Università Politecnica delle Marche

Multirate Digital Signal Processing And Adaptive Filter Banks

Confronto fra algoritmo LMS e Fast Deconvolution per la cancellazione del crosstalk

Matteo Orlandini e Jacopo Pagliuca

Prof.ssa Stefania Cecchi
Dott.ssa Valeria Bruschi

10 agosto 2021

Indice

1 Introduzione

2 Dataset

Il dataset usato è composto ad un ampio set di misurazioni della funzione di trasferimento relativa alla testa (HRTF) di un microfono dummy-head KEMAR. Le misurazioni consistono nelle risposte impulsive dell'orecchio sinistro e destro di un altoparlante Realistic Optimus Pro 7 montato a 1,4 metri dal KEMAR. Sono state utilizzate sequenze binarie pseudo-casuali per ottenere le risposte impulsive a una frequenza di campionamento di 44.1 kHz. [1]

Le misurazioni sono state effettuate nella camera anecoica del MIT. Il KEMAR è stato montato in posizione verticale su un giradischi motorizzato che può essere ruotato con precisione sotto il controllo del computer. L'altoparlante è stato montato su un supporto a braccio che ha consentito il posizionamento accurato dell'altoparlante a qualsiasi elevazione rispetto al KEMAR. Pertanto, le misurazioni sono state effettuate un'elevazione alla volta, impostando l'altoparlante all'altezza corretta e quindi ruotando il KEMAR su ciascun azimut.

I dati HRTF vengono archiviati nelle directory per elevazione. Ogni nome di directory ha il formato "elevEE", dove EE è l'angolo di elevazione. All'interno di ogni directory, il nome di ogni file ha il formato "XEEeAAAa.wav" dove X può essere "L" o "R" rispettivamente per la risposta dell'orecchio sinistro e destro, "EE" è l'angolo di elevazione della sorgente in gradi, da -40° a 90°, e AAA è l'azimut della sorgente in gradi, da 0° a 355°. Gli angoli di elevazione e azimut indicano la posizione della sorgente rispetto al KEMAR, in modo che, ad esempio, in corrispondenza dell'elevazione 0 e azimut 0 sia di fronte al KEMAR , l'elevazione 90 è direttamente sopra il KEMAR, elevazione 0 e azimut 90 è a destra del KEMAR. Ad esempio, il file "R-20e270a.wav" è la risposta dell'orecchio destro, con la sorgente 20 gradi sotto il piano orizzontale e 90 gradi a sinistra della testa.

3 Descrizione teorica degli algoritmi

3.1 LMS

Per la cancellazione del crosstalk l'algoritmo più comune è quello dell'LMS. Nonostante sia semplice e accurato ha una veloce convergenza. Una tipica situazione di ascolto con due altoparlanti con cancellazione del closstalk è rappresentata in figura... La rappresentazione in frequenza dei segnali desiderati binaurali è indicata con X_1 e X_2 per suoni che raggiungono rispettivamente orecchio destro e sinistro. Con Y_1 e Y_2 invece, si indicano i suoni che effettivamente raggiungono l'ascoltatore, attraversando il sistema. Il sistema può essere rappresentato come:

$$\mathbf{Y}{=}\mathbf{CHX} \tag{1}$$

ovvero

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \tag{2}$$

Per avere il segnale in uscita uguale a quello desiderato H dovrebbe essere l'inversa di C. La diretta inversa di C però non garantisce la cancellazione del crosstalk in quanto gli elementi non soddisfano la condizione di fase minima. La'approccio utilizzato è quindi quello dell'algoritmo LMS. La precedente equazione viene riscritta come:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \begin{bmatrix} C_{11}X_1 & C_{12}X_1 & C_{11}X_2 & C_{12}X_2 \\ C_{21}X_1 & C_{22}X_1 & C_{21}X_2 & C_{22}X_2 \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{21} \\ H_{12} \\ H_{22} \end{bmatrix} \tag{3}$$

In generale, si può scrivere il filtraggio nell'n-esimo istante di un segnale $\mathbf{a[n]} = [a[n], a[n-1], \ldots, a[n-M+1]]^T$ con un filtro $\mathbf{b[n]} = [b_0[n], b_1[n], \ldots, b_{M-1}[n]]^T$ di lunghezza M come

$$y[n] = \sum_{j=0}^{M-1} b_j[n]a[n-j] = \mathbf{b}^T[\mathbf{n}] \cdot \mathbf{a}^T[\mathbf{n}]. \tag{4}$$

Il filtraggio dei segnali di riferimento $x_i[n]$ con le hrir c_{lm} è dato dalla seguente equazione

$$r_{ilm}[n] = \sum_{j=0}^{M-1} c_{lm}[j]x_i[n-j]. \tag{5}$$

Si può applicare la formula (4) all'equazione (5) per calcolare le $r_{ilm}[n]$ nel seguente modo:

$$r_{ilm}[n] = \sum_{j=0}^{M-1} c_{lm,j}[n]x_i[n-j] = \mathbf{c_{lm}}^T[\mathbf{n}] \cdot \mathbf{x_i}^T[\mathbf{n}]. \tag{6}$$

Il segnale ricevuto ad ogni orecchio è dato da:

$$y_i[n] = r_{1i1}[n] \circledast h_{11}[n] + r_{1i2}[n] \circledast h_{21}[n] + r_{2i1}[n] \circledast h_{12}[n] + r_{2i2}[n] \circledast h_{22}[n] \tag{7}$$

dove i, l, m assumono i valori 1 o 2. Il criterio dell'algoritmo LMS è la minimizzazione della funzione costo

$$J = E[e[n]^2] = E[(d[n] - y[n])^2] \tag{8}$$

dove $e[n]$, $d[n]$ e $y[n]$ sono definiti come

$$e[n] = \begin{bmatrix} e_1[n] \\ e_2[n] \end{bmatrix}, d[n] = \begin{bmatrix} d_1[n] \\ d_2[n] \end{bmatrix}, y[n] = \begin{bmatrix} y_1[n] \\ y_2[n] \end{bmatrix} \tag{9}$$

La minimizzazione di J avviene con il metodo steepest descend. Nel dominio del tempo discreto il sistema si può scrivere come

$$\begin{bmatrix} y_1[n] \\ y_2[n] \end{bmatrix} = \begin{bmatrix} c_{11}[n] \circledast x_1[n] & c_{12}[n] \circledast x_1[n] & c_{11}[n] \circledast x_2[n] & c_{12}[n] \circledast x_2[n] \\ c_{21}[n] \circledast x_1[n] & c_{22}[n] \circledast x_1[n] & c_{21}[n] \circledast x_2[n] & c_{22}[n] \circledast x_2[n] \end{bmatrix} \circledast \begin{bmatrix} h_{11}[n] \\ h_{21}[n] \\ h_{12}[n] \\ h_{22}[n] \end{bmatrix} \tag{10}$$

Usando (5), l'equazione (10) diventa

$$\begin{bmatrix} y_1[n] \\ y_2[n] \end{bmatrix} = \begin{bmatrix} r_{111}[n] & r_{112}[n] & r_{211}[n] & r_{212}[n] \\ r_{121}[n] & r_{122}[n] & r_{221}[n] & r_{222}[n] \end{bmatrix} \circledast \begin{bmatrix} h_{11}[n] \\ h_{21}[n] \\ h_{12}[n] \\ h_{22}[n] \end{bmatrix} \tag{11}$$

Per ottenere l'errore si calcola la differenza fra il segnale desiderato è l'uscita effettiva.

$$e_1^{(1)} = d_1 - (r_{111} \circledast h_{11} + r_{112} \circledast h_{21} + r_{211} \circledast h_{12} + r_{212} \circledast h_{22})$$

l'aggiornamento avviene come di seguito:

$$h^{(k+1)} = h^{(k)} - \mu e_i^{(k)} \cdot \mathbf{r}_i^T \tag{12}$$

dove $\mathbf{r}_i$ è l'i-esima riga della matrice che contiene le r_{ilm} nell'equazione (11) e $e_i^{(k)}$ è l'errore k-esimo step e all'i-esima riga. Esplicitando l'equazione (12), si ottiene

$$
\begin{aligned}
h_{11}^{(k+1)}[n] &\leftarrow h_{11}^{(k)}[n] - \mu e_1^{(1)} \cdot r_{111}[n] \\
h_{21}^{(k+1)}[n] &\leftarrow h_{21}^{(k)}[n] - \mu e_1^{(1)} \cdot r_{112}[n] \\
h_{12}^{(k+1)}[n] &\leftarrow h_{12}^{(k)}[n] - \mu e_1^{(1)} \cdot r_{211}[n] \\
h_{22}^{(k+1)}[n] &\leftarrow h_{22}^{(k)}[n] - \mu e_1^{(1)} \cdot r_{212}[n]
\end{aligned}
$$

Allo stesso modo, per il secondo canale l'errore è definito come:

$$e_2^{(2)} = d_2 - (r_{121} \circledast h_{11} + r_{122} \circledast h_{21} + r_{221} \circledast h_{12} + r_{222} \circledast h_{22})$$

$$
\begin{aligned}
h_{11}^{(k+1)}[n] &\leftarrow h_{11}^{(k)}[n] - \mu e_2^{(2)} \cdot r_{121}[n] \\
h_{21}^{(k+1)}[n] &\leftarrow h_{21}^{(k)}[n] - \mu e_2^{(2)} \cdot r_{122}[n] \\
h_{12}^{(k+1)}[n] &\leftarrow h_{12}^{(k)}[n] - \mu e_2^{(2)} \cdot r_{221}[n] \\
h_{22}^{(k+1)}[n] &\leftarrow h_{22}^{(k)}[n] - \mu e_2^{(2)} \cdot r_{222}[n]
\end{aligned}
$$

3.2 Fast Deconvolution

La deconvoluzione [2] [3] [4], nella sua forma più elementare, può essere descritta come il compito di calcolare l'input di un sistema a tempo discreto conoscendo il suo output. Di solito si presume che il sistema sia lineare e che la relazione input output sia nota con precisione. In acustica e audio, la deconvoluzione a singolo canale è particolarmente utile poiché può compensare la risposta di trasduttori imperfetti come cuffie, altoparlanti e amplificatori. La deconvoluzione multicanale è necessaria nella progettazione di sistemi di cancellazione della diafonia e sistemi di imaging di sorgenti virtuali.

Nel progetto siamo interessati alle tecniche di deconvoluzione allo scopo di progettare filtri digitali per la riproduzione del suono su due canali. Più specificamente, dato un set di altoparlanti S, l'obiettivo è riprodurre un campo sonoro desiderato nei punti R dello spazio nel modo più accurato possibile. Questo principio è applicato dai cosiddetti sistemi di cancellazione della diafonia che vengono utilizzati per riprodurre registrazioni binaurali su due altoparlanti. In questo caso, viene utilizzata una matrice 2×2 di filtri digitali per compensare sia la risposta ambientale che la risposta degli altoparlanti, e anche per annullare la diafonia dall'altoparlante sinistro all'orecchio destro e viceversa. Un problema correlato è quello di ottenere una perfetta "dereverberazione" della risposta di una stanza in una posizione del microfono utilizzando due filtri digitali per calcolare l'ingresso a due altoparlanti. La fast deconvolution un metodo molto veloce per calcolare una matrice di filtri digitali che può essere utilizzata per controllare le uscite di un impianto multicanale. Questo metodo è tipicamente più veloce di diversi ordini di grandezza rispetto ai metodi nel dominio del tempo. Combina i principi dell'inversione dei minimi quadrati nel dominio della frequenza e il metodo di regolarizzazione di ordine zero che viene tradizionalmente utilizzato quando ci si trova di fronte a un problema di inversione mal condizionata.

La regolarizzazione dipendente dalla frequenza viene utilizzata per prevenire picchi elevati nella risposta in ampiezza dei filtri ottimali. Un ritardo di modellazione viene utilizzato per garantire che la rete di cancellazione del cross-talk funzioni bene non solo in termini di ampiezza, ma anche in termini di fase. L'algoritmo presuppone che sia possibile utilizzare filtri ottimali lunghi, e funziona bene solo quando due parametri di regolarizzazione, un fattore di forma e un fattore di guadagno, sono impostati in modo appropriato. In pratica, i valori dei due parametri di regolarizzazione sono determinati più facilmente da esperimenti per tentativi.

Consideriamo una funzione di costo del tipo

$$J = E + \beta V(f) \tag{13}$$

dove E è una misura dell'errore della pressione sonora

$$E = |Y_1 - X_1|^2 + |Y_2 - X_2|^2 \tag{14}$$

e V è una funzione della frequenza che indica il costo computazionale. Il numero $\beta \geq 0$ è un parametro di regolarizzazione che determina quanto peso assegnare alla funzione V. Poiché non sappiamo a priori se la matrice C è non singolare per determinate frequenze, la matrice H di cancellazione del crosstalk può contenere valori molto alti. All'aumentare di β da zero a infinito, J cambia gradualmente dalla minimizzazione della sola funzione di errore E alla minimizzazione dello sforzo computazionale V.

Siano S i segnali trasferiti agli altoparlanti facendo passare il segnale X attraverso la matrice di cancellazione del crosstalk H. Otteniamo

$$V(f) = S_b^+ S_b \tag{15}$$

con

$$S_b = BS = BHX \tag{16}$$

dove B è una matrice 2×2 e il simbolo $+$ indica l'inversa generalizzata della matrice S_b. La soluzione approssimata della funzione J è definita da

$$H(z) = \left[C^T(z^{-1})C(z) + \beta B^T B)\right]^{-1} C^T(z^{-1}) \tag{17}$$

dove l'apice T denota la trasposta della matrice. Se la matrice B è uguale alla matrice identità I, si ottiene $S_b = S$, dunque l'equazione (17) diventa

$$H(z) = \left[C^T(z^{-1})C(z) + \beta I)\right]^{-1} C^T(z^{-1})z^{-m} \tag{18}$$

dove la componente z^{-m} implementa un ritardo di m campioni.

Le equazioni (17) e (18) rappresentano una espressione di $H(z)$ nel dominio continuo della frequenza. Se viene usata una FFT a N punti per campionare la risposta in frequenza $H(z)$, allora il valore di $H[k]$ è dato da

$$H[k] = \left[C^H[k]C[k] + \beta I)\right]^{-1} C^H[k] \tag{19}$$

dove k indica la k-esima frequenza corrispondente a $\exp(j2\pi k/N)$ e l'apice H denota l'operatore Hermitiano che fa la trasposta coniugata del suo argomento. Dall'equazione (19) si può osservare che ponendo $\beta = 0$ si ottiene $H = C^{-1}$. In questo caso, poiché $Y = CHX = CC^{-1}X = IX = X$, si ottiene in uscita il segnale d'ingresso.

Per calcolare la risposta impulsiva del filtro occorre seguire i seguenti passi:

1. si calcola la matrice 2×2 $C[k]$ tramite una FFT delle risposte impulsive $c_{ij}[n]$ con $i = 1, 2$ e $j = 1, 2$. Ad esempio, $C_{11}[k]$ contiene l'ampiezza della k-esima armonica della FFT di c_{11},
2. si calcola $H[k]$ usando la formula (19),
3. si calcola $h[n]$ facendo una FFT inversa a N punti,
4. si implementa uno shift ciclico di m campioni per ogni elemento di $h_{ij}[n]$ con $i = 1, 2$ e $j = 1, 2$.

Dato che l'uscita Y è data da

$$Y = CHX = \begin{bmatrix} C_{11}X_1 & C_{12}X_1 & C_{11}X_2 & C_{12}X_2 \\ C_{21}X_1 & C_{22}X_1 & C_{21}X_2 & C_{22}X_2 \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{21} \\ H_{12} \\ H_{22} \end{bmatrix} \tag{20}$$

nell'implementazione pratica non si può calcolare tutta l'uscita con la sola operazione matriciale (20), occorre usare la tecnica dell'overlap and save per filtrare l'ingresso X con i filtri C e H. L'overlap and save è utile per eseguire un filtraggio in real time con un filtro a risposta impulsiva finita.

Questa tecnica viene usata per fare la convoluzione a blocchi tra un segnale di ingresso $x[n]$ molto lungo e un filtro FIR $h[n]$:

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{+\infty} h[m] \cdot x[n-m] = \sum_{m=1}^{M} h[m] \cdot x[n-m] \tag{21}$$

poiché $h[m] = 0$ per $m \in [1, M]$.

L'overlap and save permette di calcolare dei blocchi di $y[n]$ di lunghezza L e concatenarli insieme per formare il segnale di uscita completo. Si definisce il k-esimo blocco d'ingresso come

$$x_k[n] = \begin{cases} x[n+kL], & 1 \leq n \leq L+M-1 \\ 0 & \text{altrove} \end{cases} \tag{22}$$

quindi il k-esimo blocco di uscita è dato da

$$y_k[n] = x_k[n] * h[n] = \sum_{m=1}^{M} h[m] \cdot x_k[n-m]. \tag{23}$$

Si può definire l'uscita per $kL + M \leq n \leq KL + L + M - 1$, o in modo equivalente per $M \leq n - kL \leq L + M - 1$, nel seguente modo:

$$y[n] = \sum_{m=1}^{M} h[m] \cdot x_k[n - kL - m] = y_k[n - kL]. \tag{24}$$

L'implementazione dell'overlap and save con un filtro composto da $(L+M-1)$ tappi consiste nei seguenti passi:

1. si divide il segnale di ingresso in blocchi di lunghezza L
2. nel caso del primo blocco si aggiungono $M - 1$ zeri all'inizio, altrimenti si aggiungono all'inizio del blocco gli ultimi $M - 1$ campioni del blocco precedente, come mostrato in figura 1,
3. si fa la FFT a $L + M - 1$ punti del k-esimo blocco di ingresso,
4. si fa la FFT a $L + M - 1$ punti del filtro FIR $h[n]$
5. si moltiplicano le FFT calcolate per trovare la risposta in frequenza del k-esimo blocco di uscita,
6. si fa la FFT inversa a $L + M - 1$ punti del k-esimo blocco di uscita,

7. si scartano i primi $M-1$ punti, ottenendo il k-esimo blocco in uscita di lunghezza L, come mostrato nei blocchi di uscita y_k di figura 1.

Un'implementazione più efficiente è invece quella descritta di seguito, in cui i primi due punti precedentemente elencati rimangono uguali:

3. si fa la FFT a $2 \cdot (L+M-1)$ punti del k-esimo e del $(k-1)$-esimo blocco di ingresso lunghi rispettivamente $(L+M-1)$ campioni,

4. si fa la FFT a $2 \cdot (L+M-1)$ punti del filtro FIR $h[n]$ con un padding di $(L+M-1)$ zeri

5. si moltiplicano le FFT calcolate per trovare la FFT del blocco di uscita,

6. si fa la FFT inversa del blocco di uscita,

7. si scartano i primi $L+M-1$ punti nel tempo dell'uscita.

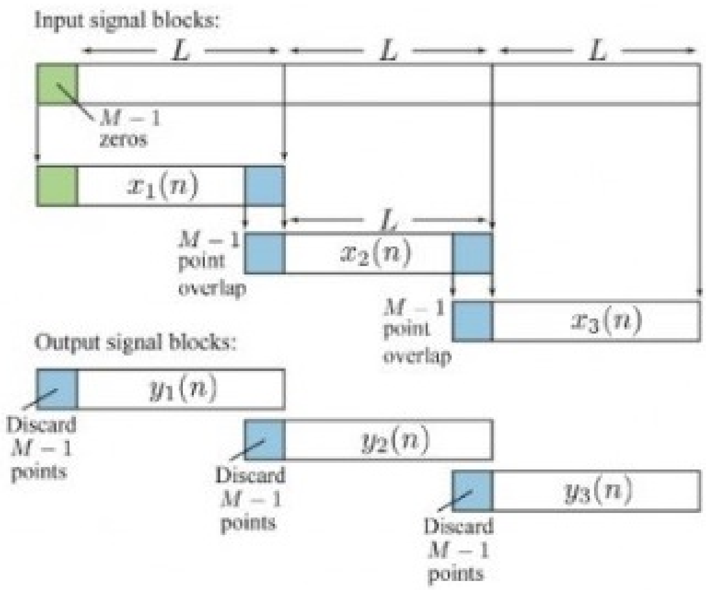

Figura 1: Metodo Overlap and Save

4 Codice Matlab

4.1 LMS

Nel codice dell'algoritmo LMS in Matlab si carica il file audio in formato wav "Daft Punk - Get Lucky_cut.wav". Poiché questo formato contiene i campioni del canale sinistro e di quello destro, si dividono i due canali salvando il primo nella variabile `x1` e il secondo in `x2`.

```
[x, Fsample] = audioread('Daft Punk - Get Lucky_cut.wav');
x1 = x(:, 1);
x2 = x(:, 2);
```

Codice 1: Caricamento del file audio

Allo stesso modo vendono caricate le quattro HRIR `c11`, `c12`, `c21` e `c22`.

```
% c11: HRIR left loudspeaker - left ear
[c11,Fs] = audioread('HRTF_measurements/elev0/L0e330a.wav');
% c12: HRIR right loudspeaker - left ear
[c12,~] = audioread('HRTF_measurements/elev0/L0e030a.wav');
% c21: HRIR left loudspeaker - right ear
[c21,~] = audioread('HRTF_measurements/elev0/R0e330a.wav');
% c22: HRIR right loudspeaker - righ ear
[c22,~] = audioread('HRTF_measurements/elev0/R0e030a.wav');
```

Codice 2: Caricamento delle HRIR

Viene salvata la lunghezza dei filtri c_{11}, c_{12}, c_{21} e c_{22} nella variabile `M`. Si ha che $M = 512$ e si imposta il ritardo temporale `tau` pari a `M`. Conoscendo il valore del ritardo temporale, è possibile costruire i due segnali desiderati `d1` e `d2` mettendo `tau` zeri all'inizio del vettore d'ingresso.

```
M = length(c11);      % lunghezza dei filtri c11, c12, c21, c22
% il segnale desiderato e' x ritardato di tau campioni
tau = M;    % ritardo temporale
d1 = [zeros(tau,1); x1(1:end-tau)]; % segnale desiderato 1 ...
    (left)
d2 = [zeros(tau,1); x2(1:end-tau)]; % segnale desiderato 2 ...
    (right)
```

Codice 3: Costruzione del segnale desiderato

Vengono inizializzati i vettori `y1` e `y2` di uscita, i vettori `e1` e `e2` che contengono l'errore per i due canali, i vettori $\mathtt{r}_{\mathtt{ilm}}$ che contengono il filtraggio di `x1` e `x2` con `c11`, `c12`, `c21` e `c22`. Questi vettori sono tutti di lunghezza pari a quella de segnale `x`, cioè `L`.

```
L = length(x1);
y1 = zeros(L,1);  % output 1 (left)
y2 = zeros(L,1);  % output 2 (right)

e1 = zeros(L,1);  % errore 1 (left)
e2 = zeros(L,1);  % errore 2 (right)

r111 = zeros(L,1);    % uscita di x1 filtrato da c11 per ...
   output y1
r112 = zeros(L,1);    % uscita di x1 filtrato da c12 per ...
   output y1
r211 = zeros(L,1);    % uscita di x2 filtrato da c11 per ...
   output y1
r212 = zeros(L,1);    % uscita di x2 filtrato da c12 per ...
   output y1
r222 = zeros(L,1);    % uscita di x2 filtrato da c22 per ...
   output y2
r221 = zeros(L,1);    % uscita di x2 filtrato da c21 per ...
   output y2
r122 = zeros(L,1);    % uscita di x1 filtrato da c22 per ...
   output y2
r121 = zeros(L,1);    % uscita di x1 filtrato da c21 per ...
   output y2
```
Codice 4: Inizializzazione dei vettori di lunghezza L

Successivamente, vengono inizializzati i vettori di lunghezza `M`, le variabili **`h11`**, **`h12`**, **`h21`** e **`h22`** contengono i filtri di ricostruzione, i buffer **`x1buff`** e **`x2buff`** necessari per il filtraggio con i c e i buffer delle uscite r_{ilm} dove $i, l, m = 0, 1$.

```
x1buff = zeros(M, 1);
x2buff = zeros(M, 1);

r111buff = zeros(M,1);
r112buff = zeros(M,1);
r211buff = zeros(M,1);
r212buff = zeros(M,1);
r222buff = zeros(M,1);
r221buff = zeros(M,1);
r122buff = zeros(M,1);
r121buff = zeros(M,1);
```
Codice 5: Inizializzazione dei vettori di lunghezza M

Nel codice 6 viene usato un $\mu = 10^{-3}$ e viene implementata l'equazione (6) che effettua il filtraggio tra le hrir e il segnale di ingresso. Per fare questo, si utilizza un buffer che aggiunge in testa il campione entrante invertendo il segnale di ingresso. Facendo il prodotto scalare dei buffer di ingresso con le rispettive c_{lm}, si ottiene il filtraggio desiderato.

```
mu = 1e-3;
for n = 1:L
    x1buff = [x1(n); x1buff(1:end-1)];
    x2buff = [x2(n); x2buff(1:end-1)];

    r111(n) = c11'*x1buff;
    r112(n) = c12'*x1buff;
    r211(n) = c11'*x2buff;
    r212(n) = c12'*x2buff;
    r222(n) = c22'*x2buff;
    r221(n) = c21'*x2buff;
    r122(n) = c22'*x1buff;
    r121(n) = c21'*x1buff;
```

Codice 6: Filtraggio del segnale di ingresso con le hrir

L'equazione (6) viene nuovamente applicata nel codice 7 per calcolare le uscite. I segnali intermedi r_{ilm} vanno a costituire un buffer e attraversano i filtri h. Le quattro uscite dei filtri h vengono infine sommate per ottenere `y1` e `y2`, come mostrato nell'equazione (7).

```
    r111buff = [r111(n); r111buff(1:end-1)];
    r112buff = [r112(n); r112buff(1:end-1)];
    r211buff = [r211(n); r211buff(1:end-1)];
    r212buff = [r212(n); r212buff(1:end-1)];
    r222buff = [r222(n); r222buff(1:end-1)];
    r221buff = [r221(n); r221buff(1:end-1)];
    r122buff = [r122(n); r122buff(1:end-1)];
    r121buff = [r121(n); r121buff(1:end-1)];

    y1(n) = ...
        h11'*r111buff+h21'*r112buff+h12'*r211buff+h22'*r212buff;
    y2(n) = ...
        h11'*r121buff+h21'*r122buff+h12'*r221buff+h22'*r222buff;
```

Codice 7: Calcolo delle uscite

Viene quindi calcolato l'errore come la differenza fra il segnale desiderato e le uscite.

```
    e1(n) = d1(n)-y1(n);
    e2(n) = d2(n)-y2(n);
```

Codice 8: Calcolo dell'errore

I valori dell'errore vengono poi utilizzati per aggiornare i tappi dei filtri `h11`, `h12`, `h21` e `h22` tramite un altro ciclo for, utilizzando la formula (12).

```
    for k = 1:M
        h11(k) = ...
            h11(k)+mu*(e1(n)*r111buff(k)+e2(n)*r121buff(k));
        h12(k) = ...
            h12(k)+mu*(e1(n)*r211buff(k)+e2(n)*r221buff(k));
        h21(k) = ...
            h21(k)+mu*(e1(n)*r112buff(k)+e2(n)*r122buff(k));
        h22(k) = ...
            h22(k)+mu*(e1(n)*r212buff(k)+e2(n)*r222buff(k));
    end
end
```

Codice 9: Aggiornamento del filtro

4.2 Fast Deconvolution

Nella Fast Deconvolution, come per il codice della LMS, si carica il file audio e le risposte impulsive `c11`, `c12`, `c21` e `c22` come mostrato nei codici 1 e 2. A differenza dell'algoritmo LMS, nella Fast Deconvolution si usa un overlap and save quindi occorre scegliere come dividere in blocchi il segnale di ingresso. Si implementa l'overlap and save come descritto nel capitolo 3.2, con dei blocchi di lunghezza $L = 4096$, pari alla lunghezza di default del frame di Nu-Tech, e un overlap al 50%, quindi $M = L/2 = 2048$. La lunghezza totale del frame con overlap sarà quindi $\text{fs} = L + M - 1 = 4096 + 2048 - 1 = 6143$, per una FFT efficiente si sceglie la potenza di 2 più vicina a 6143, quindi $\text{fftLen} = 2^{13} = 8192$.

```
L = 4096;
M = L/2;
fs = L + M - 1; % frame size
fftLen = 2.^nextpow2(fs);
```

Codice 10: Parametri overlap and save

Per eseguire l'overlap and save è necessaria la FFT di lunghezza pari a **`fftLen`** dei filtri C e H. Poiché i filtri H non sono ancora stati calcolati, vengono inizializzati a zero.

```
C11 = fft(c11, fftLen);     % HRTF left loudspeaker - left ear
C12 = fft(c12, fftLen);     % HRTF right loudspeaker - left ear
C21 = fft(c21, fftLen);     % HRTF left loudspeaker - right ear
```

```
C22 = fft(c22, fftLen);     % HRTF right loudspeaker - right ear

H11 = zeros(fftLen,1);     % filtro di cancellazione del ...
   crosstalk input 1 output 1
H12 = zeros(fftLen,1);     % filtro di cancellazione del ...
   crosstalk input 2 output 1
H21 = zeros(fftLen,1);     % filtro di cancellazione del ...
   crosstalk input 1 output 2
H22 = zeros(fftLen,1);     % filtro di cancellazione del ...
   crosstalk input 2 output 2
```

Codice 11: FFT delle risposte impulsive del canale e inizializzazione dei filtri di cancellazione del crosstalk

L'implementazione dell'equazione (18) è mostrato nel codice 12 con $\beta = 0.1$ e $B = I$, dove I è la matrice identità. Per ogni frequenza k dei filtri C_{11}, C_{12}, C_{21} e C_{22}, composti da `fftLen` punti in frequenza, viene costruita una matrice $C[k]$ nel seguente modo:

$$C[k] = \begin{bmatrix} C_{11}[k] & C_{12}[k] \\ C_{21}[k] & C_{22}[k] \end{bmatrix}.$$

Per trovare i filtri H_{11}, H_{12}, H_{21} e H_{22} si calcola la matrice H come mostrato nell'equazione (18) dove H è una matrice 2×2 costruita nel seguente modo:

$$H[k] = \begin{bmatrix} H_{11}[k] & H_{12}[k] \\ H_{21}[k] & H_{22}[k] \end{bmatrix}.$$

I filtri H_{11}, H_{12}, H_{21} e H_{22} sono presi dagli elementi della matrice H.

```
beta = 0.1;
B = [1 0; 0 1];
for k = 1:length(C11)
    C = [C11(k) C12(k); C21(k) C22(k)];
    H = (C'*C+beta*(B'*B))^(-1)*C';
    H11(k) = H(1, 1);
    H12(k) = H(1, 2);
    H21(k) = H(2, 1);
    H22(k) = H(2, 2);
end
```

Codice 12: Calcolo dei filtri di cancellazione del crosstalk

Dopo aver calcolato i filtri H, si procede con l'implementazione dell'overlap and save inizializzando due buffer `x1Buff` e `x2Buff` con `fs` zeri.

```
x1Buff = zeros(fs, 1);
```

```
x2Buff = zeros(fs, 1);
```

Codice 13: Inizializzazione dei buffer per l'overlap and save

Successivamente si divide il segnale di ingresso, lungo `nPoints`, in blocchi di lunghezza `L`, risultando in un numero di blocchi pari a `nPoints/L`. Poiché non sappiamo a priori se questo valore è intero, lo si approssima per difetto usando la funzione `floor` di Matlab. Per l'`i`-esimo blocco si copiano `L` campioni del segnale di ingresso nel buffer dall'indice M in poi, come mostrato in figura 1.

```
nPoints = length(x1);
for i = 1 : floor(nPoints/L)
    copy L values of x vector into the buffer from M onwards
    x1Buff(fs - L + 1 : fs) = x1((i - 1) * L + 1 : i * L);
    x2Buff(fs - L + 1 : fs) = x2((i - 1) * L + 1 : i * L);
```

Codice 14: Copia di L campioni del segnale di ingresso nel buffer

Dopo aver riempito il buffer, si fa la FFT a `fftLen` punti del buffer di ingresso.

```
    X1BUFF = fft(x1Buff, fftLen);
    X2BUFF = fft(x2Buff, fftLen);
```

Codice 15: FFT del buffer di ingresso

Si calcola il buffer di uscita in frequenza implementando l'equazione (20) come mostrato nel codice 16.

```
    Y1BUFF = ...
        (C11.*H11+C12.*H21).*X1BUFF+(C11.*H12+C12.*H22).*X2BUFF;
    Y2BUFF = ...
        (C21.*H11+C22.*H21).*X1BUFF+(C21.*H12+C22.*H22).*X2BUFF;
```

Codice 16: Calcolo del buffer di uscita in frequenza

Usando il buffer di uscita in frequenza, si calcola la IFFT e si prendono solamente i valori reali.

```
    y1Buff = real(ifft(Y1BUFF));
    y2Buff = real(ifft(Y2BUFF));
```

Codice 17: IFFT del buffer di uscita

L'aggiornamento dell'uscita avviene scartando i primi $M - 1$ campioni e dunque prendendo gli ultimi L campioni del buffer precedentemente calcolato.

```
    % discard first M - 1 values
    y1((i-1) * L + 1 : i * L) = y1Buff(fs - L + 1 : fs);
    y2((i-1) * L + 1 : i * L) = y2Buff(fs - L + 1 : fs);
```

Codice 18: Aggiornamento dell'uscita

Infine, si aggiornano i buffer di ingresso prendendo gli ultimi $M-1$ punti del buffer di ingresso precedente e mettendoli all'inizio del buffer di ingresso successivo.

```
    % first M - 1 values of the new vector must be the last ...
        ones of the previous array
    x1Buff(1 : M - 1) = x1Buff(fs - M + 2 : fs);
    x2Buff(1 : M - 1) = x2Buff(fs - M + 2 : fs);
end
```

Codice 19: Aggiornamento del buffer di ingresso

5 Codice C

5.1 LMS

5.2 Fast Deconvolution

Nella libreria `Plugin.h` viene inizialmente definito il nome del Nuts come "Fast Deconvolution".

```
#define NUTS_NAME    "Fast Deconvolution"
```

Riferimenti bibliografici

[1] B. Gardner. «HRTF Measurements of a KEMAR Dummy-Head Microphone». In: 1994.

[2] O. Kirkeby, P.A. Nelson, H. Hamada e F. Orduna-Bustamante. «Fast deconvolution of multichannel systems using regularization». In: *IEEE Transactions on Speech and Audio Processing* 6.2 (1998), pp. 189–194. DOI: 10.1109/89.661479.

[3] O. Kirkeby, P. Rubak e Angelo Farina. «Analysis of ill-conditioning of multi-channel deconvolution problems». In: feb. 1999, pp. 155–158. ISBN: 0-7803-5612-8.

[4] Ole Kirkeby, Per Rubak, Philip Nelson e Angelo Farina. «Design of Cross-Talk Cancellation Networks by Using Fast Deconvolution». In: (nov. 2000).

[5] Dan Li, Zhong-Hua Fu, Lei Xie e Yanning Zhang. «Comprehensive comparison of the least mean square algorithm and the fast deconvolution algorithm for crosstalk cancellation». In: *2012 International Conference on Audio, Language and Image Processing*. 2012, pp. 224–229. DOI: 10.1109/ICALIP.2012.6376616.