잡음 제거 기술을 통한 라디오 음질 개선

Enhancement of Radio Sound Quality Using Noise Reduction Algorithm

멘 토 조 교 : KAIST 이 준

연 구 자 : 대전동신과학고등학교 김찬진

대전동신과학고등학교 김찬형

대전동신과학고등학교 유형민

대전동신과학고등학교 이돈영

대전동신과학고등학교 임시우

대전동신과학고등학교 장성규

연구결과보고서

잡음 제거 기술을 통한 라디오 음질 개선

Enhancement of Radio Sound Quality Using Noise Reduction Algorithm

Student : 대전동신과학고등학교 2학년 김찬진 guardian7610@naver.com
대전동신과학고등학교 2학년 김찬형 cksgud0630@naver.com
대전동신과학고등학교 2학년 유형민 1312pp@naver.com
대전동신과학고등학교 2학년 이돈영 dy4849526@naver.com
대전동신과학고등학교 2학년 임시우 siwlim8086@naver.com
대전동신과학고등학교 2학년 장성규 jangsgkevin@naver.com

Mentor : 이 준 조교 jun8572@kaist.ac.kr

1.서론

1) 연구 동기 및 배경

우리가 살아가는 세상에는 다양한 소리들이 존재한다. 그 중에서도 자동차나 기차, 비행기와 같은 교통수단으로부터의 소음, 공장에서 나는 기계음, 심지어는 작게 바스락거리는 소리조차도 일부의 사람들에게는 소음으로 느껴질 수 있다. 소음으로 인한 피해는 인간의 개인차나 컨디션, 주변 상황에 따라 들쭉날쭉하기 때문에 크게 중요하게 여겨지지 않았지만, 최근 아파트 층간 소음 문제가 살인 사건으로 이어지는 등 관련 문제가 발생하면서 더욱 이슈화되고 있다. 이러한 이슈와는 별개로 음향기기의 발전으로 노이즈캔슬링이 이어폰이나 헤드폰 등에 적용되면서 이 또한 화제가 되고 있다. 전자기기 회사인 A사에서 새로 출시한 무선 이어폰이 큰 인기를 끌고 있는 이유 또한 노이즈캔슬링에서 찾아볼 수 있다. 이처럼 소음 및 청각 장치에 대한 사람들의 시선과 관심은 나날이 높아지고 있는 중이다.

2) 연구 목적

앞서 설명했던 소음 제거 기술과 생활 속 잡음 제거라는 목표를 달성하기 위해 본 연구를 통해 일상생활에서 간단하게 적용 가능한 잡음 제거 디바이스를 만들고, 이를 실생활에 적용하는 부분을 시연하는 것을 목표로 하였다. 보다 구체적으로 적용하자면, 라디오를 청취 시 발생하는 잡음을 줄이고 원본 목소리만 또렷하게 송출할 수 있도록 하는 장치를 제작하는 것으로 목표를 설정하였다. 본 연구

를 통해 라디오 청취 시 잡음 제거뿐만 아니라, 무전 통신, 통화 잡음 개선 등 생활 속 여러 분야에서 개발한 기술을 활용할 수 있을 것으로 예상된다.

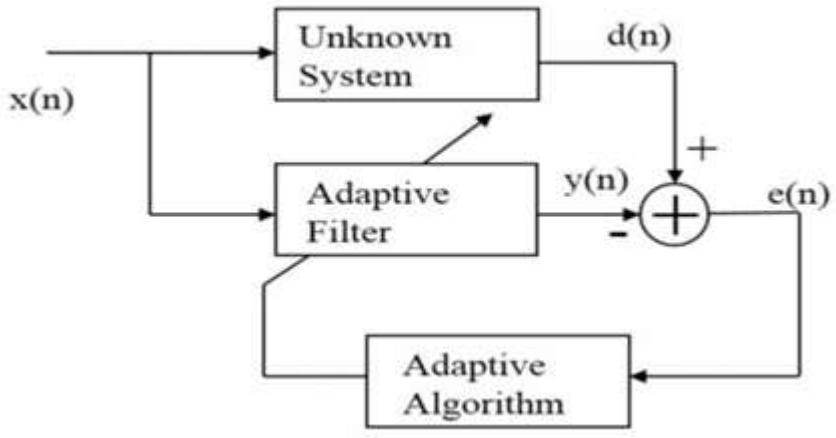
2. 연구 내용

1) 이론적 배경

본격적인 코딩 및 시제품 제작에 앞서 선행 연구에 대한 조사와 탐구를 진행하기로 하였다.

(1) NLMS 필터 알고리즘

NLMS(Normalized Least Mean Square Algorithm) 필터는 정규화된 LMS 필터를 의미한다. LMS란 Least Mean Square의 약자로, 여러 변량들의 편차의 제곱합을 최소로 만드는 어떠한 x값을 구하는 알고리즘이라고 할 수 있다.

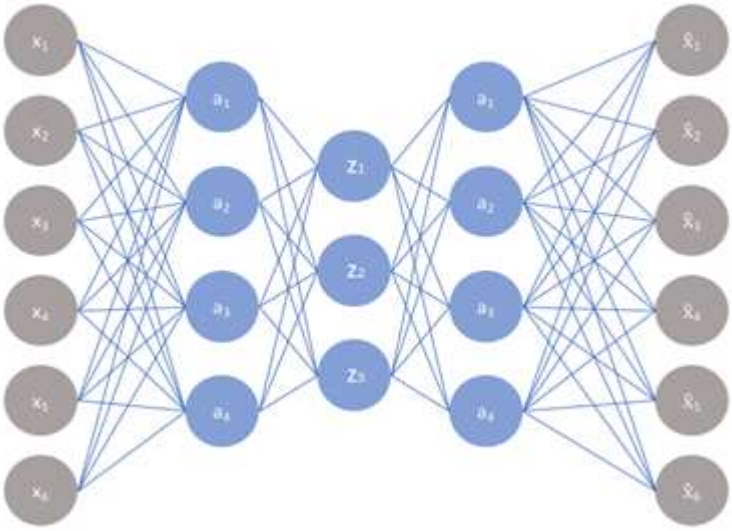


[그림 1] NLMS 알고리즘의 순서도

일반적인 LMS 필터가 아닌 '정규화된' LMS 필터를 잡음 제거에 이용하는 이유는 입력하는 데이터가 주변 환경, 즉 조용한 곳인지 시끄러운 곳인지에 따라서 그 크기가 달라지기 때문이다. LMS 필터는 위 그림 1과 같이 원본 입력 데이터인 x(n)과 노이즈가 섞인 데이터인 d(n)이 있을 때 d(n)에 일단 필터를 한번 거쳐 y(n)을 만들고, 해당 소리가 x(n)에 가까워지도록, 수치적으로 말한다면 e(n) = y(n)-x(n)이며, e(n)을 최소로 만드는 n 값을 국소 시간대 내애서 찾아내는 방법을 사용하는 필터이다. 일단 해당 n값을 찾아내었다면, 다음 DT(Discrete Time)에는 해당 정보를 피드백하여 조금 더 e(n)의 절댓값을 줄여나가는 쪽으로 필터의 방향을 변경하는 것이다. 이러한 방법의 가장 큰 특징은, 적응형이라는 것에 있는데, 즉 국소 시간 내에서 잡음의 주파수 대역은 동일하다는 가정에서 출발하는 것이다. 이러한 이유에서 비행기나 환풍기에서 나는 소음 등 주기적이고, 잡음의 대역이 잘 변하지 않는 경우는 잘 제거하는 모습을 보이는 반면에, 불규칙적이고, 불연속적인 소리의 경우 소음을 잘 제거하지 못하기도 한다. 또한, Adaptive Filter에 입력해 주어야 하는 데이터가 원본 데이터인 x(n)과 노이즈가 섞인 데이터인 d(n)으로 총 2가지 종류의 데이터를 모두 입력하여 주어야 한다는 점에서, 빠르게 노이즈가 섞인 소리만을 이용하여 소음 제거를 해야 하는 상황에는 부적합하다.

(2) 오토인코더 알고리즘

오토인코더(Autoencoder Algorithm) 방식은, 머신러닝의 일종인 비지도 학습(Unsupervised Learning)을 이용하여 데이터의 노이즈를 제거하는 방식으로, 인코더를 통하여 입력 데이터의 특징을 추출하고 디코더를 통하여 원본 데이터를 재구성하는 학습방식이다.

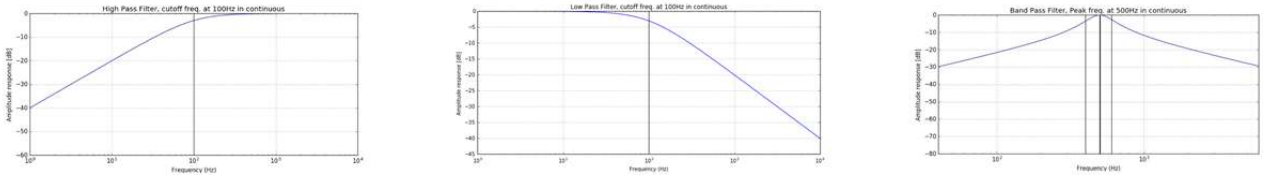


[그림 2] 오토인코더 방식에서의 노드

오토인코더 방식은 그림에서 알 수 있듯 중간 단계의 노드 크기가 작다는 특징이 있다. 이러한 특징 덕분에 오토인코더 방식은 그림 파일의 화질을 개선하는 알고리즘에 주로 적용된다.

(3) 밴드 패스 필터 알고리즘

밴드 패스 필터(Band-Pass Filter) 방식은 위의 방식과는 다르게 특정 주파수를 증폭시키고 특정 주파수는 차단하는 필터를 고정적으로 적용하는 방식이다. 특별히 고 주파수 대역이나 저 주파수 대역만을 남기고 나머지를 차단하는 필터를 의미한다.



[그림 3] 밴드 패스 필터의 주파수별 출력 그래프

해당 필터의 경우에는 특정 주파수 대역만을 증폭시키고 나머지를 차단하는 방식이므로, 시간에 따라 노이즈의 주파수가 변하는 경우에는 추가적인 작업을 통하여 노이즈의 주파수 대역을 찾아주는 작업이 필요하다.

(4) 푸리에 변환(Fourier Transform)

푸리에 변환은 원래 함수를 수많은 기저 함수들의 합으로 나타내는 변환을 의미하며 각각의 기저 함수는 원래 함수에 미치는 영향인 기여도에 해당하는 계수와 고유 주파수를 가지는 삼각함수로 표현된다.

$$g(t) = \int_{-\infty}^{\infty} G(f)s(t)df \quad \text{..... 식 1}$$

g(t) - 원본함수, G(f) 계수, s(t) - 기저함수

$$c(n) = \frac{1}{T}\int_{-T/2}^{T/2} \tilde{f}(t)e^{i2\pi ft}dt \text{..... 식 2}$$

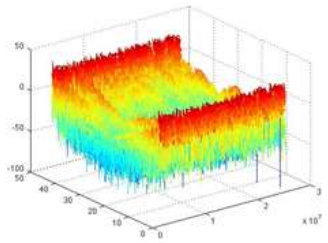
이후 T를 무한대로 보냄으로써 모든 시간에 대한 함수를 분해하였을 때 나타나는 기저함수의 계수를 구할 수 있다.

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \quad \text{..... 식 3}$$
$$T \to \infty, \omega_0 \to 0$$

2πf를 ω0로, nω0를 ω로 보고 f(t) 신호를 실수 전체 범위에서 t에 관해 적분하면 F(ω)로 변환이 되는데 이것이 푸리에 변환이며 이 함수를 T로 나누어 준다면 푸리에 급수의 계수인 Cn을 구할 수 있게 된다. 여기까지가 기본적인 푸리에 변환의 개념이고 추가적으로 푸리에 변환의 알고리즘을 좀 더 쉽고 빠르게 하기 위한 고속 푸리에 변환 등이 있다.

(5) Short-Time Fourier Transform (STFT)

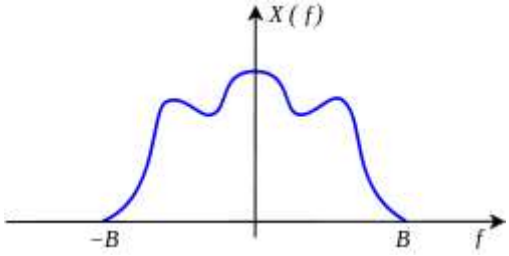
FFT 변환은 어떠한 자료를 한번의 분석을 통하여 주파수 피크점을 파악하는 기법이라 할 수 있는데, 저주파수대에 피크점이 있는 Part A와 고주파수대에 피크점이 있는 Part B가 합쳐진 음성파일을 FFT 변환을 할 시 저주파수대와 고주파수대에 동시에 피크점이 생겨, 어떤 시간이 저주파수이고, 어떤 시간이 고주파수인지 알 수 없다. 그렇기에 STFT 변환은 등시간 간격으로 파일을 나누어 여러번 FFT를 진행하는 방법으로 시간구간마다 확실한 피크값을 파악할 수 있다.



[그림 4] STFT 3차원 그래프

(6) Nyquist-Shiannon Sampling Theorem

푸리에 변환을 위해 아주 짧은 시간인 DT로 나누게 되는데, 실제로는 연속적인 신호를 얼마나 자주 샘플링 해야 그 신호를 손상없이 사용할 수 있을지를 판단하기 위해 만들어진 이론이다.



[그림 5] 주파수에 따른 신호의 세기

위 그림 5에서의 신호를 보면 –B에서 B까지로 주파수의 크기가 한정되어있다는 것을 의미한다. 이를 Bandlimited 라고 한다. 이러한 신호를 정보의 손실 없이 수집하려면 해당 신호를 1/2B만큼의 간격(즉 1/2B를 주기로 설정)으로 샘플링 한다면 정보를 빠짐없이 수집가능하다. 이 때의 주파수인 2B를 특별히 Nyquist Frequency라고 한다.

(7) Python 라이브러리

우리는 라즈베이 파이에 접목하기 쉬운 파이썬(Python) 프로그래밍 언어를 이용하기로 하였다. 이중 수학적 프로그래밍에 사용되는 여러 라이브러리를 조사하였다.

1) NumPy 라이브러리

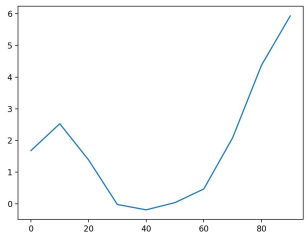
NumPy 라이브러리는 행렬이나 일반적으로 대규모 다차원 배열을 쉽게 처리 할 수 있도록 지원하는 파이썬의 라이브러리이다. NumPy는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공한다. 배열 생성, 선형 대수학, 최근접 이웃 탐색 문제 등 다양한 문제에 적용할 수 있다. 데이터 구조는 ndarray(N-dimension Array) 구조로 말 그대로 n차원 배열이다. ndarray 구조는 list 구조와 다르게 한 가지의 데이터 타입만 담을 수 있고, 따라서 더 간결하다는 장점이 있다.

2) Scipy 라이브러리

Scipy 라이브러리는 과학기술계산을 위한 파이썬 라이브러리이다. Scipy는 파이썬을 기반으로 하여 과학, 분석, 그리고 엔지니어링을 위한 과학적 컴퓨팅 영역의 여러 기본적인 작업을 위한 라이브러리이다. Scipy는 기본적으로 Numpy, Matplotlib, pandas, Sympy등 과 함께 동작한다. Scipy는 수치적분 루틴과 미분방정식 해석기, 방정식의 근을 구하는 알고리즘, 표준 연속/이산 확률분포와 다양한 통계 관련 도구 등을 제공한다. 본 연구에서 사용되는 푸리에 변환의 알고리즘도 Scipy를 통해서 구현해 낼 수 있다.

3) matplotlib 라이브러리

matplotlib 라이브러리는 파이썬에서 자료를 차트나 플롯으로 시각화하기 위한 라이브러리이다. 본 연구에서는 matplotlib 라이브러리를 wav파일을 시각화하기 위한 도구로 사용할 것이다. 오른쪽 그림은 matplotlib를 사용하여 임의의 플롯을 출력한 결과이다. 그림과 같은 직선 형태 뿐만 아니라 히스토그램 형태, 점선 형태 등 다양한 방식으로 시각화를 할 수 있다.



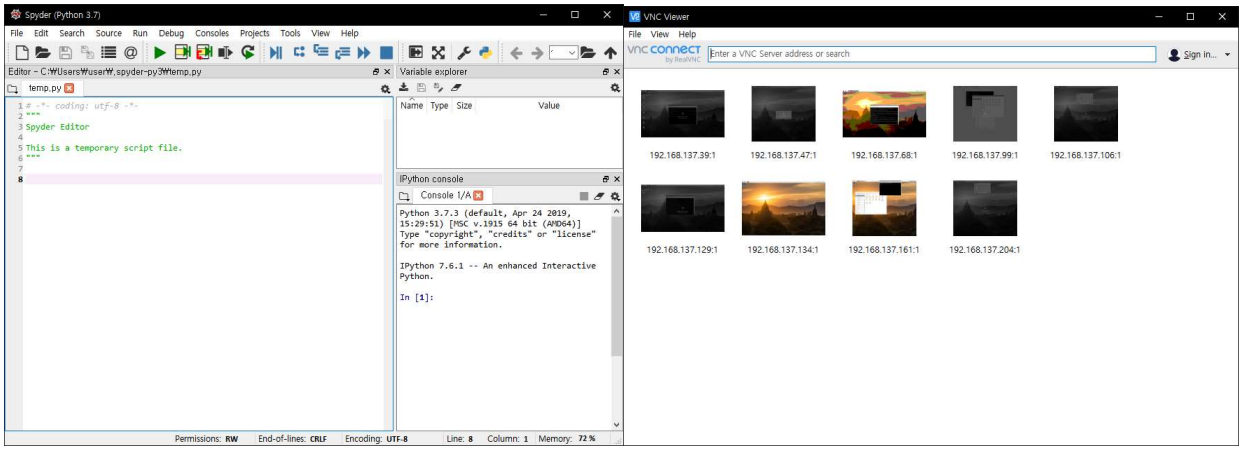
[그림 6] 여러 가지 처리가능한 그래프의 형태

(6) 선행 연구 검토 내용

현재 Krisp라는 회사에서 통화시 주변 잡음을 제거하는 노이즈 캔슬 소프트웨어를 출시한 상태이다. 이 회사의 앱을 사용하며 주로 게임을 하거나 음성 채팅, 통화를 할 때, 통화 상대의 마이크를 통해 사람 말 뿐만 아니라 키보드 소리, 개 짖는 소리 등의 잡음까지 들려서 방해가 되는 잡음을 제거 할 수 있다. 단, 회사의 개인적인 기술이기 때문에 어떤 원리로 잡음을 제거하는지는 알 수 없다. 하지만 이 앱 은 엄청나게 많은 양의 음성 데이터를 학습 시켜 인간이 내는 소리와 그 외의 것들을 구분할 수 있는 알고리즘을 구현했을 것으로 추측한다. 그러나 이번 연구에서는 음성 데이터를 축적 시켜 사람의 육성만을 뽑아내는 것이 아니라 음성데이터를 축적하여 외부 잡음이라고 판단되는 소리가 있을 시에 노이즈 캔슬링 프로그램을 작동시키도록 구현 할 것이다. 또한 이 노이즈 캔슬링 프로그램은 FFT 변환을 통해 잡음이라고 판단되는 부분을 확인한 후 필터링을 통해 잡음을 제거하는 프로그램이 될 것이다.

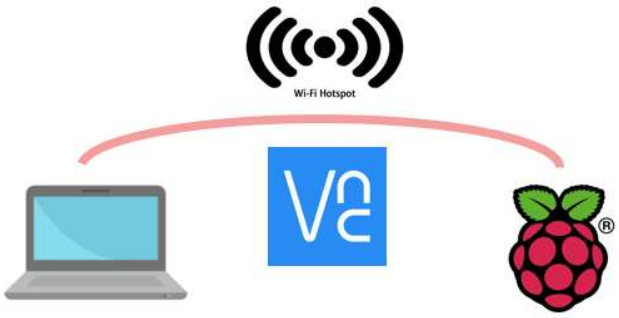
2) 코딩

소프트웨어 코딩을 하기 위해 Python과 Python용 IDE인 Spyder 프로그램을 설치하여 작업을 진행하였다. 또한 제작한 소프트웨어를 넣을 하드웨어인 라즈베리파이를 원활하게 사용하기 위하여 라즈베리파이와 노트북을 핫스팟을 이용하여 연결시켜, 라즈베리파이 환경을 노트북을 이용하여 제어할 수 있도록 하였다. 또한 더 편한 작업을 위하여 친숙한 GUI 환경을 VNC를 이용하여 불러오는 과정을 통하여 그래픽 환경에서 편하게 작업을 진행할 수 있었다.

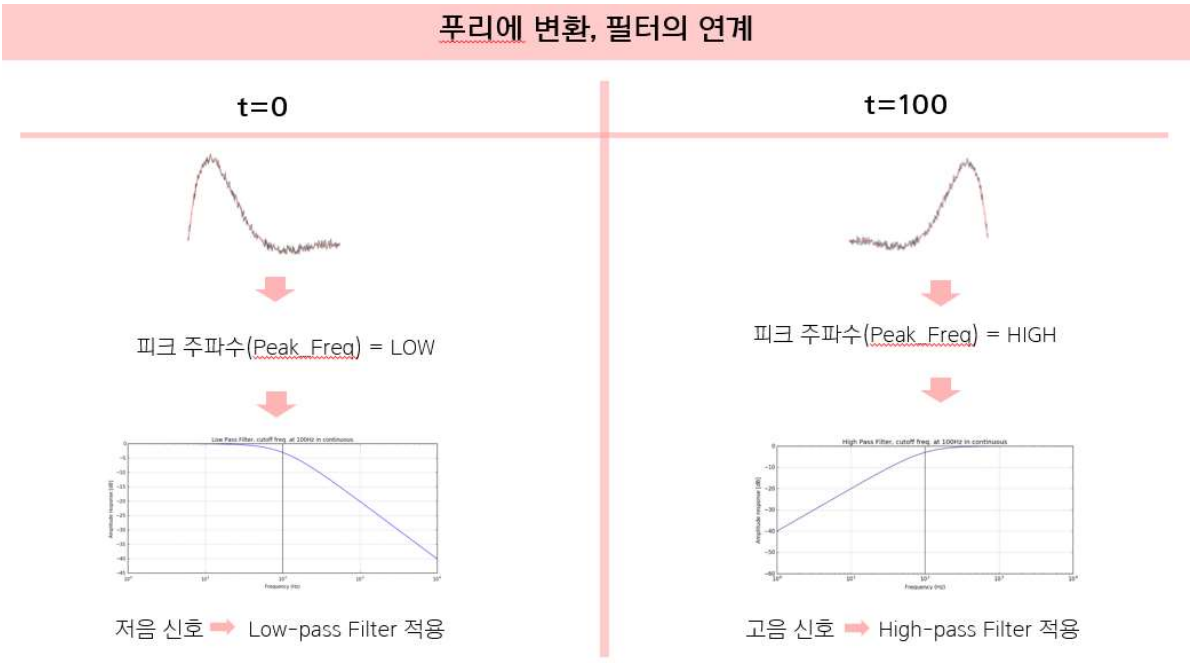


[그림 7] Spyder(좌측)의 구동 모습과 VNC Viewer(우측)의 구동 모습

먼저, 전체적인 시스템을 구성하기 위해 하드웨어의 연결과 소프트웨어 명령어 등을 한 그림에 종합하여 나타내 보았다.



[그림 8] VNC와 핫스팟을 이용한 라즈베리파이와 노트북의 연결 개략도



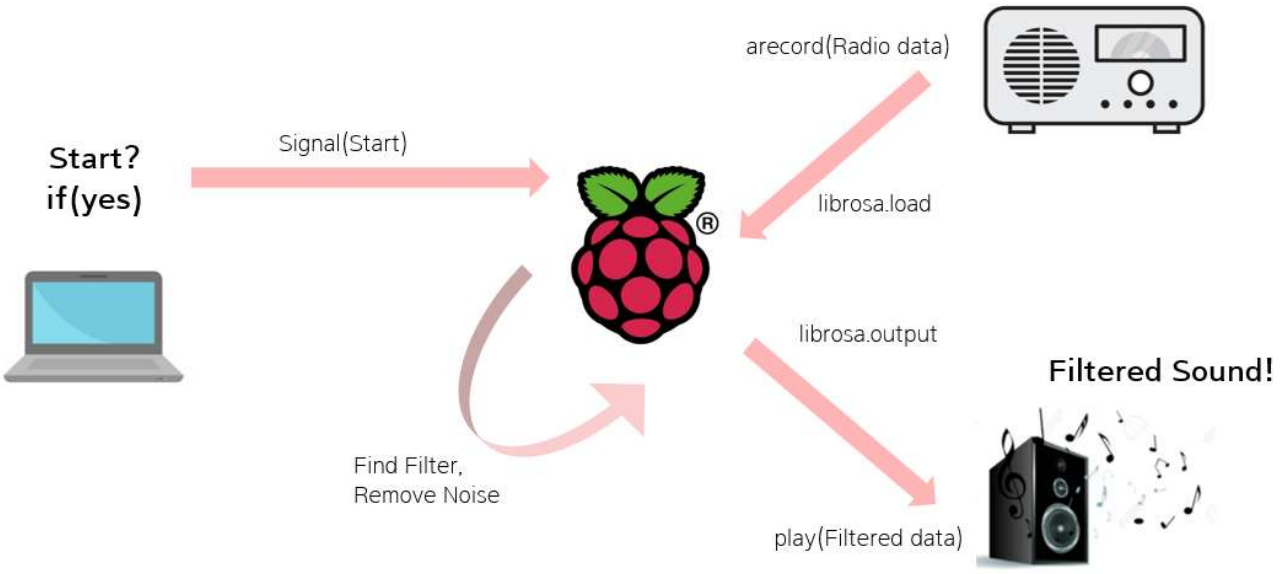
[그림 9] 푸리에 변환의 활용방법

먼저, 푸리에 변환을 직접 Python으로 나타내기는 쉬웠으나, 푸리에 변환인 FFT를 활용한다면 우리가 원하는 시간대별로 다른 주파수 대역을 파악하여 해당 주파수만 패스하는 필터를 적용하는 방법을 적용할 수가 없기에 짧은 시간으로 나누어 연속적으로 FFT를 적용하는 STFT를 고민해볼 수 밖에 없었다. 그러나 실제로 존재하는 STFT 명령어의 경우 특정 DT(짧은 시간)을 지정하기 어려운 문제가 있어 사용하기에는 어려움이 있었다. 그렇기 때문에 우리는 사용하기 쉬운 FFT 알고리즘을 수동으로 STFT알고리즘으로 변환시키기 위해 임의의 시간으로 짧게 주어지는 입력 신호를 나누어 이를 각각 FFT 한 뒤 해당 피크 신호를 해당 시간에 필터로 적용하는 방식을 사용하기로 했다.

먼저, 라즈베리 파이에는 기본적인 녹음 함수와 재생 함수인 arecord와 aplay라는 함수가 존재한다. 또한, 음성 파일을 불러오고 이를 다시 wav 파일로 출력하는 librosa 라이브러리의 여러 함수도 존재한다.

여러 가지 함수

1. arecord : 라즈베리 파이 내장 녹음 함수로, -d 태그를 이용하여 지속시간을 지정하고, 지정하지 않는다면 외부 명령어로 인해 interrupt되기 전까지 녹음된다.
2. librosa.load(File Directory) : wav 파일을 로딩하는 함수로, wav 원본 파일을 비트레이트에 맞게 3차원 복소수의 형태로 array로 출력한다.
3. librosa.output(Name) : 3차원 복소수 형태의 array를 다시 비트레이트에 맞게 wav 파일로 바꾸어 출력하는 함수이다.



[그림 10] 노트북 커맨드를 이용한 전체 시스템의 제어 개략도

그림 10과 같이, 노트북에서 시작하라는 커맨드를 보내면 라즈베리 파이 내장 커맨드인 arecord를 이용하여 연결된 라디오로부터 음성 데이터를 받아온다. 이후 라즈베리파이에서는 음성 데이터를 매우 작은 시간 단위로 쪼개어 FFT를 수행하고, FFT 결과에 따른 필터를 그 국소 시간에 적용하는 방법으로 필터링을 한다. 이후 librosa.output 명령어를 이용하여 필터링

된 소리를 스피커를 통해 출력하는 것이 전체 시스템의 대략적인 관계도이다.

arecord 명령어는 –d를 이용하여 지속시간을 정하지 않으면 계속 녹음이 진행되는 특징이 있다. 그렇기에 원하는 시간만큼 잘라서 녹음을 정지하기 위해서 외부 프로세스를 이용하여 arecord를 정지시키는 과정이 필요하였다. 그렇기에 python을 이용하여 arecord의 pid를 알아내어 이를 interrupt 시키는 방법으로 원하는 시간에 프로세스를 정지시킬 수 있었다.

pid 획득 및 강제종료 함수

1. arecordProcessID = subprocess.Popen('ps | grep arecord',shell = True, stdout = subprocess.PIPE, stderr = subprocess.PIPE)
(pid 획득 명령어)
2. os.kill(pid, 9) (pid 획득 후 프로세스 강제 종료 명령어)

또한, FFT 알고리즘의 경우는 librosa.load를 이용하여 입력받은 복소수 array를 전처리하였다.

```
n = len(data)
    k = np.arange(n)
    T = n/fs
    freq = k/T
    freq = freq[range(int(n/2))]
    FFT_data = fft(data)/n
    FFT_data = FFT_data[range(int(n/2))]
```

이후, FFT된 데이터를 특정 시간인 DT로 잘게 쪼개어 이를 다시 FFT 알고리즘에 대입하였다.

```
i=0
while i in range(int(data_len/combine_num)):
    data_tmp.append(data[combine_num*i:combine_num*i+combine_num])
    tmp_data = data_tmp[i]
    tmp_datafft = np.abs(fft(tmp_data))


    tmp_n = combine_num
    tmp_k = np.arange(tmp_n)
    tmp_T = tmp_n/Fs
    tmp_freq = tmp_k/tmp_T
    tmp_freq = tmp_freq[range(int(tmp_n/2))]
    tmp_FFT_data = fft(tmp_data)/tmp_n
    tmp_FFT_data = tmp_FFT_data[ran 1: int np_n/2))]
    tmp_fftindexraw = (tmp_datafft+1)/2
    tmp_index_fft = (np.argmax(tmp_datafft)+1)/2


    tmp_f_peak = tmp_index_fft
    tmp_w0_peak = 2*np.pi*tmp_f_peak
    tmp_bandWidth = bandlength
    tmp_Q = tmp_f_peak/tmp_bandWidth
    tmp_H = 1/tmp_w0_peak
    tmp_H0 = tmp_H/tmp_Q

    num = np.array([tmp_H0*tmp_w0_peak**2, 0])
    den = np.array([1, tmp_w0_peak/tmp_Q, tmp_w0_peak**2])
    numz, denz = sig.bilinear(num, den , Fs)
    filteredOut = direct2FormModel(tmp_data, denz[1], denz[2], numz[0], numz[1], numz[2])
    total_data.extend(filteredOut)
    i=i+1
```

이후 librosa.output을 이용하여 조작한 데이터를 출력하였다.

3) 하드웨어 부품 제작 및 2D CAD를 이용한 디자인

[그림 11] 라디오 회로를 이용한 USB 라디오 모듈 제작

라즈베리파이에서 기본적으로 마이크 입력을 입력 신호로 받아들이기 때문에 라디오 모듈을 직접 연결하기에는 무리가 있었다. 또한 라디오에서 출력되는 신호를 다시 마이크로 녹음해서 입력 신호로 받다들이기에는 너무 잡음이 많아지므로, 라디오 회로를 분해하여 출력부와 마이크의 입력부를 연결시키는 방법으로 라디오의 소리를 마이크 입력으로 인식되게 만들었다.

소음 제거 소프트웨어를 라즈베리파이에 설치한 후 마이크나 스피커 등 여러 기기를 추가하여 완성된 시제품을 제작해야 하기 때문에 2D CAD을 이용하여 라즈베리파이와 기타 장비가 들어가는 케이스를 제작하였다. 제작한 잡음 제거 라디오의 기본 틀은 기존의 라즈베리파이 케이스에서 아이디어를 가져왔다. 시중에 나온 케이스를 참고하여 직접 수치를 일일이 재서 2D CAD로 도면을 제작하였다.

만든 도면을 응용하여 라디오와 마이크 모듈 그리고 연결된 많은 선이 잘 정리돼서 들어갈 수 있게 수납공간을 만들었고 부피를 최소화하기 위해 3층으로 제작했다. 1층에는 사진에서 볼 수 있듯이 기존의 라즈베리파이를 회로를 넣었고, 2층에는 마이크 모듈, 3층에는 라디오를 부착했다. 라디오의 버튼의 경우 3D프린터를 이용해 버튼 케이스를 만들어 완성된 시제품의 밖에서도 조작이 용이하게 제작하였고, 안테나는 시제품의 천장에 원형의 구멍을 만들어 밖으로 빼 보다 원활한 통신이 가능하게 제작하였다.

[그림 12] 2D CAD를 이용한 라즈베리파이 케이스 제작

1층에는 기존 라즈베리 파이의 케이스 도면을 참고한 라즈베리 파이 거치대를 제작하였고, 2층과 3층에는 추가로 부착해야하는 마이크와 라디오 모듈을 부착하기 위한 거치대를 제작하였다. 특별히 라디오 모듈은 라즈베리 파이의 전력을 사용하는 것이 아니라 외부 전력인 3V 건전지를 사용하므로, 이를 위해서 건전지 거치대용 공간을 따로 만들 필요가 있었다.

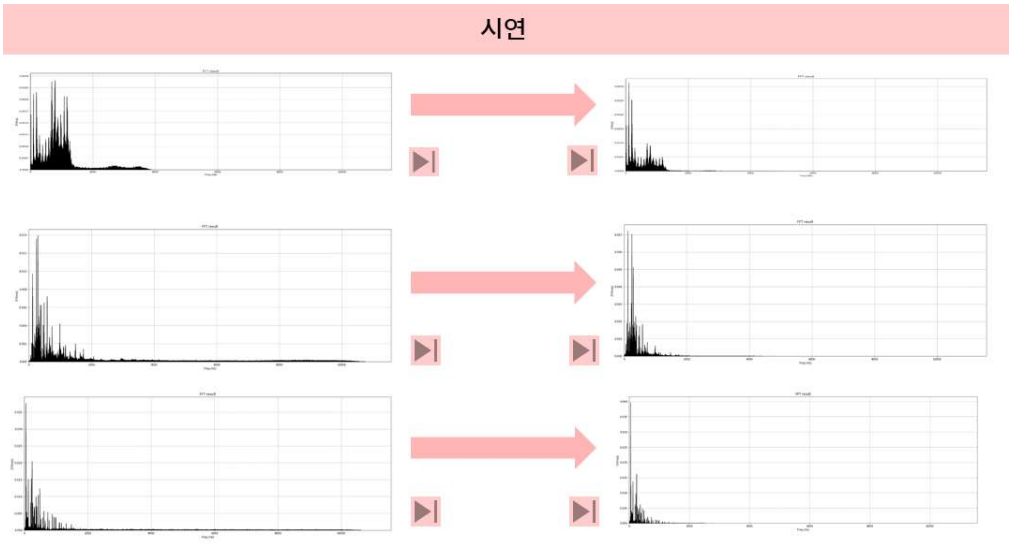
[그림 13] 제작한 라즈베리 파이 케이스에 라즈베리파이와 부품을 장착하는 과정

그림 12와 같이 라즈베리파이 케이스에 라즈베리파이와 라디오마이크, 스피커 등의 부품을 장착하여 가로x세로x높이 각각 55x80x65mm의 직육면체 모양의 소음 제거가 가능한 완성형 라디오를 제작할 수 있었다.

3. 연구 결과

1) 제작한 제품을 통한 소음제거 효과

먼저, 소음 제거 알고리즘은 기존 여러 프로그램에도 존재하는 기능이기에, 객관적인 비교를 하기는 어려웠지만, 시제품 시연을 통한 기능의 우수성을 입증할 수 있었다.



[그림 14] 소음 제거 전의 잡음이 섞인 소리(좌측)과 소음 제거 후의 소리(우측)

먼저, 제작한 시제품을 이용하여 여러 가지 조건을 대입하여 많은 종류의 소리를 비교해 보았다. 원하는 짧은 시간 단위로 STFT를 시행할 수 있기 때문에, 시간 단위인 DT와 Band-Pass Filter의 민감도를 여러 가지 조건을 두고 조절하였다. 또한, 샘플링 소리도 남성 및 여성의 대화 소리, 노래 소리, 클래식 음악, 일렉트로닉 음악 등 여러 가지 음악을 테스트 해보았다. 이를 통해 DT의 경우 0.8~1초 간격, Band Pass Filter의 민감도의 경우 최대 피크 전후로 300Hz 정도로 설정하는 것이 가장 이상적이라는 결과를 얻어내었다.

3. 연구 결론

1) 연구 결론

평소에 입출력 음향 기기를 사용하면서 심심치 않게 들을 수 있었던 음원 이외의 다양한 소리를 필터링을 통해서 어느 정도 소음을 제거하는 것으로 사용자로 하여금 좀 더 좋은 음질의 음원을 제공하는 것이 가능해졌다. 뿐만 아니라 실시간으로 음원 신호를 받는 라디오의 경우 주파수 대에 따라 다양한 노이즈를 갖게 되는데 이런 노이즈를 FFT 변환만으로 노이즈 영역대를 구분하고 이를 토대로 필터링 함으로써 라디오 시청자들이 쾌적한 환경에서 음원을 청취 할 수 있게 되었다.

추가적으로 FFT를 이용하여 STFT를 직접 구현하는 코드를 제작하였는데, 이를 통하여 디지털 신호처리에 관련된 지식을 얻을 수 있었고, 향후 FFT에 대한 이해에 도움이 되었다.

2) 향후 전망

비록 음원의 반대파를 이용하는 노이즈 캔슬링과는 전혀 다른 방향이지만 필터링을 통해 개선 될 수 있는 상황들이 많이 존재한다. 우리는 이런 상황에서 본 조가 연구한 방법을 토대로 좋은 음질의 음원을 제공하거나 소음을 제거할 수 있게 될 것이다. 더불어서 실시간으로 노이즈 캔슬링 해야하는 경우 필요했던 고도의 기술력을 앞서 연구한 방법으로 어느 정도 커버 할 수 있을 것으로 예상된다. 마지막으로 자동차 내부 소음의 경우 흡음제를 사용하여 줄이고 있지만 이런 기술 까지 자동차에 적용한다면 내부 소음을 줄이는데 도움이 될 수 있을 것이다.

4. 소프트웨어 코드 원문

```
# coded by celenort

#definition
def toHz(value):
    from numpy import pi
    return value/2/pi

def direct2FormModel(data, a1, a2, b0, b1, b2):
    from numpy import zeros, arange

    result = zeros((len(data),))
    timeZone = zeros((len(data),))

    for n in arange(2, len(data)):
        sum0 = -a1*timeZone[n-1] - a2*timeZone[n-2]
        timeZone[n] = data[n] + sum0
        result[n] = b0*timeZone[n] + b1*timeZone[n-1] + b2*timeZone[n-2]

    return result
def draw_FFT_Graph(data, fs, **kwargs):
    import matplotlib.pyplot as plt
    from numpy.fft import fft
```

```
    graphStyle = kwargs.get('style', 0)
    xlim = kwargs.get('xlim', 0)
    ylim = kwargs.get('ylim', 0)
    title = kwargs.get('title', 'FFT result')

    n = len(data)
    k = np.arange(n)
    T = n/fs
    freq = k/T
    freq = freq[range(int(n/2))]
    FFT_data = fft(data)/n
    FFT_data = FFT_data[range(int(n/2))]

    plt.figure(figsize=(12,5))
    if graphStyle == 0:
        plt.plot(freq, abs(FFT_data), 'r', linestyle=' ', marker='^')
    else:
        plt.plot(freq,abs(FFT_data),'r')
    plt.xlabel('Freq (Hz)')
    plt.ylabel('|Y(freq)|')
    plt.vlines(freq, [0], abs(FFT_data))
    plt.title(title)
    plt.grid(True)
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.show()

#import
import numpy as np
#import matplotlib.pyplot as plt                    #matplotlib (for draw_FFT_Graph)
import scipy.signal as sig
import librosa
from numpy.fft import fft
# Create Signal Info
Fs = 22.05*10**3                # 22050Hz (Sampling Rate)
Ts = 1/Fs                       # sample Time
bandlength = 300                # band length(band filter)
combine_num = 7000              # a number of unit for fft
input_dir =  'bp3_nc_in.wav'
```

```
output_dir = 'bp3_nc_out_10000.wav'

#create var
data_tmp = []
total_data = []

#load data
data,sr = librosa.load(input_dir)
            # input audio data
data_len = len(data)
            # data_len = data's length


# while -> fft
i=0
while i in range(int(data_len/combine_num)):
            # while (i < data_length / a unit for fft)
    data_tmp.append(data[combine_num*i:combine_num*i+combine_num])
            # combining data
    tmp_data = data_tmp[i]
            # saving "i" th data to data_tmp
    tmp_datafft = np.abs(fft(tmp_data))
            # fft -> abs data -> save it to tmp_datafft


    tmp_n = combine_num
    tmp_k = np.arange(tmp_n)
    tmp_T = tmp_n/Fs
    tmp_freq = tmp_k/tmp_T
    tmp_freq = tmp_freq[range(int(tmp_n/2))]
    tmp_FFT_data = fft(tmp_data)/tmp_n
    tmp_FFT_data = tmp_FFT_data[range(int(tmp_n/2))]
    tmp_fftindexraw = (tmp_datafft+1)/2
    tmp_index_fft = (np.argmax(tmp_datafft)+1)/2


    tmp_f_peak = tmp_index_fft
    tmp_w0_peak = 2*np.pi*tmp_f_peak
    tmp_bandWidth = bandlength
    tmp_Q = tmp_f_peak/tmp_bandWidth
```

```
    tmp_H = 1/tmp_w0_peak
    tmp_H0 = tmp_H/tmp_Q
            # calculating for band pass filter

    num = np.array([tmp_H0*tmp_w0_peak**2, 0])
    den = np.array([1, tmp_w0_peak/tmp_Q, tmp_w0_peak**2])
    numz, denz = sig.bilinear(num, den , Fs)
    filteredOut = direct2FormModel(tmp_data, denz[1], denz[2], numz[0], numz[1], nu
mz[2])
    total_data.extend(filteredOut)
             # save ffted file to total_data
    i=i+1

#file wrap up and make output
                                  # i++
final_wave=np.array(total_data)
             # make numpy list -> python array
librosa.output.write_wav(output_dir, final_wave, 22050)
            # make output file
```