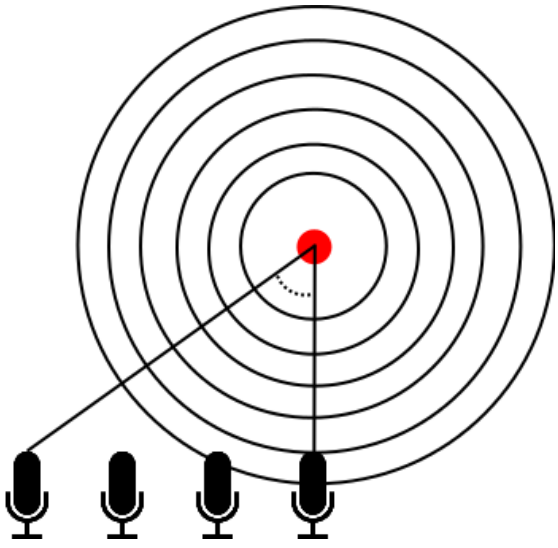
ETIN80 - Algoritmer i signalprocessorer

EigenBeamformer



Author: William Martinsson, D09 (*ada09wma@student.lu.se*)
Mattias Ahlström, D09 (*ada09mah@student.lu.se*)

5 mars 2014

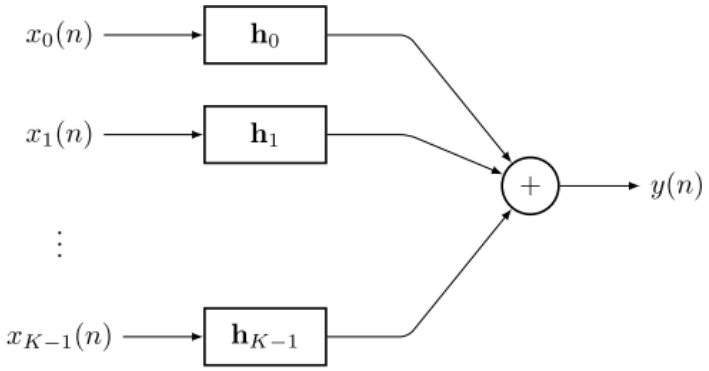
Innehåll

1 Introduktion

Kunskapen kring digital signalbehandling blir viktigare i takt med att användare och konsumenter ställer högre krav på funktionalitet och prestanda hos integreade kretsar. Det blir extra intressant när det kommer till användningsområden så som akustisk signalbehandling, biomedicinsk signalbehandling och kommunikation.
I denna kurs har mest fokus legat på akustisk signalbehandling då det är väldigt tacksamt att utforska vad som är möjligt att utföra med digital signalbehandling. Listan med förslag på olika projekt var lång och det ena projektet lät lika spännande än det andra, tillslut föll valet på att implementera en *beamformer*[1]. En beamformer eller ett spatialt filter har en mängd olika användingsområden förutom att släcka ut en viss ljudkälla i ett rum som ofta är målet när en akustisk beamformer används. Beamforming används också i t.ex. trådlösa nätverk och radar-teknologin.[2]
I denna rapport ska läsaren få en bakgrund kring hur en akustisk beamformer är uppbyggd och hur en implementation av en sådan kan se ut.

2 Bakgrund

Uppgiften att implementera en beamformer kan göras på olika abstraktionsnivåer men dimensionerades utifrån tidigare förkunskaper i ämnet samt den tid som fanns till förfogan att göra implementationen. Efter samtal med handledare för projektet föll valet att designa och implementera en *Signal to Noise and Interference Ratio* (SNIR) beamformer. Med hjälp av eigenvektor-analys är designmålet att maximera den önskade signalens energinivå och minimera energinivån hos störsignalen.



Figur 1: Beamformer där insignalerna noteras från $x_0(n)$ till $x_K - 1(n)$, filter $h_0(n)$ till $h_K - 1(n)$ och utsignal $y(n)$.

2.1 EigenBeamformer

För att kunna konstruera ett optimalt filter till varje beamformerkanal ska följande formel uppfyllas

$$h_{opt} = \arg\max \frac{h^T I h}{h^T R_{nn} h} \tag{1}$$

Ovanstående ekvation innehåller korrelationsmatrisen R_{nn} som byggs upp enligt följande formler

- En insignalvektor där k är kanalen och L är antalet sample

$$x_k(n) = \begin{bmatrix} x_k(n) & x_k(n-1) & \dots & x_k(n-L+1) \end{bmatrix}^T \tag{2}$$

- Från insignalerna x kan korrelationsmatrisen mellan två signaler defineras enligt

$$R_{x_i x_j} = E\left\{x_i(n) x_j^T(n)\right\} \tag{3}$$

- Om R_{xx} sedan expanderas fås följande uttryck

$$R_{xx} = E\left\{ \begin{bmatrix} x_0(n)x_0^T(n) & x_0(n)x_1^T(n) & \dots & x_0(n)x_K^T(n) \\ x_1(n)x_0^T(n) & x_1(n)x_1^T(n) & \dots & x_1(n)x_K^T(n) \\ \vdots & \vdots & \ddots & \vdots \\ x_K(n)x_0^T(n) & x_K(n)x_1^T(n) & \dots & x_K(n)x_K^T(n) \end{bmatrix} \right\} \tag{4}$$

- Den slutgiltiga korrelationsmatrixen är uppbyggd av submatriser där de olika submatriserna är korrelationen mellan två kanaler och ses tydligt nedan

$$R_{xx} = \begin{bmatrix} R_{x_0x_0} & R_{x_0x_1} & \dots & R_{x_0x_K} \\ R_{x_1x_0} & R_{x_1x_1} & \dots & R_{x_1x_K} \\ \vdots & \vdots & \ddots & \vdots \\ R_{x_Kx_0} & R_{x_Kx_1} & \dots & R_{x_Kx_K} \end{bmatrix} \tag{5}$$

- Den slutgiltiga storleken av korrelationsmatrisen är $K * L \times K * L$ där K är antalet kanaler och L är antalet korrelationssamples.

Från ekvation 1 vill vi nu minimera uttrycket i nämnaren och detta får vi genom att lösa det generella eigenvektorproblemet enligt nedan

$$R_{nn} h\lambda = Ih \Leftrightarrow R_{nn}^{-1} Ih = h\lambda \tag{6}$$

Från ovanstående ekvation inses lätt att om vi löser ut den egenvektor med högst värde som uppfyller ekvationen får vi vårt optimala filter.

3 Konstruktion

3.1 Matlab - Teoretisk

Eftersom signalbehandling oundvikligen innebär många beräkningar som kan vara svåra att felsöka och verifiera bestämde vi oss för att skapa en fungerande modell i matlab innan vi började programmera signalprocessorn.
Som bas fanns till vårt förfogande data motsvarande signaler från fyra mikrofoner; en signalkälla och en störkälla samt en mixad källa innehållande kombinationen av de båda andra källorna. Vi beräknade vårt filter från den källa vi

ville släcka ut och applicerade det på den mixade signalen. För att undersöka resultaten tittade vi på spektrogram av utsignalen och lyssnade på den genom sound-funktionen i matlab.
Modellen var till en början väldigt simpel och visade sig snabbt vara otillräcklig, under projektets gång upptäcktes många problem med modellen som behövde åtgärdas. Eftersom vi saknade någon referens var det svårt att verifiera modellens resultat även när vi var relativt säkra på att den var korrekt. Små och stora fel gav till synes bra resultat dels på grund av vår bristfälliga förmåga att testa och verifiera våra resultat.

3.2 Signalprocessor - Praktisk

Implementationen är baserad på en signalprocessor ADSP-21262 med fyra kanaler input och stereo-ljud ut. Programmet som körs på signalprocessorn har en kalibreringsfas, en beräkningsfas och en filtreringsfas. I testutrustningen ingick även en 4-kanals mikrofon-array med förstärkare som användes för att generera insignalen och ett par vanliga hörlurar kopplade till utsignalen.
Under kalibreringsfasen beräknas korrelationsmatriser för insignalerna löpande över kalibreringstiden, i vårt fall 10 sekunder. När kalibreringsfasen är över kan beräkningsfasen påbörjas. Där inverteras korrelationsmatrisen och det största egenvärdet med tillhörande egenvektor beräknas.
Under filtreringsfasen delas den beräknade egenvektorn upp i lika många delar som vi har insignaler och appliceras som koefficienter i våra FIR-filter. Insignalerna filtreras var för sig och adderas sedan ihop för att skapa den färdiga utsignalen.
Beräkningarna som processorn utför verifierades steg för steg med hjälp av modellen och manuellt inmatad data tills det att vi kunde mata in samma data i matlab och få ut precis samma resultat. Till en början hade vi bara två mikrofoner inkopplade men programmet testades även med fyra.

4 Resultat

4.1 Test av modellen

Den teoretiska modellen producerar förväntat resultat med de statiska testdata vi har. Den källa som ska släckas ut försvinner från utsignalen medans den andra källan hörs tydligt. Analys av spektrogram visar också att de frekvenser som finns i den oönskade källan försvinner från utsignalen.

4.2 Test med signalprocessor

Vid test med riktig hårdvara och mjukvara på signalprocessorn kan entoniga störkällor släckas ut helt men det finns ingenting som tyder på att filtret baseras på vinkeln mellan mikrofon-array och störkälla. Algoritmen verkar istället producera ett filter som eliminerar källan oavsett var den befinner sig.
Eftersom den fysiska implementationen har verifierats mot modellen verkar det rimligt att anta att modellen också är felaktig, detta i sig är svårt att verifiera eftersom vi i modellen inte har någon möjlighet att direkt påverka källornas position.

4.3 Test med sinusformat filter

Genom att undersöka filterkoefficienterna direkt kunde vi konstatera att en sinusformad kalibreringssignal ger ett sinusformat filter om och endast om vi (1) inte inverterar korrelationsmatrisen och således konstruerar en beamformer som förstärker en viss källa istället för att släcka ut den eller (2) regulariserar korrelationsmatrisen genom att addera en identitetsmatris multiplicerad med en faktor av storleksordningen 10^4.
Ett sinusformat filter med lämplig förskjutning mellan kanalerna kan i våra tester släcka ut en störkälla i form av en matchande sinus-signal baserat på vinkeln mellan källan och mikrofonerna men systemet är extremt känsligt, området där störkällan är tyst är väldigt litet och minsta rörelse låter störsignalen ljuda igenom.

5 Diskussion

Efter projektets slut hade vi inget signifikant resultat som vi kunde utläsa något ifrån. Under projektets gång hade vi ett par gånger resultat som skulle kunna liknas vid en beamformer, det vill säga att vi kunde släcka ut en signal efter att ha utfört kalibreringen på en specifik position ifrån mikrofonerna. Dessa resultat visade sig sedan vara felaktiga eftersom vid upprepade tester kunde inte detta återskapas. Vår teori är att när störsignalen flyttades närmare mikrofonerna blev störsignalen så pass stark att den visade sig på mikrofonerna efter att störsignalen hade flyttats. Eftersom vi i början av projektet inte fick något förväntat resultat från vår *fullbandsbeamformer* kunde vi inte verifiera något utan agerade i blindo och besannade våra förväntningar. Om detta projekt ska utföras på nytt med liknande upplägg skulle en rekommenderad test-uppsättning med specificerade avstånd ifrån mikrofoner och ljudnivår på dessa vara att föredra.
Vi hade gärna sett ett bättre slutresulat men har trots detta lärt oss mycket. Under projektets gång har vi fått insyn i vad som är möjligt att utföra med digital signalbehandling och framför allt hur programmeringen av signalprocessorer går till.

Referenser

[1] `http://en.wikipedia.org/wiki/Beamforming`

[2] `http://www.eetimes.com/document.asp?doc_id=1278838`

A Matlabfunktioner

A.1 snirbf.m

```
function h = snirbf (S, N, L)
% SNIRBF SNIR beamformer filter design.
%  h = SNIRBF (S, N, L) calculates SNIR beamformer filters. The signal
%  matrices S and N are the input signals and noise signals of the
%  beamformer design. Each column in S and N correspond to a sensor
%  signal, and the number of columns is the number of sensors. The
%  beamformer filters are returned as columns of h, where each filter is L
%  taps.

Rxx = stcorrmtx(N, L);

[ReigV,ReigD] = eig(inv(Rxx));

[maxD,indexD] = max(diag(ReigD));

h = ReigV(1:end, indexD);

end
```

A.2 stcorrmtx.m

```
function Rxx = stcorrmtx (X, L)
% STCORRMTX Spatiotemporal correlation matrix .
%  rxy = STCORRMTX (X, L) calculates the spatiotemporal correlation
%  matrix of the signal matrix X. Each column in X correspond to a sensor
%  signal, and the number of columns is the number of sensors . The
%  spatiotemporal correlation matrix Rxx is a stacked matrix of
%  correlation matrices of L coefficients .
 N = size(X, 2);

 Rxx = zeros(N*L);

 for i=1:N
  for j=1:N
        Xi = corrmtx(X(:, i), L-1, 'covariance');
        Xj = corrmtx(X(:, j), L-1, 'covariance');
        Rxixj = Xj'*Xi;
```

```
            Rxx(L*(i-1)+1:L*i, L*(j-1)+1:L*j) = Rxixj;
    end
  end
end
```

A.3 eigenbeam.m

```
% EIGENBEAM eigenbeam filter used to test with matlab function 'sound'.
% Extracts filters from 'snirbf' and filters with signal X

function out = eigenbeam(X, S, N, L)

h = snirbf(S, N, L);


outL1 = filter(flipud(h(1:L)), 1, X(1:end,1));
outL2 = filter(flipud(h(L+1:L*2)), 1, X(1:end,2));
outL3 = filter(flipud(h(L*2+1:L*3)), 1, X(1:end,3));
outL4 = filter(flipud(h(L*3+1:L*4)), 1, X(1:end,4));

out = outL1 + outL2 + outL3 + outL4;

end
```