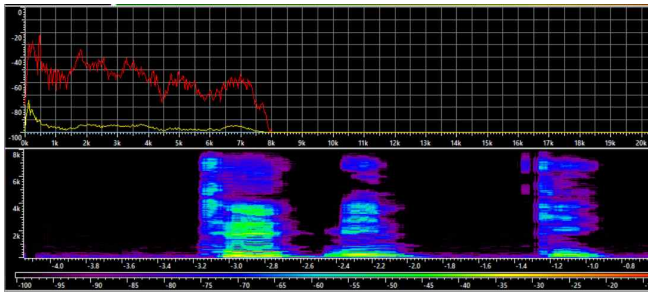
전기전자 공학부 202114160 채민기

moving average filter

원본 음성 (본인의 이름인 채 민 기를 사용하였다.)



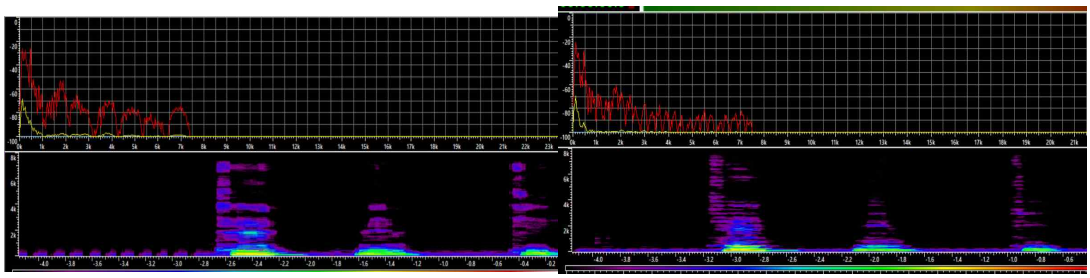
첫글자인 채에서 ㅊ이 무성음으로 고주파 성분을 띄고 있어 세 개의 스펙트로그램중 첫 번째 부분에서 고주파 성분이 비교적 높음을 확인 할 수 있었다.

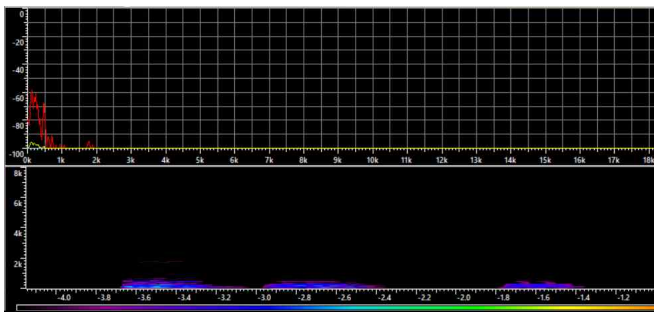
실험전 예상

moving average filter의 경우 LPF로 작용한다. 따라서 tap수를 점차 늘리면 고주파 성분이 점점 줄어들 것이라고 예상 할 수 있다.

실험 결과 및 분석

다음의 3개의 사진은 각각 tap수를 15, 51, 1999로 설정했을 때의 사진이다





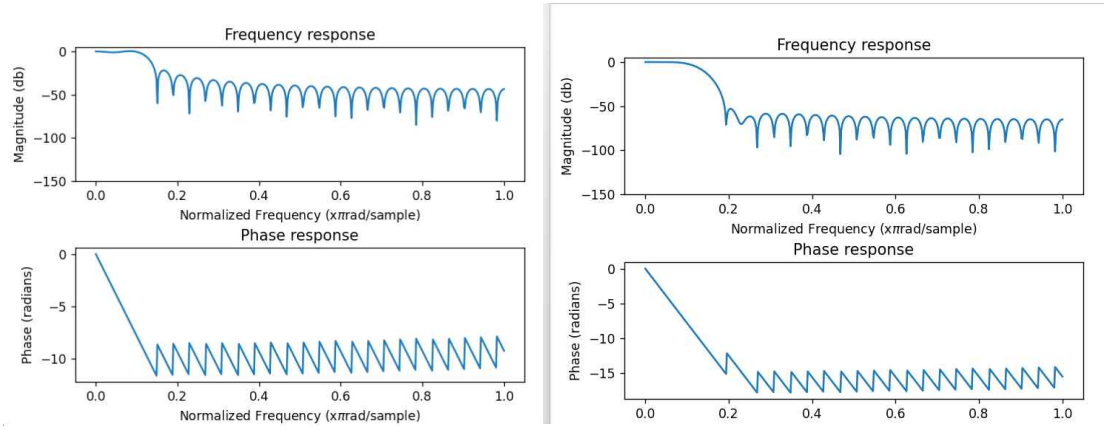
실험전 예상과 동일하게 moving average filter의 tap 수를 늘림에 따라 고주파성분이 제거 되는 것을 스펙트로 그램을 통해 확인할 수 있으며. 스펙트럼을 통해 허용하는 주파수 범위가 줄어드는 것을 확인 할 수 있다.

실제로 tap수를 늘릴수록 낮은 저음소리만이 남게 되어 소리가 점점 먹먹한 느낌을 주는 것을 확인 할 수 있었다. 또한 채 가 재로 들리는 등 무성음이 잘 들리지 않았다.

window method <rectangulat, hamming>

rectangular, hamming window의 tap수를 51로 설정하였다.

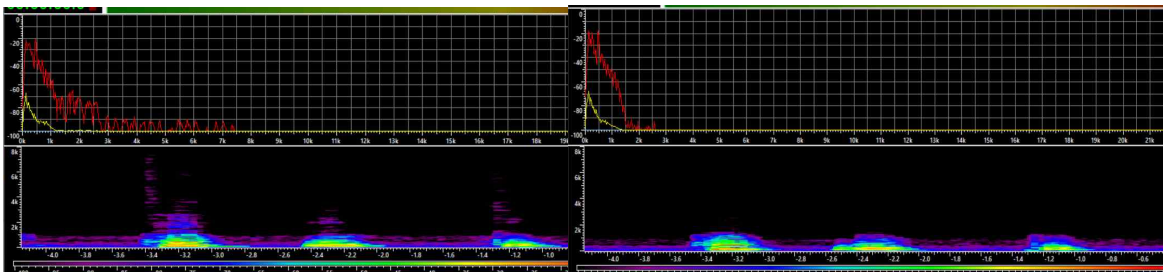
다음은 순서대로rectangular, hamming window를 적용한 필터의 모습이다.



실험전 예상

fplot을 통해 두 개의 window로 설정한 filter를 보면 이론과 같이 main lobe의 폭이 좁은 것은 rectangular window 였으며, side lobe의 크기가 작아 attenuation의 크기가 큰 것은 hamming window로 설계한 필터였다. 위의 필터 모습을 통해 같은 소리에서 hamming window를 적용한 것이 누설이 적을 것이라고 예상 할 수 있었다.

실험결과 및 분석 <좌측rectangular, 우측 hamming >



기본적으로 두 개의 필터모두 LPF로 작용하여 앞선 실험처럼 채가 재로 들리는 등 소리가 뭉게지는 느낌을 받았다. 동일한 tap 수에서 두 개의 필터는 hamming window를 적용한 경우 더 낮고 먹먹한 느낌을 주었다. 이는 설계된 필터에서 예상한 사실로 hamming window를 적용한 경우 rectangular window 에 비해 side lobe의 크기가 작아 누설현상이 비교적 적기 때문이다. 이는 위 스펙트럼에서 좌측의 rectangular window의 경우 더 많은 스펙트럼 분포를 보이고 있음을 통해 확인 할 수 있다.

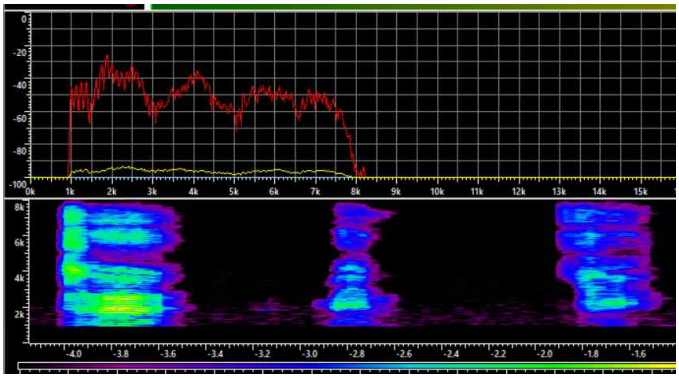
window method<HPF,BPF,BSF>

High pass filter

실험전 예상

HPF의 특성 대로 moving average filter에서 약해졌던 고주파 성분인
ㅊ소리를 잘 들을 수 있을 것이라고 예상된다.

실험결과 및 분석



LPF에 비해 얇고 날카로운 소리가 작게 들리는 것을 확인했으며 소리의 울림이 적은느낌을 받았다. ㅊ부분은 앞서 LPF에서 줄었던것과 반대로 잘 들렸다.
앞선 실험들은 LPF의 특성으로 인해 기부분의 소리가 그대로 기로 들려 이상함을 감지 하지 못했으나 HPF를 적용한 해당 실험에서는 기보다 키에 가까운 소리를 확인 할 수 있었다.

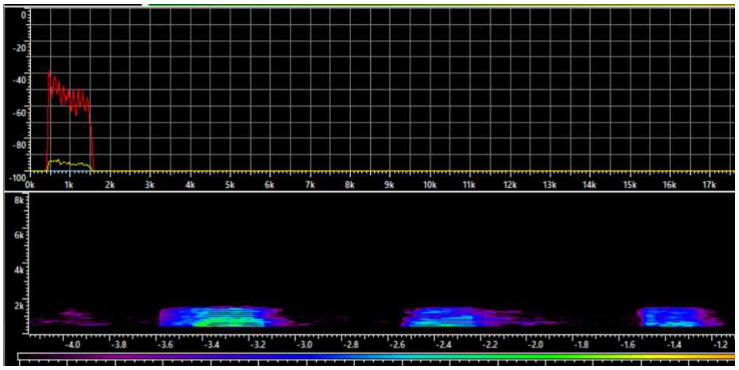
스펙트럼과 스펙트로그램을 통해 1khz의 통과 주파수를 갖는 것을 확인 할 수 있으며
저주파 성분이 차단되어 이전 실험에 비해 고주파 성분이 비교적 많이 검출 된 것을 확인 할 수 있다.

Band pass filter

실험전 예상

사람의 음성이 일반적으로 약 4kHz까지의 대역폭을 갖고 있으며
남성의 경우 목소리가 낮기 때문에 저주파 성분이 많다. 실제로 HPF에서 소리가 LPF에서보다 작게 들렸던 점에서 본인의 목소리는 저주파 성분이 많아 소리가 작게 들릴 것이라고 예상된다.

실험결과 및 분석



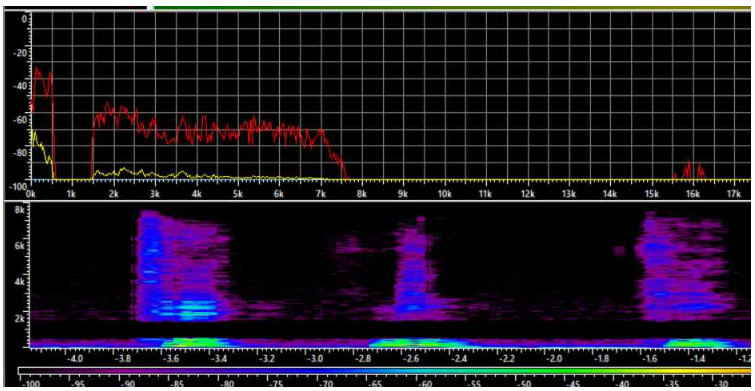
채민기 라는 이름이 태나기로 들렸다. 울림소리가 거의 없었으며 LPF에서 느꼈던 답답한 소리와 다르게 소리자체는 잘들리는 편이였다. 다른 음성태스트를 해봤을 때 ㄴ이나 ㅁ과 같은 비음은 소리가 구분되지 않고 ㄴ과 ㅇ의 중간의 소리가 들렸다.

Band stop filter

실험전 예상

BPF와 정반대의 스펙트럼을 가질것이므로 잘 들리지 않았던 비음이나 ㅊ이 잘 들릴 수 있을 것이라 예상된다. 하지만 사람의 음성 주파수 대역의 가운데를 차단하는 만큼 특정 부분의 소리가 잘 들리 지 않을 것이라 예상된다.

실험결과 및 분석



채민기 라는 본인의 이름은 다른 필터들에 비하여 가장 잘 들렸으며 특별히 소리가 안들린다고 여겨지는 부분이 없어 자음과 모음을 각각 발음해 보았다.
ㅏ와 ㅑ 모음의 소리가 ㅡ와 ㅖ의 중간 소리로 들리며 심하게 뭉게지는 듯한 느낌을 받았다.
ㅜ발음되 종종 ㅗ와 ㅡ의 중간소리로 들려 구분이 어려웠다.

음소유형	해당 음소	주파수
대부분의 자음	- b, d, m, n, z, v 등 - ㅁ, ㅂ, ㅈ, ㄴ, ㄷ 등	125 ~ 500Hz
대부분의 모음	- a, e, i, o 등 - ㅗ, ㅜ, ㅏ, ㅓ, ㅣ, ㅐ, ㅔ 등	500 ~ 1500Hz
대부분의 격음 및 복합자음	- k, t, p, ch, sh, th 등 - ㅋ, ㅌ, ㅍ, ㅊ, ㅎ, ㅅ, ㅆ 등	1500 ~ 8000Hz

해당 표에서 볼수 있듯이 차단된 500~1500Hz의 주파수 대역에는 대부분의 모음이 포함되어있다. 이러한 이유로 자음에서는 크게 다른 느낌을 받지 못하고 모음에서 차이를 느낀것이라 생각된다.

Code

```
# movig average filter  + window
import numpy as np
import pyaudio
import time
import sys
import keyboard
import scipy.signal as signal
import fplot
#sampling freq
RATE =16000
#size of buff 0.1sec  data
CHUNK =int(RATE/10)
# kerbel_size는 tap 수를 의미하므로 f
# 이 부분을 조정하여 tap 수를 조정하면 된다
# Tap 수는 홀수여야 할것
kernel_size =505
kernel = np.full(kernel_size,1/kernel_size)
in_data = np.zeros(CHUNK+kernel_size, dtype=np.int16)
# in_data = np.zeros(CHUNK+kernel_size-1, dtype=np.int16)
filter_on =False
# def filter
# cut off freq = nomalized
# sampling frequency is 16khz  0.125= 1khz
rect_kernel = signal.firwin(kernel_size,cutoff=0.125,window ="boxcar")
hamm_kernel = signal.firwin(kernel_size,cutoff=0.125,window ="hamming")
# plot the filter
print("Plot of Rectangular Filter")
```

```
fplot.mfreqz(rect_kernel)
fplot.show()
print("Plot of Hamming Filter")
fplot.mfreqz(hamm_kernel)
fplot.show()
# other filters ramdom window
# window select hanning,Black man
#highpass  f_cutoff is 1khz
highpass = signal.firwin(kernel_size,cutoff=0.125,window="hann",pass_zero='highpass')
# BPF & BSF f_cutoff = 500hz ~ 1.5khz
#bandpass
bandpass = signal.firwin(kernel_size, [0.0625, 0.1875], window="hann",
pass_zero='bandpass')
#bandstop
bandstop = signal.firwin(kernel_size, [0.0625, 0.1875], window="hann",
pass_zero='bandstop')
# 해당 부분을 통해 필터 설정
kernel = bandstop
# real time 처리를 고려
def convolution(signal, kernel):
    n_sig = signal.size
    n_ker = kernel.size
    n_conv = n_sig - n_ker
    #by favtor of 3
    rev_kernel =kernel[::-1].copy()
    result = np.zeros(n_conv,dtype=np.int16)
    for i  inrange(n_conv):
        if filter_on:
            result[i] = np.dot(signal[i:i+ n_ker],rev_kernel)
        else:
            result[i] =  signal[i+n_ker]

    signal[0:n_ker] = signal[n_sig-n_ker:n_sig]
    return result
# main routine
p = pyaudio.PyAudio()
stream                                                                    =
p.open(format=pyaudio.paInt16,channels=1,rate=RATE,input=True,output=True,
                    frames_per_buffer=CHUNK,input_device_index=2)
while(True):
    # CHUNK SIZE만큼 버퍼에 in
    # samples = input buff  string type으로 들어감 binary data로 변경필요
    samples = stream.read(CHUNK)
```

```
        # fromstring을 통해 받아와 변화 int16으로
        in_data[kernel_size:kernel_size+CHUNK] = np.fromstring(samples, dtype=np.int16)
        out = convolution(in_data,kernel)
        # out을 파일로 쓰면 파형 관측 가능
        # tostring을 통해 내보내야함  output도 string으로 나가야함
        y = out.tostring()
        stream.write(y)
        if keyboard.is_pressed('q'):
            break
        if keyboard.is_pressed('f'):
            if filter_on:
                filter_on =False
                print("Filter off.")
            else:
                filter_on =True
                print("Filter on.")
# terminate program
stream.stop_stream()
stream.close()
p.terminate
```