

## 전기전자공학부 [REDACTED]

### TTS와 ASR을 결합한 음성대화프로그램

#### 개요

##### 프로그램 제작전 확인사항

ASR, TTS의 사전정보를 탐색한다.

##### 예제 코드 preview

예제로 주어진 코드를 통해 제작전 고려사항을 확인한다.

##### 프로그램 제작 및 테스트

preview를 통해 확인한 사항을 고려하여 제작한 프로그램을 테스트한다.

#### 확인 사항

해당 프로그램을 만들기 앞서 ASR코드에서 사용되는 speech\_recognition모듈의 r.recognize\_google()과 TTS코드에서 사용되는 gtts 모듈을 테스트 하여 어떤 동작을 하는지 알아본다.

ASR과 TTS코드를 실행하고 안녕! ,안녕~을 각각 ASR에는 음성으로 TTS에는 문자로 입력을 하여 해당 코드가 억양, 운율감을 구분하는지 확인을 하도록 한다.

#### ASR preview

다음은 공식문서에서 확인 할 수 있는 내용이다.

<<https://cloud.google.com/speech-to-text/docs/speech-to-text-requests?hl=ko>>

##### 신뢰값

`confidence` 값은 0.0과 1.0 사이의 추정치이며, 오디오의 각 단어에 할당된 '가능성' 값을 집계하여 계산됩니다. 숫자가 클수록 각 단어가 정확하게 인식되었을 가능성이 높아집니다. 이 필드는 일반적으로 최상위 가설에만 제공되며 `is_final=true` 인 결과에 대해서만 제공됩니다. 예를 들어, `confidence` 값을 사용하여 대체 결과를 사용자에게 표시할지 아니면 사용자에게 확인을 요청할지를 결정할 수 있습니다.

공식 문서를 통해 확인한 내용중 google의 Asr은 신뢰값 confidence변수가 사용되는 것을 확인할 수 있었다. 이를 통해 해당 모듈은 '가능성' 즉 확률기반 HMM을 이용하여 음성인식을 한다는 것을 알 수 있다.

그러나 이 모델은 **confidence** 점수뿐만 아니라 문장 컨텍스트와 같은 다른 여러 가지 신호를 기반으로 하여 '최상의' 결과를 결정합니다. 이러한 이유로 최상위 결과의 신뢰도 점수가 최고점이 아닌 경우가 있을 수 있습니다. 대체 결과를 여러 개 요청하지 않은 경우 반환된 하나의 '최상의' 결과는 예상보다 낮은 신뢰값을 가질 수 있는데, 예를 들어 많이 쓰이지 않는 단어가 사용되었을 때 이러한 경우가 발생할 수 있습니다. 희귀어는 올바르게 인식되더라도 낮은 '가능성' 값을 가질 수 있습니다. 컨텍스트에 따라 희귀어가 가장 적합한 옵션으로 결정되면 대체 옵션보다 **confidence** 값이 낮더라도 최상위에 결과가 표시됩니다.

추가적인 내용을 통해 문장의 구조와 단어간의 문법적 관계들을 함께 고려하는 것을 알 수 있다. 문법적인 요소가 고려되므로 ASR에 사용된 모듈은 고립어 인식기가 아닌 연결어 인식기로 사용되었음을 알 수 있다.

ASR 실험을 통해 알아볼 점은 본인의 말의 억양과 운율을 구분지어 출력이 가능한지를 통해 평서문과 의문문등을 구분할 수 있는지 알아보도록 한다.

## TTS preview

Local accent	Language code ( <b>lang</b> )	Top-level domain ( <b>tld</b> )
English (Australia)	en	com.au
English (United Kingdom)	en	co.uk
English (United States)	en	us
English (Canada)	en	ca
English (India)	en	co.in
English (Ireland)	en	ie
English (South Africa)	en	co.za
French (Canada)	fr	ca
French (France)	fr	fr
Mandarin (China Mainland)	zh-CN	any
Mandarin (Taiwan)	zh-TW	any
Portuguese (Brazil)	pt	com.br

공식 문서에서 gtts 모듈은 현지 악센트(localize accent)는 파라미터로 적용이 가능하다는 사실을 확인 할 수 있다.

```
>>> from gtts import gTTS
>>> tts = gTTS('hello', lang='en', tld='com.au')
>>> tts.save('hello.mp3')
```

( tld 파라미터를 통해 영어의 억양을 호주식으로 설정하는 예시)

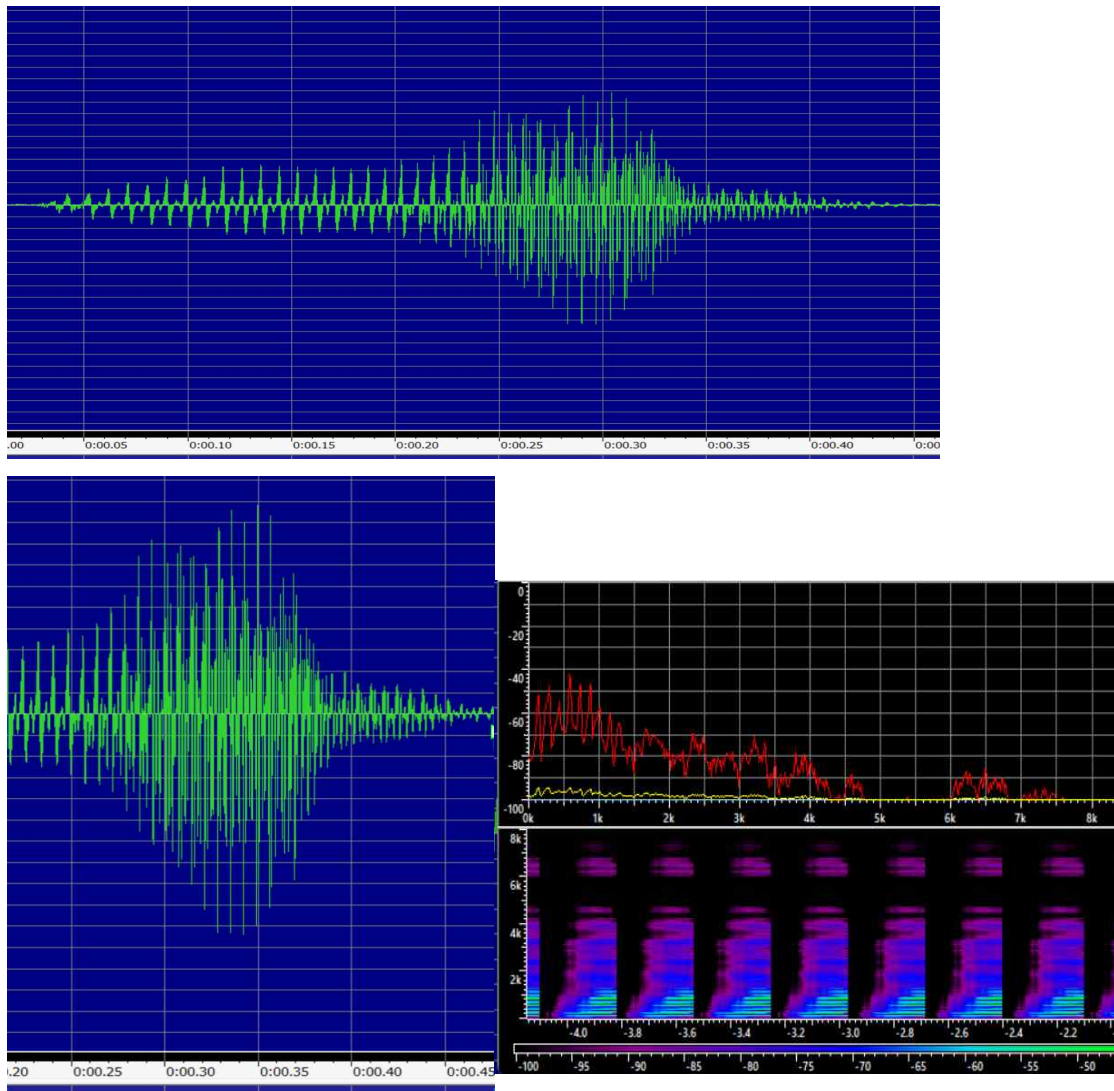
실제 실험을 통해 안녕!, 안녕~ 등을 모두 같게 인식을 하는지 확인해 보고, ASR과 마찬가지로 gtts모듈을 통해 의문문과 평서문등 억양을 구분하여 출력이 가능한지 간단한 실험을 통해 알아보려한다.

## 1. ASR

안녕! 안녕~을 구분할 수 있는가를 test 하였다.

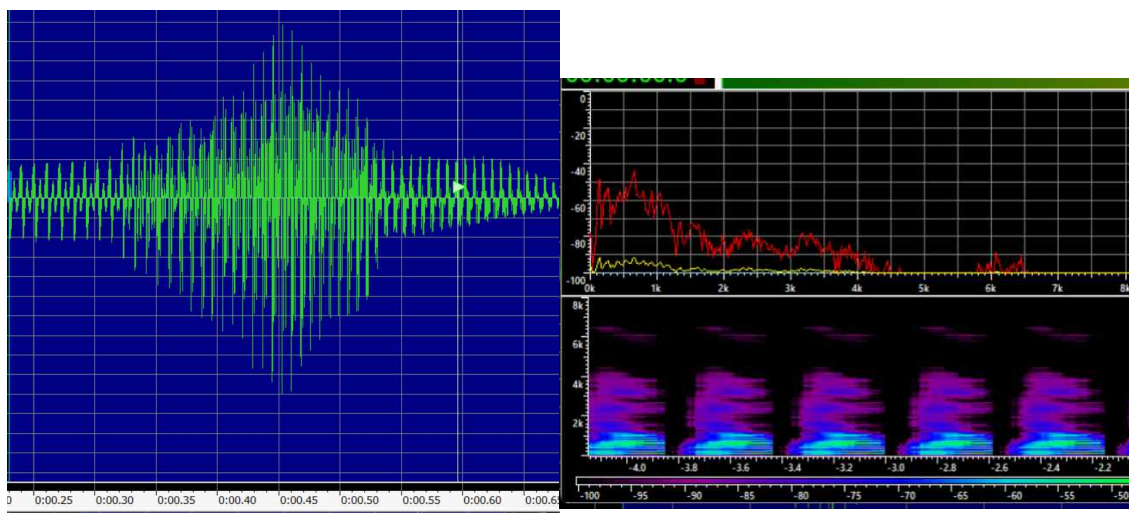
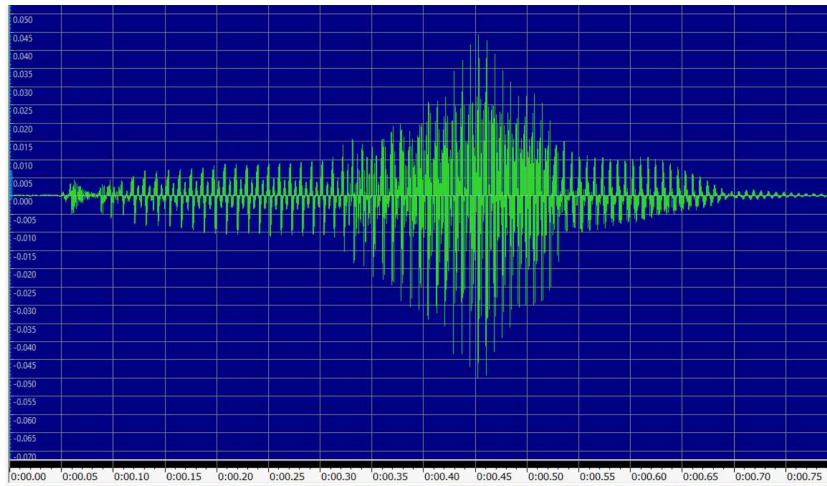
각각의 소리가 다른 것을 신호의 파형을 통해 확인하고 이때 각각의 음성신호에 따라 ASR 모듈이 인식하고 반환하는 값이 다른지를 확인한다.

‘안녕!’을 녹음한 결과



<각각 ‘안녕!’의 신호파형, ‘녕!’의 신호파형, ‘녕!’의 스펙트럼과 스펙트로그램>

‘안녕~~’을 녹음한결과



<각각 ‘안녕~~’의 신호파형, ‘녕~~’의 신호파형, ‘녕~~’의 스펙트럼, 스펙트로그램>  
 ‘안녕!’의 경우와 ‘안녕~’의 경우 음성신호의 파형 및 스펙트럼 분포에서 차이를 확인할 수 있었다. 특히 안녕!과 다르게 ‘안녕~’은 끝말을 올리지 않고 ‘녕~’부분에서 길게 끌었기 때문에 같은소리지만 녹음된 시간의 길이가 달랐으며 결정적으로 ‘안녕!’의 ‘녕!’ 부분에서는 바로 진폭이 상승하지만 ‘안녕~~’의 ‘녕~~’부분은 비교적 곧바로 진폭이 상승하지 않는 차이가 있었다.

```
PS C:\Users\vtoree> & C:/Users/vtoree/AppData/Local/Programs/Python/Python310/python.exe  
e d:/code/mmp/w6/ASR.py  
Speak for 5 sec.  
안녕  
PS C:\Users\vtoree> & C:/Users/vtoree/AppData/Local/Programs/Python/Python310/python.exe  
e d:/code/mmp/w6/ASR.py  
Speak for 5 sec.  
안녕
```

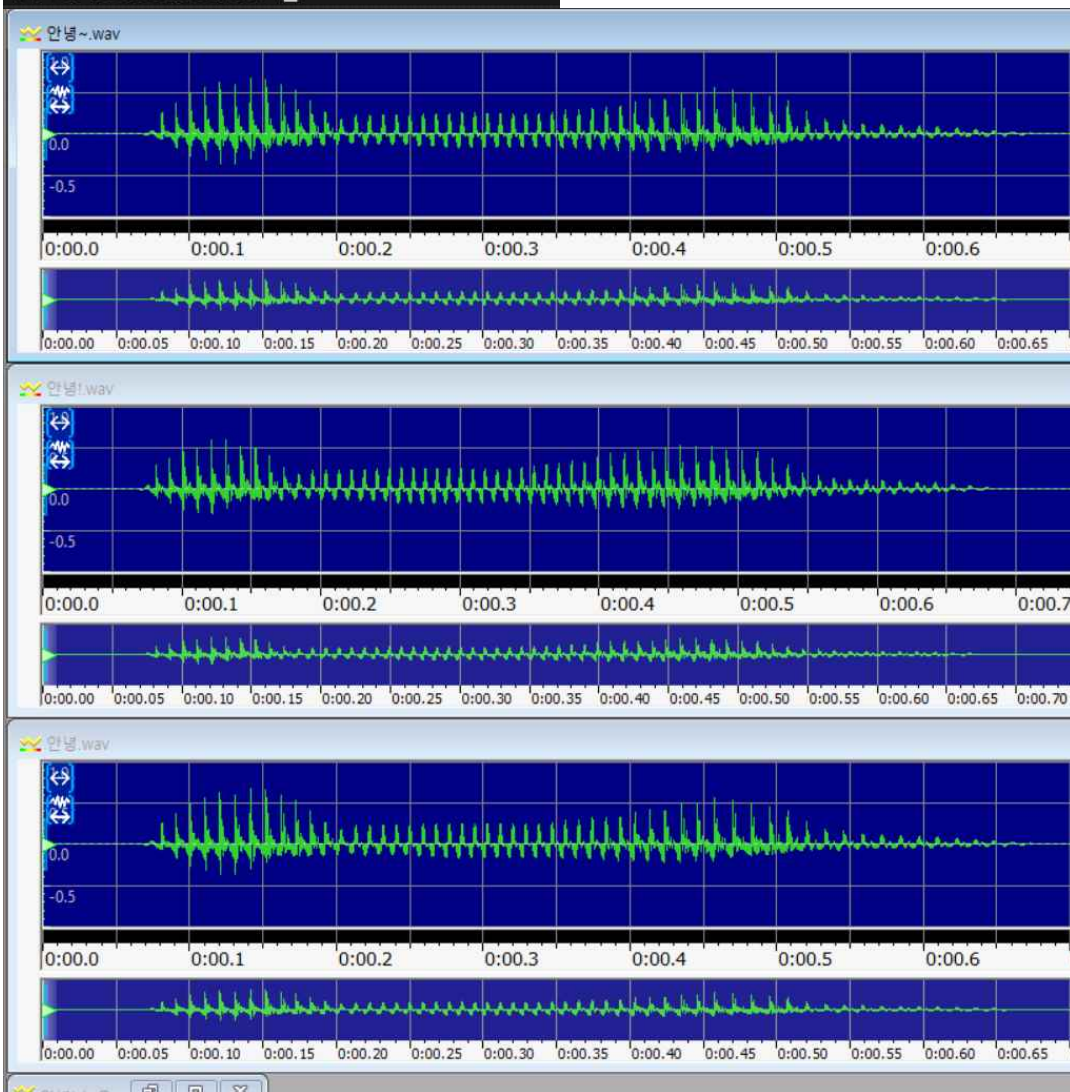
다만 두 개의 경우 모두 반환된 text는 안녕으로 억양의 차이를 반영하지 않고 단어만을 반환 받았다.

즉 해당 코드를 통해 음성을 인식시키려 한다면 억양의 차이를 반영하기는 어렵다는 점을 확인 할 수 있었다.



## 2. TTS

```
PS C:\Users\vtoree> & C:/Users/vtoree/App
e d:/code/mmp/w6/tts.py
Enter string:안녕~
PS C:\Users\vtoree> & C:/Users/vtoree/App
e d:/code/mmp/w6/tts.py
Enter string:안녕!
PS C:\Users\vtoree> & C:/Users/vtoree/App
e d:/code/mmp/w6/tts.py
Enter string:안녕
PS C:\Users\vtoree> █
```

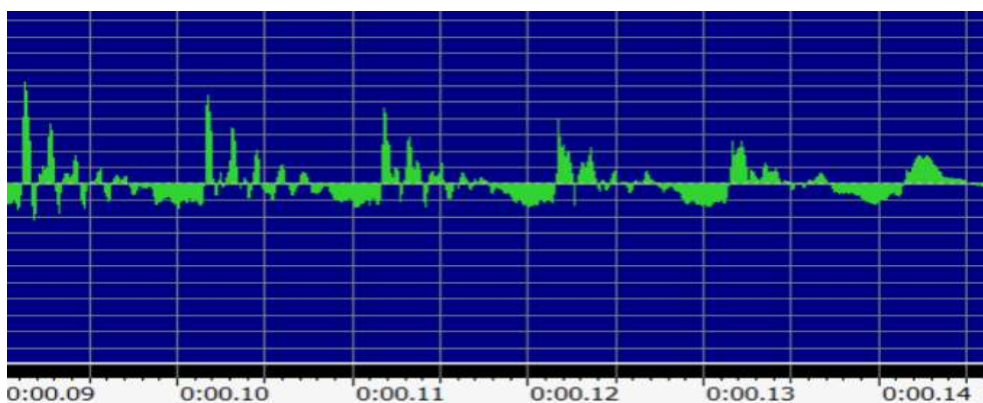
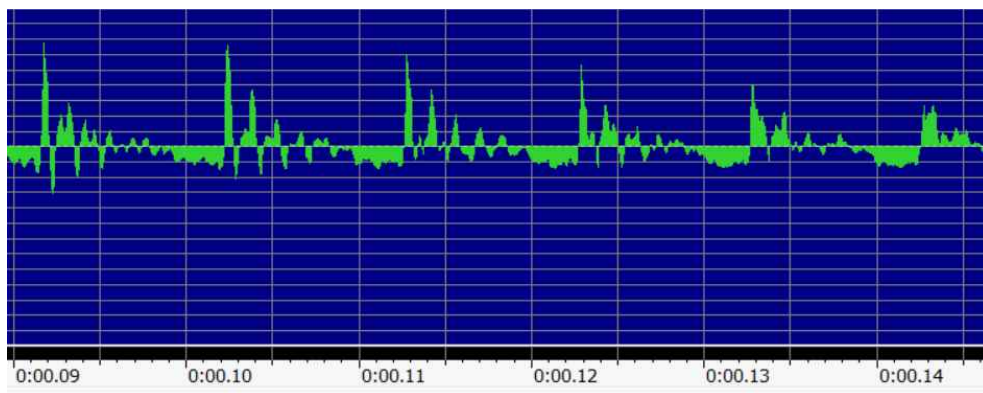
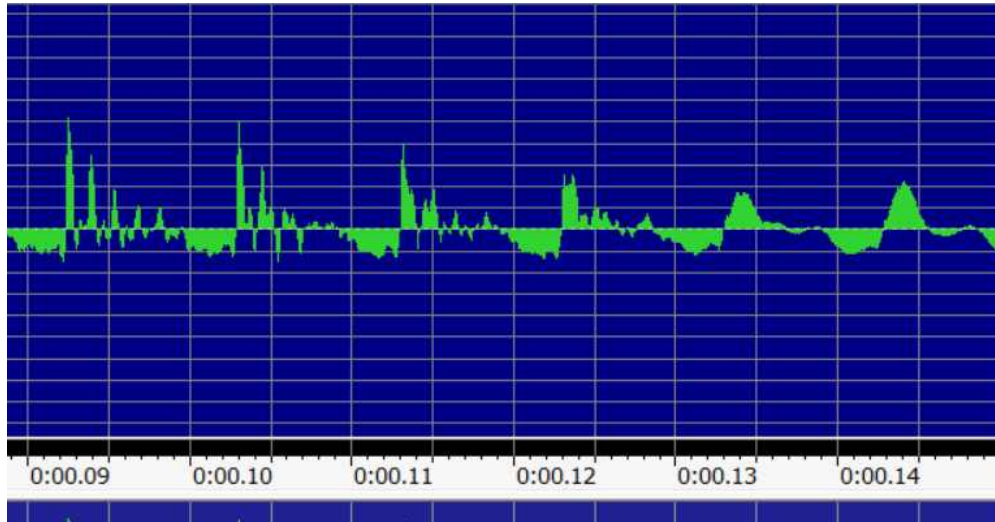


<순서대로 '안녕', '안녕!' '안녕~'의 TTS 음성신호 >

‘안녕’, ‘안녕!’ ‘안녕~’ 세가지 문자를 입력하였을 때 출력되는 TTS음성의 세가지 파형에 대한 사진이다. 실제 청감상으로는 차이를 느낄 수 없었으며 위의 사진과 같이 세가지 신호를 한번에 볼 때도 명확한 차이를 구분하기 어려웠다.

하지만 신호의 파형이 유사하게 생겼지만 미세하게 다르게 보여 앞쪽의 부분을 좀더 관찰해 보았다. 처음 ‘아’소리가 나오는 구간의 파형이 약간씩 차이를 확인할 수 있었고, 해당 부분을 좀더 확대하여 면밀하게 관찰 하였다.

<각각 ‘안녕’,안녕!’,’안녕~’에서의 ‘아’ 부분 >



동일한 소리가 나는 구간을 동일한 시간을 기준으로 확대하였을 때 아주 약간의 파형차이가 있었다. 즉 주어진 TTS코드에서 !와 ~등이 작용은 하였다고 볼 수 있으나 사람이 파악하기 어려울 정도로 효과를 주지는 못한다는 사실을 알 수 있었다.

### 3. 추가적인 실험진행.

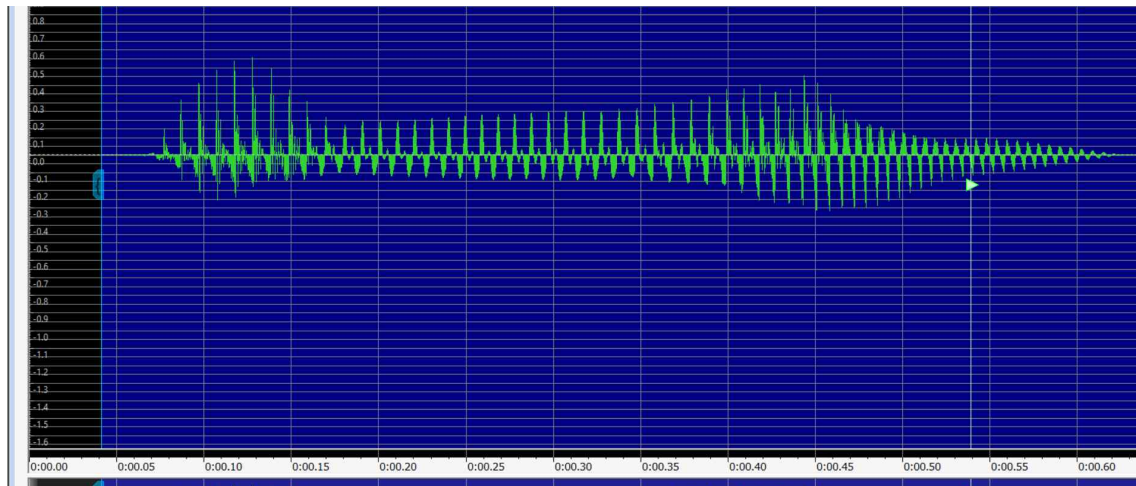
ASR을 통해 넘어가는 음성을 분석하였을 때 입력으로 들어가는 음성의 신호가 다름에도 출력되는 값은 변함없이 단어만을 출력하였다. 하지만 TTS의 경우 청감상 구분하기 어려웠으나 실제 파형들을 자세하게 관찰한결과 각각 동일한 기계음이 아닌 미묘한 차이가 있음을 발견하였다.

그렇다면 “지원하는 억양차이가 있지 않을까?” 라는 생각을 하게되었다.4

실제 preview에서 보았듯이 동일한 영어임에도 호주식 영어, 캐나다식 영어등 지역별 억양을 어느정도 지원하는 것을 확인하였다. 그래서 추가로 문법적 구분이 쉬운 의문문의 ?를 마지막으로 시도해 보려한다.

‘안녕?’을 입력한 결과

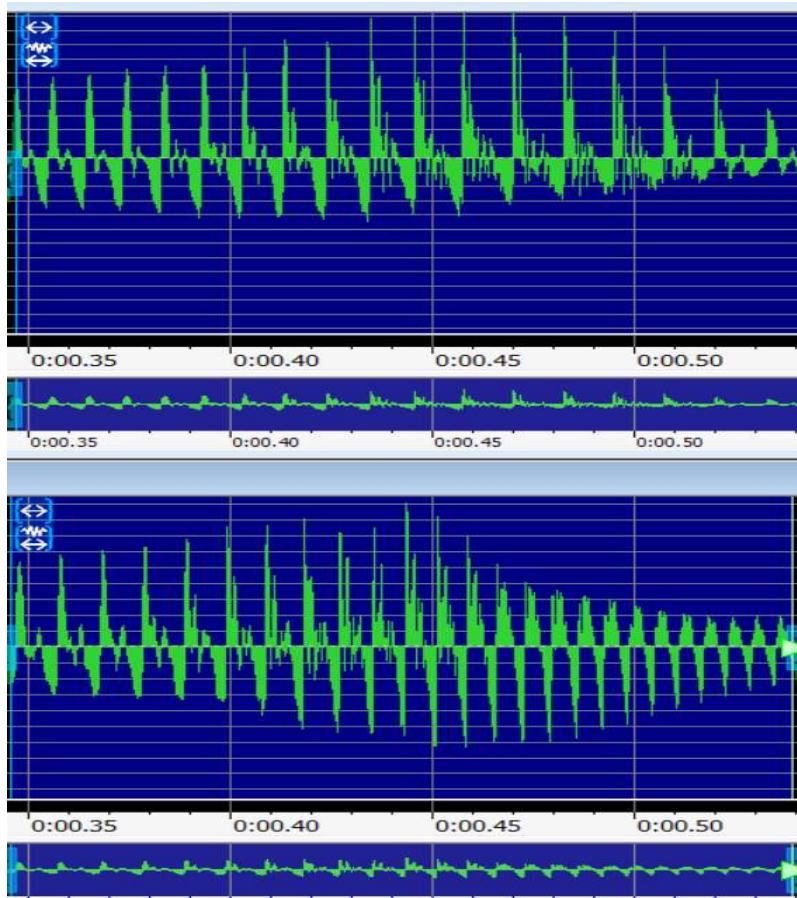
```
PS C:\Users\vtoree> & C:/Users/
e d:/code/mmp/w6/tts.py
Enter string:안녕?
PS C:\Users\vtoree> |
```



실험결과 의문형 문장부호 ‘?’를 구분하는 것을 청감상 확인 할 수 있었다.

녕의 부분은 이전실험 안녕’,안녕!’,’안녕~’ 3개의 입력에서 청감상 차이가 없이 평서문의 소리를 들었으나 ‘안녕?’에서는 ‘녕?’부분의 억약이 올라가는 차이가 발생하였다.





<위쪽은 ‘안녕’ 아래는 ‘안녕?’에서의 ‘녕’부분 음성신호다.>

실제로 ‘녕’부분의 파형을 비교하였을 때 평서문과 같은 파형은 끝부분에서 진폭에 큰 변화가 없으며 동일한 부분이 길게 발음되고 소리가 줄며 억양이 내려가는 듯한 느낌을 주었다.

아래의 ‘녕’ 부분에서는 평서문인 위의 신호와 비교하여 동일한 구간에서 진동수가 큰 것을 확인 할 수 있다. 이로 인하여 ‘녕’ 부분에서 억양이 올라간 듯한 차이를 확인 할 수 있었다.

#### 4. 음성 대화 프로그램

앞선 결과들을 가지고 다음과 같은 대화를 시도하려한다.

```
recognized_text = r.recognize_google(audio, language=lang)
print("You said: " + recognized_text)

if recognized_text == '안녕':
    response_text = "안녕하세요"
elif recognized_text == '밥은 먹었니':
    response_text = "네 그쪽은요?"
elif recognized_text == '나도 먹었어':
    response_text = "다행이네요"
```

본인의 음성인 '밥은 먹었니'의 경우 ASR로 사용되는 recognize\_google()함수가 ?를 확인해 주지 못하기 때문에 의문 부호 '?'를 포함하지 않았으며 TTS를 통해 생성되는 음성은 '?'의 억양을 구현하는 것을 확인 하였으므로 '네 그쪽은요?'와 같이 의문 부호 '?'를 적어 주었다.

진행방식은 다음과 같이 5초간 말을 할 수 있으며 답변이 나온다

각 입력된 음성과 TTS를 통한 답변들은 index를 다르게 하여 wav파일로 저장된다.

```
if run:
    print("Speak for 5 sec.")
    data = np.frombuffer(stream.read(CHUNK), dtype=np.int16)
    temp_filename = f'D:/code/mmp/w6/temp{file_index}.wav'
    response_filename = f'D:/code/mmp/w6/response{file_index}.wav'
```

## 실험을 통해 확인할 사항


1. 코드를 통해 입력으로 들어간 음성과 text로 인식된 결과를 비교한다.  
<preview에서 진행한 “안녕“과 답변 ”안녕하세요“에서의 안녕을 비교한다>
2. 텍스트를 통해 지정되어있는 답변과 출력된 음성신호를 비교한다.  
<“네 그쪽은요?“를 ”네 그쪽은요“ 로변경하여 억양의 차이를 분석한다.>
3. 기타 확인된 사항을 분석한다.

```
recognized_text = r.recognize_google(audio, language=lang)
print("You said: " + recognized_text)
```

위와 같은 코드를 통해 나의 음성이 어떻게 문자로 변환되었는지를 확인할 수 있다.

```
e d:/code/mmp/w6/talk.py
Speak for 5 sec.
You said: 안녕
Response: 안녕하세요
Speak for 5 sec.
You said: 밥은 먹었니
Response: 네 그쪽은요?
Speak for 5 sec.
You said: 나도 먹었어
Response: 다행이네요
```

<음성 입력과 TTS 출력>

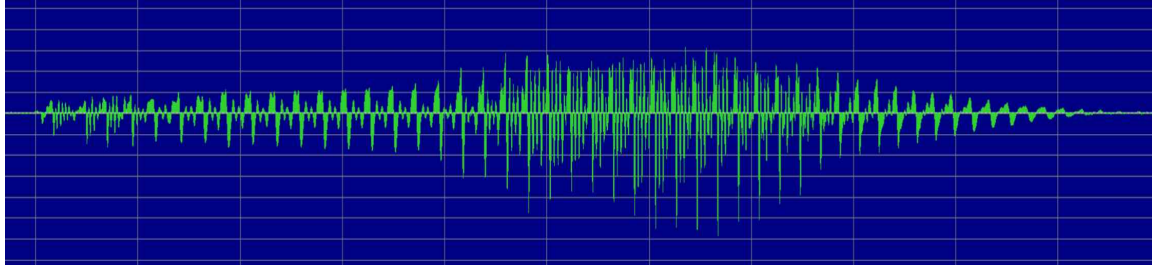


temp0  
response0  
temp1  
response1  
temp2  
response2

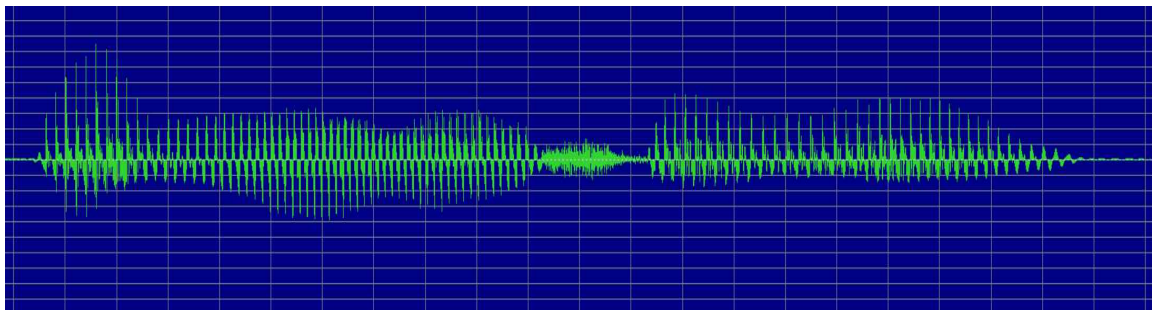
<음성입력과 TTS출력들의 파일>

첫 번째 대화 <인사>

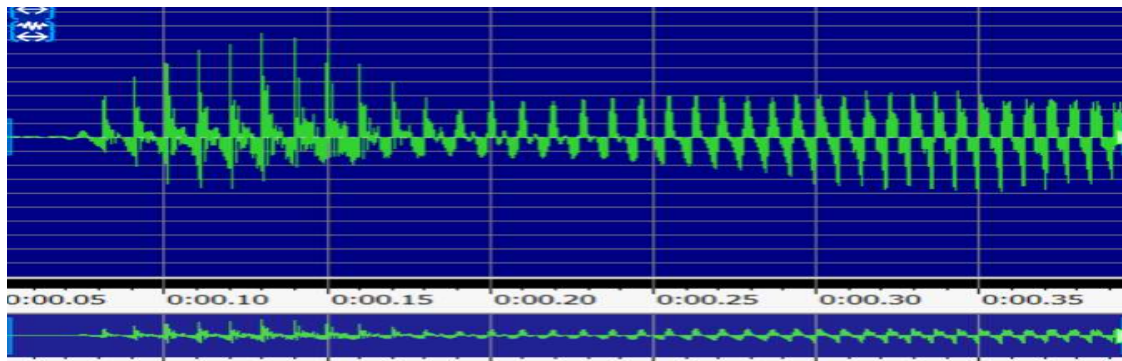
음성입력 ‘안녕’



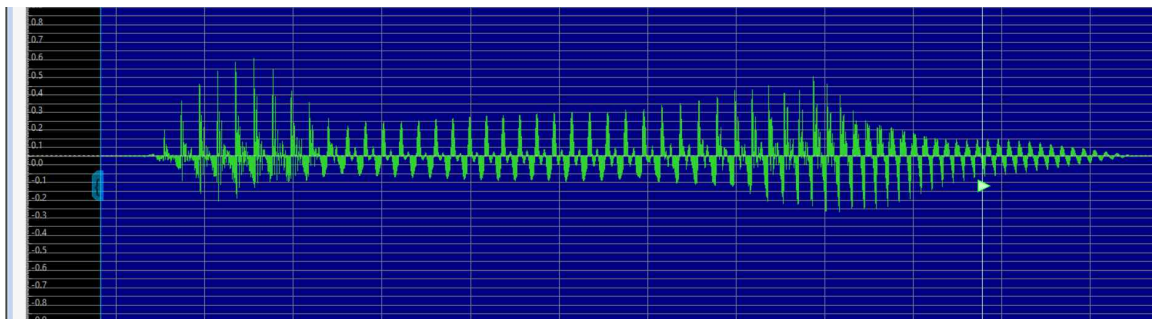
음성 출력 ‘안녕하세요’



음성 출력 ‘안녕하세요’에서 ‘안녕’부분만을 자른 것



“3. 추가실험”에서 진행한 ‘안녕?’의 파형



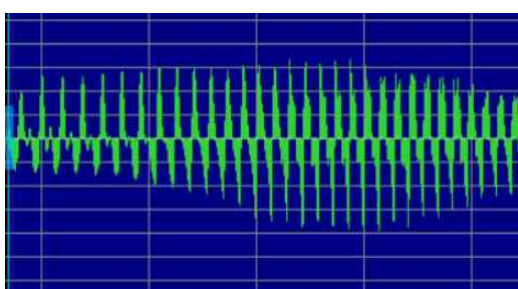
해당 4개의 사진을 통해 알 수 있는점은 같은 ‘안녕’ 이라는 말이라도 음성 신호의 파형에서 차이가 있음을 알 수 있다.

본인의 ‘안녕’에서 ‘안’은 ‘녕’보다 작은 소리였으며 TTS가 만들어낸 ‘안녕하세요’의 ‘안’은 소리가 ‘녕’보다 컸다. 이는 TTS와 본인의 화자 고유의 특성으로 보여진다.

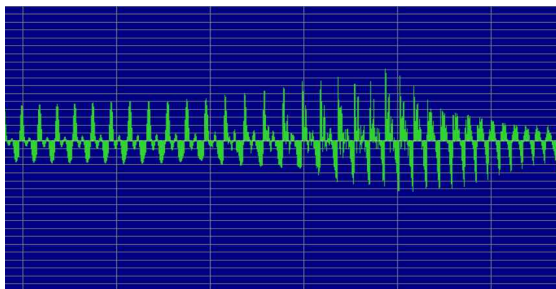
#### 조음현상을 고려한 합성 단위 선택

TTS의 ‘안녕하세요’와 ‘안녕’을 비교하면 흥미로운 사실이 있다.

‘안’부분은 동일한 파형을 보여주지만 ‘녕’에 해당 하는 뒷부분은 차이가 있다는 것이다.



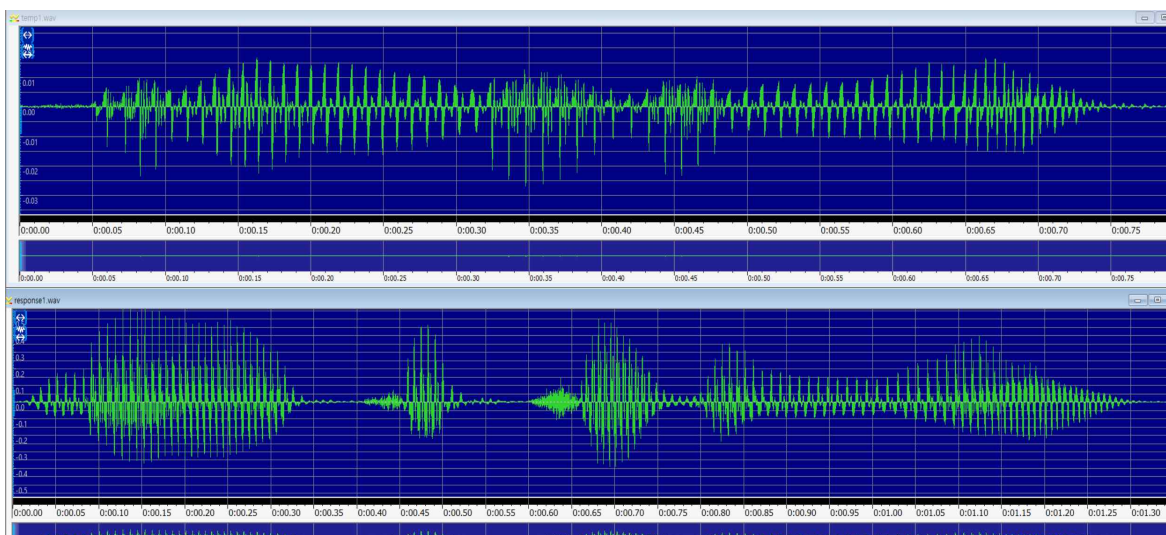
<안녕하세요>



<안녕>

이를 통해 음성 합성시 조음 현상을 고려하여 합성단위를 선택하는 것을 직접 확인 할 수 있었다. ‘안녕’과 다르게 ‘안녕하세요’는 ‘녕’ 뒤로 ‘하세요’가 오는 것을 고려하여 합성단위를 선택하여 다른 파형을 보인 것 이라고 생각된다,

#### 두 번째 대화 <밥은 먹었니?>

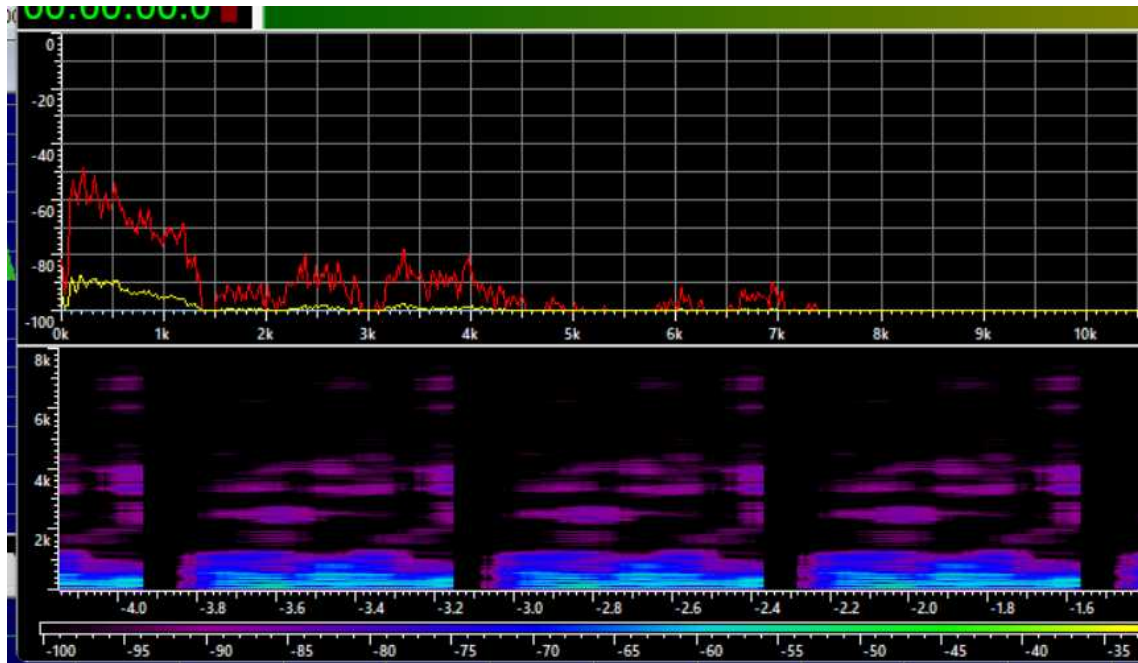


<상 :밥은 먹었니? , 하: 네 그쪽은요?>



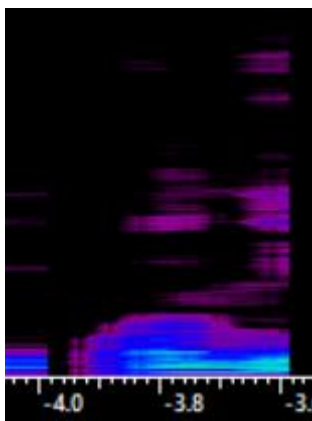
Speak for 5 sec.  
 You said: 밥은 먹었니  
 Response: 네 그쪽은요?

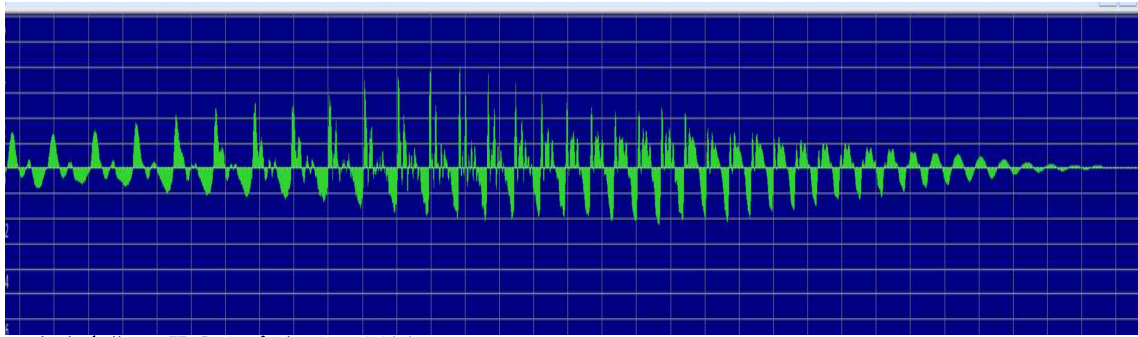
음성 입력의 경우 preview를 통해 확인했듯이 ‘?’를 구분하여 문자로 출력해주지는 않는다.



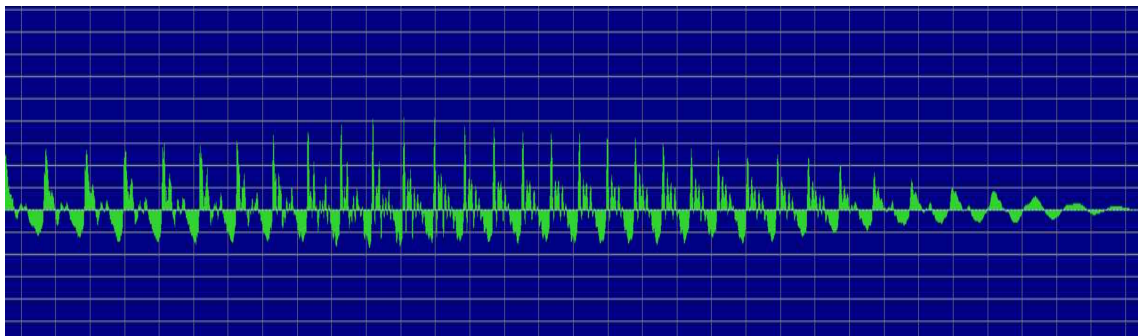
<밥은 먹었니?의 스펙트럼과 스펙트로그램>

질문이므로 의문문의 억양인 끝의 ‘니’를 올리는 소리가 들린다. 파형에서도 마지막부분에 음이 올라감에 따라 고주파 성분이 조금씩 늘어나는 것을 확인 할 수 있다.





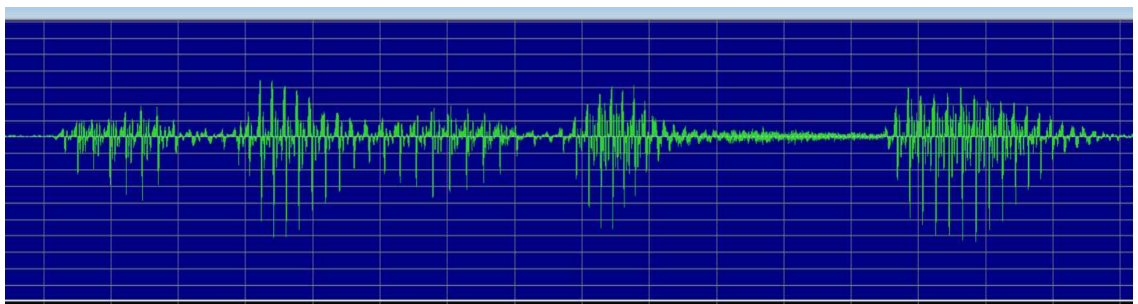
<답변 ‘네 그쪽은요?’의 요? 부분>



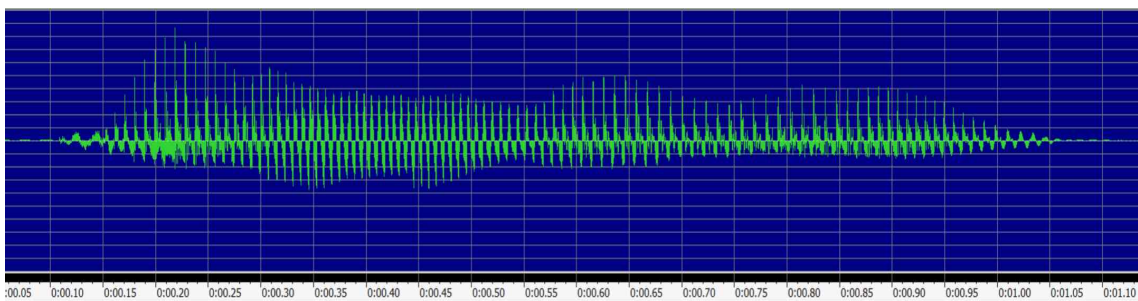
<해당 부분을 변경하고 진행한 ‘네 그쪽은요’의 ‘요’>

위의 의문문은 마지막 ‘요?’의 억양을 올려 해당 구간에서 신호의 진동수가 비교적 급격히 줄었다가 줄어드는 모습을 보이며 아래의 경우 평서문으로 비교적 천천히 진동수가 증가했다가 감소하는 것을 확인 할 수 있다.

#### 마지막 대화



<입력 : 나도 먹었어>



<TTS : 다행이네요>

마지막 3번째 대화가 끝난후 확인 나와 TTS의 고유특성을 한가지 더 확인할 수 있었다.  
나의 음성신호는 모든 단어와 상황에서 첫 음절에서 낮은 진폭을 보여주었으며  
TTS 는 반대로 첫 음절에서 항상 큰 진폭의 신호를 보였다.

실제로 나의 음성은 처음에는 말소리가 작다는 것을 확인할 수 있었으며  
TTS의 음성은 처음부터 소리가 크게 들린다는 것을 청감상 확인이 가능하였다.

## 결론

해당 실험을 통해 말을 할 때 화자 고유의 특성이 있음을 TTS 와 본인의 음성 신호들을 비교하여 확인할 수 있었으며 TTS가 ‘?’와 같은 의문부호를 인식하고 억양에 차이를 준다는 사실을 알 수 있었다. 또한 “안녕하세요”와 ‘안녕’을 비교분석하며 같은 단어라도 조음현상을 고려하여 합성 단위를 선택하기 때문에 음성신호에서 차이가 발생한다는 것을 확인할 수 있었다,

ASR이 다른 억양의 신호임에도 단순한 단어들로 구분된 문장만을 문자로 반환해 주어 추가적인 구성의 어려움이 있던것과, TTS가 많은 억양을 구분지어 출력해 주지는 못한다는 점은 해당 실험에서 아쉽게 느껴지는 부분이었다.

## REFERENCE

### gtts에 대하여

<https://gtts.readthedocs.io/en/latest/>

<https://pypi.org/project/gTTS/>

### speech recognition

<https://wicg.github.io/speech-api/>

<https://cloud.google.com/speech-to-text>