Final Project for MSP ELEN E4896
(awk2123) Anish Thunderbolt King

Introduction
This project is based on the following two iPython notebooks provided by the class:
https://github.com/dpwe/elene4896/blob/master/prac10/e4896_prac10_chords.ipynb
https://github.com/dpwe/elene4896/blob/master/prac11/e4896_coversongs.ipynb

This project uses a bank of reference video and audio clips of me counting from one to ten. Then, with a bunch of input audio clips of me also counting from one to ten (different than the reference audio) I try to match the input audio to the reference audio and display the video that corresponds to that audio. For a given sequence of input audio files, I can output (not too accurately…) the video for that number that I'm saying.

In general terms...a short project description:
- use a reference video + audio of someone talking.
- take input audio and align it to the ref audio
- output the video that belongs to the ref audio at the matching point.

Coming up with the project idea (see Appendix A for further details)
Here was an initial implementation plan

How to implement the computational lip movement to sound distance approximation application for visualizing music (songs…)

Two steps to program:

1. Given a track, figure out what the lyrics are. There are two ways to do this:
    a. Speech recognition software (I could maybe find an out-of-the-box solution, or I could write my own). I want it to output a sequence of phonemes, and I will need to represent the duration of these phonemes (possibly by fixing phonemes to last a certain amount of time and then repeating them when someone holds out a syllable), and I'll also need to represent silences in the vocals (e.g. by a "null" phoneme).
    b. Look them up on the internet. More specifically, search the track in a database to figure out what the name of it is; then, look up the song name on a website or database for song lyrics, make sure my result matches the correct artist and album, and then use the lyrics in the result. This method is much more likely to give me the correct lyrics (speech recognition is hard, especially with music); however, I do need to figure out how to match these lyrics to the rhythm in which they are sung. For this, I would suggest translating the lyrics into phonemes and then translating those phonemes into sounds. Then, I can run through the original

track and pair up the sounds of the phonemes to the lyrics on the track, recording how long each phoneme lasts and where there are silences. With this information, I can create a sequence of phonemes in a similar format to that output by method “a” above.

(Another issue with this method is parsing the lyrics. For instance, many lyric sources may have brackets to identify which vocal part speaks a line. To work this out, I mainly just need to find a lyrics source with a consistent format and tell my program how to read this format.)

2. Given a sequence of phonemes representing lyrics, make an animation of lips saying these words. The most straightforward way to do this is to record Meena (or me…) saying each phoneme in the English language. (I may need to have her say each of them with an “a” afterwards, e.g. “ma” instead of just “m.” If so, I’ll probably have to cut out the part of the video where her mouth changes from an “m” shape to an “a” shape. So, maybe pick a vowel whose mouth shape is very different from that of most consonants -- say, “o,” for instance.) Then, I can just tell the lips to play the short video clips corresponding to each phoneme one by one.

   This might give me something that looks quite natural. If so, that’s awesome! If not, then there are a few ways I can improve upon it:

   a. Use some kind of animation software to interpolate motion between each phoneme position. For this, I may want to cut my phoneme videos down to single frames, so that I aren’t switching back and forth between video clips and the interpolated animation, which might be hard to synchronize with the clips (I’m not totally sure if this will be an issue or not). I don’t really know anything about animation, so I’d have to do some research to figure out how this is done, but I’m pretty sure it shouldn’t be too hard a thing to do.
   b. Instead of having video clips for each phoneme, have clips for each pair of phonemes. I could also have clips for many common words, so that those will all look smooth. If the interpolation between phonemes (“a” above) doesn’t make things look smooth, then this probably should.

<u>This was the final implementation plan</u>

Prof. Ellis had this to say regarding implementing this project:
“The process I used for reading and writing videos frame by frame by having Python communicate directly with external ffmpeg processes is described at:
http://zulko.github.io/blog/2013/09/27/read-and-write-video-frames-in-python-using-ffmpeg/

My thoughts for your problem are:

- Start off with reference video + audio, ideally as aligned as possible, so jumps between different places will be OK.
- To generate an "edit" from the reference video that aligns to a new sound input:
  - Take spectrograms of input audio and reference audio
  - for each 40 ms block of the input audio (say, for 25 fps video)
    - take a patch of the input spectrogram, maybe 100 ms centered on the block
    - search through the reference spectrogram for the "closest" block. Basically, you have, say, NxM patches of the spectrograms. You can calculate a distance between them several ways, cosine distance has the nice property of being invariant to scaling (gain). So if a.shape == b.shape == (n, m), you can calculate a distance (where np is numpy)

      distance = np.sum(a * b) / np.sqrt(np.sum(a*a) * np.sum(b*b))
    - emit the reference video frame corresponding to the smallest-distance spectrogram patch.

You could do more to promote sequential blocks of video frames, but this would be an interesting start."

Here is the implementation plan I came up with:
Phase 1:
1. record fixed-length audio/video files (count to ten for example, say the alphabet, etc.)
1a. n pairs, (X, Y) = {(x1,y1), ... (xn,yn)}
X = audio ref
Y = video ref
2. get specgram / mfcc / or other features chroma for all of X.

Phase 2:
1. get input waveform and split it up based on the same vocabulary recorded previously. if i say the alphabet for ref, say the alphabet with same time constraints on each clip for the input audio.
1a. m files, z1, ... zm input audio files.
2. get features for all m clips

Phase 3:
1. get distance (z1-m, x1-n) using

```
def best_chroma_rotations(chroma_A, chroma_B):
    """Return a (chroma_A.shape[0], chroma_B.shape[1]) array of chroma rotations giving
per-cell greatest correlation."""
    num_frames_A, num_chroma = chroma_A.shape
    num_frames_B, num_chroma_B = chroma_B.shape
    rotated_chroma_B = np.copy(chroma_B)
    assert num_chroma == num_chroma_B
    similarities = np.zeros((num_chroma, num_frames_A, num_frames_B))
    rotate_chroma_indices = np.hstack([np.arange(1, num_chroma), 0])
    for i in xrange(num_chroma):
        similarities[i] = sklearn.metrics.pairwise.pairwise_distances(chroma_A, rotated_chroma_B,
metric='cosine')
        # Rotate chroma of B.
        rotated_chroma_B = rotated_chroma_B[:, rotate_chroma_indices]
    return np.argmin(similarities, axis=0)
```

you can change cosine to another dist measure.

L2 is a distance measurement, cosine, dtw, itakura
2. you end up with a distance matrix with n rows and m columns:

```
n     nxm
n-1
.
.
.
0 1 ... m
```

3. according to this distance matrix, find the most likely n for each m.
- use DTW path. (already implemented.)

4. find the sequence of audio file matches, and concatenate and play the matching video clips for those audio files. this means the video is playing the original reference audio sound, and not the input audio files. a next development step would be to rip the input audio and set it to the concatenated video clips so that the referecne video is playing to the input sound and not the original audio.

What I implemented

<u>Files included</u>
In the directory final_project_thunderbolt, which is released when you uncompress the .zip file of the same name, contains several files and subdirectories:

```
dyn-160-39-222-190:final_project_thunderbolt the_goat$ ls -la
total 1232
drwxr-xr-x  13 the_goat  staff     442 May  5 23:45 .
drwx------+ 14 the_goat  staff     476 May  5 23:44 ..
-rw-r--r--@  1 the_goat  staff    6148 May  5 23:46 .DS_Store
-rw-r--r--   1 the_goat  staff  100283 May  5 23:40 anish.webm
drwxr-xr-x  22 the_goat  staff     748 May  5 19:36 aud
```

```
-rw-r--r--   1 the_goat  staff    8068 May  5 01:04 beat_sync_chroma.py
drwxr-xr-x  16 the_goat  staff     544 May  5 19:39 beatchromftrs
-rw-r--r--   1 the_goat  staff  190868 May  5 23:30 final.webm
-rw-r--r--   1 the_goat  staff  141306 May  5 23:45 hello.webm
-rw-r--r--   1 the_goat  staff   99827 May  5 23:43 hi.webm
-rw-r--r--   1 the_goat  staff   69052 May  5 23:44 lips.ipynb
-rw-r--r--@  1 the_goat  staff      47 May  5 19:39 list.txt
-rw-r--r--@  1 the_goat  staff  230246 May  6 00:01 thunderbolt.finalprojectreportelene4896.pdf
drwxr-xr-x  12 the_goat  staff     408 May  5 18:47 vid
```

The various .webm files are various tested outputs.

Note that I recorded and edited all these audio and video files to be 1.2 seconds long exactly:

```
dyn-160-39-222-190:final_project_thunderbolt the_goat$ cd aud/
dyn-160-39-222-190:aud the_goat$ ls
1.mp3       4.mp3       8.mp3       four.mp3    six.mp3
10.mp3      5.mp3       9.mp3       nine.mp3    ten.mp3
2.mp3       6.mp3       eight.mp3   one.mp3     three.mp3
3.mp3       7.mp3       five.mp3    seven.mp3   two.mp3
```

```
dyn-160-39-222-190:final_project_thunderbolt the_goat$ cd vid
dyn-160-39-222-190:vid the_goat$ ls
1.mp4  10.mp4 2.mp4 3.mp4  4.mp4  5.mp4  6.mp4  7.mp4  8.mp4  9.mp4
```

```
dyn-160-39-222-190:final_project_thunderbolt the_goat$ cd beatchromftrs/
dyn-160-39-222-190:beatchromftrs the_goat$ ls
1.pkl       4.pkl       eight.pkl   seven.pkl   two.pkl
10.pkl      5.pkl       four.pkl          six.pkl
2.pkl       7.pkl       one.pkl     three.pkl
```

To get the beatchromftrs directory (which contains the .pkl pickle files that are needed to properly deal with the .mp3 input / reference audio files please type into terminal the following:

```
$ python beat_sync_chroma.py -i list.txt -o beatchromftrs -w aud
```

<u>Results</u>
Currently the code will take in the input audio files via a text file listing (it does the same with the reference audio files) although the pickle tool doesn't seem to work properly, in that not all the numbers that I say are converted properly into the pickle file serialization. The files that worked are:

1.pkl

```
2.pkl
4.pkl
5.pkl
7.pkl
10.pkl
eight.pkl
four.pkl
one.pkl
seven.pkl
six.pkl
three.pkl
two.pkl
```

Note: to see the output of block In[41] in the code please scroll through the output box fully

I tested the following, for example. The output .webm file for this is called final.webm. The other webm files are just to show the video outputs more than just two's and one's.

In the code (block In[62]):

```
ids_B = ['1', '1', '7', '7', '2', '2', '4', '4']
ids_A = ['one', 'one', 'seven', 'seven', 'two', 'two', 'four', 'four']
```

The results are then:

```
input: one
match: 2
input: one
match: 2
input: seven
match: 2
input: seven
match: 2
input: two
match: 2
input: two
match: 2
input: four
match: 1
input: four
match: 1
['2', '2', '2', '2', '2', '2', '1', '1']
```

Again, final.webm lets you listen to this.
Maybe I could have recorded better, for example, instead of recording using my computer's microphone, though some effort was made to record in a quiet environment (read empty apartment in New York City…).

You may watch the output .webm files which gives the video + reference audio that the system thinks matches best with the input audio chroma.

Conclusion

I am quite satisfied with this particular implementation of my original idea because in the process, I learnt a lot about how I would make this system more sophisticated in the future. Starting out, I had no idea how to begin implementing this, so I guess I feel I was quite successful in my goal (that I've had since mid-last semester) to get a part of this project working. I am deeply passionate about this project and wish to develop it further in the coming months, perhaps by using deep learning neural networks to draw the best path information once we have the distance information. Also, I wish to implement this via frames / blocks rather than through separate clips, which I think makes this system a lot more robust. Ultimately, I wish to have a system that is accurate 99.99% of the time with a justifiable computational complexity that musicians / artists / HAL can use to match lips to sound based entirely on matching an input audio to a reference audio + video set. Another aspect of this project that makes me happy is that (I think) this project truly embodies music signal processing because where many have approached this problem of matching lips to words using computer vision / image signal processing techniques, I always had the notion of doing this via playing with audio.

Acknowledgements

I would like to thanks Dan Ellis and Zhuo Chen for their guidance and support for this project. I would also like to thank Brad Garton, John Kymissis, and John Wright for their help along the way - from the conception of this idea to introducing me to the world of music signal processing hacking. I hope to do lots of this kind of stuff in the future and to be a python god.

Appendix A - Final Project Proposal

This document just lists the initial project scope + ideas that I wanted to implement (all closely related to what I actually implemented. This is interesting to keep in mind, given this is an engineering project. I like to look at the thought process involved when coming up with idea to seeing it through realization.

Final Project for MSP by Anish King — “LIPS”
Spring 2016 ELEN4896 with Prof. DPW Ellis

Premise

The premise is simple — build an interactive, intelligent visualizer for music / sound input. Music has evolved to a level that DJ sets can go on and on (and on) and just based on the music itself, it’s very difficult to leave a venue. The music is just that mesmerizing. As someone with a huge background in music (in terms of producing sounds as of late via computers and playing two different classical violin styles for the past 17 years) I am extremely pleased with the modern advances made in sound synthesis and progressions. That said, I am also thoroughly disappointed with the poor visuals that accompany live music shows. The visuals are never intelligent, and sadly, seldom pleasing to look at. I want to build a very artistic / sensual visualizer for live music shows based on something concrete. For this, I have chosen the motion of human lips to demonstrate intelligent, artistic music visualization.

What I have so far

I have a simple max/msp patch that reads in a video on loop (of my lips opening and closing) as an array. The RGB values of the continually moving image are read in (line by line starting from the top left corner of the image) at a set rate and these are converted into midi values of pitch, amplitude, and velocity, for RGB respectively. The image saturation is then influenced by the ambient sound amplitude, and the contrast of the image is affected by the pitch of the sounds surrounding the image system.

Directions

Interestingly enough, my project thus far is an instrument that throws out a rather simple image. While artistic and interesting, this project is far from what I’m reaching for in terms of sophistication. Like I said before, I am aiming to build the best, most intelligent visual for each and every sound that might be played. Here are some various directions.

1. Pitch tracking — I want the lips to react exclusively to various frequency ranges. In line with this — I would include other triggers to other outputs, for example, have the lips open/close to the four on the floor kick drum frequency/amplitude, while a pair of eyes open and close to a high hat / woman’s vocal frequency range.
2. Large library of lip motions. I have a friend with very nice lips and she has volunteered to have her lip motions for a variety of vowel and consonant sounds to be recorded and catalogued (by me) accordingly. Otherwise, I can try to model lip motions, which might be a difficult problem, but there may be existing computationally inexpensive models.

3. There are other triggers other than pitch / amplitude tracking. I can exploit some of these other triggers for a more sophisticated system.
4. Say you're a DJ conducting a set. You have some sounds going. You input some text into a system and set it on a loop, triggering it when you want to. When you launch the text, the lips appear on a screen behind you, they mouth the words and the text comes out in a selected voice that you, the DJ, has chosen. The crowd goes wild.
5. Your music has lyrics in it, you're with your band, the Rolling Stones. You're performing at the Royal Albert Hall and as you sing your lyrics, the iconic lips in the background are pronouncing the exact words — no sound comes out, just a speech to text to appropriate physical mouth modeling. Really sensible lyrics to visualize music.

Addendum

Music is abstract, and instrumental music without emphasis on lyrics is even more abstract. Music has an inherent ability to make people feel primitive human emotions — Mark Rothko called these "tragedy, ecstasy, and doom". Rothko became a painter to capture via painting the power of music. I get it, music is abstract, and for too long has there been a wildly varying opinion of how to visualize abstract concepts and the most primitive of human emotions. I am bringing to the table a concrete, intelligent system with set parameters to accurately portray sounds in an aesthetically pleasing way.