
# UNIVERSITÀ DEGLI STUDI DEL SANNIO

*FACOLTÀ DI INGEGNERIA INFORMATICA*

Anno Accademico 2007 / 08

CORSO DI:

# ELABORAZIONE DEI SEGNALI E DELLE INFORMAZIONI DI MISURA

Prof. P. Daponte

Ing. L. De Vito

# Riconoscimento di Frequenze di Note Musicali

**Gruppo di Lavoro:**

| | |
|---|---|
| Mario Calì | 195/001418 |
| Marco Gallucci | 195/001241 |
| Roberto De Falco | 195/001462 |
| Eugenio Macchia | 195/001134 |

# SOMMARIO

# INTRODUZIONE

Lo scopo del progetto è stato quello di riconoscere, a partire da un segnale audio musicale, la frequenza delle note riprodotte.

L a scheda DSP utilizzata è la seguente:

| SCHEDA DSP | |
|---|---|
| MARCA | Texas Instrument |
| TIPO | TMS320C6711 DSK |

Il software utilizzato è Matlab e Code Composer Studio v. 2 della Texas instruments.

Lo sviluppo del progetto è stato articolato in diverse fasi:

- Analisi del problema del riconoscimento di frequenze di note musicali e individuazione di una possibile soluzione.
- Implementazione in Matlab dell'algoritmo risolutivo e simulazione dello stesso per verificarne l'effettivo funzionamento.
- Traduzione del codice Matlab in codice C e simulazione in Code Composer Studio v. 2 a partire da segnali audio musicali noti, direttamente inseriti nel codice sorgente.
- Implementazione della gestione degli interrupt e testing su scheda DSP reale utilizzando prima un microfono come ingresso e poi tramite collegamento diretto fra sorgente audio e scheda DSP.

# ANALISI DEL PROBLEMA E ALGORITMO RISOLUTIVO MATLAB

## NOTE MUSICALI E FREQUENZA

Con l'espressione Nota Musicale si intendono fondamentalmente due cose: il segno con cui si rappresentano i suoni usati nella musica e il singolo suono stesso, generato da uno strumento o dalla voce umana.

Due note, di cui una ha frequenza doppia rispetto all'altra, sembrano molto simili e di conseguenza sono comunemente chiamate con lo stesso nome. L'intervallo determinato da queste note è detto ottava. Pertanto, per identificare una nota in modo univoco si deve indicare anche l'ottava di appartenenza.

In linea di principio, la musica può essere composta da note di frequenza arbitraria. Per ragioni storiche e psicoacustiche, si è consolidato l'uso di dodici note per ottava, specialmente nella musica occidentale. Queste note a frequenza fissa sono in relazione matematica fra loro, e sono calcolate a partire da una nota fondamentale la cui frequenza è stabilita per convenzione. Recentemente si è stabilito che il La4, corrisponda a una frequenza acustica di 440 Hz.

Ogni nota è separata dal La4 da un numero intero di semitoni. E ogni 12 semitoni si ha un raddoppio di frequenza. Quindi, la frequenza di una nota che dista n semitoni dalla fondamentale è data dalla formula:

$$\text{frequenza} = 440 \times 2^{n/12}\ \text{Hz}$$

Per esempio, troviamo la frequenza del do immediatamente sopra al la4 (do5). Per ottenere il do5 si devono aggiungere tre semitoni:

la — 1 → la♯ — 2 → si — 3 → do

$$f = 440 \times 2^{3/12} \approx 523{,}25\ \text{Hz}$$

Il segno di n è importante; per esempio, il fa immediatamente sotto il la4 è il fa4. Si devono quindi sottrarre 4 semitoni:

La —1→ la♭ —2 → sol —3 → sol♭ — 4→ fa.

Quindi:

$$f = 440 \times 2^{-4/12} \approx 349{,}23\ \text{Hz}$$

Infine, si vede che ogni dodici semitoni si ha una frequenza doppia, ovvero un intervallo di un ottava.

## STRUMENTI MATEMATICI UTILIZZATI

Per operare l'analisi in frequenza occorre partire dalla forma d'onda del segnale acquisito, molto probabilmente di tipo elettrico perché prelevata da un microfono, e operare una conversione.

Esiste una seconda metodologia d'analisi del suono chiamata "analisi di Fourier"; lo spettro del segnale si costruisce attraverso un'operazione matematica detta Trasformata discreta di Fourier (Discrete Fourier Transform – DFT). Un'analisi di questo tipo è possibile con l'ipotesi di stazionarietà e di periodicità del segnale, quindi non può essere utilizzata per segnali che variano in modo imprevedibile come la voce umana.

La base matematica su cui si appoggia l'analisi di Fourier è il TEOREMA DI FOURIER: "un suono complesso, ma periodico, è sempre rappresentabile dalla sovrapposizione di un infinito numero d'onde sinusoidali ciascuna dotata d'opportuna ampiezza e fase."

La tecnica utilizzata in queste analisi è la FFT (fast fourier transform: trasformata veloce di Fourier) che è la versione ottimizzata per massimizzare la velocità di calcolo su computer della DFT (discrete fourier transform: trasformata discreta di Fourier).

Con la FFT possiamo analizzare lo spettro di un suono e vedere le sue componenti, passando dalla visione della forma d'onda, con il tempo sull'asse X, alla visione in frequenza (frequenze sull'asse X). Proprio per questo è corretto dire che, con la FFT, si passa dal dominio del tempo a quello della frequenza.
Il principale parametro di controllo della FFT è la sua risoluzione che è determinata dal numero di campioni (detti "punti") su cui viene effettuata. Naturalmente un maggior numero di punti significa un'analisi più accurata, ma anche un maggior tempo di calcolo.

## ALGORITMO RISOLUTIVO MATLAB

La funzione Matlab risolutiva, parte da un segnale già acquisito, con i suoi campioni posti in una matrice denominata "data". Copiata la colonna di interesse in un vettore ed effettuata la trasposizione, la funzione procede all'identificazione del punto di inizio del segnale, stabilendo un valore soglia d'inizio. Successivamente, viene effettuata la FFT su un numero di campioni pari a 1024 (da una serie di prove sperimentali è emerso che 1024 è un numero ottimale per ottenere valori delle frequenze molto accurati), a partire dal valore d'inizio determinato. Trovata la frequenza pari al massimo della FFT (componente principale), si ottiene il valore della frequenza utilizzando una semplice formula (in Matlab la frequenza è decrementata di 1 perché gli array partono da indice 1 e non 0). Se si fornisce alla funzione un'array di frequenze note, è possibile determinare la frequenza più vicina attraverso un semplice ciclo for e di conseguenza stabilire il nome della nota associata alla frequenza determinata.

Le varie simulazioni in Matlab hanno mostrato che è possibile ottenere un riconoscimento molto accurato utilizzando un numero di campioni uguale a 1024. Se si scende al di sotto di questa soglia, si verifica una percentuale d'errore che abbiamo considerato inaccettabile. Grazie quindi all'accurata analisi, la funzione matlab ha restituito un corretto risultato in praticamente tutte le simulazioni effettuate.

```
function nota = trovaNota(data)
 %array delle frequenze delle note da riconoscere
freqNote = [262, 294, 330, 349, 392, 440, 494];

max = 0;
indMax = 0;
ind = 1;

%numero di campioni
camp = 1024;

%frequenza di campionamento
campFreq = 44100;

x = data(:,1);

%stabilisce il punto di inizio del segnale
while(x(ind)<0.2)
    ind = ind+1;
end

%esegue la fft del segnale
x = data(ind:(ind+(camp-1)),1);
x = abs(fft(x));

plot(x);

%trova la frequenza corrispondente al massimo della fft
for i=1:1:camp
    if x(i) > max
        max = x(i);
        indMax = i;
    end
end

max
indMax

%ricava il valore della frequenza del segnale
freq =(indMax-1)*campFreq/camp;

%stabilisce la nota musicale basandosi sulla distanza minima fra il valore di
%frequenza ricavato e quelli delle note conosciute
diff = abs(freq - freqNote(1));
index = 1;
for i=2:1:7
    diff2 = abs(freq - freqNote(i));
    if diff2 < diff
        diff = diff2;
        index = index + 1;
    end
end

nota = freqNote(index);
```

## TRADUZIONE IN LINGUAGGIO C E SIMULAZIONE IN CODE COMPOSER STUDIO

La fase successiva alla scrittura ed al testing del programma nel linguaggio Matlab, è la fase di traduzione nel linguaggio C per la successiva esecuzione nell'ambiente di sviluppo Code Composer Studio v.2 della Texas Instruments.

Poiché le prime operazioni effettuate in Code Composer Studio sono state in simulazione, abbiamo inserito i dati relativi ai campioni in un array direttamente all'interno del codice sergente C, prelevando i valori da Matlab.

La traduzione della logica ciclica (ciclo "for") e condizionale (costrutto "if") è risultata ovviamente molto semplice, vista la somiglianza fra i due linguaggi. Il vero problema che si è verificato è stato trovare una funzione C che effettuasse il calcolo della FFT. Dopo aver studiato affondo il manuale *TMS320C67x DSP Library Programmer's Reference Guide* la scelta è ricaduta sulla funzione di libreria DSPF_sp_cfftr2_dit.

La suddetta funzione lavora su numeri complessi, di conseguenza è stato introdotto un ciclo "for" per la creazione di un vettore formato da coppie parte reale – parte immaginaria, con la parte reale uguale al campione inserito, e la parte immaginaria uguale a 0, che è stato poi passato come parametro alla funzione. Dal manuale è inoltre emerso che la funzione che abbiamo scelto ha bisogno di altre due procedure per funzionare correttamente, la prima "gen_w_r2" che genera i fattori di twiddle necessari per un efficiente calcolo della FFT e "bit_rev" che effettua il bit-reverse.

Molto probabilmente l'algoritmo FFT utilizzato dalla funzione di libreria è l'algoritmo di Cooley-Tukey o una sua variante. Questo algoritmo si basa sul principio di divide et impera, e spezza ricorsivamente una DFT di qualsiasi dimensione N con N numero composto tale che N=N1N2 in DFT più piccole di dimensioni N1 e N2, insieme a O(n) moltiplicazioni per l'unità immaginaria, detti fattori twiddle. L'uso più conosciuto dell'algoritmo di Cooley-Tukey è di dividere e trasformare in due pezzi di n/2 ad ogni passo, ed è quindi ottimizzato solo per dimensioni che siano potenze di due, ma in generale può essere utilizzata qualsiasi fattorizzazione.

Il codice, riportato alla pagina seguente, è stato testato con diversi array di campioni, rappresentanti note diverse. I risultati ottenuti sono stati più che soddisfacenti, sia in termini di velocità del calcolo che in termini di affidabilità del risultato, per cui siamo passati alla prova sperimentale su scheda DSP reale.

## CODICE IN LIGUAGGIO C

```
#include <DSPF_sp_cfftr2_dit.h>#include <stdio.h>#include <stdlib.h>#include <math.h>#define N 1024
void bit_rev(float* x, int n);
void gen_w_r2(float* w, int n);
int main(){
        float xReal[N] = { campioni prelevati da matlab } ;
        float freq = 0;
        float x[N*2];
        float max = 0;
        int indMax = 0;
        float w[N];
        int i = 0;
        float freqcamp = 44100.0;

        for(i = 0; i < N; i++){
                x[2*i] = xReal[i];
                x[2*i+1] = 0.0;
         }

        gen_w_r2(w, N);
        bit_rev(w, N>>1);
        DSPF_sp_cfftr2_dit(x, w, N);
        bit_rev(x, N);
        xReal[0]=0;
        for(i = 1; i < N/2; i++) {
                xReal[i] = sqrt(x[2*i]*x[2*i] + x[2*i+1]*x[2*i+1]);
                if (xReal[i] > max) {
                    max = xReal[i];
                    indMax = i;
                }
  }
  freq =(indMax*freqcamp)/N; }
```

# IMPLEMENTAZIONE SU DSP REALE E PROVE SPERIMENTALI

## TRADUZIONE DEL CODICE DI SIMULAZIONE

Il codice prodotto in fase di simulazione, è stato modificato in questa fase per ottenere un codice in grado di gestire gli interrupt e la memorizzazione dei diversi campioni necessari per il riconoscimento del segnale. La funzione "main()", che in simulazione svolgeva tutto il lavoro, ora inizializza soltanto gli interrupt di sistema ed i vari componenti, prima di mettersi in ascolto aspettando un interrupt.

Il lavoro è ora svolto interamente dalla interrupt service routine, la quale legge un nuovo campione, e scartati i primi campioni (per problemi legati al rumore della pressione dei tasti), decide, in base al valore di due variabili booleane, se quel campione è da salvare o meno (si inizia la registrazione solo quando il primo valore ha superato un valore noto detto soglia, in questo modo è possibile evitare la registrazione del rumore iniziale). Una volta registrati i 1024 campioni di interesse, l'isr chiama una funzione denominata "trovanota", che esegue tutti i calcoli, dalla fft, alla determinazione del valore della frequenza più vicina. Fatto questo, tramite l'utilizzo di un costrutto "switch" e di una istruzione "printf", l'isr stampa a video il nome della nota suonata.

## PROVE SPERIMENTALI

Le prime prove sperimentali evidenziavano un problema nel calcolo della FFT. I valori di frequenza ottenuti erano infatti assolutamente errati, con valori altissimi. Dopo numerose prove, grazie all'utilizzo della tecnica di Debug "run by step" e all'analisi dei grafici prodotti dal software, è stato identificato il problema: la scarsa qualità del microfono in dotazione. Per ovviare a questo inconveniente, abbiamo collegato il computer portatile dal quale venivano riprodotti i suoni, direttamente alla scheda dsp, ottenendo subito risultati più che soddisfacenti. Un ultimo problema era dovuto al fatto che una singola nota di lunga durata veniva riconosciuta come due o tre esecuzioni della stessa nota. Questo problema è stato risolto innalzando ancora il valore soglia. In questo modo, solo l'alto volume iniziale della nota riesce a superare tale valore e a portare il programma nella situazione di registrazione.

In conclusione, il sistema ottenuto riconosce tutte le note della 4° scala dal DO al SI, compresi i semitoni. I tempi di esecuzione sono ragionevoli e l'affidabilità è molto alta. E' ovviamente possibile estendere il riconoscimento alle altre scale musicali semplicemente inserendo nel codice sorgente i valori delle rispettive frequenze.

Il codice finale risulta:

```
#include <DSPF_sp_cfftr2_dit.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <c6x.h>
#include <csl.h>
#include <csl_irq.h>
#include <csl_mcbsp.h>
#include <bsl.h>
#include <bsl_ad535.h>
#define N 1024

void bit_rev(float* x, int n);
void gen_w_r2(float* w, int n);
int trovaNota(void);
void codec_init(void);
void init_HWI(void);
void isr1(void);


float max = 0;
int indMax = 0;
int i = 0;
int j = 0;
int z = 0;
float freqcamp = 8000.0;
int registrando = 0;
float valoresoglia = 10000;
AD535_Handle hAD535;

AD535_Config my_AD535_Config = {
        AD535_LOOPBACK_DISABLE,
        AD535_MICGAIN_OFF,
        AD535_GAIN_0DB,
        AD535_GAIN_0DB
};

float freq = 0;
float xReal[N];

float w[N];
float x[N*2];

void main(){
        CSL_init();
        BSL_init();
        codec_init();
        init_HWI();

        while(1);
}

void codec_init() {
        hAD535 = AD535_open(AD535_localId);
        AD535_reset(hAD535);
        AD535_config(hAD535, &my_AD535_Config);
```

```
}
void init_HWI(void) {
	IRQ_globalEnable();
	IRQ_enable(IRQ_EVT_RINT0);
}

void isr1(void) {
	int nota = 0;
	short sample = (short) AD535_read(hAD535);

	if(registrando){

		if(z < 5000){
			z++;
		}

		else if(j < N){
			xReal[j] = sample;
			j++;
		}

		else {
			z = 0;
			j = 0;
			registrando = 0;
			IRQ_globalDisable();
			nota = trovaNota();

			switch (nota) {
			case 262:
			printf("nota: DO\n");
			break;
			case 277:
			printf("nota: DO#\n");
			break;
			case 294:
			printf("nota: RE\n");
			break;
			case 311:
			printf("nota: MIb\n");
			break;
			case 330:
			printf("nota: MI\n");
			break;
			case 349:
			printf("nota: FA\n");
			break;
			case 370:
			printf("nota: FA#\n");
			break;
			case 392:
			printf("nota: SOL\n");
			break;
			case 415:
			printf("nota: SOL#\n");
			break;
			case 440:
			printf("nota: LA\n");
```

```
                        break;
                        case 466:
                        printf("nota: SIb\n");
                        break;
                        case 494:
                        printf("nota: SI\n");
                        break;
                        }

                        for(i = 0; i < N; i++){
                                xReal[i] = 0;
                        }
                        for(i = 0; i < N*2; i++){
                                x[i] = 0;
                        }

                        freq = 0;
                        indMax = 0;
                        nota = 0;
                        max = 0;
                        IRQ_globalEnable();
                }
        }

        else {

                if(sample < valoresoglia) {}
                else {
                        registrando = 1;
                        xReal[0] = sample;
                        j=1;
                }
        }
}

int trovaNota() {
        int freqNote[12] = {262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494};
        float diff=0;
        float diff2=0;
        int index = 0;

        for(i = 0; i < N; i++){
                x[2*i] = xReal[i];
                x[2*i+1] = 0.0;
         }

        gen_w_r2(w, N);
        bit_rev(w, N>>1);
        DSPF_sp_cfftr2_dit(x, w, N);
        bit_rev(x, N);

        xReal[0]=0;
        for(i = 1; i < N/2; i++) {
                xReal[i] = sqrt(x[2*i]*x[2*i] + x[2*i+1]*x[2*i+1]);
                if (xReal[i] > max) {
                    max = xReal[i];
                    indMax = i;
                }
```

```
	}
	freq =(indMax*freqcamp)/N;

	 diff = sqrt((freq - freqNote[0])*(freq - freqNote[0]));

	for (i=1; i < 12; i++){
		diff2 = sqrt((freq - freqNote[i])*(freq - freqNote[i]));
		if (diff2 < diff){
			diff = diff2;
			index++;
		}
	}
	return freqNote[index];
}

/* generate real and imaginary twiddle
table of size n/2 complex numbers */
void gen_w_r2(float* w, int n) {
	int i;
	float pi = 4.0*atan(1.0);
	float e = pi*2.0/n;

	for(i=0; i < ( n>>1 ); i++) {
		w[2*i] = cos(i*e);
		w[2*i+1] = sin(i*e);
	}
}

void bit_rev(float* x, int n) {
	int i, j, k;
	float rtemp, itemp;
	j = 0;

	for(i=1; i < (n-1); i++){
		k = n >> 1;

		while(k <= j){
			j -= k;
			k >>= 1;
		}

		j += k;

		if(i < j) {
			rtemp = x[j*2];
			x[j*2] = x[i*2];
			x[i*2] = rtemp;
			itemp = x[j*2+1];
			x[j*2+1] = x[i*2+1];
			x[i*2+1] = itemp;
		}
	}
}
```