Lunds Tekniska Högskola

ETIN80 ALGORITHMS IN SIGNAL PROCESSORS SPEECH SYNTHESIS

GROUP 6

March 20, 2017

Albin Berggren, Albin.Berggren.466@student.lu.se
Nishant Gupta, nishant.gupta.4086@student.lu.se
Berta Morral, be2103mo-s@student.lu.se
Ricard Núñez-Prieto, soc15rnu@student.lu.se

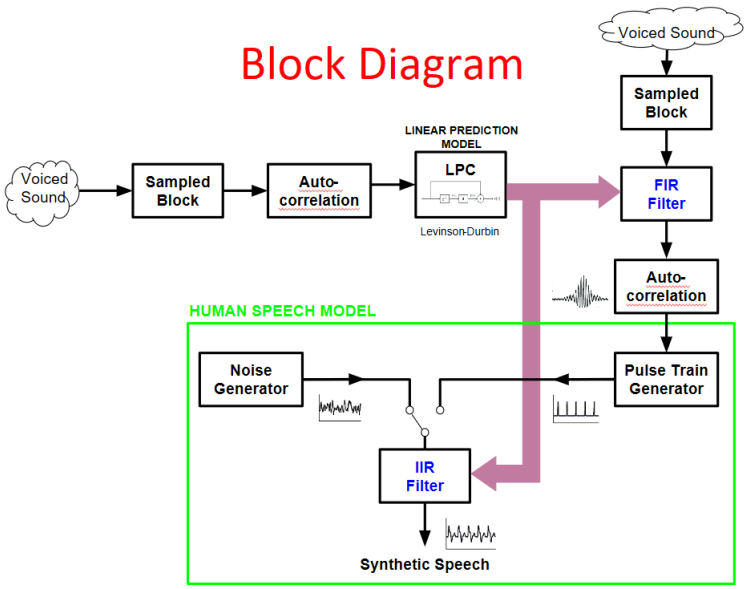
Handledare: Mikael Swartling

1 Outline

The project contains different milestones which were completed and adjusted throughout the project. This report is a reflection of the different parts of the program. We start with a brief discussion about the theory used. Then how the MATLAB model was implemented followed by the implementation in C, the code which was used on the DSP board. In the end we share our results and some conclusions about the project.

2 Theory

A known signal is autocorrelated and processed through a predictive filter which outputs an estimated error signal. The filter for the predictive model is a Wiener filter and we use the Levinson-Durbin algorithm to solve the system equations. They will provide the variance and the coefficients needed to construct an FIR filter that will estimate the value of the signal. The known signal is passed through this FIR filter which uses the coefficients obtained from the Levinson-Durbin algorithm. The signal is then autocorrelated in order to create a pulse train with the same pitch as the input signal. An IIR filter is used to recreate the original signal. The inverse filter can also be fed with e.g. a Gaussian noise signal or a pulse train with constant frequency to obtain different effects in the recreated speech.



2.1 Human speech model

Over a short time interval, about 2 to 40 ms, speech can be modeled by three parameters:

(1) The source that can be either a periodic (pulse train) or a noise excitation, where the pulse train represents the voiced sounds produced by the vibration of vocal chords and the random noise represents the fricative sounds ('f', 's') or turbulent airflow through constricted mouth. Both sources can be used independently to produce understandable human speech.

(2) In the case of using the periodic (pulse) excitation we need to know the pitch or period (the fundamental frequency of the waveform).

(3) And finally the coefficients of a recursive linear IIR filter mimicking the vocal tract response.

The reason for the use of an IIR filter is because the human speech process can be approximated to an autoregressive process which by definition specifies the output variable as a linear combination of the previous outputs. And this fits perfectly with the behavior of an IIR filter. To approximate the speech process to an autoregressive process we need first to satisfy what is called the condition of stationarity which states that if the properties of the process (in this case the vocal tract) which generates the events (in this case the speech) does not change in time, then the process is stationary.

In the case of the voice speech it is not stationary because clearly in order to produce different voiced sounds the vocal tract and the vocal chords need to change its shape but we can consider the speech process to be stationary at least during the minimum interval of time that takes for the vocal tract to change its shape to produce a new voiced sound. This time is considered to be around 20 ms. Now if you consider an audio file sampled at 16 kHz, a 20 ms interval corresponds to 320 samples which corresponds to the size of the sample blocks used in our algorithm.

Now that we already satisfy the conditions to use an IIR filter to mimic the behavior of the vocal tract, we need to know the right coefficients. We can extract these coefficients or the parameters which characterize the speech process by using a linear prediction model. If the process really is an autoregressive process, the predictive model will reveal it. By using linear prediction, the intention is to determine an FIR filter that can optimally predict future samples of our autoregressive process (speech) based on a linear combination of past samples. The difference between the actual autoregressive signal and the predicted signal is called the prediction error. This prediction error is then autocorrelated. The autocorrelated error is then used to construct the periodic excitation signal (pulse train).

2.2 Autocorrelation

In order to process the signal, it is divided into parts which are autocorrelated to find the correlation between the elements in the series. A Toeplitz matrix is created with the results which are then processed in the system.

2.3 Levinson-Durbin Algorithm

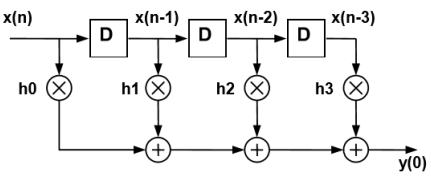
The algorithm is used in linear prediction to solve a Wiener problem, using a Toeplitz matrix created from the autocorrelated signal with the goal of minimizing the error in the signal. The base behind the recursion are to first solve a small system of small size and then use the data obtained to solve larger systems.

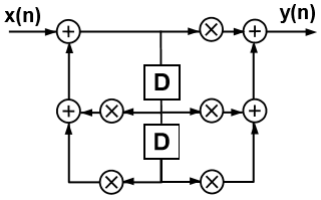
We are trying to find $A_1 = \binom{1}{a_1}$ so that $N_1 A_1 = \binom{E_1}{0}$ with N_1 as the Toeplitz matrix$\binom{R_0\ R_1}{R_1\ R_0}$. From this we find that $a_1 = -\frac{R_1}{R_0}$. We have therefore solved A_1 and also E_1, which is the variance of the signal.
If we expand the matrices used we can use the same algorithm to find the solution by iteration.

$$\begin{bmatrix} R_0 & R_1 & \cdots & R_{k+1} \\ R_1 & R_0 & \cdots & R_k \\ \vdots & \vdots & \ddots & \vdots \\ R_{k+1} & R_k & \cdots & R_0 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_k \\ 0 \end{bmatrix} = \begin{bmatrix} E_k \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

2.4 FIR-IIR filter

An FIR filter consists of an input signal, delays and the impulse response. The difference with an IIR filter is that it consists of a feedback system. The coefficients in A_1 are used as impulse response for both filters.
FIR: $y(n) = \sum h(k) \cdot x(n-k)$ IIR: $y(n) = \sum h(k) \cdot x(n-k) - \sum b(j) \cdot y(m-j)$





2.5 Pulse train

The power of the signal is concentrated into the pulses using the standard deviation gained from the Wiener filter as the square root of the variance. The signal is processed in blocks and between each of the signal blocks it is needed to adjust the placement of the pulses and the variance of the signal. This is achieved by adjusting the placement of the first pulse in a new block and using the new pitch offset for the rest of the block.

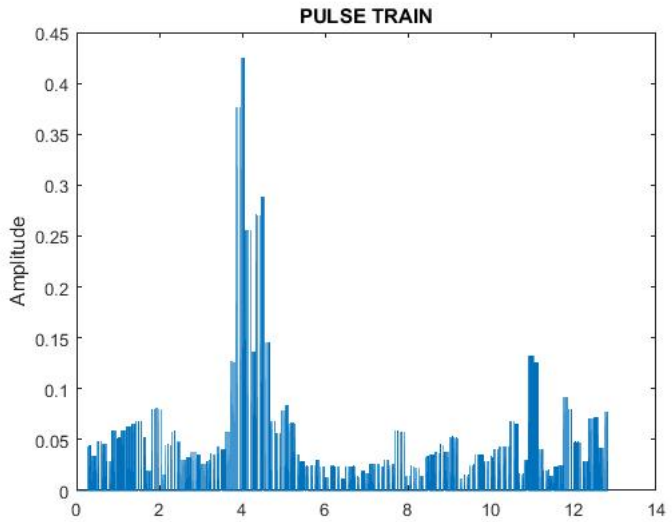


Figure 1: Pulse train. Horizontal axis represents the sample number and the units are given in tens of thousands

3 MATLAB model

Once we understood the theory and the algorithms to use, we started implementation of the algorithm in MATLAB. We began using MATLAB because of the amount of implemented functions it has and the simplicity to debug the code.

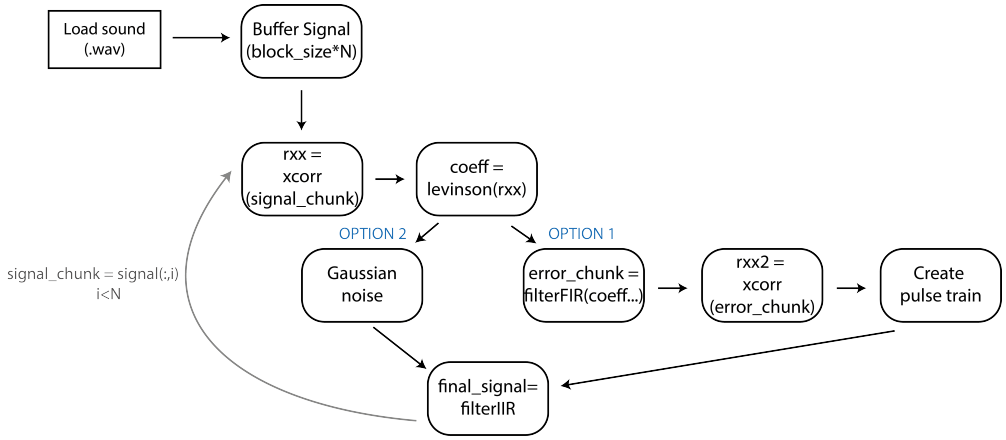


Figure 2: Schematic Matlab code

We started using a prerecorded signal as the input data, so the first step was to load this prerecorded signal and save it into a MATLAB variable. At this point we have a vector with all signal samples we need to process. As it is explained before, we must process the signal in chunks or blocks of approximately 20 ms to assure the stationary signal. Therefore, we converted the signal vector in a matrix of rows size of block size that is equivalent to 20 ms and N columns depending on the signal vector size. It allows us to process each column separately and to have a stationary signal.

After that, we have to do the autocorrelation of the signal chunk to obtain the Levinson coefficients. At this point, depending on which effect is wanted the code differs from each other.

Option 1: Once the Levinson coefficients are obtained the signal needs to be filtered using FIR filter and Levinson coefficients. Then the autocorrelation of the filtered signal has done in order to find the pitch and be able to create the pulse train. There are two different effects done depending on how we create the pulse train.

- Option 1a: Variable pitch. The pitch is founded in the chunk so it may change from chunk to chunk.

- Option 2b: Fixed pitch. Constant fixed pitch value.

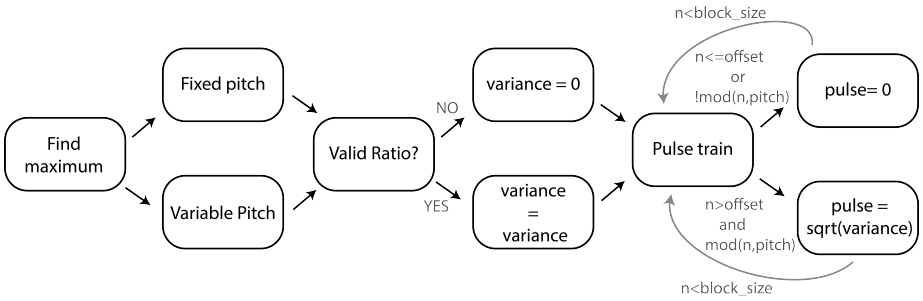


Figure 3: Create Pulse Train Block

Option 2: Raspy or harsh voice effect. At this point we just need to create a Gaussian noise vector using the variance obtained on Levinson function and the size equal to the block size variable.

Last step for both options is to use the IIR filter to reconstruct the modified signal. If the option 1 is used, the pulse train signal is needed but if the option 2 is used, the Gaussian noise signal is needed instead. In both cases the linear prediction coefficients are needed.

4 Implementation in C

Once we had MATLAB code working we started to program in C code. The first step was to create the functions that did not exist in C code:
- Filter FIR
- Filter IIR
- Find Max/Min
- Autocorrelation
- Levinson

After the functions were created and tested separately, it was the moment to start to program the whole system.

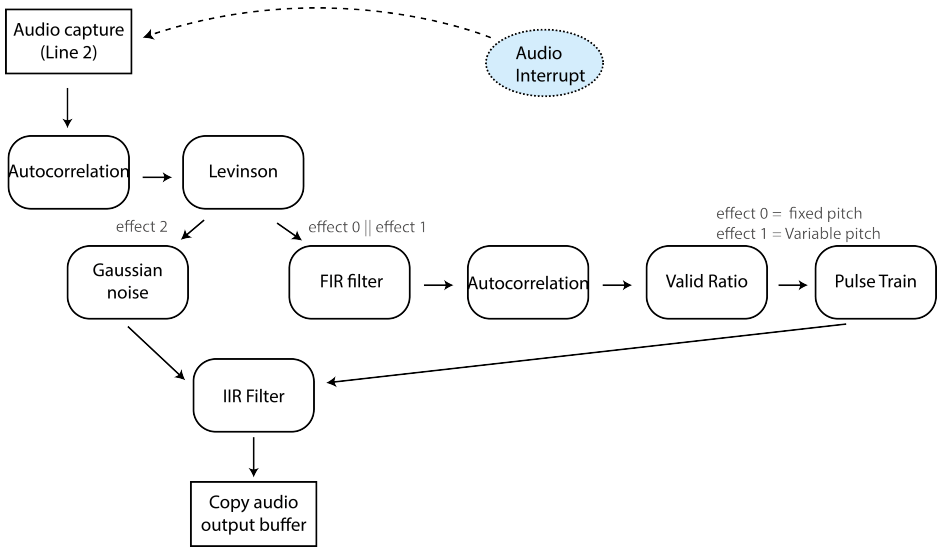


Figure 4: Schematic C code

The C program itself is almost the same it is in MATLAB but with some differences. Firstly, we should highlight that in this case we do not have an audio file to process at the beginning, we use the DSP board audio interrupt to catch the audio instead. Therefore, it allows us to synthesize the voice in real time. When the audio processing is done, the audio is sent to the output audio channels (left and right). Another important difference is how the effects are changed, in this case the push buttons are used to modify the effect applied each time.

BUTTON	EFFECT
SW1	Effect 0: Fixed Pitch Pulse Pitch 1 Pulse Pitch 2
SW2	Effect 1: Variable Pitch
SW3	Effect 2: Raspy Voice
SW4	Effect 3: No effect applied

Table 1: Push buttons vs Effects equivalences

5 Result

The result of the project is a working product implemented on the DSP board with working buttons for different operations. The following graphs show the result of the implementation in MATLAB. Note that the horizontal axis represents the sample number and the units are given in tens of thousands.

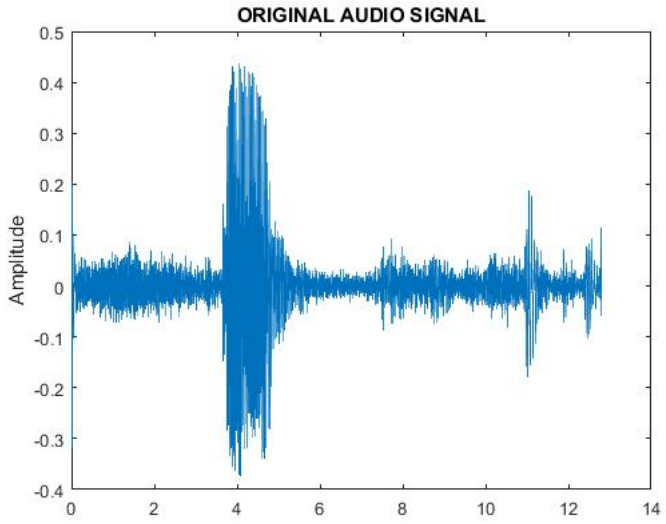


Figure 5: Original signal fed into the system

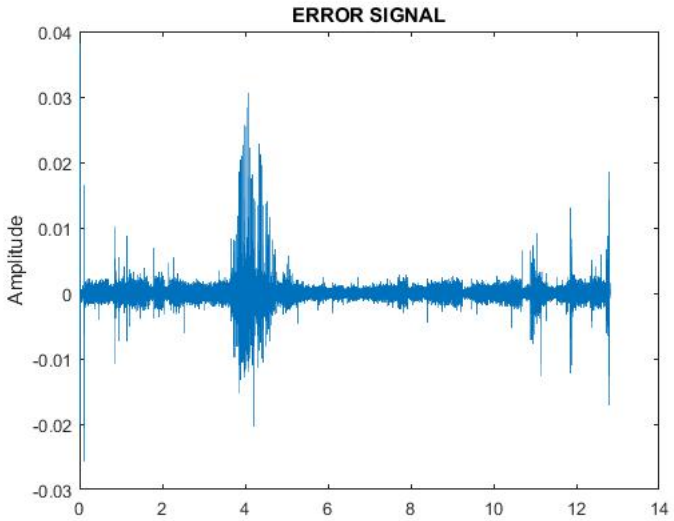


Figure 6: The error signal from the filter

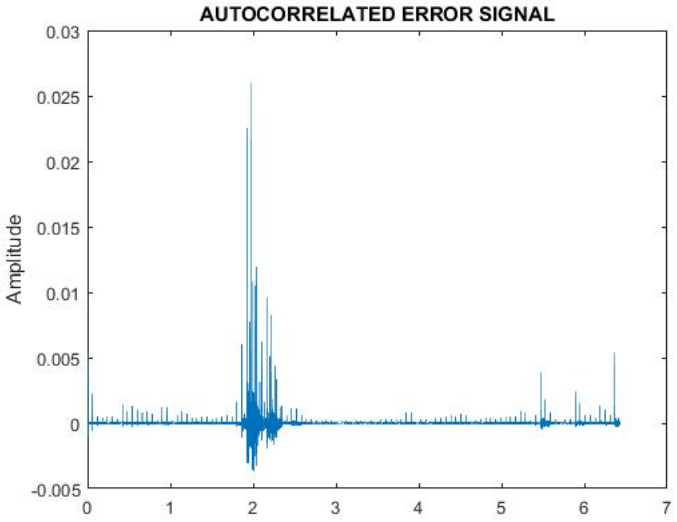


Figure 7: The auto-correlated signal

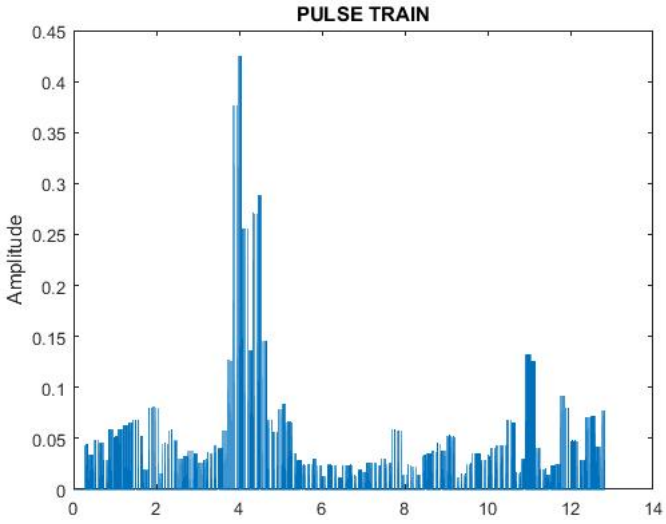


Figure 8: The pulse train

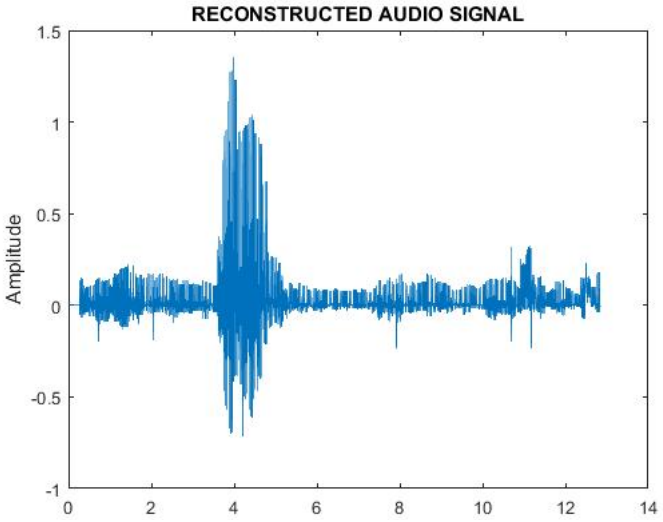


Figure 9: The reconstructed signal without effects

6 Conclusions

The graphs from the MATLAB scripts shows the functions of the internal components included in the design. The implementation of the functions in MATLAB yielded results which helped to give an understanding of the concepts and made it easier to implement the program on the DSP.

After completing the project in MATLAB and then its implementation on real time DSP board, we were able to successfully produce the planned types of voice effects. It was interesting to try the algorithms learned from other courses and implement it in a way that creates these results.

7 The code

7.1 Matlab code

```
% If the properties of the process that generates the signal DOES NOT
% change in time, then the process is stationary ( Ex.: weather is a prime
% example of a chaotic process, it cannot be considered stationary for too
% long).
%
% The Basic Properties of Speech:
% ===============================
% Speech is produced when air is forced from the lungs through the vocal
% cords and along the vocal tract.
% The tract introduces short-term correlations (of the order of 1 ms) into
% the speech signal,and can be thought of as a filter with broad resonances
% called formants.
% The frequencies of these formants are controlled by varying the shape of
% the tract, for example by moving the position of the tounge.
% An important part of many speech codecs is the modelling of the vocal
% tract as a short term filter.
% As the shape of the vocal tract varies relatively slowly,
% the transfer function of its modelling filter needs to be updated only
% relatively infrequently (typically every 20 ms or so).
% THAT's the reason why the speech process is considered a quasi-stationary
% process (and 20 ms is the typical time that is considered as the time
% that the shape of the vocal tract remains unchanged during the speech
% process).
% In other words, the shape of the vocal tract and its mode of excitation
% change relatively slowly, and so speech can be considered to be
% quasi-stationary over short periods of time (of the order of 20 ms).
%
% BANDWIDTH SPEECH: the speech signal has frequency components upto 8 kHz
% and hence 16 kHz (wideband speech) is the optimal sampling frequency.
% However, when telephone communication started, with bandwidth being a
% precious resource,the speech signal was passed through an anti-aliasing
% low pass filter with cutoff frequency of 3.3 kHz and sampled at 8 kHz
% sampling frequency (narrowband speech).
%
clear all
close all

[stereo_signal,fs] = audioread('stereophonic.wav');
% 'signal' is the 2-column matrix storing the audio record;
% 'fs' is the sampling frequency; this info can be extracted also from the
% wav file using: audioinfo();

mono_signal = stereo_signal(:,1);
%data = load('snippet.mat');
%mono_signal = data.snippet;

% the wav file is a 2-column matrix: one column for each audio channel (stereo).
% In principle both channels are the same for the files that are going to be used.
% We just need to use to use the info of one of the channels.

stationary_time = 20e-3;
block_size = floor(fs * stationary_time);

% use block processing technique; the block size can be defined as the
% number of samples generated during the 20 ms of the cuasi-stationary
% speech process;
```

```
signal = buffer(mono_signal,block_size);

% convert the signal into a matrix with multiple columns

[rows,columns] = size(signal);
lag = 100; % VARIABLE 'lag' also sets the number of coefficients that the Levinson_Durbin algorithm
lag_window = floor(block_size/2); % VARIABLE 'lag_window' sets the window for the auto-correlation o

% The variable 'lag' sets the lag interval used to calculate the
% autocorrelation sequence that describes our Auto-regressive (AR) process
% (i.e. the speech signal).
% The lag range goes from -lag to +lag (in total the obtained
% autocorrelation sequence has 2*lag+1 elements).
%
% Definition: An Autoregressive process/model is a statistical forecasting
% model in which future values are computed only on the basis of past
% values of a time series data.
%


filter_state1 = [];
filter_state2 = [];
pulse_offset=0;
% The 'filter_state' vector will be used by the FIR and IIR filter; this is
% so because when filtering large amount of data in blocks or 'chunks'
% using a FIR or IIR, there are some delay elements which pass some values
% to the next iteration.
% In its most basic form, the 'filter' function initializes the
% delay outputs to 0. This is equivalent to assuming both past inputs
% and outputs are zero. So, this produces a glitch or discontinuity in
% next iteration output.
% But by setting the initial conditions by using the information stored in
% the previous iteration, we assure to not to cause any glitches in the
% output when processing the data in chunks.

for i=1:columns
    signal_chunk = signal(:,i);
    %each 'chunk' of data corresponds to one column

    rxx = xcorr(signal_chunk, lag);
    % the autocorrelation of each chunk of data is calculated in the lag
    % interval that goes from -lag to +lag

    rxx = rxx(lag+1:end);
    % the order of the obtained autocorrelation sequence is of order
    [coeff,variance] = levinson(rxx);
    % rxx is the autocorrelation sequence that characterizes our
    % autoregressive process (speech signal) of order=length(rxx)-1.
    % The coefficients of our AR linear process
    % are obtained by using the MATLAB 'levinson' function.


    Hz_num = coeff; % H(z) numerator
    Hz_den = 1;     % H(z) denominator


    [error_chunk, filter_state1] = filter(Hz_num, Hz_den, signal_chunk, filter_state1);
    error(:,i) = error_chunk;
    % The filter function is defined by its transfer function H(z) = N(z)/D(z)
    % where N(z) = SUM(ai*z^(-i)) <from i=0 to i=P>
    % with P=feedforward filter order, i.e. amount of previous input values
```

```
%
% and D(z) = 1 + SUM(bj*z^(-j)) <from j=1 to j=Q>
% with Q=feedback filter order, i.e. amount of previous output values used
% in the filter
%
% IF D(z)=1 then it is a FIR filter (its impulse response or response to
% any finite length input is of finite duration).


%noise_chunk = sqrt(variance)*randn(block_size,1);
noise_chunk = sqrt(variance)*wgn(block_size,1,0); %white noise, 0db(1W)
% generate random noise (normal or gaussian distribution) that can be
% used instead of the pulse train to produce a synthetic speech.
% randn(sz1, sz2)  --> sz1 ROWS x sz2 COLUMNS
noise(:,i) = noise_chunk;
% construct the noise signal 'chunk by chunk'

% PULSE TRAIN
xcorr_error_chunk = xcorr(error_chunk,lag_window);
xcorr_error_chunk = xcorr_error_chunk(lag_window+1:end);

% 1) To construct the pulse train sequence corresponding to each chunk
% of input signal first we need to auto-correlate the error signal with
% itself. The lag window should be large enough that allows to find
% enough peaks in the auto-correlated signal.
% The first value of the resultant auto-correlated vector is always the
% largest of the all vector.

local_offset = 150; % increase the offset if you hear synthetic noises

% We make the search setting the window limits, starting at the
% position defined by the variable 'local_offset' up to the end of the
% vector or earlier.

[Max_Peak Q] = max(xcorr_error_chunk(local_offset:end));

% Store the sample index with the peak in the variable 'pitch'

pitch = Q + local_offset - 1;
% OPTION 1: VARIABLE PITCH. USE THIS VALUE of 'pitch' to reconstructed a
% synthetic version of the original speech.
% 'pitch' is the distance(in number of samples) between the pulses
% that conform the pulse train.
% !!NOTE: NOISE reduction and better voice quality can achieved by
% increasing the variable 'stationary_time' (f.ex.: 30 or 40 ms)

%pitch = 200;
% OPTION 2: USE THIS 'pitch' for a REAL ROBOT speech effect. It sets
% a FIXED value for the train pulse pitch. Changing this value we can
% change the pitch (tone) of the synthetic voice. Also increasing the
% number of Levinson-Durbin coefficients will improve sound quality.
% !NOTE:
% 1) for HIGH-PITCH voice, better voice quality by DECREASING the value of
% the variable 'stationary_time' (f.ex.: 10 ms) and
% also decreasing the value of the pitch 'pitch'

pulses = linspace(0, 0, block_size);
ratio = Max_Peak/xcorr_error_chunk(1);

if (ratio)<0.05
% decrease the ratio limit accordingly if you hear cuts during the speech
%if ((ratio)<0.05||xcorr_error_chunk(1)<0.0008)
```

```
 % alternatively we can try to sort out chunks with very small
 % autocorrelation value (small main peak, Index=1)
        variance = 0;
   end

% Contruct the pulse train (carry the previous pitch when needed)

        for n = 1:block_size;
           if (n <= pulse_offset)
               pulses(n)=0;
           else
               if (mod((n - pulse_offset - 1),pitch) == 0)
                    pulses(n) = sqrt(variance);
                else
                    pulses(n)=0;
               end
           end
        end

% recalculate the offset accordingly
    if (mod((block_size - pulse_offset),pitch) == 0)
        pulse_offset = 0;
    else
        pulse_offset = pitch -(mod((block_size - pulse_offset),pitch));
    end;

    pulse_train(:,i) = pulses;
    xcorr_error(:,i) = xcorr_error_chunk;
    % Construct the pulse train and the auto-correlated error signal 'chunk
    % by chunk' so they can be plotted later.



    % RECONSTRUCT OUTPUT SIGNAL
    % ========================
    % To invert the filter we only have to invert the transfer function
    % H(z)of our previous FIR filter, i.e. when passing the parameters to
    % the MATLAB 'filter' function we just swith the numerator by the
    % denominator. The input now is each one the error signal 'chunks' that
    % we obtained previously.


    [synth_speech_chunk, filter_state2] = filter(Hz_den, Hz_num, pulses, filter_state2);
%%%% OPTION 1: USING the PULSE TRAIN (FIXED PITCH OR VARIABLE PITCH)
    % NOTE: for variable pitch, the quality of the speech can be improved
    % by increasing the time 'stationary_time' (f.ex.: 50 ms)

    %[synth_speech_chunk, filter_state2] = filter(Hz_den, Hz_num, noise_chunk, filter_state2);
%%%% OPTION 2: USING the random noise (harsh, raspy voice)

    synth_speech(:,i) = synth_speech_chunk;
    % construct the synthesized signal 'chunk by chunk'

end

%%% Convert resultant signal matrices into 1-D vector for plotting purposes
error_vector = error(:);
synth_speech_vector = synth_speech(:);
xcorr_error_vector = xcorr_error(:);
pulse_train = pulse_train(:);

%%%% PLOT ORIGINAL AUDIO SIGNAL
```

```
figure;
plot(mono_signal);
title('ORIGINAL AUDIO SIGNAL');

%%%% PLOT ERROR SIGNAL
figure;
plot(error_vector);
title('ERROR SIGNAL');

%%%% PLOT ERROR SIGNAL
figure;
plot(xcorr_error_vector);
title('AUTOCORRELATED ERROR SIGNAL');


%%%% PLOT RECONSTRUCTED AUDIO SIGNAL
figure;
plot(synth_speech_vector);
title('RECONSTRUCTED AUDIO SIGNAL');

%%%% PLOT DIFFERENCE BETWEEN ORIGINAL AND RECONSTRUCTED SIGNAL
figure;
plot(mono_signal - synth_speech_vector(1:size(mono_signal)))
title('DIFFERENCE BETWEEN ORIGINAL AND RECONSTRUCTED SIGNAL');

%%%% PLOT PULSE TRAIN
figure;
plot(pulse_train)
title('PULSE TRAIN');


%%%% REPRODUCE SIGNALS
% sound(mono_signal, fs);
% pause (5); % pause of 5 seconds
sound(synth_speech_vector, fs);
```