Speech synthesizer

W. Tidelund S. Andersson R. Andersson

March 11, 2015

1 Introduction

A real time speech synthesizer is created by modifying a recorded signal on a DSP by using a prediction filter. The filter coefficients are calculated by using the Levinson-Durbin algorithm on an autocorrelated recorded signal. The algorithm then uses autocorrelation to generate a pulse train which is filtered through an FIR-filter with the prediction coefficients. This signal is then sent through an IIR-filter with the same coefficients as the FIR-filter. If the frequency of a pulse train is modified, so is the reconstructed signal. The DSP used is ADSP-21262 and it is programmed in C by using Visual DSP++ 5.0 with a preconfigured framework.

2 Implementation

2.1 Theory

The following part will briefly describe the application and each part of it will be described in more detail later. The method that was used to solve this task is called linear prediction [1], and a block diagram of the predictor can be seen in figure 1. Where x is the sampled speech signal, u is the one sample delayed speech signal, d is the desired signal, y is the predicted signal and e is the prediction error.

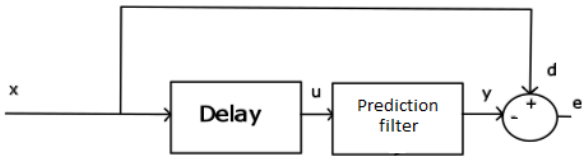

Figure 1: Block diagram of the linear predictor

An analytic filter was used to predict the n-th sample with the information acquired from the n-1 previous samples. This was done using the Levinson-Durbin algorithm, equation 4, which calculates the predictor filter taps. This filter can only predict stationary signals and since speech is stationary for about 20ms the signal was sampled in blocks of 20ms, this is represented by 320 samples with a sampling frequency of 16kHz.

Furthermore only the formants in the speech generates a stationary signal, the non-formants behaves non-stationary and transiently. This means that the prediction error mainly will contain the unpredictable non-formants and noise, also if this error signal is sent through the inverse-filter one will get the original signal back.

By manipulating the error signal and construct a pulse train out of it which is then sent through the inverse-filter one will be able to modify the original voice signal. In this project, one version of the pulse train was created from the error signal using Dirac pulses to represent the new modified voice signal. The error signal was examined and if it showed any sign of periodicity the pulse train would be implemented with pulses of that periodic frequency. The frequency of these pulse trains can then be modified to give the reconstructed signal different pitches. If no sign of periodicity was shown, noise was added instead of pulses. Another great property of the recreated signal is that it contains less data than the original signal, which is good if it is to be transmitted over a limited channel.

The following equations were used to implement the linear predictor and reconstruction of the signal:

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

Equation 1 describes the discrete filtering with an FIR-filter, where h are the filter coefficients of order M-1 acquired from Levinson-Durbin, x the input signal and y the output signal.

$$y(n) = x(n) - \sum_{k=1}^{M} a_k y(n-k) \tag{2}$$

Equation 2 shows the discrete filtering with an IIR-filter, where a_k are the filter coefficients of order M acquired from Levinson-Durbin, x the input signal and y the output signal.

$$r(n) = \sum_{k} x(n)x(n-k) \tag{3}$$

In equation 3 the autocorrelation r of the signal x is presented.

$$\begin{pmatrix} r(0) & r(1) & \cdots & r(M-1) \\ r(1) & r(0) & \cdots & r(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ r(M-1) & r(M-2) & \cdots & r(0) \end{pmatrix} \begin{pmatrix} 1 \\ a_1 \\ \vdots \\ a_M \end{pmatrix} = \begin{pmatrix} P_M \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4}$$

Equation 4 shows the forward linear prediction problem that was solved using [1] the Levinson-Durbin algorithm. Where r are the autocorrelation, a_M the resulting filter coefficients, P_M the forward prediction-error power and M the filter order.

2.2 DSP - Implementation

An offline version of the program was first created in MATLAB and tested to see if the method would work. This version partially used functions already implemented in MATLAB and these functions therefore needed to be translated into C-code in order to be implemented on the DSP.

A flowchart of the code implemented on the DSP can be seen in figure 2 which represents the whole synthesizer. To begin with, a sampled block of 320 samples is autocorrelated using equation 3. Where half of the autocorrelation vector, with the biggest value at index 0, is passed into the Levinson-Durbin algorithm. This block then solves the prediction problem in equation 4 and returns the optimum filter vector as well as the prediction-error power coefficient. The filterorder was chosen to be 15 which gives 16 coefficients. Moving on to the next block, the error signal was then obtained by filtering the original speech signal through an FIR-filter, equation 1, which consists of the coefficients from the Levinson-Durbin algorithm.

To decide whether the pulse train should consist of noise or pulses the power coefficient from the Levinson-Durbin was examined. If it was sufficiently large, meaning that the signal mostly consists of formants, pulses should be made. The frequency of these pulses were chosen by looking at the maximum correlation, in the interval of typical human speech (80-350Hz), of the autocorrelated error signal. The pulse train was then sent through an IIR-filter which is the inverse filter of the FIR-filter. By doing this the original signal was recreated and modified. To pitch the reconstructed signal, the frequency of the pulse train was either made higher or lower than the frequency found in the autocorrelation of the error. Some pseudo code is found below figure 2 and the source to these methods are attached in the appendix.

Figure 2: Block diagram of the DSP implementation

```
/* Input data */
 x         - A recorded block of 320 samples

 /* Initialization*/
xc = autoCorr(x);
levDurb = levinsonDurbin(xc); // Calculates the coefficients
xFilt = FIRfilt(x, levDurb); // Filters x with the coefficients
pulse = pulse(xc);          // Creates and modifies the pulse train
newSignal = IIRfilt(pulse, xFilt, levDurb); // Reconstructs the signal
```

Pseudo code 1: Code of the synthesizer using the same model as figure 2

3 Problems

The most persistent problem faced was how to detect a fundamental frequency in the error signal, which we could then use to create the pulse train needed to synthesize the speech. A couple of different methods were tested to combat this issue and with them entailed several further problems.

The approach finally agreed upon was the method detailed in the implementations section, using the energy given by the Levinson-Durbin algorithm as an indicator of whether the current block contains a formant or not. The problem with using this method is that the energy should be lower for the formants, since the filter resulting from the Levinson-Durbin algorithm should remove any periodic components of the speech signal (i.e. the formants). However, the energy is also highly dependent on the volume of the recorded voice. This coupled with the fact that formants typically have a higher vocal output than the non-formants, means that a static energy level above which the speech is considered to contain formants could not be consistently set. The subsequent problem consisted of determining the main frequency of the blocks which contained formants. It was decided that the autocorrelation of the error signal was to be used instead of slightly more complex solutions involving the fast Fourier transform. In order for this to work, some time lags of the autocorrelation function had to be rejected, namely those which lie outside the fundamental frequencies of human speech. For a sampling frequency of 16kHz, the lower and upper limits are 45 and 200 samples respectively. The autocorrelation sequence of the error signal of a block containing a formant can be seen in figure 3, with the sought fundamental peak within the human range and a secondary peak outside the human range both marked.

When the program was fully implemented there was a significant flaw in the finished synthesizer; the reconstructed signal did not sound as desired. In addition to this the quality of the reconstructed signal varied when the pitch was changed. Even the best result was not good enough and therefore a decision to use another method was made. This method was found during testing and uses white noise instead of a pulse train to reconstruct the signal and gave a much better result. It was now possible to distinguish most of the reconstructed words. Even when this method is used it still did not sound as desired, it is however much more acceptable than previous versions. If the implementation would have been correct it should not be possible to distinguish words from each other when using white noise. A downside to this method is that it is impossible to change the pitch of the recreated signal. Since the pulse train method should perform adequately, it was left in as an option in the DSP.

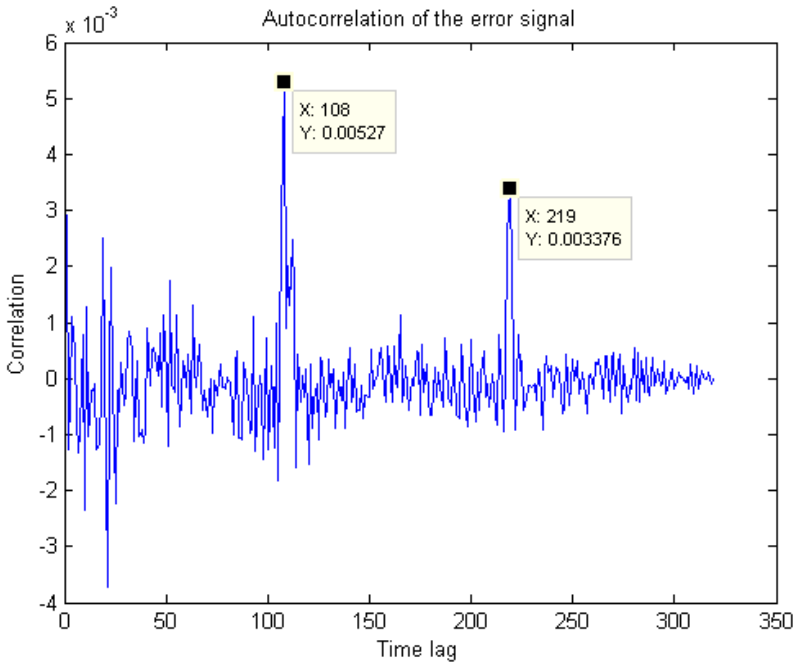

Figure 3: The autocorrelation sequence of the error signal of a block containing a formant

4 Results

A snippet of a speech signal can be seen in figure 4 and the different steps in the processing are presented in the subsequent figures 5, 6 and 7. All of the figures are results from the offline testing in MATLAB. The error signal in figure 5 is a result of filtering the input with the filter given by the Levinson-Durbin algorithm. Further, figure 6 shows that we were quite successful in finding the main frequency of the error signal and in representing this as a pulse train. Lastly, figure 7 shows the reconstructed signal which is the result of filtering the pulse train through the inverse of the filter given by the Levinson-Durbin algorithm.

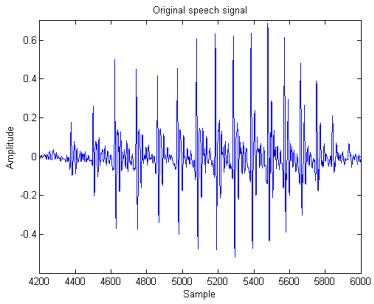

Figure 4: Part of the original speech signal

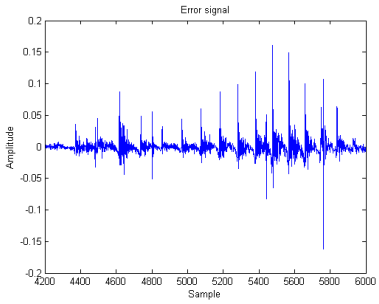

Figure 5: Part of the error signal given by filtering

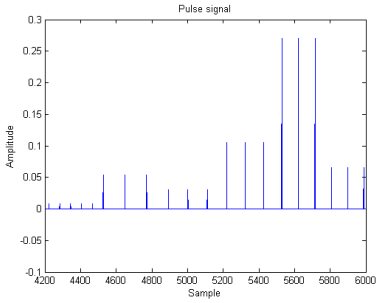

Figure 6: Part of the pulse train signal

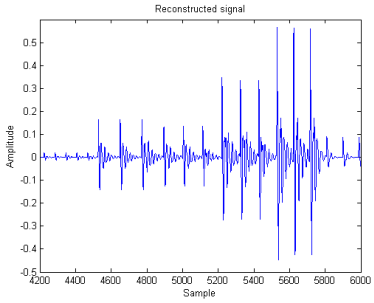

Figure 7: Part of the reconstructed signal

5 Conclusions

Although no one involved was particularly proficient in C-programming, the migration of the project from MATLAB to C went fairly smoothly in terms of getting the code to run. Getting the DSP implementation to function correctly, however, was a bit more difficult as it proved.

As can be seen in the figures 4, 5, 6 and 7 in the results section above, the recreated signal from MATLAB looks rather good. In fact, the program written in MATLAB works as intended. Since the C-program does not work exactly as advertised, there seems to be some kind of mismatch between the two. This led to mainly white noise being used to excite the IIR-filter instead of the pulse train. One possible reason for the unsatisfactory performance of the pulse train could be because of the difficulty in deciding a power threshold for the algorithm (see implementation section). This can easily be found by studying the signal offline in MATLAB, but is more arbitrarily decided in the online implementation because of all of the factors that weigh in. Of course, this discrepancy could also be an indication of the present inexperience of the

C language among the members. Despite the flaws in the implementation, the secondary goal of creating a sound effect on a speech signal has been fulfilled. Although the solution was not the most desired solution, it was made because of the time constraint of the course. Had more time been allotted, errors with the DSP implementation would likely surface.

References

[1] S. Haykin, *"Adaptive Filter Theory fifth edition"*, Pearson Education 2014