Midterm

Artificial Intelligence

Exercise 1 - Web based Audio App

Audio Recording Task

Recording the two requested poem lines involved setting the recording settings to 48.000Hz, 24-bit and mono. The headset I'm using is definitely not the best tool for good sound quality but most of the disturbance came from the notebook fan blaring due to the work load and 30 degree Celsius. I recorded multiple times to make sure there were no amplitude outliers due to harsh pronunciation. After a successful run the first task was to normalize the recording. It means bringing the amplitude level up to the maximum possible preventing clipping. Since I've had a lot of fan noise I used part of the recording that only contained background noise as template to remove that noise. The last step was to trim the silence at the beginning and the end and to save the recording as WAV file.

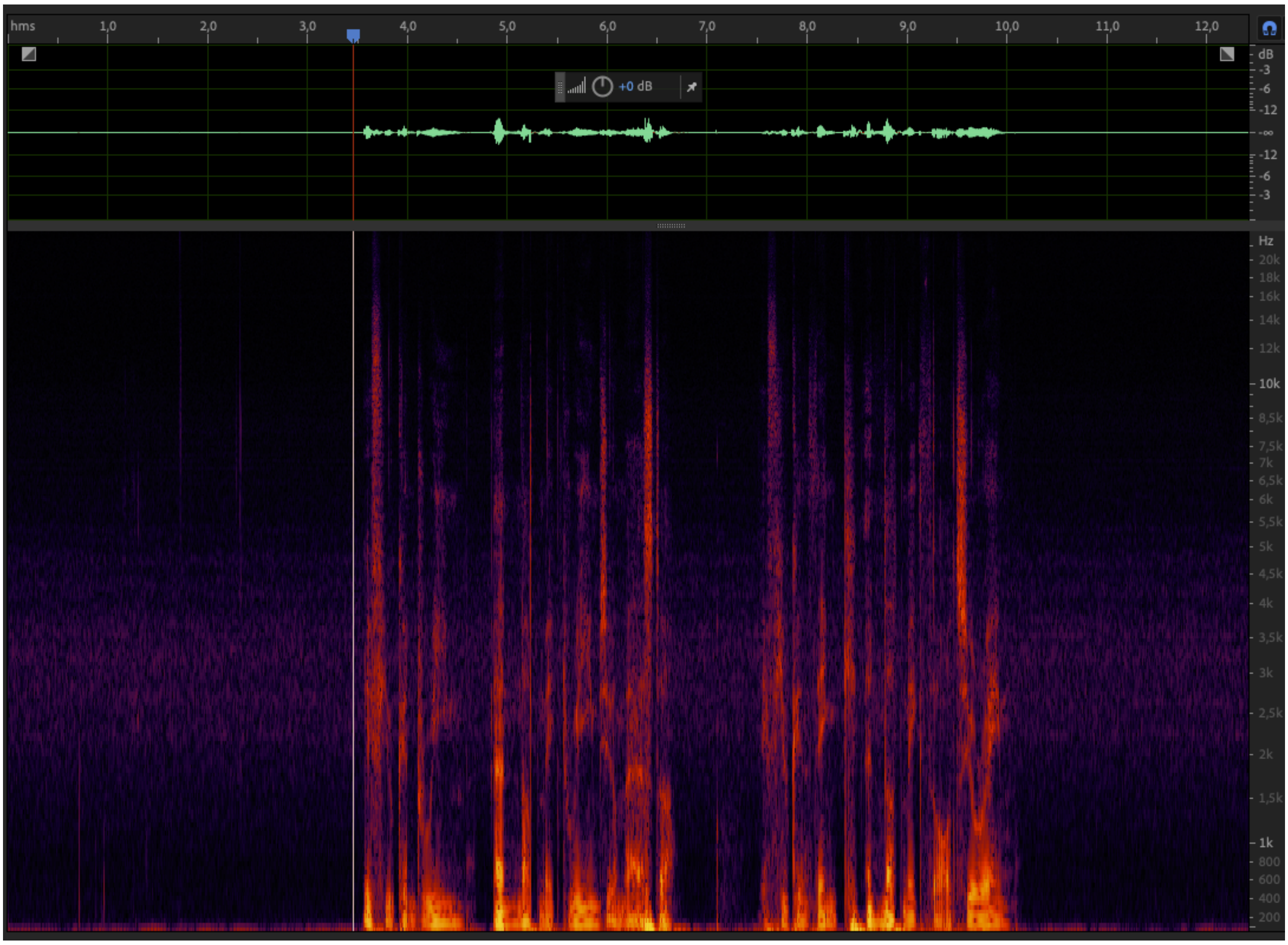


Figure 1: Original Recording

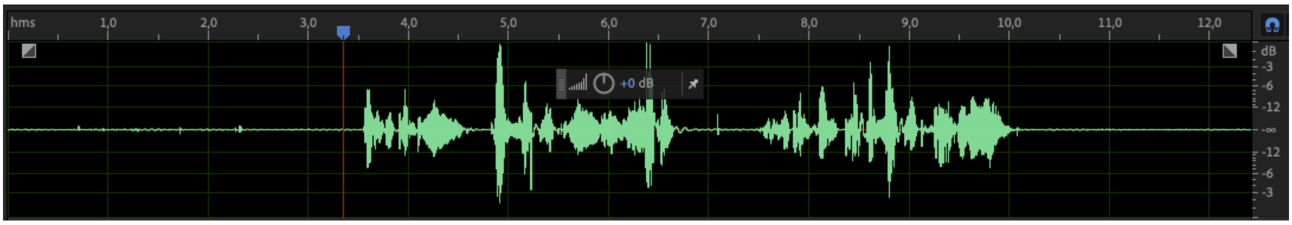


Figure 2: Normalized Recording

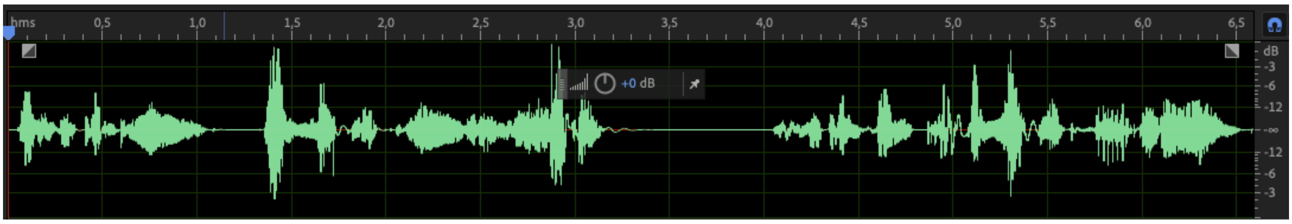


Figure 3: Trimmed Recording

App Audio Effects

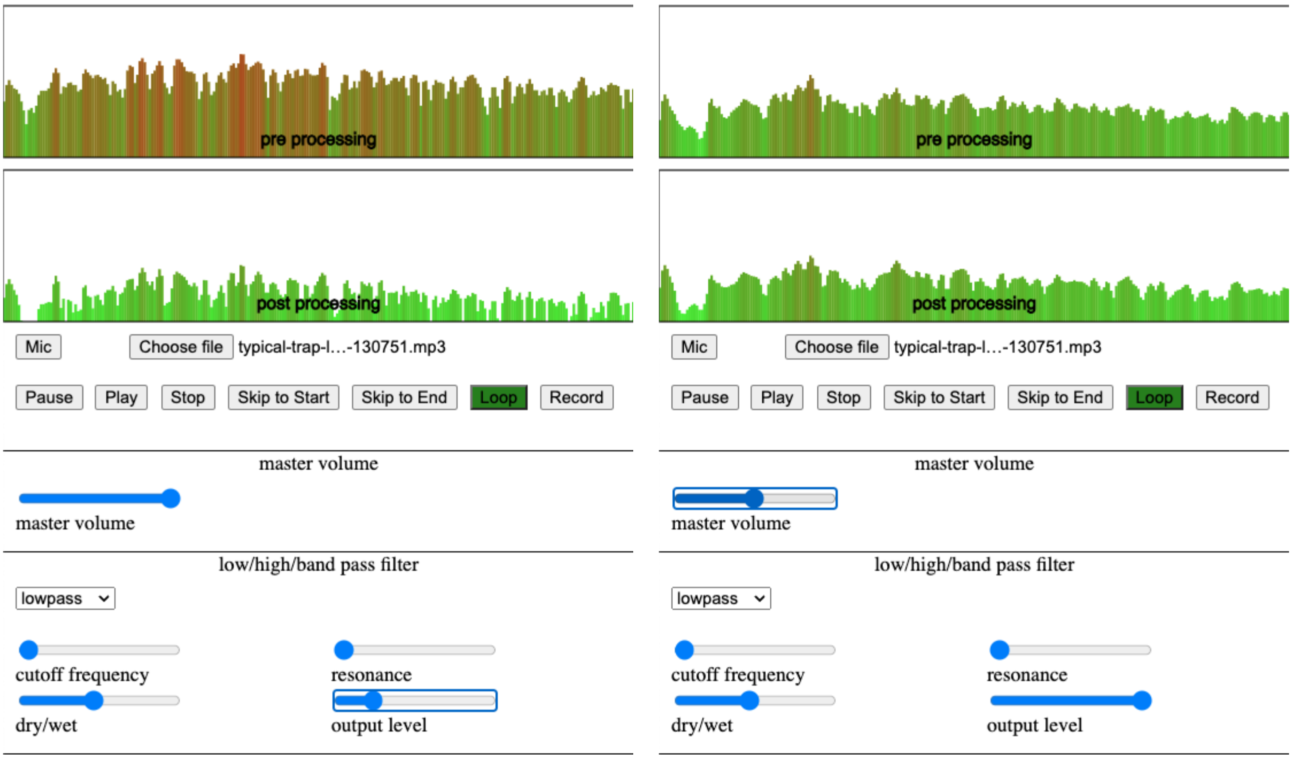
Link to Coursera Lab `https://hub.labs.coursera.org:443/connect/sharedxmwivuzz?forceRefresh=false&path=%2FvMBxTlyvUcM5DXb37Upsye6EhaJPW4z7oTLHiHdIrBJcLsAIkbpg8LFmKA6qdMB0%2F&isLabVersion=true`

All audio effects are part of the P5 Sound Effects implementation. The algorithmic implementation is already done so one only needs to look up the effect, what parameters and what the ranges of the parameters are. P5.Effect offers a chain function to tie everything together but this did not work as described (and expected) so the effects are chained manually one after another.

- **Low-Pass** and also **High-Pass** and **Band-Pass** filter is part of the P5.Filter. All three filters are allowing frequencies to a certain cutoff through unchanged and dampen all other frequencies. Low-Pass means everything above the cutoff gets attenuated, High-Pass mean everything below the cutoff and Band-Pass allows a frequency band through unchanged. Configured are the cutoff frequency and resonance.
- **Distortion** changes the original waveform algorithmicly and mangles the sound this way. Configured are the distortion amount and oversampling rate.
- **Delay** creates an echo effect that can be customized by defining the echo delay and how much feedback each echo loop has. An additional filter (i.e. Low-Pass filter) can also be applied ot modify the sound.
- **Compressor** is a tool to raise low amplitudes and lowers high amplitues creating a more even over all loudness. Configured are attack, knee, ratio, threshold and release.
- **Reverb** is again an echo effect that simulates as if the sound is played in a large physical space. Configured are the duration of the reverb, the decay rate of each echo and if the reverb is played forward or backward.

Low-Pass Filter and Master Volume Effect

The pre and post processing waveform display in the pictures above show that when the level is changed on a filter **only** the post processing signal is affected whereas the change of the master level applies to the pre processing signal (obviously also affecting the post processing signal).



(a) Filter Level (b) Master Level

Figure 4: Effects of Level Changes

Further Development

All three additional development ideas have been implemented.

- An HTML selection box allows to select between Low-Pass, High-Pass and Band-Pass filter. The type is set via the setType function of the P5.Filter object.
- To additional buttons have been added to the GUI. Allowing the user to use a microphone input or select an audio file from disk. On pressing the microphone button a p5.AudioIn object is created as audio source whereas on loading a file the loadSound function is used (like it's done for the default sound in the preload function).
- A delay filter has been added an chained in between the distortion and the compression filter. Description of the filter and configuration is given above.

Exercise 2 : Famous DJ

Task 1 : Audio Analysis

For visualisation purposes I would suggest to my famous DJ client that it's best to choose audio features that show a broad range of values in the proposed sample. It's obviously possible to use any feature but if there is not enough variability whatever is animated will not do a lot. From analysing all three sounds with Meyda audio features the first two samples showed considerable variation in the mentioned features. The third signal did not display as much range. I would have expected the *perceptualSharpness* to show interesting values but it didn't. I would have also been interested in *spectralFlux* and *spectralKurtosis*. However the first one only raised an exception when used (so was completely unusable) and the second did not correspond to the documented values which should have been in the range of 0 and 1 (but the values are all over the place). An additional factor for choosing certain features is values that are per default in the range of 0 to 255 (or at least in a pre defined value range) which makes pre processing the sound unnecessary and therefore makes it (easier) for real time animation.

sound	audio feature	justification
1	energy	A loudness of the signal indicator that (using defaults) in the range of 0 to 255.
	spectralCentroid	The brightness of a sound (or center of "gravity"). Also (per default) in the range of 0 to 255.
	spectralSpread	How noisy (frequencies all over the place) or pitched (narrow band of frequencies) a signal is. Again in the range of 0 to 255.
2	energy	Like above this features showed a good enough value range for animation.
	spectralCentroid	Same reasoning applies here.
	spectralSpread	And here. The Kurtosis values were interestingly wide spread but since the documentation was not accurate I didn't consider this (without doing more research - which there was not enough time for).
3	loudness (total)	The perceived loudness results in a better value spread in contrast to energy. (Not a fixed value range, pre processing is helpful).
	spectralCentroid	Usable value variations in the brightness of the sample.
	amplitudeSpectrum	Choosing a specific range of frequency bins results in an appropriate value range for animation.

Task 2 : Audio Feature Mapping

Link to Coursera Lab `https://hub.labs.coursera.org:443/connect/shareddbmbnsgy?forceRefresh=false&path=%2FvMBxTlyvUcM5DXb37Upsye6EhaJPW4z7oTLHiHdIrBJcLsAIkbpg8LFmKA6qdMB0%2F&isLabVersion=true`

All Meyda features used have been chosen because they provide a broad enough spectrum of values over the duration of the audio input. Meyda values have not been used 1:1 but scaled and shifted as necessary to build a visually pleasing application run.

The following Meyda features have been used for the visualisation:

- *energy* : initial size and alpha value of the drawn square; chosen because the visual size correlates nicely with the energy and it's also a nice feature to inverse the energy for the alpha value
- *spectralKurtosis* : even though the Meyda documentation is plain wrong for this feature experiments revealed a usable range that has been used for the green color value
- *spectralCentroid* : used for the red color value (and the inversion for blue)
- *spectralSpread* : used for the number of squares and the initial placement; correlates nicely with how far squares are from the center of the canvas

The attempt to use the amplitudeSpectrum feature of Meyda failed. Compared to the p5 FFT analysis it didn't produce usable values.

Further Development

Speech recognition using p5.Speech has been fully implemented in continuous mode making it possible to switch background colors and the shapes drawn. Please see the video for a demonstration.

Exercise 3 : Audio Steganography

Link to Coursera Lab https://hub.labs.coursera.org:443/connect/sharedaluwhchx?forceRefresh=false&isLabVersioning=true

Screenshots of the Jupyter Lab (in case the Lab link doesn't work properly):

Extracting a secrete message from an audio file

- load the file
- create a byte array of the audio data
- extract the least significant bit of each byte
- loop over the extracted bit array, converting every 8 bits into a byte and this into a character assembling them into a string
- cut the padding characters off
- display the message

Please see the comments in the code for more details what each line does.

```python
import wave

# read the audio data
song = wave.open('Ex3_sound1.wav', mode='rb')

# convert the audio data into a byte array
frames = bytearray(song.readframes(song.getnframes()))

# extract the message bits (LSB)
extracted = [ frames[i] & 1 for i in range(len(frames)) ]

# convert byte array back to string
string = "".join(chr(int(''.join(map(str,extracted[i:i+8])),2)) for i in range(0,len(extracted),8))

# cut off at the filler characters
decoded = string.split('###')[0]

# output the hidden message
print('Decoded message :', decoded)
```

```
Decoded message : Mom I will not be home for supper
```

Figure 5: Exercise 3.1

Finding an AM shifted message ¶

- looping over all audio files
- plotting the spectrum
- looking for peaks in the ultrasonic area (~20kHz)

Please see the comments in the code for more details what each line does.

```
from thinkdsp import read_wave, decorate, legend

# audio files to check for hidden messages
files = ['Ex3_sound2.wav', 'Ex3_sound3.wav', 'Ex3_sound4.wav']

# loop over all files
for file in files:
    # read the audio file
    wave = read_wave(file)

    # plot the audio spectrum
    spectrum = wave.make_spectrum()
    spectrum.plot(label=file)
    decorate(xlabel='Frequency (Hz)', ylabel='Amplitude', legend=True)
```

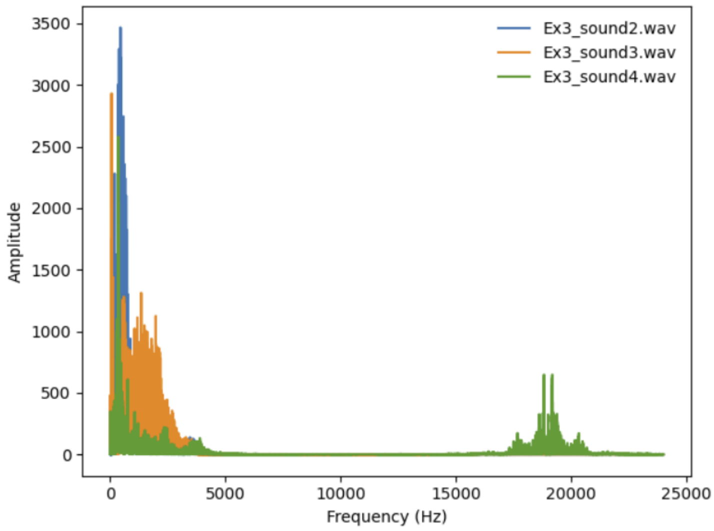


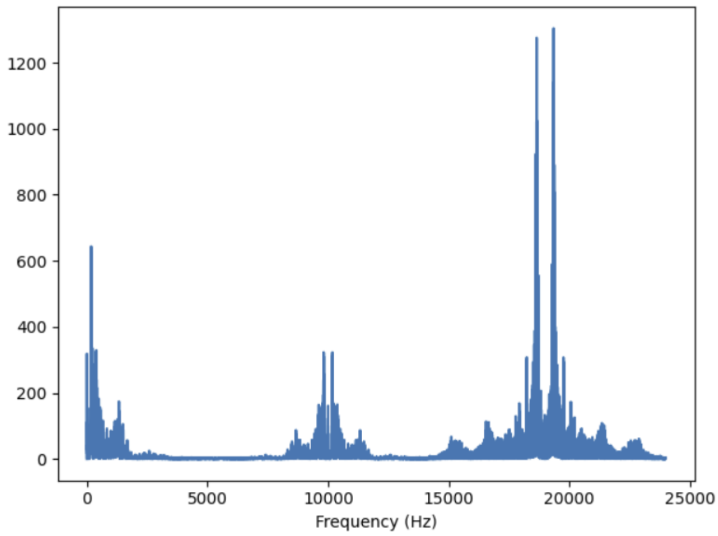
Figure 6: Exercise 3.2

Extracting the message

- create a carrier wave with a frequency that lies between the two halfes of the shifted frequencies
- modulate the original signal with the carrier wave
- plot the output (to see if the shift was accurate)
- listen to the output

```
from thinkdsp import CosSignal

# read the audio data
wave = read_wave('Ex3_sound4.wav')

# create a carrier signal matching the lenght and framerate of the loaded audio
carrier_sig = CosSignal(freq=19000)
carrier_wave = carrier_sig.make_wave(duration=wave.duration, framerate=wave.framerate)

# reverse the shift by modulating the origianal wave with the carrier wave
demodulated = wave * carrier_wave

# plot the spectrum to visualise the shift
demodulated_spectrum = demodulated.make_spectrum()
demodulated_spectrum.plot()
decorate(xlabel='Frequency (Hz)')

# make it audible
demodulated.make_audio()
```

0:06 / 0:06



The four number secret code is *1891*.

Figure 7: Exercise 3.2

Hiding a secret message with LSB audio steganography

According to some reading I've done the problem seems to be that the replacement of the LSB in the audio data will result in an imbalance in the statistical distribution of integers in the audio data. There are different ways to combat this problem. One is to not only replace the LSB but to change the frame number by adding or subtracting 1 if necessary so that the LSB of that frame matches the one of the secret message bit. This of course does not only change the LSB but can lead to many more bit changes per frame. Yet the statistical distribution of numbers is more 'natural' and the change in audio is not worse than replacing the LSB. A slight static noise as if it is an old recording is added this way.

Please see the comments in the code for more details what each line does.

Import the necessary Python libraries

```
import wave
import random
```

The following method is for storing the secred message. The method takes an input audio file name which is read and used as 'cover' for the secret 'message' given as parameter three. The output is written to a file name provided as the second parameter.

```
# store the secret 'message' in 'infile' and write it to 'outfile'
def store_secret(infile, outfile, message):
    # load the song
    audio_in = wave.open(infile, mode='rb')
    # convert audio to byte array
    frames = bytearray(audio_in.readframes(audio_in.getnframes()))

    # append filler characters to the string
    message = message + int((len(frames) - (len(message)*8*8))/8) * '#'

    # convert message string to bit array
    message_bits = []
    for l in message:
        message_bits += [ (ord(l)>>i)&1 for i in range(8) ]

    # loop over all message_bits
    # in- or decreasing the frame if necessary so the LSB fits the message bit
    for i in range(len(message_bits)):
        # check if the LSB of the frame needs to be modified
        if frames[i]%2 != message_bits[i]:
            # if the frame int is already the maximum decrease by 1
            if frames[i] == 255: s = -1
            # if the frame int is the minimum increase by 1
            elif frames[i] == 0: s = +1
            # otherwise it does not matter if it's increased or decreased
            else: s = random.choice([-1, 1])
            # apply the modification to the frame
            frames[i] += s

    # write modified frames to file
    audio_out = wave.open(outfile, mode='wb')
    audio_out.setparams(audio_in.getparams())
    audio_out.writeframes(bytes(frames))

    # close in and output file
    audio_in.close()
    audio_out.close()
```

Figure 8: Exercise 3.3

The next method takes only on parameter which is the file name to be read. The secret message is then extracted from this file.

```
def retrieve_secret(infile):
    # load the song
    audio_in = wave.open(infile, mode='rb')

    # convert audio to byte array
    frames = bytearray(audio_in.readframes(audio_in.getnframes()))

    # retrieve the message bits from each frame
    message_bits = [ int(f)%2 for f in frames ]

    # convert the bits to bytes
    message_bytes = []
    value = 0
    for i in range(len(message_bits)):
        if i != 0 and i%8 == 0:
            message_bytes.append(value)
            value = 0
        value |= message_bits[i] << i%8

    # convert the bytes to a character string
    message = ''.join([chr(l) for l in message_bytes])

    # cut off the padding characters and return the message
    return message.split('###')[0]
```

Testing the two methods:

- setting a secret message
- use method /store_secret/ to embed the message in the input audio and store it in a new file
- use method /retrieve_secret/ to get the message out of the given audio file
- compare the input and output message

```
# the secret message
msg_in = 'Father Christmas does not exist'

# embed the messsage in the given audio file and store it in a new file
store_secret('Ex3_sound5.wav', 'Ex3_sound5_secret.wav', msg_in)

# read the new file and retieve the message
msg_out = retrieve_secret('Ex3_sound5_secret.wav')

# show the input and output message
print('Message going in  :', msg_in)
print('Message coming out:', msg_out)

# compare the message and congratulate yourself
if msg_in == msg_out:
    print('Successfully stored and retrieved a secret message via LSB steganography.')
```

```
Message going in  : Father Christmas does not exist
Message coming out: Father Christmas does not exist
Successfully stored and retrieved a secret message via LSB steganography.
```

Figure 9: Exercise 3.3

Exercise 4 : Speech Recognition

Initial Problems

After some initial tests the first thing I would recommend to my client is the use of an ASR that is actively developed or at least properly maintained. It seems that DeepSpeech is no longer maintained and therefore can not be installed on a Mac (note: it's not an M1 Mac) or other system with current Python version (tested on Fedora 37).

```
(venv) hawk@BD-MB3 exercise 4 % pip install deepspeech
ERROR: Could not find a version that satisfies the requirement deepspeech (from versions: none)
ERROR: No matching distribution found for deepspeech
```

Even though the installation of DeepSpeech in the Coursera lab environment worked. The attempt at installing librosa for further analysis of the audio files failed there and put an end to this endeavour.

```
import sys
!{sys.executable} -m pip install deepspeech librosa
...
Installing collected packages: numpy, llvmlite, appdirs, typing-extensions, soxr, soundfile, pooch, numba, lazy-loader, audiorea
  Attempting uninstall: numpy
```

```
    Found existing installation: numpy 1.18.4
    Uninstalling numpy-1.18.4:
      Successfully uninstalled numpy-1.18.4
  Attempting uninstall: llvmlite
    Found existing installation: llvmlite 0.31.0
ERROR: Cannot uninstall 'llvmlite'. It is a distutils installed project and thus we cannot accurately determine which files bel
```

(Further) Development

Since every real life client is more interested in a running system than anything else for their money I did some more research for ASR systems and found Whisper. It is an open source neural net that's capable of multi language transcription (and much more) that is easily integrated in a python application. The models can be stored and the whole system run offline.

To evaluate different systems I've developed *ava_asr_test.py*. This program takes a JSON configuration file which lists languages, files and the correct transcription and tests the following ASR systems calculating the word error rate:

- Whisper - base model
- Whisper - tiny model
- Whisper - large-v2 model
- PocketSphinx
- Vosk
- SpeechRecognition - Google Cloud (for testing only!)

Some notes: Whisper has a variation of models that significantly differ in performance and of course size. PocketSphinx is barely kept alive and has had some changes done that made the use of language models other than English (and Italian) impossible. But due to the (very) poor performance of PocketSphinx that is not an issue as it would not even be considered in real life for this application. SpeechRecognition is there for comparison only as it uses a Google Cloud service (for testing ONLY) for the transcription (which only supports English).

This is an example of the JSON configuration:

```
{
    "en" : [
        {
            "file" : "samples/checkin.wav",
            "transcription" : "Where is the check-in desk?"
        },
        {
            "file" : "samples/parents.wav",
            "transcription" : "I have lost my parents."
        }
    ],
    "it" : [
        {
            "file" : "samples/checkin_it.wav",
            "transcription" : "Dove e' il bancone?"
        },
        {
            "file" : "samples/parents_it.wav",
            "transcription" : "Ho perso i miei genitori."
        },
    ...
}
```

Initially I experimented (*noise_reduction.py*) with noise reduction using noisereduce. However as it turns out it did not make a difference with the modern implementations like Whisper, Vosk and Google but it did add considerable calculation time.

The results of the ASR system evaluation and word error rate are summarized in the following table (% not scaled):

Lang	File	Whisper base	Whisper tiny	Whisper large	Vosk	Google	PocketSphinx
en	checkin.wav	0	0	0	1	0	0
en	parents.wav	0	0	0	0	0	1
en	suitcase.wav	0	0	0	0	0	1
en	what_time.wav	0	0	0	0	0	0
en	where.wav	0	0	0	0	0	0
en	homer1.wav	0.43	0.43	0.23	1	1	1
en	homer2.wav	0.44	0.56	0.11	1	1	1
it	checkin_it.wav	0.5	0.25	0.25	0	-	-
it	parents_it.wav	0.2	0.4	0	0	-	-
it	suitcase_it.wav	0.14	0.29	0	0	-	-
it	what_time_it.wav	0.43	0.86	0.14	0	-	-
it	where_it.wav	0.29	0.57	0	0	-	-
es	checkin_es.wav	0	0.5	0	0	-	-
es	parents_es.wav	0	0	0	0	-	-
es	suitcase_es.wav	0.17	0.33	0	0	-	-
es	what_time_es.wav	0.67	1.0	0	0	-	-
es	where_es.wav	0.43	0.43	0.29	0	-	-

Considering that Whisper is a state of the art system and Vosk is not (any more), Vosk (with the largest models available) did an excellent job. However it's noteworthy that each Vosk run took longer than a Whisper transcription and the WER calculation does not include deletion or addition of words which happened with Vosk transcriptions. Summing up I would suggest using Whisper with the large-v2 model.

The end product is a Python application that uses Whisper and can be run with command line parameters telling it which file to read, what language to use (optional) and what the correct transcription would be (optional for WER calculation).

```
import argparse
import whisper
from jiwer import wer
import os
import sys

# whisper defaults
# default language model
model = 'base'
# default language model directory
model_dir = '/Users/hawk/Downloads/whisper_models'

# setup the command line argument parser
parser = argparse.ArgumentParser(
    prog='Airport Virtual Assistant',
    description='Automated speech recognition system to help airport customers.',
)

# define command line options
parser.add_argument('-l', '--language', default=None, dest='lang')
parser.add_argument('-f', '--file', required=True, dest='filename')
parser.add_argument('-m', '--message', default=None, dest='message')

# parse given arguments
args = parser.parse_args()
#print(args.lang, args.filename, args.message)

# check if the file exists
if not os.path.exists(args.filename):
    print('The given file does not exist.')
    sys.exit(1)
```

```
# initialise the ASR
whisp = whisper.load_model(name=model, download_root=model_dir)

# transcribe
result = whisp.transcribe(audio=args.filename, language=args.lang)

# print the result
print('Transcription result:', result['text'])

# verification of the transcription result if a message is given
if args.message != None:
    print('The transciption should be:', args.message)

    # calculating the word error rate
    wer_whisper = round(wer(args.message.lower(), result['text'].strip().lower()), 2)
    print('WER', wer_whisper)
```