ANC 구현 및 실패 원인 분석

30503 구현승

탐구 동기 및 목적

2학년 물리학1 시간에 파동의 간섭에 대하여 직접 소리를 녹음하고 파형을 나타내어 보강 간섭과 상쇄 간섭을 구현했었다. 이 중 상쇄 간섭은 실패로 끝났었다. 상쇄 간섭이 현재 우리의 삶에 뗄래야 뗄 수 없는 존재이기에 원리를 탐구하고 이를 심화하여 노이즈 캔슬링 기술을 구현해보려 한다.

이론적 배경

노이즈 캔슬링은 외부 잡음을 상쇄, 혹은 차단하는 기술이다. 노이즈 캔슬링의 넓은 의미는 소음을 차단하는 것을 의미하며 그 방식에 따라 액티브 노이즈 캔슬링, 패시브 노이즈 캔슬링으로 나뉜다.
노이즈 캔슬링은 소음을 수집 및 분석하고 반대 파동을 생성하기 때문에 주변 소음에 실시간으로 대응하기가 어렵다. 이런 이유로 규칙적이고 지속적인 소음엔 효과적이지만 순간적으로 발생하는 문 닫는 소리, 자동차 경적 소리, 재채기나 박수 소리 등 불규칙하고 지속적이지 않은 소음은 내부 회로가 대응하기 전에 없어지기 때문데 제대로 걸러내지 못한다. 또한 고음보다는 저음에 강하다. 고음은 파동 사이의 간격이 짧은데 저음은 파동 사이의 간격이 길다. 따라서 어느 정도 시간이 걸려도 충분히 상쇄할 수 있다.

PNC(Passive Noise Cancellation)은 인위적으로 발생시킨 상쇄간섭을 이용한 소음 제거 기술이 아닌 에너지를 물리적으로 차단하는 물질들을 이용해 소리가 차단되도록 하는 것을 의미한다.
예를 들어 방음재나 흡음재, 귀마개 등이 있다.

ANC(Active Noise Cancellation)은 소리의 상쇄 간섭 현상을 이용한 기술을 의미한다.
소리는 일정한 형태를 갖는 파동의 일종이며 이와 반대되는 파동을 같은 시간에 발생시키면 서로 상쇄되는데, 이를 상쇄간섭이 일어났다고 한다. 이러한 소리의 특성을 이용해서 제거하려는 소음의 위상 및 진폭을 파악하고 이와 완전히 반대되는 위상과 진폭을 연산하여 인위적으로 발생시켜 소음을 제거하는 기술이다.

푸리에 변환은 “시간에 대한 함수를 주파수에 대한 함수로 변환하는 것”이다. 예를 들어 생즙을 보고 안에 어떤 재료(과일이나 채소)가 있는지 맞추는 것은 굉장히 어려운 일 일 것이다. 그런데 만약에 분석 장치(생즙을 원심분리한 후 몇 가니 물리적/화학적 반응을 거쳐서 원재료를 분석하는 기계)를 만들어 재료들을 맞춘다면…? 푸리에 변환이 이런 거와 비슷한 거라 할 수 있다. 어떤 복잡한 신호가 있는데, 이 복잡한 신호가 단순한 신호 뭐뭐로 구성되었는지를 알려주는 변환기 뭐 이런 것이다.

노이즈 캔슬링에서의 푸리에 변환은 특정 소리의 파형을 진폭과 주파수로 표현해 소리를 디지털로 표현하는 것이다. 이를 통해 소리가 어떤 구성요소를 가지고 있는지 파악할 수 있고 주파수 별 진폭의 크기를 알 수 있어 노이즈 캔슬링 같은 필터링 작업을 하는데 필요합니다.

ANC 구현 및 실패 원인 분석

30503 구현승

탐구 방법 및 기간

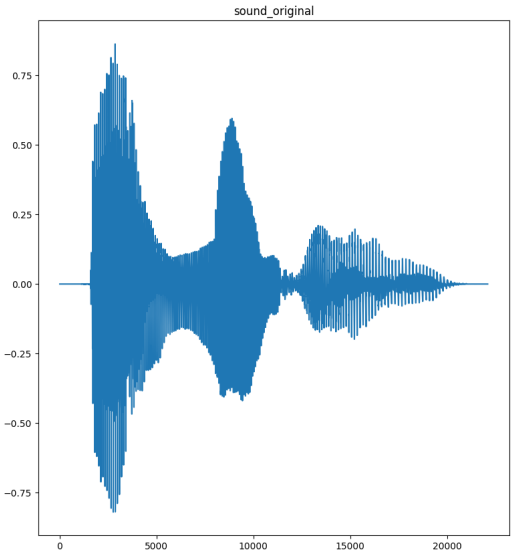
8월 15일 ~ 8월 17일

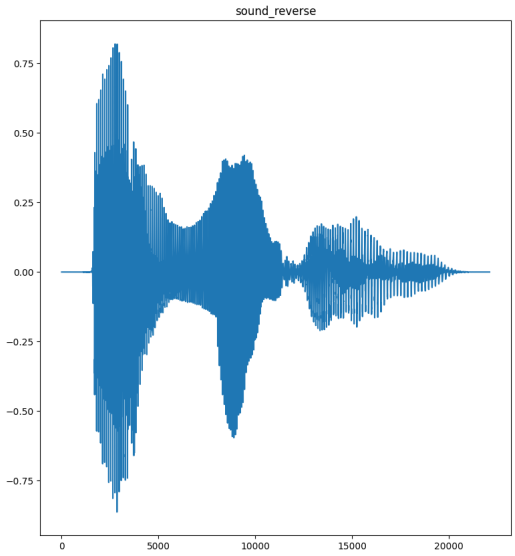
세 가지 방법으로 노이즈 캔슬링을 구현 해볼 것이고 한 가지 방법은 원음에다가 -1을 곱하여 원음과 더하여 구현할 것이고 다른 한 가지 방법은 푸리에 변환을 사용하여 노이즈 성분을 구별하고 제거할 수 있는 방법을 사용할 것이다. 마지막으로는 실시간으로 입력된 소음을 노이즈 캔슬하는 것을 구현 해볼 것이다.

처음으로 +1과 -1을 더하면 0이 되는 제로섬과 같다고 보면 된다.
음성 자료에서 intensity는 기본적으로 1~ -1 범위 내의 float 형으로 표시할 수 있다.
그렇다면 원래의 소리에 -1을 곱해서 동일한 강도면서 진동의 방향은 반대인 음원을 만들 수 있을 것이다.
그리고 원래의 소리와 -1을 곱한 소리를 더하면 0, 즉 아무 소리도 안날 것이다.

```python
import soundfile
import IPython
import matplotlib.pyplot as plt

# 사운드파일 불러오기

soundfile_path = './audio/iloveyou_kakao.mp3'
sound, sampling_rate =
soundfile.read(soundfile_path)
## sound = 사운드파일의 강도를 담은 배열
## sampling_rate = 재생 속도

# 역위상 음원 만들기
sound_reverse = sound * (-1)
```

```python
# 원음과 역위상 음원 그래프로 비교

plt.figure(figsize = (20, 10))
plt.subplot(1, 2, 1)
plt.plot(range(len(sound)), sound)
plt.title('sound_original')
plt.subplot(1, 2, 2)
plt.plot(range(len(sound_reverse)), sound_reverse)
plt.title('sound_reverse')
plt.show()
```



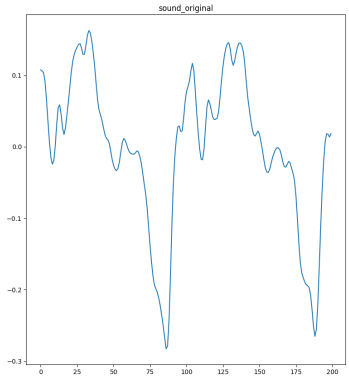


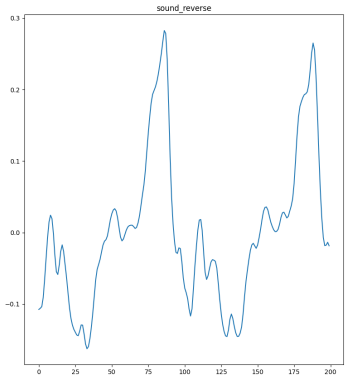
ANC 구현 및 실패 원인 분석

30503 구현승

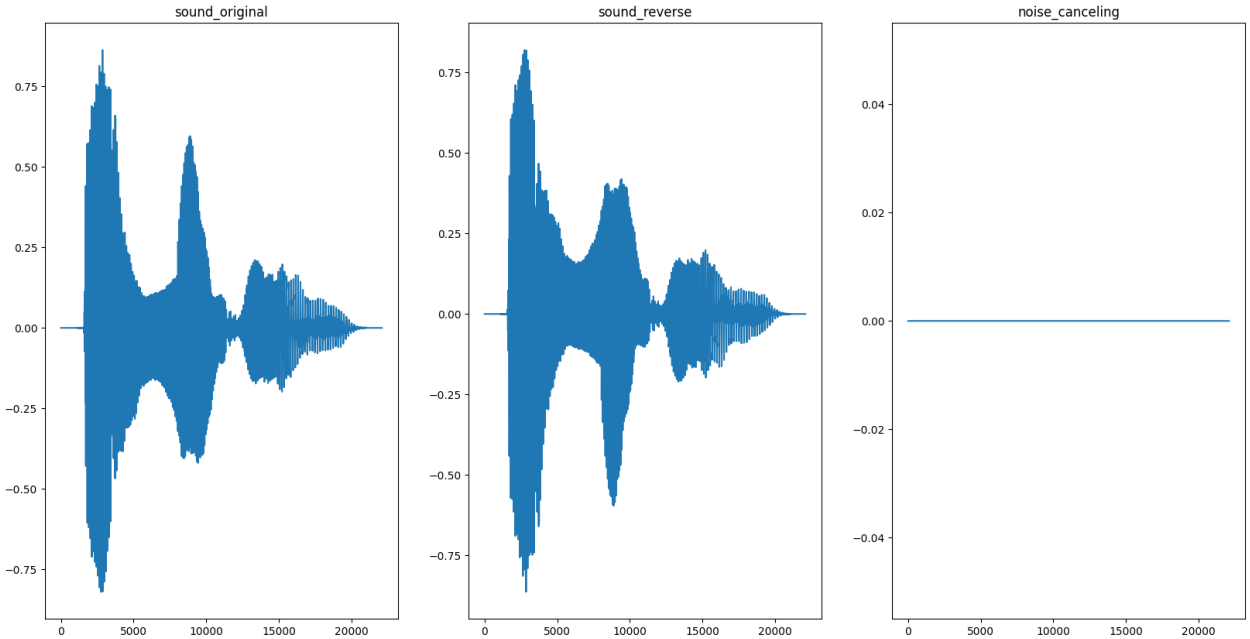
사운드 파일을 불러오고 그 파일에 -1을 곱하여 역위상 음원 그래프로 나타낸 뒤 그래프를 그린 그래프이다. 어라? 근데 -1을 곱했는데 그래프가 비슷한데…?
음성은 1과 -1 범위 내에서 진동한다는 특징이 있다. 즉 이는 -1을 곱한다고 하더라도 강도를 음수로 표현했을 뿐 동일한 진폭을 갖는다는 건 사실이다. 그래서 들리는 소리 자체는 동일하게 들리는 게 아닐까 추측한다. 시각화 그래프는 언뜻 보면 비슷하게 생겼지만, 확대해서 보면 다르다.

```
# 5000 ~ 5200 부분만 그래프로 그려보기

start = 5000
end = 5200

plt.figure(figsize = (20, 10))
plt.subplot(1, 2, 1)
plt.plot(range(len(sound[start:end])), sound[start:end])
plt.title('sound_original')
plt.subplot(1, 2, 2)
plt.plot(range(len(sound_reverse[start:end])),
sound_reverse[start:end])
plt.title('sound_reverse')
plt.show()
```





원음과 역위상 음을 더해서 나타내어 그래프로 나타내면…



다행히 성공이다.

ANC 구현 및 실패 원인 분석

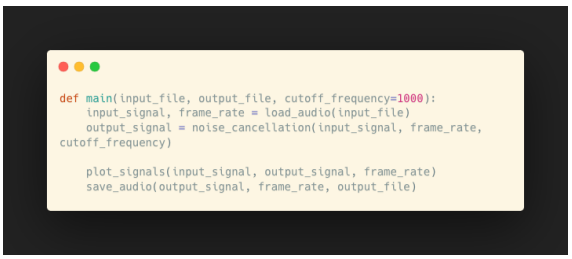
30503 구현승

두 번째 방법은 푸리에 변환을 이용한 방식이다.
오디오 신호는 시간 도메인에서 다양한 주파수 성분이 혼합된 신호다. 푸리에 변환을 사용하면 이 신호를 주파수 도메인으로 변환할 수 있어, 각 주파수 성분을 분석하고 조작할 수 있다. 또한 역 푸리에 변환을 사용하여 다시 시간 도메인으로 변환할 수 있다. 이 과정에서 원치 않는 노이즈 성분이 제거된 신호를 얻을 수 있다. 중요한 코드를 보며 살펴보겠다.

```
def noise_cancellation(input_signal, frame_rate,
cutoff_frequency=1000):
    # Fourier Transform을 사용하여 주파수 영역으로 변환
    signal_freq = rfft(input_signal)
    frequencies = rfftfreq(len(input_signal), d=1/frame_rate)

    # 특정 주파수 이상의 성분을 제거
    signal_freq[frequencies > cutoff_frequency] = 0

    # 역 Fourier Transform을 사용하여 시간 영역으로 변환
    output_signal = irfft(signal_freq)
    return output_signal
```

이 코드는 노이즈를 제거하는 함수이다.
-rfft(input_signal): 입력 신호를 주파수 도메인으로 변환한다.
-rfftfreq(len(input_signal), d=1/frame_rate): 주파수 축을 생성한다.

- signal_freq[frequencies > cutoff_frequency] = 0: 주파수 필터링을 수행하여 특정 주파수 이상의 성분을 제거한다.
- irfft(signal_freq): 필터링된 주파수 성분을 시간 도메인 신호로 변환한다.



```
def main(input_file, output_file, cutoff_frequency=1000):
    input_signal, frame_rate = load_audio(input_file)
    output_signal = noise_cancellation(input_signal, frame_rate,
cutoff_frequency)

    plot_signals(input_signal, output_signal, frame_rate)
    save_audio(output_signal, frame_rate, output_file)
```

이 코드는 메인 함수이다.

-load_audio: 입력 파일에서 오디오 신호를 로드한다.

- noise_cancellation: 노이즈를 제거한 신호를 생성한다.
- plot_signals: 결과를 시각화한다

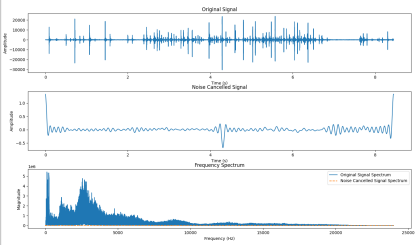
이 코드는 오디오 파일에서 노이즈를 제거하기 위해 fourier 변환을 이용하여 주파수 필터링을 수행하고, 원본 신호와 노이즈가 제거된 신호를 시각화하고 저장하는 기능을 제공한다.

cutoff_frequency(이하 cutoff)는 신호 필터링에서 어떤 주파수 이상의 성분을 제거할지를 결정하는 기준점 정도로 보면 된다. 이 값을 적절히 설정하면 불필요한 노이즈 성분을 제거하고, 중요한 신호 성분을 유지할 수 있다.

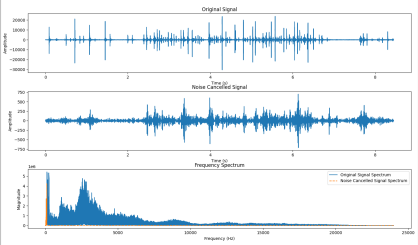
ANC 구현 및 실패 원인 분석

30503 구현승

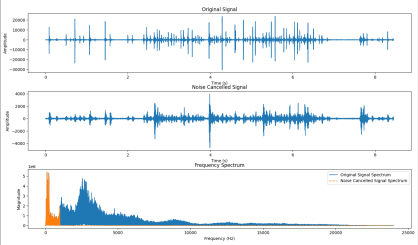
cutoff = 10



cutoff = 100



cutoff = 1000



signal_freq[frequencies > cutoff_frequency] = 0

* 여기서 cutoff는 위 코드를 사용함으로써 입력된 주파수가 cutoff의 값을 뛰어넘을 때 노이즈를 제거하라는 뜻임.

그래프를 설명해보자면 제일 위에 있는 그래프는 입력 오디오의 시각적 파형이다. x축을 시간, y축을 세기로 두었다. 두 번째 그래프는 노이즈 캔슬이 된 그래프이다. 입력된 원음을 푸리에 변환을 통해 시간의 영역을 주파수의 영역으로 바꾼 다음 cutoff 이상인 주파수 성분을 제거한다. 그리고 역푸리에 변환을 통해 주파수 영역에서 시간 영역으로 바꾼 그래프이다. 이를 주파수 필터링이라고 한다. 마지막 그래프는 주파수 스펙트럼을 그래프로 나타낸 것이다. cutoff가 원음에서 어떤 영향을 주었는지 알게 해주는 그래프이다.

*노이즈 캔슬링된 음원을 들어보니 원음과 차이가 있었다.(하단 구글 드라이브 파일 링크)

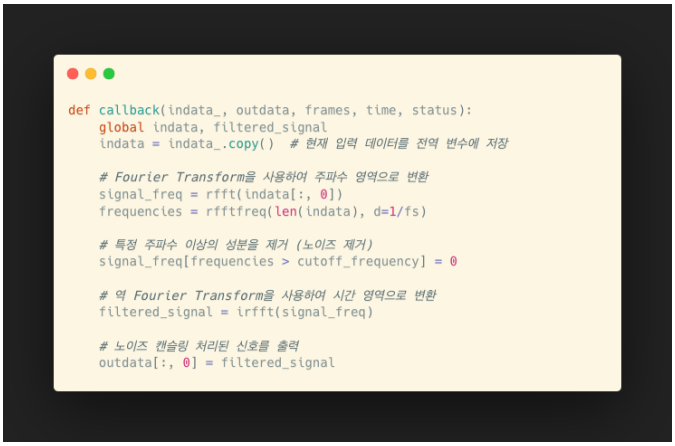
ANC 구현 및 실패 원인 분석

30503 구현승

마지막 방법으로 실시간으로 입력되는 노이즈를 캔슬하는 것을 구현해보겠다.
(이하는 중요코드 만을 다룬다.)

```python
fs = 44100  # 샘플링 레이트 (Hz)
blocksize = 1024  # 한 번에 처리할 샘플의 개수
cutoff_frequency = 1000  # 노이즈 캔슬링 필터의 차단 주파수 (Hz)

# 오디오 콜백 함수
def callback(indata_, outdata, frames, time, status):
    global indata, filtered_signal
    indata = indata_.copy()  # 현재 입력 데이터를 전역 변수에 저장
```

이 코드는 전역변수 설정이다.
fs는 샘플링 레이트로 음파를 캡처하여 디지털 오디오로 변환하는 속도이다.
blocksize는 버퍼의 크기라고도 할 수 있는데 cpu가 해야할 일의 양을 정의한 것이라고도 볼 수 있다. 콜백 함수는 실시간으로 받아들여지는 음성을 처리하기 위해서 사용된다.



```python
def callback(indata_, outdata, frames, time, status):
    global indata, filtered_signal
    indata = indata_.copy()  # 현재 입력 데이터를 전역 변수에 저장

    # Fourier Transform을 사용하여 주파수 영역으로 변환
    signal_freq = rfft(indata[:, 0])
    frequencies = rfftfreq(len(indata), d=1/fs)

    # 특정 주파수 이상의 성분을 제거 (노이즈 제거)
    signal_freq[frequencies > cutoff_frequency] = 0

    # 역 Fourier Transform을 사용하여 시간 영역으로 변환
    filtered_signal = irfft(signal_freq)

    # 노이즈 캔슬링 처리된 신호를 출력
    outdata[:, 0] = filtered_signal
```

콜백함수는 indata에 현재 실시간으로 입력되고 있는 오디오 신호를 복사하고 rfft를 사용하여 오디오 신호를 주파수 영역으로 변환시킨다.
위에서 말했다시피 cutoff를 사용함으로써 특정 주파수 이상의 성분을 제거하고 역푸리에 변환을 사용하여 주파수 도메인을 시간 도메인으로 바꾸어 출력한다.

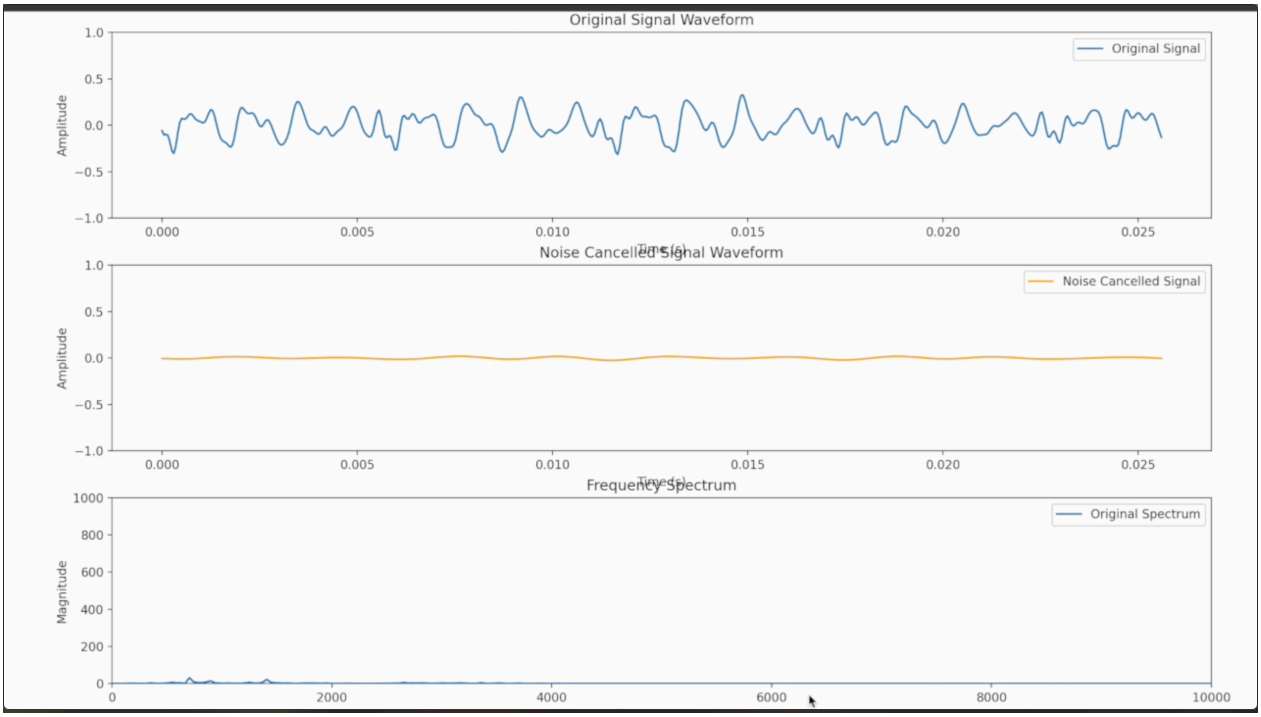
```python
def update_plot(frame):
    if indata.size > 0:
        original_line.set_ydata(indata[:, 0])
        filtered_line.set_ydata(filtered_signal)

        # 주파수 스펙트럼 업데이트
        signal_freq = rfft(indata[:, 0])
        spectrum_line.set_ydata(np.abs(signal_freq))

    return original_line, filtered_line, spectrum_line
```

그래프를 실시간으로 움직이는 것을 구현하기 위한 코드이다. (if indata.size > 0:이 의미하는 것은 현재 입력이 되고 있냐를 체크하는 것이다.)

전체 코드이다.
https://github.com/Gugo-le/PHY2-ANC/blob/main/check.py

ANC 구현 및 실패 원인 분석

30503 구현승



첫 번째 그래프는 입력 데이터의 waveform, 파형을 나타낸 것이다. 위에 방법들 처럼 주파수로 나타내려 했지만 실시간 데이터에 맞게 더 보기 좋게 waveform 형태로 나타내어주었다. 두 번째 그래프는 실시간으로 노이즈 캔슬이 되고 있는 그래프를 나타내어 준다. 이 세 번째 방법은 두 번째 방법에서 실시간 기능만 추가한 것으로 cutoff를 사용하여 실시간으로 구현하게 해주었다. 인풋 -> 인풋 복사 -> 인풋 그래프 출력 -> 푸리에 변환 - > cutoff 적용 -> 역푸리에 변환 -> 아웃풋 -> 아웃풋 그래프 출력, 이런 방식이다.

마지막으로 세번째 그래프는 주파수 스펙트럼을 시각화하기 위한 그래프이다. 특정 주파수에서 신호의 크기가 높다면 그 주파수 성분이 신호에서 많이 차지하고 있다는 것을 의미한다. 마지막 그래프를 보면 살짝 튀어나와 있는 부분이 있는데 현재 입력되고 있는 인풋에 강한 신호라는 뜻이다. 저 그래프를 보면서 노이즈 제거를 더욱 효율적이게 할 수 있다.

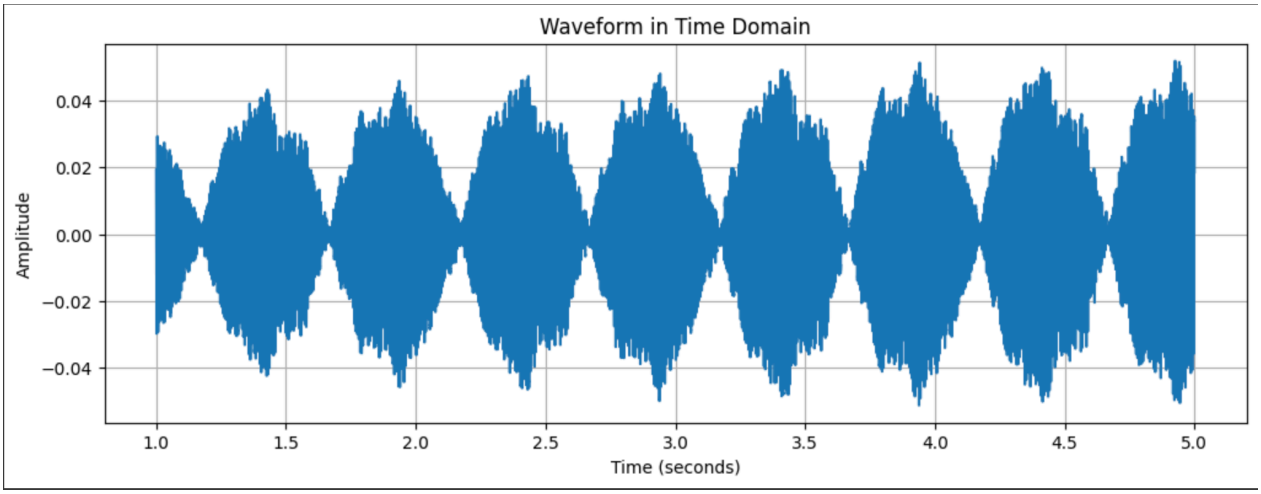
결론

주파수는 신호가 1초에 몇번 진동했는지 나타내는 수치이고, 소리는 주파수가 높을수록 높은 음으로 들리게 된다. 모든 음성은 다양한 주파수 성분들의 합으로도 표현될 수 있다. 즉, 푸리에 변환을 통해서 특정 시간 길이의 음성 조각들이 각각의 주파수 성분들을 얼마나 가지고 있는지를 의미하는 스펙트럼을 얻을 수 있다.
우리는 일상생활에서 무선 이어폰의 노이즈 캔슬링 기능을 사용하기도 한다. 이는 소리 파동에 대하여 진폭이 반전된 음파를 출력하여 서로 상쇄되게 하여 소리를 제거하는 기술이다. 이때, 소리의 파형을 알아내는 것이 필요한데 푸리에 변환을 통해서 가능하다. 푸리에 변환을 통해 얻어낸 주파수의 스펙트럼을 시계열로 나열하면 시간 변화에 따른 스펙트럼의 변화를 알 수 있다. 소리에 들어있는 각각의 주파수 성분들을 푸리에 변환을 통해 추출하는 방식으로 청취한 소리에 내재된 정보를 얻는 것이다. 음성 녹음 파일에서 거슬리는 불쾌한 고음이 존재할 때, 우리는 푸리에 변환을 사용할 수 있다. 녹음된 음성을 “magnitude - time” 함수로 나타낼 수 있다. 이를 푸리에 변환을 적용하여 시간에 대한 함수를 주파수에 대한 함수로 변환한다. 이때 높은 주파수 영역에서 나타나는 성분들을 제거해주면은 소음이 없이 녹음된 것과 같이 만들 수 있다.

ANC 구현 및 실패 원인 분석

30503 구현승

실패 원인 분석(2학년)



본래 이 음원을 상쇄 간섭을 사용해 노이즈 캔슬링 기술을 구현해보려 하였다.
2학년 때는 교과 시간에 배운 파동의 간섭을 사용하여 최대한 구현해보려고 위에 첫 번째 방법을 사용했었다.
위 그래프의 파형을 반전시켜 둘이 더하면 되지 않을까?(왜냐면 보강 간섭에서는 두 그래프 더하면 되니까)
저 말도 맞는 말이다. 2학년 때의 내가 코드를 잘못 짰을 뿐이지..

2학년 때의 코드이다.

```
audio_data2[::-1]
```

[::-1]이 의미하는 것은 리스트, 문자열, 혹은 배열 등의 순서를 뒤집는 것이다.(맨 앞 ':'이 의미하는 것은 모든 행을 의미한다.) 예를 들어 [1, 2, 3, 4] -> [4, 3, 2, 1]이 되는 것이다. 정보 교과 시간에 슬라이싱이라는 것을 배우고 그 때 깨달았다. 내가 뭘 잘못했는지. 사실 Numpy 라이브러리를 이용해서 오디오 신호를 숫자로 리스트화하고 그걸 다시 오디오 신호로 반환한다면 어쩌면 가능했을지 모르겠지만 이는 매우 복잡한 과정이다.

위에 첫 번째 방법은 엄청 간단히 그래프에 단지 -1 하나 곱해서 역위상 그래프를 만들 수 있었다.
그리고 2학년 때 생각처럼 둘이 더했을 때 상쇄 간섭을 구현할 수 있었다.

ANC 구현 및 실패 원인 분석

30503 구현승

느낀점

노이즈 캔슬링 구현을 위해 이 고군분투한 과정은 단순한 학습을 넘어, 물리학의 이론적 이해와 컴퓨터 과학적 실습을 결합한 문제 해결 경험이었다. 2학년 때, 상쇄 간섭을 이용한 노이즈 캔슬링에 도전하였지만, 코드 구현 과정에서 발생한 문제로 인해 원하는 결과를 얻지 못하였었다. 그 때의 나는 좌절을 느꼈었지만 동시에 더 깊은 이해와 개선의 필요성을 느끼게 해주었다. 노이즈 캔슬링에 대하여 더 깊게 탐구함으로써 푸리에 급수라는 것을 알게 되었고 이를 통해 주파수 분석의 중요성을 깨닫게 되었고 푸리에 급수를 통해 신호를 개별 주파수 성분으로 분해하고, 이를 필터링하여 불필요한 노이즈를 제거하는 과정은 노이즈 캔슬링의 핵심 원리를 보다 명확하게 이해할 수 있도록 해주었다.. 또한 실패를 두려워하지 않고, 실패를 통해 배운 것을 바탕으로 다시 도전하는 자세가 얼마나 중요한지도 알게 되었다.

** https://drive.google.com/drive/folders/1lxZpJ9aG4GflJZAGw7fPlr4PFlVdwpNX?usp=sharing

참고자료

https://greenacademy.re.kr/%ec%9e%90%ec%97%b0%ec%b2%a0%ed%95%99-%ec%84%b8%eb%af%b8%eb%82%98?mod=document&uid=96

https://namu.wiki/w/%EB%85%B8%EC%9D%B4%EC%A6%88%20%EC%BA%94%EC%8A%AC%EB%A7%81

https://people-analysis.tistory.com/152