stati scelti gli algoritmi che rispondevano meglio alle esigenze e si è passati all'implementazione in linguaggio C, in previsione dell'uso sulla scheda.

Nei prossimi capitoli si vedranno:

- Capitolo 2: descrizione del metodo basato sul pitch shifting by delay-line modulation e illustrazione dei risultati ottenuti tramite gli algoritmi implementati sul software Octave;
- Capitolo 3: descrizione del metodo basato sul Pitch-Synchronous OverLap and Add e illustrazione dei risultati ottenuti tramite gli algoritmi implementati sul software Octave;
- Capitolo 4: descrizione del metodo basato sul block-by-block approach (FFT/IFFT) e illustrazione dei risultati ottenuti tramite gli algoritmi implementati sul software Octave;
- Capitolo 5: Risultati ottenuti dall'implementazione in linguaggio C degli algoritmi scelti e illustrazione delle caratteristiche della scheda Audiolino Brick;
- Capitolo 6: conclusioni sui risultati ottenuti;

Capitolo 2

Pitch shifting by delay-line modulation

2.1 Descrizione del metodo

La possibilità di definire una linea di ritardo modulata in un sistema a tempo discreto appare, formalmente, piuttosto semplice: è infatti sufficiente regolare il ritardo con una funzione di modulazione. Un possibile schema realizzativo, come mostrato in figura 2.1, sfrutta una coda circolare di n campioni dove la testa è individuata dall'M-esimo campione, mentre l'uscita è posta in corrispondenza della locazione k-esima [3].

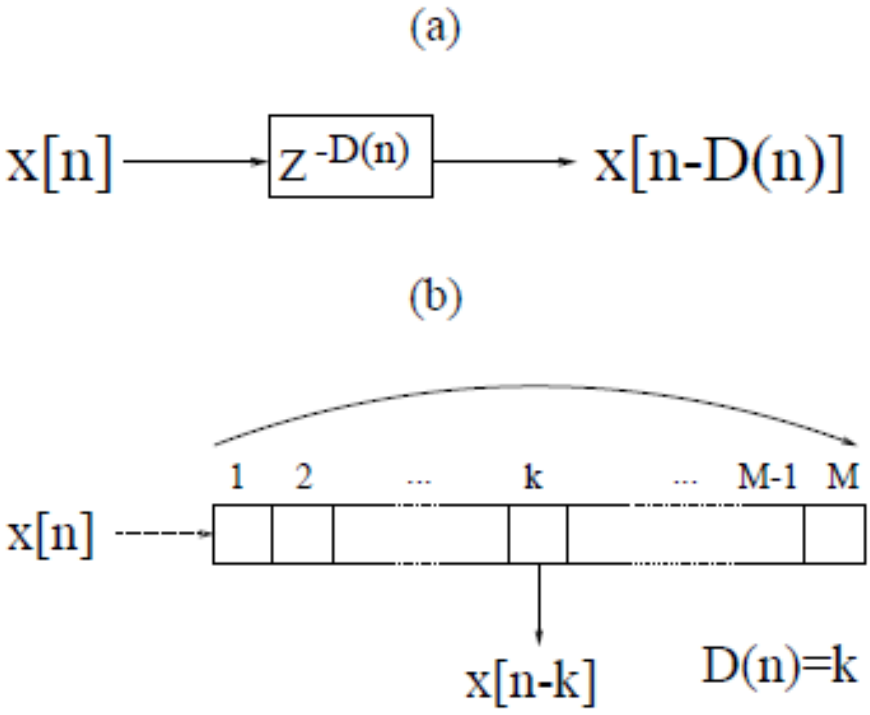


Figura 2.1 Linea di ritardo modulata (a) e sua realizzazione mediante coda circolare (b).

Poiché il valore del ritardo varia in modo continuo l'uscita desiderata si troverà a cavallo tra due campioni. Chiamando i.frac il valore associato all'uscita desiderata lungo la delay-line all'istante n-esimo, esso sarà costituito da una parte intera "i" e una frazionaria "frac". Risulta dunque necessaria un'operazione di interpolazione fra i due campioni di indice i e i+1. Tra le varie tecniche di interpolazione negli algoritmi che implementano le delay-line modulate è stata scelta l'interpolazione lineare come in figura 2.2.

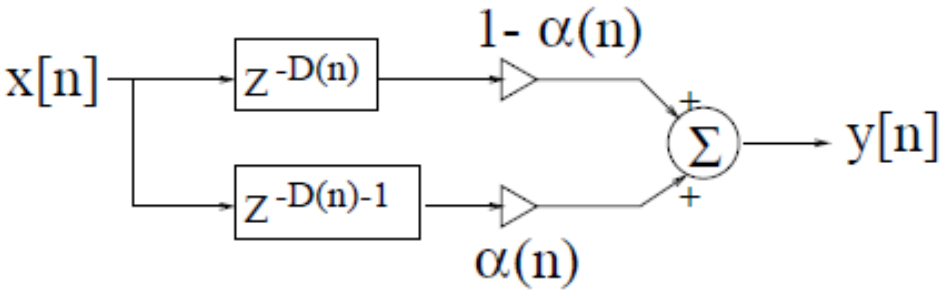


Figura 2.2 Sistema costituito da linea di ritardo modulata e interpolatore lineare.

Assumendo un segnale sinusoidale arbitrario di ampiezza costante A, fase ϕ e con Ω = 2πf, se i.frac varia linearmente vale:

$$x((n\text{-}i.frac)T) = A\cos[\Omega(n\text{-}i.frac)T + \phi] \qquad (2.1)$$

$$i.frac = NOMINAL\_DELAY + (1\text{-} \text{ pitch change ratio})n \qquad (2.2)$$

NOMINAL_DELAY è una costante espressa in un numero intero di campioni e rappresenta l'offset del puntatore. Per n=0 i.frac punta a NOMINAL_DELAY. Anche "pitch change ratio" è assunto costante. È possibile calcolare la frequenza istantanea come *(2.3)*:

$$\Omega_i = (\partial[\Omega(-NOMINAL\_DELAY + \text{pitch change ratio}*n)T+\phi])/(T\partial n)$$
$$= \Omega*\text{pitch change ratio} \qquad (2.3)$$

Si ottiene dunque un pitch shift tramite la variazione lineare di i.frac. Questa tecnica presenta però un problema insormontabile, infatti è limitata alle dimensioni massime della linea di ritardo e non può essere usata indefinitamente [4].

Per superare questo problema è possibile dividere in blocchi il segnale ed utilizzare due diverse delay-line variabili [5] con un ritardo tra esse equiva-

lente a mezza lunghezza del blocco. Per eliminare i problemi della discontinuità ed ottenere un segnale continuo le due linee vengono combinate da un blocco di cross-fading che genera il segnale di uscita, come mostrato in figura 2.3. Per ridurre il carico computazionale nel codice Octave sono state usate due finestre di Hanning. Il ritardo risulta modulato da una serie di rampe, dunque da un dente di sega, che oscilla tra 0 ed un massimo che fornisce, fissata la lunghezza dei frame, il parametro da modificare per variare l'altezza del suono. Il valore massimo dell'ampiezza del dente di sega è dato dalla lunghezza della delay-line.

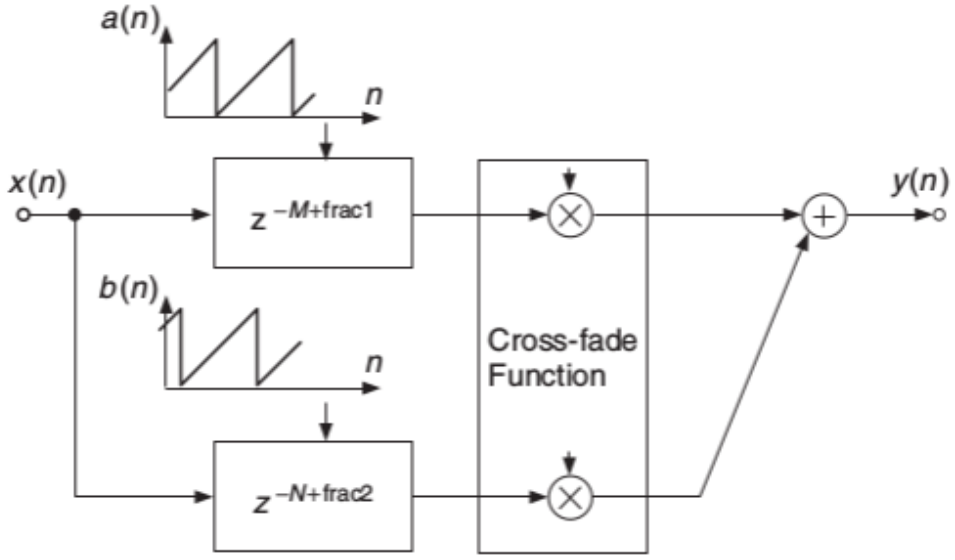


Figure 6.13 Pitch shifting.

Figura 2.3 Pitch shifting tramite il metodo a due delay-line.

Una diversa implementazione, mostrata in figura 2.4, del metodo tramite delay-line modulate prevede l'utilizzo di tre delay-line sfasate tra di esse di un terzo della lunghezza di un blocco e i cui segnali vengono poi modulati in ampiezza tramite segnali sinusoidali, anch'essi tra loro sfasati di 120°, in modo da minimizzare il rumore prodotto dalle discontinuità, e sommati per riottenere l'uscita. Gli artefatti percepiti sono simili all'effetto di phasing e sono meno fastidiosi rispetto alle discontinuità locali delle applicazioni basate su metodi con due linee [2][6].

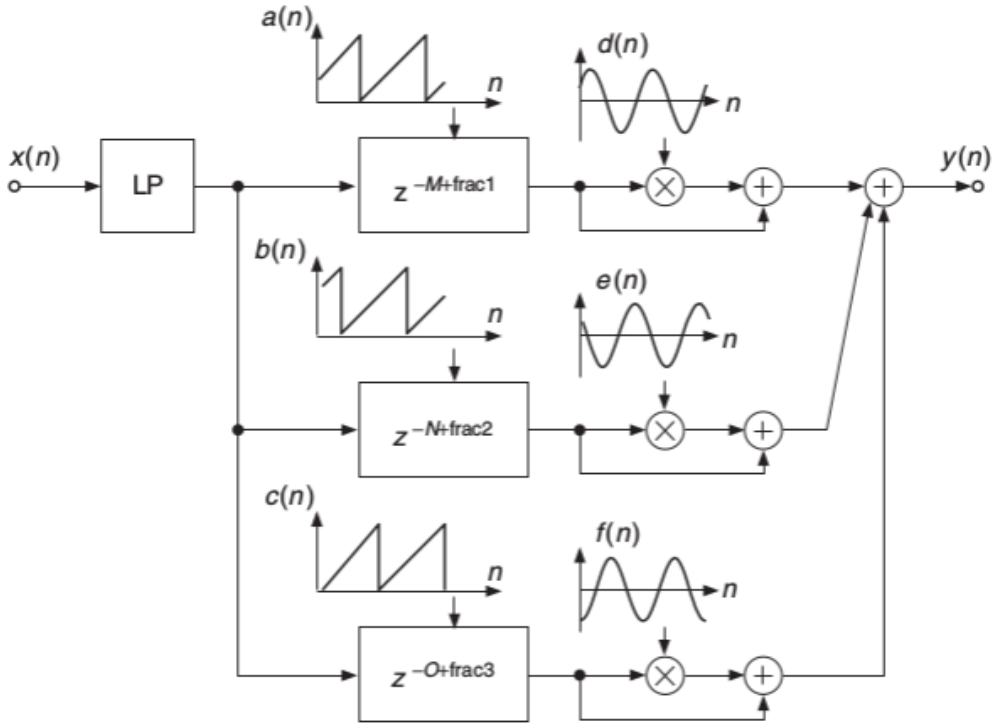


Figura 2.4 Pitch shifting tramite il metodo a tre delay-line.

L'implementazione in Octave è stata effettuata per entrambe le tecniche [7] e i test hanno mostrato che la qualità massima del metodo con tre linee è effettivamente superiore all'altra, ma la gestione delle linee è più problematica e richiede blocchi di lunghezza maggiore per ottenere un risultato qualitativamente sufficiente. Inoltre poiché bisogna elaborare e interpolare tre linee diverse è richiesto uno sforzo computazionale maggiore.

2.2 Algoritmo Octave per il metodo a 2 delay-line

L'algoritmo prevede una prima parte di creazione delle finestre e delle funzioni utilizzate come coefficienti per l'interpolazione dei campioni di ogni delay-line e per il cross-fading fra le due diverse linee [7]. I parametri che influenzano le prestazioni sia per qualità, sia per tempo di calcolo e latenza sono la lunghezza delle rampe e la quantità che indica lo shift. Quest'ultima corrisponde al numero di semitoni (positiva per shift verso frequenze maggiori, negativa per shift verso frequenze minori) di cui il segnale viene trasposto. Il numero di semitoni è per prima cosa convertito nel fattore di trasposizione "pitch change ratio", che rappresenta il rapporto fra la frequenza generata in uscita e quella originale del segnale di ingresso, tramite il rapporto che esiste fra le frequenze delle note musicali, cioè:

$$pitch\ change\ ratio = 2^{n\_semitoni/12} \qquad (2.4)$$

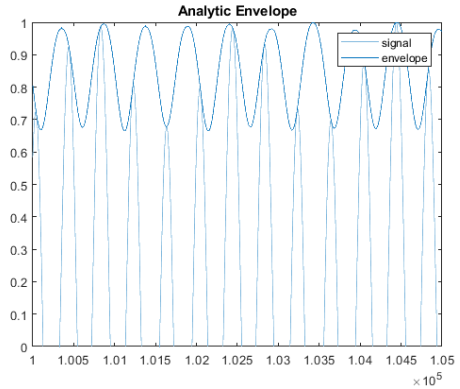
Dal valore del pitch change ratio si costruiscono le rampe che modulano la delay-line. Le finestre di cross-fading sono due finestre di Hanning lunghe come le rampe.
L'algoritmo, per come è stato implementato, risulta limitato nello shift verso frequenze superiori ad un massimo di un'ottava, poiché la delay-line è lunga quanto una finestra di segnale. Se chiamiamo con LEN la lunghezza della delay-line (pari alla lunghezza di un blocco di segnale) e con PCR il valore di pitch change ratio, è facilmente verificabile che il valore della variabile WIDTH che sta a rappresentare il massimo ritardo della delay-line vale:

$$WIDTH=(LEN-1)*|PCR-1| \qquad (2.5)$$

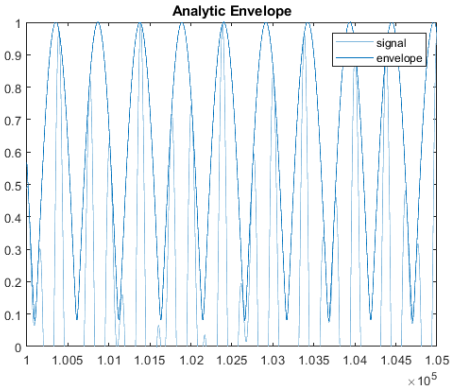
perché la funzione che rappresenta la rampa è semplicemente quella di un tratto di retta. Si può calcolare allora che per PCR ≥ 2 allora WIDTH è maggiore della lunghezza LEN della delay-line. Impostare un pitch change ratio tale che WIDTH sia maggiore di LEN significa superare l'indice massimo del vettore del segnale ritardato memorizzato nella delay-line, quindi significa andare a cercare un campione non più presente nella memoria.

Potenzialmente l'algoritmo può essere modificato per superare questo problema usando una delay-line con una lunghezza maggiore di quella della rampa o, specularmente, più rampe in uno stesso frame invece che una singola. Queste soluzioni permettono di avere una variazione maggiore, ma la qualità del risultato tende a degradare con grandi cambiamenti di pitch, per cui risulta sufficiente il limite di un'ottava ottenuto tramite l'implementazione più semplice, che permette una facilità di implementazione e gestione interna dell'algoritmo maggiore. Il risultato ottenuto è invece ottimo per variazioni di alcuni semitoni o toni e permette di avere un basso costo computazionale rispetto ai metodi più avanzati.

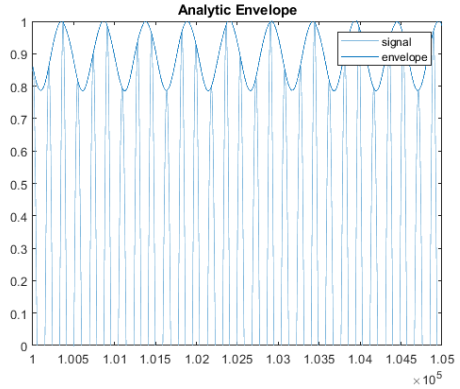
L'elaborazione di un campione è pressoché istantanea poiché si tratta di eseguire una somma e due moltiplicazioni per il cross-fading e una somma e due moltiplicazioni per l'interpolazione in ogni canale, quindi in totale si hanno tre somme e sei moltiplicazioni per ogni campione.
Nonostante l'elaborazione avvenga campione per campione è presente un tempo di latenza che può essere anche molto significativo a seconda di come si imposta inizialmente la lunghezza delle delay-line. La latenza deriva dalla necessità di andare ad interpolare campioni passati per ottenere in uscita il campione attuale. Poiché le delay-line sono sfasate della metà del massimo ritardo, in campioni sono sfasate quindi di WIDTH/2, si ha che la latenza sarà circa metà del massimo ritardo. Dunque, poiché come dimostra la formula che lega il valore del massimo ritardo WIDTH ai valori di LEN (lunghezza della rampa) e PCR (valore di pitch change ratio) il ritardo è proporzionale a questi due parametri, la latenza sarà maggiore tanto maggiori saranno LEN e PCR.
È stata riscontrata una perdita di qualità al diminuire della lunghezza delle finestre dovuta al continuo salto del segnale tra le linee di ritardo. Utilizzare linee diverse porta ad un altro problema, infatti con rampe troppo corte il valore del pitch change ratio desiderato non corrisponde a quello ottenuto in uscita. Con rampe corte l'algoritmo non modifica più il segnale come mostrato dal modello matematico precedente e quindi questo pone un limite inferiore alla lunghezza delle rampe e della delay-line. È imposto un trade-off tra qualità del segnale in uscita e latenza, quindi è necessario scegliere una lunghezza della rampa che possa garantire risultati accettabili in entrambi gli aspetti.
Essenzialmente l'algoritmo realizza per ogni campione in uscita il cross-fading tra segnali sfasati, per questo la qualità del risultato raggiunto varia non solo a seconda dello shift effettuato e della lunghezza di ogni rampa, ma anche dalla frequenza del segnale. Come mostrato dall'analisi nel tempo di vari segnali sinusoidali con diversa frequenza nella figura 2.5, a parità di pitch change ratio e di lunghezza delle rampe si nota facilmente l'effetto collaterale del cross-fading tra parti sfasate del segnale che provoca una modulazione di ampiezza che varia a seconda della frequenza del segnale.



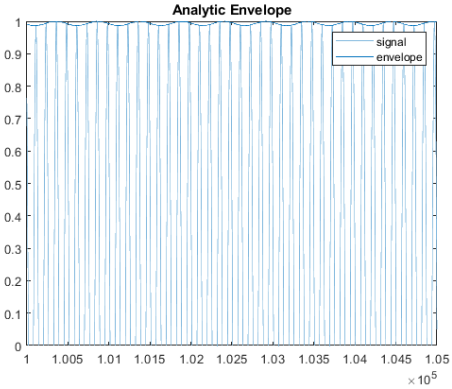
Sottofigura a) Segnale a 110Hz, PCR 0.794, LEN 1024



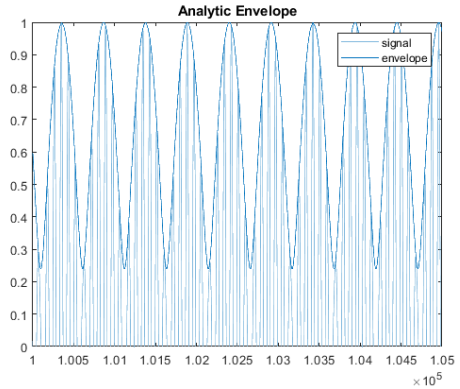
Sottofigura b) Segnale a 220Hz, PCR 0.794, LEN 1024



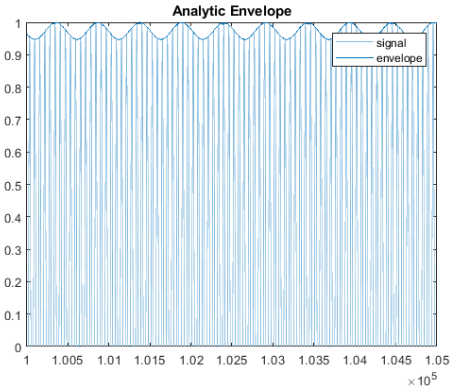
Sottofigura c) Segnale a 330Hz, PCR 0.794, LEN 1024



Sottofigura d) Segnale a 440Hz, PCR 0.794, LEN 1024



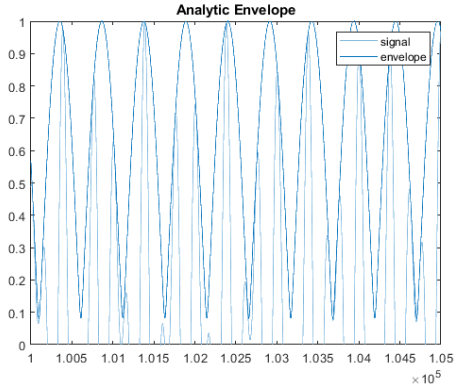
Sottofigura e) Segnale a 660Hz, PCR 0.794, LEN 1024



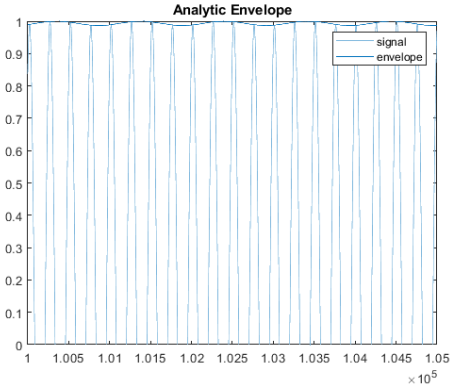
Sottofigura f) Segnale a 880Hz, PCR 0.794, LEN 1024

Figura 2.5 Porzione dei segnali di uscita dell'elaborazione di sinusoidi a varie frequenze con pitch change ratio di 0.794 e lunghezza delle rampe di 1024 campioni

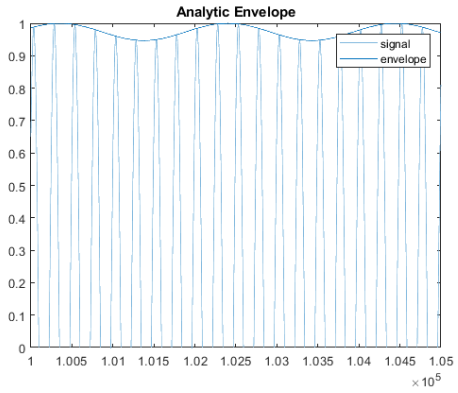
La modulazione di ampiezza varia anche se invece di variare la frequenza del segnale si varia la lunghezza delle rampe, poiché varia lo sfasamento tra le linee e dunque le porzioni di segnale che si sovrappongono (figura 2.6).



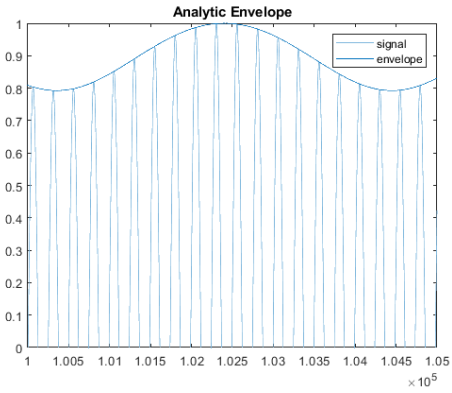
Sottofigura a) Segnale a 220Hz, PCR 0.794, LEN 1024



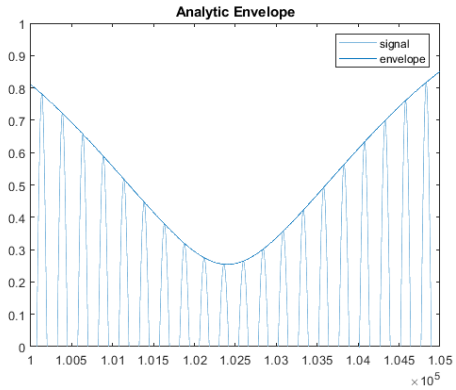
Sottofigura b) Segnale a 220Hz, PCR 0.794, LEN 2048



Sottofigura c) Segnale a 220Hz, PCR 0.794, LEN 4096



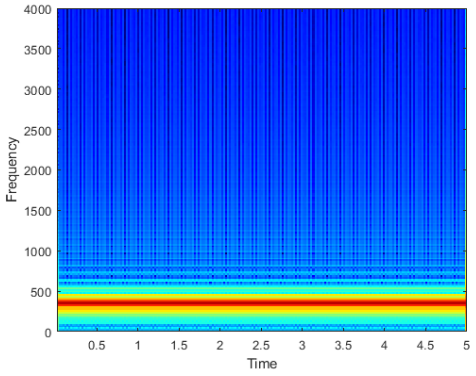
Sottofigura d) Segnale a 220Hz, PCR 0.794, LEN 8192



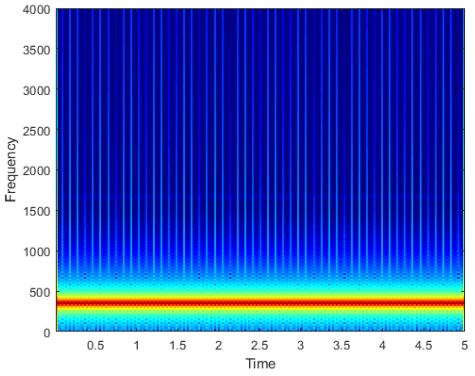
Sottofigura e) Segnale a 220Hz, PCR 0.794, LEN 16384

Figura 2.6 Porzione dei segnali di uscita dell'elaborazione di una sinusoide a frequenze 220Hz con pitch change ratio di 0.794 e lunghezza delle rampe variabile

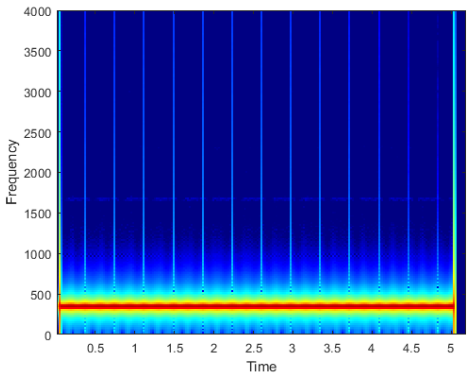
Dall’analisi in frequenza tramite lo spettrogramma si possono vedere i disturbi dovuti alle discontinuità provocate dai salti di delay-line. Come si vede dalla figura 2.7 le discontinuità si presentano periodicamente con periodo pari alla lunghezza delle rampe. Il disturbo si riduce quindi aumentando la lunghezza delle rampe.



Sottofigura a) Segnale a 440Hz, PCR 0.794, LEN 1024



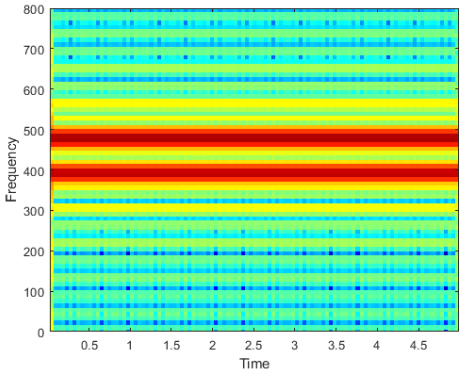
Sottofigura b) Segnale a 440Hz, PCR 0.794, LEN 4096



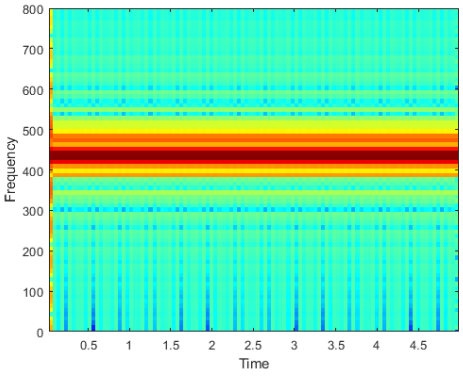
Sottofigura c) Segnale a 440Hz, PCR 0.794, LEN 16384

Figura 2.7 Spettrogrammi dei segnali di uscita dell’elaborazione di una sinusoide a frequenze 440Hz con pitch change ratio di 0.794 e lunghezza delle rampe variabile

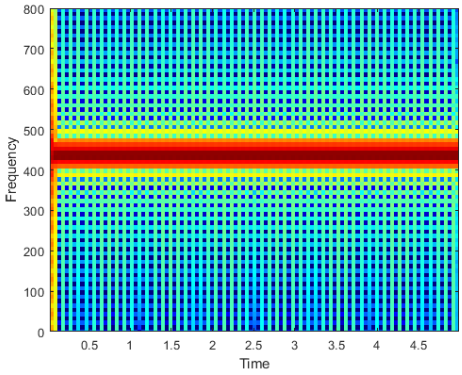
Osservando gli spettrogrammi di uno shift maggiore, come in figura 2.8 dove viene mostrato uno shift con un pitch change ratio desiderato di 1.999, si nota che l'uso di una rampa non abbastanza lunga, in questo caso 1024 campioni, dà in uscita pitch change ratio effettivo che non corrisponde a quello desiderato. Con rampe corte l'algoritmo non modifica più il segnale come richiesto.



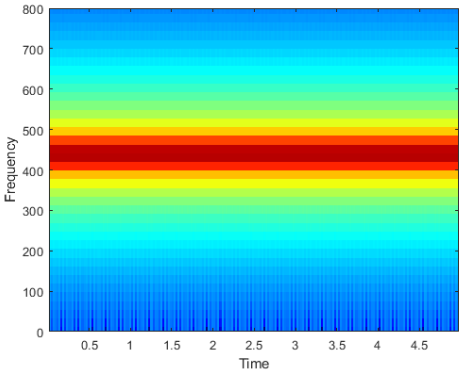
Sottofigura a) Segnale a 220Hz, PCR 1.999, LEN 1024



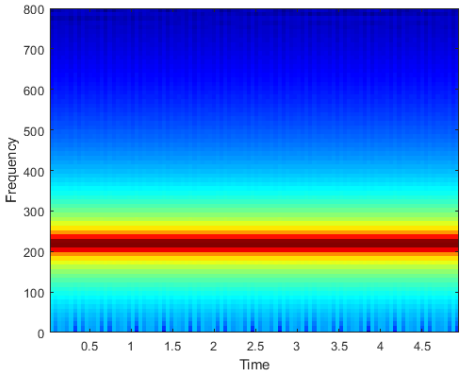
Sottofigura b) Segnale a 220Hz, PCR 1.999, LEN 2048



Sottofigura c) Segnale a 220Hz, PCR 1.999, LEN 4096



Sottofigura d) Segnale che dovrei ottenere a 440Hz



Sottofigura e) Segnale originale a 220Hz

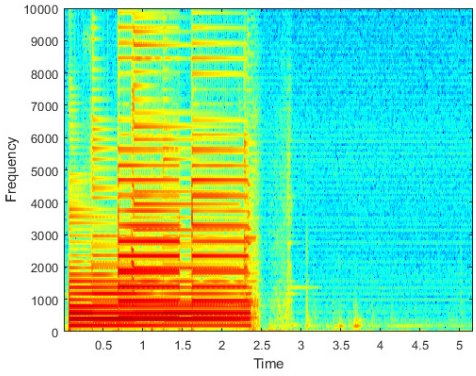
Figura 2.8 Spettrogrammi dei segnali di uscita dell'elaborazione di una sinusoide a frequenze 220Hz con pitch change ratio di 1.999 e lunghezza delle rampe variabile. Confronto con il segnale desiderato e il segnale in ingresso.

I problemi principali del metodo sono riassunti in questa tabella riepilogativa (Tabella 2.1).

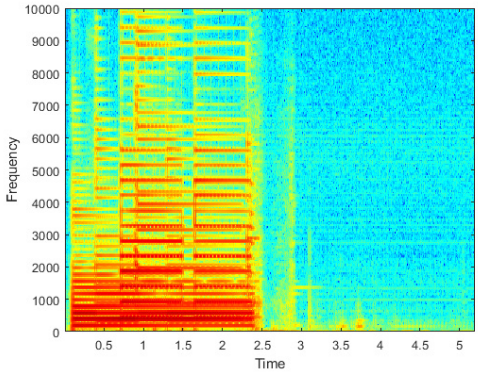
Problemi dell'algoritmo	Parametri che li influenzano	Cosa comportano
Cross-fading fra segnali uguali, ma sfasati	LEN, PCR	Modulazione in ampiezza variabile in frequenza
Cambi di delay-line	LEN	Disturbi presenti periodicamente
PCR ottenuto diverso da quello richiesto	LEN	Necessità di usare una LEN maggiore di una soglia limite

Tabella 2.1 Principali problemi del metodo a 2 delay-line

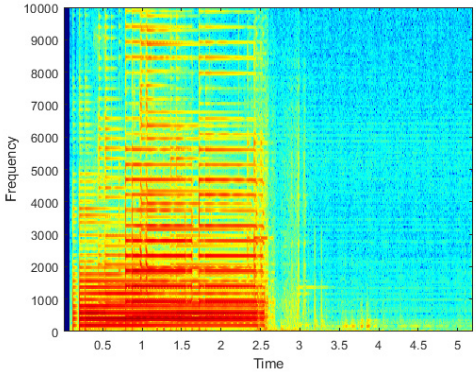
Analizzando i grafici ottenuti dal processamento di segnali più complessi, come il segnale di un synth-bass e di un arpeggio di chitarra, si può vedere come aumentando la lunghezza della finestra aumenti la nitidezza dello spettro, l'ascolto conferma l'aumento di qualità per rampe più lunghe.



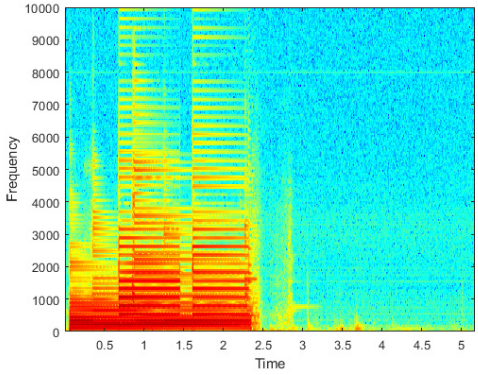
Sottofigura a) Arpeggio di chitarra, PCR 1.78, LEN 1024



Sottofigura b) Arpeggio di chitarra, PCR 1.78, LEN 4096



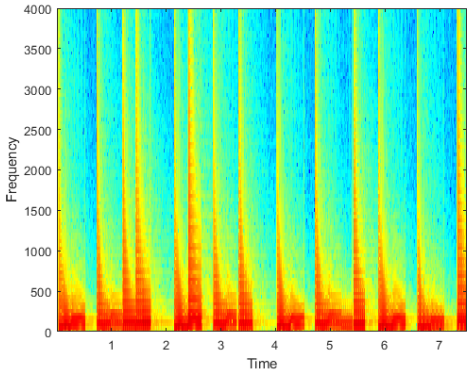
Sottofigura c) Arpeggio di chitarra, PCR 1.78, LEN 16384



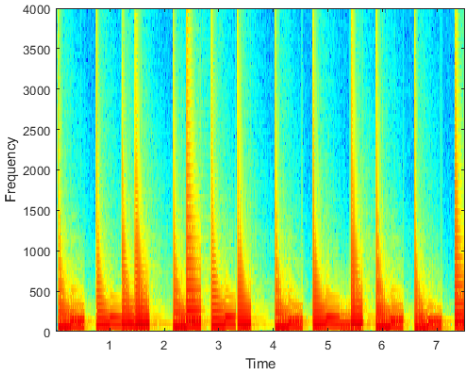
Sottofigura d) Arpeggio di chitarra segnale originale

Figura 2.9 Spettrogrammi dei segnali di uscita dell'elaborazione di un arpeggio di chitarra con pitch change ratio di 1.78 e lunghezza delle rampe variabile. Confronto con il segnale in ingresso.

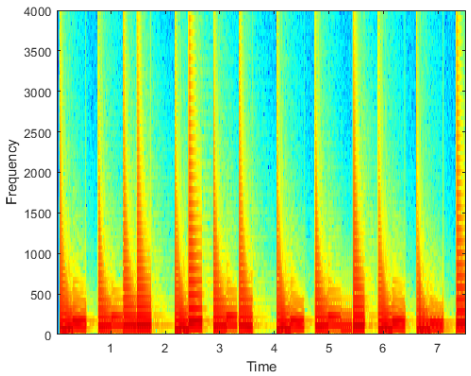
Gli spettrogrammi dell'elaborazione del synth-bass sono un po' meno chiari, ma l'ascolto conferma l'aumento di qualità per rampe più lunghe.



Sottofigura a) Giro di synth-bass, PCR 0.794, LEN 1024



Sottofigura b) Giro di synth-bass, PCR 0.794, LEN 4096



Sottofigura c) Giro di synth-bass segnale originale

Figura 2.10 Spettrogrammi dei segnali di uscita dell'elaborazione di un giro di synth-bass con pitch change ratio di 0.794 e lunghezza delle rampe variabile. Confronto con il segnale in ingresso.

2.3 Algoritmo Octave per il metodo a 3 delay-line

Molte delle considerazioni fatte per l'algoritmo a due linee valgono anche per quello a tre. Le rampe vengono costruite allo stesso modo, eccetto per lo sfasamento fra di esse, a partire dalla relazione fra PCR e il numero di semitoni dello shift [7]:

$$pitch\ change\ ratio = 2^{n\_semitoni/12} \qquad (2.4)$$

e dall'equazione per calcolare il massimo ritardo:

$$WIDTH=(LEN-1)*|PCR-1| \qquad (2.5)$$

Per effettuare il cross-fading tra le tre linee si va prima a realizzare una modulazione di ampiezza di ogni singola linea con dei segnali sinusoidali sfasati di 120°, poi si sommano i tre segnali ottenuti. Dunque rispetto al metodo a due linee la differenza a livello computazionale sta nel dover calcolare l'interpolazione di una linea in più e realizzare il cross-fading tra tre linee.

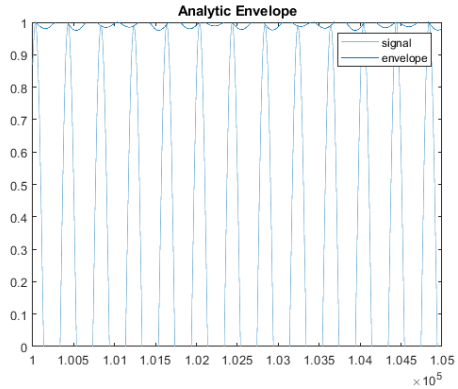
Come per l'algoritmo a due linee la scelta implementativa limita lo shift massimo a più un'ottava o meno un'ottava perché per shift elevati si perde qualità. Per funzionare correttamente l'algoritmo ha bisogno di rampe di lunghezza maggiore rispetto all'algoritmo a due rampe, il che rende difficile l'uso real-time.

I problemi sono gli stessi già presentati per l'algoritmo a due linee (Tabella 2.2).

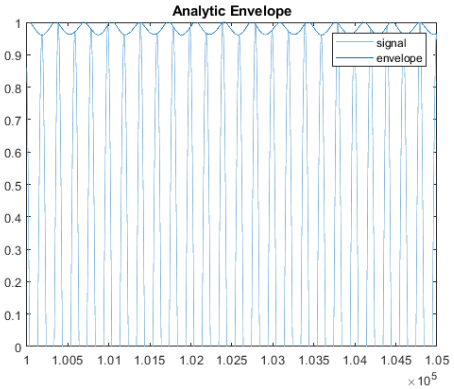
Problemi dell’algoritmo	Parametri che li influenzano	Cosa comportano
Cross-fading fra segnali uguali, ma sfasati	LEN, PCR	Modulazione in ampiezza variabile in frequenza
Cambi di delay-line	LEN	Disturbi presenti periodicamente con periodo LEN
PCR ottenuto diverso da quello richiesto	LEN	Necessità di usare una LEN maggiore di una soglia limite

Tabella 2.2 Principali problemi del metodo a 3 delay-line

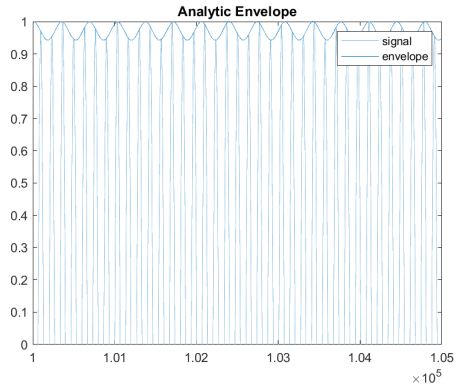
La modulazione in ampiezza dà meno fastidio nel complesso, per questo si può raggiungere una qualità maggiore, ma sono comunque presenti delle frequenze in cui è molto accentuata (vedere nella figura 2.11 il risultato alla frequenza di 660Hz).



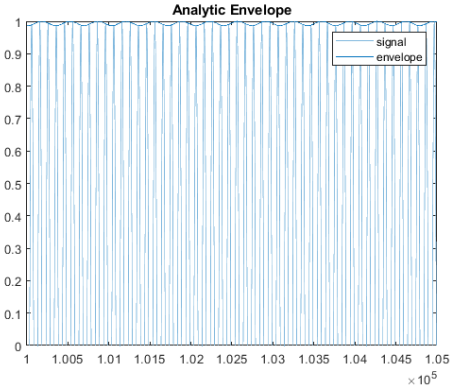
Sottofigura a) Segnale a 110Hz, PCR 1.1, LEN 1024



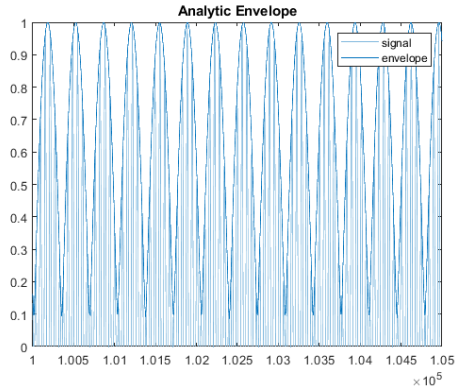
Sottofigura b) Segnale a 220Hz, PCR 1.1, LEN 1024



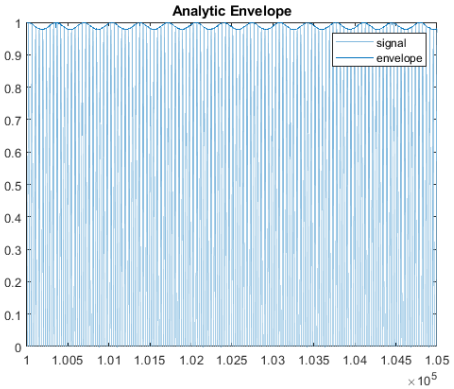
Sottofigura c) Segnale a 330Hz, PCR 1.1, LEN 1024



Sottofigura d) Segnale a 440Hz, PCR 1.1, LEN 1024



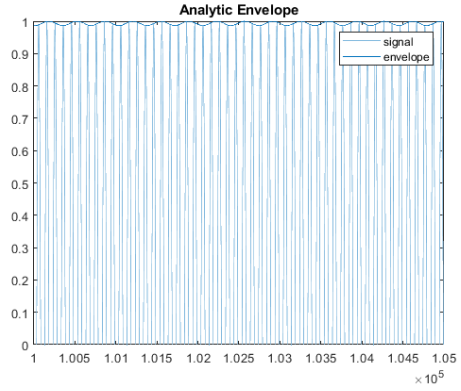
Sottofigura e) Segnale a 660Hz, PCR 1.1, LEN 1024



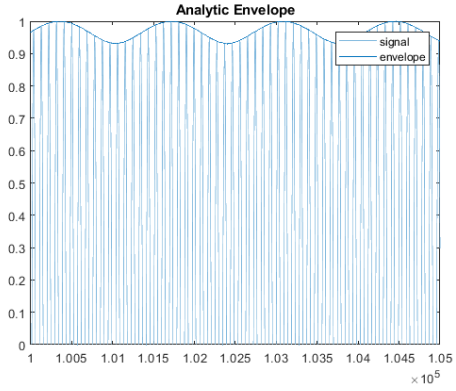
Sottofigura f) Segnale a 880Hz, PCR 1.1, LEN 1024

Figura 2.11 Porzione dei segnali di uscita dell'elaborazione di sinusoidi a varie frequenze con pitch change ratio di 1.1 e lunghezza delle rampe di 1024 campioni

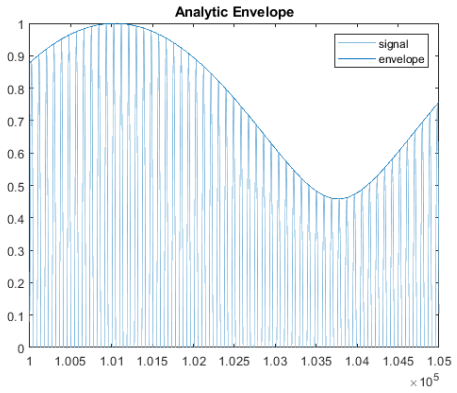
Come prima anche variando la lunghezza delle rampe varia l'effetto (figura 2.12).



Sottofigura a) Segnale a 440Hz, PCR 1.1, LEN 1024



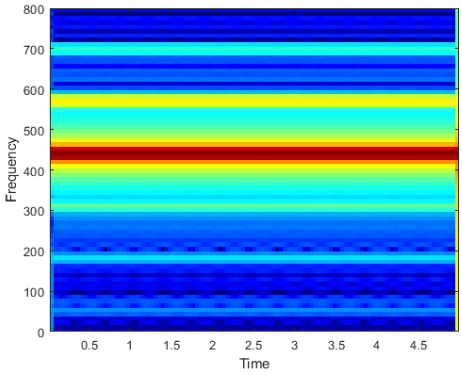
Sottofigura b) Segnale a 440Hz, PCR 1.1, LEN 4096



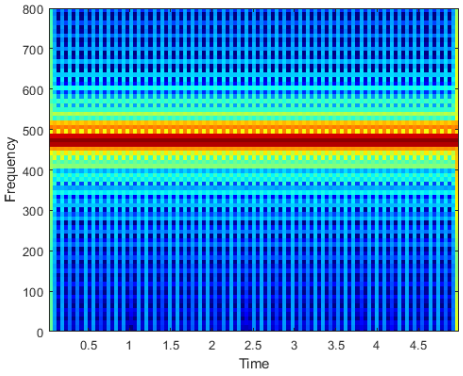
Sottofigura c) Segnale a 440Hz, PCR 1.1, LEN 16384

Figura 2.12 Porzione dei segnali di uscita dell'elaborazione di sinusoidi a frequenza 440Hz con pitch change ratio di 1.1 e lunghezza delle rampe variabile

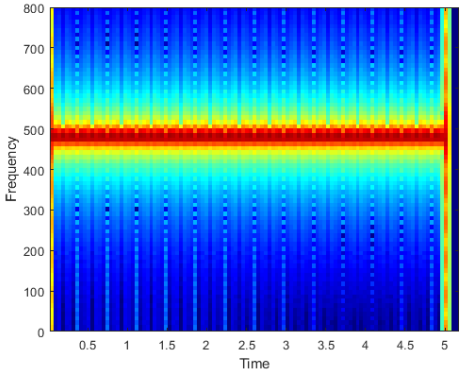
La lunghezza minima delle rampe è maggiore. Come si vede nella figura 2.13 anche con piccoli shift se la rampa è di 1024 campioni il pitch change ratio effettivo in uscita non corrisponde a quello desiderato.



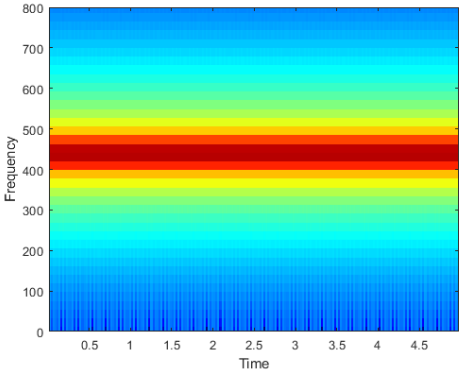
Sottofigura a) Segnale a 440Hz, PCR 1.1, LEN 1024



Sottofigura b) Segnale a 440Hz, PCR 1.1, LEN 4096



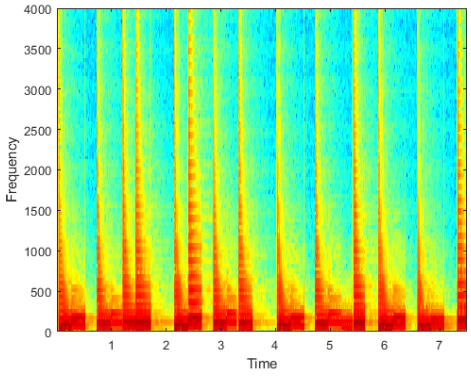
Sottofigura c) Segnale a 440Hz, PCR 1.1, LEN 16384



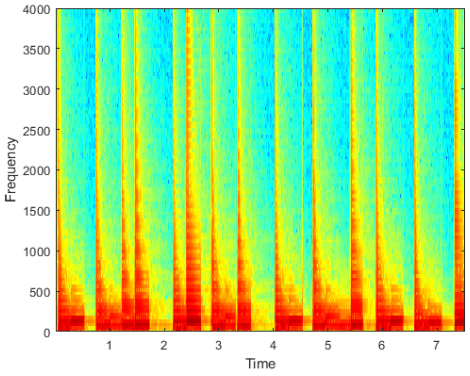
Sottofigura d) Segnale a 440Hz originale

Figura 2.13 Spettrogrammi dei segnali di uscita dell'elaborazione di una sinusoide a frequenze 440Hz con pitch change ratio di 1.1 e lunghezza delle rampe variabile. Confronto con il segnale in ingresso.

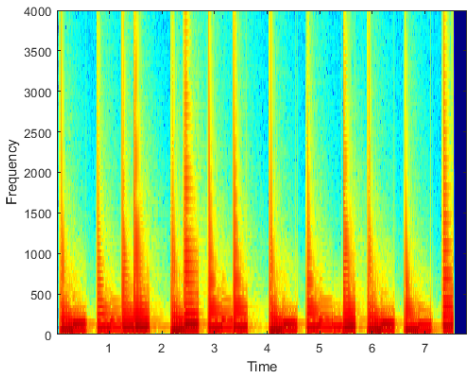
Negli spettrogrammi dei risultati dell'elaborazione di segnali più complessi si conferma quello già detto. La qualità raggiunta può essere migliore dell'implementazione a due linee, ma sono necessarie rampe molto lunghe che sono incompatibili all'uso real-time (figura 2.14).



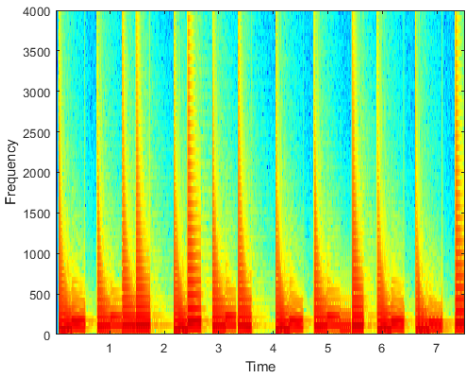
Sottofigura a) Giro di synth-bass, PCR 0.794, LEN 1024



Sottofigura b) Giro di synth-bass, PCR 0.794, LEN 4096



Sottofigura c) Giro di synth-bass, PCR 0.794, LEN 16384



Sottofigura d) Giro di synth-bass originale

Figura 2.14 Spettrogrammi dei segnali di uscita dell'elaborazione di un giro di synth-bass con pitch change ratio di 0.794 e lunghezza delle rampe variabile. Confronto con il segnale in ingresso.

Capitolo 3

PSOLA (Pitch-Synchronous Overlap and Add)

3.1 Descrizione del metodo

L'algoritmo PSOLA si basa sull'ipotesi che il suono in ingresso sia caratterizzato da un'altezza. Questo significa che il suo utilizzo è limitato a suoni monofonici come la voce umana e la sua applicazione a suoni polifonici risulta impossibile [8][2]; d'altro canto il metodo PSOLA per i suoni monofonici permette la conservazione delle formanti del suono originale. L'operazione effettuata equivale al duale del ricampionamento nel dominio del tempo, in questo caso è infatti eseguito un ricampionamento dell'inviluppo spettrale ottenuto dall'analisi di Fourier a tempo breve. L'inviluppo spettrale a tempo breve descrive una curva in frequenza passando attraverso tutte le ampiezze delle armoniche. Siano β il pitch change ratio, f_i le frequenze delle armoniche del segnale e a_i le rispettive ampiezze, cioè il valore dell'inviluppo a quella frequenza. Le armoniche sono scalate in base alla nuova $f_i^{new} = \beta \cdot f_i^{old}$, ma le ampiezze delle armoniche sono ancora determinate campionando l'inviluppo spettrale rimasto invariato, saranno dunque diverse da prima (figura3.1):

$$a_i^{new} = env(f_i^{new}) \neq a_i^{old} \qquad (3.1)$$

Mantenendo la posizione delle formanti è mantenuta l'identità timbrica dello strumento o della voce. L'idea di base consiste nello stiramento temporale della posizione dei marcatori di intonazione (o pitch markers), mentre il segmento della forma d'onda non viene modificato.

Ad esempio, il modello del segnale del parlato è un treno di impulsi filtrato attraverso un filtro variabile nel tempo corrispondente al tratto vocale. Il segmento di ingresso corrisponde alla risposta impulsiva del filtro e determina la posizione delle formanti, pertanto tale risposta non dovrebbe essere modificata. Al contrario la distanza tra i marcatori di intonazione determina il periodo del parlato e, quindi, tale distanza deve essere modificata di conseguenza. Lo scopo dell'analisi PSOLA è quello di estrarre la risposta impulsiva locale del filtro.

Lo spettro di un segmento estratto utilizzando una finestra di Hanning con una lunghezza di due periodi approssima l'inviluppo spettrale locale. Finestre più lunghe tendono ad evidenziare la struttura fine delle righe dello spettro, mentre finestre più corte tendono ad offuscare la struttura delle formanti dello spettro (figura 3.2).

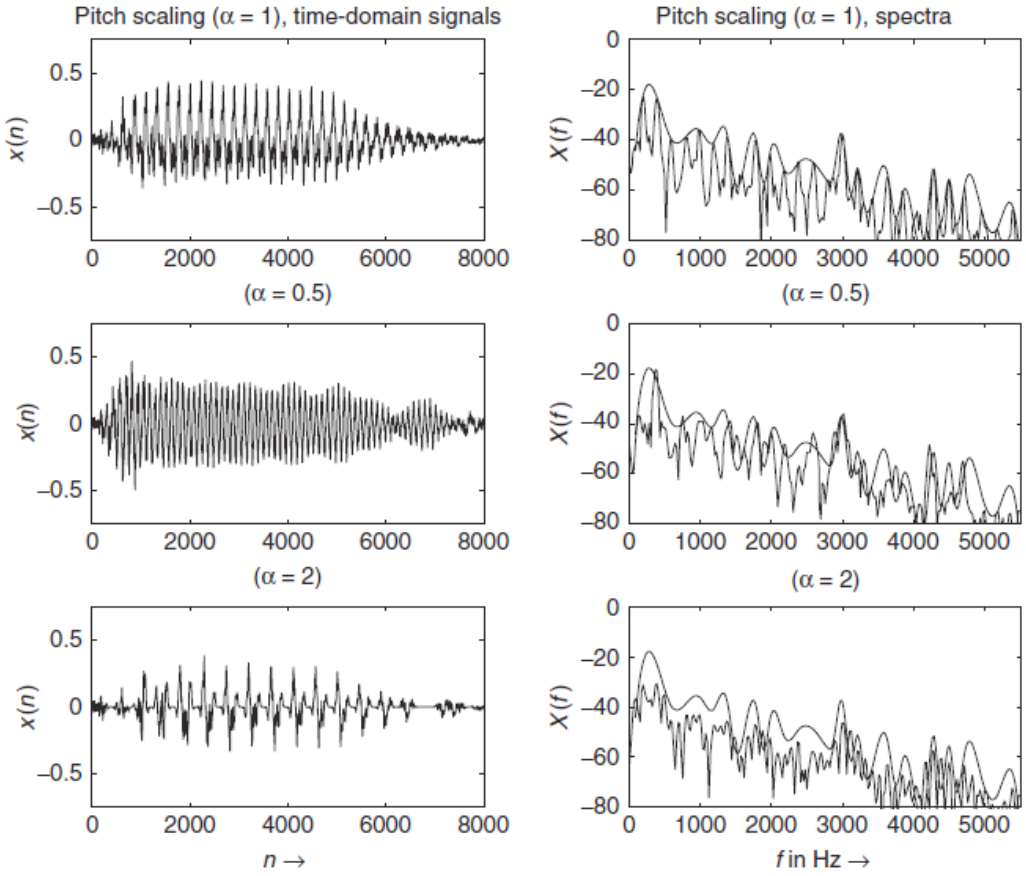


Figura 3.1 Pitch shifting tramite PSOLA: ricampionamento in frequenza dell'inviluppo spettrale

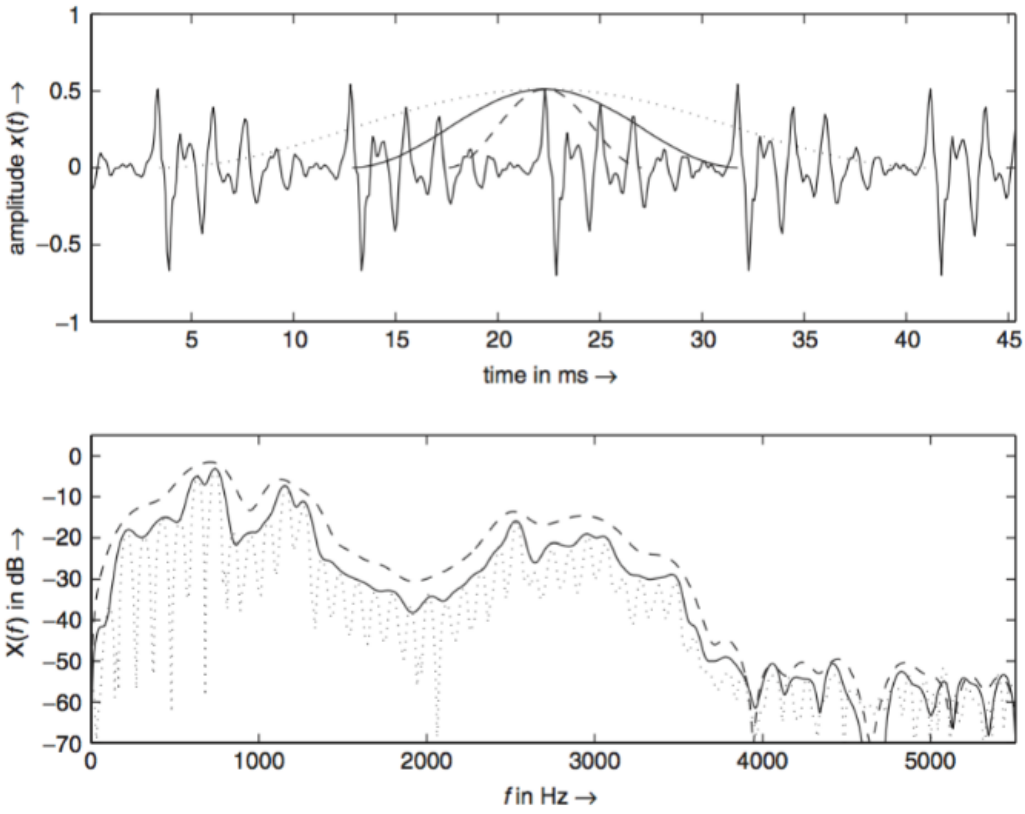


Figura 3.2 Effetto di finestre di Hanning di varia lunghezza sulla stima dell'inviluppo spettrale

Così se non si estende il segmento, la posizione delle formanti viene mantenuta. L'operazione di overlap dei segmenti alla posizione del nuovo pitch mark ricampionerà l'inviluppo spettrale alla frequenza di intonazione desiderata. Quando si desidera un cambio di intonazione di un fattore β, definito come il rapporto tra la frequenza del tono di sintesi locale rispetto a quello originale:

$$\beta = \tilde{f}_0(\tilde{t}) / f_0(t) \qquad (3.2)$$

il nuovo periodo di intonazione sarà dato da:

$$\tilde{P}(\tilde{t}) = P(t) / \beta \qquad (3.3)$$

se il tempo non è modificato oppure se viene eseguita anche un'operazione di time-stretching $\tilde{t} = \alpha t$, P è il periodo locale in ingresso.

L'algoritmo si compone di due fasi [9][2]:

Fase di analisi:

1. Determinazione del periodo di intonazione P(t) del segnale di ingresso e degli istanti t_i (pitch marks). Questi riferimenti sono definiti in corrispondenza della massima ampiezza ad un tasso sincronizzato con l'altezza durante la parte periodica del suono e definiti ad un tasso costante durante le porzioni non vocalizzate. In pratica P(t) è considerato costante $P(t) = P(t_i) = t_{i+1} - t_i$ sull'intervallo di tempo (t_i, t_{i+1}).
2. Estrazione di un segmento centrato a ogni riferimento di intonazione utilizzando una finestra di Hanning con lunghezza $L_i = 2P(t_i)$ per assicurare fade-in e fade-out.

Fase di sintesi: per ogni riferimento di intonazione $\tilde{t}_k$

1. Scelta del segmento di analisi corrispondente i (identificato dal marcatore temporale t_i) minimizzando la distanza temporale $|\alpha t_i - \tilde{t}_k|$
2. Sovrapposizione e somma del segmento selezionato. Alcuni segmenti di ingresso saranno ripetuti se $\alpha > 1$ (espansione temporale) o $\beta > 1$ (pitch più alto). Alcuni segmenti di ingresso saranno scartati se $\alpha < 1$ (compressione temporale) o $\beta < 1$ (pitch più basso).
3. Determinazione dell'istante di tempo $\tilde{t}_{k+1}$ in cui sarà centrato il successivo segmento di sintesi, al fine di preservare l'altezza locale tramite la relazione

$$\tilde{t}_{k+1} = \tilde{t}_k + \tilde{P}_k(\tilde{t}_k) = \tilde{t}_k + P(t_i) \qquad (3.4)$$

L'algoritmo permette di lasciare inalterate le formanti, ma all'occorrenza permette anche di modificarle. In effetti un aspetto versatile dello PSOLA è che permette di scalare durata, pitch e formanti indipendentemente fra di esse [10]. Per ottenere l'alterazione delle formanti si esegue la scalatura temporale dei segmenti di ingresso selezionati prima di eseguire l'overlap and add del passo di sintesi, senza alcuna modifica nel calcolo dei marcatori di intonazione. Per aumentare le frequenze dei formanti di un fattore γ, ogni segmento dovrebbe essere ridotto di un fattore 1/γ attraverso il ricampionamento.

3.2 Algoritmi Octave per il pitch detection e il pitch marking

Il pitch tracking è necessario all'implementazione dell'algoritmo e il suo costo computazionale rende più difficile l'uso real-time dello PSOLA. L'implementazione di un algoritmo di pitch tracking sufficientemente buono e rapido non è riuscita e si è optato per l'uso di un algoritmo già esistente visto che il problema del pitch tracking è un problema non banale e non è un obiettivo della trattazione.
Anche il posizionamento dei marcatori richiede un costo computazionale rilevante. Questo problema può essere risolto senza grosse perdite di qualità posizionando periodicamente i marcatori ad un tasso sincrono con il periodo del frame corrente.
Questa soluzione non permette l'uso di hop size piccoli, poiché altrimenti si corre il rischio di posizionare i marcatori fuori dal frame corrispondente. Tutto sommato questo non provoca problemi perché l'algoritmo riesce a funzionare anche per piccoli o nulli overlap. Diversamente la lunghezza dei frame incide sulla qualità del pitch tracking, una finestra troppo piccola rende difficile compiere il giusto tracciamento.

3.3 Algoritmo Octave per l'elaborazione PSOLA

Ogni finestra del segnale viene prima di tutto elaborata dal pitch tracker che individua la frequenza fondamentale istantanea del segnale [7]. Successivamente la funzione di pitch marking posiziona i marcatori temporali in base alla frequenza individuata [7]. Una volta effettuate queste due prime operazioni preliminari il segnale viene elaborato dal vero e proprio algoritmo PSOLA [7], in base al valore del pitch change ratio che viene calcolato come in precedenza tramite l'equazione:

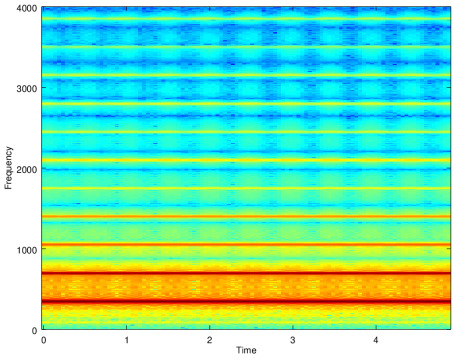
$$pitch\ change\ ratio = 2^{n\_semitoni/12} \qquad (2.4)$$

L'algoritmo esegue esattamente i passaggi descritti dal metodo descritto precedentemente. È stata implementata anche la possibilità di variare la posizione delle formanti indipendentemente dalla variazione del pitch tramite il parametro γ. La variazione della velocità realizzabile tramite un terzo parametro α è stata implementata, anche se per ovvi motivi non è possibile farne uso in real-time.

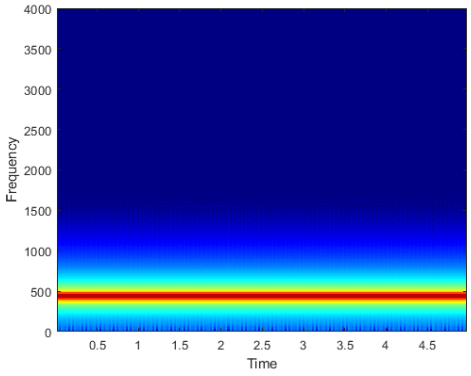
La qualità raggiungibile è superiore rispetto a quella ottenuta con il metodo tramite delay-line, inoltre per finestre compatibili con l'uso real-time l'algoritmo permette un controllo più preciso dei parametri. La scelta di posizionare i marcatori temporali in modo periodico per alleggerire i costi di elaborazione provoca delle distorsioni negli attacchi dei suoni.

Utilizzando un tono puro in ingresso e confrontando uno shift con conservazione delle formanti e uno con traslazione delle formanti pari a quella del pitch si osserva che nel primo caso compaiono delle armoniche della frequenza fondamentale, mentre nel secondo resta la presenza di un tono puro (figura 3.3).

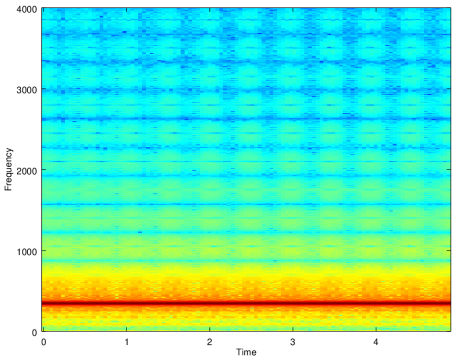
Questo comportamento dipende probabilmente dalla finestratura del segnale. Infatti, come presumibile, lasciando invariate le formanti e variando il pitch, l'ampiezza del segnale in uscita ricalca l'andamento dell'ampiezza dello spettro della finestra di Hanning usata per la finestratura del segnale.



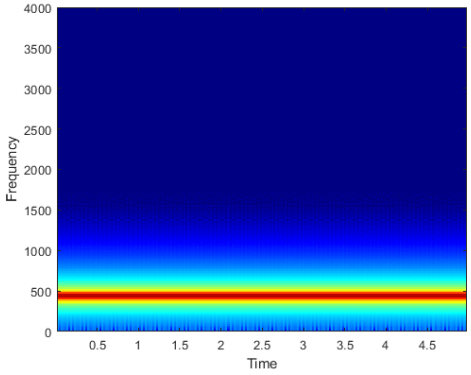
Sottofigura a) Segnale a 440Hz, PCR 0.794, LEN 1024, HOP 1024, conservazione delle formanti



Sottofigura b) Segnale a 440Hz originale



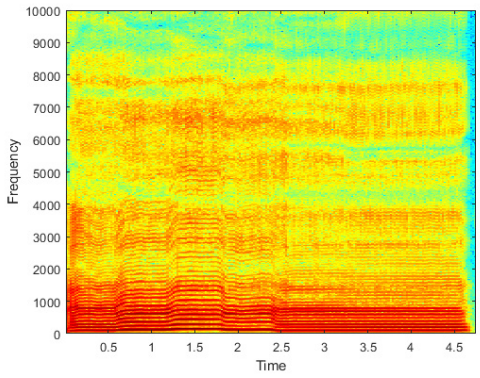
Sottofigura c) Segnale a 440Hz, PCR 0.794, LEN 1024, HOP 1024, traslazione delle formanti



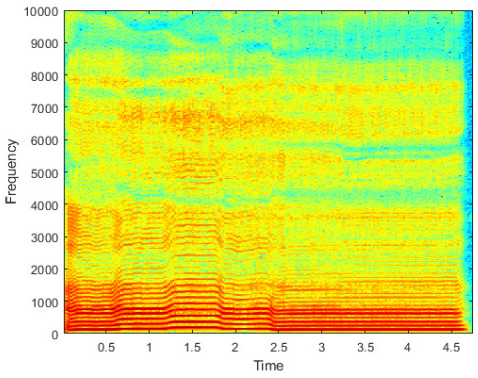
Sottofigura d) Segnale a 440Hz originale

Figura 3.3 Spettrogrammi dei segnali di uscita dell'elaborazione di una sinusoide a frequenze 440Hz con pitch change ratio di 0.794, lunghezza delle finestre 1024, hop size 1024. Confronto fra il segnale in ingresso e l'elaborazione con e senza conservazione delle formanti.

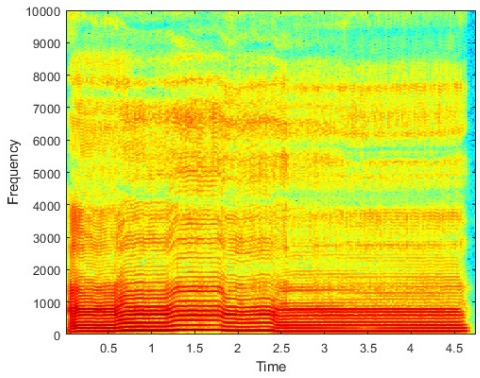
L’algoritmo PSOLA è particolarmente efficace se usato sulle voci, dove la conservazione delle formanti è importante per non creare un suono timbricamente irrealistico. Negli shift verso il basso con conservazione delle formanti di figura 3.4 si può vedere come l’efficacia dell’algoritmo mantenga la qualità elevata e permetta un controllo preciso della variazione. La differenza di qualità tra le due scelte, con e senza overlap, è praticamente impercettibile, e questo è confermato anche dall’ascolto.



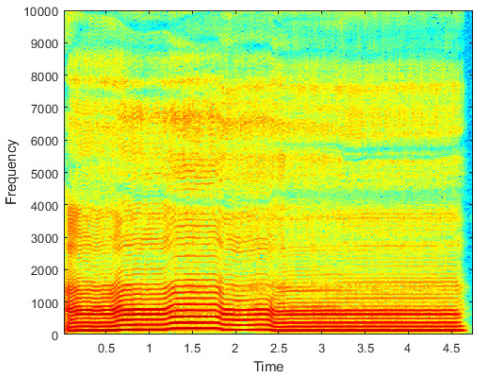
Sottofigura a) Vocalizzo, PCR 0.794, LEN 1024, HOP 512, conservazione delle formanti



Sottofigura b) Vocalizzo originale



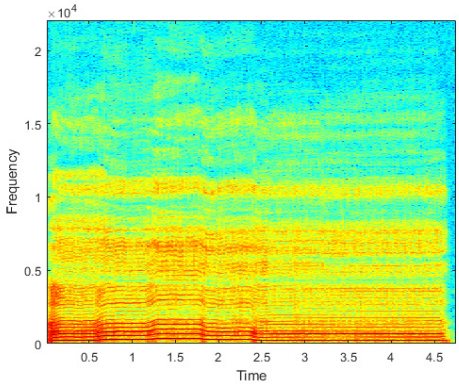
Sottofigura c) Vocalizzo, PCR 0.794, LEN 1024, HOP 1024, conservazione delle formanti



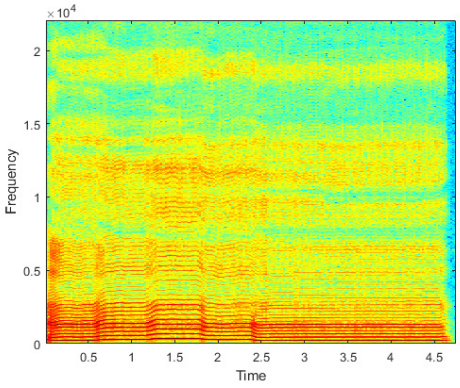
Sottofigura d) Vocalizzo originale

Figura 3.4 Spettrogrammi dei segnali di uscita dell’elaborazione di un vocalizzo con pitch change ratio di 0.794, lunghezza delle finestre 1024, hop size variabile. Confronto con il segnale in ingresso

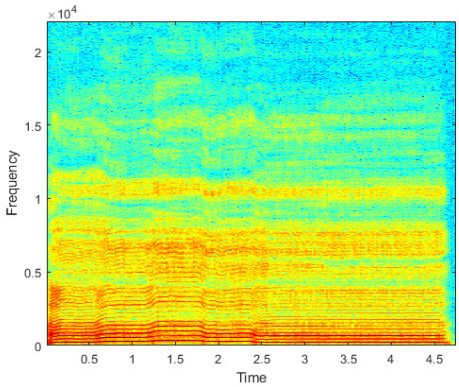
Nella figura 3.5 viene messa in evidenza la capacità dell'algoritmo di modificare le formanti del segnale indipendentemente dal pitch. Lasciando inalterate le formanti la fedeltà timbrica può essere maggiore, soprattutto per segnali vocali come quello qui considerato. L'indipendenza dei due parametri permette inoltre di ottenere maggiori possibilità espressive.



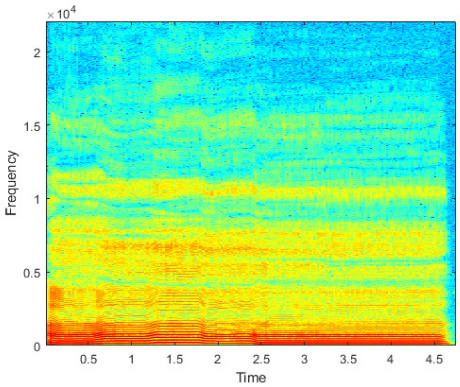
Sottofigura a) Vocalizzo, PCR 1.78, LEN 1024, HOP 1024, conservazione delle formanti



Sottofigura b) Vocalizzo, PCR 1.78, LEN 1024, HOP 1024, traslazione delle formanti



Sottofigura c) Vocalizzo, PCR 1.78, LEN 1024, HOP 512, conservazione delle formanti



Sottofigura d) Vocalizzo originale

Figura 3.5 Spettrogrammi dei segnali di uscita dell'elaborazione di un vocalizzo con pitch change ratio di 1.78, lunghezza delle finestre 1024, hop size variabile. Confronto con il segnale in ingresso dell'elaborazione con e senza conservazione delle formanti.

I problemi principali del metodo sono riassunti in questa tabella riepilogativa (Tabella 3.1).

Problemi dell'algoritmo	Parametri che li influenzano	Cosa comportano
Possibilità di uso solo su segnali monofonici	-	Nel caso di un'implementazione su scheda bisogna rendere chiaro all'utente che l'uso possibile è limitato
Difficoltà di eseguire un corretto pitch tracking	LEN	La lunghezza delle finestre deve essere tale da garantire un pitch tracking corretto
Scelta di posizionare periodicamente i pitch marks per ridurre l'onere computazionale	-	Distorsione degli attacchi dei suoni

Tabella 3.1 Principali problemi del metodo PSOLA

Capitolo 4

Block-by-block approach (FFT/IFFT)

4.1 Descrizione del metodo

Diversamente dai primi due metodi il terzo approccio compie un'elaborazione nel dominio della frequenza.
L'approccio block-by-block parte da un algoritmo di analisi che compie la FFT su finestre scorrevoli. La FFT calcola i valori delle ampiezze e delle fasi della finestra, consentendo un tempo di calcolo breve. Poiché ci sono modi diversi di interpretare una trasformata di Fourier su finestre scorrevoli, esistono più metodi di risintesi, principalmente: tramite somma di sinusoidi o tramite IFFT. Il metodo utilizzato nell'algoritmo è l'applicazione della trasformata inversa allo spettro delle finestre e l'uso dell'overlap and add per ricostruire il segnale in uscita (figura 4.1).

Viene eseguito un time-stretch e un ricampionamento, infatti il ricampionamento di un segnale, su cui è stato eseguito un time-stretch, di un fattore inverso a quello del time-stretching ratio esegue un pitch shift e torna alla durata iniziale del segnale. Dallo spettro delle finestre si può riottenere perfettamente il segnale se la somma delle finestre di overlap è unitaria.
Per prima cosa bisogna portare l'origine del tempo dalla sinistra della finestra al centro tramite una traslazione circolare [11]. In questo modo il risultato della FFT è equivalente ad un banco di filtri a fase zero. Successivamente bisogna calcolare le nuove fasi. La tecnica si basa sull'interpolazione di fase, che necessita durante il processo di analisi di un algoritmo di unwrapping o, equivalentemente, un calcolo della frequenza istantanea.

Sia R_a l'hop size di analisi, s l'indice delle finestre, $\tilde{\varphi}_u$ la fase unwrapped, *princarg* una funzione che porta una fase arbitraria in un intervallo] -π radianti; π radianti], $\tilde{\omega}_k = 2\pi k/N$ con N pari alla lunghezza della finestra della FFT e sia infine:

$$\tilde{\varphi}(n,k) = \omega_k n + \varphi(n,k) \qquad (4.1)$$

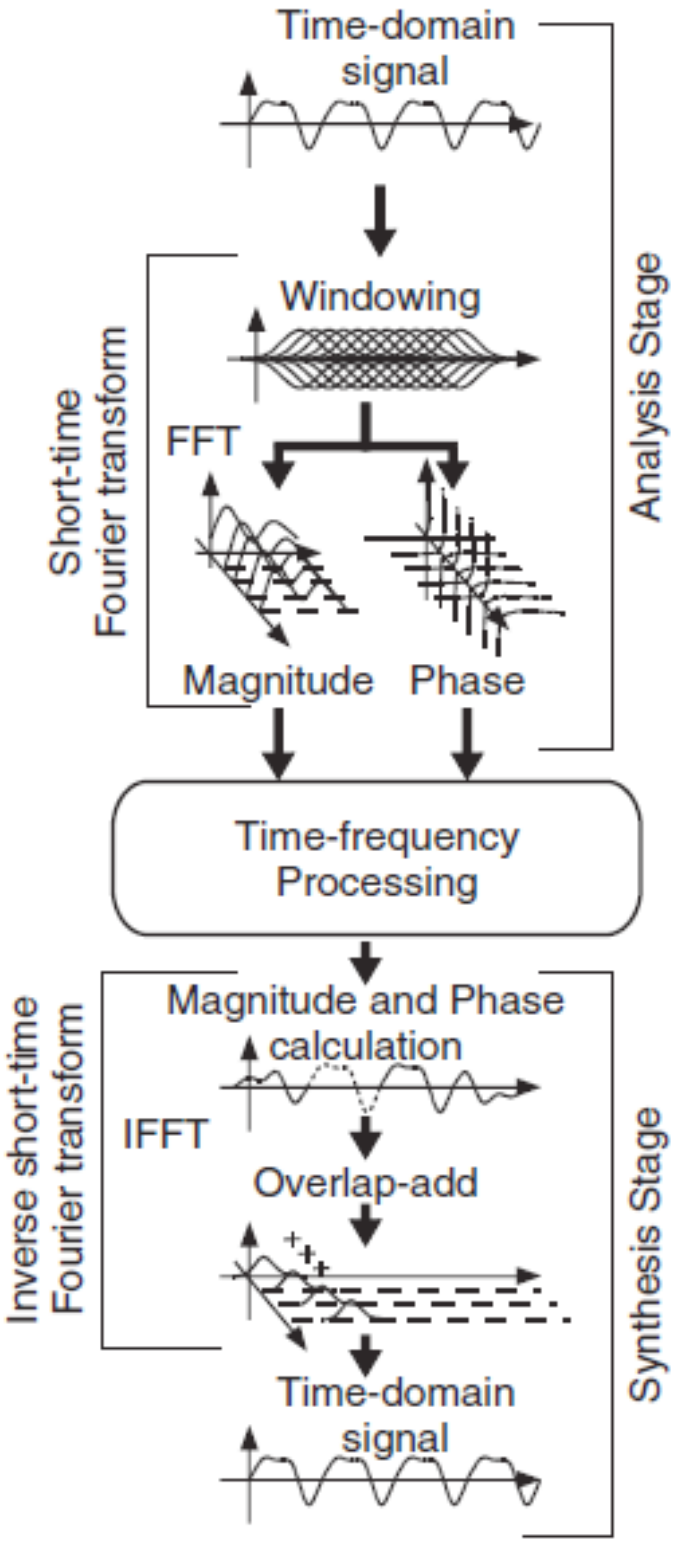


Figura 4.1 Fasi del processo di elaborazione in frequenza

Il calcolo della variazione di fase da applicare parte dalla FFT di due frame consecutivi, la fase unwrapped può essere vista come la somma di una fase target e una fase di deviazione con:

$$\tilde{\varphi}_t\,((s+1)R_a) = \tilde{\varphi}(sR_a) + \omega_k R_a \qquad (4.2)$$

$$\tilde{\varphi}_t\,((s+1)R_a) = princarg\,[\tilde{\varphi}((s+1)R_a) - \tilde{\varphi}_t\,((s+1)R_a)] \qquad (4.3)$$

Si può quindi calcolare la variazione della fase unwrapped $\tilde{\varphi}_u((s+1)R_a) - \tilde{\varphi}(sR_a)$ come:

$$\Delta\varphi((s+1)R_a) = \tilde{\varphi}_u((s+1)R_a) - \tilde{\varphi}(sR_a)$$
$$= \omega_k R_a + princarg\,[\tilde{\varphi}((s+1)R_a) - \tilde{\varphi}(sR_a) - \omega_k R_a] \qquad (4.4)$$

Una volta calcolata, questa è moltiplicata per il fattore di time-stretching e aggiunta alla fase iniziale. Viene dunque ricostruito il nuovo spettro in frequenza e riportato nel tempo tramite IFFT. Infine avviene il ricampionamento (figura 4.2) del risultato di ogni IFFT e l'overlap and add con un hop size uguale a quello dell'analisi. A condizione che R_s sia un divisore di N (lunghezza della finestra della FFT), si può ricampionare ogni risultato dell'IFFT ad una lunghezza di $(NR_a)/R_s$ e overlappare con un hop size R_a. Dunque calcolando le nuove fasi possiamo ricostruire un nuovo suono tramite l'IFFT [2].

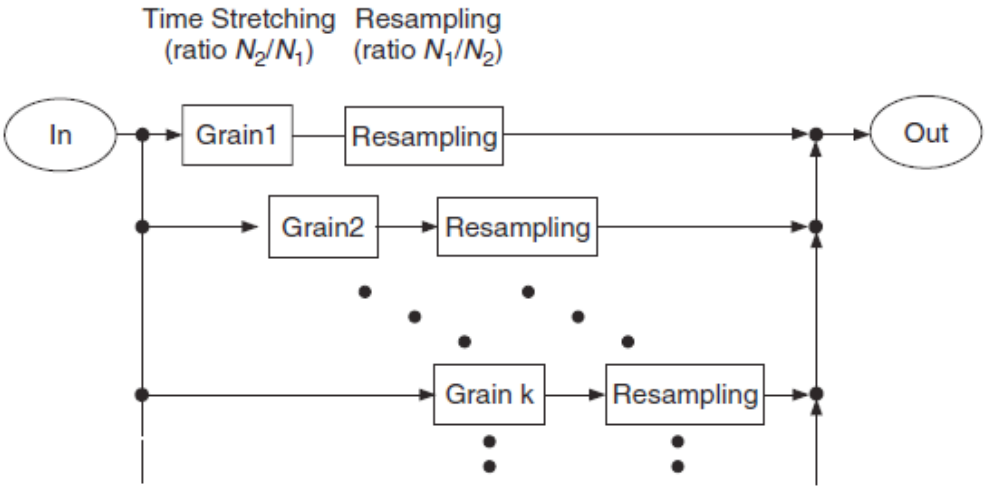


Figura 4.2 Metodo in frequenza: per ogni finestra effettuo un'operazione di time stretching e di ricampionamento. Il segnale in uscita è ricostruito tramite overlap and add.

4.2 Algoritmo Octave per il metodo block-by-block (FFT/ IFFT)

Una volta fissati la lunghezza delle finestre di analisi/sintesi e il valore dell'hop size di sintesi (n_2), dopo aver convertito il parametro dello shift dal numero di semitoni al valore di pitch change ratio con la solita formula:

$$pitch\ change\ ratio = 2^{n\_semitoni/12} \qquad (2.4)$$

si calcolano il valore dell'hop size di analisi (n_1) e del time stretch ratio [7]:

$$n_1 = round(n_1 / PCR) \qquad (4.5)$$

$$time\ stretch\ ratio = n_2 / n_1 \qquad (4.6)$$

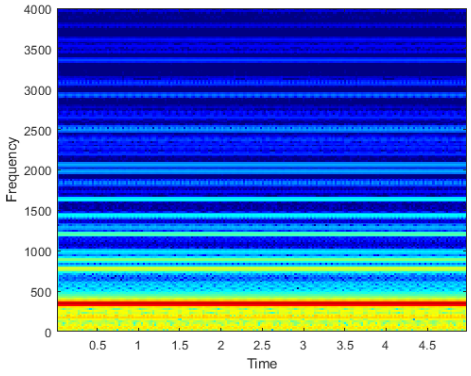
Si realizzano poi le finestre e le funzioni necessarie all'interpolazione dei campioni.
Si passa infine all'elaborazione vera e propria del segnale come è stata descritta nella sezione di descrizione del metodo.

L'algoritmo ha una buona qualità audio anche per variazioni di altezza abbastanza elevate.

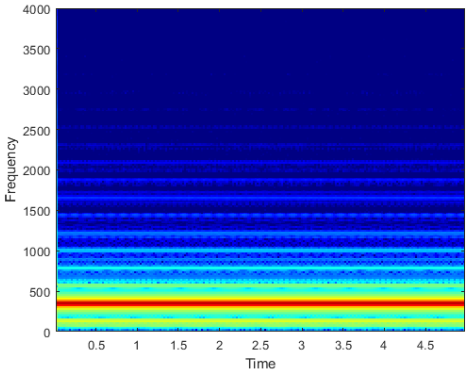
Restringendo la finestra di analisi si riduce la latenza, ma sotto una certa soglia (individuata a 512 campioni utilizzando 44,1kHz di frequenza di campionamento e finestre di lunghezza potenza di 2) vi è un brusco peggioramento del risultato che rende quasi inutilizzabile l'effetto. Se si lascia inalterato l'hop size restringendo la finestra si velocizza leggermente il processo, ma si perde qualità.
Stringendo l'hop size è possibile invece recuperare in qualità, ma come conseguenza il tempo di analisi tenderà a crescere proporzionalmente alla riduzione.
Se si tenta di lasciare inalterato l'hop size e stringere le finestre si otterrà un peggioramento della qualità del segnale in uscita, con la creazione di artefatti udibili. L'overlap ideale per avere basso costo computazionale e buona

qualità è stato individuato di circa il 75% della lunghezza dei frame, quindi l'hop size di un quarto. Con un fattore di shift maggiore di 1 si verifica un po' di foldover, probabilmente causato dalla scelta di usare l'interpolazione lineare.
Il costo computazionale è maggiore di quello dell'algoritmo tramite delay-line, ma permette di ottenere una qualità nettamente superiore. Inoltre diversamente dallo PSOLA, può essere utilizzato anche su suoni polifonici.

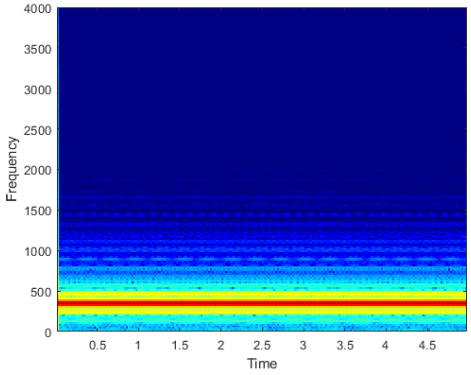
L’elaborazione di toni puri non mostra grandi problemi anche con finestre più brevi di quelle raccomandate, si vede solo dallo spettrogramma che appaiono maggiori frequenze indesiderate rispetto a finestre più lunghe.



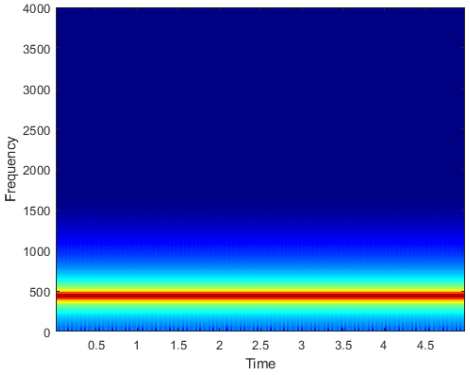
Sottofigura a) Segnale a 440Hz, PCR 0.794, LEN 256, HOP 64



Sottofigura b) Segnale a 440Hz, PCR 0.794, LEN 512, HOP 128



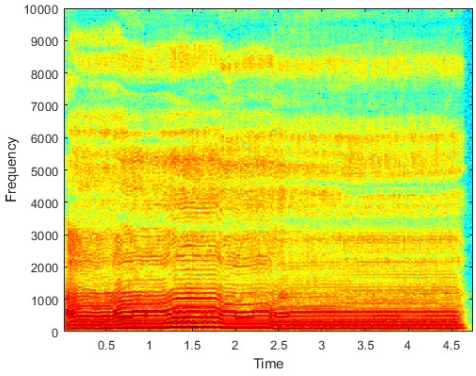
Sottofigura c) Segnale a 440Hz, PCR 0.794, LEN 1024, HOP 256



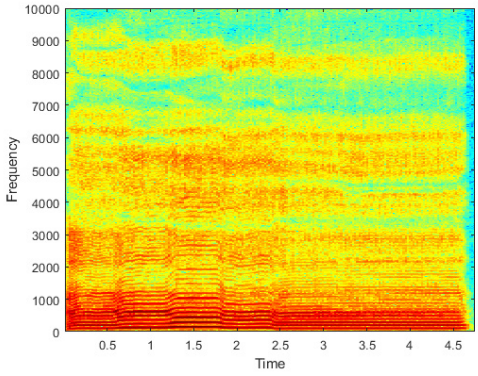
Sottofigura d) Segnale a 440Hz originale

Figura 4.3 Spettrogrammi dei segnali di uscita dell’elaborazione di una sinusoide a frequenze 440Hz con pitch change ratio di 0.794, lunghezza delle finestre variabile e hop size pari al 25%. Confronto con il segnale in ingresso

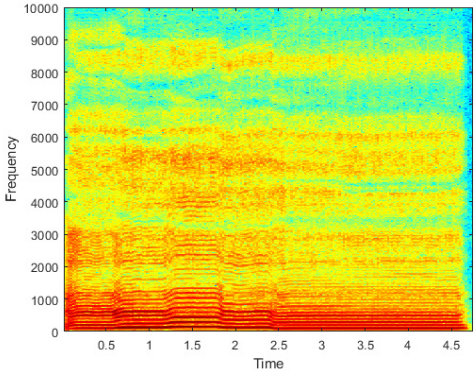
Elaborando segnali complessi si vede sia tramite gli spettrogrammi, dove le frequenze dello spettro appaiono più confuse rispetto a quelle ottenute con finestre più lunghe, e sia, soprattutto, tramite l'ascolto, dove si sentono problemi negli attacchi e presenza di repliche in sottofondo, che non è possibile scendere sotto il limite suggerito.



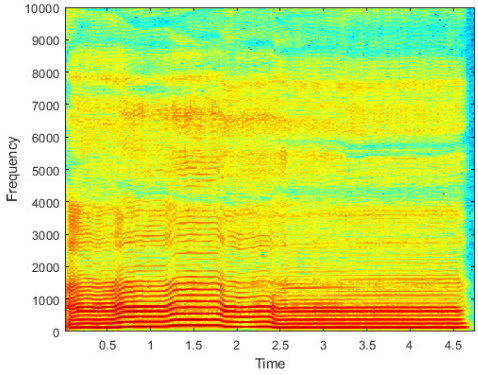
Sottofigura a) Vocalizzo, PCR 0.794, LEN 256, HOP 64



Sottofigura b) Vocalizzo, PCR 0.794, LEN 512, HOP 128



Sottofigura c) Vocalizzo, PCR 0.794, LEN 1024, HOP 256



Sottofigura d) Vocalizzo originale

Figura 4.4 Spettrogrammi dei segnali di uscita dell'elaborazione di un vocalizzo con pitch change ratio di 0.794, lunghezza delle finestre variabile e hop size pari al 25%. Confronto con il segnale in ingresso

I problemi principali del metodo sono riassunti in questa tabella riepilogativa (Tabella 4.1).

Problemi dell'algoritmo	Parametri che li influenzano	Cosa comportano
Lunghezza delle finestre	LEN	Con finestre troppo brevi l’algoritmo perde la sua efficacia
Rispetto allo PSOLA non mantiene lo spettro delle formanti	-	Per alcuni segnali, come ad esempio la voce, il risultato è meno gradevole rispetto al metodo PSOLA

Tabella 4.1 Principali problemi del metodo block-by-block

Capitolo 5

Implementazione in linguaggio C

5.1 Scelta degli algoritmi da implementare

Le quattro implementazioni su Octave non sono state tutte realizzate anche in linguaggio C. Una volta analizzati i risultati ottenuti con gli algoritmi Octave la scelta degli algoritmi da tradurre è stata basata sulle richieste e sui vincoli imposti dalla scheda.

Degli algoritmi basati sull'uso di delay-line modulate l'algoritmo a due linee è stato portato in linguaggio C poiché permette di avere un discreto pitch shift per piccole alterazioni (qualche tono o semitono) con un costo computazionale molto basso e un ritardo accettabile. Al contrario l'algoritmo basato su tre delay-line, pur permettendo di ottenere una qualità maggiore rispetto all'altro, non è stato portato in linguaggio C poiché il ritardo necessario al suo funzionamento è troppo elevato per usi real-time.
L'algoritmo PSOLA non è stato portato in C per via delle difficoltà di implementazione real-time, dovute sia alla necessità di dover non solo effettuare l'elaborazione del segnale ma anche di dover prima effettuare il pitch tracking e il pitch marking, sia perché l'algoritmo può solo essere usato su suoni monofonici e quindi ha possibilità d'uso limitate.
Infine il metodo block-by-block (FFT/IFFT) è stato portato in linguaggio C perché può essere usato real-time con buona qualità di uscita con tempi di latenza accettabili. Rispetto al metodo a due delay-line modulate rappresenta una scelta con un maggior costo computazionale, ma con qualità maggiore.

Per controllare che il risultato ottenuto con l'algoritmo in C fosse equivalente a quello ottenuto con gli algoritmi in Octave sono state salvate le elaborazioni in dei file che sono stati poi caricati su Octave. In questo modo si è potuto svolgere un confronto sia a livello audio, tramite l'ascolto dei file wav ottenuti dalla scrittura dei risultati delle elaborazioni nei due diversi linguaggi, sia a livello numerico, potendo paragonare campione per campione ciò che si aveva ottenuto. Sono stati confrontati gli spettrogrammi, le raffigurazioni dei segnali nel tempo ed è stata calcolata la differenza fra i segnali, in modo da capire quanto si discostassero a livello numerico le due implementazioni.

5.2 Algoritmo in linguaggio C per il metodo a 2 delay-line

L'implementazione in C dell'algoritmo a due delay-line mostra un'elevata corrispondenza all'implementazione in Octave con differenze massime dell'ordine di 10^{-4}, con gli stessi pregi e gli stessi difetti.

Come in Octave l'elaborazione di un campione è pressoché istantanea, le operazioni da effettuare sono sempre sostanzialmente tre somme e sei moltiplicazioni per campione [7] (figura 5.1). Per ogni secondo di segnale servono circa dieci millesimi di secondo per l'elaborazione.

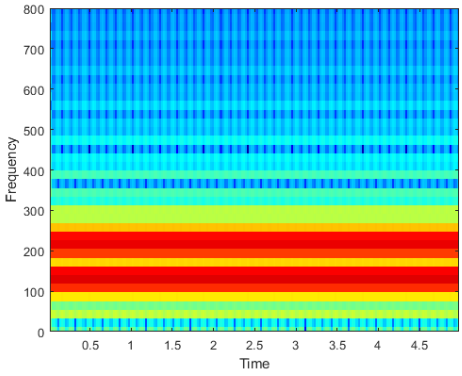
L'utilizzo di un unico puntatore è sufficiente per eseguire le operazioni necessarie a selezionare i giusti campioni del segnale originale da interpolare. Il ridotto onere computazionale, dal punto di vista hardware, permette di utilizzare l'algoritmo contemporaneamente ad altri, per la bassa richiesta di memoria e di calcoli.

La latenza derivata dall'uso dell'algoritmo deriva dal dover interpolare campioni passati per ottenere l'uscita, come già approfondito nella descrizione del metodo.

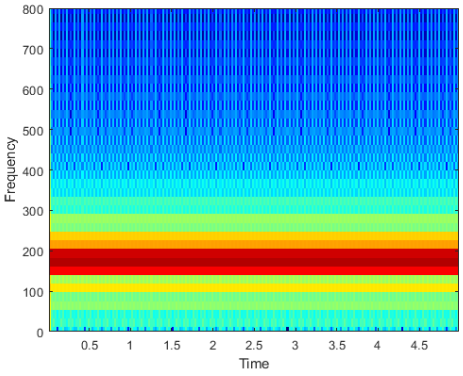
```
y1 = signal[n-ind1]*frac01 + signal[n-ind1+1]*(1-frac01) ;
y2 = signal[n-ind2]*frac02 + signal[n-ind2+1]*(1-frac02) ;

out[n] = y1*WIN1[index] + y2*WIN2[index];
```

Figura 5.1 Operazioni che determinano il tempo di calcolo dell'algoritmo in linguaggio C tramite due delay-line [7]

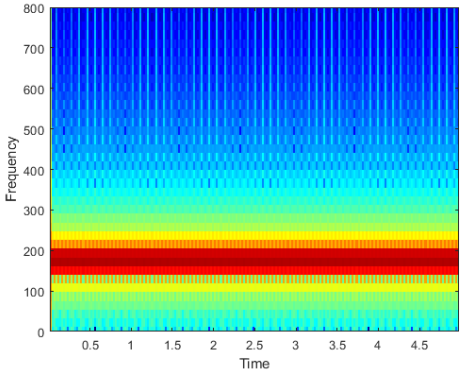
L’implementazione in linguaggio C mostra gli stessi risultati dell’implementazione in Octave. Con una rampa di 1024 campioni la qualità è troppo bassa e il pitch change ratio desiderato non corrisponde a quello ottenuto in uscita. Con rampe dai 2048 campioni in su non si ha questo problema.



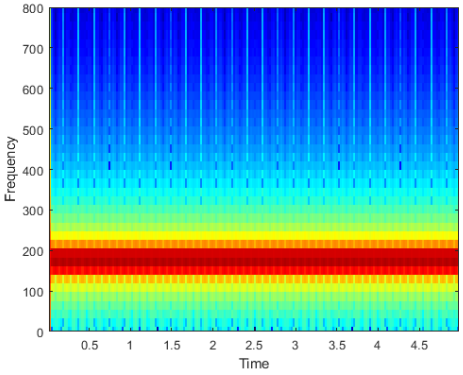
Sottofigura a) Segnale a 220Hz, PCR 0.794, LEN 1024



Sottofigura b) Segnale a 220Hz, PCR 0.794, LEN 2048



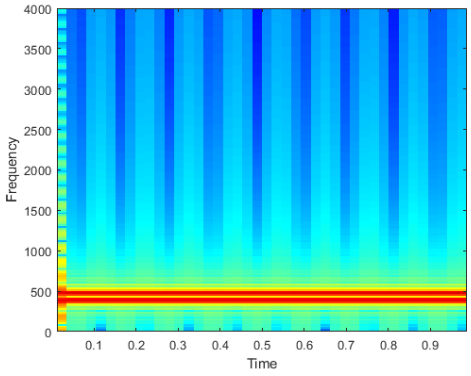
Sottofigura c) Segnale a 220Hz, PCR 0.794, LEN 4096



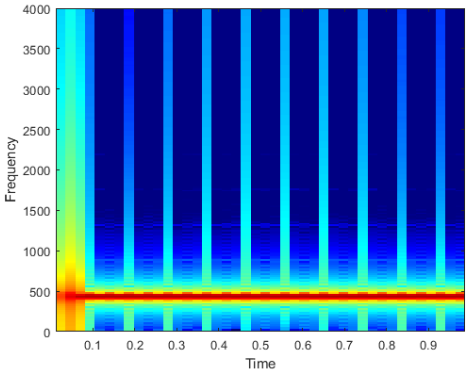
Sottofigura d) Segnale a 220Hz, PCR 0.794, LEN 8192

Figura 5.2 Spettrogrammi dei segnali di uscita di una sinusoide a frequenze 220Hz con pitch change ratio di 0.794 e lunghezza delle rampe variabile

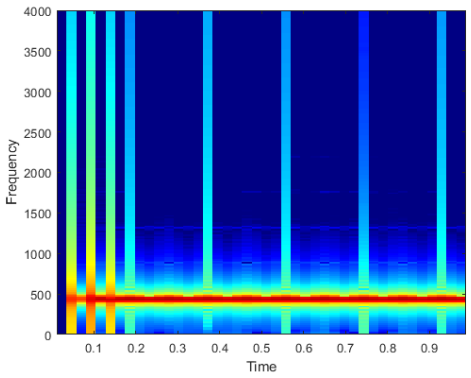
In questi spettrogrammi della porzione iniziale lunga un secondo dell'elaborazione di una sinusoide a 220Hz è chiaro l'effetto delle discontinuità. Si può vedere anche il transitorio iniziale in cui si carica la delay-line con i valori del segnale, utile a capire il concetto di latenza dell'algoritmo.



Sottofigura a) Segnale a 220Hz, PCR 1.999, LEN 1024



Sottofigura b) Segnale a 220Hz, PCR 1.999, LEN 4096



Sottofigura c) Segnale a 220Hz, PCR 1.999, LEN 8192

Figura 5.3 Spettrogrammi dei segnali di uscita di una sinusoide a frequenze 220Hz con pitch change ratio di 1.999 e lunghezza delle rampe variabile

5.3 Algoritmo in linguaggio C per il metodo block-by-block (FFT/IFFT)

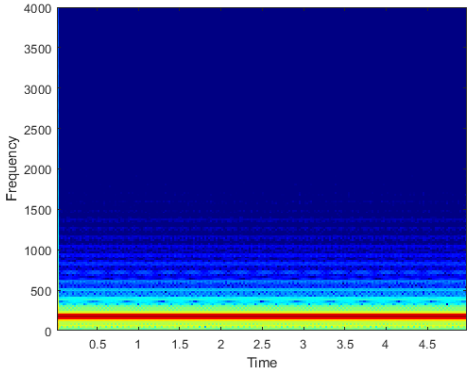
L'implementazione in C dell'algoritmo nel dominio della frequenza ha richiesto l'uso di librerie già realizzate per l'esecuzione della FFT e della IFFT.

Anch'esso mostra un'assoluta corrispondenza all'implementazione in Octave, nonostante la gestione dell'elaborazione sia diversa a causa della diversa gestione dei vettori per gli ovvi motivi delle differenze fra linguaggi [7]. Alcune funzioni che erano già presenti in Octave e che erano state utilizzate per l'algoritmo sono state create in linguaggio C perché non presenti, in modo da ottenere risultati aderenti a quelli già trattati e mostrati dell'implementazione Octave.

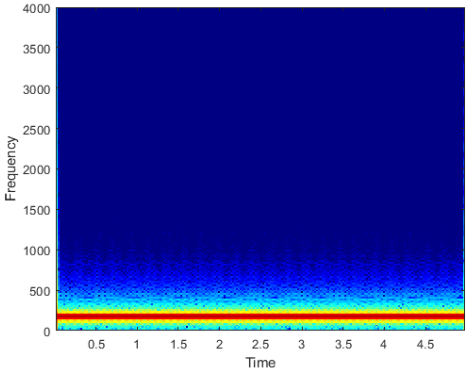
L'elaborazione di ogni campione segue comunque gli stessi esatti passi e le differenze ottenute in uscita sono per questo quasi nulle, dell'ordine di 10^{-5}. Il tempo necessario ad elaborare un secondo di segnale è di circa cento millesimi di secondo. La latenza deriva dall'attesa dell'ultimo campione di ogni finestra.

L'unico parametro disponibile alle modifiche dell'utente dell'effetto è quello della variazione del pitch, da definire in numero di semitoni. Questo è un vantaggio per la gestione dell'algoritmo e la sua scrittura nel codice API della scheda. Infatti, come verrà spiegato nella successiva sezione dedicata alla scheda "Audiolino Brick", il codice scritto in linguaggio C deve essere leggermente modificato in questo metodo per aderire alle caratteristiche di elaborazione della scheda. Lasciare fissa la lunghezza delle finestre permette di semplificare questo passaggio e di non preoccuparsi di questo parametro.

Il risultato è lo stesso ottenuto tramite Octave. Con finestre di 2048 campioni la qualità è ottima, ma si possono avere problemi di latenza, con 1024 campioni si ha un ottimo trade-off.



Sottofigura a) Segnale a 220Hz, PCR 0.794, LEN 1024, HOP 256



Sottofigura b) Segnale a 220Hz, PCR 0.794, LEN 2048, HOP 512

Figura 5.4 Spettrogrammi dei segnali di uscita di una sinusoide a frequenze 220Hz con pitch change ratio di 0.794, lunghezza delle finestre variabile e hop size pari al 25% della finestra

5.4 Audiolino board

“Audiolino brick” è una scheda DSP con audio stereo, che utilizza un microprocessore ARM Cortex a floating-point e dei convertitori ADC/DAC a 48kHz. Ogni algoritmo può girare solo singolarmente, ma è offerta la possibilità di una connessione in serie con altre schede “Audiolino brick”, così da poter realizzare un sistema multi-effetto. Il trasferimento tra le schede avviene in digitale, così le conversioni analogico-digitale e digitale-analogico avvengono solamente una volta ciascuna.

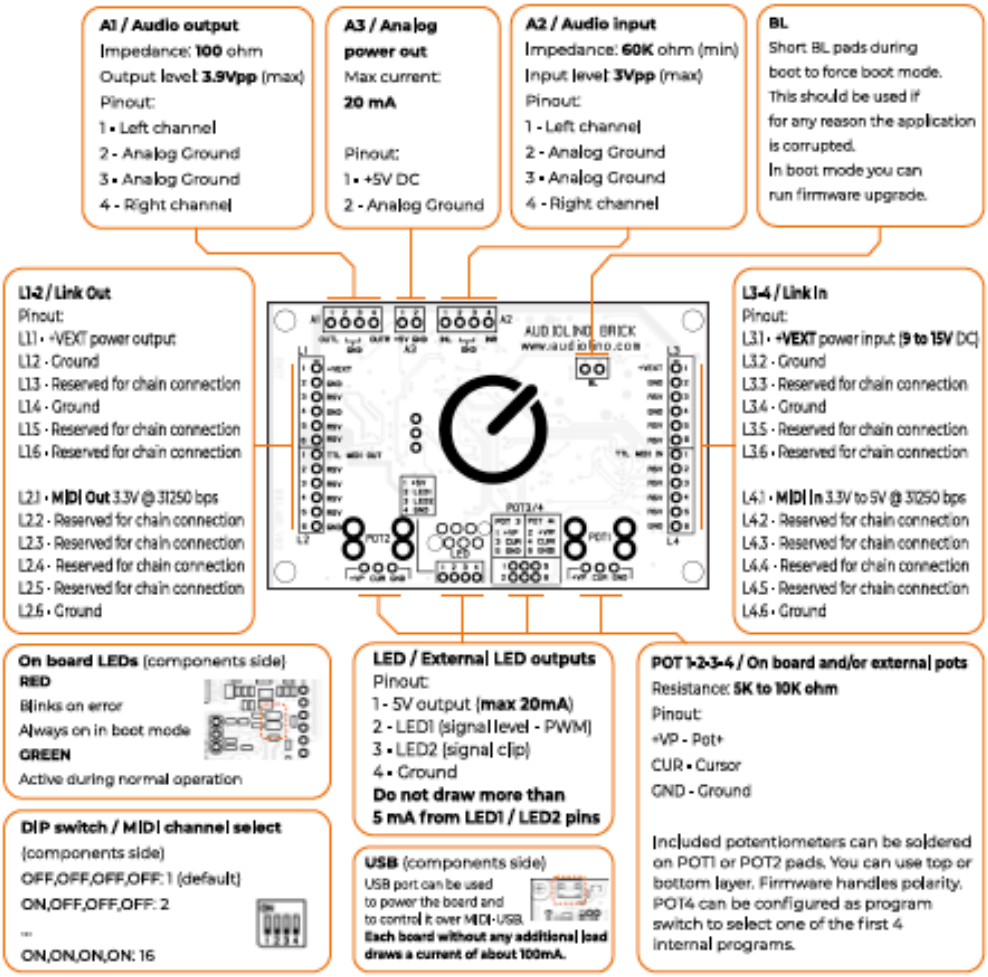


Figura 5.5 Panoramica della scheda “Audiolino Brick”

Gli algoritmi sono caricati e gestiti sulla scheda tramite il software “Audiolino brick editor” che mostra i parametri modificabili dall’utente andando a leggere i file XML che descrivono gli effetti, in accordo con quanto scritto nel codice dove i parametri sono definiti nell’array “param_desc” nel main dell’algoritmo scelto. Un file JSON descrive ogni setup e i suoi algoritmi, i parametri, le connessioni. Quando un setup è caricato viene effettuato il parsing del file JSON che permette di raccogliere le informazioni riguardo la quantità di risorse da allocare, il numero di istanze di cui si ha bisogno, quali sono le connessioni tra buffer e canali. Una volta allocate le risorse vi è una prima fase di inizializzazione dopo la quale gli algoritmi audio e di controllo possono iniziare a lavorare. Quando si deve cambiare un setup viene prima eseguita una fase di terminazione e deallocamento del setup in uso.
Gli algoritmi di elaborazione audio devono rispettare limiti hard real-time per l’uso live e l’elaborazione è eseguita per frame di 64 campioni alla volta.

Le funzioni di controllo hanno invece tempistiche meno stringenti e sono richiamate indicativamente ogni 5-10 millisecondi per aggiornare i parametri.

Per il trasferimento sulla piattaforma embedded è stato prima di tutto deciso quali parametri rendere disponibili all'utente cioè: MUTE, BYPASS, DRY/WET e il parametro chiave dell'algoritmo SHIFT, con limiti fissati a -12 e +12 semitoni.
Poiché la scheda compie un'elaborazione ogni buffer di 64 campioni, gli algoritmi sono adattati a questa necessità. Per il metodo con due delay-line modulate l'algoritmo elabora campione per campione e non necessita di modifiche.
Diversamente dal metodo tramite delay-line, nel metodo tramite FFT e IFFT l'elaborazione è fatta per finestre e questo significa che l'algoritmo deve essere frammentato in parti diverse ognuna da eseguire nel tempo di elaborazione di un buffer da 64 campioni.

Capitolo 6

Conclusioni

Con questo lavoro si è svolta un'analisi della possibile implementazione di diversi algoritmi di pitch shifting per usi real-time sulla piattaforma embedded "Audiolino brick" e si è successivamente sviluppato gli algoritmi in linguaggio C da poter utilizzare per la scrittura del codice API per la programmazione della scheda.

Dopo aver descritto le caratteristiche di un segnale audio che gli algoritmi dovevano andare a modificare e a partire dallo studio dei diversi metodi implementativi possibili sono stati innanzitutto implementati quattro algoritmi sul software Octave basati su tre diversi metodi di realizzazione di un pitch shifter. Su Octave gli algoritmi sono stati testati su segnali diversi quali: toni puri a varie frequenze, segnale vocale, arpeggio di chitarra (tranne l'algoritmo PSOLA non utilizzabile su suoni polifonici), giro di synth-bass.
Dallo studio dei risultati ottenuti sono stati scartati gli algoritmi non adatti all'obiettivo finale da raggiungere e si è invece passati alla creazione di un algoritmo in linguaggio C degli algoritmi utili allo scopo.
Sono stati realizzati due algoritmi. Il primo basato su un'elaborazione nel dominio del tempo, di qualità non elevata e bassissimo costo computazionale. Il secondo basato su un'elaborazione in frequenza, di miglior qualità, ma anche costo computazionale più elevato.

Gli algoritmi in linguaggio C ottenuti possono potenzialmente essere utilizzati anche su sistemi embedded diversi da quello preso in considerazione, ma con caratteristiche di funzionamento similari.

Bibliografia

[1] http://www.studiosoundservice.com/it/education/keynote/corso-tecnolive/acustica3.pdf

[2] D. Arfib, F. Keiler, U. Zölzer, V. Verfaille, J. Bonada. “DAFX: Digital Audio Effects” U. Zölzer, 2011.

[3] F. Fontana “Effetti nel dominio spazio–temporale”, 1999

[4] J. Dattorro “Effect Design - Part 2: Delay-Line Modulation and Chorus”, J. Audio Eng. Soc., pp. 764–88, 1997.

[5] K. Bogdanowicz, R. Belcher “Using Multiple Processors for Real-Time Audio Effects” AES7th International conference

[6] S. Disch, U. Zolzer “Modulation and delay line based digital audio effects”, Proceedings DAFX-99 Digital Audio Effects Workshop, pp. 5–8, 1999

[7] https://github.com/Fra-Bal/Tirocinio, repository degli algoritmi

[8] T. Royer “Pitch-shifting algorithm design and applications in music”, 2019

[9] F. Ahmad “Manipulation of Voice Properties using PSOLA”, Journal of Basic and Applied Engineering Research, 2017

[10] N. Schnell, G. Peeters, S. Lemouton, P. Manoury, X. Rodet “Synthesizing a choir in real-time using Pitch Synchronous Overlap Add (PSOLA)”, 2002

[11] R. E. Crochiere “A Weighted Overlap-Add Method of Short-Time Fourier Analysis/Synthesis”, IEEE Transactions on acoustics, speech, and signal processing, VQL. ASSP-28, NO. 1, FEBRUARY 1980