UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA

TESI DI LAUREA

Sistema di Allineamento Audio in Tempo Reale Basato su Dynamic Time Warping

RELATORE: Prof. Nicola Orio

LAUREANDO: Enrico Guariento
Matr. 621472-IF

ANNO ACCADEMICO 2011/2012

"The world is round and the place which may seem
like the end may also be only the beginning."
Ivy Baker Priest

Ai miei genitori e a mia sorella Elisa.
Ai miei veri amici.
To the people of 1 Highfield West,
the one place abroad I called home.

Un ringraziamento di cuore.

Enrico

Sommario

L'allineamento audio è sempre stato un argomento di grande interesse nell'ambiente del *Music Information Retrieval* (MIR); infatti, la possibilità di allineare diverse sorgenti audio, o un estratto musicale con il relativo spartito, offrono applicazioni di notevole importanza. Alcuni esempi sono: sistemi di accompagnamento automatico in tempo reale, confronto di esecuzioni diverse dello stesso brano e identificazione automatica della musica. In questa tesi, si studierà un'applicazione pratica dell'allineamento che sfrutta la programmazione dinamica, in particolare il *Dynamic Time Warping* (DTW).
L'idea di base è quella di creare un software che sia in grado di allineare l'audio di un film con un file di riferimento, allo scopo di mostrare sullo schermo i sottotitoli relativi al film in modo sincrono o veicolare contenuti alternativi (audio originale, commenti dei registi). La destinazione finale di questo software saranno i dispositivi mobili, ma ai fini di questa tesi ci si limiterà ad una versione per dispositivi fissi (desktop computer e computer portatili).
In una prima fase, il sistema sarà implementato nel modo più semplice: si utilizzerà la versione standard del DTW ed il segnale audio sarà rappresentato attraverso il modulo della *Discrete Fourier Transform* (DFT), che è una rappresentazione completa ma onerosa dal punto di vista della occupazione di memoria. Successivamente, si tenterà di migliorarne le prestazioni, sia in termini di velocità che di efficienza, apportando alcune modifiche all'algoritmo del DTW e alla rappresentazione del segnale audio. Infine saranno presentati alcuni dati sull'affidabilità e usabilità del sistema, ottenuti attraverso un processo di sperimentazione sistematica.
Per concludere poi, verranno forniti alcuni spunti per proseguire questo progetto.

Abstract

The alignment of audio sequences has always been a topic of great interest in the field of the *Music Information Retrieval* (MIR); indeed, the possibility to align different audio sources, or to align a piece of music with the corresponding score, leads to very relevant applications. For example: a real time automatic accompaniment system, the comparison of different executions of the same piece of music and the automatic identification of music. For the purpose of this thesis, we will study a practical application of the audio alignment which is based on the *Dynamic Time Warping* (DTW).

The main idea is to create a software that has to be able to align the sound of a movie, with a reference file, in order to synchronously show on the screen the subtitles of that particular movie, or to deliver alternative contents (original audio, comments of the director). The final destination of this software will be the portable devices but for now, we will just focus on the version for desktop computers and laptops.

In the first phase, we will implement the simplest system: we will use the standard version of the DTW and the audio signal will be represented by the module of the *Discrete Fourier Transform* (DFT), which is a precise representation but it's also very expensive from the point of view of memory consumption. Then, we will try to improve this system in both speed and efficiency, by making a few changes to the DTW algorithm and to the representation of the signal. Next we will present a few data concerning the reliability and the usability of the system, which have been collected through a process of systematic experimentation.

In the end we will express some ideas to continue this project.

Indice

Elenco delle figure

Acronimi

Capitolo 1

Introduzione

Negli ultimi due decenni, la quantità di dati multimediali esistenti è aumentata esponenzialmente, così come la facilità di accesso a questi dati anche da parte di utenti non esperti. Questo grazie all'enorme sviluppo di cui è stata protagonista l'*Information Technology* (IT) e al cambio di mentalità che ha operato nella società contemporanea.

1.1 Riconoscimento di File Multimediali

Tra le esigenze più comuni ci sono quelle di catalogare, etichettare e soprattutto riconoscere i dati con i quali ognuno di noi viene a contatto nella vita quotidiana. In pratica, si parla di associare, organizzare e ricercare dei metadati, che forniscono informazioni specifiche riguardanti un file multimediale. Per assecondare queste richieste, sono state sviluppate tecniche sempre più avanzate per l'estrazione di informazioni utili da grandi moli di dati. Di questi e altri problemi si occupa l'area di ricerca chiamata *Information Retrieval* (IR); in particolare nel campo della musica, si parla di *Music Information Retrieval* (MIR). Alcune applicazioni in ambito musicale/multimediale sono:

- Organizzazione di biblioteche musicali: ovvero recuperare informazioni mancanti da file audio, come titolo, artista, album, copertina. Attual-

mente ci sono diversi programmi che eseguono questa operazione, tra cui iTunes® e SongGenie®.

- Tecnologie di filtraggio per file sharing: ovvero riconoscere (e conseguentemente bloccare) la condivisione di file protetti da diritti d'autore nelle reti *peer-to-peer* (P2P) come è avvenuto per Napster.

- Riconoscimento di musica tramite ascolto: programmi che acquisiscono un breve estratto di una canzone e restituiscono i meta-dati associati come titolo, artista, album (vedi Figura 1.1). Particolarmente utile per dispositivi portatili, questa tecnologia è sfruttata ad esempio da Shazam®.

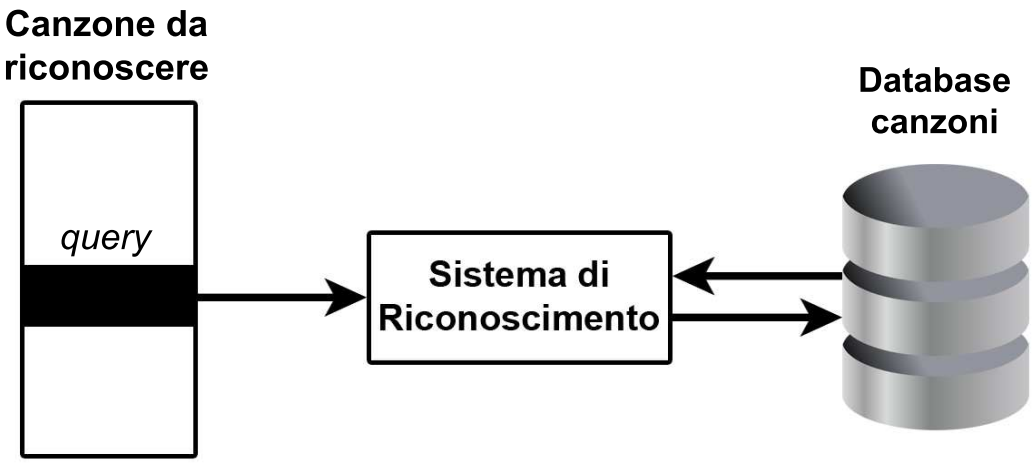

Figura 1.1: Esempio del processo di riconoscimento di un brano musicale. Il Sistema di Riconoscimento riceve in ingresso un piccolo estratto del brano da cercare (*query*) e poi esegue la ricerca in un database dove sono contenuti tutti i brani e i meta-dati ad essi associati.

1.2 Allineamento di File Multimediali

Un'altra situazione che è stata ampiamente esplorata in ambito scientifico, è quella dell'allineamento di file audio.

L'applicazione in ambito musicale è solo una delle molte possibilità che questo campo ha da offrire, infatti l'allineamento di sequenze è molto popolare in genetica, biologia molecolare e nel riconoscimento vocale. Nell'area del MIR l'allineamento è eseguito utilizzando le seguenti tecniche:

1. *Dynamic programming*, attraverso l'uso, per esempio, del *Dynamic Time Warping* (DTW);

2. *Hidden Markov Model* (HMM), un metodo che crea un modello statistico allo scopo di allineare le due sequenze.

Alcune letture interessanti che descrivono il lavoro svolto finora a riguardo sono [1], [2] e [3].
Tornando alla musica, la necessità di allineare file audio è meno comune rispetto al riconoscimento ed è raro che un utente "medio" ne abbia bisogno (vedi Figura 1.2).
Tuttavia, un appassionato di musica, o uno studente di conservatorio, potrebbero essere interessati ad ascoltare un particolare passaggio in versioni differenti dello stesso brano musicale. Per fare ciò, dovrebbero scorrere manualmente i brani fino ad arrivare alla porzione di interesse. Se invece i brani fossero allineati (e propriamente indicizzati), si potrebbe saltare subito al pezzo di interesse, senza bisogno di una scansione sequenziale. Una situazione analoga si presenterebbe se volessimo allineare le note di uno spartito con un'esecuzione dello stesso in tempo reale. Questo potrebbe essere utile ad uno studente di musica per migliorare l'esecuzione di quel particolare brano (ad esempio al pianoforte).
Questa tesi tratta il problema dell'allineamento, per cui verrà in particolare utilizzata la prima delle due tecniche citate (programmazione dinamica). A prima vista sembrerebbe un argomento che ha applicazioni estremamente specifiche; in realtà, quella che si andrà a studiare nei capitoli successivi è una situazione che ha interessanti sviluppi pratici.

1.3 Applicazione

L'idea nasce dal desiderio di rendere più piacevole l'esperienza che una persona non udente ha guardando un film. E' ovvio che se il film non è corredato di alcun sottotitolo, o comunque di sottotitoli mal costruiti, la persona sorda ha scarso interesse nel guardarlo.
Per alcuni film trasmessi in TV, è disponibile il servizio televideo che permette di aggiungere i sottotitoli, ma questo non è disponibile per tutti i film. Per i film proiettati al cinema invece, non c'è ancora soluzione a questo problema. L'applicazione di un sistema di allineamento in tempo reale trattata in questa tesi è la seguente: creare un software che permetta ai non udenti di avere a portata di mano i sottotitoli per un dato film che decidono di vedere.

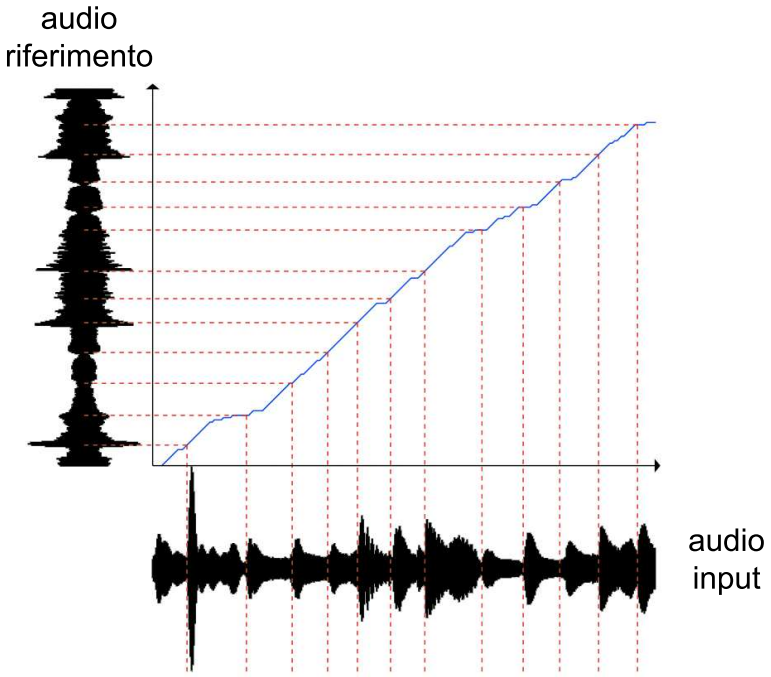

Figura 1.2: Esempio del processo di allineamento di un brano musicale con un brano di riferimento (presumibilmente lo stesso). Questa immagine è stata riadattata da [4].

Si immagini il seguente scenario: una persona sorda è seduta in una

poltrona del cinema, il film sta per cominciare; estrae il telefonino dalla tasca e lancia un'applicazione. Man mano che il film procede, l'applicazione acquisisce l'audio tramite il microfono del telefonino e mostra sullo schermo i sottotitoli in tempo reale. Stessa cosa vale se la persona in questione è seduta nel divano di casa.
Il software in questione potrebbe essere usato anche da chi ha il desiderio di leggere i sottotitoli in una lingua diversa da quella del film. Può capitare ad esempio di trovarsi in un paese straniero e di voler guardare un film, essendo però impossibilitati a causa della lingua.

1.4 Obiettivi

Il fine di questo progetto è quello di compiere un primo passo nella realizzazione di quanto appena descritto. Trattandosi di un processo sperimentale, si partirà prima dalla situazione ideale, la più semplice possibile. All'inizio ci si limiterà ad un'applicazione per dispositivi fissi (desktop computer e portatili) e si applicheranno le tecniche base per risolvere il problema dell'allineamento, senza alcuna ottimizzazione. A quel punto si passerà ad una fase di sperimentazione per studiare il comportamento del sistema. Tale processo sarà anche utile per capire quali sono i punti deboli ed individuare le situazioni più sfavorevoli. Successivamente, si studieranno delle possibili modifiche per migliorare il sistema, considerando anche la sua applicazione finale, ovvero i dispositivi mobili.

1.5 Struttura della Tesi

- Capitolo 2: concetti utili ad introdurre il lavoro svolto.
- Capitolo 3: presentazione e studio del sistema di allineamento in una versione iniziale non ottimizzata.
- Capitolo 4: miglioramento del sistema attraverso modifiche mirate.

- Capitolo 5: conclusioni e sviluppi futuri.

Capitolo 2

Background

In questo capitolo verranno presentati alcuni concetti che sono alla base del lavoro svolto in questa tesi.

2.1 Rappresentazione del Segnale Audio

Il primo passo da compiere è quello di trasformare il segnale audio percepito dall'orecchio umano, in qualcosa che si possa studiare analiticamente.
Come è noto, un generico segnale trasporta informazione, che può essere musica, video, immagini ecc... La fase di elaborazione del segnale, o *Digital Signal Processing* (DSP), ha lo scopo di estrarre queste informazioni (per una lettura approfondita si consiglia [5]). Le tecniche utilizzate per portare a termine questa operazione dipendono dal tipo di segnale e dalla natura delle informazioni contenute al suo interno. In parole povere, si punta ad ottenere una rappresentazione matematica del segnale, che sia compatta e il più possibile invariante rispetto alle distorsioni. Questa rappresentazione può essere costruita con variabili appartenenti al dominio originale (a cui appartiene anche il segnale stesso), oppure con variabili appartenenti ad un dominio diverso.
Quando pensiamo ad un segnale audio, una canzone ad esempio, ci viene naturale immaginarlo come una sequenza temporale, nella quale i suoni si

susseguono in ordine (dominio del tempo). La stessa canzone però può essere anche rappresentata nel dominio della frequenza, quello che viene chiamato spettro del segnale. Lo scopo di questa rappresentazione è quello di mettere in evidenza la distribuzione delle componenti del segnale nelle diverse frequenze. Per ottenere questo risultato si usa la *Discrete Fourier Transform* (DFT). Non ha senso però calcolare la DFT su tutto il segnale perchè questo è tempo-variante, ovvero le sue componenti cambiano frequenza continuamente e non sarebbe di alcuna utilità considerarlo tutto insieme. Quello che si fa invece è suddividere il segnale in "finestre" (o *frame*) e poi calcolare la DFT in modo indipendente su ciascun *frame*, in modo da isolare le componenti del segnale in un intervallo di tempo molto ristretto (tempo-invarianza). Così facendo si ottengono delle "fotografie" che descrivono precisamente (in frequenza) degli intervalli temporali limitati, le quali possono essere facilmente confrontate tra loro.

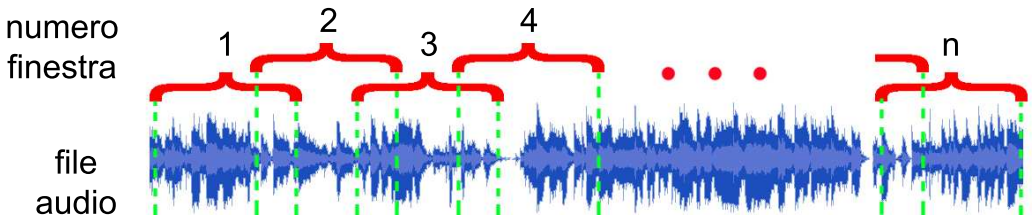

Figura 2.1: Esempio di come viene suddiviso un file audio "a finestre" per calcolare la trasformata di Fourier.

In Figura 2.1 si può vedere il procedimento con cui questo avviene; i *frame* sono sovrapposti per permettere una miglior precisione quando si passa al confronto di diversi segnali audio. Per il corretto svolgimento di questa operazione è necessario stabilire alcuni parametri fondamentali:

- la lunghezza della finestra (*window length*);
- l'ampiezza del salto tra due finestre consecutive (*hop size*).

L'unità di misura di questi due parametri è il *campione*. Per essere elaborati da un computer i segnali devono essere campionati, perciò i parametri assu-

meranno solo valori interi. Questi valori influenzano direttamente la qualità e la compattezza della rappresentazione. Se ad esempio decidessimo di porre *hop size* uguale a *window length*, si otterrebbero meno finestre perchè non ci sarebbe nessuna sovrapposizione, ma si avrebbero notevoli difficoltà nella fase di riconoscimento se i due segnali non fossero perfettamente sincroni (i *frame* risulterebbero tutti sfasati). Un compromesso è quello di porre *hop size* uguale alla metà di *window length*. Per quanto riguarda la lunghezza della finestra, si ha un *trade-off* tra la qualità della rappresentazione nel tempo e quella nella frequenza. Se si scegliesse una finestra molto ampia, la rappresentazione in frequenza sarebbe molto precisa mentre quella nel tempo grossolana e viceversa.
Conseguentemente alla trasformazione del segnale, dal dominio del tempo a quello della frequenza, deve seguire anche un cambio dell'unità di misura utilizzata per descrivere le grandezze appartenenti ai due domini. Nel dominio del tempo l'unità di misura che si utilizza è il campione, mentre nel dominio della frequenza si userà il *bin*. La relazione che lega i *bin* con la frequenza è la seguente:

$$b = \frac{f \cdot w}{f_c}$$

dove $b \in [1, winlen/2]$ è l'indice del *bin* all'interno della finestra, f è la frequenza da trasformare espressa in Hertz, w è la lunghezza della finestra di analisi e f_c è la frequenza di campionamento in Hertz.
Mentre la relazione che lega i *frame* al tempo è la seguente:

$$f = \frac{t \cdot f_c}{h}$$

dove f è l'indice del *frame* all'interno del segnale, t è il tempo da trasformare espresso in secondi, f_c è la frequenza di campionamento in Hertz e h è l'ampiezza del salto tra due finestre consecutive.
C'è una cosa molto importante da notare a questo punto. Quando si esegue la DFT di una finestra, si ottengono tanti *bin* quanti sono i campioni della finestra stessa, ovvero avviene una mappatura lineare tra i valori prodotti dalla DFT e i *bin*. Dato che i frame sono molti, la rappresentazione del

segnale audio sarebbe sicuramente di grosse dimensioni e si avrebbero delle difficoltà nell'elaborare tutti i dati estratti. Esiste un modo per ridurre la dimensione che è basato su una semplice osservazione. Il sistema uditivo umano è noto percepire meglio suoni in un *range* medio di frequenze (intorno a 1 kHz), mentre tende ad attenuare suoni ad alte frequenze. In teoria l'orecchio è in grado di sentire suoni fino a 20 kHz, ma questi devono essere ben isolati per essere udibili e comunque questa soglia si abbassa notevolmente con l'età. In generale perciò, è molto difficile trovare nell'audio di un film delle frequenze superiori ai 10 kHz, infatti non avrebbe neanche senso per i produttori introdurre dei suoni nel film che il pubblico non può sentire; inoltre le componenti più forti sono concentrate nella fascia 100-3000 Hz. Grazie a questa osservazione si capisce che si può ridurre notevolmente il numero di campioni da estrarre con la DFT senza perdere efficacia.
Ecco un esempio pratico: un file audio viene campionato ad una frequenza di 44100 Hz con una finestra lunga 1024 campioni (quindi per ogni finestra la DFT produrrebbe 1024 valori). Se considero solo la fascia 100-3000 Hz, per ogni finestra si hanno:

$$\#bin = \frac{3000 \cdot 1024}{44100} - \frac{100 \cdot 1024}{44100} = 68 \ bin$$

Si è ottenuta quindi una riduzione del 94% sulla dimensione di una singola finestra. Nonostante questo sia già un ottimo risultato, si potrebbe fare di meglio.
Come già accennato l'orecchio umano non percepisce tutti i suoni con la stessa intensità, ma la percezione dipende dalla frequenza a cui questi sono emessi. Sono state studiate delle scale apposite per descrivere questo comportamento, come la scala Mel e la scala Bark. La prima è composta da 13 bande critiche, la cui larghezza aumenta in modo logaritmico. La seconda invece è composta da 24 bande critiche, la cui larghezza è (quasi) costante fino alla frequenza di 1 kHz e poi aumenta in modo logaritmico (come per la scala Mel). L'idea è quella di utilizzare un numero di *bin* pari a quello di bande critiche, assegnando a ciascuno di essi la somma delle componenti (estratte con la DFT) corrispondenti ad un certo intervallo di frequenza.

Ad esempio utilizzando la scala Mel in Figura 2.2, nel primo *bin* si trova la somma dei coefficienti della DFT corrispondenti alla fascia 20-160 Hz.

Hz	20	160	394	670	1000	1420	1900	2450	3120	4000	5100	6600	9000	14000
bin	1	2	3	4	5	6	7	8	9	10	11	12	13	

Figura 2.2: Scala Mel e mappatura su *bin*.

Ad ogni modo, non è imperativo utilizzare una delle scale menzionate sopra, né qualsiasi altra scala sviluppata da studi di psicoacustica. Queste servono solo per dare un'idea di partenza su come raggiungere una rappresentazione del segnale più compatta e robusta. Infatti, combinando i due metodi utilizzati finora, si potrebbe creare una rappresentazione adatta alle esigenze di questo progetto. Ad esempio, si potrebbe pensare di applicare un mappatura logaritmica sull'intervallo di frequenze ristretto 100-3000 Hz. Alcune applicazioni di questo metodo si possono trovare in [6] e [7].

2.2 Dynamic Time Warping

Come descritto in [7], il DTW è una tecnica per allineare sequenze ordinate (le relazioni di precedenza devono essere rispettate) che è ben nota nell'ambito del riconoscimento vocale fin dagli anni '70 (vedi [8]).
Il DTW allinea due sequenze temporali $U = u_1, \ldots, u_m$ e $V = v_1, \ldots, v_n$, trovando un cammino di costo minimo $W = w_1, \ldots, w_l$, dove ogni w_k è una coppia ordinata (i_k, j_k), tale che se $(i, j) \in W$ significa che i punti u_i e v_j sono allineati (vedi Figura 2.3a). Nell'applicazione di interesse in questa tesi, i punti che vengono allineati con il DTW sono i *frame* appartenenti a due diversi file audio (il file di riferimento e il file acquisito in tempo reale).
L'allineamento è valutato utilizzando una funzione di costo locale $d_{U,V}(i, j)$ (talvolta chiamata anche LDM: *local distance matrix*), generalmente rappresentata da una matrice $m \times n$, che assegna un costo per l'allineamento di ogni coppia (u_i, v_j). Questo costo è nullo se l'allineamento è perfetto, altri-

menti è un valore positivo.
Una delle funzioni più utilizzate in letteratura è la distanza coseno:

$$d(u,v) = \frac{\sum_{k=1}^{S} u_k v_k}{\sqrt{\sum_{k=1}^{S} u_k^2 \cdot \sum_{k=1}^{S} v_k^2}}$$

dove u e v sono due generici *frame* da allineare e S è il numero di *bin* utilizzati (vedi Sezione 2.1). Il lato positivo di questa distanza è che non considera il modulo dei vettori che confronta perciò, in termini pratici, il volume dei segnali audio non interferisce con la misura di distanza.
Il costo di tutto il percorso è la somma dei costi locali lungo il percorso stesso (talvolta chiamato anche ADM: *accumulated distance matrix*):

$$D(W) = \sum_{k=1}^{l} d_{U,V}(i_k, j_k)$$

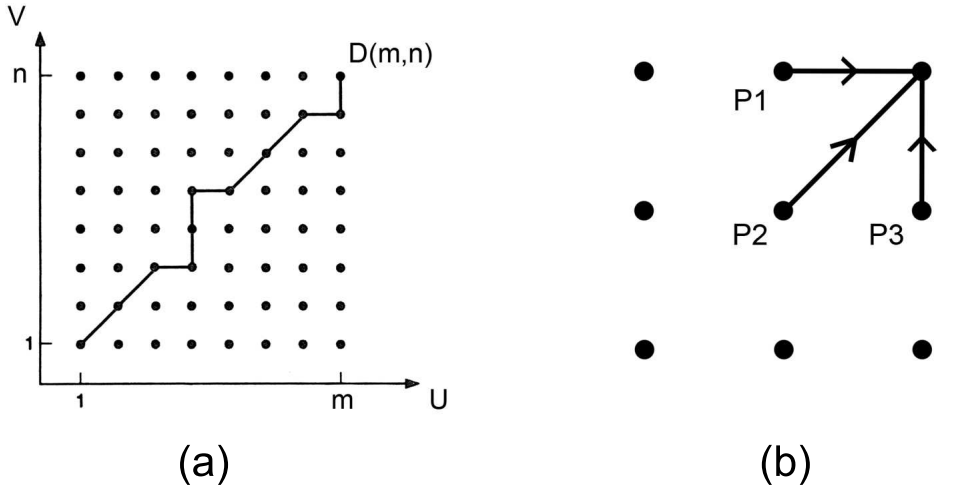

Figura 2.3: (a) Esempio di allineamento di due sequenze U e V. (b) Punti utilizzati per il calcolo del costo di una nuova coppia (u_i, v_j).

Solitamente si applicano dei vincoli al percorso W con lo scopo di limitare i calcoli da eseguire. I vincoli più banali sono:

- Il percorso è limitato dalla fine delle sequenze da allineare;
- Il percorso è monotono (questo vincolo può essere rilassato in alcune applicazioni);

- Il percorso è continuo.

Un altro vincolo ad esempio potrebbe essere quello usato in [9], per il quale il percorso è costretto a risiedere entro una certa distanza dalla diagonale (generalmente una frazione della lunghezza totale delle sequenze).
In ogni caso, il percorso di costo minimo può essere calcolato con complessità $O(n^2)$ dalla programmazione dinamica, usando la formula (vedi Figura 2.3b):

$$D(i,j) = d(i,j) + min \begin{cases} D(i,j-1) & P3 \\ D(i-1,j) & P1 \\ D(i-1,j-1) & P2 \end{cases}$$

dove $D(i,j)$ è il costo del percorso più breve da $(1,1)$ a (i,j) e $D(1,1) = d(1,1)$. I punti utilizzati per calcolare $D(i,j)$, ovvero $\{(i,j-1),(i-1,j),(i-1,j-1)\}$, sono anche chiamati vicini o *neighbours*. Il percorso può essere poi tracciato a ritroso a partire da $D(m,n)$. Un'altra cosa utile da fare, è quella di dare peso diverso ai tre casi descritti dalla formula. Solitamente è preferibile favorire il percorso che va lungo la diagonale (cioè utilizzando P2), che è quello più probabile in una situazione in cui le due sequenze avanzano con la stessa velocità.
Il problema di questo approccio però è la complessità computazionale: troppo alta per un'applicazione che deve funzionare in tempo reale. Perciò, per cercare di ridurre il numero di calcoli che l'algoritmo compie, si è introdotto un vincolo sul percorso.

Sia data una coppia di *frame* allineati (u_{i-1}, v_{j-1}), dove $u_{i-1} \in U$ è la sequenza da allineare e $v_{j-1} \in V$ è la sequenza di riferimento. Il passo successivo è quello di allineare il *frame* u_i. Invece di considerare tutti i *frame* $v \in V$, si considerano solamente i *frame* $v_{j-\Delta_1}, \ldots, v_{j+\Delta_2}$. Δ_1 e Δ_2 sono due parametri (costanti) decisi a priori che delimitano la regione in cui l'algoritmo calcola le distanze locali (vedi Figura 2.4). Essendo questa finestra di dimensione costante (non dipendente dalla lunghezza delle sequenze), la complessità dell'algoritmo si riduce a $O(n)$.

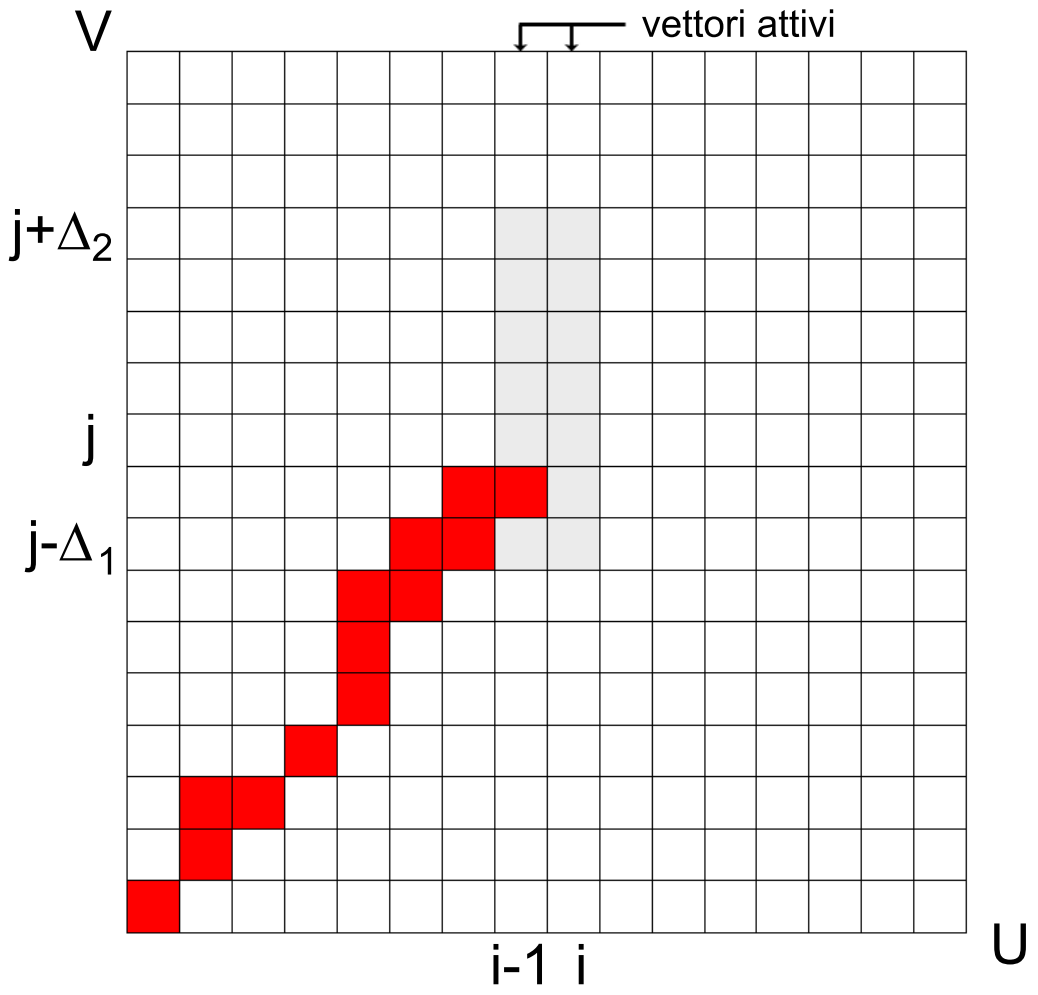

Figura 2.4: Esempio di allineamento di due *frame*. Le celle rosse indicano il percorso calcolato, mentre le celle grigie indicano per quali *frame* si calcolano i costi.

2.3 Aspetti Tecnici

Le Sezioni 2.1 e 2.2 hanno coperto gli aspetti teorici riguardanti questo progetto. In questa sezione invece, verranno affrontati gli aspetti pratici.
A questo punto risulta chiaro che per implementare quanto descritto finora devono essere effettuate due operazioni, che sono tanto fondamentali quanto complesse:

1. calcolo delle trasformate di Fourier;

2. gestione del flusso audio.

Oltre ad essere complesse, queste operazioni sono anche costose dal punto di vista computazionale; inoltre il fatto che la destinazione finale del software da creare sia un dispositivo mobile, incrementa l'impatto dei problemi di prestazioni. Date le premesse, sono state utilizzate due librerie esterne che sono in grado di svolgere questi compiti in maniera molto efficiente.

1. La prima è *Fastest Fourier Transform in the West v3* (FFTW) (vedi [13]), che serve per calcolare la DFT in modo efficiente. Questa libreria è scritta in C ed è in grado di calcolare la DFT di dati reali o complessi, in una o più dimensioni[1].
   L'oggetto chiave implementato da questa libreria è il fftw_plan, che rappresenta il "piano" di esecuzione della trasformata. Questo oggetto deve essere inizializzato come segue:

   ```
   fftw_plan fftw_plan_r2r_1d(int dimensione, double *in, double *out,
                              fftw_r2r_kind tipo, unsigned flags);
   ```

   dove dimensione indica la taglia della trasformata (cioè la grandezza della finestra di analisi), in è un puntatore all'array contenente i dati di input (provenienti dal microfono), out è un puntatore all'array che conterrà le trasformate e gli altri due sono parametri avanzati.

[1] Nel caso in esame si utilizzano trasformate reali in una dimensione.

Dopodichè, non resta che eseguire la trasformata quando necessario (ovvero non appena il microfono fornisce sufficienti dati per riempire una finestra):

```
fftw_execute(plan);
```

2. L'altra libreria utilizzata è RtAudio 4 (vedi [14]). Questo pacchetto è scritto in C++ e fornisce delle *Application Programming Interface* (API) per trattare input e output audio in tempo reale. E' adatto per interagire facilmente con le periferiche hardware di computer con sistemi operativi Linux, Macintosh OS-X e Windows.
   L'oggetto utilizzato per trattare con l'input proveniente dal microfono è di tipo RtAudio. L'operazione più importante che compie è quella di aprire il canale di comunicazione (o *stream*) con la periferica:

```
void openStream ( StreamParameters * outputParameters,
                  StreamParameters * inputParameters,
                  RtAudioFormat      format,
                  unsigned int       sampleRate,
                  unsigned int *     bufferFrames,
                  RtAudioCallback    callback,
                  void *             userData);
```

   dove outputParameters/inputParameters specificano i parametri da utilizzare per aprire lo *stream*, tra i quali è fondamentale il numero di periferica. Se non lo si conosce già, l'indice corretto può essere recuperato attraverso il programma audioprobe.cpp che si trova tra i file della libreria. Altri parametri interessanti sono sampleRate che specifica la frequenza di campionamento, callback che è un puntatore alla funzione a cui vengono forniti i dati di input che arrivano dal microfono (è quella che esegue l'allineamento) e userData, che rappresenta la struttura in cui sono contenuti tutti i dati necessari al funzionamento del programma (per esempio, i valori delle trasformate).

A questo punto, per cominciare la comunicazione si utilizza il seguente comando:

```
void startStream();
```

E' importante notare che le librerie presentate in questa sezione sono state scelte (per ragioni di efficienza e semplicità) per essere impiegate su dispositivi fissi. Perciò in futuro, quando sarà implementata la versione per dispositivi mobili, sarà necessario valutare l'applicabilità di queste librerie e, in caso di esito negativo, procedere con la scelta di librerie analoghe che svolgano le stesse operazioni.

Capitolo 3

Sistema Base di Allineamento

E' stato implementato un primo sistema rudimentale che permette di effettuare quanto descritto nella Sezione 1.3. Tale software è scritto in C++ e permette di visualizzare sullo schermo del proprio computer i sottotitoli relativi ad un film a scelta, mediante la versione base del DTW che è stata descritta nella Sezione 2.2.

3.1 Struttura del Programma

Il sistema in esame è composto da un insieme di moduli software scritti in diversi linguaggi di programmazione. Essendo un software sperimentale, si è voluto costruire ciascuna parte con il linguaggio che fosse il più semplice per portare a compimento la funzione richiesta. In Figura 3.1 è riportato lo schema di funzionamento generale, dove si può vedere come le diverse parti interagiscono tra loro.

3.1.1 Pre-elaborazione

Per il corretto funzionamento del programma, deve essere eseguita una parte di pre-elaborazione prima dell'effettiva esecuzione del programma. Questo processo serve per creare due file che devono essere forniti in input per la successiva fase di elaborazione.

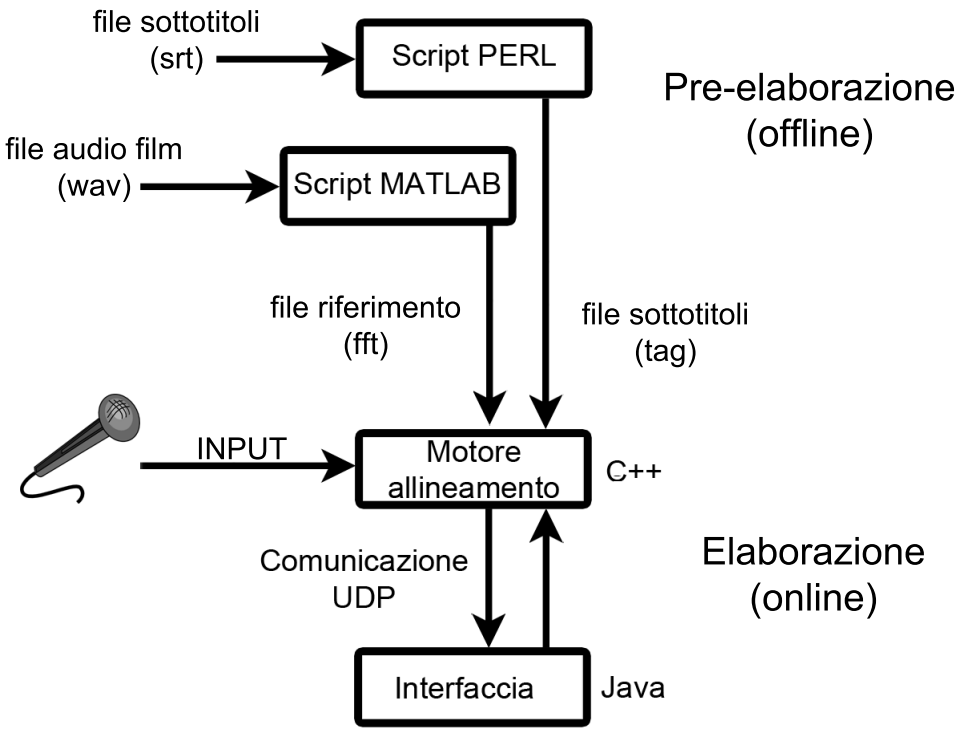

Figura 3.1: Schema di funzionamento del software.

1. **File dei sottotitoli:** è stato creato uno script in linguaggio PERL che riceve in input un file di sottotitoli nel formato standard *srt*. Questo file contiene blocchi di questo tipo:

   7
   00:00:43,575 –> 00:00:45,647
   You know, when you go on like this,
   what you sound like?
   ** riga vuota **

   La prima riga contiene l'indice del sottotitolo (numero sette). La seconda riga delimita il tempo (inizio –> fine) nel quale il sottotitolo deve apparire, con il formato *ore : minuti : secondi , millisecondi*. Le righe successive contengono il sottotitolo.
   Il risultato dopo l'esecuzione dello script è un file in formato *tag*. Anche questo file è composto da un blocco per ogni sottotitolo. Ecco come viene trasformato il blocco nell'esempio appena presentato:

   43575
   You know, when you go on like this, what you sound like?
   45647
   ** riga vuota **

   La prima riga indica il tempo di inizio in millisecondi. La seconda riga contiene il sottotitolo (tutto in una sola linea). La terza riga indica il tempo di fine in millisecondi. L'indice che compare nel file originale non ha importanza perchè l'ordine dei sottotitoli deve essere lo stesso in cui sono stati scritti, perciò questo valore non viene riportato.

2. **File di riferimento:** come si può vedere in Figura 3.1, nella parte di pre-elaborazione c'è uno script in linguaggio MATLAB® che riceve in ingresso un file *wav* che contiene l'intero audio del film.

Dal file audio, devono essere estratte certe "caratteristiche" che siano in grado di rappresentare il segnale. Questi descrittori devono essere il più possibile invarianti rispetto alle distorsioni che il segnale può subire. Un modo per raggiungere lo scopo è quello di considerare lo spettro del segnale, calcolandone quindi la DFT (vedi Sezione 2.1).
Il programma MATLAB® dunque, produce un file in formato *fft* eseguendo le seguenti operazioni:

- Riduce il file da stereo (due canali) a mono (un canale);
- Scrive nel file *fft* i parametri di creazione: frequenza di campionamento (di solito 44100 Hz), lunghezza della finestra di analisi, ampiezza del salto tra due finestre consecutive, numero di trasformate effettuate e numero di punti considerati per il calcolo della trasformata sulle finestre. L'ultimo parametro serve perchè si applica una mappatura lineare con intervallo di frequenze ridotto (vedi Sezione 2.1);
- Scrive nel file *fft* le DFT dei *frame* in modo consecutivo.

E' importante notare che la fase di pre-elaborazione è molto onerosa dal punto di vista di calcolo, in particolare la creazione del file di riferimento, che ha una complessità computazionale di $O(n\ log\ n)$ (essendo n la lunghezza del file audio).
Questo fatto comunque non deve preoccupare, dato che l'intero processo è eseguito prima dell'effettiva esecuzione del programma e non in tempo reale.

3.1.2 Elaborazione

La fase di elaborazione si riferisce all'effettivo allineamento in tempo reale.
L'input che deve essere fornito in ingresso al software è costituito da:

- Numero di periferica di input, generalmente il microfono (vedi Sezione 2.3);

- Numero di periferica di output, generalmente gli altoparlanti (vedi Sezione 2.3);
- File dei sottotitoli in formato *tag*;
- File di riferimento in formato *fft*.

Ci sono due blocchi principali di codice che vengono utilizzati.

1. Un'interfaccia utente (ancora molto primitiva) scritta in Java, che permette la visualizzazione dei sottotitoli (vedi Figura 3.2).
2. La *routine* che esegue le operazioni di allineamento dell'input proveniente dal microfono con il file di riferimento. Questo è il vero e proprio motore ed è scritto in C++ (per ragioni di efficienza).

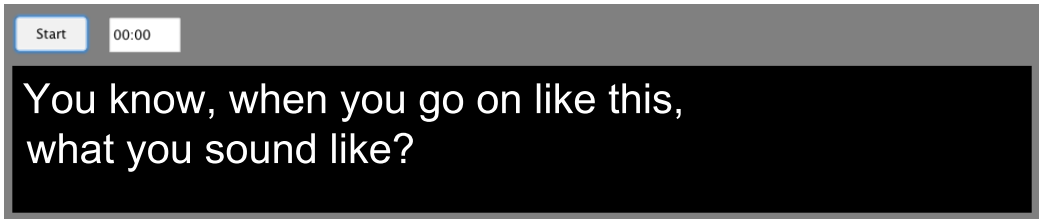

Figura 3.2: Interfaccia grafica del programma.

Queste due sezioni di codice comunicano tra di loro per scambiarsi informazioni (ad esempio il testo dei sottotitoli) via UDP. Questo protocollo è usato solitamente per la comunicazione tra due computer in internet; può essere anche utilizzato però per la comunicazione di due processi sullo stesso computer: normalmente un processo *master* e uno *slave*. In questo caso, l'interfaccia utente è il *master*, perchè è quella che comunica al motore (*slave*) quando e da che punto cominciare l'allineamento.

3.2 Ambiente Sperimentale

L'obiettivo principale di questa sezione è quello di osservare come si comporta il software in diverse situazioni, individuare quali sono i suoi limiti,

quali sono i suoi punti di forza e soprattutto capire quali sono le condizioni operative peggiori. Questo processo ha lo scopo di mettere le basi per un successivo miglioramento del software stesso.
Essendo la prima sperimentazione in assoluto, è stata utilizzata la versione più semplice del DTW, in modo da avere dei risultati di base con cui confrontare poi quelli ottenuti applicando alcune modifiche. Per fare ciò, è stato predisposto un ambiente sperimentale, di seguito definito.

3.2.1 Estratti Audio

I dati di input del programma sono dieci estratti audio, provenienti da cinque film diversi (due per ogni film).
Di seguito sono riportati i dettagli riguardanti gli estratti, ognuno dotato di un codice identificativo univoco che sarà utile in fasi successive per riferirsi ai file.

1. 2 Fast and 2 Furious (lingua inglese)
    - 2f2f_1: estratto dal minuto 1:08 al minuto 18:56;
    - 2f2f_2: estratto dal minuto 45:25 al minuto 62:58;
2. Back to the Future (lingua inglese)
    - bttf_1: estratto dal minuto 2:48 al minuto 19:35;
    - bttf_2: estratto dal minuto 46:15 al minuto 61:32;
3. Forrest Gump (lingua inglese)
    - fg_1: estratto dal minuto 3:09 al minuto 20:02;
    - fg_2: estratto dal minuto 40:13 al minuto 57:29;
4. Mr and Mrs Smith (lingua italiana)
    - mms_1: estratto dal minuto 10:28 al minuto 26:55;

- mms_2: estratto dal minuto 50:58 al minuto 68:00;

5. Pulp Fiction (lingua inglese)

   - pf_1: estratto dal minuto 00:25 al minuto 17:56;
   - pf_2: estratto dal minuto 48:24 al minuto 65:15.

Gli estratti sono stati scelti appositamente per formare un campione di studio variegato, in cui sono presenti molti suoni diversi che si potrebbero trovare in un generico film. Ad esempio: dialoghi serrati con sottofondo musicale, dialoghi serrati senza sottofondo, silenzi, rumori forti, sgommate, sparatorie, musica, ecc...

3.2.2 Parametri

I parametri con cui sono stati eseguiti i test sono i seguenti (vedi Sezioni 2.1 e 2.2 per informazioni dettagliate):

- Lunghezza della finestra di analisi: 1024 campioni;
- Ampiezza dello spostamento tra due finestre consecutive: 512 campioni;
- Frequenza di campionamento: 44100 Hz;
- Per il calcolo di un generico $D(i, j)$ si considerano i *neighbours* $\{(i, j-1),(i-1, j),(i-1, j-1)\}$;
- I pesi applicati (come somma) sono stati di 0,05 per $\{(i-1, j-1)\}$ e 0,1 per $\{(i, j-1),(i-1, j)\}$;
- Estensione della finestra per l'allineamento: 2 secondi prima e 10 secondi dopo un generico (i, j);
- Intervallo di frequenze utilizzato per il calcolo delle DFT: 150-8000 Hz.

3.2.3 Dettagli Operativi

Per la realizzazione di questi test, il software rappresentato in Figura 3.1 è stato leggermente modificato, in modo da rendere la rilevazione dei dati più facile e precisa.
La modifica sostanziale che è stata effettuata, è stata quella di fornire i dati di input da file, contrariamente a quanto avveniva in precedenza, quando i dati erano recuperati tramite le periferiche hardware del computer (microfono o line-in). Questa variazione ha permesso di aumentare la precisione delle misure, in quanto il file di input e quello di riferimento sono processati a partire dallo stesso istante (cosa che risulta virtualmente impossibile utilizzando il microfono, perchè il programma andrebbe fatto partire manualmente).
Un altro dettaglio degno di menzione riguarda il formato dei file di input. Questi sono file binari "grezzi" (formato *raw*) calcolati tramite MATLAB®, a partire dai file audio originali (formato *wav*).

3.2.4 Esperimenti

Prima di arrivare agli esperimenti veri e propri, è necessario definire in che modo verrà misurata l'efficacia del sistema.
L'idea di base è quella di osservare l'allineamento che esegue il programma e misurare ad ogni istante la "distanza" tra il *frame* del file di riferimento e il *frame* del file di input che il software allinea. Per distanza si intende la differenza tra i numeri di sequenza dei *frame* che il sistema allinea ad ogni passo (che risulta nulla se l'allineamento è perfetto).
Dato l'obiettivo di partenza, sono stati pensati diversi scenari che rispecchiano potenziali situazioni reali in cui il programma potrebbe essere utilizzato, per un totale di 150 prove distinte. In dettaglio, per ogni estratto sono state applicate o aggiunte le seguenti "distorsioni":

- Filtro passa basso con frequenza di taglio 4 kHz;
- Riverbero;

- Rumore bianco;
- Rumore rosa;
- Pause (con conseguente allungamento del file di input);
- Tagli (con conseguente accorciamento del file di input);
- Variazione di velocità (e di conseguenza anche di pitch e tempo).

Di seguito saranno presentati i risultati ottenuti dai test sotto forma di grafici, suddivisi per esperimento. Questi grafici rappresentano l'allineamento che esegue il programma, dove l'ottimo è raggiunto da una retta che taglia a metà il piano cartesiano.

Passa Basso

Questa prova è stata pensata in particolare per ricreare la condizione in cui il film sia su un vecchio supporto VHS. Come noto, le videocassette hanno uno spettro limitato di frequenze che si riduce ulteriormente quando si utilizza la modalità *Extended Play* (EP). In quest'ultimo caso, la massima frequenza impressa nel nastro è pari a 4 kHz, perciò questo valore sarà la frequenza di taglio del filtro passa basso che verrà applicato all'audio originale.
Viene riportato solo un grafico, dato che sono tutti (quasi) identici (Figura 3.3). Si può notare che l'allineamento è praticamente perfetto.

Riverbero

Questa prova è stata pensata per ricreare la condizione di ascolto di una sala cinematografica o di un ambiente di grandi dimensioni, portando l'effetto di riverberazione molto al di sopra di quello normalmente presente nelle sale in modo da valutare la robustezza del sistema in condizioni particolarmente svantaggiose. Con l'effetto riverbero è stato aggiunto un eco chiaramente udibile, con ritardo di 30ms.
Anche qui viene riportato solo un grafico, dato che sono tutti (quasi) identici (Figura 3.4). Si può notare che l'allineamento è praticamente perfetto.

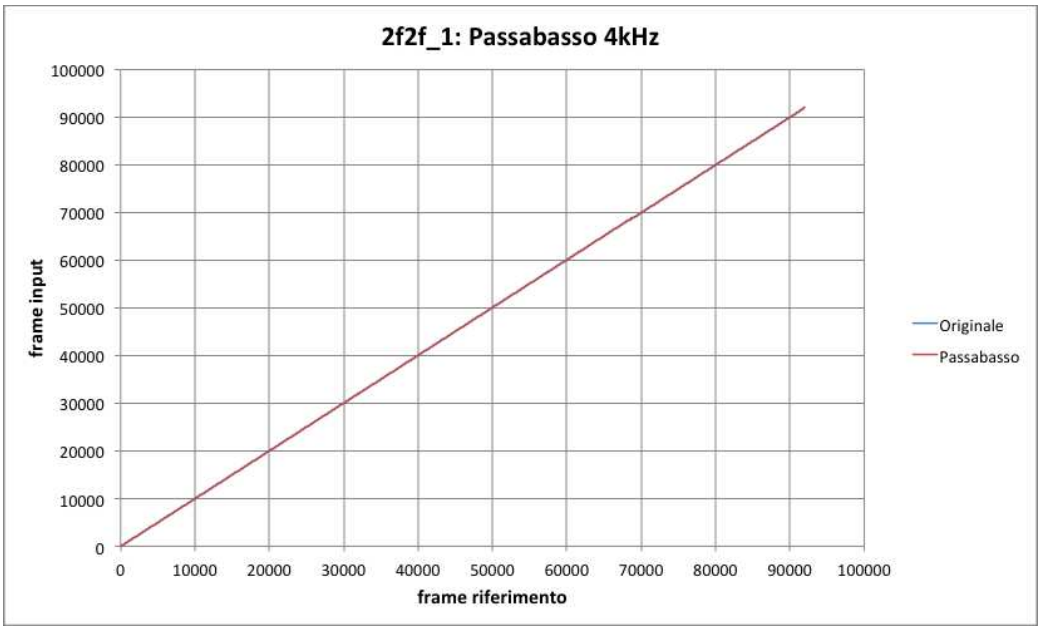

Figura 3.3: Estratto 2f2f_1 a cui è stato applicato un filtro passa basso con frequenza di taglio 4 kHz.

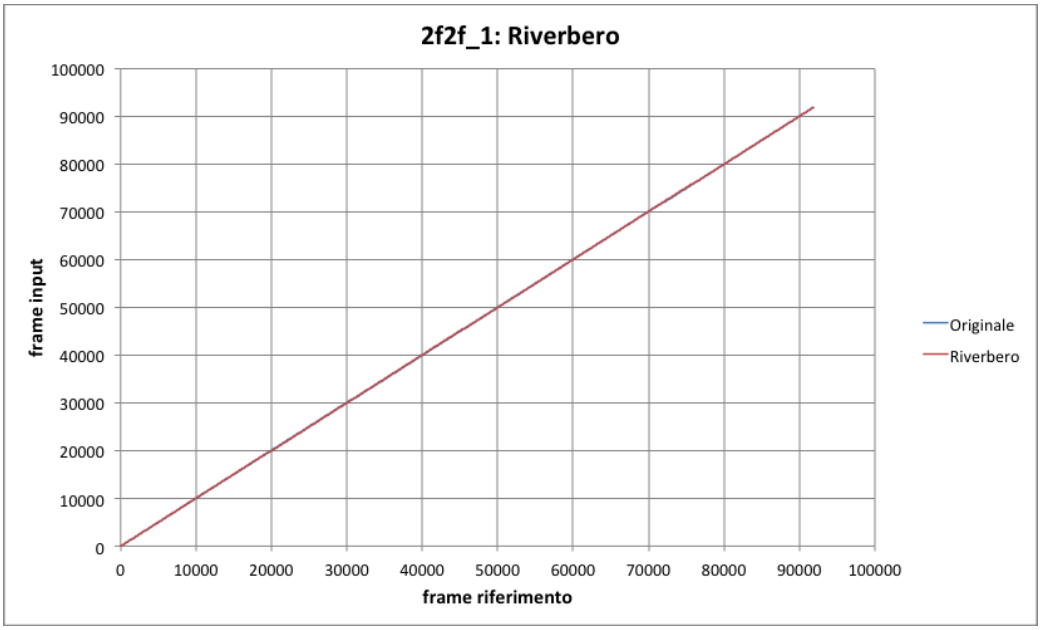

Figura 3.4: Estratto 2f2f_1 a cui è stato applicato un effetto riverbero.

Rumore Bianco

Con questo test si mira a ricreare la situazione più comune che si possa pensare, ovvero un rumore di sottofondo che non fa parte del film, ma è dovuto al particolare ambiente circostante o viene introdotto dall'uso di supporti analogici o da apparecchiature di riproduzione di bassa qualità.
A questo scopo sono state eseguite tre prove per ciascun estratto: a partire da un livello di rumore appena udibile (-39 dB) fino ad un rumore di sottofondo decisamente fastidioso (-27dB), passando per un livello intermedio (-33dB).
I grafici risultanti mostrano che il sistema incontra notevoli difficoltà di allineamento, soprattutto nel caso di rumore più forte (Figure 3.5, 3.6 e 3.7). Infatti, quando l'andamento della "curva" di allineamento non è rettilineo (tende verso l'alto), significa che il sistema fallisce.

Rumore Rosa

Questo test ha un obiettivo analogo al precedente, cambia solo il tipo di rumore. Il rumore rosa ha la caratteristica di avere una potenza maggiore a basse frequenze e si adatta quindi all'orecchio umano. Questo tipo di rumore risulta meno fastidioso e un pò più simile ad un generico rumore che si potrebbe trovare in un ambiente reale o che può essere introdotto da apparecchiature analogiche.
A causa della minore potenza di questo segnale, è stato necessario innalzare i livelli di rumore rispetto a quelli utilizzati per il rumore bianco, in modo che il volume percepito fosse lo stesso. I nuovi livelli sono quindi -26dB, -20dB e -14dB.
Anche in questo caso i risultati evidenziano dei problemi di allineamento (Figure 3.8, 3.9 e 3.10).

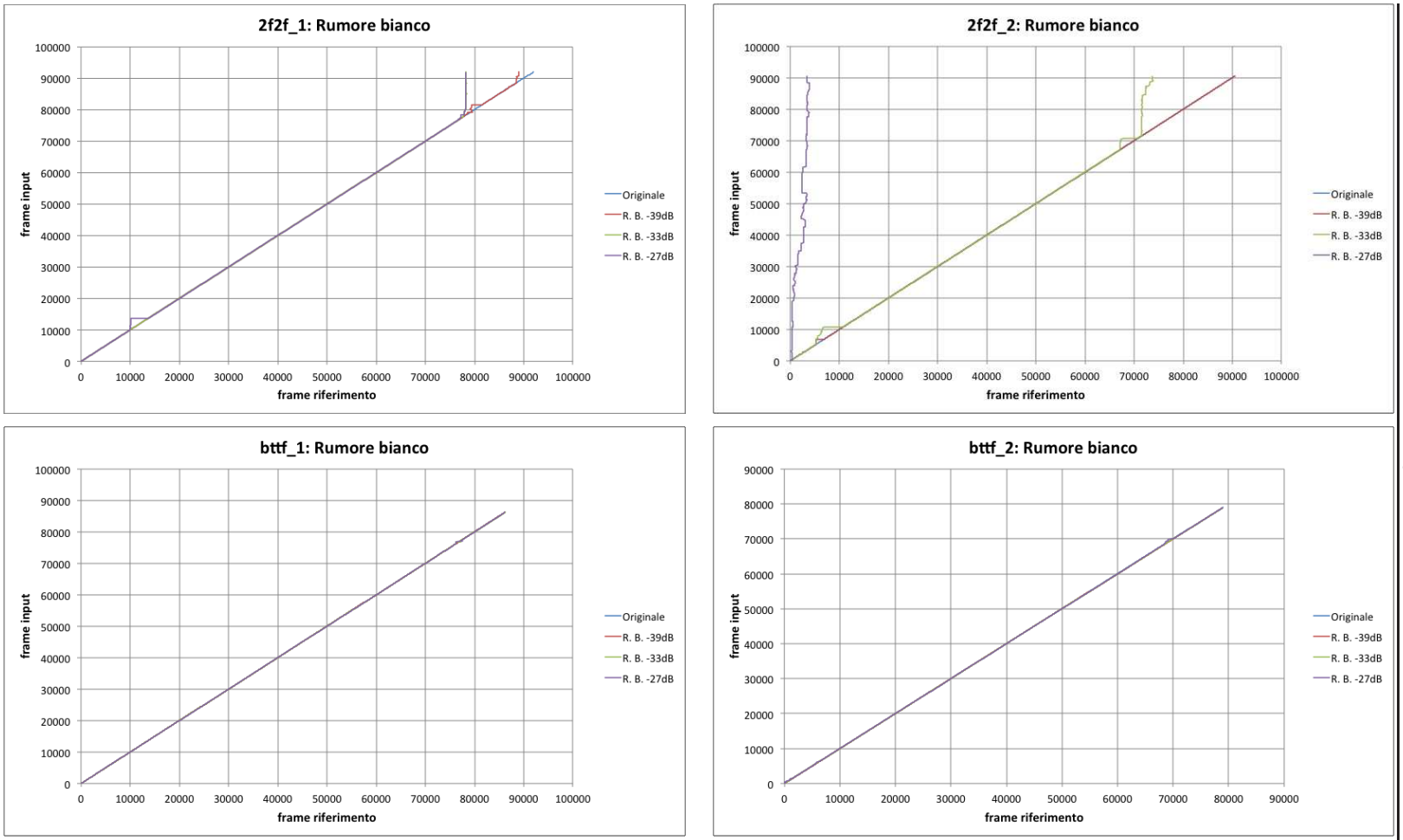

Figura 3.5: Estratti a cui è stato aggiunto un rumore bianco di sottofondo.

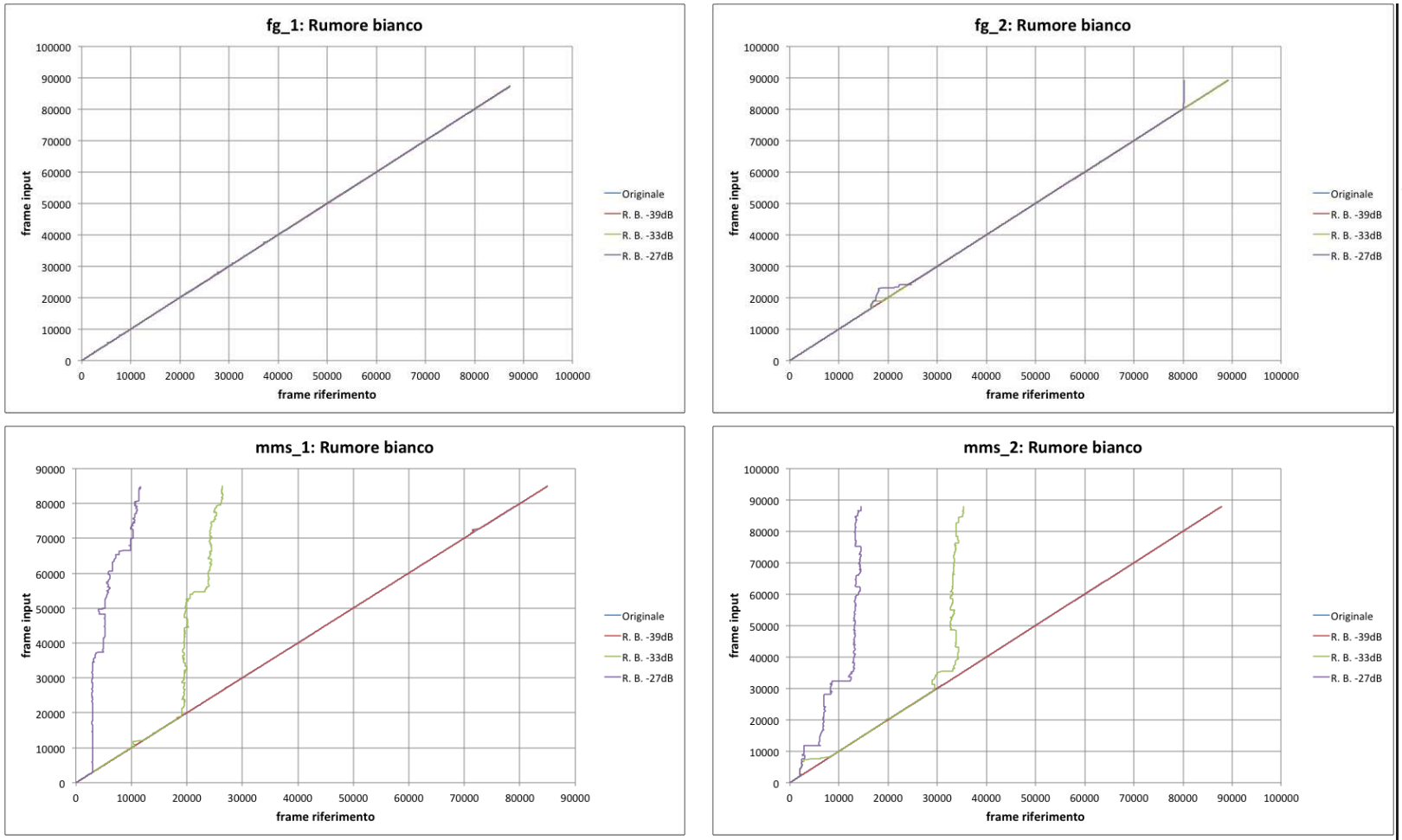

Figura 3.6: Estratti a cui è stato aggiunto un rumore bianco di sottofondo.

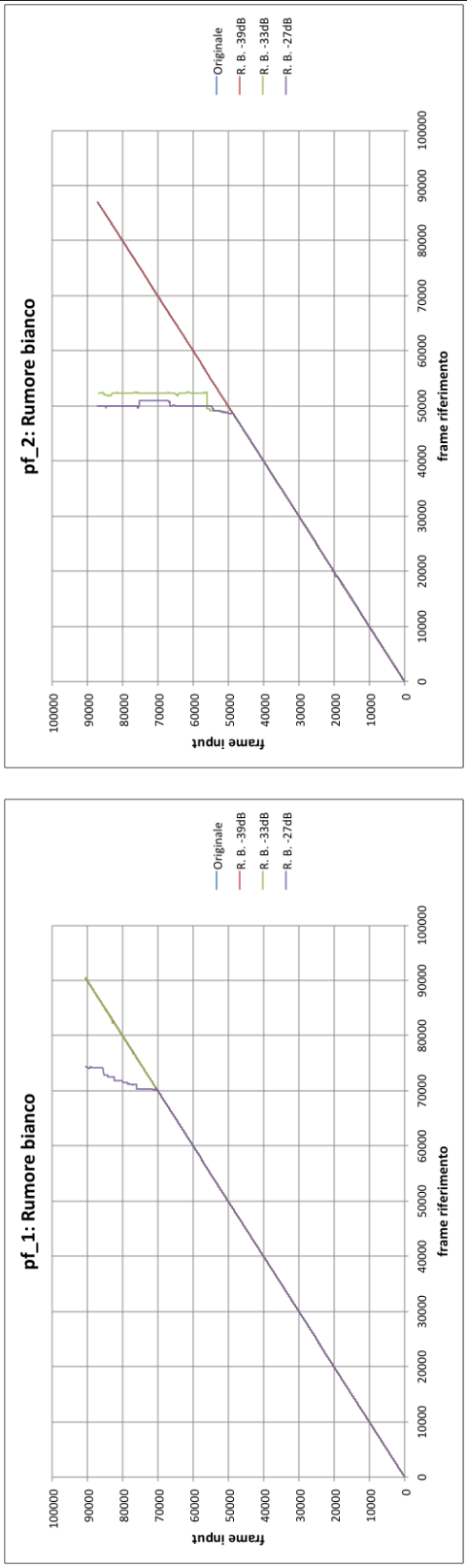

Figura 3.7: Estratti a cui è stato aggiunto un rumore bianco di sottofondo.

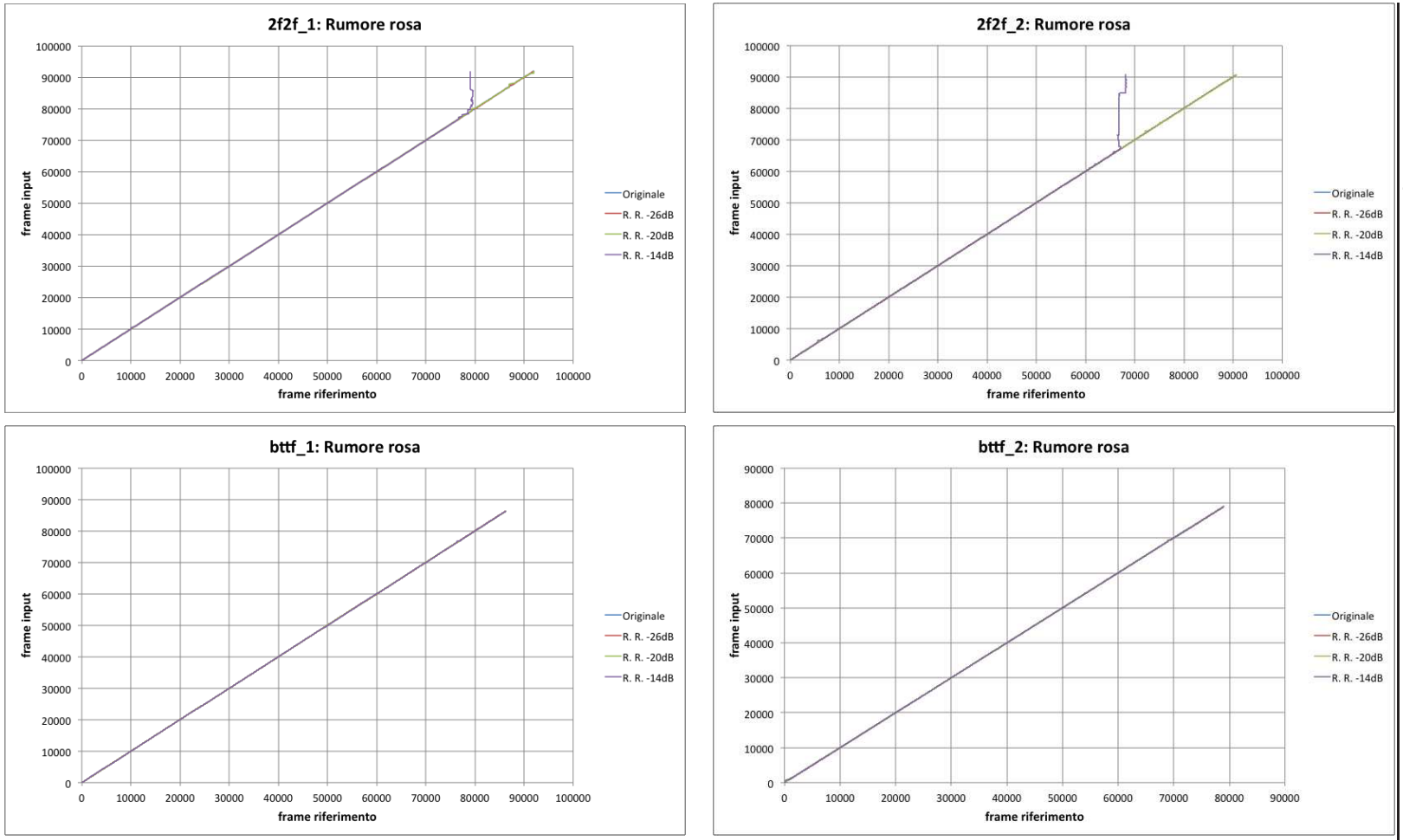

Figura 3.8: Estratti a cui è stato aggiunto un rumore rosa di sottofondo.

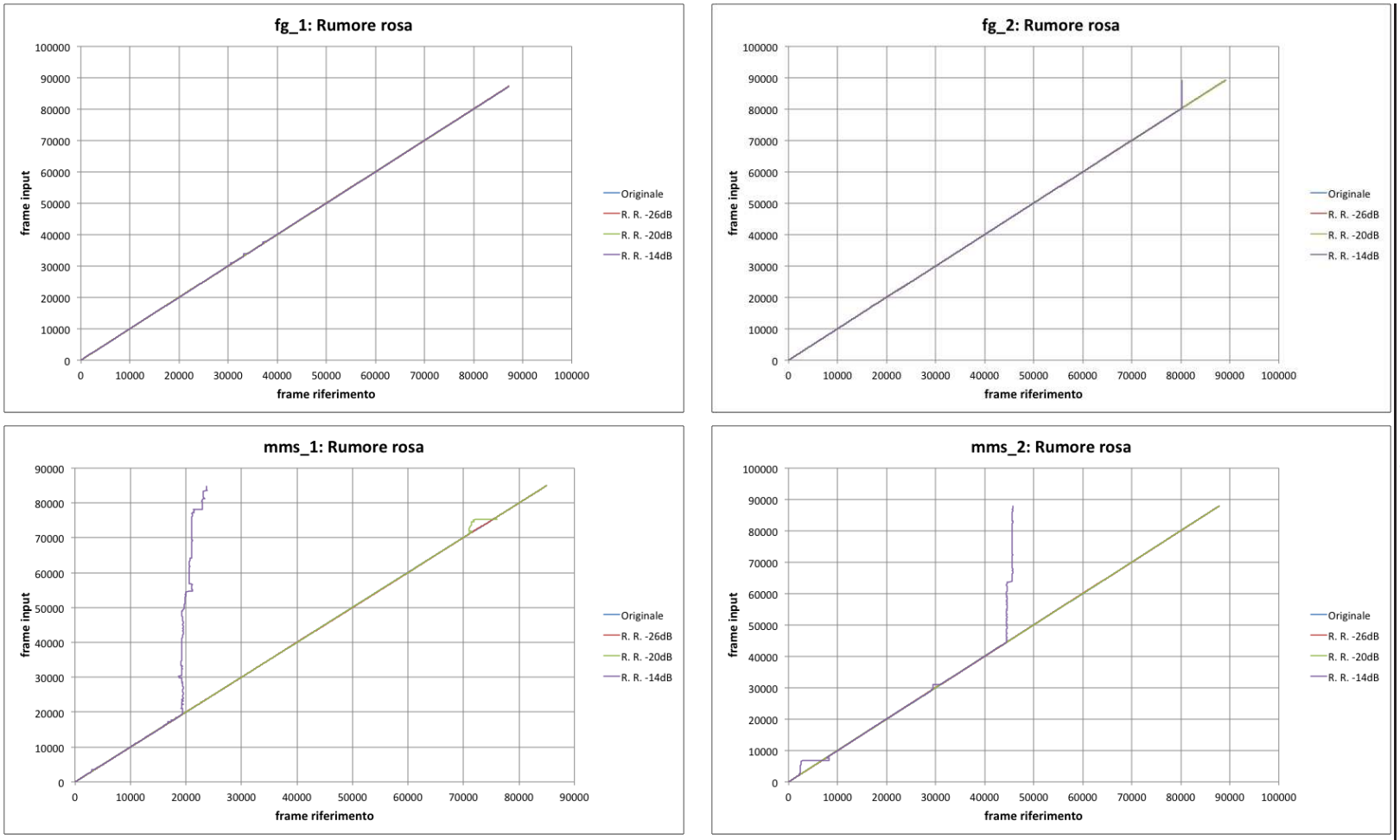

Figura 3.9: Estratti a cui è stato aggiunto un rumore rosa di sottofondo.

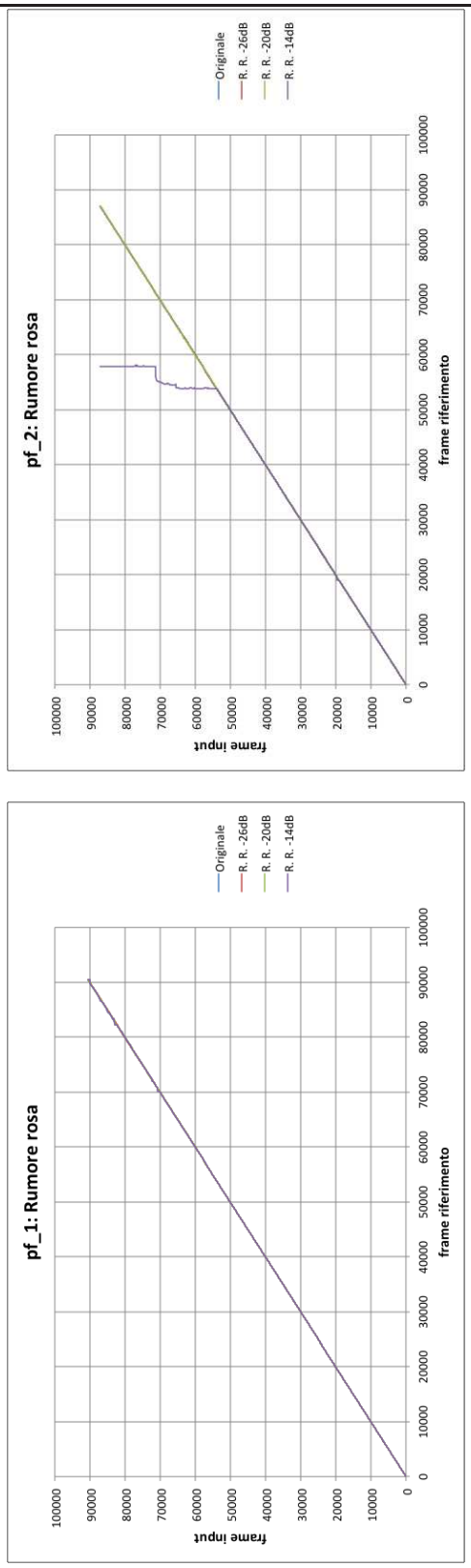

Figura 3.10: Estratti a cui è stato aggiunto un rumore rosa di sottofondo.

Pause

Questo test ha l'obiettivo di verificare l'efficacia del software nel caso in cui si metta in pausa il film che si sta guardando (come può accadere, ad esempio, guardando un DVD), o quando si verifica un "buco" di segnale in una trasmissione video.
Perciò sono stati inseriti dei silenzi (prima di 3 secondi e poi di 6 secondi) in certi punti degli estratti; precisamente intorno ai minuti 3, 6, 9 e 12. La conseguenza ovvia è che il nuovo estratto risulta più lungo dell'originale.
Quello che ci si aspetta di vedere perciò, sono dei gradini in prossimità dei *frame* 15500, 32000, 47000 e 63000.
E' importante notare che è possibile riconosce solo pause di durata inferiore rispetto all'estensione della finestra di allineamento dopo il generico punto (i, j), che in questo caso è pari a 10 secondi (vedi Sezione 3.2.2).
Come si può vedere nelle Figure 3.11, 3.12 e 3.13, i gradini sono nei posti giusti e l'allineamento risulta corretto.

Tagli

Lo scopo di questo test è affine al precedente, in quanto verifica sempre la capacità dell'algoritmo di saltare da un punto all'altro del file di riferimento. Questa volta però, si è voluta ricreare la situazione in cui vengono tagliati pezzi di film per esigenze tecniche (ad esempio per adattarli alla trasmissione in TV oppure per censurare particolari scene), o semplicemente perchè sono stati fatti dei tagli nella pellicola quando le singole pizze vengono montate tra loro.
Perciò sono stati tagliati dei pezzi (prima di 3 secondi e poi di 6 secondi) in certi punti degli estratti; precisamente intorno ai minuti 4, 8 e 12. La conseguenza ovvia è che il nuovo estratto risulta più corto dell'originale.
Quello che ci si aspetta di vedere perciò, sono dei gradini in prossimità dei frame 20500, 41000 e 62000.
E' importante notare che è possibile riconosce solo tagli di durata inferiore rispetto all'estensione della finestra di allineamento dopo il generico punto

(i, j), che in questo caso è pari a 10 secondi (vedi Sezione 3.2.2).
Come si può vedere nelle Figure 3.14, 3.15 e 3.16, i gradini sono nei posti giusti e l'allineamento viene quasi sempre portato a termine. Nel secondo grafico di Figura 3.16, si può osservare come dopo il primo taglio di 6 secondi il sistema si sia quasi perso, tornando poi ad allineare correttamente.

Variazione di Velocità

Con questo test si è voluto verificare l'efficacia dell'algoritmo in una situazione particolare, ovvero quando la velocità del file di input è diversa da quella del file di riferimento. Questa differenza si nota ad esempio tra i film trasmessi in TV e quelli proiettati al cinema. Infatti, il formato standard delle TV europee (NTSC) prevede una riproduzione a 25 frame per secondo (FPS), mentre le pellicole cinematografiche vengono proiettate a 24 FPS. La differenza tra queste velocità è poco più del 4%.
Per ogni estratto sono state create due versioni, rispettivamente con un aumento e una riduzione del 5% della velocità. Queste modifiche si ripercuotono su durata, pitch e tempo (battiti al minuto) dell'estratto.
Come si può vedere nelle Figure 3.17, 3.18 e 3.19, le rette dei test con velocità variata hanno pendenza diversa rispetto all'originale. Il sistema fallisce in certe occasioni, soprattutto quando la velocità di riproduzione è maggiorata.

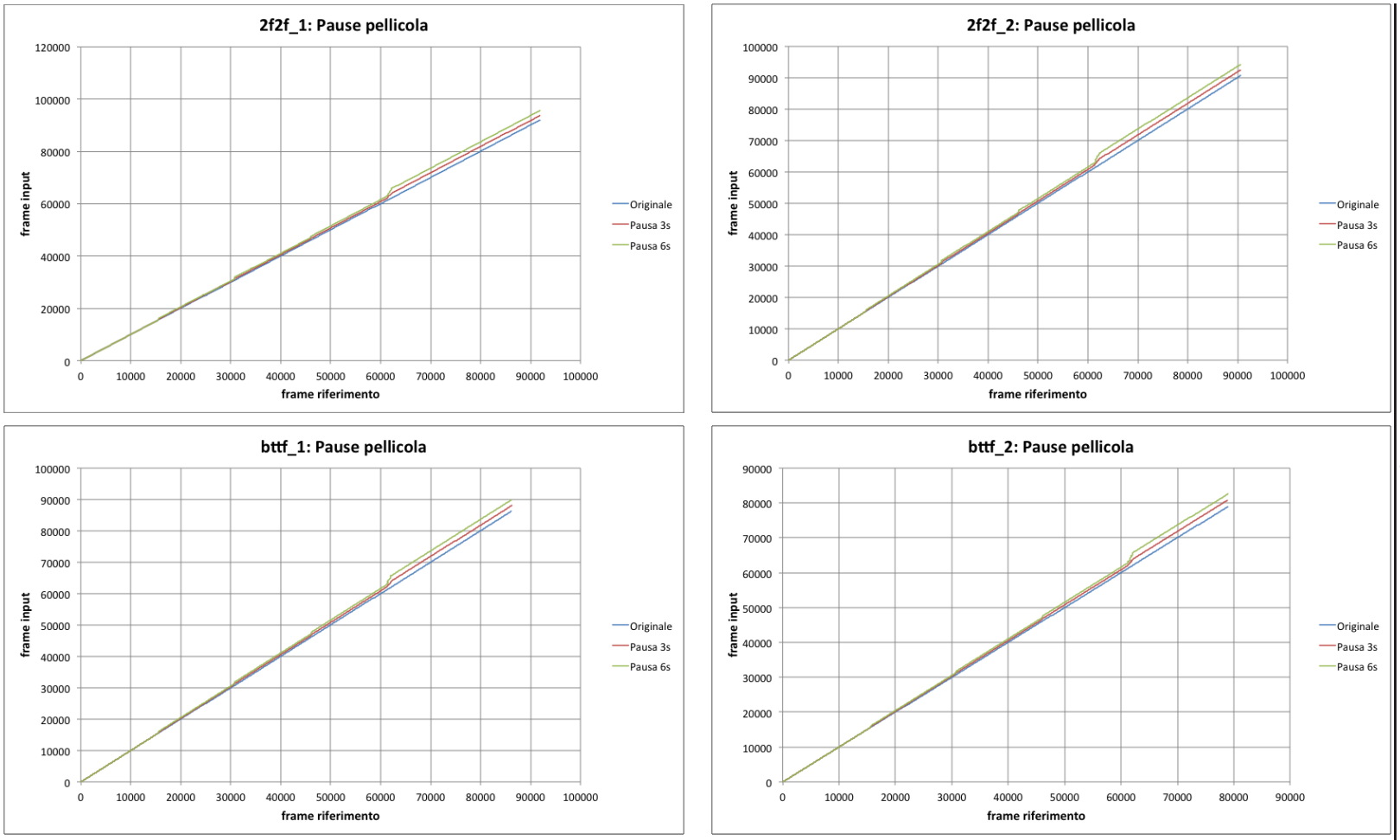

Figura 3.11: Estratti a cui sono stati aggiunti dei silenzi.

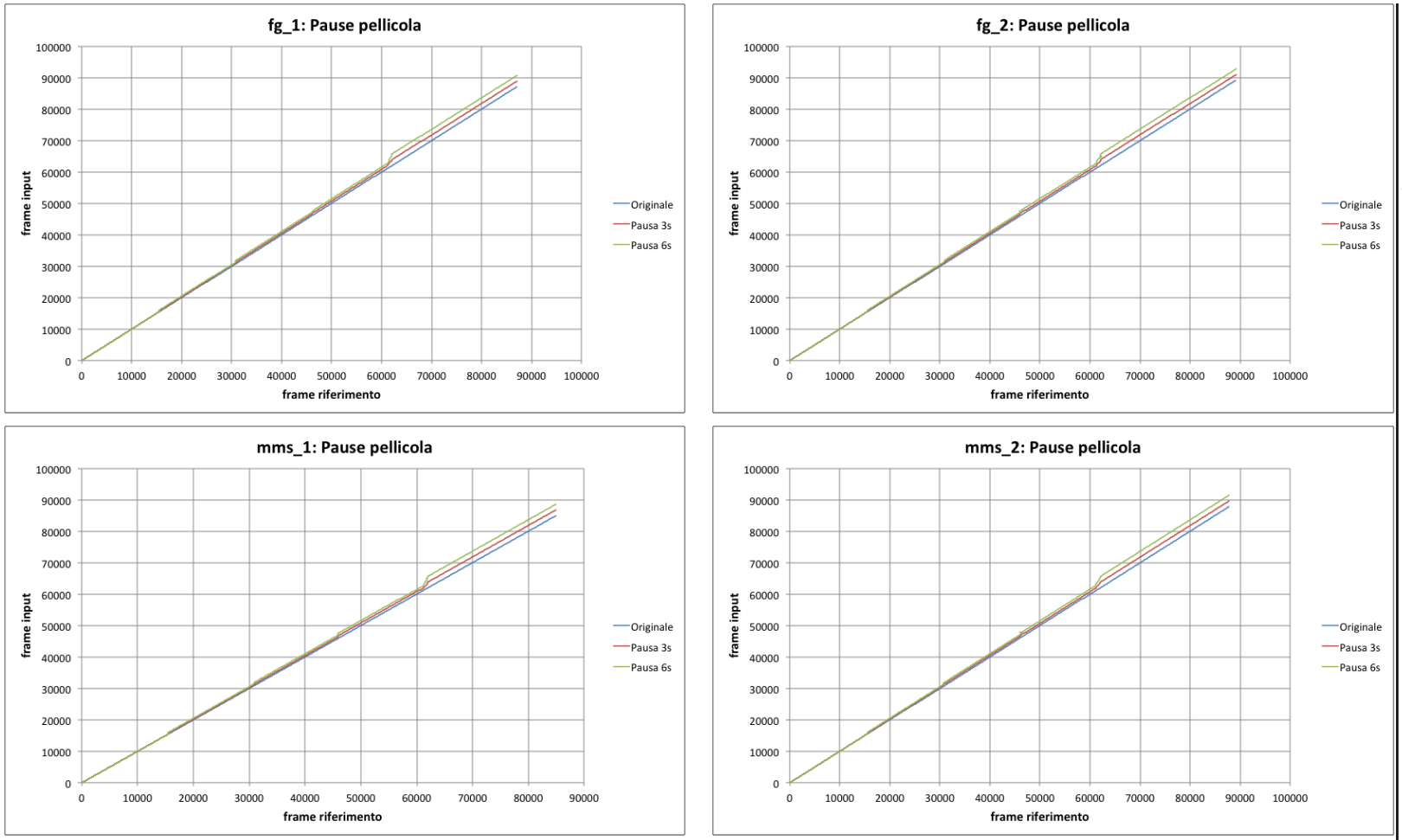

Figura 3.12: Estratti a cui sono stati aggiunti dei silenzi.

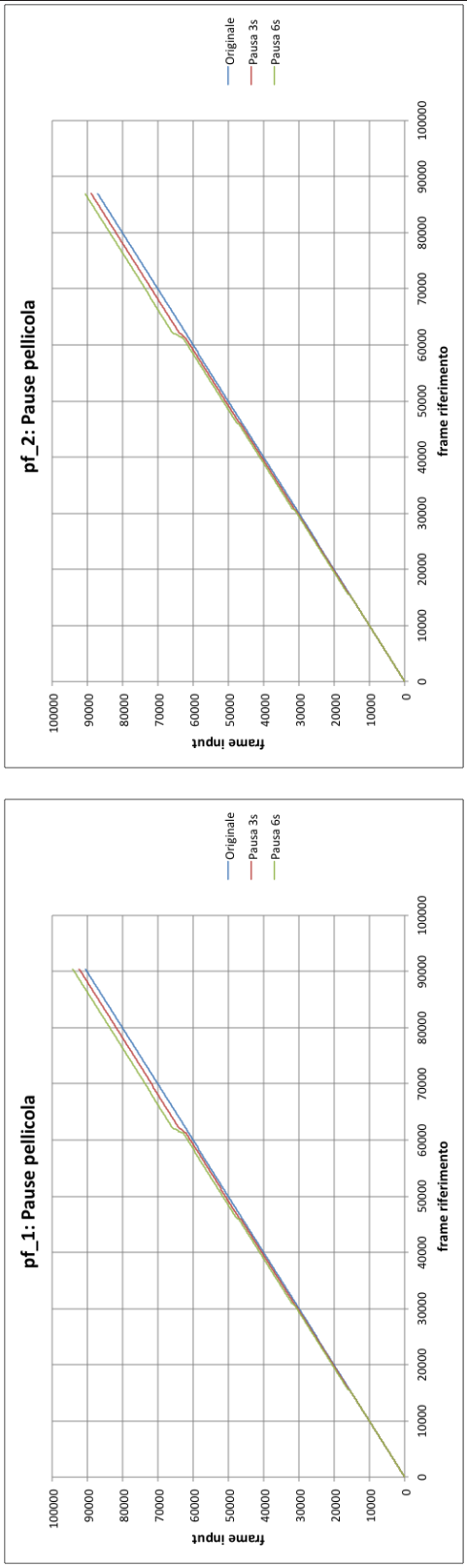

Figura 3.13: Estratti a cui sono stati aggiunti dei silenzi.

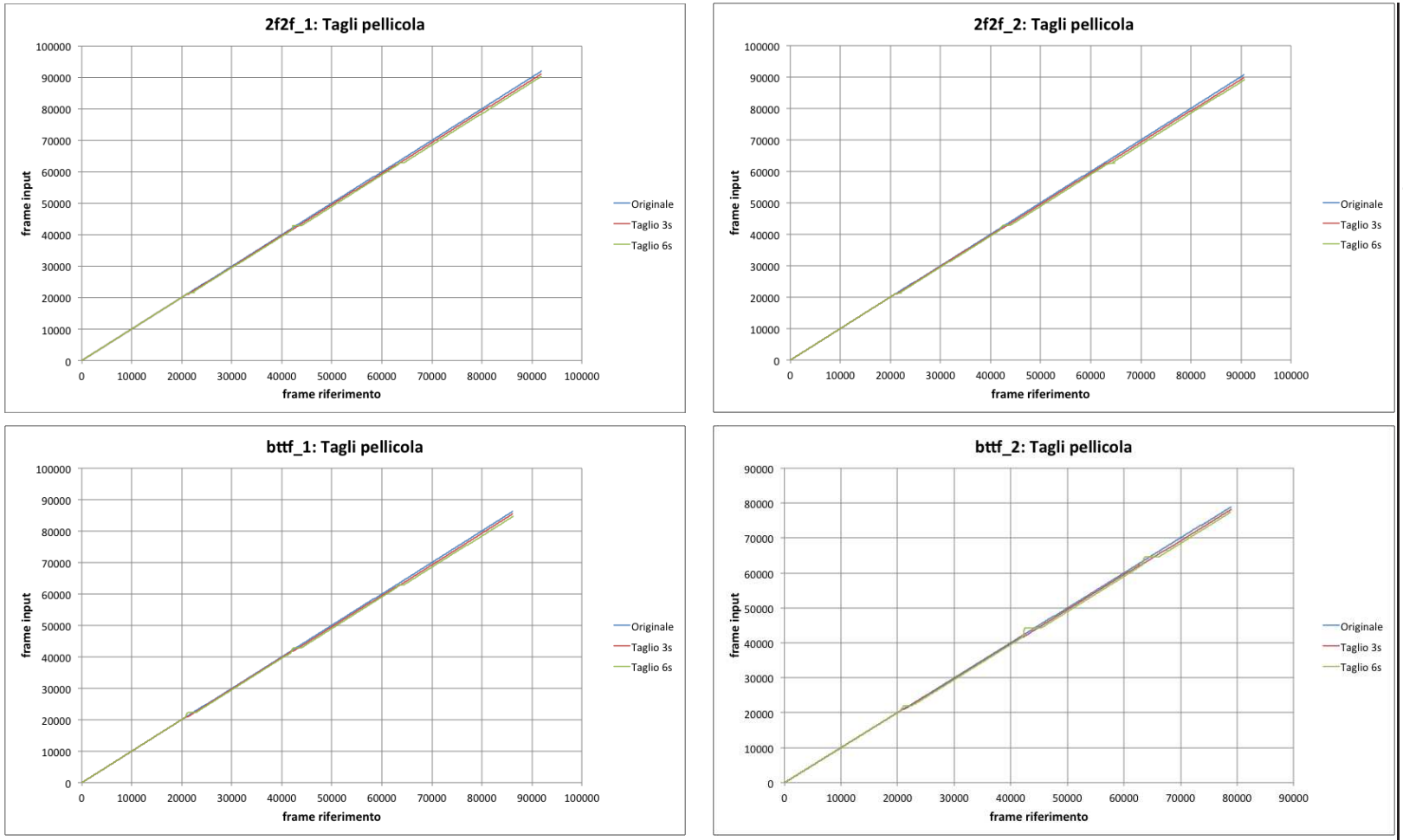

Figura 3.14: Estratti a cui sono stati tagliati dei pezzi.

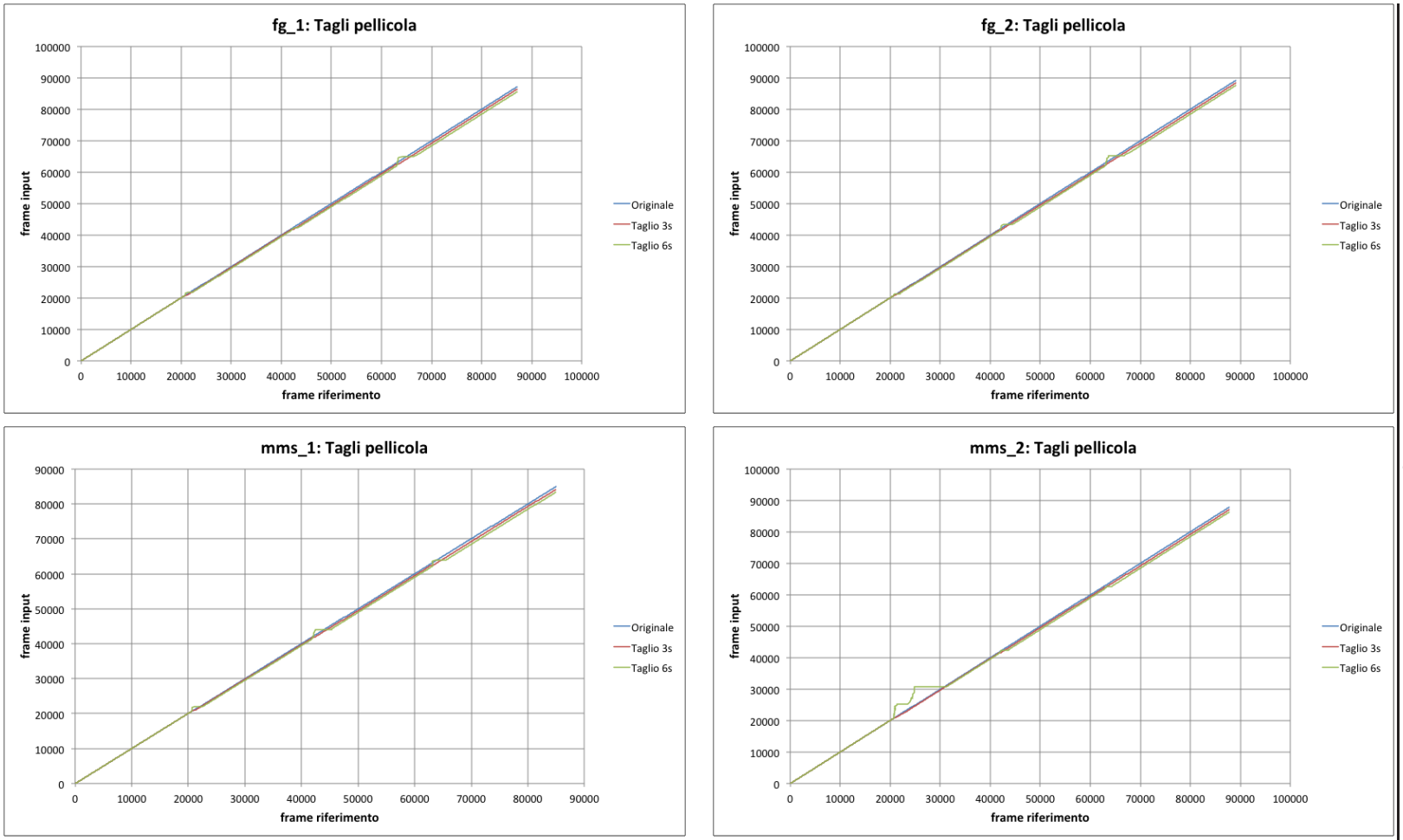

Figura 3.15: Estratti a cui sono stati tagliati dei pezzi.

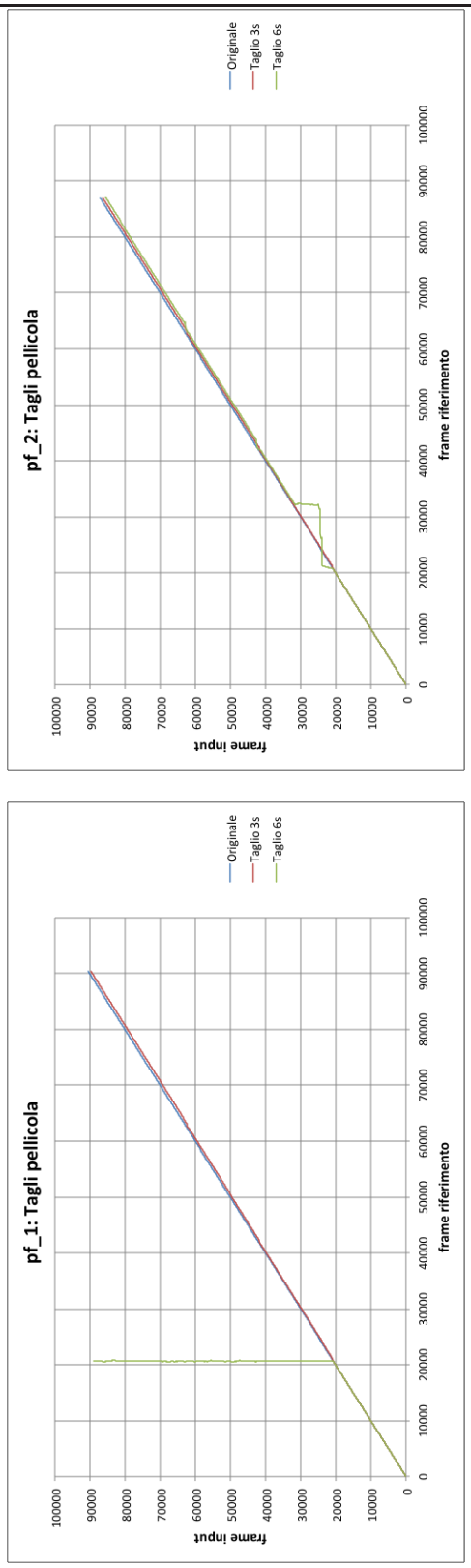

Figura 3.16: Estratti a cui sono stati tagliati dei pezzi.

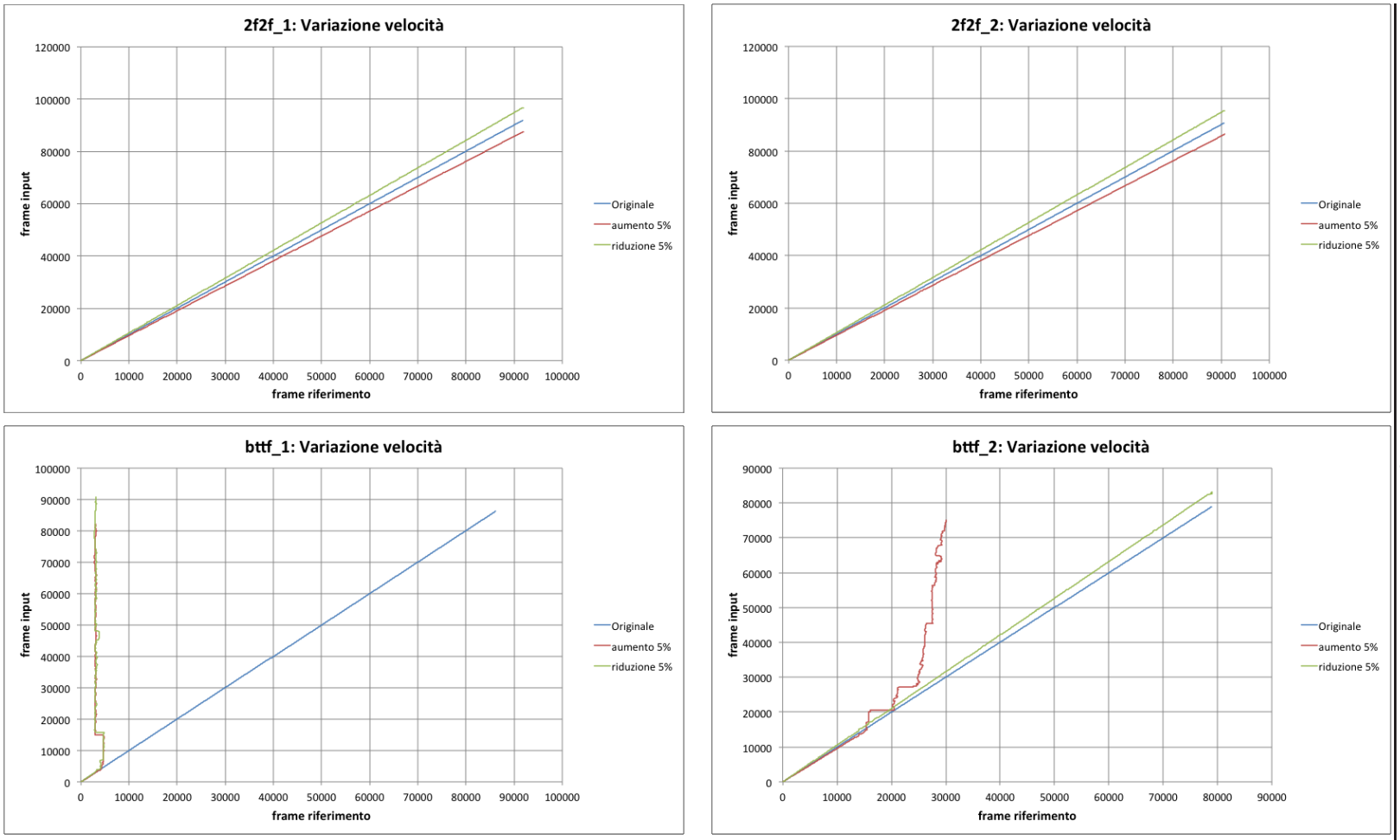

Figura 3.17: Estratti a cui è stata variata la velocità.

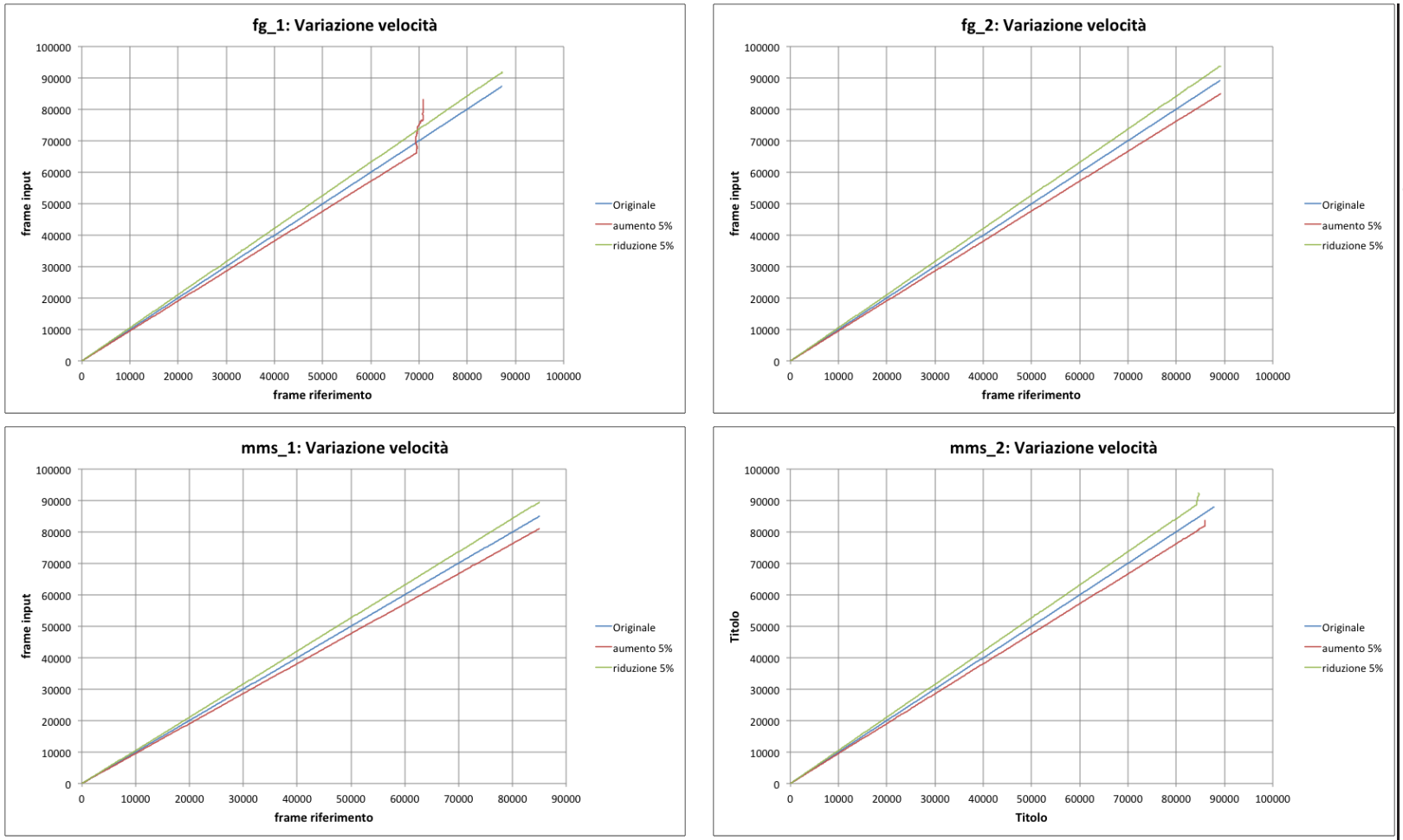

Figura 3.18: Estratti a cui è stata variata la velocità.

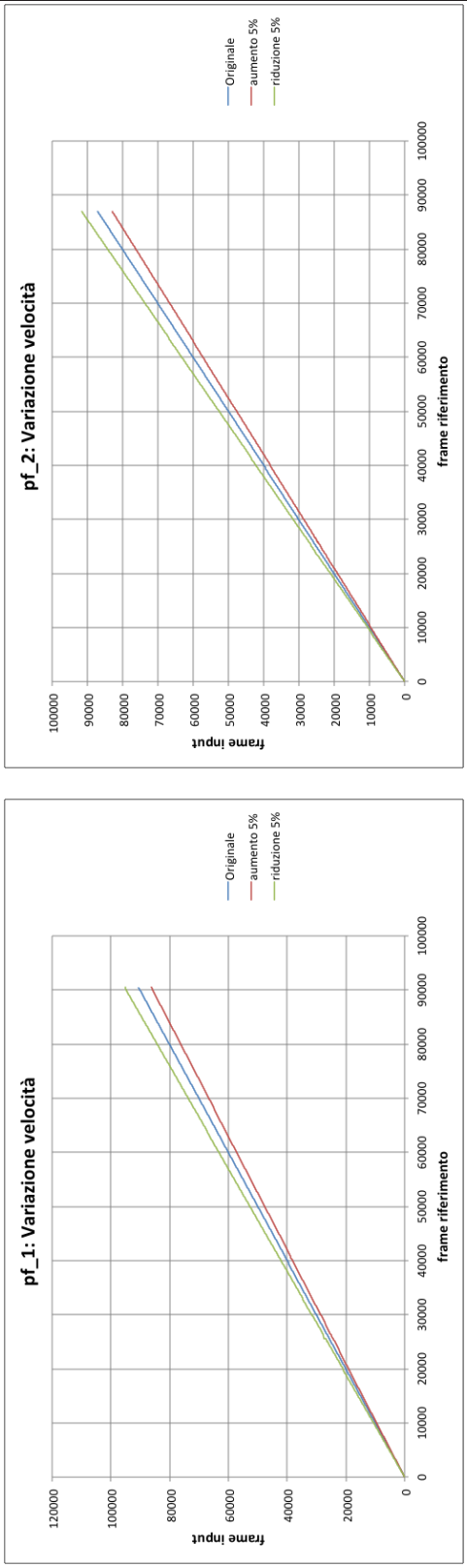

Figura 3.19: Estratti a cui è stata variata la velocità.

3.2.5 Riassunto dei Risultati

In Figura 3.20 viene riportata una tabella che riassume le percentuali di sottotitoli riconosciuti, ottenute dai test eseguiti. Le celle con sfondo rosso (e valori in grassetto) indicano che nello specifico test, il sistema non ha completato l'allineamento.

3.2.6 Prestazioni

In ogni sperimentazione c'è solitamente una sezione dedicata alle prestazioni del software in esame.
Tuttavia, in questo specifico caso, l'analisi delle prestazioni appare un processo prematuro, che fornirebbe dati non significativi. Allo stato attuale il software è in una versione appositamente progettata per la fase di implementazione, *debug* e test del software stesso. Ciò significa che il codice è ancora in uno stato primitivo e non ottimizzato, anche se funzionante.
Inoltre la principale modifica apportata, che ha reso possibile la conduzione dei test, renderebbe i dati raccolti (percentuali di utilizzo CPU/memoria) non significativi. Infatti, se l'input viene preso dal disco fisso, il sistema tende a processare i dati che ha a disposizione il più velocemente possibile (più velocemente rispetto all'acquisizione via periferica hardware). Il risultato è un utilizzo massimo della CPU, il che non rivela nulla riguardo le reali performance del software.
Tuttavia, per avere una stima della velocità del metodo di allineamento si può considerare la lunghezza dell'input e il tempo di esecuzione. Il sistema impiega poco più di 5 minuti per allineare un un estratto audio di 16 minuti[1], essendo quindi tre volte più veloce dell'esecuzione in tempo reale.

[1]Su processore dual core 2,16 GHz, 2 GB RAM.

	2f2f_1	2f2f_2	bttf_1	bttf_2	fg_1	fg_2	mms_1	mms_2	pf_1	pf_2	Media
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
Passabasso	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
Riverbero	99,8	99,4	100,0	100,0	96,9	97,8	100,0	100,0	99,2	100,0	99,3
Rumore Bianco -39dB	**88,4**	98,3	100,0	99,8	100,0	100,0	97,5	99,6	100,0	99,5	98,3
Rumore Bianco -33dB	**77,4**	**65,0**	99,6	99,6	98,9	98,3	**23,6**	**36,6**	96,8	**58,1**	75,4
Rumore Bianco -27dB	**73,8**	**3,6**	97,8	98,4	95,6	**83,4**	**10,1**	**11,1**	**78,8**	**56,7**	60,9
Rumore Rosa -26dB	95,8	99,4	100,0	100,0	99,7	99,8	99,6	100,0	100,0	99,8	99,4
Rumore Rosa -20dB	94,6	97,5	99,8	99,8	98,7	99,8	99,6	99,6	98,9	98,6	98,7
Rumore Rosa -14dB	**76,9**	**64,0**	98,5	99,6	96,6	**90,3**	**22,5**	**47,3**	96,1	**64,3**	75,6
Pause 3 secondi	100,0	99,4	99,8	100,0	99,7	99,5	100,0	100,0	99,8	99,8	99,8
Pause 6 secondi	100,0	99,6	99,8	100,0	99,0	100,0	99,6	100,0	99,6	99,8	99,7
Tagli 3 secondi	99,1	98,3	98,5	98,2	97,2	98,3	98,6	99,6	98,3	98,9	98,5
Tagli 6 secondi	97,4	97,7	96,3	92,2	95,9	97,0	91,7	87,4	**27,4**	87,0	87,0
Velocità +5%	97,6	98,5	**5,2**	**27,7**	**74,1**	94,2	100,0	**99,2**	98,7	96,9	79,2
Velocità -5%	98,8	99,2	**5,0**	96,4	90,7	94,7	99,3	**100,0**	99,2	97,1	88,0

Figura 3.20: Tabella riassuntiva delle percentuali di sottotitoli riconosciuti nei test eseguiti applicando agli estratti diverse distorsioni. I valori in grassetto e nelle celle rosse indicano che l'allineamento non è stato portato a termine.

3.2.7 Osservazioni

Gli esperimenti condotti hanno portato alla luce alcune informazioni utili alla successiva fase di sviluppo.
Per prima cosa, il software si è rivelato robusto in relazione al filtraggio con il filtro passa basso, al riverbero, alle pause, ai tagli (purchè di durata limitata) e ad un lieve livello di rumore.
Nonostante i buoni risultati ottenuti con salti di lieve entità, il software non può fare nulla nel caso (comune) in cui un film venga fatto avanzare di alcuni minuti (ad esempio come può succedere guardando un DVD).
I risultati meno soddisfacenti derivano dall'aggiunta di livelli medio/alti di rumore e dalla variazione di velocità. Per quanto riguarda il rumore, è facile immaginare la causa del problema. Sebbene il sistema confronti *frame* corrispondenti, quello derivante dal file di input è troppo alterato per essere riconosciuto in modo esatto, infatti il rumore ha l'effetto di rendere i *frame* molto simili gli uni con gli altri. Mentre per la variazione della velocità la situazione è più complessa.
Il software si aspetta di trovare i *frame* da allineare nelle stesse posizioni, o al massimo spostati di una certo valore. Con il cambio di velocità invece, è come se i *frame* fossero dilatati/ristretti, perciò il software si trova a confrontare campioni diversi, che non corrispondono. La soluzione più banale sarebbe quella di impostare come parametro la velocità a cui il file di input è riprodotto, in modo tale che se il file di riferimento è stato calcolato ad una velocità diversa, esso venga ricalcolato per tenere conto di questa differenza.
In generale, il software ha riconosciuto il 90.6% dei sottotitoli, che si può ritenere un buon risultato di partenza.
Dall'analisi dei file di *dump* creati a seguito dei test, è stato possibile individuare quali sono i suoni che rendono più difficile il funzionamento del sistema e che sono sorgenti di disallineamento.
In ordine di importanza:

1. Silenzio;

2. Dialoghi lenti (con pause lunghe tra una battuta e l'altra);
3. Rumori ripetitivi (ad esempio il suono di accelerazione di una macchina, che può durare anche per qualche secondo).

In sintesi, le parti stazionarie del segnale tendono a peggiorare le prestazioni del sistema.

Capitolo 4

Ottimizzazione del Sistema di Allineamento

Nella fase di test preliminari sono stati scoperti alcuni limiti significativi del sistema ed ora si andranno ad esplorare alcune possibilità per renderlo più robusto, con riferimento specifico a quelle situazioni in cui si è dimostrato carente.
In particolare, le difficoltà maggiori sono state riscontrate in condizioni di rumore medio/alto e di cambio di velocità di riproduzione. Inoltre, è stato osservato un calo nella percentuale di successo a seguito dei tagli sulla pellicola, anche se l'allineamento è stato portato a termine nella quasi totalità dei casi.

4.1 Funzioni di Costo

Per tentare di migliorare il comportamento dell'algoritmo nei casi peggiori, sono state implementate diverse varianti della funzione di costo usata nel DTW. Queste funzioni, che in parole povere misurano delle distanze, si possono derivare da indici di similarità, anche se il funzionamento è opposto[1].

[1]Due *frame* che sono molto simili avranno una distanza "piccola" ma un indice di similarità "grande".

D'ora in poi i termini "distanza", "funzione di costo" e "indice di similarità" saranno considerati sinonimi, anche se è ben nota la differenza.
Di seguito è riportato un elenco degli indici utilizzati.

- Coseno: $\frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}$
- α_1: $\sqrt[3]{\frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}}$
- α_2: $\sqrt{\frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}}$
- α_3: $1 - \left(1 - \frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}\right)^2$
- β_1: $1 - \sqrt{1 - \frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}}$
- β_2: $\left(\frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}\right)^2$
- β_3: $\left(\frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sqrt{\sum_{i=1}^{S} x_{ij}^2 \cdot \sum_{i=1}^{S} x_{ik}^2}}\right)^3$
- *Ruzicka*: $\frac{\sum_{i=1}^{S} min(x_{ij},x_{ik})}{\sum_{i=1}^{S} max(x_{ij},x_{ik})}$
- *Bray Curtis*: $\frac{2 \cdot \sum_{i=1}^{S} min(x_{ij},x_{ik})}{\sum_{i=1}^{S} (x_{ij}+x_{ik})}$
- *Similarity Ratio*: $\frac{\sum_{i=1}^{S} x_{ij}x_{ik}}{\sum_{i=1}^{S} x_{ij}^2 + \sum_{i=1}^{S} x_{ik}^2 - \sum_{i=1}^{S} x_{ij}x_{ik}}$

dove $1 \leq i \leq S$ e $1 \leq j \leq F$. S è il numero di *bin* della DFT utilizzati per il calcolo, ovvero il *range* di frequenze ritenuto significativo e F è il numero di *frame* totali, che dipende dalla lunghezza del file audio e da *hop size*.
Questi indici di similarità possono assumere valori nell'intervallo $[0, 1]$, dove 1 indica la perfetta corrispondenza e 0 la totale differenza. Il software però ha bisogno di valutare distanze, che come è stato già accennato funzionano al contrario degli indici di similarità, perciò tutti gli indici sono stati trasformati. Se σ è un generico indice di similarità, la distanza ad esso associata

sarà data da $\delta = 1 - \sigma$.
Prima di analizzare i risultati ottenuti, vale la pena parlare del perchè ci si è concentrati su questi particolari indici.
Il Coseno è uno degli indici più utilizzati in letteratura, perciò è una tappa quasi obbligata in questa fase di sperimentazione. La proprietà più interessante che possiede è quella di considerare solo il coseno dell'angolo dei due vettori che compara, mentre il modulo non influisce nel risultato.
Gli indici $\alpha_k, k \in [1,3]$ e $\beta_k, k \in [1,3]$, sono tutte variazioni del Coseno e hanno lo scopo di "premiare" o "scoraggiare" risultati che tendono agli estremi $\{0,1\}$. In Figura 4.1 si può vedere l'andamento degli indici α_k e β_k. Il Coseno

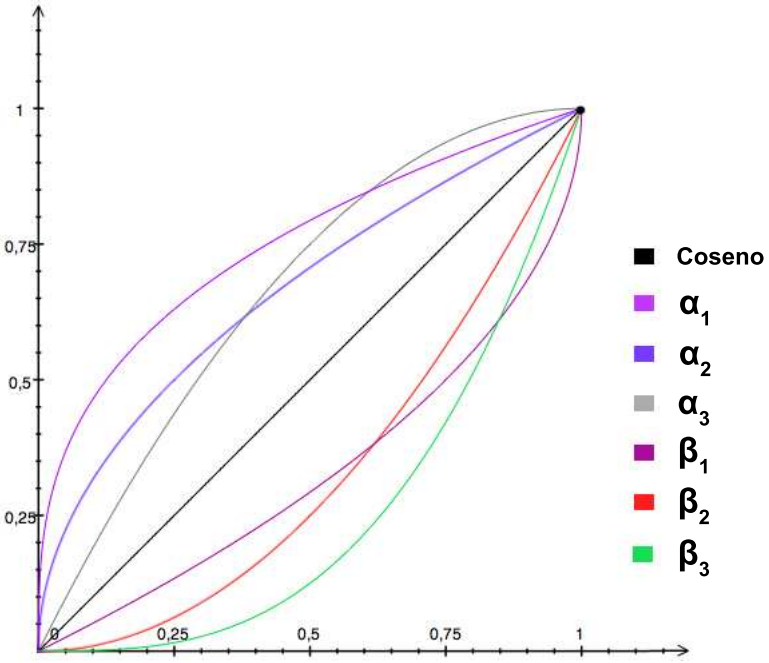

Figura 4.1: Andamento degli indici di similarità derivati dal Coseno.

è pensato come una retta. Gli α_k tendono a sovrastimare il *match*, ovvero innalzano la similarità tra *frame* (curve al di sopra la retta). I β_k invece fanno l'opposto, cioè tendono a sottostimare il *match* abbassando la similarità tra *frame* (curve al di sotto la retta).
Gli indici *Bray Curtis* e *Ruzicka* sono stati presi in esame perchè avevano

riportato risultati interessanti nell'ambito della segmentazione della musica (vedi [10]), anche se il loro principale campo d'applicazione è la genetica.
Infine, *Similarity Ratio* misura la somiglianza tra due vettori basandosi sui moduli delle singole componenti. Nel caso in cui i vettori siano normalizzati, ha prestazioni paragonabili al Coseno ma in caso contrario l'effetto del modulo dei diversi segnali può pregiudicarne l'efficenza.
In Figura 4.2 è riportata una tabella riassuntiva dei risultati ottenuti utilizzando le nuove distanze. I valori rappresentano le percentuali di sottotitoli riconosciuti con successo.
C'è da precisare che la modifica della funzione di costo ha lo scopo di migliorare l'algoritmo nei casi peggiori, perciò ci si è concentrati solo su quelli (variazione di velocità e aggiunta di disturbo). Inoltre nella tabella non compaiono tutte le prove effettuate, ma solo quelle più interessanti (indici α_2, α_3, β_1 e β_2). Gli indici *Bray Curtis*, *Ruzicka*, α_1 e β_3 si sono rivelati inadeguati per questa applicazione, mentre il *Similarity Ratio* soffre del problema della normalizzazione di cui si è accennato.
Un'altra cosa che si può notare, è che i test sul rumore sono stati limitati ad un solo livello: dai precedenti rilevamenti infatti, è stato possibile stabilire che la soglia di -36dB è il limite massimo che consente alla maggior parte degli estratti di essere allineato correttamente. Dopo tale soglia le percentuali di fallimento aumentano significativamente e quindi appare sensato provare le nuove distanze in questa situazione particolare.
Inoltre, sono stati controllati a campione anche i casi in cui l'algoritmo andava bene con la distanza Coseno, per verificare che non ci fosse un degrado delle prestazioni. Riguardo ciò, non si sono osservate variazioni significative (se non un lievissimo peggioramento per alcuni degli indici).
Dalla tabella in Figura 4.2 si possono ricavare informazioni interessanti. Intanto si nota subito che per alcuni degli indici, il numero di fallimenti sui vari estratti (celle rosse) è diminuito notevolmente. L'ultima colonna dà un'idea della bontà delle nuove distanze. Le celle rosse (con il simbolo "$-$") stanno ad indicare un risultato peggiore rispetto alla distanza Coseno, le celle verdi

(con il simbolo "+") indicano un risultato migliore e quelle non evidenziate (con il simbolo "=") indicano che non c'è stato un significativo cambiamento nelle prestazioni.
Si capisce subito dunque che l'indice più promettente è α_2, il quale porta solo miglioramenti e nessun peggioramento. Dato quanto appena scoperto, è sensato utilizzare sempre questo nuovo indice.
In questo modo si risolve il problema dovuto alla velocità di riproduzione dei film. Come si può vedere dalla tabella, con questo nuovo indice la percentuale di successo aumenta drasticamente nei casi di variazione di velocità, soprattuttto nel caso di rallentamento (98,3%). Inoltre, l'algoritmo non fallisce mai l'allineamento. Questo significa che se avessimo tutti i file di riferimento a 25 FPS, ci troveremmo sempre nel caso favorevole e non ci sarebbe bisogno di far scegliere all'utente la velocità di riproduzione. Fortunatamente, tutto ciò è possibile. Si presume infatti che l'audio di un dato film venga acquisito da DVD originale con un software apposito (ce ne sono molti gratuiti), dal quale è possibile impostare questo parametro.
Dunque, il problema dell'ambiente di utilizzo del software (casa/cinema) può considerarsi risolto, anche se solo una serie di test effettuati in una situazione reale possono confermare questa ipotesi.

	2f2f_1	2f2f_2	bttf_1	bttf_2	fg_1	fg_2	mms_1	mms_2	pf_1	pf_2	Media	
Indice α2												
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Rumore Bianco -36dB	**88,2**	97,9	100,0	99,8	100,0	100,0	97,1	99,2	99,8	99,1	98,1	=
Velocità +5%	98,8	99,4	**6,9**	98,0	95,7	96,1	100,0	100,0	99,8	98,6	89,3	+
Velocità -5%	99,3	99,4	98,3	97,8	94,1	96,9	99,6	99,6	100,0	98,2	98,3	+
Indice α3												
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Rumore Bianco -36dB	**79,2**	94,1	100,0	99,8	99,8	100,0	**31,2**	98,9	99,6	98,7	90,1	-
Velocità +5%	99,3	99,6	**6,9**	96,2	95,7	97,3	100,0	100,0	100,0	98,6	89,4	+
Velocità -5%	99,8	99,6	**6,7**	96,9	95,2	96,7	99,6	99,6	99,8	98,7	89,3	+
Indice β1												
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Rumore Bianco -36dB	**88,7**	97,2	100,0	99,6	99,7	99,7	98,2	99,2	99,6	98,7	98,1	=
Velocità +5%	97,4	96,8	**6,1**	**27,6**	**90,5**	92,2	98,9	**99,2**	98,1	96,6	80,3	+
Velocità -5%	96,7	97,9	**6,3**	96,9	90,3	94,8	98,6	100,0	98,1	97,3	87,7	-
Indice β2												
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Rumore Bianco -36dB	**79,5**	95,1	100,0	99,6	99,3	99,5	97,8	**12,6**	98,9	96,9	87,9	-
Velocità +5%	97,4	96,8	**6,3**	**29,7**	**71,0**	93,3	99,6	**99,2**	97,0	96,2	78,7	-
Velocità -5%	96,9	97,5	**6,1**	96,2	88,4	92,2	99,3	99,6	98,5	96,0	87,1	-
Intorno Tipo 2												
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Rumore Bianco -36dB	98,6	99,8	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	99,8	+
Velocità +5%	99,3	99,6	99,6	99,6	96,2	96,4	100,0	100,0	99,6	99,5	99,0	+
Velocità -5%	99,8	99,8	100,0	100,0	96,1	97,2	100,0	100,0	100,0	99,8	99,3	+
Intorno Tipo 3												
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Rumore Bianco -36dB	92,7	95,6	99,4	98,7	96,6	90,4	96,0	99,2	97,2	96,6	96,2	-
Velocità +5%	89,2	86,7	83,7	**49,1**	78,7	80,9	99,3	98,5	87,2	91,9	84,5	+
Velocità -5%	93,2	94,1	94,1	96,0	82,1	84,2	100,0	99,2	92,7	94,2	93,0	+

Figura 4.2: Tabella riassuntiva dei test eseguiti con diversi indici di similarità e diversi tipi di intorno. Le celle rosse con i valori in grassetto indicano il fallimento dell'allineamento. Nell'ultima colonna, le celle verdi indicano un miglioramento rispetto alla situazione precedente, le celle rosse indicano un peggioramento e quelle non evidenziate indicano che non c'è stata una significativa variazione.

4.2 Tipo di Intorno

Un'altra possibilità presa in considerazione, è la scelta di diversi *neighbours* per il calcolo del costo di allineamento di due *frame*. Alcuni esempi si possono trovare in [8]. Anche in questo caso è stato applicato lo stesso ragionamento seguito nella Sezione 4.1 per la scelta delle distorsioni da esaminare.

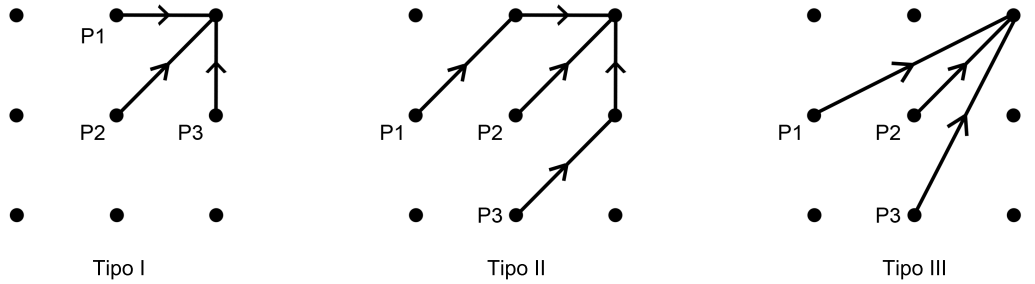

Figura 4.3: Tipi di intorni utilizzati per il calcolo del costo di allineamento, suggeriti in [8].

Come descritto nella Sezione 2.2, i test sono stati effettuati fino a questo momento utilizzando un intorno di Tipo I (Figura 4.3). Questo è il tipo di intorno più semplice che si possa immaginare, in quanto soddisfa i tre casi base che si possono incontrare, ovvero:

1. Il file di input e quello di riferimento vengono allineati linearmente, cioè corrispondono esattamente (percorso da P2);

2. Il file di input contiene meno frame rispetto al file di riferimento, come può succedere quando ci sono delle scene mancanti nel film che si sta guardando (percorso da P1).

3. Il file di input contiene più frame rispetto al file di riferimento, come può succedere quando si mette in pausa il film che si sta guardando (percorso da P3).

Nonostante il Tipo I si sia rivelato relativamente buono, sono stati implementati anche gli intorni di Tipo II e III per verificare se ci fosse margine di miglioramento. Viene infatti naturale pensare che si possano ottenere migliori risultati utilizzando un metodo che sfrutta più informazioni (si tiene conto di due istanti passati, invece che di uno solo).
Le formule per il calcolo dei costi diventano quindi, per il Tipo II:

$$D(i,j) = d(i,j) + min \begin{cases} D(i-1,j-2) + d(i,j-1) & P3 \\ D(i-2,j-1) + d(i-1,j) & P1 \\ D(i-1,j-1) & P2 \end{cases}$$

Mentre per il Tipo III:

$$D(i,j) = d(i,j) + min \begin{cases} D(i-1,j-2) & P3 \\ D(i-2,j-1) & P1 \\ D(i-1,j-1) & P2 \end{cases}$$

Nella tabella riassuntiva di Figura 4.2 sono riportati i risultati e si può chiaramente vedere che l'intuizione è corretta. I valori in tabella sono le percentuali di successo corrispondenti alle prove effettuate sui vari estratti (i colori hanno la stessa valenza descritta nella Sezione 4.1).
Con l'intorno di Tipo II si ha un netto incremento dei successi e in più, l'allineamento non fallisce mai. Con riferimento all'ultima colonna, si può osservare che l'intorno di Tipo II apporta solo miglioramenti (celle verdi), mentre per quanto riguarda l'intorno di Tipo III si ha un peggioramento (cella rossa). Oltre alle prove riportate in tabella, sono stati eseguiti dei test a campione nelle situazioni in cui l'intorno di Tipo I andava bene, per verificare che non ci fosse un degrado delle prestazioni. Questi test hanno mostrato che, a fronte dei miglioramenti riscontrati sui casi critici, non ci sono significative perdite di prestazioni nelle situazioni meno critiche.
Alla luce di quanto appena scoperto dunque, si proseguirà lo sviluppo del software utilizzando l'intorno di Tipo II.

4.3 Risultati Cumulativi delle Modifiche al DTW

Nelle sezioni precedenti di questo capitolo, sono state apportate separatamente due modifiche che influiscono in modo diretto sul DTW. E' interessante vedere a questo punto quali sono i risultati che si ottengono applicandole contemporaneamente. A questo scopo, sono stati ripetuti alcuni dei test eseguiti in precedenza, utilizzando questa volta α_2 come indice di similarità e il Tipo II di intorno.
Nella tabella di Figura 4.4a sono riportati i risultati. In generale, il sistema si è rivelato affidabile nel 99.3% dei casi. Una cosa da notare è che grazie alle modifiche apportate, la soglia limite di rumore tollerabile si è alzata di 9dB, infatti ora il sistema è in grado di riconoscere i sottotitoli con sottofondo di rumore bianco a -27dB (senza mai fallire l'allineamento).
In Figura 4.4b si può osservare come la nuova configurazione sia di beneficio, con un miglioramento generale di circa il 9%.

	2f2f_1	2f2f_2	bttf_1	bttf_2	fg_1	fg_2	mms_1	mms_2	pf_1	pf_2	Media	
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Passabasso	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	=
Riverbero	100,0	100,0	100,0	100,0	97,7	99,2	100,0	100,0	99,8	100,0	99,7	+
Rumore Bianco -27dB	99,1	99,2	100,0	100,0	99,7	100,0	99,3	100,0	100,0	100,0	99,7	+
Pause 6 secondi	98,3	98,9	97,8	99,4	98,2	98,6	98,6	98,9	98,7	99,3	98,7	=
Tagli 6 secondi	97,9	98,5	97,6	97,6	97,0	98,1	95,7	98,5	97,0	98,0	97,6	+
Velocità -5%	100,0	100,0	100,0	99,8	96,9	98,1	100,0	100,0	100,0	99,8	99,5	+

(a)

(b)

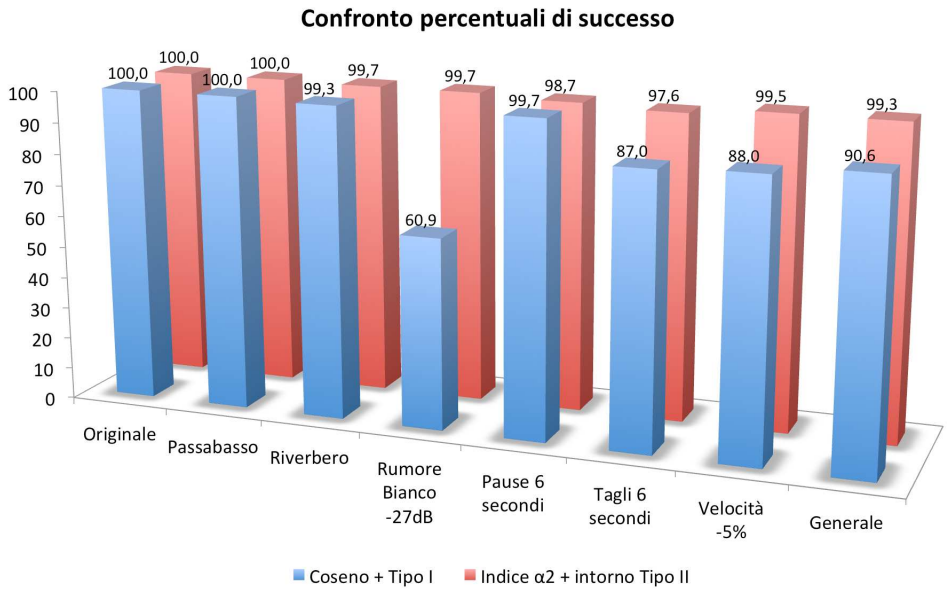

Figura 4.4: (a) Tabella riassuntiva dei test eseguiti con un intorno di Tipo II e l'indice α_2. Nell'ultima colonna, le celle verdi indicano un miglioramento. (b) Grafico di confronto tra i risultati ottenuti nella Sezione 3.2 e quelli ottenuti con un intorno di Tipo II e l'indice α_2.

4.4 Compattezza della Rappresentazione

Finora sono state apportate delle modifiche all'algoritmo del DTW, per renderlo più efficace. L'obiettivo ora è quello di agire sulla rappresentazione del segnale audio (file *fft*) in modo da renderla il più compatta possibile, senza però influire in modo negativo sulle prestazioni. Infatti, bisogna ricordare che l'ambiente di applicazione finale sarà un dispositivo mobile, il quale ha delle limitazioni dal punto di vista di potenza di calcolo e memoria.
Allo stato attuale, per un film di 1 ora e 30 minuti si avrebbe un file di riferimento di circa 645 MB (utilizzando i parametri descritti nella Sezione 3.2.2), ovviamente una dimensione inaccettabile.

4.4.1 Mappatura "a Bande"

Nella Sezione 2.1 è stata presentata una possibile strategia per generare una rappresentazione più compatta, basata sul funzionamento del sistema uditivo umano. Grazie a questa osservazione, è stata creata una funzione che mappa valori nel dominio della frequenza, in valori da inserire all'interno dei *bin* (vedi Figura 4.5).

Hz	150	190	240	303	383	484	612	774	979	1237	1564	1977	2500
bin	1	2	3	4	5	6	7	8	9	10	11	12	

Figura 4.5: Mappatura "a bande" utilizzata per ridurre la dimensione della rappresentazione del segnale audio originale.

Sono state eseguite diverse prove attraverso le quali è stato possibile capire che i risultati sono tanto migliori quanto le bande sono strette. Questo comportamento non dovrebbe sorprendere, in quanto la mappatura lineare può essere pensata come una mappatura a bande di ampiezza costante pari a $\frac{1 \cdot f_c}{w}$ Hz (notazione di Sezione 2.1), che è la massima precisione ottenibile in frequenza (corrispondente ad un *bin*). Inoltre è stato osservato che è conveniente avere una "granularità" fine a frequenza medio-bassa (fino a 1

kHz), mentre a frequenze più alte la larghezza di ciascuna banda può essere aumentata senza perdita di informazioni utili. Combinando tutte le idee menzionate finora, è stata creata la scala in Figura 4.5 che è composta da 12 bande critiche, la cui dimensione aumenta in modo logaritmico. Le frequenze considerate sono nell'intervallo 100-2500 Hz ed in ogni *bin*, deve essere inserita la somma dei valori appartenenti ad una certa banda.
Riprendendo l'esempio di prima, un film di 1 ora e 30 minuti viene rappresentato ora con un file di circa 42 MB. La dimensione si è ridotta notevolmente ma la rappresentazione è ancora troppo grande per pensare ad un'applicazione mobile.

4.4.2 Precisione della Trasformata

Un'altra possibilità per la compressione del file *fft* deriva dalla modalità di rappresentazione dei valori stessi. Fino a questo momento, per rappresentare un singolo *bin* è stato impiegato un numero decimale a doppia precisione (tipo double) di dimensione pari a 8 bytes.
L'idea è quella di impiegare un tipo di dato più compatto, che sia in grado di rappresentare ogni *bin* con sufficiente precisione e taglia ridotta. A questo scopo è stato utilizzato il tipo uint16 o unsigned short int, il quale ha una dimensione di 2 bytes e può assumere valori interi nell'intervallo $[0, 2^{16} - 1]$.
A questo punto però, nasce un problema: non c'è modo di sapere a priori se i valori da inserire nei *bin* siano confinati nel nuovo intervallo, perciò è stato necessario modificare lo script MATLAB® in modo che applicasse una normalizzazione dei valori. Il nuovo script quindi, eseguirà le seguenti operazioni:

- Riduce il file da stereo (due canali) a mono (un canale);
- Scrive nel file *fft* i parametri di creazione;
- Esegue un primo passaggio per calcolare il massimo dei valori risultanti dalla somma delle DFT sulle bande critiche;

- Esegue un secondo passaggio, in cui per ogni *frame* calcola i valori normalizzati da inserire nei *bin*;
- Scrive nel file *fft* i valori normalizzati in modo consecutivo.

Riprendendo nuovamente l'esempio di prima, un film di 1 ora e 30 minuti viene rappresentato ora con un file di circa 10 MB. A questo punto ci si può ritenere soddisfatti, in quanto la dimensione raggiunta permette al sistema di essere impiegato in dispositivi mobili. La capacità di memoria dei dispositivi mobili moderni è mediamente di 2 GB, e in alcuni casi arriva anche a 16 o 32 GB. Perciò è ragionevole pensare che un singolo dispositivo possa contenere decine o addirittura centinaia di file di riferimento, corrispondenti ad altrettanti film. Inoltre, grazie alle connessioni internet veloci (3G) e ai piani dati offerti dagli operatori telefonici, è possibile scaricare un file di riferimento di 10 MB da internet in circa 2 minuti (stima decisamente pessimistica) ad un costo ragionevole.

4.5 Risultati delle Modifiche alla Rappresentazione

Nonostante gli ottimi risultati ottenuti nel compattare la rappresentazione del segnale, è necessario controllare che ad essi non sia associato un calo delle prestazioni del DTW, anche se una leggera flessione è inevitabile (e potrebbe essere comunque accettabile se il calo di prestazioni non influisse sulla qualità di fruizione del sistema).
In Figura 4.6a si possono osservare le percentuali di successo sui vari estratti: se confrontate con quelle nella Sezione 4.3 si nota un leggerissimo (e prevedibile) peggioramento. Tuttavia, quello attuale appare essere un buon compromesso tra dimensione della rappresentazione e prestazioni del DTW, infatti in generale la percentuale di successo è del 99%.
Fino a questo momento ci si è basati solo sulla percentuale di sottotitoli riconosciuti per valutare la bontà del sistema. In Figura 4.6b sono proposte delle

statistiche più approfondite, che danno un'idea più concreta dell'usabilità del sistema sviluppato.
E' noto che il tempo di aggiornamento dell'occhio umano è circa un decimo di secondo. Perciò, i sottotitoli che vengono mostrati dopo un decimo di secondo rispetto al tempo stabilito, saranno visti dallo spettatore in ritardo. Dunque la misura della percentuale di questi sottotitoli è di particolare interesse per valutare la precisione del sistema. In tabella sono quindi proposte le seguenti misure:

- Errore Medio (misurato in *frame*);
- Errore Quadratico Medio (misurato in *frame*), inteso come la media del quadrato delle differenze tra il valore misurato e il valore atteso;
- Ritardo Medio in millisecondi;
- Percentuale di sottotitoli riconosciuti con un ritardo maggiore o uguale ad un decimo di secondo;
- Percentuale di sottotitoli riconosciuti con un ritardo maggiore o uguale a mezzo secondo;
- Percentuale di sottotitoli riconosciuti con un ritardo maggiore o uguale ad un secondo;
- Percentuale di sottotitoli non riconosciuti;

Queste misurazioni sono state fatte solo per le situazioni per le quali hanno senso[2], ovvero con l'audio originale, con l'aggiunta di rumore e di riverbero. L'errore medio rivela che il sistema non è mai in anticipo nel riconoscere i sottotitoli, ma sempre in ritardo (i valori sono sempre positivi). Il caso peggiore si ha con l'aggiunta del rumore, per il quale le percentuali di sottotitoli

[2]Nei casi di variazione di velocità, tagli e pause, è impossibile avere una misura certa perchè i *frame* sono sfasati per costruzione. Mentre il caso del filtro passa basso risulta uguale all'originale, perchè in ogni caso la rappresentazione del segnale utilizza un intervallo di frequenze più ridotto.

in ritardo sono in media del 2%. Il lato positivo invece è che il ritardo medio non è mai superiore a un decimo di secondo, perciò si può dedurre che il sistema sarebbe affidabile in una ipotetica situazione reale.

4.6 Prestazioni

Per concludere questo capitolo, è interessante dare uno sguardo alle prestazioni del sistema, in termini di velocità di calcolo e memoria richiesta. I dati che verranno esposti a seguire sono da intendersi come approssimazioni pessimistiche delle reali prestazioni del software, perchè il codice potrebbe essere ulteriormente snellito e ottimizzato (a livello di programmazione).
Per riprendere lo stesso caso citato nella Sezione 3.2.6, un file audio di 16 minuti viene allineato in circa 50 secondi. In poche parole, si è osservato un incremento di velocità dell'85%.
Per avere dei dati più precisi, la nuova versione dell'algoritmo è stata integrata con l'input da microfono, per vedere come si comporta il sistema quando deve eseguire l'allineamento in tempo reale. In questo caso, si ha un utilizzo della CPU di circa il 9% e un'occupazione di memoria RAM di 10 MB.
In conclusione, tutti i dati rilevati suggeriscono che il sistema implementato fino a questo punto potrebbe essere impiegato anche su dispositivi mobili.

(a)

	2f2f_1	2f2f_2	bttf_1	bttf_2	fg_1	fg_2	mms_1	mms_2	pf_1	pf_2	Media
Originale	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
Riverbero	99,3	99,4	100,0	100,0	97,9	97,2	100,0	100,0	100,0	100,0	99,4
Rumore Bianco -27dB	97,9	97,0	99,8	100,0	97,9	99,5	98,6	100,0	99,4	98,7	98,9
Pause 6 secondi	97,4	98,3	98,0	97,8	97,5	98,6	98,2	98,5	98,1	99,3	98,2
Tagli 6 secondi	97,6	98,3	97,0	96,2	95,6	98,9	95,7	98,1	97,2	97,8	97,2
Velocità -5%	100,0	100,0	100,0	100,0	98,5	99,7	100,0	100,0	100,0	100,0	99,8

(b)

	2f2f_1	2f2f_2	bttf_1	bttf_2	fg_1	fg_2	mms_1	mms_2	pf_1	pf_2	Media
Originale											
Err. Med.	0,03	0,03	0,01	0,02	0,02	0,03	0,03	0,02	0,02	0,01	0,02
Err. Med. Quad.	0,20	0,10	0,03	0,05	0,06	0,07	0,06	0,10	0,04	0,03	0,07
Ritardo Med. (ms)	0,3	0,3	0,1	0,2	0,2	0,3	0,3	0,2	0,2	0,1	0,3
% ritardo 0.1 s	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
% ritardo 0.5 s	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
% ritardo 1 s	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
% insuccesso	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Riverbero											
Err. Med.	1,3	1,6	1,1	0,7	1,0	1,4	1,3	1,2	1,0	1,1	1,2
Err. Med. Quad.	5,5	8,6	21,9	1,8	3,3	68,7	5,4	4,7	3,8	3,6	12,7
Ritardo Med. (ms)	15,1	18,6	12,8	8,1	11,3	16,3	15,1	13,9	11,6	12,8	13,5
% ritardo 0.1 s	0,0	1,5	0,0	0,0	0,2	0,2	0,0	0,4	0,2	0,0	0,3
% ritardo 0.5 s	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
% ritardo 1 s	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
% insuccesso	0,7	0,6	0,0	0,0	2,1	2,8	0,0	0,0	0,0	0,0	0,6
Rumore B. -27dB											
Err. Med.	1,9	6,4	1,0	1,0	0,8	1,4	7,9	4,9	3,6	3,8	3,3
Err. Med. Quad.	428,2	1286,7	149,4	91,7	35,4	171,0	1527,0	1137,0	3,9	400,4	523,1
Ritardo Med. (ms)	22,1	74,3	11,4	11,6	9,3	16,3	91,7	56,9	41,8	44,1	37,9
% ritardo 0.1 s	4,6	4,6	0,6	0,9	1,5	0,0	3,3	1,5	2,3	2,9	2,2
% ritardo 0.5 s	0,7	0,7	0,0	0,0	0,0	0,0	2,6	1,5	0,2	0,0	0,6
% ritardo 1 s	0,5	0,0	0,0	0,0	0,0	0,0	1,5	0,8	0,0	0,0	0,3
% insuccesso	2,1	3,0	0,2	0,0	2,1	0,5	1,4	0,0	0,6	1,3	1,1

Figura 4.6: (a) Tabella riassuntiva dei test eseguiti con la nuova rappresentazione del segnale. (b) Statistiche approfondite sull'usabilità del sistema finale.

Capitolo 5

Conclusioni

In questo progetto di tesi è stata studiata un'applicazione pratica dell'allineamento di file audio. Si è partiti da un sistema base che prevedeva l'uso del DTW nella sua forma più elementare, per poi essere notevolmente migliorato, sia in termini di velocità che di efficienza. Il segnale audio è stato rappresentato attraverso la DFT. Inizialmente è stata utilizzata la rappresentazione più precisa possibile, ma ovviamente era di dimensioni troppo grandi per essere impiegata in un telefono cellulare. Perciò, è stata progettata una nuova rappresentazione di dimensioni notevolmente inferiori, le cui prestazioni sono tranquillamente paragonabili a quelle della rappresentazione iniziale.
In Figura 5.1 sono stati riassunti alcuni dei dati più rilevanti.
Per raggiungere questi risultati, sono stati presi degli spunti dai seguenti lavori: [2], [6], [7], [8], [10], [11] e [12].

5.1 Strumenti Utilizzati

Per la realizzazione di questo documento si è fatto uso di:

- Linguaggio LaTeX, bastato sul paradigma *What You See Is What You Mean* (WYSIWYM), cioè ciò che vedi è quello che intendi;
- Editor di testo TeXShop 2.28.

	Prima	Dopo
Percentuale di successo complessiva	90%	99%
Tempo per allineare due file audio di 16 minuti	315 s	50 s
Dimensione rappresentazione per 90 minuti di film	645 MB	10 MB
Ritardo medio sottotitoli	13 ms	18 ms

Figura 5.1: Confronto tra le prestazioni del sistema base e quello modificato.

Per l'implementazione e il test del sistema di allineamento sono stati utilizzati i seguenti strumenti:

- MacBook Pro 15", Core Duo 2.16 GHz, 2GB RAM;
- Mac OS X Snow Leopard 10.6.8;
- NetBeans 6.8 con Java 1.6 e gcc 4.2.1;
- MATLAB®;
- Audacity 1.2.5

5.2 Sviluppi Futuri

Per concludere, sembra appropriato dare una direzione futura al lavoro svolto finora.
Per prima cosa vale la pena di esplorare altre strade per migliorare l'algoritmo del DTW. La modifica che ha dato i risultati migliori è stata l'introduzione di un nuovo tipo di intorno, mentre la scelta di una nuova distanza ha avuto

un effetto abbastanza marginale. Perciò, sarebbe interessante studiare un nuovo intorno, che sia in grado, magari, di sfruttare ancora più informazioni (ad esempio il Tipo 5 proposto in [8]).
Un'altra possibilità è quella di concentrarsi sulla rappresentazione del segnale. Sembra abbastanza difficile ridurne ulteriormente la dimensione, ma è probabile che si possa aumentarne la precisione. Si potrebbe infatti studiare una nuova distribuzione delle 12 bande critiche esistenti; non è detto infatti che la spaziatura logaritmica sia la scelta migliore in assoluto. Inoltre, una cosa interessante che non è stata studiata in questo lavoro, sono i parametri *window length* e *hop size*, i quali incidono direttamente sulla precisione della rappresentazione.
Il lavoro svolto finora ha strettamente riguardato il sistema di allineamento, senza prendere in considerazione l'interfaccia grafica e la piattaforma di utilizzo finale del software. Perciò, un'altra strada che si può percorrere è quella di sviluppare la parte *front-end*, implementando una vera interfaccia grafica e trasportando il codice in un linguaggio adatto ai dispositivi mobili (ad esempio Objective C).
Come ultimo appunto, vale la pena citare una nuova funzionalità che si potrebbe implementare. Essa riguarda il funzionamento del sistema in caso venga rilevato un "salto" (vedi Sezione 3.2.4). Allo stato attuale, se il salto è più grande dell'estensione della finestra di allineamento, il sistema fallisce. Per risolvere questo problema, si potrebbe introdurre un meccanismo di ricerca della nuova posizione all'interno del file di riferimento, attraverso tecniche di *audio fingerprinting* (vedi [12]).

Appendice A

Sviluppo Software con NetBeans

Per lo sviluppo del sistema di allineamento è stato utilizzato un *Integrated Development Environment* (IDE), allo scopo di rendere più semplice e veloce il test e il *debug* dell'applicazione. In questa piccola sezione verrà illustrato come importare il codice sorgente nell'ambiente NetBeans.

A.1 Prerequisiti

Per prima cosa è necessario scaricare ed installare NetBeans (disponibile su [15]). E' importante scegliere la versione adatta sia allo sviluppo Java che allo sviluppo C++.
Quando NetBeans è installato correttamente, si può procedere con l'installazione delle librerie esterne, già menzionate nella Sezione 2.3. I pacchetti di FFTW e RtAudio si possono trovare nei rispettivi siti web ([13] e [14]), insieme ad alcune informazioni utili riguardo le specifiche di utilizzo. In generale però, l'installazione va eseguita con i comandi:

```
./configure
make
```

make install

Nel caso si avessero dei problemi nell'installare i pacchetti, si può sempre limitarsi a compilarli (istruzioni fino a make) ed includerli nella cartella del progetto, insieme al codice sorgente. In quest'ultimo caso però bisognerà effettuare una piccola modifica nel codice.
Per RtAudio ad esempio si dovrà sostituire la riga:

#include <RtAudio.h>

con la riga:

#include "rtaudio/RtAudio."

dove rtaudio/ è la cartella che contiene la libreria.

A.2 Importazione Interfaccia Grafica

L'interfaccia grafica Java è implementata nel file sottotitola.java. Per creare un progetto con questo file bisogna eseguire i seguenti passi:

1. Cliccare su File → New Project;

2. Selezionare "Java" sotto la voce categoria e "Java Project with Existing Sources" sotto la voce progetto;

3. Dare un nome al progetto, ad esempio "Interfaccia";

4. Alla voce "Source Package folders" selezionare la cartella in cui è contenuto il file sottotitola.java;

5. Fine procedura.

A.3 Importazione del Motore

Per il sistema di allineamento vero e proprio, quello sviluppato in C++, va creato un progetto a parte.
Se si possiede l'intera cartella del progetto, sarà sufficiente selezionarla cliccando su File → Open Project e si otterrà subito un progetto funzionante.
Se si possiedono solo i sorgenti invece, il procedimento è un pò più lungo.

1. Cliccare su File → New Project;
2. Selezionare "C/C++" sotto la voce categoria e "C/C++ Application" sotto la voce progetto;
3. Dare un nome al progetto, ad esempio "Movalign";
4. Non spuntare la casella "Create Main class";
5. A questo punto è stato creato un progetto vuoto.
6. Cliccando col tasto destro sul nome del progetto, selezionare New → C++ Source File;
7. Nominare questo file loadreference.cpp ed incollare al suo interno tutto e solo il contenuto del file omonimo che si possiede;
8. Cliccando col tasto destro sul nome del progetto, selezionare New → C++ Class;
9. Nominare questo file follower.cpp ed incollare al suo interno tutto e solo il contenuto del file omonimo che si possiede;
10. Insieme a quest'ultimo file .cpp, verrà creato anche un file nominato follower.h, nel quale bisogna incollare al suo interno tutto e solo il contenuto del file omonimo che si possiede;
11. Aprire la finestra di configurazione, cliccando con il tasto destro sul nome del progetto e poi Set Configuration → Customize;

12. Entrare nel sotto-menù "C++ Compiler";
13. Alla voce "Include Directories" inserire le cartelle nelle quali sono contenute le librerie esterne;
14. Alla voce "Additional Options" inserire[1]:
    - -g -lrtaudio nella casella "All Options";
    - -O2 -Wall -DHAVE_GETTIMEOFDAY -D__MACOSX_CORE__ rtaudio/RtAudio.o -lpthread -framework CoreAudio -framework CoreFoundation -lfftw3 -lm nella casella "Additional Options";
15. Nel sotto-menù "Run" alla voce "Arguments", inserire i parametri del programma, ad esempio: 1 2 files/fg_1.fft files/fg_1.tag files/fg_1.raw;
16. Fine procedura.

A.4 Esecuzione

Ora che il sistema è configurato correttamente per funzionare nell'ambiente NetBeans, si può fare una prova di esecuzione.
La prima cosa da fare è quella di lanciare l'interfaccia grafica; per farlo basta cliccare sul nome del progetto (ad esempio "Interfaccia") e selezionare "Run".
In modo del tutto analogo poi, si farà partire il sistema di allineamento.
A questo punto è tutto pronto. Quando il film comincia (ad esempio in televisione), basta premere il pulsante "Start" presente nell'interfaccia grafica e i sottotitoli verranno mostrati.

[1] Vedi [14] per le corrette informazioni di compilazione delle piattaforme Windows e Linux.

Bibliografia

[1] Christopher Raphael. Automatic Segmentation of Acoustic Music Signals Using Hidden Markov Models. Da *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 360-370, 1999.

[2] Nicola Orio e Diemo Schwarz. Alignment of Monophonic and Polyphonic Music to a Score. Da *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba, 2001.

[3] Yoram Meron. High Quality Singing Synthesis using selection based Synthesis. Scheme, tesi di dottorato, Università di Tokyo, 1999.

[4] Simon Dixon e Fabien Gouyon. Computational Rhythm Description. Da *7th International Conference on Music Information Retrieval (ISMIR)*, 2006.

[5] Sanjit Kumar Mitra. Digital signal processing: a computer-based approach, 3rd Edition, 2005. ISBN 0071255796.

[6] Beth Logan. Mel Frequency Cepstral Coefficients for Music Modeling. Da *ISMIR 2000, International Conference on Music Information Retrieval*, 2000.

[7] Simon Dixon e Gerhard Widmer. MATCH: A Music Alignment Tool Chest. Da *ISMIR 2005, 6th International Conference on Music Information Retrieval*, 2005.

[8] Lawrence Rabiner e Biing-Hwang Juang. Fundamentals of Speech Recognition, pp. 200-226, 1993. ISBN 0132858266.

[9] Hiroaki Sakoe e Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. Da *Transactions on ASSP*, pp. 43–49, 1978.

[10] Federico Demuro. Verifica Sperimentale di una Metodologia per la Segmentazione della Musica., Tesi di laurea, Università degli Studi di Padova, a.a. 2008-2009.

[11] Ferréol Soulez, Xavier Rodet e Diemo Schwarz. Improving polyphonic and poly-instrumental music to score alignment. Da *4th International Conference on Music Information Retrieval*, 2003.

[12] Pedro Cano, Eloi Batlle, Ton Kalker e Jaap Haitsma. A Review of Audio Fingerprinting. The Journal of VLSI Signal Processing. Volume 41, Numero 3, 271-284, DOI: 10.1007/s11265-005-4151-3.

Siti Consultati

[13] FFTW (Fastest Fourier Transform in the West), http://www.fftw.org/.

[14] RtAudio, http://www.music.mcgill.ca/~gary/rtaudio/.

[15] NetBeans, http://www.netbeans.org/.