UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Progettazione dell'interazione persona-computer basata su suoni non verbali per sistemi automatici di esecuzione musicale espressiva

Relatore: Prof. Sergio Canazza

Correlatore: Dott. Pierluigi Bontempi

Laureando: Claudio Costantini

ANNO ACCADEMICO 2022-2023
Data di Laurea 16/11/2023

Sommario

L'espressività in un'esecuzione musicale è la chiave per trasformare una serie di note in un'esperienza coinvolgente ed emozionante per l'ascoltatore. È attraverso l'abilità dell'interprete di infondere ogni nota con emozione, dinamica e sfumature che la musica prende vita.
In questa tesi è presentato un sistema per l'estrazione di features da un input audio e la generazione a partire da esse di parametri utilizzabili per il controllo di un modello di generazione computazionale di espressività operante su parti musicali inespressive in formato MIDI. Oltre alla presentazione dell'algoritmo, della sua implementazione e dei risultati ottenuti sinora se ne analizzano limiti e possibili sviluppi futuri.

Indice

Elenco delle figure

Capitolo 1

Introduzione: caratteristiche musicali

1.1 Il ruolo delle caratteristiche musicali

La musica è una delle forme d'arte più universali e potenti per l'umanità, avendo il potere di trasmettere emozioni e suscitare reazioni nell'ascoltatore. Ma come avviene questa magia emotiva della musica? Perché un certo brano può farci sentire felici, mentre un altro ci porta alla tristezza e alla malinconia? È qui che entrano in gioco le caratteristiche musicali (*features*). Tra queste caratteristiche ci sono il ritmo, la melodia, l'armonia, la dinamica e il tempo di una canzone. La loro specifica combinazione ha la capacità di creare una particolare atmosfera in una composizione musicale. È poi grazie anche alla bravura del musicista nell'interpretare in modo unico la partitura, senza stravolgere l'intenzione originale del compositore, che un'esecuzione musicale diventa coinvolgente, animata e ricca di emozioni.

Attraverso l'analisi computazionale è possibile scomporre le caratteristiche musicali in parametri misurabili. Ritmo, dinamica, armonia e altro possono essere quantificati e conseguentemente modificati in base all'interpretazione musicale desiderata. Ad esempio, l'aumento del tempo metronomico, è correlato a un aumento nella percezione di emozioni come felicità e sorpresa, contemporaneamente a una diminuzione della tristezza [5]. Questo è solo uno degli esempi di ricerca condotta nel campo del Riconoscimento della Musica e delle Emozioni (MER), riconducibile più in generale all'ambito dell'affective computing, che utilizza approcci psicologici e neuroscientifici per comprendere le complesse relazioni tra la musica e le emozioni umane.

Sebbene siano presenti sul mercato svariati prodotti per la sintesi ed elaborazione sonora, come riverberi, saturatori, equalizzatori, ecc., ancora non sono fruibili programmi dedicati all'espressività musicale automatizzata di composizioni scritte in formato MIDI [2] e perciò eseguibili da strumenti virtuali all'interno di una postazione

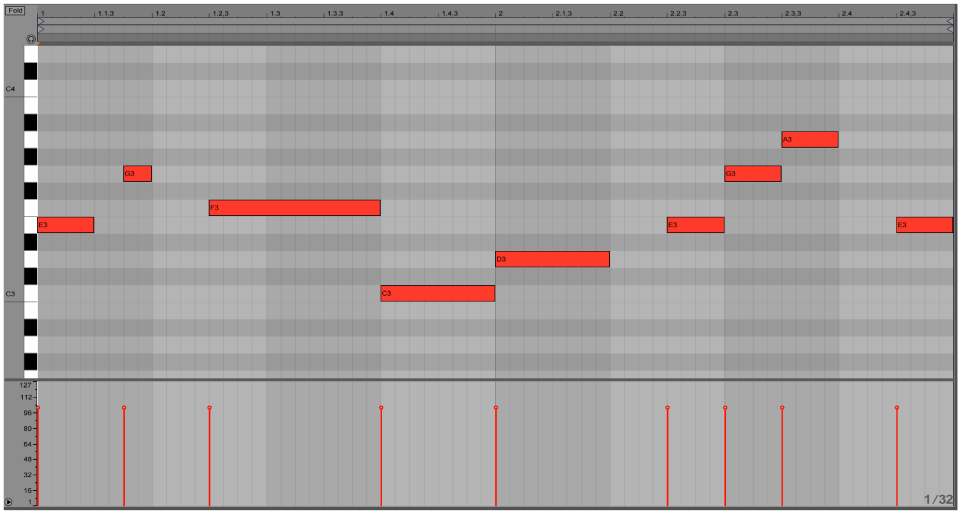

Figura 1.1: Note in formato MIDI quantizzate e a stesso volume

audio digitale (DAW - Digital Audio Workstation), con specifico riferimento alla produzione di popular music.

Lo scopo di questa tesi è presentare un sistema computazionale dedicato all'estrazione di caratteristiche da un input audio e alla generazione di parametri utili per controllare un modello di espressività applicato a parti MIDI inespressive. Questo sistema costituirà uno dei componenti di un progetto più ampio nell'ambito dell'interazione persona-computer, che si basa sull'utilizzo di suoni non verbali come input. Questo campo è ancora in gran parte inesplorato.

1.2 Le principali caratteristiche musicali

Come accennato in precedenza, sono le caratteristiche musicali a determinare se una composizione risulterà piatta e priva di anima o dinamica e vivace. Quindi, quali sono le features che hanno il maggiore impatto e che dovrebbero essere estratte, tenendo conto dell'obiettivo del progetto nel suo complesso?

Dato che il programma avrà il compito di conferire maggiore espressività a file MIDI inespressivi, ovvero file formati da note perfettamente quantizzate alla griglia temporale e aventi stesso volume (un esempio è riportato in Figura 1.1), i parametri estratti non potranno influire sulle altezze delle note, altrimenti si altererebbe la composizione musicale cambiandone il contenuto melodico e armonico. Di conseguenza, le modifiche da apportare devono concentrarsi su caratteristiche che non dipendono dalla frequenza delle note, come la melodia, che riguarda la sequenza lineare delle note, e l'armonia, che riguarda la combinazione simultanea di più note. Inoltre, dal punto di vista logico, è conveniente estrarre prima i parametri relativi alle

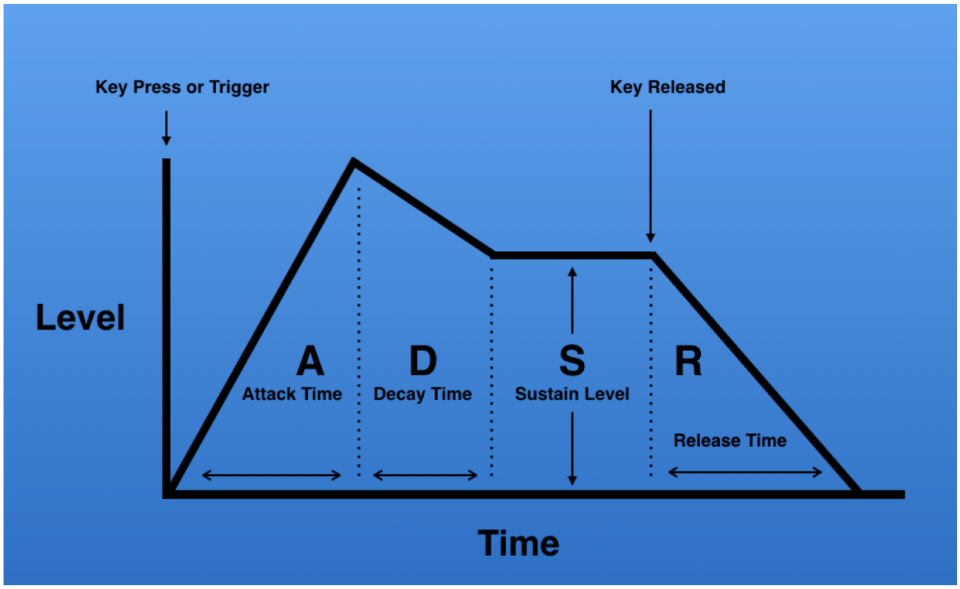

Figura 1.2: Inviluppo di una nota

caratteristiche di ciascuna nota e successivamente quelli che derivano dalle relazioni tra più note. Inizialmente, è importante focalizzarsi sull'inviluppo di una singola nota e successivamente estrarre dati relativi ai pattern di interazione tra più note. Tra le macrocaratteristiche da analizzare, due di particolare rilevanza sono il timing e il volume.

La temporalità in musica comprende un ampio spettro di concetti, tra cui ritmo, attacco delle note (*onset*), fine delle note (*offset*), misura e tempo metronomico. Alla luce delle considerazioni precedenti, è importante in primo luogo evitare i concetti che esistono solo in relazione a più note, come il ritmo e la misura. Infatti estraendo il tempo di inizio e fine di una nota si ha la possibilità di risalire anche agli altri principi musicali, cosa non sempre vera per il contrario. Inoltre, nella scrittura del codice, la conoscenza della posizione temporale di ogni nota ha un notevole vantaggio: questo permette di avere maggior facilità e precisione nell'estrazione di ulteriori features secondarie che potrebbero trovare applicazioni future, come l'altezza tonale (*pitch*).

Come prima cosa sono stati estratti i tempi di inizio e fine nota e si è ricavata la durata delle singole note. In secondo luogo, conoscendo a priori il tempo di registrazione dell'audio in battiti per minuto (BPM), è stato calcolato un parametro qualitativo riguardante la precisione temporale del segnale in ingresso rispetto alla griglia. Tale caratteristica indica quanto le note siano distanti, entro un certo range temporale, dai battiti che scandiscono il tempo della traccia audio. Questa caratteristica musicale è un aspetto fondamentale dell'interpretazione di un brano, in quanto influisce sull'andamento, sulla coerenza e sulla sensazione di una composizione. Un esecutore che ha una buona precisione ritmica è in grado di mantenere un tempo

costante, rispettare le pause e gli accenti musicali, e suonare le note esattamente quando sono previste, in modo che l'ascoltatore percepisca una performance fluida e ben sincronizzata. Tuttavia in alcuni generi musicali come il jazz e forme di musica etnica, la precisione ritmica può essere utilizzata in modo intenzionale per scopi espressivi, consentendo agli esecutori di suonare leggermente in anticipo o in ritardo rispetto al beat principale, creando così effetti ritmici e groove specifici.

In seguito, partendo nuovamente dai parametri di base delle lunghezze temporali delle note, si è estratto un parametro che concerne la modalità di esecuzione: si parla del legato e dello staccato. Una stessa melodia può essere suonata usando diverse forme di articolazione delle note come appunto legato, staccato, ma anche marcato, tenuto, staccatissimo e portamento. Questi termini si riferiscono al modo in cui una nota viene eseguita. Per esempio il marcato, il tenuto, lo staccatissimo e il portamento sono articolazioni spesso associate agli strumenti a corda, in particolare agli archi. A un livello di astrazione più elevato, ciò significa che ogni tipo di articolazione è caratterizzato da uno specifico andamento nell'inviluppo delle note. Ad esempio, le note staccate sono separate l'una dall'altra, hanno una durata breve, un attacco netto e un rilascio rapido, a differenza delle note legate, che hanno una durata maggiore e si susseguono senza soluzione di continuità. È stato osservato [3], che le note suonate in legato suscitano un maggior senso di calma e tristezza rispetto alle note in staccato, che creano più tensione ed energia. Gli altri tipi di articolazione non sono stati presi in considerazione in quanto sono riconducibili a particolari forme di staccato e legato, senza parlare del fatto che sarebbe molto complicato classificarli correttamente dal momento che non dipendono solo dai parametri temporali estremi delle note, cioè l'inizio e la fine di una nota, ma anche dalla combinazione di volume e tempo delle quattro sezioni che compongono l'inviluppo: attacco, decadimento, sostegno e rilascio. Questo modello di inviluppo ADSR, abbreviazione di attack, decay, sustain, release, è stato introdotto dall'ingegnere statunitense Robert Moog. Questa rappresentazione semplificata dell'inviluppo delle note è ampiamente utilizzata nella produzione musicale digitale, diventando uno standard diffuso per modulare la forma temporale dei suoni. Futuri sviluppi del codice potranno sicuramente includere gli altri tipi di articolazione più complessi.

Altre due caratteristiche, precedentemente menzionate, che dipendono dalla relazione temporale tra i suoni, sono il ritmo e il tempo di una traccia musicale. Il ritmo si manifesta attraverso l'organizzazione temporale di eventi e suoni, conferendo al brano quella sensazione di movimento e groove che può variare significativamente tra i diversi generi musicali. È l'elemento che organizza la melodia e l'armonia in una struttura musicalmente coerente. D'altro canto, il tempo rappresenta l'unità di

misura di questa organizzazione temporale ed è il fondamento su cui si basa il ritmo. Ogni brano ha un tempo specifico che crea una struttura di base per il ritmo. Questo concetto offre una sorta di "quadro" musicale, consentendo ai musicisti di rimanere in sincronia e agli ascoltatori di seguire il battito. Spesso nella popular music, mentre il ritmo può variare notevolmente all'interno di una canzone, creando dinamiche e groove unici, il tempo rimane generalmente costante per tutta la durata del brano, fornendo un punto di riferimento stabile. Fatte queste considerazioni, modificare il ritmo di una composizione risulterebbe in uno stravolgimento dell'identità musicale, mentre una variazione del tempo ne lascerebbe inalterata la sua integrità. Inoltre, potendo il ritmo variare molto all'interno di un brano, sarebbe difficile ricavarne dei parametri che lo possano descrivere in maniera adeguata e precisa. Alla luce di ciò, si è preso in considerazione solo l'estrazione del tempo, nella misura convenzionale di battiti per minuto.

Uno dei concetti di maggior spicco per quanto riguarda il volume è senza dubbio la dinamica. Questo concetto si riferisce alla variazione di volume all'interno di una composizione musicale, e perciò comprende anche l'aspetto temporale. Studi condotti in questo campo [6], hanno dimostrato come la dinamica sia un elemento significativo nella percezione dell'espressività emotiva e nell'apprezzamento di una composizione. Composizioni dinamiche sono generalmente più apprezzate ed espressive rispetto a composizioni statiche e piatte.

La dinamica può essere suddivisa in due categorie principali in base alla durata degli elementi considerati: la microdinamica e la macrodinamica. La microdinamica misura le variazioni di volume su suoni di breve durata temporale, come singole note. La macrodinamica, d'altra parte, riguarda le variazioni di volume in parti più estese di una composizione o nell'intera composizione stessa. Questa dimensione ha un impatto significativo sulla struttura e sull'andamento di un brano, contribuendo a definire il suo carattere complessivo. Sono state considerate entrambe le categorie nello sviluppo del codice, così da ottenere un numero maggiore di parametri da utilizzare.

1.3 Utilizzo delle caratteristiche musicali nella produzione digitale

Nel mondo della produzione digitale della musica, il rilevamento e l'uso delle caratteristiche musicali trova una vasta gamma di applicazioni. Attraverso la modifica di parametri relativi a certe feature musicali, è possibile plasmare il carattere di

Musical Term	Meaning	Symbol
pianississimo	very, very soft	*ppp*
pianissimo	very soft	*pp*
piano	soft	*p*
mezzo piano	moderately soft	*mp*
mezzo forte	moderately loud	*mf*
forte	loud	*f*
fortissimo	very loud	*ff*
fortississimo	very, very loud	*fff*

Figura 1.3: Tabella delle dinamiche musicali

un brano a proprio piacimento. Alcuni software analizzano e modificano diverse caratteristiche musicali allo stesso tempo.

Ad esempio, gli equalizzatori sono dispositivi che vanno ad agire sul livello delle frequenze di un segnale audio attraverso l'uso di filtri, apportando modifiche all'aspetto tonale e timbrico della musica. Sono particolarmente sfruttati nell'ambito del mixing e mastering. Anche i saturatori e distorsori agiscono sullo spettro del segnale audio, andando ad introdurre armoniche e cambiando il timbro del suono.

Un compressore è uno strumento che regola il volume del segnale audio. Esso può aumentare il livello delle parti audio al di sotto di una soglia specifica e diminuire quelle al di sopra di essa. D'altra parte, i modulatori di transienti operano sul volume, permettendo di incrementare o diminuire specifiche porzioni dell'inviluppo del segnale audio, rilevando i tempi di attacco, decadimento, sostegno e rilascio dei suoni.

L'Auto-Tune, molto diffuso nel genere pop moderno, è una tecnologia per la correzione dell'intonazione nelle registrazioni vocali [4]. Questa tecnologia rileva l'altezza tonale delle note e ne permette la modifica manuale o automatica in base alla scala musicale utilizzata.

Nelle DAWs odierne sono spesso presenti strumenti per l'auto slicing di file audio, ovvero la suddivisione di tracce audio in segmenti scanditi da specifici eventi sonori. Questi dispositivi si basano sul rilevamento dei transitori all'interno di una registrazione audio, cioè punti dell'audio in cui il livello del volume subisce repentini cambiamenti, come un brusco aumento o diminuzione.

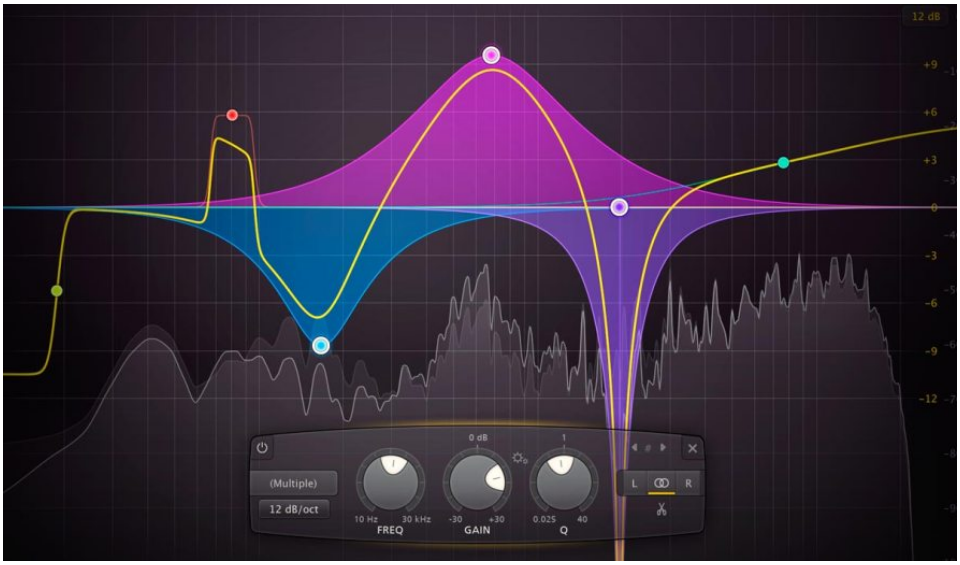

Figura 1.4: Equalizzatore digitale FabFilter Pro-Q 3

Software dedicati alla regolazione della durata temporale di un segnale audio senza alterarne il tono (*Time Stretching Plugins*) sono molto usati nella produzione musicale digitale. Questi plugin cercano di mantenere inalterato il contenuto in frequenza del segnale audio mentre ne viene estesa o compressa la durata.

In conclusione tutti questi software si avvalgono di tecniche di analisi delle diverse caratteristiche musicali di una traccia audio e utilizzano i parametri ricavati da queste analisi per l'elaborazione e la modifica dell'audio.

Capitolo 2

Scelte implementative

2.1 Rilevamento dell'attacco

Il rilevamento dell'attacco delle note musicali, noto come *onset detection*, è un elemento cruciale nell'analisi delle caratteristiche musicali. Attraverso l'uso di Python, si è sviluppato un algoritmo per identificare con la maggior precisione possibile, il momento in cui una nota inizia e finisce di suonare in una registrazione audio. Python offre una vasta gamma di librerie e framework per il trattamento del suono e l'analisi musicale. Sono state utilizzate due librerie dedicate all'analisi dell'audio, Librosa e PyDub, che forniscono funzioni utili anche per l'estrazione dell'attacco delle note per l'appunto.

Un approccio comune coinvolge l'uso di algoritmi di analisi spettrale per monitorare le variazioni di energia nell'audio. Quando una nota inizia, si verifica un aumento significativo di energia nella registrazione, che può essere catturato attraverso l'analisi spettrale. Le caratteristiche temporali e spettrali di queste variazioni di energia possono essere utilizzate per determinare con precisione il momento in cui l'attacco della nota si verifica.

Uno dei problemi che si possono incontrare tramite questo tipo di approccio è il fatto che le variazioni di energia sono presenti anche in parti dell'audio non volute nel rilevamento dell'attacco, come clic indesiderati o rumori di fondo dovuti a una registrazione non impeccabile, creando quindi falsi positivi. Anche la modalità di esecuzione di una nota come il vibrato rende questo compito più difficile, dal momento che una stessa nota se suonata in tal modo produce forti oscillazioni nell'energia spettrale, facendo sì che vengano rilevate più note quando in realtà ne è suonata solo una.

Il materiale iniziale su cui sono stati testati i vari algoritmi riportati qui di seguito è formato da sei estratti di chitarra elettrica, suonata dal Dott. Pierluigi Bontempi,

usando diversi tipi di espressività che vanno ad incidere su caratteristiche musicali quali: la dinamica, l'attacco delle note rispetto alla griglia temporale e il modo in cui sono state maggiormente eseguite le note, ovvero se in legato o staccato. Tutti i campioni sono stati registrati a un tempo di 80 bpm lasciando 8 beat vuoti all'inizio, in formato WAV mono, con frequenza di campionamento di 48 kHz e risoluzione di 24 bit.

Le funzioni per il rilevamento dell'onset fornite da Librosa si sono rivelate meno efficaci e precise di quanto sperato. Nonostante la loro semplicità e intuitività d'uso, i risultati ottenuti hanno prodotto un numero eccessivo di falsi positivi, in gran parte dovuti a clic indesiderati presenti nelle registrazioni. Per superare questa sfida, è stato necessario sviluppare un nuovo algoritmo più resistente ai rumori di breve durata temporale e alle variazioni dell'intensità.

Inizialmente, si è tentato di adattare l'algoritmo di rilevamento dell'attacco noto come *SuperFlux*, riconosciuto come uno stato dell'arte in questo campo [1]. Questo algoritmo permette di ridurre al massimo i falsi positivi rilevati per colpa del vibrato delle note. Il problema riscontrato con questo approccio è stato ancora una volta quello di essere troppo sensibile a clic indesiderati, nonostante le buone prestazioni con le note suonate in vibrato.

Per risolvere questi problemi e ottenere un algoritmo altamente robusto contro i rumori non voluti, in grado di individuare solo gli attacchi reali delle note per quanto possibile, sono state eseguite le seguenti fasi:

1. È stato applicato un filtro passa alto ed uno passa basso per isolare le frequenze utili del segnale audio nell'intervallo tra 80 Hz e 5000 Hz, intervallo di frequenze più che sufficiente per l'analisi del suono prodotto da una chitarra elettrica [7].

2. È stata calcolata la potenza dello spettro audio in scala decibel attraverso la "Trasformata di Fourier a breve termine" (STFT).

3. Per ciascuna finestra temporale in cui è stata calcolata la potenza spettrale, è stato selezionato l'elemento con la massima energia tra i vari intervalli di frequenza (frequency bin). Questo processo ha creato una lista contenente i massimi valori di potenza spettrale e le relative frequenze, eliminando i valori meno rilevanti, ad esempio quelli causati da clic indesiderati.

4. I valori delle potenze spettrali sono stati normalizzati in modo da rientrare nell'intervallo compreso tra -1 e 1.

Questo nuovo algoritmo ha successivamente utilizzato il principio di rilevare gli sbalzi di energia spettrale più marcati per determinare la presenza o l'assenza di un

onset. Sono state applicate contemporaneamente tre condizioni a mag, l'array delle potenze spettrali massime, per ogni istante temporale i:

- $mag[i] > THRESHOLD$
- $mag[i] - mag[i-1] > MIN\_INCREASE$
- $onset[i] - onset[i-1] \geq MIN\_TIME$

dove $THRESHOLD$ è la soglia minima di potenza al di sotto della quale non si possono rilevare gli attacchi delle note, $MIN\_INCREASE$ è la minima differenza di potenza necessaria affinché $mag[i]$ possa essere considerato un onset, MIN_TIME, espresso in secondi, è il tempo minimo che deve presentarsi tra due attacchi di nota vicini per eliminare eventuali rilevazioni di falsi positivi. I parametri riportati in questo capitolo sono discussi più nel dettaglio nel Capitolo 3. In definitiva, questo nuovo algoritmo si è dimostrato complessivamente più efficace nel rilevamento degli onset, riducendo i falsi positivi e fornendo risultati più accurati nelle registrazioni di chitarra, sebbene sia meno preciso con le note suonate in vibrato.

Per rilevare i tempi di fine nota è stato utilizzato il medesimo processo imponendo solo due condizioni all'array delle potenze spettrali comprese tra due onset, cioè $mag\_note = mag[onset[i] : onset[i+1]]$, per ogni onset i precedentemente calcolato:

- $mag\_note[j] \leq MIN\_THRESHOLD$
- $mag\_note[j] - mag\_note[j-1] \leq MIN\_DIFFERENCE$

dove $MIN\_THRESHOLD$ è la soglia minima di potenza sopra la quale non vengono rilevati gli attacchi delle note e $MIN\_DIFFERENCE$ è la differenza minima tra le potenze adiacenti affinché il tempo corrispondente a $mag\_note[j]$ possa essere considerato un onset. La terza condizione relativa a offset vicini non è stata necessaria, dal momento che ogni volta che viene rilevato il tempo di fine nota si passa automaticamente ad analizzare le potenze comprese nell'intervallo formato dai due onset successivi. Infine è stata calcolata la lunghezza temporale di ogni nota rivelata, espressa in secondi.

In Figura 2.1 è riportata una comparazione dei risultati ottenuti su un estratto di chitarra, dall'algoritmo qui presentato, chiamato *MagOnsetDetection* (a sinistra) e da *SuperFlux* (a destra). Si nota come alcuni attacchi delle note rilevati da *SuperFlux* siano in realtà dei falsi positivi, perché la funzione è più sensibile ai clic indesiderati, a differenza dell'algoritmo qui sopra sviluppato.

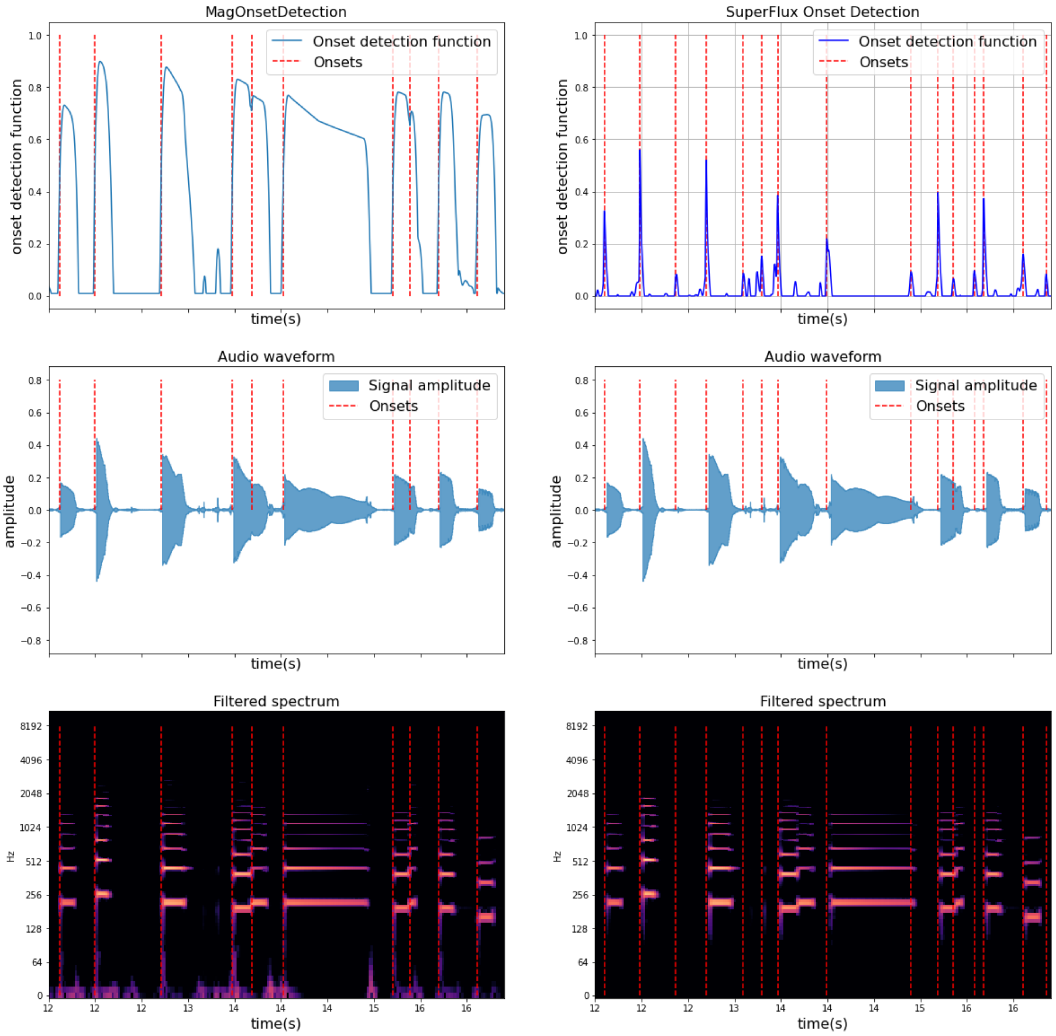

Figura 2.1: Funzioni per il rilevamento dell'attacco delle note

Una precisazione importante riguarda il funzionamento di *MagOnsetDetection*. Questo approccio si dimostra efficace quando si analizzano estratti di chitarra, poiché le note di chitarra presentano attacchi ben definiti e non coprono un ampio spettro di frequenze. Tuttavia, se si devono individuare gli attacchi in note di strumenti più complessi che producono suoni caratterizzati da inviluppi irregolari e attacchi non chiaramente definiti, è preferibile adottare un metodo basato sul rilevamento di variazioni significative nella frequenza. In questo caso, un approccio simile a quello descritto nella Sezione 2.4 diventa più adatto. In questa sezione si esamina il metodo impiegato per identificare il tono delle note. Una rivisitazione di tale strategia potrà offrire una soluzione più aderente a suoni complessi.

2.2 Precisione ritmica

Una volta ricavati gli onset e offset delle note, si è estratto un parametro di tipo qualitativo che descrive la precisione ritmica, cioè se un'esecuzione musicale è in anticipo, in ritardo o a tempo. Per ricavare tale parametro, è necessario conoscere a priori dove cadono i battiti del brano, dal momento che occorre calcolare la differenza di tempo tra i battiti e gli attacchi delle note distanti non più di un certo intervallo di tempo dai beat. Sapendo il tempo di registrazione della traccia audio, è stata prima ricavata la durata temporale di un singolo battito metrico

$$beat\_duration = \frac{60}{BPM} \tag{2.1}$$

dove BPM è il tempo, in battiti per minuto, del brano. Poi è stata creata la griglia temporale dei battiti, denominata $beats$. Successivamente è stato imposto un vincolo temporale per capire quali onset debbano essere presi in considerazione per il calcolo della precisione ritmica. Solo gli attacchi di nota a distanza di ± 60 ms dai battiti sono stati presi in considerazione e a loro sottratti. Le differenze ottenute sono state utilizzate per determinare se nel complesso la performance musicale fosse in anticipo, in ritardo, a tempo, o sia in anticipo che in ritardo rispetto al beat (tempo rubato), imponendo le seguenti condizioni:

- $ahead\_percentage > 50$
- $behind\_percentage > 50$
- $neutral\_percentage > 50$

dove $ahead\_percentage$ è la percentuale di note suonate in ritardo, $behind\_percentage$ è la percentuale di note suonate in anticipo, $neutral\_percentage$ è la percentuale di note suonate sul beat. Quest'ultima variabile è la percentuale delle differenze ottenute precedentemente, il cui valore assoluto è inferiore o uguale a 20 ms, altrimenti le differenze positive sono memorizzate in $ahead\_percentage$ e quelle negative in $behind\_percentage$.

Se la prima condizione è verificata, significa che l'esecuzione è in ritardo rispetto al tempo della traccia. Se la seconda condizione è verificata, l'esecuzione è in anticipo. Se la terza condizione è verificata, l'esecuzione è a tempo con la musica. Se nessuna di queste condizioni è verificata, significa che il tempo è rubato, il che indica che ci sono parti in cui l'esecuzione presenta ritardi, anticipi e tempistiche corrette rispetto al brano in una misura tale che nessuno dei tre tipi di esecuzione prevale sugli altri due. Ovviamente, è possibile modificare i due vincoli temporali di

± 60 ms e 20 ms, per rendere l'analisi sulla precisione ritmica più stringente oppure più flessibile.

2.3 Classificazione delle note: legato e staccato

Nella classificazione delle note, si sono utilizzati nuovamente i tempi di attacco e fine nota, insieme all'array delle potenze spettrali del segnale audio. Per determinare se una nota viene suonata in modo legato o staccato, è stato necessario valutare, per ciascun intervallo compreso tra due onset consecutivi, se la potenza spettrale è al di sotto o al di sopra di una specifica soglia che indica la presenza o l'assenza di un momento di silenzio e se la durata della nota compresa tra i due onset è inferiore a

$$MAX\_NOTE\_LENGTH = \frac{60}{2 \cdot BPM} + \sigma \tag{2.2}$$

La scelta della soglia è ricaduta sul valore precedentemente utilizzato per il rilevamento dei tempi di attacco delle note, noto come variabile $THRESHOLD$. Quando si scende al di sotto di questa soglia tra una nota e l'inizio della successiva, significa che esiste un momento in cui la potenza spettrale è molto bassa, indicando una pausa tra le due note. Inoltre se la durata temporale della nota è inferiore di MAX_NOTE_LENGTH, tale nota viene classificata come suonata in staccato. Tuttavia, se la potenza spettrale rimane costantemente sopra la soglia del silenzio e la durata della nota è superiore a MAX_NOTE_LENGTH, la nota sarà classificata come eseguita in legato.

Infine, si sono calcolate le percentuali delle note suonate in legato e staccato. Questi dati sono stati riportati in un database così da facilitare l'eventuale calcolo delle correlazioni tra le caratteristiche musicali, ad esempio quella tra il tipo di esecuzione di una nota e la sua dinamica, riportata in Figura 2.2.

2.4 Rilevamento dell'intonazione

Un parametro secondario, che potrebbe essere utile in sviluppi futuri, è l'altezza tonale delle note. Per ricavare questo parametro, per prima cosa è stata usata la funzione *librosa.pyin*, molto efficace per rilevare le frequenze fondamentali di un segnale audio. Questa funzione si basa su un algoritmo probabilistico conosciuto nello stato dell'arte con il nome di *Probabilistic YIN* [8]. Tale algoritmo è l'evoluzione del ben noto algoritmo chiamato *YIN*, con l'introduzione di un approccio probabilistico.

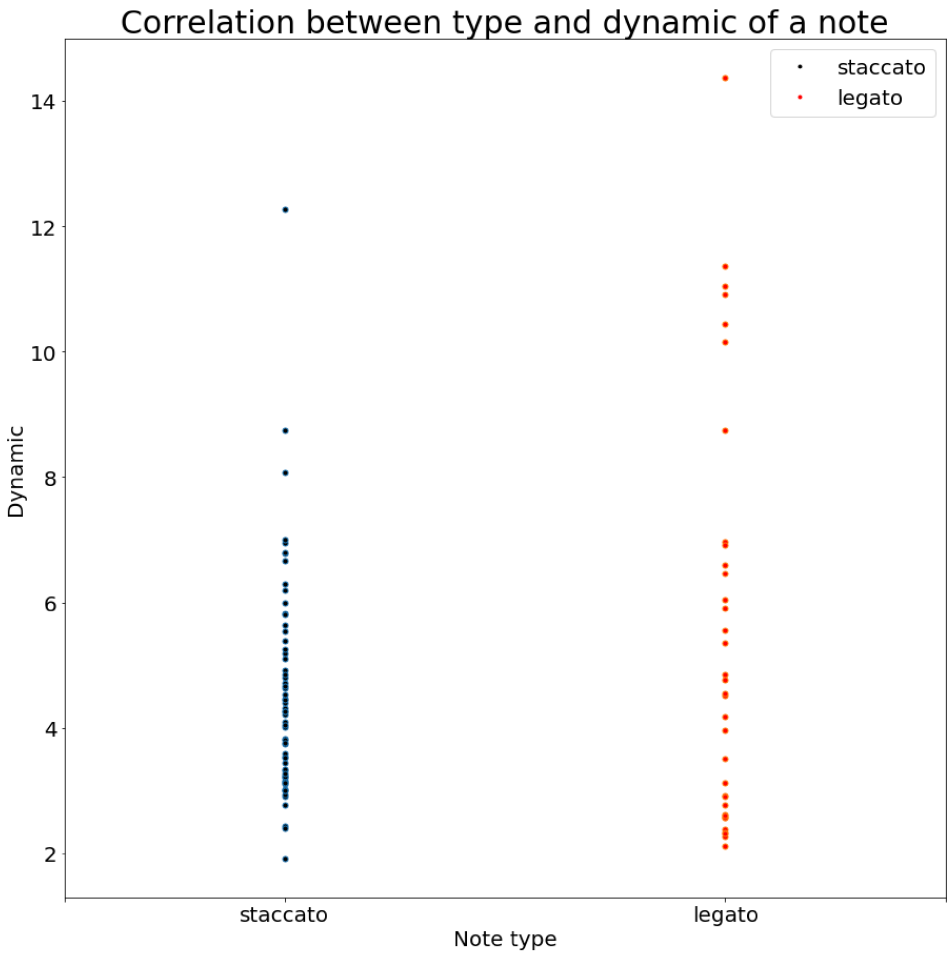

Figura 2.2: Correlazione tra tipo di nota e dinamica

Il segnale filtrato è stato dato come input a *librosa.pyin*. Una volta ottenuta la lista delle frequenze fondamentali $f0$, espresse in Hertz, per ogni nota, utilizzando i tempi di onset e offset, si sono memorizzate le sole frequenze diverse dal valore NaN. Infatti nel caso in cui la potenza spettrale sia troppo bassa, la funzione *librosa.pyin* restituisce il valore NaN, poiché in tale porzione di audio non vengono rilevati suoni intonati. Dal momento che gli attacchi delle note rilevati spesso avvengono qualche millesimo di secondo prima degli effettivi onset, nei quali la potenza spettrale è abbastanza alta da permettere l'individuazione delle frequenze fondamentali, è stato necessario svolgere questo processo di rimozione dei valori nulli a inizio nota.

Per decidere quale sia con precisione l'altezza tonale delle note, si sono considerate, per ogni nota, le frequenze fondamentali a partire dal primo valore diverso da NaN dall'attacco della nota, fino al primo valore pari a NaN. Per ogni nota, si è creata una lista chiamata $freq\_window$, che memorizza le frequenze fondamentali in valori decimali. Questi valori sono stati approssimati a numeri interi, e poi ne è stata calcolata la moda così da ottenere il valore della frequenza fondamentale che

compare in numero maggiore nell'arco di tempo sopra stabilito. Dopo aver calcolato il valore della frequenza fondamentale di ogni nota, è stata impiegata la funzione *librosa.hz_to_note* per convertire la frequenza nella corrispondente nota musicale in notazione scientifica. Questa conversione è stata effettuata con una precisione fino ai centesimi di tono.

L'algoritmo si comporta bene con la chitarra elettrica, ma potrebbe presentare problemi per strumenti con un maggiore contenuto in frequenza. Per questi strumenti sarebbe opportuno dare in input alla funzione *librosa.pyin* il segnale originale, senza filtrarlo, e cambiare il metodo statistico di selezione della frequenza fondamentale della nota in maniera adeguata.

2.5 Microdinamica e macrodinamica

Nell'analisi della dinamica sono stati svolti due passaggi principali: prima si è calcolata la dinamica di ogni singola nota (microdinamica), dopo sono stati calcolati due tipi di dinamica relativi a parti più ampie della traccia audio (macrodinamica). Come indice per la microdinamica si è usato il rapporto tra il livello di picco reale e la media dei valori quadratici medi (RMS) della nota. Il livello di picco reale tiene conto delle intensità dei singoli sample dell'audio, mentre il valore RMS medio è una misura dell'intensità acustica media di una porzione di audio. Nel caso studio la porzione di audio è costituita dalle singole note di chitarra, scandite dai tempi di attacco e fine nota precedentemente trovati. Le note estremamente dinamiche mostrano un elevato valore di picco rispetto a valori RMS bassi, mentre le note prive di dinamica presentano un andamento piatto dell'inviluppo, facendo risultare il rapporto tra picco e valore RMS medio pari a 1.

I valori delle ampiezze audio normalizzate, contenuti nella lista $signal$, sono stati utilizzati per calcolare i valori RMS delle note attraverso la funzione *librosa.feature.rms*. Per individuare le singole note, è stato adottato il seguente metodo: se tra due onset consecutivi non è presente un offset, la durata della nota corrisponde all'intervallo scandito dai due onset; altrimenti, la sua durata è determinata dal tempo che intercorre tra l'onset e il successivo offset. Per ciascun segmento del segnale audio, sono stati calcolati il valore di picco, ottenendo la media dei 3 picchi più alti, e la media dei valori RMS. Questi valori sono stati successivamente registrati come rapporto nella lista $dynamics$. Le dinamiche delle note sono state poi inserite nel database.

Per ciò che concerne la macrodinamica, sono stati calcolati due tipi di indici: il rapporto tra il valore di picco e il valore RMS medio dell'intera traccia audio

("Peak to Loudness Ratio"), e il rapporto tra il valore di picco e valore RMS medio su finestre dalla durata di 3 secondi ("Peak to Short-Term-RMS Ratio"). Entrambi i valori riguardano parti più lunghe dell'audio rispetto alla dinamica delle singole note calcolata precedentemente, e restituiscono un indice per determinare il comportamento della dinamica della traccia nella sua interezza.

Per calcolare il Peak to Loudness Ratio (PLR), sono stati identificati il valore massimo delle ampiezze in valore assoluto e il valore RMS medio del segnale audio. Questi valori sono stati convertiti in "decibel Full Scale" (dBFS), un'unità di misura che indica il livello del segnale audio in relazione al massimo livello possibile, pari a 0 dBFS. Per ottenere questi valori, sono state utilizzate le librerie PyDub e Librosa. Inizialmente, con l'ausilio di PyDub, è stata calcolata la lista delle ampiezze non normalizzate del segnale audio. Successivamente, la lista delle ampiezze è stata normalizzata dividendo i valori in essa contenuti per il massimo valore rappresentabile con la profondità di bit del segnale audio, comunemente conosciuta come "sample width". La sample width è direttamente correlata alla quantità di dettaglio o dinamica che un segnale audio possiede: una profondità di bit maggiore implica una migliore capacità di rappresentare la dinamica e i dettagli del segnale. La massima ampiezza che un sample di un segnale audio può avere, considerando una certa sample width, è definita come segue:

$$max\_value = 2^{(8 \cdot sample\_width - 1)} \tag{2.3}$$

Per la conversione dei valori di picco e RMS medio in dBFS, si è usata la rappresentazione logaritmica

$$value\_dBFS = 10 \cdot \log_{10}(value) \tag{2.4}$$

Il calcolo finale del PLR è dato dalla sottrazione tra il valore di picco reale e il valore RMS medio.

Il Peak to Short-Term-Loudness Ratio (PSR), similmente al PLR, viene calcolato sottraendo il valore di picco reale dal valore RMS medio. La principale differenza è che i calcoli del valore di picco e del valore RMS medio vengono effettuati su finestre temporali di 3 secondi, con un passo temporale di 187.5 ms. È stato eseguito un ciclo for sull'intero segnale audio per eseguire questi calcoli, risultando in una lista PSR che contiene i valori delle dinamiche. Valori elevati di PSR indicano una notevole variazione nel livello del segnale audio a breve termine, mentre periodi di segnale audio con intensità stabile presentano valori bassi di PSR.

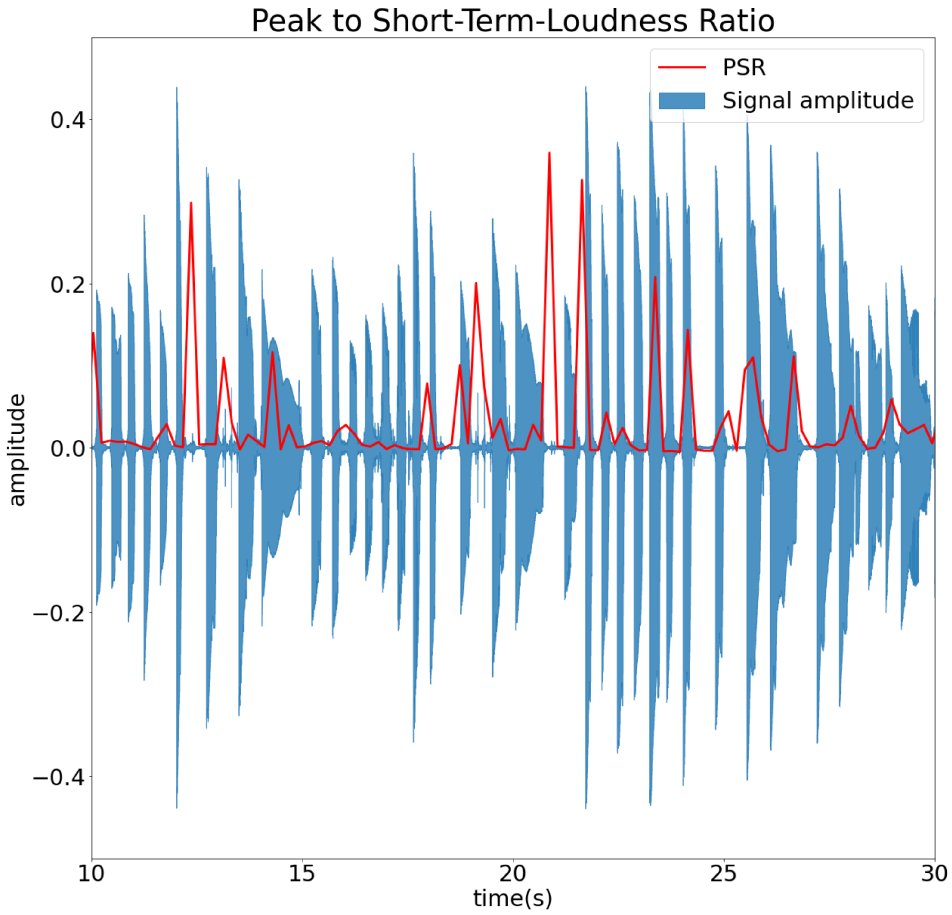

Figura 2.3: Andamento del PSR

2.6 Registrazione audio vocale

Infine, è stata sviluppata la parte di codice dedicata alla registrazione dell'audio. Questo passaggio è fondamentale in quanto da qui saranno estratti i parametri che in futuro verranno utilizzati per aggiungere espressività a composizioni MIDI inespressive. L'esecuzione di questo blocco di codice genera una finestra interattiva in cui l'utente può inserire il tempo in battiti per minuto, accettando valori compresi tra 60 bpm e 200 bpm. Questo valore è necessario per attivare il metronomo durante la registrazione. Il valore inserito viene memorizzato nella variabile BPM e successivamente utilizzato per valutare la precisione ritmica della traccia audio.

Una volta specificato il tempo desiderato, facendo clic sul pulsante "*Start*", si avvia la registrazione e il metronomo inizia a suonare. Contemporaneamente, un timer a schermo mostra il tempo di registrazione in corso. Per interrompere la registrazione, è sufficiente fare clic sul pulsante "*Stop*". In questo modo, la registrazione, il metronomo e il timer si fermano, e il file audio viene salvato in formato WAV mono con un campionamento di 44.1 kHz. Il file viene nominato

"*output*". È importante notare che il tempo massimo di registrazione è illimitato. Tuttavia, se il numero inserito per il tempo del metronomo non rientra nell'intervallo compreso tra 60 e 200, la registrazione non si avvia.

Questa parte di codice è ancora in fase iniziale di sviluppo e può essere ulteriormente migliorata per rendere l'interazione con l'utente più coinvolgente. Inoltre, si potrebbe considerare di offrire maggiore flessibilità consentendo all'utente di personalizzare il tipo di registrazione desiderato. Ad esempio, si potrebbe considerare di offrire opzioni per la selezione del formato di registrazione e della frequenza di campionamento, consentendo così agli utenti di adattare l'applicazione alle loro esigenze specifiche.

Capitolo 3

Discussione del codice

In questo capitolo, verranno esaminati più approfonditamente alcuni degli algoritmi precedentemente discussi, focalizzandosi sulle sezioni di codice più significative. Non ci si soffermerà sulle parti meno rilevanti o su quelle già discusse dettagliatamente in precedenza. Durante la discussione, si esploreranno le decisioni prese, verranno identificate le sfide connesse a tali scelte e verranno condivise alcune prospettive sui possibili sviluppi futuri finalizzati al miglioramento del codice.

3.1 MagOnsetDetection

In questa sezione sono presentati nel dettaglio i passaggi fondamentali nella realizzazione dell'algoritmo per il rilevamento degli onset.

```
n_fft = 2048
hop_length = 64

stft = librosa.stft(filtered_signal, n_fft=n_fft, hop_length=
    hop_length)

magnitude = librosa.amplitude_to_db(np.abs(stft)**2)

mag = np.zeros(magnitude.shape[1])
for i in range(0, magnitude.shape[1]):
        current_peak_bin = np.argmax(magnitude[:,i])
        mag[i] = magnitude[current_peak_bin, i]
```

In questa porzione di codice viene usato il segnale filtrato per calcolare la STFT e successivamente trovare le potenze spettrali del segnale audio in decibel. Il ciclo for, eseguito per ogni istante temporale di $magnitude$, salva nell'array mag la potenza spettrale massima relativa a tale finestra temporale. I parametri n_ftt e hop_length,

rispettivamente il numero di campioni nella finestra di analisi temporale utilizzata per calcolare la trasformata di Fourier a breve termine e il passo temporale, sono stati ricavati per via euristica.

La seguente parte di codice è la funzione che consente il rilevamento degli onset:

```
THRESHOLD = np.mean(librosa.feature.rms(y=signal)) * 10
MIN_INCREASE = THRESHOLD / 40
MIN_TIME = 0.04 #seconds
DELAY = 0.007 #seconds
times = librosa.times_like(mag, sr=sr, hop_length=hop_length)
frame_mag = duration / mag.size #duration of a mag frame
onsets = [] #onset time in seconds
onset_frame = [] #onset index in frames
dummy_onsets = []

for i in range(1, mag.size):
    current_mag = mag[i]
    prev_mag = mag[i-1]
    mag_increase = current_mag - prev_mag

    if (current_mag > THRESHOLD
    and mag_increase > MIN_INCREASE):
        dummy_onsets.append(times[i])

        if (len(onsets) == 0
        or times[i]-dummy_onsets[-2] >= MIN_TIME):
            onsets.append(times[i]+DELAY)
            onset_frame.append(i+int(DELAY/frame_mag))
```

Anche in questo caso i parametri sono stati ricavati empiricamente, tenendo conto del fatto che l'algoritmo debba funzionare per tutti gli estratti di chitarra elettrica, ognuno dei quali può presentare livelli di volume e modalità di esecuzione differenti. Il migliore parametro da tenere in considerazione è il valore RMS medio della traccia audio, così da ottenere valori di $THRESHOLD$ adattati in base al segnale audio che si sta analizzando. L'uso della costante 10 come moltiplicatore per il valore medio dell'RMS ha dimostrato di produrre risultati ottimi su tutti e sei gli estratti di chitarra testati, così come la costante 40 per il calcolo di $MIN\_INCREASE$. La variabile $THRESHOLD$ è usata come punto di partenza per il calcolo di altre variabili, ad esempio quelle per il rilevamento degli offset.

L'unico problema riscontrato è stato il rilevamento di alcuni falsi positivi nelle note suonate con un vibrato molto pronunciato. In questi casi, si è scelto di fare un trade-off, favorendo il rilevamento delle note regolari, scartando i falsi positivi

causati da clic indesiderati, a discapito di quelle in vibrato. Nel complesso, l'utilizzo di *MagOnsetDetection* si è dimostrato preferibile rispetto all'alternativa *SuperFlux*.

La funzione *librosa.times_like* restituisce il vettore dei tempi, denominato $times$, associato al vettore delle potenze spettrali mag. La variabile $time\_frame\_mag$ rappresenta l'intervallo temporale tra due campioni di mag, espresso in secondi. Le condizioni presenti nel primo blocco condizionale, all'interno del ciclo *for*, servono per identificare gli istanti temporali che possono essere considerati onset. Le condizioni nel secondo blocco condizionale gestiscono il caso limite in cui si rileva il primo onset in un segnale audio. Inoltre, tali condizioni consentono di scartare eventuali falsi positivi derivanti da onset troppo vicini tra loro, ossia con una distanza inferiore a 30 ms. Questo è un tempo assolutamente accettabile per registrazioni di chitarra elettrica. Tuttavia, è importante notare che questo valore di soglia può essere ridotto se si sta lavorando su registrazioni di chitarra elettrica in cui le note vengono suonate molto velocemente.

Un aspetto significativo consiste nell'aggiunta della costante $DELAY$ al tempo relativo del campione i. Questa costante, che equivale a 0.007 secondi, è introdotta poiché la funzione di rilevamento degli onset comporta un ritardo minimo di qualche millisecondo su tutti gli attacchi delle note. Integrando questa costante, è possibile compensare in gran parte i ritardi, evitando inoltre di tagliare la parte iniziale degli onset.

La seguente parte è la funzione dedicata al rilevamento degli offset:

```
MIN_THRESHOLD = THRESHOLD * 0.5
MIN_DIFFERENCE = -MIN_THRESHOLD * 0.05
offset_frame = []
offsets = []

for i in range(0, onsets.size):
    if i == onsets.size-1:
        mag_note = mag[onset_frame[i]: -1]

        for j in range(1, mag_note.size):
            current_mag = mag_note[j]
            prev_mag = mag_note[j-1]
            mag_diff = current_mag - prev_mag

            if (current_mag <= MIN_THRESHOLD
            and mag_diff <= MIN_DIFFERENCE):
                ind = j + onset_frame[i]
                offset_frame.append(ind)
                offsets.append(times[ind])
                break
```

```
    else:
        mag_note = mag[onset_frame[i]: onset_frame[i+1]]
        for j in range(1, mag_note.size):
            current_mag = mag_note[j]
            prev_mag = mag_note[j-1]
            mag_diff = current_mag - prev_mag

            if (current_mag <= MIN_THRESHOLD
            and mag_diff <= MIN_DIFFERENCE):
                ind = j + onset_frame[i]
                offset_frame.append(ind)
                offsets.append(times[ind])
                break
```

Anche in questo caso, $MIN\_THRESHOLD$ e $MIN\_DIFFERENCE$ sono stati determinati tramite esperimenti e dimostrano un alto grado di efficacia in tutti gli estratti di chitarra. La prima parte condizionale gestisce il caso dell'ultimo attacco di nota, poiché solo in questa situazione il vettore delle potenze compreso tra due onset, notato come mag_note, sarà costituito dalle potenze nell'intervallo tra l'ultimo onset e la fine del segnale audio. Per tutti gli altri scenari, il blocco condizionale *else* viene eseguito. Entrambi i cicli *for* annidati e le relative condizioni *if* all'interno di essi sono identici: non appena si registra una brusca diminuzione nella potenza spettrale, il tempo relativo all'evento viene contrassegnato come offset, e il ciclo annidato si interrompe.

3.2 Legato/staccato

Per classificazione delle note in legato e staccato si è scritto il seguente codice:

```
note_type = []
legato = []
staccato = []
EIGTH_NOTE = 60 / (BPM*2) #seconds
sigma = 0.03 #seconds
MAX_NOTE_LENGTH = EIGTH_NOTE + sigma
SAMPLES_BELOW_THRESHOLD = 3

for i in range(0, onsets.size):
    if i == onsets.size-1:
        current_note = mag[onset_frame[-1]:]
        below_threshold = np.count_nonzero(current_note <
    THRESHOLD)
        if (note_length[i] < MAX_NOTE_LENGTH
```

```
14          and below_threshold > SAMPLES_BELOW_THRESHOLD):
15              staccato.append(i)
16              note_type.append('staccato')
17          else:
18              legato.append(i)
19              note_type.append('legato')
20      else:
21          current_note = mag[onset_frame[i]:onset_frame[i+1]]
22          below_threshold = np.count_nonzero(current_note <
    THRESHOLD)
23          if (note_length[i] < MAX_NOTE_LENGTH
24          and below_threshold > SAMPLES_BELOW_THRESHOLD):
25              staccato.append(i)
26              note_type.append('staccato')
27          else:
28              legato.append(i)
29              note_type.append('legato')
```

La variabile $EIGHTH\_NOTE$ rappresenta la durata di un ottavo di nota o semicroma. Questa scelta è preferibile rispetto a utilizzare una costante poiché tiene in considerazione il tempo di registrazione della traccia da analizzare. Inoltre, il parametro σ è stato fissato a 30 ms, consentendo una certa flessibilità nella durata richiesta affinché le note siano considerate suonate in staccato. Le variabili impiegate ancora una volta sono frutto di test effettuati sulle tracce di chitarra. È possibile modificare tale parametro se si analizzano registrazioni di altri strumenti.

Il primo blocco condizionale gestisce il caso limite dell'ultimo onset, in cui $current\_note$ contiene potenze spettrali comprese tra l'ultimo onset e la fine del segnale audio. Questa situazione è diversa dagli altri casi in cui $current\_note$ è formata da due onset consecutivi. La variabile $below\_threshold$ conta quanti valori di $current\_note$ sono inferiori alla soglia $THRESHOLD$, che è stata calcolata in precedenza. Le condizioni nei blocchi condizionali interni verificano che la durata della nota, $note\_length[i]$, sia inferiore a MAX_NOTE_LENGTH. Inoltre, si verifica la presenza di una pausa di almeno 8.7 ms, rappresentata dalla variabile $SAMPLES\_BELOW\_THRESHOLD$, affinché la nota possa essere considerata suonata in staccato. Nel caso in cui queste due condizioni non siano soddisfatte, la nota viene classificata come suonata in legato.

3.3 Pitch detection

La parte di codice per il rilevamento del tono di ogni nota è la seguente:

```
f0, _, _ = librosa.pyin(y=librosa.effects.preemphasis(
    filtered_signal), fmin=librosa.note_to_hz('C2'), fmax=librosa
    .note_to_hz('C7'), sr=sr, frame_length=2048, hop_length=256)

f0_frame = duration/len(f0)
frame_conversion = frame_time/f0_frame
note_frequency = []
new_onsets = []

for i in range(0, onset_frame.size):
    ons_frame = int(onset_frame[i]*frame_conversion)
    onset_frequency = f0[ons_frame]
    j = 1
    while np.isnan(onset_frequency):
        onset_frequency = f0[ons_frame+j]
        j = j+1
    freq_window = []
    k = 1
    while not np.isnan(onset_frequency):
        freq_window.append(onset_frequency)
        onset_frequency = f0[ons_frame+j+k]
        k = k+1

    freq_window = np.int32(freq_window)
    freq_mode = statistics.mode(freq_window)
    note_frequency.append(freq_mode)

note_frequency = np.array(note_frequency)
pitch = librosa.hz_to_note(note_frequency, cents=True)
```

La funzione *librosa.effects.preemphasis* enfatizza le frequenze più alte del segnale audio e attenua quelle più basse. Fornendo come parametro il segnale filtrato, l'algoritmo funziona meglio rispetto al caso in cui non ci sia un filtro di pre-enfasi e venga dato come parametro il segnale audio non filtrato. Nella funzione *librosa.pyin*, il parametro sr corrisponde alla frequenza di campionamento della traccia audio, mentre $fmin$ e $fmax$ impongono rispettivamente un limite inferiore e superiore sulle frequenze fondamentali estratte.

Dato che i parametri utilizzati per il calcolo delle frequenze fondamentali $f0$ sono differenti da quelli impiegati per il calcolo delle potenze spettrali, in particolare per quanto riguarda la lunghezza della finestra audio ($frame\_length$) e il passo temporale (hop_length), è stata introdotta una variabile denominata $frame\_conversion$. Questa variabile consente di convertire gli indici degli onset nei corrispondenti indici delle frequenze fondamentali che fanno riferimento allo

stesso istante temporale all'interno della traccia audio. L'utilizzo di valori diversi per questi due parametri è stato motivato da test che hanno dimostrato una maggiore precisione dell'algoritmo con i nuovi valori. Per il resto, l'algoritmo opera in maniera coerente con quanto descritto nella Sezione 2.4.

In generale, questa parte di codice può essere adattata in sviluppi futuri, non solo per il riconoscimento dell'altezza tonale di registrazioni vocali, ma anche per il rilevamento degli attacchi delle note. Ad esempio, è possibile fissare come onset gli istanti temporali che presentano sbalzi significativi nel contenuto in frequenza della voce.

3.4 Note dynamic

Il codice seguente è utilizzato per rilevare il parametro relativo alla dinamica delle note:

```
dynamics = np.zeros(onsets.size)
abs_sig = np.abs(signal)
FRAME = 32
HOP = FRAME//4
n = 0 # offset index
n_of_peaks = 3 #number of peaks to average

for i in range(0, onsets.size):
    if i == onsets.size-1:
        current_note = signal[int(onsets[-1]*sr): int(offsets[-1]*sr)]
        sorted_current_note = sorted(np.abs(current_note), reverse=True)
        current_peak = np.mean(sorted_current_note[:n_of_peaks])
        rms = np.mean(librosa.feature.rms(y=current_note, frame_length=FRAME, hop_length=HOP))
        dynamics[i] = current_peak/rms
    else:
        if offset_frame[n] < onset_frame[i+1]:
            current_note = signal[int(onsets[i]*sr): int(offsets[n]*sr)]
            n = n+1
            sorted_current_note = sorted(np.abs(current_note), reverse=True)
            current_peak = np.mean(sorted_current_note[:n_of_peaks])
            rms = np.mean(librosa.feature.rms(y=current_note, frame_length=FRAME, hop_length=HOP))
```

```
            dynamics[i] = current_peak/rms
        else:
            current_note = signal[int(onsets[i]*sr): int(onsets[
    i+1]*sr)]
            sorted_current_note = sorted(np.abs(current_note),
    reverse=True)
            current_peak = np.mean(sorted_current_note[:
    n_of_peaks])
            rms = np.mean(librosa.feature.rms(y=current_note,
    frame_length=FRAME, hop_length=HOP))
            dynamics[i] = current_peak/rms
```

I parametri $FRAME$ e HOP usati in *librosa.feature.rms* per calcolare il valore RMS medio, corrispondono a tempi di 1.45 ms e 0.36 ms rispettivamente. Questi due parametri possono essere adattati in base al tipo di strumento musicale che viene analizzato. Alcuni strumenti possono produrre note molto brevi, e in questi casi, è opportuno ridurre tali parametri. In questo modo, saranno calcolati più valori RMS per ogni nota, ottenendo una rappresentazione più dettagliata.

Un discorso analogo si applica a n_of_peaks, che rappresenta il numero di picchi da mediare per il calcolo di $current\_peak$. Quando si analizzano registrazioni di strumenti musicali che producono note con un attacco repentino, decadimento rapido e, in generale, un inviluppo molto breve, è necessario adattare questa costante. In tal caso, è opportuno considerare solo i picchi reali delle note e escludere la possibilità di prendere valori che corrispondono alla porzione di inviluppo relativa al decadimento o al sostegno delle note.

Il calcolo del Peak to Loudness Ratio si è esattamente svolto come riportato nella Sezione 2.5. Il codice per calcolare il Peak to Short-Term-loudness Ratio è riportato qui di seguito:

```
WIN = 3 * sampling_freq
JUMP = WIN // 16
PSR =  []

for i in range(WIN//2, normalized_audio_data.size, JUMP):
    current_true_peak = np.abs(normalized_audio_data[i])
    current_loudness = (np.mean(librosa.feature.rms(y=librosa.
    effects.preemphasis(normalized_audio_data[i-WIN//2: i+WIN
    //2+1]))))
    PSR.append(current_true_peak - current_loudness)
```

Il passo temporale $JUMP$ è una frazione della finestra temporale WIN, la quale ha una durata di 3 secondi. Il valore di $JUMP$ è stato selezionato in modo euristico. Ridurre questo valore comporterebbe un maggiore dettaglio nel PSR, ma

aumenterebbe il tempo di esecuzione del codice, in relazione pressoché inversamente proporzionale.

3.5 Database

Per la creazione del database si è usato il seguente blocco di codice:

```
data = {
    "Onset(s)" : np.round(onsets, 3),
    "Duration(s)" : np.round(note_length, 3),
    "Dynamic" : np.round(dynamics, 3),
    "Type" : note_type,
    "Pitch" : pitch
}

indices = ['Note {}'.format(str(i)) for i in range(0, onsets.
    size)]
df = pd.DataFrame(data, index = indices)
```

	Onset (s)	Duration (s)	Dynamic	Type	Pitch
Note 0	5.972	0.225	4.631	staccato	C4+9
Note 1	6.700	0.277	4.757	staccato	A3-8
Note 2	7.498	0.218	3.668	legato	G3-9
Note 3	7.716	0.170	3.706	staccato	A3-8
Note 4	8.056	0.530	3.913	legato	A3-8
Note 5	9.240	0.151	3.339	legato	G3-9
Note 6	9.391	0.095	3.416	staccato	A3-8
Note 7	9.727	0.176	4.433	staccato	C4-4
Note 8	10.105	0.167	3.404	staccato	D4-4
Note 9	10.470	0.110	2.659	legato	D♯4-1
Note 10	10.581	0.147	2.647	staccato	D♯4-1
Note 11	10.857	0.196	3.285	staccato	D4-4
Note 12	11.237	0.216	5.069	staccato	C4-11
Note 13	11.623	0.193	3.931	staccato	A3-8

Figura 3.1: Estratto del database

Nel database sono stati registrati i valori di tempo di attacco, durata, dinamica, tipo di esecuzione e altezza tonale di ciascuna nota. Questo database agevola

la visualizzazione dei dati e permette il calcolo delle correlazioni tra le diverse caratteristiche musicali.

3.6 Vocal Recorder

Nell'ultimo segmento di codice, è stata sviluppata la parte relativa alla registrazione audio, che permetterà in futuro di catturare l'input vocale per estrarre i parametri utilizzati nell'espressività dei file MIDI. Di seguito, viene presentata un'illustrazione della struttura di questo codice, poiché la sezione completa risulterebbe troppo estesa:

```
class MetronomeRecorder:
    def __init__(self, root):
        ...

    def start_recording(self):
        ...

    def stop_recording(self):
        ...

    def record_audio(self):
        ...

    def update_timer(self):
        ...

    def play_metronome(self):
        ...

    def save_audio(self):
        ...

if __name__ == "__main__":
    root = tk.Tk()
    app = MetronomeRecorder(root)
    root.mainloop()
```

Inizialmente viene definita la classe $MetronomeRecorder$, imputata alla registrazione audio dal dispositivo sul quale viene eseguito il codice. Di seguito, viene fornita una breve descrizione delle funzioni svolte dai vari blocchi di codice. Il primo metodo, *__init__(self, root)*, ha lo scopo di inizializzare gli attributi dell'istanza creata durante l'esecuzione del codice, denominata app. Questo metodo è utilizzato per creare una finestra GUI (Graphical User Interface), che consente all'utente

di interagire con l'applicazione. Il metodo *start_recording(self)* viene chiamato quando il pulsante "*Start*" è premuto. Imposta il tempo dei battiti al valore inserito, avvia la registrazione audio, il metronomo e gestisce l'aggiornamento del timer. Il metodo *stop_recording(self)* è chiamato quando il pulsante "*Stop*" viene premuto. Interrompe la registrazione audio e il metronomo, salva l'audio registrato in un file e reimposta il tempo del timer a "00.00". Il metodo *record_audio(self)* avvia la registrazione audio catturando i dati audio dall'input. Il metodo *update_timer(self)* aggiorna il timer durante la registrazione, visualizzando il tempo trascorso. Il metodo *play_metronome(self)* riproduce il suono del metronomo in base al tempo dato in input. L'ultimo metodo *save_audio(self)* salva l'audio registrato in un file WAV. L'ultima parte del codice permette di creare lanciare l'interfaccia grafica per la registrazione audio, creando l'oggetto app, non appena il codice viene eseguito.

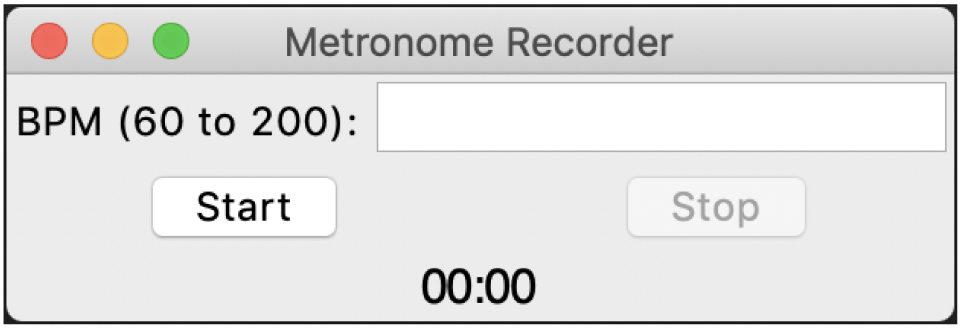

Figura 3.2: Interfaccia per la registrazione audio

Conclusioni

L'estrazione di parametri derivanti da caratteristiche musicali rappresenta una tappa essenziale nel campo del processamento di segnali audio. Oggi, esistono numerose applicazioni software specializzate nell'analisi delle caratteristiche musicali.
In questo lavoro, è stato introdotto un metodo per l'estrazione di parametri da alcune delle principali caratteristiche musicali. L'algoritmo è stato implementato in Python e testato su estratti di chitarra elettrica, con risultati molto positivi.
Nel contesto del progetto finale, di cui questa tesi rappresenta solo una parte, si intendono estrarre parametri da registrazioni vocali al fine di guidare un modello computazionale nella generazione di espressività musicale applicata a composizioni MIDI inespressive. Tuttavia, va notato che l'algoritmo potrebbe non garantire la stessa precisione in segnali audio più complessi, come la voce umana.
Il passo successivo prevede la revisione di alcune parti del codice, ad esempio il blocco di codice dedicato al rilevamento dell'altezza tonale, per ottimizzare l'algoritmo e renderlo adeguato all'analisi di segnali audio vocali.

Bibliografia

[1] Sebastian Böck and Gerhard Widmer. Maximum filter vibrato suppression for onset detection. 2013.

[2] Pierluigi Bontempi, Sergio Canazza, Filippo Carnovalini, and Antonio Rodà. Research in computational expressive music performance and popular music production: A potential field of application? *Multimodal Technologies and Interaction*, 7(2), 2023.

[3] Nathan R. Carr, Kirk N. Olsen, and William Forde Thompson. The Perceptual and Emotional Consequences of Articulation in Music. *Music Perception*, 40(3):202–219, 02 2023.

[4] Corey Cheng. Design of a pitch quantization and pitch correction system for real-time music effects signal processing. In *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*, pages 1–6, 2012.

[5] Alicia Fernández-Sotos, Antonio Fernández-Caballero, and José M Latorre. Influence of tempo and rhythmic unit in musical emotion regulation. *Front. Comput. Neurosci.*, 10:80, August 2016.

[6] Stuart Kamenetsky, David Hill, and Sandra Trehub. Effect of tempo and dynamics on the perception of emotion in music. *Psychology of Music - PSYCHOL MUSIC*, 25:149–160, 10 1997.

[7] Meng Koon Lee, Mohammad Hosseini Fouladi, and Satesh Narayana Namasivayam. Mathematical modelling and acoustical analysis of classical guitars and their soundboards. *Advances in Acoustics and Vibration*, 2016:6084230, Dec 2016.

[8] Matthias Mauch and Simon Dixon. Pyin: A fundamental frequency estimator using probabilistic threshold distributions. In *2014 IEEE International Confe-*

rence on Acoustics, Speech and Signal Processing (ICASSP), pages 659–663, 2014.