Lund University of Technology
ETIN80 - Algorithms in Signal Processors
2018-03-10

Speech Synthesis

Group 5
Krister Lestander
Xin Zhou
Isabelle Zilling

Supervisor
Mikael Swartling

Abstract

The purpose of the report was to implement speech synthesis with an ADSP-21262 SHARC Processor. A recorded speech signal was synthesised through three different ways; one with noise, another with a constant time interval and a third with a calculated time interval. The three different modified signals made the in signal in terms of speech sound different with a robotic feeling for the listener.

Table of contents

1. Introduction

The following introduction brings up the purpose, background and structure of the report.

1.1 Purpose

The purpose of the project is to implement speech synthesis with an ADSP-21262 SHARC Processor. A Digital Signal Processor (DSP) processes speech in real time with limited memory and the performance is affected by the condition of hardware structure. Before implementing the speech synthesis in DSP, an algorithm was designed and a recorded voice band in Matlab was simulated.

1.2 Background

The human speech depends on the frequency of the cords vibration and is in many ways essential for communication. Even though the words bring forth key information, the pitch in the voice is also important to interpret the message. For example the message will be perceived differently depending on the speakers age, gender and health condition (Swanson, D. 2000). Human speech sound can be classified to voiced sounds and fricative sounds. The voice sound is extracted by the lungs and associated with muscles through vocal cords. The fricative sounds act as random noise and occurs when air flow is mostly blocked by the tongue, lips and teeth. These include sounds like f, s, th, z, v, sh (Smith S. 1997).

1.3 Structure of the report

The report consists of the theory part in which relevant theory of the project is brought up and explained. Then comes the method section describing the approach to solve the task. In the result section the findings are presented. The problems that emerged during the project are described in the summary together with the learning outcomes and the final result.

2. Theory

Several theories are relevant for this project. Block processing, Wiener filter, FIR and IIR filter, Error signal, Levinson-Durbin algorithm and Autocorrelation are all essential to process speech synthesis by programming. These theories are explained further in this section to deepen the understanding of the process.

2.1 Block Processing

Figure 1 illustrates block processing. By block processing the system is processed in blocks instead of individual samples. The system is waiting for *N* samples before processing and after that all the *N* samples are processed at the same time (Swartling M. 2018). The main reason for block processing is that the speech can be seen as stationary for 20 ms. While the sample rate had a value of 16 000 Hz, the block size was calculated to 320. Since it is necessary to work with 320 samples it is only logic to process everything in blocks of 320. To use block processing instead of processing each individual sample can make the algorithm more effective.

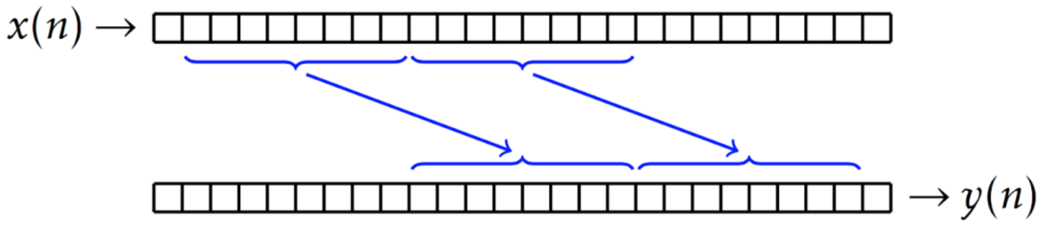

Figure 1. The figure illustrates an example of block processing where eight samples are processed at the same time (Swartling M. 2018).

2.2 Filter

In order to filter a signal both FIR and IIR filter can be used. However, as described below, these filters possess slightly different qualities. In our model the FIR filter is an important component of the Wiener filter, see *figure 2.*

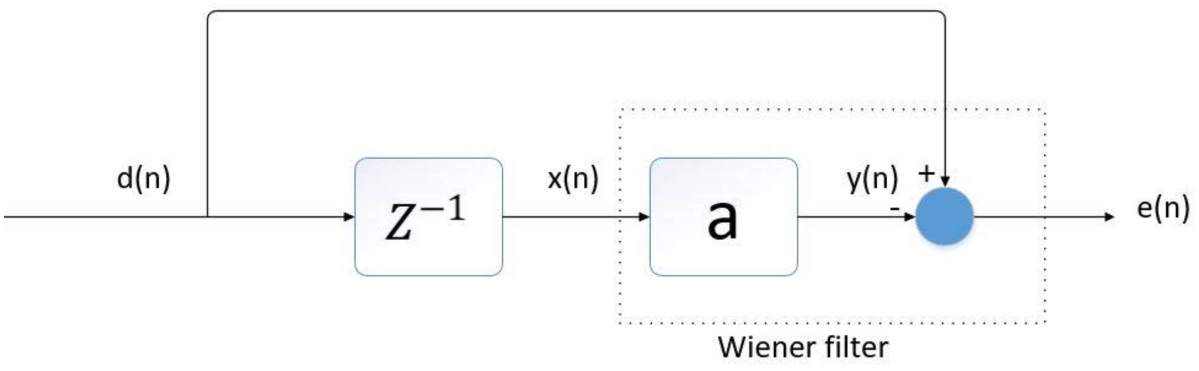

Figure 2. In the block diagram the Wiener filter is illustrated (Swartling M. 2018).

2.2.1 Wiener Filter

The Wiener filter is a linear optimum discrete-time filter. Consider the filter in *figure 2*, the input of the Wiener filter is a delayed signal. At some discrete time, the filter produces an output denoted by *y(n)*. The output is used to provide an estimate of a desired response *d(n)*. The estimation error *e(n)* is defined as the difference between the desired response *d(n)* and the filter output *y(n)*(Haykin S. 2002).

2.2.2 FIR Filter

FIR filter stands for *Finite Impulse Response* filter and it has a finite-duration impulse response. Outside of some finite time interval the impulse response for an FIR system is zero. *Figure 3* illustrates an FIR system (Proakis J et.al. 2014).

The formula for an FIR filter is the following (Proakis J et.al. 2014);

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k)$$

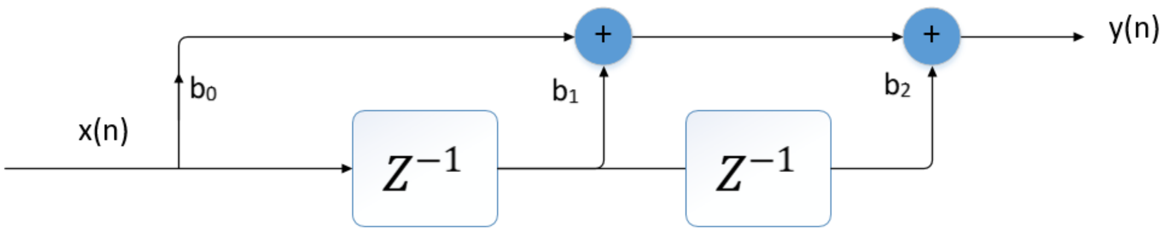

Figure 3. Realization of an FIR system of order n (Proakis J et.al. 2014).

2.2.3 IIR Filter

IIR filter stands for *Infinite Impulse Response* filter and has an infinite-duration impulse response. The difference between the FIR and IIR filter is that the IIR filter's coefficients is the inverse of the coefficients of the FIR filter (Proakis J et.al 2014).

The formula for an IIR filter is the following (Swartling M et. al. 2017);

$$y(n) = \sum_{k=0}^{M} h(k)x(n-k) - \sum_{k=1}^{N} a(k)y(n-k)$$

2.3 Error Signal

The error signal *e(n)* is illustrated in the block diagram in *figure 2* and is the difference between the input and output signal. The formula for the error signal is given by;

$$e(n) = d(n) - y(n)$$

Since the output, *y(n),* of the FIR filter is (see section 2.2.2 FIR Filter)

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k)$$

the formula for the error signal can be rewritten as;

$$e(n) = d(n) - \sum_{k=0}^{M-1} h(k)x(n-k)$$

2.4 Autocorrelation

Autocorrelation is the correlation between the input signal *x(n)* and the delayed input signal *x(n-l)*. The sequence for autocorrelation can be written as the following, where *l* is the delay (Proakis J et.al. 2014);

$$r_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n-l)x(n) = x(k) * x(-k)$$

2.5 Levinson-Durbin Algorithm

To get the coefficients that is needed for the Wiener filter the Levinson-Durbin Algorithm is used. The Levinson-Durbin algorithm is a computationally efficient algorithm to solve equations that are needed for the prediction coefficients. The calculation is made recursively with a Toeplitz matrix (Proakis J et.al. 2014).

$$\begin{bmatrix} y_{xx}(0) & y_{xx}(-1) & \cdots & y_{xx}(-p) \\ y_{xx}(1) & y_{xx}(0) & \cdots & y_{xx}(-p+1) \\ \vdots & \vdots & \cdots & \vdots \\ y_{xx}(p) & y_{xx}(p-1) & \cdots & y_{xx}(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} \sigma_\omega^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Figure 4. The Toeplitz matrix express the Yule-Walker equations and can be inverted by the Levinson-Durbin algorithm (Proakis J et.al. 2014).

Levinson-Durbin algorithm is presented for solving a general set of linear symmetric Toeplitz equation $\mathbf{R}_p\mathbf{a}_p=\boldsymbol{\varepsilon}_p\mathbf{u}_1$. The complete Levinson-Durbin recursion is as follows (Hayes M.1996):

1. Initialize the recursion

$$a_0(0) = 1$$
$$\varepsilon_0 = r_x(0)$$

2. For j=0, 1, …, p-1

$$\gamma_j = r_x(j+1) + \sum_{i=1}^{j} a_j(i) r_x(j-i+1)$$

$$\Gamma_{j+1} = -\frac{\gamma_j}{\varepsilon_j}$$

For i = 1, 2,…, j

$$a_{j+1}(i) = a_j(i) + \Gamma_{j+1} a_j(j-i+1)$$
$$a_{j+1}(j+1) = \Gamma_{j+1}$$
$$\varepsilon_j = \varepsilon_j [1 - |\Gamma|^2]$$

3. Method

First the necessary theory was read for the given task, which included the Wiener filter, Levinson-Durbin algorithm and autocorrelation.

To make the implementation of the DSP easier later on in the project, MatLab was used to simulate the process that can be seen in *figure* 5. This led to a deeper understanding for how the signal should be filtered to create speech synthesis.

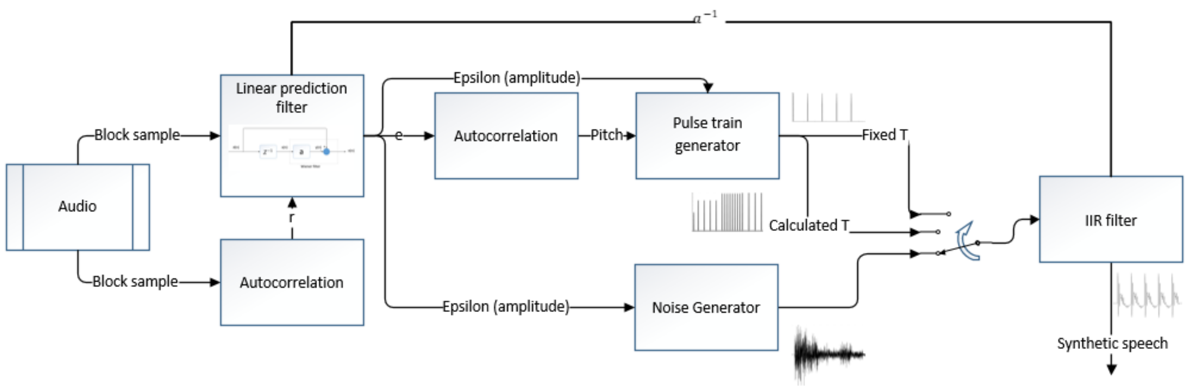

Figure 5. Flow chart for transforming input speech to synthetic speech.

The speech signal was recorded and loaded into Matlab. A notch filter was then used as a pre-filter to dampen the signal between two defined frequency values. Finally the rest of the frequencies were let through before the signal was split into blocks. The block size is defined as the multiplication of sample rate and stationary time. 20ms can be considered to be stationary for speech. The sample rate is 16000 Hz measured by Matlab. Thus, the block size is 20ms*16000Hz=320. The signal is separated into blocks with block size 320 and each block was sent to the created process function.

In the process function, to generate Wiener filter, the cross correlation function was used to compute the autocorrelation of the input signal to later be used in Levinson-Durbin algorithm. The Levinson-Durbin function generates the coefficient for FIR filter and error. The output from FIR filter that is the error signal is then used to get the pitch witch can get from the cross correlations function of the error signal. The interval **T** of the pulse train is calculated from the position of the highest and the second highest pitch. The amplitude is the square root of the error that was calculated by the Levinson-Durbin function. An IIR filter was then used to transform the processed signal either by fixed T or calculated T or by a random noise.
When a recorded speech was processed, a self-Levinson function was developed in Matlab. The main reason for creating self-made Levinson functions was because of the fact that there would not be any build-in functions to use later in the project when the code would be translated to C code in the DSP. The same thing was done for the cross correction.

After the recorded speech was simulated successfully in Matlab, the DSP was connected. The Matlab code was translated into C code in Analog Devices development environment. The structure of the program is same with Matlab, and Levinson-Durbin function, cross correction and filters are given in Matlab, but in C program the functions were made manually according to the theory and formulas. For different performances of speech synthesis, the four buttons on the DSP-card were used, see *table 1*. Button 1 was used to hear the speech with noise filtering. The performance of button 2 was robotic voice with a calculated pulse train interval T as the input in the IIR filter. Button 3 will produce robotic voice with fixed pulse train interval T for input in IIR filter. The final button shows the original speech.

To get the perfect result with measured T and fixed T some testing on filter order and scaling on the amplitude and T needed to be done in the code for the DSP.

Table 1. Buttons on the DSP with respectively out signal

Button	**Out signal calculated with;**
1	Noise
2	Calculated T
3	Fixed T
4	Original signal

4. Result

In *figure 6* a speech signal can be seen which comes from a recorded voice signal in Matlab.

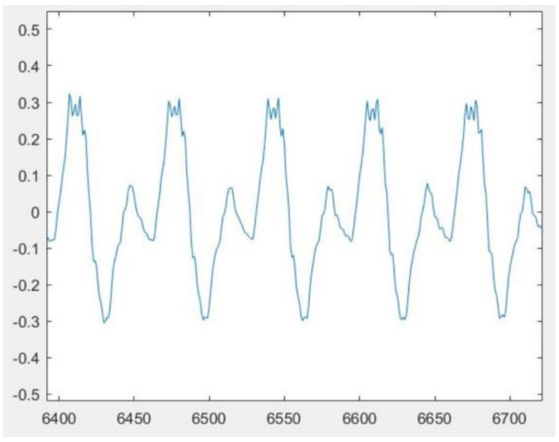

Figure 6. The figure illustrates a speech signal. The vertical axis shows the amplitude and the horizontal axis shows the time in milliseconds.

The result that can be presented is the one from Matlab and can be used to get an understanding on how different approaches is working when later implemented in the DSP. In *figure 7* the calculated T can be seen and when compared with the measured T that is seen in *figure 8* the difference in distance is shown together with the amplitude difference. This difference in amplitude and the distance between the pitches is the key to get the desirable synthetic voices from the IIR filter. In *figure 9* a random noise is shown and later used to get another version of the synthetic voice from the IIR filter.

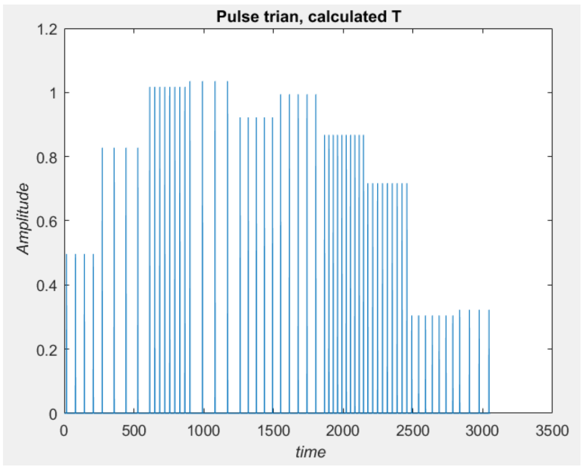

Figure 7. The figure illustrates a pulse train generated from a calculated T in Matlab that will be used in the IIR filter to get the out signal. The vertical axis shows the amplitude and the horizontal axis shows the time in milliseconds.

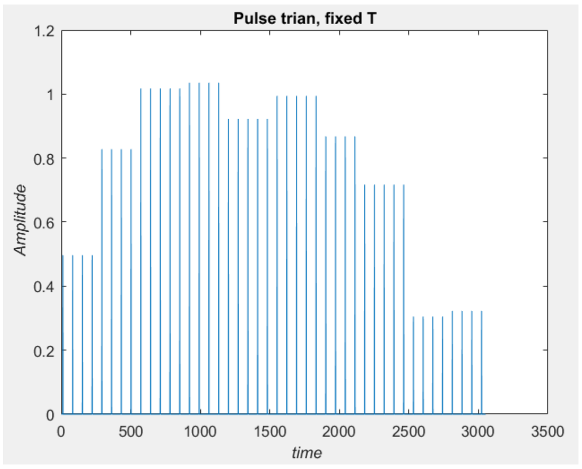

Figure 8. The figure illustrates a pulse train generated from a fixed T in Matlab that will be used in the IIR filter to get the out signal. The vertical axis shows the amplitude and the horizontal axis shows the time in milliseconds.

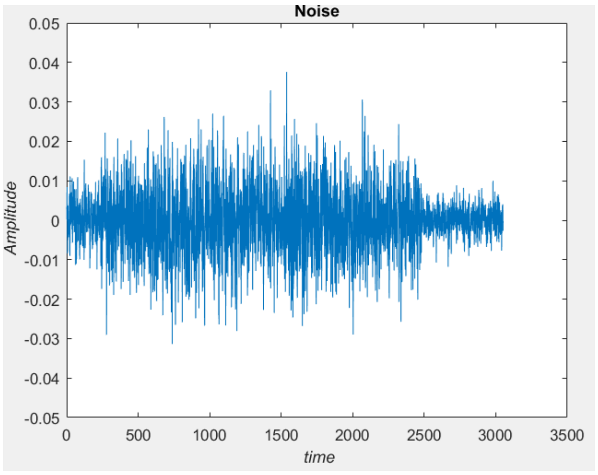

Figure 9. The figure illustrates a random generated noise in Matlab that will be used in the IIR filter to get the out signal. The vertical axis shows the amplitude and the horizontal axis shows the time in milliseconds.

The result from the DSP can be described as a robotic voice in the case where a static T was used and when a measured T was used the voice sound robotic but with a little more character in the voice. With a random noise in use the result was a lower voice with even more character than the measured T but still the person behind the voice where hidden.
To be able to switch between the different mode that been implemented button on the DSP was used and is presented in *table 2*. Button 1 will transform the original speech with noise, button 2 will sound like robotic speech and was generated by calculated T. Button 3 had a fix T and it can be heard that the robotic speeches have different pitch performance when button 2 and 3 are compared. Button 4 is sending out the original speech without any special effect.

Table 2. DSP buttons configuration.

Button	**Performance of speech**
1	Speech synthesis with noise
2	Robotic speech with calculated **T**
3	Robotic speech with fixed **T**
4	Original Speech

5. Summary

Speech synthesis was implemented through an ADSP-21262 SHARC Processor. Four different buttons on the DSP described four different out signals, with all different sounds depending on how they were modified. The signal from the number one button was a signal modified by noise. The signal from the button number two had an out signal that was modified by a calculated time interval of 70 ms and the button number three had a fixed time interval. The last and fourth button had the same in signal as the out signal, and the original speech could be heard.

Problems that emerged during the project were mostly on the coding parts when we tried to translate the Matlab code to C code for the DSP. Many things were easier to implement in Matlab compared to the DSP, mostly because of the fact that in Matlab we had build-in functions that could be used and these functions could verify our own functions as an example, the Levinson-Durbin algorithm. One problem with implementing synthetic speech in DSP is that everything needs to be done before anything can be heard from the speakers. So controlling the transfer of Matlab code to C code had to be done for each function at the time and when all of them were in the DSP the test with sound could be done. If it didn’t work the error searching was complex and hard.

6. References

Hayes Monson H.1996. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, Inc.

Haykin Simon O. 2002. *Adaptive Filter Theory*. Publishing House of Elec.

Proakis John G., Manolakis Dimitris K. 2014. *Digital Signal Processing*. Fourth Edition. Pearson Education Limited.

Swanson, David C. 2000. *Signal Processing for Intelligent Sensor Systems*. New York: Marcel Dekker, Inc.

Smith, Steven W. 1997. *The Scienetist and Engineer's Guide to Digital Signal Processing* http://www.dspguide.com/ch22/6.htm (Retrieved: 2018-02-21)

Swartling Mikael.,Grbic Nedelko., Mandersson Bengt. 2017. *Struktures lecture 12*. LTH: Digital Signal Processing http://www.eit.lth.se/fileadmin/eit/courses/eti265/lectures/notes-lecture12.pdf (Retrieved: 2018-02-26)

Swartling Mikael. 2018. Projects, lecture 2. LTH: Algorithms in Signal Processors. http://www.eit.lth.se/fileadmin/eit/courses/etin80/lectures/slides-lecture2.pdf (Retrieved: 2018-02-15)