Vocoder

Ett projekt i Signaler och system

Handledare: Mathias Johansson
Grupp: IT3
Datum: 2002-12-10

Sammanfattning

En vocoder är en ljudeffekt som ofta används för att göra om en människoröst till en robotröst. I detta projekt har en sådan implementerats i MATLAB med gott resultat. Två försök gjordes även att tillverka en realtidsvocoder, dels på en hårdvaru-DSP, och även i MATLAB-verktygslådan Simulink. Inget av dessa slutfördes pga. diverse begränsningar.

Erik Andersson
811001-1437

Jonas Jämtberg
800705-7550

Mattias Seeman
770429-0076

Mathias Thore
810830-8597

Introduktion

En vocoder är en ljudeffekt, som tar en mänsklig röst som input. I vocodern behandlas rösten, för att på andra sidan komma ut i en mera robotliknande version. Det gamla syntbandet Kraftwerk använde denna effekt flitigt i väldigt många av sina låtar (t.ex. Autobahn, die Roboter osv.). Många musiker än idag fortsätter av olika anledningar att använda vocoder, kanske mest för att det låter tufft.

Bakgrund

En människoröst består av två delar; en bärvåg och en formant. Stämbanden genererar en bärvåg vars utseende är mer eller mindre konstant medan man pratar, så när som på tonhöjd. Formanten å andra sidan beskriver hur man med hjälp av sina talorgan dämpar eller framhäver vissa frekvenser i bärvågen, för att få fram olika ljud.

Tekniken utvecklades från början för att komprimera tal genom att minska signalens bandbredd dramatiskt och används idag flitigt inom t. ex. GSM.

Problemformulering

Projektets mål

Målet med projektet är att framställa en enkel vocoder. Förutsättningen är att det förmodligen går i Matlab, eftersom det tydligen ska finnas en del funktioner där som skulle vara till nytta. Ännu häftigare vore förstås att bygga en realtidsvocoder i hårdvara. Det finns även realtidsvocoders i form av dataprogram, men det kanske är ett svårt mål att uppnå i ett så här litet projekt.

Vad en vocoder verkar göra

Vocoderns jobb är att använda röstformanten för att modulera en annan bärvåg. Genom denna operation uppstår ett robotliknande tal. Vocodern tar alltså två insignaler. Dels en röstsignal, och dels en bärvåg som ska moduleras. Hur vocodern till sist låter beror alltså naturligtvis på hur den är byggd, men också på vilken bärvåg som skickas in.

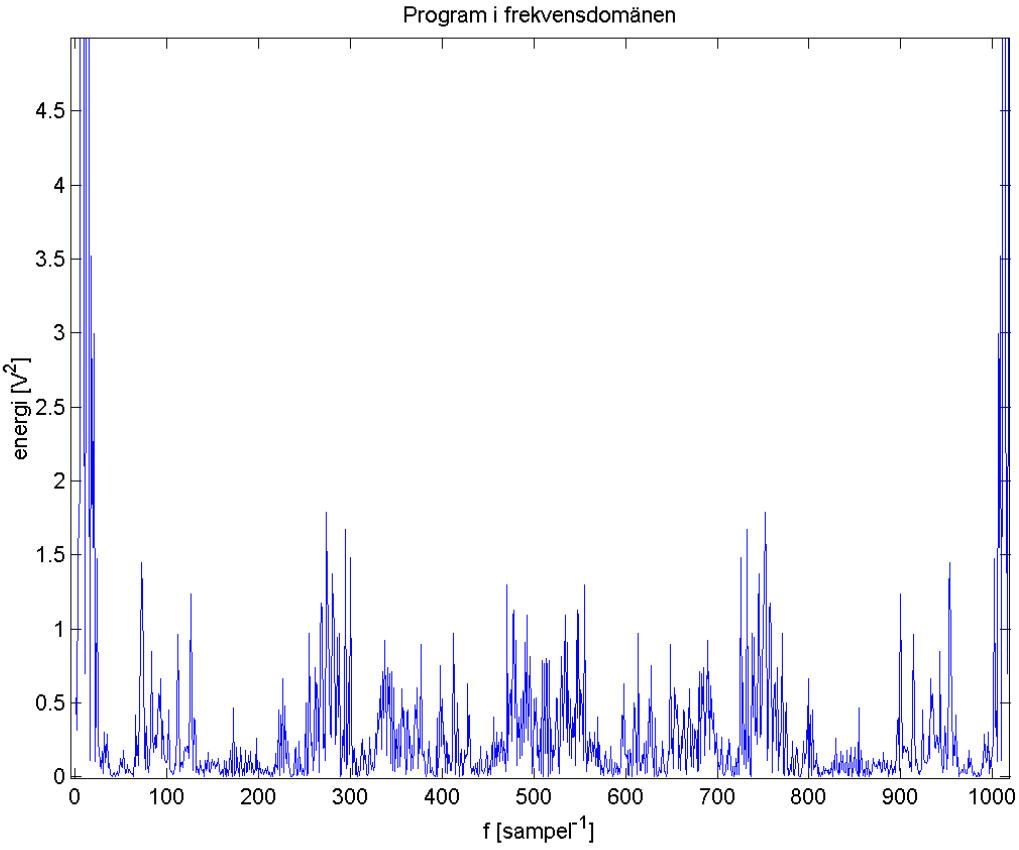
Vad vocodern verkligen gör

Röstsignalen delas upp i frekvensband (16–64 stycken verkar lämpligt) ur vilka den momentana formanten bestäms genom att ett medelvärde filter används på varje band. Bärvågen delas upp i lika många frekvensband som röstsignalen, och moduleras sedan med var sin del av formanten, för att slutligen mixas samman.

Lösning

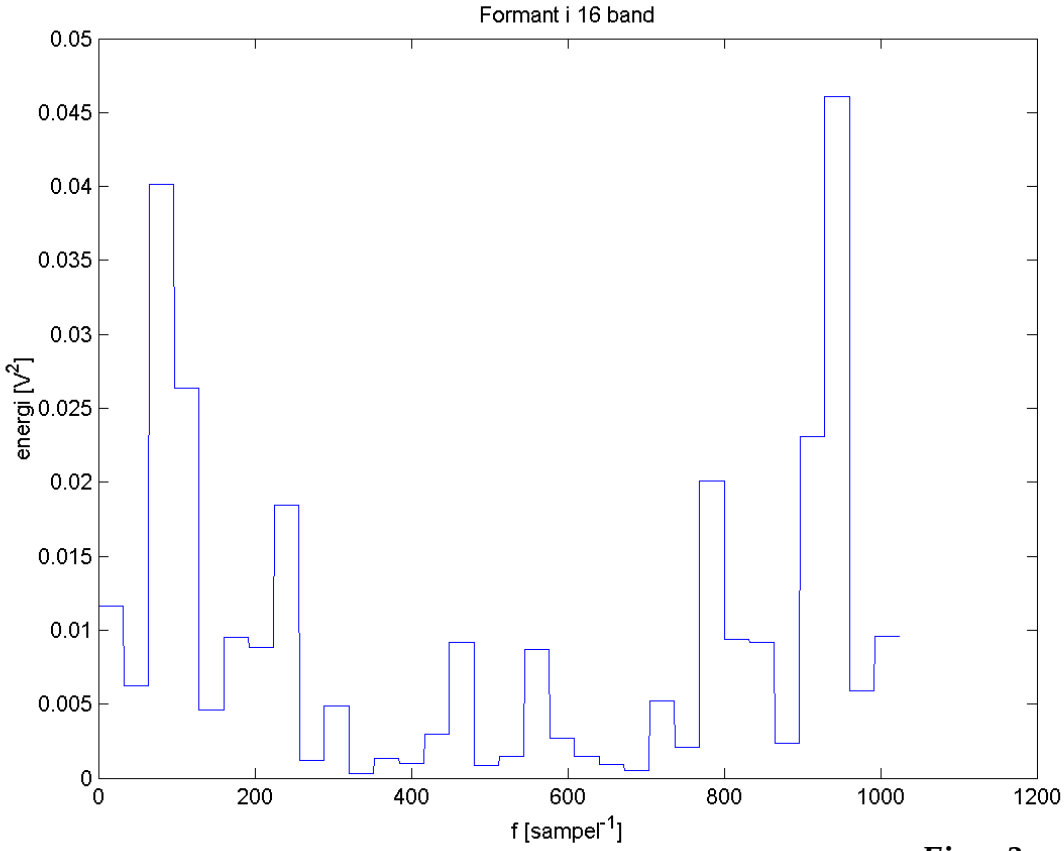
Det första som gjordes i MATLAB var att implementera en generell vocoderalgoritm. Inom gruppen fanns det nämligen två falanger, varav den ena ville använda fönstring, medan den andra inte var säker på att detta behövdes. Därför skrevs vocodern så pass allmän att fönstring enkelt skulle kunna implementeras senare om så behövdes. (se vocode.m)

Formanten togs fram genom att FFT:n av röstsamplen, programmet, beräknades. Se figur 1.



Figur 1.

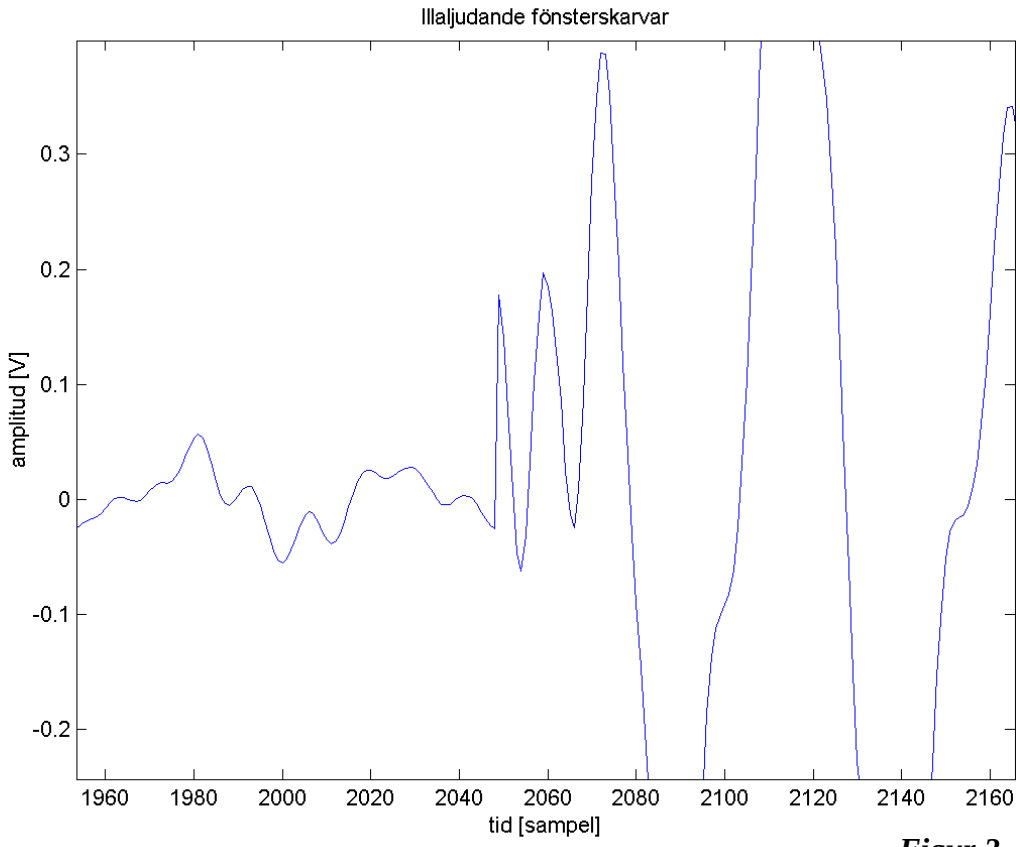
Därefter togs ett medelvärde för varje band fram. Ursprungsdata i hela bandet i formantmatrisen ersattes med detta medelvärde. På så vis ficks en diskretiserad, förenklad formant fram. Se figur 2. (se make_formant.m)



Figur 2.

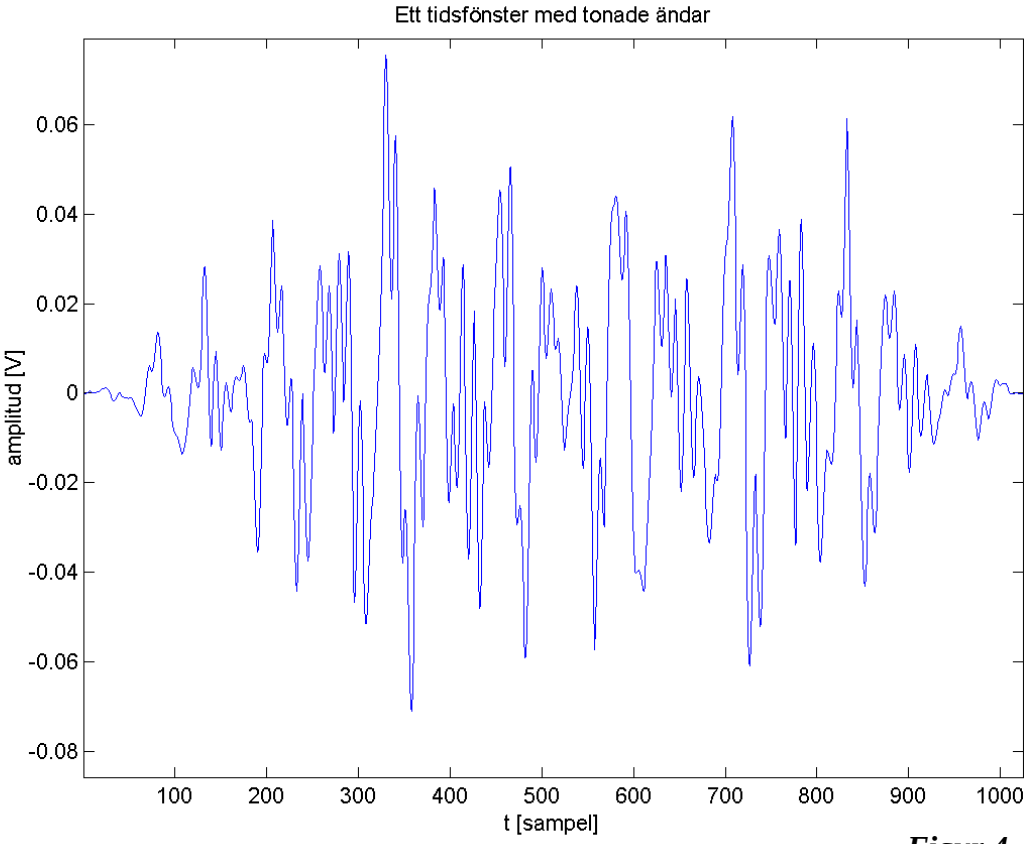
Sedan fouriertransformerades även bärvågen, varefter de två transformerna multiplicerades ihop elementvis. Det färdiga resultatet ficks sedan fram genom att inversfouriertransformera produkten.

Även om detta kanske fungerar i teorin, hördes inga välljudande resultat i hörlurarna. Enkel fönstringskod skrevs för att prova om det var där felet låg. (se window.m). Denna funktion delar upp ljudfilerna i fönster, som sedan vocodas var för sig. Genom detta kom vocodern att låta som just en vocoder. Det var dock inte ett fullständigt tillfredsställande ljud; i skarvarna mellan fönstren uppstod höga missljud på grund av diskontinuiteter. Se figur 3, där en skarv tydligt syns vid sampel 2048.



Figur 3.

Detta dög inte. Lösningar söktes i litteratur och på Internet, men inget konkret användbart kunde hittas. Lösningen visade sig vara att använda överlappande fönster, där överlappet tonades ut linjärt och mixades ihop additivt med nästföljande fönsters intonade överlapp. (se overlap25.m, fade_ends.m och figur 4). Detta gav tillräckligt välljudande resultat, och vocodern var färdig.



Figur 4.

Realtid

När väl offline-vocodern var implementerad i MATLAB och fungerade tillfredställande inleddes nästa fas av projektet; att implementera en fungerande realtidsvocoder.

Det första fruktlösa försöket gjordes med ett DSP-kort från Texas Instruments. Uppgiften dog i sin linda då inlärningströskeln för kodning av kortet visade sig vara alltför hög i relation till projektets tidsram.

Nästa idé till realtidsvocoder var att implementera den i MATLABs Simulink, ett ramverk för realtidsimulering. Verktyget är kraftfullt men tidsramen tillät oss inte att fullt ut utforska möjligheterna att slutföra fasen.

Resultat

Tyvärr resulterade projektet inte i någon fungerande realtidsvocoder men den framställda offlineversionen gav över förväntan välljudande resultat.

Slutsatser

Vi har hittat ett fint sätt att modifiera samplingar och för att skapa jättehäftiga ljud. Med en större tidsram hade en hårdvarubaserad version förmodligen kunnat byggas ihop.

Vad kan man göra mer

För att få till fina effekter i en vocoder kan man som sagt prova med olika bärvågor. Men man kan även mixtra med formanten på olika sätt. Exempelvis kan fler/färre frekvensband användas eller formanten blandas om, inverteras osv. Vocodern kan även fås att ”sjunga” genom att ändra bärvågens tonhöjd under pågående vocodning.

Referenser

http://www.sirlab.de/linux/descr_vocoder.html

Appendix

vocode.m

```
function x = vocode(formant,carrier,bands)

s = min(max(size(formant)), max(size(carrier)));

f = fft(formant);
c = fft(carrier);

f = make_formant(f,bands,s)';

%c = real(c) .* abs(f) + i*imag(c).*abs(f);
c = c .* f;

x = real(ifft(c));
```

make_formant.m

```
function x = make_formant(formant,dbands,total_freqs)
dbands = dbands * 2;

freqs_per_band = total_freqs / dbands;

x = zeros(1,total_freqs);

for i = 1:freqs_per_band:total_freqs,
      x(i:i+freqs_per_band-1) = mean(formant(i:i+freqs_per_band-1));
end
```

window.m

```
function g = window(formant, carrier, no_bands, samples_per_window)

s = size(formant);
s = s(1);

rest = mod(s, samples_per_window);

g(s) = 0;

for i = 1:samples_per_window:s-rest,
     g(i:i+samples_per_window - 1) = vocode(formant(i:i+samples_per_window -
       1), carrier(i:i+samples_per_window - 1), no_bands);
end
```

overlap25.m

```
function g = overlap25(formant, carrier, no_bands, samples_per_window)

s = min(max(size(formant)), max(size(carrier)));

rest = mod(s, samples_per_window);
qwin = samples_per_window / 4;
g(s+qwin) = 0;
%samples_per_window


for i = qwin+1:samples_per_window:s-rest-4 * qwin,

      w = samples_per_window +qwin – 1;
      win = vocode(formant(i-qwin:i+w), carrier(i-qwin:i+w), no_bands);
      win=fade_ends(win);
      g(i-qwin:i+w) = g(i-qwin:i+w) + win(1:samples_per_window*1.5);
end

g = g / max(abs(g));
```

fade_ends.m

```
function amp = fade_ends(x)

len = max(size(x));

qlen = len / 4;

amp = ones(1,len);

amp(1:qlen) = [0:qlen-1]/qlen;

amp(3*qlen+1:len) = 1 - [0:qlen-1]/qlen;

amp = amp .* x';
```