Time-based Alignment of Video Recordings: A Service and an Application

by

Zihao Yu

A Thesis submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

written under the direction of

Professor Ivan Marsic

Professor Mor Naaman

and approved by

________________________

________________________

________________________

________________________

New Brunswick, New Jersey

January, 2011
ABSTRACT OF THE THESIS

Time-based Alignment of Video Recordings: A Service and An Application

By Zihao Yu

Thesis Directors:
Professor Ivan Marsic
Professor Mor Naaman

Video content is uploaded and shared by users on video sharing websites in vast scale. It is estimated, for example, that every minute 24 hours of video is uploaded to YouTube. Many of the clips are captured at live events, for instance, a U2 concert in Giants Stadium. Increasingly, then, individuals attending the same events upload related content: there are over three hundred YouTube clips from the said concert. This overload makes relevant and interesting videos harder to find, and the event content harder to view and understand. An innovative approach to present the video content is thus necessary.

In this thesis, we propose a solution to tackle the problem above. We use the audio fingerprinting algorithm to find overlapping video clips within events, and time-align these overlapping videos based on their audio tracks. We developed a highly interactive video player that organizes and presents the time-aligned video content. The player integrates social data like view counts to help people create a better understanding of the event content, and improve the viewing experience and seeking of clips. We conducted user study sessions to understand user’s interaction and get user feedback for the video player.
The web is adopting open standard these days. The work in this thesis follows this trend by providing API services. We designed and built a set of standard API services to expose the underlying audio fingerprinting. In this way other developers can utilize our system remotely and programmatically. They can also build applications using time-align data.
Acknowledgement and Dedication

I would like to thank my advisor Professor Mor Naaman for giving me this opportunity to work on something interesting and practical, for his enthusiasm, his encouragement, his sound advice and great efforts during my research in the Social Media (SM) group at Rutgers University. I would like to thank my colleague Dr. Nicholas Diakopoulos, who gives me continuous inspiration and helps me despite all the trouble during my research at SM group. I am also grateful to my professors and friends at Department of Electrical and Computer Engineering for their emotional support and help, which made my study at Rutgers enjoyable and fruitful. I would like to thank the staff of School of Communication and Information and Department of Electrical and Computer Engineering for their timely and professional assistance and support, which provides the stable cornerstone for everything I achieved. Finally I wish to thank my family for their understanding, encouragement, and eternal love.
# Table of Contents

**ABSTRACT OF THESIS** .................................................................................................................. ii  
**Acknowledgement and Dedication** ............................................................................................... iv  
**Table of Contents** .......................................................................................................................... v  
**Introduction** ...................................................................................................................................... 1  
  1.1 Motivation and Problem Description ......................................................................................... 2  
  1.2 Contribution ............................................................................................................................... 4  
**Background and Related Work** ...................................................................................................... 5  
  2.1 Audio Fingerprinting ................................................................................................................. 5  
  2.2 Web Services API ....................................................................................................................... 7  
  2.3 Multi-perspective Player .............................................................................................................. 9  
**Alignment** ....................................................................................................................................... 13  
  3.1 Background: Audio Fingerprinting ......................................................................................... 13  
  3.2 Existing Code and Limitations .................................................................................................. 17  
  3.3 Contributions and Extension ...................................................................................................... 19  
  3.4 Evaluation .................................................................................................................................. 21  
    3.4.1 System Performance ......................................................................................................... 21  
    3.4.2 Alignment Result Evaluation ......................................................................................... 24  
**Web Services** ................................................................................................................................... 27  
  4.1 Design Goals and Framework ................................................................................................... 27  
  4.2 API ............................................................................................................................................. 27  
    4.2.1 Create A Group ................................................................................................................ 28  
    4.2.2 Add A Clip ....................................................................................................................... 29  
    4.2.3 Process A Group ............................................................................................................. 31  
    4.2.4 Get Overlap Data for A Clip: ....................................................................................... 33  
    4.2.5 Get A List of All Clips in A Group ............................................................................... 34  
    4.2.6 Delete A Group .............................................................................................................. 34  
  4.3 Use Case Scenario ...................................................................................................................... 35  
  4.4 Implementation Details ............................................................................................................. 36  
    4.4.1 YouTube Integration ...................................................................................................... 37  
    4.4.2 Task Queue and Workers ............................................................................................ 38  
    4.4.3 Matlab and Python Integration .................................................................................... 39
List of tables

Table 1: Hash table generated by audio fingerprinting algorithm ........................................... 14
Table 2: System resource consumption comparison .............................................................. 21
Table 3: Details of groups and their processing time ............................................................ 22
Table 4: Details of three test groups, including the ground truth ........................................ 24
Table 5: U2 Magnificent group test results ......................................................................... 25
Table 6: Mamma Mia group test results ............................................................................. 25
Table 7: Conan O’Brian group test results ......................................................................... 25
List of illustrations

Figure 1: Example search on YouTube ................................................................. 3
Figure 2: Matching landmarks layout ................................................................. 15
Figure 3: Time offsets between matches ........................................................... 15
Figure 4: Overview of group of clips ................................................................. 17
Figure 5: Functionality of Columbia Matlab library .......................................... 18
Figure 6: Actual curve with linear fitting ........................................................... 23
Figure 7: Threshold-Precision trend ................................................................. 26
Figure 8: Threshold-Recall trend .................................................................... 26
Figure 9: Invalid API key response example .................................................... 29
Figure 10: group_create API call response example ....................................... 29
Figure 11: Successful clip_add API call response example ............................ 30
Figure 12: Failed clip_add API call response example .................................... 31
Figure 13: Failed group_process API call response example 1 ....................... 32
Figure 14: Failed group_process API call response example 2 ....................... 32
Figure 15: Successful group_process API call response example .................... 32
Figure 16: clip_get_overlap API call response example .................................. 33
Figure 17: group_get_clips API call response example ................................... 34
Figure 18: group_delete API call response example ....................................... 35
Figure 19: Workflow of fetching and extracting video content ......................... 37
Figure 20: Relationship among different parts of our web service ................... 41
Figure 21: A draft of the video playback interface .......................................... 45
Figure 22: Player structure and working diagram .......................................... 49
Chapter 1

Introduction

Video recordings made by general public are increasingly prevalent and abundant. The use of portable video recording devices is the key to this phenomenon. Video content is uploaded and shared by users on video sharing websites in vast scale. It is estimated, for example, that every minute 24 hours of video is uploaded to YouTube. Many of the clips are captured at live events, for instance, a U2 concert in Giants Stadium. Increasingly, then, individuals attending the same events upload related content: there are over three hundred YouTube clips from the said concert. This overload makes relevant and interesting videos harder to find, and the event content harder to view and understand. An innovative approach to present the video content is thus necessary. In this thesis, we specifically focus on the large amount of video clips with overlapping audio.

There are several approaches when dealing with a large number of overlapping and redundant video clips from a single event. These include:

(i) restoring a three-dimensional environment of the scene by combining videos taken from different angles and positions [1];

(ii) extracting semantic meanings from the event, for instance, the hot topics people talk about during the show, or the key moments of the event [2];

(iii) presenting the users with different angles of views of a single event, which is usually offered by the sports broadcasting industry [3].

In this thesis, we are interested in the case where overlapping video recordings from multiple contributors are present and identified by audio fingerprinting algorithm. We then process these recordings to get the exact overlapping data information. We will do
some post-processing work to construct a unified and organized video playback interface, in which overlapping video clips are aligned appropriately and can be manipulated freely and interactively. We also integrate social data from online community into the playback interface so that users’ perception and navigation are better aided.

User study has shown that the video playback interface proposed in this paper is novel, interesting, and is particularly useful in some certain type of events, for instance, musical concerts, and live shows.

1.1 Motivation and Problem Description

Event-based management is becoming more and more popular on the Web, especially for musical events. Last.fm has an Events category, which lists all events that are happening or will happen in your neighborhood. YouTube also has an Events Near You section under Music category. It tells you the names of the people/bands that will perform, the venues, and date and time. Users can upload their videos and share their thoughts on a particular event, as well as search for content from the event, even years after it actually happened.

When users do search about a specific event or concert on video sharing websites, for instance, YouTube, they will inevitably get a long list of unfiltered, redundant results with duplicate and overlapping audio. In most cases it is because people scattered around the venue are using their recording devices to create independent records (videos) of the proceedings. Sometime the video recordings are taken by fixed cameras that are shooting the stage from different orientations. No matter what, currently, there is no good approach for users to relive the concert experience, without the trouble of going through all the search results.
For instance, Figure 1 below shows the result when you search YouTube for videos of U2 performing *No Line on the Horizon* in Dublin. The number of results returned easily tops 300. Moreover, many of them were taken from the same event, as the search term suggests. Some videos were taken far away from the stage, while some are really close to it. Thus a user may need to go through most of them if they want to get a complete view but that may be tedious and time consuming. Wouldn’t it be great if all these video clips are organized in a certain way?

Figure 1: Example search on YouTube.

Even though there exist some ways to organize video clips, we aim in this work to provide a complete solution, from collecting video into a group taken at the same event,
to processing the group and building an organized video playback interface. In addition, our playback integrates social data as well.

1.2 Contribution

The research of this time-based alignment service and application is an implementation and improvement of the system and interface proposed in [4]. Not only we actualize the time-aligned video playback concept; we also build a complete web application that serves from video group creation to representation. The key contributions of this work are thus:

(i) Adapt and use an audio fingerprinting algorithm to analyze video recordings and get exact time offset data between videos taken at the same event;

(ii) Build a web service to allow developers to utilize the underlying audio fingerprinting algorithm, and to create, process, and manage groups of videos from various events;

(iii) Design a novel and interactive video playback interface, which organizes and presents the overlapping video clips, and integrates social data.
Chapter 2

Background and Related Work

Our project addresses the nexus of three domains: video matching and synchronization by audio fingerprinting, open web services API’s, and an interactive multi-perspective video playback interface.

2.1 Audio Fingerprinting

Audio Fingerprinting, or Acoustic Fingerprinting, is a condensed digital summary, deterministically generated from an audio signal, which can be used to identify an audio sample or quickly locate similar items in an audio database. Generally, audio fingerprints are identifiers for audio tracks based on the audio data they contain.

In [5], Mueller et al. proposed that it is possible to consider audio matching problems, which is the one of key problems in our application, as an extensions of the audio identification problems. This kind of problems usually involves a short audio fragment produced by some unknown device. The goal is to clearly identify the original audio recording based on this short sample piece.

Nowadays some music identification services exist that provide phone-based systems for identifying popular music. For example, [6] introduces a music identification service, Shazam. When you walk on the street and hear a compelling song playing in one of the stores, you can record a sample of it using your phone. Shazam lets you to upload the clip to its server. The system will automatically find out the full name as well as related information like composer, year of the song, and return it to you. The system is also able to retrieve specific recordings matching the query you send to it, exactly the same version, in spite of audio quality difference. Services like this must be robust enough to tolerate
audio signal distortions caused by low-quality microphones, poor speakers, or background noises like people chatting and cars passing by.

In audio fingerprinting, acoustic recordings are characterized by local occurrences of some particular structures [4]. Two recordings could be identified if they originate from the same source quickly based on their structure. More details about the audio fingerprinting algorithm will be introduced in Section 3.1.

Most of these existing audio matching services concentrate on matching a short music excerpt against a source database and thus identifying it. On the other hand, our work matches a group of audio tracks extracted from video clips against each other, and determines which tracks belong together and further calculate their starting time offsets.

We build on the work from Columbia University that had published a Matlab library implementing a landmark-based audio fingerprinting system¹. The library as described in [7] is based on the idea behind Shazam, and is able to identify short music snippets recorded in noisy conditions. This library is capable of reading a series of audio files both locally and remotely, accepting a sample audio as the query, and returning the matching clip from the series to the user.

Our work contributes to extending the existing Matlab routines in two aspects. First, we tailored the code according to our situation, and made it work with our web framework better. Second, we created a set of web service API’s to expose the underlying Matlab algorithm to the public, so that users can utilize it from the web. We will introduce both aspects in the next section.

¹ http://www.ee.columbia.edu/~dpwe/resources/matlab/fingerprint/
We are not the first one to use audio fingerprinting algorithm. Several research efforts have been built on top of this technology. For example, the authors of [8, 9] applies audio fingerprinting technology to discover repeated audio segmentations in audio streams, and defines these segmentations as “repeated events”. In [10], Ogle and Ellis apply audio fingerprinting on personal audio recordings to identify repeated events throughout a day. In our work, we do not identify repeated events, but in fact identical events that are captured by multiple devices.

In the domain of aligning video clips, what Shrestha et al. have done is most closely related to our work here. They proposed a system which aligns and synchronizes video clips using the detection of camera flashes [11]. Later, the same authors made another contribution by aligning the video clips using audio [12], which is quite similar to the work we have done in this application. However we differ with them in that we shift our focus from designing the aligning algorithm like [7] to applying this technology to more practical and broader use. In particular, we developed other parts of this application (web services and interactive player), and brought in social data to enrich the user experience with dimensions beyond pure video and audio.

One of the most common domains where audio fingerprinting algorithm is applied is audio matching. Previously mentioned Shazam is an example, which takes a short sample of music, and identifies the song.

2.2 Web Services API

An API (Application Programming Interface) is an interface provided by a software program or a web service that allows developers to write code to interact with it. The trend of current web technologies is to make everything “open”, from protocols to
application API’s. Therefore we decided to make our background web services open, allowing developers to use them freely and easily. All they need to do is to sign up an API key with us for the authentication purpose.

Several companies have already started to offer API services for music-related operations. Echo Nest Corporation is a music intelligence company that provides an intelligent platform for music retrieval and attributes analysis (e.g., tempo, instrumentation)\(^2\). In [13] the authors sent songs to Echo Nest, and used their public API’s to get the analysis of the acoustics. Echo Nest is able to characterize the global properties of the songs users provide in a set of 18 different features, like rhythm, pitch, tempo, and instrumentation [14]. The authors of [13] believe that this service should be similar in features to what Marsyas [15] system has to offer. Songs are split into various segments of different length by Echo Nest analyzer, ranging from 80ms to multiple seconds, each section of audio with similar acoustic qualities. Then for each segment, Echo Nest API’s are able to do operations like calculating the loudness. The authors of [14] predict that the possible usage scenarios of Echo Nest API services are versatile. They can be used for visualizations, games, or DJ software. Furthermore, because Echo Nest features not only “reading and listening” the music, but also “learning” about the music\(^3\), it is possible to get the prediction (at least) about which songs will become the next greatest hits.

Although the Echo Nest services are useful and powerful, our web services differ from theirs in that we do not work on every attribute of the audio or any activity or future trend around the audio attributes or genres of the songs they submit. Instead, we give users the freedom to supply audio tracks to the service. We will put effort on user-provided data,

\(^2\) http://developer.echonest.com/docs/v4/
\(^3\) http://the.echonest.com/company/
and give users very detailed feedback regarding the data they submitted. In specific, we are not going to analyze the musical attributes of the individual songs (e.g. tempo, rhythm, or instrumentation), but rather the temporal layout of all the songs in a group. Our web service API’s will return to the users the data about overlapping information of related clips. This data will help users understand how the audio tracks are overlaid, and what the correct way to organize them is. This lays the groundwork and yields crucial data for the multi-perspective video playback interface that we built, which will be explained later.

2.3 Multi-perspective Player

In academia, the topic of multiple perspective video has been studied for a certain amount of time. In [1], Kelly et al. present a multiple perspective interactive video system to provide interactive presentation of events such as dance performances and sport games. Their system takes in video streams from multiple cameras, analyzes them, and then construct a three-dimensional model and visual representation with interactive multiple perspective video access. It tackles the problem from a Computer Vision way, meaning that it relies on the image data to build the three-dimensional world. More specifically, they use Camera Handoff, which can be considered as a scenario in which a moving object is passing from the coverage area of one camera to another. Furthermore, their system is also capable of choosing the video stream that can offer the best view of an object in an environment automatically and interactively. Jain and Wakimoto have done similar work in [16]. They propose a multiple perspective interactive video that enables viewers to view an event from multiple perspectives, or even based on content. In [26], Ballan et al computed 3D spaces and provided spatial navigation between viewpoints in videos from an event. However, both [1] and [16] do not apply time dimension to the
video. They assume that all video streams are in sync, and come from fixed, pre-arranged cameras. This setting is essential to them because they use the stream data to construct a three dimensional environment and select a camera with the best view in real-time. [26] does not take into account human aspects in the presentation of and interaction with the event content. Our work differs from them in that 1) we do not have fixed cameras for all the events as we collect video from multiple people who are randomly placed; 2) we do not rely on computer vision, but use audio fingerprinting analysis result to time-align multiple video streams, and 3) we integrate social data into the video player. Social network and social data only emerge in recent years, and hence is absent in [1, 16].

There are other attempts at summarizing and aggregating videos. In [17], Christel et al. introduce a novel and effective video browsing interface for summarizing news stories. In [2], the authors use both image and audio analysis to detect important and meaningful moments automatically. However, we do not focus on constructing a three-dimensional world, or on automatically presenting users with the best view of an object like [1], but giving users the freedom to choose views. We also let users browse through the group, and let them decide which moment or scene is important to them so that they can switch to that scene and get a larger view. Because the nature of the intended usage scenarios of our player (e.g. concerts, live shows), it is sometimes a bad idea to give system the freedom to judge the best view, since some people prefer enjoying the entire view of the concerts, while others may love to see the performers faces and emotion closely.

In industry, various efforts have addressed the domain of multi-perspective video playback interface from a number of contexts, for instance, watching sport events [3, 18].
Commercial implementations of multi-perspective video such as the HBO Video Cube\textsuperscript{4} have also explored the value of end-user interactivity in crafting the trajectory through a video space. Our multi-perspective video player organizes social video aggregated from online sources like YouTube by automatically time-aligning videos using audio fingerprinting \cite{4,12} and incorporates social metadata to aid in navigation and in creating one’s individual path or “edit” through the video material.

Another area of interest is live event presentation. In these scenarios, video streams from multiple perspectives are common and easy to get, and thus a good source for multi-perspective video play study. A large amount of research efforts have addressed the domain of media from live music events. Naci and Hanjalic in \cite{19} proposes a system with non-sequential access to music concert recordings. The system is able to use audio analysis to extract instrumental solos and applause sections in the concert to find interesting moments, and thus it can give users a higher level of excitement while browsing through the concerts. \cite{20} explores the possibility of using visual signals to index concert video clips automatically, and to create concept detectors such as “drummer” and “stage”. \cite{21} demonstrates a video interaction environment in which semantically coherent temporal segments from a group of concert videos are detected and identified automatically, and a video browsing interface which makes use of the video segmentations produced before.

All the related work mentioned above have one thing in common. They present video content that is usually professionally-produced or considered “authoritative” \cite{4}, for instance the “official” videos provided by the event organizers. Compared with them,

\footnote{http://www.hboimagine.com/}
ours is different in that it seeks not to organize *produced* multi-perspective video, but rather *social, community-contributed* video clips that have been shared by multiple people, each recording their own (potentially fragmented) views of an event. These community-provided video resources are abundant and easy to access due to the emergency of a huge number of video sharing community websites, such as YouTube and Vimeo, and the popularity of hand-held electronic devices with video recording capability. Our multi-perspective video player contributes significantly to the area of event-based management of media. In particular, we are interested in organizing video clips from an event, and presenting them in a synchronized fashion.

Recently we see some community-based methods have been proposed for live event video analysis. In [22, 23], community remix statistics is leveraged to summarize and browse media. In [20], the authors use contextual metadata for a novel and useful media experience. Several efforts have focused on extracting high-quality information from social media. Researchers have also leveraged the time dimension by using synchronized data to identify key moments of interest and extract additional metadata about a particular event [4]. What we have done in this thesis is to embed social metadata in the interactive video player with community-provided content, and present to user with a unified, organized, rich-feature interface.
Chapter 3

Alignment

Video content that is available on video sharing websites like YouTube and Vimeo does not have dependable time-based metadata. Consider a group of video clips taken from U2 concert at the Giants Stadium. None of them has reliable time information regarding the temporal relationship between itself and the entire concert, for instance, the start time of the recording in the entire concert in terms of minutes and seconds. Therefore the clips cannot be synchronized simply by using timestamps.

In this chapter, we explain how we use the *audio fingerprinting* technique to detect videos which overlap in terms of time of captured, and how find the timing synchronization between overlapping video clips.

While the core audio fingerprinting algorithm is not one of our contributions here, we describe the technique in brief for completeness. However, we do extend and tweak the existing audio fingerprinting algorithm in order to make it work better and smoother in our system. We also propose an approach to using the algorithm and applying it to synchronizing video clips.

3.1 Background: Audio Fingerprinting

In [7], Wang presented an industrial strength audio fingerprinting algorithm, which is used by Shazam. In this algorithm, a feature extractor often used to describe short segments of recordings as robust as possible against many kinds of distortions [24]. However these features do not have to relate to human perception or the actual music. They only need to be unique and robust. Then the audio fingerprints, which are sometimes considered as hash values, usually a few bytes per segment, are stored in a
database, along with pointers to the recordings where they occur. In our application the hash values are stored in a hash table in memory. After the hash table is constructed, the same feature extractor is used on the query audio sample. Therefore with the audio fingerprints from the query, matching candidates can be retrieved from the hash table quickly.

Audio fingerprinting finds frequency onsets in audio segments, which are areas of high energy. These onsets are defined as *landmarks*, and they contain both frequency and associated time information that indicates when the frequency occurs in the segment. Audio fingerprinting algorithm then generates hash values for these landmarks based on their frequency, and puts them in a hash table. Typically a hash table with N slots would look like Table 1 below.

<table>
<thead>
<tr>
<th>Hash value #1</th>
<th>Hash value #2</th>
<th>Hash value #3</th>
<th>...</th>
<th>Hash value #N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio ID</td>
<td>Audio ID</td>
<td>Audio ID</td>
<td></td>
<td>Audio ID</td>
</tr>
<tr>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>Audio ID</td>
<td>Audio ID</td>
<td>Time</td>
<td></td>
<td>Audio ID</td>
</tr>
<tr>
<td>Time</td>
<td>Audio ID</td>
<td>Time</td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Audio ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Hash table generated by audio fingerprinting algorithm.

Cells with same background colors are landmarks from audio segments. For instance, there are two landmarks detected with the same hash value #1. One of them is from Clip Red, and the other clip Green. Their associated time information is stored in the cells under ID cells. So far, there is a potential match between Red and Green, but it may be an incorrect match too. Matching clips are detected based on a series matching landmarks.
Indeed, as suggested by Kennedy and Naaman [4], the audio fingerprinting data can be used to generate synchronization information between clips. Figure 2 and 3 above are taken from Kennedy and Naaman’s paper. In Figure 2, X-axis is the timeline of Clip A and Y-axis is the timeline of Clip B. Minus symbols indicate matches, and are plotted using (Clip A time, Clip B time) coordinates.

If the two clips match, they should progress at the same pace, which means that if Clip A progresses one second, so does Clip B. This is the reason why we can see a large part of matches occur along a diagonal line with slope equal to 1. This phenomenon can be also
viewed in Figure 3, where potential time offsets between the two clips are plotted along the X-axis. The heights of the bars represent the number of matches of the same time offset. As can be seen from the figure, there is a peak of the number of matches at the potential correct offset. If we extend the line to make it intersect with X or Y-axis, the interception would be the time offset between the two clips.

Because the hash table maintains the pointers that represent the time the landmarks happen, and because audio fingerprinting algorithm can identify the time offsets between clips, as Figure 2 and 3 suggest, we can get the time offset between the query sample and its matched audio tracks, and thus time-align video clips.

In our application, we take one clip from the group as the query, and get time offset information of the overlap between it and other clips. For instance, we use clip A as the query, and get the offset of five other clips as:

- Clip B: -25s;
- Clip C: -19s;
- Clip D: -6s;
- Clip E: 9s;
- Clip F: 20s.

The data above is enough for our application to time-align all six clips. The video overview is illustrated in Figure 4. More details will be explained in Section 5.
3.2 Existing Code and Limitations

We build on and adapt the work from the “Robust Landmark-Based Audio Fingerprinting” project at Columbia University. The project consists of a Matlab library, based on the algorithm described in [7]. It implements a robust audio fingerprinting system, which is capable of identifying small noisy audio excerpt from a large number of items. Figure 5 below illustrates the functionality of this Matlab library.

The Columbia audio fingerprinting library can read a series of audio files, process them, and output a list of overlapping clips, as well as time offsets. However, there are key limitations with the existing code. Since deciding to adopt this library, we have significantly extended the code to match the requirements of our project. We have added interfaces and custom functions to the library to make it work with our web services and background scripts.
First, and most significantly, the library only returns the best match of the query from the track pool. In another word, the output contains only one entry. It cannot reflect whether other tracks in the pool match the query or not.

Second, as the library has not been fully tested, we found several errors related with the original code that we have fixed and contributed back to the Columbia codebase.

Third, the library is purely written in Matlab. The routines must be called in Matlab environment, or through Matlab command-line interface, a barrier for making the functionality available via an open web service. Consider what we would use as background service, several programs coded in Python, which will be discussed in the next section, we chose a Python library, mlabwrap\(^5\), as the connector that bridges the gap between Python and Matlab. mlabwrap is a high-level wrapper library for Python that

\(^5\) http://mlabwrap.sourceforge.net/
makes Matlab look like a standard Python library. In this way it is easy and convenient to
call Matlab functions in Python, just like calling any other regular functions.

3.3 Contributions and Extension

We need to mention again that the core audio fingerprinting algorithm is not one of our
contributions in this paper. Though we did not repeat the work others have done, we have
made the following contributions and customized modifications to the original Matlab
audio fingerprinting program, in order to make it work as expected in our system.

Our model is different from Columbia’s library. We removed the limitation on the
number of matching clips in the response XML. The original code only returns the first
matching clip, which has the largest number of hits. This kind of handling is safe and
reliable, because the clip with most hits must be the best matching candidate for the query
sample audio. In fact this concept works for other audio identification services (the
typical example-based search [4]), because these services simply return to users is the
best matching outcome for each query. However, this strategy (returning the best match)
does not work in our application. Since we turned our attention from finding a match for
a single audio clip to discovering the temporal alignment of all audio clips in a group, we
need to find all the files that are related and overlap with a query file. Therefore it is
necessary to clear the limitation.

Consider a set of clips $C_1, C_2, \ldots, C_n$. For each pair of clips $C_i$ and $C_j$, we can get the
similarity between them $sim(C_i, C_j)$ based on audio fingerprinting algorithm by counting
the landmarks of both clips in the same hash table slots. The sum is the number of
matching hits/landmarks. Columbia’s library generates a ranked list which consists of
$sim(C_{k_1}, C_j), sim(C_{k_2}, C_2), \ldots, sim(C_{k_n}, C_n)$, and returns only clip $C_{k}$ that yields the largest
sim(C_i, C_k) in the ranked list. Thus C_k is known as the best match for clip C_i. We took a step further by outputting all clips C_j such that sim(C_i, C_j) is greater than a certain threshold. In this way we can find out almost all valid match clips in the pool for a single query.

Typical use case for audio fingerprinting is focused on the detection of matches between two clips, the purpose of which can be either information retrieval or duplicates detection. However in our application, we found that the detection process can also produce extremely detailed estimation of the actual time offset between two clips (0.032 second per frame in our case). Generally, two video clips can be synchronized using the offset value between the two clips at that peak if a peak is detected by the audio fingerprinting matching algorithm [4].

The fact above provides us the fundamental method to organize the video clips inside a group. Specifically in our application, one clip at a time is selected as the “base clip”. Then it is used as the query to find matches against all the other clips in the group. We do that for every clip, and then we have the complete set of overlaps that could be detected.

Our improved algorithm works as following:

1. For each clip C:
   a. For each clip D:
      i. Compute similarity

2. For each clip C:
   a. For each clip D such that sim(C,D) > threshold:
      i. Add D to List matches(C).

Our algorithm outputs a list of matching clips of Clip C.
3.4 Evaluation

3.4.1 System Performance

There are two aspects of system performance that need considering. One is space, and the other is time. Space represents the system resources, for instance, CPU and memory, and time represents how long it takes to find matches within a group, using the audio fingerprinting algorithm.

The main audio fingerprinting Matlab program resides on our server, and is referenced from command line. This means that we do not start any GUI Matlab interface, which saves system resource consumption and speeds up the processing. The Matlab program has two modes, idle and processing. When it does not have tasks to do and is running in background, it is in idle mode. When it is processing a group, it is in processing mode.

Our server has the following specification: Intel Xeon X5460 CPU running at 3.16GHz; 1GB memory; RedHat Linux kernel 2.6.18.

Table 2 shows the system resource consumption of the audio fingerprinting process (MATLAB process) when the process is idle and in processing. When idle, it takes up little CPU time and 9.8% system memory on average. This is acceptable for us. When processing, we notice that the CPU time spikes to 90% on average, and jumping in range of plus/minus 10%, because the audio fingerprinting algorithm is mostly mathematic calculation, which is CPU extensive. 10% of CPU idle state indicates the amount of time taken by I/O operations. The memory usage is 23% on average, and is quite stable.

<table>
<thead>
<tr>
<th>Mode</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>9.8%</td>
</tr>
<tr>
<td>Processing</td>
<td>90%</td>
<td>23%</td>
</tr>
</tbody>
</table>

Table 2: System resource consumption comparison.
Another aspect is processing time. We have run experiments on six different groups of video clips, their sizes (number of clips) 4, 6, 10, 16, 32, and 79 respectively. Table 3 below displays the details of each group:

1) Total Duration. This is the summation of the durations of all clips in the group;

2) Number of Hashes. It is the total number of signatures the program finds from all the audio tracks in this group.

3) Processing Time. We get timestamp twice during each group processing, one at the beginning of the processing (start of the program), and the other at the end (after overlap data has been outputted).

What we have discovered is that the processing time is proportional to the length, which denotes total time duration of all the clips in that group. In fact, the processing time is almost linear to the total time duration of the clips. A linear fitting illustrates this very well, like Figure 6 below suggests. In Figure 6, X-axis is the total duration of the group, and Y-axis is the processing time for the group. Data from six groups are plotted as plus symbols.

<table>
<thead>
<tr>
<th>Group</th>
<th>Number of Clips</th>
<th>Total Duration of Video (seconds)</th>
<th>Number of Hashes</th>
<th>Total Processing Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>701.1436</td>
<td>19007</td>
<td>32.45</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>372.6362</td>
<td>6647</td>
<td>16.78</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>774.7561</td>
<td>22676</td>
<td>33.93</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>2929.8576</td>
<td>91926</td>
<td>159.29</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>6577.4295</td>
<td>239809</td>
<td>320.83</td>
</tr>
<tr>
<td>6</td>
<td>79</td>
<td>18893.9466</td>
<td>567631</td>
<td>984.97</td>
</tr>
</tbody>
</table>

Table 3: Details of groups and their processing time.
Therefore we get the conclusion that, for every twenty seconds of the video clips in the group, our audio fingerprinting algorithm will take about one second to process on our machine. This number will give users an estimation of how long they need to wait before their group processing jobs finish. Processing time increases as the group duration increases, and they follow an almost linear relationship. What’s more important is to us developers. The linear relationship suggests that most time is spent on extracting the signatures (hash values) and very little on comparing and finding matches. We can easily scale up our system by parallelization in the future. When a group contains a large number of clips, we can divide them and assign each part to a slave machine in the cluster. After the work is completed and hash table is created on slave machines, we can merge the hash tables into the final one. By using parallelization processing will speed up significantly.
3.4.2 Alignment Result Evaluation

The actual performance of audio fingerprinting algorithm, for instance, whether clips time offsets detected by the program are accurate and are really matches, cannot be measured by computer directly. Hence we need to compare the computation results to a ground truth generated by a human judge.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Date</th>
<th>Number of Clips</th>
<th>Possible Matches</th>
<th>Actual Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U2 Magnificent</td>
<td>Sep 23, 2009</td>
<td>7</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Mamma Mia</td>
<td>June 19, 2010</td>
<td>8</td>
<td>28</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>Conan O'Brien</td>
<td>June 1, 2010</td>
<td>10</td>
<td>45</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 4: Test Group details, including ground truth (number of actual pairwise matches in the data).

We tested the accuracy of audio fingerprinting with three groups of video clips:

(i) Seven clips about the song Magnificent from U2 concert at The Giants Stadium on September 23, 2009;

(ii) Eight clips taken from a Mamma Mia live show in Leicester Square, London on June 19, 2010;

(iii) Ten clips taken from the Conan O'Brien and Vampire Weekend show in Radio City Music Hall, New York City on June 1, 2010.

We used search terms for specific elements of a certain show on YouTube, such as “u2 magnificent giants stadium”, and picked several video clips from the search results. For each group, we found out the manually for every pair of clips, whether or not they overlap. Then we used the audio fingerprinting algorithm to detect matches. We list the ground truth as in Table 4.

Then we apply audio fingerprinting algorithm on all three groups of videos and observe the output, which is the overlap data of the groups. Similar to the ground truth, most of
the clips in each group roughly covered the same time in the event and thus overlap with each other. This finding proves our assumption that people get a lot of redundant results from YouTube searches on a specific event. The algorithm works quite well on three groups.

The audio algorithm takes in all clips in the group for finding matches, therefore wrong results, clips that actually do not match, are included in the final output as well, but at the bottom of the list due to their low hits. We tested several threshold hit numbers in the algorithm in order to exclude wrong clips while preserving actual matching clips as many as possible. We tested three different threshold values, and results for each clip in each group are listed in tables below. We tested pairwise matches, and collected True/False Positive, Precision, and Recall results for evaluation.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Identified</th>
<th>True Positive</th>
<th>False Positive</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>11</td>
<td>9</td>
<td>100%</td>
<td>55%</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>100%</td>
<td>25%</td>
</tr>
</tbody>
</table>

Table 5: U2 Magnificent group test results.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Identified</th>
<th>True Positive</th>
<th>False Positive</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>22</td>
<td>18</td>
<td>0</td>
<td>81.82%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>16</td>
<td>2</td>
<td>100%</td>
<td>88.89%</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>16</td>
<td>2</td>
<td>100%</td>
<td>88.89%</td>
</tr>
</tbody>
</table>

Table 6: Mamma Mia group test results.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Identified</th>
<th>True Positive</th>
<th>False Positive</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>41</td>
<td>24</td>
<td>0</td>
<td>58.54%</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td>24</td>
<td>0</td>
<td>92.31%</td>
<td>100%</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>24</td>
<td>0</td>
<td>96%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 7: Conan O’Brien Show group test results.
From the results, we can generalize that a threshold below 10 would be good enough to rule out most wrong clips while keeping correct clips in the matching list as many as possible. For clips that overlap, we usually observe a very high number of match hits, which is above 50. However the number of hits is related to the length of overlapping audio. So we must also consider matching pairs that only share a short period of audio. That is why the threshold cannot be too high; otherwise most of these pairs will be excluded. The threshold-precision and threshold-recall trends can also be illustrated in Figure 7 and Figure 8 below.

![Figure 7: Threshold-Precision trend.](image1)

![Figure 8: Threshold-Recall trend.](image2)
Chapter 4

Web Services

4.1 Design Goals and Framework

The alignment algorithm described in Section 3 resides on our server. To use the audio fingerprinting algorithm to process clip overlap, we use a Matlab program or command-line prompt to call the function. However, there are some prerequisites:

1) Matlab program must be installed;

2) Additional utility programs must be installed, including mpg123, mp3info, and lame.

Normally, most computers do not have these programs at the same time, so it is not easy to run our audio fingerprinting matching algorithm locally (on any computer). Therefore, in order to make the algorithm accessible by all users and developers, we developed a set of web service API’s.

With the API’s, developers are able to not only use the audio fingerprinting algorithm in our system, but also manipulate groups of video clips, and retrieve the results of audio fingerprinting matching process. Developers do not have to install the programs mentioned above. Instead they can invoke the processes and get results programmatically, and remotely.

4.2 API

There are six API’s provided by our web services. Developers can use them to create groups, delete groups, add clips to groups, get clip lists of groups, process groups, and get overlap data of a clip.
All of these API’s accept URL queries, and they all require developers to have valid API keys. Developers pass appropriate query parameters to the URL request, and our web services will return the response in XML format.

When a user visits our project API site for the first time, they will be asked to sign up an API key. All they need to give us is his email address. We generate a 32-bit unique string based on the email, as well as a brief instruction of how to use the API’s.

All of our API services are in the form of URL query strings. This means that all parameters required by individual API methods should be provided in the URL, and should be organized in key-value pairs. Our API will respond to a call in XML format, no matter the result is a success or an error.

In the following context, we will use the API-key, group-name, group-id, uid to signal the place where users should replace with their actual API key provided by our web services, their group name, the group ID in the XML response from API, and the 32-bit unique ID for the clip from XML response respectively.

The API root is http://sm.rutgers.edu/relive/af_api/. The following discussion of the API methods will use root to substitute the actual URL for simplicity.

4.2.1 Create A Group

root/group_create?api_key=your-key&name=group-name

This is usually the first step, creating a group with group_create API. It will give the user a number ID for the group. All of our API methods identify groups by their ID.

group_create method will first check the validity of the API key the user provided upon receive of the API call. If it is invalid (i.e. the key has not been registered with our web
service), the method will return with an error code and human-readable error message like the one below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>
    <code>9</code>
    <message>Invalid API key.</message>
  </status>
</response>
```

Figure 9: Invalid API key response example.

Validating API key is the first step of every API call, and the XML response like Figure 9 will be returned in every API call if an invalid key is provided. This process will not be repeated in the following text in order to reduce verbosity.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>
    <code>0</code>
    <message>Group created successfully.</message>
  </status>
  <group id="54"/>
</response>
```

Figure 10: group_create API call response example.

Figure 10 above shows an example of the group_create API call.

4.2.2 Add A Clip

`root/clip_add?api_key=your-key&ytid=youtube-id&groupId=group-id`

This API method will add the YouTube video specified by its youtube-id to the group with ID group-id. The youtube-id is the 11-digit unique identification string used by YouTube.com.

The method first checks if the target group has already been created by the current user. Second it checks if the same request has been submitted sometime earlier but has not
been fulfilled (i.e. the request is in the task queue). Third it checks if the target group already has this YouTube video. Fourth it checks if another user has the same video clip. If that is true, the video and audio files will be simply copied from his group folder to the current user’s group folder. In this way we can greatly reduce processing time and eliminate unnecessary downloading and audio extraction. Since this API request will be pushed into task queue, faster process can save more time for other tasks that follow and improve the efficiency of our web services.

If all tests above pass, this clip downloading request will get executed, and user will be given a response in XML format. Details of how API calls are executed will be explained in sub-section 4.4.

```xml
<xml version="1.0" encoding="UTF-8">
<response>
  <status>
    <code>0</code>
    <message>Clip is being added to group.</message>
  </status>
  <clip id="38b5714eebccc545f5e1d6089a41b204"/>
  <group id="54"/>
</response>
</xml>

Figure 11: Successful clip_add API call response example.

Figure 11 above shows an example of a successful clip_add API call. However, if there is any failed check at the beginning, an error will be returned in XML format immediately. For instance, if a user submits the same task for the second time before the first one is finished, a response like below will be returned.
We make error messages as instructional as possible, and give each type of errors an error code. Therefore developers who utilize our web services can program the error handlers in their applications accordingly.

4.2.3 Process A Group

root/group_process?api_key=your-key&groupId=group-id

group_process API method processes the target group and generates overlap data files for all video clips inside that group. The entire process takes up a large amount of time and system resources, so we push this type of jobs into the task queue, and let workers work on them.

Upon receiving the API call, the method checks the existence of the target group. Then it checks whether the target group is empty, already processed, or being processed. The method also checks whether all the video clips in this group have been downloaded and extracted or not. If all the checks pass, the process request will be executed, and inform user about the status. If not, an XML response with error code and message will be returned immediately, like Figure 13 and 14 below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>
    <code>3</code>
    <message>
      This task is already in the task queue. Please wait patiently.
    </message>
  </status>
</response>
```

Figure 12: Failed clip_add API call response example.
There is such possibility that users may add additional videos to groups that have already been processed. In this case, old hash table and overlap data cannot represent the current group. Therefore, every time when users add a new video clip to a group, it will invalidate the state bit of the group, so that when users request for overlap data, they will get an error saying the target group has not been processed. In this way, all clips are guaranteed to be processed and compared, and none is left out.
4.2.4 Get Overlap Data for A Clip:

```
root/clip_get_overlap?api_key=your-key&uid=uid&groupId=group-id
```

This API method responds by presenting users overlap data for a specified clip. This XML response includes the unique IDs of the matching clips sorted by number of hits in descending order. It also includes the precise time offset of the match.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>
    <code>0</code>
    <message>Clip overlap information retrieved.</message>
  </status>
  <analysis>
    <overlap id="adb8b7c5f71edddc1fdec2a383b6e5c8">
      <clip id="adb8b7c5f71edddc1fdec2a383b6e5c8" hits="10160"
        offset="0" ytid="nsfWX1mCLY"/>
      <clip id="fbc737cddb0a573dc993efa62d26d79b" hits="44"
        offset="-2657" ytid="VcO9efXg8-8"/>
      <clip id="lcc459bcwcc877f3460f644e8527a" hits="33"
        offset="456" ytid="ZoPd8XneC7w"/>
      <clip id="32195d4e8872ed3ca104b17f77f9e82b" hits="11"
        offset="-3917" ytid="QDDqGFDwpK4"/>
      <clip id="a614113abc6fda97279e5cc9a3f1c91a" hits="5"
        offset="-456" ytid="9zwUGDHRrFM"/>
      <clip id="0a57324af61001a74e38dc2ed2d276f7" hits="4"
        offset="-1555" ytid="vp6TNHYjgo"/>
      <clip id="744acc8ddab003c6e1d54c90cf0f94" hits="3"
        offset="-2569" ytid="ystHLRVw_0"/>
    </overlap>
  </analysis>
</response>
```

Figure 16: clip_get_overlap API call response example.

Figure 16 is a sample XML response of clip_get_overlap API call. Offsets are in frames. The ratio between frame and second is exact 0.032 second per frame, as defined in Matlab audio fingerprinting program, so a 6683 offset means the matching clip (ID d8b4d0fd916914ccc16f008e7c70c0d4 in this case) starts 6683 $\times$ 0.032 = 213.856 seconds ahead of the target clip (ID c5abbfb53f10430353d5af714b17bfb7).
4.2.5 Get A List of All Clips in A Group

root/group_get_clips?api_key=your-key&groupId=group-id

Besides the four steps mentioned above, there are another two API calls that help users work with their groups. One of them described in this sub-section enables users to view all the video clips that have been added to the target group. The other one will be introduced in the next sub-section.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>
    <code>0</code>
    <message>Clips in the group are retrieved.</message>
  </status>
  <clips>
    <clip id="3ae6ec04852600af24531776842cfe93"/>
    <clip id="921005fe8ad4cca2b9e689d0c00e80e80"/>
    <clip id="17fd7d80a419156299ac080bc514471a"/>
    <clip id="792d9e707e513333d102a4b8b640146"/>
    <clip id="a31a359ef3d31ce7831d7f7930b5a9694"/>
    <clip id="7f4d797d9bf853a1d27b9ef50e51a4a3"/>
    <clip id="935a49ef3141971ba153417ff3818dcc"/>
    <clip id="3418ea862c38ef006f95df836850ee03"/>
  </clips>
</response>
```

Figure 17: group_get_clips API call response example.

Figure 17 above displays a sample response XML of group_get_clips API call. Each clip is enclosed in a <clip> tag, and its unique ID is attached as attribute of the tag.

4.2.6 Delete A Group

root/group_delete?api_key=your-key&groupId=group-id

Calling group_delete API method will delete all the data related to the target group. That includes the entry of the group in the database, entries of all video clips of the target group in the database, and all physical video and audio files on our server. If any overlap data file has been generated and stored, it will be removed as well.
Figure 18 displays an example response after a successful call to the group_delete API.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <status>
    <code>0</code>
    <message>Group deleted successfully.</message>
  </status>
</response>
```

**Figure 18: group_delete API call response example.**

### 4.3 Use Case Scenario

We can think of many practical scenarios for developers using this API. A typical one can be as following. A developer is interested in getting overlap and alignment data for several video clips. Thus he can use our web API’s to create a group and get overlap data:

1) The developer goes to the homepage of our project, and signs up with his email address to get an API key. This key is required for all API calls.

2) The developer calls the group_create API function. In this step he should give the group a name, for instance, based on a group of videos they collected like “U2 Magnificent”. After the group is created successfully, a group ID will be given. The ID is used in other API calls.

3) The developer makes a series of calls to the clip_add API to add the appropriate clips to the group. This API requires API key, group ID, and YouTube ID of the clip that is being added. Each API call adds one clip to the group. It looks like that the API call is quite limited, since you can only add clips one at a time. However, developers can write small but efficient scripts to automate the process. If a clip is added successfully, system will give a clip ID (or UID) of the clip. Each clip in our system has a unique UID. It is used in group_process, group_get_clips, and clip_get_overlap calls.
4) The developer initiates the group processing using group_process API call. The developer needs to provide target group ID. This step and step 3) both take a considerate amount of time. So if the developer wants to continue to next step, or repeat the same step, they will get error notifications.

5) The developer retrieves the overlap data to be used in their application. Returned results contain UID, and offset of each UID. This data can be used to construct a group overview. In this thesis, we present a working video player that takes advantage of the overlap data in Section 5. However, developers are free to create their own applications that consume the overlap data.

4.4 Implementation Details

Because the workloads of tasks are often very large (e.g., downloading video clips from YouTube.com, extracting audio tracks, running Matlab program to analyze audio), we implemented a task queue based on the database. More on task queue can be found in the following subsection.

Our web service is a hybrid collection of programs. It features a Model-View-Controller (MVC) architecture written in PHP, and developed on top of CodeIgniter PHP web framework. This part receives all requests, handles database operations and a task queue, and outputs XML responses. As mentioned before our system must download video clips, extract audio tracks and process audio fingerprints, so we have another one master and two slave workers written in Python. They run continuously in the background to monitor the task queue. If the master worker finds any new task in the queue, it will dispatch it to one of the two slave workers, depending on the type of the task.
4.4.1 YouTube Integration

The web services and API’s are closely connected to and dependent on online video sharing website YouTube.com. After users have created groups on our web services, they can add video clips that are on YouTube to the groups. What will be done on our side next is to download the video clips from YouTube, extract audio tracks from them, and then add the clips to the group. In order to work on the connection with YouTube.com, we incorporated a Python written utility, youtube-dl\(^6\), in our task workers. The youtube-dl is originally an open source command-line Python program that downloads video clips from YouTube, as long as the YouTube URLs are given. We adapted and merged its code to our web service so that our task workers can work with it seamlessly.

Our audio fingerprinting service focuses on analyzing audio files, for example, .mp3 files. Therefore it cannot recognize any video files that are fetched from YouTube.com directly because they are in either .flv or .mp4 format. In our project, we use open source library FFmpeg to convert those video files to audio files. The workflow of fetching and extracting video content is illustrated in Figure 19.

---

\(^6\) http://bitbucket.org/rg3/youtube-dl/wiki/Home

---

Figure 19: Workflow of fetching and extracting video content
4.4.2 Task Queue and Workers

The requests are often time consuming and have heavy workload because of the mathematic calculations happened in Matlab, I/O, and database operations. For instance, clip_add API call, which is responsible for adding video clips to a group, includes retrieving video clips from YouTube, extracting audio tracks, and inserting entries to the database. These steps combined will take more than one minute for each clip on average, and will be much longer if the video duration is long and internet connection speed is slow. Therefore it is unrealistic to let the web service work while keep users waiting for it to finish, because either the user will feel bored and terminate the job request or they might get connection timeout.

Our solution is bringing background workers that focus on heavy lifting work, and a task queue in the database that stores current active request and future requests in the queue. Workers monitor the task queue, fetch a new task to work on, and update the status of the task after it is completed.

After clip_add API request starts, a new “download” type job will be inserted into the task queue along with the clip information, and inform user that the video clip is currently being processed. At the same time, workers detect the new job and start fetching video from YouTube. They then use FFmpeg to extract audio from the video and place it in the same directory.

The group_process API does even more heavy work because it involves processing in Matlab environment. After a task of this kind is retrieved from the task queue, our web services will start Matlab engine, which will read all extracted audio tracks in the group and start to construct a large hash table for the audio fingerprints of all clips. Then it will
compare each audio track against the audio fingerprints hash table to find out any matches as well as the number of hits related to that match. Data regarding clip matches are stored as text files on our server, so that they can be used later by the clip_get_overlap API to generate XML responses.

4.4.3 Matlab and Python Integration

Our workers are several Python scripts, while audio fingerprinting algorithm is coded in Matlab. The algorithm must run in Matlab environment. Therefore the workers must have the ability to invoke Matlab routines.

In this thesis, we adapted a connector between Python and Matlab, called mlabwrap. Normally mlabwrap is mostly implemented to call Matlab intrinsic functions, for instance plot(), and svd(). In our system, we do not necessarily call those functions. Instead, we have several custom Matlab functions, defined in .m files. Each one of them handles one of the several API calls (e.g. group_process, clip_get_overlap). Thus we implemented mlabwrap to call these custom functions directly from workers. After the functions are executed and returned, workers will update task status in the queue based on the outcome of Matlab functions.

4.5 API Workflow

Three parts of the web services, (i.e. the PHP web framework, the Python background workers, and Matlab audio fingerprinting routines) work together to handle user requests and finish new tasks. Their relationship is explained with Figure 20 below.

4.6 Limitation and Extension

There are some limitations that we are aware of and some extensions that we expect to the API.
First, the API’s for adding clips and processing groups lack progress indication. If a user submits the same job for the second time, he will be given a general message which says that the job has been submitted. Ideally we expect it to give more detailed information regarding the percentage of the task it has finished.

Second, the APIs are RESTful [25], but do not support all “RESTful” verbs, meaning that other HTTP methods (e.g., PUT, POST, and DELETE) are not exploited. We rely on GET methods with URL parameters to process requests.
Figure 20: Relationship among different parts of our web service.
Chapter 5

Multi-perspective Video Playback

5.1 Motivation and Sample Scenarios

With the widespread of consumer electronics products, especially digital cameras and camcorders, we now find ourselves in a world where videos recorded by anyone in the world provide an abundance of raw material for consumption and presentation. This phenomenon has the potential of enabling a new, deeper and more comprehensive way to represent these videos. On the other hand, video sharing websites in social environments like YouTube, Vimeo, and Metacafe have broader reach and higher impact on users. More and more users trust and rely on these sites to enjoy all kinds of events, from concert to speech, from sport games to entertainment awards.

Let us go back to the scenario in Section 4.5: Irish rock band U2 held a concert at Giants Stadium yesterday. When you turn on your computer and go to YouTube.com, you will most likely to find thousands of fans have uploaded videos they recorded at the event. Suppose your favorite song is *Magnificent*, you search for “U2 magnificent giants stadium” on YouTube and get hundreds of entries.

The problem now is that you probably do not want to watch all these clips while hearing the song over and over again. However, those clips were recorded from various positions of the stadium, so you may really want to check them out and see how it feels to watch Bono (the lead singer) or Adam Clayton (the bass player) on stage from different angles.

The above scenario is very likely to happen more and more often than ever before, in a world that is tightly connected to the Web and online community. There are some attributes that exist among these video clips you may find.
First, there is relevancy. Some of them may be recorded during the same song, and some across several songs. Some are longer, and some are shorter. However, there must be some temporal overlap among some of the clips, because at any given time, all the cameras that were turned on must be recording the same one actual event.

Second, there is redundancy of video. Because there is only one actual live event, and multiple video recordings, redundancy is obvious and inevitable. The clips often do not extend across the entirety of the event in time, and they usually cover varying time period. This redundancy exists in the audio, because what we hear, either *Magnificent* or *Beautiful Day* from *U2*, should be the same across all video clips, though the noise in them is likely to vary, due to different amount of applause, random talking, etc.

Third, there is uniqueness of video (image). Although all the audio tracks from the video clips are redundant, the visual content is unique. Suppose Clip A was taken from the front row, near the center of Giants Stadium, while Clip B was recorded from a high position that is farther away from the central stage. Both of them can give users very different experience. On one hand, viewers can see clearly Bono’s face and emotion from Clip A, while on the other hand, they are able to perceive what was happening in the entire stadium, such as the marvelous stage, light effects, and crowded audience, from Clip B.

Fourth, there is social data. After these video recordings were uploaded and published online, they became parts of the online community. Therefore a large amount of social data that is related to the clips will be generated and retrievable to all users. For instance, YouTube currently provides access to such metadata as average rating, view count, number of comments, and comment text for each video. This kind of data is valuable and
useful because it reflects general users’ common interest, video quality, and event’s attractiveness.

Fifth, there is video quality and capture control. The quality of the video depends on the user, their capture device, their position, and most importantly, how they record. Usually there is shaking and noise in those community-provided video clips. However, this attribute, which looks like a disadvantage of community-provided video data, enables us to think from another perspective. Would it be great if we can switch among several clips of the same event, and find the best angle at a given time?

Our multi-perspective player comes in to tackle the above redundancy-versus-uniqueness dilemma you may face. It enables users to view an event from multiple perspectives, while eliminates the boringness of listening to the audio repeatedly. Users are able to switch among several videos, jump to any time point, and control the volume of the video that is being played. Our player also supports and brings in social data to aid in navigation and selection of different views on an event. The view count information from YouTube is mapped and represented in the multi-perspective player interface.

Our system works by aggregating video clips from individual events and then uses audio fingerprinting to time-align the videos, as described in Section 3.1. This pre-processing step occurs on the server before the multi-player renders the interface. Then we utilize the synchronization data to construct an application scenario in which aggregate content from a live show is being organized and viewed. The drawbacks of this approach are that videos must come from the same aural space and poor audio quality can sometimes lead to difficulty in matching [4].
5.2 Design and Considerations

Before heading into the actual implementation of our multi-perspective video playback interface, we show below what the interface looks like. Figure 21 below shows our multi-player. It consists of four main modules roughly.

![Mamma Mia](image)

**Figure 21: A draft of the video playback interface.**

(a) The main part of the figure in the upper left. The video being displayed here is chosen from the group of video clips by the user. It is scaled up to fit the large area. If the user wishes to select another clip as the main video, the previous one will be scaled down and placed in the “switch-to” list, as explained below.
(b) A list of available “switch-to” clips simultaneously playing in the panel on the right. These are the videos that are playing at the current time point in this group, except the one that is placed on the left as the main video. As time progresses, more video clips will become active and be pushed into the list, while some will finish playing and be removed from the list. A placeholder is inserted to indicate where the main video fits in the “switch-to” stack.

(c) The multi-perspective overview panel showing the timeline layout of the different available clips and their temporal alignment to each other. Each rectangle represents a time period that a video clip covers as in the entire group, therefore the overlap areas of the rectangles suggest clips that should play simultaneously. The vertical order from top to bottom corresponds to the vertical order in the “switch-to” video list on the top right, and a hover over a temporal representation of a clip highlights its video frame as well, so that users can easily correlate the video with its timeline. To aid navigation, each timeline can have three different border layout styles: solid black, dashed gray, and none. A solid black outline indicates the main video and a dashed gray outline indicates the clips that are currently available to switch to. If the video is not available to be played, the border on its timeline is removed. Each timeline can have different filling color as well. The varying intensity of the blue color is calculated based on its view counts retrieved from YouTube, the more the deeper. Clicking a clip’s timeline in this area or in the switch-to list makes that clip the current main video. To better help users navigate, we use smooth and informative animation during the transition.

(d) The play/pause button, volume adjust, play time display area, and the progress slider. Play time is displayed in two rows. The numbers on the top indicate elapsed time versus
The length of the group’s total time is calculated from the start of the video clip that starts first, to the end of the one that ends last. The numbers on the bottom indicate elapsed time versus total time of the current main video. Traditional methods of video navigation such as start, stop, and non-linear seeking are all supported by the progress slider.

5.3 Implementation Details

5.3.1 Adobe Flex vs. HTML5

This multi-perspective video playback software will be web-based, therefore we compared two possible solutions before jumping into actual implementation. One of them is Adobe Flex, and the other is HTML5.

Adobe Flex is released by Adobe Systems for the development and deployment of cross-platform Rich Internet Applications (RIAs) based on the Adobe Flash platform. Because it can be considered as a child of Flash, it inherits many powerful capabilities of Flash, including support for animation and video playback. The language behind it is ActionScript 3.0, an object-oriented scripting programming language. Compared with ActionScript 2.0, it provides more and better control and code reusability. This is especially convenient if the Flex application that is being built is complex and large in scale. Final output of a Flex application is a .swf Flash file, which is supported and can be viewed by all web browsers as long as they have proper Flash plug-in installed. However this requirement poses a potential problem for computers without Flash plug-in and mobile devices that do not support Adobe Flash technology.

HTML5, which is currently still under development, is the next major revision of the HTML standard. Compared to the current HTML markup language, HTML5 comes up
with several new elements and attributes that are commonly found and used on modern websites. One of them is the `<video>` tag. With `<video>` tag, video clips can be embedded into webpages directly, without the need of additional plug-ins such as Adobe Flash, Apple QuickTime, or Microsoft Silverlight. This sounds promising, but we gave it up for three reasons.

First and most important, HTML5 is not mature yet. It is currently under development, and is expected to be recommended by W3C in the year 2022 or later, according Ian Hickson, editor of the HTML5 specification. (FAQ) Compared to Adobe Flex 3 which was announced in April 2007, we are not convinced to building the application in HTML5.

Second, not all web browsers support HTML5 yet. Most major browsers have to wait until next version for support of HTML5 video. Besides, browser support varies among three or more video formats.

Third, HTML5 does not have as powerful animation programming capabilities as Adobe Flex. From a human computer interaction approach, we wanted to add a reasonable amount of animation effects into the application, so that users know what is happening now, and when they interact with the playback interface (e.g., clicking, dragging), it responds accordingly. Given the animation nature of Adobe Flash, we choose it for building our interactive player.

5.3.2 Implementation

Flex / ActionScript provides various and a great number of event listeners for developers to use. These events work like signals. Once an event is triggered, ActionScript will look for the corresponding event handler if defined, and execute any code in that function. We
take advantage of this mechanism greatly in our player application to accomplish many complicated functionalities, for instance, loading social data like ratings from YouTube and switching video.

The structure and working diagram of our multi-perspective video player can be illustrated with Figure 22 below.

![Player structure and working diagram]

**Figure 22: Player structure and working diagram.**

### 5.4 Use Case

The multi-perspective video player can be used in various scenarios. For instance, fans of a band can create unique experience of their previous concerts for fun, or the band’s agent company can create a promo site by using our interface.
When a user opens up the player with video content from multiple perspectives, he will see the temporal structure of all the clips. Then he can start playing, each video will start when its start time is reached. While playing, he can switch between perspectives, either near or far away from the stage. Both details of the performance and entire stage even audience can be viewed at the same time.

5.5 Evaluation

We arranged a human subject study on how people navigate, perceive, and use a user interface designed to organize time-aligned multi-perspective social video. In particular we are interested in assessing the interface’s ability to support navigation between different viewpoints and to enable meaningful navigation using social metadata. To aid analysis, we created a logging system that logs every participant’s response (click, drag, hover, etc.) to database.

We randomly recruited ten people to participate in our user study. Their ages range from 19 to 32 years old, and have both male and female. Our procedure included three phases, one exploratory in nature, and two phases that included a task-based scenario. The tasks were geared to mimic common use scenarios of event-related online videos. At each phase, the participant interacted with content from a different event. In the first phase, the participant was given a brief overview of the interface and was asked to explore the system, viewing content from one event. After the exploration phase, the researcher explained any features that the participant had difficulty with or might have missed before moving on to the next phase. The second phase was task-based, in which the participant was asked to view the event content as if they were trying to decide whether they wanted to attend a similar event in person in the future. For the third phase, again
task based, the participant was asked to explore the event content with the intention of sharing it with a friend. We asked them to rate their interest level in the presentation of the three events that are mentioned in Section 3.4, from 1 to 5, 5 being the highest. We also asked how often they watch video clips on video sharing sites like YouTube. In Figure 23 below, X axis is the frequency of people watching video online, and Y axis is the average interest people have in our video player. Each dot represents one person’s feedback. Generally the result suggests that people who often watch videos online like our presentation of live events better.

Analysis of how users interact with the player interface shows that users interact with the “switch-to” videos significantly more than with the timelines at the bottom. They hover over and click small video previews on the right much more than the timelines. This is because “switch-to” videos on the right are more straightforward than the rectangle bars at the bottom.
Finally, we recorded participants’ response to some subjective questions regarding the appearance and functionality of the player. Overall people find the interface logical, innovative, and easy to use. Particularly they like the way to organize videos from amateurs. It could be used to see a venue before purchasing tickets and to decide whether to attend or not. They find it useful to know how long the clips would last by looking at the timelines. They also like how they can switch perspectives without interrupting the video playback.

However they also complained about the colors of the timeline bars at the bottom most. It is difficult and unobvious to connect the color with view count of that video. The possible improvement would be replacing the view counts with video ratings from YouTube. Some participants consider ratings are more important and more helpful in assisting users to navigate.

Users also expressed their expectation to take control. It would be very helpful to enable users to flag or even delete a misaligned clip from a group. If the community (users) contribute, then the multi-perspective experience could be better.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have presented a system and interface for aggregating, time-aligning, and navigating multi-perspective social video aggregated from YouTube. We are currently exploring different visual mappings for social metadata to enhance the player and are planning a laboratory study to assess users’ experience and satisfaction navigating using the interface.

6.2 Future Work

We are interested in studying the utility of the multiplayer using a variety of event content from news and speeches to concerts and other performances.

The Matlab audio fingerprinting algorithm can be extended so that

- it is able to detect inter-group overlap,
- groups of videos can be linked towards a larger group,
- the threshold for qualifying valid matches can be dynamic and adaptive to different groups,
- save clip hashes and hash table information so that when a new clip is added to the group, only the new clip has to be processed rather than the entire group.

The API services can be upgraded to fully support RESTful HTTP methods and provide progress indication, as Section 4.6 points out. By implementing a RESTful web service, users can leverage standard HTTP methods (i.e. PUT, GET, POST, DELETE) on web resources (clips or groups). Users can even make requests by sending an XML-based feed, which is much more powerful. For instance, a user can create an XML feed that contains
event name, place, time, description and other metadata, and then use PUT method to our web service. A resource for this particular event will be created. He can later on use GET method to retrieve the group details, use another PUT to update the resource, or use DELETE to remove the group entry. On the other hand, a progress indicator is a convenient feature to notify users of how their jobs are being processed. It is particularly useful when a group is significantly large.

The video playback interface can also be improved, according to the feedback from our user study. First, it could integrate more social data, for instance, location, date, venue, etc. Back trace links could be provided to users so that they can go to the original location of the video to view more information. Second, video clips could be more interactive, which is a highly requested feature from user study session. By enabling users to flag or delete clips from groups, or to save group configuration for future use, users could create a custom timeline and go through the various clips and establish one edited video of the event for viewing. Video clips could be even tagged or bookmarked. This kind of community metadata could be important and useful. Third, video content could be analyzed to get their appropriate positions in the venue. Thus we could transform the main video area into a grid of videos, or a video wall, where clips are showing different angles at their correct position in the grid.


Reference


