60. 6주차 실습 보고서

202102675 이문영

rules.py

다음 코드는 이번 실습에서 사용될 기준 주파수 설정입니다. 저는 주파수 간격을 192로 설정하여 좀 더 명확하게 주파수를 구분할 수 있게 하였습니다.

```
FREQ_START = 512          # START 기준 주파수
FREQ_STEP = 192           # 주파수 간격

HEX_LIST = ['0', '1', '2', '3', '4',
            '5', '6', '7', '8', '9',
            'A', 'B', 'C', 'D', 'E', 'F']

rules = {}

# START 신호 주파수 설정
rules['START'] = FREQ_START

# 0 ~ F 에 주파수 할당
for i, h in enumerate(HEX_LIST):
```

```
    # START보다 1 STEP 띄우고, 이후 1씩 증가 (i+1)
    rules[h] = FREQ_START + FREQ_STEP + FREQ_STEP *
(i + 1)

# END 신호 주파수는 마지막보다 2칸 더 띄우기
rules['END'] = FREQ_START + FREQ_STEP + FREQ_STEP *
(len(HEX_LIST) + 2)
```

send.py

```
import math, struct, pyaudio
from rules import rules

INTMAX = 2**15 - 1
UNIT = 0.1
FS = 48000



def send_data():
    text = input("Enter text to send: ").strip()
    #입력 받은 텍스트를 먼저 utf-8인코딩하여 16진수로
나타내줍니다.
    hex_string = text.encode('utf-8').hex().upper()

    print(f"[HEX]: {hex_string}")
    print("length:", len(hex_string))

    audio = [] #나중에 반환할 audio 데이터 리스트입
니다.
```

```
    # 인자로 주파수와 유닛을 받고 이를 sin wave = 소
리데이터로 변환 후
    # 이를 audio배열에 추가해주는 함수입니다
    def add_wave(freq, units=1):
        samples = [
            int(INTMAX * math.sin(2 * math.pi *
freq * i / FS))
            for i in range(int(FS * UNIT * units))
        ]
        audio.extend(samples)

    # 먼저 통신 소리의 시작으로 START주파수로 2unit
만큼의 sin wave를 생성하여
    # audio 리스트에 추가해줍니다.
    add_wave(rules['START'], 2)

    # 시작 통신음 이후 전송해야할 내용을 차례대로
add_wave함수로 16진수에 맵핑되는
    # 주파수 값을 기반으로 audio데이터를 추가해줍니
다.
    for h in hex_string:
        add_wave(rules[h], 1)
    # 모든 데이터를 추가한 후 통신의 끝을 알리기 위해
END주파수로 2unit만큼
    # 오디오 데이터를 추가합니다.
    add_wave(rules['END'], 2)

    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16,
channels=1, rate=FS, output=True)
    stream.write(struct.pack('<' + 'h' *
len(audio), *audio))
```

```
    stream.stop_stream(); stream.close();
p.terminate()

    print("Transmission finished.")
```

receive.py

```
import pyaudio, struct, numpy as np, scipy.fftpack
from rules import rules

INTMAX = 2**15 - 1
UNIT = 0.1
FS = 48000
CHUNK = int(FS * UNIT)
PADDING = 10  # 주변 환경에 따라 적절히 조절하였습니다.
DATA_THRESHOLD = 100  # 주변 환경에 따라 적절히 조절하였습니다.


def receive_data():
    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16,
channels=1, rate=FS, input=True)
    print("Listening...")

    hex_data = ''
    started = False
    end_count = 0
```

```
    while True:
        raw = stream.read(CHUNK)
        chunk = struct.unpack('<' + 'h'*CHUNK, raw)

        fft = np.fft.fft(chunk)
        freqs = np.fft.fftfreq(len(chunk), d=1/FS)
        freq = abs(freqs[np.argmax(abs(fft))])
        volume = np.std(chunk)

        if volume < DATA_THRESHOLD:
            print(f"[NOISE] Volume too low
({volume:.2f}) → Ignored")
            continue


        matched = None
        for k, v in rules.items():
            if abs(freq - v) <= PADDING:
                matched = k
                break

        if matched == 'START':
            print(f"[START] {matched} with
{freq:.1f} Hz")
            started = True
            hex_data = ''

        elif matched == 'END' and started:
            print(f"[END] END with {freq:.1f} Hz")
            end_count += 1
            if end_count >= 2:
                break
```

```
        elif matched and matched not in ['START',
'END'] and started:
            print(f"[DATA] {matched} with
{freq:.1f} Hz | Current data: {hex_data}")
            hex_data += matched
        else:
            print(f"[FREQ={freq:.1f}] Volume:
{volume:.2f}")


    stream.stop_stream(); stream.close();
p.terminate()
    print("raw data :", hex_data)
    print("length:", len(hex_data))


    try:
        decoded =
bytes.fromhex(hex_data).decode('utf-8')
    except:
        decoded = '[DECODE ERROR]'
    print("Decoded Text:", decoded)
```

다음은 receive_data()함수의 초반 설정입니다. 약속된 설정값을 기반으로 pyaudio 패키지의 함수를 사용하여 현재 마이크 스트림으로부터 입력을 while문으로 계속해서 어떤 입력이 들어오고 있는지 확인합니다.

```
def receive_data():
    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16,
```

```
channels=1, rate=FS, input=True)
    print("Listening...")

    hex_data = ''
    started = False
    end_count = 0

    while True:
        raw = stream.read(CHUNK)
        chunk = struct.unpack('<' + 'h'*CHUNK, raw)

        fft = np.fft.fft(chunk)
        freqs = np.fft.fftfreq(len(chunk), d=1/FS)
        freq = abs(freqs[np.argmax(abs(fft))])
        volume = np.std(chunk)
```

`CHUNK = 48000 × 0.1 = 4800` 으로 설정되어 있으므로, 매 반복마다 **0.1초 분량의 오디오 데이터를 stream으로부터 읽어와** `raw` 에 저장합니다.

이 `raw` 는 바이트 형태의 PCM(16비트 정수) 오디오 데이터이며, `struct.unpack()` 을 통해 이를 **정수 샘플 리스트인** `chunk` **로 변환**합니다.
이후 `np.fft.fft()` 를 사용해 해당 구간의 오디오 샘플에 대해 **고속 푸리에 변환(FFT)** 을 수행하고,
`np.argmax(abs(...))` 를 통해 **가장 강한 진폭(에너지)을 가진 주파수 성분을 찾아**, 이를 **dominant frequency(주성분 주파수)** 로 간주합니다.
`volume = np.std(chunk)` 는 해당 chunk의 **볼륨(진폭의 변화**

량) 을 추정하기 위한 값으로, **소음 필터링의 기준**으로 사용됩니다.

목표1

다음은 **오디오 볼륨이 일정 기준 이하일 경우, 해당 구간을 잡음(Noise)으로 간주하고 처리하지 않는 코드**입니다.

```
        if volume < DATA_THRESHOLD:
            print(f"[NOISE] Volume too low ({volume:.2f}) → Ignored")
            continue
```

`volume`은 현재 수신된 오디오 chunk의 표준편차(`np.std(chunk)`)로 계산되며,
이 값이 설정된 `DATA_THRESHOLD`보다 작을 경우, **해당 단위 구간(0.1초)** 은 의미 있는 신호가 없는 것으로 판단하여 `continue`를 통해 건너뜁니다.
이 처리는 **불필요한 잡음 감지로 인한 오작동을 방지** 하는 데 목적이 있습니다.

목표 2,3

다음 코드는 현재 수신된 주파수(`freq`)를 기준으로, 이 값이 **어떤 데이터 심볼(16진수 또는 START/END 신호)** 에 해당하는지를 판단하는 과정입니다.

```
matched = None
        for k, v in rules.items():
            if v - PADDING < freq < v + PADDING:
                matched = k
                break
```

`rules.items()`를 순회하면서, 각 심볼에 매핑된 기준 주파수(`v`)와 현재 주파수(`freq`)의 차이가 **허용 오차(PADDING)** 이내인지를 비교합니다. (저는 PADDING을 10으로 주었습니다.)

만약 `freq`가 특정 `v`와 오차 범위 내에 있다면, 해당 심볼(`k`)을 `matched` 변수에 저장하고 반복을 종료합니다.
이때 `matched`에는 `0`부터 `F`, `START`, `END` 중 하나가 저장됩니다.

다음 코드는 현재 수신된 주파수(`freq`)가 어떤 신호(`matched`)에 해당하는지를 기준으로, 해당 입력을 어떻게 처리할지를 결정하는 로직입니다.

```
if matched == 'START':
            print(f"[START] {matched} with
{freq:.1f} Hz")
            started = True
            hex_data = ''
```

```
        elif matched == 'END' and started:
            print(f"[END] END with {freq:.1f} Hz")
            end_count += 1
            if end_count >= 2:
                break
        elif matched and matched not in ['START',
'END'] and started:
            print(f"[DATA] {matched} with
{freq:.1f} Hz | Current data: {hex_data}")
            hex_data += matched
        else:
            print(f"[FREQ={freq:.1f}] Volume:
{volume:.2f}")
```

- `matched == 'START'` 인 경우
  - 데이터 수신의 시작을 의미합니다.
  - 초기 상태였던 `started = False`를 `True`로 변경하여, 이 시점부터 실제 데이터를 받기 시작하도록 설정합니다.
  - 동시에 `hex_data`를 초기화하여 새로운 수신 데이터를 저장할 준비를 합니다.
  - 다음 이미지와 같이 START signal을 2unit 감지합니다.

```
[FREQ=120.0] Volume: 129.66
[NOISE] Volume too low (49.07) → Ignored
[NOISE] Volume too low (26.31) → Ignored
[START] START with 510.0 Hz
[START] START with 510.0 Hz
[DATA] 6 with 2050.0 Hz | Current data: 6
```

- `matched`가 `'START'`, `'END'`가 아니고, `started == True`인 경우

- 현재 수신 중이고, 받은 주파수가 실제 데이터(`0`~`F`)에 해당하는 것으로 판단된 상황입니다.
- 이 경우, 해당 `matched` 값을 `hex_data` 문자열에 추가하여 수신된 전체 데이터에 누적합니다.
- 또한 현재 상태를 실시간으로 출력하여 데이터를 잘 수신하고 있는지 확인할 수 있습니다.
- 다음 이미지와 같이 이미 START signal이 있었고 matched된 데이터가 있을 경우 Current data에 추가합니다.

```
[START] START with 510.0 Hz
[DATA] 6 with 2050.0 Hz | Current data: 6
[DATA] 8 with 2430.0 Hz | Current data: 68
[DATA] 6 with 2050.0 Hz | Current data: 686
[DATA] 9 with 2620.0 Hz | Current data: 6869
[END] END with 4160.0 Hz
```

- **`matched == 'END'` 이고, 수신 중인 경우 (`started == True`)**
  - 이는 데이터 전송의 종료 신호입니다.
  - 실습 규칙에 따라 `END` 신호는 2 unit 동안 전송되므로, 이 신호가 2회 감지되었을 때 수신을 종료하도록 설계합니다.
  - 감지된 횟수는 `end_count`로 누적 관리되며, `end_count >= 2`가 되면 `while` 루프를 `break` 하여 수신을 종료합니다.
  - 다음 이미지에서 볼 수 있듯이 2unit의 END signal 이후 종료합니다.

```
[DATA] 9 with 2620.0 Hz | Current data: 6869
[END] END with 4160.0 Hz
[END] END with 4160.0 Hz
```

- `else`
  - 매칭된 데이터가 없거나, 아직 `START` 신호를 받지 않은

상태에서 감지된 주파수는 의미 없는 신호(노이즈 또는 잡음)로 간주하며,
- 해당 주파수와 볼륨만 로그로 출력합니다.

```
[NOISE] Volume too low (58.53) → Ignored
[NOISE] Volume too low (42.26) → Ignored
[NOISE] Volume too low (22.82) → Ignored
[FREQ=120.0] Volume: 129.66
[NOISE] Volume too low (49.07) → Ignored
[NOISE] Volume too low (26.31) → Ignored
[START] START with 510.0 Hz
[START] START with 510.0 Hz
```

이렇게 통신의 끝을 감지하고 `while`문을 `break`한 후 다음 코드를 진행합니다.

```
 stream.stop_stream(); stream.close();
p.terminate()
    print("raw data :", hex_data)
    print("length:", len(hex_data))

    try:
        decoded =
bytes.fromhex(hex_data).decode('utf-8')
    except:
        decoded = '[DECODE ERROR]'
    print("Decoded Text:", decoded)
```

현재 열어두었던 마이크 `stream`을 닫고 현재까지 수신한 16진수 데이터와 그 길이를 출력하고 이를 decode한 결과를 출력합

니다.

```
[END] END with 4160.0 Hz
[END] END with 4160.0 Hz
raw data : 6869
length: 4
Decoded Text: hi
```