4월 4주차 보고서 (2024.04.22 ~ 2024.04.28)

💡 2024.04.15(월) 18:00 ~ 23:00 / 오프라인 회의 / 참여자: 김지윤, 박수진, 정지원, 이다은

💡 2024.04.16(화) 18:00 ~ 23:00 / 오프라인 회의 / 참여자: 김지윤, 박수진, 정지원, 이다은

💡 2024.04.17(수) 20:00 ~ 23:00 / 오프라인 회의 / 참여자: 김지윤, 박수진, 정지원, 이다은

💡 2024.04.26(금) 18:00 ~ 23:00 / 오프라인 회의 / 참여자: 김지윤, 정지원

💡 2024.04.27(토) 09:00 ~ 14:00 / 오프라인 회의 / 참여자: 김지윤, 박수진, 정지원, 이다은

💡 2024.04.28(일) 09:00 ~ 16:00 / 오프라인 회의 / 참여자: 김지윤, 박수진, 이다은

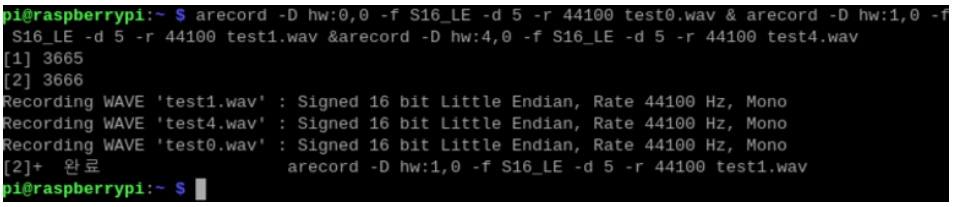
😄하드웨어 part

<4/24>

하드웨어 part - 동시녹음

3개 동시녹음

```
arecord -D hw:0,0 -f S16_LE -d 5 -r 44100 test0.wav &
arecord -D hw:1,0 -f S16_LE -d 5 -r 44100 test1.wav &
arecord -D hw:4,0 -f S16_LE -d 5 -r 44100 test4.wav
```



usb허브를 라즈베리파이에 연결하여 usb포트를 확장하였다.

확장한 포트에 사운드카드 3개를 연결한 후 마이크를 연결하여 동시 녹음 되는 것을 확인하였다.

하드웨어 part - 실시간입출력(1개)

```
arecord -l
```

→ sound card 번호 - 0번 확인

sudo nano /etc/asound.conf

```
pcm.!default {
    type hw
    card 0
    device 0
}

ctl.!default {
    type hw
    card 0
}
```

```
import pyaudio
import numpy as np

# 파이오디오 객체 생성
p = pyaudio.PyAudio()

# 입력 설정
CHUNK = 512  # 더 작은 CHUNK 크기
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
```

```
# 마이크 장치 선택 (여기서는 0번 카드의 0번 디바이스를 사용)
DEVICE_INDEX = 0

# 출력 설정
output_stream = p.open(format=FORMAT,
                       channels=CHANNELS,
                       rate=RATE,
                       output=True,
                       frames_per_buffer=CHUNK)

# 입력 스트림 열기
input_stream = p.open(format=FORMAT,
                      channels=CHANNELS,
                      rate=RATE,
                      input=True,
                      input_device_index=DEVICE_INDEX,
                      frames_per_buffer=CHUNK)

print("Recording and playing...")

try:
    while True:
        # 오디오 데이터 읽기
        data = input_stream.read(CHUNK)
        # 바이너리 데이터를 numpy 배열로 변환
        audio_data = np.frombuffer(data, dtype=np.int16)

        # 오디오 데이터 출력
        output_stream.write(data)
except KeyboardInterrupt:
    print("Stopped")

# 스트림 닫기
input_stream.stop_stream()
input_stream.close()
output_stream.stop_stream()
output_stream.close()

# 파이오디오 종료
p.terminate()
```

→라즈베리파이에서 마이크 소리를 실시간으로 출력하는 방법

다음 일정 : 두개 이상 마이크 연결해서 소리 실시간으로 출력하기

<4/25>

마이크 2개 이상 동시 입출력 테스트

- 사운드카드 2개 연결
  - sudo nano /etc/asound.conf

```
pcm.mic0 {
    type hw
    card 0
    device 0
}
```

```
pcm.mic1 {
    type hw
    card 3
    device 0
}

pcm.speaker0 {
    type hw
    card 0
    device 0
}

pcm.speaker1 {
    type hw
    card 3
    device 0
}

pcm.!default {
    type asym
    playback.pcm "speaker"
    capture.pcm "mic"
}

ctl.!default {
    type hw
    card 0
}
```

```
import pyaudio
import numpy as np

# 파이오디오 객체 생성
p = pyaudio.PyAudio()

# 입력 설정
CHUNK = 1024  # CHUNK 크기
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100

# 마이크와 스피커 장치 선택
MIC_DEVICE_INDEX_1 = 0  # 첫 번째 마이크 장치
MIC_DEVICE_INDEX_2 = 3  # 두 번째 마이크 장치
SPEAKER_DEVICE_INDEX_1 = 0  # 첫 번째 스피커 장치
SPEAKER_DEVICE_INDEX_2 = 3  # 두 번째 스피커 장치

# 출력 설정
output_stream_1 = p.open(format=FORMAT,
                         channels=CHANNELS,
                         rate=RATE,
                         output=True,
                         frames_per_buffer=CHUNK,
                         output_device_index=SPEAKER_DEVICE_INDEX_1)

output_stream_2 = p.open(format=FORMAT,
```

```
                          channels=CHANNELS,
                          rate=RATE,
                          output=True,
                          frames_per_buffer=CHUNK,
                          output_device_index=SPEAKER_DEVICE_INDEX_2)

# 입력 스트림 열기
input_stream_1 = p.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        input_device_index=MIC_DEVICE_INDEX_1,
                        frames_per_buffer=CHUNK)

input_stream_2 = p.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        input_device_index=MIC_DEVICE_INDEX_2,
                        frames_per_buffer=CHUNK)

print("Recording and playing...")

try:
    while True:
        # 오디오 데이터 읽기
        data_1 = input_stream_1.read(CHUNK)
        data_2 = input_stream_2.read(CHUNK)

        # 바이너리 데이터를 numpy 배열로 변환
        audio_data_1 = np.frombuffer(data_1, dtype=np.int16)
        audio_data_2 = np.frombuffer(data_2, dtype=np.int16)

        # 오디오 데이터 출력
        output_stream_1.write(data_1)
        output_stream_2.write(data_2)
except KeyboardInterrupt:
    print("Stopped")

# 스트림 닫기
input_stream_1.stop_stream()
input_stream_1.close()
output_stream_1.stop_stream()
output_stream_1.close()

input_stream_2.stop_stream()
input_stream_2.close()
output_stream_2.stop_stream()
output_stream_2.close()

# 파이오디오 종료
p.terminate()
```

```
import pyaudio
import numpy as np

# 파이오디오 객체 생성
```

```
p = pyaudio.PyAudio()

# 입력 설정
CHUNK = 512  # 더 작은 CHUNK 크기
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100

# 마이크 장치 선택 (카드 번호와 장치 번호는 실제 장치에 맞게 설정해야 함)
MIC_DEVICES = [(0, 0), (3, 0), (4, 0), (5, 0)]

# 출력 설정
output_stream = p.open(format=FORMAT,
                       channels=CHANNELS,
                       rate=RATE,
                       output=True,
                       frames_per_buffer=CHUNK)

# 입력 스트림 열기
input_streams = []
for card, device in MIC_DEVICES:
    input_stream = p.open(format=FORMAT,
                          channels=CHANNELS,
                          rate=RATE,
                          input=True,
                          input_device_index=p.get_device_count() - 1,
                          frames_per_buffer=CHUNK)
    input_streams.append(input_stream)

print("Recording and playing...")

try:
    while True:
        # 각 입력 스트림에서 오디오 데이터 읽기
        for input_stream in input_streams:
            data = input_stream.read(CHUNK)
            # 바이너리 데이터를 numpy 배열로 변환
            audio_data = np.frombuffer(data, dtype=np.int16)
            # 오디오 데이터 출력
            output_stream.write(data)

except KeyboardInterrupt:
    print("Stopped")

# 입력 스트림 닫기
for input_stream in input_streams:
    input_stream.stop_stream()
    input_stream.close()

# 출력 스트림 닫기
output_stream.stop_stream()
output_stream.close()

# 파이오디오 종료
p.terminate()
```

<4/26>

다이나믹 마이크 연결 성공

<4/27>

다이나믹마이크 동시 입출력 (1)

0427_0.py

```
#동시 입출력 테스트3
import pyaudio
import numpy as np

# 파이오디오 객체 생성
p = pyaudio.PyAudio()

# 장치 목록 출력
print("Available audio devices:")
for i in range(p.get_device_count()):
    device_info = p.get_device_info_by_index(i)
    print(f"Device {i}: {device_info['name']} - {device_info['maxInputChannels']} input chanr

# 파라미터 설정
CHUNK = 512  # 데이터 읽기 버퍼 크기
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100

# 마이크 입력 장치 인덱스
MICROPHONE_INDEXES = [0, 3]  # 두 개의 마이크 인덱스 (필요에 따라 변경 가능)

# 마이크 입력 스트림 생성
input_streams = []

# 장치 정보 확인 및 스트림 생성
for index in MICROPHONE_INDEXES:
    # 장치 정보 확인
    device_info = p.get_device_info_by_index(index)

    # 입력 채널 수가 1 이상인지 확인
    if device_info["maxInputChannels"] < 1:
        print(f"Device {index} does not support input channels. Skipping...")
        continue

    # 스트림 생성
    try:
        input_stream = p.open(
            format=FORMAT,
            channels=CHANNELS,
            rate=RATE,
            input=True,
            input_device_index=index,
            frames_per_buffer=CHUNK,
        )
        input_streams.append(input_stream)
        print(f"Opened input stream for device {index}: {device_info['name']}")
    except OSError as e:
        print(f"Error opening stream for device {index}: {e}")
```

```
# 입력 데이터 처리
print("Recording from multiple microphones...")

try:
    while True:
        mic_data = []

        for i, stream in enumerate(input_streams):
            try:
                # 데이터 읽기
                data = stream.read(CHUNK, exception_on_overflow=False)
                audio_data = np.frombuffer(data, dtype=np.int16)
                mic_data.append(audio_data)
            except OSError as e:
                print(f"Error reading from stream {i}: {e}")

        # 입력 데이터가 모두 0이 아닌지 확인
        if mic_data:
            for i, audio in enumerate(mic_data):
                if np.all(audio == 0):
                    print(f"Microphone {i} has no input data.")
                else:
                    print(f"Microphone {i}: {audio[:10]}")
        else:
            print("No data received from microphones.")

except KeyboardInterrupt:
    print("Recording stopped.")

finally:
    # 스트림 닫기
    for stream in input_streams:
        try:
            stream.stop_stream()
            stream.close()
        except OSError as e:
            print(f"Error closing stream: {e}")

# 파이오디오 종료
p.terminate()
```

→ 오류 수정 중

<4/28>

다이나믹마이크 동시 입출력 (2)

다이나믹 마이크 연결시 사운드카드와의 호환성 문제로 노이즈가 굉장히 심하게 발생하는 이슈 발생 → 다시 핀마이크로 테스트 진행

노이즈제거부분에서 파일로 진행해야 하기 때문에 녹음과 동시에 실시간 동시 출력 코드 작성함

10초마다 실시간 출력을 하면서 녹음파일을 시간대별로 저장하도록 함

arecord -l → 사운드카드 디바이스 번호 0, 3번

0428_0.py

```
import pyaudio
import numpy as np
import wave
import time
```

```
from datetime import datetime

# PyAudio 객체 생성
p = pyaudio.PyAudio()

# 예외 처리로 스트림 설정
try:
    # 첫 번째 마이크 입력 스트림 설정
    stream1 = p.open(
        format=pyaudio.paInt16,
        channels=1,  # 모노 입력
        rate=44100,
        input=True,
        input_device_index=0,  # 첫 번째 마이크
        frames_per_buffer=2048  # 버퍼 크기 증가
    )

    # 두 번째 마이크 입력 스트림 설정
    stream2 = p.open(
        format=pyaudio.paInt16,
        channels=1,  # 모노 입력
        rate=44100,
        input=True,
        input_device_index=3,  # 두 번째 마이크
        frames_per_buffer=2048  # 버퍼 크기 증가
    )

    # 첫 번째 스피커 출력 스트림 설정
    output_stream1 = p.open(
        format=pyaudio.paInt16,
        channels=1,  # 모노 출력
        rate=44100,
        output=True,
        output_device_index=0,  # 첫 번째 스피커
        frames_per_buffer=2048
    )

    # 두 번째 스피커 출력 스트림 설정
    output_stream2 = p.open(
        format=pyaudio.paInt16,
        channels=1,  # 모노 출력
        rate=44100,
        output=True,
        output_device_index=3,  # 두 번째 스피커
        frames_per_buffer=2048
    )

except Exception as e:
    print("스트림을 열 수 없습니다:", e)
    p.terminate()
    exit(1)

# 녹음을 위한 프레임 버퍼 생성
recorded_frames1 = []
recorded_frames2 = []

# 10초 타이머 초기화
start_time = time.time()
```

```
try:
    while True:
        # 첫 번째 마이크에서 데이터 읽기
        data1 = stream1.read(1024)

        # 두 번째 마이크에서 데이터 읽기
        data2 = stream2.read(1024)

        # 첫 번째 스피커로 데이터 출력
        output_stream1.write(data1)

        # 두 번째 스피커로 데이터 출력
        output_stream2.write(data2)

        # 녹음 프레임 버퍼에 추가
        recorded_frames1.append(data1)
        recorded_frames2.append(data2)

        # 10초마다 녹음 파일 생성
        if time.time() - start_time >= 10:
            # 파일 이름을 현재 시간으로 생성
            filename1 = datetime.now().strftime("mic1_%Y%m%d_%H%M%S.wav")
            filename2 = datetime.now().strftime("mic2_%Y%m%d_%H%M%S.wav")

            # 첫 번째 마이크 녹음 파일 생성
            with wave.open(filename1, "wb") as wf1:
                wf1.setnchannels(1)  # 모노
                wf1.setsampwidth(p.get_sample_size(pyaudio.paInt16))
                wf1.setframerate(44100)
                wf1.writeframes(b"".join(recorded_frames1))

            # 두 번째 마이크 녹음 파일 생성
            with wave.open(filename2, "wb") as wf2:
                wf2.setnchannels(1)  # 모노
                wf2.setsampwidth(p.get_sample_size(pyaudio.paInt16))
                wf2.setframerate(44100)
                wf2.writeframes(b"".join(recorded_frames2))

            # 프레임 버퍼 초기화
            recorded_frames1.clear()
            recorded_frames2.clear()

            # 타이머 재설정
            start_time = time.time()

        # 짧은 대기 시간
        time.sleep(0.1)  # 데이터 읽기 간격 조절

except KeyboardInterrupt:
    # 리소스 정리
    stream1.stop_stream()
    stream2.stop_stream()
    stream1.close()
    stream2.close()
    output_stream1.close()
```

```
    output_stream2.close()
    p.terminate()
```

→ 오류가 너무 많이 발생하여 수정 중

```
(myenv) pi@raspberrypi:~/myenv/test $ python 0428_0.py
ALSA lib pcm_asym.c:105:(_snd_pcm_asym_open) capture slave is not defined
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.front.0:CARD
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM front
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.rear
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.center_lfe
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.side
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround51.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround21
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround51.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround21
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround40.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround40
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround51.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround41
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround51.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround50
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround51.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround51
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.surround71.0
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM surround71
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.iec958.0:CAR
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM iec958
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.iec958.0:CAR
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM spdif
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.iec958.0:CAR
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM spdif
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.modem
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.modem
```

```
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.phoneline
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.phoneline
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp
ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp
ALSA lib pcm_a52.c:1001:(_snd_pcm_a52_open) a52 is only for playback
ALSA lib confmisc.c:1369:(snd_func_refer) Unable to find definition 'cards.2.pcm.iec958.0:CAR
ALSA lib conf.c:5180:(_snd_config_evaluate) function snd_func_refer returned error: No such f
ALSA lib conf.c:5703:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM iec958:{AES0 0x6 AES1 0x82 AES2 0x0 A
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib pcm_dmix.c:999:(snd_pcm_dmix_open) unable to open slave
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Expression 'ret' failed in 'src/hostapi/alsa/pa_linux_alsa.c', line: 1736
Expression 'AlsaOpen( &alsaApi->baseHostApiRep, params, streamDir, &self->pcm )' failed in 's
Expression 'PaAlsaStreamComponent_Initialize( &self->playback, alsaApi, outParams, StreamDire
Expression 'PaAlsaStream_Initialize( stream, alsaHostApi, inputParameters, outputParameters,
스트림을 열 수 없습니다: [Errno -9985] Device unavailable
```

😊 소프트웨어 PART

RMS(Root Mean Square) 계산

- 4개의 녹음 파일을 입력으로 준다.
- 4개의 녹음 파일에 대한 각각의 RMS 값을 계산한다.
- 4개의 각각의 RMS 값의 평균 RMS 값을 계산한다.
- 4개의 녹음 파일을 평균 RMS 값으로 변환하여 각각 저장한다.

RMS 계산

```
def calculate_rms(audio):
    return np.sqrt(np.mean(audio**2))
```

볼륨 조정

```
def adjust_volume(audio, target_rms, current_rms):
    return audio * (target_rms / current_rms)
```

파일 처리 및 볼륨 조정 로직

```
def process_files(file_paths):

    audios = []  # 로드된 오디오 신호와 샘플링 레이트를 저장할 리스트
    rms_values = []  # 각 오디오 파일의 RMS 값을 저장할 리스트

    # 1. 음성 파일 로드 및 RMS 계산
    for file_path in file_paths:
        audio, sr = librosa.load(file_path, sr=None)  # 파일 로드
        load_end = time.time()  # 파일 로드 종료 시간
        rms = calculate_rms(audio)  # RMS 계산
        rms_values.append(rms)  # RMS 값 저장
        audios.append((audio, sr))  # 오디오 신호와 샘플링 레이트 저장

    # 2. RMS 평균값 계산
    average_rms = np.mean(rms_values)

    # 3. 각 파일의 볼륨을 조정하고 저장
    for idx, (audio, sr) in enumerate(audios):
        current_rms = rms_values[idx]
        adjusted_audio = adjust_volume(audio, average_rms, current_rms)  # 볼륨 조정
        original_name = file_paths[idx].split(".")[0]  # 원본 파일 이름 추출
        output_path = f"RMS_{original_name}.wav"  # 저장할 파일의 이름 생성
        sf.write(output_path, adjusted_audio, sr)  # 조정된 오디오 파일 저장
```

- 결과

```
[Running] python -u "c:\Users\idaeu\기본\바탕 화면\2024캡스톤\voice-separation\tempCodeRunnerFile.py"
1.wav RMS: 0.10502325743436813 (Loaded in 1.32 seconds)
2.wav RMS: 0.1195312961935997 (Loaded in 0.00 seconds)
3.wav RMS: 0.047599293291568756 (Loaded in 0.00 seconds)
4.wav RMS: 0.041791271418333054 (Loaded in 0.00 seconds)
Average RMS: 0.07848627865314484
File RMS_1.wav saved with adjusted RMS to the average (Adjusted in 0.01 seconds)
File RMS_2.wav saved with adjusted RMS to the average (Adjusted in 0.01 seconds)
File RMS_3.wav saved with adjusted RMS to the average (Adjusted in 0.01 seconds)
File RMS_4.wav saved with adjusted RMS to the average (Adjusted in 0.00 seconds)
Total processing time: 1.35 seconds

[Done] exited with code=0 in 1.695 seconds
```

노이즈 제거 및 변환 전후 파형 출력

- RMS 계산 코드에서 노이즈를 제거하고 녹음 파일의 원본과 변환 후의 파형 출력

노이즈 제거

```
def remove_noise(audio, sr):
    noise_clip = audio[0:int(sr * 0.5)]
    noise_reduced_audio = nr.reduce_noise(y=audio, sr=sr, y_noise=noise_clip)
    return noise_reduced_audio

def process_files(file_paths):
        ...
```

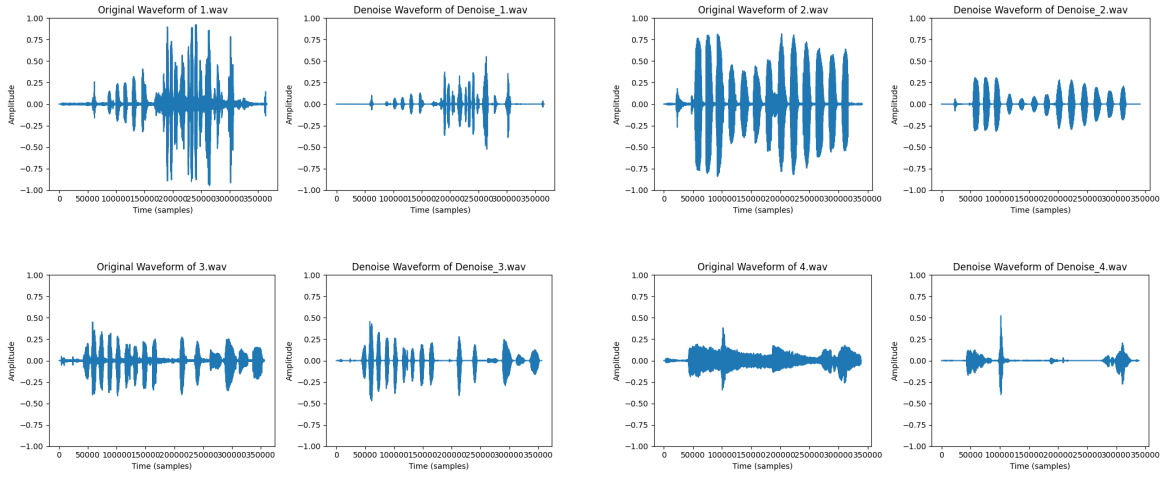
```
        for idx, (audio, sr) in enumerate(audios):
                current_rms = rms_values[idx]
                adjusted_audio = adjust_volume(audio, average_rms, current_rms)
                noise_removed_audio = remove_noise(adjusted_audio, sr)
                original_name = file_paths[idx].split(".")[0]
                output_path = f"Denoise_{original_name}.wav"
                sf.write(output_path, noise_removed_audio, sr)
                output_paths.append(output_path)
        ...
```

파형 출력

```
def plot_and_save_waveforms(original_files, Denoise_files):
    for i, (original, Denoise) in enumerate(zip(original_files, Denoise_files)):
        y_original, sr_original = librosa.load(original, sr=None)
        y_Denoise, sr_Denoise = librosa.load(Denoise, sr=None)

        plt.figure(figsize=(10, 4))
        plt.subplot(1, 2, 1)
        plt.plot(y_original)
        plt.title(f'Original Waveform of {original}')
        plt.xlabel('Time (samples)')
        plt.ylabel('Amplitude')
        plt.ylim([-1, 1])

        plt.subplot(1, 2, 2)
        plt.plot(y_Denoise)
        plt.title(f'Denoise Waveform of {Denoise}')
        plt.xlabel('Time (samples)')
        plt.ylabel('Amplitude')
        plt.ylim([-1, 1])

        plt.tight_layout()
        plt.savefig(f'Waveform_{original.split(".")[0]}.png')
        plt.close()
```

- 결과

```
[Running] python -u "c:\Users\idaeu\기본\바탕 화면\2024캡스톤\voice-separation\Denoise.py"
1.wav RMS: 0.10502325743436813 (Loaded in 0.20 seconds)
2.wav RMS: 0.1195312961935997 (Loaded in 0.00 seconds)
3.wav RMS: 0.047599293291568756 (Loaded in 0.00 seconds)
4.wav RMS: 0.041791271418333054 (Loaded in 0.00 seconds)
Average RMS: 0.07848627865314484
File Denoise_1.wav saved with adjusted RMS to the average (Adjusted in 0.19 seconds)
File Denoise_2.wav saved with adjusted RMS to the average (Adjusted in 0.18 seconds)
File Denoise_3.wav saved with adjusted RMS to the average (Adjusted in 0.17 seconds)
File Denoise_4.wav saved with adjusted RMS to the average (Adjusted in 0.15 seconds)
Total processing time: 0.90 seconds

[Done] exited with code=0 in 2.52 seconds
```



실시간 음성 입출력

- 입력과 출력을 하나의 곳으로 설정하여 실시간 음성 입출력 실험

오디오 설정

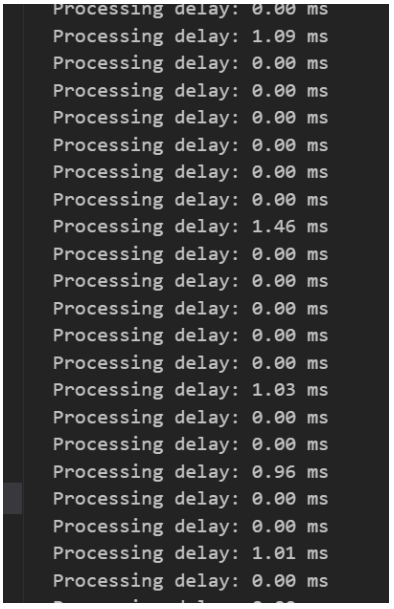
```
# 오디오 데이터 형식 및 기본 설정
FORMAT = pyaudio.paFloat32
CHANNELS = 1
RATE = 44100
CHUNK = 256  # 한 번에 처리할 샘플 수
global_average_rms = 0.02  # 전역 평균 RMS 초기값, 적절한 값을 설정
```

실시간 처리

```
def stream_callback(in_data, frame_count, time_info, status):

    global global_average_rms
    audio_data = np.frombuffer(in_data, dtype=np.float32)
    current_rms = calculate_rms(audio_data)

    # 볼륨 조정 로직
    adjusted_data = adjust_volume(audio_data, global_average_rms, current_rms)
    adjusted_data = adjusted_data.astype(np.float32).tobytes()

    print(f"Processing delay: {(end_time - start_time) * 1000:.2f} ms")  # 밀리초 단위로 딜레이 출

    return (adjusted_data, pyaudio.paContinue)

def main():
    pa = pyaudio.PyAudio()
    stream = pa.open(format=FORMAT,
                     channels=CHANNELS,
                     rate=RATE,
                     input=True,
                     output=True,
                     frames_per_buffer=CHUNK,
```

```
                    stream_callback=stream_callback)

    # 스트림 시작
    stream.start_stream()

    try:
        while stream.is_active():
            time.sleep(0.1)  # CPU 사용 감소
    except KeyboardInterrupt:
        # 사용자에 의해 종료되었을 때
        print("Stream stopped by user.")

    finally:
        # 스트림 종료 및 자원 정리
        stream.stop_stream()
        stream.close()
        pa.terminate()

if __name__ == "__main__":
    main()
```

- 결과



💡 <문제점 및 해결 방안>

1. delay 발생

   한 번에 처리할 수 있는 샘플 수를 조절하여 실시간 반응을 향상 시킨다.

   → 높이면, 사운드가 끊어지지는 않지만 실시간 반응이 느려진다.

   → 줄이면, 사운드가 끊어질 수는 있지만 실시간 반응이 빨라진다. (그럼에도 delay 발생)

2. 낮은 음질

   추가적인 조사를 통해 해결 방안 모색

3. 심한 소음

   PiDTLN 알고리즘을 통해 적용한 결과를 분석해 볼 계획이다.

실시간 노이즈 제거

- 실시간 음성 입출력 코드에서, 노이즈 제거 후 출력

> 노이즈 제거

```
def spectral_subtraction(signal, noise_estimation, alpha=4):
    transformed_signal = np.fft.rfft(signal)
    subtracted_spectrum = np.maximum(transformed_signal - alpha * noise_estimation, 0)
    cleaned_signal = np.fft.irfft(subtracted_spectrum, n=len(signal))
    return cleaned_signal

class AudioProcessor:
```

```
    def __init__(self, target_rms=0.02, update_interval=10):
        self.noise_profile = None
        self.initial_noise_data = []
        self.initial_noise_frames = int(RATE / CHUNK * 9999999999999999)  # 초기 9999999999999
        self.update_frames = int(RATE / CHUNK * update_interval)  # 노이즈 프로파일 업데이트 간격
        self.frames_processed = 0
        self.target_rms = target_rms

    def estimate_noise_profile(self, audio_data, force_update=False):
        self.initial_noise_data.append(audio_data)
        self.frames_processed += 1

        if len(self.initial_noise_data) >= self.initial_noise_frames or force_update:
            noise_data = np.concatenate(self.initial_noise_data)
            self.noise_profile = np.abs(np.fft.rfft(noise_data, n=CHUNK))
            self.initial_noise_data = []  # 데이터 클리어

        # 주기적으로 노이즈 프로파일 업데이트
        if self.frames_processed % self.update_frames == 0:
            self.estimate_noise_profile(audio_data, force_update=True)

    def process_audio_data(self, in_data):
        audio_data = np.frombuffer(in_data, dtype=np.float32)
        if self.noise_profile is None or self.frames_processed < self.initial_noise_frames:
            self.estimate_noise_profile(audio_data)
            return in_data  # 노이즈 프로파일이 충분히 설정되기 전에는 원본 데이터 반환
        else:
            cleaned_data = spectral_subtraction(audio_data, self.noise_profile)
            current_rms = calculate_rms(cleaned_data)
            adjusted_data = adjust_volume(cleaned_data, self.target_rms, current_rms)
            return adjusted_data.astype(np.float32).tobytes()
```

- 결과

> 💡 <문제점 및 해결 방안>
> 1. 초 단위를 설정한 만큼만 노이즈 제거가 실행된다.
> → 연속적으로 노이즈 제거가 이루어지지 않는다. 현재 초 설정을 크게 해놓은 상태로, 연속적으로 노이즈 제거가 가능하기는 하다.

> ✅ 하나로 입출력 모두 사용하여 실시간 처리를 진행해 보았다. → 라즈베리파이에서 입력과 출력을 각각 설정하여 개발을 진행할 것이다. 그렇기에 입력과 출력을 각각 설정하여 다시 실험을 진행해보자. 그렇다면 딜레이가 줄어들지 않을까?

---

https://github.com/SaneBow/PiDTLN

- 해당 GitHub에서 제공하는 파일 중, DTLN 서비스를 이용할 수 있도록 하는 아래의 코드를 제공하고 있다. (dtln_ns.service 파일)

```
[Unit]
Description=DTLN Noise Suppression Service
After=pulseaudio.service
```

```
[Service]
Type=simple
ExecStart=/usr/local/bin/ns -o 'Loopback 0' -D

[Install]
WantedBy=default.target
```

이를 아래와 같이 수정하고, 경로(/home/malab-c/.config/systemd/user/dtln_ns.service)를 변경하였다.

```
[Unit]
Description=DTLN Noise Suppression Service
After=pulseaudio.service

[Service]
Type=simple
ExecStart=/home/malab-c/anaconda3/envs/yolov7/bin/python3 /home/malab-c/yolov7/PiDTLN/PiDT

[Install]
WantedBy=default.target
```

이후, 서비스 활성을 위해 다음 과정을 거쳤다.

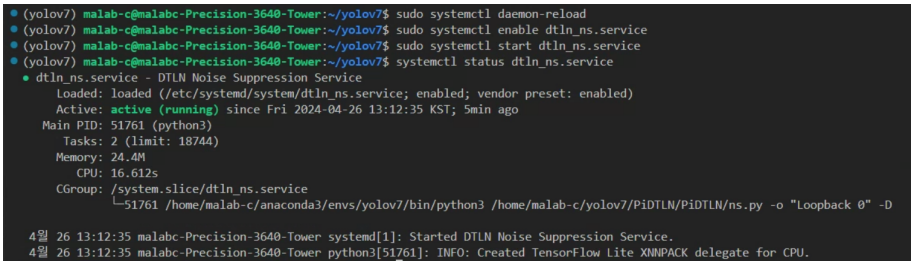
📖 <시스템 수준에서의 동작>

sudo systemctl daemon-reload

sudo systemctl enable dtln_ns.service

sudo systemctl start dtln_ns.service

systemctl status dtln_ns.service



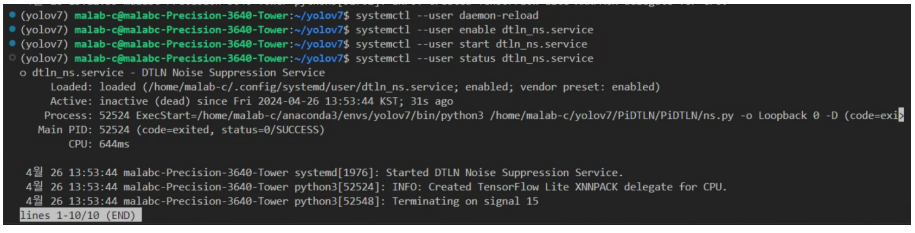
📖 <사용자 수준에서의 동작>

systemctl --user daemon-reload

systemctl --user enable dtln_ns.service

systemctl --user start dtln_ns.service

systemctl --user status dtln_ns.service



⇒ 시스템 수준에서는 서비스가 잘 동작하지만, 사용자 수준에서는 서비스가 동작하지 않는다.

⇒ 사용자 수준에서, 서비스가 inactive(dead) 상태로 바로 전환되는 것은 ns.py가종료되었다는것이다.

<ns.py 파일 수정>

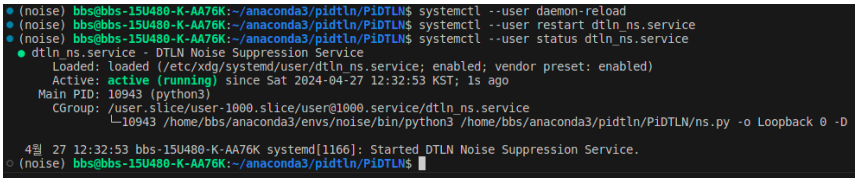
```
def main_loop():
    logging.info("Starting main loop.")
    with sd.Stream(device=(args.input_device, args.output_device),
                   samplerate=fs_target, blocksize=block_shift,
                   dtype=np.float32, latency=args.latency,
                   channels=(1, 1), callback=callback):
        try:
            while True:
                time.sleep(1)
                logging.info("Running...")
        except KeyboardInterrupt:
            logging.info("Interrupted by user")
        except Exception as e:
            logging.error("An error occurred: %s", str(e))

def main():
    logging.info("Service started.")
    if args.daemonize:
        with daemon.DaemonContext():
            main_loop()
    else:
        main_loop()

if __name__ == '__main__':
    main()
```

우리들의 서버

💡 다시, 우리의 서버에서 진행해야 한다.

<dtln_ns.service 파일 수정>

```
[Unit]
Description=DTLN Noise Suppression Service
After=pulseaudio.service

[Service]
Type=simple
ExecStart=/home/bbs/anaconda3/envs/noise/bin/python3 /home/bbs/anaconda3/pidtln/PiDTLN/ns.py
Restart=always
RestartSec=5

[Install]
WantedBy=default.target
```



✅ 샘플링 수 수정 및 입출력 채널에 대한 코드 수정이 전반적으로 필요

DTLN

GitHub - TechyNilesh/Speech-Enhancement-Noise-Suppression-Using-DTLN: Speech Enhancement: Tensorflow 2.x implementation of the
Speech Enhancement: Tensorflow 2.x implementation of the stacked dual-signal transformation LSTM network (DTLN) for Noise Suppression. - TechyNilesh

https://github.com/TechyNilesh/Speech-Enhancement-Noise-Suppression-Using-DTLN/tree/main

잡음 억제를 위한 스택형 이중 신호 변환 LSTM 네트워크(DTLN)의 Tensorflow 2.x

DTLN(Dual-Transfer Learning Network)이란? 음성 신호의 잡음을 제거하고 클린한 음성을 재구성하기 위한 딥러닝 기반의 모델. 이 모델은 노이즈가 섞인 음성 입력을 받아 잡음을 제거하고 원본 음성을 복원하는 역할 함.

DTLN의 두 가지 주요 네트워크:

1. **디노이징 네트워크(Denoising Network)**: 이 네트워크는 입력으로부터 잡음을 제거하고 클린한 음성을 추출하는 역할.
2. **음성 재구성 네트워크(Voice Reconstruction Network)**: 이 네트워크는 디노이징 네트워크에서 생성된 클린한 음성을 사용하여 원래의 음성을 복원. 이 과정에서 클린한 음성과 잡음이 혼합된 입력 음성을 원래의 클린한 상태로 되돌림.

Installing Reuired library

`pip install -r requirements.txt`

```
SoundFile>=0.12.1
numpy==1.26.4
librosa==0.10.0
lxml==4.9.4
wavinfo==2.3.0
tensorflow==2.16.1
keras2onnx==1.7.0
onnxruntime==1.17.3
sounddevice==0.4.6
```

최신 버전에 맞게 라이브러리 버전을 수정하여 설치 진행.

Importing Defining Audio Generator Classes

```
class audio_generator():
    '''
    대규모 오디오 데이터셋으로부터의 반복자를 기반으로 TensorFlow 데이터셋을 생성하는 클래스입니다.
    이 오디오 생성기는 단일 채널 오디오 파일만 지원합니다.
    '''

    def __init__(self, path_to_input, path_to_s1, len_of_samples, fs, train_flag=False):
        '''
```

```
        오디오 생성기 클래스의 생성자입니다.
        입력:
            path_to_input       혼합 오디오 파일 경로
            path_to_s1          대상 소스 데이터 경로
            len_of_samples      샘플링된 오디오 조각의 길이 (샘플 단위)
            fs                  샘플링 속도
            train_flag          파일을 섞을 플래그 활성화 여부
        '''
        # 입력값을 클래스 속성으로 설정합니다.
        self.path_to_input = path_to_input
        self.path_to_s1 = path_to_s1
        self.len_of_samples = len_of_samples
        self.fs = fs
        self.train_flag = train_flag

        # 데이터셋의 샘플 수를 계산합니다.
        self.count_samples()

        # 반복 가능한 tf.data.Dataset 객체를 생성합니다.
        self.create_tf_data_obj()

    def count_samples(self):
        '''
        데이터셋의 데이터를 나열하고 샘플 수를 계산하는 메서드입니다.
        '''
        # 디렉터리에서 .wav 파일을 나열합니다.
        self.file_names = fnmatch.filter(os.listdir(self.path_to_input), '*.wav')

        # 데이터셋에 포함된 총 샘플 수를 초기화합니다.
        self.total_samples = 0

        # 각 파일에서 샘플 수를 계산하고 총 샘플 수를 누적합니다.
        for file in self.file_names:
            info = WavInfoReader(os.path.join(self.path_to_input, file))
            self.total_samples += int(np.fix(info.data.frame_count / self.len_of_samples))


    def create_generator(self):
        '''
        반복자를 생성하는 메서드입니다.
        '''
        # 훈련 또는 검증 여부를 확인하고, 파일을 섞는지 여부를 결정합니다.
        if self.train_flag:
            shuffle(self.file_names)

        # 파일을 반복합니다.
        for file in self.file_names:
            # 혼합 오디오 파일과 대상 소스 데이터를 읽어옵니다.
            noisy, fs_1 = sf.read(os.path.join(self.path_to_input, file))
            speech, fs_2 = sf.read(os.path.join(self.path_to_s1, file))

            # 샘플링 속도가 지정 사항과 일치하는지 확인합니다.
            if fs_1 != self.fs or fs_2 != self.fs:
                raise ValueError('샘플링 속도가 일치하지 않습니다.')

            # 단일 채널 오디오 데이터만 지원합니다.
            if noisy.ndim != 1 or speech.ndim != 1:
                raise ValueError('오디오 채널이 너무 많습니다. DTLN audio_generator는 단일 채널 오디오
```

```
            # 파일 내의 샘플 수를 계산합니다.
            num_samples = int(np.fix(noisy.shape[0] / self.len_of_samples))

            # 파일의 각 샘플에 대해 오디오 조각을 생성하고 반환합니다.
            for idx in range(num_samples):
                # 오디오 파일을 조각으로 나눕니다.
                in_dat = noisy[int(idx * self.len_of_samples):int((idx + 1) * self.len_of_sam
                tar_dat = speech[int(idx * self.len_of_samples):int((idx + 1) * self.len_of_s

                # 조각을 float32 데이터로 반환합니다.
                yield in_dat.astype('float32'), tar_dat.astype('float32')


    def create_tf_data_obj(self):
        '''
        tf.data.Dataset를 생성하는 메서드입니다.
        '''
        # 반복자에서 tf.data.Dataset를 생성합니다.
        self.tf_data_set = tf.data.Dataset.from_generator(
                        self.create_generator,
                        (tf.float32, tf.float32),
                        output_shapes=(tf.TensorShape([self.len_of_samples]), \
                                       tf.TensorShape([self.len_of_samples])),
                        args=None
                        )
```

기존 코드는 다채널 오디오를 지원하지 않고 있어, 다채널 오디오를 지원하게끔 코드 수정 필요

▼ 2채널 지원하는 코드로 수정 (테스트 필요)

```
class audio_generator():
    '''
    대규모 오디오 데이터셋으로부터 이터레이터를 기반으로 Tensorflow 데이터셋을 생성하는 클래스입니다.
    이 오디오 생성기는 단일 채널 오디오 파일만 지원합니다.
    '''

    def __init__(self, path_to_input, path_to_s1, len_of_samples, fs, train_flag=False):
        '''
        오디오 생성기 클래스의 생성자입니다.
        Inputs:
            path_to_input       혼합 오디오 파일의 경로
            path_to_s1          대상 소스 데이터의 경로
            len_of_samples      샘플 단위의 오디오 스니펫 길이
            fs                  샘플링 레이트
            train_flag          파일 셔플링을 활성화하기 위한 플래그
        '''
        # 입력을 속성에 설정합니다.
        self.path_to_input = path_to_input
        self.path_to_s1 = path_to_s1
        self.len_of_samples = len_of_samples
        self.fs = fs
        self.train_flag = train_flag
        # 데이터셋 내의 샘플 수를 계산합니다. (디스크에 따라 시간이 오래 걸릴 수 있습니다)
        self.count_samples()
        # 이터러블한 tf.data.Dataset 객체를 생성합니다.
        self.create_tf_data_obj()
```

```
    def count_samples(self):
        '''
        데이터셋의 데이터를 나열하고 샘플 수를 세는 메서드입니다.
        '''

        # 디렉터리 내의 .wav 파일을 나열합니다.
        self.file_names = fnmatch.filter(os.listdir(self.path_to_input), '*.wav')
        # 데이터셋에 포함된 샘플 수를 계산합니다.
        self.total_samples = 0
        for file in self.file_names:
            info = WavInfoReader(os.path.join(self.path_to_input, file))
            self.total_samples = self.total_samples + \
                int(np.fix(info.data.frame_count / self.len_of_samples))


    def create_generator(self):
        '''
        이터레이터를 생성하는 메서드입니다.
        '''

        # 훈련 또는 검증 여부를 확인합니다.
        if self.train_flag:
            shuffle(self.file_names)
        # 파일을 반복합니다.
        for file in self.file_names:
            # 오디오 파일을 읽어옵니다.
            # 읽어들인 오디오 데이터를 2개의 오디오 입력으로 처리합니다.
            noisy, fs_1 = sf.read(os.path.join(self.path_to_input, file), channels=2)
            speech, fs_2 = sf.read(os.path.join(self.path_to_s1, file), channels=2)
            # 샘플링 레이트가 명시된 사양과 일치하는지 확인합니다.
            if fs_1 != self.fs or fs_2 != self.fs:
                raise ValueError('샘플링 레이트가 일치하지 않습니다.')
            # 오디오 채널 수가 맞지 않으면 에러를 발생시킵니다.
            if noisy.ndim != 2 or speech.ndim != 2:
                raise ValueError('오디오 채널이 일치하지 않습니다. 오디오 생성기는 \
                                  두 개의 채널 오디오 데이터만 지원합니다.')
            # 파일의 샘플 수를 계산합니다.
            num_samples = int(np.fix(noisy.shape[0] / self.len_of_samples))
            # 샘플 수만큼 반복합니다.
            for idx in range(num_samples):
                # 오디오 파일을 잘라서 처리합니다.
                in_dat = noisy[int(idx * self.len_of_samples):int((idx + 1) *
                                                                  self.len_of_samples)]
                tar_dat = speech[int(idx * self.len_of_samples):int((idx + 1) *
                                                                    self.len_of_samples)]
                # float32 데이터로 반환합니다.
                yield in_dat.astype('float32'), tar_dat.astype('float32')


    def create_tf_data_obj(self):
        '''
        tf.data.Dataset을 생성하는 메서드입니다.
        '''

        # 이터레이터에서 tf.data.Dataset을 생성합니다.
        self.tf_data_set = tf.data.Dataset.from_generator(
                        self.create_generator,
                        # 두 개의 오디오 입력에 맞게 출력 형태를 수정합니다.
```

```
                        (tf.float32, tf.float32),
                        output_shapes=(tf.TensorShape([self.len_of_samples, 2]), \
                                       tf.TensorShape([self.len_of_samples, 2])),
                        args=None
                        )
```

수정내용:

1. `create_generator` 메서드에서 `sf.read` 함수의 `channels` 매개변수를 2로 설정하여 2개의 오디오 채널을 읽어들이도록 수정했습니다.
2. 데이터를 처리하는 부분에서 두 개의 오디오 채널을 고려하여 코드를 수정했습니다. 이를 위해 다차원 배열의 차원을 확인하는 `ndim` 메서드를 사용하고, 데이터의 형태를 고려하여 새로운 차원을 추가했습니다.
3. `output_shapes` 매개변수를 수정하여 각 오디오 채널에 대한 형태를 명시했습니다. 이를 위해 각 채널에 대한 차원을 추가했습니다.
