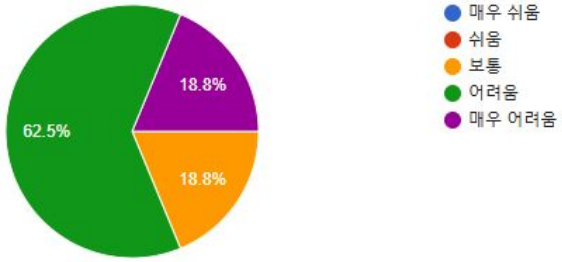
데이터통신

- 실습 6주차. Send unicode over sound with music -

충남대학교 컴퓨터융합학부
데이터네트워크 연구실

4주차 실습에 대한, **주관적인 난이도**가 어땠는지?

응답 16개



과제 난이도를 낮춰주셨으면 좋겠고, 과제 제출 기한도 다시 늘려주셨으면 좋겠습니다..

마이크 대여 가능을 아예 실습시간부터 안내해줬으면 좋겠습니다. 실습 시간 중에 코딩을 완료한 학생은 마이크를 대여한 뒤 옆의 빈 강의실 등에서 소리를 녹음하고 올 수 있도록 하면 좋겠습니다.

실습에 대한 이론이 조금 더 설명이 있었으면 좋겠다.

과제 난이도를 낮춰줬으면 좋겠습니다.

과제 난이도가 조금 낮았으면 좋겠습니다.

실습에 적용되는 이론을 더 자세히 설명해주셨으면 좋겠음. 과제 자료를 더 빨리 업로드해주셨으면 좋겠음

이론수업과 과제실습의 내용 연결이 긴밀하지 않은 것 같다. 이론수업에 대한 자료가 더 풍부했으면 좋겠다.

소리 인식이 환경 마다 결과 값이 다르게 나와요ㅠㅠ

코딩 자체는 쉬웠으나, 소리를 인식하는 과정에서 마이크에 따른 성능 편차가 너무 심해 노트북 기본마이크 등으로는 진행을 하지 못함

소리 인식이 잘 안됨. (특히나 동기화 부분에 있어서 중간에 소리가 씹히는 현상이 발생)

파이썬 자체가 익숙하지 않고 마이크 인식 되어도 내가 원하는 출력값이 잘 안나온다. 그리고 과제가 매주 이어지다 보니 앞에 주차 과제를 제대로 해놓지 않으면 뒤에 주차도 연달아서 하지 못하게 된다.

소리 인식이 잘 안되는 문제를 해결하지 못했습니다.

소리인식이 잘 안됨 코드 하나를 가지고도 테스트 결과의 차이가 심함

인식되는 소리를 모스부호로 바꾸는데, 입력시간에 따라서 ... 이나 . . . 이나 - 등 변환하는 방법을 잘 이해하지 못했음

본 학부에서는 대부분 자바를 기준으로 강의가 진행되므로 파이썬에 익숙하지 않음

이론시간에 실습 내용을 자세하게 다뤄줬으면 좋겠습니다. 이론과 실습이 연결이 되지 않는 것 같습니다.

다른 과제 제출 날짜가 금/월에 많아 이번처럼 어려운 과제의 경우 과제 기간을 일주일로 늘려주시면 딱 좋을 것 같습니다.. 3주차정도까지의 난이도면 이대로도 괜찮습니다!

실습 데모 영상에서 실험 환경이 어떻게 설정되어있는지 알려주면 좋을 것 같다.

실습에 대한 이론을 더 자세히 설명해줬으면 좋겠어요. 제출 기한을 하루만 늘려주세요..

실습에 대한 코드에 대해 설명해줬으면 좋겠고 과제 난이도를 낮춰줬으면 좋겠습니다.□

과제 관련해서 어떻게 진행을 해야할지에 대한 힌트를 진행을 해준다면 좋을 거 같습니다.
현재까지 과제중에 이런게 있다 이렇게 소개만하고 과제를 진행을 하여 그냥 아무것도 모르는 상태에서 맨땅의 헤딩하면서 과제를 진행하여 시간을 굉장히 많이 소비를 하게 되었습니다.

기존 실습의 코드는 어떻게 진행이 되는지 설명을 한 뒤에 과제에서는 이 코드를 어떻게 활요을 하면 문제를 해결 할 수 있다 이렇게 진행을 하면 좋을거 같습니다.

Feedback

- 제출률: 97.82% (40+5/46)
- 설문조사 [클릭] / 희망자 진행
- 개선 내용

1) 과제 제출 기한 하루 연장 (화-> 수)

   - 7주차 과제는 시험끝난뒤로 마감기한 연장

2) 실습에 대한 코드 / 이론 설명 추가

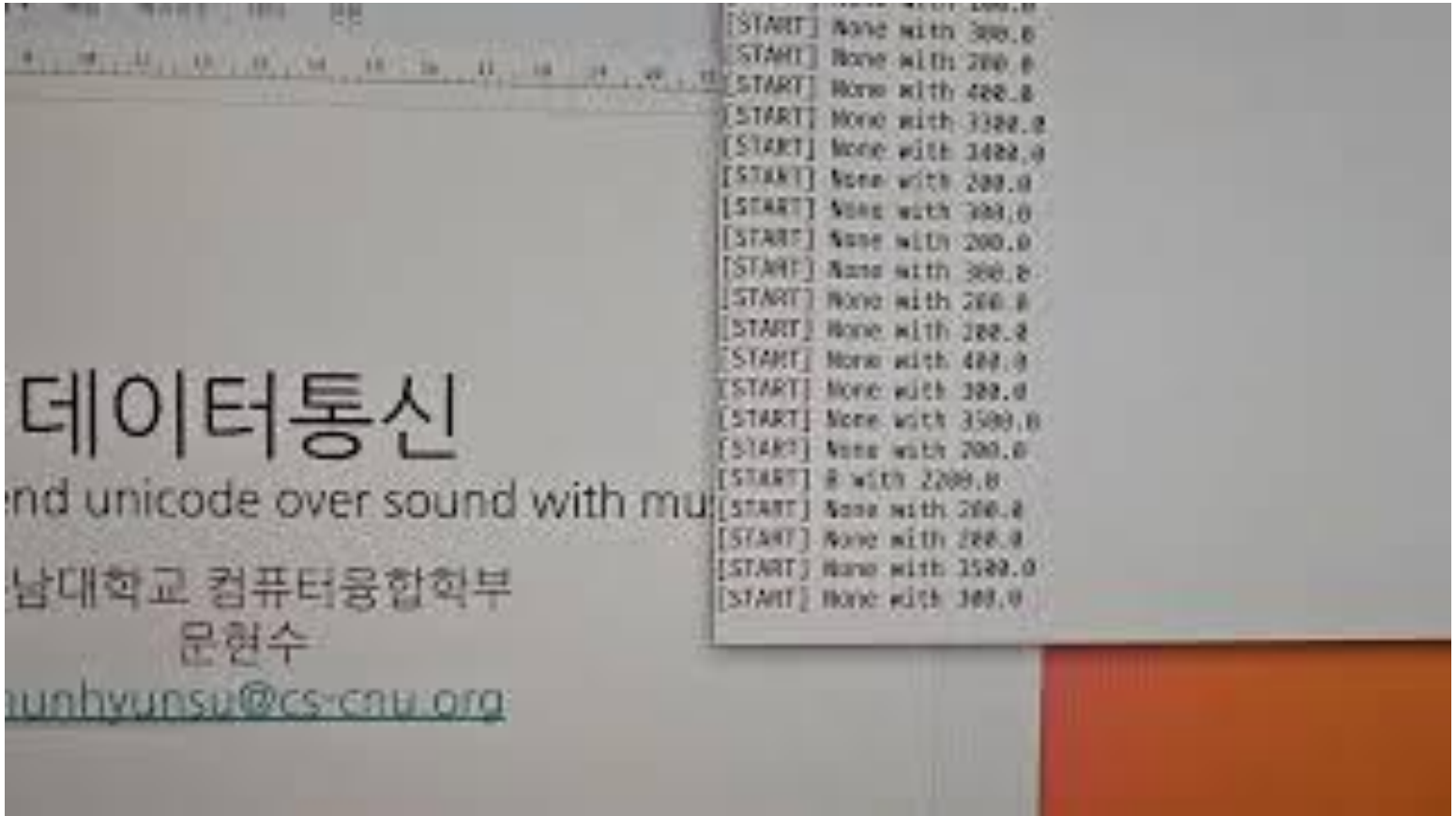
3) 소리 인식 : 반복 시도와 환경 확인이 중요!

- 마이크로 들어가는 Input 값 체크해본후, 잡음이 너무 많이 들어가지 않는지 확인해보는 것이 중요.

Data over sound with MFSK

- 물리계층 (L1) 에 어떻게 데이터를 보낼 수 있는가?
- **그런데, 소음이 있다면? ⇒ 주성분 분석!**
- **그런데, 언제 데이터가 시작하는지 모른다면? ⇒ START Signal 인식!**
- **그런데, 언제 데이터가 끝나는지 모른다면? ⇒ END Signal 인식!**
- ⇒ Send unicode over sound with music

실습 및 과제 데모


데이터통신
nd unicode over sound with mu
남대학교 컴퓨터융합학부
문현수
unhyunsu@cs.cnu.org
[START] None with 300.0
[START] None with 200.0
[START] None with 400.0
[START] None with 3300.0
[START] None with 3400.0
[START] None with 200.0
[START] None with 300.0
[START] None with 200.0
[START] None with 300.0
[START] None with 200.0
[START] None with 200.0
[START] None with 400.0
[START] None with 300.0
[START] None with 3500.0
[START] None with 200.0
[START] 0 with 2200.0
[START] None with 200.0
[START] None with 200.0
[START] None with 3500.0
[START] None with 300.0

다시 떠올리기: 단일파 (정현파)

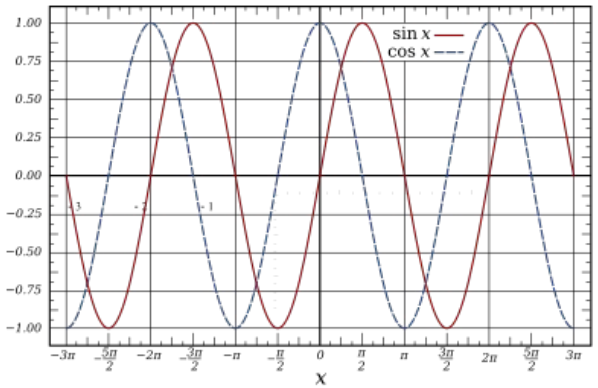
소리 (Sound): 사인파 (Sine wave)

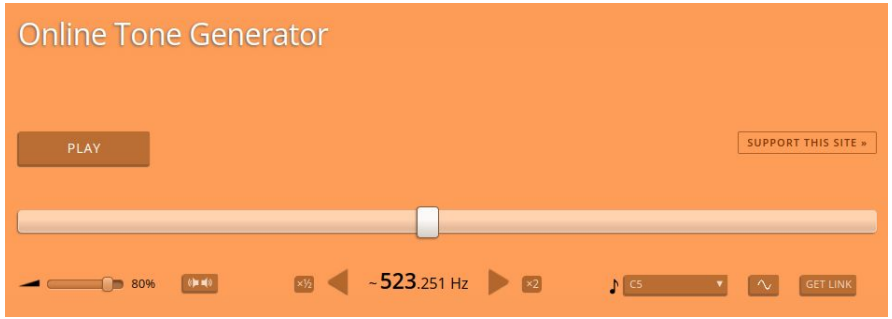
- 공기의 떨림
- 소리의 높낮이: 주파수 (Hz)
    - 소리의 세기: 진폭

주파수에 따른 소리의 높낮이 경험하기 [클릭]

(스피커 or 이어폰 필수!)

- 기준음, 가온다: C4
- 가청 주파수: 20~20,000Hz





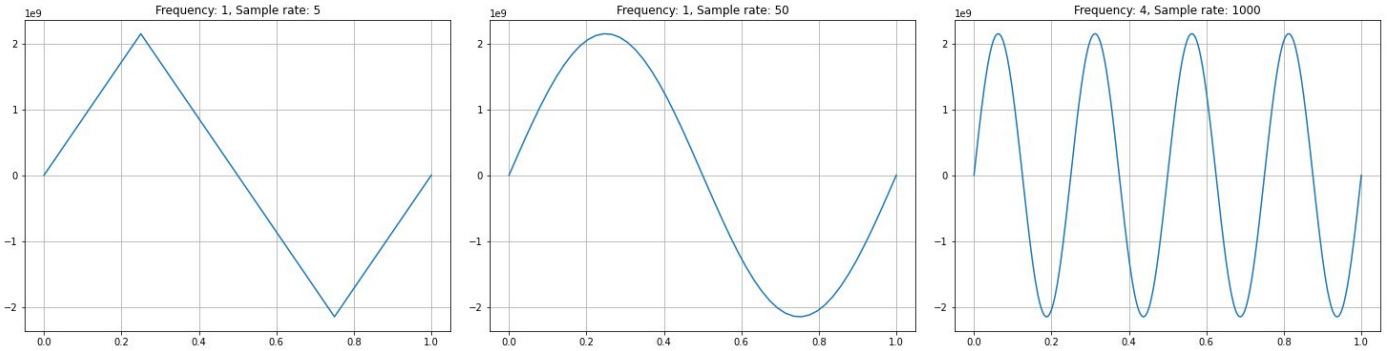
다시 떠올리기: WAV!

WAV [클릭]

- 가장 대표적인 소리 저장 포맷
- Python 3 표준 라이브러리로 제공됨!
- WAV 파일 기록시 정해야할 것!
  - Channel (채널): 1-Mono, 2-Stereo
  - Sample Width (샘플 크기): 1, 2, 4 Bytes
  - Frame Rate (샘플레이트): 초당 샘플양

Python 3 Wave [클릭]

- Scipy 를 사용하면 더 빠르게 만들 수 있으나 우리는 좀 더 깊게 이해하기 위해 직접 생성
- 샘플링부터 기록까지!

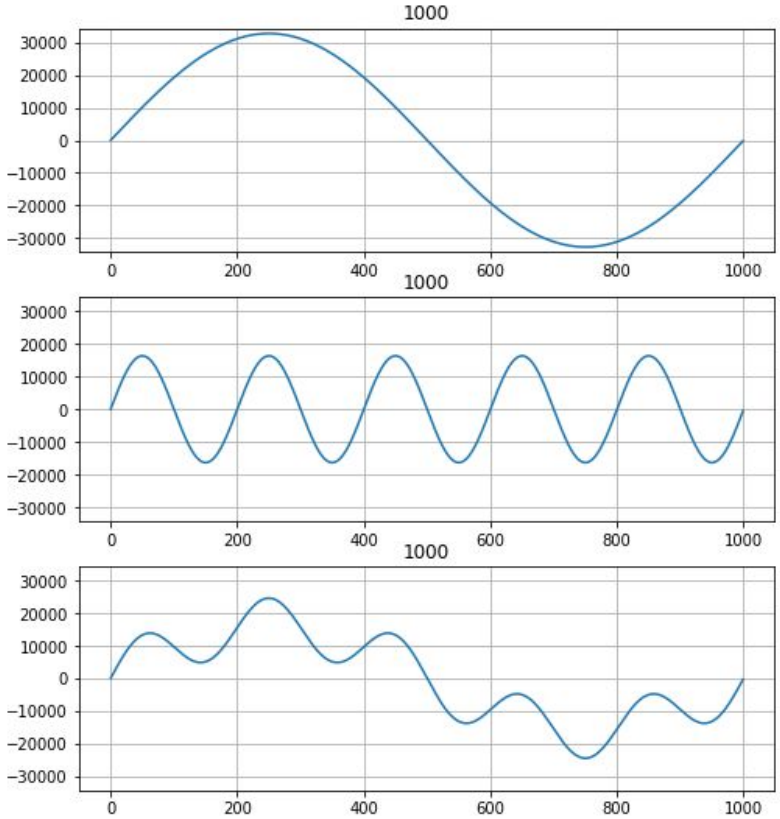


볼륨이 다른 복합 신호

- 현실에서는 여러 소리가 섞여있음
  - 여러 소리의 크기가 다름!

오른쪽 예제

- 1번: 주파수 1, 볼륨 1
- 2번: 주파수 5, 볼륨 0.5
- 3번 : 1번 + 2번



실습1) 복합 신호 들어보기

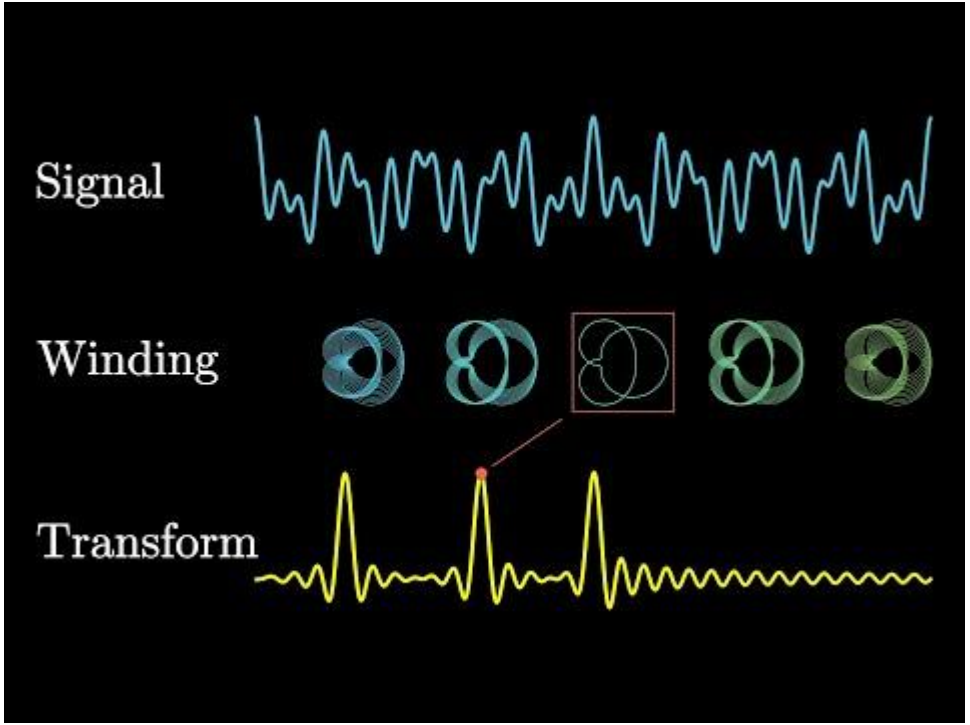
```
frequencies = [261.625, 523.251, 1046.502]
volumes = [1.0, 0.75, 0.5]
```

```
frequencies = [261.625, 523.251, 1046.502]
volumes = [0.5, 0.75, 1.0]
```

```
import pyaudio
import math
import struct
import time

INTMAX = 2**(32-1)-1
channels = 1
length = 5.0
samplerate = 48000
frequencies = [261.625, 523.251, 1046.502] # C4, C5, C6
volumes = [1.0, 0.75, 0.5]
waves = []

for frequency, volume in zip(frequencies, volumes):
    audio = []
    for i in range(int(length*samplerate)):
        audio.append(volume*INTMAX*math.sin(2*math.pi*frequency*i/samplerate))
    waves.append(audio)

track = [0]*int(length*samplerate)
for i in range(len(track)):
    for w, v in zip(waves, volumes):
        track[i] = track[i] + w[i]
    track[i] = round(track[i] / len(waves))

p = pyaudio.PyAudio()

stream = p.open(format=pyaudio.paInt32,
                    channels=channels,
                    rate=samplerate,
                    output=True)
chunk_size = 1024
for i in range(0, len(track), chunk_size):
    chunk = track[i:i+chunk_size]
    stream.write(struct.pack('<' + ('l'*len(chunk)), *chunk))

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

푸리에 변환 (Fourier Transfer)

- 소리를 다수의 사인파로 분해
  - 모든 소리는 다수의 사인파가 조합
  - 주파수의 성분 (사인파) 분석
  - 1~n개의 주성분 선택
  - 가장 진폭이 큰 사인파 -> 들을때 제일 세고 뚜렷
  - 그 소리의 음높이 파악 가능
- 오른쪽 영상 참고



실습2) 고속 푸리에 변환으로 최대 신호 추출

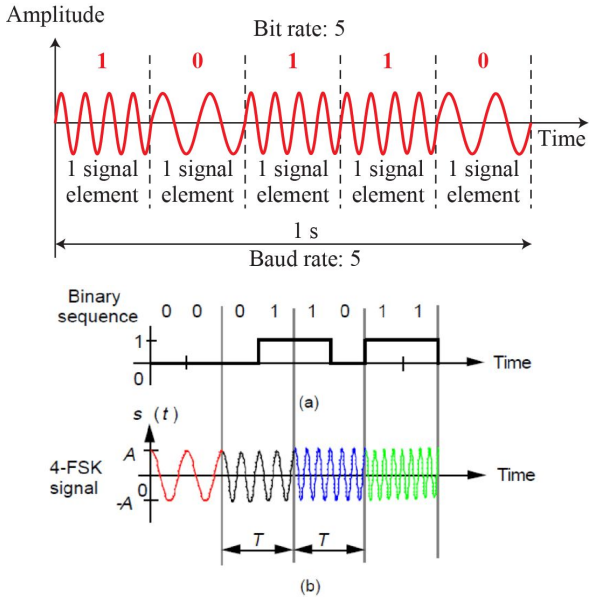
- 고속 푸리에 변환 해보기

  (직접 구현 x, scipy 라이브러리 활용)
- scipy 설치
  - `pip install scipy`
- scipy.fftpack.fftfreq() [클릭]
  - 푸리에 변환으로 반환된 배열의 index 별 주파수
- scipy.fftpack.fft(track) [클릭]
  - 고속 퓨리에

```
import math

import scipy.fftpack
import numpy as np

INTMAX = 2**(32-1)-1
channels = 1
length = 5.0
samplerate = 48000
frequencies = [261.625, 523.251, 1046.502] # C4, C5, C6
volumes = [1.0, 0.75, 0.5]
waves = []

for frequency, volume in zip(frequencies, volumes):
    audio = []
    for i in range(int(length*samplerate)):
        audio.append(volume*INTMAX*math.sin(2*math.pi*frequency*i/samplerate))
    waves.append(audio)

track = [0]*int(length*samplerate)
for i in range(len(track)):
    for w, v in zip(waves, volumes):
        track[i] = track[i] + w[i]
    track[i] = track[i] / len(waves)

freq = scipy.fftpack.fftfreq(len(track), d=1/samplerate)
fourier = scipy.fftpack.fft(track)
print(freq[np.argmax(abs(fourier))])

for i in range(len(freq)):
    if 261.125 <= freq[i] and freq[i] <= 262.125:
        print(f'{i} => {freq[i]}')
    elif 522.751 <= freq[i] and freq[i] <= 523.751:
        print(f'{i} => {freq[i]}')
    elif 1046.002 <= freq[i] and freq[i] <= 1047.002:
        print(f'{i} => {freq[i]}')
```

Multiple Frequency shift keying (MFSK)

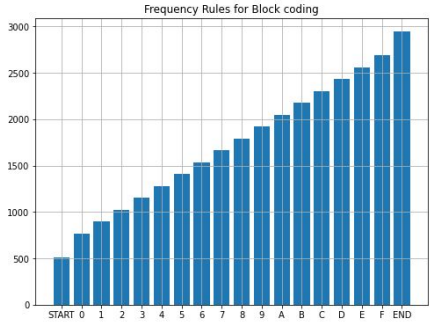
- 더 빠른 전송
- 오른쪽 예제: 주파수로 4비트씩 전송
  - Unit size가 0.1 초 일 때, 1 Byte 는 몇 초?




```
Frequency Rules:
{'START': 512,
 '0': 768,
 '1': 896,
 '2': 1024,
 '3': 1152,
 '4': 1280,
 '5': 1408,
 '6': 1536,
 '7': 1664,
 '8': 1792,
 '9': 1920,
 'A': 2048,
 'B': 2176,
 'C': 2304,
 'D': 2432,
 'E': 2560,
 'F': 2688,
 'END': 2944}
```

실습3) 주파수별 4비트 데이터 정의

- 데이터에 대한 매핑 정보는 사전에 서로 정의되어야 함!
- 이번주 실습 블록 코딩 데이터 정의
  - 전송 시작 주파수
  - 전송 끝 주파수
  - 0~F 까지의 각 주파수
  - **시작과 끝 주파수와 데이터 사이에는 2 Step**
  - **전송 시작, 전송 끝은 2 Unit 전송**
  - **데이터는 1 Unit 전송**




```
FREQ_START = 512
FREQ_STEP = 128
HEX_LIST = ['0', '1', '2', '3', '4',
            '5', '6', '7', '8', '9',
            'A', 'B', 'C', 'D', 'E',
            'F']
HEX = set(HEX_LIST)

rules = {}
print('Frequency Rules:')
rules['START'] = FREQ_START
for i in range(len(HEX_LIST)):
    h = HEX_LIST[i]
    rules[h] = FREQ_START + FREQ_STEP + FREQ_STEP*(i+1)
rules['END'] = FREQ_START + FREQ_STEP + FREQ_STEP*(len(HEX_LIST)) + FREQ_STEP*2

for k, v in rules.items():
    print(f'{k}: {v}')
```

```
Frequency Rules:
{'START': 512,
 '0': 768,
 '1': 896,
 '2': 1024,
 '3': 1152,
 '4': 1280,
 '5': 1408,
 '6': 1536,
 '7': 1664,
 '8': 1792,
 '9': 1920,
 'A': 2048,
 'B': 2176,
 'C': 2304,
 'D': 2432,
 'E': 2560,
 'F': 2688,
 'END': 2944}
```

실습4) 주파수 활용 Unicode 고속 전송

- Unicode 를 소리로 변환
  - Unit: 0.1
  - Samplerate: 48000

- 지난주 Unicode over sound: 54.6초
- 이번주 Data over sound: 4초

```
import math
import struct
import wave

import pyaudio

INTMAX = 2**(32-1)-1
channels = 1
unit = 0.1
samplerate = 48000

text = '🧡💛💚💙💜'
string_hex = text.encode('utf-8').hex().upper()

audio = []
for i in range(int(unit*samplerate*2)):
    audio.append(int(INTMAX*math.sin(2*math.pi*rules['START']*i/samplerate)))
for s in string_hex:
    for i in range(int(unit*samplerate*1)):
        audio.append(int(INTMAX*math.sin(2*math.pi*rules[s]*i/samplerate)))
for i in range(int(unit*samplerate*2)):
    audio.append(int(INTMAX*math.sin(2*math.pi*rules['END']*i/samplerate)))

p = pyaudio.PyAudio()

stream = p.open(format=pyaudio.paInt32,
                    channels=channels,
                    rate=samplerate,
                    output=True)

chunk_size = 1024
for i in range(0, len(audio), chunk_size):
    chunk = audio[i:i+chunk_size]
    stream.write(struct.pack('<' + ('l'*len(chunk)), *chunk))
```

실습5) 주파수 활용 Unicode 고속 수신

- Morse code 읽을 때와 비슷한 방법

Decode 방법

1. 주성분 분석
2. 주성분에 매핑되는 블록 코드 검색
    - 오차를 고려해서 padding 처리

```
Raw hex:
START
START
F09FA7A1F09F929BF09F929AF09F9299F09F929C
END
END
Decoded: 🧡💛💚💙💜
```

```
import wave
import struct

import scipy.fftpack
import numpy as np

unit = 0.1
samplerate = 48000
padding = 5

filename = '실습6-example4-fsk.wav'

print('Raw hex:')
text = ''
with wave.open(filename, 'rb') as w:
    framerate = w.getframerate()
    frames = w.getnframes()
    audio = []
    for i in range(frames):
        frame = w.readframes(1)
        d = struct.unpack('<l', frame)[0]
        audio.append(d)
        if len(audio) >= unit*framerate:
            freq = scipy.fftpack.fftfreq(len(audio), d=1/samplerate)
            fourier = scipy.fftpack.fft(audio)
            top = freq[np.argmax(abs(fourier))]

            data = ''
            for k, v in rules.items():
                if v-padding <= top and top <= v+padding:
                    data = k

            if data == 'END':
                print()
                print(data, end='')
            if data != 'START' and data != 'END':
                text = f'{text}{data}'
                print(data, end='')
            if data == 'START':
                print(data)

            audio.clear()
    print()

print(f'Decoded: {bytes.fromhex(text).decode("utf-8")}')
```

과제: Send Unicode via MFSK

해야할 내용

1) 송신

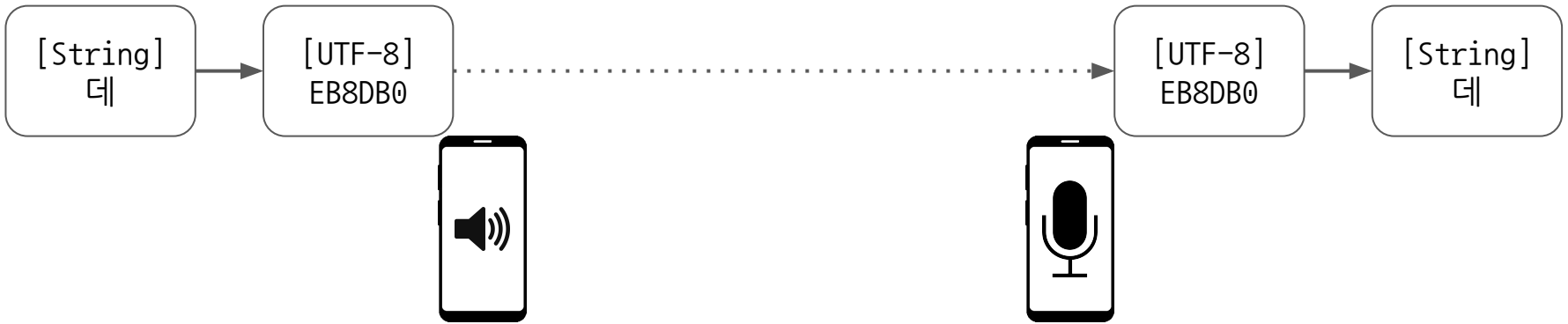
   입력받은 Text를 Unicode로 변환해서 송신 (실습코드와 거의 유사)

2) 수신

   오디오를 입력받아서, 푸리에 변환으로 주파수 계산 -> 디코딩

실습5에서 Morse code 제거: Unicode!

- Unicode Encoding 된 Hex String 을 바로 전송



Unicode over Sound 사용자 인터페이스

- 사용자가 데이터 전송, 입력 선택
- 데이터 송신을 선택하면 전송할 데이터를 다시 입력받아야 함!
- 데이터 수신을 선택하면 녹음을 시작하고 동시에 들어오는 데이터를 해석해야 함!
  - 환경에 맞게 소리에 대한 Threshold 설정!
  - START Frequency 처리!

```
import math
import statistics
import struct
import time
import wave

import pyaudio

INTMAX = 2**(32-1)-1


def send_data():
    pass


def receive_data():
    pass


def main():
    while True:
        print('Unicode over Sound with Noise')
        print('2025 Spring Data Communication at CNU')
        print('[1] Send Unicode over sound (play)')
        print('[2] Receive Unicode over sound (record)')
        print('[q] Exit')
        select = input('Select menu: ').strip().upper()
        if select == '1':
            send_data()
        elif select == '2':
            receive_data()
        elif select == 'Q':
            print('Terminating...')
            break;


if __name__ == '__main__':
```

보고서에 꼭 적어야 할 것!: 혼자서 or 둘이서

- 가능하면 컴퓨터 2대 이상! 방법이 없다면 혼자, 스마트폰을 이용해서

- 컴퓨터 (스피커, 마이크 포함) 2대 이상
  - 한쪽에서 재생
  - 다른 한쪽에서 녹음
- 컴퓨터 1대, 스마트폰 1대
  - 컴퓨터에서 재생, 동시에 스마트폰으로 녹음
  - 스마트폰으로 녹음한 파일 재생, 동시에 컴퓨터에서 녹음

목표1) 소음에 대한 처리

- Data 에 포함된 소음은 주성분 분석으로 처리될 것!
- DATA_THRESHOLD 와 같은 변수를 선언해두고 해당 변수보다 큰 값만 '소리'로 인정
    - OBS Noise gate [링크]

```
if statistics.stdev(audio_data) >= DATA_THRESHOLD:
```

목표2) 데이터가 언제 시작하는지 알아내기 (동기화)

- START Signal 을 인식해야함!
  - Synchronization 을 위하여 UNIT 보다 짧은 주기로 START 신호 처리 필요!
  - START Signal 은 2 unit!
- START Signal 도착 ⇒ 데이터의 시작
- 따라서, 계속 녹음하면서 데이터가 시작하는 순간부터 처리 시작

```
if data == 'START':
```

목표3) 데이터가 언제 끝나는지 알아내기 (동기화)

- END Signal 을 인식해야함!
  - END Signal 은 2 unit!
- END Signal 도착 ⇒ 데이터의 끝
- 따라서, END Signal 연속 N번 도착하면 녹음 종료

```
if data == 'END':
```

※입/출력요구사항※

- 입력 데이터
  - 길이: 7글자 이상
  - 알파벳 , 한글 , 특수문자, 이모지, 띄어쓰기 중 3가지 이상 포함
- 실시간 출력
  - 완료될 때까지 주기적으로 (1chunk 마다)
    - 현재 상태 [START], [DATA], [END]
    - 소음이 아닌 데이터가 인식될 때: 데이터(hex string) 출력
    - 소리 크기 출력
  - 녹음이 완료(END)되면
    - 디코딩 결과 원본 텍스트 출력

실습6 과제 정리 및 채점 배점

채점 배점:

- 보고서 (제출): 1점
- Unicode 송신 기능 완성: 3점
- Unicode 수신 기능 완성: 3점
  - 디코딩해서 출력할 것!
- 목표 1,2,3 완성 : 각각 1점
  - 데이터 도착할 때부터 연산: Current Data 출력
  - Synchronization 문제 해결 알고리즘 설명
    - START Signal 처리 방법
    - END Signal 처리 방법

- Timing 미준수시 감점

- zip 파일 압축: 6주차-학번-이름.zip
  - **보고서 (학번-이름.pdf)**
    - 목표 1, 2, 3을 어떻게 해결했는지 보고서에 포함!
      - 알고리즘 설명 or 코드 설명
    - 송신/수신 영상 제출 필수!
      - Youtube or 영상 파일
  - **소스코드 (.py)**
- e-learning 사이버캠퍼스 제출
- 설문조사: [클릭] - 희망자 작성

- 제출 기한: 2025. 04. 16(수). 23:59 (+ 추가 제출 1일)
  추가제출기간내 제출시 30% 감점.