UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

VALIDAZIONE DEGLI ALGORITMI DI DESCRIZIONE TONALE DELLE API DI FREESOUND

Elaborato finale di:
Roberto Caselli
Matricola 728156

Relatore: Prof. Luca Andrea Ludovico

Correlatore: Dott. Adriano Baratè

Anno Accademico 2010 – 2011

INDICE

Introduzione

L'informatica applicata alla musica ha ultimamente suscitato l'interesse degli sviluppatori di software e dei musicisti. In particolare possono essere importanti funzioni e programmi che interagiscano con i file audio contenenti registrazioni, come ad esempio calcolare la tonalità di un brano solamente a partire dallo spettrogramma del file, o anche calcolare il ritmo del pezzo, o gli accordi presenti. Il sito Freesound.org contiene questi tipi di funzioni. Si tratta di algoritmi sperimentali e freeware: è possibile utilizzarle, ma non è possibile studiarle o modificarne il codice sorgente. Lo scopo di questo elaborato è quello di verificare il corretto funzionamento di una parte di questi algoritmi, solo quelli di descrizione tonale, e di creare un software che ne automatizzi il funzionamento.

L'elaborato è composto da tre capitoli. Il primo descriverà il sito Freesound.org, illustrandone il contenuto e descrivendo dettagliatamente l'utilità e lo scopo degli algoritmi in esso presenti. Il secondo capitolo sarà dedicato interamente al software. Il terzo, infine, conterrà i risultati dei test degli algoritmi sui file audio.

1. Freesound

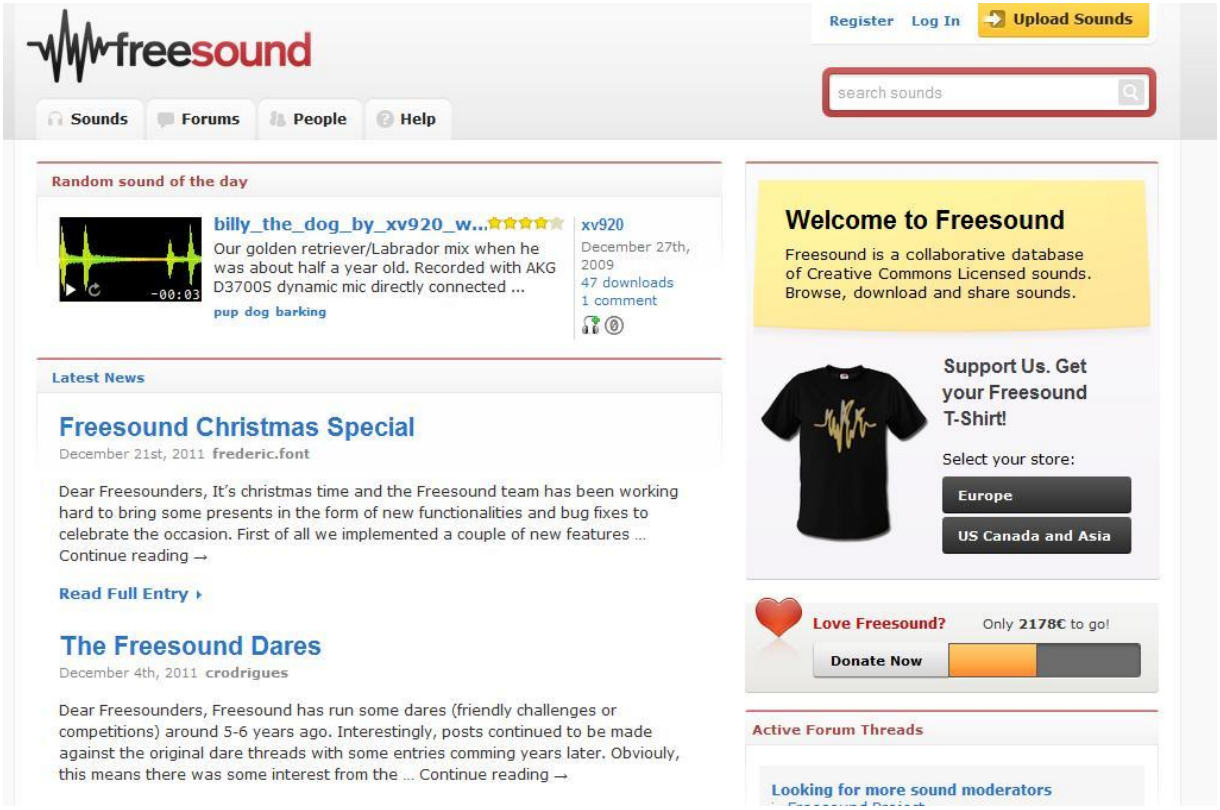
Freesound.org è un sito contenente un grandissimo numero di file audio, inseriti dagli utenti, di qualsiasi tipologia. Lo scopo del sito è infatti quello di creare un grande database di frammenti audio, campioni e registrazioni.

Il sito si divide sostanzialmente in due parti:

- Il database contenente i file audio;
- La parte di sviluppo, contenente le API, librerie che restituiscono dai file audio vari parametri.

Inoltre il sito contiene anche un Forum, una sezione in cui gli utenti possono discutere sui progetti di Freesound, o sui file audio caricati, o riportare errori e bugs del sito, oppure chiedere aiuto sul funzionamento delle API o del caricamento dei file. Per accedere al Forum, per caricare e scaricare i file audio e per usare le API è necessario effettuare la registrazione.

La home page si presenta in questo modo:



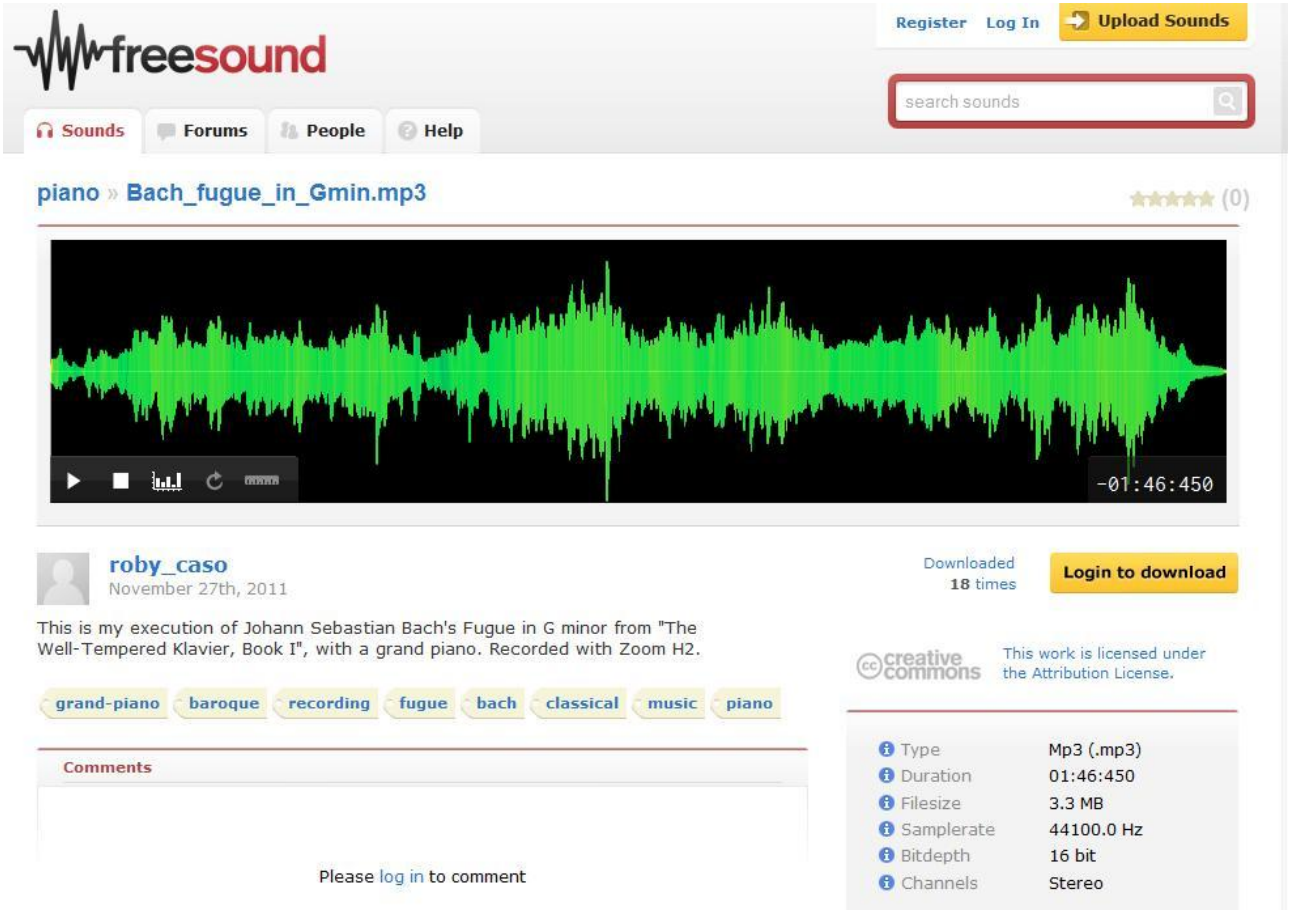
In alto a sinistra i quattro pulsanti servono rispettivamente per ricercare i file audio sul database, accedere al Forum, ricercare gli utenti iscritti al sito e le loro caratteristiche, e visualizzare la sezione "Help" per l'aiuto. Grazie alla presenza di un motore di ricerca (in alto a destra) è possibile effettuare ricerche dei file audio inserendo parole chiave. Sulla home page viene visualizzato uno

dei suoni del database scelto casualmente (Random sound of the day), e sotto ci sono le ultime novità.

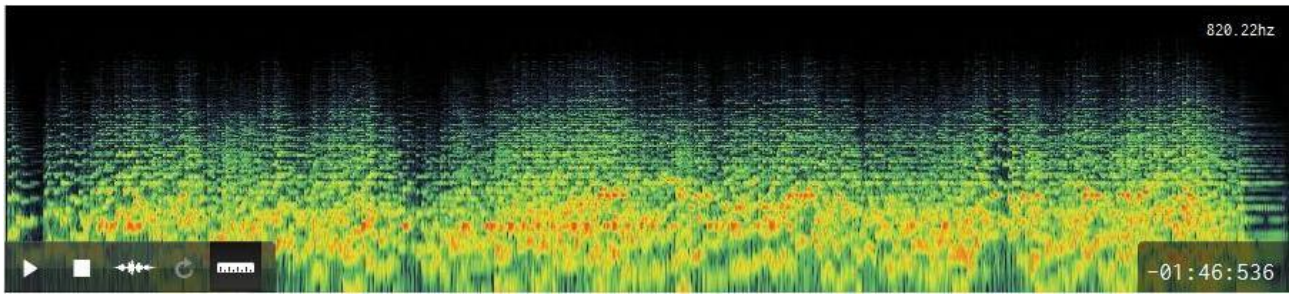
1.1 I file audio di Freesound

Il sito contiene svariati tipi di file audio, ad esempio rumori, suoni sintetizzati, voci umane, musiche, loop di accordi, suoni mixati, ecc. I formati supportati sono wav, mp3, ogg, aif e flac. Ogni file audio è identificato da un ID numerico (un numero intero). I file vengono caricati sul database del sito dagli utenti registrati.

I file audio si presentano in questo modo:



In alto è presente il titolo del brano con la cartella alla quale il file è associato. Al centro è visualizzata la forma d'onda del file audio. Il file si può mandare in play e in stop, grazie ai pulsanti posti sul grafico della forma d'onda. Il pulsante al centro con le barre serve per cambiare la visualizzazione da sonogramma a spettrogramma, e viceversa, quello con la freccia serve a mandare in loop il brano. L'ultimo pulsante serve per visualizzare l'ampiezza in decibel dell'onda nel punto in cui è posto il mouse (se si sta visualizzando il sonogramma), oppure serve per visualizzare la frequenza in Hz sempre nel punto in cui è posto il cursore (se si sta visualizzando lo spettrogramma).



Spettrogramma dello stesso file

Sotto il grafico sono presenti il nome dell'utente che ha inserito il file, la data di inserimento, il numero di download (a destra), la descrizione (scritta dall'utente al momento del caricamento del file) e i tag, inseriti dall'utente per facilitare la ricerca del file secondo parole chiave. Sotto è possibile scrivere i commenti. In basso a destra sono presenti alcune descrizioni del file: estensione (mp3 in questo caso), durata, grandezza del file in Mbyte, frequenza di campionamento, bit di quantizzazione e numero di canali.

1.2 Le API

Le API (acronimo di Application Programming Interface) di Freesound sono delle librerie che calcolano dal file audio vari parametri, come metadati, bitrate, frequenza di campionamento, ma anche ritmo, accordi presenti, tonalità, modo (maggiore o minore) e tipo di accordatura. Con le API si possono fare ricerche sulle informazioni relative agli utenti, sulle cartelle contenenti i suoni e sui suoni stessi. Si possono anche cercare suoni simili a uno dato, la ricerca viene fatta analizzando il contenuto del file. Le API di Freesound non possono essere modificate, sono di sola lettura. La maggior parte di queste librerie sono sperimentali: lo scopo di questo elaborato è appunto quello di verificare il corretto funzionamento di esse, con tutti i tipi di file audio.

Il funzionamento di questi algoritmi avviene tramite chiamate http standard. Per eseguire correttamente le chiamate è necessario essere registrati e farsi dare una API key, una chiave contenente un certo numero di valori alfanumerici, da inserire alla fine dell'URI dell'API che si vuole utilizzare. Le risposte delle API consistono in uno status code http, indicante il successo o meno della richiesta. Le richieste http sono di questo genere:

http://www.freesound.org/api/sounds/<sound_id>/analysis/<tipo_funzione>/<nome_funzione>/?api_key=<api_key>

Le parti di URI tra <> sono quelle variabili, sound_id deve contenere l'ID del file audio di cui si vuole calcolare la funzione, tipo_funzione è la tipologia di API che si vuole usare (come descrittori di metadati, o di ritmo, o tonali, ecc.), nome_funzione è il nome dell'API, e api_key è quella che viene consegnata al momento della registrazione al sito. [5]

Le funzioni sono divise in 6 gruppi:

- Descrittori di metadati
- Descrittori di alto livello (highlevel)
- Descrittori di basso livello (lowlevel)
- Descrittori del ritmo
- Descrittori sfx
- Descrittori tonali

I primi 5 gruppi di funzioni saranno descritti brevemente, mentre ci si soffermerà più a lungo sui descrittori tonali.

1.2.1 Metadata descriptors

In generale i metadati sono i dati relativi al contenuto musicale del brano, come titolo, autore, anno di pubblicazione, ecc. In Freesound ci sono 2 descrittori di metadati: audio_properties e version.

Audio_properties

Calcola dal file audio la frequenza di campionamento, il bitrate (numero di bit al secondo), il numero di canali (1 se mono, 2 se stereo), la lunghezza del brano e altri parametri.

Version

Calcola la versione dell'analizzatore.

1.2.2 Highlevel descriptors

Sono i descrittori di alto livello, cioè calcolano dal file audio parametri indipendenti dall'analisi spettrale del file, quindi informazioni di tipo musicale e musicologica, più soggettive.

Acustic

La funzione acustic dice se l'origine del file è acustica oppure no (ad esempio elettronica), e calcola la percentuale di quanto il suono sia o meno acustico (è un numero tra 0 e 1: più è vicino a 1 e più c'è la probabilità che il suono sia acustico).

Ballroom

Questa funzione dà una classificazione ritmica del brano da analizzare: restituisce il tipo di ritmo (ad esempio valzer, samba, tango, ecc.) e la percentuale (più si avvicina a 1 e più il ritmo del brano è vicino a quello calcolato).

Culture

Questa API indica se il brano ha origine occidentale o no, e indica la probabilità di quanto il risultato restituito sia vero.

Electronic

Questa funzione è simile ad acustic, con la differenza che indica se il brano è elettronico o no, e ne calcola la percentuale.

Gender

Indica se la voce che canta all'interno del brano (nel caso in cui il file audio contenga una canzone) sia di un uomo o di una donna, e anche qui ne calcola la percentuale.

Genre

Calcola il genere del brano musicale da analizzare. Questa API dovrebbe essere in grado di riconoscere molti tipi di genere musicale, come classico, jazz, blues, rock, pop, techno, house, metal, reggae, ecc. Anche in questo caso viene anche calcolata la percentuale che il genere sia veramente quello trovato.

Live_studio

La funzione live_studio calcola se l'audio proviene da una registrazione live o in studio.

Mood

Calcola dal brano musicale vari parametri soggettivi e "umani", ad esempio se il brano è rilassante o meno, se è aggressivo o meno, se è triste o meno, se è melanconico, appassionato, ecc. Per ogni categoria di sensazioni viene calcolata la percentuale.

Rhythm

Dice se la velocità del brano da analizzare è veloce, media o lenta.

Timbre

Dice se il timbro della musica è brillante o scuro.

Voice_instrumental

Dice se si sta analizzando un brano vocale o strumentale. Anche in tutte queste funzioni viene calcolata la percentuale. [5]

1.2.3 Lowlevel descriptors

I descrittori di basso livello dipendono dall'analisi spettrale del file audio. Lo spettro del segnale viene calcolato con una FFT (Fast Fourier Transform) a partire dalla forma d'onda del segnale stesso.

Average_loudness

Questo algoritmo calcola la media dell'energia del segnale.

Barkbands

Queste 4 funzioni, invece, sostanzialmente dividono lo spettro del segnale in 28 bande e per ciascuna ne calcolano la potenza.

Dissonance

Dissonance, analizzando le frequenze del segnale, calcola la percentuale di dissonanza. Si basa sul fatto che due sinusoidi di frequenze diverse sommate tra loro generano un suono che, a seconda della differenza di frequenza e di ampiezza delle sinusoidi, può essere più o meno dissonante. La dissonanza totale è derivata dalla somma di tutte le componenti frequenziali di un certo frame del segnale. In generale, un suono dato dalla sovrapposizione di più frequenze (detto intervallo se i suoni sovrapposti sono 2) è tanto più dissonante quanto più sono presenti battimenti tra gli armonici di ordine inferiore dei suoni. Invece un intervallo è consonante quando coincidono gli armonici di ordine inferiore dei suoni che lo compongono, o quando l'intervallo può essere espresso con rapporti numerici semplici (3/2, 4/3, ecc). [6]

HFC

L'HFC (High Frequency Content) dà una misura del contenuto in alta frequenza del segnale.

Pitch

Queste funzioni calcolano la frequenza fondamentale (detta, appunto, pitch) di suoni monofonici (formati, cioè, da un solo suono). La funzione silence_rate_60db restituisce un numero binario: 1 se la potenza del segnale nel frame di input è inferiore a -60 dbFS, 0 se è superiore.

Spectral

Le API spectral calcolano lo spettro del segnale del file audio da analizzare, e restituiscono vari parametri a seconda della funzione utilizzata: ad esempio l'energia dello spettro in un certo frame, l'energia in una certa banda frequenziale, l'rms, ecc.

Zerocrossingrate

Dà un'indicazione del livello di rumorosità del segnale: restituisce un valore compreso tra 0 e 1, più si avvicina a 1 e più il segnale è rumoroso. [5]

1.2.4 Rhythm descriptors

I descrittori ritmici calcolano dal file audio dei parametri relativi al ritmo, inteso come successione costante di movimenti (pulsazione).

Loudness

Dà una misura di quanto il ritmo del pezzo sia forte, dando un'indicazione del carattere del brano: restituisce un numero compreso tra 0 e 1, più è vicino a 1 e più il pezzo ha un ritmo forte e

travolgente. L'API position restituisce la posizione del beat (pulsazione) all'interno della traccia, questo serve per determinare il tempo della traccia.

BPM (Beat Per Minute)

E' la misura del tempo del brano, e indica le pulsazioni per minuto. Va da un minimo di 40 ad un massimo di 208.

Estimates

Restituisce la lista dei BPM stimati presenti nel brano, in quanto in un brano il tempo può cambiare.

Peak

Questi algoritmi calcolano il più alto valore di BPM trovato nel brano.

Onset

Queste API calcolano il numero di note o suoni in un secondo, anche queste danno un'indicazione di tempo, in generale più un brano è "denso" di note e più avrà un ritmo alto.

Rubato_start e rubato_stop

Indicano, rispettivamente, quando cominciano e quando finiscono i rallentandi all'interno del brano (se ci sono più rallentandi, le funzioni restituiranno più valori).

Second_peak

Similmente alle funzioni peak, le funzioni second_peak calcolano il secondo più alto valore di BPM trovato nel brano. [5]

1.2.5 Sfx descriptors

Inharmonicity

Calcola il grado di inarmonicità del brano musicale, più il valore si avvicina a 1 e più l'inarmonicità è grande. Questa è una caratteristica tipica del pianoforte, in quanto le armoniche delle sue frequenze non sono perfettamente esatte, si parla infatti di inarmonicità degli ipertoni. [2]

Pitch_after_max_to_before_max_energy_ratio

Questo algoritmo indica in percentuale il valore dell'energia dopo la massima energia prima del massimo valore di pitch. Suoni con pitch ascendenti avranno un valore basso (vicino a 0), mentre suoni con pitch discendente avranno un valore alto (vicino a 1).

Pitch

Queste funzioni calcolano diversi parametri in percentuale relativi alla frequenza fondamentale del suono.

Tristimulus

Questa libreria dà un'indicazione del timbro di un suono a seconda dell'energia associata ad ogni armonica. Restituisce tre valori: il primo corrisponde all'energia relativa (in percentuale) associata alla frequenza fondamentale, il secondo a quella associata alla seconda, terza e quarta armonica, e il terzo all'energia associata ai rimanenti armonici. [5]

1.2.6 Tonal descriptors

I descrittori tonali sono quel gruppo di funzioni che calcolano dal brano informazioni di tipo più "musicale", come accordi, tonalità, ecc. Si dividono a loro volta in 3 gruppi:

- Chords, che calcolano informazioni sul tipo di accordi presenti nel file audio;
- Key, i quali calcolano dal file la tonalità e il modo;
- Tuning, che restituiscono informazioni sul tipo di accordatura usata dallo strumento che sta suonando nel brano musicale analizzato.

Queste funzioni sono chiaramente pensate per file audio contenenti brani musicali. Sarebbe anche interessante scoprire come queste API si comportino in presenza di file audio contenenti suoni non musicali, come rumori, voci umane o suoni sintetizzati.

Chords_changes_rate e gli accordi

L'armonia e il carattere di un brano musicale dipendono dalla presenza, dall'alternanza e dalla varietà degli accordi in esso presenti.

Nella musica occidentale è essenziale il rapporto tra tonica (1° grado della scala) e dominante (quinto grado), perché è proprio questo rapporto a caratterizzare la tensione e la distensione di un brano musicale; in particolare la tonica è lo stato di massimo riposo, mentre la dominante è lo stato di massima tensione, che la porta a risolvere naturalmente sulla tonica. Ovviamente non esistono solo accordi di tonica e di dominante, ma esistono anche accordi di tensione "intermedia", come l'accordo di sottodominante.

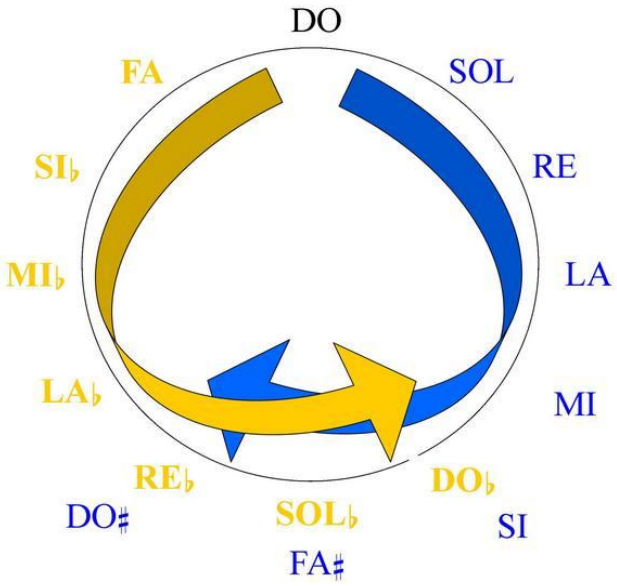
Per arricchire ulteriormente l'armonia di un pezzo è possibile aggiungere anche altri accordi, estranei alla tonalità di impianto, come le dominanti secondarie (che risolvono su un certo grado della scala). Oppure esistono i cosiddetti "giri armonici": eseguire un loop di accordi, prima di tornare alla tonalità di impianto, o anche per cambiare tonalità in modo più ricco e meno improvviso di una modulazione immediata, armonicamente più povera. Alcuni "giri" di accordi sono chiamati "progressioni": si tratta di ripetere la stessa successione di accordi in modo simmetrico, in direzione ascendente o discendente. [4]

La funzione chords_changes_rate permette di calcolare la percentuale dei cambiamenti degli accordi presenti nel brano. Restituisce un numero compreso tra 0 e 1: più è vicino a 1 e più ci sono cambiamenti accordali, questo significa che l'armonia varia maggiormente ed è quindi più ricca.

Chords_histogram e il circolo delle quinte

La funzione chords_histogram rappresenta, per ogni possibile accordo, la percentuale con la quale quell'accordo compare nel brano musicale. Restituisce una lista di 24 valori compresi tra 0 e 100, che rappresentano la percentuale della presenza degli accordi nel seguente ordine: C, Em, G, Bm, D, F#m, A, C#m, E, G#m, B, D#m, F#, A#m, C#, Fm, G#, Cm, D#, Gm, A#, Dm, F, Am (sigle anglosassoni). Questo ordine rispetta il circolo delle quinte. L'algoritmo non restituisce note con i bemolli, quindi gli accordi con i bemolli verranno restituiti nei loro enarmonici accordi con diesis (ad esempio se c'è un LAb verrà restituito SOL#).

Il circolo delle quinte è un grafico utilizzato in teoria musicale per mostrare le relazioni esistenti tra i dodici suoni che compongono la nostra scala musicale. Partendo dal DO e salendo di quinta giusta in quinta giusta, dopo 12 quinte si arriverà al SI#, che enarmonicamente coincide con il DO: in questo modo si chiude il circolo. Ad ogni quinta ascendente si aggiunge un diesis alla tonalità. Si può anche fare il contrario, scendendo di quinta giusta in quinta giusta. Ad ogni quinta discendente si aggiunge un bemolle alla tonalità. Per comodità non si usano mai più di 7 alterazioni in una scala, quindi da DO# maggiore (7 diesis), che coincide con REb maggiore (5 bemolli), salendo di una quinta si passa direttamente a LAb maggiore (SOL# maggiore avrebbe 8 diesis in chiave, quindi ci sarà FA doppio diesis).



Circolo delle quinte delle tonalità maggiori

E’ importante precisare che il circolo delle quinte funziona solo con il sistema temperato, in cui tutti i semitoni hanno la stessa distanza tra di loro. Nella scala pitagorica, ad esempio, l’intervallo di quinta è definito come 3/2: in questo caso dopo 12 raddoppi di quinta non si arriverà all’ottava, infatti $(3/2)^{12} > 2^8$, quindi SI# non coincide con DO. [6]

Chords_key

Questo algoritmo restituisce la sigla dell’accordo nel segmento di brano analizzato. Viene analizzato un frammento audio di 1 o 2 secondi, e da questo viene estratto l’accordo presente. La funzione non riconosce il modo dell’accordo, cioè non sa se l’accordo trovato sia maggiore o minore, restituisce solo il nome dell’accordo.

Chords_number_rate

Questa funzione permette di trovare la percentuale del numero di accordi presenti che vengono eseguiti per più dell’1% all’interno del brano. La percentuale è espressa da un numero reale compreso tra 0 e 1.

Chords_progression

Questa API calcola la progressione di accordi presenti nel brano analizzato, restituendo una stringa che rappresenta la sequenza di accordi della canzone. Distingue anche gli accordi maggiori da quelli minori.

Chords_scale

Questa funzione restituisce una stringa che rappresenta il modo dell’accordo del brano. Sono considerati solo accordi di triade maggiori e minori, l’algoritmo non è quindi in grado di riconoscere accordi eccedenti, diminuiti o di settima.

Chords_strength

Questo descrittore dà un’indicazione su quanto l’accordo effettivamente presente all’interno del brano sia vicino a quello calcolato. Restituisce un valore compreso tra 0 e 1: un risultato alto significa che gli accordi trovati sono molto tonali, mentre un valore basso indica che gli accordi trovati non sono molto tonali.

Accordi poco tonali può significare che il temperamento utilizzato non sia quello temperato, che si usa tipicamente nella musica occidentale. Ad esempio le scale pitagorica o quella naturale, usate prima dell’invenzione del temperamento equabile, oppure scale modali o non usuali nella musica

occidentale, portano a risultati bassi, più vicini a 0. Accordi poco tonali possono essere causati anche da uno strumento non accordato perfettamente.

HPCP e simili. Il temperamento equabile

L'Harmonic Pitch Class Profile restituisce un vettore che rappresenta l'intensità di ciascun pacchetto di frequenze della scala temperata. Tutte le API di questo paragrafo dipendono da questa funzione.

L'HPCP Trasposto (THPCP) permette di calcolare la prima posizione corrispondente al livello più alto dell'HPCP. Restituisce un vettori di numeri reali compresi tra 0 e 1.

La funzione tuning_diatonic_strength rappresenta il tonal_key_strength calcolato usando un profilo tonale di 120 pacchetti di frequenze calcolate con l'HPCP.

Tuning_equal_tempered_deviation dà una misura di quanto la scala musicale del brano può essere considerata ben temperata. Restituisce un numero reale non negativo, più il numero è elevato e più la scala del brano si discosta da una scala ben temperata.

Tuning_nontempered_energy_ratio ha la stessa funzione della precedente tuning_equal_tempered_ deviation, con la differenza che restituisce un numero compreso tra 0 e 1, più il numero è vicino a 0 e più la scala del brano si discosta da una scala ben temperata. [5]

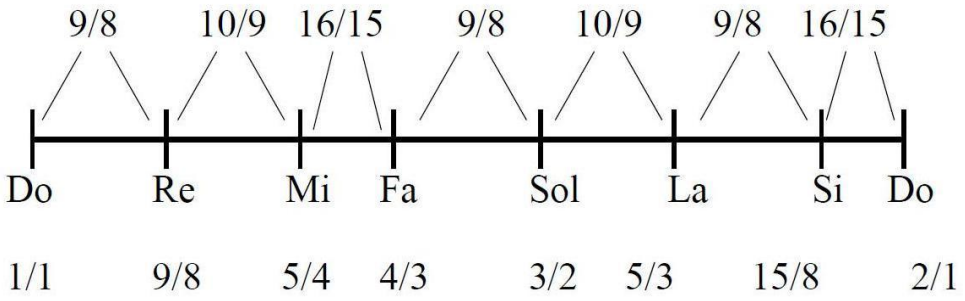
Una scala si dice temperata se i semitoni all'interno di essa hanno la stessa distanza tra di loro. La nostra scala musicale è formata da 12 semitoni, quindi, poiché una nota ha frequenza doppia rispetto a quella dell'ottava precedente, ciascun suono ha un rapporto con il suono a un semitono di distanza da esso di $\sqrt[12]{2}$. Ovviamente possono esistere altri tipi di scala temperata, in cui l'ottava è divisa in un numero diverso di 12 semitoni. La nostra scala è divisa in 12 semitoni perché in questo modo si avvicina di più alla scala naturale.

Il temperamento equabile è stato adottato nel XVII secolo grazie al trattato di Andreas Werckmeister, pubblicato nel 1691. La definitiva attuazione pratica di questo principio teorico fu operata da Johann Sebastian Bach con la pubblicazione del Clavicembalo Ben Temperato, una raccolta di 48 preludi e fughe in tutte le tonalità.

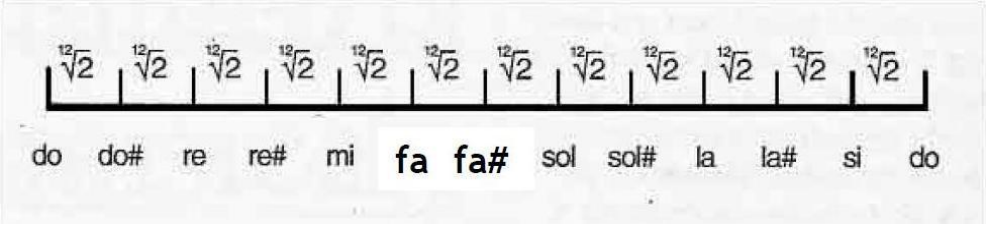
Il sistema temperato fu inventato per risolvere i problemi della scala naturale. Questa è la scala di giusta intonazione, che comprende tutte le consonanze naturali. La caratteristica di essa è che tutte le note che la compongono sono formate a partire dagli armonici naturali del suono di riferimento, riportati nell'ottava in uso. E' la scala perfetta per l'armonia, tuttavia non tutti gli intervalli suonano allo stesso modo; ad esempio la quinta RE-LA (prendendo la scala di DO) è dissonante perché costruita su armonici di ordine molto elevato, mentre DO-SOL è perfettamente consonante. Inoltre non è possibile modulare perché cambiando tonalità cambiano anche i rapporti tra i suoni, causando

dissonanze tra gli intervalli nella nuova tonalità. Poiché per rendere più ricca l'armonia di un brano occorreva modulare, cioè cambiare tonalità all'interno dello stesso brano, l'invenzione del temperamento equabile si rese indispensabile.

La scala temperata però è un'approssimazione della scala naturale, il problema è che non esistono più intervalli consonanti, poiché tutti gli intervalli sono espressi con una potenza della $^{12}\sqrt{2}$, quindi non sono espressi da rapporti numerici semplici. L'approssimazione è comunque minima, ed è difficile sentire la differenza tra scala temperata e naturale. [6]



Intervalli nella scala naturale



Intervalli nella scala temperata

Key_key

Questo algoritmo permette di conoscere la tonalità del brano analizzato (non il modo, che verrà calcolato dalla prossima funzione). Restituisce una stringa che rappresenta il nome della tonica del pezzo in sigla anglosassone.

Qualsiasi brano musicale, escludendo la musica atonale e dodecafonica del '900, pur nella libertà assoluta, durante lo svolgimento tende a gravitare su una particolare nota chiamata tonica (il 1° grado della scala), nota fondamentale su cui il brano stesso si conclude. La tonalità è quindi un concetto essenziale per la musica a partire dal primo Barocco. Si definisce tonalità l'atteggiarsi delle note in rapporto alla tonica, che oltre ad essere il centro di attrazione e di intonazione dà il suo nome alla scala o al brano musicale.

Essendo la scala musicale formata da 12 suoni, come la scala cromatica, ci saranno 24 scale diverse, 12 maggiori e 12 minori. Questo solo in pratica, infatti in teoria le scale sono 30: una senza alterazioni in chiave, 7 con i diesis e 7 con i bemolli (perché 7 sono le note musicali) più le relative

minori. In pratica sono 24 perché 6 di queste sono omofone o omologhe: SI maggiore è omofona di DOb maggiore, FA# di SOLb e DO# di REb, più le relative minori. [3]

Key_scale

Questa funzione, come anticipato prima, permette di calcolare il modo della tonalità del brano (maggiore o minore). Può restituire quindi 2 valori: major oppure minor.
Il modo è dato dalla diversa successione dei gradi della scala, cioè dalla disposizione dei toni e dei semitoni che nella scala minore risulta diversa rispetto alla scala maggiore. Si tratta sempre e comunque di scale diatoniche, cioè di scale formate da 5 toni e da 2 semitoni diatonici non consecutivi. Ogni scala di modo maggiore ha una sua relativa scala di modo minore, la quale ha come tonica il 6° grado della relativa maggiore e conserva le sue stesse alterazioni. [3]

Key_strength

Simile alla funzione chords_strength, key_strength indica quanto la tonalità effettiva si avvicini a quella calcolata dall'algoritmo. Più il valore si avvicina a 1 e più il pezzo risulta tonale. Con musiche atonali o dodecafoniche, o con musiche in cui non c'è un vero e proprio centro tonale, la funzione restituirà un numero basso vicino a 0, anche se gli algoritmi precedenti restituiranno comunque una tonalità, in quanto devono dare per forza una risposta tra quelle attese (cioè una nota della scala).

Tuning_frequency

Per accordare gli strumenti secondo una scala musicale è necessario definire una frequenza di riferimento: il LA centrale. La frequenza del LA centrale è cambiata nel corso degli anni, passando da valori variabili da 370 a 560 Hz, nel '500; 422,5 Hz nel '600, 458 Hz nel 1885 e finalmente 440 Hz nel 1939.
L'API tuning_frequency restituisce proprio la frequenza usata per accordare il brano (440 Hz di default).

1.3 Considerazioni

Dato questo elenco di API, ci si rende conto che esse possono fare quasi qualsiasi cosa sui file audio analizzati, restituendo tutti i parametri possibili; sarebbe fantastico se queste librerie funzionassero sempre e con tutti i tipi di file. Come è già stato detto, però, queste funzioni sono sperimentali, la maggior parte devono ancora essere sviluppate bene, è quindi possibile che alcune di esse non funzionino correttamente, o che restituiscano valori diversi da quelli attesi.

La maggior parte di esse, in particolare i descrittori ritmici e tonali, sono fatte apposta per funzionare con file audio di tipo musicale, contenenti cioè musiche. Ovviamente analizzando file audio contenenti suoni non musicali, come rumori o voci umane, i risultati di queste API non saranno quelli attesi. Le librerie sono progettate per restituire sempre e comunque uno dei loro valori. Se ad esempio il suono da analizzare è un rumore bianco, in cui l'energia è costante a tutte le frequenze, funzioni come key_key o rhythm_bpm o non funzioneranno, oppure restituiranno un valore non compatibile: non ha senso parlare di tonalità o di ritmo se c'è un rumore, poiché all'interno del rumore sono presenti tutte le tonalità (perché sono presenti tutte le frequenze) e non c'è un ritmo.

Le funzioni che verranno analizzate in questo elaborato sono solo i descrittori tonali. Verranno utilizzati vari tipi di file audio, diversi tra loro, per testare il funzionamento delle API. Il primo passo è quello di creare un programma per semplificare e automatizzare il funzionamento delle API tramite chiamate http. Dopodiché verranno analizzati i vari file audio, magari inserendone qualcuno di nuovo sul database del sito per verificare ogni possibile soluzione.

2. Il software

Lo scopo del software è quello di automatizzare le richieste http per far funzionare le API. Le API di descrizione tonale, ricordiamo, sono le 16 seguenti:

- chords_changes_rate
- chords_histogram
- chords_key
- chords_number_rate
- chords_progression
- chords_scale
- chords_strength
- hpcp
- key_key
- key_scale
- key_strength
- thpcp
- tuning_diatonic_strength
- tuning_equal_tempered_deviation
- tuning_frequency
- tuning_nontempered_energy_ratio

Il programma è stato realizzato nel linguaggio di programmazione a oggetti C#, usando Microsoft Visual C# 2010 Express Edition su un sistema operativo Windows Vista, ed è un'applicazione di tipo WPF (acronimo di Windows Presentation Foundation), quindi è presente sia codice in C#, sia codice in XAML (con la stessa sintassi dell'XML), che serve per la parte grafica del software. WPF è una libreria di classi del Framework .NET per lo sviluppo delle interfacce grafiche in ambiente Windows. [7]

Il programma si chiama SW_Tesi. L'idea è stata quella di evitare di dover ogni volta riscrivere l'URL sulla barra del browser, ma di far funzionare le API semplicemente inserendo l'ID del file audio in una casella di testo e cliccando un pulsante per inviare la richiesta.

Il software è composto da una sola finestra grafica, chiamata Window1.xaml, e del rispettivo file in C#, chiamato Window1.xaml.cs.

2.1 XAML

La parte grafica (XAML) si presenta in questo modo:

I controlli presenti sono 2 Label, 16 radioButton, una textBox, una listBox, 3 pulsanti e un'immagine. La sintassi dello XAML è esattamente quella dell'XML. E' presente un elemento radice, chiamato Window (che rappresenta l'intera finestra), che ha come figlio l'elemento Grid, cioè la griglia all'interno della quale sono presenti i controlli. Ancora più a basso livello, dentro la Grid, sono presenti come elementi i controlli sopra descritti, all'interno dei quali ci sono vari attributi che rappresentano la grandezza (Heigth), la posizione nella Grid (Margin, VerticalAlignment e HorizontalAlignment), il nome (Name), l'evento associato (Click o Checked, a seconda del tipo di controllo) e altri come FontFamily, FontSize e Foreground.

La finestra ha come titolo "SW validazione algoritmi di descrizione tonale", ed è stato utilizzato come immagine icona il file "logo.jpg", presente all'interno della cartella SW_Tesi, per cui questa icona comparirà al fianco del titolo. Lo sfondo è di colore CornflowerBlue. L'immagine in basso a destra è "freesound.png", ed è stata scaricata dal sito Freesound.

All'interno della Grid sono presenti due Label: "Elenco API di descrizione tonale" e "ID file audio", entrambi con testo in rosso, in Arial e con fontsize 18. Sono presenti 16 radioButton, ognuno corrispondente all'API che si vuole utilizzare in quel momento. Ad ognuno di essi è

associato l'evento checked = "tonal_checked". In un'applicazione WPF ogni evento può essere gestito come un metodo, il cui corpo è presente nel codice C#: il fatto di usare sempre lo stesso metodo per ogni radioButton ha il vantaggio di poter gestire un unico evento per tutti i radio Button, qualsiasi sia quello selezionato; in questo modo è possibile selezionare solamente una API alla volta. Quando avviene l'evento checked (cioè quando un radio Button viene selezionato) viene eseguita la funzione corrispondente. Un radio Button può essere selezionato dall'utente, ma non può essere deselezionato. [8]

La textBox a destra del Label "ID file audio" serve per immettere l'id del file audio di cui si vuole calcolare la funzione, e si chiama textBox_file.

Infine sono presenti tre bottoni. Il bottone INVIA serve per inviare la richiesta http a Freesound, a seconda dell'API selezionata e dell'ID immesso. La risposta apparirà dopo qualche secondo nella listBox.

Il pulsante SALVA serve per salvare la risposta (cioè il contenuto della listBox) in formato testuale. Il pulsante CLEAR serve per svuotare la listBox e la textBox. A questi pulsanti sono associati rispettivamente gli eventi click = "invia_Click", "salva_Click" e "clear_Click". Quando uno di questi pulsanti viene premuto, viene eseguita la funzione corrispondente al pulsante selezionato. Affinché venga creato il metodo corrispondente all'evento, basta fare doppio click sul pulsante a cui si vuole assegnare l'evento.

2.2 Il codice C#

Nel file Window1.xaml.cs vengono gestiti gli eventi tramite metodi, che sono definiti all'interno della classe Window1. Nel programma è presente solo questa classe.

```
public partial class Window1 : Window
```

2.2.1 Le librerie

Prima del vero e proprio codice C# sono presenti le librerie standard. Tutte le funzioni e le classi sono dichiarate all'interno del namespace (chiamato, in questo caso, SW_Tesi). Al momento della creazione di un nuovo progetto WPF le librerie standard già presenti nel codice sono queste:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

Le librerie che sono state aggiunte per il funzionamento del programma sono queste:

```
using Microsoft.Win32;
using System.Net;
using System.IO;
```

Il namespace Microsoft.Win32 fornisce due tipi di classi: quelle che gestiscono eventi per il sistema operativo e quelle che manipolano il registro di sistema. Quelle che interessano in questo contesto sono le prime, in particolare le classi OpenFileDialog e SaveFileDialog che servono, rispettivamente, per gestire una finestra di dialogo affinché si possa aprire un file all'interno del programma o salvare il file del programma.

Il namespace System.Net fornisce una semplice interfaccia per la maggior parte dei protocolli usati sulla rete. Le classi più importanti che derivano da questo namespace sono WebRequest e WebResponse, entrambe usate in questo programma, e servono rispettivamente per fare una richiesta http a un Uniform Resource Identifier (URI) e fornire una risposta da un URI.

Infine la libreria System.IO serve per il supporto dell'input e output, come leggere e scrivere dati su uno stream, funzione svolta dalle classi StreamReader e StreamWriter. [8]

2.2.2 La classe Window1 e i metodi

Il primo metodo che viene compilato ed eseguito è Window1(), corrispondente al Main:

```
public Window1()
{
    InitializeComponent();
}
```

Il metodo è pubblico, quindi è visibile all'interno di altre possibili classi, e il metodo InizializeComponent() serve per caricare le pagine compilate.

All'interno della classe Window1 sono state poste tre variabili:

```
string URL = "http://www.freesound.org/api/sounds/";
string tonal = "";
string api_key = "?api_key=63cd4d46a02f473b9f4843289c0301b1";
```

Queste variabili devono essere visibili in tutti i metodi della classe Window1, quindi sono stati posti fuori dai metodi. Sono variabili di tipo stringa, e rappresentano parti di URI necessari per eseguire le chiamate http per il funzionamento delle API. La stringa URL è la prima parte dell'indirizzo, e la prima parte che tutti gli indirizzi hanno in comune. La stringa tonal è inizializzata vuota, e deve contenere la parte di indirizzo contenente il nome dell'API da usare in quel momento; verrà gestita dal metodo dei RadioButton. La stringa api_key è l'ultima parte dell'indirizzo, è comune a tutti gli indirizzi e rappresenta l'API key che è stata consegnata al momento della registrazione al sito. La parte di URI mancante è quella che deve contenere l'ID del file audio da analizzare: verrà gestita in un metodo come textBox_file.Text (testo della textBox).

I metodi successivi sono tutti privati, cioè sono accessibili solamente all'interno della classe a cui appartengono. Inoltre sono tutti di tipo void, cioè non restituiscono niente (non c'è return). Ogni metodo riceve in ingresso due oggetti: un object e un RoutedEventArgs. L'oggetto object supporta tutte le classi della gerarchia di classi .NET Framework e fornisce informazioni di basso livello alle classi derivate. L'oggetto RoutedEventArgs contiene informazioni sullo stato e dati eventi associati ad un evento indirizzato. Entrambi gli oggetti sono stati creati al momento della creazione del metodo, ma non verranno utilizzati all'interno del metodo.

Il primo metodo privato è il seguente:

```
private void tonal_Checked(object sender, RoutedEventArgs e)
{
    if (sender == chords_changes_rate)
        tonal = "/analysis/tonal/chords_changes_rate/";
    else if (sender == chords_histogram)
        tonal = "/analysis/tonal/chords_histogram/";
    else if (sender == chords_key)
        tonal = "/analysis/tonal/chords_key/";
```

```
    else if (sender == chords_number_rate)
        tonal = "/analysis/tonal/chords_number_rate/";
    else if (sender == chords_progression)
        tonal = "/analysis/tonal/chords_progression/";
    else if (sender == chords_scale)
        tonal = "/analysis/tonal/chords_scale/";
    else if (sender == chords_strength)
        tonal = "/analysis/tonal/chords_strength/";
    else if (sender == hpcp)
        tonal = "/analysis/tonal/hpcp/";
    else if (sender == key_key)
        tonal = "/analysis/tonal/key_key/";
    else if (sender == key_scale)
        tonal = "/analysis/tonal/key_scale/";
    else if (sender == key_strength)
        tonal = "/analysis/tonal/key_strength/";
    else if (sender == thpcp)
        tonal = "/analysis/tonal/thpcp/";
    else if (sender == tuning_diatonic_strength)
        tonal = "/analysis/tonal/tuning_diatonic_strength/";
    else if (sender == tuning_equal_tempered_deviation)
        tonal = "/analysis/tonal/tuning_equal_tempered_deviation/";
    else if (sender == tuning_frequency)
        tonal = "/analysis/tonal/tuning_frequency/";
    else if (sender == tuning_nontempered_energy_ratio)
        tonal = "/analysis/tonal/tuning_nontempered_energy_ratio/";
}
```

Questo metodo è associato all'evento di ogni radio Button. Se uno dei radio Button viene selezionato, viene eseguito l'evento corrispondente. Come già detto, è possibile selezionare solo un radio Button alla volta, poiché gli eventi associati ai bottoni hanno lo stesso nome. Nel metodo è presente un lungo elenco di "if/else if": questo significa che, a seconda del "sender" (il bottone "checcato") la stringa tonal diventa una parte di URI corrispondente al radio Button selezionato. Questa va ad aggiungersi alle altre parti di URI, cioè alle stringhe URL e api_key.

Il seguente metodo privato è il più importante:

```
private void invia_Click(object sender, RoutedEventArgs e)
{
    if (textBox_file.Text != "" && tonal != "")
    {
        listBox1.Items.Clear();
        string comp_URL = URL + textBox_file.Text + tonal + api_key;
        Uri uri = new Uri(comp_URL);
        WebRequest req = WebRequest.Create(uri);
        try
        {
            WebResponse resp = req.GetResponse();
            Stream stream = resp.GetResponseStream();
            StreamReader sr = new StreamReader(stream);
            while (!sr.EndOfStream)
                listBox1.Items.Add(sr.ReadLine());
            sr.Close();
        }
```

```
        catch (Exception ex)
        {
            listBox1.Items.Add(ex.ToString());
        }
    }
    else
        MessageBox.Show("Prego scegliere l'API e inserire l'ID del file audio");
}
```

E' il metodo associato all'evento del Button INVIA: quando il pulsante viene premuto, viene eseguito il metodo invia_Click, viene quindi inviata una richiesta http al server, il quale restituisce una risposta che viene scritta sulla listBox.

Questo è quello che succede nello specifico.

Se sia textBox_file.Text sia tonal non sono stringhe vuote, allora vengono eseguite le istruzioni all'interno dell'if. Questo significa che le istruzioni vengono eseguite se, prima di premere il pulsante INVIA, l'utente ha scritto l'ID del file audio da analizzare nella text Box e ha premuto uno dei radio Button corrispondente all'API che si vuole usare. Se una di queste condizioni non è verificata il compilatore passa direttamente all'istruzione sotto else: verrà aperta una MessageBox, una finestra di messaggio con un pulsante 'ok', con scritto "Prego scegliere l'API e inserire l'ID del file audio". MessageBox è una classe di System.Windows e Show è il metodo necessario per mostrare la finestra di messaggio.

Se le condizioni dell'if sono verificate, prima di tutto la listBox viene svuotata, poiché può contenere una precedente risposta. Dopodiché viene montata la stringa comp_URL, cioè l'indirizzo Freesound delle API: la stringa è composta dalle variabili URL, textBox_file.Text (l'ID del file audio), tonal e api_key.

Ora l'indirizzo è pronto, occorre mandare una richiesta http al server contenente l'indirizzo, e farsi restituire la risposta. Viene creato per questo scopo l'istanza della classe Uri, la quale fornisce una rappresentazione in forma di oggetto di un identificatore URI; l'istanza viene creata con new, con l'indirizzo comp_URL specificato tra parentesi → `Uri uri = new Uri(comp_URL);`.

Viene poi effettuata la richiesta all'URI, tramite la classe WebRequest: si crea la variabile req e si istanzia la classe tramite il metodo Create(), con il quale si invia la variabile uri. WebRequest è una classe di System.Net, una delle librerie aggiunte manualmente.

```
WebRequest req = WebRequest.Create(uri);
```

Le rimanenti istruzioni dell'if sono poste all'interno di un costrutto try/catch. Sotto il try viene fornita la risposta http tramite la classe di System.Net WebResponse, inizializzando la variabile resp. L'istanza di WebRequest viene usata per sottoporre a override il metodo di WebRequest GetResponse(), in modo da restituire una risposta a una richiesta Internet.

```
WebResponse resp = req.GetResponse();
```

La risposta è arrivata, ora bisogna scriverla sulla listBox. Una risposta http viene gestita tramite flusso di dati (stream). Per prima cosa si crea la variabile stream usando la classe di System.IO Stream; la sua istanza viene utilizzata per sottoporre a override il metodo di Stream GetResponseStream(), così facendo viene restituito il flusso di dati dalla risorsa Internet.

```
Stream stream = resp.GetResponseStream();
```

Poi occorre leggere i caratteri dal flusso di bit, questa funzione è svolta dalla classe di System.IO StreamReader: viene creata la sua istanza con new, in modo tale che arrivi il flusso specificato da leggere (la variablie stream) → `StreamReader sr = new StreamReader(stream);`. Il flusso di dati è stato aperto, e rimarrà aperto finché non verrà eseguita l'istruzione sr.Close().

Ora sr contiene tutti i caratteri del flusso di dati della risposta http all'indirizzo comp_URL, adesso occorre inserirla dentro la listBox. Con un ciclo while vengono fatte scorrere le righe che compongono sr. Il ciclo continua finché sr non raggiunge la fine (`while (!sr.EndOfStream)`). Il questo modo ogni riga del flusso sr viene aggiunta alla listBox, in cui una riga del flusso corrisponde a un Item → `listBox1.Items.Add(ex.ToString());`. Quando il flusso di dati finisce si esce dal ciclo e sr viene chiuso.

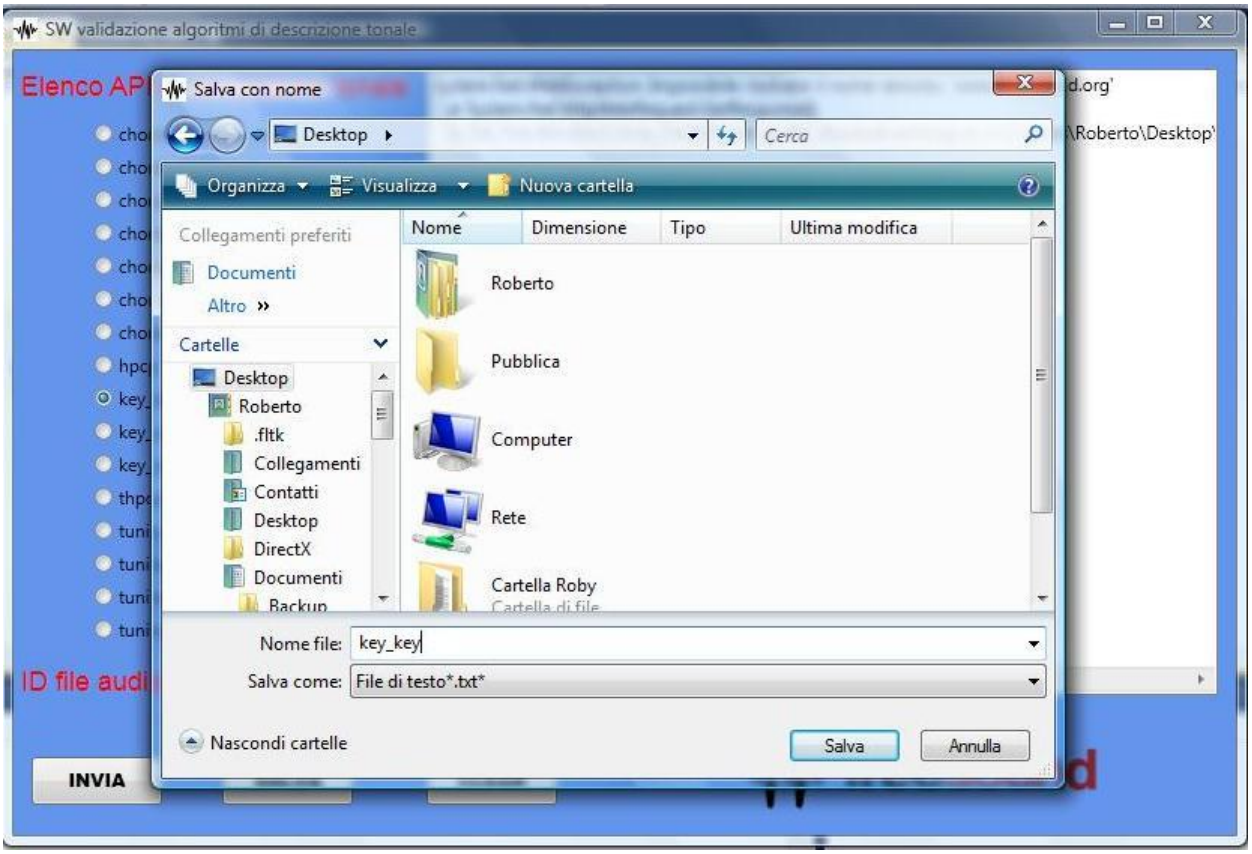
Le eccezioni all'interno del blocco try posso avvenire ad esempio perché l'indirizzo comp_URL non esiste, oppure perché manca la connessione Internet, e quindi in entrambi i casi il server non riesce a restituire la risposta. Queste eccezioni vengono gestite dentro il blocco catch. Viene creata la variabile Exception ex, che viene aggiunta alla listBox → `listBox1.Items.Add(ex.ToString());`

La classe Exception rappresenta gli errori che si verificano durante l'esecuzione dell'applicazione. Il metodo ToString() ha lo scopo di creare e restituire una rappresentazione in forma di stringa dell'eccezione corrente.

Questo metodo serve, invece, per salvare la risposta http (cioè il contenuto della listBox) come file di testo, e viene eseguito quando viene premuto il pulsante SALVA:

```
private void salva_Click(object sender, RoutedEventArgs e)
{
    SaveFileDialog saveas = new SaveFileDialog();
    saveas.Filter = "File di testo*.txt*|*.txt|Tutti i file|*.*";
    saveas.ShowDialog();
    if (saveas.FileName != "")
    {
        StreamWriter sw = new StreamWriter(saveas.FileName);
        foreach (string c in listBox1.Items)
            sw.Write(c);
        sw.Close();
    }
}
```

Prima di tutto viene aperta una finestra di dialogo con la classe di Microsoft.Win32 SaveFileDialog, la quale viene istanziata con new. Questa classe rappresenta una finestra di dialogo che permette all'utente di salvare il file con nome, selezionando la cartella di destinazione.



SaveFileDialog

Viene poi utilizzato il metodo di SaveFileDialog Filter, che ottiene o imposta la stringa di filtro che determina i tipi di file visualizzati. In questo caso l'utente può scegliere se visualizzare solo i file di testo con estensione txt, oppure tutti i file.

```
saveas.Filter = "File di testo*.txt*|*.txt|Tutti i file|*.*";.
```

Effetto del Filter nella stessa SaveFileDialog

Il metodo ShowDialog() consente di visualizzare la finestra saveas.

Se il fileName di saveas è diverso da stringa vuota, cioè se l'utente ha inserito il nome del file nella finestra di dialogo, vengono eseguite le istruzioni all'interno dell'if, permettendo di salvare il file. Anche in questo caso i caratteri del file da salvare vengono gestiti come flusso di bit, ma invece di leggere il flusso questa volta lo si deve scrivere su un file. Occorre quindi creare un'istanza della

classe StreamWriter, che è classe di System.IO come StreamReader, con new, in questo modo l'istanza viene inizializzata per il file specificato nel percorso specificato.

```
StreamWriter sw = new StreamWriter(saveas.FileName);.
```

Ora è stato creato un flusso con lo scopo di scriverlo sul file specificato dall'utente. Adesso bisogna inserire il contenuto della listBox nel flusso sw, questa operazione viene eseguita con il costrutto foreach, che serve per scorrere l'insieme di Items della listBox.

Gli Items della listBox vengono fatti scorrere affinché il contenuto di ognuno di essi venga copiato nella stringa c e, per ogni ciclo, la stringa viene scritta nel flusso sw.

```
foreach (string c in listBox1.Items)
    sw.Write(c);
```

Infine il flusso viene chiuso con `sw.Close();` e il file viene salvato nella cartella specificata.

L'ultimo metodo permette all'utente di "partire da capo", svuotando la listBox e la textBox. Viene eseguito se si preme il pulsante CLEAR.

```
private void clear_Click(object sender, RoutedEventArgs e)
{
    textBox_file.Text = ""
    listBox1.Items.Clear();
}
```

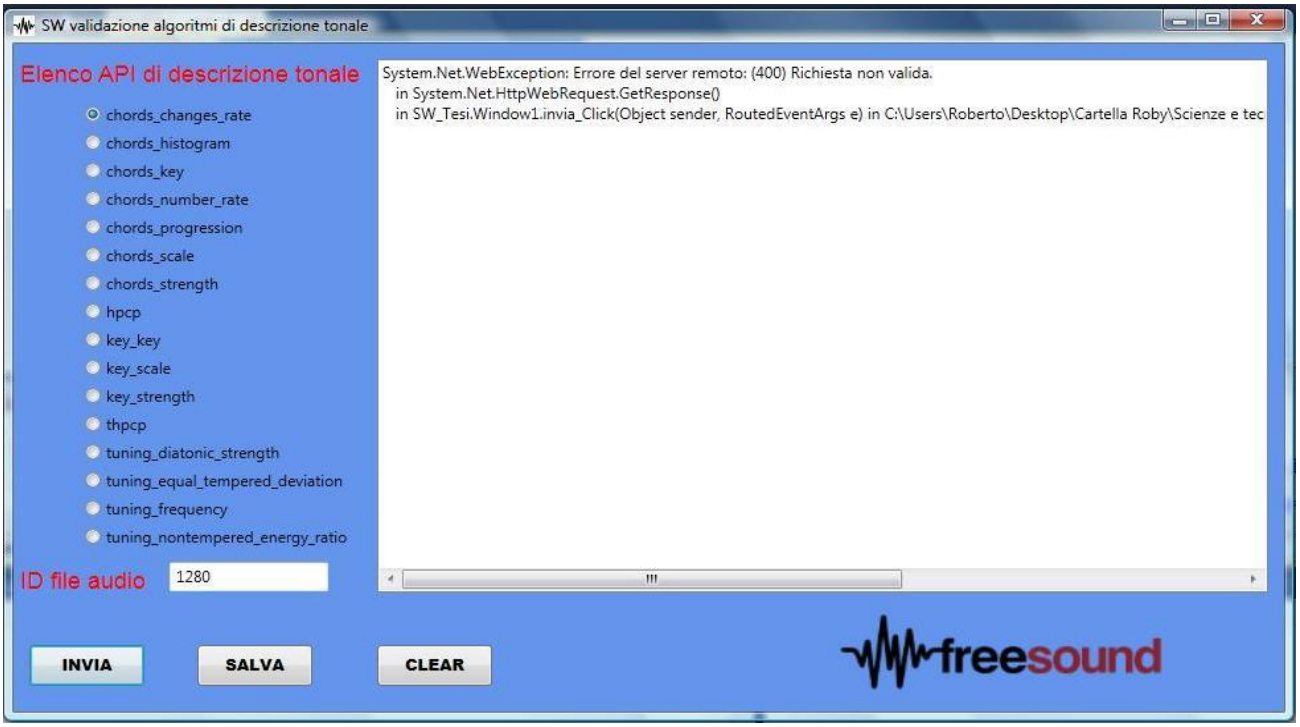
3. I test

3.1 Alcune considerazioni iniziali

Questo capitolo è dedicato ai risultati delle risposte delle API di descrizione tonale. Per farle funzionare è stato utilizzato il software; l'inserimento manuale dell'indirizzo sulla barra del browser è stato effettuato solo a scopo di test, per verificare che il programma funzionasse correttamente.
Sono stati testati una trentina di file audio di natura diversa, verranno presi in esame quelli più significativi per verificare il corretto funzionamento delle API con ciascuno di essi. Prima verranno presi in esame i file audio contenenti melodie o accordi, file coi quali le API dovrebbero funzionare senza alcun problema, in seguito verranno analizzati file meno "musicali", come voci, suoni e rumori, per verificare il comportamento delle librerie anche in questi casi.
La prima considerazione da fare è che, dei 16 algoritmi di descrizione tonale, quelli che restituiscono un risultato sono solamente 4:

- key_scale;
- key_key;
- key_strength;
- tuning_frequency.

Con tutti gli altri non solo non viene restituito alcun risultato, ma il server dà anche l'errore 400 Bad Request: significa che la richiesta non può essere soddisfatta per errori di sintassi. Gli algoritmi non funzionanti sono stati testati con tutti i tipi di file audio possibile, ma la risposta è stata sempre la stessa. Questo comportamento è un pò strano, perché la sintassi da usare è sostanzialmente la stessa per tutti gli indirizzi, l'unica cosa che cambia è il nome dell'API. Gli algoritmi che non restituiscono risposte sono tutti quelli che dipendono dall'algoritmo HPCP e il gruppo dei chords. Un rapido test sul funzionamento delle altre API che non interessano ai fini di questo elaborato ha dato il risultato che anche la maggior parte di esse non funzionano.
La spiegazione di tutto ciò è che le funzioni sono sperimentali e ancora in fase di sviluppo, per questo motivo alcune di esse non restituiscono alcun risultato. In conseguenza di ciò, l'elaborato verterà solamente sulle 4 API funzionanti.
Per controllare la tonalità dei brani è stata utilizzata una tastiera midi virtuale.



Risposta del server con API non funzionanti

3.2 Risultati dei test sui file audio

3.2.1 Brano malinconico in FA# maggiore

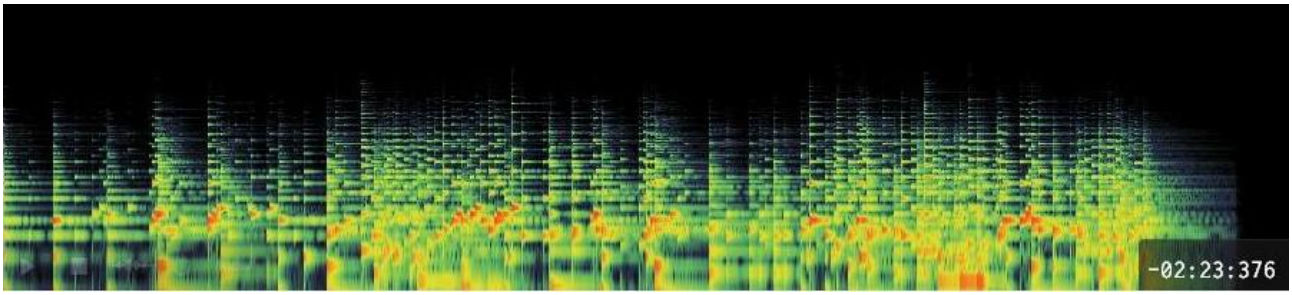
Il primo suono che verrà analizzato ha il titolo "melancholic burble in F# major.mp3". Si tratta di una melodia di pochi minuti, suonata con un pianoforte a coda, in fa# maggiore (dal titolo), molto lenta e malinconica. Dall'ascolto la melodia è relativamente semplice, senza grandi variazioni, con un accompagnamento di accordi fissi, e non sembra che ci siano modulazioni, gli accordi del brano gravitano sempre intorno alla tonica. Il file è in formato mp3 con un bitrate di 192 kbps, il suo ID è 12693.

Il risultato restituito da key_key è stato: "F#". Quello restituito da key_scale è stato: "major". Quindi gli algoritmi hanno restituito la tonalità e il modo corretti.

Key_strength ha dato come risultato "0.81457310915000003". Questo significa che c'è la probabilità dell'81% che la melodia sia veramente in fa# maggiore, per cui i risultati restituiti da key_key e da key_scale sono attendibili.

Tuning_frequency ha restituito "442.548919678", che è la frequenza sulla quale è accordato il brano; quindi la melodia è accordata su un LA leggermente crescente rispetto a quello di default (440 Hz).

Melodia in FA# maggiore



Spettrogramma dello stesso file

3.2.2 Minuetto di Boccherini

Questo file audio ha come titolo "violin minuet_boccherini (edit).wav", è in formato wav con una frequenza di campionamento di 44.100 Hz, 16 bit di quantizzazione e 2 canali. L'ID del file è 25481. Si tratta della celebre melodia del minuetto di Luigi Boccherini suonata da un violino solo senza accompagnamento (solo melodia).

La melodia è in LA maggiore ed è composta da 4 periodi. I primi 2 sono uguali, partono da LA maggiore per poi finire con una cadenza alla dominante, in MI maggiore. Il terzo è in LA minore, mentre il quarto periodo è simile ai primi 2, con la differenza che, invece di finire in dominante, termina in tonica. Quindi è una melodia, seppur abbastanza semplice, un po' modulante. Inoltre il violino, a differenza del pianoforte o della chitarra, non ha una tastiera, quindi alcune frequenze

sono “oscillanti” intorno alla nota voluta, in quanto l’esecutore, per quanto possa essere bravo e preparato, non avendo punti di riferimento visivi difficilmente riuscirà ad eseguire la nota voluta alla frequenza esatta.

I risultati restituiti da key_key e da key_scale sono, rispettivamente, “A” e “major”, quindi l’algoritmo è riuscito a trovare la tonalità giusta, nonostante le modulazioni e le note non perfettamente accordate, e nonostante si tratti solo ed esclusivamente di una melodia, senza accordi o accompagnamento.

Key_strength ha restituito “0.83609807491300003”, c’è quindi la probabilità dell’83% che la tonalità restituita sia quella effettiva, ed è una percentuale più alta di quella trovata per il file precedente.

Tuning_frequency ha dato come risultato “438.98455810500002”, quindi il violino che ha suonato questa melodia è accordato su un LA leggermente calante rispetto ai 440 Hz.

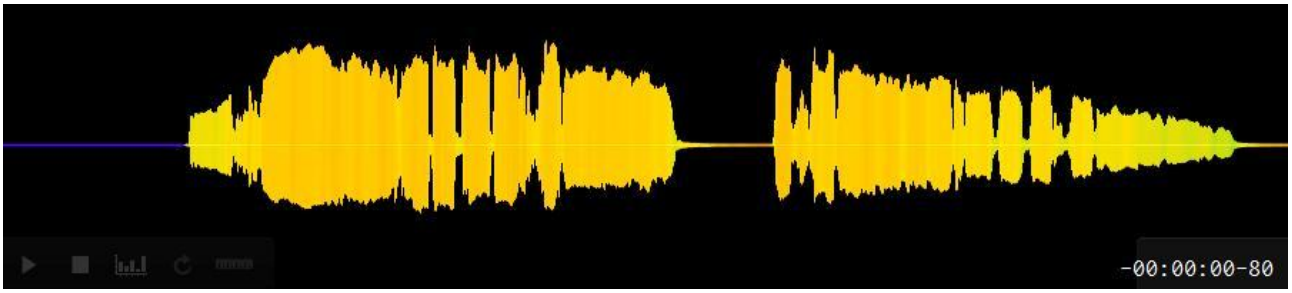
3.2.3 Melodia di tromba

Il suono dal titolo “TrumpetLoop01.aif” contiene una melodia in DO minore suonata da una tromba. Il file ha estensione aif, ha una frequenza di campionamento di 44.100 Hz e 16 bit di quantizzazione, e il suo ID è 2323.

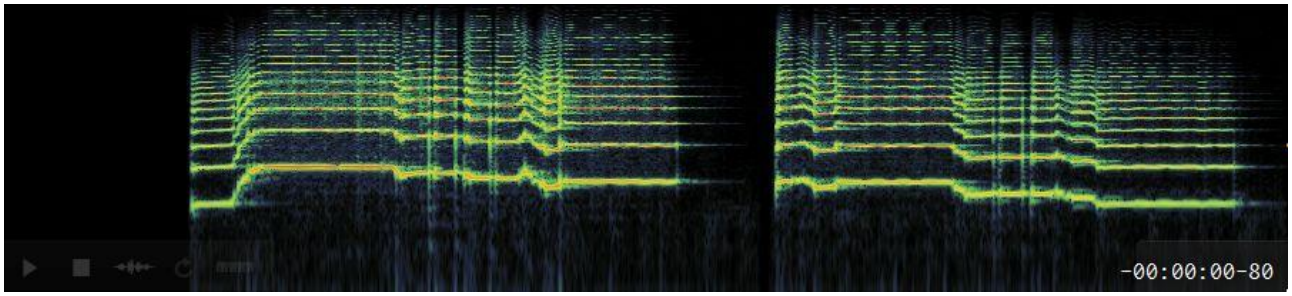
La melodia è molto breve, formata da 2 frasi, inizia con un DO e termina con il DO dell’ottava sotto. Come la melodia precedente, suonata dal violino, anche in questo caso alcune note non sono perfettamente intonate, ma oscillano intorno alla frequenza a cui si riferisce la nota, forse anche maggiormente rispetto al violino.

Key_key e key_scale hanno dato come risultati “C” e “minor”, quindi anche in questo caso, come in quello precedente, le funzioni hanno restituito la tonalità corretta.

Key_strength ha restituito “0.63064074516299995”, quindi la percentuale che la tonalità sia effettivamente quella trovata è un po’ più bassa delle precedenti, questo perché le note della melodia sono meno definite rispetto alle altre due melodie. Tuning_frequency ha restituito “437.71853637700002”.



Sonogramma della melodia di tromba



Spettrogramma della melodia di tromba

3.2.4 Frammento della “Hammerklavier” di Beethoven

Questo file audio contiene un frammento della sonata per pianoforte di Ludwig Van Beethoven “Hammerklavier”. Il file si chiama “(My 'Signature') HAMMERKLAVIER-SLOW MVT.aif”, ha ID 46215, è in formato aif, con frequenza di campionamento di 44.100 Hz e 16 bit di quantizzazione.

Il frammento della sonata è in FA# minore, rimane sempre nella stessa tonalità senza modulare, gli accordi gravitano tutti intorno alla tonica, ma finisce con un accordo di dominante (DO#).

Le API key_key e key_scale hanno restituito “F#” e “minor”, quindi hanno centrato la tonalità corretta.

Key_strength ha dato come risultato “0.55904060602200001”, quindi c’è la probabilità del 55% che la tonalità giusta sia effettivamente quella trovata dagli algoritmi precedenti. E’ strano che restituisca una percentuale così bassa, in un brano in cui il pianoforte sembra essere perfettamente accordato. Probabilmente il fatto che il frammento finisca sulla dominante influisce sul risultato di questa API.

Tuning_frequency restituisce “441.01779174799998”.

3.2.5 Melodia horror di pianoforte con effetto

Il file audio rinominato in “Music_Classic Horror Tune vers.3.mp3” e con ID 89600 contiene una melodia di pianoforte, senza accompagnamento o accordi. E’ in formato mp3, con un bitrate di 320 kbps e 2 canali. Inoltre al pezzo è stato sommato un effetto digitale che dà alle note una sensazione di tremolo.

Il pezzo è diviso in due parti perfettamente uguali, la melodia è molto semplice, la tonalità è una via di mezzo tra SOL# minore e LA minore, anche se sembra essere più vicina a SOL# minore . La

melodia è composta da un arpeggio minore, si ferma sul sesto grado e termina con una scala discendente fino alla tonica; questo ripetuto due volte.
Key_key e key_scale hanno dato come risultati "G#" e "minor", quindi, anche se l'accordatura del pianoforte è molto calante rispetto a quella corretta, gli algoritmi restituiscono comunque la tonalità che più si avvicina a quella del pezzo, cioè SOL# minore.
Key_strength ha restituito "0.57768577337299998", una percentuale relativamente bassa e molto simile a quella del brano precedente, anche se in questo caso è più veritiera rispetto all'Hammerklavier.
Tuning_frequency ha dato come risultato "434.44400024399999", quindi secondo l'algoritmo il pezzo è accordato su un LA leggermente calante rispetto ai 440 Hz. Questo però è sbagliato perché gli altri algoritmi hanno restituito la tonalità SOL# minore e il brano, rispetto a questa tonalità, è crescente, non calante.

3.2.6 Fuga in SOL minore di Bach

Quest'altro brano rappresenta una registrazione della fuga in SOL minore di Johann Sebastian Bach dal primo volume del Clavicembalo Ben Temperato, suonata con un pianoforte. Questo file è stato inserito per testare meglio il funzionamento delle API. Il file è in formato mp3, con un bitrate di 128 kbps e stereo, ha il titolo "Bach_fugue_in_Gmin.mp3" e il suo ID è 135883.
Una fuga è una composizione strumentale, in questo caso a 4 voci, composta da un elemento tematico chiamato soggetto, che viene esposto all'inizio da una sola voce. Quando la prima voce finisce di esporre il soggetto entra una seconda voce che espone lo stesso soggetto trasportato di una quinta ascendente (chiamato risposta), mentre la prima voce esegue il controsoggetto, che da ora in poi accompagnerà sempre il soggetto e la risposta. Questo procedimento continua a mano a mano che entrano le rimanenti voci, dopodiché continuano ad alternarsi soggetto e risposta, intervallati da brevi progressioni, chiamate divertimenti, di solito modulanti. La fuga finisce con gli stretti, cioè le entrate di soggetto e risposta sono più ravvicinate tra di loro rispetto all'esposizione. La fuga termina nella tonalità d'impianto, di solito con un pedale di tonica. [9]
La fuga quindi non è composta da una melodia e da un accompagnamento accordale, ma è costituita da più voci indipendenti, in cui ciascuna voce rappresenta una melodia; gli accordi sono formati dalla somma delle voci.
Questa fuga è in SOL minore, dopo l'esposizione c'è una progressione modulante che porta il pezzo nella relativa maggiore (SIb maggiore). In questa tonalità c'è una nuova esposizione di soggetto e risposta, dopodiché cominciano gli stretti, e il brano attraversa varie tonalità, da SIb maggiore a Mib

maggiore, a DO minore. Infine, dopo un lungo divertimento, torna alla tonalità di impianto con degli stretti molto più ravvicinati. La fuga termina con la terza piccarda, con un accordo di SOL maggiore, senza pedale.
Le API key_key e key_scale hanno restituito "G" e "minor": gli algoritmi hanno quindi trovato la tonalità giusta, nonostante l'assenza di un accompagnamento accordale, le modulazioni e nonostante la fuga finisca con un accordo maggiore.
La funzione key_strength ha dato come risultato "0.62900769710500004", che è una percentuale piuttosto bassa, però compatibile con il fatto che sono presenti molte modulazioni talvolta anche molto ravvicinate: spesso in una stessa battuta, a causa del veloce movimento delle voci, sono presenti anche sei o sette accordi diversi, soprattutto dominanti secondarie.
Tuning_frequency ha restituito la frequenza "437.71853637700002".

3.2.7 Loop in SOL# minore

Il file chiamato "memento_to_Bachs_fugue_II.wav" e avente ID 22125 contiene una breve melodia in SOL# minore con degli accordi di accompagnamento, suonata con un pianoforte. Il file è in formato wav, con una frequenza di campionamento di 44.100 Hz e 16 bit.
Il breve frammento audio si ripete tre volte sempre in modo identico, e rappresenta una cadenza perfetta (I – IV – V – I).
Key_key e key_scale hanno dato "G#" e "major", quindi l'algoritmo key_scale ha sbagliato: è vero che il pezzo è in SOL#, ma minore, non maggiore. Tra l'altro, a parte l'accordo di dominante, tutti gli accordi presenti sono minori, sia quello di tonica che quello di sottodominante.
Il risultato di key_strength è stato "0.77526402473400002", una percentuale stranamente non troppo alta, considerando che non ci sono dubbi sulla tonalità del brano. Forse la percentuale non è così alta come ci si aspetterebbe perché key_scale ha sbagliato il risultato. Tuning_frequency ha restituito "442.29336547899999".
Questo è stato il primo brano con cui l'API key_scale ha dato un risultato errato, e stranamente su un brano in cui la tonalità è chiara.

3.2.8 Kyrie in SI minore di Bach con canto di uccello

Questo file audio, chiamato "Bach B-Minor Mass Kyrie w:Reinsamba's Robin.aif " e con ID 44012, contiene un frammento del Kyrie in SI minore di Johann Sebastian Bach per organo mixato con il

canto di un uccello. Il file è in formato aif, con una frequenza di campionamento di 48.000 Hz e con un bitrate di 32 bit.
Il pezzo è un breve frammento di pochi secondi, sono presenti vari accordi, tra cui i più udibili sono il SI minore e il MI minore (sottodominante). Il canto di uccello in sottofondo è così forte che quasi copre la melodia.
I primi due algoritmi hanno dato come risultato "B" e "minor", per cui, nonostante il suono dell'uccello sovrastasse il brano musicale, e nonostante l'accordo che si sente maggiormente fosse MI minore, le API hanno restituito la tonalità giusta.
L'algoritmo key_strength ha dato "0.64308291673700002", che è un risultato fin troppo alto rispetto a quello atteso, poiché il suono predominante è quello dell'uccello.
Tuning_frequency ha dato come frequenza di accordatura esattamente 440 Hz: è la prima volta che l'algoritmo restituisce il risultato di default.

3.2.9 Progressione di pianoforte

Il file audio "5. PIANO FILLS-HYMN LIKE.aif" con ID 47793 è in formato aif, ha frequenza di campionamento di 44.100 Hz e 16 bit di quantizzazione.
Il frammento audio, contenente il suono di un pianoforte, è composto da una progressione di accordi nella tonalità di DO maggiore, costruita su una semplice e breve melodia. Gli accordi sostanzialmente sono tutti di tonica, sottodominante e dominante, con una cadenza plagale alla fine (I – IV – I). E' presente anche un accordo di settima maggiore.
Le API key_key e key_scale hanno dato come risultati "C" e "major", ovviamente non c'erano dubbi sulla tonalità in quanto era chiara e il brano era perfettamente intonato.
Key_strength ha restituito "0.75807666778600002", un risultato relativamente basso rispetto alla chiarezza della melodia; forse il risultato è solo 75% perché sono presenti molti accordi che confondono l'algoritmo, in particolare la cadenza plagale alla fine.
Tuning_frequency ha dato "442.03793335" Hz.

3.2.10 Tre accordi di pianoforte

Questo file audio, intitolato "Flügel Bigbeat 01 - 132.wav" e con ID 25485, è in formato wav, con frequenza di campionamento di 44.100 Hz, 16 bit, ed è stereo. Contiene 3 accordi di pianoforte, suonati in modo ritmico uno dopo l'altro: DO minore, SIb maggiore e FA maggiore, tutti perfettamente intonati con la tastiera midi virtuale. In questo caso non sembrerebbe esserci una

tonalità di impianto, ma il brano insiste soprattutto sul primo accordo, per cui la tonalità di impianto si potrebbe considerare DO minore.
Le funzioni key_key e key_scale hanno dato come risultati "F" e "major", per cui l'algoritmo ha considerato il brano in FA maggiore, che è l'accordo con il quale il pezzo si conclude.
Key_strength ha restituito "0.57492458820299996", per cui la percentuale che la tonalità vera sia proprio quella calcolata dalle precedenti API è un po' bassa, anche se sopra il 50%: questo è dovuto alla presenza dei tre accordi, e al fatto che il pezzo termina in FA maggiore, ma comincia con un altro accordo.
Infine il risultato di tuning_frequency è stato "441.01779174799998".

3.2.11 Loop di chitarra

Questo file audio, intitolato "arpeggio5lop.wav" e con ID 1280, è in formato wav, ha 16 bit di quantizzazione e 44.100 Hz di campionamento.
Contiene un loop di accordi arpeggiati suonato con una chitarra e ripetuto due volte identiche. Gli accordi sono MI minore, DO maggiore con la 9a maggiore e DO maggiore. La tonalità di impianto, se si considera il primo accordo, potrebbe essere MI minore. Dal confronto con la tastiera midi virtuale l'intonazione della chitarra sembra leggermente calante.
I primi due algoritmi hanno restituito "C" "major", quindi hanno considerato come tonalità di impianto quella data dall'ultimo accordo (DO maggiore), come è successo riguardo al file precedente. Perciò si è capito che queste due API calcolano la tonalità a seconda dell'accordo finale del brano.
Key_strength ha restituito "0.777964830399", una percentuale maggiore di quella trovata per il file precedente, forse dovuta al fatto che, in effetti, l'accordo più presente in questo caso è DO maggiore.
Tuning_frequency ha dato come risultato "432.940917969", che è compatibile con il fatto che la chitarra è calante rispetto alla tastiera midi.

3.2.12 Accordo arpeggiato di pianoforte

Questo file e il prossimo contengono degli accordi singoli. Il file di questo paragrafo si chiama "PianoMood37.wav", è in formato wav, il suo ID è 29842, frequenza di campionamento e bit di quantizzazione sono rispettivamente 44.100 Hz e 16 bit.

L'accordo in questo caso è arpeggiato ed è stato suonato con un pianoforte. Il tipo di accordo sembrerebbe RE minore con la nona maggiore, per cui un accordo un po' difficile da individuare. Inoltre al suono è aggiunto un po' di riverbero.
Key_key e key_scale hanno restituito la tonalità "D" "minor", per cui, nonostante la presenza della nona che avrebbe potuto confondere l'algoritmo, le funzioni hanno trovato la tonalità giusta.
Key_strength ha dato "0.70319098234199995", la lieve incertezza dell'algoritmo è dovuta sicuramente alla presenza della nona dell'accordo.
La risposta di tuning_frequency è stata "440.25424194300001", per cui l'accordatura è quasi perfetta.

3.2.13 DO min 7

Questo file, dal titolo "Cminor7.wav", con ID 4197 e nello stesso formato del precedente, contiene un accordo di DO minore con la settima minore, suonato con uno strumento elettronico. L'accordo di DO min 7 è formato dalle seguenti note: DO – Mib – SOL – Sib. In questo caso la nota più acuta è SOL, la settima è al basso.
Gli algoritmi key_key e key_scale hanno dato come risultato "D#" e "major" che tradotto enarmonicamente sarebbe Mib maggiore (ricordiamo che gli algoritmi restituiscono solo tonalità con i diesis). Ovviamente la tonalità restituita è errata, ma ha una sua logica: infatti all'interno dell'accordo di DO min 7 sono presenti le note Mib – SOL – Sib, che insieme costituiscono proprio l'accordo di Mib maggiore. Probabilmente gli algoritmi non hanno riconosciuto la nota DO come facente parte dell'accordo, e hanno considerato solo le altre tre. Oppure hanno considerato il DO come sesta dell'accordo di Mib maggiore.
Key_strength ha dato come risposta "0.537560462952", una percentuale piuttosto bassa, ma compatibile con il fatto che all'interno dell'accordo ci sono 4 note e gli algoritmi precedenti hanno sbagliato tonalità.
Tuning_frequency ha restituito "439.238189697", anch'essa come la precedente molto vicina all'accordatura standard.

3.2.14 Loop di chitarra elettrica

Questo file ha titolo "REV LOOPS METAL GUITAR 7.mp3", l'ID è 41975, è in formato mp3 con un bitrate di 320 kbps. Si tratta di una sequenza di note suonate da una chitarra metal elettrica 4 volte, senza accordi. Dal confronto con la tastiera midi la tonalità di impianto sembrerebbe RE

minore, e le note gravitano tutte intorno all'accordo di tonica. Le note eseguite sono, senza contare le ripetizioni: RE, FA, RE, SOL, RE, LA, SOL, FA, SOL.
Le API key_key e key_scale hanno dato come risultati "G" e "minor", gli algoritmi hanno quindi restituito la tonalità data dall'ultima nota suonata. In effetti potrebbe essere logico, in quanto le note suonate possono appartenere anche a SOL minore naturale, e l'algoritmo avrà considerato che il pezzo partisse dalla dominante. Inoltre alla fine della sequenza la melodia gira intorno al SOL.
Key_strength ha restituito "0.61811745166800003", una percentuale non troppo alta, compatibile con il fatto che la sequenza parte da RE e fa un accordo di RE minore.
Tuning_frequency ha dato come frequenza di accordatura "434.69500732400002" Hz.

3.2.15 Coro

Su Freesound si trovano veramente pochi brani cantati, e quelli che si trovano sono poco significativi. Questo file, dal titolo "sound-singing9.wav", in formato wav con frequenza di campionamento 44.100 Hz e 16 bit, è uno dei più significativi.
Contiene una breve melodia vocale polifonica, cantata da due persone, una che esegue la melodia e l'altra l'accompagnamento. Non ci sono strumenti musicali. La melodia è in SOL maggiore, senza modulazioni o dominanti secondarie.
Key_key e key_scale hanno restituito "C" e "major", quindi, nonostante la tonalità sia molto chiara, i due algoritmi hanno sbagliato, seppur di poco, poiché DO maggiore è una tonalità vicina a SOL maggiore.
Key_strength ha dato "0.70685440301900004", una percentuale non troppo bassa a causa del fatto che le API precedenti hanno sbagliato.
Tuning_frequency ha restituito "432.44104003899997", un valore abbastanza veritiero, considerando che rispetto alla midi keyboard le voci sembrano leggermente calanti.

3.2.16 Donna che canta

Questo rappresenta l'ultimo file musicale che verrà analizzato. Ha titolo "katy-sings-laaoooaaa.wav", l'ID è 39914, è in formato wav, con una frequenza di campionamento di 44.100 Hz, 16 bit, ed è mono.
Contiene una melodia cantata da una donna, senza accompagnamento. La melodia è in MI maggiore è insiste principalmente sulla terza (SOL#), nota con la quale inizia e finisce il pezzo. L'estensione del brano va da DO# a LA della stessa ottava.

Key_key e key_scale hanno dato come risultati “C#” “minor”, che è la relativa minore di MI maggiore, ma non è la tonalità corretta. Probabilmente l’algoritmo ha dato questa tonalità perché il pezzo ad un certo punto passa sulla nota DO#, per poi salire sulla tonica.
Key_strength ha restituito “0.71664738655100002”, una percentuale molto simile a quella del file precedente poiché anche in questo caso la tonalità restituita è sbagliata, ma di poco.
Il risultato di tuning_frequency è stato 440 Hz precisi.

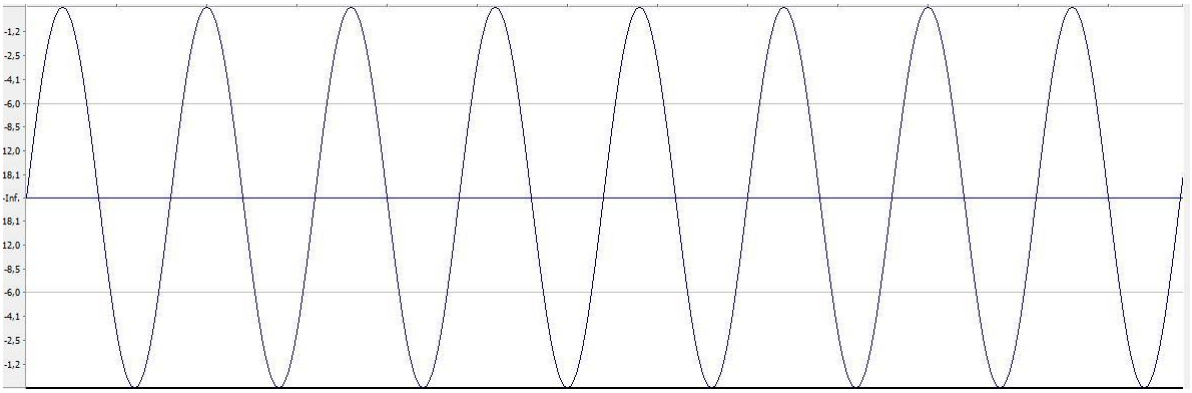
3.2.17 Suono grave

Questo file audio, dal titolo “SteamPipe 7 DarkPad C2.wav” e con ID 6663, contiene un suono grave creato con un sintetizzatore elettronico. Nonostante il titolo indichi chiaramente che la frequenza fondamentale del suono sia il DO della seconda ottava, dal confronto con la tastiera midi in realtà si tratta di un Mib (o RE#, come forse restituirà l’algoritmo). Inoltre si sentono un po’ di armonici. Il file audio è in formato wav, con frequenza di campionamento di 44.100 Hz e 16 bit di quantizzazione.
I primi due algoritmi hanno restituito la tonalità “G#” “minor”. Prima di tutto è ovvio che in questo caso non ha senso parlare di modo maggiore o minore, poiché il file è formato da un unico suono più una serie di armonici: ha senso parlare di modo in presenza di più suoni. L’algoritmo, però, deve comunque restituire una risposta (major o minor), e ha dato quella che a lui sembrava più pertinente. Anche il pitch è sbagliato, ma si può giustificare in quanto il SOL# è la quinta di RE#, quindi appartiene alla serie degli armonici di questo suono: probabilmente l’armonico corrispondente alla quinta ha più energia degli altri armonici, quindi l’algoritmo ha restituito quello.
Key_strength ha dato come la risposta “0.61131960153599996”, un risultato relativamente basso in quanto la frequenza trovata da key_key non è quella corretta.
Tuning_frequency ha restituito “436.960693359”.

3.2.18 Suoni contenenti toni puri

Questo frammento audio ha titolo “bleep10.wav”, è in formato wav con 44.100 Hz di campionamento e 16 bit e il suo ID è 24052. Contiene un tono puro di 10 secondi a 1000 Hz. Un tono puro è un suono composto da una sola frequenza, senza armoniche. [1]
La nota musicale che più si avvicina a 1000 Hz è il SI della quinta ottava, con frequenza 988 Hz, mentre il DO 6 ha una frequenza di 1046 Hz, quindi è più lontano. In effetti, dal confronto con la virtual midi keyboard il tono puro sembra un SI leggermente crescente.

Key_key e key_scale hanno restituito “E” e “minor”. Come nel caso di prima anche qui non ha senso parlare di modo, per lo stesso motivo. Però la frequenza trovata è proprio sbagliata: nel caso di prima poteva esserci il dubbio a causa degli armonici, in questo caso armonici non ce ne sono, quindi il dubbio non c’è, la frequenza è una ed è approssimativamente un SI. Non si capisce come mai l’API abbia restituito un risultato così lontano dalla nota più vicina alla frequenza di 1000 Hz.
Key_strength ha dato come risultato “0.57718229293800005”, un risultato fin troppo elevato, considerando che gli altri algoritmi hanno sbagliato in pieno.
La frequenza di riferimento restituita dalla funzione tuning_frequency è “445.62707519499997”. Anche in questo caso non si capisce come l’algoritmo abbia fatto a trovare una frequenza così vicina a quella standard.
Per quanto riguarda questo frammento audio le 4 API funzionanti di descrizione tonale hanno sbagliato completamente, ed è la prima volta che succede. Per verificare che questo sia stato solo un caso, sono stati testati altri due file audio simili a questo, contenenti sempre toni puri.



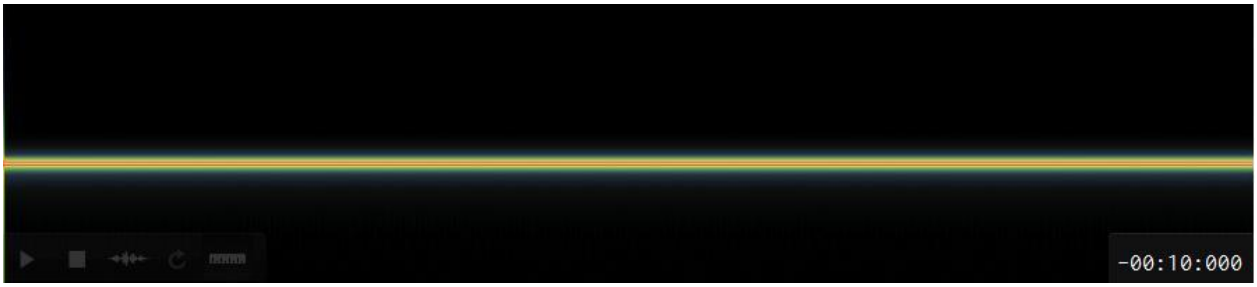
Forma d'onda di un tono puro, creata con Sound Forge

Il primo file audio si chiama “cassette tone.wav”, come il precedente è in formato wav, 16 bit, ma una frequenza di campionamento di 44.100 Hz e mono, il suo ID è 22684. Contiene una sequenza di toni puri consecutivi, con frequenze 100 Hz, 200 Hz, 500 Hz, 1000, 2000, 4000 e 8000 Hz. La prima e la seconda frequenza sono una via di mezzo tra SOL e SOL#, la prima all’ottava inferiore rispetto alla seconda; tutti gli altri, come nel file precedente, sono dei SI leggermente crescenti. Quindi in questo file sono presenti soprattutto frequenze simili al SI.
Con stupore i risultati delle API per questo file audio sono pressoché identici rispetto a quelli del file precedente: key_key e ley_scale danno “E” e “minor”, gli altri due algoritmi danno la stessa percentuale e la stessa frequenza di accordatura. Probabilmente key_key ha considerato come “tonica” sempre la frequenza di 1000 Hz e i suoni multipli.

Il secondo file si chiama “tone - 1 second sine.mp3”, ha ID 7437 ed è in formato mp3 con bitrate 192 kbps. Questa volta il file contiene un tono puro della durata di 1 secondo alla frequenza di 440 Hz.
Key_key e key_scale hanno incredibilmente restituito “D” e “minor”, risultati molto simili a quelli dati dai file audio precedenti: il modo è sempre minore, e la tonalità restituita è in tutti e tre i casi alla distanza di una quarta giusta dalla nota reale, prima da SI ha restituito MI, ora da LA ha restituito RE.
Key_strength ha dato come risultato una percentuale pressoché identica rispetto alle precedenti, cioè circa 57,7 %. Stavolta invece tuning_frequency ha restituito 440,25 Hz circa, e non 445 Hz circa come in precedenza.

Il file dal titolo “waveshaped F#3.wav”, in formato wav con 44.100 Hz di campionamento, 16 bit, 122944 di ID e mono, contiene un tono puro corrispondente a un FA#.
Anche in questo caso key_key e key_scale hanno restituito “B” e “minor”, quindi modo minore e tonalità alla distanza di una quarta giusta ascendente dalla frequenza reale. Key_strength ha dato circa 58,22% e tuning_frequency circa 439,49 Hz.

L’ultimo test è rappresentato dal file intitolato “ringing ears.wav”, con 48.000 Hz di campionamento e 16 bit. Contiene un tono puro a 13.500 Hz.
Nonostante la nota musicale a cui si riferisce sia diversa da quella data dai 1000 Hz, i primi due algoritmi restituiscono ancora “E” “minor”, key_strength circa 59% e tuning_frequency circa 435 Hz. In questo caso la tonalità restituita non dista una quarta rispetto al tono vero, in quanto 13.500 Hz corrisponde circa a un SOL.



Spettrogramma del tono puro di 1000 Hz

3.2.19 Canto tibetano

Il file “tibetan chant 4 colargol 2.aif ”, con ID 15488, è in formato aif ed ha una frequenza di campionamento di 44.100 Hz e 16 bit di quantizzazione.

Contiene un frammento di pochi secondi di un canto di un monaco tibetano. La caratteristica principale di questo canto è la capacità di una sola persona di produrre con la voce più note contemporaneamente. Comincia su una nota molto grave, per poi salire e rimanere circa sulla nota DO# della seconda ottava. Verso la fine la voce ridiscende nelle note più gravi. Non è possibile parlare di tonalità, poiché non è un canto che si basa sul sistema musicale occidentale, però il risultato più logico che l'algoritmo potrebbe dare è DO#, che è la nota che il monaco tiene più a lungo.

Key_key e key_scale hanno restituito "A#" e "minor". La nota che l'API ha dato, quindi, non è quella attesa, forse perché il canto tibetano non rimane sempre sulla stessa nota, e alla fine discende.

Key_strength ha dato "0.40712758898700002", la percentuale più bassa fin'ora restituita da questo algoritmo, ovviamente perché in questo caso non si può parlare né di tonalità né di modo, e gli algoritmi precedenti hanno dato risultati diversi da quelli più logici.

Tuning_frequency ha restituito "434.193115234": anche in questo caso non ha senso parlare di frequenza di accordatura, perché la voce umana non può essere accordata su una data frequenza, specie se il canto non è della tradizione occidentale. L'algoritmo, però, ha dovuto comunque restituire un risultato.

Anche in questo caso, quindi, le funzioni hanno dato risultati errati, o comunque non compatibili con quelli attesi.

3.2.20 Suono di un oscillatore digitale

Il file "010503C.mp3", in formato mp3 con un bitrate di 128 kbps, contiene il suono di un oscillatore digitale. Il suo ID è 9775.

Un oscillatore digitale è un dispositivo che permette la generazione di qualsiasi forma d'onda periodica semplicemente memorizzandone un periodo in una tabella che viene letta ciclicamente con passi diversi. L'oscillatore consente inoltre il controllo della fase ed una definizione molto precisa della frequenza di oscillazione. [10]

In questo file audio l'oscillatore digitale produce più suoni consecutivi con varie armoniche. Alcuni di essi sono a distanza di circa una terza tra loro, gli ultimi aumentano leggermente di frequenza, producendo un effetto di glissando, per cui è anche difficile dire il nome delle note che vengono prodotte. Si può dire però che alcuni di questi suoni, compreso il primo, sono circa DO# a diverse ottave.

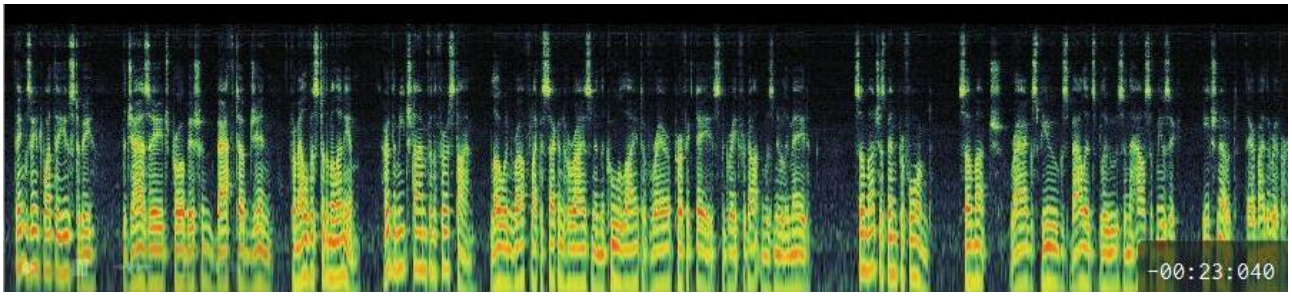
Le API key_key e key_scale hanno restituito “C#” e “minor”, quindi, come si è visto dal confronto con la tastiera midi virtuale, l’algoritmo ha riconosciuto le note principali come DO#, anche se è difficile dire che si tratti di modo minore, in quanto non è presente una melodia.
Key_strength ha dato come risultato “0.54376512765899998”, una percentuale non molto alta, compatibile con il fatto che, anche se l’algoritmo precedente ha restituito un risultato soddisfacente, il frammento audio non è una melodia, quindi non si può parlare di tonalità.
Tuning_frequency ha dato “445.62707519499997”, una frequenza che non è possibile commentare a causa del fatto che molti suoni non hanno una frequenza fissa.

3.2.21 Yeah

Il file audio “yeah.mp3”, con ID 19446, è in formato mp3, con un bitrate di 96 kbps e mono. Rappresenta un frammento audio di 1 secondo contenente la parola “Yeah” gridata da un gruppo di bambini.
Come tonalità è stata restituita “A” “major”: in effetti, confrontando il pitch della voce dei bambini con quella della tastiera midi, sembrerebbe che le voci si fermino in particolare sulla nota LA. Ovviamente però non ha senso parlare di modo.
Key_strength ha dato come risposta “0.62344676256200005”, una percentuale tutto sommato alta in quanto la nota che l’algoritmo precedente ha restituito è compatibile con il pitch della voce dei bambini.
Tuning_frequency ha restituito “444.341888428”. Anche qui non ha molto senso parlare di frequenza di riferimento.

3.2.22 Voce umana

Il file “David C Scott – Music River.mp3” ha ID 13630 ed ha un bitrate di 128 kbps. Contiene la voce di un uomo che recita una poesia di David Scott.



Spettrogramma della voce umana

Key_key e key_strength hanno restituito “C#” e “minor”, un risultato non veritiero in quanto l’uomo sta parlando, non sta cantando, e in una fase parlata non ha senso parlare di tonalità. Gli algoritmi avranno restituito il pitch intorno al quale la voce dell’uomo passa più spesso.
Key_strength ha dato come risultato “0.433148771524”, una percentuale abbastanza bassa compatibile con il fatto che il frammento audio non contiene un brano musicale.
Tuning_frequency ha restituito “443.57257080099998”.

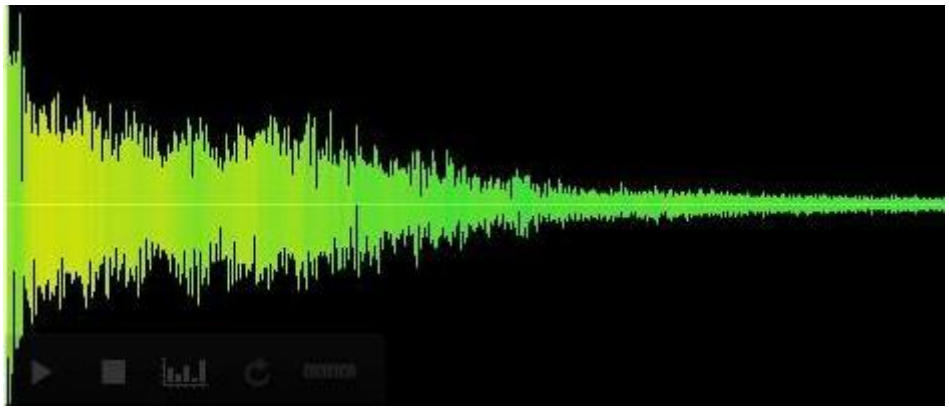
3.2.23 Cane che abbaia

Questo file audio si chiama “bigdogbarking-02.wav”, il suo ID è 24965, è in formato wav con una frequenza di campionamento di 48.00 Hz, 24 bit di quantizzazione ed è mono. Contiene il suono di un cane di grossa taglia che abbaia, per quattro volte.
Key_key e key_scale hanno dato i seguenti risultati: “C#” e “minor”. Nonostante il cane emetta quattro suoni circa alla stessa frequenza, gli algoritmi hanno restituito gli stessi risultati di quelli del file precedente, in cui l’uomo parlava per più di 20 secondi. Anche in questo caso le API avranno restituito il pitch che più si avvicina a quello del cane.
Key_strength ha restituito “0.53438353538500005”, una percentuale un po’ più alta di quella per il file precedente in quanto in questo caso le frequenze della voce del cane sembrano essere più definite rispetto a quelle della voce umana: il cane abbaia quattro volte mantenendo circa lo stesso pitch, mentre il pitch delle sillabe pronunciate dall’uomo varia maggiormente.
La risposta di tuning_frequency è stata “453.416503906”.

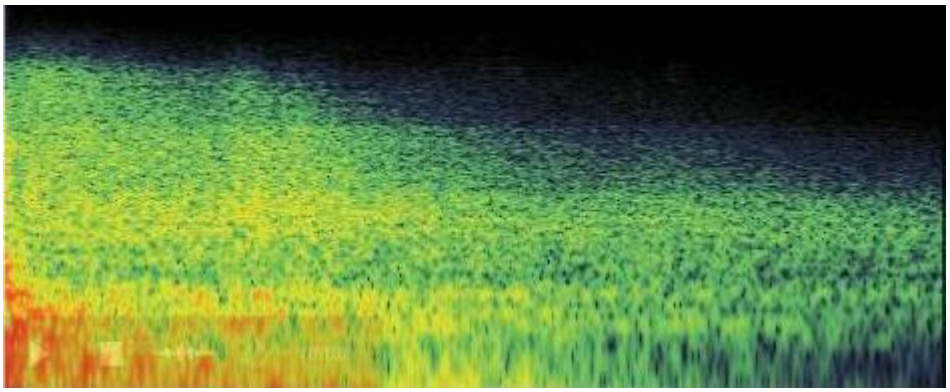
3.2.24 Esplosione

Gli ultimi cinque suoni contengono solo rumore, cioè suoni a banda larga o impulsivi, di cui non è possibile calcolare il pitch. Questo file si chiama “USAT BOMB.wav”, è in formato wav con una frequenza di campionamento di 44.100 Hz e 16 bit. Il suo ID è 35643.
Contiene il suono di un’esplosione di una bomba, con un’eco della stessa. Un’esplosione è un suono impulsivo, cioè un segnale con una grande energia concentrata in un intervallo di tempo infinitesimo. In questo piccolissimo intervallo di tempo il segnale contiene tutte le frequenze. Anche l’attacco del suono di un pianoforte è un segnale impulsivo, in questo caso l’impulso è dato dal contatto del martelletto sulla corda; dopo che la corda è stata percossa il martelletto si allontana, lasciando vibrare la corda alla sua frequenza. Quindi nel caso del pianoforte solamente l’attacco

contiene tutte le frequenze, mentre il corpo del suono ha una frequenza ben definita. Nel caso dell'esplosione, invece, tutto il suono è composto da rumore, non compare una frequenza definita.
L'impulso in questo caso è caratterizzato da un attacco molto intenso e da un lungo decadimento esponenziale, causato dalle riflessioni (echi). [11]



Forma d'onda dell'impulso



Spettrogramma dello stesso impulso: si noti il grande contenuto frequenziale dell'attacco, soprattutto sulle basse frequenze

I primi due algoritmi hanno dato "F" e "minor", key_strength ha restituito "0.65692150592800003", una percentuale decisamente alta, considerando che non è possibile parlare di tonalità, in quanto in un rumore, essendo presenti più o meno tutte le frequenze, sono presenti tutte le tonalità.
Tuning_frequency ha dato come risultato "434.44400024399999". Anche qui non ha senso parlare di frequenza di accordatura.

3.2.25 Pioggia e tuoni

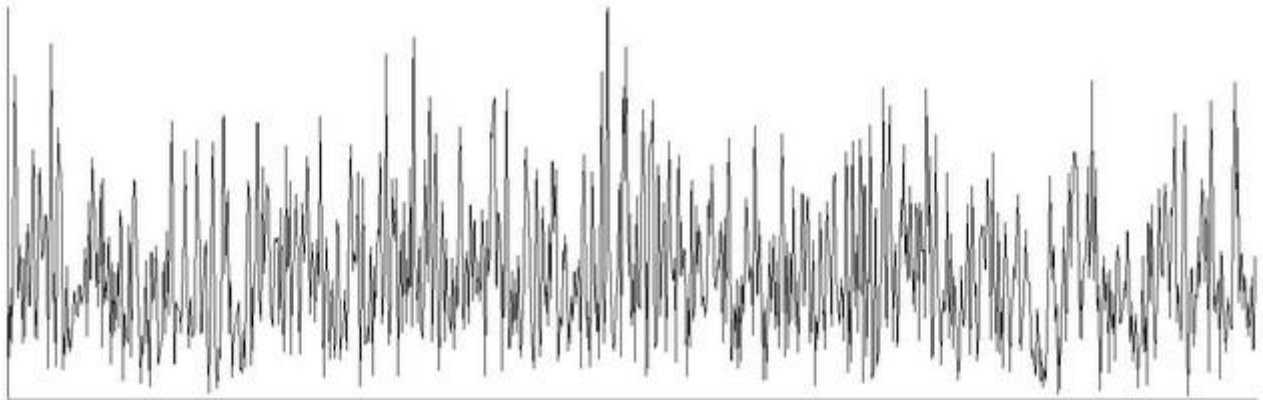
Questo file si chiama "rbh thunder storm.wav", ha ID 2523, è in formato wav con una frequenza di campionamento di 44.100 Hz e 16 bit di quantizzazione. Contiene il suono della pioggia,

intervallata ogni tanto dal rumore di un tuono. Lo scroscio della pioggia è un rumore bianco, a banda larga, mentre i tuoni sono rumori impulsivi. Il frammento audio dura 3 minuti e mezzo.
Key_key e key_scale hanno restituito "G" e "minor", mentre key_strength ha restituito "0.31094294786499999", che è la percentuale più bassa fin'ora trovata.
Tuning_frequency ha dato come risultato "431.44308471699998".

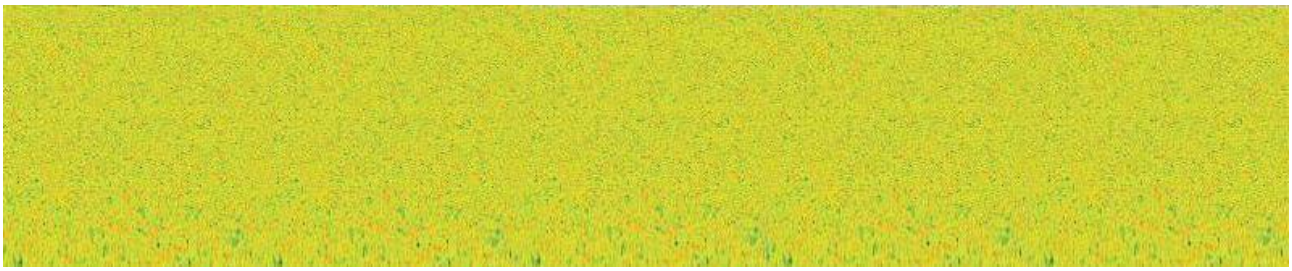
3.2.26 Rumore bianco

Verranno analizzati due file contenenti puro rumore bianco, uno in formato wav, l'altro in mp3, per confrontare i risultati degli algoritmi.
Il rumore bianco è un rumore di ampiezza costante su tutto lo spettro di frequenza, avendo componenti spettrali significative in tutte le frequenze audio. La caratteristica più importante del rumore è l'imprevedibilità. L'energia associata ad ogni ottava, però, non è costante, ad esempio l'energia compresa nella banda 20Hz-40Hz non sarà la stessa di quella della banda 5KHz-10KHz. Quest'ultima banda avrà un'energia associata maggiore pur essendo sempre la larghezza pari ad un'ottava in quanto il secondo intervallo di frequenze è molto più largo del primo: in altre parole contiene più frequenze, dunque complessivamente più energia. Il rumore bianco è detto anche fruscio, ed è causato dal calore insito in qualsiasi componente elettronico. [12]
Il file in formato wav si chiama "Static.wav", ha una frequenza di campionamento di 44.100 Hz, 32 bit di quantizzazione, è mono e ha come ID 17804. Contiene 10 secondi di rumore bianco, intervallati da frazioni di secondo di silenzio. Il file in formato mp3 si chiama "White Noise.mp3", ha un bitrate di 128 kbps, è stereo e il suo ID è 28024. Contiene 20 secondi di rumore bianco, senza silenzi. La qualità dell'audio è decisamente peggiore rispetto a quella del file wav, probabilmente perché è stato usato un formato di codifica di scarsa qualità.
Per quanto riguarda il primo file, key_key e key_strength hanno restituito "F#" "minor", mentre per quanto riguarda il secondo gli stessi algoritmi hanno dato come risultati "A#" "minor". In ogni caso non si può dire che le funzioni abbiano sbagliato, in quanto nel rumore sono presenti tutte le frequenze, quindi tutte le tonalità: avrebbero potuto restituire qualsiasi tonalità. Quello che è strano è il fatto che, nonostante entrambi i file contengano rumore bianco, i due algoritmi abbiano dato risultati diversi. Una possibile spiegazione di tutto questo è che probabilmente l'energia associata a quella data nota sia leggermente maggiore rispetto alle altre: in generale il rumore bianco è costante su tutto lo spettro, ma ciò non significa che in un certo intervallo di tempo l'energia sia costante su tutte le frequenze, poiché la caratteristica del rumore è l'imprevedibilità.

Key_strength ha dato come risultato “0.21050964295899999“ per il file wav, “0.50841164588900001” per quello mp3. La bassa percentuale del primo file è compatibile con il fatto che non si possa parlare di tonalità, mentre la percentuale del secondo è decisamente troppo elevata. Forse la scarsa qualità del formato di codifica ha introdotto nel rumore alcuni artefatti, come ad esempio dare maggiore energia ad una certa banda di frequenza, in questo caso intorno alla nota SI bemolle; per questo motivo la percentuale è relativamente alta.
Tuning_frequency ha dato “446.91595459000001” per il file wav e “439.49197387700002” per quello mp3. Non si capisce in che modo questa API abbia potuto restituire valori così vicini alla frequenza di riferimento standard.



Sonogramma del rumore bianco



Spettrogramma del rumore bianco

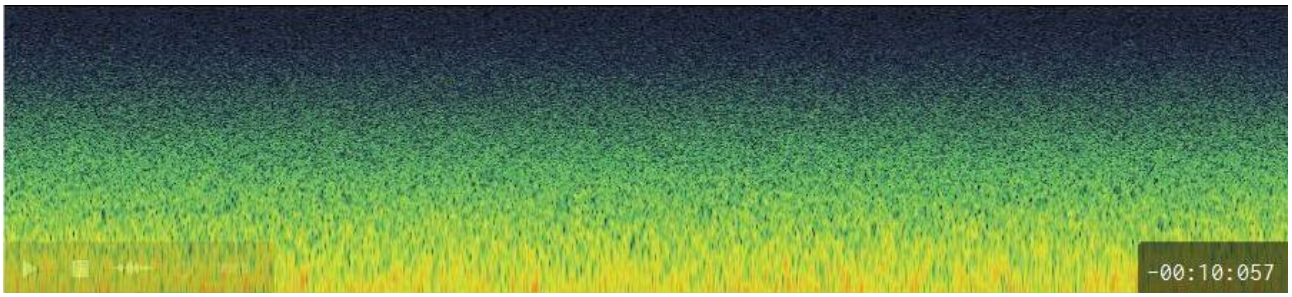
3.2.27 Rumore marrone

L’ultimo file che verrà preso in esame si chiama “brown noise.wav”, il suo ID è 3941 ed ha lo stesso formato del file wav precedente contenente rumore bianco. Contiene 10 secondi di rumore marrone.
A differenza del rumore bianco, il rumore marrone non ha l’energia costante su tutto lo spettro di frequenze, ma presenta una caduta di 6 dB per ogni raddoppio di frequenza: questo vuol dire che

l'energia associata ad un'ottava è 6 dB superiore rispetto a quella associata all'ottava successiva. E' un filtro passa basso del rumore bianco, le basse frequenze hanno più energia rispetto alle alte frequenze. Una via di mezzo tra rumore bianco e marrone è il rumore rosa, in cui c'è un decremento di 3 dB ad ogni raddoppio di ottava. [12]

Le API key_key e key_strength hanno dato come risultati "A#" "minor", la stessa tonalità data dal file mp3 precedente, probabilmente per pura casualità. Key_strength ha restituito "0.42335066199299998". Le considerazioni da fare per questi risultati sono le stesse fatte per i file precedenti contenenti rumore bianco.

Infine la risposta di tuning_frequency è stata "454.46530151399998".



Spettrogramma del rumore marrone: si noti la maggiore energia sulle basse frequenze rispetto al rumore bianco

3.3 Tabella riassuntiva

Di seguito verrà illustrata la tabella riassuntiva dei risultati delle API per ogni file audio testato. La quarta colonna indica se gli algoritmi key_key e key_scale hanno restituito il risultato corretto.

Descrizione	Key_key	Key_scale		Key_strength	Tuning_frequency
Brano malinconico in fa# maggiore	F#	major	ok	0.81457310915000003	442.548919678
Minuetto di Boccherini	A	major	ok	0.83609807491300003	438.98455810500002
Melodia di tromba	C	minor	ok	0.63064074516299995	437.71853637700002
Frammento della "Hammerklavier" di Beethoven	F#	minor	ok	0.55904060602200001	441.01779174799998
Melodia horror di pianoforte	G#	minor	ok	0.57768577337299998	434.44400024399999
Fuga in sol minore di Bach	G	minor	ok	0.62900769710500004	437.71853637700002
Loop in sol# minore	G#	major	no	0.77526402473400002	442.29336547899999

Descrizione	Key_key	Key_scale		Key_strength	Tuning_frequency
Kyrie in si minore di Bach con canto di uccello	B	minor	ok	0.64308291673700002	440
Progressione di pianoforte	C	major	ok	0.75807666778600002	442.03793335
Tre accordi di pianoforte	F	major	no	0.57492458820299996	441.01779174799998
Loop di chitarra	C	major	no	0.777964830399	432.940917969
Accordo arpeggiato	D	minor	ok	0.70319098234199995	440.25424194300001
DO min 7	D#	major	no	0.537560462952	439.238189697
Loop di chitarra elettrica	G	minor	no	0.61811745166800003	434.69500732400002
Coro	C	major	no	0.70685440301900004	432.44104003899997
Donna che canta	C#	minor	no	0.71664738655100002	440
Suono grave	G#	minor	no	0.61131960153599996	436.960693359
Tono puro a 1000 Hz	E	minor	no	0.57718229293800005	445.62707519499997
Toni puri a 500 Hz, 1, 2, 4, 8 kHz	E	minor	no	0.57718229293800005	445.62707519499997
Tono puro a 440 Hz	D	minor	no	0,577 approssimato	440,25
Tono puro corrispondente a fa#	B	minor	no	0,5822 approssimato	439,49
Tono puro a 13.500 Hz	E	minor	no	0,59 approssimato	435
Canto tibetano	A#	minor	no	0.40712758898700002	434.193115234
Suono di un oscillatore digitale	C#	minor	no	0.54376512765899998	445.62707519499997
Yeah	A	major	no	0.62344676256200005	444.341888428
Voce umana	C#	minor	no	0.433148771524	443.57257080099998
Cane che abbaia	C#	minor	no	0.53438353538500005	453.416503906
Esplosione	F	minor	no	0.65692150592800003	434.44400024399999
Pioggia e tuoni	G	minor	no	0.31094294786499999	431.44308471699998
Rumore bianco in wav	F#	minor	no	0.21050964295899999	446.91595459000001
Rumore bianco in mp3	A	minor	no	0.50841164588900001	439.49197387700002
Rumore marrone	A#	minor	no	0.42335066199299998	454.46530151399998

3.4 Conclusioni

Di questi quattro algoritmi analizzati solamente i primi 2, key_key e key_scale, sembrano funzionare in generale in modo corretto, o comunque in linea con i risultati attesi. Le due API danno il meglio di sé in presenza di file audio contenenti melodie: con le melodie di pianoforte, o di tromba, o di violino, in cui la tonalità è molto ben definita, i risultati restituiti sono stati corretti.
In presenza di soli accordi, invece, non sempre le funzioni restituiscono risultati corretti. In generale le funzioni tendono a dare come tonica l'ultima nota del brano musicale, quindi se il file contiene accordi in mi minore, ma il loop termina in do maggiore, le API danno come risposta DO maggiore. Ovviamente non sempre viene trovata la tonalità giusta, come nel caso della melodia cantata dal coro.
Invece non si capisce per quale motivo, in presenza di toni puri, gli algoritmi diano un risultato decisamente errato. Probabilmente il calcolo della tonalità viene fatto tenendo anche conto degli armonici dei suoni: i toni puri, non avendo armonici, non vengono riconosciuti correttamente.
Per quanto riguarda file contenenti rumori o suoni non musicali i risultati sono ovviamente errati, gli algoritmi avrebbero potuto restituire qualsiasi tonalità. E' curioso il fatto che in questi casi key_scale restituisca sempre "minor".
Le API key_strength e tuning_frequency, a differenza delle altre due, non sembrano funzionare in modo corretto. Anche se la tonalità del brano è perfettamente chiara, la percentuale di key_strength non è mai eccessivamente elevata. Inoltre le percentuali restituite non superano mai l'85%, anche nel caso di melodie, e non sono mai troppo basse, anche nel caso di rumori. La percentuale più bassa trovata è stata 21%, calcolata sul file wav contenente rumore bianco, mentre sul file mp3 la funzione ha calcolato circa 50%.
Tuning_frequency stranamente restituisce risultati che sono sempre molto vicini a 440 Hz, anche se gli algoritmi per il calcolo della tonalità hanno dato risultati errati. Questo è molto strano in quanto, prendendo come esempio il file contenente il canto corale, se la tonalità reale è SOL maggiore, ma le API restituiscono DO maggiore, tuning_frequency dovrebbe dare una frequenza di accordatura completamente diversa dai 440 Hz.
Un possibile sviluppo futuro potrebbe essere quello di capire il motivo della mancata risposta delle API di descrizione tonale non funzionanti, in modo tale da farle lavorare correttamente. Inoltre occorre anche migliorare il funzionamento delle quattro API analizzate in questo elaborato, ad esempio, per quanto riguarda key_key, considerando sia l'accordo iniziale, sia quello finale, o calcolare quale sia l'accordo più frequente all'interno del file. Queste funzioni sono molto

interessanti e potrebbero essere utili per lo sviluppo di eventuali software che aiutino musicisti e informatici ad analizzare file audio musicali.

4. Bibliografia

[1] Mario Malcangi, *Informatica Applicata al Suono per la Comunicazione Musicale – Musical Digital Audio: Teoria e pratica*, Maggioli Editore, Milano, 2006.

[2] Sergio Cingolani, Renato Spagnolo, *Acustica musicale e architettonica*, Città Studi Edizioni, 2008

[3] Luigi Rossi, *Teoria musicale*, Edizioni Carrara, 1977

[4] Renato Dionisi, *Lezioni di armonia complementare*, Edizioni Curci, Milano 1982

[5] Freesound API Analysis Descriptor Documentation:
http://www.freesound.org/docs/api/analysis_docs.html

[6] Antonio Mancuso, *Scale musicali*, dispensa del corso di Psicofisiologia della Percezione Musicale, Università degli Studi di Milano, 2010

[7] John Sharp, *Microsoft Visual C# 2008 Step by Step*, Microsoft Press Editon, 2008

[8] MSDN C# Reference: *http://msdn.microsoft.com/en-us/library/618ayhy6.aspx*

[9] Renato Dionisi, *Appunti di analisi formale*, Edizioni Curci, Milano

[10] Le garzantine, *Enciclopedia della musica*, Garzanti Editore, Milano 1996

[11] Alberto Bertoni, Paola Campadelli, Giuliano Grossi, *Introduzione all'elaborazione dei segnali,* dispensa del corso di Elaborazione Numerica dei Segnali, Università degli Studi di Milano, 2008

[12] Simone Coen, *Rumore,* dispensa del corso di Tecnologie Informatiche per il Restauro dell'Informazione Musicale, Università degli Studi di Milano, 2011