ETIN80 - Speech Recognition

Jens Nilsson

1 Introduction

Speech recognition is something that recognizes speech or words, this can be used for various things example turning on a light with the use of your voice. The aim of this project is to find an algorithm and implement it on a DSP for exactly this use.

2 Theory

When making a speech recognition system a learning part and a recognition part is needed. The learning part consists of a speech analyzer which calculates the reflection coefficients. Then doing a vector quantization of these reflection coefficients. "Training" of the system is done to learn specific words, making a databank with these words. The recognition part consists of the same analyzer and then comparing and matching these reflection coefficients to the vector quantization done in the learning part, see figure 4. When matching the words with the databank a state machine is used.

2.1 Learning part

2.1.1 Speech analyzer

The speech analyzer consists of a pre-emphasizer, signal framing and vector calculation see figure 1.

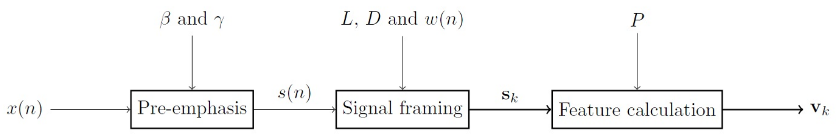

Figure 1: Speech Analyzer

Pre-emphasis: The purpose of the pre-emphasis is to remove negative effects of noise, making the in signal $x(n)$ less noisy. The filter is a first order system:

$$H_{pre}(z) = \beta(1 - \gamma z^{-1}) \quad \text{where } |\gamma| < 1$$

Signalframing: The signal framing part divides the input signal $s(n)$ into frames. The size of the frame depends on how big L is set. L is calculated according to: $L = blocklength/f_s$. Where *blocklength* is the length of a frame in milli seconds, f_s is the sampling frequency of the input signal. In other words L is the amount of samples you will have in each frame. D is how much overlap each frame L has. $w(n)$ is the window function to multiply each frame with, where the window Hanning is used. This procedure is then done for all frames of the signal $s(n)$, creating a vector $\mathbf{s_k}$.

Vector calculation (calculating the reflection coefficients): For each frame that is windowed an autocorrelation is done. Then the first P samples is chosen from the autocorrelation that will be used for calculating the reflection coefficients. To calculate the reflection coefficients the Schur recursion is used which gives the coefficients in a more efficient way than the Levinson Durbin algorithm. After iterated over all frames a vector $\mathbf{v}$ with k blocks will be made, this can be seen as a Pxk matrix, see figure 2.

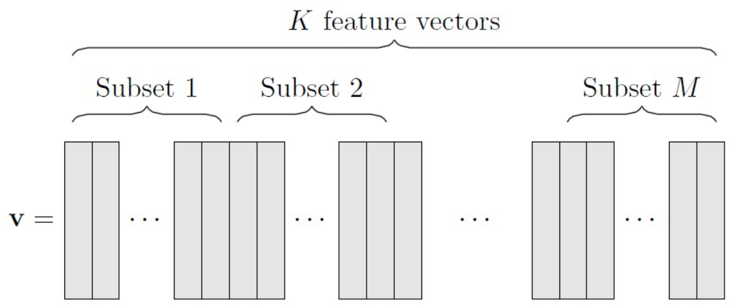

Figure 2: Reflection coefficients divided into subset

2.1.2 Vector quantization (making the databank)

When making the databank a quantization of the vector $\mathbf{v}$ must be done according to figure 2. How many blocks that will be in each subset are chosen by M. When calculating the subsets each row with all columns in that subset is summed and divided by the number of columns in that subset. This is done for all rows and columns for each subset, for all subset, see figure 3.

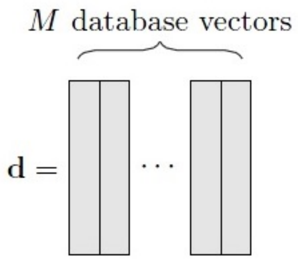

Figure 3: Database vector with M blocks (M columns)

2.2 Recognition part

This part uses the same analysis stage as described above and a state machine to evaluate the input signal with the database created, see figure 4.

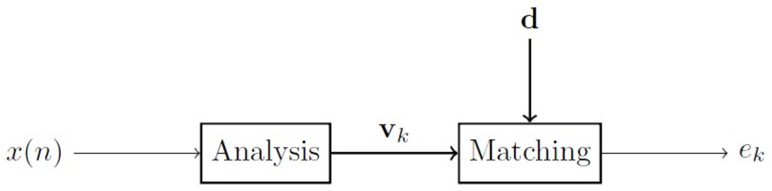

Figure 4: Recognition part

The state machine works as follow:

$$d_{k,s} = |\mathbf{v_k} - \mathbf{d_s}|$$
$$d_{k,s+1} = |\mathbf{v_k} - \mathbf{d_{s+1}}|$$

Where the state machine compares the incoming vector $\mathbf{v_k}$ with the database vector $\mathbf{d_s}$ where s is a block according to figure 3. The state of the state machine denoted by s and whose is initialy set to $s = 1$, determines which block in the database is currently being tracked. For each feature vector from the analysis stage, it is decided whether the state machine should remain in its current state or jump to the state $s + 1$, see equation bellow:

$$s = \begin{cases} s & d_{k,s} \leq d_{k,s+1} \\ s+1 & \text{otherwise} \end{cases}$$

The state machine is running until either the end of the signal is reached or until the state machine enters the state $s = M$. The error between each sequence is calculated as:

$$e_k = e_{k-1} + \min(d_{k,s}, d_{k,s+1})$$

Where the initial error is $e_0 = 0$. The final error is the result from the state machine and if the error is small enough according to what threshold is given it will be word that is in the databank.

3 Implementation

3.1 Matlab implementation

When initiating the implementation these constants where used:

$Beta = 1$
$Gamma = 0.95$
$f_s = 16000\text{Hz}$
$Blocklength = 20ms$
$L = blocklength/f_s$
$D = L/2$
Window function = *hanning*
$P = 10$
$M = 8$

When creating the databank and when recording the signal that would be compared to the databank an interval of 1.5 seconds was made for recording the speech. When implementing the theory in to Matlab, I first tried to do the Schur recursion according to the pseudo code on page 245 in the book "Statistical Digital Signal Processing and Modeling" written by Monson H. Hayes. The problem I got when doing this was that the reflection coefficients were not stable, i.e some coefficients were bigger or smaller than 1 respectively -1. I later used Matlabs own function "schurrc" instead.

When implementing the database I needed to get the same amount of vectors in each subset, this was done by taking the floor function of the number of columns in the vector **v** divided by M. When doing this the last samples gets discarded, but since usually the last samples contain background noise this discarding was ok.

One problem with the state machine was that it often found matches pretty early of the signal and then not searching through the whole vector $\mathbf{v_k}$ which led to bad results. To solve this problem another method in Matlab was made to instead compare the database vectors to each other instead of using the state machine, i.e $norm(d - d')$ where d is the databank and d' is the signal to compare. This worked better but was still not that stable and the result could vary much.

Since I made a recording of 1.5 seconds a lot of background noise would be before and after the word to be processed. To get rid of these noise two methods was done in Matlab. When using these two methods before sending the signal to the analysis section made a huge difference and finally some decent results were made.

3.2 VisualDSP++ implementation

The pros with Matlab is that you can store the whole signal and then process it. Since this isent possible in VisualDSP, an real-time method is used instead, that is treating each frame directly instead of taking each frame from an array of the full signal. The Matlab code was translated to C-code and no changes was done in the code of the learning part and recognition part (except using my own method which compared databanks instead of the state machine), except that you process each frame instead of the whole signal. The part that differed from Matlab is how to manage the input signal in the correct way and setting a threshold and a time length for the input signal.

4 Result

A databank of the word “Alfa” was made with three different recordings in matlab, these where then hardcoded into the C-code for comparison of the signal that should be compared. To get a small error i.e better result when using the DSP I’ve noticed that you have to talk aggressive/loud yet clear. Since the amplitude of the signal shoudn’t be a factor when calculating the reflection coefficients, the only reason for this is that if you talk aggressive you will drown out the backround noise. In figure 5 you can see the input signal of the word “Alfa”. The signal is pretty loud but not clipping when listening to it.

A second databank of the word “Freeze” was also made. In matlab this worked good and a low error could be made and matlab could distinguish between the word “Alfa” and “Freeze” and neglect other random words. But when using the Freeze databank with the DSP it didnt work. I dont know why but the error was to high when using the DSP. In figure 6 you can see the input signal for “Freeze”, and my guesses are that either the word “Freeze” contains to much noise like sounds or the input signal is to weak compared to the input of “Alfa” and not drowning out background noise.

An attempt to use the word “Bravo” will be done to see if this databank will work better with the DSP.

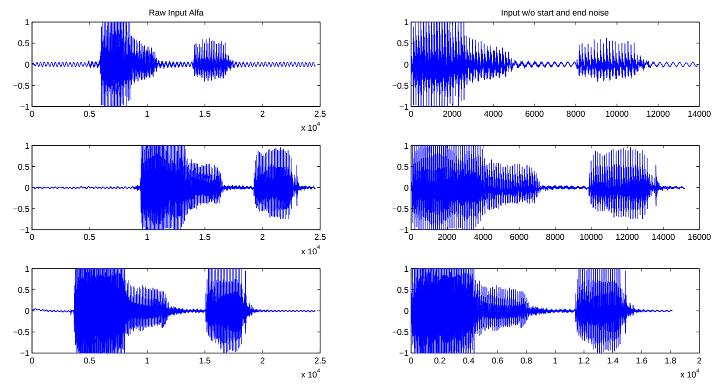

Figure 5: Recorded input signal of the word "Alfa"

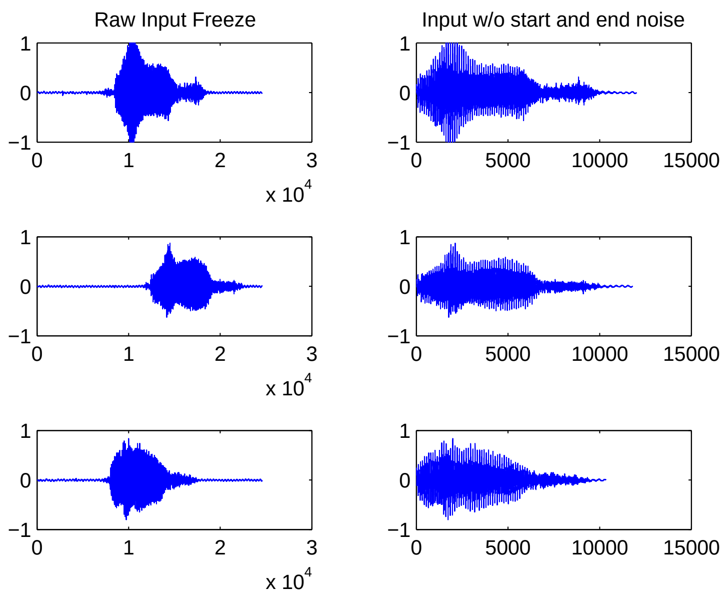

Figure 6: Recorded input signal of the word "Freeze"

5 Conclusions

After doing this project I've learned that the cleaner input signal you have the better result you will get. To make the result even better some form of noise reduction would be appropriate to implement, but since I was alone on this project there was too little time. The state machine method maybe would have worked after I removed all noise from the beginning and end of the input signal, but since the method with just comparing the databases to each other worked good I stuck to that solution since it was less to translate to C-code, time was a factor in this project when working alone.

6 References

Mikael Swartling, PhD. Department of EIT, Faculty of Engineering (LTH), Lund University.

Monson H. Hayes: Statistical Digital Signal Processing and Modeling.

7 Matlab code

7.1 Record Input Signal

```
close all;
clear all;
%%
%%%%%RECORD A INPUT AND SAVE AS DOUBLE ARRAY%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vid inspelning av ljudfil
r = audiorecorder(16000, 16, 1);
record(r);     % speak into microphone...
pause(1.5)
stop(r);
p = play(r);   % listen to complete recording
alfabank1= getaudiodata(r, 'double'); % get data as double array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

7.2 Learning (Run method)

```
%
% THIS M-FILE DOES THE SIGNAL FRAMING, ANALYSIS STAGE, CREATES THE DATABA
%
%

close all;
clear all;

%%
%Initiation of constants
gamma= 0.95; %Pre-emphasis
fs = 16000; %Samplingsfrequence
time = 1/fs;
blocklength = 20*10.^(-3); % blocklength 20 ms
L = blocklength/time; % 320 block at 20 ms
D= L/2; %D= L/2; %overlap
win = 'hanning'; %window function
P=10; %prediction
M=8;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Creates database for alfa
load alfabank1;
%sound(alfabank1, fs)
figure
title_a = subplot(3,2,1);
```

```
plot(alfabank1);

alfabank1 = remove_startNoise(alfabank1);
alfabank1 = remove_endNoise(alfabank1);

title_b = subplot(3,2,2);
plot(alfabank1);

v = Analysis(alfabank1, L, D, win , gamma , P);
d_alfabank1 = Create_Database(v,M);
clear v;

title(title_a,'Raw Input Alfa');
title(title_b,'Input w/o start and end noise');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Creates database for alfa2
load alfabank2;
%sound(alfabank2, fs)
subplot(3,2,3);
plot(alfabank2);

alfabank2 = remove_startNoise(alfabank2);
alfabank2 = remove_endNoise(alfabank2);

subplot(3,2,4);
plot(alfabank2);

v = Analysis(alfabank2, L, D, win , gamma , P);
d_alfabank2 = Create_Database(v,M);
clear v;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Creates database for alfa3
load alfabank3;
%sound(alfabank3, fs)

subplot(3,2,5);
plot(alfabank3);

alfabank3 = remove_startNoise(alfabank3);
alfabank3 = remove_endNoise(alfabank3);

subplot(3,2,6);
plot(alfabank3);

v = Analysis(alfabank3, L, D, win , gamma , P);
```

```
d_alfabank3 = Create_Database(v,M);
clear v;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Creates database for freeze
load freezebank1;
%sound(freezebank1, fs)
figure
title_c = subplot(3,2,1);
plot(freezebank1);

freezebank1 = remove_startNoise(freezebank1);
freezebank1 = remove_endNoise(freezebank1);

title_d = subplot(3,2,2);
plot(freezebank1);

v = Analysis(freezebank1, L, D, win , gamma , P);
d_freezebank1 = Create_Database(v,M);
clear v;

title(title_c,'Raw Input Freeze');
title(title_d,'Input w/o start and end noise');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Creates database for freeze2
load freezebank2;
%sound(freezebank2, fs)
subplot(3,2,3);
plot(freezebank2);

freezebank2 = remove_startNoise(freezebank2);
freezebank2 = remove_endNoise(freezebank2);

subplot(3,2,4);
plot(freezebank2);

v = Analysis(freezebank2, L, D, win , gamma , P);
d_freezebank2 = Create_Database(v,M);
clear v;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Creates database for freeze3
load freezebank3;
%sound(freezebank3, fs)

subplot(3,2,5);
plot(freezebank3);
```

```
freezebank3 = remove_startNoise(freezebank3);
freezebank3 = remove_endNoise(freezebank3);

subplot(3,2,6);
plot(freezebank3);

v = Analysis(freezebank3, L, D, win , gamma , P);
d_freezebank3 = Create_Database(v,M);
clear v;
```

7.3 Recognition (Run method)

```
% This m-file does the recognition and compares the "SoundToCompare" with
% the databank and writes the result

close all;

a=1;
while(a==1)
    %%%%%RECORD A SOUND%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%Vid inspelning av ljudfil
    r = audiorecorder(16000, 16, 1);
    record(r);     % speak into microphone...
    pause(1.5)
    stop(r);
    p = play(r);   % listen to complete recording
    SoundToCompare = getaudiodata(r, 'double'); % get data as double arra
    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    subplot(2,1,1)
    plot(SoundToCompare) %Plot of the signal recorded

    SoundToCompare = remove_startNoise(SoundToCompare);
    SoundToCompare = remove_endNoise(SoundToCompare);
    %SoundToCompare =SoundToCompare';

    subplot(2,1,2)
    plot(SoundToCompare) %Plot of the signal recorded without noise

    % Analysis stage
    clear v;
    v = Analysis(SoundToCompare, L, D, win , gamma , P);

    %Create the Create_Database
    dprime = Create_Database(v,M);

    % Compare to the databank "Alfa"
    score1 = Compare_Databank(d_alfabank1,dprime);
    score2 = Compare_Databank(d_alfabank2,dprime);
    score3 = Compare_Databank(d_alfabank3,dprime);

    % Compare to the databank "Freeze"
    score4 = Compare_Databank(d_freezebank1,dprime);
    score5 = Compare_Databank(d_freezebank2,dprime);
    score6 = Compare_Databank(d_freezebank3,dprime);

    %Calculate mean value
    score_medel_alfa = (score1+score2+score3)/3
    score_medel_freeze = (score4+score5+score6)/3

    %Check if the input is below the limit
    if score_medel_alfa <0.70 %Limit at 0.70
        a=0;
        ALFA = score_medel_alfa
        break
    end

    if score_medel_freeze <0.70 %Limit at 0.70
        a=0;
        FREEZE = score_medel_freeze
        break
    end

    % This part is a heads up for the next recording
    % (next iteration of the while-loop) if not under the limit value.
    pause(3)
    Two = 'Two'
    pause(1)
    One = 'One'
    pause(1)
    Record = 'Record'
end
```

7.4 Analysis

```
function v = analysis (x, L, D, win , gamma , P)
% ANALYSIS Analysis stage of speech recognition .
% v = ANALYSIS (x, L, D, window , gamma , P) returns the feature vector o
% the signal x. The framing and windowing is controlled by L, D and
% window , and gamma specifies the pre - emphasize filter . The feature
% vectors from the analysis stage are the reflection coefficients of the
% prediction filter of order P for each signal frame . v(:, k) is the
% feature vector for signal frame k.

s = filter([1 -gamma],1,x);
stepsize = L-D;
numberOfFrames = length(s)/stepsize-1;

%Initiation of the first 320 blocks
samp1 = 1;
samp2 = L;

for i =1:numberOfFrames
    window = hanning(L);
    s(samp1:samp2);
    s_k = window.*(s(samp1:samp2)');
    rxx = xcorr(s_k,'biased');

    %prediction of order 10
    g=0;
    for e=1:P+1 %I must have 11 here to get 10 rows of refl.coeff. schurr
        g(e) = rxx(e+L-1);
    end
    g;
    K = schurrc(g); % K is refl.coef
    v(:,i)=K; %v(:,i)=K;
    samp1 = samp1 + stepsize;
    samp2 = samp2 + stepsize;
end

v;
```

7.5 Create Database

```
function d = Create_Database (v, M)
% DATABASE Converts feature vectors into a database for speech recognitio
% d = DATABASE (v, M) converts the feature vectors v into a database by
% grouping the feature vectors into M adjacent subsets . d(:, m) is the
% database vector for the m:th feature vector subset .

[rader koloner] = size(v);
d_koloner = floor(koloner/M);
```

```
sampel1 = 1;
sampel2 = d_koloner;
i= 1;
for m = 1:d_koloner*M
    if m == sampel2
        for q = 1:rader
            d(q,i) = 1/d_koloner * sum(v(q,sampel1:sampel2))';
        end
        i=i+1;
        sampel1 = sampel2+1;
        sampel2 = sampel2 + d_koloner;
    end
end
d;
```

7.6 Compare Databank

```
function c = Compare_Databank(d, dprime)
% compareDatabank Matches an analyzed signal with a database for speech r
% c = compareDatabank(d, dprime) evaluates the analyzed signal d for the
% speech recognition .

[rader, koloner] = size(dprime);
c=0;
for k = 1:koloner
    d_k(k)= norm(d(:,k) - dprime(:,k));
    c=c + d_k(k);

end

c=c/koloner;
```

7.7 Remove Start Noise

```
function output = remove_startNoise(input)
% remove_startNoise discard the noise in the beginning of the signal
%
%
index = 1;
for i = 1:length(input)
    if input(i) > 0.1 % Limit for the noise. The magnitud of the noise wi
        temp = input;
        input = 0;
        for n = i:length(temp)
            output(index) = temp(n);
            index = index + 1;
        end
        clear temp;
        break
    end
end
```

7.8 Remove End Noise

```
function output = remove_endNoise (input)
% Discards the end noise of the signal by checking the mean value for the
% 10 last indexes of the signal
%
stepsize= 10;
samp1 =1;
samp2 = stepsize;
temp = 0;
for i= 1:length(input)
    output(i) = input(i);
    for n = samp1:samp2
        if n > length(input)%If the signal does not contain noise in the
                            %then add zero to the end of the array
            input(n)=0;
        end
        temp = temp + abs(input(n));
    end
    temp = temp/stepsize;%Mean value of the noise
    if temp < 0.001% Noise level. Best with limit at 0.001?
        break
    end
    samp1 = samp1+1;
    samp2 = samp2+1;
end
```

8 C code

To be included, code needs cleaning first.