Audio Fingerprinting for Timeline Reconstruction

Ramsey Natour, Tom Mayo-Smith, Rishi Khanna, Ferdinand Dowouo

Introduction

For events such as the Boston Bombing or September 11 attacks, forensics experts must construct a timeline of the event to support their investigation. This process is time consuming, especially in cases where there is much video footage available on public sites such as YouTube. The videos have different start and stop times and come from many different angles; however, they contain common audio signatures that can be used to match overlapping segments together. We have implemented an algorithm to reconstruct audio signals by merging sound recordings based on common "fingerprints." These fingerprints are computed using a variant of the algorithm behind the popular music identification service Shazam. Our results demonstrate that audio fingerprinting can be useful to create a cohesive a timeline of video footage for events.

Procedure

Given a set of videos of a particular event, possibly recorded by different devices, our algorithm first goes through a stage which involves processing the audio data to make it more easily comparable. Its important to keep in mind our algorithm only uses audio data to stitch together different overlapping videos and thus this is the only necessary data from needed from the video files. After this stage is done, time sensitive hashes of the the most important frequencies of power spectral density of each of these audio data are computed to allow these audio data to be compared with speed and minimized noise interference. A matching function is used to compute the most likely overlap of each combination of two audio files and the confidence that an overlap actually exists. To be clear, each audio file will likely have randomly matching hashes at different time offsets and thus a most likely overlap period can be computed; however, if the confidence this overlap exists in the first place is low then our algorithm assumes it was a pure coincidence the hashes matched. Lastly, an alignment function takes the predicted overlap periods and confidence indicators computed for each combination of audio files and attempts to stitch the original videos together.

Preprocessing

During our tests, data files were pulled from a collection of miscellaneous sound files as opposed to using the audio data from a collection of video files. For each sound file, the audio channels were combined to form a single channel and the single channel was down sampled to the lowest sampling frequency of the pulled audio files. Each file was randomly split into overlapping regions and then gaussian white noise with different variances was added to each of these segments. These split files were used as the input to the algorithm. The sound files had a frequency cutoff above which all values were not considered.

Hashing

Hashing was only performed on the peak values of the spectrogram of each audio segment in an attempt to limit the effect of the added noise. For each time step, the frequency is divided into 6 logarithmic bands representative of the manner with which humans perceive differences in sound (0-10 Hz) (10-20 Hz), (20-40 Hz), (40-80 Hz) (80-160 Hz) (160-512 Hz). The frequency with the highest energy from each band is recorded along with the mean of these frequencies. Recorded frequencies higher than the mean are kept and those which are lower are ignored. The figure below displays the final result of this process.

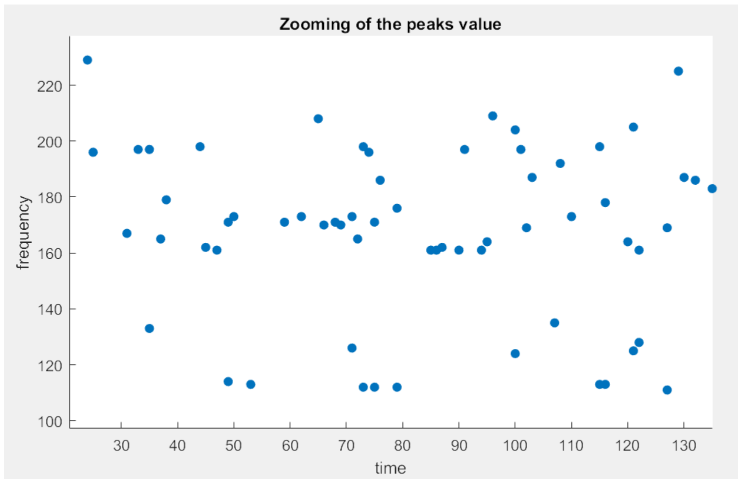


Figure 1: Most Important Frequencies of Spectrogram

The reasons behind the method used to generate the hashes is complex and will not be discussed in this paper; however, more information regarding this method can be found in the Shizam paper referenced in the final section. For the purposes of understanding the next sections it is worth noting the hashes for each peak frequency were generated using its own frequency value, the values of other peak frequencies surrounding the peak frequency and the time difference between these frequencies. The hash for each peak frequency was paired with an absolute time representing approximately where the peak frequency occurred in within the sound segment. This absolute time is referred to as an anchor point.

Computing The Offset

In this phase of our algorithm the offset between two audio files is attempted to be computed given the hashes of these files generated in the previous step. When a common hash is found between the hash sets of of the two files the anchor times corresponding to those hashes are subtracted and the result is recorded. Each of these results is a potential offset and thus the next step is to determine which one is most likely. Figure A is a graph of offsets vs the total number of times they were tallied. Because an offset of around 10 seconds is observed much more frequently than any other offset it is very likely the two sets of hashes being compared belonged to audio files offset by 10 seconds.

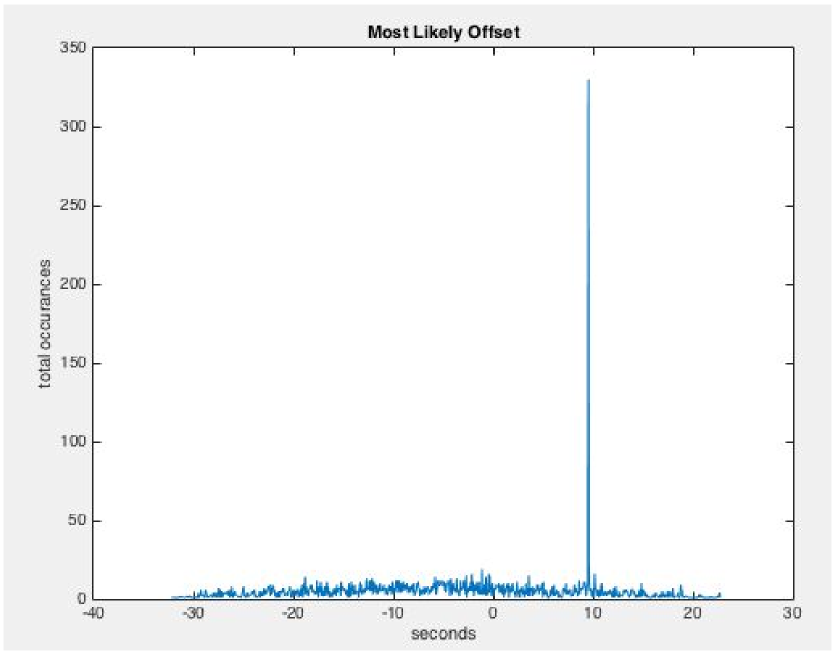


Figure 2: offsets frequencies for overlapping audio files

In contrast Figure B displays a set of offsets likely generated by two files with little to no overlap.

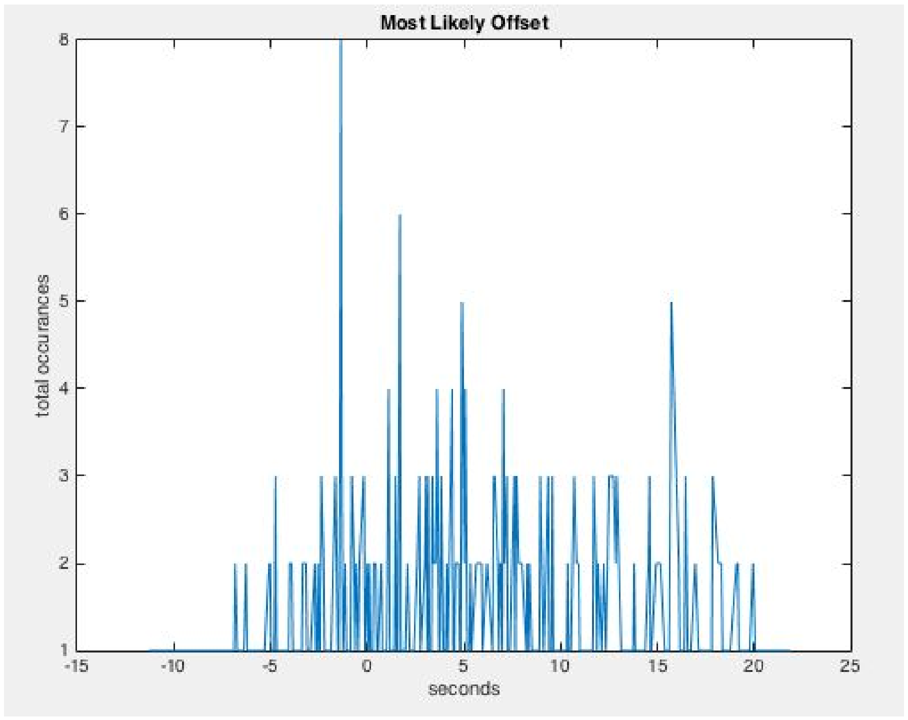


Figure 3: offsets frequencies for disjoint audio files

The most notable difference between these two graphs is the sharpness of their peeks. Thus it follows logically that a good way to measure the confidence that a particular maxim offset is a true offset as opposed an offset that was found via subtracting the anchor points of randomly matching hashes is to measure the sharpness of the peaks.

This is done in our algorithm by computing the peak-to-sidelobe ratio.

Dmax = maximum occurrences of a particular offset
u = mean number of occurrences of all offsets
sigma = standard deviation of occurrences of all offsets
PSR = (Dmax - u)/sigma

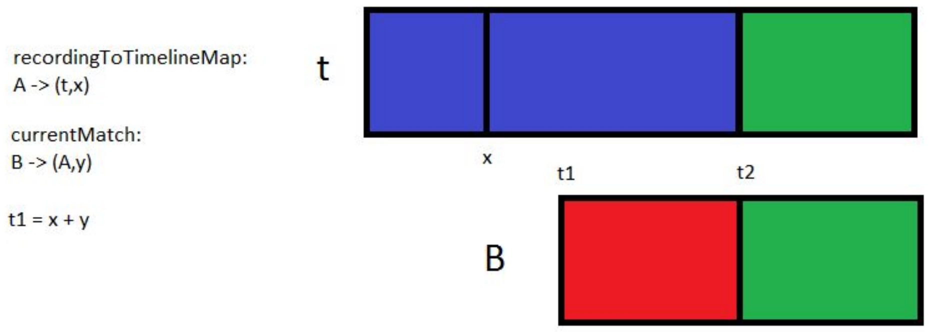
Alignment

Once each matching pair of recordings is assigned a match strength and best offset, our algorithm merges the recordings into timelines. We initialize the first timeline to be a recording from the first pair. We then iterate through the list of matches in order. For each match we check if the recordings are a part of a current timeline and case on the result:

1. If one is, we merge the other recording with that timeline.
2. If both are, we merge the two timelines if they are distinct. Merging timelines is identical to merging a recording with a timeline, except they are aligned based on each recording's offset.
3. If neither is, we create a new timeline out of the match pair and append it to the list of timelines.

The iteration ends when the match strengths fall below a threshold representing our tolerance for noise.

There are multiple ways to merge a recording with a timeline based on its alignment. For non-overlapping time intervals, we append or prepend the timeline with the truncated region of the recording. For overlapping time intervals [*t1*,*t2*], we define *f(t(t1:t2),r(t1:t2))* to compute the timeline values for that region as a function of the current values and the new recording values. The same holds from timeline-to-timeline merging.



In Figure 4, recording *B* matches to recording *A* at offset *y*. Since *A* is currently at offset *x* in timeline *t*, *B* is truncated from [*t2*,*length(b)*] and added to *t*. The blue region is the unaffected portion of the timeline, the red region is the truncated portion of *B*, and the green area is the truncated portion of *B* appended to *t*.

In our implementation, we first sort the match pairs by match strength and define *f(t(t1:t2),r(t1:t2)) = t(t1:t2)*. This leads to a greedy approach where we assume the best values for an overlapping region lie in the already constructed timeline. The intuition behind our approach is that recordings with a higher match strength were more likely to be measuring the true audio signal. We use two hashmaps in our implementation: one to lookup the timeline and index of each recording and another to lookup the list of recordings for a given timeline. These two hashmaps are sufficient to reconstruct video timelines and are returned along with the audio reconstructions.

Results

The data set used to test our results consisted of 18 unique audio files each broken up into three overlapping segments. The offsets required to be added to each of these three segments in order for them to be aligned was recorded before the segments were passed to our algorithm. Six overlapping segments were chosen at a time from two unique audio files and mixed together before being passed into our algorithm. After this our algorithm was tasked with reconstructing the two original audio files using the mixed segments. Each time our algorithm reconstructed a pair of audio files the six offsets added to the segments which composed these reconstructed audio files were compared with the prerecorded offsets. The 18*3 = 54 predicted offsets differed from the 54 prerecorded offsets on average by 0.0133 seconds. This shows the level of precision with which our algorithm is able to reconstruct audio.

Conclusion

In this project , we have used audio fingerprinting to reconstruct a timely cohesive footage of an even. The algorithm is based on, signal processing technique and a little bit of optimization. Front the spectrogram analysis of the audio recording, we found the peaks that are characteristic of the landmark signature. Then using a hashing function, we were able to match the audio and then find the offset between them and based on these offset, a timeline of the event could be reconstructed.