Rilevazione in tempo reale delle note di una chitarra elettrica

Pier Angelo Vendrame

22 Giugno 2017

Indice

1 Introduzione

1.1 Obiettivi

L'obiettivo di questo programma è il riconoscimento in tempo reale delle note che sono state suonate su una chitarra elettrica e la loro visualizzazione su una finestra.

Lo strumento musicale richiesto per questo progetto non deve soddisfare particolari requisiti, come avere dei pickup MIDI o esafonici e il suo collegamento deve poter avvenire mediante una normale scheda audio dotata di ingresso per microfoni.

1.2 Motivazioni

La scelta di questo progetto nasce dalla volontà di mettere assieme due mie passioni: la chitarra e lo sviluppo di software.

Da molto tempo avevo quest'idea, ma solo gli studi universitari mi hanno permesso di ottenere gli strumenti necessari a implementarla, come, ad esempio, delle basi per l'analisi dei segnali.

2 Architettura e progettazione dell'applicazione

L'applicazione è stata progettata per essere il più modulare possibile.

In particolare, si può dividere in tre parti:

- acquisizione di dati in formato PCM dalla scheda audio;
- riconoscimento della frequenza nel segnale acquisito;
- GUI per la visualizzazione della nota suonata.

2.1 Condizioni per l'acquisizione

Per l'utilizzo dell'applicazione si assume che la chitarra sia completamente isolata da altri strumenti, il che è sempre vero se viene collegata direttamente.

Inoltre devono essere suonate note monofoniche, ovvero devono essere suonata una dopo l'altra e non devono essere sovrapposte nel tempo.

Il programma non è in grado di riconoscere alcun tipo di accordo, perché si basa su un algoritmo di riconoscimento della frequenza di un singolo tono che prevale su tutti gli altri.

Per favorire le prestazioni, l'acquisizione avviene direttamente in virgola mobile a singola precisione, con ampiezze comprese tra -1.0 e 1.0. Poiché i conti che vengono effettuati necessitano di un codominio con ampiezza elevata, usando per esempio valori interi a 16 bit senza trasformali in decimali normalizzati, le operazioni intermedie sui dati andrebbero spesso in overflow.

2.2 Riconoscimento della frequenza tramite autocorrelazione

Esistono diversi approcci per il riconoscimento della frequenza di un segnale.

La prima distinzione si basa sul dominio usato:

- *dominio del tempo*: autocorrelazione, YIN, etc...
- *dominio della frequenza*: FFT, etc...

Secondo quanto riportato in letteratura[2], con la chitarra l'autocorrelazione ha prestazioni, in termine di errori commessi, migliori.

Per definizione l'autocorrelazione per segnali continui è data da

$$R_{ff}(t) = \int_{-\infty}^{+\infty} f(t)\bar{f}(t-\tau)d\tau \tag{1}$$

dove $\bar{f}(t)$ rappresenta il complesso coniugato di $f(t)$.

È uno strumento matematico che consente di valutare quanto un segnale è simile a una copia di se stesso traslata nel tempo di un tempo t, detto anche ritardo.

Il periodo T del segnale può dunque essere trovato in questo modo:

$$T = \underset{t \in \mathbb{R}_{>0}}{\arg\max}\, R_{ff}(t) \tag{2}$$

Dal punto di vista del calcolo numerico, è però più comodo normalizzare l'autocorrelazione, ovvero farle assumere valori compresi tra -1.0 e 1.0 indipendenti sia dall'ampiezza del segnale, che dal valore t del ritardo.

In questo modo trovare per il periodo si procede con il calcolo dell'autocorrelazione per un intervallo di possibili valori, e il periodo è il valore per cui l'autocorrelazione è più vicina a 1.

Si noti che usando questo metodo in un computer, l'integrale diventa una somma e l'unità di misura del periodo non sono i secondi, ma il numero di campioni.

Per trovare la frequenza f in Hz, bisogna sfruttare la seguente relazione con la frequenza di campionamento f_S:

$$f = \frac{f_S}{T} \tag{3}$$

2.3 Semitoni

Come unità di misura per far interagire diverse parti del software, è stata scelta il *semitono*.

Definizione 1 *Un semitono è il più piccolo intervallo che separa due note.*

Rispetto alla frequenza ha il vantaggio di essere una scala intera e lineare, anziché decimale ed esponenziale.

Inoltre c'è un'importante identità tra semitono e tasto della chitarra: tra un tasto e il successivo infatti la distanza è proprio di un semitono.

Trasformando dunque la frequenza in semitono, e sapendo qual è l'accordatura dello strumento, si possono trovare tutte le posizioni in cui la nota acquisita può essere suonata tramite somme o sottrazioni tra interi.

Una grande differenza, però, tra frequenze e semitoni, è che le prime sono delle misure assolute, ovvero c'è individuano sempre solo una nota, invece i seocndi sono una grandezza relativa, ovvero indicano l'intervallo tra due note.

Come convenzione, per il programma, si è deciso di prendere sempre i semitoni relativi al *La0*, ovvero la prima nota del pianoforte, avente frequenza $f_{A0} = 27,5$ Hz.

In generale, tra due note, una di frequenza f, l'altra di frequenza f_R e separate da Δn semitoni, vale la seguente relazione:

$$f = 2^{\frac{\Delta n}{12}} f_R \tag{4}$$

3 Sviluppo e implementazione

Per l'implementazione sono state usate ove possibile le librerie standard, negli altri casi invece si è carcato di usare librerie cross-platform e con licenza libera.

3.1 Acquisizione audio

Per l'acquisizione audio è stata usata LibSoundIO (`http://libsound.io/`), una libreria rilasciata sotto licenza MIT e disponibile per Linux, Windows e Mac OS.

L'obiettivo principale di questa libreria è offire una API unificata ma che aggiunga il meno possibile alle varie interfacce native dei suoi backend.

La funzione che registra l'audio è stata implementata con l'idea che ad essa sarebbe stato dedicato un thread: fondamentalmente si compone infatti di un ciclo che continuamente controlla se è possibile riempire un buffer con i dati acquisiti e poi li elabora.

Il thread però non viene gestito dal modulo dell'audio, ma da un modulo di controllo che è confluito nella GUI. Ciò consente di trascurare problemi di sincronizzazione:

- per segnalare la fine della registrazione alla funzione di registrazione basta semplicemente ritornare;
- per l'interruzione del ciclo di acquisizione-elaborazione è stato previsto un flag che viene passato come puntatore ad una costante: non essendo modificabile non necessita di un mutex;
- per la visualizzazione dei risultati il problema della sincronizzazione viene delegato al toolkit grafico.

3.2 Autocorrelazione

Per l'autocorrelazione si è deciso di partire da un'implementazione già esistente, anch'essa rilasciata sotto licenza MIT.

Essendo molto semplice, consente di comprenderne completamente il funzionamento.

Vi sono state apportate però alcune modifiche:

- rimozione del `vector` di *C++* in favore di un buffer su heap allocato tramite `malloc`;
- divisione del corpo della funzione in più funzioni separate;
- spostamento di alcune operazioni per aumentarne le prestazioni;
- controlli sui risultati, sia parziali che finali, per prevenire crash dell'intero programma.

3.2.1 Crash legati a segnali particolari

In diverse fasi di test sono emersi alcuni crash legati ad accessi non validi alla memoria.

Vanno ricondotti al fatto che l'autocorrelazione viene calcolata a tempo discreto, ma, a causa della proporzionalità della frequenza a $1/T$, a volte i risultati

avrebbero errori troppo grandi. Per ovviare al problema viene effettuata un'interpolazione basata sul valore massimo di autocorrelazione, sul valore precedente e sul successivo:

$$T = T_0 + \frac{R_{ff}(T_0+1) - R_{ff}(T_0-1)}{2R_{ff}(T_0) - R_{ff}(T_0+1) - R_{ff}(T_0-1)} \tag{5}$$

dove T è il valore che sarà considerato il periodo, T_0 è il valore che rende massimizza l'autocorrelazione R_{ff}.

Se $R_{ff}(T_0)$, $R_{ff}(T_0+1)$ e $R_{ff}(T_0-1)$ sono troppo vicini, il contributo dell'interpolazione può diventare maggiore di quello di T_0 e si potrebbe addirittura ottenere un periodo negativo.

Questo può condurre a crash perché nell'algoritmo di ricerca delle ottave, il periodo trovato in questo modo viene ritrasformato in un numero intero e poi usato come indice di un array.

La soluzione adottata è quella di ignorare l'interpolazione quando il suo valore diventa troppo grande in proporzione al valore di T_0.

Si ha un altro caso particolare di crash molto simile quando il segnale in ingresso presenta dei picchi impulsivi, causati perlopiù dai componenti elettromeccanici della chitarra (i potenziometri di regolazione del volume, l'interruttore di selezione dei pickup, etc...).

Certe schede audio per eventi del genere usano il valore `NaN`, previsto dallo standard *IEEE 754*.

Si è deciso dunque di ritenere corrotti eventuali segnali aventi un periodo di valore `NaN` e di segnalare un errore al chiamante della funzione `estimatePeriod`.

3.2.2 Il problema delle ottave

L'autocorrelazione è soggetta al problema delle ottave, ovvero a volte rileva un periodo che non è quello fondamentale, ma un suo sottomultiplo.

L'implementazione dell'autocorrelezione include anche un algoritmo per la ricerca della frequenza fondamentale, che si può riassumere così:

1. Suppongo che il periodo non sia T_0 ma sia kT_0.
2. Le ampiezze delle armoniche che si avrebbero con questo nuovo periodo superano tutte una certa soglia?
   - *Sì*: il periodo corretto è kT_0
   - *No*: ripeto da 1 usando $k' = k - 1$

3.2.3 Rilevazione della corda suonata

L'autocorrelazione permetterebbe di trovare anche in che punto è stata suonata una corda[1].

Siano L_s la distanza tra il ponte e il tasto premuto, f la frequenza della nota suonata, T_{min} l'indice corrispondente al minimo valore dell'autocorrelazione, f_S la frequenza di campionamento, allora si può calcolare il punto in cui è stata suonata la nota in termini della sua distanza dal ponte d_p:

$$d_p = LT_{min}\frac{f}{f_S} \tag{6}$$

Assumento che le note vengano sempre suonate in un certo range di distanze dal ponte, e visto che tutti i dati sono noti, si può calcolare per d_p per ogni corda e vedere quale rientra meglio nella range di valori atteso.

Questo metodo però non è stato implementato perché dal punto di vista pratico le ipotesi risultano estremamente restrittive: spesso più di una corda rientrava nel range ammesso e favorire quella centrale in molti casi non era corretto.

3.3 Trasformazione da frequenza a nota

Come metriche per determinare che nota è stata suonata sono state prese: frequenza, ampiezza, valore dell'autocorrelazione, effettiva possibilità di suonare quella nota.

3.3.1 Frequenza e ampiezza

La frequenza è il fattore determinante per la rilevazione di una nota: un cambio di frequenza implica il più delle volte che l'utente ha suonato una nota nella chitarra diversa dall'eventuale nota suonata in precedenza.

Però da sola non è sufficiente a determinare quando la stessa nota viene risuonata. Per farlo bisogna usare anche le informazioni sull'ampiezza: assieme alla ripetizione inevitabilmente si deve verificare un calo brusco di ampiezza del segnale seguito da una immediata crescita molto elevata.

Al contrario, il non verificarsi di un tale andamento dell'ampiezza serve a evitare le rilevazioni scorrette: se vengono rilevate delle armoniche della nota precedente, vengono ignorate a meno che non si riconosca questo pattern prima decrescente e poi crescente.

Per analizzare il valore di ampiezza si sfrutta la natura periodica del segnale e dunque si vede come per ogni periodo cambia il picco, anziché usare una media mobile. Questa decisione è dettata da osservazioni nella fase di test.

3.3.2 Valore dell'autocorrelazione

Per come è stata normalizzata, l'autocorrelazione può assumere solo valori compresi tra -1 e 1: questo permette di trattare il valore dell'autocorrelazione come "qualità della periodicità": il massimo dell'autocorrelazione di un segnale perfettamente periodico è 1, invece meno il segnale si ripete uguale a se stesso, più la sua autocorrelazione si allontana da 1.

Quindi segnali con una qualità di periodicità bassa saranno scartati come rumore. Come valore di soglia, sperimentalmente si è deciso di prendere $0,85$, ma si è visto che spesso le note hanno valori superiori a $0,99$.

3.3.3 Possibilità di suonare una nota

Se una nota non può essere suonata in nessuna posizione, è altamente improbabile che sia stata suonata, quindi viene considerata rumore.

Esistono due casi in cui questo non è vero, ma ignorarli è comunque una decisione ragionevole:

- *bending* [1] sull'ultimo tasto della prima corda [2]: di fatto questi bending consentono di uscire dal range delle frequenze normalmente suonabili, tuttavia non si potrebbe rappresentare una nota così prodotta sull'interfaccia grafica;
- armonici molto acuti: anche questi non sarebbero rappresentabili, poiché fuori dalla tastiera.

3.3.4 Timeout di una nota

Normalmente il programma è in grado di riconoscere quando l'ampiezza del segnale cala sotto una certa soglia minimi e di identificare quindi la fine di una nota.

Ci sono dei casi, però, in cui il rumore lo impedisce: vengono quindi contati quanti campioni di seguito vengono scartati perché considerati rumore e superata una certa soglia, il programma decide che la nota non è più valida e la considera finita.

È stato deciso di impostare questa soglia pari alla frequenza di campionamento, ovvero dopo un secondo che viene solo rilevato rumore lo stato della nota correntemente suonata viene resettato.

3.4 Interfaccia grafica

L'interfaccia grafica è stata realizzata usando la libreria GTK.

3.4.1 Manico della tastiera

Il manico della tastiera è stato disegnato in grafica vettoriale utilizzando Inkscape e viene renderizzato con la libreria RSVG.

Internamente il file è diviso in due livelli, uno con il manico della chitarra e l'altro contenente un pallino per ogni tasto su ogni possibile corda. Ogni pallino ha come id `dot_corda_tasto`.

Con la funzione `rsvg_handle_render_cairo_sub` il primo livello viene renderizzato sempre, e poi si renderizzano solo i pallini necessari.

In generale il disegno avviene nel thread principale tramite un sistema di *segnali* e viene eseguito solo in caso di determinati eventi, per esempio quando la finestra viene ridimensionata, oppure se viene effettuata la richiesta manuale.

Per aggiornare lo stato del manico il software prevede due funzioni (`guiHighlightFrets` e `guiResetHighlights`) che prima aggiornano la variabile globale `gFrets` del file `gui.c` e poi richiedono a GTK di ridisegnare il manico.

Queste funzioni vengono normalmente chiamate dal thread secondario dell'acquisizione dei dati, ma non ci sono problemi di sincronizzazione: mentre la registrazione è in corso, `gFrets` viene modificata solo dal quel thread, invece dal thread della GUI può essere modificata solo dalla funzione `stopRecording`, che però lo fa solo dopo il *join* del thread dell'acquisizione.

La richiesta di ridisegno è gestita tramite meccanismi thread safe di GTK.

[1] Il *bending* è una tecnica che permette di alzare la frequenza di una nota inclinando la corda.

[2] Le corde della chitarra vengono numerate da 1 a 6. La corda 1 è quella più sottile che produce suoni più acuti, la corda 6 invece produce i suoni più gravi.

3.4.2 Finestra dalle impostazioni

La finestra delle impostazioni permette di scegliere il backend da usare con LibSoundIO e il dispositivo per registrare.

L'unico accorgimento che ha richiesto è stata la creazione di un'altra istanza di SoundIO per la popolazione dei vari menù di scelta, in modo da non interrompere la registrazione e doverla ricominciare solo per mostrare i dispositivi disponibili.

4 Valutazione e collaudo

A fasi di sviluppo si sono sempre alternate fasi di collaudo.

Per l'algoritmo di stima del periodo e per le varie utilità per fare operazioni su frequenze e semitoni i test sono stati svolti automaticamente con il framework di unit testing *Check*, per altre parti invece sono stati effettuati collaudi manuali.

In ogni caso il collaudo ha permesso di affinare i diversi parametri scelti inizilamente.

Inoltre il collaudo con segnali reali ha rivelato alcuni bug che durante la stesura del codice non sarebbero mai stati individuati, come l'effetto dei componenti elettromeccanici sul segnale.

4.1 Volume di acquisizione

Il più grande problema per il corretto funzionamento dell'applicazione è l'impostazione del volume di acquisizione.

Guadagni troppo elevati portano più velocemente alla saturazione e introducono una notevole quantità di rumore. Come risultato si hanno spesso rilevazioni errate e frequenti cambi di nota che non sono realmente accaduti.

Viceversa, un guadagno troppo basso, non permette di distinguere le note effettivamente suonate dal rumore e quindi spesso non avviene la rilevazione. Ciò accade in particolar modo sulle note più acute.

4.2 Effetti dei diversi pickup

I test sono stati effettuati con due chitarre diverse: una Fender Stratocaster e una Squier Telecaster. La prima ha dei pickup migliori e si è notato anche nella qualità della periodicità indicata da `estimatePeriod`: non scendeva quasi mai sotto $0,99$, a differenza della seconda, in cui spesso scendeva anche a $0,97$.

Probabilmente anche le chitarre acustiche, adeguatamente microfonate, funzionerebbero e, avendo tenuto molte caratteristiche dello strumento parametriche, con poche modifiche al codice, anche i bassi elettrici.

4.3 Cross compatibilità e hardware di acquisizione

L'applicazione è stata sviluppata secondo un'ottica cross platform, tuttavia è stata testata solo sulla distribuzione Debian su macchine X86.

È stata testata però su diversi computer. Da ciò è emerso che la qualità dell'alimentatore influisce parechio sul rumore del segnale.

Come scheda audio è stata usata sempre quella integrata del computer. Per mancanza di hardware non sono stati fatti test con schede audio esterne, che normalmente sono consigliate per attaccare gli strumenti musicali direttamente ai computer.

5 Conclusioni e lavoro futuro

Il risultato ottenuto finora, pur non avendo molte funzoinalità, è abbastanza soddisfacente, perché è abbastanza preciso e ha dei bassi tempi di latenza.

Come prossimo sviluppo sarebbe interessante salvare le note suonate come tablatura.

Un altro sviluppo che però potrebbe anche essere intrapreso è quello di trasformarlo in uno strumento MIDI sfruttanto le varie API dei diversi sistemi operativi. Ciò permetterebbe di far interfacciare la chitarra direttamente con le *DAW* (*Digital Audio Workstation*).

Anche la GUI potrebbe essere estesa, in modo da consentire per esempio di aggiungere la possibilità di scegliere un'accordatura diversa da quella standard, cosa che per il momento non è stata fatta per non aggiungere ulteriore complessità, pur essendo già stata prevista nelle parti di rilevazione delle note.

Riferimenti bibliografici

[1] H. Penttinen, J. Siiskonen, and V. Valimaki. Acoustic guitar plucking point estimation in real time, March 2005.

[2] C. P. Singh and T. K. Kumar. Efficient pitch detection algorithms for pitched musical instrument sounds: A comparative performance evaluation, September 2014.