Course: 김상효 교수님 Signals and Systems
Theme: Design of a signal processing system
ID/Name: 2017312605 김요셉
Date: 2022-05-22

1. 구현 목표

우리는 본 설계 보고서를 통해 선형 신호 처리 시스템 구현의 목표와 구현 결과, 그리고 검증과정을 확인할 수 있습니다. 본 신호처리 설계에서는 (1) LPF 를 fir1 함수를 통해 구현하며, 이를 확장하여 (2) Filtering (LPF, HPF, BPF)을 FIR 로 구현합니다, 또한 BPF 를 통해 (3) Equalizer 를 구현합니다. fir 이외에도 과제를 통해 익힌 $fft()$ 함수의 사용법과 chapter3 에서 익힌 convolution 의 duality 를 활용하여 (4) Echo 효과를 구현합니다. 마지막으로 주파수 도메인에서의 독립변수의 크기 변화를 통해 시간 도메인에서의 종속변수의 축소와 확장을 하는 (5) pitch 조절 기능을 구현합니다.

2. 구현

2.1 LPF

2.1.1 기능 설명

MATLAB 의 신호처리 tool 중 하나인 fir1 함수를 통해 이를 구현합니다. Fir1 함수는 N 개의 값에 대해 수행되기 때문에 일반적으로 안정적이고 선형 구현이 가능합니다. 디지털 필터로 구현이 용이하다고 하며, 시간영역에서 수행되는 fir1 함수로 MATLAB 에서 사용됩니다. 소스코드로 제공된 Example 코드의 내용과 동일합니다. 여기서 얻어진 b 와 a 의 값은 MATLAB toolbox 의 filter() 함수에서 파라미터로 사용됩니다.

2.1.2 소스코드

```
function [b, a] = get_lpf(fc, fs, N)
    % FIR Window Lowpass filter designed using the FIR1 function.
    % All freq in Hz.
    % fs: Sampling Frequency
    % fc: Cutoff Frequency
    % N: Order

    win = 2 * fc / fs; % (rad/sample /pi)
    b = fir1(N, win, 'low'); % coefficients of x in difference equations
    a = [1];
end
```

Figure 2.1 MATLAB function to get LPF

2.2 Filtering (LPF, HPF, BPF)

2.2.1 기능 설명

기본적으로 LPF 의 기능을 확장했으며, 동일하게 fir1 함수를 통해 구현됩니다. HPF 의 경우 *'high pass'* 를 입력하여 동일하게 얻을 수 있으며, BPF 의 경우 'bandpass' 를 fir1 함수의 인수로 입력하여 얻습니다. 여기서 얻어진 b 와 a 의 값은 filter 함수에서 input data 의 필터링에 사용됩니다.

2.2.2 소스코드

```
function [b, a] = get_hpf(fc, fs, N)
    % FIR Window Lowpass filter designed using the FIR1 function.
    % All freq in Hz.
    % fs: Sampling Frequency
    % fc: Cutoff Frequency
    % N: Order

    win = 2 * fc / fs; % (rad/sample /pi)
    b = fir1(N, win, 'high'); % coefficients of x in difference equations
    a = [1];
end
```

Figure 2.2.1 MATLAB function to get HPF

```
function [b, a] = get_bpf(fc1, fc2, fs, N)
    % FIR Window Bandpass filter designed using the FIR1 function.
    % All frequency values are in Hz.
    % fs: Sampling Frequency
    % N: Order
    % fc1: First Cutoff Frequency
    win = blackman(N + 1);
    % return
    b = fir1(N, 2*[fc1 fc2] / fs, 'bandpass', win, 'scale');
    a = [1];
end
```

Figure 2.2.2 MATLAB function to get BPF

2.3 Equalizer (Base, Middle, Treble)

2.3.1 기능 설명

equalizer 는 음원의 특정 음역대를 강화하거나 감소시키는 역할을 할 수 있습니다. 구현하기 위해 다양한 방법이 있지만, 기존에 구현한 Filter 중 BPF 를 통해 구현하는 방법이 가능해 보여서 이를 적용해 보았습니다. BPF 를 통해 특정 음역대만 통과시킨 후, 나머지 원본 파일에서 이를 제거해 줍니다 (inverse filtering) 이후, 통과된 특정 음역대를 정해진 gain 에 따라 강화/약화시키고 그 밖의 영역은 반대로 약화/강화합니다. 이후 두 값을 더하면 특정 영역대에 대하 강조/약화된 결과를 얻을 수 있습니다.

우리는 가청 음역대가 (20Hz, 20kHz) 사이에 분포되어 있다는 사실과, 일반적으로 오디오 소프트웨어에서 Bass, Middle, 그리고 Treble 이 각각 (20Hz, 300Hz), (300Hz, 6kHz), (6kH, −Inf)로 간주한다는 사실을 바탕으로 Bass, Middle, Treble 을 각각 강조하는 Equalizer 효과를 구현했습니다. 아래의 subsection 을 통해 Equalizer 모듈과 이를 활용하여 bass, middle, 그리고 treble 을 강조하는 코드가 설명됩니다.

*fname*은 확장자를 포함한 파일명, channel 은 채널 번호$(1, 2, \dots)$, gain 은 특정 음역대에 대한 강조/약화 수치, fc1 은 해당 음역대의 lower bound, fc2 는 해당 음역대의 upper bound 입니다. N 은 BPF 의 fir1 함수에서 사용되는 차수이며, 값이 높을수록 더 왜곡이 감소되는 것으로 알려져 있습니다.

2.3.2 소스코드

```
function [equalized_sound, fs] = get_equalized_sound(fname, channel, gain, fc1, fc2, N)
    [x_tot, fs] = audioread(fname);
    x = x_tot(:, channel);
    [b, a] = get_bpf(fc1, fc2, fs, N);
    adder_x = filter(b, a, x);
    %return
    equalized_sound = (x - adder_x)*(1/gain)+ (gain)*adder_x;

    get_cmp_fig(0:1:size(x, 1), fs, x, equalized_sound);
end
```

Figure 2.3 MATLAB function for equalizer

2.4 Echo

2.4.1 기능 설명

Echo 함수는 Dirac delta 함수를 통해 구현했습니다. 위에서 구현한 필터를 사용하지 않기 때문에 바로 결과 데이터와 sampling rate 값을 반환하며, *fade_rate*은 뒤의 echo 값이 어느 정도인지를 결정합니다. 1 일 경우 동일한 값이 울리고, 감소할수록 echo 된 소리의 크기가 감소하여 마치 산에서 울리듯 실감나게 처리됩니다. *Bool_want_plot*은 어떤 결과인지 알아보기 위해 넣은 것으로, true 일 때 plot 을 보여줍니다. *num_echo*의 값이 많아질수록 메아리의 수도 많아지는데, 테스트 코드에서는 2 개와 4 개인 경우를 각각 보였습니다.

2.4.2 소스코드

```
function [echo_sound, fs] = get_echo_sound(fname, channel, num_echo, bool_want_plot, fade_rate)
    [x_tot, fs_tot] = audioread(fname);
    fs = fs_tot;
    x = x_tot(:,channel);

    echo_frac_n = linspace(0, 1, num_echo+1);

    h = zeros(num_echo, size(x, 1)+1);

    % number of index
    n = 0:1:size(x, 1);

    %calculate impulse response
    for i = 1:num_echo
        h(i,:) = dirac(n - round(echo_frac_n(i)* fs));
        index = h(i,:) == Inf;
        h(i, index) = fade_rate^((1/num_echo)*(i-1));
    end

    h_tot = sum(h);

    %get echo sound
    echo_sound = conv(x,h_tot);
    %remove silent part at the end
    echo_sound = echo_sound(1, 1:size(x, 1) + (fs * 0.75));

    if(bool_want_plot)
        get_cmp_fig(n, fs, x, echo_sound);
    end

end
```

Figure 2.4 MATLAB function for echo effector

2.5 pitch up & down

2.5.1 기능 설명

Pitch up & Down 은 3 단원에서 학습한 Fourier Transform 의 time domain scaling 이 frequency domain 에서 magnitude 를 변화시킨다는 점을 사용하여 만든 간단한 구현입니다. 단순히 fs 값에 gain 을 곱했을 뿐인데 그 감소된 값의 역수만큼 time scaling 이 이루어집니다. 예를 들어 더 넓은 frequency band 를 제공하면 소리의 pitch 가 증가되게 되고 반대로 time domain 에서는 compress 됩니다. 이와는 대조적으로 fs 값의 감소는 time domain 의 확장을 가져오고, pitch 의 감소를 야기합니다.

2.5.2 소스코드

```
% pitch up -> time domain compression
audiowrite('../dat/output/processed_pitch_up.wav', X, fs*2);
% pitch down -> time domain tension
audiowrite('../dat/output/processed_pitch_down.wav', X, fs/2);
```

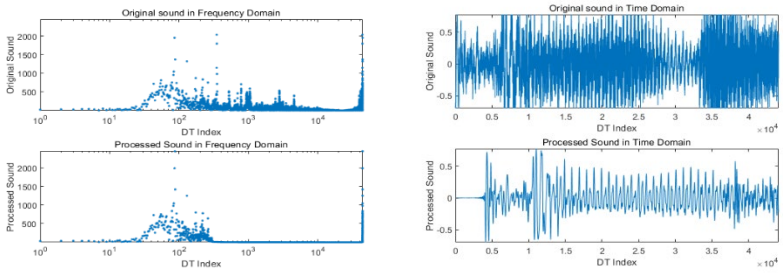
Figure 2.5 MATLAB Implementation of pitch up and down

3. 구현 검증 및 확인

첨부된 *exe.m* 스크립트 파일을 실행하여 4sceconds.wav 를 신호처리한 결과물의 time-domain 과 frequency domain 에서 결과물입니다. 주파수 영역의 경우 fft함수로 푸리에 변환을 시행한 후 그 magnitude spectrum 을 구하여 ω 값의 순서에 해당되는 DT index 값에 axis 에 대한 *semilogx* 플롯을 그렸습니다.

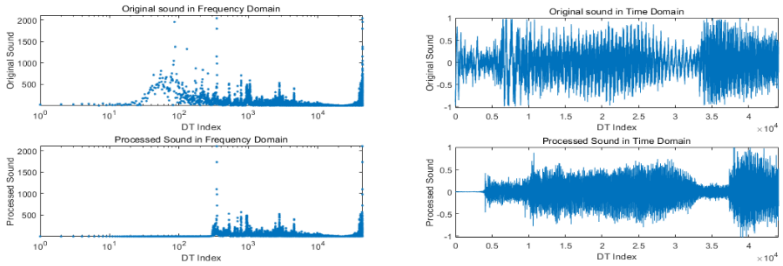
3.1 LPF

LPF 와 HPF 의 경우 Bass 라인인 300Hz 까지를 cutoff frequency 로 하여 실험했습니다. LPF 의 결과는 다음과 같습니다.

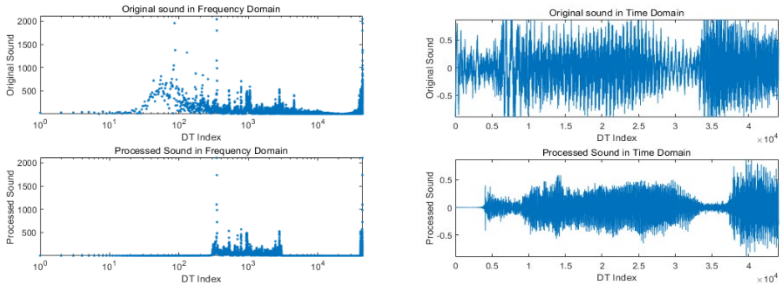


3.2 Filtering

3.2.1 HPF

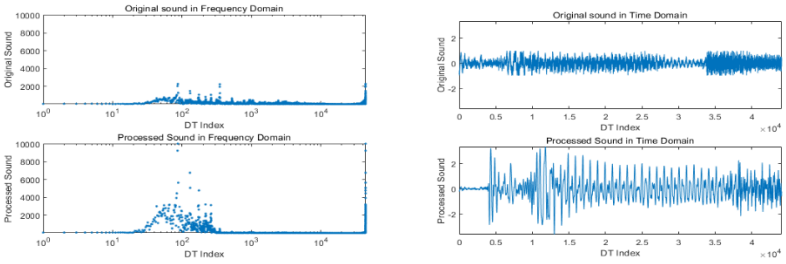


3.2.2 BPF

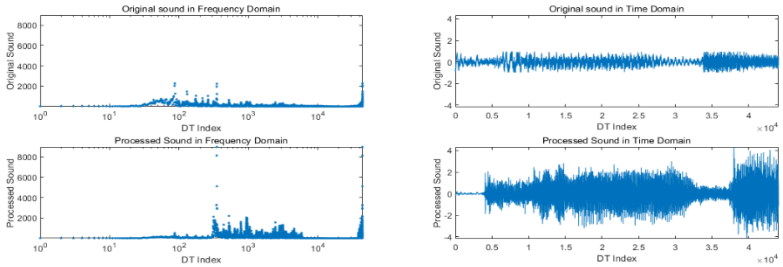


3.3 Equalizer

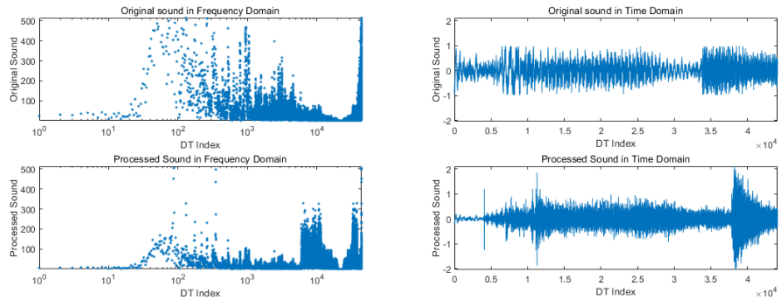
3.3.1 Bass Booster (20Hz, 300Hz)



3.3.2 Middle Booster (300Hz, 6kHz)

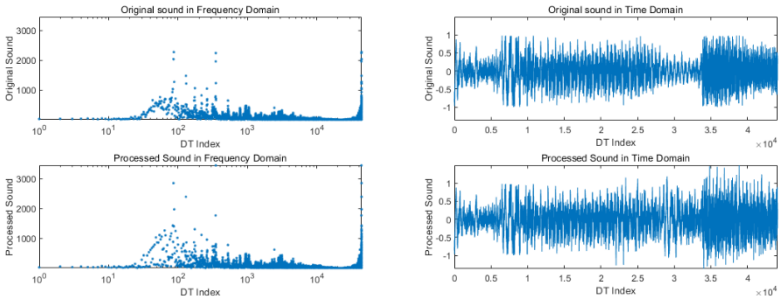


3.3.3 Treble Booster (6kHz, 20kHz)

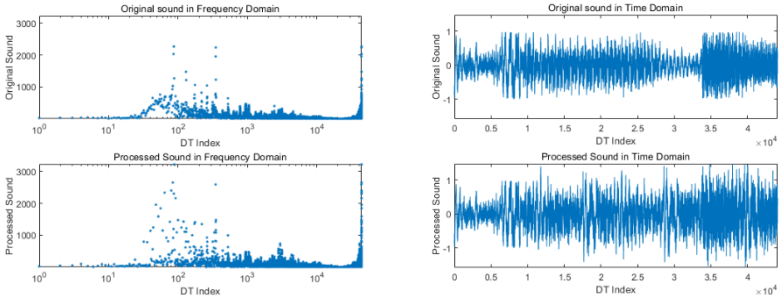


3.4 Echo

3.4.1 Double Echo



3.4.2 Quad Echo



3.5 Discussion

Pitch Up & Down 의 경우, 단순히 주파수 영역에서의 magnitude spectrum 의 값이 증가/감소하게 됩니다. 그 밖에도 Band Pass Filter 를 통해 우리는 특정 역대의 noise 만 추출하여 이 값을 원 음원에서 빼고 noise reduction 을 수행할 수 있습니다. 대표적으로 무선 이어폰 등에 구현된 noise filter 도 이러한 형태로 구현되었을 것으로 생각되며, 추가적인 연구를 통해 더 효율적이고 왜곡이 적은 방식의 구현이 가능할 것으로 기대됩니다.