

40. 4주차 실습 보고서

202102675 이문영

설정 값들

MoresCodeOver.py

```
INTMAX = 2**15 - 1 # 32767
UNIT = 0.1          # 1 unit = 100ms
FREQ = 523.251      # C5 tone
FS = 48000          # 샘플레이트
MORSE_THRESHOLD = 700 # 임계값 (환경에 따라 조정 필요)
CHUNK_SIZE = 1024
FORMAT = pyaudio.paInt16
```

pyaudio.paInt16을 사용하였으며, MORSE_THRESHOLD 값은 스피커의 음성 크기와 마이크에서 인식하는 값을 관찰 후 적절한 임계값으로 설정하였습니다. 이 임계값 이상일 경우 신호로 인식하도록 로직을 구성하였습니다.

send_data()

```
# Morse Code 테이블
MORSE_CODE_DICT = morse_data.MORSE_DICT

def send_data():
    text = input("Enter text to send (A-Z, 0-9 only): ").strip()
    morse = textToMorse.text2morse(text)
    print("Morse:", morse)

    p = pyaudio.PyAudio()
    stream = p.open(format=FORMAT,
```

```

        channels=1,
        rate=FS,
        output=True)

def play_tone(units):
    num_samples = int(FS * UNIT * units)
    samples = [
        int(INTMAX * math.sin(2 * math.pi * FREQ * (i /
FS)))
        for i in range(num_samples)
    ]
    packed = struct.pack('<' + 'h' * len(samples), *samples)
    stream.write(packed)

def play_silence(units):
    num_samples = int(FS * UNIT * units)
    silence = [0] * num_samples
    packed = struct.pack('<' + 'h' * len(silence), *silence)
    stream.write(packed)

# Morse 재생
for symbol in morse:
    if symbol == '.':
        play_tone(1)          # 1 unit
        play_silence(1)       # 기호 사이
    elif symbol == '-':
        play_tone(3)          # 3 units
        play_silence(1)       # 기호 사이
    elif symbol == ' ':
        play_silence(3)       # 문자 사이
    elif symbol == '/':
        play_silence(7)       # 단어 사이

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

```

send_data() 핵심 로직

```
# Morse 재생
for symbol in morse:
    if symbol == '.':
        play_tone(1)      # 1 unit
        play_silence(1)   # 기호 사이
    elif symbol == '-':
        play_tone(3)      # 3 units
        play_silence(1)   # 기호 사이
    elif symbol == ' ':
        play_silence(3)   # 문자 사이
    elif symbol == '/':
        play_silence(7)   # 단어 사이
```

위와 같이 입력 받은 모스 부호를 표준 해석에 맞게 소리를 냅니다. dot과 dash는 각각 1번의 tone과 3번의 tone으로 이루어져 있으며 각 dot과 dash 사이는 1 unit silence를 호출합니다.

공백일 경우 문자 사이로 인지하여, 표준에 맞게 3 unit silence를, '/'일 경우 7 unit silence를 호출하게 합니다.

play_tone()

```
def play_tone(units):
    num_samples = int(FS * UNIT * units)
    samples = [
        int(INTMAX * math.sin(2 * math.pi * FREQ * (i /
FS)))
        for i in range(num_samples)
    ]
    packed = struct.pack('<' + 'h' * len(samples), *samples)
    stream.write(packed)
```

play_tone 함수는 입력받은 units의 개수 만큼 소리를 내줍니다.

`num_samples = int(FS * UNIT * units)` 에서 총 샘플의 개수를 저장합니다. $FS = 48000 * 0.1 * units$ 으로 계산 되며, 만약 `units = 1` 을 인자로 받았다면 샘플 수를 4800이라 저장합니다.

```

samples = [
    int(INTMAX * math.sin(2 * math.pi * FREQ * (i / FS)))
    for i in range(num_samples)
]

```

실제 소리 데이터를 num_samples(샘플 개수)만큼 생성하는 코드입니다. 파이썬의 리스트 컴프리헨션 문법을 사용하여, 각 i에 대한 `int(INTMAX * math.sin(2 * math.pi * FREQ * (i / FS)))` 계산 결과를 samples 리스트에 저장합니다.

```

packed = struct.pack('<' + 'h' * len(samples), *samples)
stream.write(packed)

```

저장된 샘플 데이터를 16bit PCM 오디오 형식으로 이진 데이터를 생성하고 이를 출력 스트림에 write하여 사인파 소리를 출력합니다.

play_silence()

```

def play_silence(units):
    num_samples = int(FS * UNIT * units)
    silence = [0] * num_samples
    packed = struct.pack('<' + 'h' * len(silence), *silence)
    stream.write(packed)

```

`silence = [0] * num_samples` 모든 값이 0인 리스트를 생성하여 무음을 출력하게 합니다.

recv_data()

```

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

```

```

audio = []
started = False
unseen_count = 0

while True:
    raw = stream.read(unit_samples)
    chunk = struct.unpack('<' + 'h' * unit_samples, raw)
    std = statistics.stdev(chunk)

    if not started:
        if std > MORSE_THRESHOLD:
            print("Signal detected! Starting recording...")
            started = True
            audio.extend(chunk)
        else:
            print("Waiting... std =", std)
    else:
        print(f"Recording... std = {std:.3f}")
        audio.extend(chunk)
        if std < MORSE_THRESHOLD:
            unseen_count += 1
        else:
            unseen_count = 0

        if unseen_count >= 30: # 3초간 무음이면 종료
            print("Silence too long. Stopping recording.")
            break

    stream.stop_stream()
    stream.close()
    p.terminate()
    print(f"Recording finished. Total units recorded:
{len(audio)//unit_samples}")

# 분석 및 해독
morse_str =
decode_morse_from_audio.decode_morse_from_audio(audio, FS, UNIT,
MORSE_THRESHOLD)

print("Detected Morse:", morse_str)

```

```
decoded_text = morseToText.morse_to_text(morse_str)
print("Decoded Text:", decoded_text)
```

핵심 로직 (과제 목표 1, 2, 3)

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

수신부는 신호 감지 이전부터 지속적으로 입력 스트림을 모니터링하며, 신호가 감지되면 녹음을 시작하고, 일정 시간 이상 신호가 감지되지 않으면 녹음을 종료하는 방식으로 구성되어 있습니다.

먼저, PyAudio의 입력 스트림으로부터 `unit_samples = int(FS * UNIT)` 만큼의 샘플(0.1초에 해당)을 읽어옵니다.

이 샘플들은 `struct.unpack()` 을 통해 정수 리스트 형태로 변환되며, 해당 구간의 **진폭의 표준편차(standard deviation)** 를 `statistics.stdev()` 로 계산하여 `std` 에 저장합니다.

이 `std` 값을 통해 해당 구간에 실제 소리가 있었는지(신호인지 무음인지) 를 판단합니다.

아래 코드는 위 반복문 내의 코드입니다.

```
if not started:
    if std > MORSE_THRESHOLD:
        print("Signal detected! Starting recording...")
        started = True
        audio.extend(chunk)
    else:
        print("Waiting... std =", std)
else:
    print(f"Recording... std = {std:.3f}")
    audio.extend(chunk)
    if std < MORSE_THRESHOLD:
        unseen_count += 1
```

```

else:
    unseen_count = 0

    if unseen_count >= 30: # 3초간 무음이면 종료
        print("Silence too long. Stopping recording.")
        break

```

이 코드는 `std` 값을 기반으로 녹음을 시작할 시점과 종료할 시점을 판단하는 핵심 로직입니다.

1. 초기에는 `started = False` 상태이며,
이 때 `std > MORSE_THRESHOLD` 조건을 만족하면 소리(신호)가 감지되었다고 판단하고, `started = True` 로 바꾸며 녹음을 시작합니다.
동시에 해당 오디오 샘플(`chunk`)을 `audio` 리스트에 저장합니다.
2. 반대로, 아직 신호가 감지되지 않은 상태(`started = False`)이고 `std < MORSE_THRESHOLD` 일 경우에는
"Waiting..." 메시지를 출력하며 계속해서 다음 단위를 기다립니다.
3. `started = True` 인 경우에는 실제 녹음 중인 상태로, 매 유닛 단위로 계속 `chunk` 를 `audio` 에 저장하고,
표준편차 `std` 가 다시 일정 기준 미만으로 떨어지면 (즉, 무음 상태가 감지되면) `unseen_count` 를 1 증가시킵니다.
4. 반대로 `std > MORSE_THRESHOLD` 인 경우(소리가 다시 들어온 경우)는 `unseen_count` 를 0으로 초기화합니다.
5. 마지막으로, 무음 상태(`std < threshold`)가 `30 unit` 이상 지속되면 (즉, 3초 이상 무음)
`break` 를 통해 녹음을 종료합니다.
(`unseen count`는 매 유닛 = 0.1s마다 ++되니 `30unit = 3s`)

녹음 된 aaudio 해독

`recv_data()`의 녹음 로직이 끝난 후 녹음된 오디오 리스트를 모스코드로 변환하고 이를 텍스트로 해석하는 과정을 거칩니다.

```

# 분석 및 해독
morse_str =

```

```

decode_morse_from_audio.decode_morse_from_audio(audio, FS, UNIT,
MORSE_THRESHOLD)
    print("Detected Morse:", morse_str)

    decoded_text = morseToText.morse_to_text(morse_str)
    print("Decoded Text:", decoded_text)

```

decode_morse_from_audio

```

def decode_morse_from_audio(audio, FS, UNIT, MORSE_THRESHOLD):
    units_per_chunk = int(FS * UNIT) # 4800 samples
    morse_bits = []

    # 0.1초 단위로 나눠서 표준편차 측정
    for i in range(0, len(audio), units_per_chunk):
        chunk = audio[i:i + units_per_chunk]
        if len(chunk) < units_per_chunk:
            break
        stdev = statistics.stdev(chunk)
        if stdev > MORSE_THRESHOLD:
            morse_bits.append(1)
        else:
            morse_bits.append(0)

    return bits_to_morse(morse_bits)

```

이 함수는 입력으로 해석할 오디오 샘플 리스트 `audio`, 샘플레이트(`FS = 48000`), 모스 부호의 단위 시간(`UNIT = 0.1초`), 소리 감지를 위한 임계값(`MORSE_THRESHOLD`)을 인자로 받습니다.

함수 내부에서는 오디오 데이터를 `UNIT` 단위(0.1초, 즉 4800 샘플씩)로 잘라가며, 각 구간의 표준편차(`stdev`)를 계산하여 해당 구간에 소리가 있었는지(1) 혹은 무음인지(0)를 판단합니다.

이렇게 생성된 1 과 0 의 비트 시퀀스는 `morse_bits` 리스트에 저장되며, 최종적으로 이 리스트는 `bits_to_morse(morse_bits)` 함수에 전달되어 모스 부호 문자열로 변환됩니다.

bits_to_morse

```
def bits_to_morse(bits):
    morse = ""
    count = 0
    current = bits[0]

    for bit in bits:
        if bit == current:
            count += 1
        else:
            morse += interpret_sequence(current, count)
            current = bit
            count = 1
    morse += interpret_sequence(current, count)
    return morse
```

이 함수는 1과 0으로 이루어진 비트 시퀀스에서 **같은 값이 연속된 구간의 길이** (`count`)를 측정하고, 그 값과 길이를 기반으로 `interpret_sequence()` 함수를 호출하여 모스 부호 문자(`.`, `-`, `' '`, `' / '`)로 해석하는 역할을 수행합니다.

연속된 비트 값이 바뀌는 지점(즉, `1 → 0` 또는 `0 → 1`)이 생기면 그 구간의 길이를 해석한 후 `morse` 문자열에 결과를 누적합니다. 루프가 끝난 후 마지막 비트 열에 대한 해석도 누락되지 않도록 한 번 더 `interpret_sequence()` 를 호출합니다.

interpret_sequence

```
def interpret_sequence(bit, count):
    if bit == 1: # tone
        if count <= 2:
            return '.' # dit
```

```

elif count <= 5:
    return '-' # dah
elif bit == 0: # silence
    if count <= 2:
        return ' ' # 기호 사이
    elif count <= 5:
        return ' ' # 문자 사이
    elif count >= 6:
        return ' / ' # 단어 사이
return ''

```

이 함수는 `bit` 값이 1 (소리)인지 0 (무음)인지와 그 값이 얼마나 연속되었는지 (`count`)를 인자로 받아, 모스 부호 규칙에 따라 해당 구간을 `.`, `-`, 공백 등으로 변환합니다.

원칙적으로는 1 unit의 소리는 `.` (dit), 3 unit의 소리는 `-` (dah)로 정해져 있지만,

실제 녹음 환경에서는 신호가 정확히 끊기지 않거나 샘플 수에 미세한 오차가 생길 수 있습니다.

따라서 실제 감지된 비트 수에 일정 범위의 오차를 허용하도록 다음과 같은 기준을 적용했습니다

- 소리 (`bit == 1`)
 - 1~2 unit → `.`
 - 3~5 unit → `-`
- 무음 (`bit == 0`)
 - 1~2 unit → 기호 간 무음 (해석하지 않음)
 - 3~5 unit → 문자 간 공백 `' '`
 - 6 unit 이상 → 단어 간 공백 `' / '`